# A hybrid dragonfly algorithm with extreme learning machine for prediction

Mustafa Abdul Salam[1], Hossam M. Zawbaa[2,3], E. Emary[4,5], Kareem Kamal A.Ghany[3], B. Parv[2]
[1]*Computer Science Department, Modern Academy for Science and Technology, Egypt*
[2]*Faculty of Mathematics and Computer Science, Babes-Bolyai University, Romania*
[3]*Faculty of Computers and Information, Beni-Suef University, Egypt*
[4]*Faculty of Computers and Information, Cairo University, Egypt*
[5]*Faculty of Computer Studies, Arab Open University, Egypt*

***Abstract*—In this work, a proposed hybrid dragonfly algorithm (DA) with extreme learning machine (ELM) system for prediction problem is presented. ELM model is considered a promising method for data regression and classification problems. It has fast training advantage, but it always requires a huge number of nodes in the hidden layer. The usage of a large number of nodes in the hidden layer increases the test/evaluation time of ELM. Also, there is no guarantee of optimality of weights and biases settings on the hidden layer. DA is a recently promising optimization algorithm that mimics the moving behavior of moths. DA is exploited here to select less number of nodes in the hidden layer to speed up the performance of the ELM. It also is used to choose the optimal hidden layer weights and biases. A set of assessment indicators is used to evaluate the proposed and compared methods over ten regression data sets from the UCI repository. Results prove the capability of the proposed DA-ELM model in searching for optimal feature combinations in feature space to enhance ELM generalization ability and prediction accuracy. The proposed model was compared against the set of commonly used optimizers and regression systems. These optimizers are namely, particle swarm optimization (PSO) and genetic algorithm (GA). The proposed DA-ELM model proved an advance overall compared methods in both accuracy and generalization ability.**

***Index Terms*—Extreme Learning Machine, Dragonfly Algorithm, Bio-inspired Optimization, Prediction models.**

## I. Introduction

Neural networks, especially Single hidden layer feed-forward neural network (SLFN) is considered one of the most common machine learning models used in regression and classification domains [1]. Learning algorithm is considered the cornerstone of the neural network. Classical gradient-based learning algorithms such as Levenberg Marquardt (LM) and Scaled Conjugate Gradient (SCG) are suffering from over-fitting, local minima, and they consume long time to learn [2], [3], [4]. Extreme Learning Machine (ELM) [5]was proposed to overcome the problems found in gradient-based learning algorithms. ELM is used as a supervised learning method for SLFN method. ELM has high accuracy and fast prediction speed while solving numerous real-life problems [2], [6]. ELM randomly selects the input weights and hidden layer biases instead of fully tuning all the internal parameters such as gradient-based algorithms. ELM could analytically determine the output weights [2]. Due to random choosing of input weights and hidden layer biases, ELM needs more hidden neurons than gradient-based learning algorithms [7], [8].

978-1-4673-9910-4/16/$31.00 

The bio-inspired algorithms were used in optimizing ELM to overcome its drawbacks. In [7], differential evolution (DE) algorithm was applied to select input weights and biases to determine the output weights of ELM. DE-ELM achieved good generalization performance with a compact structure. In [9] DE-ELM was used for the classification of hyperspectral images, and it improved classification accuracy and computation time. In [10], evolutionary ELM based on PSO algorithm is proposed and PSO algorithm improved the performance of traditional ELM. In [11], a new method combined ELM with an improved PSO called is offered to improve the convergence performance of ELM. In [12], an evolutionary approach for constituting extreme learning machine (ELM) ensembles is introduced to direct the selection of base learners and produced an optimal solution with ensemble size control. In [13], genetic ensemble of ELM is introduced. In [14], a new real-coded genetic algorithm approach called 'RCGA-ELM' is proposed to select the optimal number of hidden neurons, input weights and bias values of ELM model which results in better performance. In [15] ELM based genetic algorithm was applied in power system economic dispatch. In [16], a new Flower Pollination Algorithm (FPA) is proposed to optimize ELM model and achieved better results compared to classical ELM model in stock market prediction.

