Available online at www.sciencedirect.com
SCIENCE DIRECT®

Neural Networks 18 (2005) 934–950

Neural Networks

www.elsevier.com/locate/neunet

Biological engineering applications of feedforward neural networks designed and parameterized by genetic algorithms

Konstantinos P. Ferentinos*

Department of Biological and Environmental Engineering, Cornell University, Ithaca, NY 14850, USA

Received 3 November 2003; revised 3 March 2005; accepted 3 March 2005

Abstract

Two neural network (NN) applications in the field of biological engineering are developed, designed and parameterized by an evolutionary method based on the evolutionary process of genetic algorithms. The developed systems are a fault detection NN model and a predictive modeling NN system. An indirect or 'weak specification' representation was used for the encoding of NN topologies and training parameters into genes of the genetic algorithm (GA). Some a priori knowledge of the demands in network topology for specific application cases is required by this approach, so that the infinite search space of the problem is limited to some reasonable degree. Both one-hidden-layer and two-hidden-layer network architectures were explored by the GA. Except for the network architecture, each gene of the GA also encoded the type of activation functions in both hidden and output nodes of the NN and the type of minimization algorithm that was used by the backpropagation algorithm for the training of the NN. Both models achieved satisfactory performance, while the GA system proved to be a powerful tool that can successfully replace the problematic trial-and-error approach that is usually used for these tasks.
© 2005 Elsevier Ltd. All rights reserved.

Keywords: Genetic algorithms; Neural networks; Biological engineering; Automatic neural network design; Training parameterization

1. Introduction

This work deals with neural network applications in the field of biological engineering and more specifically in modern hydroponic plant production systems. Neural networks (NNs) have been used to model a variety of biological and environmental processes (e.g. Altendorf, Elliott, Stevens, & Stone, 1999; Bhat, Minderman, McAvoy, & Wang, 1990; Hong, Tan, & McCall, 2000; Lacroix, Salehi, Yang, & Wade, 1997; Seginer, Boulard, & Bailey, 1994; Sridhar, Seagrave, & Bartlett, 1996), but not in the specific area of plant cultivation. Hydroponics is the growing of plants in a water and fertilizer solution that contains the necessary nutrients for optimal plant growth. In other words, hydroponics is a soilless cultivation of plants. There are several types of hydroponic systems, the most popular being Nutrient Film Technique (NFT), deep trough hydroponics and hydroponics on substrates like cocosoil or pumice. Knowledge of the highly nonlinear dynamics of interacting biological systems within hydroponically grown plants is limited. Moreover, hydroponic systems can be monitored with a level of detail that permits one to collect extensive sets of data about the system. Thus, the NN approach shows promise as a means to avoid the need to model internal plant processes and still achieve a prediction capability suitable for control.

The application of NNs in modeling nonlinear processes has a central drawback: the lack of a precise method to choose the most appropriate network topology, type of activation functions and parameters of the training algorithm. These tasks, in the case of engineering applications, are usually based on a trial-and-error procedure performed by the developer of the model. In that way, optimality or even near-optimality is not guaranteed, as the explored space is just a small portion of the whole search space and the type of search is rather random. To overcome the problems associated with human network design and training parameterization, an automated method, based on the evolutionary properties of genetic algorithms (GAs), is

* Current address: Informatics Laboratory, Agricultural University of Athens, 75 Iera Odos Street, Athens 118 55, Greece. Tel.: +30 210 764 8288; fax: +30 210 529 4199.
E-mail address: kpf3@cornell.edu

0893-6080/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2005.03.010

developed. GAs evolve several network designs with different activation functions and several minimization algorithms so that the best possible combination is finally chosen.

In the vast majority of real-world engineering applications of neural networks, the common technique of trial-and-error performed by a human designer is used. However, during the last 15 years, an effort of making this process of designing a NN more sophisticated and less human-dependent is underway. Because of the 'combinatorial optimization' aspect of the designing problem, evolutionary computation methods and particularly GAs (Goldberg, 1989a; Holland, 1975) and evolutionary programming (Fogel, 1994, 1995) are used as the optimal-designing tool. These algorithms perform a global exploration of the search space and use several heuristics to avoid getting trapped in local optima. On the other hand, several other techniques of manipulating NN architectures, like the pruning methods (Reed, 1993) and the constructive methods (Frean, 1990; Gallant, 1993; Parekh, Yang, & Honavar, 2000) are susceptible to becoming trapped at structural local optima and they only investigate restricted topological subsets rather than the complete class of network architectures (Angeline, Saunders, & Pollack, 1994). This is also the case for the weight sharing methods (Nowlan & Hinton, 1992) for network size reduction.

The initiation of the research in the area of evolving NN topologies appears to have been made with the introduction of two different approaches to the subject. In the first approach, the indirect or 'weak-specification' representation of the implementation was proposed (Harp, Samad, & Guha, 1989). For the first time, the problems of human-based exploration of a large space as large as the domain of possible network architectures were addressed. GAs, with their relative immunity to high dimensionality, local *minima* and noise, were considered to be the best tool for automation of the design task. The version of the genetic algorithm used by Harp et al. was the common one, with elitism. The weak specification representation that was chosen, used relatively abstract genetic 'blueprints' that were then translated into network phenotypes. The representation of the weak specification category uses specific correspondences of specific binary strings to specific network architectures that are pre-defined by the user. The approach of Harp et al. used bit strings of variable length, which complicated drastically the genetic algorithm representation, making the encoding more difficult and time consuming. However, this representation has many degrees of freedom, making the range of possible network architectures that can be constructed very wide.

The second approach to the subject of evolving NN topologies is the direct or 'strong-specification' representation of the genetic algorithm implementation (Miller, Todd, & Hegde, 1989). This representation encodes explicitly every network connection from node to node in a way that someone, by just looking at a bit string, can encode the corresponding network topology. A bit of 1 corresponds to a connection while a bit of 0 corresponds to lack of connection. This type of representation requires very large binary strings and can be problematic with all the restrictions that feedforward NNs comprise (i.e. no cycles, no connections between nodes of non-successive layers or between nodes of the same layer, etc.), as it is explained in more detail below. The main representation used by Miller et al. was that of translating a network architecture initially into a two dimensional matrix of zeros and ones, where a value of one at an element *ij* of the matrix represented a connection from node *i* to node *j*, while a zero represented no connection. With special techniques, certain constraints could be included in this representation. Then, each row of the matrix was put in line so that the final string of the genetic algorithm was formed. This approach has the main disadvantage that it requires very long strings to represent large networks and, in addition, the strings are of variable length. From the results extracted from some application cases, the authors suggested that weaker specification schemes should be explored.

Thus, it is evident that the direct encoding scheme is suitable for precise and deterministic handling of only small neural networks. Also, an important property of the indirect encoding scheme is that it is biologically more plausible than the direct encoding one because genetic information in real chromosomes cannot specify the whole nervous system directly and independently (Yao, 1993).

In the work presented here, the indirect or 'weak representation' scheme is used. Except for the network architecture, the types of activation functions of the hidden and output nodes, as well as the type of the minimization approach of the backpropagation algorithm, are also included in the GA encoding. In this way, not only are more aspects of NN application evolved, but also the major problems of deception (Goldberg, 1989c) and multimodality (Yao, 1993) are avoided. The search space of NN architectures is deceptive and multimodal, deceptive because similar network architectures may have dramatically different performances, multimodal because NNs with quite different architectures may have very similar capabilities. Genetic algorithms, which operate with building blocks (Goldberg, 1989b) according to the schema theorem (Reeves, 1995), seem to be problematic in environments with these characteristics, mostly because of the crossover operator (Angeline et al., 1994). However, the search spaces of NN activation functions and the possible types of training algorithms do not have the characteristics of deception and multimodality, thus the combined GA encoding of these matters with the network architecture that was used here, overcomes those problems. In addition, the indirect representation of specific architectures of the feedforward NN, overcomes the permutation problem (Hancock, 1992), i.e. the fact that similar networks may have the hidden units defined in different orders so that they have very dissimilar genetic strings, preventing successful recombination of building blocks.

This paper is organized as follows. Section 2 describes the materials and systems used in the presented applications. Section 3 gives a review of several approaches in the literature for neural network design, similar to the one developed here. Section 4 describes in detail the specific GA encoding scheme that was used in the proposed model and gives an overview of the algorithm of the model. Section 5 presents some comparisons of the developed model with the trial-and-error procedure, in the real-world bioengineering applications described in Section 2 and finally, Section 6 states the basic characteristics of the system and the conclusions based on the results of the previous section.

2. Materials and methods of application systems

Two feedforward neural network applications in the field of biological engineering were developed: a fault detection NN system of a deep-trough hydroponic system (Ferentinos & Albright, 2003) and a predictive NN model of a similar hydroponic system (Ferentinos & Albright, 2002). Hydroponic systems, as mentioned above, are soilless cultivation systems where the plants are grown on appropriate substrates or directly on nutrient solution. They are usually monitored electronically and their main variables, like pH and electrical conductivity (EC) of the nutrient solution, are accurately controlled. In addition, these structures are situated inside greenhouses, where the environmental parameters, like temperature, relative humidity and lighting, are also monitored and controlled. The large amount of operation data that are collected in hydroponics is suitable for development of several models of the measured variables or other types of models like fault detection models, intelligent control, etc. Deep-trough hydroponics are systems in which the plants are placed on tanks that contain the nutrient solution required by the plants. The composition of the nutrient solution is automatically controlled. Here, two quite similar deep-trough hydroponic systems were used. In both cases, lettuce was the cultivated plant. The technical characteristics of the arrangements were identical. The only difference was that the hydroponic system used in the fault detection NN model was of continuous production, that is, every 2 days a number of large plants were harvested and the same number of small plants were transplanted in the system. In other words, the system always had a constant range of plant ages. On the other hand, the hydroponic system of the predictive NN model was simpler: all plants were transplanted in the tanks at the same time, and were all harvested at the end of their growing period. Thus, the biochemical dynamics of the two cases were similar, but not identical.

