International Journal of Computational Intelligence Systems
Vol. **12(2)**, 2019, *pp.* 1144–1161
DOI: https://doi.org/10.2991/ijcis.d.190930.003; ISSN: 1875-6891; eISSN: 1875-6883
https://www.atlantis-press.com/journals/ijcis/

Efficient Time-Series Forecasting Using Neural Network and Opposition-Based Coral Reefs Optimization

Thieu Nguyen[1], Tu Nguyen[1], Binh Minh Nguyen[1,*], Giang Nguyen[2]

[1] *School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi, Vietnam*
[2] *Institute of Informatics, Slovak Academy of Sciences, Dubravska cesta 9, Bratislava, Slovakia*

ARTICLE INFO

Article History
Received 14 Apr 2019
Accepted 26 Sep 2019

Keywords
Meta-heuristics
Coral reefs optimization
Opposition-based learning
Neural networks
Time series forecasting
Nature-inspired algorithms
Distributed systems

ABSTRACT

In this paper, a novel algorithm called opposition-based coral reefs optimization (OCRO) is introduced. The algorithm is built as an improvement for coral reefs optimization (CRO) using opposition-based learning (OBL). For efficient modeling as the main part of this work, a novel time series forecasting model called OCRO-multi-layer neural network (MLNN) is proposed to explore hidden relationships in the non-linear time series data. The model thus combines OCRO with MLNN for data processing, which enables reducing the model complexity by faster convergence than the traditional back-propagation algorithm. For validation of the proposed model, three real-world datasets are used, including Internet traffic collected from a private internet service provider (ISP) with distributed centers in 11 European cities, WorldCup 98 contains request numbers to the server in football world cup season in 1998, and Google cluster log dataset gathered from its data center. Through the carried out experiments, we demonstrated that with both univariate and multivariate data, the proposed prediction model gains good performance in accuracy, run time and model stability aspects as compared with other modern learning techniques like recurrent neural network (RNN) and long short-term memory (LSTM). In addition, with used real datasets, we intend to concentrate on applying OCRO-MLNN to distributed systems in order to enable the proactive resource allocation capability for e-infrastructures (e.g. clouds services, Internet of Things systems, or blockchain networks).

1. INTRODUCTION

Time series analytics and forecasting play an important role in many real-life areas, especially when the today's development of information technology and data explosion are blooming. The list of these areas gets longer (e.g. predicting human behaviors using collected data from wearable devices, flood forecasting based on historical water levels of rivers and springs, or monitoring financial stock exchange rates). In terms of time series forecasting, past observations are used to model relationships hidden in the data and to predict values in advance. In fact, this modeling way is very useful when there is a little knowledge about underlying data generation process. It is also useful when no adequate mathematical model is available to describe relationships between the predictor and other variables. Over the past several decades, the great effort has been devoted to the development and improvement of time series forecasting using various techniques.

The idea of time series forecasting for the value y at time point t (denoted as y_t) based on the y values at previous time points $y_{t-1}, y_{t-2}, ..., y_{t-k}$, and adding/subtracting error terms was presented in the monograph *"Time Series Analysis: Forecasting and Control"* [1]. The work showed that non-stationary data could be made stationary by differencing the series. The statistical model goes further with the famous autoregressive integrated moving average (ARIMA) model [2], typically expressed in the form of $ARIMA\left(p, d, q\right)$; where p, d, q are autoregressive (AR) terms, non-seasonal differences needed for stationarity and lagged forecast errors in the prediction equation.

To overcome the stationary assumption of statistical models, artificial neural network (ANN or NN) has been extensively studied and widely used also in time series forecasting [3]. The main advantage of ANN is the adaptability to nonlinear models i.e. the model is formulated based on the features detected from the data instead of specifying a concrete model template. This data-based approach is consistent with many empirical data sets, but there is no theoretical guideline to propose a generic data preparation process. Particularly in this direction, deep learning (DL), especially recurrent neural network (RNN) [4] are notable for sequence forecasting. This ability comes from RNN internal self-looped cells able to recognize sequence information. The most well-known RNN building block is long short-term memory (LSTM) [5], which also works well in processing long term time series data. However, DL techniques are compute-intensive that implies longer runtime.

Another way to deal with time series prediction goes through optimization using evolutionary bio-inspired algorithms [6], especially when most real-world optimizations are highly nonlinear and under various complex constraints. This kind of techniques trades

* *Corresponding author. Email: minhnb@soict.hust.edu.vn*

in solution quality for runtime, by finding very good solutions, but not necessarily the optimal one within feasible time. Genetic algorithms (GAs) are a well-known evolutionary nature-inspired algorithm with a wide range of applications. Other algorithms can be listed here are particle swarm optimization (PSO), bacterial foraging optimization (BFO), coral reefs optimization (CRO) and many more.

Based on the state-of-the-art context, our main interest in this work is as follows:

If it is possible to develop a time series prediction model with simple structure, which can tackle the non-linear models and achieve comparable performance in the mean of accuracy and stability but with less runtime requirements in comparisons with other well-known models.

In order to test and evaluate the built model, optimizing operations of distributed applications is considered in this work. Nowadays, the number of distributed applications has been arising due to the extensive advances in distributed systems technology as well as the availability of data in various forms and formats. However, from the developer viewpoint, there are many difficult issues (e.g. consistency management, fault tolerance, security, location transparency, scalability, and performance) must be addressed to build complex distributed applications. The observation suggests that a serious effort to build an e-infrastructure for wide-area state sharing is highly recommended. Such infrastructure could help current applications scale better and significantly increase the rate at which complex distributed applications are deployed in the future. By applying time series data analytics techniques to e-infrastructures provided resources for distributed applications, the infrastructure can achieve the scalability as practical requirement mentioned above. This is also our foreseen of practical usage of this work as (but not limited to) an intelligent module for resource management for distributed/decentralized applications run on modern computing paradigms like clouds, Internet of Things systems, and blockchain networks.

The structure of this paper is organized as follows. Section 2 contains a survey with classification and analytics of existing studies to highlight the aim and contributions of our work. Section 3 and Section 4 present our proposed prediction model designs based on the opposite-base coral reefs optimization (OCRO) for multi-layer neural network (MLNN). In Section 5, we present experiments as well as evaluations for the proposed time series prediction model to prove its efficiency. Section 6 concludes and defines some of our future work directions.

2. RELATED WORK

2.1. Statistic Modeling

One of the most widely known and used time series prediction models is ARIMA [7]. The popularity of ARIMA is due to its statistical properties as well as effectiveness. It is flexible in representing different time series modeling such as standard AR, standard moving average (MA) or AR and MA combinations (ARMA) series. However, their main limitation is the linear form and they are appropriated for a time series that is stationary i.e. it's mean, variance, and autocorrelation should be approximately constant through time [8]. Besides, ARIMA models can not properly capture nonlinear patterns, so the approximation of these linear models to complex real-world problems is not enough to cover the variety characteristic like chaotic or nonlinear dynamic time series data [9].

2.2. Artificial Neural Networks

As stated in Section 1, ANNs have been applied to many areas for time series forecasting in order to overcome the limitations of linear models.

The work [10] employed a four-layered feed-forward neural network (FFNN), which is trained using back-propagation to make hourly predictions of electric load for a power system. Prediction accuracies with 1.07% error on weekdays and 1.80% on weekends were achieved that were superior of prediction accuracy over existing traditional time series forecasting methods. Authors of the work [11] presented an eight-step procedure to design a neural network forecasting model for financial and economic time series. In [12], the authors proposed the use of FFNN for inflows synthesis. FFNN offers a viable alternative also for multivariate modeling of water resources time series. The work [13] did not only employed ANNs for forecasting British pound and US dollar exchange rates but also evaluated the impact of input and hidden neurons number and data size on ANN model performance. The sensitivity analyses showed that the input affect effectiveness more than the hidden neurons and improved accuracies can be gained with larger sample size. In [14], the authors introduced an information gain technique used with machine learning for data mining to evaluate the predictive relationships of numerous financial and economic variables, and used FFNNs to forecast future values. The results show that trading strategies guided by the classification models generate higher risk-adjusted profits than the buy-and-hold strategy as well as those guided by the level-estimation based forecasts of NN and linear regression models. In [15], the authors designed three-layer FFNNs to predict the hourly solar radiation data, the result showed that FFNN outperforms linear prediction filters. Most of the studies reported above were simple applications of using traditional time series approaches and ANNs (FFNNs/MLNNs). In [14], the authors analyzed the drawbacks of MLNNs, which were usually not very stable since the training process may depend on the choice of a random start. Training is also computationally expensive in terms of the times used to determine the appropriate network structure. The degree of success, therefore, may fluctuate from one training pass to another.

The most well-known deep neural network (DNN) types for time series forecasting is RNN. The network type has cyclic connections in its structure, the activations from each time step are stored in the internal state of the network acts as temporal memory. This capability makes RNNs better suited for sequence modeling (i.e. time series prediction) and sequence labeling tasks. So, it has been more general models than FFNNs and has been widely used tool for the prediction of time series [16–20]. However, there are still two issues associated with the traditional RNN models in prediction problem, which are the number of time steps ahead has to be predetermined for most RNNs, and to achieve a better accuracy, finding the optimal time lag setting largely relies on the trial-and-error method. Previous studies [5] have confirmed that the traditional RNNs fail

to capture the long temporal dependency for the input sequence, training the RNN with 510 time lags is proven difficult due to the vanishing gradient and exploding gradient problems. But RNNs are computationally and valuable approximation results more superior than FFNN prediction problems [21–23].

To address these drawbacks of RNNs, LSTM [5] was developed for forecast issue. Unlike traditional RNNs, LSTM is able to learn the time series with long time spans and automatically determine the optimal time lags for prediction. In the past two decades, LSTM has been successfully applied to robot control, speed recognition, handwriting recognition, human action recognition, transportation, etc. Especially, there are so many studies in time series prediction problem using LSTM such as [24–28]. Moreover, LSTM comes with a number of different variances such as gated recurrent unit (GRU), bidirectional LSTM, LSTM autoencoders [29,30], that promise further model quality improvements. However, the cost is often higher complexity, which requires a longer training.

Here also is suitable to mention that convolution neural network (CNN) is a strong candidate for sequence forecasting [31,32]. The combination of CNN and attention function brings the new architecture of temporal convolution nets (TCN), which outperforms RNN in several language-to-language translation benchmarks [33] at both speed and accuracy performances. In comparison with RNN, CNN structure is more natural to parallelize and this can be taken advantage to exploit accelerator supports.

Even DL and RNNs are considered as current state-of-the-art in the time series prediction field for a broad audience, it still has drawbacks. Concretely, DL requires a lot of data to train in order to gain more accurate, as well as a computational requirement because of the model complexity [34–36]. Particularly, RNNs are not parallelism-friendly due to their nested loop structures. However, the researches in this direction are interesting with high dynamics, that bring many advanced computing methods with promising results. It is also clear that the concern about performance from the speed viewpoint, as well as computational requirements still remains.

2.3. Neuro-Evolution

In "neuro-evolution" research community, GAs [37] are known as efficient search algorithms. Recently, GA-based research has been applied to NN models [38,39] in order to address computational cost, easy implement in hardware, and better accuracy. This direction is attracted to lots of researcher in many fields such as manufacturing process [40], stock markets [41], energy consumption [42], medical diagnosis [43], resource usage in cloud [44]. Unfortunately, recent research [45] identified some of the limitations in performance of GAs with problems have high epistasis targeting functions, the performance degradation is huge, and GA's early convergence lowers its performance and reduces its search capabilities.

In the term of *combining NNs and optimization* algorithms, swarm intelligent (SI) optimization motivated by social behavior in the biological system also has attracted many researchers. PSO was stimulated by swarm behavior of bird flocking or fish schooling has been used in several real-life area [46]. BFO, which is an evolutionary computing technique inspired based on the principle of bacterial movement like tumbling, swimming, or repositioning to food-seeking [47]. An improved version of BFO is Adaptive BFO with life-cycle and social learning (ABFO) was developed and presented in [48]. It has been tested on several sets of benchmark functions with multiple dimensions and used in distributed systems like cloud computing [49]. In a recent development, a bunch of meta-heuristics swarm-based intelligence algorithms were developed, which can be used to optimized NNs. The list contains Mothflame optimization algorithm [50], Whale optimization algorithm [51], Salp Swarm Algorithm [52], Grasshopper optimization algorithm [53], Harris Hawks optimization [54], Butterfly optimization algorithm [55], Sailfish optimization algorithm [56], and many more.

2.4. Coral Reefs Optimization

This technique also belongs to SI optimization algorithms motivated by social behavior in the biological systems like PSO, BFO, and other swarm-based heuristics. It is proposed in [57] tackling optimization problems by modeling and simulating corals reproduction and formation. A lot of applications have been carried out in many areas such as energy prediction [58,59], sensor networks [60], and cloud resource allocation [61].