In this paper, a new proposed DA-ELM model that integrates ELM with a novel dragonfly algorithm (DA) is applied to regression problems. DA is proposed to optimize the input weights and hidden biases of ELM [17]. The rest of this paper is organized as follows: Section II presents the background information of extreme learning machine (ELM) model and dragonfly algorithm (DA) algorithm. Section III is devoted to the proposed approach and its implementation to prediction problems. The experimental results are explained in Section IV, while the main conclusion of the proposed model is presented in Section V.

## II. Preliminaries

### A. Extreme Learning Machine (ELM)

Extreme learning machine (ELM) model has been proposed for single hidden layer feed-forward neural networks (SLFNs). In ELM model, the connections between the input layer and the hidden neurons are randomly selected and remain unchanged during the learning process. The output connections are then tuned via minimizing the cost function through a linear system [18].

Suppose we are training SLFNs with $N$ hidden neurons and activation function $f(x)$ to learn $M$ distinct samples$(x_i, t_i)$,

where: $x_i = [x_{i1}, x_{i2}, ..., x_{ik}]^T \in R^k$ ,$t_i = [t_{i1}, t_{i2}, ..., t_i d]^T \in R^d$. By doing so, the nonlinear system has been converted to a linear system:

$$H\beta = T, \tag{1}$$

where $H$ is the hidden-layer output matrix denoted by:

$$H = \begin{bmatrix} f(w_1.x_1 + b_1) & \dots & f(w_N.x_1 + b_N) \\ \vdots & \ddots & \vdots \\ f(w_1.x_M + b_1) & \dots & f(w_N.x_M + b_N), \end{bmatrix} \tag{2}$$

where $w_j = [w_{j1}, w_{j2}, ..., w_{jk}]^T, (j = 1, 2, ..., N)$ is the weight vector connecting $j_{th}$ hidden neuron and input neurons, and $b_j$ denotes the bias of $j_{th}$ hidden neuron; $w_j.x_i (i = 1, ..., M)$ denotes the inner product of $w_j$ and $x_i$; $\beta = [\beta_1, \beta_2, ..., \beta_N]^T$, the matrix of output weights and $\beta_j = [\beta_j 1, \beta_j 2, ..., \beta_j d]^T$, $(j = 1, 2, ..., N)$ denotes the weight vector connecting the $j_{th}$ hidden neuron and output neurons; $T = [\hat{t_1}, \hat{t_2}, ..., \hat{t_M}]^T$ is the matrix of targets (desired output).

In the case where the $SLFN$ perfectly approximates the data, the errors between the estimated outputs $\hat{t_i}$ and the actual outputs $t_i$ are zero and the relation is:

$$t_i = \sum_{j=1}^{N} \beta_j f(w_j.x_i + b_j), \tag{3}$$

Thus, the determination of the output weights (linking the hidden layer to the output layer) is determined by the least square solution to the linear system represented by equation (4) and the minimum norm least-square (LS) solution to the linear system is:

$$\hat{\beta} = H * T, \tag{4}$$

where $H*$ is the Moore-Penrose (MP) generalized inverse of matrix $H$. The minimum norm LS solution is unique and has the smallest norm among all the LS solutions. ELM using such MP inverse method tends to obtain good generalization performance with dramatically increased learning speed [2].

### B. Dragonfly algorithm (DA)

*1) Inspiration:* Dragonfly algorithm (DA) was recently proposed by Seyedali Mirjalili in 2015 [17]. Dragonflies are amazing insects and there are about 3000 different species of this fancy insect [19]. The life cycle of a dragonfly consists of two main phases: nymph and adult. In the first phase, dragonflies spend the major portion of their lifespan. Then, they undergo to second phase to become adult [19]. Dragonflies look like small predators that hunt almost all other small insects in nature. DA mimics the static (hunting) and the dynamic (migration) swarming behaviours of dragonflies in nature. In static swarm (exploration phase), dragonflies make small groups and fly-back and forth over a small area to hunt other flying preys such as butterflies and mosquitoes [20]. Local movements and abrupt changes in the flying path are the main characteristics of a static swarm. In dynamic swarms (exploitation phase), however, a massive number of dragonflies make the swarm for migrating in one direction over long distances [21].

