Journal of Theoretical and Applied Information Technology
31st December 2017. Vol.95. No 24
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

USING MODIFIED BAT ALGORITHM TO TRAIN NEURAL NETWORKS FOR SPAM DETECTION

[1] AMAN JANTAN, [2]WAHEED A. H. M. GHANEM, [3]SANAA A. A. GHALEB

[1, 2] School of Computer Sciences, Universiti Sains Malaysia (USM), Penang, Malaysia

[2] Faculty of Engineering & Faculty of Education-Saber, University of Aden, Aden, Yemen

[3]Center for Instructional Technology and Multimedia, Universiti Sains Malaysia

E-mail: [1]aman@cs.usm.my, [2]waheed.ghanem@gmail.com, [3]saag15_sim002@student.usm.my

ABSTRACT

Nowadays a monumental amount of spam and junk email clutter email inboxes and storage facilities. Spam email has a significant negative impact on individuals and organizations alike, and is a serious waste of resources, time and effort. The task of filtering spam or junk e-mail is complex and very difficult to solve. Hence, learning-based filtering is considered an important method for detecting spam emails as the filtering technique requires training to epitomize the knowledge that can be used for detecting the spam. Thus, Artificial Neural Networks are being relied on to create a learning based filter. In this article, we particularly propose the Feedforward Neural Network (FFNN) for identification of e-mail spam; the weights and biases of this network model are set to optimum using a new modified bat algorithm (EBAT). Experiments and results based mainly on two datasets (SPAMBASE and UK-2011 WEBSPAM datasets) show that the developed FFNN model trained by EBAT achieves high generalization performance compared to other optimization methods.

Keywords: *Artificial Intelligent (AI), Swarm Intelligence (SI), Feed-forward Neural Network (FFNN), Bat Algorithm (BAT), Spam Email, Spam Detection.*

1. INTRODUCTION

It is a common occurrence for a user to receive heaps of emails daily of which 92% are spam [1]. This includes advertisements for a variety of merchandise and services, such as pharmaceuticals, electronics, software, jewelry, stocks, gambling, loans, pornography, phishing, and malicious attempts [2]. Not only does the spam consume users' time by forcing them to identify unwelcomed messages, but also wastes mailbox space and network bandwidth. Therefore, spam detection is posing a tremendous prerequisite and challenge at the same time to individuals as well as organizations.

In brief, spam can be defined as irrelevant or unsolicited messages sent in a large volume over the internet that negatively affects networks bandwidth, servers storage, user time and productivity [3]-[6]. In the internet context, spammers usually exploit several applications including email systems, social network platforms, web blogs, web forums and search engines [7]. The email spam is often used for advertising products and services typically related to adult entertainment, quick money and other attractive merchandises [8]. In a single affiliate program, it is estimated that, spammer revenue could top one million dollars per month [9]. Hence, statistics show that there is a general consensus towards criminalizing commercial spam.

Moreover, the percentage of spam containing malicious contents have recently increased compared to the one advertising legitimate products and services [10]. Several attacks, such as phishing, cross-site scripting, cross-site request forgery and malware infection utilize email spam as part of their attack vectors [11].

Email spam is the focus of this paper since it is the most common form of spam. Even with overlooking the complexity of spam detection, it is founded on the assumption that the spam's content differs from that of a legitimate email in ways that can be quantified. However, the detection accuracy is affected by several factors including the subjective nature of spam, obfuscation, language issues, processing overhead and message delay, and the irregular cost of filtering errors [12].

Broadly, two approaches are used for spam detection; rule-based filtering and learning-based filtering. In rules-based filtering, various parts of the messages such as header, content and source address are inspected in order to create patterns and populate the detection database. Received email messages are analyzed against these rules, and if a pattern matches any of the detection policies, then the message is classified as a spam. This approach requires a large number of rules to be affective. Moreover, rule-based filter could be evaded by forging the source of email and/or disguising the mail content [5] [12].

On the other side, learning-based filter is trained to excerpt the knowledge that can be used to detect the spam. This requires a large email dataset with both spam and legitimate ones. Most of these filters use Machines Learning (ML) algorithms such as Naive Bayes Classifier [13], Support Vector Machines [14] and Artificial Neural Networks [3].In addition, several ML techniques are merged together for a more accurate detection [15]-[17]. For instance, Artificial Neural Network (ANN) is a commonly used technique as it produces accurate classification results [6] [15] [16] [18] [19]. ANNs are inspired by the biological neural systems. The most popular and applied type of ANNs is the Feedforward Neural Network (FFNN).

FFNN model requires a considerable time for parameter selection and training [20] which has incentivized researchers to look for ways to optimize the process. Conventionally, FFNN networks are optimized by gradient based techniques such as Backpropagation algorithm. However, gradient based techniques are infamous of suffering some major setbacks namely slow convergence, high dependency on the initial parameters and the high probability of trapping in local minima [21]. Therefore, many researchers have proposed more stochastic methods for training FFNNs that are based on generating a number of random solutions for any given problem. The nature-inspired metaheuristic algorithms are an example of stochastic methods that are becoming more popular in training neural networks. In this category the following algorithms fall: Genetic Algorithm (GA) [18], Differential Evolution (DE) [22], Ant Colony Optimization (ACO) [23], Particle Swarm Optimization (PSO) [24], and Bat algorithm [25].

In order to optimize the performance of identifying spam the authors in [18] suggested training FFNN networks with Genetic Algorithm. The results have been promising as the hybridized method has outperformed the traditional FFNN neural network.

In this article, we develop a FFNN neural network model that is being trained with our new enhanced bat algorithm based Optimization (EBAT) [26], which was previously developed for identifying e-mail spam. EBAT is a recently developed metaheuristic algorithm inspired by the bat natural process. In this work, FFNN is trained using EBAT based on two different spam datasets and compared with other FFNNs trained with the common metaheuristic algorithms: ACO, BAT, DE, GA and PSO.

This article is organized as follows: Section 2 gives a broad description of feed-forward artificial (FFNN) Neural Networks, the enhanced bat algorithm (EBAT), EBAT for training feed-forward neural network, and in finally the datasets that used to evaluate the FFNN-EBAT approach; Section 3 exposes the experiments and analyzes the results obtained; and finally, Section 4 introduces the conclusions.

