Lecture 11

Hybrid intelligent systems:

Neural expert systems and neuro-fuzzy systems

- Introduction
- Neural expert systems
- Neuro-fuzzy systems
- ANFIS: Adaptive Neuro-Fuzzy Inference System
- Summary

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

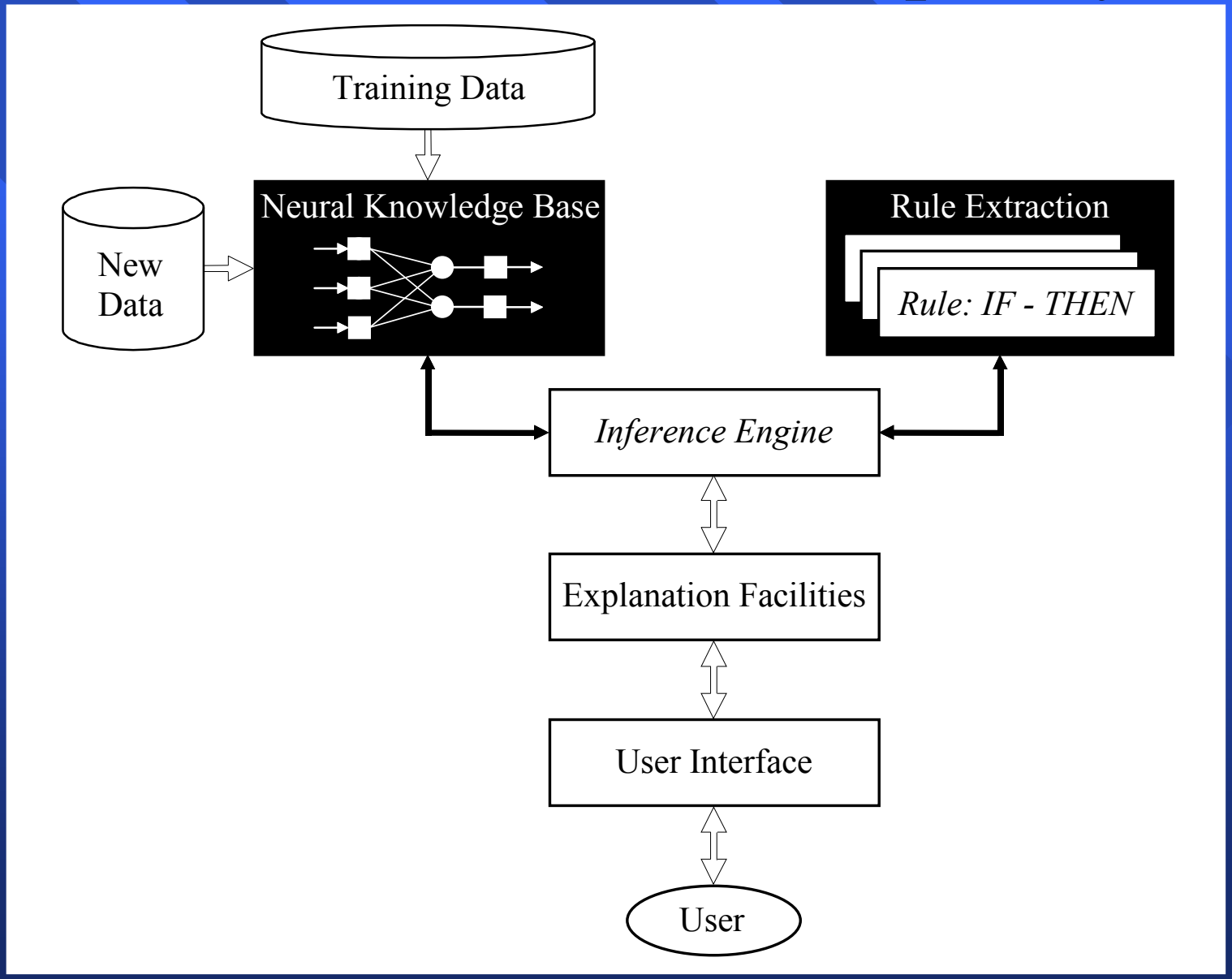
- Expert systems rely on logical inferences and decision trees and focus on modelling human reasoning. Neural networks rely on parallel data processing and focus on modelling a human brain.
- Expert systems treat the brain as a black-box. Neural networks look at its structure and functions, particularly 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**.

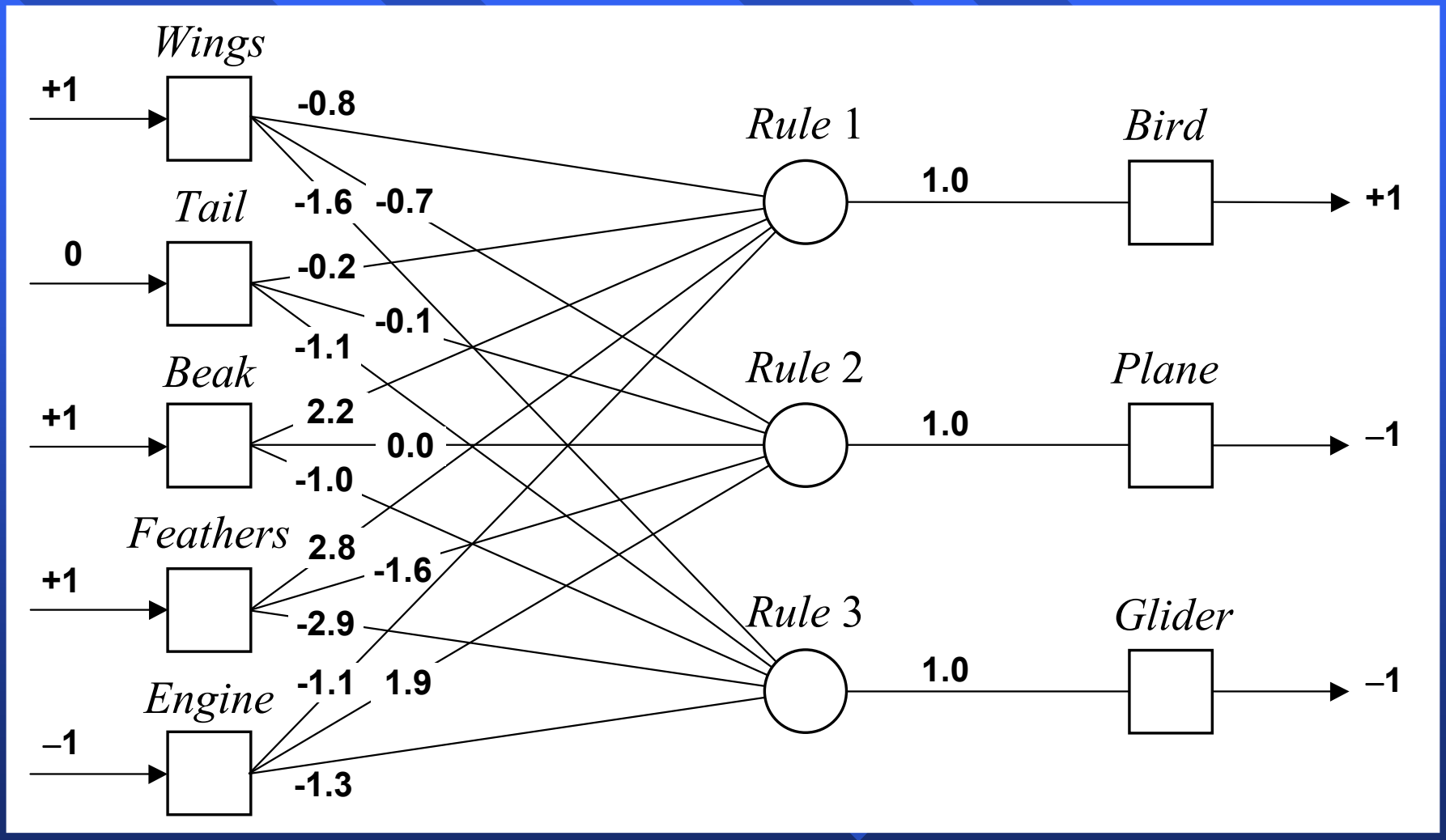
Approximate reasoning

- 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

KNOWN = $1 \cdot 2.8 = 2.8$

UNKNOWN = $|-0.8| + |-0.2| + |2.2| + |-1.1| = 4.3$

KNOWN < UNKNOWN

Enter initial value for the input Beak:

> +1

KNOWN = $1 \cdot 2.8 + 1 \cdot 2.2 = 5.0$

UNKNOWN = $|-0.8| + |-0.2| + |-1.1| = 2.1$

KNOWN > 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)

