ISSN 2278-3091
Volume 9, No.1.1, 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse8991.12020.pdf
https://doi.org/10.30534/ijatcse/2020/8991.12020

A Modified Cuckoo Search-Markov Chain Monte Carlo: The Alternative Gradient Free Optimisation Algorithm

Noor Aida Husaini*, Rozaida Ghazali[2], Lokman Hakim Ismail[3], Nureize Arbaiy[4], Habib Shah[5]
[1,2,4, 5]Faculty of Computer Science and Information Technology,
[3]Faculty of Civil and Environmental Engineering, Universiti Tun Hussein Onn Malaysia,
86400 Parit Raja, Batu Pahat, Johor
*noor.aida.husaini@gmail.com, rozaida@uthm.edu.my, lokman@uthm.edu.my, nureize@uthm.edu.my, habibshah.uthm@gmail.com

ABSTRACT

This paper aims to investigate the ability of the proposed Modified Cuckoo Search-Markov chain Monte Carlo (MCS-MCMC) algorithm for two (2) types of Higher Order Neural Networks (HONNs); Pi-Sigma Neural Networks and Functional Link Neural Networks that will influence the performance of searching ability, even at high numbers of dimensions. We validated the proposed MCS-MCMC algorithm alongside several benchmark test functions The proposed MCS-MCMC were tested on three (3) different time-series data; relative humidity, temperature and laser datasets. The performance of those HONNs is benchmarked against the performance of Multilayer Perceptron. The simulation results shows that, by incorporating MCS-MCMC algorithm in both HONNs can improve convergence rate and decrease the prediction error.

Key words : MCS-MCMC, modified cuckoo search, cuckoo search, Markov chain Monte Carlo.

1. INTRODUCTION

Artificial Neural Networks (ANNs) have been successfully applied to a variety of real-world classification task in industry, business and science [1, 2]. For prediction task, ANNs needs to be "trained" to produce the desired input-output mappings. The realism of using such gradient base optimisation techniques has been reduced by the difficulty of generating automatically objective functions and their derivatives for highly non-linear engineering problems [3-5]. Accordingly, the gradient search techniques in those ANNs prone to easily get trapped into local minima during training phase and may lead to poor network performance. The most common architecture of ANNs is the multilayer feedforward network or mostly known as multilayer perceptron (MLP) which utilises a supervised learning technique called backpropagation (BP) for training the network [2, 6-8]. Thus, since the BP learning algorithm is a gradient descent local optimisation technique, there is still have several major problems needed to be solved. Therefore, further investigations to improve the learning algorithm in ANNs are still desired [2, 9].

In the era of 1950s and 1960s, researchers investigated the prospect of applying the concepts of evolutions to a subclass of gradient free methods [3, 4, 10]. These intelligent mechanisms, which comprise Cuckoo Search (CS), offers great benefits over conventional modelling, including the proficiencies to employs high level techniques in exploring and exploiting the search space [10]. They are simple, more generic and robust. However, the CS will always find the optimum if it been given enough computation. Therefore, it is not guaranteed whether the exploration can converge faster or not.

After all, Walton et al. introduced two (2) modifications in the Modified Cuckoo Search (MCS) algorithm by changing the Lévy flight step size, α and adding up the information exchange between the top eggs to speed up convergence rate [4]. Despite of this huge success of MCS, there are many open problems which remain unanswered. We know how these heuristic algorithms work, and we also partly understand why these algorithms work. However, it is difficult to analyse mathematically why these algorithms are so successful, and along with others.

For all population-based metaheuristics, multiple search agents form multiple interacting Markov chains [4, 10]. Therefore, we possibly replacing the Lévy flight found in the MCS algorithm with Markov chain Monte Carlo (MCMC) random walk. A major advantage of the MCMC is that it learns better parameter automatically whilst ruling good parameter values by little user intervention. This paper explores the prediction capability of 2 types of HONNs; Pi-Sigma Neural Networks and Functional Link Neural Networks by considering the MCS-MCMC algorithm as the learning algorithms, replacing the current backpropagation (BP) learning, in order to reduce the prediction error.

The paper is mainly organised according to nine (9) sections. In the first section, a brief introduction regarding the area of the study is discussed. In the second part, the key concept regarding the random walks briefly explained. The third section comprises the network architecture of HONNs. In the fourth and fifth sections, the MCS and MCS-MCMC algorithm being explained. In the sixth and seventh sections, the experimental settings and benchmark test functions are presented. In the eighth section, simulation results are discussed. In the last section, some conclusions and future works are untaken.

2. RANDOM WALKS

The term "random walk" was originally proposed by Karl Pearson in 1905. It is a mathematical formalisation of a path that consists of a succession of random steps [11, 12]. The random walk usually deals with randomness which are lacking aim or method and not uniform. The random walk is assumed homogeneous, symmetric, irreducible, and having zero mean and finite variance of jumps. The areas of random walk includes the field of ecology [13], economics, computer science [12] and so on. Random walks explain the observed behaviours of processes in these fields, and thus serve as a fundamental model for the recorded stochastic activity. Some random walks are on graphs, others on the line, in the plane, or in higher dimensions, while some random walks are on groups [12].

2.1 Lévy Flight

A Lévy flight is a type of random walk that the distribution of the step-lengths is bounded in a heavy-tailed probability distribution. The typical properties of this kind of distribution is the positive exponential moments are infinite (do not have finite mean and variance). The term "Lévy flight" was invented by Benoît Mandelbrot [14] who used this for one specific definition of the distribution of step sizes. This gives yet another slogan:

"Cauchy flight", if the distribution of step size is a Cauchy distribution.

"Rayleigh flight", if the distribution is a normal distribution.

"Lévy flight", when the directions of the step sizes are in isotropic random directions.

Those random directions is defined by the survivor function (commonly known as the survival function) of the distribution of step-sizes, U , as in functional notation (1):

$$\Pr(\mathrm{U} > \mathrm{u}) = \begin{cases} 1 & : \ u < 1, \\ u^{-D} & : \ u \geq 1. \end{cases} \quad (1)$$

which says $D =$ parameter in the fractal dimension in Pareto distribution.

