Appl Intell (2015) 43:150–161
DOI 10.1007/s10489-014-0645-7

How effective is the Grey Wolf optimizer in training multi-layer perceptrons

Seyedali Mirjalili

Published online: 17 January 2015

Abstract This paper employs the recently proposed Grey Wolf Optimizer (GWO) for training Multi-Layer Perceptron (MLP) for the first time. Eight standard datasets including five classification and three function-approximation datasets are utilized to benchmark the performance of the proposed method. For verification, the results are compared with some of the most well-known evolutionary trainers: Particle Swarm Optimization (PSO), Genetic Algorithm (GA), Ant Colony Optimization (ACO), Evolution Strategy (ES), and Population-based Incremental Learning (PBIL). The statistical results prove the GWO algorithm is able to provide very competitive results in terms of improved local optima avoidance. The results also demonstrate a high level of accuracy in classification and approximation of the proposed trainer.

Keywords Grey Wolf optimizer · MLP · Learning neural network · Evolutionary algorithm · Multi-layer perceptron

1 Introduction

Neural Networks (NN) are one of the greatest inventions in the field of Computational Intelligence. They mimic the neurons of human brain to mostly solve classification problems. The rudimentary concepts of NNs were first proposed in 1943 [1]. There are different types of NNs proposed in the literature: Feedforward network [2], Kohonen self-organizing network [3], Radial basis function (RBF) network [4], Recurrent neural network [5], and Spiking neural networks [6]. In Feedforward NNs (FNN) the information is cascaded in one direction throughout the networks. As its name implies, however, recurrent NNs share the information between the neurons in two directions. Finally, spiking NNs activates neurons with spikes.

Despite the differences of NNs, they are common in learning. Learning refers to the ability of a NN in learning from experience. Similarly to biological neurons, the Artificial NNs (ANN) have been equipped with mechanisms to adapt themselves to a set of given inputs. There are two common types of learning here: supervised .[7, 8] and unsupervised [9, 10]. In the former case, the NN is provided with feedbacks from an external source (supervisor). In unsupervised learning, however, a NN adapts itself to inputs (learn) without any extra external feedbacks.

Generally speaking, the method that provides learning for a NN is called trainer. A trainer is responsible for training NNs to achieve the highest performance for new sets of given inputs. In the supervised learning, a training method first provides NNs with a set of samples called training samples. The trainer then modifies the structural parameters of the NN in each training step in order to improve the performance. Once the training phase is completed, the trainer is omitted and NN is ready to use. The trainer can be considered as the most important component of any NNs.

There are two types of learning methods in the literature: deterministic versus stochastic. Back-Propagation (BP) [11] and gradient-based methods are considered as deterministic methods. In such techniques, the training phase results in the same performance if the training samples remain

S. Mirjalili (✉)
School of Information and Communication Technology, Griffith University, Nathan, Brisbane, QLD 4111, Australia
e-mail: seyedali.mirjalili@griffithuni.edu.au

S. Mirjalili
Queensland Institute of Business and Technology, Mt Gravatt, Brisbane, QLD 4122, Australia

consistent. The trainers in this group are mostly mathematical optimization techniques that aim to find the maximum performance (minimum error). In contrast, stochastic trainers utilize stochastic optimization techniques in order to maximize performance of a NN.

The advantages of the deterministic trainers are: simplicity and speed. Deterministic training algorithm usually starts with a solution and guides it toward an optimum. The convergence is very fast, but the quality of the obtained solution highly depends on the initial solution. In addition, there is a high probability of local optima entrapment. Local optima refer to the sub-optimal solutions in a search space that might mistakenly be assumed as the global optimum. The challenge here is the unknown number of runs that a trainer needs to be restarted with different initial solutions to find the global optimum.

In contrast, stochastic trainers start the training process with random solution(s) and evolve it (them). Randomness is the essential component of the stochastic trainers that apply to both initial solutions and method of solution's improvement during the training process. The advantage of such methods is high local optima avoidance, but they are mostly much slower than deterministic approaches [12, 13]. The literature shows that stochastic training methods have gained much attention recently, which is due to the high local optima avoidance.

Stochastic training algorithms can be divided into two groups: single-solution and multi-solution based algorithms [14]. In the former class, a trainer starts the training with a single randomly constructed NN and evolves it iteratively until the satisfaction of an end criterion. Examples of single-solution-based trainers are Simulated Annealing (SA) [15, 16] and hill climbing [17, 18]. In contrast to single-solution-based methods, a multiple-solution-based algorithm initiates the training process by a set of randomly created NNs and evolves them with respect to each other until the satisfaction of an end criterion. The literature shows that stochastic algorithms with multiple solutions provide higher ability of resolving local optima stagnation [19].

Some of the most popular multi-solution trainers in the literature are: Genetic Algorithm (GA) [19, 20], Particle Swarm Optimization (PSO) [21, 22], Ant Colony Optimization (ACO) [23, 24], Artificial Bee Colony (ABC) [25, 26], and Differential Evolution (DE) [27, 28]. The recent evolutionary training algorithms are: hybrid Central Force Optimization and Particle Swarm Optimization (CFO-PSO) [29], Social Spider Optimization algorithm (SSO) [30], Chemical Reaction Optimization (CRO) [31] Charged System Search (CSS) [32], Invasive Weed Optimization (IWO) [33] and Teaching-Learning Based Optimization (TLBO) trainer [34]. The reason as to why such optimization techniques have been employed as training algorithms is their high performance in terms of approximating the global optimum. This also motivates my attempts to investigate the efficiencies of my recently proposed algorithm called Grey Wolf Optimizer (GWO) [14] in training FNNs. In fact the main motivations of this work are twofold:

- Current multi-solution stochastic trainers still prone to local optima stagnation
- GWO shows high exploration and exploitation that may assist it to outperform other trainers in this field

The rest of the paper is organized as follows.

Section 2 presents the preliminary and essential definition of an MLP. The GWO algorithm is then presented in Section 3. The proposed GWO-based trainer is discussed in Section 4. Results and conclusion are provided in Sections 5 and 6 respectively.

