

# Evolutionary Design of Neural Networks

- Motivation
- Evolutionary training
- Evolutionary design of the architecture
- Evolutionary design of the learning rules

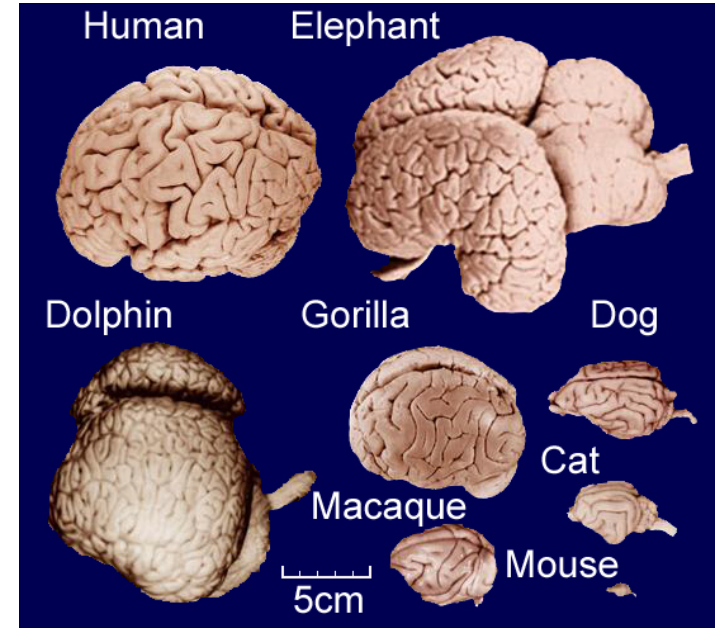
# Evolutionary Design of Neural Networks

- **Motivation.** Neural networks design consists of:
- **Choice of the architecture** (network topology + connectivity)
  - Has an influence on the network ability to solve the problem
  - Usually is a trial-and-error process
- **Train the network**
  - Is an optimization problem = find the parameters (weights) which minimize the error on the training set
  - The classical methods (e.g. gradient-based methods as is BackPropagation) have some drawbacks :
    - Risk of getting stuck in local minima
    - They cannot be applied if
      - the activation functions are not differentiable
      - the error function cannot be expressed directly as a function of the parameters (e.g. for recurrent networks)

# Evolutionary Design of Neural Networks

Idea: use an evolutionary process

- Inspired by the biological evolution of the brain
- The system is not explicitly designed but its structure derives by an evolutionary process involving a population of encoded neural networks
  - Genotype = the network codification (structural description)
  - Phenotype = the network itself, which can be simulated (functional description)



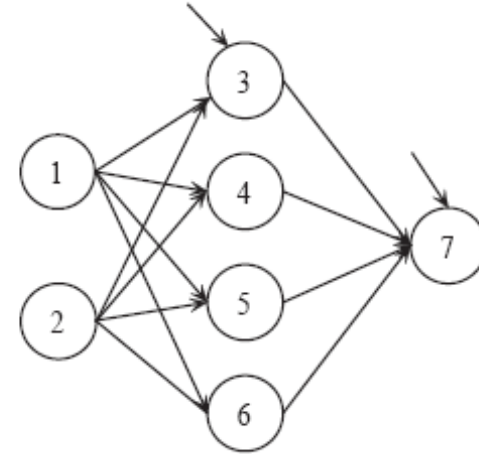
# Evolutionary Design of Neural Networks

## Variants

- Evolutionary training:
  - Estimate the weights using a global optimization metaheuristic
  - It can be used for networks with discontinuous activation functions and for recurrent networks
- Evolving architecture:
  - Evolve the number of units, types of activation, connections
- Evolving adjustment rules in the training process

# Evolutionary Training

- Use an evolutionary algorithm to solve the problem of minimizing the mean squared error on the training set



- Parameters: synaptic weights and biases

Training set :  $\{(x^1, d^1), \dots, (x^L, d^L)\}$

$$\text{Error function : } E(W) = \frac{1}{L} \sum_{l=1}^L (d^l - y^l)^2$$

$$W = \{w_{31}, w_{32}, w_{30}, w_{41}, w_{42}, w_{40}, \dots, w_7, w_{71}, \dots, w_{76}\}$$

# Evolutionary Training

Evolutionary algorithm components:

- ❑ **Encoding:** each element of the population is a real vector containing all adaptive parameters (similar to the case of evolution strategies)
- ❑ **Evolutionary operators:** typical to evolution strategies or evolutionary programming or other population-based metaheuristics (e.g. particle swarm optimization, differential evolution)
- ❑ **Evaluation:** the quality of an element depends on the mean squared error (MSE) on the training/validation set; an element is better if the MSE is smaller

# Evolutionary Training

## Applications:

- For networks with non-differentiable or non-continuous activation functions
- For recurrent networks (the output value cannot be explicitly computed from the input value, thus the derivative based learning algorithms cannot be applied)

## Drawbacks:

- More costly than traditional non-evolutionary training
- It is not appropriate for fine tuning the parameters

## Hybrid versions:

- Use an EA to explore the parameter space and a local search technique to refine the values of the parameters

# Evolutionary Training

Remark. EAs can be used to preprocess the training set

- Selection of attributes
- Selection of examples



# Evolutionary Pre-processing

## Selection of attributes (for classification problems)

- Motivation: if the number of attributes is large the training is difficult
- It is important when some of the attributes are not relevant for the classification task
- The aim is to select the relevant attributes
- For initial data having  $N$  attributes the encoding could be a vector of  $N$  binary values (0 – not selected, 1 – selected)
- The evaluation is based on training the network for the selected attributes (this corresponds to a wrapper-like technique of attributes selection)

