

International Journal of Computational Intelligence Systems

Efficient Time-series Forecasting using Neural Network and Opposition-based Coral Reefs Optimization

--Manuscript Draft--

Manuscript Number:	IJ-CIS-D-19-00158
Full Title:	Efficient Time-series Forecasting using Neural Network and Opposition-based Coral Reefs Optimization
Article Type:	Regular paper
Keywords:	Meta-heuristic algorithms; Neural Network; Coral Reefs Optimization; Opposition-Based Learning; Time series forecasting; Google trace dataset
Corresponding Author:	Minh Binh Nguyen, Ph.D. Hanoi University of Science and Technology Hanoi, Vietnam VIET NAM
Corresponding Author E-Mail:	minhnb@soict.hust.edu.vn
Corresponding Author's Institution:	Hanoi University of Science and Technology
Other Authors:	Thieu Nguyen, MSc.
	Tu Nguyen, BSc.
	Giang Nguyen, Ph.D.
Abstract:	<p>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 effective modeling as the main part of this work, a novel time series forecasting model called OCRO-MLNN is proposed to explore hidden relationships in the non-linear time series data. The model thus combines OCRO with Multi-Layer Neural Network (MLNN) for data processing, which enables reducing the model complexity by faster convergence than traditional back-propagation algorithm. For validation of the proposed model, three real-world datasets are used, including Internet traffic collected from a private ISP with distributed centers in 11 European cities, World Cup98 contains request numbers to 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 RNN and 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 those infrastructures.</p>
Suggested Reviewers:	
Opposed Reviewers:	



Efficient Time-series Forecasting using Neural Network and Opposition-based Coral Reefs Optimization

Thieu Nguyen¹, Tu Nguyen¹, Binh Minh Nguyen^{1*}, Giang Nguyen²

¹ School of Information and Communication Technology, Hanoi University of Science and Technology,
Hanoi, 10000, VietNam

E-mail: nguyenthieu2102@gmail.com, nguyenanhtu3198@gmail.com, minhnb@soict.hust.edu.vn*

² Institute of Informatics, Slovak Academy of Sciences,
Bratislava, 81421, Slovakia

E-mail: giang.ui@savba.sk

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 effective modeling as the main part of this work, a novel time series forecasting model called OCRO-MLNN is proposed to explore hidden relationships in the non-linear time series data. The model thus combines OCRO with Multi-Layer Neural Network (MLNN) for data processing, which enables reducing the model complexity by faster convergence than traditional back-propagation algorithm. For validation of the proposed model, three real-world datasets are used, including Internet traffic collected from a private ISP with distributed centers in 11 European cities, World Cup98 contains request numbers to 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 RNN and 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 those infrastructures.

Keywords: Meta-heuristic algorithms; Neural Network; Coral Reefs Optimization; Opposition-Based Learning; Time series forecasting; Google trace dataset

1. Introduction

Today time series analytics is playing an important role in many real-life areas when the development of information technology and data explosion are blooming. The list of these areas gets longer, for example: predicting human behaviors using collected data from wearable devices, forecasting flooding based on historical water level data of rivers and

springs, or monitoring 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 approach is very useful when there is a little knowledge of underlying data generation process as well as when there is no adequate mathematical model to explain the relationship between predictor and other variables. Over the past several decades,

* Corresponding author.

there is a great effort devoted to the development and improvement of time series analytics using various techniques.

The idea of time-series forecasting of value y at time point t based on the y values at previous time points (i.e. $t - 1, t - 2, \dots, t - k$), and adding/subtracting error terms from previous time points was presented in the “*Time Series Analysis: Forecasting and Control*” monograph [2]. In which they showed that non-stationary data could be made stationary by differencing the series. The statistical model goes further with the most famous ARIMA model [27] that typically is expressed in the form of $ARIMA(p, d, q)$, where p , d , and q is the number of autoregressive terms, nonseasonal differences needed for stationarity, and lagged forecast errors in the prediction equation, respectively.

To overcome the stationary assumption of statistical models, artificial neural network (ANN) has been extensively studied and wisely used also in time series forecasting [55]. The main advantage of ANN is the ability to adapt to nonlinear models (i.e. instead of specifying a concrete model template, the model is formulated based on the features detected from the data). This data-based approach is consistent with many empirical data sets, where there is no theoretical guideline to propose a suitable data generation process. Particularly in this direction, deep learning (DL), especially recurrent neural network (RNN) [38] with internal self-looped cells have the ability to remember time series information and thus the network suits forecast goal for the data type. RNN built from Long Short-Term Memory (LSTM) [12] blocks work well in processing long term time series data, which is currently considered as state-of-the-art for time-series forecasting performance from the in accuracy viewpoint. However, DL techniques are compute-intensive that implies longer runtime.

Another way to deal with time-series prediction goes through optimization using various evolutionary bio-inspired algorithms [53], especially when most real-world optimizations are highly nonlinear and under various complex constraints. This kind of techniques trades in solution quality for runtime, by finding very good (but not necessarily op-

timal) solutions within feasible time. Genetic Algorithms (GAs) are very popular evolutionary approach with a wide range of applications. Other algorithms can be listed such as 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 that 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 existing state-of-the-art models.

In order to test and evaluate the built model, in this work, optimizing operations of distributed applications is considered. 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 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 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 applications run on modern computing paradigms like clouds.

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 presents our proposed prediction model designs based on the

Opposite-base Coral Reefs Optimization (OCRO) for Multi-Layer Neural Network (MLNN). In the next Section, we present experiments as well as evaluations for the proposed time series prediction model to prove its efficiency. The last Section concludes and defines some of our future work directions.

2. Related Work

Statistic modeling. One of the most widely known and used time series prediction models is autoregressive integrated moving average (ARIMA). The popularity of ARIMA is due to its statistical properties as well as effectiveness. Thus, ARIMA [25] is quite flexible in representing different time series modeling such as standard autoregressive (AR), standard moving average (MA) or AR and MA combinations (ARMA) series. However, their main limitation is the linear form and they are appropriated only for a time series that is stationary (i.e. its mean, variance, and autocorrelation should be approximately constant through time) [31]. Besides, ARIMA models can not properly capture non-linear 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 [16].

Artificial Neural Networks (ANNs) or Neural Networks (NNs). 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 [47] 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 [15] presented an eight-step procedure to design neural network forecasting model for financial and economic time series. In [34], the authors proposed use of FFNN for inflows synthesis. FFNN offers a viable alternative also for multivariate modeling of water resources time series. Zhang and Hu

in the work [54] 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 a larger sample size. In [7], 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 the neural network and linear regression models. In [11], the authors designed 3-layer FFNNs to predict the hourly solar radiation data, the result shown that FFNN outperform linear prediction filters. Most of the studies reported above were simple applications of using traditional time series approaches and ANNs (FFNNs/MLNNs). In [7], the authors analyzed 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.

