Engineering Science and Technology, an International Journal xxx (xxxx) xxx

HOSTED BY

Contents lists available at ScienceDirect

Engineering Science and Technology, an International Journal

journal homepage: www.elsevier.com/locate/jestch

A hyper-heuristic based reinforcement-learning algorithm to train feedforward neural networks

Fehmi Burcin Ozsoydan [a,*], İlker Gölcük [b]

[a] Department of Industrial Engineering, Faculty of Engineering, Dokuz Eylül University, İzmir 35397, Turkey
[b] Department of Industrial Engineering, İzmir Bakırçay University, İzmir 35665, Turkey

ARTICLE INFO

Article history:
Received 28 February 2022
Revised 21 June 2022
Accepted 7 September 2022
Available online xxxx

Keywords:
Artificial neural networks
Machine learning
Hyper-heuristics
Particle swarm optimization
Flower pollination algorithm
Differential evolution algorithm

ABSTRACT

Artificial Neural Networks (ANNs) offer unique opportunities in numerous research fields. Due to their remarkable generalization capabilities, they have grabbed attention in solving challenging problems such as classification, function approximation, pattern recognition and image processing that can be quite complex to model mathematically in practice. One of the most vital issues regarding the ANNs is the training process. The aim at this stage is to find the optimum values of ANN parameters such as weights and biases, which indeed embed the whole information of the network. Traditional gradient-descent-based training methods include various algorithms, of which the backpropagation is one of the best-known. Such methods have been shown to exhibit outstanding results, however, they are known have two major theoretical and computational limitations, which are slow convergence speed and possible local minima issues. For this purpose, numerous stochastic search algorithms and heuristic methods have been individually used to train ANNs. However, methods, bringing diverse features of different optimizers together are still lacking in the related literature. In this regard, this paper aims to develop a training algorithm operating based on a hyper-heuristic (HH) framework, which indeed resembles reinforcement learning-based machine learning algorithm. The proposed method is used to train Feed-forward Neural Networks, which are specific forms of ANNs. The proposed HH employs individual metaheuristic algorithms such as Particle Swarm Optimization (PSO), Differential Evolution (DE) Algorithm and Flower Pollination Algorithm (FPA) as low-level heuristics. Based on a feedback mechanism, the proposed HH learns throughout epochs and encourages or discourages the related metaheuristic. Thus, due its stochastic nature, HH attempts to avoid local minima, while utilizing promising regions in search space more conveniently by increasing the probability of invoking relatively more promising heuristics during training. The proposed method is tested in both function approximation and classification problems, which have been adopted from UCI machine learning repository and existing literature. According to the comprehensive experimental study and statistically verified results, which point out significant improvements, the developed HH based training algorithm can achieve significantly superior results to some of the compared optimizers.

© 2022 Karabuk University. Publishing services by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

With striking characteristics and notable advances, Artificial Neural Networks (ANNs) [1] open up new opportunities in modeling complex and non-linear problems. As mathematically and biologically inspired by the brain processing system of human beings [2], ANNs are referred to as information processing systems that are capable of performing various tasks related to artificial intelligence research field. Since ANNs are shown to exhibit outstanding performances in a wide range of research fields such as regression analysis, image processing, classification, pattern recognition, prediction, robotics, and signal processing [3], they grab attention of researchers from machine learning and artificial intelligence community.

One of the most important issues related to ANNs is the training stage. Assuming that the problem data is properly and correctly preprocessed, generalization capability of ANNs mostly depend on the success of training stage. Therefore, the algorithm adopted at this stage becomes vital. Generally speaking, the algorithms that can be used at the training stage can be categorized into two main

* Corresponding author.
E-mail address: burcin.ozsoydan@deu.edu.tr (F.B. Ozsoydan).
Peer review under responsibility of Karabuk University.

https://doi.org/10.1016/j.jestch.2022.101261
2215-0986/© 2022 Karabuk University. Publishing services by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Please cite this article as: F.B. Ozsoydan and İ. Gölcük, A hyper-heuristic based reinforcement-learning algorithm to train feedforward neural networks, Engineering Science and Technology, an International Journal, https://doi.org/10.1016/j.jestch.2022.101261

categories such as deterministic and heuristic-based algorithms. The former category includes mostly gradient-descent-based training methods of which the backpropagation [4] is one of the best-known. Such methods have been shown to exhibit remarkable results, however, they are known have two major theoretical and computational limitations, which are slow convergence speed and possible local minima issues. Moreover, they are quite dependent to the initial coordinates of search space. As the latter category, metaheuristic algorithms are known to be capable of avoiding local optima due to their intrinsic stochastic search mechanisms. Moreover, as in gradient-descend-based methods, derivative information of the activation functions and the error function are not used, thus chosen activation functions and the error function do not necessarily have to be differentiable. However, metaheuristic algorithms require for more CPU usage and they are slower than deterministic optimization algorithms [5]. Nevertheless, as discussed in the next section, they have been widely used for training ANNs due to such remarkable advantages.

Metaheuristics have been shown to exhibit outstanding performances in various research fields and they can be further improved by several reinforcement mechanisms. In this context, hyper-heuristics (HH) [6] emerge as high-level reinforcement mechanisms, which use sets of low-level heuristics. They can be enhanced by a learning mechanism based on feedbacks gathered throughout search. Moreover, they can deploy various metaheuristic algorithms to utilize their individual and distinctive features. As one can see from the related literature, methods, bringing diverse features of such optimizers together are still lacking.

As motivated by such issues, this paper aims to develop a training strategy operating based on a HH framework to train Feed-forward Neural Networks (FNN), which are indeed specific forms of ANNs. The proposed HH employs Particle Swarm Optimization (PSO) [7–8], Differential Evolution (DE) Algorithm [9] and Flower Pollination Algorithm (FPA) [10] as low-level heuristics. It is enhanced by a reinforcement learning mechanism that encourages or discourages the related algorithm according to their performances. Thus, HH attempts to avoid local minima by occasionally invoking naïve algorithms while utilizing promising regions in search space more conveniently by increasing the probability of invoking more promising algorithms during training. The proposed training strategy is tested in both function approximation and classification problems, which have been adopted from UCI machine learning repository and existing literature. According to the comprehensive experimental study, statistically verified significant improvements can be acquired by the proposed method.

The rest of this paper is organized as follows: Section 2 reviews the related literature. All used algorithms and corresponding details are explained in details in Section 3. Experimental study and concluding remarks are reported in Section 4 Section 5, respectively.

2. Related work

Sexton and Gupta [11] compare the performances of Genetic Algorithms (GAs) and backpropagation method in training ANNs. One year later, Kuo et al. [12] introduce a GA based fuzzy neural network as an intelligent stock trading decision support system. Lu et al. [13] report a PSO-trained multi-layer-perceptron (MLP) to predict pollutant levels in downtown area of Hong Kong. According to the results presented in this study, the proposed approach achieve promising results. In the same year, Gudise and Venayagamoorthy [14] compare the performances of backpropagation and PSO.

Pendharkar and Rodger [15] use GAs that adopt various recombination techniques in order to train an MLP. Juang [16] uses a hybrid of GA and PSO to find optimum design parameters of recurrent neural networks (RNNs). Another GA to evolve FNNs for multispectral image classification is reported by Liu et al. [17]. One year later, Da and Xiurun [18] present a modified PSO that can select also the non-improving particles with respect to a probabilistic selection mechanism. Yu et al. [19] propose an improved PSO, which tunes its own parameters during search to evaluate parameters of an MLP for classification problems. According to the reported results, PSO shows a great potential for training MLPs. In the same year, Zhang et al. [20] make use of a hybrid of backpropagation method and PSO to solve both classification and function approximation problems. In the proposed method, if PSO stagnates on the same coordinates for a period, backpropagation is invoked to make a detailed search around the found promising region. Yu et al. [21] propose two PSO variants to determine both the structure and the weights of an FNN for classification tasks. In this regard, the authors propose a specialized encoding technique that can simultaneously handle these two distinct problems. Kiranyaz et al. [22] propose an evolutionary neural network that is based on PSO to find the optimal hyper-parameters of FNNs.

