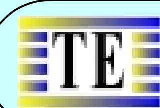
# DESIGN OF ARTIFICIAL NEURAL NETWORKS

# USING EVOLUTIONARY COMPUTATION


Francisco Sandoval

Departamento de Tecnología Electrónica
Universidad de Málaga, Espagne

# CONTENTS

**1. Introduction**

2. Genetic algorithms

3. Designing ANN with evolutionary computation

    3.1. Connection weights in a defined architecture

    3.2. Evolution of architectures

    3.3. Node transfer functions

    3.4. Evolution of architecture and connection weights

    3.5. Evolution of learning rules

    3.6. ANN input data selection

4. Applications

5. Conclusions

# INTRODUCTION-I

- ANNs are a biologically-inspired attractive paradigm of computation for many applications: pattern recognition, system identification, cognitive modeling, etc.

- Properties of ANNs are:
  - Capability of “learning” and “self-organizing” to carry out a given task: ill-defined and input/output mapping.
  - Potential for massively parallel computation.
  - Robustness in the presence of noise.
  - Resilience to the failure of components.

- Practical applications require the choice of a suitable network topology, the learning rule and the processing function computed by individual units

# INTRODUCTION-II

- ANNs are often hard to design because:
  - ➪ Many basic principles governing information processing are difficult to understand.
  - ➪ Complex interactions among networks unit make engineering techniques inapplicable.
  - ➪ The scaling problem: some solutions are not good if the complexity of the problem increases.

- In addition to experience and trial and error methods to design an ANN, how could we appeal to more efficient automated procedures? Using EVOLUTIONARY COMPUTATION.

- Genetic Algorithms, Evolution Strategies and Evolutionary Programming: population-based stochastic search algorithms, inspired in principles of natural evolution

## CONTENTS

1. Introduction

**2. Genetic Algorithms**

3. Designing ANN with evolutionary computation

    3.1. Connection weights in a defined architecture

    3.2. Evolution of architectures

    3.3. Node transfer functions

    3.4. Evolution of architecture and connection weights

    3.5. Evolution of learning rules

    3.6. ANN input data selection

4. Applications

5. Conclusions

# SEARCH METHODS

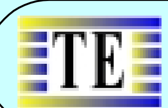
- Based on calculus
  - Direct
  - Indirect
- Random
  - Stochastic scalator (Simulated Annealing)
  - Evolutionary Algorithms
    - **Genetic Algorithms**
      - Parallel
      - Sequential
    - Evolution Strategies
    - Evolutionary Programming
  - Tabu
- Enumeratives
  - Dynamic Programming

# GENETIC ALGORITHMS-I

- GAs are based on the Darwin's evolution and natural selection concepts.
- GAs are systematic methods to resolve searching and optimization problems, applying analog methods than the biological evolution: selection based on population, reproduction and mutation.
- An optimization problem consist of:
  - Space of search $\Sigma$ with M symbols and dimension $M^N$. Any point in this space is a vector of N components of the M symbols.
  - A fitness or cost function $F: \Sigma \to \boldsymbol{R}$, to be optimized.

- General framework of a GA.

1. Generate the initial population P(0) at random, and set i=0;
2. REPEAT
   a) Evaluate each individual in the population;
   b) Select parents from P(i) base on their fitness in P(i);
   c) Applied search operators to parents and produce offspring which form P(i+1);
   d) i=i+1;
3. UNTIL termination criterion is satisfied.



Dpto. Tecnología Electrónica
Universidad de Málaga

# GENETIC ALGORITHMS-II

- The individuals of the population are chromosomes or genotypes: arbitrary data structure that encodes a certain solution to the problem. Its meaning, in relation with the problem at hand, is a phenotype. Different genotypes may encodes for the same solution or phenotypes, but the reverse does not hold.

- Each component of the chromosome represents a gene, and the molecular base of the gene is the ADN.

- The gene is said to be in several states, called alleles (feature values), and its position in the string, loci (string positions):

$$A=a_1a_2a_3a_4a_5a_6a_7$$
$$A= 0\ 1\ 1\ 1\ \ 0\ 0\ 0$$

# GENETIC ALGORITHMS-III

Advantages:

- It is enough with a small, and even null, knowledge a priori of the characteristics of the problem  to be solved.
- It is not necessary the direct control of the environment.
- Neither is necessary to search in the whole space of possible solutions.
- They use global information
- The characteristics acquired by the parents during the evolution are inherited by the offspring.
- Robustness. GAs are capable of finding a good solution

Limitations:

- The epistasis problem: iterations not wanted among the genes, in which a gene suppresses the expression of another. To solve it, elitist selection.
- Premature convergence. Selection to get o good trade-off between exploration and exploitation.
- Long computation times. Parallelization of the algorithm or distribute the population
- Fall in local minima

## EXAMPLE:

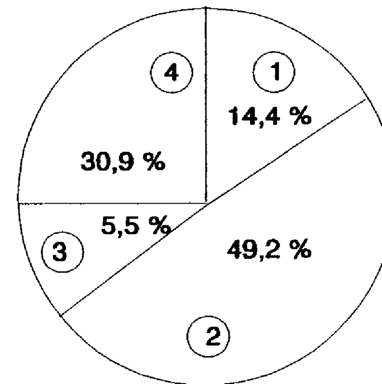
Evaluation and selection:

- Evaluation is performed through the fitness function.
- Selection probability

$$p_s(i) = \frac{f(i)}{\sum_{i=1}^{N} f(i)}$$

- Fitness proportionate selection with the roulette wheel method

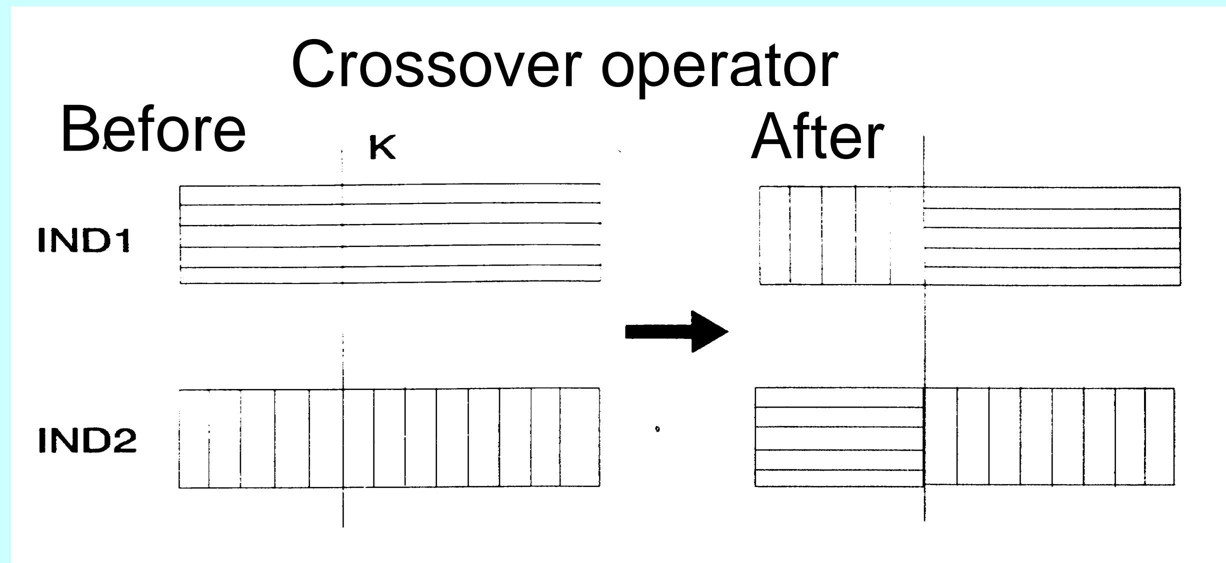
| Nº | CADENA | FITNESS | % DEL TOTAL |
|---|---|---|---|
| 1 | 0 1 1 0 1 | 169 | 14,4 |
| 2 | 1 1 0 0 0 | 576 | 49,2 |
| 3 | 0 1 0 0 0 | 64 | 5,5 |
| 4 | 1 0 0 1 1 | 361 | 30,9 |
| TOTAL | | 1170 | 100,0 |



## EXAMPLE:

Reproduction: crossover and mutation operators:

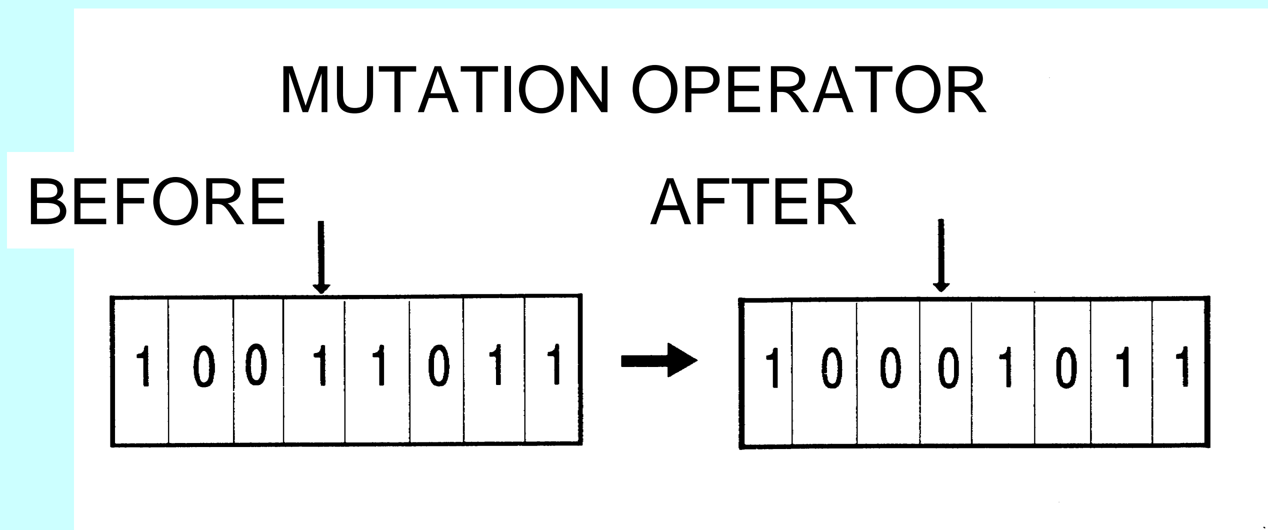
➪ In crossover o recombination first couples are randomly formed with all parents individuals, with a crossover rate, $p_c$. Then, the two bit strings are cut at the same random position and the second halves are swapped between the two individuals, each containing characters from both parents.



## EXAMPLE:

➪ Mutation operator: it simulate transcription error that can happen with very low probability, $p_m$, when a chromosome is duplicated.

➪ Mutation contributes with new information, essential for the evolution, that is, it introduces diversity in the population.

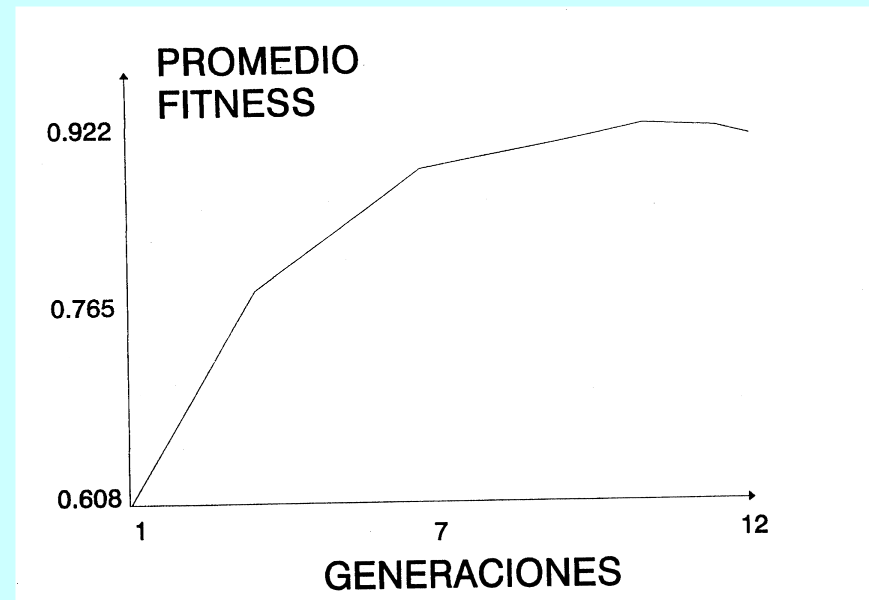


## EXAMPLE:

➪ The performance of a GA depends on: dimension of the population, application frequency of the selection, crossover and mutation operators, and how these operators are used.

