Direct and Indirect Evolutionary Designs of Artificial Neural Networks

O. Alba-Cisneros, A. Espinal, G. López-Vázquez, M. A. Sotelo-Figueroa, O. J. Purata-Sifuentes, V. Calzada-Ledesma, R. A. Vázquez and H. Rostro-González

Abstract Evolutionary Algorithms (EAs) and other kind of metaheuristics are utilized to either design or optimize the architecture of Artificial Neural Networks (ANNs) in order to adapt them for solving a specific problem; these generated ANNs are known as Evolutionary Artificial Neural Networks (EANNs). Their architecture components, including number of neurons or their kind of transfer functions, connectivity pattern, etc., can be defined through direct or indirect encoding schemes; the former, directly codifies ANN architecture components into the genotype of solutions, meanwhile the last one, requires to transform the solution's genotype through mapping processes to generate an ANN architecture. In this work, architectures of three-layered feed-forward ANNs are designed by using both encoding schemes to solve several well-known benchmark datasets of supervised classification problems. The results of designed ANNs by using direct and indirect encoding schemes are compared.

Keywords Artificial neural networks · Differential evolution · Direct encoding scheme · Indirect encoding scheme · Pattern classification

O. Alba-Cisneros · A. Espinal (✉) · M. A. Sotelo-Figueroa · O. J. Purata-Sifuentes
Departamento de Estudios Organizacionales, DCEA-Universidad de Guanajuato, Guanajuato, Mexico
e-mail: aespinal@ugto.mx

G. López-Vázquez
División de Estudios de Posgrado e Investigación, Tecnológico Nacional de México / Instituto Tecnológico de León, León, Guanajuato, Mexico

V. Calzada-Ledesma
Tecnológico Nacional de México / Instituto Tecnológico Superior de Purísima del Rincón. Purísima del Rincón, Guanajuato, Mexico

R. A. Vázquez
Grupo de Sistemas Inteligentes, Facultad de Ingeniería, Universidad de La Salle, Delegación Cuauhtémoc, Ciudad de México, Mexico

H. Rostro-González
Departamento de Electrónica, DICIS-Universidad de Guanajuato, Salamanca, Guanajuato, Mexico

O. Castillo et al. (eds.), *Intuitionistic and Type-2 Fuzzy Logic Enhancements in Neural and Optimization Algorithms: Theory and Applications*, Studies in Computational Intelligence 862, https://doi.org/10.1007/978-3-030-35445-9_31

1 Introduction

Artificial Neural Networks (ANNs) are mathematical models that mimic in some degree the neuronal dynamics observed in brains of living beings. They have been successfully applied in different disciplines both theoretical and practical to solve problems such as pattern recognition [28], forecasting [27], etc. With respect of an ANN's performance, it is well-known that it is highly related with its architecture or topology; which is mainly defined around three components: computing units, communication links and message type [17].

The designing process of an ANN's architecture is commonly performed by human experts that define several of its structural and other kind of components through rules of thumb, heuristics, by trial-and-error and expertise. However, such process may be time consuming or represent a challenge; e.g. combinatorial problems may arise when designing it [1] or issues related to learning an specific task given a fixed architecture of an ANN [2, 3, 16, 17].

The ANNs' designing and learning problems may be partially or completely overcome by using metaheuristic algorithms; these ANNs are known as Evolutionary Artificial Neural Networks (EANNs). Metaheuristics can be used as a replacement of traditional learning rules or to improve their learning speed, among other aspects. Besides, they can be implemented as mechanisms to design or modify architectures of ANNs [8, 24, 26]. The use of such algorithms allow to prescind in some manner of human experts in ANN's design phase.

Focusing in design of ANNs' architectures by means of metaheuristic algorithms, there can be distinguished two encoding schemes to represent architecture components of an ANN; which are: direct and indirect [6, 20]. Basically, the former scheme codifies ANN architecture components into the genotype of solutions, meanwhile, the last one requires to transform the solution's genotype through mapping processes to generate an ANN architecture.

