UNIVERSITÀ DEGLI STUDI
DI MILANO

DOTTORATO DI RICERCA IN INFORMATICA
XIX CICLO
SETTORE SCIENTIFICO DISCIPLINARE INF/01 INFORMATICA

A New Genetic Approach for Neural Network Design and Optimization

Tesi di Dottorato di Ricerca di:
Antonia Azzini

Relatore:
Prof. Andrea G.B. Tettamanzi

Coordinatore del Dottorato:
Prof. Vincenzo Piuri

Anno Accademico 2005/06

to my family,
to Maria and Angelo

Acknowledgments

First of all I would like to express my gratitude to my advisor, Prof. Andrea G.B. Tettamanzi, for his teaching, support and advices.

I would like to thank my referees, Stefano Cagnoni, Juan Julián Merelo Guervós and Xin Yao for their careful reading of my thesis and for their precious suggestions.

A particular thanks to my family for their love, support and patient, and to my grandparents Angelo and Maria, who are always in my heart.

Thanks to all my dear friends and to all the people who believe in me, and who have encouraged me, sharing joys and satisfactions, but also difficulties and disappointments.

Contents

Chapter 1

Introduction

1.1 Problem Definition

Soft computing is a general term covering a number of methodologies which have characteristics that make themselves unique in that area. They are tolerant of imprecision, uncertainty and partial truth and they do not suffer from the inflexibility of other standard algorithms. For these reasons they offer adaptability, allowing to track changing problem environments quite well. Examples of these techniques are represented by the Artificial Neural Networks (ANNs) and the Evolutionary Algorithms (EAs).

Artificial neural networks are well-defined computational models belonging to soft computing techniques. Different from traditional algorithms, ANNs have a key difference, based on the biological features of natural neural network, that define an ANN as a model that learns to perform a task rather than being directly programmed. In this sense many neural network solutions exist either because a solving program is very difficult to write, or because the neural network 'learnt solutions' provide improved performances. ANN also admit a degree of imprecision of a different kind at different levels; indeed, data used to train the neural networks can be noisy to some extent without affecting learning too much, allowing a network to work with variable data.

Artificial neural network solutions are growing more sophisticated and, in recent times, they find their way into important industrial applications fields, such as credit card detection, stock market forecasting, human health monitoring and diagnosis, electrical engine monitoring, and many others.

A large number of successful applications demonstrate that ANN design is improved by considering it in conjunction with evolutionary algorithms, since neural and evolutionary techniques can combine in a synergetic way. Different kinds of evolutionary techniques become very useful in global optimization problems, and several studies carried out in the literature demonstrate how these algorithms represent a more integrated way of optimizing classifier systems since they do not require any expert knowledge of the problem.

The study carried out in this thesis is developed in the area of soft computing, and it defines a robust and easy approach, that is able to solve complex problems, by using soft computing techniques. In particular, the thesis consists in the definition and the implementation of an evolutionary approach for the design and the optimization of neural complex models. A particular kind of artificial neural networks, Multi-Layer Perceptrons (MLP), are considered in this study.

1.2 Motivations

The attractiveness of ANNs comes from the remarkable information processing characteristics of the biological system such as nonlinearity, high parallelism, robustness, fault and failure tolerance, learning, ability to handle imprecise information, and their capability to generalize.

The success of an ANN application usually requires a high number of experiments. Moreover, several parameters of an ANN can affect, during the design, how easy a solution is to find. Some of these parameters are related to the architecture design of the neural network, concerning the number of layers and nodes, and the connection weights. Some others consider the selection of data that will define the training, the test and the validation set, in order to guarantee their availability and their integrity. Other important factors consider, then, the handling of local minima and the training of the ANN, trying to avoid the entrapment in local minima, and to avoid the overfitting of the network. Finally, a good deal of attention must be paid to the data set definition, so that the network will generalize correctly on data which has not been trained on. There is no standard design that is able to solve all these questions for a given problem.

A particular type of evolving systems, namely neuro-genetic systems, have become a very important topic of study in neural network design. They define so-called EANNs, that are biologically-inspired computational models that use evolutionary algorithms in conjunction with neural networks to solve problems.

A survey of the state of the art shows several approaches that are considered in order to apply evolutionary algorithms in neural network design. Some consider the setting of the weights in a fixed topology network, some others optimize network topologies, or the evolution of the learning rules, the input feature selection, or the transfer function optimization. Several approaches present an interesting conjunction of the evolution of network architecture and weights, carried out simultaneously.

This thesis presents an evolutionary approach based on the same last idea, which considers the simultaneous evolution of architecture and weights in an ANN design. Nevertheless this is a well-known evolutionary technique, its application in different kinds of problems represents an important aspect in actual researches. This thesis wants to study the most important issues of the joint conjunction of the evolutionary algorithms and artificial neural networks, more precisely the design of MLP neural networks, and would like to give an improvement to these evolutionary approaches. Several aspects can be identified in order to motivate the study conducted in this thesis. Particular attention is given to study in depth evolutionary operators based on the well-defined idea of Evolution Programs [77], that share the common principle by which a population of individuals undergoes some transformations, and during this evolution process the individuals strive for survival. An important aspect of evolution programs is that the individual structure represents, in a complete form, all the information necessary to carry out genetic operators. Evolution programs use then the current encoding, by involving the appropriate data structures, and define suitable genetic operators.

The use of evolution strategies in the approach implemented improves the evolutionary process of weight perturbations, since evolution strategies offer a simplified method for self-adapting information about connection weights of each neural network.

This thesis also wants to underline the importance of the use of fine local search algorithms, in this case by implementing training with backpropagation algorithm, in conjunction with global optimization algorithms, evolutionary algorithms, in order to obtain better results. Another effort is given in further study of the implementation of the crossover, a very hard issue, due to the detrimental effects on the population. Finally, this thesis is motivated by the will to define an evolutionary approach, able to reduce the working load of a human expert, and supporting it with a computational effort of distributed machines.

1.3 Objectives of the Thesis

An important objective of this thesis is to define a so called 'neuro-genetic' approach, based on evolutionary algorithms, in order to handle an optimized design of classifier systems, defined with artificial neural networks. The attention is focused on the important contribution that this solution brings in the neural networks. A reduction of the detailed knowledge of a complex optimization problem and of the computational effort by any human expert will contribute to give the optimal solution in ANN design.

In particular, this work presents an approach to the joint optimization of neural network structure and weights, which can take advantage of the backpropagation algorithm as a specialized decoder. Important aspects of the simultaneous evolution underline that an evolutionary algorithm allows all aspects of a neural network design to be taken into account at once, without requiring any expert knowledge of the problem. Furthermore, the conjunction of weights and architecture evolution overcomes the possible drawbacks of each single technique and combines their advantages. The most important aspect of evolving weights is to simulate the learning process of a neural network, avoiding the drawbacks of the traditional gradient descent techniques, such as BP.

The algorithm implemented can be considered as a hybrid algorithm, since a local search gradient descent technique, backpropagation, can be used as a local optimization operator on a given data set. The basic idea is to exploit the ability of the EA to find a solution close enough to the global optimum, along with the ability of the BP algorithm to finely tune a solution and reach the nearest local minimum.

1.4 Strategy and Methodology

This thesis contributes several key items to the field of evolutionary algorithms for neural network design and optimization. The evolutionary process is a more integrated and rational way of designing ANNs since it allows single aspects of the design to be taken into account as well as several interacting aspects at once and does not require any expert knowledge of the problem. Evolutionary algorithms are especially useful for complex optimization problems where the number of parameters is large and the analytical solutions are difficult to obtain, and they can help to find out the optimal solution globally over a domain. Evolutionary algorithms become, in this sense, helpful and they represent a suitable solution to solve the problem of the ANN design.

This kind of evolutionary learning for ANNs has also been introduced to reduce and, if possible, to avoid the problems of traditional gradient descent techniques, such as backpropagation, that lies in the trapping in local minima. In this thesis the backpropagation is used in order to improve the global search optimization with fine local tuning. An empirical methodology is followed in the definition of the evolutionary approach implemented. The methodology is usual in the field of the evolutionary computation, defining genetic operators as selection, mutation and recombination, that are applied to the individuals of a population, in an evolutionary cycle. A close behavioral link between parent and offspring is however maintained by applying techniques like weight mutation and partial training, in order to reduce behavioral disruption. The methodology then includes a validation process of the approach on toy and real-world problems.

1.5 Synopsis

A brief overview of the so-called ‘neuro-genetic’ approach, object of this thesis, is described as follows. A new population can be created either by loading a previously saved population, or by generating a new one. All individuals have not pre-established topology, the population is initialized with different hidden layer sizes and different number of neurons for each individual, in order to maintain diversity between all the individuals in the new population. Two exponential distributions are used to determine the number of hidden layers and the number of neurons for each layer in each individual, while a normal distribution is used to initialize all the weights and biases values of the new network. Variance matrices are also defined for all weights and bias matrices, and will be applied in conjunction with evolutionary strategies in order to perturb network weights and bias.

The evolutionary cycle is applied to the population, with selection, mutation and recombination, until termination conditions are not satisfied. A peculiar aspect of this approach is that BP is not used as some genetic operator, instead, the EA optimizes both topology and weights of the networks; BP is optionally used to decode a genotype into a phenotype NN. Accordingly, the genotype undergoes the genetic operators and reproduces itself, whereas the phenotype is used only for calculating the genotype fitness.

Two kinds of fitness function are defined in this approach and, for each application considered, only one has been chosen and applied to each individual of the population in the evolutionary process. Both depend on the cost of each individual. For this reason, the convention that the best fitness corresponds to the lowest fitness is adopted, defining the objective of each task as a cost minimization problem. The fitness function implemented in this approach also works as a controller and selector, because it penalizes large networks.

The first is a Mean Square Error (MSE) fitness function, that depends both on the accuracy, that is its mean square error, and on the cost of each individual. The second is a Correlation Coefficient fitness, proportional to the statistical correlation coefficient and to the cost of each individual.

There is no a standard definition in the literature on the way testing and validation are named; in this neuro-genetic approach the test set is used to avoid the overfitting problem, while validation set is used to validate the approach. In the evolutionary process, two fitness values are calculated for each individual: the fitness f, used by the selection operator, and a test fitness $\hat{f}$. The fitness $\hat{f}$ is calculated according to the equation of the objective function considered, by using the MSE over the test set, or the correlation coefficient. When BP is used, i.e., if $bp = 1$, then $f = \hat{f}$, that corresponds to consider only fitness calculated on test set, since the network has learned the train set in the learning process. In the opposite case, when ($bp = 0$), the fitness f is calculated according to its equation, by using the MSE (or correlation coefficient) over the training and test sets together.

The selection operator considered in this approach is the truncation selection. Elitism is also considered, allowing the survival of the best individual unchanged into the next generation and the solutions to get better over time. All the individual in a population are initially ranked according to ascending order of their fitness values, since the neuro-genetic approach considers a minimization problem. Each solution is assigned to an element of a ranked vector. Starting from a population of n individuals, the worse $\lfloor n/2 \rfloor$ (with respect to f) are eliminated. The remaining individuals are duplicated in order to replace those eliminated. Finally, the population is randomly permuted. The population created as result of selection operator will become the population of the parents for the new population that have to be created for the next generation.

In each new generation, a new population has to be created. The first operator implemented is selection. The first half of the new population corresponds to the best parents that has been selected with the truncation operator, while the second part of the new population is defined by creating offspring from the previously selected parents.

The evolutionary process attempts to mutate weights before performing any structural mutation; however, all different kinds of mutation are applied before the training process. Weight mutation is carried out before topology mutation, in order to perturb the connection weights of the neurons in a neural network. Weight perturbation is carried out by applying evolution strategies. After each weight mutation, a weight control is carried out, in order to delete neurons whose contribution is negligible with respect to the overall network output. This allows to obtain, if possible, a reduction of the computational cost of the entire network before any architecture mutation.

Topology mutation is then implemented with four types of mutation, considering respectively neurons and layer addition and elimination. The elimination of a neuron is carried out only if the contribution of that neuron is negligible respect to the overall network output, while the addition and the elimination of a layer and the insertion of a neuron are applied with independent probabilities.

Two types of recombination operator are studied in this approach: the first is a kind of single point crossover, in which two cutting points are chosen for each of the two selected parents, and then are used to cut the two individuals. The offspring is created by swapping the parts of the parents. The second is defined as a vertical crossover, in which the neural structure of the new individual is created by adding the number of neurons in any hidden layer of each parent, excepted for input and output layer (they are the same for each neural network). Crossover is again a critical issue in evolutionary algorithms, as reported from successful approaches presented in the literature. For this reason future work of this thesis will develop further studies of this operator.

At the end of each evolutionary generation the best individual, corresponding to the best neural network, found in that iteration is saved.

This approach has been successfully validated on a linear regression model and on two benchmark problems, and then applied to three different real-world problems, regarding respectively an electrical engine fault diagnosis problem, a brain wave signal processing classification problem, and a financial application.

1.6 Organization

The remainder of the dissertation is arranged as follows. Chapter 2 provides brief but necessary overviews of evolutionary computation, followed by a brief discussion about the basically aspects of evolutionary techniques and the description of the principal kinds of evolutionary algorithms presented in the soft computing area.

Chapter 3 presents the main aspects of the artificial neural networks, defining the mainly different types of network models. Particular attention is given to the definition of the main critical issues that have to be considered in a neural network design. Chapter 4 provides a survey of the state of the art in the evolutionary artificial neural network design. In this chapter evolutionary techniques, described in the recent literature, have been presented for each typology of neural optimization considered.

Chapter 5 presents the neuro-genetic approach implemented in this P.h.D. study in order to define a novel evolutionary approach for neural network design and optimization, taking advantage of the joint optimization of the simultaneous evolution of structure and weights of neural networks.

Chapter 6 validates the neuro-genetic approach implemented, by comparing it first with a linear regression model, and then with two benchmark problems, that are respectively the Pima Indian Diabetes problem and the Breast Cancer Wisconsin problem.

Chapter 7 and the two following chapters, discusses three real-world application, subject of this P.h.D. dissertation, that are defined as classification problems. This chapter considers an incipient fault diagnosis in an electrical drives application.

Chapter 8 describes an application to brain wave signal processing, in particular as a classification algorithm in the analysis of P300 Evoked Potential.

Chapter 9, finally, presents two different financial problems. The first explains the construction of factor models of financial instruments, and a sample statistical arbitrage is also presented in this financial modeling, providing satisfactory results and significant profits. In the second financial application considered, the possibility of forecasting a financial time-series is tested by using the neuro-genetic approach. In particular, the approach uses different financial instruments to forecast the next-day closing price of the Dow Jones Industrial Average (DJIA).

The conclusion discusses in more detail how this study contributes to the evolutionary approaches for neural network optimization. In addition, attempting to make certain that this research follows a clear methodological framework, establishing a working model for how such analysis can be conducted in the future.

Chapter 2

Evolutionary Algorithms

2.1 An Introduction to Evolutionary Computation

Evolutionary computation (EC) defines the quite young field of the study of computational systems based on the idea of natural evolution and adaptation [62, 37, 117, 7, 8, 135]. The term evolutionary computation is recent and it represents the effort to bring together researches that are based on the aspects that form the essence of the evolution. Neo-Darwinian paradigm represents a widely accepted collection of evolutionary theories, that define the evolutionary processes of reproduction, mutation, and selection as the main physical processes operating within individuals in a population [56]. In this sense evolutionary computation becomes the inescapable result of interacting these basic physical statistical processes.

Recent advances in this research area [136, 135] formalized the aim of the evolutionary computation as follow:

> ‘the primary aims of evolutionary computation are to understand the mechanism of such computational systems and to design highly robust, flexible, and efficient algorithms for solving real-world problems that are very difficult for conventional computing methods.’

Compared to conventional methods, the major advantages, defined in the literature [42], for the evolutionary computation approaches, regard the conceptual and computational simplicity, well applicability to broad classes of problems, suitable real-world problem solvers, potential to use domain knowledge and to hybridize with other methods, capability of self-optimization, etc.

The advantages of evolutionary computation approaches make them very suitable for problems with dynamically changing environment and multi-objective optimization requirements. Traditional optimization applications require finding a parameter values for the considered problem, such that a certain quality criterion, called *objective function*, is maximized (or, equivalently, minimized), and they are not designed for processing inaccurate, noisy and complex data. Noisy and time-varying objective function values require robust global optimization methods when these traditional approaches fail.

A classical simplex method requires a problem to be formulated in exact and accurate mathematical forms; however it does not work well with problems in which the objective function cannot be mathematically represented with linear function or when non-differentiability has to be considered.

The history of evolutionary computation goes back to the 1960s with the introduction of ideas and techniques such as genetic algorithms (GA), evolution strategies (ES) and evolutionary programming (EP) [40].

Fogel [43] defined that evolution provides inspiration for computing the solutions to appeared difficult problem by considering the key foundation for the efforts spent in evolution strategies (ES). Introduced by Rechenberg [97], ES are algorithms that imitate the principles of natural evolution for parameter optimization problems. Temporal problems consider then, as main development requirement, the changing of behavioral strategies, depending on their evolution. In this sense Holland proposed with the definition of genetic algorithms (GA), methods able to recombine the successful pieces of competing strategies, taking the knowledge by independent individuals. Fogel also described the intelligence of a machine as the capability of a system to adapt its behavior to meet desired goals in a range of environments. He also presented an alternative approach to generating machine intelligence that simulated the evolution on a class of predictive algorithms. This was the foundation for the evolutionary programming (EP) research [44]. A more recent development is so-called genetic programming (GP), proposed by Koza [66]; it is considered as a special sub-branch of GA, useful to search for the fittest computer program which solve a particular task. Further recent developments present other different types of evolutionary techniques, as hybrid algorithms [35, 77], that combine global search using evolutionary algorithms and local search using individual learning algorithms, and memetic algorithms [82, 125], also for describing genetic algorithms that use local search heavily. Other studies define multiobjective evolutionary algorithms, like that implemented by Merelo and colleagues [24], able to optimize both error types in classification, as false positive and false negative errors, and to optimize the design of the structure of the MLP neural network considered.

Whilst all these evolutionary techniques differ slightly in their actual implementations, they use the same metaphor of mapping problem solving onto a simple model of evolution. In the rest of the chapter the basic idea of such techniques are described.

2.2 Evolutionary Algorithms

In recent years, the general term of *evolutionary algorithms* has been used to define algorithms implementing evolutionary computation. Evolutionary algorithms (EAs) [117] are algorithms based on models that consider ‘artificial’ or ‘simulated’ genetic evolution of individuals in a defined environment. In evolutionary computation different evolutionary techniques are defined, and when at least one of these are in place, whether in nature or in a computer, evolution is the inevitable outcome [6]. They are a broad class of stochastic optimization algorithms, inspired by biology and in particular by those biological processes that allow populations of organisms to adapt to their surrounding environment: genetic inheritance and survival of the fittest. Different types of evolutionary algorithms are defined in the literature.

Evolutionary algorithms are especially useful for complex optimization problems where the number of parameters is large and the analytical solutions are difficult to obtain. EAs can help to find out the optimal solution globally over a domain. As defined in [7], optimization does not imply perfection, yet evolution can discover highly precise functional solutions to particular problems posed by an organism environment, and even though the mechanisms that are evolved are often overly elaborate from an engineering perspective, function is the sole quality that is exposed to natural selection, and functionality is what is optimized by iterative selection and mutation. In this sense, evolutionary algorithms can be described as useful methods to solve difficult engineering optimization problems.

At the algorithmic level, they differ mainly in their representations of potential solutions and their operators, used to modify the solutions, even though, from a computational point of view, representation and search are their main key issues.

Evolutionary algorithm have been applied in different areas, that can all overlap to some extent, and many applications could rightly appear in more than one of the categories. Some of them regard applications of evolutionary computation approaches in planning, including one of the best known combinatorial optimization problems like the traveling salesman problem [41], or the transportation problem [77]. Other applications consider design problems. In this area EC techniques have been widely to artificial neural networks, both in the design of network topologies and in the search for optimum sets of weights. This is also the subject of this thesis. A significant amount of EC research has concerned the implementation of classifier systems in many applications, considering different system implementations, as the case of a fuzzy hybrid system used for financial decision making [86]. Other examples of real-world classification cases are described in this P.h.D. dissertation.

2.2.1 Principles of Evolutionary Algorithms

All evolutionary algorithms rely on three basic properties which characterize the prototype of a general evolutionary algorithm and which distinguish it from other search algorithms.

- Evolutionary algorithms are all population-based, and they use the collective learning process of a population of individuals. The strength of each evoluationary algorithm is essentially due to their updating of a whole population of possible solutions at each iteration of evolving algorithm; this is equivalent to carry out parallel explorations of the overall search space in a problem. The initial population may be either a random sample of the solution space or may be seeded with solutions found by simple local search procedures, if these are available. Given enough time, the resulting process tends to find globally optimal solutions to the problem much in the same way as in nature populations of organisms tend to adapt to their surrounding environment.

- A population is evolved by using stochastic operators, such as mutation, recombination and selection. Mutation corresponds to an erroneous self-replication of individuals; recombination allows parent to pass on some of their characteristics to offspring, and selection process favors better individuals to reproduce more often than those that are relatively worse.

- A measure of the quality of the individuals on their environment can be assigned to each individual. A comparison of individual fitness is possible and the selection process is based on that quality measurements to carry out the selection.

All properties are general, and different representations of individuals and schemes for implementing fitness evaluation, selection and search operators are adopted in different evolutionary algorithms. These basic differences in the utilization of these principles characterize the mainstream instances of the several evolutionary computation techniques previously defined.

The evolutionary pseudocode of the general framework implemented by each algorithm, is as shown in Figure 2.1.

```
*generation* = 0
Create the initial Population
**while not** termination condition **do**
  *generation* = *generation* + 1
  Calculate Fitness of the Population
  Select parents from Population
  Recombine the Population (with $p_{cross}$)
  Mutate the Population (with $p_{mut}$)
**end while**
```

Figure 2.1: Pseudocode of a general evolutionary algorithm.

The main loop of this framework is iterated for a number of generations until the maximum allowed computing time is reached or a sufficiently well performing solution is found. Mutation and recombination depend on algorithm-specific probabilities. Further parameters can be defined in each evolutionary algorithm for the operators and the representation of the individuals.

Texts of reference and synthesis in the field of evolutionary algorithms are [77, 7], and other studies on global optimization by evolutionary algorithms are reported in [132]. Recent discussion about evolutionary computation are presented in [62, 37, 12]. A gentle introduction to evolutionary computation is given in [134, 10] and other recent advances in evolutionary computation are described in [136, 135], in which these approaches attract increasing interest from both academic and industrial society.

2.2.2 Evolutionary Approach

An evolutionary algorithm maintains a population of candidate solutions for the problem at hand, and makes it evolve by iteratively applying a (usually quite small) set of stochastic operators, such as *selection*, *mutation*, and *recombination*. Some of these operators are very useful in solving a particular class of optimization problems. Here some commonly used stochastic operators are described. In the following section, each example reported in order to better explain how the operators work, consider only binary strings and it refers to the canonical style defined by Goldberg [46], in genetic algorithm (GA) approaches. Other operators will be presented in the following sections, although they do not represent the whole set of all possible search operators.

Selection

Selection is one of the main operators in evolutionary algorithms whose primary aim is to choose the best solutions in a population, and, therefore, to concentrate the use of the available computational resources in promising regions of the search space. This operator does not create any new solution, but selects relatively suitable solutions from a population while deleting the remaining, not so good solutions. Exploration and exploitation are correlated aspects of the search and, the higher the pressure exerted by selection toward a concentration of the computational effort, the smaller the fraction of the exploration resources. On the contrary, if selective pressure decreases, the exploration resources increase and an evolutionary algorithm will tend to randomly sample the space of feasible solutions. At the other extreme, if the selective pressure increases, the evolutionary algorithm will degenerate into a local gradient search method. Further discussion will be carried out in this P.h.D. dissertation in Chapter 5.

In a population, the identification of satisfactory or bad solutions is usually accomplished by their fitness values, the idea being that a solution with good fitness will have a higher probability to be selected. Selection operators differ in the way individuals are chosen for the creation of the new individual generation. Evolutionary algorithms can be regarded as a trade-off between these two extremes, and selection is the instrument to adjust it. Some of the most common selection methods used are briefly described.

Fitness Proportionate Selection In this method the selection probability p_i of each individual is directly proportional to all other individuals in the population, with the following correlation function:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{2.1}$$

This method presents some drawbacks: the first is represented by the so-called *superindividuals*. A superindividual is an individual x_k whose fitness is bigger than all other individuals: $f(x_k) >> f(x_i), \forall i \neq k$, being $f(x_k) << f_{Optimal}$, where $f_{Optimal}$ corresponds to the global optimum to find in the search space. A superindividual is selected with a very high probability by fitness proportionate selection and, in a few generations, it ends up overwhelming any other genotypes initially in the population, causing premature convergence. Another difficulty occurs when individuals in a population have very similar fitness values, since their selection probability will be very similar, and it will be difficult for the algorithm to identify the best individual with this selection scheme. These drawbacks could be solved by appropriately modifying the fitness function or, more simply, resorting to alternative selection schemes.

Rank Based Selection This scheme differs from fitness within the population selection by computing selection probabilities according to the rank of individuals, sorted by decreasing fitness, rather than to their fitness values. Different kinds of this scheme are defined. In linear ranking selection, the selection probability for the i^{th} individual in a population of n individuals is defined as:

$$p(i) = \frac{1}{n}[\beta - 2(\beta - 1)\frac{i-1}{n-1}] \tag{2.2}$$

where $0 <= \beta <= 2$ is a parameter that can be viewed as the expected sampling rate of the best individual across n independent extractions with re-insertion.

Truncation Selection Truncation selection was originally used by Schwefel [107] and Rechenberg[97], and, later, by the breeder genetic algorithm approach [84]. Truncation is the selection scheme used in the neuro-genetic approach implemented in this work, and will be described in Chapter 5.

Breeders measure selection through the selective differential S, defined as the difference between the average fitness of the individuals selected for reproduction $f(\bar{x_t})$ and the average fitness of the entire population $f(x_t)$:

$$S_t = f(\bar{x_t}) - f(x_t), \bar{x_t} \subseteq x_t \tag{2.3}$$

Truncation selection consists in selecting for reproduction just the best individuals and discarding the rest of the population.

Tournament Selection In rank-based and fitness proportional selections communication overhead can increase since these selection methods are based on global information about the whole population. This could be a problem in implementing evolutionary algorithms in a parallel machine. Tournament Selection extracts only k individuals from the population with uniform probability and makes them play a tournament, which is won, in the deterministic case, by the fittest individual among the participants.

The tournament may be probabilistic as well, in which case the probability for an individual to win is generally proportional to its fitness. If k is equal to the population size, this scheme degenerates into the truncation selection scheme or fitness proportionate selection in the case of probabilistic tournament selection.

Elitist Selection The cycle of birth and death of individuals is hard-linked to the management of population, and a lifetime is associated to each individual. During evolution, the expected lifetime of an individual is typically one generation, but in some EA systems it can be longer. Elitism links the lifetimes of individuals to their fitnesses, keeping satisfactory solutions in the population for more than one generation. In particular, the individuals with better fitness will have longer lifetime in the evolution. Elitism always copies the best individual into the next generation without any modification, and more than one individual may be copied in creating the new population.

Elitism is usually implemented in addition to other selection schemes and, in the neuro-genetic approach object of this thesis, is considered along with truncation selection.

Mutation

Mutation randomly perturbs a candidate solution in an evolutionary cycle. As previously indicated, the purpose of mutation is to simulate the effect of transcription errors that can occur with a very low probability p_{mut}, the mutation rate, when a chromosome is duplicated.

In some evolutionary algorithms, like genetic algorithms, mutation is generally considered as a background operator, and its main function is to introduce new genetic materials and maintain some diversity in the population, since in this algorithm recombination does not introduce any new genetic components. In some other evolutionary approaches, like evolutionary programming, this operator is the most important, since in these methods no crossover is carried out. Some examples of different kinds of mutation implemented in such evolutionary algorithms are described in the following.

Bit-Swapping Mutation Generally, in this kind of mutation, each character in a string is replaced by another randomly chosen character, different from the one to be replaced, with a defined mutation probability. In a bit-string representation, the mutation flips a bit randomly from 0 to 1 or vice-versa, with low probability. An example of this kind of mutation is shown in Figure 2.2, a bit-swapping is carried out with a mutation probability p_{mut}.

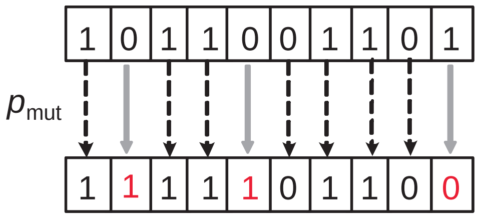


Figure 2.2: Example of mutation on a binary string.

Gaussian Mutation It is usually implemented in evolution strategies. This mutation is based on number, randomly brought from a Gaussian distribution with mean 0 and standard deviation σ. Then, the number will be added to the parent in order to create the offspring. The new individual is defined by the following expression:

$$x_i^{'} = x_i + N_i(0, \sigma_i) \tag{2.4}$$

$N_i(0, \sigma_i)$ is a random number calculated with a Normal distribution with mean 0 and standard deviation σ_i.

Strategy Parameters Mutation This kind of mutation represents one of the first methods that have been proposed for self-adapting the mutation concurrently with the evolutionary search. The most common implementations in use currently, derive from the work of Schwefel [106] and Fogel and colleagues [44]. In each case, the vector of objective variables x is accompanied by a vector strategy parameters σ, where σ_i represents the standard deviation be used when applying a zero-mean Gaussian mutation to a particular component in the considered parent. Schwefel and Fogel carried out an update of mutation strategy parameters. Schwefel defined the following update procedure:

$$x_i^{'} = x_i + N_i(0, \sigma_i^{'}) \tag{2.5}$$

$$\sigma_i^{'} = \sigma_i e^{\tau_0 N(0,1) + \tau N_i(0,1)} \tag{2.6}$$

where the constants are defined by the equations:

$$\tau_0 = \frac{1}{\sqrt{2n}} \tag{2.7}$$

$$\tau = \frac{1}{\sqrt{2\sqrt{n}}} \tag{2.8}$$

$N(0, 1)$ is a standard Gaussian random variable sampled once for all n dimensions and $N_i(0, 1)$ is a standard Gaussian random variable re-sampled for each of the n dimensions. The author claims that his procedure offers a general control for all dimensions and such control also offers a simplified method for self-adapting a single parameter σ. The values of $\sigma^{'}$ are defined as log-normal perturbations of their parent parameter vector σ.

Fogel et al implemented an independent method, in which:

$$x_i^{'} = x_i + N_i(0, \sigma_i) \tag{2.9}$$
$$\sigma_i^{'} = \sigma_i + \chi N(0, \sigma_i) \tag{2.10}$$

The parameters of the parent are used in the strategy to define offspring before being mutated themselves, and the mutation of the strategy parameters is achieved using a Gaussian distribution scaled by χ and the standard deviation for each dimension. This procedure also requires incorporating a rule such that if any component $\sigma_i^{'}$ becomes negative is reset to an arbitrary small value ϵ.

Several comparisons have been carried out between these two methods and both of them have also been extended to include possible correlation across the dimensions, defining a multivariate Gaussian mutation with arbitrary covariance, rather than using the traditional independent random perturbations. Several studies confirmed the usefulness of these approaches and that the work implemented by Schwefel [106] generated generally a statistically significant optimization across a series of standard test functions.

Recombination

The aim of recombination is to decompose distinct solutions and then exchange their parts to form novel solutions, representing the offspring. The inheritance of information from two or more parents is then carried out by offspring.

Different kinds of crossover are considered in different evolutionary algorithms, and some of the most important are presented. In particular, discrete and intermediate recombinations are used with real-valued individual representation, and they are mostly used in evolution strategies. On the other hand, uniform and k-point crossover are used for binary representation of individuals, mainly implemented in genetic algorithms. Generally, crossover may not be applied to every selected pair of parents, but is controlled by the crossover rate, that represents the probability of applying crossover.

From the results obtained in several application, is obvious that not only for evolution strategies but also for canonical genetic algorithms, mutation is an important search operator that cannot be neglected in their implementation. Moreover, it is also possible to release the user of a genetic algorithm from the problem of finding an appropriate mutation rate control or fine-tuning a fixed value by transferring the strategy parameter self-adaptation principle from the evolution strategies to genetic algorithms.

Discrete Recombination In this method, the components of an offspring real-value vector come from two or more parent vectors. In other words, given two parents, each component of the offspring vector will be created taking the corresponding component of the first parent with a pre-defined recombination probability, otherwise taking the corresponding component of the other parent. Another offspring will be set as the complement of the other one.

Intermediate Recombination This recombination differs from discrete recombination, because each component of the offspring is defined as a linear combination (average) of its parent's corresponding components, according to a linear function like the following:

$$x_i^{'} = x_i + \alpha(y_i - x_i) \tag{2.11}$$

where α is a weighting parameter defined in the $(0, 1)$ interval. The second offspring y' will be similarly defined.

Uniform Crossover This crossover is usually applicable to strings of any alphabet. With this operator, each element of an offspring is created by taking the corresponding element in one of the two parents with equal probability. The parent from which the bit or character is to be taken is uniformly chosen at random. The two offspring will be complementary. Usually a mask can be used to implement crossover. An example is shown in Figure 2.3.

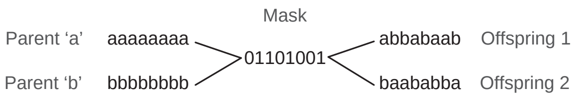


Figure 2.3: Example of Uniform Crossover

In the mask depicted in this figure, a bit equal to 0 means that the corresponding bit of the first offspring will be copied from parent 'a', while value 1 means that it will be copied from parent 'b'. The second offspring will be complementary of the first one.

K-Point Crossover This crossover is implemented by cutting the strings of the two parents at k randomly chosen positions, that are uniformly generated without repetition. An offspring is created by taking segments of the parent strings alternatively. Figure 2.4 shows an example of 2-point crossover on a bit string.

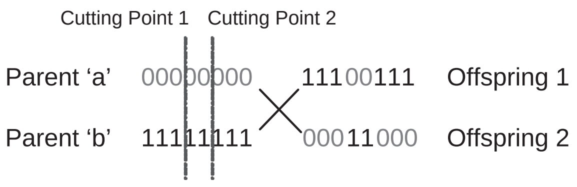


Figure 2.4: Example of 2-Point Binary Crossover

2.3 Genetic Algorithms

Genetic algorithms (GAs) are a class of evolutionary algorithms identified by a representation independent of the problem, usually fixed-length binary strings. Genetic algorithms stress genetic encoding of potential solutions into chromosomes and require a modification of an original problem into another suitable form. This would include mapping between potential solutions and binary representation, taking care of decoders or adjusting the algorithms. The representation encodes an individual and gives rise to a dual representation scheme. A crucial issue in applying genetic algorithms to a problem is how to find a representation which can be searched efficiently. Figure 2.5 shows the pseudocode of a canonical genetic algorithm.

The ability to create better solutions in a genetic algorithm relies mainly on the genetic recombination operator. The standard algorithm performs a so-called *single-point*

```
*generation* = 0
Generate the initial population
**while not** termination condition **do**
    Evaluate the fitness of each individual of population
    Select parents from Population
    Apply crossover operator to the selected parents
    Apply mutation operator to new individuals
    *generation* = *generation* + 1
    Define new population by replacing parents with the offspring
**end while**
```

Figure 2.5: Pseudocode of a canonical genetic algorithm.

crossover, where two individuals are randomly chosen from the population, and their phenotypes are divided in two separated parts by a cutting point. A new individual is created by swapping the two parts of the parents and concatenating them. The standard mutation operator considered in this algorithm is carried out by inverting bits with a slow associated probability. The benefits of the genetic operators come from the ability of forming connected substrings of the representation that correspond to problem solutions.

The analysis of genetic algorithms has led to an important result, the *Schema theory*, which tries to analyze GAs in terms of their expected schema sampling behavior. A schema denotes a particular kind of similarity template, and the schema theorem claims that the canonical genetic algorithm provides a near-optimal sampling strategy for schemata. A detailed description of this theory is presented in the literature [117, 10].

The benefits of the genetic operators come from their ability of create connected substrings of the representation that correspond to problem solutions. The recombination operator is not effective in environments where the fitness of an individual of the population is not correlated with the expected quality of its representational components. The corresponding environment is called deceptive [45].

In genetic algorithm implementation, the issue of how to encode possible solutions to the problem into chromosomes can become a very hard question, since a poor representation could make a problem difficult to solve. Furthermore, during the fitness evaluation phase, each chromosome normally has to be decoded back to the original problem domain in order to compute fitness, then encoding and decoding operators have to be defined.

2.3.1 Genetic Programming

J. Koza introduced a new evolutionary approach, [66], extending the genetic model of learning to the space of programs, since he suggested that the desired program should evolve during the evolutionary process. GP is considered as a special sub-branch of genetic algorithms, where a population of executable computer programs is created and evolved with stochastic operators.

For each problem considered, some previous parameter values have to be carried out, like the selection of terminal set T and of a function set F for the language of the program, the identification of the evaluation function, the selection of the parameters of the evolutionary algorithm and the selection of the termination condition. A simple example of function and terminal sets are represented like:

$$F = +, -, *, / \tag{2.12}$$

$$T = A, B, C, D \tag{2.13}$$

The structure which undergoes evolution is a hierarchically structured computer program. The search space is a hyperspace of possible solutions, valid programs, which can be viewed as a space of rooted trees, each composed by the functions and terminals appropriate to the particular problem domain. Usually, the population is composed of such trees; however, GP representation is not restricted to trees and other program representations have been proposed such as linear and graph based representation [13].

During evolution, the evaluation function assigns a fitness value to each tree (program which measures its performance. The selection is fitness proportionate, each tree has a probability of being selected to the next generation proportional to its fitness. The primary operator is crossover, a kind of single-point crossover as the one implemented in canonical genetic algorithms. Crossover starts by selecting a random cutting point in each parent tree and then exchanging the sub-trees, giving rise to two offspring trees, as shown in Figure 2.6. Mutation is implemented by randomly replacing a subtree with another, randomly generated one.

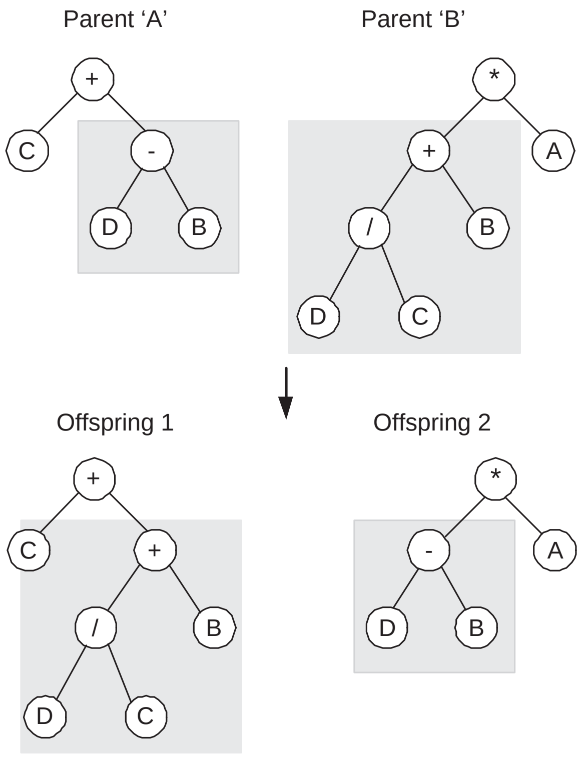


Figure 2.6: Example of Genetic Programming Crossover

Two important issues in genetic programming are the choice of the appropriate language for a given problem, and the size for the GP trees. The depth of the trees can in principle increase without limits under the influence of crossover, defining a so-called 'bloating' effect. Parameters are introduced in GP in order to control the tree growth. Another common approach introduces a size penalty term into the fitness function. Also in genetic programming a divide-and-conquer strategy can be used to decompose the problem into sub-problems. In this case it has been observed that during evolution some subtrees appear repeatedly as parts of successful individuals. In order to improve the performance, Koza encapsulated these subtrees into modules, which are seen as single units in the evolutionary process. He defined the so-called Automatically Defined Functions (ADF) [67], that are methods for automatically identifying and using these modules within GP.

