UNIT-5 ARTIFICIAL NEURAL NETS

PRESENTED BY

Dr. G. Ravi & Dr. S. Peerbasha

PG & Research Department of Computer Science

Jamal Mohamed College (Autonomous)

Trichy-20.

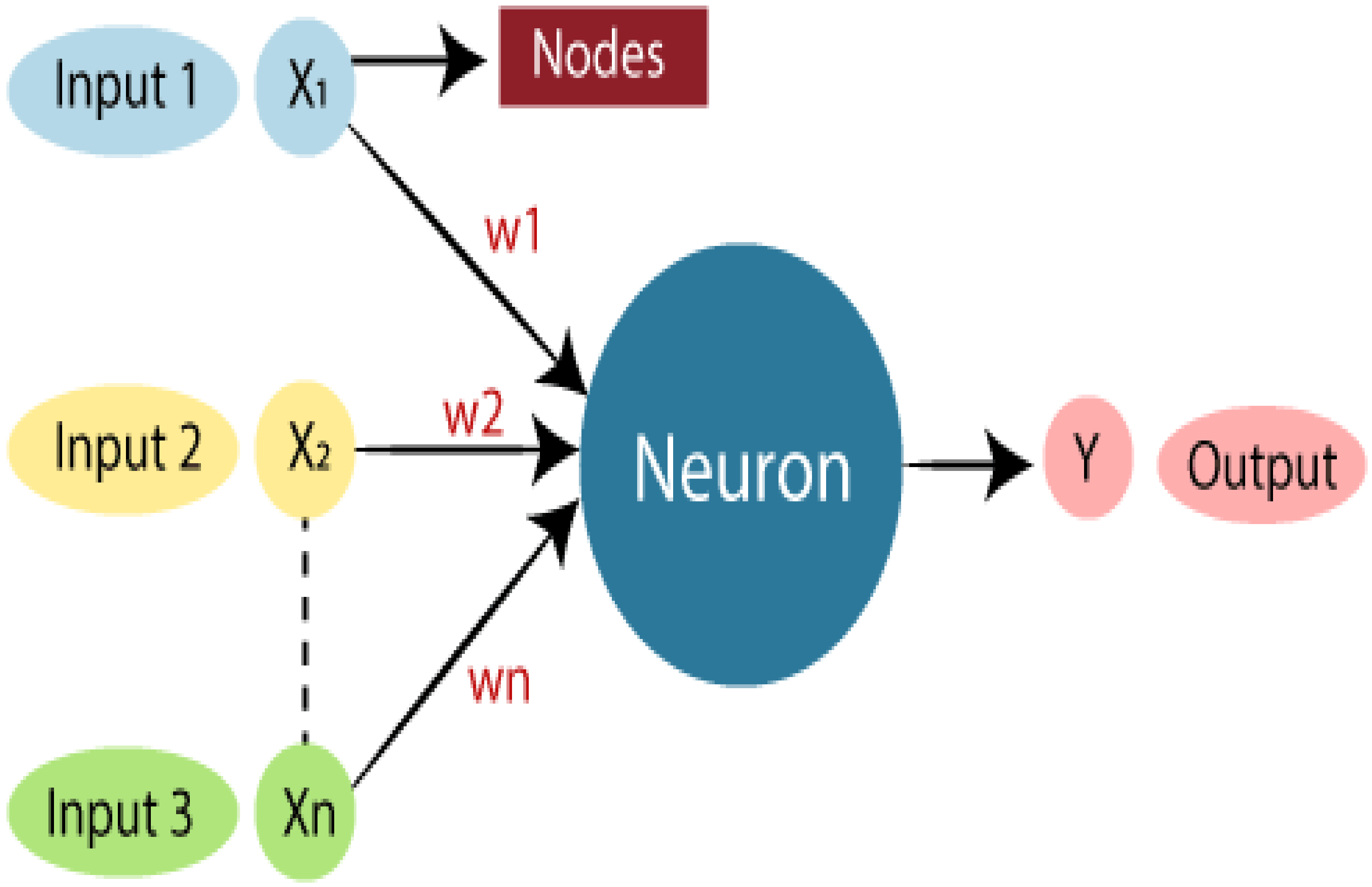
ANN BASICS

The term "**Artificial Neural Network**" is derived from Biological neural networks that develop the structure of a human brain.