2.1. Experimental arrangement

The experiments were conducted in a section of the Kenneth Post Laboratory (KPL) Greenhouses at Cornell University, in Ithaca, NY. The greenhouse section had a floor area of about 85 m^2 and was fully equipped with a staged ventilation system, an evaporative cooling system, a lighting system and a movable shading system. The controlled environmental variables were the daily integral of light (photosynthetically active radiation—PAR) and the air temperature. Light intensity, air temperature, relative humidity and CO_2 concentration were continuously monitored. During the experiments, temperature setpoints were 24 °C during day and 19 °C during night and were achieved within ±0.5 °C. The daily PAR integral setpoint was 17 mol m^{-2} and was achieved by using supplemental lighting from 21 high-pressure sodium, 400 W, lamps during winter and the shading screen during summer. The relative humidity was maintained between 30 and 70%.

The cultivation system was a deep trough hydroponic system that consisted of three small growing tanks filled with nutrient solution. Each one had an area of 0.75 m^2 (1.25 m×0.60 m) and root zone control was completely independent of the others. In this way, the systems could be monitored and controlled in parallel and, thus, more data sets could be constructed. The monitoring and control system consisted of a personal computer (PC) running LabVIEW software (available from National Instruments), a data acquisition board connected to the computer and several meters, sensors and actuators connected to the board. The monitored parameters were the pH, the electrical conductivity (EC), the temperature, the dissolved oxygen (DO), the weights of two of the three ponds (for measuring evapotranspiration) and the control signals of pH and DO. For each pond there was a metering pump used to control the pH of the nutrient solution by adding acid or base, and solenoid valves that controlled the DO by adding oxygen. The program monitored and controlled the system every 10 s and logged its data every 5 min. The control methodology was based on the pseudo-derivative feedback (PDF) control algorithm (Phelan, 1977). The pH setpoint was 5.8 and the DO was maintained between 6.5 and 7 mg l^{-1}. The EC was not controlled automatically but its values were kept between the recommended setpoints of 115–125 mS m^{-1} by adjusting manually every 2 days by adding reverse-osmosis water to replace evapotranspiration and solution stocks to maintain the EC.

2.2. Fault detection system

The purpose of the first NN application presented in this paper was the early detection of specific malfunctions (faults) in a modern hydroponic system for plant cultivation. The NN model should read the measured conditions of the system and in real time detect and diagnose possible faults in the operation. The procedure of training the NN model requires, first, an accurate definition of ‘normal operation’, defined in our case as unstressed plants in a system which is in control. It is also required to define the ‘faulty operation’ and categorize this kind of operation into different types of faulty operations, one for each different kind of fault. In order to obtain data sets for each kind of fault, one has to impose those faults and take the corresponding measurements of the microenvironment variables. When the environmental and nutrient solution variables are within their limits and the plants appear healthy, the growth rate is optimised and operation of the system is considered ‘normal’. The ‘faulty operation’ consisted of three different kinds of faults: (i) *Actuator/mechanical faults*. These are failures in some actuator or some mechanical part of the hydroponic system. The actuator fault considered was the malfunction of the pH control pump (Fault Type 1) and the mechanical fault was the malfunction of the nutrient solution circulation pump (Fault Type 2). (ii) *Sensor faults*. These are failures in the sensors of the system. The ones considered were pH sensor failure (Fault Type 3) and EC sensor failure (Fault Type 4). Thus, four different faults were considered. For all these faulty operations, real data existed because sensor and actuator failures were encountered during daily operation of the system. In addition, several faults were especially imposed in order to train the NN model and investigate its inherent fault detection capabilities.

The training data set consisted of a matrix with columns that represented the average measurements of 10-min periods and were of the form

$$[\mathrm{pH}(t)\mathrm{pH}(t-1)\mathrm{pH}(t-2)\mathrm{EC}(t)\mathrm{EC}(t-1)\mathrm{EC}(t-2)\mathrm{DO}(t)$$
$$\ldots\mathrm{DO}(t-1)\mathrm{DO}(t-2)T_{\mathrm{s}}(t)T_{\mathrm{air}}(t)L(t)\mathrm{RH}(t)U_{\mathrm{pH}}(t)U_{\mathrm{DO}}(t)]^{\mathrm{T}}$$

where t is a specific time, $t-1$ is the previous time step (i.e. 10 min before), $t-2$ is two steps before, pH is the solution acidity, EC is the electrical conductivity, DO is the dissolved oxygen, T_{s} is the temperature of the nutrient solution, T_{air} is the temperature of the air inside the greenhouse, RH is the relative humidity inside the greenhouse, U_{pH} is the control signal for the pH and U_{DO} is the control signal for the DO, and T is the transpose operator. Thus the NN model had 15 inputs and was formed to have five outputs: one for normal operation, two for the actuator/mechanical faults and two for the sensor faults. The values of the outputs used for training the NN were binary. An output of unity for a specific output denoted the existence of the corresponding fault type (or of normal operation when referring to the first output). The values of the other outputs were zero. The outputs during testing of the model were, of course, continuous, with values from 0 to 1. In this way, a decision process was formed to determine above which value an output should be meaningful in terms of the existence of a fault.

2.3. Predictive modeling system

The second NN model presented in this paper consists of a system that predicts one-step-ahead values of the pH and the electrical conductivity of a modern hydroponic system for plant production. In all the experiments that were performed for the collection of training and testing data, pH, EC, and nutrient solution temperature were logged, as well as air temperature, relative humidity, and light intensity of the greenhouse environment. All these measurements, which constituted inputs to the NN model, were collected every 10 s and values averaged over 20 min were logged. In addition, the pH control signals of the system were used as input to the NN. Finally, one more input was included, the plant age estimator, to represent the age of the plants. The dynamics of the system change with time because of plant growth, so the model should take into consideration the age of the plants. This leads to an adaptive NN, with the plant age estimator being the adaptation parameter. The plant age was measured in 12-h intervals, and the growing period of the lettuce was 25 days, so plant age estimator values ranged from 1 to 50.

The training data set of the NN consisted of a matrix with columns that represented the average measurements of 20 min periods and were of the form

$$[\mathrm{pH}(t)\mathrm{EC}(t)T_{\mathrm{s}}(t)T(t)\mathrm{RH}(t)L(t)u_{\mathrm{pH}}\mathrm{A}(t)u_{\mathrm{pH}}\mathrm{B}(t)\mathrm{PAE}(t)]^{\mathrm{T}}$$

where t is a specific time period, pH is the solution acidity, EC is the solution electrical conductivity (μS cm^{-1}), T_{s} is the solution temperature (°C), T is the air temperature inside the greenhouse, RH is the relative humidity inside the greenhouse, L is the light intensity at the level of the plants, u_{pH}A is the amount of added acid for controlling the pH, u_{pH}B is the amount of added base and PAE is the plant age estimator. These were the nine inputs of the NN, while the outputs where the values of the pH and EC of the next 20-min time step.

3. Literature review

This work develops a genetic algorithm system for the optimal design and training parameterization of the NN models presented above. After the initiation of the subject of evolving neural network architectures by the two works presented in the introduction, several applications as well as improvements and advancements have been reported in the literature (for an excellent review, see Yao (1999)).

The successful application of NNs in modeling engineering problems is highly influenced by three major factors: network architecture, type of activation functions and type of training algorithm (Fine, 1999). Even within specific types of NNs (e.g. feedforward or recurrent networks), different architectures can give quite different performances. Similarly, different types or combinations of types of activation functions can differentiate the performance of apparently similar networks. In addition, especially in the case of engineering applications, even different minimization approaches of some specific training algorithm, like, for example, the backpropagation training algorithm (Rumelhart, Hinton, & Williams, 1986) that is most widely used, can result in perceptibly different performances.

In most of the relevant works in the literature, the minimization algorithm of the backpropagation learning process is not considered in the encoding of the genetic algorithm system. With the exception of some limited number of works (Iyoda & von Zuden, 1999; Liu & Yao, 1996; White & Ligomenides, 1993), the types of activation functions of the neural network are not considered either. In addition, some possible a priori knowledge of the system characteristics and possible general intuitions about the expected topology of the neural network, were not taken into account. Such an a priori knowledge can drastically limit the huge search space of the problem of neural network design, and more dimensions of the problem, like the minimization algorithm or the types of activation functions, can also be encoded into the genetic algorithm without making the encoding extremely complex and difficult to be optimized.

Kitano (1990) used genetic algorithms to develop an automated NN designing system that solves in some degree the problems of large string lengths and the inability to represent other than the network topology parameters, problems that were encountered in the previous works. His representation scheme, which can be categorized in the weak specification schemes but with a quite different approach than that of Harp et al. (1989), used a graph grammatical encoding that encoded graph generation grammar to the GA strings. Bebis, Georgiopoulos, and Kasparis (1997) proposed the coupling of GAs with weight elimination (Weigend, Rumelhart, & Huberman, 1991) to search the architecture by pruning oversized networks. Filho and de Carvalho (1997) used genetic algorithms with an indirect representation scheme to automate the designing of NNs. Their approach had a genetic algorithm string consisting of two parts: the parameter part and the layer part. In the first part, the learning rate and the momentum term were encoded, while in the second part, the size of each layer was encoded. A main difference from the works presented before is that a real-value version of the genetic algorithms was used and not a binary one. The main drawback of this approach is that, very often, invalid (unfeasible) architectures were constructed by the genetic operations of crossover and mutation. The application that was used for testing the automated design system showed that its performance was satisfactory compared to other methods.

