CRISP SETS

1. Definitions

Universe of discourse : The universe of discourse or universal set is the set which, with reference to a particular context, contains all possible elements having the same characteristics and from which sets can be formed.

The universal set is denoted by E.

Example

(i) The universal set of all numbers in Euclidean space.

(ii) The universal set of all students in a university.

Set : A set is a well-defined collection of objects. Here, well defined means the object either belongs to or does not belong to the set.

Given a set A whose objects are a_1, a_2 ,..., a_n,, we write A as A = { a_1, a_2 ,..., a_n }. Here,

a_1, a_2 ,..., a_n are called the members of the set. Such a form of representing a set is known as *list form*.

A set may also be defined based on the properties the members have to satisfy. In such a

case, a set A is defined as A= {x/P(x))

Here, P(x) stands for the property P to be satisfied by the member x.

Example:

A = {x/x is an odd number}

B = {y /y> 0 and y mod 5 =0}

Membership : An element x is said to be a member of a set A if x belongs to the set A. The membership is indicated by 'ϵ'and is pronounced "belongs to".

Example:

Given A = {4, 5, 6, 7, 8, 10}, for x =3 and y=4, we have x ϵ A and y∉A

Here, observe that each element either belongs to or does not belong to a set.

Cardinality: The number of elements in a set is called its cardinality. Cardinality of a set A is denoted as n(A) or |A| or #A.

Example:

If A = {4, 5, 6, 7} then |A| = 4

Family of sets : A set whose members are sets themselves, is referred to as a -family of sets.

Example :.

A={ {1, 3, 5},{2, 4, 6}, {5, 10}} is a set whose members are the sets {1, 3, 5}, {2, 4, 6), and (5, 10).

Null Set/Empty Set : A set is said to be a null set or empty set if it has no members. A null set is indicated as Φ or {} and indicates an impossible event. Also, | Φ | = 0.

Example:

The set of all prime ministers who are below 15years of age.

Singleton Set: A set with a single element is called a singleton set. A singleton set has cardinality of 1.

Subset : Given sets A and B defined over E the universal set, A is said to be a subset of B if A is fully contained in B, that is, every element of A is in B.

Denoted as $A \subset B$, we say that A is a subset of B, or A is a proper subset of B. On the other hand, if A is contained in or equivalent to that of B then we denote the subset relation as $A \subseteq B$. In such a case, A is called the improper subset of B.

Superset : Given sets A and B on E the universal set, A is said to be a superset of B if every element of B is contained in A. Denoted as $A \supset B$, we say A is a superset of B or A contains B. If A contains B and is equivalent to B, then we denote it as $A \supseteq B$.

Power set : A *power set* of set A is the set of all possible subsets that are derivable from A including null set. A power set is indicated as P(A) and has cardinality of $|P(A)| = 2^{|A|}$

Example :

Let A = { 3, 4, 6, 7} then

P(A) = {{3}, {4}, {6}, {7}, {3, 4}. {4, 6}, {6, 7}, {3, 7}, {3. 6}, {4, 7}, {3, 4, 6}, {4, 6, 7}, {3, 6, 7}, {3, 4, 7}, {3, 4, 6, 7}, Φ}

Here, $|A| = 4$ and $|P(A)| = 2^4 = 16$.

2. Operations on crisp sets:

Union (∪)

The union of two sets *A* and *B (A ∪ B)* is the set of all elements that belong to *A* or *B* or both.

$A \cup B = \{x / x \in A \text{ or } x \in B\}$

Intersection (∩)

The intersection of two sets *A* and *B (A ∩B)* is the set of al 1 elements that belong to *A* and *B*

$A \cap B = \{x/x \in A \text{ and } x \in B\}$

Any two sets which have $A \; n \; B = 0$ are called *Disjoint Sets.*

Complement (^{c})

The complement of a set *A* is the set of all elements which are in *E* but not in *A.*

$A^c = \{x/x \in A, x \in E\}$

Difference (-)

The difference of the set A and B is A — B, the set of all elements which are in A but not in B

$A\text{-}B=\{x/\ x \in A.\ and\ x \notin B\ \}$

3. Properties of Crisp sets

Law of Commutativity:	$(A \cup B) = (B \cup A)$
	$(A \cap B) = (B \cap A)$
Law of Associativity:	$(A \cup B) \cup C = A \cup (B \cup C)$
	$(A \cap B) \cap C = A \cap (B \cap C)$
Law of Distributivity:	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
Idempotent Law:	$A \cup A = A$
	$A \cap A = A$
Identity Law:	$A \cup \Phi = A \Rightarrow A \cup E = E$
	$A \cap \Phi = \Phi \Rightarrow A \cap E = A$
Law of Absorption:	$A \cup (A \cap B) = A$
	$A \cap (A \cup B) = A$
Involution Law:	$(A^c)^c = A$
Law of Transitivity:	$If\ A \subseteq B, B \subseteq C, then\ A \subseteq C$
Law of Excluded Middle:	$(A \cup A^c) = E$
Law of Contradiction:	$(A \cap A^c) = \Phi$
De-morgan laws:	$(A \cup B)^c = A^c \cap B^c$
	$(A \cap B)^c = A^c \cup B^c$

4. Crisp Relations

Cartesian Product

The *Cartesian product* of two sets A and B denoted by $A \times B$ is the set of all ordered pairs such that the first element in the pair belongs to A and the second element belongs to B.

i.e. $$A \times B = \{(a,b)/a \in A,\ b \in B\}$$

If $A \neq B$ and A and B are non-empty then $A \times B \neq B \times A$.

The Cartesian product could be extended to n number of sets

$$\underset{i=1}{\overset{n}{\times}} A_i = \{(a_1,a_2,a_3,\ldots,a_n)/a_i \in A_i \text{ for every } i = 1, 2,\ldots, n\} \tag{6.45}$$

Observe that $$\left|\underset{i=1}{\overset{n}{\times}} A_i\right| = \prod_{i=1}^{n} |A_i|$$

Example

Given $A_1 = \{a, b\}, A_2 = \{1,2\}, A_3 = \{\alpha\}$,

$$A_1 \times A_2 = \{(a, 1), (b, 1), (a, 2), (b, 2)\},\ |A_1 \times A_2| = 4, \text{ and } |A_1| = |A_2| = 2$$

Here, $|A_1 \times A_2| = |A_1| \cdot |A_2|$

Also, $$A_1 \times A_2 \times A_3 = \{(a, 1, \alpha), (a, 2, \alpha), (b, 1, \alpha), (b, 2, \alpha)\}$$

$$|A_1 \times A_2 \times A_3| = 4 = |A_1| \cdot |A_2| \cdot |A_3|$$

Other crisp relations

An *n-ary* relation denoted as $R(X_1, X_2,\ldots, X_n)$ among crisp sets $X_1, X_2,\ldots, X_n$ is a subset of the Cartesian product $\underset{i=1}{\overset{n}{\times}} X_i$ and is indicative of an association or relation among the tuple elements.

For $n = 2$, the relation $R(X_1, X_2)$ is termed as a *binary* relation; for $n = 3$, the relation is termed *ternary*; for $n = 4$, *quarternary*; for $n = 5$, *quinary* and so on.

If the universe of discourse or sets are finite, the *n-ary* relation can be expressed as an n-dimensional *relation matrix*. Thus, for a binary relation $R(X, Y)$ where $X = \{x_1, x_2,\ldots, x_n\}$ and $Y = \{y_1, y_2,\ldots, y_m\}$, the relation matrix R is a two dimensional matrix where X represents the rows, Y represents the columns and $R(i, j) = 1$ if $(x_i, y_j) \in R$ and $R(i,j) = 0$ if $(x_i, y_j) \notin R$.

Example

Given $X = \{1, 2, 3, 4\}$,

$$X \times X = \left\{\begin{matrix}(1,1)(1,2)(1,3)(1,4)(2,1)(2,2)(2,3)(2,4)\\(3,1)(3,2)(3,3)(3,4)(4,1)(4,2)(4,3)(4,4)\end{matrix}\right\}$$

Let the relation R be defined as

$$R = \{(x,y)/y = x+1,\ x, y \in X\}$$

$$R = \{(1, 2)(2, 3)(3, 4)\}$$

The relation matrix R is given by

$$R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

5. Operations on Relations

Given two relations R and S defined on X x Y and represented by relation matrices, the following operations are supported by R and S

Union: $R \cup S$

$$R \cup S(x,y) = \max\left(R(x,y), S(x,y)\right)$$

Intersection: $R \cap S$

$$R \cap S(x,y) = \min\left(R(x,y), S(x,y)\right)$$

Complement: $\bar{R}$

$$\bar{R}(x,y) = 1 - R(x,y)$$

Composition of relations: $R \circ S$

Given R to be a relation on X, Y and S to be a relation on Y, Z then $R \circ S$ is a composition of relation on X, Z defined as

$$R \circ S = \left\{(x,z)/(x,z) \in X \times Z,\ \exists\ y \in Y \text{ such that } (x,y) \in R \text{ and } (y,z) \in S\right\} \tag{6.50}$$

A common form of the composition relation is the *max-min composition*.

Max-min composition:

Given the relation matrices of the relation R and S, the max-min composition is defined as

For $$T = R \circ S$$

$$T(x,z) = \max_{y \in Y}\left(\min(R(x,y), S(y,z))\right) \tag{6.51}$$

Example

Let R, S be defined on the sets $\{1, 3, 5\} \times \{1, 3, 5\}$

Let $R: \{(x,y) \mid y = x + 2\}$, S: $\{(x,y) \mid x < y\}$

$R = \{(1,3)(3,5)\}$, $S = \{(1,3)(1,5)\ (3,5)\}$

The relation matrices are

$$R:\begin{array}{c} \\ 1 \\ 3 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 3 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{array} \qquad S:\begin{array}{c} \\ 1 \\ 3 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 3 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

Using max-min composition

$$R \circ S = \begin{array}{c} \\ 1 \\ 3 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 3 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

(i.e., R_oS (x,z) = Max (min (x^{th} row elements in R and Z^{th} column elements in S))

since

$$R \circ S(1,1) = \max\{\min(0, 0), \min(1, 0), \min(0, 0)\}$$
$$= \max(0, 0, 0) = 0.$$
$$R \circ S(1,3) = \max\{0, 0, 0\} = 0$$
$$R \circ S(1,5) = \max\{0, 1, 0\} = 1.$$

Similarly,

$$R \circ S(3, 1) = 0.$$
$$R \circ S(3, 3) = R \circ S(3, 5) = R \circ S(5,1) = R \circ S(5,3) = R \circ S(5,5) = 0$$

$R \circ S$ from the relation matrix is $\{(1, 5)\}$.

Also,

$$S \circ R = \begin{array}{c} \\ 1 \\ 3 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 3 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

Fuzzy set

6. Crisp Set Vs Fuzzy Set

Crisp Set	**Fuzzy Set**
1. A crisp set is defined as a collection of distinct objects, which share certain characteristics 2. Each individual entity in a set is called a member or an element of the set. 3. The Universe of Discourse is split into 2 groups: members and Non-	1. "FUZZY means "vagueness". Fuzziness occurs when boundary of a piece of information is not clear-cut. 2. Fuzzy set theory permits gradual assessment of membership of elements in a set, described with

members with a bivalent condition – an element either belongs to (1) or does not belong to(0) the set. 4. Used in Digital design	the aid of a membership function valued in the real unit interval [0,1] 3. Infinite-valued 4. Used in fuzzy controllers

7. Characteristics of Fuzzy Logic

Here, are some important characteristics of fuzzy logic:

- Flexible and easy to implement machine learning technique
- Helps you to mimic the logic of human thought
- Logic may have two values which represent two possible solutions
- Highly suitable method for uncertain or approximate reasoning
- Fuzzy logic views inference as a process of propagating elastic constraints
- Fuzzy logic allows you to build nonlinear functions of arbitrary complexity.

Fuzzy logic should be built with the complete guidance of experts

8. Fuzzy sets

Fuzzy sets support a flexible sense of membership of elements to a set. While in crisp set theory: an element either belongs to or does not belong to a set, in fuzzy set theory many degrees of membership (between 0 and 1) are allowed. Thus, a membership function $\mu_A(x)$is associated with a fuzzy set A such that the function maps every element of the universe of discourse X (or the reference set) to the interval [0, 1].

A fuzzy set is defined as:

If X is a universe of discourse and x is a particular element of X, then a fuzzy set A defined on X may be written as a collection of ordered pairs

A= {(x, μ_A(x)), x€X }

Example

Let $X = \{g_1, g_2, g_3, g_4, g_5\}$ be the reference set of students. Let $\tilde{A}$ be the fuzzy set of "smart" students, where "smart" is a fuzzy linguistic term.

$$\tilde{A} = \{(g_1,0.4)(g_2,0.5)(g_3,1)(g_4,0.9)(g_5,0.8)\}$$

Membership Function

The membership function values need not always be described by discrete values. Quite often, these turn out to be as described by a continuous function.

The fuzzy membership function for the fuzzy linguistic term "cool" relating to temperature may turn out to be as illustrated in Fig. 6.10.

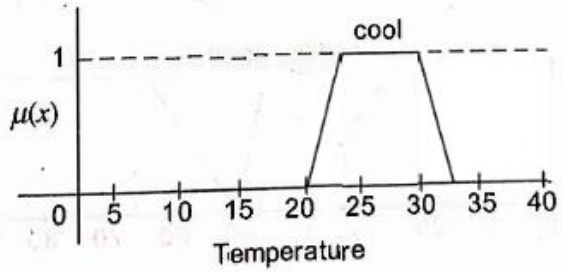


Fig. 6.10 Continuous membership function for "cool".

A membership function can also be given mathematically as

$$\mu_{\tilde{A}}(x) = \frac{1}{(1+x)^2}$$

The graph is as shown in Fig. 6.11.

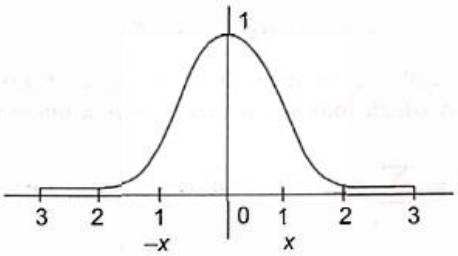


Fig. 6.11 Continuous membership function dictated by a mathematical function.

Different shapes of membership functions exist. The shapes could be triangular, trapezoidal, curved or their variations as shown in Fig. 6.12.

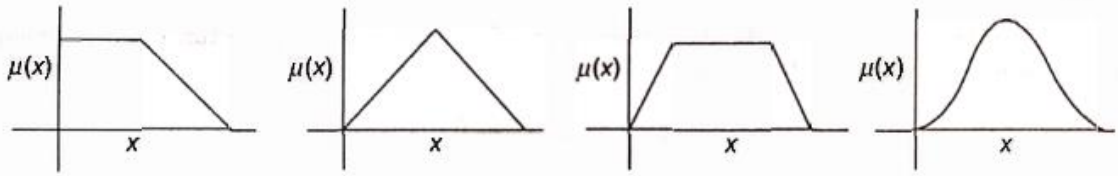


Fig. 6.12 Different shapes of membership function graphs.

Height of Fuzzy Set: is defined as the highest membership value of a set

9. Fuzzy Relations

Fuzzy relation is a fuzzy set defined on the Cartesian product of crisp sets $X_1, X_2, \ldots, X_n$ where the n-tuples $(x_1, x_2, \ldots, x_n)$ may have varying degrees of membership within the relation. The membership values indicate the strength of the relation between the tuples.

Example

Let R be the fuzzy relation between two sets X_1 and X_2 where X_1 is the set of diseases and X_2 is the set of symptoms.

X_1 = {typhoid, viral fever, common cold}

X_2 = {running nose, high temperature, shivering}

The fuzzy relation R may be defined as

	Running nose	*High temperature*	*Shivering*
Typhoid	0.1	0.9	0.8
Viral fever	0.2	0.9	0.7
Common cold	0.9	0.4	0.6

10. Fuzzy Cartesian Product

Let $\tilde{A}$ be a fuzzy set defined on the universe X and $\tilde{B}$ be a fuzzy set defined on the universe Y the Cartesian product between the fuzzy sets A and B indicated as $\tilde{A}$ x $\tilde{B}$ and resulting in a fuzzy relation R is given by

$$\tilde{R} = \tilde{A} \times \tilde{B} \subset X \times Y$$

where $\tilde{R}$ has its membership function given by

$$\begin{aligned} \mu_{\tilde{R}}(x,y) &= \mu_{\tilde{A}\times\tilde{B}}(x,y) \\ &= \min\left(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(y)\right) \end{aligned}$$

Example:

Let $\tilde{A} = \{(x_1, 0.2), (x_2, 0.7), (x_3, 0.4)\}$ and $\tilde{B} = \{(y_1, 0.5), (y_2, 0.6)\}$ be two fuzzy sets defined on the universes of discourse $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2\}$ respectively. Then the fuzzy relation $\tilde{R}$ resulting out of the fuzzy Cartesian product $\tilde{A} \times \tilde{B}$ is given by

$$\tilde{R} = \tilde{A} \text{ x } \tilde{B} = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cc} y_1 & y_2 \end{array} \\ \begin{bmatrix} 0.2 & 0.2 \\ 0.5 & 0.6 \\ 0.4 & 0.4 \end{bmatrix} \end{array}$$

$$\tilde{R}(x_1,y_1)=\min(\mu_{\tilde{A}}(x_1),\mu_{\tilde{B}}(y_1))=\min(0.2,0.5)=0.2$$
$$\tilde{R}(x_1,y_2)=\min(0.2,0.6)=0.2$$
$$\tilde{R}(x_2,y_1)=\min(0.7,0.5)=0.5$$
$$\tilde{R}(x_2,y_2)=\min(0.7,0.6)=0.6$$
$$\tilde{R}(x_3,y_1)=\min(0.4,0.5)=0.4$$
$$\tilde{R}(x_3,y_2)=\min(0.4,0.6)=0.4$$

11. Operations on fuzzy Relations

Let $\tilde{R}$ and $\tilde{S}$ be fuzzy relations on $X \times Y$.

Union

$$\mu_{\tilde{R}\cup\tilde{S}}(x,y) = \max(\mu_{\tilde{R}}(x,y),\ \mu_{\tilde{S}}(x,y))$$

Intersection

$$\mu_{\tilde{R}\cap\tilde{S}}(x,y) = \min(\mu_{\tilde{R}}(x,y),\ \mu_{\tilde{S}}(x,y))$$

Complement

$$\mu_{\tilde{R}^c}(x,y) = 1-\mu_{\tilde{R}}(x,y)$$

Composition of relations

The definition is similar to that of crisp relation. Suppose $\tilde{R}$ is a fuzzy relation defined on $X \times Y$, and $\tilde{S}$ is a fuzzy relation defined on $Y \times Z$, then $\tilde{R} \circ \tilde{S}$ is a fuzzy relation on $X \times Z$. The fuzzy max-min composition is defined as

$$\mu_{\tilde{R}\circ\tilde{S}}(x,z) = \max_{y\in Y}(\min(\mu_{\tilde{R}}(x,y),\ \mu_{\tilde{S}}(y,z)))$$

Example

$X = \{x_1, x_2, x_3\}$ $Y = \{y_1, y_2\}$ $Z = \{z_1, z_2, z_3\}$

Let $\tilde{R}$ be a fuzzy relation

	y_1	y_2
x_1	0.5	0.1
x_2	0.2	0.9
x_3	0.8	0.6

Let $\tilde{S}$ be a fuzzy relation

	z_1	z_2	z_3
y_1	0.6	0.4	0.7
y_2	0.5	0.8	0.9

Then $R \circ S$, by max-min composition yields,

$$R \circ S = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{ccc} z_1 & z_2 & z_3 \end{array} \\ \begin{bmatrix} 0.5 & 0.4 & 0.5 \\ 0.5 & 0.8 & 0.9 \\ 0.6 & 0.6 & 0.7 \end{bmatrix} \end{array}$$

$$\begin{aligned} \mu_{\tilde{R} \circ \tilde{S}}(x_1, z_1) &= \max\,(\min\,(0.5, 0.6), \min\,(0.1, 0.5)) \\ &= \max\,(0.5, 0.1) \\ &= 0.5 \end{aligned}$$

(i.e., Max (min (x_1^{th} row elements in R and Z_1^{th} column elements in S))

Fuzzy Rule base system

Fuzzy linguistic descriptions are formal representations of systems made through fuzzy IF-THEN rules. They encode knowledge about a system in statements of the form—
IF (a set of conditions) are satisfied THEN (a set of consequents) can be inferred.
Fuzzy IF-THEN rules are coded in the form—

$$\text{IF } (x_1 \text{ is } \tilde{A}_1, x_2 \text{ is } \tilde{A}_2, \ldots, x_n \text{ is } \tilde{A}_n) \text{ THEN } (y_1 \text{ is } \tilde{B}_1, y_2 \text{ is } \tilde{B}_2, \ldots, y_n \text{ is } \tilde{B}_n).$$

where linguistic variables x_i, y_j take the values of fuzzy sets A_i and B_j respectively.

Example

If there is heavy rain and strong winds
then there must be severe flood warning.

Here, heavy, strong, and severe are fuzzy sets qualifying the variables rain, wind, and flood warning respectively.

A collection of rules referring to a particular system is known as a *fuzzy rule base*. If the conclusion C to be drawn from a rule base R is the conjunction of all the individual consequents C_i of each rule, then

$$C = C_1 \cap C_2 \cap \ldots \cap C_n$$

where

$$\mu_C(y) = \min\,(\mu_{C_1}(y), \mu_{C_2}(y), \ldots, \mu_{C_n}(y)), \forall y \in Y$$

where y is the universe of discourse.

12. Fuzzy Logic rule base for cruise control

This controller is used to maintain a vehicle at a desired speed. The system consists of two fuzzy inputs, namely speed difference and acceleration, and one fuzzy output, namely throttle control as illustrated in Fig. 7.5.

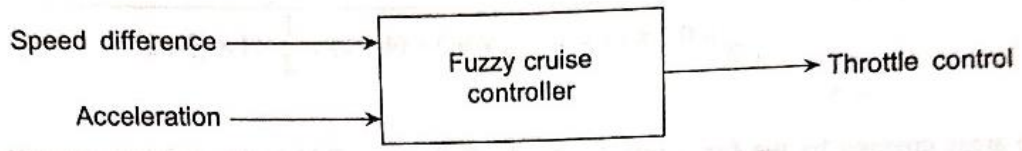


Fig. 7.5 Fuzzy cruise controller.

