UNIVERSITAT POLITECNICA DE CATALUNYA

DEPARTAMENT DE LLENGUATGES I SISTEMES INFORMATICS

PROGRAMA DE DOCTORAT EN INTEL·LIGENCIA ARTIFICIAL

TESI DOCTORAL

Heterogeneous Neural Networks: Theory and Applications

June 2000

Memòria presentada per en Lluís A. Belanche Muñoz per a optar al títol de Doctor en Informàtica.

Directors: Julio José Valdés Ramos i Renato Alquézar Mancho

Chapter 6

Evolutionary Training of Heterogeneous Networks

Los dioses seleccionan entre los hombres,
eliminan los peores y los mejores, y sólo
dejan envejecer a los raros mortales que han
vivido sin odio ni exceso, para que conduzcan
la generación siguiente.

Aké Loba

A wide range of difficult problems or subproblems in Artificial Intelligence (AI) can be cast in the form of a function optimization problem. Among the global searching methods, *Evolutionary Algorithms* have been shown to be adaptable and general tools that have often outperformed traditional *ad hoc* methods. This Chapter is entirely devoted to *Evolutionary algorithms.* It serves the twofold purpose of introducing the field at the level needed by their use in this Thesis, and to explore how they can be used as effective neural weight optimizers.

6.1 Introduction

The quest for better and more general searching algorithms has never stopped. Since the 70's, new and powerful evolutionary methods have emerged that are particularly well suited for optimization —although this was not exactly their original purpose— mainly because of their generality, robustness, and conceptual (though not necessarily analytical) simplicity. In addition, the constant need for general-purpose optimization techniques has widen their horizon and boosted their widespread use.

The term Evolutionary Algorithms (EA) [Bäck, 96] is very general and includes many methods that have been (and are being) developed independently in the last 30 years. All of them are based on techniques that mimic or are inspired in population genetics, and have the added appeal of being easily realizable in parallel processes (both intuitively and physically).

Among them, the Breeder Genetic Algorithm (BGA) has been one of the last to emerge [Mühlenbein and Schlierkamp-Voosen, 93]. Nevertheless, despite its promising initial results when compared to other methods (evolutionary or not) it has not attracted a great deal of attention, possibly because of the enormous impact of other, in a sense already classical, Evolutionary Algorithms: Evolution Strategies (ES) and Genetic Algorithms (GA).

Evolutionary methods are immediate candidates for a learning algorithm oriented to neural network optimization (and, in particular, for *heterogeneous* neural networks). They are generally free from restrictive assumptions on the nature of the search space, as continuity requirements or existence of derivatives in the function computed by the network. The arguments of this function may or may not be discrete, and may be ordered or unordered. They also alleviate the problem of local minima -see (§2.1.8). Known drawbacks are *stagnation* (premature convergence), *epistasis* (unwanted gen interactions) and high computational demands. In return they have been shown to be generally robust and usually able to find reasonably good solutions for a great variety of problems [Bäck, Fogel and Michalewicz, 97].

In this Chapter, after introducing a generic evolutionary algorithm and two of its representatives (the standard or *canonical* GA and the BGA), we show how they can be enhanced to represent and manipulate heterogeneous information, in order to train a heterogeneous neural network. For the BGA, this implies the extension of genetic operators. A set of experiments is also carried out to investigate the use of the BGA for the weight optimization task in neural network training. The obtained settings are used in the experiments of Chapter (§9).

The Chapter is developed as follows. In Section 2 the basics of an Evolutionary Algorithm are outlined from a conceptual point of view. The two EA used in this Thesis (the GA and the BGA) are then surveyed as particular cases, and extended to cope with the problem of HNN training, the former in Section 3 and the latter in Section 4. The generic use of evolutionary learning algorithms for this task was reviewed in (§2.1.8). An additional summary is included specifically for the BGA, of which little use has been made to solve this particular problem. The Chapter ends with the conclusions derived from the study of these two EA as feasible candidates for neural network training.

6.2 Basics of an Evolutionary Algorithm

The term Evolutionary Algorithms refers to a big family of search methods based on concepts taken from Darwinian evolution of species and natural selection of the fittest. Some concepts from genetics are also present. Given a problem to be solved, usually in the form of a function to be optimized, an EA maintains a population of *individuals* that represent potential solutions to it. Each individual in the population is represented by a *chromosome* consisting of a string of atomic elements called *genes*. Each gene contains (represents) a variable, either for the problem or for the algorithm itself. The possible values of a gene are called *alleles* and the gene's position in the chromosome is called *locus* (pl. *loci*). There is also a distinction between the *genotype*, the genetic material of an individual, and the *phenotype*, the individual result of genotype development (that is, the born living thing). In EA the genotype coincides with the chromosome, and the phenotype is simulated via a *fitness function*, a scalar value —similar to a reinforcement— expressing how well an individual has come out of a given genotype[1]. However, there are many differences with natural evolution, reviewed in [Bäck and Schwefel, 96].

The search process usually starts with a randomly generated population and evolves over time in a quest for better and better individuals where, from generation to generation, new populations are formed by application of three fundamental kinds of operators to the individuals of a population, forming a characteristic three-step procedure:

1. *Selection* of the fittest individuals, yielding the so-called *gene pool*;
2. *Recombination* of (some of) the previously selected individuals forming the gene pool, giving rise to an offspring of new individuals;
3. *Mutation* of (some of) the newly created individuals.

By iterating this three-step mechanism, it is hoped that increasingly better individuals will be found (that is, will appear in the population). This reasoning is based in the following ideas:

1. The selection of the fittest individuals ensures that only the best ones[2] will be allowed to have offspring, driving the search towards good solutions, mimicking the natural process of selection, in which only the more adapted species are to survive.
2. By recombining the genetic material of these selected individuals, the possibility of obtaining an offspring where *at least* one child is better than any of its parents is high.
3. Mutation is meant to introduce new traits, not present in any of the parents. It is usually performed on freshly obtained individuals by slightly altering some of their genetic material.

There is a last operation involved, the *replacement* criterion, that basically says which elements, among those in the current gene pool and their newly generated offspring, are to be given a chance of survival onto the next generation. There are two basic strategies, generically denoted by (μ, λ) (the *comma* strategy) and $(\mu + \lambda)$ (the *plus* strategy). The letter μ denotes the population size and the letter λ the number of offspring to be generated out of the μ elements. In the plus case, both the parents and their (recombined and mutated) offspring are taken into account to form a new generation of again μ elements. In the comma case the parents, after generating offspring, die off and are considered to form the next generation.

An EA may be seen as a non-empty sequence of ordered operator applications: fitness evaluation, selection, recombination, mutation and replacement. The entire process iterates until one of the following criteria is fulfilled:

[1]In other disciplines, like Artificial Life methods, the phenotype is a real (or simulated) entity that interacts with an environment.

[2]Or the luckiest in some EA instances, like most GA.

1. *Convergence*: it happens because the individuals are too similar. Fresh and new ideas are needed, but recombination is incapable of providing them because the individuals are very close to one another, and mutation alone is not powerful enough to introduce the desired variability. Convergence can be monitored by two measures, called on-line performance (defined as the average of *average* individuals) and off-line (defined as the average of the *best* individuals) throughout the generations;

2. *Problem solved*: the global optimum is found up to a satisfactory accuracy (if the optimum is known);

3. *End of resources*: the maximum number of function evaluations has been reached.

Evolutionary Algorithms are effective mainly because their search mechanism keeps a well-balanced tradeoff between *exploration* (trying to always drive the search to the discovery of new, more useful, genetic material) and *exploitation* (trying to fine-tune good already-found solutions). Exploration is mainly dealt with by the mutation operator. Exploitation is carried out by the selection process and the use of recombination operators, although mutation may also play a role in the fine-tuning of solutions. The fitness function is built out of the function to be optimized (called the *objective function*). All EA represent the decision variables in the chromosome in one way or another, either directly as real values (like ES) or resorting to a discrete coding, usually binary (like most GA). The particular coding scheme is the classical knowledge representation problem in AI, and completely conditions the results. In addition, some algorithms (like ES) append their own variables to the representation in the form of auxiliary information that evolves with time like the other variables.

According to the representation scheme chosen, there must be a decoding method Γ — equivalent to the genotype to phenotype development— to decode the decision variables from their chromosomic representation:

$$\Gamma : i \in \Pi_t \rightarrow \vec{x}_i \in D$$

where Π_t stands for the population at a certain generation t, and D is the original problem space. Once decoded, these variables can readily be used as arguments of an objective function $F : D \rightarrow \mathbb{R}^+ \cup \{0\}$ to yield a fitness value. The fittest individuals are those with a lowest (in case of minimization) fitness value. Thus, the fitness function Φ associated to an individual i is defined as $\Phi(i) = F(\Gamma(i))$. Some EA require a form of post-processing such as a global rescaling function, but it is much more convenient to consider it as part of the selection mechanism itself.

Given Φ, Γ -usually the only problem-specific knowledge- an EA can be formally described by the conceptual algorithm in Fig. (6.1), parameterised by a tuple:

$$< \text{EA-Setup} >=< \Pi_0, \mu, \lambda, \Upsilon, \Omega, \Psi, \Theta, \Phi, \Xi > \tag{6.1}$$

where $\Pi_t = (i_1^t, i_2^t, \ldots, i_\mu^t)$ is the population at time t and thus Π_0 is the, usually random, initial population, μ the population size, λ the offspring size (out of μ), Υ the selection operator, Ω the recombination operator, Ψ the mutation operator, Θ the termination criterion, Ξ

the replacement criterion and Φ the fitness function. In this algorithm, operator sequencing on the population is as follows: Π_t represents the population at time or generation t, Π_t^{Υ} the same population after selection, Π_t' after recombination and Π_t'' after mutation, to form a new population Π_{t+1}, after application of the replacement criterion.

```
Procedure Evolutionary-Algorithm (<EA-Setup>)
{
        t:=0;
        evaluate $\Phi(i), \forall i \in \Pi_0$;
        while not($\Theta(\Pi_t)$) do
        {
            /* Create the gene pool $\Pi_t^{\Upsilon}$ */
               select: $\Pi_t^{\Upsilon} := \Upsilon(\Pi_t)$;

            /* Apply genetic operators */
               recombine: $\Pi_t' := \Omega(\Pi_t^{\Upsilon})$;
               mutate: $\Pi_t'' := \Psi(\Pi_t')$;

            /* Evaluate their effect */
               evaluate $\Phi(i), \forall i \in \Pi_t''$;

            /* Form the new generation */
               replace: $\Pi_{t+1} := \Xi(\Pi_t'' \cup \Pi_t^{\Upsilon})$;
               t := t+1
        }
}
```

Figure 6.1: Evolutionary Algorithm.

The three main representatives of EA are: Genetic Algorithms, proposed by Holland [Holland, 62], then settled [Holland, 75], and made popular [Goldberg, 89]; Evolution Strategies, developed by Rechenberg [Rechenberg, 65] and Schwefel [Schwefel, 65], during the 60's and more or less settled in the 70's [Rechenberg, 73], [Schwefel, 77]; and Evolutionary Programming (EP), introduced by Fogel [Fogel, 62] and spread by him and his coworkers [Fogel, Owens and Walsh, 66], an approach that resembles ES although they were developed independently. One of the main references to EA is [Bäck, 96]; another, good and brief survey is [Bäck and Schwefel, 96]. An excellent state-of-the-art and review of EA, and a useful departure point because of its rich set of references is [Bäck and Schwefel, 93]. Modern surveys and introductions to specific algorithms are [Bäck and Schwefel, 91] and [Bäck, 95] for ES; [Michalewicz, 92] for GA and [Fogel, 92] for EP.

6.3 Genetic Algorithms

6.3.1 Description of the Algorithm

The principles of GA were established in [Holland, 75]. A GA is a stochastic search procedure characterized by:

- A **population** of discrete structures (individuals) representing candidate solutions for the problem being solved;
- A **selection** mechanism based on the aptitude (fitness) of each individual, relative to the population;
- A set of idealized **genetic operators** that modify the individuals to create new genetic material.

In a standard GA the individuals are fixed-length strings of length L defined over an alphabet Σ, called chromosomes. The fitness function Φ gives a numeric and positive value to the adequacy of a given chromosome as a solution of the task at hand, that is: $\Phi : \Sigma^L \rightarrow \mathbb{R}^+ \cup \{0\}$.

The selection mechanism has the mission of favouring the better fit individuals to enter the gene pool, for reproduction and mutation, as the basis to form the next generation. The probability of being selected is directly proportional to the relation between the fitness of an individual and the fitness of the population. The simplest way to do so is:

$$p(i) = \frac{\Phi(i)}{\sum_{j=1}^{\mu} \Phi(j)} \tag{6.2}$$

where μ is the population size. If we view the population as mapped onto a roulette wheel, each individual i is assigned a fraction of space proportional to its ratio $p(i)$, which can be viewed as the probability of being selected. By spinning the wheel, the individuals are chosen to form the intermediate gene pool. This method is called *stochastic sampling with replacement.*

Let us define, for each individual, $p^*(i) = \mu p(i)$. In *remainder stochastic sampling*, for each individual fulfilling $p^*(i) \geq 1$ –that is, for above-average individuals– the number $[p^*(i)]$ (integer part) indicates how many copies are *directly* selected, with no intervention of chance; next, for these individuals, $p^*(i)$ is updated as $p^*(i) := p^*(i) - [p^*(i)]$. Then, *all* individuals place copies with probability $p'(i) = \frac{p^*(i)}{\sum_{j=1}^{\mu} p^*(j)}$, like in (6.2), until the gene pool is filled.

This procedure is efficiently implemented using a method known as *stochastic universal sampling*. The population is laid out in a random order on a roulette wheel, with space allotted in proportion to $p(i)$. The wheel has this time μ equally-spaced pointers. A single spin of the wheel simultaneously picks all the μ selected individuals. This selection method can be shown to be unbiased [Baker, 87].

A *scaling* mechanism is usually included in a GA before selection takes place, to keep roughly equal selection pressure across the generations. Early on in the search, there is a tendency for a few (initially) highly fit individuals to begin dominating the population. Also, in a mature population, unless using a form of rank-based selection –in which only the relative order is important–, selective pressure can become very low and the search stagnates. A simple and useful way of reducing these effects is by introducing a linear *scaling mechanism* [Goldberg, 89], as follows. Let Φ denote the fitness function and $\Phi_{min}, \Phi_{max}, \Phi_{avg}$ be the *minimum*, *maximum* and *average* fitness values in a generation and denote s the scaling function. It is required that:

$$\begin{aligned} s(\Phi_{avg}) &= \Phi_{avg} \\ s(\Phi_{max}) &= c\Phi_{avg} \end{aligned} \tag{6.3}$$

where $c \in (1,2]$ is the *scaling factor.* For example, for $c = 2$, this forces that the new Φ value for Φ_{max} be twice the average, which remains the same. However, by doing this, worse-than-average individuals can get a negative fitness, which cannot be accepted. A usually adopted solution is to scale with the maximum c' that does not yield any negative value:

$$c' = \underset{c\in(1,2]}{\text{argmax}} \left\{ \Phi_{min} - \frac{c\Phi_{avg} - \Phi_{max}}{c-1} \geq 0 \right\} \tag{6.4}$$

so that we get a new scaling s':

$$\begin{aligned} s'(\Phi_{avg}) &= \Phi_{avg} \\ s'(\Phi_{min}) &= 0 \end{aligned} \tag{6.5}$$

with $s'(\Phi_{max}) = c'\Phi_{avg}$ and typically $c' < c$. After selection takes place, genetic operators are applied to the members of the gene pool. The result of this application is a new generation. There are two main classes of genetic operators:

Crossover (a type of discrete recombination) is applied to randomly paired chromosomes with a certain probability, denoted P_{cross}. Typical values are $P_{cross} \in [0.6, 1.0]$. The outcome crossover (a new pair of chromosomes) is inserted into an intermediate population, where mutation will take place.

The way crossover operates is best seen with an example. Consider the following scenario for $\Sigma = \{0,1\}, L = 8$, where a cut point has been set at random:

$$101|00001 \longrightarrow 001|00001$$

$$001|11011 \longrightarrow 101|11011$$

where the chromosomic material of the two individuals has been crossed over. This is called *one point* crossover, and others are possible.

Mutation is applied after crossover to the generated individuals. Each gene in the intermediate population is altered with some probability, denoted P_{mut}. A typical value is $P_{mut} = 0.01$. More generally, P_{mut} is set to $\frac{1}{L}$. For example,

$$101\boxed{1}1011 \longrightarrow 101\boxed{0}1011$$

After the process of selection, crossover and mutation has taken place, the intermediate population is joined to the actual population (the generation at time t) to form, via the replacement criterion, a definite new population (the generation at time $t+1$). Two basic criterions are *worst individual*, where the two worst among the parents and their offspring are discarded, and *parent replacement*, where the parents are replaced unconditionally.

A *schema* can be roughly defined as a template describing a substring where some of its positions are defined while others are not, and its *defining length* is the maximum distance (number of loci) between two defined positions; e.g., in $010...1*0110**...0$, the inner part is a schema of length five. In a GA, it is more accurate to think that schemata, and not specific individuals, are what survives from generation to generation. Crossover tends to cut schemata of bigger lengths, which have a lower chance to survive. A lower bound on the *crossover survival probability* P_s of a schema with defining length δ, for one-point crossover is given by:

$$P_s \geq 1 - P_{cross}\frac{\delta}{l-1} \qquad (6.6)$$

where l is the overall length of the schema. With selection, the probability of survival of a schema depends on the average fitness (relative to population) of the schema instances present in the population, and on its defining length. The *Schema Theorem* states that high-fit schemata with short defining length are propagated exponentially [Goldberg, 89]. These short schema are called *building blocks.* In a GA, crossover leads the search in the genetic material towards finding building blocks (which can be seen as *partial solutions*) trying to assemble them in the hope that the obtained full-length chromosome represents a highly fit individual. This is the *building blocks hypothesis* (BBH) [Holland, 75]. When multiple-parameter problems are coded in a chromosome, very complex interactions arise, some of them induced by the coding and thus undesired (*epistatic effects*); therefore, it is general advice to position related genes close together.

6.3.2 The GA as a HNN trainer

The encoding of a ANN into a binary GA chromosome ($\Sigma = \{0,1\}$) is carried out –given a fixed architecture– by concatenation of the different weight representations, unit by unit, and layer by layer, in a precise (arbitrary) order. A GA chromosome is thus a long bitstring of constant length. Let $I = integer[i, i+k]$ the integer represented by the alleles contained in the genes from loci i to $i+k$, in base two. In this Thesis, the representation of each kind of weight is as follows:

Real-valued weights are represented in the usual form of unsigned fixed-point integers [Goldberg, 89], and decoded as:

$$r(I) := \frac{I}{I_{max}}(r^+ - r^-) + r^- \tag{6.7}$$

where $r(I)$ is the decoded real, I is the integer represented by the examined substring, I_{max} its maximum value, in this case $I_{max} = 2^k - 1$ ($I_{min} = 0$) and $[r^-, r^+]$ the desired real interval to be mapped.

Discrete weights, either ordinal or nominal are decoded directly as I.

Set weights are represented by taking each bit in $[i, i+k]$ as the characteristic function of a given element to the set, where $k-1$ is equal to the set cardinality.

Fuzzy quantities are represented as two real-valued numbers, mode and spread.

Missing values are encoded by specifying a desired proportion $\frac{1}{m}$ in the genotype. This means that one out of each m alleles in $[i, i+k]$ are to be decoded as missing values. Specifically, a decoded integer I is interpreted as missing if I **mod** $m = 1$. The modulus is compared to one to allow the zero to be decoded as such. Assuming $k = 32$, a typical value can be $m = 256$.

The *fitness* function is simply the inverse of the chosen error (e.g.. the mean square error).

6.4 The Breeder Genetic Algorithm

Although a clever design of specialized genetic operators would definitely improve GA performance (as a consequence of introducing problem-specific knowledge) we believe that other evolutionary techniques better suit the problem of minimizing a (non-differentiable) error function that has such a heterogeneity in its constituting variables, many of them continuous, difficult for the binary coding of the GA.

When coding an ANN into such a chromosome, highly complex interactions develop, due to the influence that a given weight on a hidden unit has on the whole network computation. What is more, the binary (base two) representation of the real-valued weights carries with it extra interactions between non-neighbouring genes, thus inducing strong epistatic effects in the GA processing, which only knows of a long chromosome where the atomic pieces are bits[3]. In these conditions, it is at least doubtful that the BBH can be applied.

Therefore, a step forward in HNN training can be made by using a method that does not need any encoding scheme –thus working at the data type level– while keeping the simplicity and generality of a GA. Such compromise has been achieved in the development of the following evolutionary algorithm.

[3]In some early experiments on the *Horse Colic* Problem –see (§9.2.2)– the chromosomic length reached 24,000.

6.4.1 Description of the algorithm

The Breeder Genetic Algorithm [Mühlenbein and Schlierkamp-Voosen, 93] is in midway between GA and ES. While in GA selection is stochastic and meant to mimic —to some degree— Darwinian evolution, BGA selection is named *truncation* selection, a deterministic procedure driven by the so-called *breeding* mechanism[4], an artificial selection method stating that only the best individuals —usually a fixed percentage τ of total population size— are selected and enter the gene pool to be recombined and mutated, as the basis to form a new generation[5]. Recombination/mutation operators are applied by randomly and uniformly selecting two parents until the number of offspring equals $\mu - q$. Then, the former q best elements are re-inserted into the population, forming a new generation of μ individuals that replaces the previous one. This guaranteed survival of some of the best individuals is called *elitism* whatever the EA. For the BGA, the typical value is $q = 1$. The BGA selection mechanism is then deterministic (there are no probabilities), extinctive (the best elements are guaranteed to be selected and the worst are guaranteed *not* to be selected) and 1-elitist (the best element is always to survive from generation to generation). Self-mating is always prohibited. This is a form of the comma strategy (μ, λ) employed by ES because the parents are not included in the replacement process, with the exception of the q previous best. Note that, given q (that is fixed) in the BGA only μ needs to be specified, since the number λ of offspring[6] can be calculated as $\lambda = \mu - q$. In other words, the BGA criterion is to generate $\lambda = \mu - q < \mu$ offspring to partially replace the old population, that is completed with the former q best. The full BGA procedure is depicted in figure 6.2, where τ is the truncation percentage for selection.

The other strong resemblance of the BGA to ES is that, unlike GA, the BGA uses a direct representation, that is, a gene *is* a decision variable (not a way of coding it) and its allele is the value of the variable[7]. An immediate consequence is that, in the absence of other conditionings as constraint handling, the fitness function equals the function to be optimized, $\Phi(\vec{x}) = F(\vec{x})$. In addition, in a BGA chromosome there are no additional variables other than the x_i, that is to say, the algorithm does not self-optimize any of its own parameters, as is done in ES and in some meta GA. Chromosomes are thus potential solution vectors $\vec{x}$ of n components, where n is the problem size, the number of free variables of the function to be optimized. This issue is of crucial importance because:

1. It eliminates the need of choosing a coding function (e.g., binary, Gray) to be used for all data types.
2. It allows the direct manipulation of different kinds of variables, other than real numbers (e.g., fuzzy quantities, discrete quantities, etc).
3. It permits the design of data-dependent genetic operators.

[4]This method is employed in livestock breeding.

[5]It is interesting to note that Tournament Selection in GA is a stochastic form of rank-based selection, of which truncation selection is the most used instance.

[6]In this case, $\lambda < \mu$ and the BGA mechanism deviates from that of ES.

[7]Of course, in a digital machine, we still have a coding, namely, that of the floating point representation but the decoding is transparent to the high level treatment of real numbers.

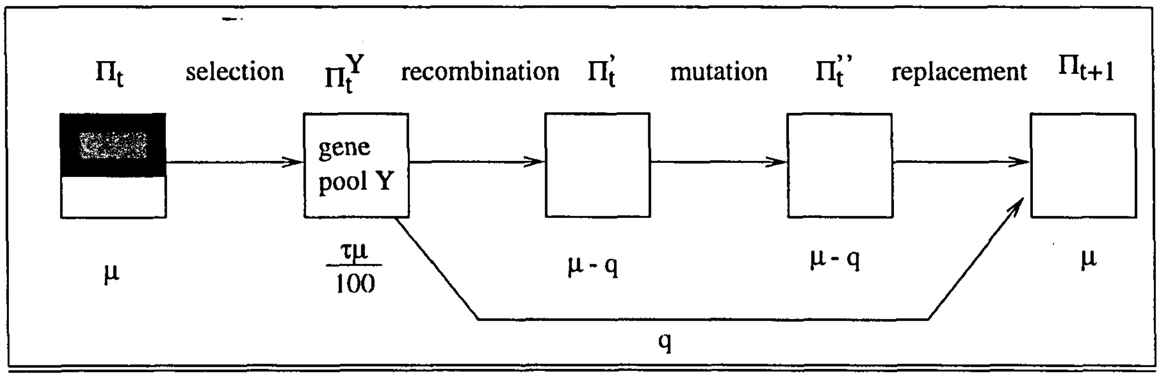


Figure 6.2: A scheme of the BGA procedure. Each box represents the population at different stages in the process to form a new generation. Notation on top of the boxes names the population at that point (see text) and the label from box to box (above the arrows) denotes operator sequencing (from left to right). The expressions at the bottom of the boxes indicate the population size at each step. Note how the final population size μ is formed by summing its two incoming values.

The common aspect of the BGA with an ordinary GA is the fact that both are mainly driven by recombination, with mutation regarded as an important but background operator intending to reintroduce some of the alleles lost in the population. This view is conceptually right for GA, because the cardinality of the alphabet used to code variables into the chromosome (the number of alleles per gene) is usually very small (two, in most cases). But in the case of algorithms that make use of real-valued alleles, like the BGA, mutation has to be seen in the double role of solution fine-tuner (for very small mutations) and as the main discovery force (for moderate ones). In fact, the initial BGA formulation readily acknowledged this superiority and remarked that it is the *synergistic* effect of their combined and iterated application what extracts the most from an EA [Mühlenbein and Schlierkamp-Voosen, 93]. What is more, in ES and EP, the roles are exchanged and mutation is the driving force, in the form of a very powerful self-adapting operator that tries to take the (unknown) relationships between variables into account, such that optimization is performed in several dimensions simultaneously. Because of this, for neural network weight optimization, the use of ES can convey a very high amount of parameters to be optimized. This is the main reason why we have considered the BGA over ES, although this last algorithm could certainly be of use. We will now briefly describe the different possibilities for the genetic operators. The reader is referred to [Belanche, 99d] for a detailed description.

6.4.2 Recombination

Any operator Ω combining the genetic material of the parents is called a recombination operator. In a BGA, recombination is applied unconditionally, $\Pr(\Omega) = 1$. Let $\vec{x} = (x_1, \ldots, x_n)$, $\vec{y} = (y_1, \ldots, y_n)$ be two selected gene-pool individuals $\vec{x}, \vec{y}$ such that $\vec{x} \neq \vec{y}$. Let $\vec{z} = (z_1, \ldots, z_n)$ be the result of recombination and $1 \leq i \leq n$. The following are some of

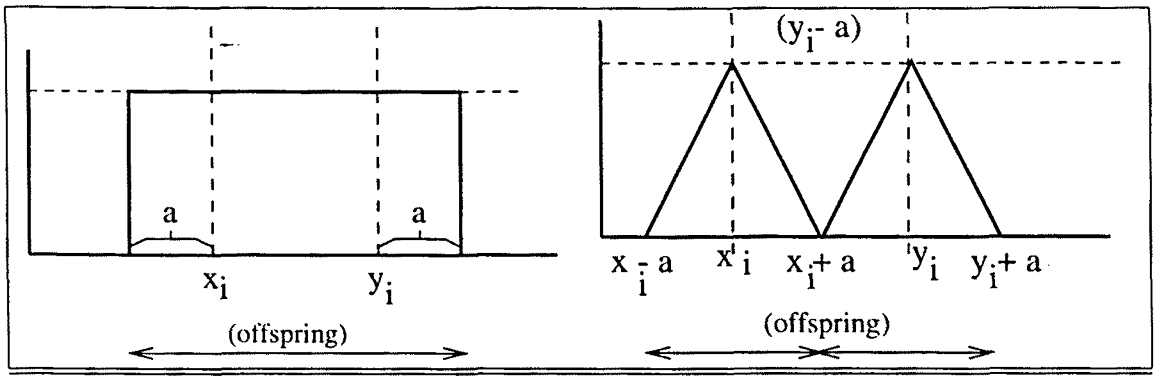


Figure 6.3: Potential zones for offspring and their probabilities. (Left) for the EIR operator, uniform *pdf* with $a = \delta|y_i - x_i|$. (Right) Bimodal *pdf* for the FR operator, where $a = e|y_i - x_i|$, for $0 < e < 1$ (shown for $e = 0.5$).

the more common possibilities to obtain an offspring $\vec{z}$:

1. Discrete Recombination (DR).

$$z_i \in \{x_i, y_i\} \qquad \text{(chosen with equal probability)} \tag{6.8}$$

2. Line Recombination (LR).

$$z_i = x_i + \alpha(y_i - x_i) \tag{6.9}$$

   with a fixed $\alpha \in [0,1]$. Typically, $\alpha = 0.5$.

3. Extended Intermediate Recombination (EIR).

$$z_i = x_i + \alpha_i(y_i - x_i) \tag{6.10}$$

   with $\alpha_i \in [-\delta, 1+\delta]$ chosen with uniform probability. The δ parameter expresses to what degree an offspring can be generated out of the parents's scope, the imaginary line that joins them in $\mathbb{R}$. More precisely, it works by controlling the maximum fraction $a = \delta|y_i - x_i|$ of the distance between parents where the offspring can be placed, either left to the leftmost parent or right to the rightmost parent -Fig. (6.3), left. A typical value for $\delta = 0.25$, although any non-negative real number is a potential value. Reasonable values should not exceed $\delta = 0.5$, since the bigger the δ, the more the effect of the parents is diminished in creating offspring. A method for dynamically setting its value is called *range*$_\delta$, and has been shown to have a remarkable effect in performance on a classical set of test functions [Belanche, 99d]. It works as follows:

$$z_i = y_i + \alpha_i(x_i - y_i), \qquad \text{with } x_i \geq y_i$$

   such that $\alpha_i \in [-\delta_i^-, 1+\delta_i^+]$ with uniform probability and,

$$\delta_i^- = \frac{y_i - r_i^-}{r_i^+ - r_i^-}; \qquad \delta_i^+ = \frac{r_i^+ - x_i}{r_i^+ - r_i^-} \tag{6.11}$$

This procedure assigns different values for the left (δ_i^-) and right (δ_i^+) limits of the interval from which α_i is to be selected, and does never generate a value outside the range $[r_i^-, r_i^+]$ for the variable i, an aspect not fulfilled by the other methods that otherwise has to be dealt with a posteriori.

4. Fuzzy Recombination (FR), introduced in [Voigt, Mühlenbein and Cvetkovic, 95]. This operator basically replaces the uniform *pdf* (probability distribution function) by a bimodal one, where the two modes are located at x_i and y_i, the two parents, that is $\Pr(z_i) \in \{\Pr_{x_i}(z_i), \Pr_{y_i}(z_i)\}$ thus favouring offspring values close to them, and not in any intermediate point with equal probability, as with previous operators. The label "fuzzy" comes from the fact that the two parts $\Pr_{x_i}(t), \Pr_{y_i}(t)$ of the probability distribution resemble fuzzy numbers (triangular in the original formulation), such that they fulfill the general conditions (where $y_i \geq x_i$):

$$x_i - e|y_i - x_i| \leq \ t \ \leq x_i + e|y_i - x_i|$$

$$y_i - e|y_i - x_i| \leq \ t \ \leq y_i + e|y_i - x_i|$$

stating that the offspring t lies in one (or both) of the intervals, being $e > 0$ the fuzzy number's spread, the same for both parts. The favour for offspring values near the parents is thus stronger the closer the parents are. This operator is depicted in Fig. (6.3), right. In the simplest case, assuming $e = 0.5$ –that is, the two parts meet at the median and this point has zero probability, as in the figure– an offspring z_i is obtained with probability

$$\Pr(z_i) = \mathrm{BT}(z_i)\{e|y_i - x_i|, x_i, e|y_i - x_i|, y_i\} \tag{6.12}$$

where $\mathrm{BT}(t)$ is a Bimodal Triangular *pdf*, defined by the four notable points: the two modes and their left-right spread (these are equal in our case).

6.4.3 Mutation

A mutation operator Ψ is applied to each gene with probability $\Pr(\Psi) = \frac{1}{n}$ so that, on average, one gene is mutated for each individual. Let $\vec{z} = (z_1, \ldots, z_n)$ denote the result of mutation of an individual $\vec{x}$. The elements of $\vec{z}$ are formed as follows:

1. Discrete Mutation (DM).

$$z_i = x_i + sign \cdot range_i \cdot \delta \tag{6.13}$$

with $sign \in \{-1,+1\}$ chosen with equal probability, $range_i = \rho(r_i^+ - r_i^-)$, $\rho \in [0.1, 0.5]$ and

$$\delta = \sum_{i=0}^{k-1} \varphi_i 2^{-i}$$

where $\varphi_i \in \{0,1\}$ from a Bernouilli probability distribution where $\Pr(\varphi_i = 1) = \frac{1}{k}$. In this setting $k \in \mathrm{N}^+$ is a parameter originally related to the *precision* with which the optimum was to be located, a machine-dependent constant. Modern machines, capable of `double` precision, would in principle allow for higher values of k (e.g. 24, 32) than those traditionally used (e.g. 8, 16). In practice, however, the value of k is related to the *expected* value of mutation steps: the higher k is, the more fine-grained is the resultant mutation operator [Belanche, 99d]. The factor ρ is the *range ratio*, related to the *maximum* step that mutation is allowed to produce as a ratio of variable range. This scheme favours small values but cannot generate all possible representable points[8], but only a discrete amount and prefers small values in an approximately (on average) logarithmic ($\log_2$) scale, always up to a precision of $range_i \cdot 2^{-k+1}$.

2. Continuous Mutation (CM). Same as DM but with

$$\delta = 2^{-k\beta} \tag{6.14}$$

where $\beta \in [0,1]$ with uniform probability.

6.4.4 The BGA as a neural network trainer

A concise review on the generic use of evolutionary learning algorithms for neural optimization was given in (§2.1.8). To the best of our knowledge, the BGA has only been used for some specific neural optimization tasks or application examples:

- In [De Falco *et al.*, 97] a hybrid methodology, in which the BGA is used to find an adequate architecture is combined with a derivative-based method (DBM) –Back-Propagation, in this case– in an application example concerned with a non-linear system identification task. Also, the BGA is replaced by a GA and the results are compared. The combination (BGA, DBM) is found to be superior to (GA, DBM).
- In [De Falco *et al.*, 98] the numerical optimization problem is left to the BGA, given a fixed neural architecture, to solve the Mackey-Glass time series. The results are compared to those obtained by a DBM (a globally enhanced Back-Propagation). The DBM is found to be superior to the BGA and this, in turn, superior to the GA.
- In [Zhang and Mühlenbein, 93], a BGA variant called BGP or Breeder Genetic Programming is introduced. It is basically a BGA with variable-length chromosomes and a

[8]By this we mean *machine*-representable. We assume that there is a machine-dependent floating point constant ϵ equal to the smallest positive representable number in a chosen precision. For example, in our machines, such number for `double` precision is $\epsilon \approx 2.22 \cdot 10^{-16}$.

next-ascent hill-climbing mutation mechanism to improve on the individuals found. All the work is left to the BGP and the search is biased to solutions representing minimal networks. The two tasks addressed are binary problems and the focus is on the ability of the system to find the minimal known solutions.

In all, the results point the BGA to be markedly superior to traditional GA techniques although still inferior to DBM, especially if the latter are enhanced by a means to escape from local optima. However, these are initial studies and the application of the BGA to neural network training does not constitute yet a widely spread methodology. It is our belief that EA -particularly, the continuous ones- are in need of specific research devoted to ascertain their general validity as alternatives to DBM in neural network optimization. Theoretical as well as practical work oriented to tailor specific EA parameters for this task, together with a specialized operator design should pave the way to a fruitful assessment of validity, both in terms of effectiveness and efficiency.

As an initial step toward this end, in the following the BGA is used for the numerical optimization task of a specific classification problem, the Pima Indians database, often used for neural benchmarking [Prechelt, 94]. The intention is not to solve this problem to a full, or achieve the best possible solution, but to use it to investigate its possibilities in the place of a DBM, in a task known not to be an easy one [Zheng, 93]. Specifically, the experiments are to address the following issues:

1. Determine whether and how can the BGA -as a particular EA technique- cope with this problem;
2. Explore the way different configurations of the algorithm (choice of genetic operators and their parameters) affect its performance, investigating the existence of configurations potentially better suited for neural network training;
3. With the experience gained, select the best of those settings and re-run the BGA with the purpose of finding good solutions;
4. Compare the solutions found by the BGA, in case they are found to be within reasonable values, to those found by a powerful DBM.