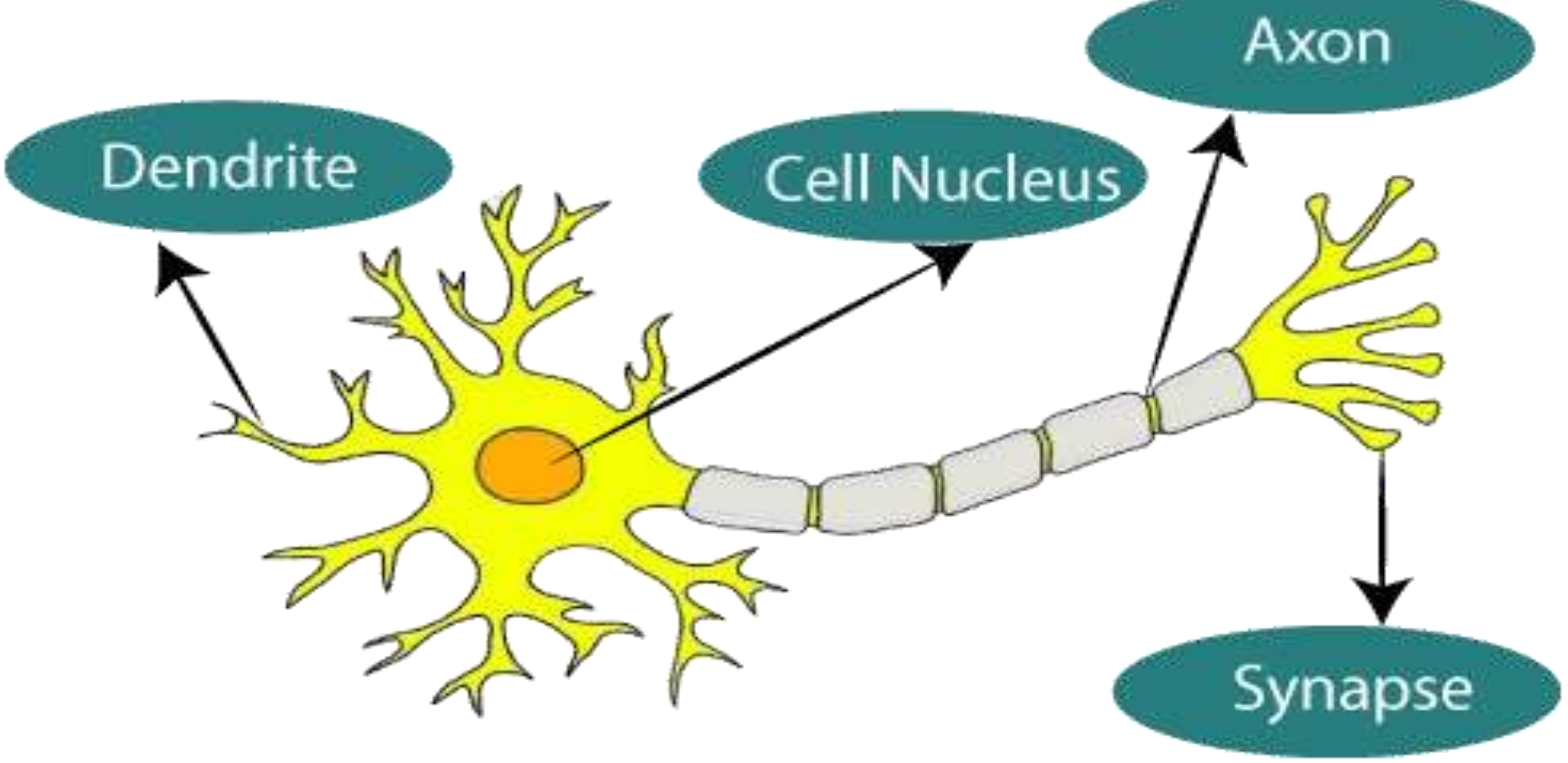
Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks.

These neurons are known as nodes.

ANN BASICS



ANN BASICS



ANN – LEARNING PROCESS

➔The procedure used to perform the learning process is called learning algorithm.

Types of learning procedures used in ANN are:-

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

SUPERVISED LEARNING

➔A popular paradigm of learning of training is called training with a teacher or supervised learning.

➔Here, the network training is done by the input data and is matched with the output patterns.

➔The training pairs are provided by the external supervisor or by the system itself which is having the neural network. It is called as Self-Supervised systems.

UNSUPERVISED LEARNING

➔In Unsupervised learning (Self-Organization), the output layer is trained to respond according to the patterns within the input.

➔In this paradigm, the system is supposed to discover statically salient features of the input population.

REINFORCEMENT LEARNING

➔It is an intermediate form of supervised and unsupervised learning.

➔Here, the learning machine does some action on the environment and gets a feedback response from the environment.

➔Based on the environmental response, the learning systems grades its action good or bad.

TYPES OF NETWORKS

➔The neurons are organized into layers.

➔With every neuron in each layer connected to every other neuron in the next layer.