2. MATERIALS AND METHODS

2.1 Feed-Forward Artificial Neural Networks

An ANN comprises of an arrangement of preparing Unites Figure 1, otherwise called counterfeit neurons or nodes, which are interconnected with each other [27, 28]. Yield of the i^{th} artificial neuron can be depicted by Equation (1). Each simulated neuron gets inputs (signals) either from the earth or from different ANs. To each info (flag), xi is related a weight, w_{ij} to reinforce or drain the information flag. The ANs processes the net info flag, and uses an initiation work f_i, to figure the yield flag, i^{th} given the net information. Where , y_i is the yield of the neuron, xi is the i^{th} input to the neuron, w_{ij} is the association weight between the neuron and info x_i, θ_i is the limit (or inclination) of the neuron, and f_i is the neuron actuation work. For the most part, the neuron initiation work is a nonlinear capacity, for example, a heaviside work, a sigmoid capacity, a Gaussian capacity, and so forth.

$$y_i = f_i\left(\sum_{i=1}^{n} W_{ij} x_i + \theta_i\right) \tag{1}$$

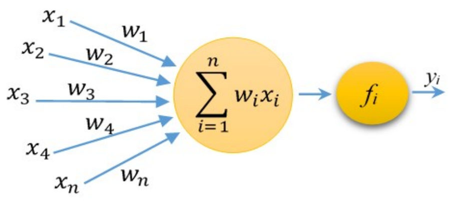


Figure 1: The Processing Unit (Neuron)

Feed-forward neural networks have been widely used, with two layers of the FFNNs (Fig. 2). Actually, FFNNs with two layers are the most popular neural network in practical applications such as approximate functions [29, 30, 31], and it is suitable for classifications of nonlinearly separable patterns [32, 33]. It has been proven that two layer FNNs can approximate any continuous and discontinuous function.

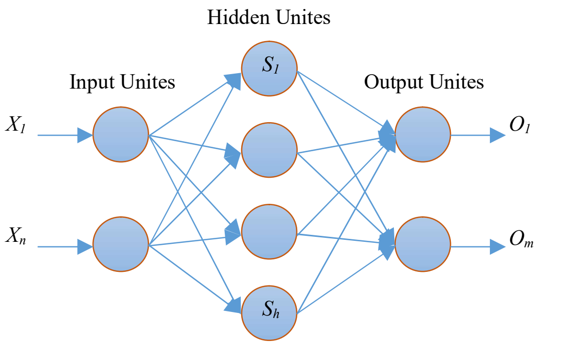


Figure 2: A Two-Layered Feed-forward Neural Network Structure

Figure 2. Demonstrates a FFNN with two layers (one input, one hidden, and one output layer), where the number of input neurons is equal to ***n***, the number of hidden neurons is equal to ***h***, and the number of output neurons is ***m***. The most vital undertakings that ought to be centered on, when utilizing FFNNs include: First, to obtain an improvement in the method of finding a combination of weights and biases which provide the minimum error for a FFNN. Second errand is to locate an appropriate structure for a FFNN. Last errand is to utilize an evolutionary algorithm to adjust the parameters of a gradient-based learning algorithm, such as the learning rate and momentum [21]. According to [21, 31, 34], the convergence of the BP algorithm is highly dependent on the initial values of weights, biases, and its parameters. These parameters incorporate learning rate and momentum. In the literature, utilizing novel heuristic optimization methods or evolutionary algorithms is a popular solution to enhance the problems of BP-based learning algorithms.

2.2 The Enhanced Bat Algorithm

This section presents the Enhanced BAT (EBAT) algorithm, which depends on the standard BA as introduced in the previous work [25, 26]. The standard bat algorithm has the ability to exploit the search space, however, it also at times falls into the trap of local optima, which affects its performance with respect to global search. In order to avoid trapping into local optima in BA, there is a need to increase the diversity of the search.

The fundamental thought behind the algorithm we introduce in this article is to augment the BA with a very effective operator. This operator is a set of random based modifications that aim to increase the diversity of BA and allow for more mutations in the inspected solutions within the BA search, and hence jump out of potential local optima traps. In other words, the BA's ability to exploit solutions in the local neighborhood is backed with the ability to explore new areas in the search space. The difference between EBAT and BA is that the added mutation operator is used to improve the original BA generating a new solution for each bat. In the light of this rule, an *exploitation* and *exploration* are two important crucial characteristics in the design of an effective optimization algorithm [35, 36, 37].

A minor change to the proposed algorithm is that we use fixed loudness **A** instead of various loudness A_i^t. Similar to BA, each bat in EBAT is defined by its position x_i^t, velocity v_i, the emission pulse rate r_i^t and the fixed loudness in a **d**-dimensional search space. The new solutions $x_i^t$and velocities v_i^t at time step ***t*** are given by equations (2), (3) and (4). The main improvement to the proposed algorithm is to add the mutation operator in order to increase the diversity of the population to improve the search efficiency and speed up the convergence to the optimal value.

$$f_i = f_{min} + (f_{max} - f_{min})\beta, \tag{2}$$

$$v_i^t = v_i^{t-1} + (x_i^t - x_*)f_i, \tag{3}$$

$$x_i^t = x_i^{t-1} + x_i^t, \tag{4}$$

$$x_{new} = x_{old} + \varepsilon A^t \tag{5}$$

$$A_i^{t+1} = \alpha A_i^t, \qquad r_i^{t+1} = r_i^0[1 - exp(-\gamma t)], \tag{6}$$

The proposed algorithm is similar to the standard BA on the side of local search: a new solution is first obtained by a local random walk from the best available solution (Eq. 5).The generation of this first solution is subject to the condition that a random real number drawn from a uniform distribution is larger than the *pulse rate* parameter.