Recently, Jiang, Zhao, and Ren (2003) developed a GA system that designs the optimal combination of feedforward and RBF NNs (namely, structural modular NNs). The general philosophy is similar to the one used here; however, no evolution of different activation functions and types of the backpropagation training algorithm was included. Sigmoid and Gaussian activation functions were used for the feedforward and the RBF networks, respectively. Only the Levenberg–Marquardt algorithm was tested in backpropagation training. A similar approach but exclusively for multilayer perceptrons with one hidden layer was used by Castillo, Merelo, Prieto, Rivas, and Romero (2000).

All previously presented methods tried to evolve only the architecture of the NN model. However, a substantial effort has been carried out in the simultaneous evolution of both architectures and weights of the NN, that is, using the evolutionary computation method for training as well as designing the network.

Yao and Liu (1997, 1998) used evolutionary programming to evolve both architectures and connection weights of neural networks. During that evolution, they gave specific emphasis on the behavioral link between parents and offspring, using various techniques such as partial training after each architectural mutation and node splitting, while the parsimony of the evolved networks was encouraged by favoring node deletion to node addition. Their technique is quite demanding in computational power, but it succeeds in both their goals of behavioral continuity during evolution and network parsimony. Garcia-Pedrajas, Hervas-Martinez, and Munoz-Perez (2003) recently developed an evolutionary model that evolved not only the weights and architectures of neural networks, but also the appropriate combination and cooperation of several (sub)networks in order to obtain better results than having a single neural network. Their methodology, even though quite time-consuming, gave good results when compared to other methods.

Another genetic algorithm system was used by Collins and Jefferson (1991) to evolve both architecture and weights that also allowed recurrent connections in the neural network, while Schmitz and Aldrich (1999) considered dynamic addition of groups of nodes based on combinatorial evolution in oriented ellipsoidal basis functions networks. Finally, a different type of GA called the structured GA was used by Dasgupta and McGregor (1992) to successfully design and train NNs for specific applications, like the XOR function and the 4×4 encoder/decoder problem.

Iyoda and von Zuben (1999) presented a quite different but rather limited application of GAs in the design of NNs. The developed GA system was used to find the best types of activation functions of the nodes of the network. Several activation functions like logistic, linear, Gaussian, trigonometric ones, etc. were encoded. The system was tested in both exactly modeled and approximately modeled functions. The performance in both cases was satisfactory. Liu and Yao (1996) used evolutionary programming to evolve NNs with both sigmoidal and Gaussian activation functions. Simultaneously, the methodology allowed the addition or deletion of nodes of the network. Something similar, but with specific percentages between sigmoidal and Gaussian activation functions, was done by White and Ligomenides (1993). Finally, Hwang, Choi, and Park (1997), in addition to network topology and activation functions, also included evolution of connection weights in the process, for projection neural networks.

The literature review shows the lack of an integrated system that takes into account, in addition to network topology and activation functions, the type of minimization methodology of the training algorithm, which, especially in real-world engineering applications, constitutes a major factor in the final performance of the NN model.

4. Genetic algorithm system

The main aspect in an evolutionary system like the one developed here, is the way of encoding the several possible phenotypes of the NN into specific genotypes. A phenotype, in our case, consists of the NN topology, its activation functions in the hidden and output nodes and in addition, the minimization algorithm that is used by the backpropagation algorithm during training. A genotype is a sequence of bits (0 or 1) with a specific constant length. Each genotype corresponds to a unique phenotype. The representation treats the problem as a combinatorial one.

4.1. Bit-string representation

The weak specification scheme that was developed and used here incorporates three tasks of the NN design and training parameterization:

(i) the selection of the minimization algorithm used by the backpropagation training algorithm
(ii) the architecture of the NN
(iii) the types of the activation functions of the hidden nodes and of the output nodes.

4.1.1. Representation of the minimization algorithm used by the backpropagation algorithm

The backpropagation training algorithm (Rumelhart et al., 1986) was used for the adaptation of the connection weights of the networks (i.e. training) during the evolution process. There are four different multidimensional minimization algorithms considered by the GA: steepest descent, quasi-Newton, Levenberg–Marquardt and conjugate gradient algorithms. There is no general rule regarding which algorithm is suitable for which application. It is known that feedforward NN training is an NP-complete problem (Blum & Rivest, 1992; Judd, 1990) and there is little reason to expect that a uniformly best algorithm for training can exist (Fine, 1999). The selection of the best training algorithm is a task based exclusively on trial-and-error and user experience and insight about any specific application.

All NN parameters (weights and biases) are denoted by a vector $\underline{w} \in W \subset R^p$, where p is the total number of parameters of the network. In the backpropagation algorithm, the vector of the network parameters $\underline{w}$ is updated in each iteration by the equation

$$\underline{w}_{k+1} = \underline{w}_k - \alpha_k \cdot M_k \cdot \underline{g}_k$$

where α_k is the learning rate, $\underline{g}_k$ is the gradient of the error function and M_k is an approximation of the inverse of the Hessian matrix. This matrix is positive definite in order to ensure the descent. All these quantities are for the kth iteration. The negative product $-M_k \cdot \underline{g}_k$ gives the search direction, $\underline{d}_k$. Each multidimensional minimization algorithm (steepest descent, conjugate gradient, quasi-Newton and Levenberg–Marquardt algorithm) calculates the search direction $\underline{d}_k$ in a different way. The first three are general optimization methods that simply try to minimize a quadratic error function. Of course most error surfaces are not quadratic but they will be so in sufficiently small neighborhoods of local minima. The fourth algorithm needs the Jacobian matrix to be calculated and is specifically adapted to minimize an error function that arises from a quadratic criterion of the form we are assuming.

In the steepest descent algorithm $M_k = I$, where I is the identity matrix, so the search for a minimum takes the opposite direction of the gradient ($-\underline{g}_k$), i.e. the direction of the steepest descent of the error function's 'surface'. The conjugate gradient algorithm is similar to the steepest descent, but the search directions are noninterfering in the sense that successive minimizations along these directions do not undo the progress made by previous minimizations. This is achieved if the gradient $\underline{g}_k$ at each iteratively selected parameter value $\underline{w}_k$ is orthogonal to all previous search directions $\underline{d}_{k-1}, \underline{d}_{k-2}, \ldots, \underline{d}_1$. Thus, it is necessary to have:

$$(\forall i \leq k-1) \cdot \underline{g}_k^{\mathrm{T}} \underline{d}_i = 0.$$

In quasi-Newton algorithm the positive definite approximation of the inverse of the Hessian M_k satisfies

$$\underline{p}_k = M_k \cdot \underline{q}_k$$

where $\underline{p}_k = \underline{w}_{k+1} - \underline{w}_k$ and $\underline{q}_k = \underline{g}_{k+1} - \underline{g}_k$. In the algorithm used in this work, the approximation was made using the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method (Fletcher, 1987; Suykens, Vandewalle, & De Moor, 1996). In the Levenberg–Marquardt algorithm, the inverse of the Hessian (M_k) is approximated by the quantity

$$[\varepsilon \cdot I + J \cdot J^{\mathrm{T}}]^{-1}$$

where J is the Jacobian matrix $[J_{ij}]$ with $J_{ij} = (\partial e_j / \partial w_i)$ and ε is a positive value left to experimentation (the larger its value is, the closer the algorithm gets to simple steepest descent).

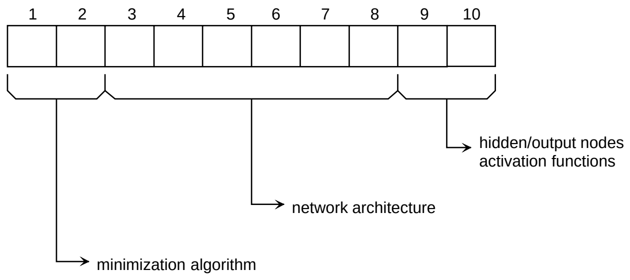

Fig. 1. Binary representation of NN topology and training parameterization.

A crucial point in all versions of the backpropagation algorithm is the choice of the learning rate α, which must be such that minimizes the error of the next step (iteration). In the large majority of works in the literature, a fixed value learning rate is used during the training of the NN. In order to minimize the manually-explored parameters during NN training, an on-line adjustable learning rate was used in this work. In the steepest descent algorithm, the Hessian was used to solve for the 'best' learning rate at each iteration. Because of the high computational demands of calculating the Hessian, for the other three algorithms the 'best' learning rate was calculated with an approximate line search using a cubic interpolation.

These four algorithms can be represented with two binary entries, as follows:

00: steepest descent
01: quasi-Newton
10: Levenberg–Marquardt
11: conjugate gradient

and they form the first two bits of the binary string representation (Fig. 1).

4.1.2. Representation of the network architecture

The next six binary entries of the string (Fig. 1) represent 64 possible network architectures of one-hidden-layer (1-HL) and two-hidden-layer (2-HL) networks. The choice of the range of possible architectures is case-specific. This is why some a priori knowledge about the model system is required. This knowledge helps limit the finite search space of network architectures. In both applications that were used for testing and validation of the system (see Section 4.2), the same range of architectures were used (the correspondences are shown in Table 1), because the systems to be modeled were based on the same biological process (plant cultivation in hydroponic systems). In general, in NN applications where a wider search space of possible architectures is initially required, more bit strings can be added to this category of strings, or the search space can be grouped into 64 categories and the entire GA system can be reapplied in 64 possible architectures that are close to the category that gave the best results in the first place. This can be repeated as many times as necessary, according to the initial size of the search space. However, the size of 64 possible architectures is rather enough in most applications for which some a priori knowledge about the modeled system exists.