6.5 An investigation in BGA performance

6.5.1 Experimental design

The Pima Indians Database has been taken from the Proben Archive [Prechelt, 94]. It consists of 768 examples of a two-class classification task, a positive or negative diagnose of diabetes. There are 8 input real variables, normalized to lie in the interval $[0, 1]$. Class membership comes coded as a 1-out-of-2 scheme. The number of training cases used is half this quantity -actually the first half, which happens to be class-split into 145 (37.76%) and 239 (62.24%), totaling $p = 384$ training cases. The number of hidden units chosen is fixed to $h_1 = 6$, that is, a neural architecture $8-6-2$ is the one to be used in all of the experiments performed. A MLP with scalar product plus bias, and the logistic as activation, is used. This means that a total of 68 free parameters are to be optimized. The activation function is the logistic (3.9) for all the hidden units, and the identity for the output ones. The cost function to be optimized (equal to the fitness) is the square error accumulated over all training examples. All the weights are let to vary within the (possibly too generous) real interval $[-10, 10]$.

Note at this point that the BGA (or any other algorithm for that matter) is likely to find different results for different numbers of hidden units, basically due to two compromising reasons:

1. A different form for the function realized by the neural network is going to approximate a training set more easily the higher is h_1, in theory regardless of the training algorithm.
2. Given a fixed size for the input and output spaces, an increasing number h_1 of hidden units means a linearly increasing number of free parameters and any training algorithm is likely to have more trouble optimizing a function in a higher number of dimensions.

Provided we are not concerned in these experiments with selecting the best architecture, but on assessing BGA performance for the numerical optimization task with varying learning parameters, h_1 is kept constant for all the experiments. An important hypothesis is then that should we select a different h_1, the overall results would have been different, but only in *absolute* terms. In our case, the value of $h_1 = 6$ has been chosen deliberately small -although not to the point of being too restrictive- so that a learning algorithm has a difficult time accommodating the rather large training set of cases. This way, differences in performance are to show themselves more markedly. In what concerns the BGA, we are primarily interested in addressing the following issues:

- Choice of mutation operator Ψ and its parameters ρ and k.
- Choice of recombination operator Ω and its parameter δ (only for EIR).
- Determination of the truncation threshold τ.
- Study of performance as a function of population size μ.

The stopping criterion is based on the number of fitness evaluations permitted (given by the variable `FFEvals`). In particular, given a finite number of `FFEvals`, the algorithm will stop each run whenever $\lfloor \frac{\texttt{FFEvals}}{\mu} \rfloor$ generations are reached. This stopping criterion allows to compare different general settings in a fair way, since, for example, a smaller population will be allotted more generations, but always keeping the number of fitness evaluations in similar values. For each configuration, a number of independent runs are performed -denoted by NRuns- keeping track of the mean and best solutions found. For all of the experiments, unless otherwise stated, `FFEvals`=40,000 and NRuns=20. Elitism is set to $q = 1$. In accordance with the studies performed on some classical optimization problems [Belanche, 99d] the following representative subset of possibilities is explored:

Mutation operator $\Psi \in \{\text{CM, DM}\}$; parameters $\rho \in \{0.1, 0.5\}$, $k \in \{8, 16, 24, 32\}$. Number of different configurations: 16.

Recombination operator $\Omega \in \{$DR, LR (α=0.5), EIR ($\delta \in \{0, 0.15, 0.25, 0.35, 0.45\}$), EIR ($\delta = range_\delta$), FR (e=0.5)$\}$ and a simple *random* recombination (explained below). Number of different configurations: 10.

Truncation threshold $\tau \in \{5, 8, 11, \ldots, 50\}$. Number of different configurations: 16.

Population size $\mu \in \{2, 4, 6, \ldots, 100\}$. Number of different configurations: 50.

This experimental design is suboptimal because not all the possible combinations of mutation with recombination or with values of τ are tested. As we shall see, this number is too high to allow a full study to be performed and, furthermore, it is our belief that many of the combinations can readily be discarded *a priori* by a smaller but more effective experimental design. Also, even with the full results, the conclusions could not in any case be general ones since there is probably no configuration that is optimal for every conceivable (even reasonably) network optimization task, not even if, as in this work, the study is circumscribed to a specific search algorithm. Clearly, an exhaustive search over configurations is computationally infeasible, even limiting ourselves to a finite number of possibilities for every issue tackled (in our case, as in can be readily checked, this number amounts to 128,000). Hence, instead of performing an exhaustive (in a sense, multiplicative) number of experiments, a greedy (additive) strategy can be applied in the following way:

1. Set to their values all parameters that are constant across all the experiments.
2. Set an initially *standard* setting for the issues to be explored, except for the first.
3. Select a suitable ordering for the issues to be explored;
4. Perform the experiment for the *first* of the issues according to the order chosen. Determine the *best* (or better two) setting, and hold it constant for the rest of the experiments, replacing the old value;
5. Perform the experiment for the *next* of the issues according to the order chosen. Determine the *best* (or better two) setting, and hold it constant for the rest of the experiments, replacing the old value;
6. If not done (all issues explored), go to step 5.

The strategy is then to perform several consecutive experiments that are kept simple and use the knowledge found up to the point for the remaining experiments. By proceeding this way, the number of configurations is to shrink to a minimum of 92, assuming that only the best setting is selected for each issue. If the better two are kept, this number will be between 92 and 184 (the double), depending on the ordering. In any case the number of configurations is now manageable. The order chosen is that of the previous description. First: selection of mutation operator; next: selection of recombination operator, truncation threshold and finally population size. There is no strong reason behind this ordering: it has been the case because we are primarily interested in finding out whether there are mutation and recombination settings generally better suited for neural network training, and *then*, to explore their performance across different truncation thresholds and population sizes.

6.5.2 Experimental results

Experimental results on mutation

For this experiment, $\mu = 100$ and NRuns=10 for each setting of the algorithm shown in Fig. (6.4), where the choice for recombination, Ω = EIR ($\delta = 0.25$), fairly standard, has been selected. The results are presented as follows. Two tables are given, separated in continuous (CM), Table (6.1)), and discrete (DM), Table (6.2), mutation. For each configuration (ρ, k) the *average* and *best* solution found throughout the NRuns are kept. Instead of giving a separate entry for each such configuration, some simple additional computations are performed to compact the information. The entries in the column for ρ are obtained averaging out the results **forall** k **in** {8, 16, 24, 32}. Similarly, the entries in the column for k are those obtained averaging out **forall** ρ **in** {0.1, 0.5}. By proceeding this way, one has to deal with less information and the obtained values are more representative. For instance, the entries for ρ are averages over 20 runs and those for k over 40 runs. Top value shown is the average and bottom value (in parentheses) is the best result.

```
Procedure Mutation-Test ()
{
  $(\mu, \tau)$ := (100, 25);
  $\Omega$ := EIR ($\delta = 0.25$);
  forall $k$ in {8, 16, 24, 32}
    forall $\rho$ in {0.1, 0.5}
      forall $\Psi$ in {CM, DM}
        BGA ($\mu, k, \rho, \Psi$, NRuns, FFEvals, $\Omega$, $\tau$);
}
```

Figure 6.4: Mutation-Test Algorithm pseudocode.

- ρ -		- k -			
0.1	0.5	8	16	24	32
130.0 (120.1)	**124.7 (115.7)**	**123.4 (114.9)**	**124.9 (116.8)**	131.4 (120.2)	129.8 (119.6)

Table 6.1: Results for Continuous Mutation (CM). See text for an explanation of entries. Markedly good results are boldfaced.

All figures correspond to the direct fitness values obtained, corresponding to the accumulated square error, to be minimized. This value is a monotonic function of the usually reported error and computationally cheaper. To get the actual root mean square error (RMSE), just divide each entry by 768 (number of patterns times number of outputs) and take the square

$-\rho-$		$-k-$			
0.1	0.5	8	16	24	32
130.2 (117.3)	**127.1** (117.1)	**126.1** (118.6)	128.6 (120.1)	131.1 **(115.3)**	128.6 **(114.9)**

Table 6.2: Results for Discrete Mutation (DM). See text for an explanation of entries. Markedly good results are boldfaced.

root of the result. The results are of course non-conclusive because the number of runs is not too high and the explored possibilities are also a small –though representative– sample. However, by looking at Tables (6.1) and (6.2) several aspects are noteworthy.

- Firstly, the results for CM are very neatly defined: performance is superior for $\rho = 0.5$ over $\rho = 0.1$, and for $k \in \{8,16\}$ over $k \in \{24,32\}$, both on average and for the best value obtained. This corresponds to the boldfaced rectangular region in Table (6.1).
- For DM, the results are less vigorously defined, but in general accordance. It can be checked that again $\rho = 0.5$ is superior to $\rho = 0.1$ and performance for $k \in \{8,16\}$ is *at least as good* as $k \in \{24,32\}$. This last point, however, is not fulfilled by the absolute best values found; though these are less representative quantities than the averages, this behaviour is in need of an explanation.
- The overall results (high values of ρ combined with low values of k) are also in strong accordance with previous studies on non-trivial functions [Belanche, 99d].
- Secondly, CM seems to be slightly superior to DM, a fact that shows itself especially for the best results, that is, for $\rho = 0.5$ and $k \in \{8,16\}$. For $k \in \{24,32\}$, CM is better but only marginally and in any case the results are worse than those for $k \in \{8,16\}$.

By looking at the average results for individual settings (not shown) we find that performance for CM (0.5, 8) is equal to 122.2, while that for CM (0.5, 16) is 121.5, and hence this last setting is the one finally selected, although we conclude that a value of $k = 8$ is an almost equally valid choice.

Experimental results on recombination

In the case of recombination operators, up to ten different settings are tested, as follows: DR, LR with $\alpha = 0.5$, EIR with $\delta = 0, 0.15, 0.25, 0.35$ and 0.45, EIR with $range_\delta$ and FR with $e = 0.5$. The mutation operator is fixed to CM with $\rho = 0.5, k = 16$, as a result of the previous experiment on mutation. The BGA procedure in the algorithm of Fig. (6.5) is run for this experiment with $\mu = 100$ and NRuns=20 for each setting. As an additional reference, a completely uninformed recombination operator is included, which simply yields an offspring generated randomly within the range (not taking into account the parents's information). This operator is denoted RR (Random Recombination).

The results are presented in a single table for ease of reading –Table (6.3)– summarizing the information for all of the recombination variations tested. For each configuration, the

```
Procedure Recombination-Test ()
{
  $(\mu, \tau) := (100, 25)$;
  $\Psi :=$ CM $(\rho = 0.5, k = 16)$;
  forall $\Omega$ in { RR, DR, LR $(\alpha = 0.5)$,
              EIR $(\delta = 0, 0.15, 0.25, 0.35, 0.45)$,
              EIR $(range_\delta)$, FR $(e = 0.5)$ }
      BGA $(\mu, F_i, k, \rho, \Psi,$ NRuns, FFEvals, $\Omega$, $\tau)$;
}
```

Figure 6.5: `Recombination-Test` Algorithm pseudocode.

average and best (in parentheses) solutions found throughout the NRuns are shown. Several points are noteworthy:

1. The computed average performance of EIR across a fixed $\delta \in \{0.15, 0.25, 0.35, 0.45\}$ is equal to 123.6. This figure compares favourably to those for the other operators (RR, DR, LR and FR), not counting EIR ($range_\delta$).

2. FR is somewhat in between on average. Note that its best result (112.8) is very similar to that for the averaged EIR with fixed δ (112.7). It seems then that this last operator (EIR) can yield potentially better results, provided we find the right δ. In any case, averaged EIR mean behaviour (123.6) is better than FR (127.4).

3. With respect to EIR ($range_\delta$), it is clearly the best setting on average (117.6) though it achieves only middling results for the best value. A closer study on performance with a greater number of runs should give more definite results. In this line of thinking, it is interesting to observe that EIR ($\delta = 0.45$) has the absolute best result, although its mean performance is the worst across all EIR. All this confirms that the best values found are subject of much greater variability and cannot be readily predicted by the average results, this last quantity being the one to be primarily taken into account to judge among different settings.

RR	DR	LR	EIR	EIR	EIR	EIR	EIR	EIR	FR
		$\alpha = 0.5$	$\delta = 0$	$\delta = 0.15$	$\delta = 0.25$	$\delta = 0.35$	$\delta = 0.45$	$range_\delta$	$e = 0.5$
399.5 (193.6)	138.2 (114.4)	129.8 (119.5)	124.5 (114.0)	121.7 (112.1)	124.2 (115.8)	122.5 (111.2)	124.9 (110.6)	117.6 (114.8)	127.4 (112.8)

Table 6.3: Results for the different settings of recombination operators. Each entry shows the average and best results (in parentheses) across NRuns=20 runs.

To sum up, note that the average performance of EIR is *always* better than the rest (RR, DR, LR and FR), for all δ. This shows EIR as a robust operator, and notably when using the method $range_\delta$.

Experimental results about the truncation threshold

Sixteen different values for the truncation threshold $\tau = \{5, 8, 11, \ldots, 50\}$ are tested, as samples of the full (discrete) interval $[5, 50]$. As before, the mutation operator is fixed to CM with $\rho = 0.5, k = 16$. As a result of the previous experiment on recombination two settings, EIR ($\delta = 0.35$) and EIR ($range_\delta$), are to be used. The latter is the one with the best average performance, while the former -besides being a representative of EIR with a fixed δ- has the best all-round performance (crudely defined as the mean of average and max). The BGA procedure -the algorithm in Fig. (6.6)- is run for this experiment with $\mu = 100$ and NRuns=20 for each setting. The results are presented graphically in Fig. (6.7) for EIR ($\delta = 0.35$) (left) and EIR ($range_\delta$) (right). As usual, for each configuration, average and best performance throughout the NRuns are shown.

```
Procedure Truncation-Test ()
{
  $\mu := 100$; $\Psi :=$ CM ($\rho = 0.5$, $k = 16$);
  forall $\Omega$ in {EIR ($\delta = 0.35$), EIR ($range_\delta$)}
    forall $\tau$ in $\{5, 8, 11, \ldots, 50\}$
      BGA ($\mu, F_i, k, \rho, \Psi$, NRuns, FFEvals, $\Omega$, $\tau$);
}
```

Figure 6.6: Truncation-Test Algorithm pseudocode.

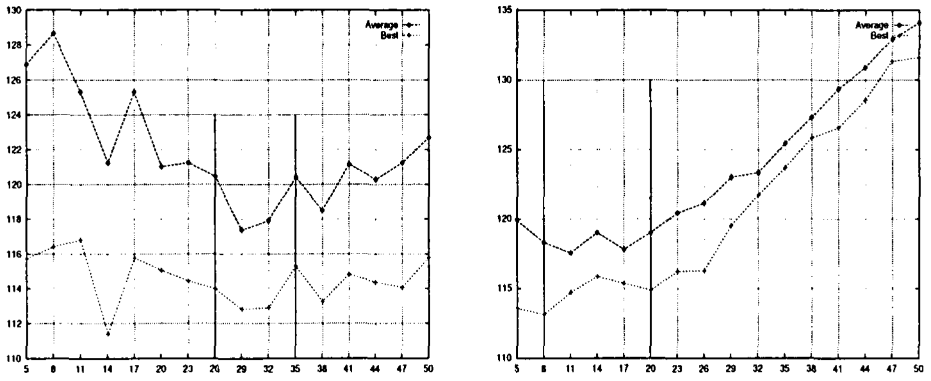


Figure 6.7: Results as a function of truncation parameter τ. Left: EIR ($\delta = 0.35$). Right: EIR ($range_\delta$). Each point is the result of NRuns=20 runs.

The most prominent observation is the radically different behaviour of both settings: whereas for EIR (δ=0.35) performance firmly increases up to a bending zone (about [26, 35], peaking[9] at 29 and 32) and then begins to decline, for EIR ($range_\delta$) the best region appears very soon, in [8, 20], peaking at 11 and 17. These zones are marked in the plots of Fig. (6.7) (left and right) by two vertical bars. In between (in [20, 26]) there is a transition zone, where

[9]The peak points are orientative because not all points have been sampled. They should be read as "around" marks.

performance for both is in the [118, 122] mark. More precisely, the crossing lies in [23, 26], where both operators are in the [120, 122] mark. This comparative behaviour is clearly seen in Fig. (6.8), where average results for the two operator settings are plotted together. It is also clear from this last plot that both configurations achieve comparable average results, although for quite opposite values of τ.

Our motivation here is clear: no single value is going to be always the best suited, because the particular threshold for truncation is likely to be tailored to a specific problem; instead, it makes more sense to ascertain a generally correct interval and then choose a value or values inside it. In our case, the two intervals are very definite but different for the two settings. For the sake of avoiding duplicate experiments or favour one of the two settings, the point of equal performance is selected: the midpoint of [23, 26] being 24.5, we take 25. This approximate cross-point value happens to be consistent with usual proposals (usually $\tau \in [20, 25]$) in the neural network context [De Falco *et al.*, 98], [Zhang and Mühlenbein, 93].

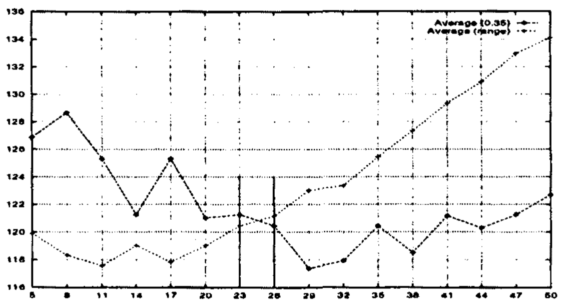


Figure 6.8: Average results for the two recombination settings. The cross-point is around 25.

Experimental results varying the population size

Once the best configurations for mutation and recombination and a reasonable value for the truncation threshold have been set, it is of interest to perform an experiment on how the population size μ affects performance. In this line of argument, it is very clear that no single value for μ is going to be generally "best", even for a particular task. Rather, we are concerned with studying the curves of performance as a function of μ, keeping other parameters constant. It is of added interest that all the experiments are to be allotted the same total number of fitness evaluations (given by FFEvals), thus allowing a truly fair comparison. Two questions arise: is a bigger population size going to give (significatively) better results, provided it is allowed to resort on the same number of fitness evaluations? Is there any clear trend in performance as a function of μ?

To give an answer, fifty different values for the population size are tested, sampling the discrete interval [2, 100] of even numbers, to get a smooth transition between consecutive results. To keep the computational burden within reasonable values, the limit FFEvals is set to 4000, thereby allowing for 2000 generations for $\mu = 2$ and only 40 for $\mu = 100$. In practical applications, this would constrain the absolute performance figures to unacceptably low values. However, in this experiment (as in the majority of experiments in this study) we are primarily interested in the *relative* performance with varying μ. In a sense, we are going to explore how does the algorithm behave *in the first 4,000* evaluations of what would normally have been a longer execution. It has been observed [Zhang and Mühlenbein, 93] that the typical performance curves for a BGA follow the trends shown in Fig. (6.9) (depending on problem difficulty). Hence the initial performance is able to illustrate how the algorithm is going to behave for a particular setting. Expected variations due to specially lucky (or unlucky) runs are to be mitigated by computing the average over the NRuns runs.

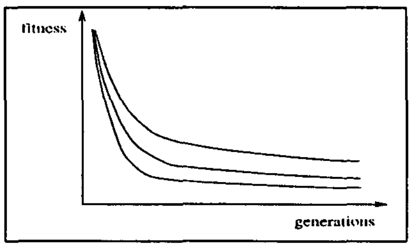


Figure 6.9: Typical (idealized) BGA performance curves.

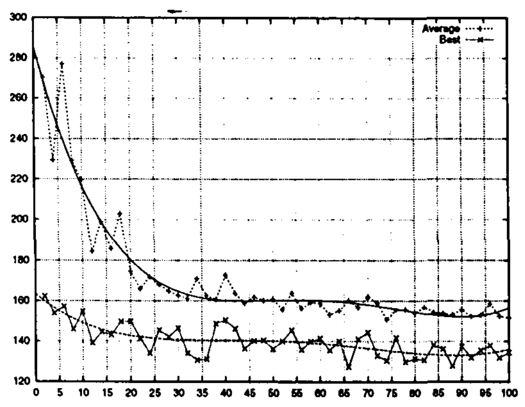
```
Procedure PopulationSize-Test ()
{
  $\tau$ := 25; $\Psi$ := CM ($\rho = 0.5$, $k = 16$);
  forall $\Omega$ in {EIR ($\delta = 0.35$), EIR ($range_\delta$)}
    forall $\mu \in \{2, 4, 6, \ldots, 100\}$
      BGA ($\mu, F_i, k, \rho, \Psi$, NRuns, FFEvals, $\Omega$, $\tau$);
}
```

Figure 6.10: PopulationSize-Test Algorithm pseudocode.

Operators are set to CM with $\rho = 0.5, k = 16$ for mutation, and EIR ($\delta = 0.35$), EIR ($range_\delta$) for recombination, with $\tau = 25$ and NRuns=20 for each setting. The procedure is that of algorithm in Fig. (6.10). The results are again presented graphically, in Fig. (6.11, left) for EIR ($\delta = 0.35$) and (right) for EIR ($range_\delta$). For each configuration, average and best performance curves across the NRuns are shown. In addition, a quartic function (polynomial of degree 4) is fitted to the data and plot.

Both sets of curves are similar in that they exhibit the same trend: a sudden initial decrease (more marked for the average performance) up to a turning point from which further improvements are slow but consistent. This clear tendency to better performance answers both questions affirmatively: the first one because of the trend, and the second one because of its smoothness. The bigger μ, the better the results even though the effective number of generations is getting lower and lower.

More precisely, the interval [35, 40] seems to be the turning point in both cases, then there is a flat plateau ([40, 65]) and from 65 on, the curve decreases in both cases, reaching the lowest zone at [85, 95] for EIR ($\delta = 0.35$) and at [60, 80] for EIR ($range_\delta$). It finally seems to slightly raise again, possibly because the number of FFEvals is too small for $\mu = 100$. This collective behaviour is more clearly seen in Fig. (6.12, left), where average curves for



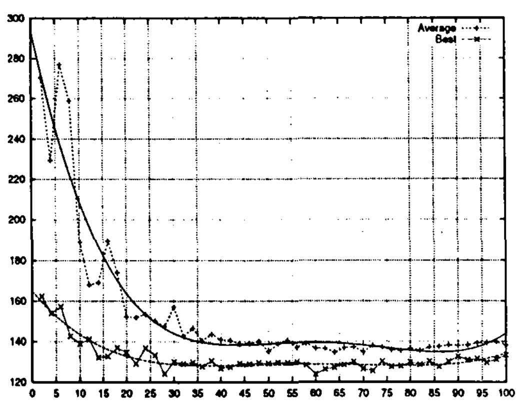
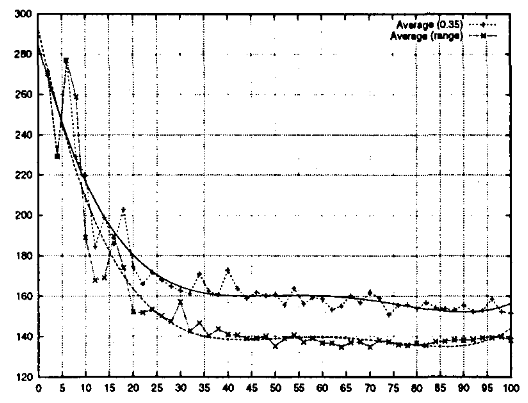


Figure 6.11: Average and Best BGA performance as a function of μ. Left: for EIR ($\delta = 0.35$). Right: for EIR ($range_\delta$). Each point is the result of NRuns=20 runs.



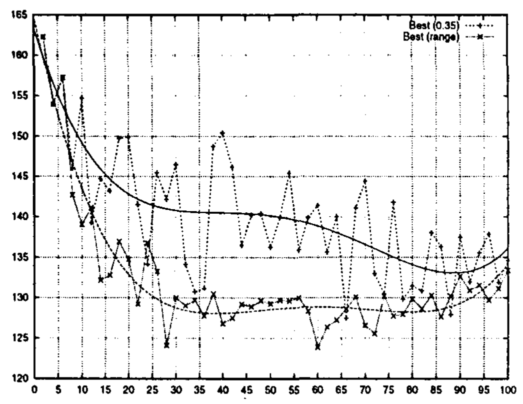


Figure 6.12: Compared BGA performance as a function of μ. Left: Average. Right: Best results. Each point is the result of NRuns=20 runs.

both settings are plotted side by side. Both average curves have nonetheless similar trends, showing the slight increase at the end. It is also clear from these plots that the curve for EIR ($range_\delta$) is consistently better than that of EIR ($\delta = 0.35$). The same arguments are valid for the Best result curves (Fig. 6.12, right). In this case, the increase at the end is more marked.

A comparison of performance

With the knowledge gained so far, it is of interest to perform a further batch of experiments with the clear intention of finding relevant solutions, and to compare them to those found by DBM. To this¯end, we select the two settings that have been used so far, but this time allowing the algorithm for FFEvals = 100,000 across NRuns = 10. The two recombination settings are used with the following parameters: ($\mu = 90$, $\tau = 30$) for EIR ($\delta = 0.35$) and ($\mu = 50$, $\tau = 18$) for EIR ($range_\delta$). As can be seen, these are rather temptative values chosen *inside the regions* of best performance, following the results of (§6.5.2).

Algorithm	- MSE -	
	Average $\pm \frac{\sigma}{\sqrt{n}}$	Best
BGA (EIR, $\delta = 0.35$)	0.1461 ± 0.0016	0.1354
BGA (EIR, $\delta = range_\delta$)	0.1471 ± 0.0012	0.1407
Annealing + ConjGD	0.1409 ± 0.0053	0.1134

Table 6.4: Comparative results found by the three optimization algorithms.

These two representative BGA configurations are to be compared to a DBM, in as much equal conditions as possible, though this is in general extremely difficult. The method chosen is a powerful combination of two classical optimization methods. It consists of a Conjugate Gradient descent coupled to a Simulated Annealing schedule [Ackley, 87]. This hybrid method is allowed to resort to 10 restarts to match the 10 runs of the BGA, and it optimizes the same cost on the same data set, using an identical network architecture. The results are shown in Table (6.4). This time, data for MSE are directly shown in the format Average $\pm \frac{\sigma}{\sqrt{n}}$, where σ is the standard deviation and n the number of runs, along with the best solution found. There are some interesting observations:

1. The average values found by the two BGA and the Annealing + Conjugate Gradient are very close indicating a comparable average performance, which is by itself very notable, given the sophisticated nature of the chosen DBM. However, the latter method finds an extremely good solution not matched by any BGA setting. In spite of this, the comparable average behaviour and the markedly lower variability shown by the BGA indicate a general feasibility for the task, perhaps contrary to what one could pessimistically expect of an evolutionary algorithm.

2. The fact that EIR ($range_\delta$) gives poorer results is a little deceptive in light of its previous results. This operator is in need of a thorough experimental investigation, which falls beyond the scope of this Thesis. However, the lower deviations w.r.t. the DBM may be an indication that it has not been used in or near its optimal population sizes and truncation thresholds. This could also apply to EIR ($\delta = 0.35$).

3. In what regards the duration of training, the DBM took about six times (actually six hours) the time used by a BGA execution (counting all the NRuns), on a shared SUN$^{\text{TM}}$ Ultra-60 System.

The obtained performance curves are shown in Fig. (6.13) for the two selected recombination settings. Each of the curves traces the evolution of the best solution found up to a given generation. They can be compared against those in Fig. (6.9). By looking at them, it

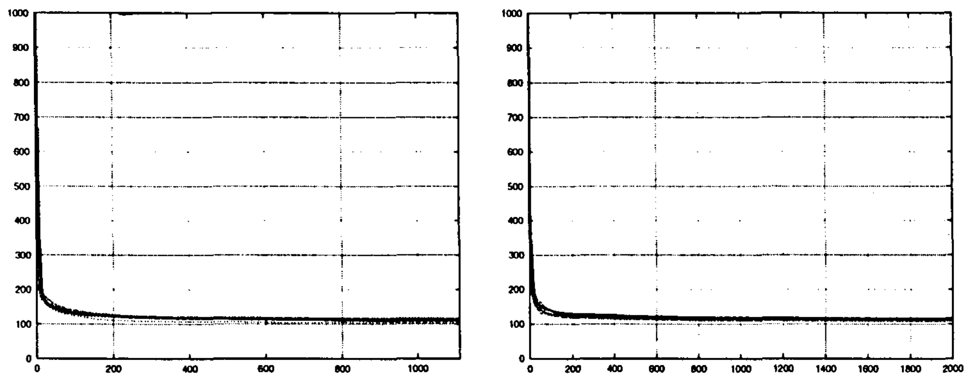


Figure 6.13: Actual BGA performance curves. Left: for EIR ($\delta = 0.35$). Right: for EIR ($range_\delta$). The x-axis shows the generations and the y-axis shows the fitness. Note the different number of generations due to the different values of μ used. In each case, all ten curves are plotted, illustrating in both situations the very low variability.

can be seen how the BGA quickly (in a couple of hundred generations) finds relatively good solutions, then stabilizes and continues to minimize the cost but at a very low pace.

6.5.3 Summary of findings

We are now in a position to summarize the main findings of the experiments. For some aspects, specific configurations (one or two) are shown to be specially better while, in other cases, a range of values can be proposed. The intention is then to provide an interval from which a careful problem-dependent experimental setup could draw samples. For instance, in our case study, it is improbable that we have chosen the best possible value for τ or μ in the comparison to the DBM, but the chances that we are in a meaningful interval are high and a specialised search for practical solutions to this problem would surely improve on the solutions temptatively obtained. Recall that the same τ has been used for two radically different recombination settings to obtain the results on μ. In a real application, separate experiments should be performed. A summary of recommended BGA operators and parameters follows:

Mutation operator: Continuous Mutation with parameters $\rho = 0.5$ and $k \in \{8, 16\}$. Discrete Mutation could also be a choice, although not for the problem studied. For both operators, this parameter setting is in general favour of bigger mutation steps.

Recombination operators: Extended Intermediate Recombination (EIR) with a high $\delta = 0.35$ or $\delta = 0.45$ (good behaviour on average and w.r.t. the best result found) and EIR with dynamic δ calculation (best average behaviour).

Truncation threshold τ: found to be approximately $\tau \in [26, 35]$ for EIR ($\delta = 0.35$) and $\tau \in [8, 20]$ for EIR ($\delta = range_\delta$) for $\mu = 100$. This value is likely to depend on μ.

Population size:¯for the selected τ of 25, performance reaches the optimum zone in the [85, 95] interval for EIR (0.35) and within [60, 80] for EIR ($range_\delta$). Again, the interaction between τ and μ prevents from drawing far-sighted conclusions, but the definite performance trends found for both parameters and their relationship are very clear and deserve a closer attention.

6.6 Extension of the BGA to heterogeneous network training

The manipulation of each kind of heterogeneous weight is carried as follows:

Real-valued weights are directly treated as such, initialized at random within a pre-declared range, and recombined and mutated with the operators described in (§6.4.2) and (§6.4.3). Values eventually generated outside the boundaries are clipped.

Ordinal weights are represented as positive natural numbers in the interval $[1, m]$ -following the discussion in (§4.3)- an initialized at random within the interval. For recombination, there are three possibilities, which mimic the real-valued operators: discrete recombination (6.8) (which is generally valid but ignores the order), line recombination (6.9) (which respects the order), and extended intermediate recombination (6.10) (idem, but needs an extra parameter to be set). Some preliminary investigations lead to the choice of line recombination with fixed $\alpha = 0.5$, that is, the *median* of the parents. In case of an odd number of elements between the parents, the offspring was selected with equal probability among the two choices.

Mutation involves an *increase* (to the immediately following value w.r.t. the linear order relation) or a *decrease* (idem, but in the opposite sense), and the decision is taken with equal probability. Values eventually generated outside the boundaries are wrapped around.

Nominal weights are also represented as positive natural numbers in the interval $[1, m]$, where m is the cardinality, but no order relation is assumed. They are initialized at random within the interval. The clear choice for recombination in this case is *discrete* recombination, being the only one ignoring any underlying order. Mutation is realized by switching to a new value in the interval, with equal probability.

Fuzzy quantities. The BGA extension to handle fuzzy numbers is given by a tuple of reals (three in the general case, only two if the chosen representation is symmetric triangular or Gaussian). Linguistic variables are described by their anchor points on the abscissa axis (four in the case of trapezoidal membership functions). Actually, the two spreads are taken as offsets to simplify the manipulation.

The BGA can in a sense be seen as a *fuzzy* BGA because it directly deals with fuzzy quantities, as long as the algorithm manipulates the involved variables as representing a unique entity at all levels (e.g. in initialization, recombination or mutation).

The initialization of **fuzzy numbers** is as follows: the mode is assigned a random value within the pre-declared range. The fuzziness involves the generation of a new random value within the same range, which is then manipulated in accordance with the way fuzzy numbers are constructed for the corresponding input (e.g., a fixed percentage). This is so since the weights are not restricted to have the same kind of fuzziness as the inputs (and that is why two real-valued numbers are needed in their representation).

Recombination of fuzzy numbers is developed as the corresponding extension of the operators for real-valued quantities. In particular, for the EIR operator, the mode is obtained following (6.10) (involving the selection of a δ), and the spread is computed using (6.10) with the *same* α. This makes sense whenever the spread is proportional to the mode. Fig. (6.14) provides an example.

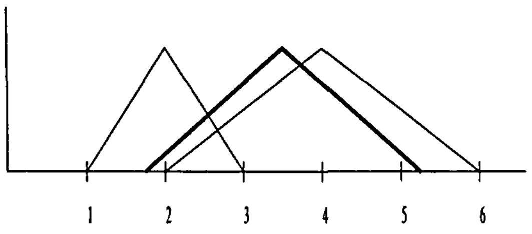


Figure 6.14: EIR recombination for fuzzy numbers with $\delta = 0.25$, and $\alpha = 0.75$ uniformly chosen in $[-0.25, 1.25]$. Mode and spread for the two parents are 2.0, 1.0 and 4.0, 2.0. The thicker number is the result of recombination. As for real numbers, the value of α makes offspring resemble its bigger parent more (a factor of $\frac{3}{4}$) than its smaller one. The resulting mode is 3.5 and the spread 1.75.

Mutation of fuzzy numbers is also developed as an extension of the real-valued operators, by taking into account that mode and spread are collectively expressing a single (fuzzy) number. Both continuous and discrete operators can be used, as follows. First, the change on the mode is determined as in (6.14) or (6.13), respectively. The change on the spread is done in the same way, but using the *same* sign and δ (which are the terms depending on probabilities) as used for the mode. It has to be said that, initially, the proposed change for the spread was altered as a small percentage of the proposed change in the mode, but this ended in very small changes in the spread that made no real difference to the algorithm. Hence, this last correction was abandoned.

The initialization of **linguistic variables** is as follows: first, the left mode is assigned a random value within the pre-declared range, and then the right mode is chosen in the interval between the first mode and the right limit of the range. Alternatively, the right mode can be first chosen in the whole range, and then the left one is chosen in the interval between the left limit and the second mode. To lessen the bias of the method, one of the two strategies is chosen with equal probability and applied. Then, the two spreads (left and right) are selected independently (of each other and of the modes).

Recombination is again developed as an extension of the operators for real-valued quantities. In particular, for the EIR operator, the procedure is analogous as for fuzzy numbers, that is, using the same α for all the involved quantities. In this case, however, the source of uncertainty is different, and there is no need for the spreads of the offspring to be in a proportion to their modes similar to that of the parents, and other operators could be conceivable. Fig. (6.15) provides an example.

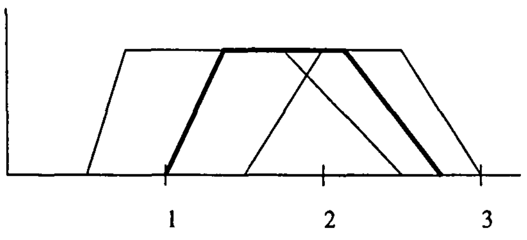


Figure 6.15: EIR recombination for linguistic terms, with $\delta = 0.25$ and α uniformly chosen in $[-0.25, 1.25]$, and equal this time to 0.50, for clarity. The thicker set is the result of recombination. In this case, both parents are equally responsible of the obtained offspring ($\alpha = 0.5$).

Finally, mutation of linguistic terms is also developed as an extension of the operator for fuzzy numbers, where both continuous and discrete operators can be used. A single step change is proposed according to (6.14) or (6.13), which affects all the constituting points (modes and spreads) in the same way. This can be thought of as a *translation* of the linguistic term in the universe of discourse (in the geometric sense).

Missing values are dealt differently than for a GA. They are initially generated according to the estimated probability of a missing value in the variable. This makes sense since for variables containing high numbers of missing values, the probability of placing one in the corresponding weight (of a given unit) increases. In the limit, if *all* the values for a variable were missing, the weights would also be so.

If this probability is zero (since no value was missing for the variable) a missing value for a weight could still be introduced by mutation (signaling the temporal loss of a gene or trait). A mutation operator sets a missing value in the allele with a certain probability (usually very low). If this change leads to improved performance in the corresponding network, it will be retained. A missing value cannot be mutated back to a non-missing one. A definite value can only be recovered by recombination to the (non-missing) gene of another individual.

