Artificial Neural Networks [ETIC-405]

Dr. A K Yadav
Amity School of Engineering and Technology
(affiliated to GGSIPU, Delhi)
akyadav1@amity.edu
akyadav@akyadav.in
www.akyadav.in
+91 9911375598

December 3, 2020

Introduction

- What is Artificial Intelligence?
  The broad discipline of creating intelligent machines
- What is Machine Learning?
  The systems that can learn from experience
- What is Deep Learning?
  The systems that learn from experience on large data sets
- What is Artificial Neural Network?
  The models of human neural networks that are designed to help computers learn
- What is Genetic Algorithms?
  The optimization algorithms that behaves human genetics to improves performance

- What is Intelligent System?
  An intelligent system is a machine with an embedded, Internet-connected computer that has the capacity to gather and analyze data and communicate with other systems. intelligent systems can also include sophisticated AI-based software systems, such as chatbots, expert systems and other types of software.
- What is Expert System?
  In artificial intelligence, an expert system is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning through bodies of knowledge, represented mainly as if–then rules rather than through conventional procedural code.
  With the help of expert systems, we can make system intelligent.

Introduction to Artificial Neural Networks

- Neural Networks characteristics
- History of Artificial Neural Networks
- What is an Artificial Neural Networks?
- Artificial neural terminology
- Model of a neuron
- Topology

Neural Networks characteristics

- It is neurally implemented mathematical model.
- It contains huge number of interconnected processing elements called neurons to do all operations.
- Information stored in the neurons are basically the weighted linkage of neurons

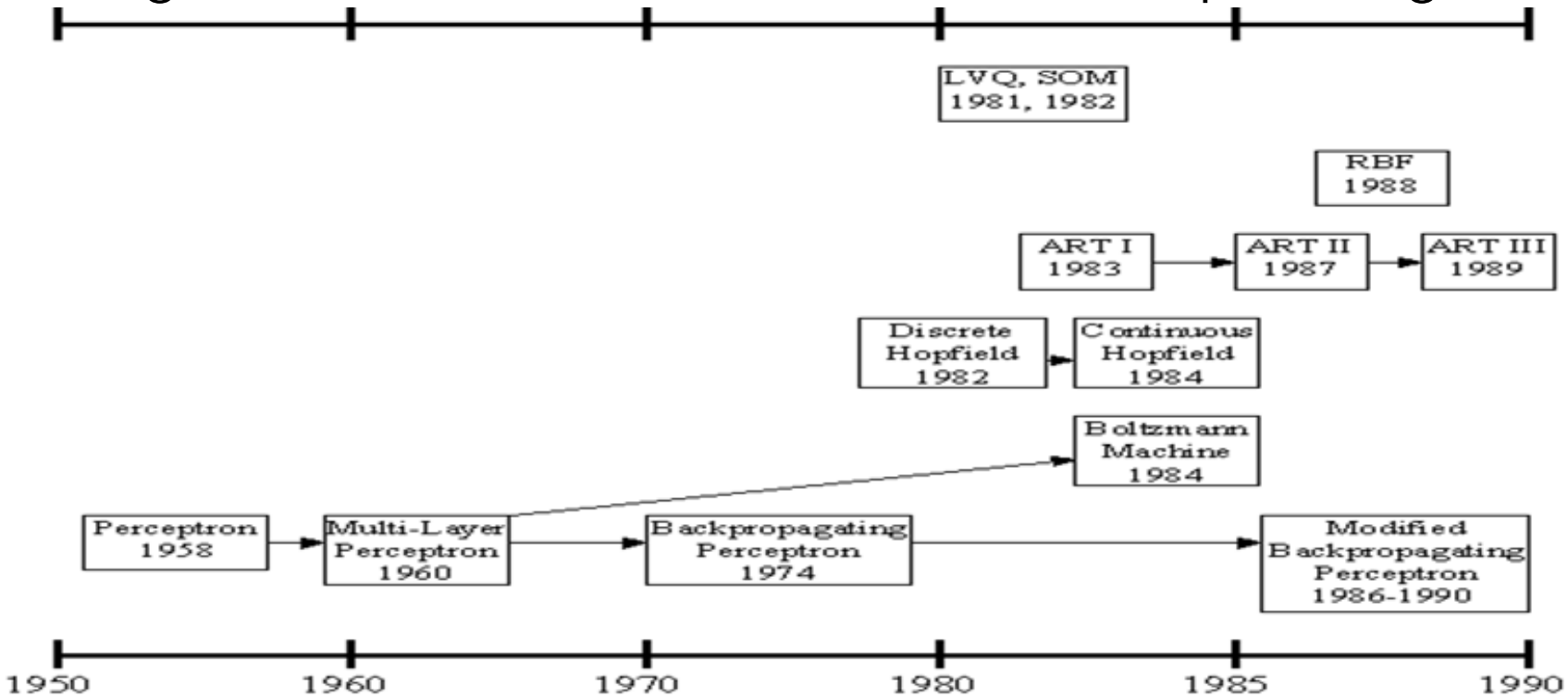
History of Artificial Neural Networks

- History of the ANNs stems from the 1940s, the decade of the first electronic computer.
- However, the first important step took place in 1957 when Rosenblatt introduced the first concrete neural model, the perceptron. Rosenblatt also took part in constructing the first successful neurocomputer, the Mark I Perceptron.
- Rosenblatt's original perceptron model contained only one layer. From this, a multi-layered model was derived in 1960. At first, the use of the multi-layer perceptron (MLP) was complicated by the lack of a appropriate learning algorithm.
- In 1974, Werbos came to introduce a so-called backpropagation algorithm for the three-layered perceptron network.

- In 1986, The application area of the MLP networks remained rather limited until the breakthrough when a general back propagation algorithm for a multi-layered perceptron was introduced by Rummelhart and Mclelland.
- In 1982, Hopfield brought out his idea of a neural network. Unlike the neurons in MLP, the Hopfield network consists of only one layer whose neurons are fully connected with each other.
- Since then, new versions of the Hopfield network have been developed. The Boltzmann machine has been influenced by both the Hopfield network and the MLP.
- In 1988, Radial Basis Function (RBF) networks were first introduced by Broomhead & Lowe. Although the basic idea of RBF was developed 30 years ago under the name method of potential function, the work by Broomhead & Lowe opened a new frontier in the neural network community.

- In 1982, A totally unique kind of network model is the Self-Organizing Map (SOM) introduced by Kohonen. SOM is a certain kind of topological map which organizes itself based on the input patterns that it is trained with. The SOM originated from the LVQ (Learning Vector Quantization) network the underlying idea of which was also Kohonen's in 1972.

- Since then, research on artificial neural networks has remained active, leading to many new network types, as well as hybrid algorithms and hardware for neural information processing.



What is an Artificial Neural Networks?

An artificial neural network consists of a pool of simple processing units which communicate by sending signals to each other over a large number of weighted connections.
A set of major aspects of a parallel distributed model include:

- a set of processing units (cells).
- a state of activation for every unit, which equivalent to the output of the unit.
- connections between the units. Generally each connection is defined by a weight.
- a propagation rule, which determines the effective input of a unit from its external inputs.
- an activation function, which determines the new level of activation based on the effective input and the current activation.

- an external input for each unit.
- a method for information gathering (the learning rule).
- an environment within which the system must operate, providing input signals and if necessary error signals.

Computers vs. Neural Networks

Normal Computers

- one CPU
- fast processing units
- reliable units
- static infrastructure

Neural Networks

- highly parallel processing
- slow processing units
- unreliable units
- dynamic infrastructure

Why Artificial Neural Networks?

There are two basic reasons why we are interested in building artificial neural networks (ANNs):

- **Technical viewpoint:** Some problems such as character recognition or the prediction of future states of a system require massively parallel and adaptive processing
- **Biological viewpoint:** ANNs can be used to replicate and simulate components of the human (or animal) brain, thereby giving us insight into natural information processing.

Tutorial 1

1. How do you define terms Artificial Intelligence, Artificial Neural Network, Machine Learning and Deep Learning in relation to each other?
2. What do you mean by Intelligent System, Expert System?
3. What is Genetic Algorithm and where it is used?
4. What are the differences in the manner of solving problems by human beings and by machines? Illustrate with examples.
5. What is a rule-based expert system? Why do we say such systems are 'brittle'? Discuss your answer with an illustration.
6. Discuss the developments in artificial intelligence that led to the interest in exploring new models for computing.

Artificial neural terminology

- **Processing Unit**
  - We can consider an artificial neural network (ANN) as a highly simplified model of the structure of the biological neural network.
  - It consists of interconnected processing units
  - The general model of a processing unit consists of a summing part followed by an output part.
  - The summing part receives N input values, weights each value, and computes a weighted sum.
  - The weighted sum is called the activation value.
  - The output part produces a signal from the activation value.
  - The sign of the weight for each input determines whether the input is excitatory (positive weight) or inhibitory (negative weight).
  - The inputs could be discrete or continuous data values, and likewise the outputs also could be discrete or continuous.

- The input and output could also be deterministic or stochastic or fuzzy.

- **Interconnections**
  - All processing units are interconnected according to some topology to accomplish a pattern recognition task.
  - The inputs to a processing unit may come from the outputs of other processing units, and/or from external sources.
  - The output of each unit may be given to several units including itself.
  - The value of the output of one unit received by another unit depends on the strength of the connection between the units
  - It is denoted as the weight value associated with the connecting link.
  - If there are N units in a given ANN, then at any instant of time each unit will have a unique activation value and a unique output value.
  - The set of the N activation values of the network defines the activation state of the network at that instant.

- The set of the N output values of the network defines the output state of the network at that instant.
- Depending on the discrete or continuous nature of the activation and output values, the state of the network can be described by a discrete or continuous point in an N-dimensional space.

- **Operations**
  - Each unit of an ANN receives inputs from other connected units and/or from an external source.
  - A weighted sum of the inputs is computed at a given instant of time.
  - The activation value determines the actual output from the output function unit e.g. the output state of the unit.
  - The output values and other external inputs determine the activation and output states of the other units.
  - Activation dynamics determines the activation values of all the units i.e. the activation state of the network as a function of time.

- The activation dynamics also determines the dynamics of the output state of the network.
- The set of all activation states defines the activation state space of the network.
- The set of all output states defines the output state space of the network.
- A network, defined by the units and their interconnections with appropriate weights, the activation states determine the short term memory function of the network.
- Normally the activation dynamics is followed to recall a pattern stored in a network.
- In order to store a pattern in a network, it is necessary to adjust the weights of the connections in the network.
- The set of all weights on all connections in a network form a weight vector.
- The set of all possible weight vectors define the weight space.
- When the weights are changing, then the synaptic dynamics of the network determines the weight vector as a function of time.

- Synaptic dynamics is followed to adjust the weights in order to store the given patterns in the network.
- The process of adjusting the weights is referred to as learning.
- Once the learning process is completed, the final set of weight values is consistent with the long term memory function of the network.
- The procedure to incrementally update each of the weights is called a learning law or learning algorithm.

- **Update**
  - There are several options available for both activation and synaptic dynamics.
  - In one case, the updating of the output states of all the units could be performed synchronously or asynchronously.
  - In synchronously, the activation values of all the units are computed at the same time, assuming a given output state throughout.
  - From the activation values, the new output state of the network is derived.

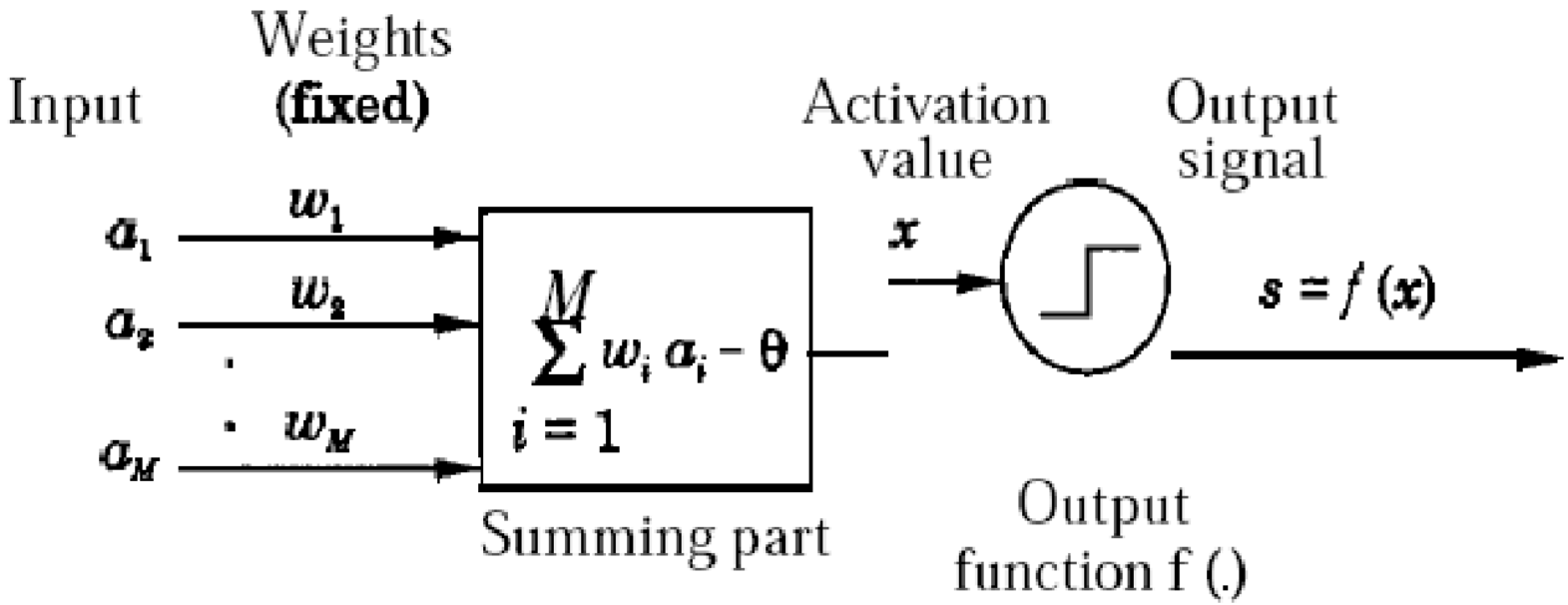
- In an asynchronous update, each unit is updated sequentially, taking the current output state of the network into account each time.
- For each unit, the output state can be determined from the activation value either deterministically or stochastically.
- The ANN models along with the equations governing the activation and synaptic dynamics are designed according to the pattern recognition task to be handled

Model of a neuron

There are three classical models for an artificial neuron or processing unit.

- McCulloch-Pitts Model
- Perceptron
- Adaline

McCulloch-Pitts Model



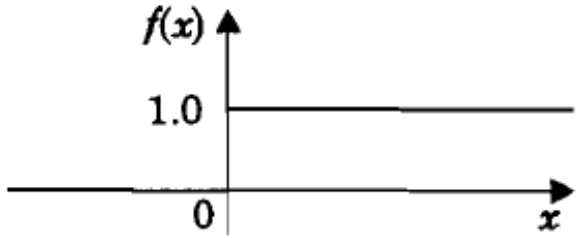
- In McCulloch-Pitts (MP) model the activation x is given by a weighted sum of its M input values a_i and a bias term θ
- The output signal s is typically a nonlinear function $f(x)$ of the activation value x.

- The following equations describe the operation of an MP model:

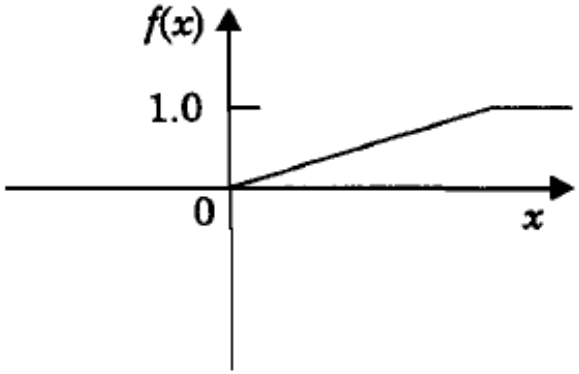
Activation: $x = \sum_{i=1}^{M} w_i a_i - \theta$

Output signal: $s = f(x)$

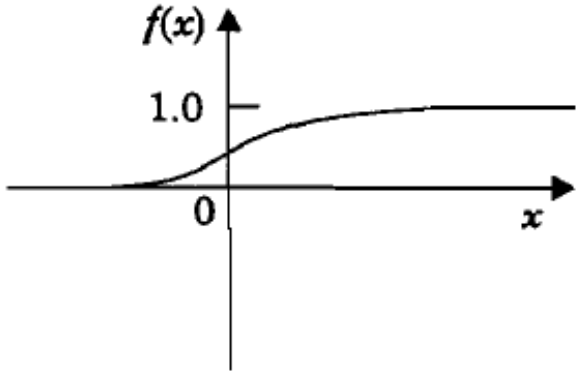
- Three commaply used nonlinear functions (binary, ramp and sigrnoid) are shown in below figure



(a) Binary



(b) Ramp



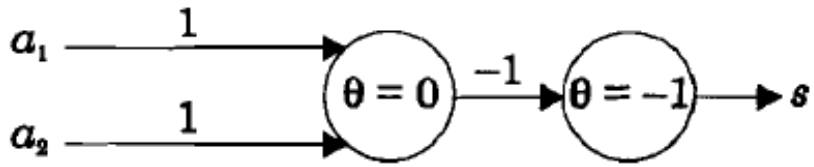
(c) Sigmoid

- Networks consisting of MP neurons with binary (on-off) output signals can be configured to perform several logical functions
- A single input and a single output MP neuron with proper weight and threshold gives an output a unit time later.
- This unit delay property of the MP neuron can be used to build sequential digital circuits.
- With feedback, it is also possible to have a memory cell which can retain the output indefinitely in the absence of any input.
- In the MP model the weights are fixed.
- Hence a network using this model does not have the capability of learning.
- The original model allows only binary output states, operating at discrete time steps

- Example of MP

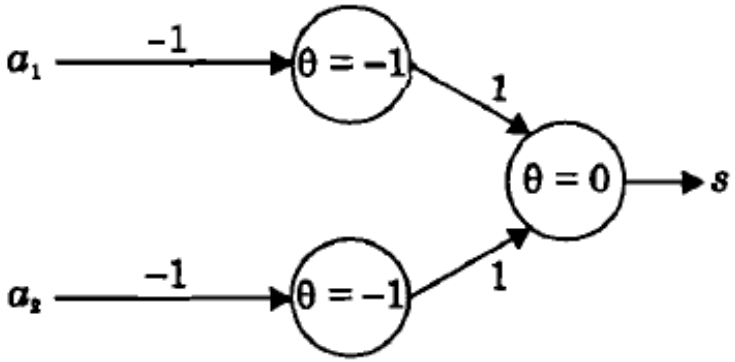
(a) NOR gate

a_1	a_2	s
0	0	1
0	1	0
1	0	0
1	1	0

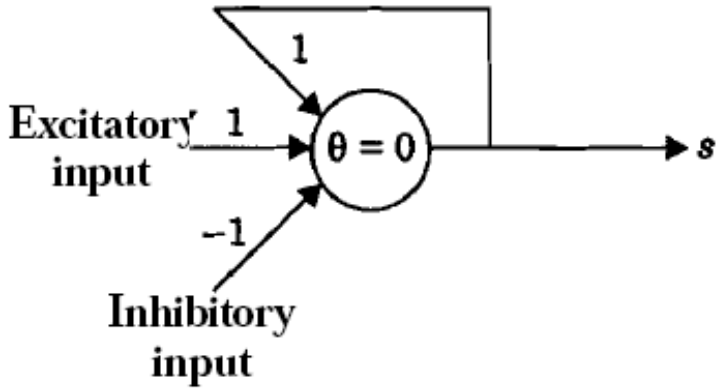


(b) NAND gate

a_1	a_2	s
0	0	1
0	1	1
1	0	1
1	1	0

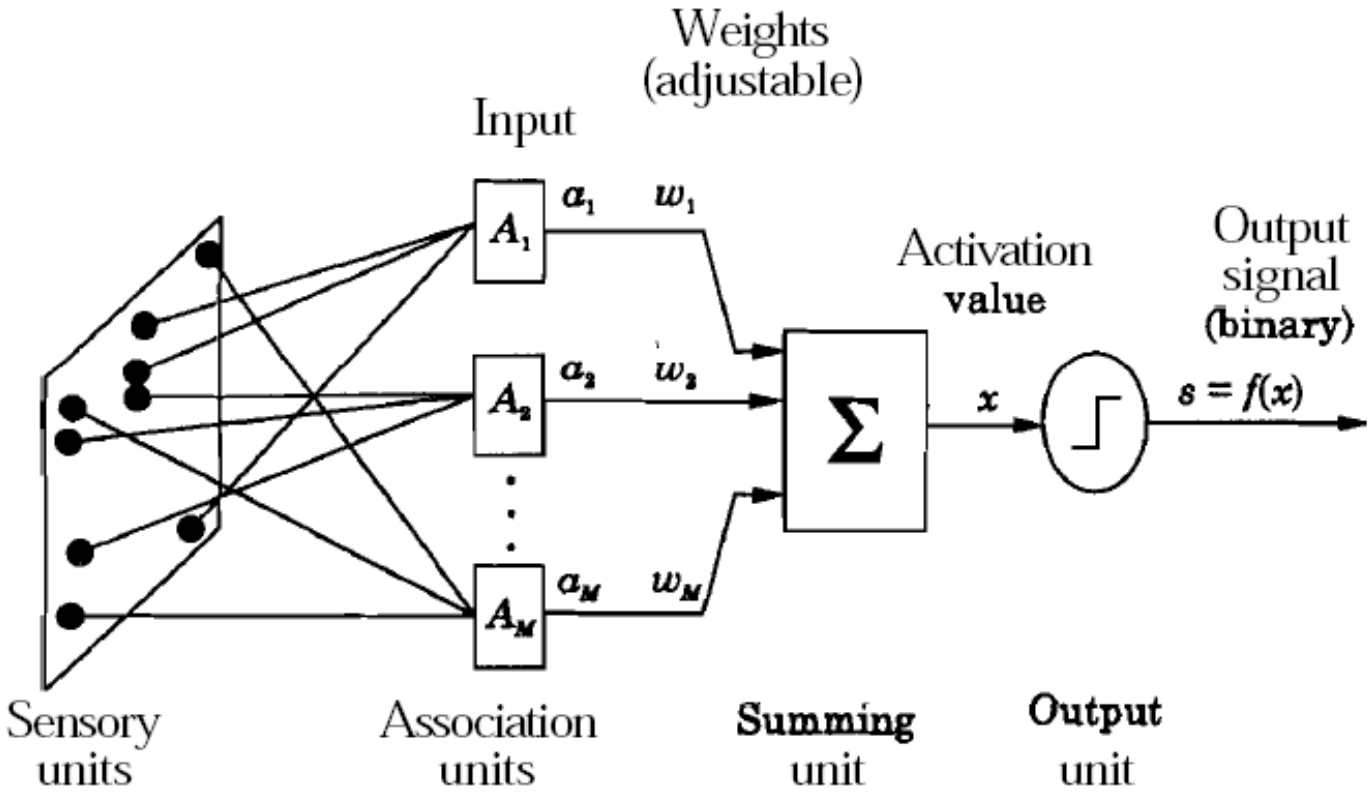


(c) Memory cell assuming unit delay for neuron. An initial excitatory input 1 sustains the output 1 and an initial inhibitory input +1 sustains the output 0



Perceptron

- The Rosenblatt's perceptron model for an artificial neuron consists of outputs from sensory units to a fixed set of association units, the outputs of which are fed to an MP neuron



- The association units perform predetermined manipulations on their inputs.
- The main difference from the MP model is that learning (i.e., adjustment of weights) is incorporated in the operation of the unit.
- The desired or target output b is compared with the actual binary output s, and the error e is used to adjust the weights.
- The following equations describe the operation of the perceptron model of a neuron:

Activation: $x = \sum_{i=1}^{M} w_i a_i - \theta$

Output signal: $s = f(x)$

Error: $\delta = b - s$

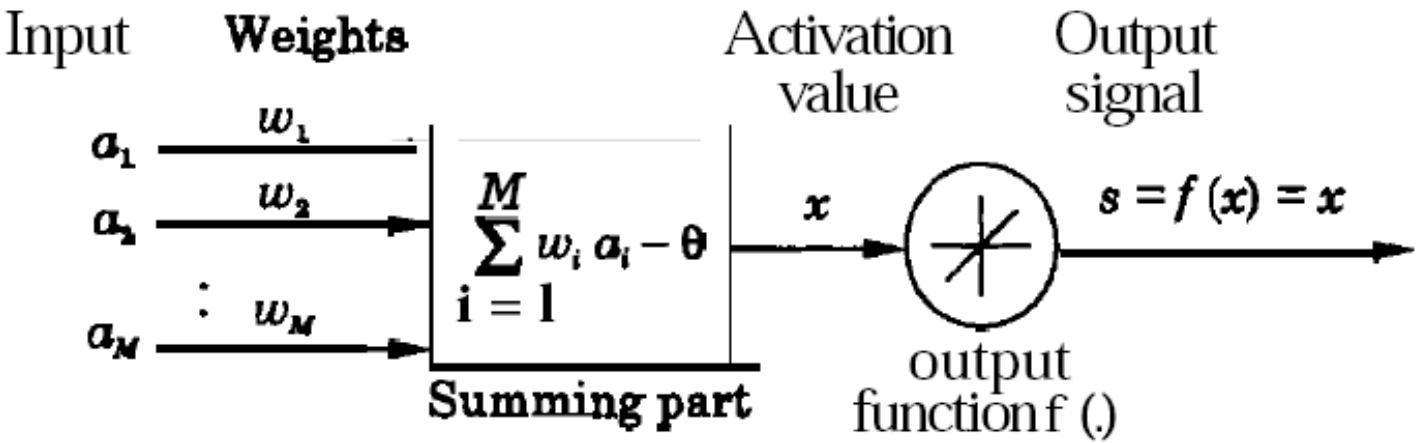
Weight change: $\Delta w = \eta \delta a_i$

where η is the learning rate parameter.

- Perceptron learning law gives a step-by-step procedure for adjusting the weights.
- Whether the weight adjustment converges or not depends on the nature of the desired input-output pairs to be represented by the model.
- The perceptron convergence theorem enables us to determine whether the given pattern pairs are representable or not.
- If the weight values converge, then the corresponding problem is said to be represented by the perceptron network otherwise not.

Adaline

- ADAptive LINear Element (ADALINE) is a computing model proposed by Widrow



- The main difference between the Rosenblatt's perceptron model and the Widrow's Adaline model is that, in the Adaline the analog activation value (x) is compared with the target output (b).
- It means the output is a linear function of the activation value (x)

- The equations that describe the operation of an Adaline are as follows:

  Activation: $x = \sum_{i=1}^{M} w_i a_i - \theta$

  Output signal: $s = f(x) = x$

  Error: $\delta = b - s = b - x$

  Weight change: $\Delta w = \eta \delta a_i$

  where η is the learning rate parameter.
- This weight update rule minimises the mean squared error δ^2, averaged over all inputs.
- It is called Least Mean Squared (LMS) error learning law.
- This law is derived using the negative gradient of the error surface in the weight space.
- It is also known as a gradient descent algorithm.

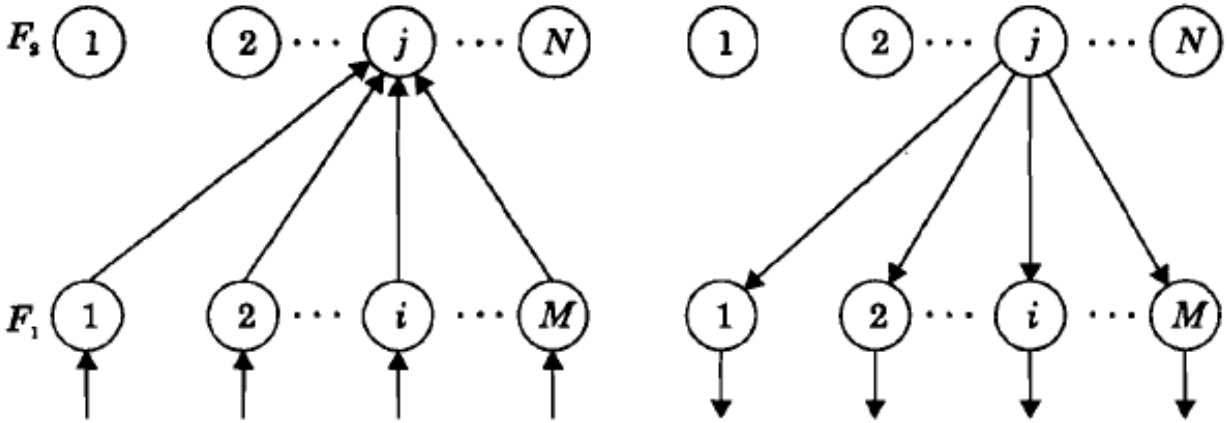
Topology

The arrangement of the processing units, connections, and pattern input/output is referred to as topology

- Topology of a neural network refers to the way the Neurons are connected
- It is an important factor in network functioning and learning.
- Artificial neural networks are useful only when the processing units are organised in a proper way
- ANNs are normally organized into layers of processing units.
- The units of a layer are similar that is they all have the same activation dynamics and output function.
- Connections can be made either from the units of one layer to the units of another layer (interlayer connections)
- or among the units within the layer (intralayer connections) or both interlayer and intralayer connections.

- The connections across the layers and among the units within a layer can be organised either in a feedforward manner or in a feedback manner.
- In a feedback network the same processing unit may be visited more than once.
- The most common topology in supervised learning is the fully connected, three-layer, feedforward network (Backpropagation, Radial Basis Function Networks).
- A common topology in unsupervised learning is a direct mapping of inputs to a collection of units that represents categories e.g. (Self-organizing maps).

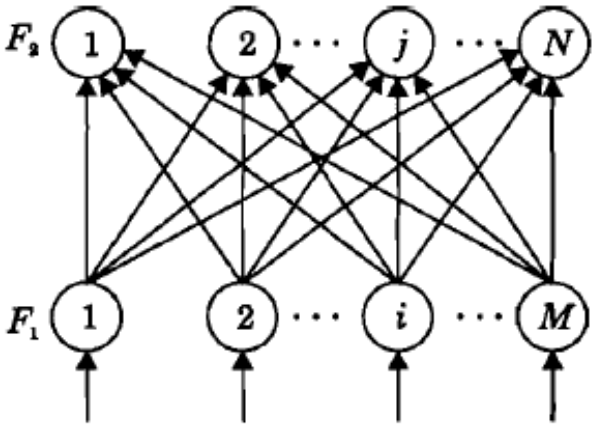
- Different topologies as shown in the following figures.



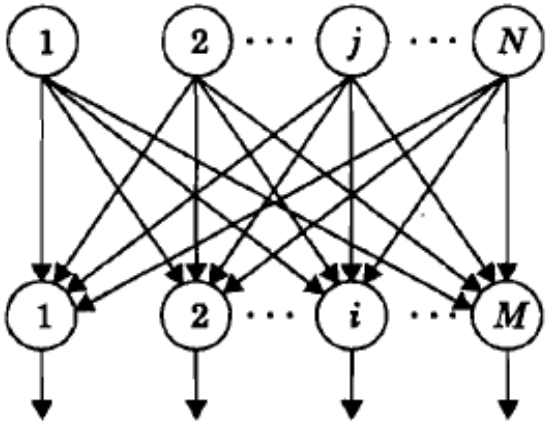
(a) **Instar**

(b) **Outstar**

- Let us consider two layers F_1 and F_2 with M and N processing units as shown in above Figure.
- From all the units in the F_1 layer providing connections to the j^{th} unit in the F_2 layer
- We get two network structures instar and outstar from this configuration.



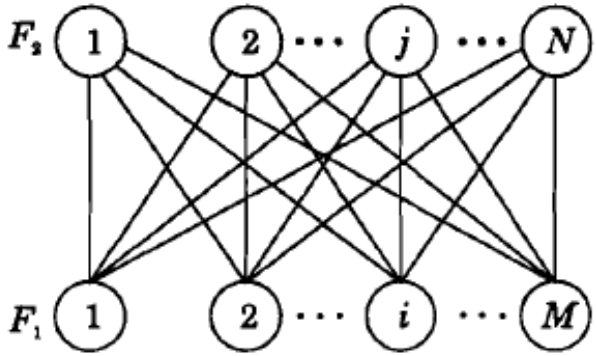
(c) Group of **instars**



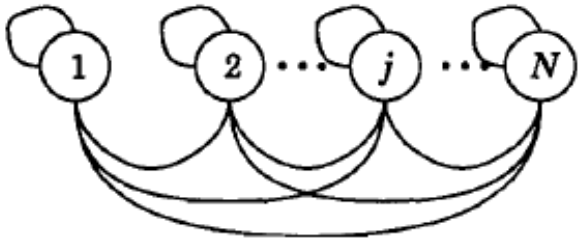
(d) Group of **outstars**

- When all the connections from the units in F_1 to F_2 are made as shown in above figure

- Then network can be viewed as a group of instars if the flow is from F_1 to F_2
- If the flow is from F_2 to F_1, then the network can be viewed as a group of outstars



(e) Bidirectional associative memory



(f) Autoassociative memory

- When the flow is bidirectional as shown in above figure, we get a bidirectional associative memory
- In this case either of the layers can be used as input/output.
- If the two layers F_I and F_2 coincide and the weights are symmetric i.e. $w_{i,j} = w_{j,i}; i \neq j$ then we obtain an autoassociative memory

- In this case each unit is connected to every other unit and to itself

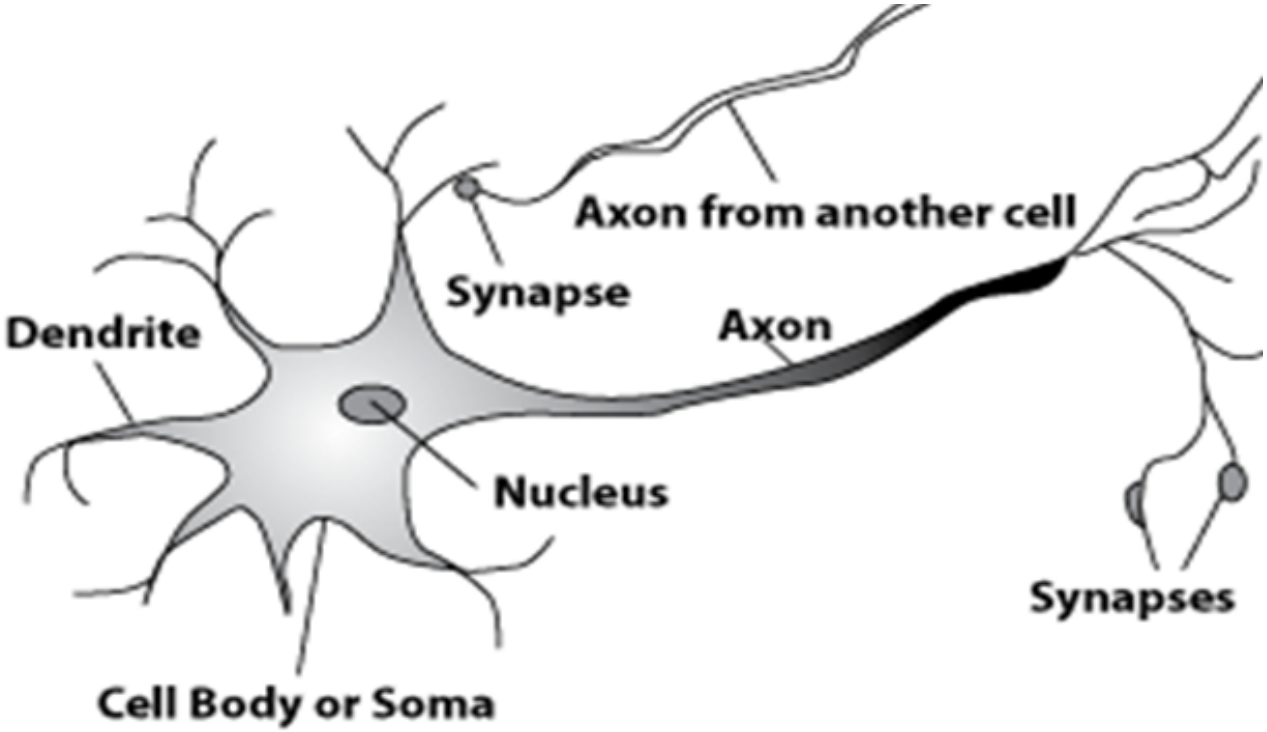
Biological concepts and neuron model

- An artificial neural network (ANN) is either a hardware implementation or a computer program which try to simulate the information processing capabilities of its biological system
- ANNs are typically composed of a great number of interconnected artificial neurons.
- The artificial neurons are simplified models of their biological counterparts.
- ANN is a technique for solving problems by constructing software that works like our brains.