Fuzzy rule base

A sample fuzzy rule base *R* governing the cruise control is as given in Table 7.5.

Table 7.5 Sample cruise control rule base

Rule 1	If (speed difference is NL) and (acceleration is ZE) then (throttle control is PL).
Rule 2	If (speed difference is ZE) and (acceleration is NL) then (throttle control is PL).
Rule 3	If (speed difference is NM) and (acceleration is ZE) then (throttle control is PM).
Rule 4	If (speed difference is NS) and (acceleration is PS) then (throttle control is PS).
Rule 5	If (speed difference is PS) and (acceleration is NS) then (throttle control is NS).
Rule 6	If (speed difference is PL) and (acceleration is ZE) then (throttle control is NL).
Rule 7	If (speed difference is ZE) and (acceleration is NS) then (throttle control is PS).
Rule 8	If (speed difference is ZE) and (acceleration is NM) then (throttle control is PM).

Key

NL – Negative Large
ZE – Zero
PL – Positive Large
NM – Negative Medium
PM – Positive Medium
NS – Negative Small
PS – Positive Small

13. Advantages and disadvantages of fuzzy logic systems

Advantage	Disadvantage
(1) Similar to human reasoning.	(1) For more accuracy, needs more fuzzy grades which results to increase exponentially the rule.
(2) Based on linguistic model.	(2) The lower speed and also longer run time of system.
(3) Using simple mathematics for nonlinear, integrated and complex systems.	(3) Lack of real time response.
(4) Reasoning and knowledge of human in and in shape of membership rules and functions.	(4) Does not simply capable to receiving feedback for implementation of learning strategy.
(5) Method of nonlinear control and ability to be used efficiently for HVAC systems.	(5) Restricted number of usage of input variables.
(6) High precision.	(6) Inability to straightforwardly advance optimal number of fuzzy rules and determine the membership function parameters.
(7) Rapid operation.	

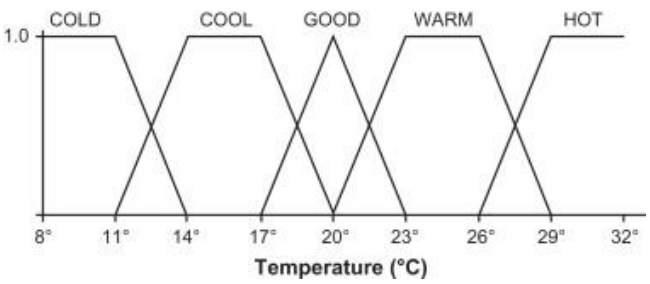
1. Fuzzification

Fuzzification is the process of decomposing a system input and/or output into one or more fuzzy sets. Fuzzification can be thought of as representing any real world feeling or quantity such as warm, happy, love, hungry and so on, into a mathematical model. Many types of curves and tables can be used, but triangular or trapezoidal-shaped membership functions are the most common, since they are easier to represent in embedded controllers.

Each fuzzy set spans a region of input (or output) values graphed against membership. Any particular input is interpreted from this fuzzy set, and a degree of membership is obtained. The membership functions should overlap, in order to allow smooth mapping of the system. The process of fuzzification allows the system inputs and outputs to be expressed in linguistic terms to allow rules to be applied in a simple manner to express a complex system.

Consider a simplified implementation of an air-conditioning system with a temperature sensor. The temperature might be read by a microprocessor that has a fuzzy algorithm that processes output to continuously control the speed of a motor which keeps the room at a “good temperature”; it also can direct a vent upward or downward as necessary. Below Figure 7.18 illustrates the process of fuzzification of the air temperature.

There are five fuzzy sets for temperature: COLD, COOL, GOOD, WARM, and HOT.



The membership function for fuzzy sets COOL and WARM are trapezoidal, the membership function for GOOD is triangular, and those for COLD and HOT are half triangular, with shoulders indicating the physical limits for such a process (staying in a place with a room temperature lower than 8°C or above 32°C would be quite uncomfortable). The way to design such fuzzy sets depends solely on the designer’s experience and intuition. The figure shows some non-overlapping fuzzy sets, which can indicate any nonlinearity in the modeling process. There an input temperature of 18°C would be considered COOL to a degree of 0.75 and would be considered GOOD to a degree of 0.25. To build the rules that will control the air-conditioning motor, we could watch how a human expert adjusts the settings to speed up and slow down the motor in accordance with the temperature, obtaining the rules empirically. For instance, if the room temperature is good, keep the motor speed medium; if it is warm, turn the knob of the speed to fast, and blast the speed if the room is hot. On the other hand, if the temperature is cool, slow down the speed, and stop the motor if it is cold. The beauty of fuzzy logic is the way it turns common sense, and linguistic descriptions, into a computer-controlled system.

Example2: let's consider "beautiful" to be fuzzified.

Let's quantify it into a mathematical model. The question to be asked is "how much beautiful?". Beauty can be classified into Very beautiful (Pretty), Average (Good Looking), Not at all beautiful (Ugly or ordinary). Now let's give them mathematical values...1.0 - 0.7 can be thought of as very beautiful, 0.7 - 0.4 can be thought of as average and 0.4 - 0.0 can be thought of as ordinary, where 1.0 is regarded as more beautiful than 0.8. Based on this , we can classify an object into ranges or values based on its appearance.
This is what fuzzification is.

7.3 FUZZY LOGIC

In crisp logic, the truth values acquired by propositions or predicates are 2-valued, namely *True, False* which may be treated numerically equivalent to (0, 1). However, in fuzzy logic, truth values are multivalued such as *absolutely true, partly true, absolutely false, very true,* and so on and are numerically equivalent to (0–1).

Fuzzy propositions

A *fuzzy proposition* is a statement which acquires a fuzzy truth value. Thus, given $\tilde{P}$ to be a fuzzy proposition, $T(\tilde{P})$ represents the truth value (0–1) attached to $\tilde{P}$. In its simplest form, fuzzy propositions are associated with fuzzy sets. The fuzzy membership value associated with the fuzzy set $\tilde{A}$ for $\tilde{P}$ is treated as the fuzzy truth value $T(\tilde{P})$.

i.e. $T(\tilde{P}) = \mu_{\tilde{A}}(x)$ where $0 \le \mu_{\tilde{A}}(x) \le 1$ (7.17)

Example

$\tilde{P}$: Ram is honest.

$T(\tilde{P}) = 0.8$, if $\tilde{P}$ is partly true.

$T(\tilde{P}) = 1$, if $\tilde{P}$ is absolutely true.

Fuzzy connectives

Fuzzy logic similar to crisp logic supports the following connectives:

(i) *Negation* : –

(ii) *Disjunction* : $\vee$

(iii) *Conjunction* : $\wedge$

(iv) *Implication* : $\Rightarrow$

Table 7.3 illustrates the definition of the connectives. Here $\tilde{P}$, $\tilde{Q}$ are fuzzy propositions and $T(\tilde{P})$, $T(\tilde{Q})$, are their truth values.

Table 7.3 Fuzzy connectives

Symbol	Connective	Usage	Definition
–	Negation	$\bar{\tilde{P}}$	$1 - T(\tilde{P})$
$\vee$	Disjunction	$\tilde{P} \vee \tilde{Q}$	$\max(T(\tilde{P}), T(\tilde{Q}))$
$\wedge$	Conjunction	$\tilde{P} \wedge \tilde{Q}$	$\min(T(\tilde{P}), T(\tilde{Q}))$
$\Rightarrow$	Implication	$\tilde{P} \Rightarrow \tilde{Q}$	$\bar{\tilde{P}} \vee \tilde{Q} = \max(1 - T(\tilde{P}), T(\tilde{Q}))$

$\tilde{P}$ and $\tilde{Q}$ related by the '$\Rightarrow$' operator are known as antecedent and consequent respectively. Also, just as in crisp logic, here too, '$\Rightarrow$' represents the IF-THEN statement as

IF x is $\tilde{A}$ THEN y is $\tilde{B}$, and is equivalent to

$$\tilde{R} = (\tilde{A} \times \tilde{B}) \cup (\overline{\tilde{A}} \times Y) \tag{7.18}$$

The membership function of $\tilde{R}$ is given by

$$\mu_{\tilde{R}}(x, y) = \max(\min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(y)), 1 - \mu_{\tilde{A}}(x)) \tag{7.19}$$

Also, for the compound implication IF x is $\tilde{A}$ THEN y is $\tilde{B}$ ELSE y is $\tilde{C}$ the relation R is equivalent to

$$\tilde{R} = (\tilde{A} \times \tilde{B}) \cup (\overline{\tilde{A}} \times \tilde{C}) \tag{7.20}$$

The membership function of $\tilde{R}$ is given by

$$\mu_{\tilde{R}}(x, y) = \max(\min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(y)), \min(1 - \mu_{\tilde{A}}(x), \mu_{\tilde{C}}(y))) \tag{7.21}$$

Example

$\tilde{P}$: Mary is efficient, $T(\tilde{P}) = 0.8$

$\tilde{Q}$: Ram is efficient, $T(\tilde{Q}) = 0.65$

(i) $\overline{\tilde{P}}$: Mary is not efficient.

$$T(\overline{\tilde{P}}) = 1 - T(\tilde{P}) = 1 - 0.8 = 0.2$$

(ii) $\tilde{P} \wedge \tilde{Q}$: Mary is efficient and so is Ram.

$$\begin{aligned} T(\tilde{P} \wedge \tilde{Q}) &= \min(T(\tilde{P}), T(\tilde{Q})) \\ &= \min(0.8, 0.65) \\ &= 0.65 \end{aligned}$$

(iii) $T(\tilde{P} \vee \tilde{Q})$: Either Mary or Ram is efficient.

$$\begin{aligned} T(\tilde{P} \vee \tilde{Q}) &= \max(T(\tilde{P}), T(\tilde{Q})) \\ &= \max(0.8, 0.65) \\ &= 0.8 \end{aligned}$$

(iv) $\tilde{P} \Rightarrow \tilde{Q}$: If Mary is efficient then so is Ram.

$$\begin{aligned} T(\tilde{P} \Rightarrow \tilde{Q}) &= \max(1 - T(\tilde{P}), T(\tilde{Q})) \\ &= \max(0.2, 0.65) \\ &= 0.65 \end{aligned}$$

Example 7.10

Let $X = \{a, b, c, d\}$ $Y = \{1, 2, 3, 4\}$

and $\tilde{A} = \{(a, 0)(b, 0.8)(c, 0.6)(d, 1)\}$

$\tilde{B} = \{(1, 0.2)(2, 1)(3, 0.8)(4, 0)\}$

$\tilde{C} = \{(1, 0)(2, 0.4)(3, 1)(4, 0.8)\}$

Determine the implication relations

(i) IF x is $\tilde{A}$ THEN y is $\tilde{B}$.

(ii) IF x is $\tilde{A}$ THEN y is $\tilde{B}$ ELSE y is $\tilde{C}$.

Solution

To determine (i) compute

$$\tilde{R} = (\tilde{A} \times \tilde{B}) \cup (\bar{\tilde{A}} \times Y) \quad \text{where}$$

$$\mu_{\tilde{R}}(x, y) = \max(\min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(y)), 1 - \mu_{\tilde{A}}(x))$$

$$\tilde{A} \times \tilde{B} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.2 & 0.8 & 0.8 & 0 \\ 0.2 & 0.6 & 0.6 & 0 \\ 0.2 & 1 & 0.8 & 0 \end{bmatrix} \end{array}$$

$$\bar{\tilde{A}} \times Y = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0.2 & 0.2 & 0.2 & 0.2 \\ 0.4 & 0.4 & 0.4 & 0.4 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Here, Y the universe of discourse could be viewed as $\{(1, 1)\ (2, 1)\ (3, 1)\ (4, 1)\}$ a fuzzy set all of whose elements x have $\mu(x) = 1$.

Therefore,

$$\tilde{R} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0.2 & 0.8 & 0.8 & 0.2 \\ 0.4 & 0.6 & 0.6 & 0.4 \\ 0.2 & 0.1 & 0.8 & 0 \end{bmatrix} \end{array}$$

which represents IF x is $\tilde{A}$ THEN y is $\tilde{B}$.

To determine (ii) compute

$$\tilde{R} = (\tilde{A} \times \tilde{B}) \cup (\overline{\tilde{A}} \times \tilde{C}) \text{ where}$$

$$\mu_{\tilde{R}}(x, y) = \max(\min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(y)), \min(1 - \mu_{\tilde{A}}(x), \mu_{\tilde{C}}(y)))$$

$$\tilde{A} \times \tilde{B} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.2 & 0.8 & 0.8 & 0 \\ 0.2 & 0.6 & 0.6 & 0 \\ 0.2 & 1 & 0.8 & 0 \end{bmatrix} \end{array}$$

$$\overline{\tilde{A}} \times \tilde{C} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0.4 & 1 & 0.8 \\ 0 & 0.2 & 0.2 & 0.2 \\ 0 & 0.4 & 0.4 & 0.4 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Therefore,

$$\tilde{R} = \max((\tilde{A} \times \tilde{B}), (\overline{\tilde{A}} \times \tilde{C})) \text{ gives}$$

$$\tilde{R} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0.4 & 1 & 0.8 \\ 0.2 & 0.8 & 0.8 & 0.2 \\ 0.2 & 0.6 & 0.6 & 0.4 \\ 0.2 & 1 & 0.8 & 0 \end{bmatrix} \end{array}$$

The above $\tilde{R}$ represents IF x is $\tilde{A}$ THEN y is $\tilde{B}$ ELSE y is $\tilde{C}$.

7.3.1 Fuzzy Quantifiers

Just as in crisp logic where predicates are quantified by quantifiers, fuzzy logic propositions are also quantified by *fuzzy quantifiers*. There are two classes of fuzzy quantifiers such as

(i) Absolute quantifiers and

(ii) Relative quantifiers

While absolute quantifiers are defined over $\Re$, relative quantifiers are defined over [0–1].

Example

Absolute quantifier	Relative quantifier
round about 250	almost
much greater than 6	about
some where around 20	most

7.3.2 Fuzzy Inference

Fuzzy inference also referred to as *approximate reasoning* refers to computational procedures used for evaluating linguistic descriptions. The two important inferring procedures are

(i) Generalized Modus Ponens (GMP)

(ii) Generalized Modus Tollens (GMT)

GMP is formally stated as

$$\begin{array}{ll} \text{IF} & x \text{ is } \tilde{A} \text{ THEN } y \text{ is } \tilde{B} \\ & x \text{ is } \tilde{A}' \\ \hline & y \text{ is } \tilde{B}' \end{array} \tag{7.22}$$

Here, $\tilde{A}$, $\tilde{B}$, $\tilde{A}'$ and $\tilde{B}'$ are fuzzy terms. Every fuzzy linguistic statement above the line is analytically known and what is below is analytically unknown.

To compute the membership function of $\tilde{B}'$, the max-min composition of fuzzy set A' with $\tilde{R}(x, y)$ which is the known implication relation (IF-THEN relation) is used. That is,

$$\tilde{B}' = \tilde{A}' \circ \tilde{R}(x, y) \tag{7.23}$$

In terms of membership function,

$$\mu_{\tilde{B}'}(y) = \max\left(\min\left(\mu_{\tilde{A}'}(x), \mu_{\tilde{R}}(x, y)\right)\right) \tag{7.24}$$

where $\mu_{\tilde{A}'}(x)$ is the membership function of $\tilde{A}'$, $\mu_{\tilde{R}}(x, y)$ is the membership function of the implication relation and $\mu_{\tilde{B}'}(y)$ is the membership function of $\tilde{B}'$

On the other hand, GMT has the form

$$\begin{array}{ll} \text{IF} & x \text{ is } \tilde{A} \text{ THEN } y \text{ is } \tilde{B} \\ & y \text{ is } \tilde{B}' \\ \hline & x \text{ is } \tilde{A}' \end{array}$$

The membership of $\tilde{A}'$ is computed on similar lines as

$$\tilde{A}' = \tilde{B}' \circ \tilde{R}(x, y)$$

In terms of membership function,

$$\mu_{\tilde{A}'}(x) = \max\left(\min\left(\mu_{\tilde{B}'}(y), \mu_{\tilde{R}}(x, y)\right)\right) \tag{7.25}$$

Example

Apply the fuzzy Modus Ponens rule to deduce Rotation is quite slow given

(i) If the temperature is high then the rotation is slow.

(ii) The temperature is very high.

Let $\tilde{H}$ (High), $\tilde{VH}$ (Very High), $\tilde{S}$ (Slow) and $\tilde{QS}$ (Quite Slow) indicate the associated fuzzy sets as follows:

For $X = \{30, 40, 50, 60, 70, 80, 90, 100\}$, the set of temperatures and $Y = \{10, 20, 30, 40, 50, 60\}$, the set of rotations per minute,

$$\tilde{H} = \{(70, 1)\ (80, 1)\ (90, 0.3)\}$$

$$\tilde{VH} = \{(90, 0.9)\ (100, 1)\}$$

$$\tilde{QS} = \{(10, 1)\ (20, 0.8)\}$$

$$\tilde{S} = \{(30, 0.8)\ (40, 1)\ (50, 0.6)\}$$

To derive $\tilde{R}(x, y)$ representing the implication relation (i), we need to compute

$$\tilde{R}(x, y) = \max(\tilde{H} \times \tilde{S}, \bar{\tilde{H}} \times Y)$$

$$\tilde{H} \times \tilde{S} = \begin{array}{c} \\ 30 \\ 40 \\ 50 \\ 60 \\ 70 \\ 80 \\ 90 \\ 100 \end{array} \begin{array}{c} \begin{array}{cccccc} 10 & 20 & 30 & 40 & 50 & 60 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 0 & 0 & 0.3 & 0.3 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$\bar{\tilde{H}} \times Y = \begin{array}{c} \\ 30 \\ 40 \\ 50 \\ 60 \\ 70 \\ 80 \\ 90 \\ 100 \end{array} \begin{array}{c} \begin{array}{cccccc} 10 & 20 & 30 & 40 & 50 & 60 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

$$\tilde{R}(x,y) = \begin{array}{c} \\ 30 \\ 40 \\ 50 \\ 60 \\ 70 \\ 80 \\ 90 \\ 100 \end{array} \begin{array}{c} \begin{array}{cccccc} 10 & 20 & 30 & 40 & 50 & 60 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

To deduce Rotation is quite slow we make use of the composition rule

$$\tilde{QS} = V\tilde{H} \circ \tilde{R}(x,y)$$

$$= [0\ 0\ 0\ 0\ 0\ 0\ 0.9\ 1] \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= [1\ 1\ 1\ 1\ 1\ 1]$$

7.4 FUZZY RULE BASED SYSTEM

Fuzzy linguistic descriptions are formal representations of systems made through fuzzy IF-THEN rules. They encode knowledge about a system in statements of the form—
IF (a set of conditions) are satisfied THEN (a set of consequents) can be inferred.
Fuzzy IF-THEN rules are coded in the form—

$$\text{IF } (x_1 \text{ is } \tilde{A}_1, x_2 \text{ is } \tilde{A}_2, \ldots, x_n \text{ is } \tilde{A}_n) \text{ THEN } (y_1 \text{ is } \tilde{B}_1, y_2 \text{ is } \tilde{B}_2, \ldots, y_n \text{ is } \tilde{B}_n).$$

where linguistic variables x_i, y_j take the values of fuzzy sets A_i and B_j respectively.

Example

If there is heavy rain and strong winds
then there must be severe flood warning.

Here, heavy, strong, and severe are fuzzy sets qualifying the variables rain, wind, and flood warning respectively.

A collection of rules referring to a particular system is known as a *fuzzy rule base*. If the conclusion C to be drawn from a rule base R is the conjunction of all the individual consequents C_i of each rule, then

$$C = C_1 \cap C_2 \cap \ldots \cap C_n \tag{7.26}$$

where

$$\mu_C(y) = \min(\mu_{C_1}(y), \mu_{C_2}(y), \ldots, \mu_{C_n}(y)), \forall y \in Y \tag{7.27}$$

where Y is the universe of discourse.

On the other hand, if the conclusion C to be drawn from a rule base R is the disjunction of the individual consequents of each rule, then

$$C = C_1 \cup C_2 \cup C_3 \ldots \cup C_n \tag{7.28}$$

where

$$\mu_C(y) = \max(\mu_{C_1}(y), \mu_{C_2}(y), \ldots, \mu_{C_n}(y)), \forall y \in Y \tag{7.29}$$

7.5 DEFUZZIFICATION

In many situations, for a system whose output is fuzzy, it is easier to take a crisp decision if the output is represented as a single scalar quantity. This conversion of a fuzzy set to single crisp value is called *defuzzification* and is the reverse process of *fuzzification.*

Several methods are available in the literature (Hellendoorn and Thomas, 1993) of which we illustrate a few of the widely used methods, namely *centroid method, centre of sums,* and *mean of maxima.*

Centroid method

Also known as the *centre of gravity* or the *centre of area* method, it obtains the centre of area (x^*) occupied by the fuzzy set. It is given by the expression

$$x^* = \frac{\int \mu(x)\, x\, d\, x}{\int \mu(x)\, d\, x} \tag{7.30}$$

for a continuous membership function, and

$$x^* = \frac{\sum_{i=1}^{n} x_i . \mu(x_i)}{\sum_{i=1}^{n} \mu(x_i)} \tag{7.31}$$

for a discrete membership function.

Here, n represents the number of elements in the sample, x_i's are the elements, and $\mu(x_i)$ is its membership function.

Centre of sums (COS) method

In the centroid method, the overlapping area is counted once whereas in *centre of sums*, the overlapping area is counted twice. COS builds the resultant membership function by taking the algebraic sum of outputs from each of the contributing fuzzy sets $\tilde{A}_1$, $\tilde{A}_2$,..., etc. The defuzzified value x^* is given by

$$x^* = \frac{\sum_{i=1}^{N} x_i \cdot \sum_{k=1}^{n} \mu_{\tilde{A}_k}(x_i)}{\sum_{i=1}^{N} \sum_{k=1}^{n} \mu_{\tilde{A}_k}(x_i)} \tag{7.32}$$

Here n is the number of fuzzy sets and N the number of fuzzy variables. COS is actually the most commonly used defuzzification method. It can be implemented easily and leads to rather fast inference cycles.