2 Feed-forward neural network and multi-layer perceptron

As discussed in the preceding section, FNNs are those NNs with only one-way and one-directional connections between their neurons. In this type of NNs, neurons are arranged in different parallel layers [2]. The first layer is always called the input layer, whereas the last layer is called the output layers. Other layers between the input and output layers are called hidden layer. A FNN with one hidden layer is called MLP as illustrated in Fig. 1.

After providing the inputs, weights, and biases, the output of MLPs are calculated throughout the following steps [35]:

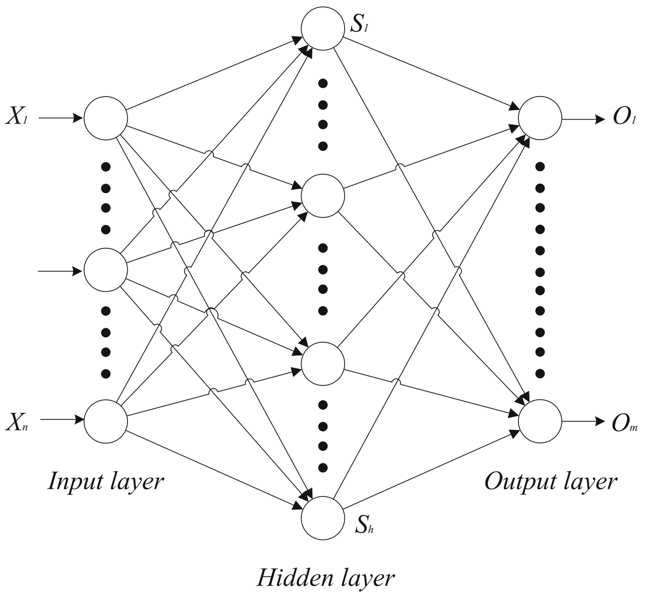

Fig. 1 MLP with one hidden layer

1. The weighted sums of inputs are first calculated by (2.1).

$$s_j = \sum_{i=1}^{n} (W_{ij}.X_i) - \theta_j, \quad j = 1, 2, \ldots h \tag{2.1}$$

where n is the number of the input nodes, W_{ij} shows the connection weight from the ith node in the input layer to the jth node in the hidden layer, θ_j is the bias (threshold) of the jth hidden node, and X_i indicates the ith input.

2. The output of each hidden node is calculated as follows:

$$S_j = sigmoid\left(s_j\right) = \frac{1}{\left(1 + \exp\left(-s_j\right)\right)}, \, j = 1, 2, \ldots h \tag{2.2}$$

3. The final outputs are defined based on the calculated outputs of the hidden nodes as follows:

$$o_k = \sum_{j=1}^{h} (w_{jk}.S_j) - \theta_k^{'}, k = 1, 2, \ldots, m \tag{2.3}$$

$$O_k = sigmoid\left(o_k\right) = \frac{1}{\left(1 + \exp\left(-o_k\right)\right)}, k = 1, 2, \ldots, m \tag{2.4}$$

where w_{jk} is the connection weight from the jth hidden node to the kth output node, and $\theta_k^{'}$ is the bias (threshold) of the kth output node.

As may be seen in the (2.1) to (2.4), the weights and biases are responsible for defining the final output of MLPs from given inputs. Finding proper values for weights and biases in order to achieve a desirable relation between the inputs and outputs is the exact definition of training MLPs. In the next sections, the GWO algorithm is employed as a trainer for MLPs

3 Grey Wolf Optimizer

The GWO is a recently proposed swarm-based meta-heuristic [14]. This algorithm mimics the social leadership and hunting behaviour of grey wolves in nature. In this algorithm the population is divided into four groups: alpha (α), beta (β), delta (δ), and omega (ω). The first three fittest wolves are considered as α, β, and δ who guide other wolves (ω) toward promising areas of the search space. During optimization, the wolves update their positions around α, β, or δ as follows:

$$\vec{D} = \mid \vec{C} \cdot \overrightarrow{X_p}(t) - \vec{X}(t) \mid \tag{3.1}$$

$$\vec{X}(t+1) = \overrightarrow{X_p}(t) - \vec{A} \cdot \vec{D} \tag{3.2}$$

where t indicates the current iteration, $\vec{A} = 2a \cdot \vec{r}_1 a$, $\vec{C} = 2 \cdot \vec{r}_2$, $\vec{X}_p$ is the position vector of the prey, $\vec{X}$ indicates the position vector of a grey wolf, a is linearly decreased from 2 to 0, and $\vec{r}_1$, $\vec{r}_2$ are random vectors in [0,1].

The concepts of position updating using (3.1) and (3.2) are illustrated in Fig. 2. It may be seen in this figure that a wolf in the position of (X, Y) is able to relocate itself around the prey with the proposed equations. Although seven of the possible locations have been shown in Fig. 2, the random parameters A and C allow the wolves to relocate to any position in the continuous space around the prey.

In the GWO algorithm, it is always assumed that α, β, and δ are likely to be the position of the prey (optimum). During optimization, the first three best solutions obtained so far are assumed as α, β, and δ respectively. Then, other wolves are considered as ω and able to re-position with respect to α, β, and δ. The mathematical model proposed to re-adjust the position of ω wolves are as follows [14]:

$$\overrightarrow{D_\alpha} = \mid \vec{C}_1 \cdot \overrightarrow{X_\alpha} - \vec{X} \mid \tag{3.3}$$

$$\overrightarrow{D_\beta} = \mid \vec{C}_2 \cdot \overrightarrow{X_\beta} - \vec{X} \mid \tag{3.4}$$

$$\overrightarrow{D_\delta} = \mid \vec{C}_3 \cdot \overrightarrow{X_\delta} - \vec{X} \mid \tag{3.5}$$

where $\vec{X}_\alpha$ shows the position of the alpha, $\vec{X}_\beta$ shows the position of the beta, $\vec{X}_\delta$ is the position of delta, $\vec{C}_1$, $\vec{C}_2$, $\vec{C}_3$ are random vectors and $\vec{X}$ indicates the position of the current solution.