The new mutation operator in the EBAT algorithm offers a new pair of tuning parameters, *Limit1* and *Limit2*, based on the previous researches [35-38]. The research [38] was based on the hybridization of the harmony search algorithm with the standard bat algorithm. In the mutation operator stage, if a random value is less than the value of Limit1, then a solution x_v^t is randomly chosen from the population of NP as shown in Eq. 8.

$$v_r = rand * NP \quad (7)$$

$$x_v^t = x_{v_r}^t \quad (8)$$

Where the r ∈ (1, 2… NP);

Further, if a random value is less than Limit2, more mutation is introduced into the elements of the current solution, drawing the search back to a better position with respect to the best and worst solutions recorded so far. This mutation proves very useful in case the BA component traps in a local optimum that is far from the actual global one. The modification of the mutation operator is formulated in Equations (9) and (10).

$$x_v^t = 0.5 \times (x_{worst}^t - x_v^t) \times rand[0,1) \quad (9)$$

$$x_v^t = 0.5 \times (x_v^t - x_{best}^t) \times rand[0,1) \quad (10)$$

In the equations above, x_v^t is a new solution of the tth iteration; x_v^t is the random solution selected by Eq. (8); and variables x_{worst}^t and x_{best}^t represent the worst and best solutions ever found, respectively. Otherwise, the randomization rule intends to add population diversity, it helps the mutation operator to explore the search space very efficiently, leading to increase the probability of finding the global optimal solution. Therefore, the randomization rule generates a new value for the ith element in the individual x_v^t as illustrated in Eq. 11.

$$x_v^t = Lb + rand\ (Ub - Lb) \quad (11)$$

The introduced mutation maintains the attractive features of the original bat algorithm, especially in terms of fast convergence, while allowing the algorithm to make use of more mutation towards a better diversity. Based on the aforementioned analyses, the pseudocode of the EBAT algorithm is shown in *Algorithm 1*.

Algorithm 1

```
Begin
Step 1: Initialization
        Set the generation counter $t = 1$; Initialize the population of NP bats P randomly where each
        bat corresponds to a potential solution for the given problem; define loudness A, pulse
        frequency $f_i$ and the initial velocities $v_i$; set pulse rate $r$, parameter Limit1 and limit2.
Step 2: evaluate the all the elements population by objective function f(x) and select the best and worst
solution of population NP.
Step 3: While the termination criterion is not satisfied or (t < MaxGeneration) do
For i = 1 to NP (all bat) do
  Generate a new solution by adjusting frequency, and updating velocity and location/solution by:
        $f_i = f_{min} + (f_{max} - f_{min})\beta,$
        $v_i^t = v_i^{t-1} + (x_i^t - x_*)f_i,$
        $x_i^t = x_i^{t-1} + v_i^t,$
  If (rand > r) then
    Select a solution among the best solutions $x_*$;
    Generate a local solution around the selected best solution by:
    $x_i^t = x_* + \varepsilon A$
  End if
  If (rand <Limit1) then
    $v_r = rand * NP;$
    $x_v^t = x_{v_r}^t$; Where the $v_r$ ∈ (1, 2, …, NP)
    If (rand <Limit2) then
      Update the solution by Eq. (10),
      $x_v^t = 0.5 \times (x_{worst}^t - x_v^t) \times rand[0,1)$;
    Else
      Update the solution by Eq. (11);
      $x_v^t = 0.5 \times (x_v^t - x_{best}^t) \times rand[0,1)$;
    End if
  Else
      $x_v^t = Lb + rand\ (Ub - Lb)$
  End if
  Evaluate the solution $x_i^t$ and $x_v^t$ by objective function f(x), and select the solution $x_{new}^t$
  which is has best fitness among the $x_i^t$ and $x_v^t$.
  Generate a new solution by flying randomly
  If (rand <A) and ($x_i$< f ($x_*$))
    Accept the new solution;
  End if
  Update the best and worst parameter.
  Rank the bats and find the current best solution $x_*$.
        $t = t + 1;$
Step4: End while
Step 5: Post-processing the results and visualization.
End
```

2.3 EBAT For Training Feed-Forward Neural Networks

In the last years, many of the researchers have used a heuristic algorithm in order to train the feed forward neural networks. And replaced the traditional algorithm with the heuristic algorithm, which showed better results than the traditional algorithm. There are three methods of using a heuristic algorithm for training FFNNs, these methods are as follows:

1. It is utilized for finding a combination of weights and biases which provide the minimum error for an FFNN.
2. It is utilized to find a proper structure for an FFNN in a particular problem.
3. It is utilized to use an evolutionary algorithm to tune the parameters of a gradient-based learning algorithm, such as the learning rate and momentum.

When utilizing artificial neural networks, the first step which must be carried out, is to determine the fixed structure for the neural Network, which will be trained by the training algorithm. The main objective of this algorithm is to find the appropriate values for all connection weights and biases, in order to reduce error rate in FFNNs. Besides this it is possible that a training algorithm is applied to an FFNN to determine the best structure for a certain problem. Which is made by manipulating the connections between neurons, the number of hidden layers, and the number of hidden neurons in each layer of the FFNN.

2.3.1 The Two-Layered Feed-Forward Neural Network

In this article, our work is based on training an artificial neural network, to find the appropriate values for all weights and biases in FFNNs. The algorithms used in this work are ACO, BAT, DE, EBAT, GA, and PSO. These mechanisms are called FFNN-ACO, FFNN-BAT, FFNN-DE, FFNN-EBAT, FFNN-GA, and FFNN-PSO, respectively. ACO, BAT, DE, EBAT, GA, and PSO are used to find a combination of weights and biases which yield the minimum error for the FFNN.

The structure of the FFNN is fixed; with two layered structures. Suppose that the input layer has *n* neurons; the hidden layer has *h* hidden neurons and the output layer has *m* output neurons. Figure 2 shows the structure of a two layered feed-forward neural network. According to the figure, a corresponding fitness function was given.

