HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
School of Information and Comunication Technology
*****

GRADUATION THESIS

Topic:

Thesis title

Author: **Trung Tran**

Supervisor: **PhD. Binh Minh Nguyen**

Hanoi, 12/2019

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problems

1.2 Summary

Chapter 2

Materials and background

2.1 Meta-heuristic Optimization

2.1.1 Idea and motivation

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 (see Fig ***): 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. 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) [23], which mathematically mimicks the Darwinian's evolutionary laws. Other later algorithms are Generic Programming (GP) [31],

Differential Evolution (DE) [14], Biogeography-based optimization (BBO) [43] and Coral Reefs Optimization Algorithm (CRO) [42].

The second category is physics-based methods, which imitate physical principles in the universe including Big-Bang Big-Crunch(BBBC) [13], Gravitational Search Algorithm (GSA) [38], Charged System Search (CSS) [26], Central Force Optimization (CFO) [15], Artificial Chemical Reaction Optimization Algorithm (ACROA) [1], Black Hole (BH) algorithm [21], Ray Optimization (RO) algorithm [25], Small-World Optimization Algorithm (SWOA) [11].

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, so swarm-based optimization is 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 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) [12], 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) [10], 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 ***. 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.

It is worth mentioning here that there is a group of methods derived from human's activities in daily life. They are human-based optimization algorithms. Some of the well-known algorithms are Harmony Search (HS) [17], Teaching Learning Based Optimization (TLBO) [**?**], League Championship Algorithm (LCA) [24], Tabu Search (TS) [9] and Colliding Bodies Optimization (CBO) [28].

2.1.2 Particle Swarm Optimization (PSO)

PSO [29] 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.

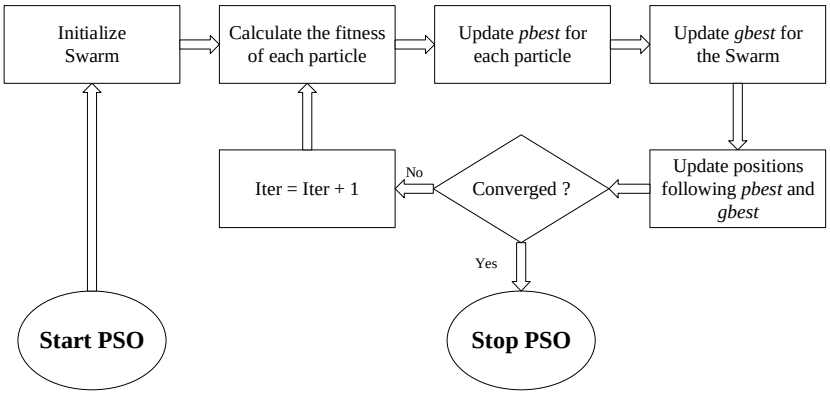


Figure 2.1: PSO flowchart

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;

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: $X_*, f(X_*)$

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)

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

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

- Tracking and chasing the prey using their senses.

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

Those behaviors is the inspiration for the Sea Lion Optimization (SLnO) which was first introduced in [35]. 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**

   Sea lions can identify the location of the prey and gather other members that will join the subgroup to organize the net following the encircling mechanism. This sea lion plays an important role as a leader for this hunting behavior and other members' position will be updated following the position of the prey. In SLnO algorithm, the prey is considered as the current best solution or the solution closest to the optimal solution. This behaviors is presented mathematically using Eq. (2.3) 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, helping the search agent find optimal or near optimal position. $SL(t+1)$ is the new position of search agent after updating and C is linearly decreased from 2 to 0 over the course of 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.

2.2 Artificial Neural Network (ANN)

2.2.1 Activation functions

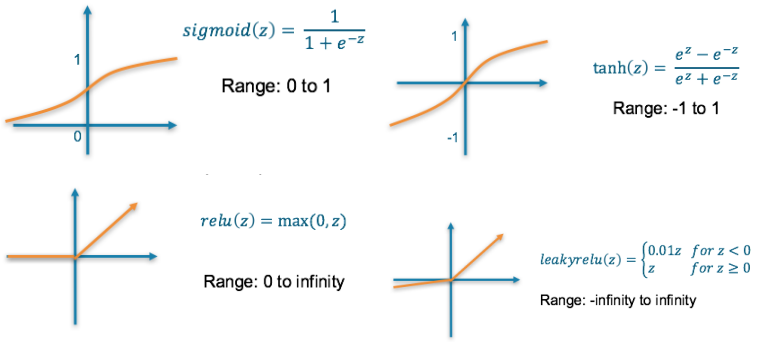


Figure 2.2: Activation functions. Source: https://medium.com/@srnghn/deep-learning-overview-of-neurons-and-activation-functions-1d98286cf1e4

Activation functions also known as transfer functions are used to map input nodes to output nodes in certain fashion. Activation functions are extremely important for an ANN to learn and figure out features and characteristics of data which need a non-linear transformation to become outputs. There are the four most popular functions used in Deep Learning, their names are Sigmoid, Hyperbolic Tangent (Tanh), Rectified Linear Units (ReLU), Leaky ReLU and Exponential Linear Unit (ELU) (See in Fig. 2.2).

- **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.

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*;

2.2.3 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 [41]. 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).

2.2.3.1 Forward propagation phase

1. The input values will be fed into ANN through input layer, going forward to hidden layers, and finally to output layer, creating predicted output values. While propagation process, each layer uses its own activation function (sec. 2.2.1)
2. Error values are calculated by the loss function and propagated back to previous layers.

2.2.3.2 Weights updating phase

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

These two phases are repeated in each iteration during training. The algorithm will be stopped when the error from loss function reach a acceptable value or when the training iteration is large enough. The algorithm's pseudo code is presented in short in Algorithm 2.

2.3 Time-series prediction and Auto-scaling problem in Cloud Computing

2.3.1 Cloud Computing

2.3.2 Auto-scaling problem

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

Recent developments in cloud computing including resource management have resulted in a significant interest in resource usage prediction . Various methods have been proposed for solving this problem with different aspects, objectives and applications [2]. In this section, we focus on several Artificial Neural Network (ANN) models that are used for tackling the time-series characteristic in resource usage forecast in cloud computing environment. Deep Feed-forward Neural Network, also called Feed-forward Neural Network (FFNN) are the quintessential deep learning models. The goal of all FFNN is to approximate some functions f^*. In Regression problems, $y = f^*(x)$ maps an input x to a value y. A feed-forward network defines a mapping $y = f(x, \theta)$ and learns the value of the parameters θ that result in the best function approximation. [19]. These models are called feed-forward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y. There are no feedback connections in which outputs of the model are fed back into itself. When feed-forward neural networks are extended to include feedback connections, they are called recurrent neural networks, which will be discussed in 2.3.3.3.