The original CRO algorithm simulates a coral reef, where different corals grow and reproduce in coral colonies, fighting by choking out other corals for space in the reef. This fight for space produces a robust meta-heuristic algorithm powerful for solving hard optimization problems. According to [62], the exploitive power of CRO is controlled by broadcast spawning, which carries out most of the global searching and brooding that could help jump out of the local optima. In this approach, a fraction of healthy reef can duplicate itself and larvae setting process controls local searching by a simulated annealing (SA) alike process as exploitive power (mostly) executed in the budding process.

Like other meta-heuristic algorithms, CRO has also the problem with local optimums. The diversity of corals in a reef quickly decreases after a certain number of iterations, which causes stuck in a local area and it is hard to jump out of it. In order to improve the CRO's searching power, we propose an improvement that combines the technique with a well-regarded mathematical concept called opposition-based learning (OBL) initially proposed in [63]. OBL can attain the opposite locations for candidate solutions for a given task. The new location can provide a new chance to become aware of a neighboring point to the best position. In this work, we call our new improvement under the short-name *OCRO*, which stands for OBL in combination with CRO.

2.5. Contributions of the Work

Currently based on our knowledge, there is no study that applies the CRO to MLNN and OCRO to MLNN. In comparison with abovementioned works, the main differences and contributions of our work are as follows.

1. Proposing a new improvement called opposition-based coral reefs optimization (OCRO), which improves the original CRO algorithm using OBL. OCRO aims to improve searching power

and jumping out from the local minimum of the meta-heuristic CRO.

2. From the theoretical viewpoint, when the combination of CRO and OBL is property applied to MLNN, it improves the drawbacks of gradient descent algorithm, enables fast convergence to optimal values as well as reducing computational cost as the whole.
3. Proposing the new time series forecasting model called OCRO-MLNN, which is designed based on MLNN variance with OCRO algorithm to train the forecasting model instead of back-propagation.
4. Carrying out comparisons among the novel proposed prediction model with five well-known ones such as MLNN, GA-MLNN, CRO-MLNN, RNN, and LSTM for time series forecast.
5. Evaluating and proving the effectiveness of OCRO-MLNN as compared with others using three kinds of real-world datasets (i.e. above-mentioned European (EU) traffic, world cup (WC), and Google trace). The gained outcomes show that our prediction model OCRO-MLNN provide good performance in predictive accuracy, model stability, as well as runtime.

3. OPPOSITION-BASED CORAL REEFS OPTIMIZATION

3.1. Coral Reefs Optimization

As mentioned above, CRO is an optimization algorithm (bio-) inspired by behaviors of corals reproduction and reef formation. It was originally introduced in [57]. The skeleton of the original CRO is summarized in Algorithm 1 that includes two main below-described parts.

Algorithm 1: CRO algorithm

Initialization
Create a $N \times M$ square grid and randomly assign some squares to be occupied
while *not stopCriteria()* **do**:
Broadcast spawning
A fraction of p_k coral larvae formed by external reproduction
Brooding
The rest of $(1 - p_k)$ larvae formed by internal sexual reproduction
Larvae setting
Asexual reproduction
Depredation in polyp phase
Return the best solution.

Part 1: Initialization of CRO parameters.

The main control parameters of CRO are as follows: a coral reef consists of an $N \times M$ square grid that is similar to the population size in GA. The girds are selected randomly and can be assigned to a coral or colony of coral, representing a solution to the given problem, which is encoded as a string of numbers in a given alphabet. The rate p_0 between selected grids (occupied squares) and not selected ones (free/empty squares), which is an important factor to control the exploration ability of algorithm (note that $0 < p_0 < 1$). The health function f is similar to the GA fitness function. The underlying idea behind CRO is like the reef progress, the healthier corals are, the better the chance they can survive. The healthy corals present a better solution to solving the problem.

Part 2: Reef information.

1. *Broadcast Spawning* (external sexual reproduction):
   (a) Select uniformly at random a fraction of existing corals (p_k) in the reef to be broadcast spawner, denoted as F_b. The remaining ones denoted as $(1 - F_b)$ will be formed by internal sexual reproduction.
   (b) Broadcast spawner couples will produce a coral larva by sexual crossover. These couple selection can be done uniformly at random or by resorting to any fitness proportionate selection approach (e.g. roulette wheel). Note that once two corals have been selected to be the parents of larvae, they are not selected anymore in the broadcast spawning phase.
2. *Brooding* (internal sexual reproduction): As described in Step (1), the fraction $(1 - F_b)$ of corals will be reproduced in this stage by means of a random mutation (called brooding modeling or brooding-reproductive coral). The newly produced larvae are released to the water together with larvae formed in Step (1).
3. *Larvae setting:* When all the larvae are formed by the broadcast spawning or by brooding, the process of setting and growing in the reef will be performed. Firstly, the health function of each larva is evaluated. Then, each larva will randomly settle down in the *grid* (i, j) of the reef. If the grid is free space, coral can grow regardless of its value of health function. However, if the grid is already occupied by a coral, new larvae will set in this place only if its health function is better than the existing one. We define a number k of attempts for a larva to set in the reef, after k unsuccessful tries, it will be deprecated by animals.
4. *Asexual reproduction* (budding or fragmentation): According to the values of health function, existing corals are rearranged in the reef. A proportion of F_a of the existing corals will copy itself and attempt to settle in a different part of the reef by Step (3). Note that a maximum number of identical corals μ are allowed in the reef.
5. *Depredation in polyp phase:* In the process of reef formulation, corals also die, and their space is freed up for newly generated corals. The depredation operator is applied with a very small probability P_d and exclusively to a fraction F_d of the worse health corals. For the sake of simplicity in the parameter setting of CRO algorithm, the value of this fraction may be set to $F_d = F_a$. Any other assignment may be applied based on $F_d + F_a \leq 1$ i.e. no overlap between the asexually reproduced and the deprecated coral sets.

3.2. Opposition-based Coral Reefs Optimization

According to [62], the ability to exploit of CRO is controlled by broadcast spawning. This carries out most of the global searching

and brooding, which helps to jump out of the local optimal. As for exploitive power, mostly executed by building process, where a fraction of healthy reef can duplicate itself and larvae setting process controls local searching by SA like process. Like other meta-heuristic algorithms, CRO is also faced with trapping in local optimal. The diversity of corals in reef quickly decrease after several iterations, causes the algorithm stuck in the local area and hard to jump out of it.

In order to improve the CRO's searching power, we propose an improvement (called OCRO in short name) that combine it with OBL [63] mathematical concept well-known in reinforcement learning, ANNs, and fuzzy systems. All code and related materials are stored as open-source at [64]. OBL indicates that for finding the unknown optimal solution, searching both a random direction and its opposite simultaneously gives a higher chance to find the promising regions and to enhance the algorithm performance [65]. Our improvement for CRO based on OBL is built as follows.

- OBL attains opposite locations for candidate solutions for a given task. The new location can provide a new chance to become aware of a neighboring point to the best position.
- OBL is applied in the CRO depredation phase instead of removing the worst health fraction of reef. We calculate the potential of the oppositional solution C^{op} for every C in worse health corals and compare it with the original one, then retain if it had better fitness.

Oppositional coral is generated by Equation (1):

$$C^{op} = L_b + U_b - C^{best} + r \cdot \left(C^{best} - C\right) \tag{1}$$

where

C^{op}	is the position of opposite coral C inside the search space,
L_b	is the lower bound of the search space,
U_b	is the upper bound of the search space,
C^{best}	is the best coral (the best solution),
r	is a random vector with elements inside range $(0, 1)$.

The improvement of depredation step is formed through Algorithm 2.

Algorithm 2: OCRO - Improvement of CRO depredation step

Randomly select a number of worse health coral $DeCorals = F_d \cdot totalCorals$
for C in $DeCorals$ **do**
$C^{op} = L_b + U_b - C^{best} + r \cdot \left(C^{best} - C\right)$
if C^{op} health better than C health **then**
C^{op} replace C in reef
else
Depredate C and free space in reef

To prevent premature convergence to a local optimum, in our improvement, we restart the search process when the best solution is not improved after several generations based on [66]. When the reef is restarted, the best coral is maintained (elitism criterion) and the rest of corals are randomly initialized. The results of our new proposed OCRO algorithm, which is the combination and improvement of CRO with OBL, are optimistic as presented and discussed below in Section 5.2.

4. NONLINEAR TIME SERIES FORECASTING USING NEURAL NETWORK AND OPTIMIZATION

4.1. Time-Series Modeling Using Neural Network

ANNs (in general) and MLNNs (in particular) are flexible computing frameworks for modeling a broad range of nonlinear problems. In comparison with other nonlinear models, MLNN advantages are their ability as universal approximators with a large class of functions producing high accuracy degree. This advantage comes not only from the fact that NN model is based on data characteristics (hence, no prior assumption of the model form is required), but also the ability to process information from the data in parallel. MLNNs are often used for pattern classification and recognition. Recently, FFNNs with single hidden layer are also widely used as time series forecaster [3]. The model includes three layers which are input, hidden, and output layer as represented by Figure 1 (left side). Each of them has simple processing units connected by acyclic links. The relationship between outputs (y_{t+k}) and inputs ($y_{t-1}, y_{t-2},..., y_{t-p}$) is described in Equation (2).

$$y_{t+k} = \beta_{0k} + \sum_{j=1}^{q} \beta_{jk} \cdot g\left(\alpha_{0j} + \sum_{i=1}^{p} \alpha_{ij} \cdot y_{t-i}\right) \tag{2}$$

$$g(\gamma, x) = \begin{cases} \gamma \cdot (e^x - 1), & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \tag{3}$$

where

α_{ij}	is connection weight for $i = 0, 1, ..., p$ and $j = 1, 2, ..., q$,
β_{jk}	is connection bias for $i = 0, 1, ..., p$ and $j = 1, 2, ..., q$,
p	is the number of input nodes,
q	is the number of hidden nodes,
w	is the number of output nodes,
$g(\gamma, x)$	is activation function of input x and γ.

In the past, logistic functions were often used as activation function but recently exponential linear unit (ELU) presented by Equation (3) (general formula) gains more interests ($\gamma = 1$) and thus it is widely used. If $\gamma = 0$ then ELU becomes rectified linear unit (ReLU). If the number of hidden neurons is large enough, the network is shown in Equation (2) can approximate arbitrary function [67]. In practice, a simple network structure with a small number of hidden nodes often works well in forecast applications. This may be due to the over-fitted model, which has a good fit for the sample used but has poor generalization ability for data out of those samples when the number of hidden nodes too large. The characteristics of MLNN structure covers:

- The choice of q depends on data and there is no systematic rule in deciding this parameter.

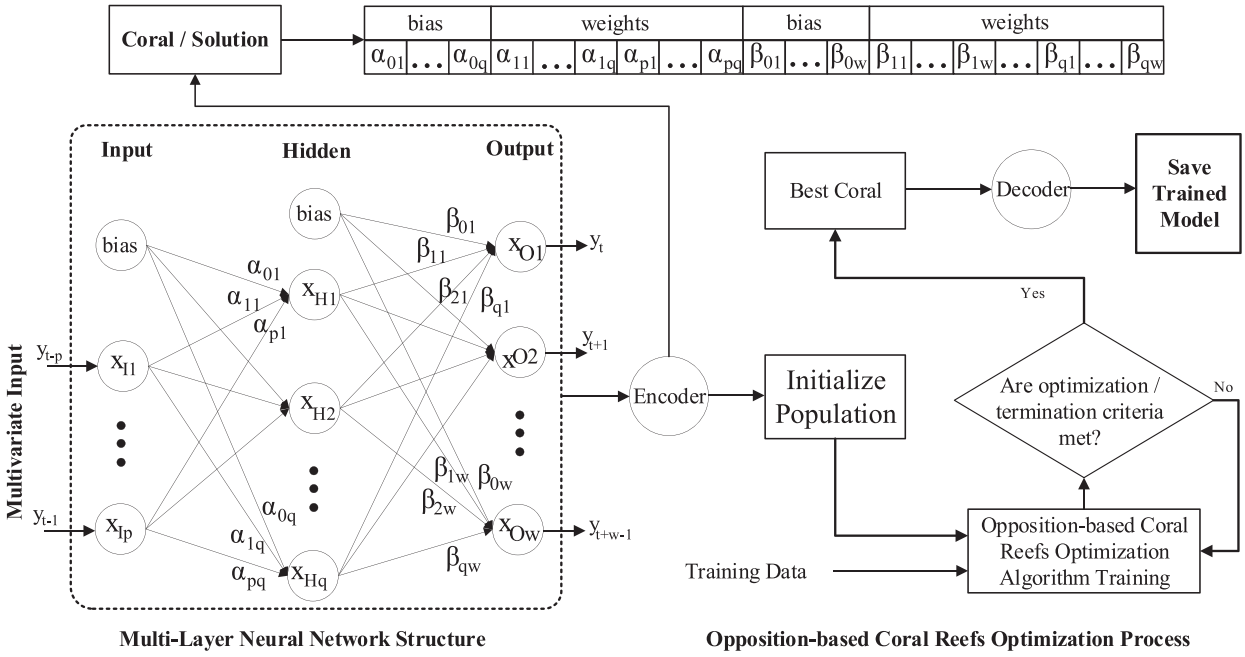

Figure 1 | Opposition-based coral reefs optimization (OCRO)-multi-layer neural network (MLNN) training process.

