USN | | | | | | | | | | |

Internal Assessment Test 3 – March 2024

Sub:	Artificial Intelligence and Machine Learning 3					Sub Code:	21CS54	Branch:	ISE
Date:	16/3/24	Duration:	90 Minutes	Max Marks:	50	Sem / Sec:	**5 A,B,C**		**OBE**

	Answer any FIVE FULL Questions		MARKS	CO
1a	Radial basis function neural network Definition Algorithm	 2M 6M	8	CO5
1b	clustering algorithm: Advantages Disadvantages	 1M 1M	2	CO5
2a	Naïve Bayes Algorithm Solution Limitation	 6M 2M	8	CO4
2b	Naïve Bayes algorithm Definition	2M	2	CO5
3a	Definition Euclidean Distance City block Distance Chebyshev Distance	 1M 1M 1M	3	CO5
3b	XOR Boolean function Solution	7M	7	CO4
4a	Activation functions Types	 7M	7	CO5
4b	Maximum A Posteriori (MAP) Hypothesis Maximum Likelihood (ML) Hypothesis	$1_{1/2}$ M $1_{1/2}$ M	3	CO4
5a	MLP Definition Algorithm	 2M 8M	10	CO5
6a	ANN Definition Types of ANN	 2M 3M	5	CO5
6b	self-organizing feature map Iteration: 1 Explanation Algorithm	 2M 3M	5	CO3

1a. Explain the Radial basis function neural network in detail with the algorithm?

10.7 RADIAL BASIS FUNCTION NEURAL NETWORK

Radial Basis Function Neural Network (RBFNN) was introduced by Broomhead and Lowe in 1988. It is a type of Multi Layer Perceptron which has one input layer, one output layer and with strictly one hidden layer. The hidden layer uses a non-linear radial basis function as the activation function, which converts the input parameters into high dimension space which is then fed into the network to linearly separate the problem. An XOR function is not linearly separable and requires at least one hidden layer to classify it. The RBFNN uses the hidden layer to derive the feature vector whose dimension is increased in space. This neural network is useful for interpolation, function approximation, time series prediction, classification and system control.

Typical Radial Basis Functions (RBF) are:

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

$$H(x) = e^{\frac{-(x-c)^2}{r^2}} \tag{10.17}$$

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

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

$$H(x) = \sqrt{\frac{r^2 + (x-c)^2}{r}} \tag{10.18}$$

RBFNN architecture includes:

1. An input layer that feeds the input vector of n-dimension to the network (x_1, x_2, ..., x_n).
2. A hidden layer that comprises 'm' non-linear radial basis function neurons where $m \geq n$. The hidden layer implements the Radial Basis Function called Gaussian function. The output of a hidden layer neuron for an input vector x is given as in Eq. (10.17):

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

where, x is the input vector, c is the centre and r is the radius.

Each RBF neuron in the hidden layer compares the input vector with the centre of the neuron which is a bell curve and outputs a similarity value between 0 and 1. If the input is equal to the neuron centre, then the output is 1 but as the difference increases, the activation value or the output of the neuron falls off exponentially towards 0.

3. An output layer that computes the linear weighted sum of the output of each neuron from the hidden layer neurons:

$$F(x) = \sum_{i=1}^{m} w_i H_i(x) \tag{10.19}$$

where,

w_i is the weight in the link from the Hidden Layer neuron i to the Output Layer.

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

The architecture of a Radial Basis Function Neural Network is shown in Figure 10.12.

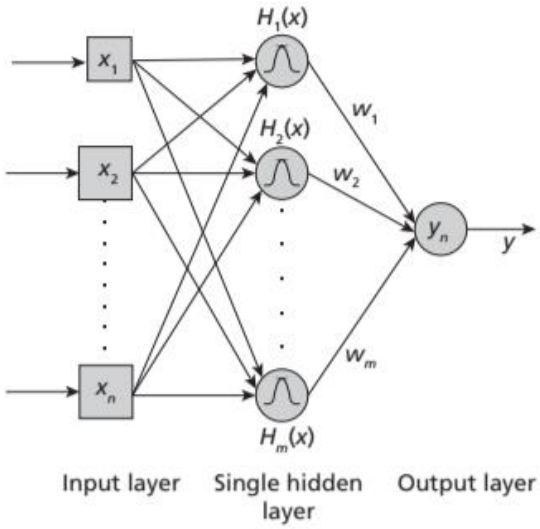


Figure 10.12: Architecture of RBFNN

Training or learning with RBFNN is very fast and the neural network is very good at interpolation.