Recurrent Neural Networks (RNNs). 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 [3, 10, 32, 33, 39]. 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. Pre-

vious studies [12] 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 [9, 17, 40]. Furthermore, recently, to address these drawbacks of RNNs, LSTM [12] 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 [1, 22, 48, 50, 56]. Even LSTM is considered as current state-of-the-art in the time series prediction field, it still has drawbacks. Concretely, it requires a lot of data to train in order to gain more accurate, as well as computational requirement due to the model complexity [6]. In general, RNNs are not parallelism-friendly due to their nested loop structures.

Neuro-evolution. In "neuro-evolution" research community, GAs [13] are known as efficient search algorithms. Recently, GA-based research has been applied to NN models [26, 51] in order to address computational cost, easy implement in hardware, and better accuracy. In this way, the research direction has been attracted to lots of researcher in many fields such as manufacturing process [4], stock markets [20], energy consumption [23], medical diagnosis [18], resource usage in cloud [29]. Unfortunately, recent research [35] identified some of 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 term of combining NN and optimization algorithms, swarm intelligent optimization motivated by social behavior in biological system also has attracted many researchers. Particle Swarm Optimiza-

tion (PSO) was stimulated by swarm behavior of bird flocking or fish schooling has been used in several real-life area [19]. Bacterial Foraging Optimization (BFO), which is an evolutionary computing technique inspired based on principle of bacterial movement (i.e. tumbling, swimming or repositioning) to food-seeking [30]. An improved version of BFO is Adaptive Bacterial Foraging Optimization with life-cycle and social learning (ABFO) was developed and presented in [52]. It has been tested on several set of benchmark functions with multiple dimensions, and used in distributed systems like cloud computing [28].

Coral Reefs Optimization (CRO). This technique also belongs to swarm intelligent optimization algorithms motivated by social behavior in biological system like PSO or BFO. It is proposed in [43] tackling optimization problems by modeling and simulating corals reproduction and formation. A lot of applications have been carried out in many areas such as prediction energy [42, 44], sensor networks [21], and cloud resource allocation [8].

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 [45], 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 budding process.

Like other meta-heuristic algorithms, CRO has also problem with local optimums. The diversity of corals in a reef quickly decreases after 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 [49].

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.

Currently based on our knowledge, there is no study that applies the CRO to MLNN and OCRO to MLNN. The combinations of CRO and OBL property when applying to MLNN not only improve the drawbacks of gradient decent algorithm, but also enables fast convergence to optimal values as well as reducing computational cost. In comparison with above-mentioned 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 Coral Reefs Optimization (CRO) algorithm using Opposition-Based Learning (OBL). OCRO aims to improve searching power and jumping out from local minimum of the meta-heuristic CRO.
2. Proposing the new time-series forecasting model called OCRO-MLNN, which is designed based on MLNN variance with OCRO algorithm to train forecasting model instead of back-propagation.
3. Carrying out comparisons among the novel proposed prediction model with five well-known ones (i.e. MLNN, GA-MLNN, CRO-MLNN, RNN and LSTM) for time series forecast.
4. Evaluating and proving the effectiveness of OCRO-MLNN as compared with others using three kinds of real-world datasets (i.e. above-mentioned 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 (OCRO)

3.1. Coral Reefs Optimization (CRO)

As mentioned above, CRO is an optimization algorithm (bio-)inspired by behaviors of corals reproduction and reef formation. It was originally introduced in [43]. The skeleton of original CRO is summarized in Algorithm ?? that includes two main below-described parts.

Part 1: Initialization of CRO parameters.

The main control parameters of CRO are as follows: a coral reef consists of a $N \times M$ square grid that is similar to the population size in GA. The grids 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 problem.

Part 2: Reef information (consists of the following steps).

- (1) *Broadcast Spawning (external sexual reproduction):*
 - 1.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.
 - 1.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 a larvae, they are not selected any more 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 reef. If the grid is free space, coral can grow regardless to 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 F_a of the existing corals will copy itself and attempt to settle in a different part of 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, 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 (OCRO)

According to [45], the ability to exploit of CRO is controlled by broadcast spawning. This carries out the most of 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 a simulated annealing 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 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 Opposition-Based Learning (OBL) [49] mathematical concept well-known in reinforcement learning, ANNs, and fuzzy systems. 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 [24]. 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 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:

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

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 ???. To prevent premature convergence to a local optimum, in our improvement, we restart search process when the best solution is not improved after several generations based on [41]. 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 [55]. The model includes three layers which are input, hidden, and output layer as represented by Fig. 1 (left side). Each of them has simple processing units connected by acyclic links. The re-

lationship between outputs (y_{t+k}) and inputs ($y_{t-1}, y_{t-2}, \dots, y_{t-p}$) is described in Equation 2.

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

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

where

- α_{ij} is connection weight for $i = 0, 1, \dots, p$ and $j = 1, 2, \dots, q$,
- β_{jk} is connection bias for $i = 0, 1, \dots, p$ and $j = 1, 2, \dots, q$,
- 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. When $\gamma = 0$ then ELU becomes the well-known Rectified Linear Unit (ReLU). When the number of hidden neurons is large enough, the network shown in Equation 2 can approximate arbitrary function [14]. 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 to 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.
- The same situation as mentioned above also occurs when choosing the number of lagged observations p as the dimension of 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 number of ahead steps. For example, when ($w = 1$), it means one-step-ahead forecasting. ($w > 1$) means multi-step-ahead prediction. 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 back-propagation technique. The number of variables estimated by OCRO is as follows.

$$q \cdot (p + 1) + w \cdot (q + 1) \quad (4)$$

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 Fig. 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 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 Fig. 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 achievement of error goal.

Algorithm ?? 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.
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)" [5] collected from a private Internet Service Provider (ISP)