2.4 Evolutionary Programming

Evolutionary Programming (EP) was first proposed by Fogel [44]. EP aimed at evolution of artificial intelligence by evolving finite state machine, selected as chromosomal representation of individuals, emphasizing behavioral evolution, based on interpretation of output symbols, rather than genetic evolution.

This approach maintains a population of finite state machines, in which each individual represents a potential solution, a particular behavior, of the problem. When used for numerical optimization, evolutionary programming is very similar to evolution strategies in terms of algorithm. Indeed, at first step offspring is created and later individuals are selected for the next generation. EP does not use any recombination or crossover operator, and offspring are created by random mutations of the parent population. There are five possible mutation operators that can be implemented:

1. changing an output symbol,
2. changing a state in the transition function definition,
3. adding a new state,
4. removing a state,
5. changing the initial state.

There are some constraints on the maximum and minimum number of internal states; these mutation operators are chosen according to some probability distribution, that in canonical evolutionary programming is based on a Gaussian or a uniform distribution.

The selection is usually implemented by competition, a kind of tournament selection, in which a comparison between each individual and k others randomly selected is carried out, and the best n individuals are chosen. Selection can also be implemented according to two other techniques, like standard fitness proportionate selection, and truncation, in which the best n solutions are retained to become the parents for the next generation. An example of pseudocode of the classical evolutionary programming is shown in Figure 2.7.

2.5 Evolution Strategies

Evolution strategies (ES) are approaches that imitate natural evolution as a method to solve parameter optimization problems [97, 9, 106]. They were applied to problems with continuously changing parameters, and then they were extended also to discrete problems. They differ from genetic algorithms for several aspects, applying deterministic selection after

```
Generate the initial population with $n$ individuals randomly
generation=1
Evaluate fitness of each individual
while not termination condition do
  Create one offspring from each individual
  Evaluate the fitness of each offspring
  for each individual of the population with size $2n$ do
    conduct $m$ competitions
  end for
  Select best individuals from the population with selection operators
  generation=generation + 1
end while
```

Figure 2.7: Pseudocode of the classical evolutionary programming.

reproduction, Gaussian mutation and discrete or intermediate recombination. As EP, ESs simulate evolution at the phenotypic level, emphasizing behavioural evolution.

Two kinds of deterministic selection schemes are defined in the ES, (n, m) and $(n+m)$ strategies. In the first one, from a population of n individuals, $m > n$ offspring are produced and the best n of them are kept for the next generation. The n parents are always discarded to make room for the best offspring. In $(n + m)$ strategies, on the contrary, the best n individuals among the m offspring and the n parents survive into the next generation: this kind of strategy never discards the best solution so-far, as an elitist selection scheme, guaranteeing a monotonic improvement of the population. This strategy could have drawbacks when some problem features change during evolution, in particular when these features correspond to the strategy parameters, and when a small population size is considered. In that case efforts can be spent to obtain parameters setting and results in a satisfactory way, with low computational cost.

Mutation in the evolution strategies is often implemented by adding a Gaussian random number to a parent, with a Normal distribution with mean 0 and standard deviation σ_i. The expression of a simple mutation is defined as:

$$x_i^{'} = x_i + N_i(0, \sigma_i) \tag{2.14}$$

The setting of the standard deviation value is important in determining the performance of evolution strategies, and, furthermore, its optimal value is problem dependent. A solution was found by considering the standard deviation as a part of an individual so that it can be evolved automatically. This so-called *self-adaptation* in evolution strategies has been proposed by Schwefel [106] and Fogel[44]. Also this solution may have some drawbacks, since the independent mutation of different components of a vector could not be appropriate for problems where the components are not independent at all. To address this problem, the variance/co-variance matrix has been added as part of an individual. One of the major advantages of the $(n + m)$ strategies is the ease of using adaptive strategic parameter, like variance and co-variance, but a problem of these kind of strategies is that they often get stuck in local optima. Again, as indicated above, the (n, m) strategies are still preferred.

The most widely used recombination approaches in evolution strategies are discrete and intermediate recombinations. In discrete recombination the components of the two parents are mixed to create the first offspring, while the second is the complement of the first. In intermediate recombination the value of each component of the child individual is a linear combination of the corresponding component of all parents participating in the operation. It has been observed that the best results are obtained by applying discrete recombination to the object problem parameters and intermediate recombination to the strategy parameters. Furthermore, it has been proved that recombination of the latter is required for self-adaptation. The pseudocode of the (n, m) evolutionary strategy is represented in Figure 2.8.

```
Generate the initial population of $n$ individuals
*generation* = 1
Evaluate fitness value for each individual
**while not** termination condition **do**
   Create $m/n$ offspring on average from each individual, so that a total of $m$
   children are generated
   Evaluate fitness of each offspring
   Sort offspring into a non-descending order according to their fitness values
   Select the $n$ best children out of $m$ to be parents of the next generation
   *generation* = *generation* + 1
**end while**
```

Figure 2.8: Pseudocode of the (n, m) evolutionary strategy.

2.6 Evolution Programs

Evolution programs are a particular kind of genetic algorithms, introduced by Michalewic. A detailed description of his work is presented in [77]. Evolution Programs consider the idea of Davis [34], who wrote that:

'[...] I believe that genetic algorithms are the appropriate algorithms to use in a great real-world application. I also believe that one should incorporate real-world knowledge in one's algorithm by adding it to one's decoder or by expanding one's operator set'.

Michalewicz considers such modified genetic algorithms as evolution programs.

In classical genetic algorithm a modification of an original problem into an appropriate form, suitable for them, is required. This operation would include a mapping between potential solutions and a binary representation as a first step. Then, a decoding scheme will have to be considered, taking care of decoders and repair algorithms.

Evolution programs would leave the problem unchanged, modifying a chromosome representation of a potential solution, using a natural data structure, and applying appropriate genetic operators. In these algorithms a possible solution is directly mapped in an encoding scheme. They offer a major advantage over genetic algorithms when evolving ANNs since the representation scheme allows manipulating networks directly, avoiding the problems associated with a dual representation.

Although the idea implemented by Michalewicz is nowadays still actual and it is generally considered one important issue in evolutionary computation, and it is adopted in several recent works, and obviously presented in the literature, the terminology 'evolution programs' does not become standard yet and it is not commonly used. In this thesis the term 'evolution programs' refers to the idea of Michalewicz.

2.7 Coevolutionary Algorithms

These algorithms are a particular kind of evolutionary algorithms, that are based on continuous interactions between populations that are evolving. This is different from the customary evolutionary paradigm where a single population evolves under the selection pressure of a given fixed fitness function that plays the role of the environment. In nature, interactions between populations are omnipresent, and it is easy to think evolution as being a co-evolutionary process where changes in a certain species (population) influence the other ones, i.e., the environment is altered.

Co-evolutionary learning is a specialization of general evolutionary learning, that refers to two different forms of co-evolution. The first considers co-evolution at the population level, i.e., between two or more populations evolving at the same time. In this approach, the fitness of an individual in one population depends on the individuals in another population. The second form considers co-evolution at the individual level, evolving only one population. In this case the fitness of an individual depends on other individuals in the same population.

One advantage of these methods is that a global fitness function has not to be necessarily specified, only relative fitness is needed. This can be useful in complex problems, where an adequate global fitness function is difficult to define.

The methods based on co-evolution can roughly be classified as being either *cooperative* or *competitive*.

- *Cooperative co-evolution* [92] is a paradigm in the area of evolutionary computation based on the evolution of co-adapted subcomponents without the intervention of any agent external to the evolutionary process. In this type of co-evolution a number of species are defined and are evolved together. The cooperation among the individuals is based on how well they cooperate to solve a target problem. A recent P.h.D. dissertation concerning an analysis of cooperative coevolutionary algorithms has been presented by Wiegand [126].

- *Competitive co-evolution* is a kind of co-evolution in which no cooperation is defined between populations, but a competitive approach is used. One of the first successful competitive co-evolutionary approaches was implemented by Hillis [54].

2.8 Hybrid Algorithms

In the course of an evolutionary optimization, solutions are often generated with low phenotypic fitness even though the corresponding genotype may be close to an optimum. Without additional information about the local fitness landscape, such genetic 'near misses' would be overlooked under strong selection. In order to overcome this problem, near misses could be ranked by performing a local search and scoring them according to distance from the nearest optimum.

In this sense, there is some experimental evidence, in the works presented by Davis [35] and Michalewicz [77], that the enhancement of evolutionary methods by some additional, problem specific, heuristics, domain knowledge, or existing algorithms, can result in a system with outstanding performance. Such systems, called 'hybrid' systems, enjoy a significant popularity in evolutionary computation, and have been used successfully in many application areas. The near misses evaluation are the aim of hybrid algorithms, which combine global search using evolutionary algorithms and local search using individual learning algorithms.

Davis explained in [35] how he suggests to hybridize the genetic algorithm and local search algorithms by employing three principles:

- Use the current encoding: use the current algorithm encoding technique in the hybrid algorithm.
- Hybridize where possible: incorporate the positive features of the current algorithm in the hybrid algorithm.
- Adapt the genetic operators: create crossover and mutation operators for the new type of encoding by analogy with bit string crossover and mutation operators. Incorporate domain-based heuristics as operators as well.

The above three principles emerged as result of several researches, with the common aim to create the best algorithm for a particular problem. In this sense, various application-specific variations of evolutionary algorithms have been reported in the literature: some of them include variations of the structure elements, like dimension and encoding, some others carried out experiments with modified genetic operators.

Moscato and Norman [82] have introduced the particular term *memetic algorithm* to describe evolutionary algorithms in which local search plays a significant part. This term is motivated by the notion of a meme as a unit of information that reproduces itself as people exchange ideas. A key difference exists between genes and memes: before a meme is passed on, it is typically adapted by the person who transmits it as that person thinks, understands and processes the meme, whereas genes get passed on whole. Moscato and colleague liken this thinking to local refinement, and therefore promote the term memetic algorithm to describe genetic algorithms that use local search heavily.

Usually there exist several heuristic algorithms applicable to a given problem. Besides being incorporated for the purpose of initialization, some of these algorithms transform one solution into another by imposing a change in the solution encoding. One can incorporate such transformations into the operator set of the evolutionary system, which is usually a very useful addition.

As already defined in this chapter, there is a strong relationship between encodings of individuals in the population and operators, hence the operators of any evolutionary system must be chosen carefully in accordance with the selected representation of individuals. Davis, in [35], also said that crossover function and encoding techniques, used to handling the chromosomes of parents and offsprings, have to be combined, before applying the crossover operator. The situation is similar for mutation operators, that can be global or local, but also in this case they have to combine the encoding with their mutation functions, since a mutation operator is an operator that introduces variations into the chromosome of an individual.

Very often, hybridization techniques make use of local search operators, which are considered as ‘intelligent mutations’. Some of these incorporate gradient-based methods as ways to achieve a local improvement of individuals; an example is represented by the backpropagation algorithms; while some others incorporate hill-climbing methods. It is also not unusual to combine simulated annealing techniques with some evolutionary algorithms. The class of hybrid evolutionary algorithms described so far consists of systems which extend evolutionary paradigms by incorporating additional features, like local search or problem-specific representations and operators. Such hybrid algorithms can also be divided in two classes, that can exploit learning respectively either actively, via Lamarckian inheritance, or passively, via the Baldwin effect.

In the first case, the performance gains from individual learning are mapped back into the genotype used for the production of the next generation. This is analogous to Lamarckian inheritance in evolutionary theory, whereby characters acquired during a parent lifetime are passed on to their offspring. With this approach it is difficult to envision a process by which acquired information can be transferred into the gametes. Nevertheless, the practical utility of this algorithm has been demonstrated in some evolutionary optimization applications.

The Baldwin effect [11, 55, 3, 7] considers that individual learning influences the evolutionary process, and facilitates the assimilation of new genetic innovations. In this technique, learning guides evolution by assigning 'partial credit' for the genetic near misses, defined above. Individuals with useful genetic variations are thus maintained by learning, and the corresponding genes increase in frequency in the subsequent generation. As genetic components necessary for a complex structure accumulate in the gene pool, functions that previously required supplemental learning are replaced by genetically determined systems. Further discussion about Baldwin effect is presented in Chapter 5, where issues raised with computational models are discussed.

Chapter 3

Neural Networks

3.1 Introduction

Neural Networks are models inspired by the working of the brain, although they do not pretend to be accurate models of the central nervous system. Even if they are biologically inspired systems, they are best regarded as basically non linear statistical models [117].

There are several texts of reference and synthesis in the field of neural networks [19, 53, 21] and they can be considered as a combination of neurons and synaptic connections, which are capable of transmitting data through multiple layers. The end result is a system which is able to solve different problems like pattern recognition and classification. Artificial Neural Networks (ANNs) have their origin in the attempt to simulate by mathematical means the elementary processing units of the brain and their interconnections. Then, an important feature of these networks is their adaptive nature which allows to achieve learning by examples.

This feature makes ANNs very useful in problem solving. On the other hand, a neural network may be considered as an adaptive system that progressively self-organizes in order to approximate the solution, making the problem solver free from the need to accurately and unambiguously specify the steps towards the solution. Moreover, ANNs have the ability to progressively improve their performance on a given task by executing learning.

An example of highly simplified mathematical representation of a neuron encoding is shown in Fig. 3.1.

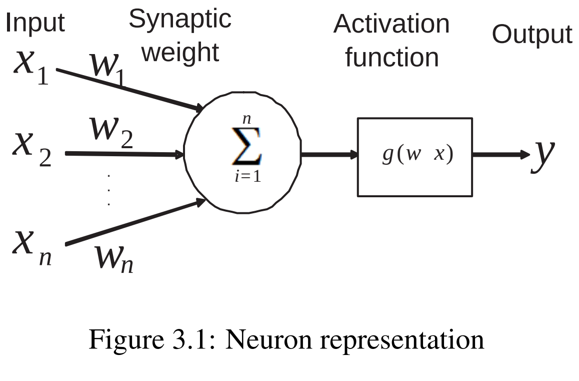


Figure 3.1: Neuron representation

where $g(\boldsymbol{w} \cdot \boldsymbol{x})$ is called activation function.

All these network units, or neurons, are simple processors whose computing ability is typically restricted to a rule for combining input signals and an activation rule that takes the combined input to calculate an output signal. Output signals may be sent to other units along weighted connections. The weights, associated to a connection, usually excite or inhibit the signal that is being communicated. The pattern of connectivity refers to the way in which the units are connected. In a type of network each unit may be connected to all other units; in another network model units may be arranged into an ordered hierarchy of layers where connections are only allowed between units in immediately adjacent layers. Again, other networks allow feedback connections between adjacent layers, or within a layer, or for units to send signals back to themselves.

A connection is specified by some parameters, like the unit it connects from, the unit it connects to, and a number, usually a real value, that denotes the weight value. A negative weight value will inhibit the activity of the connected-to unit, whilst a positive weight value will excite the connected-to unit. The absolute weight value specifies the strength of the connection. Usually, in feedforward and fully connected neural network types, the two parameters that represent respectively the connection source and destination are not defined. The pattern of connectivity is usually described with a matrix form, $\boldsymbol{W}$, where the entry w_{ij} represents the weight from unit j to unit i, or in some cases, viceversa.

3.2 Neural Network Types

The architecture of an ANN is determined by its topological structure, i.e., by the overall connectivity and transfer function of each node in the network.

There are different kinds of neural network architectures, some common examples being:

- **Multi-Layer Perceptron network (MLP)**: it usually consists of a feedforward fully connected network with an input layer of neurons, one or more hidden layers and an output layer. The output value is obtained through the sequence of activation functions defined in each hidden layer. Usually, in this kind of network, the supervised learning process is the backpropagation algorithm, which uses gradient descent search in the weight space to minimize the error between the target output and the actual value.

- **Single-layer Perceptron network**: this is a particular kind of MLP neural network, with only one hidden layer.

- **Radial Basis Function network (RBF)**: it consists of three layers, with an input, an hidden and an output layer. The hidden layer is used to cluster inputs of the network: for this reason, neurons in that layer are called cluster centres. This model uses a Gaussian kernel function to calculate the activations of the neurons in the first layer. The neurons in the output layer perform an ordinary linear weighted sum of these activations. Learning process is carried out in two stages. First an unsupervised learning process is carried out and a clustering algorithm is applied to cluster centres. Then supervised learning is applied to the weights of the output layer which associates the basis function outputs with specific classes. The performance of a RBF network depends on the way the inputs are clustered.

- **Hopfield network**: autoassociative network that acts like a memory and can recall a stored pattern even when input, with a noisy version of that pattern. This network is defined by a fully connected single layer network. After an input pattern is presented, the network will converge by means of a state update rule to a stable pattern. A Hopfield network has no input units since an input vector simply defines the initial activation of each unit. This activation value will be sent to other units, and at any point in time the state of the network is the vector of all unit states.

- **Self-organizing map**: has a set of input units that correspond in number to the dimension of the training vectors and output units that act as prototypes. The input units only serve to distribute the input vector to the network's output units, that shall be referred to as cluster units. Usually, the map has two layers of neurons and the signals from input neurons are fed to every neuron in the feature map, while learning the network generates a two dimensional representation of the input space. During training all units can be considered as competing to be awarded the training vectors. When a training vector is presented, the distance to all cluster units is calculated and the unit that is closest to the training vector is denoted as the winning unit. This will then adapt its weights in a way that moves that cluster unit even closer to the training vector.

- **recurrent network**: this term refers to a particular kind of networks with recurrent connections. A backpropagation network need not be strictly feedforward, and can have recurrent connections so that a unit can feed activation back to itself, or to other units in the same or lower levels. In some cases, this type of neural network can have connections that feed back from the output to the input layer and some of the input layer units feed back to themselves. In each case, for every recurrent network there is a feedforward network with identical behavior. Generally, recurrent networks are used to process patterns that can have variable lengths, that will be treated as sequences, divided in parts, and presented to the network at a different time step. These methods demonstrate the ability to predict the next part of data in a sequence from the past history of data.

Other two learning methods also implemented for classification and regression consider:

- **Support Vector Machine (SVM)**: a set of related supervised learning methods used for classification and regression. Their common factor is the use of a technique known as the 'kernel trick' to apply linear classification techniques to non-linear classification problems. For classification, SVM operates by finding a hypersurface in the space of possible inputs. This hypersurface attempts to split the positive examples from the negative examples. The split is chosen to have the largest distance from the hypersurface to the nearest of the positive and negative examples.

- **Independent Component Analysis (ICA)**: is a statistical and computational technique for revealing hidden factors that underlie sets of random variables, measurements, or signals. ICA works on learning process that minimizes the dependency between the output components.

Among the different kind of neural networks, feedforward Multi-Layer Perceptron (MLP) neural networks receive great attention due to their relative simplicity and computational capabilities. In a feedforward neural network an input pattern is transformed into an output pattern through the processing performed by a series of layers of interconnected nodes, defining layers of neurons. The layers between inputs and outputs are defined ‘hidden layers’ and the neurons that belong to these layers are called ‘hidden nodes’, because they are not directly connected to the external system through the inputs and the outputs. Each node performs a transfer function, represented by the following equation:

$$y_i = g_i(\sum_{i=1}^{n} w_i x_i) \tag{3.1}$$

where $g(\boldsymbol{w} \cdot \boldsymbol{x})$ is always the activation function. Commonly used continuous transfer functions are linear, hyperbolic tangent and Gaussian, even if most researchers prefer to use the sigmoid function, whose analytic form is:

$$y(x) = \frac{1}{1 + e^{-(bx-c)}} \tag{3.2}$$

The sigmoid, as the tangent hyperbolic, is a poor activation function because it is differentiable and it saturates to the horizontal asymptotic axes $y = 0$ and $y = 1$. The transfer function implemented in the neuro-genetic approach, described in the following chapters, is a kind of sigmoid function, the *tan-sigmoid* function, that tends asymptotically to -1 and +1 at the extremes. Figure 3.2 shows the shape of this function. As indicated in [117], networks of neurons with real-valued inputs and sigmoid transfer function can be used to approximate mathematical functions, allowing the parameterization of the latter. This is very useful in situations where the exact expression of the function is unknown.

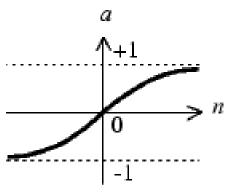


Figure 3.2: Tangent sigmoidal Function

An example of architecture of a feedforward MLP neural network is depicted in Fig. 3.3, where each activation function a_i depends on the contribution of the previous subnetwork topology.

The expressions of the activation values defined for the example reported in Figure 3.3 are here defined:

$$\boldsymbol{a_1} = f_1(\boldsymbol{IW_{1,1}} \cdot \boldsymbol{x} + \boldsymbol{b_1}) \tag{3.3}$$

$$\boldsymbol{a_2} = f_2(\boldsymbol{LW_{2,1}} \cdot \boldsymbol{a_1} + \boldsymbol{b_2}) \tag{3.4}$$

$$\boldsymbol{a_3} = f_3(\boldsymbol{LW_{3,2}} \cdot \boldsymbol{a_2} + \boldsymbol{b_3}) \tag{3.5}$$

The activation function obtained at each hidden layer will become the input for the successive layer, with the expression reported in the Equation 3.6.

$$\boldsymbol{a_3} = f_3(\boldsymbol{LW_{3,2}} f_2(\boldsymbol{LW_{2,1}} f_1(\boldsymbol{IW_{1,1}} \cdot \boldsymbol{x} + \boldsymbol{b_1}) + \boldsymbol{b_2}) + \boldsymbol{b_3}) \tag{3.6}$$

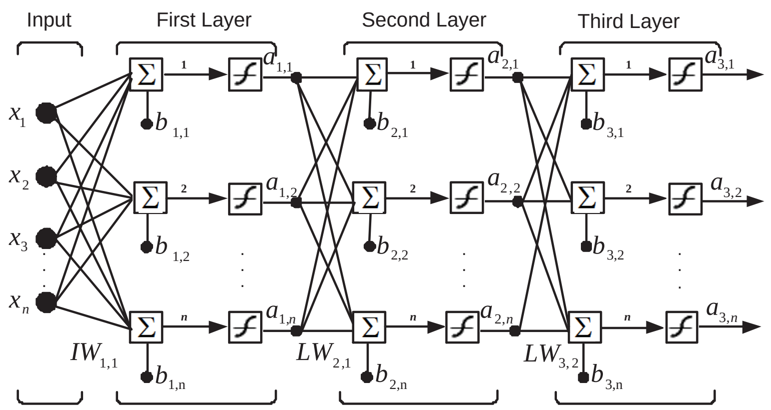


Figure 3.3: Feedforward MLP Structure

3.3 Learning in Neural Networks

The adaptive nature of neural network learning by examples is a very important feature of these computational paradigms, and explains the distinctive features of ANNs, which defines the training process as an algorithm that replaces the absence of a pre-defined set of instructions to follow.

The training process modifies the weights of the ANN, in order to improve a pre-defined performance criterion, that corresponds to an objective function, over time. This process is also called *learning rule* and it can take place in two ways:

- *supervised learning*, in which a net is defined with a dataset of input/output pairs, namely the *training set*. The learning process consists in updating the weights at each training step so that, for a given input, an error measure between the network's output and the desired known target value is reduced.

- *unsupervised learning*, in which an input/output relationship that has to be produced by the network is maintained, but no feedback is provided from the environment as to the correctness of the mapping. In this kind of learning the network must be able to discover by itself any categories of features possibly presented in the data. Networks that are able to infer pattern relationships without being supervised are so called *self-organizing*.

A third commonly used form of training makes use of the concept of *reinforcement learning*. In this case the value of training input/output pairs is not a measure of the difference between the desired and the obtained value, as in supervised learning, but rather, an evaluation of the result as 'wrong' or 'right' result.

3.3.1 Backpropagation Algorithm

The first and most popular method for performing supervised learning is the Backpropagation algorithm (BP) [103]. BP has been applied to a number of different learning tasks and has emerged as the standard algorithm for the training of multi-layer perceptron networks. As conjugate gradient, BP is based on gradient descent [79]. It generally uses a least-square optimality criterion, defining a method for calculating the gradient of the error with respect to the weights for a given input, by propagating error backwards through the network. Error backpropagation is essentially a search procedure that attempts to minimize a whole network error function such as the sum E of the squared error of the network output over an ensemble of training input/output pairs:

$$E = \frac{1}{2}\sum_{j=1}^{m}(t_j - o_j)^2 \tag{3.7}$$

where t_j is the target and o_j is the actual jth output of the network.

The backpropagation algorithm defines two sweeps of the network: first a forward sweep from the input layer to the output layer, and then a backward sweep from the output to the input layer. In the first step input vectors are propagated through the network to produce outputs of the network. The backward step is similar to the forward one, except that error values, calculated by 3.7, are propagated back through the network, in order to determine how the weights are to be changed during the training phase. The basic procedure for the backpropagation algorithm is reported in Figure 3.4.

```
Initialize network weights randomly
while not termination condition do
    Assign as net input to each unit in the input layer its corresponding element
    in the input vector. The output for each unit is its net input
    Calculate network output by forwarding input signals in the network
    Calculate the error of each output neuron
    for all hidden neurons do
        calculate weights updates
        propagate the error back through the network.
    end for
    Update weights of the network.
end while
```

Figure 3.4: Pseudocode of the backpropagation algorithm.

The training is supervised by having a target pattern associated with an input pattern. A pattern is presented to the network and an error vector is calculated to determine how the weights should change; the process is then repeated for each pattern. An epoch is a complete cycle through each pattern. The patterns are continually presented to the network, epoch after epoch, and training continues until the change in the absolute value of the averaged squared error falls to within some tolerance between one epoch and the next one.

3.4 Consideration for ANN design

In the ANN design, the successful application of a neural network usually demands much experimentation. There are a number of parameters to be set that can affect how easy a solution is to find. Some of these parameters are the kind of neural network that have to be considered, the number of layers and nodes that define the network architecture, and the connection weights. The training data are also an important factor and a good deal of attention must be paid to the test data to make sure that the network will generalize correctly on data which has not been trained on.

There are no easy answers to these questions and no standard design recipes exist for designing a neural network to solve a given problem.

3.4.1 Architecture

The architecture design is crucial in the successful application of ANNs because has significant impact on a network information processing capabilities. Given a learning task, an ANN with only a few connections and linear nodes may not be able to perform the task at all due to its limited capability, while an ANN with a large number of connections and nonlinear nodes may overfit noise in the training data and fail to have good generalization ability. The main problem is that there is no systematic way to design a near-optimal architecture for a given task automatically.

A survey on supervised learning by evolving MLPs was presented by Ribert and colleagues [100], where several kinds of neural architecture evolving techniques are presented. In particular, four techniques are discussed in [100], implemented in incremental and decremental algorithms to achieve training convergence. They considered progressive error minimization, transformation of a neural tree into an MLP, active data selection and evolutionary algorithms approaches.

Research on incremental and decremental algorithms represents an effort toward the automatic design of architectures:

- In *Incremental algorithms* hidden neurons are added to a network of minimum size until the required precision is reached. This algorithm starts with a minimal network (network with minimal number of hidden layers, nodes, and connections) and adds new layers, nodes, and connections when necessary, during training, until the error is sufficiently small. The main problem with these approaches is that huge ANNs are usually obtained and the redundant information stored in the weights is never eliminated.

- *Decremental algorithms* start with a huge network and delete unnecessary layers, nodes, and connections during training. The problem with these algorithms is that they start with excessively large networks, which slows training down; furthermore, the units to eliminate and their elimination order have to be guessed correctly.

However, as indicated by Angeline and colleagues [4], ‘Such structural hill climbing methods are susceptible to becoming trapped into structural local optima’. In addition, they ‘only investigate restricted topological subsets rather than the complete class of network architectures’ [4].

Yao [133] considers the design of the optimal architecture for an ANN as a search problem in the architecture space, where each point represents an architecture. Given some performance (optimality) criteria, i.e., lowest training error, lowest network complexity, etc., about architectures, the performance level of all architectures forms a discrete surface in the space. The optimal architecture design is equivalent to finding the highest point on this surface. Miller et al. [78] report several characteristics of such a surface, which are summarized below:

- the surface is infinitely large since the number of possible nodes and connections is unbounded;

- the surface is nondifferentiable since changes in the number of nodes or connections are discrete and can have a discontinuous effect on EANN performance;
- the surface is complex and noisy since the mapping from an architecture to its performance is indirect, strongly epistatic, and dependent on the evaluation method used;
- the surface is deceptive since similar architectures may have quite different performance;
- the surface is multimodal since different architectures may have similar performance.

All these characteristics reported by Miller [78], make EAs a better candidate for searching the surface than incremental and decremental algorithms mentioned above.

3.4.2 Selecting Data

One of the most important factors for training neural networks is the availability and the integrity of data. They should represent all possible states of the problem considered, and they should have enough patterns for building also the test and validation set.

The data selected for the training set must be representative of the complete space that a class might occupy. For instance, if two classes are located very close together, it is important to include the data from the boundary that separates the classes to ensure that the network be able to correctly identify the two regions; otherwise samples in that part could be misclassified.

The consistency of all data has to be guaranteed, since it is common to find errors in a large data set, introduced by an operator or simply missing data in one or more pattern. Outlayers are exceptions that sometimes can exist in the data. Outliers are points that stand out from the rest of the data and can produce errors due to incorrect recording of the original information. In this situation, care has to be taken because outliers can be treated as representative of useful information. For example, some faults can occur during data real measurements.

The training data must be representative of the problem, but if the network is allowed to adapt too well to the training data by being given it too many degrees of freedom, then the residual error will be very small but the network is likely to fail to correctly map new, previously unseen input data of the same class, i.e., it will have poor generalization capabilities. This phenomenon is called *overfitting*. In order to avoid overfitting, a new method called *early stopping* is implemented. With this approach the overall data set can be divided into three separated sets, training, test and validation set.

Test and validation sets should be selected at random and they also need to be representative of the problem. There is no agreement in the literature on the way these sets are named: some approaches use the convention that the validation set is used to avoid overfitting, while the test set is used to validate the approach. Instead, in some other conventions, test set is used to monitor network training, avoiding overfitting, while the validation set is used in order to assess the quality of the neural network. In this thesis, the second convention is adopted. A graphical representation of the use of training, test and validation sets is shown in Figure 3.5.

In this case, training set is used for computing the gradient and updating the network weights and biases. This set can be halted periodically and the test data passed through, and overfitting can be prevented by monitoring the training with the test set. The error on

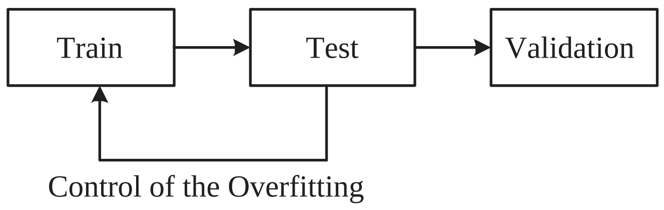


Figure 3.5: Representation of the training, test and validation set correlation.

the test set is monitored during the training process. Such error will normally decrease during the initial phase of training, as does the training set error, and when data start being overfit the error on the test set will begin to rise. At this point training is stopped, and the weights and biases at the minimum of the test set error are returned. A validation set acts like another test except that its data have not been presented to the network in any form during training. The validation set uses its data in order to validate the neural network.

3.4.3 Handling local minima

The backpropagation algorithm is based on a gradient descent technique. Usually, a gradient descent algorithm is used to adapt the weights based on a comparison between the desired and actual network response to a given input stimulus. In each iteration of backpropagation, the gradient of the search surface is calculated and network weights are changed in a direction opposite to the gradient. Gradient descent works fine while the slope of the error function is smoothly heading downhill, but in practice the error surface for a large network will have many valleys, hills and folds.

Therefore, it is not uncommon for the backpropagation algorithm to get stuck in local minima. A local minimum might be close to a solution but not close enough to satisfy the problem requirements; furthermore, backpropagation is not able to finding a global minimum if the error function is multimodal and/or non-differentiable.

This can be computationally expensive if a large number of iterations is required to find an acceptable network and to avoid local minima entrapment. Moreover, BP is sensitive to initial conditions and it can become slow. Oscillations may occur during learning, and, if the error function is shallow, the gradient is very small leading to small weight changes.

3.4.4 Generalization

Generalization represents the ability of a learning system to correctly map new inputs, not used during training into the correct output. Good generalization depends on the training set and the network architecture.

An improvement of the generalization ability of a neural network is given by the early stopping method, defined in Section 3.4.2, which also prevents the overfitting problem.

The appropriate number of hidden units and the number of training data are also related and play a role in the quality of the resulting network. A large network may create complex functions, but a small architecture may not have enough power to fit the data. As previously indicated, the problem is that is difficult to know beforehand how large a network should be for a specific application. A method for improving generalization is called *regularization*. This involves the modification of a performance function, which is normally chosen like the following equation, similar to the error backpropagation function:

$$MSE = \frac{1}{N}\sum_{j=1}^{N}(t_j - o_j)^2 \tag{3.8}$$

that corresponds to the typical performance function used for training feedforward neural networks. This is called Mean Square Error value (MSE). With this method, networks with high dimensions will be penalized. The problem with regularization is that the optimum value for the performance ratio parameter is difficult to determine. Too large a value of this parameter may give overfitting; on the other hand, if the ratio is too small, the network will not adequately fit the training data.

Evolutionary Algorithms can be considered as useful methods to solve this problem. Indeed, with EAs, the performance function of a neural network can be 'evolved' proportionally to the fitness function of the evolutionary algorithm. This thesis will present in the following chapters an example of this approach, in which fitness function of each individual depends on the MSE of the correlated network.

Considering the size of data sets, there are no rigorous results here, but a rule of thumb asserts that the number of hidden units needed will increase as the number of training data increases for a given level of performance. General guidelines exist for network design, like those suggested by Haykin [53], who defines the training data set size as a function of the number of the network weights and of the percentage of errors allowed in the testing phase.

3.4.5 Training process

Another important issue in the ANN design considers the training process, carried out by adjusting the connection weights iteratively, so that learned ANNs can perform the desired task. Weight training is usually formulated as the minimization of an error function, such as the mean square error between target and actual outputs averaged over all examples, by iteratively adjusting connection weights. In methods which are most frequently used to train neural networks, BP has emerged as the standard algorithm for finding a set of satisfactory connection weights and biases [48], like previously indicated. There have been some successful applications of BP in various areas [65], but BP has different drawbacks due to its use of gradient descent, previously indicated as local minima trapping, non-differentiability of the function and sensitivity to initial conditions.

One way to overcome the shortcomings of gradient-descent-based training algorithms is to adopt EAs, i.e., to formulate the training process as the evolution of connection weights in the environment determined by the architecture and the learning task.

EAs can then be used effectively to find a near-optimal set of connection weights globally without computing gradient information. The fitness of an ANN can be defined according to different needs. Two important factors which often appear in the fitness (or error) function are the error between target and actual outputs and the complexity of the ANN. Unlike in gradient-descent-based training algorithms, the fitness (or error) function does not have to be differentiable or even continuous since EAs do not depend on gradient information.

Chapter 4

Evolutionary Artificial Neural Networks

4.1 Evolutionary Learning

Chapter 2 uses the general term ‘evolutionary algorithms’ to define algorithms implementing evolutionary computation. They are useful for complex optimization problems, where the number of parameters is large and the analytical solutions are difficult to obtain, and they help to find out the optimal solution globally over a domain.

A particular type of evolving systems, namely neuro-genetic systems, have become a very important topic of study in evolutionary computation. They are included in the framework of evolutionary learning and, as defined by Yao et al. [137], they are biologically-inspired computational models that use evolutionary algorithms in conjunction with neural networks (NNs) to solve problems.

In this sense, evolutionary artificial neural networks (EANNs) become an important topic of study in ANN design. EANNs refer to a special class of artificial neural networks in which evolution is another fundamental form of adaptation in addition to learning [133]. One distinct feature of EANNs is their adaptability to a dynamic environment. In other words, EANNs can adapt to an environment as well as to changes in the environment. The two forms of adaptation, i.e., evolution and learning in EANNs, make their adaptation to a dynamic environment much more effective and efficient. EANNs can be regarded as a general framework for adaptive systems, i.e., systems that can change their architectures and learning rules appropriately without human intervention.

Much research effort has been spent to improve the performance of EAs and different selection schemes and genetic operators have been proposed in the literature. As previously indicated, evolutionary learning for ANNs has also been introduced to reduce and, if possible, to avoid the problems of traditional gradient descent techniques, such as BP, that lies in the chance of being trapped in local minima. EAs are generally much less sensitive to initial conditions of training. They always search for a globally optimal solution, while a gradient descent algorithm can only find a local optimum in a neighborhood of the initial solution. EANNs provide a solution to these problems and an alternative for the task of controlling the complexity of the network.

One of the most important benefits of evolutionary artificial neural networks is that, considering generalization, a near-optimal artificial neural network with both structure and weights can be evolved automatically without going through a tedious trial-and-error manual design process.

Several researches regarded the design of an ANN as an optimization problem. In [117], discussions about evolutionary systems and their interaction with neural and fuzzy systems are presented. Some EAs have implemented search over the topology space, or a search for the optimal learning parameters and neural network transfer functions. Some others focus on weight optimization: these can be regarded as alternative training algorithms, and in this case the evolution of weights assumes that the architecture of the network must be static. An interesting area for new research directions lies in the conjunction of these techniques, combining the advantages of each of them.

The aim of this chapter is to provide, seven years later, an update of Yao's milestone review [133], taking into account the most recent literature about EANNs. Evolutionary algorithm and state of the art design of Evolutionary Artificial Neural Network (EANN) were also introduced by Abraham [2], followed by the proposed MLEANN framework. In that framework, in addition to the evolutionary search of the connection weights and architectures, local search techniques were used to fine-tune the weights (meta-learning).

This chapter is organized as follows: Section 4.2 defines the approaches to evolving ANNs, and is linked to the corresponding main categories of evolutionary computation. Section 4.3 focuses on weight optimization describing some work presented in the literature. The process of learning-rules optimization is then reported in Section 4.4, briefly describing one of the most important works, carried out by Chalmers, and other approaches carried out later, based on the same idea, while the transfer function optimization process is described in Section 4.5. The evolution of network topologies is presented in Section 4.7, introducing a distinction between direct and indirect encoding, and then considering some related work. One of the most important evolutionary approaches regards the simultaneous evolution of architecture and weights described in Section 4.8; also in this section several applications are described.

4.2 Evolving ANN Approaches

There are several approaches to evolving ANNs and EAs which are used to perform various tasks, such as connection weight training, architecture design, learning rule adaptation, input feature selection, connection weight initialization, rule extraction from ANNs, etc. Three of them are considered as the most popular approaches at these levels:

- *Connection weights*, that concentrates just on weight optimization, assuming that the architecture of the network must be static. The evolution of connection weights introduces an adaptive and global approach to training, especially in the reinforcement learning and recurrent network learning paradigm where gradient-based training algorithms often experience great difficulties.