Recombination is treated as *discrete* (DR) whenever at least one of the parents have a missing trait. This is coherent with the philosophy of EA: recombination stands for the *transmission* of the parents's genetic material to their offspring. If a parent is lacking a gene, this characteristic has to be given the chance to be passed on. Besides, if the trait or gene is lacking for both parents, it will be so for the offspring, since nothing can be "invented from scratch". This will eventually be the role of mutation. In summary, given Ω a recombination operator (possibly heterogeneous), it is extended to a $\Omega_{\mathcal{X}}$ (where $\mathcal{X}$ denotes the missing value) as:

$$\Omega_{\mathcal{X}}(x_i, y_i) = \begin{cases} \Omega(x_i, y_i) & \text{if } x_i \neq \mathcal{X} \wedge y_i \neq \mathcal{X} \\ DR(x_i, y_i) & \text{if } x_i = \mathcal{X} \veebar y_i = \mathcal{X} \\ \mathcal{X} & \text{otherwise} \end{cases} \tag{6.15}$$

where $\veebar$ denotes exclusive-or. All this manipulation for missing values differs from the one done in the GA, in that in the latter, this value was treated as any other value, and generation and propagation were carried out blindly and completely at random. The proposed treatment for the BGA has the added advantage of being simple, and natural from the point of view of an EA (in the sense that it is taken as a missing *gene*) and is independent of the data type.

6.7 Conclusions

We have presented an overview of Evolutionary Algorithms, focusing on the two instances used in many of the experiments carried out in this Thesis, a standard GA and the BGA. Both have been extended with means to represent and manipulate heterogeneous information (including missing values), in order to use them as trainers for an HNN. For the BGA, this has involved the extension of the repertoire of genetic operators, adapting them to specific data types. The algorithm has also been investigated in this task.

The search for generally adequate genetic operators and other BGA parameters —like optimal population size μ and truncation threshold for selection τ— for supervised training of a neural network is of great interest and has to be carried in a principled way, if the aim is to tailor the BGA specifically for this kind of problem. The reasons for this research are numerous, and among them we mention the following:

1. We believe that there effectively *are* BGA configurations (specially, genetic operators) in favour of the particularities of this task. Specialised operators can be devised, but the knowledge gained on the existing ones constitutes a good departure point. Moreover, genetic operators behave at their best for certain balanced combinations of μ and τ. Thus, the relationship between these two important parameters must be further clarified in this context, specially in light of the quite opposite influence they have on the two recombination settings that have stood out -EIR ($\delta = 0.35$) and EIR ($ranges_\delta$).

2. It is also our belief that the potential of EA (and especially of the BGA) to solve this kind of task has not yet been fully employed; the experiments in this work are initial steps in this direction. Hybrid methods in which an EA takes the model selection part and the numerical optimization is left to traditional DBM not only are very time consuming but they do not make the most of EA. This is because, due to the high computational demands, only small populations run for a very low number of generations can be run. Also, the search is biased by an external factor -the DBM- that, due to its nature, is to yield a suboptimal solution that also depends heavily on the initial weights, learning rate, momentum, and other DBM parameters (not controlled by the EA). In addition,

the (reasonable) model space is discrete and usually small, and it makes sense to take profit of incremental/decremental steps until some criterion is fulfilled. Note that one is interested in generalization ability and thus the guiding error is going to be that over the validation set, not over the training set, which is known to decrease with models of increasing complexity (and hence the use of incremental/decremental methods).

3. In the same vain, the hybridization of methods involves a double set of parameters to be optimized at the same time. Not only the learning parameters of the DBM have to be set, but the parameters of the EA have also to be chosen over a rich variety. And it is likely that interactions arise between both methods, making the search on the joint parameter space an unfeasible task.

For these reasons, we postulate for EA to solve the numerical optimization problem only, among which continuous EA (like the BGA or ES) are possibly better suited than traditional binary-coded GAs. To this end, the classical testbeds for EA on which the BGA has already been tested [De Falco *et al.*, 96], [Belanche, 99d], [Belanche, 99e] should be widened with those used in neural benchmarking. This would surely open new directions in its development.

A partial solution for the effect of recombination -at least one that alleviates the problem- is the use of recombination operators with high variance, able to leave the scope of the parents, possibly in a controlled way, depending on how close they are. In addition, the mutation operator should be vigorous enough as to tune (exploit) a solution proposing non-trivial changes, since a fairly small change (e.g. in the fourth decimal) in one of the weights (that is, the expected change in a BGA) is not going to affect network performance. However, a stronger change (e.g. in the second decimal) *can* affect it, either for the better (it will be kept) or for the worse (it will be kept if it still marks as one of the best τ percent).

This intuitive thinking is supported by the results of this work. For recombination, the EIR family of operators stands out from the rest. These operators are the only ones that allow to exit the parents's scope, the amount of which is controlled by their parameter δ. Specifically, a big value of $\delta = 0.35$ and the highly-allowing method $range_\delta$, which dynamically sets it to the maximum feasible value, have been shown to be the best. For mutation, high values of ρ combined with low values of k (actually, the respective high and low extreme of the intervals sampled) yield the best results. This setting favours higher average mutation steps, up to reasonable values. For example, some simple calculations show that, for the selected values $\rho = 0.5$ and $k = 16$, and the weight range of $[-10, 10]$, the expected mutation step is around 0.04 (positive or negative), very approximately the mean change in the second decimal.

The results presented in this Chapter, corresponding to an initial study -and specifically a study of *relative performance* between parameter settings- should not be taken as a useful comparison to those obtained by DBM. Moreover, the choice of the cost function and the range of weights is likely to certainly exert an influence in BGA performance because, although the underlying task is basically the same, what the BGA sees is a different fitness function and has a different range where to generate solutions to it. For instance, it is specially likely that a function not suitable for DBM training, the number of correctly classified examples, could give good results. All this remains to be worked out in the future.

Chapter 7

Experiments on Real-World Problems

I know why there are so many people who love chopping wood.
In this activity, one immediately sees the results.

Albert Einstein

7.1 Introduction

If a new approach is developed for its assumed utility, an empirical comparison is a good means to assess it. In this setting, it should be verified that new algorithms or, in our case, new models, perform well for some *real* problems, as these are the only tests that are guaranteed to have practical relevance, not only for the specific problem being considered, but also as indications of a generic applicability of the approach to the always challenging domain of real-world problems.

An additional and most important question has to be addressed: for what kinds of problems is the approach best suited or recommended? In our case, the answer has been pointed out from the outset, and constitutes one of the thesis of the present work: whenever there is a natural and modelable heterogeneity in the data or there exists an explicit knowledge (which is task-dependent) that can be written in the form of a similarity relation defined on the data patterns.

7.2 Contents

The following material consists mainly of experimental work carried out on three quite different real problems, one in Medicine (the first one presented) and the other two in the field of Environmental Sciences. It has been extracted from [Belanche and Valdés, 98c], [Belanche, Valdés and Alquézar, 98a], [Nieto, 00], [Valdés, Belanche and Alquézar, 00] and

[Belanche and Valdés, 99a].

The first problem aims at the finding of models for the controllers in the central nervous system control (CNSC). Three variants of the task are considered, differing in the difficulty and in the variety of data used.

The second problem is an environmental study in which two heterogeneous models are used in an imprecise classification task, aimed at detecting underground cavities.

The third problem tackles another environmental task, this time a multi-class classification, for the identification of valid models in the context of a geochemical study on arctic natural waters.

A word of warning is in order about the experiments. Due to the different timings in the availability of the data and the fact that they were worked out in different stages of the Thesis, they are embedded in methodological and experimental settings that are not always fully coincident. Nonetheless, an effort of unified presentation has been made to minimize this effect, but without altering the original settings or fundamental results, because this would have meant to redo them completely. In practical terms, it only means that the different experiments should be looked at individually.

7.3 Experimental setting

In all the experiments, an heterogeneous neuron model grounded on the measure in (§4.4.1) (with $s_{max} = 1$) is used, based on a simple additive similarity aggregation operator, followed by a non-linear similarity-keeping or $\check{s}$ function, acting as a logistic activation function by adapting it to the real domain $[0, 1]$. The neuron model computes a function $F_i(\vec{x})$ as follows:

$$F_i(\vec{x}) = \check{s}\left(\frac{\sum_{k=1}^{n} s_k(x_k, w_{ik})\,\delta_k(\vec{x}, \vec{w}_i)}{\sum_{k=1}^{n} \delta_k(\vec{x}, \vec{w}_i)}\right) \tag{7.1}$$

where $\check{s}(z) = g(z, k)$ (4.74) with $k = 0.1$ and, being $\mathcal{X}$ the missing information symbol,

$$\delta_k(\vec{x}, \vec{y}) = \begin{cases} 1 & \text{if } x_k \neq \mathcal{X} \wedge y_k \neq \mathcal{X} \\ 0 & \text{otherwise} \end{cases}$$

The partial measures s_k between the variables are computed using the partial similarity functions defined in (§4.3), chosen accordingly to each data set as described individually for each of them. In the considered problems, all information is originally continuous and the two possible models (with or without considering the underlying uncertainty) are consistently explored. The used partial measures for continuous variables are always distance-based ones, based upon the measure defined in (4.51) and, unless otherwise stated, using (4.76) with $d = 1, \alpha = 1$, that is, the function $\hat{s}(z) = 1 - z$ or S0_0 in the Table (4.1) of similarity transforming functions, corresponding to the basic measure (4.52). The measure (4.60) is the one used for fuzzy numbers.

In this respect, it is interesting to note that, in effect, the variables in the experiments exhibit a significant amount of imprecision. Two generic models –the continuous and the fuzzy continuous– are studied in the form of two heterogeneous neuron models, conveniently denoted as the h and f models (for *heterogeneous* and *fuzzy*, respectively). The reference model for comparison is the P-neuron, denoted as the n model (standing for *normal*). Other techniques are sometimes considered, such as the complex neuron model [Birx and Pipenberg, 92], consequently denoted the c model, or the k-nearest neighbours algorithm [Fukunaga, 90].

In most of the studies, several architectures are explored, varying the neuron model and the number of hidden units. To this end, the following notation in introduced:

Let q_x denote a *single* layer of q neurons of type x, where possibilities for x are n, h, f and c, corresponding to architectures with no hidden neurons. Accordingly, $p_x q_y$ denotes a feed-forward network composed of a hidden layer of p neurons of type x and an output layer of q neurons of type y. For example $4_h 5_n$ is a network composed of a hidden layer of 4 neurons of type h and an output layer of 5 neurons of type n. Shortcut (direct input to output) connections are not considered.

Regarding the neural training procedure, several learning algorithms are employed, including evolutionary techniques (such as genetic algorithms) and derivative-based ones (such as backpropagation or the conjugate gradient). In any case, all neural architectures are trained in each particular experiment using the same algorithm with the same set of control parameters, to eliminate this source of variation from the analysis. Sometimes, classical neuron models are trained with two methods (one evolutionary and one derivative-based) to have a supplementary set of results.

A standard genetic algorithm (denoted SGA), enhanced to deal with missing values, as explained in (§6.3.2), is always used with the following characteristics: binary-coded values, probability of crossover $P_{cross} = 0.6$, probability of mutation $P_{mut} = 0.01$, a linear scaling with factor $c = 1.5$, selection mechanism: stochastic universal, and a replacement procedure given by the worst individuals. To avoid misunderstandings, population size and other particular settings for this and other methods are made explicit for each experiment.

7.4 Learning models of the CNSC

7.4.1 Preliminaries

The study and prediction of time-varying processes is a fundamental problem with a long tradition in the literature. For this study, the availability of several time-series of cardiology data from a patient and the knowledge of previous attempts to induce accurate models out of these data using *Fuzzy Inductive Reasoning* [Nebot *et al.*, 98] brought an opportunity to bring the different neural models and networks into comparison, in the context of the Heterogeneous Neural Network approach.

The cardiovascular system –see Fig. (7.1)– is composed of the hemodynamical system and the Central Nervous System Control (CNSC). The CNSC generates the regulating signals for the blood vessels and the heart, and it is composed of five controllers: *heart rate*, *peripheral resistance*, *myocardial contractility*, *venous tone* and *coronary resistance*. All of these controllers are single-input/single-output (SISO) systems driven by the same input variable, the *carotid sinus pressure.*

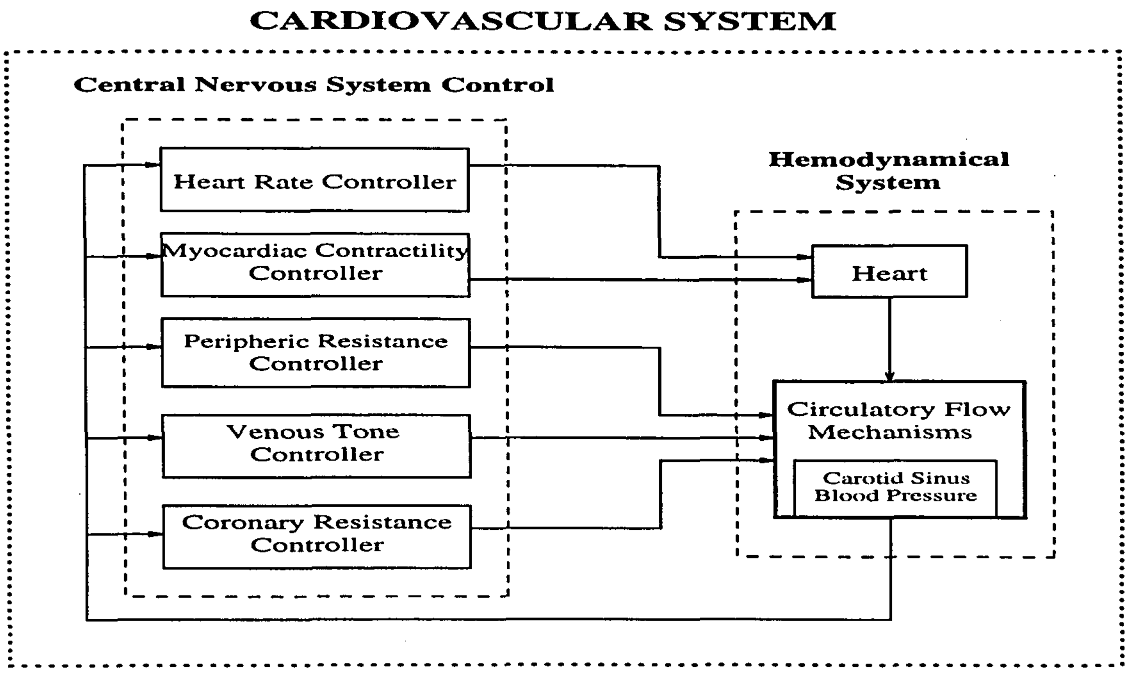

CARDIOVASCULAR SYSTEM
Central Nervous System Control
Heart Rate Controller
Myocardiac Contractility Controller
Peripheric Resistance Controller
Venous Tone Controller
Coronary Resistance Controller
Hemodynamical System
Heart
Circulatory Flow Mechanisms
Carotid Sinus Blood Pressure

Figure 7.1: The cardiovascular system.

Whereas the structure and functioning of the hemodynamical system are well known and a number of quantitative models have already been developed that capture its behavior fairly accurately, the CNSC is, at present, still not completely understood and no good deductive models exist able to describe the CNSC from physical and physiological principles. Although some differential equation models for the CNSC have been postulated [Leaning *et al.*, 83], these models are not accurate enough, and therefore, the use of other modeling approaches -like neural networks or qualitative methodologies- has been shown in [Cueva, Alquézar and Nebot, 97] to offer an interesting alternative to classical quantitative modeling approaches, such as differential equations and NARMAX techniques [Vallverdú, 93].

7.4.2 Experiments

Experiment 1

The first set of experiments compares the effectiveness of heterogeneous models with that of more classical models, like P-neurons and complex neurons, in a time-series forecasting setting, focused on the first of these signals, the *heart rate.*

Models tested

Since the use of the heterogeneous neuron as a brick for configuring network architectures can be done in several ways, in this first set of experiments several architectures are explored, restricting ourselves to networks with one or no hidden layers. The possible combinations include fully heterogeneous networks, in which all neurons (including the output ones) are of the same heterogeneous type and compute the same similarity relation. In the present study, there is just one output to be predicted (the *heart rate*). Consequently, the output layer is always composed of a single neuron. The hidden layer (if any) will always have $h_1 = 3$ neurons. All of them are to be trained with the SGA in exactly the same conditions.

It should be noted that there has been no attempt to find better architectures (different number of hidden-layer neurons and/or more than one hidden layer) nor to improve GA performance on this particular problem by tuning its parameters or devising specialized operators. It is reasonable to believe that this would probably have improved the results obtained with the different neuron models. However, these setting are very likely to be changing for different models, and would have introduced a strong bias. Our main concern is to have them compared in a (perhaps crude) but absolutely fair way, using reasonable, although maybe not optimal, settings.

The neural models tested correspond to those denoted by the code letters n, h and f, as introduced in (§7.3). The architectures under study will then be: $1_n, 1_h, 1_f, 3_n1_n, 3_h1_n, 3_f1_n, 3_n1_h, 3_n1_f, 3_h1_h$ and 3_f1_f.

The only difference between h and f neurons is that –according to the fuzzy heterogeneous model– the latter have their inputs and weights fuzzified. In this experiment, original crisp data were converted into (triangular) fuzzy numbers in the form of a 5% of imprecision w.r.t. the original value. Though this percentage is probably an upper-bound for modern measuring devices, it was considered adequate for the task.

In order to better assess the performance of the different HNN architectures, another powerful neural approach was also employed to infer a model for the task at hand: a feed-forward neural network working in the complex plane, of which a brief comment is due.

Complex neural models

The complex neural network (CNN) is an advanced model [Birx and Pipenberg, 92] which operates in the complex plane, having inputs, weights and outputs given by complex numbers. They have been used very successfully in the analysis of many complex dynamic systems and in difficult classification problems (e.g. [Birx and Pipenberg, 93]). In these networks, the transfer function is a direct translation of the scalar product to complex arithmetic. Let $z = x + iy \in \mathbb{C}$ be the complex neuron net input as given by the scalar product. The squashing function used is given by $f(z) = f(x+iy) = p\,x + i\,p\,y$ where $p = \frac{tanh(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$. Following our terminology, we will use a 3_c1_c architecture, where c denotes a complex neuron. In this case, the output neuron will use a linear activation function. The training procedure chosen is a combination of the simulated annealing and conjugate gradient-descent techniques explained in Chapter (2).

Data preparation

The input and output signals of the CNSC were recorded with a sampling rate of 0.12 seconds from simulations of the purely differential equation model. The model had been tuned to represent a specific patient suffering from an at least 70% coronary arterial obstruction, to agree with the measurement data taken from the patient. The full set of data consists of 7,869 timed measurements. From these, the first 1,500 were used as training set and the immediately following 1,000 as the test set to be forecast. To give a graphical impression, the input and output variables of the *heart rate* controller subsystem are displayed in Figs. (7.2) and (7.3). Note that both signals exhibit high-frequency oscillations modulated by a low-frequency signal.

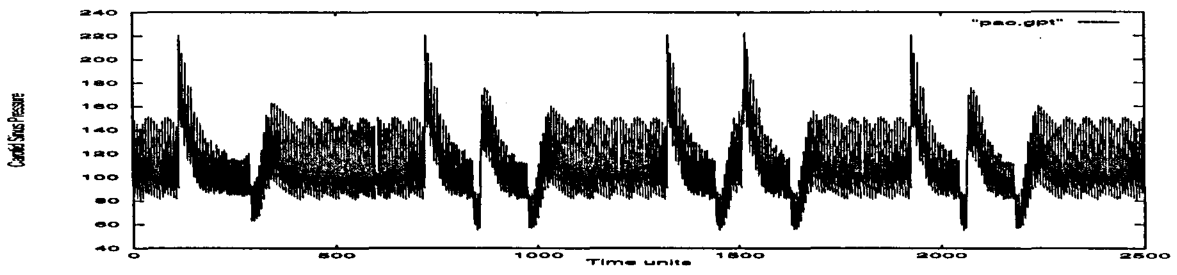


Figure 7.2: Input signal: *Carotid Sinus Pressure.*

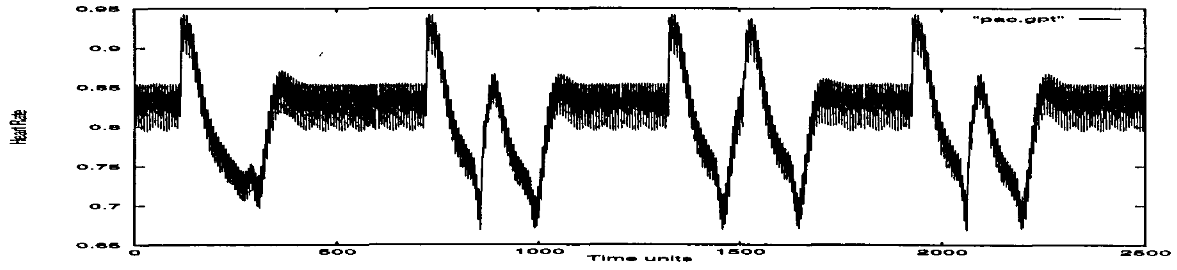


Figure 7.3: Output signal: *Heart Rate Controller*, measured in seconds between beats.

In previous studies of the data at hand [Nebot *et al.*, 98], the obtained results were found to be greatly improved by performing a prior Markov analysis on the data, in search of single-dependency variable-order significant time delays. This hybrid technique had been very successfully applied to the task at hand using data from different patients and controllers and different training and test set regimes. It was found that in both input $x(t)$ and output $y(t)$ signals there were two specific time delays (1 and 6 sampling intervals), highly significant from the point of view of exhibiting a Markov chain behavior when the continuous process is discretized.

Accordingly, a training set consisting of four inputs $x(t-1), x(t-6), y(t-1), y(t-6)$ and one output $y(t)$ was constructed. This generic model was then used for all the approaches. For the CNN, this information was given as two complex numbers $x(t-1)+x(t-6)i, y(t-1)+y(t-6)i$. Note that the weights of the CNN are also complex.

Results

For each neural architecture, five different training trials were run using different random initial populations, in an attempt to reduce the effect of a specially lucky (or unlucky) strike by the SGA. Average and best mean-square errors (MSE) on *test* set were then calculated and are the ones shown -Tables (7.1) and (7.2). The CNN was given 50 different annealing

Architecture	1_n	1_h	1_f
Average	2.150e-02	9.855e-04	5.723e-04
Best	9.965e-04	9.657e-04	3.510e-04

Table 7.1: MSE errors for the different single-layer HNN architectures.

Architecture	3_n1_n	3_n1_h	3_n1_f	3_h1_n	3_h1_h	3_f1_n	3_f1_f
Average	1.640e-03	2.621e-03	1.594e-03	1.114e-04	1.661e-04	7.817e-05	7.683e-05
Best	1.216e-03	2.603e-03	1.036e-03	9.405e-05	9.424e-05	6.652e-05	6.527e-05

Table 7.2: MSE errors for the different multilayer HNN architectures.

restarts and the final (overall best) result attained was $8.136e - 05$. Notice the decrease of MSE in orders of magnitude due to the increasing presence of heterogeneous neurons, until a comparable (and slightly better) performance to that obtained by the CNN is reached.

Experiment 2

The second set of experiments compares the effectiveness of heterogeneous models exclusively with that of classical models (P-neurons) but using three kinds of architectural dispositions and learning algorithms: a time-delay neural network trained with backpropagation, the same architecture trained with the conjugate gradient plus simulated annealing method, and an augmented single-layer recurrent network or ASLRNN, trained by a true gradient-descent method. This time the target is composed of all five controllers of the CNSC.

Neural approaches used in the experiments

In general, two types of neural network architectures can be used for learning tasks involving a dynamic input/output relation, such as prediction and temporal association: *time-delay neural networks* (TDNNs) and *recurrent neural networks* (RNNs). The HNN model is is used here as a TDNN and compared to a RNN and to two other different TDNN models, as described below.

Time-delay neural networks

If some fixed-length segment of the most recent input values is considered enough to perform the task successfully, then a temporal sequence can be turned into a set of spatial patterns on the input layer of a multi-layer feed-forward net trained with an appropriate algorithm such as backpropagation. These architectures are called TDNNs, since several values from an external signal are presented simultaneously at the network input using a moving window (shift register or tapped delay line) [Hertz, Krogh and Palmer, 91]. A main advantage of TDNNs in front of RNNs is their lower cost of training, which is very important in case of long training sequences. TDNNs have been applied extensively in recent years to different tasks, in particular to prediction and system modeling [Lapedes and Farber, 87].

In the case of learning a SISO controller, with an input real-valued variable $x(t)$ and an output real-valued variable $y(t)$, the output layer of a TDNN consists of a single output unit that will provide the predicted value for $y(t)$, whereas the input layer holds some previous values $y(t-1), \ldots, y(t-m)$ and some recent values of the input variable $x(t), x(t-1), \ldots, x(t-p)$, from which the value $y(t)$ could be estimated (i.e. a total number of $m+p+1$ *model* inputs). Additionally, a layer of h_1 hidden units (h_1 to be determined) is required. In the present study, two different TDNN approaches that differ in the training method have been tested: a standard backpropagation algorithm (TDNN-BP) using sinusoidal units, and the hybrid procedure composed of repeated cycles of simulated annealing coupled with a conjugate gradient algorithm (TDNN-AC) described in [Ackley, 87]. For the latter, hyperbolic tangent units form the hidden layer whereas the output layer is composed by a linear neuron. It should be noted that the HNN model as used here (TD-HNN) can be viewed as a TDNN that incorporates heterogeneous neurons and is trained by means of genetic algorithms.

Recurrent neural networks

In recent years, several RNN architectures including feedback connections, together with their associated training algorithms, have been devised to cope naturally with the learning and computation of tasks involving sequences and time series. A type of RNN that has been proven useful in grammatical inference through next-symbol prediction is the first-order augmented single-layer RNN (or ASLRNN) [Sopena and Alquézar, 94], which is similar to Elman's SRN [Elman, 90] except that is trained by a true gradient-descent method, using backpropagation for the feed-forward output layer and Schmidhuber's RTRL algorithm [Schmidhuber, 92] for the fully-connected recurrent hidden layer. Although the use of sigmoidal activation functions has been common in both RNNs and backpropagation networks, a better learning performance can be achieved using other activation functions such as the sine function [Sopena and Alquézar, 94]. Such networks with sinusoidal units can be seen as generalized discrete Fourier series with adjustable frequencies [Lapedes and Farber, 87]. Hence, the ASLRNN model used here was built up with sinusoidal units.

Experiment setup

The data used in the training and test phases of the experiments came from a single subject. All five CNSC models, namely, *heart rate*, *peripheral resistance*, *myocardial contractility*, *venous tone* and *coronary resistance*, were inferred for this subject by means of the neural approaches aforementioned. The input and output signals of the CNSC controllers were recorded with a sampling rate of 0.12 seconds from simulations of a purely differential equation model. This model had been tuned to represent a specific patient suffering from coronary arterial obstruction, by making the four *different* physiological variables (right auricular pressure, aortic pressure, coronary blood flow, and heart rate) of the simulation model agree with the measurement data taken from the patient. The training set was composed of 1,500 data points for each controller, whereas six data sets not used in the training process (600 points each) were used as forecasting targets, containing signals that represent specific morphologies. The HNN and the TDNN architectures were fixed to include 1 output unit, 8 hidden units, and 7 input units, corresponding to the values $x(t)$, $x(t-1)$, $x(t-2)$, $x(t-3)$, $y(t-1)$, $y(t-2)$ and $y(t-3)$, where $x(t)$ denotes the current value of the input variable and $y(t-1)$ denotes the value of the controller output in the previous time step. All inputs to the HNN were treated as fuzzy numbers with an uncertainty of a 5%, and the accordingly defined similarity relation was used. The first-order ASLRNN architecture also included 1 output and

8 hidden units, but just 2 input units, corresponding to the values $x(t)$ and $y(t-1)$, though in this case the hidden layer incorporated additional weights for the feed-back connections.

In the testing process, the normalized mean square error (in percentage) between the predicted output value, $\hat{y}(t)$, and the controller output, $y(t)$, was used to determine the quality of each of the inferred models. This error is given by:

$$NMSE = \sqrt{\frac{\sum_i \|\vec{\zeta}_i - \vec{y}_i\|^2}{\sum_i \| < \vec{y}_i > - \vec{y}_i\|^2}} \cdot 100\%, \qquad (7.2)$$

where $\vec{\zeta}_i = \mathcal{F}_{\vec{w}}(\vec{x}_i)$ is the network's response to input pattern $\vec{x}_i$, and $< \vec{y}_i >$ represents the mean of the target data over the required set.

For each CNSC controller and neural approach three different training trials were run using a different random weight initialization. The HNN was trained using the SGA as explained in (§7.3), with 100 individuals. The algorithm stopped when no improvement was found for the last 1,000 generations (typical values were about 5,000). On the other hand, the TDNN-BP and ASLRNN nets were allotted 3,000 epochs using a small learning rate of $\alpha = 0.025$ to allow a smooth minimization trajectory. These parameters were tuned after some preliminary tests. For each run, the network yielding the smallest NMSE error on the training set during learning was taken as the controller model. The TDNN-AC was trained in only one run and the process was stopped after 20 annealing restarts.

Results

The nets resulting from the training phase were applied to the training set and to the six test data sets associated with each controller. The normalized MSE errors for these sets were calculated, together with their averages for the different training runs and test sets. The summary of the errors obtained by the different neural approaches is displayed in Table (7.3).

Controller	TD-HNN		TDNN-BP		TDNN-AC		ASLRNN	
	Train.	Test	Train.	Test	Train.	Test	Train.	Test
HRC	0.11%	0.18%	1.15%	1.52%	0.15%	0.13%	1.63%	1.91%
PRC	0.09%	0.12%	0.94%	1.27%	0.26%	0.14%	0.84%	1.10%
MCC	0.03%	0.06%	0.81%	1.33%	0.09%	0.08%	0.71%	1.18%
VTC	0.03%	0.06%	0.81%	1.33%	0.09%	0.08%	0.71%	1.18%
CRC	0.10%	0.11%	0.47%	0.66%	0.03%	0.04%	0.41%	0.53%
mean	0.07%	0.11%	0.84%	1.22%	0.12%	0.09%	0.86%	1.18%

HRC *heart rate*, **PRC** *peripheral resistance*, **MCC** *myocardial contractility*
VTC *venous tone*, **CRC** *coronary resistance*

Table 7.3: Average normalized MSE errors for the training sets (left) and test sets (right) of the CNSC controller models inferred by each neural approach.

It is interesting to observe the excellent results yielded by the models inferred by both the HNN and the TDNN-AC, especially as compared to the TDNN-BP and ASLRNN, which showed an almost identical prediction performance, possibly caused by a short depth of temporal dependencies in the modeled system (i.e. all relevant past information could be included in the moving window that selects the inputs of a TDNN).

Experiment 3

The third set of experiments compares the effectiveness of distance-based heterogeneous models with that of scalar product. This is an interesting setting because, being all variables continuous, the experiments permit to focus on different similarity measures defined exclusively on continuous data; it also means that they all can be trained with a derivative-based method, which in this case is the conjugate gradient plus annealing methodology explained in Chapter (2). This involved the computation of analytic expressions for the derivatives of all the neuron models described below. The presence of missing information is specially investigated, including its effect on scalar product-driven models.

Targets are two of the CNSC controllers: the *heart rate* (HRC) and the *coronary resistance* (CRC). These two series are shown, for their first 1,500 samples, in Figs. (7.4) and (7.5), respectively. For each of them, the following model is built:

$$y(t+\tau) = F\{x(t-1), x(t-2), x(t-3), y(t), y(t-1), y(t-2), y(t-3)\}, \tau \in \mathrm{N}^{+} \qquad (7.3)$$

where $y(t)$ is the time-series (HRC or CRC) and $x(t)$ is the control signal, in both cases the *Carotid Sinus Pressure*, as explained in (§7.4.1) and depicted in Fig. (7.2).

Experimental setting

For each of the two controllers, six sets of experiments are performed, with a varying $\tau \in \{1, 4\}$ and a percentage of missing information $X\% \in \{0\%, 10\%, 30\%\}$. For each experiment, a 3-fold cross-validation procedure is worked out on the 1,500-sized data sets. This means that three partitions are investigated, with 1,000 points for training and 500 for validation. An independent set of 600 samples is used as a test set to assess generalization performance. Each training run is carried out to end of resources, given by a limit of 1,000 epochs (presentations of the training set, with learning purposes). The results reported are the average of the three runs. All the variables have been normalized to lie in the interval $[0, 1]$.

The motivation behind the introduction of missing data is to evaluate its impact on the performance of the considered neural models. To this end, the original data sets are used as is (0% of missing values), and altered by randomly and uniformly seeding missing values (10% and then 30%), done equally for training, validation and test sets. This makes sense in the studied domain, an in general in Medicine, where data can be absent by a manifold of circumstances: values that get lost in the patient history, invalid results, improper handling by the patient, etc. In the present case, the data were originally clean because they came from a simulation, thus not being fully realistic (complete and perfectly crisp).

Models tested

All the models set forth based on a distance computation correspond to similarity models

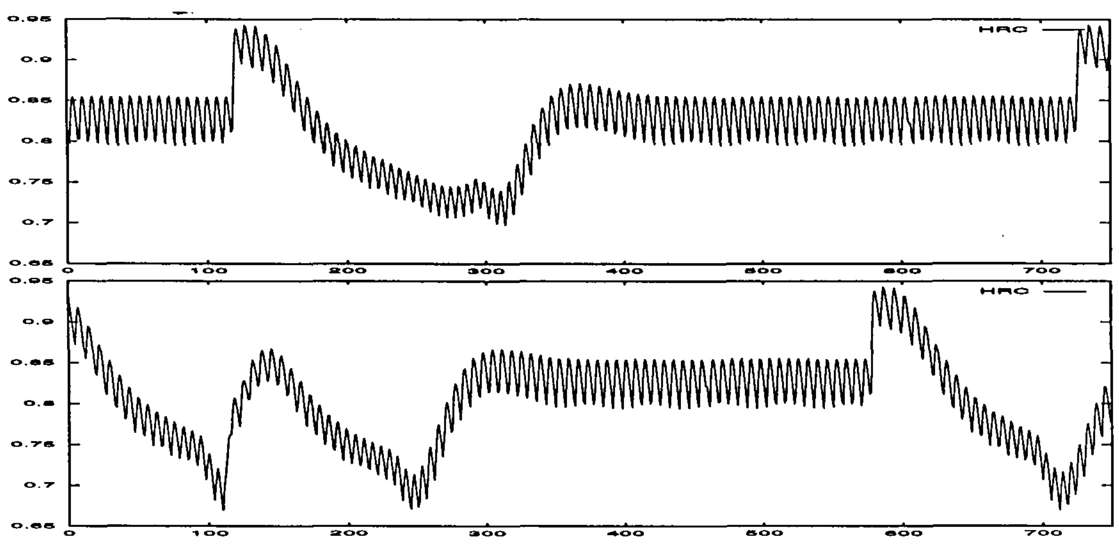


Figure 7.4: Output signal: *Heart Rate* Controller, shown in two parts, totalling 1,500 consecutive samples.

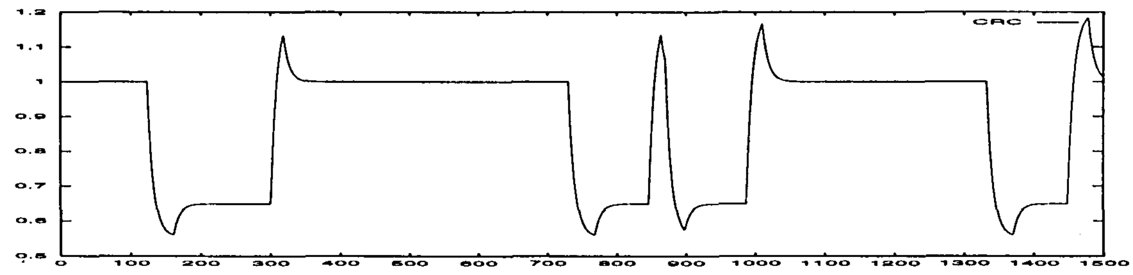


Figure 7.5: Output signal: *Coronary Resistance* Controller, of 1,500 consecutive samples.

of type (A) -obtained by transformation from a global distance- and are thus kinds of RBF units (in the wide sense), whereas the scalar-product based neurons correspond to similarity models of type (C). To describe the distance-based models, we follow the notation introduced in (§4.2.1), and cast them as instances of a generic family of weighted Minkowskian distances, as defined in (4.4). They are all displayed in Table (7.4).

The names SPR and SPN will refer, respectively, to the standard scalar product and to a normalized counterpart, enhanced to handle missing values (see below). They both use the hyperbolic tangent as activation function. All the other models employ the adapted similarity keeping function $g(z, 0.25)$ (4.74). The difference between EU1 and EU2 is that the former uses $\hat{s}_1(z) = \frac{1}{1+z}$ as similarity transforming function, while the latter makes use

Index	Parameters			Description
	n'	q	$\vec{v}$	
GOW	n	1	$\vec{\sigma}_{dev}$	Mean City-block distance, weighted by the inverse of the maximum deviation
CLA	1	2	$\vec{1}$	Coefficient of divergence or Clark distance
CAN	1	1	$\vec{1}$	Canberra distance
EU0	1	2	$\vec{1}$	Basic unweighted Euclidean distance
EU1	1	2	$\vec{\sigma}_{dev}$	Euclidean distance weighted by the inverse of the maximum deviation
EU2	n	2	$\vec{\sigma}_{dev}$	Mean Euclidean distance weighted by the inverse of the maximum deviation
CYB	1	1	$\vec{1}$	Unweighted City-block distance
MN4	1	4	$\vec{1}$	A basic unweighted Minkowskian distance
PEA	1	2	$\vec{\sigma}^2$	Pearson distance

$\vec{\sigma}_{dev}$: vector of maximum deviations, $\vec{\sigma}^2$: vector of variances, $\vec{1}$: unity vector

Table 7.4: The different distances used in the experiments.

of $\hat{s}_0(z) = 1 - z$. The other function that uses $\hat{s}_0(z)$ is GOW; all other distance functions make use of $\hat{s}_1(z)$. In all cases, $s_{max} = 1$.

