Metaheuristic Optimization Algorithms for Training Artificial Neural Networks

Ahmad AL Kawam, Nashat Mansour
Computer Science Department, Lebanese American University
Beirut, Lebanon
Emails: ahmad.alkawam@lau.edu, nmansour@lau.edu.lb

Abstract—Training neural networks is a complex task that is important for supervised learning. A few metaheuristic optimization techniques have been applied to increase the effectiveness of the training process. The Cuckoo Search (CS) algorithm is a recently developed meta-heuristic optimization algorithm which is suitable for solving optimization problems. In this paper, Cuckoo search is implemented in training a feed forward multilayer Perceptron network (MLP). We then evaluate the trained MLP‟s accuracy by applying four benchmark classification problems. Furthermore, the results obtained are compared to those attained using another competing meta-heuristic which is the Particle Swarm Optimization (PSO). Also, Guaranteed Convergence Particle Swarm Optimization (GCPSO) which is a PSO variant is implemented and its results are compared with CS and PSO. CS proved to be superior to PSO and GCPSO in all benchmark problems.

Metaheuristic Algorithms; ANN Training; MLP; Cuckoo Search; Particle Swarm Optimization.

I. INTRODUCTION

Multi-layer Perceptron Networks (MLP) are feed-forward artificial neural networks which is a famous model for machine learning. MLPs are widely used as classifiers [1] and this type of neural networks needs to go through a training phase before using it to classify test data. According to [2], the process of training an ANN is to adjust the weights of the interconnections. Training is usually done using Back-Propagation (BP) algorithm but research showed that this algorithm has some disadvantages including slow convergence and getting trapped in local minima [3]. Many attempts have been made to enhance the performance of BP. Other approaches adapted the use of meta-heuristic algorithms to replace BP in the training phase [4].

The literature provides many examples where metaheuristics have been used in optimizing different machine learning models such as ANNs, Bayesian Networks, Markov Networks, and others. These metaheuristics include: Stochastic Local Search[5], Simulated annealing (SA) [6], Tabu Search [7], Evolutionary algorithms [8], and Swarm Intelligence. In [9] the authors apply Iterated Local Search for exploring the structure space for Markov Logic Networks. Whereas in [10] the authors apply a Stochastic Local Search to compute Most Probable Explanations for a Bayesian Network. As for ANNs, metaheuristics are usually applied in the training phase of the MLP: In [3] the authors apply Differential Evolution (DE) making use of its low computational complexity to explore the search space for a set of weights for a non-linear test application. The authors of [11] apply a Tabu Search algorithm which keeps a memory of previously visited weights in search of the best weight set. In [12] different variants of Particle Swarm Optimization (PSO), which is based on the swarm intelligence of fish and birds, are applied and compared. In [13] DE and PSO are compared for training an MLP and PSO was found to perform better. In [14] Genetic algorithms (GA) and SA are also compared for training an MLP and GA was found to perform better. In [15] another swarm intelligence meta-heuristic called Ant Colony Optimization (ACO) is applied for the same purpose of training neural networks.

Cuckoo Search (CS) is a recent metaheuristic algorithm developed by Yang and Deb in 2009 [16]. It is a type of swarm intelligence that is based on the brooding behavior of some kinds of cuckoo birds. CS is used in solving complex optimization problems, and according to [16] the optimal solutions obtained by CS are far better than the best solutions obtained by other swarm intelligence algorithms.

In this paper, three fairly recent metaheuristics are designed, implemented, and experimentally compared for the problem of training the MLP. The benchmark tests are four datasets that are taken from the UCI Machine Learning Repository. CS is compared with two versions of PSO which are the standard PSO and the guaranteed convergence PSO (GCPSO). PSO has been found to give good results and outperform other meta-heuristics such as DE on many optimization problems [4]. Also, according to [16], Yang and Deb proved that CS outperforms PSO in some optimization problems. In this paper, these algorithms are developed for the MLP training with the objective of minimizing the quadratic error. GCPSO is a variant of PSO and is applied because it is found to give better results than standard PSO in most optimization problems [12].