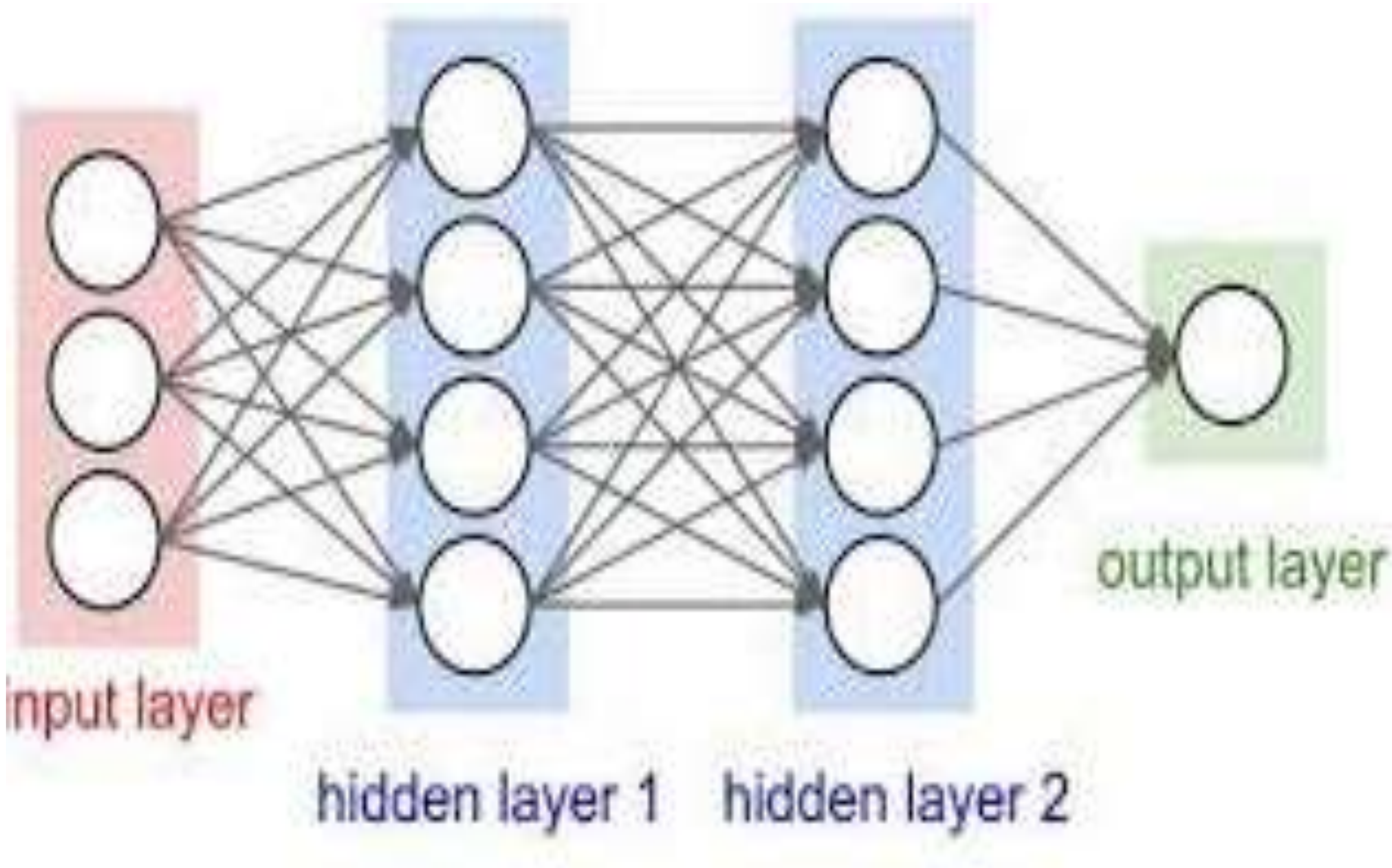
➔It is broadly classified into

1. Fully connected feed-forward network

2. Partially connected recurrent network

3. Fully connected recurrent network

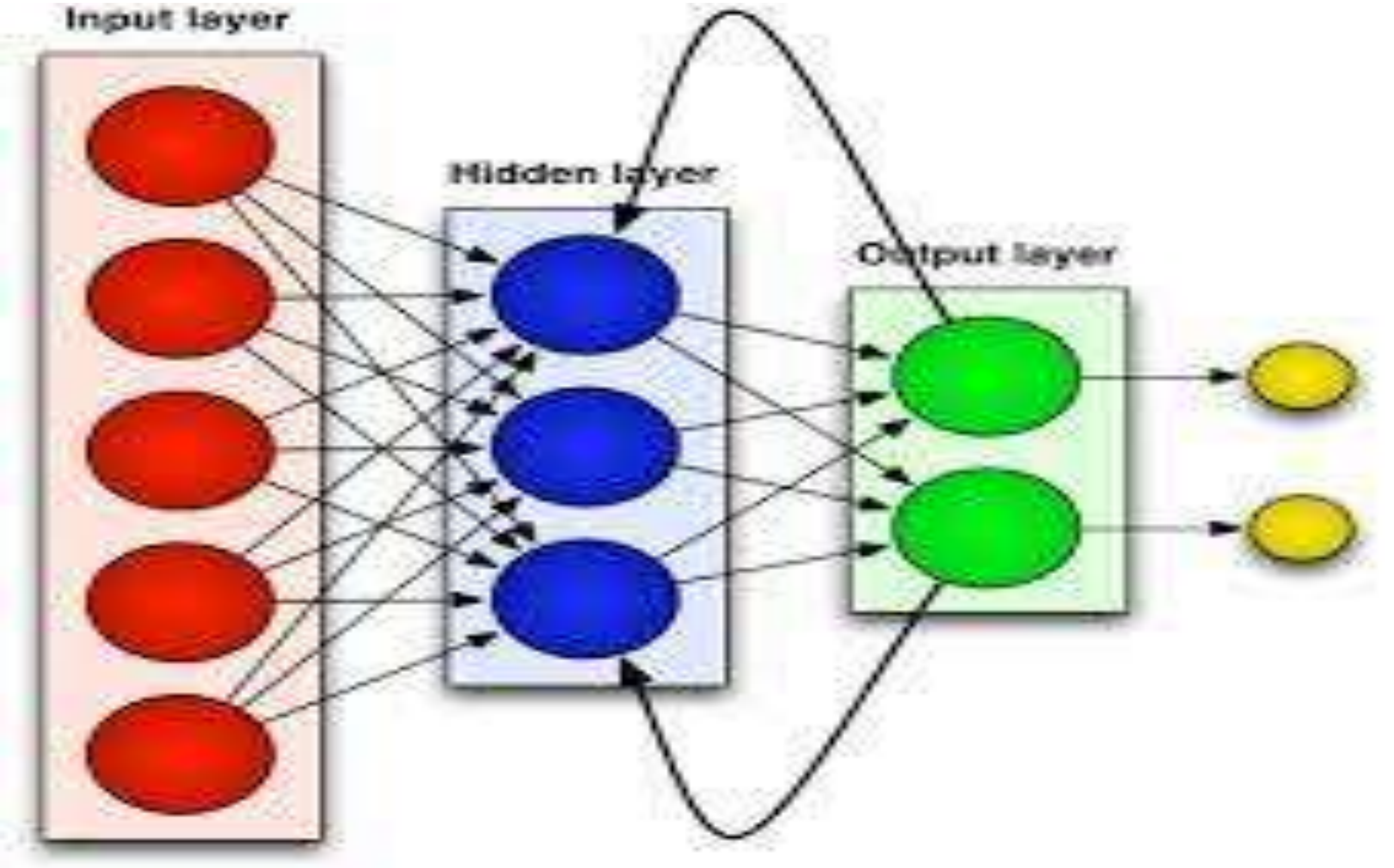
FULLY-CONNECTED FEED-FORWARD NETWORK



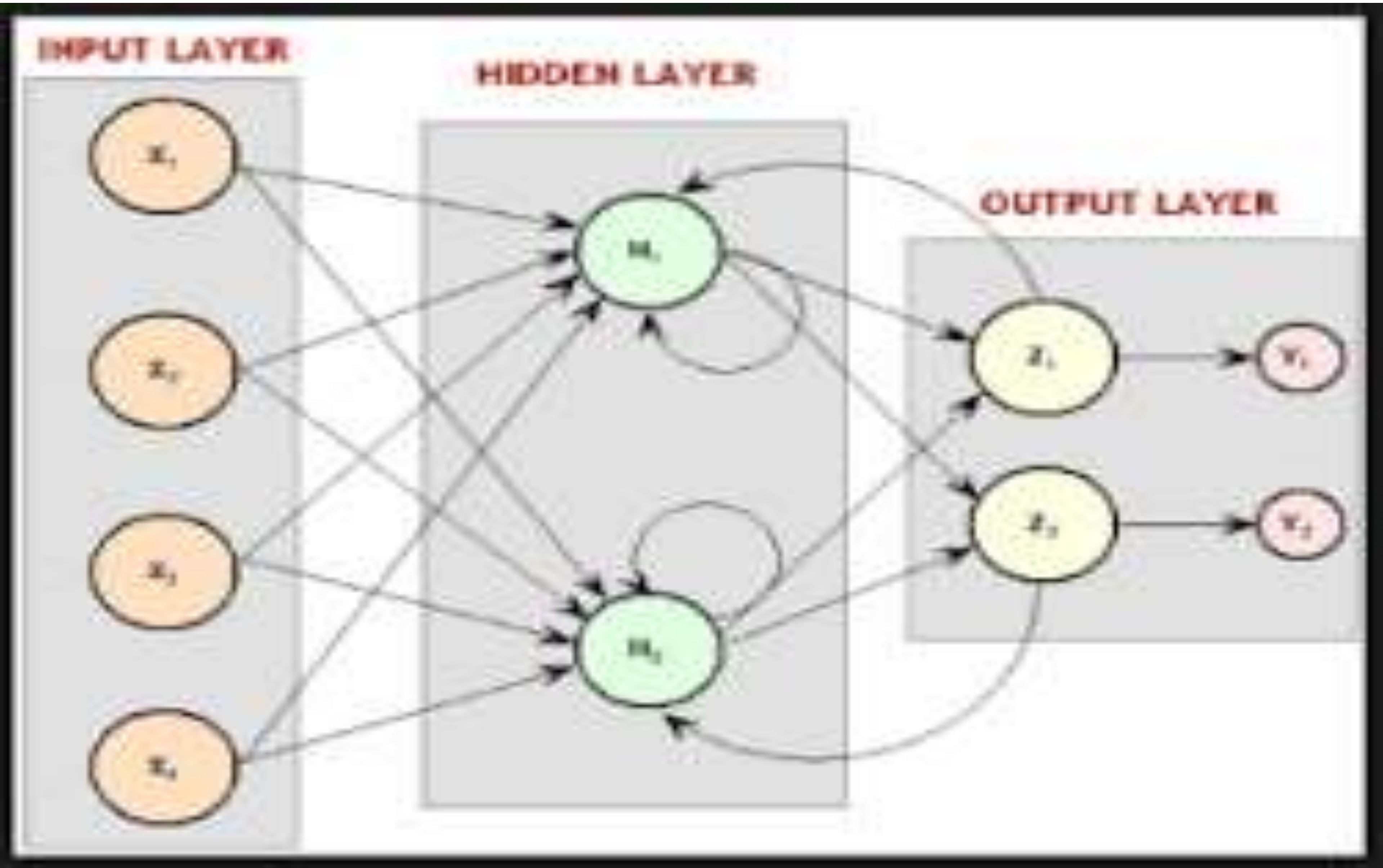
- The data flow from the input to the output units is strictly feed forward.
- The data processing can extend over multiple layers of units but there are no feedback connections extending from the outputs of units to the inputs of units in the same layer or previous layers.
- A multi-layer feed-forward network consists of an input layer, one or more hidden layers and an output layer.
- The neurons in the input layer connect the elements of the input vector or input pattern to the next layer of the network.

- The hidden layers do the processing or computation.
- The activation function of the hidden layers will be non-linear.
- The output of neurons in each layer is given as the input to the next layer.
- The set of outputs from the output layer forms the overall response of the network to the given input pattern.