Assuming that the hidden transfer function is sigmoid function, and the output transfer function is a linear activation function. The Fitness function using the error of the FFNN should be defined to evaluate fitness in FFNN-ACO, FFNN-BAT, FFNN-DE, FFNN-EBAT, FFNN-GA, and FFNN-PSO. An encoding strategy should be defined to encode the weights and biases of the FFNN for the agents of FFNN-ACO, FFNN-BAT, FFNN-DE, FFNN-EBAT, FFNN-GA, and FFNN-PSO. These elements are described below:

2.3.2 Fitness Function

We follow the same manner that used in [21, 31] in order to calculate the fitness function. From figure 2, we have seen that FFNNs with two layers contain one input, one hidden, and one output layer; the number of input neurons is equal to (*n*), the number of hidden neurons is equal to (*h*), and the number of output neurons is (*m*). The output of the *i*th hidden node is calculated as follows:

$$f(S_j) = 1/\left(1 + exp\left(-\left(\sum_{i=1}^{n} W_{ij}.X_i - \theta_j\right)\right)\right) \quad (12)$$

Where $S_j = \sum_{i=1}^{n} W_{ij}.X_i - \theta_j,\ j = 1, 2, \ldots, h$, n is the number of the input neurons , W_{ij} is the connection weight from the *i*th node in the input layer to the *j*th node in the hidden layer, θ_j is the bias (threshold) of the *j*th hidden node, and $X_i$is the *i*th input. After calculating outputs of the hidden neurons, the final output can be defined as follows:

$$O_k = \sum_{i=1}^{n} W_{kj}.f(S_j) \ - \theta_k \quad (13)$$

Where W_{kj} , $k = 1, 2, \ldots, m$,is the connection weight from the *j*th hidden node to the *k*th output node and θ_k is the bias (threshold) of the *k*th output node. Finally, the learning error E (fitness function) is calculated as follows:

$$E_k = \sum_{i=1}^{m} \left(o_i^k - d_i^k\right)^2 \quad (14)$$

$$E = \sum_{k=1}^{q} \frac{E_k}{q} \quad (15)$$

Where q is the number of training samples, d_i^k is the desired output of the *j*th input unit when the *k*th training sample is used, and O_i^k is the actual output of the *i*th input unit when the *k*th training sample is used.

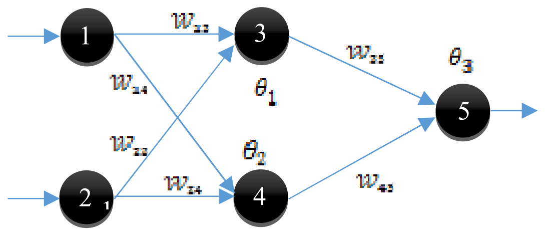


Figure 3: FFNN With A 2-2-1 Structure

Therefore, the fitness function of the ith training sample can be defined as follows:

$$\text{Fitness}\ (x_i) = E\ (x_i) \quad (16)$$

2.3.3 Encoding Strategy

Is a strategy used to represent the weights and biases of the FFNN [21, 31], we use it to represent the weights and biases for agents of the six algorithms FFNN-ACO, FFNN-BAT, FFNN-DE, FFNN-EBAT, FFNN-GA, and FFNN-PSO. For this, each agent represents all the weights and biases of the FFNNs structure. There are three strategies for representing the weights and biases of FFNNs for every agent in evolutionary algorithms (EA). Those strategies are the vector, matrix, and binary encoding strategies. In vector encoding, every agent is encoded as a vector to train an

FFNN, in matrix encoding, every agent is encoded as a matrix. In binary encoding, agents are encoded as binary bits.

In this article, we use the matrix encoding strategy because, this strategy is very suitable for the training processes of neural networks, also, the encoding strategy makes it easy to execute decoding for neural networks [21, 31]. As example, we execute encoding strategy for the FFNN on Figure 3, which is appears as follows:

$$\text{Agent}\ (;;\ i) = [w_1,\ b_1,\ w_2,\ b_2] \qquad (22)$$

$$w_1 = \begin{bmatrix} w_{13} & w_{23} \\ w_{14} & w_{24} \end{bmatrix}, b_1 = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}, w_2 = \begin{bmatrix} w_{35} \\ w_{45} \end{bmatrix}, b_2 = [\theta_3] \qquad (23)$$

Where w_1 is the weight matrix for the hidden layer, b_1 is the bias matrix for the hidden layer, w_2 and b_2 are the weight & bias, respectively, for output layer.

2.4 Datasets

In this article, the proposed FFNN-EBAT is evaluated in the identification of spam email using two different datasets: the SPAM dataset and the UK-2011 WEBSPAM dataset. The first dataset is obtained from the University of California at Irvine (UCI) Machine Learning Repository [39]. This dataset contains 57 features and 4601 instances of which approximately 39.4% are spam emails. The grouping of features in this set of data has been based on the frequency of some selected words and special characters in the e-mails. The second dataset, UK-2011, consist of 3766 instances with 11 features. Each example in the data is labeled as Ham or Spam. The data include 1768 ham emails and 1998 spam emails. The percentage of spam emails form approximately 53% of the emails, which makes the data imbalanced and therefore more challenging. The full description of the features can be found in [40].

3. EXPERIMENTAL RESULTS AND DISCUSSION

In this section, we layout the experimental setup through which we have evaluated the proposed algorithm, FFNN-EBAT. Following the usual methodology in the metaheuristic literature, we expose the algorithm to two datasets related to spam email, which are described in the previous section 2.4, and compare the performance of the algorithm with other known and published metaheuristic techniques. In this work, we have implemented six sets of experiments (three for each dataset), comparing the performance of the proposed algorithm with different number of neurons in the hidden layer of the trained network. One experiment uses the same number of input neurons in the hidden layer, the second experiment uses (number of input neurons * 2 + 1) neurons for the hidden layer, while the last experiment uses (number of input neurons * 4 + 1) neurons in the hidden layer.

All the experiments were conducted on a laptop with an Intel® Core™ i5-2430 CPU @ 2.40 GHz, and equipped with 8 GB of RAM. Our implementations of the algorithms were compiled using MATLAB R2009b (V7.9.0.529) running under Windows 7 Home premium SP1.The software implementation of the proposed FFNN-EBat algorithm was based on the implementation of EBat in [26]. In all experiments, the population size *NP* was set to 50. The maximum number of generations was 50. To mitigate the impact of randomness in individual runs, we report the results over a 10 implementation runs for each algorithm on each dataset.