Mean of maxima (MOM) defuzzification

One simple way of defuzzifying the output is to take the crisp value with the highest degree of membership. In cases with more than one element having the maximum value, the mean value of the maxima is taken. The equation of the defuzzified value x^* is given by

$$x^* = \frac{\sum_{x_i \in M} x_i}{|M|} \tag{7.33}$$

where $M = \{x_i \,|\mu(x_i)$ is equal to the *height* of fuzzy set$\}$
$|M|$ is the cardinality of the set M. In the continuous case, M could be defined as

$$M = \{x \in [-c, c] \,|\mu(x) \text{ is equal to the height of the fuzzy set}\} \tag{7.34}$$

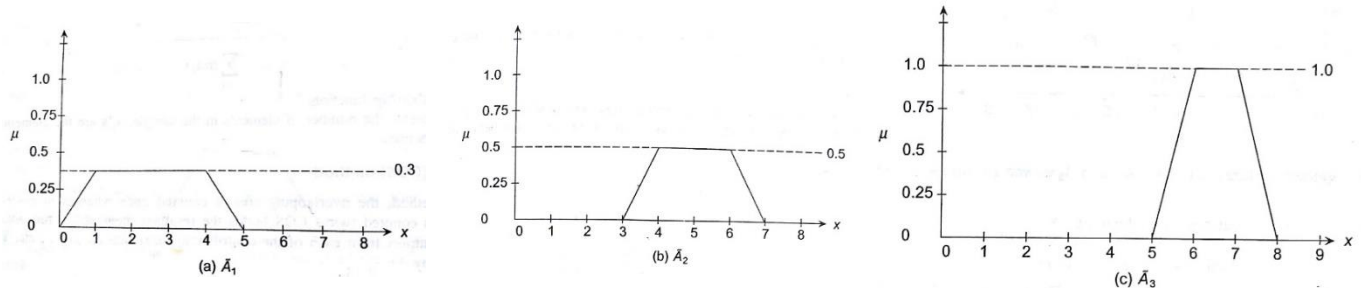
In such a case, the *mean of maxima* is the arithmetic average of mean values of all intervals contained in M including zero length intervals.

The *height* of a fuzzy set A, i.e. $h(A)$ is the largest membership grade obtained by any element in that set.

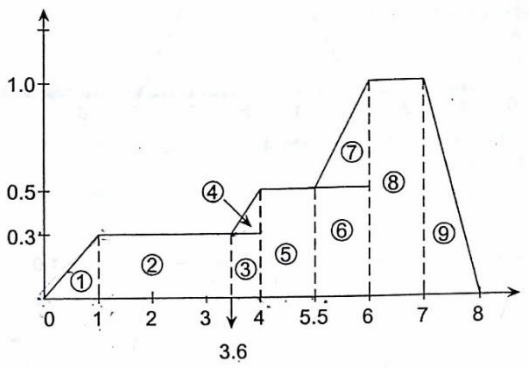
Example

$\tilde{A}_1$, $\tilde{A}_2$, and $\tilde{A}_3$ are three fuzzy sets as shown in Fig. 7.1(a), (b), and (c). Figure 7.2 illustrates the aggregate of the fuzzy sets.

Find the defuzzified output



i. Centroid Method



Area segment no.	Area (A)	$\bar{x}$	$A\bar{x}$
1	$\frac{1}{2} \times 0.3 \times 1 = 0.15$	0.5	0.075
2	$2.6 \times 0.3 = 0.78$	2.3	1.794
3	$0.3 \times 0.4 = 0.12$	3.8	0.456
4	$\frac{1}{2} \times 0.4 \times 0.2 = 0.04$	3.8667	0.1546
5	$1.5 \times 0.5 = 0.75$	4.75	3.5625
6	$0.5 \times 0.5 = 0.25$	5.75	1.4375
7	$\frac{1}{2} \times 0.5 \times 0.5 = 0.125$	5.833	0.729
8	$1 \times 1 = 1$	6.5	6.5
9	$\frac{1}{2} \times 1 \times 1 = 0.5$	7.33	3.665

ii. Centre of sums method

$$x^* = \frac{\frac{1}{2} \times 0.3 \times (3+5) \times 2.5 + \frac{1}{2} \times 0.5 \times (4+2) \times 5 + \frac{1}{2} \times 1 \times (3+1) \times 6.5}{\frac{1}{2} \times 0.3 \times (3+5) + \frac{1}{2} \times 0.5 \times (4+2) + \frac{1}{2} \times 1 \times (3+1)}$$

$$= 5$$

iii. Mean of Maxima Method

X^*= mean of elements with maximum membership value

= (6+7)/2 = 6.5

2. Fuzzy Logic rule base for cruise control

This controller is used to maintain a vehicle at a desired speed. The system consists of two fuzzy inputs, namely speed difference and acceleration, and one fuzzy output, namely throttle control as illustrated in Fig. 7.5.

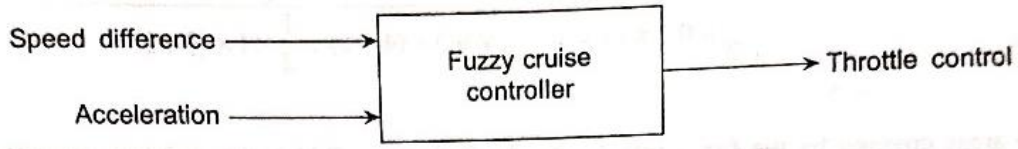


Fig. 7.5 Fuzzy cruise controller.

Fuzzy rule base

A sample fuzzy rule base *R* governing the cruise control is as given in Table 7.5.

Table 7.5 Sample cruise control rule base

Rule 1	If (speed difference is NL) and (acceleration is ZE) then (throttle control is PL).
Rule 2	If (speed difference is ZE) and (acceleration is NL) then (throttle control is PL).
Rule 3	If (speed difference is NM) and (acceleration is ZE) then (throttle control is PM).
Rule 4	If (speed difference is NS) and (acceleration is PS) then (throttle control is PS).
Rule 5	If (speed difference is PS) and (acceleration is NS) then (throttle control is NS).
Rule 6	If (speed difference is PL) and (acceleration is ZE) then (throttle control is NL).
Rule 7	If (speed difference is ZE) and (acceleration is NS) then (throttle control is PS).
Rule 8	If (speed difference is ZE) and (acceleration is NM) then (throttle control is PM).

Key

NL – Negative Large
ZE – Zero
PL – Positive Large
NM – Negative Medium
PM – Positive Medium
NS – Negative Small
PS – Positive Small

NEURAL NETWORKS

1.0.0 Introduction

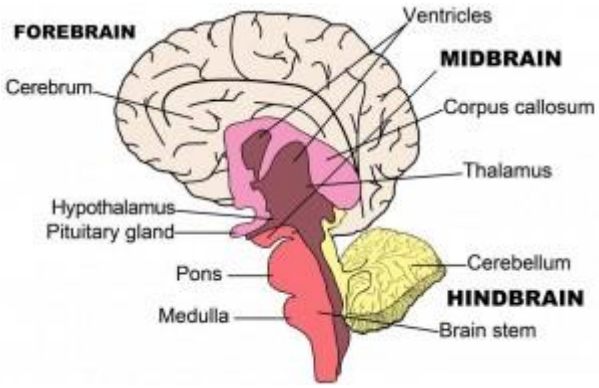
The recent rise of interest in neural networks has its roots in the recognition that the brain performs computations in a different manner than do conventional digital computers. Computers are extremely fast and precise at executing sequences of instructions that have been formulated for them. A human information process sing system is composed of neurons switching at speeds about a million times slower than computer gates. Yet, humans are more efficient than computers at computationally complex tasks such as speech understanding. Moreover, not only humans, but also even animals, can process visual information better than the fastest computers.

Artificial neural systems, or neural networks (NN), are physical cellular systems, which can acquire, store, and utilize experiential knowledge. The knowledge is in the form of stable states or mappings embedded in networks that can be recalled in response to the presentation cues. Neural network processing typically involves dealing with large-scale problems in terms of dimensionality, amount of data handled, and the volume of simulation or neural hardware processing. This large-scale approach is both essential and typical for real-life applications. By keeping view of all these, the research community has made an effort in designing and implementing the various neural network models for different applications. Now let us formally define the basic idea of neural network:

Definition: ***A neural network is a computing system made up of a number of simple, highly interconnected nodes or processing elements, which process information by its dynamic state response to external inputs.***

2.1.0 The structure and function of the human brain

The brain structure is composed of three main parts: the forebrain, midbrain and hindbrain, each with multiple parts.

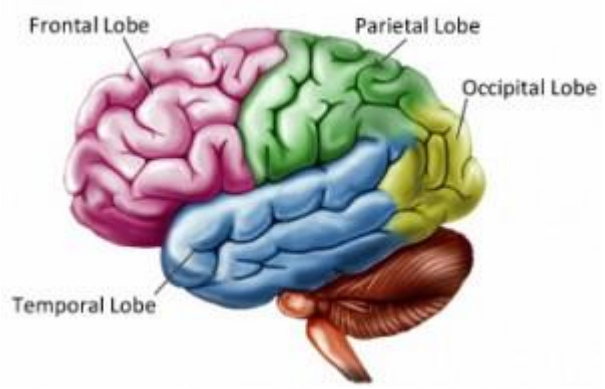


Forebrain

The Cerebrum: Also known as the cerebral cortex, the cerebrum is the largest part of the human brain, and it is associated with higher brain function such as thought and action. Nerve cells make up the gray surface, which is a little thicker than our thumb. White nerve fibers beneath the surface carry signals between nerve cells in other parts of the brain and body. Its wrinkled surface increases the surface area, and is a six-layered structure found in mammals, called the neocortex. It is divided into four sections, called “lobes”. They are; the frontal lobe, the parietal lobe, the occipital lobe and the temporal lobe.

Functions of The Lobes:

Frontal Lobe – The frontal lobe lies just beneath our forehead and is associated with our brain's ability to reason, organize, plan, speak, move, make facial expressions, serial task, problem solve, control inhibition, spontaneity, initiate and self-regulate behaviors, pay attention, remember and control emotions.



Parietal Lobe – The parietal lobe is located at the upper rear of our brain, and controls our complex behaviors, including senses such as vision, touch, body awareness and spatial orientation. It plays important roles in integrating sensory information from various parts of our body, knowledge of numbers and their relations, and in the manipulation of objects. Portions are involved with our visuospatial processing, language comprehension, the ability to construct, body positioning and movement, neglect/inattention, left-right differentiation and self-awareness/insight.

Occipital Lobe – The occipital lobe is located at the back of our brain, and is associated with our visual processing, such as visual recognition, visual attention, spatial analysis (moving in a 3-D world) and visual perception of body language; such as postures, expressions and gestures.

Temporal Lobe – The temporal lobe is located near our ears, and is associated with processing our perception and recognition of auditory stimuli (including our ability to focus on one sound among many, like listening to one voice among many at a party), comprehending spoken language, verbal memory, visual memory and language production (including fluency and word-finding), general knowledge and autobiographical memories.

2.2.0 The Biological Prototype of neural network

The human nerve system built of cells called neurons is of staggering complexity. It contains approximately ten thousand million (10^{11}) basic neurons. Each of these neurons is connected to about ten thousand (10^4) others. The connection of each neuron with other neurons forms a densely network called a neural network. These massive interconnections provide an exceptionally large computing power and memory. The neuron accepts many inputs, which are all added up in some fashion. If enough active inputs are received at once, then the neuron will be activated at once, then the neuron will be activated and "fire"; if not, then the neuron will remain in its inactive, quit state. The schematic diagram of biological neuron is shown in Fig.2.1. From a systems theory, the neuron considered to be as a multiple-input-single-output (MISO) system as shown in Fig.2.2.

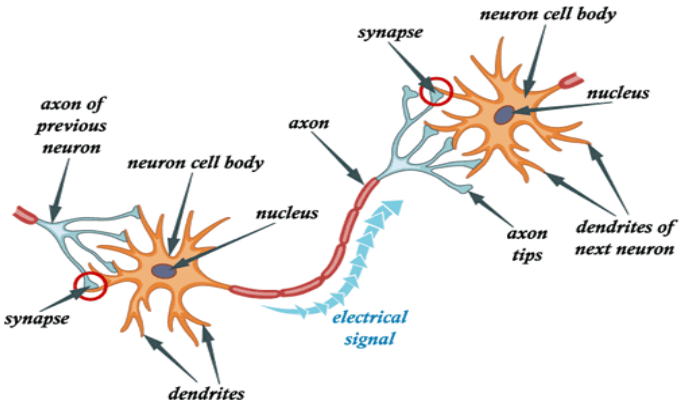


Fig. 2.1. A Schematic view of the biological neuron

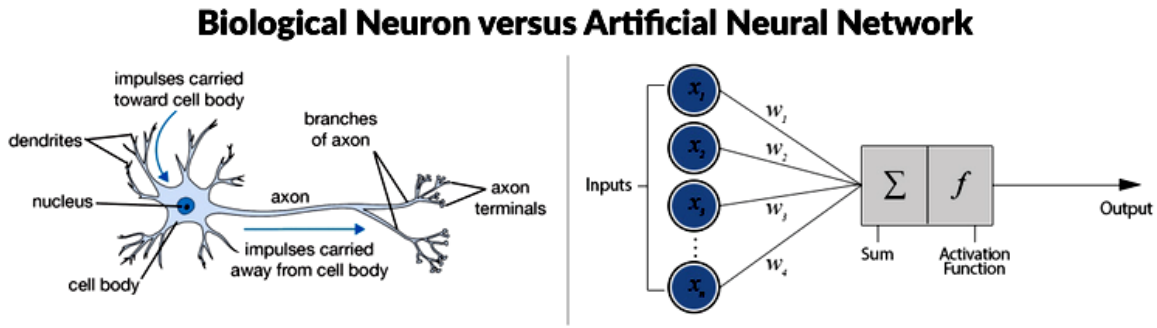


Fig.2.2 Model representation of a biological neuron with multiple inputs

Human	Artificial
Neuron	Processing Element
Dendrites	Combining Function
Cell Body	Activation Function
Axons	Element Output
Synapses	Weights

In general the function of the main elements can be given as,

Dendrite - Receives signals from other neurons

Soma - Processes all the incoming signals

Axon - When a particular amount of input is received, then the cell fires. It transmits signal through axon to other cells.

Synapses – Connections between neurons

The soma is the body of the neuron. Attached to the soma there are long irregularly shaped filaments, called dendrites. These nerve processes are often less than a micron in

diameter, and have complex branching shapes. The dendrites act as the connections through which all the inputs to the neuron arrive. These cells are able to perform more complex functions than simple addition on the inputs they receive, but considering simple summation is a reasonable approximation.

Another type of nerve process attached to the soma is called an axon. This is electrically active, unlike the dendrite, and serves as the output channel of the neuron. Axons always appear on output cell, but are often absent from interconnections, which have both inputs and outputs on dendrites. The axon is a non-linear threshold device, producing a voltage pulse, called an action potential, that last about 1 millisecond (10^{-3} sec) when the resting potential within the soma rises above a certain critical threshold.

The axon terminates in a specialized contact called a synapse that couples the axon with the dendrite of another cell. There is no direct linkage across the junction; rather, it is temporally chemical one. The synapse releases chemicals called neurotransmitters when its potential is raised sufficiently by the action potential. It may take the arrival of more than one action potential before the synapse is triggered. The neurotransmitters that are released by the synapse diffuse across the gap, any chemically activate gates on the dendrites, which, when open, allow charged ions to flow. It is this flow of ions that alters the dendrite potential, and provides a voltage pulse on the dendrite, which is then conducted along into the next neuron body. Each dendrite may have many synapses acting on it, allowing massive interconnectivity to be achieved.

1.1.0 Humans and Computers

Human beings are more intelligent than computers. Computers could only do logical things well. But in case of solving cross word puzzles, vision problem, controlling an arm to pick it up or something similar, that requires exceptionally complex techniques. Like these problems, human beings do better than computers.

Computers are designed to carry out one instruction after another, extremely rapid, whereas our brains work with many slower units. Whereas a computer can typically carry out a few million operations every second, the units in the brain respond about ten per second. However, they work on many different things at once, which computer can't do.

The computer is a high-speed, serial machine and is used as such, compared to the slow, highly parallel nature of the brain. Counting is an essentially serial activity, as is adding, with the thing done one after another, and so the computer can beat the brain any time. For vision, or speech recognition, the problem is a highly parallel one, with many different and conflicting inputs, triggering many different and conflicting ideas and memories, and it is only the combining of all these different factors that allow us to perform such feats, but then, our brains are able to operate in parallel easily and so we leave the computers far behind.

1.2.0 Comparison between brain and computer

The main differences between the brain and the computer are:

- Biological neurons, the basic building blocks of the brain, are slower than silicon logic gates. The neurons operate- in milliseconds, which is about six orders of magnitude slower than the silicon gates operating in the nanosecond range.
- The brain makes up for the slow rate of operation with two factors:
    - a huge number of nerve cells (neurons) and interconnections between them. The human brain contains approximately 10^{14} to 10^{15} interconnections.
    - the function of a biological neuron seems to be much more complex than that of a logic gate.
- The brain is very energy efficient. It consumes only about 10-16 joules per operation per second, comparing with 10-6 joules per operation per sec, for a digital computer.
- The brain is a highly complex, non-linear, parallel information processing system. It performs tasks like pattern recognition, perception, motor control, many times faster than the fastest digital computers.
- Consider an efficiency of the visual system which provides a representation of the environment which enables us to interact with the environment. For example, a complex task of perceptual recognition, e.g. recognition of a familiar face embedded in an unfamiliar scene can be accomplished in 100-200 ms, whereas tasks of much lesser complexity can take hours if not days on conventional computers.

Table 2.3 shows the major differences between the biological and the artificial neural network.

Charact-eristics	**ANN**	**Biological Neural Network**
Speed	Neural networks are faster in processing information. The cycle time corresponding to execution of one step of a program in the central processing unit is in the range of few nano seconds.	Biological neurons arc slow in processing the information. The cycle time Corresponding to a neural event prompted by an external stimulus occurs in a milli second range.
Processing	Many programs have large number of instructions. and they operate in a sequential mode one instruction after another on a conventional computer.	Biological neural networks can perform massively parallel operations. The brain possesses the capability to operate with massively parallel operations, each of them having only few steps.
Size and Complexity	These do not involve as much computational neurons. Hence it is difficult to perform complex pattern recognition.	The number of neurons in the brain is estimated to about $l0^{11}$ and the total number of interconnections to be around 10^{15}.

		The size and complexity of connections gives the brain the power of performing complex pattern recognition tasks, which cannot be realized on a computer.
Storage	In a computer, the information is stored in the memory, which is addressed by its location. Any new information in the same location destroys the old information. Hence here it is strictly replaceable.	Neural networks store information in the strengths of the interconnections. Information in the brain is adaptable, because new information is added by adjusting the interconnection strengths. Without destroying the old information.
Fault tolerance	Artificial nets are inherently not fault tolerant, since the information corrupted in the memory cannot be retrieved.	They exhibit fault tolerance since the infom1ation is distributed in· the connections throughout the network. Even though if few connections are not working the information is still preserved due to distributed nature of the encoded information
Control Mechanism	There is a control unit, which monitors all the activities of computing.	There is no central control for processing information in the brain. The neuron acts based on the information locally available and transmits its output to the neurons connected to it. There is no special control mechanism external to the computing task.

2.0.0 **Keyword definitions**

Action potential: The pulse of electrical potential generated across the membrane of a neuron (or an axon) following the application of a stimulus greater than threshold value.

Axon: The output fiber of a neuron, which carries the information in the form of action potentials to other neurons in the network.

Dendrite: The input line of the neuron that carries a temporal summation of action potentials to the soma.

Excitatory neuron: A neuron that transmits an action potential that has excitatory (positive) influence on the recipient nerve cells.

Inhibitory neuron: A neuron that transmits an action potential that has inhibitory (negative) influence on the recipient nerve cells.

Lateral inhibition: The local spatial interaction where the neural activity generated by one neuron is suppressed by the activity of its neighbors.

Latency: The time between the application of the stimulus and the peak of the resulting action potential output.

Refractory period: The minimum time required for the axon to generate two consecutive action potentials.

Neural state: A neuron is active if it's firing a sequence of action potentials.

Neuron: The basic nerve cell for processing biological information.

Soma: The body of a neuron, which provides aggregation, thresholding and nonlinear activation to dendrite inputs.

Synapse: The junction point between the axon (of a pre-synaptic neuron) and the dendrite (of a post-synaptic neuron). This acts as a memory (storage) to the past-accumulated experience (knowledge).

1.3.0 History of artificial neural networks

- The field of neural networks is not new. The first formal definition of a synthetic neuron model based on the highly simplified considerations of the biological model proposed by McCulloch and Pitts in 1943. The McCulloch-Pitts (MP) neuron model resembles what is known as a binary logic device.
- The next major development, after the MP neuron model was proposed, occurred in 1949, when D.O. Hebb proposed a learning mechanism for the brain that become the starting point for artificial neural networks (ANN) learning (training) algorithms. He postulated that as the brain learns, it changes its connectivity patterns.
- The idea of learning mechanism was first incorporated in ANN by E. Rosenblatt 1958.
- By introducing the least mean squares (LMS) learning algorithm, Widrow and Hoff developed in 1960 a model of a neuron that learned quickly and accurately. This model was called ADALINE for ADAptive LInear NEuron. The applications of ADALINE and its extension to MADALINE (for Many ADALINES) include pattern recognition, weather forecasting, and adaptive controls. The monograph on learning machines by Nils Nilsson (1965) summarized the developments of that time.
- In 1969, research in the field of ANN suffered a serious setback. Minsky and Papert published a book on perceptrons in which they proved that single layer neural networks have limitations in their abilities to process data, and are capable of any mapping that is linearly separable. They pointed out, carefully applying mathematical techniques, that are logical Exclusive-OR (XOR) function could not be realized by perceptrons.
- Further, Minsky and Papert argued that research into multi-layer neural networks would be unproductive. Due to this pessimistic view of Minsky and Papert, the field of ANN entered into an almost total eclipse for nearly two decades. Fortunately, Minsky and Papert's judgment has been disapproved; multi-layer perceptron networks can solve all nonlinear separable problems.

