Experienced Grey Wolf Optimizer through Reinforcement Learning and Neural Networks

E. Emary[1], Hossam M. Zawbaa[2,3], Crina Grosan[2,4]
[1]*Faculty of Computers and Information, Cairo University, Egypt*
[2]*Faculty of Mathematics and Computer Science, Babes-Bolyai University, Romania*
[3]*Faculty of Computers and Information, Beni-Suef University, Egypt*
[4]*College of Engineering, Design and Physical Sciences, Brunel University, London, United Kingdom*

***Abstract*—In this paper, a variant of Grey Wolf Optimizer (GWO) that uses reinforcement learning principles combined with neural networks to enhance the performance is proposed. The aim is to overcome, by reinforced learning, the common challenges of setting the right parameters for the algorithm. In GWO, a single parameter is used to control the exploration/exploitation rate which influences the performance of the algorithm. Rather than using a global way to change this parameter for all the agents, we use reinforcement learning to set it on an individual basis. The adaptation of the exploration rate for each agent depends on the agent's own experience and the current terrain of the search space. In order to achieve this, an experience repository is built based on the neural network to map a set of agents' states to a set of corresponding actions that specifically influence the exploration rate. The experience repository is updated by all the search agents to reflect experience and to enhance the future actions continuously. The resulted algorithm is called Experienced Grey Wolf Optimizer (EGWO) and its performance is assessed on solving feature selection problems and on finding optimal weights for neural networks algorithm. We use a set of performance indicators to evaluate the efficiency of the method. Results over various datasets demonstrate an advance of the EGWO over the original GWO and other meta-heuristics such as genetic algorithms and particle swarm optimization.**

***Index Terms*—Reinforcement learning, Neural Network, Grey Wolf Optimization, Adaptive Exploration Rate.**

I. Introduction

A common challenge for learning algorithms is the efficient setting of their parameters, which is usually done empirically, after a number of trials. In the majority of cases, a setting of parameters is valid for that particular problem only, and every time the algorithm is applied, a new set of parameters has to be initialized. The parameter values are at most sub-optimal, as there are never sufficient trials to get the optimal values.

According to [5], learning algorithms suffer from the following:

- *The curse of dimensionality*: many algorithms need to discretize the state space, which is impossible for control problems with high dimensionality because the number of discrete states is enormous.
- *A large number of learning trials*: most algorithms need a large number of learning trials, specifically if the state space size is high, so it is very difficult to apply reinforcement learning to real world tasks.
- *Finding proper parameters for the algorithms*: many algorithms work well, but only with the right parameter setting. Searching for an appropriate parameter setting is, therefore, crucial, in particular for time-consuming learning processes. Thus, algorithms which work with fewer parameters or allow a wider range of parameter setting are preferable.
- *The need of a skilled learner*: since defining the reward function is not enough, we must also determine a good state space representation or a proper function approximator, choose an appropriate algorithm and set the parameters of the algorithm. Consequently, much knowledge and experience are needed when dealing with such learning.

In this paper, we focus on the automatic setting of parameters used by Grey Wolf Optimization (GWO) algorithm, in particular, on the setting of a parameter that plays a significant role in the final result: the exploration/exploitation rate. The meta-learner employed for the automatic parameter setting uses reinforcement learning principles [1] [2].

In the conventional reinforcement learning model, an agent is connected to its environment via perception and action. On each step of interaction, the agent receives as input some indication of the current state of the environment and chooses an action that changes the state, and the value of this state transition is reached to the agent through a scalar reinforcement signal [3]. When the search agent chooses an action, it obtains a feedback for that action and uses the feedback to update its data of state-action map. The goal is to determine the actions that tend to increase the long-run sum of values of the reinforcement signal [4].

In order to determine the right set of actions and, in particular, the right action at each time step in GWO, we propose to use neural networks. The input of the neural network is the set of all individual action history of all the agents, and the output is the action to be taken by a particular agent. Which means, the action of each agent is individually set in a reinforcement learning manner and a neural network learns it. The methodology is explained in detail in section III, after a brief introduction of the original GWO algorithm in section II. The validation of the proposed parameter learning methodology is achieved by considering two optimization problems: feature selection and weight training in neural networks. We perform two types of experiments, one related to feature selection and the other related to multi-layer artificial neural networks (ANNs) weights optimization. Experiments

on 21 datasets are performed in section IV for feature selection and on 10 datasets for ANNs weight training and are validated using various metrics and statistical tests. The results indicate a very promising performance of the proposed methodology compared to the manual parameter setting in the original GWO. A similar approach could be either extended to other learning algorithms or, even more complex, generalized to classes of similar algorithms.

II. Preliminaries

Bio-inspired optimization methods are becoming common among researchers due to their simplicity and extensibility. GWO is a relatively new optimization algorithm inspired by the social hierarchy and hunting behavior of the grey wolves in nature [6]. A modified GWO (mGWO) is proposed which employed a good balance between the exploration and exploitation [7]. A multi-objective variant of GWO is developed to optimize the multi-objective problems [8]. A new binary version of GWO (BGWO) is proposed and applied to feature selection problem [9]. In the next subsections, we briefly explain the general principles of GWO algorithm and the role of the parameters in the search process.

A. Grey Wolf Optimization (GWO)

GWO computationally simulates the hunting mechanism of grey wolves. Grey wolves live in a pack with a strict hierarchy: on the top are the *alpha* wolves, responsible for decision making, followed by *beta* and *delta* wolf. The rest of the pack are called *omegas* [6]. Table I shows the corresponding search and optimization stages of the GWO steps. Naturally, the prey location is the optimal solution and the wolves represent potential solutions in the search space. The wolves closer to the prey are the *alpha* wolves and they are the best solutions so far. Hierarchically, the *beta* wolves are the second best solutions and the *delta* wolves are the third-best solutions. Their location in the search space is represented as X_α, X_β, X_δ. *Omegas* update their position in the search space based on their relative positions from *alpha*, *beta* and *delta* wolves. Figure 1 shows the positioning of the wolves and prey and the parameters involved in the equations used for updating the positions of the wolves in the search space. For hunting prey, a set of steps are to be applied as follows: prey encircling, hunting, attack, and search again.

1) **Prey encircling**: the pack encircles a prey by repositioning individual agents according to the prey location; as in Eq. (1):

$$\overrightarrow{X}(t+1) = \overrightarrow{X}_p(t) + \overrightarrow{A}.\overrightarrow{D}, \tag{1}$$

where t is the iteration, $\overrightarrow{X}_p$ is the prey position, $\overrightarrow{X}$ is the grey wolf position, the . operator indicates vector entry-wise multiplication, and $\overrightarrow{D}$ is as defined as:

$$\overrightarrow{D} = |\overrightarrow{C}.\overrightarrow{X}_p(t) - \overrightarrow{X}(t)|, \tag{2}$$

where $\overrightarrow{A}$, $\overrightarrow{C}$ are coefficient vectors calculated as in Eqs. (3) and (4).

TABLE I
Correspondence between GWO stages and space search

GWO Stage	Interpretation
Prey encircling: adapt wolves position in the search space around the prey in any random position	Update solution positions in the search space around the current optimum
Hunting: adapt wolves positions based on the position of *alpha*, *beta*, *delta* wolves	Update solution position based on the position of the best solutions, considering hierarchically three levels
Attack: move to a position between current position and the prey	Update solution position so that it gets closer to the current optimum by moving towards the optimum
Search: exploration of the search space by diverging the wolves from each other for more prey so that they will converge again for another attack	Exploration of the search space by allowing solution to investigate new areas in order to find a better optimum

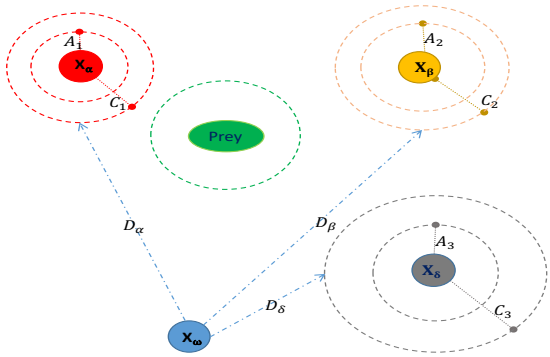


Fig. 1. GWO in the search space positioning (adapted from [6])

$$\overrightarrow{A} = 2a.\overrightarrow{r_1} - a, \tag{3}$$

$$\overrightarrow{C} = 2\overrightarrow{r_2}, \tag{4}$$

where a is linearly diminished over the course of iterations controlling *exploration* and *exploitation*, and $\overrightarrow{r_1}$, $\overrightarrow{r_2}$ are random vectors in the range of [0, 1]. The value of a is the same for all wolves. These equations indicate that a wolf can update its position in the search space around the prey in any random location.