II. MULTILAYER PERCEPTRON

One of the most used ANN models is the Multi-Layer Perceptron (MLP) [17]. In every MLP network there is one input layer, one output layer and, one or more hidden layers. All nodes, except those of the input layer, are composed of neurons. The output of each layer is connected to the input of the next layer. The number of nodes in each layer varies depending on the problem at hand. The larger the number of hidden layers and number of nodes, the more complex the architecture will be.
Training an MLP is to find a set of weights that would give desired values at the ANN‟s output when presented with different patterns at its input. Figure 1 shows an example of an MLP.

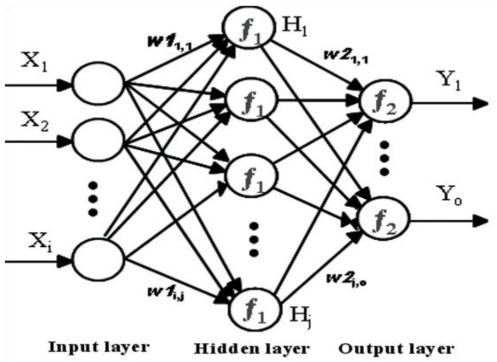

Figure 1 Example MLP Network

III. PARTICLE SWARM OPTIMIZATION (PSO)

PSO metaheuristic algorithm was introduced in [18] by Kennedy and Eberhart as a population based stochastic optimization algorithm for an n-dimensional problem space. It aims to minimize (or maximize) the objective function of the problem. The PSO was inspired by the flight of a flock of some birds when they search for resources [19].

The algorithm starts by creating a swarm of n particles. A particle of the swarm is represented by a position x(t) and a velocity v(t). Every position represents a solution and the goal of the algorithm is to move the particles to better positions. This is done using the particle‟s velocity which depends on its previous velocity, the best position found by the particle, and the best position found by the whole swarm.

Initially, all particles have their positions, x(0), randomly generated and their initial velocity, v(0) set to zero.

Then, the velocity of each particle is updated depending on its current velocity, its own best position (*individual best),* and the best position of the whole swarm (*global best)* according to the formula:

$$v_{ij}(t+1) = \omega v_{ij}(t) + c_1 r_1 \left(y_{ij}(t) - x_{ij}(t)\right) + c_2 r_2 \left(\hat{y}_j(t) - x_{ij}(t)\right) \qquad (1)$$

$$1 \le i \le s, 1 \le j \le n.$$

And then the particle‟s next position is updated by:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \qquad (2)$$

$$1 \le i \le s, 1 \le j \le n.$$

$y_{ij}(t)$ is the individual best position and $\hat{y}_{ij}(t)$ is the global best position.

w, c1, r1, c2, r2 are parameters of the algorithm. *n* is the dimension of the optimization problem.

The pseudo code of PSO can be summarized as follows:

```
Randomly initialize population of particles
repeat
  for each particle i of the population do
      if $f(x_{i(t)}) < f(y_{i(t)})$ then
          $y_i(t) = x_i(t)$
      end if
      If $f(y_{i(t)}) < f(\hat{y}(t))$ then
          $\hat{y}(t) = y_{i(t)}$
      end if
  end for
  Update velocity and position of each particle
according to eq. (1) and (2).
until stop criteria being satisfied
```

IV. Guaranteed Convergence PSO (GCPSO):

The PSO has a tendency to converge prematurely when $xi = yi = \hat{y}i$. In this case this particle will push the other particles to move towards it. GCPSO solves this problem by adding a term *r(t)* which is a random variable from U(0,1).

$$v_{ij}(t+1) = -x_{ij}(t) + \hat{y}_j(t) + \omega v_{ij}(t) + \rho(t)\big(1 - 2r(t)\big) \quad (3)$$

Where ρ(t) is an adaptive scaling factor given by:

$$\rho(t+1) = \begin{cases} 2\rho(t) & if\#success > Sc \\ 0.5\rho(t) & if\#failures > fc \\ \rho(t) & otherwise \end{cases} \quad (4)$$

In which #successes and #failures denote the number of consecutive successes and failures of the search in minimizing the objective function, and sc and fc are threshold parameters with initial values generally 5.

V. CUCKOO SEARCH OPTIMIZATION

Cuckoo Search (CS) [16] is a recent metaheuristic algorithm proposed in 2009 by Yang and Deb. The algorithm is inspired by the behavior of some species of cuckoo birds combined with the flight routines of many species of birds and flies, called levy flights.