The distribution of step sizes can be any distribution for which the Lévy exponent of approximately 2 (also called power law) can provide a higher efficiency than other exponents:

$$PR(U > u) = O(u^{-\beta}), 1 < \beta < 3 \quad (2)$$

2.2 Markov chain Monte Carlo

MCMC methods are constructed based on a Markov chain that has the desired distribution as its equilibrium distribution. Due to the ability of MCMC that provides insight into large, complex Bayesian problems, it has been one of the most important developments in modern statistics [15]. The state of the chain after a large number of steps is then used as a sample of the desired distribution. The quality of the sample improves as a function of the number of steps. Typically, the formulation of a Markov chain with the desired properties is not an issue. However, researchers had risen up the issues on how to define the steps and how to minimize the number of steps required until it converges to an equilibrium distribution. The distribution is said to be in equilibrium state when the process develop gradually in a random way until it reaches a certain state, which it remains subsequently distribution [15]. As there is always some outstanding effect of the starting position, the MCMC sampling can only approximate the target distribution.

The MCMC sampling regularly used in calculating multi-dimensional integrals numerically, whereas a group of "walkers" moves around randomly. At each point where the walker steps, the integrand value at that point is calculated close to the integral. Then, a number of tentative steps around the area is taken, wherever a place with reasonably have high contribution to the integral being chose so that later on, they may move into the next step [15].

3. THE NETWORKS

3.1 Pi-Sigma Neural Network

Pi-Sigma Neural Network (PSNN) is a form of HONNs and was first introduced by Shin & Ghosh [16]. The mainspring behind the network is due to the fact that a polynomial of input variables is formed by a product ("pi") of several weighted linear combinations ("sigma") of input variables. That is why this network is called pi-sigma instead of sigma-pi. The PSNN exhibits fast learning while greatly reducing network complexity by utilising an efficient form of polynomials for many input variables. This special polynomial form helps the PSNN to dramatically reduce the number of weights in its structure. Fig. 1 shows the architecture of PSNN:

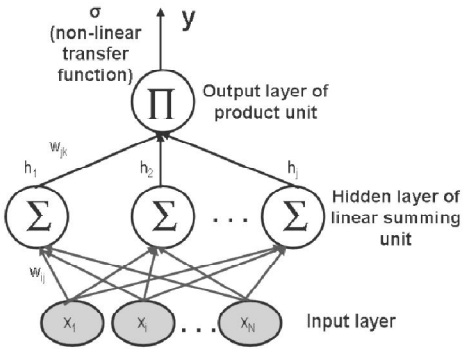

Figure 1: Structure of j^{th} Order PSNN

Input x is an N dimensional vector and x_i is the $_i{}^{\text{th}}$ component of x . The weighted inputs are fed to a layer of j

linear summing units; h_{ij} is the output if the j^{th} summing units for the k^{th} output y_k, viz:

$$y = \sigma\left(\Pi\left(\sum_j w_{ij}x_i + \theta_{ij}\right)\right) \quad (3)$$

where w_{ij} and θ_{ij} are adjustable coefficients, and σ is the nonlinear transfer function [16]. The number of the summing units in PSNN reflects the network order. By using an additional summing unit, it will increase the network's order by 1 at the same time as preserving old connections and maintaining network topology.

In PSNN, weights from summing layer to the output layer are fixed to unity, resulting to a reduction in the number of tuneable weights. Therefore, it can reduce the training time. Sigmoid and linear functions are adopted in the summing layer and output layer, respectively. The use of linear summing units makes the convergence analysis of the learning rules for the PSNN more accurate and tractable [17].

Since weights from hidden layer to the output are fixed at 1, the property of PSNN significantly reduces the training time. This network was successfully applied for function approximation [18], pattern recognition, classification, and many more.

3.2 Functional Link Neural Network

In forecasting, software growth effort using Functional Link Neural Network (FLNN) architecture which is flat ANNs involving of one input layer and an output layer. The FLNN is basically a single layer network and the layer of hidden layer was being detached and automatically produce the simplest learning algorithm compared to MLP [19]. The FLNN produces output through increasing the early inputs and then processing to the final output layer. Each input neuron corresponds to a component of an input vector. The output layer consists of one output neuron that computes the software development effort as a linear weighted sum of the outputs of the input layer [19]. The functional expansion efficiently growths the dimensionality of the input vector and later the hyper planes produced by the FLNN deliver better perception capability in the input pattern space.

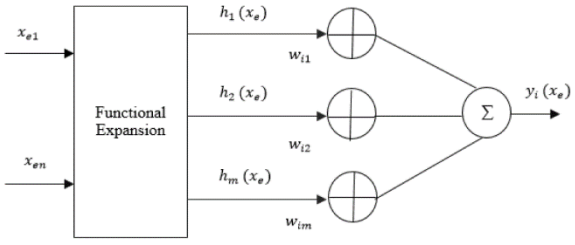

Figure 2: The Structure of a Functional Link Neural Network (FLNN)

The structure of an FLNN is presented in Fig. 2 where x_e is the input vector and $y_i(x_e)$ is the output.

The FLNN model apply a single-layer ANNs structure possessing network to overcome the higher computational load compare to the MLP structure, by expands their input vectors. The component of the input pattern before expansion:

$$x(i), 1 < i < d \quad (4)$$

where the component of $x(i)$ is functionally expanded as,

$$x_n(i), 1 < n < N \quad (5)$$

while N = number of expansion for every input component. Expansion of each input pattern is stated as below:

$$x_1(i) = z(i), x_2(i = f_1(z(i))), \ldots, x_N(i) = f_N(z(i)) \quad (6)$$

where, d: for the set of structures in the group of data.

The hidden layer implements a functional expansion on the input data, which maps the input space of dimension n_1 into a new space of increased dimension $M > n_1$. The output layer consists of m nodes, each one, in fact, a linear combiner. The input-output connection of the FLNN as derive as below:

$$y_i(x_e) = \sum w_{ij} h_j(x_e) M_j = 1, 1 \le i \le m \quad (7)$$

where,

x_e : input vector

$h_j(x_e)$: weights on connection from input unit

M : nodes

$y_i(x_e)$: output unit

Polynomial expansion is the most used in functional expansion technique. Due to the case, the expansion results, $h_j(x_e)$, are a sequence of monomials of x_e. Costa et al. [20] introduced an adjustment on the model of the FLNN, where the output set by Eq. (8) was transformed by an invertible nonlinear activation function. The new equation as stated as below:

$$y_i(x_e) = f_1 \sum w_{ij} h_j(x_e) M_j = 1, 1 \le i \le m \quad (8)$$

where f_1 is an invertible nonlinear function which is, the sigmoid function. This alteration was created to growth the non-linear approximation capability of the FLNN. The training of the algorithm was done using a changed output which is the original output changed through the inverse of the activation function, f [21].