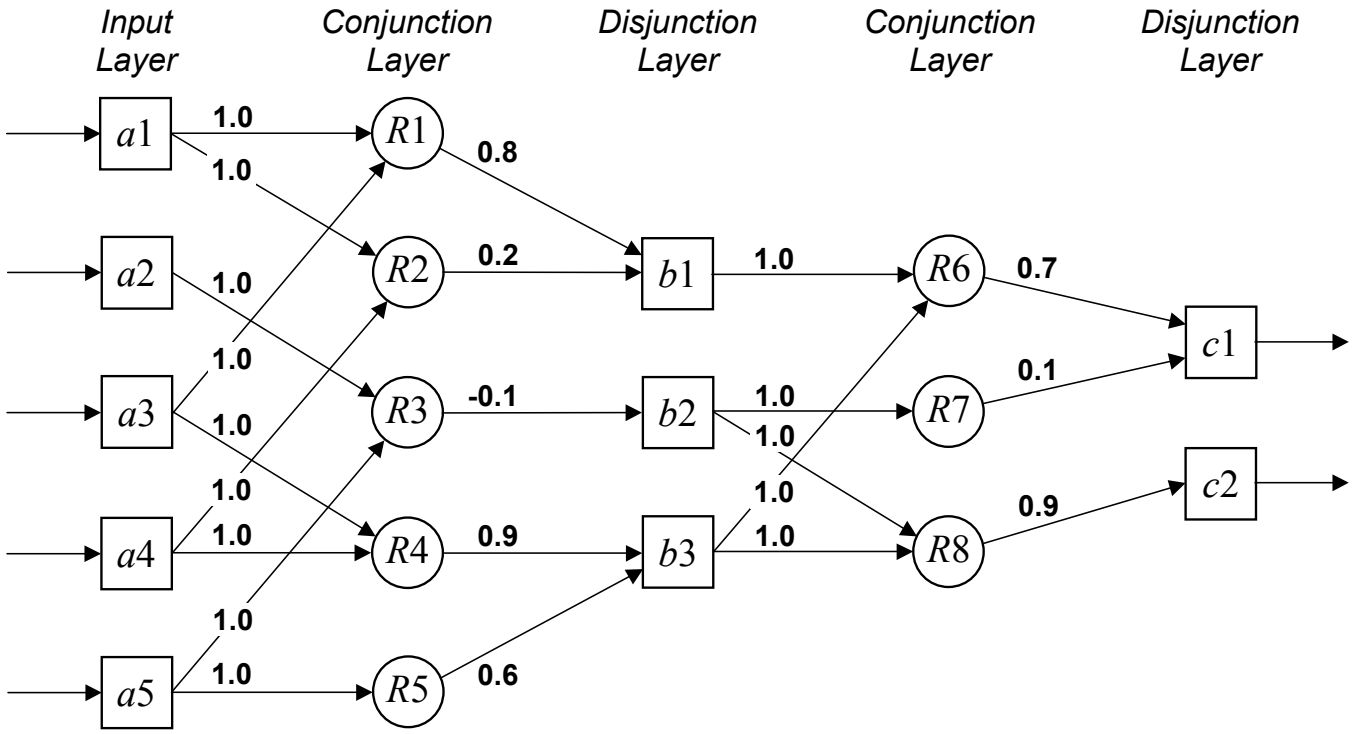
Rule 4:
IF *a*3 AND *a*4 THEN *b*3 (0.9)

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)



Neuro-fuzzy systems

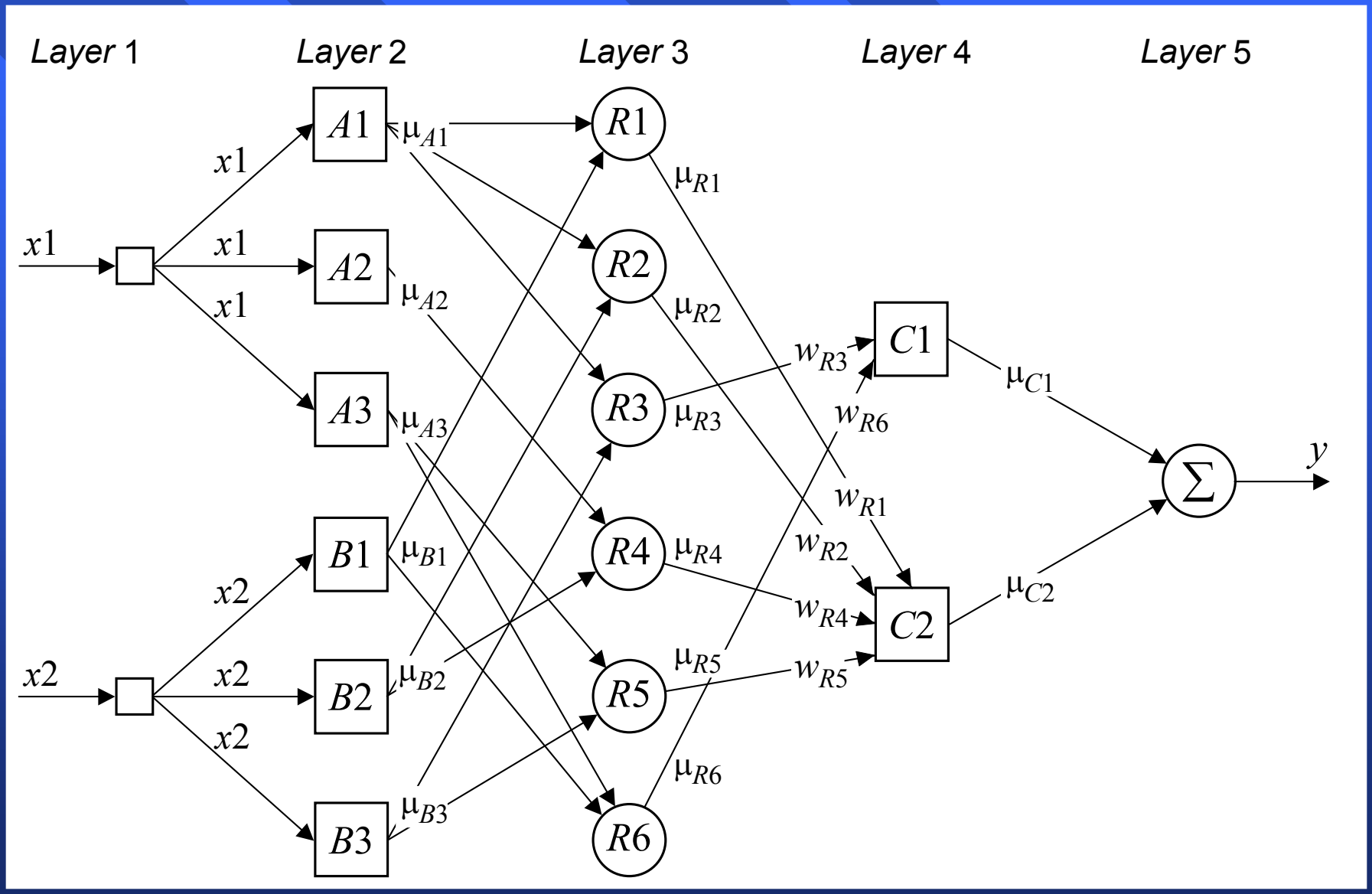
- Fuzzy logic and neural networks are natural complementary tools in building intelligent systems. While neural networks are low-level computational structures that perform well when dealing with raw data, fuzzy logic deals with reasoning on a higher level, using linguistic information acquired from domain experts. However, fuzzy systems lack the ability to learn and cannot adjust themselves to a new environment. On the other hand, although neural networks can learn, they are opaque to the user.

- Integrated neuro-fuzzy systems can combine the parallel computation and learning abilities of neural networks with the human-like knowledge representation and explanation abilities of fuzzy systems. As a result, neural networks become more transparent, while fuzzy systems become capable of learning.

- A neuro-fuzzy system is a neural network which is functionally equivalent to a fuzzy inference model. It can be trained to develop IF-THEN fuzzy rules and determine membership functions for input and output variables of the system. Expert knowledge can be incorporated into the structure of the neuro-fuzzy system. At the same time, the connectionist structure avoids fuzzy inference, which entails a substantial computational burden.

- The structure of a neuro-fuzzy system is similar to a multi-layer neural network. In general, a neuro-fuzzy system has input and output layers, and three hidden layers that represent membership functions and fuzzy rules.

Neuro-fuzzy system



Each layer in the neuro-fuzzy system is associated with a particular step in the fuzzy inference process.

Layer 1 is the **input layer**. Each neuron in this layer transmits external crisp signals directly to the next layer. That is,

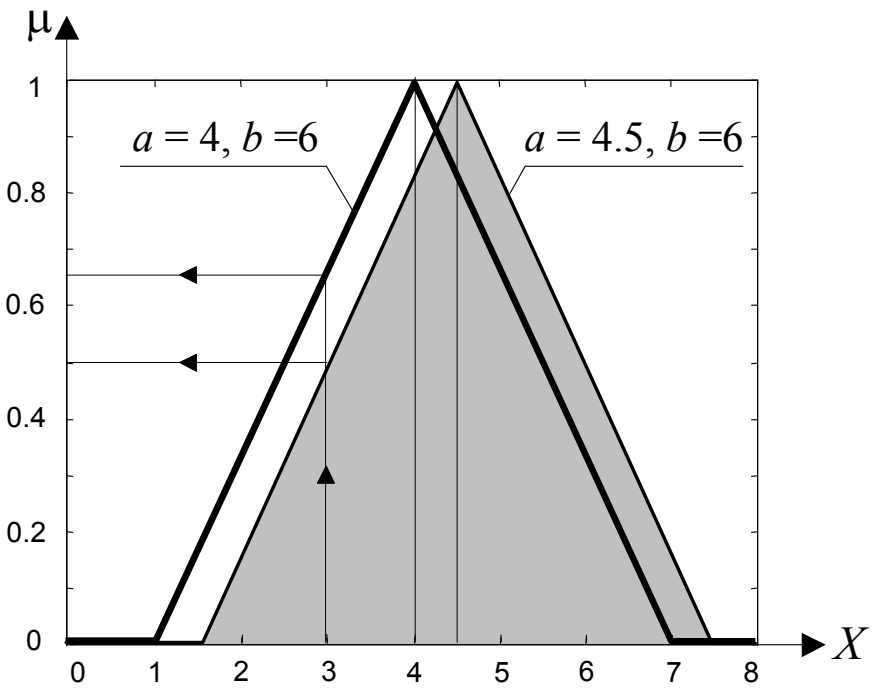
$$y_i^{(1)} = x_i^{(1)}$$

Layer 2 is the **fuzzification layer**. Neurons in this layer represent fuzzy sets used in the antecedents of fuzzy rules. A fuzzification neuron receives a crisp input and determines the degree to which this input belongs to the neuron's fuzzy set.

