Hybrid Bat & Levenberg-Marquardt Algorithms for Artificial Neural Networks Learning

Nazri Mohd Nawi[1], M. Z. Rehman[1], Abdullah Khan[1], Arslan Kiyani[1], Haruna Chiroma[2] and Tutut Herawan[2]

[1]Software and Multimedia Centre
Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia
86400 Parit Raja, Batu Pahat, Johor, Malaysia
[2]Faculty of Computer Science and Information Technology
University of Malaya
50603 Lembah Pantai, Kuala Lumpur, Malaysia

nazri@uthm.edu.my, zrehman862060@gmail.com, hi100010@siswa.uthm.edu.my, arslan.kiyani@gmail.com, hchiroma@acm.org, tutut@um.edu.my

Abstract: The Levenberg-Marquardt (LM) gradient descent algorithm is used extensively for the training of Artificial Neural Networks (ANN) in the literature, despite its limitations, such as susceptibility to the local minima that undermine its robustness. In this paper, a bio-inspired algorithm referring to the Bat algorithm was proposed for training the ANN, to deviate from the limitations of the LM. The proposed Bat algorithm-based LM (BALM) was simulated on 10 benchmark datasets. For evaluation of the proposed BALM, comparative simulation experiments were conducted. The experimental results indicated that the BALM was found to deviate from the limitations of the LM to advance the accuracy and convergence speed of the ANN. Also, the BALM performs better than the back-propagation algorithm, artificial bee colony trained back-propagation ANN, and artificial bee colony trained LM ANN. The results of this research provide an alternative ANN training algorithm that can be used by researchers and industries to solve complex real-world problems across numerous domains of applications.
Keywords: Bat algorithm; Levenberg-Marquardt Algorithm; Artificial Neural Networks; Training.

1. Introduction

Artificial Neural Networks' (ANN) processing elements translate the synaptic behaviour of neurons in the human nervous system to the mathematical form [1-2]. The ANN consists of a large number of interrelated processing components known as neurons that function together to solve complex real-world problems [3]. The ANN have been implemented successfully in engineering fields such as biological modelling, decision and control, health and medicine, manufacturing, marketing, ocean exploration, etc. [4-9]. A back-propagation ANN (BPNN) is a method for training a multilayer feed-forward ANN [10-11]. However, the BPNN algorithm suffers from two major drawbacks: i.e. low convergence rate and instability, which lead the ANN towards local minima [12-14]. In the past decade, several new algorithms have been proposed to overcome the problems of gradient descent-based systems. These algorithms include a direct enhancement method, using a polytope algorithm [14], a global search procedure, such as evolutionary programming [15], and genetic algorithm (GA) [16]. The standard gradient-descent BPNN is not path-driven, but population-driven. However, the improved learning algorithms have explorative search topographies. Therefore, these approaches are expected to avoid local minima by promoting exploration of the search space. The Stuttgart Neural Network Simulator (SNNS) [17], which was developed in the recent past, used many different algorithms, including error back-propagation [13], quick prop algorithm [18], resilient error back-propagation [19], back-propagation, delta-bar-delta, cascade correlation [20] etc.

All these algorithms are derivatives of steepest gradient search, thus ANN training was found to be relatively slow. For fast and efficient training, second-order learning algorithms have to be used.

The most effective method is the Levenberg-Marquardt (LM) algorithm [21], which is a derivative of the Newton method [22]. LM is a multidimensional algorithm since not only the gradient, but also the Jacobian matrix should be computed. This ranks LM as one of the most efficient algorithms for small- and medium-sized patterns. As such, LM is considered as one of the most successful algorithm in increasing the convergence speed of the ANN with multi layered perceptron (MLP) architecture [23]. LM inherits speed from the Newton method and the convergence ability of the steepest descent, therefore it converges quickly on quadratic surfaces. Despite the robustness of the LM over other gradient-descent training algorithms, the LM is not able to avoid local minima in cases of complex surfaces [25-27].

In order to overcome the slow convergence and local minima problems, this paper proposed a new algorithm that combines the Bat [28] and Levenberg-Marquardt (LM) algorithms (BALM). The proposed BALM algorithm is compared with conventional BPNN, Artificial Bee Colony (ABC), BPNN (ABC-BP), and ABC-LM algorithms on 10 classification datasets.

The next two sections explain briefly the Bat algorithm, followed by the proposed BALM algorithm. In Section 3, the modification of the Bat is presented. Section 4, the performance of the proposed BALM on experimental datasets is discussed. The paper is finally concluded in Section 5.

2. Bat Algorithm

Bat is a meta-heuristic optimization algorithm developed by Yang [28]. The Bat algorithm is based on the echolocation behaviour of microbats, with varying pulse rates of emission and loudness. Yang [28] idealized the following rules to model Bat algorithm:

a. All bats use echolocation to sense distance, and they also 'know' the difference between food/prey and background barriers in some magical way.
b. A bat flies randomly with velocity (v_i) at position (x_i) with a fixed frequency ($f_{\min}$), varying wavelength λ and loudness A_0 to search for prey. It can automatically adjust the wavelength (or frequency) of its emitted pulses and adjust the rate of pulse emission $r \in [0,1]$, depending on the proximity of its target.
c. Although the loudness can vary in many ways, Yang [28] assumes that the loudness varies from a large (positive) A_0 to a minimum constant value A_{min}.

The pseudo-code for the Bat algorithm is shown in Figure 1 [28];

Objective function $f(x)$, $x = x_1, \dots, x_d{}^T$
Initialize the bat population x_i $(i = 1,2, \dots, n)$ *and* v_i
Define pulse frequency f_i *at* x_i
Initialize pulse rates r_i *and the loudness* A_i
while $(t <$ *Max number of iterations)*
Generate new solutions by adjusting frequency,
and updating velocity and locations/ solutions
    if $(rand > r_i)$
    Select a solution among the best solutions
    Generate a local solution around the selected best solution
    end if
    Generate a new solution by flying randomly
    if $(rand < A_i \,\&\, f(x_i) < f(x_*))$
    Accept the new solutions
    Increase r_i *and decrease* A_i
    end if
    Rank the Bats and find the current best x_*
end while
post process results and visualization

Figure 1 Pseudo-code of the Bat Algorithm by Yang [28]

Bat is a population-based optimization algorithm, and like other meta-heuristic algorithms, it starts with a random initial population. Figure 1 shows that in the Bat algorithm, each virtual bat flies

randomly with a velocity v_i at some position x_i, with a varying frequency f_i and loudness A_i, as explained in the previous Section. As it searches and finds its prey, it changes frequency, loudness and pulse emission rate r_i. Search is intensified by a local random walk. Selection of the best continues until stopping criteria are met. To control the dynamic behaviour of a colony of bats, the Bat algorithm uses a frequency-tuning technique, and searching and usage are controlled by changing the algorithm-dependent parameters [28-31, 40-41].

3. The Proposed BALM Algorithm

The flow diagram of the proposed BALM algorithm is shown in Figure 2.