In fact, these patterns of the expanded input are joining to the single layer ANNs to gain the desired output paragraph. For a certain mapping, the nonlinearity of the complex task may not be suitable for function expansion. Some of the set of function may not assimilate to the expansion of the input dataset. Nevertheless, in term of dimensionality problem, it will

produce very high and aggregate the dimensionality extends beyond expectation and resulting not a good choice [20].

Subsequently, it is preferable to select a minor group of substitute functions, which can plot the function to the preferred range. Therefore, Patra [21] has reserved an FLNN, via a flat network structure that competent to produce an arbitrarily complex result region using producing nonlinear decision limitations. In FLNN, the unseen layers have been removed to produce lower computational difficulty and higher convergence speed rate compared MLP structure.

4. MODIFIED CUCKOO SEARCH

As stated by Yang and Deb [10], the CS will always find the optimum if it been given enough computation. Basically, the search technique in CS being done by considering the whole area on random walks. However, it is not guaranteed whether the exploration can converged faster or not. Typically, the parameters α in CS are kept constant which resulting the efficiency of the algorithm to tail off. To cope with this issue, Walton et al. [4] created 2 modifications: 1) change the Lévy flight step size α . In ordinary CS, α was kept constant by employing $\alpha = 1$ [10] and 2) adding up the information exchange between the eggs in order to hasten up the convergence rate. Since the exploration in the ordinary CS is performed by their own selves, therefore, there is no information exchange between individuals. In the MCS, the eggs were evaluated twice. The first evaluation involves putting a sub of the eggs with the best fitness into a group of top eggs. For each of the top eggs, a second egg in this group is evaluated by picking up randomly before a new egg being generated on the line that connects these two top eggs. In order to get the best fitness, the new location of the new egg (distance along the line) is calculated by using the inverse of the golden ratio $\varphi = \left(1 + \sqrt{5}\right)/2$. If the same fitness value found in both eggs, the new egg is generated at the center point.

There are 2 parameters that need to be adjusted in the MCS, which refer to the fraction of nests to be abandoned and the fraction of nests to generate the top nests. An initial value of the Lévy flight step size $A = 1$ is chosen. At each generation, a new Lévy flight step is calculated using $\alpha \leftarrow A/\sqrt{G}$, where G specifies the number of generation. This exploration searching is only can be used for the fraction of nests to be abandoned. There is a probability that, in this measurement, the same egg is chosen twice. Therefore, by performing a local Lévy flight search on the randomly picked nest with step size $\alpha \leftarrow A/G^2$ can handle these problems.

5. MODIFIED CUCKOO SEARCH-MARKOV CHAIN MONTE CARLO ALGORITHM

The CS will always find the optimum if it been given enough computation [10]. Even though the searching process in CS is based on whole area on random walks, there is no assurance that the algorithm might converge faster as usual. Based on this problem, 2 modifications has been created in the MCS, with the aim to increase the convergence rate, that will make the method more practical for a wider range of application without losing the attractive features of original method [22].

Therefore, we extend the work of Walton et al. [10] by substituting the Lévy flight in the MCS with Markov chain Monte Carlo (MCMC) random walk. The motivation of using MCMC in the MCS algorithm is because the MCMC have the capability to performs full-dimensional jumps at each iteration. It also has higher polynomial rates of convergence due to the existence of central limit theorem (CLT) for higher moment. The MCMC is involved 2 parts. In this step, we apply the MCMC random walk due to the benefits: higher polynomial convergence rates due to the existence of CLT for higher moment [23]. This is because, MCMC use local moves based on certain types of target density thus leading to qualitatively better algorithms. The step-by-step processes in the MCMC are presented in Algorithm 1.

$A \leftarrow MaxLevyStepSize$

$\varphi \leftarrow GoldenRatio$

Initialise a population of n nests $x_i \left(i = 1,2,\ldots,n\right)$

FOR $\forall x_i$, do

Calculate fitness $F_i = f\left(x_k\right)$

ENDFOR

Generation number $G \leftarrow 1$

WHILE

$NumberObjectiveEvaluations < MaxNumberEvaluations$, do

$G = G + 1$

Sort nests by order of fitness

FOR all nests to be abandoned, do

Current position x_i

Calculate MCMC random walk

Perform MCMC random walk from x_i to generate new egg x_k

$x_i \leftarrow x_k$

$F_i \leftarrow f\left(x_k\right)$

ENDFOR

FOR all of the top nests do

Current position x_i

Pick another nest from the top nests at random x_j

IF $x_i = x_j$ then

Calculate MCMC random walk

Perform MCMC random walk from x_i, to generate new egg x_k, $F_k = f(x_k)$

Choose a random nest *l* from all nests

IF $(F_k > F_l)$ do

$x_l \leftarrow x_k$

$F_l \leftarrow F_k$

ENDIF

ELSE

$dx = |x_i - x_j| / \varphi$

Move distance dx from the worst nest to the best nest to find x_k, $F_k = f(x_k)$

Choose a random nest *l* from all nests

IF $(F_k > F_l)$ then

$x_l \leftarrow x_k$

$F_l \leftarrow F_k$

ENDIF

ENDIF

ENDFOR

ENDWHILE

Algorithm 1: MCS-MCMC

The MCS-MCMC is operated by the following steps [23]:

At first, we generate the initial value, θ that satisfies $f(\theta) > 0$, by considering the target probability density function (PDF):

$$p(\theta) = C.\theta^{-n/2}.\exp\left(\frac{-a}{2\theta}\right) \quad (9)$$

where $n = 5$ and $a = 4$.

