Artificial Neural Networks
Topic-01

Background
Structure & Function of Biological Neuron
Artificial Neural Networks: Bio-inspired Computing Paradigm

Biological Information Systems

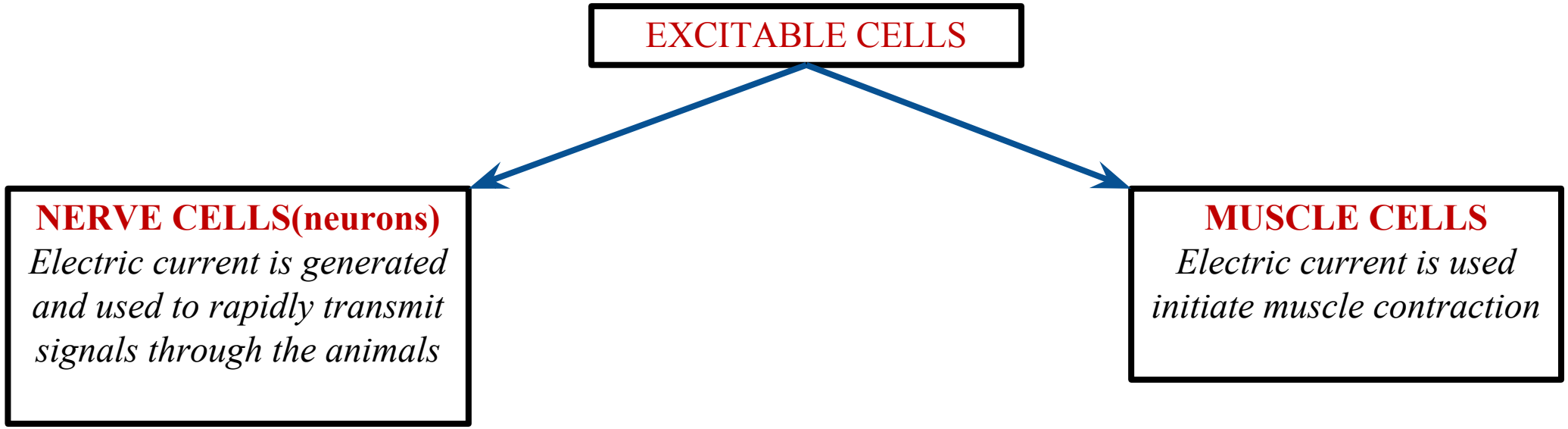
- There are two different types of biological information system that exist.
- The most distinguishing characteristics of living things is their ability to store, utilize and pass on information at an intra-cellular level . This information is encoded in DNA (that present within a cell) as a specific sequence of four different nucleotides(without any known exception). So DNA contains functional and structural information about an organism in extraordinary details. Bioinformatics strives to decode what information is biologically important and to interpret how it is used to precisely control the chemical environment within living organism.
- The brain is a highly *complex, nonlinear, and massively parallel information-processing system that occurs at inter-cellular level*.