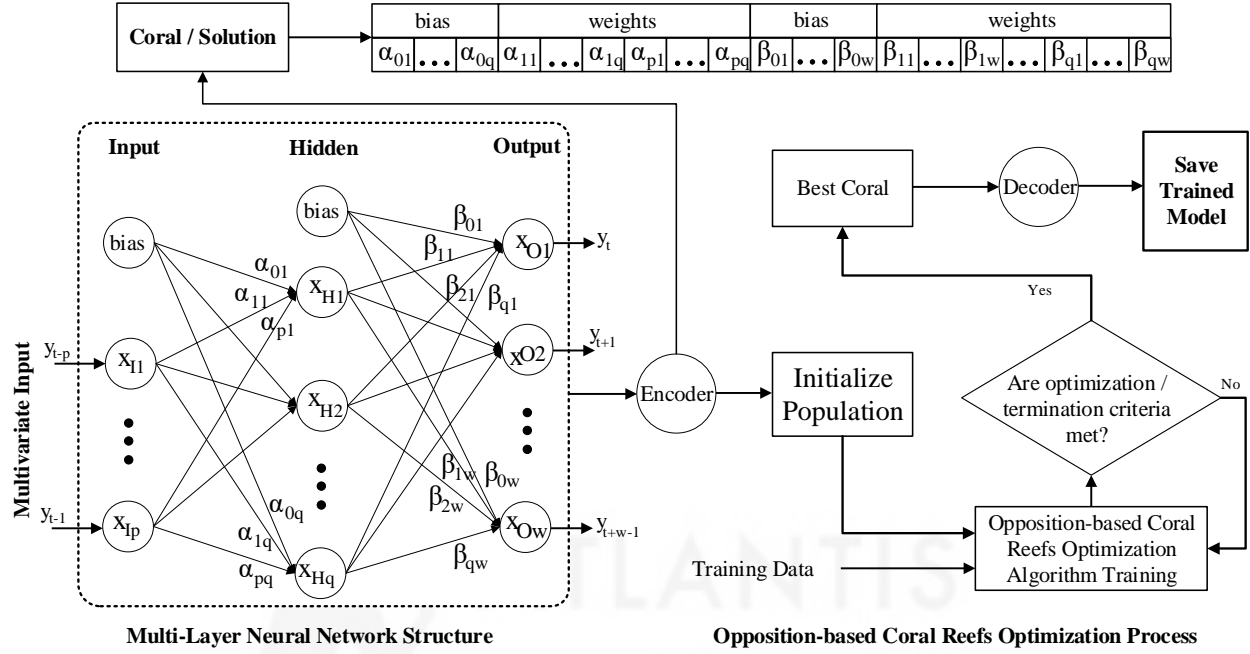


Figure 1: OCRO-MLNN training process

with distributed centers in 11 European 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 [36] and [37]. Based on our prior analysis presented in [29], we select both resource usages, which are 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 Fig. 2.

5.1.2. Measurement methods

Accuracy. In our work, Root Mean Square Error (RMSE) (Equation. 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), Mean Absolute Error (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 Equation. 6, Equation. 7, Equation. 8 and Equation. 9 respectively.

$$fitness = health = RMSE = \sqrt{\frac{\sum_{i=1}^N (forecast(i) - actual(i))^2}{N}} \quad (5)$$

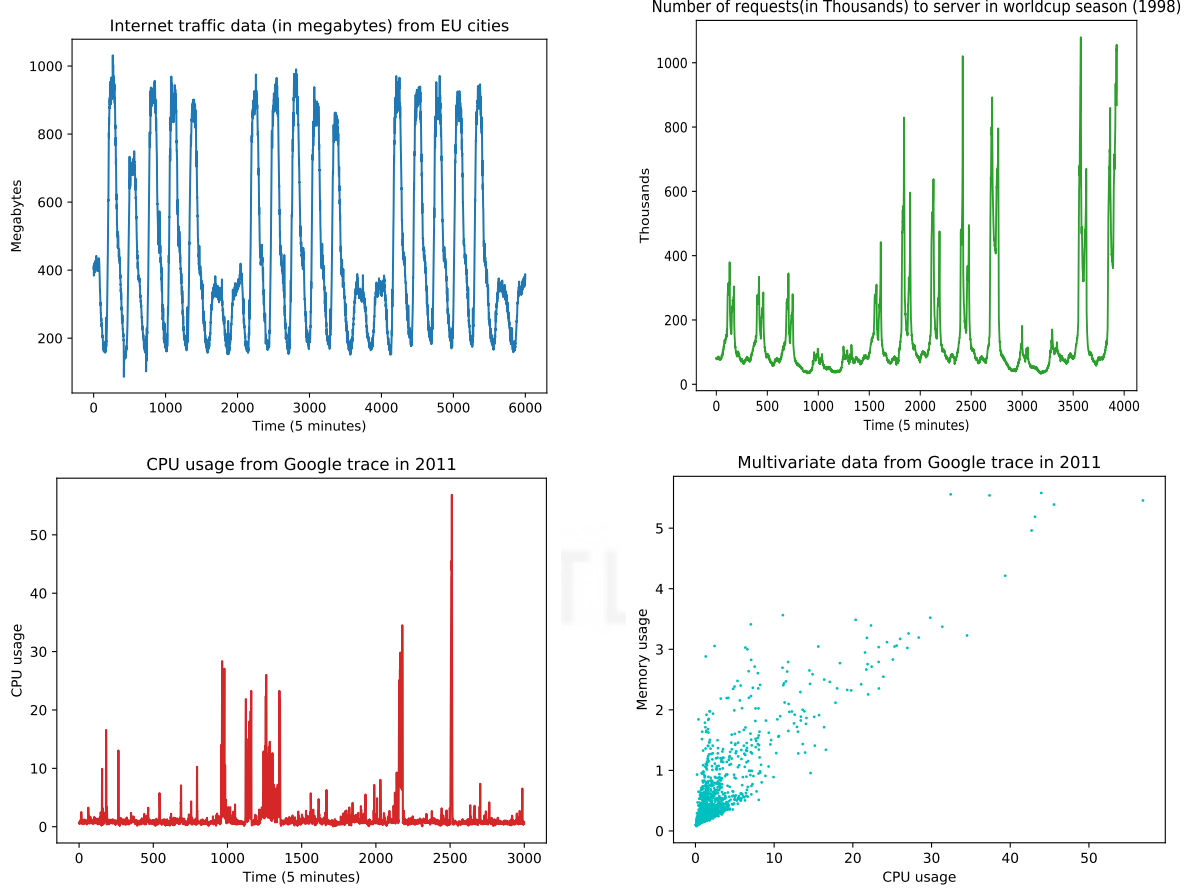


Figure 2: Dataset visualizations: univariate data (upper row) and multivariate data (lower row)

$$R^2 = 1 - \frac{\sum_{i=1}^N \left(\text{actual}(i) - \text{forecast}(i) \right)^2}{\sum_{i=1}^N \left(\text{actual}(i) - \frac{1}{N} \sum_{j=1}^N \text{actual}(j) \right)^2} \quad (6)$$

$$MAE = \frac{\sum_{i=1}^N |\text{forecast}(i) - \text{actual}(i)|}{N} \quad (7)$$

$$MAPE = \frac{100\%}{N} \sum_{i=1}^N \left| \frac{\text{actual}(i) - \text{forecast}(i)}{\text{actual}(i)} \right| \quad (8)$$

$$SMAPE = \frac{100\%}{N} \sum_{i=1}^N \frac{|\text{forecast}(i) - \text{actual}(i)|}{(|\text{actual}(i)| + |\text{forecast}(i)|)/2} \quad (9)$$

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 (pre-processing, 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 are is configured with the same structure configuration (three layers (input, hidden and output layer)). The architecture is similar to typical topology of one RNN/LSTM block [22].
- 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 [29]. 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 [29].
- For PSO, based on [46], 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 [29], 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 [43], 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 [43].
- 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, MLNN and ABFO-MLNN models did not bring performance well in accuracy mean as 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 get 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. Fig. 3 and Fig. 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 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 issue causes 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 complicated structure, which is hard to optimize. Besides, the time cost also depends on 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 its slow convergence, hence they need to train with more epochs. The overall effect is also longer time on train.
- OCRO-MLNN and CRO-MLNN provide good runtime performance to complete all work with encouraged results. The reason is their short runtime per epoch and quick convergence.
- EU traffic dataset: OCRO-MLNN's runtime $t_s = 353.79s$ ($k = 2$) in comparison with GA-MLNN's $t_s = 657.77s$ and PSO-MLNN's $t_s = 642.73s$.
- WC dataset: OCRO-MLNN's runtime $t_s = 242.33s$ ($k = 5$) in comparison with CRO-MLNN's $t_s = 256.59s$ and MLNN's $t_s = 459.92s$

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 $cv = 0.3836$.