There are 2 parameters, number of samples (iterations) and samples drawn from the target PDF, $p(\theta)$. For the number of samples, we set the value as the same as the dimension of each test function. Then, we marked-out U from a uniform distribution at random, and accept θ subject to $U \le P$ where $P < 1, \theta$.

For the procedure, we calculate the density ratio at the candidate point, θ and current points, θ_{t-1},

$$P = \frac{p(\theta)}{p(\theta_{t-1})} = \frac{f(\theta)}{f(\theta_{t-1})} \quad (10)$$

We can summarise that the sampling as first computing, and then accept the probability, P

$$P = \min \frac{f(\theta)}{f(\theta_{t-1})}, 1 \quad (11)$$

6. EXPERIMENTAL SETTINGS

We developed the standard MCS and MCS-MCMC algorithm in MATLAB. We then replaced the Lévy flight found in the standard MCS algorithm with MCMC random walk. Then, we run the code to get *d* dimensional points based on dimensionality of the benchmark test functions.

The experiment consists of 2 main runs. Firstly, we tested the standard MCS and MCS-MCMC to all benchmark test functions. During the experiments, 15 host nests with an egg survival probability of 0.25 were used. The maximum iterations for all the algorithms are set to 100 with a total of 30 simulation runs on each function. Average and standard deviation is used to find any variations in the average trial values. The less indicates the better.

Secondly, we apply the datasets into seven (7) different network architectures which is; standard PSNN, PSNN-MCS, PSNN-MCMC, standard FLNN, FLNN-MCS, FLNN-MCMC and standard MLP to see the performance on various ranges of data. Those models were trained and tested using time-series data; relative humidity, temperature and laser data collected from National Forecast Office, Malaysian Meteorological Department (MMD) and benchmark data from USCI. Each data signal was divided into 3 parts; 60 % training, 20% testing and 20% validation. Table 1 shows the number of data points used for each signal.

Table 1: Portion of Training, Testing and Validation Set

Dataset	Relative Humidity	Temperature	Laser
Training Set	30,504	1087	2384
Training Set	10,168	363	794
Validation Set	10,168	363	794
Total	50, 840	1813	3972

The training set aids the model for training purposes, so, it must be larger and adequate compared to two other datasets. The training encompasses the weights adjustment by testing the initial set of weights against each input vector. If an input vector is found for which the recognition fails, weights are adjusted to suit the certain input vector. During the training process, the actual and predicted outputs are compared and weights are adjusted by using the BP algorithm [24] (for PSNN, FLNN and MLP), MCS algorithm (for PSNN-MCS and FLNN-MCS) and MCS-MCMC algorithm (for PSNN-MCMC and FLNN-MCMC), to a satisfactory level with proper network setup.

Despite of that, the testing set is use to evaluate the network performances and to preserve some features for adjustment purpose [25], for the purpose of producing appropriate outputs for those input samples which were not encountered during training.

Meanwhile, the validation set has dual-function: 1) to implement an early stopping in order to prevent the training data from overfitting and 2) to select the best predictions from a number of ANNs's simulations.

7. BENCHMARK TEST FUNCTIONS

Test functions are important to validate and compare optimisation algorithms, especially newly developed algorithms. For this purpose, we compile thirteen (13) benchmark test functions with diverse properties in terms of modality, separability, and valley landscape [10]. Table 2 present the average and standard deviation of number of

function evaluation of both runs for all benchmark test functions.

Table 2: Comparison of Standard MCS and MCS-MCMC

Functions	Standard MCS	MCS-MCMC
Ackley's	0.62211 ± 0.18207	0.10891 ± 0.019736
Rosenbrock's	68.553 ± 93.7092	36.0144 ± 46.4826
Bohachevsky	5.06E-07 ± 0.00000258	0.000000624 ± 2.64E-08
Matyas	7.07E-09 ± 22.394E-09	1.00E-09 ± 2.6477E-09
Booth	5.11E-07 ± 1.82E-06	4.11E-07 ± 1.41E-06
Three-Hump Camel	5.01E-10 ± 20.461E-10	1.03E-10 ± 4.42E-10
Eggholder	-19.2085 ± 9.56E-08	-19.2085 ± 3.31E-08
Himmelblau	2.34E-07 ± 6.6002	3.84E-08 ± 0.00000710116
Schaffer N. 2	12.521E-11 ± 5.00E-10	2.98E-11 ± 1.37E-10
Styblinski-Tang	-78.3323 ± 2.71E-08	-78.3323 ± 1.82E-08
Rastrigin	1.82E-08 ± 4.29E-08	6.21E-09 ± 2.24E-08
Schwefel	735.1754 ± 235.1188	574.1585 ± 184.4498
McCormick	-1.9105 ± 8.64E-06	-1.9105 ± 6.87E-06

From all the test results tabularised in Table 2, the average functions evaluations of the MCS-MCMC are smaller than those obtained by the standard MCS. It indicates that the initialisation of MCS algorithm using MCMC had improved its searching steps. Using this result, the improvement also can be seen in the test result of the high dimensional input space functions such as the Ackley's 120-dimensions benchmark test function. For the Ackley's, the function evaluation average for MCS-MCMC outperformed standard MCS by 23.2% reduction.

To put the result into a more detailed perspective, Fig. 3 presents an example of the Fitness Value of Number of Generation for 120 dimensional Ackley's. As demonstrated in Fig. 3, the standard MCS (blue line) converge quickly at first 10-points but the MCS-MCMC (red line) converged quickly to its optimal solution. For Rosenbrock's, MCS-MCMC overtook standard MCS by 45.9% (refer to Fig. 4). Generally, the combination of MCS with MCMC had clearly leaded to better results in all benchmark test functions. It means that the fitness value is considerably closer to global optima.