Khan and Sahai [23] use several metaheuristic algorithms including PSO and GA to train FNNs. The authors compare the found results to those of the found by gradient-descent-based training algorithms. As the authors report, the results of the metaheuristic algorithms are found as superior. In the same year, Kawam and Mansour [24] report several bio-inspired algorithms to train FNNs for classifications tasks. In the same year, Rakitianskaia and Engelbrecht [5] propose a dynamic PSO that is capable of handling dynamic decision boundaries for dynamic classification problems, modeled by ANNs. Mirjalili et al. [25] employ metaheuristic algorithms to train FNNs for both classification and approximation problems. One year later, Askarzadeh and Rezazadeh [26] employ a bio-inspired metaheuristic algorithm to train FNNs. In their study, Piotrowski et al. [27] compare several metaheuristics to train an MLP that is used to forecast river temperature. Mirjalili [28] adopts a bio-inspired algorithm to train MLPs. According to the reported results, the proposed algorithm exhibits promising performance for both function approximation and classification tasks. Kowalski and Łukasik [29] employ another bio-inspired metaheuristic to train an FNN for classification tasks. In the same year, Faris et al. [30] test the performance of a more recent bio-inspired algorithm in training MLPs. According to the reported results, the proposed algorithm is found to be promising. Göçken et al. [31] use a GA to select the best sub-set of attributes that can represent the desired output. The authors also use gradient-descent-based approaches to find the weights of the FNN.

Ghasemiyeh et al. [32] use metaheuristic algorithms to train an FNN to predict the stock prices. In order to train FNNs for classification problems, several other metaheuristic algorithms are adopted by Aljarah et al. [33]. Khatir et al. [34] employ a PSO to train an MLP that conducts damage assessment in composite laminates. In the same year, Tran-Ngoc et al. [35] report another study that introduces an ANN for damage detection in bridges and beam-like structures. The authors use a bio-inspired metaheuristic algorithm to train the proposed ANN. One year later, Rojas-Delgado et al. [36] propose a continuation method to reduce the computational complexity of MLPs. The authors report that the proposed method can reduce the required time to complete training in about 5–30 % without statistically significant loss of accuracy. Turkoglu and Kaya [3] compare the performances of several population based metaheuristic algorithms in training FNNs by using various classification data sets. Ansari et al. [37] apply a hybridized metaheuristic algorithm to train another FNN to be used for bankruptcy prediction. Tran-Ngoc et al. [38] employ a hybrid evolutionary based metaheuristic algorithm to train an MLP. According to the reported results, the proposed method is found to be promising.

Meng et al. [39] use an improved version of the same algorithm to train MLPs for both function approximation and classification tasks. Al Thobiani et al. [40] present a study that reports bio-inspired algorithms for defining parameters of ANNs. A recent study that is introduced by Ouladbrahim et al. [41] employ another bio-inspired metaheuristic to train ANNs for predicting the crack length based on different input values. In the same year, Ouladbrahim et al. [42] report an ANN to predict the parameters of the Gurson-Tvergaard-Needleman damage model coupled with the hardening law that goes through the prediction of traction and impact properties of API X70 steel pipe depending on chemical composition. Khatir et al. [43] address an enhanced damage indicator using a flexibility index and authors apply it to different complex structures to predict the exact location of damage. Benaissa et al. [44] propose a new metaheuristic algorithm with a search space reduction capability guided by a few simple formalism. Khatir et al. [45] introduce two-staged approaches to analyze damage detection, localization and quantification in Functionally Graded Material plate structures. Finally, a detailed a survey regarding the used metaheuristics in training ANNs is recently presented by Chong et al. [46].

As one can see from the related literature, various metaheuristic algorithms have been used by researchers. However, methods, bringing diverse features of such optimizers together by making use of reinforcement learning methods are still lacking. This constitutes the main motivation of the present study.

3. Used solution methods

3.1. Artificial Neural Networks

Inspired by human brains, ANNs are actually mathematical models, which are capable of performing tasks such as regression, image processing, classification, pattern recognition, prediction, robotics, and signal processing. An ANN is a network that is comprised of a set of processing units (neurons) aligned in layers and weights (synapsis) connecting those neurons each other. Early studies in ANNs show that they are capable of modeling only linear spaces. However, as theoretically demonstrated by later studies [5,47,48] an ANN with a hidden layer that is comprised of enough number of neurons has the capability of mapping between the input and target features in nonlinear and complex spaces.

One can easily generate an ANN, if the number of layers, the number of neurons in each layer and the values of the weights are given. The merit here is to find the optimum values for these parameters. Assuming that the number of layers and the number of neurons in each layer are defined, the remaining process, in which the weights are optimized is referred to as training stage. Training is one of the most important stages since it directly affects the generalization capability of ANNs. A successfully trained ANN is capable of approximating outputs for data patterns it has never seen before. Therefore, the information of an ANN is actually embedded to its weights.

An ANN with a flow in the forward direction is referred to as FNNs. In FNNs, signals are propagated only in the forward direction. Moreover, ANNs, with at least a single hidden layer are referred to as MLPs and they are one of the most commonly used ANNs due their capabilities summarized above. An illustration of an FNN with a single hidden layer is depicted by Fig. 1.

The three layers In Fig. 1 represent the input, hidden and output layers in sequence. The number of neurons in the input layer is denoted by n, the number of neurons in the hidden layer and output layer are represented by h and m, respectively. While the weights connecting the ith neuron in the input layer to the jth neuron in the hidden layer are denoted by w_{ij}^1, the weights connecting the jth neuron in the hidden layer to the kth neuron in the output layer are denoted by w_{jk}^2. Finally, I_i, β^1 and β^2 represent the input values, bias values for the hidden layer and bias values for the output layer, respectively. Accordingly, for an FNN, which uses logistic (sigmoid) activation function, output values can be evaluated based on the Eqs. 1–2, where outH_j and out$\hat{O}_k$ represent the output values of the neurons in hidden and output layers, respectively. Finally, mean-square-error (MSE), which is an error function to be minimized, is formulated by Eq. (3), where S, out$\hat{O}_k{}^p$ and out$O_k{}^p$ represent the number of samples, the predicted value of the kth neuron for the pth instance and the actual output value of the kth neuron for the pth instance in the output layer. The aim here is to minimize the error between the predicted and the actual values by finding the optimum values of weights and biases so as to increase the generalization capability of ANN.

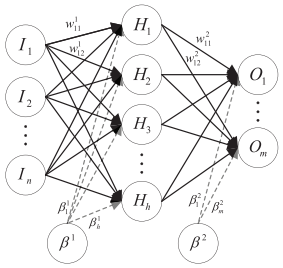


Fig. 1. An illustration of an FNN with a single hidden layer.

$$\text{out}H_j = \frac{1}{1+e^{-\left(\sum_{i=1}^{n} I_i \times w_{ij}^1 + \beta_i^1\right)}} \forall j \tag{1}$$

$$\text{out}\hat{O}_k = \frac{1}{1+e^{-\left(\sum_{j=1}^{h} outH_j \times w_{jk}^2 + \beta_k^2\right)}} \forall k \tag{2}$$

$$MSE = \frac{1}{S}\sum_{p=1}^{S}\sum_{k=1}^{m}\left(\text{out}\hat{O}_k{}^p - \text{out}O_k{}^p\right)^2 \tag{3}$$

3.2. Neural network encoding

A solution vector is comprised of two main parts. The first part is used to encode bias values, while next part is devoted to keep values of all existing weights between layers in sequence. A population is comprised of *popSize* solution vectors each with of length *nDim*, which is equal to the number of total biases and weights that can also be calculated by $n \times h + h \times m + h + m$, where n,h and m represent the number of neurons in the input, hidden and output layers, respectively. Solution vectors in population correspond to different FNNs. In other words, a population can encode *popSize* different FNNs and each of these FNNs are evolved by sharing information and some inheritance mechanisms throughout iterations.