- The same situation as mentioned above also occurs when choosing the number of lagged observations p as the dimension of the input vector. This is the most important parameter to estimate in NN models due to its major role in determining the (nonlinear) autocorrelation structure of time series. However, there is no theory guide to select p. Hence, experiments are often conducted to choose an appropriate p as well as q.
- In time series forecasting, w parameter represents predicting the number of ahead steps.One-step-ahead forecasting comes with ($w = 1$). Multi-step-ahead prediction comes with ($w > 1$) and w is usually chosen based on the work's purpose.

Once the structure (p, q, w) is specified, the NN is ready for training. The parameters are estimated by minimizing overall accuracy criteria such as mean absolute error (MAE), and mean squared error (MSE) rather using efficient nonlinear optimization algorithms such as GA and CRO than back-propagation. Based on this approach, our new improvement OCRO-MLNN is described in the following Subsection.

4.2. OCRO-MLNN Forecasting Model

As presented before, our prediction model uses OCRO algorithm to optimize the selection of weights and biases of hidden and output layers. Thus, we focus on improving the overall performances, including accuracy, speed of convergence and stability of the MLNN network instead of using the back-propagation technique. The number of variables estimated by OCRO is as follows.

$$q\left(p+1\right)+w\left(q+1\right) \tag{4}$$

where p, q, w are the number of nodes in input, hidden, and output layers of MLNN accordingly.

Range of individual variables is set to [−1, 1]. The used health function as given in Equation (7).

The learning process of OCRO-MLNN is illustrated by Figure 1. To obtain the multivariate input, raw data is preprocessed by the following steps.

1. Firstly, a time series data is scaled into the range of [0, 1].
2. Secondly, time series data is transformed to supervised data using the sliding method with window width k, where k is prior observations before time t used to predict the observation at t.
3. Finally, all metric types are grouped into multivariate input. Encoder and decoder are two important components in Figure 1. In which, encoder encodes NN weights and biases into our domain solution (i.e. coral presented as real-value vector). Decoder decodes corals back into NN weights and biases.
4. In order to avoid long training time, a termination criterion is designed for earlier-stopping after the number of epochs (training cycles) is predefined or after the achievement of error goal.

Algorithm 3 describes the MLNN operation with OCRO. The parameters are shown in Table 1.

5. EXPERIMENTS AND EVALUATIONS

5.1. Evaluation Approach

Under assumption of the same datasets, settings, and testing environment, two experiment groups are carried out to evaluate the proposed OCRO-MLNN model, covering:

1. Evaluating the proposed model on both univariate and multivariate data of different datasets.

Algorithm 3: OCRO-MLNN forecasting model

Input: $g_{max}, N, M, \rho_o, F_b, F_a, F_d, k, \tau$
Output: C_{best} the best coral in reef based on their health.
1: Initializing reef space includes $N \times M$ square grid, Assign some squares to be occupied by corals with respect to rate ρ_o = free/occupied, Find the C_{best} and $n_{iters} \leftarrow 0$
2: **for** i = 0 to g_{max} **do**
3: $larvaes \leftarrow \phi$
4: $C_{broadcast} \leftarrow$ take out a fraction F_b of coral reefs.
5: $C_{brooding} \leftarrow$ Corals in reef, exclusive in $C_{broadcast}$
6: for C_1, C_2 in $C_{broadcast}$ **do**
7: Crossover C_1 and C_2 to create new larva and then add that larva to *larvaes*
8: for C_b in $C_{brooding}$ **do**
9: Mutation C_b to create new larva and then add that larva to *larvaes*
10: **for** *larva* in *larvaes* **do**
11: **for** j = 0 to k **do**
12: square $\leftarrow$ select a random square in reef
13: **if** square is empty or $f(larva) \geqslant f(square)$ **then**
14: square $\leftarrow$ *larva*
15: **break**
16: $C_{duplicate} \leftarrow F_a$ fraction of healthy corals
17: $C_{depredation} \leftarrow F_d$ fraction of worse health corals
18: Run step 11 to 16 (Larvaes setting) for $C_{duplicate}$
19: **for** C_d in $C_{depredation}$ **do**
20: $C_{op} \leftarrow L_b + U_b - C_{best} + r \cdot (C_{best} - C_d)$
21: **if** $f(C_{op}) \geqslant f(C_d)$ **then**
22: Replace C_d by C_{op}
23: **else**
24: Free space at C_d position
25: Find the best current coral C'_{best}
26: **if** C'_{best} is the same as C_{best} **then**
27: $n_{iters} \leftarrow n_{iters} + 1$
28: **if** $n_{iters} \geqslant \tau$ **then**
29: Restart searching, keep C_{best} and $n_{iters} \leftarrow 0$
30: Return C_{best}

Table 1 Opposition-based coral reefs optimization (OCRO) parameters.

Name	Description
g_{max}	Maximum number of generations
$N \times M$	Number of square grids in reef
ρ_o	The rate between free/occupied squares in reef at the beginning of this algorithm
F_b	The fraction of broadcast spawners with respect to the overall amount of existing corals
$1 - F_b$	The fraction of corals that will reproduce by brooding
F_a	A fraction of reef that will duplicates itself
F_d	A fraction of worse health corals will be depredated
k	Number of attempts for a larva to set in the reef
τ	Number of iterations before restarting the algorithm

2. For each test, prediction accuracy, run-time, and model stability among various NN models are compared with our OCRO-MLNN model.

Other NN models used for comparison include traditional MLNN and its variances with optimization techniques such as GA-MLNN, PSO-MLNN, ABFO-MLNN, CRO-MLNN, traditional RNN and the state-of-the-art LSTM.

5.1.1. Datasets

Univariate data. Univariate data used in our experiments are two real and well-known datasets, covering:

- "Internet traffic data (in megabytes)" [68] collected from a private internet service provider (ISP) with distributed centers in 11 EU cities (in short from here EU dataset). The dataset corresponds to a transatlantic link and was collected from June 7th to 11:17 hours on July 31st, 2005 in five minutes intervals.
- The second dataset (called WorldCup98, in short from here WC dataset) contains request numbers (in thousands) to servers in world-cup season between April 30th, 1998 and July 26th, 1998. The dataset is processed into five minutes intervals in our work.

Multivariate data. Multivariate data used in our experiments are gathered by Google from their production data center clusters (called Google trace dataset, in short from here Google dataset). The log records come from approximately 12000 servers for one month [69] and [70]. Based on our prior analysis presented in [44], we select both resource usages, which are central processing unit (CPU) and memory metric as multivariate inputs for our proposed models. The dataset also was further processed into five minutes intervals in the experiments. Visualization of both data type is illustrated in Figure 2.

Due to the characteristics of time series data, which is ordered time-dependency sequence, hold-out validation with 70:15:15 ratio is used as the splitting parameter for training/validation/testing datasets. This splitting approach is designed also as adjustable time slider frame for cross-validation for potential time-growing datasets [71,72].

5.1.2. Measurement methods

Accuracy. In our work, root mean square error (RMSE) (Eq. 5) is used mainly to evaluate error between predicted and ground true value. It is also employed as the fitness function in GA, PSO, ABFO, and as the health function in CRO and OCRO. However, for objectiveness, other metrics such as Coefficient of Determination (R^2), MAE, mean absolute percentage error (MAPE), and symmetric mean absolute percentage error (SMAPE) are also used to comparison goal. The mathematical formulations of those metrics are presented in Equations (6–9), respectively, where y contains ground truth values, $\hat{y}$ contains forecasting values and $fitness = health = RMSE$.

$$RMSE = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} \left(y_i - \widehat{y_i}\right)^2} \tag{5}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(y_i - \widehat{y_i}\right)^2}{\left(\sum_{i=1}^{n} \left(y_i - \frac{1}{n} \sum_{j=1}^{n} y_j\right)\right)^2} \tag{6}$$

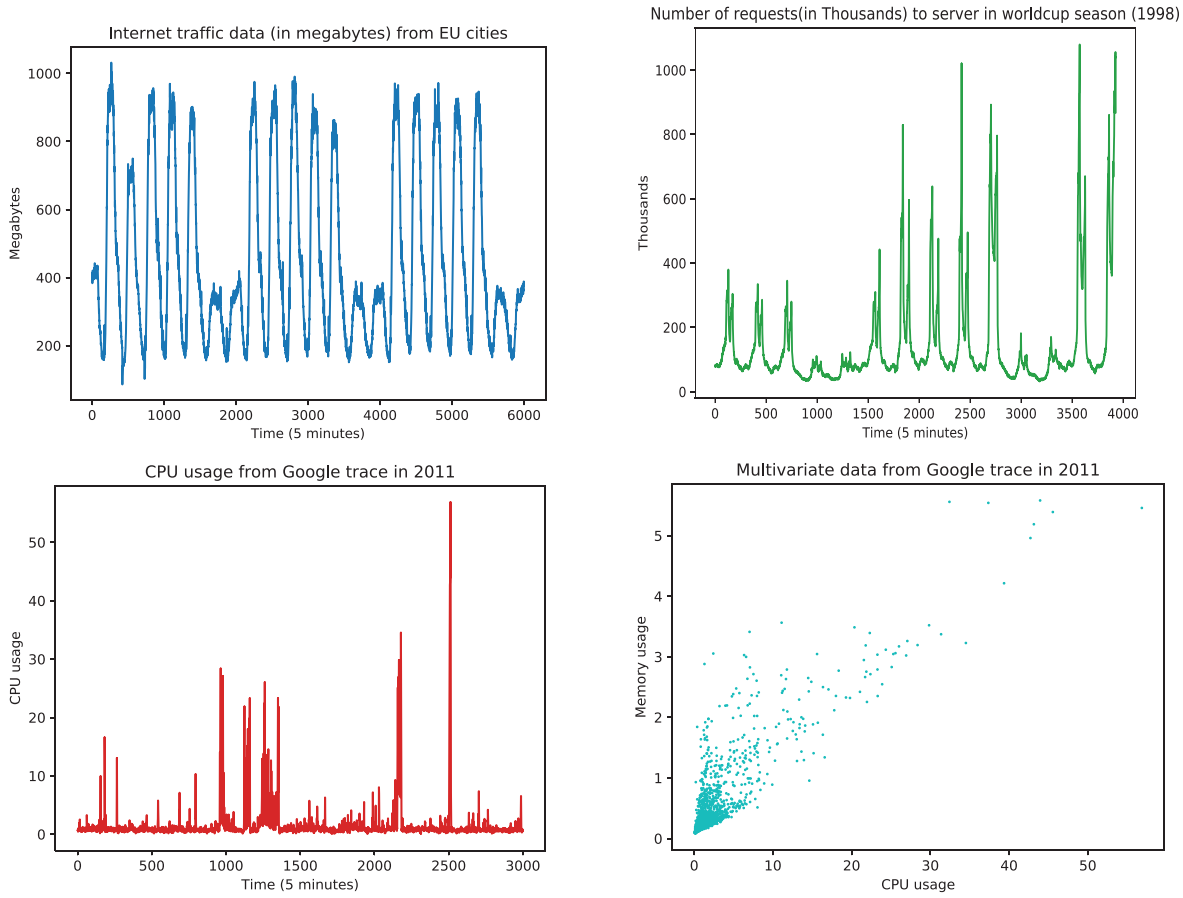

Figure 2 Dataset visualizations: univariate data (upper row) and multivariate data (lower row).

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y}_i| \tag{7}$$

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \widehat{y}_i}{y_i}\right| \tag{8}$$

$$SMAPE = 100\%\frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \widehat{y}_i|}{\left(|y_i| + |\widehat{y}_i|\right)/2} \tag{9}$$

Runtime. The proposed model OCRO-MLNN is also compared with other models in runtime performance-wise (speed) aspect. Currently, RNN and particularly LSTM are state-of-the-art for time series prediction models. The fact is their complicated structure and training runtime often is long. Hence, it is desired to have a model with at least comparable good prediction performance but less expensive on computational requirement. The runtime comparison in our experiments is done according to three factors:

- t_e is the average time for 1 epoch (second/epoch).
- t_p is time to predict test data (in second).
- t_s is the total time of all system processing (preprocessing, training, and testing) The unit is also second.