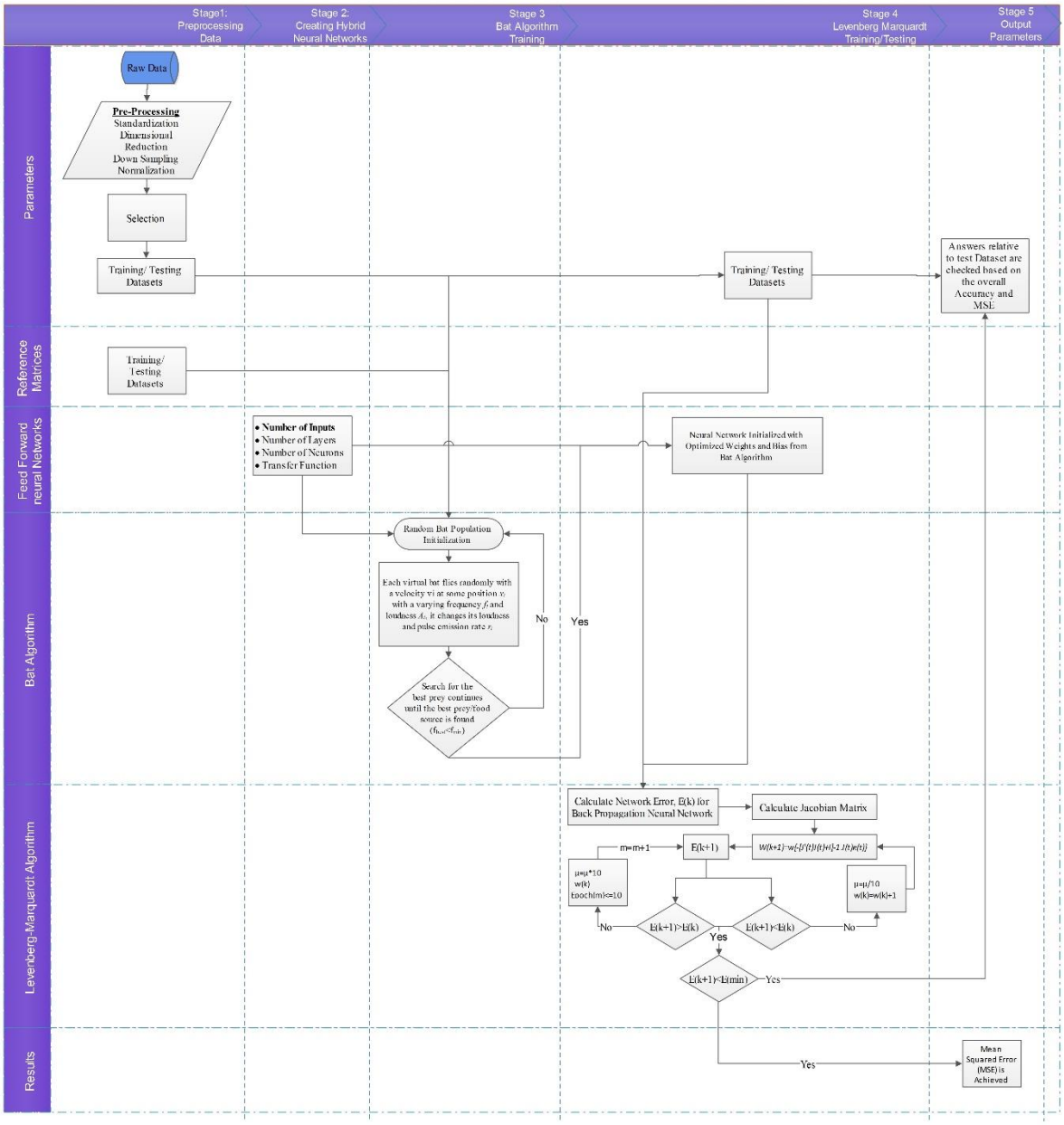

Figure 2 Proposed BALM algorithm

In Figure 2, each position represents a possible solution (i.e.; the weight space and the corresponding biases for LM optimization). The weight-optimization problem and the position of a food source represents the quality of the solution. In the first epoch, the best weights and biases are

initialized with Bat, and then those weights are passed on to the BPNN. The weights in BPNN are calculated and then passed on to the Levenberg-Marquardt (LM) algorithm. The main idea of this combinatorial algorithm is that the Bat algorithm is used at the initial stage of searching for the optimum to select the best initial weights. Then the training process is continued with the LM algorithm, using the best weights from the Bat algorithm. In the next cycle, Bat again updates the weights with the best possible solution, and Bat continues to pass the best weights to LM, until either the last cycle/epoch of the network is reached, or the mean square error (MSE) is achieved.
In the proposed BALM algorithm, the MSE for each weight matrix consists of all input pattern matrix through LM ANN. The MSE is considered as a performances index for the proposed BALM algorithm. The weight value of a matrix is computed as follows:

$$W_c = \sum_{c=1}^{m} a.(rand - \frac{1}{2}) \quad (1)$$

$$B_c = \sum_{c=1}^{m} a.(rand - \frac{1}{2}) \quad (2)$$

Where $W_c = c^{th}$ weight value in a weight matrix. The $rand$ in the Equation (1) is the random number having value between [0 1], a is any constant parameter having a value less than one and B_c is the bias value. So, the list of weight matrix is as follows;

$$W^s = [{W_c}^1, {W_c}^2, {W_c}^3, \ldots\ldots {W_c}^{n-1}] \quad (3)$$

From BPNN, MSE is easily calculated for every weight matrix in W^s.The net input to the unit i in layer j is expressed as:

$$y_i = f(\sum_{j=1}^{N} W_{c(i\,j)} a_j + B_{cj}) \quad (4)$$

The net output of m unit for the output layer can expressed as:

$$X_m = f(\sum_{m=1}^{M} W_{c(j\,m)} y_i + B_{cm}) \quad (5)$$

Where, X_m is network output, f is transfer function, $W_{c(j\,m)}$ represents weights matrix, and y_i is the net output from neuron. The task of the network is to learn association between a specified set of input-output pairs, $\{(a_{1,}T_{1,}), (a_{2,}T_2), (a_{3,}T_{3,})\ldots (a_{r,}T_r)\}$. The error can be computed as:

$$e_r = (T_r - X_r) \quad (6)$$

The performance index for the network is calculated using the following Equations (7-8)

$$V_t(x) = \frac{1}{2}\sum_{r=1}^{R}(T_r - X_r)^T(T_r - X_r) \quad (7)$$

$$V_F(x) = \frac{1}{2}\sum_{r=1}^{R} {e_r}^T . e_r \quad (8)$$

In the proposed BALM algorithm, the average MSE is considered as the performance index computed based on Equation (9).

$$V_\mu(x) = \frac{\sum_{j=1}^{N} V_F(x)}{P_i} \quad (9)$$

Where, y_r is the output of the network when the r^{th} input to a_r is presented. And $e_r = (T_r - X_r)$ is the error for the r^{th} input, $V_\mu(x)$ is the average performance, $V_F(x)$ is the performance index, and P_i is the number of bats in i^{th} iteration. The weights and bias are calculated according to the back propagation method. The sensitivity of one layer is calculated from the previous one and the

calculation of the sensitivity start from the last layer of the network and moves backward. To speed up convergence, LM is selected as the learning algorithm. The LM algorithm is an approximation to Newton's method to get faster training speed. Assume the error function is expressed as:

$$E(t) = \frac{1}{2}\sum_{i=1}^{N} {e_r}^2(t) \tag{10}$$

Where, $e(t)$ is the error; N is the number of vector elements, and $E(t)$ is the MSE function, then the gradient is calculated as:

$$\nabla E(t) = J^T(t)e(t) \tag{11}$$

$$\nabla^2 E(t) = J^T(t)J(t) \tag{12}$$

Where, $\nabla E(t)$ is the gradient; $\nabla^2 E(t)$ is theHessian matrix of E (t); and $J\ (t)$ is the Jacobin matrix which is calculated in Equation (13);

$$J(t) = \begin{bmatrix} \frac{\partial e_1(t)}{\partial t_1} & \frac{\partial e_1(t)}{\partial t_2} & \cdots & \frac{\partial e_1(t)}{\partial t_n} \\ \frac{\partial e_2(t)}{\partial t_1} & \frac{\partial e_2(t)}{\partial t_2} & \cdots & \frac{\partial e_2(t)}{\partial t_n} \\ & \vdots & & \\ \frac{\partial e_n(t)}{\partial t_1} & \frac{\partial e_n(t)}{\partial t_2} & \cdots & \frac{\partial e_n(t)}{\partial t_n} \end{bmatrix} \tag{13}$$

For Gauss-Newton Method:

$$\nabla w = -[J^T(t)J(t)]^{-1}J(t)e(t) \tag{14}$$

For the LM as the variation of Gauss-Newton Method:

$$w(k+1) = w(k) - [J^T(t)J(t) + \mu I]^{-1}J(t)e(t) \tag{15}$$

Where $\mu > 0$ and is a constant; I is identity matrix, so that the algorithm can approach Gauss-Newton, which should provide faster convergence. Note that when parameter μ is large, the above expression approximates gradient descent (with learning rate $1/\mu$) while for a small μ, the algorithm approximates the Gauss-Newton method. The LM is an enhancement to BPNN algorithm which is calculated according to the following steps:

a. Present all inputs to the network and compute the corresponding network outputs and errors using Equations (5-6) over all inputs. Compute the sum of square of error over all input.
b. Compute the Jacobin matrix using Equation (13).
c. Solve Equation (12) to obtain ∇w.
d. Re-compute the sum of squares of errors using Equation (15), if this new sum of squares is smaller than the computed sum of square in Step 1, then reduce $\mu$by λ=10, update weight using $w(k+1) = w(k) - \nabla w$ and go back to Step 1. If the sum of squares is not reduced, then increase μ by λ=10, and go back to Step 3.
e. The algorithm is assumed to have converged when the norm of the gradient in Equation (11) is less than some prearranged value, or when the sum of squares has been compact to some error goal.