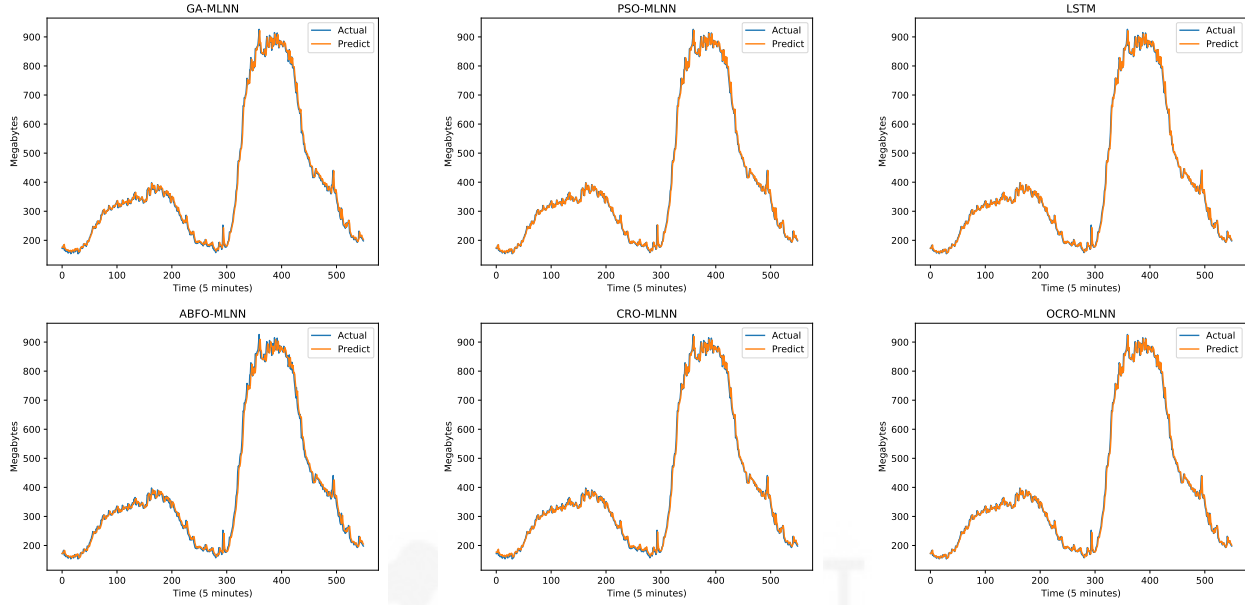


Figure 3: Comparison of OCRO-MLNN prediction with other models (sliding window $k = 2$; EU dataset)

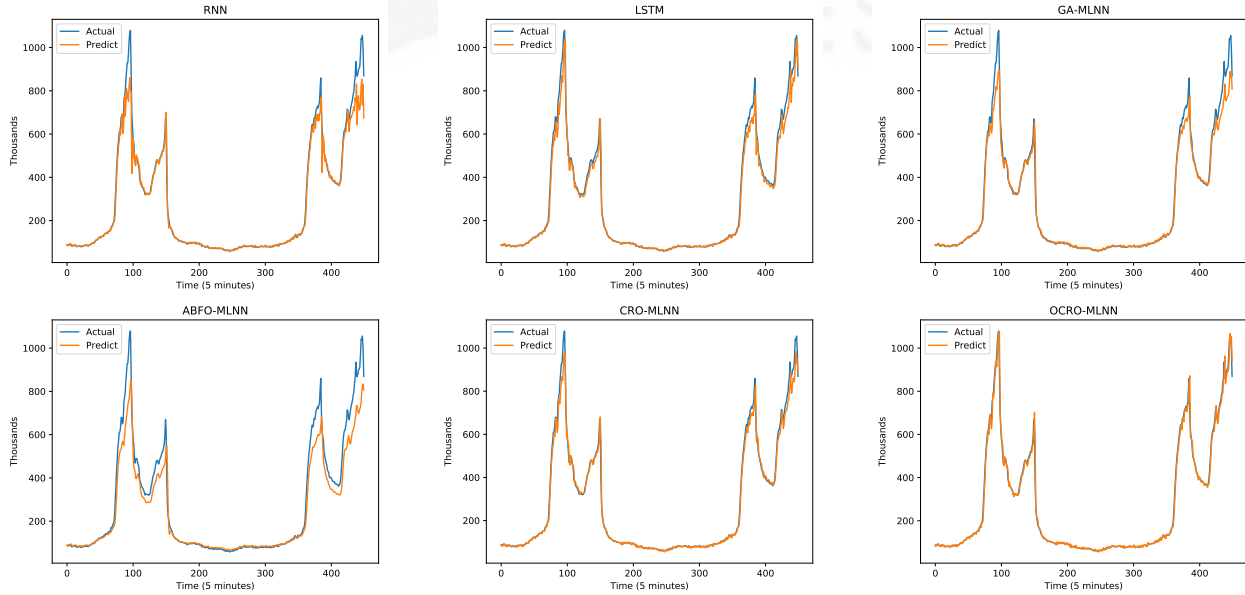


Figure 4: Comparison of OCRO-MLNN prediction with other models (sliding window $k = 5$; WC dataset)

- 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.

Fig. 9 visualizes RMSE range of 15 run times average 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 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 achieves the worst results in this test. The low accuracy problem of this model cause by gradient decent when searching for an optimal solution in a complex dimension.
- OCRO-MLNN obtains the best results indicating high accuracies in both multivariate 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.
 - 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 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. Fig. 5 and Fig. 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. Fig. 8 visualizes system time of our OCRO-MLNN model (green color) in comparison with the rest.

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 to apply meta-heuristic OCRO to optimization problems gives more stability than gradient decent 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).

