Artificial Neural Networks

The term "Artificial neural network" refers to a biologically inspired sub-field of artificial intelligence modelled after the brain.

An Artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain.

Similar to a human brain has neurons interconnected to each other, artificial neural networks also have neurons that are linked to each other in various layers of the networks. These neurons are known as nodes.

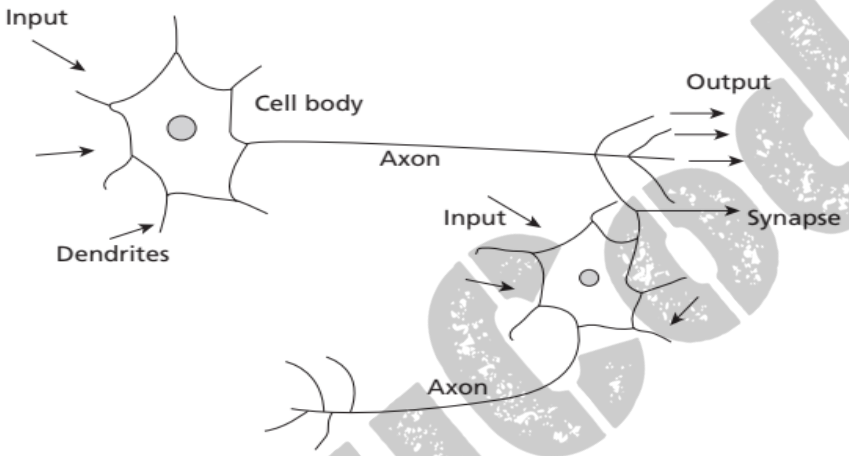
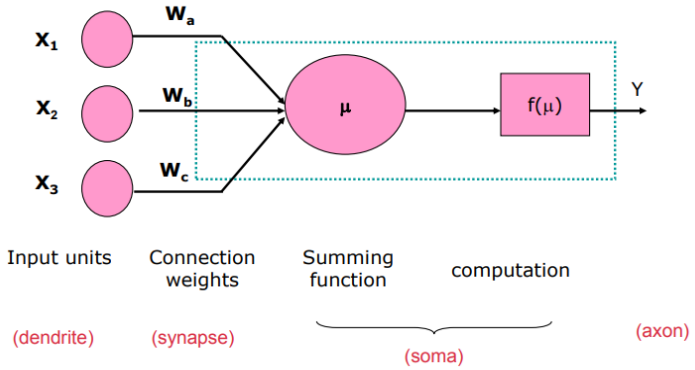


Figure 10.1: A Biological Neuron

The biological neuron consists of main **four** parts:

- **dendrites**: nerve fibres carrying electrical signals to the cell .
- **cell body:** computes a non-linear function of its inputs
- **axon:** single long fiber that carries the electrical signal from the cell body to other neurons
- **synapse:** the point of contact between the axon of one cell and the dendrite of another, regulating a chemical connection whose strength affects the input to the cell.

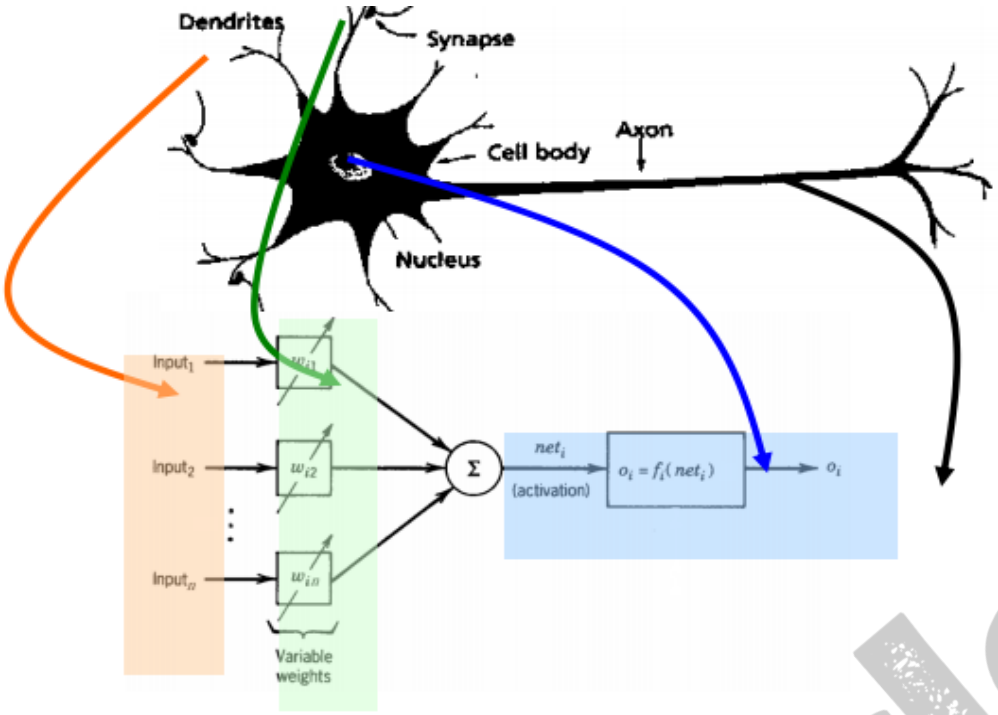


Dendrites are tree like networks made of nerve fiber connected to the cell body.

An **Axon** is a single, long connection extending from the cell body and carrying signals from the neuron. The end of axon splits into fine strands. It is found that each strand terminated into small bulb like organs called as synapse. It is through synapse that the neuron introduces its signals to other nearby neurons. The receiving ends of these synapses on the nearby neurons can be found both on the dendrites and on the cell body. There are approximately 104 synapses per neuron in the human body. Electric impulse is passed between synapse and dendrites. It is a chemical process which results in increase/decrease in the electric potential inside the body of the receiving cell. If the electric potential reaches a thresh hold value, receiving cell fires & pulse / action potential of fixed strength and duration is send through the axon to synaptic junction of the cell. After that, cell has to wait for a period called refractory period.

Difference between biological and Artificial Neuron

Characteristics	Artificial Neural Network	Biological(Real) Neural Network
Speed	Faster in processing information. Response time is in nanoseconds.	Slower in processing information. The response time is in milliseconds.
Processing	Serial processing.	Massively parallel processing.
Size & Complexity	Less size & complexity. It does not perform complex pattern recognition tasks.	Highly complex and dense network of interconnected neurons containing neurons of the order of 1011 with 1015 of interconnections.<strong
Storage	Information storage is replaceable means new data can be added by deleting an old one.	Highly complex and dense network of interconnected neurons containing neurons of the order of 1011 with 1015 of interconnections.
Fault tolerance	Fault intolerant. Information once corrupted cannot be retrieved in case of failure of the system.	Information storage is adaptable means new information is added by adjusting the interconnection strengths without destroying old information
Control Mechanism	There is a control unit for controlling computing activities	No specific control mechanism external to the computing task.



ARTIFICIAL NEURONS:

Artificial neurons are like biological neurons that are linked to each other in various layers of the networks. These neurons are known as nodes.

A node or a neuron can receive one or more input information and process it. artificial neurons are connected by connection links to another neuron. Each connection link is associated with a synaptic weight. The structure of a single neuron is shown below:

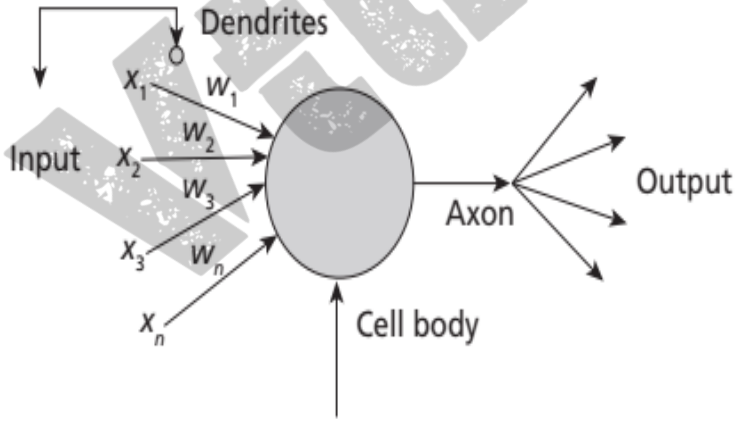


Figure 10.2: An Artificial Neuron

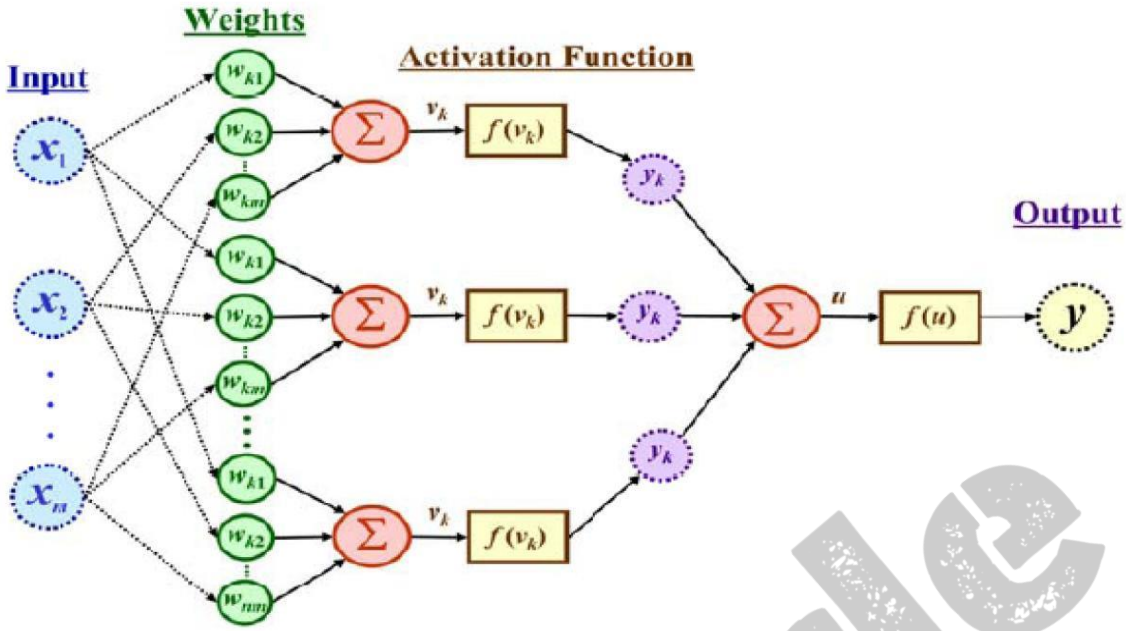


Fig: McCulloch-Pitts Neuron Mathematical model.

Simple Model of an ANN

The first mathematical model of a biological neuron was designed by McCulloch-Pitts in 1943.

It includes 2 steps:

1. It receives weighted inputs from other neurons.
2. It operates with a threshold function or activation function.

Basically, a neuron takes an input signal (dendrite), processes it like the CPU (soma), passes the output through a cable like structure to other connected neurons (axon to synapse to other neuron's dendrite).

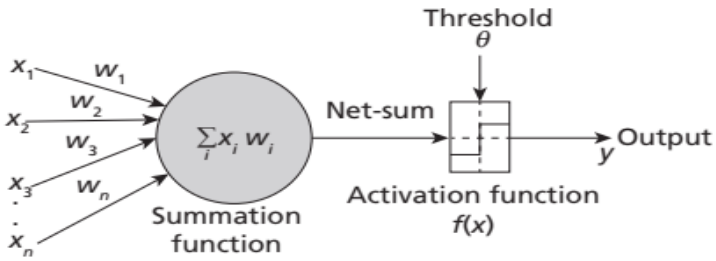
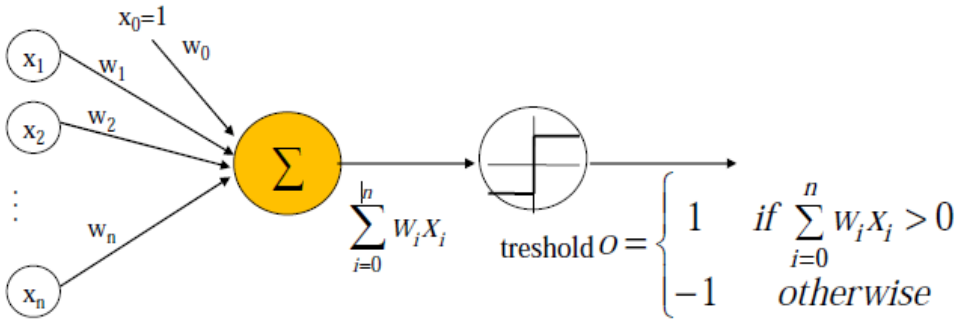


Figure 10.3: McCulloch & Pitts Neuron Mathematical Model

OR

$$W_0 + W_1X_1 + \ldots + W_nX_n = \sum_{i=0}^{n} W_iX_i = \vec{W} \cdot \vec{X}$$



Working:

The received input are computed as a weighted sum which is given to the activation function and if the sum exceeds the threshold value the neuron gets fired.The neuron is the basic processing unit that receives a set of inputs x1,x2,x3,….xn and their associated weights w1,w2,w3,….wn. The summation function computes the weighted sum of the inputs received by the neuron.

Sum=$\sum x_i w_i$

Activation functions:

- To make work more efficient and for exact output, some force or activation is given. Like that, activation function is applied over the net input to calculate the output of an ANN. Information processing of processing element has two major parts: input and output. An integration function (f) is associated with input of processing element.

- Several **activation functions** are there.

1. **Identity function or Linear Function:** It is a linear function which is defined as $f(x) = x \; for \; all \; x$

The output is same as the input ie the weighted sum. The function is useful when we do not apply any threshold. The output value ranged between –∞ and +∞

2. **Binary step function**: This function can be defined as

$f(x) = \{ 1 \quad if \; x \geq \theta$

$0 \quad if \; x < \theta$

Where, θ represents threshhold value. It is used in single layer nets to convert the net input to an output that is binary (0 or 1).

3. **Bipolar step function:** This function can be defined as

$f(x) = \{ 1 \quad if\ x \geq \theta$

$-1\ if\ x < \theta$

Where, θ represents threshold value. It is used in single layer nets to convert the net input to an output that is bipolar (+1 or -1).

4. **Sigmoid function:** It is used in Back propagation nets.

Two types:

a) **Binary sigmoid function**: It is also termed as logistic sigmoid function or unipolar sigmoid function. It is defined as

$$f(x) = \frac{1}{1 + e^{-\lambda x}}$$

where, λ represents steepness parameter. The range of sigmoid function is 0 to 1

b) Bipolar sigmoid function: This function is defined as

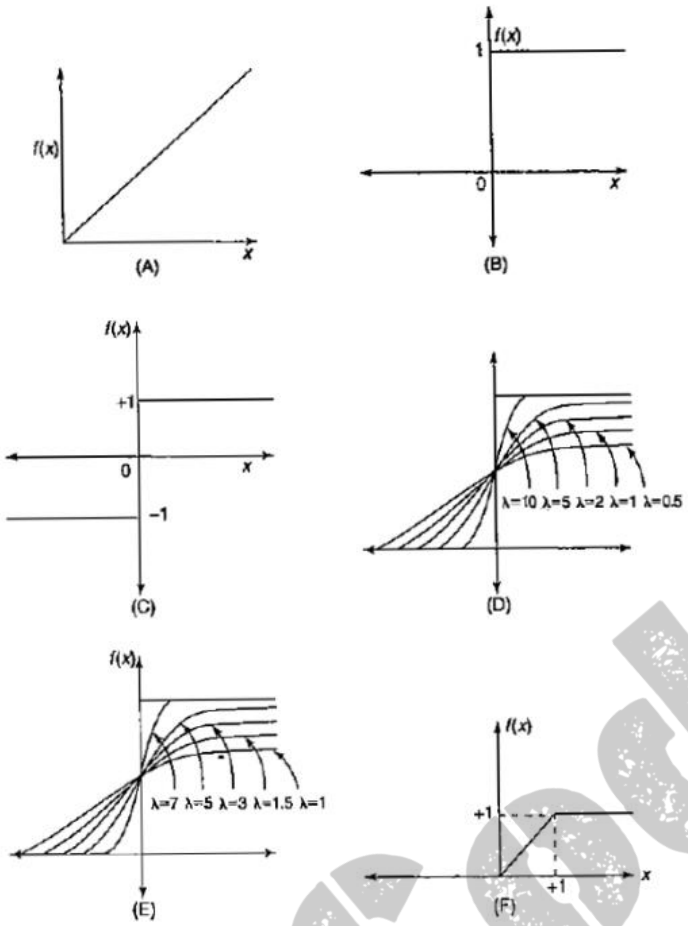
$$f(x) = \frac{2}{1 + e^{-\lambda x}} - 1 = \frac{1 - e^{-\lambda x}}{1 + e^{-\lambda x}}$$

Where λ represents steepness parameter and the sigmoid range is between -1 and +1.