# Evolutionary Pre-processing

**Example:** identify patients with cardiac risk

**Total set of attributes:**

(age, weight, height, body mass index, blood pressure, cholesterol, glucose level)

**Population element:** (1,0,0,1,1,1,0)

**Corresponding subset of attributes:**

(age, body mass index, blood pressure, cholesterol)

**Evaluation:** train the network using the subset of selected attributes and compute the accuracy; the fitness value will be proportional to the accuracy

**Remark:**

- This technique can be applied also for non neural classifiers (ex: Nearest-Neighbor)
- It is called “wrapper based attribute selection”

# Evolutionary Pre-processing

## Selection of examples

- Motivation: if the training set is large the training process is costly and there is a higher risk of overfitting
- It is similar to attribute selection
- Binary encoding (0 – not selected, 1 – selected)
- The evaluation is based on training the network (using any training algorithm) for the subset specified by the binary encoding

# Evolving architecture

## Non-evolutionary approaches:

- Growing networks
  - Start with a small size network
  - If the training stagnates add a new unit
  - The assimilation of the new unit is based on adjusting, in a first stage, only its weights
  -
- Pruning networks
  - Start with a large size network
  - The units and connection which do not influence the training process are eliminated

# Evolving architecture

## Elements which can be evolved:

- Number of units
- Connectivity
- Activation function type

## Encoding variants:

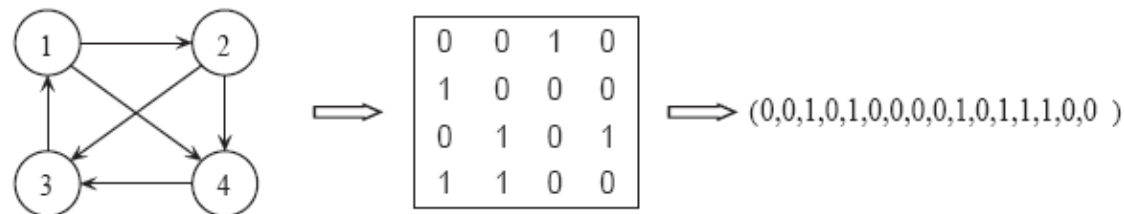
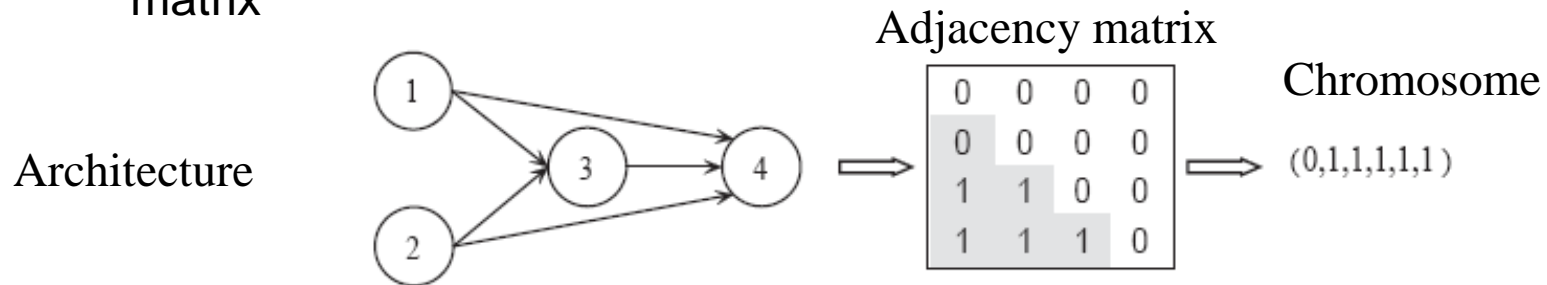
- Direct
- Indirect

# Evolving architecture

Direct encoding: each element of the architecture appears explicitly in the encoding

- Network architecture = oriented graph
- The network can be encoded by the adjacency matrix

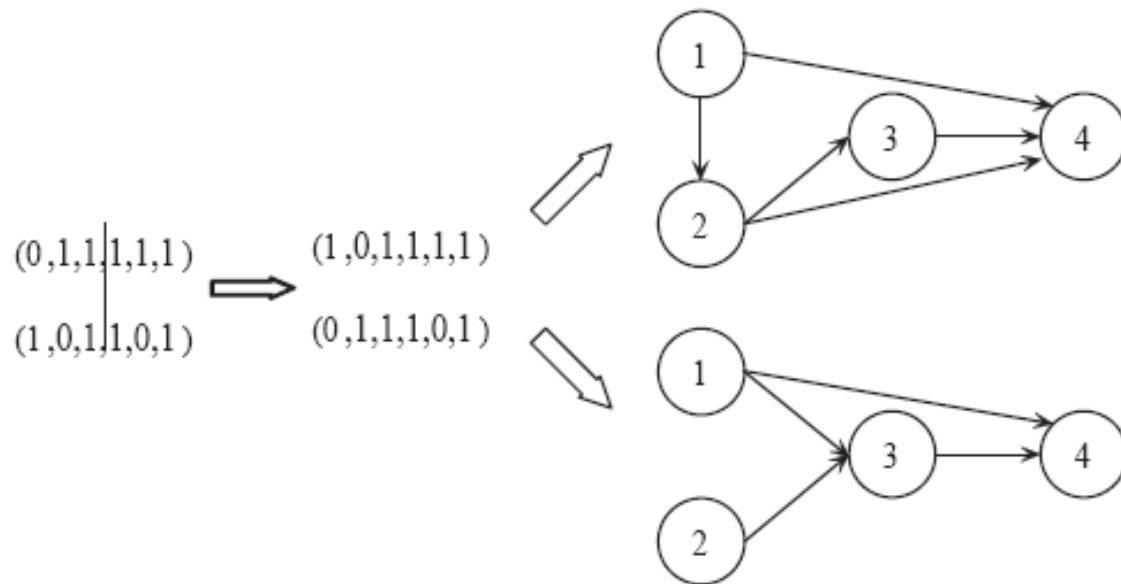
Rmk. For feedforward networks the units can be numbered such that the unit  $i$  receives signals only from units  $j$ , such that  $j < i \Rightarrow$  inferior triangular matrix