➪ Optimization of a polynomial function:

. Binary encoding
. Population= 100 individuals
. Length of each individual = 30
. Study interval:$[0,2^{30}]$
. $p_c = 0.5$
. $p_m = 0.03$

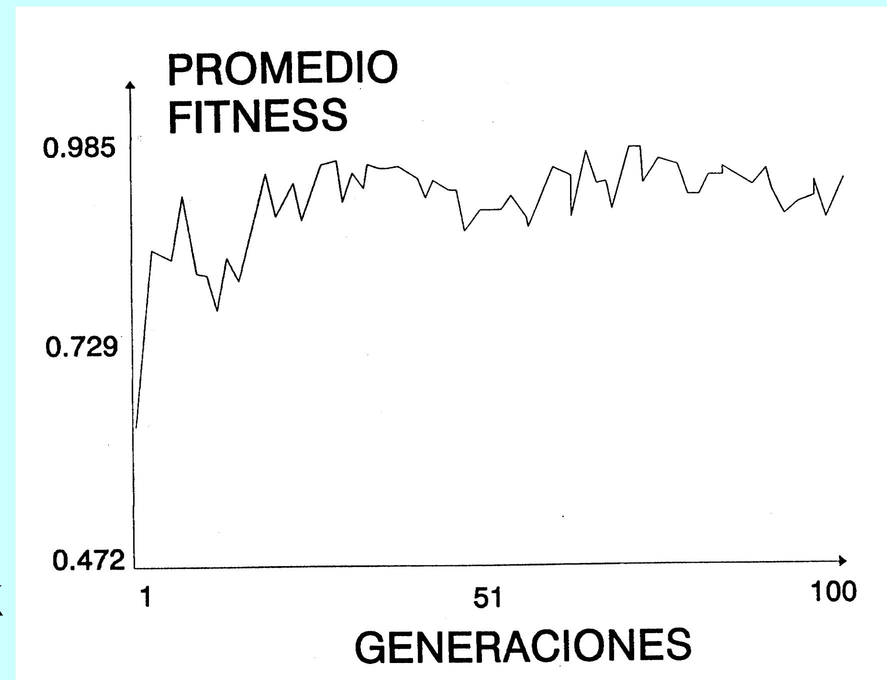



After 7 generations, the average fitness has been stabilized.

# EXAMPLE:

➪ Optimization of a polynomial function:

. Binary encoding
. Population= 10 individuals
. Length of each individual = 10
. Study interval:$[0,2^{10}]$
. $p_c = 0.5$
. $p_m = 0.03$

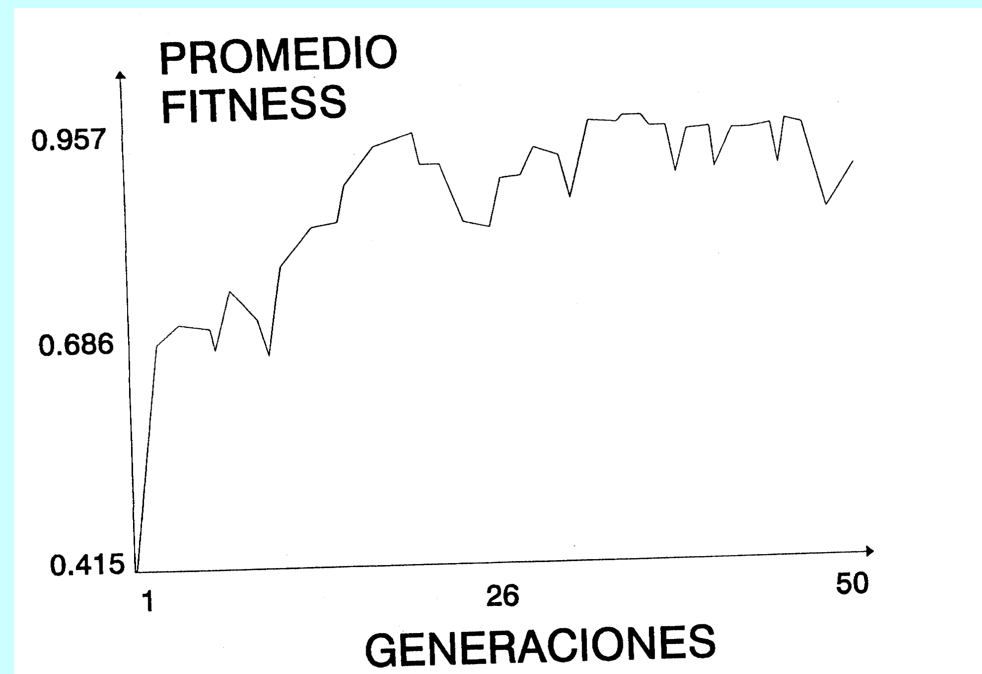



Small population and short length of individuals makes the effect of mutation operator very active and the GA does not work properly.

## EXAMPLE:

➪ Optimization of a polynomial function:

. Binary encoding
. Population= 10 individuals
. Length of each individual = 10
. Study interval:$[0,2^{10}]$
. $p_c = 0.3$
. $p_m = 0.03$




The GA behaves incorrectly due to low $p_c = 0.3$.

# THE SCHEMA THEOREM

- Binary alphabet $\Omega$={0,1}; extended alphabet $\Omega$'= {0,1,*}
- A schema E represents all strings, which match it on all positions other than *. The schema (*101) matches two strings: {(0101),(1101)}
- Number of possible schematas=(base+metasymbol)$^m$, with *m* the length of an individual.
- Every schema E matches $2^r$ strings, with *r* de number of metasymbols. The schema (*1*0) matches four strings
- Each string of the length m is matched by $2^m$ schemata.
- Order O(E)= Number of fixed positions in the template string.

O(10**11)=4; O(1****)=1

- Length L(E)= Distance between the first and last fixed positions in its template string. It is a measure of compactness information.

L(10**11)=6-1=5; L(1*****)=1-1=0

- Fitness at time t, F(E,t)= Average fitness of all strings in the population matched by the schema E.

## MEANING OF OPERATORS

Selection: $$S(E,t+1) = S(E,t) \cdot \frac{\bar{f}(E)}{\bar{f}}$$

Crossover: $$p_s = 1 - p_d = 1 - \frac{p_c \cdot L(E)}{m-1} \qquad p_s \geq 1 - \frac{p_c \cdot L(E)}{m-1}$$

Mutation: $$p_s = (1 - p_m)^{O(E)}; p_m << 1 \Rightarrow p_s \approx 1 - O(E) \cdot p_m$$

Combined effects:

$$S(E,t+1) \geq S(E,t) \cdot \frac{\bar{f}(E)}{\bar{f}}(1 - \frac{p_c \cdot L(E)}{m-1} - O(E) \cdot p_m)$$

Building Block hypothesis: A genetic algorithm seeks near optimal performance through the juxtaposition of short, low-order, high performance schemata.

## CONTENTS

1. Introduction

2. Genetic algorithms

**3. Designing ANN with evolutionary computation**

- 3.1. Connection weights in a defined architecture
- 3.2. Evolution of architectures
- 3.3. Node transfer functions
- 3.4. Evolution of architecture and connection weights
- 3.5. Evolution of learning rules
- 3.6. ANN input data selection

4. Applications

5. Conclusions

# DESIGN STEPS OF AN ANN

1st DATA PREPROCESSING

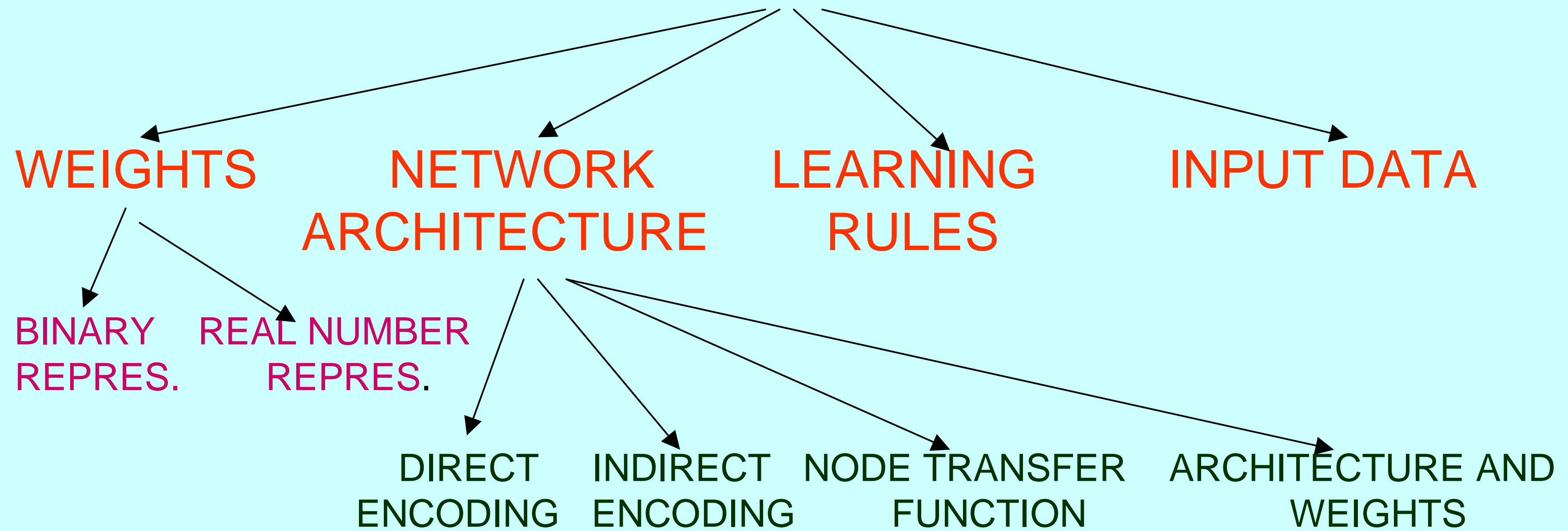
2nd CONFIGURATION

- SELECT INPUTS AND OUTPUTS
- SELECT ENCODING
- TOPOLOGY
- CONNECTIVITY

3rd TRAINING

*Dpto. Tecnología Electrónica*
*Universidad de Málaga*

# EVOLUTIONARY DESIGN OF ANNs



# CONTENTS

1. Introduction

2. Genetic algorithms

3. Designing ANN with evolutionary computation

    **3.1. Connection weights in a defined architecture**

    3.2. Evolution of architectures

    3.3. Node transfer functions

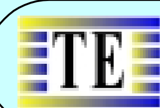
    3.4. Evolution of architecture and connection weights

    3.5. Evolution of learning rules

    3.6. ANN input data selection

4. Applications

5. Conclusions



# CONNECTION WEIGHTS IN A DEFINED ARCHITECTURE

➪ To find the suitable weights to perform the task at hand.
➪ Classical learning algorithms (backpropagation, conjugate gradient, ...) get trapped in local minimun and never find it if the error function is multimodal and/or non-differenciable.
➪ We may formulate the training process as the evolution of the connecting weights for a determined architecture and learning task.

➪ Two steps: weights representation and evolutionary process.

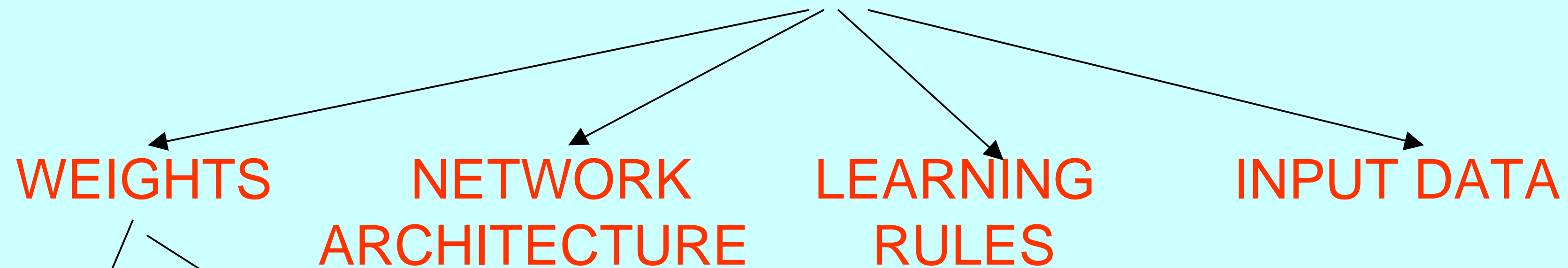
1) Decode each individual into a set of connection weights and construct a corresponding ANN with the weights.

