Engineering Applications of Artificial Intelligence 39 (2015) 1–13

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

Artificial neural network development by means of a novel combination of grammatical evolution and genetic algorithm

Fardin Ahmadizar [a,*], Khabat Soltanian [b], Fardin AkhlaghianTab [b], Ioannis Tsoulos [c]

[a] Department of Industrial Engineering, University of Kurdistan, Pasdaran Boulevard, Sanandaj, Iran
[b] Department of Software Engineering & Information Technology, University of Kurdistan, Pasdaran Boulevard, Sanandaj, Iran
[c] Department of Computer Engineering, Technological Institute of Epirus, Arta, Greece

ARTICLE INFO

Article history:
Received 28 April 2014
Received in revised form
24 September 2014
Accepted 7 November 2014
Available online 6 December 2014

Keywords:
Neural networks
Grammatical evolution
Genetic algorithm
Adaptive penalty approach
Classification problems

ABSTRACT

The most important problems with exploiting artificial neural networks (ANNs) are to design the network topology, which usually requires an excessive amount of expert's effort, and to train it. In this paper, a new evolutionary-based algorithm is developed to simultaneously evolve the topology and the connection weights of ANNs by means of a new combination of grammatical evolution (GE) and genetic algorithm (GA). GE is adopted to design the network topology while GA is incorporated for better weight adaptation. The proposed algorithm needs to invest a minimal expert's effort for customization and is capable of generating any feedforward ANN with one hidden layer. Moreover, due to the fact that the generalization ability of an ANN may decrease because of overfitting problems, the algorithm utilizes a novel adaptive penalty approach to simplify ANNs generated through the evolution process. As a result, it produces much simpler ANNs that have better generalization ability and are easy to implement. The proposed method is tested on some real world classification datasets, and the results are statistically compared against existing methods in the literature. The results indicate that our algorithm outperforms the other methods and provides the best overall performance in terms of the classification accuracy and the number of hidden neurons. The results also present the contribution of the proposed penalty approach in the simplicity and generalization ability of the generated networks.

1. Introduction

Artificial neural networks (ANNs) have risen from 1960s as a novel way to mimic the human brain. The learning ability of ANNs makes them a powerful tool for various applications, such as classification (Zhang, 2000; Cantu-Paz and Kamath, 2005; Castellani and Rowlands, 2009; Rivero et al., 2010; Castellani, 2013), clustering (Du, 2010), vision (Weimer et al., 2013; Zaidan et al., 2014), control systems (Li et al., 2014; Czajkowski et al., 2014), prediction (Muttil and Chau, 2006; Chen et al., 2008; Chen and Wang, 2010; Taormina et al., 2012; Weizhong, 2012; Khashei and Bijari, 2012; Coban, 2013), and many others (Rabuñal and Dorado, 2005). Designing the network architecture and training it are the most important problems with exploiting ANNs. In supervised problems, the task of training is the adaptation of the network weights so that the ANN can map a set of predefined input patterns to the desired corresponding outputs. BackPropagation (BP) algorithm (Rumelhart et al., 1986) is the most known ANNs training algorithm. Selecting effective input features, determining the number of hidden neurons, and the connectivity pattern of neurons is the task of designing the ANN topology that affects the network's leaning capacity and generalization, and usually needs to be performed by experts; they have to experiment on different topologies to find a suitable one.

Evolutionary neural networks (ENNs) refer to a class of research in which evolutionary algorithms (EAs) are used in the ANN designing and/or training. EAs are population-based, stochastic search algorithms stimulating the natural evolution. As EAs search globally and can handle infinitely large, non-differentiable and multimodal search space of architectures, this field has been interested by many researchers and ANN developers.

ENNs can be followed in three major groups. The first is the adaptation of the ANN weights by EAs, a form of the ANN training. The second group of works is the use of EAs for designing the architecture of ANNs, and the third is the Topology and Weight Evolving of ANNs (TWEANN) that includes the methods proposed to simultaneously evolve the weights and architecture. In the

* Corresponding author. Tel./fax: +98 871 6660073.
E-mail address: f.ahmadizar@uok.ac.ir (F. Ahmadizar).

http://dx.doi.org/10.1016/j.engappai.2014.11.003
0952-1976/

following, some of the important research lines in ENNs are reviewed; the more complete and state-of-the art reviews can be found in Yao (1999) and Azzini and Tettamanzi (2011).

1.1. ANN training algorithms

In the evolutionary training of ANNs, the network architecture is firstly determined, usually by experts. As the training of ANNs can be formulated as a search problem, the EA is used for exploring and exploiting the search space for finding an optimized set of weights. Binary representation of the connection weights is one of the earlier works in this area (Whitley, 1989; Caudell and Dolan, 1989; Whitley et al., 1990; Cantu-Paz and Kamath, 2005). However, some of researchers have used suitable real encoding EAs (Montana and Davis, 1989; Deb et al., 2002). Since evolution startegy (ES) and evolutionary programming (EP) algorithms are well-suited for real vector optimization, they have been employed in this area (Fogel et al., 1995; Heidrich-Meisner and Igel, 2009). An ANN training algorithm is always a well-known benchmark for new approaches developed in EA's research area; the generalized generation gap parent-centric recombinatin (G3PCX) algorithm introduced by Deb et al. (2002), for example, has been used in the empirical study performed by Cantu-Paz and Kamath (2005) to compared with a binary encoded genetic algorithm (GA). In order to improve the ANN training process, a local search may be embedded into an EA (Topchy and Lebedko, 1997).

1.2. ANN designing algorithms

The architecture of an ANN is of great importance because it affects the learning capacity and generalization capability of the ANN. Gradient-based search approaches such as constructive and destructive algorithms may be used in the automatic design of the architecture of ANNs (Frean, 1990; Sietsma and Dow, 1991). Nevertheless, the main drawback of such methods is that they are quite susceptible to fall in local optima (Angeline et al., 1994).

In the evolutionary design of the architecture, there are two approaches for representing solutions. In the first approach, called direct encoding, all of the network architecture details are encoded in a chromosome. Assuming the ANN as a directed graph and using an adjacent matrix for representing its genotype is a common direct encoding method. In this way, each entry is a binary number representing the presence or absence of a connection between two nodes (Miller et al., 1989; Kitano, 1990; Belew et al., 1991; Cantu-Paz and Kamath, 2005). This representation can also be employed to prune the connections of a network trained with full connectivity pattern (Reed, 1993). The other approach is indirect encoding, where only some characteristics of the architecture of an ANN are encoded in a chromosome. In the indirect encoding approach, some aspects of the destination network and/or network generation (mapping process) are predefined. For example, if we know that a fully connected architecture is suitable for our problem, it is sufficient to only encode the number of hidden layers, the number of neurons in each layer and the parameters of BP algorithm in a chromosome. The knowledge added to the algorithm reduces the search space and usually leads to a compact encoding.

There are several types of indirect encoding in the literature. Kitano (1990) has introduced a grammar based indirect representation encoding production rules in a chromosome instead of an adjacent matrix of the network. In the genotype to phenotype mapping, the production rules are decoded to generate the adjacent matrix; transforming the matrix to the corresponding network is then straightforward. Compact genotype is the main advantage of this method. Siddiqi and Lucas (1998) have shown that direct encoding can be at least as good as the Kitano's method. Fractal representation inspired by regularity, symmetry and self-similarity of live organisms is another indirect encoding scheme that may be more plausible than other encoding schemes (Merrill and Port, 1991). Gruau (1993, 1994) has introduced a cellular encoding as an indirect representation that is motivated by cell division in biology.

Cantu-Paz and Kamath (2005) have empirically evaluated, in addition to ENN methods for ANN training, also various ENN methods for feature selection (Kohavi and John, 1997; Yang and Honavar, 1998) and ANN designing on classification problems. Yang and Chen (2012) have proposed an evolutionary constructive and pruning algorithm to design the network topology, where its weights are optimized through BP. Furthermore, Soltanian et al. (2013) have applied a grammatical evolution (GE) algorithm (Ryan et al., 1998), called GE-BP, to design the architecture of an ANN, where the BP algorithm is used for the network evaluations against training data.

1.3. Simultaneous evolution of the ANN topology and weights

Yao and Liu (1997) have developed a TWEANN method based on EP, called EPNet, for simultaneously evolving the architecture and weights of an ANN, in which no crossover operation is utilized. NeuroEvolution of Augmenting Topologies (NEAT), presented by Stanley and Miikkulainen (2002), is another successful system in this area. A hypercube based NEAT with an indirect encoding has also been introduced by Stanley et al. (2009). Compact genotype, input as well as output scalability, and utilizing the geometry of the problem domain are the most important advantages of their system. Motsinger et al. (2006) and Tsoulos et al. (2008) have used GE for construction and training ANNs with one hidden layer. Castellani and Rowlands (2009) have developed a TWEANN method for recognizing wood veneer defects, and reported that there are no differences in accuracy between architectures using one and two hidden layers. Rivero et al. (2010) have applied genetic programming (GP) to design and train a feedforward ANN with any arbitrary architecture, and Oong and Isa (2011) have presented a global-local balanced GA to simultaneously design and train an arbitrary connected feedforward ANN. More recently, Castellani (2013) has compared evolutionary ANN designing and whole ANN development algorithms with classical feature selection and designing methods.

However, there are also other types of ANNs and EAs combination in the literature. Evolving the neuron transfer functions (Stork et al., 1990), the learning rule and parameters of BP (Bengio et al., 1990; Baxter, 1992), and ANN ensembles (Yao and Islam, 2008; Huanhuan and Yao, 2010; Felice and Yao, 2011; Donate et al., 2013; Ghazikhani et al., 2013) are other research lines in the ANN evolution. ANN ensembles originate from the idea of divide-and-conquer algorithms. Experiments have shown that EAs are good choice to automatically divide the problem space, and thus, ENN ensembles have attracted many researchers attention in the literature (Yao and Islam, 2008).