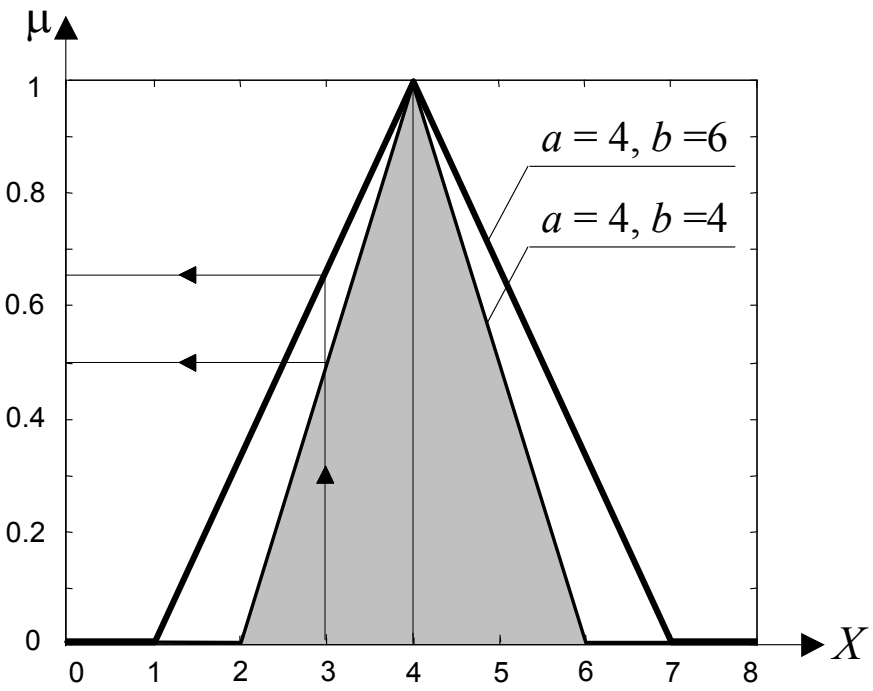
The activation function of a membership neuron is set to the function that specifies the neuron's fuzzy set. We use triangular sets, and therefore, the activation functions for the neurons in *Layer* 2 are set to the **triangular membership functions**. A triangular membership function can be specified by two parameters $\{a, b\}$ as follows:

$$y_i^{(2)} = \begin{cases} 0, & \text{if } x_i^{(2)} \le a - \dfrac{b}{2} \\ 1 - \dfrac{2\left|x_i^{(2)} - a\right|}{b}, & \text{if } a - \dfrac{b}{2} < x_i^{(2)} < a + \dfrac{b}{2} \\ 0, & \text{if } x_i^{(2)} \ge a + \dfrac{b}{2} \end{cases}$$

Triangular activation functions



(*a*) Effect of parameter *a*.



(*b*) Effect of parameter *b*.

Layer 3 is the **fuzzy rule layer**. Each neuron in this layer corresponds to a single fuzzy rule. A fuzzy rule neuron receives inputs from the fuzzification neurons that represent fuzzy sets in the rule antecedents. For instance, neuron $R1$, which corresponds to *Rule* 1, receives inputs from neurons $A1$ and $B1$.

In a neuro-fuzzy system, intersection can be implemented by the **product operator**. Thus, the output of neuron i in *Layer* 3 is obtained as:

$$y_i^{(3)} = x_{1i}^{(3)} \times x_{2i}^{(3)} \times \ldots \times x_{ki}^{(3)}$$

$$y_{R1}^{(3)} = \mu_{A1} \times \mu_{B1} = \mu_{R1}$$

Layer 4 is the **output membership layer**. Neurons in this layer represent fuzzy sets used in the consequent of fuzzy rules.

An output membership neuron combines all its inputs by using the fuzzy operation **union**. This operation can be implemented by the **probabilistic OR**. That is,

$$y_i^{(4)} = x_{1i}^{(4)} \oplus x_{2i}^{(4)} \oplus \ldots \oplus x_{li}^{(4)}$$

$$y_{C1}^{(4)} = \mu_{R3} \oplus \mu_{R6} = \mu_{C1}$$

The value of μ_{C1} represents the integrated firing strength of fuzzy rule neurons *R*3 and *R*6.

Layer 5 is the **defuzzification layer**. Each neuron in this layer represents a single output of the neuro-fuzzy system. It takes the output fuzzy sets clipped by the respective integrated firing strengths and combines them into a single fuzzy set.

Neuro-fuzzy systems can apply standard defuzzification methods, including the centroid technique.

We will use the **sum-product composition** method.

The sum-product composition calculates the crisp output as the weighted average of the centroids of all output membership functions. For example, the weighted average of the centroids of the clipped fuzzy sets $C1$ and $C2$ is calculated as,

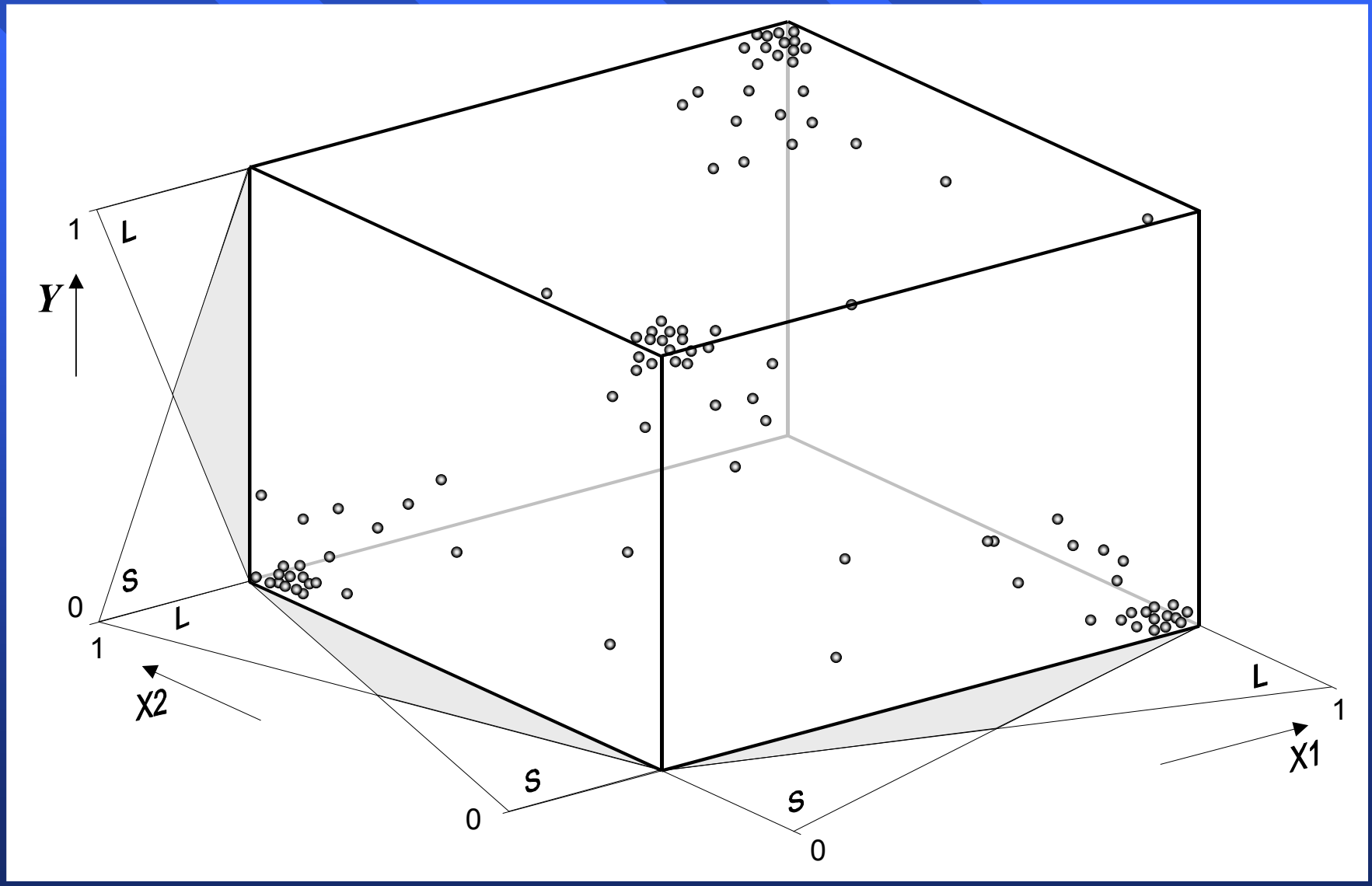
$$y = \frac{\mu_{C1} \times a_{C1} \times b_{C1} + \mu_{C2} \times a_{C2} \times b_{C2}}{\mu_{C1} \times b_{C1} + \mu_{C2} \times b_{C2}}$$

How does a neuro-fuzzy system learn?

A neuro-fuzzy system is essentially a multi-layer neural network, and thus it can apply standard learning algorithms developed for neural networks, including the back-propagation algorithm.