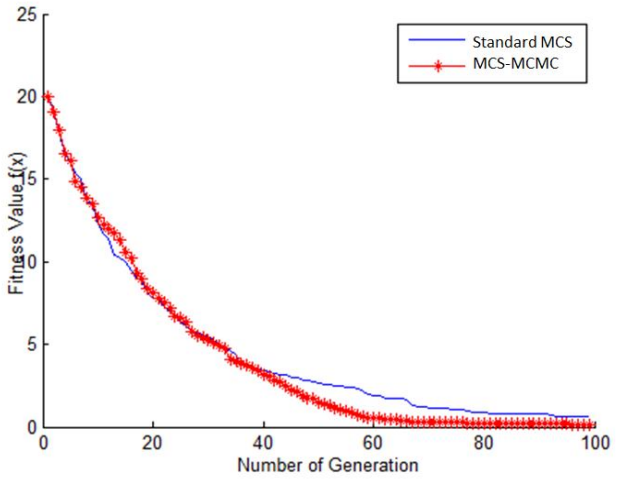

Figure 3: The Fitness Value of Number of Generation (Ackley, *d=120*)

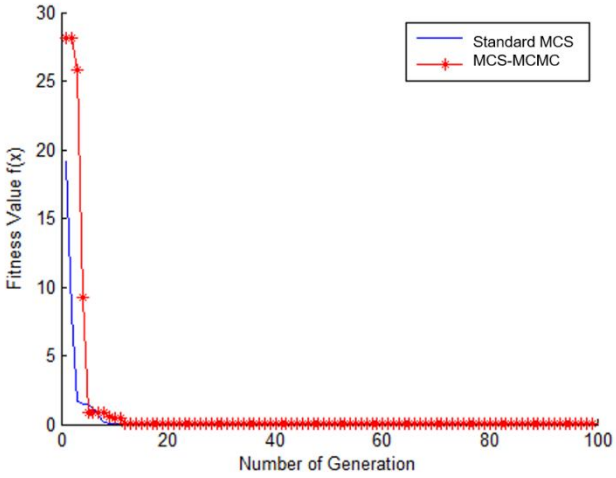

Figure 4: The Fitness Value of Number of Generation (Bohachevsky)

8. SIMULATION RESULTS

Designing the right architecture involves several steps: selecting the number of layers, the amount of neurons to be used in each layer and choosing the appropriate neurons' transfer function. Therefore, the parameter are set to be; the networks combination of five (5) different numbers of input nodes ranging from 5 to 7 [26], hidden layer/higher order terms from 2 until 5 and a single neuron for the output layer [17]. The performance of the MCS-MCMC algorithm are evaluated in 7 different network architectures which is; standard PSNN, PSNN-MCS, PSNN-MCMC, standard FLNN, FLNN-MCS, FLNN-MCMC and standard MLP.

Referring to Tables 3 to 5, the MSE results for relative humidity Inputs 5 to 7 are tabulated. According to the results, we can see that for Input 5, FLNN-MCMC, PSNN-MCMC and PSNN-MCS lead the ranking. For Input 6, PSNN-MCMC, PSNN-MCS and FLNN-MCMC lead the ranking. As for Input 7, the ranking goes to PSNN-MCMC, PSNN-MCS and FLNN-MCMC, respectively. Seemingly, based on the results,

the performances of the network in which the learning method had been replaced by MCMC algorithm are much preferable compared to the networks with standard MCS algorithm.

Table 3: MSE for Relative Humidity 5 Inputs

Networks	Rank	Iteration	MSE
PSNN	7	14	0.0475
PSNN-MCS	3	6	0.001621
PSNN-MCMC	2	5	0.001621
FLNN	6	91.2	0.0046
FLNN-MCS	4	100	0.00183
FLNN-MCMC	1	23	0.001323
MLP	5	21	0.0044

Table 4: MSE for Relative Humidity 6 Inputs

Networks	Rank	Iteration	MSE
PSNN	7	19	0.2263
PSNN-MCS	2	7	0.000606
PSNN-MCMC	1	6	0.000606
FLNN	6	19	0.0044
FLNN-MCS	4	95	0.000943
FLNN-MCMC	3	10	0.000859
MLP	5	23	0.0042

Table 5: MSE for Relative Humidity 7 Inputs

Networks	Rank	Iteration	MSE
PSNN	7	100	0.2421
PSNN-MCS	2	8	0.000486
PSNN-MCMC	1	7	0.000486
FLNN	6	156.6	0.0013
FLNN-MCS	4	96	0.001579
FLNN-MCMC	3	10	0.001296
MLP	5	150	0.0013

Fig. 5 graphically shows the performance comparison for all the networks on relative humidity for 5 inputs. According to the results, we clearly see that FLNN-MCMC shows the least MSE and minimum iteration compared to all networks generated or inversely, PSNN pointed the error to 0.0475 stopped at 14th iteration.

As indicated by the blue line in Fig. 6, PSNN-MCMC partaking the smallest minimum error 0.000606, in which the error is on the par with PSNN-MCS. From Fig. 7, FLNN-MCMC shows the least minimum error 0.000502 with 100 iterations. Based on these 2 figures, we can conclude that the presence of MCMC in the network assist in minimising the error, indirectly able to network to converge faster.