- Nevertheless, a few dedicated researchers such as Kohonen, Grossberg, Anderson and Hopfield continued their efforts.
- The study of learning in networks of threshold elements and of the mathematical theory of neural networks was pursued by Sun - Ichi – Amari (1972, 1977). Also Kunihiko Fukushima developed a class of neural network architectures known as neocognitrons in 1980.
- There have been many impressive demonstrations of ANN capabilities: a network has been trained to convert text to phonetic representations, which were then converted to speech by other means (Sejnowsky and Rosenberg 1987); other network can recognize handwritten characters (Burr 1987); and a neural network based image-compression system has been devised (Cottrell, Munro, and Zipser 1987). These all use the backpropagation network, perhaps the most successful of the current algorithms. Backpropagation, invented independently in three separate research efforts (Werbos 1974, Parker 1982, and Rumelhart, Hinton and Williams 1986) provides a systematic means for training multi-layer networks, thereby overcoming limitations presented by Minsky.

1.4.0 Characteristics of ANN

Artificial neural networks are biologically inspired; that is, they are composed of elements that perform in a manner that is analogous to the most elementary functions of the biological neuron. The important characteristics of artificial neural networks are learning from experience, generalize from previous examples to new ones, and abstract essential characteristics from inputs containing irrelevant data.

1.4.1 Learning

The NNs learn by examples. Thus, NN architectures can be 'trained' with known

examples of a problem before they are tested for their 'inference' capability on unknown instances of the problem. They can, therefore, identify new objects previously untrained. ANN can modify their behavior in response to their environment. Shown a set of inputs (perhaps with desired outputs), they self-adjust to produce consistent responses. A wide variety of training algorithms has been discussed in later units.

1.4.2 Parallel operation

The NNs can process information in parallel, at high speed, and in a distributed manner.

1.4.3 Mapping

The NNs exhibit mapping capabilities, that is, they can map input patterns to their associated output patterns.

1.4.4 Generalization

The NNs possess the capability to generalize. Thus, they can predict new outcomes from past trends. Once trained, a network's response can be to a degree, insensitive to minor variations in its input. This ability to see through noise and distortion to the pattern that lies within is vital to pattern recognition in a real-world environment. It is important to note that the ANN generalizes automatically as a result of its structure, not by using human intelligence embedded in the form of adhoc computer programs.

1.4.5 Robust

The NNs are robust systems and are fault tolerant. They can, therefore, recall full patterns from incomplete, partial or noisy patterns

1.4.6 Abstraction

Some ANN's are capable of abstracting the essence of a set of inputs. i.e. they can extract features of the given set of data, for example, convolution neural networks are used to extract different features from images like edges, dark spots, shapes ..etc. Such networks are trained for feature patterns based on which they can classify or cluster the given input set.

1.4.7 Applicability

ANN's are not a panacea. They are clearly unsuited to such tasks as calculating the payroll. They are preferred for a large class of pattern-recognition tasks that conventional computers do poorly, if at all.

1.5.0 Applications

Neural networks are preferred when the task is related to large-amount data processing. The following are the potential applications of neural networks:

- Classification
- Prediction
- Data Association
- Data Conceptualization
- Data Filtering
- Optimization

In addition to the above fields, neural networks can apply to the fields of Medicine, Commercial and Engineering, etc.

2.3.0 Artificial Neuron Model

The artificial neuron is developed to mimic the first-order characteristics of the biological neuron. In similar to the biological neuron, the artificial neuron receives many inputs representing the output of other neurons. Each input is multiplied by a corresponding weight, analogous to the synaptic strength. All of these weighted inputs are then summed and passed

through an activation function to determine the neuron input. This artificial neuron model is shown in Fig.2.3.

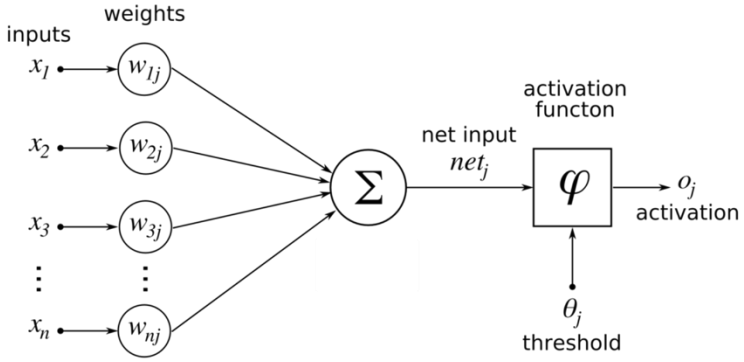


Fig. 2.3 An artificial neuron

The mathematical model of the artificial neuron may written as

$$u(t) = w_1x_1 + w_2x_2 + w_3x_3 + \ldots\ldots\ldots + w_nx_n$$

$$= \sum_{i=1}^{n} w_i x_i + \theta = \sum_{i=0}^{n} w_i x_i \qquad (2.1)$$

Assuming $w_0 = \theta$ and $x_0 = 1$

$$y(t) = f[u(t)] \qquad (2.2)$$

where f[.] is a nonlinear function called as the activation function, the input-output function or the transfer function. In equation (2.1) and Fig.2.3, $[x_0, x_1, \ldots, x_n]$ represent the inputs, $[w_0, w_1, \ldots, w_n]$ represents the corresponding synaptic weights. In vector form, we can represent the neural inputs and the synaptic weights as

$X = [x_0, x_1, \ldots, x_n]^T$, and $W = [w_0, w_1, \ldots, w_n]$

Equations (2.1) and (2.2) can be represented in vector form as:

$$U = WX \qquad (2.3)$$

$$Y = f[U] \qquad (2.4)$$

The activation function f[.] is chosen as a nonlinear function to emulate the nonlinear behavior of conduction current mechanism in biological neuron. The behavior of the artificial neuron depends both on the weights and the activation function. Sigmoidal functions are the commonly used activation functions in multi-layer static neural networks. Other types of activation functions are discussed in later units.

2.1.0 McCulloch-Pitts Model

The McCulloch-Pitts model of the neuron is shown in Fig. 2.4(a). The inputs x_i, for i= 1,2, . . .,n are 0 or 1, depending on the absence or presence of the input impulse at instant k. The neuron's output signal is denoted as **Y.** The firing rule for this model is defined as follows

$$\mathbf{Y}^{k+1} = \begin{cases} 1 & if \quad \sum_{i=1}^{n} w_i x_i^k \geq T \\ 0 & if \quad \sum_{i=1}^{n} w_i x_i^k < T \end{cases} \qquad 1$$

where super script k = 0,1,2,. . . . , denotes the discrete – time instant, and w_i is the multiplicative weight connecting the i'th input with the neuron's membrane. Note that w_i = +1 for excitatory synapses, w_i = -1 for inhibitory synapses for this model, and T is the neuron's threshold value, which needs to be exceeded by the weighted sum of the signals for the neuron to fire.

This model can perform the basic operations NOT, OR and AND, provided its weights and thresholds are approximately selected. Any multivariable combinational function can be implemented using either the NOT and OR, or alternatively the NOT and AND, Boolean operations. Examples of three-input NOR and NAND gates using the McCulloch-Pitts neuron model are shown in (Fig.2.4 (b) and Fig 2.4(c)).

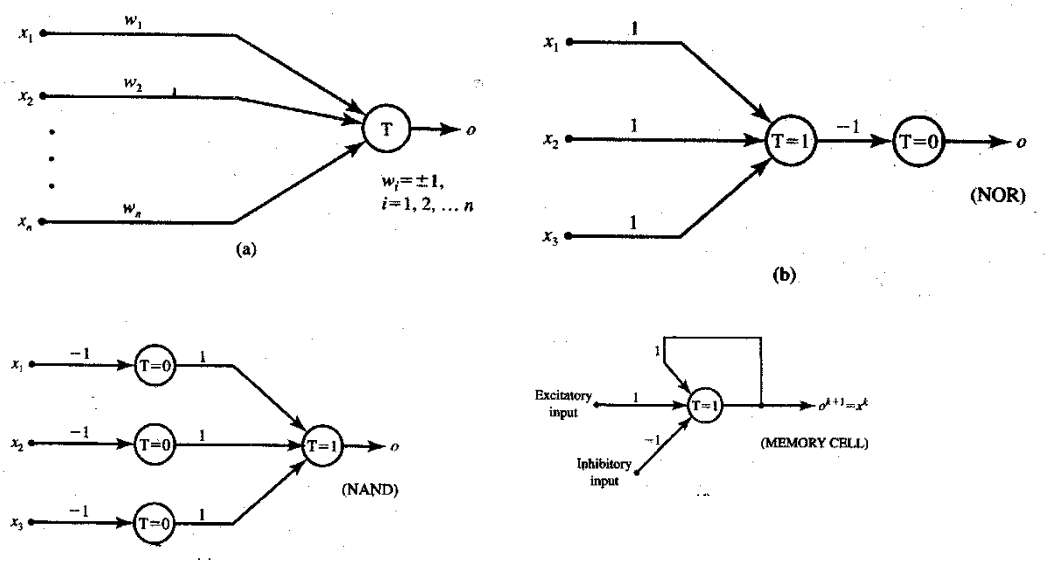


Fig.2.4 McCulloch-Pitts

3.1.0 **Operations of Artificial Neuron**

The schematic diagram of artificial neuron is shown in Fig.3.1. The artificial neuron mainly performs two operations, one is the summing of weighted net input and the second is passing the net input through an activation function. The activation function also called nonlinear function and some time transfer function of artificial neuron.

The net input of j^{th} neuron may be written as

$NET_j = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n + \theta_j$ (3.1)

where θ_j is the threshold of j^{th} neuron,

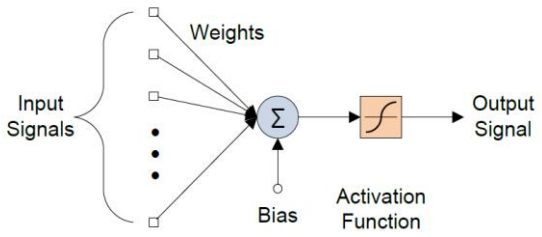


Fig. 3.1 Artificial neuron.

$X = [x_1\ x_2 \ldots x_n]$ in the input vector and $W = [w_1\ w_2 \ldots w_n]$ is the synaptic weight vector. The NET_j signal is processed by an activation function F to produce the neuron's output signal.

$$OUT_j = F(NET_j) \qquad (3.2)$$

What functional form for F(.) should be selected? Can it be – square root, log, e^x , x^3 and so on. Mathematicians and computer scientists, however, have found that, the sigmoid (S-shaped) function is more useful. In addition to this sigmoid function, there are number of other function are using in artificial neural networks. They are discussed in the next section.

<u>Activation Functions</u>

3.2.0 **Types of activation functions**

The behavior of the artificial neuron depends both on the synaptic weights and the activation function. Sigmoid functions are the commonly

used activation functions in multi-layered feed forward neural networks. Neurons with sigmoid functions bear a greater resemblance to the biological neurons than with other activation functions. The other feature of sigmoid function is that it is differentiable, and gives a continuous values output. Some of the popular activation functions are described below along with their other characteristics.

1. **Sigmoid function (Unipolar sigmoid)** . The characteristics of this function is shown in Fig.3.2 and its mathematical description is

$$y(x) = f(x) = \frac{1}{1+e^{-x}} \qquad (3.1)$$

and its range of signal is 0<y<1.

The derivative of the above function is written as

$$y'(x) = f'(x) = f(x)\,(1-f(x)) \qquad (3.2)$$

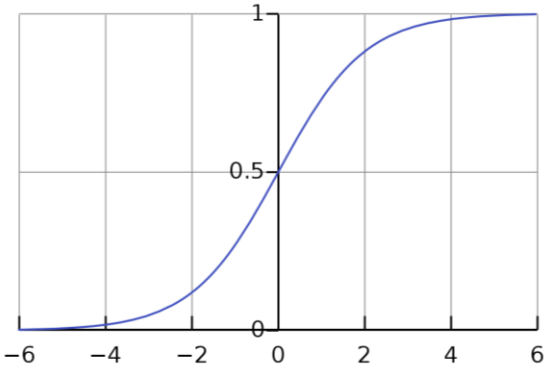


Fig.3.2 A sigmoid (S-shaped) function

Moreover, sigmoid functions are continuous and monatonic, and remain finite even as x approaches to ±∞. Because they are monatonic, they also provide for more efficient network training.

2. **Hyperbolic tangent (bipolar sigmoid) function**. The characteristics of this function is shown in Fig.3.3 and its mathematical description is

$$y(x)=f(x)=\tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad (3.3)$$

range of signal is -1<y<1 and its derivative can be obtained as

$$y' = f'(x) = 1-[f(x)]^2 \qquad (3.4)$$

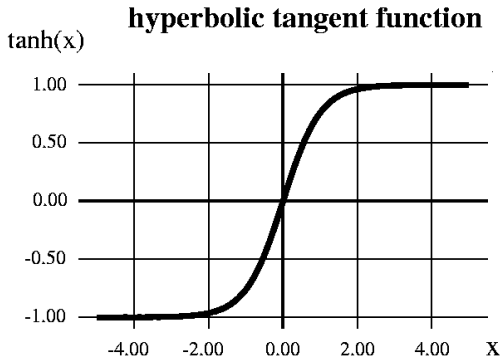


Fig.3.3 A hyperbolic tangent (bipolar sigmoid) function

3. **Radial basis function**: The Gaussian function is the most commonly used " radially symmetric" function, the characteristics of this function is shown in Fig.3.4 and its mathematical description is

$$y = f(x) = \exp\left(\frac{-x^2}{2}\right) \qquad (3.5)$$

Range of signal is 0<y<1 and its derivative can be obtained as

$$y' = f'(x) = -x\,\exp\left(\frac{-x^2}{2}\right) \qquad (3.6)$$

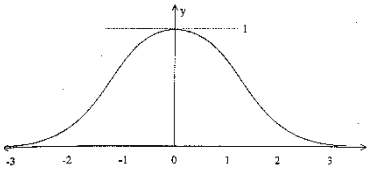


Fig.3.4. A Gaussian function

The function has maximum response, f(x) =1, when the input is x=0, and the response decreases to f(x)=0 as the input approaches x=±∞.

4. **Hard Limiter**: The hard limiter function is the mostly used in classification of patterns, the characteristics of this function is shown in Fig.3.5 and its mathematical description is

$$f(u(t))= \textbf{sign}\ (u(t)) = \begin{cases} +1, & u(t) > 0 \\ -1 & u(t) < 0 \end{cases} \qquad (3.7)$$

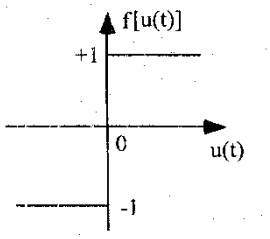


Fig. 3.5 Hard Limiter

This function is not differentiable. Therefore it cannot be used for continuation type of applications.

5. **Piecewise linear**: The piecewise linear function characteristics is shown in Fig.3.6 and its mathematical description is

$$f(u(t)) = \begin{cases} +1 & \text{if } gu > 1 \\ gu & \text{if } |gu| < 1 \\ -1 & \text{if } gu < -1 \end{cases} \qquad (3.8)$$

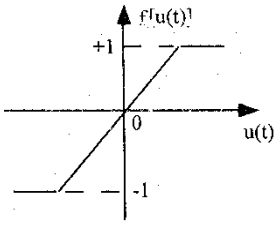


Fig. 3.6 Piecewise linear

6. **Linear**: The Linear function characteristics is shown in Fig.3.8 and its mathematical description is

$$f(u(t)) = gu(t) \qquad (3.10)$$

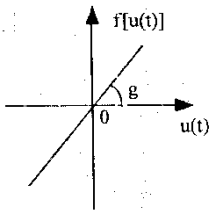


Fig. 3.8 Linear function

It is differentiable and is mostly used for output nodes of the networks.

3.3.0 **Selection of activation function**

The selection of an activation function is depends upon the application to which the neural network used and also the level (in which layer) neuron. The activation functions that are mainly used are the sigmoid (unipolar sigmoidal), the hyperbolic tangent (bipolar sigmoid), radial basis function, hard limiter and linear functions. The sigmoid and hyperbolic tangent functions perform well for the prediction and the process-forecasting types of problems. However, they do not perform as well for classification networks. Instead, the radial basis function proves more effective for those networks, and highly recommended function for any problems involving fault diagnosis and feature categorization. The hard limiter suits well for

classification problems. The linear function may be used at output layer in feed forward networks.

Classification of Artificial Neural Networks

4.0.0 Introduction

The development of artificial neuron based on the understanding of the biological neural structure and learning mechanisms for required applications. This can be summarized as (a) development of neural models based on the understanding of biological neurons, (b) models of synaptic connections and structures, such as network topology and (c) the learning rules. Researchers are explored different neural network architectures and used for various applications. Therefore the classification of artificial neural networks (ANN) can be done based on structures and based on type of data.

A single neuron can perform a simple pattern classification, but the power of neural computation comes from neuron connecting networks. The basic definition of artificial neural networks as physical cellular networks that are able to acquire, store and utilize experimental knowledge has been related to the network's capabilities and performance. The simplest network is a group of neurons are arranged in a layer. This configuration is known as single layer neural networks. There are two types of single layer networks namely, feed-forward and feedback networks. The single linear neural (that is activation function is linear) network will have very limited capabilities in solving nonlinear problems, such as classification etc., because their decision boundaries are linear. This can be made little more complex by selecting nonlinear neuron (that is activation function is nonlinear) in single layer neural network. The nonlinear classifiers will have complex shaped decision boundaries, which can solve complex problems. Even nonlinear neuron single layer networks will have limitations in classifying more close nonlinear classifications and fine control problems. In recent studies shows that the nonlinear neural networks in multi-layer structures can simulate more complicated systems, achieve smooth control, complex classifications and have capabilities beyond those of single layer networks. In this unit we discuss first classifications, single layer neural networks and multi-layer neural networks. The structure of a neural network refers to how its neurons are interconnected.

4.2.0 Applications

Having different types artificial neural networks, these networks can be used to broad classes of applications, such as (i) Pattern Recognition and Classification, (ii) Image Processing and Vision, (iii) System Identification and Control, and (iv) Signal Processing. The details of suitability of networks as follows:

(i) **Pattern Recognition and Classification**: Almost all networks can be used to solve these types of problems.

(ii) **Image Processing and Vision**: The following networks are used for the applications in this area: Static single layer networks, Dynamic single layer networks, BAM, ART, Counter – propagation networks, First – Order dynamic networks.

(iii) **System Identification and Control**: The following networks are used for the applications in this area: Static multi layer networks, Dynamic multi layer networks of types time-delay and Second-Order dynamic networks.

(iv) **Signal Processing:** The following networks are used for the applications in this area: Static multi layer networks of type RBF, Dynamic multi layer networks of types Cellular and Second-Order dynamic networks.

4.3.0 Single Layer Artificial Neural Networks

The simplest network is a group of neuron arranged in a layer. This configuration is known as single layer neural networks. This type of network comprises of two layers, namely the input layer and the output layer. The input layer neurons receive the input signals and the output layer neurons receive the output signals. The synaptic links carrying the weights connect every input neuron to the output neuron but not vice-versa. Such a network is said to'be *feedforward* in type or acyclic in nature. Despite the two layers, the network is termed single layer, since it is the output layer alone Which performs computation. The input layer merely transmits the signals to the output layer. Hence, the name single layer feedforward network. Figure 4.2 illustrates an example network.

There are two types of single layer networks namely, feed-forward and feedback networks.

4.3.1 **Feed forward single layer neural network**

Consider m numbers of neurons are arranged in a layer structure and each neuron receiving n inputs as shown in Fig.4.2.

Output and input vectors are respectively

$$O = [o_1 \; o_2 \; \ldots \; o_m]^T \qquad (4.1)$$

$$X = [x_1 \; x_2 \; \ldots \; x_n]^T$$

Weight w_{ji} connects the j^{th} neuron with the i^{th} input. Then the activation value for j^{th} neuron as

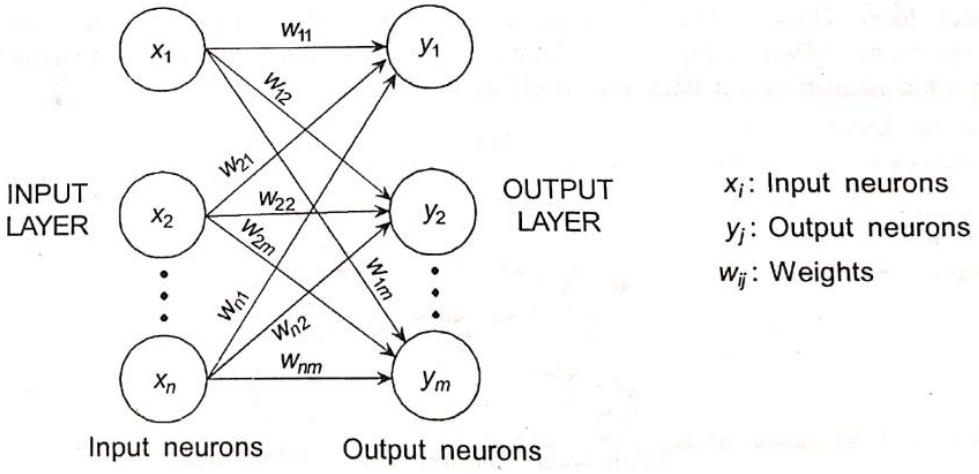
$$net_j = \sum_{i=1}^{n} w_{ji} x_i \qquad \text{for } j = 1,2, \ldots m \qquad (4.2)$$

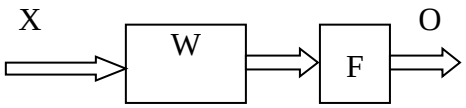
The following nonlinear transformation involving the activation function $f(net_j)$, for j=1,2,. . .m, completes the processing of X. The transformation will be done by each of the m neurons in the network.

$$o_j = f(W_j^t X), \quad \text{for } j = 1, 2, \ldots m \qquad (4.3)$$

where weight vector w_j contains weights leading toward the j^{th} output node and is defined as follows

$$W_j = [w_{j1} \; w_{j2} \; \ldots \; w_{jn}] \qquad (4.4)$$





(b) Block diagram of single layer network

Fig. 4.2 Feed forward single layer neural network

Introducing the nonlinear matrix operator F, the mapping of input space X to output space O implemented by the network can be written as

$$O = F(W X) \qquad (4.5a)$$

Where W is the weight matrix and also known as connection matrix and is represented as

$$W = \begin{bmatrix} w_{11} & w_{12} & . & . & w_{1n} \\ w_{21} & w_{22} & . & . & w_{2n} \\ . & . & . & . & . \\ . & . & . & . & . \\ w_{m1} & w_{m2} & . & . & w_{mn} \end{bmatrix} \qquad (4.5b)$$

The weight matrix will be initialized and it should be finalized through appropriate training method.

$$F(.) = \begin{bmatrix} f(.) & 0 & . & . & 0 \\ 0 & f(.) & . & . & 0 \\ . & . & . & . & . \\ . & . & . & . & . \\ 0 & 0 & . & . & f(.) \end{bmatrix} \qquad (4.5c)$$

The nonlinear activation function $f(.)$ on the diagonal of the matrix operator F(.) operates component-wise on the activation values net of each neuron. Each activation value is, in turn, a scalar product of an input with the respective weight vector, X is called input vector and O is called output vector. The mapping of an input to an output is shown as in (4.5) is of the feed-forward and instantaneous type, since it involves no delay between the input and the output . Therefore the relation (4.5a) may be written in terms of time t as

$$O(t) = F(W X(t)) \qquad (4.6)$$

This type of networks can be connected in cascade to create a multiplayer network. Though there is no feedback in the feedforward network while mapping from input X(t) to output O(t), the output values are compared with the "teachers" information, which provides the desired output values. The error signal is used for adapting the network weights. The details about will be discussed in later units.