The major drawback of the ANN training algorithms described in Section 1.1 is the expert's effort needed for designing the network topology. The major disadvantage of the ANN designing algorithms described in Section 1.2 is that the search space is complex and noisy because the fitness assigned to a given architecture is dependent to the learning method. The methods described in Section 1.3 are free from these problems. However, most of them suffer from inability of using the problem domain knowledge, while some of them are extremely dependent to the expert.

As mentioned earlier, the TWEANN method proposed by Tsoulos et al. (2008) uses GE for designing the network topology as well as optimizing its weights. That is, their method encodes

both the network topology and its weights using a context free grammar (CFG) in Backus–Naur form (BNF). The GE approach, firstly introduced by Ryan et al. (1998), has been applied to successfully solve a range of problems (see, e.g., Ryan et al., 2002; O'Neill and Ryan, 2003; Motsinger et al., 2006; Chen et al., 2008; Chen and Wang, 2010). As stated by Tsoulos et al. (2008), the use of GE to evolve ANNs has the benefit of allowing easy shaping of the resulting search, in addition to leading to a compact encoding. However, GE does not seem to be much suitable for real vector optimization, i.e., for optimizing the connection weights, and may cause some problems such as high destructiveness of variation operators, thus destroying information evolved during the search. Accordingly, the algorithm introduced by Soltanian et al. (2013) uses GE only for designing the network topology, while its weights are optimized through BP. Nevertheless, this type of combination leads to a complex and noisy landscape, the major disadvantage of ANN designing algorithms. In order to overcome the drawbacks of the methods of Tsoulos et al. (2008) and Soltanian et al. (2013), this paper proposes a new TWEANN method, called GEGA, in which GE is adopted to design the network topology while GA is incorporated for weight adaptation. Indeed, GEGA employs a hybrid of direct and indirect encodings: a grammatical encoding to indirectly represent the topology and a real encoding to directly represent the connection weights with the aim of achieving a better global–local search in the space of weights. To the best of our knowledge, this is the first paper of applying such a combination to the whole development of ANNs. It is noteworthy that, although there are some other algorithms for real vector optimization (such as ES and EP algorithms), GA is well-matched for weight adaptation due to the fact that GE is mainly a GA but with a grammatical encoding, and consequently, the similarities between GA and GE allow us to combine them in one algorithm. Like the methods of Tsoulos et al. (2008) and Soltanian et al. (2013), GEGA is capable of generating any feedforward ANN with one hidden layer. GEGA can utilize expert knowledge about the problem on hand in order to more efficiently search in the infinite space of topologies, although it needs to invest a minimal expert's effort for customization. Moreover, another contribution of this paper is to propose a novel adaptive penalty approach that encourages smaller topologies. This allows GEGA to generate much simpler ANNs that have better generalization ability and also are easy to implement. To evaluate the performance of GEGA, extensive experiments are performed on real world classification datasets, and the results are statistically compared against other well-known and state-of-the-art algorithms in the literature.

The rest of the paper is organized as follows. The next section describes the proposed algorithm. The contributions of some GEGA components and computational results are analyzed in Section 3, followed by Section 4 discussing the characteristics of GEGA. Finally, Section 5 concludes the paper.

2. Proposed algorithm

The topology and the connection weights of an ANN are simultaneously determined by GEGA. The network topology includes the neurons in the input layer, i.e., the selected features, the neurons in the hidden layer, and the connectivity among the neurons. Since it is expected that simpler ANNs lead to more generalization ability, a penalty approach is proposed to simplify ANNs through the evolution process. The general structure of GEGA whose output is a feedforward ANN is shown in Fig. 1. The representation, genetic operators and the proposed penalty method are then described in details in the following subsections.

2.1. Representation

Designing an appropriate genotype representation is the first step in applying an EA to a particular problem. In GEGA, a hybrid of direct and indirect encodings is employed in which each chromosome includes two parts: one for the network topology and the other for its weights. The topology is indirectly represented by a vector of integer numbers in the range of [0, 255]. The number of genes in the topology part of each chromosome in the initial population is set to be equal to 100. This part is then decoded to the corresponding topology by a generation grammar; some characteristics of the network are pre-determined in the

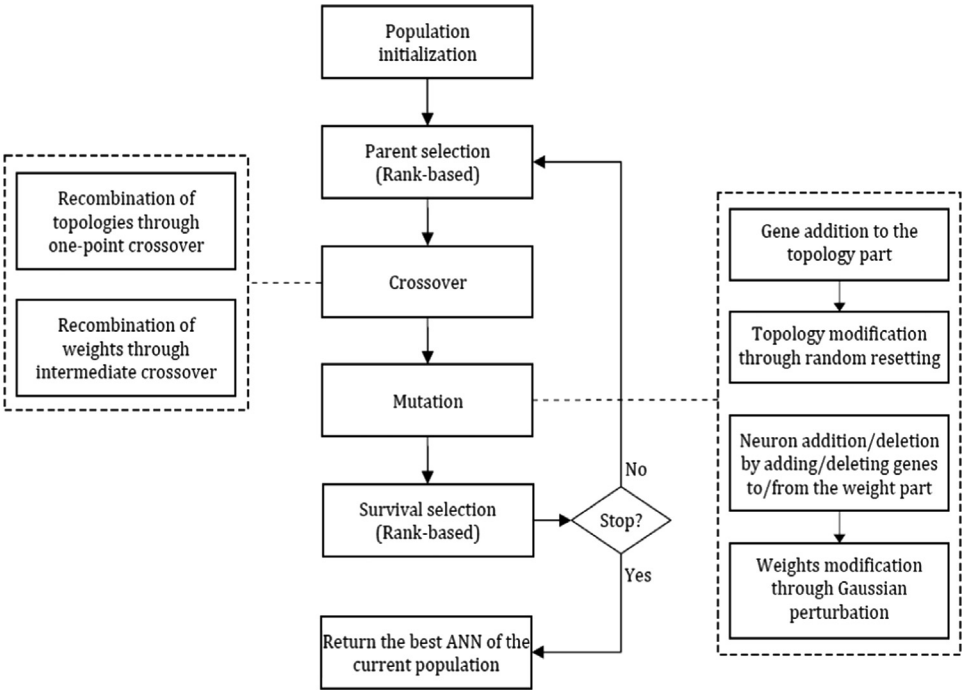

Fig. 1. The structure of the proposed algorithm.

generation grammar. However, the connection weights are directly encoded as a vector of real numbers. If there is m input neurons and h hidden neurons, the number of connections is at most $h(m+2)$ because each hidden neuron has, respectively, one output connection, (at most) m connections with the input neurons, and one connection for the bias. Accordingly, the number of genes in the weight part of a chromosome in the initial population is set to be equal to $int(m+2)$, where int is an integer number randomly selected between 1 and 10 for each chromosome. It is noted that the number of genes in the weight part as well as in the topology part of chromosomes may change during the evolution process.

To generate the network topology, a BNF grammar is employed. BNF grammars are widely used as tools to describe programming languages and expressions. BNF is a notation for expressing a CFG by means of production rules, and has been used as the main grammar description tool of languages by the founders of GE (Ryan et al., 1998; O'Neill and Ryan, 2001). A language's BNF grammar contains some symbols (terminal symbols) that can be presented in a valid sentence of the language, and some non-terminal symbols used in the generation process of a valid sentence. A BNF grammar is shown by a tuple $\{N, T, P, S\}$, where N denotes the set of non-terminal symbols, T the set of terminal symbols, P the production rules and S the start symbol beginning the generation process. The proposed CFG, which is able to generate any feedforward ANN with one hidden layer, is shown in Fig. 2, where x_1, …, x_n are input features, w stands for weights, and sig denotes the sigmoid activation function for hidden neurons (of course, for the output neuron as well), calculated as $sig(x) = 1/(1+e^{-x})$. The sigmoid activation function returning values between 0 and 1 is frequently used in ANNs, although one can easily embed a different nonlinear function in the proposed CFG for neuron activation. Mapping a chromosome to the corresponding ANN is done by generating the network topology through the grammar, and then, assigning the weights to the network's connections.

Predecessor	::=	Productions	Index of Product
<S>	::=	<Node>	0
		|<Node> + <S>	1
<Node>	::=	w * sig(<Sum> + w)	0
<Sum>	::=	w * <xxList>	0
		|w * <xxList> + <Sum>	1
<xxList>	::=	x_1	0
		|x_2	1
		|…	…
		|x_n	n

Fig. 2. The proposed CFG to generate the ANN topology.

Weight part	5.2	5.9	4.3	3.8	4.0	1.5	0.2	1.0
Topology part	111	47	58	12	39	28	26	125

Fig. 3. An example of chromosome encoding.

For a problem with two input features x_1 and x_2 and one output O, Fig. 3 illustrates an example of a chromosome representation. The steps of the generation process are presented in Fig. 4. The start symbol $<S>$ begins the process. Since it is a non-terminal symbol, it must fire a production rule; in the proposed grammar, $<S>$ has two production rules $<Node>$ and $<Node>+<S>$ to be fired. To determine which one is fired, the remainder of the first entry of the topology part when divided by the number of production rules is used; the remainder is 1 and so the production rule with index 1, that is, $<Node>+<S>$, is fired. By continuing in this fashion, until there is no non-terminal symbol in the generated sentence, the generation process terminates successfully. It is noted that, by means of wrapping (Ryan et al., 1998), genes can be used multiple times. In other words, the integers from the beginning of the topology part of a chromosome are used again when the end of the vector is reached. However, after a prespecified number of wraps, if the sentence is not valid, the chromosome is called an invalid chromosome.