# Evolving architecture

## Operators

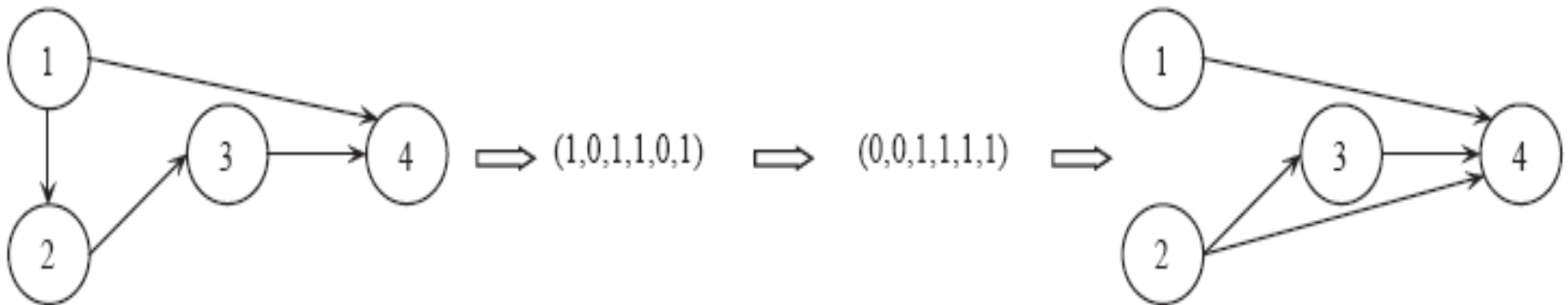
- Crossover similar to that used for genetic algorithms



# Evolving architecture

## Operators:

- Mutation similar to that used for genetic algorithms





# Evolving architecture

Evolve the number of units and connections

Hypothesis:  $N$  – maximal number of units

Encoding:

- Binary vector with  $N$  elements
  - 0: inactivated unit
  - 1: active unit
- Adjacency matrix  $N \times N$ 
  - For a zero element in the unit vector the corresponding row and column in the matrix are ignored.

# Evolving architecture

Evolving the activation function type:

Encoding :

- Binary vector with N elements
  - 0: inactivated unit
  - 1: active unit with activation function of type 1 (ex: tanh)
  - 2: active unit with activation function of type 2 (ex: logistic)
  - 3: active unit with activation function of type 3 (ex: linear)

Evolution of weights

- The adjacency matrix is replaced with the matrix of weights
  - 0: no connection
  - $\neq 0$ : weight value

# Evolving architecture

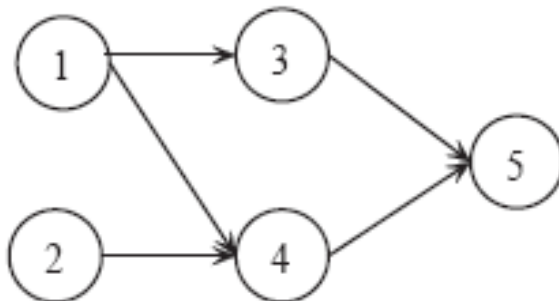
## Evaluation:

- The network is trained
- The training error is estimated ( $E_a$ )
- The validation error is estimated ( $E_v$ )
- The fitness is inverse proportional to:
  - Training error
  - Validation error
  - Network size

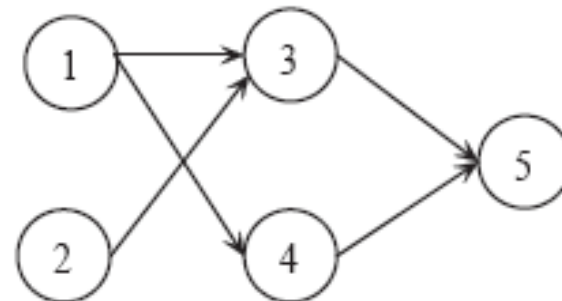
# Evolving architecture

## Drawbacks of the direct encoding:

- It is not scalable
- Can lead to different representations of the same network ([permutation problem](#))
- It is not appropriate for modular networks



(0,1,0,1,1,0,0,0,1,1)



(0,1,1,1,0,0,0,0,1,1)