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.

1b. Write the advantages and disadvantages of the clustering algorithm.

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.

2a. Assess a student's performance using Naïve Bayes Algorithm with the dataset provided in the table. Predict whether the student Gets a Job or not in his Final Year of course.

S.No.	CGPA	Interactiveness	Practical Knowledge	Communication Skills	Job Offer
1.	≥9	Yes	Very good	Good	Yes
2.	≥8	No	Good	Moderate	Yes
3.	≥9	No	Average	Poor	No
4.	<8	No	Average	Good	No
5.	≥8	Yes	Good	Moderate	Yes
6.	≥9	Yes	Good	Moderate	Yes
7.	<8	Yes	Good	Poor	No
8.	≥9	No	Very good	Good	Yes
9.	≥8	Yes	Good	Good	Yes
10.	≥8	Yes	Average	Good	Yes

Test Data = CGPA>=9, Interactiveness=Yes, Practical Knowledge=Average, Communication Skills=Good).

- Solution: The training dataset T consists of 10 data instances with attributes such as 'CGPA', 'Interactiveness', 'Practical Knowledge' and 'Communication Skills' as shown in Table 8.1.
- The target variable is Job Offer which is classified as Yes or No for a candidate student.
- Step 1: Compute the prior probability for the target feature 'Job Offer'. The target feature 'Job Offer' has two classes, 'Yes' and 'No'.
- It is a binary classification problem.
- Given a student instance, we need to classify whether 'Job Offer = Yes' or 'Job Offer = No'.
- From the training dataset, we observe that the frequency or the number of instances with 'Job Offer = Yes' is 7 and 'Job Offer = No' is 3.
- The prior probability for the target feature is calculated by dividing the number of instances belonging to a particular target class by the total number of instances.
- Hence, the prior probability for 'Job Offer = Yes' is 7/10 and 'Job Offer = No' is 3/10 as shown in Table 8.2.

Table 8.2: Frequency Matrix and Prior Probability of Job Offer

Job Offer Classes	No. of Instances	Probability Value
Yes	7	*P* (Job Offer = Yes) = 7/10
No	3	*P* (Job Offer = No) = 3/10

- Step 2: Compute Frequency matrix and Likelihood Probability for each of the feature. Step 2(a): Feature – CGPA Table 8.3 shows the frequency matrix for the feature CGPA.
- Table 8.4 shows how the likelihood probability is calculated for CGPA using conditional probability.

Table 8.4: Likelihood Probability of CGPA

CGPA	*P* (Job Offer = Yes)	*P* (Job Offer = No)
≥9	*P* (CGPA ≥9 | Job Offer = Yes) = 3/7	*P* (CGPA ≥9 | Job Offer = No) = 1/3
≥8	*P* (CGPA ≥8 | Job Offer = Yes) = 4/7	*P* (CGPA ≥8 | Job Offer = No) = 0/3
<8	*P* (CGPA <8 | Job Offer = Yes) = 0/7	*P* (CGPA <8 | Job Offer = No) = 2/3

- As explained earlier the Likelihood probability is stated as the sampling density for the evidence given the hypothesis.
- It is denoted as P (Evidence | Hypothesis), which says how likely is the occurrence of the evidence given the parameters.

- It is calculated as the number of instances of each attribute value and for a given class value divided by the number of instances with that class value.
- For example P (CGPA ≥9 | Job Offer = Yes) denotes the number of instances with 'CGPA ≥9' and 'Job Offer = Yes' divided by the total number of instances with 'Job Offer = Yes'.
- From the Table 8.3 Frequency Matrix of CGPA, number of instances with 'CGPA ≥9' and 'Job Offer = Yes' is 3. The total number of instances with 'Job Offer = Yes' is 7. Hence, P (CGPA ≥9 | Job Offer = Yes) = 3/7.
- Similarly, the Likelihood probability is calculated for all attribute values of feature CGPA.
-

Table 8.3: Frequency Matrix of CGPA

CGPA	Job Offer = Yes	Job Offer = No
≥9	3	1
≥8	4	0
<8	0	2
Total	7	3

- Step 2(b): Feature – Interactiveness Table 8.5 shows the frequency matrix for the feature Interactiveness.

