Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks

Dervis Karaboga, Bahriye Akay, and Celal Ozturk

Erciyes University, Engineering Faculty, Department of Computer Engineering
karaboga@erciyes.edu.tr,
bahriye@erciyes.edu.tr, celal@erciyes.edu.tr

Abstract. Training an artificial neural network is an optimization task since it is desired to find optimal weight set of a neural network in training process. Traditional training algorithms has some drawbacks such as getting stuck in local minima and computational complexity. Therefore, evolutionary algorithms are employed to train neural networks to overcome these issues. In this work, Artificial Bee Colony (ABC) Algorithm which has good exploration and exploitation capabilities in searching optimal weight set is used in training neural networks.

1 Introduction

Since Artificial Neural Networks (ANNs) are quite successful in modelling non-linearity and have characteristics such as being capable of generalizing, adaptability, self-organizing, real time operation and fault tolerance, they are involved in so many applications in research fields. Finding a suitable network structure and finding optimal weight values make design of ANNs difficult optimization problems. In other words, the success of ANNs largely depends on the architecture, the training algorithm, and the choice of features used in training.

Artificial neural network training has traditionally been carried out using the back-propagation (BP) gradient descent (GD) algorithm [1]. But this technique has some drawbacks such as dependence of error surface shape, initial values of connection weights, parameters. If the error surface is multimodal, the gradient descent based algorithms are trapped at local minima. Involving differentiation of error function is another issue with this kind of algorithms [2]. Saturation may occur if the output is pushed towards its extremes at some point before convergence is reached, so that the derivative is too small to make further significant weight changes, causing the network to settle in an incorrect local minimum or reach a state of network paralysis . This saturation may occur for a number of reasons, most of which are easily avoided [3].In order to overcome the disadvantages of gradient based algorithms, many global optimization methods have been proposed for training feed-forward neural networks such as Genetic Algorithms [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17], The Particle Swarm Optimization algorithm [18, 19, 20, 22, 21, 23], Differential Evolution [24, 25, 26, 27, 28, 29] and

Evolutionary Programming algorithms [30,31,32]. Also, some hybrid techniques combining traditional techniques such as back propagation and evolutionary algorithms are proposed for training neural networks [33]. Not all of this algorithms handle with only connection weights, they also optimize the structure of the network. However, when the neural network training becomes a large scale, the number of network parameters grows drastically. For example,learning a huge number of hidden layer weights in a multi-layer perceptron (MLP) neural network can be considered as a large scale optimization problem.

Karaboga has described Artificial Bee Colony (ABC) algorithm based on the foraging behaviour of honey bees for numerical optimization problems [34], and Karaboga and Basturk have compared the performance of the ABC algorithm with those of other well-known modern heuristic algorithms such as Genetic Algorithm (GA), Differential Evolution (DE), Particle Swarm Optimization (PSO) on unconstrained problems [35]. In this work, the ABC algorithm is employed in training feed-forward neural networks and the performance of the algorithm is compared with Genetic Algorithm (GA) from evolutionary algorithms and Back-Propagation (BP) Algorithm. The paper is organized as follows: In Section 2, training an artificial neural network is described. In Section 3, implementation of artificial neural network training by using the ABC algorithm is introduced. In Section 4, experiments and results of produced by ABC, GA, and BP are presented and discussed.

2 Training Feed Forward Artificial Neural Networks

An ANN consists of a set of processing elements (Fig. 1), also known as neurons or nodes, which are interconnected with each other [14]. Output of the ith neuron can be described by Eq. 1

$$y_i = f_i(\sum_{j=1}^{n} w_{ij} x_j + \theta_i) \tag{1}$$

where y_i is the output of the node, x_j is the jth input to the node, w_{ij} is the connection weight between the node and input x_j, θ_i is the threshold (or bias) of the node, and f_i is the node transfer function. Usually, the node transfer function is a nonlinear function such as a heaviside function, a sigmoid function, a Gaussian function, etc.

Generally, the adaptation can be carried out by minimizing (optimizing) the network error function E. The error function is given by Eq. 2:

$$E(\boldsymbol{w}(t)) = \frac{1}{n} \sum_{j=1}^{n} \sum_{k=1}^{K} (d_k - o_k)^2 \tag{2}$$

where, $E(\boldsymbol{w}(t))$ is the error at the tth iteration; $\boldsymbol{w}(t)$, the weights in the connections at the tth iteration; d_k, the desired output node; o_k, the actual value of the kth output node;K, the number of output nodes; n, the number of patterns.

