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

NAZRI MOHD NAWI1, MUHAMMAD ZUBAIR REHMAN1, ABDULLAH KHAN1, ARSLAN KIYANI1, HARUNA CHIROMA2 AND TUTUT HERAWAN2

1Software and Multimedia Centre
Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia
Johor, 86400 Malaysia

2Faculty of Computer Science and Information Technology
University of Malaya
Lumpur, 50603 Malaysia

2Universitas Teknologi Yogyakarta
2AMCS Research Center, Yogyakarta, Indonesia
E-mail: nazri@uthm.edu.my

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, optimization, swarm intelligence

1. INTRODUCTION

Artificial Neural Networks’ (ANN) processing elements translate the synaptic behavior 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

Received May 24, 2015; revised July 17, 2015; accepted August 28, 2015.
Communicated by Jen-Tzung Chien.
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 modification of the Bat (i.e. BALM algorithm is presented in Section 3). In 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 behavior 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 \( (v_t) \) at position \( (x_t) \) with a fixed frequency \( (f_{\text{min}}) \), varying wavelength \( \lambda \) 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 \( f_{\text{min}} \).

The pseudo-code for the Bat algorithm is shown in Fig. 1 [28];
Bat is a population-based optimization algorithm, and like other meta-heuristic algorithms, it starts with a random initial population. Fig. 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 behavior 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 Fig. 2. In Fig. 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:

\[ W_c = \sum_{c=1}^{n} a.(\text{rand} - \frac{1}{2}), \]  
\[ B_c = \sum_{c=1}^{n} a.(\text{rand} - \frac{1}{2}), \]  

Where \( W_c^{ij} \) is the weight value in a weight matrix. The rand in the Eq. (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:

\[ W^S = [W_1^{ij}, W_2^{ij}, W_3^{ij}, \ldots, W_n^{ij}], \]
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:

$$y_i = f\left(\sum_{j=1}^{N} W_{c(i,j)}a_j + B_j\right).$$

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

$$X_m = f\left(\sum_{m=1}^{M} W_{c(m)}y_i + B_m\right).$$

Where, $X_m$ is network output, $f$ is transfer function, $W_{c(jm)}$ 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:

$$e_r = (T_r - X_r).$$

The performance index for the network is calculated using the following Eqs. (7) and (8):

$$V_f(x) = \frac{1}{2} \sum_{r=1}^{R} e_r = \sum_{r=1}^{R} (T_r - X_r)^T (T_r - X_r),$$

$$V_r(x) = \frac{1}{2} \sum_{r=1}^{R} e_r = \sum_{r=1}^{R} e_r e_r.$$

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

$$V_\mu(x) = \sum_{r=1}^{R} V_r(x) / P_i.$$

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, $V_r(x)$ 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 starts 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:

$$E(t) = \frac{1}{2} \sum_{r=1}^{R} e_r^2(t).$$
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:

$$\nabla E(t) = J^T(t)e(t),$$  \hspace{1cm} (11)

$$\nabla^2 E(t) = J^T(t)J(t).$$  \hspace{1cm} (12)

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

$$J(t) = \begin{bmatrix}
\frac{\partial e_1(t)}{\partial t_1} & \frac{\partial e_1(t)}{\partial t_2} & \cdots & \frac{\partial e_1(t)}{\partial t_N} \\
\frac{\partial e_2(t)}{\partial t_1} & \frac{\partial e_2(t)}{\partial t_2} & \cdots & \frac{\partial e_2(t)}{\partial t_N} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial e_N(t)}{\partial t_1} & \frac{\partial e_N(t)}{\partial t_2} & \cdots & \frac{\partial e_N(t)}{\partial t_N}
\end{bmatrix}.$$  \hspace{1cm} (13)

For Gauss-Newton Method:

$$\nabla w(t) = -J^T(t)J(t) e(t).$$  \hspace{1cm} (14)

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

$$w(k+1) = w(k) - [J^T(t)J(t)\mu I]^{-1}J(t)e(t).$$  \hspace{1cm} (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 $\mu$ is large, the above expression approximates gradient descent (with learning rate $1/\mu$) while for a small $\mu$, 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 Eqs. (5) and (6) over all inputs. Compute the sum of square of error over all input.