Table 8.5: Frequency Matrix of Interactiveness

Interactiveness	Job Offer = Yes	Job Offer = No
YES	5	1
NO	2	2
Total	7	3

Table 8.6 shows how the likelihood probability is calculated for Interactiveness using conditional probability.

Table 8.6: Likelihood Probability of Interactiveness

Interactiveness	P (Job Offer = Yes)	P (Job Offer = No)
YES	*P* (Interactiveness = Yes | Job Offer = Yes) = 5/7	*P* (Interactiveness = Yes | Job Offer = No) = 1/3
NO	*P* (Interactiveness = No | Job Offer = Yes) = 2/7	*P* (Interactiveness = No | Job Offer = No) = 2/3

Step 2(c): Feature – Practical Knowledge

Table 8.7 shows the frequency matrix for the feature Practical Knowledge.

Table 8.7: Frequency Matrix of Practical Knowledge

Practical Knowledge	Job Offer = Yes	Job Offer = No
Very Good	2	0
Average	1	2
Good	4	1
Total	7	3

Table 8.8 shows how the likelihood probability is calculated for Practical Knowledge using conditional probability.

Table 8.8: Likelihood Probability of Practical Knowledge

Practical Knowledge	*P* (Job Offer = Yes)	*P* (Job Offer = No)
Very Good	*P* (Practical Knowledge = Very Good | Job Offer = Yes) = 2/7	*P* (Practical Knowledge = Very Good | Job Offer = No) = 0/3
Average	*P* (Practical Knowledge = Average | Job Offer = Yes) = 1/7	*P* (Practical Knowledge = Average | Job Offer = No) = 2/3
Good	*P* (Practical Knowledge = Good | Job Offer = Yes) = 4/7	*P* (Practical Knowledge = Good | Job Offer = No) = 1/3

Step 2(d): Feature – Communication Skills

Table 8.9 shows the frequency matrix for the feature Communication Skills.

Table 8.9: Frequency Matrix of Communication Skills

Communication Skills	Job Offer = Yes	Job Offer = No
Good	4	1
Moderate	3	0
Poor	0	2
Total	7	3

Table 8.10: Likelihood Probability of Communication Skills

Communication Skills	*P* (Job Offer = Yes)	*P* (Job Offer = No)
Good	*P* (Communication Skills = Good | Job Offer = Yes) = 4/7	*P* (Communication Skills = Good | Job Offer = No) = 1/3
Moderate	*P* (Communication Skills = Moderate | Job Offer = Yes) = 3/7	*P* (Communication Skills = Moderate | Job Offer = No) = 0/3
Poor	*P* (Communication Skills = Poor | Job Offer = Yes) = 0/7	*P* (Communication Skills = Poor | Job Offer = No) = 2/3

Step 3: Use Bayes theorem Eq. (8.1) to calculate the probability of all hypotheses.

Given the test data = (CGPA ≥9, Interactiveness = Yes, Practical knowledge = Average, Communication Skills = Good), apply the Bayes theorem to classify whether the given student gets a Job offer or not.

P (Job Offer = Yes | Test data) = (*P*(CGPA ≥9 | Job Offer = Yes) *P* (Interactiveness = Yes | Job Offer = Yes) *P* (Practical knowledge = Average | Job Offer = Yes) *P* (Communication Skills = Good | Job Offer = Yes) *P* (Job Offer = Yes)))/(*P* (Test Data))

We can ignore *P* (Test Data) in the denominator since it is common for all cases to be considered.