2) **Hunting:** is performed by the whole pack based on the information coming from the *alpha*, *beta*, and *delta* wolves which are expected to know the prey location, as given in Eq. (5):

$$\overrightarrow{X}(t+1) = \frac{\overrightarrow{X_1} + \overrightarrow{X_2} + \overrightarrow{X_3}}{3}, \tag{5}$$

where $\overrightarrow{X_1}$, $\overrightarrow{X_2}$, $\overrightarrow{X_3}$ are defined as in Eqs. (6), (7), and (8) respectively.

$$\overrightarrow{X_1} = |\overrightarrow{X_\alpha} - \overrightarrow{A_1}.\overrightarrow{D_\alpha}|, \tag{6}$$

$$\overrightarrow{X_2} = |\overrightarrow{X_\beta} - \overrightarrow{A_2}.\overrightarrow{D_\beta}|, \tag{7}$$

$$\overrightarrow{X_3} = |\overrightarrow{X_\delta} - \overrightarrow{A_3}.\overrightarrow{D_\delta}|, \tag{8}$$

where $\overrightarrow{X_\alpha}$, $\overrightarrow{X_\beta}$, $\overrightarrow{X_\delta}$ are the first three best solutions at a given iteration t, $\overrightarrow{A_1}$, $\overrightarrow{A_2}$, $\overrightarrow{A_3}$ are defined as in Eq. (3), and $\overrightarrow{D_\alpha}$, $\overrightarrow{D_\beta}$, $\overrightarrow{D_\delta}$ are defined using Eqs. (9), (10), and (11) respectively.

$$\overrightarrow{D_\alpha} = |\overrightarrow{C_1}.\overrightarrow{X_\alpha} - \overrightarrow{X}|, \tag{9}$$

$$\overrightarrow{D_\beta} = |\overrightarrow{C_2}.\overrightarrow{X_\beta} - \overrightarrow{X}|, \tag{10}$$

$$\overrightarrow{D_\delta} = |\overrightarrow{C_3}.\overrightarrow{X_\delta} - \overrightarrow{X}|, \tag{11}$$

where $\overrightarrow{C_1}$, $\overrightarrow{C_2}$, and $\overrightarrow{C_3}$ are defined as in Eq. (4).

This is interpreted by the fact that *alpha*, *beta* and *delta* wolves know best the position of the prey and all the other wolves adapt their positions base on the position of these wolves.

3) **Attacking stage:** the agents approach the prey, which is achieved by decrementing the exploration rate a. Parameter a is linearly updated in each iteration to range from 2 to 0 as in Eq. (12):

$$a = 2 - t\frac{2}{Max_{Iter}}, \tag{12}$$

where t is the iteration number and Max_{Iter} is the total number of iteration allowed for the optimization. According to [6], exploration and exploitation are guaranteed by the adaptive values of a allowing GWO to transit smoothly between exploration and exploitation while half of the iterations are dedicated to the exploration and the other half is assigned to exploitation. This stage is interpreted as wolves moving or changing their position to any random position between their current and the prey positions.

4) **Search for prey:** wolves diverge from each other to search for prey. This behavior is modeled by setting large values for parameter a to allow for exploration of the search space. Hence, the wolves diverge from each other to better explore the search space and then converge again to attack when they find a better prey. Any wolf can find a better prey (optimum). If they get closer to the prey, they will become the new *alphas* and the other wolves will be split into *beta*, *delta* and *omega* according to their distance from the prey. Parameter a gives random weights to the prey and shows the impact of the prey in characterizing the separation of wolves as in the Eqs. (1) and (2). That helps GWO to demonstrate a more random behavior, favoring exploration and local optima evasion. It is worth mentioning that a provides random values at all times keeping in mind the aim to accentuate exploration not only at the beginning of the optimization process but until its end.

GWO is described in Algorithm 1.

Algorithm 1: Grey wolf optimization (GWO)

Input: Number of grey wolves (n), maximum iterations (Max_{Iter}).

Result: The optimal wolf position and its fitness.

1) Initialize a population of n grey wolves positions randomly.
2) Find α, β, and δ as the first three best solutions based on their fitness values.
   while $t \leq Max_{Iter}$ **do**
   foreach $Wolf_i \in pack$ **do**
   Update current wolf's position according to Eq. (5).
   end
   - Update a, A, and C as in Eqs. (2) and (3).
   - Evaluate the positions of individual wolves.
   - Update α, β, and δ positions as in Eqs. (9), (10), and (11).
   end
3) Select the optimal grey wolf position.

B. Feed-forward artificial neural networks (ANNs)

ANNs are a family of models inspired by biological neural networks. They are represented as systems of interconnected neurons that exchange messages between each other. The neural connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning [10]. The primary building block of a neural network is the neuron which has a set of weights. ANN weights are attached to the inputs and an internal nonlinear function maps the input of the neuron to the output given the transfer function as follows:

$$O_i = f(w_i^T.x + b_i), \tag{13}$$

where o_i is the output of neuron i, f is the activation function attached to neuron i, w_i is a vector of weights attached to neuron i, x is the input vector of neuron i, and b_i is a bias scalar value. Each neural network possesses knowledge that is contained in the connections weights values. Changing the knowledge stored in the network as a function of experience implies a learning rule for modifying the values of the weights. Hence, the most challenging problem in using the ANN models is to choose the appropriate weight bias adaptation method [10]. Usually, gradient descent method is used to adapt neural network weights based on the following formula:

$$w_{kj}^{t+1} = w_{kj}^t - \eta\frac{\partial E}{\partial w_{kj}}, \tag{14}$$

where w_{kj}^t is the weight linking neuron k to neuron j at time t, and E is suitable error function that computes the deviation between the targeted and the actual output.

According to the availability of training data, neural models can be to be passive or online. In passive neural models, the neural model is trained using the whole dataset at once while in the case of online models, the data points are not presented as a whole but it is given one at a time [10].

III. The proposed Experienced Grey Wolf Optimizer (EGWO)

Intensification and *diversification* are two key components of any meta-heuristic algorithm [11]. *Intensification* or *exploitation*, uses the available local information to generate better solutions. *Diversification* or *exploration* explores the search space to generate diverse solutions. The balance between exploration and exploitation controls the search behavior of a given optimizer. Excess exploitation makes the optimization process to converge quickly, but it may lead to premature convergence. Excess exploitation increases the probability of finding the global optimum, but often slows down the process with a much lower convergence rate. Therefore, resolving the trade-off between exploration and exploitation is a must to ensure fast global convergence of the optimization process [12]. In almost all modern swarm intelligence optimizers, the exploration rate is adapted throughout the optimization iterations in some predetermined manner to allow for further exploration at some iterations, commonly, at the beginning of the optimization process, and to allow for extra exploitation at some iterations, commonly, at the end of the optimization process. Some optimizers use exploration rate that is linearly inverse proportional to the iteration number. The original GWO linearly decrements the exploration rate as optimization progresses:

$$ExpRate = 2 - (t) * (2/T), \tag{15}$$

where t is the current iteration, T is the total number of iterations, and $ExpRate$ is the rate of exploration. Although this formula proves efficient for solving numerous optimization problems, it still possesses the following drawbacks:

1) *Stagnation*: once the optimizer approaches the end of the optimization process, it becomes difficult to escape the local optima and find better solutions because its exploration capability becomes very limited.

2) *Sub-optimal selection*: at the beginning of the optimization process, the optimizer has very high exploration capability but with this enhanced explorative power it may leave promising regions to less promising ones.

3) *Uniform behaviour*: it is not uncommon to have the same value for the exploration rate for all the search agents, which forces the whole set of wolves to search in the same manner.

The remarks above motivate our research in finding a way to adapt the exploration and to model it individually for each search agent. We proposed to use *experience* as a guide for automatically adjusting the exploration rate. The primary goal is to find a mechanism that maps the different states of search agents to an action set so that the long run goal or fitness function is optimized. To reach such goal and to confront the considerations mentioned above regarding such learning, we defined the following components of the system:

(i) **State-action map**: the mapping between the states and actions is commonly nonlinear and can be modeled using any nonlinear mapping model e.g., neural network model. *Incremental* neural networks rather than the *batch* version of the neural network are more adequate, as every time the agent obtains new data, must use it to update the neural network model.

(ii) **Action set**: for each individual wolf, in order to adapt its exploration rate, we propose a set of actions as follows:

1) *Increase the exploration rate*: takes place as a result of wolf's self-confidence and expertise. This action commonly happens when the wolf finds itself succeeding in some consecutive iterations. The success at a given iteration t is defined by the capability of the search agent to maintain a fitness at time t that is better than its fitness at time $t-1$. This increases its own confidence and hence increases its exploration rate. Another situation that may motivate such action is that when a successive failure occurs, the agent may need to scout in the search space hoping to find better prey.

2) *Decrease exploration rate*: agent's oscillation of fitness may motivate such action and it reflects wrong decision taken by the agent and hence it should be cautious in its movements.

3) *Keep exploration rate*: the current exploration rate is kept static as there is no motivation for neither increase nor decrease it.

The above three action will directly affect the exploration rate at the next iteration as follows:

$$ExpRat_i^{t+1} = \begin{cases} ExpRat_i^t * (1 + \triangle)(\textit{Increase} \text{ action}) \\ ExpRat_i^t * (1 - \triangle)(\textit{Decrease} \text{ action}) \\ ExpRat_i^t(\textit{Keep} \text{ action}), \end{cases} \tag{16}$$

where $ExpRat_i^t$ is the exploration rate for agent i at time t and $\triangle$ is the change factor.