- *Learning rules*, that can be regarded as a process of 'learning to learn' in ANNs, where the adaptation of the rules is achieved through evolution. It can also be regarded as an adaptive process of automatic discovery of novel learning rules.

- *Architecture*, that enables ANNs to adapt their topologies to different tasks without human intervention, and provides an approach to automatic ANN design, as both

ANN connection weights and structures are evolved. In this kind of optimization method, a further subdivision can be made by distinguishing between a 'pure' architecture evolution and a simultaneous evolution of both architectures and weights.

Other approaches consider the input feature selection and the evolution of the transfer functions of a neural network, but they are usually applied in conjunction with one of the three methods above in order to obtain better results.

All these approaches usually fall into different categories of evolutionary computation. The most important are described in Chapter 2 and regard genetic algorithms, which define other two particular sub-branches like genetic programming and evolution programs, evolutionary strategies and evolutionary programming techniques.

Another class of optimization programs has been presented in the recent literature in [88], where two classes of evolutionary methods are identified. The first one considers a typical approach, which uses a population of gradient-learning ANN undergoing weight adaptation through BP training and structure evolution through EA. The second considers approaches that rely solely on EA for both ANNs structure evolution and weight adaptation. They are so-called *invasive* and *non-invasive* approaches:

- *non-invasive* method: it uses evolutionary algorithms, but fitness evaluation requires local methods, like BP or other gradient training techniques. In this approach, the explicit separation between network adaptation by local optimal approaches and structure evolution by EAs often requires the development of a dual representation scheme, so it is natural for this approach to adopt a GA-type evolution. It does not change heavily the typical learning mechanisms of the individual network. The EA is only used as a background process during evolution. Its successful performance still heavily relies on the proper initialization of BP parameters and the proper choice of BP implementation. Individual network still undergoes gradient error minimization which is prone to the 'local optima' problem. An approach that implements a non-invasive method is also defined a hybrid algorithm, like previously described in Chapter 2.

- *invasive* method: it relies solely on an EA for ANN evolution. Since weight adaptation and structure evolution are carried out directly using the perturbation functions of evolutionary algorithms, it avoids the mapping problem by representing individuals at the genotype level. By using direct representation and avoiding BP fitness evaluation, the main EA operations are fast and make it feasible to use a larger population size for a more robust search coverage. Important issues that have to be considered in this approach are the development of an appropriate encoding scheme, supporting causality and evolvability, and the development of an appropriate stopping criterion to avoid premature learning and overlearning.

One major issue defined by Palmes and colleagues [88] in their work considers the reliance to dual representations in ANN genetic algorithm implementations. Like indicated in Chapter 2, during the mapping from genotype to phenotype deceptive problems [45] may occur. Some of the major consequences of these problems include the permutation problem or the many-to-one problem and the opposite one-to many problem, like indicated in [137]. Such problems also occur in architecture optimization, along with evolvability and causality problems. In each mapping phase must be ensured that small changes in the genotype must corresponding to small changes in its phenotype, trying to prevent evolution from becoming inefficient and hard to control.

Another problem that can arise in the non-invasive method regards the use of BP. Indeed the high sensitivity of BP to the initial setup of its parameters causes a noisy evaluation of the corresponding fitness function, which makes the entire process unreliable. Furthermore, no well defined stop-training criterion is defined in this algorithm. For these reasons, several approaches preferred invasive approaches.

The use of evolutionary learning for designing neural networks is no more than two decades old. However, a lot of work has been made in these years, which has produced many approaches and working models for different ANNs optimizations. Some of these are reported below.

A synoptic table, 4.1, is reported in order to summarize the approaches presented in the literature and also in this chapter, for each of the different evolutionary ANN techniques considered.

Table 4.1: Synoptic table of some evolving ANNs techniques presented in literature.

EANNs	
Techniques	Examples in Literature
Weight Optimization	GA with real encoding (Montana et al) GENITOR (Whitley et al) Mutation-based EAs (Keesing et al) Improved GA (Yang et al) NN weight evolution (Zalzala et al) MLP training using GA (Seiffert) STRE (Pai)
Parameter Optimization	GA for competitive learning NNs (Merelo Guervós et al) G-prop II/III (Merelo Guervós et al) ANOVA (Castillo et al)
Rule Optimization	GA for learning rules (Chalmers) GP for learning rules (Poli et al)
Transfer Function Optimization	EANNs through EPs (Yao et al) Hybrid method with GP (Poli et al)
Input Data Selection	EAs for fast data selection (Brill et al) Selecting Training set (Reeves et al)
Architecture Optimization: Constructive and destructive algorithms	Design of ANN (Yao et al) Design NN using GA (Miller et al) NEAT EP-Net Evo-design for MLP (Filho et al) genetic design of NNs (Harp et al)
Simultaneous Evolution of Architecture and Weights	ANNA ELEONORA (Maniezzo) EP-Net (Yao et al) Improved GA (Leung et al) COVNET (Pedrajas et al) CNNE (Yao et al) MGNN (Palmes et al) GNARL (Angeline et al) GAEPNet (Tan)

4.3 Weight Optimization

The evolution of weights can be regarded as an alternative to training algorithms, and assumes that the architecture of the network must be static. The primary motivation for using evolutionary techniques to establish the weight values rather than traditional gradient descent techniques such as BP [103], lies in avoiding being trapped in local minima and the requirement that the activation function be differentiable. For this reason, rather than adapting weights based on local improvement only, EAs evolve weights based on the fitness of the whole network.

Several approaches in this direction have been presented for network weight optimization which use genetic algorithms.

Genetic algorithms with real encoding Montana and Davis [80] encode the biases and the weights of a neural network in a list of real numbers and initialize them by choosing a random probability distribution function, that reflects the empirical observation that optimal solutions tend to contain weights with small absolute values. The rationale of this choice is to allow the genetic algorithm to explore the range of all possible solutions, favoring those solutions which are *a priori* defined as the most likely. In their work, Montana and Davis created different types of genetic operators, that can be grouped in three basic categories, namely mutations, crossovers and gradients. Mutation applies random perturbation to some of the entries in the chromosome of an individual (i.e. neural network), in order to create its offspring. Crossover generates one or two children containing some of the genetic material of each of two selected parents. The gradient operator defines the child of a selected individual by adding to its entries a multiple of the gradient value with respect to the evaluation function. The goal of their approach was to find out how different operators perform in different situations, and thus be able to select a good set of operators for that problem. This idea was carried out by retaining useful feature detectors formed around hidden nodes during evolution. Their results showed that the evolutionary training process was much faster than BP for the problems they considered.

GENITOR: genetic algorithms with binary encoding Whitley and colleagues implemented a purely genetic approach using binary encodings of weights, called GENITOR [123]. In this algorithm each string was evaluated and the population was sorted by ranking strings in terms of their evaluations. A random selection function with a linear bias towards the higher ranked strings was used to stochastically choose two parents for recombination. The parents were then recombined to produce a single offspring (in this algorithm two offspring were created, but one was randomly discarded). After the offspring was evaluated, it replaced the lowest ranked string in the population and was inserted into its appropriate rank location. Generally speaking, one or two offspring can be created and replacement can be probabilistic such that lower ranked strings are typically replaced.

Whitley and colleagues carried out, in [105], a comparison between the GENITOR algorithm and that implemented by Montana and Davis. Considering the latter, the main differences regard the kinds of network representations, with real-valued strings implemented instead of binary encoding. Then, the representation of each weight was by a single real value so that recombination occurred only between weights. In this algorithm, too, a small population was considered, and mutation rates were higher than those used by most traditional genetic algorithms. The approach of Montana and Davis also provided an option that improved offspring using backpropagation, obtaining better solutions than those implemented by considering only BP.

Whitley and colleagues then implemented, in [122, 124], a modified version of the GENITOR algorithm, to reflect the algorithm changes used by Montana and Davis. They defined a new network representation, based on real-valued encoding, a small population and relatively high mutation rates. With this new approach they found that the modified genetic algorithm produced better results, competitive with the backpropagation algorithm.

Mutation-based evolutionary approaches In other work presented in the literature, an EA and a gradient descent algorithm have been combined [63]. Usually, as described in Chapter 2, a classical genetic algorithm requires a modification of an original problem into an appropriate form, suitable for it, as well as a well-defined dual representation and an algorithm for mapping the genotype into phenotype of the neural network. In order to overcome these problems of GAs, like in [133], a natural way to evolve real-number chromosome representation would be defined by the use EP or ES, since they are particularly well-suited for treating continuous optimization. Unlike GAs, the primary evolutionary operator in these techniques is mutation. One of the major advantages of using mutation-based evolutionary approaches is that they can reduce the negative impact of the permutation problem, providing a more efficient process.

Improved genetic algorithm Few years ago, Yang and colleagues [131] proposed an improved genetic algorithm based on a kind of evolutionary strategy. Often, during the application of GAs, some problems of premature convergence and stagnation of solution can also occur [46]. Indeed, higher selective pressure often leads to the loss of diversity in the population, causing solutions to converge prematurely. A genetic algorithm, based on evolutionary stable strategy (ESSGA), was implemented to keep the balance between population diversity and convergence speed during evolution. This is obtained by means of a kind of mutation operator in conjunction with a controller stable factor: this mutation operator only acts on some of the preponderant individuals under the control of such stable factor. ESS was proposed by Smith [111], which defines a controller that keeps the percentage of quantity of preponderant individuals to stable quantity of the population dimension in each generation.

With this solution the population diversity is maintained by restricting the over reproduction of preponderant individuals, and the search space is enlarged as well. The authors confirm that, with their approach, the selective pressure can be alleviated, and the problem of premature convergence can be avoided without increasing the running time. Yang and colleagues carried out experiments with the "XOR-problem", showing an increase in speed and accuracy.

NN evolution with mutation and multi-point crossover Zalzala and Mordaunt [81] studied an evolutionary NN suitable for gait analysis of human motion evolving the connection weights of a predefined feed-forward neural network structure. As previously indicated, traditional methods for training neural networks can have some problems in finding an optimal solution when the error surface is multi-modal. In this case evolutionary algorithms are not influenced by the impact of the random initialization of network weights. In Zalzala's work, real number representation was chosen to evolve weights, and evolution was analyzed with mutation and a multi-point crossover separately implemented as the best combination search mechanism. In the mutation operator, mutation rate and step size are varied, the latter being the maximum value that can be added/subtracted to/from a gene value, while in the recombination algorithm the variable parameters are crossover rate and the number of crossover points. In order to implement multi-point crossover, two binary chromosomes, equal in length to the actual, real number, chromosomes were created, one with all zeros and the other with all ones. These new chromosomes were split into portions and a single point crossover was performed on each portion. Multi-point crossover was performed on the binary chromosomes and these were decoded to produce the offspring of the original real number chromosomes. Simulation results showed that both mutation and crossover type evolution produce systems that are good classifiers when compared with the control MLP network. The results indicate also that the BP algorithm is over-fitting the training data, giving an error more than that obtained with the evolutionary approach.

MLP training using GA Training of MLPs using genetic algorithms was also carried out by Seiffert [108], who described an approach to completely substitute a traditional gradient descent algorithm by a genetic algorithm in the training phase. In this work the architecture of the neural network was predefined and remained fixed after initialization, and the chromosome solely consisted of the weight values and did not contain any topological or structural information. Genetic operators like selection and reproduction were considered as the main operators to perturb weights, while mutation was only taken as secondary operator. Several benchmark problems were implemented by the author in order to make a comparison with BP solutions. Results demonstrated that the more complex the problems, the more BP failed, and the more considering its substitution by a GA was advantageous.

STRE: Short Term Reproduction Expectancy Recently, Pai [87] proposed a genetic approach employing a genetic inheritance operator, the so-called Short Term Reproduction Expectancy (STRE), that has been implemented to determine the weights of an EANN, considering a multi-layer feedforward neural network with a predefined fixed topology. The most relevant aspect of the STRE scheme is that a selective set, among the best fit parent chromosomes, is allowed a short-term life expectancy and participates in the subsequent reproduction process, along with a portion of highly-fit offspring chromosomes. In other words, the life expectancy of a portion of the best-fit parents in a population is extended by a short term of at least one more generation, while the rest of the chromosomes in the mating-pool are the best-fit offspring.

Unlike evolutionary strategies, the author does not use mutation operators in the evolution cycle, but only defines a two-point crossover for reproduction, and applies it to decimal coded chromosomes, corresponding to non-binary encoding, with values between 0 and 9. The inverse of the root mean square of the error obtained while learning the training data is set as the fitness value for each chromosome.

The author made a the performance analysis of this scheme on the application of prediction of uplift capacity in the field of geotechnical engineering, and concluded that STRE allowed for faster convergence and reduced learning error in comparison with its predecessor approach. The new method made a closer prediction to the expected values than the previous solution.

4.4 Learning Rules Optimization

Supervised learning algorithms are, as previously defined in Chapter 3, the most frequently used methods to train ANNs, and, among them, standard backpropagation is the first and most applied method for training multilayer networks. Unfortunately, this method presents some drawbacks, like its sensitivity to initial conditions and trapping in local minima. Furthermore, since oscillations may occur during learning, usually an increase in the learning rate results in an unfruitful attempt to speed up convergence. The design of training algorithms, in particular the learning rules used to adjust connection weights, depends on the type of architecture of the considered neural network. Several standard learning rules have been proposed, but designing an optimal learning rule becomes very difficult when there is little priori knowledge about the network topology, producing a very complex relationship between evolution and learning.

The evolution of learning rules is considered as an interesting application of evolutionary algorithms to the design of neural networks, and has been applied in several works in the past. The evolutionary approach is important not only in providing an automatic way of optimizing leaning rules and in modeling the relationship between learning and evolution, but also in modeling the creative process since newly evolved learning rules can deal with a complex and dynamic environment.

4.4.1 Parameter Optimization

The first kind of optimization considers the adjustment of learning parameters and can be seen as a first attempt to evolve learning rules. Learning parameters comprise BP parameters, like the learning rate and momentum, and genetic parameters, like mutation and crossover probabilities. They can be difficult to assign by hand, and therefore become good candidates for evolutionary adaptation. Typically, the parameters are encoded into the gene code of each individual and allowed to evolve.

Several studies have been carried out in this direction: Merelo and colleagues [49], presented a search for the optimal learning parameters of multilayer competitive-learning neural networks. These authors presented also another well-known approach, the so-called G-prop [26, 25, 27]. This algorithm can also be considered as an evolutionary approach for architecture and weights evolution, as indicated in Section 4.8. Indeed, as indicated by Merelo and colleagues, the algorithm is designed to determine the learning parameters, the initial weights, and a suitable hidden-layer size of multilayer perceptrons, setting the parameters that the method requires.

This algorithm is considered a hybrid algorithm, as described in Chapter 2, since trains multi-layer perceptrons based on a GA-BP approach. G-prop uses neither binary nor real encoding, i.e. representations of the networks in a binary or real number string, nor an indirect coding. Instead, the initial parameters of the network, initial weights and learning constants, are evolved using specific genetic operators, such as mutation, crossover, addition, and elimination of hidden neurons and the training QP operator, which act on the MLP data structure. In this approach, an EA is implemented to find a solution close to the global optimum, in conjunction with local search algorithm, the backpropagation algorithm, in order to tune a solution and reach the nearest local minimum by means of local search from the solution found by EA.

The genetic operators are included directly on the ANN object, but only initial weights and the learning constant are subject to evolution, not the weights obtained after training. In particular, in the mutation operator, the learning rate is modified by adding a small random number with a uniform distribution. The crossover operator performs a multi-point crossover between two chromosome nets and swaps the corresponding learning parameters. G-prop have been evaluated using different classification problems, presented in the literature. Experiments proved that this method can achieve better results than other BP algorithms, giving also more information like network size and learning parameter values.

Moreover, the ANalysis Of the VAriance (ANOVA) tool was applied by the authors [26] to determine whether the influence of a change in the parameter values over the obtained error and/or size is significant to establish the most suitable value for these parameters. This is equivalent to obtain satisfactory solutions as fast as possible, and decide whether the element operation, for example genetic operator or selector, is as desired. In most of these cases, the goal is to obtain the smallest generalization error, while keeping the size as small as possible. The parameters considered by the authors took into account evolutionary and learning processes, and were defined by Merelo and colleagues to be the number of EA generations, the size of the population, the initial weight range, that was also depending on the initial point of the search space, the selection rate, the number of training epochs and finally the genetic operators application priority. the ANOVA was used to determine whether the effect of all these parameters on the error and size obtained was statistically significant. The most suitable value was determined using the ANOM tool. A high number of parameters was involved in this step, and they were grouped into different sets. ANOVA was applied in four steps to different sets of parameter simulations.

The results presented in this work showed that those parameters that affected directly the MLP, like the training operator application priority and the learning constant mutation range, had greater effects on the error and size obtained. Then, initial weight generation range and selection percentage, that are two parameters related to the initial population, influenced the results obtained, and were statistically significant as expected. Merelo and colleagues concluded confirming that the fact that parameters directly related to the MLP and not to the EA had significant effects, indicated that such parameters were even most important in the hybrid methods, contrary to the claims of those researchers which used only a GA to train MLP, and justifying also their aim of optimizing the parameters by means of the EA.

Another method to search for the optimal set of weights, the optimal topology and learning parameters, using EA and BP, was proposed by Castillo et al. [23]. In this work, however, the learning constant was set by hand.

4.4.2 Rule Optimization

Considering the field of learning-rule optimization, one of the first studies was conducted by Chalmers [28]. The aim of his work was to see if the well-known delta rule, or a fitter variant, could be evolved automatically by a genetic algorithm. Chalmers fixed the changes in the weight of a given connection to be a function only of information local to the connection. The learning rule was expressed as a quadratic function. A number of assumptions were made at the outset, in order to restrict the form of the learning algorithm to a linear function of the relevant parameters, corresponding to the dependent variables: the input, output and target values, as well as the weight change and a scale parameter similar to the learning rate constant. The pairwise products of the parameters were also used in the linear combination. The function had the following form:

$$\begin{aligned}\Delta w_{ij} \quad = \quad & k_0(k_1 w_{ij} + k_2 o_{jp} + k_3 o_{ip} + k_4 t_{ip} \\ & + k_5 w_{ij} o_{jp} + k_6 w_{ij} o_{ip} + k_7 w_{ij} t_{ip} \\ & + k_8 o_{jp} o_{ip} + k_9 o_{jp} t_{ip} + k_{10} t_{ip} o_{ip})\end{aligned} \tag{4.1}$$

The network architecture was feedforward with input and output layers only, which can only learn mappings that are linearly separable. With a suitable chromosome encoding and using a number of linearly separable mappings for training, Chalmers was able to evolve a rule analogous to the delta rule, as well as some of its variants. Although this study was limited to somewhat constrained network and parameter spaces, it paved the way for further progress.

Chalmers also noticed that discovering complex learning rules using GAs is not easy, mainly due to the fact that the discovery of learning rules used highly complex genetic coding, that makes the search space large and hard to explore, while GAs used a simpler coding which allows known learning rules as a possibility, making the search very biased. In order to overcome limitations caused by GAs during learning-rule evolution, Chalmers suggested that GP, particular kind of GA, might have an advantage over GAs. Several studies have been carried out in this direction and some of them are described, along with a new approach, in the work presented by Poli and colleague [96].

Their approach considered a genetic programming algorithm with the objective to explore a larger space of rules using different parameters and two different rules for the hidden and output layers. Their long-term objective was to obtain a rule which was general, like standard backpropagation algorithm, but that was faster and more stable than the canonical approach. The function identified to discover learning rules was with the following form:

$$\Delta w_{ij}^{l}(s) = F_o(w_{ij}^{l}, o_{jp}^{l}, t_{ip}, o_{ip}^{l+1}) \tag{4.2}$$

$$F_h(w_{ij}^{l}, o_{jp}^{l}, o_{ip}^{l+1}, \epsilon_{ip}^{l+1}) \tag{4.3}$$

The first equation $F_o$was defined for the output layer, while the second one, F_h for the hidden layers. In these equations o_{jp}^{l} was the output of neuron x_j^l when pattern p was presented to the network, and ϵ_{ip}^{l+1} was the error of the considered layer.

In the first stage of the algorithm, Poli and colleague used GP to evolve rules for the output layer, while the hidden layers were trained with the standard backpropagation algorithm. In the second stage, they used GP to evolve rules for the hidden layers, while the output layer was trained with the best rule discovered in the first stage. In the experiments carried out, they used two different activation functions to attempt to obtain activation function-independent learning rules, and, in order to test the generality, additional experiments were carried out on several problems, like recognition and classification, different from those implemented to obtain the rules.

The most interesting result of their experiments was that BP has discovered a learning rule in which the weaknesses of a supervised learning rule, like standard backpropagation, were removed by combining it with an unsupervised learning rule, the Hebbian rule. So, their experiments suggested that a good learning rule for the hidden layers should have the following form:

$$LR_h = \eta SBP + \beta HB \tag{4.4}$$

with η and β representing the learning rates, respectively . To see if the rule could be improved even more by using some of the SBP speed-up techniques, the authors compared, in another set of tests, its convergence behavior with SBP, with and without the Momentum and Rprop speed-up algorithms. In this run the learning rule NLR defined with the GP approach achieves its target output at the same epoch as SBP with Momentum, while NLR with Momentum converges much more quickly than the other algorithms. Further runs were carried out, by comparing NLR with Rprop and SBP with Rprop: in the character recognition problem considered, the NLR outperformed SBP.

The results obtained indicated that rules found perform very favourably with respect to canonical algorithms, with faster optimized rules. The authors concluded that their work indicates that there are supervised learning algorithms that perform better and which are more stable than standard backpropagation learning rule and that genetic programming can help to discover them.

4.5 Transfer Function Optimization

Usually the transfer function of all neurons of a neural networks is defined at the beginning and maintained fixed during the evolution, although some attempts have been made to allow its adaptation over the generations. Transfer function perturbations can begin with a fixed function, like linear, Sigmoidal or Gaussian, and allow the genetic algorithm to adapt to a useful combination according to the situation. In this direction some work has been carried out by Yao and colleague in [129] in order to apply a transfer function adaptation over generations. In order to obtain better and more interesting solutions, often, this kind of evolution is carried out together with the other kinds of neural network optimizations, described in the other sections.

In this section also another work is presented in the literature, carried out by Poli and colleagues [95]. Their work considered a particular hybrid method, where genetic programming evolved a mapping function to adapt the weights, whereas a genetic algorithm-based approach evolved the architecture. GP was previously used in several other works to evolve architecture and weights simultaneously [66], or to evolve rules for constructing neural networks [96]. Here it is used to build a non iterative mapping function, that is evolved concurrently to the architecture.

The authors defined a parse tree to implement the mapping function in their algorithm. In particular, all individuals, genotypes, in the population were structure which had one part to describe the architecture of the encoded network, and a second part to represent the function to map the raw random weights, which were fixed throughout a run, into the values used to evaluate the network performance. In order to evolve the architecture, they implemented a modified version of a previous work [94], in which a two dimensional individual representation was defined.

In this work, the most important issue regarded the representation of the individuals of the population, and particular attention was given to the function representation. Indeed, in order to define the parse tree in the second part of the genotype, a set of terminals, different from those used for the first part of the genotype, and a set of functions was defined. The set of terminals included a variable representing the weight which the function was applied to, and also a variable used to initialize random constants when the individuals of the initial population were created. In order to simultaneously evolve such parts of each element, Poli and colleague defined also a particular kind of 'combined' crossover operator, that was applied first to the architecture of both parents, and then to the encoding of the mapping function. In this second part, a standard GP crossover, described in Chapter 2, was performed by replacing subtrees of the parents.

Several experiments were carried out in this work, also considering benchmark problems. In particular, two applications to feedforward and to recurrent neural networks were described respectively. Results compared well with other approaches and were promising. Furthermore, the authors said that their method not only had suitable results, but it also opened new possibilities for GP in EANNs.

4.6 Input Data Selection

EAs are attractive for dimensionality reduction of the input data set when it is quite large. Indeed, in these situations there may be some redundancy among different inputs. A large number of inputs to an ANN increase its size and thus require more training data and longer training times in order to achieve a reasonable generalization ability. The input data reduction can be made with EAs without loss in performance [20]. In this approach, each individual in the population represents a portion of the input data. The ANN is trained with these individuals and the result is part of its fitness. Another related issue is the partitioning of the input data of a network into a training and a validation set. This operation is almost always done quite arbitrarily, although it may influence the network performance significantly. For this reason, Reeves and colleagues [99] apply genetic algorithms to select training sets for a kind of neural network.

4.7 Architecture Optimization

Architecture design is an important issue in the successful application to the ANN evolution, because the architecture has significant impact on the network information processing capabilities. There is no systematic way to design a near-optimal architecture for a given task automatically. For this reason, pattern classification approaches [121] can be used to design the network structure, and constructive and destructive algorithms represents an effort toward the automatic design of network topologies [133]. The constructive algorithm starts with a small network. Hidden layers, nodes, and connections are added to expand the network dynamically [137]. The destructive algorithm starts with a large network. Hidden layers, nodes, and connections are then deleted to contract the network dynamically [83]. These algorithms define incremental and decremental neural networks respectively, as described in detail in Chapter 3.

The design of an optimal NN architecture can be formulated as a search problem in the architecture space, where each point represents an architecture. As pointed out by Yao [137, 139, 133], given some performance (optimality) criteria, e.g., minimum error, learning speed, lower complexity, etc., about architectures, the performance level of all these forms a surface in the design space. Determining the optimal architecture design is equivalent to finding the highest point on this surface. There are several features of the surface considered which make the case for using EAs for searching for the best network topology, and which make them better candidates rather than incremental and decremental algorithms [78, 113]. Characteristics presented by Miller [78] all refer to the surface of possible solutions, and they are described in detail in Chapter 3.

Stanley and Miikkulainen in [113, 114] presented a neuro-evolutionary method using augmenting topologies (NEAT), that is designed to take advantage of structure as a way of minimizing the dimensionality of the search space of connection weights. So, this approach can also be included into the class of algorithms for simultaneous evolution of architecture and weights. NEAT was designed specifically to address three main issues, like outperforming the solutions that employ a principled method of crossover of different topologies, protecting structural innovation using speciation, and incrementally growing from minimal structure. Structural mutation expanded the genome, adding a connection between two previously unconnected nodes, or adding a node; in this case an existing connection was split and the new node placed where the old connection used to be; the old connection was disabled and two new connections were added to the genome. This method of adding nodes was chosen by the authors to integrate new nodes immediately into the network. Speed was asserted to be one benefit of this approach, since smaller structures optimized faster, so the system was able to optimize the minimal number of connections necessary to obtain a solution. Furthermore, being trapped in local minima may be avoided by adding new connections to network topologies. Crossover was implemented by taking a chronology of every gene in the system, in order to cross over only genomes with the same historical origin. Stanley and colleagues implemented this method in order to solve the problem of competing conventions for disparate topologies, and avoiding the need for expensive topological analysis.

The authors also underlined that, in contrast with methods generally implemented to seed the initial population, in their work the population was seeded with a uniform distribution, with no hidden nodes. Since NEAT protected innovation using speciation, a network started from a minimal configuration and the topology was grown only as necessary. Structure were modified, and only those structures that were found to be useful through fitness evaluations survived. In this way they also said that the number of generations necessary to find a solution were significantly reduced.

In NEAT also the fitness landscape was altered by structure-mutation operators. The authors concluded that the ablation studied demonstrated that historical markings, speciation, and incremental growth from minimal structure were all integral components of efficient evolution of network structure.

4.7.1 Critical Issues

One of the most important forms of deception in ANNs structure optimization arises from the many-to-one and from one-to-many mapping from genotypes in the representation space to phenotypes in the evaluation space. The existence of networks functionally equivalent and with different encodings makes evolution inefficient. This problem is usually termed as the *permutation problem* [50] or the *competing convention problem* [105]. It is clear that the evolution of pure architectures has difficulties in evaluating fitness accurately. As a result, evolution would be very inefficient. Other important issues in evolving architecture regard the genotype representation scheme and the definition of the EA used to evolve the network topology. In the encoding phase, an important issue is to decide how much information about an architecture should be encoded into a chromosome (genotype).

Then, the performance of MLPs strongly depends on the topology of the networks, considering size and structure. As a result, the definition of the network topology characterizes networks features like its learning process speed, learning precision, noise tolerance and generalization capacity. There are two major ways in which EAs have been used for searching network topologies: either all aspects of a network architecture are encoded into an individual or a compressed description of the network is evolved. The first case defines a *direct* encoding, while the second leads to an *indirect* encoding.

Direct Encoding

In direct encoding each parameter of the neural network is exactly specified and little effort in decoding is required, since a direct transformation of genotypes into phenotypes is defined. Under this scheme, the connection topology is represented by means of an adjacency matrix, that is, an N-node architecture is represented by an $N \times N$ matrix A, where $a_{ij} = 1$ means that there is a connection between units i and j and $a_{ij} = 0$ stands for no connection. An individual in a population of architectures is simply the string resulting from the concatenation of successive rows of the matrix. Several examples of this approach are shown in the literature, like in [78, 124]. Another work is presented in [137], in which the direct encoding scheme is used to represent ANN architectures and connection weights (including biases). EP-Net [137] is based on evolutionary programming with several different sophisticated mutation operators and is described also in Section 4.8.

This encoding is easy to understand and to implement, but it also has a major drawback, that refers to the fact that it does not scale well, since an N-node network potentially has on the order of N^2 connections, leading to very long chromosomes. As a consequence, training a whole population of networks by backpropagation or similar methods can be extremely slow. Another problem of this approach is that incorrect structures can be produced, i.e, it can, for example, produce feedback connections for feedforward networks. Direct encoding is thus only useful for small architectures.

Indirect Encoding

In view of the scalability problems brought about by direct encoding methods and of their consequences in terms of performance, several researchers focused their efforts on techniques for developing or growing neural networks, rather than looking for a complete network description at the individual level. Indirect representations, on the other hand, require a considerable effort for neural network decoding, but, in some cases, the network can be pre-structured, using restrictions in order to rule out undesirable architectures, which makes the searching space much smaller. A few sophisticated encoding method is implemented based on network parameter definitions. These parameters may represent the number of layers, the size of the layers, i.e., the number of neurons in each layer, the bias of each neuron and the connections among them.

Although no general conclusions as to whether this approach is actually better can be drawn yet, indirect encoding is an interesting idea that has been further pursued by other researchers, like Filho and colleagues [39], and Harp and colleagues [51]. Their method is aimed at the choice of the architecture and connections, and uses a representation which describes the main components of the networks, dividing them in two classes, i.e., parameter and layer sections. In particular, the parameter section specifies the learning rate and the momentum term for all network connections; while the layer section specifies the number of units in each layer. In this algorithm, network performance evaluation is based on the error costs of the network and the losses caused by the network in each transaction, giving a fitness which is representative of network performance. In the reproduction phase, the roulette wheel method was used to select the candidates and networks with higher fitness were preferentially chosen. Furthermore, an elitist policy was implemented, and the architecture with the highest fitness is automatically copied in the next generation. The genetic operators implemented in this work consider crossover as the predominant operator, while mutation is defined as the secondary operator, only responsible for slight qualitative changes in the network features architecture.

4.8 Simultaneous Evolution of Architecture and Weights

One solution to takle the effects of the noisy fitness evaluation problem in ANNs structure optimization is to consider a one-to-one mapping between genotypes and phenotypes of each individual. This is possible by performing simultaneous evolution of the architecture and the network weights. The advantage of combining these two basic elements of a NN is that a completely functioning network can be evolved without any intervention by an expert.

Some methods that evolve both the network structure and the connection weights were proposed in the literature. Castillo and colleagues [25] present a method to search for the optimal set of weights, the optimal topology and learning parameters using a genetic algorithm for the network evolution and backpropagation for network training, even though the initial learning constant is set by hand.

ANNA ELEONORA: simultaneous evolution with genetic algorithm Maniezzo proposed a genetic algorithm [75], ANNA ELEONORA, implemented for learning both topology and connection weights, considering two design techniques. The first is a genetic operator, called GA-simplex [17]. The second one is an encoding procedure, called granularity encoding [73, 74], that allows the algorithm to autonomously identify an appropriate suitable length of the coding string. In this approach an extended direct encoding scheme is defined, where each connection is represented directly by its binary definition. This kind of representation uses the network nodes as basic functional units and encodes all information relevant for a node in nearby positions, including its input connectivity pattern and the relative weight distribution. Connectivity is coded by presence/absence, i.e. 1/0, bit values, defining the *connectivity bits*. When connection is present, after each connectivity bit there is the binary encoding of the relative weight. The first byte of the string specifies the *granularity*, i.e., the number of bits according to which the weights of the present connections have been codified in the binary representation. In the algorithm implemented, coding granularity is a control parameter that evolves concurrently to the net structure, in order to find the best solution in the space of the granularities. This approach employs four genetic operators, reproduction, crossover, mutation and GA-simplex, and two versions, sequential and parallel, were considered respectively. A particular attention was given to the GA-simplex algorithm. This is a ternary operator implemented to exploit the fitness landscape identified by the solutions. GA-simplex algorithm operates on three individuals of the population in order to generate a new individual. The new individual is generated considering the binary string encodings of the three individuals, ranked by their fitness values. A bit to bit comparison is carried out between the three individuals. If the first and the second bits are the same, the new string maintains, for that bit position, the same bit value of the first individual, otherwise, the opposite bit value of the third individual is assigned to the new string. The ANNA ELEONORA algorithm has been validated with several problems. Results of this approach, applied to Rumelhart's test suite, showed the effectiveness of GA-simplex algorithm for architecture evolution, even though applying it too often can lead to early convergence.

EP-Net: evolve feedforward ANN with EP An evolutionary system, EP-Net [137], was also presented for evolving feedforward ANNs. The evolutionary algorithm implemented in EP-Net to evolve ANNs is based on Fogelś evolutionary programming [44]. The idea behind this approach is to put more emphasis on evolving ANN behaviors; in particular, a number of techniques have been adopted to maintain a close behavioral link between parents and their offspring. In EP-Net, the only mutation operation for modifying ANN weights is implemented by a hybrid training algorithm, consisting of a modified backpropagation (MBP), with adaptive learning rules, and a simulated annealing (SA) algorithm. It could be regarded as two mutations driven by BP and SA algorithms separately. The main purpose of this weight mutation algorithm is to discourage architectural mutation if training, which often introduces smaller behavioral changes in comparison with architectural mutations, can produce a satisfactory ANN. Only when the hybrid training algorithm fails to reduce the error of the neural network, architectural mutations will take place. In the architecture mutation algorithm node or connection deletions are always attempted before connection or node additions, in order to encourage the evolution of small ANNs. Connection or node additions will be tried only after deletions fail to produce good offspring. EPNet evolves ANN architectures and weights simultaneously in order to reduce noise in fitness evaluation even though evolution simulated by this approach is closer to Lamarckian than to Darwinian evolution. In their follow-up study [138] Yao and colleagues combined the solutions of EP-Net population ensembles, and produced solutions which compared well with those obtained from isolated networks. Since the best solution found in the first work was obtained only on a validation set, without considering a testing set, Yao and colleagues considered then all individuals of the last generation, and linearly combined them together, forming an esemble. They considered this as a method to show the importance of using population information, but they also said that non-linear combination methods could give better results.

Improved genetic algorithm Leung and colleagues developed a new system [69] for tuning the structure and parameters of a neural network in a simple manner. A given fully connected feedforward neural network may become a partially connected network after training. The topology and weights are tuned simultaneously using a proposed improved GA. In this approach the weights of the network links govern the input-output relationships mapping of the NN, while the structure of the neural network is governed by introducing switches elements for each NN connection.

COVNET: co-evolutionary models Simultaneous evolution of architecture and weights of a network was also implemented in a new kind of models for evolving ANNs, named *co-evolutionary models*. García Pedrajas and colleagues presented COVNET, an example of cooperative co-evolutionary method [89]. In COVNET each species is a subnetwork that constitutes a partial solution of a problem; the combination of several individuals from different species makes up the network that is to be applied to the specific problem. Here a population of networks evolves by means of a steady-state genetic algorithm that keeps track of the best combinations of modules for solving the problem, and different species must cooperate in order to be rewarded with high fitness values. In this work the diversity is maintained all along evolution due to the fact that each species is evolved without exchanging genetic material, since this may produce non-viable offspring, and may reduce population diversity. Although usually direct comparison with other work is difficult because the algorithms and methods to obtain the generalization of the models are different, the results of the model in solving three real problems were compared with a modular network, and with the results of other work presented in the literature. In particular, for each of the three problems, Pedrajas and colleagues presented results comparable with those obtained with EPNet algorithm [137] and with an ensemble of networks evolved with EPNet [138].

CNNE: constructive algorithm with NN ensembles Another recent work by Yao defines a new constructive algorithm, called constructive NN ensemble (CNNE) [58], for training cooperative NN ensembles. This approach emphasises both accuracy and diversity among individual NNs in an ensemble. In order to maintain accuracy among individual NN, the number of hidden nodes in individuals are also determined by a constructive approach. Incremental training based on negative correlation is used to train individual NNs for different numbers of training epochs, which are determined automatically by its training process. CNNE algorithm determines automatically also the number of individuals in an ensemble. The cost function considered in this work, for determining ensemble architecures, is the ensemble error. This is quite different from some previous works that divide the cost function into accuracy and diversity. In this sense a major drawback of CNNE lies in the inherent difficulty in weighing and combining the two factors in one function. Another important feature of this approach regards architecture modifications. Indeed, in CNNE, only when the criteria for node or NN addition are met, will architecture modifications take place. For architecture modifications, node additions are always attempted before NN additions. CNNE maintains diversity among individual NNs primarily by using negative-correlation training. Hidden nodes are added to individual NNs in a constructive fashion to improve the ensemble accuracy. Network additions to an ensemble will be attempted only after adding a certain number of hidden nodes to individual NNs has failed to reduce the ensemble error significantly. Such an ordering by which training is preferred to adding hidden nodes in existing NNs and growing the existing NNs is preferred to adding new NNs bears certain similarity to previous Yaoś work on the order of application of architectural mutation in evolving NNs [137].

MGNN: mutation based genetic neural network Further work was carried out in order to address drawbacks of BP gradient descent approach previously introduced. P.P. Palmes and colleagues [88] implemented a mutation-based genetic neural network (MGNN) to replace BP by using an invasive approach based on the mutation strategy of local adaptation, typical of evolutionary programming (EP), to perform weight learning. This algorithm also dinamically evolves structure and weights at the same time and a stopping criterion is implemented, by monitoring overfitness occurrences in order to avoid premature learning and overlearning. Individuals are represented by vectors and matrices of real numbers as connection weights of the NN. These vectors and matrices are subjected to random permutations during mutation to improve the ANN fitness. Dynamic structure changes and local adaptation of weights through mutation are implemented using a defined perturbation function, based on an adapted strategy parameter and on a mutation strength intensity parameter, dynamically computed during evolution.

In MGNN a gaussian perturbation is implemented together with a stochastic (GA-inspired) and a scheduled stochastic (EP-inspired) mutation. In the first method each weight in connection matrices and threshold vectors have the same probability of perturbation.

On the other hand, the scheduled mutation assigns higher rather than lower probabilities. MGNN works under the principle that individuals located away from the best solution need drastic changes to improve their fitness than those located near the optimal solution [4].

The stopping criterion is based on the observation that a good training performance does not necessarily imply a good validation performance. MGNN uses the validation data to measure overfitness occurrences in an interval sampling, *sliding-window*. The validation performance of the currently fittest network is compared with the validation performance of the second fittest networks: if there is overfitness in n consecutive windows, the stopping criteria will can the training to stop. The last best network found before the training stops is the optimal solution found by MGNN.