Hence, *P* (Job Offer = Yes | Test data) = (*P*(CGPA ≥9 |Job Offer = Yes) *P* (Interactiveness = Yes | Job Offer = Yes) *P* (Practical knowledge = Average | Job Offer = Yes) *P* (Communication Skills = Good | Job Offer = Yes) *P* (Job Offer = Yes)

= 3/7 × 5/7 × 1/7 × 4/7 × 7/10

= **0.0175**

Similarly, for the other case 'Job Offer = No',

We compute the probability,

P (Job Offer = No| Test data) = (*P*(CGPA ≥9 |Job Offer =No) *P* (Interactiveness = Yes | Job Offer = No) *P* (Practical knowledge = Average | Job Offer = No) *P* (Communication Skills = Good | Job Offer =No) *P* (Job Offer = No))/(*P*(Test Data)).

P (CGPA ≥9 |Job Offer = No) *P* (Interactiveness = Yes | Job Offer = No) *P* (Practical knowledge = Average | Job Offer = No) *P* (Communication Skills = Good | Job Offer = No) *P* (Job Offer = No)

= 1/3 × 1/3 × 2/3 × 1/3 × 3/10

= **0.0074**

Step 4: Use Maximum A Posteriori (MAP) Hypothesis, h_{MAP} Eq. (8.2) to classify the test object to the hypothesis with the highest probability.

Since *P* (Job Offer = Yes | Test data) has the highest probability value, the test data is classified as '**Job Offer = Yes**'.

2b. Define the Naïve bayes algorithm and write the algorithm?]

- It is a supervised binary class or multi class classification algorithm that works on the principle of Bayes theorem.
- There is a family of Naïve Bayes classifiers based on a common principle.
- These algorithms classify for datasets whose features are independent and each feature is assumed to be given equal weightage.
- It particularly works for a large dataset and is very fast. It is one of the most effective and simple classification algorithms.
- This algorithm considers all features to be independent of each other even though they are individually dependent on the classified object.

Each of the features contributes a probability value independently during classification and hence this algorithm is called as Naïve algorithm.

Algorithm 8.1: Naïve Bayes

1. Compute the prior probability for the target class.
2. Compute Frequency matrix and likelihood Probability for each of the feature.
3. Use Bayes theorem Eq. (8.1) to calculate the probability of all hypotheses.
4. Use Maximum A Posteriori (MAP) Hypothesis, h_{MAP} Eq. (8.2) to classify the test object to the hypothesis with the highest probability.

3a. Define the following with the formula.
i). Euclidean Distance
ii). City block Distance
ii). Chebyshev Distance

Quantitative Variables

Some of the qualitative variables are discussed below.

Euclidean Distance It is one of the most important and common distance measures. It is also called as L_2 norm. It can be defined as the square root of squared differences between the coordinates of a pair of objects.

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

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

The advantage of Euclidean distance is that the distance does not change with the addition of new objects. But the disadvantage is that if the units change, the resulting Euclidean or squared Euclidean changes drastically. Another disadvantage is that as the Euclidean distance involves a square root and a square, the computational complexity is high for implementing the distance for millions or billions of operations involved.

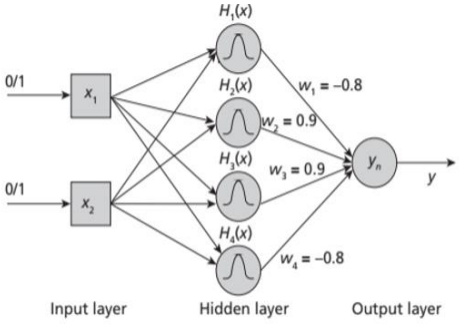
City Block Distance City block distance is known as Manhattan distance. This is also known as boxcar, absolute value distance, Manhattan distance, Taxicab or L_1 norm. The formula for finding the distance is given as follows:

$$\text{Distance}(x_i, x_j) = \sum_{k=1}^{n}\left|x_{ik} - x_{jk}\right| \tag{13.2}$$

Chebyshev Distance Chebyshev distance is 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 L_{max} or L_∞ norm. The formula for computing Chebyshev distance is given as follows:

$$\text{Distance}(x_i, x_j) = \max_k\left|x_{ik} - x_{jk}\right| \tag{13.3}$$

3b. Consider XOR Boolean function that has 4 patterns (0,0) (0,1)(1,0) and (1,1)in a 2 dimensional input space.