Equations (3.3), (3.4), and (3.5) calculate the approximate distance between the current solution and alpha, beta, and delta respectively. After defining the distances, the final position of the current solution is calculated as follows:

$$\overrightarrow{X_1} = \overrightarrow{X_\alpha} - \overrightarrow{A_1} \cdot \left(\overrightarrow{D_\alpha}\right) \tag{3.6}$$

$$\overrightarrow{X_2} = \overrightarrow{X_\beta} - \overrightarrow{A_2} \cdot \left(\overrightarrow{D_\beta}\right) \tag{3.7}$$

$$\overrightarrow{X_3} = \overrightarrow{X_\delta} - \overrightarrow{A_3} \cdot \left(\overrightarrow{D_\delta}\right) \tag{3.8}$$

$$\vec{X}(t+1) = \frac{\overrightarrow{X_1} + \overrightarrow{X_2} + \overrightarrow{X_3}}{3} \tag{3.9}$$

where $\vec{X}_\alpha$ shows the position of the alpha, $\vec{X}_\beta$ shows the position of the beta, $\vec{X}_\delta$ is the position of delta, $\vec{A}_1$, $\vec{A}_2$, $\vec{A}_3$ are random vectors, and t indicates the number of iterations.

As may be seen in these equations, the equations (3.3), (3.4) and (3.5) define the step size of the ω wolf toward α, β, and δ respectively. The equations (3.6), (3.7), (3.8), and (3.9) then define the final position of the ω wolves. It may also be observed that there are two vectors: $\vec{A}$ and $\vec{C}$. These two vectors are random and adaptive vectors that provide exploration and exploitation for the GWO algorithm. As shown in Fig. 2, the exploration occurs when $\vec{A}$

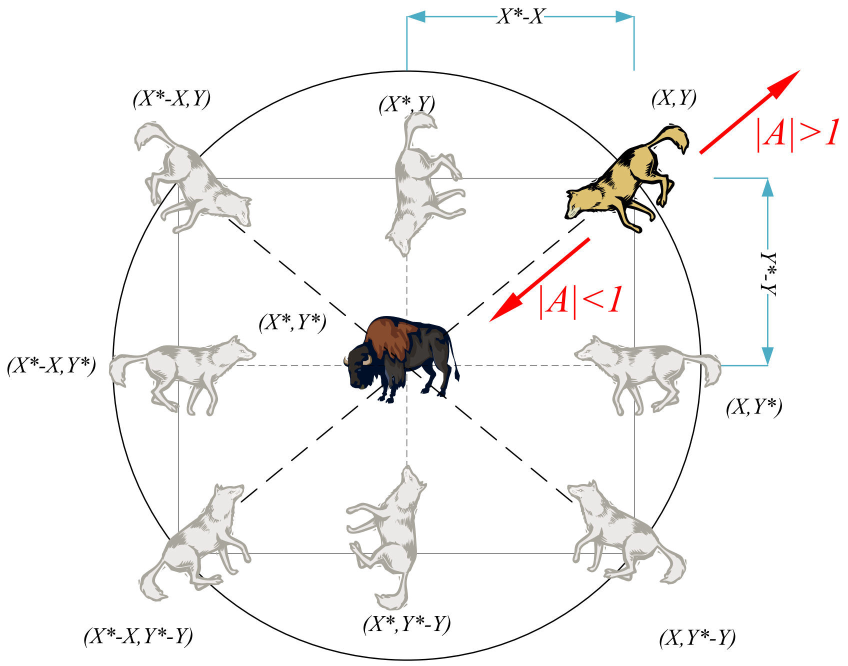

Fig. 2 Position updating mechanism of search agents and effects of A on it

is greater than 1 or less than -1. The vector C also promotes exploration when it is greater than 1. In contrary, the exploitation is emphasized when $|A|<1$ and $C<1$ (see Fig. 2). It should be noted here that A is decreased linearly during optimization in order to emphasize exploitation as the iteration counter increases. However, C is generated randomly throughout optimization to emphasize exploration/exploitation at any stage, a very helpful mechanism for resolving local optima entrapment.

After all, the general steps of the GWO algorithm are as follows:

a) Initialize a population of wolves randomly based on the upper and lower bounds of the variables
b) Calculate the corresponding objective value for each wolf
c) Choose the first three best wolves and save them as α, β, and δ
d) Update the position of the rest of the population (ω wolves) using equations (3.3) to (3.9)
e) Update parameters a, A, and C
f) Go to step b if the end criterion is not satisfied
g) Return the position of α as the best approximated optimum

Mirjalili *et al.* showed that the GWO algorithm is able to provide very competitive results compared to other well-known meta-heuristics [14]. For one, the exploration of this algorithm is very high and requires it to avoid local optima. Moreover, the balance of exploration and exploitation is very simple and effective in solving challenging problems as per the results of the real problems in [14]. The problem of training MLPs is considered as a challenging problem with an unknown search space, which is due to varying datasets and inputs that may be provided for the MLP. High exploration of the GWO algorithm, therefore, requires it to be efficient theoretically as an MLP learner. The following sections propose a GWO-based MLP trainer and investigate its effectiveness experimentally.

4 GWO-based MLP trainer

The first and most important step in training an MLP using meta-heuristics is the problem representation [36]. In other words, the problem of training MLPs should be formulated in a way that is suitable for meta-heuristics. As mentioned in the introduction, the most important variables in training an MLP are weights and biases. A trainer should find a set of values for weights and biases that provides the highest classification/approximation/prediction accuracy.

Therefore, the variables here are the weights and biases. Since the GWO algorithm accepts the variables in the form of a vector, the variables of an MLP are presented for this algorithm as follows:

$$\vec{V} = \{\overrightarrow{W}, \vec{\theta}\} = \{W_{1,1}, W_{1,2}, \cdots, W_{n,n}, h, \theta_1, \theta_2, \cdots, \theta_h\} \tag{4.1}$$

where n is the number of the input nodes, W_{ij} shows the connection weight from the ith node to the jth node, θ_j is the bias (threshold) of the jth hidden node.

After defining the variables, we need to define the objective function for the GWO algorithm. As mentioned above, the objective in training an MLP is to reach the highest classification, approximation, or predication accuracy for both training and testing samples. A common metric for the evaluation of an MLP is the Mean Square Error (MSE). In this metric, a given set of training samples is applied to the MLP and the following equation calculates the difference between the desirable output and the value that is obtained from the MLP:

$$MSE = \sum_{i=1}^{m} \left(o_i^k - d_i^k \right)^2 \tag{4.2}$$

where m is the number of outputs, d_i^k is the desired output of the ith input unit when the kth training sample is used, and o_i^k is the actual output of the ith input unit when the kth training sample appears in the input.

