Training Feed-Forward Neural Networks Using Firefly Algorithm

Ivona BRAJEVIC, Milan TUBA
Faculty of Computer Science
University Megatrend Belgrade
Bulevar umetnosti 29, N. Belgrade
SERBIA
ivona.brajevic@googlemail.com, tuba@ieee.org

Abstract: - In this work, firefly algorithm (FA) is used in training feed-forward neural networks (FNN) for classification purpose. In experiments, three well-known classification problems have been used to evaluate the performance of the proposed FA. The experimental results obtained by FA were compared with the results reported by artificial bee colony (ABC) algorithm and genetic algorithm (GA). Also, since the choice of transfer functions may strongly influence the performance of neural networks, the FA results obtained by using traditional sigmoid transfer function and by using sine transfer function were compared.

Key-Words: - Swarm intelligence, Firefly algorithm, Feed-Forward Neural Networks

1 Introduction

Artificial Neural Network (ANN) consists of an interconnected group of simple processing elements (artificial neurons) with internal adjustable parameters (weights). These networks can learn an arbitrary vector mapping from the space of input to the space of output by modification of these adjustable parameters. There are different ways in which information can be processed by a neuron, and different ways of connecting the neurons to one another. A variety of neural network structures have been proposed for signal processing, pattern recognition, control, etc. One of the major emphasis in neural network research is on learning algorithms and architectures. Finding a suitable network structure and finding optimal weight values make design of artificial neural networks difficult optimization problems.

A feed-forward neural network (FNN) is an artificial neural network where connections between the nodes do not form a directed cycle. These artificial networks have a number of properties which make them particularly suited to complex pattern classification problems. On the other side, the success of their application to some real world problems depends on a training algorithm. They need a training algorithm which reliably finds a nearly globally optimal set of weights in a relatively short time. Traditional algorithm for training FNN, called backpropagation, can often find a good set of weights in a reasonable amount of time [1]. Backpropagation is a variation of gradient search and the key to backpropagation is a method for calculating the gradient of the error with respect to the weights for a given input by propagating error backwards through the network. However, there are some drawbacks to backpropagation such as getting stuck in local minima and computational complexity [1].

Many global optimization methods have been proposed for training FNN in order to overcome the disadvantages of gradient based algorithms. Specially, some nature inspired metaheuristics such as genetic algorithm (GA), inspired by Darwin's theory about evolution, particle swarm optimization (PSO) algorithm, inspired by the social behavior of birds or fishes and artificial bee colony (ABC) algorithm, based on honey bee foraging behavior, have been successfully applied for training FNN [2], [3], [4], [5], [6].

The Firefly algorithm (FA) is a novel swarm intelligence metaheuristic based on the idealized behaviour of the flashing characteristics of fireflies [8], [10]. In this paper, its application to the training of FNN is investigated. Considering the fact that the choice of transfer functions may strongly influence the performance of neural networks [7], [9], [11], the FA results are produced by using sigmoid transfer function, as a traditional transfer function, and by using sine transfer function, as a less used transfer function. The FA results obtained by using sigmoid transfer function are compared with the results reported by artificial bee colony (ABC) and genetic algorithms (GA) [5], which are also

This research is supported by Ministry of Science, Republic of Serbia, Project No. 44006

obtained by using sigmoid transfer function for the same benchmarks. Also, in order to investigate the impact of the transfer function on the performance of firefly learning algorithm, and hence on FNN performance, the FA results provided by sigmoid and by sine transfer functions are compared. This paper is organized as follows. In Section 2, training FNN is described. Section 3 presents firefly algorithm. Benchmark problems considered in this work are described in Section 4. In Section 5, experiments and results are presented. Our conclusions are provided in Section 6.

2 Training Feed-Forward Artificial Neural Networks

The basic FNN produces its output by transforming input data, and consists of an input layer, one or more hidden layers, and an output layer. Each layer contains nodes or neurons. Interconnections within the network are such that neurons in layer *i* are connected to neurons in layer *i+1*, that is, each neuron in layer *i* is connected to every neuron in the adjacent layer *i+1*. Each interconnection has a scalar weight associated with it that is adjusted during the training phase. Fig.1 shows three layered feed-forward networks with one hidden layer.