(iii) **Agent state**: the actions performed change the state of a search agent and the agent is repositioned in the search space and hence acquire a new fitness value. A history of fitnesses of a given agent is used to control its next action so that the agent can accumulate and make use of its past decisions. Formally, the state increases, decreases, or keeps constant the fitness value for a search agent over a time span of T past successive rounds:

$$\begin{aligned} State_t^i = [...., sign(f_{t-3}^i - f_{t-4}^i), sign(f_{t-2}^i - f_{t-3}^i), \\ sign(f_{t-1}^i - f_{t-2}^i), sign(f_t^i - f_{t-1}^i)], \end{aligned} \tag{17}$$

where $State_t^i$ is the state vector attached to a given agent i at time t, $f(t)^i$ is the fitness function value for agent i at time t and $sign(x)$ is defined as:

$$sign(x) = \begin{cases} 1 \ if x < 0 \\ -1 \ if x > 0 \\ 0 \text{ otherwise} \end{cases} \tag{18}$$

(iv) **Feedback**: when an agent leaves a region with good fitness for a region with worse fitness, it receives a negative feedback. On the contrary, when the search agent leaves a bad region to a better one, it receives positive feedback. The agent's own fitness is an indicator of the search region around and decides whether the agent should receive positive or negative feedback. Such feedback can be formulated as:

$$Feedback_i^t = \begin{cases} +1 \ if f(agent_i^{t+1}) < f(agent_i^t) \\ -1 \ if f(agent_i^{t+1}) \geq f(agent_i^t), \end{cases} \tag{19}$$

where $Feedback_i^t$ is the feedback for agent i at time t, $f(agent_i^t)$ is the fitness of agent i at time t.

Models that map situations to actions to maximize some reward function are commonly called reinforcement learning models [13]. In such models, the learner is not told which actions to take but instead discovers which actions yield to higher reward by trying them [1]. Two distinguishing features are commonly used to identify such models: (1) *trial-and-error search* and (2) *delayed reward*. According to our problem formulation, we consider a set of *actions*, *state*, *feedback*, and *the state-action mapping model*. The state action mapping model is a neural network with a single hidden layer. ANNs are commonly used to map an unknown (generally nonlinear) function and have well-established training methods [10]. The set of previously mentioned actions *increase*, *decrease*, and *keep* exploration rate are encoded into neural network nodes in the output layer. Therefore, the network has three nodes in the output layer corresponding to the three applicable actions, with 1 on the node indicating the action that will be applied and a 0 value on the other two nodes. The state vector described in Eq. (17) with length T is used as input to the neural model. So, the number of nodes in the input layer is exactly T nodes. The hidden layer has a $2 * T + 1$ nodes [10].

Common to reinforcement learning, the feedback signal, computed as in Eq. (19), is used to adapt the experience of the model through training. The neural weights are adapted according to the action taken and the feedback received at time t. The weights of the neural model are adapted by *rewarding* or *punishing* the winning node. The winning node, representing the applied action, is rewarded by moving its current output to be closer to 1 in the case of receiving *positive* feedback through adapting its attached weights [14]. In the case of receiving *negative* feedback, the node attached to the applied action is *punished* by moving its output to be closer to 0 through updating its assigned weights. Eq. (20) employs the gradient descent method to adapt the weights of the output layer attached to the winning node given a target value of 1 or 0 in the case of positive or negative feedback:

$$w_i^{t+1} = w_i^t + \eta x(d_i - y_i)y_i(1 - y_i), \tag{20}$$

where w_i^t is the weight for output node i at time t, x is the input state, y_i is the actual output on node i, and d_i is the desired output on node i which is set either to 1 when receiving positive feedback or 0 when receiving negative feedback. The update is propagated to the hidden weights as follows:

$$w_i^{t+1} = w_i^t + \eta x y_j(1-y_j)\sum_{i}^{o} w_{ij}((d_i - y_i)y_i(1-y_i)), \tag{21}$$

where η is the learning rate, o is the number of actions, y_j is the output of the hidden node j, y_i is the output of node i, and x is the input state. The wining node/action selection can be formulated as follows:

$$Winner = \min_{i=1}^{3} |1 - o_i|, \tag{22}$$

where o_i is the value of the output node i.

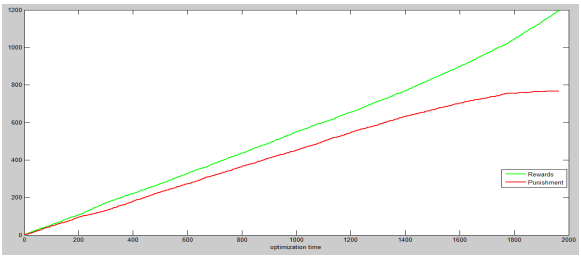


Fig. 2. The number of punishments and rewards received by all the agents during the optimization.

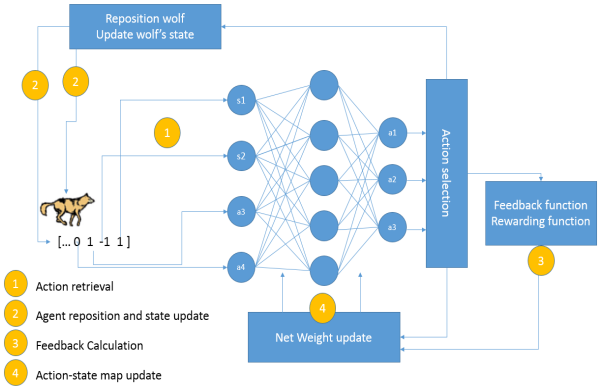


Fig. 3. Flow chart describing the state change of ANN actions

The main assumption to propose such a training model is that the state-action mapping applies a static unknown function to be estimated in a trial and error manner, but if such mapping is not static or changes slightly, the system tends to take random actions and no experience is acquired. Such a training manner is very natural when the trainability is lost in case of fast changing environment [15]. Figure 2 depicts the total number of punishments and rewards acquired by all search agents during the optimization process. We can remark from the figure that as optimization progresses the experience of the search agents is enhanced and hence the number of correct actions (*rewards*), increases, while the number of wrong actions (*punishments*) decreases. This proves the capability of the proposed strategy to converge to optimal state-action map and hence to optimally timed parameters. The proposed algorithm is called *Experienced Grey Wolf Optimizer* (EGWO), and is formally given in Algorithm 2. Figure 3 outlines the flow of state change as a response to actions produced by the neural network.

IV. Experimental Results and Discussions

This section summarizes the results from applying the proposed experienced GWO on two main applications namely *feature selection* and *neural network weight updatation*. The two subsections contain the results and the analysis for each application. We use a set of qualitative measures in order to analyze the results obtained by the methods we apply. The first three metrics give a measure of the mean, best, and worst expected performance of the algorithms. The fourth measure is adopted to show the ability of the optimizer to converge to

Algorithm 2: Experienced Grey wolf optimization (EGWO)

Input: Number of grey wolves (n), maximum iterations (Max_{Iter}), state vector (T), exploration change factor $\triangle$.
Result: The optimal wolf position and its fitness.

1) Randomly initialize a population of n wolves.
2) Initialize a set of n values a_i representing the exploration rate attached to each wolf.
3) Find α, β, and δ solutions based on fitness values.
4) Initialize the state-action neural network to random values.
5) t=0
while $t \leq Max_{Iter}$ **do**
foreach $Wolf_i \in pack$ **do**
- Update wolf's current position according to Eq. (5).
- Calculate the $Feedback_i^t$ according to Eq. 19.
- Update the state-action neural network given $Action_{t-1}$, $state_{t-1}$ and $Feedback_t$ according to Eqs. (20) and (21).
- Update $Wolf_i$ state vector according to Eq. (17).
- Update the exploration parameter a_i attached to $Wolf_i$ as in Eq. (16).
end
- Update A and C as in Eqs. (2) and (3).
- Evaluate the positions of individual wolves.
- Update α, β, and δ positions as in Eqs. (9), (10), and (11).
t=t+1
end
6) Select the optimal grey wolf position.

the same optimal solution. The fifth and sixth metrics show the accuracy of the classification. The seventh and eighth metrics are a measure of the size of the selected features set (we expect a low number of features and a high accuracy). The last three metrics are used for directly comparing two algorithms and show whether the difference between them is significant or not.

1) **Mean fitness:** is an average value of all the solutions in the final sets obtained by an optimizer in a number of individual runs [16].

2) **Best fitness:** is the best solution found by an optimizer in all the final sets resulted from a number of individual runs [16].

3) **Worst fitness:** is the worst solution found by an optimizer in all the final sets resulted from a number of individual runs [16].

4) **Standard deviation (Std):** is used to ensure that the optimizer convergences to the same optimal and ensures repeatability of the results. It is computed over all the sets of final solutions obtained by an optimizer in a number of individual runs [17].

5) **Classifier mean square error (CMSE):** is a measure of classifier's average performance on the test data. It is averaged over all final sets in all the independent runs [18].

6) **Root mean square error (RMSE):** measures the root average squared error of the difference between actual output and the predicted one. It is averaged over all final sets in all the independent runs [18].