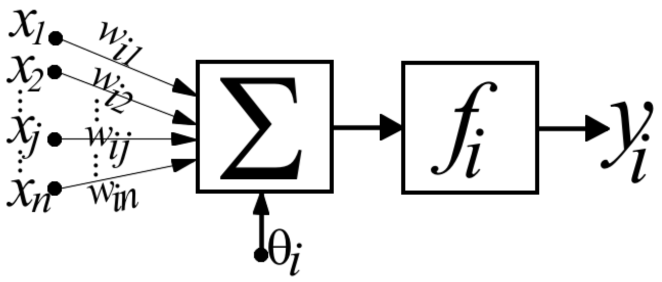

Fig. 1. Processing unit of an ANN (neuron)

The optimization goal is to minimize the objective function by optimizing the network weights $\boldsymbol{w}(t)$. In Evolutionary Algorithms, the major idea underlying this synthesis is to interpret the weight matrices of the ANNs as individuals, to change the weights by means of some operations such as crossover and mutation, and to use the error E produced by the ANNs as the fitness measure which guides selection. This leads to the following evolutionary training cycle [36]:

1. Formation of the next population of ANNs by means of operators such as crossover and mutation and fitness–oriented selection of the the weight matrices. (The initial population is randomly created.)
2. Evaluation of the fitness values of the ANNs.
3. If the desired result is obtained, then stop; otherwise goto step 1.

3 Artificial Bee Colony Algorithm

Artificial Bee Colony (ABC) algorithm was proposed by Karaboga for optimizing numerical problems in 2005 [34]. The algorithm simulates the intelligent foraging behaviour of honey bee swarms. It is a very simple, robust and population based stochastic optimization algorithm. Karaboga and Basturk have compared the performance of the ABC algorithm with those of other well-known modern heuristic algorithms such as Genetic Algorithm (GA), Differential Evolution (DE), Particle Swarm Optimization (PSO) on unconstrained problems [35].

Detailed pseudo-code of the ABC algorithm is given below:

```
1: Initialize the population of solutions $x_i, i = 1 \ldots SN$
2: Evaluate the population
3: cycle=1
4: repeat
5:    Produce new solutions $v_i$ for the employed bees by using (4) and evaluate
      them
6:    Apply the greedy selection process
7:    Calculate the probability values $p_i$ for the solutions $x_i$ by (3)
8:    Produce the new solutions $v_i$ for the onlookers from the solutions $x_i$ se-
      lected depending on $p_i$ and evaluate them
```

```
 9:     Apply the greedy selection process
10:     Determine the abandoned solution for the scout, if exists, and replace it
        with a new randomly produced solution $x_i$ by (5)
11:     Memorize the best solution achieved so far
12:     cycle=cycle+1
13: **until** cycle=MCN
```

In ABC algorithm, the position of a food source represents a possible solution to the optimization problem and the nectar amount of a food source corresponds to the quality (fitness) of the associated solution. The number of the employed bees or the onlooker bees is equal to the number of solutions in the population. At the first step, the ABC generates a randomly distributed initial population $P(G = 0)$ of SN solutions (food source positions), where SN denotes the size of population. Each solution x_i ($i = 1, 2, ..., SN$) is a D-dimensional vector. Here, D is the number of optimization parameters. After initialization, the population of the positions (solutions) is subjected to repeated cycles, $C = 1, 2, ..., MCN$, of the search processes of the employed bees, the onlooker bees and scout bees. An employed bee produces a modification on the position (solution) in her memory depending on the local information (visual information) and tests the nectar amount (fitness value) of the new source (new solution). Provided that the nectar amount of the new one is higher than that of the previous one, the bee memorizes the new position and forgets the old one. Otherwise she keeps the position of the previous one in her memory. After all employed bees complete the search process, they share the nectar information of the food sources and their position information with the onlooker bees on the dance area. An onlooker bee evaluates the nectar information taken from all employed bees and chooses a food source with a probability related to its nectar amount. As in the case of the employed bee, she produces a modification on the position in her memory and checks the nectar amount of the candidate source. Providing that its nectar is higher than that of the previous one, the bee memorizes the new position and forgets the old one.