(b) Compute the Jacobin matrix using Eq. (13).

(c) Solve Eq. (12) to obtain $\nabla w$.

(d) Re-compute the sum of squares of errors using Eq. (15), if this new sum of squares is smaller than the computed sum of square in Step 1, then reduce $\mu$ by $\lambda=10$, 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 $\mu$ by $\lambda=10$, and go back to Step 3.
(e) The algorithm is assumed to have converged when the norm of the gradient in Eq. (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:

\[
MSE_i = \{V_{\mu}(x_1), V_{\mu}(x_2), V_{\mu}(x_3), \ldots, V_{\mu}(x_n)\}.
\]  

(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:

\[
x_j = \text{Min}\{V_{\mu}(x_1), V_{\mu}(x_2), V_{\mu}(x_3), \ldots, V_{\mu}(x_n)\}.
\]  

(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 Eq. (18):

\[
x_i^{t+1} = x_i^{t} + v_i.
\]  

(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 Eqs. (19) and (20):

\[
f_i = f_{\text{min}} + (f_{\text{max}} - f_{\text{min}}) \beta,
\]  

(19)

\[
v_i = v_i^{t-1} + (x_i^{t-1} - x_i^{*})f_i.
\]  

(20)

Where \( \beta \) 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 Eq. (19) is used to control the pace and range of the movement of the bats. The variable \( x_i^{*} \) 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 \( x_i^{*} \) can be drawn from Eqs. (21-22). If \( \text{rand}[0, 1] > r_i \), then

\[
V = x_i + \text{rand} \cdot (x_i - x_i^{*}).
\]  

(21)

Else, if \( \text{rand}[0, 1] < A_i \) \& \& \( f(x_i) < f(x_i^{*}) \) The bats can move from \( x_i \) toward \( x_j \) randomly using Eq. (22).

\[
\nabla V_i = x_i + v_i \sim 0.01 \cdot (V - X_{\text{best}}).
\]  

(22)

Increase \( r_i \) and decrease \( A_i \).
Where \( A_i^t \) 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 \( A_i^t \) and the emission pulse rate \( r_i \) are updated, as follows:

\[
A_i^{t+1} = A_i^t, \tag{23}
\]

\[
r_i^{t+1} = r_i^0 \left[ 1 - \exp(-\gamma r_i) \right]. \tag{24}
\]

Where \( \alpha \) and \( \gamma \) 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, \( A_i^0 \in [1, 2] \) and \( r_i^0 \in [0, 1] \) \( \forall V_i \) is a small movement of \( x_j \) towards \( x_i \). The weights and biases for each layer is then adjusted as:

\[
W_i^{n+1} = W_i^n - \nabla V_i, \tag{25}
\]

\[
B_i^{n+1} = B_i^n - \nabla V_i. \tag{26}
\]

The pseudo-code for the proposed BALM is given in Fig. 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 Eq. (6).
7. Calculate the minimum error using Eq. (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 Eqs. (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. Solve the Jacobin matrix using Eq. (13).
11. Solve Eq. (14) to obtain \( \nabla w \).
12. Recompute the sum of squares of errors using Eq. (15). If this new sum of squares is smaller than that computed in Step 8, then reduce \( \mu \) by \( \lambda = 10 \), 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 \( \mu \) by \( \lambda = 10 \), and go back to Step 11.
13. The algorithm is assumed to have converged when the norm of the gradient in Eq. (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.
16. Build new solution using Eq. (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.

Fig. 3. Proposed BALM pseudo-code.
4. RESULTS AND DISCUSSIONS

4.1 Experimental Setup

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].