Construct a RBFNN as shown in figure that classifies input Pattern:
(0,0)->0
(0,1)->1
(1,0)->1
(1,1)->0

Solution: Define 4 hidden layer neurons with Gaussian RBF:

$$H_1(x) = e^{\frac{-(x-c1)^2}{r^2}}; c1 = (0, 0)$$

$$H_2(x) = e^{\frac{-(x-c2)^2}{r^2}}; c2 = (0, 1)$$

$$H_3(x) = e^{\frac{-(x-c3)^2}{r^2}}; c3 = (1, 0)$$

$$H_4(x) = e^{\frac{-(x-c4)^2}{r^2}}; c4 = (1, 1)$$

For input pattern (0, 0)	**For input pattern (0, 1)**
Distance squared of x from $c1 = (0, 0)$ $= (0-0)^2 + (0-0)^2 = 0$ $H_1(x) = e^{\frac{-(x-c1)^2}{r^2}} = e^{-\frac{0}{2}} = 1.0$	Distance squared of x from $c1 = (0, 0)$ $= (0-0)^2 + (1-0)^2 = 1$ $H_1(x) = e^{\frac{-(x-c2)^2}{r^2}} = e^{-\frac{1}{2}} = 0.6$
Distance squared of x from $c2 = (0, 1)$ $= (0-0)^2 + (0-1)^2 = 1$ $H_2(x) = e^{\frac{-(x-c2)^2}{r^2}} = e^{-\frac{1}{2}} = 0.6$	Distance squared of x from $c2 = (0, 1)$ $= (0-0)^2 + (1-1)^2 = 0$ $H_2(x) = e^{\frac{-(x-c2)^2}{r^2}} = e^{-\frac{1}{2}} = 1.0$
Distance squared of x from $c3 = (1, 0)$ $= (0-1)^2 + (0-0)^2 = 1$ $H_3(x) = e^{\frac{-(x-c3)^2}{r^2}} = e^{-\frac{1}{2}} = 0.6$	Distance squared of x from $c3 = (1, 0)$ $= (0-1)^2 + (1-0)^2 = 2$ $H_3(x) = e^{\frac{-(x-c3)^2}{r^2}} = e^{-\frac{2}{2}} = 0.4$
Distance squared of x from $c4 = (1, 1)$ $= (0-1)^2 + (0-1)^2 = 2$ $H_4(x) = e^{\frac{-(x-c4)^2}{r^2}} = e^{-\frac{2}{2}} = 0.4$	Distance squared of x from $c4 = (1, 1)$ $= (0-1)^2 + (1-1)^2 = 1$ $H_4(x) = e^{\frac{-(x-c4)^2}{r^2}} = e^{-\frac{1}{2}} = 0.6$
$\sum_{j=1}^{m} w_j H_j(x) = -0.8 \times 1.0 + 0.9 \times 0.6 + 0.9 \times 0.6 +$ $-0.8 \times 0.4 = -0.04$	$\sum_{j=1}^{m} w_j H_j(x) = -0.8 \times 0.6 + 0.9 \times 1.0 + 0.9 \times 0.4 +$ $-0.8 \times 0.6 = 0.3$

Forward Phase Calculation

Input		$H_1(x)$	$H_2(x)$	$H_3(x)$	$H_4(x)$	$\sum_{j=1}^{m} w_j H_j(x)$	Output
0	0	1.0	0.6	0.6	0.4	–0.04	0
0	1	0.6	1.0	0.4	0.6	0.3	1
1	0	0.6	0.4	1.0	0.6	0.3	1
1	1	0.4	0.6	0.6	1.0	–0.04	0
		$w_1 = -0.8$	$w_2 = 0.9$	$w_3 = 0.9$	$w_4 = -0.8$		

4a. Explain different types of Activation functions used in ANN.