3.3. Particle Swarm Optimization

PSO [7–8] is a swarm-based algorithm inspired by the behaviors of bird flocks. It searches the problem space by making use of the Eqs. 4–5, where current position (x_{ij}), particle best solution ($pbest$) and global best solution (X_{best}) affect the movement speed of particles (solution vectors). The parameters v_{ij}, w,c_1 and c_2 in these equations represent the speed of the ith particle at the jth dimension, the inertia coefficient and the accelerating coefficients, respectively. Although it can be used as a fixed parameter through-

out search, as proposed by Mirjalili et al. [25] inertia coefficient for the tth iteration w^t is calculated by using Eq. (6) so as to decrease the effect of inertia towards the end of search. In this equation, w^{max}, w^{min} and *maxIter* represent 3 user-supplied parameters denoting the maximum and the minimum values of w and the maximum number of iterations, respectively. A pseudo code for the canonical PSO is presented by Algorithm 1.

$$v_{ij} = w v_{ij} + c_1 \times rand \times \left(pbest_{ij} - x_{ij}\right) + c_2 \times rand \times \left(x_{best,j} - x_{ij}\right) \quad (4)$$

$$x_{ij} = x_{ij} + v_{ij} \quad (5)$$

$$w^t = w^{max} - t\left(w^{max} - w^{min}\right)/maxIter \quad (6)$$

Algorithm 1. A pseudo code for PSO.

```
initialize PSO
create initial particles, evaluate fitness values
while (termination criterion not met)
  for (i = 1 to popSize)
    for (j = 1 to dim)
      evaluate v_ij via Eq. (4)
      update x_ij via Eqs. 5–6
    end for
    evaluate fitness of the new particle
    update X_best and pbest if necessary
  end for
end while
print X_best
// popSize:population size
// dim: dimension of a particle
```

3.4. Differential Evolution Algorithm

DE [9] is another optimizer used in the present study. It evolves through generations based on the evaluated distances between individuals. DE adopts inheritance operators such as crossover, mutation and selection from evolutionary algorithms. Although, there exist various modifications of DE, DE/rand/1/bin, which is one of the most used extensions, is employed here.

DE starts with a random population that is generated by Eq. 7, where ${x_{ij}}^t$ represents the value of the jth dimension of the ith solution vector at iteration t. In this equation, $rand_j \in (0,1)$ is a random number for the jth dimension, ub_j and lb_j are the upper and lower bounds for the corresponding dimensions. Next, for each target vector ${X_i}^t$, a mutant vector ${M_i}^{t+1}$ is generated based on the Eq. (8), where r_1, r_2, r_3 are random integer numbers, representing the index of the randomly chosen individual and F is a scalar that is used for adjusting the effect of the difference vector. Subsequently, the mutant vector, which is formulated by Eq. (9), is crossed with the target vector in order to produce trial vector ${U_i}^{t+1}$, where $rand_j \in (0,1)$ is a random number again and $CR \in [0,1]$ is the crossover rate. Finally, fitness of the ith trial vector ${U_i}^{t+1}$ is evaluated. If it is better than the fitness of the target vector ${X_i}^t$, ${U_i}^{t+1}$ replaces ${X_i}^{t+1}$, otherwise ${X_i}^t$ is carried over to the next generation. A pseudo code for the canonical DE is presented by Algorithm 2.

$${x_{ij}}^1 = rand_j \times \left(ub_j - lb_j\right) + lb_j \; i \in [1, popSize], j \in [1, dim] \quad (7).$$

$${M_i}^{t+1} = {X_{r_3}}^t + F \times \left({X_{r_1}}^t - {X_{r_2}}^t\right) r_1, r_2, r_3 \in [1, popSize]; r_1 \neq r_2 \neq r_3 \neq i \quad (8)$$

$${u_{ij}}^{t+1} = \begin{cases} {m_{ij}}^{t+1} if \left(rand_j \leq CR\right) \\ {x_{ij}}^t otherwise \end{cases} \quad (9)$$

Algorithm 2. A pseudo code for DE.

```
initialize DE
create initial population, evaluate fitness values
while (termination criterion not met)
  for (i = 1 to popSize)
    create the mutant vector M_i^{t+1}
    apply xover to create trial vector U_i^{t+1}
    evalaute f(U_i^{t+1})
      if f(U_i^{t+1}) outperforms f(X_i^t)
        X_i^{t+1} ← U_i^{t+1}
        f(X_i^{t+1}) ← f(U_i^{t+1})
      else
        X_i^{t+1} ← X_i^t
        f(X_i^{t+1}) ← f(X_i^t)
      end if
    end for
    update X_best if necessary
  end while
print X_best
// popSize:population size
```

3.5. Flower Pollination Algorithm

As global and local pollination, FPA [10] has two main procedures. In global pollination procedure, new solutions are generated based on the global best position that is already denoted by X_{best}. This procedure adopts Lévy flights to mimic the movements of insects, flies, etc. Global pollination is formulated by Eq. (10), where x_i^t, X_{best} and L represent the ith solution vector at the tth iteration, global best solution found by the algorithm and Lévy distribution given by Eq. (11). In this equation $\Gamma(\lambda)$ represents the standard gamma function valid for large steps. As proposed by Yang [10], λ is fixed to 1.5 in all experiments also here. The second procedure, which is referred to as local pollination (abiotic pollination), is formulated by Eq. (12), where ϵ is a random vector $\in (0,1)$, x_j^t and x_k^t are two randomly chosen solution vectors at the tth iteration, respectively. FPA switches between movement procedures in regard to a user-provided parameter denoted by $p \in [0,1]$. A pseudo code for the canonical FPA is presented by Algorithm 3.

$$x_i^{t+1} = x_i^t + L\left(x_i^t - X_{best}\right) \quad (10)$$

$$L \sim \frac{\lambda \Gamma(\lambda) \sin(\pi\lambda/2)}{\pi s^{1+\lambda}} \left(s \gg s_0 > 0\right) \quad (11)$$

$$x_i^{t+1} = x_i^t + \epsilon\left(x_j^t - x_k^t\right) k \neq j \quad (12)$$

3.6. Hyper-heuristic based reinforcement learning

A hyper-heuristic is a reinforcement learning based high-level heuristic that employs a set of low-level heuristics [6]. Although they are usually reinforced by feedback based learning mechanisms, they can also exhibit random behavior in selecting low-level heuristics. While those low-level heuristics can perform some

simple trajectory-based moves such as swap, drop, insert, etc., they can also be chosen as some metaheuristics, which are already standalone stochastic search algorithms.

Algorithm 3. A pseudo code for the standard FPA.

```
initialize FPA
create initial solutions, evaluate fitness values
while (termination criterion not met)
  for (i = 1 to popSize)
    if rand < p
      generate a vector L from Lévy distribution (Eq. (11))
      apply global (biotic) pollination (Eq. (10))
    else
      generate the random vector ϵ
      apply local (abiotic) pollination (Eq. (12))
    end if
    evaluate fitness of the new solution
    update new solutions if necessary
  end for
  update the global best solution X_best if necessary
end while
print X_best
// popSize:population size
```

In the HH framework used in the present study, PSO, DE and FPA are employed as low-level heuristics. The feedback mechanism [49,50,51] that is shown to be capable of achieving promising results in previous publications is adopted here as the learning mechanism, which promotes the more successful algorithms while implicitly discouraging the remaining algorithms. In this method, which is formulated by the Eqs. 13–15, Δ_{HH} represent the learning rate of HH, η_k deonte the relative improvement achieved by kth low-level heuristic and δ_k is a variable to calculate ω_k, which represents the normalized selection probability of the kth low-level algorithm.

$$\eta_k = \left| \frac{f_{new} - f_{gbest}}{f_{gbest}} \right| \forall k = 1,2,3 \tag{13}$$

$$\delta_k = \omega_k + \Delta_{HH} \times \eta_k \forall k = 1,2,3 \tag{14}$$

$$\omega_k = \frac{\delta_k}{\sum_{t=1}^{3} \delta_t} \forall k = 1,2,3 \tag{15}$$

In the proposed approach, all algorithms work on the same population. For example, while PSO evolves the population at the tth iteration, any other algorithm can be selected to evolve the population at the iteration $t + 1$. Selection probability is calculated based on the Eq. (15), which normalizes the relative improvement performances, calculated by Eq. (13). Whenever an algorithm is to be invoked, a random number $rand \in (0,1)$ is generated and based on the ω_k values, a roulette-wheel procedure is conducted to define the winner, which is given the chance to evolve population. It is clear that greater relative improvements generate greater ω_k values, which increases the selection chance of the related algorithm.