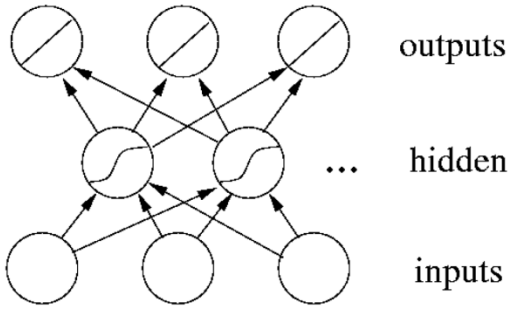

Fig.1 Typical structure of a feed-forward artificial neural network

In FNN, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. Output of the i^{th} neuron can be described by:

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

where y_i is the output of the node, x_j is the j^{th} 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. Bias terms can be interpreted as additional weights.

The transfer function is often the same for all the nodes in a FNN. The node transfer function is usually a nonlinear function such as a sigmoid function, although there is no a priori reason why models based on this function should always provide optimal decision borders [7].

When utilizing evolutionary algorithms to train FNN, the neural network is trained by minimizing an error function. The error function used in our experiments is the Mean Squared Error (MSE), which can be calculated according to Eq.(2):

$$MSE(w_{ij}) = \frac{1}{pm} \sum_{i=1}^{p} \sum_{j=1}^{m} (T_{ij} - F_{ij})^2 \quad (2)$$

where p denotes the number of training patterns, *m* the number of FNN outputs, T_{ij} the target, F_{ij} the actual value, both for the output *j* and the *i* input pattern. In nature inspired algorithms (NIA), the main idea is that each solution in the population represents a vector of connection weights of the FNN. Appropriate NIA operators are used to change the weights and the error produced by the FNNs is used as the fitness measure which guides selection.

3 Firefly algorithm

Flashing characteristics of fireflies can be summarized by the following three rules [8]:

1. All fireflies are unisex so that one firefly will be attracted to other fireflies regardless of their sex.
2. Attractiveness is proportional to firefly brightness. For any couple of flashing fireflies, the less bright one will move towards the brighter one. The brightness decreases when the distance between fireflies is increased. The brightest firefly moves randomly, because there is no other bug to attract it.
3. The brightness of a firefly is affected or determined by the landscape of the objective function to be optimized.

The attractiveness function of the firefly is established by:

$$\beta(r) = \beta_0 \cdot e^{-\gamma \cdot r^2} \quad (3)$$

where β_0 is the firefly attractiveness value at $r = 0$ and γ is the media light absorption coefficient.

Fireflies movement is based on the principles of attractiveness: when firefly *j* is more attractive than firefly *i* the movement is determined by the following equation:

$$x_{ik} = x_{ik} + \beta_0 \cdot e^{-\gamma \cdot r_{ij}^2} \cdot (x_{ik} - x_{jk}) + \alpha \cdot S_k \cdot \left(rand_{ik} - \frac{1}{2} \right) \quad (4)$$

where $k = 1,2,...,D$ (D is dimension of problem). In Eq. (4) third term is randomization term where $\alpha \in [0,1]$, S_k are the scaling parameters and $rand_{ik}$ is random number between 0 and 1. The solution quality can be improved by reducing the randomization parameter α with a geometric progression reduction scheme. In our proposed approach this reduction scheme can be described by:

$$\alpha(t) = \alpha_0 \cdot 0.9^t \quad (5)$$

where t is the current number of iteration. In addition, the scaling parameters S_k in all D dimensions are determined by:

$$S_k = |u_k - l_k| \quad (6)$$

where l_k and u_k are the lower and upper bound of the parameter x_{ik}.