Example: To illustrate the computation of output O(t), of the single layer feed forward network consider an input vector X(t) and a network weight matrix W (say initialized weights), given below. Consider the neurons uses the hard limiter as its activation function.

$$X = [-1 \;\; 1 \;\; -1]^T \qquad W = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 0 & -2 \\ 0 & 1 & 0 \\ 0 & -1 & -3 \end{bmatrix}$$

The out vector may be obtained from the (4.5) as

$$O = F(W X) = [\, \text{sgn}(-1-1) \;\; \text{sgn}(+1+2) \;\; \text{sgn}(1) \;\; \text{sgn}(-1+3) \,]$$

$$= [\, -1 \;\; 1 \;\; 1 \;\; 1]$$

The output vector of the above single layer feedforward network is = [-1 1 1 1].

4.4 Types of connections

There are three different options are available for connecting nodes to one another, as shown in Fig. 4.5 They are:

Intralayer connection: The output from a node fed into other nodes in the same layer.

Interlayer connection: The output from a node in one layer fed into nodes in other layer.

Recurrent connection: The outputs from a node fed into itself.

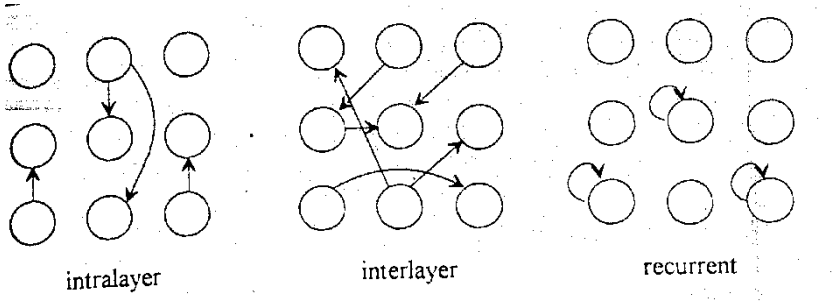


Fig. 4.5 Different connection option of Neural Networks

In general, when building a neural network its structure will be specified. In engineering applications, suitable and mostly preferred structure is the interlayer connection type topology. Within the interlayer connections, there are two more options: (i) feed forward connections and (ii) feedback connections, as shown in Fig. 4.6.

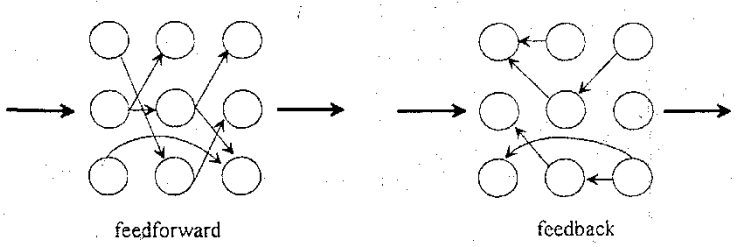


Fig. 4.6 Feed forward and feed back connections of Neural Networks

4.4.0 **Multi Layer Artificial Neural Networks**

Cascading a group of single layers networks can form the feed forward neural network. This type networks also known as feed forward multi layer neural network. In which, the output of one layer provides an input to the subsequent layer. The input layer gets input from outside; the output of input layer is connected to the first hidden layer as input. The output layer receives its input from the last hidden layer. The multi layer neural network provides no increase in computational power over single layer neural networks unless there is a nonlinear activation function between layers. Therefore, due to nonlinear activation function of each neuron in hidden layers, the multi layer neural networks able to solve many of the complex problems, such as, nonlinear function approximation, learning generalization, nonlinear classification etc.

A multi layer neural network consists of input layer, output layer and hidden layers. The number of nodes in input layer depends on the number of inputs and the number of nodes in the output layer depends upon the number of outputs. The designer selects the number of hidden layers and neurons in respective layers. According to the **Kolmogorov's** theorem single hidden layer is sufficient to map any complicated input – output mapping.

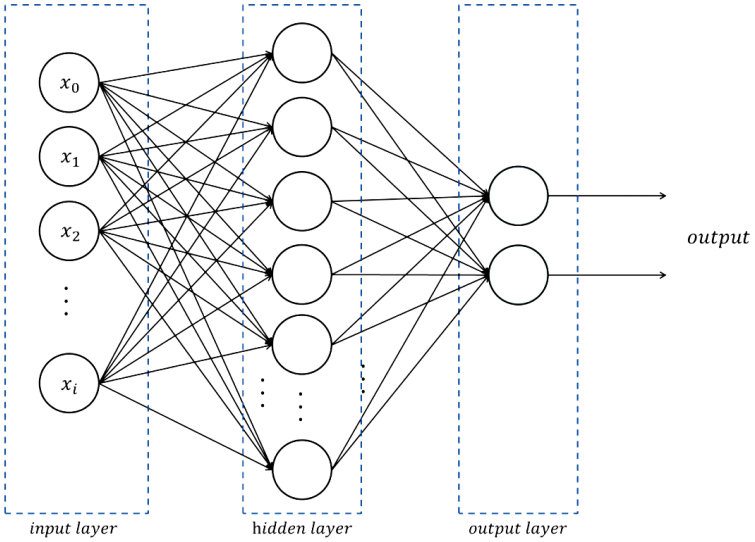


Fig. 4.7. Multilayer feedforward network

Recurrent Networks

These networks differ from feedforward network architectures in the sense that there is atleast one feedback loop. Thus, in these networks, for example, there could exist .one layer with feedback connections as shown in Fig. 4.8. There could also be neurons with self-feedback links, i.e. the output of a neuron is fed back into itself as input.

The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps.

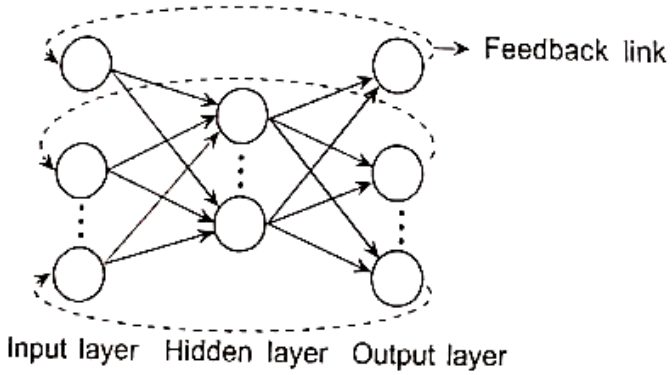


Fig. 4.8. A recurrent neural network.

Training Methods of Artificial Neural Networks

6.0.0 Introduction

The dynamics of neuron consists of two parts. One is the dynamics of the activation state and the second one is the dynamics of the synaptic weights. The Short Term Memory (STM) in neural networks is modeled by the activation state of the network and the Long Term Memory is encoded the information in the synaptic weights due to learning. The main property of artificial neural network is that, the ability of the learning from its environment and history. The network learns about its environment and history through its interactive process of adjustment applied to its synaptic weights and bias levels. Generally, the network becomes more knowledgeable about its environment and history, after completion each iteration of learning process. It is important to distinguish between representation and learning. Representation refers to the ability of a perceptron (or other network) to simulate a specified function. Learning requires the existence of a systematic procedure for adjusting the network weights to produce that function. Here we will discuss most of popular learning rules.

6.1.0 Definition of learning

There are too many activities associated with the notion of learning and we define learning in the context of neural networks [1] as

"Learning is a process by which the free parameters of neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the way the parameter changes takes place"

Based on the above definition the learning process of ANN can be divided into the following sequence of steps:

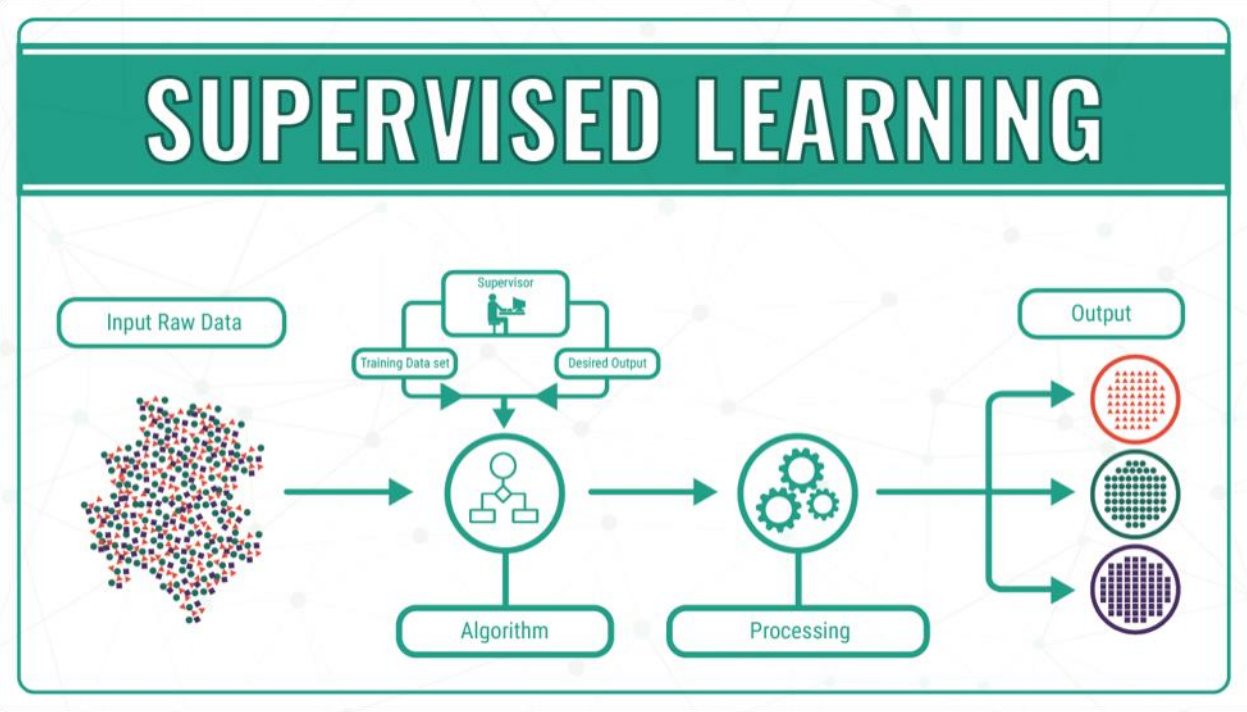
1. The ANN is stimulated by an environment.
2. The ANN undergoes changes in its free parameters as a result of the above stimulation.
3. The ANN responds in a new way to the environment because of the changes that have occurred in its internal structure.

6.1.1 Types of Learning Methods / Learning Strategies

A set of defined rules for the solution of a learning problem is called algorithm. There are different approaches to train an ANN. Most of the methods fall into one of two classes namely supervised learning and unsupervised learning.

***Supervised learning*:** An external signal known as teacher controls learning and incorporates information.

Supervised training requires the pairing of each input vector with a target vector representing the desired output; together these are called a training pair. Usually a network is trained over several such training pairs. An input vector is applied, the output of the network is calculated and compared to the corresponding target vector and the difference (error) is fed back through the network and weights are changed according to an algorithm that tends to minimize the error. The vectors of the training set are applied sequentially, and errors are calculated, and weights adjusted for each vector until the error for the entire training set is at the acceptably low value.



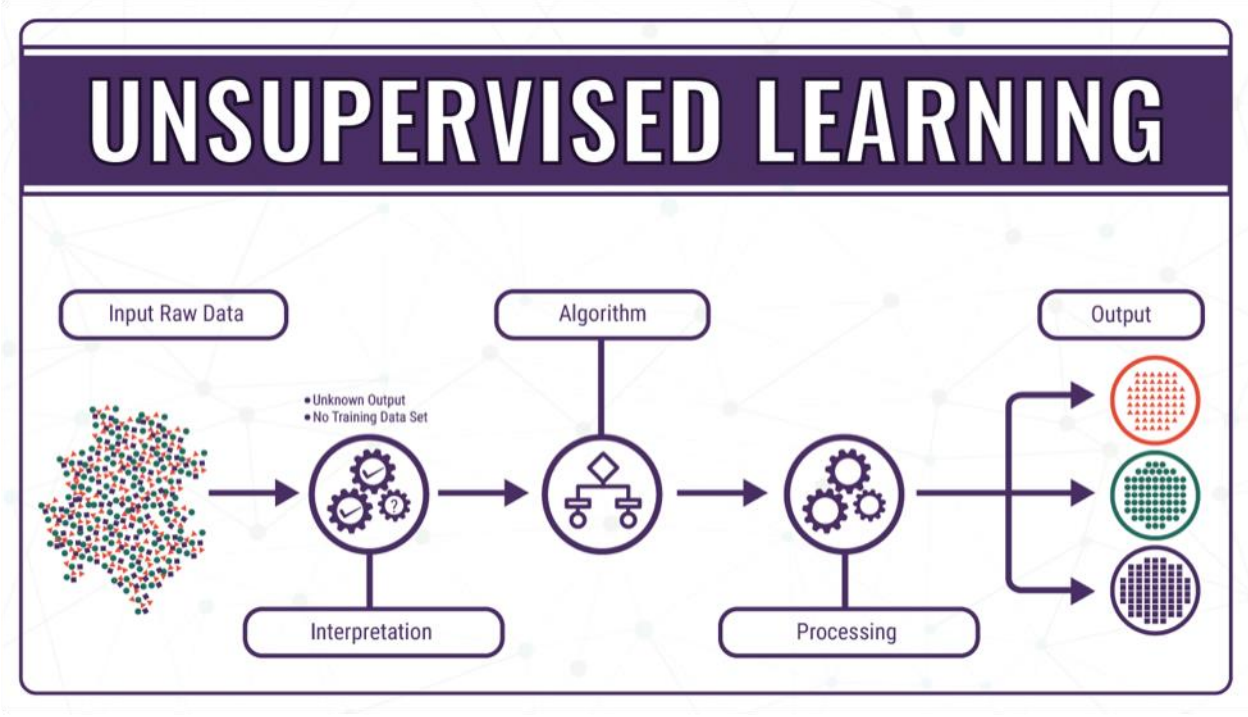
***Unsupervised learning*:** No external signal (teacher) is used in the learning process. The neural network relies upon both internal and local information.

In a neural net, if for the training input vectors, the target output is not known, the training method adopted is called as unsupervised training. The net may modify the weight so that the most similar input vector is assigned to the same output unit. The net is found to form a exemplar or code book vector for each cluster formed.

Unsupervised training is a far more plausible model of training in the biological system. Developed by Kohonen (1984) and many others, it requires no target vector for the outputs, and hence, no comparisons to predetermined ideal responses. The training set consists solely of input vectors. The training algorithm modifies network weights to produce output vectors that consistent; i.e., both application of one of the training vectors and application of a vector that is sufficiently similar to it will produce the same patterns of outputs.

- The training process, therefore, extracts the statistical properties of the training set and group's similar vector into classes.

- Applying a vector from a given class as a input will produce a specific output vector, but there is no way to determine prior to training which specific output pattern will be produced by a given input vector class. Hence, the outputs of such a network must generally be transformed into a comprehensible form subsequent to the training process.



Reinforcement Training

In this method, a teacher is also assumed to be present, but the right answer is not presented to the network. Instead, the network is only presented with an indication of whether the output answer is right or wrong. The network must then use this information to improve its performance. Reinforcement learning is a very general approach to learning that can be applied when the knowledge required to apply supervised learning is not available. If sufficient information is available, the reinforcement learning can readily handle a specific problem. However, it is usually better to use other methods such as supervised and unsupervised learning, because they are more direct, and their underlying analytical basis is usually well understood.

Reinforcement training is related to supervised training. The output in this case may not be indicated as the desired output, but the condition whether it is 'success' (+ 1) or 'failure' (0) may be indicated. Based on this, error may be calculated, and the training process may be continued. The error signal produced from reinforcement training is found to be binary. Reinforcement learning attempts to learn the input-output mapping through trial and error with a view to maximize a performance index called the reinforcement signal. The system knows whether the output is correct or not but does not know the correct output.

6.1.2 Types of basic learning mechanisms

In general the training of any artificial neural network has to use one of the following basic learning mechanisms.

The basic learning mechanisms of neural networks are:

Hebbian learning rule

Hebb's learning rule is the oldest and most famous of all. learning rules. It states that, "when an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased". This learning can also be called correlational learning. '

This statement may be split into a two-part rule:

(i) If two neurons on either side of a synapse are activated simultaneously, then the strength of that synapse is selectively increased.

(ii) If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened-or eliminated.

In this, the input-output pattern pairs (X_i, Y_i) are associated by the weight matrix W, known as the correlation matrix. It is computed as

$$\Delta W = \Sigma X_i . Y_i^T$$

The simplest form (element wise) of Hebbian learning is described by, $\Delta w = x_i . y$

3.3.2 Perceptron Learning Rule

For the perceptron learning rule, the learning signal is the difference between the desired and actual neuron's response. This type of learning is supervised.

The fact that the weight vector is perpendicular to the plane separating the input patterns during the learning processes, can be used to interpret the degree of difficulty of training a perceptron for different types of input.

The perceptron learning rule states that for a finite 'n' number of input training vectors,

x (n) where n = 1 to N

each with an associated target value, *t (n) where n = 1 to N*

which is +1 or -1, and an-activation function f(y_{in}), where, (ϑ is the threshold)

$$y = \begin{cases} 1 & \text{if} \quad y > \vartheta \\ 0 & \text{if} \quad -\vartheta \le y_{in} \le \vartheta \\ -1 & \text{if} \quad y_{in} < -\vartheta \end{cases}$$

the weight updation is given by

$W_{new} = W_{old} + tx$ *if* $y \neq t$, *then*

$W_{new} = W_{old}$ *if* $y = t$, (i.e. no change in weights)

The perceptron learning rule is of central importance for supervised learning of neural networks. The weights can be initialized at any values in this method. There is a ***perceptron learning rule convergence theorem*** *which states, "If there is a weight vector w* such that f(x(p) w*) = t(p) for all p, then for any starting vector w_1 the perceptron learning rule will converge to a weight vector that gives the correct response for all training patterns, and this will be done in a finite number of steps".*

3.3.4 Competitive Learning Rule

In this learning, the output neurons of a neural network compete among themselves to become active. The basic idea behind this rule is that there are a set of neurons that are similar in all aspects except for some randomly distributed synaptic weights, and therefore respond differently to a given set of input patterns. However, a limit is imposed on the strength of the neurons. This rule has a mechanism that permits the neurons to compete for the right to respond to a given subset of inputs, such that only one output neuron, or only one neuron per group, is active at a time. The winner neuron during competition is called **winner—takes-all neuron**.

For a neuron P to be the winning neuron, its induced local field v_p, for a given particular input pattern must be largest among all the neurons in the network. The output signal of winning neuron is set to one and the signals that lose the competition are set to zero. Hence,

$$N = \begin{cases} 1 & \text{if } v_p > v_q \quad \text{for all } q, \quad p \neq q \\ 0 & \text{otherwise} \end{cases}$$

This rule is suited for unsupervised network training. The winner-takes-all or the competitive learning is used for learning statistical properties of inputs. This uses the standard Kohonen learning rule.

A neuron then learns by shifting weights from its inactive to active input modes. If a neuron does not respond to a particular input pattern, no learning takes place in that neuron. If a particular neuron wins the competition, its corresponding weights are adjusted.

Using standard competitive rule, the change Δw_{ij} is given as,

$$\Delta w_{ij} = \begin{cases} \propto (x_j - w_{ij}) & \text{if neuron i wins the competition} \\ 0 & \text{if neuron i losses the competition} \end{cases}$$

where α is the learning rate. This rule has the effect of moving the weight vector w, of winning neuron towards the input pattern x. Through competitive learning, the neural network can perform clustering.

3.3.6 Boltzmann Learning

The learning is a **stochastic learning**. A neural net designed based on this learning is called Boltzmann learning. In this learning, the neurons constitute a recurrent structure and they work in binary form. This learning is characterized by an energy function, E, the value of which is determined by the particular states occupied by the individual neurons of the machine, given by,

$$E = -\frac{1}{2}\sum_{i}\sum_{j} w_{ji}\, x_i x_j \qquad where \;\; i \neq j$$

where x_i is the state of neuron i and w_{ji} is the weight from neuron i to neuron j. The value $i \neq j$ means that none of the neurons in the machine has self-feedback. The operation of machine is performed by choosing a neuron at random:

The neurons of this learning process are divided into two groups; visible and hidden. In visible neurons there is an interface between the network and the environment in which it operates but in hidden neurons, they Operates independent of the environment. The visible neurons might be clamped onto specific states determined by the environment, called as clamped condition. On the other hand, there is free-running condition, in which all the neurons are allowed to operate freely.