All the neuron models (except SPR) make use of the same built-in treatment for missing values consistently used throughout the Thesis: a normalization by the number of actually performed (partial) computations. All are are trained (including the two scalar products) in exactly the same conditions and make use of the same data for all the experiments, so that any difference in performance is only attributable to the different models. The neurons are arranged in two different architectures, with $h_1 \in \{16, 32\}$ hidden units. The output neurons are always linear P-neurons (that is, they only perform linear combinations of the hidden units).

Presentation of results

Due to the high volume of information, the results are presented in a more compact form, always averaged for the two architectures and displayed in two different formats: according to the $t + \tau$ target (averaging out for the different percentages of missing data) and conversely, according to the different percentages of missing data, and averaging out for the two $t + \tau$ targets. The results for different entities are averaged as follows: let $R(h, \tau, X)$ denote the results achieved -for fixed model and controller- with h hidden neurons, in $t + \tau$ and with a X% of missing information. Let $\overline{R}[x]$ denote the weighted average w.r.t. argument x of R, with outer correction factors equal to the inverse sum of the weightings, as follows:

$$\overline{R}[h](h, \tau, X) = \frac{2}{3}\left(\frac{1}{2}R(16, \tau, X) + R(32, \tau, X)\right) \tag{7.4}$$

$$\overline{R}[\tau](h, \tau, X) = \frac{1}{2}\left(R(h, 1, X) + R(h, 4, X)\right) \tag{7.5}$$

$$\overline{R}[X](h,\tau,X) = \frac{2}{3}\left(R(h,\tau,0)+\frac{1}{3}R(h,\tau,10)+\frac{1}{6}R(h,\tau,30)\right) \tag{7.6}$$

The first type of information is displayed in Table (7.5) together for HRC (left part) and CRC (right part). The second type is displayed in Table (7.6) for HRC and in Table (7.7) for CRC. In all the tables, the best three results for each column are shown boldfaced.

	HRC				CRC			
Index	$t+1$		$t+4$		$t+1$		$t+4$	
	Train	Test	Train	Test	Train	Test	Train	Test
SPN	3.183	3.993	8.633	10.856	5.431	7.281	6.151	8.197
GOW	2.291	3.112	8.420	11.372	1.215	1.929	2.326	3.161
CLA	**1.749**	**2.404**	**6.968**	**8.809**	0.910	1.449	**2.037**	**2.868**
CAN	**1.793**	**2.418**	**7.309**	**9.598**	1.297	2.008	2.198	3.197
EU0	**1.983**	**2.563**	7.489	9.796	**0.745**	**1.237**	**1.953**	**2.726**
EU1	2.406	2.990	8.167	11.580	**0.876**	**1.446**	2.119	3.305
EU2	2.819	3.600	8.679	23.048	1.304	2.651	2.519	3.789
CYB	1.984	2.791	7.788	11.211	1.023	1.501	2.128	3.411
MN4	2.118	2.753	**7.083**	**9.632**	**0.548**	**0.992**	**1.601**	**2.389**
PEA	2.460	3.758	9.007	11.444	4.476	6.831	6.142	9.867

Table 7.5: Performance results according to the two $t+\tau$ targets.

We use the informal notation X > Y to express that X is better than Y. The first notable point, in *both* controllers, is the coherence of results, not only for $t+1$ but also for the $t+4$ delay; hence, we make a collective discussion. To begin with, CYB > GOW, signaling that a normalization to a distance in [0,1] may not always be the best choice. However, GOW uses $\hat{s}_0(z)$, which is linear, whereas CYB uses $\hat{s}_1(z)$, which is not. This could explain the different behaviour. Besides, there is another city-block based distance (PEA) using $\hat{s}_1(z)$, and this one gives very poor results. Similarly, for the Euclidean distances, EU0 > EU1 > EU2, signaling again that normalizations are no good for the task at hand. It should be recalled that the variables had already been normalized (by the maximum deviation) prior to training.

For HRC, the best measures are clearly CLA and CAN, followed by MN4 and EU0. This pattern is similar for CRC, though in inverse order: the best measures are clearly MN4 and EU0, followed by CLA and EU1. This points CLA, EU0 and MN4 as the more robust measures. In both cases, the worst measures are PEA, EU2 and SPN (not counting the basic SPR, not shown since it does not accept missing values). The overall behaviour is perhaps more clearly seen from the perspective of the increasing presence of missing information, as shown in Tables (7.6) and (7.7).

For the HRC data, and in the case of no missing values, it is noteworthy how SPN > SPR, with SPN yielding approximately half the errors of SPR, indicating that normalization is an interesting feature for the scalar product. In general, though, when compared to the distance-based measures, SPN performs badly, being one of the worst measures. This is mainly caused by the missing information. For complete data, SPN is among the best models, second only

Index	0%		10%		30%		Average	
	Train	Test	Train	Test	Train	Test	Train	Test
SPR	2.987	3.737	-	-	-	-	-	-
SPN	**1.892**	**2.363**	9.654	12.779	22.511	27.084	5.908	7.424
GOW	2.145	2.999	8.126	11.098	19.078	24.986	5.355	7.242
CLA	2.220	2.900	**6.281**	**8.244**	**13.343**	**16.569**	**4.359**	**5.607**
CAN	2.322	3.005	**6.706**	**8.933**	**13.614**	**18.176**	**4.551**	**6.008**
EU0	**1.890**	**2.338**	7.731	10.295	**15.818**	**20.992**	4.736	**6.179**
EU1	1.914	**2.369**	8.127	12.681	19.846	25.988	5.287	7.285
EU2	1.920	2.373	8.436	37.154	23.349	31.372	5.749	13.324
CYB	2.113	3.641	7.244	10.085	16.809	20.997	4.886	7.001
MN4	**1.892**	2.410	**6.864**	**9.878**	16.326	21.520	**4.601**	6.193
PEA	1.955	2.470	8.631	11.481	22.614	30.628	5.734	7.601

Table 7.6: HRC: Performance results according to the amount of missing information.

Index	0%		10%		30%		Average	
	Train	Test	Train	Test	Train	Test	Train	Test
SPR	0.492	0.841	-	-	-	-	-	-
SPN	**0.231**	**0.311**	10.631	14.401	29.475	38.985	5.791	7.739
GOW	0.273	0.406	2.489	3.691	9.319	13.088	1.771	2.545
CLA	0.290	0.450	**1.952**	3.314	**7.618**	**10.099**	**1.473**	**2.159**
CAN	0.308	0.451	2.642	4.356	8.597	12.004	1.747	2.603
EU0	0.261	**0.359**	**1.866**	**2.647**	**6.838**	**10.386**	**1.349**	**1.982**
EU1	**0.257**	0.380	2.004	3.475	7.928	12.152	1.498	2.376
EU2	0.260	0.362	2.272	3.626	11.097	19.556	1.911	3.220
CYB	0.278	0.842	2.236	**2.894**	8.038	11.266	1.575	2.456
MN4	**0.255**	**0.359**	**1.382**	**2.631**	**5.374**	**7.797**	**1.074**	**1.690**
PEA	0.303	0.396	6.018	8.931	33.927	54.907	5.309	8.349

Table 7.7: CRC: Performance results according to the amount of missing information.

to EU0, and in the line of results of EU1, EU2 and MN4. It is remarkable that these five models are the best both in training and in test. A very similar picture can be seen regarding the CRC data.

For a 10% or a 30% of missing values, the situation changes radically, and the best models are clearly, in this order, CLA and CAN, followed by MN4 and EU0 for HRC, and MN4, EU0 followed by CLA, for CRC. SPN, however, accepts badly the lack of information and its performance decays vigorously.

Concluding remarks

Missing information exerts a strong influence on performance, affecting all the models, though ones more markedly than others. For complete data, SPN, EU0 and MN4 are clearly the best ones. In the presence of missing information (10% or 30%) CLA, EU0 and MN4 go in the lead. This points these three distances (and CAN to a lesser degree) as the more robust overall. On the contrary, SPN is not able to cope with missing data, at least with the used mechanism of normalizing by the number of non-absent components in a pattern. The alternative is to have them encoded in the data, and then use a classical (possibly normalized) scalar product. However, the use of this normalization seems to greatly increase performance to the level of the best distance-based measures, although is not enough in presence of missing values. This is interesting, because the use of measures that carry out a normalized computation is one of the generic postulates for heterogeneous neuron models. All these results, and this is most remarkable, happen *both* for training and test data.

7.4.3 Conclusions

Heterogeneous neural networks have been successfully tested in a signal forecasting task, in order to learn controller models for the central nervous system control. The experiments show how the use of fuzzy heterogeneous networks can significatively increase the accuracy of the models obtained. These networks have been compared to the standard multi-layer perceptron and to a complex neural network, for the task at hand. The results obtained show a remarkable increase in performance when departing from the classical neuron and a similar one when compared to other current powerful neural techniques, such as the CNN. The learning and generalization performance of time-delay HNNs are also comparable to that of other TDNNs trained with sophisticated optimization algorithms, and better than that of TDNNs trained with backpropagation and RNNs trained with a true gradient-descent algorithm. Finally, the performance of several distance-based models and scalar product in a thorough study with varying percentages of missing information points to a general better adequacy of the former models to handle this important problem, and how normalization can lead to better performance in the case of scalar product.

7.5 Handling imprecise classification problems

7.5.1 Preliminaries

This second problem is an environmental study –using geophysical data processing– in which two heterogeneous models are used in an imprecise classification task aimed at detecting underground cavities.

An environmental investigation made in the tropics, dealing with the detection of underground caves using geophysical measurements collected at the surface of the earth is the departing point for the different experiments. First, some words describing the problem are necessary.

Karstification is a peculiar geomorphological and hydrogeological phenomenon produced mostly by rock solution as the dominant process. As a consequence, earth's surface is covered by exotic irregular morphologies, like lapiaz, closed depressions (*dolinas*), sinks, potholes and the like, with the development of underground caves. This implies that the surface drainage network is usually poorly developed or simply does not exist at all, while vertical infiltration of rain waters forms an underground drainage system where water flows through fissures, galleries and caves. The studied area is located 30 km to the south of Havana City (Cuba) in the so called Havana-Matanzas Karstic Plain composed of porous, fractured and heavily karstified limestones of Middle Miocen age with abundance of a variety of clay minerals. Under the high temperatures and humidity typical of tropical conditions, weathering processes develop an overburden composed by reddish insoluble materials (*tera rossa*) coming from solution processes on the limestones.

Negative karst forms on the surface (the lapiaz, sinks, dolinas, etc.) are partially or totally covered by an overburden of variable depth. These forms often connect with caves in the underground, some of them big. Direct detection is very difficult or impossible and geophysical methods are necessary, as they usually are for tasks like geological mapping and construction of cross sections. This is a very important problem from the point of view of civil engineering, geological engineering and environmental studies in general in this kind of regions.

In a selected square area (340 m side), geophysical methods complemented with a detailed topographic survey [Valdés and Gil, 84] were used with the purpose of characterizing the shallower horizons of the geological section and their relation with underlying karstic phenomena. Targets were zones of intense fracture and karstification, filled depressions, overburden pockets and the presence of underground caves. The set of geophysical methods included the spontaneous electric potential of earth's surface, the gamma radioactive intensity and the electromagnetic field in the VLF region of the spectrum [Valdés and Gil, 84]. In particular, two different surveys of spontaneous electric potential were performed, in the dry and rainy season respectively, since strong negative anomalies are due to infiltration potentials associated with electrochemical processes taking place as water infiltrates into the underground via fissures and joints. These four measurements, along with the surface topography, constitute the five variables to be used by the neural models. The complexity of these measured geophysical fields in the area is illustrated, as an example, by the distribution of gamma ray intensity and the surface topography. While radioactivity is highly noisy, topography shows few features. They are shown in Figs. (7.6) and (7.7), respectively.

Geophysical survey methodologies consider independent sets of measurements in order to account for the different kind of errors and the natural variability of such kind of information. In order to be considered acceptable, each survey must have an error no greater than 5% when comparing the original and the independent measurements. This means that the reported values of all geophysical fields (i.e, the available data), have an inherent uncertainty which must be considered. In the area, a gentle variation in geological conditions for both the bedrock and the overburden was suspected by geologists and also a large underground cave with a single gallery was known to exist in the central part of the area. The cave has about 300 meters long with cross sections ranging from less than one square meter in the narrowest part, to chambers having 40 meters wide and 30 meters high, reaching the surface in the form of a gorge in the bottom of a depression.

An isolation of the different geophysical field sources was necessary in order to focus the study on the contribution coming from underground targets, trying to minimize the influence

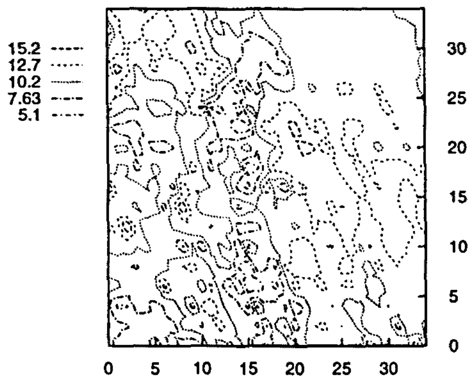


Figure 7.6: Distribution of gamma ray intensity in the studied area.

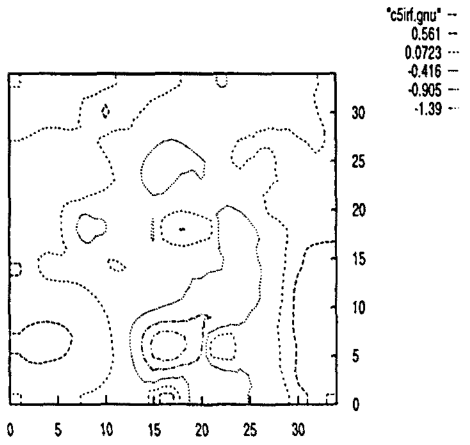


Figure 7.7: Surface topography of the studied area.

of both the larger geological structures, and the local heterogeneities. According to the *a priori* geological ideas, each geophysical field was assumed to be described by the following additive two-dimensional model composed by trend, signal and random noise:

$$f(x,y) = t(x,y) + s(x,y) + n(x,y)$$

where f is the physical field, t is the trend, s the signal, and n the random noise component, respectively. In order to isolate an approximation of the signals produced by the underground target bodies, a linear trend term $t'(x,y) = c_0 + c_1 x + c_2 y$ was computed (by least squares) and subtracted from the original field. The residuals $r(x,y) = f(x,y) - t'(x,y)$ were then filtered by direct convolution with a low pass finite-extent impulse response two-dimensional filter in order to attenuate the random noise component [Dudgeon and Mersereau, 84]. Such convolution is given by:

$$s'(x,y) = \sum_{k_1=-N}^{N} \sum_{k_2=-N}^{N} h(k_1,k_2)\, r(x-k_1, y-k_2)$$

where $r(x,y)$ is the residual, $s'(x,y)$ is the signal approximation and $h(k_1,k_2)$ is the low-pass zero-phase shift digital filter.

7.5.2 Experiments

In order to study the behavior of the heterogeneous neural models, a comparison was made regarding geological-geophysical accuracy of classification. This kind of knowledge, as well as results from previous non-supervised classification techniques [Valdés, 97] had shown the existence of two multivariate populations within the studied area: one representing more karstified zones with large interconnected underground cavities, and another in which karstification is not so intense. Since the hypothesis of two hyperspherical classes in pattern space was tenable, and the purpose of this work is to assess the relative merits of the three considered neuron models (classical, heterogeneous and fuzzy heterogeneous) in the task at hand (imprecise classification using data which are also imprecise), a network consisting of a *single* neuron was the architecture selected. Clearly, other multilayer layouts are possible and should deserve future attention, but this is a useful reference for initial comparisons. Together with a small training set (relative to test), it should make the problem much more difficult than it really is, so the differences should be more evident.

The experiments were conceived in two phases as follows. In phase one, a comparison is made between the classical real P-neuron and the H-neuron with real inputs and weights. In a second stage, the latter is compared to a fuzzy H-neuron. Also, the experiments were designed following geological criteria. From this point of view, it is known that the number of observable caves in any karstic area is only a small fraction of the actually existing ones, making class structure itself *imprecise*, a situation usual in complex problems like those from environmental studies. Moreover, there are no sharp boundaries between rock volumes containing caves and those containing less or none. One could say that the notion of "caveness" degrades smoothly, which is another reason to use fuzzy models.

The training was supervised (in the usual mean-squared-error sense) by the information given by the topographic map of a large cave present in the area, so that those surface measurement points lying *exactly above* the known cave were considered as class 1 patterns and those outside as belonging to class 2 –the resulting cave is shown in Fig. (7.8). This procedure for class assignment was too conservative but, otherwise, one would have been forced to provide as output the exact caveness degree for each point. This value, besides being very difficult to estimate, would have introduced a strong subjective bias. The computation of this degree is precisely the task we want the model to perform.

Selected data from the northern half were used for training, whereas the rest was used to test the trained network (consisting of a single neuron only). More precisely, the training set was composed by the 31 points from the northern half located exactly above the known cave (representing class 1), plus 32 others homogeneously distributed in the east-west sides –see

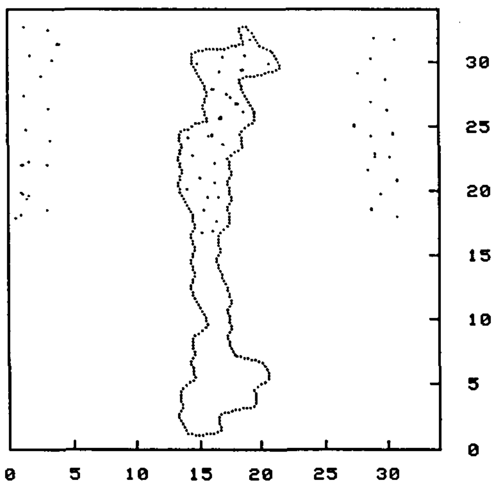


Figure 7.8: The known cave borders: see text for an explanation of what is considered as cave and what is not. Dots indicate the (approximate) location of the points used for training. Units are in tens of meters.

Fig. (7.8). As test set we used the remaining 567 patterns from the whole area.

Phase 1

Here we have a classical real-valued neuron (in this study, having scalar product as net input and the hyperbolic tangent as a squashing activation function). The training procedure for this neuron is a combination of conjugate gradient with simulated annealing [Ackley, 87], whereas the heterogeneous neuron is trained using a standard genetic algorithm with the following characteristics: binary-coded values, probability of crossover: 0.6, probability of mutation: 0.01, number of individuals: 50, linear scaling with factor $c = 1.5$, selection mechanism: stochastic universal, replace procedure: worst individual.

The results obtained by both models are shown in Figs. (7.9) and (7.10), respectively, where caveness prediction is plotted in five equally-spaced α-cut sets. Clearly, the distribution of the two-dimensional sets for the heterogeneous neuron reflects much better the distribution of the known cave than the classical neuron, for various reasons. First, the classical neuron fails to detect the southernmost part of the known cave, whereas the heterogeneous counterpart does.

Second, the classical neuron predicts complete cave areas in the south-east and south-west zones, which are misleading. These are also signaled by the heterogeneous neuron, but always with a degree of 0.5 or less. The only exception is a small area located in coordinates $(7 - 8, 12 - 15)$, where other geophysical methods (seismic and DC-resistivity) not used in this study had signaled cave anomalies. And third, the general layout of the actual cave (north-south main axis, slightly bended and narrower in the middle part) is better reflected by the heterogeneous neuron.

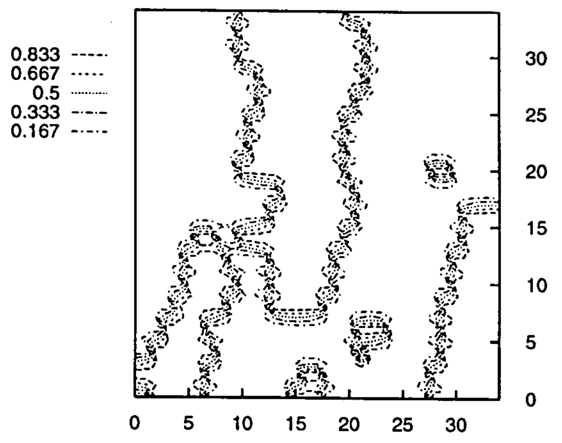


Figure 7.9: Results of phase 1: α-cut sets for the classical neuron.

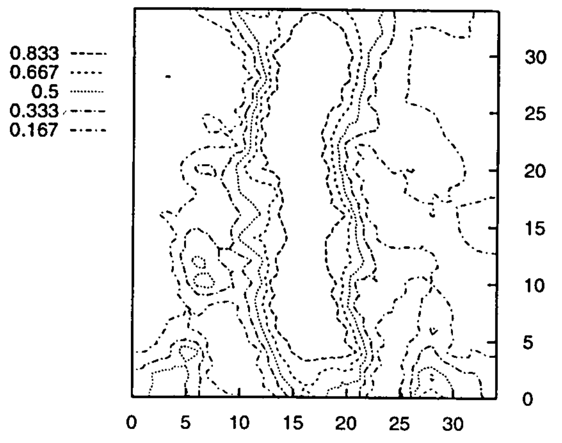


Figure 7.10: Results of phase 1: α-cut sets for the heterogeneous neuron.

Phase 2

In a second stage, a fuzzy heterogeneous neuron was trained in the same experiment setting, but this time using fuzzy inputs. This means that all neuron weights were *fuzzy sets* (actually triangular fuzzy numbers), and both training and test vectors represented by fuzzy numbers (the mode was given by the corresponding observed value, and the spread by a $\pm 5\%$ of it). This is in accordance with the upper bound of the measurement errors reported for the geophysical field surveys made. It should be noted that this criteria was conservative, since some surveys actually had less than 5% of error.

The results -shown in Fig. (7.11) are again qualitatively satisfactory, in what regards to the general layout of the cave. But now a quantitative factor comes into the picture: the cave is much more neatly defined, a fact that shows in two ways: first, the different α-cut sets are much closer, showing a gradual but firm transition between classes of 2 units in the map on average (roughly 20 meters in the field) —a very reasonable value. That is, this narrow belt w.r.t. the trace of the known cave represents the *transition* zone between the rock volumes,

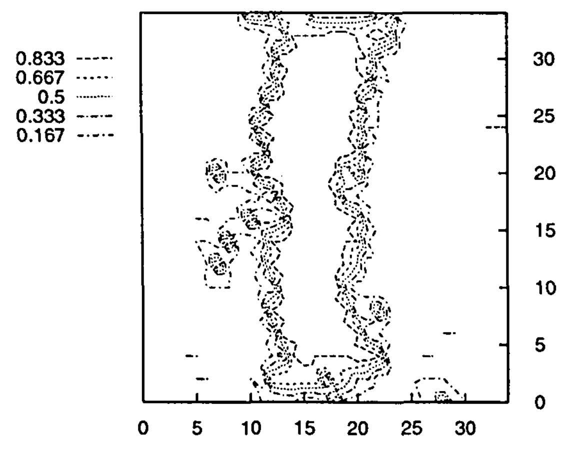


Figure 7.11: Results of phase 2: α-cut sets for the fuzzy heterogeneous neuron.

more and lesser affected by big underground cavities. Second, the extensive anomalous zones predicted by the heterogeneous neuron in the eastern and south-western zones completely disappear, with the exception of a small region in coordinates $(25-30, 0-2)$, which should be specifically checked. What is more, the strongest region where the presence of a secondary cave is signaled by the fuzzy heterogeneous neuron is precisely the one aforementioned and confirmed to exist by other means. This a nice result, since allowing imprecise inputs and weights for all of the five variables does not degrade the overall performance. On the contrary, the results can be said to be even more accurate. Notice that all of the neurons are using the same small training set but, in practice, this situation is less favourable for the fuzzy neuron, which would need an enlarged training set to compensate for the imprecision.

7.5.3 Conclusions

Experiments have been made with complex multivariate space-dependent data -coming from a real world problem in the domain of environmental studies. The results have shown that better models can be found by treating data with its natural imprecision, rather than considering them as crisp quantities, as is usually the case. In this respect, allowing imprecise inputs and using heterogeneous (fuzzy) neurons based on similarity yields more accurate representations (because of their greater flexibility) than those found via classical crisp real-valued models, in a problem for which one is not so much interested in crude train/test set classification errors but in its ability to model the imprecise structure of the domain.

7.6 Classification of natural waters

7.6.1 Preliminaries

This is again an investigation in the field of environmental sciences, namely, the geochemical study of natural waters in the Arctic (Spitzbergen). The elements studied include classification accuracy, the effect of working with crisp or fuzzy inputs, the use of traditional scalar product *vs.* similarity based functions, and the presence of missing data. A description of the problem follows.

During the scientific expedition Spitzbergen'85, organized by the University of Silesia (Poland), a scientific team composed of specialists from this university, the National Center for Scientific Research (Cuba), and the Academy of Sciences of Cuba, performed glaciological and hydrogeological investigations in several regions of the Spitzbergen island (Svalbard archipelago, about 76^oN to 80^oN). The purpose was to determine the mass and energy balance within experimental hydrogeological basins, the study of the interaction between natural waters and rock-forming minerals in the severe conditions of polar climate and their comparison with similar processes developed in tropical conditions. This has been a long-term research of several Polish universities (Silesia, Warsaw and Wroclaw) and the Polish Academy of Sciences since the First Geophysical Year in 1957, and represents an important contribution to the evaluation of the impact of global climatic changes.

In this respect, almost all of the studied glaciers had a negative mass balance and are experimenting severe recessions with an increasing trend. Thus, massive meltings are taking place in polar summers, with the corresponding acceleration of denudation rates, both mechanical and chemical. These affect glaciers, morraines, the permafrost, the fjords, etc, and there are complex interactions due to peculiar geological, geomorphological and hydrogeological conditions which, in the end, reflect in water geochemistry.

Variable	Max	Min	Mean	SVar
Temperature	12.50	-0.20	1.00	2.49
pH	9.00	5.00	7.34	0.46
Conductivity	946.00	12.00	205.63	35183.09
Hydrocarbonate	2.95	0.15	0.89	0.29
Chloride	1.76	0.04	0.26	0.04
Sulphate	6.66	0.06	0.81	0.90
Calcium	5.80	0.00	0.92	0.94
Magnesium	4.00	0.00	0.47	0.26
Sodium-Potasium	1.86	0.18	0.57	0.06

Svar: sample variance.

Table 7.8: Basic statistical descriptors for the available variables.

In this study, a collection of water samples were taken from different hydrogeological zones in two Spitzbergen regions (the Grondfjord and the Hornsund fjords). They were representative of many different zones: subglaciar, supraglaciar, endoglaciar, springs (some hydrothermal), lakes, streams, snow, ice, the tundra and coastal. Among the physico-chemical parameters determined for the water samples, the following nine were used for the present study: temperature, pH, electrical conductivity, hydrocarbonate, chloride, sulphate, calcium, magnesium and sodium-potasium. Basic statistical descriptors are shown in Table (7.8).

Previous geochemical and hydrogeological studies of these data had shown a relation between the different hydrogeological conditions present in Spitzbergen and the chemical composition of their waters, reflecting the existence of several families of waters. That is, an indirect assessment of their hydrogeological origin is in principle possible from the information present in the geochemical parameters, thus enabling the use of a learning algorithm [Fagundo, Valdés and Pulina, 90], [Fagundo, Valdés and Rodríguez, 96].

7.6.2 Experiments

General Information

The available set of $N = 114$ water samples from Spitzbergen, corresponding to $c = 5$ hydrogeological families of waters, was used for comparative studies of supervised classification accuracy using different neural architectures, described below. To express the distribution of samples among classes we introduce the notation n_k to denote that there are n samples of class k. This way, the actual distribution was $37_1, 29_2, 10_3, 11_4, 27_5$. *Default accuracy* (relative frequency of the most common class) is then 37/144 or 32.5%. *Entropy*, calculated as $-\sum_{k=1}^{c}(n_k/N)\ log_2(n_k/N)$, is equal to 2.15 bits. There were no missing data and all measurements were considered to have a 5 % of imprecision w.r.t. the reported value. This aspect will be taken into account when considering uncertainty in the form of fuzzy inputs, since the fact that the physical parameters characterizing the samples as well as their chemical analysis were done *in situ* –in the extremely hard climatic and working conditions of the Arctic environment– makes them particularly suited to a kind of processing in which uncertainty and imprecision are an explicit part of the models used. Accordingly, feed-forward networks composed of a first (hidden) layer of heterogeneous neurons, collected in an output layer by classical P-neurons is the basic architectural choice for this case study. These *hybrid* architectures will be compared to their fully classical counterparts –under the same experimental settings– in order to assess their relative merits.

We recall the notation explained at the beginning of the chapter concerning architectural settings, with q_x denoting a *single* layer of q neurons, where possibilities for x are:

n Classical: real inputs, scalar-product net input and logistic activation.

h Heterogeneous: real inputs, similarity-based net input and (adapted) logistic activation.

f Fuzzy heterogeneous. Triangular fuzzy inputs, obtained from the original crisp reported value by adding a 5% of imprecision. Similarity-based net input and (adapted) logistic activation.

Accordingly, $p_x q_y$ denotes a feed-forward network composed of a hidden layer of p neurons of type x and an output layer of q neurons of type y. For example $4_h 5_n$ is a network composed of a hidden layer of 4 neurons of type h and an output layer of 5 neurons of type n. All units use the logistic as activation.

All neural architectures are trained using a standard genetic algorithm (SGA), as mentioned at the beginning of the chapter. Additional characteristics are: number of individuals: 52 and the fact that the algorithm was stopped unconditionally after 5,000 generations or if there was no improvement for the last 1,000. This last criterion helps in evaluating the goodness of the architecture being trained and saves unuseful computing time.

Experimental Settings

In the present study, *all* models (including the classical feed-forward one) were trained using exactly the same procedure and parameters in order to exclude this source of variation from the analysis. Of course, fully classical architectures need not be trained using the SGA. They could instead be trained using any standard (or more sophisticated) algorithm using gradient information. However, this would have made direct comparison much more difficult, since one could not attribute differences in performance exclusively to the different neuron models, but also to their training algorithms. The experiment settings were the following:

Training regime The training set was composed of 32 representative samples (28% of the whole data set), whereas the remaining 82 (72%) constituted the test set, a deliberately chosen hard split for generalization purposes. Class distribution is $8_1, 7_2, 5_3, 5_4, 7_5$ in training and $29_1, 22_2, 5_3, 6_4, 20_5$ in test. Default accuracies are 25.0% and 35.4%, respectively.

Architectures We will explore the following architectures: $5_x, 2_x5_n, 4_x5_n, 6_x5_n$ and 8_x5_n, for x in n, h, f. Note that the output layer is always composed of five units, one for each water class.

Number of runs Every architecture was allowed $R = 5$ runs varying the initial population. All of them were included in the results.

Weight range The weights concerning units of type n were limited to be in the range $[-10.0, 10.0]$, to prevent saturation, whereas heterogeneous weights adopt (by definition of the heterogeneous neuron) the same range as their corresponding input variable.

Error functions The target error function to be minimized by the training algorithms is the usual **least squared error**, defined as follows:

$$\text{LSE} = \sum_i^p \sum_j^m [\hat{y}_j^i - y_j^i]^2$$

where y_j^i is the j-th component of the output vector $\vec{y^i}$ computed by the network at a given time, when the input vector $\vec{x^i}$ is presented, and $\hat{y}_j^i = \phi_j(\vec{x^i})$, is the target for x_j^i, where ϕ_j represents the characteristic function for class j. The error displayed will be the **mean squared error**, defined as $\text{MSE} = \frac{1}{mp}\text{LSE}$, where m is the number of outputs and p the number of patterns.

Presentation of the Results (I)

Let the classification accuracy for training (TR) and test (TE) sets, calculated with a winner-take-all strategy, be denoted $\mathrm{CA}_{\mathrm{TR}}(r)$ and $\mathrm{CA}_{\mathrm{TE}}(r)$, respectively, for a given run r. The errors $\mathrm{MSE}_{\mathrm{TR}}(r)$ and $\mathrm{MSE}_{\mathrm{TE}}(r)$ are similarly defined. For each neural architecture, the following data is displayed:

Accuracy: Mean classification accuracy on training $\mathrm{MCA}_{\mathrm{TR}} = \frac{1}{R}\sum_{run=1}^{R} \mathrm{CA}_{\mathrm{TR}}(run)$, mean classification accuracy on test $\mathrm{MCA}_{\mathrm{TE}} = \frac{1}{R}\sum_{run=1}^{R} \mathrm{CA}_{\mathrm{TE}}(run)$ and best classification accuracy (BCA) defined as the pair $< \mathrm{CA}_{\mathrm{TR}}(r), \mathrm{CA}_{\mathrm{TE}}(r) >$ with higher $\mathrm{CA}_{\mathrm{TE}}(r)$.

Error: Mean MSE in training defined as $\mathrm{MMSE}_{\mathrm{TR}} = \frac{1}{R}\sum_{run=1}^{R} \mathrm{MSE}_{\mathrm{TR}}(run)$, sample variance in training defined as

$$\mathrm{SVMSE}_{\mathrm{TR}} = \frac{1}{R-1}\sum_{run=1}^{R} [\mathrm{MSE}_{\mathrm{TR}}(run) - \mathrm{MMSE}_{\mathrm{TR}}]^2$$

and similarly defined values $\mathrm{MMSE}_{\mathrm{TE}}$ and $\mathrm{SVMSE}_{\mathrm{TE}}$ for the test set.

The results are collectively shown in Table (7.9). As an additional reference measure of performance, the k-nearest neighbours algorithm (with $k = 5$) is also run on the data -with the same train/test partition- yielding an accuracy in test equal to 58.5%.

Net	Training			Test			BCA
	$\mathrm{MCA}_{\mathrm{TR}}$	$\mathrm{MMSE}_{\mathrm{TR}}$	$\mathrm{SVMSE}_{\mathrm{TR}}$	$\mathrm{MCA}_{\mathrm{TE}}$	$\mathrm{MMSE}_{\mathrm{TE}}$	$\mathrm{SVMSE}_{\mathrm{TE}}$	
5_n	54.4%	0.1075	2.4e-04	46.6%	0.1661	5.8e-05	65.6% 53.7%
5_h	66.3%	0.1084	8.0e-06	67.1%	0.1202	1.6e-05	75.0% 76.8%
5_f	99.4%	0.0338	3.0e-06	69.3%	0.0917	1.1e-05	100% 75.6%
2_n5_n	41.9%	0.1314	4.3e-04	45.4%	0.1420	4.9e-04	68.8% 67.1%
2_h5_n	71.9%	0.0968	2.0e-04	69.5%	0.1088	2.6e-04	81.3% 85.4%
2_f5_n	86.3%	0.0635	1.2e-04	71.7%	0.0995	9.3e-05	81.3% 81.7%
4_n5_n	70.6%	0.0785	6.1e-05	58.3%	0.1288	3.5e-05	71.9% 61.0%
4_h5_n	90.0%	0.0614	1.0e-05	79.0%	0.0786	2.9e-05	93.8% 82.9%
4_f5_n	98.1%	0.0201	1.4e-04	81.2%	0.0620	1.3e-04	100% 86.6%
6_n5_n	70.0%	0.0802	2.6e-04	55.4%	0.1389	7.7e-05	81.3% 58.5%
6_h5_n	91.3%	0.0508	5.0e-05	83.7%	0.0803	5.6e-05	93.8% 87.8%
6_f5_n	100%	0.0106	3.0e-06	84.9%	0.0553	1.1e-05	100% 90.2%
8_n5_n	87.6%	0.0396	5.7e-05	63.7%	0.1231	2.2e-04	87.5% 68.3%
8_h5_n	93.8%	0.0456	1.9e-05	86.6%	0.0603	4.0e-05	93.8% 90.2%
8_f5_n	100%	0.0064	4.0e-06	80.5%	0.0541	4.3e-05	100% 84.1%

Table 7.9: Results of the experiments. See text for an explanation of entries.

Analysis of the results (I)

As previously stated, the experiments were oriented to reveal the influence of several factors:

a) the kind of neural model used (heterogeneous *vs.* classical)

b) the effect of considering imprecision (fuzzy inputs *vs.* crisp inputs), and

c) the effect of missing data in the test set.

The effect of factor (a) can be assessed by comparison, for all the architectures, of the first entry against the other two, column by column. The effect of (b) reflects in the difference between the second vs. the third. The effect of (c) will be discussed later on.

Single-layer architectures

Let us begin by analysing the results for the architectures with no hidden units, that is, the first three rows of table 7.9. The approximation capabilities of the three neuron models can be seen by comparing the value of MCA_{TR}. The mean error MMSE_{TR} is also a good indicator. The robustness (in the sense of expected variability) can also be assessed by the value of SVMSE_{TR}. It can be seen how the heterogeneous neurons are in general better and much more robust than the classical one. Especially, the fuzzy neuron can learn from the data set to almost perfection very robustly. Similar results are achieved in the test set. Again, an increasing accuracy and decreasing errors and variance indicate an overall better performance. However, the f units are clearly overfitting the data, a fact that shows in the highly unbalanced TR and TE accuracy ratios (both in average and in the best pair BCA).