Obviously, an MLP should adapt itself to the whole set of training samples in order to be effective. Therefore, the performance of an MLP is evaluated based on the average of MSE over all the training samples as follows:

$$\overline{MSE} = \sum_{k=1}^{s} \frac{\sum_{i=1}^{m} \left(o_i^k - d_i^k \right)^2}{s} \tag{4.3}$$

where s is the number of training samples, m is the number of outputs, d_i^k is the desired output of the ith input unit when the kth training sample is used, and o_i^k is the actual output of the ith input unit when the kth training sample appears in the input.

After all, the problem of training an MLP can be formulated with the variable and average MSE for the GWO algorithm as follows:

$$Minimize : F(\vec{V}) = \overline{MSE} \tag{4.4}$$

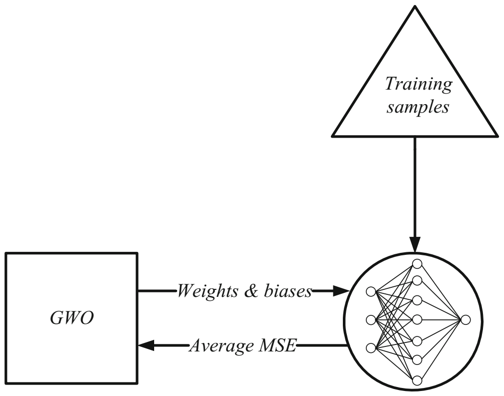

Fig. 3 GWO provides MLP with weights/biases and receives average MSE for all traning samples

Figure 3 shows the overall process of training an MLP using GWO. It may be seen that the GWO algorithm provides MLP with weights/biases and receives average MSE for all training samples. The GWO algorithm iteratively changes the weights and biases to minimize average MSE of all training samples.

The conceptual model of the way that search agents of the GWO algorithm train MLP is illustrated in Fig. 4.

As may be seen in this figure, the variables of ω wolves have higher tendency to move toward α, β and δ respectively. This is the reasons that the next position of the ω wolf is closer to α and β. In other words, the contribution of the α wolf in re-defining the weights and biases of ω wolves is the highest. Fig. 4 shows that the number of weights and biases that turn to purple when the ω wolf moves is higher than β and δ. Less contribution of β and δ on the next position of the ω can are also clearly highlighted by blue and green colours in Fig. 4. It should be noted that although similar colours are used in Fig. 4, the weights and biases are not completely replaced by α, β and δ. In fact, a value between the values of the current weight/biases of ω are calculated using equation (3.9) and replaced with the current values.

Since the weights and biases tend to move toward the bests MLPs obtained so far, there would be a high probability of having an improved MLP in each iteration. With this method, there is no absolute guarantee for finding the most optimal MLP for a dataset using GWO due to the stochastic nature of this algorithm. Since we evolve MLPs using the best MLPs obtained so far in each iteration, however, the overall average MSE of the population is decreased over the course of iterations. In other words, GWO eventually converges to a solution that is better than the random initial solutions if this algorithm is iterated with an enough number of iterations.

The next section investigates the merits of the GWO algorithm in training an MLP in practice.

5 Results and discussion

In this section the proposed GWO-based MLP trainer is benchmarked using five standard classification datasets obtained from the University of California at Irvine (UCI) Machine Learning Repository [37]: XOR, balloon, iris, breast cancer, and heart. In addition there are three function-approximation datasets obtained from [38]: sigmoid, cosine, and sine.

5.1 Experimental setup

The results are also compared with PSO, GA, ACO, ES [39–41], and Population-based Incremental Learning

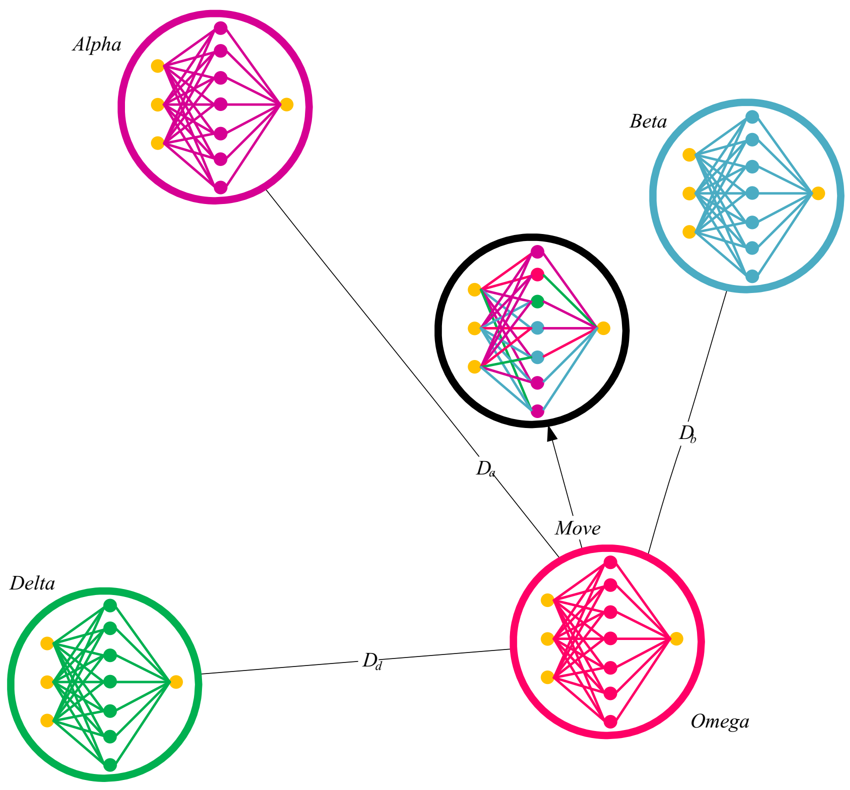

Fig. 4 Conceptual model of trainign MLP by search agents of GWO

(PBIL) [42] for verification. It is assumed that the optimization process starts with generating random weights and biases in the range of [−10,10] for all datasets. Other assumptions for the GWO and other algorithms are presented in Table 1.