The proposed FFNN-EBAT algorithm inherits a set of control parameters from its underpinning algorithm, BA, and introduces a couple of random control parameters. The set of FFNN-EBAT'S parameters include the *loudness*, which was set to 0.95, *pulse rate*, which was set to 0.1, and the new *limit1* and *Limit2* parameters, which were set to 0.8 and 0.5, respectively. These are the default parameter values in all our experiments. The choice of the control parameters is very important for the performance of the optimization metaheuristic and can vary with different applications. The parameters of all methods used in this work are presented below:

- **ACO**: The ACO method involves many parameters, which were set as follows: pheromone update constant Q = 20, local pheromone decay rate $q_l = 0.5$, global pheromone decay rate $q_g = 0.9$, exploration constant $q_0 = 1$, pheromone sensitivity $s = 1$, visibility sensitivity $b = 5$, and initial pheromone value $s_0 = 1\text{E-}6$.
- **GA**: This method uses the roulette wheel in the selection phase, while in the crossover phase it uses a single point with probability of one and the mutation probability is equal to 0.01.
- **DE**: This method is based on two parameters, the area crossover constant CR = 0.5 and the weighting factor F = 0.5.
- **PSO**: the population size in this method is also set to 50, inertial constant = 0.3, cognitive constant = 1, and social constant for swarm interaction = 1.

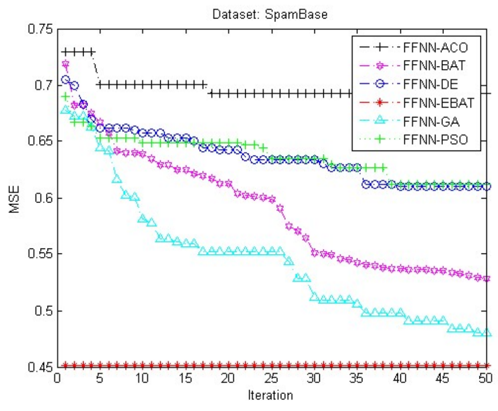
3.1 Experiment 1: *Performance analysis of the proposed approach (FFNN-EBAT) against five other methods with the SPAMBASE email dataset.*

In this experiment, we compare our approach with the other five approaches, against the Spam email dataset, which has been mentioned in section 2.4. The experiments in this section involve three sets of results, corresponding to three experiments based on diversity in the number of neurons in the hidden layer (57, 115, and 229) for each approach. We compare FFNN-ACO, FFNN-BAT, FFNN-DE, FFNN-GA, FFNN-PSO, and FFNN-EBAT based on the *best*, *mean*, *median*, and *standard deviation* of the Mean Square Error (MSE) for training samples (Spam Email Dataset) over 10 independent runs. The criterion for finishing the training process is to complete the maximum number of iterations (equal to 50 in this article) and the population size is 50. The experimental results for Spam Email Dataset are listed in Table 1. The best results are indicated in bold type.

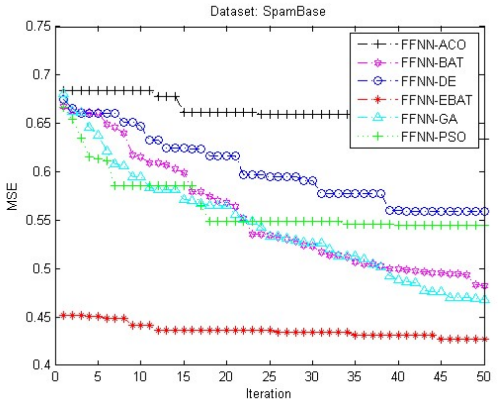
We first show a representative sample of the convergence plots of FFNN-ACO, FFNN-BAT, FFNN-DE, FFNN-GA, FFNN-PSO, and FFNN-EBAT in Fig. 4a–c, before discussing the results in Table 1. These figures show the best Mean Square Error (MSE) values found by each compared approach for all training samples, per search iteration, over the 50 maximum generations. Figure 4a shows the results reached by the six approaches when applied to Spam Email dataset with 57 neurons in the hidden layer. We can observe from the figure that FFNN-EBAT is significantly superior to the other approaches over the process of optimization in terms of both convergence speed and final result, while the FFNN-GA approach performs the second best outperforming the FFNN-BAT approach on this dataset.

Figure 4b shows the results achieved by all the approaches against Spam Email Dataset with 115 neurons in the hidden layer. The FFNN-EBAT approach shows a notable better convergence and final result in this dataset as well. The rate of convergence in the case of FFNN-GA and FFNN-BAT is similar overall, but there is a significant difference in the final result. From the figure, the FFNN-EBAT obviously outperforms the other approaches.

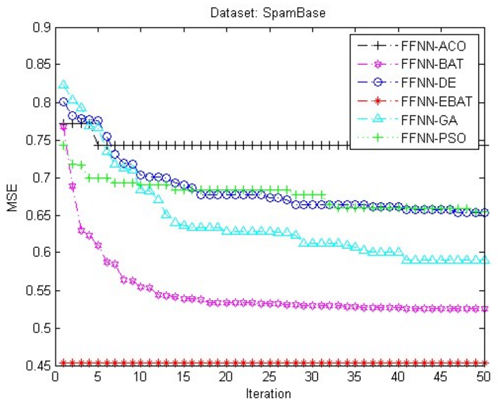
Finally, Fig. 4c depicts the results obtained from applying Spam Email Dataset with 229 neurons in the hidden layer to the six approaches. FFNN-EBAT in this experiment shows again a very fast convergence to a superior final optimum and outperforms the other approaches. The FFNN-BAT approach performs the second best after FFNN-EBAT on this dataset



(a)



(b)



(c)

Figure 4: Convergence Curves Of All Approaches Based On Averages Of MSE For All Training Samples Over 10 Independent Runs Against SPAMBASE Dataset. (A), (B), And (C) Are The Convergence Curves For Fnns With S = 57, 115, And 229, Respectively.

From Table 1, it can be seen that the FNN-EBAT performs better than the FFNN-ACO, FFNN-BAT, FFNN-DE, FFNN-GA, and FFNN-PSO for the *mean*, *median*, and *standard deviation* of MSE. The results of these statistical variables prove that FNN-EBAT has the best ability to avoid local minima.

FFNN-GA has better results in terms of the *best* MSE when the number of neurons in the hidden layer is 57, while FFNN-BAT records *best* MSE when the number of hidden layer neurons is 229. Overall, the results show that FNN-EBAT is more accurate than other approaches.

