Hybrid intelligent systems: Neural Expert Systems and Evolutionary Neural Networks

- Introduction
- Neural Expert Systems
- Evolutionary Neural Networks

Introduction

- A **hybrid intelligent system** is one that combines at least two intelligent technologies. For example, combining a neural network with a fuzzy system results in a hybrid neuro-fuzzy system.
- The combination of probabilistic reasoning, fuzzy logic, neural networks and evolutionary computation forms the core of **soft computing**, an emerging approach to building hybrid intelligent systems capable of reasoning and learning in an uncertain and imprecise environment.

- Although words are less precise than numbers, precision carries a high cost. We use words when there is a tolerance for imprecision. Soft computing exploits the tolerance for uncertainty and imprecision to achieve greater tractability and robustness, and lower the cost of solutions.
- We also use words when the available data is not precise enough to use numbers. This is often the case with complex problems, and while "hard" computing fails to produce any solution, soft computing is still capable of finding good solutions.

- Lotfi Zadeh is reputed to have said that a good hybrid would be “British Police, German Mechanics, French Cuisine, Swiss Banking and Italian Love”. But “British Cuisine, German Police, French Mechanics, Italian Banking and Swiss Love” would be a bad one. Likewise, a hybrid intelligent system can be good or bad – it depends on which components constitute the hybrid. So our goal is to select the right components for building a good hybrid system.

Comparison of Expert Systems, Fuzzy Systems, Neural Networks and Genetic Algorithms

	ES	*FS*	*NN*	*GA*
Knowledge representation	✓	✔	□	▨
Uncertainty tolerance	✓	✔	✔	✔
Imprecision tolerance	□	✔	✔	✔
Adaptability	□	▨	✔	✔
Learning ability	□	□	✔	✔
Explanation ability	✔	✔	□	▨
Knowledge discovery and data mining	□	▨	✔	✓
Maintainability	□	✓	✔	✓

* The terms used for grading are:

□ - bad, ▨ - rather bad, ✓ - rather good and ✔ - good

Neural expert systems

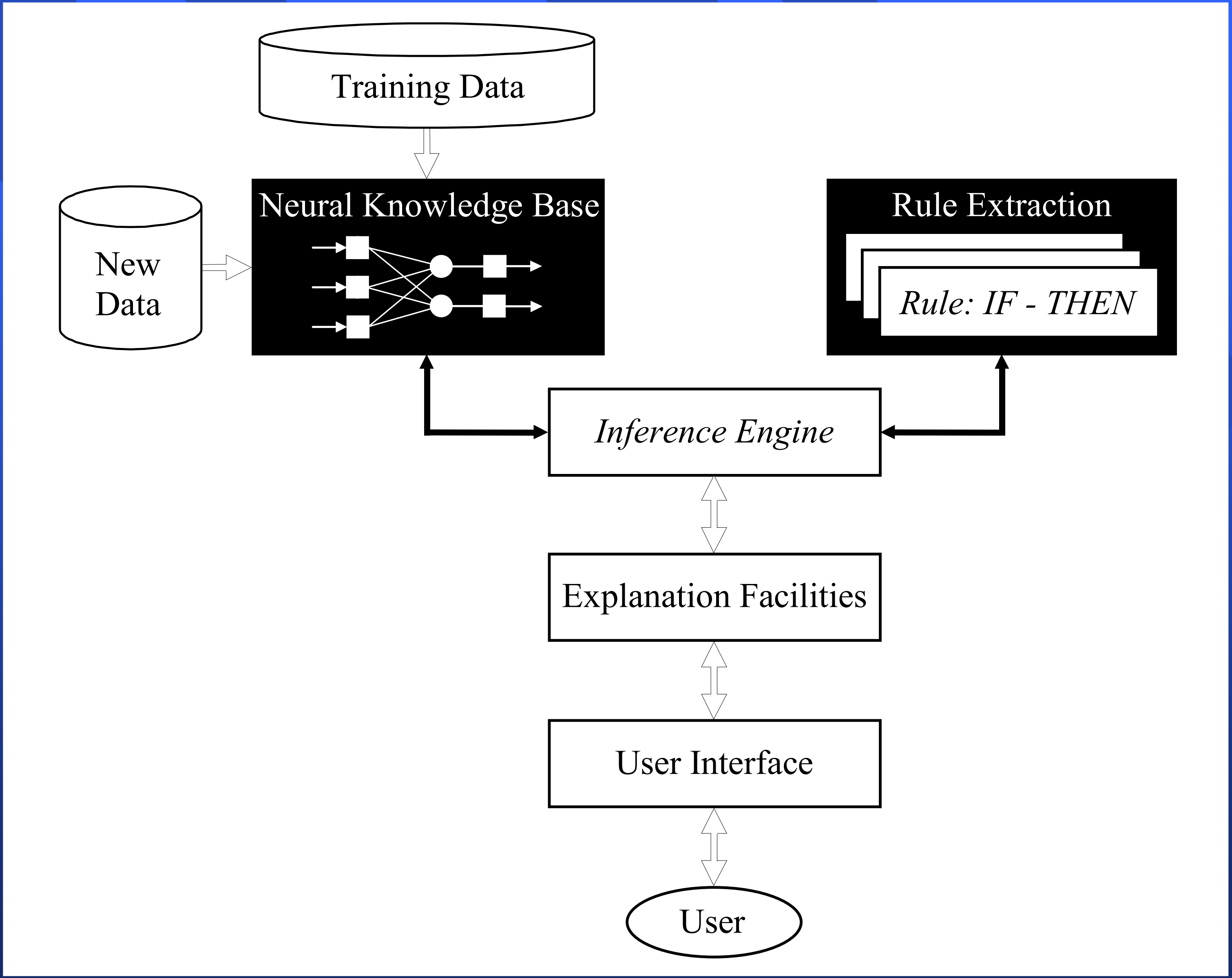
- Expert systems rely on logical inferences and decision trees and focus on modelling human reasoning. Neural networks rely on parallel data processing and focus on modelling a human brain.
- Expert systems treat the brain as a black-box. Neural networks look at its structure and functions, particularly at its ability to learn.
- Knowledge in a rule-based expert system is represented by IF-THEN production rules. Knowledge in neural networks is stored as synaptic weights between neurons.