According to [20], the cuckoo birds have a peculiar and aggressive brooding strategy. Certain Cuckoos lay their eggs in the nests of birds of other species. Cuckoos may also remove some of the other eggs in the nest to increase the hatching probability of their own eggs. The cuckoo eggs usually resemble the host eggs in order to reduce the risk of being discovered by the host birds. In case the host birds discover the cuckoo eggs, they either throw away the eggs or completely abandon the nest and build a new nest somewhere else.

In the implementation of the CS algorithm Yang and Deb(2009) made the following assumptions:

The solutions are represented as eggs where each egg is a solution and a cuckoo egg is a new solution. The cuckoo egg should be an improved solution and aims to replace a worse solution in the nest. In this implementation each nest constitutes of one egg.

The CS is implemented with these three rules:

• Each cuckoo lays one egg at a time, and dumps it in a randomly chosen nest;

• The best nests with the best of eggs (solutions) will move on to the next generations;

• The number of available host nests is fixed, and the cuckoo egg may be discovered with a probability pa from [0,1] . If discovered the nest is replaced by a new nest with random solutions.

The pseudo code is presented as:

```
Generate an initial population of $n$ host nests,
each nest containing one random solution;
**While** (t<MaxGeneration)
  Get a nest randomly (say, i) from the current
  best nests (top 1 - $p_a$nests) and improve its fitness
  by performing Lévy flights;
  Evaluate its fitness $F_i$
      [For maximization $F_i \propto f(x_i)$ ];
  Choose a nest among n (say, j) randomly;
  **if** ($F_i > F_j$),
      Replace j by the new solution;
  **end if**
  A fraction ($p_a$) of the worse nests are replaced
  by new random solutions;
  Keep the best nests;
  Rank the nests and find the current best;
  Pass the current best solutions to the next
  generation;
**end while**
Return the best nest.
```

When generating new solutions at (t +1) for the ith cuckoo, the following Lévy flight is performed

$$x_{ij}(t+1) = x_{ij}(t) + \alpha * \text{Lévy}(\lambda) \quad (5)$$

where α >0 is the step size, which depends on the scale of the problem. The product $\oplus$ is an entry-wise multiplication. A Lévy flight step size is represented by:

$$\text{Lévy u} = \text{t}^{-\lambda}, 1 \le \lambda \le 3 \quad (6)$$

The Lévy step size is a probability distribution with an infinite variance and the flight steps form a random walk process which obeys a power-law step-length distribution with a heavy tail.

According to [21], there are a few ways of achieving random numbers in levy flights, but one of the most efficient and yet straightforward ways is to use the so-called Mantegna algorithm for a symmetric Levy stable distribution, where „symmetric‟ means that the steps can be positive and negative.

In Mantegna‟s algorithm, the step length s can be calculated by:

$$s = \frac{u}{|v|^{1/\beta}} \qquad (7)$$

Such that:

$$u \sim N(0, \sigma_u^2), \qquad v \sim N(0, \sigma_v^2) \qquad (8)$$

Where:

$$\sigma_u = \left\{ \frac{\Gamma(1+\beta)\sin(\frac{\pi\beta}{2})}{\Gamma[(1+\beta)/2]\beta 2^{(\beta-1)/2}} \right\}^{1/\beta}, \quad \sigma_v = 1. \qquad (9)$$

Here Γ (z) is the gamma function:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt. \qquad (10)$$

VI. SIMULATION RESULTS, DISCUSSION AND ANALYSIS

A. *Dataset description*

Four well-known dataset instances were chosen from the *UCI Machine Learning Repository* to be applied to the training algorithms:

Glass: The Glass dataset is a study to classify glass into 7 types for police-work purposes. This dataset has a total of 214 instances divided into 7 classes where each instance has 10 attributes.

Ionosphere: The ionosphere dataset is a system that consisting of 16 high-frequency antennas placed as a phased array in Groose Bay, Labrador. Each instance in the database constitutes of 34 attributes that correspond to the complex values that represent the electromagnetic signal measured. There is a total of 351 instances classified to two classes, *good* or *bad.* This classification indicates whether there is evidence of some structure in the ionosphere or not.

Irish Plant: The Irish Plant dataset is a study to classify the Irish Plant into its three types: Iris Setosa, Iris Versicolour and Iris Virginica. The study depends on four attributes representing length, sepal width, petal length and petal width. The dataset has a total of 150 instances where each class has 50 instances.