GNARL: evolutionary algorithm for recurrent neural networks Another 'invasive' evolutionary approach, namely the GNARL algorithm [4], has been carried out by sharing some of features employed by the previous MGNN [88], such as the use of structural and weight learning by mutation and the definition of sparse connections instead of uniform and symmetric topologies. In this algorithm, the number of hidden nodes and connection links for each network is randomly chosen within a defined range from 0 to a user-specified limit, and network weights are defined as real values. In the genetic core of this approach the selection strategy chooses the better half of the entire population to become the parents of the new generation. The reproduction is then carried out using two types of mutation: parametric mutation, that is carried out by changing network weights using Gaussian noise, structural mutation, that involves addition or deletion of nodes or connections, that are selected uniformly within a user-defined interval. GNARL defines also three fitness functions, namely sum of squared errors, sum of absolute errors and sum of exponential absolute errors, respectively. In any case, since the algorithm does not use any gradient information, like in MGNN, changes in the fitness function have no significant bearing on the evolutionary process.

GAEPNet: hybrid evolutionary algorithm A recent hybrid evolutionary approach for designing neural networks for classification was presented [116]. This work introduced a linear combination crossover operating on a real-valued multi-matrix encoding. A hybrid evolutionary algorithm, the so-called HEA, was proposed to combine the crossover operator in GAs with the mutaiton operator in EP, defining a new method called GAEPNet, where connection weights and architecture of ANNs evolve simultaneously. In this approach, the real-valued multi-matrix encoding scheme was presented to encode each feedforward ANN with one hidden layer as a genotype encompassing the matrices, which described all weight and bias values of all neurons. Such a scheme encoded both architecture and weights and eliminated the need for an interpretation function, indicating that the dual space problem was avoided. It was also a flexible scheme as the only restriction on the architecture was to set a maximum number of hidden neurons.

To benefit from both EP and GAs, Tan introduced a hybrid algorithm where the proportion of crossover and mutation changed adaptively. Tan set crossover as the dominant variation operator, and the conventional crossover used to generate offspring was arithmetic recombination operator; the mutation operators were parametric and structural; the latter one was implemented only when the parametric had been failed to increase the fitness of an ANN. Both genetic operators were followed by a partial training using backpropagation algorithm, with a fixed number of epochs aiming at increasing the behavioural link between the parents and their offspring, and avoiding overfitting. Experiments were carried out on benchmark problems. After a comparison with other approaches in the literature, the author concluded that his hybrid algorithm was capable of generating ANN with high stability and generalization, and defined as a further improvement the significant reduction in computational time.

4.9 Hybrid EANNs and Baldwin Effect

Hybrid evolutionary algorithms, previously described in Chapter 2, become useful in the ANN design. Hybrid neuro-evolutionary approaches may be inspired on Darwinian or Lamarckian evolution. Several studies that consider the lamarckian or the darwinian mechanisms are presented in the literature, and Merelo Guervós and colleagues presented in a recent work [22] a state of the art conducted in this direction. The aim of their work was to carry out an experimental study into how learning can improve G-Prop genetic search on MLP neural networks. Two ways of combining learning and genetic search were explored: one exploited the Baldwin effect, while the other used a Lamarckian strategy. Their experiments showed that using a Lamarckian operator makes the algorithm find networks with a low error rate, and the smallest size, while using the Baldwin effect obtains MLPs with the smallest error rate.

Some studies presented in the literature have investigated whether a strategy based on a hybrid evolutionary neural network approach that take advantage of the Baldwin effect is better or worse than one implementing Lamarckian mechanisms, but the results obtained are different and problem dependent.

In the case of Darwinian evolution, the Baldwin effect, that is, the progressive incorporation of learned characteristics to the genotypes, can be observed and leveraged to improve the search. Baldwin effect is a consequence of hybrid algorithms. Several studies presented in the literature have demonstrated the Baldwin effect using a variety of learning algorithms. Hinton and Nowlan [55] presented a contribution to understand the interactions between learning and evolution, in order to better explain the main issues raised when a Baldwin effect is considered in a computational model.

In order to make some of these issues more explicit, studies of Baldwin effect under the general assumption of quantitative genetics has been described by Anderson [7, 3]. In this sense it is important to underline that all essential elements of an evolutionary process subject to the Baldwin effect determine the entire evolutionary process. These elements include the generation of new genotypes thorough mutation and/or recombination, the mapping from genotype to phenotype and the assumption that learning allows an individual to modify its phenotype in response to its environment. An important aspect that has to be considered is such cases is that the introduction of individual learning can radically alter fitness landscapes. This is especially true if the learning algorithm operates on phenotypes according to a fundamentally different process. Anderson also defined in his work [7] that, under certain conditions, learning slows genetic change by protecting suboptimal genotypes from selection. Thus, the benefits of individual learning probably are accrued early in optimization, when the population is far from equilibrium, and learning can eventually impede algorithmic convergence. Accordingly, for optimizations on fixed fitness landscapes, a 'variable learning investment' strategy, where the computational resources applied toward learning are subject to change, should be considered [3].

In the approach implemented in this thesis, the learning process of all the individuals in the population can be carried out by using the backpropagation algorithm, which produces also a Baldwin effect, because BP allows to each learnt individual (e.g. neural network) to modify its phenotype in response to its environment.

4.10 Situating the State of the Art

The three most important lines in the evolutionary artificial neural network research have summarized in this chapter as the optimization of basic elements of a neural network, as the connection weights, the learning rules and the architecture. Starting from these, also other different lines can be depicted, and one of the most interesting defines the simultaneous evolution of the architecture and weights of a neural network.

When a connection weight evolution is considered, the architecture of the neural network is fixed. One of the most critical aspects that can rise from this technique is that the structure that will be used to define all neural networks in a population has to be set at first time, giving some problems when such a topology is difficult to define at first step. Also in learning rule evolution, the design of training algorithms, in particular the learning rules used to adjust connection weights, depends on the type of the neural network architecture. Therefore, the design of such rules can become very difficult when there is a little priori knowledge about the network topology, giving a complex relationship between evolution and learning. The architecture evolution has an important impact on the neural network evolution, and also the recent literature shows how the evolution of pure architecture presents difficulties in evaluating fitness accurately. For this reason, in recent works, sophisticated mutation operators are defined in order to overcome these problems. Nowadays, the simultaneous evolution of architecture is one of the most interesting evolutionary ANNs techniques. Different works are carried out in this directions, considering different kinds of genetic operators that act on weights and topologies, taking advantage from the combination of these techniques.

Chapter 5

Neuro-Genetic Approach

5.1 Introduction

As defined in Chapter 1 the subject of this P.h.D. dissertation aspires to the definition of a robust, well defined and easy approach, that is able to solve complex problems, that could be identified with patter recognition, classification, etc., by using soft computing techniques. In particular, the aim of this work regards the definition and the implementation of an approach in order to design and optimize artificial neural networks by using evolutionary algorithms.

The attractiveness of ANN, as described in Chapter 3, comes from the remarkable information processing characteristics such as nonlinearity, high parallelism, robustness, fault and failure tolerance, learning, ability to handle imprecise information, and their capability to generalize. It is important to underline that the successful application of an ANN usually requires a high number of experimentation, and all the features are available only in a well defined ANN. Moreover, several parameters of an ANN can affect, during the design, how easy a solution is to find. Some of these parameters are related to the architecture design of the neural network, concerning the number of layers and nodes, and the connection weights. Some others consider the selection of data that will define the training, the test and the validation set, in order to guarantee their availability and their integrity. Other important factors consider, then, the handling of local minima and the training of the ANN, trying to avoid the entrapment in local minima, and to avoid the overfitting of the network. Finally, a good deal of attention must be paid to the data set definition, so that the network will generalize correctly on data which it has not been trained on.

There is no standard design that is able to solve all these questions for a given problem. Evolutionary algorithms become, in this sense, helpful and they represent a good solution to solve the problem of the ANN design. As previously described in Chapter 2, the evolutionary process is a more integrated and rational way of designing ANNs since it allows single aspects of the design to be taken into account at the same time as several interacting aspects and does not require any expert knowledge of the problem. Evolutionary algorithms are especially useful for complex optimization problems where the number of parameters is large and the analytical solutions are difficult to obtain, and they can help to find out the optimal solution globally over a domain.

A particular type of evolving systems, namely neuro-genetic systems, have become a very important topic of study in evolutionary computation and, as presented in Chapter 4, they define the so-called EANNs, that are biologically-inspired computational models that use evolutionary algorithms in conjunction with neural networks to solve problems.

5.2 Evolutionary Algorithm

Evolutionary algorithms can be applied to neural networks in several ways. The most important are already described in the previous chapter, and consist of setting the weights in a fixed topology network, determining network structures, evolving learning rules and input feature selection. Several systems also allow an interesting conjunction of the evolution of network architecture and weights, carried out simultaneously. The last one also corresponds to the approach implemented in this thesis.

Important aspects of the simultaneous evolution underline that an evolutionary algorithm allows all aspects of a neural network design to be taken into account at once, without require any expert knowledge of the problem. Furthermore, the conjunction of weights and architecture evolution overcomes the possible drawbacks of each single technique and joins their advantages. The main advantage of weight evolution is to simulate the learning process of a neural network, avoiding the drawbacks of the traditional gradient descent techniques, such as BP. Given then some performance optimality criteria about architectures, as momentum, learning rate, etc., the performance level of all these forms a surface in the design space. The advantage of such representation is that determine the optimal architecture design is equivalent to find the highest point on this surface. The simultaneous evolution of architecture and weights limits also the negative effects of a noisy-fitness evaluation in a ANN structure optimization, by defining a one-to-one mapping between genotypes and phenotypes of each individual.

The general idea implemented is similar to other approaches presented in the literature, but it differs from them in the novel aspects implemented in the genetic evolution.

This work can be considered as a hybrid algorithm, since a local search based on the gradient descent technique, backpropagation, can be used as local optimization operator on a given data set. The basic idea is to exploit the ability of the EA to find a solution close enough to the global optimum, together with the ability of the BP algorithm to finely tune a solution and reach the nearest local minimum.

The approach is designed to take advantage of BP when possible and beneficial; however, it can also do without it. Backpropagation becomes useful when the minimum of the error function currently found is close to a solution but not close enough to solve the problem; BP is not able to find a global minimum if the error function is multimodal and/or non-differentiable. Moreover, the adaptive nature of NN learning by examples is a very important feature of these methods, and the training process modifies the weights of the ANN, in order to improve a pre-defined performance criterion, that corresponds to an objective function over time. In several methods to train neural networks, BP has emerged as a suitable solution for finding a set of good connection weights and biases. The neuro-genetic approach defines the possibility to take into account different learning process for neural network training by defining a parameter of the algorithm: in the case in which the user would implement a training process of the neural network with backpropagation algorithm, a bp parameter will be set to 1. Otherwise, the training process will be carried out only considering the genetic evolution, and the bp parameter will be set to 0. A supervised learning scheme, as presented in Chapter 3, is implemented in the approach, in which each instance of the data set includes all input attributes and the target output. During the training process the set of training data is repeatedly applied to the network until the difference between the outputs and the target values are within a desired tolerance.

A peculiar aspect of this approach is that BP is not used as some genetic operator, as is the case in some related work [93]. Instead, the EA optimizes both topology and weights of the networks; BP is optionally used to decode a *genotype* into a *phenotype* NN. Accordingly, it is the genotype which undergoes the genetic operators and which reproduces itself, whereas the phenotype is used *only* for calculating the genotype's fitness.

Feedforward neural networks are a basic type of ANNs capable of approximating generic classes of functions, including continuous and integrable ones. The neuro-genetic approach restricts the attention to a specific subset of feedforward neural networks, namely MLP. This neural network has features such as the ability to learn and generalize, smaller training set requirements, fast operation, ease of implementation and with simple structures. MLPs are described in detail Chapter 3, and they are feedforward NNs with one layer of input neurons, one layer of one or more output neurons and one or more "hidden" (i.e., internal) layers of neurons in between; neurons in a layer can take inputs from the previous layer only. In the algorithm, fully connected feedforward MLP will be considered for simplicity.

The idea proposed in this work is close to the solution presented in EPNet [137]: a new evolutionary system for evolving feedforward ANNs, that puts emphasis on evolving ANNs behaviors. This neuro-genetic approach evolves ANNs architecture and connection weights simultaneously, as EPNet, in order to reduce noise in fitness evaluation.

Close behavioral link between parent and offspring is maintained by applying different techniques, like weight mutation and partial training, in order to reduce behavioral disruption. Genetic operators defined in the approach include:

- Truncation Selection,
- Mutation, divided into:
  - weight mutation
  - topology mutation
- Crossover.

that will be described in detail in the rest of this chapter.

In this context, the evolutionary process attempts to mutate weights before performing any structural mutation; however, all different kinds of mutation are applied before the training process. Weight mutation is carried out before topology mutation, in order to perturb the connection weights of the neurons in a neural network. After each weight mutation, a weight control is carried out, in order to delete neurons whose contribution is negligible with respect to the overall network output. This allow to obtain, if possible, a reduction of the computational cost of the entire network before any architecture mutation.

Particular attention has to be given to all these operators, since they are defined in order to emphasize the evolutionary behavior of the ANNs, reducing disruptions between them. Examples are also reported in the description of the weights and topology mutations, described in detail in the rest of this chapter.

As indicated in the introduction, this thesis is primed by an industrial application [2, 1], in which a neural engine-controller design is implemented, with particular attention to reduced power consumption and silicon area occupation. The validity of the resulting approach, however, is by no means limited to hardware implementations of NNs. A further application describes a brain wave signal processing system[5], in particular a classification algorithm for the analysis of P300 Evoked Potential. Finally, the third application considers two financial problems, whereby a factor model capturing the mutual relationships among several financial instruments is sought for, and a time series prediction problem.

5.2.1 Detailed Description of the Algorithm

The general framework of the evolutionary process can be described by the pseudo-code shown in Figure 5.1. Individuals in a population compete and communicate with other individuals through genetic operators applied with independent probabilities, to perform certain tasks. In this thesis the objective function of the problem is directly proportional to the global cost of each neural network considered. For this reason, the convention that the best fitness corresponds to the lower fitness is adopted, defining the objective of each task as a cost minimization problem. The fitness function defined for each individual of the population is described in detail in section 5.4.

```
Initialize the population, either by generating new random individuals or by
loading a previously saved population, with a specified dimension parameter
for each genotype do
  Create the corresponding MLP
  calculate its mean square error (MSE), its cost, and its fitness value
end for
Save the best individual as the best-so-far individual with the best (lowest) fit-
ness value
while not termination condition do
  Apply genetic operators to each network
  Decode each new genotype into the corresponding network
  Compute the fitness value for each network
  Save statistics of the generation
end while
```

Figure 5.1: Pseudocode of the evolutionary process.

The genetic cycle iterates until the termination conditions are reached. In this approach they are defined as a maximum number of executions, including those carried out during the training phase, to be reached; or the maximum number of evolution generations, or, simply the achievement of a satisfactory solution.

The genetic operators are described by the pseudo-code depicted in Figure 5.2. They are used to generate offspring, new individuals, from parents, existing individuals, and they define the evolutive cycle.

5.2.2 Initialization

The framework of the neuro-genetic approach requires that a new population be created either by loading a previously saved population, or by generating a new one. In this P.h.D. work, all applications implemented create a new population at first. In this case, a number of individuals that corresponds to the dimension of the population, previously defined, is

```
Select from the population (of size $n$) $\lfloor n/2 \rfloor$ individuals by truncation
Create a new population of size $n$ with copies of the selected individuals
for all individuals in the population do
    Perform crossover
    Mutate the weights and the topology of the offspring
    Train the resulting network using the training and test sets if $bp = 1$
    Calculate $f$ and $\hat{f}$ (see Section 5.4), i.e. the fitness values calculated on train
    and test sets respectively
    Save the individual with lowest $\hat{f}$ as the best-so-far individual if the $\hat{f}$ of the
    previously saved best-so-far individual is higher (worse)
end for
Save the statistics of the generation
```

Figure 5.2: Pseudocode of the evolutionary process.

created and all individuals have not pre-established topology, the population is initialized with different hidden layer sizes and different number of neurons for each individual, in order to maintain diversity between all the individuals in the new population. Two exponential distributions are used to determine the number of hidden layers and the number of neurons for each layer in each individual, while a normal distribution is used to initialize all the weights and biases values of the new network. Variance matrices are also defined for all weights and bias matrices, and will be applied in conjunction with evolution strategies in order to perturb network weights and bias. Variance matrices are initialized with matrices of all ones.

Unlike other approaches, like [139], the maximum size, that corresponds to the number of the hidden layers, and the maximum number of neurons in each layer is not determined in advance, nor bounded. This method is implemented in order to avoid the definition of more algorithm parameters, like the boundaries of network topology dimension. In this direction, the fitness function implemented in this approach works as a controller and selector, because it penalizes large networks.

With this solution the problem of over-sized neural networks is avoided.

5.3 Individual Encoding

Each individual is encoded in a structure that maintains basic information on the network as illustrated in Table 5.1. The values of all elements that define the individual representation are affected by the genetic operators during evolution. Such genetic operators, and in particular the topology mutation operator, enable the algorithm to perform incremental (adding hidden neurons or hidden layers) and decremental (pruning hidden neurons or hidden layers) learning.

An example of a simple full connected feedforward MLP neural network, defined in the neuro genetic approach, is depicted in Figure 5.3. The neural network structure is created with a number of hidden layers (together with the output layer) equal to the genotype vector size. The input layer is equal for all the neural networks, so is not specified in the genotype vector. The number of hidden nodes in the i^{th} hidden layer corresponds to the number specified in the i^{th} element in the genotype vector. All the nodes in each layer are connected

Table 5.1: Individual Representation.

Element	Description
l	Length of the topology string, corresponding to the number of layers.
topology	String of integer values that represent the number of neurons in each layer.
$\boldsymbol{W}^{(0)}$	Weight matrix of the input layer neurons of the network.
$\mathbf{Var}^{(0)}$	Variance matrix of the input layer neurons of the network.
$\boldsymbol{W}^{(i)}$	Weight matrix for the ith layer, $i = 1, \dots, l$.
$\mathbf{Var}^{(i)}$	Variance matrix for the ith layer, $i = 1, \dots, l$.
b_{ij}	Bias of the jth neuron in the ith layer.
$\mathbf{Var}(b_{ij})$	Variance of the bias of the jth neuron in the ith layer.

to all the nodes of the following one with the corresponding matrix of weights, defined for any layers. For each weight matrix, the corresponding Variance matrix is also created, and is used to perturb, in the mutation, the corresponding weights. For each layer also the bias vector is defined, which contains, in each element, the bias value of the corresponding node in the neural network.

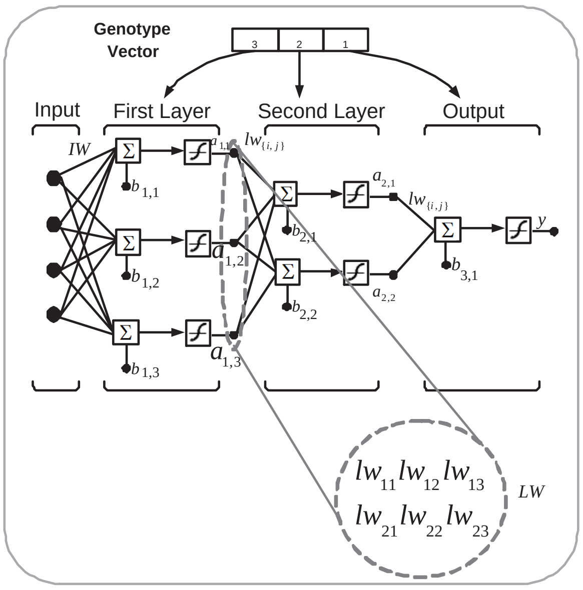


Figure 5.3: Representation of an individual.

Table 5.2 lists all the parameters of the algorithm, and specifies the values that they assume when the default setting is taken for a considered problem.

Some problem-specific parameters of the algorithm are the costs α of a neuron and β of a synapsis used to establish a parsimony criterion for the network architecture; a *bp* parameter, which enables the use of BP if set to 1, and other parameters like probability values used to define topology, weight distribution and *ad hoc* genetic operators.

All the experiments are carried out by specifying the algorithm parameters and by tun-

Table 5.2: Parameters of the Algorithm.

Symbol	Meaning	Default Value
n	Population size	60
seed	Previously saved population	none
bp	Backpropagation selection	1
p^{+}_{layer}	Probability of inserting a hidden layer	0.1
p^{-}_{layer}	Probability of deleting a hidden layer	0.05
p^{+}_{neuron}	Probability of inserting a neuron in a hidden layer	0.05
p_{cross}	Probability of crossover	0.2
r	Parameter used in weight mutation for neuron elimination	1.5
ϵ	Alternative threshold used for neuron elimination	0
h	Mean for the exponential distribution	3
N_{in}	Number of network inputs	*
N_{out}	Number of network outputs	*
α	Cost of a neuron	2
β	Cost of a synapsis	4
λ	Desired tradeoff between network cost and accuracy	0.5
k	Constant for scaling cost and MSE in the same range	10^{-5}

*) Problem-specific dependent.

ing the genetic parameters to obtain the best solution. A few experiments are carried out in each problem without considering backpropagation, i.e. by setting BP parameter equal to 0. However, the results obtained in that cases have been showed solutions worse with respect to those obtained in experiments with backpropagation. An example of such results is showed in Section 7.3.1 in Tables 7.2 and 7.3.

Particular attention has to be given to the bp parameter, because its use allows one to adopt two different types of genetic encoding schemes, as defined in Chapter 3:

- *Direct Encoding*, that specifies, in the genome, every connection and node that will appear in the phenotype;
- *Indirect Encoding*, that usually only specifies rules for constructing a phenotype. These rules can be layer specifications or growth rules through cell division.

Generally, indirect encoding allows for a more compact representation than direct encoding, because not every connection and node are specified in the genome, although they can be derived from it. On the other hand, the major drawback of indirect schemes is that they require more detailed genetic and neural knowledge. In this evolutionary process, if no BP-based network training is employed, a direct encoding is defined, in which the network structure is directly translated into the corresponding phenotype; otherwise, there is an indirect encoding of networks, where the phenotype is obtained by training an initial (embryonic) network using BP.

While promising results can be obtained by combining backpropagation and evolutionary search, fast variants of backpropagation are sometimes required to speed up the efficiency of these algorithms. In this work, considering the computational trade-offs between local and evolutionary search, the Resilient backpropagation algorithm RPROP [98] is adopted as the local search method. The aim of this algorithm is to eliminate the harmful effects of the magnitudes of the partial derivatives, during the training phase. In particular, in this method only the sign of the derivative is used to determine the direction of the weight update, while the magnitude has no effect on them, as is modified by a separate update process.

5.4 Fitness Function

An optimization problem requires a solution such that a certain quality criterion, called *objective function*, is maximized (or, equivalently, minimized). The fitness of each individual in a population, reflects its objective function value with respect to the particular objective to be optimized.

Although it is customary in EAs to assume that better individuals have higher fitness, the convention that a lower fitness means a better NN is adopted in the neuro-genetic approach. This maps directly to the objective function of the genetic problem, which is a cost minimization problem.

In this work two kinds of fitness function are defined, and, for each application considered, only one has been chosen and applied to each individual of the population in the evolutionary process.

As defined in successful evolutionary approaches to neural network evolution presented in the literature [137], the general difficulty in using a fitness function like $f = f_{error} + \alpha f_{complexity}$ in practice lies in the selection of suitable coefficient α, which often involves tedious trial-and-error experiments.

Moreover, each of the two fitness function implemented in this thesis has to solve a multi-objective problem, because each of them is defined by combining two aspects that usually come into conflict, which are respectively the cost and the accuracy of the neural network considered. In particular an arbitrary parameter λ which specifies the desired trade-off between network cost and accuracy is defined in the two fitness functions. Accordingly an array of Pareto-optimal designs should be identified by the approach.

The two kinds of fitness functions are defined as follows:

- *MSE Fitness Function*: this kind of fitness depends both on the accuracy, that is its mean square error, and on the cost of each individual. Therefore, the fitness is proportional to the value of the MSE and to the cost of the considered network. It is defined as in Equation 5.1:

$$f = \lambda k c + (1 - \lambda)\mathrm{MSE} \tag{5.1}$$

  where $\lambda \in [0, 1]$ is a parameter which specifies the desired trade-off between network cost and accuracy, k is a constant for scaling the cost and the MSE of the network to a comparable scale, and c is the overall cost of the considered network. This cost is defined by Equation 5.2:

$$c = \alpha N_{hn} + \beta N_{syn} \tag{5.2}$$

  where N_{hn} is the number of hidden neurons, and N_{syn} is the number of synapses.

The MSE depends on the *Activation Function*, the same for all nodes of a layer of the neural network. In this work the *tangent sigmoid* transfer function is implemented according to Equation 5.3:

$$y = \frac{2}{1+e^{-2x}} - 1 \tag{5.3}$$

- $\mathrm{Corr}_{\mathrm{coeff}}$ *Fitness Function*: this fitness function is proportional to the statistical correlation coefficient, so-called $\mathrm{Corr}_{\mathrm{coeff}}$ and to the cost of each individual. The new coefficient is defined by Jones in [60]. The correlation fitness is defined between the target output and the output obtained from the neuro-genetic evolution. This coefficient is expressed by Equation 5.4:

  $$\mathrm{Corr}_{\mathrm{coeff}(i,j)} = \frac{\mathrm{Cov}(i,j)}{\sigma_i \sigma j} \tag{5.4}$$

  where $\mathrm{Cov}(i,j)$ represents the Covariance of the considered values, σ_i and σj represent the corresponding standard deviation values.

  Then, the fitness function is defined with the equation 5.5:

  $$f = \lambda k c + (1-\lambda)(1 - \mathrm{Corr}_{\mathrm{coeff}}) \tag{5.5}$$

  in which the parameters λ, k, and c, correspond to the same parameters of the MSE fitness function 5.1, and they respectively represent the desired trade-off between network cost and accuracy, the constant for scaling the cost and the MSE, and the overall cost of the considered network, that is defined by the same equation 5.2.

For both objective functions, the rationale behind introducing a cost term in the objective function is the search for networks which use a reasonable amount of resources (neurons and synapses), which makes sense in particular when a hardware implementation is envisaged. Simulations carried out with both these types of objective functions produce very similar solutions, resulting a substantially statistically equivalent from this point of view.

In the evolutionary process, two fitness values are calculated for each individual: the fitness f, used by the selection operator, and a test fitness $\hat{f}$. Following the commonly accepted practice of machine learning, the problem data are partitioned into three sets:

- *training set*, used to train the network;
- *test set*, used to decide when to stop the training and avoid overfitting;
- *validation set*, used to test the generalization capabilities of a network.

There is no agreement in the literature on the way these sets are named: some work consider the definitions of test and validation sets exchanged.

The fitness $\hat{f}$ is calculated according to the equation of the objective function considered, by using the MSE over the test set, or the correlation coefficient. When BP is used, i.e., if $bp = 1$, then $f = \hat{f}$, which corresponds to considering only the fitness calculated on the test set, since the network has learned the train set in the learning process. In the opposite case, when ($bp = 0$), the fitness f is calculated according to its equation, by using the MSE (or correlation coefficient) over both the training and test sets.

5.5 Selection

In the evolution process two important and closely related issues are population diversity and selective pressure. Indeed, an increase in the selective pressure decreases the diversity of the population, and vice versa. Like indicated by Michalewicz [77], it is important to strike a balance between these two factors, and sampling mechanisms are attempted to achieve this goal. As observed in that work, many of the parameters used in the genetic search affect these factors. In this sense as selective pressure is increased, the search focuses on the top individuals in the population, causing a loss of diversity population. Using larger population, or reducing the selective pressure, increases exploration, since more genotypes are involved in the search.

In the work by De Jong [61], several variations ot the simple selection method were considered; the first variation, named *elitist model*, enforces the genetic algorithm by preserving the best chromosome during the evolution. An important result, by G. Rudolph [102], is that elitism is a necessary condition for convergence of an evolutionary algorithm; of course, convergence is only probabilistic, and there is no guarantee that just one run of an evolutionary algorithm for a given number of generations will yield the globally optimal solution.

The *selection* method implemented in this work is based on breeder genetic algorithm approach [84], that differs from natural probabilistic selection since the evolution of a population considers only the individuals that better adapt themselves to the environment. Elitism is considered also in this work, allowing the best individual to survive unchanged in the next generation and solutions to monotonically get better over time.

The selection strategy implemented in this genetic algorithm is *truncation*. This kind of selection is not a novel solution in this thesis, indeed, several studies in the literature consider evolutionary approaches described the truncation selection, as in [25]. In this, and perhaps in other works, truncation is chosen in order to prevent the population from remaining too static and perhaps not evolving at all. Moreover, this kind of selection is a very simple technique and produces satisfactory solutions through conjunction with other strategies, like elitism.

All the individual in a population are initially ranked in ascending order of fitness, since the neuro-genetic approach considers a minimization problem. Each solution is assigned to an element of a ranked vector. Starting from a population of n individuals, the worse $\lfloor n/2 \rfloor$ (with respect to f) are eliminated. The remaining individuals are duplicated in order to replace those eliminated. Finally, the population is randomly permuted. The population created as a result of the selection operator will become the population of the parents for the new population of the next generation.

In each new generation, a new population has to be created. The first operator implemented is selection. The first half of the new population corresponds to the best parents that have been selected with the truncation operator, while the second part of the new population is defined by creating offspring from the previously selected parents.

5.6 Mutation

The main function of this operator is to introduce new genetic materials and to maintain diversity in the population. As indicated in Chapter 2, in some evolutionary algorithms, like genetic algorithms, mutation is generally considered as a main operator.

Generally, the purpose of mutation is to simulate the effect of transcription errors that can occur with a very low probability p_{mut}, the mutation rate, when a chromosome is duplicated.

In the neuro-genetic approach, two types of *mutation* operators are used: a general random perturbation of weights, applied before the BP learning rule, and four mutation operators which affect the network architecture. All these kinds of mutation are defined with different and independent probability parameters, considered as algorithm parameters. They could be set with pre-defined constant values, but in most cases, they are, together with other probability parameters like for crossover operator, problem dependent. For this reason, in each application described in this thesis, several settings have been tested in the experiments, in order to define the better set of related algorithm parameters.

In all real-world problems considered, and obviously in all benchmark problems during the validation, mutation probabilities have been defined with low values, defined between 0 and 1, in order to limit the disruptive effects of mutation.

In this work, the weight mutation is applied first, based on the concept of evolutionary strategies, while topology mutations are applied only after a weight control operator. This operator is always carried out in the genetic algorithm after weight mutation, because a perturbation of weight values changes the behavior of the network with respect to the activation functions; in this case, all neurons whose contribution become negligible with respect to the overall behavior, are deleted from the structure.

All these kinds of network mutations are applied to the neural networks in the evolutionary cycle, and they are implemented in order to provide a high correlation between parents and offspring in the population, and to reduce the negative effects that can occur in mutation operator.

5.6.1 Weight Mutation

This kind of mutation defines a Gaussian distribution for the Variance matrix values of each network weight. This solution wants to be similar to the approach implemented by Schwefel [106], who defined evolution strategies, described in detail in Chapter 2, algorithms in which the strategy parameters are proposed for self-adapting the mutation concurrently with the evolutionary search. The main idea behind these strategies is to allow a control parameter, like mutation variance, to self-adapt rather than changing their values by some deterministic algorithm.

Evolution strategies perform very well in numerical domains, since they are dedicated to (real) function optimization problems. They are defined by Michalewicz [77] as examples of evolution programs which use appropriate data structures, corresponding to floating value vectors extended by control strategy parameters.

All the weight matrices $\mathbf{W}^{(i)}$, $i = 0, \ldots, l$ and the biases are perturbed by using variance matrices and evolution strategies applied to the number of synapses of the entire neural network N_{syn}. This mutation is implemented by the following equation:

$$W_j^{(i)} \leftarrow W_j^{(i)} + N(0,1) \cdot \mathrm{Var}_j^{(i)} \tag{5.6}$$

$$\mathrm{Var}_j^{(i)} \leftarrow \mathrm{Var}_j^{(i)} \cdot e^{\tau' N(0,1) + \tau N(0,1)} \tag{5.7}$$

with

$$\tau' = \frac{1}{\sqrt{2N_{syn}}} \tag{5.8}$$

$$\tau = \frac{1}{\sqrt{2\sqrt{N_{syn}}}} \tag{5.9}$$

As Schwefel has defined for his first procedure [106], this kind of mutation offers a simplified method for self-adapting each single value of the Variance matrix $Var_j^{(i)}$, whose values are defined as log-normal perturbations of their parent parameter values. The weight perturbation implemented in this neuro-genetic approach allows network weights to change in a simple manner, by using evolution strategies.

Weight Control and Neuron Elimination

After this perturbation has been applied, neurons whose contribution to the network output is negligible are eliminated, based on a threshold. The neuron elimination is theoretically included into the topology mutation, and also is the consequence generated by the weight control process when a node in the network has a negligible contribution. For this reason the neuron elimination is not applied with an independent probability, as for the other kinds of topology mutations, that are described in the next section. An example of neuron elimination is shown in Figure 5.4. When a neuron is deleted, all weight connections both to the previous and to the next layer are deleted, and the corresponding matrices are updated, as well as the biases vector of the considered layer. The node and the corresponding activation function block are deleted from the network structure.

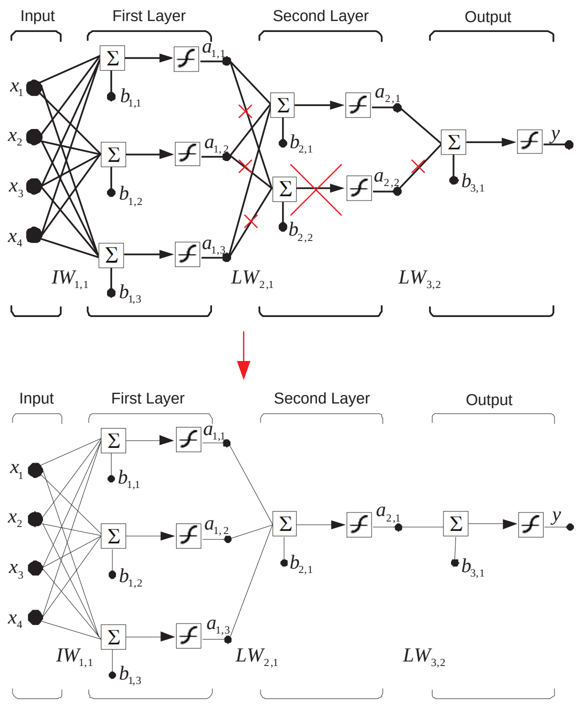


Figure 5.4: Neuron elimination

In the neuro-genetic approach two different kinds of thresholds are considered and alternatively applied to weight perturbation.

- *Fixed threshold* The first is a fixed threshold, simply defining a ϵ parameter, set before execution. The following pseudocode is implemented in mutation operator by applying a comparison between that parameter and all weight matrices values.

  ```
  for $i = 1$ to $l - 1$ do
  if $N_i > 1$
    for $j = 1$ to $N_i$ do
      if $||W_j^{(i)}|| < \epsilon$
        delete the $j^{th}$ neuron
  ```

  where N_i is the number of neurons in the ith layer, and $W_j^{(i)}$ is the j^{th} column of matrix $\mathbf{W}^{(i)}$.

  This solution presents the drawback that the fixed threshold value ϵ could be difficult to set for different real-world application. A solution to this problem has been implemented in this approach by defining a variable threshold, as following described.

- *Variable threshold* In this case the new threshold is defined, depending on a norm (in this case L_∞) of the weight vector for each node, as well as a relevant average and standard deviation of the norms of the considered layer. This task is carried out according to the following pseudo-code:

  ```
  for $i = 1$ to $l - 1$ do
  if $N_i > 1$
    for $j = 1$ to $N_i$ do
      if $||W_j^{(i)}|| < (\text{avg}_k(||W_k^{(i)}||) - r \cdot \sigma_k(||W_k^{(i)}||))$
        delete the $j^{th}$ neuron
  ```

  where N_i is the number of neurons in the ith layer, $W_j^{(i)}$ is the j^{th} column of matrix $\mathbf{W}^{(i)}$, and r is a parameter which allows the user to tune how many standard deviations below the layer average the contribution of a neuron must be before it is deleted. In this solution the settings of r parameter is only for tuning standard deviation and corresponding variances are not so invasive in mutation.

5.6.2 Topology mutation

This operator affects the network structure, i.e., the number of neurons in each layer and the number of hidden layers. In particular, the three mutations consider the insertion of a new hidden neuron or a new hidden layer, and the elimination of a hidden layer. The fourth kind of topology mutation, that regards the elimination of a neuron, is not considered with an independent probability in order to avoid disruptive effects, but is carried out as a consequence of the weight control process, as previously described in Section 5.6.1.

The three topology mutations are defined with the following steps:

1. *Insertion of one hidden layer*: with probability p^{+}_{layer}, a hidden layer i is randomly selected and a new hidden layer $i+1$ with the same number of neurons is inserted after it, with $\mathbf{W}^{(i+1)} = \mathbf{I}(N_i)$ and $b_{i+1,j} = b_{ij}$, with $j = 1, \ldots, N_i$, where $\mathbf{I}(N_i)$ is the $N_i \times N_i$ identity matrix. Figure 5.5 shows an example of layer insertion.

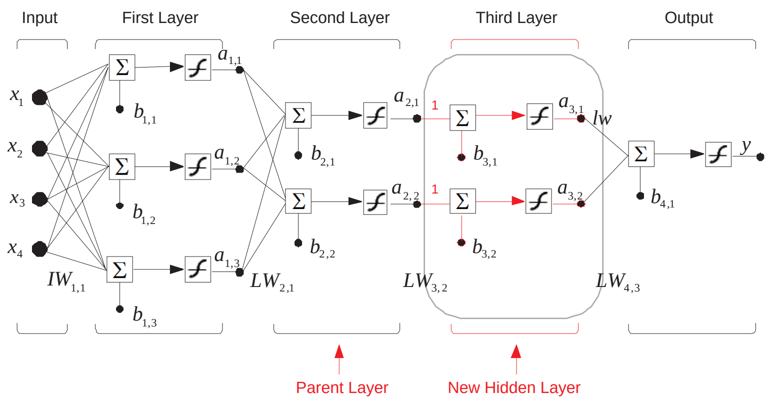


Figure 5.5: Hidden layer insertion

An important aspect underlined in the neuro-genetic approach also regards the weight connections of the new network. Indeed, in the insertion of a new hidden layer, all the nodes of the parent layer are connected to all the nodes of the offspring by defining an Identity matrix. The previous weight connections of the parent are then shifted to the output connections of the offspring layer. These operations allow to maintain such a mutation as much neutral as possible.

2. *Deletion of one hidden layer*: the deletion of a whole hidden layer is carried out with probability p^{-}_{layer}. A hidden layer i is randomly selected, but the conditions that the ith layer must have exactly one neuron and that the neural network must have at least two hidden layers have to be verified before deleting that layer. Then, the layer i is removed from the structure and the connections between the $(i-1)^{th}$ layer and the $(i+1)^{th}$ layer (to become the i^{th} layer) are rewired as follows:

$$\mathbf{W}'^{(i-1)} \leftarrow \mathbf{W}'^{(i-1)} \cdot \mathbf{W}'^{(i)}.$$

The weight connections between the $(i-1)^{th}$ layer and the $(i+1)^{th}$ layer are not randomly initialized, but they are redefined by combining their values in order to reflect the influence of the hidden layer that is removed from the neural network.