*2) Dragonfly algorithm:* The static and dynamic swarming behaviours are similar to exploration and exploitation of meta-heuristics optimization. The behaviour of swarms follows three primitive principles [22]:

- *Separation* is the static collision avoidance of the individuals from others in the neighbourhood. It can be calculated as given in equation (5).

$$S_i = -\sum_{j=1}^{N} X - X_j \tag{5}$$

  where $X$ is the position of the current individual, $X_j$ shows the position $j_{th}$ neighbouring individual, and $N$ is the number of neighbouring individuals.
- *Alignment* indicates velocity matching of individuals to the other individuals in neighbourhood. It can be calculated as given in equation (6).

$$A_i = \frac{\sum_{j=1}^{N} V_j}{N} \tag{6}$$

  where $V_j$ shows the velocity of $j_{th}$ neighbouring individual.
- *Cohesion* refers to the tendency of individuals towards the centre of the mass of the neighbourhood. It can be calculated as given in equation (7).

$$C_i = \frac{\sum_{j=1}^{N} X_j}{N} - X \tag{7}$$

  where $X$ is the position of the current individual, $N$ is the number of neighbourhoods, and $X_j$ shows the position $j_t h$ neighbouring individual.

Attraction towards a food source is calculated as in equation (8).

$$F_i = X^+ - X \tag{8}$$

where $X$ is the position of the current individual, and $X^+$ shows the position of the food source.

Distraction outwards an enemy is calculated as in equation (9).

$$E_i = X^- + X \tag{9}$$

where $X$ is the position of the current individual, and $X^-$ shows the position of the enemy.

To update the position of artificial dragonflies in a search space and simulate their movements, two vectors are considered: step $(\Delta X)$ and position $X$. The step vector is as the velocity vector in PSO. The step vector shows the direction of the movement of the dragonflies. It can be defined as in equation (10). Position vectors are calculated as in equation (11).

$$\Delta X_{t+1} = (sS_i + aA_i + cC_i + fF_i + eE_i) + w\Delta X_t \tag{10}$$

where $s$ shows the separation weight, $S_i$ indicates the separation of the $i_th$ individual, $a$ is the alignment weight, $A$ is the alignment of $i_th$ individual, $c$ indicates the cohesion weight, $C_i$ is the cohesion of the $i_th$ individual, $f$ is the food factor, $F_i$ is the food source of the $i_th$ individual, $e$ is the enemy factor, $E_i$ is the position of enemy of the $i_th$ individual, $w$ is the inertia weight, and $t$ is the iteration counter.

$$X_{t+1} = X_t + \Delta X_{t+1} \tag{11}$$

To improve the randomness, stochastic behaviour, and exploration of the artificial dragonflies, they are required to fly around the search space using a random walk (Levy flight) when there is no neighbouring solutions. In this case, the position of dragonflies is updated using the equation (12). The details of dragonfly algorithm (DA) is presented in algorithm 1.

$$X_{t+1} = X_t + Lev'y(d) \times X_t \tag{12}$$

where $t$ is the current iteration, and $d$ is the dimension of the position vectors.

**input** : $N$ Maximum number of dragonflies.
$T$ Number of iterations for optimization.
$s$ Separation weight.
$a$ Alignment weight.
$c$ Cohesion weight.
$f$ Food factor.
$e$ Enemy factor.
$w$ Inertia weight.

**output**: $f_{best}$ Optimal dragon fly position,
$f(f_{best})$ Fitness value for $f_{best}$.

1) Initialize the dragonflies population $X_i$ $(i = 1, 2, ..., n)$.
2) Initialize step vectors $\Delta(X_i)$ $(i = 1, 2, ..., n)$.
3) **while** *Stopping criteria not met* **do**
   - Calculate the fitness of all the $n$ dragon flies.
   - Update the food source and enemy.
   - Update $w$, $s$, $a$, $c$, $f$, and $e$.
   - Calculate $S$, $A$, $C$, $F$, and $E$ according to equations (5), (6), (7), (8), and (9).
   - Update neighbouring radius.

   **if** *A dragonfly has at least one neighbouring dragonfly* **then**
   - Update velocity vector using equation (10).
   - Update position vector using equation (11).