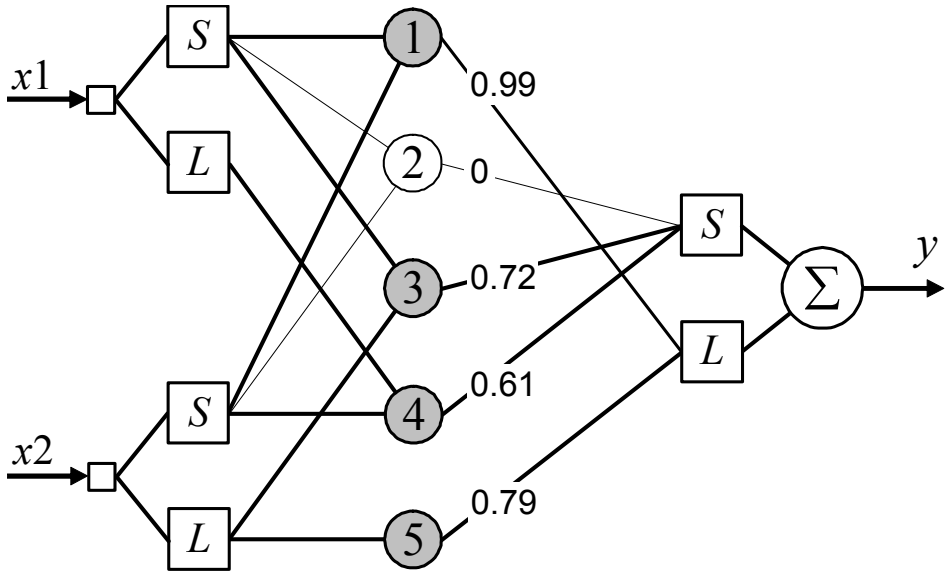
- When a training input-output example is presented to the system, the back-propagation algorithm computes the system output and compares it with the desired output of the training example. The error is propagated backwards through the network from the output layer to the input layer. The neuron activation functions are modified as the error is propagated. To determine the necessary modifications, the back-propagation algorithm differentiates the activation functions of the neurons.
- Let us demonstrate how a neuro-fuzzy system works on a simple example.

Training patterns

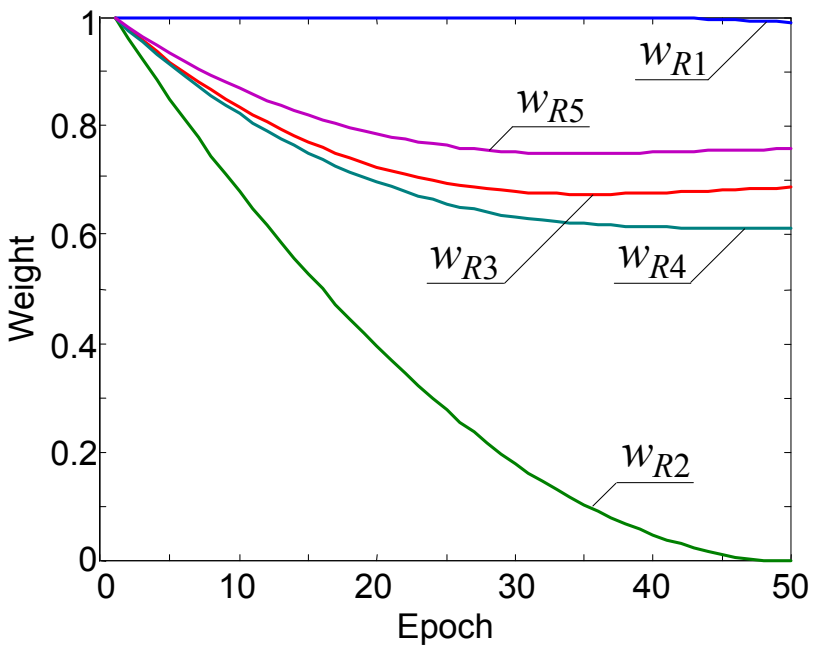


The data set is used for training the five-rule neuro-fuzzy system shown below.

Five-rule neuro-fuzzy system



(*a*) Five-rule system.

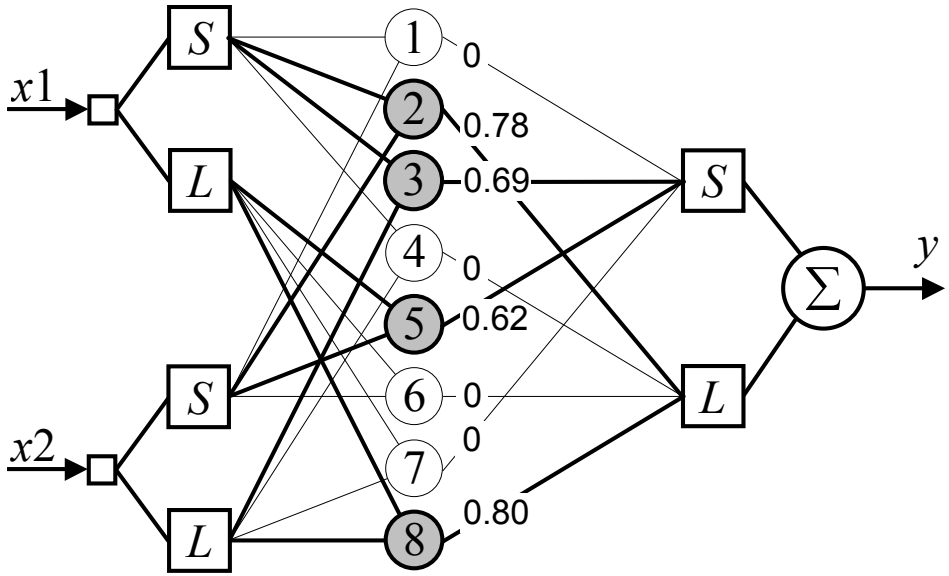


(*b*) Training for 50 epochs.

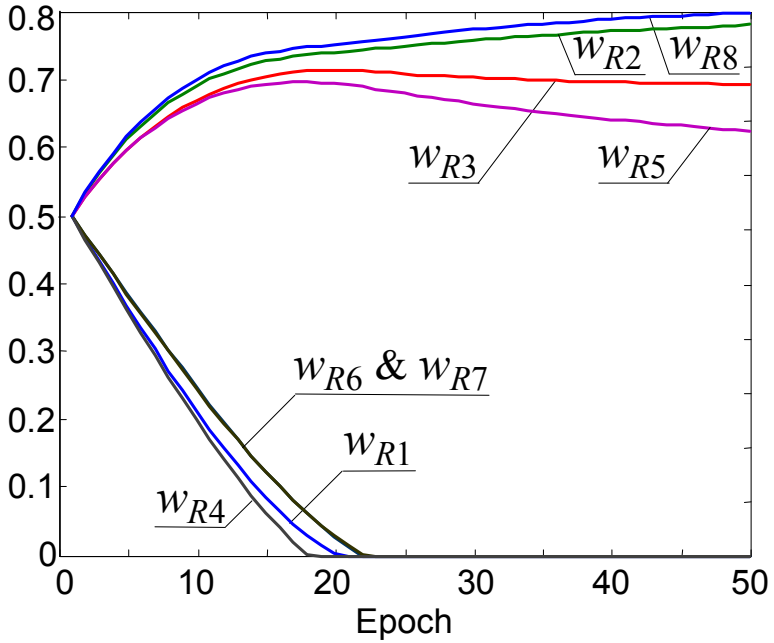
- Suppose that fuzzy IF-THEN rules incorporated into the system structure are supplied by a domain expert. *Prior* or existing knowledge can dramatically expedite the system training.
- Besides, if the quality of training data is poor, the expert knowledge may be the only way to come to a solution at all. However, experts do occasionally make mistakes, and thus some rules used in a neuro-fuzzy system may be false or redundant. Therefore, a neuro-fuzzy system should also be capable of identifying bad rules.

- Given input and output linguistic values, a neuro-fuzzy system can automatically generate a complete set of fuzzy IF-THEN rules.
- Let us create the system for the XOR example. This system consists of $2^2 \times 2 = 8$ rules. Because expert knowledge is not embodied in the system this time, we set all initial weights between *Layer* 3 and *Layer* 4 to 0.5.
- After training we can eliminate all rules whose certainty factors are less than some sufficiently small number, say 0.1. As a result, we obtain the same set of four fuzzy IF-THEN rules that represents the XOR operation.

Eight-rule neuro-fuzzy system



(*a*) Eight-rule system.



(*b*) Training for 50 epochs.

Neuro-fuzzy systems: summary

- The combination of fuzzy logic and neural networks constitutes a powerful means for designing intelligent systems.
- Domain knowledge can be put into a neuro-fuzzy system by human experts in the form of linguistic variables and fuzzy rules.
- When a representative set of examples is available, a neuro-fuzzy system can automatically transform it into a robust set of fuzzy IF-THEN rules, and thereby reduce our dependence on expert knowledge when building intelligent systems.