Distance r_{ij} between fireflies i and j is obtained by Cartesian distance form by:

$$r_{ij} = \sqrt{\sum_{k=1}^{D} \left(x_{i,k} - x_{j,k} \right)^2} \quad (7)$$

The pseudo code of the firefly algorithm is given below:

Input

Objective function $f(x), x = (x_1,..., x_D)^T$ {*cost function}*

S=[l_k, u_k]; *k = 1, 2,..., D {given constraints}*

$\alpha_0, \beta_0, \gamma$ {a*lgorithm's parameters}*

Output

$x_{\min}$ {*obtained minimum location*}

Begin

Initialize a population of fireflies x_i *(i = 1, 2, ..., n)* randomly

Initialize algorithm's parameters α, β_0, γ

while *(t <MaxGeneration)*

for *i* = 1 : *n*

for *j* = 1 : *i*

if $((f(x_j) < f(x_i)))$

Obtain attractiveness by Eq.(3)

Move firefly *i* towards *j* in all *D* dimensions by Eq.(4)

Evaluate and update the new solution

end if

end for *j*

end for *i*

Reduce the randomization parameter α by Eq.(5)

Rank the fireflies and find the current best

end while

4 Benchmark problems

To evaluate the performance of firefly algorithm in feed-forward neural network training and to also carry out a comparison with state-of-the-art algorithms for training artificial neural networks three standard benchmark problems are used. These problems are listed below:

Exclusive-OR (XOR) Boolean function is the first problem used in the experiments. This problem is a classification problem mapping two binary inputs to a single binary output. Input and output patterns for test problem XOR are: (0 0; 0 1; 1 0; 1 1) → (0; 1 ;1; 0).

In the experiments, three feed-forward neural network structures were used: a 2-2-1 feed-forward neural network with six connection weights and no biases (having six parameters, XOR6), 2-2-1 feed-forward neural network with six connection weights and three biases (having 9 parameters, XOR9), 2-3-1 feed-forward neural network with nine connection weights and four biases totally thirteen parameters (XOR13).

Three bit parity problem is the second problem considered in the experiments. The N-bit parity function is a mapping defined on 2^N distinct binary vectors that indicates whether the sum of the N components of a binary vector is odd or even, i.e. if the number of binary inputs is odd, the output is 1, otherwise it is 0. The 3-Bit Parity Problem is a special case of N-Bit Parity Problem for N = 3, as well as XOR problem is a special case of N-Bit Parity Problem for N = 2. Input and output patterns for 3-Bit Parity test problem are: (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).

The XOR/parity problem has a long history in the study of neural networks [12]. A 3-3-1 feed-forward neural network structure is used for the 3-Bit Parity problem and it has twelve connection weights and four biases, totally sixteen parameters.

4-Bit Encoder/Decoder Problem is the third problem used in the experiments. This problem has 4 distinct input patterns, each having only one bit turned on. The output is a duplication of the inputs

Input and output patterns for this test problem are: (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). 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.

The parameter ranges, dimension of the problems, and the network structures used in these experiments are the same as in [5] and the values are presented in Table 1.

Problem	Range	NN structure	D
XOR6	[-100,100]	2-2-1 with 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

Table1. Parameters of the problems considered in the experiments. D: Dimension of the problem.

5 Experimental study

The firefly training algorithm has been implemented in Java programming language. Tests were done on a PC with Intel® Core™ i3-2310M processor @2.10 GHz with 2GB of RAM and Windows 7x64 Professional operating system. In the experiments, for each benchmark problem, the results obtained by FA using sigmoid transfer function and by FA using sine transfer function are generated and recorded. For each benchmark problem experiments were repeated 30 times and training processes were stopped when the mean squared error of the outputs associated with inputs was equal to or less than 0.01 (MSE≤0.01) or when the maximum cycle number has been reached.

5.1 Parameter settings

The parameter values utilized by FA in all the experiments are the following: size of population SP is 50, the value of parameter γ is 1, the initial value of attractiveness β_0 is 1, the initial value of parameter α is 0.9, i.e. the reduction scheme described by Eq.(5) was followed by reducing α from 0.9.

