Mathematical Biosciences and Engineering

MBE, 21(2): 2970–2990.
DOI: 10.3934/mbe.2024132
Received: 04 October 2023
Revised: 05 January 2024
Accepted: 22 January 2024
Published: 30 January 2024

http://www.aimspress.com/journal/MBE

Research article

Evolutionary optimization framework to train multilayer perceptrons for engineering applications

Rami AL-HAJJ[1,*], Mohamad M. Fouad[2] and Mustafa Zeki[1]

[1] College of Engineering and Technology, American University of the Middle East, Kuwait
[2] Department of Computer Engineering and Systems, Faculty of Engineering, Mansoura University, Mansoura 35516, Egypt

* **Correspondence:** Email: rami.alhajj@aum.edu.kw.

Abstract: Training neural networks by using conventional supervised backpropagation algorithms is a challenging task. This is due to significant limitations, such as the risk for local minimum stagnation in the loss landscape of neural networks. That may prevent the network from finding the global minimum of its loss function and therefore slow its convergence speed. Another challenge is the vanishing and exploding gradients that may happen when the gradients of the loss function of the model become either infinitesimally small or unmanageably large during the training. That also hinders the convergence of the neural models. On the other hand, the traditional gradient-based algorithms necessitate the pre-selection of learning parameters such as the learning rates, activation function, batch size, stopping criteria, and others. Recent research has shown the potential of evolutionary optimization algorithms to address most of those challenges in optimizing the overall performance of neural networks. In this research, we introduce and validate an evolutionary optimization framework to train multilayer perceptrons, which are simple feedforward neural networks. The suggested framework uses the recently proposed evolutionary cooperative optimization algorithm, namely, the dynamic group-based cooperative optimizer. The ability of this optimizer to solve a wide range of real optimization problems motivated our research group to benchmark its performance in training multilayer perceptron models. We validated the proposed optimization framework on a set of five datasets for engineering applications, and we compared its performance against the conventional backpropagation algorithm and other commonly used evolutionary optimization algorithms. The simulations showed the competitive performance of the proposed framework for most examined datasets in terms of overall performance and convergence. For three benchmarking datasets, the proposed framework provided increases of 2.7%, 4.83%, and 5.13% over the performance of the second best-performing optimizers, respectively.

Keywords: cooperative optimization algorithm; evolutionary computation; neural networks; multilayer perceptrons; machine learning; heuristic optimization; evolutionary training; optimizing neural networks; optimizing multilayer perceptrons

1. Introduction

Neural networks (NNs) are non-parametric machine learning models inspired by the structure and operation of the biological brain cells. They consist of interconnected computational units, called perceptrons, organized into layers [1]. In recent decades, NNs have been used in various classification, regression, and forecasting tasks [2–5]. Their architecture may vary depending on the specific tasks they are designed to solve, and their efficiency is highly affected by their learning process. Multi-layer perceptrons (MLP) are feedforward NNs consisting of sequential layers of neurons, also called perceptrons, that are interconnected through weighted synapses [1]. A simple MLP consists of three layers of perceptrons that are arranged as follows. The input layer receives the input signals, whereas the middle and output layers perform complex calculations and make the MLP capable of approximating any continuous function. MLPs are often trained on a set of labeled training instances that are prepared as input-target pairs to learn the association between inputs and outputs of the MLP model. The learning process of an MLP, which belongs to the group of NP-hard problems, involves adjusting the hyperparameters and weights of that MLP model by implementing two main categories of optimization techniques: gradient descent backpropagation algorithms and heuristic methods. The gradient-based algorithms such as gradient descent, conjugate gradient, and Levenberg Marquardt algorithms are conventional back-propagation supervised learning algorithms [6–9]. There are a few significant challenges associated with most of the gradient descent-based learning algorithms, namely, (1) the tendency to get stuck in local minima or saddle points in the loss function's landscape of the MLP; (2) the slow convergence of the learning process, especially in vanishing and exploding gradient phenomena [10]; and (3) the strong dependency on initial weights and learning rates [11]. All of those limitations, and others, make it difficult for the MLP to learn effectively.

Recently, heuristic optimization algorithms have shown several advantages over conventional gradient-descent methods in the training of MLPs. Many evolutionary and swarm-based optimization algorithms are extensively used to solve complex optimization problems and have been suggested in the literature as competitive alternatives to optimize the weights and hyperparameters of MLPs [12–14]. In contrast to gradient-based learning, heuristic optimization methods can prevent the occurrence of local minima since they are designed to search for global optima across complex search spaces of weights. This is particularly convenient in complex MLPs with many parameters, where finding a global optimum is essential to improve the overall performance [15].

Besides, heuristic algorithms for optimization can efficiently optimize the MLPs with non-differentiable or discontinuous loss functions since they do not count on gradient information. Therefore, those optimization algorithms are suitable for problems with non-smooth objective functions [16,17].

Although there are many suggested evolutionary approaches to optimize MLP models, there is still an open problem of stagnation in local optima because of the likely weak exploration and/or exploitation of some of those heuristic approaches. Furthermore, evolutionary optimization approaches are generally slow since they require computing an objective function value for each candidate solution in the evolving populations. Motivated by these reasons, in this research, a new framework based on the recently introduced dynamic group-based cooperative optimizer (DGCO) optimization algorithm is suggested and validated for the training of MLPs with a single hidden layer. The DGCO, a meta-heuristic evolutionary computing algorithm for optimization, is inspired by the dynamic collaboration of teamwork agents. The DGCO dynamically manages the exploration and exploitation tasks over all iterations until the final one is reached to solve complex optimization problems [18]. We examined the proposed approach for the optimization of MLP models by using five classification benchmarking datasets for engineering applications. The simulation results are compared to those obtained by commonly used optimizers in the literature, including cooperative and competitive evolutionary algorithms for optimization, namely the genetic algorithm (GA), grey wolf optimization GWO algorithm [19], and particle swarm optimization (PSO) algorithm, as well as those obtained by the conventional gradient descent-based algorithm.

In general, the evolutionary computation (EC) algorithms deployed for the automatic learning of MLP models mainly aim to optimize either the structure of those models or the weights of the connections among their perceptrons. In this research, we validate our approach to optimizing the weights and biases of MLP models for classification. In this work, we introduce the DGCO as a competitive optimizer to optimize the parameters of MLP models for applications with limited training datasets, as shown in the simulation results in Section 6.2.

The main contributions of this paper are as follows:

1) Introducing a novel framework to optimize feedforward MLP models by using a cooperative EC algorithm. The framework validated the ability of the DGCO algorithm to optimize a machine learning model. The suggested optimizer showed its competitive performance in in its ability to avoid local optima as well as its fast convergence and optimization time.

2) We validate the proposed framework's ability to optimize the weights of MLP models even in cases of reduced or limited training datasets.

3) We compare the performance of the proposed framework with other competitive evolutionary optimizers in terms of accuracy and run time.

The remainder of this paper is organized as follows. Section 2 shows a background and introduces related work, and Section 3 briefly describes MLP models and their basic concepts. Section 4 overviews the DGCO evolutionary algorithm and its equations and main features. In Section 5, we detail the proposed DGCO-based framework. The simulation results are depicted and discussed in Section 6. Finally, we conclude our work and summarize its limitations and how they can affect our future research perspectives in Section 7.

2. Background and related work

Swarm-based algorithms are the most investigated heuristic methods among the EC algorithms for the optimization of MLP models. Those population-based algorithms are nature-inspired, where a population consists of agents representing solution candidates for the optimization problem. The initially randomized solutions in the early populations are evolved and updated until a satisfactory solution is found or a stopping criterion is reached. The incorporated randomness in the EC algorithms permits the exploration of the search space and the move from a local search to a global one, which makes those algorithms suitable for global optimization [15].