   **else**
   - Update position vector using equation (12).

   **end**
4) Check and correct the new positions based on the boundaries of variables.

**end**

**Algorithm 1:** Dragonfly algorithm (DA)

## III. The Proposed DA-ELM Model

Extreme learning machines have the advantage of low training time while keeping acceptable classification and regression performance on the condition that a huge number of hidden nodes are selected in the model. The huge number of nodes in the hidden layer slows down the testing performance of ELM while there is no grantee of optimality of the setting of weights on the hidden layer. Therefore, in this hybrid model we it is required to selected less number of nodes in the hidden layer to speed-up the performance of the ELM while ensuring optimality by the appropriate selection of hidden layer weights and biases. The proposed model still exploiting the same strategy for setting the weights and biases of the output layer.

DA algorithm is used to optimally select the weights and biases of the hidden layer that maximizes the overall performance of the ELM. The number of variables to be estimated by the DA are $M$ variables with $M = (N_H + 1) * N_I$ where $N_H$ and $N_I$ are the number of nodes in the hidden and input layers in order. The range of individual variables is set to the range [-1, 1]. The used fitness function as given in equation (13).

$$f(w) = \sum_{j=1}^{N_{Validation}} \sum_{i=1}^{N_O} \mid O_{ij} - d_{ij} \mid \tag{13}$$

where $N_{Validation}$ is the number of data points in the validation set, $N_O$ is the number of nodes in the output layer, $O_{ij}$ is the actual output on node $i$ as a result of applying sample $j$ to the ELM, and $d_{ij}$ is the desired output on node $i$ for sample $j$.

We can briefly describe the main components of the algorithm as follows:

1) Initialization: a set of dragonflies positions with each position represents a neural's hidden layer weights and biases and ranges from [-1, 1].
2) Dragonfly position evaluation: a dragonfly position represents a whole set of weights and biases for the hidden

TABLE I: List of data sets used in the experiments

| Name | No. of features | No. of samples |
|---|---|---|
| **CASP** | 9 | 45730 |
| **CBM** | 17 | 11934 |
| **CCPP** | 4 | 47840 |
| **ENB2012_Y1** | 8 | 768 |
| **ENB2012_Y2** | 8 | 768 |
| **ForestFire** | 12 | 517 |
| **Housing** | 13 | 506 |
| **RelationNetwork** | 22 | 53413 |
| **Slump_test** | 10 | 103 |
| **Yacht_hydrodynamics** | 6 | 308 |

TABLE II: Parameters setting

| Algorithm | Parameter | Value |
|---|---|---|
| **DA** | No. of dragonflies | 7 |
| | No. of Iterations | 100 |
| **PSO** | Inertia factor | 0.1 |
| | Individual-best factor | 0.1 |
| **GA** | Crossover_Fraction | 0.8 |
| | Migration_Fraction | 0.2 |
| **ELM** | Input nodes | 7 |
| | Hidden nodes | 100 |
| | Activation fun. | Sigmoid |
| | Output nodes | 1 |
| | No. of Iterations | 1000 |

layer and hence to calculate its fitness the whole neural network is constructed given the hidden layer weights and biases and hence the output layer weights and biases are calculated using the MP generalized inverse matrix given the training data. Hence, the fitness of the agent us calculated as sum square error calculated over the validation data set.

3) Dragonfly repositioning: dragonfly updates their positions according to the native DA algorithm given the calculated fitness.

## IV. Experimental Results

### A. Data Description

The proposed DA-ELM model was trained and tested with 10 regression data sets from the UCI machine-learning repository [23] that used in the experiments and comparisons results. The data sets were selected to have various numbers of *features* and *instances* as representatives of various kinds of issues that the proposed technique will be tested on. In addition, we selected a set of respectively high dimensional data to ensure performance of optimization algorithms in huge search spaces as shown in table (I). Individual data set is divided in cross validation manner for evaluation. In K-fold cross-validation, $K-1$ folds are used for training and validation and the fold remain is used for testing. This process is repeated $M$ times. Hence, individual optimizer is evaluated $K*M$ times for individual data set. The data for training, validation, and testing are equally sized. $Training$ part is used to train the used classifier through optimization and at the final evaluation. $Validation$ part used to assess the performance of the classifier at the optimization time. $Testing$ part is used to evaluate the finally selected features given the trained classifier.