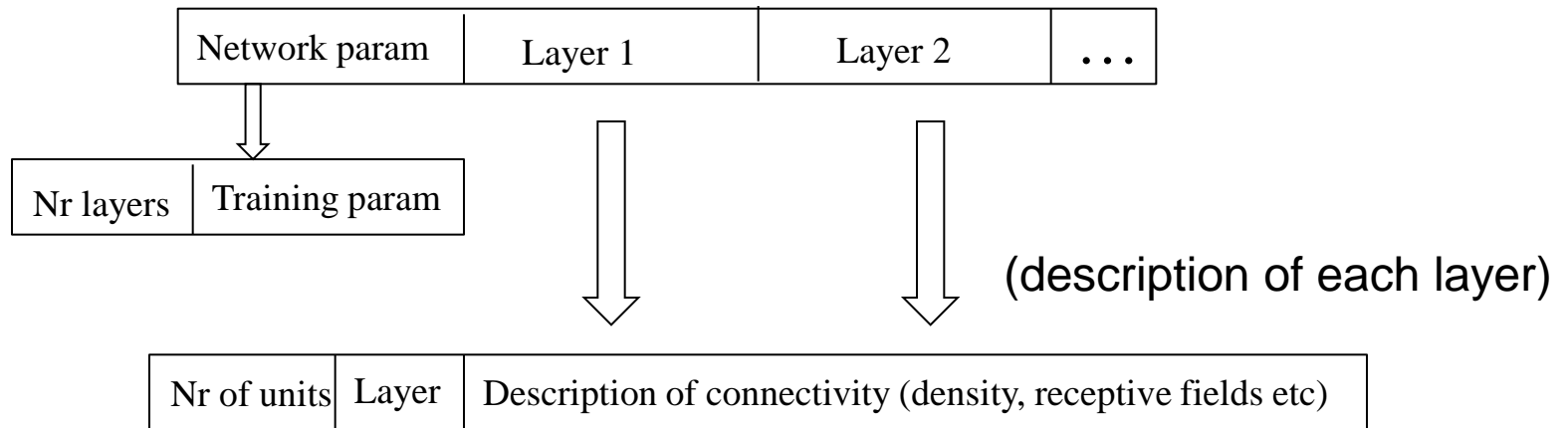
# Evolving architecture

Indirect encoding:

- Biological motivation
- Parametric encoding
  - The network is described by a set of characteristics (fingerprint)
  - Particular case: feedforward network with variable number of hidden units
  - The fingerprint is instantiated in a network only for evaluation
- Rules-based encoding

# Evolving architecture

- Parametric encoding



**Instantiation:** random choice of connections according to the specified characteristics

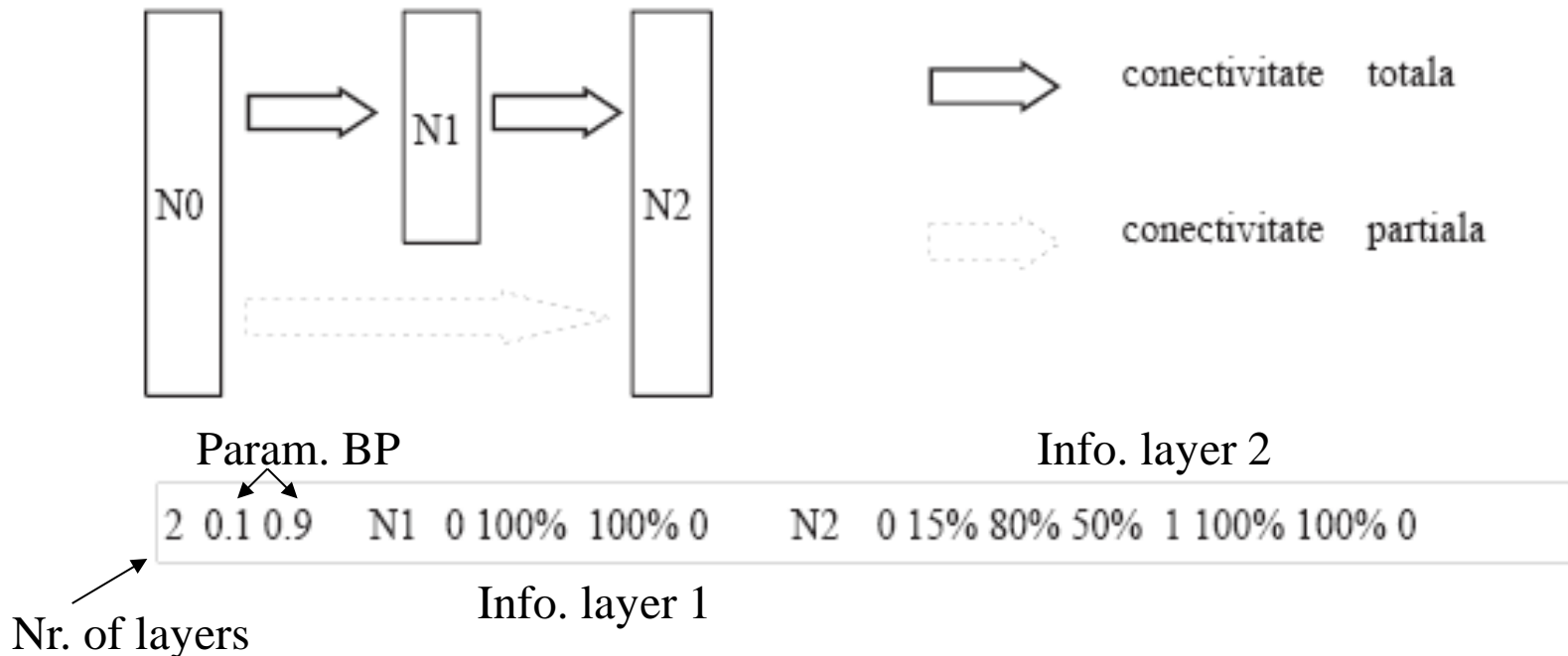
# Evolving architecture

Example:

Operators:

**Mutation:** change the network characteristics

**Recombination:** combine characteristics of layers



# Evolving architecture

Rule-based encoding (similar to Grammar Evolution) :

General rule  $sn \rightarrow \begin{pmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{pmatrix}$