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

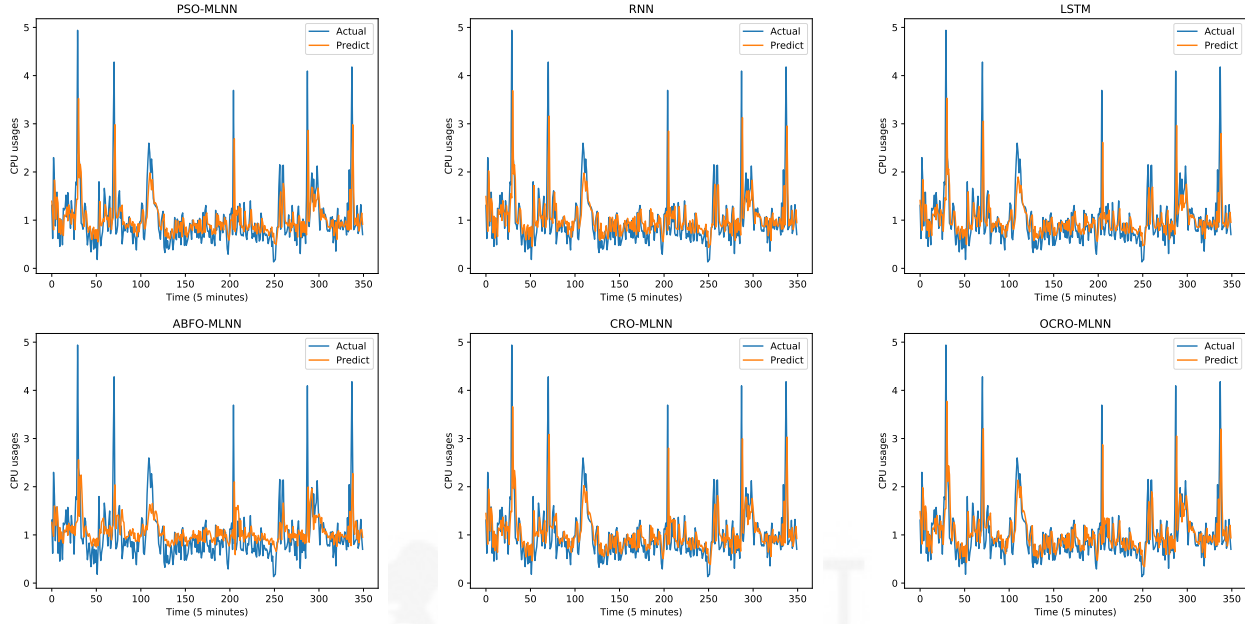


Figure 5: Comparison of OCRO-MLNN prediction with other models (sliding window $k = 2$; multivariate data (CPU))

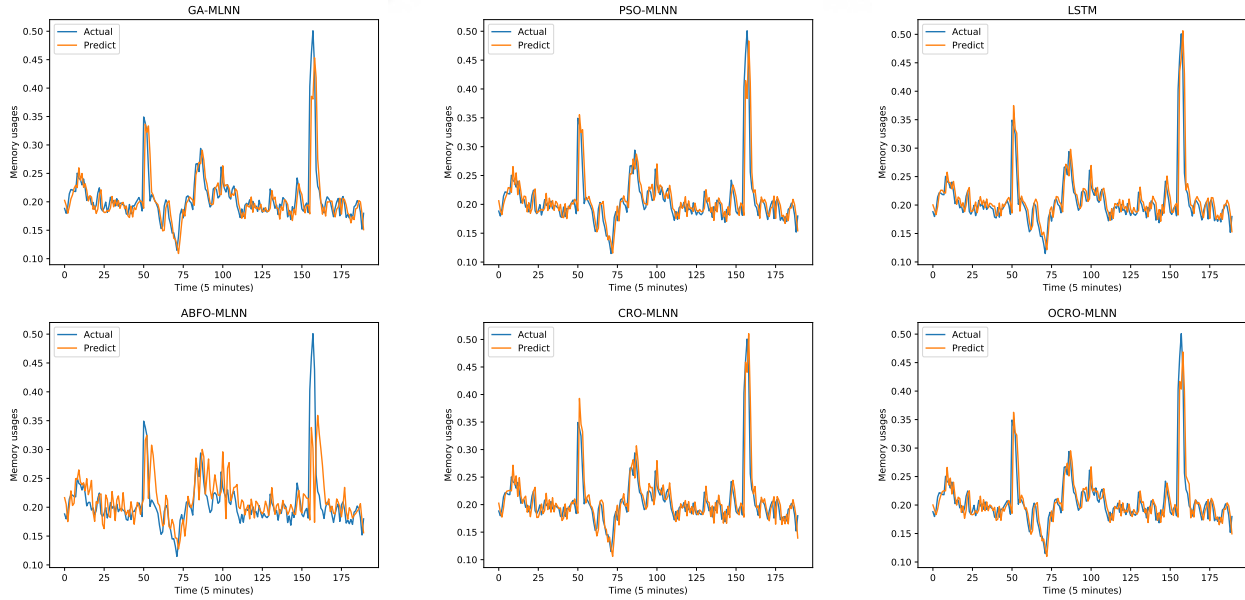


Figure 6: Comparison of OCRO-MLNN prediction with other models (sliding window $k = 5$; multivariate data (RAM))

dataset).

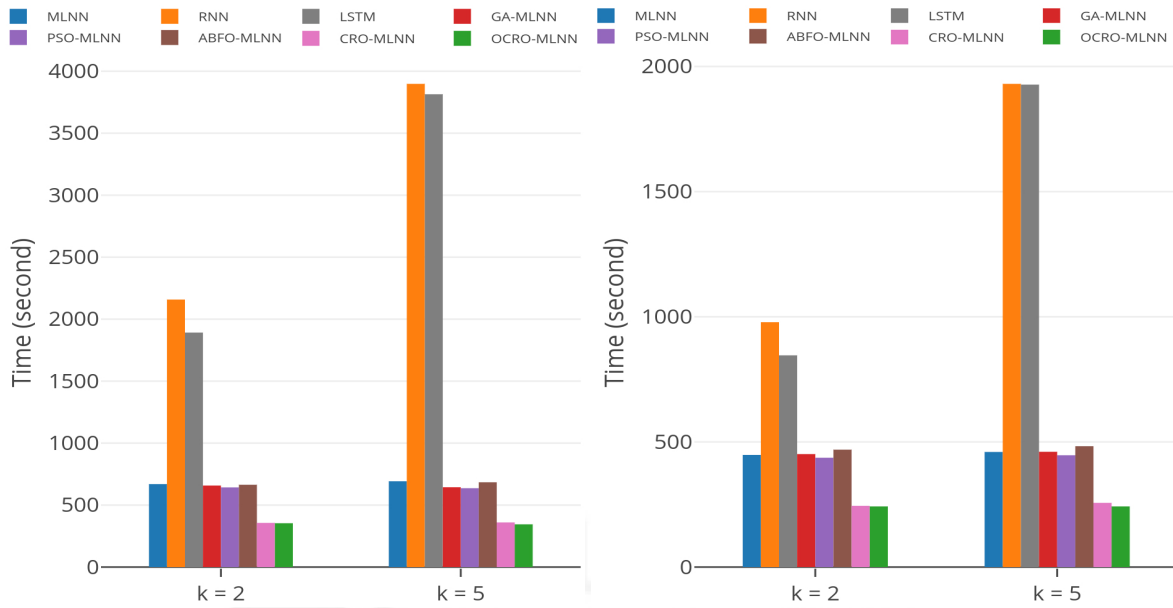


Figure 7: Runtime comparison: OCRO-MLNN (green), sliding window $k = 2$ and $k = 5$; univariate data: EU (left), WC (right)