5. **Ramp function**: The ramp function is defined as:

$$f(x) = \begin{cases} 1\ if x > 1 \\ x\ if\ 0 \leq x \leq 1 \\ 0\ if\ x < 0 \end{cases}$$

It is a linear function whose upper and lower limits are fixed.



Depiction of activation functions: (A) identity function; (B) binary step function; (C) bipolar step function; (D) binary sigmoidal function; (E) bipolar sigmoidal function; (F) ramp function.

6. **Tanh-Hyperbolic tangent function :** Tanh function is very similar to the sigmoid/logistic activation function, and even has the same S-shape with the difference in output range of -1 to 1. In Tanh, the larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

$$f(x) = \frac{\left(e^{x} - e^{-x}\right)}{\left(e^{x} + e^{-x}\right)}$$

7. ReLU Function

ReLU stands for **Rectified Linear Unit.**

Although it gives an impression of a linear function, ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient.

The main catch here is that the ReLU function does not activate all the neurons at the same time.

The neurons will only be deactivated if the output of the linear transformation is less than 0

8. **Softmax function:** Softmax is an activation function that scales numbers/logits into probabilities. The output of a Softmax is a vector (say v) with probabilities of each possible outcome. The probabilities in vector v sums to one for all possible outcomes or classes.

$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$$

where,

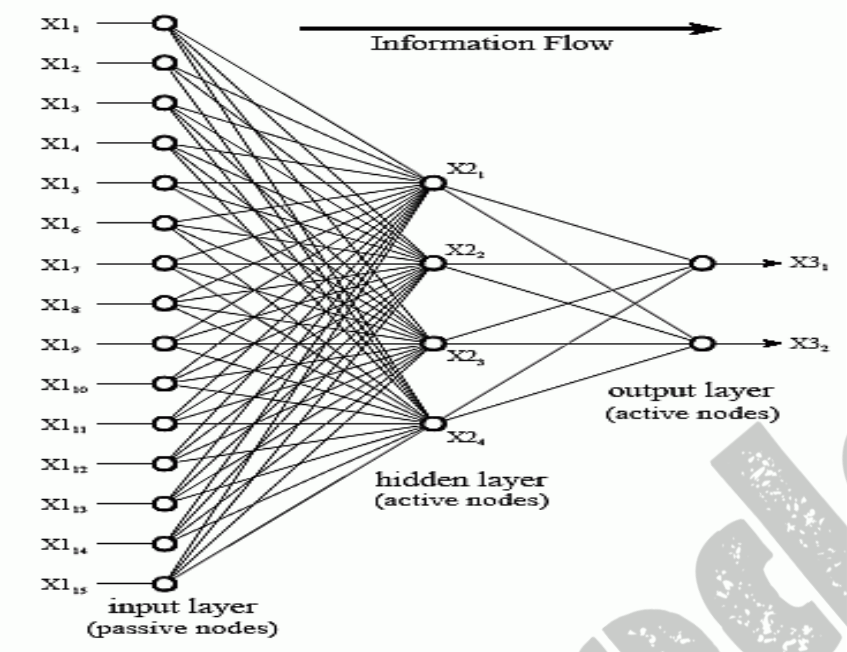
y	is an input vector to a softmax function, S. It consist of n elements for n classes (possible outcomes)
y_i	the i-th element of the input vector. It can take any value between -inf and +inf
$exp(y_i)$	standard exponential function applied on y_i. The result is a small value (close to 0 but never 0) if $y_i < 0$ and a large value if y_i is large. eg • $\exp(55) = 7.69e{+}23$ (A very large value) • $\exp(-55) = 1.30e$-24 (A very small value close to 0) **Note**: $\exp(*)$ is just $\mathbf{e}^*$ where $\mathbf{e} = 2.718$, the Euler's number.
$\sum_{j=1}^{n} \exp(y_j)$	A normalization term. It ensures that the values of output vector $S(y)_i$ sums to 1 for i-th class and each of them and each of them is in the range 0 and 1 which makes up a valid probability distribution.
n	Number of classes (possible outcomes)

Artificial Neural Network Structure

- Artificial Neural Networks Computational models inspired by the human brain: – Massively parallel, distributed system, made up of simple processing units (neurons) – Synaptic connection strengths among neurons are used to store the acquired knowledge.
  - Knowledge is acquired by the network from its environment through a learning process.

- **The Neural Network is constructed from 3 type of layers:**
- Input layer — initial data for the neural network.

- Hidden layers — intermediate layer between input and output layer and place where all the computation is done.
- Output layer — produce the result for given inputs.



PERCEPTRON AND LEARNING THEORY

- The perceptron is also a simplified model of a biological neuron.
- The perceptron is an algorithm for supervised learning of binary classifiers. It is a type of linear classifier, i.e. a classification algorithm that makes all of its predictions based on a linear predictor function combining a set of weights with the feature vector.
- One type of ANN system is based on a unit called a perceptron.

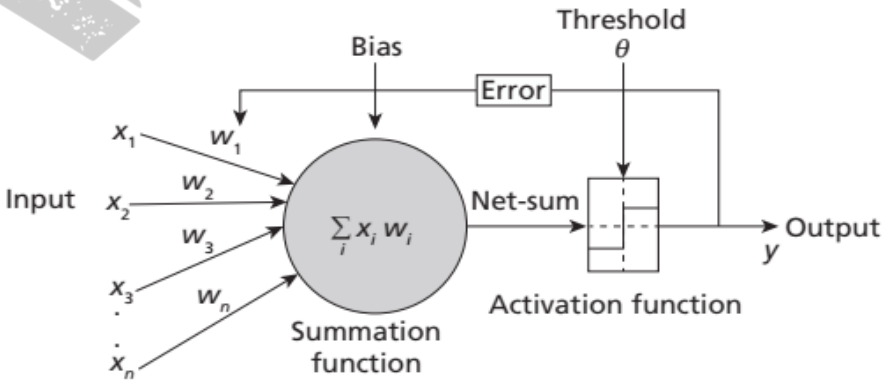
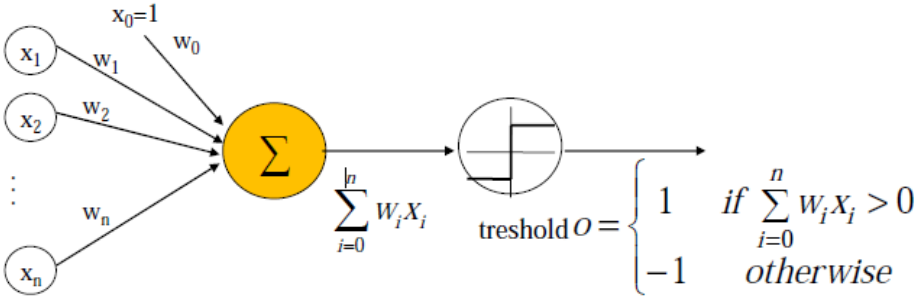


Figure 10.5: Perceptron Model

OR

$$W_0 + W_1X_1 + \ldots + W_nX_n = \sum_{i=0}^{n} W_iX_i = \vec{W} \cdot \vec{X}$$



- The perceptron can represent all boolean primitive functions AND, OR, NAND , NOR.
- Some boolean functions can not be represented .
  - E.g. the XOR function.

Major components of a perceptron

- Input
- Weight
- Bias
- Weighted summation
- Step/activation function
- output

WORKING:

- Feed the features of the model that is required to be trained as input in the first layer. All weights and inputs will be multiplied – the multiplied result of each weight and input will be added up.The Bias value will be added to shift the output function .This value will be presented to the activation function (the type of activation function will depend on the need)
  The value received after the last step is the output value.
  The activation function is a binary step function which outputs a value 1, if f(x) is above the threshold value Θ and a 0 if f(x) is below the threshold value Θ. Then the output of a neuron is:

$$f(x) = \begin{cases} 1 & if\ f(x) \geq \theta \\ 0 & if\ f(x) < \theta \end{cases}$$

Algorithm 10.1: Perceptron Algorithm

Set initial weights $w_1, w_2, \ldots, w_n$ and bias θ to a random value in the range [–0.5, 0.5].

For each Epoch,

1. Compute the weighted sum by multiplying the inputs with the weights and add the products.
2. Apply the activation function on the weighted sum:
$$Y = \text{Step}\ ((x_1 w_1 + x_2 w_2) - \theta)$$
3. If the sum is above the threshold value, output the value as positive else output the value as negative.
4. Calculate the error by subtracting the estimated output $Y_{estimated}$ from the desired output $Y_{desired}$:
$$\text{error } e(t) = Y_{desired} - Y_{estimated}$$
[If error $e(t)$ is positive, increase the perceptron output Y and if it is negative, decrease the perceptron output Y.]
5. Update the weights if there is an error:
$$\Delta w_i = \propto \times e(t) \times x_i$$
$$w_i = w_i + \Delta w_i$$
where, x_i is the input value, $e(t)$ is the error at step t, $\propto$ is the learning rate and Δw_i is the difference in weight that has to be added to w_i.

PROBLEM:

Design a 2 layer network of perceptron to implement NAND gate. Assume your own weights and biases in the range of [-0.5 0.5]. Use learning rate as 0.4.

Solution:

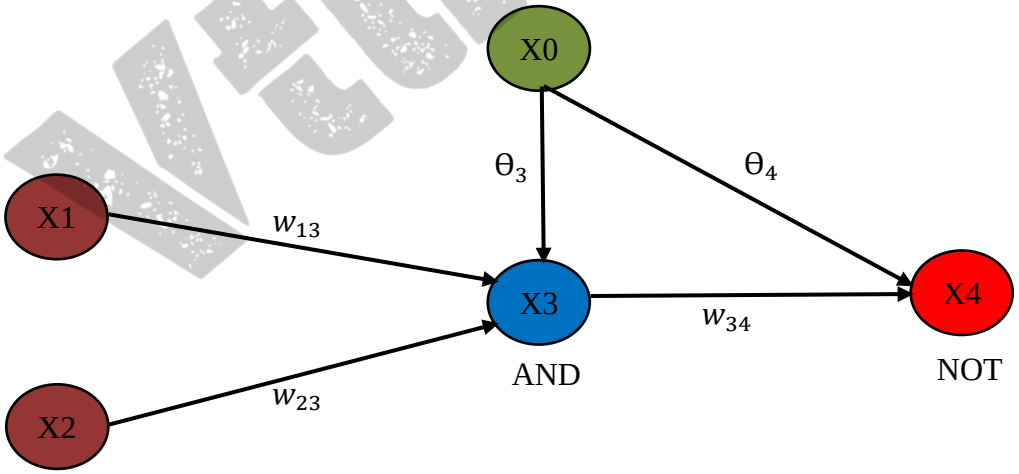


Figure 1 Two Layer Network for NAND gate

Table 1: Weights and Biases

X_1	X_2	$O_{desired}$	w_{13}	w_{23}	w_{34}	θ_3	θ_4	X_0
0	1	1	0.1	-0.4	0.3	0.2	-0.3	1

Table 2: Truth Table of NAND Gate

X_1	X_2	$X_1\ AND\ X_2$	$NAND = NOT(X_1\ AND\ X_2)$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

ITERATION 1:

Step 1: FORWARD PROPAGATION

1. **Calculate net inputs and outputs in input layer as shown in Table 3.**

Table 3: Net Input and Output Calculation

Input Layer	I_j	O_j
X_1	0	0
X_2	1	1

2. **Calculate net inputs and outputs in hidden and output layer as shown in Table 4.**

Table 4: Net Input and Output Calculation in Hidden and Output layer

$Unit_j$	$Net\ Input\ I_j$	$Net\ output\ O_j$
X_3	$I_3 = X_1W_{13} + X_2W_{23} + X_0\Theta_3$ $= 0(0.1) + 1(-0.4) + 1(0.2)$ $= -0.2$	$O_3 = \frac{1}{1+e^{-I_3}}$ $= \frac{1}{1+e^{-(-0.2)}}$ $= 0.450$
$Unit_k$	$Net\ Input\ I_k$	Net output O_k
X_4	$I_4 = O_3W_{34} + X_0\Theta_4$ $= (0.450 * 0.3) + 1(-0.3)$ $= -0.165$	$O_4 = \frac{1}{1+e^{-I_4}}$ $= \frac{1}{1+e^{-(-0.165)}}$ $= 0.458$

3. **Calculate Error**

$$Error = O_{desired} - O_{estimated}$$
$$= 1 - 0.458$$

$$Error = 0.542$$

Step 2: BACKWARD PROPAGATION

1. **For each $unit_k$ in the output layer**

$$Error_k = O_k * (1 - O_k) * (O_{desired} - O_k)$$

For each $unit_j$ in the hidden layer

$$Error_j = O_j * (1 - O_j) * (\sum Error_k * W_{jk})$$

Table 5: Error Calculation

For each output layer $unit_k$	*$Error_k$*
X_4	$Error_k = O_k * (1 - O_k) * (O_{desired} - O_k)$ $= 0.458(1 - 0.458)(1 - 0.458)$ $= 0.134$
For each hidden layer $unit_j$	*$Error_j$*
X_3	$Error_j = O_j * (1 - O_j) * (\sum Error_k * W_{jk})$ $= 0.450 * (1 - 0.450) * 0.134 * 0.3$ $= 0.0099$

2. **Update Weights and biases**

Table 6: Weight and Bias Calculation

w_{ij}	*$w_{ij} = w_{ij} + (\alpha * Error_j * O_i)$*	*Net Weight*
w_{13}	$w_{13} = w_{13} + (0.4 * Error_3 * O_1)$ $= 0.1 * (0.4 * 0.0099 * 0)$	0.1
w_{23}	$w_{23} = w_{23} + (0.4 * Error_3 * O_2)$ $= -0.4 * (0.4 * 0.0099 * 1)$	-0.396
w_{24}	$w_{24} = w_{24} + (0.4 * Error_4 * O_2)$ $= 0.3 * (0.4 * 0.134 * 0.450)$	0.324

Θ_j	$\Theta_j = \Theta_j + (\alpha * Error_j)$	*Net Bias*
Θ_3	$\Theta_3 = \Theta_3 + (0.4 * Error_3)$ $= 0.2 + (0.4 * 0.0099)$	0.203
Θ_4	$\Theta_4 = \Theta_4 + (0.4 * Error_4)$ $= -0.3 + (0.4 * 0.134$	-0.246

ITERATION 2:

Step 1: FORWARD PROPAGATION

1. **Calculate net inputs and outputs in hidden and output layer**

Table 7: Inputs and Outputs in Hidden and Output layer

$Unit_j$	*Net Input* I_j	*Net output* O_j
X_3	$I_3 = X_1W_{13} + X_2W_{23} + X_0\Theta_3$ $= 0(0.1) + 1(-0.396) + 1(0.203)$ $= -0.193$	$O_3 = \frac{1}{1+e^{-I_3}}$ $= \frac{1}{1+e^{-(-0.193)}}$ $= 0.451$
$Unit_k$	*Net Input* I_k	*Net output* O_k
X_4	$I_4 = O_3W_{34} + X_0\Theta_4$ $= (0.451 * 0.324) + 1(-0.246)$ $= -0.099$	$O_4 = \frac{1}{1+e^{-I_4}}$ $= \frac{1}{1+e^{-(-0.099)}}$ $= 0.475$

2. **Calculate Error**

$$Error = O_{desired} - O_{estimated}$$
$$= 1 - 0.475$$
$$Error = 0.525$$

ITERATION	*ERROR*
1	0.542
2	0.525

=0.542-0.525
=0.017

In iteration 2 the error gets reduced to 0.525. This process will continue until desired output is achieved.

How a Multi-Layer Perceptron does solves the XOR problem. Design an MLP with back propagation to implement the XOR Boolean function.