- Activation functions are mathematical functions associated with each neuron in the neural network that **map input signals to output signals**.
- It decides **whether to fire a neuron or not** based on the input signals the neuron receives.
- These functions normalize the output value of each neuron either between **0 and 1 or between -1 and +1.**
- Typical activation functions can be **linear or non-linear**.
- Linear functions are useful when the input values can be classified into any **one of the two groups** and are generally used in binary perceptrons.
- Non-linear functions, on the other hand, are continuous functions that map the input in the range of **(0, 1) or (-1, 1),** etc.
- These functions are useful in learning high-dimensional data or complex data such as **audio, video and images.**

Below are some of the activation functions used in ANNs:

1. Identity Function or Linear Function

$$f(x) = x \ \forall x \tag{10.4}$$

The value of $f(x)$ increases linearly or proportionally with the value of x. This function is useful when we do not want to apply any threshold. The output would be just the weighted sum of input values. The output value ranges between $-\infty$ and $+\infty$.

2. Binary Step Function

$$f(x) = \begin{cases} 1 & if\ f(x) \geq \theta \\ 0 & if\ f(x) < \theta \end{cases} \tag{10.5}$$

The output value is binary, i.e., 0 or 1 based on the threshold value θ. If value of $f(x)$ is greater than or equal to θ, it outputs 1 or else it outputs 0.

3. Bipolar Step Function

$$f(x) = \begin{cases} 1 & if\ f(x) \geq \theta \\ -1 & if\ f(x) < \theta \end{cases} \tag{10.6}$$

The output value is bipolar, i.e., +1 or −1 based on the threshold value θ. If value of $f(x)$ is greater than or equal to θ, it outputs +1 or else it outputs −1.

4. Sigmoidal Function or Logistic Function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{10.7}$$

It is a widely used non-linear activation function which produces an S-shaped curve and the output values are in the range of 0 and 1. It has a vanishing gradient problem, i.e., no change in the prediction for very low input values and very high input values.

5. Bipolar Sigmoid Function

$$\sigma(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{10.8}$$

It outputs values between −1 and +1.

6. Ramp Functions

$$f(x) = \begin{cases} 1 & if\ x > 1 \\ x & if\ 0 \leq x \leq 1 \\ 0 & if\ x < 0 \end{cases} \tag{10.9}$$

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

7. Tanh – Hyperbolic Tangent Function

The Tanh function is a scaled version of the sigmoid function which is also non-linear. It also suffers from the vanishing gradient problem. The output values range between −1 and 1.

$$\tan h(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{10.10}$$

8. ReLu – Rectified Linear Unit Function

This activation function is a typical function generally used in deep learning neural network models in the hidden layers. It avoids or reduces the vanishing gradient problem. This function outputs a value of 0 for negative input values and works like a linear function if the input values are positive.

$$r(x) = \max(0, x) = \begin{cases} x & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases} \quad (10.11)$$

9. Softmax Function

This is a non-linear function used in the output layer that can handle multiple classes. It calculates the probability of each target class which ranges between 0 and 1. The probability of the input belonging to a particular class is computed by dividing the exponential of the given input value by the sum of the exponential values of all the inputs.

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{k} e^{x_j}} \text{ where } i = 0 \ldots k \quad (10.12)$$

4b. Define Maximum A Posteriori (MAP) Hypothesis, $h_{MAP,}$ and Maximum Likelihood (ML) Hypothesis, $h_{ML.}$

Maximum A Posteriori (MAP) Hypothesis, h_{MAP}

Given a set of candidate hypotheses, the hypothesis which has the maximum value is considered as the *maximum probable hypothesis* or *most probable hypothesis*. This most probable hypothesis is called the Maximum A Posteriori Hypothesis h_{MAP}. Bayes theorem Eq. (8.1) can be used to find the h_{MAP}.

$$\begin{aligned} h_{MAP} &= \max_{heH} P(Hypothesish \mid Evidence\ E) \\ &= \max_{heH} \frac{P(Evidence\ E \mid Hypothesis\ h) P(Hypothesis\ h)}{P(Evidence\ E)} \\ &= \max_{heH} P(Evidence\ E \mid Hypothesis\ h) P(Hypothesis\ h) \end{aligned} \quad (8.2)$$