- In expert systems, knowledge can be divided into individual rules and the user can see and understand the piece of knowledge applied by the system.
- In neural networks, one cannot select a single synaptic weight as a discrete piece of knowledge. Here knowledge is embedded in the entire network; it cannot be broken into individual pieces, and any change of a synaptic weight may lead to unpredictable results. A neural network is, in fact, a **black-box** for its user.

Can we combine advantages of expert systems and neural networks to create a more powerful and effective expert system?

A hybrid system that combines a neural network and a rule-based expert system is called a **neural expert system** (or a **connectionist expert system**).

Basic structure of a neural expert system



The heart of a neural expert system is the **inference engine**. It controls the information flow in the system and initiates inference over the neural knowledge base. A neural inference engine also ensures **approximate reasoning**.

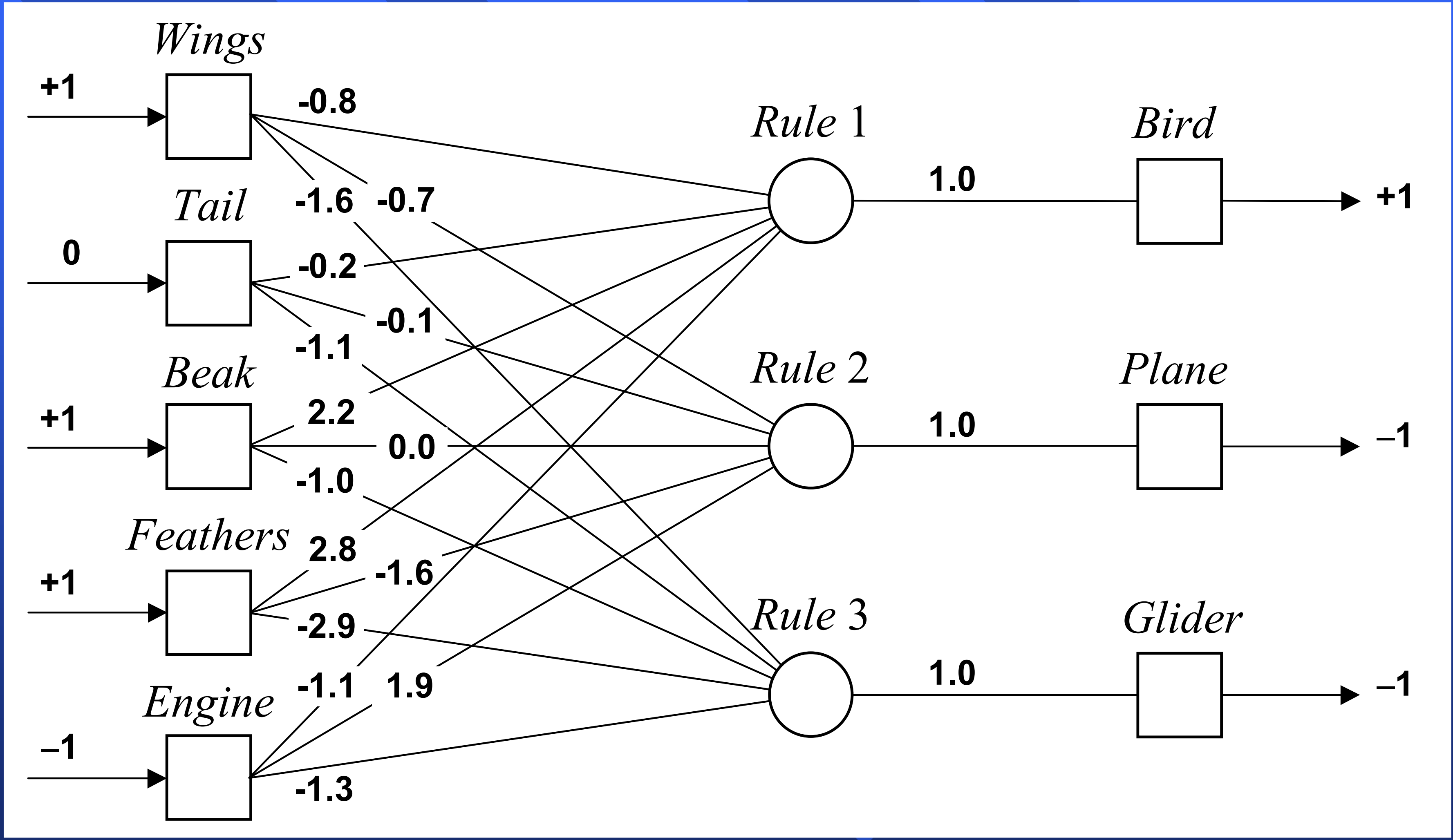
Approximate reasoning

- In a rule-based expert system, the inference engine compares the condition part of each rule with data given in the database. When the IF part of the rule matches the data in the database, the rule is fired and its THEN part is executed. The **precise matching** is required (inference engine cannot cope with noisy or incomplete data).
- Neural expert systems use a trained neural network in place of the knowledge base. The input data does not have to precisely match the data that was used in network training. This ability is called **approximate reasoning**.

Rule extraction

- Neurons in the network are connected by links, each of which has a numerical weight attached to it.
- The weights in a trained neural network determine the strength or importance of the associated neuron inputs.

The neural knowledge base



If we set each input of the input layer to either +1 (true), –1 (false), or 0 (unknown), we can give a semantic interpretation for the activation of any output neuron. For example, if the object has *Wings* (+1), *Beak* (+1) and *Feathers* (+1), but does not have *Engine* (–1), then we can conclude that this object is *Bird* (+1):

$$X_{Rule\,1} = 1\cdot(-0.8) + 0\cdot(-0.2) + 1\cdot 2.2 + 1\cdot 2.8 + (-1)\cdot(-1.1) = 5.3 > 0$$

$$Y_{Rule\,1} = Y_{Bird} = +1$$

We can similarly conclude that this object is not *Plane*:

$$X_{Rule\,2} = 1\cdot(-0.7) + 0\cdot(-0.1) + 1\cdot 0.0 + 1\cdot(-1.6) + (-1)\cdot 1.9 = -4.2 < 0$$

$$Y_{Rule\,2} = Y_{Plane} = -1$$

and not *Glider*:

$$X_{Rule\,3} = 1\cdot(-0.6) + 0\cdot(-1.1) + 1\cdot(-1.0) + 1\cdot(-2.9) + (-1)\cdot(-1.3) = -4.2 < 0$$

$$Y_{Rule\,3} = Y_{Glider} = -1$$

By attaching a corresponding question to each input neuron, we can enable the system to prompt the user for initial values of the input variables:

Neuron: *Wings*

Question: Does the object have wings?

Neuron: *Tail*

Question: Does the object have a tail?

Neuron: *Beak*

Question: Does the object have a beak?

Neuron: *Feathers*

Question: Does the object have feathers?

Neuron: *Engine*

Question: Does the object have an engine?

An inference can be made if the known net weighted input to a neuron is greater than the sum of the absolute values of the weights of the unknown inputs.

$$\sum_{i=1}^{n} x_i w_i > \sum_{j=1}^{n} \left| w_j \right|$$

where $i \in$ known, $j \notin$ known and n is the number of neuron inputs.