From the generated topology string shown in Fig. 4, the resulting network has two hidden neurons; the first hidden neuron, H1, has two connections with x_1 and x_2, while the second one, H2, has only one connection with x_2. If the weights given in Fig. 3 are inserted in the generated sentence, the following string, which is then the input to the output neuron of the ANN shown in Fig. 5, is yielded:

$$5.2\ sig\,(5.9\times 1+4.3x2+3.8)+4.0\ sig\,(0.2\times 2+1.0)$$

Since H2 has no connection with x_1, the corresponding weight indicated in Fig. 5 with a grey colored background is ignored; recall that the positions of the weights corresponding to each hidden neuron in the weight part of a chromosome are fixed.

Finally, we would like to mention that when the network resulting from the topology part of a chromosome has a number of hidden neurons (say ht) that is not equal to the number of neurons served by the weight part (say hw), the smaller one of ht and hw will dictate the number of hidden neurons. Therefore, if ht is smaller than hw, the additional genes in the weight part are ignored, otherwise zero weight is assigned to each connection associated with the additional hidden neurons of the resulting network, thus ignoring those neurons.

Expression	Gene value	Number of productions	Index of the production selected
<S>	111	2	111 mod 2 = 1
<Node>+<S>	---	1	0
w*sig(<Sum>+w)+<S>	47	2	47 mod 2 = 1
w*sig(w*<xxList>+<Sum>+w)+<S>	58	2	58 mod 2 = 0
w*sig(w*x1+<Sum>+w)+<S>	12	2	12 mod 2 = 0
w*sig(w*x1+w*<xxList>+w)+<S>	39	2	39 mod 2 = 1
w*sig(w*x1+w*x2+w)+<S>	28	2	28 mod 2 = 0
w*sig(w*x1+w*x2+w)+<Node>	---	1	0
w*sig(w*x1+w*x2+w)+w*sig(<Sum>+w)	26	2	26 mod 2 = 0
w*sig(w*x1+w*x2+w)+w*sig(w*<xxList>+w)	125	2	125 mod 2 = 1
w*sig(w*x1+w*x2+w)+w*sig(w*x2+w)			← Generated topology string

Fig. 4. Decoding process of the topology part of Fig. 3 through the proposed CFG.

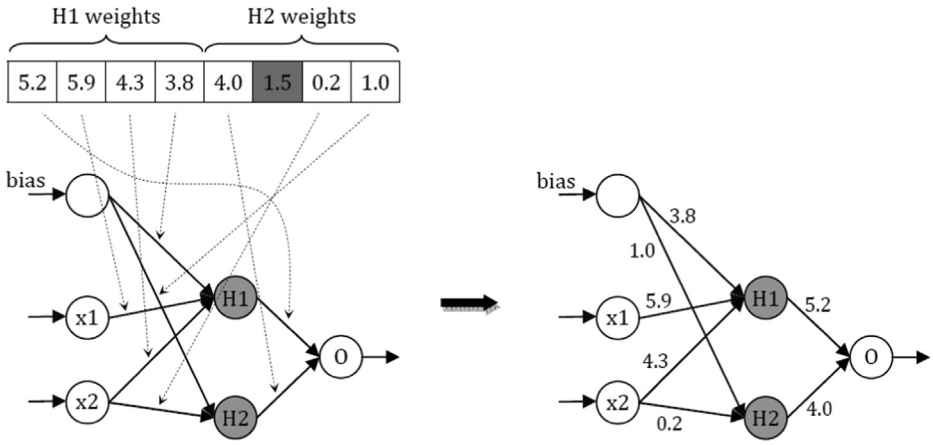

Fig. 5. The ANN corresponding to the chromosome shown in Fig. 3.

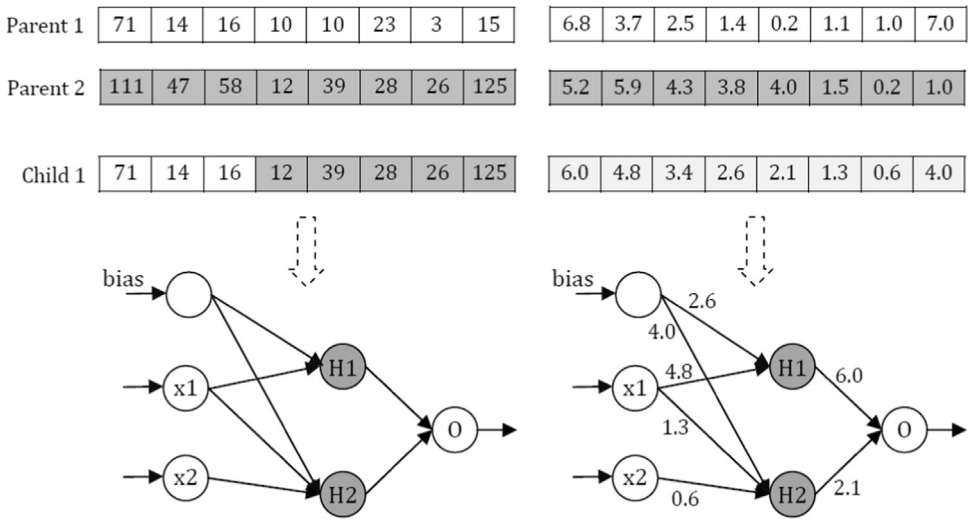

Fig. 6. An example of the crossover operation.

2.2. Crossover operator

By recombining the gene-codes of two parents, crossover as the main genetic operator produces two solutions in as yet unvisited regions of the search space. For each pair of parents, the crossover operator which is a combination of two different crossovers is applied according to a crossover probability P_c. Among many possible crossovers, the one-point crossover, which is the most common crossover operator in the literature and is faster than other operators, is executed on the topology part of the two chromosomes, while the intermediate recombination is applied to the corresponding weight part working by taking the average of the two parental alleles for each gene. Fig. 6 illustrates an example of the crossover operation; the second offspring is created in the same way with the parental roles reversed.

2.3. Mutation operator

After the crossover process, the mutation operator performed by varying the gene values of a chromosome is applied to each resulting offspring. It aims at preventing the algorithm from trapping in local optima by exploring new solution spaces. The proposed algorithm utilizes four types of mutation operators. First, a new gene is added to the topology part of the chromosome with a probability of P_m; the gene position as well as its value is randomly chosen. For every gene in the topology part, with probability P_m a new value is then chosen at random from the range [0, 255]. Moreover, with probability $P_{m/2}$ a neuron is added or deleted to/from the weight part of the chromosome by adding/deleting $(m+2)$ genes. Then, for every gene in the weight part, an amount randomly drawn from a Gaussian distribution with mean zero and a standard deviation, which is selected at random from {0.2, 0.5, 1, 2, 5}, is added to the current gene value. Fig. 7 illustrates an example of the mutation operation.

2.4. Selection mechanisms

The parent selection mechanism is applied to the current population with a size of μ to select, with replacement, a number of parents equal to μ. The parents then undergo the crossover and mutation operations. Among many possible selection techniques, the rank-based selection mechanism (Eiben and Smith, 2003) is adopted in this paper to select individuals for reproduction. This mechanism allows a chromosome to become parent with a probability that is based on its rank in the population, rather than based on its absolute fitness value. All individuals are first sorted according to their fitness. If the fittest individual is assigned rank μ and the worst rank 1, the probability of selecting individual with rank j is then calculated linearly as follows:

$$\Pr(j) = \frac{(2-s)}{\mu} + \frac{2(j-1)(s-1)}{\mu(\mu-1)} \quad (1)$$

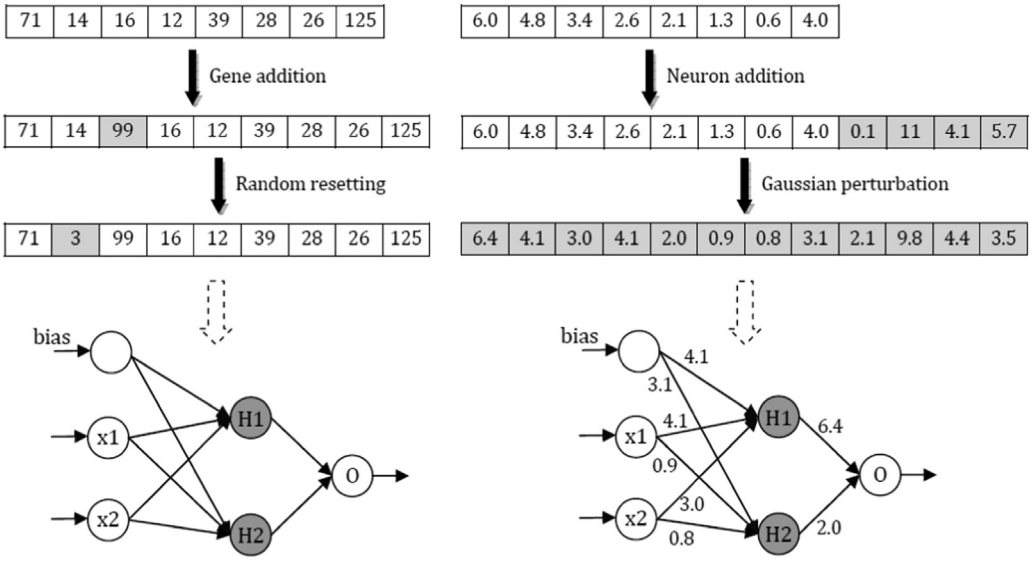

Fig. 7. An example of the mutation operation.

where s is a parameter between 1 and 2 determining the selection pressure. Obviously, this selection mechanism preserves a constant selection pressure in the evolutionary search, thus avoiding the drawbacks encountered by fitness proportionate selection.

Furthermore, after having created μ offspring of the parents, the survivor selection mechanism is applied to select μ individuals among both the current population and their offspring to form the next generation. The above rank-based selection mechanism but with an elitist strategy is adopted to select individuals for survival. Elitism is frequently used in addition to other selection mechanisms in order to avoid losing the best found individuals during generations. First, a predefined percentage P_{elit} of the new population is filled by the best individuals, and then, the rest by the result of the rank-based selection mechanism.