Table 1
Correspondences between binary string representations and NN architectures

Bit sequence	Architecture	Bit sequence	Architecture	Bit sequence	Architecture
000000	1-HL–2	010110	1-HL–24	101100	2-HL–5/6
000001	1-HL–3	010111	1-HL–25	101101	2-HL–5/7
000010	1-HL–4	011000	1-HL–26	101110	2-HL–5/8
000011	1-HL–5	011001	1-HL–27	101111	2-HL–6/3
000100	1-HL–6	011010	1-HL–28	110000	2-HL–6/4
000101	1-HL–7	011011	1-HL–29	110001	2-HL–6/5
000110	1-HL–8	011100	1-HL–30	110010	2-HL–6/6
000111	1-HL–9	011101	2-HL–3/3	110011	2-HL–6/7
001000	1-HL–10	011110	2-HL–3/4	110100	2-HL–6/8
001001	1-HL–11	011111	2-HL–3/5	110101	2-HL–7/3
001010	1-HL–12	100000	2-HL–3/6	110110	2-HL–7/4
001011	1-HL–13	100001	2-HL–3/7	110111	2-HL–7/5
001100	1-HL–14	100010	2-HL–3/8	111000	2-HL–7/6
001101	1-HL–15	100011	2-HL–4/3	111001	2-HL–7/7
001110	1-HL–16	100100	2-HL–4/4	111010	2-HL–7/8
001111	1-HL–17	100101	2-HL–4/5	111011	2-HL–8/4
010000	1-HL–18	100110	2-HL–4/6	111100	2-HL–8/5
010001	1-HL–19	100111	2-HL–4/7	111101	2-HL–8/6
010010	1-HL–20	101000	2-HL–4/8	111110	2-HL–8/7
010011	1-HL–21	101001	2-HL–5/3	111111	2-HL–8/8
010100	1-HL–22	101010	2-HL–5/4		
010101	1-HL–23	101011	2-HL–5/5		

1-HL-*x*: one-hidden-layer NN with *x* hidden nodes; 2-HL-*x*/*y*: two-hidden-layer NN with *x* hidden nodes in the first hidden layer and *y* hidden nodes in the second hidden layer.

Table 2
Binary encoding of activation function combinations for hidden and output nodes of the NN

Bit sequence	Hidden nodes' activation function	Output nodes' activation function
00	Logistic	Logistic
01	Logistic	Linear summation
10	tanh	tanh
11	tanh	Linear summation

4.1.3. Representation of the activation functions

Two activation functions were considered for the hidden nodes of the NN: the logistic function and the hyperbolic tangent function. In addition, the output nodes were allowed to have either the same activation function as the hidden nodes, or a linear summation function. Thus, four different combinations were included in the genetic representation and were encoded with two final binary entries in the bit string, as shown in Table 2.

Thus, each binary string (genotype) of an individual of the GA that represents a specific phenotype of NN architecture and training parameterization, has a total of 10 bits, as shown in Fig. 1.

4.2. The algorithm

The algorithm of the developed system for NN design and training parameterization consists of three main parts: the 'user level' part, the 'genetic algorithm optimization' part and the 'training' part. The first part deals with the input/output processes, while the other two parts, which contain several sub-sections, interact with each other and with the first part to complete the desired procedure. The algorithm is shown schematically in Fig. 2. Each box in Fig. 2 represents a separate function and the names of these functions are shown at the top of each box. The interactions between functions are shown with the appropriate arrows. An explanation of the algorithm and the symbols involved follows.

Initially, the user gives some parameters of the algorithm, which are: the training set of the neural network (T), a set of parameters for the backpropagation training algorithms (param), the number of generations of the GA (G), the size of population of the GA (M) and, finally, the probabilities of crossover (P_c) and mutation (P_m) of the GA. An initial population of several random binary strings, each of which represents a specific network topology and set of training parameters, is generated ($X_{initial}$). All this

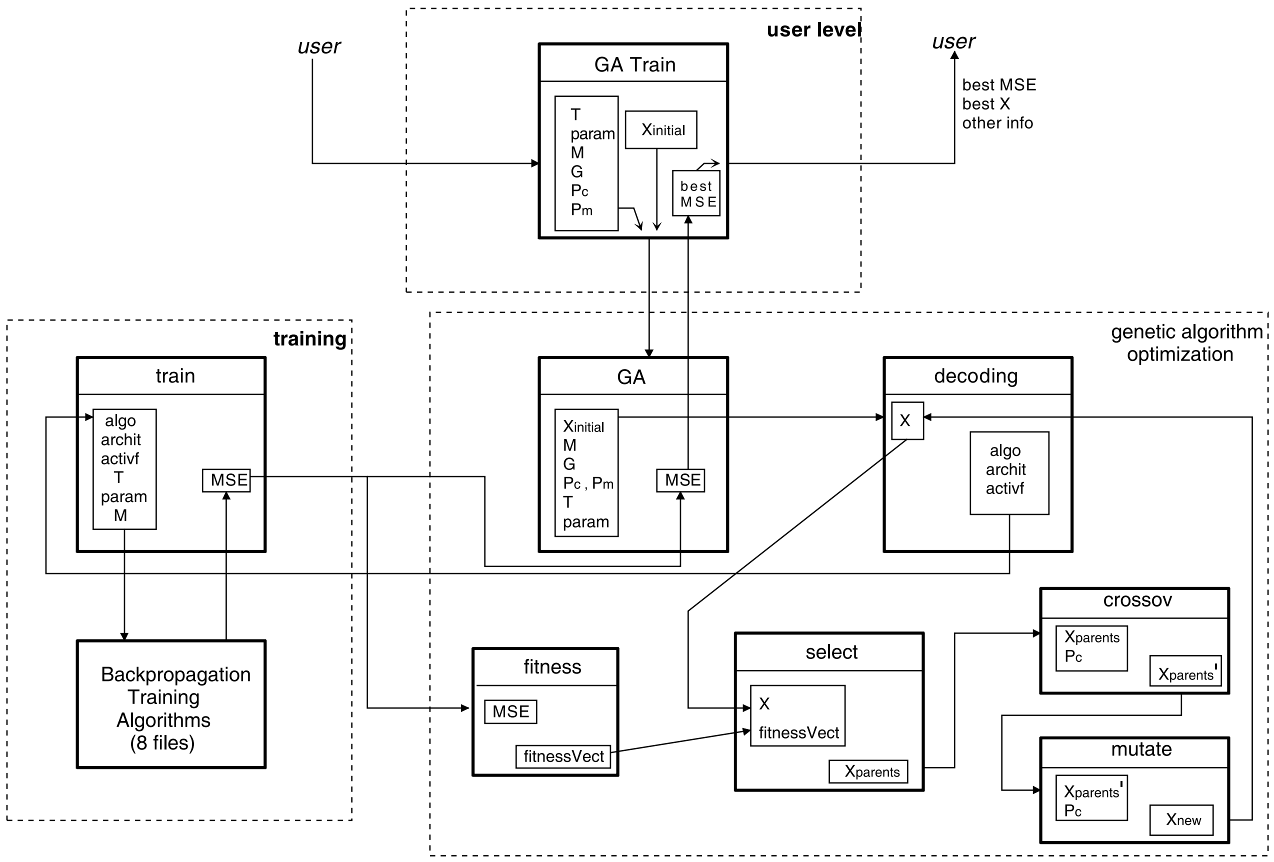

Fig. 2. Schematic representation of the algorithm of the genetic algorithm system for NN design and training parameterization.

information is passed to the 'GA' function, which is the main part of the genetic algorithm. The population of binary strings is passed to the 'decoding' function. There, each string of binary values is decoded into explicit information about the three parts that are shown in Fig. 1, that is, the minimization algorithm (algo), the network architecture (archit) and the activation functions of the network (activf). This explicit information goes to the 'train' function. There, each network topology with the specific activation functions is trained with the appropriate training algorithm. The mean squared error of each individual string (MSE) after training is calculated and sent to the 'fitness' function where the fitness of each sting is calculated (fitnessVect). The fitness is simply the value that the GA tries to maximize. Here, the fitness of each individual string is the negative of the MSE but, in order to keep its values always positive, the final fitness is given by the formula

$$\text{fitness} = 10^6 - \frac{\sum_{i=1}^{n} ||y_i - t_i||^2}{n}$$

where y_i are the NN output values for the n available training samples and t_i are the corresponding real system outputs. According to this fitness, the selection process of the GA (function 'select') selects the new group of strings, which constitute the parents of the next generation of the GA (X_{parents}). Each string is assigned a probability of reproduction (Fogel, 1995) and is selected according to this probability. The probability is usually proportional to the fitness of each string. These strings are then subject to the evolutionary operations of crossover and mutation, after which the final population is formed (X_{new}). This process repeats until the maximum number of generations (G) is reached. After that, the best string, that is the string that gave the maximum fitness (or, the minimum MSE), is returned to the user, together with its corresponding minimum MSE (bestMSE) and some other information useful for statistical analysis.

During training, the two most common techniques of controlling the complexity of large neural networks, validation and regularization, were used (Fine, 1999). By including these techniques in the training part of the algorithm, the GA system itself did not aim in designing the simplest possible network but rather, the most accurate one. The simplicity of the network was considered by including validation and regularization within the training processes. More specifically, the two methods can be described as follows:

- *Validation*. A part of the training set was used as validation set, in order to keep track of the validation error. Random samples from the training set were used to form the validation set of each application, while the size of the validation set was equal to 10% of the corresponding training set. The validation error decreases at the beginning of training, but at some point it starts to increase even though the training error still decreases. The best parameters of the network were those of a minimum validation error, so the learning was stopped at that point. In this way, overfitting of the data was avoided.
- *Regularization*. A penalty term was added to the error. The term used here was $\lambda \cdot ||w||^2$, where $\underset{\sim}{w}$ is the network parameters' vector, so the complexity of the network was penalized. The most appropriate value of λ was determined experimentally during the training process. Several values of λ were tested for each algorithm (from 10^{-4} to 10^{-1}, in orders of 10), steering between variance and bias, until the best results occurred.