In general, the EC optimization algorithms are considered as either competitive, like the GA, or cooperative, like the PSO algorithm [20,21]. GAs, introduced in [22–24], constitute one type of the most investigated evolutionary heuristic algorithms for the optimization of MLP models [25–28]. Several swarm-based optimization algorithms are reported in the literature as NN optimizers, namely, the ant colony optimization algorithm [29,30] and the artificial bee colony (ABC) algorithm [31]. Those algorithms, with their variations, were investigated for their ability to optimize MLP models [32–36]. Besides, other EC algorithms, like the differential evolution (DE) [37] and the brain storm optimization algorithms [38] that simulate brainstorming process, are also employed to optimize the weights and biases of MLPs, effectively tweaking an MLP model to improve its performance [39,40]. Recently, optimizing MLP models by using heuristic evolutionary optimizers for medical, agriculture, food quality, and mineral grading applications have been reported in the literature [41–43]. In [43] the authors compared the performance of several EC heuristic optimizers like ABC, PSO, and the GA to estimate gold grade with highly sparse data. On the other hand, several papers have suggested combined approaches that hybridize gradient-based learning with EC for global optimization [44,45], while in others, hybridized heuristics are proposed to optimize MLP models [46]. In [44], the authors introduced and validated a framework that combines the DE optimizer with the local conjugate gradient optimization algorithm. The authors claimed that the DE-GC approach outperformed other optimizers, including two variants of DE. In a forecasting application, the authors of [45] proposed a hybrid PSO and conjugate gradient learning of MLP models to predict air pollution parameter data, namely the suspended particulate matter. The authors claimed that the hybrid approach provided better predictions than the gradient conjugate and PSO separately. In [46], the authors proposed a framework that hybridizes the GA algorithm and ABC algorithm to optimize MLP models for medical applications to detect diabetes and breast cancer cases.

3. Multilayer perceptrons models

MLPs are artificial NNs of the feedforward type where sequential layers of processing units called perceptrons are interconnected through weighted connections [1]. An MLP unit is a cell of computation called a perceptron. Each perceptron has an activation function that decides the final output of that perceptron. A simple MLP may consist of three layers of perceptrons that are arranged as follows: the input layer receives the input signals to map them to the next MLP layer, whereas the middle and output layers make the decisions about the input signals. An MLP may include an arbitrary number of hidden layers that perform complex calculations and make the MLP capable of approximating any continuous function for regression or classification. MLP layers are, in general, interconnected in a one-directional fashion. The connections are represented by real numbers that hold the knowledge of the MLP.

The MLP models are trained on labeled input-target pairs to learn how to model the associations between input vectors and output targets. The learning process involves adjusting their parameters and weights through a supervised learning process. The supervised learning of MLP models is completed by using the classical gradient descent backpropagation algorithm or one of its variants or, more recently, by an EC-based learner. Figure 1 depicts the general structure of an MLP model with one hidden layer. The outcome of each perceptron is computed in two steps. First, the weighted sum of the inputs is calculated by using Eq (1).

$$U_j = \sum_{i=1}^{n} W_{ij}\ x_i + \ b_j \tag{1}$$

where $\boldsymbol{x_i}$ is the input to the jth perceptron and $\boldsymbol{w_{ij}}$ is the weight of the connection between the input $\boldsymbol{i}$ and the perceptron $\boldsymbol{j}$ in the next layer. The bias $\boldsymbol{b_j}$ of the perceptron $\boldsymbol{j}$ decides the output of that perceptron against the value of the weighted input $\boldsymbol{W_{ij}}\ \boldsymbol{x_i}$. In the second step, the final output of the neuron $\boldsymbol{j}$, denoted by $\varphi(U_j)$ is computed by applying an activation function $\varphi(.)$ on the summation value $\boldsymbol{U_j}$ computed in Eq (1). Several types of continuous activation functions are commonly used in MLP models [1]. Equation (2) shows the computation of the output of neuron $\boldsymbol{j}$ by using the sigmoid function that is widely used in the literature;

$$\varphi(U_j) = \frac{1}{1+\ e^{-aU_j}} \tag{2}$$

where $\boldsymbol{U_j}$ is the weighted summation of inputs to perceptron $\boldsymbol{j}$, and $\boldsymbol{a}$ is the slant parameter of the sigmoid function.

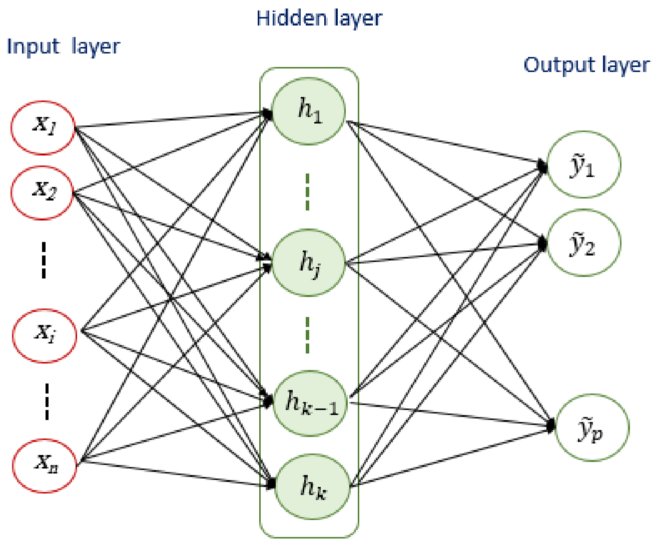

Figure 1. An MLP NN with a single hidden layer.

4. DGCO

The DGCO [18] is influenced by the cooperative behavior adopted by swarm individuals to achieve their global goals. The optimization process handled by the DGCO starts with an initial random population of individuals representing candidate solutions to the problem being solved. Then, the fitness of each individual of the population is calculated. After that, the DGCO divides the population into two groups: the exploration group and the exploitation group (Figure 2). The responsibility of the exploration group is to search for promising areas in the search space. To do that, individuals in the exploration group apply one of two techniques. The first one searches the area around a solution. This strategy is mathematically modeled by using the following equations:

$$\vec{D} = r_1.(\vec{S}(t) - 1) \tag{3}$$

$$\vec{S}(t+1) = \vec{S}(t) + \vec{D}.(\ 2\vec{r_2} - 1) \tag{4}$$

where $\vec{r_1}$ and $\vec{r_2}$ are coefficient vectors in the intervals [0, 2] and [0, 1] respectively, t refers to the current iteration, $\vec{S}$ is the current solution vector, and $\vec{D}$ is the distance between the current agent and the solution agent.

The second strategy that is adopted by individuals of the exploration group is Mutation, which improves the diversity in the population and therefore helps to prevent stagnation into local optima. In general, these two strategies help to increase the DGCO's exploration capabilities.

Besides, individuals in the exploitation group are responsible for finding better positions in the search space. To achieve that, individuals in the exploitation group apply two different techniques. The first one allows the search agents to move toward the best solution found so far by using random walks. This task is mathematically modeled by using the following equations:

$$\vec{D} = \ \vec{r_3}\ .\ \ (\vec{L}(t) - \ \vec{S}(t)) \tag{5}$$

$$\vec{S}(t+1) = \vec{S}(t) + \vec{D} \tag{6}$$

where $\boldsymbol{\vec{r_3}}$ is a random vector of values in the interval [0, 2] that controls the moving steps toward the best solution, also referred to as the solution, $\boldsymbol{t}$ refers to the current iteration, $\vec{S}$ is the vector of the current solution, $\vec{L}$ is the vector of the best solution, and $\vec{D}$ indicates the distance vector.