2.5. Fitness evaluation

An appropriate measure to evaluate the fitness of an ANN is the mean squared error (MSE), where the error per pattern is the difference between the ANN's output and the desired (target) output; here, low MSE means high fitness. Let r be the number of training patterns. The MSE for individual i (which is an ANN) is then calculated as follows:

$$MSE(i) = \frac{\sum_{z=1}^{r} (d_z - o_{iz})^2}{r} \tag{2}$$

where d_z and o_{iz} are the desired output and the output of individual i for pattern z, respectively.

However, GEGA aims at providing an ANN that has not only low MSE on the training set, but also high generalization on unseen data. As the generalization ability of an ANN may decrease because of overfitting problems, a novel adaptive penalty approach is incorporated into GEGA to reach much simpler ANNs, thus leading to more generalization ability. In addition to having good generalization capability, simple ANNs are easy to implement and also provide better opportunities to extract knowledge. Accordingly, in order to reward parsimony, the error (i.e., the training fitness function to be minimized in the evolution process) for individual i is defined as follows:

$$E(i) = MSE(i) + p^{co} h(i) \tag{3}$$

where $h(i)$ is the number of hidden neurons of individual i and p^{co} the penalty coefficient.

Rivero et al. (2010) have tested the values 0, 0.00001, 0.0001, 0.001, 0.01 and 0.1 for the penalty coefficient, and claimed that 0.00001 returns good results for most of the problems. However, determining an appropriate coefficient is a problem-dependent, time-consuming process that needs to try-and-test different values. Besides, in the literature on constrained optimization, there are various automatic techniques to determine a penalty coefficient. Nevertheless, such methods are appropriate when the boundary between the feasible and infeasible regions is known; whereas there is no exact upper bound on the number of hidden neurons when evolving an ANN.

In this paper, however, an adaptive method exploiting the population information is proposed to determine the penalty coefficient so that it prevents the algorithm from unnecessarily producing complex ANNs, i.e., ANNs with a large number of hidden neurons. Let $\overline{MSE}$ and $\overline{h}$ be, respectively, the average MSE and the average number of hidden neurons over the current population. The penalty coefficient is then calculated at each generation as follows:

$$p^{co} = q(\overline{h})/\overline{MSE} \tag{4}$$

where $q(\overline{h})$ is an increasing function in $\overline{h}$.

The rationale behind this definition is to allow a higher penalty if the algorithm reaches more complex ANNs. More specifically, whenever it reaches ANNs with a small number of hidden neurons but with a high MSE, a low penalty then tends to generate more complex ANNs in subsequent generations with the hope of improving the MSE. Obviously, in the beginning of the evolution, the algorithm is expected to attain such ANNs due to the encoding-specific process employed for initializing the population. Hence, incorporating the proposed penalty approach into GEGA, which most likely starts with small network structures, has an advantage of forcing it to incrementally explore more complex structures by decreasing the penalty coefficient. On the other hand, whenever the algorithm reaches ANNs with a large number of hidden neurons having a low MSE, a high penalty then tends to generate simpler ANNs with the same or even a lower MSE in subsequent generations. Note that as the evolutionary process proceeds, the algorithm is expected to attain ANNs having a low MSE. So, another advantage of the penalty approach is that it encourages smaller topologies leading to more generalization ability when needed.

Furthermore, in order to make the contribution of the penalty to the fitness no more than that of the MSE, following preliminary

experiments the increasing function $q(\bar{h})$ is defined by

$$q(\bar{h}) = \frac{\bar{h}}{h_{\max}^{1.4}} \quad (5)$$

where $h_{\max}$, an approximate upper bound on the number of hidden neurons, is set to be equal to $r-1$, as a three-layered feedforward ANN with $r-1$ hidden neurons (with sigmoid activation functions) can give any r input-target relations exactly (Tamura and Tateishi, 1997). It is noted that $(r-1)^{1.4}$ is used to scale the contribution of the penalty term. Taking Eqs. (4) and (5) into account, the error stated in Eq. (3) can then be rewritten as

$$E(i) = MSE(i) + \frac{\bar{h}}{(r-1)^{1.4}\overline{MSE}} h(i) \quad (6)$$

3. Performance evaluation

To validate GEGA and make comparisons with other ENN algorithms, seven well-known classification problems used by most methods are chosen from the University of California at Irvine (UCI) repository of machine learning databases (Blake and Merz., 1998). Prior to conducting experiments, however, the numeric features in the data are linearly normalized to the interval $[-1, +1]$, the nominal features (except for the class labels) are encoded with the 1-in-C coding, where for a C-class problem one of the C outputs is set to 1 and the others to 0, and the binary values are encoded as a single 0 or 1. Such a preprocessing phase is the usual approach used for classification purposes (see, e.g., Cantu-Paz and Kamath, 2005). Table 1 presents the brief description of the datasets after preprocessing the data; for each dataset, the number of input features, classes as well as patterns is given.

Due to the stochastic nature of the proposed algorithm, it must be run multiple times for each dataset, and then, comparisons with other ENN algorithms are made by performing proper statistical tests. When comparing classification algorithms, the most common technique is k-fold cross-validation to measure the accuracy of the algorithms and then the use of t-tests to confirm if the results are significantly different (Cantu-Paz and Kamath, 2005; Rivero et al., 2010). In k-fold cross-validation, the data D is randomly divided into k non-overlapping sets $D_1, D_2, \ldots, D_k$. The algorithm runs k times; in each run i $(1 \le i \le k)$, the algorithm is trained with $D \backslash D_i$ sets and tested on D_i. In this paper, all experiments are performed in 2-fold cross-validation, where the data is divided in two halves and the algorithm runs two times; in each fold, the algorithm is trained with one half and tested on the other half, and vice versa. More specifically, in each run of the algorithm, the training set is used to calculate the error stated in Eq. (6) during the evolutionary search, while the test set is used to measure the classification accuracy of the best ANN found in the search process as an estimation of its generalization performance. Moreover, each experiment is repeated 5 times, where the data division in each experiment is done independently, thus having 10 runs of the algorithm. This method of experiment, suggested by Dietterich (1998), is known as 5×2 cross-validation and has been used by many other ENN algorithms, including those with which GEGA is compared.

Furthermore, to assess the significance of the differences between the results obtained by GEGA and other ENN methods, t-tests are performed with a confidence level of 95% (i.e., a p-value under 0.05). To statistically compare the performances of ENN methods, the two following criteria are used: the average classification accuracy as the primary criterion and the average number of hidden neurons as the secondary one. This latter criterion is important since simple ANNs are easy to implement and provide better opportunities to extract knowledge, in addition to leading to high performance in the generalization measured by the classification accuracy. In a similar way as in Castellani (2013), if there is no statistically significant difference between the performances of two algorithms on a given dataset according to the primary criterion, the secondary criterion is then taken into account. If again there is no statistically significant difference, the two algorithms are said to perform equally on the dataset, and both are awarded 1 point. But, in the case where there is statistically significant difference between their performances according to the primary or secondary criteria, the algorithm performing better is awarded 2 points and the other zero point. The overall performance of each algorithm is then calculated by summing all the points achieved in the pairwise comparisons on all the datasets. Note that for a given dataset an algorithm can obtain at least one point, since it is compared with itself as well.

Table 1
Description of the datasets used in this paper.

Dataset	No. of inputs	No. of classes	No. of patterns
Breast Cancer	9	2	699
Credit-German	62	2	1000
Heart-Cleveland	26	2	303
Ionosphere	34	2	351
Iris	4	3	150
Sonar	60	2	208
Wine	13	3	178

Table 2
The parameter settings of GEGA.

Population size (μ)	Max no. of generations	Crossover rate (P_c)	Probability of mutation (P_m)	s (in Eq. (1))	P_{elit}
500	500	0.9	0.05	1.8	2%

In the remainder of this section, we first analyze the contributions of some GEGA components and then compare the performance of GEGA against many other well-known and state-of-the-art ENN algorithms in the literature. However, before conducting formal experiments, it is necessary to set the numeric parameters involved in GEGA. We have empirically determined these parameters; in a preliminary experiment, different combinations of the parameter values have been tested. The settings observed to be superior are given in Table 2. In the experiment, it has been observed that the performance of GEGA is not too sensitive to make small changes in the values chosen for the numeric parameters (in the following, some experimental results concerning other settings of one of the parameters, P_m, will be shown, although relatively large changes are made in the value given in Table 2).

3.1. Performance analysis of GEGA components

In this subsection, the contributions of the mutation operator and the adaptive penalty approach to the performance of GEGA are analyzed and validated through experiments. Following Rivero et al. (2010), to evaluate different settings of the proposed algorithm, the MSE, rather than the classification accuracy, of the generated ANNs on test sets in 5×2 cross-validation is measured.

3.1.1. The contribution of the mutation operator

Mutation is applied to one solution and produces one new solution. Thus, by creating the diversity within the population, it can prevent the algorithm from trapping in local optima. However, if the probability of mutation is selected to be at a high level, mutation

Table 3
The results for different settings of the proposed mutation operator.