As one can see from this procedure, also the naïve algorithms with lower ω_k values have the chance to be selected. This actually avoids the search from local optima and premature convergence issues, which are closely related to the learning speed, since; over-encouraging or over-discouraging the used algorithms might yield to poor results. The speed of learning can be tuned by using different Δ_{HH} values. While too small Δ_{HH} values can yield to too slow convergence, greater values might also yield to local optima problems. Finally, one should note that all algorithms are initialized with the same ω_k values. Brining all mechanisms together, a pseudo code for the proposed HH mechanism is presented by Algorithm 4.

Algorithm 4. A pseudo code for the proposed HH.

```
initialize PSO
initialize DE
initialize FPA
create initial solutions, evaluate fitness values
while (termination criterion not met)
  generate rand ∈ (0,1)
  if rand ≤ ω1 // invoke PSO
    k ← 1
    newPop ← PSO(pop)
  end if
  if rand > ω1 and rand ≤ ω1 + ω2 // invoke DE
    k ← 2
    newPop ← DE(pop)
  end if
  if rand > ω1 + ω2 and rand ≤ 1.00 // invoke FPA
    k ← 3
    newPop ← FPA(pop)
  end if
    evaluate η_k and δ_k
    update ω_k for the kth algorithm
    update the global best if necessary
end while
print X_best
// pop: current population
// newPop: new population
```

4. Experimental results

4.1. Fine tuning of algorithm parameters

Parameter calibration in the present study can be categorized into two categories such calibration of ANN hyper-parameters and calibration of the optimizer parameters. The FNNs used in the present study are assumed to have a single hidden layer. Therefore, appropriate value for the number of neurons in the hidden layer is attempted to be calibrated for different datasets individually. In order to find appropriate parameters, a comprehensive preliminary work is conducted by trying various values. In accordance, found promising values are presented by Table 1, which brings all optimizer related parameters together. These values are used throughout all experiments, which are carried out on a PC with i7-4700 HQ CPU and 16 GB of RAM.

Initial populations including all weights and biases for all algorithms are generated randomly between [-1,1]. Velocity values of PSO are randomly initialized as well. While one-hot-encoding technique is used to handle categorical features in classification problems, real-valued features are normalized to be between [0,1] with respect to Eq. (16), where x_d, x^{min},x^{max} and x'_d represent the dth data of the related feature, minimum and maximum values of the same feature and normalized value of the dth data, respectively.

$$x'_d = \frac{x_d - x^{min}}{x^{max} - x^{min}} \forall d \tag{16}$$

4.2. Function approximation

In the present study, the function $y = sin(2x)e^{-x}$ [25] is used as the approximation function, which is picturized by Fig. 2. Since it

Table 1
Optimizer parameters.

	parameters										
popSize		*maxIter*	c_1	c_2	w^{max}	w^{min}	*F*	*CR*	*p*	λ	Δ_{HH}
PSO	100	1000	1.80	1.80	0.10	0.01	*Na*	*Na*	*Na*	*Na*	*Na*
DE	100	1000	*Na*	*Na*	*Na*	*Na*	0.5	0.1	*Na*	*Na*	*Na*
FPA	100	1000	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	0.8	1.5	*Na*
HH	100	1000	1.80	1.80	0.10	0.01	0.50	0.10	0.8	1.5	0.1

has a single input and a single output, the trained FNN has a topology of 1-*h*-1, where *h* is the number of neurons in the hidden layer. As proposed by Mirjalili et al. [25], FNNs with *h* = 3, 4, 5, 6, and 7 are tested. A dataset in the interval $[0,\pi]$ with increments of 0.015 is used. Generated data is divided for training and testing by 80 % and 20 %, respectively. Therefore, while the total number of data is 210, the number of training data and testing data are 168 and 42, respectively. Both training and testing data is selected from whole dataset at random.

All obtained results for this benchmark are presented by Table 2. While the leftmost column in this table represents the number of neurons in the hidden layer, the rest of the columns show the average, median, standard deviance and best-found results of mean-squared-error (MSE) for both training and testing data sets. The CPU column represents the required average CPU time for the related optimizer. All reported results are evaluated over 10 independent replications.

As one can see from Table 2, as the number of neurons in the hidden layer increases, algorithms tend to obtain better results to some extent. Using *h* values greater than 5 does not make apparent contributions for some algorithms in terms of the best MSE values. In this regard, while *hh*FNN achieves the best results in terms of the best MSE for both training and testing data sets where *h* equals 3, 5 and 6, *pso*FNN acquires the same results for both training and testing in the remaining data sets where *h* equals 4 and 7. In terms of average MSE values, while *hh*FNN performs best for *h* = 5, *pso*FNN, *de*FNN and *fpa*FNN achieves the best results for *h* = 4 for training dataset. Similar results can be reported also for median MSE performance metric except for *de*FNN and *fpa*FNN, which perform best for *h* = 3 for training dataset again. Considering the standard deviances of MSE values, it can be put forward that *pso*FNN and *de*FNN perform better than *hh*FNN and *fpa*FNN in terms of achieving the minimum standard deviance values. As reported by the last column of Table 2, similar CPU times are required for each of the related optimizers.

The final reinforced probabilities of *hh*FNN are obtained as 0.5160, 0.2049 and 0.2791 for *pso*FNN, *fpa*FNN and *de*FNN, respectively. This also validates that the success of the *hh*FNN comes mostly from using a combination of *pso*FNN and *de*FNN, which is a parallel finding to the previous one. As an overall assessment, *hh*FNN and *pso*FNN seem to be exhibiting more promising performances in comparison to *de*FNN and *fpa*FNN. In the following subsection, a further statistical analysis is conducted to demonstrate whether significant differences exist.

Mean convergence plots of all optimizers for *h* = 5 are depicted by Fig. 3. Accordingly, convergence for training dataset, convergence for testing data set, convergence of *hh*FNN on training and testing datasets and convergence of the reinforced probabilities of *hh*FNN are presented by Fig. 3a, Fig. 3b, Fig. 3c and Fig. 3d, respectively. As one can see from Fig. 3a and Fig. 3b, *hh*FNN exhibits the most promising convergence for both training and testing data sets, while *fpa*FNN cannot be considered as promising as the other optimizers. This circumstance is also validated by Fig. 3.d, where each data point represents ω_k (Eq. (15)) for the related iteration. It is clear that sum of each data point should be equal to unity. According to Fig. 3d, *hh*FNN learns throughout iterations that invoking *fpa*FNN should be given the least chance, while *pso*FNN and *de*FNN should be promoted. Finally, a generalization that is performed by *hh*FNN is illustrated by Fig. 4, where blue squares represent testing data and the rest of the data points denote training data.

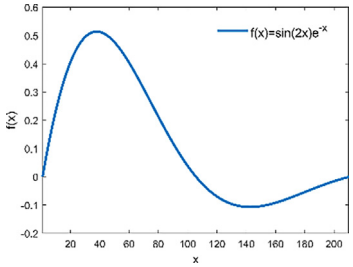


Fig. 2. An illustration for..$y = sin(2x)e^{-x}$

4.3. Classification problems

As mentioned before, classification datasets include both categorical and numeric data. While one-hot-encoding technique is used to handle categorical features, real-valued features are normalized with respect to Eq. (16). FNNs with single hidden layer are used for these problems as well. The number of neurons in the hidden layer is fixed to one less than twice the number of features for the related dataset. All used datasets are obtained from UCI machine learning repository[1] and all obtained results are presented by Table 3.