Maximum Likelihood (ML) Hypothesis, h_{ML}

Given a set of candidate hypotheses, if every hypothesis is equally probable, only $P(E \mid h)$ is used to find the *most probable hypothesis*. The hypothesis that gives the maximum likelihood for $P(E \mid h)$ is called the Maximum Likelihood (ML) Hypothesis, h_{ML}.

$$h_{ML} = \max_{heH} P(Evidence\ E \mid Hypothesis\ h) \quad (8.3)$$

5. Write a definition and algorithm for learning in an MLP?

- This ANN consists of multiple layers with **one input layer, one output layer and one or more hidden layers.** Every neuron in a layer is connected to all neurons in the next layer and thus they are fully connected.
- The information flows in both the directions. In the forward direction, the inputs are multiplied by weights of neurons and forwarded to the activation function of the neuron and output is passed to the next layer.
- If the output is incorrect, then in the backward direction, error is back propagated to adjust the weights and biases to get correct output. Thus, the network learns with the training data.
- A multi-layer perceptron is a type of Feed Forward Neural Network with **multiple neurons arranged in layers**. **All the neurons in a layer are fully connected to the neurons in the next layer.**

- The network has **atleast three layers** with an input layer, one or more hidden layers and an output layer.
- The **input layer** is the visible layer. It just passes the input to the next layer. The layers following the input layer are the hidden layers.
- The **hidden layers** neither directly receive inputs nor send outputs to the external environment.

The final layer is the **output layer** which outputs a single value or a vector of values

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***

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

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_{ik} is the weight in the link from Node j to Node k.

6a. Define ANN & explain different types of ANN with diagrams?

- ANNs consist of multiple neurons arranged in layers.
- There are different types of ANNs that differ by the network structure, activation function involved and the learning rules used.
- In an ANN, there are three layers called input layer, hidden layer and output layer.
- Any general ANN would consist of one input layer, one output layer and zero or more hidden layers.

10.5.1 Feed Forward Neural Network

This is the simplest neural network that consists of neurons which are arranged in layers and the information is propagated only in the forward direction. This model may or may not contain a hidden layer and there is no back propagation. Based on the number of hidden layers they are further classified into single-layered and multi-layered feed forward networks. These ANNs are simple to design and easy to maintain. They are fast but cannot be used for complex learning. They are used for simple classification and simple image processing, etc. The model of a Feed Forward Neural Network is shown in Figure 10.7.

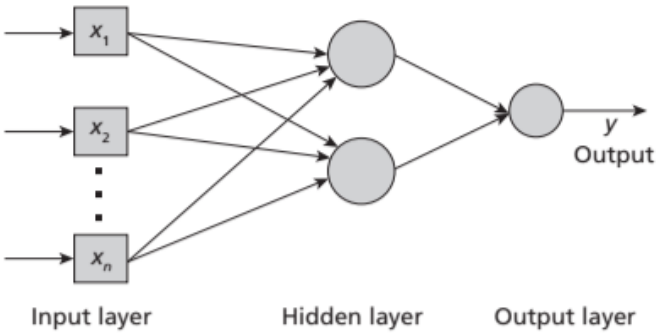


Figure 10.7: Model of a Feed Forward Neural Network

10.5.1 Feed Forward Neural Network

This is the simplest neural network that consists of neurons which are arranged in layers and the information is propagated only in the forward direction. This model may or may not contain a hidden layer and there is no back propagation. Based on the number of hidden layers they are further classified into single-layered and multi-layered feed forward networks. These ANNs are simple to design and easy to maintain. They are fast but cannot be used for complex learning. They are used for simple classification and simple image processing, etc. The model of a Feed Forward Neural Network is shown in Figure 10.7.

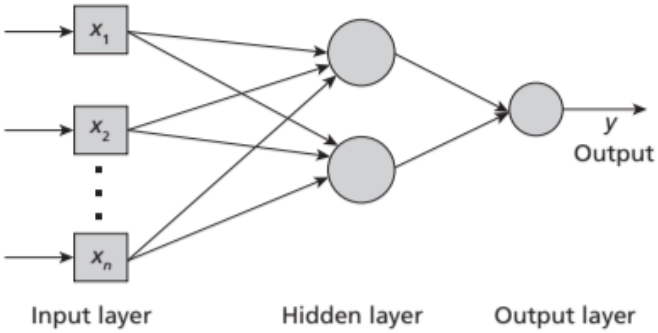


Figure 10.7: Model of a Feed Forward Neural Network

10.5.2 Fully Connected Neural Network

Fully connected neural networks are the ones in which all the neurons in a layer are connected to all other neurons in the next layer. The model of a fully connected neural network is shown in Figure 10.8.