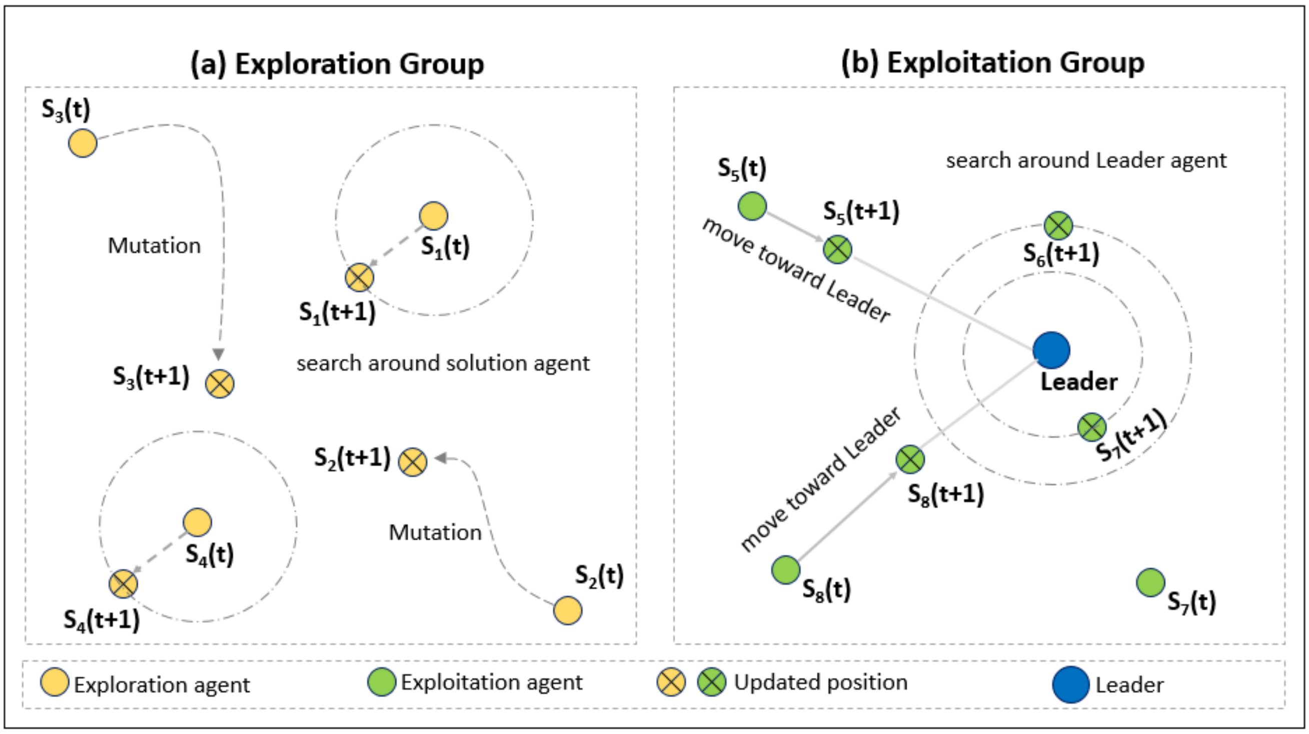

Figure 2. Exploration and exploitation groups of the DGCO.

The second technique used by individuals in the exploitation task permits to randomly search around the best solution. This strategy is modeled by using the following equations:

$$\vec{D} = \vec{L}(t) * (\vec{k} - \ r_4) \tag{7}$$

$$\vec{S}(t+1) = \vec{S}(t) + \vec{D}.(\ 2\vec{r_5} - 1) \tag{8}$$

$$\vec{k} = 2 - \frac{2 \times t^2}{iters\_count^2} \tag{9}$$

where $\vec{r_4}$ and $\vec{r_5}$ are random vectors of numbers in the interval [–1, 1], $\vec{k}$ decreases exponentially from 2 to 0 throughout iterations, $\vec{L}$ is the vector of the best solution, $\vec{S}$ is the current solution vector, and $\vec{D}$ indicates the diameter of the circle in which the solution looks for better solutions.

One of the advantages of the DGCO algorithm is its ability to achieve exploration and exploitation balance up to the last iterations of the algorithm. The DGCO dynamically changes the number of individuals in each sub-group during each iteration based on its recorded convergence history. Initially, the DGCO starts with a 70/30 schema, which assigns 70% of the population to the exploration group and the remaining 30% to the exploitation group. Starting with more individuals in the exploration group helps the model to find more promising areas in the search space.

Throughout iterations, the DGCO decreases the number of individuals in the exploration group and increases the number of individuals in the exploitation group. The DGCO applies an elitism operation by electing the best solution in each iteration to be in the next population. This facilitates improvement of the convergence behavior of the algorithm.

At any iteration, the DGCO may increase the number of exploring individuals and decrease the number of individuals in the exploitation group if the fitness of the leader solution does not improve significantly for three consecutive iterations. Finally, the DGCO may randomly interchange the roles of individuals of each group in each iteration to guarantee the diversity of the population. The pseudo-code of the DGCO algorithm is presented in Algorithm 1.

Algorithm 1. Pseudo-code of **DGCO** optimization algorithm

```
Initialize the population $\vec{S} = \{S_1, S_2, \dots, S_d\}$
L = the best solution (leader)

While t < iters_count
    Calculate fitness of each solution, calculate best solution
    k = 2 – 2 * t / iter_count
    Update the number of solutions in each group
    if best fitness did not change from previous 2 iterations
      Increase the number of solutions in the exploration group
    end if

    for each solution in the exploration group
         update $r_1$, $r_2$, and p
          elitism of the best solution
         if p >= 0.5
            mutate the solution
         else
              search around current solution (Eq (4))
```

```
        end if
    end for

    for each solution in the exploitation group
        elitism of the best solution
        update $r_2$, $r_3$, $r_4$, p
          if p >= 0.5
            move towards the best solution (Eq (6))
          else
              search around the best solution (Eq (8))
          end if
    end for
    amend solutions that go beyond the search space
    update prev_fitness1, prev_fitness2
end while
```

5. DGCO-based framework for training the MLP

The proposed framework is based on the DGCO algorithm for training MLP models with a single hidden layer. We will refer to this framework by the abbreviation DGCO-MLP. Two main factors need to be carefully selected when a heuristic optimization algorithm is designated to train an MLP model, namely, the representation of the search agents and the selection of the fitness function, also named the objective function. In the DGCO-MLP, each search agent is represented by a one-dimensional vector that represents the hyperparameters of a candidate MLP model. A group of search agents, also referred to as the population, consists of a number of search agents that evolve from one training epoch to the next one. Each vector that represents an agent in a population contains the weights and biases of the MLP candidate related to that agent. The dimension of each vector is equal to the total number of connections among the perceptrons in the sequentially connected layers of the MLP model, plus the number of biases of those perceptrons, as shown in Eq (10).

$$Dimension\ \ of\ \ agent's\ \ vector = (n \times h) + (h \times 2) + 1 \tag{10}$$

where $\boldsymbol{n}$ is the dimension of the input layer and $\boldsymbol{h}$ is the dimension of the hidden layer. Equation (10) includes the number of connections among input units and hidden perceptrons $(\boldsymbol{n} \times \boldsymbol{h})$, the number of connections among the hidden perceptrons and the output perceptron $(h \times 1)$, and the number of biases of the hidden perceptrons $(h \times 1)$ and the output perceptron $(+1)$. Figure 3 illustrates the compositions of a search agent's vector in the DGCO-MLP framework.

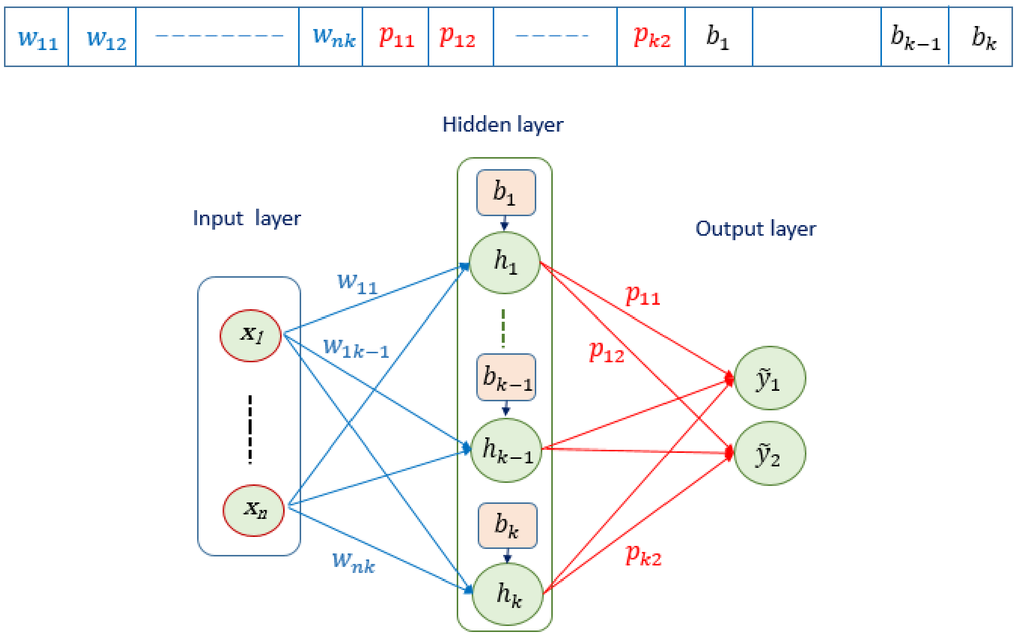

Figure 3. The compositions of a search agent's vector in the DGCO-MLP framework.