4.2 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. Fig. 4 illustrates the convergence performance of the BALM, ABC-BP, ABC-LM, and conventional BPNN algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>BPNN</th>
<th>ABC-BP</th>
<th>ABC-LM</th>
<th>BALM</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Time</td>
<td>42.64</td>
<td>172.34</td>
<td>123.95</td>
<td>241.32</td>
<td>53 %</td>
</tr>
<tr>
<td>EPOCHS</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>563</td>
<td>78 %</td>
</tr>
<tr>
<td>MSE</td>
<td>0.2206</td>
<td>2.39E-4</td>
<td>0.125</td>
<td>7.08E-5</td>
<td>162724 %</td>
</tr>
<tr>
<td>Accuracy</td>
<td>54.61</td>
<td>96.47</td>
<td>71.69</td>
<td>99.99</td>
<td>25 %</td>
</tr>
</tbody>
</table>
4.3 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.

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 Fig. 5, the convergence performance of the BALM, ABC-BP, ABC-LM, and conventional BPNN algorithms is shown.

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>BPNN</th>
<th>ABC-BP</th>
<th>ABC-LM</th>
<th>BALM</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU TIME</td>
<td>50.03</td>
<td>172.34</td>
<td>123.79</td>
<td>63</td>
<td>83 %</td>
</tr>
<tr>
<td>EPOCHS</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>56</td>
<td>1685 %</td>
</tr>
<tr>
<td>MSE</td>
<td>0.25</td>
<td>0.08</td>
<td>0.01063</td>
<td>5.69E-6</td>
<td>1995389 %</td>
</tr>
<tr>
<td>Accuracy</td>
<td>47.63</td>
<td>86.47</td>
<td>78.83</td>
<td>99.99</td>
<td>29 %</td>
</tr>
</tbody>
</table>
HYBRID BAT AND LEVENBERG-MARQUARDT ALGORITHMS FOR ANN LEARNING

4.4 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. Fig. 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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>BPNN</th>
<th>ABC-BP</th>
<th>ABC-LM</th>
<th>BALM</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUMTIME</td>
<td>63.280</td>
<td>162.49</td>
<td>118.72</td>
<td>36.82</td>
<td>211 %</td>
</tr>
<tr>
<td>EPOCHS</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>73</td>
<td>1269 %</td>
</tr>
<tr>
<td>MSE</td>
<td>0.053</td>
<td>1.91E-10</td>
<td>1.82E-10</td>
<td>4.41E-6</td>
<td>400504 %</td>
</tr>
<tr>
<td>Accuracy</td>
<td>89.83</td>
<td>99.97</td>
<td>99.99</td>
<td>99.99</td>
<td>3 %</td>
</tr>
</tbody>
</table>
4.5 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. Fig. 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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>BPNN</th>
<th>ABC-BP</th>
<th>ABC-LM</th>
<th>BALM</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU TIME</td>
<td>22.19</td>
<td>183.39</td>
<td>134.88</td>
<td>70.98</td>
<td>59 %</td>
</tr>
<tr>
<td>EPOCHS</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>33</td>
<td>2930 %</td>
</tr>
<tr>
<td>MSE</td>
<td>0.260</td>
<td>0.217</td>
<td>0.083</td>
<td>5.92E-06</td>
<td>3153053 %</td>
</tr>
<tr>
<td>Accuracy</td>
<td>85.12</td>
<td>82.13</td>
<td>69.14</td>
<td>99.99</td>
<td>21 %</td>
</tr>
</tbody>
</table>
4.6 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 tumor 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. Fig. 8 demonstrates the superior convergence performance of the proposed BALM algorithm.

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.
Table 5. Comparison results in terms of cpu time, epochs, mse and accuracy for breast cancer dataset.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>BPNN</th>
<th>ABC-BP</th>
<th>ABC-LM</th>
<th>BALM</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUTIME</td>
<td>95.46</td>
<td>1482.91</td>
<td>1880.65</td>
<td>34.04</td>
<td>3287%</td>
</tr>
<tr>
<td>EPOCHS</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>7</td>
<td>14285%</td>
</tr>
<tr>
<td>MSE</td>
<td>0.271</td>
<td>0.184</td>
<td>0.014</td>
<td>4.04E-06</td>
<td>3869537%</td>
</tr>
<tr>
<td>Accuracy</td>
<td>90.72</td>
<td>92.02</td>
<td>93.83</td>
<td>99.99</td>
<td>8%</td>
</tr>
</tbody>
</table>

Fig. 8. MSE convergence performance for: (a) BPNN; (b) ABC-BP; (c) ABC-LM; and (d) BALM on breast cancer dataset.