How do our mind works?

- The Brain is a massively parallel information processing system.
- Our brains are a huge network of processing elements.
- A typical brain contains a network of 10 billion neurons.

Model of a human neuron

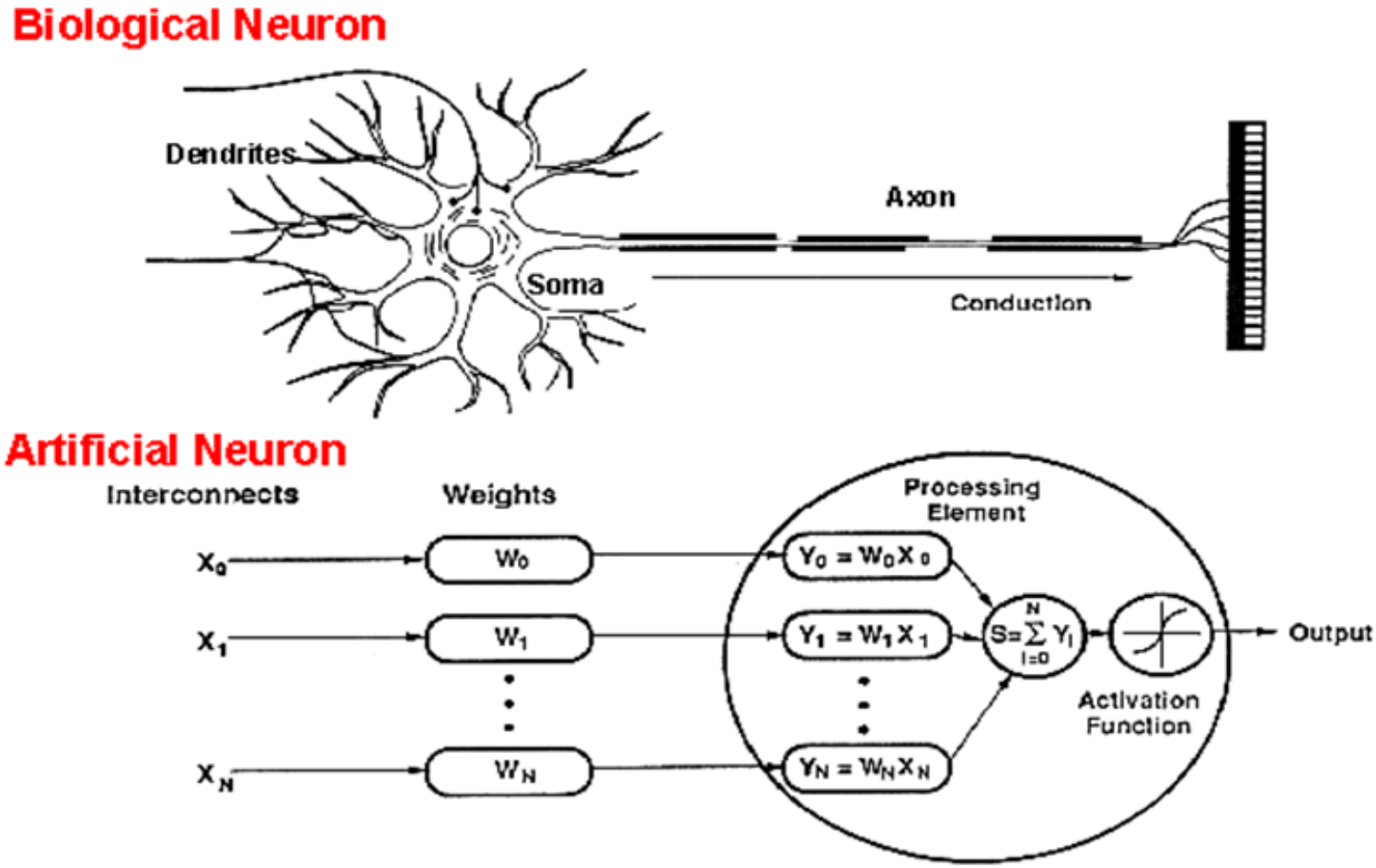


- Input: Dendrites
- Processing Unit: Cell body
- Link/Connections: Synaptic
- Output: Axon

- A neuron is connected to other neurons through about 10,000 synapses
- A neuron receives input from other neurons and all inputs are combined.
- Once input exceeds a critical level, the neuron discharges a spike - an electrical pulse that travels from the body, down the axon, to the next neuron(s)
- The axon endings almost touch the dendrites or cell body of the next neuron.
- Transmission of an electrical signal from one neuron to the next is effected by neurotransmitters.
- Neurotransmitters are chemicals which are released from the first neuron and which bind to the second.
- This link is called a synapse.

- The strength of the signal that reaches the next neuron depends on factors such as the amount of neurotransmitter available.

- An artificial neuron is an imitation of a human neuron as shown in below figure



Tutorial 2

1. Describe some attractive features of the biological neural network that make it superior to the most sophisticated Artificial Intelligence computer system for pattern recognition tasks.
2. Briefly explain the terms cell body, axon, synapse, dendrite, and neuron with reference to a biological neural network.
3. Compare the performance of a computer and that of a biological neural network in terms of speed of processing, size and complexity, storage, fault tolerance and control mechanism.
4. Give two examples of pattern recognition tasks to illustrate the superiority of the biological neural network over a conventional computer system.

5. What are the main differences among the three models of artificial neuron, namely, McCulloch-Pitts, perceptron and adaline?
6. What is meant by topology of artificial neural networks? Give a few basic topological structures of artificial neural networks.

Transfer/ Activation Function

The weighted sums of each node is passed through a function known as an activation function or transfer function. There are five type of transfer functions listed below.

- Unit step: The output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value.

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

- Sigmoid: The sigmoid function consists of 2 functions, logistic and tangential. The values of logistic function range from 0 and 1 and -1 to +1 for tangential function.

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

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

- Piecewise Linear: The output is proportional to the total weighted output.

$$f(x) = \begin{cases} 0 & \text{if } x \leq min \\ ax + c & \text{if } min < x < max \\ 1 & \text{if } max \leq x \end{cases}$$

- Gaussian: Gaussian functions are bell-shaped curves that are continuous. The node output (high/low) is interpreted in terms of class membership (1/0), depending on how close the net input is to a chosen value of average.

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

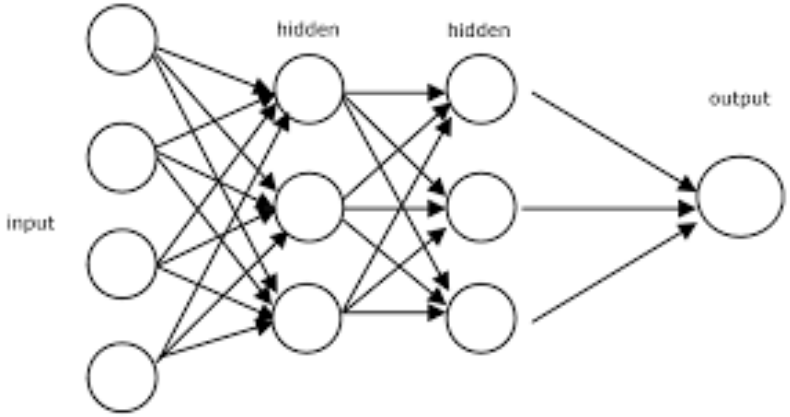
- Linear: Like a linear regression, a linear activation function transforms the weighted sum inputs of the neuron to an output using a linear function.

$$f(x) = ax + c$$

Feed forward network

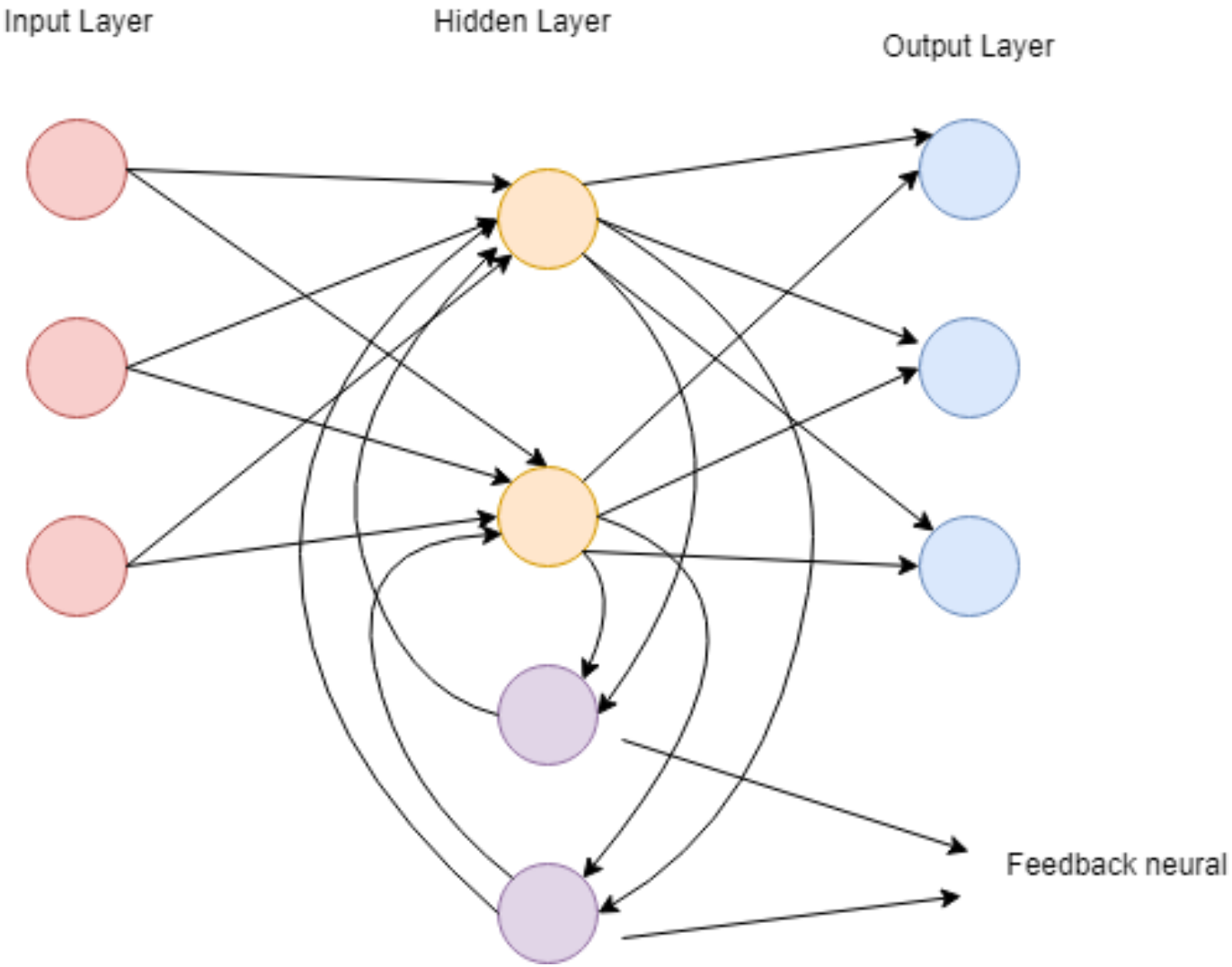
- A feed forward network is a non-recurrent network which contains inputs, outputs, and hidden layers; the signals can only travel in one direction.
- Input data is passed onto a layer of processing elements where it performs calculations.
- Each processing element makes its computation based upon a weighted sum of its inputs.
- The new calculated values then become the new input values that feed the next layer.
- This process continues until it has gone through all the layers and determines the output.
- A threshold transfer function is sometimes used to quantify the output of a neuron in the output layer.

- Feed forward networks include Perceptron (linear and non-linear) and Radial Basis Function networks.
- Feed forward networks are often used in data mining.



Feedback network

- A feedback network has feed-back paths meaning they can have signals travelling in both directions using loops.
- All possible connections between neurons are allowed.
- Since loops are present in this type of network, it becomes a non-linear dynamic system which changes continuously until it reaches a state of equilibrium.
- Feedback networks are often used in associative memories and optimization problems where the network looks for the best arrangement of interconnected factors.


Input Layer
Hidden Layer
Output Layer
Feedback neural

Types of Learning

- Supervised Learning
  - Classification
  - Regression
- Unsupervised Learning
  - Clustering
  - Association
- Reinforcement Learning: feedback wrong result but not how to correct.
- Semi supervised Learning is a combination of supervised and unsupervised learning

Supervised Learning

- Supervised learning is the machine learning task of learning a function that maps an input to an output based on example of input-output pairs.
- It infers a function from labeled training data consisting of a set of training examples.
- Each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal).
- Input and output data are labelled for classification to provide a learning basis for future data processing.
- Chatbots, self-driving cars, facial recognition programs, systems and robots are among the systems that may use either supervised or unsupervised learning.

- Supervised learning systems are mostly associated with retrieval-based AI, but they may also be capable of using a generative learning model.
- In supervised learning for image processing, for example, an AI system might be provided with labelled pictures of vehicles in categories such as cars and trucks.
- After an enough observation, the system should be able to distinguish between and categorize unlabelled images, at which time training can be said to be complete.
- Supervised learning models have some advantages over the unsupervised approach, but they also have limitations.
- If a system with categories for cars and trucks is presented with a bicycle, for example,
- It would have to be incorrectly lumped in one category or the other.

- If the AI system was generative, however, it may not know what the bicycle is but would be able to recognize it as belonging to a separate category.
- A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples.
- An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instance .
- This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way. There are some of the points for learning different Fruits category:-
  - You already learn from your previous work about the physical characters of fruits.
  - So, arranging the same type of fruits at one place is easy now.
  - Your previous work is called as training data in data mining.
  - so, you already learn the things from your train data, this is because of response variable.

- Response variable mean just a decision variable.
- You can observe response variable below (FRUIT NAME) .
- Suppose you have taken a new fruit from the basket then you will see the size , colour and shape of that fruit.
- If size is Big , colour is Red , shape is rounded shape with a depression at the top, you will conform the fruit name as apple, and you will put in apple group.
- Likewise, for other fruits also.
- Job of groping fruits was done and happy ending.
- You can observe in the table that a column was labelled as "FRUIT NAME" this is called as response variable.
- If you learn the thing before from training data and then applying that knowledge to the test data(for new fruit), This type of learning is called as Supervised Learning.
- Classification comes under Supervised learning.

Steps taken to implement supervised algorithm

In order to solve a given problem of supervised learning, one has to perform the following steps:

- Determine the type of training examples. Before doing anything else, the user should decide what kind of data is to be used as a training set. In case of Handwriting Analysis, for example, this might be a single handwritten character, an entire handwritten word, or an entire line of handwriting.
- Gather a training set. The training set needs to be representative of the real-world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, either from human experts or from measurements.

- Determine the input feature representation of the learned function. The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object. The number of features should not be too large, because of the curse of dimensionality; but should contain enough information to accurately predict the output.
- Determine the structure of the learned function and corresponding learning algorithm. For example, the engineer may choose to use support vector machines or decision trees.
- Complete the design. Run the learning algorithm on the gathered training set. Some supervised learning algorithms require the user to determine certain control parameters. These parameters may be adjusted by optimizing performance on a subset (called a validation set) of the training set, or via cross-validation.

- Evaluate the accuracy of the learned function. After parameter adjustment and learning, the performance of the resulting function should be measured on a test set that is separate from the training set

Major issues in supervised learning

- **Bias-variance tradeoff**
  A first issue is the tradeoff between bias and variance. Imagine that we have available several different, but equally good, training data sets. A learning algorithm is biased for a particular input . The prediction error of a learned classifier is related to the sum of the bias and the variance of the learning algorithm. But if the learning algorithm is too flexible, it will fit each training data set differently, and hence have high variance.

- **Function complexity and amount of training data**
  The second issue is the amount of training data available relative to the complexity of the "true" function (classifier or regression function). If the true function is simple, then an "inflexible" learning algorithm with high bias and low variance will be able to learn it from a small amount of data. But if the true function is highly complex (e.g., because it involves complex interactions among many different input features and behaves differently in different parts of the input space), then the function will only be learn able from a very large amount of training data and using a "flexible" learning algorithm with low bias and high variance.

- **Dimensionality of the input space**

  A third issue is the dimensionality of the input space. If the input feature vectors have very high dimension, the learning problem can be difficult even if the true function only depends on a small number of those features. This is because the many "extra" dimensions can confuse the learning algorithm and cause it to have high variance. Hence, high input dimensional typically requires tuning the classifier to have low variance and high bias This is an instance of the more general strategy of dimensionality reduction, which seeks to map the input data into a lower-dimensional space prior to running the supervised learning algorithm.

- **Noise in the output values**

  A fourth issue is the degree of noise in the desired output values (the supervisory target variables). If the desired output values are often incorrect (because of human error or sensor errors), then the learning algorithm should not attempt to find a function that exactly matches the training examples.In such a situation, the part of the target function that cannot be modeled "corrupts" your training data - this phenomenon has been called deterministic noise. When either type of noise is present, it is better to go with a higher bias, lower variance estimator.

Supervised Learning Algorithms

- Support Vector Machines
- Linear regression
- Logistic regression
- Naive Bayes
- Linear discriminant analysis
- Decision trees
- K-Nearest neighbor algorithm
- Neural Networks (Multilayer perceptron)
- Similarity learning

Learning

- The synaptic weights are adjusted to learn the pattern information in the input samples.
- Typically, learning is a slow process, and the samples containing a pattern may have to be presented to the network several times before the pattern information is captured by the weights of the network.
- A large number of samples are normally needed for the network to learn the pattern implicit in the samples.
- Pattern information is distributed across all the weights, and it is difficult to relate the weights directly to the training samples.

- The only way to demonstrate the evidence of learning pattern information is that, given another sample from the same pattern source, the network would classify the new sample into the pattern class of the earlier trained samples.
- Another interesting feature of learning is that the pattern information is slowly acquired by the network from the training samples, and the training samples themselves are never stored in the network.
- That is why we say that we learn from examples, not store the examples themselves.
- The adjustment of the synaptic weights is represented by a set of learning equations, which describe the synaptic dynamics of the network.
- The learning equation describing a synaptic dynamics model is given as an expression for the first derivative of the synaptic weight w_{ij} connecting the unit j to the unit i.

- The set of equations for all the weights in the network determine the trajectory of the weight states in the weight space from a given initial weight state.
- Learning laws refer to the specific manners in which the learning equations are implemented.
- Depending on the synaptic dynamics model and the manner of implementation, several learning laws have been proposed in the literature

Requirements of learning laws

- The learning law should lead to convergence of weights.
- The learning or training time for capturing the pattern information from samples should be as small as possible.
- An on-line learning is preferable to an off-line learning. That is, the weights should be adjusted on presentation of each sample containing the pattern information.
- Learning should use only the local information as far as possible. That is, the change in the weight on a connecting link between two units should depend on the states of these two units only. In such a case, it is possible to implement the learning law in parallel for all the weights, thus speeding up the learning process.

- Learning should be able to capture complex nonlinear mapping between input-output pattern pairs, as well as between adjacent patterns in a temporal sequence of patterns.
- Learning should be able to capture as many patterns as possible into the network. That is, the pattern information storage capacity should be as large as possible for a given network.

Categories of learning

- Learning can be viewed as searching through the weight space in a systematic manner to determine the weight vector that leads to an optimum (minimum or maximum) value of an objective function.
- The search depends on the criterion used for learning.
- There are several criteria which include minimization of mean squared error, relative entropy, maximum likelihood, gradient descent, etc
- There are several learning laws in use, and new laws are being proposed to suit a given application and architecture.
- There are some general categories:
- **Supervised or Unsupervised**

- In supervised learning the weight adjustment is determined based on the deviation of the desired output from the actual output
- Supervised learning may be used for structural learning or for temporal learning
- Unsupervised learning discovers features in a given set of patterns, and organizes the patterns accordingly.
- There is no externally specified desired output in this case.
- Unsupervised learning uses mostly local information to update the weights.
- The local information consists of signal or activation values of the units at either end of the connection for which the weight update is being made

Structural or Temporal

- Structural learning is concerned with capturing in the weights the relationship between the given input-output pattern pairs.
- Temporal learning is concerned with capturing in the weights the relationship between neighbouring patterns in a sequence of patterns.

- **Off-line or On-line**
  - In an off-line learning all the given patterns are used together to determine the weights.
  - On the other hand, in an on-line learning the information in each new pattern is incorporated into the network by incrementally adjusting the weights.
  - An on-line learning allows the neural network to update the information continuously.
  - An off-line learning provides solutions better than an on-line learning since the information is extracted using all the training samples in the case of off-line learning.
- **Deterministic or Stochastic or Fuzzy**
  - The training patterns can be considered as samples of random processes.
  - Accordingly, the activation and output states could also be considered as samples of random processes.
  - Randomness in the output state could also result if the output function is implemented in a probabilistic manner rather than in a deterministic manner.

- Input, activation and output variables may also be viewed as fuzzy quantities instead of crisp quantities

- **Discrete or Continuous**
  - Variables may be discrete or continuous.
  - The update of weight values also may be in discrete steps or in continuous time.

Learning Methods

- There are several methods of learning.
- The learning methods are organized into different groups as below:-

1. Categories of learning
   - Supervised, reinforcement and unsupervised
   - Off-line and on-line
   - Deterministic, stochastic and fuzzy
   - Discrete and continuous
   - Structural or Temporal
2. Hebbian learning
   - Basic Hebbian learning
   - Differential Hebbian learning
   - Stochastic versions
3. Competitive learning-learning without a teacher
   - Linear competitive learning

- Differential competitive learning
- Linear differential competitive learning
- Stochastic versions

4. Error correction learning-learning with a teacher
   - Perceptron learning
   - Delta learning
   - Least Mean Square (LMS) learning
5. Reinforcement learning-learning with a critic
   - Fixed credit assignment
   - Probabilistic credit assignment
   - Temporal credit assignment
6. Stochastic learning
   - In multilayer perceptron
   - In Roltzmann machine
7. Other learning methods
   - Sparse coding
   - Min-max learning
   - Principal component learning
   - Drive-reinforcement learning

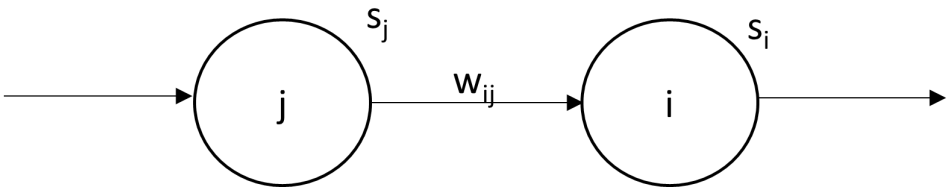
Tutorial 3

1. What are the requirements of learning laws for effective implementation?
2. What are the different categories of learning?
3. What are the different types of Hebbian learning?
4. What are the different types of competitive learning?
5. What is the difference between supervised and unsupervised learning? where are they used?
6. What is the difference between on-line and off-line learning? Which one provides the better solutions and why?
7. What are the major issues in supervised learning?
8. What are the steps to be taken to implement supervised algorithm?

9. What is transfer function? Discusses different types of transfer function.
10. Explain feed forward and feedback networks with diagram.

Hebbian Learning

- The basis for the class of Hebbian learning is the changes in the synaptic strength is proportional to the correlation between the firing of the post- and pre-synaptic neurons



- The synaptic dynamics equation is given by a decay term $-w_{ij}(t)$ and a correlation term $(s_i s_j)$ as

$$\dot{w}_{ij}(t) = -w_{ij}(t) + s_i s_j \tag{1}$$

where $s_i s_j$ is the product of the post-synaptic and pre-synaptic neuronal variables for the i^{th} unit.

- These variables could be
  activation values $s_i s_j = x_i(t)x_j(t)$,
  or an activation value and an external input $s_i s_j = x_i(t)a_j(t)$,
  or an output signal and an external input $s_i s_j = f_i(x_i(t))a_j(t)$,
  or output signals from two units $s_i s_j = f_i(x_i(t))f_j(x_j(t))$,
  or some other parameters related to the post-synaptic and pre-synaptic activity.
- If s_i and s_j represent variables which are deviations from their respective mean values $(\bar{s_i}, \bar{s_j})$, then the resulting correlation term $(s_i - \bar{s_i})(s_j - \bar{s_j})$ is called covariance correlation term.
- We will assume that $s_i = s_i(t) = f_i(x_i(t))$ and $s_j = s_j(t) = f_j(x_j(t))$, unless otherwise specified
- The solution of Eq. 1 is given by

$$w_{ij}(t) = w_{ij}(0)e^{-t} + e^{-t}\int_0^t s_i(\tau)s_j(\tau)e^{\tau}\,d\tau \qquad (2)$$

- where $w_{ij}(0)$ is the initial value of the weight at time $t = 0$.
- The first term shows that the past knowledge will decay exponentially to zero, which is equivalent to forgetting.
- The second term corresponds to correlation encoding with fading memory due to the exponential weight factor in the integral term.
- The Hebbian learning accumulates the correlation terms, giving more weightage to the recent terms.
- The stochastic version of the Hebbian learning given in Eq. 1 is approximated to the following stochastic differential equation:

$$\dot{w}_{ij}(t) = -w_{ij}(t) + s_i s_j + n_{ij}(t) \tag{3}$$

where $n_{ij}(t)$ is assumed to be a zero-mean Gaussian white noise process, independent of the signal process $-w_{ij}(t) + s_i s_j$.

- Synaptic equilibrium in the deterministic case is given by the steady state condition:

$$\dot{w}_{ij}(t) = 0 \tag{4}$$

That is, there is no further change of weights

- In the stochastic case, the weights reach stochastic equilibrium when the weight changes are only due to the noise component. That is

$$\dot{w}_{ij}(t) = n_{ij}(t) \tag{5}$$

- In these cases the synaptic processes fluctuate randomly at stochastic equilibrium as the weights approach the asymptotic values.
- Note that the stochastic equilibrium corresponds to the deterministic equilibrium on the average.

- That is the average or expectation $E[\dot{w}_{ij}(t)] = 0$, for all t after w_{ij} reaches equilibrium.
- Some version of the Hebbian learning given by other authors are:

$$\dot{w}_{ij}(t) = -w_{ij}(t) + (s_i - \bar{s}_i)(s_j - \bar{s}_j) \tag{6}$$

$$\dot{w}_{ij}(t) = -w_{ij}(t) + (s_i - \bar{s}_i)s_j \tag{7}$$

$$\dot{w}_{ij}(t) = -w_{ij}(t) + w_{ij}(t)s_i s_j \tag{8}$$

Competitive Learning

- Learning laws which modulate the difference between the output signal and the synaptic weight belong to the category of competitive learning.
- The general form of competitive learning is given by the following expression for the synaptic dynamics

$$\dot{w}_{ij}(t) = s_i[s_j - w_{ij}(t)] \tag{9}$$

where, $s_i = f_i(x_i(t))$ is the output signal of the unit i, and $s_j = f_j(x_j(t))$ is the output signal of the unit j.

- This is also called the deterministic competitive learning law and also can be written as

$$\dot{w}_{ij}(t) = -s_i w_{ij}(t) + s_i s_j \tag{10}$$

- The Eq. 10 is similar to the deterministic Hebbian learning Eq. 1, except that the forgetting term $-s_i w_{ij}(t)$, is non-linear in this case, whereas it was linear in the Hebbian case
- Here learning or adjustment of weights takes place only when there is a non-zero post-synaptic signal s_i
- If $s_i = 0$, then $\dot{w}_{ij}(t) = 0$, means synaptic weights do not change.
- Unlike in the Hebbian, in the competitive learning the system does not forget the past learning when the post-synaptic signal is zero.
- In the Hebbian, for $s_i = 0, \dot{w}_{ij}(t) = -w_{ij}(t)$, which results in forgetting the knowledge already acquired
- If the input layer units are linear, i.e., $s_j = x_j$, then the resulting learning is called linear competitive learning, and is given by

$$\dot{w}_{ij}(t) = s_i[x_j - w_{ij}(t)] \tag{11}$$

- The stochastic versions of the competitive learning are approximated to the following stochastic differential equations
  - Random competitive learning

$$\dot{w}_{ij}(t) = s_i[s_j - w_{ij}(t)] + n_{ij}(t) \tag{12}$$

  - Random Linear competitive learning

$$\dot{w}_{ij}(t) = s_i[x_j - w_{ij}(t)] + n_{ij}(t) \tag{13}$$

    where $n_{ij}(t)$ is assumed to be a zero-mean Gaussian white noise and independent of the signal.
- If the input space is partitioned into K different non-overlapping subspaces then a reinforcement function for an input pattern a is defined as

$$r_i(a) = \begin{cases} 1 & \text{if } a \in D_i \\ -1 & \text{if } a \notin D_i \end{cases}$$

where D_i is the non-overlapping subspaces and $D_i \cap D_j = 0$ for $i = 1, 2, \ldots, K, j = 1, 2, \ldots, K$ and $i \neq j$

- Using the above reinforcement function, the following supervised learning laws are defined:
  - Random supervised competitive learning

$$\dot{w}_{ij}(t) = r_i(a)s_i[s_j - w_{ij}(t)] + n_{ij}(t) \tag{14}$$

  - Random supervised Linear competitive learning

$$\dot{w}_{ij}(t) = r_i(a)s_i[x_j - w_{ij}(t)] + n_{ij}(t) \tag{15}$$

    where $n_{ij}(t)$ is assumed to be a zero-mean Gaussian white noise and independent of the signal.
- Where we can apply competitive learning?
  - The competitive learning works in a situation where an external input is presented to an input layer of units and these units feed signals to each of the units in the output layer.
  - The signals from the units in the output layer compete with each other, leaving eventually one of the units (say i) as the winner.

- The weights leading to this unit are adjusted according to the learning law.
- This is also called the 'winner take-all' situation, since only one unit in the output layer will have a non-zero output eventually.
- The corresponding weights w_{ij} from all the input units j are adjusted to match the input vector.
- The Hebbian learning is generally distributed, i.e., all the weights are adjusted for every input pattern, the competitive learning is not distributed.
- In fact the input vectors leading to the same winning unit in the competitive layer will produce a weight vector for that unit which is an average of all the corresponding input vector.

Differential Competitive Learning

- Differential competition means that learning takes place only if there is a change in the post-synaptic neuronal activation.
- The deterministic differential competitive learning is described by

$$\dot{w}_{ij}(t) = \dot{s}_i[s_j - w_{ij}(t)] \tag{16}$$

- This is the combination of the competitive learning and differential Hebbian learning.
- Linear differential competitive learning law is described by

$$\dot{w}_{ij}(t) = \dot{s}_i[x_j - w_{ij}(t)] \tag{17}$$

- The stochastic versions of the above learning equations are obtained by adding a noise term to the right hand side of these differential competitive leaning equations.

- Random differential competitive learning

$$\dot{w}_{ij}(t) = \dot{s}_i[s_j - w_{ij}(t)] + n_{ij}(t) \tag{18}$$

- Random Linear differential competitive learning

$$\dot{w}_{ij}(t) = \dot{s}_i[x_j - w_{ij}(t)] + n_{ij}(t) \tag{19}$$

where $n_{ij}(t)$ is assumed to be a zero-mean Gaussian white noise and independent of the signal.

Learning Process

- The operation of a neural network is governed by neuronal dynamics.
- Neuronal dynamics consists of two parts:
  - one corresponding to the dynamics of the activation state
  - other corresponding to the dynamics of the synaptic weights.
- The Short Term Memory (STM) in neural networks is modelled by the activation state of the network.
- The Long Term Memory (LTM) corresponds to the encoded pattern information in the synaptic weights due to learning
- Learning laws are simply implementation models of synaptic dynamics.
- Typically, a model of synaptic dynamics is described in terms of expressions for the first derivative of the weights.
- They are called learning equations

- Learning laws describe the weight vector for the i^{th} processing unit at time instant $(t+1)$ in terms of the weight vector at time instant (t) as follows

  $$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

  where $\Delta w_i(t)$ is the change in the weight vector
- There are different methods for implementing the learning feature of a neural network accordingly there are several learning laws
- Some basic learning laws are discussed in the next section
- All these learning laws use only local information for adjusting the weight of the connection between two units.

Learning Rules

Learning rule or learning process is a method, mathematical logic or algorithm which improves the network's performance or training time by updating the weights and bias values for a particular data set.

- Hebb's Law
- Perceptron Learning Law
- Delta Learning Law
- Widrow and Hoff LMS Learning Law
- Correlation Learning Law
- Instar (Winner-take-all) Learning Law
- Outstar Learning Law

Hebb's Law

- The change in the weight vector is given by

$$\Delta w_i = \eta f(w_i^T a) a$$

- Therefore, the j^{th} component of Δw_i is given by

$$\begin{aligned}\Delta w_{ij} &= \eta f(w_i^T a) a_j \\ &= \eta s_i a_j \\ &\text{for } j = 1, 2, \ldots, M\end{aligned}$$

where s_i is the output signal of the i^{th} unit.

- The law states that the weight increment is proportional to the product of the input data and the resulting output signal of the unit.
- This law requires weight initialization to small random values around $w_{ij} = 0$ prior to learning.
- This is used for unsupervised learning.

Perceptron Learning Law

- The change in the weight vector is given by

$$\Delta w_i = \eta[b_i - sgn(w_i^T a)]a$$

where $sgn()$ is a sign function and b_i is the desired output

- Therefore, the j^{th} component of Δw_i is given by

$$\begin{aligned}\Delta w_{ij} &= \eta[b_i - sgn(w_i^T a)]a_j \\ &= \eta[b_i - s_i]a_j \\ &\text{for } j = 1, 2, \ldots, M\end{aligned}$$

- This law is applicable only for bipolar output functions $f(.)$.
- This is also called discrete perceptron learning law.

- The expression for Δw_{ij} shows that the weights are adjusted only if the actual output s_i is incorrect, since the term in the square brackets is zero for the correct output.
- This is a supervised learning law, as the law requires a desired output for each input.
- In implementation, the weights can be initialized to any random initial values, as they are not critical.
- The weights converge to the final values eventually by repeated use of the input-output pattern pairs, provided the pattern pairs are representable by the system.

Delta Learning Law

- The change in the weight vector is given by

$$\Delta w_i = \eta[b_i - f(w_i^T a)]\dot{f}(w_i^T a)a$$

where $\dot{f}(x)$ is the derivative with respect to x.

- So the equation becomes

$$\begin{aligned}\Delta w_{ij} &= \eta[b_i - f(w_i^T a)]\dot{f}(w_i^T a)a_j \\ &= \eta[b_i - s_i]\dot{f}(x_i)a_j \\ &\text{for } j = 1, 2, \ldots, M\end{aligned}$$

- This law is valid only for a differentiable output function, as it depends on the derivative of the output function $\dot{f}(.)$

- It is a supervised learning law since the change in the weight is based on the error between the desired and the actual output values for a given input.
- Delta learning law can also be viewed as a continuous perceptron learning law.
- The weights can be initialized to any random values as the values are not very critical.
- The weights converge to the final values eventually by repeated use of the input-output pattern pairs.
- The convergence can be more or less guaranteed by using more layers of processing units in between the input and output layers.
- The delta learning law can be generalized to the case of multiple layers of a feedforward network.

Widrow and Hoff LMS Learning Law

- The change in the weight vector is given by

$$\Delta w_i = \eta[b_i - w_i^T a]a$$

- So the equation becomes

$$\Delta w_{ij} = \eta[b_i - w_i^T a]a_j$$
$$\text{for } j = 1, 2, \ldots, M$$

- This is a supervised learning law and is a special case of the delta learning law, where the output function is assumed linear, i.e., $f(x_i) = x_i$.

- In this case the change in the weight is made proportional to the negative gradient of the error between the desired output and the continuous activation value, which is also the continuous output signal due to linearity of the output function.
- Hence, this is also called the Least Mean Squared (LMS) error learning law.
- This is same as the learning law used in the Adaline model of neuron.
- The weights may be initialized to any values.
- The input-output pattern pairs data is applied several times to achieve convergence of the weights for a given set of training data.
- The convergence is not guaranteed for any arbitrary training data set.