Solution:

X_1	X_2	Y
0	0	1
0	1	0
1	0	0
1	1	1

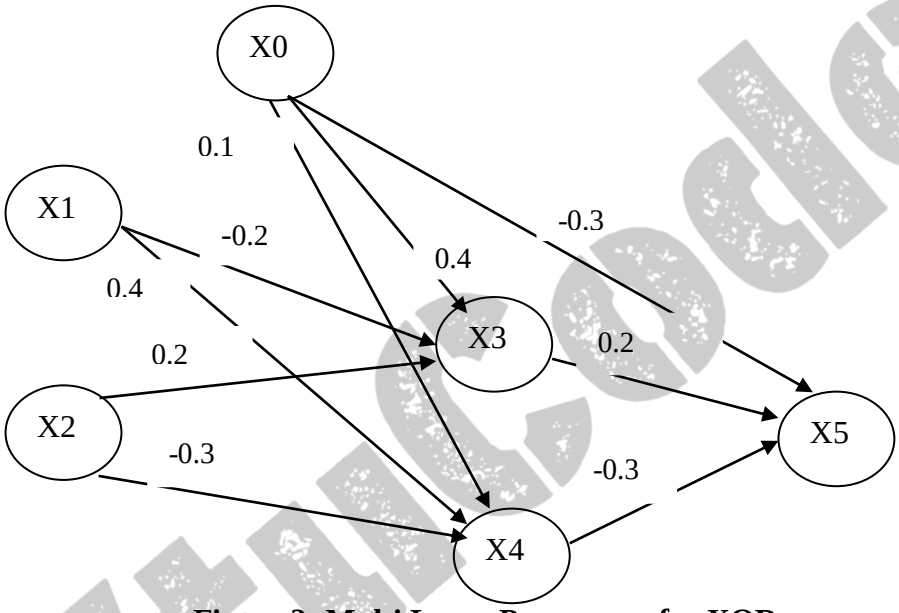


Figure 2: Multi Layer Perceptron for XOR

Learning rate: =0.8

Table 8: Weights and Biases

X_1	X_2	W_{13}	W_{14}	W_{23}	W_{24}	W_{35}	W_{45}	$\theta 3$	$\theta 4$	$\theta 5$
1	0	-0.2	0.4	0.2	-0.3	0.2	-0.3	0.4	0.1	-0.3

Step 1: Forward Propagation

1. Calculate Input and Output in the Input Layer shown in Table 9.

Table 9: Net Input and Output Calculation

Input Layer	I_j	O_j
X_1	1	1
X_2	0	0

2. Calculate Net Input and Output in the Hidden Layer and Output Layer shown in Table 10.

Table 10: Unit j at Hidden Layer and Output Layer – Net Input and Output Calculation

Unit j	**Net Input I_j**	**Output O_j**
X_3	$I_3 = X_1*W_{13} + X_2*W_{23} + X_0*\theta_3$ $I_3 = 1*-0.2 + 0*0.2 + 1*0.4 = 0.2$	$O_3 = \frac{1}{1+e^{-I3}} = \frac{1}{1+e^{-0.2}} = 0.549$
X_4	$I_4 = X_1*W_{14} + X_2*W_{24} + X_0*\theta_4$ $I_4 = 1*0.4 + 0*-0.3 + 1*0.1 = 0.5$	$O_4 = \frac{1}{1+e^{-I4}} = \frac{1}{1+e^{-0.5}} = 0.622$
X_5	$I_5 = O_3 * W_{35} + O_4*W_{45} + X_0*\theta_5$ $I_5 = 0.549 * 0.2 + 0.622 * -0.3 + 1*-0.3 = -0.376$	$O_5 = \frac{1}{1+e^{-I5}} = \frac{1}{1+e^{0.376}} = 0.407$

3. Calculate Error = $O_{desired} - O_{Estimated}$

So error for this network is,

Error = $O_{desired} - O_7 = 1 - 0.407 = 0.593$

Step 2: Backward Propagation

1. Calculate Error at each node as shown in Table 11.

For each unit k in the output layer, calculate

Error $_k = O_k (1-O_k) (Y_N - O_k)$

For each unit j in the hidden layer, calculate

Error $_j = O_j (1-O_j) \sum_k Error_k W_{jk}$

Table 11: Error Calculation for each unit in the Output layer and Hidden layer

For Output Layer Unit $_k$	**Error$_k$**
X_5	Error $_5 = O_5 (1-O_5) (1 - O_5)$ $= 0.407 * (1-0.407) * (1- 0.407)$ $= 0.143$
For Hidden layer Unit $_j$	**Error$_j$**
X_4	Error $_4 = O_4 (1-O_4) \sum_k Error_k W_{jk} = O_4 (1-O_4) Error_5 W_{45}$ $= 0.622 (1-0.622) *- 0.3 *0.143$ $= -0.010$
X_3	Error $_3 = O_3 (1-O_3) \sum_k Error_k W_{jk} = O_3 (1-O_3) Error_5 W_{35}$ $= 0.549 (1- 0.549) * 0.143 * 0.2$

	= -0.007

2. Update weight using the below formula,

Learning rate $\alpha = 0.8$

$\Delta W_{ij} = \propto * \text{Error}_j * O_i$

$W_{ij} = W_{ij} + \Delta W_{ij}$

The updated weight and bias is shown in Table 12 and Table 13.

Table 12: Weight Updation

W_{ij}	**$W_{ij} = W_{ij} + \propto * \text{Error}_j * O_i$**	**New Weight**
W_{13}	$W_{13} = W_{13} + 0.8 * \text{Error}_3 * O_1$ $= -0.2 + 0.8 * 0.007 * 1$	-0.194
W_{14}	$W_{14} = W_{14} + 0.8 * \text{Error}_4 * O_1$ $= 0.4 + 0.8 * -0.01 * 1$	0.392
W_{23}	$W_{23} = W_{23} + 0.8 * \text{Error}_3 * O_2$ $= 0.2 + 0.8 * 0.007 * 0$	0.2
W_{24}	$W_{24} = W_{24} + 0.8 * \text{Error}_4 * O_2$ $= -0.3 + 0.8 * -0.001 * 0$	-0.3
W_{35}	$W_{35} = W_{35} + 0.8 * \text{Error}_5 * O_3$ $= 0.2 + 0.8 * 0.143 * 0.4$	0.154
W_{45}	$W_{45} = W_{45} + 0.8 * \text{Error}_5 * O_4$ $= 0.3 + 0.8 * 0.143 * 0.1$	-0.288

Update bias using the below formula,

$\Delta\theta_j = = \propto * \text{Error}_j$

$\theta_j = \theta_j + \Delta\theta_j$

Table 13: Bias Updation

θ_j	**$\theta_j = \theta_j + \propto * \text{Error}_j$**	**New Bias**
$\theta 3$	$\Theta_3 = \theta_3 + \propto * \text{Error}_3$ $= 0.4 + 0.8 * 0.007$	0.405
$\theta 4$	$\theta_4 = \theta_4 + \propto * \text{Error}_4$ $= 0.1 + 0.8 * - 0.01$	0.092
$\theta 5$	$\theta_5 = \theta_5 + \propto * \text{Error}_5$ $= -0.3 + 0.8 * 0.143$	-0.185

Iteration 2

Now with the updated weights and biases,

1. Calculate Input and Output in the Input Layer shown in Table 14.

Table 14: Net Input and Output Calculation

Input Layer	Ij	Oj
X_1	1	1
X_2	0	0

2. Calculate Net Input and Output in the Hidden Layer and Output Layer shown in Table 15.

Table 15: Net Input and Output Calculation in the Hidden Layer and Output Layer

Unit j	Net Input Ij	Output Oj
X_3	$I_3 = X_1*W_{13} + X_2*W_{23} + X_0*\theta_3$ $I_3 = 1*-0.194 + 0*0.2 + 1*0.405 = 0.211$	$O_3 = \frac{1}{1+e^{-I3}} = \frac{1}{1+e^{-0.211}} =$ 0.552
X_4	$I_4 = X_1*W_{14} + X_2*W_{24} + X_0*\theta_4$ $I_4 = 1*0.392 + 0*-0.3 + 1*0.092 = 0.484$	$O_4 = \frac{1}{1+e^{-I4}} = \frac{1}{1+e^{-0.484}} =$ 0.618
X_5	$I_5 = O_3 * W_{35} + O_4*W_{45} + X_0*\theta_5$ $I_5 = 0.552* 0.154 + 0.618* -0.288 + 1*-0.185 =$ -0.282	$O_5 = \frac{1}{1+e^{-I5}} = \frac{1}{1+e^{0.282}} =$0.429

The output we receive in the network at node 5 is 0.407.

Error = 1 - 0.429= 0.571

Now when we compare the error, we get in the previous iteration and in the current iteration, the network has learnt which reduces the error by 0.022.

Error is reduced by 0.055: 0.593 – 0.571.

Consider the Network architecture with 4 input units and 2 output units. Consider four training samples each vector of length 4.

Training samples

i1: (1, 1, 1, 0)

i2: (0, 0, 1, 1)

i3: (1, 0, 0, 1)

i4: (0, 0, 1, 0)

Output Units: Unit 1, Unit 2

Learning rate $\eta(t) = 0.6$

Initial Weight matrix

$$\begin{bmatrix}\text{Unit 1}\\ \text{Unit 2}\end{bmatrix}:\begin{bmatrix}0.2 & 0.8 & 0.5 & 0.1\\ 0.3 & 0.5 & 0.4 & 0.6\end{bmatrix}$$

Identify an algorithm to learn without supervision? How do you cluster them as we expected?

Solution:

Use Self Organizing Feature Map (SOFM)

Iteration 1:

Training Sample X_1: (1, 1, 1, 0)

Weight matrix

$$\begin{bmatrix}\text{Unit 1}\\ \text{Unit 2}\end{bmatrix}:\begin{bmatrix}0.2 & 0.8 & 0.5 & 0.1\\ 0.3 & 0.5 & 0.4 & 0.6\end{bmatrix}$$

Compute Euclidean distance between X_1: (1, 1, 1, 0) and Unit 1 weights.

$d^2 = (0.2 - 1)^2 + (0.8 - 1)^2 + (0.5 - 1)^2 + (0.1 - 0)^2$

$= 0.94$

Compute Euclidean distance between X_1: (1, 1, 1, 0) and Unit 2 weights.

$d^2 = (0.3 - 1)^2 + (0.5 - 1)^2 + (0.4 - 1)^2 + (0.6 - 0)^2$

$= 1.46$

Unit 1 wins

Update the weights of the winning unit

New Unit 1 weights = [0.2 0.8 0.5 0.2] + 0.6 ([1 1 1 0] - [0.2 0.8 0.5 0.2])

= [0.2 0.8 0.5 0.2] + 0.6 [0.8 0.2 0.5 -0.2]

= [0.2 0.8 0.5 0.2] + [0.48 0.12 0.30 -0.12]

= [0.68 0.92 0.80 0.08]

$$\begin{bmatrix}\text{Unit 1}\\ \text{Unit 2}\end{bmatrix}:\begin{bmatrix}0.68 & 0.92 & 0.80 & 0.08\\ 0.3 & 0.5 & 0.4 & 0.6\end{bmatrix}$$

Iteration 2:

Training Sample X_2: (0, 0, 1, 1)

Weight matrix

$$\begin{bmatrix}\text{Unit 1}\\ \text{Unit 2}\end{bmatrix}:\begin{bmatrix}0.68 & 0.92 & 0.80 & 0.08\\ 0.3 & 0.5 & 0.4 & 0.6\end{bmatrix}$$

Compute Euclidean distance between X_2: (0, 0, 1, 1) and Unit 1 weights.

$d^2 = (0.68 -0)^2 + (0.92 – 0)^2 + (0.80 -1)^2 + (0.08 – 1)^2$

$= 2.1952$

Compute Euclidean distance between X_2: (0, 0, 1, 1) and Unit 2 weights.

$d^2 = (0.3 -0)^2 + (0.5 – 0)^2 + (0.4 -1)^2 + (0.6– 1)^2$

$= 0.86$

Unit 2 wins

Update the weights of the winning unit

New Unit 2 weights = [0.3 0.5 0.4 0.6] + 0.6 ([0 0 1 1] - [0.3 0.5 0.4 0.6])

= [0.3 0.5 0.4 0.6] + 0.6 [-0.3 -0.5 0.6 0.4]

= [0.3 0.5 0.4 0.6] + [-0.18 -0.30 0.36 0.24]

= [0.12 0.2 0.76 0.84]

$$\begin{bmatrix}\text{Unit 1}\\ \text{Unit 2}\end{bmatrix}:\begin{bmatrix}0.68 & 0.92 & 0.80 & 0.08\\ 0.12 & 0.2 & 0.76 & 0.84\end{bmatrix}$$

Iteration 3:

Training Sample X_3: (1, 0, 0, 1)

Weight matrix

$$\begin{bmatrix}\text{Unit 1}\\ \text{Unit 2}\end{bmatrix}:\begin{bmatrix}0.68 & 0.92 & 0.80 & 0.08\\ 0.12 & 0.2 & 0.76 & 0.84\end{bmatrix}$$

Compute Euclidean distance between X_3: (1, 0, 0, 1) and Unit 1 weights.

$d^2 = (0.68 -1)^2 + (0.92 – 0)^2 + (0.80 -0)^2 + (0.08 – 1)^2$

$= 2.44$

Compute Euclidean distance between X_3: (1, 0, 0, 1) and Unit 2 weights.

$d^2 = (0.12 -1)^2 + (0.2 – 0)^2 + (0.76 -0)^2 + (0.84– 1)^2$

$= 1.42$

Unit 2 wins

Update the weights of the winning unit

New Unit 2 weights = [0.12 0.2 0.76 0.84] + 0.6 ([1 0 0 1] - [0.12 0.2 0.76 0.84])

= [0.12 0.2 0.76 0.84] + 0.6 [0.88 -0.2 -0.76 0.16]

= [0.12 0.2 0.76 0.84] + [0.53 -0.12 -0.46 0.096]

= [0.65 0.08 0.3 0.94]

$$\begin{bmatrix}\text{Unit 1}\\ \text{Unit 2}\end{bmatrix}:\begin{bmatrix}0.68 & 0.92 & 0.80 & 0.08\\ 0.65 & 0.08 & 0.3 & 0.94\end{bmatrix}$$

Iteration 4:

Training Sample X_4: (0, 0, 1, 0)

Weight matrix

$$\begin{bmatrix}\text{Unit 1}\\ \text{Unit 2}\end{bmatrix}:\begin{bmatrix}0.68 & 0.92 & 0.80 & 0.08\\ 0.65 & 0.08 & 0.3 & 0.94\end{bmatrix}$$

Compute Euclidean distance between X_4: (0, 0, 1, 0) and Unit 1 weights.

$d^2 = (0.68 -0)^2 + (0.92 –0)^2 + (0.80 -1)^2 + (0.08 – 0)^2$

= 1.36

Compute Euclidean distance between X_1: (0, 0, 1, 0) and Unit 2 weights.

$d^2 = (0.65- 0)^2 + (0.08 – 0)^2 + (0.3 -1)^2 + (0.94– 0)^2$

= 1.8025

Unit 1 wins

Update the weights of the winning unit

New Unit 1 weights = [0.68 0.92 0.80 0.08] + 0.6 ([0 0 1 0] - [0.68 0.92 0.80 0.08])

= [0.68 0.92 0.80 0.08] + 0.6 [-0.68 -0.92 0.2 -0.08]

= [0.68 0.92 0.80 0.08] + [-0.408 -0.552 0.12 -0.258]

= [0.27 0.37 0.92 -0.178]

$$\begin{bmatrix}\text{Unit 1}\\ \text{Unit 2}\end{bmatrix}:\begin{bmatrix}0.27 & 0.37 & 0.92 & -0.178\\ 0.65 & 0.08 & 0.3 & 0.94\end{bmatrix}$$

Best mapping unit for each of the sample taken are,

X_1: (1, 1, 1, 0) → Unit 1

X_2: (0, 0, 1, 1) → Unit 2

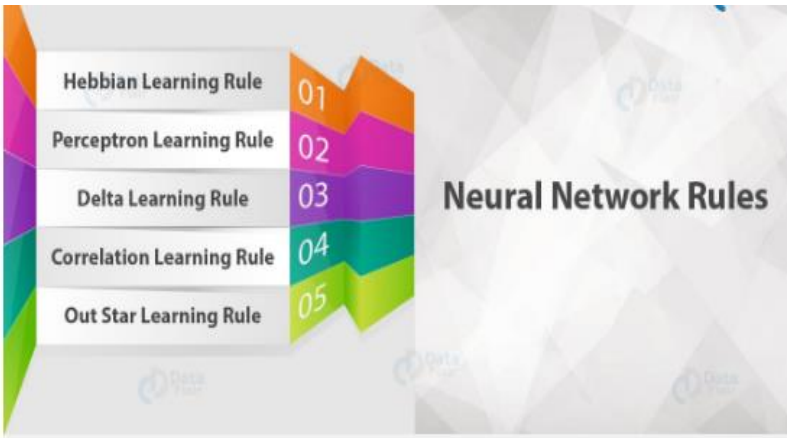
X_3: (1, 0, 0, 1) → Unit 2

X_4: (0, 0, 1, 0) → Unit 1

This process is continued for many epochs until the feature map doesn't change.