4.7 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 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 sec-
onds, while the other algorithms still have large MSEs and less accuracy than the proposed BALM algorithm. Fig. 9 presents the superior convergence performance of the proposed BALM algorithm.

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>BPNN</th>
<th>ABC-BP</th>
<th>ABC-LM</th>
<th>BALM</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUTIME</td>
<td>28.47</td>
<td>156.43</td>
<td>171.52</td>
<td>8.35</td>
<td>1322 %</td>
</tr>
<tr>
<td>EPOCHS</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>4</td>
<td>249 %</td>
</tr>
<tr>
<td>MSE</td>
<td>0.311</td>
<td>0.155</td>
<td>0.058</td>
<td>1.62E-06</td>
<td>10781790 %</td>
</tr>
<tr>
<td>Accuracy</td>
<td>87.19</td>
<td>86.87</td>
<td>79.56</td>
<td>99.99</td>
<td>15</td>
</tr>
</tbody>
</table>

Fig. 9. MSE convergence performance for: (a) BPNN; (b) ABC-BP; (c) ABC-LM; and (d) BALM on Iris dataset.

4.8 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 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 Fig. 10.

4.9 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 Fig. 11.

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>BPNN</th>
<th>ABC-BP</th>
<th>ABC-LM</th>
<th>BALM</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUTIME</td>
<td>57.05</td>
<td>4257.32</td>
<td>2805.09</td>
<td>136.91</td>
<td>1633%</td>
</tr>
<tr>
<td>EPOCHS</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>17</td>
<td>5782%</td>
</tr>
<tr>
<td>MSE</td>
<td>0.269</td>
<td>0.201</td>
<td>0.141</td>
<td>0.017</td>
<td>1098%</td>
</tr>
<tr>
<td>Accuracy</td>
<td>84.95</td>
<td>91.46</td>
<td>65.09</td>
<td>97.26</td>
<td>17%</td>
</tr>
</tbody>
</table>

4.10 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 algo-
Algorithm has achieved a smaller 9.17E-06 MSE within 133 epochs. The convergence performance of BALM and other algorithms can be seen in Fig. 12.

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>CPUTIME</th>
<th>EPOCHS</th>
<th>MSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPNN</td>
<td>32.74</td>
<td>1000</td>
<td>0.364</td>
<td>94.04</td>
</tr>
<tr>
<td>ABC-BP</td>
<td>1715.95</td>
<td>1000</td>
<td>0.026</td>
<td>91.36</td>
</tr>
<tr>
<td>ABC-LM</td>
<td>1336.19</td>
<td>1000</td>
<td>0.005</td>
<td>93.96</td>
</tr>
<tr>
<td>BALM</td>
<td>197.88</td>
<td>133</td>
<td>9.17E-06</td>
<td>99.99</td>
</tr>
</tbody>
</table>

Fig. 12. MSE convergence performance for: (a) BPNN; (b) ABC-BP; (c) ABC-LM; and (d) BALM on Glass dataset.

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>BPNN</th>
<th>ABC-BP</th>
<th>ABC-LM</th>
<th>BALM</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU TIME</td>
<td>24.43</td>
<td>6894.25</td>
<td>4213.01</td>
<td>1418.11</td>
<td>161 %</td>
</tr>
<tr>
<td>EPOCHS</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>69</td>
<td>1349 %</td>
</tr>
<tr>
<td>MSE</td>
<td>0.271</td>
<td>0.173</td>
<td>0.055</td>
<td>0.001</td>
<td>16533 %</td>
</tr>
<tr>
<td>Accuracy</td>
<td>88.89</td>
<td>89.99</td>
<td>77.78</td>
<td>99.83</td>
<td>14 %</td>
</tr>
</tbody>
</table>

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, 77.78%, and 88.89%, respectively. The MSE convergence performance of the proposed BALM and other algorithms can be seen in Fig. 13.