Correlation Learning Law

- The change in the weight vector is given by

$$\Delta w_i = \eta b_i a$$

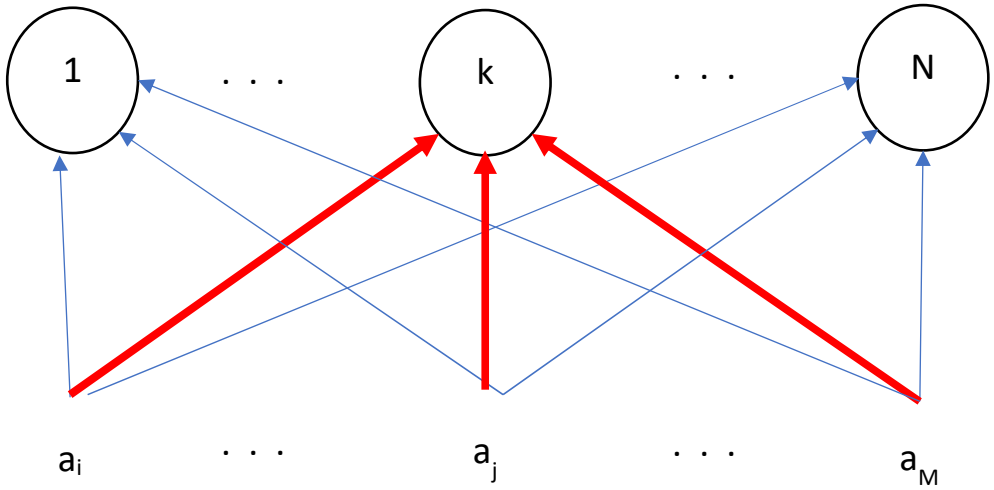
- So the equation becomes

$$\Delta w_{ij} = \eta b_i a_j$$
$$\text{for } j = 1, 2, \ldots, M$$

- This is a special case of the Hebbian learning with the output signal s_i being replaced by the desired signal b_i.
- But the Hebbian learning is an unsupervised learning, whereas the correlation learning is a supervised learning, since it uses the desired output value to adjust the weights.
- In the implementation of the learning law, the weights are initialised to small random values close to zero, i.e., $w_{ij} = 0$.

Instar (Winner-take-all) Learning Law

- All the inputs are connected to each of the units in the output layer in a feed-forward manner as shown in the below fig.



- For a given input vector a, the output from each unit i is computed using the weighted sum $w_i^T a$.
- The unit k that gives maximum output is identified. That is

$$w_k^T a = \max(w_i^T a), \text{ for } i = 1, 2, \ldots, N$$

- The weight vector leading to the k^{th} unit is adjusted as follows:

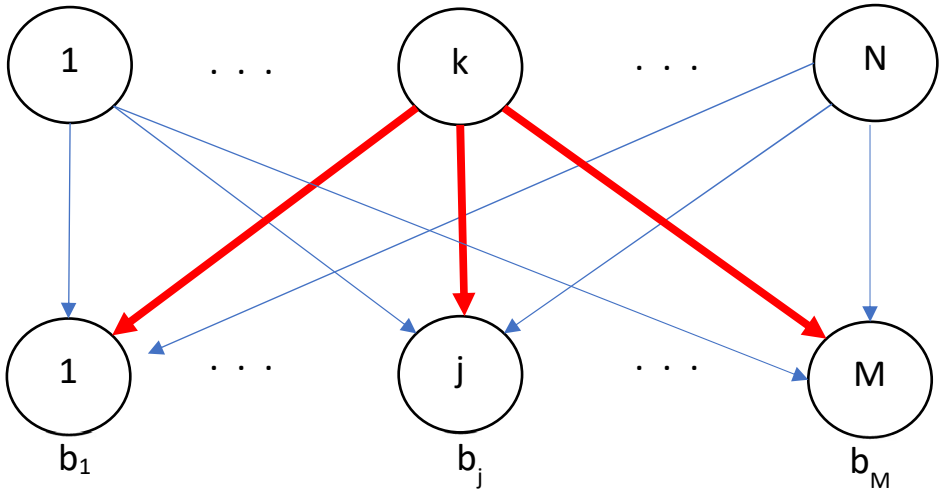
$$\Delta w_k = \eta(a - w_k)$$

$$\Delta w_{kj} = \eta(a - w_{kj}), \text{ for } j = 1, 2, \ldots, M$$

- The final weight vector tends to represent a group of input vectors within a small neighbourhood.
- This is a case of unsupervised learning.
- The values of the weight vectors are initialized to random values prior to learning, and the vector lengths are normalized during learning.

Outstar Learning Law

- In outstar learning, group of units arranged in a layer as shown in below Fig



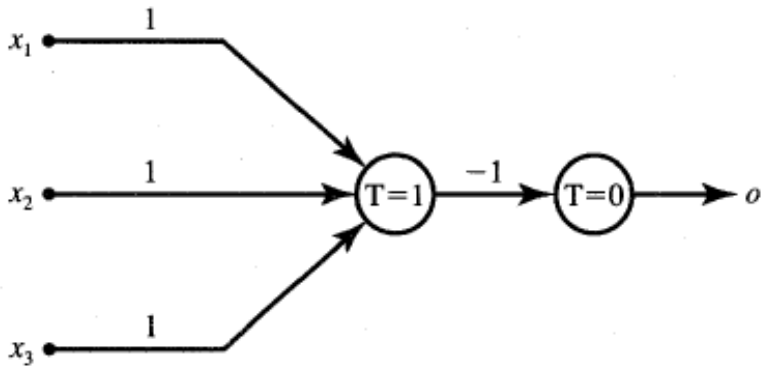
- The weights are adjusted so as to capture the desired output pattern characteristics as per the eq.

$$\Delta w_{jk} = \eta(b_j - w_{jk}), \text{ for } j = 1, 2, \ldots, M$$

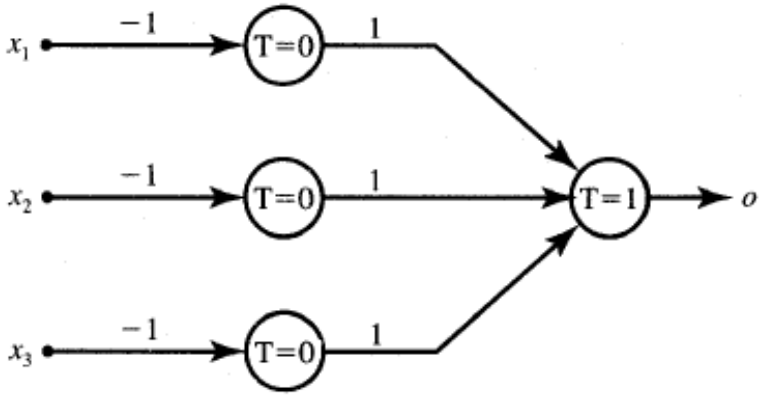
- The k^{th} unit is the only active unit in the input layer.
- The vector $b = (b_1, b_2, \ldots, b_M)^T$ is the desired response from the layer of M units.
- The outstar learning is a supervised learning law
- It is used with a network of instars to capture the characteristics of the input and output patterns for data compression.
- The weight vectors are initialized to zero prior to learning.

Tutorial 4

1. What are the basic learning laws?
2. Explain the significance of the initial values of weights and the learning rate parameter in the seven basic learning laws.
3. Identify supervised and unsupervised basic learning laws.
4. Compare LMS, perceptron and delta learning laws.
5. Find the truth table for the given MP neuron model and identify the gate output.



6. Find the truth table for the given MP neuron model and identify the gate output.



7. Illustrates Hebbian learning with binary ($sgn(x)$) and continuous activation functions ($f(x) = 1/(1 + e^{-x})$ of a very simple network. Assume the network with the initial weight vector

$$w = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

needs to be trained using the set of three input vectors as below

$$a_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, \; a_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, \; a_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

for an arbitrary choice of learning constant $\eta = 1$

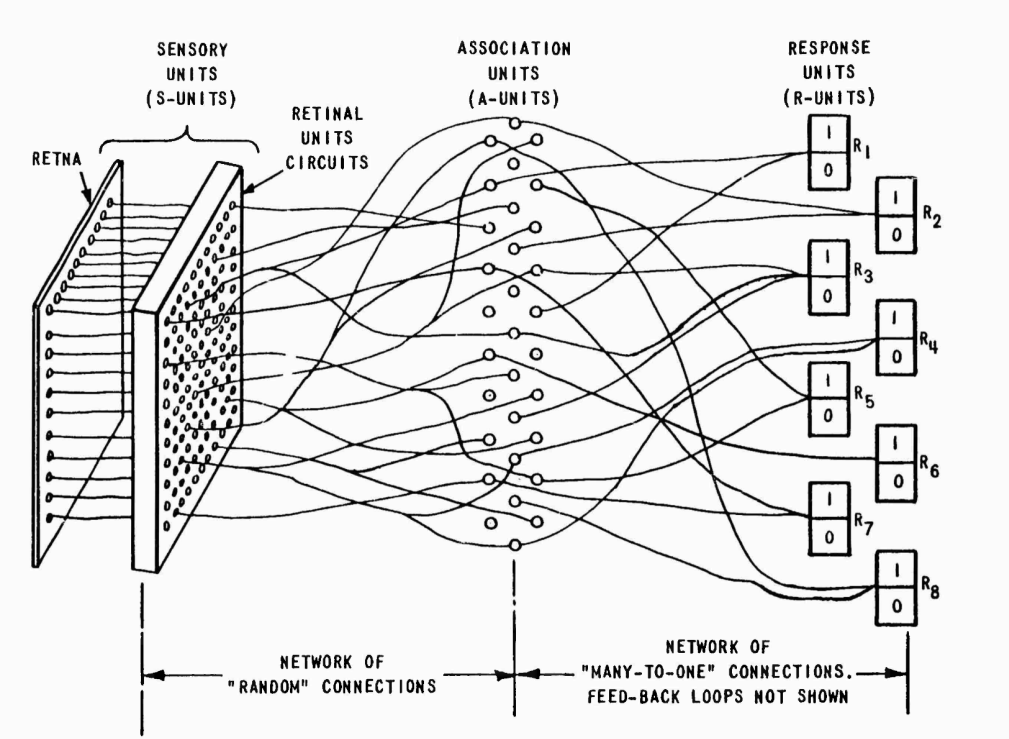
8. Explain the Perceptron Learning rule using the below data:

$$a_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, \; a_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, \; a_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}, \; w = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

and desired output is $b_i = -1, b_2 = -1, b_3 = 1$, and learning constant $\eta = 0.1$

Single Layer Perceptron Classifier

- The perceptron which is basically derived from Mark I perceptron is shown in below fig.



- It consists of three layers of units called the S (sensor), A (association) and R (response) units.
- Only the last of these, the R units, have adjustable weights
- The term 'perceptron' has become synonymous with the units in the R layer.
- Why this network would be considered to be a single-layered perceptron, because there is only one layer of elements with adjustable weights
- In Mark I, the S layer consists of a digitized image, where 1 is used to represent white pixels and 0 for black pixels.
- Groups of pixels are randomly connected to units in the A layer.
- These produce binary 0/1 outputs as a function of the inputs - in other words they are logical feature detectors.
- For example, one of the units might fire that is produce a output 1 when the majority of pixels connected to it are 1.

- The outputs from the A layer are connected to units in the R layer
- In R unit, Inputs are weighted and summed, and then an output of either 0 or 1 is generated, depending on the value of the weighted sum.
- The inputs and outputs are binary values of 0 and 1
- One input as bias we keep fix in this, so for input space there will be m+1 size input vector.
- The error is measured as the difference between the desired output and the actual output from the non-linear output function.
- The weight adjustment is incremental, in the sense that only unit changes are made that is +1,-1 or no change.

$$\Delta w_i = a_i(Desired - Actual) = a_i(b_i - s_i)$$

- The above description is the Rosenblatt's original perceptron, but there have been many variations includes now:
  - lifting the restriction that inputs are 0 or 1, allowing any real number
  - including a learning coefficient η
- The importance of perceptrons in the development of neural networks is partly due to the perceptron convergence theorem.
- This states that if a solution can be implemented on a perceptron, the learning rule will find a solution in a finite number of steps.
- Implement $x_1 \vee x_2$
- Implement $\neg(x_1 \wedge x_2)$
- Implement $x_1 \wedge x_2$
- Only linearly separable functions can be implemented on a single layer network, so the perceptron convergence theorem only applies to single-layered perceptrons.

- The limitation of only being able to implement linearly separable functions can be overcome using multi-layered networks.

Multilayer Feedforward Network

- The objective is to develop a learning algorithm for a multilayer feedforward neural network, so that the network can be trained to capture the mapping implicit in the given set of input-output pattern pairs.
- The approach to be followed is basically a gradient descent along the error surface to arrive at the optimum set of weights.
- The error is defined as the squared difference between the desired output (i.e., given output pattern) and the actual output obtained at the output layer of the network due to application of an input pattern from the given input-output pattern pair.
- The output is calculated using the current setting of the weights in all the layers.

- The optimum weights may be obtained if the weights are adjusted in such a way that the gradient descent is made along the total error surface.
- The desirable characteristic of any learning law is to specify the incremental update of the weights of the network for each presentation of an input-output pattern pair.
- While this may result in a suboptimal solution, in most cases of practical significance the result is acceptable
- A learning law, called generalized delta rule or backpropagation law, is derived here
- Let $(a_i, b_i), i = 1, 2, ..., L$ be the set of training pattern pairs.
- It is not necessary to have all the training data set at one time, nor the training data set to be a finite set.
- The objective is to determine the weight update for each presentation of an input-output pattern pair

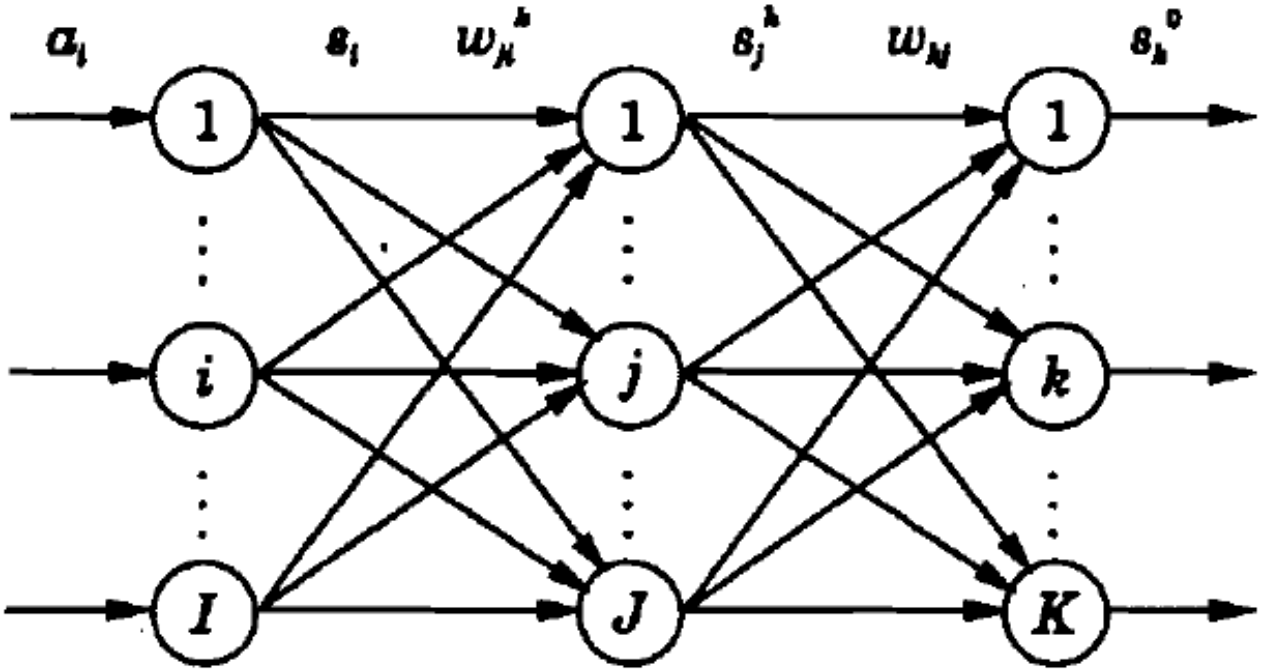
- Since the given data may be used several times during training, let us use the index m to indicate the presentation step for the training pair at step m.
- For training a multilayer feedforward neural network, we use the following estimate of the gradient descent along the error surface to determine the increment in the weight connecting the units i and j:

$$\Delta w_{ij} = -\eta \frac{\partial E(m)}{\partial w_{ij}}$$

where $q > 0$ is a learning rate parameter, which may also vary for each presentation of the training pair.

- The weight update is given by

$$w_{ij}(m+1) = w_{ij}(m) + \Delta w_{ij}(m)$$

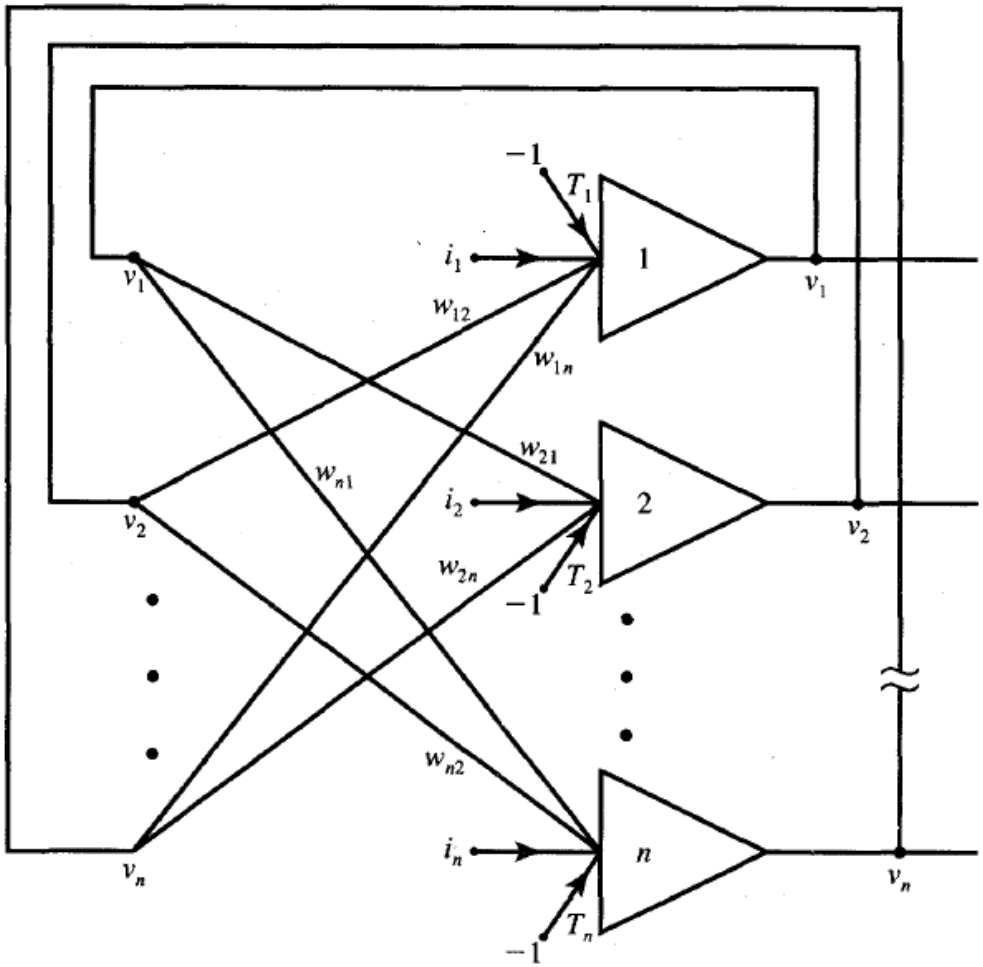


Tutorial 5

1. Drive the weight values for $x_1 \vee x_2$ for single layer perceptron.
2. Implement $x_1 \wedge x_2$ using single layer perceptron.
3. Explain every step to find out the weight values for $\neg(x_1 \vee x_2)$
4. Find out the step by step weight values for $\neg(x_1 \wedge x_2)$
5. Can you design a single layer perceptron for logical XOR? If not then why?
6. What is the perceptron convergence theorem?
7. Draw the block diagram for logical XOR using multilayer perceptron.

Single Layer Feedback Networks

- Fig below is an example of Single Layer Feedback Networks



- It consists of n neurons having threshold values T_i.
- The feedback input to the i^{th} neuron is equal to the weighted sum of neuron outputs v_j, where $j = 1, 2, ..., n$.
- w_{ij} is the weight value connecting the output of the j^{th} neuron with the input of the i^{th} neuron
- Total input net_i of the i^{th} neuron will be

$$net_i = \sum_{j=1, j \neq i}^{n} w_{ij} v_j + i_i - Ti \text{ for } i = 1, 2, \ldots, n \tag{20}$$

where i_i is the external input to the i^{th} neuron

- Introducing the vector notation for synaptic weights and neuron output, Eq 20 can be rewritten as

$$net_i = w_i^t v + i_i - T_i \text{ for } i = 1, 2, \ldots, n \tag{21}$$

where w_i is the weight vector containing weights connected to the input of the i^{th} neuron,

$$w_i = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \vdots \\ w_{in} \end{bmatrix}$$

v is the neural network output vector, also called the output vector.

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

- The complete matrix description of the linear portion of the system is given by

$$net = Wv + i - t \tag{22}$$

where $net = \begin{bmatrix} net_1 \\ net_2 \\ \vdots \\ net_n \end{bmatrix}$, $i = \begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{bmatrix}$, $t = \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_n \end{bmatrix}$, $W = \begin{bmatrix} w_1^t \\ w_2^t \\ \vdots \\ w_n^t \end{bmatrix}$

- Matrix W, sometimes called the connectivity matrix, is an $n \times n$ matrix containing network weights arranged in rows of vectors w_i^t and can be written in expended form as

$$W = \begin{bmatrix} 0 & w_{12} & w_{13} & \cdots & w_{1n} \\ w_{21} & 0 & w_{23} & \cdots & w_{2n} \\ w_{31} & w_{32} & 0 & \cdots & w_{2n} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ w_{n1} & w_{n2} & w_{n3} & \cdots & 0 \end{bmatrix}$$

- We assume the weight matrix W in this model is symmetrical, i.e., $w_{ij} = w_{ji}$, and with diagonal entries equal explicitly to zero, i.e., $w_{ii} = 0$.
- Assume that the neuron's activation function is $sgn(.)$ then i^{th} neuron output will be

$$v_i = \begin{cases} -1 & \text{if } net_i < 0 \\ +1 & \text{if } net_i > 0 \end{cases}$$

- Update algorithm for a discrete-time recurrent network will be

$$v_i^{k+1} = sgn(w_i^t v^k + i_i - T_i) \tag{23}$$

for $i = 1, 2, \ldots, n$ and $k = 0, 1, \ldots$

Types of learning

- Supervised
- Unsupervised
- Reinforcement learning

Reinforcement learning

- In supervised learning, we saw algorithms that tried to make their outputs mimic the labels y given in the training set.
- In that setting, the labels gave an unambiguous "right answer" for each of the inputs x.
- In contrast, for many sequential decision making and control problems, it is very difficult to provide this type of explicit supervision to a learning algorithm.
- For example, if we have just built a four-legged robot and are trying to program it to walk, then initially we have no idea what the "correct" actions to take to make it walk, and so do not know how to provide explicit supervision for a learning algorithm to try to mimic.

- In the reinforcement learning framework, we will instead provide our algorithms only a reward function, which indicates to the learning agent when it is doing well, and when it is doing poorly.
- In the four-legged walking example, the reward function might give the robot positive rewards for moving forwards, and negative rewards for either moving backwards or falling over.
- It will then be the learning algorithm's job to figure out how to choose actions over time so as to obtain large rewards.
- Reinforcement learning has been successful in applications as diverse as autonomous helicopter flight, robot legged locomotion, cell-phone network routing, marketing strategy selection, factory control, and efficient web page indexing.
- Markov decision processes (MDP) provides the formalism in which RL problems are usually posed.

- RL main idea is: An agent must learn how to behave in the environment he is located at in order to optimize the long term reward received from the environment.
- Some points that makes RL so important:
  * RL framework basically describes a closed loop system. The agent (controller) performs actions (control signal), observes the environment state (system output) and receives a reward that has to be optimized.
  * Not necessarily the agent must have previous knowledge about the environment. Those algorithms are known as model-free methods.
  * The learning basically occur by trial-and-error. This mimics really well the learning process of a human.
  * Also, being an optimization problem, it is possible to obtain nice convergence properties for many algorithms.

Why to use Reinforcement Learning

- A model of the environment is known, but an analytic solution is not available
- Difficult to solve
- Computation is offline
- Exact knowledge of dynamics is required
- Maximizes Performance
- Sustain Change for a long period of time

Knowledge, Representation, and Acquisition

- Knowledge is a facts, information, and skills acquired through experience or education about some topic, subject, situation.
- Knowledge acquisition is the process of extracting, structuring and organizing knowledge from one source.
- Knowledge representation is the way to store or keep knowledge in mind on in computer to make it easy and fast access whenever we need it.
  - Logical Representation
  - Semantic Network
  - Frame Representation
  - Production Rules

Hopfield Model

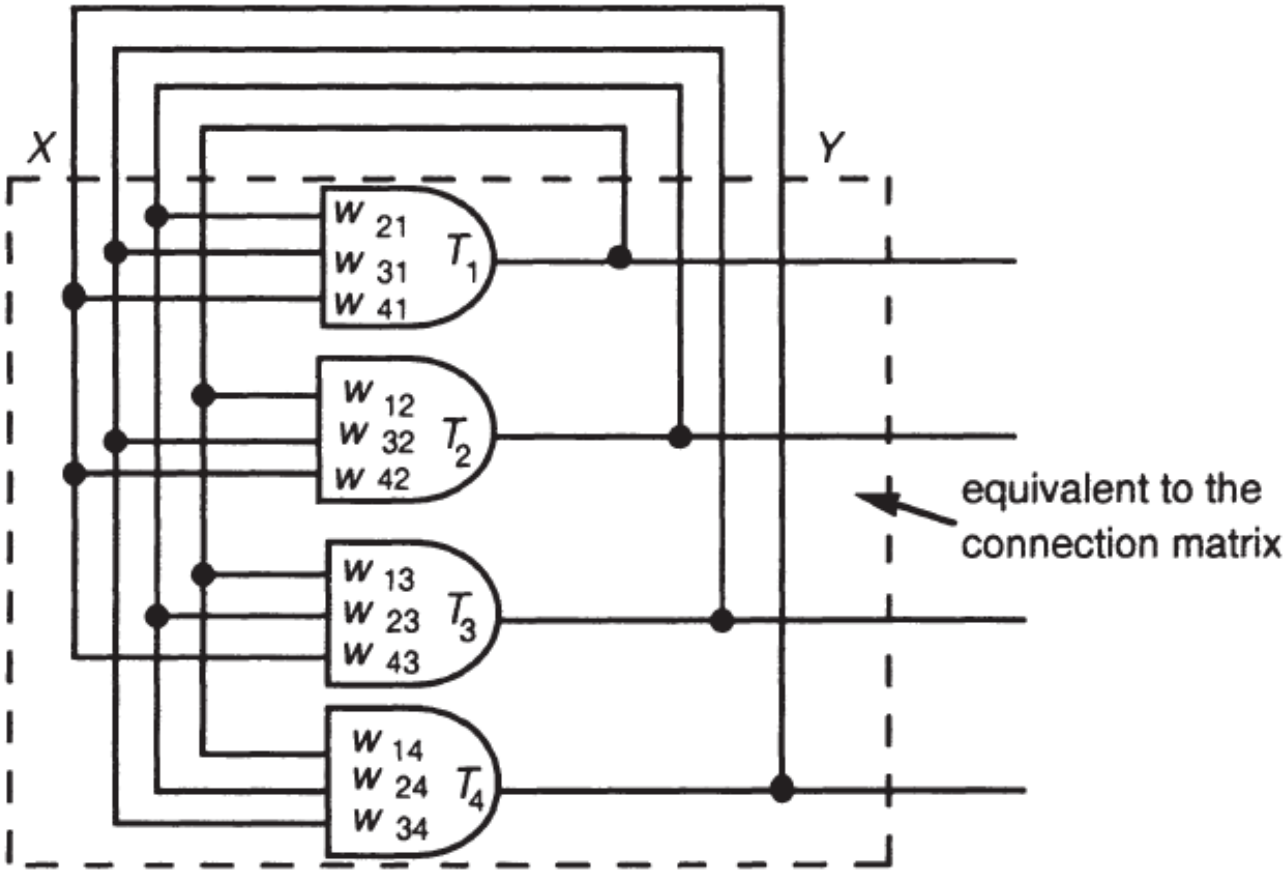
- Let single layer feedback network, where the output of each unit is fed to all the other units with weights w_{ij}, for all i and j.
- Let the output function of each of the units be bipolar (± 1) so that

$$s_i = f(x_i) = sgn(x_i) \tag{24}$$

and

$$x_i = \sum_{j=1}^{N} w_{ij} s_j - \theta_i \tag{25}$$

where θ_i is the threshold for the unit i and we will assume $\theta_i = 0$ for simplicity



- The state of each unit is either $+1$ or -1 at any given instant of time.

- Due to feedback, the state of a unit depends on the states of the other units.
- The updating of the state of a unit can be done synchronously or asynchronously.
- In the synchronous update all the units are simultaneously updated at each time instant, assuming that the state of the network is frozen until update is made for all the units.
- In the asynchronous update a unit is selected at random and its new state is computed.
- Another unit is selected at random and its state is updated using the current state of the network.
- The updating a unit by random selection is continued until no further change in the state takes place for all the units.
- That is, the state at time $(t+1)$ is the same as the state at time t for all the units.

$$s_i(t+1) = s_i(t) \text{ for all } i \tag{26}$$

- In this situation we can say that the network activation dynamics reached a stable state. We assume asynchronous update throughout the following discussion.
- If the network is to store a pattern $a = (a_1, a_2, \ldots, a_N)^T$, then in a stable state we must have the updated state value to be the same as the current state value. That is

$$sgn\left(\sum_{j=1}^{N} w_{ij} a_j\right) = a_i \text{ for all } i \tag{27}$$

This can happen if $w_{ij} = \frac{1}{N} a_i a_j$, as

$$\sum_{j=1}^{N} w_{ij} a_j = \sum_{j=1}^{N} \frac{1}{N} a_i a_j a_j = \frac{a_i}{N} \sum_{j=1}^{N} a_j^2 = a_i \tag{28}$$

as $a_j^2 = 1$ for bipolar states ± 1

- For storing L patterns, we could summation the terms for each pattern.

$$w_{ij} = \frac{1}{N}\sum_{l=1}^{L} a_{li}a_{lj} \tag{29}$$

- Then the state a_k will be stable if

$$sgn\left(\frac{1}{N}\sum_{j=1}^{N}\sum_{l=1}^{L} a_{li}a_{lj}a_{kj}\right) = a_{ki} \text{ for all } i \tag{30}$$

- Taking $l = k$ term in the summation

$$sgn\left(\frac{1}{N}\sum_{j=1}^{N}\left(\sum_{l=1,l\neq k}^{L} a_{li}a_{lj}a_{kj} + a_{ki}a_{kj}a_{kj}\right)\right) = a_{ki}$$

$$sgn\left(\frac{1}{N}\sum_{j=1}^{N}\sum_{l=1,l\neq k}^{L} a_{li}a_{lj}a_{kj} + \frac{1}{N}\sum_{j=1}^{N} a_{ki}a_{kj}a_{kj}\right) = a_{ki} \text{ for all } i$$

- After simplification as $a_{kj}^2 = 1$

$$sgn\left(a_{ki} + \frac{1}{N}\sum_{j=1}^{N}\sum_{l=1,l\neq k}^{L} a_{li}a_{lj}a_{kj}\right) = a_{ki} \text{ for all } i \tag{31}$$

- Eq 31 is true for all a_{ki} as $a_{ki} = \pm 1$ and provided both the term having same sign or $|a_{ki}| >>$ other term
- Eq 31 is satisfied if the number of patterns L is limited to the storage capacity of the network, i.e., the maximum number of patterns that can be stored in the network

Hopfield Network Algorithm for Bipolar Patterns

- Let the network have N fully connected units
- Each unit having hard-limiting bipolar threshold output function.
- Let $a_l, 1 = 1, 2, ..., L$ be the vectors to be stored.
- The vector a_l are assumed to have bipolar components, i.e, $a_{li} = \pm 1$ for $i = 1, 2, ..., N$.

1. Assign the connection weights

$$w_{ij} = \begin{cases} \frac{1}{N} \sum_{l=1}^{L} a_{li} a_{lj} \text{ for } i \neq j \\ 0 \text{ for } i = j \end{cases} \qquad 1 \leq i, j \leq N$$

2. Initialize the network output with the given unknown input pattern a

$$s_i(0) = a_i, \text{for } i = 1, 2, \ldots, N$$

where $s_i(0)$ is the output of the unit i at time $t = 0$

3. Repeat until convergence

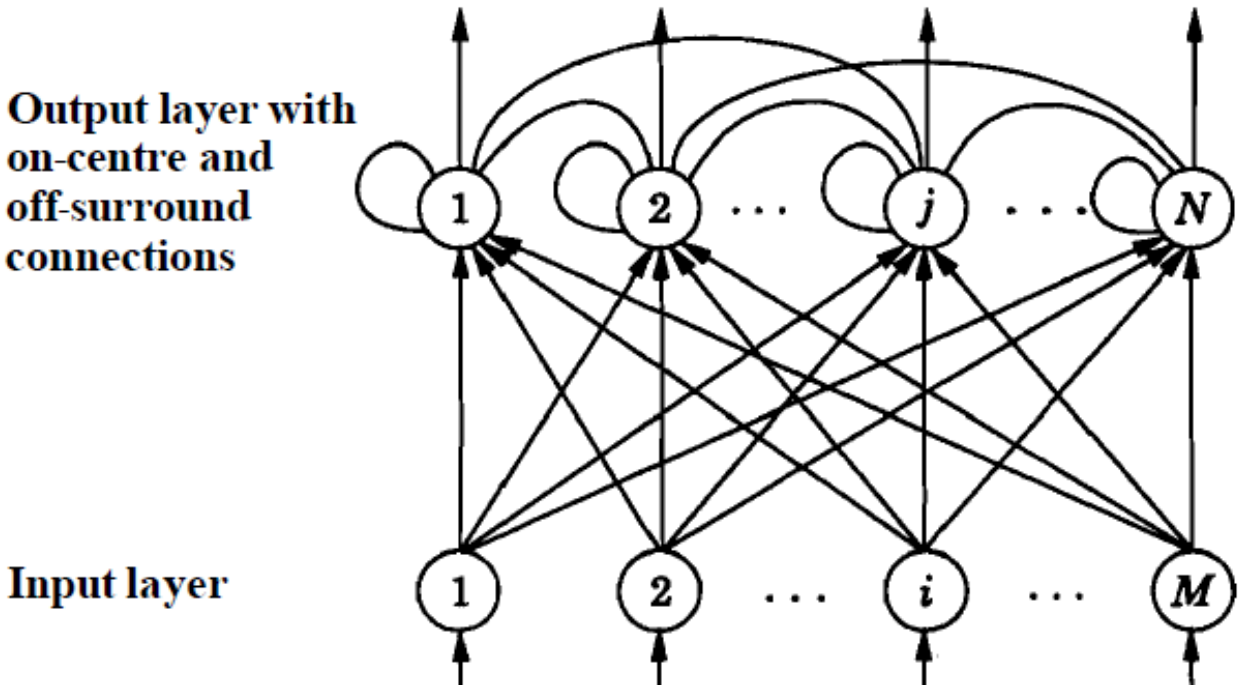
$$s_i(t+1) = sgn\left(\sum_{j=1}^{N} w_{ij} s_j(t)\right), \text{ for } i = 1, 2, \ldots, N$$

Tutorial 6

1. What is the significance of the nonlinear output function of the units in feedback neural networks?
2. Explain the meaning of activation state and energy landscape of a feedback network.
3. What is meant by capacity of a feedback network?
4. What is the Hopfield model of a neural network?
5. Explain the differences between discrete and continuous Hopfield models in terms of energy landscape and stable states.
6. What are the four fundamental types of knowledge representation techniques?
7. How does the knowledge can be categorized?

8. List the major sources of knowledge.
9. What are the difficulties in Knowledge Acquisition?
10. What are the different types of learning? Where does Reinforcement Learning is used?

Competitive learning



- The network consists of an input layer of linear units.
- The output of each of these units is given to all the units in the second layer (output layer) with adaptive (adjustable) feedforward weights.