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.

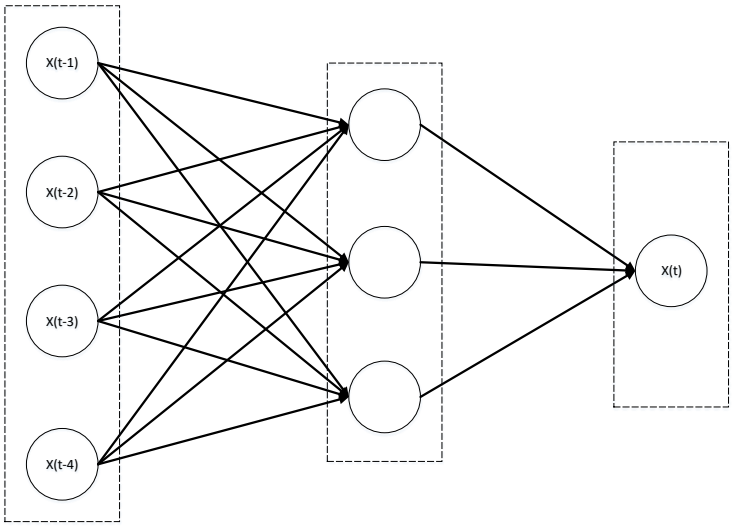


Figure 2.3: An example of MLPs model in time-series prediction.

2.3.3.1 Multi-Layer Perceptrons (MLPs)

The additional layers added between input layer and output layer make network architecture contain hidden layers and called Multi-Layer Perceptrons (MLPs). The input data is fed through input layer to hidden layers in the weighted form. The information from input data X is distributed to the neurons in hidden layers and then processed by an activation function. The activation function in hidden layers are non-linear function playing a role as a transfer function, helping MLPs learn non-linear characteristics of the data. The information after being processed by hidden layers then are sent to output layer in the weighted sum, and also go through an activation function as well, creating the output value y. MLPs model is used in predicting time series data [3], [30]. 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.

2.3.3.2 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 [47]. 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.

2.3.3.3 Recurrent Neural Network (RNN)

Recurrent neural networks (RNNs) are dynamical systems that are specifically designed for temporal problems, as they have both feed-back and feed-forward connections (Fig. 2.3.3.3). 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 [49], [7], [5]. There are two popular

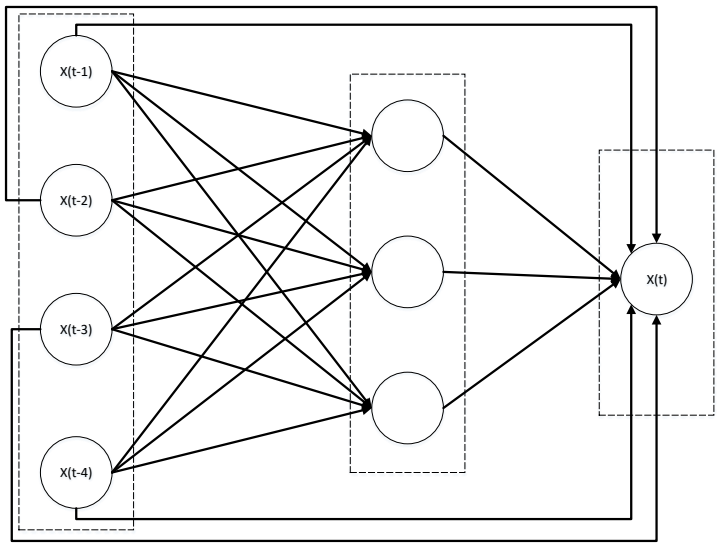


Figure 2.4: An example of CFNN model in time-series prediction.

and efficient RNN models that work really well: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) which are discussed below.

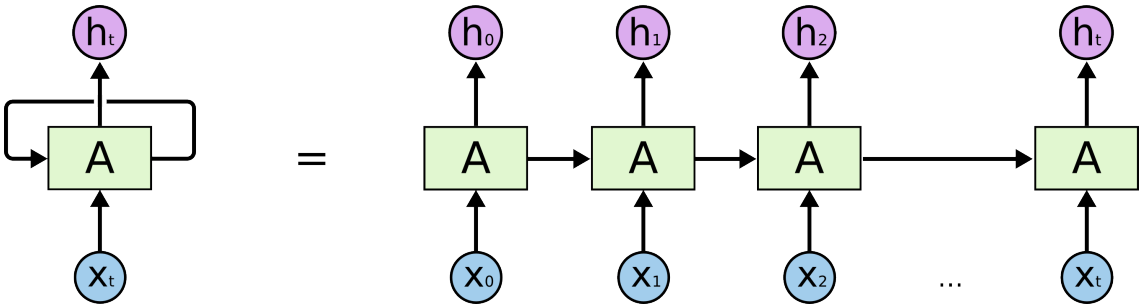


Figure 2.5: Traditional RNN architechture. Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

2.3.3.4 Long Short-Term Memory (LSTM)

Long short-term memory (LSTM) [22] 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 back-propagation. 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 [18], [20], [16]. 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.