Finally, during training applications, both inputs and outputs (target values) were standardised to have mean zero and standard deviation one. The initial conditions, that is, the initial weights and biases of the NN, were uniformly generated (randomly) in the range $(-1/\sqrt{d}, 1/\sqrt{d})$ (Duda, Hart, & Stork, 2001), where d is the number of network inputs, and several experiments with different initial weights and biases were performed in order to take the best possible results.

5. Results

This section presents the results of the application of the previously described GA system on the design and training parameterization of the two NN models for fault detection and predictive modelling of hydroponic systems. The performance and generalization capabilities of the GA-generated models are investigated and compared with those of models developed with the common technique of trial-and-error.

5.1. Fault detection system application results

The training set for the fault detection neural network (FDNN) model, with data collected as described in Section 2, was fed into the GA system. The GA methodology is a stochastic optimization technique. Therefore, the entire optimization process must be repeated several times, starting from different random initial populations each time. The basic parameters of GAs that must be explored are the population size M, the probability of crossover P_c, the probability of mutation P_m and the number of generations G. A number of experiments had to be carried out to determine the best possible parameters. The type of crossover used was the most common one, the one-point crossover (Goldberg, 1989a). The first experimentation dealt with the determination of the best values of P_c and P_m. These runs of the system were made with a population size of 20 strings and for 30 generations. The best performance was achieved with a value for P_c of 0.9 and for P_m of 0.05. After the best solution was found by the optimal (concerning its parameters) GA system, that

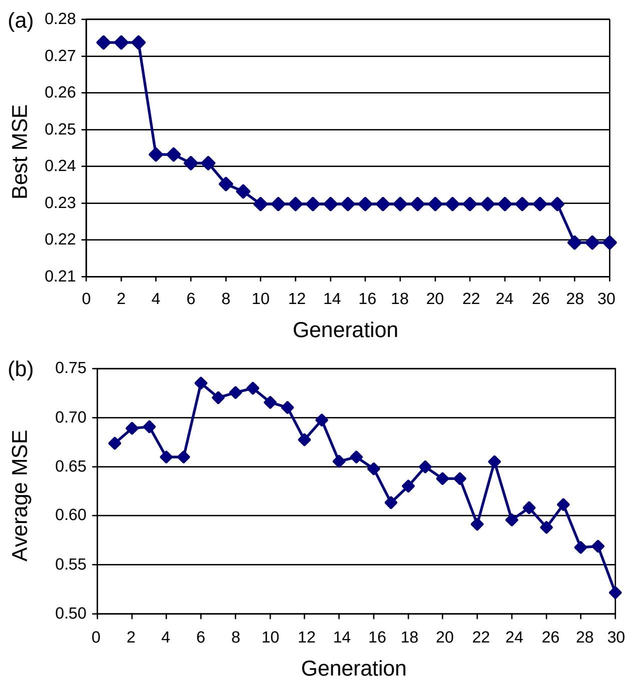

Fig. 3. (a) Best mean squared error (MSE) found during the best genetic algorithm run for the fault detection system; (b) average mean squared errors of the entire population during the best genetic algorithm run.

solution, i.e. best network architecture, training algorithm and activation functions combination, was further trained. Finally, the value of the FDNN model was evaluated on the performance achieved in new, testing data sets, with samples that contained specific faults or normal data.

Fig. 3a shows the best MSEs found after each generation during the best run of the GA system. In this run, the best solution was found after 27 generations. In Fig. 3b, the corresponding average MSEs of the entire population after each generation are shown. One can see that the average performance of the population generally improves. The best solution found was the string [0 0 0 1 1 0 1 0 1 1], which is interpreted as a 1-HL NN with 28 nodes in the hidden layer, hyperbolic tangent activation functions in hidden nodes and linear summation functions in output nodes, trained with the steepest descent backpropagation algorithm. This solution gave a value for the MSE of 0.2193. The second and third best solutions, with slightly worse performances, were 1-HL networks with 22 and 18 nodes, respectively. They were each trained with the steepest descent algorithm and each had hyperbolic tangent activation functions in hidden nodes and linear summation functions in output nodes. The values for their MSEs were 0.2220 and 0.2297, respectively. From these results it seems that generally the steepest descent algorithm was the most appropriate algorithm for the specific application and, in addition, hyperbolic tangent activation functions in hidden nodes and linear summation in output nodes had better performance than the other combinations of activation functions. Also, one hidden layer architectures seemed to outperform in general the two hidden-layer ones.

Thus, the final neural network designed by the GA system consisted of 15 inputs, one hidden layer with 28 nodes and 5 outputs. It contained hyperbolic tangent activation functions in hidden nodes and linear summations in output nodes and was trained with the steepest descent algorithm. Further training was performed in this NN, with several random initial values of weights and thresholds and a variety of algorithm parameters. The MSE of the final NN was 0.1148 after 1000 iterations.

The testing process on the FDNN consisted of presenting new data sets to the network, namely the testing sets, each of which contained some specific fault imposed at the moment that the set started, and exploring its performance. In addition, testing sets that contained only normal data were included in the testing process, to investigate the ability of the network to avoid false alarms. The testing tests were distributed throughout the entire period of data collection. Twelve data sets were constructed. The first three contained fault type 1 (pH control pump failure), the next six contained fault types 2 (circulation pump failure), 3 (pH sensor failure) and 4 (EC sensor failure), respectively, in pairs, and the last three data sets represented normal operation and differed from each other by periods of 1 month. The decision support approach that was used to decide whether a fault has been indicated or not, in order to initially evaluate the performance of the NN model, was the following: output values above 0.6 indicate a fault, values below 0.4 indicate normal operation and values in the interval [0.4, 0.6] indicate the previously known condition of the output.

Table 3 shows the percentages of the testing sets of each fault type that were classified correctly (that is, the corresponding fault was detected) according to the time of detection after each fault was imposed. For example, within

Table 3
Percentages of testing sets of each fault that were correctly classified after specific time periods from the initiations of the faults

Fault type	Correct classification of fault (%)								
	Test period (h)								
	0.17	0.33	0.5	1	1.5	3.5	6	8	13
Actuator fault, type 1	0	0	0	0	0	0	33	66	100
Mechanical fault, type 2	50	100							
pH sensor fault, type 3	0	0	0	0	50	100			
EC sensor fault, type 4	0	50	100						

EC, electrical conductivity.

Table 4
Classification percentages of data samples of normal and faulty operations

Tested data set of type	Classification (%)					
	Normal	Fault 1	Fault 2	Fault 3	Fault 4	Unknown fault
Normal	99.2	0.2	0.4	0.2	0	0
Fault 1	25.5	70.1	0.2	0	4.2	0
Fault 2	1.9	0	92.4	0	3.8	1.9
Fault 3	0	0	1.5	92.1	3.9	2.5
Fault 4	1.8	0	1.7	2.4	92.9	1.2

Fault 1: actuator; fault 2: mechanical; fault 3: pH sensor; fault 4: electrical conductivity sensor.

1.5 h, 100% of the testing sets containing faults 2 and 4 were correctly classified (i.e. their faults were detected), 50% of the testing sets containing fault 3 were correctly classified and none of the testing sets containing fault 1 was correctly classified. Fault types 2 and 4 were the most rapidly detected ones. Fault type 2 (circulation pump failure) was detected in average after 10–20 min and fault type 4 (EC sensor failure) was detected after 20–30 min. The detection of fault type 3 (pH sensor failure) took much longer; this fault was detected in 1.5–3.5 h. Finally, fault type 1 (pH control pump failure) was the slowest detected fault, as it was detected on average more than 9 h after the fault occurred. However, it should be noted here that this specific fault is situation-dependent because the pH control pump does not operate continuously. Thus, if in specific situations the pump does not operate, the fault cannot be detected. When the pump should have started to operate, the fault was usually detected within 30 min.

Table 4 shows the classification probabilities of all normal and faulty data samples into the 'normal operation', the four fault types, or the 'unknown fault' case. The grey boxes represent values of the correct classification probabilities. The percentages in the last column represent the probabilities of 'unknown fault' classification. All other entries represent values of misclassification probabilities (or false alarm probabilities in the 'normal operation' column).

Fig. 4 shows some examples of the NN model outputs during four testing data sets, one for each type of fault, The time step is 10 min. It should be noted here that each fault in these sets starts at the beginning of the tests. This means that the first interval shown in each plot represents the outputs of the network for measurements taken 10 min after the initiation of the fault. In Fig. 4a, the outputs of the network during the existence of fault type 1 (pH control pump failure) are shown. In this specific data set, the fault was detected in about 8 h after its occurrence, when the values of the first output of the NN (normal operation) fall to near zero and the values of the second output (fault type 1) increase drastically. The faulty indication of 'fault type 2', failure in the circulation pump, shown as a peak in the third NN output around interval 35, cannot be interpreted as an actual false alarm because at the following time step, that output value has returned to normal values. Fig. 4b shows the NN outputs during the exploration of a testing set that contains data during the existence of 'fault type 2', which is the circulation pump failure. The fault was detected 20 min after its occurrence (NN output 3). The values of 'normal' and 'fault type 4' outputs were quite high during the entire data set, but with the exception of 'fault type 4' output towards the end of the set, they were kept below the upper threshold of 0.6. In Fig. 4c, the NN outputs during a testing set that contains data during the existence of 'fault type 3' (pH sensor failure) are shown. The fault was detected after approximately 1.5 h. However, from the moment that the fault was imposed, the output that corresponds to the fault (output no. 4), was continuously giving values around 0.5, while the output of 'normal' operation was almost zero. This indicated that something wrong was happening from the beginning. As before (Fig. 4a), some isolated instances of some outputs cannot be interpreted as indications of the corresponding faults. However, there seems to be a periodic reduction (but not below 0.5) of the values of 'fault type 3' output. This can be explained by the nature of the imposed pH sensor fault, which was a periodic sine-wave noise added to the sensor readings. In that way, at periods when the noise was close to zero, the faulty indication returned to normal values. However, normal operation was not indicated for those periods, thus the system 'knew' that there was something wrong, even though the pH values were normal. In addition, this effect is not crucial, as the fault had already been detected and diagnosed. Fig. 4d shows the NN outputs during a testing set that contains data during the existence of 'fault type 4', which is the EC sensor failure. The fault was detected after 20 min. Before that, an instantaneous indication of 'fault type 3' was given, but the value of that output returned to normal in the next time step. The periodic fluctuations of the 'fault type 4' output and, at the same time, the indication of normal operation (output no. 1) happened, like in the case of the pH sensor fault, because of the periodic nature of the imposed fault, which consisted of adding a sine-wave noise to the sensor readings. At periods when the noise was close to zero, the NN outputs indicated normal operation. This effect was stronger in this type of fault than it was in the pH sensor fault because EC