- The output functions of the units in the second layer are either linear or non-linear depending on the task for which the network is to be designed.
- The output of each unit in the second layer is feed back to itself in a self-excitatory manner and to the other units in the layer in an excitatory or inhibitory manner depending on the task.
- Generally the weights on the connections in the feedback layer are non-adaptive or fixed.
- Such a combination of both feedforward and feedback connection layers results in some kind of competition among the activations of the units in the output layer.
- Such networks are called competitive learning neural networks.
- Different choices of the output functions and interconnections in the feedback layer of the network can be used to perform different pattern recognition tasks.

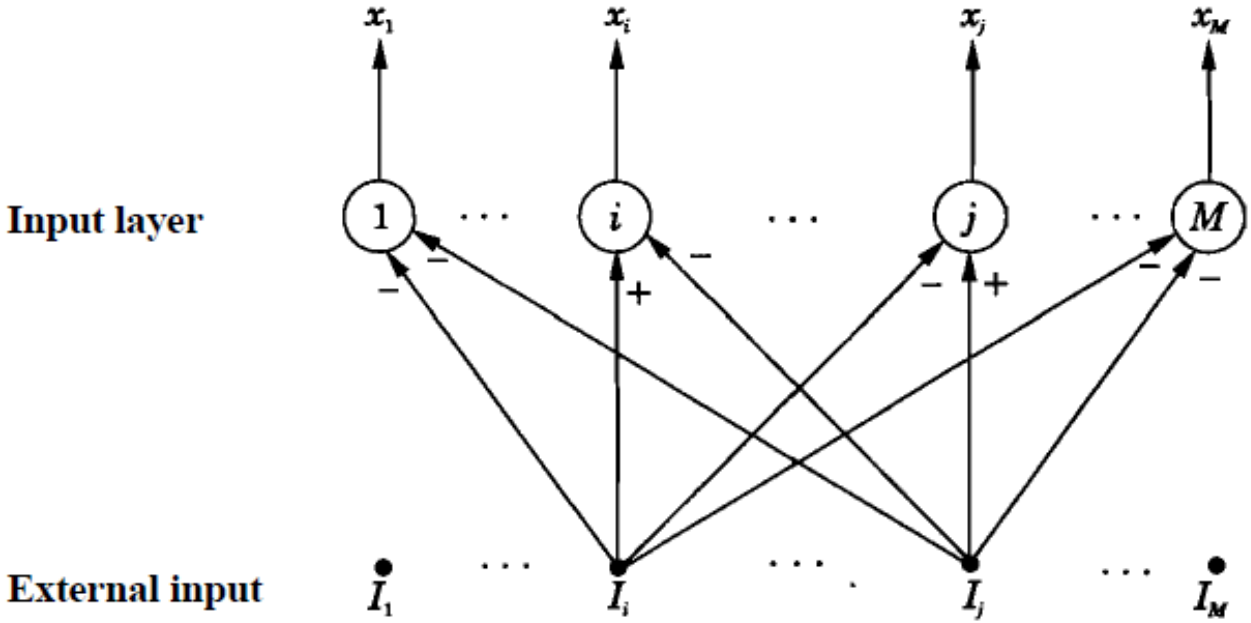
- For example, if the output functions are linear, and the feedback connections are made in an on-centre off-surround fashion, the network performs the task of storing an input pattern temporarily.
- If the output functions of the units in the feedback layer are made non-linear, with fixed weight on-centre off-surround connections, the network can be used for pattern clustering.
- If the output functions of the units in the feedback layer are non-linear and the units are connected in such a way that connections to the neighbouring units are all made excitatory and to the farther units inhibitory, the network then can perform the task of feature mapping. The resulting network is called a self-organization network

Components of a Competitive Learning Network

- The Input Layer
- The Instar
- Feedback Layer
- Basic Competitive Learning

Input Layer

- The purpose of this layer is to distribute the given external input pattern vector to the feedforward portion of the network as shown in the below fig.



- The input vectors may be of varying magnitude, even though they may contain the same pattern information.

- It is necessary to have the inputs bounded to some limits.
- In an on-line situation, the input layer should not feed background noise to the feedforward portion of the competitive learning network.
- A shunting activation model with on-centre off-surround configuration takes care of the noise-saturation problem of the input layer

$$\dot{x} = -Ax_i + (B - x_i)I_i - (C + x_i)\sum_{j \neq i} I_j \tag{32}$$

- The steady state activation value of the i^{th} unit is given by

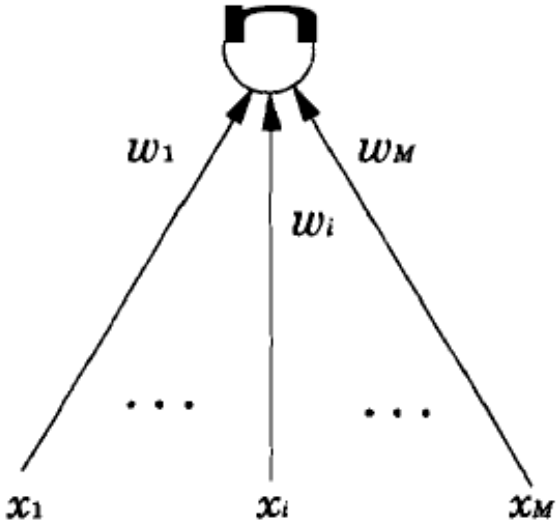
$$x_i(\infty) = \frac{(B + C)I_i - CI}{A + I} = \left[\frac{I_i}{I} - \frac{C}{B + C}\right]\frac{B + C}{1 + \frac{A}{I}} \tag{33}$$

where $I = \sum_{i=1}^{M} I_i$ and all the inputs I_i are non-negative.

- The above equations show that in the steady state the activation value of the i^{th} unit is confined to the range $[-C, B]$.
- The output function of these units is assumed to be linear for $x > 0$
- The output of the units will be zero as long as the inputs $I_i < CI/(B + C)$
- The input should be greater than some minimum value before it can make the activation of the unit positive.
- The units in the input layers do not respond to noise input, if the noise amplitude is below some threshold value.
- Therefore the input to the feedforward portion is always positive and is limited to a maximum value of B.
- This input layer not only normalizes the input data values, but also takes care of the noise-saturation problem with an on-line input data.

The Instar

- Each unit in the feedback layer receives inputs from all the input units as shown in the below fig.



- A configuration where a unit receives weighted inputs from several units of another layer is called an instar.
- Let $x = (x_1, x_2, \ldots, x_M)^T$ is the input vector and $w = (w_1, w_2, \ldots, w_M)^T$ is weight vector.
- The net input to the instar processing unit is given by $w^T x$.

- The activation dynamics of the instar processing unit is given by the following additive model with a passive decay term and the net input term:

$$\dot{y}(t) = -y(t) + w^T x \tag{34}$$

where we have assumed the decay constant to be 1.

- The solution of the above equation is

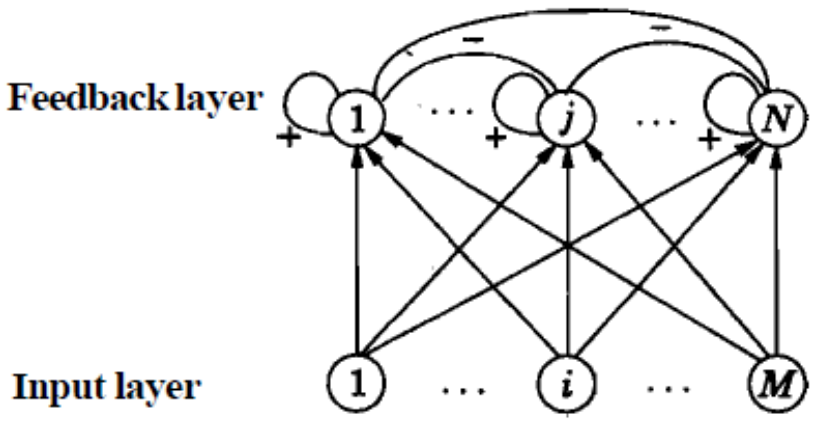
$$y(t) = y(0)e^{-t} + w^T x(1 - e^{-t}) \tag{35}$$

- The steady state activation value will be zero when the external input x is removed and given by

$$y(\infty) = w^T x \tag{36}$$

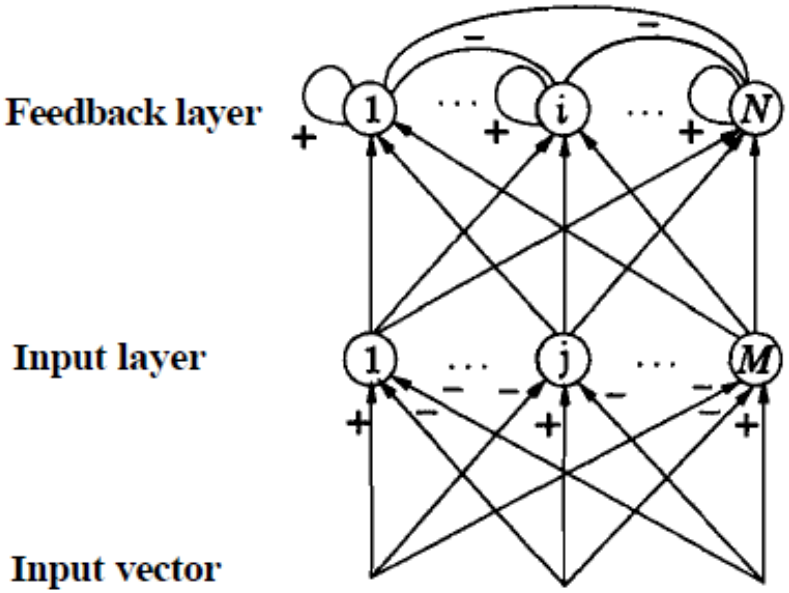
Feedback Layer

- The category of an input vector can be identified by observing the instar processing unit with maximum response.
- The maximum response unit can be determined by the system itself if the outputs of the instar processing units are feed back to each other.
- If the processing units of a group of instars are connected as an on-centre off-surround feedback network, as shown in the below fig then the feedback layer is called a competitive layer



- In this layer there is an excitatory self-feedback to each unit, and an inhibitory feedback from a given unit to all other units.
- The excitatory feedback is indicated by a positive $(+)$ weight, and the inhibitory feedback is indicated by a negative $(-)$ weight.
- Normally these weights are fixed a priori.
- The nature of the output function of the instar processing units will determine the behaviour expected from such a competitive layer.

- The complete competitive learning network discussed so far is shown in the below fig.

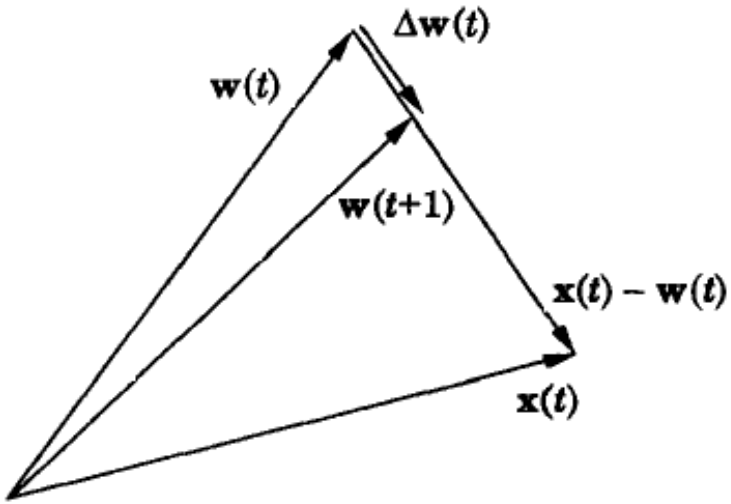


Tutorial 7

1. What is the significance of hidden units in a feedback network?
2. What do you meant by stochastic update of a neuron?
3. Explain the concept of equilibrium in stochastic neural networks.
4. What are the components of a competitive learning network?
5. Describe the operation of an input layer when it is directly connected to the environment.
6. What is an instar network?
7. Describe the basic learning feature of an instar and discuss its application.

Basic Competitive Learning

- The steady activation value with an external input depends on the angle between the input and weight vectors as shown in below fig



- For the instar to respond maximally for a given input vector x, the weight vector is moved towards the input vector

- The adjustment of the weight vector is governed by the following synaptic dynamics equation

$$\dot{w} = [x - w]f(y) \tag{37}$$

where $f(y)$ is the output of the instar processing unit.

- In discrete implementation the change in the weight $\Delta w(t)$ is given by

$$\Delta w(t) = \eta[x - w(t)]f(y) \tag{38}$$

where η is a learning rate parameter.

- For binary output function, $f(y) = 1$ or 0. Therefore the weights in the instar are adjusted only when the output of the instar processing unit is 1.
- The increment in the weight is given by

$$\Delta w(t) = \eta[x - w(t)] \tag{39}$$

- The updated weight is given by

$$w(t+1) = w(t) + \Delta w(t) \tag{40}$$

- The weight vector is adjusted in such a way that it is moved towards the input vector to reduce the angle between them
- This adjustment is repeated several times for a given set of input patterns.
- When the weight vector reaches an average position, the weight adjustment will be such that the average movement around the weight vector will be zero:

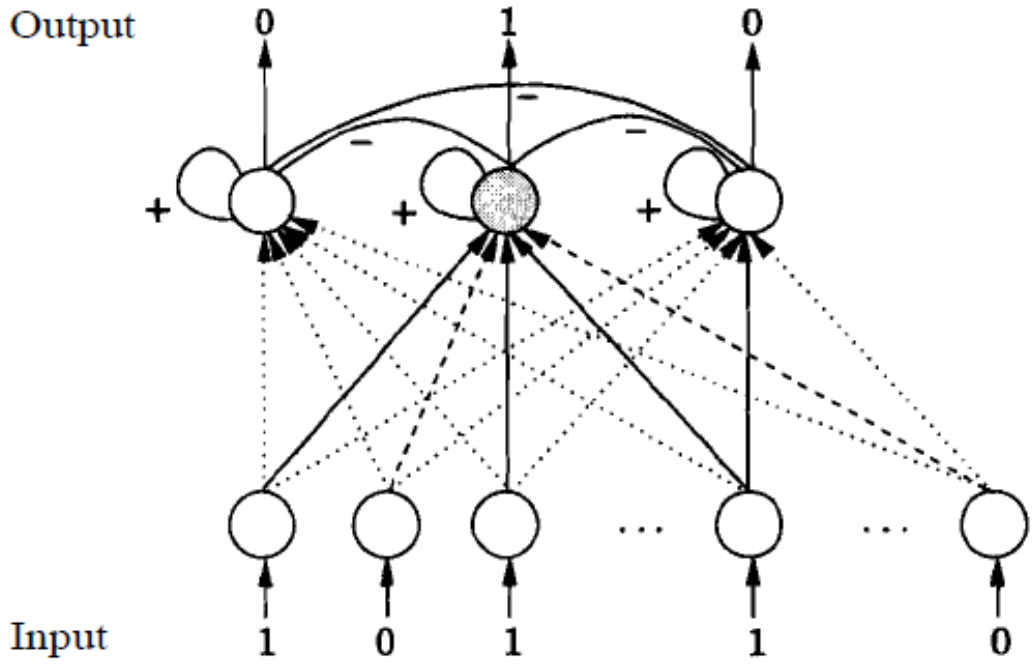
$$\varepsilon[\Delta w] = 0 = \eta(\varepsilon[x] - w) \tag{41}$$

Therefore the weight vector w will be equal to the average of the set of input vectors.

Pattern clustering networks

- The competitive learning networks are used for clustering or grouping of input patterns, they can also be called pattern clustering networks
- In the pattern clustering task, the pattern classes are formed on unlabelled input data, and hence the corresponding learning is unsupervised.
- In the competitive learning the weights in the feedforward path are adjusted only after the winner unit in the feedback layer is identified for a given input pattern.

- There are three different methods of implementing the competitive learning as shown if the below fig



- It is assumed that the input is a binary $(0, 1)$ vector
- The activation of the i^{th} unit in the feedback layer for an input vector $x = (x_1, x_2, \ldots, x_M)^T$ is given by $y_i = \sum_{j=1}^{M} w_{ij} x_j$, where w_{ij} is the $(i, j)^{th}$ element of the weight matrix W, connecting the j^{th} input to the i^{th} unit.

- Let $i = k$ be the unit in the feedback layer such that

$$y_k = \max_i(y_i) \tag{42}$$

so $w_k^T x \geq w_i^T x$, for all i.

- Assume that the weight vectors to all the units are normalized that is $\| w_i \| = 1$ for all i
- The meaning of the above step is that the input vector x is closest to the weight vector w_k among all w_i. That is

$$\| x - w_k \| \leq \| x - w_i \|, \text{ for all } i \tag{43}$$

- Start with some initial random values for the weights.
- The given set of input vectors are applied one by one in a random order.

- For each input the winner unit in the feedback layer is identified, and the weights leading to the unit are adjusted in such a way that the weight vector w_k moves towards the input vector x by a small amount, determined by a learning rate parameter η.
- By doing this we are making that unit to respond more for that input.
- A straight forward implementation of the weight adjustment is

$$\Delta w_{kj} = \eta(x_j - w_{kj}) \tag{44}$$

$$\begin{aligned} w_k(m+1) &= w_k(m) + \Delta w_k(m) \\ &= w_k(m) + \eta(x - w_k(m)) \end{aligned} \tag{45}$$

- This looks more like Hebb's learning with a decay term if the output of the winning unit is assumed to be 1.

- It works best for pre-normalized input vectors.
- This is called the standard competitive learning.
- If the input vectors are not normalized, then they are normalized in the weight adjustment formula as follows:

$$\Delta w_{kj} = \eta \left[\frac{x_j}{\sum_i x_i} - w_{kj} \right] \tag{46}$$

  only for those j for which $x_j = 1$
- In the case of binary input vectors, for the winner unit, only the weights which have nonzero input would be adjusted. That is

$$\Delta w_{kj} = \begin{cases} \eta \left[\frac{x_j}{\sum_i x_i} - w_{kj} \right] & \text{for } x_j = 1 \\ 0 & \text{for } x_j = 0 \end{cases} \tag{47}$$

- Only the connections which have both 1 at either end will be adjusted
- This can be called minimal learning (solid lines in the above fig)
- In the minimal learning there is no automatic normalization of weights after each adjustment. That is
- To overcome this problem, Malsburg suggested the following learning law, in which all the weights leading to the winner unit will be adjusted

$$\Delta w_{kj} = \eta \left[\frac{x_j}{\sum_i x_i} - w_{kj} \right] \text{ for all } j \tag{48}$$

- If a unit wins the competition, then each of its input lines gives up some portion of its weights, and that weight is distributed equally among the active connections for which $x_j = 1$

- This can be called Malsburg learning (dashed lines in the above fig)
- The unit i with an initial weight vector w_i far from any input vector, may never win the competition.
- Since a unit will never learn unless it wins the competition.
- So another method called leaky learning law is proposed (dotted lines in the above fig)
- In this case, the weights leading to the units which do not win also are adjusted for each update as follows:
  if i wins the competition, i.e., $i = k$

$$\Delta w_{ij} = \eta_w \left[\frac{x_j}{\sum_m x_m} - w_{ij} \right] \text{ for all } j \tag{49}$$

if i loses the competition, i.e., $i \neq k$

$$\Delta w_{ij} = \eta_l \left[\frac{x_j}{\sum_m x_m} - w_{ij} \right] \text{ for all } j \tag{50}$$

where η_w and η_l are the learning rate parameters for the winning and losing units, respectively and $(\eta_w >> \eta_l)$.

- In this case the weights of the losing units are also slightly moved for each presentation of an input.

Feature Mapping Network

-

Mid Term Exam Solutions

1. 1.1 What are the differences in the manner of solving problems by human beings and by machines? Illustrate with examples. (5)
   1.2 Describe some attractive features of the biological neural network that make it superior to the most sophisticated Artificial Intelligence computer system for pattern recognition tasks. (5)
2. 2.1 What are the different categories of learning? (5)
   2.2 What is transfer function? Discusses different types of transfer function. (5)

3. 3.1 Illustrates Hebbian learning with binary ($sgn(x)$) and continuous activation functions ($f(x) = 1/(1 + e^{-x})$) of a very simple network. Assume the network with the initial weight vector

$$w = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

needs to be trained using the set of three input vectors as below

$$a_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, \; a_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, \; a_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

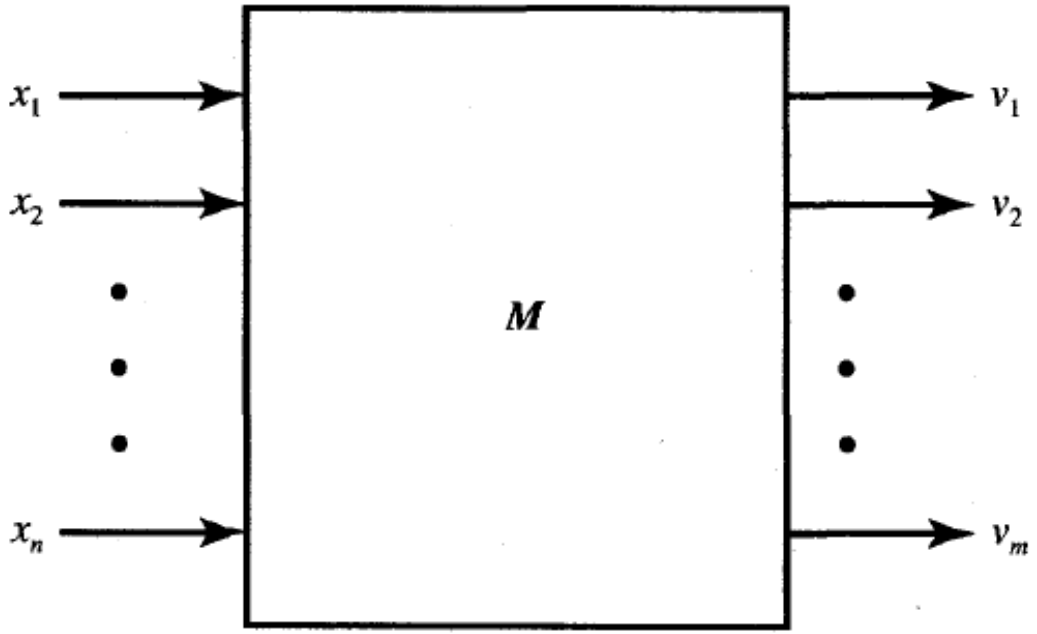
for an arbitrary choice of learning constant $\eta = 1$ (5)

3.2 Explain Hopfield model of a neural network? (5)

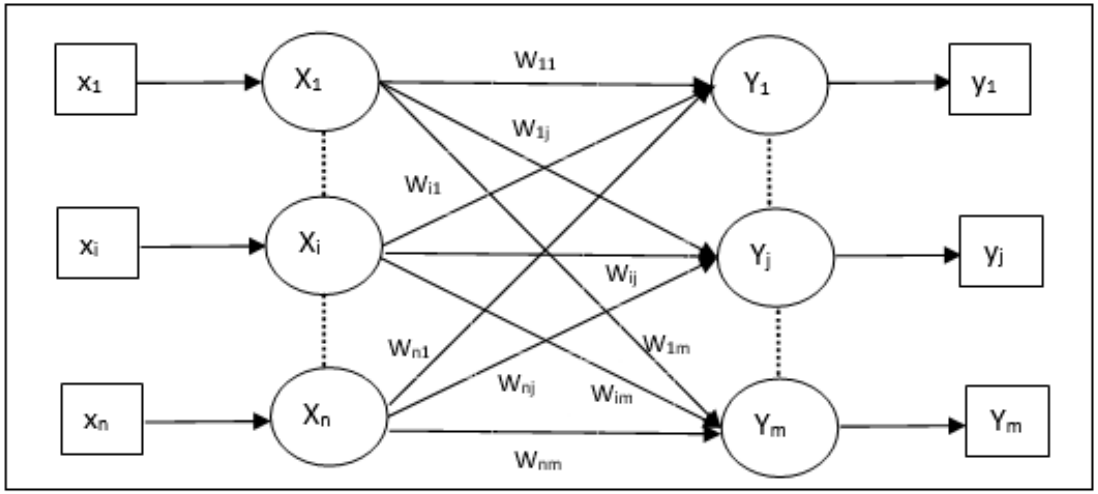
4. 4.1 Discuss the developments in artificial intelligence that led to the interest in exploring new models for computing. (5)
   4.2 Drive the weight values for $x_1 \vee x_2$ for single layer perceptron. (5)

Autoassociative Memory

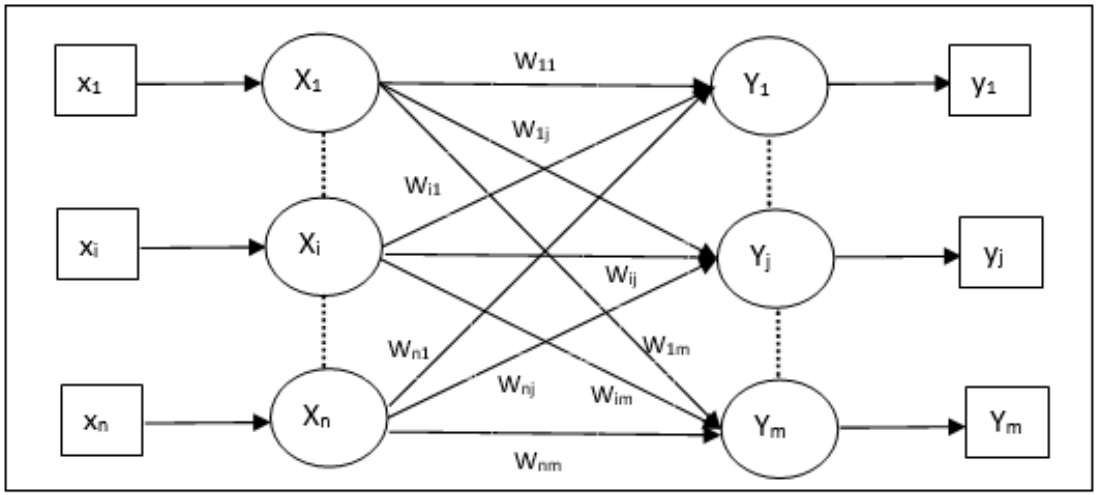
- $v = M(x)$
- Block diagram of an associative memory



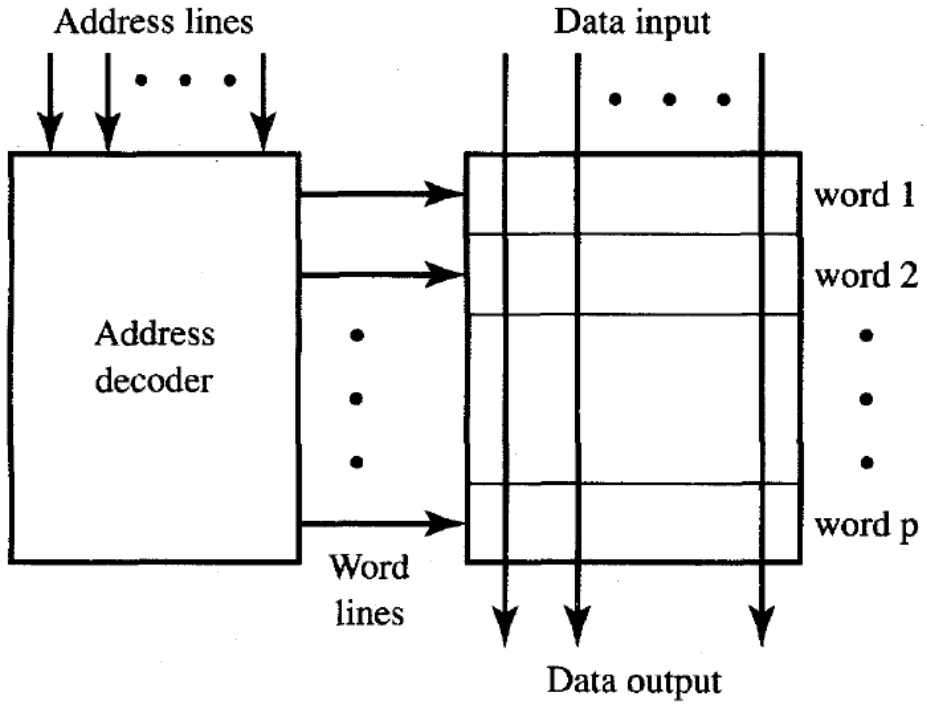
- Autoassociative network

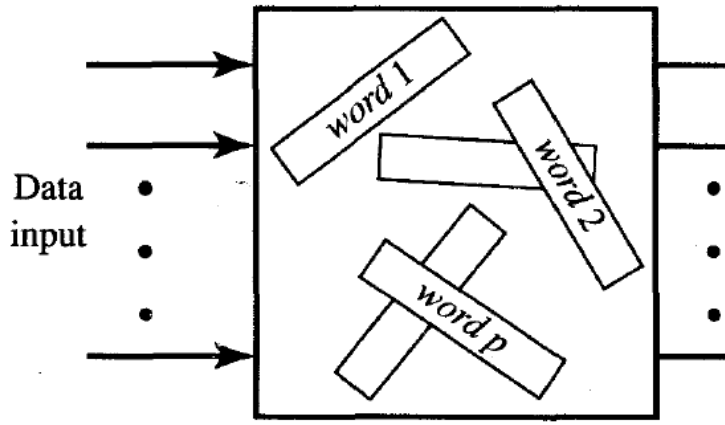


- Heteroassociative network

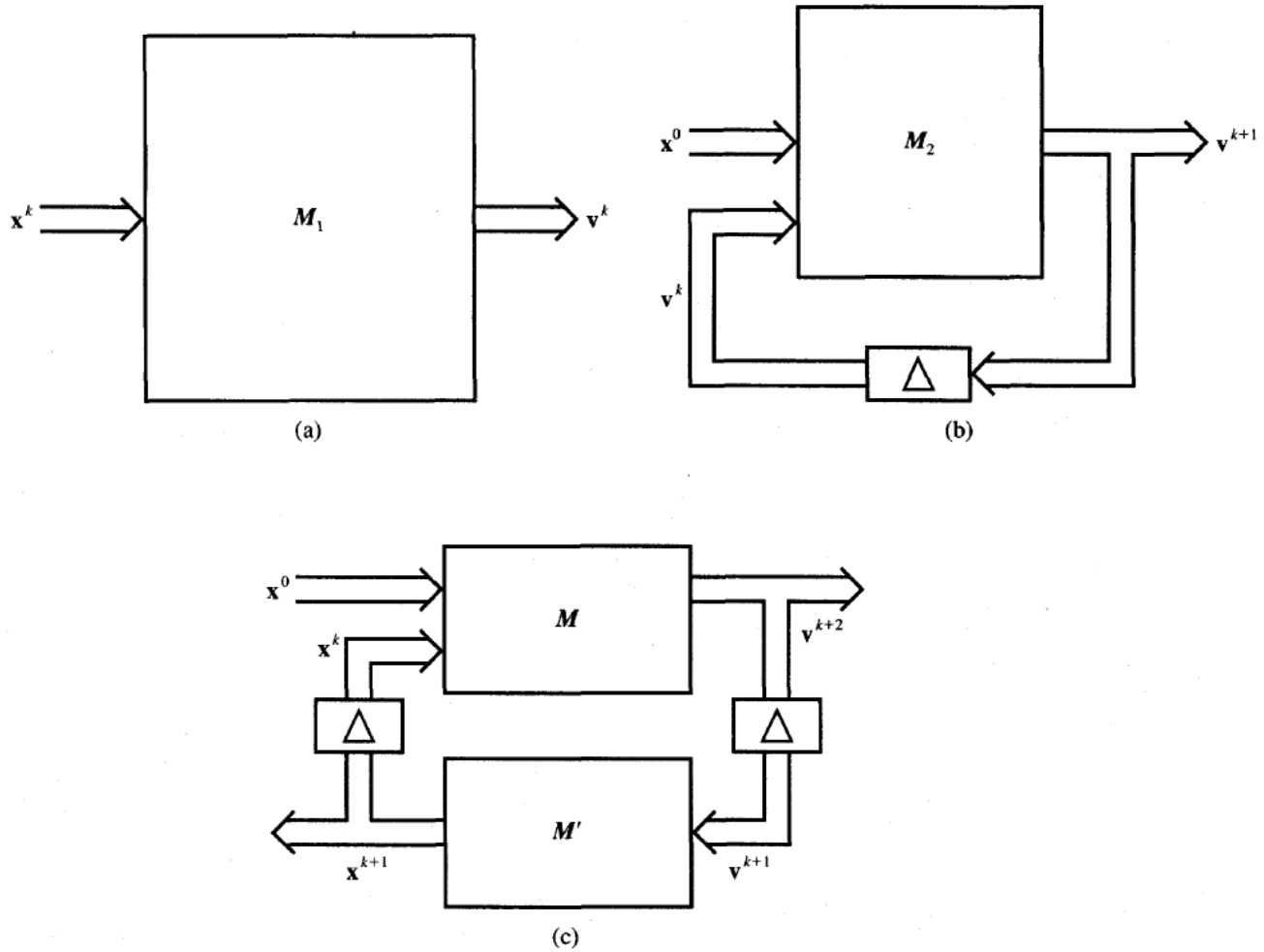


- Addressing modes for memories: (a) address-addressable memory (b) content-addressable memory.