In this paper, there are developed three-layered feed-forward ANNs with partial synaptic connections between their layers though evolutionary metaheuristics by using both direct and indirect encoding schemes. The partially-connected feed-forward EANNs contain a subset of the possible synaptic connections of their counterpart fully-interlayer-connected; which are smaller topologies that can have similar or better performances than their fully-connected version [7]. Besides, jointly with the evolutionary design process it is possible to attend some suggestions to ease the ANNs' learning such as constraint either the ANNs' architecture or the problem to deal with, or to have a learning algorithm that modifies the ANNs' architecture during training process [1, 17]. The rest of the paper is organized as follows. In Sect. 2, there are described the concepts required for the development of this work. The implementation details of designing methodologies using direct and indirect encoding schemes are explained in Sect. 3. The experimental configuration and obtained results are shown in Sect. 4. Finally, Sect. 5 presents the conclusions and future work.

2 Background

2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are bioinspired models constituted by computing units (neurons) interconnected in a wiring pattern (synaptic connections) sharing messages of some type (neural activity) [17]; such elements define the architecture or topology of an ANN. Focusing in a three-layered feed-forward ANN, it can be considered as a system that maps an input signal ($x \in \mathbb{R}^N$) into an output signal ($y \in \mathbb{R}^M$) [15]; such mapping process is carried out by sending x through input, hidden and output layers. Each layer has computing units connected to other units in the contiguous layer; such connections can be full or partial. The number of computing units in input and output layers can be defined according the size of input and output signals; N and M nodes respectively. However, there is not a well-established way to define the number of units that hidden layer must have; it is usually define by trial-and-error, heuristics or human expertise.

Computing units in input layer passes directly their signal to nodes in hidden layer. Meanwhile, each unit in next two layers calculates and evaluates its input to produce its output. The input inp_i of i-th computing unit is commonly calculated by Eq. 1

$$inp_i = b_i + s \cdot w_i \tag{1}$$

where b_i is the bias, s is the input signal and w_i is the synaptic weight vector. The output of i-th unit is produced by means of an activation function $f_i(inp_i)$; which can be linear or nonlinear [14] and it is usually chosen according the problem to solve.

2.2 Differential Evolution

Differential Evolution (DE) [23] is an evolutionary metaheuristic for solving d-dimensional real-valued optimization problems. In DE, a population of NP solutions of d-dimensions is uniformly create over the search space and evaluated according a fitness function. For each individual x_i in the population is generated a mutant vector v_i by using Eq. 2

$$v_i = x_{r1} + F(x_{r2} - x_{r3}) \tag{2}$$

where r_1,r_2 and r_3 are three randomly selected indexes ($i \neq r_1 \neq r_2 \neq r_3$ and $r_1, r_2, r_3 \in \{1, \ldots, NP\}$) and $F \in (0, 2]$ is a differential weight. Next, a trial vector u_i is created based on x_i and v_i by using Eq. 3

$$u_{j,i} = \begin{cases} v_{j,i} & \text{if } r_j \leq C_r \text{ or } j = J_r \\ x_{j,i} & \text{if } r_j > C_r \text{ or } j \neq J_r \end{cases} \tag{3}$$

where $j = 1, \ldots, d$, $Cr \in [0, 1]$ is a crossover rate, $J_r \in \{1, \ldots, d\}$ is a random index generated for current individual i and $r_j \in U[0, 1)$ is an uniform random number generated per each dimension. An individual is selected for the next generation according to Eq. 4

$$x_i = \begin{cases} u_i \text{ if } f(u_i) \leq f(x_i) \\ x_i \qquad \text{otherwise} \end{cases} \tag{4}$$

where $f(\bullet)$ is the fitness function. This process is repeated until a stop criterion is met.