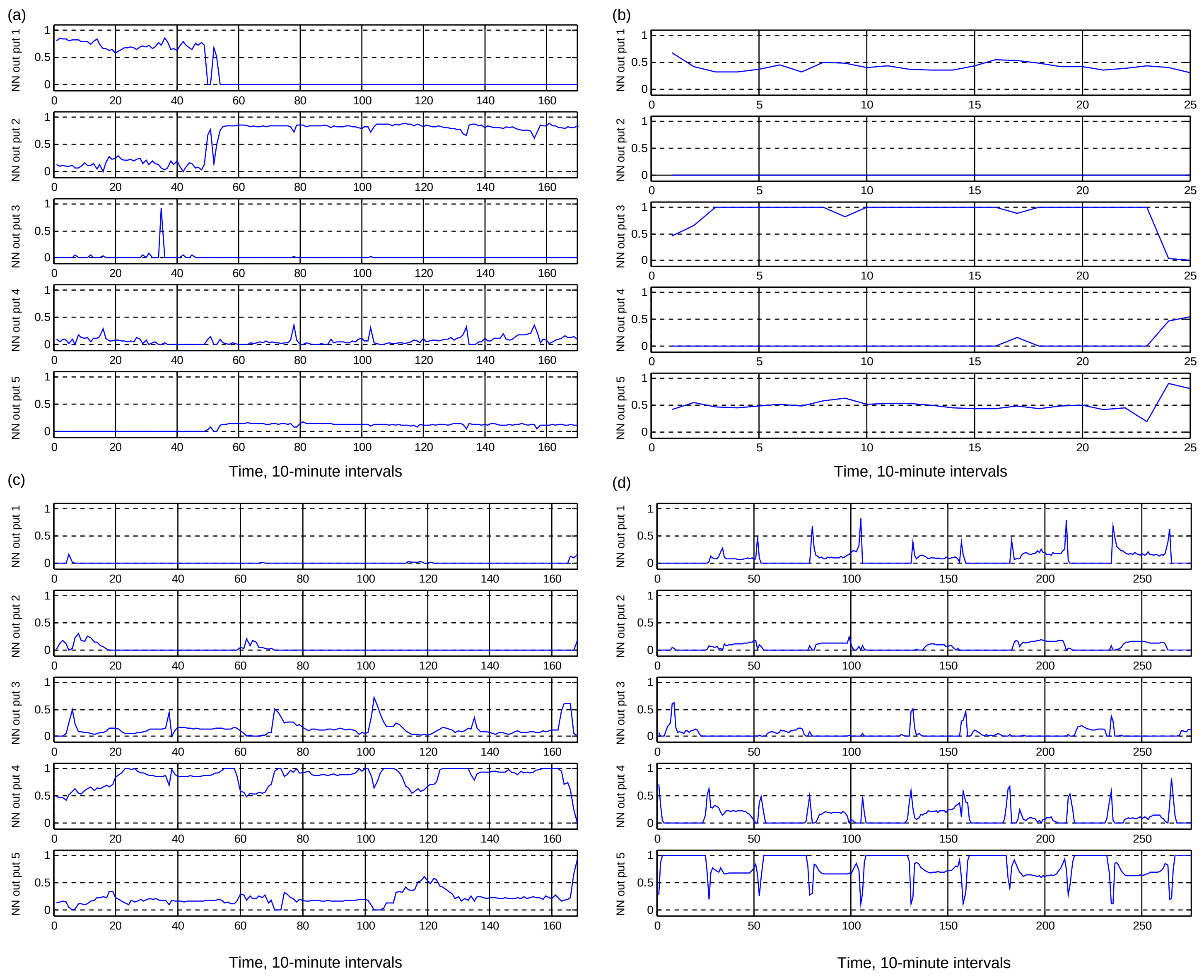

Fig. 4. NN outputs during a: (a) ‘Fault type 1’ data set, (b) ‘Fault type 2’ data set, (c) ‘Fault type 3’ data set, (d) ‘Fault type 4’ data set. Output 1 is for normal operation, outputs 2–5 are for fault type 1–4, respectively.

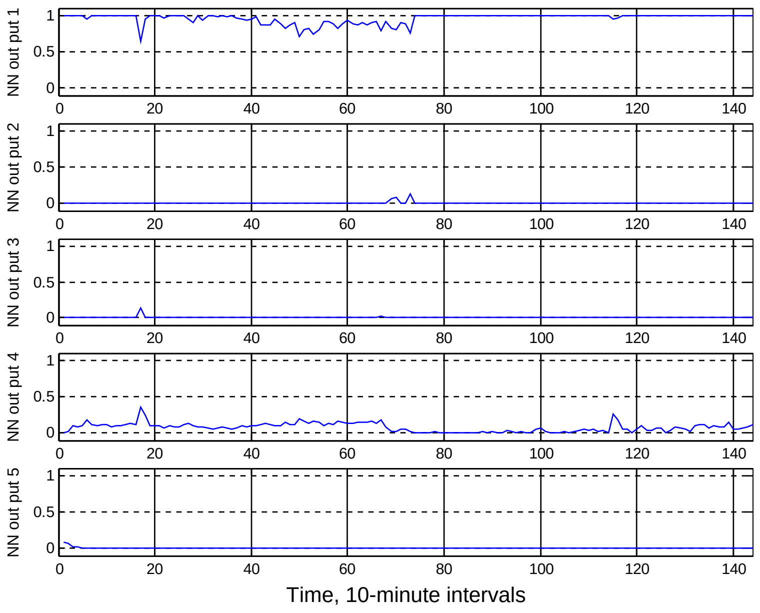

Fig. 5. NN outputs during a data set of normal operation. Output 1 is for normal operation, outputs 2–5 are for fault type 1–4, respectively.

was not controlled automatically. Thus, the fault detection system did not have additional information concerning EC, as it had for pH, from the values of the pH control actuator (pH control pump). That additional information gave to the NN the capability of not indicating normal operation during those periods in the case of the pH sensor ('normal' output always close to zero); something that did not happen in the case of the EC sensor fault. Finally, Fig. 5 shows the NN outputs during a testing set that contains data of normal operation. The normal operation was indicated throughout the entire set, without any false alarms.

From these results, it is evident that the GA system was capable of finding a good solution that gave satisfactory accuracy in training and testing of the NN. A direct comparison of the solutions achieved by the GA system and by a trial-and-error approach cannot lead to some absolute decision on which is the best method. However, it can give an insight into the degree of success of the GA system. Thus, a preliminary training was performed, using the traditional trial-and-error experimentations, in order to find the best network architecture and training algorithm combination. Any information extracted from the results of the GA system that had already been run was assumed to be unknown. However, because it was practically unfeasible to explore these combinations with all four possible combinations of activation functions for the hidden and output nodes ('logistic/logistic', 'logistic/linear summation', 'tanh/tanh' and 'tanh/linear summation'), only the combinations containing linear summation functions were explored. This could be considered as 'cheating' because no previous information or experience existed or could exist for this decision, which was completely based on the results of the GA system which gave clear indication that the 'logistic/logistic' and 'tanh/tanh' combinations seemed to perform poorly compared to the ones that had linear summation functions in the output nodes.

The preliminary explorations mainly focused on 1-HL architectures because most of the 2-HL ones that were investigated at the beginning of the experimentations performed poorly compared to the results of the 1-HL networks. The best solution according to the trial-and-error approach was the 1-HL NN with 30 hidden nodes, hyperbolic tangent activation functions in hidden nodes and linear summation functions in output nodes, trained with the steepest descent algorithm. This solution gave a value for the MSE of 0.2081, which is slightly better than the best solution of the GA system, and is the same as the fourth best solution of the GA system. Also, the second best solution of the trial-and-error method was the one that was actually found by the GA system and was used as the final network (1-HL with 28 hidden nodes). These results are summarized in Table 5.

Table 5
Characteristics of best networks found with the GA system and the trial-and-error procedure for the fault detection application

	Trial-and-error	GA system (first choice)	GA system (second choice)	GA system (third choice)
Architecture	1-HL	1-HL	1-HL	1-HL
Nodes	30	28	22	18
Activation fn.	tanh/L.S.	tanh/L.S.	tanh/L.S.	tanh/L.S.
Training alg.	SD	SD	SD	SD
MSE	0.2081	0.2193	0.2220	0.2297

MSE, mean squared error; L.S., linear summation; tanh, hyperbolic tangent; SD, steepest-descent algorithm.

The high similarity of the results given by the two approaches is an indication that the performance of the GA system was indeed successful. Only the fourth solution of the trial-and-error approach proposes a different training algorithm (conjugate gradient algorithm), but its performance is much worse than the performances of the GA system solutions. In addition, if hyperbolic tangent activation functions had not been used in the trial-and-error approach, then the best solution of the trial-and-error method would have been much worse than the solution of the GA system.