PERCEPTRONS

8.0.0 **Introduction**

We know that perceptron is one of the early models of artificial neuron. It was proposed by Rosenblatt in 1958. It is a single layer neural network whose weights and biases could be trained to produce a correct target vector when presented with the corresponding input vector. The perceptron is a program that learns *concepts*, i.e. it can learn to respond with *True* (1) or *False* (0) for inputs we present to it, by repeatedly "studying" examples presented to it. The training technique used is called *the perceptron learning rule*. The perceptron generated great interest due to its ability to *generalize* from its training vectors and work with randomly distributed connections. Perceptrons are especially suited for simple problems in pattern classification. In this also we give the perceptron convergence theorem.

8.1.0 Perceptron Model

In the 1960, perceptrons created a great deal of interest and optimism. Rosenblatt (1962) proved a remarkable theorem about perceptron learning. Widrow (Widrow 1961, 1963, Widrow and Angell 1962, Widrow and Hoff 1960) made a number of convincing demonstrations of perceptron like systems. Perceptron learning is of the supervised type. A perceptron is trained by presenting a set of patterns to its input, one at a time, and adjusting the weights until the desired output occurs for each of them.

The schematic diagram of perceptron is shown inn Fig. 8.1. Its synaptic weights are denoted by $w_1, w_2, \ldots w_n$. The inputs applied to the perceptron are denoted by $x_1, x_2, \ldots . x_n$. The externally applied bias is denoted by b.

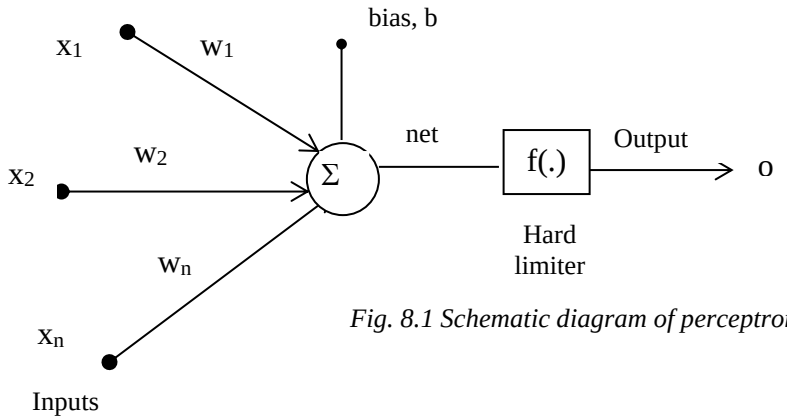


Fig. 8.1 Schematic diagram of perceptron

The net input to the activation of the neuron is written as

$$net = \sum_{i=1}^{n} w_i x_i + b \qquad (8.1)$$

The output of perceptron is written as $o = f(net)$ (8.2)

where f(.) is the activation function of perceptron. Depending upon the type of activation function, the perceptron may be classified into two types

i) Discrete perceptron, in which the activation function is ***hard limiter*** or ***sgn(·)*** function

ii) Continuous perceptron, in which the activation function is sigmoid function, which is differentiable. The input-output relation may be rearranged by considering w_0=b and fixed bias x_0 = 1.0. Then

$$net = \sum_{i=0}^{n} w_i x_i = WX \qquad (8.3)$$

where $W = [w_0, w_1, w_2, \ldots . w_n]$ and $X = [x_0, x_1, x_2, \ldots x_n]^T$.

The learning rule for perceptron has been discussed in unit 7. Specifically the learning of these two models is discussed in the following sections.

8.2.0 Single Layer Discrete Perceptron Networks

For discrete perceptron the activation function should be *hard limiter* or *sgn()* function. The popular application of discrete perceptron is a pattern classification. To develop insight into the behavior of a pattern classifier, it is necessary to plot a map of the decision regions in n-dimensional space, spanned by the n input variables. The two decision regions separated by a hyper plane defined by

$$\sum_{i=0}^{n} w_i x_i = 0 \qquad (8.4)$$

This is illustrated in Fig. 8.2 for two input variables x_1 and x_2, for which the decision boundary takes the form of a straight line.

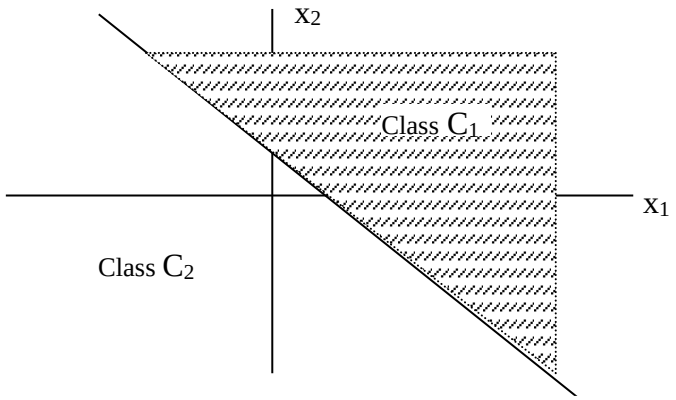


Fig. 8.2 Illustration of the hyper plane (in this example, a straight lines) as decision boundary for a two dimensional, two-class patron classification problem.

For the perceptron to function properly, the two classed C_1 and C_2 must be linearly separable. This in turn, means that the patterns to be classified must be sufficiently separated from each other to ensure that the decision surface consists of a hyper plane. This is illustrated in Fig. 8.3.

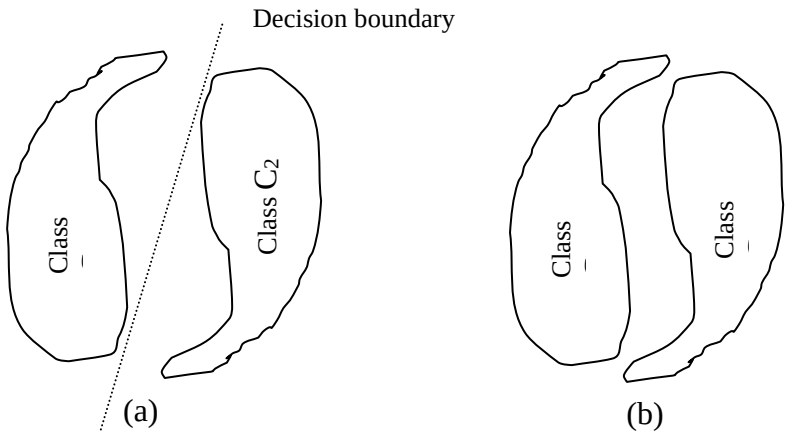


Fig. 8.3 (a) A pair of linearly separable patterns
(b) A pair of nonlinearly separable

In Fig. 8.3(a), the two classes C_1 and C_2 are sufficiently separated from each other to draw a hyper plane (in this it is a straight line) as the decision boundary. If however, the two classes C_1 and C_2 are allowed to move too close to each other, as in Fig. 8.3 (b), they become nonlinearly separable, a situation that is beyond the computing capability of the perceptron.

Suppose then that the input variables of the perceptron originate from two linearly separable classes. Let æ$_1$ be the subset of training vectors $X_1(1), X_1(2), \ldots$, that belongs to class C_1 and æ$_2$ be the subset of train vectors $X_2(1), X_2(2), \ldots$, that belong to class C_2. The union of æ$_1$ and æ$_2$ is the complete training set æ. Given the sets of vectors æ$_1$ and æ$_2$ to train the classifier, the training process involves the adjustment of the W in such a way that the two classes C_1 and C_2 are linearly separable. That is, there exists a weight vector W such that we may write,

$$\left.\begin{array}{l} WX > 0 \quad \text{for every input vector X belonging to class } C_1 \\ WX \le 0 \quad \text{for every input vector X belonging to class } C_2 \end{array}\right\} \qquad (8.5)$$

In the second condition, it is arbitrarily chosen to say that the input vector X belongs to class C_2 if WX = 0.

The algorithm for updating the weights may be formulated as follows:

1. If the k^{th} member of the training set, X_k is correctly classified by the weight vector W(k) computed at the k^{th} iteration of the algorithm, no correction is made to the weight vector of perceptron in accordance with the rule.

   $W^{k+1} = W^k$ if $W^k X_k > 0$ and X_k belongs to class C_1 (8.6)

   $W^{k+1} = W^k$ if $W^k X_k \le 0$ and X_k belongs to class C_2 (8.7)

2. Otherwise, the weight vector of the perceptron is updated in accordance with the rule.

   $W^{(k+1)T} = W^{kT} - \eta X_k$ if $W^k X_k > 0$ and X_k belongs to class C_2 (8.8a)

   $W^{(k+1)T} = W^{kT} + \eta X_k$ if $W^k X_k \le 0$ and X_k belongs to class C_1 (8.8b)

where the learning rule parameter η controls the adjustment applied to the weight vector. Equations (8.8a) and (8.8b) may be written general expression as

$$W^{(k+1)} = W^{kT} + \frac{\eta}{2}(d_k - o_k)X_k \qquad (8.9)$$

8.2.1 Summary of the discrete perceptron training algorithm

Given are P training pairs of patterns

{X_1, d_1, X_2, d_2, ... Xp, dp}, where X_i is (n×1), d_i is (1×1), i = 1, 2, . . . P. Define w_0=b is bias and X_0 = 1.0 , then the size of augmented input vector is X_i ((n+1)×).

In the following, k denotes the training step and p denotes the step counter with the training cycle.

Step 1: $\eta > 0$ is chosen and define E_{max}.

Step 2: Initialize the weights at small random values, W = [w_{ij}] , augmented size is (n+1)×1 and initialize counters and error function as:

$k \leftarrow 1, \qquad p \leftarrow 1 \qquad E \leftarrow 0.$

Step 3: The training cycle begins. Apply input and compute the output:

$X \leftarrow X_p, \qquad d \leftarrow d_p, \qquad o \leftarrow \text{sgn}(WX)$

Step 4: Update the weights: $W^T \leftarrow W^T + \eta(d - o)X$

Step 5: Compute the cycle error: $E \leftarrow \frac{1}{2}(d - o)^2 + E$

Step 6: If p < P, the p $\leftarrow$ p+1, k $\leftarrow$ k+1 and go to step 3, otherwise go to step 7.

Step 7: The training cycle is completed. For E< E_{max} terminates the training session with output weights and k. If E > E_{max} , then E $\leftarrow$0, p$\leftarrow$1 and enter the new training cycle by going to step 3.

In general, a continuous perceptron element with sigmoidal activation function will be used to facilitate the training of multi layer feed forward networks used for classification and recognition.

8.3.0 Single-Layer Continuous Perceptron networks

In this, the concept of an error function in multidimensional weight space has been introduced. Also the hard limiter (*sgn(.)*) with weights will be replaced by the continuous perceptron. By introduction of this continuous activation function, there are two advantages (i) finer control over the training procedure and (ii) differential characteristics of the activation function, which is used for computation of the error gradient.

The gradient or steepest descent is used in updating weights starting from any arbitrary weight vector W, the gradient $\nabla E(W)$ of the current error function is computed. The next value of W as obtained by moving in the direction of the negative gradient along the multidimensional error surface. Therefore the relation of modification of weight vector may be written as

$$W^{(k+1)T} = W^{kT} - \eta \nabla E(W^k) \qquad (8.10)$$

where η is the learning constant and is the positive constant and the superscript k denotes the step number. Let us define the error function between the desired output d_k and actual output o_k as

$$E_k = \frac{1}{2}(d_k - o_k)^2 \qquad (8.11a)$$

or

$$E_k = \frac{1}{2}\left[d_k - f(W^k X)\right]^2 \qquad (8.11b)$$

where the coefficient ½ in from of the error expression is only for convenience in simplifying the expression of the gradient value and it does not effect the location of the error function minimization. The error minimization algorithm (8.10) requires computation of the gradient of the error function (8.11) and it may be written as

$$\nabla E(W^k) = \frac{1}{2}\nabla\left[d - f(net_k)\right]^2 \qquad (8.12)$$

The n+1 dimensional gradient vector is defined as

$$\nabla E(W^k) = \begin{bmatrix} \frac{\partial E}{\partial w_0} \\ \frac{\partial E}{\partial w_1} \\ . \\ . \\ \frac{\partial E}{\partial w_n} \end{bmatrix} \qquad (8.13)$$

Using (8.12), we obtain the gradient vector as

$$\nabla E(W^k) = -(d_k - o_k) f'(net_k) \begin{bmatrix} \frac{\partial (net_k)}{\partial w_0} \\ \frac{\partial (net_k)}{\partial w_1} \\ . \\ . \\ \frac{\partial (net_k)}{\partial w_n} \end{bmatrix} \qquad (8.14)$$

Since $net_k = W^k X$, we have

$$\frac{\partial (net_k)}{\partial w_i} = x_i, \qquad \text{for } i = 0, 1, \ldots n. \qquad (8.15)$$

(x_0=1 for bias element) and equation (8.15) can be written as

$$\nabla E(W^k) = -(d_k - o_k) f'(net_k) X \qquad (8.16a)$$

or

$$\frac{\partial E}{\partial w_i} = -(d_k - o_k) f'(net_k) x_i \qquad \text{for } i = 0, 1, \ldots n \qquad (8.16b)$$

$$\therefore \Delta w_i^{\,k} = -\eta \nabla E(W^k) = \eta (d_k - o_k) f'(net_k) x_i \qquad (8.17)$$

Equation (8.17) is the training rule for the continuous perceptron. Now the requirement is how to calculate $f'(net)$ in terms of continuous perceptron output. Consider the bipolar activation function f(net) of the form

$$f(net) = \frac{2}{1+\exp(-net)} - 1 \qquad (8.18)$$

Differentiating the equation (8.18) with respect to net: $f'(net) = \dfrac{2 \times \exp(-net)}{[1+\exp(-net)]^2}$ (8.19)

The following identity can be used in finding the derivative of the function.

$$\frac{2 \times \exp(-net)}{[1+\exp(-net)]^2} = \frac{1}{2}(1 - o^2) \qquad (8.20)$$

The relation (8.20) may be verified as follows:

$$\frac{1}{2}(1 - o^2) = \frac{1}{2}\left[1 - \left(\frac{1-\exp(-net)}{1+\exp(-net)}\right)^2\right] \qquad (8.21)$$

The right side of (8.21) can be rearranged as

$$\frac{1}{2}\left[1 - \left(\frac{1-\exp(-net)}{1+\exp(-net)}\right)^2\right] = \frac{2\exp(-net)}{[1+\exp(-net)]^2} \qquad (8.22)$$

This is same as that of (8.20) and now the derivative may be written as

$$\therefore f'(net_k) = \frac{1}{2}(1 - o_k^{\,2}) \qquad (8.23)$$

The gradient (8.16a) can be written as $\nabla E(W^k) = -\dfrac{1}{2}(d_k - o_k)(1 - o_k^{\,2})X$ (8.24)

and the complete delta training for the bipolar continuous activation function results from (8.24) as

$$W^{(k+1)T} = W^{kT} + \frac{1}{2}\eta (d_k - o_k)(1 - o_k^{\,2}) X_k \qquad (8.25)$$

where k denotes the reinstated number of the training step.

The weight adjustment rule (8.25) corrects the weights in the same direction as the discrete perceptron learning rule as in equation (8.8). The main difference between these two is the presence of the moderating factor $(1-o_k^2)$. This scaling factor is always positive and smaller than 1. Another main difference between the discrete and continuous perceptron training is that the

discrete perceptron training algorithm always leads to a solution for linearly separable problems. In contrast to this property, the negative gradient-based training does not guarantee solutions for linearly separable patterns.

8.3.1 Summary of the Single Continuous Perceptron Training Algorithm

Given are P training pairs

$\{X_1, d_1, X_2, d_2, \ldots . X_p, d_p\}$, where X_i is $((n+1)\times 1)$, d_i is (1×1), for i = 1, 2, . . . P

$$X_i = \begin{bmatrix} x_{i0} \\ x_{i1} \\ . \\ . \\ x_{in} \end{bmatrix}, \text{ where } x_{i0} = 1.0 \text{ (bias element)}$$

Let k is the training step and p is the step counter within the training cycle.

Step 1: $\eta > 0$ and $E_{max} > 0$ chosen.

Step 2: Weights are initialized at W at small random values, W = $[w_{ij}]$ is $(n+1)\times 1$. Counter and error function are initialized.

$k \leftarrow 1, \qquad p \leftarrow 1 \qquad E \leftarrow 0.$

Step 3: The training cycle begins. Input is presented and output is computed.

$X \leftarrow X_p, \qquad d \leftarrow d_p, \qquad o \leftarrow f(WX)$

Step 4: Weights are updated: $W^T \leftarrow W^T + \frac{1}{2}\eta(d-o)(1-o^2)X$

Step 5: Cycle error is computed: $E \leftarrow \frac{1}{2}(d-o)^2 + E$

Step 6: If p < P, the $p \leftarrow p+1$, $k \leftarrow k+1$ and go to step 3, otherwise go to step 7.

Step 7: The training cycle is completed. For $E < E_{max}$ terminated the training session with output weights, k and E. If $E \geq E_{max}$, then $E \leftarrow 0$, $p \leftarrow 1$ and enter the new training cycle by going to step 3.

8.4.0 Perceptron Convergence Theorem

(Separate document sent)

8.5.0 Problems and Limitations of the perceptron training algorithms

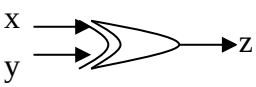
It may be difficult to determine if the caveat regarding linear separability is satisfied for the particular training set at hand. Further more, in many real world situations the inputs are often time varying and may be separable at one time and not at another. Also, these is no statement in the proof of the perceptron learning algorithm that indicates how many steps will be required to train the network. It is small consolation; to know that training will only take a finite number of steps if the time it takes is measured in geological units.

Further more, there is no proof that perceptron training algorithm is faster than simply trying all possible adjustment of the weights; in some cases this brute force approach may be superior.

8.5.1 Limitations of perceptrons

There are limitations to the capabilities of perceptrons however. They will learn the solution, if there is a solution to be found. First, the output values of a perceptron can take on only one of two values (True or False). Second, perceptrons can only classify *linearly separable* sets of vectors. If a straight line or plane can be drawn to separate the input vectors into their correct categories, the input vectors are linearly separable and the perceptron will find the solution. If the vectors are not linearly separable learning will never reach a point where all vectors are classified properly. The most famous example of the perceptron's inability to solve problems with linearly non-separable vectors is the Boolean exclusive-OR problem.

Consider the case of the exclusive-or (XOR) problem. The XOR logic function has two inputs and one output, how below.



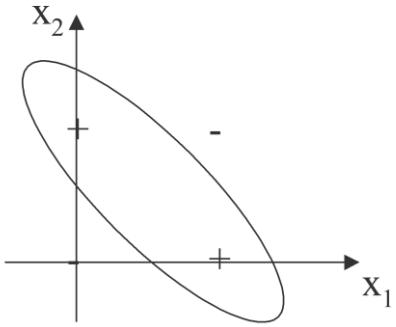
(a) Exclusive –OR gate

x	y	Z
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 8.4 (b) Truth Table

It produces an output only if either one or the other of the inputs is on, but not if both are **off** or both are **on.** It is shown in above table. We can consider this has a problem that we want the perceptron to learn to solve; output a 1 of the x is **on** and y is **off** or y is **on** and x is **off**, otherwise output a '0'. It appears to be a simple enough problem.

We can draw it in pattern space as shown in Fig. (8.5). The x-axis represents the value of x, the y-axis represents the value of y. The shaded circles represent the inputs that produce an output of 1, whilst the un-shaded circles show the inputs that produce an output of 0. Considering the shaded circles and un-shaded circles as separate classes, we find that, we cannot draw a straight line to separate the two classes. Such patterns are known as linearly inseparable since no straight line can divide them up successfully. Since we cannot divide them with a single straight line, the perceptron will not be able to find any such line either, and so cannot solve such a problem. In fact, a single-layer perceptron cannot solve any problem that is linearly inseparable.



2.9.2 ADALINE Network

The Adaptive Linear Neural Element Network framed by Bernard Widrow of Stanford University, makes use of supervised learning. Figure 2.19 illustrates a simple ADALINE network. Here, there is only one output neuron and the output values are bipolar (–1 or +1). However, the inputs x_i could be binary, bipolar or real valued. The bias weight is w_0 with an input link of $x_0 = +1$. If the weighted sum of the inputs is greater than or equal to 0 then the output is 1 otherwise it is –1.

The supervised learning algorithm adopted by the network is similar to the perceptron learning algorithm. Devised by Widrow-Hoff (1960), the learning algorithm is also known as the Least Mean Square (LMS) or Delta rule. The rule is given by

$$W_i^{new} = W_i^{old} + \alpha\,(t - y)x_i \tag{2.15}$$

where, α is the learning coefficient, t is the target output, y is the computed output, and x_i is the input.

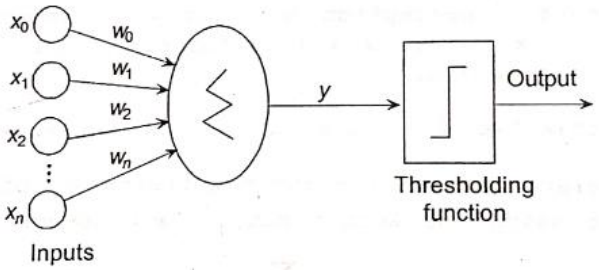


Fig. 2.19 A simple ADALINE network.

ADALINE network has had the most successful applications because it is used virtually in all high speed modems and telephone switching systems to cancel the echo in long distance communication circuits.