Because each model has different epoch configuration, t_e can give the most objective insight into the model complexity instead of the total time of training process. However, t_s is definitively the decisive factor that everyone cares about the most.

5.1.3. Settings parameters for algorithms and models

As described above, OCRO-MLNN is validated against other modes such as MLNN, GA-MLNN, PSO-MLNN, ABFO-MLNN, CRO-MLNN, RNN, and LSTM.

- MLNN part in the all models listed above is configured with the same structure configuration with three layers: input, hidden, and output layer. The architecture is similar to typical topology of one RNN/LSTM block [25].
- Input size $p = k \cdot m$, where k is sliding window and m is the number of metric types in dataset (i.e. $m = 1$ for univariate and $m > 1$ for multivariate). The number of neurons in the hidden layer is set by $q = 5$ for all models. The output size $w = 1$.
- In RNN and LSTM models, NN is composed from one input layer, one RNN/LSTM block and one output layer with similar parameter settings. Activation function used in the models is ELU as described above in Section 4.
- To be fair in all test, the population size p_s and maximum number of generations g_{max} of meta-heuristics such as GA, PSO, ABFO, CRO, and OCRO is set to the same 200 and 700, respectively [44]. The number of generations of MLNN is set to

5000 because of the slow convergence of BP algorithm. Meanwhile, the number of the epoch in RNN and LSTM is 1000 to avoid the massive timing cost.

Besides above-described common specifications for fairness testing, the following settings are applied to algorithms for their best tested performance.

- For GA, probability of two individuals exchanging crossovers p_c is set to 0.95 and probability of individual mutation p_m is set to 0.025 based on our prior work [44].
- For PSO, based on [73], inertia factor w is set linearly reducing with iteration from 0.9 to 0.4, cognitive learning rate $c1 = c2 = 1.2$.
- For ABFO, according to our prior experiment [49], swimming length is set $N_s = 4$, probability for eliminate bacteria $P_e = 0.25$, parameters used to control and adjust split and dead criterion $N_{split} = 30$, $N_{adapt} = 5$, step size at the beginning of process $C_s = 0.1$ $(U_b - L_b)$, step size at the end of process $C_e = 0.00001$ $(U_b - L_b)$, where U_b and L_b are referred to the upper and lower bound of the variables.
- For CRO, based on [57], initial free/occupied square (p_o) should be enough to allow new poorer solutions in order to have enough survival probability, so p_o is set by 0.4. The maximum number of generations $g_{max} = 1000$. The depredation probability P_d linearly increases from 0 as initial value to 0.1 at the end of algorithm operation. A high value of corals applying broadcast spawning is needed in order to ensure an efficient exploration of the search space. On the other hand, a small value of brooding and asexual reproduction is advisable, therefore we set $F_b = 0.9$ and $F_a = 0.1$. We also find out that with 250 corals in reef and number k of attempts for a larva is set in the reef equals to 3 that is good for our problems [57].
- For OCRO, most of parameters are set similar to CRO except broadcast spawning fraction F_b, depredation fraction F_d, and τ, which is the number of iterations before restarting algorithm. In order to make opposition based on searching work efficiently, a high fraction of corals is supposed to be applied by this technique. But on the other hand, the number of corals in reef also need to be stable during the training. So, with a small change in fraction of broadcast spawning, and budding that helps more larva can be created during sexual reproduction phase.
  - For τ, it must be small enough to prevent premature convergence but also not too small that may cause algorithm only working on global searching and cannot converge to optimal solution.
  - Our proposed F_b is set to 0.8 (more corals for budding that is one coral create one larva, while in broadcast spawning we need a couple to have a new larva).
  - $F_d = 0.3$ and $\tau = 55$ are supposed to be proper after testing on our datasets.

5.2. Experiment Results

5.2.1. Univariate data

Prediction accuracy. Table 2 describes obtained experiment results with above-mentioned models on two datasets. The overall evaluation is as follows.

- In general, the accuracy of MLNN and ABFO-MLNN models did not bring performance well as the comparison with the rest.
- With the EU dataset, our proposed OCRO-MLNN performances has the best with (RMSE, MAE) = (15.535, 11.058) and (15.381, 10.918) for $k = 2$ and $k = 5$, respectively.
- LSTM get the best results with index of R^2, which are 0.9958 and 0.9959 for $k = 2$ and $k = 5$, respectively. In comparison with CRO-MLNN ($R^2 = 0.9954$ for $k = 2$ and 0.9950 for $k = 5$) and OCRO-MLNN ($R^2 = 0.9955$ for $k = 2$ and 0.9956 for $k = 5$), LSTM is just slightly better.
- With the WC dataset, for all four metrics R^2, MAE, MAPE, and SMAPE with sliding window $k = 5$, the obtained results with OCRO-MLNN are remarkable when getting the best accuracies that marked in bold numbers in the table.

Table 2 shows that OCRO-MLNN not only improves the original CRO-MLNN but also gets better accuracy as compared with LSTM and others. In which the bold numbers indicate the best gained

Table 2 RMSE, MAPE and R^2 comparison (sliding window $k = 2$ and $k = 5$; univariate datasets).

Data	Model	R^2		RMSE		MAE		MAPE		SMAPE	
		$k = 2$	$k = 5$	$k = 2$	$k = 5$	$k = 2$	$k = 5$	$k = 2$	$k = 5$	$k = 2$	$k = 5$
EU	MLNN	0.9931	0.9936	20.484	19.599	15.826	15.359	4.357	4.362	4.268	4.260
	GA-MLNN	0.9953	0.9946	15.807	16.952	11.452	12.225	3.078	3.267	3.062	3.281
	ABFO-MLNN	0.9942	0.9917	17.552	20.959	13.338	15.536	3.921	4.216	3.851	4.308
	PSO-MLNN	0.9954	0.9955	15.712	15.474	11.347	11.139	3.072	3.002	3.061	3.002
	CRO-MLNN	0.9954	0.9950	15.586	16.364	11.100	11.840	2.939	3.216	2.940	3.207
	RNN	0.9956	0.9958	16.263	16.866	11.624	11.320	2.954	2.894	2.977	2.911
	LSTM	0.9958	0.9959	15.857	15.786	11.358	11.497	2.899	2.973	2.902	2.956
	OCRO-MLNN	0.9955	0.9956	**15.535**	**15.381**	**11.058**	**10.918**	2.933	2.903	2.930	**2.900**
WC	MLNN	0.9805	0.9565	29.562	44.164	13.287	20.083	5.708	8.892	5.884	8.737
	GA-MLNN	0.9912	0.9834	19.876	27.253	8.173	10.758	4.806	4.882	4.655	4.822
	ABFO-MLNN	0.9800	0.9763	27.487	29.971	10.223	10.264	4.555	4.690	4.588	4.675
	PSO-MLNN	0.9736	0.9918	34.385	19.176	12.244	8.447	4.823	4.633	4.900	4.637
	CRO-MLNN	0.9915	0.9910	19.475	20.111	8.535	8.603	4.439	4.367	4.429	4.411
	RNN	0.9780	0.9854	28.833	23.468	11.440	10.442	4.644	4.388	4.677	4.347
	LSTM	0.9880	0.9892	21.300	20.195	8.776	8.519	4.455	4.319	4.435	4.328
	OCRO-MLNN	0.9856	**0.9937**	25.388	**16.808**	10.128	**7.290**	**4.383**	4.340	**4.348**	**4.320**

test results of OCRO-MLNN with different measurements. Figure 3 and Figure 4 express the prediction results to illustrate the gained data.

Runtime comparison. Table 3 shows runtime measurements of experiments. Note that epoch settings are described in detail in 5.1.3. There are some observations, which can be made as follows.

- RNN and LSTM have a quite high runtime. For instance, with $k = 2$, RNN has $t_s = 2158.05$ seconds and LSTM's t_s is 1891.72 seconds. This issues causing by long epoch runtime. Concretely, with $k = 2$, RNN's t_e is 2.1575 seconds and LSTM's t_e is 1.8913 seconds. It is well-known that the models built from blocks have a complicated structure, which is hard to optimize. Besides, the time cost also depends on the hardware configuration of machines, which the models run on.
- Traditional MLNN's have significant small training time per epoch in comparison with others model. The disadvantage of MLNNs is that it converges slowly, requiring many epoch to train. Therefore the training time will be more and cause a slow effect on overall.
- OCRO-MLNN and CRO-MLNN provide good runtime performance to complete all work with encouraging results. The reason is their short runtime per epoch and quick convergence.

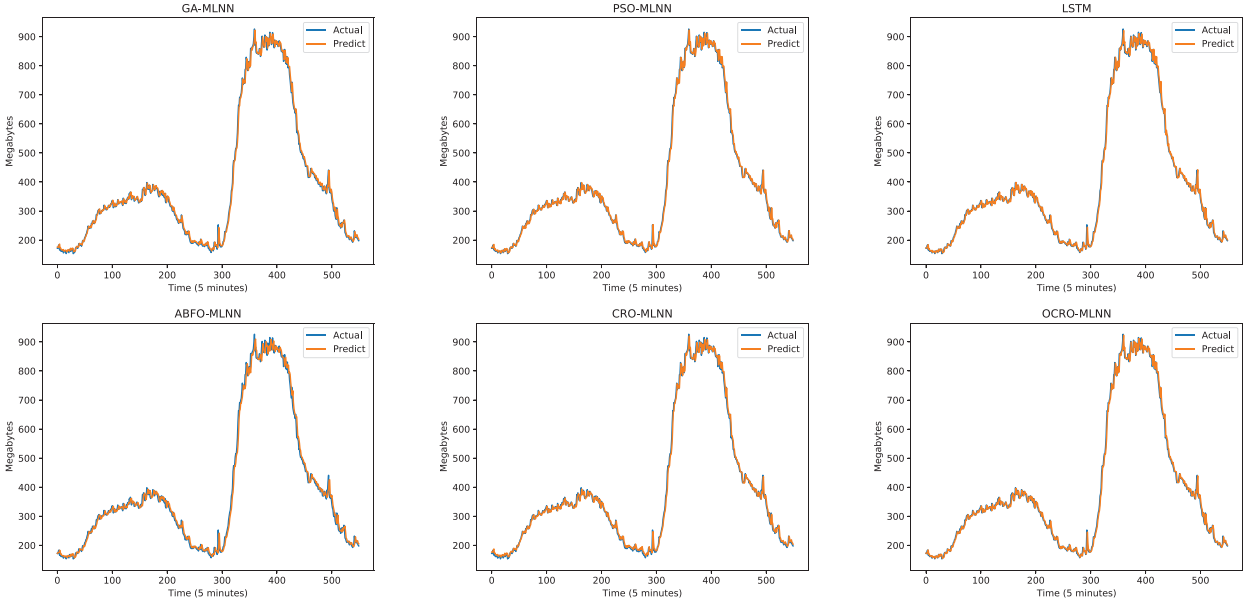

Figure 3 Comparison of opposition-based coral reefs optimization (OCRO)-multi-layer neural network (MLNN) prediction with other models (sliding window = 2; EU dataset).

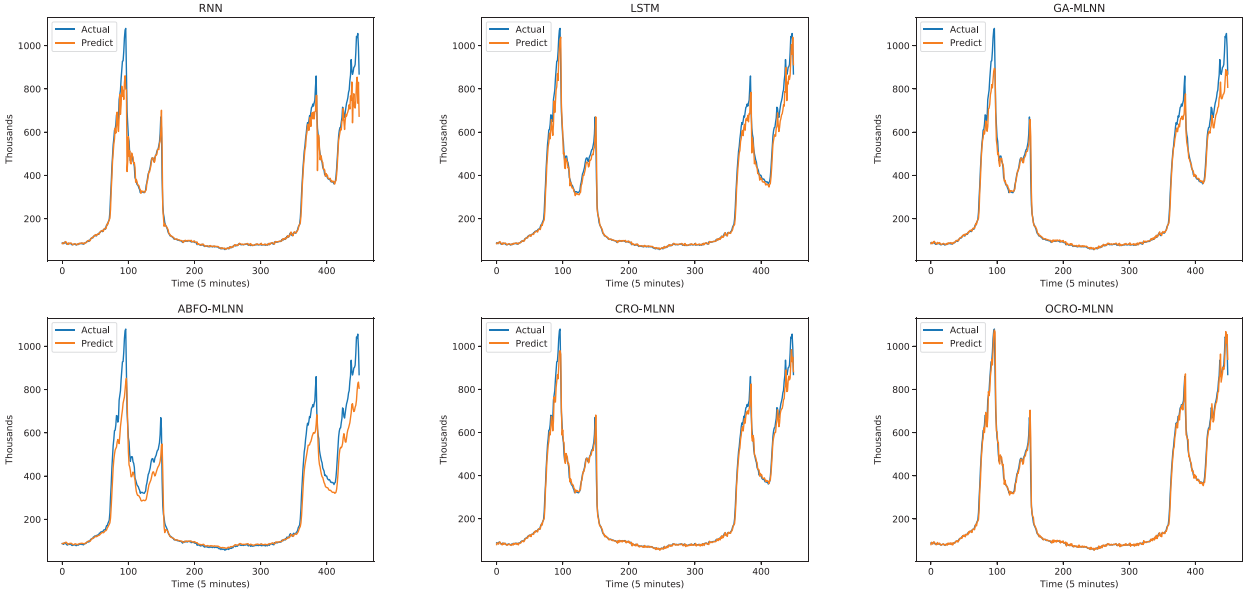

Figure 4 Comparison of opposition-based coral reefs optimization (OCRO)-multi-layer neural network (MLNN) prediction with other models (sliding window = 5; world cup (WC) dataset).