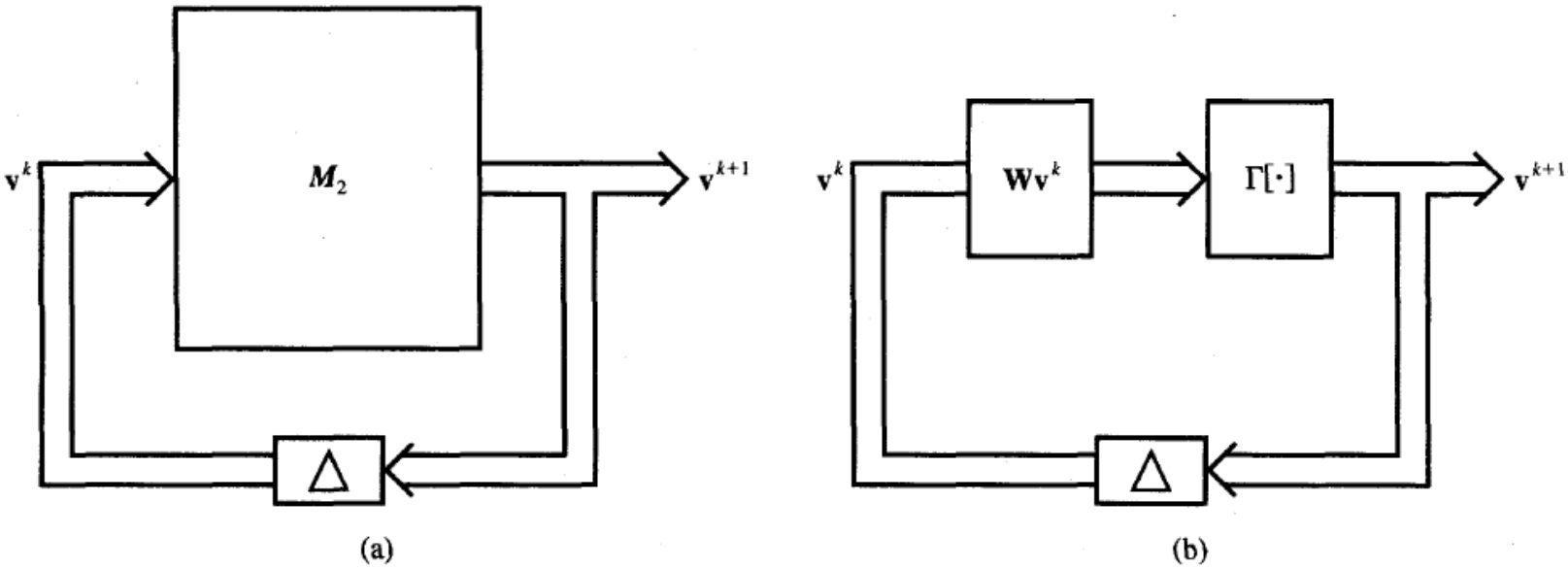




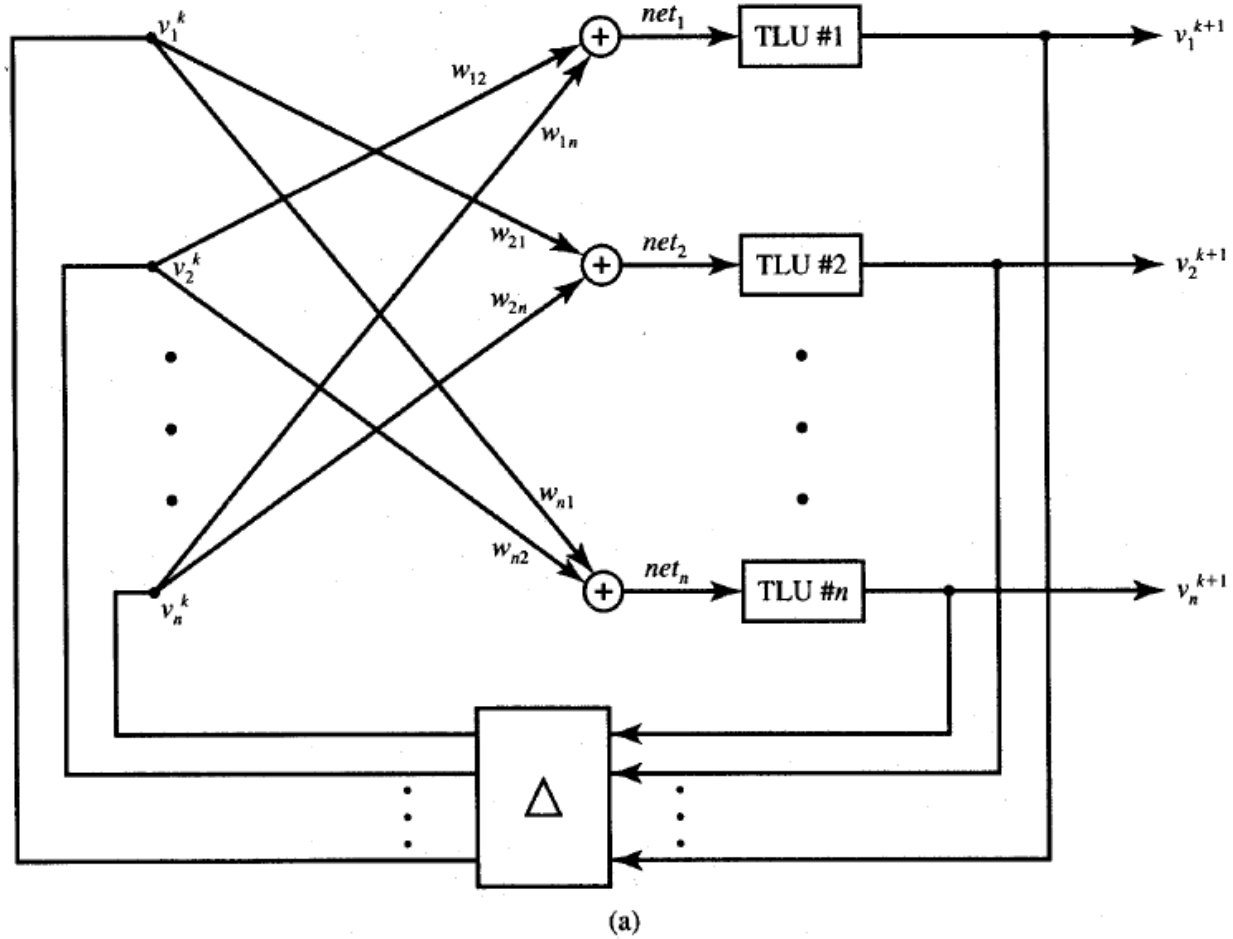
- Block diagram representation of associative memories: (a) feedfotward network, (b) recurrent autoassociative network, and (c) recurrent heteroassociative network



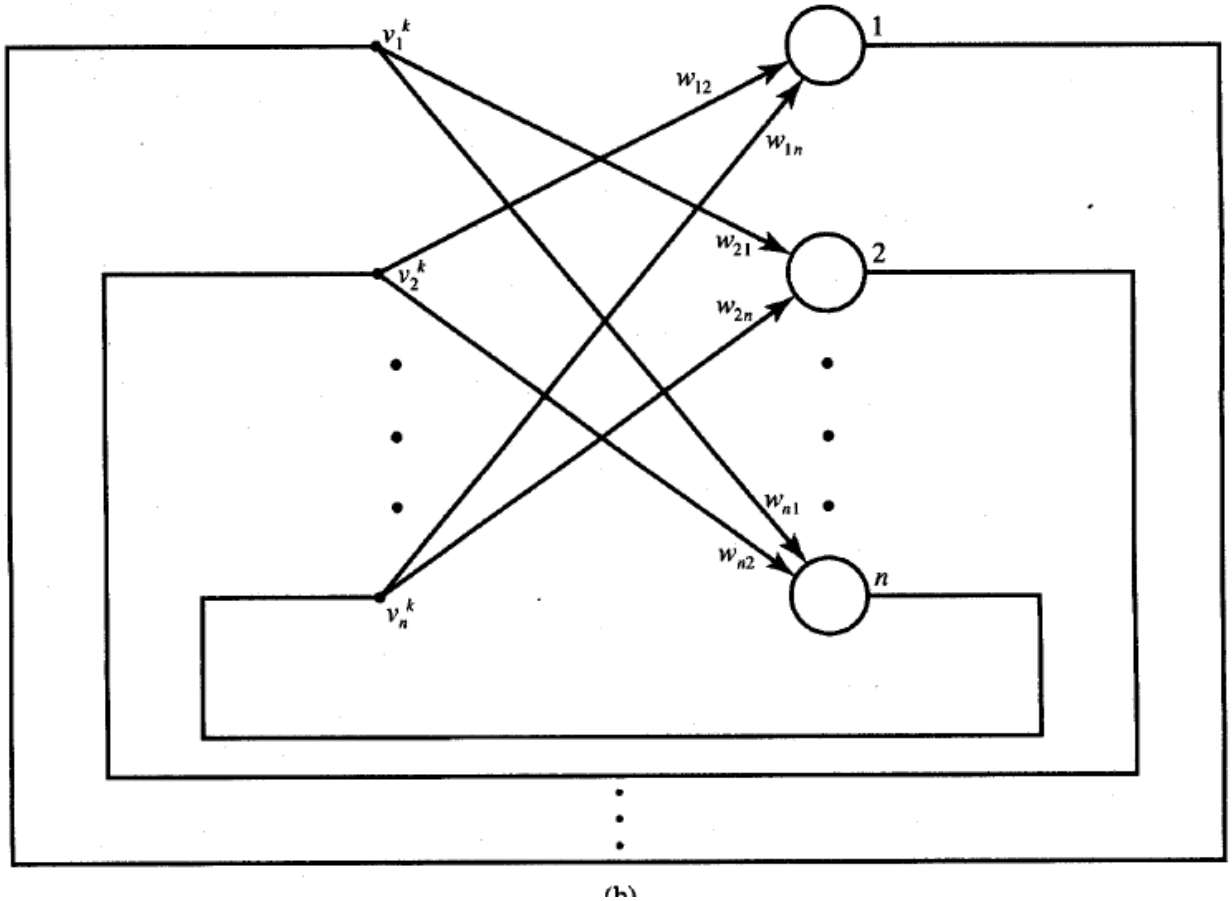
- Autoassociative recurrent memory: (a) block diagram (b) expanded block diagram



- Hopfield model autoassociative memory (recurrent autoassociative memory) (a) expanded view and (b) simplified diagram



(a)



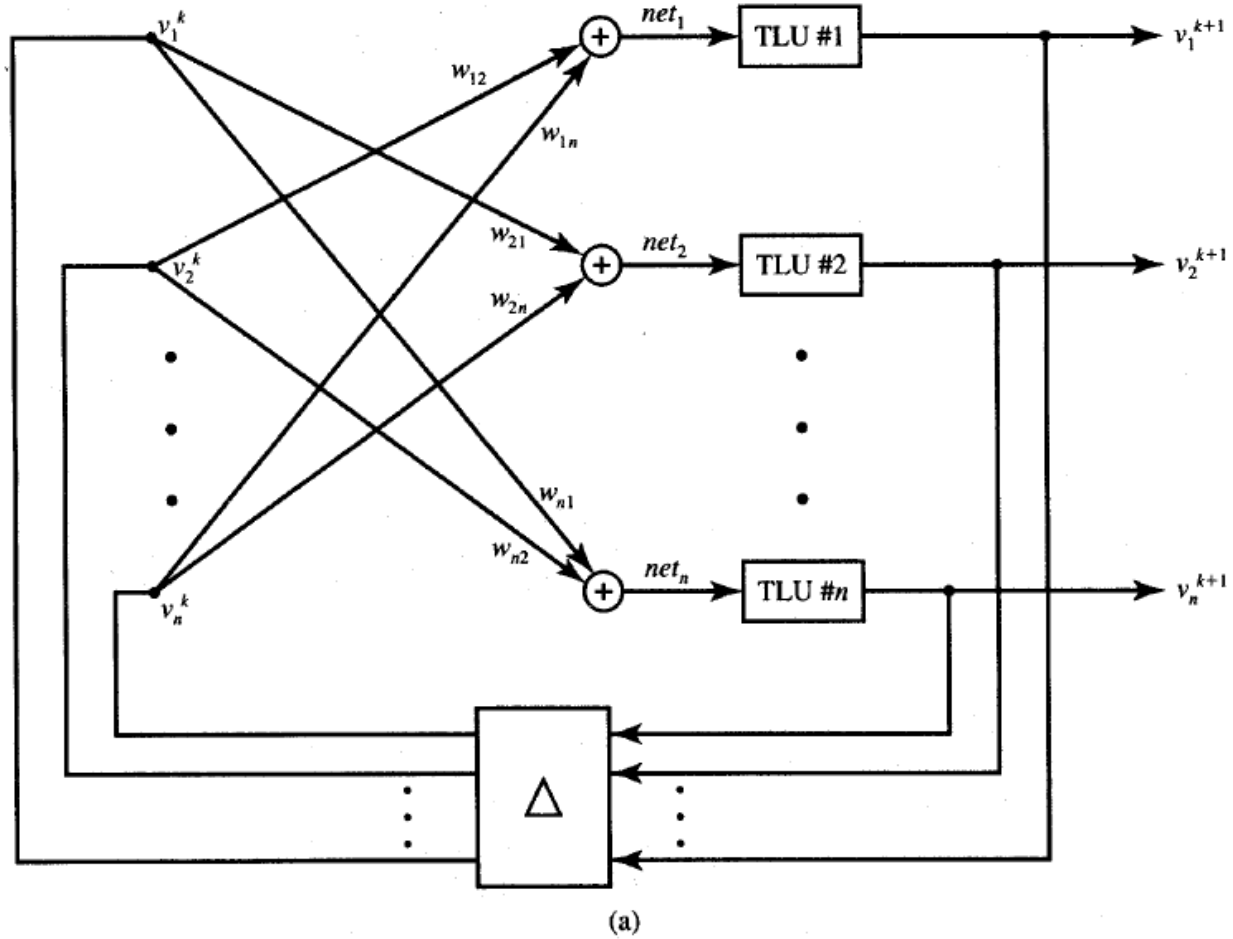
(b)

Analysis of Recurrent Autoassociative Memory

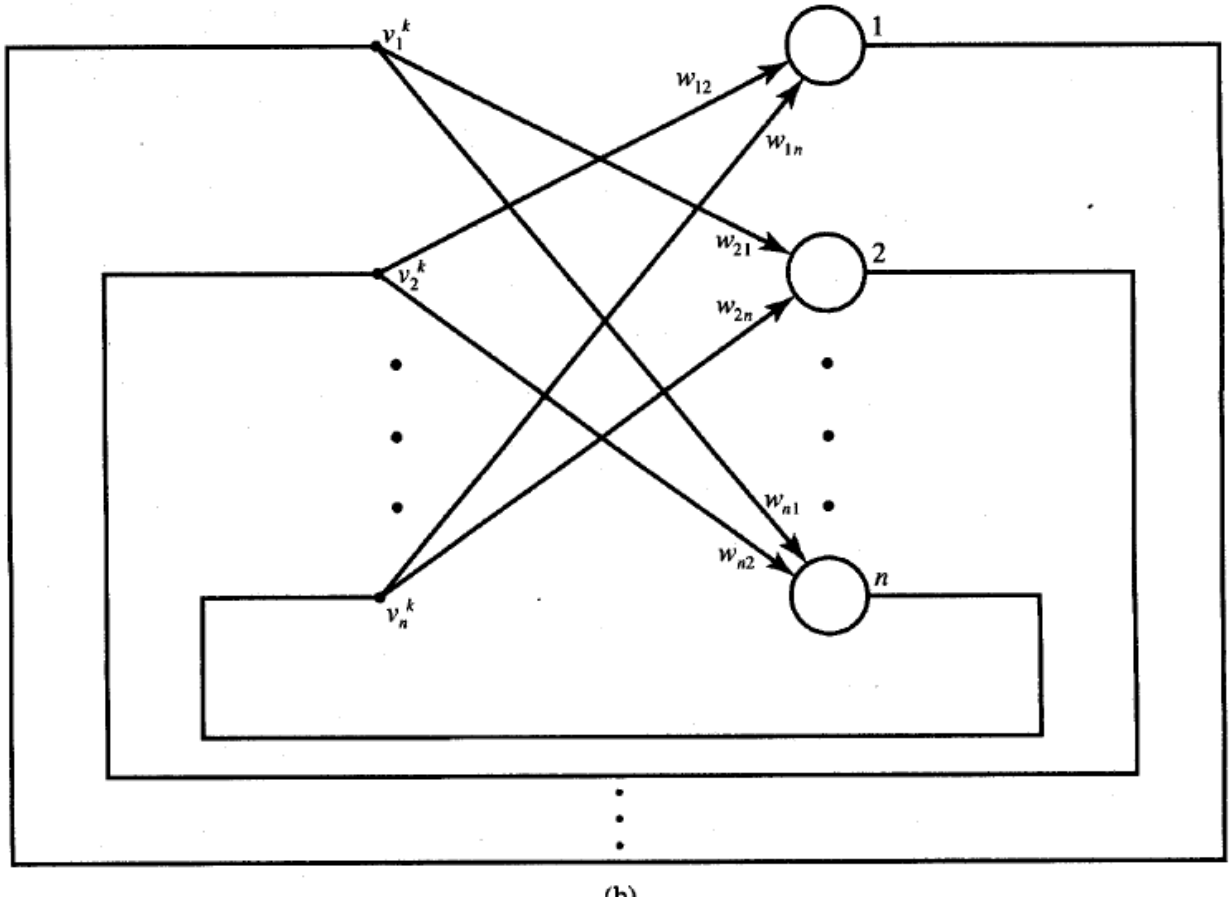
- The repetitive process of recycling of the output followed by a feedforward pass through the network can be performed by a recurrent neural network.
- Such a network is similar to the linear associative memory
- However, it has feedback, nonlinear mapping in the form of thresholding, and is dynamical.
- The memory is essentially a single-layer feedback network as a discrete-time network originally proposed by Hopfield.
- Here we will discuss a more detailed view of the model and its performance as an associative memory
- Our focus is mainly on the dynamical performance of recurrent autoassociative memories.
- In contrast to continuously updating single-layer feedback networks, associative memories are updated in discrete time.

- Under the asynchronous update mode, only one neuron is allowed to compute, or change state, at a time, and then all outputs are delayed by a time Δ produced by the unity delay element in the feedback loop.
- This symbolic delay allows for the time-stepping of the retrieval algorithm embedded in the update rule
- Note that the time step and the neurons' thresholding function have been suppressed on the figure.
- The computing neurons represented in the figure as circular nodes need to perform summation and bipolar thresholding and also need to introduce a unity delay.
- Note that the recurrent autoassociative memories studied here provide node responses of discrete values ± 1.
- The domain of the n-tuple output vectors in R^n are vertices of the n-dimensional cube $[-1, 1]$.

- Hopfield model autoassociative memory (recurrent autoassociative memory) (a) expanded view and (b) simplified diagram



(a)



(b)

Retrieval Algorithm

- Hopfield autoassociative memory can be expressed in the form

$$v_i^{k+1} = sgn\left(\sum_{j=1}^{n} wijv_j^k\right)$$

where k is the index of recursion and i is the number of the neuron currently undergoing an update.

- The update rule has been obtained under the simplifying assumption that both the external bias i_i and threshold values T_i are zero for $i = 1, 2, ..., n$.
- These assumptions will remain valid for the remainder of this chapter.
- The asynchronous update sequence considered here is random.

- Thus, assuming that recursion starts at v^o, and a random sequence of updating neurons $m, p, q, \ldots$ is chosen, the output vectors obtained are as follows

$$v^1 = [v_1^0 v_2^0 \ldots v_m^1 \ldots v_p^0 \ldots v_q^0 \ldots v_n^0]^t$$

$$v^2 = [v_1^0 v_2^0 \ldots v_m^1 \ldots v_p^2 \ldots v_q^0 \ldots v_n^0]^t$$

$$v^3 = [v_1^0 v_2^0 \ldots v_m^1 \ldots v_p^2 \ldots v_q^3 \ldots v_n^0]^t$$

- Considerable insight into the Hopfield autoassociative memory performance can be gained by evaluating its respective energy function.
- The energy function for memory network simplifies to

$$E(v) = -\frac{1}{2} v^t W v$$

- We consider the memory network to evolve in a discrete-time mode, for $k = 1, 2, ...,$ and its outputs are one of the 2^n bipolar binary n-tuple vectors, each representing a vertex of the n-dimensional $[-1, +1]$ cube.
- We also know the fact that the asynchronous recurrent update never increases energy computed for $v = v^k$, and that the network settles in one of the local energy minima located at cube vertices.
- We can now easily observe that the complement of a stored memory is also a stored memory.
- For the bipolar binary notation the complement vector of v is equal to $-v$.
- It is easy to see that

$$E(-v) = -\frac{1}{2} v^t W v$$

- Both energies $E(v)$ and $E(-v)$ are identical.
- Therefore, a minimum of $E(v)$ is of the same value as a minimum of $E(-v)$.
- This provides us with an important conclusion that the memory transitions may terminate as easily at v as at $-v$.
- The crucial factor determining the convergence is the "similarity" between the initializing output vector, and v and $-v$.

Tutorial 8

-

Bidirectional Autoassociative Memory

-

Associative Memory of Spatio-temporal Patterns

-

Radial Basis Function Neural Networks

- Background for Pattern Mapping Networks
- Architecture of Radial Basis Function (RBF) Networks
- Function approximation by radial basis function
- RBF Networks for Function Approximation
- RBF Networks tor Pattern Classification

Background for Pattern Mapping Networks

- The multilayer feedforward neural network with error backpropagation learning was primarily developed to overcome the limitation of a single layer perceptron for classification of hard problems (nonlinearly separable classes)
- To overcome the problem of training a multilayer perceptron (due to hard-limiting output function) for these hard problems.
- The objective of backpropagation network is to capture (in the weights) the complex nonlinear hypersurfaces separating the classes.
- The complexity of the surface is determined by the number of hidden units in the network.
- Strictly speaking, any classification problem specified by the training set of examples can be solved using a network with sufficient number of hidden units.

- In such a case, the problem is more of a pattern association type than of a classification type, with no restrictions on the associated patterns as in the case of a linear associative network
- In a classification problem the input patterns belonging to a class are expected to have some common features which are different for patterns belonging to another class.
- The idea is that, for a pattern belonging to any of the trained classes, the network is supposed to give the correct classification.
- In other words, for a classification problem, the trained neural network is expected to perform some kind of generalization, which is possible only if there are some features common among the input patterns belonging to each class, and these features are captured by the network during training.

- Generalization has no meaning for arbitrary association of one pattern to another as in the case of arbitrary Boolean functions.
- Generalization also has no meaning if the training set consists of all possible input patterns as in the XOR problem.
- Some special association tasks may have common features hidden too deep in the input, like in the parity problem.
- In this case the feature characterizing the similarity of patterns belonging to the same class is not reflected directly in the bit pattern of the input vector.
- For example, 00110000 and 00001111 both belong to the same class of even parity, and 01110000 and 00000111 both belong to the class of odd parity, although the odd parity pattern 01110000 is closer to the even parity pattern 00110000 in the Hamming distance sense.

- In these cases the representation of the input patterns is crucial to achieve generalization by a network.
- Preprocessing of the input vectors is to be performed separately to extract the desired features for feeding the features to a neural network.
- A neural network by itself may not be able to extract automatically the desired features due to limitations of operations it can perform on the input data, and more importantly, due to masking of the desired features by the other undesirable, but dominating features in the input.
- Therefore a trained neural network is expected to exhibit the generalization property for the classification problems in which groups of the input patterns belonging to a class possess some common features within each group.
- Another way of looking at this problem is that there should be some redundancy among the patterns in each group in order to develop a system for classification.

- Note that the class label that is assigned to a group or a collection of groups could be quite arbitrary.
- In other words, the mapping of a group of input patterns to an output or a class label need not have any restrictions.
- Another class of problems deals with capturing the mapping function implied in the given training set of input-output pattern pairs.
- Here the mapping function represents the system that produced the output for a given input for each pair of patterns.
- A trained neural network is expected to capture the system characteristics in their weights.
- The network is supposed to have generalized from the training data, if for a new input the network produces the same output which the system would have produced.

- The generalization of this mapping function in the network can be tested by a set of test input-output pattern pairs.
- Note that in this case the mapping function must exhibit some smoothness or redundancy, as the given training data is usually not adequate to sample the function at all points.
- Note also that the set of input patterns themselves could be quite arbitrary.
- So generalization in these cases is possible only if the mapping function satisfies certain constraints.
- Otherwise, the problem of capturing the mapping function from the training data set will be an ill-posed problem
- Assuming that the constraints are satisfied by the mapping function, they are forced on the approximating function using regularization methods, so that the ill-posed problem becomes well posed.

- Finally, generalization by a network is also possible in situations where both the input patterns in a group and the mapping function have redundancies displayed in the form of common features among the patterns in a group and smoothness in the function, respectively.
- A multilayer perceptron (MLP) architecture is suggested to address arbitrary pattern association tasks which could not be solved by either a linear associative network due to restriction on the type and number of input patterns or by a single layer perceptron due to linear separability constraint on the classification task specified by the input-output mapping.
- A multilayer feedforward neural network (MLFFNN) can be used to realize an approximation to a multilayer perceptron (MLP) for complex (arbitrary) pattern association tasks.

- It is not intended specifically to solve a pattern classification or pattern mapping problem, as both require generalization based on 'closeness' property in classification and 'smoothness' property in mapping, respectively.
- In other words, a MLFFNN trained with backpropagation learning is neither designed to exploit the property of 'closeness' for generalizing a classification task, nor is it designed to exploit the property of 'smoothness' to generalize a function approximation task.
- It is designed mainly to provide discrimination between patterns belonging to different classes.
- Mere manipulation of the structure of a neural network and learning of a MLFFNN are not likely to achieve the generalization required for a given problem.
- Even if the generalization behaviour of a trained MLFFNN is confirmed by cross-validation, it is only an ad hoc solution.

- There is no guarantee of obtaining the desired result.
- This is because, the network is not designed specifically to address the generalization problem.
- Moreover, it is not generally possible to analyze a MLFFNN to understand the task each layer is performing.
- In fact, if the given problem is known to be a classification problem based on the closeness of data in the input vectors, then specific architectures can be evolved to achieve generalization.
- We will consider the tasks of pattern classification and multivariate function approximation (or pattern mapping).
- In both cases the learned network should generalize well, which means that the network should give the correct classification for a new (test) data input in the case of a classification task and a reasonable approximation to the true function value for a new (test) data input in the case of a function approximation task.

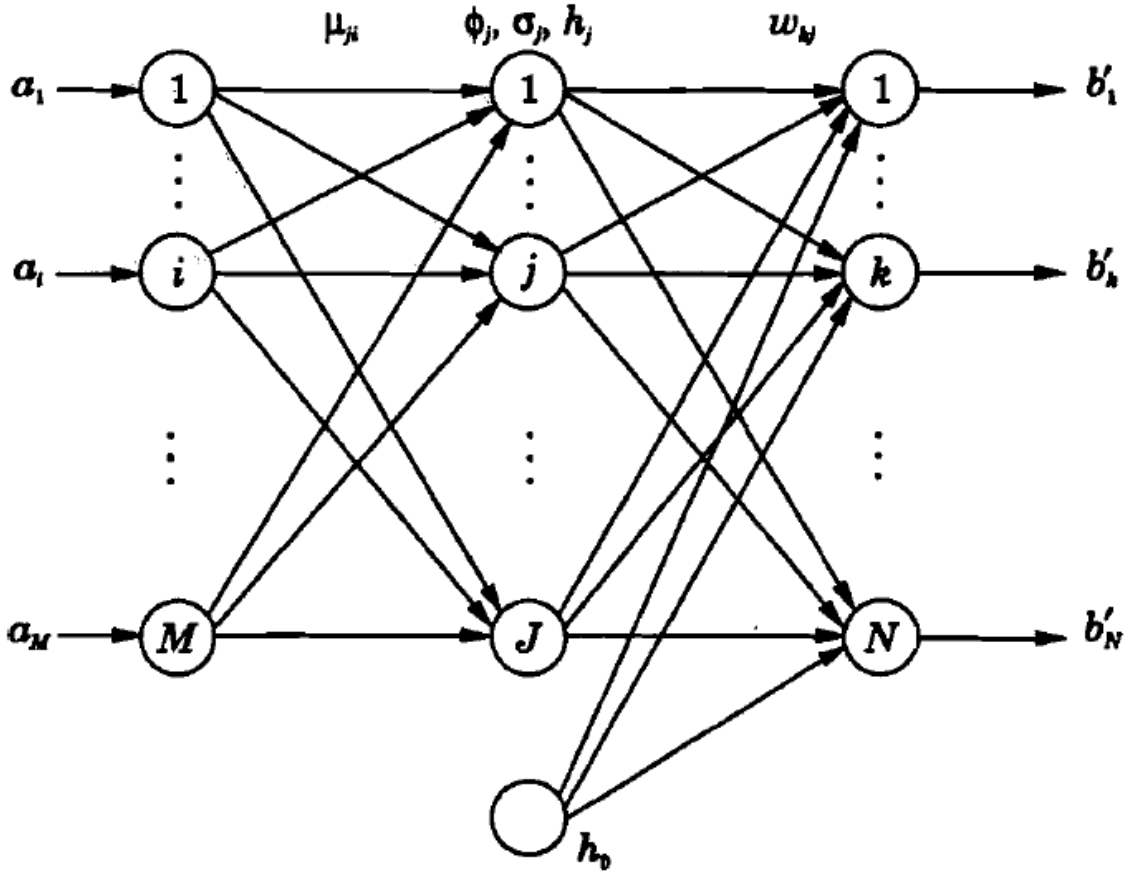
- We will discuss architectures of the Radial Basis Function (RBF) network suitable for these tasks.
- We assume that the 'closeness' property for the classification tasks and the 'smoothness' property for the function approximation tasks are satisfied in the given training set data.

Tutorial 9

-

Architecture of Radial Basis Function Networks

The architecture of a radial basis function network is shown in the below Fig.



- It consists of a single hidden layer with nonlinear units, followed by an output layer with linear units
- The output of the k^{th} unit in the output layer of the network is given by

$$b_k^{'} = \sum_{j=0}^{J} w_{kj} h_j \tag{51}$$

where $h_j = \phi(\| a - \mu_j \| / \sigma_j), j = 1, 2, \ldots, J$ and $h_0 (= -1)$ is the output of the bias unit, so that w_{k0} corresponds to the bias on the k^{th} output unit.

- The nonlinear basis function $\phi_j(.)$ of the j^{th} hidden unit is a function of the normalized radial distance between the input vector $a = (a_1, a_2, \ldots, a_M)^T$ and the weight vector $\mu_j = (\mu_{j1}, \mu_{j2}, \ldots, \mu_{jM})^T$ associated with the unit.
- The normalizing factor σ_j decides the range of influence of the j^{th} unit.

- If the basis function is a Gaussian function of the type $\phi(x) = e^{-\frac{x^2}{2}}$, then the weight vector μ_j, corresponds to the mean value of the function and σ_j corresponds to the standard deviation.
- For a multidimensional vector x with zero mean, the Gaussian function is given by $\phi(x) = e^{-x^T R^{-1} x}$, where R is the covariance matrix of the input vectors, namely, expectation of xx^T.
- The significance of the radial basis function is that the output of the unit is a function of the radial distance, i.e., $h_j = \phi_j(\| a - \mu_j \| / \sigma_j)$.
- On the other hand, in a multilayer feedforward neural network, $h_j = \phi_j(a^T w_j)$, i.e., the output is a nonlinear function of the scalar product of the input vector and the weight vector.

Function approximation by radial basis function

- Let A be a compact subset of R^M and $F(x)$ be a continuous function on A.
- Then for any $\varepsilon > 0$, there exist an integer N and parameters w_i and ε_i such that $|\hat{F}(x) - F(x)| < \varepsilon$ for all $x \in A$
- Where w_i are M-dimensional parameter vectors corresponding to the centroids of clusters, so that

$$\hat{F}(x) = \hat{F}(x_1, x_2, \ldots, x_M) = \sum_{i=0}^{N} \varepsilon_i g(\| x - w_i \|) \tag{52}$$

where $g(.)$ is a nonlinear function with unique maximum centered at w_i.

- The argument of the function $g(.)$ forms the basis of the function.

- Since the argument is the radial distance between the variable vector x from the centroid vector w_i, the function $g(.)$ is called the radial basis function.
- The function approximation itself is called the radial basis function approximation.
- Gaussian function is one of the commonly used nonlinear functions for $g(.)$
- This show the existence of the parameters to approximate a given function.
- In the context of neural networks a feedforward network with a single hidden layer with nonlinear units can approximate any arbitrary function.
- But this do not suggest any method of determining the parameters, such as the number of hidden units and weights in order to achieve a given accuracy for the approximation of the function

RBF Networks for Function Approximation

- In the RBF networks the weights (μ_j, σ_j) of the hidden layer units are determined directly from the data. No learning is involved.
- The weights w_k of the output layer are determined by supervised learning
- For function approximation task the error to be minimized in the supervised learning is given by

$$E(F) = E_D(F) + \lambda E_R(F) \tag{53}$$

where E_D is the error due to the given data in the form of input output pair (a_l, b_l),
E_R is the contribution due to regularization, and λ is the regularization parameter.

- The error term E_D is given by

$$E_D(F) = \frac{1}{2}\sum_{l=1}^{L}\sum_{k=1}^{N}(b_{lk} - b'_{lk})^2 \tag{54}$$

- Note that b'_{ik} is a function of the input data and the parameters of the network, i.e., $b'_{ik} = F(a_l, w_k)$
- The regularization term E_R depends on the prior knowledge of the function or some global knowledge derived from the given data.
- This knowledge is normally represented in the form of a smoothness constraint on the mapping function between the input and the output.

- In one form of smoothing, the weights of the network are constrained using the following expression for the regularization term

$$E_R(F) = \frac{1}{2} \sum_j \sum_k w_{kj}^2 \tag{55}$$

- Inclusion of this term favours small values of the weights.
- It can also be viewed as a 'weight decay' term, since the weight change is proportional to the negative gradient of the error and the negative gradient of $E_R(F)$ gives a decay term in the learning equation.

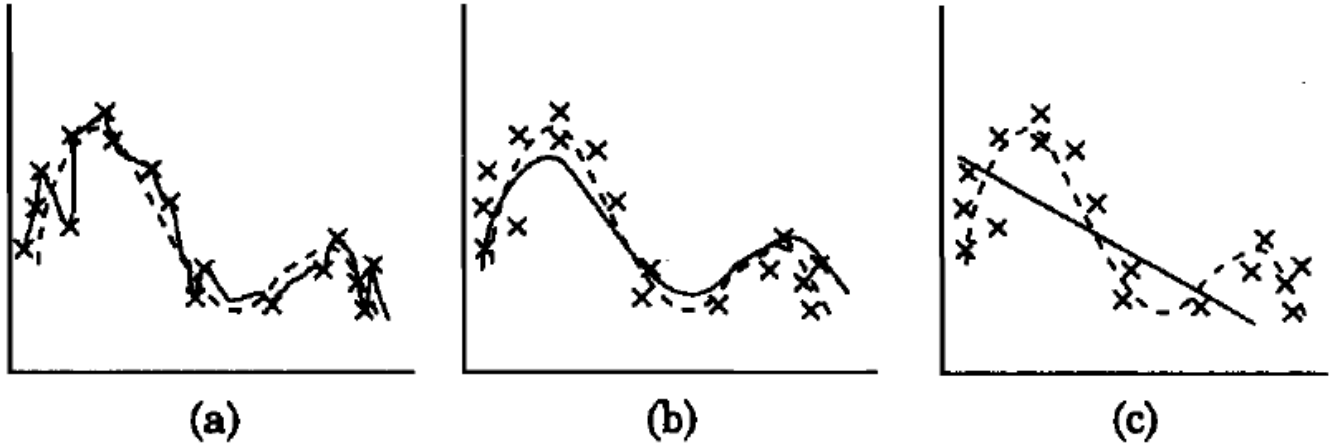
- Smoothing constraint is also expressed in the form of the square of the derivatives of the mapping function:

$$E_R(F) = \frac{1}{2} \parallel PF \parallel^2 \tag{56}$$

where P is a linear differential operator, and $\parallel . \parallel$ is the $L2$ norm. For example, P could be a simple second derivative of F with respect to w.

- Then minimization of the square of the derivative restricts the discontinuous jumps in the function.

- Function approximation for different values of regularization parameter λ: (a) λ too small, (b) λ near optimal and (c) λ too large. 'x' indicates actual points. Dashed line is the desired function. Solid line is the realized function.



- The regularization problem is to find the function $F(.)$ that minimizes the error given by Eq. 53, where the second term due to regularization is a constraint on the desired level of smoothness of the function.
- The smoothness constraint is usually in the form of a differential operator as we have stated before.

- Let us consider one dimensional case, i.e., the function $F(.)$ is a scalar function and hence there is only one output unit for the network. Then

$$E_D(F) = \frac{1}{2}\sum_{l=1}^{L}[d_l - F(a)]^2 \tag{57}$$

We have $E_R(F) = \frac{1}{2} \parallel PF \parallel^2$ where P is a linear differential operator.

- The minimization problem reduces to solving the following differential equation

$$P^*PF = \frac{1}{\lambda}\sum_{l=1}^{L}[d_l - F(a)]\delta(a - a_l) \tag{58}$$

where P^* is the adjoint of the differential operator P and $\delta(.)$ is a delta function

- Let G denote the Green's function for the operator P^*P, so that

$$P^*PG(a : a_l) = \delta(a - a_l) \tag{59}$$

- Then the solution of the regularization problem is given by

$$F(a) = \sum_{l=1}^{L} w_l G(a : a_l) \tag{60}$$

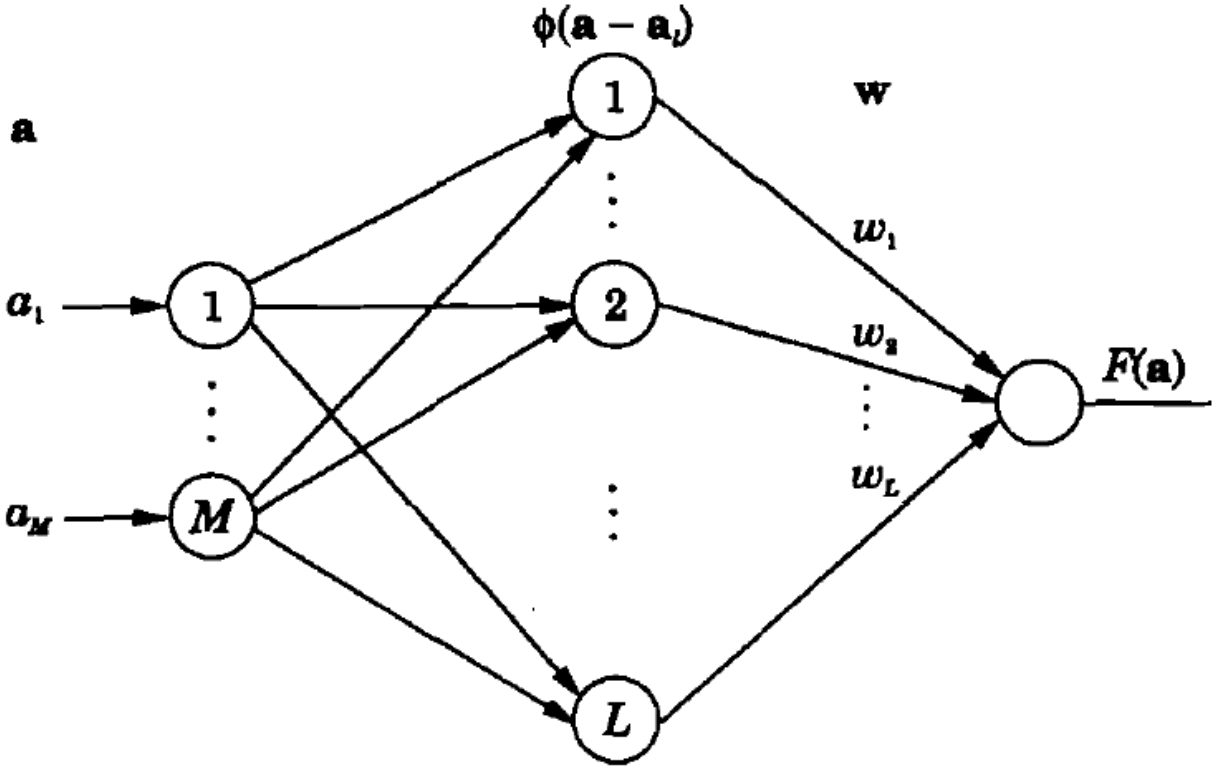
where

$$w_l = \frac{1}{\lambda}[d_l - F(a_l)] \tag{61}$$

G is the Green's function for the l^{th} input pattern.