PARTIALLY RECURRENT NEURAL NETWORK

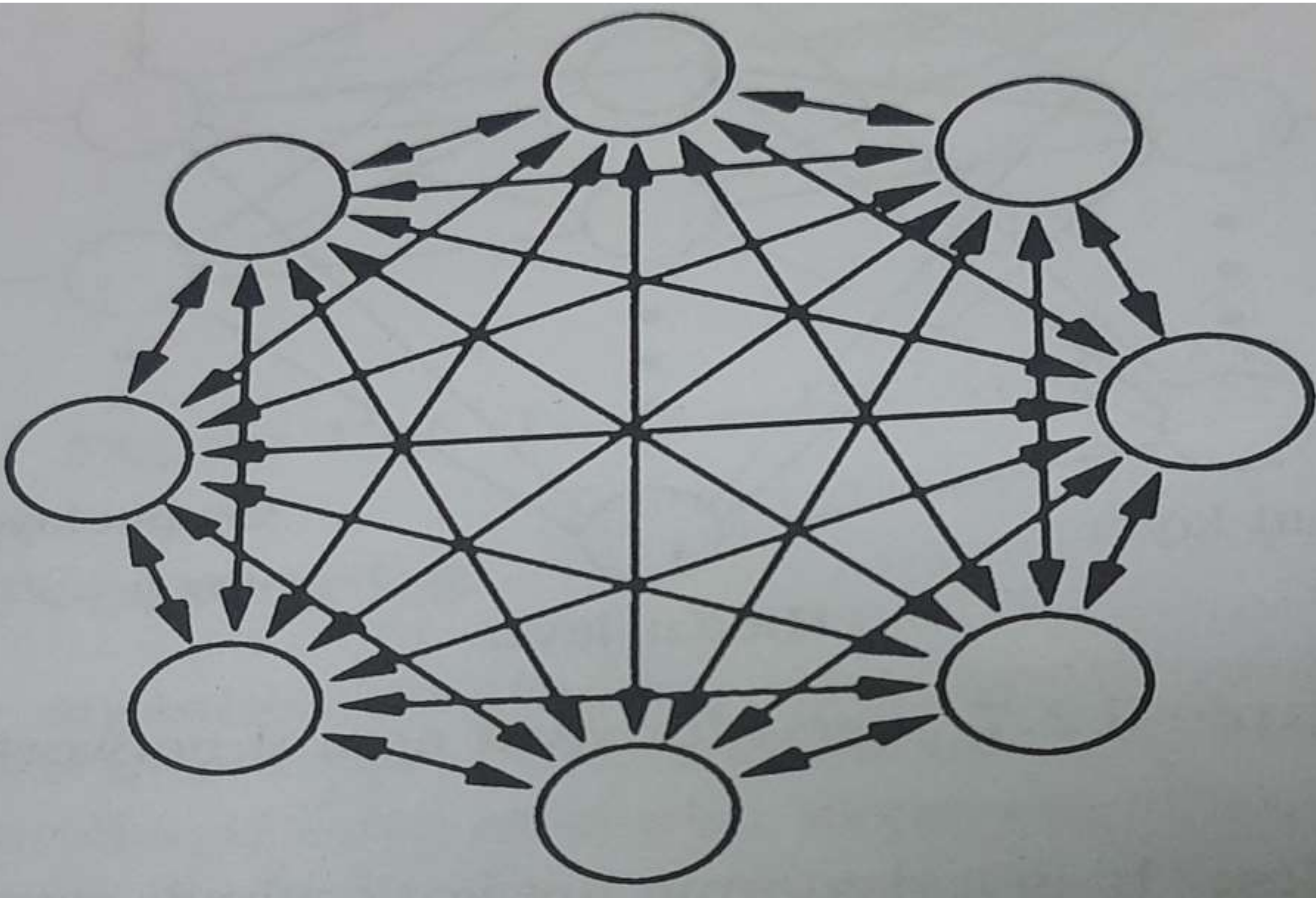


FULLY RECURRENT NEURAL NETWORK



- It is a dynamic network which contains feedback connections.
- Feedback is said to exist in a dynamic system whenever the output of an element in the system influences in part the input applied to that particular element, thereby giving one or more closed paths to the signal transmission.
- Multilayer Perceptron trained with the back propagation algorithm can be used for pattern recognition problems.
- The Hopfield network is a recurrent neural network that has feedback loops from its output to its inputs.

ASSOCIATE NEURAL NETWORK



- There is no hierarchical arrangement in associative networks.
- The connection in this network can be bidirectional.
- In terms of development time and resources, ANN-based solutions are extremely efficient.
- In many different problems, ANN provide performance that is difficult to match with other technologies.
- At present, ANN is an established technology of choice for many applications such as pattern recognition, prediction, system identification and control.

12.4 Perceptron

The perceptron was one of the first processing elements that was capable of learning. The learning used was an iterative supervised learning paradigm. In a supervised adapting scheme, the initial random weights vector w is chosen and the perceptron is given by a randomly chosen data pair and desired output d_1. The perceptron learning algorithm is an error-correction rule that changes the weights proportional to the error $e_1 = d_1 - o_1$ between the actual output o_1 and the desired output d_1. Using a simple rule $w_2 = w_1 + \Delta w_1 = w_1 + \eta(d_1 - o_1)x_1$, the new weights are calculated, where w_1 and w_2 are previous and new weights, respectively. The next data pair is drawn randomly from the dataset and the whole scheme is repeated. η is called the learning rate. Gradually, the error rate is reduced to zero iteratively. The computing scheme of the perceptron is given for an input vector x,

$$u = \sum_{i=1}^{n+1} w_i x_i = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + w_{n+1} x_{n+1} \tag{12.4}$$

which produces an output of +1 if u is positive, otherwise the output is −1.