Since $\mathbf{W}^{(i-1)}$ is a row vector and $\mathbf{W}^{(i)}$ is a column vector, the result of the product of their transposes is a $N_{i+1} \times N_{i-1}$ matrix. An example of layer elimination is depicted in Figure 5.6.

3. *Insertion of one neuron*: with probability p^{+}_{neuron}, the j^{th} neuron in the hidden layer i is randomly selected for duplication. A copy of it is inserted into the same layer i as the $(N_i+1)^{th}$ neuron; the weight matrices are then updated as follows:

   (a) a new row is appended to $\mathbf{W}^{(i-1)}$, which is a copy of jth row of $\mathbf{W}^{(i-1)}$;

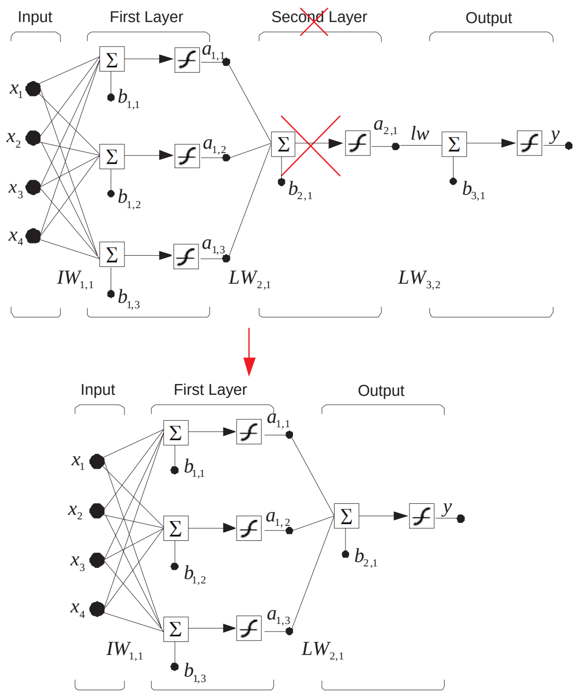


Figure 5.6: Hidden layer elimination

(b) a new column $W^{(i)}_{N_i+1}$ is appended to $\mathbf{W}^{(i)}$, where

$$W^{(i)}_j \leftarrow \frac{1}{2} W^{(i)}_j,$$
$$W^{(i)}_{N_i+1} \leftarrow W^{(i)}_j.$$

A graphical representation of this mutation is depicted in Figure 5.7. In this mutation the high correlation between parents and offspring is achieved, first, by duplicating the input connection weights of the parent in order to define the connection weights of the offspring. Then, the output weights of the parent are equally divided and the half is assigned to the offspring. The rationale for halving the output weights from both the j^{th} neuron and its copy is that, by doing so, the overall network behavior remains unchanged, and this kind of mutation is neutral.

All three topology mutation operators are aimed at minimizing their impact on the behavior of the network; in other words, they are designed to be as little disruptive, and as much neutral, as possible, preserving the behavioral link between the parent and the offspring better than by adding random nodes or layers.

5.7 Recombination

It is well known that recombination of neural networks arbitrary structure is a very hard issue, due to the detrimental effect of the permutation problem [137]. No satisfactory solutions have been proposed so far in the literature. As a matter of facts, the most successful

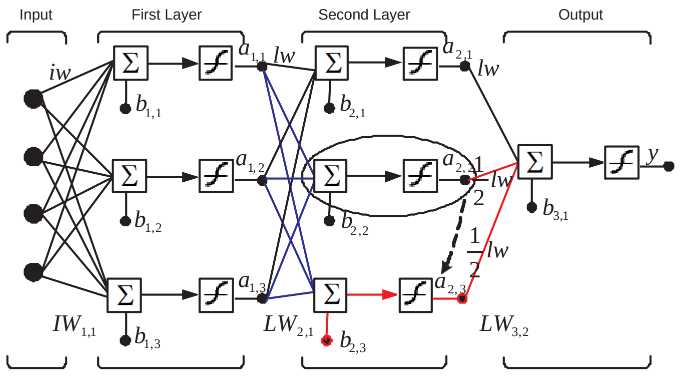


Figure 5.7: Neuron Insertion

approaches to neural network evolution do not use recombination at all [137]. Recombination contributes to population variability. For this reason, as also indicated in [75], there has been some debate in the literature about the opportunity of applying crossover to ANN evolution, based on the disruptive effects that could have on the neural model.

In this neuro-genetic approach two kinds of crossover are independently studied and implemented, in order to improve the behaviour between parents and offspring in an evolving population. In particular, two kinds of crossover are defined. The first is a kind of single-point crossover with different cutting points for each of the two parents; the second implements a kind of vertical crossover, defining a *merge-operator* between the topologies and weight matrices of two parents in order to create the offspring. The proportion of parents undergoing crossover during a generation is controlled, in the neuro-genetic approach, by the crossover rate p_{cross}, a parameter of the genetic algorithm, which has a value defined in [0,1], and determines how frequently the crossover operator is invoked.

The results obtained from the applications of the two crossover operators appear promising, even though they do not boost the performance of the neuro genetic approach significantly in the present form. Further studies of new crossover design could improve it, by being as little disruptive as possible.

5.7.1 Single-Point Crossover

This operator represents a kind of single-point crossover, where cutting points are independently extracted for each parent, because in this neuro-genetic approach, the genotype length of each individual is variable, and not bounded. The only ‘restriction’ implemented in this method is that genotypes can be cut only in ‘meaningful’ places, i.e., only between one layer and the next, avoiding so-called ‘diagonal’ cutting points. This means that a new weight matrix has to be created to connect the two layers at the crossover point in the offspring. These new weight matrices are initialized from a normal distribution, while corresponding variance matrices are set to matrices of all ones.

In this crossover operator, the individuals can be seen as vectors, and for two parents vectors a and b, that can have different dimensions, a single-point crossover is performed by selecting two different cutting points, "Cutting point a" and "Cutting point b", and exchanging the elements occurring after cutting point "a" in parent "A", with those that occur after point ‘b’ in the corresponding parent "B". This kind of crossover is illustrated in

Figure 5.8.

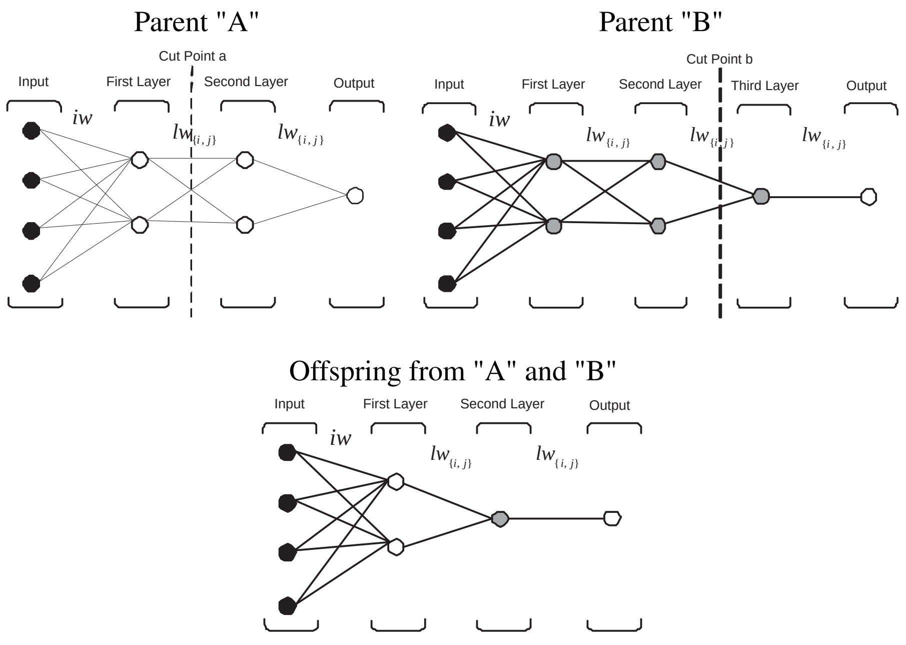


Figure 5.8: Single-Point Crossover Representation.

5.7.2 Vertical Crossover

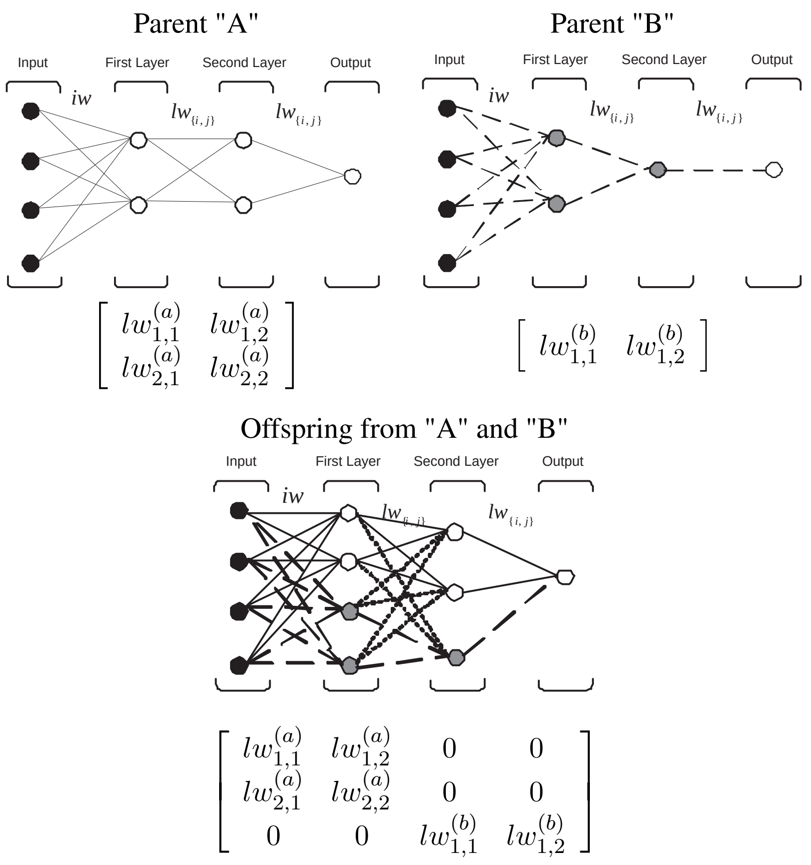
The second type of crossover operator is a kind of "vertical" crossover and is implemented as shown in Figure 5.9.

Once the new population has been created by the *selection* operator previously described in this chapter, two individual are chosen for coupling and their neural structures are compared. If there are some differences in the topology length l, the hidden layer insertion mutation operator will be applied to the shortest neural topology in order to obtain individuals with the same number of layers.

Then a new individual will be created, the child of the two parents selected. The neural structure of the new individual is defined by adding the number of neurons in each corresponding hidden layer of each parent, except for the input and output layers (they are the same for each neural network).

The new input-weights matrix $\mathbf{W}^{(0)}$ and the corresponding variance matrix $\mathbf{Var}^{(0)}$ are respectively obtained by appending the matrix of the second parent to the matrix of the first parent. Then, the new weight matrix $\mathbf{W}^{(i)}$ and the corresponding variance matrix $\mathbf{Var}^{(i)}$ for each hidden layer of the new individual are respectively defined as the block diagonal matrix made up of the matrix of the first parent and the matrix of the second parent. Bias values and corresponding variance matrices of two parents are concatenated in order to obtain the new values for the new biases b_{ij} and variances $\mathbf{Var}(b_{ij})$.

All the weights associated the inputs to the new output layer will be set to be the half of the corresponding weights in the parents. The rationale of this choice is that, if both parents were ‘good’ networks, they would both supply the appropriate input to the output layer; without halving it, the contribution from the two subnetworks would add and yield



$$\begin{bmatrix} lw^{(a)}_{1,1} & lw^{(a)}_{1,2} \\ lw^{(a)}_{2,1} & lw^{(a)}_{2,2} \end{bmatrix}$$

$$\begin{bmatrix} lw^{(b)}_{1,1} & lw^{(b)}_{1,2} \end{bmatrix}$$

$$\begin{bmatrix} lw^{(a)}_{1,1} & lw^{(a)}_{1,2} & 0 & 0 \\ lw^{(a)}_{2,1} & lw^{(a)}_{2,2} & 0 & 0 \\ 0 & 0 & lw^{(b)}_{1,1} & lw^{(b)}_{1,2} \end{bmatrix}$$

Figure 5.9: Merge-Crossover Representation.

an approximately double input to the output layer. Therefore, halving the weights helps to make the operator as little disruptive as possible.

5.8 Conclusion

The neuro-genetic approach implemented in this thesis performs the simultaneous evolution of topology and weight connections of a population of neural networks, combining the advantages of these two techniques. Furthermore, the backpropagation algorithm can be used to improve the learning of each evolutionary neural network. The direct encoding of each individual, i.e. neural network, in the population allows each of them to have all the information available in the encoding. With such a network encoding, few constant values related to the structure definition of each neural network have to be set at initialization, and a few algorithm parameters are set at first time, while the genetic parameters are tuned during the experiments in order to find the best settings, even if all the obtained results show how the neuro-genetic approach is robust with respect to that parameter setting.

The behavioral link between parents and offspring during the evolution is allowed by the genetic operators implemented in the approach.

Different kinds of mutation operators are defined, considering the mutation of the weights and the biases of each neural network and the mutation of the architecture, with independent probabilities. All the experiments, carried out in order to validate the neuro-genetic approach, show the satisfactory performances resulting from the simultaneous application of these mutation operators.

The neuro-genetic approach also considers the recombination operator, even if it is known that is a hard issue, due to the disruptive effects that can occur during different network crossover. Nevertheless, two kinds of possible crossover operators are presented and a study about implementation and result is also carried out.

Chapter 6

Validation on Benchmark Problems

6.1 Research Applications

The neuro-genetic approach presented in Chapter 5 has been applied to three different real-world problems, and, in order to validate this approach, also two benchmark problems are considered and described in this chapter.

Chapter 7 describes an industrial application for neural engine controller design.

The second application presented in Chapter 8 concerns a neural classification algorithm for brain wave signal processing, considering in particular the analysis of P300 Evoked Potential.

Finally, the third application presented in Chapter 9, considers financial modeling, whereby a factor model capturing the mutual relationships among several financial instruments is sought for.

Generally, in each real-world problem, the availability and the integrity of the data is one of the most important aspects that has to be considered in each approach that attempts to solve it. Data analysis in neural network training is a crucial factor, because, in that process, the data should fully represent all possible states of the problem being tackled, and furthermore, they should be sufficient and valid for the defined problem. A discussion about such aspect of neural network training is described in Chapter 3.

Data have to be usually specified in an operational range of the network, defined for each problem considered. For example, the target data for a backpropagation network with sigmoid activation functions needs to lie between 0 and 1, because that is the range defined for the sigmoid function, and in some cases, different individual features might need to be scaled differently.

In each of the three applications and in the validation problems, all data are analyzed before applying evolutionary approach, and, since the network training process considers the tangent sigmoid transfer function to calculate all layer outputs, all the target data will have to be specified in the range of definition of that function. For this reason, a preprocessing of the data is carried out when necessary, so that their values will fall in the interval of the defined transfer function, that corresponds to [-1,1].

The following sections present the validation of the approach on a non-linear regression problem and on two benchmark problems.

6.2 Synthetic non linear Validation

As a preliminary validation of the approach, a non-linear regression problem has been set up, in which a synthetic function is defined as a combination of 32 input values. Each input randomly generated, belongs to the interval $[-1, +1]$. The synthetic output equation is defined as:

$$y = \frac{1}{32}\sum_{i=1}^{32} \sin\left(\frac{2\pi x_i}{\nu_i} + \phi_i\right) \tag{6.1}$$

where x_i is the input, ν_i and ϕ_i are, respectively, the frequency and the phase vectors related to x_i. Functions of this general form combine different parameters in a highly non-linear fashion and are usually difficult to predict. For simplicity, in this case ν_i is set to i for each input, while ϕ_i is kept equal to zero, giving the equation:

$$y = \frac{1}{32}\sum_{i=1}^{32} \sin\left(\frac{2\pi x_i}{i}\right) \tag{6.2}$$

Three datasets are created with 1000 cases for training, 250 cases for test and 100 cases for validation, respectively.

Problem parameters are set to their default values, while several simulations have been carried out with different mutation parameter settings in order to identify the combination that returns the best solution. For each setting, 10 runs have been carried out and average and standard deviation of fitness values obtained on the test set are reported in Table 6.1.

Table 6.1: Experimental results of non-linear synthetic function.

Setting	Parameter Setting			BP=1	
	p^{+}_{layer}	p^{-}_{layer}	p^{+}_{neuron}	avg	stdev
1	0.05	0.05	0.05	0.1703	0.0031
2	0.05	0.05	0.1	0.1710	0.0039
3	0.05	0.05	0.2	0.1696	0.0049
4	0.05	0.1	0.05	0.1680	0.0023
5	0.05	0.1	0.1	0.1744	0.0064
6	0.05	0.1	0.2	0.1687	0.0030
7	0.05	0.2	0.05	0.1691	0.0057
8	0.05	0.2	0.1	0.1703	0.0043
9	0.05	0.2	0.2	0.1675	0.0068
10	0.1	0.05	0.05	0.1689	0.0035
11	0.1	0.05	0.1	0.1689	0.0015
12	0.1	0.05	0.2	0.1711	0.0018
13	0.1	0.1	0.05	0.1674	0.0070
14	0.1	0.1	0.1	0.1746	0.0150
15	0.1	0.1	0.2	0.1681	0.0024
16	0.1	0.2	0.05	0.1689	0.0028
17	0.1	0.2	0.1	0.1692	0.0031
18	0.1	0.2	0.2	0.1704	0.0068
19	0.2	0.05	0.05	0.1693	0.0018
20	0.2	0.05	0.1	0.1687	0.0022
21	0.2	0.05	0.2	0.1706	0.0024
22	0.2	0.1	0.05	0.1691	0.0039
23	0.2	0.1	0.1	0.1711	0.0064
24	0.2	0.1	0.2	0.1703	0.0030
25	0.2	0.2	0.05	0.1687	0.0021
26	0.2	0.2	0.1	0.1675	0.0026
27	0.2	0.2	0.2	0.1701	0.0037

The best model has been found by the algorithm taking advantage of backpropagation, and is a multi-layer perceptron with a phenotype of type [2, 1], which obtained a fitness on the test set of 0.1530 and a mean square error of 0.2791 on the same set. The agreement between the output of the best model with the output of the synthetic non-linear function implemented is shown in Figure 6.1. These two output shapes are obtained on the validation set and they have a high correlation coefficient equal to 0.9183.

In order to validate the quality of the model, a comparison with a linear regression model of the same data has been performed. The linear regression yields a linear model represented by Equation 6.3:

$$y = \sum_{i=1}^{32} w_i x_i, \tag{6.3}$$

The neuro-genetic solution obtained with the neuro-genetic approach has a MSE of $6.7642 * 10^{-4}$, better than $7.3724 * 10^{-4}$, MSE of the simulation based on the linear regression on the same validation set.

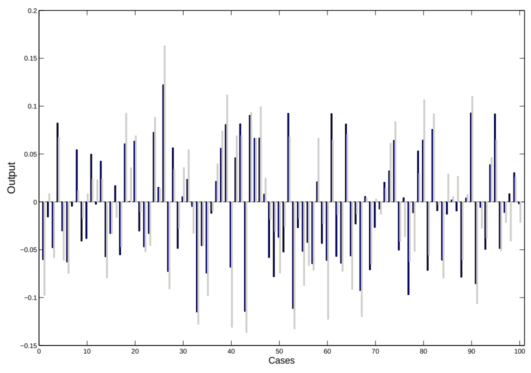


Figure 6.1: Comparison between the output of the best model (black bars) with the desired values (white bars) of the validation set.

6.3 Benchmark Problems

The application to benchmark problems considered two well-known historical problems, defined in the literature, that have been used also for several comparison works. The first problem approached is the Pima Indians Diabetes problem, while the second is the Breast Cancer Wisconsin problem; they are both classification problems.

All data included in the training, validation and test sets are acquired from the UCI Machine Learning Repository [85]. It is a repository of databases and data generators that are used by the machine learning community for the empirical analysis of machine learning algorithms. In this repository, different and several benchmark problems are collected and all the corresponding data can be fully downloaded.

6.3.1 Pima Indian Diabetes Problem

The Pima Indians Diabetes database data refers to a medical problem, in which the diagnosis is carried out on several patients, in order to investigate whether a patient shows signs of diabetes according to World Health Organization criteria (i.e., if the 2 hour post-load plasma glucose is at least 200 mg/dl at any survey examination or if it has been found during routine medical care).

The dataset available for this problem has been uploaded into the UCI repository in 1990, and includes 768 instances, composed of 8 attributes plus a binary class value, which corresponds to the target classification value. A value equal to 1 for this attribute means that the patient tested positive for diabetes, while a 0 value means that the test was negative for that disease. All input and output features are summarized in Table 6.2.

Table 6.2: Set of features considered for PIMA Indian Diabetes problem.

Number	Attribute
1	Number of times pregnant
2	Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3	Diastolic blood pressure (mm Hg)
4	Triceps skin fold thickness (mm)
5	2-Hour serum insulin (mu U/ml)
6	Body mass index, with weight expressed in kg and height expressed in m (kg/m^2)
7	Diabetes pedigree function
8	Age (years)
9	Class variable (0 or 1)

A past test on these data is the one by Smith and colleagues [112]. The authors tested the ability of an early neural network model, ADAP, to forecast the onset of diabetes mellitus in a high risk population of Pima Indians. ADAP is an adaptive learning routine that generates and executes digital analogs of perceptron-like devices. Their ADAP algorithm developed a real-valued prediction between 0 and 1. This was transformed into a binary decision using a threshold of 0.448. As previously indicated, all data necessary to create the data sets were acquired from the UCI Machine Learning Repository. The ADAP algorithm used 576 instances for the training set, while the remaining 192 instances were used for the validation set (according to the terminology used in this thesis). After running the learning algorithm, accuracy of their algorithm was 76% on the validation set. A relevant information concerning their approach was that several constraints have been placed on the selection of these instances from a larger database. In particular, all patients here were females at least 21 years old of Pima Indian heritage.

Classification with neuro-genetic approach

The validation of the neuro-genetic approach is carried out considering all the data available from UCI repository. The first 500 instances were used to create the training set, the following 134 to create the test, and, finally, the remaining 134 were used for the validation set. As previously reported in Chapter 5, concerning the fitness function implementation, it is important to point out that there is no agreement in the literature on the way these sets are named, as the naming of validation and test set are often exchanged. In this thesis, the test set is used to decide when to stop training, avoiding overfitting, while the validation set is used to test the generalization capabilities of a neural network. This terminology is also adopted in this thesis to compare the neuro-genetic approach with the other approaches.

The neuro-genetic approach is implemented by choosing backpropagation as training algorithm, and the evolutionary approach is carried out with all parameters set to the default values shown in Table 5.2. Several settings of the three mutation probabilities relevant to structural mutation, p^+_{layer}, p^-_{layer} and p^+_{neuron}, have been explored in order to assess the robustness of the approach and to determine the optimal solution. Ten runs are executed for each setting, and the average and the standard deviation values of the test fitness of the best individual found are summarized in Table 6.3.

Table 6.3: Experimental results for the Pima Indian Diabetes problem.

setting	p^+_{layer}	p^-_{layer}	p^+_{neuron}	Avg	Std Dev
1	0.05	0.05	0.05	0.72861	0.0233
2	0.05	0.05	0.1	0.7365	0.0172
3	0.05	0.05	0.2	0.7374	0.0140
4	0.05	0.1	0.05	0.7264	0.0261
5	0.05	0.1	0.1	0.7410	0.0167
6	0.05	0.1	0.2	0.7281	0.0231
7	0.05	0.2	0.05	0.7377	0.0229
8	0.05	0.2	0.1	0.7308	0.0189
9	0.05	0.2	0.2	0.7300	0.0252
10	0.1	0.05	0.05	0.7383	0.0251
11	0.1	0.05	0.1	0.7267	0.0311
12	0.1	0.05	0.2	0.7331	0.0234
13	0.1	0.1	0.05	0.7416	0.0276
14	0.1	0.1	0.1	0.7378	0.0280
15	0.1	0.1	0.2	0.7345	0.0148
16	0.1	0.2	0.05	0.7339	0.0130
17	0.1	0.2	0.1	0.7305	0.0254
18	0.1	0.2	0.2	0.7229	0.0187
19	0.2	0.05	0.05	0.7205	0.0202
20	0.2	0.05	0.1	0.7395	0.0177
21	0.2	0.05	0.2	0.7312	0.0220
22	0.2	0.1	0.05	0.7283	0.0150
23	0.2	0.1	0.1	0.72486	0.0275
24	0.2	0.1	0.2	0.7196	0.0306
25	0.2	0.2	0.05	0.7342	0.0129
26	0.2	0.2	0.1	0.7366	0.0257
27	0.2	0.2	0.2	0.7359	0.0163

The best solution, on average, is found with $p^+_{layer} = 0.2$,$p^-_{layer} = 0.1$ and $p^+_{neuron} = 0.2$. The best solution is a MLP neural network with a phenotype of $[2, 2, 1]$, which obtained an error percentage on the validation set equal to 24.5%, with a sensitivity on the validation set equal to 75.5%. The summary of the comparison between the ADAP algorithm and the neuro-genetic approach is carried out in Table 6.4.

Table 6.4: Comparison between accuracy of neuro-genetic approach and ADAP algorithm.

Problem	Experimental Results	
PIMA Indian Diabet	ADAP Algorithm	Neuro-Genetic Approach
Instances of		
Training set	576	500
Test set	–	134
Validation set	192	134
Accuracy on Validation set	76%	75.5%

Comparison to the Literature

In order to better validate the approach proposed in this P.h.D. dissertation, a comparison with a list of some, more recent works presented in the literature, is shown in Table 6.5. All these works have used the Pima Indian Diabetes problem as a benchmark. The results show that the neuro genetic approach compares well with respect to the other works, obtaining good consistency and accuracy.

Table 6.5: Comparison of classification performance on the Pima Indian Diabetes problem.

Author (year)	Method	Accuracy (%)
G. Arulampalam et al (2001)	SIANN LM	78.7
	SIANN GDA	73.9
	MLP LM	78.9
	MLP GDA	78.33
H.A. Abbass (2003)	Multiobjective EA:	
	MPANN	74.9
	SPANN	70.7
J. Basak (2006)	ExOADT-K-NN	77.49
	K-NN	70.44
	Naive Bayes	75.78
	SVM	73.30
G.L. Tsirogiannis et al (2004)	AdaBoost with Decision Trees	74.5
	Fuzzy with Decision Trees	72.4
X. Yao et al (1997)	EPNet	77.6
X. Yao et al (2003)	CNNE	77.8
N. García Pedrajas et al	COVNET	80.1
A. Azzini et al (2006)	Neuro genetic approach	75.5

Briefly, the basic idea implemented in each work reported in Table 6.5 are presented here. G.Arulampalam and colleague developed a shunting inhibitory artificial neural network model [5], in which neurons interact between each other through a non linear mechanism called shunting inhibition, allowing the neurons to operate as adaptive nonlinear filters. The performances of the SIANN topologies were compared in that work with those obtained by MLP neural networks, and some results are reported in Table 6.5.

H.A. Abbass presented an optimization algorithm [1], comprising a multiobjective evolutionary algorithm and a gradient-based local search as classification approach. This was referred to as the Memetic Pareto Artificial Neural Network algorithm (MPANN), with a performance on Pima Indian Diabetes problem equal to 74.9%. He also presented a self-adaptive version, called SPANN, to reduce the time for parameter tuning, that obtained an accuracy of 70.7%.

J. Basak asserted in his work [14] that the decision trees can be trained in the online adaptive (OADT) mode. In this work an architecture based on OADT, ExOADT, was described, which can handle multiclass classification tasks, able to perform function approximation. Some of the results showed that ExOADT was structurally similar to OADT extended with a regression layer. ExOADT with K-NN obtained an accuracy equal to 77.49%, while for K-NN it was equal to 70.44%, for naive bayes equal to 75.78% and, for SVM to 73.30%.

G.L. Tsirogiannis and colleagues proposed a new meta-classifier approach [118], which combined several different combination methods, in analogy to the combination of simple classifiers. Among the meta classifiers considered, AdaBoost with decision trees had an accuracy equal to 74.5%, while the fuzzy classifier with decision trees, had an accuracy equal to 72.4%.

X. Yao and colleagues presented in their most successful approaches to neural network evolution also the results obtained, with the evolutionary algorithm presented in [137] and with the constructive algorithm presented in [58]. The first algorithm was based on Fogel's evolutionary programming (EP), in order to evolve ANN architecture and connection weights and biases simultaneously. In this approach the accuracy was equal to 77.6%. The second work by Yao and other colleagues presented a more sophisticated idea, based on constructive algorithm for training cooperative neural network ensembles, with an accuracy of 77.8%.

N. García Pedrajas and colleagues defined a cooperative coevolutionary model for evolving artificial neural networks [89]. Their model was based on the idea of co-evolving subnetworks that must cooperate to form a solution, instead of evolving complete networks. They showed in their work how this idea can achieve very high satisfactory results on this classification problem, with very high accuracy, equal to 80%.

The experiments carried out with the neuro-genetic approach implemented in this thesis confirm that the results obtained are comparable with those obtained from other simulations carried out in the literature on the same data of this problem, giving a satisfactory assessment of the neuro genetic approach on this benchmark problem. Although in the present state of research the results appear worse than those obtained by García Pedrajas and colleagues in [89], the neuro-genetic work is in progress and further studies of co-evolutionary approaches may be considered in future.

Figure 6.2 shows the sensitivity and the specificity shapes obtained with the experiments carried out with the neuro-genetic approach on the validation set, which correspond to the true positive and to 1 - false positive cases, respectively.

In signal detection theory, a receiver operating characteristic (ROC) [76], also receiver operating curve, is a graphical plot of the sensitivity versus (1 - specificity) for a binary classifier system as its discrimination threshold is varied. The best possible prediction method would yield a graph that is a point in the upper left corner of the ROC space,

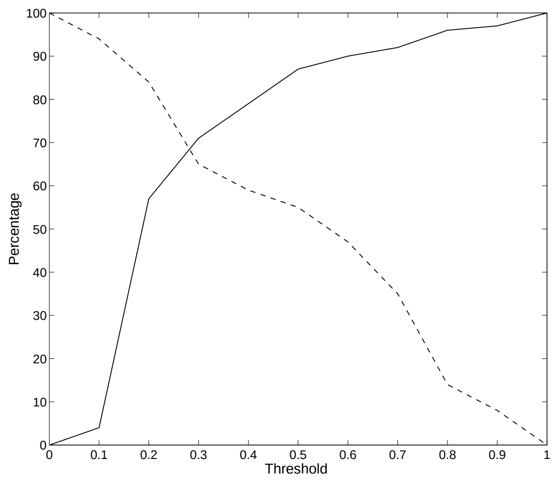


Figure 6.2: Sensitivity (dashed line) and specificity (solid line) shapes of the best model on the validation set.

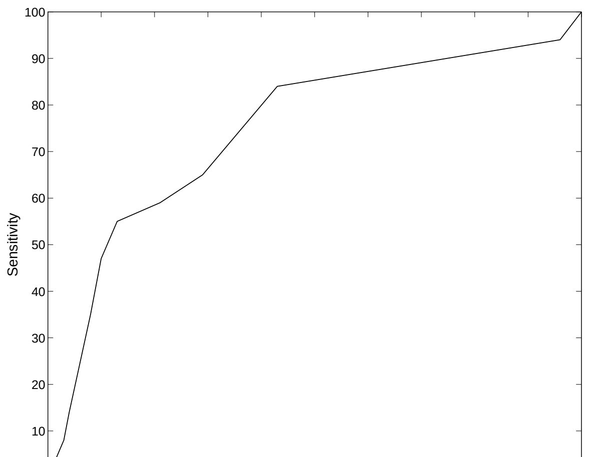


Figure 6.3: ROC Shape of the best solution on the validation set.

i.e. 100% sensitivity (all true positives are found) and 100% specificity (no false positives are found). The ROC statistic is often used in classification problems, since this measure can be interpreted as the probability that when one positive and one negative example are randomly picked, the classifier will assign a higher score to the positive example than to the negative.

Figure 6.3 shows the ROC curve defined on the sensitivity and the specificity values obtained from the classifier by varying the threshold.

6.3.2 Breast Cancer Problem

The second benchmark problem considers a breast cancer classification problem. The breast cancer database was obtained from Dr. W.H. Wolberg of the University of Wisconsin Hospitals, Madison. This problem was approached in the past in several works that considered a multisurface method of pattern separation [127], or a pattern recognition approach defined through linear programming [72, 16].

Relevant information has to be considered about the data set creation, because samples were provided by Dr. Wolberg, who periodically reported his clinical cases. Therefore, the database reflects the chronological grouping of the data, and all the approaches carried out in the literature have been implemented by considering different data set dimensions. Since 1992 a dataset of 699 instances has been completed and now is available in the UCI machine learning repository [85], even if some features have missing values.

In all data instances are represented by 10 attributes for the input values and 1 for the target values: each instance has one of two possible classes: benign or malignant. All the features considered in this problem are listed in Table 6.6.

Table 6.6: Set of Features considered for Breast Cancer Wisconsin problem.

Number	Attribute
1	Sample code number
2	Clump Thickness
3	Uniformity of Cell Size
4	Uniformity of Cell Shape
5	Marginal Adhesion
5	Single Epithelial Cell Size
6	Bare Nuclei
7	Bland Chromatin
8	Normal Nucleoli
9	Mitoses
10	Class: malignant (4) /benign (2)

The class distribution is not equally divided between the two cases, because 458 are benign, corresponding to 65.5% of the instances at this time, while 241 are malignant, corresponding to the 34.5% of the entire actual database. For this reason the creation of a balanced dataset is generally difficult to obtain.

The first work on this problem carried out by Wolberg and colleagues had a number of instances equal to 369 (at that point in time). In their work a linear classifier was constructed to separate benign from malignant samples. Classification results on an equally divided dataset, in order to create training and validation set, respectively, gave an accuracy equal to 93.5%. During the simulations, three pairs of parallel hyperplanes were found to be consistent with 67% of data, while accuracy on the remaining 33% of data was increased to 95.9%.

Zhang [140] presented another approach in the literature, in which the size of the data set was the same as in the approach previously described, 369. He applied 4 instance-based learning algorithms, collecting classification results averaged over 10 trials. The best accuracy was obtained using 1-nearest neighbor: 93.7%. The simulation was carried out on a training set of 200 instances, and tested on the other 169.

Classification with the neuro-genetic approach

Although in total there were 699 individual measurements (of the set of all parameters), 16 instances have been found with missing parameters which occurred for measurements. The neuro-genetic approach have been carried out on this problem considering all instances available today, that have no missing, unavailable, data. Therefore, all incomplete instances are removed from the set, leaving a total of 683 data vectors for network training, validation and test. All the data are divided in order to create the three sets, using 400 instances for the training, 140 for the test and 140 for the validation set.

Backpropagation is implemented for the training process, and all parameters are set to the default values. Several settings of the three mutation probabilities, p^{+}_{layer}, p^{-}_{layer} and p^{+}_{neuron}, have been explored in order find the optimal solution. Ten runs are executed for each setting, and the average and the standard deviation values of the test fitness of the best individual are reported in Table 6.7.

The best solution, on average, is found with very low probabilities, respectively with $p^{+}_{layer} = 0.05$, $p^{-}_{layer} = 0.05$ and $p^{+}_{neuron} = 0.05$. The best solution is a neural network with a phenotype equal to $[5, 1]$, which obtained a percentage of error on the validation set equal to 0.7%.

The experiments carried out with the neuro-genetic approach implemented on this problem confirm that the results obtained are satisfactory, reinforcing the validation of the approach. Table 6.8 shows the size of each dataset considered during all the experiments carried out.

The accuracy of the different models is also reported in Table 6.8.

Comparison to the Literature

As for the other clinical diagnosis problems, classifiers have been developed for breast cancer diagnosis problem, too. A great variety of methods were used, obtaining high classification accuracies. Among these, a comparison has been carried out by considering some works most recently presented in the literature. Table 6.9 shows the comparison between the neuro genetic classification performance for the breast cancer problem with classification accuracies obtained by other methods in the literature.

Briefly, the basic idea implemented in each work are also described. FNNCA [109] implemented by Setiono described a rule classifier, that obtained a percentage of accuracy on the Breast Cancer Diagnosis problem equal to 98.10%.

S.G. Pierce and colleagues defined in [90] a non-probabilistic approach to develop a classification algorithm based on MLP neural networks. The information-gap robustness

Table 6.7: Experimental results for the Wisconsin Breast Cancer problem.

setting	p^+_{layer}	p^-_{layer}	p^+_{neuron}	Avg	Std Dev
1	0.05	0.05	0.05	0.0204	0.0076
2	0.05	0.05	0.1	0.0237	0.0055
3	0.05	0.05	0.2	0.0264	0.0069
4	0.05	0.1	0.05	0.0274	0.0050
5	0.05	0.1	0.1	0.0249	0.0056
6	0.05	0.1	0.2	0.0250	0.0066
7	0.05	0.2	0.05	0.0256	0.0084
8	0.05	0.2	0.1	0.0208	0.0067
9	0.05	0.2	0.2	0.0284	0.0075
10	0.1	0.05	0.05	0.0242	0.0054
11	0.1	0.05	0.1	0.0243	0.0055
12	0.1	0.05	0.2	0.0250	0.0060
13	0.1	0.1	0.05	0.0249	0.0063
14	0.1	0.1	0.1	0.0248	0.0085
15	0.1	0.1	0.2	0.0257	0.0042
16	0.1	0.2	0.05	0.0248	0.0073
17	0.1	0.2	0.1	0.0224	0.0064
18	0.1	0.2	0.2	0.0244	0.0072
19	0.2	0.05	0.05	0.0212	0.0063
20	0.2	0.05	0.1	0.0209	0.0065
21	0.2	0.05	0.2	0.0248	0.0058
22	0.2	0.1	0.05	0.0257	0.0073
23	0.2	0.1	0.1	0.0231	0.0058
24	0.2	0.1	0.2	0.0239	0.0066
25	0.2	0.2	0.05	0.0256	0.0042
26	0.2	0.2	0.1	0.0246	0.0056
27	0.2	0.2	0.2	0.0259	0.0043

Table 6.8: Comparison between experimental results.

Problem	Experimental Results			
Wisconsin Breast Cancer	Wolberg (1)	Wolberg (2)	Zhang	Neuro Genetic
Instances of				
Training set	185	247	200	400
Test set	–	–	–	140
Validation set	184	122	169	140
accuracy on Test set	93.5%	95.9%	93.7%	99.3%

Table 6.9: Comparison of classification performance on the Wisconsin Breast Cancer problem.