Example:

Enter initial value for the input Feathers:

> +1

$\text{KNOWN} = 1 \cdot 2.8 = 2.8$

$\text{UNKNOWN} = |-0.8| + |-0.2| + |2.2| + |-1.1| = 4.3$

$\text{KNOWN} < \text{UNKNOWN}$

Enter initial value for the input Beak:

> +1

$\text{KNOWN} = 1 \cdot 2.8 + 1 \cdot 2.2 = 5.0$

$\text{UNKNOWN} = |-0.8| + |-0.2| + |-1.1| = 2.1$

$\text{KNOWN} > \text{UNKNOWN}$

CONCLUDE: Bird is TRUE

An example of a multi-layer knowledge base

Rule 1:
IF *a*1 AND *a*3 THEN *b*1 (0.8)

Rule 2:
IF *a*1 AND *a*4 THEN *b*1 (0.2)

Rule 3:
IF *a*2 AND *a*5 THEN *b*2 (-0.1)

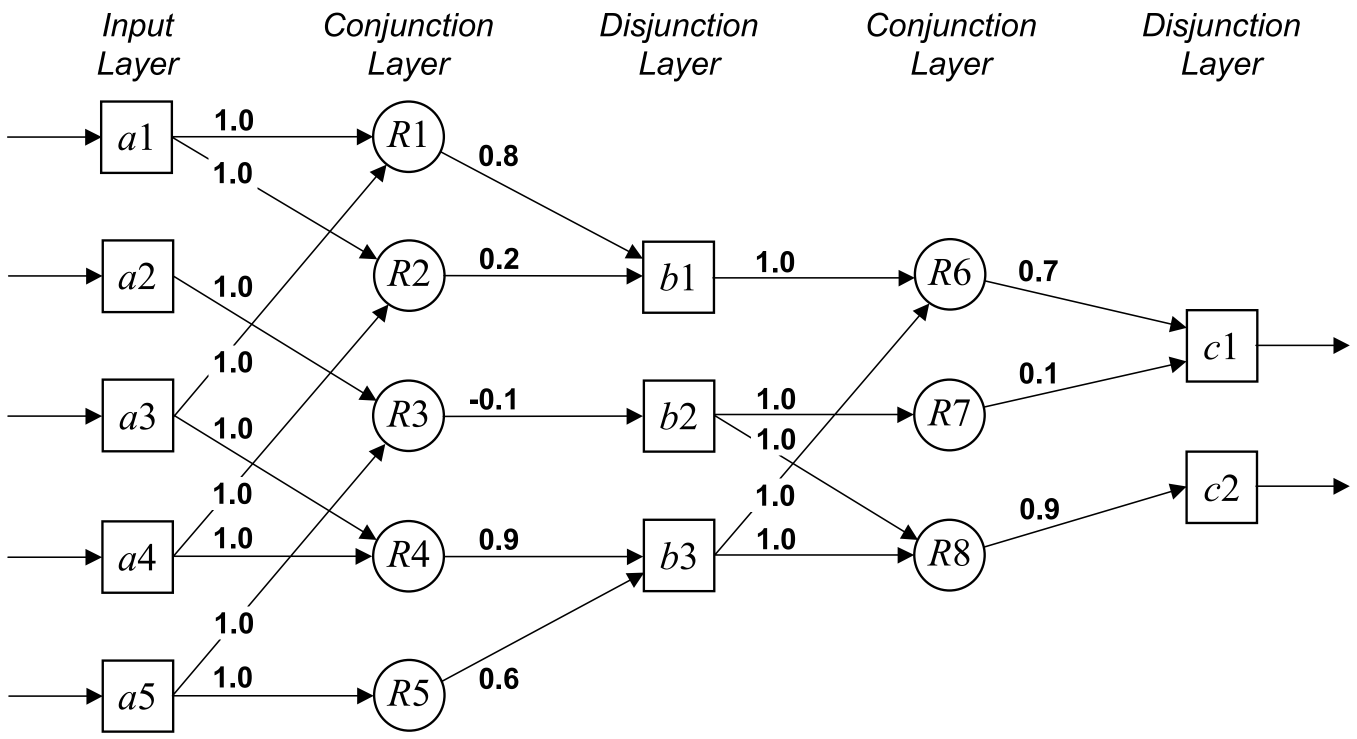
Rule 4:
IF *a*3 AND *a*4 THEN *b*3 (0.9)

Rule 5:
IF *a*5 THEN *b*3 (0.6)