Dataset	Breast Cancer	Heart-Cleveland	Ionosphere	Iris	Wine	*Average*
P_m=0.01	0.03321	0.15601	0.08829	0.04879	0.06448	0.078156
P_m=0.02	0.03218	0.14499	0.10629	0.04879	0.05277	0.077004
P_m=0.04	0.03122	0.15384	0.09107	0.03949	0.05071	0.073266
P_m=0.05	0.03042	**0.13356**	0.09034	0.03871	0.05137	**0.068880**
P_m=0.07	0.03089	0.14335	0.10239	0.04035	**0.03894**	0.071184
P_m=0.08	0.02918	0.14200	0.09083	0.04426	0.05161	0.071576
P_m=0.10	0.02994	0.14200	0.09083	0.03584	0.05769	0.071260
Case 1	0.03400	0.14695	**0.08278**	**0.03558**	0.07001	0.073864
Case 2	**0.02797**	0.18279	0.13827	0.03605	0.09381	0.095778
Case 3	0.04401	0.13923	0.08548	0.03571	0.08681	0.078248
Case 4	0.04689	0.18229	0.10746	0.07321	0.31448	0.144866
Case 5	0.20468	0.24399	0.22523	0.60676	0.56859	0.369850

would destroy information evolved during the search, that is, the solutions resulting from this operation may inherit nothing from their parents. In other words, the probability of mutation can have a significant effect on the algorithm's performance. Accordingly, experiments with P_m=0.01, 0.02, 0.04, 0.05, 0.07, 0.08, and 0.10 are performed. The results concerning average MSE of the generated ANNs for five randomly selected datasets are summarized in Table 3, where best values are indicated in boldface. To show the impact of each of the four components of the mutation operator, Table 3 also gives the results of GEGA for the following cases:

Case 1. There is no gene addition to the topology part,

Case 2. There is no topology modification,

Case 3. There is no neuron addition/deletion to/from the weight part,

Case 4. There is no weight modification,

Case 5. There is no mutation at all.

It can be seen from Table 3 that P_m=0.05 is a suitable choice for most of the datasets and provides the best performance on average, while GEGA has the worst performance in case 5, i.e., when mutation is not applied at all. It is noted that the results reported in Table 3 for cases 1–4 are concerned with P_m=0.05. Moreover, the results indicate that all the components of the mutation operator have significant impact on the performance of GEGA, especially weights modification through Gaussian perturbation and topology modification through random resetting.

3.1.2. The contribution of the adaptive penalty method

To evaluate the contribution of the proposed adaptive penalty approach, experiments with four other alternatives are performed. One alternative is the case in which no penalty approach is incorporated into GEGA at all, i.e., when the penalty coefficient p^{co} is set to 0. As other alternatives, the static penalty method suggested by Rivero et al. (2010) is incorporated into GEGA in which the penalty coefficient p^{co} takes the values of 0.00001 (as recommended by the authors), 0.001, and 0.0001. The results obtained for five randomly selected datasets are summarized in Table 4, which gives for each dataset and each scenario (including the proposed penalty approach and the four other alternatives) the average MSE and number of hidden neurons of the generated ANNs.

It can be seen from Table 4 that the proposed penalty approach results in a low MSE for most of the datasets and provides the best performance on average, while GEGA has the worst performance when $p^{co}=0$. Moreover, in terms of the number of hidden neurons, the proposed approach leads to the best results for almost all datasets, while $p^{co}=0$, and 0.00001 provide the worst performance on average.

For the Breast Cancer dataset, Fig. 8 illustrates the effect of the proposed penalty approach on the evolution of GEGA in terms of the average MSE and the average number of hidden neurons in the population, that is, $\overline{MSE}$ and $\overline{h}$. In order to track the progress of the evolutionary process, the $\overline{MSE}$ values measured on the training set are plotted in Fig. 8.

As shown in Fig. 8, in the beginning of the evolutionary process, GEGA reaches ANNs with a high MSE and a small number of hidden neurons. Allowing a low penalty then tends to incrementally explore greater topologies in the subsequent generations, thus generating more complex ANNs with a lower MSE. Whenever the algorithm reaches ANNs with a large number of hidden neurons having a low MSE, allowing a high penalty then tends to generate simpler ANNs with a lower or the same MSE in the subsequent generations. As can be seen in Fig. 8, GEGA with the proposed adaptive penalty approach is able to find ANNs with a low MSE and a small number of hidden neurons after about 50 generations.

3.2. Comparison with other ENN methods

In this subsection, computational experiments are conducted to evaluate the performance of GEGA and make comparisons with many other alternative ENN methods. The average and standard deviation of the classification accuracies of the generated ANNs on test sets in 5×2 cross-validation, as well as the average and standard deviation of the number of hidden neurons of the generated ANNs are measured; the standard deviation may be used as a measure of stability.

Due to the fact that the CPU time needed to solve a problem is affected by a lot of factors such as computer platforms and programming skills, it is not a practicable measure to use when comparing results with any other previously reported ones. So, to measure computational time, the computational effort which is a more universal measure is used. Following Rivero et al. (2010), the computational effort is calculated for each algorithm by multiplying the effort needed in evaluating every individual with the population size and the number of generations. The effort needed in evaluating an individual is measured using the number of times training patterns is passed through the corresponding ANN. After calculating the computational efforts of alternative ENN methods, to conduct a fair comparison, GEGA is allowed to run for almost the same computational effort.

3.2.1. Comparison with training algorithms

The first category of the alternative ENN methods consists of ANN training algorithms evolving only the connection weights. G3PCX (Deb et al., 2002) and GA (Cantu-Paz and Kamath, 2005) are two methods that fall in this category. G3PCX and GA use real and binary representations of the connection weights, respectively, to train three-layered, fully connected feedforward ANNs. G3PCX and GA require less computational efforts, and consequently, the population size of GEGA has been set in this experiment to 100, instead of 500. Table 5 provides a comparison between the results of GEGA and those of G3PCX and GA taken from Cantu-Paz and Kamath (2005). The best results concerning average classification accuracy (in percentage) and average number of hidden neurons are indicated in boldface. The computational efforts of G3PCX as well as GA are calculated based on the corresponding results reported in Cantu-Paz and Kamath (2005).

It can be seen from Table 5 that GEGA results in a high classification accuracy for most of the datasets and provides the best performance on average. Moreover, in terms of the number of

Table 4
Comparison of the proposed penalty approach with four alternatives.

Dataset		$p^{co}=0$	$p^{co}=0.001$	$p^{co}=0.0001$	$p^{co}=0.00001$	Proposed approach
Breast Cancer	MSE	0.03425	0.03089	0.03063	0.03196	**0.03042**
	No. of hidden neurons	4.8	3.3	2.9	5.1	**2.0**
Heart-Cleveland	MSE	0.14190	0.14469	0.14092	0.14349	**0.13356**
	No. of hidden neurons	5.0	2.7	2.5	2.6	**2.4**
Iris	MSE	0.03954	0.04024	0.04162	0.03877	**0.03871**
	No. of hidden neurons	4.6	3.0	4.5	4.8	**2.1**
Sonar	MSE	0.20169	0.19438	0.19587	**0.18673**	0.19346
	No. of hidden neurons	3.7	3.5	4.1	4.2	**2.1**
Wine	MSE	0.07428	0.05494	**0.04777**	0.05632	0.05137
	No. of hidden neurons	2.8	3.1	**2.6**	3.3	2.7
Average	*MSE*	*0.098332*	*0.093028*	*0.091362*	*0.091454*	***0.089504***
	No. of hidden neurons	*4.2*	*3.1*	*3.3*	*4.0*	***2.3***

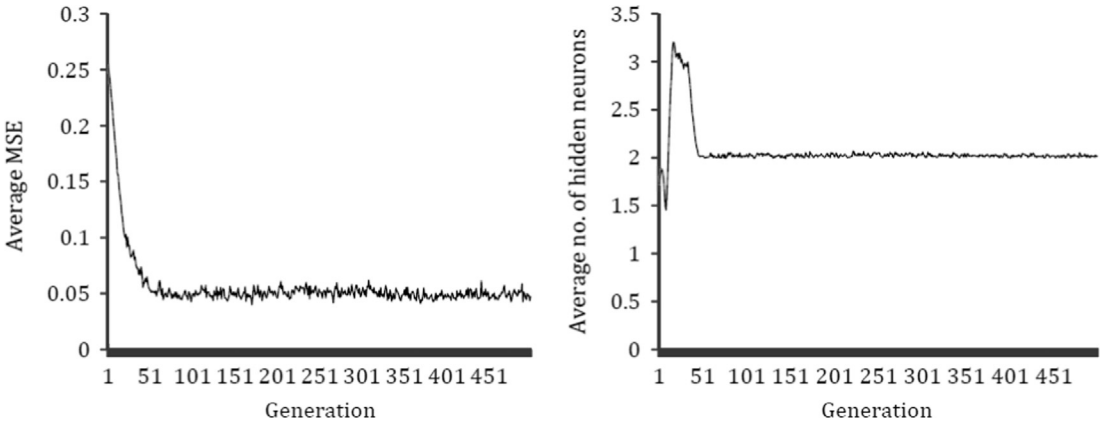

Fig. 8. The effect of the proposed penalty method on the evolution of GEGA: Breast Cancer dataset.

Table 5
Comparison between GEGA and ANN training algorithms.

Dataset	G3PCX		GA		GEGA	
Breast Cancer	**98.94**[a]	2.35[b]	98.88	0.32	96.61	0.81
	5[c]	0[d]	5	0	**3.8**	1.0
	11500[e]		9200		9200	
Credit-German	30.00	1.46	**70.94**	1.49	70.33	2.61
	10	0	10	0	**4.8**	1.0
	39450		31500		31500	
Heart-Cleveland	**90.42**	2.12	87.72	3.42	79.55	3.65
	5	0	5	0	**2.9**	1.3
	19000		15200		15200	
Ionosphere	64.10	2.04	74.10	1.94	**86.94**	2.83
	10	0	10	0	**5.2**	1.7
	29350		23500		23500	
Iris	89.73	11.70	88.67	6.09	**96.13**	1.59
	5	0	5	0	**2.7**	1.1
	10250		8200		10250	
Sonar	67.40	5.44	**73.65**	2.55	72.01	4.23
	10	0	10	0	**3.9**	1.0
	38800		31000		31000	
Wine	84.94	11.46	**92.47**	4.55	**92.47**	2.20
	5	0	5	0	**2.1**	0.3
	16350		13000		13000	
Average	*75.08*	*5.22*	*83.78*	*2.91*	***84.86***	*2.56*
	7.1	*0*	*7.1*	*0*	***3.6***	*1.1*
	23529		*18800*		*19093*	

[a] Average classification accuracy.
[b] Standard deviation of the classification accuracy.
[c] Average number of hidden neurons.
[d] Standard deviation of the number of hidden neurons.
[e] Computational effort.

hidden neurons, GEGA leads to the best results for all datasets. Table 6 statistically compares the performances of GEGA and the two alternative ENN methods by means of the mentioned approach.