Wine: The wine dataset is a study to classify wine into its three types depending on the values of 13 constituents. There is a total of 178 instances divided into 3 classes and with 13 attributes for each instance.

B. *Solution representation*

Because of the uniqueness of each dataset, each dataset had its own particular structure. For simplicity purposes only one hidden layer was used. Also, the number of inputs was equal to the number of attributes of each dataset. The number of outputs was equal to the number of classes. And the number of hidden nodes was equal to double the number of the inputs minus one. All hidden and output neurons have the sigmoid function as their activation function.

In the implementation of all algorithms each potential solution was a complete set of weights represented in an array of real numbers as shown in Figure 2:

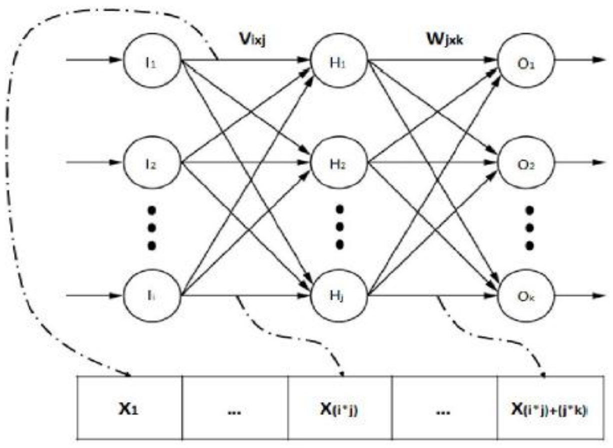

Figure 2. Solution Encoding

C. *Fitness function evaluation*

Each fitness evaluation was a forward pass through the network using the weights that are being evaluated. After the forward pass, the difference between the expected outputs and the actual outputs is calculated and then the quadratic error [12] is calculated as:

$$Quadratic\ Error = \sum_{p=1}^{M} \sum_{k=1}^{m} \left(t_k^{(p)} - y_k^{(p)}\right)^2$$

Where: M = number of instances, m = number of output neurons, t = target output, y = current output.

The fitness function is the inverse of the quadratic error. Our goal is minimize the quadratic error.

D. *Meta-heuristics parameters*

For all meta-heuristics the fitness function was the inverse of the quadratic error. Our objective is to minimize the difference between the current output and the desired output and thus maximize the fitness function.

Cuckoo search setup: 20 nests were used. The step size α = 0.01, β = 0.5 and the nest discovery parameter Pa = 0.1.

PSO setup: PSO used 20 particles, c1 = c2 = 2.05, and w decreased linearly from 0.9 to 0.4.

GCPSO setup: GCPSO used 20 particles, w= 0.4, and ρ_i=1.

E. Results

Each of the CS, PSO, and GCPSO algorithms was implemented in the training of an MLP of the same structure for the same benchmark dataset. The four datasets are: Iris, Glass, Wine, and Ionosphere. The results recorded on every run were the Quadratic Error for each dataset. After 20 runs on each dataset the results were recorded in Table 1 in the following format: average quadratic error (standard deviation).

TABLE I. RESULTS

Dataset	**Average Quadratic Error (&standard deviation)**		
	CS	**PSO**	**GCPSO**
Iris	**29.2 (14.1)**	62.2 (21.5)	44.7 (16.6)
Glass	**77.2 (22.5)**	164.8 (43.6)	156.5 (63.8)
Wine	**93.6 (16.7)**	105.6 (17.6)	115.2 (17.8)
Ionosphere	**32.6 (6.2)**	70.8 (18.9)	62.7 (17.7)

The results show that CS outperforms both PSO and GCPSO. On all datasets, CS yields lower average quadratic error and lower standard deviation. We can attribute this superiority of CS to two main reasons. The first reason is that CS has a good balance between intensification found in local search and diversification found in the exploration of the whole search space. The second reason is that CS has less parameters than PSO and GCPSO making it easier to tune. Also [16] shows that CS is not sensitive to its second parameter p_a which also decreases the algorithm"s dependency on its parameters.

VII. CONCLUSION

Training algorithms play an important role in enhancing the quality of ANNs. The usual algorithm for training ANNs is BP which has two problems: slow convergence and premature convergence. Various attempts have been made to improve the quality of BP. Metaheuristic algorithms are known for their ability to produce optimal or near optimal solutions for optimization problems. In the recent years meta-heuristic algorithms were used to train ANN and yielded good results.