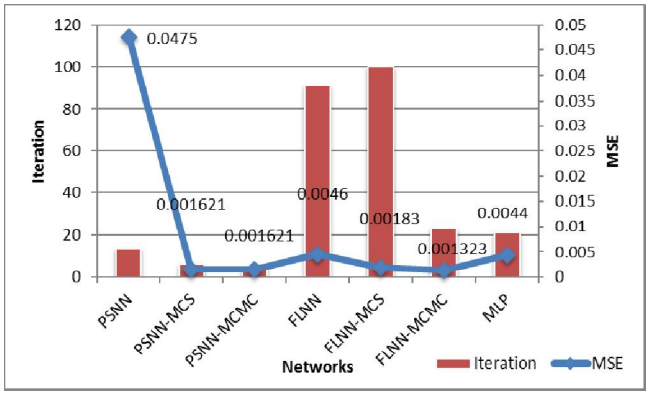

Figure 5: Performance Comparison on Relative Humidity for 5 Inputs

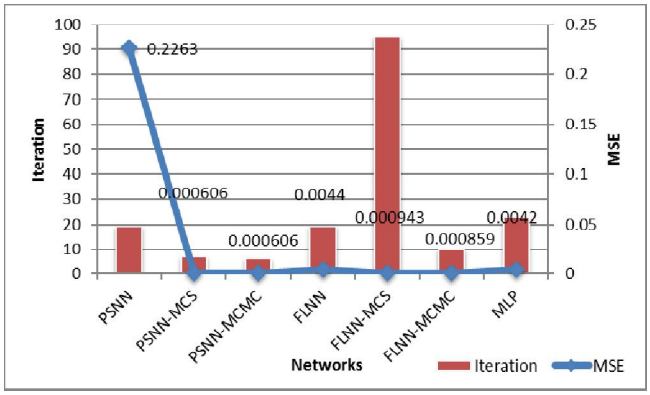

Figure 6: Performance Comparison on Relative Humidity for 6 Inputs

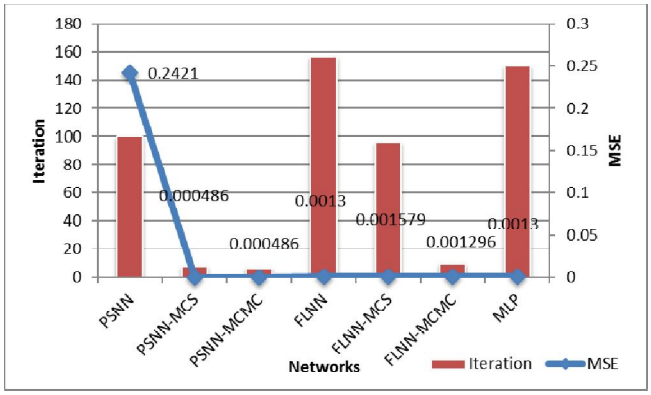

Figure 7: Performance Comparison on Relative Humidity for 7 Inputs

As can be noticed from Tables 6 to 8, the networks with the presence of MCMC algorithm (FLNN-MCMC and PSNN-MCMC) made the least MSE for ranges of inputs for temperature data when compared to other network models. As for example, for 5 inputs, FLNN-MCMC outperforms other network models by 93.67% while PSNN-MCMC takes 91.92%. All the networks show a very small degree of deviation from the means, as the indicator that they produce consistent behaviour. The largest value of error found in MLP with the value of 0.0061.

Table 6: MSE for Temperature 5 Inputs

Networks	Rank	Iteration	MSE
PSNN	4	100	0.0059
PSNN-MCS	7	100	0.006026
PSNN-MCMC	2	100	0.002551
FLNN	5	1000	0.0059
FLNN-MCS	3	100	0.003101
FLNN-MCMC	1	6	0.001999
MLP	6	551	0.0061

Table 7: MSE for Temperature 6 Inputs

Networks	Rank	Iteration	MSE
PSNN	5	100	0.0059
PSNN-MCS	4	100	0.00406
PSNN-MCMC	3	78	0.003975
FLNN	6	1000	0.0059
FLNN-MCS	2	99	0.002758
FLNN-MCMC	1	67	0.002162
MLP	7	390	0.006

Table 8: MSE for Temperature 7 Inputs

Networks	Rank	Iteration	MSE
PSNN	5	100	0.0059
PSNN-MCS	4	94	0.004903
PSNN-MCMC	3	98	0.003677
FLNN	7	1000	0.0061
FLNN-MCS	2	99	0.001066
FLNN-MCMC	1	100	0.000502
MLP	6	733	0.006

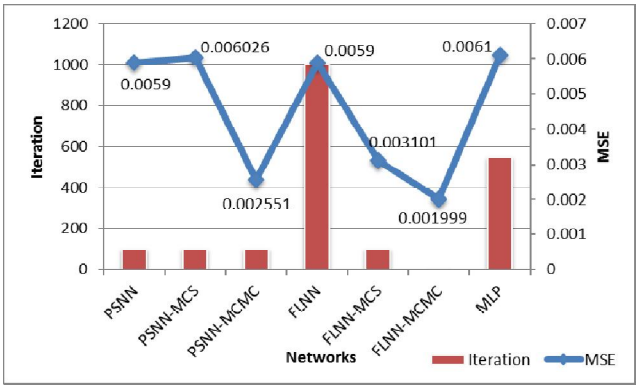

Figure 8: Performance Comparison on Temperature for 5 Inputs

Fig. 8 graphically shows the performance comparison for all the networks on temperature for 5 inputs. According to the results, the FLNN-MCMC indicates 0.001999 while PSNN-MCMC gives 0.002551 for the MSEs (blue line in Fig. 8). From this figure, we can roughly say, by plug-in the MCS-MCMC into both networks might help in minimising the error rate thus helping the network to converge faster.

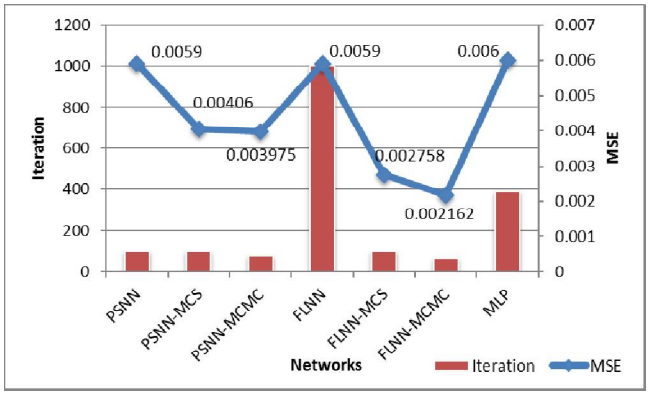

Figure 9: Performance Comparison on Temperature for 6 Inputs

The performance comparison for all the networks on temperature for 6 inputs are tabulated in Fig. 9. As it has been pointed out, FLNN-MCMC shows the least MSE, 0.002162 compared to all networks generated. Thus, by having the least MSE, it incorporates both the variance of the estimator and its bias on how far off the average estimated value is from the truth.