- EU traffic dataset: OCRO-MLNN's runtime $t_s = 353.79$s ($k = 2$) in comparison with GA-MLNN's $t_s = 657.77$s and PSO-MLNN's $t_s = 642.73$s.
- WC dataset: OCRO-MLNN's runtime $t_s = 242.33$s ($k = 5$) in comparison with CRO-MLNN's $t_s = 256.59$s and MLNN's $t_s = 459.92$s

Model stability. Table 4 presents the comparison of OCRO-MLNN with other models on RMSE (average statistics after 15 run times). There are several short evaluations and remarks as follows.

- MLNN model obtains the worst results with very high standard deviation (*std*) and coefficient of variation (*cv*). Concretely, experiments with WC dataset for $k = 2$, MLNN has RMSE in range from 18.771 up to 67.5236, $std = 12.6737$ and $cv = 0.3836$.
- The proposed OCRO-MLNN obtains in experiments RMSE values in range from 16.6236 to 24.7484, $std = 1.8681$ and $cv = 0.0932$). The reason of such results are disadvantages of back-propagation algorithm as analyzed above in Section 2.
- ABFO-MLNN and RNN model also gain quite bad results in stability when comparing to LSTM, CRO-MLNN, and OCRO-MLNN. For more detail, with the EU dataset, sliding window $k = 5$, RNN's $std = 2.2523$, ABFO-MLNN's $std = 2.6538$, LSTM's $std = 0.8462$, CRO-MLNN's $std = 0.5448$, and OCRO-MLNN's $std = 0.2194$ (i.e. a quarter of LSTM's *std*), which is the most stable one in comparison with the rest.

Figure 9 visualizes RMSE range of 15 run times averages with tested models using box-and-whisker plots. While the intermittent lines in the middle of boxes represent the mean values, the upper and lower boundaries of the boxes represent the upper and lower quantiles of the distributions.

5.2.2. Multivariate data

Prediction accuracy. Table 5 shows the gained experiments results with all tested models on multivariate data in the aspect of accuracy. The evaluations can be made as follows.

- MLNN model achieves the worst results in this test. The low accuracy problem of this model caused by gradient descent when searching for an optimal solution in a complex dimension.
- OCRO-MLNN model obtains the best results indicating high accuracies in both multivariates tested data types (shown by Table 5).
- In comparison OCRO-MLNN with LSTM, both models gain the top results alternately with small difference. Concretely, the outcomes are as follows.

Table 3 Runtime comparison (sliding window $k = 2$ and $k = 5$; univariate datasets).

Data	Model	$k = 2$			$k = 5$		
		t_e	t_p	t_s	t_e	t_p	t_s
EU	MLNN	0.1338	0.1692	669.42	0.1384	0.1273	692.22
	GA-MLNN	0.9397	0.0011	657.77	0.9202	0.0014	644.13
	ABFO-MLNN	0.9487	0.0011	664.11	0.9772	0.1112	684.04
	PSO-MLNN	0.9182	0.0015	642.73	0.9089	0.0014	636.25
	CRO-MLNN	0.5091	0.0014	356.36	0.5145	0.0014	360.18
	RNN	2.1575	0.5054	2158.05	3.8974	0.7084	3898.13
	LSTM	1.8913	0.4376	1891.72	3.8137	0.6273	3814.30
	OCRO-MLNN	0.5054	0.0010	**353.79**	0.4925	0.0012	**344.78**
WC	MLNN	0.0895	0.2358	448.04	0.0919	0.2466	459.92
	GA-MLNN	0.6449	0.0012	451.43	0.6580	0.0018	460.58
	ABFO-MLNN	0.6701	0.0009	469.10	0.6899	0.0008	482.97
	PSO-MLNN	0.6241	0.0008	436.86	0.6382	0.0008	446.76
	CRO-MLNN	0.3496	0.0008	244.74	0.3665	0.0009	256.59
	RNN	0.9773	1.1465	978.41	1.9293	1.2888	1930.61
	LSTM	0.8450	0.8934	845.93	1.9264	1.2296	1927.63
	OCRO-MLNN	0.3462	0.0008	**242.33**	0.3537	0.0012	**242.33**

Table 4 RMSE comparison (average of 15 times; univariate dataset)

Data	Model	$k = 2$					$k = 5$				
		min	*max*	*mean*	*std*	*cv*	*min*	*max*	*mean*	*std*	*cv*
EU	MLNN	15.8611	30.4864	19.2223	4.1926	0.2181	16.2603	31.9792	25.6531	4.0866	0.1593
	RNN	16.0049	20.984	17.7525	1.8485	0.1041	16.0111	21.908	17.9194	2.2523	0.1257
	LSTM	15.585	16.5124	16.0292	0.2314	0.0144	15.7024	18.6302	16.4398	0.8462	0.0515
	GA-MLNN	15.5587	19.7204	17.152	1.3099	0.0764	15.6322	20.7341	17.5186	1.3395	0.0765
	PSO-MLNN	15.5566	16.2584	15.7078	0.1961	0.0125	15.3195	17.8479	16.1036	0.7924	0.0492
	ABFO-MLNN	15.9053	19.5867	17.1062	1.1966	0.07	16.4715	25.7909	21.5049	2.6538	0.1234
	CRO-MLNN	15.5244	16.4715	15.822	0.278	0.0176	15.5357	17.8479	16.1219	0.5448	0.0338
	OCRO-MLNN	15.5151	15.5667	15.5441	0.0142	0.0009	15.3013	15.996	15.5697	0.2194	0.0141
WC	MLNN	18.771	67.5236	33.0351	12.6737	0.3836	19.5462	61.4193	38.9827	12.5872	0.3229
	RNN	20.488	35.0887	27.4246	3.8517	0.1404	18.0851	30.6916	23.1305	3.3629	0.1454
	LSTM	17.6447	31.5707	24.6919	4.5824	0.1856	18.035	27.9699	22.1468	3.5013	0.1581
	GA-MLNN	18.0851	30.6916	22.7368	3.2264	0.1419	18.6348	35.3217	23.3347	5.1492	0.2207
	PSO-MLNN	16.7875	34.5682	26.4276	5.2702	0.1994	16.8053	29.5606	20.3284	3.8017	0.187
	ABFO-MLNN	16.6236	38.7891	23.2257	6.8395	0.2945	19.2615	41.0411	28.1293	6.8486	0.2435
	CRO-MLNN	16.7806	28.7189	21.4259	3.4962	0.1632	17.5025	33.8349	23.4025	4.5302	0.1936
	OCRO-MLNN	16.6236	24.7484	20.0339	1.8681	0.0932	16.9945	27.0219	20.5392	2.5978	0.1265

- For CPU metric: sliding window $k = 5$, OCRO-MLNN's (MAE, MAPE) = (0.294, 35.297), and LSTM's (MAE, MAPE) = (0.298, 35.523).
- For random-access memory (RAM) metric: sliding window $k = 5$, OCRO-MLNN's (R^2, MAE) = (0.6875, 0.021), and LSTM's (R^2, MAE) = (0.6924, 0.018).

These experiments presented above indicate that with multivariate data, our OCRO-MLNN model performs well as compared with others. Figures 5 and 6 are used to show predictive results of different models gained via the experiments.

Runtime comparison. Table 6 presents outcomes of runtime experiments with different models. It can be seen that the obtained results of CRO-MLNN and OCRO-MLNN, which outperform other models on system runtime similarly as with univariate datasets. Concretely, for CPU metric with $k = 5$: CRO-MLNN and OCRO-MLNN runtime are 252.78s and 240.21s, respectively. LSTM consumes approximately 5 times more (i.e. 1324.65s) to complete all training cycles. The difference is significant when the time series dataset for prediction is large-scale. Figures 7 and 8 visualizes system time of our OCRO-MLNN model (green color) in comparison with the rest against univariate and multivariate data respectively.

Model stability. Table 7 gives summary statistics of RMSE error with the tested models after 15 run times. The evaluations gained through these experiments with multivariate data include:

- MLNN and RNN are quite unstable (as usually) due to back-propagation dependence on the initial weights. LSTM has smaller fluctuation of RMSE's error at the cost of its complex structure. Specifically, for CPU metrics, $k = 2$, standard deviation (*std*) of MLNN, RNN, and LSTM model are $std = 0.0615$, 0.0353, and 0.0171, respectively.
- The proposed approach is applying meta-heuristic OCRO to optimization problems gives more stability than gradient

Table 5 RMSE, MAPE and R^2 comparison (sliding window $k = 2$ and $k = 5$; multivariate dataset; the bold numbers are the best achieved results).

Data	Model	R^2		RMSE		MAE		MAPE		SMAPE	
		$k = 2$	$k = 5$	$k = 2$	$k = 5$	$k = 2$	$k = 5$	$k = 2$	$k = 5$	$k = 2$	$k = 5$
CPU	MLNN	0.0828	0.0742	0.529	0.504	0.363	0.344	36.089	39.629	42.218	34.075
	GA-MLNN	0.1012	0.1339	0.465	0.445	0.302	0.316	36.822	36.193	31.347	29.858
	ABFO-MLNN	0. 0861	0.0779	0.494	0.500	0.324	0.315	39.914	35.130	31.839	31.646
	PSO-MLNN	0.1312	0.1370	0.453	0.452	0.307	0.303	36.550	35.859	30.705	30.393
	CRO-MLNN	0.1461	0.1523	0.449	0.448	0.304	0.300	36.461	35.687	30.512	30.251
	RNN	0.1332	0.1568	0.485	0.478	0.315	0.305	38.648	35.841	31.280	30.558
	LSTM	0.1664	0.1839	0.476	0.452	0.302	0.294	36.385	35.297	30.227	31.287
	OCRO-MLNN	**0.1812**	**0.1922**	**0.440**	**0.437**	**0.294**	0.298	**35.403**	35.523	**29.710**	29.977
RAM	MLNN	0.5411	0.4652	0.038	0.068	0.027	0.058	12.008	27.195	11.547	23.307
	GA-MLNN	0.6663	0.6426	0.032	0.033	0.021	0.021	9.923	9.613	9.745	9.514
	ABFO-MLNN	0.6860	0.5241	0.033	0.043	0.022	0.034	9.621	15.962	9.535	14.616
	PSO-MLNN	0.6758	0.6560	0.032	0.033	0.020	0.021	8.883	9.357	8.880	9.317
	CRO-MLNN	0.6572	0.6455	0.033	0.033	0.021	0.022	9.358	9.772	9.365	9.811
	RNN	0.6927	0.6684	0.034	0.034	0.019	0.022	9.521	9.756	9.577	9.974
	LSTM	0.7069	0.6924	0.033	0.034	0.021	0.018	9.203	9.243	9.135	9.329
	OCRO-MLNN	**0.7092**	0.6875	**0.031**	**0.032**	**0.018**	0.021	**8.792**	**9.091**	**8.789**	**9.165**

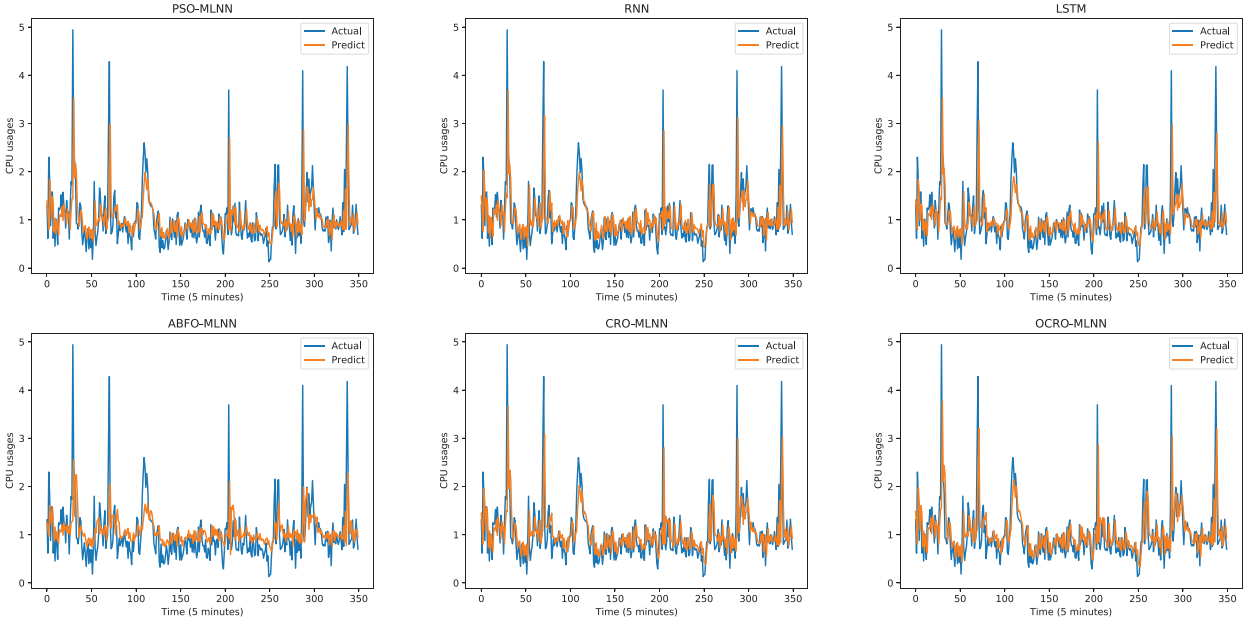

Figure 5 Comparison of opposition-based coral reefs optimization (OCRO)-multi-layer neural network (MLNN) prediction with other models (sliding window = 2; multivariate data - CPU).