Multi-layer architectures

For the four groups of architectures selected (the p_x5_n), there are two aspects amenable to be discussed. First, the relative behaviour of elements of the form p_x5_n, for a fixed p. Second, their relative behaviour for a fixed x. These two dimensions will collectively give light on any coherent behaviour present in the results.

To begin with, it can be seen that for all the architectures 2_x5_n, 4_x5_n, 6_x5_n and 8_x5_n, as we go through the sequence n, h, f, the behaviour is *consistent*: mean accuracies increase, and mean errors and their variances decrease, *both* in training and in test, with the only exception of the test accuracy in the case 8_x5_n, due to an overfit. This shows a general superior performance of h neurons over n neurons, and of f neurons over h. The absolute differences between neuron models are also noteworthy. In all training respects, the p_f5_n families show very good approximation capabilities, explaining the 100% of the TR set *starting from* $p = 4$ in BCA and from $p = 6$ in MCA_{TR}. This trend is followed –to a less extent– by the p_h5_n. The same consistent behaviour is observed in all test indicators. Here the two heterogeneous families show a similar behaviour, with the f neurons slightly above the h ones, until for $p = 8$, the architectures p_f5_n end up overfitting the data so strongly that their performance in test begins to fall down.

As for the second aspect, p_x5_n fixing x, it can be checked that all neuron models show an increasing approximation ability when the number of hidden neurons is increased, as can reasonably be expected. In conclusion, for all of the architectures it is clear that the use of heterogeneous neuron models leads to higher accuracy rates in the training and test sets. Moreover, when imprecision is allowed by accepting that each value is endowed with the above mentioned uncertainty, the fuzzy heterogeneous model also outperforms its crisp counterpart.

Presentation of the Results (II)

The neural nets obtained in the previous experiment can now be used to assess the effect of factor (c), the influence of missing values in the data. The purpose of this experiment is twofold: first, it is useful studying to what extent missing information degrades performance. This is an indication of robustness and is important from the point of view of the *methods*. Second, in this particular problem, studying the effect of missing data is very interesting, because it can give an answer to the following questions:

1. What *predictive* performance could we expect if we do not supply all the information? (and just a fraction of it).

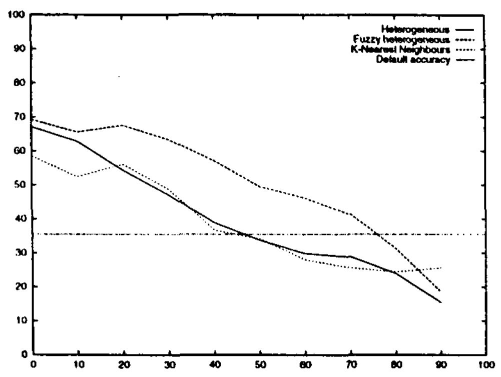
2. What would have happened had we presented to the net *incomplete training* information from the outset?

This scenario makes sense in our case study, for which a rich set of complete data may be impossible to obtain, because of lack or damage of resources, physical or practical unfeasibility, lack of time, climatic conditions, etc. Note that it is not that a particular variable cannot be measured (we could readily remove it) but that *some* realizations of (potentially) *all* variables may be missing. These experiments were performed with the same nets found in the previous section. This time, however, they were each run on different test sets, obtained by artificially and randomly (with a uniform distribution) adding different percentages of missing information. These percentages range from 10% to 90%, in intervals of 10%. These experiments were not performed for the p_n5_n architectures, for they do not directly accept missing information. Although there are estimation techniques, they are not an integrated part of the models, and would have introduced a bias. The results are presented, for the whole set of heterogeneous architectures displayed in Table (7.9), in a graphical form, through Figs. (7.12, a) to (7.12, e). The x-axis represents the total percentage of missing values in the test set, while the y-axis stands for the MCA_{TE} (that is, again, data shown for each point is the average for $R = 5$ runs). The horizontal line represents the size of the major class (35.4%) to be taken as a reference, and the same k-nearest neighbours algorithm is run and shown in Fig. (7.12, a).

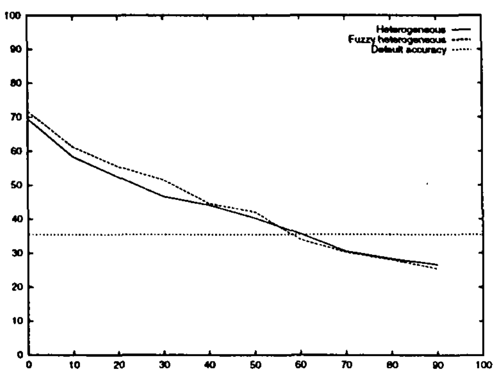
Analysis of the Results (II)

Both neuron models h, f are very robust, a fact that shows in the curves, which follow a quasilinear decay. The accuracies are consistently higher for the fuzzy model than for the crisp counterpart for all the network architectures, again showing that allowing imprecision increases effectiveness and robustness. Performance, in general, is well above the default accuracy until a 50% – 60% of missing information is introduced. In many cases, mean classification accuracy is still above for as far as 70% – 90%, which is very remarkable.

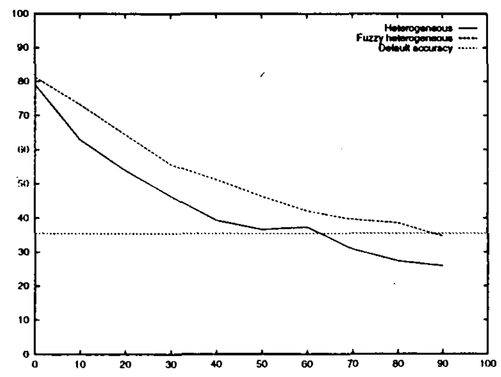
The last figure –Fig. (7.12, f)– shows the effect of a different training outset. Choosing what seems to be the best group of architectures for the given problem, the 6_h5_n and 6_f5_n, these networks were trained again, this time with a modified training set: adding to it a 30% of missing information, in the same way it was done for the test set, and using them



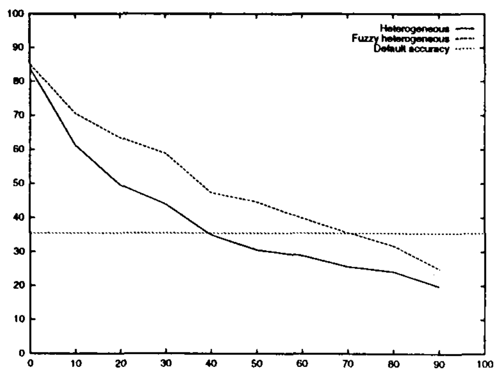
(a) 5_h and 5_f



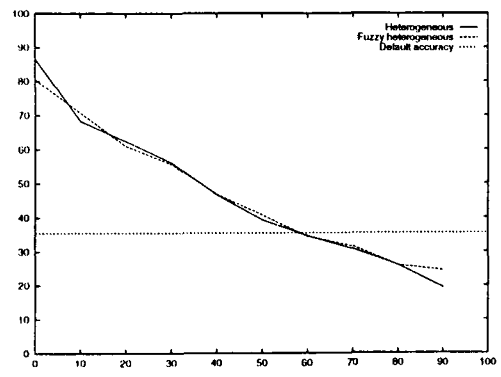
(b) 2_h5_n and 2_f5_n



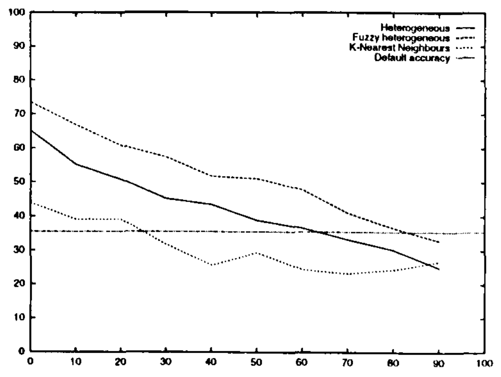
(c) 4_h5_n and 4_f5_n



(d) 6_h5_n and 6_f5_n



(e) 8_h5_n and 8_f5_n



(f) 6_h5_n and 6_f5_n

Figure 7.12: Increasing presence of missing data in test. Mean test classification accuracy for the heterogeneous (p_h5_n) and fuzzy heterogeneous (p_f5_n) families. (a) 5_h and 5_f (b) 2_h5_n and 2_f5_n (c) 4_h5_n and 4_f5_n (d) 6_h5_n and 6_f5_n (e) 8_h5_n and 8_f5_n (f) Mean test classification accuracy for 6_h5_n and 6_f5_n when trained with a 30% of missing information. x-axis: percentage of missing values in test. y-axis: percentage of accuracy in test.

again to predict the increasingly diluted test sets. As usual, the horizontal line represents the size of the major class and k-nearest neighbours performance is also shown. Training accuracies were this time lower (as one should expect) and equal to $\text{MCA}_{\text{TR}} = 88.8\%$ for 6_h5_n and to $\text{MCA}_{\text{TR}} = 96.3\%$ for 6_f5_n. However, the differences with previous performance are relatively low. Some simple calculations show that, although the amount of data is 70% that of the previous situation, new accuracies are as much as 97.3% and 96.3% of those obtained, with full information, for 6_h5_n and 6_f5_n, respectively. Performance in test sets is also noteworthy: although the new curves begin at a lower point than before, the degradation is still quasilinear. What is more, the slope of this linear trend is lower (in absolute value), resulting in a slight raising up of the curves (in both of them).

7.6.3 Conclusions

This last group of experiments, carried out with data coming from a real-world problem in the domain of environmental studies, have shown that allowing imprecise inputs, and using fuzzy heterogeneous neurons based on similarity, yields much better prediction indicators –mean accuracies, mean errors and their variances and absolute best models found– than those from classical crisp real-valued models. Especially noteworthy is the graceful degradation in the increasing presence of missing information, a particular feature of heterogeneous models that should not be overlooked, since it is a very desirable feature in any model willing to be useful in real-world problems.

7.7 General conclusions

In this chapter, heterogeneous neural networks have been successfully tested in a variety of tasks. They have been compared to the standard multi-layer perceptron and shown to lead to a remarkable increase in performance when departing from this classical neuron.

The experiments show how the use of fuzzy heterogeneous networks can significatively increase the accuracy of the models obtained. The moral seems to be that better models can be found by treating data with its natural imprecision, rather than considering them as crisp quantities, as is usually the case. Allowing imprecise inputs and weights in a neural model translate in more accurate representations, because of their greater flexibility in front of those found with crisp models. This is so to the point that the fuzzy models have been observed to have a strong tendency to overfit the data, a fact that shows in all the experiments presented. This is not dangerous as long as it entails a careful control of the proposed architectures, and the use of a model selection technique. Such methodology is followed, together with the use of validation sets in the training process, in the experiments of chapter (9).

In all, these results for heterogeneous networks confirm the features observed in other studies, concerning their mapping effectiveness and their robustness in the presence of uncertainty and missing data. Their ability to directly consider imprecise data and their remarkable resistance under those circumstances deserve special attention, due to their implications for real-world problems and from the point of view of neurofuzzy systems.

Chapter 8

A study in Wastewater Treatment Plants

Lo que el hombre hace no puede hacerlo
la naturaleza; si bien el hombre, para hacerlo,
se vale de todas las leyes de ésta.

Louis Kahn

The control and prediction of Wastewater Treatment Plants (WWTP) poses an important goal in order to always keep the system in stable operating conditions under a wide range of working circumstances, thereby avoiding the risk of breaking the environmental balance. In this respect, the availability of models characterizing WWTP behaviour as a dynamic system is a necessary first step. However, due to the high complexity of the involved processes and the heterogeneity, incompleteness and imprecision of WWTP data, finding suitable models entails substantial problems.

In *system identification*, the studied model is provided with information coming from the system history of behaviour, possibly in the form of a fixed number of appropriately delayed inputs and outputs. The interest in making models out of observations is in its application to the prediction, control or simply in a better understanding of the modelled system. In this Chapter, several acceptable submodels are found, able to characterize WWTP behaviour in a statistically satisfactory sense and performing better than other well-established techniques. The material presented has been compiled from [Belanche *et al.*, 98b], [Belanche *et al.*, 99c], [Belanche *et al.*, 99b] and [Belanche *et al.*, 00].

8.1 Introduction

Dirty water is both the world's greatest killer and its biggest single pollution problem [Lean and Hinrichsen, 94]. The large amount of wastewater generated in industrialized societies is one of the main environmental pollution aspects that must be seriously considered.

New Directives and Regulations have guaranteed the construction of specific plants to treat these wastewaters, being the **activated sludge** process the one most extensively used nowadays.

The proper management of wastewaters in modern industrialized societies is hence not only an option, but a necessity. The main objective is to maintain natural water systems at as high a quality level as possible, and to ensure equilibrium between supply and demand through a rational use and management of water resources. Moreover, the wastewater treatment helps to reach the attainment of rivers as biological corridors, translating in a good quality of life for animals and vegetals living in the water.

Wastewater coming from different municipal uses contains a wide variety of contaminants. Among them, the most commonly found in municipal wastewater are total suspended solids (TSS), organic matter —measured as biochemical oxygen demand (BOD) and chemical oxygen demand (COD)— pathogens, and nutrients. The basis of wastewater treatment processes lies in oxidizing biodegradable organics from raw water into stabilized, low-energy compounds, maintaining a mixture of microorganisms and supplying oxygen by aerators [WEF, 96].

The European directive of the Council 91/271/EEC on urban wastewater treatment sets forth a precise standard regarding the degree of treatment and purification to be required for diverse types of population centers. It foresees wastewater treatment for all urban concentrations greater than $2,000$ inhabitants-equivalents before the end of year 2005, excepting those of less than 150,000 discharging in the sea. To achieve these purposes, the autonomous Government of Catalonia has drawn up its *Pla de Sanejament.* To date, more than 225 WWTP have already been built in Catalonia, treating an average daily wastewater flow superior to $2,000,000\ m^3$.

Although it is very important to ensure the quality of the treated wastewater prior to discharge, the correct control and operation of the process carried out in the WWTP is not a well-established task. Some of the factors which affect the real-time control of the process are [WEF, 92]:

- the biological nature of the process, involving the presence of a true trophic web;
- the great complexity and variability of the influent composition;
- the lack of reliable on-line sensors and signals;
- the delay of the analytical results from the laboratory: minutes, hours or even days, according to the different TSS (30 minutes), COD (2 hours) or BOD (5 days) determinations, and
- the dynamically changing state of the process, due to the fact that the involved factors have very different dynamics (e.g., the inflow characteristics change in seconds, the quality of the outflow changes in hours and the biomass changes at a pace that may take days).

Research contributions in this field have been formulated from many different points of view. However, a direct cause-effect relationship for WWTP performance has been established only in a few cases and, even in those, experimental results could lead to contradictory conclusions [Capodaglio *et al.*, 91], avoiding the formulation of deterministic cause-effect relationships that could be used as prediction models. The identification of a model that could predict in real-time and with reasonable accuracy is thus of great practical importance in view of a potential improvement of treatment plant efficiency and cost savings [Novotny *et al.*, 90].

To tackle such task, several staged studies have been performed towards the development of input-output behaviour models for WWTP, in which the temporal behaviour of the main outgoing variables (COD-AT, BOD-AT and TSS-AT, see below) is acceptably captured and reproduced. The long-term aim of this work -which is only in its initial stages- is to find a model capable of short-term prediction, taking into account only the actually relevant variables and accommodating some of the characteristics of real WWTP data: imprecision, heterogeneity, and high incidence of missing information.

The chapter is organized as follows. Section 2 briefly describes the basics of a WWTP while Section 3 describes the problem at hand, the particular WWTP under study, and its characteristics. Section 4 overviews the experiments to follow and the used techniques other than those already introduced in previous parts of the work. The next three Sections deal with the experimental part itself, the experiment setup and the obtained results. Finally, Section 8 presents the conclusions of the chapter.

8.2 Basics of a WWTP

The definition of waste water includes any combination of liquid flow coming from human establishments, public installations and industrial settings, eventually supplemented by underground, surface or rain waters. In a WWTP, different stages -physical, chemical and biological- are combined to form the process diagram. The global process can be best described following the water flow-line [Balaguer *et al.*, 98]:

1. A *pre-treatment* stage, to eliminate or reduce the impact of the bigger solid material, consisting on a sequence of screens followed by the removal of grit, sand and grease, the former two by sedimentation and the latter thanks to its floutability.

2. A *primary treatment* stage, where the water is allowed to rest for a while (some hours) in order to settle part of the organic matter, and the grit or inorganic matter not eliminated in the pre-treatment.

3. A *secondary treatment* stage, the most important part of the process, in which a population of specific microorganisms degrades the organic matter dissolved in the water. This treatment takes place at the oxidation ditchs or *bioreactors*. Roughly speaking, the microorganisms use the oxygen present in the water to consume the substrate (the organic molecules). As a result, the microorganisms get the necessary amount of living energy and are able to reproduce.

4. Finally, a second settlement process is carried out in order to achieve a good separation between the treated water and the biomass. A clarification and chlorination process are performed at the end, prior to the release of water to a natural emissary.

5. As a side effect, WWTP generate a large amount of a by-product called *sludge* (basically a liquid mixture of microorganisms and particulated organic matter) that must also be treated. Thickening, stabilisation and dewatering are the three main unit operations to convert the sludge into a stable product for ultimate disposal.

A scheme of a WWTP water flow-line providing primary and secondary treatment using the activated sludge process is illustrated in Fig. (8.1).

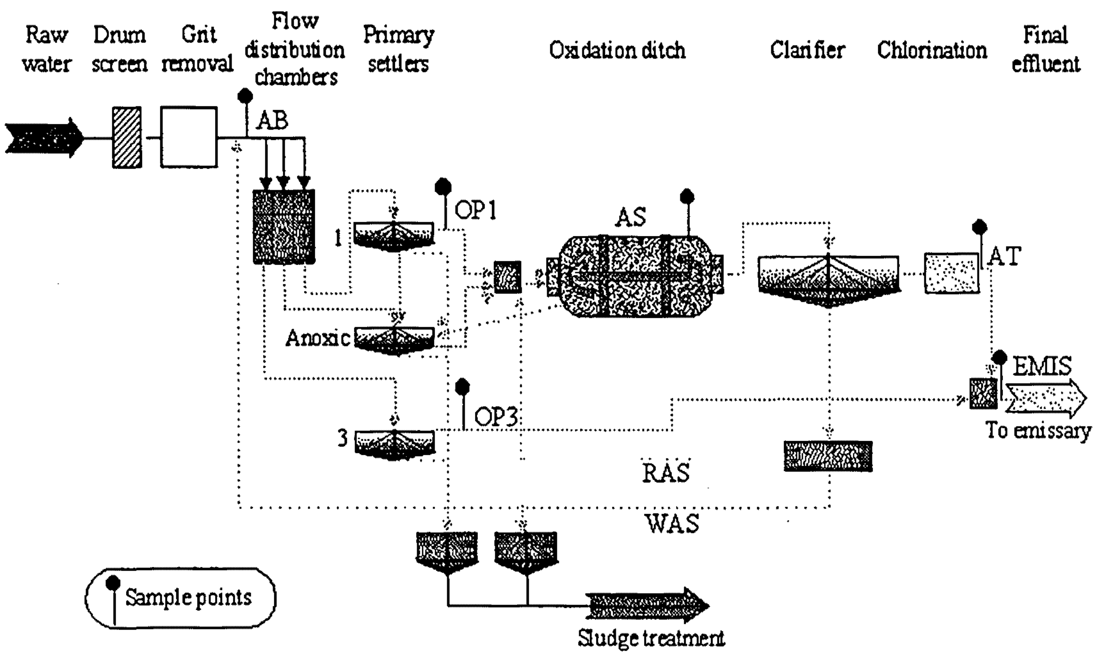


Figure 8.1: Schematic of a WWTP water flow-line providing primary and secondary treatment. The line consists of a pre-treatment (drum screen and grit removal), primary treatment (flow distribution chambers and three primary settlers), secondary treatment based on the activated sludge process (an oxidation ditch as bioreactor followed by a secondary settler), and final chlorination prior to discharge the treated water to an emissary.

8.3 A WWTP case study

The database utilized in the forthcoming experiments corresponds to a WWTP of a touristic resort situated in Costa Brava (Catalonia, Spain). This plant follows the schema of Fig. (8.1), providing primary and secondary treatment using the activated sludge process to remove organic load and suspension solids contained in the raw water of about 30,000 inhabitants-equivalents in winter and about 150,000 in summer.

The available historical data comprises a large amount of quantitative and qualitative information corresponding to an exhaustive characterization of the main points of the plant,

Sample Point	On-line Data (flow rates)	Analytical Data	Qualitative Data
AB (inflow)	Q-AB (inflow)	COD-AB, BOD-AB (organic matter) TSS-AB (suspended solids)	-
AS (bioreactor)	Q-R (biological recycle) Q-P (biological purge) Q-A (biological aeration)	-	Presence of foam Microfauna (*Aspidisca*, *Vorticella* ...) Filamentous bacteria (*Nocardia*, *Thiothrix* ...)
AT (outflow)	-	COD-AT, BOD-AT, TSS-AT	Look (appearance)

Table 8.1: Selected variables characterizing the behaviour of the studied WWTP.

such as the inflow, the bioreactor, and the outflow (indicated in Table (8.1) with suffixes -AB, -AS, and -AT, respectively). Quantitative information includes analytical results of water quality: organic matter –measured as chemical (COD) and biochemical (BOD) oxygen demand– and Total Suspended Solids (TSS), together with on-line signals coming from sensors: inflow, recycle, purge and aeration flow rates. Qualitative data include information about the presence of foam in the bioreactor ("Presence-foam"), the subjective appearance of outflow ("Look"), and daily microscopic examinations (basically, presence of microfauna –e.g. *Aspidisca*, *Vorticella*– and some filamentous organisms –e.g. *Nocardia*, *M. Parvicella*). This information is also being used in other approaches to improve WWTP operation [Comas *et al.*, 98].

The first work was focused on selecting an homogeneous amount of days, to cover a representative period of time. Then, it was necessary to select the most relevant variables of the process, corresponding to the analysis of water quality and flow-rates at different points of the plant. These variables are presented in Table (8.1), distinguishing between on-line and analytical values, and specifying the sample point (AB or influent, AS or bioreactor and AT or effluent). Global process variables are related to the three control actions that the plant manager can modify when removal efficiency decreases, in order to reconduct the process to normal performance: purge (Q-P), recycle (Q-R) and biological aeration (Q-A) flow rates. To simplify the description of the influent characteristics, the set of internal variables (Q-OP1, COD-OP1, BOD-OP1 and TSS-OP1) at the primary settlers has been excluded from the analysis.

The final data set covers an homogeneous representative period of 609 consecutive days, where each day is considered as a new sample. The second work has comprised a statistical analysis of the studied database variables. Basic statistical descriptors of the variables in the database are shown in Table (8.2) (for quantitative variables) and Table (8.3) (for qualitative ones). The relative abundance of qualitative variables is categorized in three different levels: *none*, *some* and *many*, with the exception of the outflow appearance (that is, "Look-AT"), categorized as *poor*, *fair* and *good*.

The most relevant feature of the database is the extremely high incidence of missing values (between 60% and 80%, approximately), basically due to different time measurements of the variables, and to the cost (in time and money) of performing some of the analytical tests. This is specially true in the case of the outflow variables COD-AT, BOD-AT and TSS-AT –more suitable as targets for developing prediction models– variables characterizing water quality at the inflow COD-AB, BOD-AB and TSS-AB, and qualitative variables characterizing the microorganisms. Clearly, this situation makes the search for models to characterize WWTP behaviour considerably hard. They must always be taken into account in evaluating the quality of the learned models.

Variable	Unit	Missing	Mean	StDev	Min	Max
Q-AB	m^3/d	18	10707	3634	0.0	23681
COD-AB	mg/l	380	795.8	198.0	150.0	1644
BOD-AB	mg/l	480	390.7	95.70	70.0	620
TSS-AB	mg/l	380	315.9	91.35	69.0	647
Q-R	m^3/d	1	5597.7	2287.1	0.0	12086
Q-P	Kg TSS/d	11	771.6	756.6	0.0	6523
Q-A	Kg O_2/d	61	4138.6	1878.4	0.0	8643
COD-AT	mg/l	380	55.8	18.52	20.0	134
BOD-AT	mg/l	480	8.96	4.876	2.3	32
TSS-AT	mg/l	376	9.56	5.750	2.0	42

Table 8.2: Basic statistical descriptors for selected WWTP variables (in 609 days).

Variable	No. of	Category		
(609 days)	Missing	*none*	*some*	*many*
Presence-foam	394	17	153	45
Zooglea	394	117	69	29
Nocardia	399	90	51	69
Thiothrix/021N	396	112	85	16
Type 0041	397	140	44	28
M. Parvicella	395	156	23	35
Aspidisca	503	8	82	16
Euplotes	438	154	16	1
Vorticella	501	4	89	15
Epistylis	501	9	81	18
Opercularia	450	126	27	6
Carniv. ciliates	394	160	48	7
Flagellates >20μm	394	184	23	8
Flagellates <20μm	394	176	24	15
Amœbae	394	173	38	4
Testate *amœbae*	394	206	8	1
Rotifer	394	117	97	1
		poor	*fair*	*good*
Look-AT	394	9	168	38

Table 8.3: Basic statistical descriptors for qualitative WWTP variables. The last three columns show the number of days for each variable and category.

The linear intercorrelation structure among variables is shown in Fig. (8.2) as a hierarchical clustering of the (absolute) correlation matrix of variables. With the exception of incoming water discharge (Q-P), the actuation (Q-), output (-AT) and input (-AB) variables are clustered into three -not too homogeneous- groups. The fact that the highest intercorrelations are observed in output variables (0.736-0.764) indicates that once a reasonable model is found for one of them, similar ones should be also found for the rest.

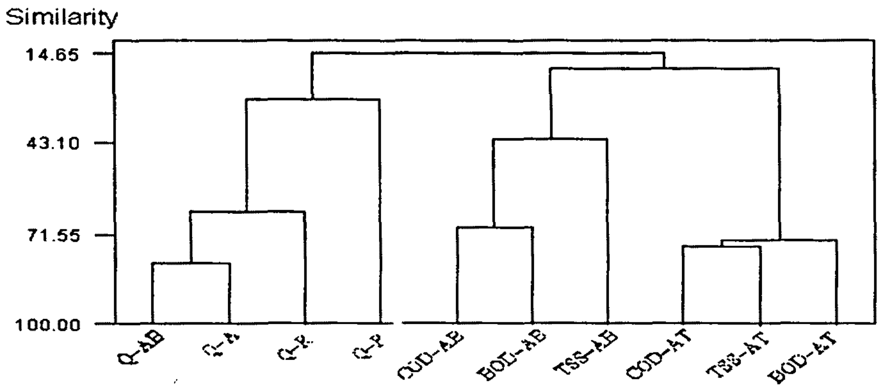


Figure 8.2: Hierarchical clustering of the absolute correlation for the studied WWTP variables.

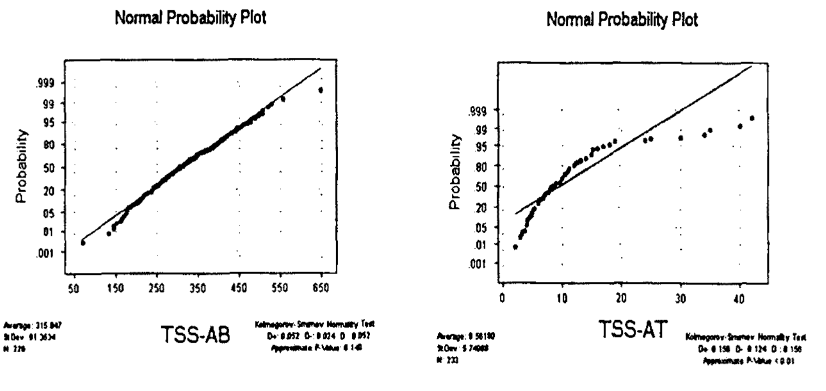


Figure 8.3: Normal probability plots of Total Suspended Solids for the Kolmogorov-Smirnov test, for incoming (TSS-AB) and outgoing (TSS-AT) total suspended solids.

The complexity of the WWTP behaviour problem is reflected in the frequency distribution of its variables. As an example, Kolmogorov-Smirnov tests applied to the incoming TSS-AB and outgoing TSS-AT variables confirm what direct inspection suggests: whilst the first variable is distributed normally, the second does not. Actually it has a right-skewed distribution, reflecting strong non-linear distortions introduced by the WWTP dynamics -see Fig. (8.3).

8.4 Experiments

In the first two experiments to be presented, the time behaviour of two outgoing variables (COD-AT and BOD-AT), expressing the quality of effluent water, is modelled as a function of influent characteristics and control actions. The obtained models take the imprecision inherent in the samples into account, and make use of all available information despite the significant presence of missing data.

The next natural step is to take into account *qualitative* variables -not considered in the previous studies- and to explore how it affects the formation of these predictive models. This qualitative information, although known to exert an influence in the process and conveying a great amount of information, is usually put aside because of its nature and the high levels of missing values that it brings along. These two features are a nuisance -if not a problem- for many neural learning algorithms and models, which have to accommodate qualitative and missing information in a deformative preprocessing.

Description of the methods

Four techniques are employed in this work in search of valid models of behaviour or to study the influence of specific variables: a heterogeneous neural network (trained with genetic algorithms), a classical neural network (the multi-layer perceptron, trained with simulated annealing plus the conjugate gradient), a probabilistic network (trained as a Bayes-Parzen classifier) and the k-nearest neighbours algorithm. Rough set theory is also used to perform a reduction of dimension. The three neural techniques are used as time-delay neural networks.

We begin by briefly outlining the methods employed in the experiments and not introduced in other sections of the Thesis; specifically, rough sets and probabilistic neural networks. The concept of a *time-delay* neural network (TDNN) was already presented in (§7.4.2).

Rough Sets.

An important issue in the analysis of dependencies among variables is the identification of information-preserving reduction of redundant variables. In particular, the task is to find a minimal subset of interacting variables having the same discriminatory power as the original ones, which would lead to the elimination of irrelevant or noisy variables, without the loss of essential information. Rough Sets [Pawlak, 91] exploit the idea of approximating a set by other sets. Given a finite set of objects U (the universe of discourse), a set $X \subseteq U$ and an equivalence relation R, two subsets can be associated, called the *lower* (R_L) and *upper* (R_U) approximation, respectively, as follows:

$$R_L = \{Y \in U/R \mid Y \subseteq X\}$$
$$R_U = \{Y \in U/R \mid Y \cap X \neq \emptyset\}$$

where U/R is the set of equivalence classes (a partition) induced by R. The lower approximation, also called the *positive region* $POS_R(X)$, is the set of elements which can be certainly classified as elements of X, whereas the upper approximation is the set of elements which can be possibly classified as elements of X. The *dependency coefficient* is defined as the ratio between positive region size and universe size. A set of variables P is independent w.r.t. the set of objects Q if for every proper subset R of P, $POS_P(Q) \neq POS_R(Q)$; otherwise P is said to be dependent w.r.t. Q. Moreover, the set of variables R is a minimal subset or *reduct* of P, if R is an independent subset of P w.r.t. Q, such that $POS_R(Q) = POS_P(Q)$. A variable $a \in P$ is superfluous if $POS_P(Q) = POS_{P\setminus\{a\}}(Q)$; otherwise a is said to be *indispensable* in P. The set of all indispensable relations is the *core*. An important property of the core is that it is equal to the intersection of all reducts.

Rules of the form <*condition*> $\Rightarrow$ <*decision*> can be generated by using the information contained in the reducts and the objects, concerning their condition and decision attributes. The *condition* part of the rule is a conjunction of attribute–value pairs. The *decision* part, in this study, is a single pair composed of the object's decision attribute. Three different strategies are used in the following experiments for rule generation from reducts, as follows:

Strategy 1 : for each object, this strategy finds a single relative optimal reduct (in the sense of its length), using heuristics for preserving the dependency coefficient. This strategy is usually the fastest;

Strategy 2 : for each object, the shortest relative reduct (in the explicit sense) is computed and used for constructing the rule;

Strategy 3 : this strategy operates in a class-wise manner by finding all shortest relative reducts whose rules cover some element of the corresponding class.

In all cases, repeated rules are not included. Criteria for matching objects with rules are based on a notion of distance, defined as the number of unmatched attributes taken from the set of predictor variables appearing in the rule. Missing attributes are considered in an optimistic sense, i.e., always matching. Two classification methods are used to test the performance of the generated rule sets:

Method 1 : Find the most frequent decision among rules with minimum distance from a given sample object.

Method 2 : Select first all the rules with minimum distance from a given sample object and then, for every selected rule, count the number of matched objects, choosing as decision the one corresponding to the rule with the highest such number.

Probabilistic neural networks.

This learning model [Specht, 90] is a reformulation of the Bayes-Parzen classifier –a classical pattern recognition technique [Duda and Hart, 73]– in the form of an artificial neural network. The fact that the Bayes classifier is optimal in the sense of the expected misclassification cost makes the use of this kind of network very attractive, specially for smooth classification problems, and in which all variables are relevant.

Besides the input layer, there is a so-called *pattern* layer with as many neurons as patterns are included in the training set. Next, a *summation* layer contains one neuron for each class, then leading to the output layer. Each pattern-layer neuron computes a distance measure between the input and the training sample associated with the neuron. The activation functions of these neurons are Parzen windows used to collectively approximate the probability density functions required by the classifier. The cornerstone of this method lies in its approximation of the multivariate population density function, estimated from the training set as the average of separate multivariate distributions, each centered in a sample from the training set. The main drawbacks are the curse of dimensionality, like all kernel-based methods, and the limited ability to ignore irrelevant variables, which may be a cause of poor generalization ability [Sarle, 99].

8.5 Experiment 1

The purpose of this first investigation is to assess whether partial models of the plant can effectively be found by neural techniques. The developed models characterize the effluent quality (measured as BOD-AT and COD-AT) as a function of the influent characteristics and control actions, by means of developing a model for each variable. The aim of this experiment has been to find, as a first step, models able to capture the time variation of basic outgoing WWTP variables.

Experiment setup

In this first experiment, two different TDNN approaches that differ in the training method have been tested: a hybrid procedure composed of repeated cycles of simulated annealing coupled with conjugate gradient algorithm (TDNN-AC) [Ackley, 87] and the HNN model presented, where a neuron model of the form (4.73) is used, with (4.74) as the activation function, as described in (§4.4.1). All the variables are taken to be fuzzy numbers with a 5% of imprecision, according to the estimated upper-bound of uncertainty. Their similarity is computed by using the measure in (4.60).

The TDNN-AC hidden layer uses the hyperbolic tangent as activation function. Both networks have an output layer composed by a linear neuron. It should be noted that the HNN model as used here (TDNN-HG) is viewed as a TDNN that incorporates heterogeneous neurons and is trained by means of genetic algorithms. The TDNN-HG and TDNN-AC architectures were thus fixed to include one output unit and 8 hidden units, corresponding to the model:

$$y(t+1) = F < x(t), x(t-1), x(t-2), y(t-1) >$$

where $x(t)$ denotes the current value of the input variable and $y(t)$ denotes the corresponding output. Selected inputs were Q-AB, Q-A, Q-P and Q-R, that is, the incoming flow rate and the three actuation variables and the output is considered in the 2-day delay. Hence, the model is composed of a total of 13 inputs. In the testing process, the normalized mean square error (in percentage), given by (7.2), between the predicted output value and the real output, is used to determine the quality of each of the inferred models.

For each studied output variable (BOD-AT and COD-AT), the TDNN-HG was trained using a standard genetic algorithm -enhanced to deal with missing values (§6.3.2)- with the following characteristics: binary-coded values, probability of crossover $P_{cross} = 0.6$, probability of mutation: $P_{mut} = 0.01$, population size: $\lambda = 150$, linear scaling with factor $c = 1.5$, selection mechanism: stochastic universal, replace procedure: worst individual. The algorithm stopped when no improvement was found for the last $1,000$ generations (typical values were about $7,000$). The TDNN-AC was trained in only one run and the process was stopped when a reasonable error was attained. In both cases, the training set chosen was the first half of available data (about 300 days).

Results of the experiment

The WWTP characterization produced via neural networks trained with the hybrid simulated annealing-conjugate gradient procedure was worse than the corresponding one obtained by using a fuzzy heterogeneous neural network model, as illustrated by normalized squared errors shown in Table (8.5) for BOD-AT and COD-AT output variables. In both cases the same neural architecture was used but the errors obtained are appreciably lower for the heterogeneous model w.r.t. the classical neural one, although it uses a very sophisticated training procedure.

	TDNN-AC	TDNN-HG
BOD-AT	45.55%	20.74%
COD-AT	30.76%	11.64%

Table 8.4: Normalized MSE errors of the two neural network models used for characterizing two of the outgoing variables.

The relation between the BOD-AT output variable, as estimated by the heterogeneous neural network, and the corresponding observed values is shown in Fig. (8.4, left). There is a significant linear correlation between both values, and model adequacy is revealed by the fact that almost all points are enclosed by the 95% confidence band.

The corresponding time behaviour is illustrated in Fig. (8.5), where the observed BOD-AT values are displayed together with the 95% confidence band given by the neural network model (upper and lower dashed curves). In spite of the fact that 78.7% of the data, corresponding to the 300 day period chosen for the characterization were missing, almost all observed values are within the confidence band with only very slight exceptions. A similar behaviour is exhibited by the COD-AT variable -Figs. (8.4, right) and (8.6).