Human Brain

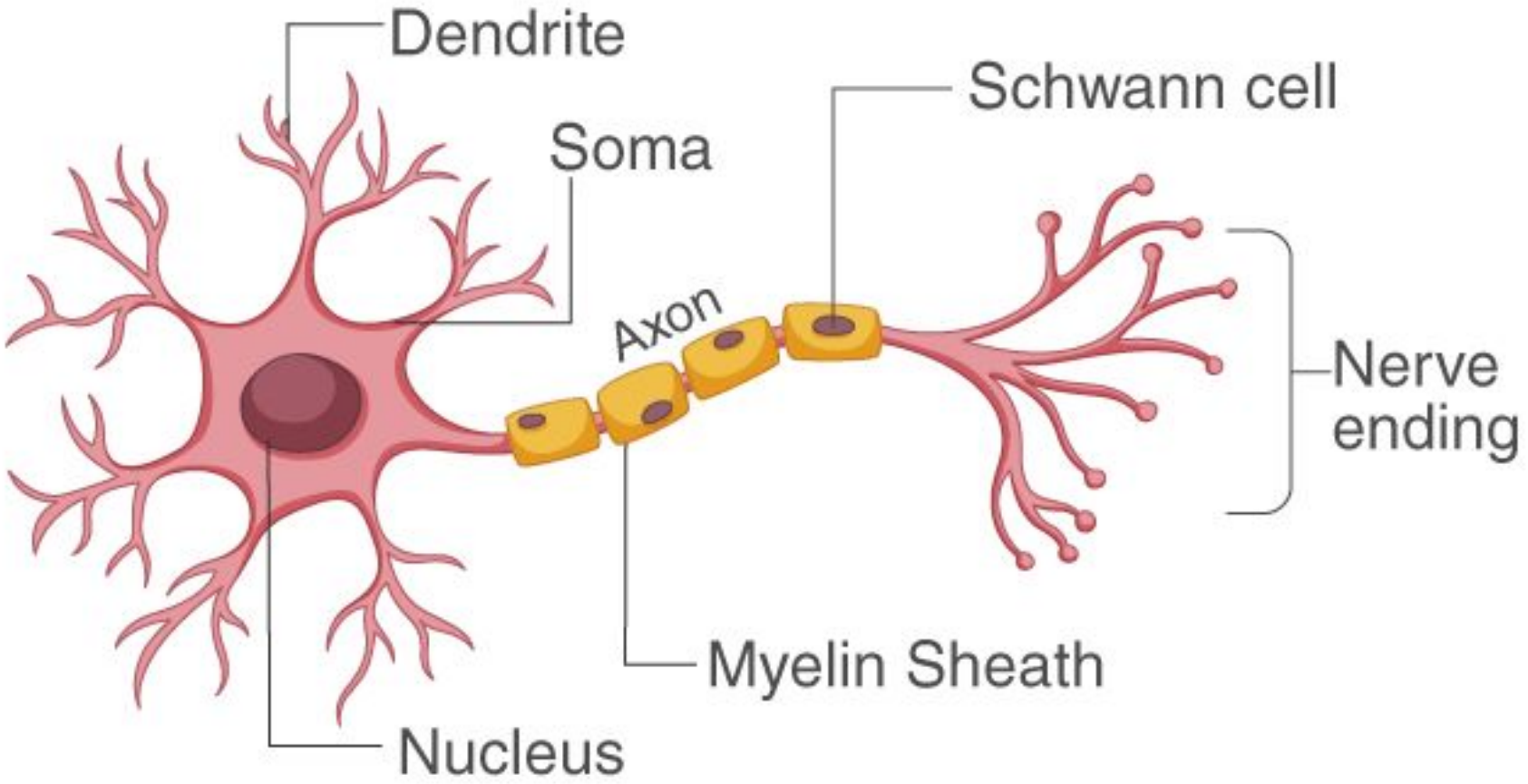
- The human nervous system can be viewed as a three stage system. Central to the system is the brain represented by biological neural net, which continually receives information from receptors, perceive it and makes appropriate decisions.
- The receptors convert stimuli from human body or the external environment(through different types of receptors that are present in eye nose, skin, ear and tongue) into electrical impulse that covey information to (biological) neural net (brain) by the sensory neurons.
- The effectors converts electrical impulse generated by the neural network sent through the motor neurons into discernible response as system output.
- Human brain weighs approximately 3 pound and the volume is 90 cubic inch. Accordingly, it was estimated to contain 100 billion cells out of which 10% are neurons (the fundamental computing element of the brain). So there are about 10^{10} neurons present in the brain.
- The total number of interconnections in the brain is 10^{14}, this implies that every neuron connected to 10000 other neurons on an average.

Excitable Cells

Excitable cells are those cells , which can be stimulated to create tiny electric current



STRUCTURE OF NEURON



- Dendrites: These are branch-like structures that receive messages from other neurons and allow the transmission of messages to the cell body.
- Cell Body: Each neuron has a cell body with a nucleus, Golgi body, endoplasmic reticulum, mitochondria and other components.
- Axon: Axon is a tube-like structure that carries electrical impulse from the cell body to the axon terminals that passes the impulse to another neuron.
- Synapse: It is the chemical junction between the terminal of one neuron and dendrites of another neuron.

Neuron Types

Neurons can be classified according to the direction in which they send information.

1. Sensory Neurons: These neurons send information from sensory receptors (present in skin. Eye, nose, tongue, ear etc.) towards the central nervous system.
2. Motor Neurons : These neurons send information away from the central nervous system to muscle or gland.
3. Inter-neurons : these neurons receives information from sensory neurons and convey response through motor neuron