ANFIS:
Adaptive Neuro-Fuzzy Inference System

The Sugeno fuzzy model was proposed for generating fuzzy rules from a given input-output data set. A typical Sugeno fuzzy rule is expressed in the following form:

IF x_1 is A_1
AND x_2 is A_2
. . . . .
AND x_m is A_m
THEN $y = f(x_1, x_2, \ldots, x_m)$

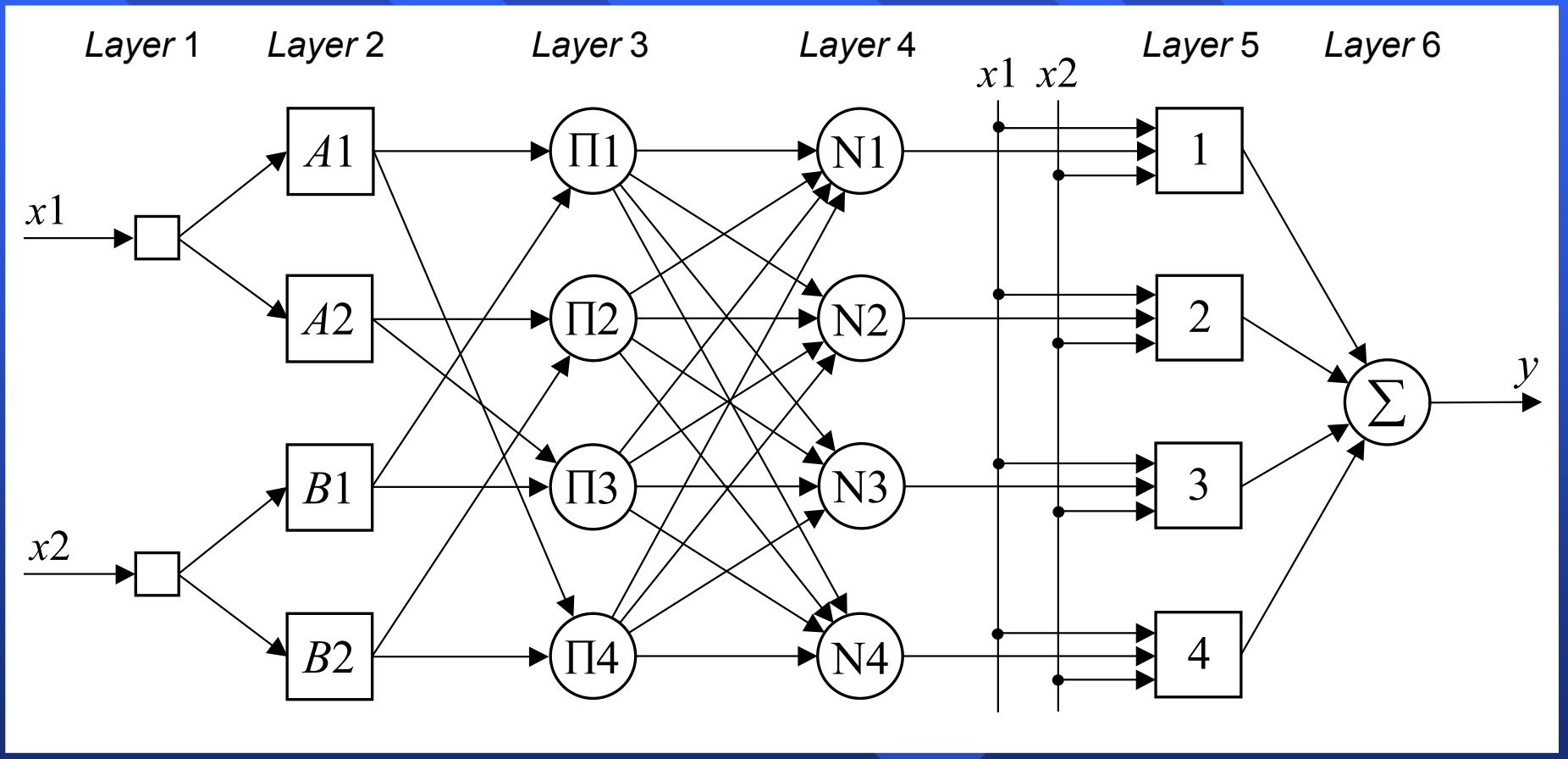
where $x_1, x_2, \ldots, x_m$ are input variables; $A_1, A_2, \ldots, A_m$ are fuzzy sets.

- When y is a constant, we obtain a **zero-order Sugeno fuzzy model** in which the consequent of a rule is specified by a singleton.
- When y is a first-order polynomial, i.e.

$$y = k_0 + k_1 x_1 + k_2 x_2 + \ldots + k_m x_m$$

we obtain a **first-order Sugeno fuzzy model**.

Adaptive Neuro-Fuzzy Inference System



Layer 1 is the **input layer**. Neurons in this layer simply pass external crisp signals to *Layer* 2.

Layer 2 is the **fuzzification layer**. Neurons in this layer perform fuzzification. In Jang's model, fuzzification neurons have a **bell activation function**.

Layer 3 is the **rule layer**. Each neuron in this layer corresponds to a single Sugeno-type fuzzy rule. A rule neuron receives inputs from the respective fuzzification neurons and calculates the firing strength of the rule it represents. In an ANFIS, the conjunction of the rule antecedents is evaluated by the operator **product**. Thus, the output of neuron *i* in *Layer* 3 is obtained as,

$$y_i^{(3)} = \prod_{j=1}^{k} x_{ji}^{(3)}$$

$$y_{\Pi 1}^{(3)} = \mu_{A1} \times \mu_{B1} = \mu_1,$$

where the value of μ_1 represents the firing strength, or the truth value, of *Rule* 1.

Layer 4 is the **normalisation layer**. Each neuron in this layer receives inputs from all neurons in the rule layer, and calculates the **normalised firing strength** of a given rule.

The normalised firing strength is the ratio of the firing strength of a given rule to the sum of firing strengths of all rules. It represents the contribution of a given rule to the final result. Thus, the output of neuron *i* in *Layer* 4 is determined as,

$$y_i^{(4)} = \frac{x_{ii}^{(4)}}{\sum_{j=1}^{n} x_{ji}^{(4)}} = \frac{\mu_i}{\sum_{j=1}^{n} \mu_j} = \bar{\mu}_i$$

$$y_{\mathrm{N1}}^{(4)} = \frac{\mu_1}{\mu_1 + \mu_2 + \mu_3 + \mu_4} = \bar{\mu}_1$$

Layer 5 is the **defuzzification layer**. Each neuron in this layer is connected to the respective normalisation neuron, and also receives initial inputs, x_1 and x_2. A defuzzification neuron calculates the weighted consequent value of a given rule as,

$$y_i^{(5)} = x_i^{(5)} \left[k_{i0} + k_{i1}\, x1 + k_{i2}\, x2\right] = \bar{\mu}_i \left[k_{i0} + k_{i1}\, x1 + k_{i2}\, x2\right]$$

where is the input and is the output of defuzzification neuron i in *Layer* 5, and k_{i0}, k_{i1} and k_{i2} is a set of consequent parameters of rule i.

Layer 6 is represented by a single **summation neuron**. This neuron calculates the sum of outputs of all defuzzification neurons and produces the overall ANFIS output, y,

$$y = \sum_{i=1}^{n} x_i^{(6)} = \sum_{i=1}^{n} \bar{\mu}_i \left[k_{i0} + k_{i1}\, x1 + k_{i2}\, x2 \right]$$

Can an ANFIS deal with problems where we do not have any prior knowledge of the rule consequent parameters?

It is not necessary to have any prior knowledge of rule consequent parameters. An ANFIS learns these parameters and tunes membership functions.

Learning in the ANFIS model

- An ANFIS uses a hybrid learning algorithm that combines the least-squares estimator and the gradient descent method.
- In the ANFIS training algorithm, each epoch is composed from a forward pass and a backward pass. In the forward pass, a training set of input patterns (an input vector) is presented to the ANFIS, neuron outputs are calculated on the layer-by-layer basis, and rule consequent parameters are identified.

- The rule consequent parameters are identified by the least-squares estimator. In the Sugeno-style fuzzy inference, an output, y, is a linear function. Thus, given the values of the membership parameters and a training set of P input-output patterns, we can form P linear equations in terms of the consequent parameters as:

$$\begin{cases} y_d(1) = \bar{\mu}_1(1) f_1(1) + \bar{\mu}_2(1) f_2(1) + \ldots + \bar{\mu}_n(1) f_n(1) \\ y_d(2) = \bar{\mu}_1(2) f_1(2) + \bar{\mu}_2(2) f_2(2) + \ldots + \bar{\mu}_n(2) f_n(2) \\ \vdots \\ y_d(p) = \bar{\mu}_1(p) f_1(p) + \bar{\mu}_2(p) f_2(p) + \ldots + \bar{\mu}_n(p) f_n(p) \\ \vdots \\ y_d(P) = \bar{\mu}_1(P) f_1(P) + \bar{\mu}_2(P) f_2(P) + \ldots + \bar{\mu}_n(P) f_n(P) \end{cases}$$

In the matrix notation, we have

$$\boldsymbol{y}_d = \mathbf{A}\,\boldsymbol{k},$$

where $\boldsymbol{y}_d$ is a $P \times 1$ desired output vector,

$$\boldsymbol{y}_d = \begin{bmatrix} y_d(1) \\ y_d(2) \\ \vdots \\ y_d(p) \\ \vdots \\ y_d(P) \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} \bar{\mu}_1(1) & \bar{\mu}_1(1)x_1(1) & \cdots & \bar{\mu}_1(1)x_m(1) & \cdots & \bar{\mu}_n(1) & \bar{\mu}_n(1)x_1(1) & \cdots & \bar{\mu}_n(1)x_m(1) \\ \bar{\mu}_1(2) & \bar{\mu}_1(1)x_1(2) & \cdots & \bar{\mu}_1(2)x_m(2) & \cdots & \bar{\mu}_n(2) & \bar{\mu}_n(2)x_1(2) & \cdots & \bar{\mu}_n(2)x_m(2) \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ \bar{\mu}_1(p) & \bar{\mu}_1(p)x_1(p) & \cdots & \bar{\mu}_1(p)x_m(p) & \cdots & \bar{\mu}_n(p) & \bar{\mu}_n(p)x_1(p) & \cdots & \bar{\mu}_n(p)x_m(p) \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ \mu_1(P) & \bar{\mu}_1(P)x_1(P) & & \bar{\mu}_1(P)x_m(P) & & \bar{\mu}_n(P) & \bar{\mu}_n(P)x_1(P) & & \bar{\mu}_n(P)x_m(P) \end{bmatrix}$$

and $\boldsymbol{k}$ is an $n\,(1 + m) \times 1$ vector of unknown consequent parameters,

$$\boldsymbol{k} = [k_{10}\ \ k_{11}\ \ k_{12} \ldots k_{1m}\ \ k_{20}\ \ k_{21}\ \ k_{22} \ldots k_{2m} \ldots k_{n0}\ \ k_{n1}\ \ k_{n2} \ldots k_{n\,m}]^T$$

- As soon as the rule consequent parameters are established, we compute an actual network output vector, y, and determine the error vector, e

  $$e = y_d - y$$

- In the backward pass, the back-propagation algorithm is applied. The error signals are propagated back, and the antecedent parameters are updated according to the chain rule.