Examples:

$$S \rightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \quad A \rightarrow \begin{pmatrix} a & a \\ a & a \end{pmatrix}, \quad B \rightarrow \begin{pmatrix} b & b \\ b & a \end{pmatrix}, \quad C \rightarrow \begin{pmatrix} b & a \\ a & c \end{pmatrix}, \quad D \rightarrow \begin{pmatrix} a & d \\ a & d \end{pmatrix},$$
$$a \rightarrow \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad b \rightarrow \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad c \rightarrow \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \quad d \rightarrow \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}.$$

Structure of an element:

$(A, B, C, D, a, a, a, a, b, b, b, a, b, a, a, c, a, d, a, d, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0)$

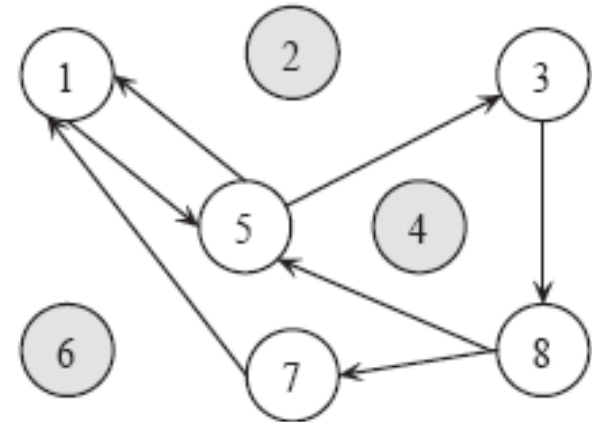


# Evolving architecture

Deriving an architecture:

$$S \rightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix} \rightarrow \begin{pmatrix} a & a & b & b \\ a & a & b & a \\ b & a & a & d \\ a & c & a & d \end{pmatrix} \rightarrow$$

$$\rightarrow \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



# Evolving architecture

Drawback of separate evolution of the architecture and weights:

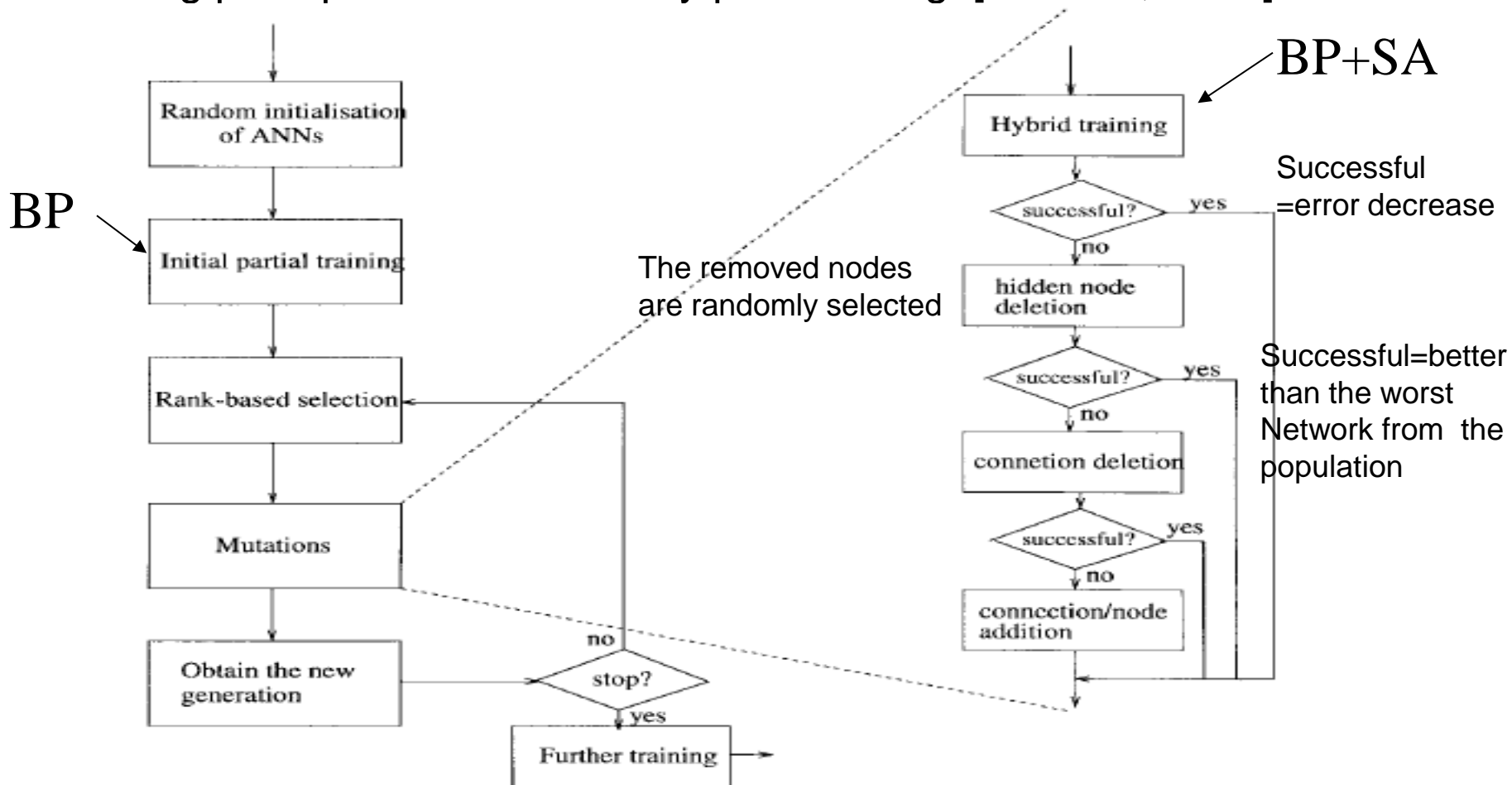
- Since the behaviour of an architecture depends on the weights values, at different evaluations steps, same architecture can have various fitness values (caused by different training processes) - thus the fitness is noisy