Resting Membrane Potential

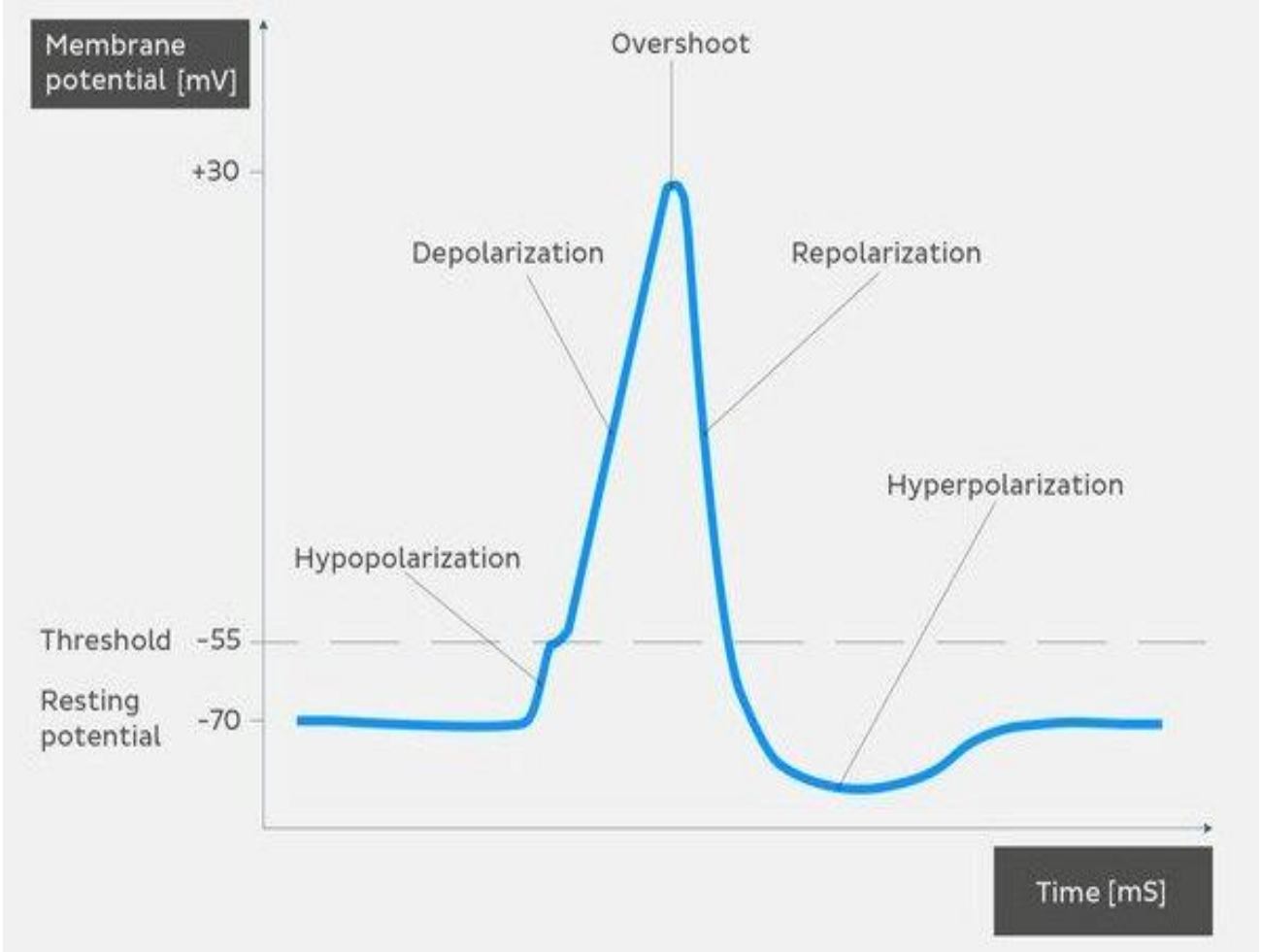
- Resting state of a neuron is the state when it is not transmitting any signal
- At resting state, the inside of the neuron is negative relative to outside extra-cellular fluid. This is because the cell membrane is semi-permeable. It permits only some ion to pass through it and blocks some other ions there by preventing balance of the concentration of different ions on either sides.
- The resting potential arises from two activities:
    1. Na^+K^+ ATPase enzyme located in the cell membrane acts like a pump to push two K^+ ions inside the cell for every three Na^+ ions out of the cell thereby causing net loss of positive charge.
    2. Some potassium channels are leaky which allow diffusion of K^+ ions out of cells
- Thus each resting cell is in a polarized state with an electrical potential across its membrane called Resting Membrane Potential.
- In case of neuron, the resting potential is near about -65mV

Action Potential

- Action potentials are nerve signals. Neurons generate and conduct these signals in order to transmit them to the target tissues. Upon stimulation, they will either be stimulated, inhibited, or modulated in some way .
- An action potential is defined as a sudden, fast, transitory, and propagating change of the resting membrane potential. Only neurons and muscle cells are capable of generating an action potential; that property is called the excitability.
- From an electrical aspect, action potential is caused by a stimulus with certain value expressed in milli-volts Not all stimuli can cause an action potential. Adequate stimulus must have a sufficient electrical value which will reduce the negativity of the nerve cell to the threshold of the action potential.
- There are sub-threshold, threshold, and supra-threshold stimuli.
  - Sub-threshold stimuli cannot cause an action potential.
  - Threshold stimuli are of enough energy or potential to produce an action potential (nerve impulse).
  - Supra-threshold stimuli also produce an action potential, but their strength is higher than the threshold stimuli.

- An action potential is a momentary reversal of membrane potential from -65mV to +40mV, lasting less than 1 ms,, followed by a restoration of its original resting membrane potential.
- Action potential are triggered by any depolarisation of membrane beyond a critical value, called the voltage threshold.
- Action potential are all or none meaning that any stimulation above the voltage threshold results in the action potential response

Action Potential Curve



Different Phases of Action Potential

- An action potential has several phases;
    1. Hypopolarization
    2. Depolarization
    3. Overshoot
    4. Repolarization
    5. Hyperpolarization
- **Hypo-polarization Phase**: It is the initial increase of the membrane potential to the value of the threshold potential.
- **Depolarization Phase**: During this phase the membrane potential exceeds the threshold opens hundreds of voltage-gated sodium channels and around 7000 sodium ions rushes into the cell and consequently the cell becomes more positively charged inside.

- Overshoot Phase. Inside of the cell becomes more and more electropositive during the depolarization phase, until the potential gets closer the electrochemical equilibrium for sodium of +61 mV. The phase of extreme positivity is the overshoot phase. During this phase the sodium ion influx slows down.
- Repolarization Phase: After the overshoot, the sodium permeability suddenly decreases due to the closing of its channels. The overshoot value of the cell potential opens voltage-gated potassium channels, which causes a large potassium efflux, decreasing the cell's electro-positivity. The purpose of this phase is to restore the resting membrane potential
- Hyperpolarization Phase: Repolarization always first leads to hyperpolarization state in which the membrane potential becomes more negative than the resting potential and this period is known as refractory period. But soon after that, the membrane establishes again the values of the resting potential.