At the end of each epoch the list of average sum of squared error of i^{th} iteration MSE can be calculated as:

$$MSE_i = \{V_\mu(x_1), V_\mu(x_2), V_\mu(x_3) \ldots . V_\mu(x_n)\} \tag{16}$$

The bat search is imitating the minimum MSE and it is found when all the input is processed for each population of the bat. Thus, the bat swarm x_j is calculated as:

$$x_j = Min\{V_\mu(x_1), V_\mu(x_2), V_\mu(x_3) \ldots . V_\mu(x_n)\} \quad (17)$$

The rest of the average sum of square is considered as other bats. A new solution ${x_i}^{t+1}$ for Bat i is generated using Equation (18).

$${x_i}^{t+1} = {x_i}^{t-1} + v_i^t \quad (18)$$

For each time step t, the movement of the virtual bats is given by updating their velocity v_i and frequency, f_i using Equations (19-20):

$$f_i = f_{min} + (f_{max} - f_{min})\beta \quad (19)$$

$$v_i^t = v_i^{t-1} + \left(x_i^{t-1} - x_*\right)f_i \quad (20)$$

Where β denotes a randomly generated number within the interval [0,1], x_i^t denotes the value of decision variable j for bat i at time step t. The result of f_i in Equation (19) is used to control the pace and range of the movement of the bats. The variable x_* represents the current global best solution which is located after comparing all the solutions among all the *n* bats.
The movement of the bats x_i towards x_j can be drawn from Equation (21-22):

$$\begin{aligned} &if\ rand[0,1] > r_i \\ &V = x_i + rand \cdot \left(x_j - x_i\right) \\ &if\ rand[0,1] < A_i\ \&\&\ f_{(x_i)} < f_{(x_*)}\ , \end{aligned} \quad (21)$$

The bats can move from x_i toward x_j randomly

$$\nabla V_{\mathrm{i}} = x_i + v_i^t \sim 0.01\ .(V - X_{best}) \quad (22)$$

Increase r_i and decrease A_i.
Where *A*t stands for the average loudness of all the bats at time t, and $\epsilon \in [-1, 1]$ is a random number. For each iteration of the algorithm, the loudness Ai and the emission pulse rate ri are updated, as follows:

$$A_i^{t+1} = \propto A_i^t \quad (23)$$

$$r_i^{t+1} = r_i^0[1 - exp(-\gamma t)] \quad (24)$$

Where α and γ are constants. At the first step of the algorithm, the emission rate, r_i^0 and the loudness, A_i^0 are often randomly chosen. Generally, $A_i^0 \epsilon [1,2]$ and $r_i^0 \epsilon [0,1]$. ∇V_{i} is a small movement of x_i towards x_j. The weights and biases for each layer is then adjusted as:

$${W_c}^{n+1} = {W_c}^n - \nabla V_{\mathrm{i}} \quad (25)$$

$${B_c}^{n+1} = {B_c}^n - \nabla V_{\mathrm{i}} \quad (26)$$

The pseudo-code for the proposed BALM is given in Figure 3.

1. Initialize Bat population size and LM structure.
2. Load the training data (i.e. Inputs and the Label Class).
3. **While** MSE<stopping criteria.
4. Pass the best solutions calculated by bats as best weights to network.
5. Feed-forward network runs using the weights initialized with Bat.
6. Calculate the error using Equation (6).
7. Calculate the minimum error using Equation (9) and store the best nest as (weight) for the network.
8. Present all inputs to the network with the stored best nest as (weight), and compute the corresponding network outputs and errors using Equations (5) and (6) over all inputs. Compute sum of square of error over all input.
9. The sensitivity of one layer is calculated from its previous one, and the calculation of the sensitivity starts from the last layer of the network and moves backward.
10. Compute the Jacobin matrix using Equation (13).
11. Solve Equation (14) to obtain ∇w .
12. Recompute the sum of squares of errors using Equation (15). If this new sum of squares is smaller than that computed in Step 8, then reduce μ by λ=10, update weight using $w(k+1) = w(k) - \nabla w$ and go back to Step 8. If the sum of squares is not reduced, then increase μ by λ=10, and go back to Step 11.
13. The algorithm is assumed to have converged when the norm of the gradient in Equation (11) is less than some pre-arranged value, or when the sum of squares has been compacted to some error goal.
14. Minimize the error by adjusting network parameters using Bat.
15. Generate new bats (x_j) randomly.

$$V_i = x_j$$

16. Build new solution using Equation (18) to replace the old ones.
17. Bat keeps on calculating the best possible weight at each epoch until the network is converged.
18. **End While.**

Figure 3 Proposed BALM pseudo-code

4. Results and Discussions

Experimental set-up

In order to illustrate the performance of the proposed algorithm, BALM is trained on 10 datasets. The simulation experiments were performed on an Intel Core i5 processor and 8 GB of RAM, using MATLAB 2012 software. The proposed BALM algorithm was compared with the state of the art ABC-LM, ABC-BP and BPNN algorithms based on the MSE and the number of epochs. The maximum number of epochs and MSE were set to 1000 and 0.00001 respectively. The network stops when the target MSE was achieved or after the maximum number of epochs was reached [39]. The three-layer feed-forward neural networks are used for each problem: i.e. input layer, one hidden layer, and output layers. The number of hidden nodes comprises five neurons. In the network structure the bias nodes are also used and the log sigmoid activation function is placed as the activation function for the hidden and output layers. On each algorithm 20 trials are repeated. For computing the relative accuracy improvement of the proposed BALM algorithm with respect to BPNN, ABC-BP, and ABC-LM i.e.; how much better BALM performs in terms of accuracy the following formula is used [32]. For all experiments, best tuning parameters indicated by Yang [28] were used for Bat. The default parameters on which Bat performs best were loudness A=0.5, pulse rate = 0.5 and population size = 20 [40-41].

Two-Bit Exclusive-OR Problem

The first test problem is the exclusive-OR (XOR) which is a Boolean function of two binary inputs to a single binary output. In the simulations we used a 2–5–1, feed-forward neural network structure for a two-bit XOR problem. Table 1 shows the CPU time, number of epochs, accuracy, and MSE for the two-bit XOR test data set with five hidden neurons. From Table 1 it can be clearly seen that the proposed BALM algorithm converged to the global minima within 563 epochs, with an MSE of 7.08E-5, which is accurate up to 5 decimal points. Also, the BALM algorithm shows a 3.52 percent improvement in accuracy during convergence. Meanwhile, the ABC-BP algorithm falls behind with

an accuracy of 96.47 and an MSE of 2.39E-4. Although the BALM algorithm took more CPU time to converge, it achieved better MSE and accuracy than the comparison algorithms. Figure 4 illustrates the convergence performance of the BALM, ABC-BP, ABC-LM, and conventional BPNN algorithms.

Table 1 Comparison results in terms of CPU time, epochs, MSE and accuracy for 2-Bit XOR dataset

Algorithm	BPNN	ABC-BP	ABC-LM	BALM	Improvement
CPU Time	42.64	172.34	123.95	241.32	53 %
EPOCHS	1000	1000	1000	563	78 %
MSE	0.2206	2.39E-4	0.125	7.08E-5	162724 %
Accuracy	54.61	96.47	71.69	99.99	25 %