An artificial onlooker bee chooses a food source depending on the probability value associated with that food source, p_i, calculated by the following expression (3):

$$p_i = \frac{fit_i}{\sum\limits_{n=1}^{SN} fit_n} \tag{3}$$

where fit_i is the fitness value of the solution i which is proportional to the nectar amount of the food source in the position i and SN is the number of food sources which is equal to the number of employed bees (BN).

In order to produce a candidate food position from the old one in memory, the ABC uses the following expression (4):

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \tag{4}$$

where $k \in \{1, 2,..., SN\}$ and $j \in \{1, 2,..., D\}$ are randomly chosen indexes. Although k is determined randomly, it has to be different from i. $\phi_{i,j}$ is a random number between [-1, 1]. It controls the production of neighbour food sources around $x_{i,j}$ and represents the comparison of two food positions visually by a bee. As can be seen from (4), as the difference between the parameters of the $x_{i,j}$ and $x_{k,j}$ decreases, the perturbation on the position $x_{i,j}$ gets decrease, too. Thus, as the search approaches to the optimum solution in the search space, the step length is adaptively reduced.

The food source of which the nectar is abandoned by the bees is replaced with a new food source by the scouts. In ABC, this is simulated by producing a position randomly and replacing it with the abandoned one. In ABC, providing that a position can not be improved further through a predetermined number of cycles, then that food source is assumed to be abandoned. The value of predetermined number of cycles is an important control parameter of the ABC algorithm, which is called "*limit*" for abandonment. Assume that the abandoned source is x_i and $j \in \{1, 2,..., D\}$, then the scout discovers a new food source to be replaced with x_i. This operation can be defined as in (5)

$$x_i^j = x_{\min}^j + \mathrm{rand}(0,1)(x_{\max}^j - x_{\min}^j) \tag{5}$$

After each candidate source position $v_{i,j}$ is produced and then evaluated by the artificial bee, its performance is compared with that of its old one. If the new food source has an equal or better nectar than the old source, it is replaced with the old one in the memory. Otherwise, the old one is retained in the memory. In other words, a greedy selection mechanism is employed as the selection operation between the old and the candidate one.There are three control parameters in the ABC: The number of food sources which is equal to the number of employed or onlooker bees (SN), the value of *limit*, the maximum cycle number (MCN).

In a robust search process, exploration and exploitation processes must be carried out together. In the ABC algorithm, while onlookers and employed bees carry out the exploitation process in the search space, the scouts control the exploration process.

4 Experimental Study

In this work, three problems are considered: XOR, 3-Bit Parity and 4-Bit Encoder-Decoder problems, which are benchmark problems used in training neural networks.

4.1 The Exclusive-OR Problem

The first test problem we used in the experiments is the exclusive-OR (XOR) Boolean function which is a difficult classification problem mapping two binary inputs to a single binary output as (0 0;0 1;1 0;1 1)→(0;1;1;0). In the simulations we use a 2-2-1 feed-forward neural network with six connection weights, no biasses (having six parameters, XOR6) and a 2-2-1 feed-forward neural network with six connection weights and three biases (having 9 parameters, XOR9) and a 2-3-1 feed-forward neural network having nine connection weights and four biases totally thirteen parameters (XOR13). For XOR6, XOR9 and XOR13 problems, the parameter ranges [-100,100], [-10,10] and [-10,10] are used, respectively.

4.2 3-Bit Parity Problem

The second test problem is the three bit parity problem. The problem is taking the modulus 2 of summation of three inputs. In other words, if the number of binary inputs is odd, the output is 1, otherwise it is 0. (0 0 0;0 0 1;0 1 0;0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1)→(0;1;1;0;1;0;0;1) We use a 3-3-1 feed-forward neural network structure for the 3-Bit Parity problem. It has twelve connection weights and four biasses, totally sixteen parameters. The parameter range was [-10,10] for this problem.