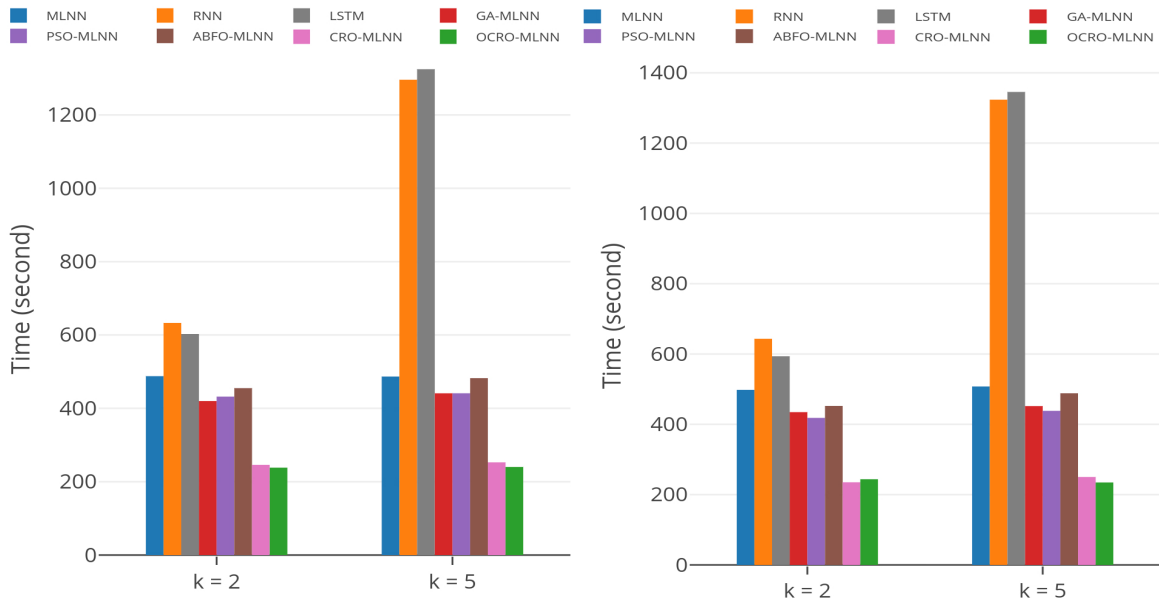


Figure 8: Runtime comparison: OCRO-MLNN (green), sliding window $k = 2$ and $k = 5$; multivariate data, CPU (left), RAM (right)

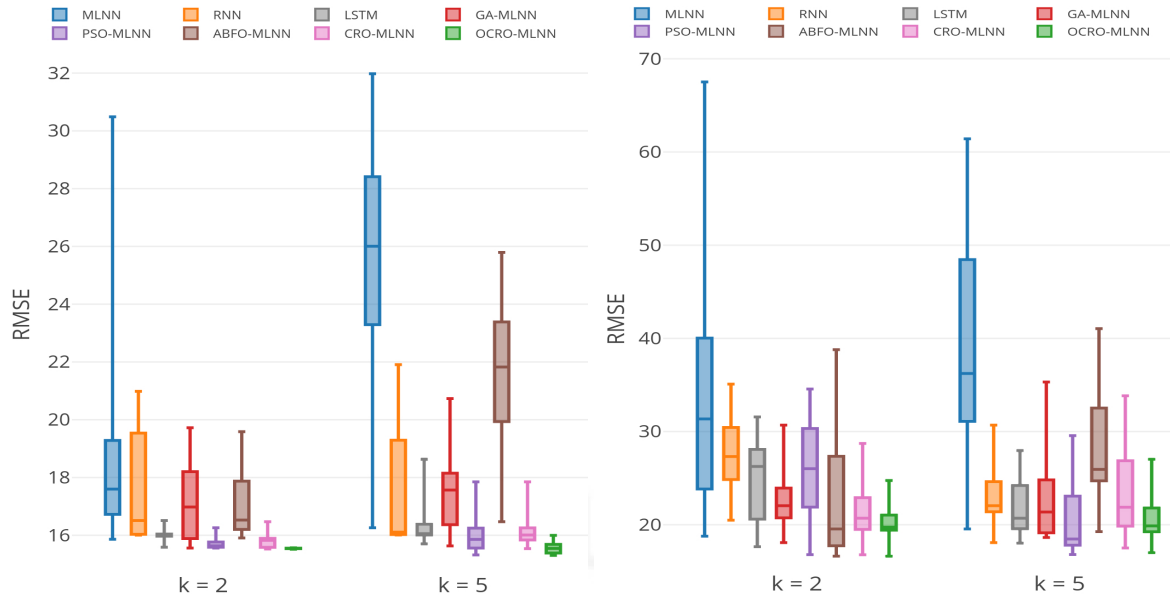


Figure 9: RMSE comparison: average of 15 times; univariate data: EU (left), WC (right)