$$o = \text{sign}(u) = \text{sign}\left(\sum_{i=1}^{n+1} w_i x_i\right) \tag{12.5}$$

sign stands for the signum function
i.e.,

$$o = \text{sign}(u) = \left\{ \begin{array}{rl} +1; & \text{for } u > 0 \\ 0; & \text{for } u = 0 \\ -1; & \text{for } u < 0 \end{array} \right\} \tag{12.6}$$

12.4.1 Multilayer Perceptron

Figure 12.7 represents a general structure of multilayer perceptron, which comprises of multiple layers of perceptron. A multilayer perceptron is a feed-forward neural network. The activation function for the structure shown is sigmoid and can approximate $R^n \rightarrow R^1$ nonlinear mapping (n-dimensional input is mapped into one-dimensional output). In a multilayer perceptron, x_{n+1} will be the constant term equal to 1, called bias. The bias weights vector b can simply be integrated into the hidden layer weights matrix v as its last column.

A multilayer perceptron is a representative of nonlinear basis function expression (approximation)

$$o = f_a(x, w, v) = \sum_{i=1}^{N} w_i \psi_i(x, v_i) \tag{12.7}$$

where $f_a(x, w, v)$ is a set of given functions (usually sigmoid functions) such as the logistic function or tangent hyperbolic, o is the output from a model, and N is the number of hidden layer neurons. The output layer's weight vector w and the hidden layer's weight vector v and free parameters are subjects of learning. Input vector x, bias weights vector b, hidden layer weights matrix v and output weights vector w are as follows:

$$x = [x_1, x_2, \ldots, x_n]^T \tag{12.8}$$

$$v = v_{i,j}; i = 1, 2, \ldots, n; j = 1, 2, \ldots, J \tag{12.9}$$

$$b = [b_1, b_2, \ldots, b_J]^T \tag{12.10}$$

$$w = [w_1, w_2, \ldots, w_J, w_{J+1}]^T \tag{12.11}$$

The output layer neuron may have linear activation functions (for regression-type problems) or sigmoid activation functions (for classification or pattern-recognition tasks).

Back-propagation part

1. Calculate the output layer neuron's error signal δo_{kp}.

$$\delta o_{kp} = (d_{kp} - o_{kp}) f'_{ok}(u_{kp}); \; k = 1, \ldots, K \tag{12.17}$$

2. Calculate the hidden layer neuron's error signal δy_{jp}

$$\delta y_{jp} = f'_{nj}(u_{jp}) \sum_{k=1}^{K} \delta o_{kp} w_{kjp}; \; j = 1, \ldots, J-1 \tag{12.18}$$

3. Calculate the updated output layer weights $w_{kj,p+1}$

$$w_{kj,p+1} = w_{kjp} + \eta \delta o_{kp} \gamma_{jp} \tag{12.19}$$

4. Calculate the updated hidden layer weights $v_{ji,p+1}$

$$v_{ji,p+1} = v_{jip} + \eta \delta y_{jp} x_{ip} \tag{12.20}$$

5. If $p < P$, go to step 3.
6. The learning epoch is completed when $p = P$. Learning is terminated when $E_p < E_{\text{des}}$. Otherwise, go to step 3 and start new learning epoch with $p = 1$.

The practical aspects of back-propagation learning to be considered are the number of hidden layers, the number of neurons in hidden layers, the type of activation function, weight initialisation, the choice of learning rate, the choice of the error stopping function and the momentum term.

Derivation

Error can be detected as

$$E = \frac{1}{2} \sum_{p=1}^{P} \sum_{k=1}^{K} (\delta_{pk} - o_{pk})^2 \tag{12.21}$$

where K is the number of output layer neurons and P is the number of training data pairs.

A feed-forward network that has at least one hidden layer, each layer comprising neurons that receive input from the preceding layer and sending outputs to the neurons in the succeeding layer, is analysed. There is no feedback connection within this layer. Figure 12.7 shows simple multilayer architecture.

The derivation of the learning rule for the weight change Δv_{ij} of any hidden layer neuron is the first-order gradient procedure

$$\Delta v_{i,j} = -\eta \frac{\partial E}{\partial v_{ij}}; j = 1, \ldots, J-1; i = 1, \ldots, I \tag{12.22}$$

The Jth node in Figure 12.7 is the augmented bias term $y_J = \pm 1$ and, no weights go to this 'neuron'. If there is no bias term, then $j = 1, \ldots, J$ and the Jth neuron is a processing unit receiving signal from the preceding layer.

$$\frac{\partial E}{\partial v_{i,j}} = \frac{\partial E \partial u_j}{\partial u_j \partial v_{ji}} \tag{12.23}$$

Now, the weights adjustment from Eq. (12.22) is

$$\Delta v_{i,j} = -\eta \frac{\partial E}{\partial v_{ij}} = \eta \delta y_j x_i; j = 1, \ldots, J-1; i = 1, \ldots, I \tag{12.24}$$

and the error signal term for the hidden layer weight is

$$\delta y_j = -\frac{\partial E}{\partial u_j}; j = 1, \ldots, J-1 \tag{12.25}$$

The derivation of the expression for δy_j was a major breaktthrough in the learning procedure for neural networks. u_j contributes to the error at all output layer neurons (u_k affected only the kth neuron's output and its corresponding error e_k). Applying the chain rule from Eq. (12.25),

$$\delta y_j = \frac{\partial E \partial y_j}{\partial y_j \partial u_j}; j = 1, \ldots, J-1 \tag{12.26}$$

The calculation is relatively straight forward.