Learning Rules

Learning in NN is performed by adjusting the network weights in order to minimize the difference between the desired and estimated output.



Introduction to Learning Rules in Neural Network

Delta Learning Rule and Gradient Descent

- Developed by ***Widrow*** **and** ***Hoff,*** the delta rule, is one of the most common learning rules.
- It is **supervised** learning.
- Delta rule is derived from gradient descent method(Back-propogation).
- It is **Non-linearly separable**. Also called as continuous perceptron Learning rule.
- It updates the connection weights with the difference between the target and the output value. It is the least mean square learning algorithm.
- The Delta difference is measured as an **error function** or also called as **cost** function.

$$\text{Training Error} = \frac{1}{2}\sum_{d \in T}(O_{Desired} - O_{Estimated})^2$$

where, T is the training dataset, $O_{Desired}$ and $O_{Estimated}$ are the desired target output and estimated actual output, respectively, for a training instance d.

The principle of gradient descent is an optimization approach which is used to minimize the cost function by converging to a local minimal point moving in the negative direction of the gradient and each step size during movement is determined by the learning rate and the slope of the gradient.

TYPES OF ANN

1. Feed Forward Neural Network
2. Fully connected Neural Network
3. Multilayer Perceptron
4. Feedback Neural Network

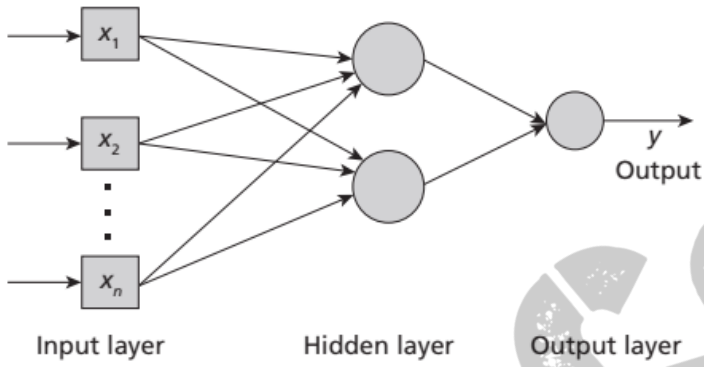
Feed Forward Neural Network:

Feed-Forward Neural Network is a single layer perceptron. A sequence of inputs enters the layer and are multiplied by the weights in this model. The weighted input values are then summed together to form a total. If the sum of the values is more than a predetermined threshold, which is normally set at zero, the output value is usually 1, and if the sum is less than the threshold, the output value is usually -1.

The single-layer perceptron is a popular feed-forward neural network model that is frequently used for classification.

The model may or may not contain hidden layer and there is no backpropagation.

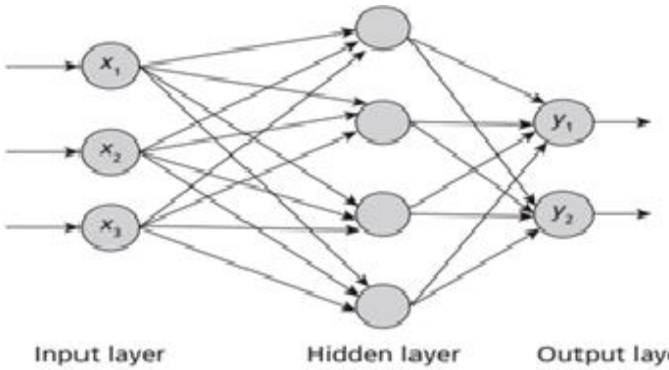
Based on the number of hidden layers they are further classified into single-layered and multilayered feed forward network.



Model of a Feed Forward Neural Network

Fully connected Neural Network:

- A fully connected neural network consists of a series of fully connected layers that connect every neuron in one layer to every neuron in the other layer.
- The major advantage of fully connected networks is that they are "structure agnostic" i.e. there are no special assumptions needed to be made about the input.



Model of a Fully Connected Neural Network

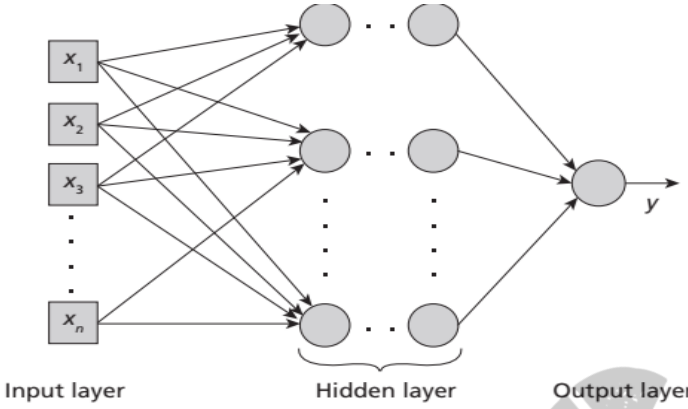
Multilayer Perceptron:

A multi-layer perceptron has one input layer and for each input, there is one neuron (or node), it has one output layer with a single node for each output and it can have any number of hidden layers and each hidden layer can have any number of nodes.

The information flows in both directions.

The weight adjustment training is done via **backpropagation**.

Every node in the multi-layer perception uses a sigmoid activation function. The sigmoid activation function takes real values as input and converts them to numbers between 0 and 1 using the sigmoid formula.



Model of a Multi-Layer Perceptron

Feedback Neural Network:

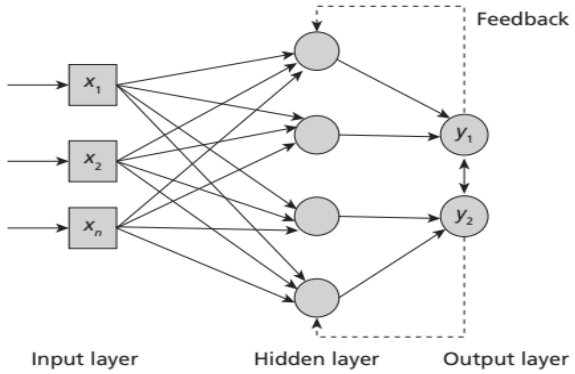
Feedback networks also known as **recurrent neural network** or interactive neural network are the deep learning models in which information flows in **backward** direction.

It allows feedback loops in the network. Feedback networks are dynamic in nature, powerful and can get much complicated at some stage of execution

Neuronal connections can be made in any way.

RNNs may process input sequences of different lengths by using their internal state, which can represent a form of memory.

They can therefore be used for applications like speech recognition or handwriting recognition.



Model of a Feedback Neural Network

LEARNING OF MULTI LAYER PERCEPTRON

WHY MULTI LAYER PERCEPTRON?

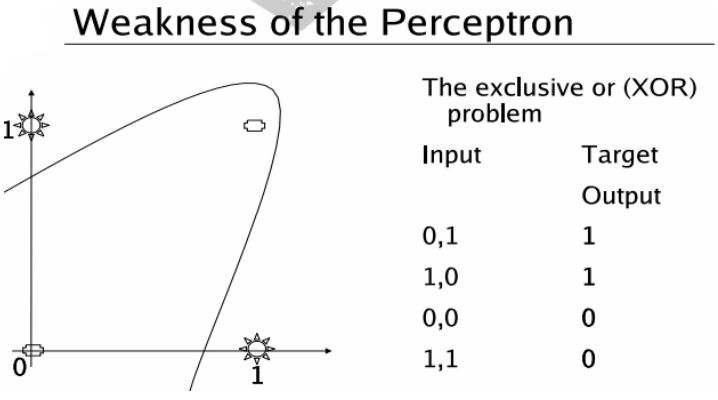
Imagine a group of 7-year-old students who are working on a math problem, imagine that each of them can only do arithmetic with two numbers. But you are giving them an equation like this 5 x 3 + 2 x 4 + 8 x 2, how can they solve it?

To **solve** this problem, we can break it down into smaller parts and give them to each of the students. One student can solve the first part of the equation "5 x 3 = 15" and another student can solve the second part of the equation "2 x 4 = 8". The third student can solve the third part "8 x 2 = 16".
Finally, we can simplify it to 15 + 8 + 16. Same way, one of the students in the group can solve "15 + 8 = 23" and another one can solve "23 + 16 = 39", and that's the answer

So here we are breaking down the large math problem into different sections and giving them to each of the students who are just doing really simple calculations, but as a result of the teamwork, they can solve the problem efficiently.

This is exactly the idea of how a **multi-layer perceptron (MLP) works.** Each neuron in the MLP is like a student in the group, and each neuron is only able to perform simple arithmetic operations. However, when these neurons are connected and work together, they can solve complex problems.

The principle **weakness of the perceptron** was that it could only solve problems that were **linearly separable.**



A multilayer perceptron (MLP) is a **fully connected feed-forward** artificial neural network with at least three layers input, output, and at least one hidden layer.

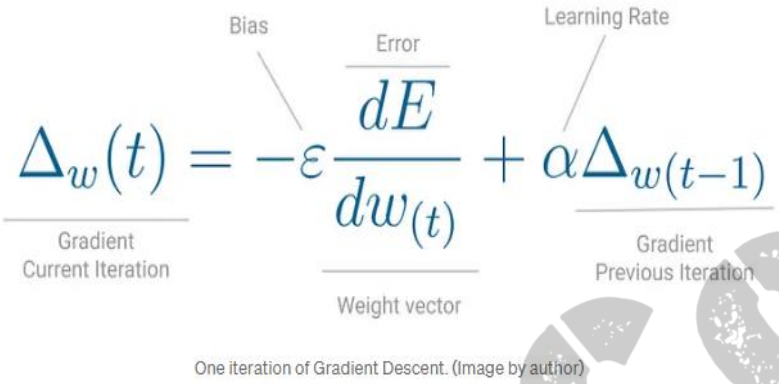
The mapping between inputs and output is **non-linear. (Ex: XOR gate)**

In Perceptron the neuron must have an activation function that imposes a threshold, like ReLU or sigmoid, neurons in a Multilayer Perceptron can use any arbitrary activation function.

MLP networks are uses back propagation for supervised learning network.

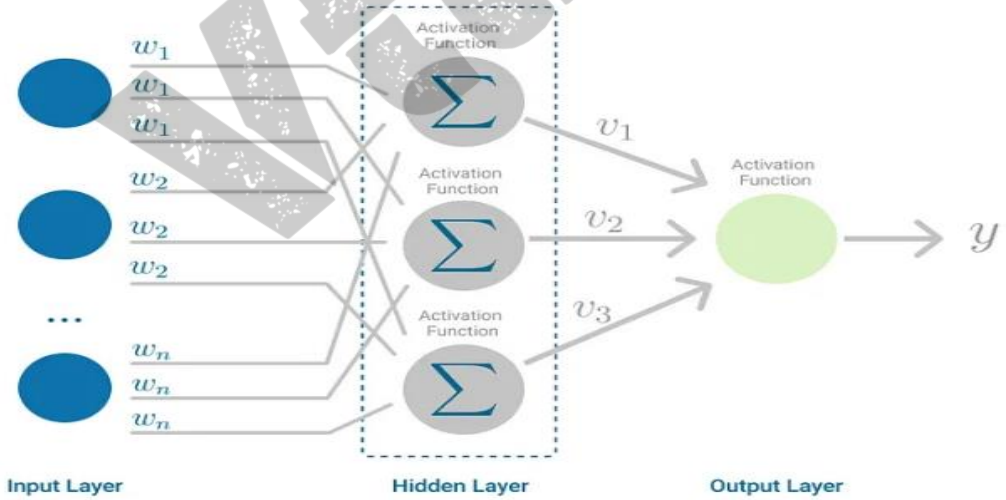
The **activation functions** used in the layers can be linear or Non-linear depending on the type of a problem.

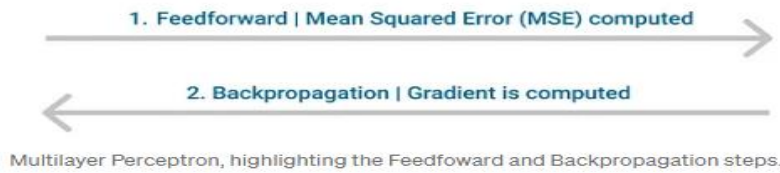
NOTE : In each iteration, after the weighted sums are forwarded through all layers, the gradient of the **Mean Squared Error** is computed across all input and output pairs. Then, to propagate it back, the weights of the first hidden layer are updated with the value of the gradient. That's how the weights are propagated back to the starting point of the neural network.



One iteration of Gradient Descent. (Image by author)

This process keeps going until gradient for each input-output pair has converged, meaning the newly computed gradient hasn't changed more than a specified *convergence threshold*, compared to the previous iteration.





Multilayer Perceptron, highlighting the Feedfoward and Backpropagation steps.

Works in 2 stages.

1. Forward phase
2. Backward phase

Starting from initial random weights, multi-layer perceptron (MLP) minimizes the loss function by repeatedly updating these weights. After computing the loss, a backward pass propagates it from the output layer to the previous layers, providing each weight parameter with an update value meant to decrease the loss.

ALGORITHM

Algorithm 10.2: Learning in an MLP

Input: Input vector ($x_1, x_2, ..., x_n$)
Output: Y_n
Learning rate: α

Assign random weights and biases for every connection in the network in the range [−0.5, +0.5].

Step 1: Forward Propagation

1. **Calculate Input and Output in the *Input Layer*:**

   (Input layer is a direct transfer function, where the output of the node equals the input).

   Input at Node j 'I$_j$' in the Input Layer is

$$I_j = x_j$$

where,

x_j is the input received at Node j

Output at Node j 'O$_j$' in the Input Layer is

$$O_j = I_j$$

2. **Calculate Net Input and Output in the *Hidden Layer* and *Output Layer*:**

*Net Input at Node j in the **Hidden Layer** is*

$$I_i = \sum_{i=1}^{n} x_i w_{ij} + x_0 \times \theta_j$$

where,

x_i is the input from Node i

w_{ij} is the weight in the link from Node i to Node j

x_0 is the input to bias node '0' which is always assumed as 1

θ_j is the weight in the link from the bias node '0' to Node j

*Net Input at Node j in the **Output Layer** is*

$$I_j = \sum_{i=1}^{n} O_i w_{ij} + x_0 \times \theta_j$$

where,

O_i is the output from Node i

w_{ij} is the weight in the link from Node i to Node j

x_0 is the input to bias node '0' which is always assumed as 1

θ_j is the weight in the link from the bias node '0' to Node j

Output at Node j

$$O_j = \frac{1}{1 + e^{-I_j}}$$

where,

I_j is the input received at Node j

3. **Estimate error at the node in the *Output Layer*:**

$$\text{Error} = O_{Desired} - O_{Estimated}$$

where,

$O_{Desired}$ is the desired output value of the Node in the Output Layer

$O_{Estimated}$ is the estimated output value of the Node in the Output Layer

Step 2: Backward Propagation

1. **Calculate Error at each node:**

*For each Unit k in the **Output Layer***

$$\text{Error}_k = O_k(1 - O_k)\,(O_{Desired} - O_k)$$

where,

O_k is the output value at Node k in the Output Layer.

$O_{Desired}$ is the desired output value of the Node in the Output Layer.

*For each unit j in the **Hidden Layer***

$$\text{Error}_j = O_j\,(1 - O_j)\,\Sigma_k\,\text{Error}_k\,w_{jk}$$

where,

O_j is the output value at Node j in the Hidden Layer.

Error_k is the error at Node k in the Output Layer.

w_{jk} is the weight in the link from Node j to Node k.

2. **Update all weights and biases:**

Update weights

$$\Delta w_{ij} = \propto \times \text{Error}_j \times O_i$$
$$w_{ij} = w_{ij} + \Delta w_{ij}$$

where,

O_i is the output value at Node i.

Error_j is the error at Node j.

$\propto$ is the learning rate.

w_{ij} is the weight in the link from Node i to Node j.

Δw_{ij} is the difference in weight that has to be added to w_{ij}.

Update Biases

$$\Delta\theta_j = \propto \times \text{Error}_j$$
$$\theta_j = \theta_j + \Delta\theta_j$$

where,

Error_j is the error at Node j.

$\propto$ is the learning rate.

θ_j is the bias value from Bias Node 0 to Node j.