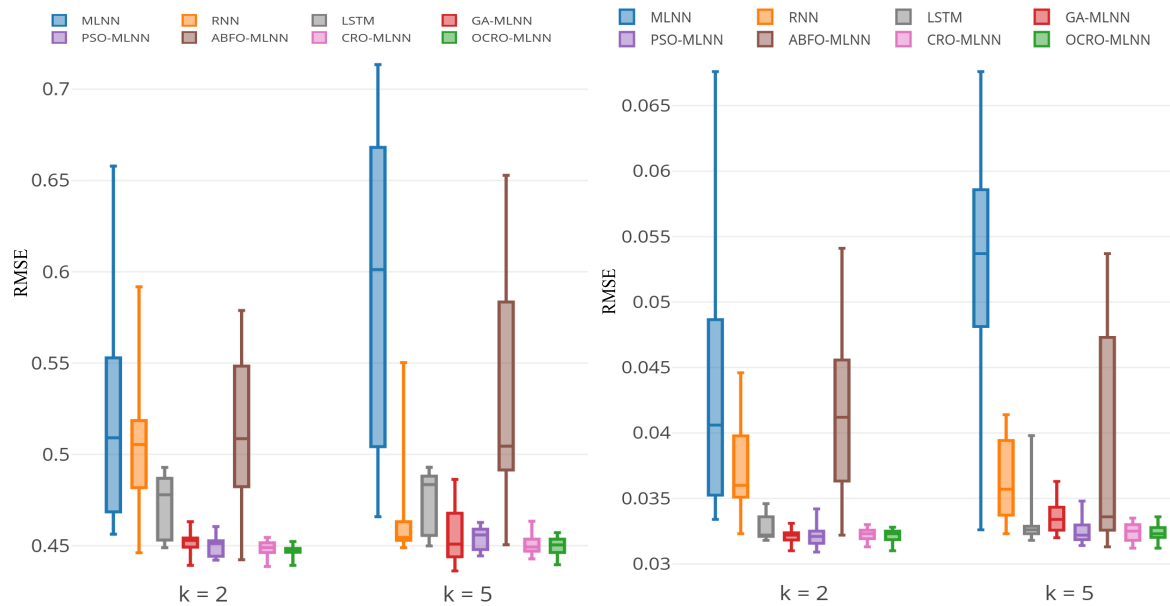


Figure 10: RMSE comparison: average of 15 times; multivariate data: CPU (left), RAM (right)

5.3. Practical usage of the proposed forecasting approach

In 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, networks, storages), a scheduler 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 main disadvantage in operation and management of distributed applications: there are 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.

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 scaling issue. Moreover, the model also brings the 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 ap-

proach is based on idea that build a model with simple structure, stable, low run-time training, but still brings a good accuracy as compare 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 Coral Reefs Optimization (CRO) using Opposition-Based Learning (OBL). Thus, we used OCRO together with multi-layer neural network to deal with the forecast problem in the non-linear time series data. By the improvement, our OCRO-MLNN reducing complexity of the model due to faster convergence in comparison with 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 prove OCRO-MLNN's feasibility in applying to real systems in practice.

The study made contributions in four areas. First, we built OCRO technique by using OBL mechanism for CRO. The improvement helps searching process and avoid 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 the 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 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 cloud and high-performance systems. 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.

Acknowledgments

This research is funded by the Hanoi University of Science and Technology (HUST) under project number T2018-PC-017, Slovak APVV-17-0619 and EU H2020-777533 PROCESS.

References

- [1] A. Azzouni and G. Pujolle. A long short-term memory recurrent neural network framework for network traffic matrix prediction. *arXiv preprint arXiv:1705.05690*, 2017.
- [2] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time series analysis: forecasting and control*. John Wiley and Sons, 2015.
- [3] J. T. Connor, R. D. Martin, and L. E. Atlas. Recurrent neural networks and robust time series prediction. *IEEE transactions on neural networks*, 5(2):240–254, 1994.
- [4] D. F. Cook, C. T. Ragsdale, and R. Major. Combining a neural network with a genetic algorithm for process parameter optimization. *Engineering applications of artificial intelligence*, 13(4):391–396, 2000.
- [5] P. Cortez, M. Rio, M. Rocha, and P. Sousa. Multi-scale internet traffic forecasting using neural networks and time series methods. *Expert Systems*, 29(2):143–155, 2012.
- [6] E. Culurciello. Computation and memory bandwidth in deep neural networks. Medium, 5 2017.
- [7] D. Enke and S. Thawornwong. The use of data mining and neural networks for forecasting stock market returns. *Expert Systems with applications*, 29(4):927–940, 2005.
- [8] M. Ficco, C. Esposito, F. Palmieri, and A. Castiglione. A coral-reefs and game theory-based approach for optimizing elastic cloud resource allocation. *Future Generation Computer Systems*, 78:343–352, 2018.
- [9] R. Gencay and T. Liu. Nonlinear modelling and prediction with feedforward and recurrent networks. *Physica D: Nonlinear Phenomena*, 108(1-2):119–134, 1997.
- [10] M. Han, J. Xi, S. Xu, and F.-L. Yin. Prediction of chaotic time series based on the recurrent predictor neural network. *IEEE transactions on signal processing*, 52(12):3409–3416, 2004.
- [11] F. O. Hocaoglu, O. N. Gerek, and M. Kurban. Hourly solar radiation forecasting using optimal coefficient 2-d linear filters and feed-forward neural networks. *Solar energy*, 82(8):714–726, 2008.
- [12] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [13] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

- [14] K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feed-forward networks. *Neural networks*, 3(5):551–560, 1990.
- [15] I. Kaastra and M. Boyd. Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 10(3):215–236, 1996.
- [16] Y. Kajitani, K. W. Hipel, and A. I. McLeod. Forecasting nonlinear time series with feed-forward neural networks: a case study of canadian lynx data. *Journal of Forecasting*, 24(2):105–117, 2005.
- [17] K. Kalaitzakis, G. Stavrakakis, and E. Anagnostakis. Short-term load forecasting based on artificial neural networks parallel implementation. *Electric Power Systems Research*, 63(3):185–196, 2002.
- [18] A. G. Karegowda, A. Manjunath, and M. Jayaram. Application of genetic algorithm optimized neural network connection weights for medical diagnosis of pima indians diabetes. *International Journal on Soft Computing*, 2(2):15–23, 2011.
- [19] J. Kennedy. Particle swarm optimization. In *Encyclopedia of machine learning*, pages 760–766. Springer, 2011.
- [20] H.-j. Kim and K.-s. Shin. A hybrid approach based on neural networks and genetic algorithms for detecting temporal patterns in stock markets. *Applied Soft Computing*, 7(2):569–576, 2007.
- [21] M. Li, C. Miao, and C. Leung. A coral reef algorithm based on learning automata for the coverage control problem of heterogeneous directional sensor networks. *Sensors*, 15(12):30617–30635, 2015.
- [22] X. Ma, Z. Tao, Y. Wang, H. Yu, and Y. Wang. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, 54:187–197, 2015.
- [23] L. Magnier and F. Haghighat. Multiobjective optimization of building design using trnsys simulations, genetic algorithm, and artificial neural network. *Building and Environment*, 45(3):739–746, 2010.
- [24] S. Mahdavi, S. Rahnamayan, and K. Deb. Opposition based learning: A literature review. *Swarm and evolutionary computation*, 39:1–23, 2018.
- [25] E. McKenzie. General exponential smoothing and the equivalent arma process. *Journal of Forecasting*, 3(3):333–344, 1984.
- [26] D. J. Montana and L. Davis. Training feed-forward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767, 1989.
- [27] R. Nau. Arima models for time series forecasting, 2019.
- [28] T. Nguyen, B. M. Nguyen, and G. Nguyen. Building resource auto-scaler with functional-link neural network and adaptive bacterial foraging optimization. *Lecture Notes in Computer Science 11436*, 2019.
- [29] T. Nguyen, N. Tran, B. M. Nguyen, and 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)*, pages 49–56. IEEE, 2018.
- [30] K. M. Passino. Biomimicry of bacterial foraging for distributed optimization and control. *IEEE control systems*, 22(3):52–67, 2002.