4.3 4-Bit Encoder/Decoder Problem

The third problem is 4-bit encoder/decoder problem. The network is presented with 4 distinct input patterns, each having only one bit turned on. The output is a duplication of the inputs. (0 0 0 1;0 0 1 0;0 1 0 0;1 0 0 0)→(0 0 0 1;0 0 1 0;0 1 0 0;1 0 0 0). This is quite close to real world pattern classification tasks, where small changes in the input pattern cause small changes in the output pattern [37]. A 4-2-4 feed-forward neural network structure is used for this problem and it has totally 22 parameters including sixteen connection weights and six biases. For this problem, the parameter range is [-10,10].

The parameter ranges, dimension of the problems, and the network structures are presented in Table 1.

Table 1. Parameters of the problems considered in the experiments. D: Dimension of the problem.

	Range	**NN Structure**	**D**
XOR6	[-100,100]	2-2-1 without Bias	6
XOR9	[-10,10]	2-2-1 with bias	9
XOR13	[-10,10]	2-3-1 with bias	13
3-Bit Parity	[-10,10]	3-3-1 with bias	16
4-Bit Enc.-Dec.	[-10,10]	4-2-4 with bias	22

4.4 Settings

Experiments were repeated 30 times for each case and each run was started with a random population with different seeds. In each network selected for each problem, sigmoid function is employed as transfer function. Training processes were stopped when the mean squared error of the outputs associated with inputs was equal to or less than 0.01 (MSE$\leq$0.01) or when the maximum generation or cycle or epoch has been reached. Since the difficulty of each problem is different, different parameter settings were used for each of them. Among all of the problems, XOR6 is the most difficult one since it does not employ biases. Therefore, all algorithms were run through more generations/cycles/epochs for this problem.

ABC Settings: The value of "*limit*" is equal to SN x D where D is the dimension of the problem. Colony size $(2 * SN)$ is 50 for all problems

GA Settings: In the experiments, roulette wheel selection scheme, single point crossover with the rate of 0.8, uniform mutation with the rate of 0.05 are employed. Generation gap is set to 0.9. The population size in GA were 50 for all problems.

BP Settings: In back-propagation experiments, NNs were trained by using Levenberg-Mardquart (LM) and Gradient Descent (GD) training algorithms. Learning rate for GD is 0.8.

These settings are summarized in Table 2.

Table 2. Values for control parameters of the algorithms. LR: Learning Rate, Pop: Population Size, CR: Crossover Rate, MR: Mutation Rate, GP: Generation Gap, SN: Colony Size.

BP	**GA**	**ABC**
LR: 0.8	Pop: 50 CR: 0.8 MR: 0.05 GP: 0.9	SN: 50 limit: SN*D

Maximum cycle number (MCN) for ABC and maximum generation number for GA were 7500,100,75,1000 1000 for XOR6, XOR9, XOR13, 3-Bit Parity and 4-Bit Encoder-Decoder problems, respectively. Hence, the total objective function evaluation numbers were 375 000, 5000, 3750, 50000 and 50000 for the problems, respectively. In case of BP algorithm, the number of epochs for problems were 32000, 500, 250, 1600 and 2100, respectively. These values are presented in Table 3.

Table 3. Maximum Cycle/Generation/Epoch numbers for ABC/GA/BP and the total Objective Function Evaluation (OFE) numbers for algorithms

	Cycle/Gen.	**OFE**	**Epoch**
XOR6	7500	375000	32000
XOR9	100	5000	500
XOR13	75	3750	250
3-Bit Parity	1000	50000	1600
4-Bit Enc.-Dec.	1000	50000	2100