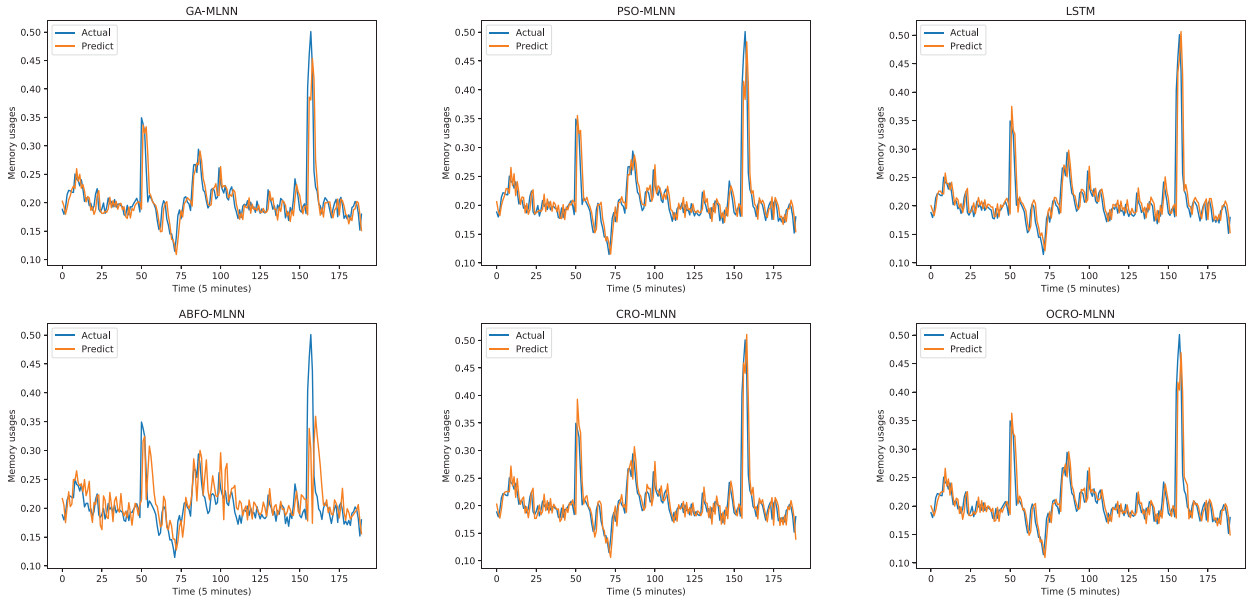

Figure 6 Comparison of opposition-based coral reefs optimization (OCRO)-multi-layer neural network (MLNN) prediction with other models (sliding window = 5; multivariate data - RAM).

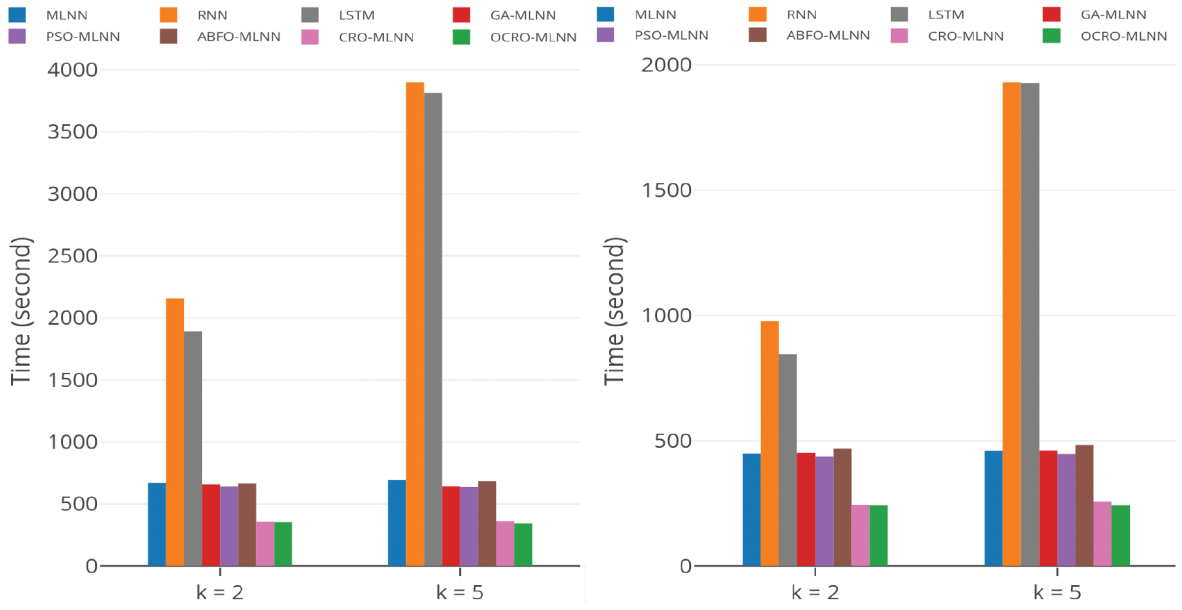

Figure 7 Runtime comparison: opposition-based coral reefs optimization (OCRO)-multi-layer neural network (MLNN) (green), sliding window = 5; univariate data: EU (left), WC (right).

descent approach. The reason is that OCRO is considered as an efficient and powerful searching algorithm to help it jump out the local optimal and search for the best solution. So that when combining OCRO with traditional MLNN, the achieved model can work well under different conditions.

- Compared to CPU metrics and $k = 5$, LSTM's $std = 0.0172$ and OCRO-MLNN's $std = 0.0058$ (i.e. one third of LSTM's std value). Meanwhile, compared to RAM metrics, $k = 2$, LSTM's $cv = 0.0293$ and OCRO-MLNN's $cv = 0.0149$ (i.e. a half of LSTM's cv value).

Figure 10 expresses average RMSE after 15 run times of all models (the same way as univariate dataset).

5.3. Practical Usage of the Proposed Forecasting Approach

In the distributed environment, one of the core issues is how to optimize scheduling resources under heavy and changeable computation loads that are made by complex requirements coming from applications. With a set of available resources (e.g. servers,

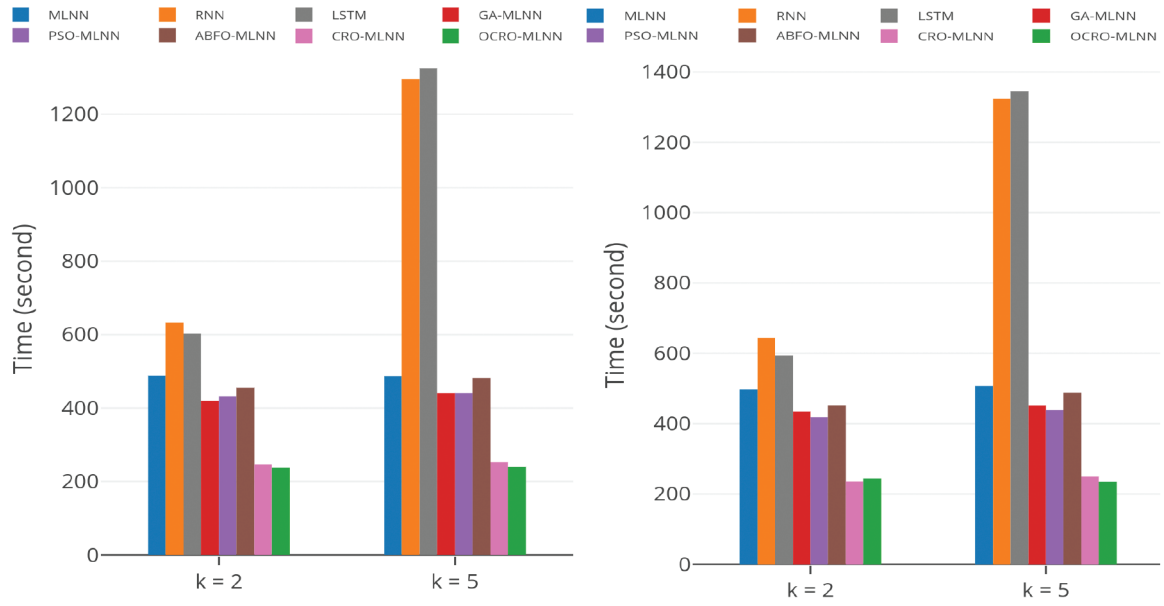

Figure 8 Runtime comparison: opposition-based coral reefs optimization (OCRO)-multi-layer neural network (MLNN) (green), sliding window = 5; multivariate data, CPU (left), RAM (right).

Table 6 Runtime comparison (sliding window $k = 2$ and $k = 5$, multivariate dataset; the bold numbers are the best achieved results).

Data	Model	$k = 2$			$k = 5$		
		t_e	t_p	t_s	t_e	t_p	t_s
CPU	MLNN	0.0973	0.5281	487.77	0.0971	0.5110	486.71
	GA-MLNN	0.5999	0.0008	419.94	0.6300	0.0008	441.02
	ABFO-MLNN	0.6502	0.0007	455.11	0.6891	0.0011	482.37
	PSO-MLNN	0.6172	0.0010	432.03	0.6302	0.0011	441.13
	CRO-MLNN	0.3514	0.0008	245.96	0.3611	0.0041	252.78
	RNN	0.6306	2.3131	632.91	1.2935	2.4592	1295.96
	LSTM	0.6005	2.2273	602.73	1.3223	2.3590	1324.65
	OCRO-MLNN	0.3404	0.0009	**238.28**	0.3431	0.0009	**240.21**
RAM	MLNN	0.0993	0.5809	498.00	0.1012	0.5904	507.65
	GA-MLNN	0.6209	0.0007	434.62	0.6455	0.0008	451.85
	ABFO-MLNN	0.6462	0.0009	452.31	0.6976	0.0012	488.32
	PSO-MLNN	0.5976	0.0010	418.30	0.6262	0.0011	438.32
	CRO-MLNN	0.3359	0.0009	235.13	0.3574	0.0008	250.18
	RNN	0.6406	2.7545	643.35	1.3208	2.7987	1323.56
	LSTM	0.5912	2.4918	593.65	1.3428	2.7694	1345.54
	OCRO-MLNN	0.3483	0.0009	243.80	0.3351	0.0009	**234.55**

Table 7 RMSE comparison (average of 15 times; multivariate dataset).