We adopted the mean square error (MSE) as the value of the fitness of each search agent. The MSE is calculated as the difference between the actual output of the MLP $\tilde{\boldsymbol{y}}$ and the exact value $\boldsymbol{y}$ that is available for each instance in the training dataset. Equation (11) shows the calculation of the MSE for one epoch of training.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y - \tilde{y})^2 \qquad (11)$$

where $\boldsymbol{n}$ is the number of training instances in the training dataset.

The main steps of the DGCO adopted in the DGCO-MLP framework are as follows:

Step 1. Initialization: The first generation of the search agent's population is randomly initialized. Each search agent designates a potential MLP candidate.

Step 2. Fitness estimation: The potential of each search agent in the pool of candidates is estimated by using a designated fitness function, also named "cost function". This is realized by assigning the weights and biases stored in each vector that represents a search agent to an MLP. Then, the obtained MLP is evaluated by using training and validation datasets. The heuristic-based training aims to find an MLP candidate that produces the minimal MSE during the training phase.

Step 3. Updating the population: The positions of the search agents are dynamically updated by using the equations described in Section 4.

Steps 2 and 3 are repeated until an adopted stopping criterion is reached. The MLP with the optimal fitness score is returned as the solution for the optimization problem. That MLP is then validated on an unseen testing subset.

Figure 4 shows the main steps of the DGCO-MLP approach for MLP learning.

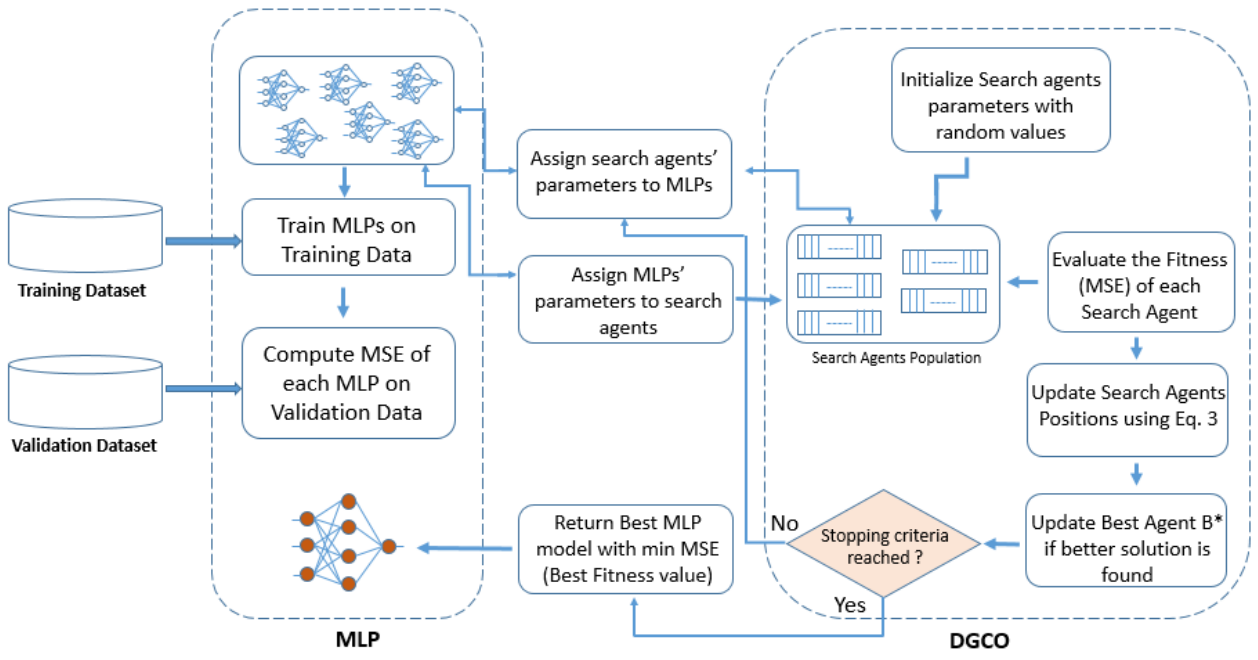
Training Dataset
Validation Dataset
Train MLPs on Training Data
Compute MSE of each MLP on Validation Data
MLP
Assign search agents' parameters to MLPs
Assign MLPs' parameters to search agents
Return Best MLP model with min MSE (Best Fitness value)
Initialize Search agents parameters with random values
Search Agents Population
Evaluate the Fitness (MSE) of each Search Agent
Update Search Agents Positions using Eq. 3
Update Best Agent B* if better solution is found
Stopping criteria reached ?
No
Yes
DGCO

Figure 4. Main chart of the DGCO-MLP framework.

6. Simulations and discussion

We evaluated the proposed DGCO-based approach for training MLPs by using five standard benchmarking datasets for classification. We selected those datasets from the University of California at Irvine machine learning repository [47] and the Kaggle repository [48]. Table 1 shows some parameters of these datasets in terms of the number of training and testing samples, classes, and features. Every selected dataset has a particular number of each of those coordinates. Therefore, we created several MLP models with different numbers of input units to validate the DGCO-MLP against each dataset. Table 2 depicts the MLP structure for each examined dataset. In the literature, several approaches are used to select the number of perceptrons in the hidden layer with no standard that guarantees superiority. In our simulations, we adopted the following formula to decide the number of hidden units in the hidden layer of the examined MLP;

$$h = \frac{n+m}{2} \tag{12}$$

where $\boldsymbol{n}$ is the number of input units, which, in our case, is equal to the number of features in the dataset, and $\boldsymbol{m}$ is the number of output units.

We validated and compared the accuracy of the proposed DGCO-MLP framework against the classical gradient descent backpropagation algorithm and other commonly used heuristic evolutionary optimizers, namely GWO, PSO, and the GA. Table 3 lists the controlling parameters of the examined evolutionary optimizers with their related initial values.

Table 1. Datasets for classification to validate the evolutionary optimization approach.

Dataset	# Classes	# Features	Training Samples	Testing Samples
Parkinson’s	2	22	130	65
Ionosphere	2	34	234	127
Hepatitis	2	10	104	51
Vertebral	2	6	208	103
Water Potability	2	9	1529	764

Table 2. MLP structure for each examined dataset.

Dataset	# Features	MLP Structure
Parkinson’s	22	22-12-1
Ionosphere	34	34-18-1
Hepatitis	10	10-6-1
Vertebral	6	6-4-1
Water Potability	9	9-5-1

Table 3. Controlling parameters of the examined evolutionary optimizers.