In this research a recent meta-heuristic, Cuckoo Search, was designed for the training of a feed forward MLP and compared with PSO and GCPSO on four benchmark problems. On the four test problems the CS proved to be superior to PSO and GCPSO with a remarkable advantage. Further research will explore the development of a parallel implementation of the cuckoo search algorithm to apply it to the training of large neural networks.

REFERENCES

[1] J. Heaton. “Introduction to Neural Networks for Java, Second Edition”. Heaton Research, Inc., 2008.
[2] E. Valian, S. Mohanna and S. Tavakoli. “Improved Cuckoo Search Algorithm for Feed-Forward Neural Network Training”. International Journal of Artificial Intelligence & Applications (IJAIA), Vol.2, No.3, July 2011
[3] L. Mingguang and L. Gaoyang. “Artificial Neural Network Co-optimization Algorithm based on Differential Evolution”. Second International Symposium on Computational Intelligence and Design. 2009.
[4] A. Espinal, M. Sotelo-Figueroa, J. A. Soria-Alcaraz, M. Ornelas, H. Puga, M. Carpio, R. Baltazar, J.L. Rico. “Comparison of PSO and DE for training neural networks” 10th Mexican International Conference on Artificial Intelligence. 2011.
[5] H. H. Hoos & T. Stützle, “Stochastic Local Search Foundations and Applications”. Morgan Kaufmann / Elsevier, 2004.
[6] S. Kirkpartik, C.D. Dellat Jr. and M.P. Vecchi, "Optimization by simulated annealing", Science, 220: 671-680, 1983.

[7] F. Glover, "Future paths for integer programming and links to artificial intelligence", Computers and Operation Research, Vol. 13, pp. 533-549, 1986.

[8] E. Eiben; J. E. Smith. Introduction to Evolutionary Computing. Natural Computing Series. MIT Press. Springer. Berlin. (2003).

[9] M. Biba, S. Ferilli, F. Esposito, "Structure Learning of Markov Logic Networks through Iterated Local Search". ECAI 2008: 361-365.

[10] Mengshoel, O.J. "Initialization and Restart in Stochastic Local Search: Computing a Most Probable Explanation in Bayesian Networks". IEEE Transactions on Knowledge and Data Engineering, 2011.

[11] B. Dengiz, C. Alabas-Uslu, and O. Dengiz . "a tabu search algorithm for the training of neural networks". Journal of the Operational Research Society 60, 282 --291 2009

[12] M Carvalho, T. B. Ludermir. "An Analysis of PSO Hybrid Algorithms For Feed-Forward Neural Networks Training. Proceedings of the Ninth Brazilian Symposium on Neural Networks (SBRN'06).

[13] A. Espinal, M. Sotelo-Figueroa, J. A. Soria-Alcaraz, M. Ornelas, H. Puga, M Carpio, R Baltazar, J.L. Rico. "Comparison of PSO and DE for training neural networks". 10th Mexican International Conference on Artificial Intelligence 2011

[14] R.S. Sexton, R.E. Dorsey and J.D. Johnson, "Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing", European Journal of Operational Research (114)pp.589-601,1999.

[15] C. Blum and K. Socha. "Training feed-forward neural networks with ant colony optimization: An application to pattern classification"", Fifth International Conference on Hybrid Intelligent Systems (HIS‟05), pp. 233-238, 2005.

[16] XS Yang, S. Deb. "Cuckoo search via Lévy flights". Proc. of World Congress on Nature & Biologically Inspired Computing. 2009. pp 210-214.

[17] D. E. Rumelhart, G. E. Hilton, R. J. Willians. "Learning representations of back-propagation erros". Nature(London), vol.323, pp. 523-536. Oct 1986

[18] Kennedy, J.; Eberhart, R. "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942–1948, 1995.

[19] C. Maurice, Particle Swarm Optimization. USA: Wiley-ISTE, 2006.

[20] RB Payne, MD Sorenson, K Klitz "The Cuckoos". Oxford University Press.2005.

[21] Yang XS, Deb S. "Engineering Optimisation by Cuckoo Search", Int. J. Mathematical Modelling and Numerical Optimisation. Vol. 1, No. 4, pp 330–343. 2010.