$\Delta\theta_j$ is the difference in bias that has to be added to θ_j.

Radial Basis Function Neural Network

This networks have a fundamentally different architecture than most neural network architectures. Most neural network architecture consists of many layers and introduces nonlinearity by repetitively applying nonlinear activation functions.

RBF network on the other hand only consists of an input layer, a single hidden layer, and an output layer.

The input layer is not a computation layer, it just receives the input data and feeds it into the special hidden layer of the **RBF** network. The computation that is happened inside the hidden layer is very different from most neural networks, and this is where the power of the RBF network comes from. The output layer performs the prediction task such as classification or regression.

RBF Neural networks are conceptually similar **to K-Nearest Neighbor (k-NN) models.**

It is useful for interpolation, function approximation ,time series prediction and classification.

Typical Radial Basis Functions (RBF) are:

The Gaussian RBF which monotonically decreases with distance from the centre.

$$H(x) = e^{\frac{-(x-c)^2}{r^2}}$$

where, c is the centre and r is the radius.

A Multiquadric RBF which monotonically increases with distance from the centre.

$$H(x) = \sqrt{\frac{r^2 + (x-c)^2}{r}}$$

RBFNN Architecture :

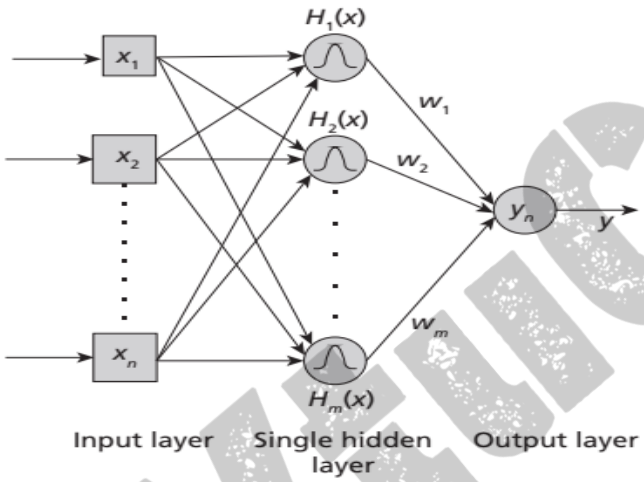


Figure : Architecture of RBFNN

Algorithm 10.3: Radial Basis Function Neural Network

Input: Input vector ($x_1, x_2, ..., x_n$)

Output: Y_n

Assign random weights for every connection from the Hidden layer to the Output layer in the network in the range [–1, +1].

Forward Phase:

Step 1: Calculate Input and Output in the Input Layer:

(Input layer is a direct transfer function, where the output of the node equals the input).

Input at Node i 'I_i' in the Input Layer is

$$\boxed{I_i = x_i}$$

where,

x_i is the input received at Node i.

Output at Node i 'O_i' in the Input Layer is

$$O_i = I_i$$

Step 2: For each node j in the Hidden Layer, find the centre/receptor c and the variance r. Define hidden layer neurons with Gaussian RBF whose output is:

$$H_j(x) = e^{\frac{-(x-c_j)^2}{r^2}}$$

where, x is the input, c_j is the centre and r is the radius.

Compute $(x - c_j)^2$ applying Euclidean distance measure between x and c_j.

Step 3: For each node k in the Output Layer, compute linear weighted sum of the output of each neuron k from the hidden layer neurons j.

$$F_k(x) = \sum_{j=1}^{m} w_{jk} H_j(x)$$

where,

w_{jk} is the weight in the link from the Hidden Layer neuron j to the Output Layer neuron k.

$H_j(x)$ is the output of a Hidden Layer neuron j for an input vector x.

Backward Phase:

Step 1: Train the Hidden layer using Back propagation.

Step 2: Update the weights between the Hidden layer and Output layer.

Self-organizing Feature Map

SOM is trained using unsupervised learning.

SOM doesn't learn by backpropagation with Stochastic Gradient Descent(SGD) ,it use **competitive learning** to adjust weights in neurons. Artificial **neural networks** often utilize **competitive learning** models to classify input without the use of **labeled data.**

Used: In dimension reduction to reduce our data by creating a spatially organized representation, also it help us to discover the correlation between data.

Self organizing maps have two layers, the first one is the input layer and the second one is the output layer or the feature map.

SOM doesn't have activation function in neurons, we directly pass weights to output layer without doing anything.

Network Architecture and operations

It consists of 2 layers:

1. **Input layer**
2. **Output layer**

   No Hidden layer.

The initialization of the weight to vectors initiates the mapping processes of the Self-Organizing Maps.

The mapped vectors are then examined to determine which weight most accurately represents the chosen sample using a sample random vector. Neighboring weights that are near each weighted vector are present. The chosen weight is allowed to turn into a vector for a random sample. This

encourages the map to develop and take on new forms. In a 2D feature space, they typically form hexagonal or square shapes. More than 1,000 times are spent repeatedly performing this entire process.

To put it simply, learning takes place in the following ways:

- To determine whether appropriate weights are similar to the input vector, each node is analyzed. The best matching unit is the term used to describe the appropriate node.
- The Best Matching Unit's neighborhood value is then determined. Over time, the neighbors tend to decline in number.

The appropriate weight further evolves into something more resembling the sample vector. The surrounding areas change similarly to the selected sample vector. A node's weights change more as it gets closer to the Best Matching Unit (BMU), and less as it gets farther away from its neighbor. For N iterations, repeat step two.

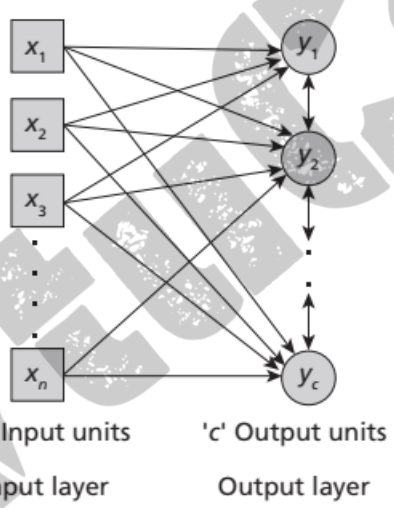


Figure : Network Architecture of Self Organizing Feature Map

Algorithm 10.4: Self-Organizing Feature Map

Input: 'm' Training Vectors (x_1, x_2, ..., x_m), each of length 'n':

$$\begin{bmatrix} x_{1,1} & x_{1,2} \ldots . x_{1,n} \\ x_{2,1} & x_{2,2} \ldots . x_{2,n} \\ & \\ x_{m,1} & x_{m,2} \ldots . x_{m,n} \end{bmatrix}$$

Output: 'c' categories
Input Layer: 'n' Input units (Length of the training vector)
Output Layer: 'c' Output units (Number of categories)
Step 1: Initialize random weights w_j between 0 and 1, for each Output unit j.
(A Weight vector of length 'n' is associated with each Output unit)
Step 2: For each training sample x_s:
(a) Compute Euclidean distance score between the training sample x_s and the weight vector w_j of each output unit:

$$d^2 = (\text{Euclidean distance})^2 = \sum_{k=1}^{n} (x_{s,k} - w_{j,k}(t))^2$$

(b) Choose the winning output unit which has smaller distance to the input sample as the best matching unit.

(c) Update the weights for the winning unit:

$$w_j(t+1) = w_j(t) + \eta(t)(x_s - w_j(t))$$

where,

$\eta(t)$ is the Learning rate.

$w_j(t)$ is the weight of the winning unit at step t.

$w_j(t+1)$ is the updated weight of the winning unit at step $(t+1)$.

Step 3: Repeat until the feature map doesn't change.

ANN learning mechanisms are used in many complex applications that involve modelling of non-linear processes. ANN is a useful model that can handle even noisy and incomplete data. They are used to model complex patterns, recognize patterns and solve prediction problems like humans in many areas such as:

1. Real-time applications: Face recognition, emotion detection, self-driving cars, navigation systems, routing systems, target tracking, vehicle scheduling, etc.
2. Business applications: Stock trading, sales forecasting, customer behaviour modelling, Market research and analysis, etc.
3. Banking and Finance: Credit and loan forecasting, fraud and risk evaluation, currency price prediction, real-estate appraisal, etc.
4. Education: Adaptive learning software, student performance modelling, etc.
5. Healthcare: Medical diagnosis or mapping symptoms to a medical case, image interpretation and pattern recognition, drug discovery, etc.
6. Other Engineering Applications: Robotics, aerospace, electronics, manufacturing, communications, chemical analysis, food research, etc.

Advantages and Disadvantages of ANN

1. ANN can solve complex problems involving non-linear processes.
2. ANNs can learn and recognize complex patterns and solve problems as humans solve a problem.
3. ANNs have a parallel processing capability and can predict in less time.
4. They have an ability to work with inadequate knowledge. It can even handle incomplete and noisy data.
5. They can scale well to larger data sets and outperforms other learning mechanisms.

Limitations of ANN

1. An ANN requires processors with parallel processing capability to train the network running for many epochs. The function of each node requires a CPU capability which is difficult for very large networks with a large amount of data.
2. They work like a 'black box' and it is exceedingly difficult to understand their working in inner layers. Moreover, it is hard to understand the relationship between the representations learned at each layer.

3. The modelling with ANN is also extremely complicated and the development takes a much longer time.
4. Generally, neural networks require more data than traditional machine learning algorithms, and they do not perform well on small datasets.
5. They are also more computationally expensive than traditional learning techniques.

Challenges of Artificial Neural Networks

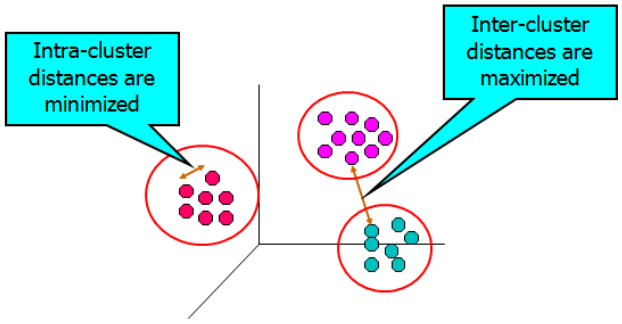
The major challenges while modelling a real-time application with ANNs are:

1. Training a neural network is the most challenging part of using this technique. Overfitting or underfitting issues may arise if datasets used for training are not correct. It is also hard to generalize to the real-world data when trained with some simulated data. Moreover, neural network models normally need a lot of training data to be robust and are usable for a real-time application.
2. Finding the weight and bias parameters for neural networks is also hard and it is difficult to calculate an optimal model.

Chapter 13

CLUSTERING ALGORITHMS

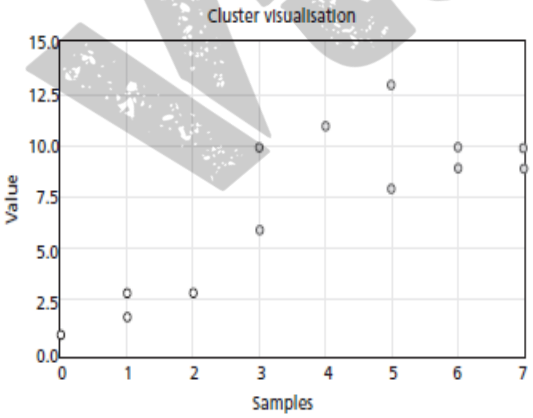
- **Clustering:** the process of grouping a set of objects into classes of similar objects
  - Documents within a cluster should be similar.
  - Documents from different clusters should be dissimilar

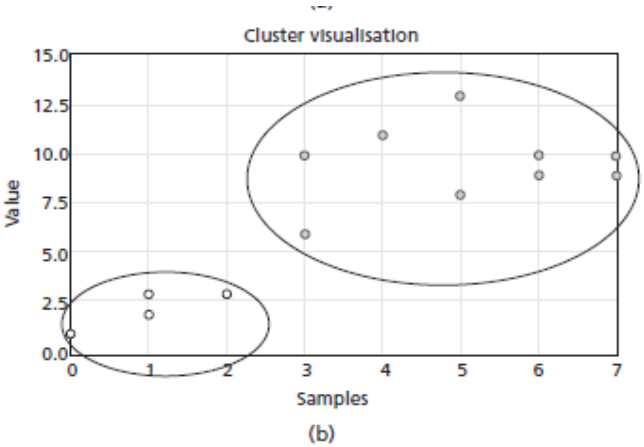


- Finding similarities between data according to the characteristics found in the data and grouping similar data objects into clusters.

- **Unsupervised** learning: no predefined classes.

- Example: Below fig: shows the data points with two features shown in different shaded samples.





(b)

If few similarities then manually we can do , but when examples have more features then cannot be done manually, so automatic clustering is required.

Clusters are represented by centroids.

Example:

(3,3),(2,6) and(7,9).

Centroid : (3+2+7,3+6+9)=(4,6). The clusters should not overlap and every cluster should represent only one class.

Difference between Clustering and Classification

S.No.	Clustering	Classification
1.	Unsupervised learning and cluster formation are done by trial and error as there is no supervisor	Supervised learning with the presence of a supervisor to provide training and testing data
2.	Unlabelled data	Labelled data
3.	No prior knowledge in clustering	Knowledge of the domain is must to label the samples of the dataset
4.	Cluster results are dynamic	Once a label is assigned, it does not change

Applications of Clustering

Applications of Clustering

1. Grouping based on customer buying patterns
2. Profiling of customers based on lifestyle
3. In information retrieval applications (like retrieval of a document from a collection of documents)
4. Identifying the groups of genes that influence a disease
5. Identification of organs that are similar in physiology functions
6. Taxonomy of animals, plants in Biology
7. Clustering based on purchasing behaviour and demography
8. Document indexing
9. Data compression by grouping similar objects and finding duplicate objects

Advantages and Disadvantages

Table 13.2: Advantages and Disadvantages of Clustering Algorithms

S.No.	Advantages	Disadvantages
1.	Cluster analysis algorithms can handle missing data and outliers.	Cluster analysis algorithms are sensitive to initialization and order of the input data.
2.	Can help classifiers in labelling the unlabelled data. Semi-supervised algorithms use cluster analysis algorithms to label the unlabelled data and then use classifiers to classify them.	Often, the number of clusters present in the data have to be specified by the user.
3.	It is easy to explain the cluster analysis algorithms and to implement them.	Scaling is a problem.
4.	Clustering is the oldest technique in statistics and it is easy to explain. It is also relatively easy to implement.	Designing a proximity measure for the given data is an issue.

Challenges of Clustering Algorithms

1. Collection of data with higher dimensions.
2. Designing a proximity measure is another challenge.
3. The curse of dimensionality

PROXIMITY MEASURES

Clustering algorithms need a measure to find the similarity or dissimilarity among the objects to group them. Similarity and Dissimilarity are collectively known as proximity measures. This is used by a number of data mining techniques, such as clustering, nearest neighbour classification, and anomaly detection.

Distance measures are known as **dissimilarity** measures, as these indicate how one object is different from another.

Measures like **cosine** similarity indicate the similarity among objects.

Distance measures and similarity measures are two sides of a same coin, as **more distance indicates more similarity and vice-versa.**

The properties of the distance measures are:

1. D_{ij} is always positive or zero.
2. $D_{ij} = 0$, i.e., the distance between the object to itself is 0.
3. $D_{ij} = D_{ji}$. This property is called symmetry.
4. $D_{ij} \leq D_{ik} + D_{kj}$. This property is called triangular inequality.

If all the **conditions** are satisfied, then the distance measure is called **metric**.

Some of proximity measures:

1. **Quantitative variables**

a) **Euclidean distance:** It is one of the most important and common distance measure. It is also called L2 norm.

The Euclidean distance between objects x_i and x_j with k features is given as follows:

$$\text{Distance}(x_i, x_j) = \sqrt{\sum_{k=1}^{n}(x_{ik} - x_{jk})^2} \qquad (13.1)$$

Advantage: The distance does not change with the addition of new object.

Disadvantage: i) If the unit changes, the resulting Euclidean or squared Euclidean Changes drastically.

ii) Computational complexity is high, because it involves square root and square.

b) City Block Distance: Known as Manhattan Distance or L1 norm.

$$\text{Distance}(x_i, x_j) = \sum_{k=1}^{n}\left|x_{ik} - x_{jk}\right|$$

c) Chebyshev Distance: Also known as maximum value distance. This is the absolute magnitude of the differences between the coordinates of a pair of objects.This distance is called supremum distance or Lmax or L_∞ norm.

$$\text{Distance}(x_i, x_j) = \max_k \left|x_{ik} - x_{jk}\right|$$

d) Minkowski Distance: In general, all the above distances measures can be generalized as:

$$\text{Distance}(x_i, x_j) = \left(\sum_{k=1}^{n}\left|x_{ik} - x_{jk}\right|^{\alpha}\right)^{1/\alpha}$$

Example Suppose, if the coordinates of the objects are (0, 3) and (5, 8), then what is the Chebyshev distance?