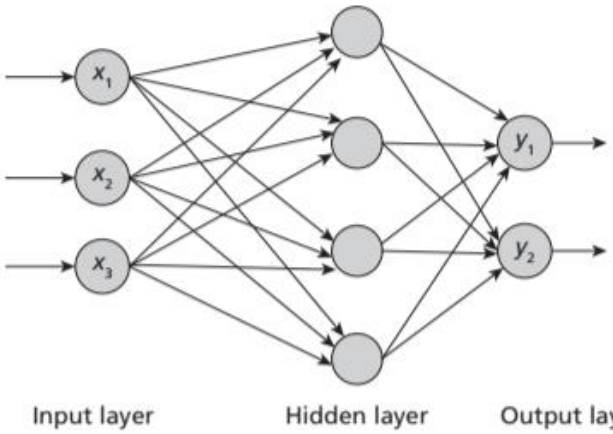


Figure 10.8: Model of a Fully Connected Neural Network

6b. Explain the self-organizing feature map and write the algorithm for the same.

Self-Organizing Feature Map (SOFM) is a special type of Feed Forward Artificial Neural Network developed by Dr Teuvo Kohonen in 1982. Kohonen network is a competitive learning network or also called as adaptive learning network.

SOFM is an unsupervised learning model that clusters data by mapping a high-dimensional data into a two-dimensional map (neurons) or plane.

The model learns to cluster or self organize a high-dimensional data without knowing the class membership of the input data, and hence the name self-organizing nodes.

These self-organizing nodes are also called as feature maps. The mapping is based on the relative distance or similarity between the points and the points that are near to each other in the input space are mapped to nearby output map units in the SOFM.

- Network Architecture and Operations The network architecture consists of only two layers called the Input layer and the Output layer, and there are no Hidden layers.
- The number of units in the Input layer is based on the length of the input samples which is a vector of length ‘n’. Each connection from the Input units in the Input layer to the output units in the Output layer is assigned with random weights.
- There is one weight vector of length ‘n’ associated with each output unit. Output units have intra layer connections with no weights assigned between these connections but used for updating the weights.
- The network architecture of SOFM is shown in Figure 10.14.

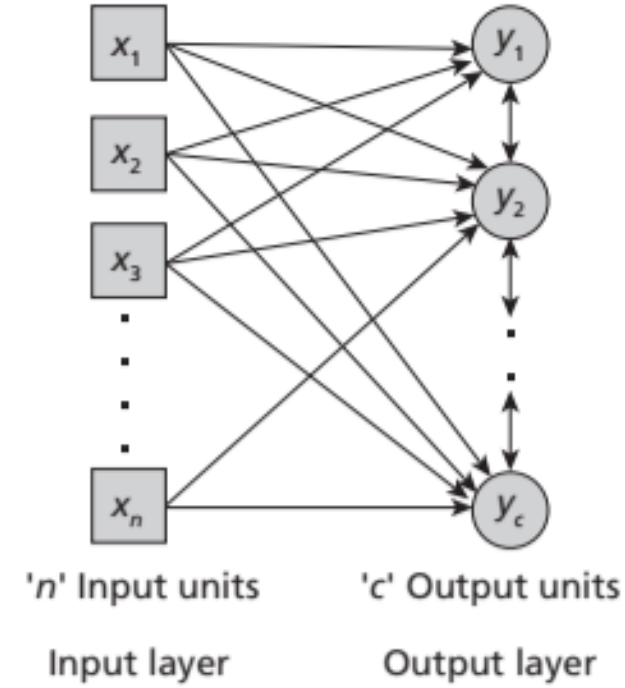


Figure 10.14: Network Architecture of Self Organizing Feature Map

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.