Rule 6:
IF *b*1 AND *b*3 THEN *c*1 (0.7)

Rule 7:
IF *b*2 THEN *c*1 (0.1)

Rule 8:
IF *b*2 AND *b*3 THEN *c*2 (0.9)



Evolutionary neural networks

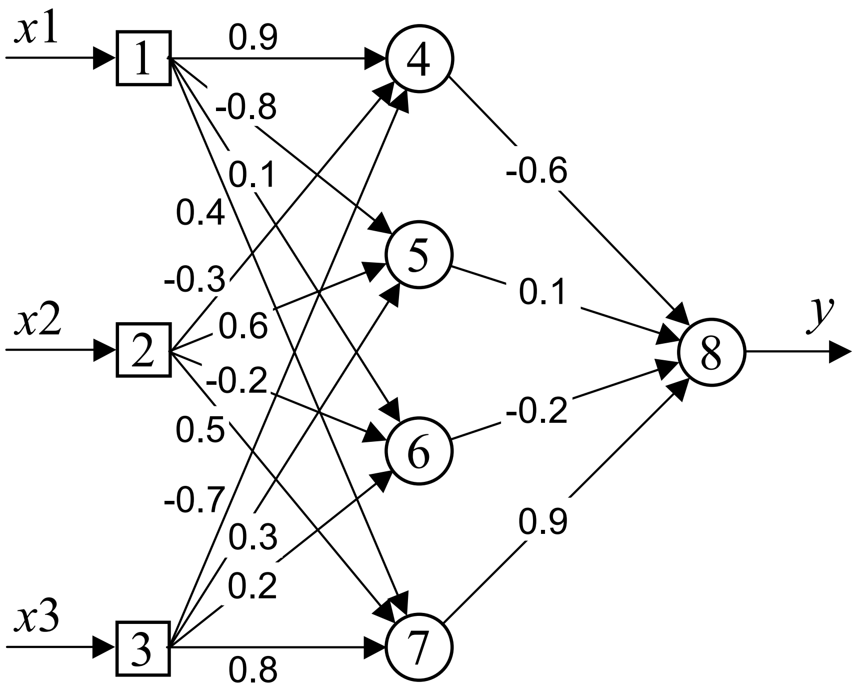
- Although neural networks are used for solving a variety of problems, they still have some limitations.
- One of the most common is associated with neural network training. The back-propagation learning algorithm cannot guarantee an optimal solution. In real-world applications, the back-propagation algorithm might converge to a set of sub-optimal weights from which it cannot escape. As a result, the neural network is often unable to find a desirable solution to a problem at hand.

- Another difficulty is related to selecting an optimal topology for the neural network. The "right" network architecture for a particular problem is often chosen by means of heuristics, and designing a neural network topology is still more art than engineering.
- Genetic algorithms are an effective optimisation technique that can guide both weight optimisation and topology selection.

Encoding a set of weights in a chromosome

From neuron: / *To neuron*:

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0.9	-0.3	-0.7	0	0	0	0	0
5	-0.8	0.6	0.3	0	0	0	0	0
6	0.1	-0.2	0.2	0	0	0	0	0
7	0.4	0.5	0.8	0	0	0	0	0
8	0	0	0	-0.6	0.1	-0.2	0.9	0