Table 6
Statistical comparison between GEGA and ANN training algorithms.

Dataset	G3PCX	GA	GEGA
Breast Cancer	**4**	**4**	1
Credit-German	1	3	**5**
Heart-Cleveland	**4**	**4**	1
Ionosphere	1	3	**5**
Iris	2	2	**5**
Sonar	1	3	**5**
Wine	2	2	**5**
Overall performance	*15*	*21*	***27***

As seen, GEGA evolving both the topology and the connection weights of ANNs provides the best overall performance as well.

3.2.2. Comparison with designing and TWEANN algorithms

The second category of the alternative ENN methods consists of ANN designing algorithms evolving only the network topology, in which the connection weights are optimized through BP. The methods presented by Cantu-Paz and Kamath (2005), namely the GA feature selection (GA-FS), the matrix method (MM), the pruning method (PRM) and the parameter method (PAM), and GE-BP (Soltanian et al., 2013) are five methods that fall in this category. GA-FS uses a binary encoded GA to select a subset of the input features, and MM as well as PRM utilizes a binary matrix representing the connectivity of an ANN. In PAM, the number of hidden neurons, the parameters for BP and the range of initial weights are found by means of a GA. Moreover, GE-BP uses a vector of integers for encoding the network topology, with a CFG-based decoder. The results of GA-FS, MM, PRM and PAM are taken from Cantu-Paz and Kamath (2005) (of course, no results

Table 7
Comparison between GEGA and ANN designing and TWEANN algorithms.

Dataset	Designing methods										TWEANN methods					
	GA-FS		MM		PRM		PAM		GE-BP		GE		GP		GEGA	
Breast Cancer	96.48[a]	1.38[b]	**96.77**	1.10	96.31	1.21	96.69	1.13	95.88	3.64	96.02	0.44	96.27	0.91	96.19	0.89
	5[c]	0[d]	–		–		–		4.6	0.8	3.0	0.5	3.3	0.5	**2.0**	0.0
	30000[e]		115000		1250		125000		92000		92000		92000		92000	
Credit-German	71.00	3.16	71.76	2.06	69.70	2.42	71.70	5.75	71.72	2.16	71.52	2.52	–		**72.46**	1.85
	10	0	–		–		–		5.0	1.1	**3.4**	1.0	–		**3.4**	0.8
	51750		585000		4050		187500		250000		250000		–		250000	
Heart-Cleveland	84.72	12.49	76.78	7.87	**89.50**	3.36	65.89	13.55	80.80	5.53	79.60	2.64	80.71	2.45	82.24	2.40
	5	0	–		–		–		2.5	0.7	3.3	1.4	2.9	0.3	**2.4**	1.0
	60000		380000		2100		250000		200000		200000		200000		200000	
Ionosphere	87.00	1.82	87.06	2.14	83.66	1.90	85.58	3.08	88.94	2.91	**89.93**	4.00	87.83	3.14	89.55	3.00
	10	0	–		–		–		6.6	1.6	5.0	0.8	9.3	0.4	**2.6**	0.7
	60000		580000		3100		250000		200000		200000		200000		200000	
Iris	93.60	2.41	92.40	2.67	92.40	1.40	91.73	8.26	**95.72**	6.47	94.93	2.33	95.22	2.05	95.70	1.55
	5	0	–		–		–		4.5	0.7	3.5	0.6	8.9	0.5	**2.1**	0.3
	120000		400000		1400		500000		250000		250000		320000		250000	
Sonar	72.98	5.03	71.34	4.09	**73.94**	3.98	72.59	4.16	69.82	4.63	69.52	4.03	–		73.17	1.95
	10	0	–		–		–		4.5	1.4	5.0	1.2	–		**2.1**	0.3
	103500		1155000		4150		375000		250000		250000		–		250000	
Wine	86.06	10.59	90.56	3.14	93.37	2.33	94.04	2.37	90.08	3.88	90.89	2.67	–		**94.16**	3.34
	5	0	–		–		–		2.5	0.7	**2.8**	1.0	–		3.2	0.8
	22500		120000		1675		93750		75000		75000		–		75000	
Average	*84.55*	*5.27*	*83.81*	*3.30*	*85.55*	*2.37*	*82.60*	*5.47*	*84.71*	*4.17*	*84.63*	*2.66*	***90.01***	*2.14*	*86.21*	*2.14*
	7.1	*0*	–		–		–		*4.3*	*1.0*	*3.7*	*0.9*	*6.1*	*0.4*	***2.5***	*0.6*
	63964		*476429*		*2532*		*254464*		*188143*		*188143*		*203000*		*188143*	

[a] Average classification accuracy.
[b] Standard deviation of the classification accuracy.
[c] Average number of hidden neurons.
[d] Standard deviation of the number of hidden neurons.
[e] Computational effort.

Table 8
Statistical comparison between GEGA and ANN designing and TWEANN algorithms.

Dataset	Designing methods					TWEANN methods		
	GA-FS	MM	PRM	PAM	GE-BP	GE	GP	GEGA
Breast Cancer	5	8	8	8	5	9	9	**12**
Credit-German	4	7	6	7	6	9	–	**10**
Heart-Cleveland	5	7	**14**	2	9	8	9	10
Ionosphere	5	9	2	5	10	12	7	**14**
Iris	7	6	5	8	7	11	6	**14**
Sonar	4	7	8	7	7	6	–	**10**
Wine	3	4	9	**11**	5	6	–	**11**
Overall performance	33	48	52	48	49	61	31	**81**

concerning number of hidden neurons of the ANNs generated by MM, PRM and PAM have been reported in Cantu-Paz and Kamath (2005)); while, to have a suitable comparison, the results of GE-BP are obtained from the experiments performed. Besides, the computational efforts of GA-FS, MM, PRM, as well as PAM are calculated based on the corresponding results reported in Cantu-Paz and Kamath (2005).

The last category of the alternative ENN methods consists of TWEANN algorithms that, like GEGA, simultaneously evolve the topology and the connection weights. GE (Tsoulos et al., 2008) and GP (Rivero et al., 2010) are two methods that fall in this category. GE uses a vector of integers for encoding the topology and the connection weights and a BNF grammar for decoding, while GP is a method to design and train ANNs by means of a tree-based GP algorithm. The results of GP, including its computational efforts, are taken from Rivero et al. (2010); while, to have a suitable comparison, the results of GE are obtained from the experiments performed.

Table 7 provides a comparison between the results of GEGA and those of the designing and TWEANN algorithms mentioned above. The best results concerning average classification accuracy (in percentage) and average number of hidden neurons are indicated in boldface. GEGA, GE-BP and GE are allowed to run for almost the same computational effort as MM, PAM and GP, since GA-FS and in particular PRM require relatively less computational efforts.

It can be seen from Table 7 that GEGA results in a high classification accuracy for most of the datasets. Although GP provides the best performance on average, it has not used all the datasets; if only the four datasets used by GP are considered, GEGA has an average of 90.92 which is higher than that of GP. Moreover, in terms of the number of hidden neurons, GEGA leads to the best results for almost all datasets. Table 8 statistically compares the performances of GEGA and the seven alternative ENN methods by means of the mentioned approach. As seen, GEGA leads to the best results for almost all datasets and provides the best overall performance as well.

4. Discussion

As shown in the previous section, in terms of the classification accuracy and the number of hidden neurons, GEGA evolving both the topology and the connection weights of ANNs provides the best overall performance among the ENN methods in the study. In this section, however, some other characteristics of GEGA are described in details.

4.1. Minimal dependency with the expert

In the evolutionary training of ANNs, the network topology must first be determined, which may require an excessive amount of expert's effort. In the evolutionary design of ANNs, some human efforts are required too; PRM (Cantu-Paz and Kamath, 2005), for example, initially needs a trained network. Moreover, the other TWEANN methods are usually not completely independent from the expert; GP (Rivero et al., 2010), for example, has, in addition to

a number of GP parameters, also three parameters including the penalty coefficient, tree height and maximum number of inputs to each neuron that have to be set by the expert. However, one characteristic of the TWEANN method proposed in this paper is that it needs to invest a minimal expert's effort for customization. GEGA has six numeric parameters listed in Table 2 to be set by the user; clearly, the determination of the appropriate values of such parameters is unavoidable for any other ENN algorithms too. Except for the parameters, there is only the increasing function $q(\bar{h})$ used by the proposed penalty approach to be defined. Nevertheless, as the problems used in the experiments have a variety of complexities, it can be concluded from the results that the function defined by Eq. (5) is useful for other classification problems as well.

4.2. Automatic selection of the features

Removing redundant and irrelevant features is very important in the generation of classifiers. These features yield no additional and useful information for the classification task, and even may reduce the accuracy. Automatic techniques of feature selection are typically divided into filter, wrapper, and embedded methods (Blum and Langley, 1997). Filter methods select a subset of the features prior to learning based on some criterion. Since these methods consider the features independently and the feature selection criterion may not perfectly match to the classification accuracy, they are prone to failure. To select a subset of the features in a wrapper method, a number of candidate solutions, subsets of the input features, are evaluated on the classification accuracy results, thus allowing a more precise evaluation. Moreover, embedded methods select a subset of the features and train the classifier simultaneously. The method proposed in this paper, GEGA, falls in the category of embedded feature selection approaches. To see that GEGA can avoid redundant and irrelevant features, Fig. 9, for example, presents one of the networks generated for the Sonar dataset with an accuracy of 73.08. As can be seen in the figure, only 12 out of 60 input features are selected to solve the problem with high accuracy, that is, not so many features are significant to solve it.