8.6 Experiment 2

The previous experiment showed how models characterizing WWTP behaviour can be found. The next step to be taken is to develop a model able to predict future WWTP output in

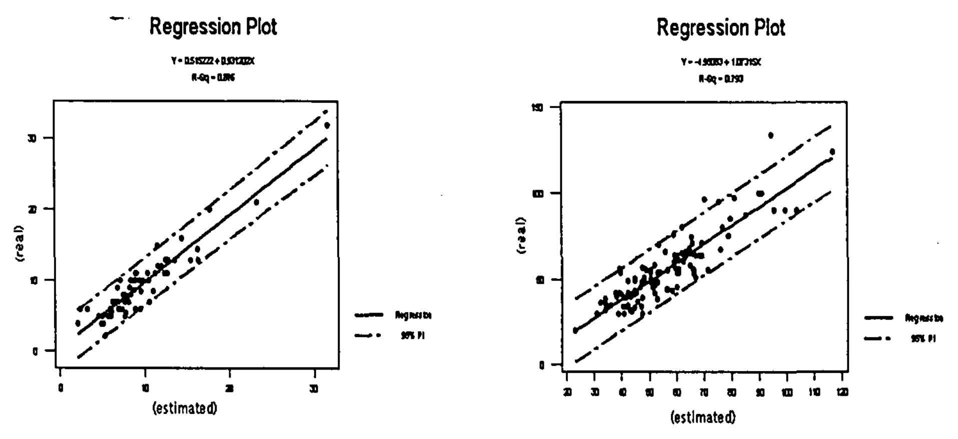


Figure 8.4: Relation between estimated vs. real BOD-AT (left) and estimated vs. real COD-AT (right).

situations never seen before (that is, not used in the formation of the model) again in light of available past values of its variables. This is a very difficult issue, again complicated by the presence of missing data in a set of characterizing variables that is already very heterogeneous in nature and plays strong non-linear interaction roles in the overall process. A further complication that arises is the different time scales of the variables. While some of them are available almost at will, others may take days (as, for example, the BOD-AT, that takes 5 days). For this reason, the COD-AT variable has been the one chosen as a prediction target. This variable is available in about 2 hours and, thus, previous values w.r.t. the present day are always known.

To achieve this, a still more careful selection and a clever treatment of the data to be used as training is a means to pave the way. Specifically, three treatments have been performed:

- First, the correlation structure reflected in Fig. (8.2) shows that the groups of variables (Q-AB, Q-A), (COD-AB, BOD-AB) and (COD-AT, TSS-AT, BOD-AT) are reasonably similar. This suggests the use of Q-AB, Q-R, Q-P, COD-AB and TSS-AB as input variables when considering the construction of prediction models. The choice of COD-AB is favored by the fact that it is a much simpler and faster analytical procedure than BOD-AB from the chemical point of view.
- Second, we observed that part of the errors of the models inferred in the previous experiments were due to the high peaks present in both studied variables (BOD-AT and COD-AT). For this reason, COD-AT was log_{10} transformed.
- Third, the delays used in such models were proposed intuitively, but without any regard to actual underlying significance. This is where rough set theory comes to play.

No doubt that one of the most important tasks when finding useful dependencies from the point of view of constructing prediction models is the discovery of those time delays in the

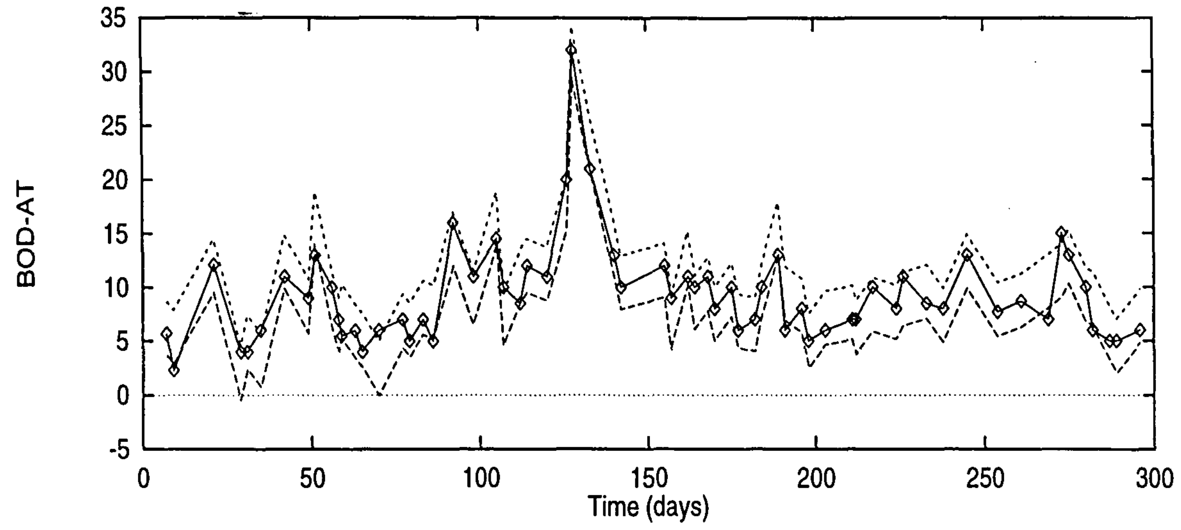


Figure 8.5: Time behaviour of BOD-AT during the first 300 days (solid line) with observed points. Upper and lower dashed lines indicate the 95% confidence estimation interval (according to the TDNN-HG model).

input variables, and in the predicted variable itself, carrying essential functional relationship. In the present study an experiment was made by forming a data matrix containing the information concerning the behaviour for each day of the last 10 days for variables Q-AB, Q-R, Q-P, COD-AB, TSS-AB and the target variable COD-AT itself. This makes a total of 60 new variables potentially related with the value of the COD-AT for each day, with a *dependency coefficient* found to be 0.9699 (a value of 1 means that the selected variables convey *all* the information present in the whole data available).

The continuous process represented by these data was transformed into a discrete one by analysing the empirical probability distribution of all variables involved and defining suitable categories introducing corresponding cut-point values. In particular, the following were set: Q-AB $[0,8500),[8500,13000),[16500,\infty)$, COD-AB $[0,650),[650,950),[950,\infty)$, TSS-AB $[0,250),[250,400),[400,\infty)$, Q-R $[0,5000),[5000,7000),[7000,\infty)$, Q-P $[0,1000),[1000,\infty)$ and log_{10}(COD-AT) $[0,1.65),[1.65,1.85),[1.85,\infty)$.

The core and reducts were computed for the discrete process obtained via categorization of the original data and it was found that, from the original 60 potential predictor variables, only 13 were really indispensable, whereas adding another 7 makes them an optimal reduct. That is to say, optimal from the point of view of relative size of the positive region defined by these 20 variables, and w.r.t. the positive region defined by the whole set.

The core itself was composed of the following variables: Q-AB (delay 1), TSS-AB (delay 7), Q-R (delays 1, 2, 3, 4, 5, 7, 9, 10) and log_{10} (COD-AT) (delays 5, 7). The optimal reduct is completed by variables Q-AB (delays 5, 10), COD-AB (delays 2, 4), Q-P (delay 5) and log_{10}(COD-AT) (delays 2, 3). It is interesting to observe that almost all information coming from the recirculation flow was considered essential (a variable controlled by the WWTP

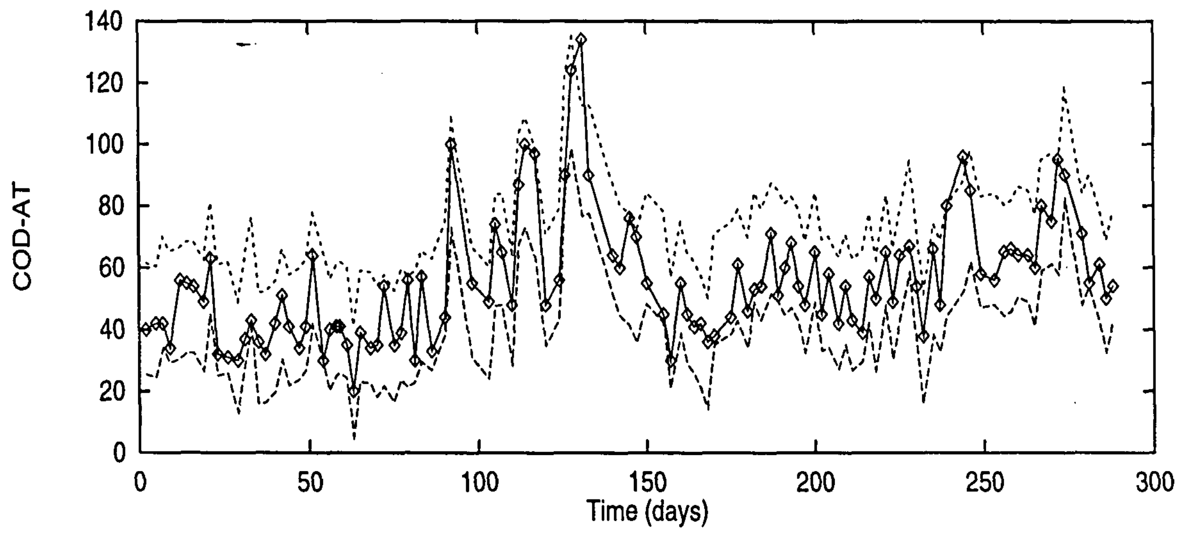


Figure 8.6: Time behaviour of COD-AT during the first 300 days (solid line) with observed points. Upper and lower dashed lines indicate the 95% confidence estimation interval (according to the TDNN-HG model).

human operator). This delay information given by the optimal reduct was used to set up a prediction model based on the HNN as in Experiment 1, this time with log_{10}(COD-AT) with delay 0 as target.

A very simple HNN architecture consisting of just 2 hidden neurons was utilized (20 inputs, 1 output), with the same training set used for the previous experiments (50% of the total available). The last 25% (56 days) as data to be predicted. Also, very prudent GA settings were used (26 individuals, only 500 generations) to avoid excessive data overfitting.

The behaviour of the predicted COD-AT values w.r.t. the real observed ones is shown in Fig. (8.7). In spite of the fact that the fit is not as accurate as before, the relation between the two is highly significant, both from the *linear correlation* coefficient and the *linear regression* points of view, as tested with the corresponding *t-test* for the correlation coefficient and the *F-test* for the analysis of variance (for 95% confidence in both cases). Actual numbers for the *t-test* are: $R = 0.504$ with 54 degrees of freedom ($t = 4.288$). The result for the *F-test* is 18.39 for one degree of freedom in the numerator and 54 in the denominator -Fig. (8.8). All this shows that the model, although far from perfect, does capture prediction information and is able to prognose outputs within a 95% confidence band. This is particularly important in view of the WWTP complexity and the vast quantity of missing information (78.7% of the data in the period chosen for the characterization were missing).

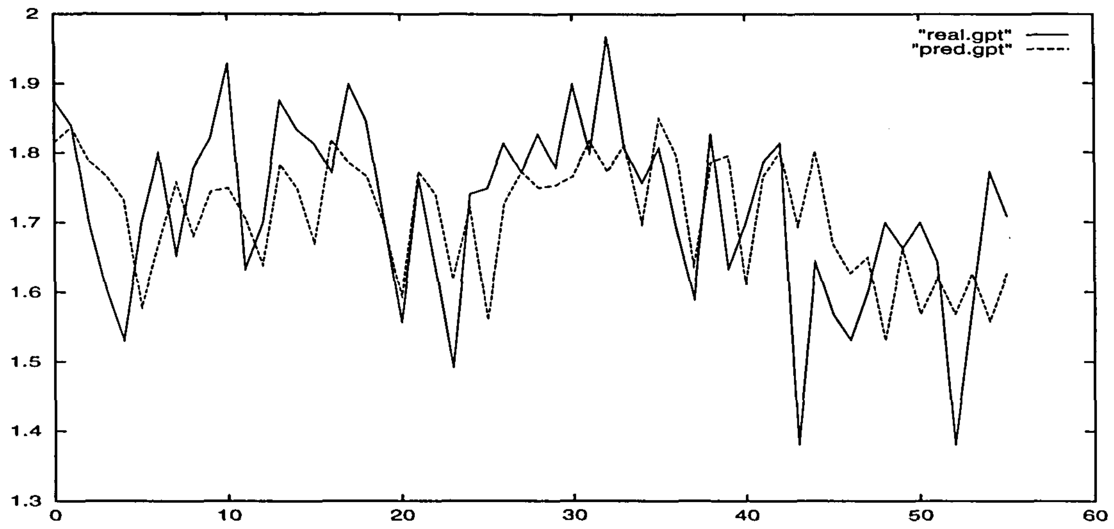


Figure 8.7: Actual time behaviour of COD-AT during the last 56 days (solid line) with observed points against prediction according to the TDNN-HG model. The normalized MSE for prediction is 60.0%.

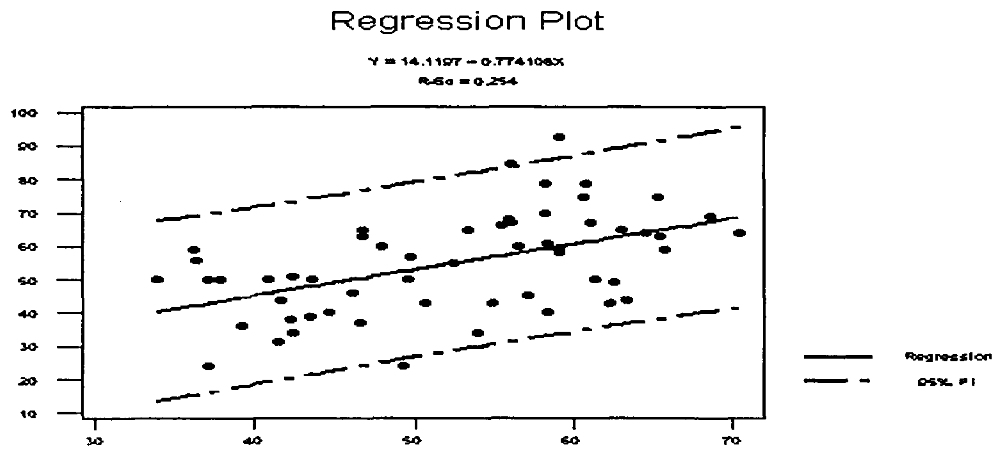


Figure 8.8: Relation between predicted vs. real COD-AT (solid line). Upper and lower dashed lines indicate the 95% confidence interval (according to the TDNN-HG model).

8.7 Experiment 3

8.7.1 Preliminaries

The purpose of this third investigation is to present several experiments performed using *qualitative* information, either *per se* or together with quantitative information, such as influent characteristics and control actions. It is known that qualitative features —including microscopic examinations of microfauna and bacteria, and some subjective information— are useful indicators of overall process performance, and strongly influence the activated sludge process. Specifically, the influence on effluent TSS levels is studied, as an indication of plant performance and fulfillment of regulations. There is also a second need to handle uncertain or imprecise information, a characteristic present in all kinds of variables, specially in all numeric measurements coming from on-line analyzers, but also in analytical determinations and qualitative observations.

The results show that qualitative information exerts a considerable influence on plant output, although quite variable, since high degrees of information redundancy are discovered. Comparable predictive capabilities are obtained when working with a much reduced set of variables, which coincide with those highly rated by WWTP experts. Also, a common upper-bound in classification accuracy is discovered, in light of the coherent results yielded by methods that are very different in nature. In addition, despite the high levels of missing information, very reasonable prediction models are found.

8.7.2 The bulking phenomenon

As explained in (§8.2), in an activated sludge process, the wastewater, which contains organic matter, suspended solids and nutrients, goes into an aerated tank where it is mixed with biological floc particles. After a sufficient contact time, this mixture is discharged into a settler that separates the suspended biomass from the treated water. Most of the biomass is recirculated to the aeration tank, while a little amount is purged daily [WEF, 96] –see Fig. (8.9).

Activated sludge is a clear example of an environmental process that is really difficult to understand, and thus difficult to be correctly operated and controlled. The inflow is variable (both in quantity and in quality); not only there is a living catalyst (the microorganisms) but also a population that varies over time, both in quantity and in the relative number of species; the knowledge of the process is scarce; there are few and unreliable on-line analyzers; and most of the data related to the process is subjective and cannot be numerically quantified.

Most of the problems of poor activated sludge effluent quality result from the inability of the secondary settler to efficiently remove the suspended biomass from the treated water. When the biomass is strongly colonised by long filamentous bacteria, holding the flocs apart and hindering sludge settlement, the amount of Total Suspended Solids (TSS) at the outflow of the plant increases seriously. Although this phenomenon, called *bulking*, has been extensively studied, the interrelations and diversity of the many bacterial species involved, and the uncertainty about the factors triggering their growth constitute obstacles to a thorough and clearcut understanding of the problem.

Previous works have applied stochastic models and neural networks to accurately predict the occurrence of future bulking episodes [Capodaglio *et al.*, 91]. This study uses 14 months of *complete* daily measurements of quantitative data only, from the Jones Island WWTP in Milwaukee (Wisconsin, USA). Although the study is based on real data, it is not common (at least in Europe) to make daily analytical measurements of all process variables (*e.g.*, organic matter and TSS are typically measured two or three times a week). As a result, the databases

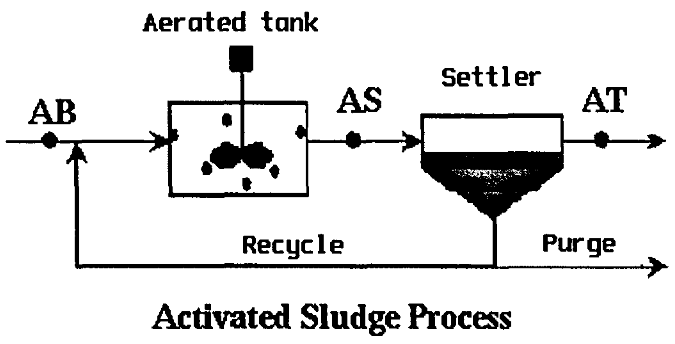


Figure 8.9: Flow diagram of the **Activated Sludge Process**. The influent stream (sample point **AB**) is combined with the sludge return stream **Recycle** and sent to the aerated tank (sample point **AS**) for biological oxidation of the organic matter. A settler is then used to remove treated water (sample point **AT**) from biomass and to thicken it. The withdrawn sludge **Purge** is concentrated to a higher solids content in the sludge line of process.

are full of missing values, evenly distributed all over the time. Incidental equipment failures also bring along compact chunks of missing data. This high incidence of missing information is the main reason why most of other studies are based on simulated data.

There is then a clear interest in a model of the process. This model should allow to obtain an accurate estimation of TSS ranges at the outflow of the plant, based on the relationship among the most relevant variables of the process, both quantitative (e.g. flow rates and analytical results) and qualitative (biomass microscopic examinations and process observations), in order to know whether the plant is meeting the discharge permit requirements.

In this study, the final database processed includes only those days with a recorded value in the target variable TSS-AT, causing the initial data matrix to shrink from 609 to only 233 days –Table (8.2), last row. Nevertheless, the rate of missing values is still extremely high among potential predictor variables.

8.7.3 Setup and specification of the methods

Three different TDNN approaches that differ in the neuron model and training method have been tested: a multi-layer perceptron trained by means of the hybrid procedure composed of repeated cycles of simulated annealing coupled with the conjugate gradient algorithm (which we will call TD_{MLP}), our HNN model (*id.* TD_{HNN}), incorporating heterogeneous neurons and trained by means of genetic algorithms, and the probabilistic neural network (TD_{PNN}). Four architectures formed by a hidden layer of 2, 4, 6 and 8 neurons and an output layer of a linear neuron were studied. The TD_{HNN} was again trained using a standard genetic algorithm with

$P_{cross} = 0.6$, $P_{mut} = 0.01$, two explored population sizes of $\lambda \in \{26, 52\}$ individuals, a linear scaling with factor $c = 1.5$, stochastic universal selection and a replacement procedure given by the worst individuals. The algorithm was allowed 5 runs for each population size and stopped after 1,000 generations unconditionally. The TD$_{\text{MLP}}$ uses the hyperbolic tangent instead of the logistic, and is trained in one long run for every architecture, in which the number of annealing restarts was fixed to 50. In all cases, average and best results found across the architectures are shown.

The TD$_{\text{PNN}}$ was used here with a Gaussian kernel. During training, each variable and class units were allowed to have their own variance, with values optimized during the process (possible values ranged from 0.001 to 10). Also, the k−nearest neighbours (KNN) algorithm (with $k = 3$) was tested against the data as a further reference (recall that this algorithm has no training phase). The TD$_{\text{HNN}}$ treats qualitative and missing information directly, and original real values as triangular fuzzy numbers, by considering a $\pm5\%$ of imprecision w.r.t. the reported value. The other two neural approaches codify all information as real-valued and a missing input as zero (no input).

8.7.4 Description of the experiments

The effluent quality of the WWTP process given by the TSS-AT was discretized by categorizing the original continuous values into three classes $\{[0, 5), [5, 13.5], (13.5, \infty)\}$, expressing *low*, *normal* and *high* values. Four main sets of experiments were performed, all in accordance with the general model:

$$y(t) = F\{x_1(t-2), x_1(t-1), \cdots, x_m(t-2), x_m(t-1), y(t-2), y(t-1)\} \ \forall t \geq 3 \qquad (8.1)$$

where m is the number of input variables, for a total of $\hat{m} = 2m+2$ *model* input variables. Each $x_i(t)$ denotes the value of the i-th input variable and $y(t)$ the value of the target TSS-AT output variable, at time t. The number m varies and will be specified accordingly.

For each experiment, a preliminary study of the training data matrices via rough set analysis is first presented, with the aim of evaluating the actual predictive capacity of the considered model and thus what can be expected on its influence in the output. Next, the matrices are processed by using the three different strategies for rule generation, and the generated rules, using the two classification methods, are applied to the test matrix, yielding corresponding percentages of correct classification. For the training set, the number of generated rules in each case is shown too. In addition, the results obtained by training and testing the three neural methods (classical, heterogeneous and probabilistic) and the k-nearest neighbours (KNN) algorithm are collectively shown and discussed. The advantage of this fanning out of methods is that, being so different in nature, are able to analyze the data from very different perspectives, allowing to draw more general conclusions. It has to be noted that, throughout all the experiments, all the methods are applied to the data in exactly the same experimental conditions. A description of the four groups of experiments follows.

Experiment 1: Qualitative.

Oriented to reveal the influence of qualitative variables when studied *per se*; in particular, to reveal their predictive ability on the TSS classes, taking as inputs x_i the qualitative variables of Table (8.3) (thus $m = 18, \hat{m} = 38$). This leads to a matrix of qualitative information 145 days long, split into a balanced (in the sense of class frequencies) training part (the first 115, 79.3%) and test part (the subsequent 30 consecutive days, 20.7%) to be forecast. It should be noted that the initially formed matrix (232 days long) had a portion of missing information so severe that entire rows had to be removed because *all* information was missing. After that, figures for missing information still are 57.8% in training and 56.9% in test. As a further reference, the percentage of *normal* days (the majority class) in the test matrix is 73.3%.

Experiment 2: Reduced-Qualitative.

The previous results via rough set analysis are used in this second experiment as an attempt to reduce the number of model input variables. This, besides being beneficial for the majority of learning methods, will shed some light on the relevance of variables in relation to the TSS-AT. The new matrices consist of the same days as in Experiment 1, though only 12 of the original 38 model variables are to be used.

Experiment 3: Combined.

Aims at discovering how the variables in Table (8.3) behave when joined to five selected quantitative variables: those corresponding to inflow characteristics (Q-AB, COD-AB, TSS-AB) and control actions (Q-P and Q-R). These last variables are counted among the most relevant of the overall process, according to their linear intercorrelation structure –see (§8.3). Model parameters are thus ($m = 23, \hat{m} = 48$). The heterogeneous data matrix generated covers the whole period of days since this time none had to be removed from the matrix, although figures for missing information were 64.2% in training and 63.4% in test. It was split into a training part (the first 191, 82.3%) and a test part (the subsequent 41 days, 17.7%) to be forecast. The percentage of *normal* days in the test matrix is 70.7%.

Experiment 4: Reduced-Combined.

The model of Experiment 3 is reduced, again via rough set analysis, leading to a model with less variables and to much lesser missing information percentages, of 31.6% in training and 29.8% in test.

8.7.5 Experimental results

The displayed information includes average and best *predictive* accuracies obtained with each method. Training information is also shown. For the rough set approach, this information is given for every strategy and method, along with the number of rules generated.

	Str. 1 (70 rules)		**Str. 2** (72 rules)		**Str. 3** (18 rules)		TD$_{HNN}$		TD$_{MLP}$		TD$_{PNN}$
	Met. 1	Met. 2	Met. 1	Met. 2	Met. 1	Met. 2	Best	Avg.	Best	Avg.	
Train	75%	74%	79%	74%	69%	74%	87.0%	82.2%	86.9%	82.2%	76.5%
Test	73.3%	73.3%	73.3%	73.3%	73.3%	73.3%	80.0%	76.3%	73.3%	47.5%	73.3%
Train	78%	74%	79%	74%	67%	74%	85.2%	81.5%	82.6%	81.3%	83.5%
Test	73.3%	73.3%	73.3%	73.3%	73.3%	73.3%	76.7%	75.4%	76.7%	70.2%	16.7%

Table 8.5: Rough set approach and Neural approaches: correct classification percentages for **Experiment 1** (top two rows) and **Experiment 2** (bottom two rows), along with the number of rules needed.

Experiment 1: Qualitative

Beginning with the preliminary analysis, under the rough set approach, the relative reducts and the core were computed. The *dependency coefficient* between the 38 model variables and the predicted TSS-AT in the training set was found to be zero, indicating that no element can be classified with absolute security and, therefore, that the set of variables is rather incomplete. A total of 68 relative reducts were found, with a core composed of 11 variables. The frequency distribution of variables in the reducts reveal that 12 do appear in 75% or more of all the reducts; specifically, the 11 of the core plus an extra variable. On the other hand, another 14 variables from the original set of 38 are superfluous (they occur in no reduct). All this means that information dependency is unevenly distributed in the set of variables, as 32% of them is conveying the major part, while another 37% is carrying no information.

The results of the rule generation process and the three neural approaches are given in Table (8.5, top two rows) as percentages of correct classification. All the methods and strategies are signaling the same prediction ability, 73.3%, which coincides with the majority class. This poor performance is nonetheless reflecting the complexity of the data set, with a high rate of missing values affecting all variables, and classes showing severe overlappings, revealed by the null dependency coefficient. It is interesting to observe that Strategy 3 for rule generation needed only 23% of the rules required by the other two while keeping the same effectiveness. The result achieved by KNN is 76.7%.

For the neural methods, several aspects are noteworthy. First, the results are quite similar and consistent both for training and test sets. In other words, no method clearly outperforms the rest. Second, there seems to be a limit in training set accuracy around 87.0% and at 80.0% in test, which is not a bad result for such messy data. Also interesting to note are the solid results achieved by the TD$_{HNN}$, the poor average achieved by the TD$_{MLP}$ and the comparatively good KNN performance.

Experiment 2: Reduced-Qualitative

In order to assess the viability of smaller models, a new data matrix was constructed as in Experiment 1, using only those model variables –twelve, see Table (8.6)– occurring most frequently (in 75% or more) in the collection of reducts. Note in the table that selected variables

Variable	Delay
Q-AB	$t-2$
Q-AB	$t-1$
COD-AB	$t-2$
TSS-AB	$t-2$
Q-R	$t-2$
Q-R	$t-1$
Q-P	$t-2$
Q-P	$t-1$
Nocardia	$t-2$
Thiothrix/21N	$t-2$
Aspidisca	$t-1$
TSS-AT	$t-2$
TSS-AT	$t-1$

Table 8.8: Reduced set of *combined* variables.

in this plant (*Nocardia* and *Thiothrix* or type 021N) causing bulking sludge, and a protozoa (*Aspidisca*), the absence of which may indicate a decrease in plant performance and poor settling characteristics. It is also remarkable the fact that these three variables also appeared in the previous reduced set of qualitative information, and are the sole survivors when mixed with the numerical information.

And fourth, again, the predicted variable itself (TSS-AT) (at both delays) is considered amongst the most informative. The behaviour of this model Table (8.7, bottom two rows) is similar to that of the previous, in the sense that classification performances for training and test sets are slightly less, showing that the effect of the 35 discarded variables was in fact small. The result achieved by KNN is 63.4%.

Turning the attention to the neural models –Table (8.7, bottom two rows)– it is interesting to observe that the overall results are consistent with those obtained in the different experiments, specially in what concerns the test set. Moreover, since the TD_{PNN} is asymptotically optimal in the sense of the Bayes classifier, this might indicate a limit in what is achievable with the available information. Also, the fact that the TD_{HNN} model gives slightly but consistently higher results and a more balanced training/test ratio than all of the other methods has been observed in other application contexts and can be attributed to its better treatment of missing values and qualitative information.

8.8 Conclusions

For the WWTP under study, three main aspects have been found that deeply characterize the processes that are taking place. First, with the exception of the water discharge flow (Q-P), actuation, outgoing and incoming variables are clearly distinguished from one another, reflecting an internal structure that must be taken into account during the search for accurate models of the process. Second, the process dynamics introduce strong non-linear distortions between incoming and outgoing variables. Third, these outgoing variables are significantly related and, therefore, could be described by similar models. The used techniques have shown capable to describe the behaviour of some of these processes in a statistically significant sense, despite the imprecision associated to raw real-world information and the high degree of incompleteness and fragmentation, due to the number of missing values and their time distribution in many small chunks. The fact that the TD_{HNN} model outperformed the classical TD_{MLP}, suggests that it fits better the especial requirements posed by the WWTP problematic. In all, acceptable prediction models are found that show the interplay between variables and give insight to the dynamics of the process.

The influence of qualitative information on WasteWater Treatment Plants (WWTP) has also been studied regarding the quality of effluent suspended solids, one of the measures of plant performance. We found that qualitative information exerts a considerable influence on the output, although very unevenly. A high degree of information redundancy was discovered, since comparable predictive capabilities are obtained when working with much reduced subsets of variables, obtained by rough set analysis. However, it should be noted that this redundancy refers only to the prediction of *bulking* episodes in the process, and the use of these variables is necessary to guarantee the performance of the entire activated sludge process.

The analysis produces homogeneous groups of variables; for qualitative variables only, it signals the greater importance of 2-day delayed data in the process dynamics, instead of 1-day data. When qualitative and numerical information are collectively considered, the latter are found to be amongst the more informative, always in both delays. None the less, there are certain qualitative variables (the intersection of Tables 6 and 9) playing a significant role in the process. In both cases, these selected variables are highly rated by WWTP experts. They also tend to be the ones that show the lower amount of missing values, thus reducing the relative overall amount.

In addition, a common upper-bound in predictive classification accuracy has been discovered, located around an 80%, which is a very nice result for such messy data. In this respect, our conclusion is that the generalized and (relatively) poor performance can be attributed almost entirely to the data -besides to the problem complexity- in light of the consistent results yielded by methods that are so different in nature; the fact that they are based on very different principles allows to derive broader conclusions from the available data. The possibilities of some of these methods (especially the TD_{HNN}) are also noteworthy, provided they can handle heterogeneity, imprecision and missing values, aspects that characterize the data in a real WWTP process.

In conclusion, the observed patterns of behaviour are very coherent. The next step should be oriented towards adding information in the form of better delays (*e.g.* the weekly effect) and a more accurate selection of variables, guided by the findings reported herein. Ulterior studies with data coming from other plants are needed to determine whether these patterns are specific or represent a more general property of WWTPs. A further goal in the future is the development of a predictive model for control variables (Q-P and Q-R). These models will supply the plant manager with a useful tool to improve plant control and operation.

Chapter 9

Experimental Results on Benchmarking Problems

Theory is something that is good, but a good experiment remains forever.

Peter L. Kapitsa

9.1 Introduction

In this Chapter we proceed to carry out an extensive experimental evaluation of the approach on several well-known benchmarking problems. These problems are, for the most part, real problems which have been used through the literature of pattern recognition in such a way that they have become fairly standard. They have been included as a subject of study to complement the results on real-world problems presented in Chapters (7) and (8). Moreover, the fact that they are benchmarks permits a comparative estimate to related results found in the literature of neural networks.

A preliminary set of experiments involving part of the studied data sets has been reported in [Belanche, 00c]. In the present experiments, this work is extended by using more data sets, a selection of different architectures, ten partitions of the data and a more comprehensive analysis of the results.

The main source for information about the problems (the input/output examples along with a description of their features) has been the well-documented Proben neural repository [Prechelt, 94], which in turn is based on the large UCI repository of machine learning databases [Murphy and Aha, 91]. Since these archives contain mostly classification tasks, two additional regression problems are used.

The chosen problems have been selected as representatives because of their variety, which shows in three aspects:

1. The diversity in the underlying kind of problem;

2. The richness in data heterogeneity and amount of missing information;

3. The varying degrees of domain knowledge.

In addition, the selected classification problems are among the hardest in the mentioned repositories, in the sense that reported results consistently indicate a poor generalization performance (about or superior to 20% of errors). Some of them have been found to be very resistant to be learnt completely (that is, attaining a 0% of error in the training set).

Concerning the artificial problems, while it is generally true that the obtained results cannot be extrapolated to real problems, their interest relies in that most of their features can be precisely controlled (as the degree of non-linearity, number of irrelevant variables or amount and type of noise). Hence, their inclusion is interesting because it permits evaluation of the models over a wider variety of tasks and virtually eliminates the contingency of biasing the experiments towards given classes of problems.

9.2 Problem description

9.2.1 General description

The problems in these archives were originally meant for general machine learning approaches; in fact, most of them cannot be readily used by traditional neural systems because of the presence of non-continuous or missing information. In all, they are representative of the kinds of variables typically found in real problems, while displaying different degrees of missing information (from 0% to 26%).

The following data sets are studied: the well-known *Pima Diabetes, Horse Colic, Credit Card, Heart Disease* and *Solar Flares* taken from the Proben repository, *Sinus-Cosinus* from [Bersini and Bontempi, 97] and *SISO-Bench* from [Su and Sheen, 92], for a total of seven learning tasks[1]. These last two problems have continuous variables only, *Solar Flares* has not any, and the other four display a good mixture of variables, and varying percentages of missing information. Their main characteristics are displayed in Table 9.1.

There is also a documentation for all the problems –except for *Credit Card*– in what regards the meaning of variables; this allows for a finer assessment of the more appropriate treatment. There is hence an explicit *transfer of knowledge* from the domain knowledge to the heterogeneous neuron model.

9.2.2 Detailed description

Pima Diabetes

The Pima Indians database contains relevant information to diagnose diabetes on a number of female Indians. It consists of personal data and the results of medical examination. The

[1]The last two have been given a name here, for convenience.

Name	Type	#P	Def.	Missing	Miss. #P	In→Out	Data
Pima Diabetes	C	768	65.1%	10.6%	48.8%	$8 \rightarrow 2$	6R, 0N, 2I
Credit Card	C	690	55.5%	0.65%	5.4%	$15 \rightarrow 2$	6R, 9N, 0I
Horse Colic	C	364	61.5%	26.1%	98.1%	$20 \rightarrow 3$	5R, 5N,10I
Heart Disease	C	920	55.3%	16.2%	67.5%	$13 \rightarrow 2$	3R, 6N, 4I
Solar Flares	R	1066	-	0.0%	0.0%	$9 \rightarrow 3$	0R, 5N, 4I
Sinus-Cosinus	R	400	-	0.0%	0.0%	$2 \rightarrow 1$	2R, 0N, 0I
SISO-Bench	R	500	-	0.0%	0.0%	$2 \rightarrow 1$	2R, 0N, 0I

C classification R regression R real N nominal I ordinal

Table 9.1: Some basic characteristics of the data sets. #P: number of cases, Def.: default accuracy. Missing: total percentage of missing values. Miss. #P: percentage of patterns with at least one missing value. In→Out: number of problem inputs and outputs. The last column shows original data heterogeneity.

task is to decide whether an individual is diabetes positive or not. Of the 768 records, 500 (65.1%) correspond to the negative case.

This data set is very interesting in the sense that a moderate number of values are zero. They correspond to variables for which such value is physically impossible (e.g. diastolic blood pressure or body mass). These values are most probably originally missing and consequently treated as such. Of the eight variables, six are originally continuous and two are ordinal variables (1. number of times pregnant and 8. age in years).

Particular comments: Two of the continuous variables (the less precise) are converted into fuzzy numbers with a low fuzziness, estimated at a 0.5%, reflecting the uncertainty derived from imprecise measurements. The ordinal variables have crisp boundaries and are thus kept as such.

A description follows:

1. Number of times pregnant [*ordinal*]
2. Plasma glucose concentration [*continuous*]
3. Diastolic blood pressure (mm Hg) [*continuous*]
4. Triceps skin fold thickness (mm) [*continuous*]
5. 2-Hour serum insulin (mu U/ml) [*continuous*]
6. Body mass index (weight in kg/height in m^2) [*continuous* → *fuzzy number*]
7. Diabetes pedigree function [*continuous* → *fuzzy number*]
8. Age (years) [*ordinal*]

Total: 4 continuous variables, 2 fuzzy numbers (0.5% fuzziness) and 2 ordinal variables. Percentage of missing information: 10.6%.