2) Evaluate each ANN by computing its total mean square error between actual and expected outputs. The fitness is determined by the error.

3) Select parents for reproduction based on their fitness.

4) Apply search operators to generate offspring.

# EVOLUTIONARY DESIGN OF ANNs




BINARY REPRES. | REAL NUMBER REPRES.

DIRECT ENCODING | INDIRECT ENCODING | NODE TRANSFER FUNCTION | ARCHITECTURE AND WEIGHTS

# CONNECTION WEIGHTS IN A DEFINED ARCHITECTURE

## Binary representation:

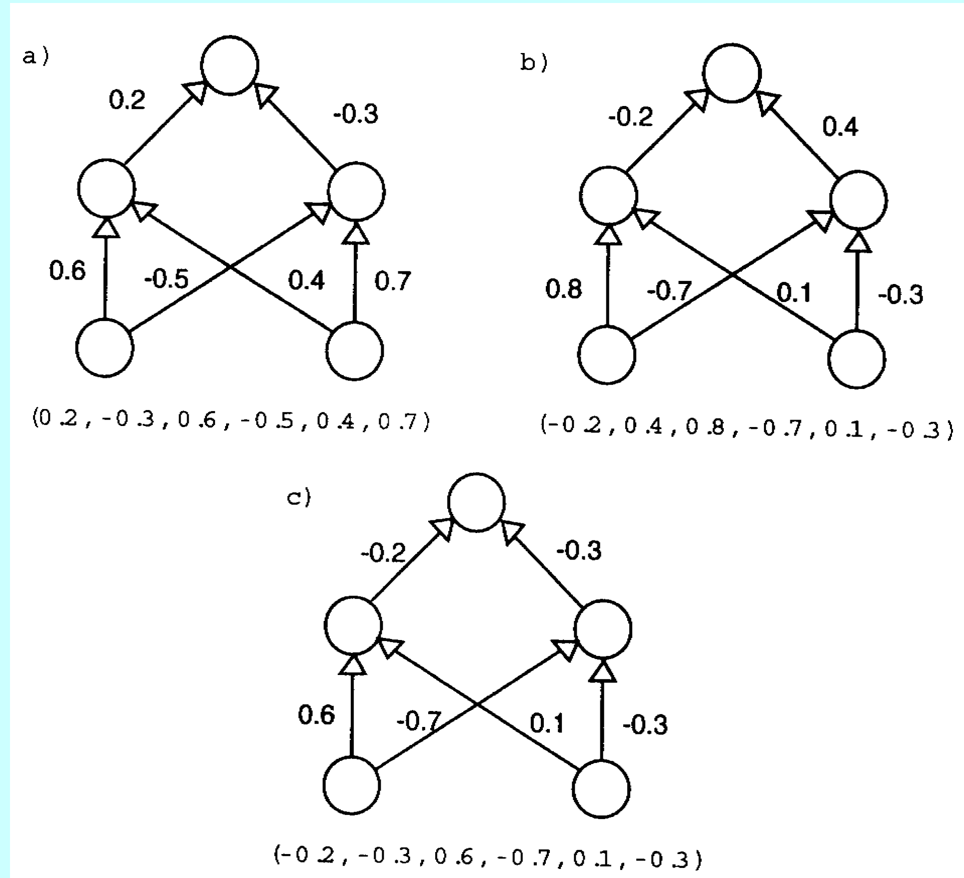
- The chromosome representing the net is a list of weights in a predefined order, each weight represented by a binary string to be decoded into real values between, for instant, -1 and +1.
- It facilitates the hardware implementation of ANNs
- In binary representation several encoding methods can be used: Gray, BCD, exponential, etc.

- Drawbacks: poor scalability, i.e., if the number of connections increases, the length of the string increases severely, degrading the features of the GA.
- If higher precision is needed in the weight value, more bits have to be added to the binary representation

# CONNECTION WEIGHTS IN A DEFINED ARCHITECTURE

## Real number representation:

- Each chromosome is a real vector, being each component a weight.
- Search space becomes very large.
- It has many advantages, but requires specialized genetic operators.
- Evolutionary Programming and Evolution Strategies are techniques well-suited in this kind of representation.

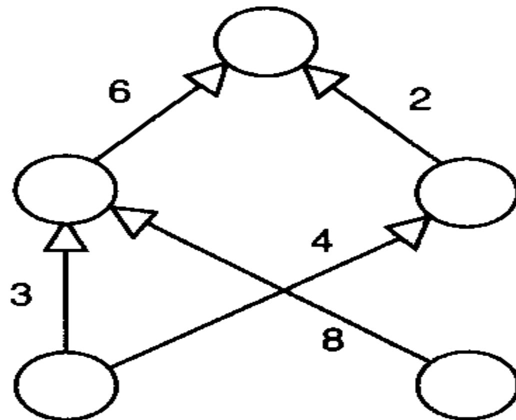


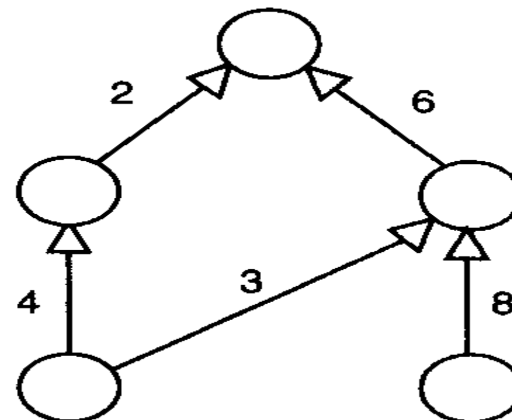

**Crossover operator**

# CONNECTION WEIGHTS IN A DEFINED ARCHITECTURE

## Permutation problem:

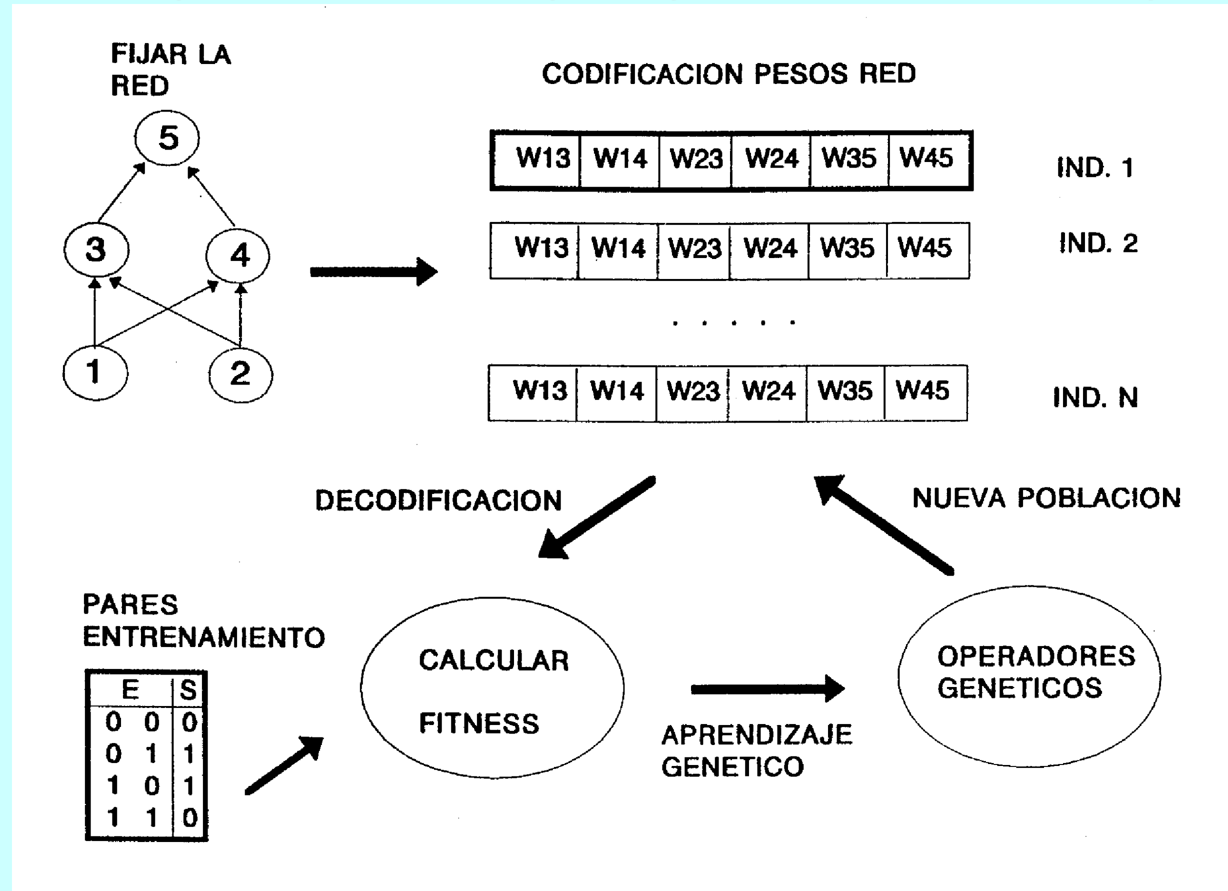
- Networks topologically equivalent but different encoding. Any permutation of the hidden nodes produce equivalent network functions and fitness measurements.
- Limit the crossover standard operator. Evolutionary programming does not use crossover



# CONNECTION WEIGHTS IN A DEFINED ARCHITECTURE

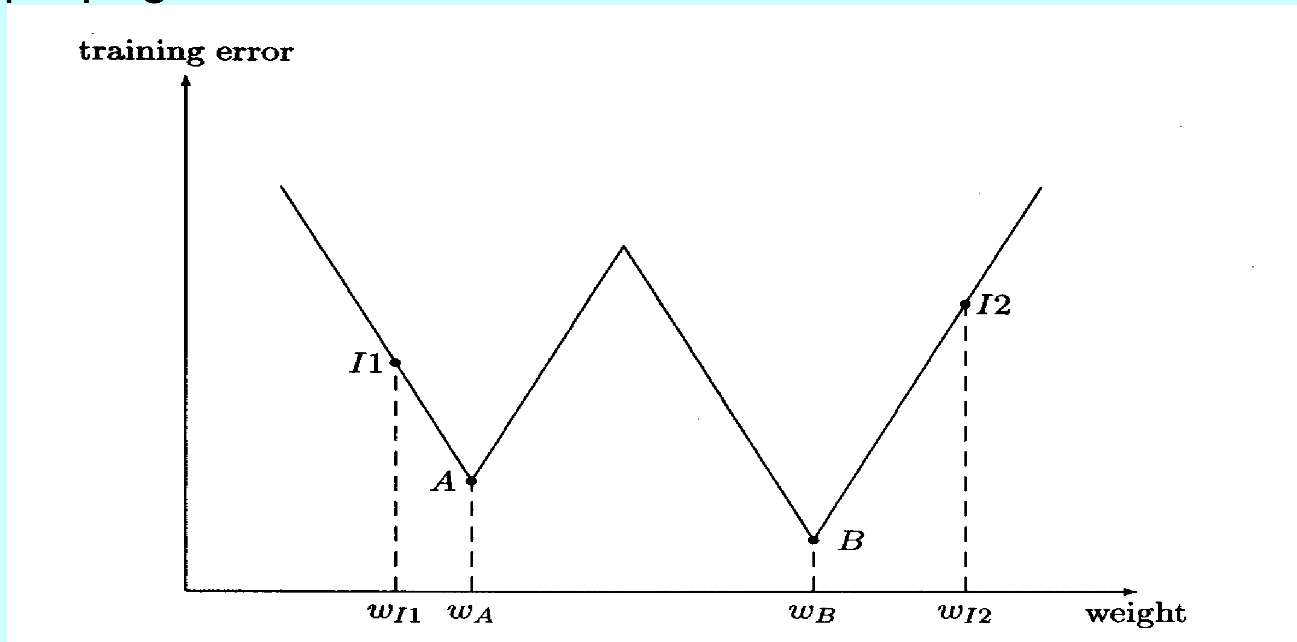
## Flow diagram for designing connection weights:



# CONNECTION WEIGHTS IN A DEFINED ARCHITECTURE

## Hybrid training:

- GAs are not efficient in fine-tuned local search, although they are good at global search.
- Finding initial weights with GAs and then local search with backpropagation.