Data	Model	$k = 2$					$k = 2$				
		min	*max*	*mean*	*std*	*cv*	*min*	*max*	*mean*	*std*	*cv*
CPU	MLNN	0.4563	0.6579	0.5232	0.0615	0.1175	0.4659	0.7135	0.5896	0.0902	0.153
	RNN	0.4461	0.5918	0.5059	0.0353	0.0698	0.4489	0.5503	0.465	0.0279	0.0599
	LSTM	0.4489	0.4929	0.4706	0.0171	0.0364	0.4499	0.4929	0.4732	0.0172	0.0364
	GA-MLNN	0.4392	0.4631	0.4523	0.0058	0.0129	0.4362	0.4863	0.4557	0.0148	0.0324
	PSO-MLNN	0.4421	0.4605	0.4494	0.0055	0.0121	0.4444	0.4627	0.4537	0.0064	0.0142
	ABFO-MLNN	0.4423	0.5788	0.5119	0.0424	0.0829	0.4505	0.6528	0.5302	0.0596	0.1124
	CRO-MLNN	0.4386	0.4545	0.4485	0.004	0.009	0.4428	0.4634	0.4503	0.0051	0.0114
	OCRO-MLNN	0.4392	0.4523	0.4474	0.0028	0.0064	0.4396	0.4571	0.4492	0.0058	0.0128
RAM	MLNN	0.0334	0.0676	0.0436	0.0103	0.2363	0.0326	0.0676	0.0528	0.0091	0.173
	RMM	0.0323	0.0446	0.0372	0.0037	0.1008	0.0323	0.0414	0.0363	0.0031	0.0868
	LSTM	0.0318	0.0346	0.0327	0.001	0.0293	0.0318	0.0398	0.0331	0.0019	0.0583
	GA-MLNN	0.031	0.0331	0.0321	0.0005	0.0156	0.032	0.0363	0.0336	0.0013	0.0389
	PSO-MLNN	0.0309	0.0342	0.0322	0.0008	0.0259	0.0314	0.0348	0.0325	0.0009	0.0291
	ABFO-MLNN	0.0322	0.0541	0.0414	0.0064	0.1549	0.0313	0.0537	0.0395	0.0082	0.2065
	CRO-MLNN	0.0313	0.033	0.0322	0.0005	0.0154	0.0312	0.0335	0.0324	0.0007	0.0222
	OCRO-MLNN	0.031	0.0328	0.0322	0.0005	0.0149	0.0312	0.0336	0.0324	0.0007	0.0218

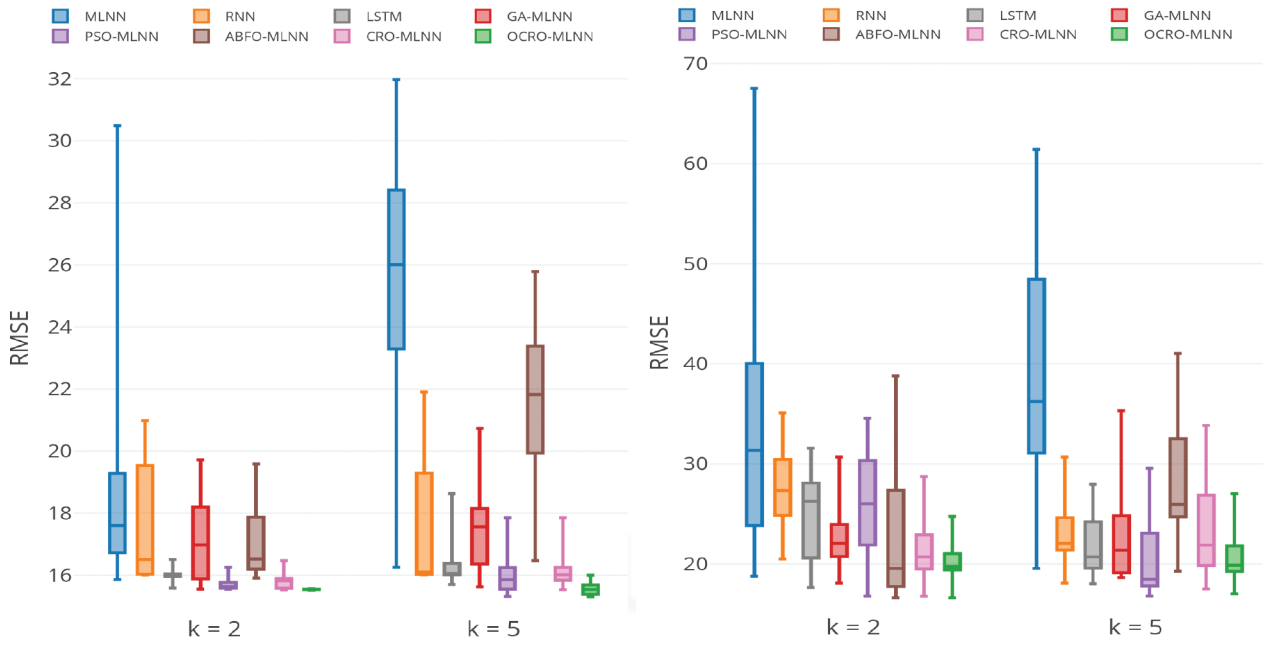

Figure 9 | RMSE comparison: average of 15 times; univariate data: EU (left), world cup (WC) (right).

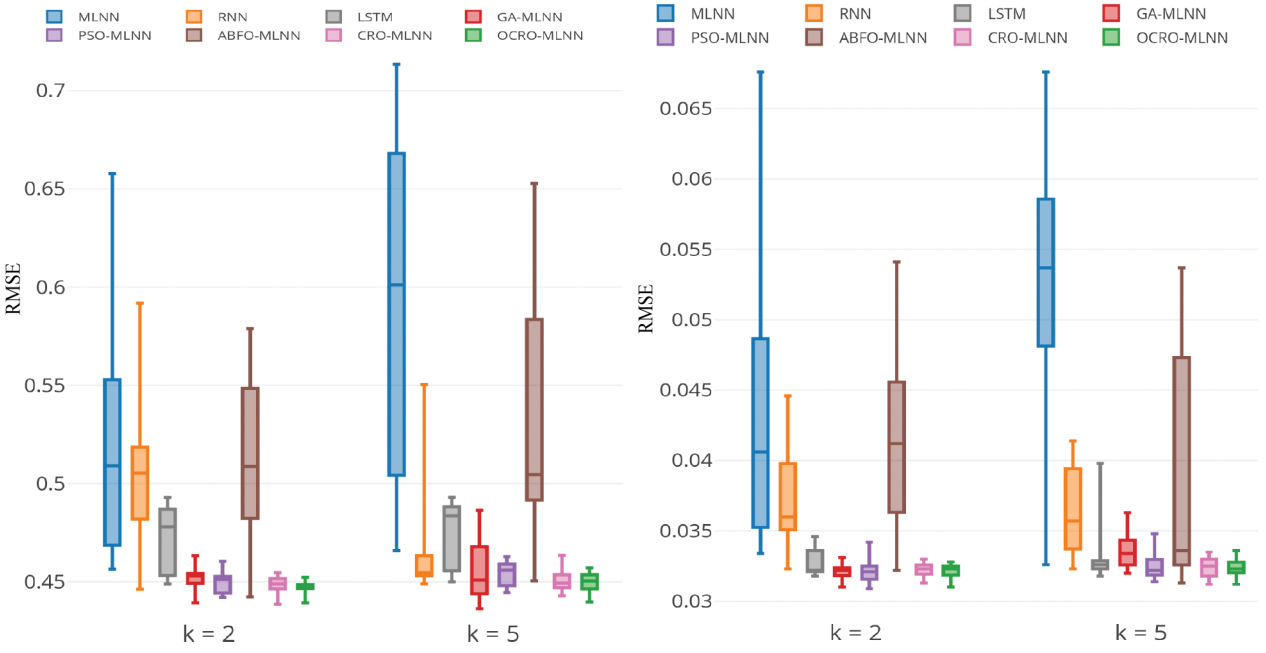

Figure 10 | Root mean square error (RMSE) comparison: average of 15 times; multivariate data: central processing unit (CPU) (left), RAM (right).

networks, storages), a scheduler (74) must have the capability to make scale in/out decisions to provide computation powers for those applications. Although the traditional technique for the scaling problem with predefined resource usage thresholds is employed widely in fact, this method has the main disadvantage in operation and management of distributed applications: there is always latency between the decision-making moment and its efficiency. This leads to the resources offered to applications does not match with demand in real-time. To deal with the issue, scheduler deployed in distributed systems requires precise estimates resource consumptions. To achieve that capability, the scheduler must have a certain component to predict expected performances of infrastructure systems. The forecast module also is designed in the manner of minimizing costs such as simple prediction model, stability, and runtime constraint, but keeping the good accuracy in order to well operate in practice.

Within our work in this direction, the prediction-based autoscaling systems for cloud resource management was considered in [44] taking into account quality of service (QoS) defined in service-layer agreements (SLA). The aim is to reduce cloud resource over-provisioning that causes wasted energy consumption as well

as e-infrastructure production cost. The cloud resources provision under SLA conditions is extended in [49] with experimental results to ensure QoS using the number of provided virtual machines (VMs) instead of cloud monitoring metrics to make scaling decisions. The work presented in this paper continues in improvements of the intelligent module functionalities. It is oriented not only for resource management but also for network traffic monitoring.

With the application direction presented above, in this work, through experiments, we proved our proposed OCRO-MLNN model can be applied to schedulers in distributed systems to resolve the scaling issue. Moreover, the model also brings efficiency as desired as compared with other modern learning techniques. Indeed, the tested data was collected from the computing infrastructures (i.e. Google's cluster workload and Internet traffic) and distributed applications (Football world cup's website connection) is used to carry out the model evaluation experiments.

6. CONCLUSION AND FUTURE WORKS

This work presented an approach and technique for prediction problem with time series data. The approach is based on the idea to build a stable model with simple structure, low run-time training, but still brings good accuracy as compared with other modern models. In this way, we created a novel algorithm called opposition-based coral reefs optimization (OCRO). Specifically, the proposed algorithm is the combination of CRO using OBL. Thus, we used OCRO together with the MLNN to deal with the forecast problem in the nonlinear time series data. By the improvement, our OCRO-MLNN reducing complexity of the model due to faster convergence in comparison with the back-propagation technique.

We are interested in applying the created OCRO-MLNN to distributed systems in order to increase the effectiveness as well as efficiency in allocating resources for applications in the manner of proactiveness. Due to its simplification, fast training, stabilization, and forecast performance, the proposed model has the feasibility in applying to the resource schedulers. With the application direction of using in distributed systems, we already tested the proposed OCRO-MLNN with real dataset gathered from those systems, including Google trace cluster data, Internet traffic, and popular website's connections. The achieved results through experiments shown that our OCRO-MLNN has advantages in all the proposed criteria listed above as compared with other state-of-the-art models in both univariate and multivariate data. This proves OCRO-MLNN's feasibility in applying to real systems (e.g. clouds, decentralized ledgers applications etc.) in practice.

The study made contributions in four areas. First, we built the OCRO technique by using OBL mechanism for CRO. The improvement helps searching process and avoid the local minimum of traditional CRO. Second, a novel forecast model is introduced with the combination of our OCRO technique with MLNN. In which OCRO is used to replace back-propagation. Third, in terms of evaluation, we conducted comparisons among different prediction models and OCRO-MLNN for time series data. Four, with three real datasets collected from well-known systems, we tested the performance of the proposed model as well as other popular methods above. The tests were carried out under three factors, including accuracy, stability, and runtime.

In the future, there are two routes that we plan to do. The first route is to design and develop a resource allocation scheduler which we would like to employ to decentralized systems. In our project funded by VINIF, we have deployed an auto-scaler for controlling computation resources for blockchain nodes of the V-Chain platform using ORCO-MLNN prediction model proposed in this work. The goal of this use is to optimize cloud resource consumption as well as monitor network bandwith of the system. The scheduler thus must process multivariate data with diverse metrics as usual scaling requirement. Besides, a resource allocation decision module also is demanded for the scheduler. The second route is to use and improve several other nature-inspired algorithms, then combine with machine techniques like neural network to serve forecast problem in proactive auto-scalers of distributed systems.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHORS' CONTRIBUTIONS

T. Nguyen and Tu Nguyen designed the program; T. Nguyen, Tu Nguyen, and B.M. Nguyen tested; T. Nguyen, Tu Nguyen, and B.M. Nguyen prepared and wrote the original draft; B.M. Nguyen and G. Nguyen reviewed and edited; and B.M. Nguyen conducted the project administration.

ACKNOWLEDGMENT

Research is supported by Vingroup Innovation Foundation (VINIF) in project code VINIF.2019.DA07.

REFERENCES

[1] G.E. Box, G.M. Jenkins, G.C. Reinsel, G.M. Ljung, Time Series Analysis: Forecasting and Control, John Wiley and Sons, Hoboken, 2015.
[2] R. Nau, ARIMA models for time series forecasting, in: Statistical Forecasting: Notes on Regression and Time Series Analysis, Duke University, Durham, 2017.
[3] G. Zhang, B.E. Patuwo, and M.Y. Hu, Forecasting with artificial neural networks:: The state of the art, Int. J. Forecast. 14 (1998), 35–62.
[4] D.E. Rumelhart, G.E. Hinton, and R. J. Williams, Learning representations by back-propagating errors, Nature, 323 (1986), 533.
[5] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997), 1735–1780.
[6] X.S. Yang. Metaheuristic optimization, Scholarpedia. 6 (2011), 11472.
[7] E. McKenzie, General exponential smoothing and the equivalent arma process, J. Forecast. 3 (1984), 333–344.
[8] A.C. Petricua, S. Stancu, A. Tindeche, Limitation of arima models in financial and monetary economics, Theor. Appl. Econ. 23 (2016), 19–42.
[9] Y. Kajitani, K.W. Hipel, A.I. McLeod, Forecasting nonlinear time series with feed-forward neural networks: a case study of canadian lynx data, J. Forecast. 24 (2005), 105–117.