7) **Average selected feature:** represents the average size of the selected features subset. The average is computed for each final set of solutions in multiple individual runs.

8) **Average Fisher score:** evaluates a feature subset such that in the data space spanned by the selected features, the distances between data points in different classes are as large as possible, while the distances between data points in the same class are as small as possible [19]. Fisher score in this work is calculated for individual features given the class labels; as follows:

$$F_j = \frac{\sum_{k=1}^{c} n_k(\mu_k^j - \mu^j)^2}{(\sigma^j)^2}, \tag{23}$$

where F_j is the Fisher index for feature j, μ^j and $(\sigma^j)^2$ are the mean and std of the whole dataset, n_k is the size of class k, and μ_k^j is the mean of class k. The Fisher for a set of features is defined as:

$$F_{tot} = \frac{1}{S}\sum_{i=1}^{S} F_i, \tag{24}$$

where S is the number of selected features. The average Fisher score over a set of N runs is defined as:

$$Fishr - score = \frac{1}{N}\sum_{i=1}^{N} F_{tot}^i, \tag{25}$$

F_{tot}^i is the Fisher score computed for selected feature set on run i.

9) **Wilcoxon rank sum test:** is a nonparametric test for significance assessment. The test assigns rank to all the scores considered as one group and then sums the ranks of each group [20]. The test statistic relays on calculating W as in Eq. (26):

$$W = \sum_{i=1}^{N} sgn(x_{2,i} - x_{1,i}).R_i), \tag{26}$$

where $x_{2,i}, x_{1,i}$ is the best fitness obtained from second and first optimizers on run i, R_i is the rank of difference between $x_{2,i} and x_{1,i}$ and $sgn(x)$ is the standard sign function.

10) **T-test:** measures the statistical significance and decides whether or not the difference between the average vaalues of two sample groups reflects the real difference in the population (set) the groups were sampled from[21], as in Eq. (27):

$$t = \frac{\bar{x} - \mu_0}{\frac{S}{\sqrt{n}}} \tag{27}$$

where μ_0 is the mean of the t-distribution and $\frac{S}{\sqrt{n}}$ is its std.

11) **Average run time:** fis the time (in seconds) required by an optimization algorithm for a number of different runs.

A. EGWO applied for feature selection

Finding a feature combination that maximizes a given classifier performance is a challenging problem. A dataset with k features has 2^k different possible choices. The wrapper-based method for feature selection is a very common and reasonably efficient approach but it comes with a huge processing cost as the classifier must be evaluated at each selected feature combination [22]. The main characteristic of the wrapper-based method is the use of the classifier as a guide to feature selection procedure [23]. The ***classifier*** adopted in this study is K-nearest neighbor (KNN) [24]. KNN is a supervised learning algorithm that classifies an unknown sample instance based on the majority of the K-nearest neighbor category. According to the direct formulation of wrapper-based approach for feature selection, the ***evaluation criteria*** (the fitness to optimize) is formulated as:

$$Fitness(D) = \alpha E_M(D) + \beta \frac{\mid M \mid}{\mid N \mid}, \tag{28}$$

where $E_M(D)$ is the error rate for the classifier of condition feature set D, M is the size of selected feature subset, and N is the total number of features. $\alpha \in [0, 1]$ and $\beta = 1 - \alpha$ are constants which control the importance of classification accuracy and feature reduction. Any possible combination of features D can be evaluated and hence this function is continuously defined on the whole feature space with each component of the vector D in the range [0, 1] and such fitness function is generally non-differentiable.

The ***search methods*** employed in the study are the proposed EGWO, GWO, particle swarm optimization (PSO) [25], and genetic algorithms (GA) [26].

1) Initialization: Four initialization methods are adopted in our study differing from one another with respect to the following aspects:

- *Population diversity*: the ability of an optimizer to produce variants of the given initial population is a valuable property.
- *Closeness to expected optimal solution*: the capability to efficiently search the space for the optimal solution is a must for a successful optimizer. Hence, it is intended to force the initial search agents to be apart from or close to the expected optimal solution.
- *Resemblance to forward and backward selection*: each of these has its own strengths and weaknesses, and hence we would like to assess the initialization impact on the feature selection process.

Figure 4 shows the initial wolves positions using four initialization methods and the details about these techniques are as follows:

i) **Small initialization**: search agents are initialized with a small number of randomly selected features. Therefore, if the number of agents is less than the number of features, we will see that each search agent will have a single dimension with value 1. Of course, the optimizer will search for feature(s) to be set to 1 to enhance the fitness function value as in the standard forward selection of features. This method is expected to test the *global search* ability of an optimizer as the initial search agents' positions are commonly away from the expected optimum. Therefore, the optimizer has to use global search operators to derive better solutions.

ii) **Mixed initialization**: half of the search agents are initialized using the small initialization and the other half are initialized using the large initialization method with more random features. Some search agents are close to the expected optimal solution and the other search agents are away from it. Hence, it provides much more *diversity* of the population as the search agents are expected to be far from each other. This method takes both the merits of small and large initialization [27].

iii) **Uniform initialization**: each feature has the same probability of being selected. This method is the most common initialization where the agents are randomly placed in the search space.

iv) **MRMR initialization**: the minimum redundancy maximum relevance (MRMR) combines two criteria for feature selection [28] namely *relevance* with the target class and *redundancy* to other features. In order to define MRMR, we will use the mutual dependence between two random variables X and Y given as:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x;y) log \frac{p(x;y)}{p(x)p(y)}, \tag{29}$$

where $p(x;y)$ is joint probability distribution of x and Y, $p(x)$ and $p(y)$ are the marginal probability distribution functions of X and Y.

For our feature selection problem, let F_i and F_j be two features from the set F of N features and c the class label. The *maximum relevance* method selects the top M best features based on their mutual dependence with class c:

$$\max_F \frac{1}{N} \sum_{F_i \in F} I(F_i; c), \tag{30}$$

These top M features may not be the best M features as there might be correlations among them. Thus, the redundancy is removed using the *minimum redundancy* criterion:

$$\min_F \frac{1}{N^2} \sum_{F_i, F_j \in F} I(F_i; F_j), \tag{31}$$

The *MRMR* initialization combines both these objectives. Iteratively, if $M-1$ features are selected in the set S_{M-1}, the $M-th$ feature is selected by maximizing the single variable relevance minus redundancy:

$$\max_{F_i \in F - S_{M-1}} (I(F_i; c) - \frac{1}{M-1} \sum_{F_j \in S_{M-1}} I(F_i; F_j)), \tag{32}$$

Using the concept of mutual information, the MRMR method selects variables that have the *highest relevance* with the target class and *minimally redundant* as well, i.e. dissimilar to each other. MRMR is normalized along the feature set as:

$$P_i^d = \frac{MRMR_i}{\max_{j=1}^{d} MRMR_j}, \tag{33}$$

TABLE II
DATASETS DESCRIPTION

Data No.	Dataset	No. Features	No. instances
1	**Breastcancer**	9	699
2	**BreastEW**	30	569
3	**Clean1**	166	476
4	**Clean2**	166	6598
5	**CongressEW**	16	435
6	**Exactly**	13	1000
7	**Exactly2**	13	1000
8	**HeartEW**	13	270
9	**IonosphereEW**	34	351
10	**KrvskpEW**	36	3196
11	**Lymphography**	18	148
12	**M-of-n**	13	1000
13	**PenglungEW**	325	073
14	**Semeion**	265	1593
15	**SonarEW**	60	208
16	**SpectEW**	22	267
17	**Tic-tac-toe**	9	958
18	**Vote**	16	300
19	**WaveformEW**	40	5000
20	**WineEW**	13	178
21	**Zoo**	16	101

where P_i is the search agent position in dimension d, $MRMR_i$ is the MRMR value for feature i, and d is the problem dimension.

MRMR is used to initialize one search agent and the rest of search agents are set at random positions in the search space.

Small

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Mixed

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Uniform

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

MRMR

$$\begin{bmatrix} 0.8 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0.1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0.1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Fig. 4. Sample initial wolves positions using different initialization with 9 search agents and 6 dimensions

2) Datasets: 21 datasets in Table II from the UCI machine learning repository [29] are used for tests and comparisons. The datasets are selected to ensure a large variety regarding the number of features and the number of instances. Each dataset is divided randomly into 3 different equal parts for *validation*, *training*, and *testing* using cross-validation. Each experiment iwith each algorithm s repeated 30 times to ensure the stability and the statistical significance of the results.