Solution: The Euclidean distance using Eq. (13.1) is given as follows:

$$\text{Distance}(x_i, x_j) = \sqrt{(0-5)^2 + (3-8)^2}$$

$$= \sqrt{50} = 7.07$$

The Manhattan distance using Eq. (13.2) is given as follows:

$$\text{Distance}(x_i, x_j) = |(0-5) + (3-8)| = 10$$

The Chebyshev distance using Eq. (13.3) is given as follows:

$$\text{Max}\{|0-5|, |3-8|\} = \text{Max}\{5, 5\} = 5$$

Binary Attributes: Binary Attributes have only two values. Distance measures have discussed above **cannot** be applied to find the distance between objects that have binary attributes. For finding the distance among objects with binary objects, the **contingency** table is used.

Table 13.3: Contingency Table

Attributes Matching	0	1
0	a	b
1	c	d

Simple Matching Coefficient (SMC) SMC is a simple distance measure and is defined as the ratio of number of matching attributes and the number of attributes. The formula is given as:

$$\frac{a+d}{a+b+c+d}$$

The values of a, b, c, and d can be observed from the Table 13.4.

Jaccard Coefficient Jaccard coefficient is another useful measure for and is given as follows:

$$J = \frac{d}{b+c+d}$$

Example If the given vectors are $x = (1, 0, 0)$ and $y = (1, 1, 1)$ then find the SMC and Jaccard coefficient?

Solution: It can be seen from Table 13.2 that, $a = 0$, $b = 2$, $c = 0$ and $d = 1$.

The SMC using Eq. (13.5) is given as $\frac{a+d}{a+b+c+d} = 0+1/3 = 0.33$

Jaccard coefficient using Eq. (13.6) is given as $J = \frac{d}{b+c+d} = 1/3 = 0.33$

Hamming Distance: Hamming distance is a metric for comparing two binary data strings. While comparing two binary strings of equal length, Hamming distance is the number of bit positions in which the two bits are different. It is used for error detection or error correction when data is transmitted over computer networks.

Example

Suppose there are two strings 1101 1001 and 1001 1101.

11011001 ⊕ 10011101 = 01000100. Since, this contains two 1s, the Hamming distance, d(11011001, 10011101) = 2.

2. Categorical variables

Categorical Variables

In many cases, categorical values are used. It is just a code or symbol to represent the values. For example, for the attribute Gender, a code 1 can be given to female and 0 can be given to male.

To calculate the distance between two objects represented by variables, we need to find only whether they are equal or not. This is given as:

$$\text{Distance}(x, y) = \begin{cases} 1 & \text{if } x \neq y \\ 0 & \text{if } x = y \end{cases}$$

Ordinal Variables

Ordinal Variables

Ordinal variables are like categorical values but with an inherent order. For example, designation is an ordinal variable. If job designation is 1 or 2 or 3, it means code 1 is higher than 2 and code 2 is higher than 3. It is ranked as $1 > 2 > 3$.

$$\text{Distance}(X, Y) = \frac{|position\ (X) - position\ (Y)|}{n - 1}$$

Cosine Similarity

- Cosine similarity is a metric used to measure how similar the documents are irrespective of their size.
- It measures the cosine of the angle between two vectors projected in a multi-dimensional space.
- The cosine similarity is advantageous because even if the two similar

documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together.

- The smaller the angle, higher the cosine similarity.
- Consider 2 documents P1 and P2.
  - If distance is more, then less similar.
  - If distance is less, then more similar.

$$sim(X,Y) = \frac{X \bullet Y}{\|X\| \times \|Y\|} = \frac{\sum_{i=1}^{n} X_i \times Y_i}{\sqrt{\sum_{i=1}^{n} X_i^2} \times \sqrt{\sum_{i=1}^{n} Y_i^2}}$$

Example 13.3: If the given vectors are $A = \{1, 1, 0\}$ and $B = \{0, 1, 1\}$, then what is the cosine similarity?

Solution: The dot product of the vector is $1 \times 0 + 1 \times 1 + 0 \times 1 = 1$. The norm of the vectors A and B is $\sqrt{2}$.

So, the cosine similarity using Eq. (13.9) is given as $\frac{1}{\sqrt{2}\sqrt{2}} = \frac{1}{2} = 0.5$

1. Consider the following data and, calculate the Euclidean, Manhattan and Chebyshev distances.

a. (2 3 4) and (1 5 6)

Solution

Euclidean distance = $\sqrt{(2-1)^2 + (3-5)^2 + (4-6)^2} = \sqrt{9} = 3$

Manhattan distance = $|2-1| + |3-5| + |4-6| = 1+2+2 = 5$

Chebyshev Distance = $\max\{|2-1|, |3-5|, |4-6|\} = \max\{1,2,2\} = 2$

b. (2 2 9) and (7 8 9)

Euclidean Distance = $\sqrt{(2-7)^2 + (2-8)^2 + (9-9)^2} = \sqrt{25+36+09} = \sqrt{61} = 7.81$

Manhattan Distance = $|2-7| + |2-8)| + |9-9| = 5+6+0 = 11$

Chebyshev Distance = $\max\{|2-7|+|2-8)|+|9-9|\} = \{5,6,0\} = 6$

2. Find cosine similarity, SMC and Jaccard coefficients for the following binary data:

a. (1 0 1 1) and (1 1 0 0)

Solution

1 0 1 1

1 1 0 0

C = 2, b = 1, d = 1,

SMC = $\frac{a+d}{a+b+c+d} = \frac{1}{4} = 0.25$

Jaccard Coefficient = $\frac{d}{b+c+d} = \frac{1}{4} = 0.25$

Cosine Similarity = $\frac{(1\times1+0\times1+1\times0+1\times0)}{\sqrt{3}\sqrt{2}} = \frac{1}{\sqrt{3}\sqrt{2}}$

b. (1 0 0 0 1) and (1 0 0 0 0 1)

Solution

No match

(1 0 0 0 1) and (1 1 0 0 0)

1 0 0 0 1

1 1 0 0 0

A=2, b= 1, c = 1, d= 1

SMC = $\frac{a+d}{a+b+c+d} = \frac{2}{5} = 0.5$

Jaccard Coefficient = $\frac{d}{b+c+d} = \frac{1}{3} = 0.33$

Cosine Similarity = $\frac{(1\times1+0\times1+0\times0+0\times0+1\times0)}{} = \frac{1}{} = 0.5$

$$\frac{}{\sqrt{2}\sqrt{2}} \qquad \frac{}{\sqrt{2}\sqrt{2}} \quad \frac{}{2}$$

3. Find Hamming distance for the following binary data:

a. (1 1 1) and (1 0 0)

Solution

It differs in two positions; therefore Hamming distance is 2

b. (1 1 1 0 0) and (0 0 1 1 1)

Solution

It differs in four positions; therefore, Hamming distance is 4

4. Find the distance between:

a. Employee ID: 1000 and 1001

Solution

They are not equal. Therefore, distance is 0

b. Employee name – John & John and John & Joan

Solution

The distance between John and John is 1

The distance between John and Joan is 0

5. Find the distance between:

a. (Yellow, red, green) and (red, green, yellow)

Solution

Yellow = 1, red = 2, Green = 3

Therefore, the distance between (yellow, red) $= \left|\frac{1-2}{2}\right| = \left|\frac{-1}{2}\right| = \frac{1}{2} = 0.5$

Distance between (red, green) $= \left|\frac{2-3}{2}\right| = \left|\frac{-1}{2}\right| = \frac{1}{2} = 0.5$

Distance between (green, yellow) $= \left|\frac{3-1}{2}\right| = \left|\frac{2}{2}\right| = 1$

Therefore, distance between (Yellow, red, green) and (red, green, yellow) is (0.5,0.5,1).

b. (bread, butter, milk) and (milk, sandwich, Tea)

Solution

Bread =1, Butter =2, Milk = 3, Sandwich = 4, Tea = 5

The distance between (bread, milk) = $\left|\frac{1-3}{5-1}\right| = \left|\frac{-2}{4}\right| = \frac{1}{2}$

The distance between (butter, sandwich) = $\left|\frac{2-4}{5-1}\right| = \left|\frac{-2}{4}\right| = \frac{1}{2}$

The distance between (Milk, Tea) = $\left|\frac{3-5}{5-1}\right| = \left|\frac{-2}{4}\right| = \frac{1}{2}$

Therefore, the distance between (bread, butter, milk) and (milk, sandwich, Tea) =

$$\begin{pmatrix} 1 & 1 & 1 \\ -, & -, & - \\ 2 & 2 & 2 \end{pmatrix}$$

Hierarchical Clustering Algorithms

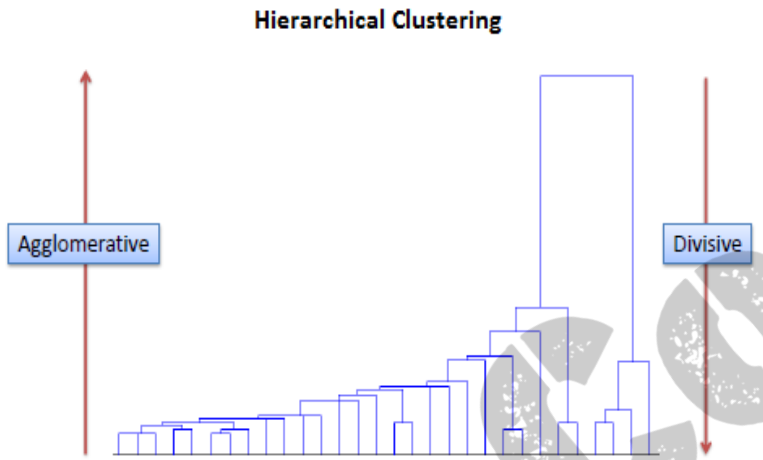
Hierarchical clustering involves creating clusters that have a predetermined ordering from top to bottom.

For example, all files and folders on the hard disk are organized in a hierarchy.

Hierarchical relationship is shown in the form of dendogram.

There are two types of hierarchical clustering.

- *Divisive* and *Agglomerative*.



- **Divisive method :** In *divisive* or *top-down clustering* method we assign all of the observations to a single cluster and then partition the cluster to two least similar clusters. Finally, we proceed recursively on each cluster until there is one cluster for each observation. There is evidence that divisive algorithms produce more accurate hierarchies than agglomerative algorithms in some circumstances but is conceptually more complex.
- **Agglomerative method:** In *agglomerative* or *bottom-up clustering* method we assign each observation to its own cluster. Then, compute the similarity (e.g., distance) between each of the clusters and join the two most similar clusters. Finally, repeat steps 2 and 3 until there is only a single cluster left. The related algorithm is shown below.

Algorithm 13.1: Agglomerative Clustering

1. Place each N sample or data instance into a separate cluster. So, initially N clusters are available.
2. Repeat the following steps until a single cluster is formed:
   (a) Determine two most similar clusters.
   (b) Merge the two clusters into a single cluster reducing the number of clusters as N–1.
3. Choose resultant clusters of step 2 as result.

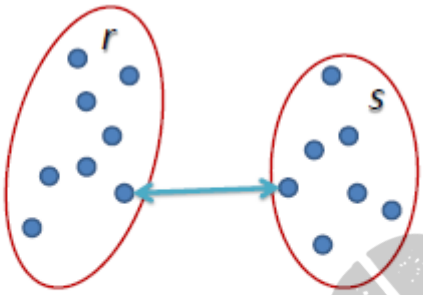
- The following three methods differ in how the distance between each cluster is

measured.

1. Single Linkage
2. Average Linkage
3. Complete Linkage

Single Linkage or MIN algorithm

In single linkage hierarchical clustering, the distance between two clusters is defined as the *shortest* distance between two points in each cluster. For example, the distance between clusters "r" and "s" to the left is equal to the length of the arrow between their two closest points.



$$L(r,s) = \min(D(x_{ri}, x_{sj}))$$

Example 13.4: Consider the array of points as shown in the following Table. 13.4.

Table 13.4: Sample Data

Objects	X	Y
0	1	4
1	2	8
2	5	10
3	12	18
4	14	28

Apply single linkage algorithm.

Solution: The similarity table among the variables is computed as shown in Table 13.4. Euclidean distance is computed as shown in the following Table 13.5.

Table 13.5: Proximity Matrix

Objects	0	1	2	3	4
0	–	4.123	7.211	17.804	27.295
1		–	3.606	14.142	23.324
2			–	10.630	20.124
3				–	10.198
4					–

The minimum distance is 3.606. Therefore, the items 1 and 2 are clustered together. The resultant is shown in Table 13.6.

Table 13.6: After Iteration 1

Clusters	{1, 2}	0	3	4
{1, 2}	–	4.123	10.630	20.124
0		–	17.804	27.295
3			–	10.198
4				–

The distance between the group {1, 2} and items {0}, {3} and {4} is computed using the formula:

$$D_{SL}(C_i, C_j) = \text{minimum}_{a \in C_i, b \in C_j} d(a, b) \quad (13.10)$$

Here, D_{SL} is the single linkage distance, C_i, C_j are clusters and $d(a, b)$ is the distance between the elements a and b.

Thus, the distance between {1, 2} and {0} is:

Minimum {{1, 0}, {2, 0}} = Minimum {4.123, 7.211} = 4.123

The distance between {1, 2} and {3} is given as:

Minimum {{1, 3}, {2, 3}} = Minimum {14.412, 10.630} = 10.630

The distance between {1, 2} and {4} is given as:

Minimum {{1, 4}, {2, 4}} = Minimum {23.324, 20.124} = 20.124

This is shown in Table 13.6. The minimum distance of Table 13.6 is 4.123.

Therefore, {0, 1, 2} is clustered together. This result is shown in Table 13.7.

Table 13.7: After Iteration 2

Clusters	{0, 1, 2}	3	4
{0, 1, 2}	–	10.630	20.124
3		–	10.198
4			–

Thus, the distance between {0, 1, 2} and {3} using Eq. (13.10) is given as

Minimum {{0, 3}, {1, 3}, {2, 3}} = Minimum {17.804, 14.142, 10.630} = 10.630.

Thus, the distance between {0, 1, 2} and {4} is:

Minimum {{0, 4}, {1, 4}, {2, 4}} = Minimum {27.295, 23.324, 20.124} = 20.124.

This is shown in Table 13.7. The minimum is 10.198.

Therefore, the items {3, 4} are merged. And, the items {1, 2, 3} and {4, 5} are merged. The resultant is shown in Table 13.8.

Table 13.8: After Iteration 3

Clusters	{0, 1, 2}	{3, 4}
{0, 1, 2}	–	?
{3, 4}		–

The computation of ? in Table 13.8 is not needed as no other items are left. Therefore, the clusters {0, 1, 2} and {3, 4} are merged.

Dendrograms are used to plot the hierarchy of clusters. Dendrogram for the above clustering is shown in the following Figure 13.2.

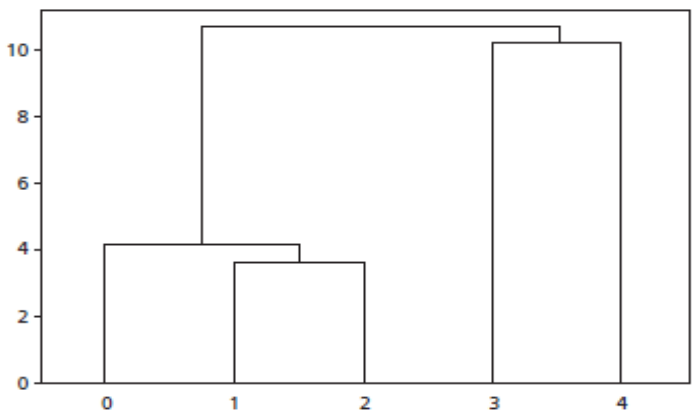
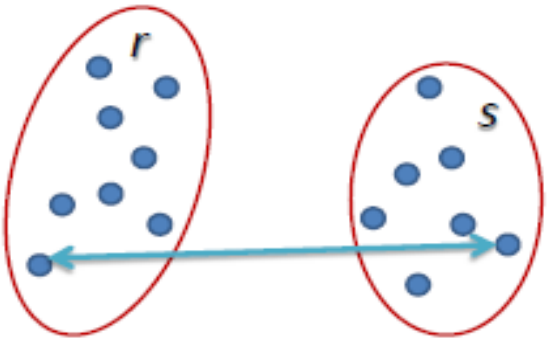


Figure 13.2: Dendrogram for Table 13.4

- **Complete Linkage :** In complete linkage hierarchical clustering, the distance between two clusters is defined as the *longest* distance between two points in each cluster. For example, the distance between clusters "r" and "s" to the left is equal to the length of the arrow between their two furthest points.



$$L(r,s) = \max(D(x_{ri}, x_{sj}))$$

OR

$$D_{CL}(C_i, C_j) = \text{maximum}_{a \in C_i,\, b \in C_j} d(a, b)$$

Dendrogram for the above clustering is shown in Figure 13.3.