[10] D. Srinivasan, A. Liew, and C. Chang, A neural network short-term load forecaster, Electr. Pow. Syst. Res. 28 (1994), 227–234.
[11] I. Kaastra, M. Boyd, Designing a neural network for forecasting financial and economic time series, Neurocomputing. 10 (1996), 215–236.
[12] H. Raman, N. Sunilkumar, Multivariate modelling of water resources time series using artificial neural networks, Hydrological Sci. J. 40 (1995), 145–163.
[13] G. Zhang and M.Y. Hu, Neural network forecasting of the british pound/us dollar exchange rate, Omega. 26 (1998) 495–506.
[14] D. Enke, S. Thawornwong, The use of data mining and neural networks for forecasting stock market returns, Expert Syst. Appl. 29 (2005), 927–940.
[15] F.O. Hocaoglu, O.N. Gerek, M. Kurban, Hourly solar radiation forecasting using optimal coefficient 2-d linear filters and feed-forward neural networks, Solar Energy. 82 (2008), 714–726.
[16] J.T. Connor, R.D. Martin, L.E. Atlas, Recurrent neural networks and robust time series prediction, IEEE Trans. Neural Netw. 5 (1994), 240–254.
[17] M. Han, J. Xi, S. Xu, F.L. Yin, Prediction of chaotic time series based on the recurrent predictor neural network, IEEE Trans. Signal Process. 52 (2004), 3409–3416.
[18] A. Petrosian, D. Prokhorov, R. Homan, R. Dasheiff, D. Wunsch II, Recurrent neural network based prediction of epileptic seizures in intra-and extracranial eeg, Neurocomputing. 30 (2000), 201–218.
[19] G. Pollastri, D. Przybylski, B. Rost, P. Baldi, Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles, Proteins Struct. Funct. Bioinf. 47 (2002), 228–235.
[20] E.W. Saad, D.V. Prokhorov, D.C. Wunsch, Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks, IEEE Trans. Neural Netw. 9 (1998), 1456–1470.
[21] R. Gencay, T. Liu, Nonlinear modelling and prediction with feedforward and recurrent networks, Phys. D Nonlin. Phenomena. 108 (1997), 119–134.
[22] K. Kalaitzakis, G. Stavrakakis, E. Anagnostakis, Short-term load forecasting based on artificial neural networks parallel implementation, Electric Power Syst. Res. 63 (2002), 185–196.
[23] S. Saha and G. Raghava, Prediction of continuous b-cell epitopes in an antigen using recurrent neural network, Proteins. 65 (2006), 40–48.
[24] A. Azzouni, G. Pujolle, A long short-term memory recurrent neural network framework for network traffic matrix prediction, arXiv preprint arXiv:1705.05690, 2017.
[25] X. Ma, Z. Tao, Y. Wang, H. Yu, Y. Wang, Long short-term memory neural network for traffic speed prediction using remote microwave sensor data, Transp. Res. Part C Emerg. Technol. 54 (2015), 187–197.
[26] Y. Tian and L.Pan, Predicting short-term traffic flow by long short-term memory recurrent neural network, in 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity), IEEE, 2015, 153–158.
[27] N. Tran, T. Nguyen, B.M. Nguyen, G. Nguyen, A multivariate fuzzy time series resource forecast model for clouds using lstm and data correlation analysis, Procedia Comput. Sci. 126 (2018), 636–645.
[28] Z. Zhao, W. Chen, X. Wu, P.C. Chen, J. Liu, Lstm network: a deep learning approach for short-term traffic forecast, IET Intell. Transp. Syst. 11 (2017), 68–75.
[29] T. Hollis, A. Viscardi, S.E. Yi, A comparison of lstms and attention mechanisms for forecasting financial time series, arXiv preprint arXiv:1812.07699, 2018.
[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in Advances in neural information processing systems, 2017, pp. 5998–6008,
[31] S. Bai, J.Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, arXiv preprint arXiv:1803.01271, 2018.
[32] S.Y. Shih, F.K. Sun, H.Y. Lee, Temporal pattern attention for multivariate time series forecasting, arXiv preprint arXiv:1809.04206, 2018.
[33] W. Vorhies. Temporal convolutional nets (tcns) take over from rnns for nlp predictions, 5 2018.
[34] E. Culurciello, Computation and memory bandwidth in deep neural networks, Medium. 5 (2017), 1–4.
[35] E. Culurciello, V. Gokhale, A. Zaidy, A. Chang, Hardware Accelerator for Convolutional Neural Networks and Method of Operation Thereof, Google Patents, 2018.
[36] G. Nguyen, B.M. Nguyen, D. Tran, L. Hluchy, A heuristics approach to mine behavioural data logs in mobile malware detection system, Data Knowl. Eng. 115 (2018), 129–151.
[37] J.H. Holland, Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, MIT Press, Cambridge, 1992.
[38] D.J. Montana, L. Davis. Training feedforward neural networks using genetic algorithms, in International Joint Conference on Artificial Intelligence (IJCAI), Detroit, 1989, vol. 89, pp. 762–767.
[39] D. Whitley, T. Starkweather, C. Bogart, Genetic algorithms and neural networks: Optimizing connections and connectivity, Parallel Comput. 14 (1990), 347–361.
[40] D.F. Cook, C.T. Ragsdale, R. Major, Combining a neural network with a genetic algorithm for process parameter optimization, Eng. Appl. Artif. Intell. 13 (2000), 391–396.
[41] H.J. Kim, K.S. Shin, A hybrid approach based on neural networks and genetic algorithms for detecting temporal patterns in stock markets, Appl. Soft Comput. 7 (2007), 569–576.
[42] L. Magnier, F. Haghighat, Multiobjective optimization of building design using trnsys simulations, genetic algorithm, and artificial neural network, Building Environ. 45 (2010), 739–746.
[43] A.G. Karegowda, A. Manjunath, M. Jayaram, Application of genetic algorithm optimized neural network connection weights for medical diagnosis of pima indians diabetes, Int. J. Soft Comput. 2 (2011), 15–23.
[44] T. Nguyen, N. Tran, B.M. Nguyen, G. Nguyen, A resource usage prediction system using functional-link and genetic algorithm neural network for multivariate cloud metrics, in The 11th IEEE International Conference on Service-Oriented Computing and Applications (SOCA'2018), IEEE, Paris, 2018, pp. 49–56.
[45] S. Ramesh, A. Krishnan, Modified genetic algorithm based load frequency controller for interconnected power system, Int. J. Electr. Power Eng. 3 (2009), 26–30.

[46] J. Kennedy, Particle swarm optimization, in: C. Sammut, G.I. Webb (Eds.), Encyclopedia of Machine Learning, Springer, New York, 2011, pp. 760–766.
[47] K.M. Passino, Biomimicry of bacterial foraging for distributed optimization and control, IEEE Control Syst. 22 (2002), 52–67.
[48] X. Yan, Y. Zhu, H. Zhang, H. Chen, B. Niu, An adaptive bacterial foraging optimization algorithm with lifecycle and social learning, Discrete Dyn. Nat. Soc. 2012 (2012), 1–20.
[49] T. Nguyen, B.M. Nguyen, G. Nguyen, Building resource auto-scaler with functional-link neural network and adaptive bacterial foraging optimization, in International Conference on Theory and Applications of Models of Computation, Lecture Notes in Computer Science, Springer, Kitakyushu, 2019, pp. 501–517.
[50] S. Mirjalili, Moth-flame optimization algorithm: a novel nature-inspired heuristic paradigm, Knowl. Based Syst. 89 (2015), 228–249.
[51] S. Mirjalili, A. Lewis, The whale optimization algorithm, Adv. Eng. Softw. 95 (2016), 51–67.
[52] S. Mirjalili, A.H. Gandomi, S.Z. Mirjalili, S. Saremi, H. Faris, S.M. Mirjalili, Salp swarm algorithm: a bio-inspired optimizer for engineering design problems, Adv. Eng. Softw. 114 (2017), 163–191.
[53] S. Saremi, S. Mirjalili, A. Lewis, Grasshopper optimisation algorithm: theory and application, Adv. Eng. Softw. (2017), 30–47.
[54] A.A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, H. Chen, Harris hawks optimization: algorithm and applications, Future Gener. Comput. Syst. 97 (2019), 849–872.
[55] S. Arora,S. Singh, Butterfly optimization algorithm: a novel approach for global optimization, Soft Comput. 23 (2019), 715–734.
[56] S. Shadravan, H. Naji, V.K. Bardsiri, The sailfish optimizer: A novel nature-inspired metaheuristic algorithm for solving constrained engineering optimization problems, Eng. Appl. Artif. Intell., 80 (2019), 20–34.
[57] S. Salcedo-Sanz, J. Del Ser, I. Landa-Torres, S. Gil-López, J. Portilla-Figueras, The coral reefs optimization algorithm: a novel metaheuristic for efficiently solving optimization problems, Scientific World Journal. 2014 (2014), 1–15.
[58] S. Salcedo-Sanz, C. Casanova-Mateo, A. Pastor-Sanchez, M. Sanchez-Giron, Daily global solar radiation prediction based on a hybrid coral reefs optimization–extreme learning machine approach, Sol. Energy. 105 (2014), 91–98.
[59] S. Salcedo-Sanz, A. Pastor-Sánchez, J. Del Ser, L. Prieto, Z.W. Geem, A coral reefs optimization algorithm with harmony search operators for accurate wind speed prediction, Renew. Energy, 75 (2015), 93–101.
[60] M. Li, C. Miao, C. Leung, A coral reef algorithm based on learning automata for the coverage control problem of heterogeneous directional sensor networks, Sensors. 15 (2015), 30617–30635.
[61] M. Ficco, C. Esposito, F. Palmieri, A. Castiglione, A coral-reefs and game theory-based approach for optimizing elastic cloud resource allocation, Future Gener. Comput. Syst. 78 (2018), 343–352.
[62] S. Salcedo-Sanz, A. Pastor-Sanchez, L. Prieto, A. Blanco-Aguilera, R. Garcia-Herrera, Feature selection in wind speed prediction systems based on a hybrid coral reefs optimization–extreme learning machine approach, Energy Convers. Manag. 87 (2014), 10–18.
[63] H.R. Tizhoosh, Opposition-based learning: a new scheme for machine intelligence, in International conference on Computational intelligence for modelling, control and automation, 2005 and international conference on intelligent agents, web technologies and internet commerce, IEEE, 2005, vol. 1, pp. 695–701.
[64] T. Nguyen, Ocro code, 2019. https://github.com/chasebk/code_ocro_mlnn.
[65] S. Mahdavi, S. Rahnamayan, K. Deb, Opposition based learning: a literature review, Swarm Evol. Comput. 39 (2018), 1–23.
[66] S. Salcedo-Sanz, C. Camacho-Gómez, D. Molina, F. Herrera, A coral reefs optimization algorithm with substrate layers and local search for large scale global optimization, in 2016 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2016, pp. 3574–3581.
[67] K. Hornik, M. Stinchcombe, H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural Netw. 3 (1990), 551–560.
[68] P. Cortez, M. Rio, M. Rocha, P. Sousa, Multi-scale internet traffic forecasting using neural networks and time series methods, Expert Syst. 29 (2012), 143–155.
[69] C. Reiss, A. Tumanov, G.R. Ganger, R.H. Katz, M.A. Kozuch, Heterogeneity and dynamicity of clouds at scale: Google trace analysis, in Proceedings of the Third ACM Symposium on Cloud Computing, ACM, San Jose, 2012, p. 7.
[70] C. Reiss, J. Wilkes, J.L. Hellerstein, Google Cluster-Usage Traces: Format+ Schema, Google Inc., White Paper, 2011, pp. 1–14.
[71] S. Arlot, A. Celisse, *et al.*, A survey of cross-validation procedures for model selection, Stat. Surv. 4 (2010), 40–79.
[72] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, Berlin, Heidelberg, 2006.
[73] Y. Shi, *et al.*, Particle swarm optimization: developments, applications and resources, in Proceedings of the 2001 Congress on Evolutionary Computation, 2001, IEEE, 2001, vol. 2, pp. 81–86.
[74] B.M. Nguyen, D. Tran, G. Nguyen, Enhancing service capability with multiple finite capacity server queues in cloud data centers, Cluster Comput. 19 (2016), 1747–1767.