Algorithm	Parameter	Value
GWO	population size	200
	maximum number of iterations	350
	randomization interval	[–1, 1]
PSO	population size	200
	maximum number of iterations	350
	acceleration constants c_1 , c_2	{2.1, 2.1}
	inertia weight values	[0.6, 0.9]
GA	population size	200
	maximum number of iterations	350
	crossover probability	0.9
	mutation rate	0.1
	selection mechanism	roulette wheel
DGCO	population size	200
	maximum number of iterations	350
	randomization interval	[–1, 1]
	exploration initialrate	70%

6.1. Simulation setup

In our simulations, we used Python as the primary programming language to implement our version of the DGCO and all of the examined optimization algorithms. Training and validation routines were also implemented and validated with Python frameworks and libraries. We have benefited from the extensive Python libraries and tools such as Scikit-learn 0.24.2 packages that require Python 3.6 or newer versions [49]. Scikit-learn provides many machine learning algorithms, preprocessing techniques, and evaluation tools. All simulation related tasks, including training and testing of examined optimizers and models, were implemented by using the following processor: Intel® Core™ i7-2720QM CPU 2.20 GHz, with 4.00 GB RAM.

All examined datasets were divided into 70% for training and 30% for testing. In a preprocessing step, we normalized each dataset's feature values to eliminate the effect of features with different scales in their values. We adopted the min-max normalization described by Eq (13).

$$p_{scaled} = \frac{p - min(P)}{max(P) - min(P)} \tag{13}$$

where $\boldsymbol{p}$ is the exact value of parameter $\boldsymbol{P}$. $\boldsymbol{max(P)}$ and $\boldsymbol{min(P)}$ are the maximum and minimum values of the parameter $\boldsymbol{P}$, respectively.

All optimization experiments were executed for thirty runs, each with 350 iterations (generations). The simulations were run with the standard DGCO parameter values described in [18].

6.2. Results and discussions

Table 4 shows the statistical scores obtained over the 30 runs of each optimizer on each benchmarking dataset, namely the average (AVG), the standard deviation (STD), the best accuracy (BEST), the worst score (WORST), and the median (MEDIAN).

Table 5 shows each examined trainer's average run time (in seconds), including the heuristic optimizers and the backpropagation learning algorithm. The run time scores in each row were computed as the average elapsed time of an optimizer over thirty runs on each dataset. For heuristic optimizers, each run consists of 350 iterations (generations) of evolution. For the gradient-based trainer, each run consisted of 350 epochs of backpropagation training.

Table 4. Performance scores of tested trainers. Scores in bold rank first, and the underlined rank second.

Dataset		BP	GWO	PSO	GA	DGCO
Parkinson's	AVG	0.65448	0.71791	0.72089	0.71791	0.74039
	STD	0.01626	0.01990	0.02227	0.02213	0.02582
	BEST	0.68656	0.77611	0.77611	0.76119	0.80597
	WORST	0.64179	0.68656	0.68656	0.68656	0.70149
	MEDIAN	0.64179	0.71641	0.71641	0.71641	0.73380
Ionosphere	AVG	0.86416	0.95416	0.93241	0.88875	0.86061
	STD	0.11939	0.01365	0.04871	0.06663	0.07192
	BEST	0.95000	0.98333	0.96666	0.96666	0.97800
	WORST	0.55	0.93333	0.75	0.73333	0.67333
	MEDIAN	0.9125	0.95416	0.94166	0.90416	0.89133
Hepatitis	AVG	0.66666	0.66132	0.66320	0.66698	0.69918
	STD	0.00000	0.02966	0.04297	0.04468	0.03631
	BEST	0.66667	0.73585	0.75472	0.75662	0.78872

Continued on next page

Dataset		BP	GWO	PSO	GA	DGCO
	WORST	0.66666	0.603773585	0.58490	0.56603	0.67490
	MEDIAN	0.66666	0.66037	0.66981	0.66981	0.69811
Vertebral	AVG	0.70755	0.817925	0.816038	0.83396	0.82316
	STD	0.00000	0.026525	0.024003	0.026928	0.021512
	BEST	0.70755	0.84905	0.84905	0.88679	0.86449
	WORST	0.70755	0.754717	0.745283	0.792453	0.79266
	MEDIAN	0.71	0.82	0.82	0.83	0.82
Water Potability	AVG	0.60124	0.61333	0.60000	0.60729	0.64476
	STD	0.00877	0.02818	0.02572	0.02961	0.04716
	BEST	0.60320	0.66667	0.63333	0.66250	0.83167
	WORST	0.56395	0.53333	0.52916	0.55	0.61233
	MEDIAN	0.60319	0.61666	0.60416	0.6125	0.63916

Table 5. Average elapsed time (in seconds) of optimizers against each examined dataset over the simulation runs. Scores in bold indicate the fastest, and the underlined rank second.

Dataset	BP	GWO	PSO	GA	DGCO
Parkinson's	0.369	369.006	294.528	236.692	178.086
Ionosphere	0.508	518.267	401.460	230.194	229.363
Hepatitis	0.193	113.812	111.629	114.144	93.058
Vertebral	0.152	103.754	97.330	108.404	89.642
Water Potability	0.065	156.262	145.451	151.856	131.714

The scores illustrated in Table 4 show that, on most of the datasets, the DGCO-based trainer outperforms the other heuristic and gradient-based trainers to produce the best performing MLP either with the best average of correct classifications over the runs (AVG), or with the best score in one of the elapsed runs (BEST). On other datasets, the DGCO trainer performed competitively on the task of optimizing the MLP models, ranking in the second best place (underlined scores), as shown for the *Ionosphere* and *Vertebral* datasets.

For the datasets *Parkinson's*, *Hepatitis*, and *Water Potability*, the DGCO-based framework provided an amelioration of 2.7%, 4.83%, and 5.13% relative to the second best performing optimizers, respectively. The amelioration for *Water Potability* is given by Eq (14):

$$5.13\% = \frac{0.64476\ -\ 0.61333}{0.61333} \times 100 \tag{14}$$

To quantify the statistical significance of the observed results, we applied the Student's t-test [50]. This statistical significance test aims to check if the differences between the obtained mean values in Table 4 are significant for the cases when the DGCO-MLP optimizer is the best performing. In our experiments, we used a significance level of 0.05. That indicates a 5% risk of concluding that a difference exists when there is no actual difference between the examined means. Table 6 shows the results of the Student's t-test for the scores obtained on the datasets *Parkinson's*, *Hepatitis*, and *Water Potability* where the DGCO-MLP rank first.

The comparative classification scores provided by the DGCO trainer indicate that the DGCO-

MLP framework is a competitive optimizer that can prevent premature convergence toward local minima in the search for the best hyperparameters of an MLP model, namely the synaptic weights and biases. This is due to the excellent exploration capabilities of the DGCO in spaces with either moderate or high numbers of dimensions. Furthermore, the competitive performance of the DGCO-MLP trainer is shown with the datasets that include limited training data records, as illustrated in Tables 1 and 4. The problem of reduced training datasets is an open challenge in machine learning and data science projects, mainly when researchers apply gradient-based trainers.

Table 6. Student's t-test as a statistical significance test.

dataset: *Parkinson's*		
variables	variable 1: PSO	variable 2: DGCO
observations	30	30
means	0.72089	0.74039
variance	0.000495	0.000666
$\boldsymbol{P}(\boldsymbol{T} \leq$t) One Tail	$0.01003 \leq .05$	
$\boldsymbol{P}(\boldsymbol{T} \leq$t) Two Tail	$0.02006 \leq .05$	

dataset: *Hepatitis*		
variables	variable 1: GA	variable 2: DGCO
observations	30	30
means	0.66698	0.69918
variance	0.001996	0.001555
$\boldsymbol{P}(\boldsymbol{T} \leq$t) One Tail	$0.01030 \leq .05$	
$\boldsymbol{P}(\boldsymbol{T} \leq$t) Two Tail	$0.02060 \leq .05$	

dataset: *Water Potability*		
variables	variable 1: GWO	variable 2: DGCO
observations	30	30
means	0.61333	0.64476
variance	0.000794	0.002224
$\boldsymbol{P}(\boldsymbol{T} \leq$t) One Tail	$0.00732 \leq .05$	
$\boldsymbol{P}(\boldsymbol{T} \leq$t) Two Tail	$0.01464 \leq .05$	

The average run times of the examined learners in Table 5 show that the gradient-based backpropagation learner's run time is much shorter than those related to the competing heuristic optimizers. This is expected since those global optimizers involve computing the objective function of multiple solutions in the evolving population in each iteration (generation). In contrast, the gradient descent learner involves the calculation of gradients of an error function. On the other hand, the scores in Table 5 indicate that the DGCO-MLP optimizer is much faster than the other benchmarked optimizers. This could be justified by the dynamic mechanism of controlling the exploration in the DGCO during the evolved populations up to the final steps of this optimizer. Figure 5 shows the convergence curves for the GWO, GA, PSO, and DGCO optimizers when examined on each of the five benchmarking datasets. The curves show the MSEs produced by the MLPs that have been optimized by the five optimizers for a randomly selected run among the thirty independent runs. For clarity, Figure 5 shows the MSE for the first 30 training iterations on each benchmark dataset. The charts indicate that the DGCO exhibits a fast convergence in the first iterations for two datasets. For other classification datasets, the DGCO shows competitive performance compared to the best method in each case.