Author (year)	Method	Accuracy (%)
S.G. Pierce et al (2006)	Information GAP:	
	for Maximum-Likelihood training	80.4
	for Bayesian-Evidence training	78.5
K. Polat et al (2005)	FS-AIRS	98.51
E. D. Ubeyli (2005)	Mixture of Experts (ME)	98.85
Y. Sun et al (2002)	approaches implemented:	
	NEFCLASS	89.70
	RBFs	91.24
	K-mean	97.09
	Fuzzy C-Mean	96.77
Z.H. Tan (2004)	GAEPNet	99.27
J. Basak (2006)	ExOADT K-NN	97.18
	K-NN	96.13
	Naive Bayes	93.15
	SVM	96.48
A. Abraham et al (2003)	Fuzzy rule classifiers:	
	Mean & Deviation	85.94
	Histogram	84.36
	Simple Grid	62.39
	Modified Grid	85.96
H.A. Abbass (2003)	Multiobjective EA:	
	MPANN	98.1
	SPANN	98.3
Setiono (2000)	Neuro Rule	98.10
J.J. Merelo et al (1999)	G-prop III with QP	99
X. Yao et al (1997)	EPNet	98.62
X. Yao et al (2003)	CNNE	98.90
A. Azzini et al (2006)	Neuro genetic approach	99.3

function was used in the algorithm to select the network with the highest robustness of uncertainty in the validation. The authors obtained an accuracy percentage of 80.4% on the validation set, by applying maximum-likelihood training, while 78.5% by applying the bayesian-evidence update training.

K. Polat and colleagues [91] obtained 98.51% by implementing a feature selection artificial immune recognition system algorithm: the number of the features was reduced in the feature-selection program by forming rules related to data with a decision tree algorithm. The aim of this algorithm was to eliminate useless features in determining malignancy. The performance was evaluated using 10 fold cross validation method, dividing the data set into 10 clusters each having similar values for the features.

E.D. Ubeyli obtained 98.85% accuracy with a modular neural network architecture (ME) [119]. The expectation-maximization algorithm was used for training the ME so that the learning process was decoupled in a manner that fit well with the modular structure. The algorithm considered MLP neural networks.

Y. Sun and colleagues [115] obtained different classification performances by considering different approaches. They obtained an accuracy of 89.70% with NEFCLASS, a neuro-fuzzy classification algorithm, based on a MLP structure, where the shared weights were encoded as fuzzy sets and the activation function used fuzzy operators. Different membership functions were defined, corresponding to broken-line and triangular functions. In [115], a comparison with other standard classification approaches was carried out on the same data set, showing that K-mean obtained a percentage of accuracy of 97.09%, RBFs obtained 91.24%, and fuzzy c-mean obtained 96.77%.

Z.H. Tan obtained a very high accuracy rate equal to 99.27% in [116] with a hybrid evolutionary algorithm, implemented to the structural and parametric learning of MLP with one layer neural networks.

A. Abraham and colleague presented a comparison of the performance of four fuzzy rule generation methods on the breast cancer problem. These methods were described in detail in their work in [59]. The first approach, that generated fuzzy rules using the mean and the standard deviation of the attribute values, obtained an accuracy of 85.94%, while for the second, which generated rules using the histogram of the attribute values, accuracy was 84.36%. The accuracy of the third, that used homogeneous fuzzy sets, was equal to 62.39%, while for the last, a grid based approach, the accuracy was 85.96%.

Also in the work previously reported for Pima Indian Diabetes problem of H.A. Abbass [1], who presented a multi-objective evolutionary algorithm with a gradient-based local search, the application to Wisconsin Breast Cancer problem was considered, obtaining an accuracy equal to98.1% for MPANN, and to 98.3% for SPANN.

In the work already presented in Pima Indian Diabet problem, J. Basak [14] discussed a further comparison of the accuracy of ExOADT with K-NN, equal to 97.18%, with other classification models regarding the breast cancer Wisconsin problem. With K-NN accuracy was 96.13%, while with naive bayes was 93.15%, and in SVM with gaussian kernels 96.48%.

J.J. Merelo and colleagues [25] presented an evolutionary algorithm to obtain global optimization of multi-layer perceptrons, based on a genetic algorithm and a implementing a Lamarckian training process. This approach provided highly satisfactory results in classification problems; the accuracy obtained in this problem was equal to 99%.

X. Yao and colleagues also reported in their most successful approaches to neural network evolution [137, 58], already presented for the Pima Indian Diabetes problem, their performances for Breast Cancer problem. In these approaches the accuracy was equal to 98.62% in [137], while was equal to 98.90% in [58].

The neuro-genetic approach presented in this thesis, provides an accuracy equal to 99.3%. Figure 6.4 shows the sensitivity and the specificity of the classifier with different threshold values, and Figure 6.5 shows the ROC shape, that gives a satisfactory performance in generalizing classifiers.

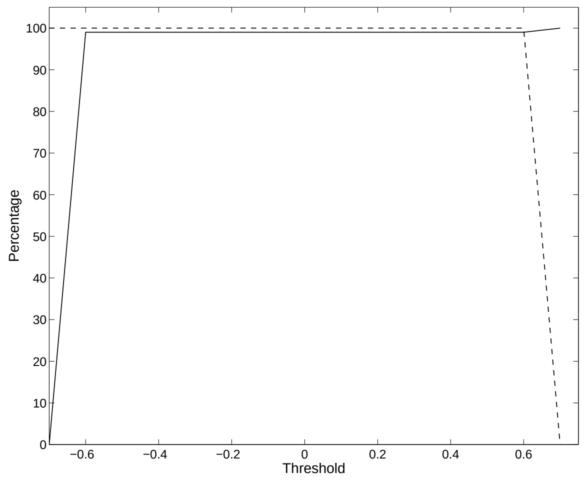


Figure 6.4: Sensitivity (dashed line) and specificity (solid line) shapes of the best model on the validation set.

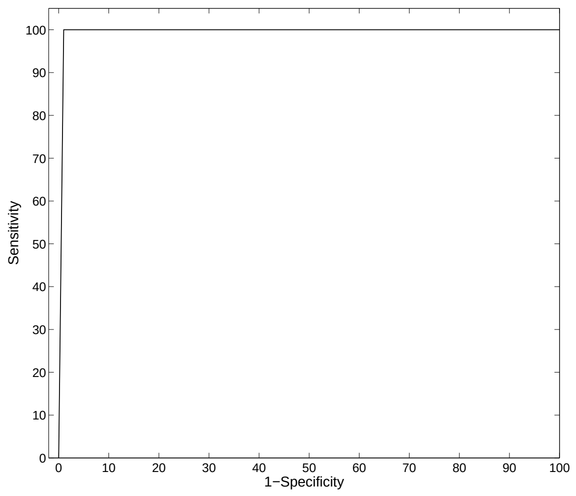


Figure 6.5: ROC Shape of the best solution on the validation set.

Chapter 7

Fault Diagnosis

7.1 Introduction

The usefulness of the approach, subject of this P.h.D. dissertation, has then been validated on real-world problems, as already mentioned in Chapter 5. The neuro-genetic approach works as a selector of possible solutions, providing the neural network design that corresponds, in an evolved population, to the best model for a given task. In this chapter, the first real-world problem considered is the design of an incipient fault detector in an electrical drives monitoring.

7.2 Application

Every industrial application requires a suitable monitoring system for its processes in order to identify any decrease in efficiency and any loss. A monitoring system consists of a set of devices, procedures and diagnostic tools that follow every single step of a process. Early detection of operating conditions of the electrical drive that deviate from the optimal may avoid subsequent failures, or even faults.

In the diagnostic tool considered it is reasonable to assume that the only accessible points of the system are the alternating current (AC) input terminals (having in mind the recent trend to more and more integrated systems where the drive can be considered as a black-box). The opportunity of combining diagnostic and monitoring operations on an AC motor drive without using dedicated sensors cannot achieve a diagnosis as reliable as that provided by totally customized systems. Nevertheless, useful diagnostic indications can be obtained by this low-cost extension of the monitoring activity, and the reliability of the obtained indications can be significantly increased by considering the combination of advanced time-frequency, or time-scale, transforms, such as the wavelet transform, and the novel neuro-genetic approach described in Chapter 5. The output of the ANN, designed by this approach, will represent an index, that can be considered a risk coefficient giving the user the likelihood of being in fault conditions.

The investigated real-world system is depicted in Figure 7.1. The current signals are acquired at the input terminal of a Pulse-Width Modulation (PWM) inverter connected to a three-phase induction motor (230V, 50Hz). In the utilized induction motor both stator and rotor winding are available to the operator.

The Direct Current (DC) link between the Rectifier and the PWM inverter performs a filtering action with respect to the AC input, theoretically suppressing most information about the output circuits of the drive and the motor. Instead, it was proved that the faulty operating condition of the AC motor will appear on the AC side as a transient phenomenon or a sudden variation in the load power. The presence of this electrical transient in the current suggests an approach based on time-frequency or, better, time-scale analysis. In particular, the use of Discrete Wavelet Transform (DWT) [110] could be efficiently used in the process of separating the information.

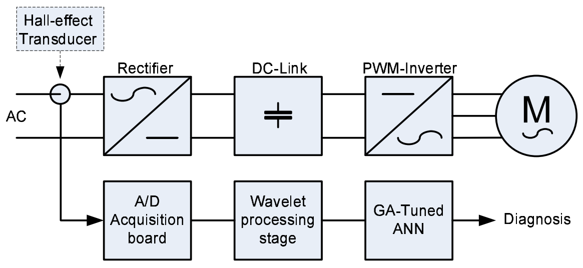


Figure 7.1: Block diagram of an AC drive with highlight on the measurement location and processing flow.

A classification of most common faults in an AC drive-motor system is reported in Figure 7.2. In particular, in this work only motor faults are considered.

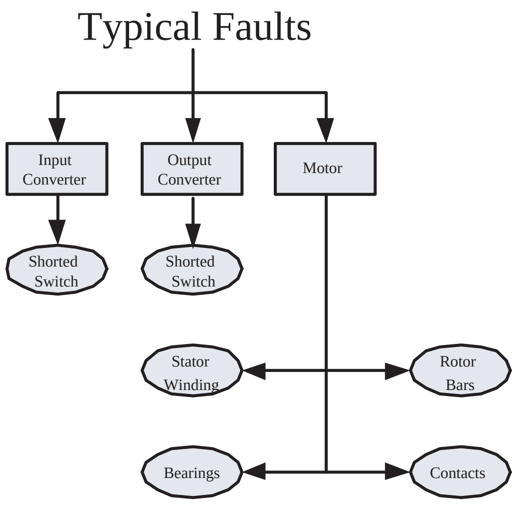


Figure 7.2: Classification of common fault types.

The application of the neuro-genetic approach to this problem involves the analysis of the signal, the load current, through wavelet series decomposition. The decomposition results in a set of coefficients, each carrying local time-frequency information. An orthogonal basis function is chosen, thus avoiding redundancy of information and allowing for easy computation.

The computation of the wavelet series coefficients can be efficiently performed with the Mallat algorithm [71]. Figure 7.3 shows the bandpass filtering, which is implemented as a lowpass $g_i(n)$ - highpass $h_i(n)$ filter pair which has mirrored characteristics.

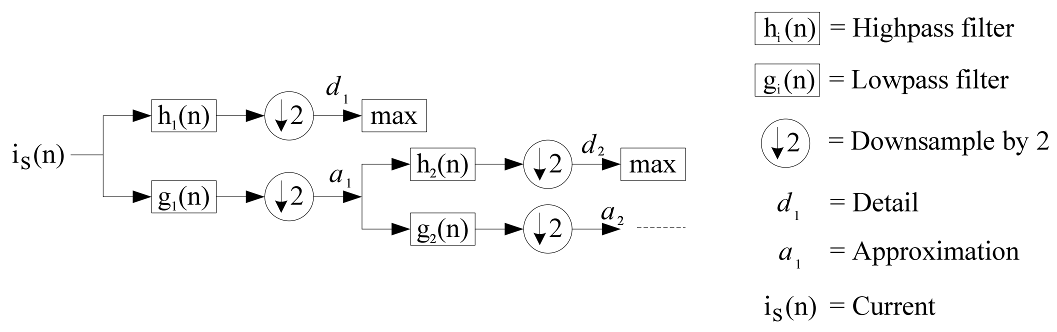


Figure 7.3: Bandpass filtering.

In particular, in the neuro-genetic application the 6-coefficient Daubechies wavelet [33] was used. In Table 7.1 the filter coefficients of the utilized wavelet are reported.

Table 7.1: Filter coefficients of the 6-coefficient Daubechies wavelet.

Filter Coefficients	Low-pass filter decomposition	High-pass filter decomposition
1	0.035226	-0.33267
2	-0.085441	0.80689
3	-0.13501	-0.45988
4	0.45988	-0.13501
5	0.80689	0.085441
6	0.33267	0.035226

The wavelet coefficients allow a compact representation of the signal to be achieved and to condense the features of the normal operating or faulty conditions in the wavelet coefficients. Conversely, the features of given operating modes can be recognized in the wavelet coefficients of the signal and the operating mode can be identified.

Employing the wavelet analysis, both the identification of the drive operating conditions (faulty or normal operation) and the identification of significant parameters for the specific condition have been obtained.

Figure 7.5 depicts the logical structure of the data describing a case of study. Each vector is known to have been originated in faulty or non-faulty conditions, so it can be associated with a fault index C equal to 1 (faulty condition), or 0 (normal condition).

The numerical representation of the current signals are then elaborated by a Matlab function to perform a wavelet transform [33, 71], as shown in Figure 7.4.

In this pseudocode the wavelet decomposition is carried out by iterating the algorithm described in Figure 7.1 for each wavelet decomposition level. Then, for each condition, the wavelet coefficients vector is created: in each decomposition level the maximum value of the wavelet coefficients is taken and considered as one element of the wavelet coefficients vector, whose size corresponds to the number of decompositions carried out. The process is defined by Equation 7.1, where i is the index correlated to the coefficient and j is the index correlated to the decomposition level.

```
for each case of study do
    Read the sample current signal
    Perform the wavelet decomposition
    for each level of decomposition do
        Extract the coefficients with maximum value
    end for
    Build an array of the max coefficients
    Normalize the array to infinity Norm
end for
```

Figure 7.4: Pseudocode of wavelet processing.

$$D_j = \max_{i=1}^{n} \{|w_{ij}|\} \tag{7.1}$$

Then, each vector is normalized with infinity norm, by applying Equation 7.2

$$d_j = \frac{D_j}{\|D\|_\infty} \tag{7.2}$$

This problem has been already approached with a neuro-fuzzy network, whose structure was defined *a priori*, trained with BP [31], as indicated in Figure 7.6.

w_8	w_7	w_6	w_5	w_4	w_3	w_2	w_1	C

Figure 7.5: A depiction of the logical structure of a case of study in the fault diagnosis problem. The elements w_1 to w_8 are the maximum coefficients for each level of wavelet decomposition; C indicates whether the case is faulty or not.

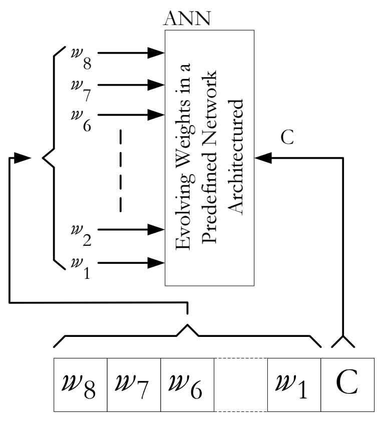


Figure 7.6: Approach previously implemented for the fault diagnosis problem by means of a neuro-fuzzy network with pre-defined topology trained by BP.

The neuro-genetic approach, subject of this P.h.D. dissertation, defines in this problem a suitable ANN, that represents a good classifier. In particular, the best EANN identified

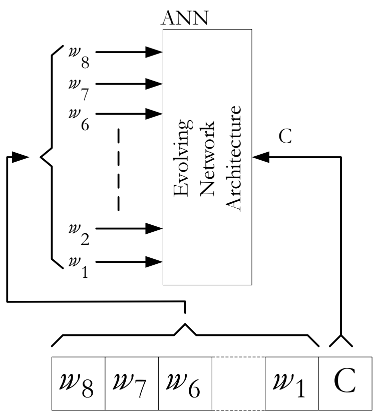


Figure 7.7: Neuro-genetic approach to the fault diagnosis problem.

will be able to recognize both complete and partial faults as indicated in Figure 7.8. Cases of faults have been obtained when the resistor value of a phase was artificially changed. A breakage of contiguous bars in a rotor can be, indeed, modeled by a varying winding resistance of one of the windings of the three-phase wound machine [15].

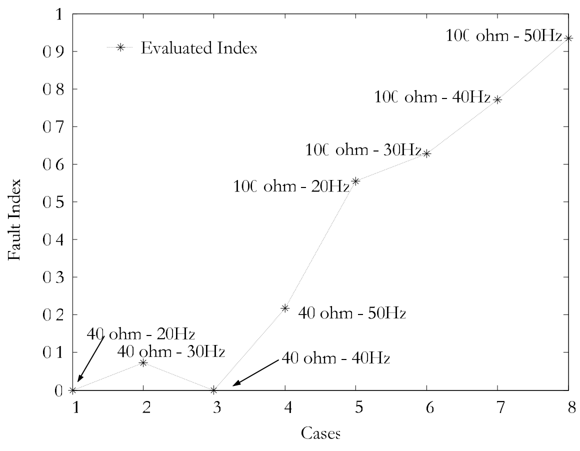


Figure 7.8: Fault index obtained: the test was performed considering the case of incipient fault situation.

In the simulations carried out all instances of the data are defined with 8 input attributes, corresponding to the maximum coefficients for each level of wavelet decomposition, and the parameter C, the fault index, is fixed, to simplify the problem, equal to 1 or 0 during the learning, testing, and validation steps, in order to respectively represent faulty ($C = 1$) or non faulty ($C = 0$) conditions.

In order to improve the usefulness of the neuro-genetic approach defined in this work, interesting studies could consider cases of partial fault, by defining different real values for the fault index C, for example $C = 0, 0.2, 0.4, ..., 1$, in the creation of virtual or real measurements. In this case, different instances with different graduated indexes would be defined in each neural training, test, and validation set.

Two kinds of simulations are carried out in this application: the first considers a data set created from a virtual model simulator. This simulation constitutes a kind of validation step, and at this point experiments are carried out by considering different mutation parameter settings and different population sizes, in order to find out the number of individuals that would be likely to give the best results for real simulations.

In the second kind of simulation, the data used for learning are obtained from a real engine. The following two sections describe in detail all the experiments carried out with the two different datasets.

7.3 Experiments on Virtual Simulator

All the data used for the 'simulated classification' have been obtained from a Virtual Test Bed (VTB) model simulator of a real engine. Few instances are obtained from the virtual simulator to create the datasets, giving 28 cases for the training, 12 for the test, and only 4 cases for the validation set. Several settings of five parameters, backpropagation bp, population size n, and the three mutation probabilities relevant to structural mutation, p^{+}_{layer}, p^{-}_{layer}, and p^{+}_{neuron}, have been explored in order to assess the robustness of the approach and to determine an optimal set-up. The p_{cross} parameter, that defines the probability to crossover, is set to 0 for all runs, because neither single-point crossover nor merge crossover give satisfactory results for this problem, due to the reasons explained in Section 5.7. All other parameters are set to the default values shown in Table 5.2.

For each run of evolutionary algorithm all the network evaluations are allowed (i.e. simulations of the network on the whole training set), including those performed by the backpropagation algorithm. All runs have been allocated the same fixed amount of neural network executions, to allow for a fair comparison between the cases with and without backpropagation. The results are summarized respectively in Table 7.3 and in Table 7.2. Ten runs are executed for each setting, of which the average and standard deviation for the best solutions found are reported.

7.3.1 Results

A first comment can be made regarding population size. In most cases it is possible to observe that the solutions found with a larger population are better than those found with a smaller population. With $bp = 1$, 15 settings out of 27 give better results with $n = 60$, while with $bp = 0$, 19 settings out of 27 give better results with the larger population.

In addition, it is possible to observe that, for this problem, there is a clear tendency for the runs using backpropagation ($bp = 1$) to consistently obtain better quality solutions (lower average fitness and smaller standard deviation).

In all cases, the corresponding standard deviation is sufficiently small to guarantee that suitable results can be found in a few runs. The experiments showed that the algorithm is somewhat robust with respect to the setting of its parameters, i.e., its performance is little sensitive of the fine tuning of the parameters.

7.4 Experiments on Real Simulator

After completion of the above phase, the neuro-genetic approach is applied to another kind of data set. Here, the data used for learning have been obtained from experiments of a real

Table 7.2: Experimental results for the engine fault diagnosis problem with BP=0.

setting	p^{+}_{layer}	p^{-}_{layer}	p^{+}_{neuron}	$bp = 0$			
				$n = 30$		$n = 60$	
				avg	stdev	avg	stdev
1	0.05	0.05	0.05	0.14578	0.013878	0.13911	0.0086825
2	0.05	0.05	0.1	0.1434	0.011187	0.13573	0.013579
3	0.05	0.05	0.2	0.13977	0.014003	0.13574	0.010239
4	0.05	0.1	0.05	0.14713	0.0095158	0.13559	0.011214
5	0.05	0.1	0.1	0.14877	0.010932	0.13759	0.014255
6	0.05	0.1	0.2	0.14321	0.0095505	0.1309	0.012189
7	0.05	0.2	0.05	0.14304	0.014855	0.13855	0.0089141
8	0.05	0.2	0.1	0.13495	0.015099	0.13655	0.0079848
9	0.05	0.2	0.2	0.14613	0.010733	0.14165	0.013385
10	0.1	0.05	0.05	0.13939	0.013532	0.13473	0.0085242
11	0.1	0.05	0.1	0.13781	0.0094961	0.13991	0.012132
12	0.1	0.05	0.2	0.13692	0.017408	0.13143	0.012919
13	0.1	0.1	0.05	0.13348	0.009155	0.1363	0.013102
14	0.1	0.1	0.1	0.13785	0.013465	0.13836	0.0094587
15	0.1	0.1	0.2	0.14076	0.01551	0.13994	0.011786
16	0.1	0.2	0.05	0.1396	0.0098416	0.13719	0.016372
17	0.1	0.2	0.1	0.13597	0.012948	0.14091	0.014344
18	0.1	0.2	0.2	0.14049	0.013535	0.13665	0.011426
19	0.2	0.05	0.05	0.13486	0.0079435	0.14068	0.013874
20	0.2	0.05	0.1	0.13536	0.0112	0.12998	0.013489
21	0.2	0.05	0.2	0.13328	0.0087402	0.1314	0.0088282
22	0.2	0.1	0.05	0.13693	0.0096481	0.13456	0.012431
23	0.2	0.1	0.1	0.13771	0.015971	0.13939	0.0092643
24	0.2	0.1	0.2	0.13204	0.010325	0.1378	0.01028
25	0.2	0.2	0.05	0.14062	0.012129	0.14005	0.011195
26	0.2	0.2	0.1	0.14171	0.008802	0.13877	0.0094973
27	0.2	0.2	0.2	0.14216	0.015659	0.13965	0.015732

Table 7.3: Experimental results for the engine fault diagnosis problem with BP=1.

setting	p^+_{layer}	p^-_{layer}	p^+_{neuron}	$bp=1$			
				$n=30$		$n=60$	
				avg	stdev	avg	stdev
1	0.05	0.05	0.05	0.11114	0.0070719	0.106	0.0027268
2	0.05	0.05	0.1	0.10676	0.003172	0.10606	0.0029861
3	0.05	0.05	0.2	0.10776	0.0066295	0.10513	0.0044829
4	0.05	0.1	0.05	0.10974	0.0076066	0.10339	0.0036281
5	0.05	0.1	0.1	0.1079	0.0067423	0.10696	0.0050514
6	0.05	0.1	0.2	0.10595	0.0035799	0.10634	0.0058783
7	0.05	0.2	0.05	0.10332	0.0051391	0.10423	0.0030827
8	0.05	0.2	0.1	0.10723	0.0097194	0.10496	0.0050782
9	0.05	0.2	0.2	0.10684	0.007072	0.1033	0.0031087
10	0.1	0.05	0.05	0.10637	0.0041459	0.10552	0.0031851
11	0.1	0.05	0.1	0.10579	0.0050796	0.10322	0.0035797
12	0.1	0.05	0.2	0.10635	0.0049606	0.10642	0.0042313
13	0.1	0.1	0.05	0.10592	0.0065002	0.10889	0.0038811
14	0.1	0.1	0.1	0.10814	0.0064667	0.10719	0.0054168
15	0.1	0.1	0.2	0.10851	0.0051502	0.11015	0.0055841
16	0.1	0.2	0.05	0.10267	0.005589	0.10318	0.0085395
17	0.1	0.2	0.1	0.10644	0.0045312	0.10431	0.0041649
18	0.1	0.2	0.2	0.10428	0.004367	0.10613	0.0052063
19	0.2	0.05	0.05	0.10985	0.0059448	0.10757	0.0045103
20	0.2	0.05	0.1	0.10593	0.0048254	0.10643	0.0056578
21	0.2	0.05	0.2	0.10714	0.0043861	0.10884	0.0049295
22	0.2	0.1	0.05	0.10441	0.0051143	0.10789	0.0046945
23	0.2	0.1	0.1	0.1035	0.0030094	0.1083	0.0031669
24	0.2	0.1	0.2	0.10722	0.0048851	0.1069	0.0050953
25	0.2	0.2	0.05	0.10285	0.0039064	0.1079	0.0045474
26	0.2	0.2	0.1	0.10785	0.008699	0.10768	0.0061734
27	0.2	0.2	0.2	0.10694	0.0052523	0.10652	0.0050768

engine, and the Table 7.4 shows some of the engine parameters settings used in the real measurement process.

Table 7.4: Experimental Setup Settings.

Motor			Inverter
Condition	Location	Fault Type	Frequency
Fault	Rotor	Additional 50 Ω on KL phase	10
Fault	Rotor	Additional 100 Ω on KL phase	25
Fault	Rotor	Additional 200 Ω on KL phase	30
Fault	Rotor	Additional 500 Ω on KL phase	45
Non Fault	No Location	No Fault	10
Non Fault	No Location	No Fault	25
Non Fault	No Location	No Fault	30
Fault	Rotor	Phase K open	10
Fault	Rotor	Phase L open	25
Fault	Rotor	Phase M open	30
...	...	...	...

All parameters are set to default values as indicated in Table 5.2. The bp parameter is kept to 1 for all runs, because, as seen in the previous application on virtual simulator, experiments without backpropagation algorithm do not give satisfactory results with respect to those with backpropagation. Also in this case, several settings of the three mutation probabilities relevant to structural mutation, p^{+}_{layer}, p^{-}_{layer} and p^{+}_{neuron}, have been explored in order to identify an optimal set-up also for the fault diagnosis application: the results are summarized in Table 7.5.

For each run of the evolutionary algorithm, up to 25,000 network evaluations are allowed, and as for the virtual simulator experiments, including those performed by the backpropagation algorithm. Ten runs were executed for each setting, for which the average and standard deviation of the test fitness values are reported. The approach is substantially robust with respect to the setting of mutation parameters, and the best solutions, on average, have been found with $p^{+}_{\text{layer}} = 0.1$, $p^{-}_{\text{layer}} = 0.2$, and $p^{+}_{\text{neuron}} = 0.05$.

The best model is a multi-layer perceptron with a phenotype of type $[4, 4, 4, 4, 1]$, which obtained a mean square error of 0.0050 on the test set. Figure 7.9 shows the median and minimum fitness functions of the best solution.

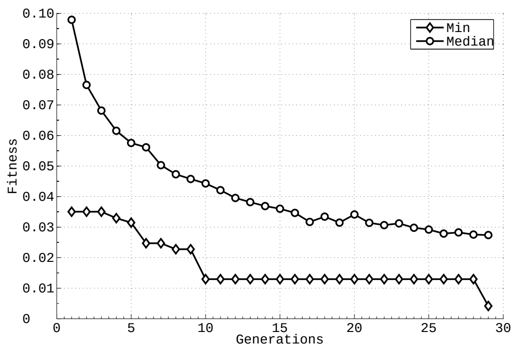


Figure 7.9: Median and Minimum fitness functions of best solution. The evolutionary process is stopped by the reaching of the maximum number of executions.

Table 7.5: Experimental results for the engine fault diagnosis problem.

setting	p^{+}_{layer}	p^{-}_{layer}	p^{+}_{neuron}	Avg	Std Dev
1	0.05	0.05	0.05	0.016754	0.0049053
2	0.05	0.05	0.1	0.018100	0.0092000
3	0.05	0.05	0.2	0.015825	0.0044010
4	0.05	0.1	0.05	0.014598	0.0032488
5	0.05	0.1	0.1	0.019499	0.0039437
6	0.05	0.1	0.2	0.015624	0.0020334
7	0.05	0.2	0.05	0.016075	0.0055915
8	0.05	0.2	0.1	0.017830	0.0041311
9	0.05	0.2	0.2	0.015934	0.0034586
10	0.1	0.05	0.05	0.016972	0.0072452
11	0.1	0.05	0.1	0.016880	0.0077800
12	0.1	0.05	0.2	0.015212	0.0039362
13	0.1	0.1	0.05	0.017032	0.0048141
14	0.1	0.1	0.1	0.017127	0.0056274
15	0.1	0.1	0.2	0.020672	0.0045185
16	0.1	0.2	0.05	0.013074	0.0059628
17	0.1	0.2	0.1	0.016231	0.0067702
18	0.1	0.2	0.2	0.015693	0.0026016
19	0.2	0.05	0.05	0.017530	0.0070005
20	0.2	0.05	0.1	0.019999	0.0044805
21	0.2	0.05	0.2	0.015552	0.0060033
22	0.2	0.1	0.05	0.014446	0.0042192
23	0.2	0.1	0.1	0.014501	0.0058740
24	0.2	0.1	0.2	0.013158	0.0062964
25	0.2	0.2	0.05	0.016855	0.0049215
26	0.2	0.2	0.1	0.015516	0.0059256
27	0.2	0.2	0.2	0.013769	0.0055727

The validation set is created with further measurements on the real engine, by defining parameter settings different from those used for the learning step, as indicated in Table 7.6.

Table 7.6: Validation Engine Settings.

Motor		Inverter	Values	
Condition	Fault Type	Frequency	Expected	Predicted
No Fault	Load	20	0	0.5729
Fault	Load	10	1	0.9187
No Fault	Load	25	0	0.3574
No Fault	Load	35	0	0.0780
Fault	Load	15	1	0.8326
No Fault	No Load	35	0	0.1598
Fault	Load	20	1	0.8668
Fault	No Load	10	1	0.7399
No Fault	No Load	50	0	0.9137
Fault	No Load	50	1	0.7354
Fault	No Load	25	1	0.8185
Fault	No Load	50	1	0.9164
Fault	No Load	10	1	0.9111
Fault	No Load	50	1	0.8185
Fault	No Load	10	1	0.7534

7.4.1 Results

The results obtained from the execution of the best solution on the validation set are depicted in Figure 7.10, in which a comparison between the expected values (solid marker) and values predicted by the neuro-genetic model (asterisk marker) is reported. The first observation is that the major part of the neuro-predicted values compare well with respect to expected values, giving a success percentage of 87%. As reported in Figure 7.10, only two of 15 validation cases do not compare well with respect to real engine simulations. This fact is substantially due to the use of unbalanced set of 0 or 1 cases for training and a test sets during the neuro-genetic simulation.

Better results are then obtained by considering output threshold values different from those considered in the validation set, equal to 0.5. Indeed, as Figure 7.10 shows, a setting of the output threshold equal to 0.6 improve the results previously obtained on the same validation set, because in that case only 1 validation case does not compare well with respect tho real engine experiments, increasing the percentage of success up to 93%.

The sensitivity and the specificity shapes of the best solution found with the neuro-genetic approach on the validation set are depicted in Figure 7.11, while the ROC shape on the same set is shown in Figure 7.12.

The results are also successfully compared with those obtained with a traditional neural network implementation [30, 31]. A comparison with the results obtained in [32] for a handcrafted neuro-fuzzy network did not reveal any significant difference. This is an extremely positive outcome, given the expert time and effort spent in hand-crafting the neuro-fuzzy network, as compared to the practically null effort required to set up the experiments carried out with the neuro-genetic approach. On the other hand, the amount of required computing resources was substantially greater with the neuro-genetic approach, and the relative standard deviation is sufficiently small to guarantee finding a satisfactory solution in a few runs.

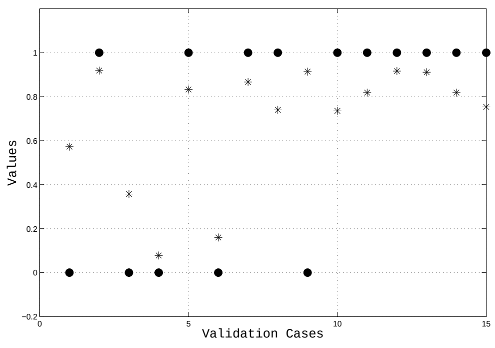


Figure 7.10: Comparison between expected validation outputs (solid marker) and predicted validation outputs (asterisk marker).

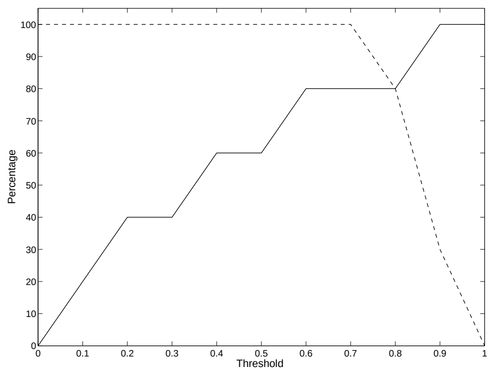


Figure 7.11: Sensitivity (dashed line) and specificity (solid line) shapes of the best model on the validation set.

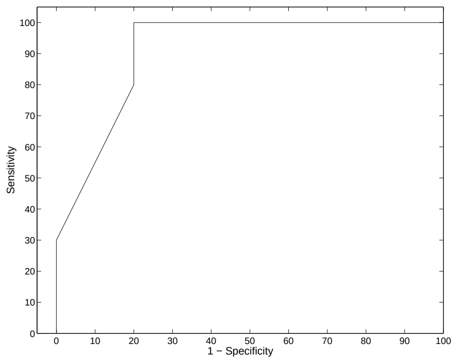


Figure 7.12: ROC Shape of the best solution on the validation set.

A further comparison can be carried out with a handcrafted neural network, with a predefined topology set to [8, 4, 1] and with weight connections randomly initialized. The neural network is trained with the backpropagation algorithm, on the same training set, obtaining a fitness value on the same test set equal to 0.2074 and a MSE of 0.4110. Figure 7.13 shows the sensitivity and the specificity shapes, while Figure 7.14 shows the ROC shape of that neural network.

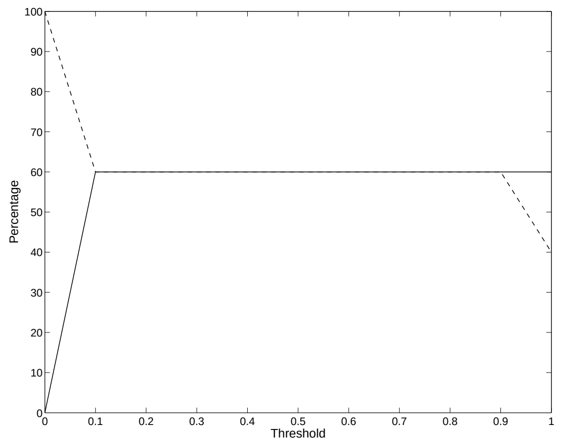


Figure 7.13: Sensitivity (dashed line) and specificity (solid line) shapes of the trial neural network on the validation set.

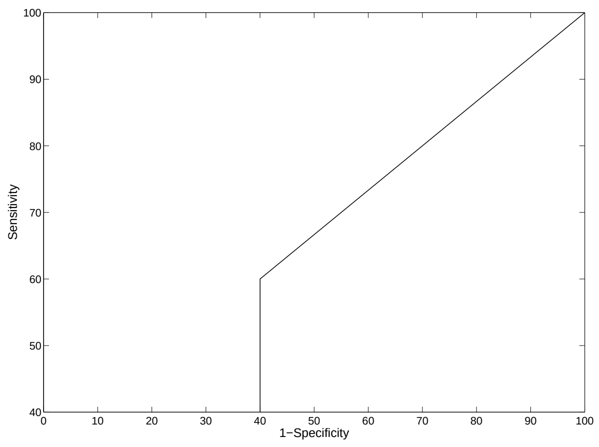


Figure 7.14: ROC Shape of the trial neural network on the validation set.

The results obtained from all the experiments show how the best solution found with the neuro-genetic approach is better than the solution obtained from a handcrafted neural network, both in terms of MSE values and in terms of the receiver operating characteristic.

Chapter 8

Brain Wave Analysis

8.1 Introduction

Brain Computer Interfaces (BCI) define a mode of communication for people affected by neuromuscular impairment. Neural diseases can break the slim and fragile line between thoughts and actions of a person affected by these disorders. In these cases, signal processing algorithms, like wavelets decomposition or Independent Component Analysis algorithms, may become helpful in exploiting of the residual functions of the brain at their best. Problems regarding the identification of healthy and pathological cases are solved by considering well-defined approaches, like Genetic Algorithms, Support Vector Machine, Neural Network models, etc, and several works presented in the literature have been used to classify electroencephalogram signals (EEG) and guess user intentions.

An example is recently presented in the literature by J.J. Merelo Guervós and colleagues [101], in which they consider an evolutionary Brain Computer Interface (BCI) design, based on a MLP neural network, that is trained by an evolutionary algorithm implemented with Evolving Objects (EO).

An application to a classification problem regarding brain stimuli is also considered in this chapter and the neuro-genetic approach, subject of this P.h.D. dissertation, is applied as a different sort of classifier to such a problem.

8.2 Problem Description

Brain Computer Interfaces (BCI) represent a new communication option for those suffering from neuromuscular impairment that prevents them from using conventional input devices, such as mouses, keyboards, joysticks, etc. This methodology has been developing quickly during the last few years, thanks to the increase of computational power and the availability of new algorithms for signal processing that can be used to analyze brain waves. During the first international meeting on BCI technology, Jonathan R. Wolpaw formalized the definition of the BCI systems as follows:

> A brain-computer interface (BCI) is a communication or control system in which the user's messages or commands do not depend on the brain's normal output channels. That is, the message is not carried by nerves and muscles, and, furthermore, neuromuscular activity is not needed to produce the activity that does carry the message [128].

According to this definition, BCI system appears as a possible and sometimes unique mode of communication for people with severe neuromuscular disorders like spinal cord injury or cerebral paralysis. In this sense, the exploitation of the residual functions of the brain may allow those patients to communicate.

8.3 Brain Wave Application

As pointed out by several works presented in the literature by Donchin, Wolpaw and colleagues[128, 36, 18], the human brain has an intense chemical and electrical activity, partially characterized by peculiar electrical patterns, which occur at specific times and at well-localized brain sites. All that activity is observable with a certain level of repeatability under well-defined environmental conditions. These simple physiological issues can lead to the development of new communication systems.

One of the most utilized electrical activities of the brain for BCI is the so-called P300 Evoked Potential. This wave is a late-appearing component of an Event Related Potential (ERP) which can be auditory, visual or somatosensory. It has a latency of about 300 ms and is elicited by rare or significant stimuli, when these are interspersed with frequent or routine stimuli. Its amplitude is strongly related to the unpredictability of the stimulus: the more unexpected the stimulus, the higher the amplitude. This particular wave has been used to make a subject choose by using different stimuli [38, 36].

The general idea of Donchin's solution is that the patient is able to generate this signal without any training. This is due to the fact that the P300 is the brain response to an unexpected or surprising event and is generated naturally. Donchin developed a BCI system able to detect an elicited P300 by signal averaging techniques (to reduce the noise) and used a specific method to speed up the overall performance.