- The Green's function is a result of the smoothness constraint expressed in the form of a differential operator.

- If the differential operator P^*P is invariant to rotation and translation, then the Green's function is a function of the magnitude of the difference of its arguments, i.e., $G(a : a_I) = G(\| a - a_I \|)$.
- In such a case G is called a radial basis function.
- For some special cases of the differential operator P, the radial basis function becomes a multivariate Gaussian function.
- Thus the solution of the regularization problem leads to a radial basis function as shown in below Fig for I-D case, where the Green's function is shown by $\phi(.)$



- The weights are determined by minimizing the error E in Eq. 53 which consists of the squared error between the desired and actual output values, and the regularization term, the extent of which is determined by the parameter h.

- In the above formulation the number of hidden units is equal to the number of training samples (L) and the centres of the basis function are located at the sample values a_l.
- A suboptimal solution to the function approximation is obtained using fewer basis functions. That is, using the radial basis functions, the function F is given by

$$F(a) = \sum_{i=1}^{H} w_i \phi(\| a - \mu_i \|) \tag{62}$$

where $H < L$, and μ_i are the centres of the basis functions to be determined from the given data. The weights of the output units and the centres of the radial basis function can be determined by computation using all the training set data.

- For a 1-D output function, the desired output d_l is a scalar for the given input vector a_l. That is,

$$d_l = F(a_l) \tag{63}$$

- Minimizing the error in Eq. 53 without regularization ($h = 0$), the optimum weight vector $w = [w_1, w_2, \ldots, w_H]^T$ is given by

$$w = \Phi^+ d \tag{64}$$

where $d = [d_l, d_2, \ldots, d_L]^T$, and Φ^+ is the pseudoinverse of the matrix

$$\Phi = \begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1H} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2H} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{L1} & \phi_{L2} & \cdots & \phi_{LH} \end{bmatrix} \tag{65}$$

- The elements of the matrix are given by

$$\phi_{ji} = \phi(\| a_j - \mu_i \|), i = 1, 2, \ldots, H, j = 1, 2, \ldots, L \tag{66}$$

- The pseudoinverse of Φ^+ is given by

$$\Phi^+ = (\Phi^T \Phi)^{-1} \Phi^T \tag{67}$$

- If regularization ($h \neq 0$) is used in the error function, then the corresponding expression for the optimal weight vector is given by

$$w = (\Phi^T \Phi + h\Phi_0)^{-1} \Phi^T d \tag{68}$$

where Φ_0 s a $H \times H$ matrix whose ji^{th} element is $\phi(\| \mu_j - \mu_i \|)$.

- The weights can also be determined by using supervised learning methods like LMS algorithm, in which the weight adjustment at each stage is given by

$$\Delta w_i = -\eta[d_i - f(a_i)] \tag{69}$$

where η is the learning rate parameter

RBF Networks tor Pattern Classification

The pattern classification problem can be posed as follows:

- Given an input pattern a, determine the class label C_i such that the posteriori probability $P(C_i|a)$ of the class C_i is maximum among all classes.
- This probability can be computed using the probabilities $p(a|C_i)$ and and $p(a)$,

$$P(C_i|a) = \frac{p(a|C_i)P(C_i)}{p(a)} \tag{70}$$

where $p(a|C_i)$ gives the probability distribution of the data generated for the class C_i and $p(a)$ is the probability distribution of the data vector irrespective of the class.

- These probability distributions can be expressed in terms of a linear combination of some standard distribution $\Psi(a)$, say Gaussian, in order to reflect the possible multi-modal nature of the distribution of the data a belonging to any class.
- These mixture distributions can be written as

$$P(a) = \sum_k \alpha_k \Psi_k(a) \tag{71}$$

$$P(a|C_i) = \sum_j \beta_j^i \Psi_j(a)$$

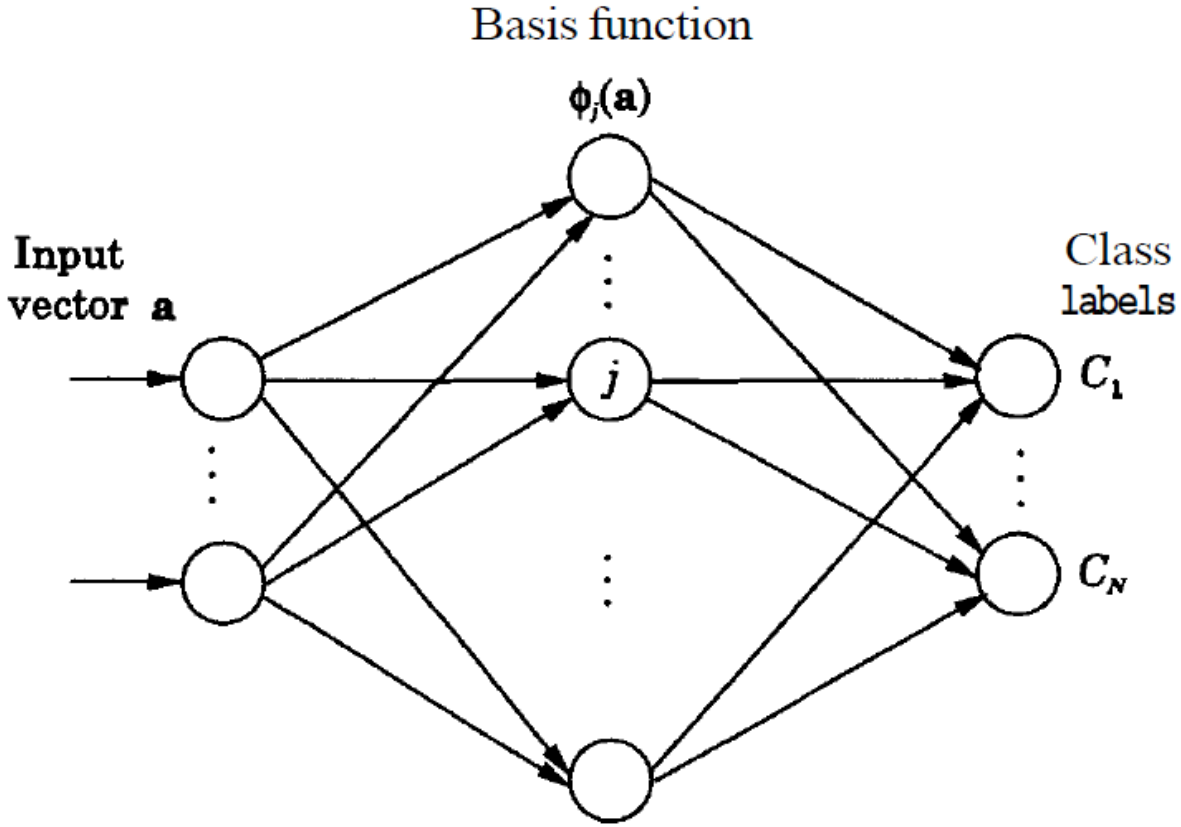
where α_k and β_j^i are coefficients of the standard distribution functions.

- So $P(C_i|a)$ can be written as:

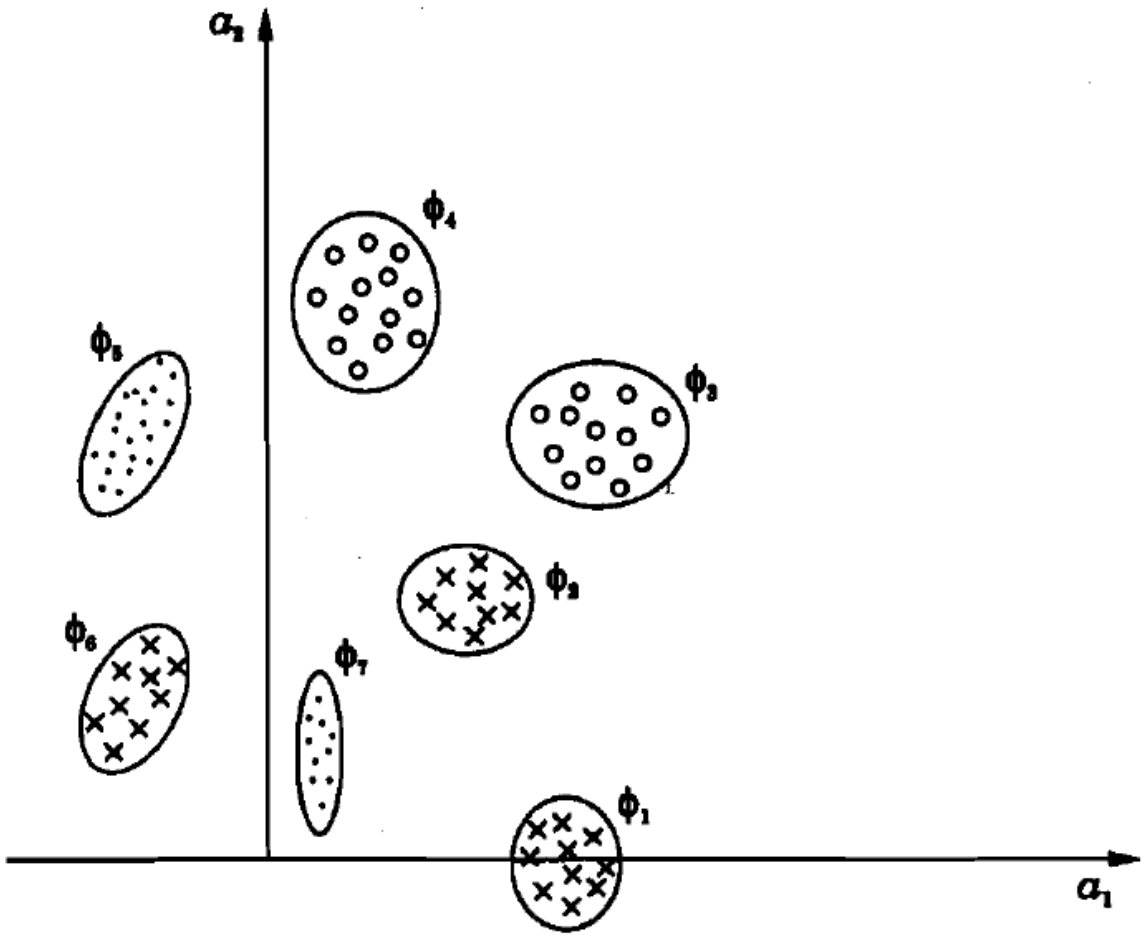
$$P(C_i|a) = \frac{p(a|C_i)P(C_i)}{p(a)} \tag{72}$$
$$= \frac{\sum_j \beta_j^i \Psi_j(a) P(C_i)}{\sum_k \alpha_k \Psi_k(a)}$$
$$= \sum_j P(C_i) \frac{\beta_j^i \Psi_j(a)}{\sum_k \alpha_k \Psi_k(a)}$$
$$= \sum_j \frac{P(C_i)\beta_j^i}{\alpha_j} \frac{\alpha_j \Psi_j(a)}{\sum_k \alpha_k \Psi_k(a)}$$
$$= \sum_j w_{ij} \phi_j(a)$$

where $\phi_j(a)$ is the normalized basis function and $w_{ij} = \frac{P(C_i)\beta_j^i}{\alpha_j}$ is the contribution of the output of the basis function to the output unit corresponding to the i^{th} class C_i.

- So the expression for $P(C_i|a)$ can be viewed as a basis function formulation of the classification problem, and the corresponding radial basis function network is shown in Fig



- To illustrate the steps involved in a pattern classification task, let us consider the 2-D cluster points given in the below Fig where each class is indicated by a separate symbol like 'x','o ','.' etc.



- The first step is to determine the clusters using any standard clustering algorithm or by any of the unsupervised learning methods
- The number of clusters can be fixed a priori, or a criterion may be used to determine the optimum number of clusters
- Then a basis function is derived for each cluster.
- If a 2-D Gaussian function is assumed, then the mean and covariance matrix are derived for each cluster to represent the corresponding Gaussian basis function $\phi(a)$ as:

$$\phi(a) = \frac{1}{\sqrt{(2\pi)^M |R|}} e^{(a-\mu)^T R^{-1}(a-\mu)} \tag{73}$$

where μ is the mean vector of the cluster points and R is the covariance matrix of the cluster

- The ij^{th} element of the matrix R is given by

$$R_{ij} = \sum_{l}(a_{li} - \mu_i)(a_{lj} - \mu_j), i, j = 1, 2 \tag{74}$$

where i is the index for the sample pattern in the cluster, and $\mu_i = \bar{a}_i$ and $\mu_j = \bar{a}_j$ are the mean values of the i^{th} and j^{th} components of the input vectors and are given by $\bar{a}_i = \sum_l a_{li}$, for all i.

- For the k^{th} cluster, $\phi = \phi_k$ and the mean vector and covariance matrix can be indicated by $\mu = \mu^k$ and $R = R^k$ respectively.
- Using the training data set, which consists of a set of input vectors and the desired class labels, the output weights w_{ij} can be determined by any of the following methods:

1. **Determination of weights by matrix inversion:**
   For the l^{th} pattern pair, the error between the desired and actual outputs is given by

$$E_l = \sum_{i=1}^{N} \left[d_{li} - \sum_{j=1}^{H} w_{ij} \phi_j(a_l) \right]^2 \tag{75}$$

   The total error $E = \sum_l E_l$ is minimized to determine the optimum weight matrix. This requires computation of pseudo-inverse of a matrix and uses all the training data in the computation of the matrix

2. **Determination of weights by learning using LMS algorithm:**
   Since the output units are assumed linear, the instantaneous error can be used to adjust the weights as in the LMS algorithm. That is

$$\Delta w_{ij} = -\eta \left[d_{li} - \sum_{j=1}^{H} w_{ij} \phi_j(a_l) \right] \tag{76}$$

   where η is the learning rate parameter.

- For the optimum weight matrix, the network output $F_i(a)$ from the i^{th} unit is an approximation to the conditional expectation $\varepsilon(d_i|a)$ in the mean squared error minimizing sense.
- The conditional expectation $\varepsilon(d_i|a)$ in turn is equal to the probability $P(C_i|a)$

- So a trained radial basis function for classification gives as output the *a* posteriori probabilities $P(C_i|a), i = 1, 2, \ldots, N$ for a given input vector.
- The class C_k for which $P(C_i|a)$ is maximum for all i is the class to which the input vector a belongs.

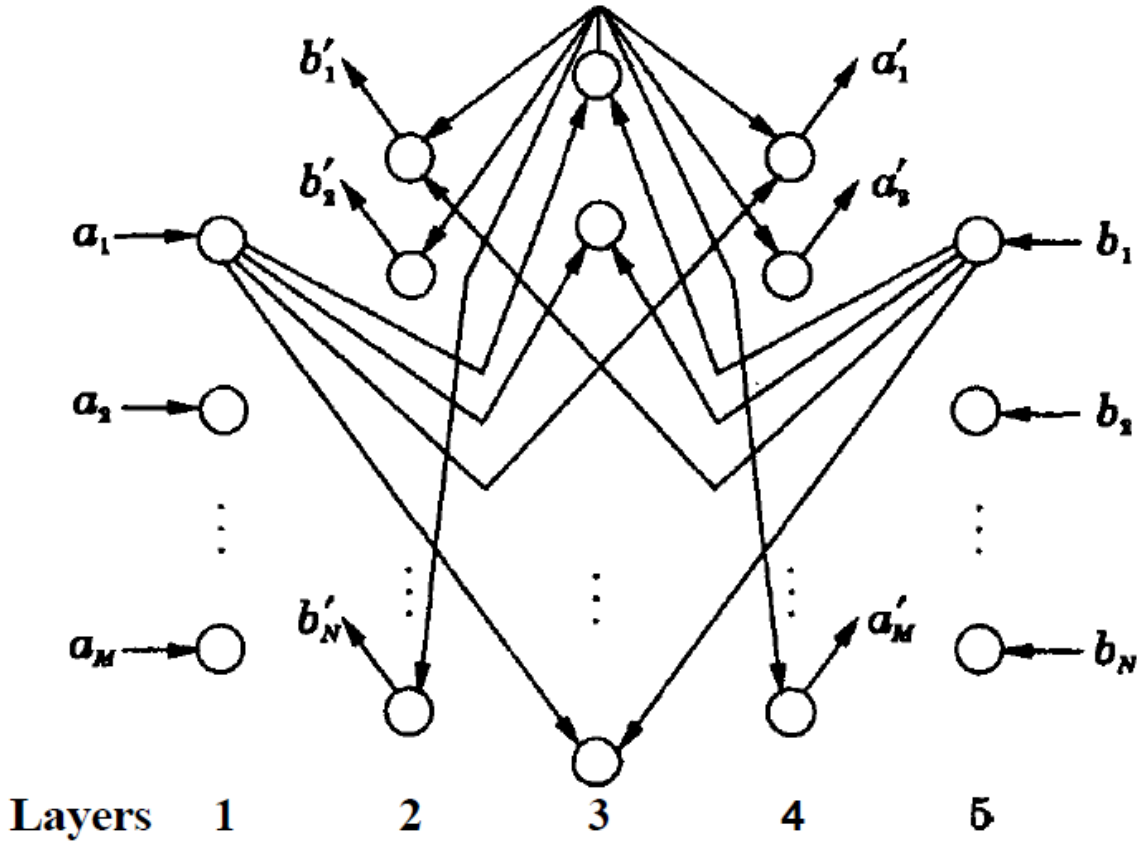
Tutorial 10

1. Why a general MLFFNN is not likely to generalize a problem always?
2. Explain the concept of 'closeness' of data and 'smoothness' of a mapping function.
3. Explain why is it that an MLFFNN does not take closeness of data into account.
4. Give the architecture of a basis function network.
5. What is the significance of the regularization term in the cost function for a function approximation problem?

Counter Propagation Networks

- In a multilayer feedforward neural network the training process is slow, and its ability to generalize a pattern mapping task depends on the learning rate and the number of units in the hidden layer.
- In the use of radial basis functions the unsupervised part of the learning involves determination of the local receptive field centres and the spread in the input data corresponding to each hidden unit.
- The centres are determined using a vector quantization approach.
- This could be done either by computation or by learning from the input data.

- On the other hand, Counter propagation network uses winner-take-all instar learning for the weights from the units in the input layer to the units in the hidden layer.
- The counter propagation network (CPN) provides a practical approach for implementing a pattern mapping task, since learning is fast in this network



- The network consists of two feedforward networks with a common hidden layer as shown the above Fig.

- The feedforward network formed by layers 1,3 and 2 is used for forward mapping and the network formed by layers 5, 3 and 4 is used for inverse mapping (if it exists) between the given input-output pattern pairs.
- Each feedforward network uses a combination of instar and outstar topologies.
- The first and second (hidden) layers of a feedforward network form a competitive learning system and the second (hidden) and third layers form an outstar structure.
- Learning takes place in the instar structure of the competitive learning system to code the input patterns a_l and in the outstar structure to represent the output patterns b_l.
- The training of the instar and outstar structures are as follows:

Training instars of CPN

1. Select an input vector a_l from the given training set $(a_l, b_l), l = 1, 2, \ldots, L$.
2. Normalize the input vector and apply it to the CPN competitive layer.
3. Determine the unit that wins the competition by determining the unit k whose vector w_k is closest to the given input.
4. Update the winning unit's weight vector as

$$w_k(m+1) = w_k(m) - \eta(w_k(m) - a_l)$$

5. Repeat Steps 1 through 4 until all input vectors are grouped properly by applying the training vectors several times.

After successful training, the weight vector leading to each hidden unit represents the average of the input vectors corresponding to the group represented by the unit

Training outstars of CPN

1. After training the instars apply a normalized input vector a_l to the input layer and the corresponding desired output vector b_l to the output layer.
2. Determine the winning unit k in the competitive layer.
3. Update the weights on the connections from the winning competitive unit to the output units

$$v_k(m+1) = v_k(m) - \eta(v_k(m) - b_l)$$

4. Repeat Steps 1 through 3 until all the vector pairs in the training data are mapped satisfactorily.

After successful training, the outstar weight vector for each unit in the hidden competitive layer represents the average of the subset of the output vectors corresponding to the input vectors belonging to that unit.

- Depending on the number of units in, the hidden layer, the network can perform any desired mapping function.
- In the extreme case, if a unit is provided in the hidden layer for each input pattern, then any arbitrary mapping (a_l, b_l) can be realized.
- But in such a case the network fails to generalize and it merely stores the pattern pair.
- By using a small number of units in the hidden layer, the network can accomplish data compression.
- The network can also be trained to capture the inverse mapping as well, i.e., $a_l = \phi^{-1}(b_l)$, provided such a mapping exists and it is unique.
- The name counter propagation is given to this architecture due to the network's ability to learn both forward and inverse mapping functions

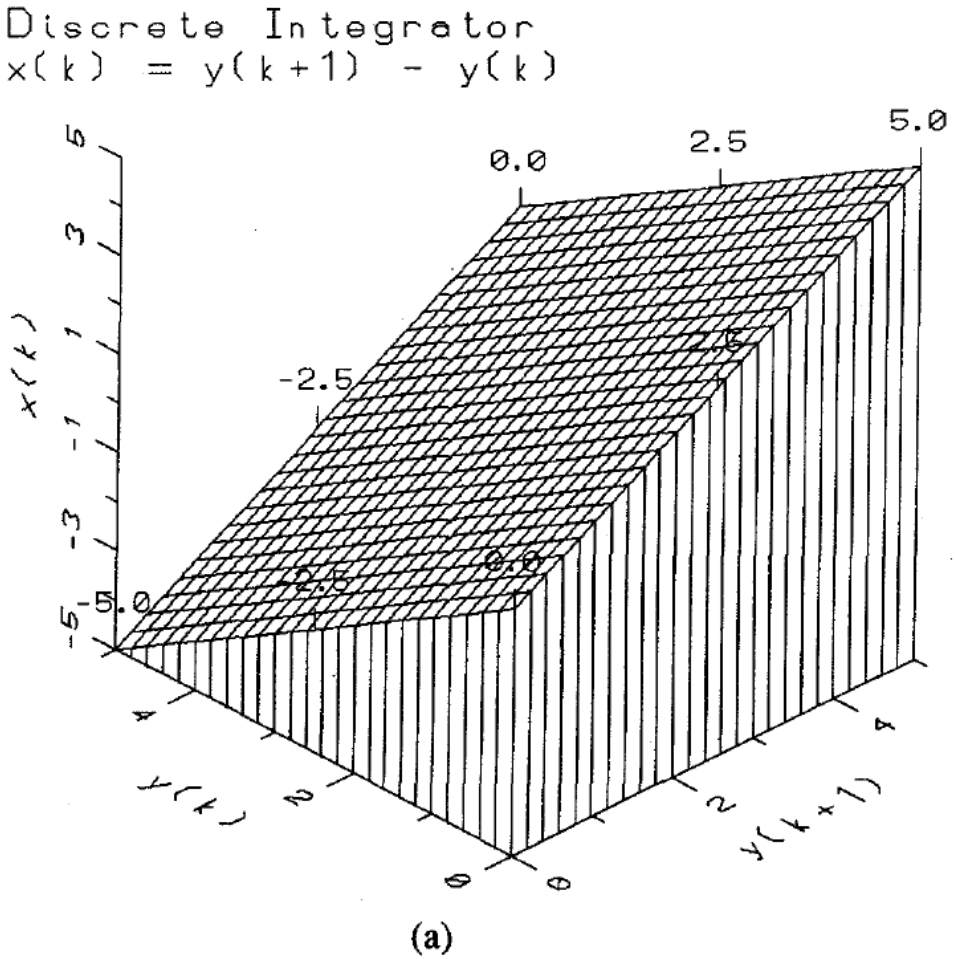
Cerebellar Model Articulation Controller (CMAC)

- Let us consider the case of a first-order plant that needs to be controlled to yield a known, desired output signal.
- A sample discrete-time first-order plant is described with the difference equation

$$y(k+1) = ay(k) + bx(k) \tag{77}$$

where $y(k)$ is the system output, $x(k)$ is the input, k is the time step number, and a and b are constant coefficients

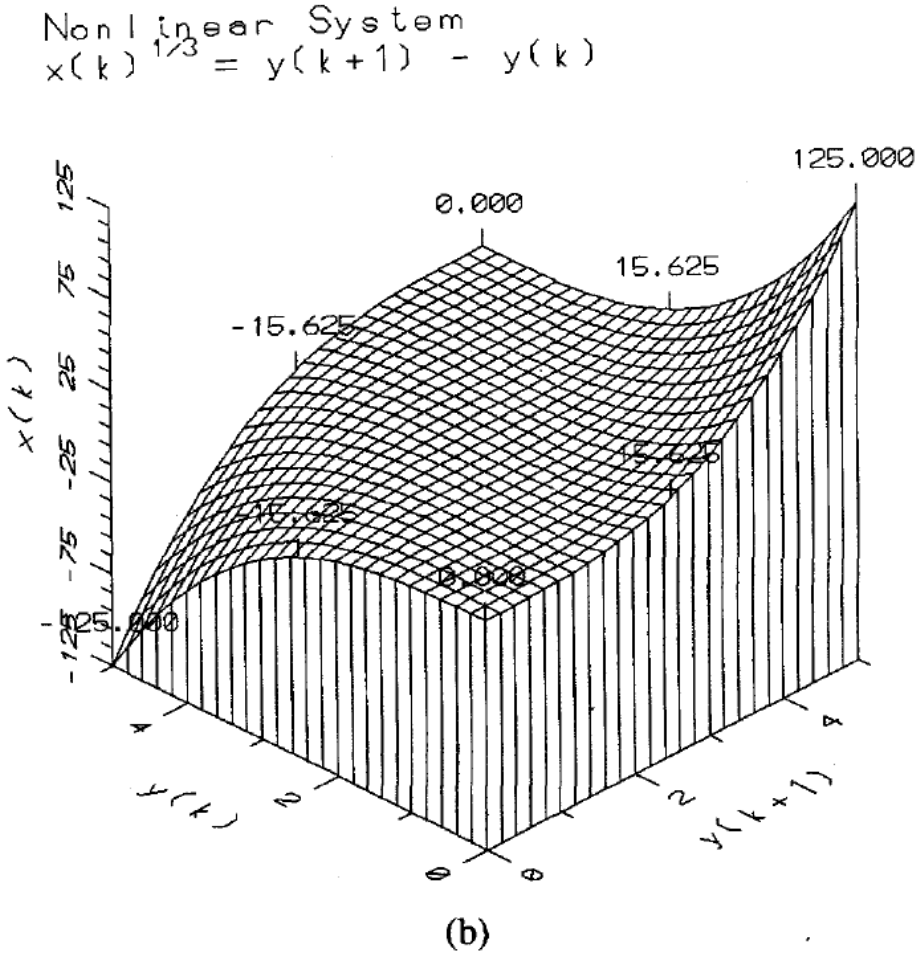
- Two examples of characteristic surfaces are illustrated in below Figure.



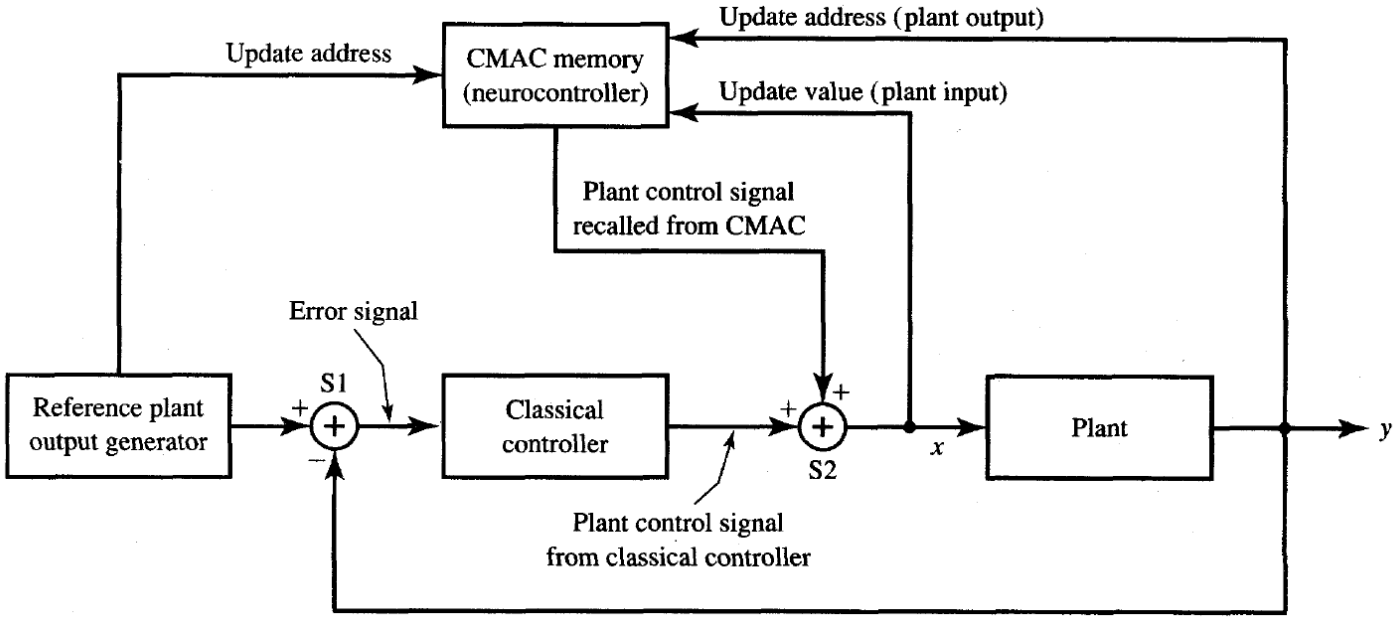
(a)

- A sample nonplanar surface for the equation

$$y(k+1) = y(k) + [x(k)]^{1/3} \tag{78}$$



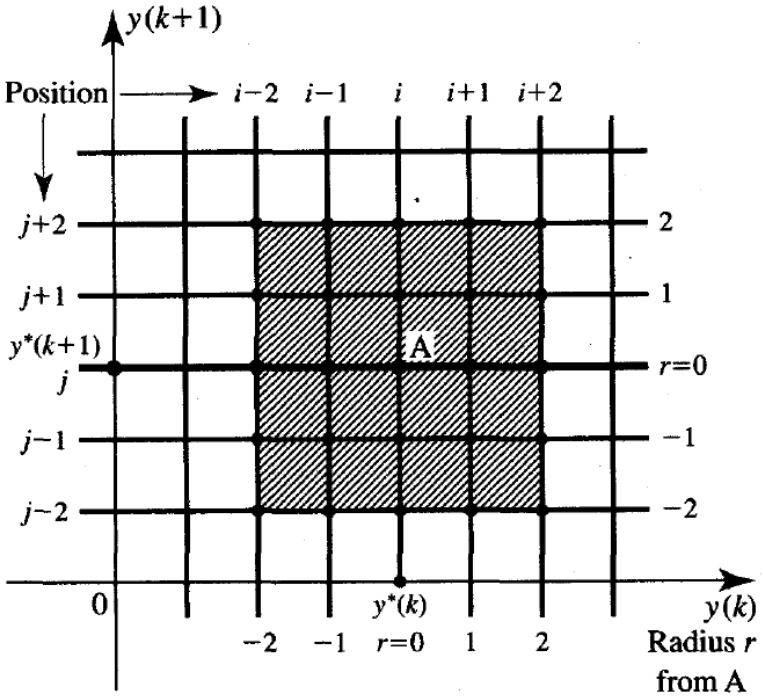
- CMAC neurocontrol block diagram



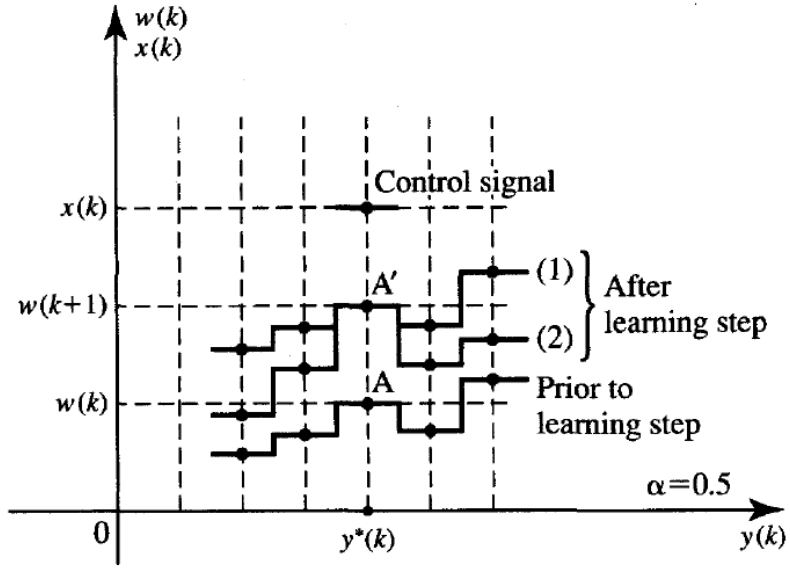
- The characteristic surface adapts in each control step according to the following learning law

$$w(k+1) = w(k) + \alpha[x(k) - w(k)] \tag{79}$$

- Learning of CMAC neurocontroller with generalization: (a) example of neighborhood learning radius of 2 and (b) learning at cross section $y^*(k+1)$ with generalization (two cases).