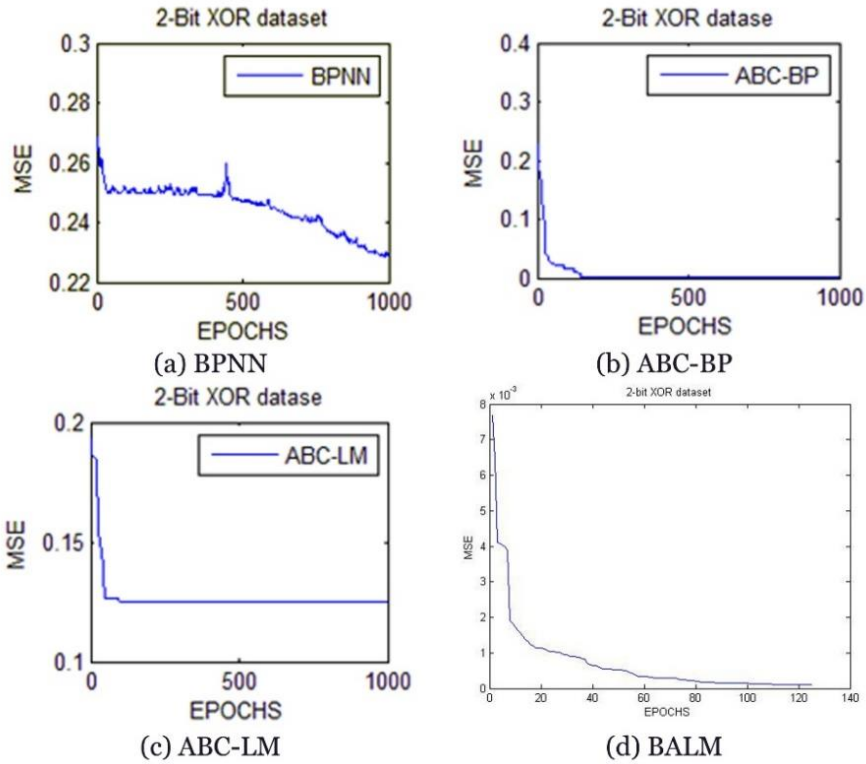

Figure 4 MSE Convergence Performance for: (a) BPNN; (b) ABC-BP; (c) ABC-LM; and (d) BALM on 2-Bit XOR dataset

Three-Bit Exclusive-OR Problem

In the second phase, a three-bit XOR dataset, which comprises three inputs and a single binary output, is used. The parameter range used for the 3–5–1 network consists of twenty connection weights and six biases. Table 2 shows the CPU time, number of epochs, MSE, and accuracy for the three-bit XOR test problems with five hidden neurons.

Table 2 Comparison results in terms of CPU time, epochs, MSE and accuracy for 3-Bit XOR dataset

Algorithm	BPNN	ABC-BP	ABC-LM	BALM	Improvement
CPU TIME	50.03	172.34	123.79	63	83 %
EPOCHS	1000	1000	1000	56	1685 %
MSE	0.25	0.08	0.01063	5.69E-6	1995389 %

Accuracy	47.63	86.47	78.83	99.99	29 %

For the three-bit XOR, the BALM algorithm converges to global minima within 56 epochs and 99.99 percent accuracy. The proposed BALM algorithm took 60.79 fewer CPU cycles, 944 fewer epochs, and showed 21.16 percent improved accuracy than the best performing ABC-LM algorithm. Overall from Table 2, it is clear that the proposed BALM algorithm has better performance than the compared algorithms in Figure 5, the convergence performance of the BALM, ABC-BP, ABC-LM, and conventional BPNN algorithms is shown.

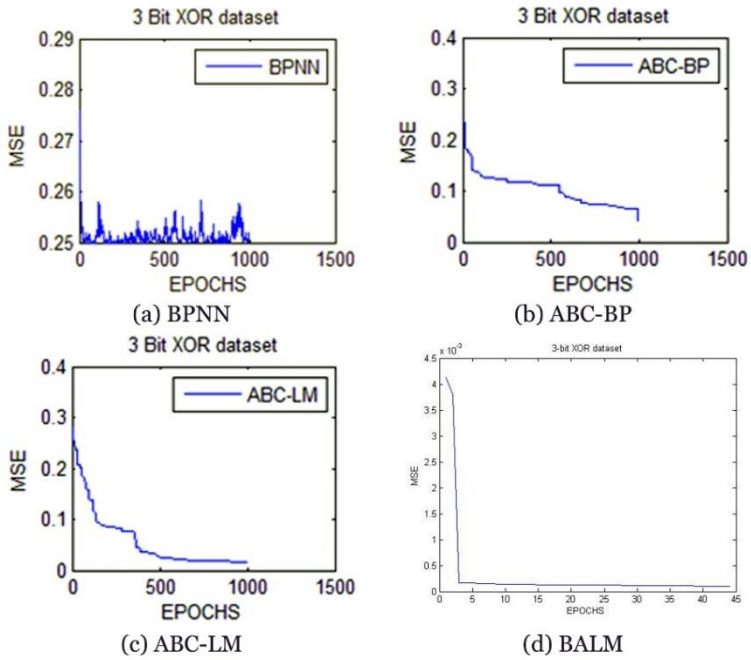

Figure 5 MSE Convergence Performance for: (a) BPNN; (b) ABC-BP; (c) ABC-LM; and (d) BALM on 3-Bit XOR dataset

Four-Bit OR Problem

The network structure for four-bit OR is similar to the two- and three-bit XOR problem. In four-bit OR, if the number of inputs all is 0, the output is 0, otherwise the output is 1. The four-bit network consists of 4 inputs, 5 hidden neurons in the hidden layer, and 1 output. The 4–5–1 feed-forward neural network structure is created with twenty-five connection weights and six biases. Table 3 illustrates the CPU time, epochs, and MSE performance of the proposed BALM, ABC-BP, ABC-LM and BPNN algorithms, respectively. From Table 3, it can be observed that the proposed BALM algorithm converges to global minima using 81.9 fewer CPU cycles, and 927 fewer epochs than the other algorithms. Figure 6 shows the convergence performance of the BALM, ABC-BP, ABC-LM, and conventional BPNN algorithms for the 4–5–1 network architecture.

Table 3 Comparison results in terms of CPU time, epochs, MSE and accuracy for 4-Bit OR dataset

Algorithm	BPNN	ABC-BP	ABC-LM	BALM	Improvement
CPUTIME	63.280	162.49	118.72	36.82	211 %
EPOCHS	1000	1000	1000	73	1269 %
MSE	0.053	1.91E-10	1.82E-10	4.41E-6	400504 %
Accuracy	89.83	99.97	99.99	99.99	3 %

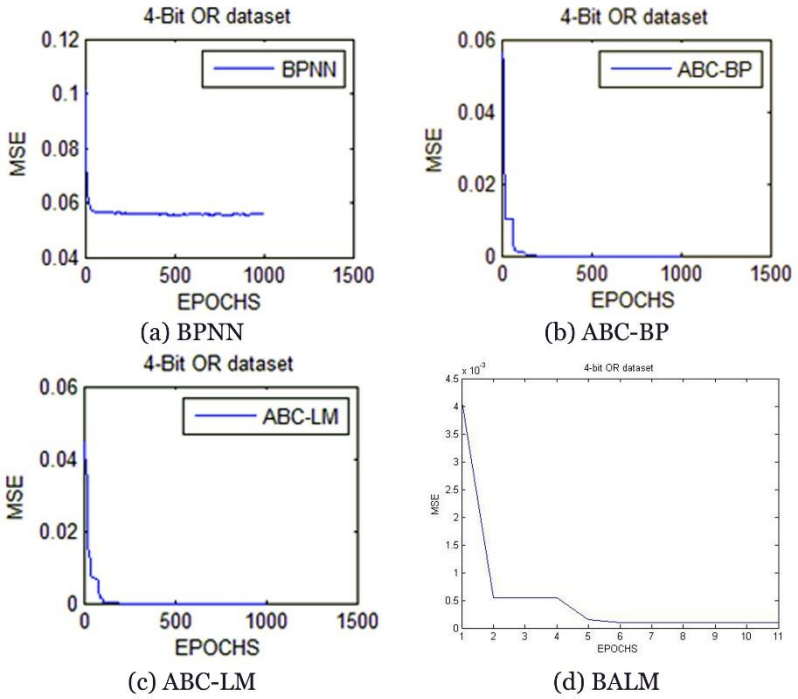

Figure 6 MSE Convergence Performance for: (a) BPNN; (b) ABC-BP; (c) ABC-LM; and (d) BALM on 4-Bit OR dataset

7 Bit Parity Dataset

For a seven-bit parity dataset, the network architecture consists of 7 inputs, 5 hidden neurons in the hidden layer, and 1 output. It has 40 connection weights and six biases. Table 4 confirms the CPU time, number of epochs, MSE, and accuracy for the seven-bit parity test problem with five hidden neurons. The proposed BALM's convergence rate is found optimum than the other techniques in terms of CPU time and number of epochs, MSE, and accuracy. The BALM algorithm shows superior performance than the comparison algorithms, and converted to an MSE of 5.92E-06 within 33 epochs, while the ABC-LM and ABC-BP have larger MSEs of 0.083 and 0.217, respectively. Also, the BALM algorithm shows 63.90 less CPU time, 967 fewer epochs, and 30.85 percent more accuracy than the best performing ABC-LM algorithm used in this study. Figure 7 illustrates the superior convergence performance of the proposed BALM algorithm.