In this table, according to the results of the Iris dataset, all used algorithms achieve the same best MSE values in both training and testing. However, *pso*FNN and *de*FNN seem to be more promising in terms of average MSE, median MSE and std. dev. MSE in this dataset. In the Ionosphere dataset, while *hh*FNN achieves the best results for both training and testing in terms of the best MSE, surprisingly *fpa*FNN achieves the same best MSE with that of the *hh*FNN's. However, *pso*FNN and *hh*FNN seem to be exhibiting better performances again in comparison to *de*FNN and *fpa*FNN in terms of finding better average MSE, median MSE and std. dev. MSE in this dataset. According to the Breast dataset, while *hh*FNN achieves the best results at the training stage, *de*FNN has a better generalization capability with 100 percent accuracy. In accordance, while *fpa*FNN and *hh*FNN are more promising in training in terms of finding the best average MSE and median MSE, *de*FNN and *pso*FNN have better generalization capabilities for the same performance metrics. Finally, for the Australian dataset, *hh*FNN and *pso*FNN are found to be successful in finding the most promising best MSE values at the training and testing stages, respectively, while *de*FNN and *fpa*FNN perform better at the training stages in terms of finding the best average MSE values.

Mean convergence plots of all optimizers for classification datasets are illustrated by Fig. 5. As one can see from Fig. 5c, while *hh*FNN exhibits the most promising convergence at the training

[1] https://archive.ics.uci.edu/ml/index.php.

Table 2
Performance of the used algorithms on function approximation problem.

h	*algorithms*	*average MSE*		*median MSE*		*std. dev. MSE*		*best MSE*		*CPU*
		train	*test*	*train*	*test*	*train*	*test*	*train*	*test*	*(sec.)*
3	*pso*FNN	0.0021	0.0018	0.0021	0.0015	0.0008	0.0007	0.0011	0.0010	32.72
	*de*FNN	0.0016	0.0014	0.0011	0.0010	0.0006	0.0006	0.0011	0.0009	33.03
	*fpa*FNN	0.0040	0.0042	0.0028	0.0022	0.0036	0.0054	0.0022	0.0016	31.49
	*hh*FNN	0.0019	0.0021	0.0021	0.0024	0.0004	0.0005	0.0010	0.0008	34.52
4	*pso*FNN	0.0010	0.0015	0.0009	0.0015	0.0005	0.0006	0.0003	0.0004	32.15
	*de*FNN	0.0018	0.0018	0.0021	0.0022	0.0005	0.0007	0.0009	0.0007	31.84
	*fpa*FNN	0.0036	0.0049	0.0034	0.0047	0.0012	0.0016	0.0021	0.0031	35.07
	*hh*FNN	0.0024	0.0021	0.0019	0.0016	0.0015	0.0013	0.0009	0.0008	35.03
5	*pso*FNN	0.0017	0.0017	0.0016	0.0017	0.0006	0.0006	0.0009	0.0008	36.82
	*de*FNN	0.0015	0.0046	0.0015	0.0018	0.0007	0.0069	0.0007	0.0007	34.73
	*fpa*FNN	0.0041	0.0112	0.0030	0.0074	0.0025	0.0088	0.0020	0.0019	33.04
	*hh*FNN	0.0007	0.0006	0.0008	0.0007	0.0003	0.0003	0.0001	0.0001	36.42
6	*pso*FNN	0.0015	0.0014	0.0012	0.0009	0.0009	0.0012	0.0008	0.0006	37.19
	*de*FNN	0.0016	0.0050	0.0020	0.0026	0.0006	0.0064	0.0005	0.0007	36.05
	*fpa*FNN	0.0035	0.0112	0.0034	0.0112	0.0011	0.0089	0.0021	0.0019	36.92
	*hh*FNN	0.0032	0.0023	0.0013	0.0016	0.0054	0.0027	0.0003	0.0003	37.14
7	*pso*FNN	0.0013	0.0014	0.0011	0.0012	0.0007	0.0008	0.0002	0.0002	38.62
	*de*FNN	0.0017	0.0022	0.0022	0.0027	0.0005	0.0008	0.0008	0.0009	35.22
	*fpa*FNN	0.0036	0.0033	0.0036	0.0029	0.0012	0.0009	0.0022	0.0026	36.16
	*hh*FNN	0.0017	0.0021	0.0011	0.0014	0.0017	0.0018	0.0003	0.0006	37.19

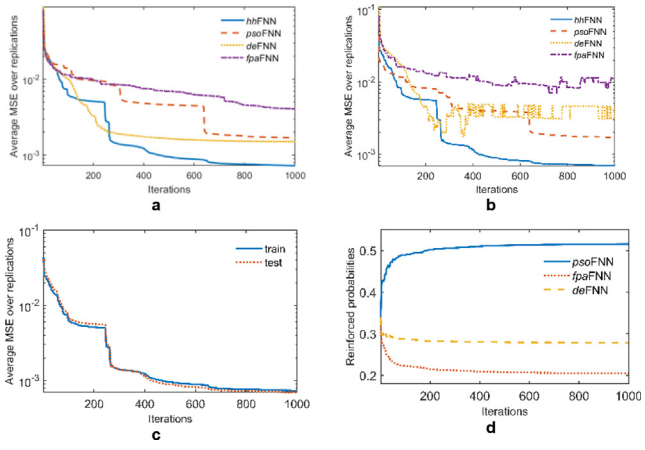


Fig. 3. a) Average MSE values for training dataset **b)** Average of MSE for testing dataset **c)** Average of MSE of training and testing datasets for *hh*FNN **d)** Reinforced probabilities for *hh*FNN over iterations.

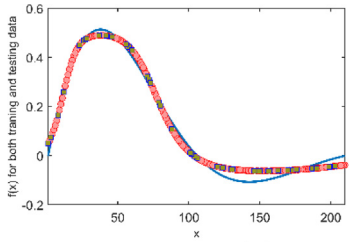


Fig. 4. An illustration for approximation of $y = sin(2x)e^{-x}$ for both training and testing data.

stage of Ionosphere dataset, Fig. 5a and Fig. 5d validate that *de*FNN exhibits the best convergence in Iris and Australian datasets, respectively. Finally, *hh*FNN and *fpa*FNN seem to have competitive convergence capabilities at the training stage of the Breast dataset.

Finally, reinforced probabilities of *hh*FNN are depicted by Fig. 6. This also validates that the success of *hh*FNN comes mostly from invoking *pso*FNN and *fpa*FNN more frequently in classification problems except for the Australian dataset, since; *fpa*FNN has the least chance to be invoked here. These findings validate the results reported by Table 3, however; a detailed statistical verification is required to define the promising optimizers. The next subsection is devoted to such analysis.

4.4. Statistical analysis

As reported by Derrac et al. [52], safe use of parametric tests are dependent to some particular conditions such as normality, independence and homoscedasticity, which are not fulfilled in the present study. Thus, non-parametric tests are adopted here. In this regard, Friedman Tests for both function approximation and classification problems are individually applied. The obtained Friedman ranks and *p*-values (adjusted for ties) for problems regarding the training and testing performances separately, are presented by Tables 4-5. Lower ranks point out better performing algorithms in these tables. Additionally, it should be noted that since Iris dataset includes only ties according to the best MSE values, it is excluded in this analysis. Finally, obtained *p*-values suggest the rejection of the null hypothesis that is based on the equality of medians for significance level $\alpha = 0.10$.