Table 1: Best, Median, Standard Deviation, And Mean Of MSE For All Training Samples Over 10 Independent Runs For Six Approaches Against SPAMBASE Dataset.

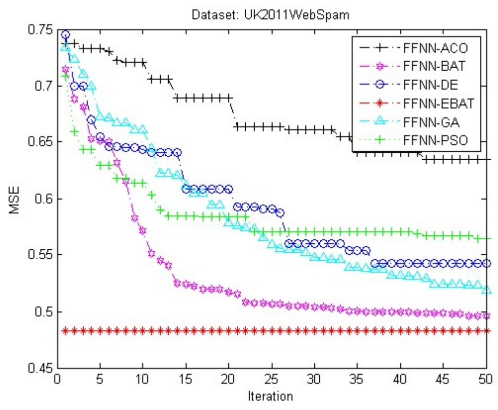
No. of neurons in hidden layer = 57				
Alg.	Best	Median	Mean	Std. dev.
ACO	6.07E-01	6.89E-01	6.93E-01	7.00E-02
BAT	4.78E-01	5.10E-01	5.29E-01	5.30E-02
DE	5.26E-01	6.23E-01	6.10E-01	4.86E-02
GA	**4.48E-01**	5.71E-01	5.44E-01	6.50E-02
PSO	5.82E-01	6.27E-01	6.12E-01	2.38E-02
EBAT	4.49E-01	**4.52E-01**	**4.51E-01**	**1.47E-03**
No. of neurons in hidden layer = 115				
Alg.	Best	Median	Mean	Std. dev.
ACO	5.26E-01	6.48E-01	6.34E-01	6.22E-02
BAT	3.64E-01	4.72E-01	4.83E-01	1.07E-01
DE	5.18E-01	5.59E-01	5.59E-01	**3.87E-02**
GA	3.64E-01	4.61E-01	4.64E-01	8.73E-02
PSO	4.72E-01	5.36E-01	5.45E-01	5.19E-02
EBAT	**3.27E-01**	**4.52E-01**	**4.27E-01**	5.61E-02
No. of neurons in hidden layer = 229				
Alg.	Best	Median	Mean	Std. dev.
ACO	6.48E-01	7.47E-01	7.43E-01	6.26E-02
BAT	**4.43E-01**	5.70E-01	5.25E-01	6.78E-02
DE	6.44E-01	6.56E-01	6.53E-01	6.07E-03
GA	5.71E-01	5.90E-01	5.90E-01	1.38E-02
PSO	6.24E-01	6.47E-01	6.54E-01	2.94E-02
EBAT	4.52E-01	**4.52E-01**	**4.53E-01**	**1.36E-03**

3.2 Experiment 2: *Performance analysis of the proposed approach (FFNN-EBAT) against five approaches with the UK-2011 WEBSPAM dataset*

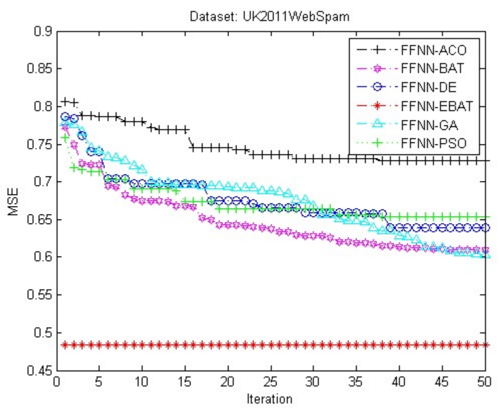
In this experiment, we compare our approach with the other five approaches against the UK-2011 WEBSPAM dataset, which has been mentioned in section 2.4. The experiments in this section encompass three sets of results, similar to the experiments presented in the previous section. The results correspond to three experiments based on different number of neurons in the hidden layer (11, 23, and 45) for each approach. The comparisons in this experiment are the same as those in the experiment 1. The only differences between the two are the used dataset and the number of neurons in the hidden layer.

Figure 5a shows the results reached by the six approaches when applied to UK-2011 WEBSPAM dataset with 11 neurons in the hidden layer. We can observe from the figure that FFNN-EBAT is significantly superior to the other approaches over the process of optimization in terms of both convergence speed and final result, while the FFNN-BAT approach performs the second best after our approach on this dataset.

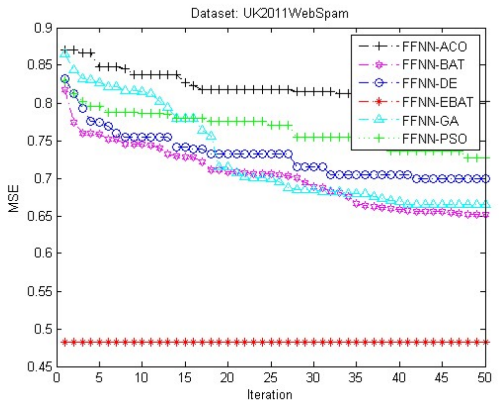
Figure 5b-c shows sample convergence plots of the same set of experiments described previously, with 23 and 45 neurons in the hidden layer, respectively. The convergence plots in both figures show that FFNN-EBAT has a superior performance to all other approaches in terms of fast convergence towards its final optimal value. These figures confirm that FFNN-EBAT seemingly has the best convergence rate for the FNNs with various number of neurons in the hidden layer.



(a)



(b)



(c)

Figure 5: Convergence Curves Of All Approaches Based On Averages Of MSE For All Training Samples Over 10 Independent Runs In UK-2011 WEBSPAM. (A), (B), And (C) Are The Convergence Curves For Fnns With S = 11, 23, And 45, Respectively

Table 2 shows the results for *best*, *median*, *standard deviation*, and *mean* of MSE for training samples over 10 independent runs. There are three sets of results, each corresponding to a different number of neurons in the hidden layer of the neural network (i.e. 11, 23, and 45). These results were obtained from experimenting against the UK-2011 WEBSPAM dataset. We can see from Table 2 and Figure 5 that our approach, FFNN-EBAT, outperforms the other approaches. The statistical variables from the three experiments proves that FNN-EBAT has the best ability to avoid local minima and the most optimum results, leading to a better accuracy of neural network classification.