4.5 Results and Discussion

Statistical results and the success rates of the algorithms for XOR6, XOR9, XOR13, 3-Bit Parity and 4-Bit Encoder-Decoder problems are given in Tables 4-6. For XOR6 problem, back propagation algorithm trained by GD has 3 % success and back propagation algorithm trained by LM has 6% success. Genetic algorithm has 0% success while the ABC algorithm has 100 % success. On XOR6 problem, GA could not find the global minima. Mean cycle number of ABC algorithm is 2717.4 for XOR6 problem. Mean cycle number of ABC is more than the mean epoch number of the BP (LM), but the LM algorithm has got stuck with the local minima of XOR6 problem. For XOR9 problem, BP (GD) has 0% success while BP (LM) has 66.66 % success with the mean of 13 epochs. GA has 40 % success with a mean of 77.067 generations. The ABC algorithm has 100 % success and the mean of cycles is 32. As seen from the mean OFE and success numbers, the BP (LM) algorithm has very fast convergence speed but it gets stuck to the local minima while ABC algorithm goes on searching without being stuck in local minima. For XOR13 problem, BP (GD) has 0% success with an average of 250 epochs while BP (LM) has 96.66% success with an average of 9 epochs. On XOR13 problem, GA has 76.66 % success rate and ABC has 100 % success with an average of 28.2 cycles. Although the problems were the same in the case of XOR6, XOR9 and XOR13, the network structures employed in the experiments were different. The NN used for XOR13 problem has 3 hidden neurons while the networks employed for XOR6 and XOR9 have 2 hidden neurons. More complex network structures do not always facilitate the problems or small size networks can not produce good results for the problems. It is known that finding adequate network structure is another design problem as well as finding optimum weights in training process. For 3-Bit parity problem, BP(GD) could not find the optima in any of runs. BP(LM) could find the optimum with a success rate of 86.66%. The success rate of GA was 63.33% on this problem. The ABC algorithm was able to find the desired network output in each run with an average cycle of 179.06. For Encoder-Decoder problem, BP(GD) has 2% success and BP(LM) has 73.33 %. GA has 86.66 % and the ABC algorithm has 100% success rate on this problem. Consequently, ABC outperforms other algorithms on all problems considered in this work for the same evaluation number and can consistently find the optimum weight set for the networks. For all problems, the success rate of ABC algorithm is 100 %. An algorithm that only uses the gradient steepest descent will be trapped in a local optima, but any search strategy that analyzes a wider region will be able to cross the valley among the optima and achieve better results. In order to obtain good results for multimodal problems, the search strategy must combine the exploratory and exploitative components efficiently. Since the ABC algorithm combine the exploration and exploitation processes succesfully, it shows high performance on training feed-forward ANNs for classification problems considered in this work.

Table 4. Experimental Results 30 runs of ANN training process results produced by BP, GA and ABC Algorithms. MMSE:Mean of Mean Squared Errors of 30 Runs, SDMSE: Standard Deviation of Mean Squared Errors of 30 runs.

Algorithms		XOR6	XOR9	XOR13	3-Bit Parity	Enc. Dec.
BP (GD)	MMSE	0.1182	0.212	0.2468	0.2493	0.0809
	SDMSE	0.0763	0.0369	0.008	0.0025	0.0756
BP (LM)	MMSE	0.1107	0.0491	0.0078	0.0209	0.0243
	SDMSE	0.0637	0.0646	0.0223	0.043	0.0424
GA	MMSE	0.099375	0.047968	0.015200	0.028725	0.016400
	SDMSE	0.02785	0.052000	0.022850	0.032900	0.031300
ABC	MMSE	0.007051	0.006956	0.006079	0.006679	0.008191
	SDMSE	0.002305	0.002402	0.003182	0.002820	0.001864

Table 5. Experimental Results 30 runs of ANN training process results produced by BP, GA and ABC Algorithms. ME: Mean of Epoch Numbers, SDE: Standard Deviation of Epoch Numbers, MG: Mean of Generation Numbers, SDG: Standard Deviation of Generation Numbers, MC: Mean of Cycle Numbers, SDC: Standard Deviation of Cycle Numbers.

Algorithms		XOR6	XOR9	XOR13	3-Bit Parity	Enc. Dec.
BP (GD)	ME	31603	500	250	1600	2020
	SDE	2176	0	0	0	236.8019
BP (LM)	ME	67.53	13	9	21.333	83.3667
	SDE	59.3759	6.8304	3.2056	10.0046	174.1683
GA	MG	7500	77.067	38.6000	501.1333	400.1333
	SDG	0	33.394	25.0236	415.8687	340.4838
ABC	MC	2717.4	32	28.2	179.066666	185
	SDC	3.359377	0.182827	1.241569	12.792384	5.842378

Table 6. Success Rates of BP(GD), BP(LM), GA and ABC Algorithms

Algorithms	XOR6	XOR9	XOR13	3-Bit Parity	Enc. Dec.
BP (GD)	3	0	0	0	2
BP (LM)	6	66.66	96.66	86.66	73.33
GA	0	40	76.6667	63.3333	86.6667
ABC	100	100	100	100	100

5 Conclusion

In this work, Artificial Bee Colony Algorithm which is a new, simple and robust optimization algorithm has been used to train feed-forward artificial neural networks for classification purpose. The performance of the algorithm has been compared with the traditional back propagation algorithm and the genetic algorithm which is a well-known evolutionary algorithm. Results of the experiments show that the Artificial Bee Colony algorithm can be successfully applied to train feed-forward neural networks. The application of ABC to other classification test problems such as iris, diabetes, cancer classification and the implementation of the algorithm for optimizing the network structure as well as optimizing weights remain as future works.