2.3 Grammatical Evolution

Grammatical Evolution (GE) [22] is a grammar-based form of Genetic Programming [18]. In GE, the genotypic representation of individuals is formed by strings of numbers that are changed to the functional phenotypic representation through a mapping process [4] (also known as indirect representation). The mapping process uses a Context-Free Grammar (CFG) in Backus-Naur Form (BNF) to derive the phenotypic representation of an individual. A metaheuristic is used to perform the search process by modifying the genotype of the individual with only the knowledge of its fitness value.

A CFG is defined as 4-tuple as follows [19]: $G = \{\Sigma_N, \Sigma_T, P, S\}$ where Σ_T is the set of terminals, Σ_N is the set of non-terminals, $S \in \Sigma_N$ is the start symbol and P is the productions set; A production is usually written $A \rightarrow \alpha$. The BNF is a notation used to describe productions of CFGs as follows [25]:

- The symbol $\models$ is used instead of $\rightarrow$.
- Every non-terminal symbol is enclosed in brackets of the form <>.
- All production with the same non-terminal in the left-hand side members are combined into one statement with all the right-hand side members, separated by vertical bars (|).

The canonical GE's mapping process, the Depth-first [22]; is used to choose the next production based on the current left-most non-terminal symbol in the expression. The Depth-first mapping process is given in Eq. (5):

$$Rule = c\%r \tag{5}$$

where c is the current component value of individual's genotype and r is the number of production rules available for the current non-terminal. The selected production is used to replace the current left-most non-terminal in the word being derived.

3 Design Methodologies

The methodologies' purpose is to design three-layered feed-forward ANNs by means of evolutionary algorithms through direct and indirect encoding schemes. The methodologies have the following considerations to design ANNs' architectures:

- Computing units in hidden layer. The methodologies must define in some manner the number of computing units in hidden layer.
- Activation functions of units in hidden and output layers. The available activation functions for designing process are shown in Table 1.
- Synaptic connections and weights. At the end of designing process, the result must be a designed ANN with its connections defined and weights calibrated.

3.1 Design Methodology with Direct Codification

The ANN design methodology with direct codification is based on Garro's works [9–13], where there were designed three-layered feed-forward ANNs with inter and supra layer connections and variable activation function types; i.e. computing units of input layers can be connected to units of hidden and output layers and units of hidden layer can only be connected to units in output layer. In this work, there are only developed three-layered feed-forward ANNs with inter layer connections. The number of computing units Q in the whole ANN is defined by Eq. 6

$$Q = (N + M) + \frac{(N + M)}{2} \tag{6}$$

Table 1 Activation functions

Name	ID	Label	Function
Logistic	0	LS	$f(inp) = \frac{1}{1+e^{-inp}}$
Hyperbolic Tangent	1	HT	$f(inp) = \frac{e^{inp}-e^{-inp}}{e^{inp}+e^{-inp}}$
Sinusoidal	2	SN	$f(inp) = \sin(inp)$
Gaussian	3	GS	$f(inp) = e^{-inp^2}$
Linear	4	LN	$f(inp) = inp$
Hard limit	5	HL	$f(inp) = \begin{cases} 1 \text{ if } inp \geq 0 \\ 0 \text{ otherwise} \end{cases}$
Rectified linear	6	RL	$f(inp) = \begin{cases} x \text{ if } inp > 0 \\ 0 \text{ otherwise} \end{cases}$

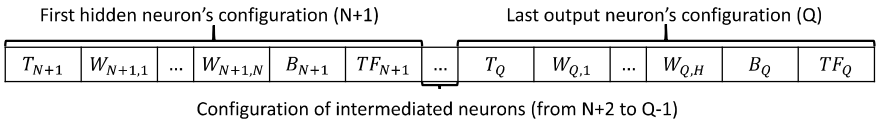

Fig. 1 Scheme for representing the design parameters of an ANN using the direct encoding scheme

where N and M are the size of input and output vector, respectively. Then, the number of computing units in the hidden layer is given by Eq. 7.