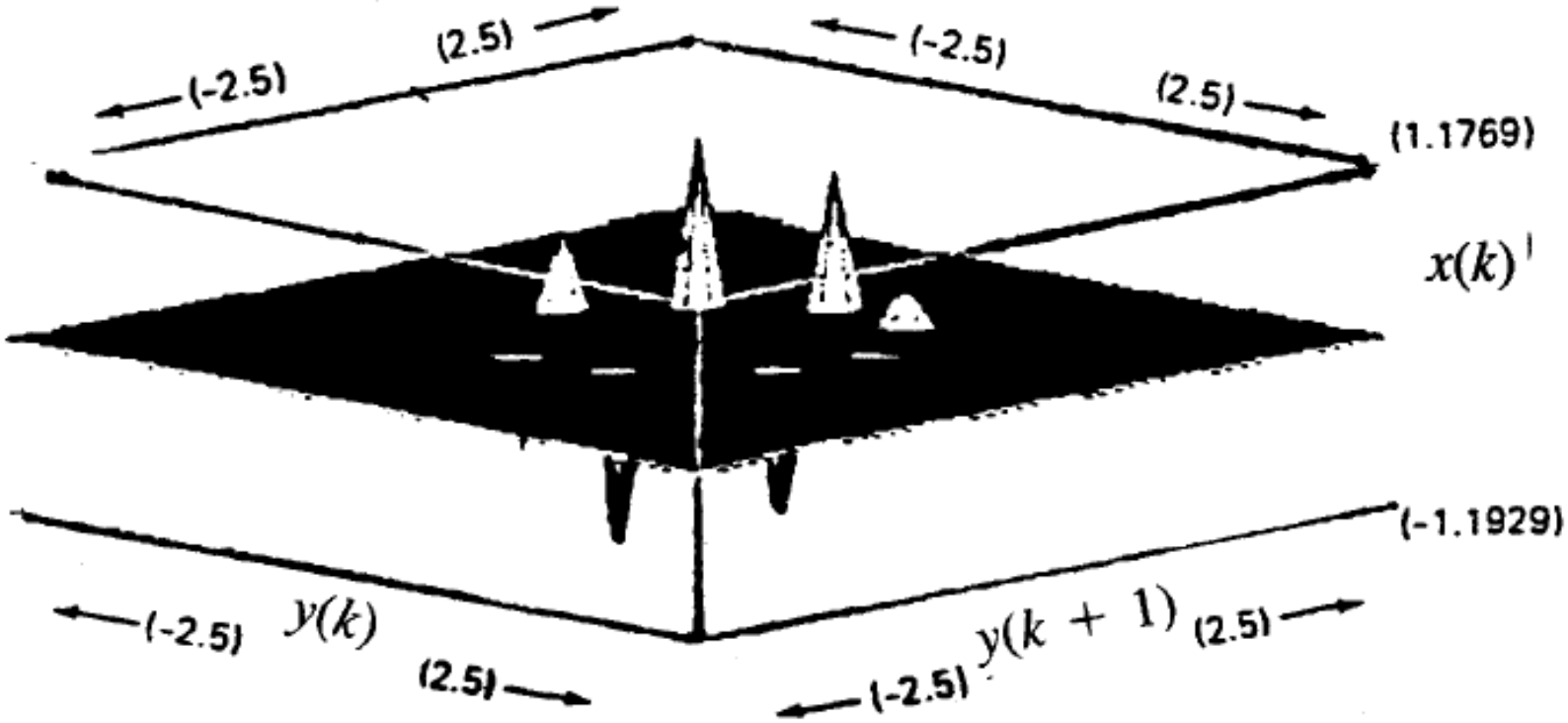


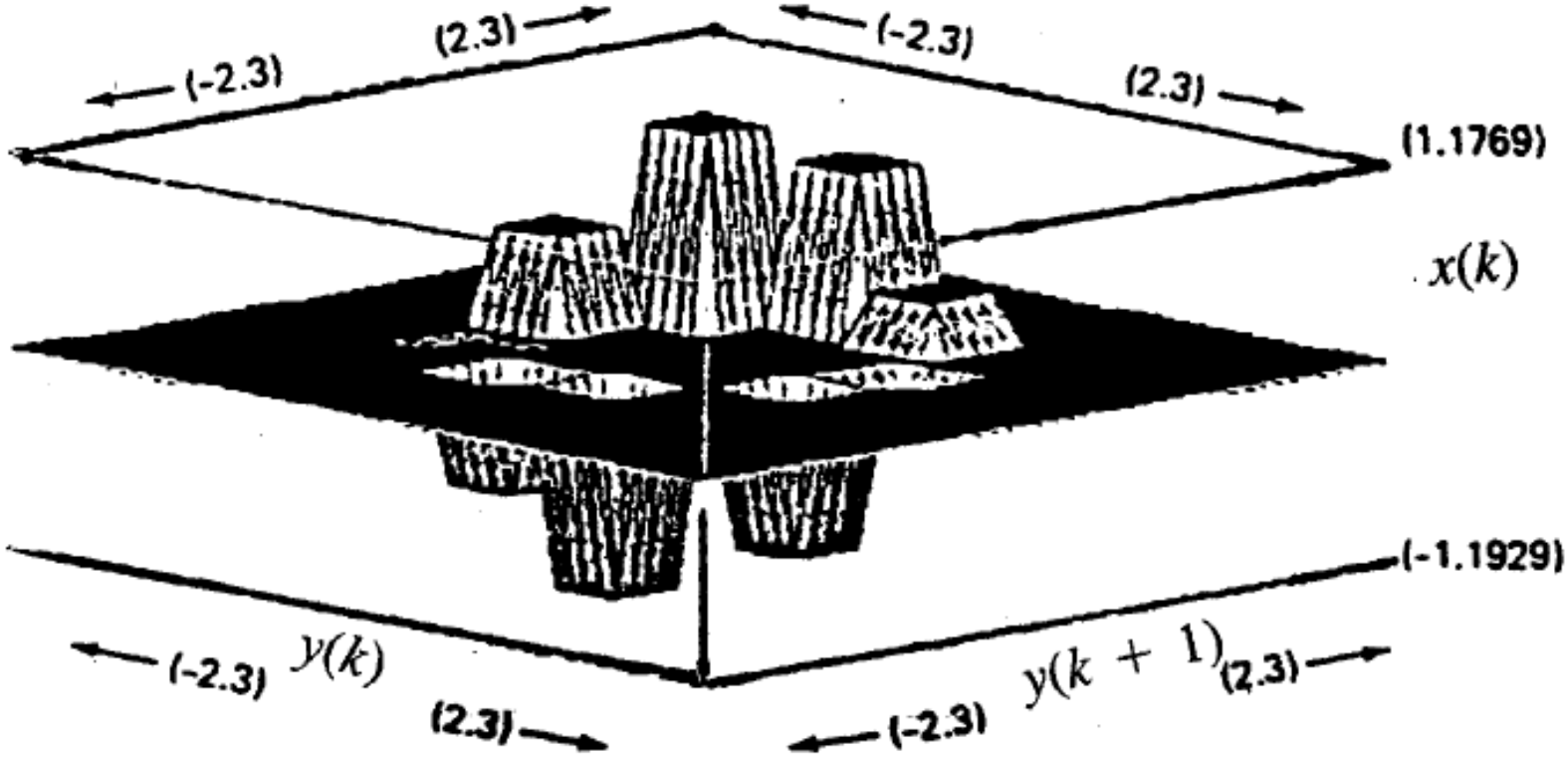
(a)



(b)

- Learning of the characteristic surface: (a) pointwise learning mode (without generalization) and (b) neighborhood learning mode (with generalization)





(b)

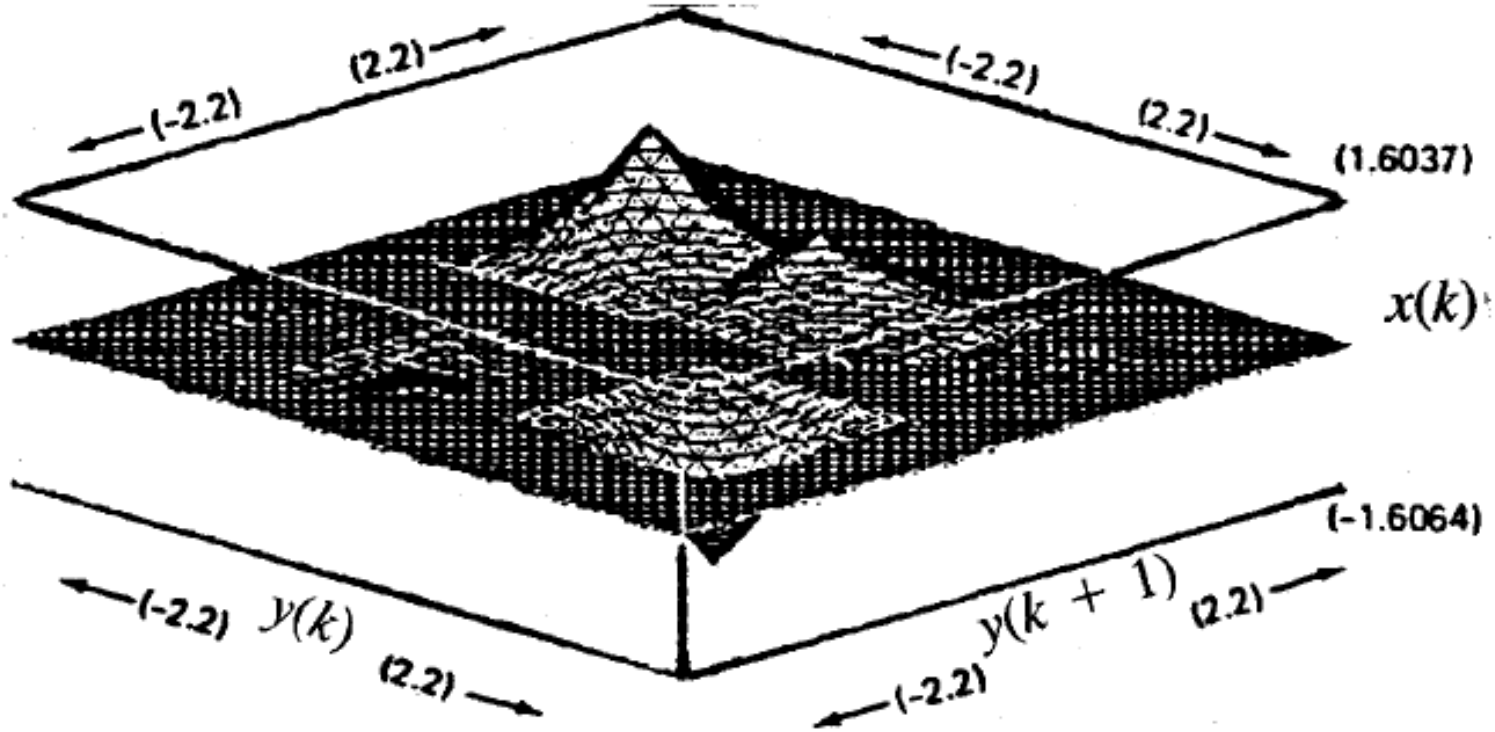
- The difference equation of the integrator-filter, which produces the desired sequence $y(k)$ at the output of the plant, is

$$y(k+1) - 0.1y(k) - 0.9r(k) = 0 \tag{80}$$

- The plant had the linear characteristic surface augmented by the nonlinear term $-0.4y(k)^{l/3}$ to the form

$$y(k+1) = y(k) - 0.4y(k)^{l/3} + x(k) \tag{81}$$

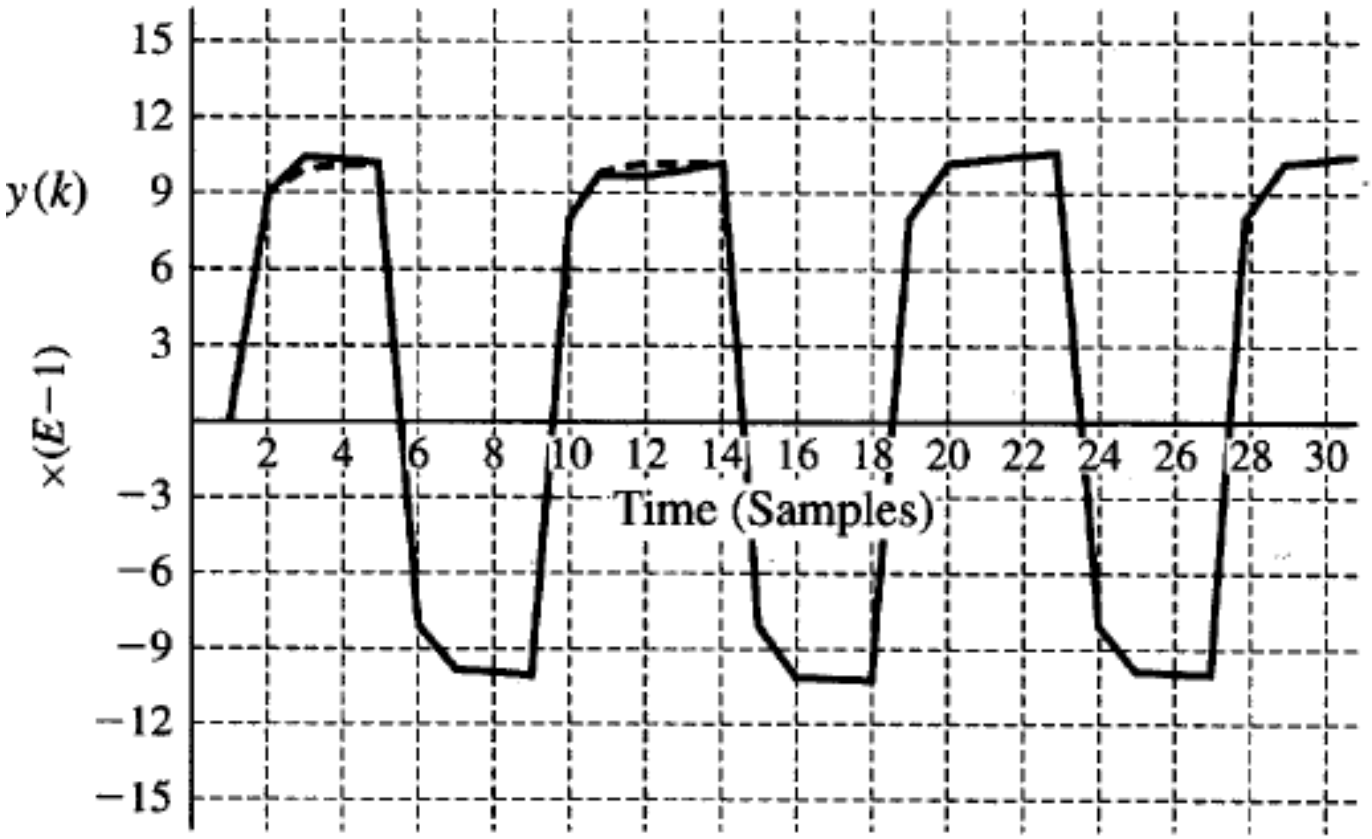
- CMAC learning: (a) characteristic surface learned and (b) control signals computed for the known plan

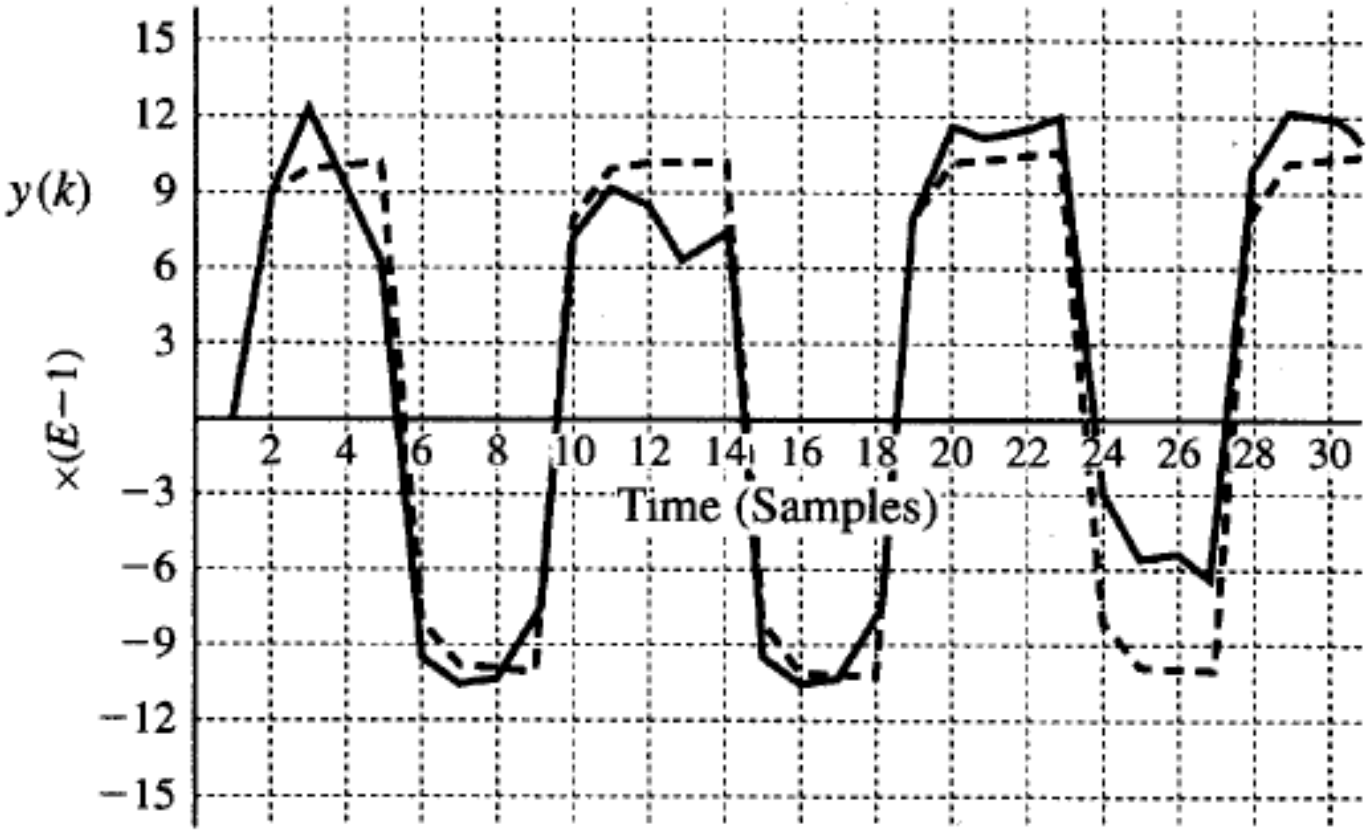

(-2.2)
(2.2)
(-2.2)
(2.2)
(1.6037)
x(k)
(-1.6064)
(-2.2)
y(k)
(2.2)
(-2.2)
y(k + 1)
(2.2)

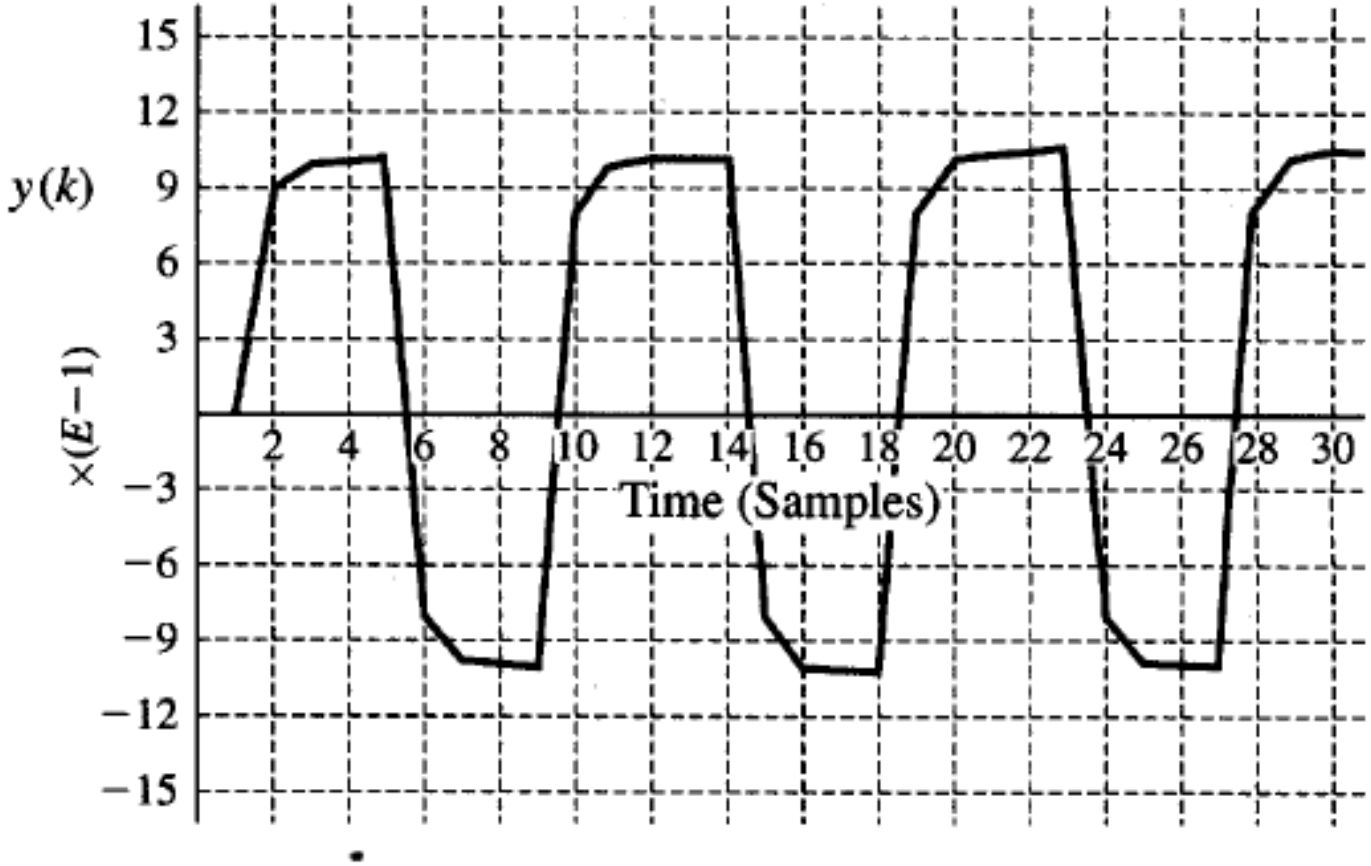
Time Instant	Reference Wave	Plant Output		Control Signal
		Present	Next	
k	$r(k)$	$y(k)$	$y(k+1)$	$x(k)$
0	0	0	0	0
1	1	0	0.9	0.9
2	1	0.9	0.99	0.09
3	1	0.99	0.999	0.009
4	1	0.999	0.9999	0.0009
5	−1	0.9999	−0.8	−1.7999
6	−1	−0.8	−0.98	−0.1799
7	1	−0.98	−0.998	−0.018
8	1	−0.998	−0.9998	−0.0018
9	0	−0.9998	−0.1	0.8998
10	1	−0.1	0.8	0.9
11	1	0.8	0.98	0.18
12	1	0.98	0.998	0.018

$y(k+1) = 0.1y(k) + 0.9r(k)$ (filter)
$x(k) = y(k+1) - y(k)$ (plant)

- CMAC-generated plant response (dashed lines denote desired output): (a) plant as in Eq. 77, a = b = 1, (b) same as part (a) but noise added, and (c) noise free nonlinear plant Eq. 81

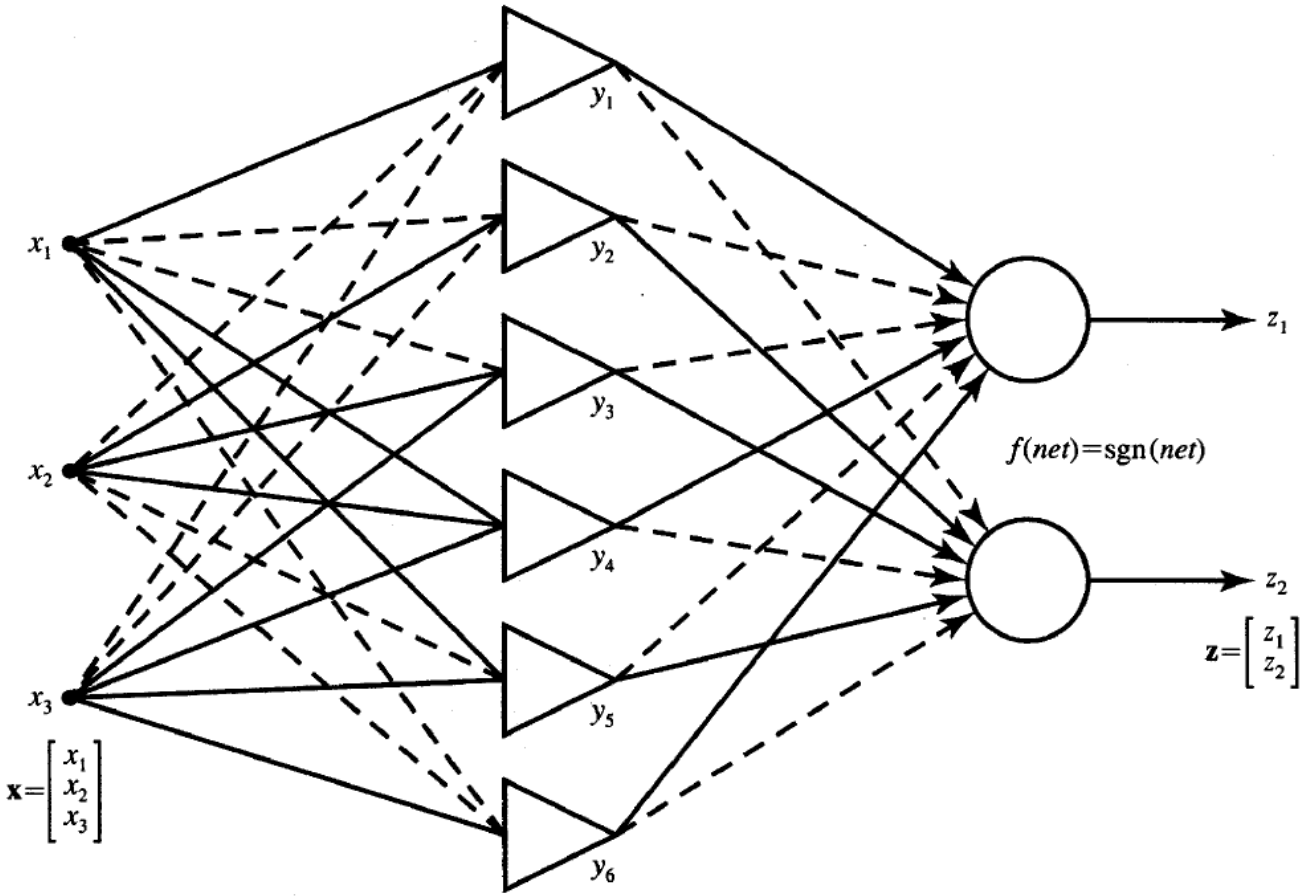



15
12
9
6
3
−3
−6
−9
−12
−15
y(k)
×(E−1)
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
Time (Samples)


15
12
9
6
3
−3
−6
−9
−12
−15
y(k)
×(E−1)
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
Time (Samples)

Tutorial 11

1. Analyse the following counter-propagation network for which the feedforward part is only operational. The network is shown in Figure. Assume the bipolar binary input and output vectors. Provide results for three- and two-dimensional cubes indicating cluster centers, which the network is supposed to detect and subsequently map into output vectors.



$$f(net)=\frac{1}{1+e^{-net}} \text{ (training)}$$

$$f(net)=\begin{cases} 0 & net \leq 0 \\ 1 & net > 0 \end{cases} \text{ (recall)}$$

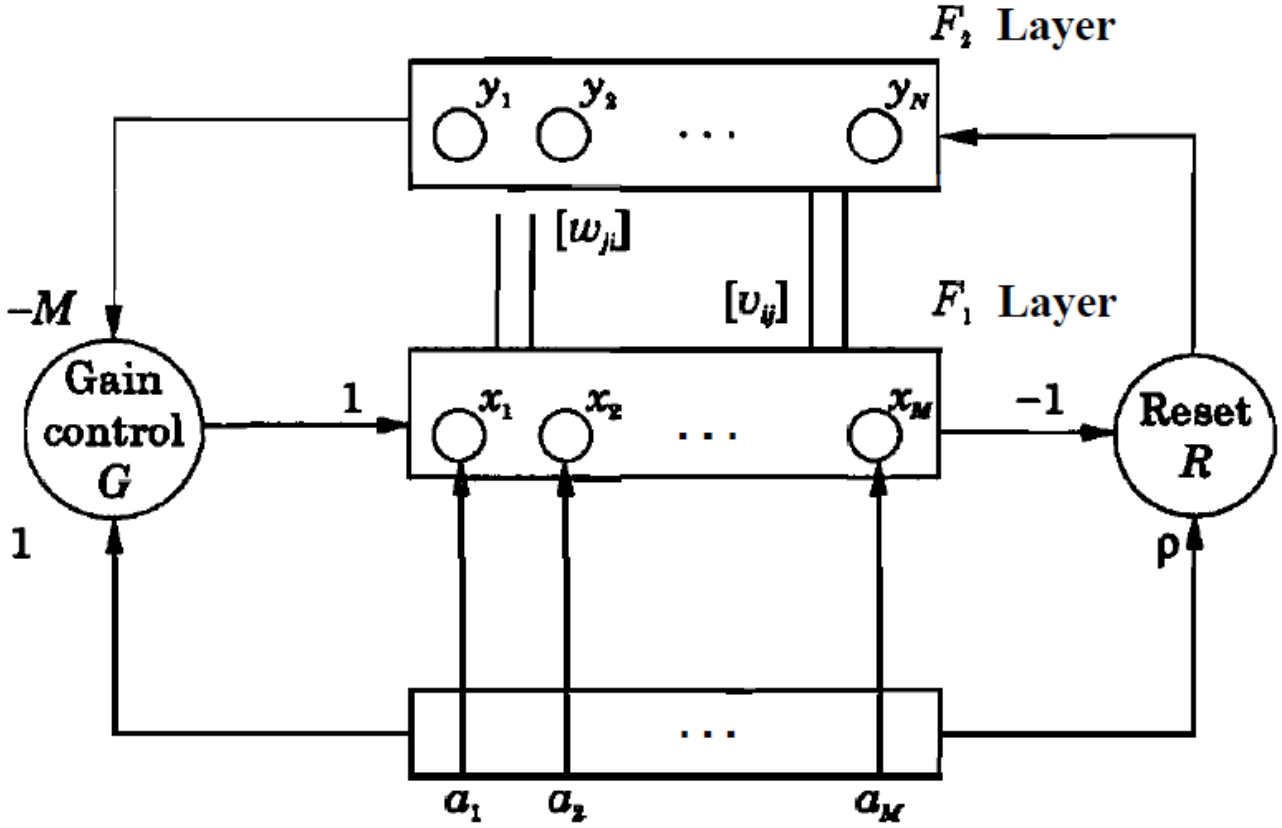
Weight $=+1$ ——————

Weight $=-1$ – – – –

Adaptive Resonance Theory (ART)

- Many pattern mapping networks can be transformed to perform pattern classification or category learning tasks.
- However these networks have the disadvantage that during learning the weight vectors tend to encode the presently active pattern, thus weakening the traces of patterns it had already learnt.
- Moreover, any new pattern that does not belong to the categories already learnt is still forced into one of them using the best match strategy, without taking into account how good even the best match is.
- The lack of stability of weights as well the inability to accommodate patterns belonging to new categories, led to the proposal of new architectures for pattern classification.

- These architectures are based on adaptive resonance theory (ART) and are specially designed to take care of the so called stability-plasticity dilemma in pattern classification
- ART also uses a combination of instar-outstar networks as in the CPN, but with the output layer merged with the input layer, thus forming a two-layer network with feedback as shown in Figure



- The minimal ART network includes a bottom-up competitive learning system (F_1 to F_2) combined with a top-down (F_2 to F_1) outstar pattern learning system.
- The number of units in the F_2 layer determines the number of possible categories of the input patterns.

- When an input pattern a_i is presented to the F_1 layer, the system dynamics initially follows the course of competitive learning, leading to a winning unit in the competitive F_2 layer depending on the past learning of the adaptive weights of the bottom-up connections from F_1 to F_2.
- The signals sent back from the winning unit in the F_2 layer down to F_1 via a top-down outstar network correspond to a prototype vector.
- This prototype vector is compared to the input pattern vector at the F_1 layer.
- If the two vectors match well, then the winning unit in the F_2 layer determines the category of the input pattern.
- If the match is poor, as determined by a vigilance parameter, then the winning unit in the F_2 layer does not represent the proper class for the input pattern a.

- That unit is removed from the set of allowable winners in the F_2 layer.
- The other units in the F_2 layer are likewise skipped until a suitable match is obtained at the F_1 layer between the top-down prototype vector and the input vector.
- When a match is obtained, then both the bottom-up and top-down network weights are adjusted to reinforce the input pattern.
- If no match is obtained, then an uncommitted unit (whose category is not identified during training) in the F_2 layer is committed to this input pattern, and the corresponding weights are adjusted to reinforce the input.
- The above sequence of events conducts a search through the encoded patterns associated with each category, trying to find a sufficiently close match with the input pattern.
- If no category exists, a new category is made.

- The search process is controlled by two subsystems, namely the orienting subsystem and the attentional subsystem.
- The orienting subsystem uses the dimensionless vigilance parameter that establishes the criterion for deciding whether the match is good enough to accept the input pattern as an exemplar of the chosen category.
- The gain control process in the attentional subsystem allows the units in the F_1 layer to be engaged only when an input pattern is present, and it also actively regulates the learning
- Stability is achieved in the ART network due to dynamic matching and control in learning.
- Plasticity is achieved in the ART due to its ability to commit an uncommitted unit in the F_2 layer for an input pattern belonging to a category different from the categories already learnt.

- In ART information from units resonate back and forth between two layers.
- Once the proper patterns develop, the neural network can be said to be in resonance.
- During this resonance period the adaptive weights are adjusted.
- No learning takes place before the network reaches a resonant state.
- ARTI network was proposed to deal with binary input patterns.
- The algorithm for binary valued ARTI is as follows

ART algorithm

- w_{ji} is the weight from the i^{th} unit in the F_1 layer to the j^{th} unit in the F_2 layer.
- w_j is the weight vector leading to the j^{th} unit in the F_2 layer from all units in the F_1 layer.
- v_{ij} is the weight from the j^{th} unit in the F_2 layer to the i^{th} unit in the F_1 layer.
- v_j is the weight vector (prototype vector) emanating from the j^{th} unit in the F_2 layer to all the units in the F_1 layer.
- Initially set all the components of all the prototype vectors to 1 i.e. $v_{ij} = 1$, for all i and j.
- This will enable the uncommitted units in the F_2 layer also to compete in the same way as the learned units.
- Initialize all w_{ij} to random values in the range 0 to 1.

1. Enable all the units in the F_2 layer.
2. For an input binary pattern a to the F_1 layer, determine the winner unit k in the F_2 layer by computing

$$k = \arg[\max_j w_j^T a]$$

3. A similarity measure between the winning prototype v_k and the input a is computed and compared with a vigilance parameter $(0 < p < 1)$.
4. The similarity measure gives the fraction of bits of a that are also present in v_k

$$\frac{v_k^T a}{\sum_{i=1}^{M} a_i} \geq p$$

5. If the prototype associated with the winner unit k passes the vigilance test, then go to Step 8 to adjust the weight vectors associated with the k^{th} unit both in the forward and backward directions.
6. If the vigilance test fails, then the output unit k is disabled and another winner is selected by repeating Steps 2 to 4
7. If none of the committed units in the F_2 layer passes the vigilance test, then an uncommitted unit is committed to the input pattern and the corresponding prototype vector v_k is set equal to the input pattern a i.e. $v_k = a$.

8. The weights are adjusted as follows:

$$v_{ik}(m+1) = v_{ik}m \wedge a_i, i = 1, 2, \ldots, M$$

$$w_{ki}(m+1) = \frac{v_{ik}(m+1)}{0.5 + \sum_{i=1}^{M} v_{ik}(m+1)}, i = 1, 2, \ldots, M$$

where $\wedge$ is the logical AND operation and $w_k(m+1)$ can be viewed as normalized version of $v_k(m+1)$ normalized with the number of 1's in the vector.

- ART1 network operates automatically through the use of the gain parameter (G) and the reset parameter (R).
- The gain control unit operates as follows:
- If all the units in the F_2 layer are OFF, then $G = 1$.
- If anyone of the units in the F_2 layer is ON, then $G = 0$.

- The gain parameter G is generated using the function

$$G = f\left(\sum_{i=1}^{M} a_i - M\sum_{j} y_j - 0.5\right)$$

- For any winning unit in the F_2 layer, one of the y_j will be 1, then the argument of $f(.)$ is less than 0. Hence $G = 0$.
- The output of the i^{th} unit in the F_1 layer is given by

$$x_i = f\left(a_i + \sum_{j=1}^{N} v_{ij} y_j + G - 1.5\right)$$

- The reset value R is computed as follows:

$$R = f\left(p\sum_{i=1}^{M} a_i - \sum_{i=1}^{M} x_i\right)$$

- If the vigilance test

$$\frac{\sum_i x_i}{\sum a_i} = \frac{\sum_i a_i \wedge v_{ik}}{\sum a_i} = \frac{v_k^T a}{\sum a_i} > p$$

succeeds, then the argument of $f(.)$ will be negative, and hence $R = 0$. That is there is no reset.

- On the other hand, if the vigilance test fails, then the argument of $f(.)$ is positive and hence $R = 1$. Then the current winning unit is disabled and all the units in the F_2 layer are reset to OFF. Hence $G = 1$. Therefore another winning unit will be selected.
- ART models belong to the class of match-based learning as opposed to error-based learning of the backpropagation networks.