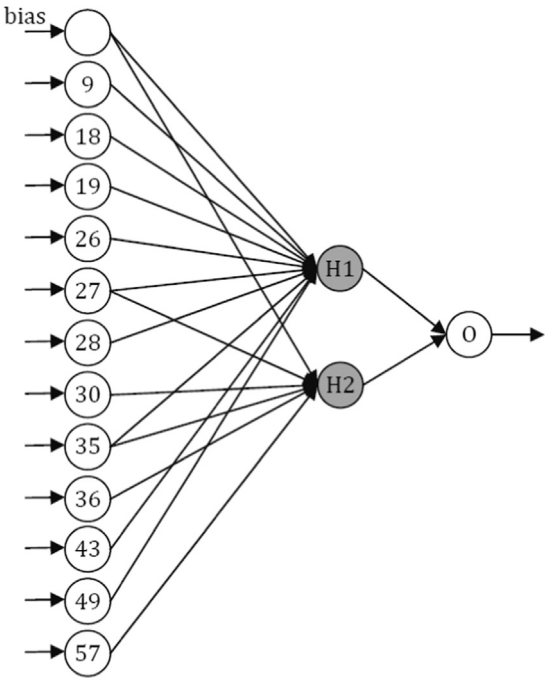

Fig. 9. A network generated through GEGA for the Sonar dataset.

Table 9
Comparison of the feature selection's ability between GEGA and GA-FS.

Dataset	GA-FS	GEGA
Breast Cancer	70.00	**68.89**
Credit-German	47.74	**26.72**
Heart-Cleveland	48.46	**40.95**
Ionosphere	47.65	**39.75**
Iris	**57.50**	92.50
Sonar	53.17	**33.67**
Wine	**65.38**	67.69
Average	55.70	**52.88**

To further see the ability of GEGA to automatically select the features, a comparison between the average percentage number of features selected by GEGA for each dataset and that of GA-FS (Cantu-Paz and Kamath, 2005) is given in Table 9, where best values are indicated in boldface. GA-FS is a wrapper method which uses a binary encoded GA to select a subset of the input features that will then be used to train the network. The results in Table 9 correspond to the experiment previously reported in Table 7. In addition, the results of GA-FS are calculated based on the corresponding results reported in Cantu-Paz and Kamath (2005). The comparison shows that GEGA outperforms GA-FS in terms of the feature selection, in particular for the three datasets with higher number of features, namely, the Credit-German, Sonar, and Ionosphere datasets. Recall that, from Table 7, GEGA results in a higher classification accuracy than GA-FS, except for the Breast Cancer and Heart-Cleveland datasets. In conclusion, although GEGA selects a much smaller number of features for the three largest datasets used in the study in particular, it results in a higher classification accuracy than GA-FS; this confirms the ability of GEGA to select the most relevant input features.

4.3. Generation of simple ANNs

The main goal of this paper is undoubtedly to propose a system that can produce ANNs with not only low error on the training set but also high generalization on previously unseen data, measured by the classification accuracy on the test set. However, the generalization ability of an ANN may decrease as a result of overfitting problems occurring when the ANN topology is too large. Accordingly, as stated previously, the proposed adaptive penalty approach is incorporated into GEGA to simplify ANNs generated through the evolution process, thus leading to more generalization ability. The results given in Tables 5 and 7 indicate that GEGA outperforms on average all the nine other algorithms in terms of the classification accuracy (as explained earlier), and also produces ANNs with a much smaller number of hidden neurons. Taking into account these results as well as the ability of GEGA to select a small subset of the input features, it can thus be concluded that GEGA which takes advantage of a desirable penalty approach generates sufficiently simple ANNs that have high generalization capability.

In Table 5, when comparing GEGA with the two ANN training algorithms, GEGA generates ANNs with an average of 3.6 hidden neurons which is much smaller than that of G3PCX as well as GA. This is, of course, not surprising since G3PCX and GA train ANNs with fixed topologies, namely three-layered, fully connected feedforward ANNs. Moreover, in Table 7, when comparing GEGA with the two ANN designing algorithms with known results concerning average number of hidden neurons and the two TWEANN algorithms, again GEGA producing ANNs with an average of 2.5 hidden neurons is considerably superior; however, unlike the training algorithms, all these algorithms design the network topology. Indeed, due to the fact that during the evolution

```
<S>       ::= <Node> | <Node> + <S>
<Node>    ::= w * sig(<Sum> + w)
<Sum>     ::= w * <xxList> | w * <xxList> + <Sum>
<xxList>  ::= x1 | x1 | x2
```

Fig. 10. A modified version of the proposed CFG.

process GEGA penalizes, in an appropriate manner, the number of hidden neurons, it can produce ANNs with a much smaller number of hidden neurons (and thus connections). We observe, for example, that the network shown in Fig. 9 has only 2 hidden neurons and 18 connections, which is much simpler than the network traditionally used for the Sonar dataset having 10 hidden neurons and 620 connections (Opitz and Maclin, 1999). Simple ANNs not only lead to high performance in generalization but also are important because of their easy implementation as well as better opportunities they provide to extract some knowledge.

4.4. Exploiting expert knowledge

Another advantage of GEGA which lies in using GE to design the network topology is that it can exploit expert knowledge about the problem on hand in order to more efficiently search in the infinite space of topologies. Indeed, expert knowledge can be exploited through introducing bias into the proposed CFG. Suppose we know that some of the input features are more significant than the others to solve the problem—such information can even be achieved through other feature selection techniques, such as filter methods requiring the least computation time. The proposed CFG can then be modified to incorporate this knowledge, so that the given features are more likely to be selected than the others. Fig. 10, for example, presents a modified version of the CFG for a problem with two input features in which the importance of the first feature is twice the importance of the other. With this modified grammar, we can expect that the algorithm converges easily towards promising regions of the search space.

Furthermore, a minimum number of hidden neurons can be considered for the population's individuals by easily modifying the first production rule. For example, the following rule generates topologies with at least 2 hidden neurons:

$$< S > \rightarrow :: = < \text{Node} > + < \text{Node} > | < \text{Node} > + < S >$$

while the following rule generates topologies with at least 3 hidden neurons:

$$< S > \rightarrow : \\ : = < \text{Node} > + < \text{Node} > + < \text{Node} > | < \text{Node} > + < S >$$

5. Conclusions and future work

To simultaneously evolve the topology and the connection weights of ANNs, this paper proposes a new combination of GE and GA that needs to invest a minimal expert's effort for customization. To efficiently search the infinite space of topologies as well as connection weights, the proposed system uses grammatical encoding for the topology representation and real encoding for the weights representation in a chromosome. Using GE to design the network topology can also help the user introduce other advices into the algorithm to make the search more directed. Moreover, to simplify ANNs generated through the evolution process, the system utilizes a novel adaptive penalty approach that encourages smaller topologies leading to more generalization ability when needed. To assess the performance of the proposed algorithm, extensive experiments are performed on real world classification datasets and the results are statistically compared against other well-known and state-of-the-art ENN algorithms in the literature. The computational results demonstrate the superiority of our algorithm over the other methods as it provides the best overall performance in terms of the classification accuracy and the number of hidden neurons. The proposed system is well-suited to the whole development of ANNs for classification, and thus for disease diagnosis, image classification, fault diagnosis, text classification, etc., even though it may be applicable to other situations such as prediction and control systems after some adjustments.

The algorithm proposed in this study, however, needs some mechanism to prevent the weights from taking extreme or too small values. It also needs some more suitable crossover operator that will be more adapted to the suggested chromosome encoding and not some generic crossovers as those in the paper. These issues point to directions for future research efforts. As another direction for future research, it would be interesting to add the ability to generate ANNs with more than one hidden layer and even recurrent ANNs. Moreover, it would be interesting to evolve the neuron activation functions, in addition to evolving the topology and the connection weights of ANNs.

References

Angeline, P.J., Saunders, G.M., Pollack, J.B., 1994. An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans. Neural Netw. 5, 54–65.

Azzini, A., Tettamanzi, A., 2011. Evolutionary ANNs: a state of the art survey. Intelligenza Artificiale 5, 19–35.

Baxter, J., 1992. The evolution of learning algorithms for artificial neural networks. In: Green, D., Bossomaier, T. (Eds.), Complex Systems. IOS Press, Amsterdam, pp. 313–326.

Belew, R., McInerney, J., Schraudolph, N., 1991. Evolving networks: using the genetic algorithm with connectionist learning. In: Proceedings of the Second Artificial Life Conference, Addison-Wesley, New York, pp. 511-547.

Bengio, Y., Bengio, S., Cloutier, J., 1990. Learning a Synaptic Learning Rule (Technical Report 751). Université de Montréal, Département d'informatique et de recherche opérationnelle, Montréal, Cananda.

Blake, C.L., Merz., C.J., 1998. UCI Repository of Machine Learning Databases.

Blum, A., Langley, P., 1997. Selection of relevant features and examples in machine learning. Artif. Intell. 97, 245–271.

Cantu-Paz, E., Kamath, C., 2005. An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. IEEE Trans. Syst. Man Cybern. Part B: Cybern. 35, 915–927.

Castellani, M., 2013. Evolutionary generation of neural network classifiers—an empirical comparison. Neurocomputing 99, 214–229.

Castellani, M., Rowlands, H., 2009. Evolutionary artificial neural network design and training for wood veneer classification. Eng. Appl. Artif. Intell. 22, 732–741.