References

1. Rumelhart, D.E., Williams, R.J., Hinton, G.E.: Learning internal representations by error propagation. Parallel Distributed Processing: Explorations in the Microstructure of Cognition 1, 318–362 (1986)
2. Liu, Z., Liu, A., Wang, C., Niu, Z.: Evolving neural network using real coded genetic algorithm (GA) for multispectral image classification. Future Generation Computer Systems 20, 1119–1129 (2004)
3. Craven, M.P.: A Faster Learning Neural Network Classifier Using Selective Backpropagation. In: Proceedings of the Fourth IEEE International Conference on Electronics, Circuits and Systems, Cairo, Egypt, December 15-18, 1997, vol. 1, pp. 254–258 (1997)
4. Whitley, D., Starkweather, T., Bogart, C.: Genetic algorithms and neural networks: Optimizing connections and connectivity. Techn. Rep. CS 89-117, Department of Computer Science, Colorado State University (1989)
5. Rudnick, M.: A bibliography of the intersection of genetic search and artificial neural networks. Techn. Rep. No. CS/E 90- 001. Oregon Graduate Center, Beaverton, OR (1990)
6. Weiß, G.: Combining neural and evolutionary learning: Aspects and approaches. Techn. Rep. FKI-132-90. Institut fur Informatik, Technische Universit at Munchen (1990)
7. Weiß, G.: Towards the synthesis of neural and evolutionary learning. In: Omidvar, O. (ed.) Progress in neural networks, ch. 5, vol. 5, Ablex, Norwood, NJ (1993)
8. Albrecht, R.F., Reeves, C.R., Steele, N.C. (eds.): Artificial neural nets and genetic algorithms. In: Proceedings of the International Conference in Innsbruck, Austria. Springer, Heidelberg (1993)
9. Jones, A.J.: Genetic algorithms and their applications to the design of neural networks. Neural Computing & Applications 1, 32–45 (1993)
10. Ling, S.H., Lam, H.K., Leung, F.H.F., Lee, Y.S.: A Genetic Algorithm Based Variable Structure Neural Network, Industrial Electronics Society, 2003. In: IECON '03. The 29th Annual Conference of the IEEE, vol. 1, pp. 436–441 (2003)
11. Marshall, S.J., Harrison, R.F.: Optimization and Training of Feedforward Neural Networks By Genetic Algorithms, Artificial Neural Networks, 1991. In: Second International Conference on, November 18-20, 1991, pp. 39–43 (1991)
12. Verma, B., Ghosh, R.: A novel evolutionary neural learning algorithm, Evolutionary Computation, 2002. In: CEC '02. Proceedings of the 2002 Congress on, vol. 2, pp. 1884–1889 (2002)
13. Gao, Q., Lei, K.Q.Y., He, Z.: An Improved Genetic Algorithm and Its Application in Artificial Neural Network, Information, Communications and Signal Processing, 2005. In: Fifth International Conference on, December 06-09, 2005, pp. 357–360 (2005)
14. Yao, X.: Evolutionary artificial neural networks. International Journal of Neural Systems 4(3), 203–222 (1993)