Figure 6 shows the boxplots for each of the benchmarking classification datasets. The boxplots are depicted for 30 MSEs that were obtained by each optimizer at the beginning of the training phase. Each box in this plot is related to the interquartile range. The whiskers depict the minimum and maximum MSEs, and the bars inside the boxes represent median values. Besides, the small circles represent outliers when they exist.

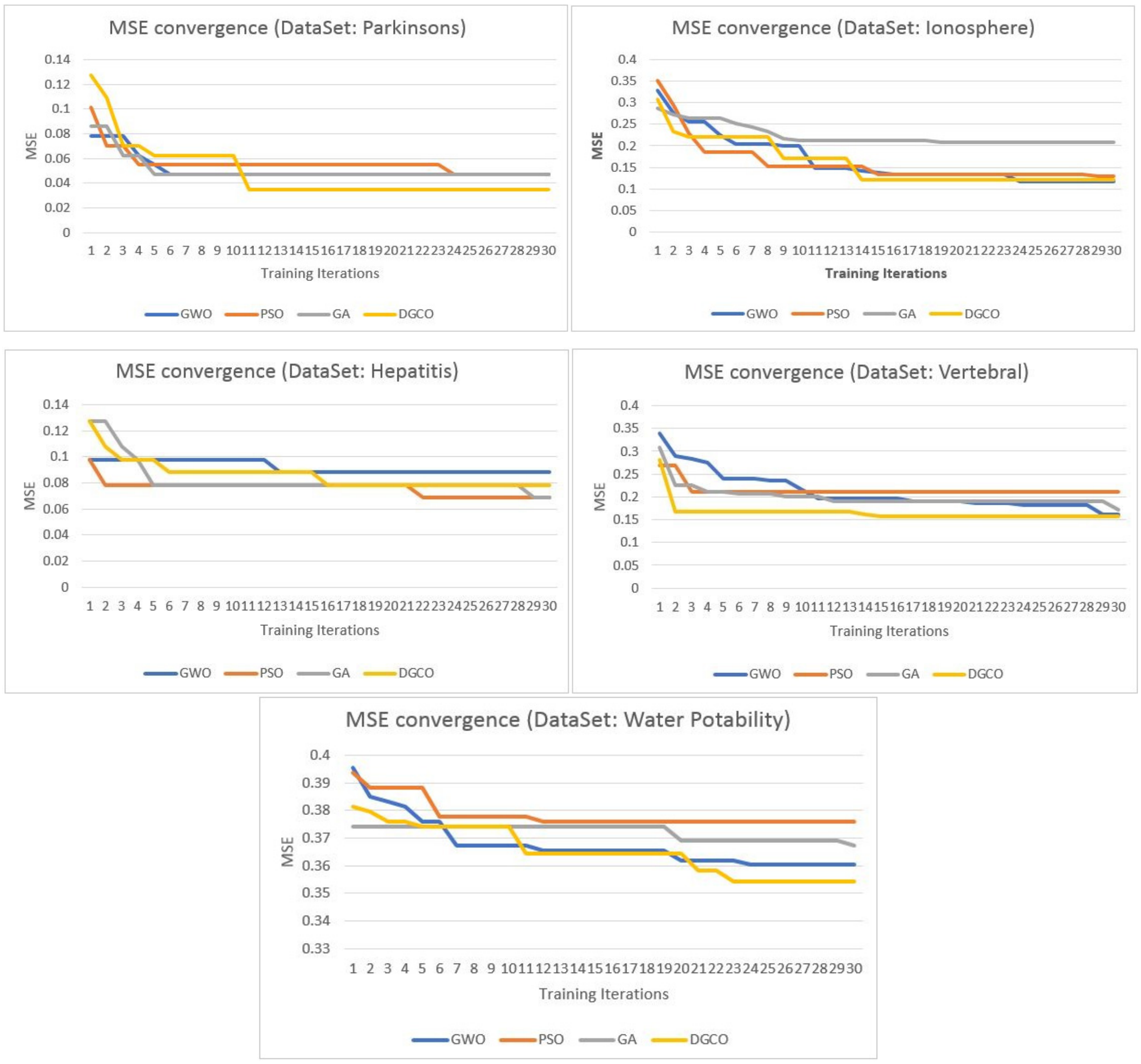

Figure 5. MSE convergence curves for evolutionary optimizers on different classification datasets.

The dynamic changes in the number of individuals in each of the two sub-groups of exploration and exploitation in each iteration is the main characteristic that empowers the DGCO algorithm. This particular feature of the DGCO ensures a progressive exploration of the promising areas in the search space, which helps the model to avoid local minimum points. The algorithm uses the convergence history of the individuals to handle the dynamic changing of individuals' numbers in each sub-group, which causes a balance between exploration and exploitation performance. On the other hand, the exploitation formula permits the rigorous investigation of the local neighbors of the current best agent in the exploitation team. The convergence of the DGCO-MLP trainer is ensured by the elitism mechanism that holds the current best agent $B*$ and transfers it to the next population. Moreover, the search agents in the DGCO tend to search locally around the promising candidate agents. The cooperative behavior of the exploration agents in each generation leads to a high exploration characteristic of the evolutionary algorithm. That allows it to show better local minimum avoidance.

The results indicate that most of the benchmarked evolutionary algorithms, including the DGCO, show competitive scores when dealing with datasets with a small number of training data records as shown for most of the examined datasets. The issue of minor or reduced training datasets in the training of NNs is a significant challenge in machine learning.