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

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

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

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

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

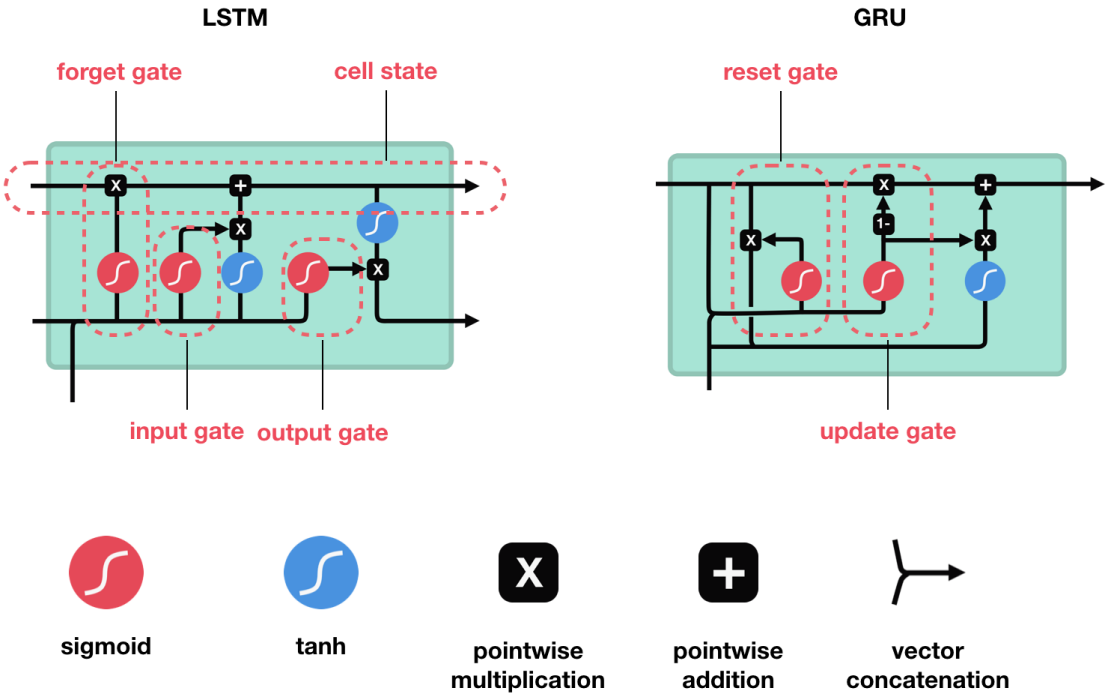


Figure 2.6: LSTM and GRU architechture. Source: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

2.3.3.5 Gated Recurrent Units (GRU)

Gated recurrent unit (GRU) [6] 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 [32] and [4] 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.

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

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

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

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 [35], 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

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: $X_*, f(X_*)$

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 [12] where the velocity of a particular particle is influenced by both the best individual and best personal information. **********.

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

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

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

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

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) [45], which is successfully applied for enhancing bio-inspired optimization algorithms such as GA [45], PSO [46] [44] in solving several optimization problems including finding parameter for deep learning models [39] [37] 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.

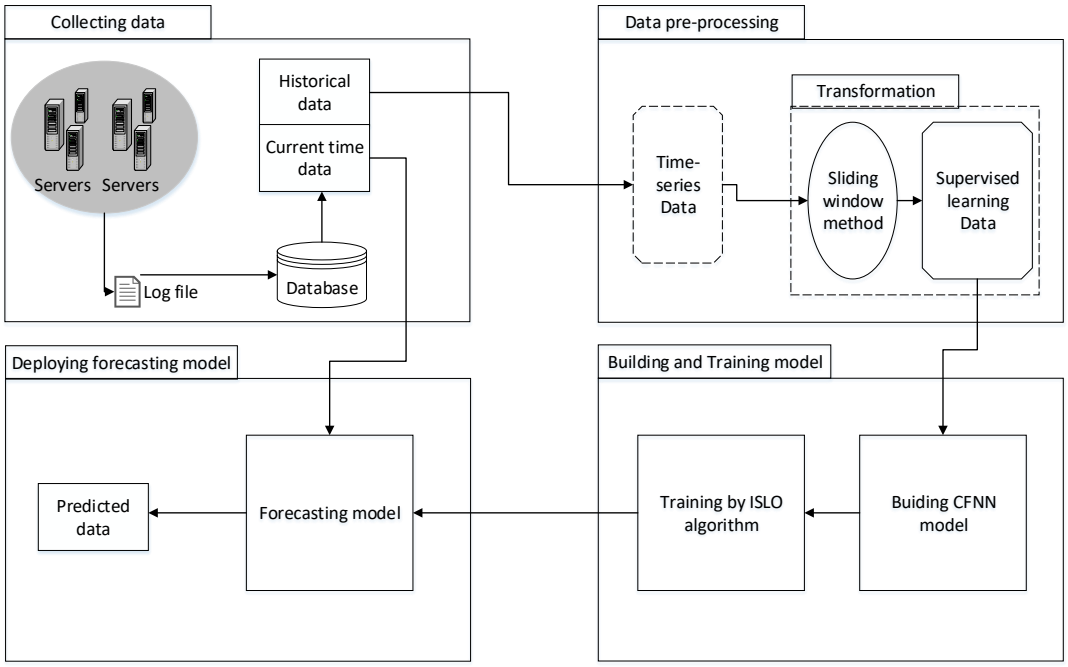


Figure 3.1: Proposed model design

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

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

3.2.1 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 steps as follows:

- Evaluate and choose carefully which columns of data are needed for the forecasting model.

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

- 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).

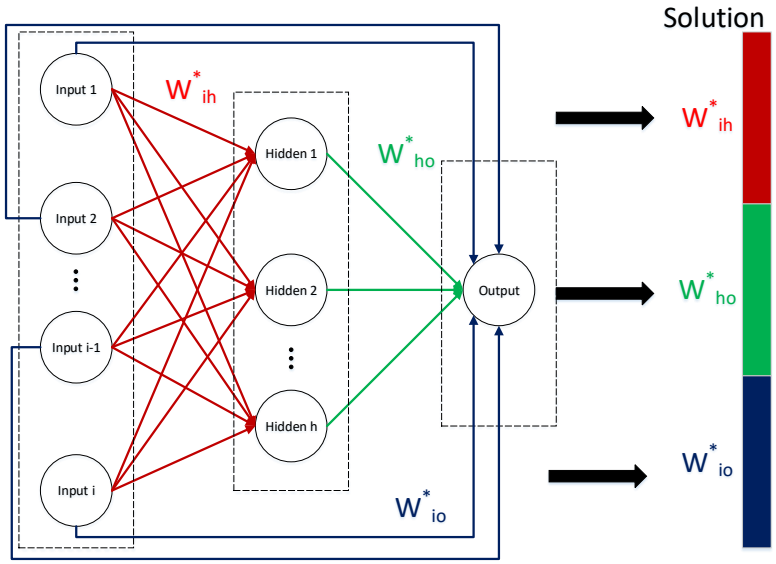


Figure 3.2: Encoding process transforming a parameter set in CFNN into an agent in ISLO algorithm. (W^* indicates weights and biases between 2 layers)

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.