Solutions:

- Repeat the training of the same architecture and compute an averaged fitness => high computational costs
- Simultaneous evolution of the architecture and weights (it will ensure a one-to-one mapping between the genotype – the architecture - and the phenotype – the trained network)

# EPNet

Exemplu: EPNet = evolutionary design of feedforward neural networks using principles of evolutionary programming [Xin Yao, 1996]

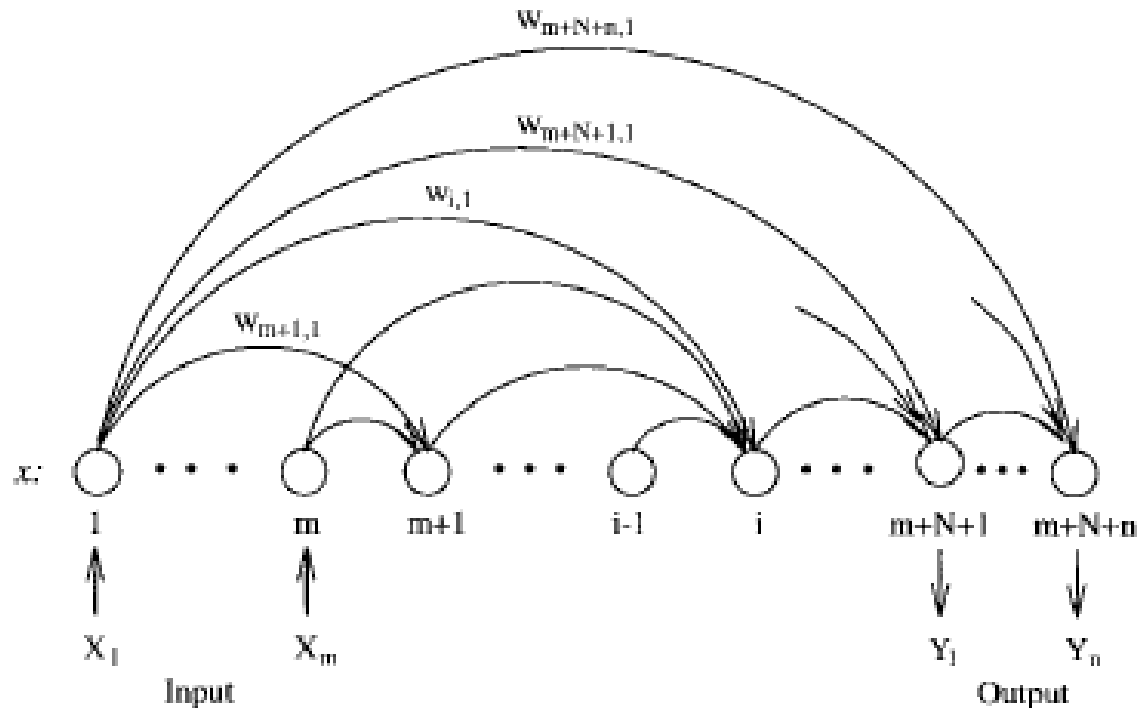


# EPNet

Network encoding:

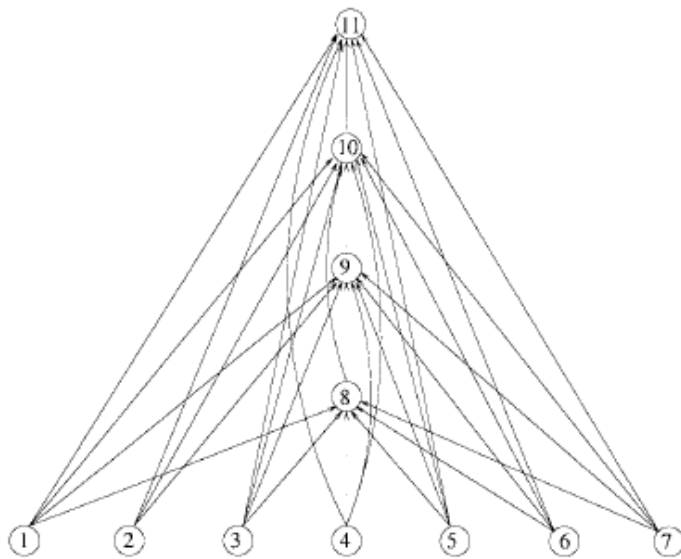
list of hidden units +  
Connectivity matrix+  
Weight matrix

**Example:** each neuron (except for the first  $m$  which are input neurons) is connected to all previous neurons

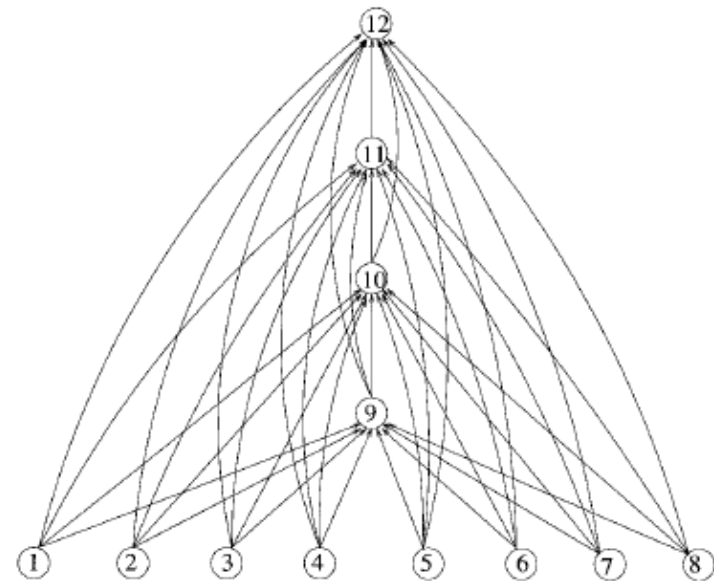


# EPNet

Architectures evolved by EPNet for the parity problem



n=7



n=8

# NEAT

NEAT = NeuroEvolution of Augmenting Topologies  
(<http://nn.cs.utexas.edu/?neat>)