Furthermore, we can see that in all the six experiment results, FNN-ACO does not record a good performance because of the slow searching process of the ACO algorithm, which affects FNN-ACO exploitation ability. Learning algorithms for FNNs need not only strong exploration ability but also precise exploitation ability. As also shown from the results, FFNN-DE and FFNN-PSO perform better than FNN-ACO due to the more precise exploitation ability of DE and PSO, but they are still suffering from the problem of trapping in local minima. This deficiency means that FFNN-DE and FNN-PSO have unstable performance. The results obtained by FNN-EBAT prove that it has both strong exploitation and good exploration abilities. In other words, the strengths of the EBAT algorithm have been successfully utilized, giving outstanding performance in training FNNs. In particular, FNN-EBAT is capable of solving the problem of trapping in local minima and gives fast convergence speed.

Table 2: Best, Median, Standard Deviation, And Mean Of MSE For All Training Samples Over 10 Independent Runs For Six Approaches In UK-2011 WEBSPAM Dataset.

No. of neurons in hidden layer = 11				
Alg.	Best	Median	Mean	Std. dev.
ACO	5.82E-01	6.17E-01	6.34E-01	4.44E-02
BAT	4.89E-01	4.89E-01	4.96E-01	9.12E-03
DE	5.04E-01	5.46E-01	5.42E-01	2.58E-02
GA	4.93E-01	5.08E-01	5.07E-01	1.18E-02
PSO	5.46E-01	5.60E-01	5.64E-01	1.78E-02
EBAT	**4.76E-01**	**4.84E-01**	**4.83E-01**	**3.87E-03**
No. of neurons in hidden layer = 23				
Alg.	Best	Median	Mean	Std. dev.
ACO	6.79E-01	7.39E-01	7.28E-01	4.64E-02
BAT	4.84E-01	5.89E-01	6.10E-01	1.13E-01
DE	5.95E-01	6.34E-01	6.39E-01	3.55E-02
GA	4.91E-01	5.70E-01	5.51E-01	3.55E-02
PSO	5.96E-01	6.24E-01	6.53E-01	5.50E-02
EBAT	**4.82E-01**	**4.83E-01**	**4.84E-01**	**1.81E-03**
No. of neurons in hidden layer = 45				
Alg.	Best	Median	Mean	Std. dev.
ACO	7.77E-01	8.03E-01	8.02E-01	1.66E-02
BAT	5.76E-01	6.44E-01	6.51E-01	6.43E-02
DE	6.80E-01	6.98E-01	6.99E-01	1.29E-02
GA	5.97E-01	6.67E-01	6.65E-01	6.03E-02
PSO	6.48E-01	7.29E-01	7.27E-01	5.64E-02
EBAT	**4.79E-01**	**4.84E-01**	**4.83E-01**	**2.50E-03**

4. CONCLUSION

The main contribution in this article is the use of a recent meta-heuristic algorithm called EBAT, which was developed from the original bat algorithm, in training feedforward neural networks for spam detection. The new developed approach FFNN-EBAT was applied to classify emails into normal or spam or junk e-mail based on a number of content-related features. The FFNN-EBAT algorithm was evaluated and compared with other neural networks trained by ant-colony optimization algorithm, bat algorithm, differential evolution algorithm, genetic algorithm, and particle swarm optimization. The experiments have shown that FFNN-EBAT has better quality results than the other training algorithms. In future works, we plan to verify the performance of the EBAT algorithm with other neural network types, and investigate its effectiveness with other spam email datasets.

5. ACKNOWLEDGMENTS

This research work was funded by the Fundamental Research Grant Scheme (FRGS) for "Content-Based Analysis Framework for Better Email Forensic and Cyber Investigation" [203/PKOMP/6711426]. The research was also partially supported by Universiti Sains Malaysia under USM Fellowship [APEX(308/AIPS/415401)] from Institute of Postgraduate Studies, Universiti Sains Malaysia.

REFRENCES:

[1] D. DeBarr, H. Wechsler, Spam detection using random boost, Pattern Recogn. Lett. Vol.33 (10), 2012, pp.1237–1244.

[2] S. Heron, Technologies for spam detection, Netw. Secur. 2009 (1), 2009, 11–15.

[3] Guzella, Thiago S., and Walmir M. Caminhas. "A review of machine learning approaches to spam filtering." Expert Systems with Applications Vol.36, no. 7, 2009, pp.10206-10222.

[4] Rao, Justin M., and David H. Reiley. "The economics of spam." The Journal of Economic Perspectives Vol.26, no. 3, 2012, pp.87-110.

[5] Stern, H., *et al.* "A Survey of Modern Spam Tools". CiteSeer, 2008.

[6] Su, Mu-Chun, Hsu-Hsun Lo, and Fu-Hau Hsu. "A neural tree and its application to spam e-mail detection." Expert Systems with Applications Vol.37, no. 12, 2010, pp.7976-7985.

[7] Kanich, Chris, Nicholas Weaver, Damon McCoy, Tristan Halvorson, Christian Kreibich, Kirill Levchenko, Vern Paxson, Geoffrey M. Voelker, and Stefan Savage. "Show Me the Money: Characterizing Spam-advertised Revenue." In USENIX Security Symposium, 2011, pp. 15-15.

[8] Cranor, Lorrie Faith, and Brian A. LaMacchia. "Spam!." Communications of the ACM 41, no. 8, 1998, pp.74-83.

[9] Stone-Gross, Brett, Thorsten Holz, Gianluca Stringhini, and Giovanni Vigna. "The Underground Economy of Spam: A Botmaster's Perspective of Coordinating Large-Scale Spam Campaigns." LEET 11, 2011, pp. 4-4.

[10] Gudkova, Darya, Tatiana Kulikova, Katerina Kalimanova, and Daria Bronnikova. "Kaspersky security bulletin." Spam Evolution (2013).

[11] Faris, Hossam, Khalid Jaradat, Malek Al-Zewairi, and Omar Adwan. "Improving knowledge based spam detection methods: The effect of malicious related features in imbalance data distribution." International Journal of Communications, Network and System Sciences 8, no. 5 (2015): 118.