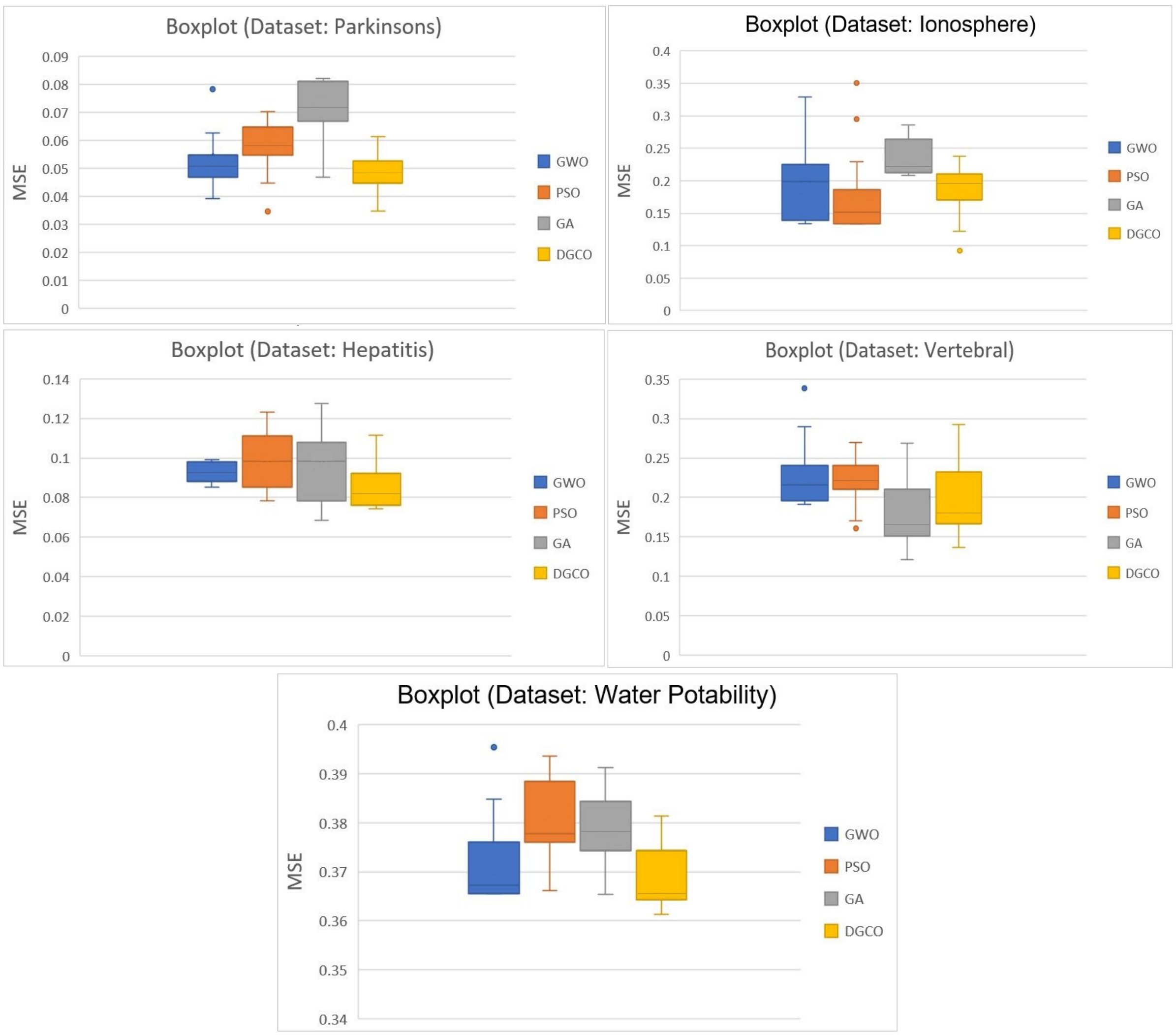

Figure 6. Boxplot charts for different classification datasets.

7. Conclusions

In this research, we propose and validate an EC framework to train MLP models. The consistent convergence and the high local optima avoidance were the essential motivations for introducing the recently proposed evolutionary cooperative optimization algorithm, the DGCO. The objective of the optimization algorithm was to minimize the MSE of MLP responses by optimizing the synaptic weights and biases of those MLPs through EC. Using to the different numbers of features in each of the five examined datasets, we built an MLP model with a particular structure for each dataset to validate our optimization framework against that dataset. Therefore, each MLP had its particular number of input and hidden perceptrons to suit the classification problem described in each dataset. We compared the performance of the proposed DGCO-based against the conventional backpropagation algorithm and several commonly used evolutionary optimization algorithms, namely the GA, GWO, and PSO. The simulation results showed the competitive performance of the proposed optimizer against other examined ones on most of the benchmarked datasets in terms of overall performance and convergence. The DGCO showed its ability to avoid local minima in the loss landscape of the objective function, which is, in our case, the MSE of an MLP response. This is due to the ability of the DGCO to balance between exploration and exploitation capabilities by applying its dynamic mechanism to manage the number of agents in the two cooperative teams of exploration and exploitation. The simulation results concluded that the DGCO-MLP trainer is reliable for training MLPs to classify datasets with different difficulty levels including the number of features and the available training data records.

The DGCO-MLP performed well on the task of optimizing MLP models with a competitive run time on all examined datasets. Moreover, it outperformed the benchmarked optimizers on the task of training MLP models on very limited datasets that include less than 135 data records for training. Optimizing MLPs with limited data is still one of the open challenges.

A minor limitation of our current work is that it studies the optimization of one type of artificial NN, namely the MLPs. Another limitation could be the exploration of the capacity of our framework for the engineering application of classification, as well as the comparison of the performance of this framework with commonly used heuristic optimizers in the literature that did not include comparing it against hybrid optimization approaches. Hence, extending the current simulations to different datasets with more features and many data records is one of the next steps for future work. Another perspective involves applying the proposed framework to optimize the MLP to approximate regression and forecasting task functions. Combining the DGCO evolutionary algorithm with other types of NNs can also be a valuable contribution.

Use of AI tools declaration

The authors declare that they have not used artificial intelligence tools in the creation of this article.

Conflict of interest

The authors declare that there is no conflict of interest.

References

1. S. Haykin, *Neural Networks and Learning Machines*, Prentice Hall, 2011.
2. O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, H. Arshad, State-of-the-art in artificial neural network applications: A survey, *Heliyon*, **4** (2018). https://doi.org/10.1016/j.heliyon.2018.e00938
3. F. Li, M. Sun, EMLP: Short-term gas load forecasting based on ensemble multilayer perceptron with adaptive weight correction, *Math. Biosci. Eng.*, **18** (2021), 1590–1608. https://doi.org/10.3934/mbe.2021082
4. A. Rana, A. S. Rawat, A. Bijalwan, H. Bahuguna, Application of multi layer (perceptron) artificial neural network in the diagnosis system: a systematic review, in *2018 International Conference on Research in Intelligent and Computing in Engineering (RICE)*, (2018), 1–6. https://doi.org/10.1109/RICE.2018.8509069
5. L. C. Velasco, J. F. Bongat, C. Castillon, J. Laurente, E. Tabanao, Days-ahead water level forecasting using artificial neural networks for watersheds, *Math. Biosci. Eng.*, **20** (2023), 758–774. https://doi.org/10.3934/mbe.2023035
6. S. Hochreiter, A. S. Younger, P. R. Conwell, Learning to learn using gradient descent, in *Artificial Neural Networks—ICANN 2001: International Conference Vienna*, (2001), 87–94. https://doi.org/10.1007/3-540-44668-0_13
7. L. M. Saini, M. K. Soni, Artificial neural network-based peak load forecasting using conjugate gradient methods, *IEEE Trans. Power Syst.*, **17** (2002), 907–912. https://doi.org/10.1109/TPWRS.2002.800992
8. H. Adeli, A. Samant, An adaptive conjugate gradient neural network-wavelet model for traffic incident detection, *Comput. Aided Civil Infrast. Eng.*, **15** (2000), 251–260. https://doi.org/10.1111/0885-9507.00189
9. J. Bilski, B. Kowalczyk, A. Marchlewska, J. M. Zurada, Local Levenberg-Marquardt algorithm for learning feedforwad neural networks, *J. Artif. Intell. Soft Comput. Res.*, **10** (2020), 299–316. https://doi.org/10.2478/jaiscr-2020-0020
10. R. Pascanu, T. Mikolov, T. Y. Bengio, On the difficulty of training recurrent neural networks, in *International Conference on Machine Learning*, (2013), 1310–1318.
11. H. Faris, I. Aljarah, S. Mirjalili, Training feedforward neural networks using multi-verse optimizer for binary classification problems, *Appl. Intell.*, **45** (2016), 322–332. https://doi.org/10.1007/s10489-016-0767-1
12. M. Črepinšek, S. H. Liu, M. Mernik, Exploration and exploitation in evolutionary algorithms: A survey, *ACM Comput. Surv.*, **45** (2013), 1–33. https://doi.org/10.1145/2480741.2480752
13. G. Xu, An adaptive parameter tuning of particle swarm optimization algorithm, *Appl. Math. Comput.*, **219** (2013), 4560–4569. https://doi.org/10.1016/j.amc.2012.10.067
14. S. Mirjalili, S. Z. M. Hashim, H. M. Sardroudi, Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm, *Appl. Math. Comput.*, **218** (2012), 11125–11137. https://doi.org/10.1016/j.amc.2012.04.069
15. X. S. Yang, Random walks and optimization, in *Nature Inspired Optimization Algorithms*, Elsevier, (2014), 45–65. https://doi.org/10.1016/B978-0-12-416743-8.00003-8