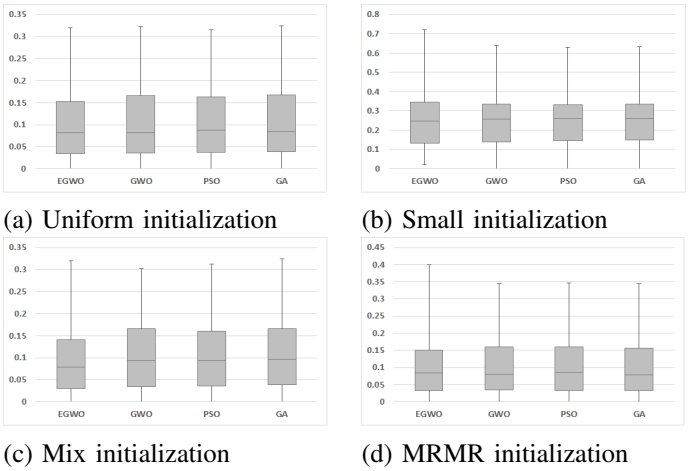
3) Parameter settings: The global and optimizer-specific parameter settings are outlined in Table III. All the parameters

TABLE III
PARAMETER SETTINGS FOR EXPERIMENTS

Parameter	Value(s)
No of search agents	8
No of iterations	70
Problem dimension	Number of features
Search domain	[0 1]
No. runs of each optimizer	30
α in the fitness function	0.99
β in the fitness function	0.01
Δ for changing the exploration rate	0.1
η learning rate for neural network	0.2
T the length of state vector	5
Crossover Fraction in GA	0.8
Inertia factor of PSO	0.1
Individual-best acceleration factor of PSO	0.1

are set either according to domain-specific knowledge as the α, β parameters of the fitness function, or based on a trial and error methodology on small simulations, or from previous experiments reported in the literature in the case of the rest parameters.

4) Results and Discussions: Figure 5-a shows the statistical indicators computed for GWO, EGWO, PSO and GA using the uniform initialization method. We can observe that the performance of EGWO is superior to that of GWO in the average performance. We can also see that the std of EGWO is comparable to that of GWO, which ensures repeatability of results and convergence to the same optimum. The enhanced performance of EGWO is due to the fact that the search agents learn the ill-promising regions of the search space and jump out of these regions to more promising regions. That is possible at all stages, not only at the beginning of search process due to the way in which the exploration rate parameter a_i is controlled. Since every search agent has its own exploration parameter, the diversity of behavior is assured and hence tolerates for stagnation particularly in the end stages of optimization. Individual agents in the EGWO swarm can still increase the exploration rate even in the latter stages of the optimization process, allowing for escaping local minima and premature convergence. In the standard GWO, the exploration rate is linearly decreased during the search process, allowing for a large exploration at the beginning and more local search towards the end. The same conclusion can be derived from the results obtained using small, mix, and MRMR initialization methods in Figures 5-b, 5-c and 5-d, when EGWO performs better in average than the other methods. Small initialization forces the search agents to be initialized away from the expected optimal solution and the agents are forced to keep diversity. Hence, it adds difficulty to the optimizer to escape from such situation. EGWO has an advantage again here as it adapts its exploration rate based on (what has learned from) experience rather than using a formula that does it iteratively and ignoring previous performance and can quickly adapt its exploration rate to approach the global optima while leaving the non-promising regions. Figure 6 outlines snapshot of the exploration rates for 5 agents where we can see the difference in such rate between the different agents which motivates the global fitness function to reach the optimum.



(a) Uniform initialization (b) Small initialization

(c) Mix initialization (d) MRMR initialization

Fig. 5. Box-plot of fitness values for the different initializations

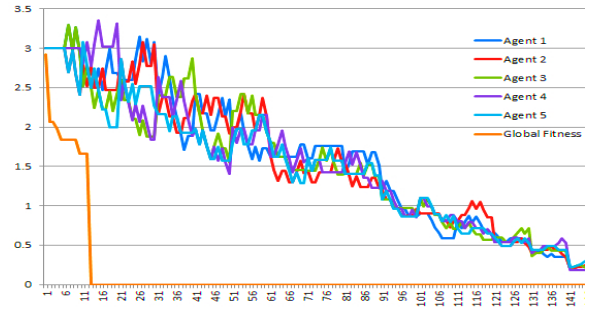


Fig. 6. Five agents with their exploration rates and the global fitness value

When using a *mixed initialization*, all optimizers performs better than using *small initialization* and *uniform initialization* which can be interpreted by the variability of the initial position in the search space, some search agents being initialized close to the expected optimal solution. We can remark that the EGWO still performs better. Figure 7 shows the behavior of a single search agent where we can see that the search agent adapts its own exploration in response to its own fitness and such behavior allows the search agent to keep its exploration capability even at the end stages of the optimization, trying to find better promising regions.

We can observe very good results using the MRMR initialization for all the algorithms. The good performance can

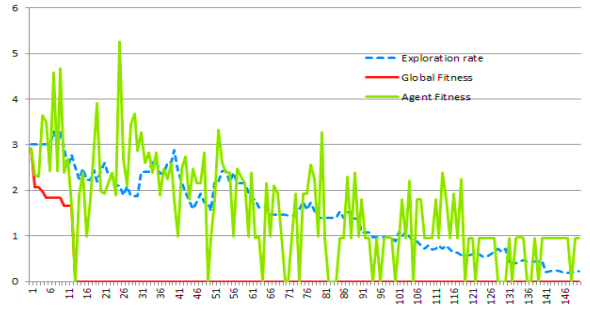


Fig. 7. Single agent with its exploration rate, its fitness and the global fitness

be explained by the fact that the initial population already contains a nearly optimal solution and the optimizer is required only to enhance such solution. All these methods include a mechanism by which the solutions in the swarm tend to follow the best solution found so far. But, there are still situations in which the nearly optimal solution can be far away from the global optimum, requiring again very good explorations of the search space. This is why the exploration ability of EGWO brings it again in advantage compared to the other methods, not only regarding the accuracy of the convergence but also the size of the selected features, EGWO being able to detect the smallest number of relevant features for most of the test cases.

Tables IV-VII show the performance on the test data in terms of classification accuracy, the average size of the selected features set and the average Fisher score. For the uniform initialization where the initial population is very random, EGWO still obtains the smallest features set for about 77% of the datasets, with the best classification accuracy for 72% of the datasets and best Fisher score for over 66% of the data while for the remaining datasets still obtains very good results. For the small initialization, EGWO still dominates the other methods in over 50% of the datasets. Results are better in the case of mixed initialization and MRMR initialization. EGWO obtains the smallest number of features for over 76% of the datasets, the best accuracy for over 57% of the datasets, and the best Fisher score for over 50% of the tests.

Regarding the running time (results in Table VIII), EGWO and GWO perform slower than PSO and GA for the majority of the datasets, but the difference is not really high in all cases. The extra time is consumed in the continual training of the neural network model and the retrieval of action decision. Such increase in time consumption in comparison to enhanced optimization seems to be tolerable in many applications. This could be possibly improved by applying only reinforcement learning at quantized optimization steps. Regarding the added storage cost for the proposed algorithm, the ANN weights could be stored and the short history per wolf could also be kept. The neural model in the current implementation is stored as two matrices with sizes $T \times (2T + 1)$ and $(2T + 1) \times 3$. The wolf's short history is a vector of length T.

Significance tests have the role of comparing two optimizers in order to find a statistically significant difference between them. Table IX shows the results of T-test and Wilcoxon's test calculated on the different initialization methods for all the four optimizers. We are interested in the performance of EGWO against the other algorithms, thus we report all the comparisons with EGWO only. EGWO performs better than GWO, PSO and GA as per T-test results at a significance level of 5% when using uniform and MRMR methods while less importance value is observed when using the small and mixed initializations. This confirms that the learning of the adaptation rate in EGWO really helps the optimization process, regardless of the initial position of the agents, whether close to an optimum or random over the search space. We can also observe that the proposed EGWO has significant advance over PSO and GA using Wilcoxon and T-test at a significance level of 5% regardless of the used initialization method.

TABLE IV
AVERAGE CMSE, SELECTED FEATURE, AND FISHER SCORE USING UNIFORM INITIALIZATION

	CMSE				Selected feature				Fisher score			
Data No.	**EGWO**	**GWO**	**PSO**	**GA**	**EGWO**	**GWO**	**PSO**	**GA**	**EGWO**	**GWO**	**PSO**	**GA**
1	0.040	0.040	0.046	**0.037**	0.457	0.442	0.443	**0.439**	**0.685**	0.676	0.590	0.651
2	**0.058**	0.068	0.065	0.067	**0.326**	0.344	0.340	0.352	**0.255**	0.210	0.214	0.242
3	**0.226**	0.240	0.249	0.239	**0.235**	0.281	0.283	0.293	**0.085**	0.006	0.075	0.046
4	0.045	0.046	**0.042**	0.048	**0.278**	0.345	0.349	0.363	**0.005**	**0.005**	**0.005**	0.046
5	0.060	0.062	**0.053**	0.069	**0.201**	0.222	0.222	0.225	0.160	0.161	**0.189**	0.122
6	0.302	**0.264**	0.264	0.268	**0.376**	0.470	0.476	0.475	0.001	0.001	**0.014**	0.001
7	0.253	0.243	**0.238**	0.250	0.239	0.137	0.145	**0.129**	0.061	0.001	**0.069**	0.057
8	0.236	0.233	**0.226**	0.236	0.470	0.427	0.429	**0.426**	0.088	0.081	0.010	**0.100**
9	**0.151**	0.155	0.153	0.155	**0.222**	0.240	0.241	0.246	**0.048**	0.020	0.041	0.029
10	**0.054**	0.087	0.090	0.092	**0.281**	0.324	0.330	0.321	**0.018**	0.018	0.012	0.010
11	**0.272**	0.273	0.275	0.276	**0.272**	**0.272**	0.273	0.277	**0.120**	0.088	0.107	0.011
12	**0.064**	0.068	0.065	0.077	**0.427**	0.479	0.474	0.489	0.027	**0.029**	0.026	0.006
13	**0.294**	0.327	0.331	0.336	**0.157**	0.188	0.194	0.181	0.121	**0.143**	0.108	0.117
14	**0.042**	0.043	0.048	0.047	**0.232**	0.288	0.290	0.291	**0.006**	**0.006**	0.005	0.003
15	0.312	0.298	**0.298**	0.298	0.289	0.276	0.284	**0.273**	**0.024**	0.013	0.003	0.015
16	**0.193**	0.203	0.197	0.196	0.293	**0.237**	0.237	0.256	**0.018**	0.016	0.016	0.017
17	**0.255**	**0.255**	0.258	0.263	**0.580**	0.617	0.625	0.626	**0.059**	0.050	0.057	0.024
18	**0.070**	0.081	0.083	0.071	**0.208**	0.229	0.234	0.235	0.140	0.136	0.092	**0.157**
19	**0.208**	0.220	0.218	0.221	**0.367**	0.386	0.389	0.389	**0.218**	0.120	0.201	0.068
20	0.065	0.062	**0.060**	0.067	**0.342**	0.373	0.381	0.392	**0.474**	0.461	0.454	0.428
21	0.158	0.153	**0.143**	0.159	**0.292**	0.322	0.328	0.334	**12.576**	11.377	11.282	11.294