$$H = Q - (N + M) \tag{7}$$

In the adapted version for this work, the number of parameters dim_d for the designing process is calculated by Eq. 8; the codification rule is as explained in [13] (see Fig. 1), but there are only considered parameters of computing units in hidden and output layers. For each computing units there are codified next parameters: topology (presynaptic connections) T_{index}, set of synaptic weights W_{index}, bias B_{index} and activation function TF_{index}.

$$dim_d = [H * (N + 3)] + [M * (H + 3)] \tag{8}$$

3.2 Design Methodology with Indirect Codification

The ANN design methodology with indirect codification is based on Quiroz's works [21], where there were designed three-layered feed-forward ANNs with inter layer connections with logistic activation functions. In this work, there are only developed three-layered feed-forward ANNs with inter layer connections and variable activation function types. In Fig. 2 is show the generic indirect representation of an ANN's architecture, it has two main parts: hidden layer configuration and output layer configuration. The former allows to define the configuration of computing units in hidden layer without directly limiting the number of units; the configuration of hidden units includes: activation function, bias weight, presynaptic connection and weights. The last part defines the activation functions and bias weights of output units.

The CFG in BNF which allows to deploy expressions like In Fig. 2 is given in In Grammar. 1.1.

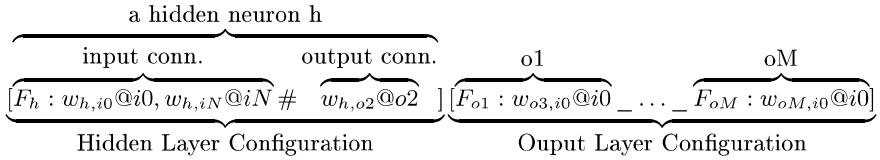

Fig. 2 Scheme for representing the design parameters of an ANN using the indirect encoding scheme

⟨network⟩ ⊨ [⟨hiddenNeurons⟩][⟨outputNeurons⟩]
⟨hiddenNeurons⟩ ⊨ ⟨hiddenNeuron⟩ | ⟨hiddenNeuron⟩_⟨hiddenNeurons⟩
⟨hiddenNeuron⟩ ⊨ ⟨func⟩:⟨weight⟩@i0,⟨inputs⟩#⟨outputs⟩
⟨outputNeurons⟩ ⊨ ⟨func⟩:⟨weight⟩@i0_ . . . _ ⟨func⟩:⟨weight⟩@i0
⟨func⟩ ⊨ LS | HT | SN | GS | LN | HL | RL
⟨inputs⟩ ⊨ ⟨input⟩ | ⟨input⟩,⟨inputs⟩
⟨outputs⟩ ⊨ ⟨output⟩ | ⟨output⟩,⟨outputs⟩
⟨input⟩ ⊨ ⟨weight⟩@⟨inputID⟩
⟨output⟩ ⊨ ⟨weight⟩@⟨outputID⟩
⟨inputID⟩ ⊨ i1 | . . . | iN
⟨outputID⟩ ⊨ o1 | . . . | oM
⟨weight⟩ ⊨ ⟨sign⟩⟨digitList⟩.⟨digitList⟩
⟨sign⟩ ⊨ + | -
⟨digitList⟩ ⊨ ⟨digit⟩ | ⟨digit⟩⟨digitList⟩
⟨digit⟩ ⊨ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Grammar 1.1 BNF grammar for designing second-generation ANNs, as in [21]

4 Experiments and Results

The ANNs' methodologies design with direct and indirect encoding schemes were tested to assess the quality of generated EANNs. In Table 2 there are shown the well-

Table 2 Description of supervised classification benchmark datasets (obtained from [5])

Name	# Features	# Classes	# Instances
Balance	4	3	625
BCW	9	2	683
Glass	9	6	214
Ionosphere	33	2	351
Iris Plant	4	3	150
Wine	13	3	178

known supervised classification benchmarks that were used to test and compare both methodologies.