The parameter values utilized by ABC in all the experiments are the following*:* the value of "*limit*" is equal to *SN* x *D* where *D* is the dimension of the problem, colony size is 50 for all problems [5].

The parameter values utilized by GA in all the experiments are the following: 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 is 50 for all problems [5].

As in [5] maximum cycle number (*MCN*) for ABC, GA and FA 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.

5.2 Results and Discussion

Comparative results of the GA, ABC and FA produced by using sigmoid transfer function are given in Table 2.

When comparing FA with respect to GA, we can see that both algorithms show similar performances for problems XOR6 and 4-bit encoder/decoder problem. For the remaining three benchmark problems FA obtained better results, specially for problems XOR9 and XOR13. For problems XOR9 and XOR13, FA has 100% success, while the GA has 40% and 77% success respectively. Also, from the mean and standard deviations of cycle numbers it can be seen that FA converges significantly faster than GA.

When comparing FA with respect to ABC, we can see that FA performs worse on problems XOR6, 4-bit encoder/decoder and 3-Bit Parity problem. For these problems, FA shows tendency to be trapped in the local minima, while ABC has 100% success. For problems XOR9 and XOR13, both algorithms have 100% success, but from the mean and standard deviations of mean squared errors, as well as from the mean and standard deviations of cycle numbers, it can be seen that FA performs better. FA converges about three times faster to the global minima than ABC on these two problems.

Comparative FA results produced by using sigmoid transfer function and produced by using sine transfer function are summarized in Table 3.

As observed in Table 3, the FA results produced by using sine transfer function are much better that the FA results obtained by using sigmoid transfer function for XOR6 and 3-Bit Parity problems. For these problems, for sine transfer function, FA has 100% success and converges very fast to the global minima. On the other side, for problem XOR6, for sigmoid transfer function, FA could not reach the global minima in any run. For Encoder-Decoder problem, XOR9, and XOR13, FA results produced by using sine transfer function are similar with the FA results produced by using sigmoid transfer function.

ISBN: 978-1-61804-162-3

Algorithm	Statistics	XOR6	XOR9	XOR13	3-Bit Parity	Enc. Dec.
GA	MMSE	0.099375	0.047968	0.015200	0.028725	0.016400
	SDME	0.02785	0.052000	0.022850	0.032900	0.031300
	MC	7500	77.067	38.6000	501.1333	400.1333
	SDC	0	33.394	25.0236	415.8687	340.4838
	Success Rate	0	40	76.6667	63.3333	86.6667
ABC	MMSE	**0.007051**	0.006956	0.006079	**0.006679**	**0.008191**
	SDME	**0.002305**	0.002402	0.003182	**0.002820**	**0.001864**
	MC	**2717.4**	32	28.2	**179.066666**	**185**
	SDC	**3.359377**	0.182827	1.241569	**12.792384**	**5.842378**
	Success Rate	**100**	**100**	**100**	**100**	**100**
FA	MMSE	0.078939	**0.004971**	**0.003660**	0.015399	0.017473
	SDME	0.018168	**0.002677**	**0.002417**	0.023297	0.022646
	MC	7500	**9.3**	**7.866667**	181.633333	216.033333
	SDC	0	**3.089229**	**3.063041**	366.053137	391.988646
	Success Rate	0	**100**	**100**	83.333333	80

Table2. Experimental Results of 30 runs of FNN training process produced by GA, ABC and FA using sigmoid transfer function. MMSE: Mean of Mean Squared Errors, SDMSE: Standard Deviation of Mean Squared Errors, MC: Mean of Cycle Numbers, SDC: Standard Deviation of Cycle Numbers.