Horse Colic

The Horse Colic database contains information to predict the fate of a horse that has a colic. It consists of data coming from a veterinary examination. The task is to decide whether an specimen will survive, will die or will be euthanized. Class distribution among the 364 records is 224 (61.5%), 88 (24.2%) and 52 (14.3%) for the three outcomes, respectively.

This data set shows a good variety of variable heterogeneity and a considerable amount of missing information (one of every four values is absent). Of the twenty variables, five are originally continuous, five are nominal and ten are ordinal variables.

Particular comments: Variables number 3 and 4 are ordinal because it makes much more sense than consider them as continuous. This is confirmed by the fact that their measurements are always natural numbers in the data set. Their crisp nature makes them ordinal variables. However, there are a number of variables that, besides being clearly endowed with an underlying ordering relation, also display a source of vagueness –coming from their subjective character– that has to be considered. This is the case of variables number 5, 6, 10, 11, 12 and 15. These are treated as linguistic variables by respecting the number and order of the initially crisp linguistic terms. In absence of precise information, the cut points are set at the 0.5 level, as is usually done. An interesting case is variable number 13; it is treated as ordinal –and not as linguistic– because the inter-value boundaries are crisp. A similar argument can be said about variable number 8. In this case, though, there are only two possible values and thus the order information cannot be used; it is then declared as nominal. Variable number 7 could also be ordinal but we are not aware of any order.

The continuous variables are converted into fuzzy numbers, in the same way than was done for the Pima Indians data set, with a low fuzziness of a 0.5%. This means that, for example, in variable number 2 (rectal temperature in degrees Celsius) an original measurement of 40^o degrees has a reasonable uncertainty of $\pm 0.2^o$ degrees. Finally, the first variable is taken as *linguistic* (and not *nominal*) because the inter-value boundary between the two possible values (*young* or *adult*) is vague.

A description follows:

1. Age (young, adult) [*ordinal* → *linguistic*]
2. Rectal temperature (degrees Celsius) [*continuous* → *fuzzy number*]
3. Pulse (beats per minute) [*ordinal*]
4. Respiratory rate (times per minute) [*ordinal*]
5. (Subjective) temperature of extremities (cold, cool, normal, warm) [*ordinal* → *linguistic*]
6. (Subjective) peripheral pulse (absent, reduced, normal, increased) [*ordinal* → *linguistic*]

7. Mucous membranes color (normal pink, bright pink, pale pink, pale cyan, bright red, dark cyan) [*nominal*]
8. Capillary refill time judgement (less than 3 secs., 3 or more secs.) [*nominal*]
9. (Subjective) pain estimation[2] (no pain/alert, depressed, intermittent mild pain, intermittent severe pain, continuous severe pain) [*nominal*]
10. Peristalsis: gut activity (absent, hypomotile, normal, hypermotile) [*ordinal* $\rightarrow$ *linguistic*]
11. Abdominal distension (none, slight, moderate, severe) [*ordinal* $\rightarrow$ *linguistic*]
12. Nasogastric tube gas emission (none, slight, significant) [*ordinal* $\rightarrow$ *linguistic*]
13. Nasogastric reflux (none, less than 1 liter, more than 1 liter) [*ordinal*]
14. Nasogastric reflux PH [*continuous* $\rightarrow$ *fuzzy number*]
15. Rectal examination of feces (absent, decreased, normal, increased) [*ordinal* $\rightarrow$ *linguistic*]
16. Abdomen (normal, firm feces in the large intestine, distended small intestine, distended large intestine, other) [*nominal*]
17. Packed cell volume [*continuous* $\rightarrow$ *fuzzy number*]
18. Total protein (grs./dl) [*continuous* $\rightarrow$ *fuzzy number*]
19. Abdominocentesis appearance (clear, cloudy, serosanguinous) [*nominal*]
20. Abdominocentesis total protein (grs./dl) [*continuous* $\rightarrow$ *fuzzy number*]

Total: 5 fuzzy numbers (0.5% fuzziness), 3 ordinal variables, 7 linguistic variables and 5 nominal variables. Percentage of missing information: 26.1%.

Credit Card

The Credit Card database contains relevant information to predict the approval or rejection of a credit card to a customer. It consists of personal data and economic conditions of bank customers. The task is to decide whether an individual will be granted the credit card or not. Of the 690 records, 307 (44.5%) are positive and the remaining 383 (55.5%) negative.

This data set is also very interesting because of the great heterogeneity in the data. Of the fifteen variables, six are originally continuous and nine are nominal, and there are nominal variables with small number of possibilities (just two) and with a large number (up to fourteen). There is also a tiny amount of missing information.

Particular comments: For this dataset, the meaning of the individual attributes has been kept confidential, so that there is no knowledge about existing orderings on discrete variables or about the exact nature of numerical information. For this reason, all discrete variables are

[2]Documentation explicitly says not to treat this feature as ordered.

taken as nominal. A preliminary experiment was carried out in the original data, by treating numerical variables as crisp or fuzzy numbers. A moderate amount of fuzziness (0.5%) was used, in order to better assess its influence. The results showed a similar generalization performance, with the fuzzy model showing a much better approximation ability, signaling a possible source of uncertainty in the continuous variables. The decision is taken therefore in favour of this last option.

Total: 6 fuzzy numbers (0.5% fuzziness) and 9 nominal variables. Percentage of missing information: 0.65%.

Heart Disease

The Heart Disease database contains information to diagnose a heart disease by deciding whether at least one of four major vessels is reduced in diameter by more than a 50%. It consists of a mixture of personal data, subjective patient descriptions and the results of several medical examinations. The task is to make a decision about "negative patients" (no vessel is reduced) and positive ones (one or more vessels reduced). Of the 920 records, 411 (44.7%) correspond to negative patients and 509 (55.3%) to positive ones.

This data set is in fact obtained as the union[3] of four independent locations:

1. Cleveland Clinic Foundation (303 records)
2. Hungarian Institute of Cardiology, Budapest (294 records)
3. University Hospital, Zurich, Switzerland (123 records)
4. V.A. Medical Center, Long Beach, CA (200 records)

While the databases have 76 raw attributes, only 14 of them have been actually used in past experiments (including the output one).

This data set shows also a very good variety of heterogeneity and a considerable amount of missing information (16.2%), affecting most of the variables. Of the thirteen variables, three are originally continuous, six are nominal and four are ordinal variables.

Particular comments: Similar arguments as those for *Pima Indians* and *Horse Colic* are applied to this data set. Variables number 1, 8 and 12 are ordinal because it makes much more sense than consider them as continuous, and the original measurements are always natural numbers. Their crisp nature makes them ordinal variables. The other ordinal variable (number 11) has not been converted to linguistic variable because the value boundaries appear to be crisp. The imprecise continuous variable, number 10, is converted into fuzzy numbers as usual, with a low fuzziness of a 0.5%. Variable number 6 is taken as nominal because again we are not interested in the order, but in the distinction of the two possible values. Absence of knowledge about possible ordering relations (not very likely, on the other hand) are the reason to take variables 3, 7 and 13 as nominal.

[3]Thanks to Andras Janosi, William Steinbrunn, Matthias Pfisterer and Robert Detrano, members of these institutions, for collecting the data.

A description follows:

1. Age in years [*ordinal*]
2. Sex (male, female) [*nominal*]
3. Chest pain type (typical angina, atypical angina, non-anginal pain, asymptomatic) [*nominal*]
4. Resting blood pressure (mm Hg) [*continuous*]
5. Serum cholesterol (mg/dl) [*continuous*]
6. Fasting blood sugar (less or more than 120 mg/dl) [*nominal*]
7. Resting electrocardiographic results (normal, ST-T wave abnormality, left ventricular hypertrophy) [*nominal*]
8. Maximum heart rate achieved [*ordinal*]
9. Exercise induced angina (yes, no) [*nominal*]
10. ST depression induced by exercise relative to rest [*continuous* → *fuzzy number*]
11. Slope of the peak exercise ST segment (downsloping, flat, upsloping) [*ordinal*]
12. Number of major vessels colored by flourosopy [*ordinal*]
13. Heart test (normal, fixed defect, reversable defect) [*nominal*]

Total: 2 continuous variables, 1 fuzzy number (0.5% fuzziness), 4 ordinal and 6 nominal variables. Percentage of missing information: 16.2%.

Solar Flares

The Solar Flares database contains information relevant for the prediction of solar activity. The task is to guess the number of solar flares of small, medium and large size that will happen during the next 24-hour period in a fixed and active region of the surface. The database consists of variables describing previous solar activity and the type and history of the region.

The distribution of the 1066 records is as follows:

	0	1	2	3	4	5	6	7	8	Total
C-class flares	884	112	33	20	9	4	3	0	1	1066
M-class flares	1030	29	3	2	1	0	1	0	0	1066
X-class flares	1061	4	1	0	0	0	0	0	0	1066

This data set is interesting in that there is no continuous information. Of the nine variables, five are nominal and four are ordinal variables. In addition, the information is complete. Notice that 81% of the cases are zero in all three output values. We have removed original attribute number 10, which happened to be constant for all the records.

Particular comments: The ordinal variables, as in other data sets, have been considered as linguistic because they represent subjective appreciations.

A description follows:

1. Code for class (modified Zurich class) (A,B,C,D,E,F,H) [*nominal*]
2. Code for largest spot size (X,R,S,A,H,K) [*nominal*]
3. Code for spot distribution (X,O,I,C) [*nominal*]
4. Activity (reduced, unchanged) [*ordinal* $\rightarrow$ *linguistic*]
5. Evolution (decay, stationary, growth) [*ordinal* $\rightarrow$ *linguistic*]
6. Previous 24 hour flare activity code (nothing as big as an M1, one M1, more activity than one M1) [*ordinal* $\rightarrow$ *linguistic*]
7. Historically complex (yes, no) [*nominal*]
8. Did region become historically complex on this pass across the sun's disk (yes, no) [*nominal*]
9. Area (small, large) [*ordinal* $\rightarrow$ *linguistic*]

From all these features three classes of flares are predicted, which are represented in the three outputs:

- C-class flares production. Number in the following 24 hours (common flares)
- M-class flares production. Number in the following 24 hours (moderate flares)
- X-class flares production. Number in the following 24 hours (severe flares)

Total: 5 nominal and 4 linguistic variables. Percentage of missing information: 0.0%.

The following two tasks are characterized by a lack of data heterogeneity –the two variables are perfectly continuous– and the absence of missing information.

Sinus-Cosinus

This task is a two-dimensional benchmark function cited in [Bersini and Bontempi, 97], where it is used for the comparison of several multimodeling approaches. It is defined as:

$$f : [-1, +1]^2 \rightarrow \mathbb{R}$$

where

$$f(x_1, x_2) = 4sin(\pi x_1) + 2cos(\pi x_2) + N(0, 0.5) \tag{9.1}$$

with $N(0, 0.5)$ a normal noise with zero mean and 0.5 standard deviation.

A learning data set of 400 input/output pairs is constructed by uniformly sampling the domain $[-1, +1]^2$.

Total: 2 continuous variables. Percentage of missing information: 0.0%.

SISO-Bench

This task is included as a representative of a continuous non-linear system identification problem (Single-input/Single-output), out of a number of points obtained by consecutively sampling the function dynamics. It has been used as a two-dimensional benchmark function elsewhere [Su and Sheen, 92, De Falco *et al.*, 97]. It is defined as:

$$y(k) = y_1(k-1) + y_2(k-1)$$

where

$$\begin{aligned} y_1(k) &= 2.5y(k)sin[\pi e^{-u^2(k)-y^2(k)}] \\ y_2(k) &= u(k)[1 + u^2(k)] \end{aligned} \tag{9.2}$$

The output $y(k)$ depends on the previous input $u(k-1)$ and on the previous output $y(k-1)$. A learning data set of 500 input/output pairs is constructed by setting $y(0) = 0$ and randomly exciting the system using a signal $u(k)$ uniformly drawn from $[-2.0, 2.0]$.

Total: 2 continuous variables. Percentage of missing information: 0.0%.

9.3 Experimental methodology

Comparability of the obtained results is assured by the fact that all the experiments are performed in exactly the same experimental conditions, in what regards to the general methodology, training procedure and original data sets. We have also made an effort to supply all relevant information, so that the experiments can be reproduced or extended. The methodology followed is in general accordance with suggested guidelines for neural network training experiments and presentation of results [Flexer, 95].

9.3.1 General methodology

The network architectures are fixed to one single layer of h_1 neurons plus as many output units as required by the task, sigmoidal for classification problems and linear otherwise.

The explored architectures are those with $h_1 \in \{4, 8, 12, 16\}$ neurons in the hidden layer. The intention here is not to search for the best model (in the sense of the best number of hidden layers and units per layer), which would need a full model selection process [Moody, 94]. Rather, the intention is to show how the results are consistent across different reasonable choices for the architecture. In addition, the obtained results could surely be improved by several means; for instance, using other error measures for classification problems (as the cross-entropy, the number of misclassified examples or a combination thereof). There is also the possibility that a more powerful form of evolutionary algorithm (as Evolution Strategies) or even a method based on derivative information could outperform the one used in these experiments. None the less, these are factors *external* to the neuron models, having to do with the learning algorithm.

The training methodology used is a simplified form of nested k-fold non-stratified cross-validation [Ripley, 92]. Non-stratified means that there is no guarantee that the folds are balanced with respect to the classes –in case it is a classification task– or to any other criterion. This can lead to significatively different results for the various partitions. To avoid predetermined initial arrangements, the data sets are randomly shuffled prior to the application of the methodology. Early stopping is used in conjunction with this technique because is simple and has been reported to be superior to regularization methods in many cases (e.g. [Finnof, Hergert and Zimmermann, 93]).

The basic idea behind k-fold cross-validation (CV) dates from the early seventies [Stone, 74]. Cross-validation is a very general framework using no special model assumptions. Although it cannot "validate" a model, it gives an unbiased estimate of generalization ability. The entire data set in divided into k pieces (the folds); one of them is kept for validation and the remaining $k-1$ are used for training. This is repeated k times changing the validation fold, and the results on these folds are averaged. Common values for k are 5 or 10.

The main advantage of using these resampling techniques is that they do not depend on assumptions on data statistics or on specific forms or properties of the approximating functions (i.e., by the different neural networks). Among their problems –besides the high computational demands– are the high variability and the need of a third and independent group of data to assess the actual performance of the models (since the method is biased to the validation data) and the difficulty to establish a stopping criterion for the training parts. The need for a third, *test* set, comes from the fact that, since the validation set is used in the training process, its error is not a reliable estimate of generalization error and can also lead to some overfit in the validation data. The obvious solution, consisting in holding back this third test set is data wasteful, since it should be sufficiently large, to be of any use.

The idea of nested or *double* cross-validation again dates from the seventies [Mosteller and Tukey, 78], and works as follows: divide the whole data set into k_1 pieces, keeping each back in turn for independent testing, and using the rest for k_2-fold CV. This fits $k_1 k_2$ models. If we use one of the k_1 pieces for test, one for validation and the remaining $k_1 - 2$ for training, this is equivalent to set $k_2 = k_1 - 1$, so that a total of $k_1(k_1 - 1)$ fits are to be done.

This scheme has two advantages. First, it offers a test set, different for each partition, not used in its corresponding training process, on which to estimate generalization performance.

Repeated use of a single test set for evaluating different models can lead to an overestimation of performance of a particular method; hence, retaining multiple test sets is a prudent measure [Fiesler and Beale, 97]. It also offers a validation set to be used to stop training, following a determined user criterion. The measured error on the unseen test set is then an unbiased estimate of generalization error [Cherkassky and Mulier, 98]. Second, it exploits available data to a greater extent than CV, and the size of the validation and test folds can be made equally balanced and more significant in size than would have been in a 10-fold CV (only 10% of the data), so that the error generalization estimate is more accurate. In particular, [Michie, Spiegelhalter and Taylor, 94] recommend to hold back approximately a 20% of the data for testing.

Early stopping is used in conjunction with cross-validation as follows. The network is trained on the training part while keeping track of the best validation error. In each fit, the network can be trained to convergence or to end of resources. Both criteria can be utilised in general, regardless of the particular training algorithm, or else they can be combined. In the present case, we use the convention introduced in Chapter (6), given by a maximum number of error function evaluations. Strictly speaking, there is no early stopping here because the process is not halted when a minimum of the validation error is attained. Rather, the networks are always trained to end of resources because we are interested in assessing both the ability to approximate and the ability to generalize of each model. After the process has ended, the first measure is given by the final error attained on the training part, while the second is given by the error on the test part using the network that produced the lowest error in the validation part. Incidentally, this scheme avoids the problem of deciding when to halt training in an early stopping process [Prechelt, 98].

In general, the number of resulting fits $F(k)$ is obtained by considering the possible ways of selecting 2 sets (test and validation) out of k parts of the entire data set, and where the order of the former two is important. That is,

$$F(k) = 2\binom{k}{2} = k(k-1)$$

In these experiments, we work out a simplification to avoid an excessive computational overhead, by performing only a number of the computations involved. In particular, we set $k = 5$, so that three folds go for training, one for validation and one for the test (thus forming 60%-20%-20% partitions). A representative half (ten possibilities) of the total amount ($F(5)$ or twenty) can be obtained by selecting a balanced subset of combinations such that each of the 5 folds appears twice as a validation fold, twice as a test, and 6 times (three times two) in one of the three training positions. It is also ensured that no two training parts are generated out of the same three folds but in a different order. This could certainly make a difference for learning algorithms that are dependent on the order of presentation of training data (such as on-line versions of backpropagation). It is not an issue here because the BGA is insensitive to such order.

The resulting method is analogous to simple 10-fold CV in what regards the amount of computation and number of fits, and allows the use of both validation and test sets, making full treatment of the data available. For each of the ten selected partitions, ten runs are carried out, varying the random initial population of the BGA. This means that *a total of 100 fits* per dataset, architecture and model are performed.

9.3.2 Models tested

For each architecture and data set, three instances of the general heterogeneous neuron model are compared. Two of them correspond to the P-neuron and R-neuron commonly used in MLP and RBF networks, respectively, as defined in Chapter (2.1.2). These will be generally referred to as the standard or classical models. The third neural model is obtained by application of the proposed approach in accordance with the decisions taken in (§9.2.2). This one will be simply referred to as the H-neuron. Specifically, given an input pattern $\vec{x}$ the models compute the following functions $F_i(\vec{x})$:

P-neuron The standard scalar-product neuron plus a bias weight, using a logistic as activation function:

$$F_i(\vec{x}) = g(\vec{x} \cdot \vec{w_i} + \theta_i) \tag{9.3}$$

where $\vec{w_i}$ is the weight vector of neuron i, $g(z) = \frac{0.55z}{|z|+1} + 0.5$ is a sigmoidal similar to the logistic (though smoother and cheaper to compute) and θ_i is the bias weight.

R-neuron A radial basis function neuron based on Euclidean distance, followed by a Gaussian with its own variance (to be learned) as activation function:

$$F_i(\vec{x}) = g(||\vec{x} - \vec{w_i}||_2) \tag{9.4}$$

where $g(z) = e^{-\frac{1}{2}\frac{z^2}{\sigma_i^2}}$ and σ_i^2 is the variance.

H-neuron An heterogeneous neuron model based on the measure in (§4.4.1) (with $s_{max} = 1$), obtained as a simple additive similarity aggregation operator, followed by a non-linear similarity-keeping or $\check{s}$ function, acting as a logistic activation function by adapting it to the real domain $[0, 1]$. The neuron model is as follows:

$$F_i(\vec{x}) = \check{s}\left(\frac{\sum_{k=1}^{n} s_k(x_k, w_{ik})\, \delta_k(\vec{x}, \vec{w_i})}{\sum_{k=1}^{n} \delta_k(\vec{x}, \vec{w_i})}\right) \tag{9.5}$$

where $\check{s}(z) = g(z, k)$ (4.74) with $k = 0.1$, and $\delta_k(\vec{x}, \vec{y}) = \begin{cases} 1 & \text{if } x_k \neq \mathcal{X} \wedge y_k \neq \mathcal{X} \\ 0 & \text{otherwise} \end{cases}$, being $\mathcal{X}$ the missing information symbol.

The partial similarities s_k between the variables are computed using the partial similarity measures defined in (4.3), chosen accordingly to each data set as described in (§9.2.2). Specifically, the partial measures used are those defined in (4.52) for continuous variables (using $d = 1, \alpha = 4$) –corresponding to the function S0_3 in the table of similarity transforming functions (4.1). For ordinal, nominal, fuzzy and linguistic variables, their scores can be computed by using the measures in (4.46), (4.45), (4.60) and (4.63), respectively.

The expression (9.5) is an example of a similarity index designed out of partial measures of two basic types, distance-based (continuous, ordinal, nominal), and direct (fuzzy numbers and linguistic variables), whilst the RBF (9.4) and MLP (9.3) models are examples of measures purely of type (A) and (C), respectively –see (§4.2.4).

We define the *network complexity* as the number of network free parameters. In general, this quantity depends on the number $\hat{n}$ of model inputs (per unit) and the number and disposition of the units. In our present study, given a network with a hidden layer of h_1 neurons and an output layer of m P-neurons, the complexity $\hat{p}$ of a network is given by:

$$\hat{p}(\hat{n}, h_1, m) = h_1\hat{n} + (h_1 + 1)m \tag{9.6}$$

For the standard models, $\hat{n} = n_s + 1$, where n_s is the number of standard *model* inputs. For the heterogeneous neurons, $\hat{n} = n_n + n_o + n_r + 2n_f + 4n_v$, being n_n, n_o, n_r, n_f, n_v the respective cardinalities of the different data types, as defined in (§4.4), for each considered problem. Their sum corresponds to the number of heterogeneous *model* inputs.

9.3.3 Data preparation

The original information is the same for all the models. The decisions taken concerning the type of each variable is valid for the three models. In other words, the type is decided by the variable itself. Then, each of them treats this information in its own particular way until it finally yields performance results. The real-valued input variables are normalized to $[0, 1]$. This is not needed by the H-neuron because it computes a normalized measure, but is beneficial for the standard models. The output is not normalized.

The corresponding data sets for the standard neurons are constructed using the more widespread and possibly less distorting techniques of those explained in (§2.1.9). Specifically, ordinal variables are mapped to an equidistant linear scale, a 1-out-of-k encoding is used for nominal ones and an extra input is added for those variables with missing values.

The weights (including biases and standard deviations) for the classical models are let to vary in $[-10, 10]$, a sufficiently wide range given the normalization chosen and the number of hidden neurons; the same interval is used for the hidden-to-output weights in all the networks.

9.3.4 Training procedure

The training procedure used is the *Breeder Genetic Algorithm* (BGA) developed in Chapter (6). It is used with exactly the same parameter setup for all the experiments, regardless of the architecture, data set or neuron model, to exclude this source of variation from the analysis. The BGA task is to minimize MSE (mean square error) on the training part, until 30,000 error evaluations are used –end of resources– in each run. As commented, ten independent runs are performed for each specific training scenario.

The BGA is set to the following parameters: $\mu = 100, \tau = 25$, EIR recombination with $\delta = 0.45$, and continuous mutation with $\rho = 0.5, k = 8$, following the recommendations for ANN optimization found in Chapter (§6), concerning possible parameter sets and genetic operators. For *ordinal* variables, Line Recombination (LR) is used. A total of 300 (= 30,000/100) generations are carried out in each run.

9.4 Results

9.4.1 Presentation of results

For each tested model, the main reported quantity is the mean *normalized mean square error* (NMSE) given in (7.2), plus/minus one normalized standard deviation, in the usual form. This measure gives an impression of how good a result is and, being normalized, permits the comparison across different data sets. Reasonable values are to be expected in $[0, 1]$. Values lower than 0.5 indicate fair fits. Good fits are approximately signaled by values lower than 0.1, whereas values greater than 1.0 indicate a poor performance. A value of 1.0 actually indicates a model as good as the average predictor $\vec{\zeta_i} =< \vec{y_i} >$ in (7.2).

For simplicity, let X in {TR, VA, TE} stand for a training, validation and test part, respectively. A summary of displayed information follows:

NMSE (X) Mean NMSE across the 100 fits in part X.

% (X) Mean classification accuracy across the 100 fits in part X.

NMSEb (X) Mean of the best NMSE found across the 10 partitions in part X.

%b (X) Mean of the best classification accuracies found across the 10 partitions in part X.

The full results are presented in table format at the end of the Chapter. For every model, the first two columns of a table correspond to training results, computed using the net found at the end of the training process, and collectively measure the approximation ability of the studied model. Shown are **NMSE (TR)** and **% (TR)** (mean performance) and **NMSEb(TR)** and **%b(TR)** (mean of best performances). The training results are included to assess the extent to which the different models can approximate the training set (towards the theoretical, though in general undesirable optimum of 100% accuracy or a null square error). Mean classification accuracies are shown where appropriate.

The next two columns stand for the same information relative to test. The shown values for **NMSE (TE)**, **% (TE)**, **NMSEb (TE)** and **%b (TE)** are computed using the net found, at any time of each training process, having the lowest NMSE in the VA part, and collectively measure the generalization ability of the model. The estimations (very likely to be optimistic) for the VA part are not shown. All mean NMSE values are displayed plus/minus one normalized standard deviation, as indicated, in the format $\hat{\mu} \pm \frac{\hat{\sigma}}{\sqrt{n}}$, where $\hat{\mu}, \hat{\sigma}$ denote the sample mean and sample standard deviation over the number of samples $n = 100$. These values are the confidence intervals: with 99% probability, the true value fo the observed mean will be within $\hat{\mu} \pm 2.58\frac{\hat{\sigma}}{\sqrt{n}}$; with 95% probability, it will be within $\hat{\mu} \pm 1.96\frac{\hat{\sigma}}{\sqrt{n}}$. Confidence intervals allow to compare the results of two different methods to the same data. Roughly speaking, if the two intervals are non-overlapping, there is a statistically significant difference between the two means; otherwise, an adequate test should be performed [Flexer, 95]. In our case, we will carry out a Mann-Whitney non-parametric test on performance data [Steel and Torrie, 80].

The most important quantities are undoubtedly **NMSE (TE)** and **% (TE)**, (the third column) which indicate the average performance of the models in terms of generalization ability. They are representative of what can be expected of a single run in an undetermined partition. The values **NMSEb (TE)** and **%b (TE)** (the fourth column) are indicative of what can be achieved on average in a modest number of runs (ten).

Also shown are the mean initial and final similarities $\hat{\beta}_i$, $\hat{\beta}_f$, measures of network sensitivity, computed as in (4.90) for the initially random and finally trained networks, respectively, and the network complexity —as number of *model* inputs $\hat{n}$ and number of free network parameters $\hat{p}$ (the latter in parentheses)— computed according to (9.6). Recall that the number of *problem* inputs was shown in Table (9.1).

9.4.2 Summary of results

In general, performance results can be measured across a handful of coordinates: generalization ability, model complexity, readability and computational cost.

The obtained **generalization results** for all architectures, data sets and models explored are summarized as follows:

1. In Table (9.2), for the networks corresponding to the three studied neuron models, the obtained **NMSE (TE)** values are tested for significance under a Mann-Whitney non-parametric test [Steel and Torrie, 80]. Each entry $t[i,j]$ in the table expresses the number of significant tests, with the hypothesis "i is less than j", performed for the three models on all combinations of data sets and architectures.

	P-neurons	R-neurons	H-neurons
P-neurons	-	16 (57.1%)	3 (10.7%)
R-neurons	12 (42.9%)	-	8 (28.6%)
H-neurons	25 (89.3%)	20 (71.4%)	-

Table 9.2: Number of significant Mann-Whitney tests for "less than" at the 95% confidence level, among the total performed.

The total number of tests between any two neuron models is $n_t = 7 \times 4 = 28$ (7 data sets and 4 architectures). Note that $t[i,j] + t[j,i] = n_t, i \neq j$. The percentage (w.r.t. n_t) is given in parentheses.

2. In Table (9.3), the results are summarized for the different data sets, averaged over the four architectures.

Problem	P-neuron	R-neuron	H-neuron
Pima Diabetes	0.6916	0.7738(*)	0.7001
Horse Colic	0.9255(*)	0.7137	0.7246
Heart Disease	0.6611(*)	0.5857	0.5813
Credit Card	0.6768	0.7397(*)	0.5330
Solar Flares	1.1816(*)	0.8553	0.9374
Sinus-Cosinus	0.0830	0.3839(*)	0.0424
SISO-Bench	0.0218	0.1145(*)	0.0234

Table 9.3: Results for each data set, averaged over the four architectures. An asterisk (*) means the result is clearly worse than any of the other two, in the sense that, for *all* of the four architectures, Mann-Whitney tests for "greater than" are significant at the 95% level, w.r.t. *both* of the other two models.

For these difficult data sets, there is no result outstandingly better than the other two. As it can be seen, the cornerstone of the networks grounded on H-neurons relies in its good average behaviour. The results for these networks are never the worse of the three (that is, they are always one of the better two) but are the best in 3 out of 7 data sets. When second best, they are close to the best one.

3. Additionally, in Table (9.4), the results are summarized from the perspective of the architectures, averaged over the seven data sets (this makes sense because the errors are normalized).

Architecture	P-neuron	R-neuron	H-neuron
4 hidden	0.5757	0.6115	0.5056
8 hidden	0.5801	0.5966	0.4991
12 hidden	0.6160	0.5880	0.5037
16 hidden	0.6455	0.5850	0.5157

Table 9.4: Results for the architectures, averaged over the seven data sets.

In this last table an interesting side remark can be made, concerning the way the different networks behave w.r.t. the number of hidden units.

- The generalization ability of the MLP (based on P-neurons) tends to be *worse* with increasing numbers of hidden units.
- The generalization ability of the RBF (based on R-neurons) tends to be *better* with increasing numbers of hidden units.

- The HNN (based on H-neurons) does not show a definite trend. In a sense, it looks more stable, though this effect should be investigated in detail in further experiments.

Concerning the **network sensitivities** (expressing average similarities across all patterns and hidden neurons), there is much information contained in the initial estimation $\hat{\beta}_i$ (random, untrained networks) and final $\hat{\beta}_f$ (trained networks), telling us about the nature of the involved nets:

1. As could be expected, for all neuron models (P, R and H), the $\hat{\beta}_i$ yield the same value for same problems, regardless of the number of hidden units. In particular, for the P-neuron model, these values (in all cases shown rounded to the third decimal) are the same for all data sets (and approximately equal to 0.5).

2. Concerning the final value for the P-neurons, it shows an increasing trend with increasing numbers of hidden units. The specific values taken are dependent on the data set.

3. Regarding the initial value for the R-neurons, it always begins at a very low value, reflecting the initially random placement of the RBF centers in input space. The final values follow an inverse trend compared to the P-neurons, decreasing with increasing numbers of hidden units, since neurons in a bigger network can be more specialized.

   In general, though, the $\hat{\beta}_f$ are always substantially higher than the $\hat{\beta}_i$, showing the effect of the learning process, which has placed the centers in locations where they yield a significant response, according to the probability distribution of the training set.

4. The H-neurons do not follow any particular trend with changing numbers of hidden units. Rather, the final values attained are found to be reasonably similar, as if there were a preferred range of values for each data set. Specifically, these ranges are: $[0.202, 0.226]$ for *Pima Diabetes*, $[0.103, 0.137]$ for *Horse Colic*, $[0.126, 0.195]$ for *Credit Card*, $[0.411, 0.421]$ for *Heart Disease*, $[0.050, 0.052]$ for *Solar Flares*, $[0.330, 0.336]$ for *Sinus-Cosinus* and $[0.286, 0.335]$ for *SISO-Bench*.

   This behaviour is specific of the HNN (the other two models are also within given ranges, but these are wider and, as stated, show definite monotonic trends).

Concerning the **network complexities**, these are lower for the HNN (sometimes substantially), while the MLP and RBF have equal numbers. It can be argued that, in some problems, the number of network free parameters $\hat{p}$, could well be comparable to that of the classical models, and even exceed it, specially in the presence of linguistic variables, although this does not happen in any of the studied datasets. Nonetheless, this would stand to reason; in the case of linguistic variables, for example, the variable is actually expressing a vague concept, and this requires a representation with four (or whatever number) real parameters. Yet, this is not an *encoding*, but the parameters themselves characterizing the variable. Besides, they are all treated as a whole by the neuron model *and* the learning algorithm, and not as separate, independent entities.

The influence of having high numbers of network parameters is seen in the results in various places, as a manifestation of the curse of dimensionality: first, it shows in the obtained zeros in the RBF results for $\hat{\beta}_i$ for *Horse Colic* and *Credit Card* (the actual values are on the order of 10^{-5}). These two are, by large, the data sets with a bigger number $\hat{n}$ of input dimensions (54 and 51, respectively). As a consequence, for these data sets, the input space is initially seldom sampled.

Second, for 16 hidden units, the effect begins to come into play in the training process itself. The increase in free parameters leads to values $\hat{p} = 931$ for *Horse Colic* and $\hat{p} = 866$ for *Credit Card*. In general, a bigger number of parameters is less likely to be properly constrained by a limited size data set [Bishop, 95]. The number of training patterns is 182 and 345, respectively. This translates in the quality of the training results, which for *Horse Colic* are worse than those with 12 hidden units, for the MLP, and about the same for the RBF. The HNN ($\hat{p} = 787$) still gives a somewhat lower error than for 12 units. For the *Credit Card* problem, analogous results are obtained: both MLP and RBF networks give worse results, contrary to the HNN, having only $\hat{p} = 370$ parameters.

The **readability** of the obtained solutions is illustrated in Table (9.5) as follows. We show the heterogeneous weights of a hidden neuron taken at random from one of the hundred networks delivered by cross-validation for the *Horse Colic* problem. This task is chosen because it displays a good amount of heterogeneity —see (9.2.2) for a description of the variables.

The (triangular) fuzzy numbers are shown in numerical form –rounded to one decimal– for clarity. Note how the obtained linguistic terms are symmetric, a characteristic found by the network itself. Although it would require an expert judgement, by looking at the obtained weights, the neuron could be regarded as the *soft* prototype of a somewhat "standard" young horse, showing values within normal tolerances for the considered variables.

An added advantage comes from the fact that we *know* how this information is used by the neuron to yield an output value s in presence of an input vector $\vec{x}$: s is a similarity degree in $[0,1]$ between $\vec{x}$ and the weights $\vec{w}^i$, whose precise form is set a priori by the network designer. Particular choices for the similarities between the different data types were chosen, and the overall result is, in this case, the average of their outcomes. This permits to assign a well-defined meaning to a neuron outcome and at the same time to interpret the trained network in the original input domain (e.g., a set of measurements about horses).

In contrast, the weight vector for a hidden P-neuron or R-neuron consists for this problem of 467 real numbers in the interval $[-10, 10]$. That is, for these neurons all values are independently and equally treated, in the form $\sum_j w_{ij}x_i$ and $\sum_j (w_{ij} - x_i)^2$, respectively, where $\vec{x}, \vec{w}^i$ are points in $\mathbb{R}^{467}$.

The **computational cost** associated to each neuron model is estimated as follows. From the total CPU time of each training session (for all data sets and architectures) the total time taken by a hidden neuron is computed, normalizing by the number of patterns, and averaging across the different data sets. The resulting quantity is then divided by the number of times a neuron is asked to process the entire data set which, in this study, amounts to $10 \times 10 \times 30,000$ (10 folds, 10 runs per fold, 30,000 data set evaluations per run). The obtained quantities are

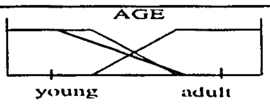
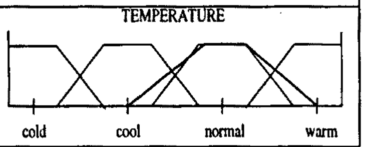
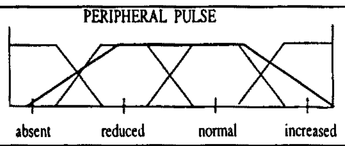
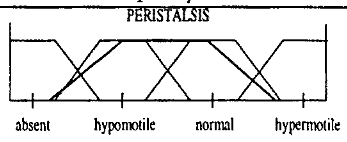
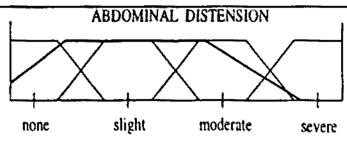
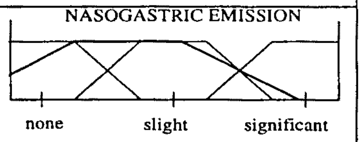
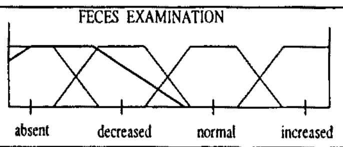
#	Name	Type	Value
1	Age	*linguistic*	
2	Rectal temperature (celsius)	*fuzzy number*	37.3 ± 2.1
3	Pulse (beats per minute)	*ordinal*	70
4	Respiratory rate (times per minute)	*ordinal*	64
5	(Subjective) temperature of extremities	*linguistic*	
6	(Subjective) peripheral pulse	*linguistic*	
7	Mucous membranes color	*nominal*	normal pink
8	Capillary refill time judgement	*nominal*	less than 3 secs.
9	(Subjective) pain estimation	*nominal*	no pain/alert
10	Peristalsis: gut activity	*linguistic*	
11	Abdominal distension	*linguistic*	
12	Nasogastric tube gas emission	*linguistic*	
13	Nasogastric reflux	*ordinal*	none
14	Nasogastric reflux PH	*fuzzy number*	2.1 ± 0.3
15	Rectal examination of feces	*linguistic*	
16	Abdomen	*nominal*	distended small intestine
17	Packed cell volume	*fuzzy number*	34.1 ± 3.1
18	Total protein (grs./dl)	*fuzzy number*	73.6 ± 19.8
19	Abdominocentesis appearance	*nominal*	cloudy
20	Abdominocentesis total protein (grs./dl)	*fuzzy number*	1.4 ± 1.4

Table 9.5: Heterogeneous weights of a hidden neuron corresponding to one of the networks obtained for the *Horse Colic* problem.

reported in Table_(9.6), as the average number of CPU milliseconds that a hidden neuron takes to evaluate one pattern of the training set, across all the data sets[4].