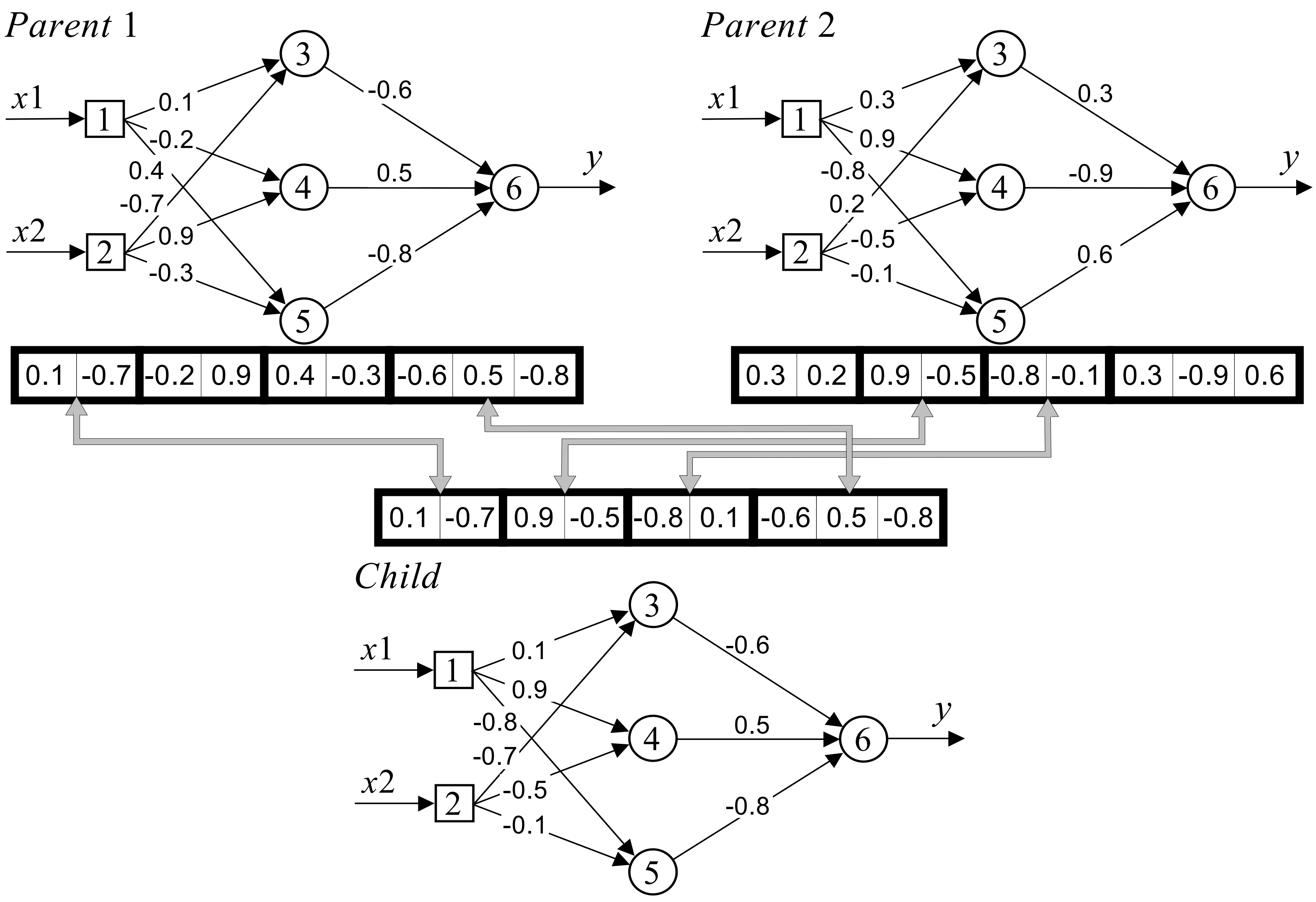
Chromosome:

0.9	-0.3	-0.7	-0.8	0.6	0.3	0.1	-0.2	0.2	0.4	0.5	0.8	-0.6	0.1	-0.2	0.9

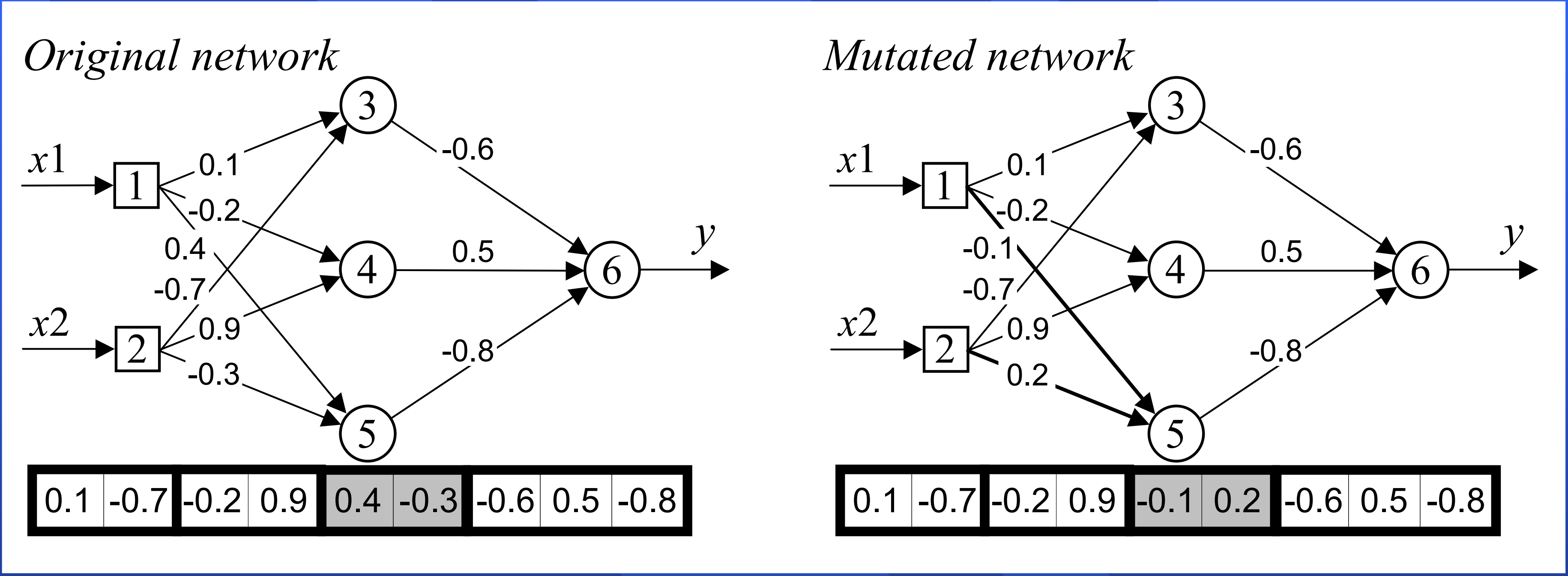
- The second step is to define a fitness function for evaluating the chromosome's performance. This function must estimate the performance of a given neural network. We can apply here a simple function defined by the sum of squared errors.
- The training set of examples is presented to the network, and the sum of squared errors is calculated. The smaller the sum, the fitter the chromosome. **The genetic algorithm attempts to find a set of weights that minimises the sum of squared errors.**