- Direct encoding:
  - List of nodes (neurons)
    - Type of the nodes: input, hidden, output, bias
  - List of connections; for each connection:
    - In-node
    - Out-node
    - Connection weight
    - Activation bit (0 – active connection, 1- disabled connection)
    - Innovation value

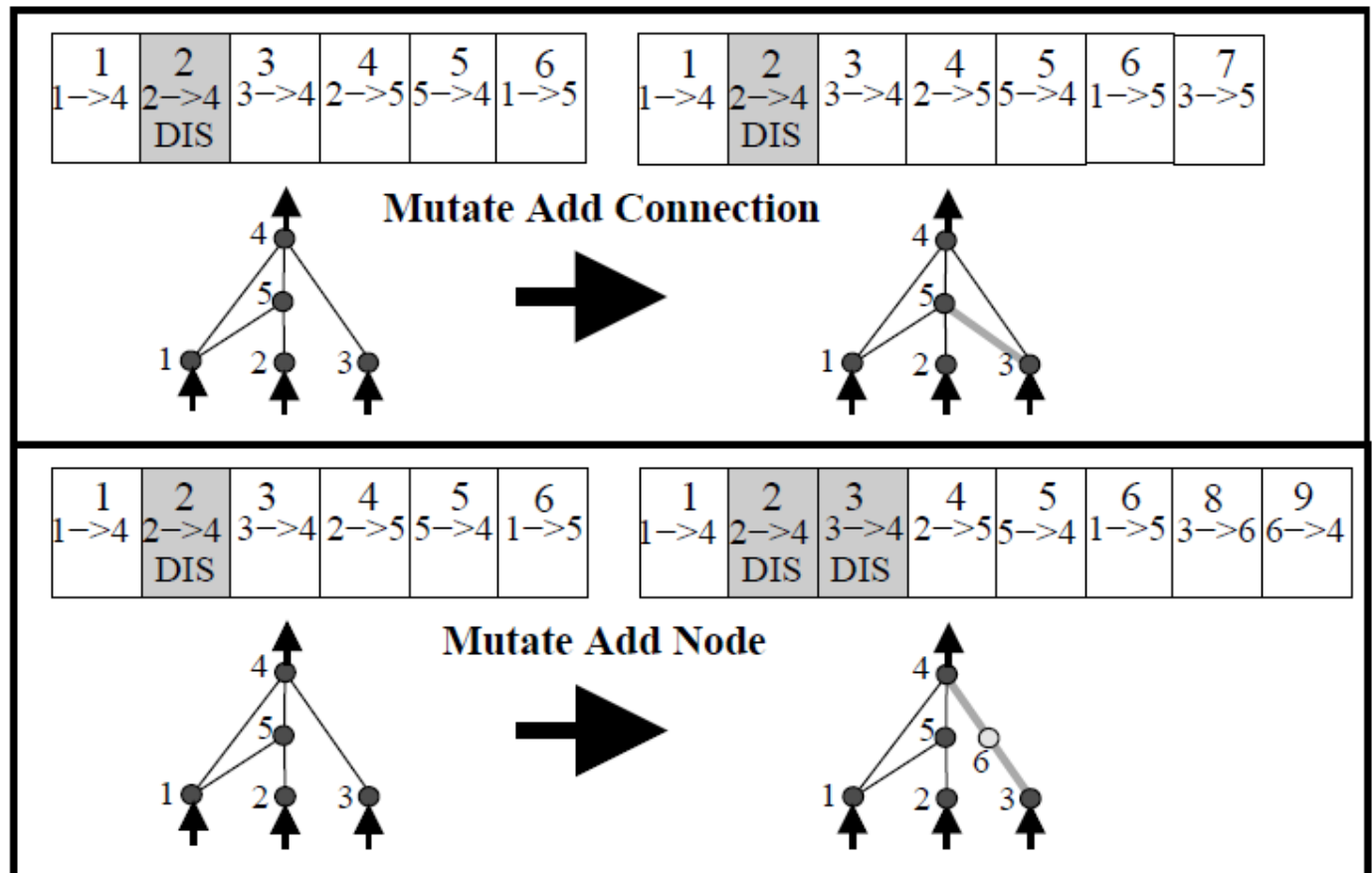
# NEAT

NEAT = NeuroEvolution of Augmenting Topologies  
(<http://nn.cs.utexas.edu/?neat>)

- The initial population consists of simple architectures (only input and output layers)
- Mutation variants:
  - **Node adding:** insert a new node between two already connected nodes (the old connection is removed and two other connections are added: that entering the new node has the weight=1, that going out from the new node has the weight of the removed connection)
  - **Connection adding:** a new connection (with a random weight) is added between two previously unconnected nodes

# NEAT

Mutation example: (K.Stanley, R. Miikulainen – Evolving Neural Networks through Augmenting Topologies, Evol.Comput. 2002)





# NEAT

## Crossover:

2 parents ---- 1 offspring

Similar to uniform crossover used in genetic algorithms

**Step 1:** identify the matching genes from the two parents based on the innovation values

- Two genes match if they have the same innovation value (this value is assigned when the gene is created)
- The non-matching genes are disjoint or in excess genes