5.2. Predictive modelling application results

The training set for the predictive modelling application, with data collected as described in Section 2, was fed into the GA system. A number of experiments were carried out to determine the best possible GA parameters. The type of crossover used was again the one-point crossover (Goldberg, 1989a,b,c). The results of that preliminary experimentation led to the following optimal parameters of the GA: population size of 20 individuals, maximum number of generations equal to 30, probability of crossover $P_c=0.9$ and probability of mutation $P_m=0.08$. Fig. 6a shows the best MSEs found after each generation during the best run of the GA system. In this run, the best solution was found after 25 generations. In Fig. 6b, the corresponding average MSEs of the entire population after each generation are shown. Again, it is evident that the average performance of the population generally improves.

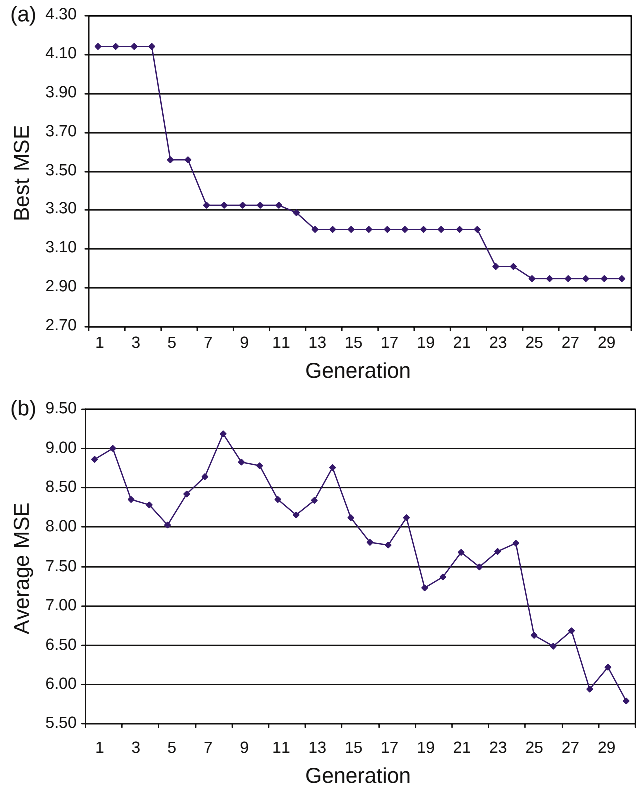

Fig. 6. (a) Best mean squared errors (MSEs) found during the best genetic algorithm run for the predictive modelling system; (b) average mean squared errors of the entire population during the best genetic algorithm run.

Table 6
Characteristics of best networks found with the GA system and the trial-and-error procedure for the predictive modeling application

	Trial-and-error	GA system (first choice)	GA system (second choice)	GA system (third choice)
Architecture	1-HL	1-HL	1-HL	1-HL
Nodes	9	30	18	18
Activation fn.	log/L.S.	tan h/L.S.	log./L.S.	tan h/L.S.
Training alg.	QN	QN	QN	QN
pH MSE	4.6×10^{-2}	3.5×10^{-2}	4.4×10^{-2}	4.6×10^{-2}
EC MSE	3.3750	2.9451	3.3197	3.2411

MSE, mean squared error; log, logistic; L.S., linear summation; tanh, hyperbolic tangent; QN, quasi-Newton algorithm.

The best solution found was the string [0 1 0 1 1 1 0 0 1 1], which is interpreted as a 1-HL NN with 30 nodes in the hidden layer, hyperbolic tangent activation functions in hidden nodes and linear summation functions in the output nodes, trained with the quasi-Newton backpropagation algorithm. This solution gave a value for the MSE of 3.5×10^{-2} for the pH and 2.9451 μS cm^{-1} for the EC, in the training data set. The details of this optimum selection together with the next two selections of the GA system and the best network of the trial-and-error procedure are shown in Table 6. From these results, it seems that generally the quasi-Newton algorithm was the most appropriate algorithm for the specific modeling application. Also, 1-HL architectures seemed to outperform in general the 2-HL ones. Finally, the best performance is given by networks with linear summation functions in the output nodes. The comparison of the GA-constructed networks' performance in the training set with that of the trial-and-error, shows that the GA system found larger networks to be more appropriate for the modeling task. The trial-and-error network had 9 hidden nodes, while the GA-constructed networks have 30 and 18 hidden nodes. The training MSEs of all three networks proposed by the GA system are smaller than the training error of the best NN found by the trial-and-error procedure.

Thus, the final NN model constructed by the GA system consisted of nine inputs, one hidden layer with 30 nodes and two outputs. It contained hyperbolic tangent activation functions in hidden nodes and linear summations in output nodes and was trained with the quasi-Newton backpropagation algorithm. Further training was performed in this NN, with several random initial values of weights and thresholds and a variety of algorithm parameters, until the final network was achieved.

The final NN model was then tested on new data, namely the testing data set. The performance of the network on this new data was again better than the performance of the trial-and-error constructed NN model. Fig. 7 shows the frequency distributions of the absolute prediction errors on the testing data (Fig. 7a) and its cumulative frequency

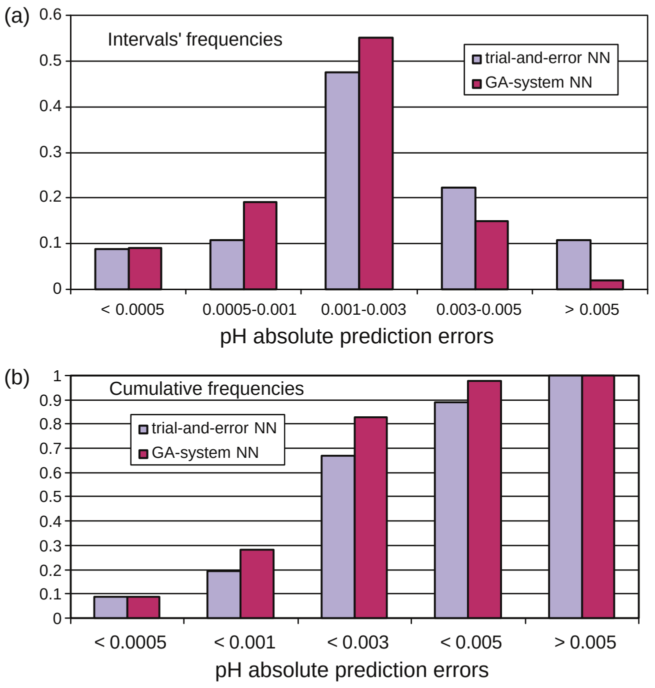

Fig. 7. Intervals' (a) and cumulative (b) frequency distributions of absolute prediction errors on testing data for pH, for the two differently constructed NNs.

distributions (Fig. 7b), for the pH. The results of both the NNs constructed by the trial-and-error method and the GA-system method are shown. The corresponding frequency distributions and cumulative frequency distributions for the EC, for both networks, are presented in Fig. 8.

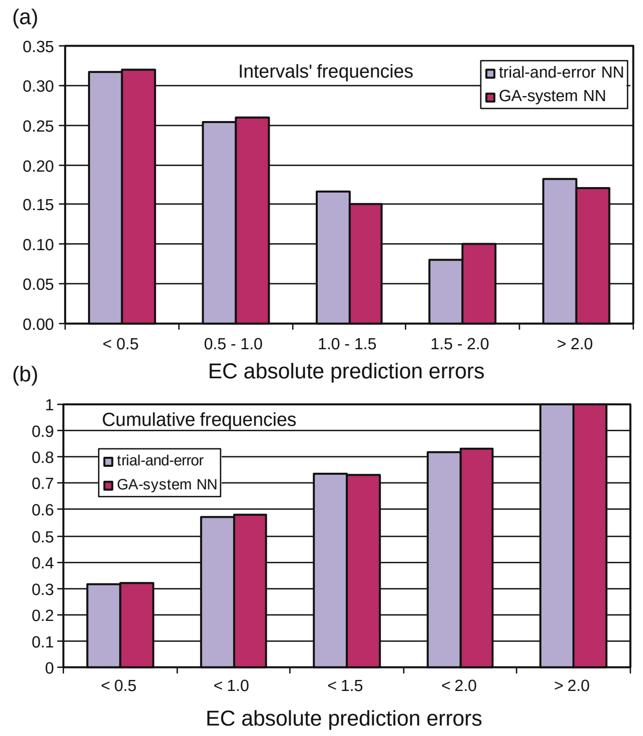

Fig. 8. Intervals' (a) and cumulative (b) frequency distributions of absolute prediction errors on testing data for EC, for the two differently constructed NNs.

In these figures, the generalization capabilities of the two NN models can be observed. More specifically, about the NN constructed by the GA system, it can be observed that:

- About 98% of prediction errors for pH differed from real measurements by less than 0.005, more than 80% by less than 0.003 (Fig. 7b), while 55% were between 0.001 and 0.003 pH units (Fig. 7a).
- 83% of prediction errors for EC differed from real measurements by less than 2.0 μS cm^{-1}, almost 75% by less than 1.5 μS cm^{-1} (Fig. 8b), while more than 25% were between 0.5 and 1 μS cm^{-1} (Fig. 8a).

By comparing the two differently constructed NN models from their results as shown in Figs. 7 and 8, one can report that:

- The NN constructed by the GA system has considerably higher frequencies in the lower error intervals for the pH predictions than the NN constructed by the trial-and-error procedure (Fig. 7a). This is also shown in the cumulative frequencies (Fig. 7b), where for example, 83% of the pH absolute errors are below 0.003 for the GA-system constructed NN model, while the corresponding percentage of the trial-and-error NN is about 65%.
- The performances of the two NN models as far as the EC predictions are concerned, are quite similar. However, again the GA-system constructed NN seems to give slightly lower errors, as the lower error intervals have slightly larger frequencies for this model.