- The third step is to choose the genetic operators – crossover and mutation. A crossover operator takes two parent chromosomes and creates a single child with genetic material from both parents. Each gene in the child's chromosome is represented by the corresponding gene of the randomly selected parent.
- A mutation operator selects a gene in a chromosome and adds a small random value between –1 and 1 to each weight in this gene.

Crossover in weight optimisation



Mutation in weight optimisation



Can genetic algorithms help us in selecting the network architecture?

The architecture of the network (i.e. the number of neurons and their interconnections) often determines the success or failure of the application. Usually the network architecture is decided by trial and error; there is a great need for a method of automatically designing the architecture for a particular application. Genetic algorithms may well be suited for this task.

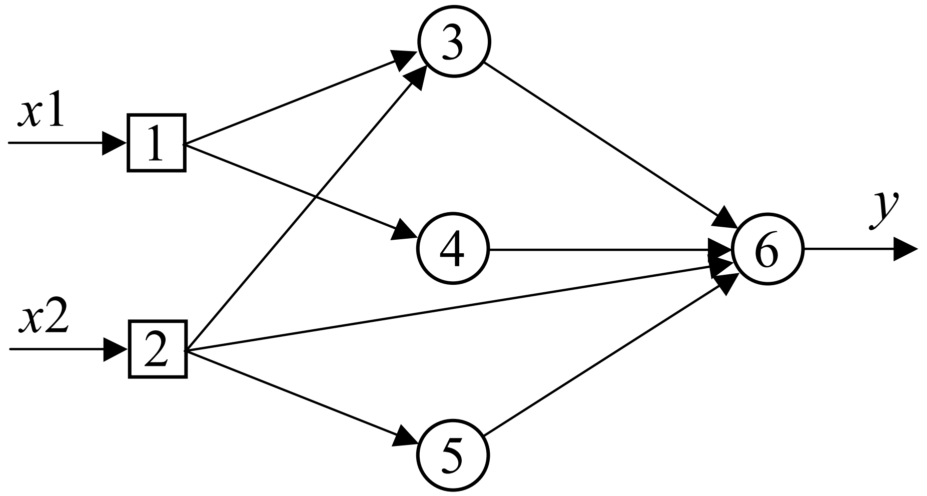
- The basic idea behind evolving a suitable network architecture is to conduct a genetic search in a population of possible architectures.
- We must first choose a method of encoding a network's architecture into a chromosome.

Encoding the network architecture

- The connection topology of a neural network can be represented by a square connectivity matrix.
- Each entry in the matrix defines the type of connection from one neuron (column) to another (row), where 0 means no connection and 1 denotes connection for which the weight can be changed through learning.
- To transform the connectivity matrix into a chromosome, we need only to string the rows of the matrix together.