# NEAT

## Crossover:

2 parents ---- 1 offspring

Similar to uniform crossover used in genetic algorithms

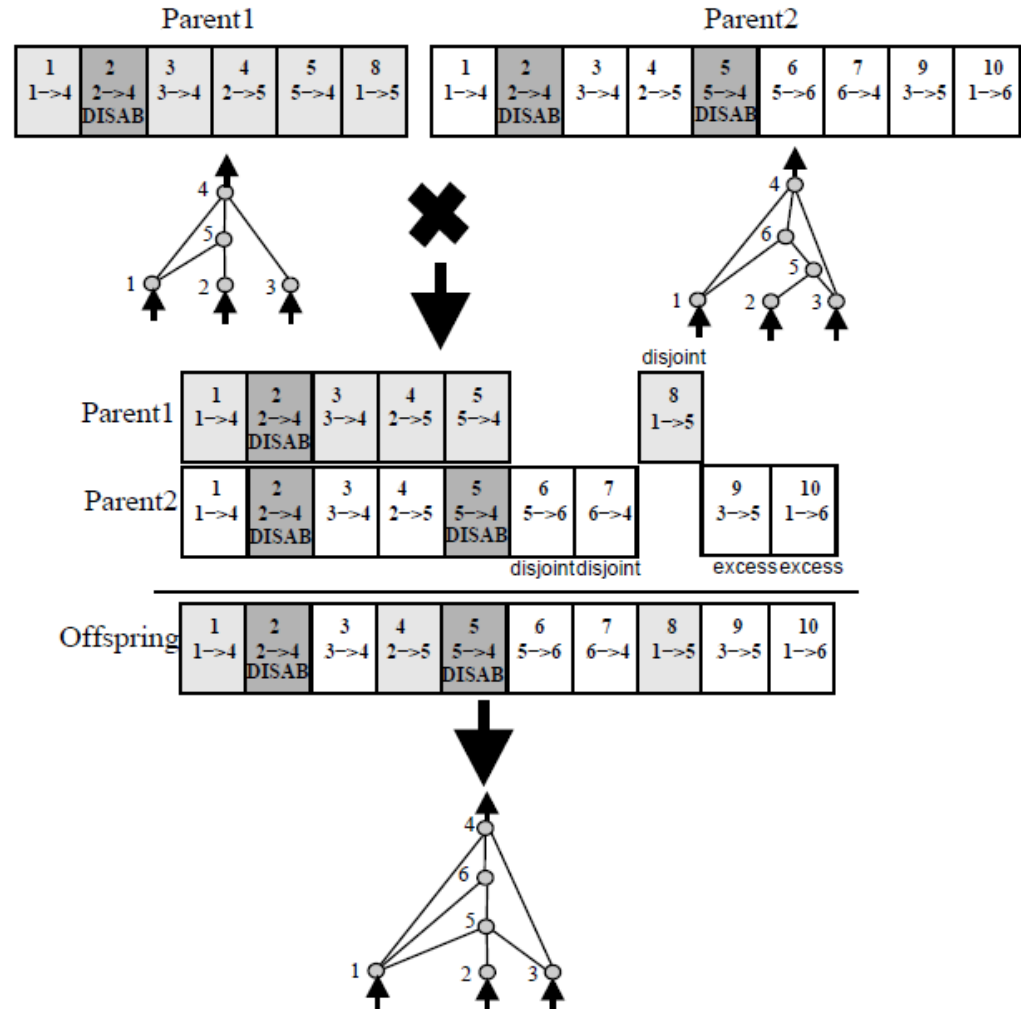
## Step 2: offspring construction

- For matching genes the offspring will receive the gene from one of the parents (randomly selected)
- The in excess/disjoint genes are transferred into the offspring either based on a probabilistic decision or based on the fitness of the parents (the gene its transferred if it belongs to the better parent)

# NEAT

Crossover example

(Stanley,  
Miikulainen,  
2002)



# Evolving learning rules

General form of a local adjustment rule

$$w_{ij}(k+1) = \varphi(w_{ij}(k), x_i, y_i, x_j, y_j, \delta_i, \delta_j, \alpha)$$

$x_i, x_j$  – input signals

$y_i, y_j$  – output signals

$\alpha$  – control parameters (ex: learning rate)

$\delta_i, \delta_j$  – error signal

Example: BackPropagation

$$w_{ij}(k+1) = w_{ij}(k) + \eta \delta_i y_j$$

# Evolving learning rules

Elements which can be evolved:

- Parameters of the learning process (ex: learning rate, momentum coefficient)
- The adjusting expression (see Genetic Programming)

Evaluation:

- Train networks using the corresponding rule

Drawback: very high cost

# Evolutionary Deep Neural Networks

M.J.Shafiee, A. Mishra, and A. Wong - Deep Learning with Darwin: Evolutionary Synthesis of Deep Neural Networks, 2016

**DeepNN** = neural networks with many layers

**Motivation:** evolve efficient Deep NN (instead of trying to compress existing DeepNN in order to make them more efficient)

**Main ideas:**

- the deep neural networks architecture are encoded using synaptic probability models (interpreted as network DNA)
- new networks are synthesized using these probability models which are further trained

**Implemented mechanisms:** heredity, natural selection, random mutation

# Evolutionary Deep Neural Networks

M.J.Shafiee, A. Mishra, and A. Wong - Deep Learning with Darwin: Evolutionary Synthesis of Deep Neural Networks, 2016

Particularities:

- usage of the exponential distribution as probability model for synaptic weights
- Impose constraints on the number of synapses (e.g. an offspring has at most 50% of the synapses of its parent).