In the ANFIS training algorithm suggested by Jang, both antecedent parameters and consequent parameters are optimised. In the forward pass, the consequent parameters are adjusted while the antecedent parameters remain fixed. In the backward pass, the antecedent parameters are tuned while the consequent parameters are kept fixed.

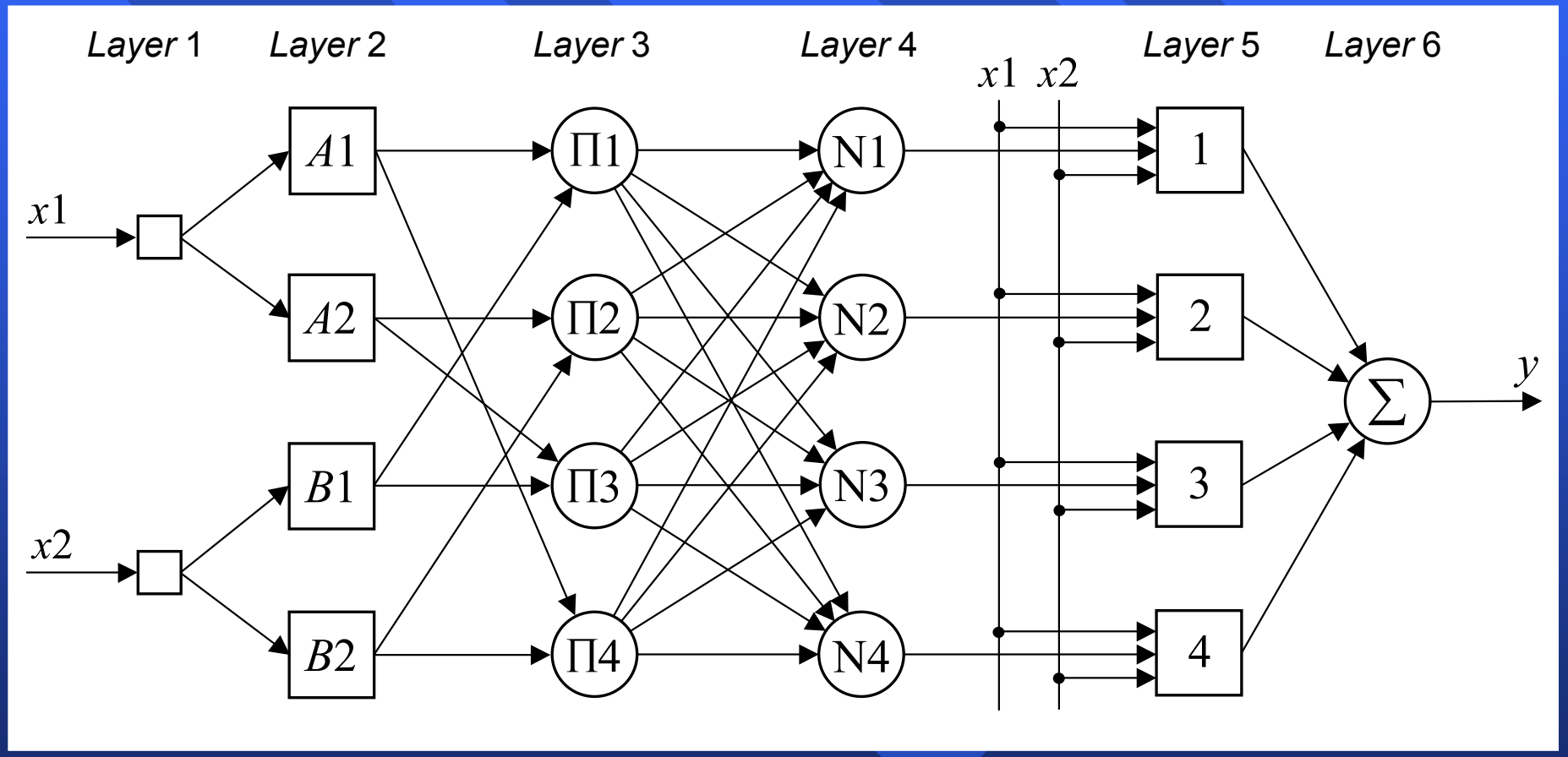
Function approximation using the ANFIS model

- In this example, an ANFIS is used to follow a trajectory of the non-linear function defined by the equation

$$y = \frac{\cos(2\,x1)}{e^{x2}}$$

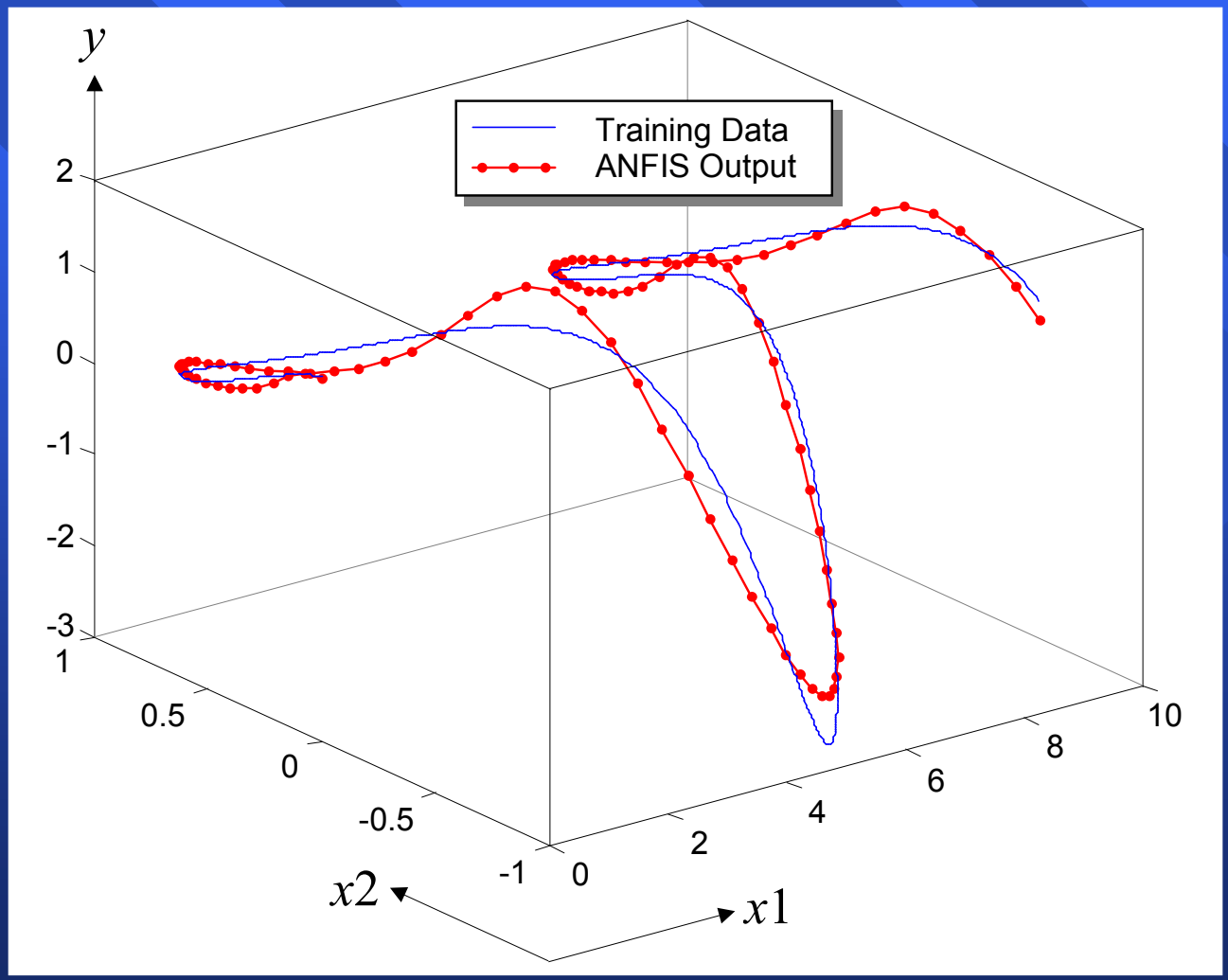
- First, we choose an appropriate architecture for the ANFIS. An ANFIS must have two inputs – $x1$ and $x2$ – and one output – y.
- Thus, in our example, the ANFIS is defined by four rules, and has the structure shown below.