Tables 2 and 3 show the specifications of the datasets. It may be observed in Table 2 that the easiest dataset is XOR with 8 training/test samples, 3 attributes, and two classes. The balloon dataset is more difficult that the XOR and has 4 attributes, 16 training samples, 16 test samples, and 2 classes. The Iris dataset has 4 attributes, 150 training/test samples, and three classes. In addition, the breast cancer dataset has 9 attributes, 599 training samples, 100 test samples, and 2 classes. Eventually, the heart dataset includes 22 attributes, 80 training samples, 187 test samples, and 2 classes. These classification datasets were deliberately chosen with different training/test samples and levels of difficulty to test the performance of the proposed GWO-based MLP trainer effectively. The difficulty of function-approximations datasets also are increased, in which the sigmoid is the easiest dataset and the sine is the hardest dataset.

A similar problem representation and objective function is utilized to train MLPs with the algorithms in Table 1. The datasets are then solved 10 times using each algorithm to generate the results. The statistical results that are presented are average (AVE) and standard deviation (STD) of the obtained best MSEs in the last iteration by the algorithms. Obviously, lower average and standard deviation of MSE in the last iteration indicates the better performance. The statistical results are presented in the form of (AVE ±STD) in Tables 4, 6, 7, 8, 9, 10, 11 and 12. Please note that the best classification rates or test errors obtained by each of the algorithms during 10 runs are reported as another metrics of comparison.

Normalization is an essential step for MLP when solving datasets with attributes in different ranges. The normalization used in this work is called min-max normalization, which is formulated as follows:

$$X' = \frac{(x-a)\times(d-c)}{(b-a)} + c \tag{5.1}$$

This formula maps x in the interval of $[a, b]$ to $[c, d]$

Another key factor in the experimental setup is the structure of MLPs. This work does not concentrate on finding the optimal number of hidden nodes and consider them equal to $2 \times N + 1$ where N is the number of features (inputs) of the datasets. The structure of each MLP that employed for each dataset is presented in Table 5.

As the size of the neural network becomes larger, obviously, the more weights and biases would be involved in the system. Consequently, the training process also becomes more challenging.

Table 1 The initial parameters of algorithms

Algorithm	Parameter	Value
GWO	$\vec{a}$	linearly decreased from 2 to 0
	Population size	50 for XOR and Balloon, 200 for the rest
	Maximum number of generations	250
PSO	Topology	Fully connected
	Cognitive constant (C_1)	1
	Social constant (C_2)	1
Inertia constant (w)	0.3	
	Population size	50 for XOR and Balloon, 200 for the rest
	Maximum number of iterations	250
GA	Type	Real coded
	Selection	Roulette wheel
	Crossover	Single point (probability=1)
	Mutation	Uniform (probability=0.01)
	Population size	50 for XOR and Balloon, 200 for the rest
	Maximum number of generations	250
ACO	Initial pheromone (τ)	1e-06
	Pheromone update constant (Q)	20
	Pheromone constant (q)	1
	Global pheromone decay rate (p_g)	0.9
	Local pheromone decay rate (p_t)	0.5
	Pheromone sensitivity (α)	1
	Visibility sensitivity (β)	5
	Population size	50 for XOR and Balloon, 200 for the rest
	Maximum number of iterations	250
ES	λ	10
	σ	1
	Population size	50 for XOR and Balloon, 200 for the rest
	Maximum number of generations	250
PBIL	Learning rate	0.05
	Good population member	1
	Bad population member	0
	Elitism parameter	1
	Mutational probability	0.1
	Population size	50 for XOR and Balloon, 200 for the rest
	Maximum number of generations	250

Table 2 Classification datasets

Classification datasets	Number of attributes	Number of training samples	Number of test samples	Number of classes
3-bits XOR	3	8	8 as training samples	2
Balloon	4	16	16 as training samples	2
Iris	4	150	150 as training samples	3
Breast cancer	9	599	100	2
Heart	22	80	187	2

Table 3 Function-approximation datasets

Function-approximation datasets	Training samples	Test samples
Sigmoid: $y = 1/(1+e^{-x})$	61: x in [−3:0.1:3]	121: x in [−3:0.05:3]
Cosine: $y = (\cos(x\pi/2))^7$	31: x in [1.25:0.05:2.75]	38: x in [1.25:0.04:2.75]
Sine: $y = \sin(2x)$	126: $x$in [-2π:0.1:2π]	252: x in [-2π:0.05:2π]

5.2 XOR dataset

As may be seen in Table 2, this dataset has 3 inputs, 8 training/test samples, and 1 output. The MLP for this dataset has to return the XOR of the input as the output. The statistical results of GWO, PSO, GA, ACO, ES, and PBIL on this dataset are presented in Table 4. Note that the training algorithms are named in the form of Alglorithm-MLP in the following tables.

The results show that the best average MSE belongs to GA-MLP and GWO-MLP. This shows that these two algorithms have the highest ability to avoid local optima, significantly better than other algorithms. GWO-MLP and GA-MLP also classifies this dataset with 100 percent accuracy. The GA-MLP algorithm is an evolutionary algorithm which has a very high level of exploration. These results show that the GWO-based trainer is able to show very competitive results compare to the GA-MLP algorithm

Another fact worth noticing here is the poor performance of ACO-MLP on this dataset. This might be due to the fact that this algorithm is more suitable for combinatorial problems.

5.3 Balloon dataset

This dataset has 4 attributes, 18 training/test samples, and 2 classes. The trainers for this dataset have dimensions of 55. The results are provided in Table 6.

The first thing that can be observed in the results is the similar classification rate for all algorithms. This behaviour is due to simplicity of this dataset. However, the average and standard deviation of the MSE over 10 runs are different for algorithms. Similarly to the previous dataset, GWO-MLP and GA-MLP show the best local optima avoidance as per the statistical results of MSEs. This again justifies high performance of the GWO algorithm in training MLPs.