Table 4 Comparison results in terms of CPU time, epochs, MSE and accuracy for 7-Bit Parity dataset

Algorithm	BPNN	ABC-BP	ABC-LM	BALM	Improvement
CPUTIME	22.19	183.39	134.88	70.98	59 %
EPOCHS	1000	1000	1000	33	2930 %
MSE	0.260	0.217	0.083	5.92E-06	3153053 %
Accuracy	85.12	82.13	69.14	99.99	21 %

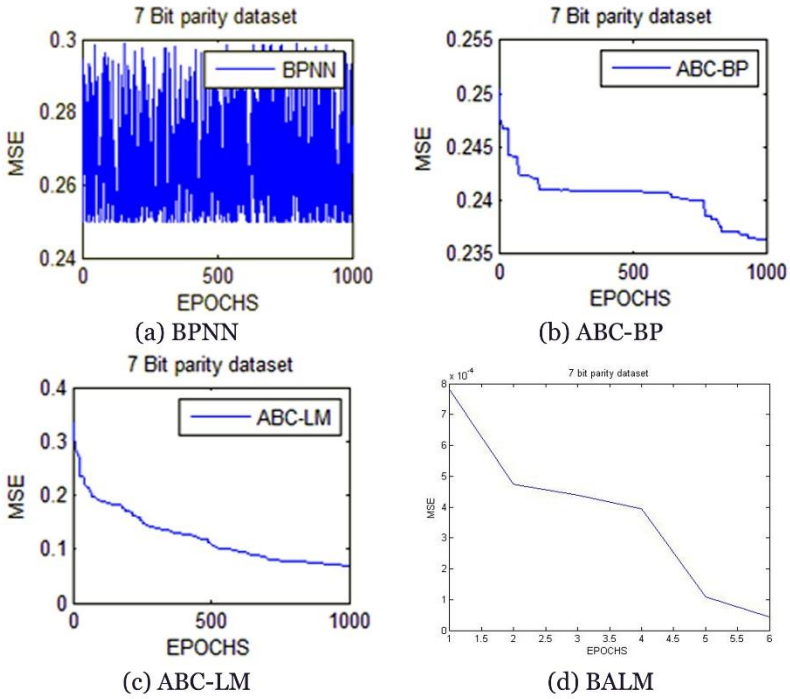

Figure 7 MSE Convergence Performance for: (a) BPNN; (b) ABC-BP; (c) ABC-LM; and (d) BALM on 7-Bit Parity dataset

Breast Cancer Classification Dataset

The Breast Cancer (Wisconsin) dataset is taken from the UCI Machine Learning Repository. Created by Dr William H. Wolberg, this problem tried to diagnose breast cancer by classifying a tumour as either benign or malignant [33]. This dataset consists of 9 inputs and 2 outputs, with 699 instances. The input attributes are the thickest clump, uniformity of cell size, uniformity of cell shape, amount of marginal adhesion, single epithelial cell size, frequency of bare nuclei, bland chromatin, normal nucleoli, and mitoses. The selected network architecture used for the breast cancer classification dataset consists of 9 inputs nodes, 5 hidden nodes and 2 output nodes. Figure 8 demonstrates the superior convergence performance of the proposed BALM algorithm.

Table 5 Comparison results in terms of CPU time, epochs, MSE and accuracy for Breast Cancer dataset

Algorithm	BPNN	ABC-BP	ABC-LM	BALM	Improvement
CPUTIME	95.46	1482.91	1880.65	34.04	3287 %
EPOCHS	1000	1000	1000	7	14285 %
MSE	0.271	0.184	0.014	4.04E-06	3869537 %
Accuracy	90.72	92.02	93.83	99.99	8 %

From Table 5 it is clear that the proposed BALM method has better performance than the comparison algorithms. The proposed BALM algorithm takes 993 fewer epochs, and 1846.61 less CPU time, offering a 6.16 percent increase in accuracy when compared with the best performing ABC-LM algorithm. The proposed BALM algorithm achieves 4.04E-06 MSE, which is accurate up to 6 decimal points, and achieves 99.99 percent accuracy, which is better than the other algorithms, while the other methods, such as conventional BPNN, ABC-BP, and ABC-LM, have MSEs of 0.271, 0.184 and 0.0139, respectively.

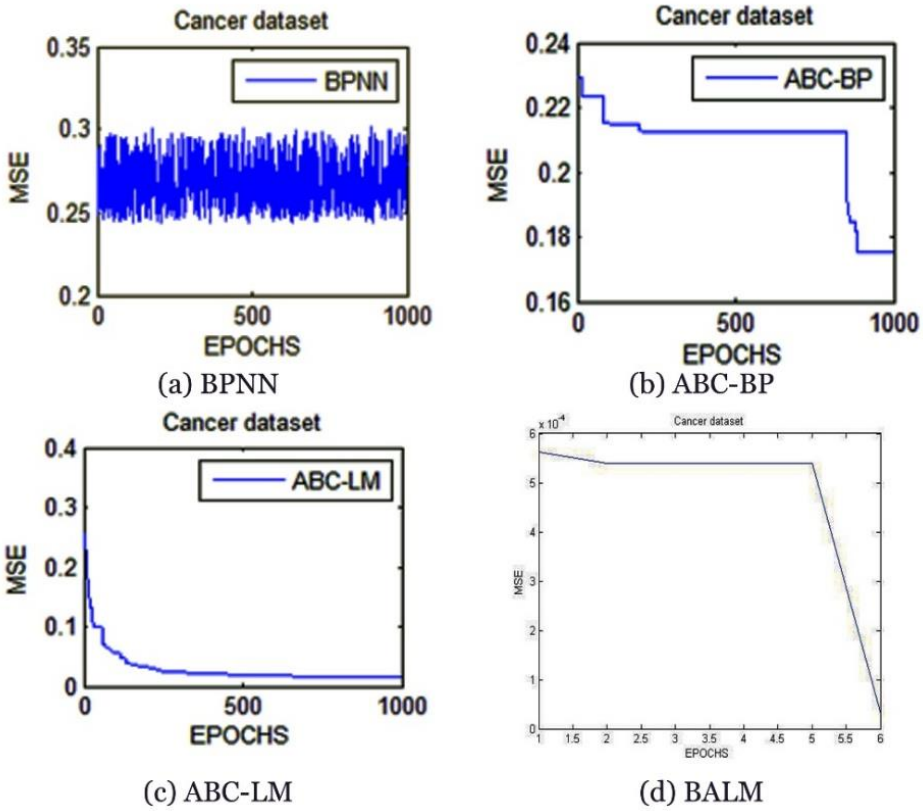

Figure 8 MSE Convergence Performance for: (a) BPNN; (b) ABC-BP; (c) ABC-LM; and (d) BALM on Breast Cancer dataset

IRIS Classification Dataset

Created by Sir Ronald Aylmer Fisher, the Iris classification dataset is the most famous dataset found in the pattern-recognition literature [34]. It consists of 150 instances, 4 inputs, and 3 outputs. The classification of the Iris dataset involves the data of petal width, petal length, sepal length, and sepal width into three classes of species, which consists of Iris Santos, Iris Versicolor, and Iris Virginica. The selected network structure of the Iris dataset consists of 4 input nodes, 5 hidden nodes and 3 output nodes. To train the network, 80 instances are used for the training set and the rest for the testing set.

Table 6 Comparison results in terms of CPU time, epochs, MSE and accuracy for Iris dataset

Algorithm	BPNN	ABC-BP	ABC-LM	BALM	Improvement
CPUTIME	28.47	156.43	171.52	8.35	1322 %
EPOCHS	1000	1000	1000	4	249 %
MSE	0.311	0.155	0.058	1.62E-06	10781790 %
Accuracy	87.19	86.87	79.56	99.99	15 %

Table 6 displays the result of the proposed BALM on Iris dataset. From the Table 6, it's clear that the proposed BALM shows better performance than other methods, in terms of MSE and accuracy. Within 6 epochs the proposed BALM algorithm gets 1.62E-06 of MSE, with an average accuracy of 99.99 percent, and takes 8.35 CPU seconds, while the other algorithms still have large MSEs and less accuracy than the proposed BALM algorithm. Figure 9 oresents the superior convergence performance of the proposed BALM algorithm.