2.9.3 MADALINE Network

A MADALINE (Many ADALINE) network is created by combining a number of ADALINES. The network of ADALINES can span many layers. Figure 2.20 illustrates a simple MADALINE

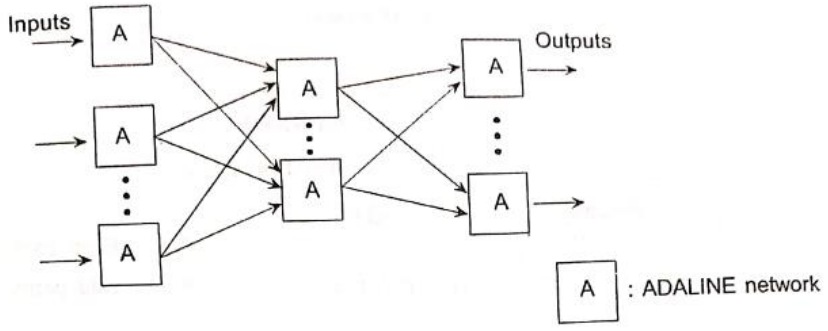


Fig. 2.20 MADALINE network.

network. *The use of multiple ADALINES helps counter the problem of non-linear separability*. For example, the MADALINE network with two units exhibits the capability to solve the XOR problem (refer Fig. 2.21). In this, each ADALINE unit receives the input bits x_1, x_2 and the bias input $x_0 = 1$ as its inputs. The weighted sum of the inputs is calculated and passed on to the bipolar threshold units. The logical 'and'ing (bipolar) of the two threshold outputs are computed to obtain the final output. Here, if the threshold outputs are both +1 or –1 then the final output is +1. If the threshold outputs are different, (i.e.) (+1, –1) then the final output is –1. Inputs which are of even parity produce positive outputs and inputs of odd parity produce negative outputs. Figure 2.22 shows the decision boundaries for the XOR problem while trying to classify the even parity inputs (positive outputs) from the odd parity inputs (negative outputs).

9.0.0 Backpropagation Algorithm

(Separate document Attached)

9.1.0 Selection of parameters of feedforword backpropagation network

Initial Weights

It will influence whether the net reaches a global (or only a local) minima of the error and if so how rapidly it converges. If the initial weight is too large the initial input signals to each hidden or output unit will fall in the saturation region where the derivative of the sigmoid has a very small value (f(net) = 0). If initial weights are too small, the net input to a hidden or output umt will approach zero, Which then causes extremely slow learning. To get the best result the initial welghts (and biases) are set to random numbers between -O.5 and 0.5 or between -1 and 1. The initialization of welghts (bias) can be done ramomly and there is also a specific approach.

Selection of Learning Rate

A high learning rate leads to rapid learning but the weights may oscillate, while a lower learning rate leads to slower learning. Methods suggested for adopting learning rate are as follows:
(i) Start with a high learning rate and steadily decrease it. Changes in the weight vector must be small in order to reduce oscillations or any divergence.
(ii) A simple suggestion is to increase the learning rate in order to improve performance and t0 decrease the learning rate in order to worsen the performance.
(iii) Another method is to double the learning rate until the error value worsens.

Emperical results suggest to use value between 0.4 to 0.9 for better output.

How Long Should We Train a Net?

The motivation for applying back propagation net is to achieve a balance between memorization and generalization; It is not necessarily advantageous to continue training until the error reaches a minimum value. The two disjoint sets of data used during training are:
(i) set of training patterns
(ii) set of training-testing patterns.
The weight adjustments are based on the training patterns. As along as error the for validation decreases training continues. Whenever the error begins to increase. the net is starting to memorize the training patterns. At this point training is terminated.

Number of training pairs

A simple thumb rule used to find the required minimum number of training pairs to get best results is

$$p > \frac{w}{1-a} \; log \frac{n}{1-a}$$

Where p = number of training patterns required
w = number of weights to be trained
n = number of nodes and
a = expected accuracy of the test set

Number of Hidden Units

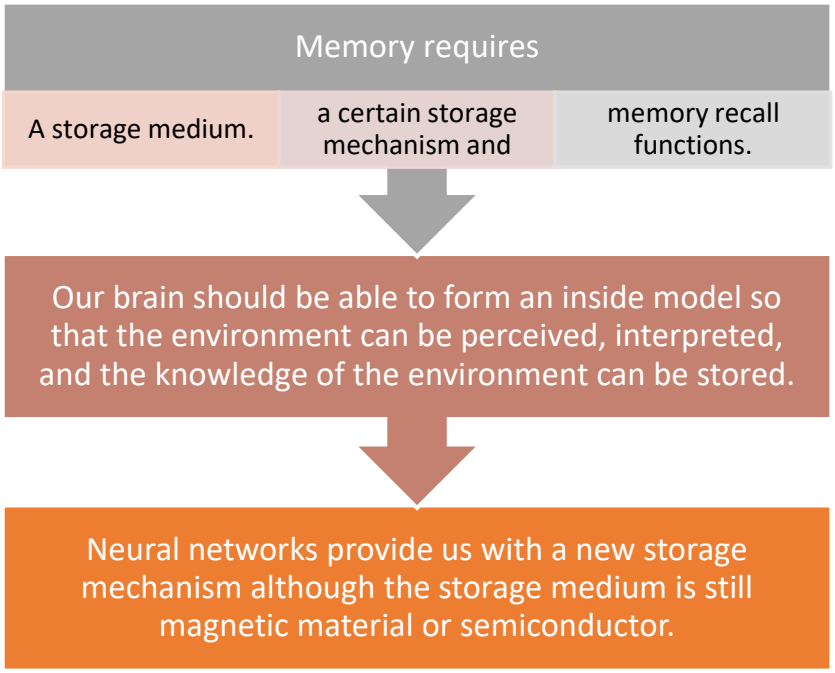
It is difficult to find the best suited hidden layers or units in each hidden layer for a given application. There is no fixed rule derived till date to find it. However, it can be seen that a n – input, m-output function requires at most $2n+1$ hidden units. Trial and error method is followed by varying the number of neurons in each hidden layers around this $2n+1$.

Associative Memory

1. Introduction

How does our memory works? As for all memory there should be a storage medium. For a type of storage medium, there must be a certain storage mechanism and memory recall functions. In some sense, for the purposes of recognition, reasoning, and behavior control, our brain should be able to form an inside model so that the environment can be perceived, interpreted, and the knowledge of the environment can be stored. Neural networks provide us with a storage mechanism, although the storage medium is still magnetic material or semiconductor. It is recognized, that brain stores information by modification of synaptic junctions between neurons. Such synaptic junction modifications are distributed. We cannot identify which modification corresponds to which piece of information.

Artificial memory has been in the form of localized memory before the beginning of neural network research. Think about the way we make files of documents, or the method in which we write our diary. In the diary all messages are indexed by date. We put those messages of certain events where the date of the event is indicated. When we look for a message, we should first find the date, which is acting as the address here.



Nowadays, information storage inside a computer is still in the form of localized memory. Localized memory is simple and easy to implement. It does not matter whether we use semiconductor memory or magnetic memory, information recording and retrieval are all done through addresses. Addresses are formed in terms of a minimum unit, such as byte for semiconductor memory and the sector for magnetic storage. When searching for a particular data item, we should first find its correct address. This is exactly the same as the situation when you are wandering among bookshelves in a library looking for a book, you cannot get it until you know where it is.

Nevertheless, there is no any evidence that in our brain the information storage is localized. We recall information by its content. Memory that works this way is referred to as content-addressable memory. One may say that our libraries and database systems work in a content-addressable memory manner too, because every time we seek a particular book or data item, we use key words, which are an abstract of the content. Indeed, key words are used in libraries and databases, but these key words should be first converted into an absolute address through indexes and then the book or data item can be physically retrieved. It is virtually content-addressable, and actually localized.

Our memories function in what is called an *associative* or *content-addressable* fashion. That is, a memory does not exist in some isolated fashion, located in a particular set of neurons. All memories are in some sense strings of memories - you remember someone in a variety of ways - by the color of their hair or eyes, the shape of their nose, their height, the sound of their voice, or perhaps by the smell of a favorite perfume. Thus memories are stored in *association* with one another. These different sensory units lie in completely separate parts of the brain, so it is clear that the memory of the person must be distributed throughout the brain in some fashion. Indeed, PET scans reveal that during memory recall there is a pattern of brain activity in many widely different parts of the brain.

Notice also that it is possible to access the full memory (all aspects of the person's description for example) by initially remembering just one or two of these characteristic features. We access the memory by its contents not by where it is stored in the neural pathways of the brain. This is very powerful; given even a poor photograph of that person we are quite good at reconstructing the persons face quite accurately. This is very different from a traditional computer where specific facts are located in specific places in computer memory. If only partial information is available about this location, the fact or memory cannot be recalled at all.

Paradigms of Associate Memory

The common paradigm of memory here may be described as follows. There is some underlying collection of data which is ordered and interrelated in some way and which is stored in memory. The data may be thought of, therefore, as forming a stored pattern. In the recollection examples below, it is the cluster of memories associated with the celebrity or the phase in childhood. In the case of character recognition, it is the parts of the letter (pixels) whose arrangement is determined by an archetypical version of the letter. When part of the pattern of data is presented in the form of a sensory cue, the rest of the pattern (memory) is recalled or *associated* with it. Notice that it often doesn't matter which part of the pattern is used as the cue, the whole pattern is always restored.

2. General concepts of associative memory

A general block diagram of associative memory is depicted in Fig.1. An input pattern $\boldsymbol{a} = \{a_1, ..., a_N\}$ is input into associative memory which is characterized by $\boldsymbol{M}$. The output of the memory is pattern $\boldsymbol{b} = \{b_1, ..., b_p\}$. The memory system can be written as $\boldsymbol{b} = f(a, \boldsymbol{M})$.

Assume that for a set of input patterns $\boldsymbol{A} = \{\boldsymbol{a^1}, \ldots, \boldsymbol{a^k}\}$, and with associative memory we are able to get a set of output patterns $\boldsymbol{B} = \{\boldsymbol{b^1}, \ldots, \boldsymbol{b^k}\}$with a one-to-one correspondence

$$\boldsymbol{a^1} \rightarrow \boldsymbol{b^1}$$

$$\cdot$$

$$\cdot$$

$$\boldsymbol{a^k} \rightarrow \boldsymbol{b^k}$$

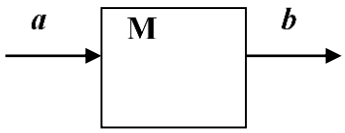


Fig.1. Block diagram of associative memory

We then call this memory system hetero-associative memory. Auto-associative memory can be considered a special case of hetero-associative memory. Temporal association is another extension of

hetero-association where a time series can be associated with its initial state, when its initial state is presented the whole time series is recalled.

Auto-Associative Memory: Assumes $y_i = x_i$ and implements a mapping, ϕ, of X to X such that $\phi(x_i) = x_i$, and if some arbitrary x is closer x_i than to any other x_j, j = 1, . . .L, then $\phi(x) = x_i$.

Hetero-associative memory: Implements a mapping, ϕ, of X to Y such that $\phi(x_i) = y_i$, and if an arbitrary x is closer to x_i than to any other x_j, j = 1, . . .L, then $\phi(x) = y_i$. In this and the following definitions closer means with respect to Hamming distance.

Interpolative Associative Memory: Implements a mapping, ϕ, of X to Y such that $\phi(x_i) = y_i$, but if the input vectors differs from one of the exemplars by the vector d, such that $x = x_i + d$, then the output of the memory also differs from one of the exemplars by some vector e; $\phi(x) = \phi(x_i+d) = y_i + e$.

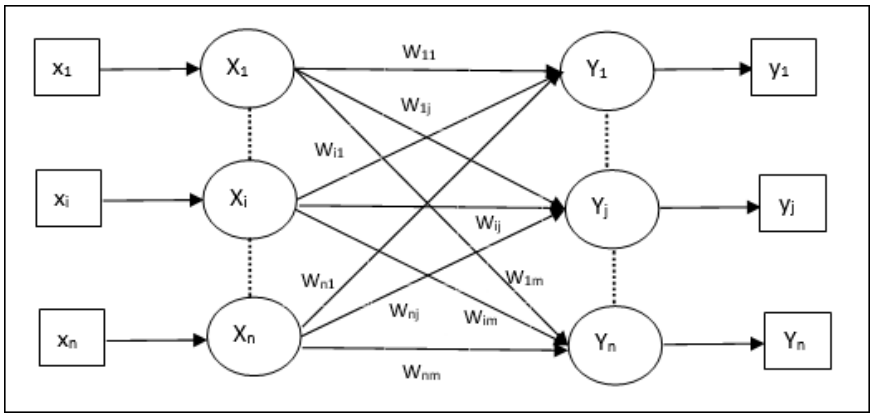
Hamming Distance: The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different.

3. Auto Associative Memory

This is a single layer neural network in which the input training vector and the output target vectors are the same. The weights are determined so that the network stores a set of patterns.

Architecture

As shown in the following figure, the architecture of Auto Associative memory network has **'n'** number of input training vectors and similar **'n'** number of output target vectors.



Training Algorithm

For training, this network is using the Hebb or Delta learning rule.

Step 1 – Initialize all the weights to zero as $w_{ij} = 0$ $i=1$ *to* n, $j=1$ *to* n

Step 2 – Perform steps 3-4 for each input vector.

Step 3 – Activate each input unit as follows –

$$x_i=s_i \quad (i=1 \text{ to } n)$$

Step 4 – Activate each output unit as follows –

$$y_j=s_j \quad (j=1 \text{ to } n)$$

Step 5 – Adjust the weights as follows –

$$w_{ij}(new)=w_{ij}(old)+x_i y_j$$

Testing Algorithm

Step 1 – Set the weights obtained during training for Hebb's rule.

Step 2 – Perform steps 3-5 for each input vector.

Step 3 – Set the activation of the input units equal to that of the input vector.

Step 4 – Calculate the net input to each output unit **j = 1 to n**

$$y_{inj} = \sum_{i=1}^{n} x_i w_{ij}$$

Step 5 – Apply the following activation function to calculate the output

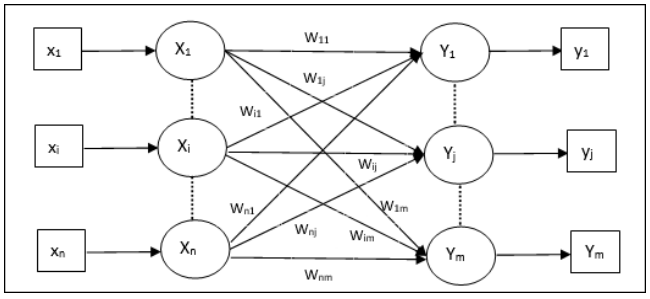
$$y_j = f(y_{inj}) = \begin{cases} +1 & if\ y_{inj} > 0 \\ -1 & if\ y_{inj} \leqslant 0 \end{cases}$$

4. Hetero Associative memory

Like Auto Associative Memory network, this is also a single layer neural network. However, in this network the input training vector and the output target vectors are not the same. The weights are determined so that the network stores a set of patterns. Hetero associative network is static in nature, hence, there would be no non-linear and delay operations.

Architecture

As shown in the following figure, the architecture of Hetero Associative Memory network has **'n'** number of input training vectors and **'m'** number of output target vectors.



Training Algorithm

For training, this network is using the Hebb or Delta learning rule.

Step 1 – Initialize all the weights to zero as $\mathbf{w_{ij}} = \mathbf{0}, i=1\ to\ n,\ j=1\ to\ m$

Step 2 – Perform steps 3-4 for each input vector.

Step 3 – Activate each input unit as follows –

$$x_i=s_i\ \ (i=1\ to\ n)$$

Step 4 – Activate each output unit as follows –

$$y_j=s_j(j=1\ to\ m)$$

Step 5 – Adjust the weights as follows –

$$w_{ij}(new)=w_{ij}(old)+x_i y_j$$

Testing Algorithm

Step 1 – Set the weights obtained during training for Hebb's rule.

Step 2 – Perform steps 3-5 for each input vector.

Step 3 – Set the activation of the input units equal to that of the input vector.

Step 4 – Calculate the net input to each output unit **j = 1 to m;**

$$y_{inj} = \sum_{i=1}^{n} x_i w_{ij}$$

Step 5 – Apply the following activation function to calculate the output

$$y_j = f(y_{inj}) = \begin{cases} +1 & if\ y_{inj} > 0 \\ 0 & if\ y_{inj} = 0 \\ -1 & if\ y_{inj} < 0 \end{cases}$$

5. The Discrete Hopfield Net

The discrete Hopfield net is a fully interconnected neural net with each unit connected to every other unit. The net has symmetric weights with **no self-connections** i.e. all the diagonal elements of the weight matrix of a Hopfield net are zero.

$W_{ij}=W_{ji}$ and $W_{ii} = 0$

The two main differences between Hopfield and iterative auto associative net (recurrent associative net) are that, in the Hopfield net,

• only one unit updates its activation at a time, and,

• each unit continues to receive an external signal in addition to the signal from the other units in the net.

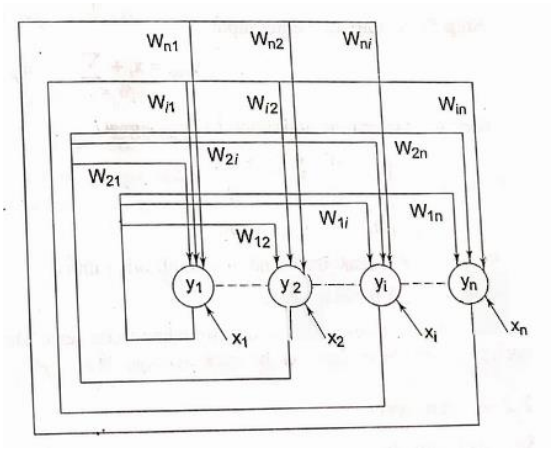
The asynchronous discrete time updating of the units allows a function known as an energy function or Lyapunov function to be found for the net. This function proves that the net will converge to a stable set of activations. The formulation. of the discrete Hopfield net- shows the usefulness of the net as a content addressable memory.

The crucial property of the Hopfield network, which renders it useful for simulating memory recall, is the following:

- we are *guaranteed* that the pattern will settle down after a long enough time to some fixed pattern.
- Certain nodes will be always "*on*" and others "*off*".
- Furthermore, it is possible to arrange that these *stable firing patterns* of the network correspond to the desired memories we wish to store. The reason for this is somewhat technical but we can proceed by analogy.

Architecture

The architecture shown in Fig. consists of 'n' number of x input neurons and Y output neurons. It should be noted that apart from receiving a signal from input, the y_1 neuron receives signal from its other output neurons also. This is the same for the all other output neurons. Thus, there exists a feedback output being returned to each output neuron. That is why the Hopfield network is called a feedback network.



Training Algorithm

Discrete Hopfield net is described for both binary as well as bipolar vector patterns.

The weight matrix to store the set of **binary input patterns** $s(p), p = 1, \ldots P$, where

$$s(p) = (s_1(p), \ldots s_i(p), \ \ldots \ s_n(p))$$

can be determined with the help of Hebb rule discussed in 6.2.1.

The weight matrix can be determined by the formula.

$$w_{ij} = \sum_p (2s_i(p) - 1)(2s_j(p) - 1) \text{ for } i \neq j \text{ and } W_{ii} = 0$$

For **bipolar input patterns**, the weight matrix is given by,

$$w_{ij} = \sum_p s_i(p) s_j(p) \text{ for } i \neq j \text{ and } W_{ii} = 0$$

Application Algorithm

The weights to be used for the application algorithm are obtained from the training algorithm. Then the activations are set for the input vectors. The net input is calculated and applying the activations, the output is calculated. This output is broadcasted to all other units. The process is repeated until the convergence of the net is obtained. The application algorithm of a discrete Hopfield net is given as follows:

Step 1: Initialize weights to store pattern (use Hebb rule from Section 6.2.1).
While activations of the net are not converged perform Steps 2 to 8.

Step2: For each input vector x, repeat Steps 3 to 7.

Step 3: Set initial activations of the net equal to the external input vector x, $y_i = x_i (i = 1, \ldots n)$

Step 4: Perform Steps 5 to 7 for each unit y_i

Step 5: Compute the net input.

$$y_{-in\,i} = x_i + \sum_j y \cdot w_{ji}$$

Step 6: Determine activation (output signal)

$$y_i = \begin{cases} 1, & \text{if } y_{-ini} > \theta_i; \\ y_i, & \text{if } y_{-in} = \theta_i; \\ 0, & \text{if } y_{-ini} < \theta_i; \end{cases}$$

Step 7: Broadcast the value of y_i to all other units.

Step 8: Test for convergence.

The value of threshold θ_i is usually taken to be zero. The order of update of the unit is random but each unit must be updated at the same average rate.

Storage Capacity

In the Hopfield net the number of binary patterns that can be stored and recalled in a net with reasonable accuracy is given by,

P=0.15n where n is the number of neurons in the net.

If bipolar patterns are used, then P = $\frac{n}{2 \log_2 n}$

6. BAM Architecture

The bi-directional associative memory (BAM) consists of two layers of processing elements that are fully interconnected between the layers. It is an hetero-associative, that is, it accepts an input vector on one set of neurons and produces a related, but different, output vector on another set. It has the characteristic of a content addressable memory rather than needing the address of a piece of information you can ask for the location of the information that you want. Applications for content-addressable memories include paging systems (i.e. get me the physical address of the virtual address. The virtual address is the content and it is what we know. The physical address is where the virtual address resides.).

Since a BAM is bi-directional, it can take input in either layer, however this implementation currently only supports loading inputs into the input layer. Although a BAM can be binary or bipolar, this version also only has the weight formula for bipolar and the transfer function (activation function) for bipolar (with bias is fixed at 0). The BAM is capable of generalization, producing correct outputs despite corrupted inputs. Also adaptive versions can abstract, extracting the ideal from a set of noisy examples.

As in other neural networks, in the BAM architecture there are weights associated with the connections between processing elements. The general network architecture of BAM is shown in Fig. 2. In this we have considered as n units on the X-layer and m units on the Y layer. For convenience, we shall define the X vector, $x \in R^n$ as input vector and the Y-vector, $y \in R^m$ as its output vector. **All connections between units are bi-directional**, with weights at each end. Information passes back and forth from one layer to the other, through these connections. In this network, the weights can be determined in advance, if all of the training vectors, can be identified.

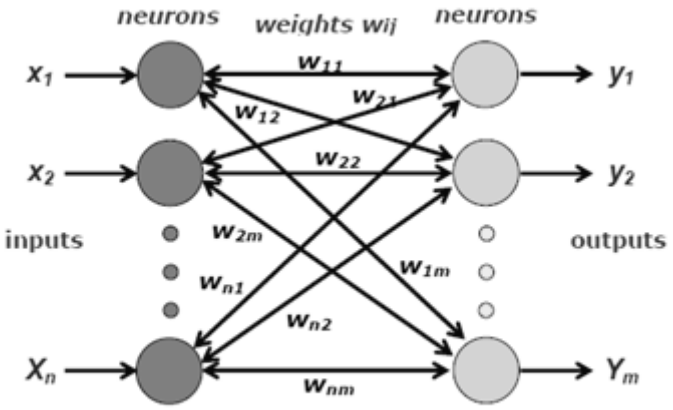


Fig. Bidirectional Associative Memory model

Fig.2. The BAM here has n units in X layer, and m units on the Y layer.