Refractory Period

- During the refractory period, the excitable cell cannot produce another action potential. There are two sub-phases of this period, absolute and relative refractoriness.
- Absolute Refractoriness: This overlaps the depolarization and around 2/3 of repolarization phase. A new action potential cannot be generated during this period because all the voltage-gated sodium channels are already opened or being opened at their maximum speed. During early repolarization, a new action potential is impossible since the sodium channels are inactive and need the resting potential to be in a closed state, from which they can be in an open state once again. Absolute refractoriness ends when enough sodium channels recover from their inactive state.
- Relative Refractoriness: During this period the generation of a new action potential is possible, but only upon a suprathreshold stimulus. This period overlaps the final 1/3 of repolarization phase.

Propagation of Action Potential

- An action potential is generated in the body of the neuron and propagated through its axon. Propagation doesn't decrease or affect the quality of the action potential in any way, so that the target tissue gets the same impulse no matter how far they are from the neuronal body.
- The action potential generated at one spot of the cell membrane. It propagates along the membrane with every next part of the membrane being sequentially depolarized. This means that the action potential *doesn't move* but rather causes a new action potential of the adjacent segment of the neuronal membrane.
- The action potential always propagates forward, never backwards. This is due to the refractoriness of the parts of the membrane that were already depolarized, so that the only possible direction of propagation is forward. Because of this, an action potential always propagates from the neuronal body, through the axon to the target tissue.

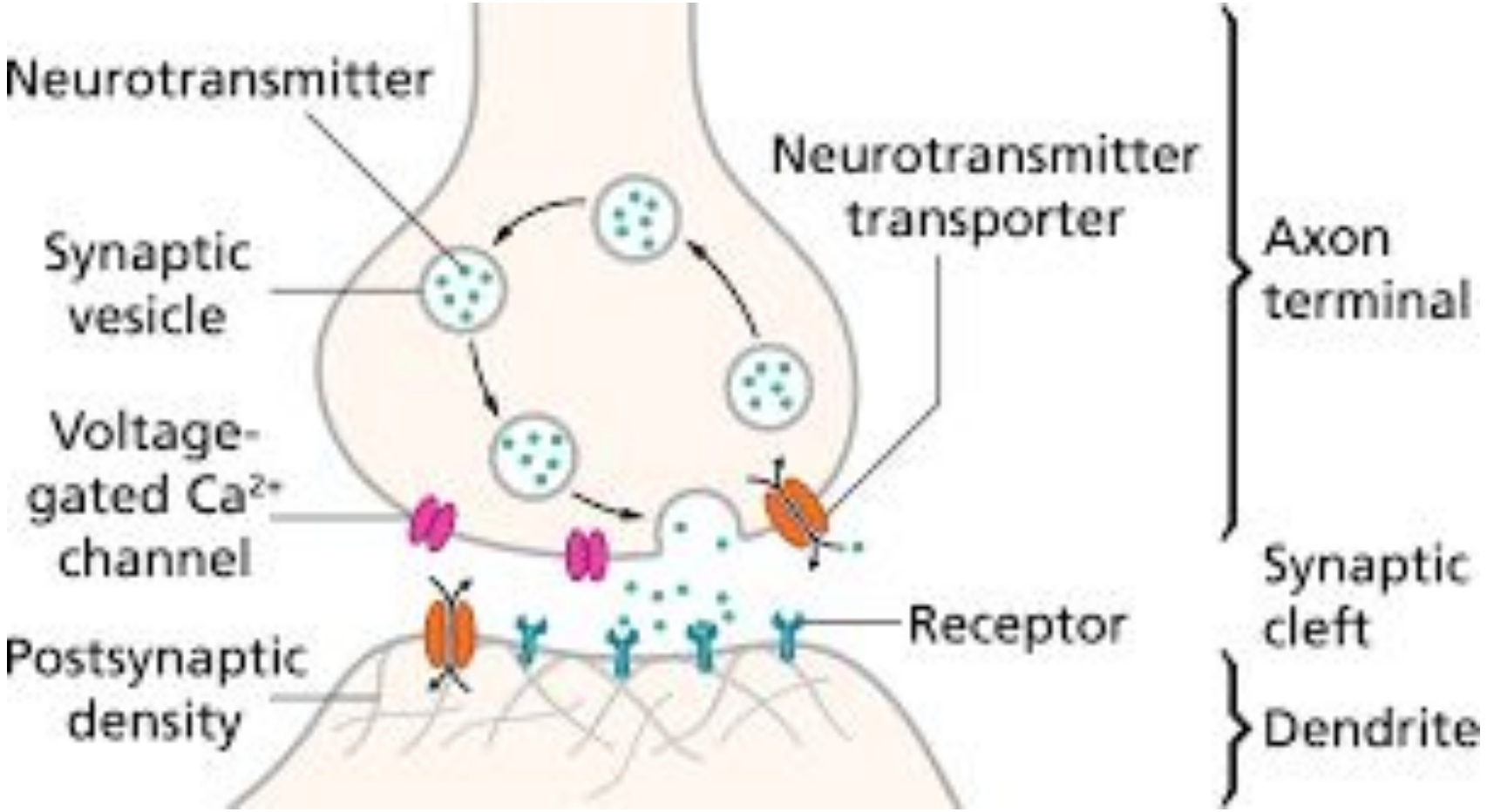
- In non-myelinated axon the action potential are conducted continuously.
- In Myelinated axon the action potential is conducted saltatory(jumping).
  - Sheath of Schwann cells which envelop the axon called myelin sheath. It act as a biological electrical insulator creating a region of high electrical resistance.
  - Nodes of Ranvier separate each Schwann cell from the next. Ion channels are located in the nodes of Ranvier only. Schwann cells prevent continuous conduction
  - Action potential jumps as an electrical current from one node to the next.
  - When the current reaches a node, opens sodium ion channels such that it generate PD large enough to create the current to reach the next node to pass on.
  - It is a very fast form of conduction(130m/s)