TABLE V
AVERAGE CMSE, SELECTED FEATURE, AND FISHER SCORE USING SMALL INITIALIZATION

	CMSE				Selected feature				Fisher score			
Data No.	**EGWO**	**GWO**	**PSO**	**GA**	**EGWO**	**GWO**	**PSO**	**GA**	**EGWO**	**GWO**	**PSO**	**GA**
1	**0.065**	0.066	0.066	0.077	**0.250**	0.259	0.251	0.266	**0.384**	0.368	0.360	0.347
2	**0.152**	0.164	0.175	0.155	**0.031**	0.044	0.035	0.033	0.134	0.127	**0.140**	0.115
3	0.385	0.383	0.386	**0.379**	**0.009**	0.010	0.016	0.013	**0.750**	0.740	0.400	0.011
4	**0.106**	**0.106**	0.109	0.109	0.016	0.013	0.024	**0.012**	0.009	0.006	0.014	**0.015**
5	**0.129**	0.142	0.140	0.141	0.118	0.111	**0.105**	0.125	**0.064**	0.061	0.048	0.062
6	0.312	0.312	0.311	**0.287**	**0.077**	**0.077**	0.087	0.080	0.002	0.002	**0.022**	0.002
7	**0.242**	**0.242**	0.243	0.246	0.077	0.077	0.075	**0.072**	**0.009**	0.008	0.008	0.001
8	0.307	0.280	**0.252**	0.284	**0.128**	0.171	0.171	0.169	0.023	0.028	**0.029**	0.016
9	**0.227**	0.236	0.232	0.244	**0.052**	0.056	0.059	0.061	0.006	0.007	**0.019**	0.002
10	0.445	0.418	0.408	**0.405**	0.043	**0.040**	0.054	0.054	0.001	0.001	**0.009**	0.001
11	0.388	0.365	0.378	**0.357**	0.080	0.080	0.103	**0.075**	**0.068**	0.018	0.030	0.020
12	**0.310**	0.315	0.333	0.316	0.128	0.103	0.107	**0.087**	0.005	0.006	**0.010**	0.001
13	0.535	0.527	0.533	**0.521**	0.006	0.005	0.014	**0.003**	0.004	**0.005**	0.003	**0.005**
14	0.099	0.099	**0.084**	0.107	**0.004**	**0.004**	0.007	0.015	0.004	**0.005**	0.002	0.001
15	0.384	**0.351**	0.368	0.360	**0.020**	0.030	0.024	0.037	0.002	0.002	**0.014**	0.005
16	0.206	0.223	0.215	**0.205**	0.051	0.051	**0.035**	0.049	0.003	0.003	0.002	**0.006**
17	**0.323**	0.334	0.361	0.338	0.185	**0.160**	0.172	0.178	0.003	0.002	**0.023**	0.009
18	0.152	**0.124**	0.136	0.132	0.104	0.104	0.098	**0.098**	**0.076**	0.060	0.072	0.055
19	**0.502**	0.508	0.505	0.507	**0.039**	**0.039**	0.046	0.042	0.011	0.009	0.009	**0.024**
20	0.242	0.148	0.154	**0.136**	**0.165**	0.179	0.174	0.170	**0.282**	0.265	0.270	0.265
21	**0.346**	0.408	0.410	0.412	**0.110**	0.118	0.125	**0.110**	**3.818**	3.044	3.065	3.041

B. Multi-layer ANNs weight adaptation

Feed-forward multi-layer ANNs have been widely used as nonlinear function approximator that maps a given input to a given output. Commonly, the mapping function is either classification function, when the output is in discrete form, or regression, when the output is in continuous form. The main challenge is to estimate the weight set for such a network in order to achieve a proper mapping. Back propagation is commonly used to train the weights but, since it is gradient based, it suffers from convergence to local optima [30]. Recently, other algorithms were employed for such tasks such as GA [31] or PSO [32]. A generic representation of such models selects a weight set that minimizes the total error over all training data (34):

$$minimizeE(W) = \frac{1}{q * O} \sum_{k=1}^{q} \sum_{i=1}^{O} (y_i^k - C_i^k)^2, \qquad (34)$$

where q is the number of training samples, O is the number of nodes in the output layer, y_i^k and C_i^k are the actual and the desired output of the training point k on the output node i, and W is the vector of all neural network weights and biases. The weight range is usually set to [0, 1] or [-1, 1] [30]. The error function is used as a fitness function in the optimization problem of selecting appropriate weights that minimize the error between the predicted and the actual output. The network structure was assumed to be static and the challenge is to select the weight set in the range $[-1, 1]$.

TABLE VI
AVERAGE CMSE, SELECTED FEATURE, AND FISHER SCORE USING MIXED INITIALIZATION

	CMSE				Selected feature				Fisher score			
Data No.	**EGWO**	**GWO**	**PSO**	**GA**	**EGWO**	**GWO**	**PSO**	**GA**	**EGWO**	**GWO**	**PSO**	**GA**
1	0.042	0.042	0.039	**0.036**	**0.457**	0.543	0.544	0.549	0.681	0.746	0.742	**0.751**
2	0.063	0.060	0.059	**0.047**	0.330	0.333	**0.324**	0.339	0.194	0.207	0.202	**0.226**
3	**0.212**	0.235	0.242	0.234	**0.246**	0.255	0.248	0.257	0.005	0.005	0.007	**0.01**
4	**0.046**	**0.046**	0.055	0.064	**0.355**	0.414	0.407	0.394	0.006	0.007	0.011	**0.028**
5	0.071	**0.055**	0.072	0.070	0.250	**0.222**	0.224	0.226	**0.167**	0.158	0.153	0.155
6	0.281	**0.262**	0.297	0.299	**0.513**	0.547	0.564	0.554	0.001	0.001	**0.008**	0.004
7	**0.250**	0.256	0.261	0.263	0.436	0.436	0.433	**0.428**	0.001	0.001	0.004	**0.009**
8	0.237	0.207	**0.204**	0.210	**0.453**	0.513	0.514	0.513	0.077	0.089	0.080	**0.117**
9	0.144	0.149	**0.140**	0.151	**0.222**	0.258	0.264	0.254	**0.033**	0.022	0.025	0.031
10	0.048	0.039	0.051	**0.027**	**0.358**	0.407	0.406	0.424	0.020	0.020	0.007	**0.025**
11	**0.275**	0.282	0.282	0.283	**0.284**	0.370	0.363	0.370	0.110	0.101	0.086	**0.114**
12	**0.017**	0.040	0.041	0.050	**0.462**	0.504	0.516	0.506	**0.038**	0.031	0.037	0.031
13	**0.282**	0.323	0.320	0.322	**0.142**	0.175	0.177	0.165	**0.144**	0.135	0.128	0.137
14	0.042	0.038	0.046	**0.032**	**0.284**	0.350	0.357	0.343	**0.026**	0.008	0.022	0.012
15	**0.281**	0.282	0.297	0.282	**0.265**	0.302	0.310	0.288	**0.042**	0.016	0.015	0.031
16	**0.200**	0.211	0.208	0.213	**0.247**	0.379	0.399	0.386	**0.045**	0.021	0.022	0.033
17	**0.253**	0.259	0.267	0.269	**0.679**	0.753	0.773	0.759	**0.010**	0.006	0.009	0.004
18	**0.070**	0.071	0.081	0.082	0.208	0.181	**0.165**	0.173	**0.138**	0.126	0.134	0.133
19	0.218	0.224	**0.215**	0.216	**0.533**	0.544	0.543	0.573	**0.141**	0.135	0.118	0.123
20	**0.080**	**0.080**	0.092	0.092	0.359	0.376	**0.356**	**0.485**	0.462	0.450	0.480	0.446
21	**0.114**	0.125	0.119	0.137	**0.292**	0.319	0.302	0.323	**15.528**	14.580	14.565	14.593