Transfer function	Statistics	XOR6	XOR9	XOR13	3-Bit Parity	Enc. Dec.
Sigmoid	MMSE	0.078939	**0.004971**	**0.003660**	0.015399	0.017473
	SDME	0.018168	**0.002677**	**0.002417**	0.023297	0.022646
	MC	7500	9.3	7.866667	181.633333	216.033333
	SDC	0	3.089229	3.063041	366.053137	391.988646
	Success Rate	0	**100**	**100**	83.333333	**80**
Sine	MMSE	**0.002308**	0.005418	0.005922	**0.006472**	**0.015747**
	SDME	**0.001884**	0.002289	0.002520	**0.002307**	**0.019705**
	MC	**1.066667**	**3.366667**	**5.1**	**35.766667**	**217.266667**
	SDC	**0.249443**	**2.651834**	**4.019536**	**8.389213**	**391.372791**
	Success Rate	**100**	**100**	**100**	**100**	**80**

Table3. Experimental Results of 30 runs of FNN training process produced by FA using sigmoid transfer function and FA using sine transfer function. MMSE: Mean of Mean Squared Errors, SDMSE: Standard Deviation of Mean Squared Errors, MC: Mean of Cycle Numbers, SDC: Standard Deviation of Cycle Numbers.

Conclusion

In this paper, the firefly algorithm is applied to train feed-forward artificial neural networks. Three well known classification problems are considered in this work. In the experiments, for each benchmark problem, the FA results are produced by using sigmoid transfer function and by using sine transfer function. The FA results obtained by using sigmoid transfer function are compared with the results reported by artificial bee colony (ABC) and genetic algorithms (GA), which are also

obtained for these benchmark problems by using sigmoid transfer function. According to these experimental results it can be concluded that FA performs better than GA algorithm, but worse than ABC algorithm for the majority of benchmark problems. Although for sigmoid transfer function firefly learning algorithm shows tendency to be trapped in the local minima, it has fast convergence speed for several benchmarks.

Further, in order to investigate the impact of the transfer function on the performance of firefly learning algorithm, and hence on FNN performance, the FA results provided by sigmoid and by sine transfer functions are compared. These experimental results show that for sine transfer function the tendency of firefly learning algorithm to be trapped in the local minima is significantly reduced. In this case FA has 100% success and a very fast convergence speed for almost all benchmarks.

Considering the experimental results, the possibility of simultaneously evolving weights and transfer functions may be the part of our future work. Also, the possible modifications in the search strategy of FA in order to provide better balance of exploration and exploitation will be investigated.

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, Volume 1, 1986, pp. 318–362

[2] D. J. Montana and L. Davis, Training Feedforward Neural Networks Using Genetic Algorithms, Proceedings of the International Joint Conference on Articial Intelligence, 1989, pp. 762-767

[3] 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, Volume 17, Issue 1, 2006, pp.69-80

[4] 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, 2002, pp. 1895–1899

[5] Karaboga D., Akay B., Ozturk C., Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks, LNCS: Modeling Decisions for Artificial Intelligence, Volume 4617, 2007, pp.318-329

[6] D. Karaboga, C. Ozturk, Neural Networks Training by Artificial Bee Colony Algorithm on Pattern Classification, Neural Network World, Volume 19, Number 3, 2009, pp.279-292

[7] X.S. Yang, Nature-Inspired Metaheuristic Algorithms, Luniver Press (2008)

[8] X.S.Yang, Firefly algorithms for multimodal optimization, in: Stochastic Algorithms: Foundations and Applications, SAGA 2009, Lecture Notes in Computer Sciences, Vol.5792, 2009, pp. 169-178

[9] W. Duch and N. Jankowski, Survey of neural transfer functions, Neural Computing Surveys, Volume 2, Issue 1, pp. 163—212, 1999.

[10] W. Duch and N. Jankowski. Transfer functions: hidden possibilities for better neural networks, In 9th European Symposium on Artificial Neural Networks, Brusells, Belgium, 2001, pp. 81–94

[11] M. Annunziato, I. Bertini, M. Lucchetti, S. Pizzuti, Evolving Weights and Transfer Functions in Feed Forward Neural Networks, Proc. EUNITE2003, Oulu, Finland, 2003

[12] D. Liu, M.E. Hohil, S. H. Smith, N-bit parity neural networks: new solutions based on linear programming, Neurocomputing, Vol.48, no.1-4, 2002, pp. 477–488