# CONTENTS

1. Introduction
2. Genetic algorithms
3. Designing ANN with evolutionary computation
    3.1. Connection weights in a defined architecture
    **3.2. Evolution of architectures**
    3.3. Node transfer functions
    3.4. Evolution of architecture and connection weights
    3.5. Evolution of learning rules
    3.6. ANN input data selection
4. Applications
5. Conclusions

# EVOLUTION OF ARCHITECTURES

➪ Selection of architecture is a very important task: connectivity and transfer function of each node:

- Few connections and linear nodes $\rightarrow$ limited capability
- Many connections and non-linear nodes $\rightarrow$ overfit noise in the training data and poor generalization.

➪ Usually, architecture design is performed in an unsystematic manner by guesswork and trial and error.

➪ Methods such as constructive-destructive algorithms are susceptible to becoming trapped at local minima and depends on the initial topology.

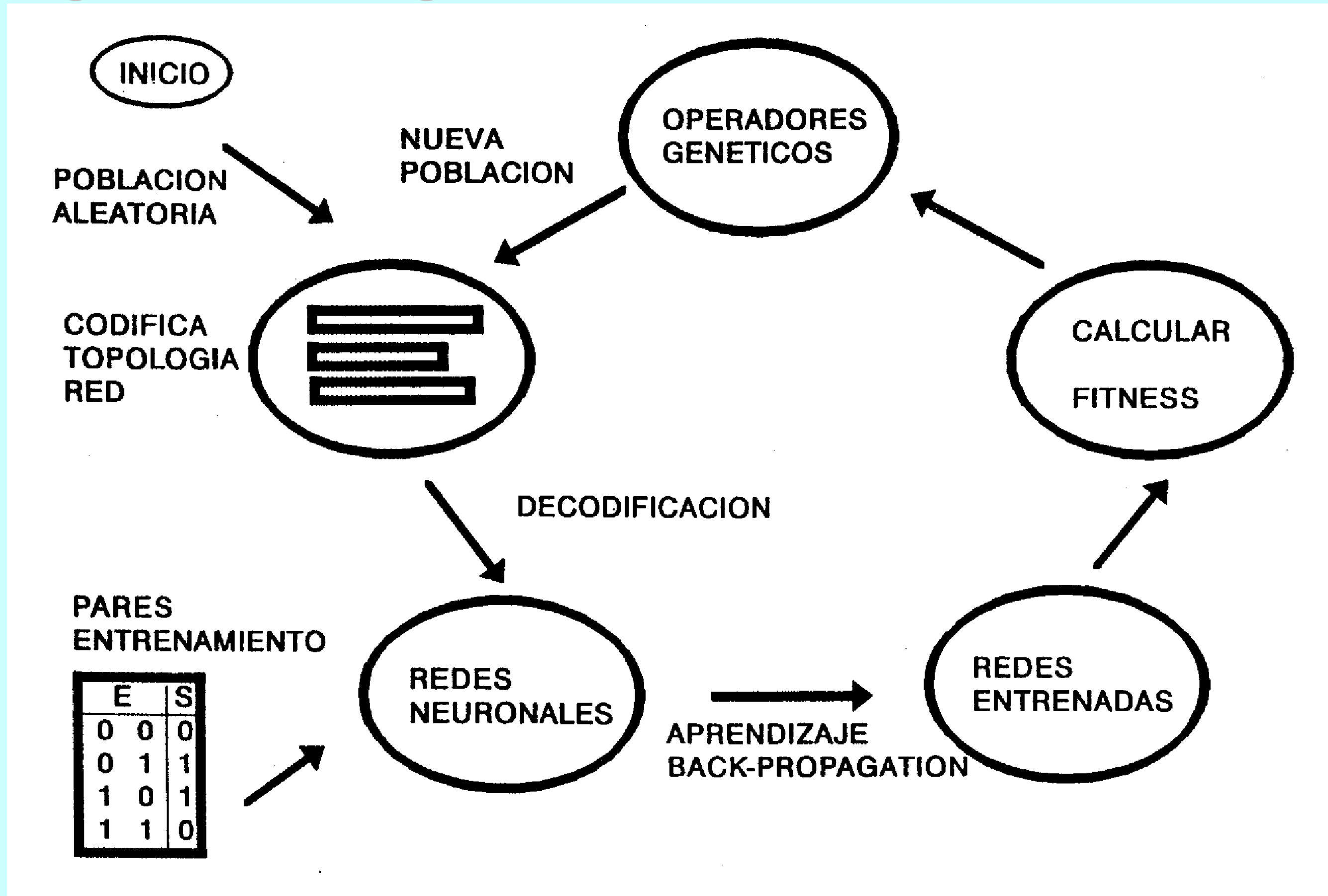
➪ Design the architecture of an ANN is a search problem in the architecture space where each point represents an architecture. So GAs are a promising technique.

# EVOLUTION OF ARCHITECTURES

- Why GAs are good candidates:
  - The surface of search is infinitely large: number of possible nodes and connections is unbounded.
  - The surface is non-differentiable: changes in the number of nodes or connections are discrete and can have a discontinuous effect on GAs performance.
  - The surface is complex and noisy: mapping from an architecture to its performance is indirect, epistatic and dependant on the evolution method.
  - The surface is deceptive: similar architectures may have quite different performance.
  - The surface is multimodal: different architectures may have similar performance.

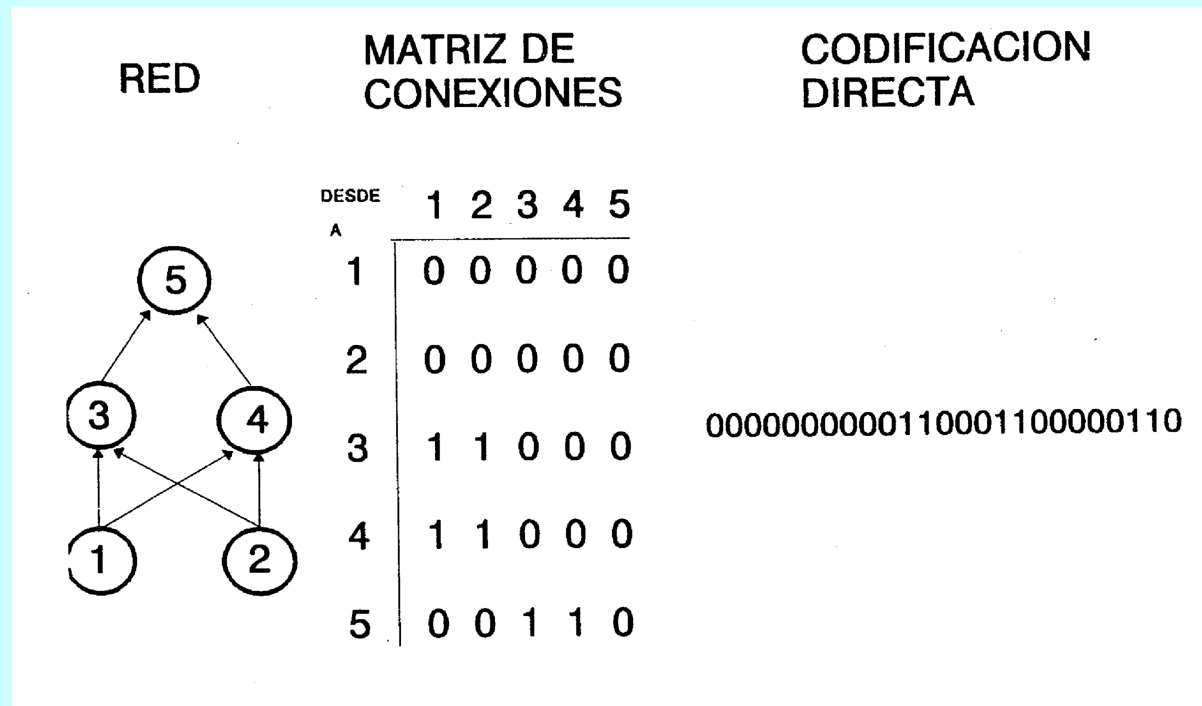
# EVOLUTION OF ARCHITECTURES

Flow diagram to design the architecture of an ANN:



# EVOLUTION OF ARCHITECTURES

- Direct encoding: all aspects of a network architecture (connections and nodes) are encoded into the chromosome.
- The X-OR problem of two bits: connectivity or adjacency matrix

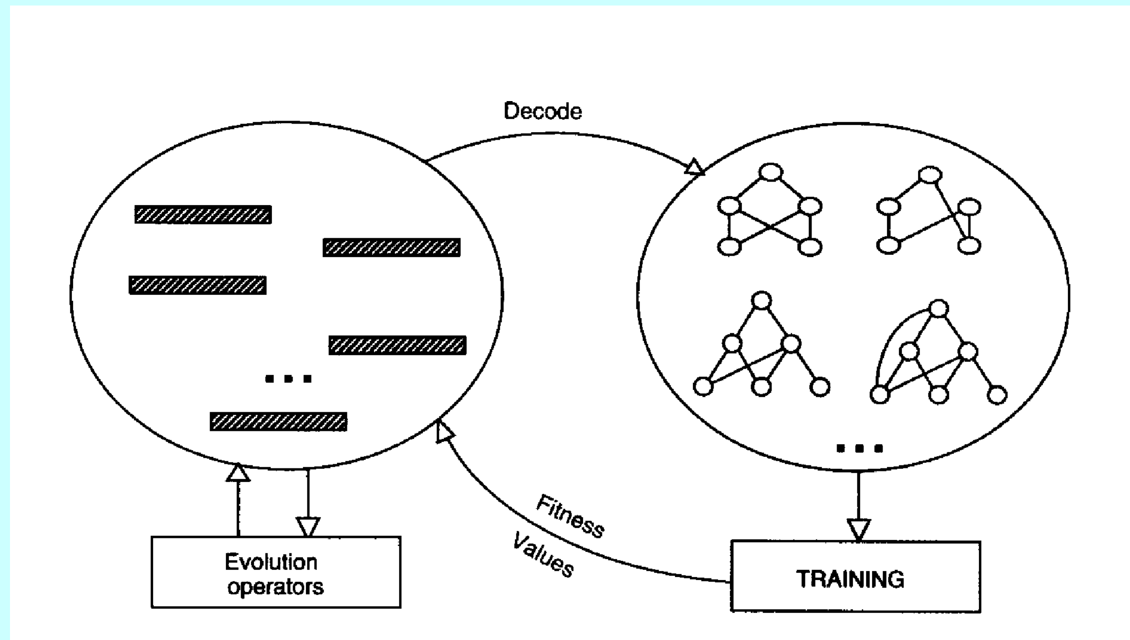



Matrix lower triangular: feedforward
Matrix triangular: feedback

Dpto. Tecnología Electrónica
Universidad de Málaga

# EVOLUTION OF ARCHITECTURES

- Direct encoding: relationships between genotypes and phenotype




- Drawbacks:scaling problem. Very long chromosomes
- The permutation problem
- Useful for small architectures

# EVOLUTION OF ARCHITECTURES

- Indirect encoding: only the most important parameters of an architecture are encoded (hidden layers and hidden nodes in each layer). The details of each connection is either predefined or specified by deterministic developmental rules.

- Last tendency in indirect encoding is to use grammatical encoding: technique for developing or growing ANN, rather than looking for a complete network description at the individual level.

- Grammatical encoding presents better scalability, reduces the effect of the permutation problem, and allows to find building blocks of general utility and of reusing developmental rules for general classes of problems.