$$\frac{\partial y_j}{\partial u_j} = f_j'(u_j) \tag{12.27}$$

Error in equation (12.22) and the first term in Eq. (12.26) can be written as

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j}\left\{\frac{1}{2}\sum_{k=1}^{K}[d_k - f(u_k(y))]\right\}^2 \tag{12.28}$$

i.e.,

$$\frac{\partial E}{\partial y_j} = -\sum_{k=1}^{N}(d_k - o_k)\frac{\partial}{\partial y_j}\{f[u_k(y)]\} \tag{12.29}$$

$$\frac{\partial E}{\partial y_j} = -\sum_{k=1}^{K}\delta o_k w_{kj} \tag{12.30}$$

since

$$u_k = w_{k1}y_1 + w_{k2}y_2 + \ldots + w_{kj}y_j + \ldots + w_{kJ}y_J \tag{12.31}$$

and $(d_k - o_k)f'(u_k)$ is δo_k.

Combining Eqs. (12.26), (12.27) and (12.30), we get

$$\delta y_j = f_j'(u_j)\sum_{k=1}^{N}\delta o_k w_{kj} \tag{12.32}$$

Finally, weight adjustment from Eq. (12.24) is

$$\Delta v_{i,j} = \eta f_j'(u_j)x_i\sum_{k=1}^{K}\delta o_k w_{kj}; j = 1, \ldots, J-1; \; i = 1, \ldots, I \tag{12.33}$$

ANN model development is achieved by a training process. It consists of the adjustment of weight between the nodes in the neural network. Weight adjustment is an iterative process and many a times, the system works by the concept of back-propagation thereby providing firm training.

12.5 RBF Networks

Radial functions are functions with special conditions. Their characteristic feature is that their response decreases or increases monotonically with distance from a central point. The Gaussian function is a typical radial function in the case of a scalar input, given by

$$h(x) = \exp\left(-\frac{(x-c)^2}{r^2}\right) \tag{12.34}$$

where c is its centre and r is its radius. A typical radial basis function (RBF) network is given in Figure 12.10 along with its basis function (Figure 12.11).

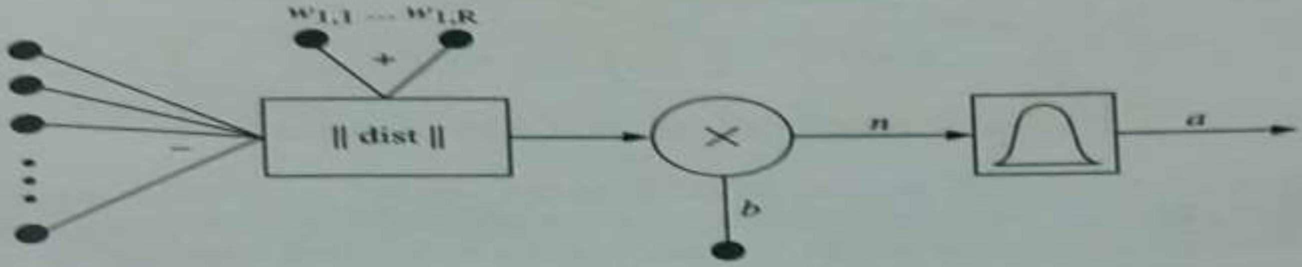


Figure 12.10 Radial basis function.

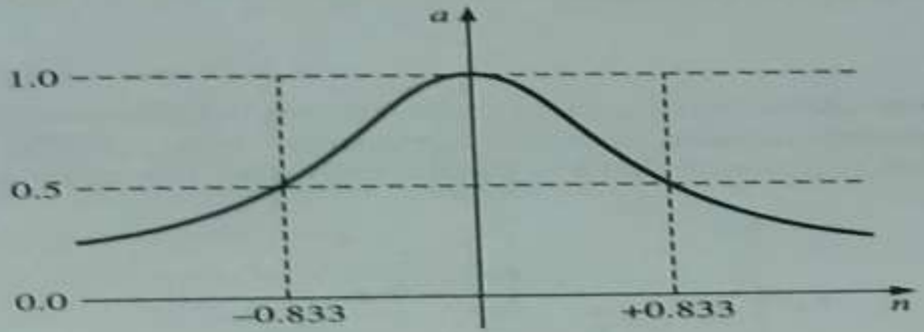


Figure 12.11 Gaussian function in RBF.

Radial basis networks can be designed in a fraction of the time it takes to train standard feed-forward networks. RBF networks work the best when many training vectors are available. This network is like neural net and fuzzy neural net. RBF is a universal approximated network. It can be used in many applications. Each node of RBF consists of an RBF as

$$y = \Sigma_i W_k \Phi(|r - x_i|) \tag{12.35}$$

where $\Phi(x) = e^{-x^2/2\sigma^2}$. An RBF network (Figure 12.12) consists n components of the input vector x which feeds forward to m basis functions whose outputs are linearly combined with weights $\{W_{j=1}^m\}$ into the network output $f(x)$. To train an RBF network, linear algorithm can be used. The training process is very similar to the neural network.