15. Eberhart, R.C., Dobbins, R.W.: Designing Neural Network explanation facilities using Genetic algorithms. In: IEEE International Joint Conference on Publication, November 18-21,1991, vol. 2, pp. 1758–1763 (1991)
16. Jelodar, M.S., Fakhraie, S.M., Ahmadabadi, M.N.: A New Approach for Training of Artificial Neural Networks using Population Based Incremental Learning (PBIL). In: International Conference on Computational Intelligence, pp. 165–168 (2004)
17. Tsai, J.T., Chou, J.H., Liu, T.K.: Tuning the Structure and Parameters of a Neural Network by Using Hybrid Taguchi-Genetic Algorithm. IEEE Transactions on Neural Networks 17(1) (2006)
18. El-Gallad, A.I., El-Hawary, M., Sallam, A.A., Kalas, A.: Swarm-intelligently trained neural network for power transformer protection. In: Canadian Conference on Electrical and Computer Engineering, vol. 1, pp. 265–269 (2001)
19. Mendes, R., Cortez, P., Rocha, M., Neves, J.: Particle swarm for feedforward neural network training. In: Proceedings of the International Joint Conference on Neural Networks, vol. 2, pp. 1895–1899 (2002)
20. Van der Bergh, F., Engelbrecht, A.: Cooperative learning in neural networks using particle swarm optimizers. South African Computer Journal 26, 84–90 (2000)
21. Ismail, A., Engelbrecht, A.: Global optimization algorithms for training product unit neural networks. In: International Joint Conference on Neural Networks IJCNN 2000, vol. 1, pp. 132–137. IEEE Computer Society, Los Alamitos, CA (2000)
22. Kennedy, J., Eberhart, R.: Swarm Intellegence. Morgan Kaufmann Publishers, San Francisco (2001)
23. Meissner, M., Schmuker, M., Schneider, G.: Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training. BMC Bioinformatics 7, 125 (2006)
24. Fan, H., Lampinen, J.: A Trigonometric Mutation Operation to Differential Evolution. Journal of Global Optimization 27, 105–129 (2003)
25. Ilonen, J., Kamarainen, J.I., Lampinen, J.: Differential Evolution Training Algorithm for Feed-Forward Neural Networks
26. Pavlidis, N.G., Tasoulis, D.K., Plagianakos, V.P., Nikiforidis, G., Vrahatis, M.N.: Spiking Neural Network Training Using Evolutionary Algorithms, Neural Networks, 2005. In: IJCNN '05. Proceedings 2005 IEEE International Joint Conference, vol. 4, pp. 2190–2194 (2005)
27. Plagianakos, V.P., Vrahatis, M.N.: Training Neural Networks with Threshold Activation Functions and Constrained Integer Weights, Neural Networks, 2000. In: IJCNN 2000. Proceedings of the IEEE-INNS-ENNS International Joint Conference, vol. 5, pp. 161–166 (2000)
28. Plagianakos, V.P., Vrahatis, M.N.: Neural Network Training with Constrained Integer Weights, Evolutionary Computation, 1999. In: CEC 99. Proceedings of the 1999 Congress, vol. 3, p. 2013 (1999)
29. Yu, B., He, X.: Training Radial Basis Function Networks with Differential Evolution, Granular Computing, 2006. In: IEEE International Conference on, May 10-12, 2006, pp. 369–372 (2006)
30. Yao, X., Liu, Y.: A New Evolutionary System for Evolving Artificial Neural Networks. IEEE Transactions on Neural Networks 8(3), 694–713 (1997)
31. Gao, W.: New Evolutionary Neural Networks, 2005. In: First International Conference on Neural Interface and Control Proceedings, May 26-28, 2005, Wuhan, China (2005)

32. Davoian, K., Lippe, W.: A New Self-Adaptive EP Approach for ANN Weights Training, Enformatika Transactions on Engineering, Computing and Technology, Barcelona, Spain, October 22-24, 2006, vol. 15, pp. 109–114 (2006)
33. Leung, C., Member, Chow, W.S.: A Hybrid Global Learning Algorithm Based on Global Search and Least Squares Techniques for Backpropagation Networks, Neural Networks,1997. In: International Conference on, vol. 3, pp. 1890–1895 (1997)
34. Karaboga, D.: An Idea Based On Honey Bee Swarm For Numerical Optimization, Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department (2005)
35. Basturk, B., Karaboga, D.: An Artificial Bee Colony (ABC) Algorithm for Numeric function Optimization. In: IEEE Swarm Intelligence Symposium, May 12-14, 2006, Indianapolis, Indiana, USA (2006)
36. Weiß, G.: Neural Networks and Evolutionary Computation. PartI: Hybrid Approaches in Artificial Intelligence. In: International Conference on Evolutionary Computation, pp. 268–272 (1994)
37. Fahlman, S.: An empirical study of learning speed in back-propagation networks, Technical Report CMU-CS-88-162, Carnegie Mellon University, Pittsburgh, PA 15213 (September 1988)