According to the results of Table 4, *pso*FNN and *hh*FNN are found as the most promising algorithms at the training stage of function approximation problem. While *de*FNN follows these two algorithms, *fpa*FNN follows *de*FNN. However, generalization capabilities of the FNNs trained by these optimizers differ for the training stage. As one can see from the testing column of Table 4, *pso*FNN, which is followed by *hh*FNN has the best performance. In the same column, while *de*FNN follows *hh*FNN, it is followed by *fpa*FNN.

Similarly, according to the results of Table 5, *hh*FNN is found as the most promising classifier at the training stage of classification problems. Surprisingly, *fpa*FNN is found as the second most promising algorithm, which is followed by *pso*FNN. Finally, *de*FNN is found as the naïve algorithm in this comparison. Generalization capability of *pso*FNN is found to be the most promising one, which is followed by *hh*FNN that is also found to be better than *de*FNN. Finally, *fpa*FNN seems to be exhibiting a naïve generalization performance in classification problems.

Initial findings and *p*-values of Friedman Tests (Tables 4-5) point out that there is at least one pair of algorithms, which significantly differ from each other. In order to find those significantly different pairs, $N \times N$ multiple comparisons, where N represents

Table 3
MSE performances of the used algorithms on classification problems.

datasets	*algorithms*	*average MSE*		*median MSE*		*std. dev. MSE*		*best MSE*		*CPU*
		train	*test*	*train*	*test*	*train*	*test*	*train*	*test*	*(sec.)*
Iris	*pso*FNN	0.0088	0.0133	0.0083	0.0222	0.0038	0.0114	**0.0055**	**0.0000**	425.89
	*de*FNN	0.0072	0.0266	0.0055	0.0222	0.0026	0.0229	**0.0055**	**0.0000**	438.63
	*fpa*FNN	0.0105	0.0155	0.0111	0.0111	0.0040	0.0182	**0.0055**	**0.0000**	432.09
	*hh*FNN	0.0183	0.0466	0.0138	0.0444	0.0161	0.0285	**0.0055**	**0.0000**	438.71
Ionosphere	*pso*FNN	0.0596	0.1000	0.0607	0.0985	0.0109	0.0193	0.0392	0.0563	543.82
	*de*FNN	0.1303	0.1676	0.1392	0.1690	0.0321	0.0462	0.0750	0.0985	542.56
	*fpa*FNN	0.0882	0.1169	0.0839	0.1126	0.0277	0.0518	0.0428	**0.0423**	557.29
	*hh*FNN	0.0367	0.1000	0.0357	0.0985	0.0103	0.0285	**0.0250**	**0.0423**	551.14
Breast	*pso*FNN	0.0222	0.0226	0.0215	0.0248	0.0019	0.0104	0.0197	0.0070	432.28
	*de*FNN	0.0283	0.0141	0.0286	0.0141	0.0020	0.0088	0.0250	**0.0000**	436.15
	*fpa*FNN	0.0168	0.0574	0.0161	0.0567	0.0031	0.0070	0.0143	0.0496	426.37
	*hh*FNN	0.0164	0.0255	0.0179	0.0248	0.0026	0.0068	**0.0107**	0.0141	430.21
Australian	*pso*FNN	0.1030	0.1258	0.1016	0.1223	0.0068	0.0170	0.0943	**0.1007**	642.25
	*de*FNN	0.0865	0.1446	0.0862	0.1438	0.0085	0.0128	0.0744	0.1366	627.61
	*fpa*FNN	0.0976	0.1985	0.0970	0.2014	0.0054	0.0209	0.0889	0.1726	615.34
	*hh*FNN	0.1052	0.1424	0.1070	0.1438	0.0171	0.0106	**0.0707**	0.1223	612.68

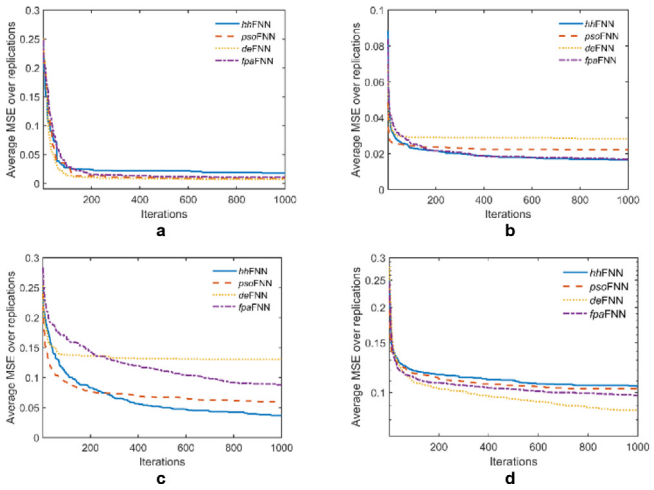


Fig. 5. Average MSE values for training datasets of **a)** Iris **b)** Breast **c)** Ionosphere **d)** Australian.

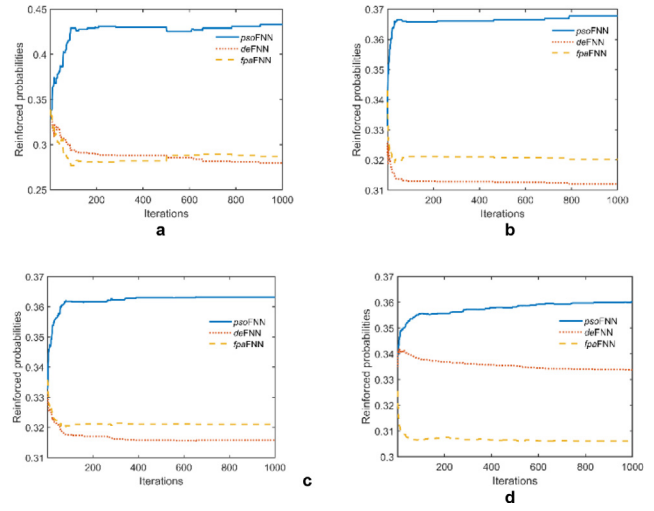


Fig. 6. Reinforced probabilities for *hh*FNN over iterations on datasets **a)** Iris **b)** Breast **c)** Ionosphere **d)** Australian.

Table 4
Friedman ranks for function approximation.

algorithms	*average ranks over replications* *training*	*testing*
*fpa*FNN	3.7800	3.6000
*de*FNN	2.1800	2.2000
*pso*FNN	2.0200	2.0200
*hh*FNN	2.0200	2.1800
p-values	0.0000	0.0000

Table 5
Friedman ranks for classification problems.

algorithms	*average ranks over replications* *training*	*testing*
*de*FNN	3.0667	2.6375
*pso*FNN	2.7667	1.9250
*fpa*FNN	2.2667	3.1250
*hh*FNN	1.9000	2.3125
p-values	0.0019	0.0001

the number of algorithms, are conducted [52]. One final issue to handle with is the Family-Wise-Error-Rate that emerges in multiple comparisons. In order to avoid this, a post-hoc procedure, namely, Bonferroni-Dunn procedure is performed to adjust the unadjusted *p*-values. Obtained results and decisions on null hypotheses for significance level α = 0.10 are presented by Tables 6-7.

According to the results presented by Table 6, at the training stage of function approximation, all optimizers are found to be significantly better than *fpa*FNN. However, significant differences do not exist between the rest of the algorithms, *hh*FNN, *pso*FNN and *de*FNN although *hh*FNN and *pso*FNN are found to be better than *de*FNN as presented in the training column of Table 4. Similar conclusion applies also for the generalization capabilities of these algorithms. As presented by the testing column of Table 6, all algorithms are found to be significantly better than *fpa*FNN. However, significant differences do not exist between the rest of the algorithms, *hh*FNN, *pso*FNN and *de*FNN again. These results validate the finding presented by Fig. 3.d, where the reinforced probabilities are illustrated. As one can see from this figure, *fpa*FNN has the least chance to be invoked while *pso*FNN and *de*FNN are more likely to be invoked by the hyper-heuristic learning mechanism.