### B. Parameters Settings

The proposed and compared models were trained with one thousand iterations. ELM has seven nodes in input layer representing the six technical indicators and monthly close price. It has one hundred hidden nodes since it needs more hidden nodes than classical gradient training algorithms. It has one node in output layer representing the next week close price. The parameters settings of DA, and ELM algorithms are summarized in table (II).

### C. Performance Evaluation Criteria

Proposed and compared models were evaluated according to three evaluation criteria. These criteria measure the prediction accuracy and the price trend or direction. The evaluation criteria are calculated as follows:

- *Statistical mean:* is the average performance of a stochastic optimization algorithm applied $M$ times and is given in equation (14).

$$Mean = \frac{1}{M}\sum_{i=1}^{M} g_*^i, \tag{14}$$

  where $g_*^i$ is the optimal solution that resulted at the $i-th$ application of the algorithm.
- *Statistical standard deviation (std):* is used as an indicator of the optimizer stability and robustness: when *std* is small, the optimizer always converges to the same solution, whereas large values of *std* represent close to random results, as shown in equation (15).

$$Std = \sqrt{\frac{1}{M-1}\sum (g_*^i - Mean)^2}, \tag{15}$$

- *Root mean square error (RMSE):* measuring the root mean squared error as difference between the actual output and predicted one as given in equation (16).

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(A_i - P_i)^2} \tag{16}$$

  where:
  - $N$ is the number of instances in the test set;
  - $A_i$ is the actual outputs;
  - $P_i$ is the predicted outputs.
- *Average computational time:* is the run time for a given optimization algorithm in millisecond that calculated over the different runs as given in equation (17).

$$T_o = \frac{1}{M}\sum_{i=1}^{M} RunTime_{o,i}, \tag{17}$$

  where $M$ is the number of runs for the optimizer $O$, and $RunTime_{o,i}$ is the computational time in millisecond for optimizer $o$ at run number $i$.

TABLE III: Statistical mean of different optimization algorithms

| Data set | Full | DA | GA | PSO |
|---|---|---|---|---|
| **CASP** | 6.112 | **5.889** | 6.001 | 5.897 |
| **CBM** | 0.008 | 0.007 | 0.007 | **0.007** |
| **CCPP** | 17.084 | **7.012** | 10.077 | 7.390 |
| **ENB2012_Y1** | 9.970 | 6.308 | **3.965** | 3.977 |
| **ENB2012_Y2** | 9.752 | 3.962 | 3.670 | **3.656** |
| **ForestFire** | 43.714 | **42.766** | 52.767 | 52.314 |
| **Housing** | 9.228 | 8.032 | 7.263 | **7.124** |
| **RelationNetwork** | 0.127 | 0.065 | 0.060 | **0.058** |
| **Slump_test** | 7.870 | 4.751 | **4.073** | 4.986 |
| **Yacht_hydrodynamics** | 14.830 | 8.302 | 8.570 | **7.787** |
| **Average** | 11.870 | **8.709** | 9.645 | 9.320 |

TABLE IV: Statistical standard deviation of different optimization algorithms

| Data set | Full | DA | GA | PSO |
|---|---|---|---|---|
| **CASP** | **0.014** | 0.102 | 0.026 | 0.125 |
| **CBM** | 0.000 | 0.000 | **0.000** | 0.000 |
| **CCPP** | 0.482 | 0.968 | **0.025** | 2.257 |
| **ENB2012_Y1** | 0.582 | 0.556 | 0.337 | **0.168** |
| **ENB2012_Y2** | 0.484 | 0.274 | 0.197 | **0.148** |
| **ForestFire** | 17.792 | 18.095 | **16.767** | 18.207 |
| **Housing** | 0.696 | 0.436 | **0.306** | 0.457 |
| **RelationNetwork** | 0.002 | 0.009 | 0.001 | **0.000** |
| **Slump_test** | **0.007** | 1.173 | 0.980 | 1.164 |
| **Yacht_hydrodynamics** | 1.264 | 0.943 | 1.017 | **0.899** |
| **Average** | 2.229 | 2.402 | **2.071** | 2.503 |