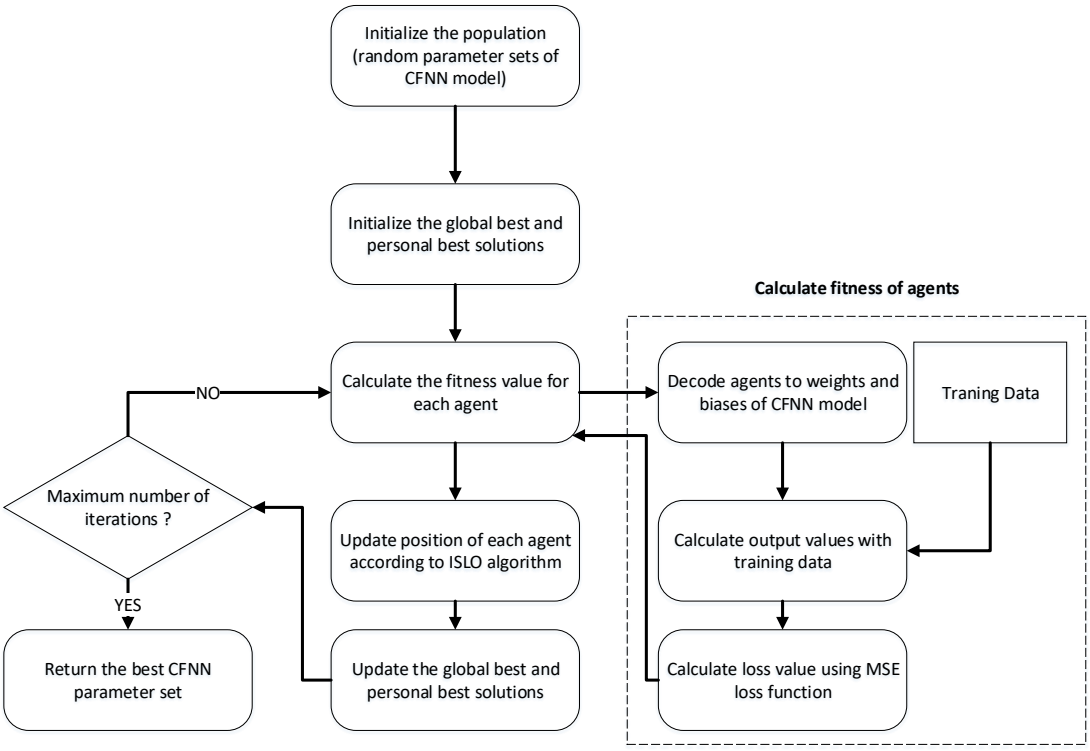


Figure 3.3: The work flow of ISLO-CFNN model.

The workflow of ISLO applied in this work for training CFNN are depicted in Fig. 3.3,

and can be generally presented by the following steps:

1. Initialization: pre-define the number of search agents in ISLO, which are randomly generated in the range [-1, 1]. Each parameter set of CFNN model is encoded to a vector playing a role as an agent in ISLO population. (See in Fig. 3.2)
2. Calculate fitness value for each search agent: The quality of a solution is measured by the loss value of CFNN output. After being decoded into weights and biases vectors, a solution will be applied to form a CFNN model. Data samples in training set is then feed-forwarded through the network, generating predicted output values. Finally, the fitness value is calculated as the difference between predicted and actual output values through the MSE loss function.
3. Update positions of all search agents following formulas of ISLO algorithm.
4. Steps 2 and 3 are repeated until the difference is close enough, or the maximum number of iterations is reached.
5. Return the best CFNN parameter set.

3.2.4 Deploy prediction model

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

Chapter 4

Experiments

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 world.

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 [33] and [34] 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.

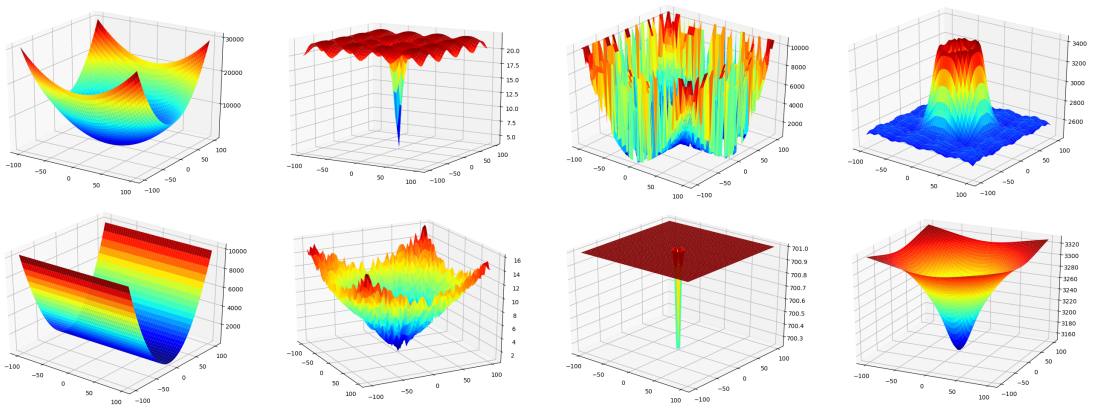
4.1 Theoretical experiments

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

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
$f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert$	[-500, 500]	0
$f_3(x) = max_{i=1,2,...,n} \lvert x_i \rvert)$	[-500, 500]	0
$f_4(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	[-500, 500]	0
$f_5(x) = \sum_{i=1}^{n} i x_i^2$	[-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

- 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 meta-heuristic algorithms. Specifically, they are:

- Generic Algorithm (GA) [48] (Evolutionary algorithms)
- Particle Swarm Optimization (PSO) [29], Whale Optimization Algorithm (WOA) [36] and the original Sea Lion Optimization Algorithm (SLnO) [35] (Swarm-based algorithms)

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

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 RastriginâĂŹs Functions	[-500, 500]	1800
f_{19} (function 19 in CEC 2014) p = [0.2, 0.2, 0.3, 0.3] GriewankâĂŹs, Weierstrass, RosenbrockâĂŹs and SchafferâĂŹs Functions	[-500, 500]	1900
f_{20} (function 20 in CEC 2014) p = [0.2, 0.2, 0.3, 0.3] HGBat , Discus, Expanded GriewankâĂŹs plus RosenbrockâĂŹs and RastriginâĂŹs 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] ScafferâĂŹs , HGBat ,RosenbrockâĂŹs, Modified SchwefelâĂŹs and High Conditioned Elliptic Functions	[-500, 500]	800