According to the results presented by Table 7, *hh*FNN and *fpa*FNN seem to be exhibiting better performances at the training stage of classification problems. In this regard, while *hh*FNN is

Table 6
Multiple comparisons for function approximation.

training comparisons	*unadjusted p-values*	*adjusted p-values*	*decision on H_0*	*testing comparisons*	*unadjusted p-values*	*adjusted p-values*	*decision on H_0*
*hh*FNN *vs fpa*FNN	0.000000	0.000000	*Reject*	*pso*FNN *vs fpa*FNN	0.000000	0.000000	*Reject*
*pso*FNN *vs fpa*FNN	0.000000	0.000000	*Reject*	*hh*FNN *vs fpa*FNN	0.000000	0.000000	*Reject*
*de*FNN *vs fpa*FNN	0.000000	0.000000	*Reject*	*de*FNN *vs fpa*FNN	0.000000	0.000000	*Reject*
*hh*FNN *vs de*FNN	0.535470	1.000000	*Cannot Reject*	*pso*FNN *vs de*FNN	0.485717	1.000000	*Cannot Reject*
*pso*FNN *vs de*FNN	0.535470	1.000000	*Cannot Reject*	*pso*FNN *vs hh*FNN	0.535470	1.000000	*Cannot Reject*
*hh*FNN *vs pso*FNN	1.000000	1.000000	*Cannot Reject*	*hh*FNN *vs de*FNN	0.938258	1.000000	*Cannot Reject*

Table 7
Multiple comparisons for classification problem.

training comparisons	*unadjusted p-values*	*adjusted p-values*	*decision on H_0*	*testing comparisons*	*unadjusted p-values*	*adjusted p-values*	*decision on H_0*
*hh*FNN *vs de*FNN	0.000465	0.002790	*Reject*	*pso*FNN *vs fpa*FNN	0.000318	0.001908	*Reject*
*hh*FNN *vs pso*FNN	0.009320	0.055920	*Reject*	*hh*FNN *vs fpa*FNN	0.014789	0.088734	*Reject*
*fpa*FNN *vs de*FNN	0.016395	0.098370	*Reject*	*pso*FNN *vs de*FNN	0.032557	0.195342	*Cannot Reject*
*fpa*FNN *vs pso*FNN	0.133614	0.801684	*Cannot Reject*	*de*FNN *vs fpa*FNN	0.143604	0.861624	*Cannot Reject*
*hh*FNN *vs fpa*FNN	0.271289	1.000000	*Cannot Reject*	*pso*FNN *vs hh*FNN	0.245032	1.000000	*Cannot Reject*
*pso*FNN *vs de*FNN	0.368120	1.000000	*Cannot Reject*	*hh*FNN *vs de*FNN	0.32956	1.000000	*Cannot Reject*

found to be significantly better than *pso*FNN and *de*FNN, *fpa*FNN is significantly better than *de*FNN at the training stage. However, according to the generalization performances, *pso*FNN and *hh*FNN are found to be significantly better than *fpa*FNN while there is not enough statistical evidence to put forward further significant differences between the rest of the comparisons in the testing column of Table 7. All these results also validate the findings regarding the hyper-heuristic mechanism and reinforced probabilities in the classification problems presented by Fig. 6. As one can see from this figure, hyper-heuristic mechanism tends to invoke *pso*FNN more frequently, while *fpa*FNN and *de*FNN are less likely to be invoked.

A final analysis is carried out for the required CPU times of the used optimizers. According to the obtained *p*-values separately for function approximation and classification data sets, significant differences between the algorithms based on the equivalence of medians for the significance level $\alpha = 0.10$ do not exist.

5. Concluding remarks

The present study aims to develop an Artificial Neural Network (ANN) training algorithm based on a hyper-heuristic (HH) framework, which indeed resembles reinforcement learning-based machine learning algorithm. The proposed HH algorithm works according to a feedback mechanism that depends on the performances of the low-level heuristics taking place in the heuristics pool. In this context, it employs individual metaheuristic algorithms such as Particle Swarm Optimization (PSO), Differential Evolution (DE) Algorithm and Flower Pollination Algorithm (FPA) as low-level heuristics, which can also be employed as standalone optimizers. Based on the mentioned feedback mechanism, the proposed HH learns throughout epochs and encourages or discourages the related metaheuristic according to their performances. In this regard, advantages of deploying such a mechanism is twofold: due to its stochastic nature, HH is able to avoid local minima, and secondarily, by increasing the probability of invoking relatively more promising metaheuristics during training stage it has the capability of utilizing promising regions in search space more conveniently. Thus, regardless of its standalone performance, the mentioned machine learning based HH algorithm can also be employed so as to determine the more promising optimizers, as an algorithm recommendation procedure.

All developed algorithms are tested in both function approximation and classification problems, which have been adopted from UCI machine learning repository and existing literature. Comprehensive experimental studies for both of these datasets including appropriate statistical tests are conducted. According to the statistically verified results, which point out significant improvements, the developed HH based training algorithm can achieve significantly superior results to some of the compared optimizers. Moreover, as reported in the present study, it is shown to have the capability of learning and choosing the more promising optimizers throughout epochs. As one can see from the related literature, studies, bringing diverse features of different optimizers together are still lacking. Thus, the present study contributes to the relevant literature in this regard. Analyzing the performances of different optimizers and various machine learning algorithms such as Q-learning are scheduled as future works.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

[1] W.S. McCulloch, W. Pitts, A Logical Calculus of the Ideas Immanent in Nervous Activity, Bull. Math. Biophys. 5 (4) (1943) 115–133.
[2] E. Alpaydin, Introduction to machine learning, MIT Press, Cambridge, 2004.
[3] B. Turkoglu, E. Kaya, Training multi-layer perceptron with artificial algae algorithm, Eng. Sci. Technol. Int. J. 23 (6) (2020) 1342–1350.
[4] J. Li, J.H. Cheng, J.Y. Shi, Huang F, in: D. Jin, S. Lin (Eds.), Advances in Computer Science and Information Engineering, Berlin, Springer, Heidelberg, 2012, pp. 553–558.
[5] A.S. Rakitianskaia, A.P. Engelbrecht, Training feedforward neural networks with dynamic particle swarm optimisation, Swarm. Intell.-US 6 (3) (2012) 233–270.