Table 4 Experimental results for the XOR dataset

Algorithm	MSE (AVE ±STD)	Classification rate
GWO-MLP	0.009410 ±.029500	100.0 %
PSO-MLP	0.084050 ± 0.035945	37.50 %
GA-MLP	0.000181 ±0.000413	100.00 %
ACO-MLP	0.180328 ±0.025268	62.50 %
ES-MLP	0.118739 ±0.011574	62.50 %
PBIL-MLP	0.030228 ±0.039668	62.50 %

5.4 Iris dataset

The Iris dataset is indeed one of the most popular datasets in the literature. It consists of 4 attributes, 150 training/test samples, and 3 classes. Therefore, the MLP structure for solving this dataset is of 4-9-3 and the problem has 75 variables. The results of training algorithms are presented in Table 7.

This Table shows that the GWO-MLP outperforms other algorithms in terms of not only the minimum MSE but also the maximum classification accuracy. The results of GWO-MLP follow by those of GA-MLP and PBIL-MLP. Since the difficulty of this dataset and MLP structure is high for this dataset, these results are strong evidences for the efficiencies of GWO in training MLPs. The results testify that this algorithm has superior local optima avoidance and accuracy simultaneously.

5.5 Breast cancer dataset

This dataset is another challenging well-known dataset in the literature. There are 9 attributes, 599 training samples, 100 test samples, and 2 classes in this dataset. As may be seen in Table 3, an MLP of 4-9-1 is chosen to be trained and solve this dataset. Therefore, every search agent of the trainers in this work have 209 variables to be optimized. This dataset is solved by the trainers 10 times, and the statistical results are provideed in Table 8.

Table 5 MLP structure for each dataset

Classification datasets	Number of attributes	MLP structure
3-bits XOR	3	3-7-1
Balloon	4	4-9-1
Iris	4	4-9-3
Breast cancer	9	9-19-1
Heart	22	22-45-1
Sigmoid	1	1-15-1
Cosine	1	1-15-1
Sine	1	1-15-1

Table 6 Experimental results for the balloon dataset

Algorithm	MSE (AVE ±STD)	Classification rate
GWO-MLP	9.38e-15 ±2.81e-14	100.00 %
PSO-MLP	0.000585 ±0.000749	100.00%
GA-MLP	5.08e-24 ±1.06e-23	100.00%
ACO-MLP	0.004854 ±0.007760	100.00%
ES-MLP	0.019055 ±0.170260	100.00%
PBIL-MLP	2.49e-05 ±5.27e-05	100.00%

Table 8 Experimental results for the breast cancer dataset

Algorithm	MSE (AVE ±STD)	Classification rate
GWO-MLP	0.0012 ±7.4498e-05	99.00 %
PSO-MLP	0.034881 ±0.002472	11.00 %
GA-MLP	0.003026 ±0.001500	98.00 %
ACO-MLP	0.013510 ±0.002137	40.00 %
ES-MLP	0.040320 ±0.002470	06.00 %
PBIL-MLP	0.032009 ±0.003065	07.00 %

The results of this table are consistent with those of Table 7, in which the GWO-MLP algorithm again shows the best results. The average and standard deviation provided by GWO-MLP prove that this algorithm has a high ability to avoid local minima and approximates the best optimal values for weights and biases. The second best results belong to GA-MLP and PBIL-MLP algorithms. In addition to MSEs, the best classification accuracy of the proposed GWO-MLP algorithm is higher than others. The breast cancer dataset has the highest difficulty compared to the previously discussed datasets in terms of the number of weights, biases, and training samples. Therefore, these results strongly evidence the suitability of the proposed GWO-MLP algorithm in training MLPs. For one, this algorithm shows high local optima avoidance. For another, the local search around to global optimum and exploitation are high.

It should be noted here that the classification rates of some of the algorithms are very small for this dataset because the end criterion for all algorithms is the maximum number of iteration. Moreover, the number of search agents is fixed throughout the experiments. Of course, increasing the number of iterations and population size would improve the absolute classification rate but it was the comparative performance between algorithms that was of interest. It was the ability of algorithms in terms of avoiding local minima in the classification that was the main objective of this work, so the focus was not on finding the best maximum number of iterations and population size.

5.6 Heart dataset

The last classification dataset solved by the algorithms is the heart dataset, which has 22 features, 80 training samples, 187 test samples, and 2 classes. MLPs with the structure of 22-45-1 are trained by the algorithms. The results are reported in Table 9

The results of this table reveal that GWO-MLP and GA-MLP have the best performances in this dataset in terms of improved MSE. The average and standard deviation of MSEs show that the performances of these two algorithms are very close. However, the classification accuracy of the GWO-MLP algorithm is much higher. Once more, these results evidence the merits of the proposed GWO-based algorithm in training MLPs.

5.7 Sigmoid dataset

The sigmoid function is the simplest function employed in this work for approximation. It may be seen in Table 3 that there are 61 and 121 training and testing samples. The results of the algorithms are reported in Table 10.

This table shows that the best results are provided by the GWO-MLP algorithm. This shows that the proposed trainer can be also very effective in function-approximation datasets. The low AVE ±STD shows the superior local optima avoidance of this algorithm. The results also show that the best test error belongs the the GWO-MLP algorithm. This justifies the accuracy of this algorithm as well.

Table 7 Experimental results for the iris dataset

Algorithm	MSE (AVE ±STD)	Classification rate
GWO-MLP	0.0229 ±0.0032	91.333
PSO-MLP	0.228680 ±0.057235	37.33 %
GA-MLP	0.089912 ±0.123638	89.33 %
ACO-MLP	0.405979 ±0.053775	32.66 %
ES-MLP	0.314340 ±0.052142	46.66 %
PBIL-MLP	0.116067 ±0.036355	86.66 %

Table 9 Experimental results for the heart dataset

Algorithm	MSE (AVE ±STD)	Classification rate
GWO-MLP	0.122600 ±0.007700	75.00 %
PSO-MLP	0.188568 ±0.008939	68.75%
GA-MLP	0.093047 ±0.022460	58.75 %
ACO-MLP	0.228430 ±0.004979	00.00 %
ES-MLP	0.192473 ±0.015174	71.25 %
PBIL-MLP	0.154096 ±0.018204	45.00 %

Table 10 Experimental results for the sigmoid dataset

Algorithm	MSE (AVE ±STD)	Test error
GWO-MLP	0.000203 ±0.000226	0.27134
PSO-MLP	0.022989 ±0.009429	3.35630
GA-MLP	0.001093 ±0.000916	0.44969
ACO-MLP	0.023532 ±0.010042	3.99740
ES-MLP	0.075575 ±0.016410	8.80150
PBIL-MLP	0.004046 ±2.74e-17	2.9446

Table 12 Experimental results for the sine dataset

Algorithm	MSE (AVE ±STD)	Test error
GWO-MLP	0.261970 ±0.115080	43.754
PSO-MLP	0.526530 ±0.072876	124.89
GA-MLP	0.421070 ±0.061206	111.25
ACO-MLP	0.529830 ±0.053305	117.71
ES-MLP	0.706980 ±0.077409	142.31
PBIL-MLP	0.483340 ±0.007935	149.60

5.8 Cosine dataset

The cosine dataset is more difficult than the sigmoid dataset, but the results of GWO-MLP follow a similar pattern in both datasets. Table 11 shows that once more the GWO-based trainer shows the best statistical results on not only the MSE but also the test error. The results of the GWO-MLP are closely followed by the GA-MLP algorithm.