	P-neuron	R-neuron	H-neuron
time (ms)	0.00125	0.00184	0.00175

Table 9.6: Average number of CPU milliseconds that a hidden neuron takes to evaluate one pattern of the training set.

It has to be noted that the activation function used by the P-neuron (9.3) was chosen for its cheap cost (it only involves a division as a non-trivial operation) against the classical logistic (which involves an *exp* operation and a division). The R-neuron (9.3) uses a Gaussian activation involving the costly *exp* operation. The H-neuron uses (9.5) uses a cheap (4.74) as activation (involving only a division as a non-trivial operation) thus comparable to that of the P-neuron. However, some of the partial similarities are almost free (e.g. for nominal variables) while others are costly (e.g. for linguistic ones). For ordinal and real-valued variables, implying a subtraction and a division, the cost is comparable to the P-neuron, since the normalization factors for the distances (the denominators), being constant, can be pre-calculated in the form of its inverses (thus converting the division to a multiplication).

9.5 Detailed results

The complete results follow. They are displayed in Tables (9.7) to (9.34) for the different data sets, architectures and models explored.

[4]The machine used for the experiments is a dedicated SUN^{TM} Enterprise E-250 (2 CPU ULTRA-SPARCII at 400Mhz, 128Mb RAM), rated at 16.8 SPECint95 and 13.5 SPECint_base95 marks.

Neuron Model	NMSE % (TR)	NMSEb %b (TR)	NMSE % (TE)	NMSEb %b (TE)	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.5296 ± 0.0036 80.93	0.5095 83.16	0.6759 ± 0.0109 75.05	0.6119 78.56	0.500 0.229	13 (66)
R-neuron	0.7714 ± 0.0035 69.31	0.7300 73.20	0.7856 ± 0.0060 68.82	0.7464 72.75	0.022 0.271	13 (66)
H-neuron	0.5068 ± 0.0038 82.43	0.4839 83.85	0.6826 ± 0.0094 75.13	0.6342 77.91	0.191 0.226	10 (50)

Table 9.7: Problem: *Pima Diabetes*. Architecture: 4 hidden.

Neuron Model	NMSE % (TR)	NMSEb %b (TR)	NMSE % (TE)	NMSEb %b (TE)	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.5042 ± 0.0035 82.00	0.4805 83.55	0.6777 ± 0.0102 75.08	0.6258 78.17	0.502 0.272	13 (130)
R-neuron	0.7567 ± 0.0036 70.50	0.7143 74.46	0.7724 ± 0.0060 70.14	0.7319 73.86	0.022 0.176	13 (130)
H-neuron	0.4609 ± 0.0037 84.69	0.4360 86.30	0.6885 ± 0.0110 75.27	0.6312 78.04	0.191 0.212	10 (98)

Table 9.8: Problem: *Pima Diabetes*. Architecture: 8 hidden.

Neuron Model	NMSE % (TR)	NMSEb %b (TR)	NMSE % (TE)	NMSEb %b (TE)	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.4919 ± 0.0034 82.82	0.4659 84.63	0.7012 ± 0.0121 74.71	0.6407 78.10	0.501 0.326	13 (194)
R-neuron	0.7514 ± 0.0036 71.06	0.7117 74.70	0.7677 ± 0.0059 70.55	0.7251 74.12	0.022 0.125	13 (194)
H-neuron	0.4393 ± 0.0040 85.67	0.4081 87.38	0.7082 ± 0.0127 74.71	0.6495 77.52	0.191 0.202	10 (146)

Table 9.9: Problem: *Pima Diabetes*. Architecture: 12 hidden.

Neuron Model	NMSE % (TR)	NMSEb %b (TR)	NMSE % (TE)	NMSEb %b (TE)	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.4909 ± 0.0037 83.10	0.4638 85.04	0.7115 ± 0.0124 75.12	0.6466 78.37	0.501 0.373	13 (258)
R-neuron	0.7535 ± 0.0036 70.88	0.7142 74.50	0.7695 ± 0.0055 70.42	0.7336 73.92	0.022 0.106	13 (258)
H-neuron	0.4244 ± 0.0038 86.48	0.3945 88.29	0.7210 ± 0.0135 74.58	0.6536 77.39	0.191 0.202	10 (194)

Table 9.10: Problem: *Pima Diabetes*. Architecture: 16 hidden.

Neuron	NMSE	NMSEb	NMSE	NMSEb	$\hat{\beta}_i$	$\hat{n}$
Model	% (TR)	%b (TR)	% (TE)	%b (TE)	$\hat{\beta}_f$	$(\hat{p})$
P-neuron	0.3317 ± 0.0052	0.2599	0.8035 ± 0.0089	0.7190	0.498	54
	85.71	89.50	62.69	67.22	0.389	(235)
R-neuron	0.6887 ± 0.0046	0.6441	0.7228 ± 0.0051	0.6941	0.000	54
	65.01	68.95	63.57	65.97	0.168	(235)
H-neuron	0.4734 ± 0.0031	0.4357	0.7196 ± 0.0060	0.6717	0.080	46
	79.20	81.95	64.65	69.86	0.137	(199)

Table 9.11: Problem: *Horse Colic.* Architecture: 4 hidden.

Neuron	NMSE	NMSEb	NMSE	NMSEb	$\hat{\beta}_i$	$\hat{n}$
Model	% (TR)	%b (TR)	% (TE)	%b (TE)	$\hat{\beta}_f$	$(\hat{p})$
P-neuron	0.3128 ± 0.0065	0.2218	0.9260 ± 0.0112	0.8026	0.499	54
	85.45	91.59	62.32	67.64	0.444	(467)
R-neuron	0.6667 ± 0.0031	0.6434	0.7091 ± 0.0050	0.6893	0.000	54
	66.55	68.86	64.25	66.39	0.110	(467)
H-neuron	0.3970 ± 0.0027	0.3668	0.7179 ± 0.0073	0.6509	0.080	46
	83.86	86.55	65.67	70.69	0.119	(395)

Table 9.12: Problem: *Horse Colic.* Architecture: 8 hidden.

Neuron	NMSE	NMSEb	NMSE	NMSEb	$\hat{\beta}_i$	$\hat{n}$
Model	% (TR)	%b (TR)	% (TE)	%b (TE)	$\hat{\beta}_f$	$(\hat{p})$
P-neuron	0.3444 ± 0.0079	0.2331	0.9808 ± 0.0116	0.8523	0.501	54
	82.15	89.64	62.42	67.92	0.478	(699)
R-neuron	0.6695 ± 0.0029	0.6468	0.7119 ± 0.0049	0.6933	0.000	54
	65.91	68.36	64.26	66.25	0.073	(699)
H-neuron	0.3666 ± 0.0029	0.3405	0.7239 ± 0.0076	0.6610	0.080	46
	85.70	87.91	65.24	70.42	0.110	(591)

Table 9.13: Problem: *Horse Colic.* Architecture: 12 hidden.

Neuron	NMSE	NMSEb	NMSE	NMSEb	$\hat{\beta}_i$	$\hat{n}$
Model	% (TR)	%b (TR)	% (TE)	%b (TE)	$\hat{\beta}_f$	$(\hat{p})$
P-neuron	0.4002 ± 0.0104	0.2802	0.9915 ± 0.0122	0.8687	0.500	54
	78.14	86.23	62.54	67.08	0.488	(931)
R-neuron	0.6677 ± 0.0021	0.6503	0.7111 ± 0.0050	0.6955	0.000	54
	66.28	68.32	64.26	65.97	0.057	(931)
H-neuron	0.3506 ± 0.0029	0.3224	0.7370 ± 0.0079	0.6761	0.080	46
	86.61	89.05	64.75	68.61	0.103	(787)

Table 9.14: Problem: *Horse Colic.* Architecture: 16 hidden.

Neuron Model	NMSE % (TR)	NMSEb %b (TR)	NMSE % (TE)	NMSEb %b (TE)	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.3682 ± 0.0029 88.21	0.3387 90.45	0.6104 ± 0.0077 79.94	0.5226 83.75	0.499 0.298	33 (146)
R-neuron	0.5827 ± 0.0029 81.79	0.5496 82.52	0.6002 ± 0.0034 80.11	0.5684 81.41	0.001 0.271	33 (146)
H-neuron	0.4423 ± 0.0022 85.37	0.4253 86.41	0.5552 ± 0.0062 80.91	0.5054 83.37	0.373 0.421	14 (74)

Table 9.15: Problem: *Heart Disease*. Architecture: 4 hidden.

Neuron Model	NMSE % (TR)	NMSEb %b (TR)	NMSE % (TE)	NMSEb %b (TE)	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.3203 ± 0.0032 90.44	0.2854 92.05	0.6390 ± 0.0086 80.13	0.5562 82.50	0.500 0.349	33 (290)
R-neuron	0.5635 ± 0.0025 81.91	0.5301 82.64	0.5834 ± 0.0029 80.11	0.5531 81.41	0.001 0.171	33 (290)
H-neuron	0.4143 ± 0.0023 87.22	0.3939 88.44	0.5742 ± 0.0073 81.12	0.5197 83.26	0.373 0.414	14 (130)

Table 9.16: Problem: *Heart Disease*. Architecture: 8 hidden.

Neuron Model	NMSE % (TR)	NMSEb %b (TR)	NMSE % (TE)	NMSEb %b (TE)	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.3065 ± 0.0033 91.37	0.2723 92.84	0.6767 ± 0.0094 80.28	0.5792 82.99	0.499 0.388	54 (434)
R-neuron	0.5593 ± 0.0027 81.97	0.5205 82.74	0.5793 ± 0.0034 80.09	0.5483 81.47	0.001 0.124	54 (434)
H-neuron	0.4045 ± 0.0024 87.91	0.3827 88.97	0.5817 ± 0.0082 81.42	0.5207 84.02	0.373 0.411	46 (194)

Table 9.17: Problem: *Heart Disease*. Architecture: 12 hidden.

Neuron Model	NMSE % (TR)	NMSEb %b (TR)	NMSE % (TE)	NMSEb %b (TE)	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.3083 ± 0.0033 91.57	0.2714 92.79	0.7183 ± 0.0087 79.77	0.6303 82.55	0.500 0.418	33 (578)
R-neuron	0.5610 ± 0.0027 81.89	0.5277 82.68	0.5800 ± 0.0029 80.23	0.5513 81.74	0.001 0.094	33 (578)
H-neuron	0.3959 ± 0.0023 88.62	0.3691 89.62	0.6142 ± 0.0088 80.82	0.5448 83.04	0.373 0.419	14 (258)

Table 9.18: Problem: *Heart Disease*. Architecture: 16 hidden.

Neuron	**NMSE**	**NMSEb**	**NMSE**	**NMSEb**	$\hat{\beta}_i$	$\hat{n}$
Model	**% (TR)**	**%b (TR)**	**% (TE)**	**%b (TE)**	$\hat{\beta}_f$	$(\hat{p})$
P-neuron	0.1793 ± 0.0072	0.1495	0.6754 ± 0.0288	0.5189	0.498	51
	95.10	96.26	79.78	85.58	0.363	(218)
R-neuron	0.6182 ± 0.0099	0.5442	0.7660 ± 0.0211	0.6679	0.000	51
	83.54	87.51	73.67	82.10	0.209	(218)
H-neuron	0.2711 ± 0.0062	0.2479	0.5446 ± 0.0207	0.4766	0.087	21
	92.07	93.38	81.08	84.06	0.195	(94)

Table 9.19: Problem: *Credit Card.* Architecture: 4 hidden.

Neuron	**NMSE**	**NMSEb**	**NMSE**	**NMSEb**	$\hat{\beta}_i$	$\hat{n}$
Model	**% (TR)**	**%b (TR)**	**% (TE)**	**%b (TE)**	$\hat{\beta}_f$	$(\hat{p})$
P-neuron	0.1521 ± 0.0057	0.1250	0.6592 ± 0.0308	0.5018	0.499	51
	96.06	96.91	81.07	85.58	0.391	(434)
R-neuron	0.5865 ± 0.0068	0.5467	0.7290 ± 0.0161	0.6727	0.000	51
	85.42	87.20	76.80	80.72	0.129	(434)
H-neuron	0.2166 ± 0.0066	0.1923	0.5273 ± 0.0252	0.4590	0.087	21
	94.07	95.10	81.94	85.22	0.153	(186)

Table 9.20: Problem: *Credit Card.* Architecture: 8 hidden.

Neuron	**NMSE**	**NMSEb**	**NMSE**	**NMSEb**	$\hat{\beta}_i$	$\hat{n}$
Model	**% (TR)**	**%b (TR)**	**% (TE)**	**%b (TE)**	$\hat{\beta}_f$	$(\hat{p})$
P-neuron	0.1528 ± 0.0055	0.1230	0.6923 ± 0.0321	0.5551	0.500	54
	96.08	97.00	80.51	84.57	0.425	(650)
R-neuron	0.5883 ± 0.0064	0.5527	0.7317 ± 0.0154	0.6865	0.000	54
	85.66	87.15	77.25	80.36	0.088	(650)
H-neuron	0.1981 ± 0.0068	0.1744	0.5236 ± 0.0246	0.4582	0.087	46
	94.90	95.97	82.23	85.14	0.137	(278)

Table 9.21: Problem: *Credit Card.* Architecture: 12 hidden.

Neuron	**NMSE**	**NMSEb**	**NMSE**	**NMSEb**	$\hat{\beta}_i$	$\hat{n}$
Model	**% (TR)**	**%b (TR)**	**% (TE)**	**%b (TE)**	$\hat{\beta}_f$	$(\hat{p})$
P-neuron	0.1596 ± 0.0058	0.1292	0.6802 ± 0.0317	0.5528	0.500	51
	95.91	96.86	81.64	85.65	0.422	(866)
R-neuron	0.5904 ± 0.0063	0.5520	0.7321 ± 0.0158	0.6651	0.000	51
	85.49	87.13	77.01	81.59	0.066	(866)
H-neuron	0.1910 ± 0.0069	0.1743	0.5365 ± 0.0269	0.4638	0.087	21
	95.20	96.04	82.10	85.00	0.126	(370)

Table 9.22: Problem: *Credit Card.* Architecture: 16 hidden.

Neuron Model	NMSE	NMSEb	NMSE	NMSEb	$\hat{\beta}_i$	$\hat{n}$
	–	–	–	–	$\hat{\beta}_f$	$(\hat{p})$
P-neuron	0.6445 ± 0.0126	0.5958	1.0779 ± 0.0627	0.8413	0.500	23
	-	-	-	-	0.101	(111)
R-neuron	0.7790 ± 0.0158	0.7626	0.8568 ± 0.0463	0.8284	0.003	23
	-	-	-	-	0.083	(111)
H-neuron	0.7731 ± 0.0164	0.7484	0.9177 ± 0.0469	0.8623	0.091	21
	-	-	-	-	0.050	(99)

Table 9.23: Problem: *Solar Flares*. Architecture: 4 hidden.

Neuron Model	NMSE	NMSEb	NMSE	NMSEb	$\hat{\beta}_i$	$\hat{n}$
	–	–	–	–	$\hat{\beta}_f$	$(\hat{p})$
P-neuron	0.6056 ± 0.0123	0.5645	1.0903 ± 0.0616	0.8528	0.501	23
	-	-	-	-	0.123	(219)
R-neuron	0.7803 ± 0.0162	0.7641	0.8563 ± 0.0457	0.8248	0.003	23
	-	-	-	-	0.047	(219)
H-neuron	0.7714 ± 0.0162	0.7484	0.9298 ± 0.0472	0.8623	0.091	21
	-	-	-	-	0.050	(195)

Table 9.24: Problem: *Solar Flares*. Architecture: 8 hidden.

Neuron Model	NMSE	NMSEb	NMSE	NMSEb	$\hat{\beta}_i$	$\hat{n}$
	–	–	–	–	$\hat{\beta}_f$	$(\hat{p})$
P-neuron	0.6044 ± 0.0117	0.5612	1.1888 ± 0.0720	0.8722	0.500	51
	-	-	-	-	0.194	(327)
R-neuron	0.7803 ± 0.0161	0.7683	0.8518 ± 0.0455	0.8275	0.003	51
	-	-	-	-	0.031	(327)
H-neuron	0.7712 ± 0.0163	0.7466	0.9430 ± 0.0470	0.8632	0.091	21
	-	-	-	-	0.051	(291)

Table 9.25: Problem: *Solar Flares*. Architecture: 12 hidden.

Neuron Model	NMSE	NMSEb	NMSE	NMSEb	$\hat{\beta}_i$	$\hat{n}$
	–	–	–	–	$\hat{\beta}_f$	$(\hat{p})$
P-neuron	0.6487 ± 0.0168	0.5667	1.3694 ± 0.0844	0.9018	0.501	23
	-	-	-	-	0.248	(435)
R-neuron	0.7825 ± 0.0163	0.7670	0.8564 ± 0.0458	0.8303	0.003	23
	-	-	-	-	0.023	(435)
H-neuron	0.7783 ± 0.0162	0.7517	0.9591 ± 0.0481	0.8668	0.091	21
	-	-	-	-	0.052	(387)

Table 9.26: Problem: *Solar Flares*. Architecture: 16 hidden.

Neuron Model	NMSE –	NMSEb –	NMSE –	NMSEb –	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.1417 ± 0.0056 -	0.0872 -	0.1641 ± 0.0070 -	0.0985 -	0.505 0.373	2 (17)
R-neuron	0.3810 ± 0.0127 -	0.2341 -	0.4249 ± 0.0156 -	0.2678 -	0.350 0.296	2 (13)
H-neuron	0.0515 ± 0.0012 -	0.0372 -	0.0602 ± 0.0016 -	0.0441 -	0.332 0.336	2 (13)

Table 9.27: Problem: *Sinus-Cosinus*. Architecture: 4 hidden.

Neuron Model	NMSE –	NMSEb –	NMSE –	NMSEb –	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.0563 ± 0.0024 -	0.0311 -	0.0686 ± 0.0029 -	0.0378 -	0.500 0.338	2 (33)
R-neuron	0.3768 ± 0.0106 -	0.2265 -	0.4101 ± 0.0132 -	0.2453 -	0.351 0.272	2 (33)
H-neuron	0.0321 ± 0.0006 -	0.0255 -	0.0412 ± 0.0012 -	0.0310 -	0.331 0.330	2 (25)

Table 9.28: Problem: *Sinus-Cosinus*. Architecture: 8 hidden.

Neuron Model	NMSE –	NMSEb –	NMSE –	NMSEb –	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.0406 ± 0.0014 -	0.0288 -	0.0513 ± 0.0021 -	0.0345 -	0.499 0.362	5 (49)
R-neuron	0.3347 ± 0.0107 -	0.2175 -	0.3633 ± 0.0109 -	0.2345 -	0.351 0.266	5 (49)
H-neuron	0.0266 ± 0.0004 -	0.0232 -	0.0353 ± 0.0010 -	0.0291 -	0.331 0.332	2 (37)

Table 9.29: Problem: *Sinus-Cosinus*. Architecture: 12 hidden.

Neuron Model	NMSE –	NMSEb –	NMSE –	NMSEb –	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.0389 ± 0.0012 -	0.0286 -	0.0479 ± 0.0015 -	0.0337 -	0.500 0.377	2 (65)
R-neuron	0.3078 ± 0.0104 -	0.1816 -	0.3375 ± 0.0116 -	0.2015 -	0.350 0.256	2 (65)
H-neuron	0.0240 ± 0.0003 -	0.0219 -	0.0328 ± 0.0008 -	0.0286 -	0.331 0.333	2 (49)

Table 9.30: Problem: *Sinus-Cosinus*. Architecture: 16 hidden.

Neuron Model	**NMSE** –	**NMSEb** –	**NMSE** –	**NMSEb** –	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.0217 ± 0.0002 -	0.0196 -	0.0228 ± 0.0005 -	0.0200 -	0.505 0.324	2 (17)
R-neuron	0.1193 ± 0.0011 -	0.1045 -	0.1239 ± 0.0017 -	0.1074 -	0.350 0.408	2 (17)
H-neuron	0.0525 ± 0.0022 -	0.0280 -	0.0595 ± 0.0027 -	0.0311 -	0.348 0.286	2 (13)

Table 9.31: Problem: *SISO-Bench*. Architecture: 4 hidden.

Neuron Model	**NMSE** –	**NMSEb** –	**NMSE** –	**NMSEb** –	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.0200 ± 0.0002 -	0.0177 -	0.0216 ± 0.0005 -	0.0190 -	0.500 0.325	2 (33)
R-neuron	0.1092 ± 0.0009 -	0.0942 -	0.1158 ± 0.0016 -	0.0996 -	0.352 0.363	2 (33)
H-neuron	0.0126 ± 0.0004 -	0.0076 -	0.0145 ± 0.0006 -	0.0088 -	0.348 0.309	2 (25)

Table 9.32: Problem: *SISO-Bench*. Architecture: 8 hidden.

Neuron Model	**NMSE** –	**NMSEb** –	**NMSE** –	**NMSEb** –	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.0196 ± 0.0002 -	0.0174 -	0.0216 ± 0.0005 -	0.0183 -	0.498 0.360	5 (49)
R-neuron	0.1059 ± 0.0010 -	0.0894 -	0.1101 ± 0.0013 -	0.0920 -	0.351 0.335	5 (49)
H-neuron	0.0087 ± 0.0002 -	0.0066 -	0.0101 ± 0.0003 -	0.0074 -	0.348 0.325	2 (37)

Table 9.33: Problem: *SISO-Bench*. Architecture: 12 hidden.

Neuron Model	**NMSE** –	**NMSEb** –	**NMSE** –	**NMSEb** –	$\hat{\beta}_i$ $\hat{\beta}_f$	$\hat{n}$ $(\hat{p})$
P-neuron	0.0191 ± 0.0002 -	0.0167 -	0.0213 ± 0.0005 -	0.0175 -	0.500 0.362	2 (65)
R-neuron	0.1025 ± 0.0011 -	0.0832 -	0.1084 ± 0.0014 -	0.0906 -	0.351 0.336	2 (65)
H-neuron	0.0079 ± 0.0002 -	0.0063 -	0.0094 ± 0.0003 -	0.0072 -	0.347 0.335	2 (49)

Table 9.34: Problem: *SISO-Bench*. Architecture: 16 hidden.

9.6 Conclusions and outlook

Unfortunately, the results are not *directly* comparable with those existent in the literature since, to begin with, they correspond to other training algorithms (mostly back-propagation). Besides, the training regimes (number of partitions, partition sizes and precise composition of folds) are all different. Nonetheless, the results are much in the line or above average when confronted with other reported outcomes, using a more elaborate model selection process [Prechelt, 94], [Wilson and Martinez, 97], [Wilson and Martinez, 96]. This also points the BGA as a reliable network optimizer.

The proposed approach can be said to be satisfactory in a number of senses:

1. The *generalization* ability is significantly better on the average across all the data sets.
2. The *number of parameters*, for same numbers of hidden units, is lower.
3. The readability of the obtained solutions is enhanced, because it is amenable of a direct interpretation in terms of the original *problem* inputs, and the measure computed by the hidden neurons has been specifically designed *previous* to network training.
4. The possibility of adding *prior* domain knowledge (or at least part of it) can be used to explore the effect in performance.
5. There is no need for *encoding* schemes, or pre-treatments for missing value imputation.

On the other side of the balance, there can be no guarantee that the solutions obtained are to conform with point 1. above, since we may not know what are the "optimum" choices in order to design an H-neuron. In this sense, the experiments presented constitute worked examples of use, for which a complete domain knowledge was not available and some design decisions had to be made. Therefore, there might be situations for which a different decision on how to treat a given variable could have been made, because there may be more than one reasonable way to regard it. This is, however, one of the advantages of the approach. It is our conjecture that the incorporation of correct prior knowledge, together with a methodology that respects the nature of the data, and endows the networks with a clear semantics are on the basis of the superior results.

Chapter 10

Conclusions and Future Work

In this last chapter we set forth some reflections and concluding remarks about the work. We also summarize the contributions of the work as described in the dissertation and finally, we discuss some extensions and avenues for future research or development.

10.1 Conclusions

In many real-world problems knowledge comes in the form of heterogeneous information, which may be very different in nature (e.g., continuous or discrete, ordered or lacking an order, precise or vague) and may not be complete. In this Thesis, we have presented a general framework for the development of new neuron models to be used in artificial neural networks. These models are cast into the common conceptual view of computing a kind of *similarity measure* between inputs and weights, accounting for heterogeneity or other data peculiarities. Many of the standard neuron models fall within one of the derived instances.

In a general sense, we have laid the foundations of a theoretical framework for a class of heterogeneous neuron models, of which concrete instances and realizations have been set forth. These models are characterized by their built-in treatment of information coming from heterogeneous sources (perhaps missing) and make use of an explicit *similarity* measure between entities, specific for each data source.

We have applied several instances of the framework to specific problems, most of them real-world ones, with encouraging success. The possibility of adding prior knowledge in the design of the models and the elimination of the need of a preprocessing or encoding mechanism have been shown to be beneficial for the networks. On the one hand, because it permits to express part of the solution to a given task in terms of the solving method. On the other hand, it alleviates to a reasonable extent known problems as the curse of dimensionality or the difficulty of finding a structure in the data.

A driving motivation of the work has been general-purpose applicability. To achieve this, several universal measures for different data types are provided. Therefore, the problems worked out can be best seen as application examples rather than fully-solved problems. The results could be even better by devising problem-specific neuron models, making a careful selection of partial measures, and using the full available domain knowledge and expertise. We hence believe that our contributions can be effective in a broad spectrum of situations, and at the same time offering the possibility to be tailored to specific problems.

The work has some recognized limitations. The present methodology is possibly not enough to discern what is the most convenient similarity measure for a given problem or class of problems. The precise effect on performance of the different aggregation operators (e.g., is a weighted average a desirable measure in general?) and the effect of the proposed way of coping with missing values should be clarified. Currently, there is a design process only guided by the knowledge on the problem.

In this line of thought, it is certain that no specific neuron model is going to yield superior results for all the problems to which it is applied, but this characteristic affected also the standard models existent in the literature. Rather, what is offered is precisely to *widen the choice*, by letting the designer free to construct whatever measure is considered adequate and revise it in light of the obtained results which, in addition, can be more directly amenable to interpretation than in previous models.

Other decisions were deliberately taken from the outset. For instance, we have limited the scope of the work to feed-forward architectures, though nothing prevents the use of *recurrent* networks or heterogeneous *unsupervised* networks. Moreover, the design of other kinds of similarities, either taken from the literature or devised *ex profeso* could be considered adequate in the future, motivated by their use in a particular task.

It is sensible to expect that these more adapted and problem-specific models, not only can lead to artificial networks capable of more satisfactory performance, but provide the user with a more flexible and manageable tool for neural modeling.

10.2 Contributions

A summary of the dissertation and the main contributions follows:

- In chapter (3), the initial concepts about similarity-driven and heterogeneous artificial neural networks (ANN) are outlined. The strong points of the general idea and the relation with current models are discussed. The chapter serves also as an informal motivation for many of the ideas unfolded later on.

- In chapter (4), we construct a comprehensive framework where the notions of similarity and heterogeneity are characterized in the context of ANN. We proceed to the development of heterogeneous neuron models based on similarity relations or S-neurons, derived from a somewhat larger and abstract class of models (called H-neurons). We set forth explicit ways of designing heterogeneous similarity measures by the introduction of concepts like similarity aggregation operators and transformation functions. Missing information and semantic considerations are explicitly taken into account. We then express classical neuron models –as those used in RBF or MLP networks– as forms of computing a kind of similarity measure in real Euclidean space. An additional contribution is the development of new neuron models based on scalar product.

Similarity measures for sets different than the real continuum are identified. The notion of a heterogeneous space is then defined as a cartesian product of single spaces of mixed variables. As a consequence, heterogeneous similarity measures can be devised in this space, using specific forms for aggregation operators, leading to the generic concept of a S-neuron and to Heterogeneous Neural Networks (HNN). As a particularly useful instance of S-neurons of the *real kind* (that is, models for which the codomain is a subset of the reals), a collection of new models is derived, based on a distinguished generic measure grounded on Gower's classical similarity index [Gower, 71].

- Chapter (5) explores the theoretical approximation properties of some of the introduced families of neuron models. From a theoretical point of view, the property is important because it ensures that a satisfactory solution is always to exist, and has been already proved for standard neuron models. Since a common and general proof is not amenable to be obtained, we assuming precise decompositions about the specific similarity functions being computed by the network. We show that, under certain conditions, several types of feed-forward HNN share the universal approximation property.

- Chapter (6) is devoted to the investigation of Evolutionary Algorithms (EA) in the problem of training a HNN. Specifically, the standard genetic algorithm (GA) [Goldberg, 89] and the Breeder genetic algorithm (BGA) [Mühlenbein and Schlierkamp-Voosen, 93] are enhanced in a number of ways, to accept and manipulate heterogeneous variables in their chromosomic material. Proposals for its main parameters when used in the weight optimization task are made based on extensive experimentation on a difficult benchmark dataset and in previous investigations in classical testbed optimization problems.

- Experimental work is possibly the best means to assess the validity of the work. The neuron models derived from the approach have been tested empirically in a variety of situations and experimental conditions, and explored in basically three general kinds of problems:

  - In real-world problems -Chapter (7)- using data and some amount of expertise directly available to the author, and where there was a motivation to apply the ideas developed in the work.
  - In a specific industrial setting, the operation and control of WasteWater Treatment Plants (WWTP) [Lean and Hinrichsen, 94], of great industrial and social relevance -Chapter (8).
  - In well-known neural benchmarking databases [Prechelt, 94] displaying variable degrees of heterogeneity -Chapter (9). These experiments are carried out in a very controlled experimental setting.

In most cases, valuable models are found that can be said to be satisfactory in a number of senses: absence of coding schemes, generalization ability, number of model parameters, and readability of the obtained solutions.

10.3 Future research

The work described in this dissertation leaves a number of avenues for future research and improvement and thus can be continued in many interesting directions.

- In relation with studying the effect of specific neuron models, an immediate work is the analysis of the space spanned by the outputs of the hidden units. Given a hidden layer i of h_i units, let $S = [0, s_{max}]^{h_i}$, where s_{max} is the maximum similarity yielded by the hidden neurons. Some questions naturally arise. Given a trained network:

  1. Is S uniformly covered? If not, how are the vectors of similarities $\vec{s} \in S$ distributed for a given training or test set of input patterns?
  2. How does the distribution change along the training process?
  3. Is the distribution different for well-trained networks? (i.e., for networks yielding superior performance).
  4. How does this relate to the notions of neuron and network *sensitivity* developed in chapter (4)?

  The answer to these questions will help in studying the effect of setting different neuron models for a given problem and possibly for choosing among certain generic choices of aggregation operators, similarity transforming functions, etc.

- A long-term research goal is contemplated in the study of models where the output heterogeneous space, which is also the codomain of the similarity measure, is a different space than a subset of $\mathbb{R}$. For example, neuron models of the *fuzzy kind* imply the use of fuzzy arithmetic and would compute a fuzzy similarity measure, giving rise to heterogeneous fully fuzzy networks. This is of interest whenever there is a quantifiable uncertainty in the available samples of the target function. To this end, *fuzzy* similarity measures between fuzzy numbers should be devised. There are two basic ways of doing this:

  1. Given a similarity defined for reals (that is, for crisp values), compute it in fuzzy arithmetic by considering fuzzy arguments.
  2. Given a crisp similarity defined for fuzzy numbers, such as the one proposed in this work, extend it to yield also a fuzziness for the similarity judgement, based on the fuzziness of the arguments.

  Whatever the choice, the extension of additive aggregation operators should be done also by working in simple fuzzy arithmetic. For instance, the average of n fuzzy numbers is also a fuzzy number. Non-fuzzy similarities would be considered as crisp numbers. In this respect, notice that the overall function computed by the network is still a heterogeneous measure, in the sense that not all the inputs need to be fuzzy.

  Of special interest is the further extension to fuzzy quantities, in which case the similarity neural network would be yielding linguistic terms as outputs, very well suited for imprecise classification tasks. In this case, the source of imprecision is the absence of sharply defined criteria of class membership [Zadeh, 76], e.g., a person could be labelled as "tall". Here there is an interesting possibility in establishing links to other neurofuzzy classifiers [Nauck, Klawonn and Kruse, 97].

The overall motivation behind these extensions is that it is reasonable that a function with fuzzy arguments gives a fuzzy outcome. This may be of help in creating more flexible mappings and of great interest to the approximation of fuzzy functions.

Other models, of the *ordinal* or *nominal* kind, with the correspondingly defined similarity measures, are possible. The nominal case is particularly indicated for classical crisp classification tasks, for which the task of the classifier is to assign categorical symbols to given input patterns.

In the case of ordinal similarity measures, the value supplied is an ordered and discrete judgement -e.g. "four" children- which is not necessarily numerical -e.g., "January" in the set of months of the year, or "H" in the set of letters of the latin alphabet. These extensions collectively form a big area of development of potential practical interest.

- Another important area of potential improvement consists in grouping subsets of variables by a single similarity relation. In all the models set forth in this work, the computation of similarity for heterogeneous entities is constructed as a weighted combination of partial similarities over single variables, although any problem-specific partition over *subsets* of variables (and not singletons) is conceivable.

  Note that these could be regarded as *higher-order* models. For R-neurons (units of RBF networks) this involves the computation of weighted distance measures, where all the (quadratic) cross-products are included, and to the most general form for a RBF unit -see p. 133. For the P-neuron, the scalar product (containing no cross-product terms) can be generalized to a real quadratic form (an homogeneous polynomial of second degree with real coefficients) or even further to higher degrees, leading to so-called $\Sigma\Pi$ units [Durbin and Rumelhart, 89]. For the introduced heterogeneous neuron models or H-neurons, higher-order measures have a nice conceptual interpretation as overall measures defined over partial ones on subsets of variables. An additional important consideration is that this scheme need not involve an increase in free parameters.

  Criteria for grouping subsets of variables in order to define a single similarity can be either *syntactic* or *semantic*. In the former case, the grouping could based on the data type: many heterogeneous distance measures have worked out this idea, which is possibly too restrictive. An alternative is to consider semantic criteria, that is, to use domain knowledge. For instance, one might be interested in defining a single similarity measure among a collection of variables that are known (or believed) to be strongly related. This process could be seen as the creation of macrovariables or *features*. Examples abound:

  - It is common to use delayed information in the inputs of a neural network, by means of a moving window or delay line. These networks are known as time-delay neural networks [Hertz, Krogh and Palmer, 91]. Consider the following three variables as part of the inputs of such a network:

$$\begin{gathered} x(t-1), y(t-1), z(t-1) \\ x(t-2), y(t-2), z(t-2) \end{gathered}$$

Looking at them in a horizontal way, we know that these variables (in two groups of three) share a common underlying dimension, which is equal for each group: time. Looking at them from the vertical point of view (three groups of two), we know that these different inputs are in fact the *same* measured variable, though at different times. This extra information can be supplied to the network in the form of partial measures defined on specific subsets of variables.

- In many practical situations —for example in data sets coming from street polls, or questionnaires collecting personal data— if the answer to question, say, number three is "a)", then the fourth question must be skipped (incidentally, this is a source of missing information). These two variables should not be considered as independent, because there is a relation that is known and, hence, modelable. A partial measure could take into account both variables at once and output a single similarity measurement.

- The possibility of incorporating prior knowledge into the design of the neuron model entails with it the counterpart: the *extraction* of knowledge out of a trained network. This is a clear avenue for new research. Some work has already been done for classical RBF networks, possibly the more amenable a priori to a clear interpretation, due to the local nature of the hidden units [Andrews and Geva, 96]. In our case, this interpretation accompanied by a rule extraction process is made much easier by the characteristics of the proposed neuron models.

- The consideration, as already mentioned, of more abstract input spaces. In this sense, a partially ordered subspace is an immediate extension. It is not uncommon, in practical cases, the existence of incomparable elements. In ordinal spaces, this entails the existence of *partial* orders. A distinguished situation is a *lattice*: a set where, for every pair of elements, there exist a supremum and an infimum (e.g., the totality of subsets of a set is a lattice by the partial ordering given by set inclusion). Other data types could include trees, graphs or strings in an alphabet, for which similarity relations can be found in the literature -e.g., [Honavar, 92].

- The work done concerning the universal approximation property should be considered as a preliminary study on especially interesting or representative classes of models. The generalization to more abstract classes and the integration of heterogeneous information is probably a thesis in itself. Specifically, the work on nominal information could need the definition of new topologies in these spaces. The fulfilment of the property in this and other cases, as for linguistic variables, remains an open question.

- Finally, the application of this research to real-world problems -perhaps revising some previous results in light of new advances- is undoubtedly an avenue of continuous further work. In particular, the contribution to the operation and control of wastewater treatment plants, although it has already produced some very valuable results, is currently being subject of new work.