1. *Discrete BAM*

The binary and bipolar forms of BAM are closely related. In each case the weights are found from the sum of the Hebb outer product of the bipolar form of the training vector pairs. The activation function used is a step activation function with the non-zero threshold. But generally we know that the bipolar vectors improve the performance of the net.

The weight matrix to store the set of input and target vectors s(p):t(p), p = 1,...P, where

$$s(p) = (s_1(p),...s_i(p),...s_n(p))$$

and can be determined with the help of Hebb rule.

For binary input vectors, the weight matrix can be determined by the formula.

$$w_{ij} = \sum_p (2S_i(p)-1)(2t_j(p)-1)$$

For bipolar input vectors, the weight matrix is given by,

$$w_{ij} = \sum_{p} s_i(p) t_j(p)$$

Activation Function

The activation function for discrete BAM depends on whether binary or bipolar vectors are used. The activation function is an appropriate step function. For binary input vectors, the activation function for the Y-layer is

$$y_j = \begin{cases} 1, & \text{if } y_{-inj} > 0 \\ y_j, & \text{if } y_{-inj} = 0 \\ 0, & \text{if } y_{-inj} < 0 \end{cases}$$

and the activation function for the X-layer is

$$x_i = \begin{cases} 1, & \text{if } x_{-ini} > 0 \\ x_i, & \text{if } x_{-ini} = 0 \\ 0, & \text{if } x_{-ini} < 0 \end{cases}$$

For bipolar vector, the activation function for Y-layer is

$$y_j = \begin{cases} 1, & \text{if } y_{-inj} > \vartheta_j \\ y_j, & \text{if } y_{-inj} = \vartheta_j \\ 0, & \text{if } y_{-inj} < \vartheta_j \end{cases}$$

and the activation function for the X-layer is

$$x_i = \begin{cases} 1, & \text{if } x_{-ini} > \vartheta_i \\ x_i, & \text{if } x_{-ini} = \vartheta_i \\ 0, & \text{if } x_{-ini} < \vartheta_i \end{cases}$$

2. *Continuous BAM*

The continuous BAM was introduced by Kosko, 1988. A continuous bidirectional associative memory has the capability to transfer the input smoothly and continuously into the respective output in the range between [0, 1]. The continuous BAM uses logistic sigmoid function as the activation function for all units.

For binary input vectors (S(P), t(P)), p = 1, 2, ..., P, the weights are determined by the formula.

$$w_{ij} = \sum_{P} (2 s_i(p) - 1)(2 t_j(p) - 1)$$

Training of BAM

The training of BAM network is the process of storing information. It is one step process for given set information. Given L vector pairs that constitute the set of examples that would like to store, we can construct the weight matrix as:

$$W = Y_1 X_1^T + Y_2 X_2^T + \ldots + Y_L X_L^T$$

We can make the BAM into an auto associative memory by constructing the weight matrix as square and symmetric. That is,

$$W = X_1 X_1^T + X_2 X_2^T + \ldots + X_L X_L^T$$

Recall Algorithm of BAM

Once weight matrix has been constructed, the BAM can be used to recall information, when presented with some key information. If the desired information is only partially known in advance or is noisy, the BAM may be able to complete the information.

To recall information using the BAM, we perform the following steps:

1. Apply an initial vector pair, (X_0, Y_0) to the processing elements of the BAM.
2. Propagate the information from X layer to the Y layer, and update the values on the Y-layer units.
3. Propagate the updated Y information back to the X layer and update the units there.
4. Repeat steps 2 and 3 until there is no further change in the units on each layer.

This algorithm is what gives the BAM its bidirectional nature. The terms inputs and output refer to different quantities, depending on the current direction of the propagation. For example, in going from Y to X, the 'Y' vector is considered as the input to the network, and the X vector is the output. The opposite is true when propagating from X to Y. If all goes well, the final, stable state will recall one of the exemplars used to construct the weight matrix.

If we try to put too much information in a given BAM, a phenomenon known as cross talk occurs between exemplar patterns. Cross talk occurs when exemplar patterns are too close to each other. The interaction between these patterns can result in the creation of spurious stable states. In that wise, the BAM could stabilize on meaningless vectors. The spurious stable states corresponds to minima that appear between the minima that correspond to the exemplars.

Example: For the following given pairs vectors construct weight matrix for appropriate structure of BAM.

$X_1 = (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^T$, $Y_1 = (1, -1, -1, -1, -1, 1)^T$

$X_2 = (1, 1, 1 -1, -1, -1, 1, 1, -1, -1)^T$, $Y_2 = (1, 1, 1, 1, -1, -1)^T$

$$W = Y_1 X_1^T + Y_2 X_2^T$$

$$= \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 \end{bmatrix}$$

$$= \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{matrix} \\ \begin{bmatrix} 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 \end{bmatrix} \end{array} + \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 \end{bmatrix} \end{array}$$

$$W = \begin{bmatrix} 2 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & -2 & 0 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ -2 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 2 & 0 \\ 0 & -2 & -2 & 2 & 0 & 2 & 0 & -2 & 0 & 2 \end{bmatrix}$$ X to Y and $$W^T = \begin{bmatrix} 2 & 0 & 0 & 0 & -2 & 0 \\ 0 & 2 & 2 & 2 & 0 & -2 \\ 0 & 2 & 2 & 2 & 0 & -2 \\ 0 & -2 & -2 & -2 & 0 & 2 \\ -2 & 0 & 0 & 0 & 2 & 0 \\ 0 & -2 & -2 & -2 & 0 & 2 \\ 2 & 0 & 0 & 0 & -2 & 0 \\ 0 & 2 & 2 & 2 & 0 & -2 \\ -2 & 0 & 0 & 0 & 2 & 0 \\ 0 & -2 & -2 & -2 & 0 & 2 \end{bmatrix}$$ Y to X

For our first trial, we choose an X vector with a Hamming distance of 1 from X_1;

$X_0 = (-1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t$

This situation could represent noise on the input vector. The starting Y_0 vector is one of the training vector Y_2; $Y_0 = (1, 1, 1, -1, -1)^T$. (Note that in a realistic problem you may not have prior knowledge of the output vector. Use a random bipolar vector if necessary).

We will propagate first from X to Y

$$Net^y = WX = \begin{bmatrix} 2 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & -2 & 0 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ -2 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 2 & 0 \\ 0 & -2 & -2 & 2 & 0 & 2 & 0 & -2 & 0 & 2 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

$\therefore$ $Net^y = (4, -12, -12, -12, -4, 12)^t$

The new Y vector $Y_{new} = (1\ -1\ -1\ -1\ -1\ 1)^t$ and which is also one of the training vector .

Propagating back to X layer:

$$Net^x = W^T Y = \begin{bmatrix} 2 & 0 & 0 & 0 & -2 & 0 \\ 0 & 2 & 2 & 2 & 0 & -2 \\ 0 & 2 & 2 & 2 & 0 & -2 \\ 0 & -2 & -2 & -2 & 0 & 2 \\ -2 & 0 & 0 & 0 & 2 & 0 \\ 0 & -2 & -2 & -2 & 0 & 2 \\ 2 & 0 & 0 & 0 & -2 & 0 \\ 0 & 2 & 2 & 2 & 0 & -2 \\ -2 & 0 & 0 & 0 & 2 & 0 \\ 0 & -2 & -2 & -2 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = [4\ -8\ -8\ \ 8\ -4\ \ 8\ \ 4\ -8\ -4\ \ 8]^T$$

$X_{new} = [1\ -1\ -1\ 1\ -1\ 1\ 1\ -1\ -1\ 1]^t$

Further passes result in no change, so we are finished. The BAM successfully recalled the first training set.

BAM Energy Function

In supervised learning (multiplayer perceptron), during the training process the weights form a dynamical system. That is, the weights change as a function of time, and those changes can be represented as a set of coupled differential equations.

For BAM, a slightly different situation occurs. The weights are calculated on advance, and are not part of a dynamical system. On the other hand, an unknown pattern presented to the BAM may require several passes before the network stabilizes on a final result. In this situation, the X and Y vectors change as a function of time, and they form a dynamical system.

In the theory of dynamical systems, a theorem can be proved concerning the existence of stable states that uses the concept of a function called a Lyapunov function or Energy function. If a bounded function of the state variables of a dynamical system can be found, such that all state changes results in a decrease in the value of the function, then the system has a stable solution. This function is called a *Lyapunov* function or energy function. In the case of the BAM, such a function exists we shall call it the BAM energy function. It has the form

$$E(x, y) = -Y^T WX$$

or in terms of components

$$E = -\sum_{i=1}^{M}\sum_{j=1}^{N} y_i w_{ij} x_j$$

Memory (storage) capacity of the BAM

The memory (storage) capacity of the BAM is min (n, m) where,

n is the number of units in x-layers

m is the number of units in y-layers.

Back Propagation algorithm

(or Generalized Delta Rule)

A. Derivation:

Let us say,

x is a neuron in input set, j is a neuron in hidden layer, k is a neuron in output layer

$\vec{x}_j$ = input vector for unit j (x_i = i^{th} input to the j^{th} unit)

$\vec{w}_j$ = input weight vector for unit j (w_{ji} = weight from input x_i to hidden unit j)

$\vec{w}_k$ = hidden weight vector for unit k (w_{kj} = weight from hidden unit j to output k)

$z_j = \vec{w}_j.\vec{x}_j$, the weighted sum of inputs for hidden unit j

$z_k = \vec{w}_k.\vec{Y}_j$, the weighted sum of inputs for output unit k

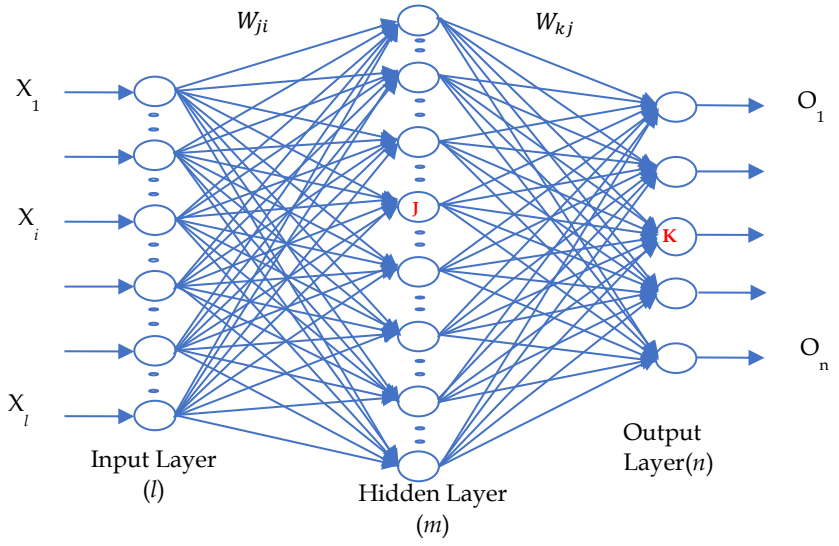
Y_j = output of unit j ($Y_j = \sigma(z_j)$) and Y_k = output of unit k ($Y_k = \sigma(z_k)$) where $\sigma(z_k)$ is sigmoid activation function defined as $\sigma(z) = \frac{1}{1+e^{-z}}$

t_k = target for unit k

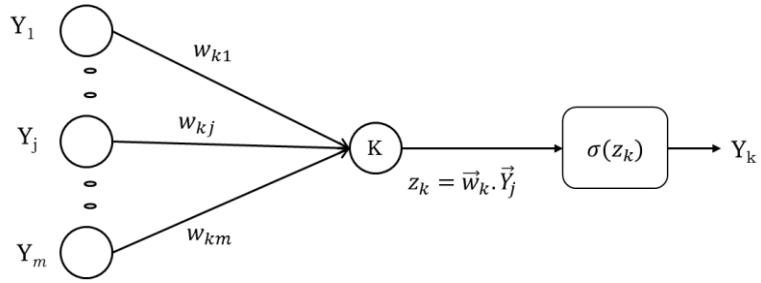
Downstream(j) = set of units whose immediate inputs include the output of j

Outputs = set of output units in the final layer

Since we update after each training example, we can simplify the notation somewhat by imagining that the training set consists of exactly one example and so the error can simply be denoted by E.



Case 1: error derivative for output unit k



We want to calculate $\frac{\partial E}{\partial w_{kj}}$ for each input weight w_{kj} for each output unit k. Note first that since z_k is a function of w_{kj} regardless of where in the network unit k is located,

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_{kj}} \quad = \frac{\partial E}{\partial z_k} Y_j$$

Furthermore, $\frac{\partial E}{\partial z_k}$ is the same regardless of which input weight of unit k we are trying to update. So we denote this quantity by δ_k.

Consider the case when $k \in Outputs$. We know

$$\boldsymbol{E = \frac{1}{2} \sum_{k=1}^{\text{outputs}} (t_k - \sigma(z_k))^2}$$

Since the outputs of all units $m \neq k$ are independent of w_{kj}, we can drop the summation and consider just the contribution to E by k.

$$\delta_k = \frac{\partial E}{\partial z_k} \quad = \frac{\partial}{\partial z_k} \frac{1}{2} (t_k - Y_k)^2$$

$$= -(t_k - Y_k) \frac{\partial Y_k}{\partial z_k}$$

$$= -(t_k - Y_k) \frac{\partial}{\partial z_k} \sigma(z_k)$$

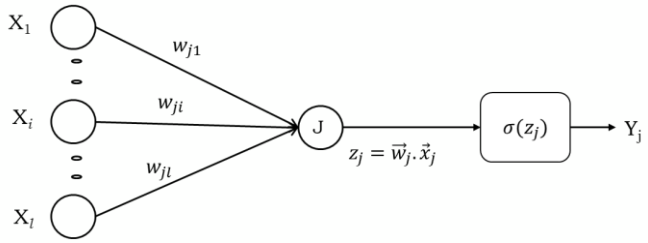
$$= -(t_k - Y_k)(1 - \sigma(z_k))\sigma(z_k)$$

$$\boldsymbol{\delta_k = -(t_k - Y_k)(1 - Y_k)Y_k}$$

Thus

$$\boldsymbol{\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}} = \eta \delta_k Y_j}$$

Case 2: error derivative for hidden layer unit *j*



Now consider the case when j is a hidden unit. Like before, we make the following two important observations.

1. For each unit k downstream from j, z_k is a function of Y_j and hence function of z_j
2. The contribution to error by all units $l \neq j$ in the same layer as j is independent of w_{ji}

We want to calculate $\frac{\partial E}{\partial w_{ji}}$ for each input weight w_{ji} for each hidden unit j. Note that w_{ji} influences just z_j which influences Y_j which influences $z_k \forall k \in Downstream(j)$ each of which influence E. So, we cannot ignore the summation, and can write

$$\frac{\partial E}{\partial w_{ji}} = \sum_{k \in Downstream(j)} \frac{\partial E}{\partial z_k} \cdot \frac{\partial z_k}{\partial Y_j} \cdot \frac{\partial Y_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ji}}$$

$$= \sum_{k \in Downstream(j)} \frac{\partial E}{\partial z_k} \cdot \frac{\partial z_k}{\partial Y_j} \cdot \frac{\partial Y_j}{\partial z_j} \cdot x_i$$

Again, note that all the terms except x_i in the above product are the same regardless of which input weight of unit j we are trying to update. Like before, we denote this common quantity by δ_j. Also note that $\frac{\partial E}{\partial z_k} = \delta_k$,

$\frac{\partial z_k}{\partial Y_j} = w_{kj}$ and $\frac{\partial Y_j}{\partial z_j} = Y_j(1 - Y_j)$. Substituting,

$$\delta_j = \sum_{k \in Downstream(j)} \frac{\partial E}{\partial z_k} \cdot \frac{\partial z_k}{\partial Y_j} \cdot \frac{\partial Y_j}{\partial z_j}$$

$$= \sum_{k \in Downstream(j)} \delta_k w_{kj} Y_j(1 - Y_j)$$

$$\boldsymbol{\delta_j = Y_j(1 - Y_j) \sum_{k \in Downstream(j)} \delta_k w_{kj}}$$

Thus

$$\boldsymbol{\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = \eta \delta_j X_i}$$

B. Formal statement of the algorithm: (Summary)

- Each training example is of the form $\langle \vec{x}, \vec{t} \rangle$ where $\vec{x}$ is the input vector and $\vec{t}$ is the target vector. η is the learning rate($0 < \eta < 1$). l, m and n are the number of input, hidden and output nodes respectively. Input from unit i to unit j is denoted x_i and its weight is denoted by ω_{ji}.
- Create a feed-forward network with ***l*** inputs, ***m*** hidden units, and ***n*** output units.
- Initialize all the weights to small random values (e.g., between -0.05 and 0.05)
- Until termination condition is met, for each training example$\langle \vec{x}, \vec{t} \rangle$, **train the network** as below.
- Input the instance $\vec{x}$ and compute the hidden layer output $Y_j = \sigma\left(\sum_{i=1}^{l} w_{ji} x_i + \theta_j\right)$
- Compute the output layer output $Y_k = \sigma\left(\sum_{j=1}^{m} w_{kj} Y_j + \theta_k\right)$
- For each output unit k, calculate $\delta_k = Y_k(1 - Y_k)(t_k - Y_k)$ and $\Delta w_{kj} = \eta \delta_k Y_j$
- For each Hidden unit j, calculate $\delta_j = Y_j(1 - Y_j) \sum_{k \in Downstream(j)} w_{kj}\, \delta_k$ and $\Delta w_{ji} = \eta \delta_j x_i$
- If no momentum term is not considered, update each network weight ω_{ji} as follows:

$$w_{ji}(t+1) \leftarrow w_{ji}(t) + \Delta w_{ji}(t)$$

- If momentum term ($0.4 < \alpha < 0.9$) is considered, then update each network weight ω_{ji} as follows:

$$w_{ji}(t+1) \leftarrow w_{ji}(t) + \Delta w_{ji}(t) + \alpha\, \Delta w_{ji}(t-1)$$

  Where t is the epoch number

C. Momentum term 'α'

The weight changes can be given some "momentum" by introducing an extra term into the weight adaptation equation that will produce a large change in the weight if the changes are currently large and will decrease as the changes become less. This means that the network is less likely to get stuck in local minima early on, since the momentum term will push the changes over the local increases in the energy function, following the overall downward trend. Momentum is of great assistance in speeding up convergence along shallow gradients, allowing the path the network takes towards the solution to pick up speed in the downhill direction. The energy landscape may consist of long gradually sloping ravines, which finish at minima. Convergence along these ravines is slow, since the

direction that has to be followed has only a slight gradient, and usually the algorithm oscillates across the ravine valley as it meander towards a solution.

This momentum term can be written as (e.g, for hidden layer weights) follows:

$$w_{ji}(\mathrm{t}+1) \leftarrow w_{ji}(\mathrm{t}) + \Delta w_{ji}(t) + \alpha\ \Delta w_{ji}(t-1)$$

where α is the momentum factor $0<\alpha<1$. (generally considered values are $0.4 < \alpha < 0.9$)

D. Learning difficulties and improvements

Occasionally the network settles into a stable solution that does not provide the correct output. In these cases, the energy function is in a local minimum. This means that in every direction in which the network could move in the energy landscape, the energy is higher than at the current position. It may be that there is only a slight "lip" to cross before reaching an actual deeper minimum, but the network has no way of knowing this, since learning is accomplished by following the energy function down in the steepest direction, until it reaches the bottom of well, at which point there is no direction to move in order to reduce the energy. There are alternative approaches to minimizing these occurrences.

E. Effect of learning rate 'η'

Learning rate coefficient determines the size of the weight adjustments made at each iteration and hence influences the rate of convergence. Poor choice of the learning rate coefficient can result in a failure in convergence. We Should keep coefficient constant through all the iterations for best results. If the learning rate coefficient is too large, the search path will oscillate and converges more Slowly than a direct descent as shown in Fig. 3.15(a). If the coefficient is too small, the descent will progress in small steps significantly increasing the time to converge

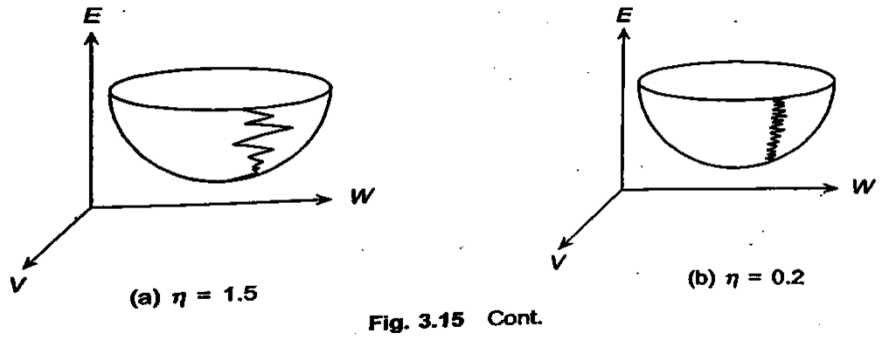


Fig. 3.15 Cont.

F. Merits and Demerits of Back Propagation Network

Merits

1. The mathematical formula present here, can be applied to any network and does not require any special mention of the features of the function to be learnt.
2. The computing time is reduced if the weights chosen are small at the beginning.
3. The batch update of weights exist, which provides a smoothing effect on the weight correction terms.

Demerits

1. The number of learning steps may be high, and also the learning phase has intensive calculations.
2. The selection of the number of hidden nodes in the network is a problem. If the number of hidden neurons is small, then the function to be learnt may not be possibly represented, as the capacity of network is small. If the number of hidden neurons is increased, the number of independent variables of the error function also increases and the computing time also increases rapidly.
3. For complex problems it may require days or weeks to train the network or it may not train at all. Long training time results in non-optimum step size.
4. The network may get trapped in a local minima even though there is a much deeper minimum nearby.
5. The training may sometimes cause temporal instability to the system.

G. Local Minima & Global Minima

Suppose we start with a weight set for the network corresponding to a point P. If we perform gradient descent, the minimum we encounter is the one at M_l not that at Mg. M_l is called local minima and corresponds to a partial solution for network in response to the training data. M_g is the global minima and unless measures are taken to escape from M_l, M_g will never be reached.