For design methodology with direct encoding scheme there was DE to drive the optimization process. Meanwhile, for the design methodology with indirect encoding scheme there were used GE with DE as search engine, the Depth-first mapping process and Grammar. 1.1 for the designing process. Both designing methodologies use the classification error rate as fitness function. The common configuration parameters of DE is as follows:

- Population size $NP = 100$.
- $F = 0.8$.
- $Cr = 0.9$.
- 1000000 function calls.

The dimension of individuals for methodology with direct encoding scheme is calculated by Eq. 8 and for the methodology with indirect encoding scheme the dimension was set to 500.

For each methodology, there were executed 35 runs per dataset; the obtained performance results in designing and testing phases for both methodologies are shown in Table 3. In bold font are marked the median value as representative result, it can be observed that methodology with direct encoding scheme got better results for most benchmark datasets in both phases.

In Figs. 3 and 4 are shown the best representative architectures obtained for BCW dataset by methodology with direct and indirect encoding schemes; respectively. Finally, there is presented a summary of architecture's characteristic obtained by both methodologies in Table 4, columns labeled as **Feat.** show the mean and standard deviation of used input features, columns labeled as **Hidden** show the mean and standard deviation the number of hidden computing units used or generated and columns labeled as **Conn.** show the mean and standard deviation of the total synaptic connections used; in this case, there can be observed that architectures generated with the methodology with indirect encoding scheme are more compact than those designed with the direct encoding scheme.

5 Conclusions and Future Work

In this work are designed EANNS through evolutionary algorithms by using direct and indirect encoding schemes. The results of designs over several well-known supervised classification benchmark dataset were numerically compared. The results show that methodology with direct encoding scheme achieves better performance than the other methodology; however the designs generated by the methodology with indirect encoding scheme are more compact that those designed through direct encoding scheme. Both methodologies show capabilities for reducing the size of input vector and allow to practically prescind of a human expert.

Table 3 Designing and testing performances obtained using direct and indirect encoding schemes for EANNs

		Direct Codification				**Indirect Codification**			
	Dataset	**Min**	**Median**	**Max**	$\mu \pm \sigma$	**Min**	**Median**	**Max**	$\mu \pm \sigma$
Designing	Balance	0.77	**0.86**	0.92	0.86 ± 0.04	0.70	0.73	0.83	0.73 ± 0.02
	BCW	0.95	**0.98**	0.99	0.97 ± 0.01	0.94	0.96	0.97	0.96 ± 0.01
	Glass	0.45	0.47	0.51	0.47 ± 0.02	0.45	**0.48**	0.51	0.47 ± 0.01
	Ionosphere	0.75	0.77	0.81	0.77 ± 0.02	0.81	**0.83**	0.87	0.84 ± 0.01
	Iris Plant	0.84	**0.92**	0.99	0.92 ± 0.03	0.67	0.68	0.88	0.71 ± 0.05
	Wine	0.66	**0.72**	0.81	0.72 ± 0.03	0.65	0.69	0.72	0.69 ± 0.02
Testing	Balance	0.73	**0.85**	0.91	0.84 ± 0.04	0.64	0.70	0.80	0.71 ± 0.04
	BCW	0.94	**0.96**	0.99	0.96 ± 0.01	0.92	0.95	0.98	0.95 ± 0.01
	Glass	0.28	0.42	0.50	0.42 ± 0.04	0.32	**0.45**	0.48	0.43 ± 0.04
	Ionosphere	0.67	0.73	0.80	0.73 ± 0.03	0.74	**0.81**	0.86	0.81 ± 0.03
	Iris Plant	0.76	**0.91**	0.97	0.89 ± 0.06	0.63	0.67	0.95	0.69 ± 0.07
	Wine	0.53	**0.68**	0.87	0.68 ± 0.08	0.59	0.67	0.70	0.67 ± 0.02