TABLE V: Root mean square error of different optimization algorithms

| Data set | Full | DA | GA | PSO |
|---|---|---|---|---|
| **CASP** | 6.136 | 5.907 | 5.945 | **5.862** |
| **CBM** | 0.007 | 0.007 | 0.007 | **0.007** |
| **CCPP** | 17.030 | 10.841 | 9.152 | **7.483** |
| **ENB2012_Y1** | 10.131 | 5.688 | 3.990 | **3.822** |
| **ENB2012_Y2** | 9.372 | 5.433 | **3.589** | 3.662 |
| **ForestFire** | 97.613 | **50.940** | 55.734 | 62.016 |
| **Housing** | **8.854** | 9.189 | 9.326 | 9.553 |
| **RelationNetwork** | **0.127** | 0.129 | 7.838 | 4.879 |
| **Slump_test** | 7.779 | 7.188 | **5.831** | 6.107 |
| **Yacht_hydrodynamics** | 15.450 | 9.125 | 8.974 | **7.103** |
| **Average** | 17.450 | **10.591** | 11.268 | 11.488 |

TABLE VI: Average computational time of different optimization algorithms

| Data set | DA | GA | PSO |
|---|---|---|---|
| **CASP** | 226.060 | **117.361** | 216.430 |
| **CBM** | 152.299 | **71.238** | 133.172 |
| **CCPP** | 134.591 | **69.183** | 131.941 |
| **ENB2012_Y1** | 68.843 | **34.604** | 64.397 |
| **ENB2012_Y2** | 68.382 | **35.692** | 63.983 |
| **ForestFire** | 69.926 | **32.903** | 71.824 |
| **Housing** | 68.958 | **34.417** | 64.367 |
| **RelationNetwork** | 622.903 | **337.764** | 623.369 |
| **Slump_test** | 59.255 | **33.744** | 55.266 |
| **Yacht_hydrodynamics** | 99.257 | **61.284** | 105.269 |
| **Average** | 157.048 | **82.819** | 153.002 |

### *D. Simulation Results analysis*

Table (III) outlines the average statistical mean for each optimization algorithms over the different data sets calculated over the 20 runs. We can remark that all used optimization algorithms outperforms the full features selected, proving the capability of wrapper-based method in feature selection problem. We can also highlight that the DA-ELM performs in general better than GA and PSO, proving the capability of DA to adaptively search the feature space for optimal feature combination and its ability to avoid premature convergence that may be caused by falling in local minima. Same conclusion can be confirmed by remarking the statistical standard deviation (std) on table (IV). DA has a comparable std values with GA and PSO. Table (V) outlines the optimization algorithms performance in prediction of test data averaged over the 20 runs. We can see from the table that DA has the less root mean square error (RMSE) that proves the capability of DA to find optimal feature combination ensuring minimum prediction error. Average computational time over the test data is to some extent compatible with the results obtained as shown in table (VI).

## V. Conclusion and Future work

In this paper, a recent bio-inspired dragonfly algorithm (DA) is proposed to optimize extreme learning machine (ELM) model. DA was used to optimally choose the input weights and hidden layer biases to make network structure more compact, instead of random choosing found in traditional ELM model. The Proposed DA-ELM model is applied to ten regression data sets from UCI repository. The proposed model convergence to a global minimum can be expected in little iterations. The proposed model overcame the over-fitting problem that found in traditional ELM model. DA-ELM model parameters are few and can be tuned easily. Proposed model achieved the lowest error value for all compared evaluation criteria. Proposed DA-ELM model outperformed both GA-ELM and PSO-ELM models. DA is very promising in optimizing ELM model and more research efforts should be devoted in this interesting area.

## 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.

## References

[1] Miche Y., Sorjamaa A., Bas P., Simula O., Jutten C., and Lendasse A. Op-elm: Optimally pruned extreme learning machine. *IEEE Transactions on Neural Networks*, 21(1):158–162, 2010.

[2] Huang G.B., Zhu Q.Y., and Siew C.K. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1):489–501, 2006.