- Tug of War Optimization (TWO) [27] (Physics-based algorithms)
- Queuing Search Optimization (QSO) [50] (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 (*std*) value (Eq. 4.1 and 4.2) of 20 times running starting from randomly generated populations for each algorithms. For all

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 , RastriginâĂŹs 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 ScafferâĂŹs 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 ScafferâĂŹs 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 , GriewankâĂŹs plus RosenbrockâĂŹs, Schwefel's, Expanded ScafferâĂŹs, High Conditioned Elliptic, Cigar and and RastriginâĂŹs 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] RastriginâĂŹs , Weierstrass, HappyCat, Schwefel's, Rosenbrock's, HGBat, Katsuura, Expanded ScafferâĂŹs, Expanded GriewankâĂŹs and Ackley Functions	[-500, 500]	1500

the algorithms, a population size and maximum iteration equal to 100 and 500 have been utilized to run on each function 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:

- *mean* values are considered at first. If in a case, an algorithm own the best *mean* value, it

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

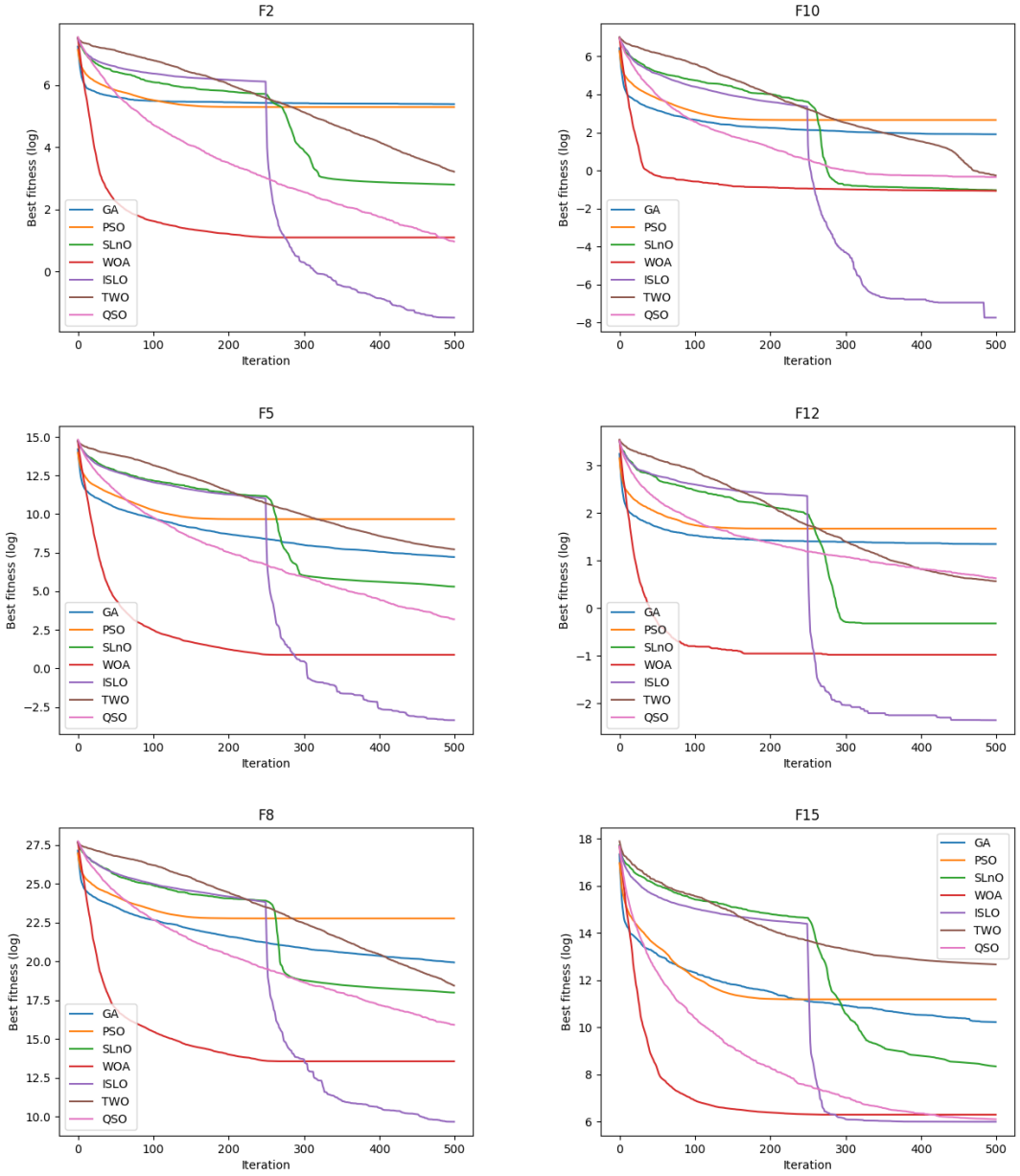
will be ranked as the best optimizer.

- 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

4.1.2.1 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 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