6. Conclusions

Two real-world biological engineering applications of neural networks were presented. An evolutionary approach to automated feedforward neural network design and training parameterization based on genetic algorithms was used. The design consisted of the appropriate selection of network topology and types of activation functions of the hidden and output network nodes, while the training parameterization consisted of the appropriate selection of multidimensional minimization algorithm of the back-propagation training. The method is general enough to permit the inclusion of more architectures than those presented here, more possible types of activation functions and even more different variations of the backpropagation algorithm. Some general advantages and disadvantages of the proposed GA system can be drawn as follows:

Advantages of the GA system:

– It is based on some optimization heuristic for combinatorial problems.
– It is automated, thus it requires much less human effort than trial-and-error.

– It incorporates some user experience about the modelling problem (mainly in the choice of the range of network architecture search space).
– It extends the user experience by using the evolutionary properties of the GA optimization.
– It is very robust in applications of feedforward neural networks.
– It can exploit the developing technology of parallel processing.

Disadvantages of the GA system:
– It is not fully automated, as the parameters of the genetic algorithm have to be adjusted.
– It can be trapped in local minima if the initial population is not good enough.

The complexity of the application systems developed in this work is such that the trail-and-error approach to network design and parameterization can become problematic. The high accuracy of the developed NN models in both applications, demonstrates the good performance of the GA system. The results of the comparisons showed that the developed GA system should be preferred over the traditional trial-and-error approach, mainly because of its higher degree of confidence that some near optimal solution of the problem of network design and training parameterization will be found. This is ensured because of the optimization principles that the GA system is based on. In addition, it is an automated system, whose results could be used as initial guidance for a more detailed trial-and-error effort for designing and training parameterization.

It should be mentioned that the developed GA system is not fully automated. There are still some exploration parameters in the process, as GAs are stochastic algorithms and several parameters, like probabilities of crossover and mutation and population size, have to be tuned. However, the exploration of these parameters is much simpler than the search for optimality in the parameters involved in the construction and parameterization of NN models, because the influence of the variation of the GA parameters is not as significant to the final performance of the system as that of the parameters of NN design and training.

Acknowledgements

The author would like to thank Prof. Bart Selman of the Computer Science Department at Cornell University and Professors Louis D. Albright and Norman D. Scott of the Department of Biological and Environmental Engineering, for their helpful advice on this research.

References

Altendorf, C. T., Elliott, R. L., Stevens, E. W., & Stone, M. L. (1999). Development and validation of a neural network model for soil water content prediction with comparison to regression techniques. *Transactions of the ASAE*, *42*(3), 691–699.

Angeline, P. J., Saunders, G. M., & Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, *5*(1), 54–65.

Bebis, G., Georgiopoulos, M., & Kasparis, T. (1997). Coupling weight elimination with genetic algorithms to reduce network size and preserve generalization. *Neurocomputing*, *17*, 167–194.

Bhat, N. V., Minderman, P. A., McAvoy, T. J., & Wang, N. S. (1990). Modeling chemical process systems via neural computation. *IEEE Control Systems Magazine*, *24*(4), 24–30.

Blum, A. L., & Rivest, R. L. (1992). Training a 3-node neural network is NP-complete. *Neural Networks*, *5*(1), 117–127.

Castillo, P. A., Merelo, J. J., Prieto, A., Rivas, V., & Romero, G. (2000). G-Prop: Global optimization of multilayer perceptrons using GAs. *Neurocomputing*, *35*, 149–163.

Collins, R., & Jefferson, D. (1991). An artificial neural network representation for artificial organisms. In R. Manner, & D. E. Goldberg (Eds.), *Parallel problem solving from nature*. Heidelberg: Springer.

Dasgupta, D., & McGregor, D. R. (1992). Design application-specific neural networks using the structured genetic algorithm. In D. Whitney, & J. D. Schaffer (Eds.), *Proceedings of the international workshop on combinations of genetic algorithms and neural networks (COGANN-92)* (pp. 87–96). Los Alamitos, CA: IEEE Computer Society Press, 87–96.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). New York: Wiley.

Ferentinos, K. P., & Albright, L. D. (2002). Predictive neural network modeling of pH and electrical conductivity in deep-trough hydroponics. *Transactions of the ASAE*, *45*(6), 2007–2015.

Ferentinos, K. P., & Albright, L. D. (2003). Fault detection and diagnosis in deep-trough hydroponics using intelligent computational tools. *Biosystems Engineering*, *84*(1), 13–30.

Filho, E., & de Carvalho, A. (1997). Evolutionary design of MLP neural network architectures. *Proceedings of the fourth Brazilian symposium on neural networks* (pp. 58–65). Goiania, GO, Brazil.

Fine, T. L. (1999). *Feedforward neural network methodology*. New York: Springer.

Fletcher, R. (1987). *Practical methods of optimization*. New York: Wiley.

Fogel, D. B. (1994). An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, *5*(2), 3–14.

Fogel, D. B. (1995). *Evolutionary computation: Toward a new philosophy of machine intelligence*. New York: IEEE Press.

Frean, M. (1990). The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, *2*, 198–209.

Gallant, S. (1993). *Neural-network learning and expert systems*. Cambridge, MA: MIT Press.

Garcia-Pedrajas, N., Hervas-Martinez, N., & Munoz-Perez, J. (2003). COVNET: A cooperative coevolutionary model for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, *14*(3), 575–596.

Goldberg, D. E. (1989a). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison.

Goldberg, D. E. (1989b). Genetic algorithms and Welsh functions: Part I, a gentle introduction. *Complex Systems*, *3*, 129–152.

Goldberg, D. E. (1989c). Genetic algorithms and Walsh functions: Part II, deception and its analysis. *Complex Systems*, *3*, 153–171.

Hancock, P. J. B. (1992). Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification. In D. Whitney, & J. D. Schaffer (Eds.), *Proceedings of* *the international workshop on combinations of genetic algorithms and neural networks (COGANN-92)* (pp. 108–122). Los Alamitos, CA: IEEE Computer Society Press, 108–122.

Harp, S. A., Samad, T., & Guha, A. (1989). Towards the genetic synthesis of neural networks. In J. D. Schaffer (Ed.), *Proceedings of the third international conference on genetic algorithms*. Fairfax, VA: Morgan Kaufmann.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.

Hong, F., Tan, J., & McCall, D. G. (2000). Application of neural network and time series techniques in wool growth modeling. *Transactions of the ASAE*, *43*(1), 139–144.

Hwang, M.W., Choi, J.Y., & Park, J. (1997). Evolutionary projection neural networks. In: *Proceedings of the IEEE international conference on evolutionary computation (ICEC'97)* (pp. 667–671). Indianapolis, IN.

Iyoda, E. M., & von Zuden, F. J. (1999). *Evolutionary hybrid composition of activation functions in feedforward neural networks Proceedings of the IEEE international joint conference on neural networks (IJCNN'99)*, Vol. 6. Washington, DC: IEEE Computer Society Press.

Jiang, N., Zhao, Z., & Ren, L. (2003). Design of structural modular neural networks with genetic algorithm. *Advances in Engineering Software*, *34*, 17–24.

Judd, J. S. (1990). *Neural network design and the complexity of learning*. Cambridge, MA: MIT Press.

Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, *4*, 461–476.

Lacroix, R., Salehi, F., Yang, X. Z., & Wade, K. M. (1997). Effects of data preprocessing on the performance of artificial neural networks for dairy yield prediction and cow culling classification. *Transactions of the ASAE*, *40*(3), 839–846.

Liu, Y., & Yao, X. (1996). Evolutionary design of artificial neural networks with different nodes. In: *Proceedings of the IEEE international conference on evolutionary computation (ICEC'96)* (pp. 670–675). Japan: Nagoya.

Miller, G. F., Todd, P. M., & Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the third international conference on genetic algorithms* (pp. 379–384). Fairfax, VA: Morgan Kaufmann, 379–384.

Nowlan, S., & Hinton, G. (1992). Simplifying neural networks by soft weight sharing. *Neural Computation*, *4*(4), 473–493.

Parekh, R., Yang, J., & Honavar, V. (2000). Constructive neural-network learning algorithms for pattern classification. *IEEE Transactions on Neural Networks*, *11*, 436–450.

Phelan, R. M. (1977). *Automatic control systems*. Ithaca, NY: Cornell University Press.

Reed, R. (1993). Pruning algorithms—A survey. *IEEE Transactions on Neural Networks*, *4*, 740–747.

Reeves, C. R. (1995). *Modern heuristic techniques for combinatorial problems*. McGraw-Hill: UK.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536.

Schmitz, G. P. J., & Aldrich, C. (1999). Combinatorial evolution of regression nodes in feedforward neural networks. *Neural Networks*, *12*, 175–189.

Seginer, I., Boulard, T., & Bailey, B. J. (1994). Neural network models of the greenhouse climate. *Journal of Agricultural Engineering Research*, *59*, 203–216.

Sridhar, D. V., Seagrave, R. C., & Bartlett, E. B. (1996). Process modeling using stack neural networks. *AIChE Journal*, *42*(9), 2529–2539.

Suykens, J., Vandewalle, J., & De Moor, B. (1996). *Artificial neural networks for modelling and control of non-linear systems*. The Netherlands: Kluwer.

Weigend, A., Rumelhart, D., & Huberman, B. (1991). Generalization by weight elimination with application to forecasting. *Advanced Neural Information Processing Systems*, *3*, 875–882.

White, D., & Ligomenides, P. (1993). GANNet: A genetic algorithm for optimizing topology and weights in neural network design. In: *Proceedings of the international workshop on artificial neural network (IWANN'93)*. (pp. 322–327). *Lecture Notes in Computer Science*. Berlin: Springer.

Yao, X. (1993). A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, *8*, 539–567.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, *87*(9), 1423–1447.

Yao, X., & Liu, Y. (1997). A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, *8*(3), 694–713.

Yao, X., & Liu, Y. (1998). Towards designing artificial neural networks by evolution. *Applied Mathematics and Computation*, *91*, 83–90.