5.9 Sine dataset

The most difficult function-approximation dataset is the sine dataset. This function has four peaks that make it very challenging to be approximated. The results of algorithms on this dataset are consistent with those of other two function-approximation datasets. It is worth mentioning that the GWO-MLP algorithm provides very accurate results on this dataset as can be inferred from test errors in Table 12.

5.10 Discussion and analysis of the results

Statistically speaking, the GWO-MLP algorithm provides superior local optima avoidance in six of the datasets (75%) and the best classification accuracy in all of the datasets (100%). The reason for improved MSE is the high local optima avoidance of this algorithm. According to the mathematical formulation of the GWO algorithm, half of the iterations are devoted to exploration of the search space (when $|A|>1$). This promotes exploration of the search space that leads to finding diverse MLP structures during optimization. In addition, the C parameter always randomly obliges the search agents to take random steps towards/outwards the prey. This mechanism is very helpful for resolving local optima stagnation even when the GWO algorithm is in the exploitation phase. The results of this work show that although evolutionary algorithms have high exploration, the problem of training an MLP needs high local optima avoidance during the whole optimization process. This is because the search space is changed for every dataset in training MLPs. The results prove that the GWO is very effective in this regard.

Another finding in the results is the poor performance of PSO-MLP and ACO-MLP. These two algorithms belong to the class of swarm-based algorithms. In contrary to evolutionary algorithms, there is no mechanism for significant abrupt movements in the search space and this is likely to be the reason for the poor performance of PSO-MLP and ACO-MLP. Although GWO is also a swarm-based algorithm, its mechanisms described in the preceding paragraph are the reasons why it is advantageous in training MLPs.

It is also worth discussing the poor performance of the ES algorithm in this subsection. Generally speaking, the ES algorithm has been designed based on various mutation mechanisms. Mutation in evolutionary algorithms maintains the diversity of population and promotes exploitation, which is one of the main reasons for the poor performance of ES. In addition, selection of individuals in this algorithm is done by a deterministic approach. Consequently, the randomness in selecting an individual is less and therefore local optima avoidance is less as well. This is another reason why EA failed to provide good results in the datasets.

The reason for the high classification rate provided by the GWO-MLP algorithm is that this algorithm is equipped with adaptive parameters to smoothly balance exploration and exploitation. Half of the iteration is devoted to exploration and the rest to exploitation. In addition, the GWO algorithm always saves the three best obtained solutions at any stage of optimization. Consequently, there are always guiding search agents for exploitation of the most promising regions of the search space. In other words, GWO-MLP

Table 11 Experimental results for the cosine dataset

Algorithm	MSE (AVE ±STD)	Test error
GWO-MLP	0.003105 ±0.0021605	0.66919
PSO-MLP	0.058986 ±0.021041	2.0090
GA-MLP	0.010920 ±0.006316	0.7105
ACO-MLP	0.050872 ±0.010809	2.4498
ES-MLP	0.086640 ±0.022208	3.1461
PBIL-MLP	0.094342 ±0.018468	3.7270

benefits from intrinsic exploitation guides, which also assist this algorithm to provide remarkable results.

According to this comprehensive study, the GWO algorithm is highly recommended to be used in hybrid intelligent optimization schemes such as training MLPs. Firstly, this recommendation is made because of its high exploratory behaviour, which results in high local optima avoidance when training MLPs. The high exploitative behaviour is another reason why a GWO-based trainer is able to converge rapidly towards the global optimum for different datasets. However, it should be noted here that GWO is highly recommended only when the dataset and the number of features are very large. Obviously, small datasets with very few features can be solved by gradient-based training algorithms much faster and without extra computational cost. In contrast, the GWO algorithm is useful for large datasets due to the extreme number of local optima that makes the conventional training algorithm almost ineffective.

6 Conclusion

In this paper, the recently proposed GWO algorithm was employed for the first time as an MLP trainer. The high level of exploration and exploitation of this algorithm were the motivations for this study. The problem of training an MLP was first formulated for the GWO algorithm. This algorithm was then employed to define the optimal values for weights and biases. The proposed GWO-based trainer was applied to five standard classification datasets (XOR, balloon, Iris, breast cancer, and heart) as well as three function-approximation datasets (sigmoid, cosine, and sine). For verification, the results of the GWO-MLP algorithm were compared to five other stochastic optimization trainers: PSO, GA, ACO, ES, and PBIL. The results showed that the proposed method is able to be very effective in training MLPs. For one, the GWO-MLP has very a high level of local optima avoidance, which enhances the probability of finding proper approximations of the optimal weights and biases for MLPs. Moreover, the accuracy of the obtained optimal values for weights and biases is very high, which is due to the high exploitation of the GWO-MLP trainer. This paper also identified and discussed the reasons for strong and poor performances of other algorithms. It was observed that the swarm-based algorithms suffer from low exploration, whereas the GWO does not.

For future work, it is recommended that the GWO algorithm be applied to find the optimal number of hidden nodes, layers, and other structural parameters of MLPs. Fine tuning of this algorithm is worth further research.