A fundamental aspect for a human machine interaction system, such as BCI, is the need for a proper development environment that allows a real time interaction between the subject and the machine (this necessity is directed by the fact that the users need a short response time to keep their attention high). For this reason, a flexible modular environment is considered, useful to develop and to experiment various BCI systems. There are essentially two parts that every BCI system needs:

1. A dedicated hardware system that manages the stimulation, the EEG information acquisition (electrodes and amplifier samplers etc) from the subject, and the feedback obtained from him (visual, acoustic, haptic,etc.).

2. A system that deals with advanced signal processing and interpretation.

Donchin's idea has been adopted by Beverina and colleagues [18], in a brain-wave classification problem, implemented with a SVM approach, based on two kinds of BCI: the first one based on the Event Related Potential (ERP) and the second one based on Steady-State Visual Evoked Potential (SSVEP).

All instances of the datasets are then used in the neuro-genetic approach in order to identify the neural network that better works as classifier in this problem.

In the rest of the chapter the task and the stimulation paradigm defined by Beverina is presented, in order to better explain how the acquisition process works, by defining the brain features that are considered for each instance.

8.3.1 Task and Stimulation Paradigm

Beverina and colleagues developed in [18] a whole system with the purpose of controlling the motion of an object on a computer screen by means of visual stimuli. A graphical representation of the system implemented for BCI development is depicted in Figure 8.1. The experimental protocol has been divided in training and testing phases. The aim of the first is to set the parameters of the typical activity of every subject, i.e. maximal amplitude, in order to fix the threshold for the classification part. Then, in the testing phase, the subjects amplitude thresholds are recalculated adaptively in order to track the changes on the individual behavior.

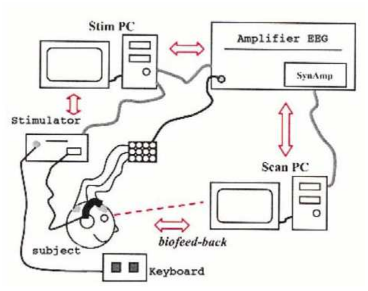


Figure 8.1: BCI Development System.

The authors show in Figure 8.2 what a subject sees during the interaction with the stimulator machine.

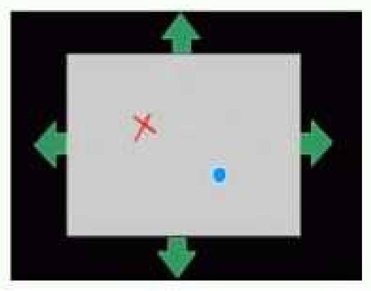


Figure 8.2: Stimuli.

The subject's task is to move an object (the blue ball) till it reaches the target (the red cross). The subject communicates to the machine which direction he is interested in, by posing his visual attention to only one of the four arrows and waiting until the arrow becomes yellow. Random target-direction visual stimuli, blinking green/yellow are presented in four positions: up arrow (as $\Uparrow$), right arrow (as $\Rightarrow$), down arrow (as $\Downarrow$), and left arrow (as $\Leftarrow$) on the computer screen.

The subject is seated in a sound attenuated room facing the computer screen, and is instructed to only look at the stimulus related with the desired object's movement direction, also called target stimulus. In that system, each stimulus consists in a single flash of an arrow which lasts for 150ms. A single trial may be defined as a sequence of user chosen target stimuli, subset of the random sequence, that allows the object to reach the end point.

Each stimulus is called *target stimulus* when it is chosen by the subject to move the object closer to the final target; otherwise it is called *non-target stimulus*. The authors suppose that every target stimulus elicits a P300 wave. Each stimulus occurred with a probability rate equal to 0.25. The trajectory of the object in each trial is decided by the subject.

Electrodes placed on the scalp of the subject capture the brain signals produced in response to the brain stimuli. At the end of the data acquisition process a new representation in a space of 78 features is given for every single sweep analyzed. Each of the instances obtained from this acquisition process defines the dataset used at first by Beverina and colleagues, and then in the experiments carried out by using the neuro-genetic approach.

8.4 Experiments and Results

The dataset provided by Beverina and colleagues consists of 700 negative cases and 295 positive cases. As previously defined, the features are based on wavelets, morphological criteria and power in different time windows, for a total of 78 real-valued input attributes, that correspond to the features of the signal response to the brain stimuli, and 1 binary output attribute, indicating the class (positive or negative) of the relevant case. A positive case is one for which the response to the stimulus is correct; a negative case is one for which the response is incorrect.

In [18], the authors created datasets with equal number of positive and negative cases. In the neuro-genetic approach two kinds of experiments are carried out. The first considers three balanced datasets of positive and negative cases, maintaining the same cardinality defined by Beverina and colleagues in their work, while the second considers the same instances, but with non-balanced datasets.

In the first experiment, for each run of the evolutionary algorithm, 218 positive cases from the 295 positive cases of the original set, and 218 negative cases from the 700 negative cases of the original set are extracted, to create a 436 case training dataset. For each run, also a 40 case test set is created, by randomly extracting 20 positive cases and 20 negative cases from the remainder of the original dataset, so that there is no overlap between the training and the test sets. The validation set contains only 10 cases equally distributed. In this experiment, for each run of the evolutionary algorithm, up to 25,000 network evaluations are allowed (i.e., simulations of the network on the whole training set), including those performed by the backpropagation algorithm.

In the first experiment several runs of the neuro-genetic approach have been carried out in order to find out optimal settings of the genetic parameters p^{+}_{layer}, p^{-}_{layer}, and p^{+}_{neuron}. All the runs are executed by considering $bp = 0$ and $bp = 1$, i.e., both without and with backpropagation, while the other parameters of the algorithm are set to the default values listed in Table 5.1.

The p_{cross} parameter is fixed to 0 for all runs, because neither single-point crossover nor merge crossover give satisfactory results also f or these simulations, due to the reasons explained in Section 5.7. The best solution has been found by using backpropagation algorithm, i.e. with the parameter $bp = 1$, and with p^{+}_{layer}=0.05, p^{-}_{layer}=0.1, and $p^{+}_{\text{neuron}} = 0.2$, although they do not differ significantly from other solutions found with $bp = 1$. The best solution is a network with a phenotype equal to $[4, 1]$ and with an accuracy on the validation set equal to 80%.

In the second experiment the training set is defined with 660 non-balanced instances, taken from the global dataset provided by Beverina and colleagues, while the test set and the validation set are defined with 143 non-balanced instances. Also in this case all runs of the neuro-genetic approach have been carried out in order to find out optimal settings of the genetic parameters p^+_{layer}, p^-_{layer}, and p^+_{neuron}, while the p_{cross} parameter is maintained to 0 for all runs. The mean and the standard deviation of the test fitness, calculated for each mutation parameter setting, are reported in Table 8.1.

Table 8.1: Brain Wave Experimental Results.

Setting	Parameter Setting			BP=1	
	p^+_{layer}	p^-_{layer}	p^+_{neuron}	avg	stdev
1	0.05	0.05	0.05	0.1493	0.0053
2	0.05	0.05	0.1	0.1505	0.0070
3	0.05	0.05	0.2	0.1489	0.0136
4	0.05	0.1	0.05	0.1488	0.0148
5	0.05	0.1	0.1	0.1557	0.0149
6	0.05	0.1	0.2	0.1498	0.0070
7	0.05	0.2	0.05	0.1487	0.0053
8	0.05	0.2	0.1	0.1453	0.0080
9	0.05	0.2	0.2	0.1494	0.0082
10	0.1	0.05	0.05	0.1536	0.0128
11	0.1	0.05	0.1	0.1486	0.0118
12	0.1	0.05	0.2	0.1500	0.0090
13	0.1	0.1	0.05	0.1504	0.0098
14	0.1	0.1	0.1	0.1493	0.0101
15	0.1	0.1	0.2	0.1499	0.0094
16	0.1	0.2	0.05	0.1547	0.0077
17	0.1	0.2	0.1	0.1541	0.0075
18	0.1	0.2	0.2	0.1530	0.0094
19	0.2	0.05	0.05	0.1480	0.0140
20	0.2	0.05	0.1	0.1501	0.0072
21	0.2	0.05	0.2	0.1503	0.0070
22	0.2	0.1	0.05	0.1486	0.0129
23	0.2	0.1	0.1	0.1502	0.0079
24	0.2	0.1	0.2	0.1495	0.0136
25	0.2	0.2	0.05	0.1550	0.0049
26	0.2	0.2	0.1	0.1517	0.0090
27	0.2	0.2	0.2	0.1510	0.0141

The best solution found is a neural network with a phenotype equal to $[211]$, with $bp = 1$ and with mutation parameter setting equal to p^+_{layer}=0.05, $p^-_{\text{layer}} = 0.2$, and $p^+_{\text{neuron}} = 0.1$. The best solution found has an average accuracy on the validation set equal to 84% with a best accuracy of 87.5%.

8.5 Comparison

A comparison of the results obtained with the neuro-genetic approach, and other previous results obtained from other works presented in the literature, are reported in Table 8.2, in which the accuracy obtained from the approaches is shown.

In the first experiment, that is carried out with balanced datasets, the sensitivity and the specificity shapes of the best solution found with the neuro-genetic approach on the validation set are depicted in Figure 8.3, and the ROC shape is shown in Figure 8.4. Then, the sensitivity and the specificity shapes of the best solution found in the second experiment, considering the unbalanced validation set, are reported in Figure 8.5, while the ROC shape on the same set is shown in Figure 8.6.

Table 8.2: Comparison between brain-wave classification approaches.

Method	Accuracy (%)
Beverina NN approach: with healthy cases	66.8
Beverina NN approach: with pathological cases	56.7
Beverina SVM approach	71.1
Neuro-genetic approach (with balanced datasets)	80
Neuro-genetic approach (with non-balanced datasets)	87.5

The first observation is that the two shapes depicted in Figures 8.5 and 8.3, respectively, show how the setting of the classification threshold does not represent a critical aspect in this problem, both with and without balanced datasets. Moreover, the first experiment does not obtain better results as the second one, which considers unbalanced datasets, probably due to the reduced dimensions of the considered datasets. However, further experiments on larger datasets could improve the global performances.

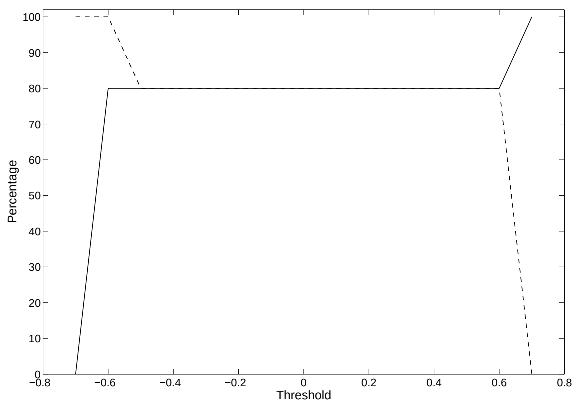


Figure 8.3: Sensitivity (dashed line) and specificity (solid line) shapes of the best model on balanced validation set.

The percentage of the accuracy obtained with the neuro-genetic approach also compares well with respect to another recent evolutionary approach implemented in the field of BCI selection and classification problems, carried out on a different dataset of P300 signals. The work is developed by Poli and colleagues [29] by using genetic algorithms. The results obtained in their approach have a percentage of accuracy equal to 86.94% by considering 3-linear terms and equal to 87.62%, by considering 4-linear terms. These results are similar to those obtained in the neuro-genetic approach by considering non-balanced datasets.

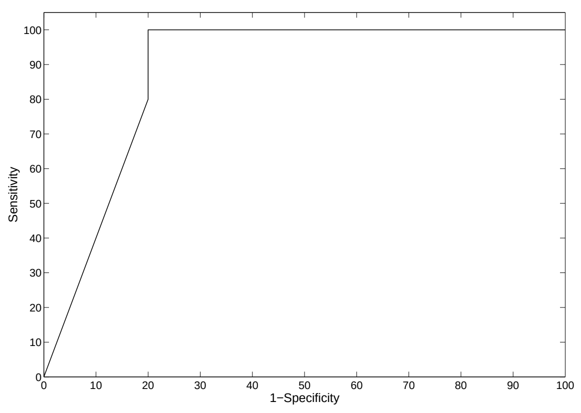


Figure 8.4: ROC Shape of the best solution on balanced validation set.

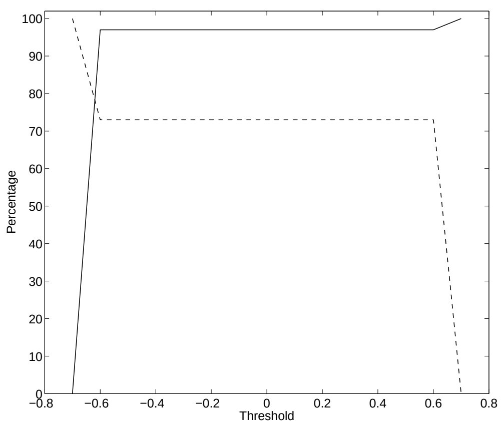


Figure 8.5: Sensitivity (dashed line) and specificity (solid line) shapes of the best model on unbalanced validation set.

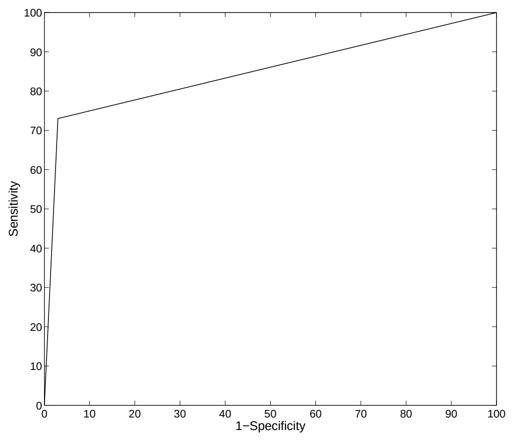


Figure 8.6: ROC Shape of the best solution on unbalanced validation set.

Chapter 9

Financial Applications

9.1 Introduction

The last application of the neural evolutionary approach, described in this P.h.D. dissertation, has been presented in [4] and in [3], and regards two different financial problems. The first application is aimed at constructing factor models of financial instruments; a sample statistical arbitrage is also present in this financial modeling, providing satisfactory results and significant profits.

In the second real-world financial application considered, the forecasting financial time-series using the neuro-genetic approach is tested. The approach uses different financial instruments to forecast the next-day closing price of the Dow Jones Industrial Average (DJIA).

9.2 Context of the Application

Market microstructure examines organized trading in instruments. Harris explains in [52] that *instruments* include common stocks, preferred stocks, bonds, convertible bonds, warrants, foreign exchange contracts, commodities, options, etc. A *market* is the place where traders gather to trade instruments.

Trading is a search problem. Buyers must find sellers, and sellers must find buyers. Every trader wants to trade at a good price. Sellers seek buyers willing to pay high prices. Buyers seek sellers willing to sell at low prices. Traders also must find traders who are willing to trade the quantities, or sizes, they desire.

Traders open two mainly kinds of positions respect to the market trading. Indeed, they have long positions when they own something. Traders with long positions profit when prices rise, and they try to buy low and sell high. Traders have short positions when they have sold something that they do not own. Traders with short positions hope that prices will fall so they can repurchase at a lower price. When they repurchase, they cover their positions, and short sellers profit when they sell high and buy at low price.

9.3 Financial Modeling

Factor models are statistical models, in this case implemented using artificial neural networks, that represent the returns of a financial instrument as a function of the returns of

Table 9.1: Input Market Indices.

Class	Ticker	Description
Foreign Exchange Rates	EURGBP	$1\text{EUR} = x\text{GBP}$
	GBPEUR	$1\text{GBP} = x\text{EUR}$
	EURJPY	$1\text{EUR} = x\text{JPY}$
	JPYEUR	$1\text{JPY} = x\text{EUR}$
	GBPJPY	$1\text{GBP} = x\text{JPY}$
	JPYGBP	$1\text{JPY} = x\text{GBP}$
	USDEUR	$1\text{USD} = x\text{EUR}$
	EURUSD	$1\text{EUR} = x\text{USD}$
	USDGBP	$1\text{USD} = x\text{GBP}$
	GBPUSD	$1\text{GBP} = x\text{USD}$
	USDJPY	$1\text{USD} = x\text{JPY}$
	JPYUSD	$1\text{JPY} = x\text{USD}$
Industry Representatives*	DNA	Biotecnologies
	TM	Motors
	DOW	Chemicals
	NOK	Communications
	JNJ	Drug Manufactures
	UN	Food
	BAB	Airlines
	XOM	Oil & Gas
	BHP	Metal & Mineral
	AIG	Insurance
	INTC	Semiconductors
	VZ	Telecom
	GE	Conglomerates
Commodities	OIL	Crude Oil $/barrel
	AU	Gold, $/Troy ounce
	AG	Silver, $/Troy ounce
US Treasury Bonds	TYX	30-year bond
	TNX	10-year note
	FVX	5-year note
	IRX	13-week bill

*) Representatives are, as a rule, the companies with largest market capitalization for their sector.

other financial instruments [52]. Factor models are used primarily for statistical arbitrage. A statistical arbitrageur builds a hedge portfolio consisting of one or more long positions and one or more short positions in various correlated instruments. When the price of one of the instruments diverges from the value predicted by the model, the arbitrageur puts on the arbitrage, by going long that instrument and short the others, if the price is lower than predicted, or short that instrument and long the others, if the price is higher. If the model is correct, the price will tend to revert to the value predicted by the model, and the arbitrageur will profit.

In order to study the capabilities of the neuro-genetic approach, a factor modeling problem is considered, whereby the Dow Jones Industrial Average (DJIA) is modeled against a number of other market indices, including foreign exchange rates, stock of individual companies taken as representatives of entire market segments, and commodity prices as shown in Table 9.1.

9.3.1 Experiments

In the factor modeling approach and in the following forecasting time series approach, the datasets are defined with the same method. The training and test sets are created by considering daily closing prices for the period since the 2nd of January, 2001 until the 30th of November, 2005. All data are divided in two different datasets, respectively with

Table 9.2: Financial Modeling Experimental Results.

Setting	Parameter Setting			BP=1	
	p^+_{layer}	p^-_{layer}	p^+_{neuron}	avg	stdev
1	0.05	0.05	0.05	0.2988	0.0464
2	0.05	0.05	0.1	0.2980	0.0362
3	0.05	0.05	0.2	0.3013	0.0330
4	0.05	0.1	0.05	0.2865	0.0368
5	0.05	0.1	0.1	0.2813	0.0435
6	0.05	0.1	0.2	0.3040	0.0232
7	0.05	0.2	0.05	0.2845	0.0321
8	0.05	0.2	0.1	0.2908	0.0252
9	0.05	0.2	0.2	0.3059	0.0208
10	0.1	0.05	0.05	0.2987	0.0290
11	0.1	0.05	0.1	0.3039	0.0341
12	0.1	0.05	0.2	0.3155	0.0396
13	0.1	0.1	0.05	0.3011	0.0395
14	0.1	0.1	0.1	0.2957	0.0201
15	0.1	0.1	0.2	0.3083	0.0354
16	0.1	0.2	0.05	0.2785	0.0325
17	0.1	0.2	0.1	0.2911	0.0340
18	0.1	0.2	0.2	0.2835	0.0219
19	0.2	0.05	0.05	0.2852	0.0292
20	0.2	0.05	0.1	0.2983	0.0309
21	0.2	0.05	0.2	0.2892	0.0374
22	0.2	0.1	0.05	0.3006	0.0322
23	0.2	0.1	0.1	0.2791	0.0261
24	0.2	0.1	0.2	0.2894	0.0260
25	0.2	0.2	0.05	0.2892	0.0230
26	0.2	0.2	0.1	0.2797	0.0360
27	0.2	0.2	0.2	0.2783	0.0369

1000 cases for the training set and 250 cases for the test set. The validation set consists of the daily closing prices for the period since the 1st of December, 2005 until the 13th of January, 2006. All time series of the three data sets for the two financial application are also preprocessed by deleting the long term components. This step is carried out by subtracting the 20 days moving average from those series.

In factor modeling all the parameters are set to the default values, defined at the initialization of the population in the neuro genetic approach. They are shown in Table 5.2 in Chapter 5. Several runs of this approach have been carried out in order to find out the optimal settings of the genetic parameters p^+_{layer}, p^-_{layer}, and p^+_{neuron}. For each run of the evolutionary algorithm, up to 100,000 network evaluations (i.e., simulations of the network on the whole training set) have been allowed, including those performed by the backpropagation algorithm.

The experiments, carried out in order to identify the best setting of the mutation probability, are presented in Table 9.2: here are reported data about the average and the standard deviation values of the fitness calculated on the test set of the best solutions found for each parameter settings over 10 runs.

Few experiments are conducted without BP algorithm and the first observation is that there is a striking superiority of the version of the algorithm which uses backpropagation. This is probably due to the fact that, whereas evolutionary algorithms are known to be quite effective in exploring the search space, they are in general quite poor at closing into a local optimum; backpropagation, which is essentially a local optimization algorithm, appears to complement well the neuro genetic approach.

Table 9.3: Input weights in the best neural network (second and third column), compared with the relevant coefficients of a linear model obtained by means of linear regression (fourth column).

Input	Weight		Linear
	1st Neuron	2nd Neuron	Regression
USDEUR	0,4953	0,4202	-16773
EURUSD	0,4214	0,3745	26979
USDGBP	0,1879	0,2300	139950
GBPUSD	-0,0095	0,1078	-56971
USDJPY	0,4642	0,4009	-630.66
JPYUSD	0,3441	0,3267	7131000
EURGBP	0,2991	0,2988	-239430
GBPEUR	0,2063	0,2414	112840
EURJPY	0,4246	0,3752	1120.3
JPYEUR	0,2048	0,2405	-17876000
GBPJPY	-0,1838	-0,0001	-438.21
JPYGBP	-0,1341	0,0307	12651000
AIG	-0,1914	-0,0048	15963
BAB	0,2319	0,2572	14326
BHP	0,2848	0,2900	12095
C	0,3638	0,3388	22405
DNA	0,2079	0,2424	3.5997
GE	0,4324	0,3813	45.966
INTC	0,0195	0,1258	23.786
JNJ	0,2774	0,2854	3.4063
NOK	0,3907	0,3555	-13.171
TM	0,8933	0,6665	3.1901
UN	0,3734	0,3448	10.95
VZ	0,4558	0,3958	25.732
XOM	0,2932	0,2952	21.554
OIL	0,3634	0,3386	-5.4066
AU	0,4418	0,3871	1.0003
AG	0,0969	0,1737	-31.656
TNX	0,3375	0,3225	316.7
TYX	0,2596	0,2744	-243.85
FVX	0,1404	0,2006	41.866
IRX	0,2127	0,2453	-93.432

Another observation is that the approach is substantially robust with respect to the setting of parameters other than bp.

The best solutions, on average, have been found with $p^{+}_{\text{layer}} = 0.2$, $p^{-}_{\text{layer}} = 0.2$, and $p^{+}_{\text{neuron}} = 0.2$, although they do not differ significantly from other solutions found with $bp = 1$.

9.3.2 Results

The best model over all runs performed has been found by the algorithm using backpropagation. The best model is a multi-layer perceptron with a phenotype of type $[2, 1]$, which obtained a mean square error of 0.39 on the test set. Figure 9.1 shows a satisfactory agreement between the output of the best model (predicted values) with the actual closing values of the DJIA on the validation set (desired values).

The weights of the inputs to the neurons of the hidden layer are shown in Table 9.3. Those neurons have biases of -1.8462 and -2.1212, and their output is connected with two weights of -1.8637 and 1.0041, respectively, to the output neuron, whose bias is -2.0586.

In order to assess the quality of the model, a comparison with simple linear regression on the same data is performed. Even though the financial modeling is represented by a non-linear function, a simple linear regression function is used in this chapter to make a comparison with the neuro genetic approach, since also several famous financial modeling problems presented in the literature have been validated by using linear regression models.

The linear regression yields the same linear model that has been used in the validation of the neuro-genetic approach, discussed in Chapter 6. The same equation has been maintained:

$$y = \sum_{i=1}^{32} w_i x_i, \tag{9.1}$$

where the w_i are those reported in Table 9.3 in the linear regression column. The prediction obtained by the linear regression model are compared with the best solution found, as shown in Figure 9.2. The solution obtained using the approach described in this P.d.D dissertation has a MSE of 1291.7, a better result compared to the MSE of 1320.5 of the prediction based on linear regression on the same validation dataset.

9.3.3 An Application to Statistical Arbitrage

In order to evaluate the usefulness of such a financial modeling, a paper simulation of a very simple statistic arbitrage strategy has been carried out starting on December 1, 2005 until Friday, January 13, 2006.

The strategy is as follows: on each day, the closing actual value of the DJIA index is considered, as well as the closing values of the 32 instruments of Table 9.1. A model estimate y of the ‘fair’ value (i.e., the value expected on the basis of the value of the other 32 instruments) of the DJIA index is obtained by applying the best neural network above described to all closing values.

Three cases may occur:

- $\ln \frac{y}{\mathrm{DJIA}} > \frac{\epsilon}{100}$: this means the DJIA is ‘cheaper’ than expected, and the strategy buys \$320,000 worth of it, while at the same time (short)-selling \$10,000 worth of each of the 32 instruments;
- $\frac{-\epsilon}{100} < \ln \frac{y}{\mathrm{DJIA}} < \frac{\epsilon}{100}$: no action is taken;
- $\ln \frac{y}{\mathrm{DJIA}} < \frac{-\epsilon}{100}$: this means the DJIA is overvalued, and the strategy (short)-sells \$320,000 worth of it, while at the same time buying \$10,000 worth of each of the 32 instruments.

Actually, buying (or selling) the same amount of the 12 exchange rates would have a zero net effect, because all pairs of transactions like ‘buy USDEUR’, ‘buy EURUSD’ would cancel. Therefore, no sensible trader would make those transactions. However, to simplify the exposition, such detail is overlooked.

As a consequence of this strategy, a hedge portfolio is created and reallocated each day. For the sake of simplicity, all transactions are carried out at the closing price and without cost. Of course, a more realistic simulation should take transaction costs into account. On the other hand, most brokers pay a short interest rebate, usually close to the federal funds rate or the LIBOR, on the deposit they require to collateralize the short selling of securities, and this factor should be taken into account as well.

Table 9.4: Simulation of a statistical arbitrage strategy based on the best solution found in the neuro genetic model. The operation in the Action column refers to the (short)-selling or buying of the DJIA, counterbalanced by a buying or (short)selling of the other instruments, as explained in the text. The NAV column shows the net asset value of the hedge portfolio at the end of each day.

Date	Action	NAV
December 1st, 2005	Sell DJIA	$0
December 2nd, 2005	Sell DJIA	$1,3730
December 5th, 2005	Buy DJIA	$5,0280
December 6th, 2005	Sell DJIA	$3,8060
December 7th, 2005	Sell DJIA	$6,1820
December 8th, 2005	Sell DJIA	$10,598
December 9th, 2005	No Action	$9,9760
December 12th, 2005	Sell DJIA	$16,910
December 13th, 2005	Buy DJIA	$7,6700
December 14th, 2005	Sell DJIA	$(4,921)
December 15th, 2005	Sell DJIA	$(7,339)
December 16th, 2005	Sell DJIA	$(7,119)
December 19th, 2005	Buy DJIA	$(5,446)
December 20th, 2005	Buy DJIA	$1,3930
December 21th, 2005	No Action	$4,2890
December 22th, 2005	Sell DJIA	$(7,704)
December 23th, 2005	Sell DJIA	$(6,613)
December 27th, 2005	Buy DJIA	$3,5260
December 28th, 2005	Buy DJIA	$8,6020
December 29th, 2005	No Action	$12,256
December 30th, 2005	Sell DJIA	$21,463
January 3rd, 2006	Buy DJIA	$21,430
January 4th, 2006	Buy DJIA	$22,490
January 5th, 2006	Buy DJIA	$20,881
January 6th, 2006	No Action	$17,637
January 9th, 2006	Sell DJIA	$14,646
January 10th, 2006	Sell DJIA	$13,535
January 11th, 2006	Sell DJIA	$12,220
January 12th, 2006	Sell DJIA	$17,160
January 13th, 2006	Sell DJIA	$17,536

Results

The net asset value (i.e., the total theoretical amount remaining after selling all long positions and covering all short positions at market value) of the hedge portfolio constructed by the above strategy during the validation market days of the simulation is shown in Table 9.4. The net asset value on Friday, January 13, 2006 can be taken as a (perhaps slightly optimistic) estimate of the profits of the arbitrage.

An inspection of Table 9.4 reveals that a starting capital of at least $2,562,000 would have been required to implement the strategy, and probably more, under the assumption that the broker demanded a cash deposit (plus about 2% more) as collateral for the short positions taken by the strategy, based on the peak exposure recorded on January 13. A profit of $17,536 would represent almost a 0.67% return on that capital. That is more than a 5.75% return on an annual basis, with a very moderate risk. This is not to suggest that the reader should take the model and put all of his or her savings on arbitrage in Wall Street; however, one can take such a result as an evidence that the model obtained actually contains some significant insight on the relationships between the financial instruments considered.

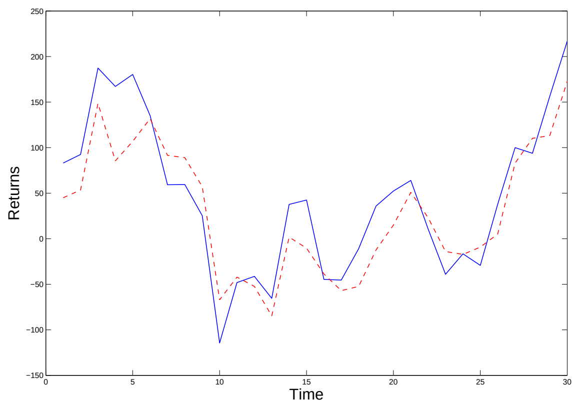


Figure 9.1: Comparison between the daily closing prices predicted by the best solution found with the neuro genetic model (dashed line)and actual daily closing prices (solid line) of the DJIA on the validation set.

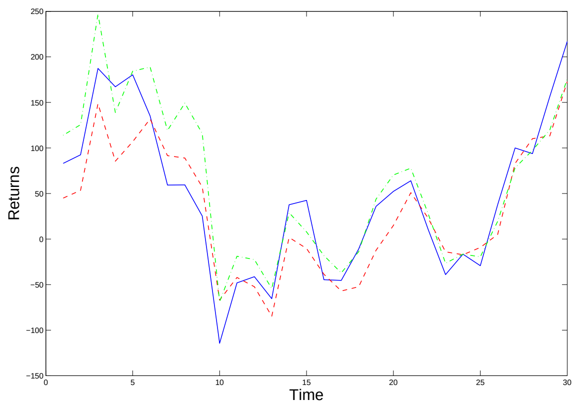


Figure 9.2: Comparison between the daily closing prices predicted by the best model (dashed line), those predicted by the linear regression (dash-dotted line), and the actual daily closing prices (solid line) of the DJIA on the validation set.

9.4 Financial Time Series Prediction

The results obtained by the neuro-genetic application for financial modeling, described in detail in the previous section and in [4], appear promising. For this reason, another financial application has been considered, focusing on time series prediction. The aim is to test the possibility of making a financial forecast of an instrument. In this second problem the same dataset defined for the first problem is maintained, adding information about the actual closing price of DJIA in the time series considered. Also the time series of this instrument are preprocessed by deleting the 20-days moving average, as for the other instruments.

Time series forecasting is one of the most widely undertaken research areas in statistics. There is a vast literature on the various linear time series models, among which the most popular traditional time series method used for various forecasting tasks is the autoregressive integrated moving average model (ARIMA) [120].

The traditional approach for time series modeling assumes that there is a underlying linear relationship between the past and the future values of a time series. Linear models are very easy to explain and implement, but their main drawback is that they may be totally inappropriate if the underlying process is nonlinear. This is the major difference from a non-linear ANN model for which there is no a priori assumptions on the relationship between the variables and the model form is defined by the given data.

As already stated in Chapter 3, ANNs possess the ability to determine non-linear relationships, and are particularly adept at dealing with noisy datasets.

ANNs are applied to predicting different time series, concerning metereology, geophysics, economy, and industrial management.

One of the most interesting application areas, along with financial modeling previously described, is financial trading, with particular attention to time-series predictions of financial instruments. Several approaches within the literature which deal with applying ANN techniques to investment and trading consider forecasting future data points using historical data sets. Research reviewed in this area generally attempts to predict the future values of some time series like base-time series data, e.g. closing prices, or time series derived from base data, (e.g., indicators - frequently used in technical analysis). Several works have been considered in this direction as representative of these techniques, implementing different kinds of neural networks. Radial Basis Function (RBF) networks have been implemented in a stock exchange market prediction in [130]. A technical analysis of different ANNs techniques is presented in [68], along with other related works presented in the literature. Several experiments of financial forecasting with NN are also reported in that work.

The main problem using ANNs is a local minimum entrapment which often occurs in a gradient descent algorithm, like backpropagation (BP) [103]. In order to overcome this drawback, evolutionary solutions are applied to financial forecasting. Evolutionary algorithms (EA) and genetic programming (GP) have been applied to financial time-series prediction by various authors since their beginning. A new system that utilizes genetic algorithms (GAs) to predict the future performances of individual stocks is presented in [70]. The system extended GAs from their traditional domain of optimization to inductive machine learning or classification; time-series forecasting is a special type of classification on which this financial application is concentrated. The implemented genetic model is compared with traditional solutions.

As indicated in [57], besides conducting an efficient exploration of the search space, with a population of models that adapt to market conditions, GP discovers automatically dependencies among the factors affecting the market and thus selects the relevant variables to be included in the model. This may be an advantage with respect to more traditional, and popular, autoregressive statistical approaches such as ARCH, GARCH and the like [47]. An application of GP to financial forecasting markets is described in [104], in which the objective is to predict a given realization of the time series of the daily closing values of the DJIA over a given period of time.

Interesting approaches consider the combination of ANNs and EA in order to combine the main advantage of each solution for financial prediction aimed to reduce some drawbacks. ANN has preeminent learning ability, but often exhibit inconsistent and unpredictable performance on data. In addition, it may not be possible to train ANN or to carry out the training task effectively without data reduction when the amount of data is so large. A solution to this problem is proposed with a new hybrid model of ANN and genetic algorithms (GAs) for instance selection in [64]. An evolutionary instance selection algorithm reduce the dimensionality of data and may delete noisy and irrelevant instances.

9.4.1 Financial Forecasting

The area of time series predictions is normally focused on attempting to predict the future values of a time series in one of two primary ways, either predicting future values of a series from the past values of that same series (intra-market analysis), or predicting future values of a series using data from different series (inter-market analysis). In many ways, these two primary prediction methodologies relate quite closely to technical analysis strategies. For example, the use (and projection) of a moving average over a series of stock prices could be regarded as predicting future values of a series (the moving average) from past values of the same series. In this financial forecasting problem, the second case is considered, and different financial instruments are used in order to predict the future closing price of the Dow Jones Industrial Average (DJIA).

Indicators in Technical Analysis are often composed by a number of data items, like price, volume, open-interest, etc. These indicators are commonly used to give indications of future direction of price. As previously defined, in this second financial application, the closing price of the DJIA for the next day is predicted from the last closing price of other market indices, by considering the time series of all instruments used in the financial modeling, already described, and the time series for the DJIA. All inputs are listed in Table 9.5.

9.4.2 Experiments

Financial forecasting model defines the training and test sets by considering the same indices as the previous financial modeling application, adding the daily DJIA closing price for the same period as a further input value. Also in this case, all data are divided into two different datasets, with 1000 cases for the training set and 249 cases for the test set, respectively. The validation set consists of the daily closing prices for the period from the 1st of December, 2005 to the 13th of January, 2006. Again, all time series of the three data sets are preprocessed to filter out the medium and long-term trends. This is accomplished by subtracting from every term of the data series the corresponding value of the 20-day moving average, thus yielding a zero-mean series of price residues.

In order to find out optimal settings for the genetic parameters, 10 runs of this approach

Table 9.5: Input Market Indices.

Class	Ticker	Description
Foreign Exchange Rates	EURGBP	1EUR = xGBP
	GBPEUR	1GBP = xEUR
	EURJPY	1EUR = xJPY
	JPYEUR	1JPY = xEUR
	GBPJPY	1GBP = xJPY
	JPYGBP	1JPY = xGBP
	USDEUR	1USD = xEUR
	EURUSD	1EUR = xUSD
	USDGBP	1USD = xGBP
	GBPUSD	1GBP = xUSD
	USDJPY	1USD = xJPY
	JPYUSD	1JPY = xUSD
Industry Representatives*	DNA	Biotecnologies
	TM	Motors
	DOW	Chemicals
	NOK	Communications
	JNJ	Drug Manufactures
	UN	Food
	BAB	Airlines
	XOM	Oil & Gas
	BHP	Metal & Mineral
	AIG	Insurance
	INTC	Semiconductors
	VZ	Telecom
	GE	Conglomerates
Market	DJIA	Dow Jones Industrial Average
Commodities	OIL	Crude Oil $/barrel
	AU	Gold, $/Troy ounce
	AG	Silver, $/Troy ounce
US Treasury Bonds	TYX	30-year bond
	TNX	10-year note
	FVX	5-year note
	IRX	13-week bill

*) Representatives show, as for financial modeling, the companies with largest market capitalization for their sector.

Table 9.6: Financial Forecasting Experimental Results.

Setting	Parameter Setting			BP=1	
	p^+_{layer}	p^-_{layer}	p^+_{neuron}	avg	stdev
1	0.05	0.05	0.05	0.2829	0.0330
2	0.05	0.05	0.1	0.2653	0.0396
3	0.05	0.05	0.2	0.2600	0.0227
4	0.05	0.1	0.05	0.2395	0.0340
5	0.05	0.1	0.1	0.2659	0.0407
6	0.05	0.1	0.2	0.2644	0.0231
7	0.05	0.2	0.05	0.2560	0.0377
8	0.05	0.2	0.1	0.2543	0.0361
9	0.05	0.2	0.2	0.2492	0.0320
10	0.1	0.05	0.05	0.2640	0.0295
11	0.1	0.05	0.1	0.2524	0.0275
12	0.1	0.05	0.2	0.2740	0.0337
13	0.1	0.1	0.05	0.2421	0.0337
14	0.1	0.1	0.1	0.2612	0.0364
15	0.1	0.1	0.2	0.2592	0.0369
16	0.1	0.2	0.05	0.2433	0.0355
17	0.1	0.2	0.1	0.2516	0.0393
18	0.1	0.2	0.2	0.2601	0.0343
19	0.2	0.05	0.05	0.2609	0.0302
20	0.2	0.05	0.1	0.2808	0.0145
21	0.2	0.05	0.2	0.2673	0.0378
22	0.2	0.1	0.05	0.2568	0.0276
23	0.2	0.1	0.1	0.2797	0.0185
24	0.2	0.1	0.2	0.2695	0.0165
25	0.2	0.2	0.05	0.2498	0.0307
26	0.2	0.2	0.1	0.2538	0.0327
27	0.2	0.2	0.2	0.2578	0.0467

have been carried out for each p^+_{layer}, p^-_{layer}, and p^+_{neuron} setting. For each run of the evolutionary algorithm, up to 100.000 network evaluation on the training set have been allowed, including those performed by BP. The average and the standard deviation of the test fitness of the best individual are reported in Table 9.6.

A few experiments conducted without backpropagation obtained, also for this real-world application, poor results, confirming that BP complements well evolution in the neuro-genetic approach. Moreover, also in this application, the evolutionary approach is robust with respect to the other parameter settings.

The best solutions, on average, have been found with $p^+_{\text{layer}} = 0.05$, $p^-_{\text{layer}} = 0.1$, and $p^+_{\text{neuron}} = 0.05$, although they are not so different from other solutions found with $bp = 1$.