# EVOLUTION OF ARCHITECTURES

➪ Developmental rule:

- Grammar: one or more productions
- Production: rewriting rule that associates a left-hand side (head) to a right-hand side (body). The two side are separated by the metasymbol $\rightarrow$.
- Terminal elements: either 1 (connection) or 0 (non-connection)
- The connectivity pattern of the architecture in the form of a matrix is constructed from a basis, i.e., a single element matrix, by repetitively applying suitable developmental rules to non-terminal elements till the matrix contains only terminal elements.

# EVOLUTION OF ARCHITECTURES

➪ Developmental rule:

$$S \longrightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

$$A \longrightarrow \begin{pmatrix} a & a \\ a & a \end{pmatrix} \quad B \longrightarrow \begin{pmatrix} i & i \\ i & a \end{pmatrix} \quad C \longrightarrow \begin{pmatrix} i & a \\ a & c \end{pmatrix} \quad D \longrightarrow \begin{pmatrix} a & e \\ a & e \end{pmatrix}$$

$$a \longrightarrow \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \quad c \longrightarrow \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \quad e \longrightarrow \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad i \longrightarrow \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

S is the initial element

$$S$$

(a)

$$\begin{matrix} A & B \\ C & D \end{matrix}$$

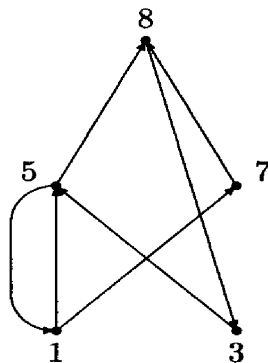
(b)

$$\begin{matrix} a & a & i & i \\ a & a & i & a \\ i & a & a & e \\ a & c & a & e \end{matrix}$$

(c)

$$\begin{matrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

(d)



(e)

Development of an architecture
from the initial element.

TE
Dpto. Tecnología Electrónica
Universidad de Málaga

# EVOLUTION OF ARCHITECTURES

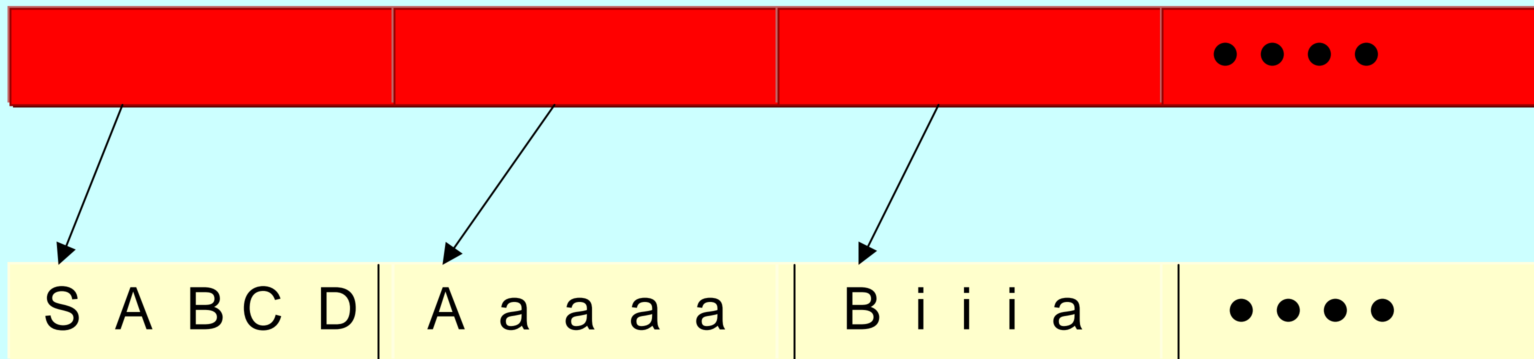
➪ Developmental rule:

$$S \longrightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

$$A \longrightarrow \begin{pmatrix} a & a \\ a & a \end{pmatrix} \quad B \longrightarrow \begin{pmatrix} i & i \\ i & a \end{pmatrix} \quad C \longrightarrow \begin{pmatrix} i & a \\ a & c \end{pmatrix} \quad D \longrightarrow \begin{pmatrix} a & e \\ a & e \end{pmatrix} \quad \cdots$$

$$a \longrightarrow \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \quad c \longrightarrow \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \quad e \longrightarrow \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad i \longrightarrow \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \cdots$$

Chromosome encoding




There are some controversial with this approach

## CONTENTS


1. Introduction
2. Genetic algorithms
3. Designing ANN with evolutionary computation
    - 3.1. Connection weights in a defined architecture
    - 3.2. Evolution of architectures
    - **3.3. Node transfer functions**
    - 3.4. Evolution of architecture and connection weights
    - 3.5. Evolution of learning rules
    - 3.6. ANN input data selection
4. Applications
5. Conclusions

# NODE TRANSFER FUNCTIONS

➪ The transfer node in the architecture is supposed to be fixed by human expert, and often the same for all the nodes, at least for all the nodes in the same layer.

➪ But node transfer function may evolve with GAs. Different approaches:

- For each individual in the initial population, 80% nodes use sigmoid transfer function and 20% Gaussian transfer function. The evolution was used to decide the optimal mixture between these two transfer function. The transfer function themselves were not evolvable.
- With sigmoid and Gaussian transfer nodes, the total number is not fixed, and is allowed growth and shrinking of the ANN by adding or deleting nodes.

# CONTENTS

1. Introduction
2. Genetic algorithms
3. Designing ANN with evolutionary computation
    3.1. Connection weights in a defined architecture
    3.2. Evolution of architectures
    3.3. Node transfer functions
    3.4. Evolution of architecture and connection weights
    3.5. Evolution of learning rules
    3.6. ANN input data selection
4. Applications
5. Conclusions

# ARCHITECTURE AND CONNECTION WEIGHTS

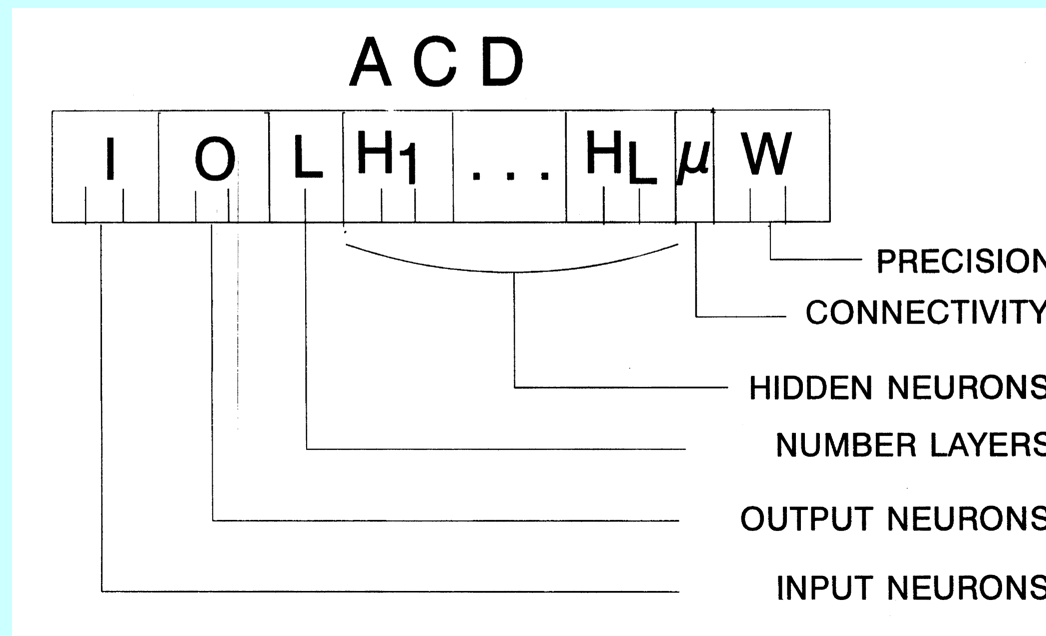
➪ Usually connection weights have to be learned after a near optimal architecture is found. Evolution of architectures without connections weights brings a mayor problem: noisy fitness evaluation.

➪ Phenotype's (an ANN with a full set of weights) fitness is used to approximate its genotype's (an ANN without any information on weights) fitness. Two mayor sources of noise:

- Different random initialization of weights may produce different training results: the same genotype could have different fitness.
- Different training algorithms may produce different training results even from the same set of initial weights.

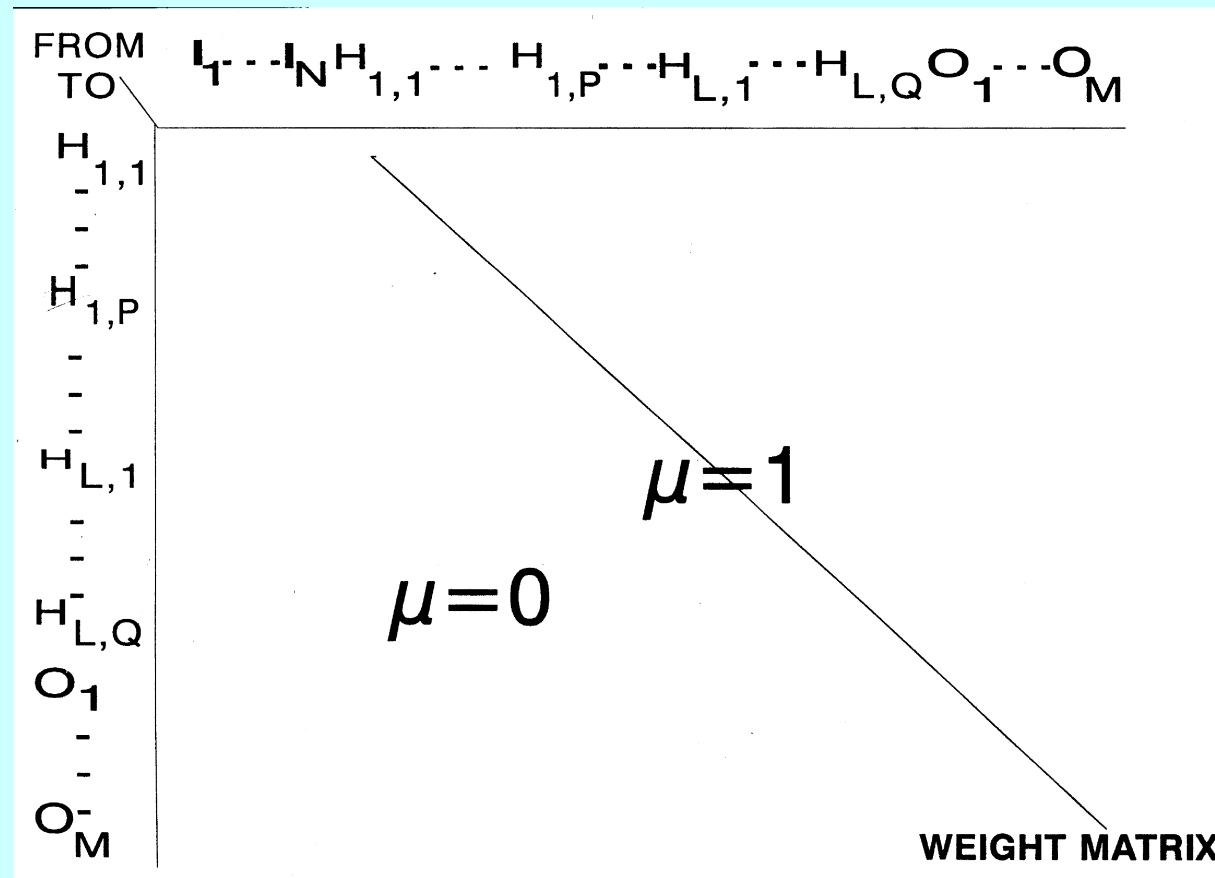
# ARCHITECTURE AND CONNECTION WEIGHTS

- Evolve architecture and weights simultaneously.
- Each individual with two chromosomes: one for the area of connectivity definition (ACD) and other for the area of weight definition (AWD).