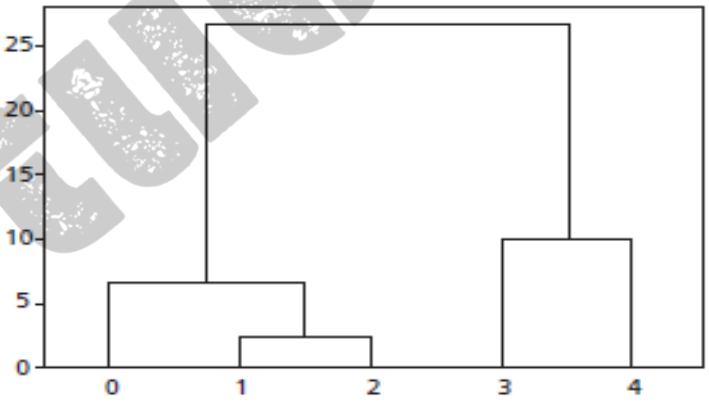
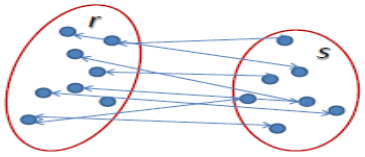


Figure 13.3: Dendrogram for Table 13.4

- **Average Linkage :** In average linkage hierarchical clustering, the distance between two clusters is defined as the average distance between each point in one cluster to every point in the other cluster. For example, the distance between clusters "r" and "s" to the left is equal to the average length each arrow between connecting the points of one cluster to the other.



$$L(r,s) = \frac{1}{n_r n_s} \sum_{i=1}^{n_r} \sum_{j=1}^{n_s} D(x_{ri}, x_{sj})$$

In case of an average linkage algorithm, the average distance of all pairs of points across the clusters is used to form clusters. The average value computed between clusters c_i, c_j is given as follows:

$$D_{AL}(C_i, C_j) = \frac{1}{m_i m_j} \sum_{a \in C_i, b \in C_j} d(a, b) \tag{13.12}$$

Here, m_i and m_j are sizes of the clusters.

The dendrogram for Table 13.4 is given in Figure 13.4.

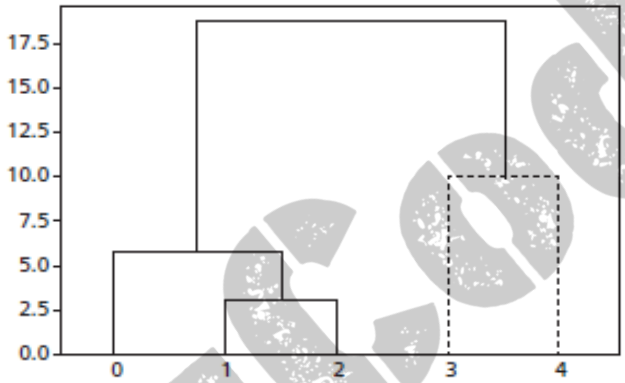


Figure 13.4: Dendrogram for Table 13.4

Mean-Shift Algorithm

Algorithm 13.2: Mean-Shift Clustering

Step 1: Design a window.

Step 2: Place the window on a set of data points.

Step 3: Compute the mean for all the points that come under the window.

Step 4: Move the center of the window to the mean computed in step 3. Thus, the window moves towards the dense regions. The movement to the dense region is controlled by a mean shift vector. The mean shift vector is given as:

$$v_s = \frac{1}{K} \sum_{x_i \in S_k} (x_i - x) \tag{13.13}$$

Here, K is the number of points and S_k is the data points where the distance from data points x_i and centroid of the kernel x is within the radius of the sphere. Then, the centroid is updated as $x = x + v_s$.

Step 5: Repeat the steps 3–4 for convergence. Once convergence is achieved, no further points can be accommodated.

Use dataset and apply hierarchical methods. Show the dendrogram.

SNo.	*X*	*Y*
1.	3	5
2.	7	8
3.	12	5
4.	16	9
5.	20	8

Table Sample Data

Solution

The similarity table among the variables is computed and is shown in Table 134.4. Euclidean distance is computed and is shown in the following Table 143.57.

Table 134.57: Proximity Matrix

Objects	0	1	2	3	4
0	-	5	9	9.85	17.26
1		-	5.83	9.49	13
2			-	5.66	8.94
3				-	4.12
4					-

The minimum distance is 4.12. Therefore, the items 1 and 4 are clustered together. The resultant table is given as shown in the following Table.

Table After Iteration 1

Clusters	{1,4}	2	3	5
{1,4}	-	5	5.66	4.12
2		-	5.83	13

3			-	8.94
5				-

The distance between the group {1, 4} and items 2, 3, 5 are computed using this formula.

Thus, the distance between {1,4} and {2} is:

Minimum { {1,4}, {2} = Minimum {(1,2),(4,2)=5

The distance between {1,4} and {3} is given as:

Minimum { {1,3}, {4,3} } = Minimum {9,5.66}=5.66

The distance between {1,4} and {5} is given as:

Minimum { {1,5}, {2,5} } = Minimum {17.26,4.12} = 4.12

The minimum distance of above table is 4.12. Therefore, {1,4} and {5} are combined. This results in the following Table.

Table After Iteration 2

Clusters	**{1,4,5}**	2	**5**
{1,4,5}	-	5	5.66
2		-	5.83
5			-

Thus, the distance between {1,4,5} and {2} is:

Minimum {(1,2),(4,2),(5,2)}= {5,9.49,13} = 5

Thus, the distance between {1,4,5} and {3} is:

Minimum { {1,3}, {4,3},{5,3)} = Minimum {9,5.66,8.94} = 5.66

The minimum is 5. Therefor {1,4,5} and {2} is combined. And finally, it is combined with {3}.

therefore, the order of cluster is {1,4} then {5}, then {2} and finally {3}.

Complete Linkage or MAX or Clique

Here from the first iteration table minimum is taken and {1,4} is combined. Then maximum is computed as

Thus, the distance between {1,4} and {2} is:

Max{ {1,4}, {2} = Max {(1,2),(4,2)= 9.49

The distance between {1,4} and {3} is given as:

Max { {1,3}, {4,3} } = Max {9,5.66}=9

The distance between {1,4} and {5} is given as:

Max{ {1,5}, {2,5} } = Max {17.26,4.12} = 17.26

This results in a Table

Clusters	**{1,4}**	2	**3**	**5**
{1,4}	-	9.49	9	17.26
2		-	5.83	13
3			-	8.94
5				-

So, the minimum is 8.94. Therefore, {3,5} is combined. This is shown in the following Table.

Clusters	**{1,4}**	{3,5}	**2**
{1,4}	-	17.26	9.49
{3,5}		-	13
2			-

The minimum is 9.49. Therefore {1,4,2} are combined. The order of cluster is {1,4}, {1,4} and {2}, and {3,5}.

Hint: The same is used for average link algorithm where the average distance of all pairs of points across the clusters is used to form clusters.

Consider the following data shown in Table 143.125. Use *k*-means algorithm with *k* = 2 and show the result.

Table Sample Data

SNO	***X***	***Y***
1.	3	5
2.	7	8
3.	12	5
4.	16	9

Solution

Let us assume the seed points are (3,5) and (16,9). This is shown in the following table as starting clusters.

Table Initial Cluster Table

Cluster 1	**Cluster 2**
(3,5)	(16,9)
Centroid (3,5)	Centroid (16,9)

Iteration 1: Compare all the data points or samples with the centroid and assigned to the nearest sample.

Take the sample object 2 and compare it with the two centroids as follows:

Dist(2,centroid 1) = $\sqrt{(7-3)^2+(8-5)^2} = \sqrt{16+9} = \sqrt{25} = 5$

Dist(2,centroid 2) = $\sqrt{(7-16)^2+(8-9)^2} = \sqrt{81+1} = \sqrt{82} = 9.05$

Object 2 is closer to centroid of cluster 1 and hence assign it to the cluster 1. This is shown in Table. For the object 3:,

Dist(3,centroid 1) = $\sqrt{(12-3)^2+(5-5)^2} = \sqrt{81} = 9$

Dist(3,centroid 2) = $\sqrt{(12-16)^2+(5-9)^2} = \sqrt{16+16} = \sqrt{32} = 5.66$

Object 3 is closer to centroid of cluster 2. and hence remains in the same cluster 1

This is shown in the following Table.

Table Cluster Table After Iteration 1

Cluster 1	**Cluster 2**
(3,5)	(12,4)
(7,8)	(10,4)
Centroid (10/2,13/2)=(5,6.5)	Centroid (28/2,14/2)=(14,7)

The second iteration is started again. Compute again,

Dist(1,centroid 1) = $\sqrt{(7-5)^2+(8-6.5)^2} = 6.25$

Dist(1,centroid 2) = $\sqrt{(12-14)^2+(8-7)^2} = \sqrt{49+1} = \sqrt{50} = 7.07$

Object 1 is closer to centroid of cluster 1 and hence remains in the same cluster. Take the sample object 3, compute again

Dist(3,centroid 1) = $\sqrt{(12-5)^2+(5-6.5)^2} = \sqrt{51.25} = 7.16$

Dist(3,centroid 2) = $\sqrt{(16-14)^2+(9-7)^2} = \sqrt{4+4} = \sqrt{8} = 2.82$

Object 3 is closer to centroid of cluster 2 and remains in the same cluster.

Therefore, the resultant clusters are

{(3,5), (7,80} and {(12,5),(16,9)}.

PARTITIONAL CLUSTERING ALGORITHMS

It is a type of clustering that divides the data into non-hierarchical groups. It is also known as the centroid-based method. The most common example of partitioning clustering is the K-Means Clustering algorithm.

In this type, the dataset is divided into a set of k groups, where K is used to define the number of pre-defined groups. The cluster center is created in such a way that the distance between the data points of one cluster is minimum as compared to another cluster centroid.

Algorithm : *k*-means Algorithm

Step 1: Determine the number of clusters before the algorithm is started. This is called *k*.

Step 2: Choose *k* instances randomly. These are initial cluster centers.

Step 3: Compute the mean of the initial clusters and assign the remaining sample to the closest cluster based on Euclidean distance or any other distance measure between the instances and the centroid of the clusters.

Step 4: Compute new centroid again considering the newly added samples.

Step 5: Perform the steps 3–4 till the algorithm becomes stable with no more changes in assignment of instances and clusters.

K means can be viewed as greedy algorithm as it involves partitioning 'n' samples to k clusterd to minimize sum of squared Error. SSE is a metric that is a measure of error that gives the sum of the squared Euclidean distances of each data to its closet centroid.

SSE= $\sum f(x) = \sum_{i=1}^{k}$**(dist(ci** ,x)2)

Here c_i = centroid of ith cluster

x=sample data

Advantages

1. Simple
2. Easy to implement

Disadvantages

1. It is sensitive to initialization process as change of initial points leads to different clusters.
2. If the samples are large, then the algorithm takes a lot of time.

Complexity

The complexity of *k*-means algorithm is dependent on the parameters like n, the number of samples, k, the number of clusters, $\Theta(nkld)$. I is the number of iterations and d is the number of attributes. The complexity of *k*-means algorithm is $O(n^2)$.

PROBLEM

Example 13.5: Consider the following set of data given in Table 13.9. Cluster it using *k*-means algorithm with the initial value of objects 2 and 5 with the coordinate values (4, 6) and (12, 4) as initial seeds.

Table 13.9: Sample Data

Objects	X-coordinate	Y-coordinate
1	2	4
2	4	6
3	6	8
4	10	4
5	12	4

Solution: As per the problem, choose the objects 2 and 5 with the coordinate values. Hereafter, the objects' id is not important. The samples or data points (4, 6) and (12, 4) are started as two clusters as shown in Table 13.10.

Initially, centroid and data points are same as only one sample is involved.

Table 13.10: Initial Cluster Table

Cluster 1	Cluster 2
(4, 6)	(12, 4)
Centroid 1 (4, 6)	Centroid 2 (12, 4)

Iteration 1: Compare all the data points or samples with the centroid and assign to the nearest sample. Take the sample object 1 (2, 4) from Table 13.9 and compare with the centroid of

Dist (1, centroid 1) = $\sqrt{(2-4)^2 + (4-6)^2} = \sqrt{8}$

Dist (1, centroid 2) = $\sqrt{(2-12)^2 + (4-4)^2} = \sqrt{100} = 10$

Object 1 is closer to the centroid of cluster 1 and hence assign it to cluster 1. This is shown in Table 13.11. Object 2 is taken as centroid point.

For the object 3 (6, 8), the Euclidean distance between it and the centroid points is given as:

Dist (3, centroid 1) = $\sqrt{(6-4)^2 + (8-6)^2} = \sqrt{8}$

Dist (3, centroid 2) = $\sqrt{(6-12)^2 + (8-4)^2} = \sqrt{52}$

Object 3 is closer to the centroid of cluster 1 and hence remains in the same cluster 1.

Proceed with the next point object 4(10, 4) and again compare it with the centroids in Table 13.10.

Dist (4, centroid 1) = $\sqrt{(10-4)^2 + (4-6)^2} = \sqrt{40}$

Dist (4, centroid 2) = $\sqrt{(10-12)^2 + (4-4)^2} = \sqrt{4} = 2$

Object 4 is closer to the centroid of cluster 2 and hence assign it to the cluster table. Object 4 is in the same cluster. The final cluster table is shown in Table 13.11.

Obviously, Object 5 is in Cluster 3. Recompute the new centroids of cluster 1 and cluster 2. They are (4, 6) and (11, 4), respectively.

Table 13.11: Cluster Table After Iteration 1

Cluster 1	Cluster 2
(4, 6)	(10, 4)
(2, 4)	(12, 4)
(6, 8)	
Centroid 1 (4, 6)	Centroid 2 (11, 4)

The second iteration is started again with the Table 13.11.

Obviously, the point (4, 6) remains in cluster 1, as the distance of it with itself is 0. The remaining objects can be checked. Take the sample object 1 (2, 4) and compare with the centroid of the clusters in Table 13.12.

Dist (1, centroid 1) = $\sqrt{(2-4)^2 + (4-6)^2} = \sqrt{8}$

Dist (1, centroid 2) = $\sqrt{(2-11)^2 + (4-4)^2} = \sqrt{81} = 9$

Object 1 is closer to centroid of cluster 1 and hence remains in the same cluster. Take the sample object 3 (6, 8) and compare with the centroid values of clusters 1 (4, 6) and cluster 2 (11, 4) of the Table 13.12.

Dist (3, centroid 1) = $\sqrt{(6-4)^2 + (8-6)^2} = \sqrt{8}$

Dist (3, centroid 2) = $\sqrt{(6-11)^2 + (8-4)^2} = \sqrt{41}$

Object 3 is closer to centroid of cluster 1 and hence remains in the same cluster. Take the sample object 4 (10, 4) and compare with the centroid values of clusters 1 (4, 6) and cluster 2 (11, 4) of the Table 13.12:

Dist (4, centroid 1) = $\sqrt{(10-4)^2 + (4-6)^2} = \sqrt{40}$

Dist (3, centroid 2) = $\sqrt{(10-11)^2 + (4-4)^2} = \sqrt{1} = 1$

Object 3 is closer to centroid of cluster 2 and hence remains in the same cluster. Obviously, the sample (12, 4) is closer to its centroid as shown below:

Dist (5, centroid 1) = $\sqrt{(12-4)^2 + (4-6)^2} = \sqrt{68}$

Dist (5, centroid 2) = $\sqrt{(12-11)^2 + (4-4)^2} = \sqrt{1} = 1$. Therefore, it remains in the same cluster. Object 5 is taken as centroid point.