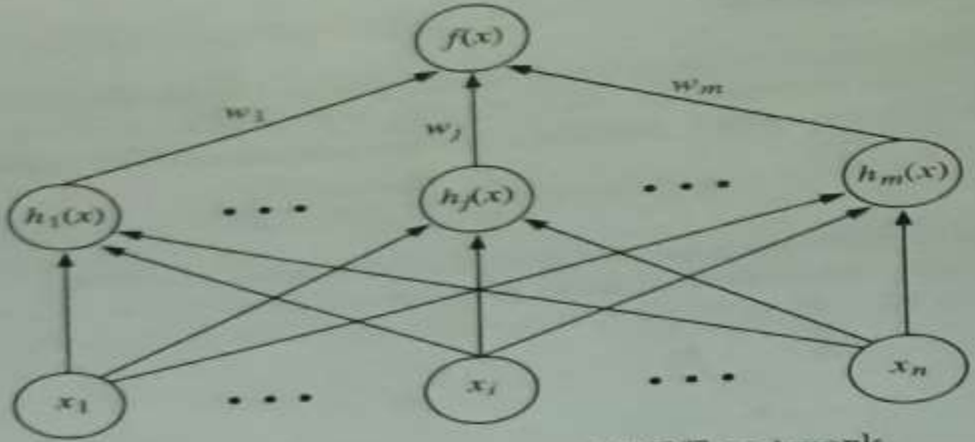


Figure 12.12 Traditional RBF network.

For an application, the design number of RBF nodes needed is typically much less than the number of inputs. This will be a determination of the centres, and the covariance of RBF is a part of the training process. The weight vectors can be calculated directly by matrix inverse operation. So, training is much more efficient than other type of neural nets. Also, RBF nodes are local, which means that only few of them contribute significantly to a certain output; this can produce closed classification regions.

MLPs versus RBFs

Major difference between these two networks is shown in Figure 12.13. In classification problems, MLPs separate classes via hyperplanes but RBFs separate classes via hyperspheres. In learning problems, MLPs use distributed learning and RBFs use localised

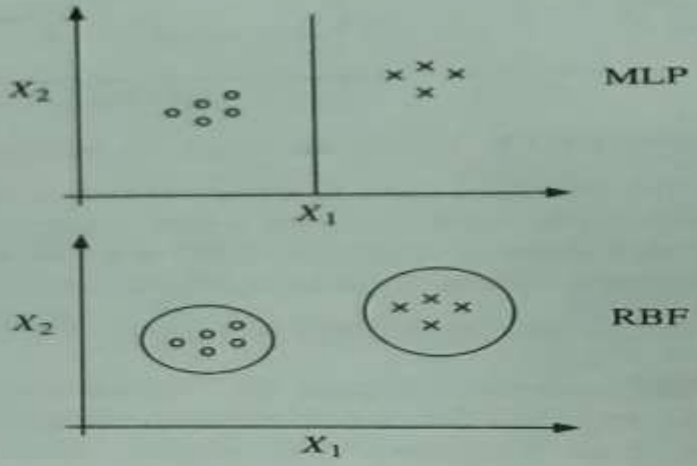


Figure 12.13 MLP versus RBF network.

12.6 ANN Summary

here are many example networks associated with ANNs. There classification is based on upervised and unsupervised learning techniques. Figure 12.14 shows a type of ANN lassification.

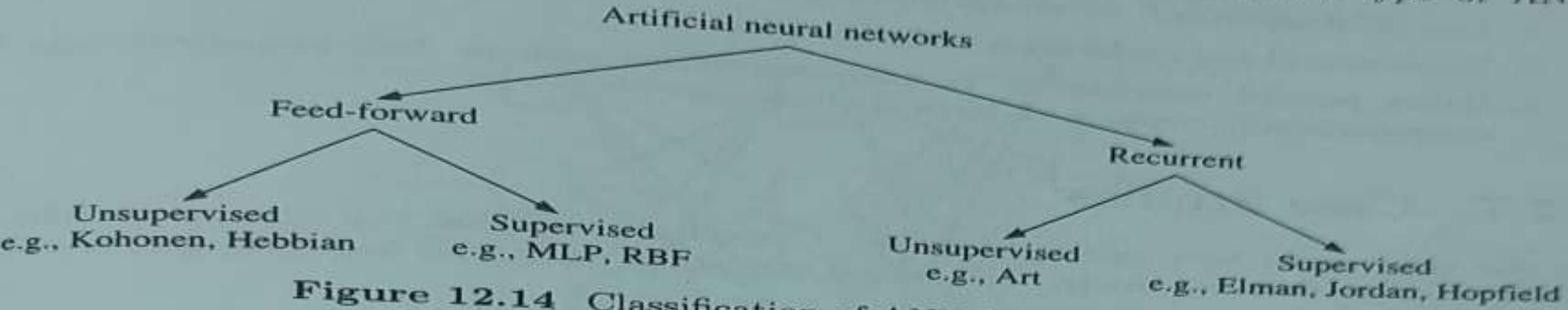


Figure 12.14 Classification of ANN based on learning.

There are 9 steps for ANN design listed as follows:

1. Collect data.
2. Separate into training and test sets.
3. Define a network structure.
4. Select a training algorithm.
5. Set parameters and value. Initialise weights.
6. Transform data to network inputs.
7. Start training and determine and revise weights.
8. Stop and test.
9. Implementation; use the network with new cases.

vantages of neural nets

1. Simple to formulate.
2. Handle mappings of high-dimensional problem space easily.
3. Standardised network model architectures available.
4. Adaptability and learning ability.
5. Generalisation ability.
6. Learning during operation is possible.

Limitations of neural nets

1. Lack of explanation capability.
2. Do not produce an explicit model.
3. Do not perform well on tasks that people do not perform well.
4. Require extensive training and testing of data.
5. No analytical knowledge about the relation between network representation and the problem to be solved is used.
6. Loss of transparency: no interpretation of the trained neural net is possible in general.
7. Experimental and exploration design process.
8. Unless parallel computation is used, algorithms such as back-propagation can be computationally complex for large applications.