An ANFIS model with four rules

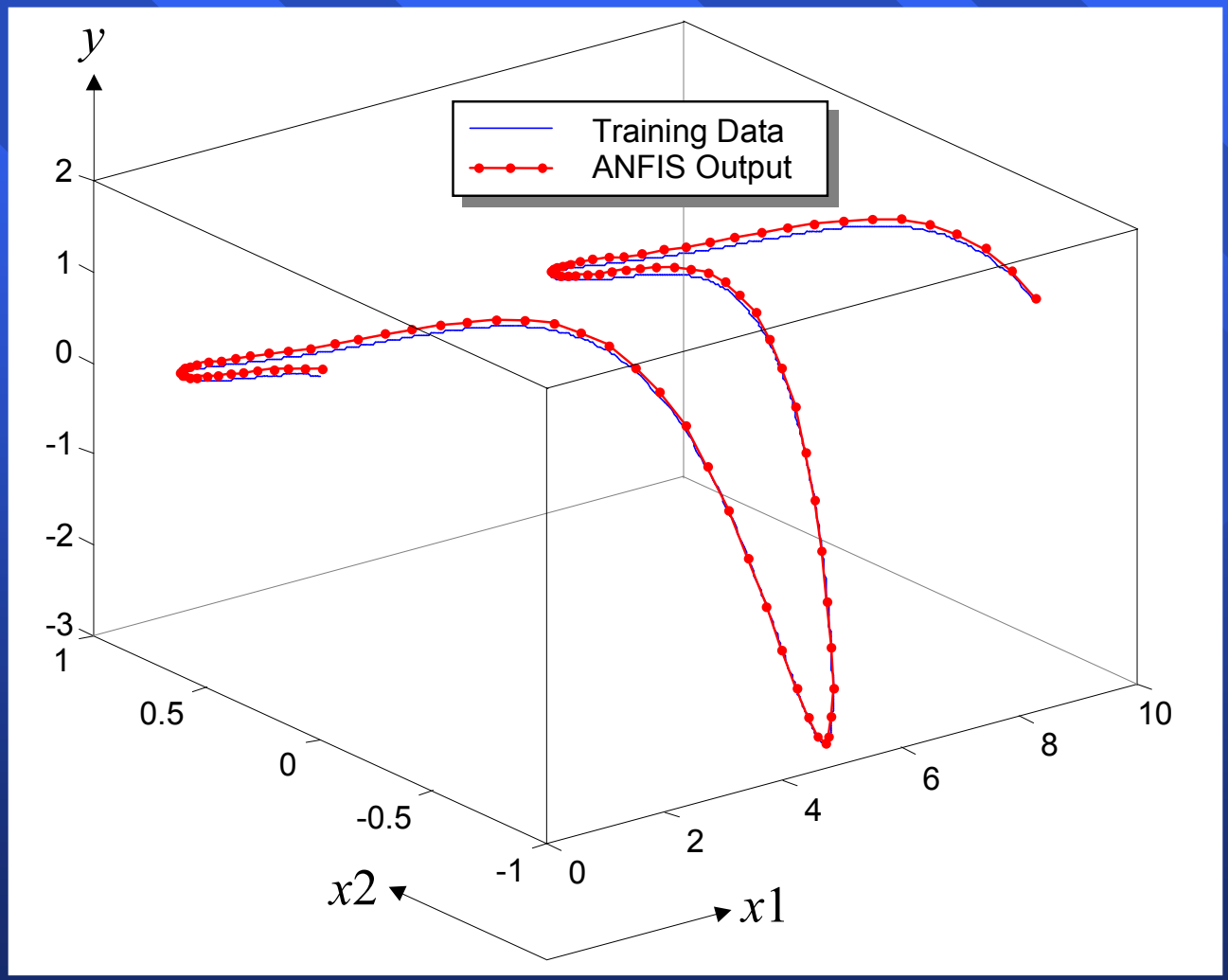


- The ANFIS training data includes 101 training samples. They are represented by a 101 × 3 matrix [$x1$ $x2$ y_d], where $x1$ and $x2$ are input vectors, and y_d is a desired output vector.
- The first input vector, $x1$, starts at 0, increments by 0.1 and ends at 10.
- The second input vector, $x2$, is created by taking sin from each element of vector $x1$, with elements of the desired output vector, y_d, determined by the function equation.

Learning in an ANFIS with two membership functions assigned to each input (one epoch)

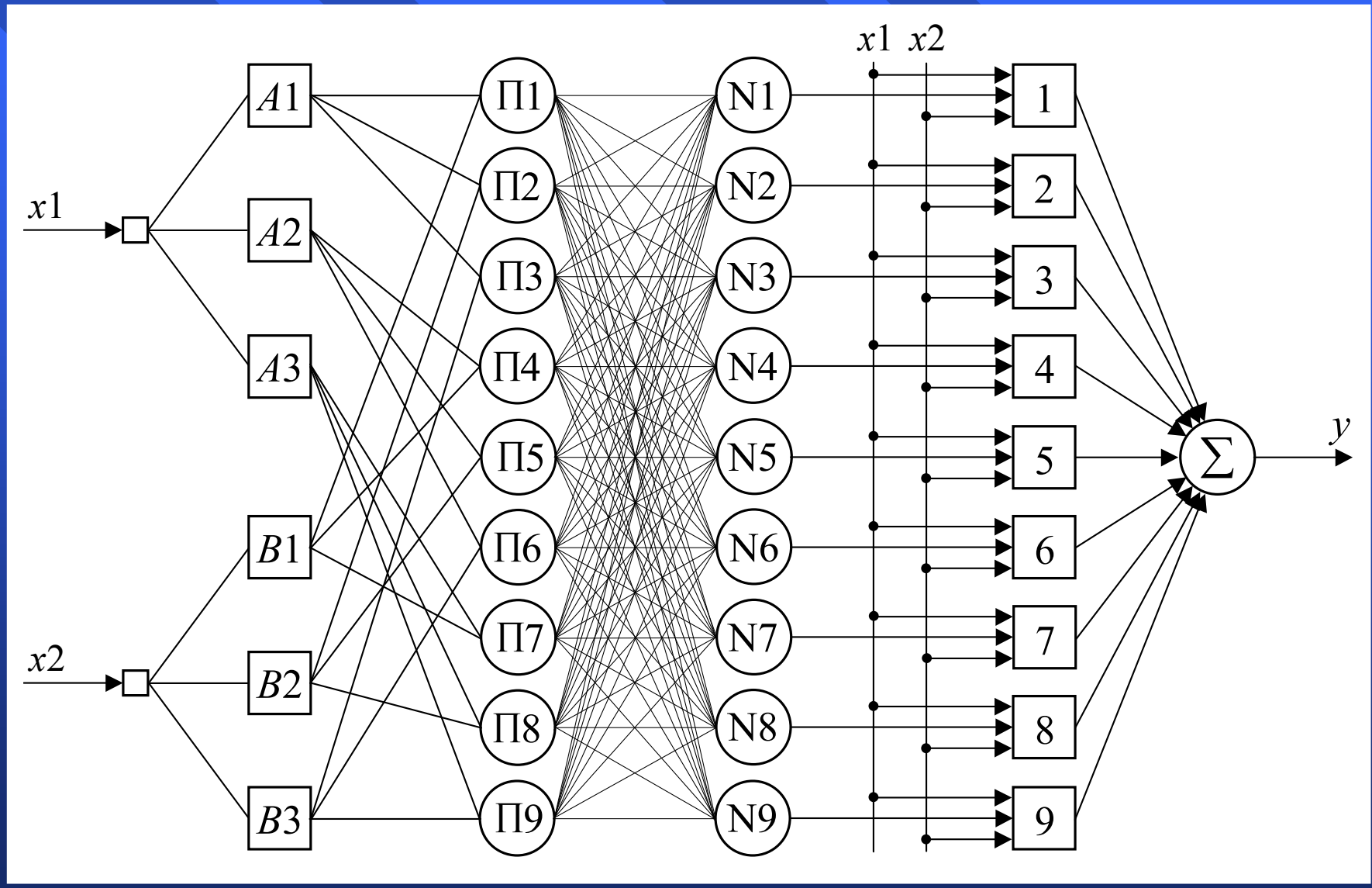


Learning in an ANFIS with two membership functions assigned to each input (100 epochs)