[12] PéRez-DíAz, Noemí, David Ruano-OrdáS, Florentino Fdez-Riverola, and José R. MéNdez. "SDAI: An integral evaluation methodology for content-based spam filtering models." Expert Systems with Applications 39, no. 16 (2012): 12487-12500.

[13] Song, Yang, Aleksander Kołcz, and C. Lee Giles. "Better Naive Bayes classification for high-precision spam detection." Software: Practice and Experience 39, no. 11 (2009): 1003-1024.

[14] Amayri, Ola, and Nizar Bouguila. "A study of spam filtering using support vector machines." Artificial Intelligence Review 34, no. 1 (2010): 73-108.

[15] Zmyślony M, Krawczyk B, Woźniak M. Combined classifiers with neural fuser for spam detection. In International Joint Conference CISIS'12-ICEUTE´ 12-SOCO´ 12 Special Sessions 2013 (pp. 245-252). Springer Berlin Heidelberg.

[16] Manjusha K, Kumar R. Spam mail classification using combined approach of bayesian and neural network. InComputational Intelligence and Communication Networks

(CICN), 2010 International Conference on 2010 Nov 26 (pp. 145-149). IEEE.

[17] Idris, Ismaila, Ali Selamat, Ngoc Thanh Nguyen, Sigeru Omatu, Ondrej Krejcar, Kamil Kuca, and Marek Penhaker. "A combined negative selection algorithm–particle swarm optimization for an email spam detection system." Engineering Applications of Artificial Intelligence 39, 2015, pp. 33-44.

[18] Arram, Anas, Hisham Mousa, and Anzida Zainal. "Spam detection using hybrid Artificial Neural Network and Genetic algorithm." In Intelligent Systems Design and Applications (ISDA), 2013 13th International Conference on, IEEE, 2013, pp. 336-340.

[19] Xu, Hao, and Bo Yu. "Automatic thesaurus construction for spam filtering using revised back propagation neural network." Expert Systems with Applications 37, no. 1, 2010, pp.18-23.

[20] Yu, Bo, and Zong-ben Xu. "A comparative study for content-based dynamic spam classification using four machine learning algorithms." Knowledge-Based Systems 21, no. 4, 2008, pp.355-362.

[21] Ghanem, Waheed A.H.M., and Aman Jantan. "Using hybrid artificial bee colony algorithm and particle swarm optimization for training feed-forward neural networks." Journal of Theoretical & Applied Information Technology 67, no. 3, 2014, pp.664-674.

[22] Idris, Ismaila, Ali Selamat, and Sigeru Omatu. "Hybrid email spam detection model with negative selection algorithm and differential evolution." Engineering Applications of Artificial Intelligence 28, 2014, pp. 97-110.

[23] El-Alfy ES. Discovering classification rules for email spam filtering with an ant colony optimization algorithm. In Evolutionary Computation, 2009. CEC'09. IEEE Congress on 2009 May 18, pp. 1778-1783.

[24] Idris, Ismaila, and Ali Selamat. "Improved email spam detection model with negative selection algorithm and particle swarm optimization." Applied Soft Computing 22, 2014, pp.11-27.

[25] Natarajan, Arulanand, S. Subramanian, and K. Premalatha. "A comparative study of cuckoo search and bat algorithm for Bloom filter optimisation in spam filtering." International Journal of Bio-Inspired Computation 4, no. 2, 2012, pp.89-99.

[26] Ghanem, Waheed A.H.M., and Aman Jantan. "An enhanced Bat algorithm with mutation operator for numerical optimization problems." Neural Computing and Applications, 2017, pp.1-35.

[27] Yao, Xin. "Evolving artificial neural networks." Proceedings of the IEEE 87, no. 9 (1999): 1423-1447.

[28] Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." Neural networks 2, no. 5 (1989): 359-366.

[29] K. Homik, M. Stinchcombe, H. White, Multilayer feed-forward networks are universal approximators, Neural Networks 2 (1989) 359–366. [30]B. Malakooti, Y. Zhou, Approximating polynomial functions by feed-forward artificial neural network: capacity analysis and design, Appl. Math. Comput. 90, 1998, pp.27–52.

[31] SA Mirjalili, SZ Mohd Hashim. "Training feed-forward neural networks using hybrid particle swarm optimization and gravitational search algorithm." Applied Mathematics and Computation 218.22, 2012, pp.11125-11137.

[32] C. Lin, Cheng-Hung, C. Lee, A self-adaptive quantum radial basis function network for classification applications, in: IEEE International Joint Conference on Neural Networks, 2004, pp. 3263–3268.

[33] Isa, Nor Ashidi Mat, and Wan Mohd Fahmi Wan Mamat. "Clustered-hybrid multilayer perceptron network for pattern recognition application." Applied Soft Computing 11, no. 1, 2011, pp.1457-1466.

[34] J.R. Zhang, J. Zhang, T.M. Lock, M.R. Lyu, A hybrid particle swarm optimization–back-propagation algorithm for feed-forward neural network training, Appl. Math. Comput. 128, 2007, pp.1026–1037.

[35] Ghanem, Waheed A.H.M., and Aman Jantan. "Hybridizing artificial bee colony with monarch butterfly optimization for numerical optimization problems." Neural Computing and Applications, 2016, pp.1-19.

[36] Ghanem, Waheed A.H.M., and Aman Jantan. “Hybridizing Bat Algorithm with Modified Pitch-Adjustment Operator for Numerical Optimization Problems”, Modeling, Simulation, and Optimization Book Series, Springer International Publishing AG, 2018.

[37] Ghanem, Waheed A.H.M., and Aman Jantan. "A Novel Hybrid Artificial Bee Colony with Monarch Butterfly Optimization for Global Optimization Problems", Modeling, Simulation, and Optimization Book Series, Springer International Publishing AG, 2018.

[38] Wang, Gaige, and Lihong Guo. "A novel hybrid bat algorithm with harmony search for global numerical optimization." Journal of Applied Mathematics 2013.

[39] Lichman, M. (2013) UCI Machine Learning Repository.

[40] Fdez-Glez, Jorge, David Ruano-Ordás, Rosalía Laza, José Ramon Méndez, Reyes Pavón, and Florentino Fdez-Riverola. (2016) "WSF2: A novel framework for filtering web spam." Scientific Programming 2016: 1.