9.4.3 Results

The best predictive model is a multi-layer perceptron with a phenotype of $[2, 1]$, which obtained a mean square error of 0.3607 on the test set. Table 9.7 shows the weights of the inputs to the neurons of the hidden layer. Those neurons have bias values equal to -4.0166 and -2.4028 respectively, and their outputs are connected with a weight of 1.2706 and -0.3339 to the output neuron, with a bias equal to 0.4474.

Also for time series prediction, the results obtained with neuro-genetic approach are compared with simple linear regression on the same data, in order to confirm its effectiveness. Also in this case the forecasting function is non-linear, but, as previously defined for financial modeling, a comparison with a linear regression function is carried out in order to

Table 9.7: Input weights in the best neural network (second and third column), compared with the relevant coefficients of a linear model obtained by means of linear regression (fourth column).

Input	Weight		Linear
	1st Neuron	2nd Neuron	Regression
USDEUR	0.0398	0.1958	-11208
EURUSD	0.1305	0.6417	3933.3
USDGBP	0.1200	0.5903	36063
GBPUSD	0.1315	0.6467	-9167.6
USDJPY	0.0227	0.1118	-19.804
JPYUSD	0.0622	0.3061	2304800
EURGBP	-0.093	-0.457	-74007
GBPEUR	0.0761	0.3743	24797
EURJPY	0.0636	0.3126	362.71
JPYEUR	0.1078	0.5300	-3280300
GBPJPY	-0.089	-0.439	-167.24
JPYGBP	0.0365	0.1796	3408900
AIG	0.1411	0.6941	1.6568
BAB	0.1488	0.7319	-1.4568
BHP	0.1184	0.5822	-13.853
C	-0.116	-0.572	-6.8453
DNA	0.1533	0.7540	-0.22727
GE	-0.014	-0.071	-1.9411
INTC	-0.051	-0.255	1.1808
JNJ	-0.023	-0.115	1.2725
NOK	0.1814	0.8924	1.9997
TM	0.0337	0.1657	0.33187
UN	0.0776	0.3815	1.4665
VZ	0.1837	0.9034	-4.0644
XOM	0.0492	0.2420	5.9805
OIL	-0.0280	-0.1375	-0.024104
AU	-0.018	-0.089	-0.068004
AG	0.0126	0.0620	-5.7084
TNX	0.0133	0.0654	540.27
TYX	0.0377	0.1855	-285.61
FVX	0.0301	0.1481	-306.06
IRX	0.0444	0.2182	25.871
DJIA P.D	0.1439	0.7079	0.92132

maintain the same validation approach presented in the literature on the same problem.

The function is the one shown in 6.3, already used in the validation of the subject of this P.h.D. dissertation, presented in Chapter 6, and used to validate the corresponding financial model. In this case the w_i are those reported in Table 9.7 in the linear regression column. The prediction obtained by the linear regression model are compared with the best ANN predictive model found, as shown in Figure 9.3. The comparison is performed by applying both models to the validation data set.

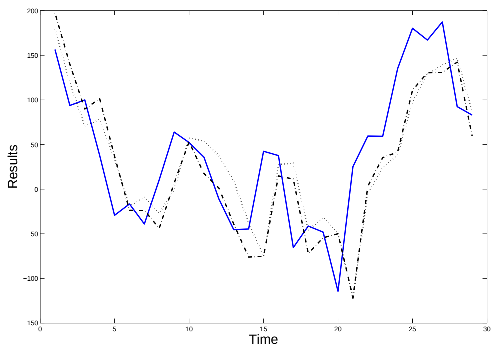


Figure 9.3: Comparison between the daily closing prices predicted by the best model (dotted line), those predicted by the linear regression (dash-dotted line), and the actual daily closing prices (solid line) of the DJIA on the validation set.

The predictive model obtained with the neuro-genetic approach has a MSE of 3353.8, while that obtained by linear regression has a MSE of 3177.6. In other words, both the ANN and the linear regression model have an equally poor performance when applied to the validation set; the difference in MSE is not statistically significant, especially when compared to the MSE of 0.3607 obtained by the ANN model on the test set.

9.4.4 Discussion

The results obtained by the neuro-genetic and linear models on the validation set in this forecasting problem seem to provide a substantial confirmation, at least for the daily DJIA time-series, of the efficient market hypothesis (EMH), which states that market prices incorporate at any moment all the relevant information available at that time.

If the EMH is true, any attempt to forecast future prices would be vain, as all price variations would depend on new information, not available at the moment the forecast is elaborated. A similar result has been obtained in a previous work focusing on the forecast of the DJIA by means of GP [104]: although that work won a 10-day DJIA prediction competition, it did so by substantially expecting that the next-day price would be the same as the most recent price.

Such an interpretation of the results is based on two observations. First of all, the neuro-genetic approach, previously validated and also applied, with excellent results, to a related financial modeling problem not involving forecast, and obtains results that are indistinguishable from those obtained by linear regression. Furthermore, as shown in Figure 9.3, it is possible to observe that the predictions obtained from both methods are essentially the series of the actual DJIA closing prices lagged by one day; this explains that both methods use the most recent price they can ‘see’ as their best guess at the next-day price.

Chapter 10

Conclusion

The work described in this thesis lies in the area of soft computing, and consists of the definition of a ‘neuro-genetic’ approach, based on evolutionary algorithms, to handle an optimized design of classifiers based on artificial neural networks. The attention is focused on the important contribution that this solution brings into the area of neural networks. Artificial neural networks offer the possibility of inferring unknown input-output relationships by supervised and unsupervised learning from existing data. As such, they are excellent at pattern recognition and function approximation. Artificial evolution helps in solving problems that are too hard for standard deterministic methods.

The work described in this thesis has shown, once more, that evolutionary algorithms represent a suitable technique to solve the problem of ANN design by yielding more than satisfactory results on a set of real-world problems. A key advantage of the approach proposed, which is computationally quite heavy, is the dramatical reduction of the effort required from a human expert, the most expensive resource of all, to design and set up an ANN for a given problem. This kind of evolutionary learning for ANNs has also been introduced to reduce and, if possible, to avoid the main problem of traditional gradient descent techniques, such as backpropagation (BP), that lies in the high chance of getting trapped in local minima. Much research effort has been spent to improve the performance of EAs and different selection schemes and genetic operators have been proposed in the literature. A survey of the state of the art has also been made in this thesis.

The evolution of neural networks for different applications has been a key issue in the ANN field. Neuro-genetic systems are coming of age, and consider different evolutionary solutions, like input data selection, learning rule optimization, architecture optimization of neural network models, connection weights optimizations. Each methodology has its strong and weak points and each can be augmented and strengthened when it is suitably combined with one or more of the others. In some cases, connection weights and architectures are difficult to design for neural networks. For this reason, in several approaches, the conjunction of topology and weight optimization has also been considered, as in the novel evolutionary approach presented in this thesis. This thesis, among other things, has attempted to propose solutions to the well-known issue of recombination of ANNs, while improving mutation operators with respect to the existing literature.

10.1 Remarks on the Neuro-Genetic Approach

Several remarks on the neuro-genetic approach better explain the most important concepts on which this thesis has been focused.

- *Backpropagation algorithm* The approach is designed to be able to take advantage of the backpropagation algorithm, that can also be used as a specialized decoder. Backpropagation becomes useful when the minimum of the error function currently found is close to a solution but not close enough to solve the problem in a satisfactory way. The results obtained in the real-world applications considered have pointed out the improvements given by the BP algorithm in the fine search in local space. In the neuro-genetic approach, BP learning produces also a Baldwin effect, allowing an individual to modify its phenotype in response to its environment, providing solutions that better adapt themselves to it.

- *Evolution Programs* The second remark underlines the fact that evolutionary operators implemented in this thesis are based on the well-defined idea of Evolution Programs (see Section 2.6), that allow each individual to represent, in a complete form, all the information necessary to apply suitable genetic operators.

- *Algorithm parameters* have to be defined before running of the algorithm. Few constant values related to the structure definition of each neural network have to be set at initialization, and a few algorithm parameters are set, while the best setting of the genetic parameters is obtained from the experiments, as shown in the applications considered in the approach. Even so, further studies will be carried out in order to reduce the number of the algorithm parameters, to improve the effectiveness of the approach. Anyway, the experiments carried out on several real-world problems strongly suggest that the performance of the approach is quite robust with respect to the setting of most parameters. In other words, it is roughly likely that the same parameter settings found by the experiments in this thesis will yield satisfactory performance when applied to other unrelated problems.

- *Individual setting* In this approach all individuals do not have a pre-determined topology, since the number of the hidden layers, and the maximum number of neurons in each layer are not determined in advance, nor bounded, maintaining diversity between all individuals in the population, and increasing the size of the search space of the global optimal solution. The problem of over-sized neural networks is avoided by means of a fitness function which contains a parsimony term, but the crossover operator represents again a critical issue when a recombination between networks with different topologies is considered.

- *Genetic Operators* An important consideration in this thesis is that a behavioral link is maintained between parents and offspring, in order to reduce behavioral disruption effects that occur in the evolutionary cycle. Weight mutation, with evolution strategies, and the topology mutation methods implemented allow to achieve higher correlation between parents and offspring in the population. Crossover still represents a critical aspect during evolution. Another aspect considers the fitness function used in this approach. As defined in successful evolutionary approaches presented in the literature, the difficulty in using a fitness function like that implemented in this thesis lies in the selection of suitable values for some coefficients of the function, which often involves tedious trial-and-error experiments. Moreover, a fitness function like that used in the neuro-genetic approach has theoretically to solve a multi-objective problem, because it is defined by combining two aspects usually conflicting as the cost and the accuracy of the network considered. A natural development of the neuro-genetic approach would be to regard the ANN design problem for what it is, namely a two-objective optimization problem. Accordingly, the approach should identify not a single 'optimal' design, but a whole array of Pareto-optimal (i.e., non-dominated) designs.

This neuro-genetic approach has been validated by comparing it first with a linear regression model, and then with two previously published solutions to benchmark problems, namely the Pima Indian Diabetes problem and the Breast Cancer Wisconsin problem.

Then the approach has been successfully applied to three different real-world applications. The first one predicts an incipient fault diagnosis in an electrical drives monitoring. The results obtained in this application compare well against alternative approaches, based on the conventional training of a predefined neuro-fuzzy network with BP. In particular, they are not significantly different from the results obtained in [32] for a handcrafted neuro-fuzzy network. This is a positive outcome, given the expert time and effort spent in a trial-and-error network design, as compared to the practically null effort required in the neuro-genetic approach.

In the second application a brain-wave signal-processing problem has been considered, in which a classification algorithm in the analysis of P300 Evoked Potential has been designed. The results obtained by the neuro-genetic approach appear promising after a comparison with a mature approach based on support vector machines.

Finally, the approach has been successfully applied to two financial problems. The first defines factor models of financial instruments. Furthermore, the results of a simulation of an arbitrage strategy which depends on the accuracy of the model, shows that the information given by the best solution found with the approach would have enabled an arbitrageur to gain significant profits. In the second financial application considered, the possibility of forecasting a financial time-series is tested. In the time series forecasting problem, the results obtained from the neuro-genetic approach appear statistically indistinguishable from those obtained by linear regression. In particular, the two results represent essentially the series of the actual DJIA closing prices lagged by one day. This confirms the efficient market hypothesis, by providing strong evidence that the next-day closing price of the DJIA is unpredictable, even by considering a variety of related financial indicators.

10.2 Discussion

All experiments carried out have not surprisingly demonstrated that the definition of classifiers (artificial neural networks) by using a trial and error approach performs worse results in classification problems than the ANNs identified by the neuro-genetic approach.

Another important result of this thesis is that all the experiments, carried out on validation tasks and on real-world applications, demonstrate that, even though a number of algorithm parameters have to be defined in the initialization step, the neuro-genetic approach is robust with respect to the setting of its parameters, allowing its performance to be little sensitive of the fine tuning of the parameters.

Of course, the approach proposed is computationally quite heavy. For example, the interpreted Matlab code that currently implements the approach has to run for 72 hours to solve the fault diagnosis problem (see Chapter 7). Nevertheless, computation time could be further reduced by using different programming language, as C++ or Java.

However, nowadays, the dollar cost of machine time is rapidly decreasing and nothing suggests a reversal of this trend is in view in the foreseeable future. The computational effort required by the neuro-genetic approach can be easily supported by distributed machines, able to perform parallel processing, thus increasing the performance of the computational model that has to be implemented. An important consequence of this is the global reduction of the work-load for human experts that can be obtained by using the neuro-genetic approach. Some of the experiments carried out in order to validate the approach (see Sections 7.3, 7.4) have made clear that expert-quality results can be replicated by the approach without requiring any intervention by a human expert.

10.3 Future Research

To conclude, an attempt to establish a working model to guide future research, is presented.

- *Efficiency and robustness* Future work will consider the study of the efficiency and the robustness of this approach even when input data are affected by uncertainty depending on errors introduced by measurement instrumentations. A further improvement could be given by removing algorithm parameters, even though this approach has been demonstrated to be robust with respect to parameter tuning.

- *Genetic operators* Further studies on new crossover operators could improve the genetic algorithm implementation, by making crossover as little disruptive as possible. The new merge-crossover implemented in this work seems to be a promising step in that direction, even though its use did not boost the performance of the algorithm significantly in the present form. An in depth study could also be carried out by considering mutation operators applied to perturb network structure and weights, and the corresponding evolution strategies implemented. Further studies of the neuro-genetic approach should involve designing better fitness functions to solve its multi-objective problem.

- *Possible improvements of evolution* could result from studying a novel coevolutionary approach, which relies on the cooperation between this neuro-genetic approach and a second one, based on genetic programming, in order to optimize the neural network design. In particular, both the genetic programming approach and the neuro-genetic approach should be used to optimize the activation functions of the neural networks defined in a population, using the resulting functions of GP.

List of Publications

[1] A. Azzini, L. Cristaldi, M. Lazzaroni, A. Monti, F. Ponci, and A.G.B. Tettamanzi. Incipient fault diagnosis in electrical drives by tuned neural networks. In *Proceeding of the IEEE Instrumentation and Measurement Technology Conference, IMTC'06*, April 2006.

[2] A. Azzini, M. Lazzaroni, and A.G.B. Tettamanzi. A neuro-genetic approach to neural network design. In F. Sartori, S. Manzoni, and M. Palmonari, editors, *AI*IA 2005 Workshop on Evolutionary Computation*. AI*IA, Italian Association for Artificial Intelligence, September 20 2005.

[3] A. Azzini and A.G.B. Tettamanzi. Financial forecasting with a neuro-genetic approach. In *Workshop italiano sulla vita artificiale e della giornata di studio italiana sul calcolo evoluzionistico GSICE 2006*. ISSN 1970-5077, Siena, September 15 2006.

[4] A. Azzini and A.G.B. Tettamanzi. A neural evolutionary approach to financial modeling. In Maarten Keijzer et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2006*. ACM Press, New York, NY, July 8-12 2006.

[5] A. Azzini and A.G.B. Tettamanzi. A neural evolutionary classification method for brain-wave analysis. In *Proceedings of the European Workshop on Evolutionary Computation in Image Analysis and Signal Processing, EVOIASP 2006*, Budapest, HU, April 10-12, 2006.

[6] A. Azzini. An evolutionary approach for neural model optimization. In M. Giacobini and Yano van Hemert, editors, *EVOPHD 2006: First European Graduate Student Workshop on Evolutionary Computation*. Budapest, HU, April 10-12, 2006.

[7] A. Azzini and A.G.B. Tettamanzi. A New Genetic Approach for Neural Network Design. Volume: Engineering Evolutionary Intellingent Systems, A. Abraham, C. Grosan and W. Pedricz, editors, Springer Verlag Publisher. To appear.

[8] A. Azzini, M. Lazzaroni, and A.G.B. Tettamanzi. Un approccio neuro-genetico per la progettazione di reti neurali. *Intelligenza Artificiale. Periodico trimestrale dell'Associazione Italiana per l'Intelligenza Artificiale*, To appear.

Bibliography

[1] H.A. Abbass. Speeding up backpropagation using multiobjective evolutionary algorithms. *Neural Computation*, 15(11):2705–2726, August 2003.

[2] A. Abraham. Meta learning evolutionary artificial neural networks. In *Neurocomputing*, volume 56, pages 1–38, 2004.

[3] R.W. Anderson. Learning and evolution: a quantitative genetics approach. *Journal of Theoretical Biology*, 175:89–101, 1995.

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

[5] G. Arulampalam and A. Bouzerdoum. Application of shunting inhibitory artificial neural networks to medical diagnosis. In *Proceedings of the Seventh Australian and New Zealand Intelligent Information Systems Conference*, November 2001.

[6] W. Atmar. Notes on the simulation of evolution. *IEEE Transactions on Neural Networks*, 5:130–147, 1994.

[7] T. Bäck, D.B. Fogel, and Z. Michalewicz, editors. *Evolutionary Computation 1, Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol and Philadelphia, Dirac House, Temple Back, Bristol, UK, 2000.

[8] T. Bäck, D.B. Fogel, and Z. Michalewicz, editors. *Evolutionary Computation 2, Advanced Algorithms and Operators*. Institute of Physics Publishing, Bristol and Philadelphia, Dirac House, Temple Back, Bristol, UK, 2000.

[9] T. Bäck, F. Hoffmeister, and H.P. Schwefel. A survey of evolutionary strategies. In R. Belew and L. Booker, editors, *Prooceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9, San Mateo, CA, 1991. Morgan Kaufmann.

[10] T. Bäck and H.P. Schwefel. Evolutionary computation: An overview. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 20–29, Nagoya, Japan, May 1996. IEEE Press.

[11] J.M. Baldwin. A new factor in evolution. *American Naturalist*, 30(354):34–41, 1896.

[12] W. Banzhaf, G. Beslon, S. Christensen, J.A. Foster, F. Képes, V. Lefort, J.F. Miller, M. Radman, and J.J. Ramsden. From artificial evolution to computational evolution:

a reserach agenda. *Nature, Reviews, Genetic Persperctives*, 7:729–735, September 2006.

[13] W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic Programming, An Introduction*. Morgan Kaufmann, San Francisco, CA, 1998.

[14] J. Basak. Online adaptive decision trees: Pattern classification and function approximation. *Neural Computation*, 18:2062–2101, 2006.

[15] A. Bellini, F. Filippetti, G. Franceschini, and C. Tassoni. Closed-loop control impact on the diagnosis of induction motors faults. *IEEE Transactions on Industry Applications*, 36(5):1318–1329, September-October 2000.

[16] D.P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. In *Optimization Methods and Software*, volume 1, pages 23–34. Gordon and Breach Science Publishers, 1992.

[17] H. Bersini and G. Seront. In search of a good crossover between evolution and optimization. *Parallel Problem Solving from Nature*, 2:479–488, 1992.

[18] F. Beverina, G. Palmas, S.Silvoni, F. Piccione, and S. Giove. User adaptive bcis: Ssvep and p300 based interfaces. *PsychNology Journal*, 1(4):331–354, 2003.

[19] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer Verlag, 2006.

[20] F.Z. Brill, D.E. Brown, and W.N. Martin. Fast genetic selection of features for neural network classifiers. *IEEE Transactions on Neural Networks*, 3(2):324–328, 1992.

[21] R. Callan. *The essence of Neural Network*. Prentice Hall Europe, Essex, England, 1999.

[22] P. A. Castillo, M. G. Arenas, J. G. Castellano, J. J. Merelo Guervós, A. Prieto, V. Rivas, and G. Romero. Lamarckian evolution and the baldwin effect in evolutionary neural networks, 2006.

[23] P. A. Castillo, J. González, J.J. Merelo Guervós, A. Prieto, V. Rivas, and G. Romero. SA-Prop: Optimization of multilayer perceptron parameters using simulated annealing. In *IWANN99*, pages 661–670, 1998.

[24] P.A. Castillo, M.G. Arenas, J.J. Merelo Guervós, V.M. Rivas, and G. Romero. Multiobjective optimization of ensembles of multilayer perceptrons for pattern classification. In T.P. Runarsson et al., editor, *Proceedings PPSN IX*, LNCS 4193, pages 453–462. Springer-Verlag, 2006.

[25] P.A. Castillo, J.J. Merelo Guervós, J. González, A. Prieto, V. Rivas, and G. Romero. G-prop-III: Global optimization of multilayer perceptrons using an evolutionary algorithm. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, page 942, Orlando, Florida, USA, 13–17 1999. Morgan Kaufmann.

[26] P.A. Castillo, J.J. Merelo Guervós, A. Prieto, I. Rojas, and G. Romero. Statistical analysis of the parameters of a neuro-genetic algorithm. *IEEE Transactions on Neural Networks*, 13(6), November 2002.

[27] P.A. Castillo, V. Rivas, J.J. Merelo Guervós, J. González, A. Prieto, and G. Romero. G-prop II: Global optimization of multilayer perceptrons using GAs. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 2022–2028, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press.

[28] D.J. Chalmers. The evolution of learning: An experiment in genetic connectionism. In In D.S. Touretzky, J.L. Elman, and G.E. Hinton, editors, *Connectionist Models: Proceedings of the 1990 Summer School*, pages 81–90. Morgan Kaufmann, San Mateo, California, 1990.

[29] L. Citi, R. Poli, C. Cinel, and F. Sepulveda. Feature selection and classification in brain computer interfaces by a genetic algorithm. In Maarten Keijzer, editor, *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, July 2004.

[30] L. Cristaldi, A. Ferrero, M. Lazzaroni, and A. Monti. A wavelet-based approach to diagnostic and monitoring for ac drives. In *Proceedings of the Instrumentation and Measurement Technology Conference, IMTC'02*, Anchorage, AK, USA, May 2002. IEEE Press.

[31] L. Cristaldi, M. Lazzaroni, A. Monti, and F. Ponci. A neurofuzzy application for ac motor drives monitoring system. *IEEE Transactions on Instrumentation and Measurement*, 53(4):1020–1027, August 2004.

[32] L. Cristaldi, M. Lazzaroni, A. Monti, F. Ponci, and F.E. Zocchi. A genetic algorithm for fault identification in electrical drives: a comparison with neuro-fuzzy computation. In *Proceeding of the IEEE Instrumentation and Measurement Technology Conference, IMTC'04*, Como, Italy, May 2004.

[33] I. Daubechies. *Ten Lectures on Wavelet*. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1992.

[34] L. Davis. Adapting operator probabilities in genetic algorithms. In J. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69. Morgan Kaufmann, San Mateo, California, 1989.

[35] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, USA, 1991.

[36] E. Donchin, K.M. Spencer, and R. Wijesinghe. The mental prosthesis: assessing the speed of a p300-based brain-computer interface. *IEEE Transactions on Rehabilitation Engineering*, 8(2):174–179, June 2000.

[37] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, 2003.

[38] L.A. Farwell and E. Donchin. Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials. *Electroencephalography and Clinical Neurophysiology.*, 70(6):510–523, December 1988.

[39] E.F.M. Filho and A.C.P. de Leon Ferreira de Carvalho. Evolutionary design of mlp neural network architectures. In *Proceedings of the The 4th Brazilian Symposium on Neural Networks*, pages 58–65. SBRN, 1997.

[40] D. Fogel. *Evolutionary Computation: The Fossil Record.* IEEE Press, New York, USA, 1998.

[41] D.B. Fogel. Applying evolutionary programming to selected traveling salesman problems. *Journal of Cybernetic Systems*, 24:27–36, 1993.

[42] D.B. Fogel. The advantages of evolutionary computation. In *Proceedings of the International Conference on Biocomputing and Emergent Computation (BCEC97)*, pages 1–11. World Scientific, 1997.

[43] D.B. Fogel. Introduction to evolutionary computation. In *Evolutionary Computation 1, Basic Algorithms and Operators*, chapter 1. Institute of Phisics Publishing, Bristol and Philadelphia, 2000.

[44] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley and Sons, New York, USA, 1966.

[45] D. E. Goldberg. Genetic algorithms and walsh functions: Part 1, a gentle introduction. *Complex Systems*, 3:129–152, 1989.

[46] D.E. Goldberg. *Genetic Algorithms in Search Optimization & Machine Learning.* Addison-Wesley, Addison-Wesley, 1992.

[47] C. Gourieroux. *ARCH Models and Financial Applications.* Springer Verlag, 1997.

[48] CORPORATE PDP Research Group. Psychological and biological models. computational models of cognition and perception. In D.E. Rumelhart J.L. McClelland, editor, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2. MIT Press, Cambridge, 1986.

[49] J.J. Merelo Guervós, M. Patón, A. Cañas, A. Prieto, and F. Morán. Optimization of a competitive learning neural network by genetic algorithms. In A.Prieto J.Mira, J.Cabestany, editor, *New Trends in Neural Computation*, number 686 in Lecture Notes in Computer Science, pages 185–192. Springer Verlag, 1993.

[50] P.J.B. Hancock. Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification. In L.D. Whitley and J.D. Schaffer, editors, *Proceedings of the Third International Workshop on Combinations Genetic Algorithms Neural Networks*, pages 108–122, 1992.

[51] S. Harp, T. Samad, and A. Guha. Towards the genetic synthesis of neural networks. *4th International Conference on Genetic Aglorithms*, pages 360–369, 1991.

[52] L. Harris. *Trading and exchanges: market microstructure for practitioners*. Oxford University Press, Oxford, 2003.

[53] S. Haykin. *Neural Networks, A comprehensive Foundation*. Macmillan College Publishing Company, New York, 1994.

[54] W.D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In C.G. Langton, C. Taylor, J.D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 313–324. Addison-Wesley, Redwood City, CA, 1992.

[55] G.E. Hinton and S.J. Nowlan. How learning can guide evolution. *Complex System*, 1:495–502, 1987.

[56] A. Hoffman. *Arguments on Evolution: a Paleontologist's Perspective*. Oxford University Press, New York, USA, 1989.

[57] H. Iba and N. Nikolaev. Genetic programming polynomial models of financial data series. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC'00*, pages 1459–1466. IEEE Press, July 2000.

[58] M.M. Islam, X. Yao, and K. Murase. A constructive algorithm for training cooperative neural network ensembles. *IEEE Transactions on Neural Networks*, 14(4):820–834, July 2003.

[59] R. Jain and A. Abraham. A comparative study of fuzzy classifiers on breast cancer data. In J. Mira, editor, *IWANN 2003*, pages 512–519. Springer-Verlag, Heidelberg, 2003.

[60] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 184–192, San Francisco, CA, 1995. Morgan Kaufmann.

[61] K.A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[62] K.A. De Jong. *Evolutionary Computation, A unified approach*. The MIT Press, Cambridge, Massachusetts, 2006.

[63] R. Keesing and D.G. Stork. Evolution and learning in neural networks: the number and distribution of learning trials affect the rate of evolution. *Advanced in Neural Information Processing Systems*, 3:805–810, 1991.

[64] K. Kim. Artificial neural network with evolutionary instance selection for financial forecasting. *Expert Systems with Applications*, 30:519–526, 2006.

[65] S. Knerr, L. Personnaz, and G. Dreyfus. Handwritten digit recognition by neural networks with single-layer training. *IEEE Transactions on Neural Networks*, 3:962–968, November 1992.

[66] J.R. Koza. *Genetic Programming*. The MIT Press, Cambridge, Massachusetts, 1992.

[67] J.R. Koza. *Genetic Programming II.* The MIT Press, Cambridge, Massachusetts, 1994.

[68] M. Lam. Neural network techniques for financial performance prediction: integrating fundamental and technical analysis. *Decision Support Systems*, 37:567–581, 2004.

[69] F.H.F. Leung, H.K. Lam, S.H. Ling, and P.K.S. Tam. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural Networks*, 14(1):54–65, January 2003.

[70] S. Mahfoud and G. Mani. Financial forecasting using genetic algorithms. *Applied Artificial Intelligence*, 10:543–565, 1996.

[71] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, San Diego, CA, 1999.

[72] O. L. Mangasarian, R. Setiono, and W.H. Wolberg. Pattern recognition via linear programming: Theory and application to medical diagnosis. In T.F. Coleman and Y. Li, editors, *Large-scale numerical optimization*, pages 22–30. SIAM Publications, Philadelphia, 1990.

[73] V. Maniezzo. Granularity evolution. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithm and Their Applications*, page 644, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers.

[74] V. Maniezzo. Searching among search spaces: Hastening the genetic evolution of feedforward neural networks. In *International Conference on Neural Networks and Genetic Algorithms, GA-ANN'93*, pages 635–642, 1993.

[75] V. Maniezzo. Genetic evolution fo the topology and weight distribution of neural networks. *IEEE Transactions on Neural Networks*, 5(1):39–53, January 1994.

[76] S.J. Mason and N.E. Graham. Areas beneath the relative operating characteristics (roc) and relative operating levels (rol) curves: Statistical significance and interpretation. *Quarterly Journal of the Royal Meteorological Society*, 128(584):2145–2166, July 2002.

[77] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Program.* Springer-Verlag, Springer-Verlag Berlin Heidelberg New York, 1996.

[78] G.F. Miller, P.M. Todd, and S.U. Hegde. Designing neural networks using genetic algorithms. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384, 1989.

[79] M.F. Moller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, 1993.

[80] D. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th International Conference on Artificial Intelligence*, pages 762–767. Morgan Kaufmann, 1989.

[81] P. Mordaunt and A.M.S. Zalzala. Towards an evolutionary neural network for gait analysis. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2, pages 1238–1243, 2002.

[82] Pablo Moscato and Michael G. Norman. A memetic approach for the travelling salesman problem, implementation of a computational ecology for combinatorial optimisation on message-passing systems. In *Proceedings of the International Conference on Parallel Computing and Transputer Applications*. IOS Press, Amsterdam, 1992.

[83] M.C. Moze and P. Smolensky. Using relevance to reduce network size automatically. *Connection Science*, 1(1):3–16, 1989.

[84] H. Muhlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm (bga). *Evolutionary Computation*, 1(4):335–360, 1993.

[85] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998.

[86] J.R. Oliver. Discovering individual decision rules: an application of genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 216–222. Morgan Kaufmann, San Mateo, California, 1993.

[87] G.A. Vijayalakshmi Pai. A fast converging evolutionary neural network for the prediction of uplift capacity of suction caissons. In *Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems*, pages 654–659, 2004.

[88] P.P. Palmes, T. Hayasaka, and S. Usui. Mutation-based genetic neural network. *IEEE Transactions on Neural Networks*, 16(3):587–600, May 2005.

[89] N.G. Pedrajas, C.H. Martinez, and J.M. Pérez. Covnet: A cooperative coevolutionary model for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 14(3):575–596, May 2003.

[90] S.G. Pierce, Y.Ben-Haim, K. Worden, and G. Manson. Evaluation of neural network robust reliability using information-gap theory. *IEEE Transactions on Neural Networks*, Accepted for inclusion in a future issue of this journal, 2006.

[91] K. Polat, S. Sahan, H. Kodaz, and S. Gunes. A new classification method for breast cancer diagnosis: Feature selection artificial immune recognition system (fs-airs). In L. Wang, K. Chen, and Y.S. Ong, editors, *ICNC 2005*, pages 830–838. Springer-Verlag, Heidelberg, 2005.

[92] M.A. Potter. *The design and analysis of a computational model of cooperative coevolution*. PhD thesis, George Mason University, Fairfax, VA, 1997.

[93] P.A. Castillo; J. Carpio; J.J. Merelo Guervós; V. Rivas; G. Romero; A. Prieto. Evolving multilayer perceptrons. *Neural Processing Letters*, 12:115–127, October 2000.

[94] J.C. Figueira Pujol and R. Poli. Evolving neural networks using a dual representation with a combined crossover operator. In *Proceedings of the IEEE International Conference on Evolutionary Computation - ICEC*, pages 416–421, 1998.

[95] J.C. Figueira Pujol and R. Poli. Evolution of neural networks using weight mapping. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1170–1177, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann.

[96] A. Radi and R. Poli. Discovering efficient learning rules for feedforward neural networks using genetic programming. Technical report, University of Essex, Department of Computer Science, Essex, UK, January 2002.

[97] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.

[98] M. Redmiller and H. Braun. A direct adaptive method for faster backpropagation learning. In *Proceedings of the International Conference on Neural Networks*, pages 586–591, 1993.

[99] C.R. Reeves and S.J. Taylor. Selection of training data for neural networks by a genetic algorithm. In A. Eiben, D. Back, M. Schoenauer, and H.P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN V, volume 1498 of Lecture Notes in Computer Science*, pages 633–642. Springer-Verlag, Heidelberg, 1998.

[100] A. Ribert, E. Stocker, Y. Lecourtier, and A. Ennaji. A survey on supervised learning by evolving multi-layer perceptrons. In *Proceedings of the Third International Conference on Computational Intelligence and Multimedia Applications, ICCIMA '99*, pages 122–126, 1999.

[101] G. Romero, M. García Arenas, P.A. Castillo Valdivieso, and J.J. Merelo Guervós. Evolutionary design of a brain-computer interface. In *Proceedings of the International Work-Conference on Artificial Neural Networks, IWANN*, pages 669–676, 2005.

[102] G. Rudolph. Convergence analysis of canonical genetic algorithms. In *IEEE Transactions on Neural Networks*, pages 96–101, 1994.

[103] D. E. Rumelhart, J. L. McClelland, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[104] M. Santini and A. Tettamanzi. Genetic programming for financial time series prediction. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038, pages 361–370, Lake Como, Italy, 2001. Springer-Verlag.

[105] J.D. Schaffer, L.D. Whitley, and L.J. Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In L.D. Whitley and J.D. Schaffer, editors, *Proceedings of the Third International Workshop on Combinations*

Genetic Algorithms Neural Networks - COGANN 92, pages 1–37, Los Almatos, CA, 1992. IEEE Computer Society Press.

[106] H.P. Schwefel. *Numerical Optimization for Computer Models*. John Wiley, Chichester, UK, 1981.

[107] H.P. Scwefel. *Evolution and Optimum Seeking*. H.N. Wiley & Sons, New York, NY, USA, 1995.

[108] U. Seiffert. Multiple layer perceptron training using genetic algorithms. In *Proceedings of the European Symposium on Artificial Neural Networks, ESANN'2001*, pages 159–164, 2001.

[109] R. Setiono. Generating concise and accurate rules for breast cancer diagnosis. *Artificial Intelligence in Medicine*, 18:205–219, 2000.

[110] C. Sidney Burrus, Ramesh A. Gopinath, and Haitao Guo. *Introduction to Wavelets and Wavelet Transorms - A Primer*. Prentice Hall, Upper Saddle River, New Jersey 07458, 1998.

[111] J. Smith and T.C. Fogarty. Self adaptation of mutation rates in a steday state genetic algorithm. In *Proceedings of the IEEE First International Conference on Evolutionary Computation*, pages 318–326, 1996.

[112] J.W. Smith, J.E. Everhart, W.C. Dickson, W.C. Knowler, and R.S. Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications and Medical Care*, pages 261–265. IEEE Computer Society Press, 1988.

[113] K. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[114] K.O. Stanley and R. Miikkulainen. Efficient evolution of neural network topologies. In *Proceedings of the Congress on Evolutionary Computation, CEC'02*, pages 1757–1762. IEEE Computer Society Press, 2002.

[115] Y. Sun and S. Al-sharhan. Hybrid soft computing techniques for heterogeneous data classification. In *Proceeding of the International Conference on Fuzzy Systems*, pages 1511–1516, Honolulu, HI, USA, May 2002. IEEE Press.

[116] Z.H. Tan. Hybrid evolutionary approach for designing neural networks for classification. Electronic Letters(15), 2002.

[117] A. Tettamanzi and M. Tomassini. *Soft computing: integrating evolutionary, neural, and fuzzy systems*. Springer Verlag, Berlin, Germany, 2001.

[118] G.L. Tsirogiannis, D. Frossyniotis, K.S. Nikita, and A. Stafylopatis. A meta-classifier approach for medical diagnosis. In G.A. Vouros and T. Panayiotopoulos, editors, *Proceeding of the SETN 2004*, pages 154–163. Springer-Verlag Berlin, Heidelberg, 2004.

[119] E.D. Ubeyli. A mixture of experts network structure for breast cancer diagnosis. *Journal of Medical Systems*, 29(5):569–579, October 2005.

[120] E. Wan. Time series prediction by using a connectionist network with internal delay lines. *Time Series Prediction: Forecasting the Future and understanding the Past*, pages 195–217, 1993.

[121] N. Weymaere and J. Martens. On the initialization and optimization of multiplayer perceptrons. *IEEE Transactions on Neural Networks*, 5:738–751, September 1994.

[122] D. Whitley and T. Hanson. Optimizing neural networks using faster, more accurate genetic search. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 391–396, 1989.

[123] D. Whitley and J. Kauth. Genitor: A different genetic algorithm. Technical report, Colorado State University, Fort Collins, CO, 1988.

[124] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel computing*, 14:347–361, 1993.

[125] D.L. Whitley. Modeling hybrid genetic algorithms. In G. Winter, J. Périaux, M. Galán, and P. Cuesta, editors, *Genetic Algorithms*, pages 191–201. John Wiley and Sons, Chichester, 1995.

[126] R.P. Wiegand. *An Analysis of Cooperative Coevolutionary Algorithms*. PhD thesis, George Mason University, Fairfax, Virginia, 2003.

[127] W.H. Wolberg and O.L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. In *Proceedings of the National Academy of Sciences*, volume 87, pages 9193–9196, 1990.

[128] J.R. Wolpaw, N. Birbaumer, W.J. Heetderks, D.J. McFarland, P.H. Peckham, G. Schalk, E. Donchin, L.A. Quatrano, J.C. Robinson, and T.M. Vaughan. Brain computer interface technology: A review of the first international meeting. *IEEE Transactions on Rehabilitation Engineering*, 8(2):164–173, June 2000.

[129] X.Yao and Y. Liu. Evolving artificial neural networks through evolutionary programming. *Evolutionary Programming*, pages 257–266, 1996.

[130] X.B. Yan, Z.Wang, S.H. Yu, and Y.J. Li. Time series forecasting with rbf neural network. In *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*. Guangzhou, August 2005.

[131] B. Yang, X.H. Su, and Y.D. Wang. Bp neural network optimization based on an improved genetic algorithm. In *Proceedings of the IEEE First International Conference on Machine Learning and Cybernetics*, pages 64–68, November 2002.

[132] X. Yao. Global optimisation by evolutionary algorithms. In *Proceedings of the IEEE Second AIZU International Symposium on Parallel Algorithms/Architecture Synthesis*, pages 282–291. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[133] X. Yao. Evolving artificial neural networks. In *Proceedings on IEEE*, pages 1423–1447, 1999.

[134] X. Yao. Evolutionary computation, a gentle introduction. In *Evolutionary Optimization*, chapter 2. Kluwer Academic Publishers, Norwell, Massachusetts, 2002.

[135] X. Yao. *Evolutionary Optimization*. Kluwer Academic Publishers, Norwell, Massachusetts, 2002.

[136] X. Yao and Y. Xu. Recent advances in evolutionary computation. *Computer Science and Technology*, 21(1):1–18, January 2006.

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

[138] X. Yao and Y.Liu. Making use of population information in evolutionary artificial neural networks. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 28(3):417–425, June 1998.

[139] X. Yao and Y.Liu. Towards designing artificial neural networks by evolution. *Applied Mathematics and Computation*, 91(1):83–90, 1998.

[140] J. Zhang. Selecting typical instances in instance-based learning. In *Proceedings of the Ninth International Machine Learning Conference*, pages 470–479. Morgan Kaufmann, Aberdeen, Scotland, 1992.