[3] Han F. and Huang D.S. Improved extreme learning machine for function approximation by encoding a priori information. *Neurocomputing*, 69(1):2369–2373, 2006.

[4] Ghany K.K.A., Hefny H.A., Hassanien A.E., and Tolba M.F. Kekre's transform for protecting fingerprint template. *13th International Conference on Hybrid Intelligent Systems*, pages 186–191, 2013.

[5] Huang G.B., Zhu Q.Y., and Siew C.K. Extreme learning machine: a new learning scheme of feedforward neural networks. In *International Joint Conference on Neural Networks (IJCNN 2004)*, pages 985–990. Budapest, Hungary, 2004.
[6] Li X., Xie H., Wang R., Cai Y., Cao J., Wang F., Min H., and Deng X. Empirical analysis: stock market prediction via extreme learning machine. *Neural Computing and Applications*, 1(3):1âĂŞ12, 2014.
[7] Zhu Q.Y., Qin A.K., Suganthan P.N., and Huang G.B. Evolutionary extreme learning machine, pattern recognition. *Pattern Recognition*, 38(10):1759–1763, 2005.
[8] Zhao G.P., Hen Z.Q., Miao C.Y., and Man Z.H. On improving the conditioning of extreme learning machine: a linear case. In *7th International Conference on Information, Communications and Signal Processing(ICICS2009)*, page 1âĂŞ5. Maucu, China, 2009.
[9] Bazi Y., Alajlan N., and Melgani F. Differential evolution extreme learning machine for the classification of hyperspectral images. *Geoscience and Remote Sensing Letters*, 11(6):1066–1070, 2014.
[10] Xu Y. and Shu Y. Evolutionary extreme learning machine-based on particle swarm optimization. *International Symposium on Neural Networks*, 3971(1):644–652, 2006.
[11] Han F., FenYao H., and HuaLing Q. An improved evolutionary extreme learning machine based on particle swarm optimization. *Neurocomputing*, 116(1):87–93, 2013.
[12] Wang D. and Alhamdoosh M. Evolutionary extreme learning machine ensembles with size control. *Neurocomputing*, 102(1):98–110, 2013.
[13] Xue X., Yao M., Wu Z., and Yang J. Genetic ensemble of extreme learning machine. *Neurocomputing*, 129(1):175–184, 2014.
[14] Suresha S., Saraswathib S., and Sundararajanc N. Performance enhancement of extreme learning machine for multi-category sparse data classification problems. *Engineering Applications of Artificial Intelligence*, 23(7):1149âĂŞ1157, 2010.
[15] Yang H., Yia J., Zhao J., and Dong Z. Extreme learning machine based genetic algorithm and its application in power system economic dispatch. *Neurocomputing*, 102(7):154–162, 2013.
[16] Hegazy O., Soliman O.S., and Abdul Salam M. Fpa-elm model for stock market prediction. *International Journal of Advanced Research in Computer Science and Software Engineering*, 5(2):1050–1063, 2015.
[17] S. Mirjalili. Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural Computing and Applications*, 1(1):1âĂŞ–21, 2015.
[18] Jiuwen C. and Zhiping L. Extreme learning machines on high dimensional and large data applications: A survey. *Mathematical Problems in Engineering*, 2015(1):1âĂŞ13, 2015.
[19] Thorp J.H. and Rogers D.C. Thorp and covich's freshwater invertebrates: ecology and general biology. *Elsevier*, 21(1):158–162, 2014.
[20] Wikelski M., Moskowitz D., Adelman J.S., Cochran J., and Wilcove D.S. Simple rules guide dragonfly migration. *Elsevier*, 2(1):325âĂŞ329, 2006.
[21] Russell R.W., May M.L., Soltesz K.L., and Fitzpatrick J.W. Massive swarm migrations of dragonflies (odonata) in eastern north america. *American Midland Naturalist*, 2(140):325–342, 1998.
[22] C.W. Reynolds. Flocks, herds and schools: a distributed behavioral model. *SIGGRAPH Computer Graphics*, 2(21):25–34, 1987.
[23] Bache K. and Lichman M. Uci machine learning repository, 2013.