- [31] A.-C. Petricua, S. Stancu, and A. Tindecu. Limitation of arima models in financial and monetary economics. *Theoretical and Applied Economics*, 23(4), 2016.
- [32] A. Petrosian, D. Prokhorov, R. Homan, R. Dasheiff, and D. Wunsch II. Recurrent neural network based prediction of epileptic seizures in intra-and extracranial eeg. *Neuro-computing*, 30(1-4):201–218, 2000.
- [33] G. Pollastri, D. Przybylski, B. Rost, and P. Baldi. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins: Structure, Function, and Bioinformatics*, 47(2):228–235, 2002.
- [34] H. Raman and N. Sunilkumar. Multivariate modelling of water resources time series using artificial neural networks. *Hydrological Sciences Journal*, 40(2):145–163, 1995.
- [35] S. Ramesh and A. Krishnan. Modified genetic algorithm based load frequency controller for interconnected power system. *Int J Electr Power Eng*, 3(1):26–30, 2009.
- [36] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 7. ACM, 2012.
- [37] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, pages 1–14, 2011.
- [38] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.
- [39] E. W. Saad, D. V. Prokhorov, and D. C. Wunsch. Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. *IEEE Transactions on neural networks*, 9(6):1456–1470, 1998.
- [40] S. Saha and G. Raghava. Prediction of continuous b-cell epitopes in an antigen using recurrent neural network. *Proteins: Structure, Function, and Bioinformatics*, 65(1):40–48, 2006.
- [41] S. Salcedo-Sanz, C. Camacho-Gómez, D. Molina, and 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)*, pages 3574–3581. IEEE, 2016.
- [42] S. Salcedo-Sanz, C. Casanova-Mateo, A. Pastor-Sanchez, and M. Sanchez-Giron. Daily global solar radiation prediction based on a hybrid coral reefs optimization–extreme learning machine approach. *Solar Energy*, 105:91–98, 2014.
- [43] S. Salcedo-Sanz, J. Del Ser, I. Landa-Torres, S. Gil-López, and J. Portilla-Figueras. The coral reefs optimization algorithm: a novel metaheuristic for efficiently solving optimization problems. *The Scientific World Journal*, 2014, 2014.
- [44] S. Salcedo-Sanz, A. Pastor-Sánchez, J. Del Ser, L. Prieto, and Z.-W. Geem. A coral reefs optimization algorithm with harmony search operators for accurate wind speed prediction. *Renewable Energy*, 75:93–101, 2015.
- [45] S. Salcedo-Sanz, A. Pastor-Sanchez, L. Prieto, A. Blanco-Aguilera, and R. Garcia-Herrera. Feature selection in wind speed prediction systems based on a hybrid coral reefs optimization–extreme learning machine approach. *Energy Conversion and Management*, 87:10–18, 2014.
- [46] Y. Shi et al. Particle swarm optimization: developments, applications and resources. In

evolutionary computation, 2001. *Proceedings of the 2001 Congress on*, volume 1, pages 81–86. IEEE, 2001.

- [47] D. Srinivasan, A. Liew, and C. Chang. A neural network short-term load forecaster. *Electric Power Systems Research*, 28(3):227–234, 1994.
- [48] Y. Tian and L. Pan. Predicting short-term traffic flow by long short-term memory recurrent neural network. In *Smart City/SocialCom/SustainCom (SmartCity), 2015 IEEE International Conference on*, pages 153–158. IEEE, 2015.
- [49] H. R. Tizhoosh. Opposition-based learning: a new scheme for machine intelligence. In *Computational intelligence for modelling, control and automation, 2005 and international conference on intelligent agents, web technologies and internet commerce, international conference on*, volume 1, pages 695–701. IEEE, 2005.
- [50] N. Tran, T. Nguyen, B. M. Nguyen, and G. Nguyen. A multivariate fuzzy time series resource forecast model for clouds using lstm and data correlation analysis. *Procedia Computer Science*, 126:636–645, 2018.
- [51] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel computing*, 14(3):347–361, 1990.
- [52] X. Yan, Y. Zhu, H. Zhang, H. Chen, and B. Niu. An adaptive bacterial foraging optimization algorithm with lifecycle and social learning. *Discrete Dynamics in Nature and Society*, 2012, 2012.
- [53] X.-S. Yang. Metaheuristic optimization, 2011.
- [54] G. Zhang and M. Y. Hu. Neural network forecasting of the british pound/us dollar exchange rate. *Omega*, 26(4):495–506, 1998.
- [55] G. Zhang, B. E. Patuwo, and M. Y. Hu. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1):35–62, 1998.
- [56] Z. Zhao, W. Chen, X. Wu, P. C. Chen, and J. Liu. Lstm network: a deep learning approach for short-term traffic forecast. *IET Intelligent Transport Systems*, 11(2):68–75, 2017.

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 iters before restarting the algorithm

Table 1: Opposition-Based Coral Reefs Optimization (OCRO) parameters

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

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	0.1562
	RNN	16.0049	20.984	17.7525	1.8485	0.1041	16.0111	21.908	17.9194	2.2523	0.1256
	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	0.0761
	PSO-MLNN	15.5566	16.2584	15.7078	0.1961	0.0125	15.3195	17.8479	16.1036	0.7924	0.0491
	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	0.3204
	RNN	20.488	35.0887	27.4246	3.8517	0.1404	18.0851	30.6916	23.1305	3.3629	0.1457
	LSTM	17.6447	31.5707	24.6919	4.5824	0.1856	18.035	27.9699	22.1468	3.5013	0.1582
	GA-MLNN	18.0851	30.6916	22.7368	3.2264	0.1419	18.6348	35.3217	23.3347	5.1492	0.2191
	PSO-MLNN	16.7875	34.5682	26.4276	5.2702	0.1994	16.8053	29.5606	20.3284	3.8017	0.1819
	ABFO-MLNN	16.6236	38.7891	23.2257	6.8395	0.2945	19.2615	41.0411	28.1293	6.8486	0.2434
	CRO-MLNN	16.7806	28.7189	21.4259	3.4962	0.1632	17.5025	33.8349	23.4025	4.5302	0.1934
	OCRO-MLNN	16.6236	24.7484	20.0339	1.8681	0.0932	16.9945	27.0219	20.5392	2.5978	0.1265

Table 5: RMSE, MAPE and R^2 comparison (sliding window $k = 2$ and $k = 5$; multivariate dataset)

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

Table 6: Runtime comparison (sliding window $k = 2$ and $k = 5$, multivariate dataset)

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	$k = 2$					$k = 5$				
		<i>min</i>	<i>max</i>	<i>mean</i>	<i>std</i>	<i>cv</i>	<i>min</i>	<i>max</i>	<i>mean</i>	<i>std</i>	<i>cv</i>
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