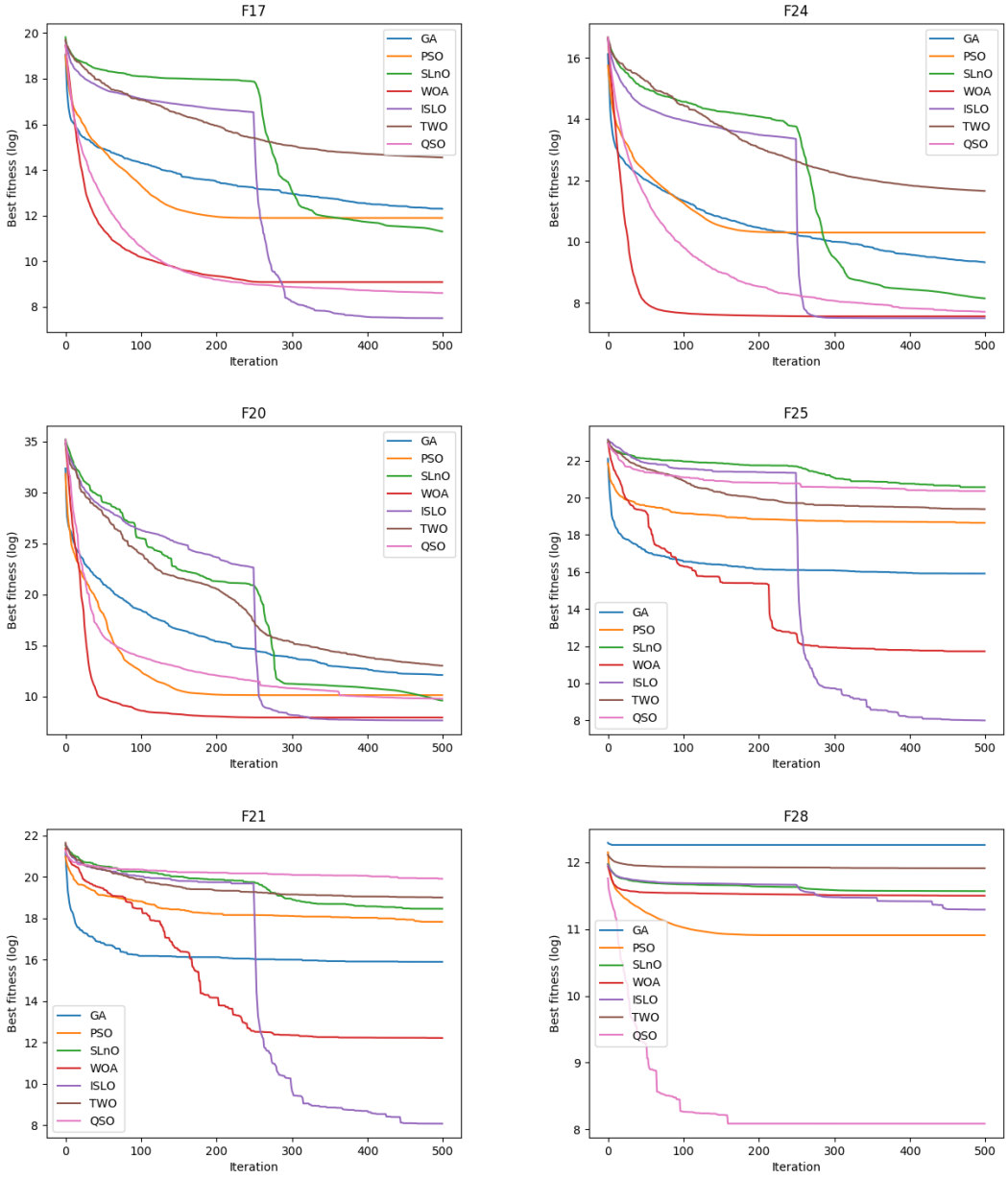
Table 4.6: Comparison of optimization results obtained for the hybrid and composition benchmark functions

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

them are extremely close to ISLO's.

4.1.2.2 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



(a) Hybrid (b) Composition

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

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, and two well-known and widely used models in time-series forecasting: LSTM and GRU in terms of accuracy, run time and the number of hyper-parameters. 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

4.2.1.1 Google Trace dataset

The most important dataset in our experiments is gathered by Google on a cluster of about 12500 machines [40] 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 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.

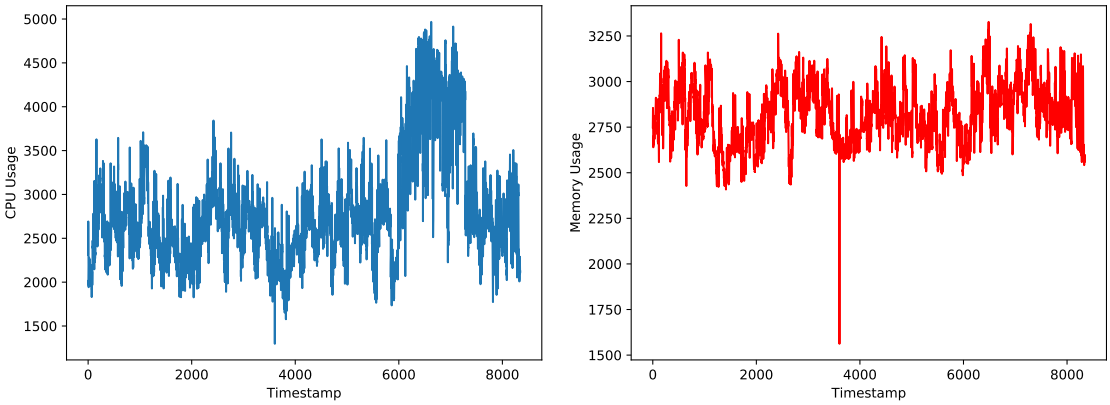


Figure 4.4: Visualization of Google Trace CPU (left) and Google Trace RAM (right) datasets.

4.2.1.2 EU Internet Traffic and UK Internet Traffic datasets

These two sets of data, which is used for experiments in [8], 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.

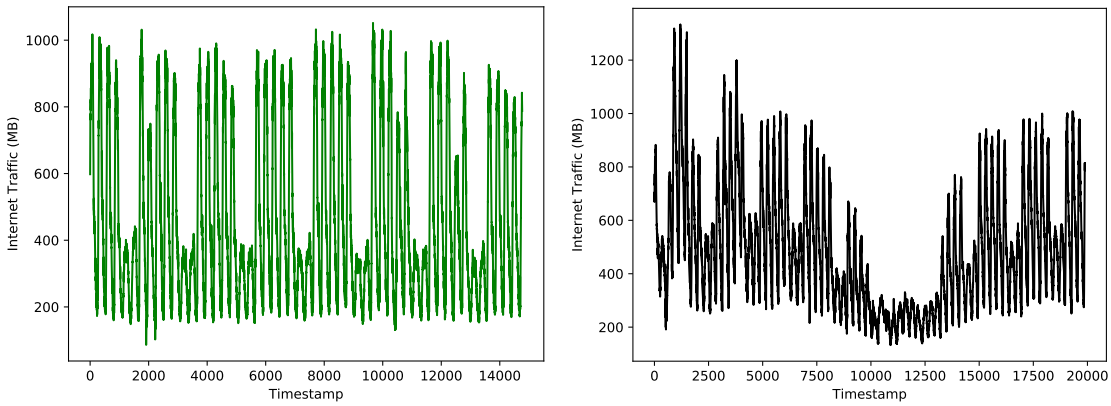


Figure 4.5: Visualization of EU Internet Traffic (left) and UK Internet Traffic (right) datasets.