[6] E.K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: A survey of the state of the art, J. Oper. Res. Soc. 64 (12) (2013) 1695–1724.
[7] R.C. Eberhart, J. Kennedy new optimizer using particle *swarm theory*. In Proceedings of the sixth IEEE International Symposium on Micro Machine and Human Science, Nagoya, Japan. Piscataway, October, 1995.
[8] Kennedy, J.R., Eberhart, *Particle swarm optimization*. In Proceedings of the IEEE International Conference on Neural Network, Perth, Australia, Novermber, 1995.
[9] R. Storn, K. Price, Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces, Technical Report TR-95-012, ICSI (1995).
[10] X.S. Yang, in: In Unconventional ComputIng and Natural Computation, Springer, Berlin, Heidelberg, 2012, pp. 240–249.
[11] R.S. Sexton, J.N. Gupta, Comparative evaluation of genetic algorithm and backpropagation for training neural networks, Inform. Sciences. 129 (1–4) (2000) 45–59.
[12] R.J. Kuo, C.H. Chen, Y.C. Hwang, An intelligent stock trading decision support system through integration of genetic algorithm based fuzzy neural network and artificial neural network, Fuzzy. Set. Syst. 118 (1) (2001) 21–45.
[13] W.Z. Lu, H.Y. Fan, S.M. Lo, Application of evolutionary neural network method in predicting pollutant levels in downtown area of Hong Kong, Neurocomputing. 51 (2003) 387–400.
[14] V.G. Gudise, G.K. Venayagamoorthy, Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks, In ProceedIngs of the IEEE Swarm Intelligence Symposium, 2023.
[15] P.C. Pendharkar, J.A. Rodger, An empirical study of impact of crossover operators on the performance of non-binary genetic algorithm based neural approaches for classification, Comput. Oper. Res. 31 (4) (2004) 481–498.
[16] C.F. Juang, A hybrid of genetic algorithm and particle swarm optimization for recurrent network design, IEEE. T. Syst. Man. Cyb. B. 34 (2) (2004) 997–1006.
[17] Z. Liu, A. Liu, C. Wang, Z. Niu, Evolving neural network using real coded genetic algorithm (GA) for multispectral image classification, Future. Gener. Comp. Sy. 20 (7) (2004) 1119–1129.
[18] Y. Da, G. Xiurun, An improved PSO-based ANN with simulated annealing technique, Neurocomputing. 63 (2005) 527–533.
[19] J. Yu, L. Xi, S. Wang, An improved particle swarm optimization for evolving feedforward artificial neural networks, Neural. Process. Lett. 26 (3) (2007) 217–231.
[20] J.R. Zhang, J. Zhang, T.M. Lok, M.R. Lyu, A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training, Appl. Math. Comput. 185 (2) (2007) 1026–1037.
[21] J. Yu, S. Wang, L. Xi, Evolving artificial neural networks using an improved PSO and DPSO, Neurocomputing. 71 (4–6) (2008) 1054–1060.
[22] S. Kiranyaz, T. Ince, A. Yildirim, M. Gabbouj, Evolutionary artificial neural networks by multi-dimensional particle swarm optimization, Neural. Networks. 22 (10) (2009) 1448–1462.
[23] K. Khan, A. Sahai, A comparison of BA, GA, PSO, BP and LM for training feed forward neural networks in e-learning context, Int. J. Intell. Sys. Appl. 4 (7) (2012) 23.
[24] A.A. Kawam, N. Mansour, Metaheuristic optimization algorithms for training artificial neural networks, Int. J. Comput. Inf. Technol 1 (2) (2012) 156–161.
[25] S. Mirjalili, S.Z.M. Hashim, H.M. Sardroudi, Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm, Appl. Math. Comput. 218 (22) (2012) 11125–11137.
[26] A. Askarzadeh, A. Rezazadeh, Artificial neural network training using a new efficient optimization algorithm, Appl. Soft. Comput. 13 (2) (2013) 1206–1213.
[27] A.P. Piotrowski, M. Osuch, M.J. Napiorkowski, P.M. Rowinski, J.J. Napiorkowski, Comparing large number of metaheuristics for artificial neural networks training to predict water temperature in a natural river, Comput. Geosci. 64 (2014) 136–151.
[28] S. Mirjalili, How effective is the Grey Wolf optimizer in training multi-layer perceptrons, Appl. Intell. 43 (1) (2015) 150–161.
[29] P.A. Kowalski, S. Łukasik, Training neural networks with krill herd algorithm, Neural. Process. Lett. 44 (1) (2016) 5–17.
[30] H. Faris, I. Aljarah, S. Mirjalili, Training feedforward neural networks using multi-verse optimizer for binary classification problems, Appl. Intell. 45 (2) (2016) 322–332.
[31] M. Göçken, M. Özçalıcı, A. Boru, A.T. Dosdoğru, Integrating metaheuristics and artificial neural networks for improved stock price prediction, Expert Syst. Appl. 44 (2016) 320–331.
[32] R. Ghasemiyeh, R. Moghdani, S.S. Sana, A hybrid artificial neural network with metaheuristic algorithms for predicting stock price, Cyb. Sys. 48 (4) (2017) 365–392.
[33] I. Aljarah, H. Faris, S. Mirjalili, Optimizing connection weights in neural networks using the whale optimization algorithm, Soft. Comput. 22 (1) (2018) 1–15.
[34] S. Khatir, S. Tiachacht, C.L. Thanh, T.Q. Bui, M.A. Wahab, Damage assessment in composite laminates using ANN-PSO-IGA and Cornwell indicator, Compos. Struct. 230 (2019) 111509.
[35] H. Tran-Ngoc, S. Khatir, G. De Roeck, T. Bui-Tien, M.A. Wahab, An efficient artificial neural network for damage detection in bridges and beam-like structures by improving training parameters using cuckoo search algorithm, Eng. Struct. 199 (2019) 109637.
[36] J. Rojas-Delgado, R. Trujillo-Rasúa, R. Bello, A continuation approach for training artificial neural networks with meta-heuristics, Pattern. Recog. Lett. 125 (2019) 373–380.
[37] A. Ansari, I.S. Ahmad, A.A. Bakar, M.R. Yaakub, A hybrid metaheuristic method in training artificial neural network for bankruptcy prediction, IEEE Access 8 (2020) 176640–176650.
[38] H. Tran-Ngoc, S. Khatir, H. Ho-Khac, G. De Roeck, T. Bui-Tien, M.A. Wahab, Efficient Artificial neural networks based on a hybrid metaheuristic optimization algorithm for damage detection in laminated structures, Compos. Struct. 262 (2021) 113339.
[39] X. Meng, J. Jiang, H. Wang, AGWO: *Advanced GWO in multi-layer perception optimization*, Expert. Syst. Appl. 173 (2021) 114676.
[40] F. Al Thobiani, S. Khatir, B. Benaissa, E. Ghandourah, S. Mirjalili, M.A. Wahab, A hybrid PSO and Grey Wolf Optimization algorithm for static and dynamic crack identification, Theor. Appl. Fract. Mec. 118 (2022) 103213.
[41] A. Ouladbrahim, I. Belaidi, S. Khatir, E. Magagnini, R. Capozucca, M.A. Wahab, Experimental crack identification of API X70 steel pipeline using improved Artificial Neural Networks based on Whale Optimization Algorithm, Mech. Mater. 166 (2022) 104200.
[42] A. Ouladbrahim, I. Belaidi, S. Khatir, E. Magagnini, R. Capozucca, M.A. Wahab, Prediction of Gurson damage model parameters coupled with hardening law identification of Steel X70 pipeline using neural network, Met. Mater. Int. 28 (2) (2022) 370–384.
[43] S. Khatir, S. Tiachacht, C. Le Thanh, H. Tran-Ngoc, S. Mirjalili, M.A. Wahab, A new robust flexibility index for structural damage identification and quantification, Eng. Fail. Anal. 129 (2021) 105714.
[44] B. Benaissa, N.A. Hocine, S. Khatir, M.K. Riahi, S. Mirjalili, YUKI algorithm and POD-RBF for elastostatic and dynamic crack identification, J. Comput. Sci-Neth. 55 (2021) 101451.
[45] S. Khatir, S. Tiachacht, C. Le Thanh, E. Ghandourah, S. Mirjalili, M.A. Wahab, An improved artificial neural network using arithmetic optimization algorithm for damage assessment in FGM composite plates, Compos. Struct. 273 (2021) 114287.
[46] H.Y. Chong, H.J. Yap, S.C. Tan, K.S. Yap, S.Y. Wong, Advances of metaheuristic algorithms in training neural networks for industrial applications, Soft. Comput. 25 (16) (2021) 11209–11233.
[47] S. Lawrence, A.C. Tsoi, A.D. *Function approximation with neural networks and local methods: bias, variance and smoothness*. In Proceedings of the Australian conference on neural networks (pp. 16-21). Canberra: Australian National University, 1996.
[48] M. Jinli, S. Zhiyi Application of combined neural networks in nonlinear function approximation. In Proceedings of the 3rd world congress on intelligent control and automation, Piscataway: IEEE, 2000.
[49] H. Wang, D. Wang, S. Yang, A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems, Soft. Comput. 13 (8) (2009) 763–780.
[50] A. Baykasoğlu, F.B. Ozsoydan, Evolutionary and population-based methods versus constructive search strategies in dynamic combinatorial optimization, Inform. Sciences. 420 (2017) 159–183.
[51] F.B. Ozsoydan, Artificial search agents with cognitive intelligence for binary optimization problems, Comput. Ind. Eng. 136 (2019) 18–30.
[52] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm. Evolut. Comput. 1 (1) (2011) 3–18.