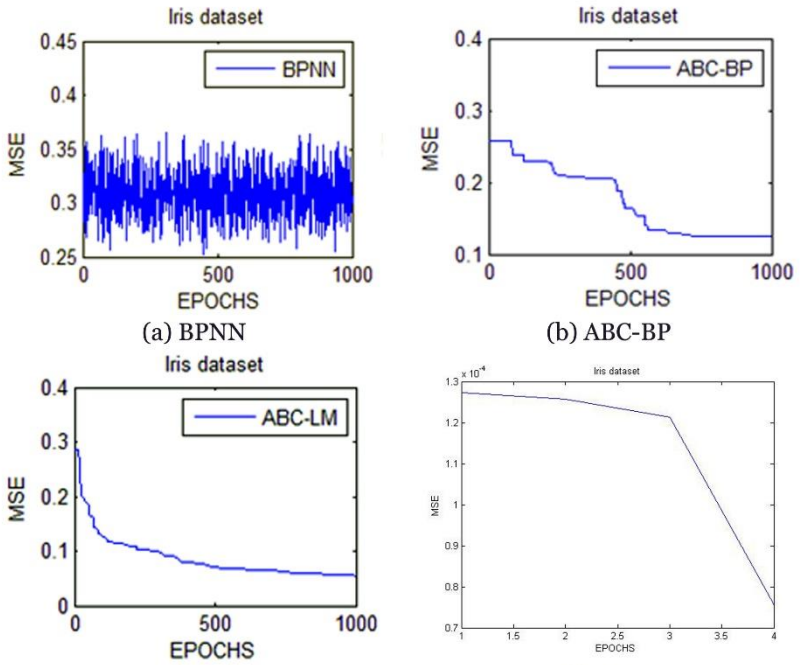

Figure 9 MSE Convergence Performance for: (a) BPNN; (b) ABC-BP; (c) ABC-LM; and (d) BALM on Iris dataset

Thyroid Classification Dataset

This dataset is also taken from the UCI Machine Learning Repository [35], and consists of 21 inputs, 3 outputs and 7200 patterns. Each case contains 21 attributes, which can be allocated to any of 3 classes, which are hyper, hypo, and normal function of the thyroid gland, based on the patient query data and the examination date. The selected network architecture for the thyroid classification dataset consists of 21 input nodes, 5 hidden nodes and 3 output nodes.

Table 7 Comparison results in terms of CPU time, epochs, MSE and accuracy for thyroid dataset

Algorithm	BPNN	ABC-BP	ABC-LM	BALM	Improvement
CPUTIME	38.43	34153.13	38382.9	6204.06	289 %
EPOCHS	1000	1000	1000	499	100 %
MSE	0.311	0.046	0.041	0.003	4322 %
Accuracy	85.88	93.28	91.66	99.65	9 %

Table 7 illustrates the simulation results for the thyroid classification problem. In Table 7 we can see that the proposed BALM method converges on the global minima with 0.003 MSE, with 99.65 percent accuracy, while the other algorithms failed to achieve high accuracy, and needed more CPU time when compared with the BALM algorithm. The convergence performance of the algorithms can be seen in Figure 10.

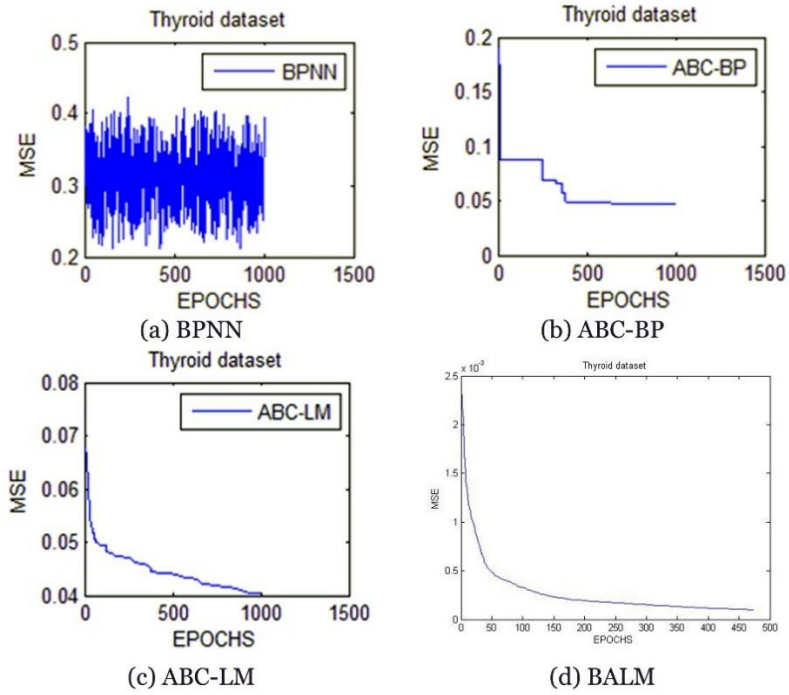

Figure 10 MSE Convergence Performance for: (a) BPNN; (b) ABC-BP; (c) ABC-LM; and (d) BALM on Thyroid dataset

Diabetes Classification Dataset

The Pima India diabetes dataset is taken from the UCI Machine Learning Repository [36], and consists of 768 examples, 8 inputs and 2 outputs, as well as all the information of the chemical change in a female body whose disproportion can cause diabetes. The feed-forward network topology for this network is set to 8–5–2. Table 8 shows the CPU time, MSE, and accuracy for the proposed BALM and conventional BPNN, ABC-BP, ABC-LM, and clearly demonstrates that the proposed BALM model performs better than the other methods, in terms of MSE and accuracy. From Table 8, it can be seen that the proposed BALM algorithm achieved 0.017 MSE with 97.26 percent accuracy, which is far better than the other algorithms. The convergence performance of the algorithms can be grasped from Figure 11.

Table 8 Comparison results in terms of CPU time, epochs, MSE and accuracy for diabetes dataset

Algorithm	BPNN	ABC-BP	ABC-LM	BALM	Improvement
CPUTIME	57.05	4257.32	2805.09	136.91	1633 %
EPOCHS	1000	1000	1000	17	5782 %
MSE	0.269	0.201	0.141	0.017	1098 %
Accuracy	84.95	91.46	65.09	97.26	17 %

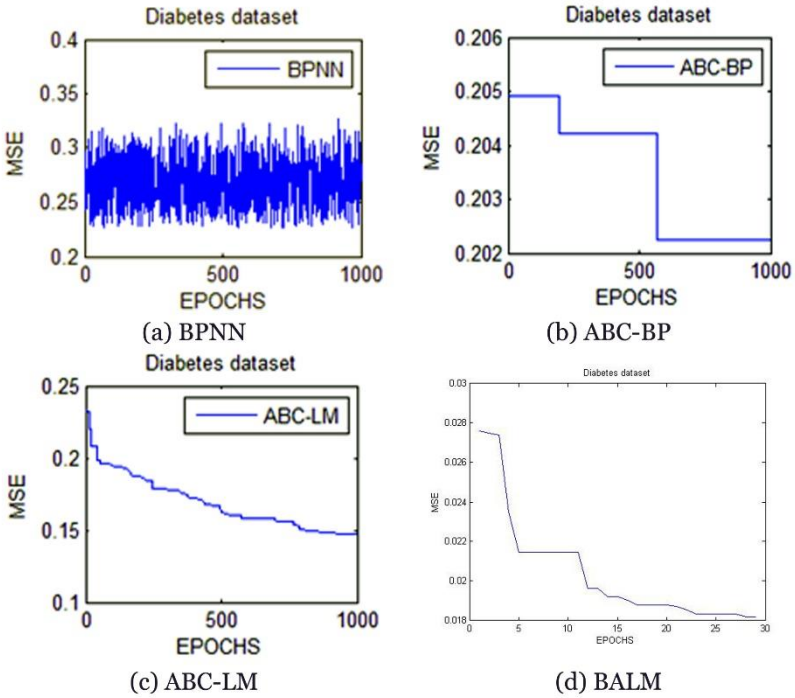

Figure 11 MSE Convergence Performance for: (a) BPNN; (b) ABC-BP; (c) ABC-LM; and (d) BALM on diabetes dataset

Glass Classification Dataset

The glass dataset is used for separating glass splinters in criminal investigation into six classes, and is taken from the UCI Machine Learning Repository [37]. The dataset consists of float-processed or non-float-processed building windows, vehicles, windows, containers, tableware, or head lamps. This dataset is made up of 9 inputs, and 6 outputs. The size of the dataset consists of 214 attributes in total. The selected feed-forward network architecture is set to 9–5–6. The simulation results for the benchmark glass classification problem are given in Table 9, where it is clear that the proposed BALM algorithm has achieved a smaller 9.17E-06 MSE within 133 epochs. The convergence performance of BALM and other algorithms can be seen in Figure 12.