# ARCHITECTURE AND CONNECTION WEIGHTS

- Area of weight definition:

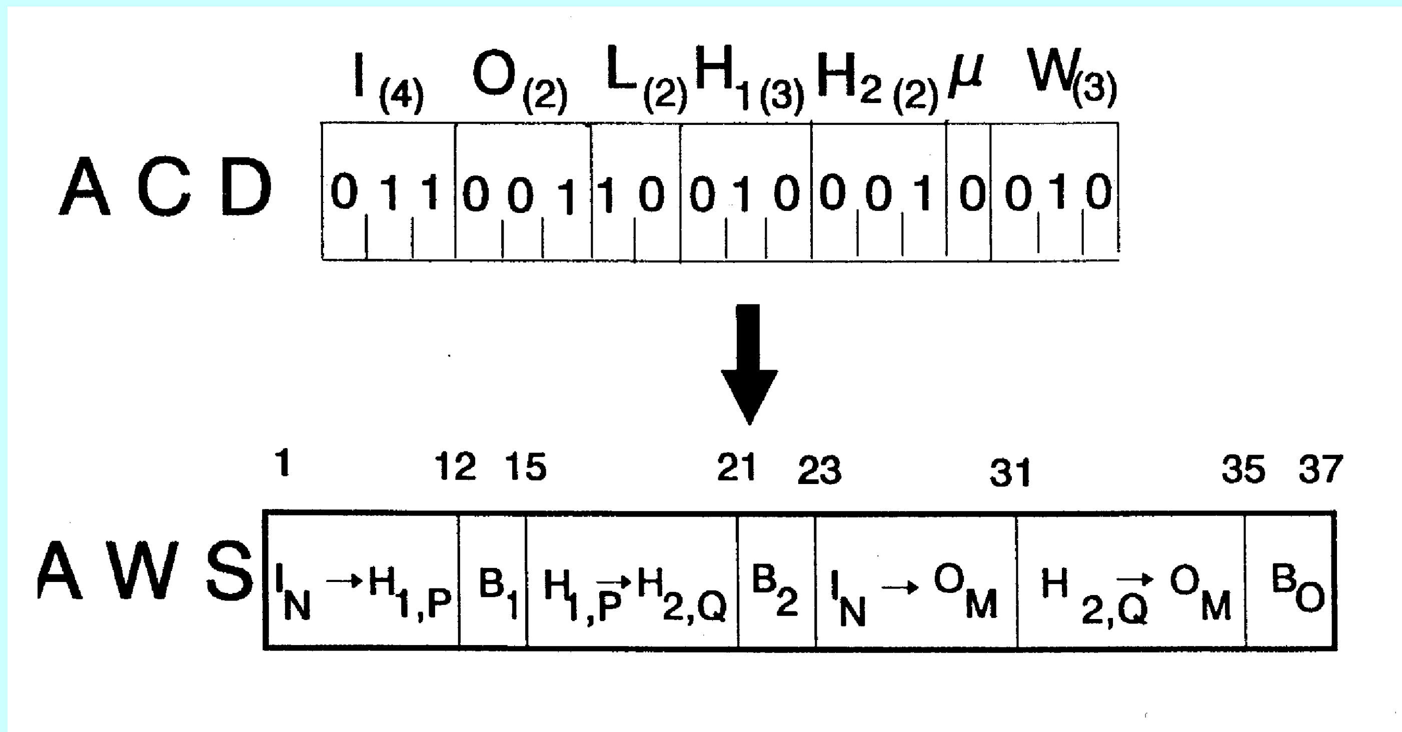



μ=0, connectivity feedforward. Lower triangular matrix.

μ=1, total connectivity. Recurrent neural networks.

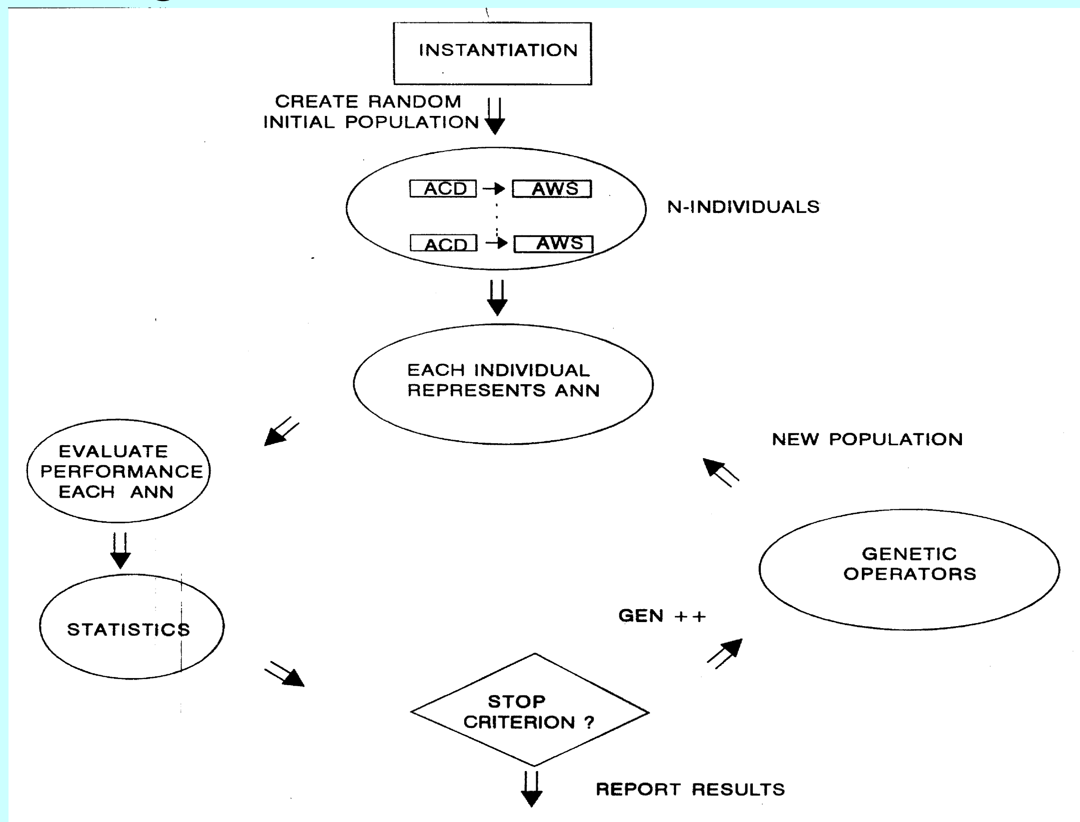
# ARCHITECTURE AND CONNECTION WEIGHTS

➪ Encoding example of an ANN with I=4, O=2, L=2, H1=3, H2=2, μ=0 and W=3.



# ARCHITECTURE AND CONNECTION WEIGHTS

➪ Flow diagram for simultaneous design of architecture and weights.




Fitness= $\theta(E)+\varsigma(H)+\eta(W)$

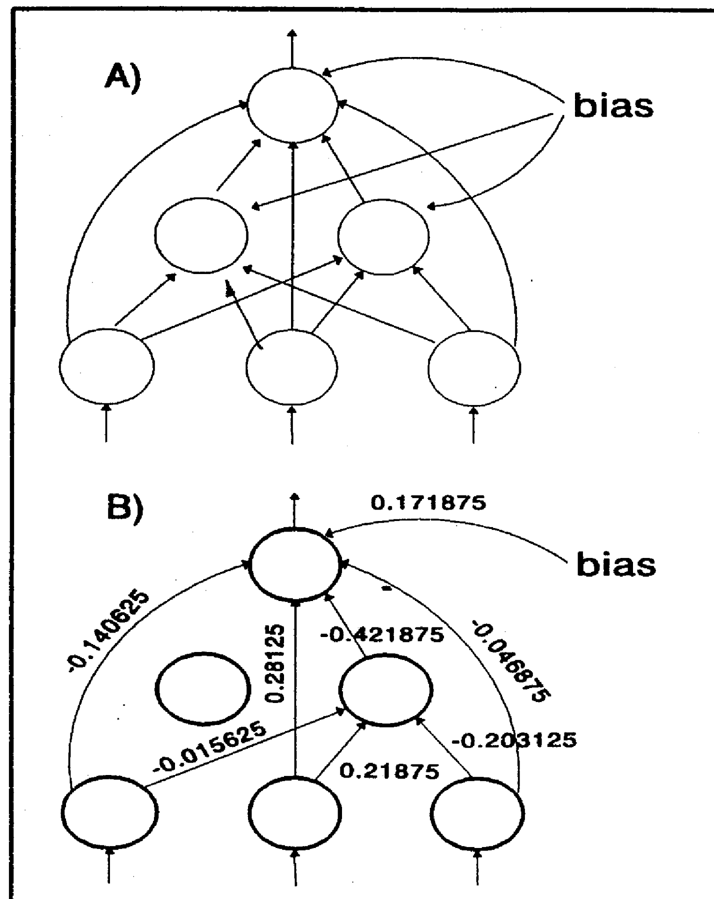
$\theta(E)=\lambda E$, Error

$\varsigma(H)= \rho\Sigma(H)$, Neurons

$\eta(W)= \delta(\Sigma abs((weights)+ \Sigma abs(bias))$, Weights

# ARCHITECTURE AND CONNECTION WEIGHTS

➪ The N-bit parity combinatorial problem. N=3.




$\mu=0$

Fitness= $\theta(E)+\varsigma(H)+\eta(W)$

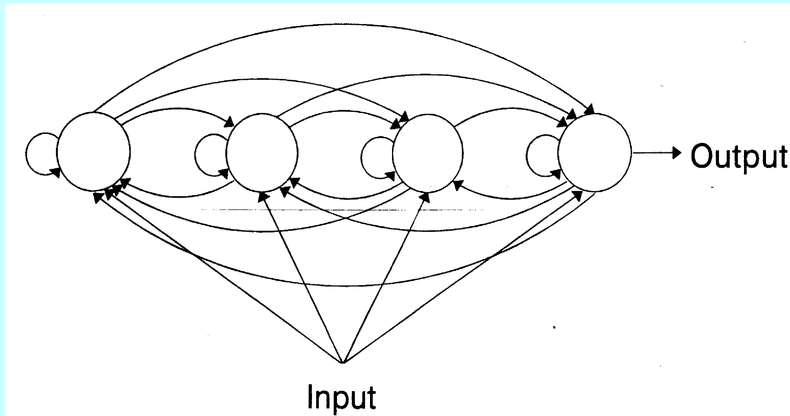
$\theta(E)=\lambda\Sigma_K\Sigma j(Yj(K)-dj(K))^2$