References

1. McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophysics 5:115–133
2. Bebis G, Georgiopoulos M (1994) Feed-forward neural networks. Potentials, IEEE 13:27–31
3. Kohonen T (1990) The self-organizing map. Proc IEEE 78:1464–1480
4. Park J, Sandberg IW (1993) Approximation and radial-basis-function networks. Neural Comput 5:305–316
5. Dorffner G (1996) Neural networks for time series processing, in Neural Network World
6. Ghosh-Dastidar S, Adeli H (2009) Spiking neural networks. Int J Neural Syst 19:295–308
7. Reed RD, Marks RJ (1998) Neural smithing: supervised learning in feedforward artificial neural networks. Mit Press
8. Caruana R, Niculescu-Mizil A (2006) An empirical comparison of supervised learning algorithms. In: Proceedings of the 23rd international conference on Machine learning, pp 161–168
9. Hinton GE, Sejnowski TJ (1999) Unsupervised learning: foundations of neural computation. MIT press
10. Wang D (2001) Unsupervised learning: foundations of neural computation. AI Mag 22:101
11. Hertz J (1991) Introduction to the theory of neural computation. Basic Books 1
12. Wang G-G, Guo L, Gandomi AH, Hao G-S, Wang H (2014) Chaotic krill herd algorithm. Inf Sci 274:17–34
13. Wang G-G, Gandomi AH, Alavi AH, Hao G-S (2013) Hybrid krill herd algorithm with differential evolution for global numerical optimization. Neural Comput App. doi:10.1007/s00521-013-1485-9
14. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61
15. Van Laarhoven PJ, Aarts EH (1987) Simulated annealing. Springer
16. Szu H, Hartley R (1987) Fast simulated annealing. Phys Lett A 122:157–162
17. Mitchell M, Holland JH, Forrest S (1993) When will a genetic algorithm outperform hill climbing? In: NIPS, pp 51–58
18. Goldfeld SM, Quandt RE, Trotter HF (1966) Maximization by quadratic hill-climbing. Econometrica: J Econ Soc:541–551
19. Mirjalili S, Mohd Hashim SZ, Moradian Sardroudi H (2012) Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm. Appl Math Comput 218:11125–11137
20. Whitley D, Starkweather T, Bogart C (1990) Genetic algorithms and neural networks: Optimizing connections and connectivity. Parallel comput 14:347–361
21. Mendes R, Cortez P, Rocha M, Neves J (2002) Particle swarms for feedforward neural network training, learning vol. 6
22. Gudise VG, Venayagamoorthy GK (2003) Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In: Proceedings swarm intelligence symposium, 2003. SIS'03, pp 110–117
23. Blum C, Socha K (2005) Training feed-forward neural networks with ant colony optimization: an application to pattern classification. In: 5th international conference on, Hybrid Intelligent Systems, 2005. HIS'05, p 6
24. Socha K, Blum C (2007) An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training. Neural Comput Appl 16:235–247
25. Karaboga D, Akay B, Ozturk C (2007) Artificial bee colony (ABC) optimization algorithm for training feed-forward neural

networks,” in Modeling decisions for artificial intelligence ed: Springer, pp 318–329
26. Ozturk C, Karaboga D (2011) Hybrid Artificial Bee Colony algorithm for neural network training. In: 2011 IEEE Congress on, Evolutionary Computation (CEC), pp 84–88
27. Ilonen J, Kamarainen J-K, Lampinen J (2003) Differential evolution training algorithm for feed-forward neural networks. Neural Process Lett 17:93–105
28. Slowik A, Bialko M (2008) Training of artificial neural networks using differential evolution algorithm. In: 2008 Conference on, Human System Interactions, pp 60–65
29. Green II RC, Wang L, Alam M (2012) Training neural networks using central force optimization and particle swarm optimization: insights and comparisons. Expert Syst Appl 39:555–563
30. Pereira L, Rodrigues D, Ribeiro P, Papa J, Weber SA (2014) Social-spider optimization-based artificial neural networks training and its applications for Parkinson’s disease identification. In: 2014 IEEE 27th international symposium on in computer-based medical systems (CBMS), pp 14–17
31. Yu JJ, Lam AY, Li VO (2011) Evolutionary artificial neural network based on chemical reaction optimization. In: 2011 IEEE congress on, evolutionary computation (CEC), pp 2083–2090
32. Pereira LA, Afonso LC, Papa JP, Vale ZA, Ramos CC, Gastaldello DS, Souza AN (2013) Multilayer perceptron neural networks training through charged system search and its Application for non-technical losses detection. In: 2013 IEEE PES conference on, innovative smart grid technologies latin America (ISGT LA), pp 1–6
33. Moallem P, Razmjooy N (2012) A multi layer perceptron neural network trained by invasive weed optimization for potato color image segmentation. Trends Appl Sci Res 7:445–455
34. Uzlu E, Kankal M, Akpınar A, Dede T (2014) Estimates of energy consumption in Turkey using neural networks with the teaching–learning-based optimization algorithm. Energy 75:295–303
35. Mirjalili S, Sadiq AS (2011) Magnetic optimization algorithm for training multi layer perceptron. In: Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference, IEEE, pp 42–46
36. Belew RK, McInerney J, Schraudolph NN (1990) Evolving networks: Using the genetic algorithm with connectionist learning
37. Blake C, Merz CJ (1998) {UCI} Repository of machine learning databases
38. Mirjalili S, Mirjalili SM, Lewis A (2014) Let a biogeography-based optimizer train your multi-layer perceptron. Inf Sci 269:188–209
39. Beyer H-G, Schwefel H-P (2002) Evolution strategies–a comprehensive introduction. Nat Comput 1:3–52
40. Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. Evolutionary Comput IEEE Trans 3:82–102
41. Yao X, Liu Y (1997) Fast evolution strategies. In: evolutionary programming VI, pp 149–161
42. Baluja S (1994) Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning, DTIC Document

Seyedali Mirjalili received his B.Sc. degree in Computer Engineering (software) from Yazd University, Iran. He obtained his M. Sc. degree in Computer Science from Universiti Teknologi Malaysia (UTM) in 2011. He was a member of Soft Computing Research Group (SCRG) at UTM. Ali is currently a Ph. D. scholar at the School of Information and Communication Technology (ICT), Griffith University, Australia. His research interests include Robust Optimisation, Multi-objective Optimisation, Swarm Intelligence, Evolutionary Algorithms, and Artificial Neural Networks. He is working on the application of multi-objective and robust meta-heuristic optimisation techniques in Computational Fluid Dynamic (CFD) problems.