Table 9 Comparison results in terms of CPU time, epochs, MSE and accuracy for glass dataset

Algorithm	BPNN	ABC-BP	ABC-LM	BALM	Improvement
CPUTIME	32.74	1715.95	1336.19	197.88	419 %
EPOCHS	1000	1000	1000	133	651 %
MSE	0.364	0.026	0.005	9.17E-06	1435742 %
Accuracy	94.04	91.36	93.96	99.99	6 %

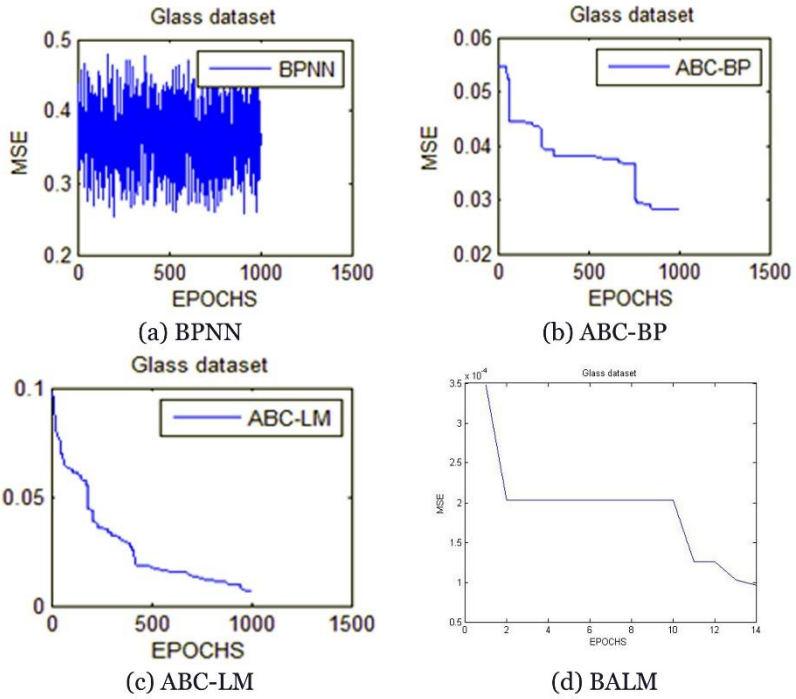

Figure 12 MSE Convergence Performance for: (a) BPNN; (b) ABC-BP; (c) ABC-LM; and (d) BALM on glass Dataset

Australian Credit Card Classification Dataset

This dataset is also taken from the UCI Machine Learning Repository [38], and consists of all the procedures of clearing a person for Credit Card approval, based on their past financial proceedings. All attributes names and values have been changed to meaningless symbols to defend the privacy of the applicant. The Australian Credit Card dataset consists of 690 instances, 51 inputs, and 2 outputs.

Table 10 displays the CPU time MSE, epochs, and accuracy of the Australian Credit dataset. From Table 10 it can be seen that the proposed BALM model achieved 99.83% accuracy with an MSE of 0.001 in 69 epochs, while the other algorithms, such as BPNN, ABC-BP, and ABC-LM, have smaller percentage accuracies of 89.99, 77.782 and 88.89, respectively. The MSE convergence performance of the proposed BALM and other algorithms can be seen in Figure 13.

Table 10 Comparison results in terms of CPU time, epochs, MSE and accuracy for Australian Credit Card dataset

Algorithm	BPNN	ABC-BP	ABC-LM	BALM	Improvement
CPUTIME	24.43	6894.25	4213.01	1418.11	161 %
EPOCHS	1000	1000	1000	69	1349 %
MSE	0.271	0.173	0.055	0.001	16533 %
Accuracy	88.89	89.99	77.78	99.83	14 %

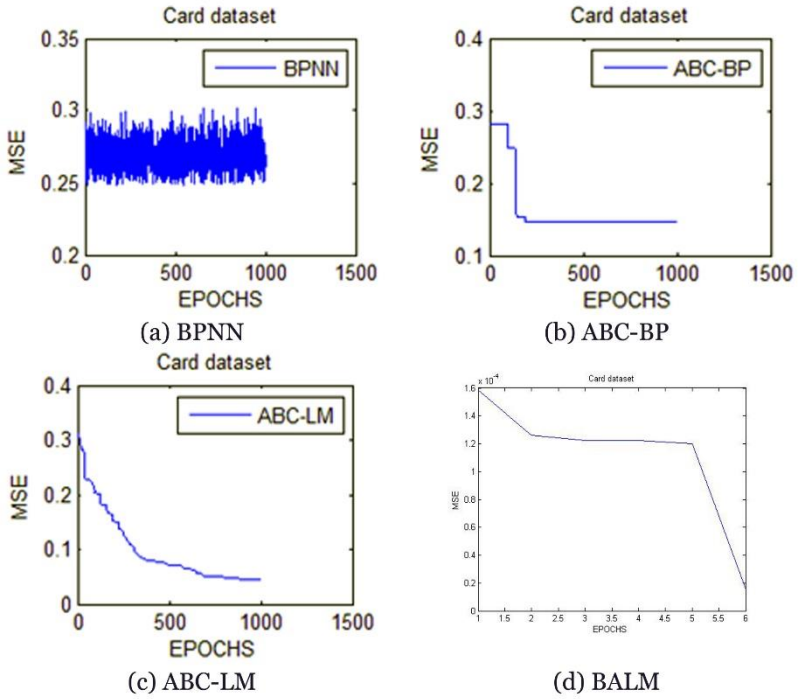

Figure 13 MSE Convergence Performance for: (a) BPNN; (b) ABC-BP; (c) ABC-LM; and (d) BALM on Australian Credit Card dataset

In the simulations, it was found that Bat can enhance the performance of the LM by deviating from the limitations of gradient descent, such as reducing the error in the gradient and escaping from the local minima. Tables 1–10 show that the proposed algorithm generally performs better than the comparison algorithms, except in a few cases (see Tables 1 and 3). The probable reason why our proposed algorithm was able to improve the performance of the state of the algorithms can best be attributed to the behaviour of the echolocation of microbats in the Bat algorithm. This feature was likely responsible for the effective searching of the search space, to locate the optimal BATLM, which contributed to the efficiency and accuracy of the BALM. The study has proved that the BALM has the potential to be more powerful than the state of the algorithms, thus BALM can be investigated further for solving problems in other application domains.

5. Conclusion

Levenberg-Marquardt (LM) is a widely used Artificial Neural Network (ANN) training algorithm. Regardless of its advantages, the LM algorithm is sluggish and susceptible to the local minima problem. In this paper, a hybrid learning algorithm that integrates the Bat algorithm and the Levenberg-Marquardt (LM) algorithm is introduced. The proposed Bat-based Levenberg-Marquardt (BALM) algorithm is trained and tested on ten benchmarked classification datasets. During the experiments, the BALM algorithm obtained high accuracy in classification within a short execution time period. Like many other meta-heuristic algorithms, BALM has the advantage of simplicity, and can be easily developed to be used in a wide range of applications. The Bat algorithm, which has led the Levenberg-Marquardt (LM) to avoid local minima in this paper, can be further enhanced by parameter tuning and dynamic parameter control.

Acknowledgment

The authors would like to thank Office of Research, Innovation, Commercialization and Consultancy Office (ORICC), Universiti Tun Hussein Onn Malaysia (UTHM) and Ministry of Higher Education (MOHE) Malaysia for financially supporting this Research under Fundamental Research Grant Scheme (FRGS) vote no 1236. This research is also supported by Gates IT Solution Sdn. Bhd under its publication scheme.

References

[1] Radhika, Y. & Shashi, M., *Atmospheric Temperature Prediction using Support Vector Machines*, International Journal of Computer Theory and Engineering, **1**(1), pp. 1793-8201. 2009.

[2] Akcayol, M.A. & Cinar, C., *Artificial Neural Network Based Modeling of Heated Catalytic Converter Performance*, Journal of Applied Thermal Engineering, **25**, pp. 2341-2350, 2005.

[3] Shereef, K.I. & Baboo, S.S., *A New Weather Forecasting Technique using Back Propagation Neural Network with Modified Levenberg-Marquardt Algorithm for Learning*, IJCSI International Journal of Computer Science, **8**(6), pp. 1694-0814, 2011.