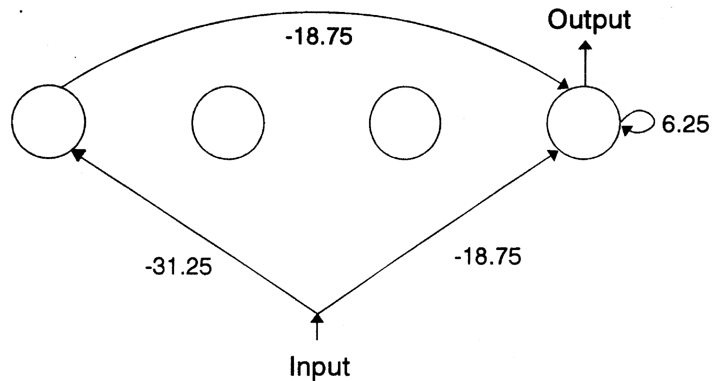
$\lambda=45$; $\rho=30$; $\delta=15$

*Dpto. Tecnología Electrónica*
*Universidad de Málaga*

# ARCHITECTURE AND CONNECTION WEIGHTS

- Sequence detection: 110



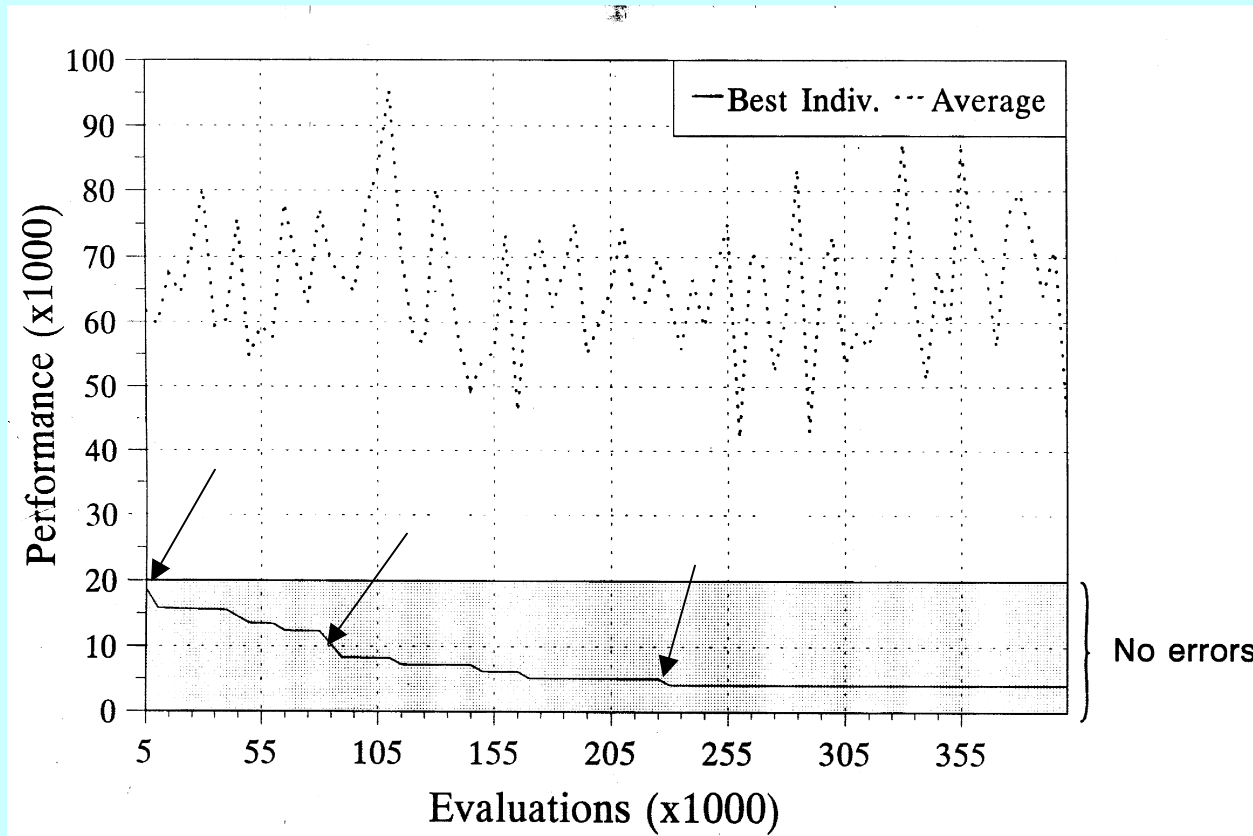




Input 10110111010111101101..
Output 00001000100000010010..

$\mu=1$
$\lambda=10^4$; $\rho=10^3$; $\delta=1$

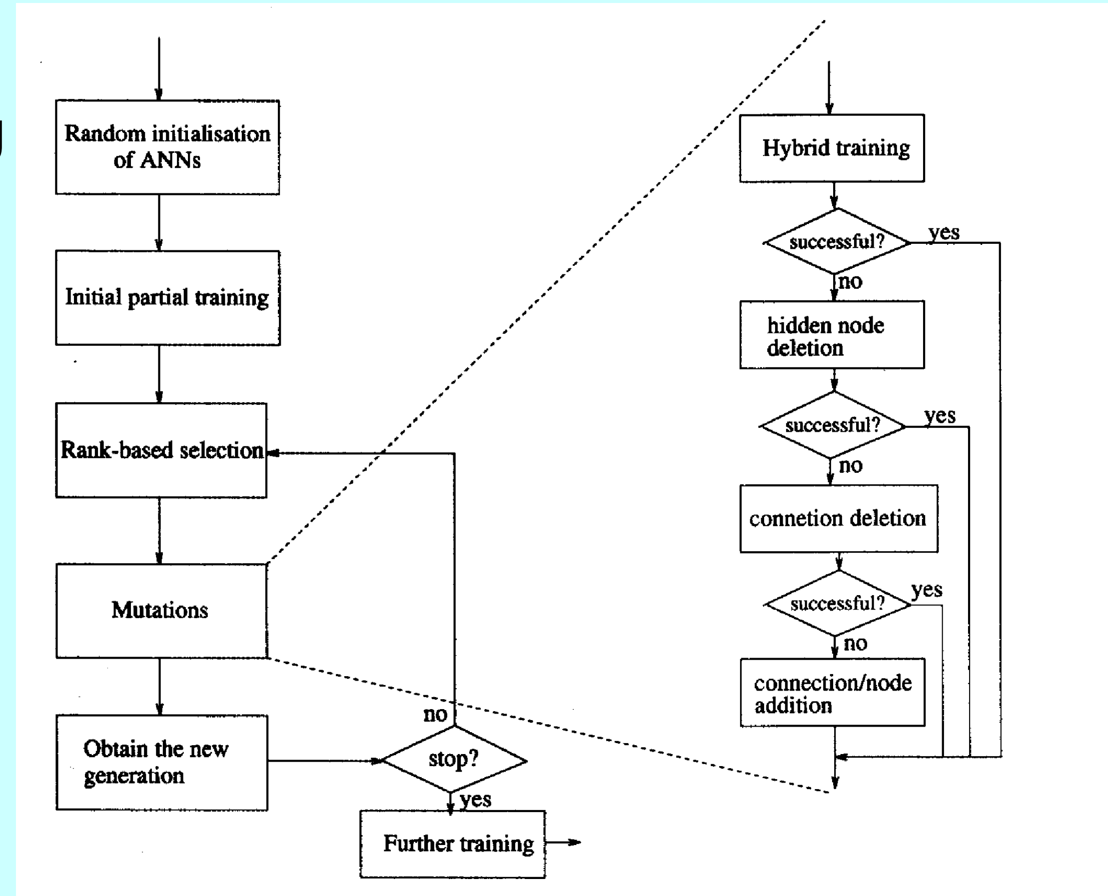
# ARCHITECTURE AND CONNECTION WEIGHTS

➪ Performance:



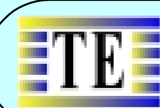
# ARCHITECTURE AND CONNECTION WEIGHTS

- Other proposal uses Evolutionary Programming to avoid crossover operators which may destroy both ANNs to be exchanged.



# CONTENTS


1. Introduction
2. Genetic algorithms
3. Designing ANN with evolutionary computation
    3.1. Connection weights in a defined architecture
    3.2. Evolution of architectures
    3.3. Node transfer functions
    3.4. Evolution of architecture and connection weights
    **3.5. Evolution of learning rules**
    3.6. ANN input data selection
4. Applications
5. Conclusions

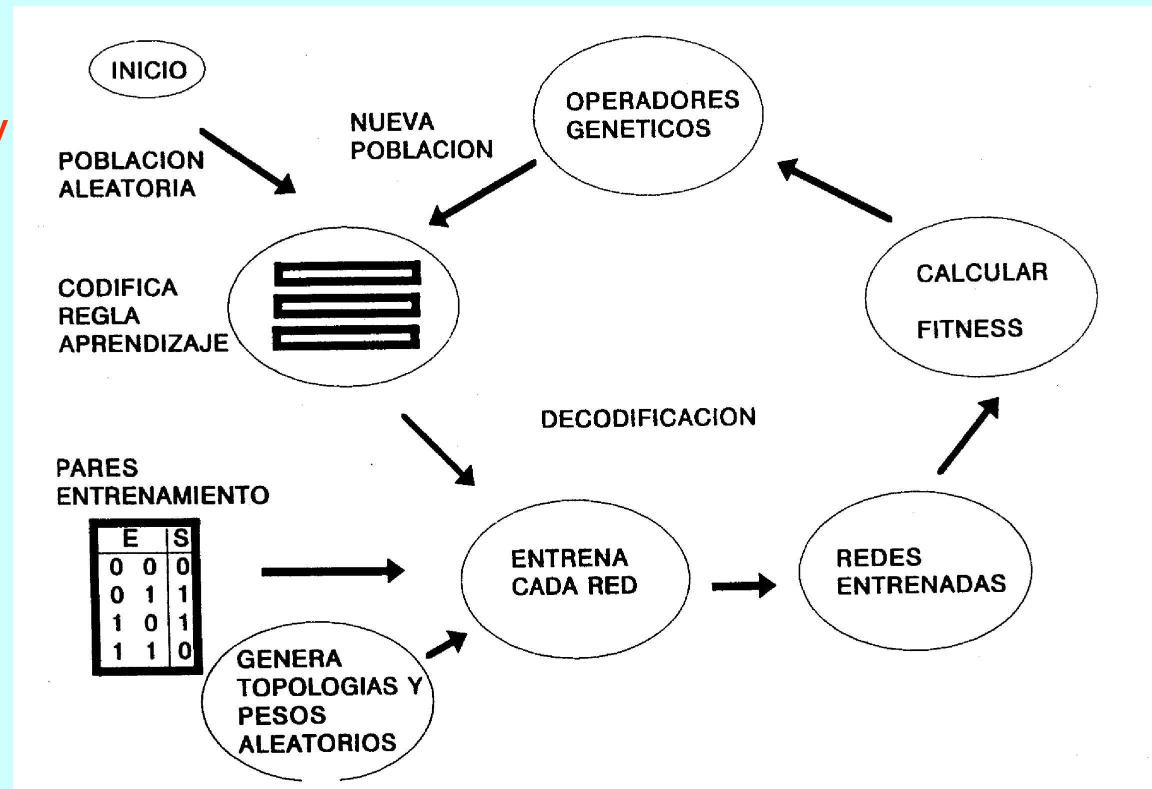



# EVOLUTION OF LEARNING RULES

➪ Training algorithm may have different performance when applied to different ANN architectures: relationship between network architecture and learning process is generally unknown.

➪ An ANN should adjust its learning rule adaptively according to its architecture and the task to be performed.

➪ GAs may guide the learning rule: evolution cycle of learning rules ⇒



# EVOLUTION OF LEARNING RULES

➪ Encoded learning is very noisy: we use Phenotype's fitness (an ANN training result) to approximate genotype's fitness (a learning rule's fitness).

➪ The evolution of learning rules has to work on the dynamic behavior of an ANN. How to encode the dynamic behavior of a learning rule into a static chromosome? Universal representation is impractical.

➪ Constrains:

- Weight-updating depends only of local information (activation of the input and output nodes, the current weight, ...).
- The learning rule is the same for all connections in an ANN.

➪ A learning rule is assumed to be a linear combination of these local variables and their products:

$$\Delta w(t) = \sum_{k=1}^{n} \sum_{i_1, i_2, \cdots, i_k = 1}^{n} \left( \theta_{i_1 i_2 \cdots i_k} \prod_{j=1}^{k} x_{i_j}(t-1) \right)$$

**t** is time, $\Delta\omega$ is the weight change, $\mathbf{x_i}$ are local variables, and **θ's** are real coefficients determined by the evolution.

## CONTENTS

1. Introduction

2. Genetic algorithms

3. Designing ANN with evolutionary computation

    3.1. Connection weights in a defined architecture

    3.2. Evolution of architectures

    3.3. Node transfer functions

    3.4. Evolution of architecture and connection weights

    3.5. Evolution of learning rules

    **3.6. ANN input data selection**

4. Applications

5. Conclusions

## INPUT DATA SELECTION

➪ In real problems, the possible inputs to an ANN can be quite large, and there may be some redundancy among different inputs.

➪ Besides statistical methods, GAs can be applied to find a near-optimal set of input features to an ANN.

➪ In the encoding of the problem, each individual in the population represents a portion of the input data.

➪ The evolution of an individual is carried out by training an ANN with these inputs and using the result to calculate its fitness value. The ANN architecture is often fixed.