SYNAPSE

- The end of an axon can be associated with several dendrires or an axon or soma of other nerve cells or with the site of either muscle or secretory cells: these associations are termed as synapses (highly specialized contacts between nerve cells that transmit signals from the presynaptic neuron to the postsynaptic cell.).
- Depending on the type of target tissue, there are central and peripheral synapses.
  - Central synapses are between two neurons in the central nervous system
  - Peripheral synapses occur between a neuron and muscle fiber, peripheral nerve, or gland.
- Each synapse consists of the:
  1. Pre-synaptic membrane – membrane of the terminal button of the nerve fiber
  2. Post-synaptic membrane – membrane of the target cell
  3. Synaptic cleft – a gap between the presynaptic and postsynaptic membranes
- There are two types of synapses
  1. Electrical synapse
  2. Chemical Synapse

- Electrical Synapse
  - These are specialized for rapid signal transmission
  - The cell are separated by a gap,
  - the synaptic cleft of only 0.2nm, so that an action potential arriving the pre-synaptic side of the cleft sufficiently depolarize postsynaptic membrane to directly trigger its action potential.
- Chemical Synapse
  - These are the common most type of synapse, consists of a bulbous expansion of nerve terminal called synaptic Knob. Synaptic knob contains numerous tiny round structure called synaptic vesicles, each vesicle consists 10,000 molecules of neurotransmitters. The cell are separated by a gap , the synaptic cleft of 20nm.
  - When a wave of depolarization reaches the presynaptic membrane, voltage gated calcium channels open. Outside concentration of calcium ion is so large , so they diffuse inside which stimulate synaptic vesicle to move to terminal membrane to fuse with it. Then ruptures to release the neuro-transmitter chemicals in the cleft.

Chemical Synapse



- These neuro-transmitter molecules rapidly pass to the other side of the gap to combine with specific receptor molecules on the post-synaptic cell to create electric current which is passed further in this cell.
- At the end of the signal, synaptic knob reabsorb some neurotransmitters and enzymes in the synapse, and neutralize others.

- In humans, synapses are chemical. If a neurotransmitter stimulates the target cell to an action, then it is an excitatory neurotransmitter. On the other hand, if it inhibits the target cell, it is an inhibitory neurotransmitter The postsynaptic membrane contains receptors for the neurotransmitters. Once the neurotransmitter binds to the receptor, the ligand-gated channels of the postsynaptic membrane either open or close. Depending on whether the neurotransmitter is excitatory or inhibitory, this will result with different responses.

Postsynaptic Potential

- Postsynaptic potentials are the changes in the membrane potential of the postsynaptic terminal of a chemical synapse
- Postsynaptic potential should not be confused with action potential although their function is to either initiate or inhibit action potential depending up on the type of neurotransmitters released by terminal button of the presynaptic neuron ‘s axon into the synaptic cleft.
- Postsynaptic potential begin to be terminated when the neurotransmitter detaches from the receptors. The receptor is then free to return to its previous structural state.
- Ion channel that had been opened by the receptor when the neurotransmitter was bound to it will now close. Once the channels are closed, the membrane is returned to its equilibrium potential.

Web links for Video Lecture on Action Potential and Synapse

- Action potential in neuron https://youtu.be/oa6rvUJlg7o
- How a synapse works https://youtu.be/OvVl8rOEncE

The function mechanism of a single neuron is known and the networks of neuron still be unknown(remains to be discovered)

Artificial Neural Networks

A Bio-inspired Computing Paradigm

- The artificial neural networks, commonly referred to as "neural networks," has been motivated by the way the human brain computes which is entirely different from the conventional digital computer.
- The brain has the capability to organize its structural constituents, known as *neurons, so as to* perform certain computations (like pattern recognition, perception, and motor control) many times faster than the fastest digital computer in existence today.
- The brain intelligence is due to its inbuilt structure and has the ability to build up its own rules of behavior through experience(learning). which is built up over time. The major development (i.e., hardwiring) of the human brain taking place during the first two years from birth, but the development continues well beyond that stage.