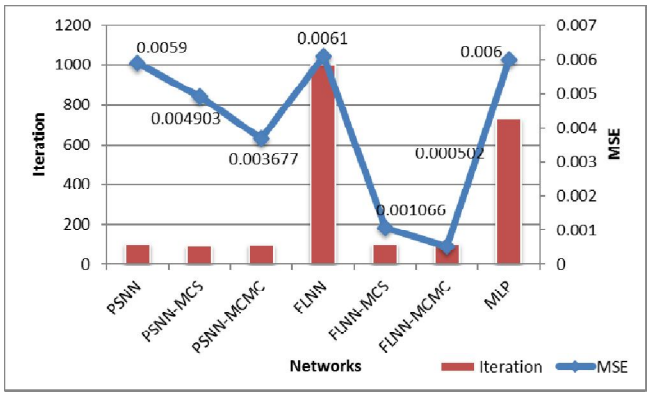

Figure 10: Performance Comparison on Temperature for 7 Inputs

Tables 9 to 11 shows that the average values of MSE for laser data for Inputs 5 to 7. Indubitably, MCS-MCMC has shown to learn the data with a comparable network size with FLNN and PSNN, with a smaller size when compared to other networks.

Table 9: MSE for Laser 5 Inputs

Networks	Rank	Iteration	MSE
PSNN	6	100	0.0082
PSNN-MCS	5	97	0.006033
PSNN-MCMC	3	9	0.005118
FLNN	7	1000	0.0082
FLNN-MCS	2	97	0.001761
FLNN-MCMC	1	97	0.001072
MLP	4	211	0.0059

Table 10: MSE for Laser 6 Inputs

Networks	Rank	Iteration	MSE
PSNN	6	67	0.008
PSNN-MCS	7	94	0.008745
PSNN-MCMC	3	6	0.00798
FLNN	4	1000	0.0069
FLNN-MCS	2	96	0.000748
FLNN-MCMC	1	96	0.000428
MLP	5	99	0.0073

Table 11: MSE for Laser 7 Inputs

Networks	Rank	Iteration	MSE
PSNN	5	64	0.0073
PSNN-MCS	7	10	0.009606
PSNN-MCMC	6	5	0.009606
FLNN	4	1000	0.0061
FLNN-MCS	2	100	0.001699
FLNN-MCMC	1	13	0.000669
MLP	3	585	0.0046

Fig. 11 to 13 depict the performance comparison for all the networks on laser data inputs ranging from 5 to 7. Throughout the results, the FLNN-MCMC shows the least MSE for the 3 input combinations. This way, we can iterate that instead of that the MSE are able to assess the quality of sample data to an estimate of a function mapping arbitrary inputs to a sample of values of some random variables.

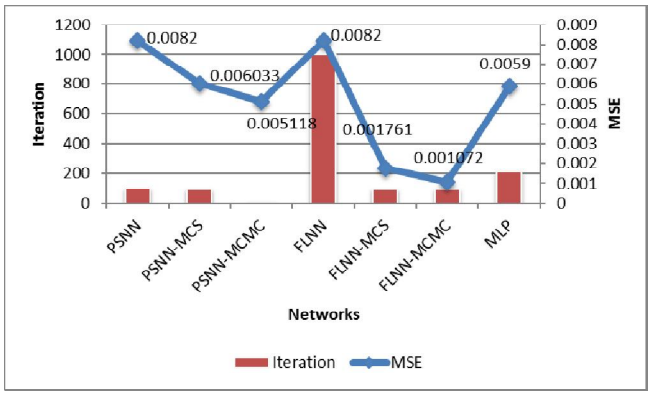

Figure 11: Performance Comparison on Laser for 5 Inputs

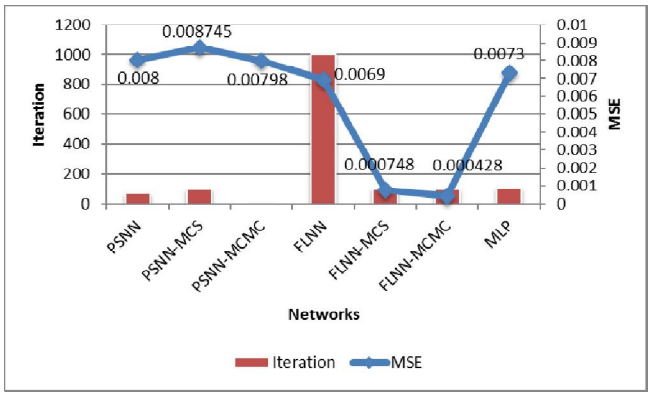

Figure 12: Performance Comparison on Laser for 6 Inputs

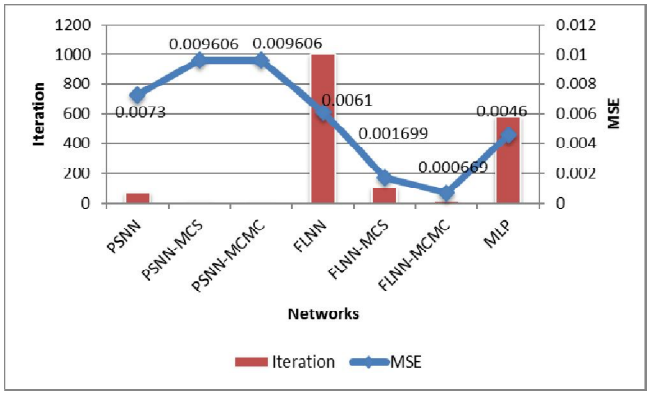

Figure 13: Performance Comparison on Laser for 7 Inputs

Table 12. Overall Rank on all Networks

Data	Inputs	PSNN	PSNN-MCS	PSNN-MCMC	FLNN	FLNN-MCS	FLNN-MCMC	MLP
Relative Humidity	5	7	3	2	6	4	1	5
	6	7	2	1	6	4	3	5
	7	7	2	1	6	4	3	5
Temperature	5	4	7	2	5	3	1	6
	6	5	4	3	6	2	1	7
	7	5	4	3	7	2	1	6
Laser	5	6	5	3	7	2	1	4
	6	6	7	3	4	2	1	5
	7	5	7	6	4	2	1	3
Mean Rank		5.9	4.6	2.7	5.7	2.8	1.4	5.1
Overall Rank		6	7	3	6	3	1	5

Referring to Table 12, it is plain to see that, on average, FLNN-MCMC performed better compared to the other networks. This is then followed by PSNN-MCMC. Consequently, we may conclude that the existence of MCS-MCMC algorithm as the learning algorithm might help the network to converge faster and reduce the error rate.