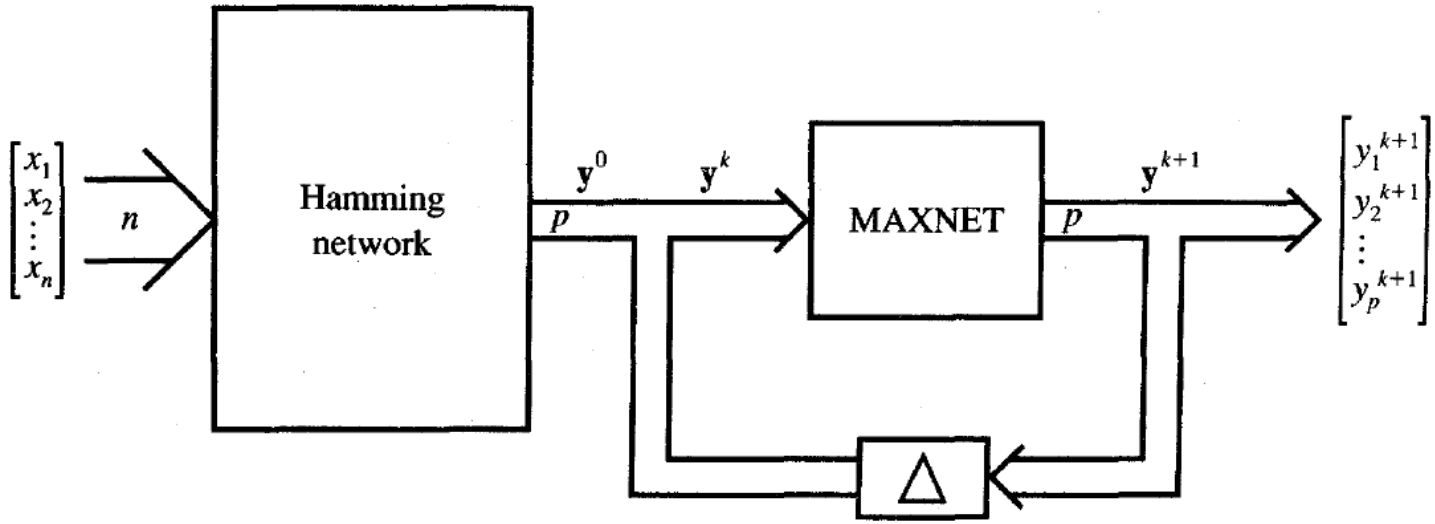
- In match-based learning the weights are adjusted only when the external input matches one of the stored prototypes, whereas in error-based learning the weights are adjusted only if there is an error between the actual output and the desired output.
- Thus match-based learning tends to group similar patterns whereas error-based learning tends to discriminate dissimilar patterns.

Tutorial 12

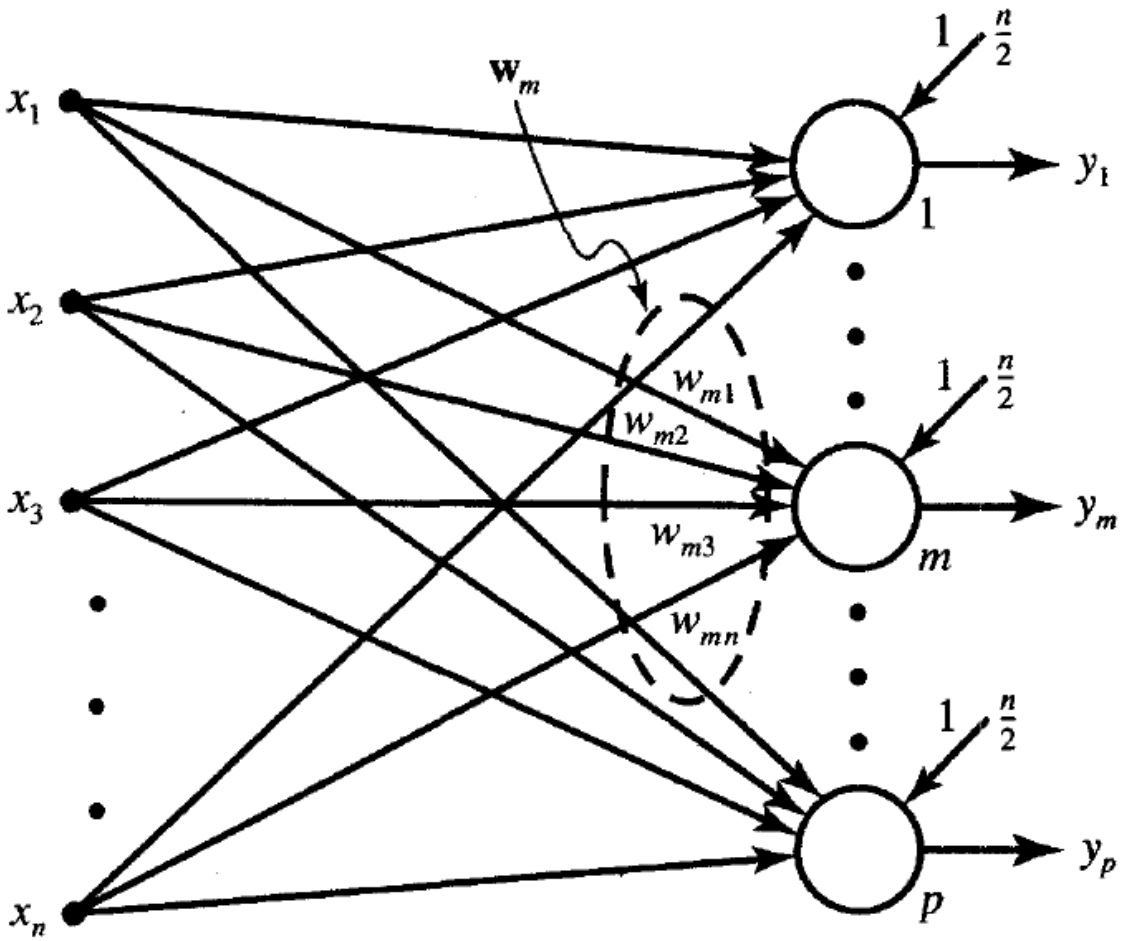
1. Explain the training algorithm of instar and outstar in CPN.
2. What are the problems in many pattern classification algorithms and how ART resolved these problems?
3. Explain ART algorithms.
4. What is similarity measure in ART algorithm?
5. Write the weight adjustment formula in ART.
6. Explain gain parameter and reset parameter of ART.
7. What is vigilance test in ART?

Hamming Net and MAXNET

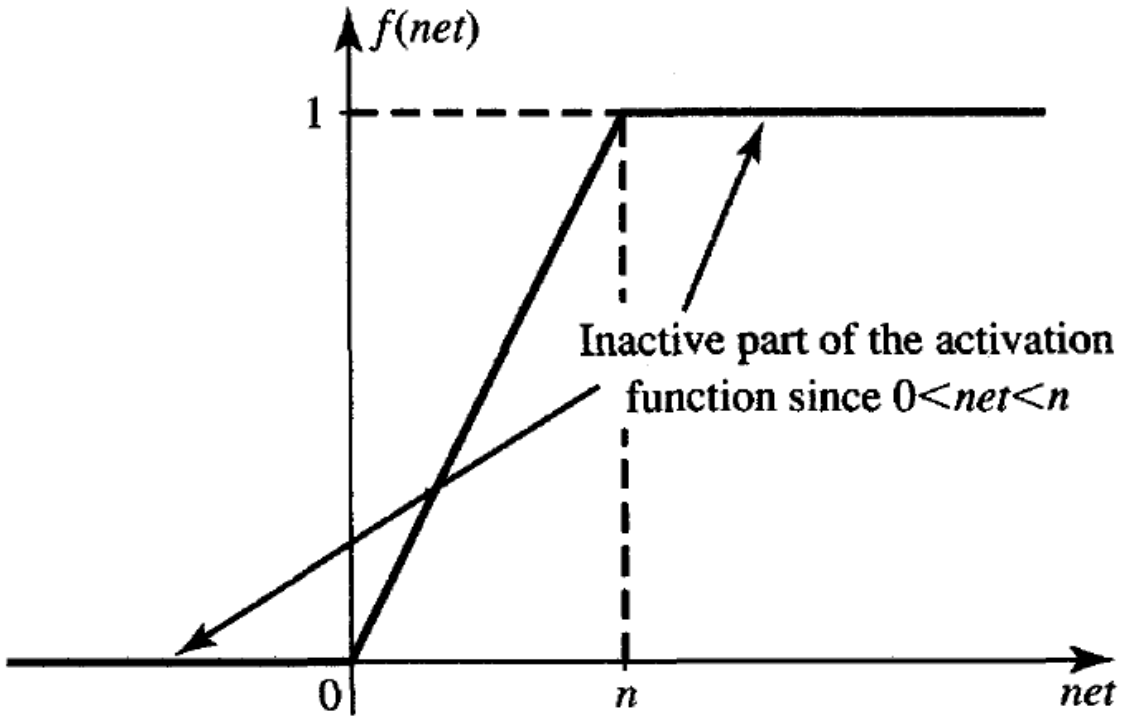
- Block diagram of the minimum HD classifier



- Hamming network for n-bit bipolar binary vectors representing p classes classifier network



- Neurons activation function for n-bit bipolar binary vectors representing p classes Hamming network



- $x^t s^{(m)}$ = total number of positions in which the two vectors agree - number of positions in which they differ.
- HD = number of different bit positions
- n = number of positions in which two vectors agree + HD

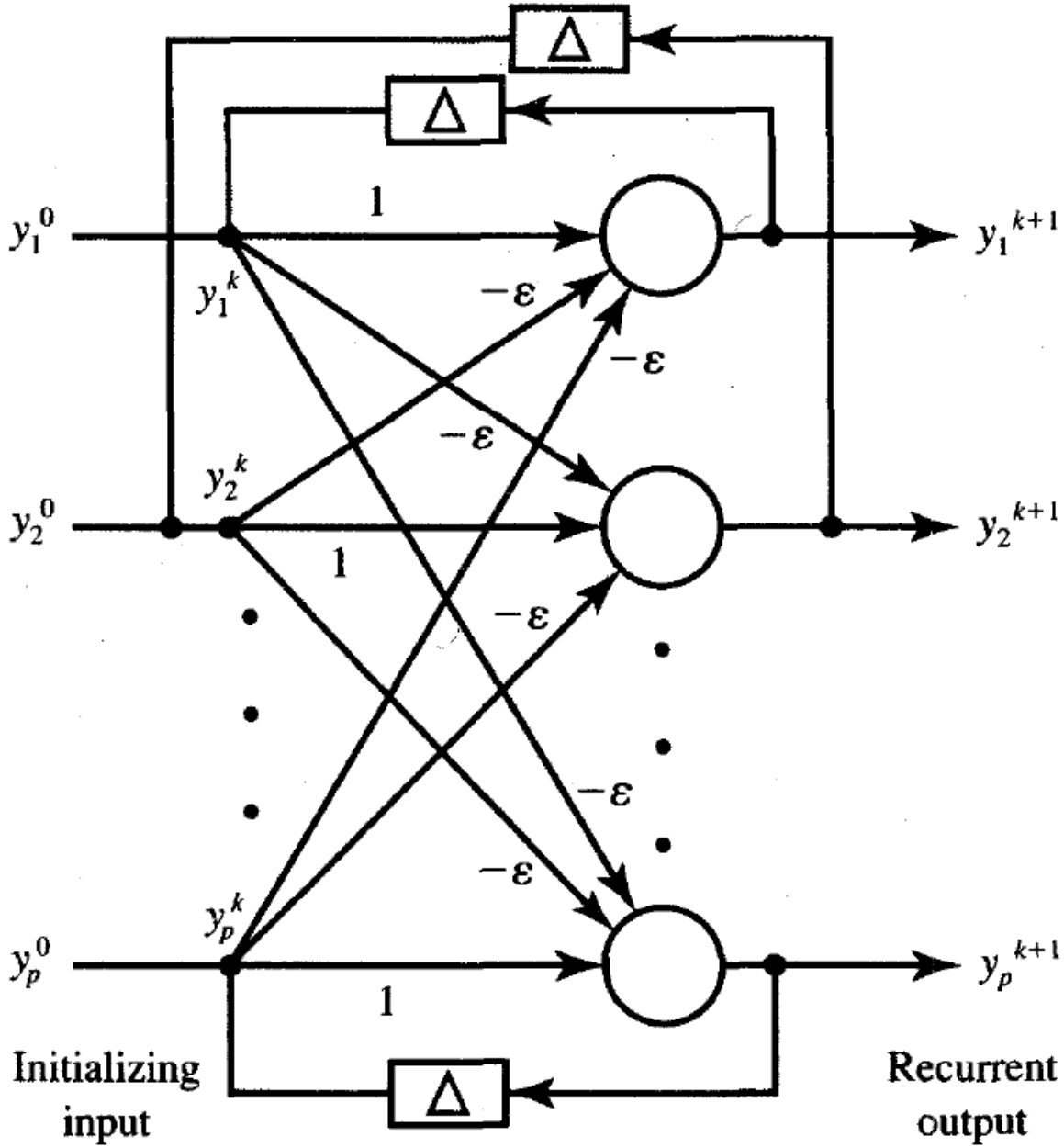
- number of positions in which two vectors agree $= n - HD$
- $x^t s^{(m)} = \left(n - HD(x, s^{(m)})\right) - HD(x, s^{(m)})$
- $x^t s^{(m)} = n - 2HD(x, s^{(m)})$
- $\frac{1}{2} x^t s^{(m)} = \frac{1}{2} n - HD(x, s^{(m)})$
- The weight matrix W_H of the Hamming network can be created by encoding the class vector prototypes as rows in the form as below

$$W_H = \frac{1}{2} \begin{bmatrix} s_1^{(1)} & s_2^{(1)} & \dots & s_n^{(1)} \\ s_1^{(2)} & s_2^{(2)} & \dots & s_n^{(2)} \\ \vdots & \vdots & \dots & \vdots \\ s_1^{(p)} & s_2^{(p)} & \dots & s_n^{(p)} \end{bmatrix}$$

- $net_m = \frac{1}{2} x^t s^{(m)} + \frac{n}{2}$
- $net_m = n - HD(x, s^{(m)})$

- $f(net_m) = \frac{1}{n} net_m$
- The response of the Hamming network essentially terminates the classification in which only the first layer of network computes the relevant matching score values.
- The classification by the Hamming network is performed in a feedforward and instantaneous manner.
- MAXNET needs to be employed as a second layer only for cases in which an enhancement of the initial dominant response of the m'th node is required.
- As a result of MAXNET recurrent processing, the m'th node responds positively, as opposed to all remaining nodes whose responses should have decayed to zero.

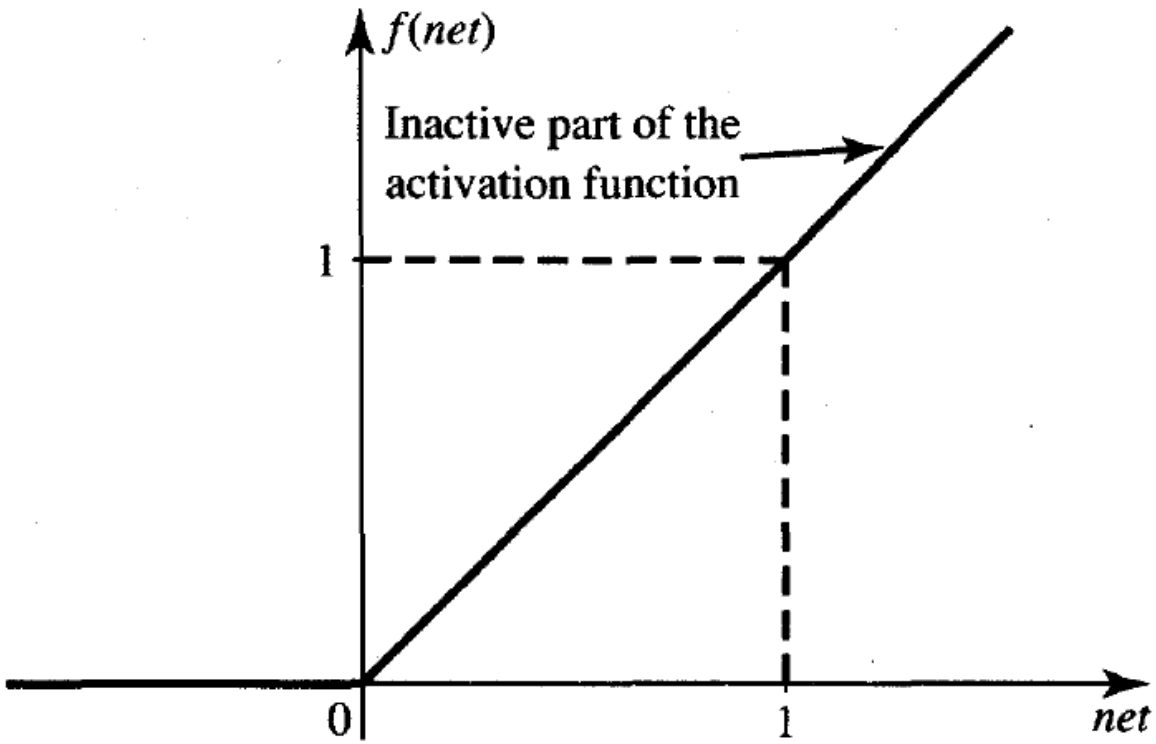
- MAXNET network architecture for p classes



- Weight matrix W_M of size $p \times p$

$$W_M = \begin{bmatrix} 1 & -\epsilon & -\epsilon & \dots & -\epsilon \\ -\epsilon & 1 & -\epsilon & \dots & -\epsilon \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\epsilon & -\epsilon & -\epsilon & \dots & 1 \end{bmatrix}$$

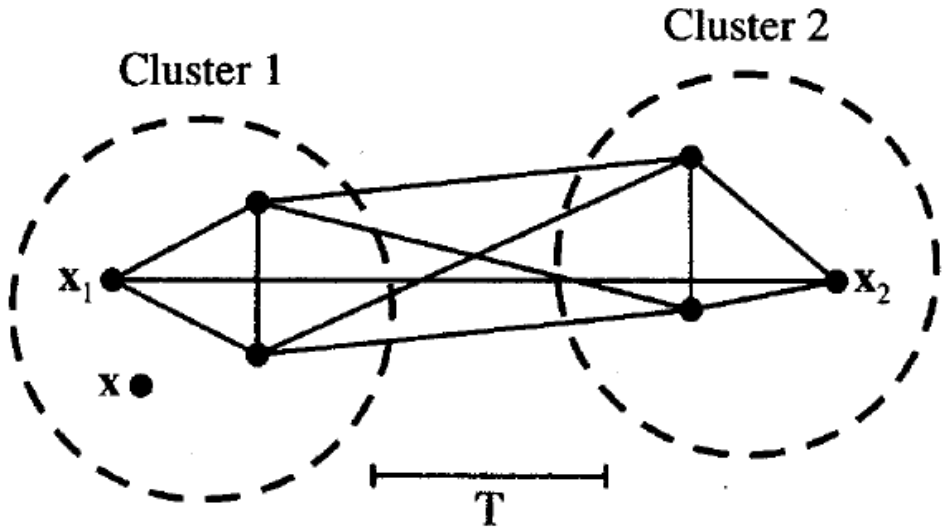
- MAXNET neuron's activation function



Unsupervised Learning of Clusters

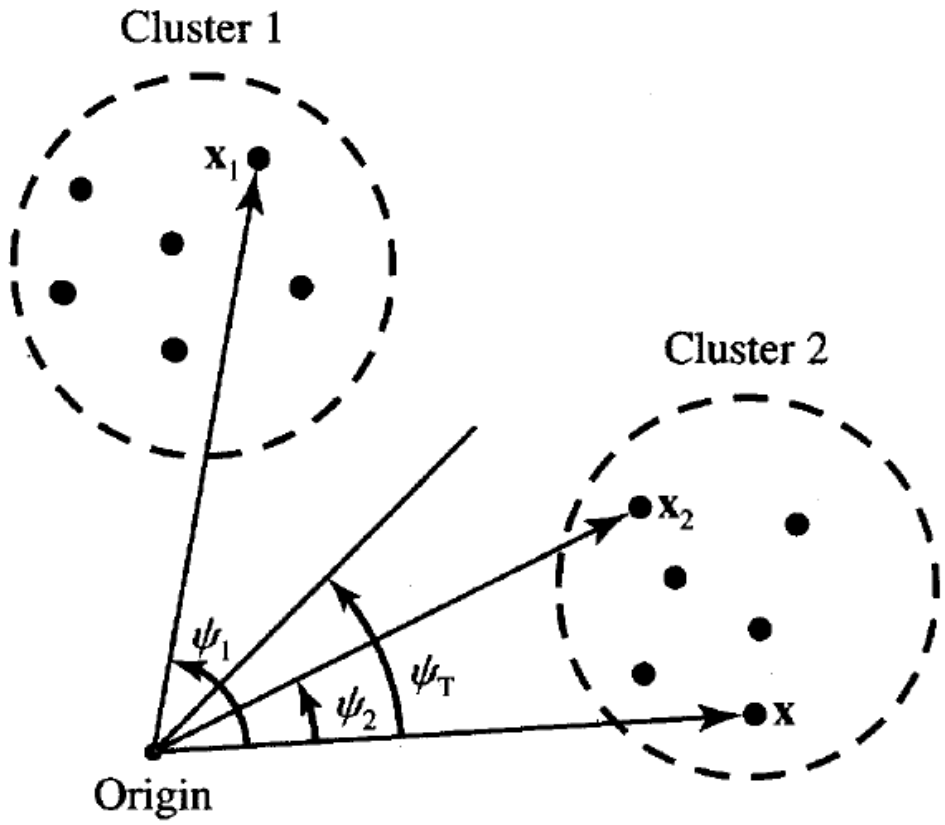
- The process of providing the category of objects with a label is usually termed as calibration.
- **Clustering and Similarity Measures**
- Euclidean distance between two patterns x and x_i

$$\| x - x_i \| = \sqrt{(x - x_i)^t (x - x_i)}$$



- normalized scalar product Another similarity rule is the cosine of the angle between x and x_i

$$cos\psi = \frac{x^t x_i}{\| x \| . \| x_i \|}$$

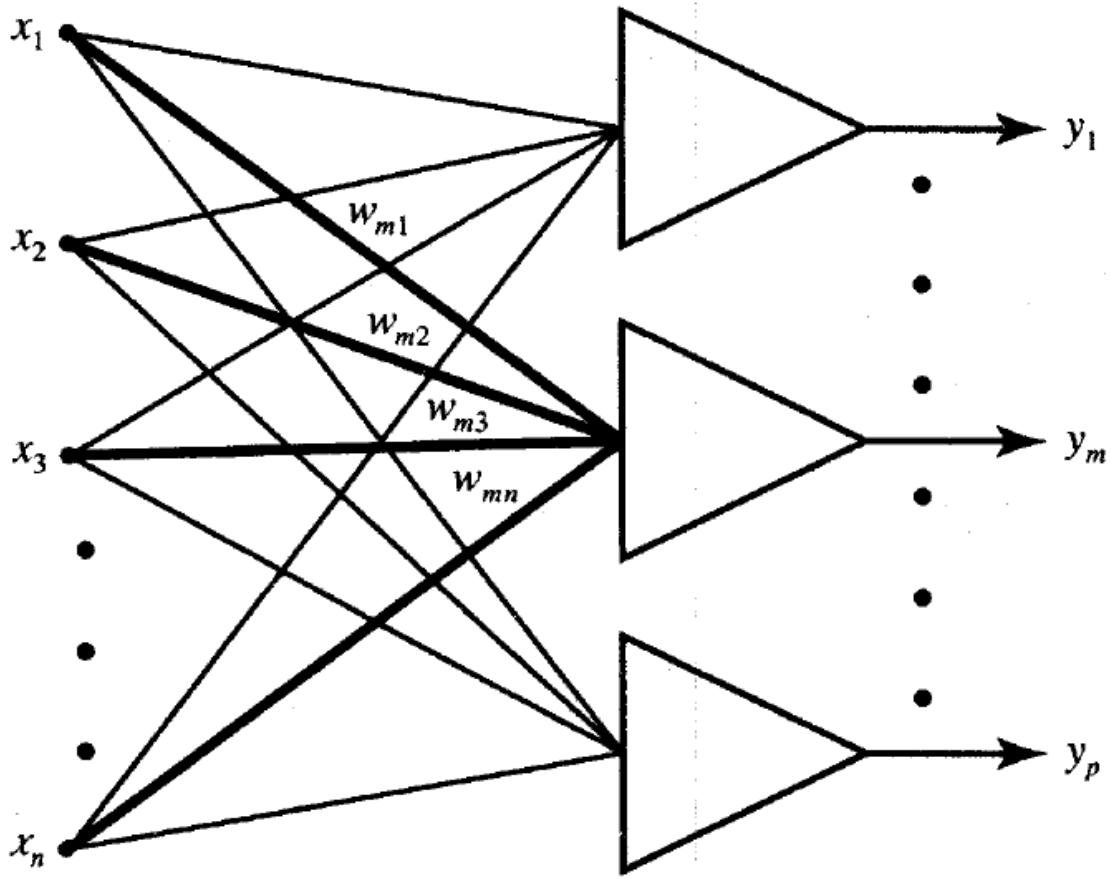


- **Winner-Take-All Learning**
- The processing of input data x from the training set $\{x_1, x_2, \ldots, x_N\}$, which represents p clusters, follows the customary expression $y = \Gamma[Wx]$ with diagonal elements of the operator Γ being continuous activation function operating component wise on entries of vector Wx.
- The processing by the layer of neurons is instantaneous and feedforward. To analyse network performance, we rearrange the matrix W to the following form:

$$W = \begin{bmatrix} w_{11} & w_{12} & \ldots & w_{1n} \\ w_{21} & w_{22} & \ldots & w_{2n} \\ \vdots & \vdots & \ldots \vdots & \\ w_{p1} & w_{p2} & \ldots & w_{pn} \end{bmatrix} = \begin{bmatrix} w_1^t \\ w_2^t \\ \vdots w_p^t \end{bmatrix}$$

$$w_i = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \vdots w_{in} \end{bmatrix} \quad \text{for } i = 1, 2, \ldots, p$$

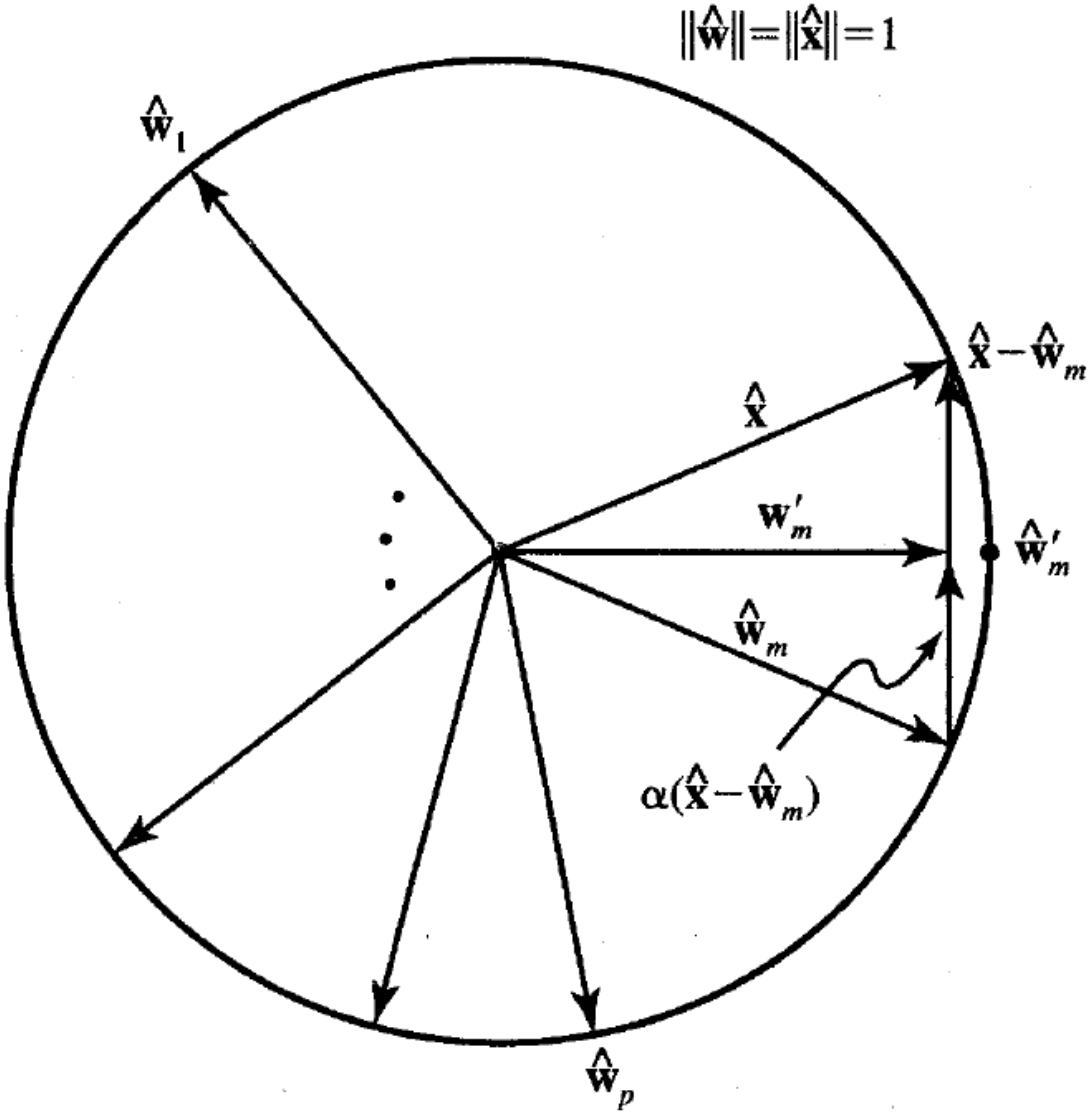
w_i is the column vector equal to the i'th row of the weight matrix W. Component weights of w, are highlighted in the below figure showing a winner-take-all learning network.



(Adapted weights highlighted)

- The learning algorithm treats the set of p weight vectors as variable vectors that need to be learned. Prior to the learning, the normalization of all weight vectors is required:

$$\hat{w}_i = \frac{w_i}{\| w_i \|}$$


$\|\hat{\mathbf{w}}\| = \|\hat{\mathbf{x}}\| = 1$
$\hat{\mathbf{w}}_1$
$\hat{\mathbf{x}} - \hat{\mathbf{w}}_m$
$\hat{\mathbf{x}}$
$\mathbf{w}'_m$
$\hat{\mathbf{w}}'_m$
$\hat{\mathbf{w}}_m$
$\alpha(\hat{\mathbf{x}} - \hat{\mathbf{w}}_m)$
$\hat{\mathbf{w}}_p$

- The weight adjustment criterion for this mode of training is the selection of $\hat{w}_i$ such that

$$\| x - \hat{w}_m \| = \min_i \| x - \hat{w}_i \| \tag{82}$$

The index m denotes the winning neuron number corresponding to the vector $\hat{w}_m$, which is the closest approximation of the current input x.

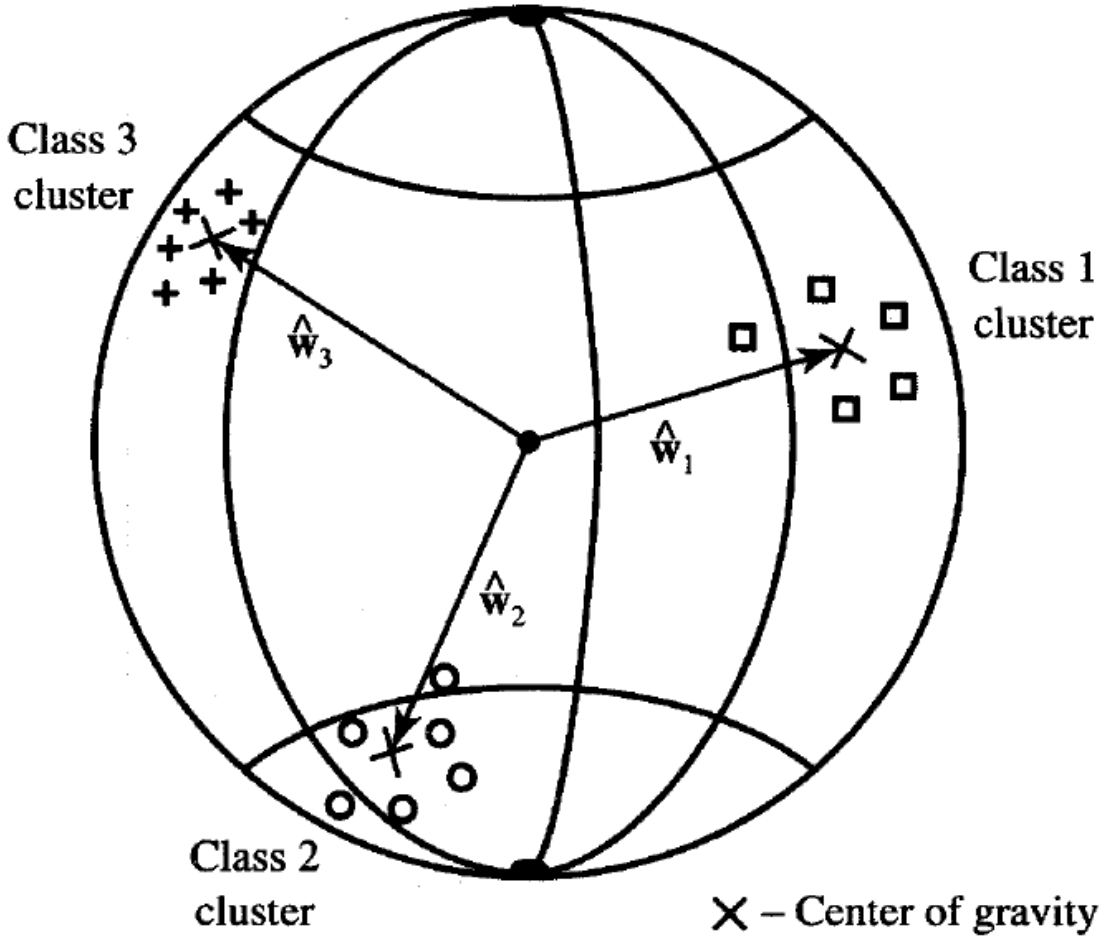
- The left side of above equation 82 can be rearranged to the form

$$\| x - \hat{w}_m \| = (x^t x - 2\hat{w}_m^t x - 1)^{1/2} \tag{83}$$

- Searching for the minimum of p distances as in the equation 82 is corresponds to finding the maximum among the p scalar products

$$\hat{w}_m^t = \max_i (\hat{w}_i^t x)$$

- weight vectors on a unity sphere for $p = n = 3$



- **Recall Mode**

  Layer now performs as a filter of the input vectors such that the largest output neuron is found as follows

  $$y_m = \max(y_1, y_2, \ldots, y_p)$$

  and the input is identified as belonging to cluster m. In general, the neurons activation functions should be continuous in this network

- **Initialization of Weights**

  vectors are initialized at the value

  $$w_i^0 = \frac{1}{\sqrt{n}}[1\ 1\ \ldots\ 1]^t$$

Tutorial 13

1. Design the Hamming network for the three following class prototype vectors

$$s^{(1)} = [1 \ -1 \ 1 \ -1 \ 1 \ -1]^t$$
$$s^{(2)} = [-1 \ -1 \ -1 \ -1 \ -1 \ 1]^t$$
$$s^{(3)} = [1 \ 1 \ 1 \ -1 \ -1 \ 1]^t$$

   1.1 Compute the weight matrix.
   1.2 Find net_m for $m = 1, 2, 3$, for the input vector $x = [1 \ 1 \ 1 \ 1 \ 1 \ 1]$ and verify that the HD parameter computed by the network agrees with the actual HD.
   1.3 Find the normalized responses $f(net_m)$ at the output of each node of the Hamming network

2. MAXNET with four input nodes has been designed for $\epsilon = 0.1$. After the two initial recurrences completed, the following network output resulted:

$$y^2 = [0.186\ 0.307\ 0.428\ 0.549]^t$$

Based on these data:

2.1 Find the initializing vector y^0 that has been applied initially, prior to recurrences.

2.2 Knowing that y^0 is the output vector of the Hamming network, determine the HD value between the closest 10-bit prototype and the Hamming network input vector originally applied to yield the computed y^0.

3. The MAXNET with four output nodes, $p = 4$, receives the input vector

$$y^0 = [0.5\ 0.6\ 0.7\ 0.8]^t$$

3.1 Find the ϵ value that would be required to suppress the output of the weakest node exactly to the zero value after the first cycle.

3.2 Find subsequent responses of the network, y^1 and y^2, for the computed value of ϵ.

Thank you

Please send your feedback or any queries to **akyadav1@amity.edu**
You can contact me on **+91 9911375598**