- This development of nervous system is synonymous with a plastic brain: Plasticity permits the developing nervous system to adapt to its surrounding environment. Just as plasticity appears to be essential to the functioning of neurons as information-processing units in the human brain(biological Neural Networks), so it is with (artificial)neural networks made up of artificial neurons.
- A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:
    1. Knowledge is acquired by the network from its environment through a learning process.
    2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

Salient Features of Artificial Neural Networks

- *Generalization Ability:* it refers to the ability of the neural network to produce the reasonable outputs for the inputs which are not encountered during the training (learning).
- *Graceful Performance Degradation (Fault Tolerance):* A neural network, inherently fault tolerant, or capable of robust computation, in the sense that its performance degrades gracefully when a neuron or its connecting links are damaged or recall of a stored pattern is impaired in quality.
- *Nonlinearity:* An artificial neuron can be linear or nonlinear. A neural network, made up of an interconnection of nonlinear neurons, is itself nonlinear. Moreover, the nonlinearity is distributed throughout the network. Nonlinearity is a highly important property, particularly if the underlying physical mechanism responsible for generation of the input signal (e.g., speech signal) is inherently nonlinear

Learning Through Input–Output Mapping

- The learning with a teacher, or supervised learning, involves modification of the synaptic weights of a neural network by applying a set of labeled training examples.
- Each training example consists of a unique input signal and a corresponding desired (target) response.
- The network is presented with an example picked at random from the training set, and the synaptic weights (free parameters) of the network are modified so as to minimize the difference between the desired response and the actual response produced by the network for the input signal.
- The training of the network is repeated for many examples in the training set, until the network reaches a steady state where there are no further significant changes in the synaptic weights. The previously applied training examples may be reapplied during the training session, but in a different order.
- Thus the network learns from the examples by constructing an input–output mapping for the problem at hand.

Artificial Neural Networks

Topic-02:

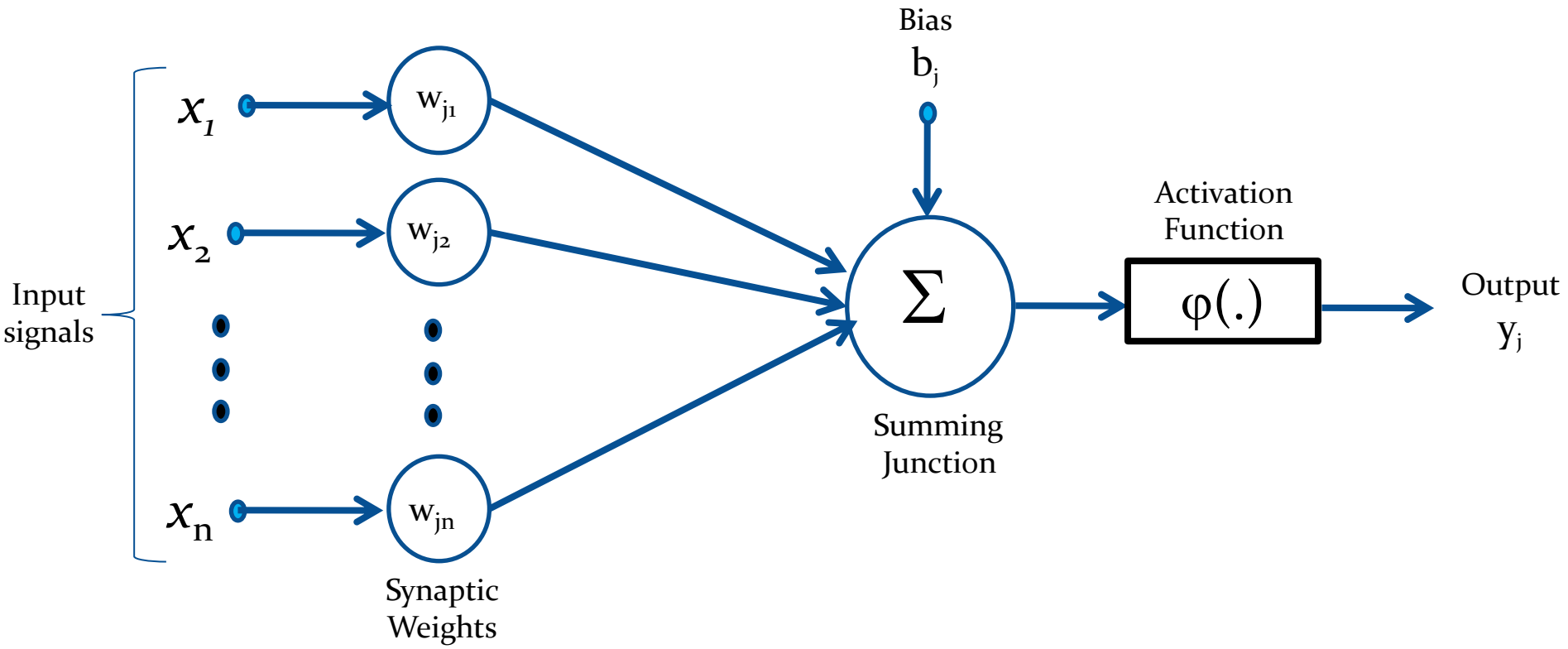
Artificial Neuron Model; Types of Activation Function and McCulloch-Pitts Model

Mathematical Model of a Neuron

A *neuron* is an information-processing unit and fundamental to the operation of a neural network. The *model* of a neuron forms the basis for designing a large family of neural networks. The three basic elements of the neural model:

1. A set of *synapses*, or *connecting links*, each of which is characterized by a *weight* or *strength* of its own. Specifically, a signal x_i at the input of synapse *i* connected to neuron *j* is multiplied by the synaptic weight w_{ji}. *Here it* is important to make a note of the manner in which the subscripts of the synaptic weight w_{ji} are written: the first subscript in w_{ji} refers to the neuron in question, and the second subscript refers to the input end of the synapse to which the weight refers. Unlike the weight of a synapse in the brain, the synaptic weight of an artificial neuron may lie in a range that includes negative as well as positive values.
2. An *adder* for summing the input signals and weighted by the respective synaptic strengths of the neuron; the operations described here constitute a *linear combiner*.
3. An *activation function* for limiting the amplitude of the output of a neuron. The activation function is also referred to as a *squashing function*, in that it squashes (limits) the permissible amplitude range of the output signal to some finite value

Non-Linear Model of a Neuron



The neural model described in the figure includes an externally applied *bias*, denoted by b_j. The bias b_j has the effect of increasing or lowering the net input of the activation function, depending on whether it is positive or negative, respectively.

In mathematical terms, the neuron j is described by writing the pair of equations.

$$u_j = \sum_{i=1}^{n} w_{ji} x_i$$

and

$$y_j = \varphi\left(u_j + b_j\right)$$

Where,

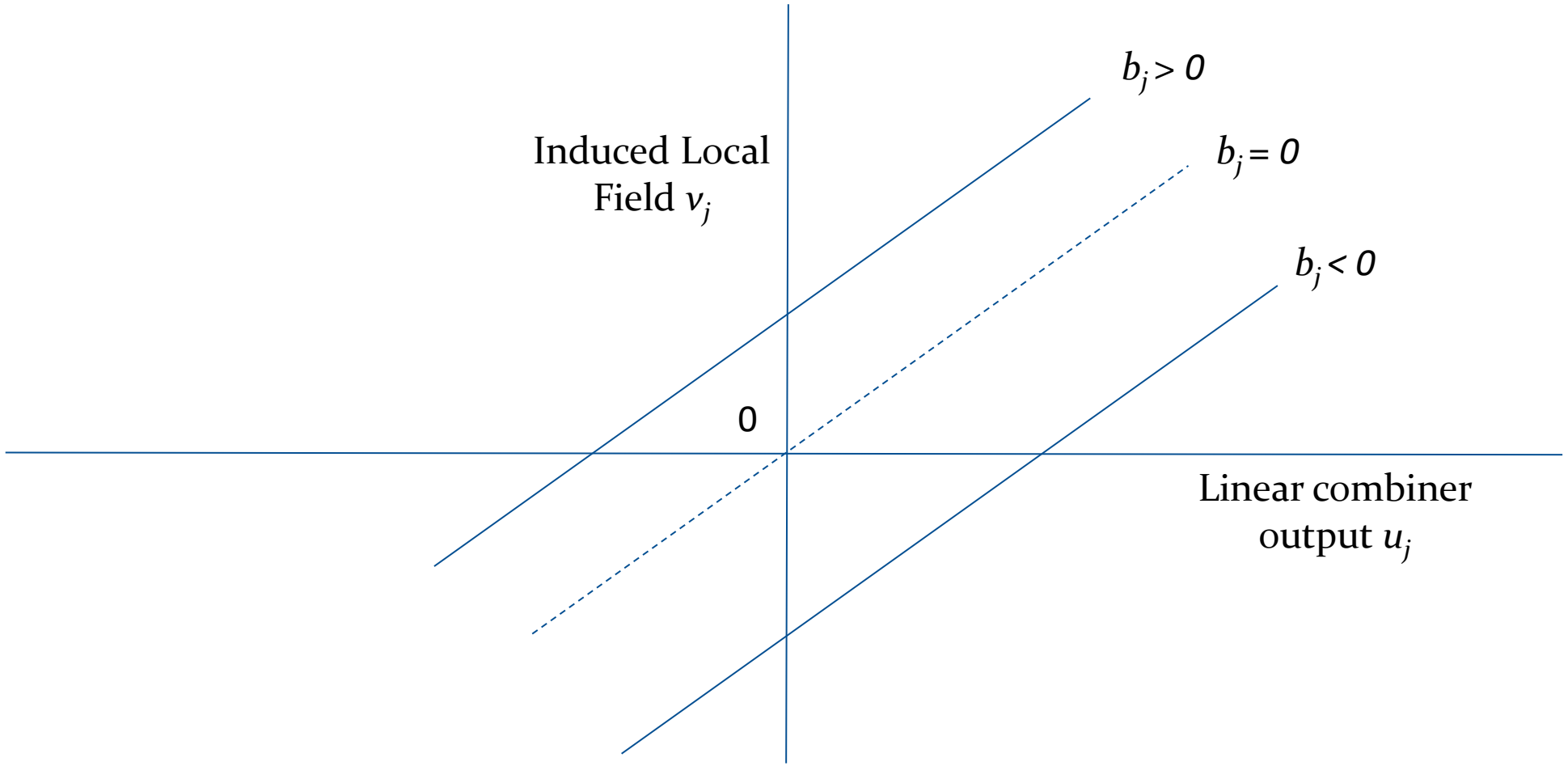
$x_1, x_2, \ldots, x_n$ are the input signals; $w_{j1}, w_{j2}, \ldots, w_{jn}$ are the respective synaptic weights of neuron ***j***

u_j is the *linear combiner output* due to the input signals; b_j is the bias;

$\varphi(\cdot)$ is the *activation function*; and

y_j is the output signal of the neuron.