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 [16].
- 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 [12], 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.

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

Dataset	Model	RMSE	MAE	MedAE	SMAPE (%)
Google Trace CPU	CFNN	199.77	126.80	80.05	4.03
	FFNN	203.54	129.52	85.69	4.18
	LSTM	**200.89**	129.88	84.65	4.10
	GRU	201.88	130.50	82.96	4.12
	RNN	**200.40**	129.30	85.49	4.10
	PSO-CFNN	203.75	127.41	79.74	4.04
	SLnO-CFNN	201.40	**126.60**	**79.64**	**4.02**
	ISLO-CFNN	203.55	**124.94**	**76.08**	**4.00**
Google Trace RAM	CFNN	52.54	35.35	24.87	1.21
	FFNN	53.61	36.51	24.29	1.25
	LSTM	48.47	30.40	20.51	1.05
	GRU	48.43	30.74	20.78	1.06
	RNN	**47.00**	**28.86**	**18.90**	**0.99**
	PSO-CFNN	49.09	31.33	20.67	1.08
	SLnO-CFNN	49.33	31.62	20.53	1.09
	ISLO-CFNN	**47.53**	**29.80**	**19.65**	**1.03**
EU Internet Traffic	CFNN	**15.85**	**11.30**	**8.01**	**2.91**
	FFNN	16.48	11.79	8.41	3.03
	LSTM	**15.84**	**11.37**	**8.30**	**2.92**
	GRU	17.39	13.09	10.29	3.51
	RNN	16.90	12.49	9.55	3.31
	PSO-CFNN	16.02	11.49	8.31	2.93
	SLnO-CFNN	16.37	11.74	8.47	3.00
	ISLO-CFNN	16.02	11.47	8.31	2.93
UK Internet Traffic	CFNN	10.74	8.09	6.31	1.56
	FFNN	11.32	8.40	6.45	1.60
	LSTM	**10.34**	7.65	5.85	1.47
	GRU	11.52	8.89	7.33	1.78
	RNN	11.27	8.38	6.39	1.62
	PSO-CFNN	10.43	**7.59**	**5.63**	**1.44**
	SLnO-CFNN	10.61	7.73	5.75	1.47
	ISLO-CFNN	**10.46**	**7.64**	**5.77**	**1.45**

- For SLnO and ISLO, hyper-parameters are set as described in original paper [35], and also, c_1 and c_2 in ISLO algorithm are the same as shown for PSO.

4.2.3 Results and Discussion

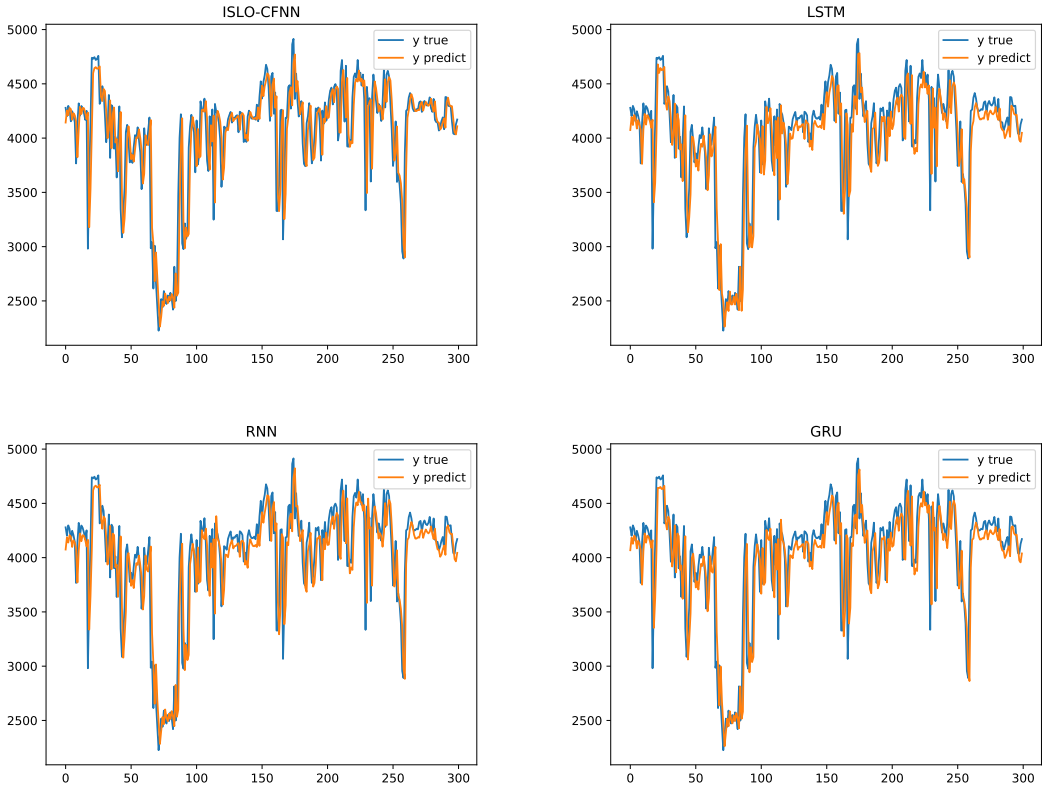


Figure 4.6: Performance comparision between ISLO-CFNN and different recurrent-based deep learning models on Google Trace CPU data.