9. CONCLUSION AND FUTURE WORKS

The current research includes trials of MCS with MCMC on various benchmark test functions. From the outcomes, it is proven that, in this research, it is affirmative that the networks with MCS-MCMC generalised well and showed least error compared to other network, which is able in representing nonlinear function. The presence of MCS-MCMC as the learning algorithm that substitutes the current BP learning algorithm, allowed fast and rapid training. A significant advantage of the MCS-MCMC is the fact that the learning algorithm can tune better parameter automatically in finding good parameter values with little user intervention. This process can be achieved by Markov chain mixing and integrated autocorrelation of a functional of interest.

A further extension of the MCS-MCMC algorithm needs to address multiobjective optimisation problems more naturally and more efficiently instead of focusing on the optimisation with a single objective or a few criteria with linear and nonlinear constraints.

ACKNOWLEDGEMENT

The authors would like to thank Universiti Tun Hussein Onn Malaysia (UTHM) and Ministry of High Education (MOHE) for financially supporting this research under Fundamental Research Grant Scheme (FRGS), Vote No 1641.

REFERENCES

[1] Collobert, R. and J. Weston, *A unified architecture for natural language processing: deep neural networks with multitask learning*, in *Proceedings of the 25th international conference on Machine learning*. 2008, ACM: Helsinki, Finland.

[2] Rumelhart, D.E., B. Widrow, and M.A. Lehr, *The basic ideas in neural networks.* Commun. ACM, 1994. **37**(3): p. 87-92. https://doi.org/10.1145/175247.175256

[3] Wang, H., et al., *Development of metamodeling based optimization system for high nonlinear engineering problems.* Advances in Engineering Software, 2008. **39**(8): p. 629-645.

[4] Walton, S., et al., *Modified cuckoo search: A new gradient free optimisation algorithm.* Chaos, Solitons & Fractals, 2011. **44**(9): p. 710-718.

[5] Shah, H., R. Ghazali, and N. Mohd Nawi. *Hybrid ant bee colony algorithm for volcano temperature prediction.* in *International Multi Topic Conference*. 2012: Springer.

[6] Leung, H. and S. Haykin, *The complex backpropagation algorithm.* IEEE Transactions on Signal Processing, 1991. **39**(9): p. 2101-2104.

[7] Yadav, A.K. and S.S. Chandel, *Solar radiation prediction using Artificial Neural Network techniques: A review.* Renewable and Sustainable Energy Reviews. **33**: p. 772-781. https://doi.org/10.1016/j.rser.2013.08.055

[8] Basheer, I.A. and M. Hajmeer, *Artificial neural networks: fundamentals, computing, design, and application.* Journal of Microbiological Methods, 2000. **43**(1): p. 3-31.

[9] Law, R., *Back-propagation learning in improving the accuracy of neural network-based tourism demand forecasting.* Tourism Management, 2000. **21**(4): p. 331-340.

[10] Yang, X.S. and S. Deb. *Cuckoo search via Levy flights*. in *Proceedings of the World Congress on Nature & Biologically Inspired Computing (NaBIC '09*. 2009. India: IEEE Publications.

[11] Pearson, K., *The problem of the random walk.* Nature, 1905. **72**(1865): p. 294.

[12] Rogel-Salazar, J., *First Steps in Random Walks: From Tools to Applications, by J. Klafter and I.M. Sokolov.* Contemporary Physics, 2012. **53**(4): p. 369-370.

[13] Bergman, C.M., J.A. Schaefer, and S. Luttich, *Caribou movement as a correlated random walk.* Oecologia, 2000. **123**(3): p. 364-374.

[14] Mandelbrot, B.B., *The Fractal Geometry of Nature*. Updated and augm. ed. 1982, New York: W. H. Freeman.

[15] University, L. *Markov Chain Monte Carlo*. Available from: http://www.lancaster.ac.uk/pg/jamest/Group/stats3.html.

[16] Shin, Y. and J. Ghosh, *The Pi-Sigma Networks: An Efficient Higher-Order Neural Network for Pattern Classification and Function Approximation.* Proceedings of International Joint Conference on Neural Networks, 1991. **1**: p. 13-18.

[17] Husaini, N.A., et al. *The effect of network parameters on pi-sigma neural network for temperature forecasting*. in *International Journal of Modern Physics: Conference Series*. 2012: World Scientific. https://doi.org/10.1142/S2010194512005521

[18] Ghazali, R. and D. al-Jumeily, *Application of Pi-Sigma Neural Networks and Ridge Polynomial Neural Networks to Financial Time Series Prediction*, in *Artificial Higher Order Neural Networks for Economics and Business*, M. Zhang, Editor. 2009, IGI Global: Hershey, New York. p. 271-293.

[19] Pao, Y.H. and Y. Takefuji, *Functional-link net computing: theory, system architecture, and functionalities.* Computer, 1992. **25**(5): p. 76-79.

[20] Costa, A.C., et al., *A Hybrid Neural Model for he Optimization of Fed-batch Fermentetions.* Brazilian Journal of Chemical Engineering, 1999. **16**: p. 53-63.

[21] Patra, J.C. and R.N. Pal, *A functional link artificial neural network for adaptive channel equalization.* Signal Processing, 1995. **43**(2): p. 181-195.

[22] Walton, S., et al., *Modified cuckoo search: A new gradient free optimisation algorithm.* Chaos, Solitons & Fractals, 2011. **44**(9): p. 710-718.

[23] Husaini, N.A., R. Ghazali, and I.T.R. Yanto. *Enhancing modified cuckoo search algorithm by using MCMC random walk*. in *Science in Information Technology (ICSITech), 2016 2nd International Conference on*. 2016: IEEE.

[24] Rumelhart, D.E., G.E. Hinton, and R.J. Williams, *Learning Representations by Back-Propagating Errors.* Nature, 1986. **323**(9): p. 533-536.

[25] Shrestha, R.R., S. Theobald, and F. Nestmann, *Simulation of Flood Flow in a River System using Artificial Neural Networks.* Hydrology and Earth System Sciences, 2005. **9**(4): p. 313-321.

[26] Lendasse, A., et al., *Non-linear Financial Time Series Forecasting - Application to the Bel 20 Stock Market Index.* European Journal of Economic and Social Systems, 2000. **14**(1): p. 81-91. https://doi.org/10.1051/ejess:2000110