The final cluster Table 13.12 is given below:

Table 13.12: Cluster Table After Iteration 2

Cluster 1	Cluster 2
(4, 6)	(10, 4)
(2, 4)	(12, 4)
(6, 8)	
Centroid (4, 6)	Centroid (11, 4)

There is no change in the cluster Table 13.12. It is exactly the same; therefore, the *k*-means algorithm terminates with two clusters with data points as shown in the Table 13.12.

Density-Based Clustering

A cluster is a dense region of points, which is separated by low-density regions, from other regions of high density.

Used when the clusters are irregular or intertwined, and when noise and outliers are present.

Density-Based Clustering refers to unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density.

Density-Based Spatial Clustering of **Applications with Noise (DBSCAN)** is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.

The DBSCAN algorithm uses two parameters:

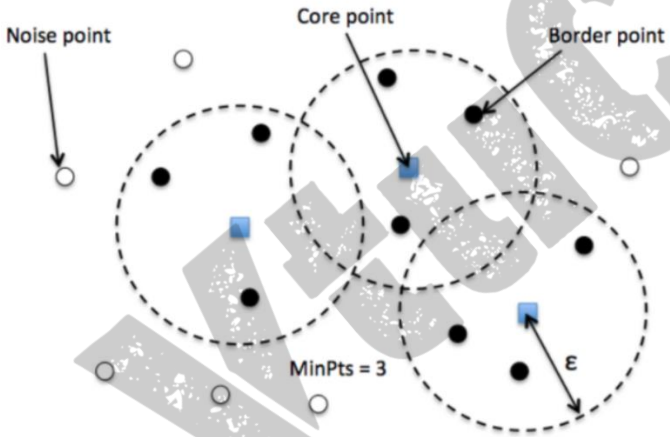
minPts: The minimum number of points (a threshold) clustered together for a region to be considered dense.

eps (ε): A distance measure that will be used to locate the points in the neighborhood of any point.

These parameters can be understood if we explore two concepts called Density Reachability and Density Connectivity.

Reachability in terms of density establishes a point to be reachable from another if it lies within a particular distance (eps) from it. Connectivity, on the other hand, involves a transitivity based chaining-approach to determine whether points are located in a particular cluster. For example, p and q points could be connected if p->r->s->t->q, where a->b means b is in the neighborhood of a.

There are **three** types of points after the **DBSCAN** clustering is complete:



- **Core** — This is a point that has at least m points within distance n from itself.
- **Border** — This is a point that has at least one Core point at a distance n.
- **Noise** — This is a point that is neither a Core nor a Border. And it has less than m points within distance n from itself.

The following connectedness measures are used for this algorithm.

1. Direct density reachable – The point X is directly reachable from Y, if:
   (a) X is the ε-neighborhood of Y
   (b) Y is a core point
2. Densely reachable – The point X is densely reachable from Y, if there is a set of core points that leads from Y to X.
3. Density connected – X and Y are densely connected if Z is a core point and thus points X and Y are densely reachable from Z.

Algorithm 13.4: DBSCAN

Step 1: Randomly select a point p. Compute distance between p and all other points.

Step 2: Find all points from p with respect to its neighbourhood and check whether it has minimum number of points m. If so, it is marked as a core point.

Step 3: If it is a core point, then a new cluster is formed, or existing cluster is enlarged.

Step 4: If it is a border point, then the algorithm moves to the next point and marks it as visited.

Step 5: If it is a noise point, they are removed.

Step 6: Merge the clusters if it is mergeable, $dist\ (c_i, c_j) < \varepsilon$.

Step 7: Repeat the process 3–6 till all points are processed.

Advantages

1. No need for specifying the number of clusters beforehand
2. The algorithm can detect clusters of any shapes
3. Robust to noise
4. Few parameters are needed

The complexity of this algorithm is $O(n \log n)$.

Grid-Based Approaches

grid-based clustering method takes a **space-driven** approach by partitioning the embedding space into *cells* independent of the distribution of the input objects.

The *grid-based clustering* approach uses a **multiresolution** grid data structure. It quantizes the object space into a finite number of cells that form a grid structure on which all of the operations for clustering are performed.

The main **advantage** of the approach is its **fast processing time**, which is typically independent of the number of data objects, yet dependent on only the number of cells.

There are three important concepts that need to be mastered for understanding the grid-based schemes. They are:

1. Subspace clustering
2. Concept of dense cells
3. Monotonicity property

Subspace Clustering

Grid-based algorithms are useful for clustering high-dimensional data, that is, data with many attributes. Some data like gene data may have millions of attributes. Every attribute is called a dimension. But all the attributes are not needed, as in many applications one may not require all the attributes. For example, an employee's address may not be required for profiling his diseases. Age may be required in that case. So, one can conclude that only a subset of features is required.

CLIQUE is a density-based and grid-based subspace clustering algorithm, useful for finding clustering in subspace.

Concept of Dense cell

CLIQUE partitions each dimension into several overlapping intervals and intervals it into cells. Then, algorithm determines whether the cells is dense or sparse. The cell is considered dense if it exceeds a threshold value.

It is **defined** as the ratio of number of points and volume of the region. In one pass, the algorithm finds the number of cells , number of points etc and then combines the dense cells. For that the algorithm uses the contiguous intervals and a set of dense cells.

Algorithm 13.5: Dense Cells

Step 1: Define a set of grid points and assign the given data points on the grid.

Step 2: Determine the dense and sparse cells. If the number of points in a cell exceeds the threshold value τ, the cell is categorized as dense cell. Sparse cells are removed from the list.

Step 3: Merge the dense cells if they are adjacent.

Step 4: Form a list of grid cells for every subspace as output.

MONOTONICITY Property

CLIQUE uses anti- monotonicity property or apriori algorithm. It means that all the subsets of a frequent itemset are frequent. Similarly if the subset is infrequent then its superset are infrequent.

Algorithm works in 2 stages:

Algorithm 13.6: CLIQUE

Stage 1:

Step 1: Identify the dense cells.

Step 2: Merge dense cells c_1 and c_2 if they share the same interval.

Step 3: Generate Apriori rule to generate $(k + 1)^{th}$ cell for higher dimension. Then, check whether the number of points cross the threshold. This is repeated till there are no dense cells or new generation of dense cells.

Stage 2:

Step 1: Merging of dense cells into a cluster is carried out in each subspace using maximal regions to cover dense cells. The maximal region is an hyperrectangle where all cells fall into.

Step 2: Maximal region tries to cover all dense cells to form clusters.

In stage two, CLIQUE starts from dimension 2 and starts merging. This process is continued till the n-dimension.

Advantages of CLIQUE

1. Insensitive to input order of objects
2. No assumptions of underlying data distributions
3. Finds subspace of higher dimensions such that high-density clusters exist in those subspaces

Disadvantage

The disadvantage of CLIQUE is that tuning of grid parameters, such as grid size, and finding optimal threshold for finding whether the cell is dense or not is a challenge.

PROBABILITY MODEL BASED METHODS

Probability model-based methods in clustering are a class of techniques that use statistical models to represent the underlying probability distributions of data points in a dataset.

These methods are used to group similar data points together into clusters based on their likelihood of belonging to a particular cluster according to the assumed probability distribution.

Two **popular** probability model-based clustering methods are Gaussian Mixture Models (GMMs) and Hidden Markov Models (HMMs). other than these we have other set of model . those are:

1. **Fuzzy Clustering**
2. **EM algorithm**

Fuzzy Clustering :

Fuzzy Clustering is a type of clustering algorithm in machine learning that allows a data point to belong to more than one cluster with different degrees of membership. Unlike traditional clustering algorithms, such as k-means or hierarchical clustering, which assign each data point to a single cluster, fuzzy clustering assigns a membership degree between 0 and 1 for each data point for each cluster.

Let us consider c_i and c_j then an element say x, can belong to both the cluster.The strength of the association of an object with the cluster is given as w_{ij} . The value of w_{ij} lies between 0 and 1. The sum of the weights of an object, if added, gives 1.

Algorithm 13.7: Fuzzy C-Means

Step 1: Choose the clusters, k, randomly.

Step 2: Assign weights w_{ij} of objects to clusters randomly.

Step 3: Compute the centroid:

$$c_j = \frac{\sum_{i=1}^{m} w_{ij}^p x_i}{\sum_{i=1}^{m} w_{ij}^p} \quad (13.15)$$

Step 4: The Sum of Squared Error (SSE) is computed as:

$$SSE = \sum_{j=1}^{k} \sum_{i=1}^{m} w_{ij}^p dist\ (x_i, c_j)^2 \quad (13.16)$$

Step 5: Minimize SSE to update the membership weights. Here, p is a fuzzifier whose value ranges from 1 to ∞. This parameter determines the influence of the weights. If p is 1, then fuzzy-c acts like k-means algorithm. A large weight results in a smaller value of the membership and hence more fuzziness. Typically, p value is 2.

Step 6: Repeat steps 3–5 till convergence is reached, which mean when there is no change in weights exceeding the threshold value.

Advantages and Disadvantages of FCM

The advantages and disadvantages of the FCM algorithm are given in Table 13.13.

Table 13.13: Advantages and Disadvantages of FCM Algorithm

S.No.	Advantages	Disadvantages
1.	Minimum intra class variance	The quality depends on the initial choice of weights
2.	Robust to noise	Local minima rather than global minimum

Expectation Maximization Algorithm

The Expectation-Maximization (EM) algorithm is a statistical method used for estimating parameters in statistical models when you have incomplete or missing data. It's commonly used in unsupervised machine learning tasks such as clustering and Gaussian Mixture Model (GMM) fitting.

Given a mix of distributions, data can be generated by randomly picking a distribution and generating the point. Gaussian distribution is a bell shaped curve.

The function of Gaussian distribution is given by:

... is given as

$$N(x : \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)}{2\sigma^2}}$$

Two parameters characterize this function – mean and standard deviation. Sometimes, variance can also be used as it is the square of standard deviation. When the mean is zero, the peak of the bell-shaped curve occurs. Standard deviation is the spread of the shape. The above function is called probability distribution function that tells how to find the probability of the observed point x.

The same gaussian function can be extended for multivariate too. In 2D, the mean is also a vector and variance takes the form of covariance matrix. Chapter 3 discusses these important concepts.

Let us assume that:

k = Number of distributions

n = Number of samples

$\theta = \{\theta_1, \theta_2, \theta_3, \cdots, \theta_k\}$, a set of parameters that are associated with the distributions.

θ_j is the parameter of the j^{th} probability distribution.

Then, $p(x_i \mid \theta_j)$ is the probability of i^{th} object coming from the j^{th} distribution. The probability that j^{th} distribution to be chosen is given by the weight w_j, $1 \le j \le k$. Then,

$$p(x_i \mid \theta) = \sum_{j=1}^{k} w_j p_j(x \mid \theta_j)$$

If all the points are generated randomly, then the entire set of objects can be denoted as:

$$p(\chi \mid \theta) = \prod_{i=1}^{n} p(x_i \mid \theta) = \prod_{i=1}^{n} \sum_{j=1}^{k} w_j p_j(x \mid \theta_j)$$

Every data is assumed to be generated by a distribution. To describe the data point, the corresponding distribution along with its parameters should be known. So, the parameters should be learnt. Since Gaussians are assumed, the probability that data belongs to that distribution should be learnt. This is given as:

$$p(\chi \mid \theta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

The parameters μ and σ should be chosen such that the above equation is maximized. This is known as maximum likelihood principle. This kind of estimation is called maximum likelihood estimation. Often, log of likelihood function is used and maximized.

Since there are no multiple distributions as only Gaussians are assumed, the problem is reduced to choosing parameters only. The objective of the EM algorithm is to maximize the likelihood of observation by selecting proper parameters. So, there is a need to find three parameters weights of the gaussian, mean and variance.

The EM algorithm works in two stages:

1. Expectation step – In this step, probability of each data point generated by K-gaussian function is computed. This is just a soft assignment.
2. Maximization step – In this step, the parameters are updated.

The EM algorithm iteratively optimizes a likelihood function in two steps: the E-step (Expectation) and the M-step (Maximization).

Here's a high-level overview of how the EM algorithm works:

1. Initialization: Start with initial estimates of the model parameters. These initial values can be random or based on some prior knowledge.
2. E-step (Expectation):
    - In this step, you compute the expected values (expectation) of the latent (unobserved) variables given the observed data and the current parameter estimates.
    - This involves calculating the posterior probabilities or likelihoods of the missing data or latent variables.
    - Essentially, you're estimating how likely each possible value of the latent variable is, given the current model parameters.
3. M-step (Maximization):
    - In this step, you update the model parameters to maximize the expected log-likelihood found in the E-step.
    - This involves finding the parameters that make the observed data most likely given the estimated values of the latent variables.
    - The M-step involves solving an optimization problem to find the new parameter values.
4. Iteration:
    - Repeat the E-step and M-step alternately until convergence criteria are met. Common convergence criteria include a maximum number of iterations, a small change in parameter values, or a small change in the likelihood.
5. Termination:
    - Once the EM algorithm converges, you have estimates of the model parameters that maximize the likelihood of the observed data.
6. Result:
    - The final parameter estimates can be used for various purposes, such as clustering, density estimation, or imputing missing data.

The EM algorithm is widely used in various fields, including machine learning, image processing, and bioinformatics.

One of its notable applications is in Gaussian Mixture Models (GMMs), where it's used to estimate the **means and covariances** of Gaussian distributions that are mixed to model complex data distributions.

It's important to note that the EM algorithm can sometimes get stuck in local optima, so the choice of initial parameter values can affect the results. To mitigate this, you may run the algorithm multiple times with different initializations and select the best result.

Algorithm 13.8: Expectation-Maximization

Step 1: Select the parameters randomly.

Step 2: In expectation stage, for each point the condition probability is computed.

Step 3: In maximization stage, the new parameters are computed.

Step 4: Repeat the steps 2–3 till change is minimal within the threshold value or parameters do not change at all.

CLUSTER EVALUATION METHODS

Evaluation of clustering algorithm is a difficult task, as domain knowledge is absent most of the times. SO, clustering algorithms validation is difficult as compared to the validation of classification algorithms.

Evaluation of Clustering

1. Internal
2. External
3. Relative

Cohesion and separation

Cohesion (or compact) measures how close the samples are inside the cluster. This ensures that the clusters are homogeneous. Cohesion is measured as sum of squared errors between the samples and the centroid. The within cluster sum is given as:

$$\sum_{k=1}^{N} \sum_{x_i \in C} (x_i - m_j)^2 \qquad (13.17)$$

Here, N – No. of cluster,

C – set of centroids

X_i – centroid

M_j – samples.

Separation indicates how well a sample differs from other clusters. This is measured as the weighted sum of the differences of the centroid of the dataset and the centroid of the generated clusters. This is given as:

$$\sum_{x_i \in C} C_i (x - x_i)^2 \qquad (13.18)$$

Here, x – centroid of the entire dataset

X_i – centroid of the cluster

C_i – size of the cluster

DUNN Index

This metric measures the ratio between the distance between the clusters and the distance within the clusters. A high Dunn index indicates that the clusters are well-separated and distinct. DUNN index is calculated as:

$$Index = \frac{\alpha \times separation}{\beta \times compactness}$$

Here,α and β are parameters. DUNN index is a useful measure that can combine both cohension and separation.

Silhouette Coefficient

This metric measures how well each data point fits into its assigned cluster and ranges from -1 to 1. A high silhouette coefficient indicates that the data points are well-clustered, while a low coefficient indicates that the data points may be assigned to the wrong cluster.

Silhouette coefficient combines both cohesion and separation. The Silhouette coefficient measures the average distance between clusters. It is given as follows:

$$s_i = \frac{b_i - a_i}{\max\{b_i, a_i\}} \quad (13.20)$$

Here, a_i is the distance between the sample and centroid of the same cluster and b_i is the distance between the sample and the nearest centroid. The silhouette coefficients of the individual objects can be summed to get for the entire cluster as S, given as:

$$S = \frac{1}{N}\sum_{i=1}^{n} s_i \quad (13.21)$$

The value of the silhouette coefficient s_i is between –1 and +1. When it is closer to 1, the clusters are well formed. The value is zero when the data points are between two clusters and negative when the clusters are not formed correctly.

1. Clustering is a technique of partitioning the objects with many attributes into meaningful disjoint subgroups.
2. Quantitative variables use distance measures, Euclidean distance, Manhattan distance and Chebyshev distance.
3. Binary attributes have only two values, 0 or 1. SMC and Jaccard distance are used to measure distance among objects with binary attributes.
4. To calculate the distance between two objects represented by nominal variables, techniques like simple matching, Jaccard method or Hamming distance are used.
5. Cosine similarity is a metric used to measure how similar the documents are irrespective of their size.

--