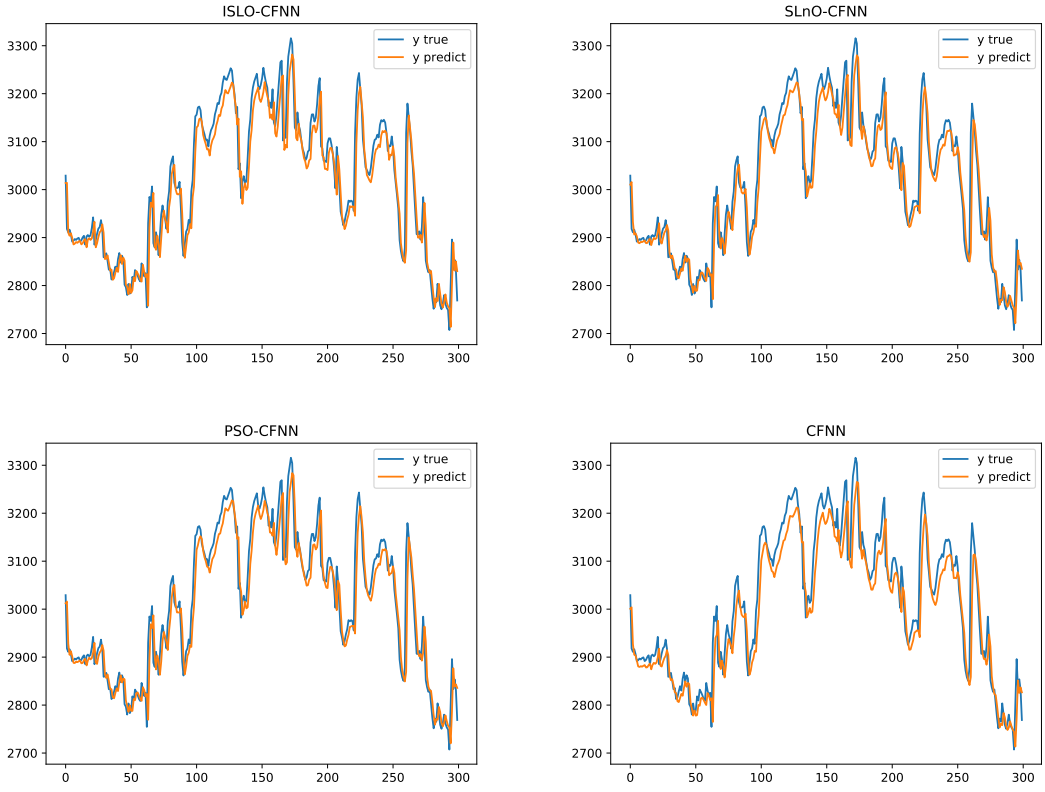


Figure 4.7: Performance comparision between ISLO and other algorithms including Gradient Descent, PSO and SLnO on optimizing CFNN (Google Trace RAM data).

Chapter 5

Conclusions

Bibliography

[1] Bilal Alatas. Acroa: artificial chemical reaction optimization algorithm for global optimization. *Expert Systems with Applications*, 38(10):13170–13180, 2011.

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

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

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

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

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

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

[8] Paulo Cortez, Miguel Rio, Miguel Rocha, and Pedro Sousa. Multi-scale internet traffic forecasting using neural networks and time series methods. *Expert Systems*, 29(2):143–155, 2012.

[9] Dominique de Werra and Alain Hertz. Tabu search techniques. *Operations-Research-Spektrum*, 11(3):131–141, 1989.

[10] Marco Dorigo and Gianni Di Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1470–1477. IEEE, 1999.

[11] Haifeng Du, Xiaodong Wu, and Jian Zhuang. Small-world optimization algorithm for function optimization. In *International Conference on Natural Computation*, pages 264–273. Springer, 2006.

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

[13] Osman K Erol and Ibrahim Eksin. A new optimization method: big bang–big crunch. *Advances in Engineering Software*, 37(2):106–111, 2006.

[14] Kelly Fleetwood. An introduction to differential evolution. In *Proceedings of Mathematics and Statistics of Complex Systems (MASCOS) One Day Symposium, 26th November, Brisbane, Australia*, pages 785–791, 2004.

[15] Richard A Formato. Central force optimization. *Prog Electromagn Res*, 77:425–491, 2007.

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

[17] Zong Woo Geem, Joong Hoon Kim, and Gobichettipalayam Vasudevan Loganathan. A new heuristic optimization algorithm: harmony search. *simulation*, 76(2):60–68, 2001.

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

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

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

[21] Abdolreza Hatamlou. Black hole: A new heuristic optimization approach for data clustering. *Information sciences*, 222:175–184, 2013.

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

[23] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[24] Ali Husseinzadeh Kashan. League championship algorithm (lca): An algorithm for global optimization inspired by sport championships. *Applied Soft Computing*, 16:171–200, 2014.

[25] A Kaveh and M Khayatazad. A new meta-heuristic method: ray optimization. *Computers & structures*, 112:283–294, 2012.

[26] A Kaveh and S Talatahari. A novel heuristic optimization method: charged system search. *Acta Mechanica*, 213(3-4):267–289, 2010.

[27] A Kaveh and A Zolghadr. A novel meta-heuristic algorithm: tug of war optimization. *Iran University of Science & Technology*, 6(4):469–492, 2016.

[28] Ali Kaveh and Vahid Reza Mahdavi. Colliding bodies optimization: a novel meta-heuristic method. *Computers & Structures*, 139:18–27, 2014.

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

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

[31] John R Koza. Genetic programming. 1997.

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

[33] JJ Liang, BY Qu, and PN 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*, 635, 2013.

[34] JJ Liang, BY Qu, PN 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*, 29:625–640, 2014.

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

[36] Seyedali Mirjalili and Andrew Lewis. The whale optimization algorithm. *Advances in engineering software*, 95:51–67, 2016.

[37] Thieu Nguyen, Tu Nguyen, Binh Minh Nguyen, and Giang Nguyen. Efficient time-series forecasting using neural network and opposition-based coral reefs optimization. *International Journal of Computational Intelligence Systems*, 12(2):1144–1161, 2019.

[38] Esmat Rashedi, Hossein Nezamabadi-Pour, and Saeid Saryazdi. Gsa: a gravitational search algorithm. *Information sciences*, 179(13):2232–2248, 2009.

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

[40] Charles Reiss, John Wilkes, and Joseph L Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, pages 1–14, 2011.

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

[42] Sancho Salcedo-Sanz, A Pastor-Sánchez, D Gallo-Marazuela, and Antonio Portilla-Figueras. A novel coral reefs optimization algorithm for multi-objective problems. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 326–333. Springer, 2013.

[43] Dan Simon. Biogeography-based optimization. *IEEE transactions on evolutionary computation*, 12(6):702–713, 2008.

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

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

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

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

[48] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

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

[50] Jinhao Zhang, Mi Xiao, Liang Gao, and Quanke Pan. Queuing search algorithm: A novel metaheuristic algorithm for solving engineering optimization problems. *Applied Mathematical Modelling*, 63:464–490, 2018.