16. M. Ghasemi, S. Ghavidel, S. Rahmani, A. Roosta, H. Falah, A novel hybrid algorithm of imperialist competitive algorithm and teaching learning algorithm for optimal power flow problem with non-smooth cost functions, *Eng. Appl. Artif. Intell.*, **29** (2014), 54–69. https://doi.org/10.1016/j.engappai.2013.11.003
17. S. Pothiya, I. Ngamroo, W. Kongprawechnon, Ant colony optimisation for economic dispatch problem with non-smooth cost functions, *Int. J. Electr. Power Energy Syst.*, **32** (2010), 478–487. https://doi.org/10.1016/j.ijepes.2009.09.016
18. M. M. Fouad, A. I. El-Desouky, R. Al-Hajj, E. S. M. El-Kenawy, Dynamic group-based cooperative optimization algorithm, *IEEE Access*, **8** (2020), 148378–148403. https://doi.org/10.1109/ACCESS.2020.3015892
19. S. Mirjalili, S. M. Mirjalili, A. Lewis, Grey wolf optimizer, *Adv. Eng. Software*, **69** (2014), 46–61. https://doi.org/10.1016/j.advengsoft.2013.12.007
20. F. Van den Bergh, A. P. Engelbrecht, A cooperative approach to particle swarm optimization, *IEEE Trans. Evol. Comput.*, **8** (2004), 225–239. https://doi.org/10.1109/TEVC.2004.826069
21. C. K. Goh, K. C. Tan, A competitive-cooperative co-evolutionary paradigm for dynamic multi-objective optimization, *IEEE Trans. Evol. Comput.*, **13** (2008), 103–127. https://doi.org/10.1109/TEVC.2008.920671
22. J. H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, 1992. https://doi.org/10.7551/mitpress/1090.001.0001
23. D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*, Addison-Wesley, 1989.
24. EK Burke, EK Burke, G Kendall, G Kendall, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Springer, 2014. https://doi.org/10.1007/978-1-4614-6940-7
25. U. Seiffert, Multiple layer perceptron training using genetic algorithms, in *Proceedings of the European Symposium on Artificial Neural Networks*, (2001), 159–164.
26. F. Ecer, S. Ardabili, S. S. Band, A. Mosavi, Training multilayer perceptron with genetic algorithms and particle swarm optimization for modeling stock price index prediction, **22** (2020), *Entropy*, 1239. https://doi.org/10.3390/e22111239
27. C. Zanchettin, T. B. Ludermir, L. M. Almeida, Hybrid training method for MLP: Optimization of architecture and training, *IEEE Trans. Syst. Man Cyber. Part B*, **41** (2011), 1097–1109. https://doi.org/10.1109/TSMCB.2011.2107035
28. H. Wang, H. Moayedi, L. Kok Foong, Genetic algorithm hybridized with multilayer perceptron to have an economical slope stability design, *Eng. Comput.*, **37** (2021), 3067–3078. https://doi.org/10.1007/s00366-020-00957-5
29. C. C. Ribeiro, P. Hansen, V. Maniezzo, A. Carbonaro, Ant colony optimization: An overview, *Essay Sur. Metaheuristics*, **2002** (2002), 469–492. https://doi.org/10.1007/978-1-4615-1507-4_21
30. M. Dorigo, T. Stützle, *Ant Colony Optimization: Overview and Recent Advances*, Springer International Publishing, (2019), 311–351. https://doi.org/10.1007/978-3-319-91086-4_10
31. D. Karaboga, B. Gorkemli, C. Ozturk, N. Karaboga, A comprehensive survey: Artificial bee colony (ABC) algorithm and applications, *Artif. Intell. Revi.*, **42** (2014), 21–57. https://doi.org/10.1007/s10462-012-9328-0

32. B. A. Garro, R. A. Vázquez, Designing artificial neural networks using particle swarm optimization algorithms, *Comput. Intell. Neurosci.*, **2015** (2015), 61. https://doi.org/10.1155/2015/369298
33. I. Vilovic, N. Burum, Z. Sipus, Ant colony approach in optimization of base station position, in *2009 3rd European Conference on Antennas and Propagation*, (2009), 2882–2886.
34. K. Socha, C. Blum, An ant colony optimization algorithm for continuous optimization: Application to feed-forward neural network training, *Neural Comput. Appl.*, **16** (2007), 235–247. https://doi.org/10.1007/s00521-007-0084-z
35. M. Mavrovouniotis, S. Yang, Training neural networks with ant colony optimization algorithms for pattern classification, *Soft Comput.*, **19** (2015), 1511–1522. https://doi.org/10.1007/s00500-014-1334-5
36. C. Ozturk, D. Karaboga, Hybrid artificial bee colony algorithm for neural network training, in *2011 IEEE Congress of Evolutionary Computation* (CEC), (2011), 84–88. https://doi.org/10.1109/CEC.2011.5949602
37. R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, *J. Global Optimization*, **11** (1997), 341–359. https://doi.org/10.1023/A:1008202821328
38. N. Bacanin, K. Alhazmi, M. Zivkovic, K. Venkatachalam, T. Bezdan, J. Nebhen, Training multi-layer perceptron with enhanced brain storm optimization metaheuristics, *Comput. Mater. Contin*, **70** (2022), 4199–4215. https://doi.org/10.32604/cmc.2022.020449
39. J. Ilonen, J. K. Kamarainen, J. Lampinen, Differential evolution training algorithm for feed-forward neural networks, *Neural Process. Lett.*, **17** (2003), 93–105. https://doi.org/10.1023/A:1022995128597
40. A. Slowik, M. Bialko, Training of artificial neural networks using differential evolution algorithm, in *2008 Conference on Human System Interactions*, (2008), 60–65. https://doi.org/10.1109/HSI.2008.4581409
41. A. A. Bataineh, D. Kaur, S. M. J. Jalali, Multi-layer perceptron training optimization using nature inspired computing, *IEEE Access*, **10** (2022), 36963–36977. https://doi.org/10.1109/ACCESS.2022.3164669
42. K. N. Dehghan, S. R. Mohammadpour, S. H. A. Rahamti, US natural gas consumption analysis via a smart time series approach based on multilayer perceptron ANN tuned by metaheuristic algorithms, in *Handbook of Smart Energy Systems*, Springer International Publishing, (2023), 1–13. https://doi.org/10.1007/978-3-030-72322-4_137-1
43. A. Alimoradi, H. Hajkarimian, H. H. Ahooi, M. Salsabili, Comparison between the performance of four metaheuristic algorithms in training a multilayer perceptron machine for gold grade estimation, *Int. J. Min. Geo-Eng.*, **56** (2022), 97–105. https://doi.org/10.22059/ijmge.2021.314154.594880
44. K. Bandurski, W. Kwedlo, A Lamarckian hybrid of differential evolution and conjugate gradients for neural network training, *Neural Process. Lett.*, **32** (2010), 31–44. https://doi.org/10.1007/s11063-010-9141-1
45. B. Warsito, A. Prahutama, H. Yasin, S. Sumiyati, Hybrid particle swarm and conjugate gradient optimization in neural network for prediction of suspended particulate matter, in *E3S Web of Conferences,* (2019), 25007. https://doi.org/10.1051/e3sconf/201912525007

46. A. Cuk, T. Bezdan, N. Bacanin, M. Zivkovic, K. Venkatachalam, T. A. Rashid, et al., Feedforward multi-layer perceptron training by hybridized method between genetic algorithm and artificial bee colony, *Data Sci. Data Anal. Oppor. Challenges*, **2021** (2021), 279. https://doi.org/10.1201/9781003111290-17-21
47. *UC Irvine Machine Learning Repository*. Available form: http://archive.ics.uci.edu/ml/
48. *Kaggel Database*. Available form: https://www.kaggle.com/datasets/
49. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, et al., Scikit-learn: Machine learning in Python, *J. Mach. Learn. Res.*, **12** (2011), 2825–2830.
50. F. Dick, H. Tevaearai, Significance and limitations of the p value, *Eur. J. Vasc. Endovascular Surg.*, **50** (2015), 815. https://doi.org/10.1016/j.ejvs.2015.07.026