Fig. 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 behavior 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. CONCLUSIONS

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. The work of Tutut Herawan is supported by University of Malaya Research Grant PRPUM CG063-2013 and Universitas Teknologi Yogyakarta Research Grant Ref number O7/UTY-R/SK/O/X/2013.

REFERENCES


**Nazri Mohd. Nawi** is Associate Professor at Faculty of Computer Science and Information Technology, University Tun Hussein Onn Malaysia. He obtained his Ph.D. in Computer Science (Data mining) from Swansea University United Kingdom, his MSc in Computer Science from the Universiti Teknologi Malaysia and his B.S. in Computer Science from the Universiti Sains Malaysia. Associate Professor Dr. Nazri Mohd Nawi has several years experience of working in Software Development Industry prior to joining University Tun Hussein Onn Malaysia. His research interests are in the field of data analysis, database system, optimization methods and data mining techniques using artificial neural network.

**Muhammad Zubair Rehman** is Ph.D. researcher at Faculty of Computer Sciences and Information Technology, University Tun Hussein Onn Malaysia. He is currently working on hybrid metaheuristic and optimization algorithms under the supervision of Associate Professor Dr. Nazri Mohd. Nawi. He obtained his M.S. in IT from UTHM in 2012, and B.S. in Computer Science from PUCIT, University of the Punjab, Pakistan in 2008. He has worked for two years in Software Development Industry. He has one year experience as a Technology Journalist and has written Computer Gadget reviews in many prestigious Magazines in Pakistan. His research interests are in the field of data mining, optimization, hybrid neural networks, and noise and vibration hazards etc.

**Abdullah Khan** is Assistant Professor in The University of Agriculture, Peshawar, Pakistan. He received his Ph.D. degree in Information Technology from University Tun Hussein Onn Malaysia, Malaysia in 2015. He previously got his M.Sc. in Computer Science from University of Science and Technology, Bannu, Pakistan. He has published several research articles in the field of optimization and metaheuristics. His main research interests include hybrid neural networks, knowledge based systems, and data mining.
**Arslan Kiyani** is currently pursuing his Ph.D. research in Macquarie University, Australia in Antenna Design. He obtained his Masters in Electrical Engineering from University Tun Hussein Onn Malaysia, Malaysia in 2014. He obtained his B.Sc. degree from COMSATS Institute of Technology, Islamabad Pakistan. His main research interests include antenna design, and optimization.

**Haruna Chiroma** received B. Tech. and M.Sc. in Computer Science from Abubakar Tafawa Balewa University and Bayero University Kano, respectively. Chiroma is currently a PhD candidate waiting for VIVA in the Artificial Intelligence Department of the University of Malaya and has published over 80 articles relevance to his research interest. He is also an invited reviewer to Applied Energy (Elsevier), Engineering Science and Technology (Elsevier), International Journal of Pervasive Computing and Communications (Emerald), etc.

**Tutut Herawan** received a Ph.D. in Computer Science from Universiti Tun Hussein Onn Malaysia in 2010. Currently, he is a Senior Lecturer at Department of Information System, University of Malaya. He has successfully co-supervised 3 Ph.Ds. and currently supervised 8 Ph.Ds. He has published many papers in various international journals, conference proceedings and book chapters. He has been appointed as editorial board member for MJCS (SCIE), IJDTA, TELKOMNIKA (scopus), IJNCAA, IJDCA and IJDIWC. He has also been appointed as a reviewer of several international journals and guest editor for several special issues such as TELKOMNIKA journal, Applied Soft Computing Elsevier, and IJMISSP Inderscience. He has served as a chair, PC chair, program committee member and co-organizer for numerous international conferences/workshops. His research area includes data mining and knowledge discovery, decision support in information system, rough and soft set theory.