TABLE VII
AVERAGE CMSE, SELECTED FEATURE, SIZE AND FISHER SCORE USING MRMR INITIALIZATION

	CMSE				Selection Size				Fisher score			
Data No.	**EGWO**	**GWO**	**PSO**	**GA**	**EGWO**	**GWO**	**PSO**	**GA**	**EGWO**	**GWO**	**PSO**	**GA**
1	0.042	0.042	0.039	**0.036**	**0.457**	0.543	0.544	0.549	0.681	0.746	0.742	**0.751**
2	0.063	0.060	0.059	**0.047**	0.330	0.333	**0.324**	0.339	0.194	0.207	0.202	**0.226**
3	**0.212**	0.235	0.242	0.234	**0.246**	0.255	0.248	0.257	**0.007**	0.005	0.007	0.001
4	**0.046**	**0.046**	0.055	0.064	**0.355**	0.414	0.407	0.394	0.006	0.007	0.011	**0.028**
5	0.071	**0.055**	0.072	0.070	0.250	**0.222**	0.224	0.226	**0.167**	0.158	0.153	0.155
6	0.281	**0.262**	0.297	0.299	**0.513**	0.547	0.564	0.554	0.001	0.001	**0.008**	0.004
7	**0.250**	0.256	0.261	0.263	0.436	0.436	0.433	**0.428**	0.001	0.001	0.004	**0.009**
8	0.237	0.207	**0.204**	0.210	**0.453**	0.513	0.514	0.513	**0.097**	0.089	0.080	0.117
9	0.144	0.149	**0.140**	0.151	**0.222**	0.258	0.264	0.254	**0.033**	0.02	0.025	0.031
10	0.048	0.039	0.051	**0.027**	**0.358**	0.407	0.406	0.424	**0.026**	0.020	0.007	0.025
11	**0.275**	0.282	0.282	0.283	**0.284**	0.370	0.363	0.370	0.110	0.101	0.086	**0.114**
12	**0.017**	0.040	0.041	0.050	**0.462**	0.504	0.516	0.506	0.031	**0.031**	**0.037**	0.031
13	**0.282**	0.323	0.320	0.322	**0.142**	0.175	0.177	0.165	**0.144**	0.135	0.128	0.137
14	0.042	0.038	0.046	**0.032**	**0.284**	0.350	0.357	0.343	0.006	0.008	**0.022**	0.012
15	**0.281**	0.282	0.297	0.282	**0.265**	0.302	0.310	0.288	**0.042**	0.016	0.015	0.031
16	**0.200**	0.211	0.208	0.213	**0.247**	0.379	0.399	0.386	**0.045**	0.021	0.022	0.033
17	**0.253**	0.259	0.267	0.269	**0.679**	0.753	0.773	0.759	**0.010**	0.006	0.009	0.004
18	**0.070**	0.071	0.081	0.082	0.208	0.181	**0.165**	0.173	**0.138**	0.126	0.134	0.133
19	0.218	0.224	**0.215**	0.216	**0.533**	0.544	0.543	0.573	**0.141**	0.135	0.118	0.123
20	**0.080**	**0.080**	0.092	0.092	0.359	0.376	**0.356**	0.385	0.462	0.450	**0.480**	0.446
21	**0.114**	0.125	0.119	0.137	**0.292**	0.319	0.302	0.323	12.528	14.580	14.565	**14.593**

GWO has been successfully applied for training multi-layer ANNs for classification and regression problems [33]. In the tests performed in [33], GWO dominates all the other methods used for comparison (GA, PSO, ant colony optimization, evolution strategies, and population-based incremental learning) for the regression tests and is only dominated by the GA for three of the classification tests. This motivates our work to trying to get even better results with our improved EGWO. In this study, a 2 layered feed-forward ANN is used for regression problems. Ten datasets given in Table X are used for experiments. Each individual dataset is divided following a k-fold cross validation manner, with k=10 and the experiments repeated 30 times. In the k-fold cross validation, k^{th} fold is used as testing data and the remaining data is used for training.

1) Results and Discussion: Table XI presents the results of the statistical measures for all the datasets. We can observe that the EGWO has better fitness function value, which proves the capability of the model to converge to better optima. The enhanced performance can be interpreted by the fact that the EGWO can exploit agents' own experience to adapt the exploration rate per search agent. The exploration rate control helps the optimizer to quickly jump to more promising regions especially in complex search space. The results in Table XII show that both GWO and EGWO have comparable std which ensures the repeatability of results regardless of the random factors used. From the performance on the test data, we can observe that EGWO has better performance than GWO. The small difference in the running time is because

TABLE VIII
AVERAGE RUNNING TIME IN SECONDS FOR ALL INITIALIZATION METHODS

Data No.	EGWO	GWO	PSO	GA
1	11.521	11.394	14.228	**9.541**
2	12.322	11.844	9.554	**7.014**
3	44.652	44.737	44.021	**42.424**
4	1100.702	1076.420	**1075.793**	1076.158
5	9.640	9.670	9.744	**4.540**
6	14.699	13.413	13.498	**13.342**
7	14.063	13.016	13.438	**12.143**
8	8.871	8.798	8.083	**4.524**
9	10.094	10.032	12.053	**8.601**
10	**79.255**	80.230	83.375	84.586
11	8.826	8.367	**5.460**	11.342
12	14.784	15.245	15.906	**12.387**
13	54.400	54.147	**49.889**	53.090
14	134.523	**130.479**	133.261	133.105
15	9.950	**9.705**	11.493	10.903
16	9.097	8.877	13.155	**8.105**
17	14.385	14.142	18.379	**13.480**
18	8.795	8.611	**5.514**	13.762
19	220.967	210.184	**209.562**	213.813
20	8.927	8.669	**4.632**	14.005
21	9.083	8.875	**4.864**	8.657

TABLE IX
WILCOXON AND T-TEST FOR ALL INITIALIZATION METHODS

Initializer	Optimizer_1	Optimizer_2	Wilcoxon	T-test
Uniform	**GWO**	**EGWO**	0.100	**0.049**
Uniform	**PSO**	**EGWO**	**0.050**	**0.040**
Uniform	**GA**	**EGWO**	**0.049**	**0.042**
Small	**GWO**	**EGWO**	0.082	0.052
Small	**PSO**	**EGWO**	**0.05**	**0.05**
Small	**GA**	**EGWO**	**0.047**	**0.048**
Mixed	**GWO**	**EGWO**	0.100	0.097
Mixed	**PSO**	**EGWO**	**0.049**	0.07
Mixed	**GA**	**EGWO**	**0.048**	**0.05**
MRMR	**GWO**	**EGWO**	0.077	**0.030**
MRMR	**PSO**	**EGWO**	**0.048**	**0.037**
MRMR	**GA**	**EGWO**	**0.049**	**0.034**

EGWO employs a more complex methodology, but even so, the difference is not really sensitive.

V. CONCLUSION AND FUTURE WORK

In this work, a variant of grey wolf optimizer that learns the exploration rate in an individual manner for each agent (wolf) is proposed. The experienced GWO (EGWO) uses reinforcement learning principles to learn the actions that should be taken at different states of the optimization and in various regions of the search space. A neural network model was used to hold the experience information. The proposed EGWO is compared with the original GWO, PSO, and GA on two main optimization applications: feature selection and ANN weight adaptation. Results were assessed in both applications using a set of performance indicators. We observe a significant improvement in the performance of EGWO while compared to the other methods. EGWO can adapt quickly to different search space terrains and can avoid premature convergence. Besides, the initialization of the search agents positions at the beginning of the optimization process plays a role in the performance of EGWO, with uniform and MRMR initialization providing more variability in the search agents, helping the experienced model to be easily trained. Our methodology is more of a proof of concept that an automatic rather than manual (via trial and error) setting of the parameters of learning algorithms is more efficient, and can be generalized to other similar algorithms, thus helping with the tedious task of always finding the right parameter configuration for a particular application.

TABLE X
DATASET USED FOR REGRESSION

Data No.	Dataset	No. Features	No. instances
1	**CASP**	9	45730
2	**CBM**	17	11934
3	**CCPP**	4	47840
4	**ENB2012_Y1**	8	768
5	**ENB2012_Y2**	8	768
6	**Housing**	13	506
7	**Relation Network**	22	53413
8	**Slump_test**	10	103
9	**Yacht_hydrodynamics**	6	308
10	**forest Fire**	12	517

VI. ACKNOWLEDGMENT

This work was partially supported by the IPROCOM Marie Curie initial training network, funded through the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007-2013/ under REA grant agreement No. 316555, and by the Romanian National Authority for Scientific Research, 333 CNDI-UEFISCDI, project number PN-II-PT-PCCA-2011-3.2-0917. .