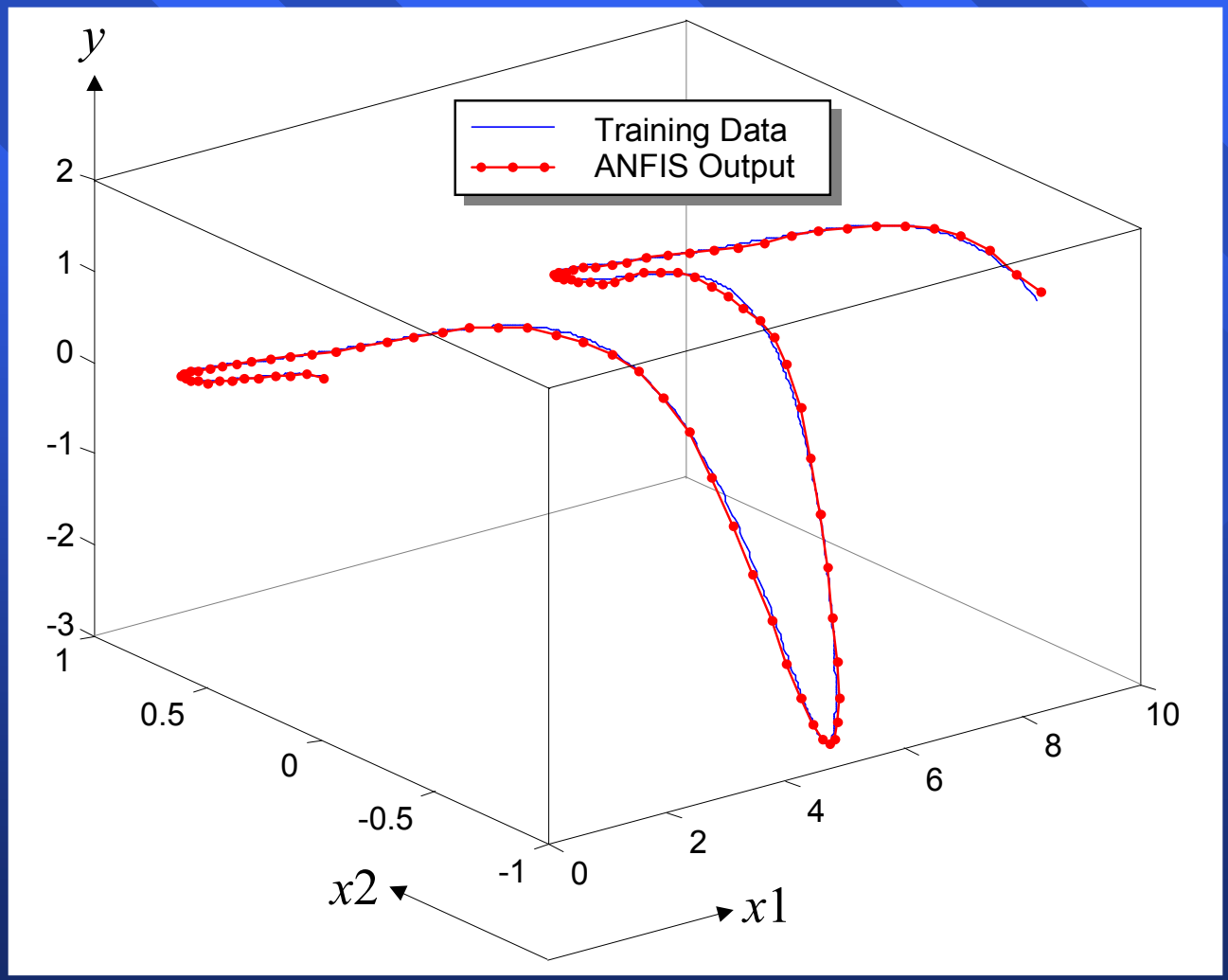


We can achieve some improvement, but much better results are obtained when we assign three membership functions to each input variable. In this case, the ANFIS model will have nine rules, as shown in figure below.

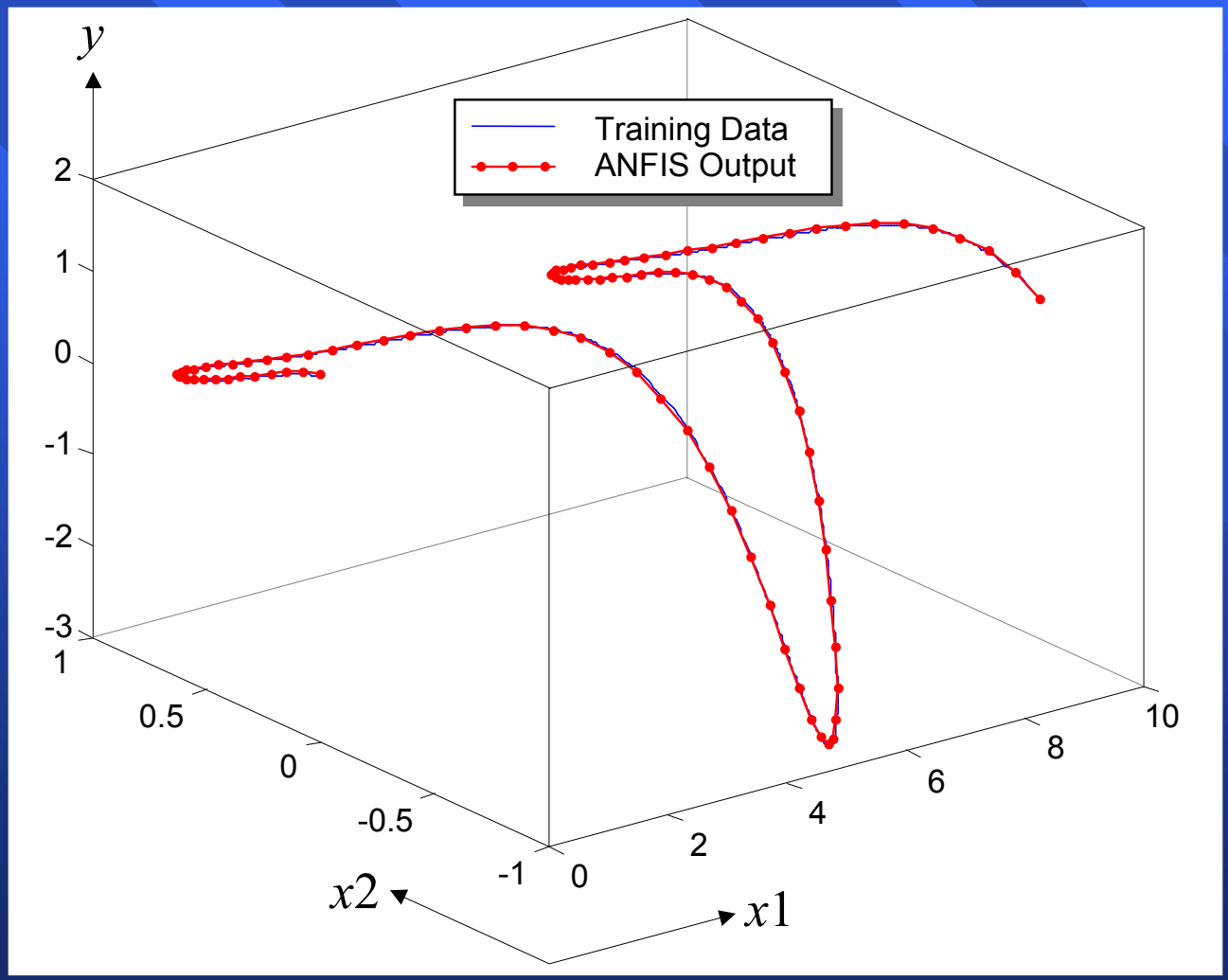
An ANFIS model with nine rules



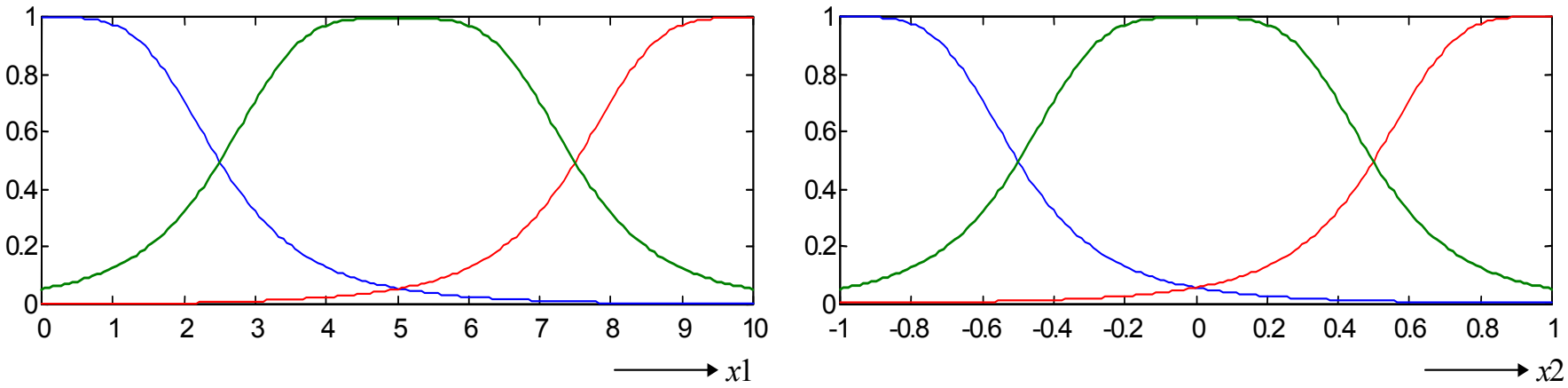
Learning in an ANFIS with three membership functions assigned to each input (one epoch)



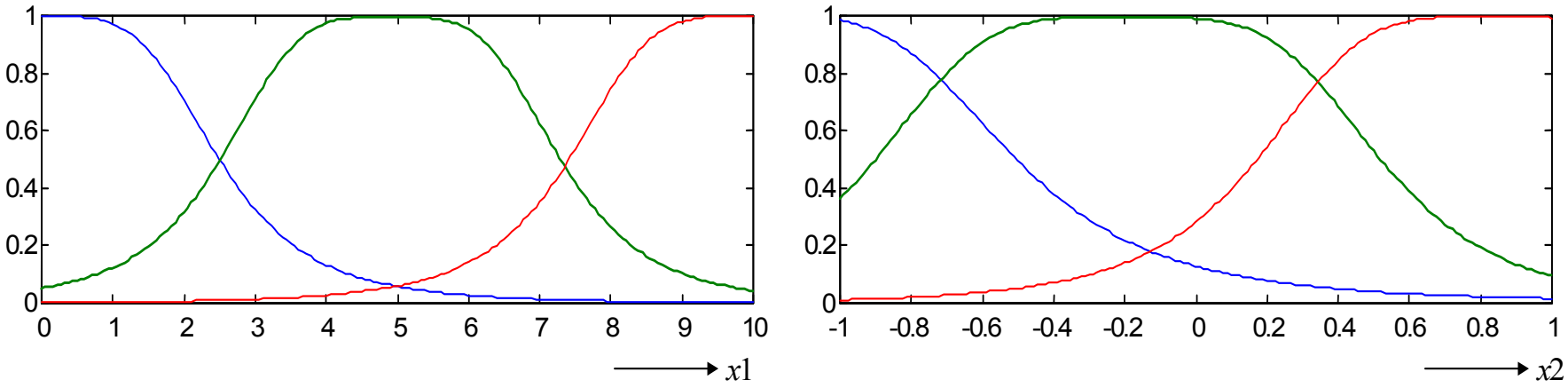
Learning in an ANFIS with three membership functions assigned to each input (100 epochs)



Initial and final membership functions of the ANFIS



(*a*) Initial membership functions.



(*b*) Membership functions after 100 epochs of training.