Encoding of the network topology

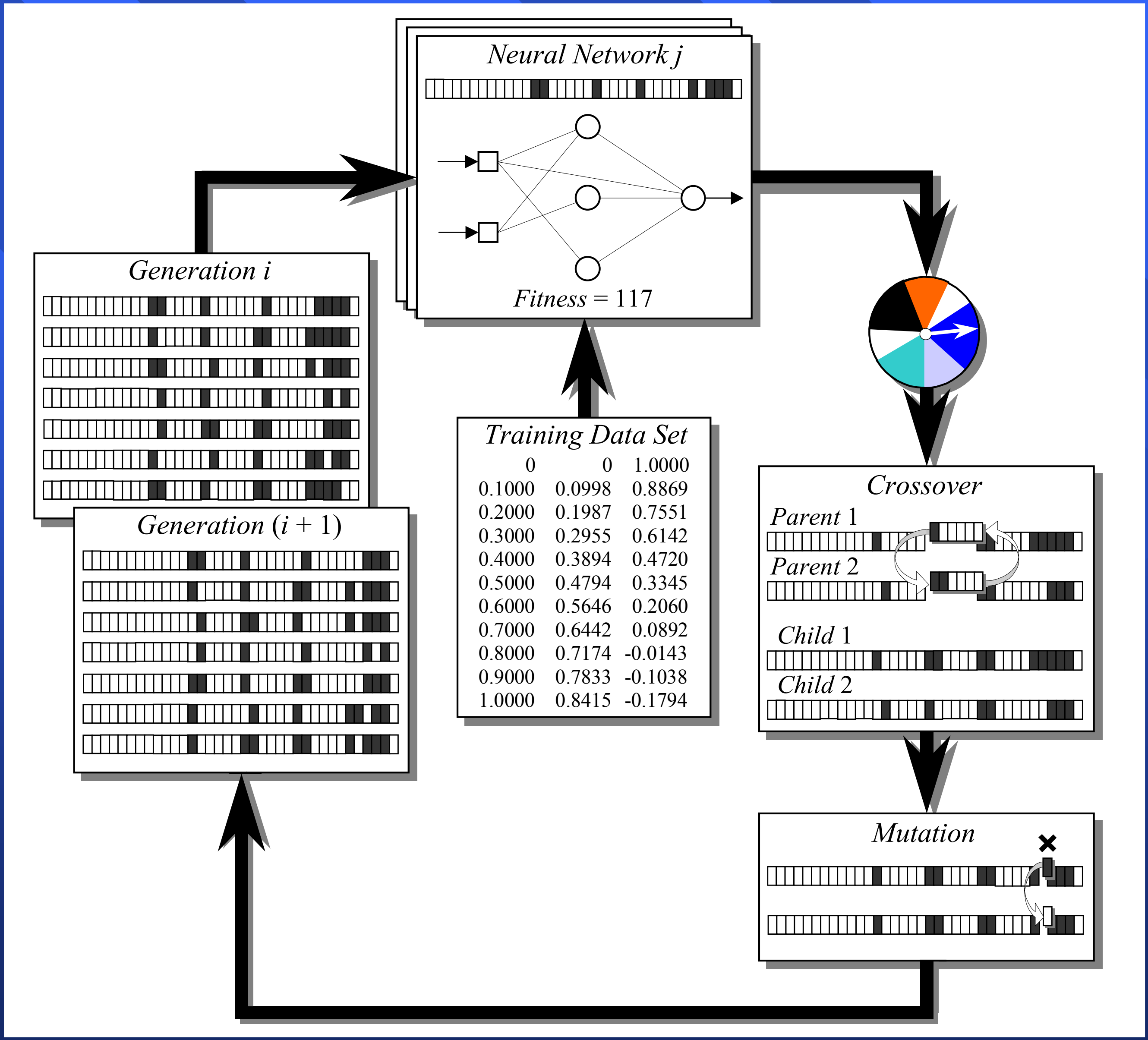
From neuron:		1	2	3	4	5	6
To neuron:	1	0	0	0	0	0	0
	2	0	0	0	0	0	0
	3	1	1	0	0	0	0
	4	1	0	0	0	0	0
	5	0	1	0	0	0	0
	6	0	1	1	1	1	0



Chromosome:

0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0

The cycle of evolving a neural network topology



Training Data Set

0	0	1.0000
0.1000	0.0998	0.8869
0.2000	0.1987	0.7551
0.3000	0.2955	0.6142
0.4000	0.3894	0.4720
0.5000	0.4794	0.3345
0.6000	0.5646	0.2060
0.7000	0.6442	0.0892
0.8000	0.7174	-0.0143
0.9000	0.7833	-0.1038
1.0000	0.8415	-0.1794