Caudell, T.P., Dolan, C.P., 1989. Parametric connectivity: training of constrained networks using genetic algorithms. In: Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, CA, pp. 370-374.

Chen, L., Tan, C.H., Kao, S.J., Wang, T.S., 2008. Improvement of remote monitoring on water quality in a subtropical reservoir by incorporating grammatical evolution with parallel genetic algorithms into satellite imagery. Water Res. 42, 296–306.

Chen, L., Wang, T.S., 2010. Modeling strength of high-performance concrete using an improved grammatical evolution combined with macrogenetic algorithm. J. Comput. Civil Eng. 24, 281–288.

Coban, R., 2013. A context layered locally recurrent neural network for dynamic system identification. Eng. Appl. Artif. Intell. 26, 241–250.

Czajkowski, A., Patan, K., Szymański, M., 2014. Application of the state space neural network to the fault tolerant control system of the PLC-controlled laboratory stand. Eng. Appl. Artif. Intell. 30, 168–178.

Deb, K., Anand, A., Joshi, D., 2002. A computationally efficient evolutionary algorithm for real-parameter optimization. Evol. Comput. 10, 371–395.

Dietterich, T.G., 1998. Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput. 10, 1895–1924.

Donate, J.P., Cortez, P., Sánchez, G.G., Miguel, A.S.de, 2013. Time series forecasting using a weighted cross-validation evolutionary artificial neural network ensemble. Neurocomputing 109, 27–32.

Du, K.L., 2010. Clustering: a neural network approach. Neural Netw. 23, 89–107.

Eiben, A.E., Smith, J.E., 2003. Introduction to Evolutionary Computing. Springer-Verlag, Berlin, Heidelberg.

Felice, M.D., Yao, X., 2011. Short-term load forecasting with neural network ensembles: a comparative study. IEEE Comput. Intell. Mag. 6, 47–56.

Fogel, D.B., Wasson, E.C., Boughton, E.M., 1995. Evolving neural networks for detecting breast cancer. Cancer Lett. 96, 49–53.

Frean, M., 1990. The upstart algorithm: a method for constructing and training feedforward neural networks. Neural Comput. 2, 198–209.

Ghazikhani, A., Monsefi, R., Sadoghi Yazdi, H., 2013. Ensemble of online neural networks for non-stationary and imbalanced data streams. Neurocomputing 122, 535–544.
Gruau, F., 1993. Genetic synthesis of modular neural networks. In: Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 318-325.
Gruau, F., 1994. Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm. Ph.D. Thesis, Ecole Normale Superieure de Lyon, France.
Heidrich-Meisner, V., Igel, C., 2009. Neuroevolution strategies for episodic reinforcement learning. J. Algorithms 64, 152–168.
Huanhuan, C., Yao, X., 2010. Multiobjective neural network ensembles based on regularized negative correlation learning. IEEE Trans. Knowl. Data Eng. 22, 1738–1751.
Khashei, M., Bijari, M., 2012. Hybridization of the probabilistic neural networks with feed-forward neural networks for forecasting. Eng. Appl. Artif. Intell. 25, 1277–1288.
Kitano, H., 1990. Designing neural networks using genetic algorithms with graph generation system. Complex Syst. 4, 461–476.
Kohavi, R., John, G., 1997. Wrappers for feature subset selection. Artif. Intell. 97, 273–324.
Li, X.L., Jia, C., Liu, D.X., Ding, D.W., 2014. Nonlinear adaptive control using multiple models and dynamic neural networks. Neurocomputing 136, 190–200.
Merrill, J.W.L., Port, R.F., 1991. Fractally configured neural networks. Neural Netw. 4, 53–60.
Miller, G.F., Todd, P.M., Hegde, S.U., 1989. Designing neural networks using genetic algorithms. In: Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, CA, 379-384.
Montana, D., Davis, L., 1989. Training feedforward neural networks using genetic algorithms. In: Proceedings of the 11th International Joint Conference on Artificial Intelligence, 762-767.
Motsinger, A.A, Reif, D.M., Dudek, S.M., Ritchie, M.D., 2006. Understanding the evolutionary process of grammatical evolution neural networks for feature selection in genetic epidemiology. In: Proceedings of the IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology, 1-8.
Muttil, N., Chau, K.W., 2006. Neural network and genetic programming for modelling coastal algal blooms. Int. J. Environ. Pollut. 28, 223–238.
O'Neill, M., Ryan, C., 2001. Grammatical evolution. IEEE Trans. Evol. Comput. 5, 349–358.
O'Neill, M., Ryan, C., 2003. Grammatical Evolution: Evolutionary Automatic Programming in an Arbitary Language. Genetic Programming. Vol. 4. Kluwer Academic Publishers, Dordrecht.
Oong, T.H., Isa, N.A.M., 2011. Adaptive evolutionary artificial neural networks for pattern classification. IEEE Trans. Neural Netw. 22, 1823–1836.
Opitz, D., Maclin, R., 1999. Popular ensemble methods: an empirical study. J. Artif. Intell. Res. 11, 169–198.
Artificial neural networks in real-life applications. In: Rabuñal, J.R., Dorado, J. (Eds.), Idea Group Inc.
Reed, R., 1993. Pruning algorithms—a survey. IEEE Trans. Neural Netw. 4, 740–747.
Rivero, D., Dorado, J., Rabuñal, J., Pazos, A., 2010. Generation and simplification of artificial neural networks by means of genetic programming. Neurocomputing 73, 3200–3223.
Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1986. Learning internal representations by error propagation. In: Rumelhart, D.E., McClelland, J.L. (Eds.), Parallel Distributed Processing: Explorations in the Microstructures of Cognition, 1. MIT Press, Cambridge, MA, pp. 318–362.
Ryan, C., Collins, J.J., O'Neill, M., 1998. Grammatical evolution: evolving programs for an arbitrary language. In: Proceedings of the First European Workshop on Genetic Programming, Springer-Verlag, Berlin, pp. 83-95.
Ryan, C., Nicolau, M., O'Neill, M., 2002. Genetic algorithms using grammatical evolution. In: Foster, J., Lutton, E., Miller, J., Ryan, C., Tettamanzi, A. (Eds.), Genetic Programming. Springer-Verlag, Berlin, Heidelberg, pp. 278–287.
Siddiqi, A.A., Lucas, S.M., 1998. A comparison of matrix rewriting versus direct encoding for evolving neural networks. In: Proceedings of the IEEE International Conference on Evolutionary Computation, 392-397.
Sietsma, J., Dow, R.J.F., 1991. Creating artificial neural networks that generalize. Neural Netw. 4, 67–79.
Soltanian, K., AkhlaghianTab, F., Ahmadizar, F., Tsoulos, I., 2013. Artificial neural networks generation using grammatical evolution. In: Proceedings of the 21st Iranian Conferebce on Electrical Engineering (ICEE), Mashhad, Iran, pp. 1-5.
Stanley, K.O., D'Ambrosio, D.B., Gauci, J., 2009. A hypercube based indirect encoding for evolving large-scale neural networks. Artif. Life 15, 185–212.
Stanley, K.O., Miikkulainen, R., 2002. Evolving neural networks through augmenting topologies. Evol. Comput. 10, 99–127.
Stork, D.G., Walker, S., Burns, M., Jackson, B., 1990. Preadaptation in neural circuits. In: Proceedings of Inernational Joint Conference of Neural Networks, Washington DC, 202-205.
Tamura, S., Tateishi, M., 1997. Capabilities of a four-layered feedforward neural network: four layers versus three. IEEE Trans. Neural Netw. 8, 251–255.
Taormina, R., Chau, K.W., Sethi, R., 2012. Artificial neural network simulation of hourly groundwater levels in a coastal aquifer system of the Venice lagoon. Eng. Appl. Artif. Intell. 25, 1670–1676.
Topchy, A.P., Lebedko, O.A., 1997. Neural network training by means of cooperative evolutionary search. Nucl. Instrum. Methods Phys. Res., Sect. A 389, 240–241.
Tsoulos, I., Gavrilis, D., Glavas, E., 2008. Neural network construction and training using grammatical evolution. Neurocomputing 72, 269–277.
Weimer, D., Thamer, H., Scholz-Reiter, B., 2013. Learning defect classifiers for textured surfaces using neural networks and statistical feature representations. Proc. CIRP 7, 347–352.
Weizhong, Y., 2012. Toward automatic time-series forecasting using neural networks. IEEE Trans. Neural Netw. Learn. Syst. 23, 1028–1039.
Whitley, D., 1989. The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In: Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, CA, 116-123.
Whitley, D., Starkweather, T., Bogart, C., 1990. Genetic algorithms and neural networks: optimizing connections and connectivity. Parallel Comput. 14, 347–361.
Yang, J., Honavar, V., 1998. Feature subset selection using a genetic algorithm. IEEE Intell. Syst. 13, 44–49.
Yang, S.H., Chen, Y.P., 2012. An evolutionary constructive and pruning algorithm for artificial neural networks and its prediction applications. Neurocomputing 86, 140–149.
Yao, X., 1999. Evolving artificial neural networks. Proc. IEEE 87, 1423–1447.
Yao, X., Islam, Md.M., 2008. Evolving artificial neural network ensembles. IEEE Comput. Intell. Mag. 3, 31–42.
Yao, X., Liu, Y., 1997. A new evolutionary system for evolving artificial neural networks. IEEE Trans. Neural Netw. 8, 694–713.
Zaidan, A.A., Ahmad, N.N., Abdul Karim, H., Larbani, M., Zaidan, B.B., Sali, A., 2014. Image skin segmentation based on multi-agent learning Bayesian and neural network. Eng. Appl. Artif. Intell. 32, 136–150.
Zhang, G.P., 2000. Neural networks for classification: a survey. IEEE Trans. Syst. Man Cybern. Part C: Appl. Rev. 30, 451–462.