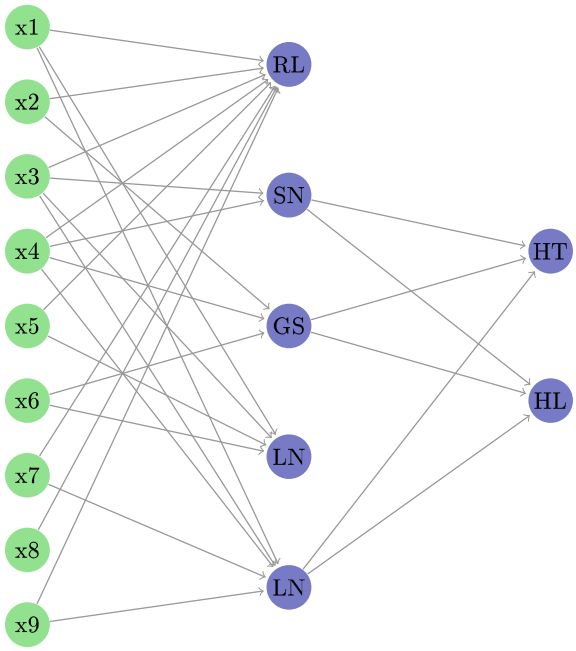

Fig. 3 Representative ANN's architecture for BCW dataset obtained by using direct encoding scheme

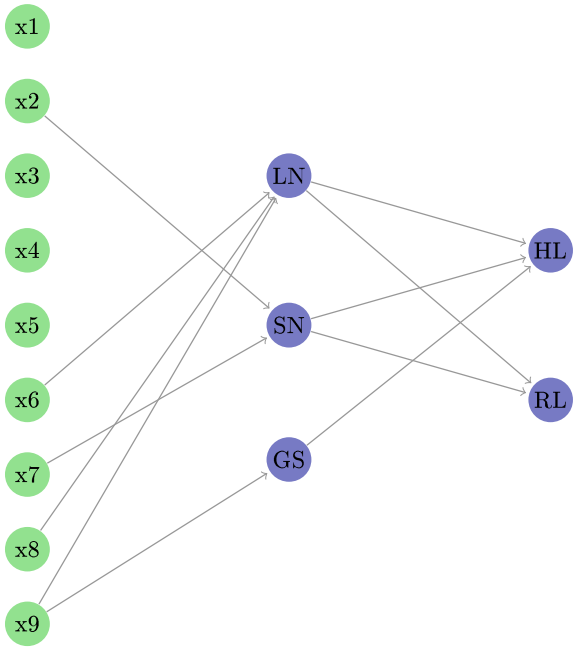

Fig. 4 Representative ANN's architecture for BCW dataset obtained by using indirect encoding scheme

Table 4 Summary of input features used, number of hidden neurons and number of synaptic connections in architectures of generated ANNs

	Direct codification			Indirect codification		
Dataset	**Feat.**	**Hidden**	**Conn.**	**Feat.**	**Hidden**	**Conn.**
Balance	3.97 ± 0.17	3.00 ± 0.00	17.57 ± 1.67	3.20 ± 0.72	1.97 ± 1.07	12.89 ± 4.01
BCW	8.54 ± 0.70	5.00 ± 0.00	33.43 ± 3.97	3.86 ± 1.80	2.06 ± 1.53	12.06 ± 6.58
Glass	8.83 ± 0.38	7.00 ± 0.00	63.69 ± 5.79	3.49 ± 1.88	2.14 ± 1.26	17.20 ± 6.13
Ionosphere	31.00 ± 0.00	17.00 ± 0.00	469.29 ± 38.72	3.83 ± 3.28	1.97 ± 1.29	11.2 ± 5.96
Iris Plant	3.40 ± 0.69	3.00 ± 0.00	16.97 ± 1.54	2.20 ± 1.11	1.97 ± 1.34	11.83 ± 5.66
Wine	13.00 ± 0.00	8.00 ± 0.00	75.14 ± 5.60	3.09 ± 2.09	2.00 ± 1.06	12.23 ± 5.00

As future work, authors attempt to combine good features of both encoding schemes to improve ANNs' designs, generalization capabilities and designing times. To test these methodologies with other architectures and kind of ANNs. Besides, to test with other classification problems and another type of problems.