REFERENCES

[1] R. Sutton and A. Barto, "Reinforcement learning: An introduction," *MIT Press*, 1998.
[2] X. Xu, Z. Huang, L. Zuo, and H. He, "Manifold-based reinforcement learning via locally linear reconstruction," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–14, 2016.
[3] D. Lee, H. Seo, and M. Jung, "Neural basis of reinforcement learning and decision making," *Annual Review Neuroscience*, vol. 35, pp. 287–308, 2012.
[4] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–12, 2016.
[5] G. Neumann, "The reinforcement learning toolbox, reinforcement learning for optimal control tasks," *Thesis on Institute of Theoretical Computer Science (IGI), University of Technology, Graz*, 2005.
[6] S. Mirjalili, S. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in Engineering Software*, vol. 69, pp. 46–61, 2014.
[7] N. Mittal, U. Singh, and B. Sohi, "Modified grey wolf optimizer for global engineering optimization," *Applied Computational Intelligence and Soft Computing*, vol. 2016, 2016.
[8] S. Mirjalili, S. Saremi, S. Mirjalili, and L. Coelho, "Multi-objective grey wolf optimizer: A novel algorithm for multi-criterion optimization," *Expert Systems with Applications*, vol. 47, pp. 106–119, 2016.
[9] E. Emary, H. Zawbaa, and A. Hassanien, "Binary grey wolf optimization approaches for feature selection," *Neurocomputing*, vol. 172, pp. 371–381, 2016.
[10] C. Bishop, "pattern recognition," *Clarendon press, Oxford*, 1995.
[11] X. Yang, Z. Cui, R. Xiao, A. Gandomi, and M. Karamanoglu, "Swarm intelligence and bio-inspired computation," *Theory and Applications*, 2013.
[12] X. Yang, S. Deb, M. Loomes, and M. Karamanoglu, "A framework for self-tuning optimization algorithm," *Neural Computing and Applications*, vol. 23, no. 7-8, pp. 2051–2057, 2013.

TABLE XI
MEAN, BEST AND WORST OBTAINED FITNESS COMPUTED FOR ANN WEIGHTS ADAPTATION ON REGRESSION PROBLEMS

	Mean fitness				Best fitness				Worst fitness			
Data No.	**EGWO**	**GWO**	**PSO**	**GA**	**EGWO**	**GWO**	**PSO**	**GA**	**EGWO**	**GWO**	**PSO**	**GA**
1	28.250	**28.064**	28.400	29.120	27.714	**27.462**	28.000	29.030	29.208	**28.940**	29.300	30.210
2	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
3	**21.129**	23.603	23.700	24.100	**19.469**	22.067	23.050	23.800	26.595	**22.455**	25.700	25.100
4	**10.465**	11.910	11.980	11.980	10.251	**9.714**	11.020	10.900	**10.713**	14.277	12.980	12.880
5	**12.478**	13.561	14.010	15.200	10.542	**9.930**	12.110	14.200	15.267	17.121	**15.010**	16.200
6	5002.097	4670.525	**4623.525**	5070.525	3484.312	**3058.983**	4521.550	4172.250	6049.616	7613.624	**4613.250**	4970.520
7	31.194	**26.965**	29.021	31.230	21.188	**19.381**	28.210	27.230	46.530	34.350	**30.210**	32.230
8	**0.008**	0.009	0.009	0.009	**0.006**	0.007	0.008	0.008	0.010	0.010	**0.009**	**0.009**
9	**7.263**	8.014	8.110	8.120	**4.281**	4.576	6.410	7.820	10.615	11.841	**8.210**	8.420
10	**55.783**	56.770	57.100	57.020	**36.114**	50.140	56.200	55.120	58.648	74.210	67.100	**57.020**

TABLE XII
AVERAGE STD, RMSE, AND RUN TIME COMPUTED FOR THE DATA REGRESSION DATASETS

	Standard dev.				RMSE				Run time			
Data No.	**EGWO**	**GWO**	**PSO**	**GA**	**EGWO**	**GWO**	**PSO**	**GA**	**EGWO**	**GWO**	**PSO**	**GA**
1	0.831	0.777	0.759	**0.071**	28.364	28.825	28.041	**27.976**	521.445	512.704	511.659	**511.299**
2	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	1.395	0.353	560.938	**555.138**	560.240	559.460
3	2.591	**1.521**	1.636	3.519	22.862	**21.395**	22.535	21.425	563.576	557.476	554.709	**554.533**
4	**0.233**	2.286	2.518	3.089	11.294	**11.261**	13.751	11.289	493.753	492.448	492.383	**490.624**
5	**2.475**	3.596	5.072	4.644	**12.502**	13.347	12.774	13.811	516.478	**510.807**	511.708	511.149
6	**1345.756**	2552.641	2549.730	2555.025	**7046.083**	9339.405	9341.492	9339.002	518.771	511.579	516.759	**509.278**
7	13.486	**7.486**	7.534	8.673	**41.986**	42.833	42.906	43.061	518.126	512.300	**511.748**	514.316
8	**0.002**	**0.002**	0.047	0.040	**0.008**	**0.008**	**0.007**	0.008	**0.008**	0.009	0.009	0.009
9	3.183	3.648	**3.066**	5.619	40.595	36.636	**36.554**	37.956	522.074	515.252	519.330	**512.944**
10	**4.888**	23.743	24.597	22.829	78.676	69.748	**69.588**	70.035	**504.27**	516.118	512.634	511.532

[13] H. Zhang, H. Jiang, Y. Luo, and G. Xiao, "Data-driven optimal consensus control for discrete-time multi-agent systems with unknown dynamics using reinforcement learning method," *IEEE Transactions on Industrial Electronics*, vol. PP, no. 99, pp. 1–1, 2016.

[14] L. Liu, Z. Wang, and H. Zhang, "Adaptive fault-tolerant tracking control for mimo discrete-time systems via reinforcement learning algorithm with less learning parameters," *IEEE Transactions on Automation Science and Engineering*, vol. PP, no. 99, pp. 1–15, 2016.

[15] C. Wei, Z. Zhang, W. Qiao, and L. Qu, "An adaptive network-based reinforcement learning method for mppt control of pmsg wind energy conversion systems," *IEEE Transactions on Power Electronics*, vol. 31, no. 11, pp. 7837–7848, 2016.

[16] S. Tilahun and H. Ong, "Prey-predator algorithm: A new metaheuristic algorithm for optimization problems," *Information Technology & Decision Making*, vol. 14, no. 6, pp. 1331–1352, 2015.

[17] L. Jia, W. Gong, and H. Wu, "An improved self-adaptive control parameter of differential evolution for global optimization," *Computational Intelligence and Intelligent Systems*, vol. 51, pp. 215–224, 2009.

[18] E. Lehmann and G. Casella, "Theory of point estimation, 2nd edition," *New York, Springer*, 1998.

[19] R. Duda, P. Hart, and D. Stork, "Pattern classification," *Wiley-Interscience, 2nd Edition*, 2000.

[20] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[21] J. Rice, "Mathematical statistics and data analysis," *Duxbury Advanced, 3rd edition*, 2006.

[22] L. Chuang, S. Tsai, and C. Yang, "Improved binary particle swarm optimization using catsh effect for attribute selection," *Expert Systems with Applications*, vol. 38, pp. 12699–12707, 2011.

[23] B. Xue, M. Zhang, and W. Browne, "Particle swarm optimisation for feature selection in classification: Novel initialisation and updating mechanisms," *Applied Soft Computing*, vol. 18, pp. 261–276, 2014.

[24] L. Chuang, H. Chang, C. Tu, and C. Yang, "Improved binary pso for feature selection using gene expression data," *Computational Biology and Chemistry*, vol. 32, pp. 29–38, 2008.

[25] J. Kennedy and R. Eberhart, "Particle swarm optimization," *IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, 1995.

[26] A. Eiben, P. Raue, and Z. Ruttkay, "Genetic algorithms with multi-parent recombination," *International Conference on Evolutionary Computation*, vol. 866, pp. 78–87, 1994.

[27] B. Xue, M. Zhang, and W. Browne, "Particle swarm optimisation for feature selection in classification: Novel initialisation and updating mechanisms," *Applied Soft Computing*, vol. 18, pp. 261–276, 2014.

[28] A. Akadi, A. Amine, A. Ouardighi, and D. Aboutajdine, "A two-stage gene selection scheme utilizing mrmr filter and ga wrapper," *Knowledge and Information Systems*, vol. 26, no. 3, pp. 487–500, 2011.

[29] A. Frank and A. Asuncion, "Uci machine learning repository," 2010.

[30] J. Zhang, J. Zhang, T. Lok, and M. Lyu, "A hybrid particle swarm optimization back-propagation algorithm for feedforward neural network training," *Applied Mathematics and Computation*, vol. 185, pp. 1026–1037, 2007.

[31] X. Yao, "A review of evolutionary artificial neural networks," *Intelligent Systems*, vol. 8, no. 4, pp. 539–567, 1993.

[32] C. Zhang, H. Shao, and Y. Li, "Particle swarm optimization for evolving artificial neural network," *System, Man, and Cybernetics*, vol. 4, pp. 2487–2490, 2000.

[33] S. Mirjalili, "How effective is the grey wolf optimizer in training multi-layer perceptrons," *Applied Intelligence*, vol. 43, no. 1, pp. 150–161, 2015.