## CONTENTS

1. Introduction

2. Genetic algorithms

3. Designing ANN with evolutionary computation

    3.1. Connection weights in a defined architecture

    3.2. Evolution of architectures

    3.3. Node transfer functions

    3.4. Evolution of architecture and connection weights

    3.5. Evolution of learning rules

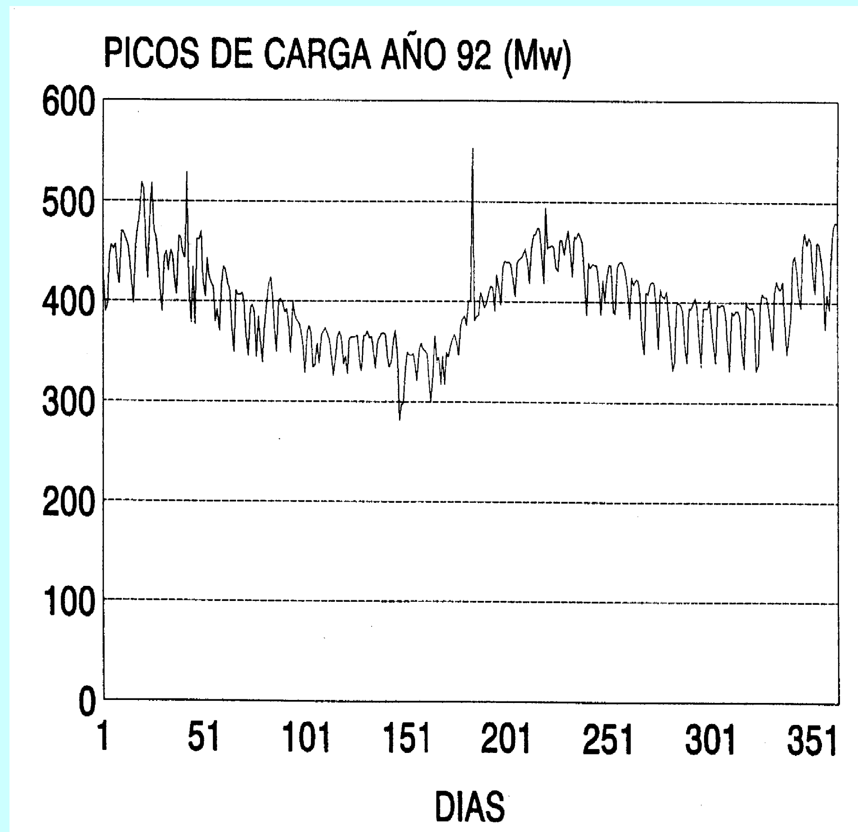
    3.6. ANN input data selection

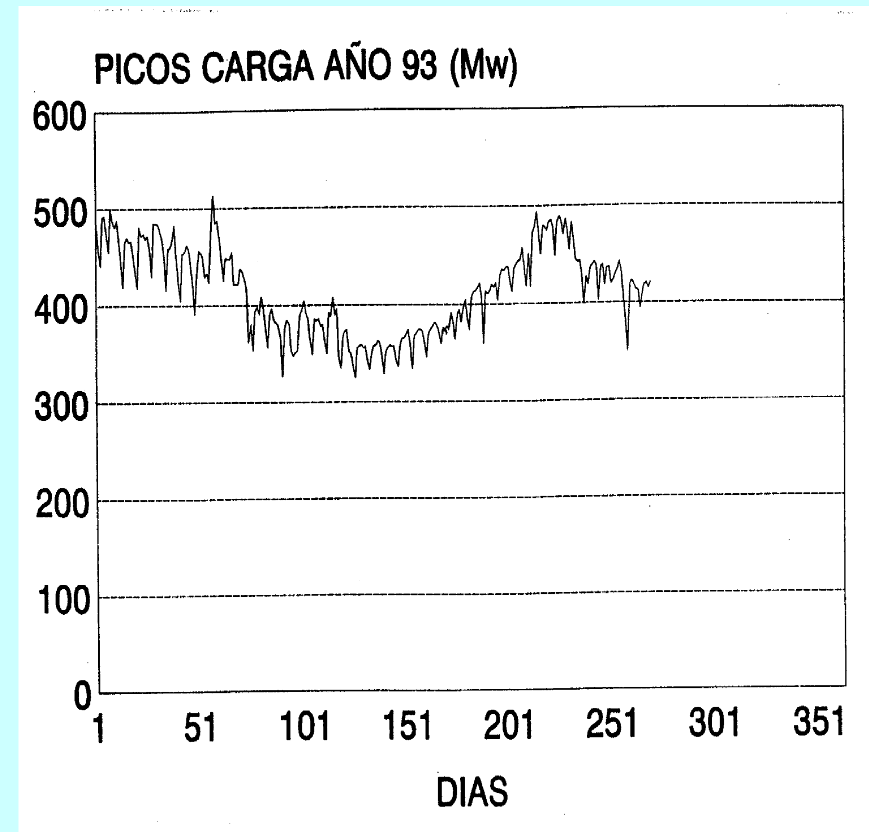
**4. Applications**

5. Conclusions

# APPLICATION TO ELECTRIC LOAD FORECASTING

➪ Power consumption in the province of Malaga during 1992-1993

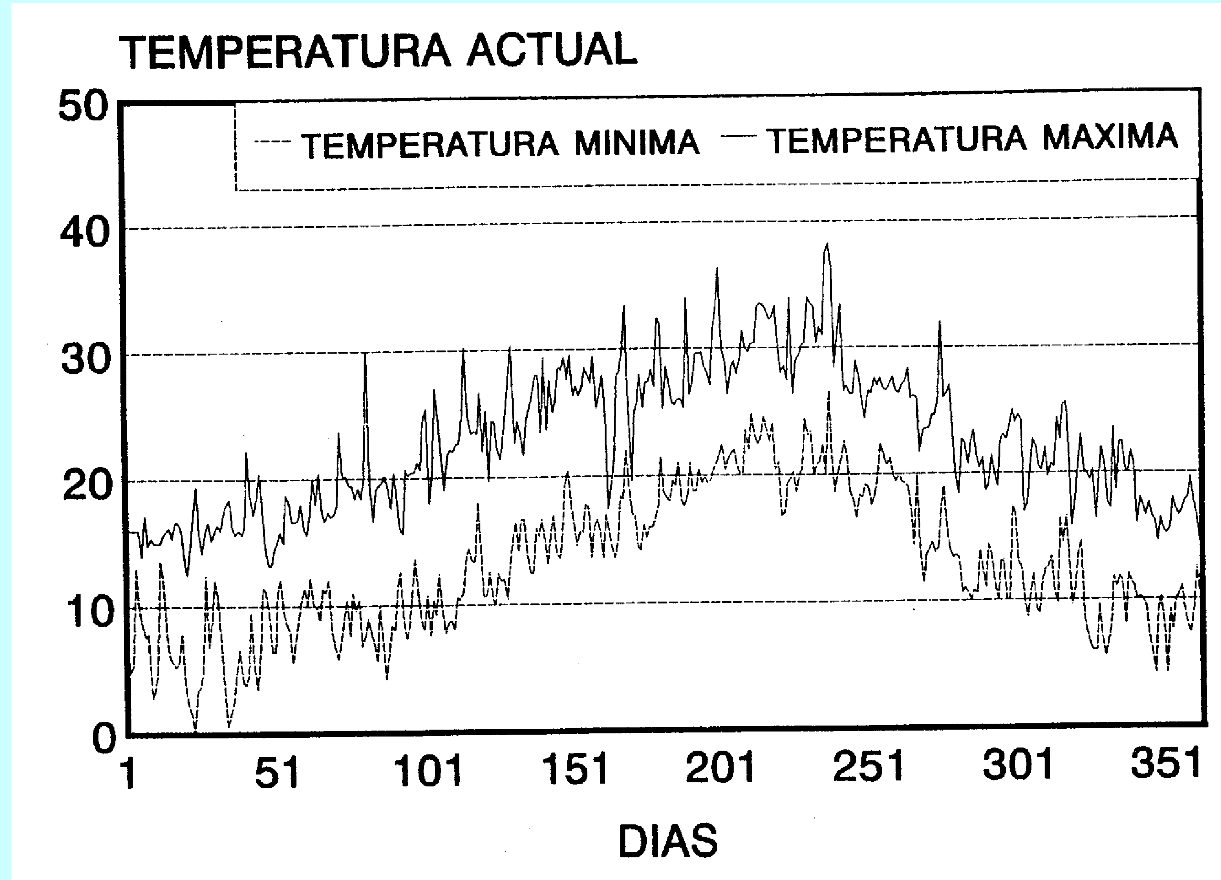


*Dpto. Tecnología Electrónica*
*Universidad de Málaga*

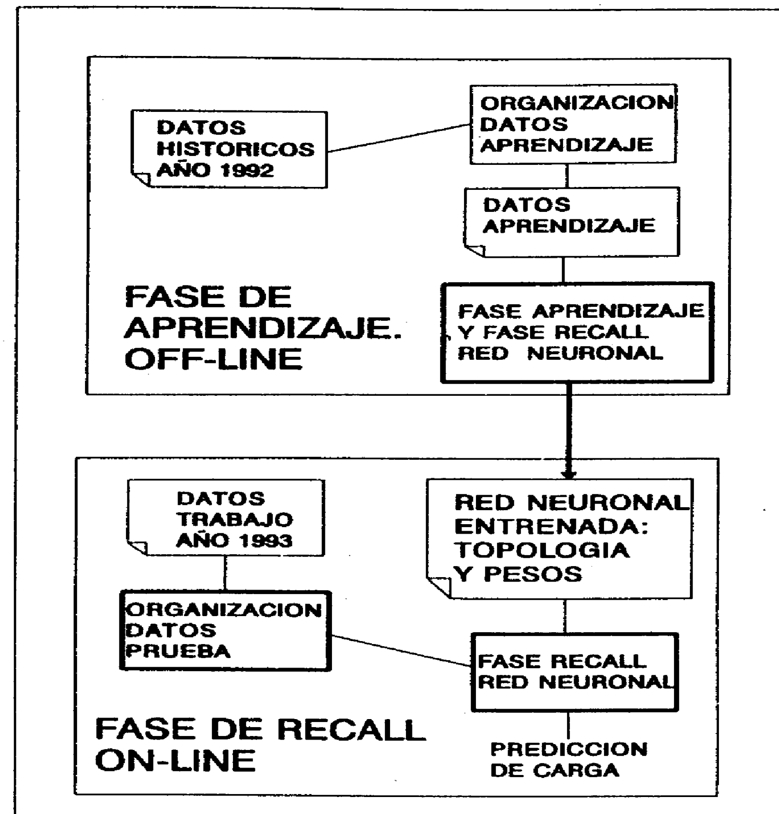
# APPLICATION TO ELECTRIC LOAD FORECASTING

➪ Maximum and minimum temperatures during the year 1992



# APPLICATION TO ELECTRIC LOAD FORECASTING

➪ Forecasting has been performed with statistical techniques, with ANNs and with genetic ANNs.



# APPLICATION TO ELECTRIC LOAD FORECASTING

➪ Results

ER92= Training relative error for year 1992.

ER93= Forecasting relative error for year 1993.

| | ER92% | ER93% | TOPOLOGÍA |
|---|---|---|---|
| ELMAN | 3,6191 | 2,8758 | 8-OCULTAS |
| ARIMAX | 3,6081 | 2,8815 | --------- |
| GÉNETICA FEED-BACK | 3,6724 | 2,8986 | 5-OCULTAS 48-PESOS |
| ARMAX | 3,6520 | 2,9071 | -------- |
| LEVENBERG MARQUARDT | 3,4077 | 2,9354 | 8-OCULTAS |
| GENÉTICA FORWARD | 3,4106 | 2,9360 | 5-OCULTAS 34-PESOS |
| HTA-GLOBAL | 3,3718 | 2,9412 | 8-OCULTAS |
| RBR | 4,0352 | 2,9595 | 8-OCULTAS |
| ARX | 3,7456 | 2,9718 | -------- |
| HTA-LOCAL | 3,3016 | 2,9728 | 8-OCULTAS |
| CC | 3,3214 | 2,9776 | 6-CASCADA |
| OBS | 3,3498 | 2,9815 | 6-OCULTAS 37-PESOS |
| OBD | 3,3818 | 2,9979 | 7-OCULTAS 46-PESOS |

# CONTENTS

1. Introduction

2. Genetic algorithms

3. Designing ANN with evolutionary computation

   3.1. Connection weights in a defined architecture

   3.2. Evolution of architectures

   3.3. Node transfer functions

   3.4. Evolution of architecture and connection weights

   3.5. Evolution of learning rules

   3.6. ANN input data selection

4. Applications

**5. Conclusions**

## CONCLUSIONS

- GAs have proved to be a powerful search method.
- Parallelization techniques are necessary to reduce the computation time.

- GAs can be introduced into the design of ANNs at many different levels:
  - The evolution of connection weights.
  - To find a near-optimal architecture automatically.
  - Simultaneous evolution of weights and architecture to get better results.
  - To allow an ANN to adapt its learning rule to its environment
- Its applicability to actual problems: electric load forecasting.