Acknowledgements Authors wish to thank Universidad de Guanajuato, Tecnológico Nacional de México y Universidad de la Salle. This work was supported by the CONACyT Project FC2016-1961 "Neurociencia Computacional: de la teoría al desarrollo de sistemas neuromórficos".

References

1. Amaldi, E., Mayoraz, E., de Werra, D.: A review of combinatorial problems arising in feed-forward neural network design. Discrete Appl. Math. **52**(2), 111–138 (1994)
2. Blum, A.L., Rivest, R.L.: Training a 3-node neural network is NP-complete. Neural Netw. **5**(1), 117–127 (1992)
3. DasGupta, B., Siegelmann, H.T., Sontag, E.: On the intractability of loading neural networks. In: Theoretical Advances in Neural Computation and Learning, pp. 357–389. Springer, Heidelberg (1994)
4. Dempsey, I., O'Neill, M., Brabazon, A.: Foundations In Grammatical Evolution For Dynamic Environments, vol. 194. Springer, Heidelberg (2009)
5. Dheeru, D., Karra Taniskidou, E.: UCI Machine Learning Repository (2017)
6. Ding, S., Li, H., Su, C., Yu, J., Jin, F.: Evolutionary artificial neural networks: a review. Artif. Intell. Rev. **39**(3), 251–260 (2013)
7. Elizondo, D., Fiesler, E.: A survey of partially connected neural networks. Int. J. Neural Syst. **8**(5–6), 535–558 (1997)
8. Floreano, D., Dürr, P., Mattiussi, C.: Neuroevolution: from architectures to learning. Evol. Intell. **1**(1), 47–62 (2008)
9. Garro, B.A., Sossa, H., Vazquez, R.A.: Design of artificial neural networks using a modified particle swarm optimization algorithm. In: Proceedings Of The 2009 International Joint Conference On Neural Networks, IJCNN'09, pp. 2363–2370, Piscataway, NJ, USA, IEEE Press (2009)
10. Garro, B.A., Sossa, H., Vázquez, R.A.: Design of artificial neural networks using differential evolution algorithm. In: International Conference on Neural Information Processing, pp. 201–208. Springer, Heidelberg (2010)
11. Garro, B.A., Sossa, H., Vázquez, R.A.: Artificial neural network synthesis by means of artificial bee colony (abc) algorithm. In: 2011 IEEE Congress of Evolutionary Computation (CEC), pp. 331–338. IEEE (2011)
12. Garro, B.A., Sossa, H., Vázquez, R.A.: Evolving neural networks: a comparison between differential evolution and particle swarm optimization. In: International Conference in Swarm Intelligence, pp. 447–454. Springer, Heidelberg (2011)
13. Garro, B.A., Vázquez, R.A.: Designing artificial neural networks using particle swarm optimization algorithms. Comput. Intell. Neurosci. **2015**, 61 (2015)
14. Hagan, M., Demuth, H., Beale, M.: Neural Network Design. Martin Hagan (2014)
15. Haykin, S.: Neural Networks: Comprehensive Foundation. Prentice Hall (1999)
16. Judd, J.S.: On the complexity of loading shallow neural networks. J. Complexity **4**(3), 177–192 (1988)
17. Judd, J.S.: Neural Network Design and the Complexity of Learning. Neural Network Modeling and Connectionism Series. MIT press, Cambridge (MA) (1990)
18. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. A Bradford Book, vol. 1. MIT press, Cambridge (MA) (1992)