[4] Kosko, B., *Neural Network and Fuzzy Systems, 1st ed*., Prentice Hall of India, 1994.

[5] Krasnopolsky, V.M. & Chevallier, F., *Some Neural Network application in environmental sciences. Part II: Advancing Computational Efficiency of Environmental Numerical Models*, Neural Networks, **16**(3-4), pp. 335-348, 2003.

[6] Coppin, B., *Artificial Intelligence Illuminated*, Jones and Bartlet Illuminated Series, USA, Chapter 11, pp. 291- 324, 2004.

[7] Basheer, I.A. & Hajmeer, M., *Artificial Neural Networks: Fundamentals, Computing, Design, and Application*, Journal of Microbiological Methods, **43**(1), pp. 03-31, 2000.

[8] Zheng, H., Meng, W. & Gong, B., *Neural Network and its Application on Machinery Fault Diagnosis*, IEEE International Conference on Systems Engineering (ICSYSE), pp. 576--579, 17-19 September, Kobe, Japan, 1992.

[9] Rehman, M. Z., and Nawi, N. M., *Improving the Accuracy of Gradient Descent Back Propagation Algorithm* (*GDAM*) *on Classification Problems*, International Journal of New Computer Architectures and their Applications (IJNCAA), **1**(4), pp.838-847, 2012.

[10] Rumelhart D.E., Hinton G.E. & Williams R.J., *Learning Representations by Back-Propagating Errors*, Nature, **323**, pp. 533-536, 1986.

[11] Lahmiri, S., *A Comparative Study of Back-propagation Algorithms in Financial Prediction*, International Journal of Computer Science, Engineering and Applications (IJCSEA), **1**(4), 2011.

[12] Nawi, N.M., Ransing, R.S. & AbdulHamid, N., *BPGD-AG: A New Improvement of Back-Propagation Neural Network Learning Algorithms with Adaptive Gain*, Journal of Science and Technology, **2**(2), 2011.

[13] Ahmed, W., Saad, E. & Aziz, E., *Modified Back Propagation Algorithm for Learning Artificial Neural Networks*, The 18th National Radio Science Conference (NRSC), pp. 345-352, 27-29 March, Mansoura, Egypt, 2001.

[14] Wen, J. Zhao, J.L., Luo. S.W. & Han, Z., *The Improvements of BP Neural Network Learning Algorithm*, 5th Int. Conf. on Signal Processing WCCC-ICSP, pp.1647-1649, 21-25 August, Beijing, China, 2000.

[15] Salchenberger, L.M., Cinar, E.M. & Lash N.A., *Neural Networks: A New Tool for Predicting Thrift Failures*, Decision Sciences, **23**(2), pp. 899-916, 1992.

[16] Sexton, R.S., Dorsey, R.E. & Johnson, J.D., *Toward Global Optimization of Neural Networks: A Comparison of the Genetic Algorithm and Back-propagation*, Decision Support Systems, **22**, pp. 171–186, 1998.

[17] SNNS (Stuttgart Neural Network Simulator), http://wwwra.informatik. unituebingen.de/SNNS/, 25th January, 2013.

[18] Fahlman, S.E., *Faster-Learning Variations on Back Propagation: An Empirical Study*, 1988 Connectionist Models Summer School by Scott E. Fahlman, pp. 38-51, San Mateo, CA, 1988.

[19] Riedmiller, M. & Braun. H., *A Direct Adaptive Method for Faster Back Propagation Learning: The RPROP Algorithm*, IEEE International Conference on Neural Networks (ICNN93), 28th March-1st April, San Francisco, CA, 1993.

[20] Fahlman, S.E. & Lebiere, C., *The Cascade-Correlation Learning Architecture*, Advances in Neural Information Processing Systems, **2**, pp. 524-532, San Mateo, Calif, 1990.

[21] Hagan, M.T. & Menhaj, M.B., *Training Feed Forward Networks with the Marquardt Algorithm*, IEEE Trans. on Neural Networks, **23**, pp. 899-916, 1994.

[22] Wilamowski, B.M., Cotton, N., Hewlett, J. & Kaynak, O., *Neural Network Trainer with Second Order Learning Algorithms*, 11th International Conference on Intelligent Engineering Systems, Budapest, Hungary, IEEE, 2007.

[23] Hagan, M.T. & Menhaj, M.B., *Training Feed Forward Networks with the Marquardt Algorithm*, IEEE Trans. Neural Network., **5**(6), pp. 989-993, 1994.

[24] Xiao-ping, C., Chang-hua, H., Zhi-qiang, Z. & Ying-jie, L., *Fault Prediction for Inertial Device Based on LMBP Neural Network*, Electronics Optics & Control, **12** (6), pp.38-41, 2005.

[25] Haykin, S., *Neural Networks*, Beijing, China Machine Press, pp. 501-522, 2004.

[26] Nawi, N. M., Khan, A. & Rehman, M.Z., *A New Levenberg Marquardt Based Back Propagation Algorithm Trained with Cuckoo Search*, ICEEI, UKM, 2013.
[27] Yan, J., Cao, H., Wang, J., Liu, Y. & Zhao, H., *Levenberg-Marquardt Algorithm Applied to Forecast the Ice Conditions in Ningmeng Reach of The Yellow River*, 5th International Conference on Natural Computation, pp. 184-188, 14-16 August, Tianjin, China, 2009.
[28] Yang, X. S.: *A new metaheuristic bat-inspired algorithm*, Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), pp. 65--74, 2010.
[29] Nawi, N. M., Rehman, M. Z., Ghazali, M. I., Yahya, M. N., Khan, A., *Hybrid Bat-BP: A New Intelligent tool for Diagnosing Noise-Induced Hearing Loss (NIHL) in Malaysian Industrial Workers*, Journal of Applied Mechanics and Materials, Trans Tech Publications, Switzerland, vol. 465-466, pp. 652-656, 2014.
[30] Nawi, N. M., Rehman, M. Z., Khan, A., *The Effect of Bat Population in Bat-BP Algorithm*, ROVISP-2013, Proceedings in LNEE Journal of Springer, Penang, Malaysia.
[31] Nawi, N. M., Rehman, M. Z., Khan, A., *A New Bat Based Back-Propagation (BAT-BP) Algorithm*, ICSS-2013, Proceedings in LNEE Journal of Springer, Wroclaw, Poland.
[32] Mamat, R., Herawan, T., and Deris, M. M., *MAR: Maximum Attribute Relative of soft set for clustering attribute selection*, *Knowledge-Based Systems*, **52**, pp. 11--20, 2013.
[33] Wolberg, W.H., and Mangasarian, O.L., *Multisurface method of pattern separation for medical diagnosis applied to breast cytology*. In: National Academy of Sciences, **87**, pp. 9193--9196, 1990.
[34] Fisher, R. A., *The use of multiple measurements in taxonomic problems*: Annual Eugenics, **7**, 179--188 (1936)
[35] Quinlan, J. R., Compton, P. J., Horn, K. A., and & Lazurus, L., *Inductive knowledge acquisition: A case study*, Second Australian Conference on Applications of Expert Systems, Sydney, Australia, 1986.
[36] Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., & Johannes, R.S., *Using the ADAP learning algorithm to forecast the onset of diabetes mellitus*, The Symposium on Computer Applications and Medical Care, IEEE Computer Society Press, pp. 261--265, 1988.
[37] Evett, I. W., and Spiehler, E. J., *Rule induction in forensic science*, KBS in Goverment, Online Publications, pp. 107--118, 1987.
[38] Quinlan, J. R., *Simplifying Decision Trees*, J. Man-Machine Studies, **27**, pp. 221--234, 1987.
[39] Chiroma, H., Abdul-kareem, S., Khan, A., Nawi, N. M., Gital, A. Y., Shuib, L., *Global Warming: Predicting OPEC Carbon Dioxide Emissions from Petroleum Consumption Using Neural Network and Hybrid Cuckoo Search Algorithm*, PLoS ONE, **10**(8), 2015.
[40] Gandomi, A. H., Yang, X. S., Alavi, A. H., Talatahari, S., *Bat Algorithm for Constrained optimization tasks*, Neural Computing and Applications, **22**(6), pp. 1239--1255, 2013.
[41] Yang, X. S., He, X., Bat algorithm: literature review and applications, *International Journal of Bio-Inspired Computation*, **5**(3), pp. 141--149, 2013.