Affine Transformation produced by the presence of a bias



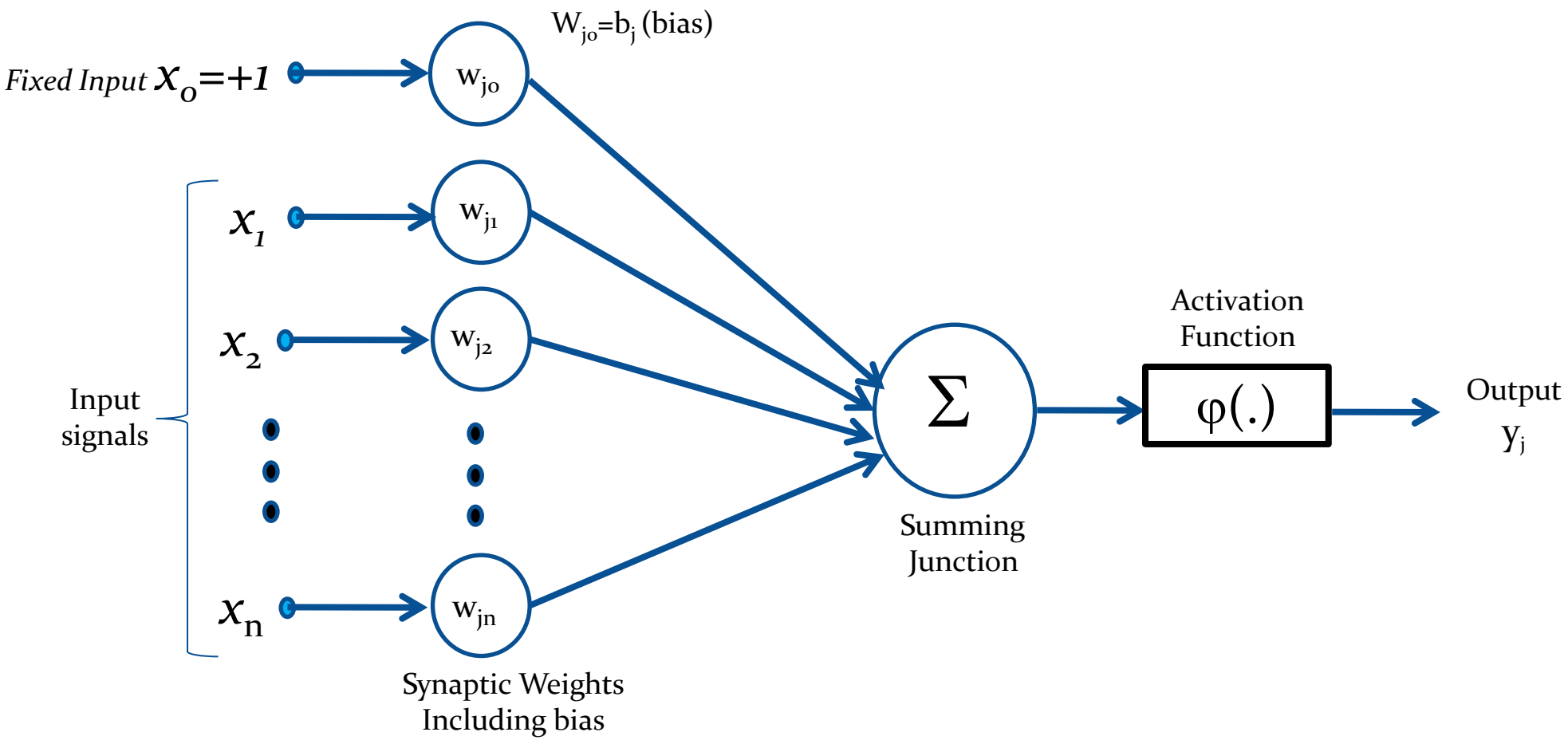
The use of bias *bj* has the effect of applying an *affine transformation* to the output u_j of the linear combiner in the model

$$v_j = u_j + b_j$$

In particular, depending on whether the bias b_j is positive or negative, the relationship between the *induced local field*, or *activation potential*, v_j of neuron j and the linear combiner output u_j is modified in the manner illustrated in the figure hereafter, these two terms are used interchangeably.

Note that as a result of this affine transformation, the graph of v_j versus u_j no longer passes through the origin.

Another Equivalent Non-Linear Model of a Neuron



The bias b_j is an external parameter of neuron j. Now, equivalently it can rewritten as

$$v_j = \sum_{i=0}^{n} w_{ji} x_i$$

and

$$y_j = \varphi(v_j)$$

Here, a new synapse is added with its input x_0=+1 and it weight $w_{j0}=b_j$

Therefore, the model of neuron j reformulate *as shown in the figure with* the effect of the bias is accounted for by doing two things:

1. Adding a new input signal fixed at +1, and
2. Adding a new synaptic weight equal to the bias b_j.

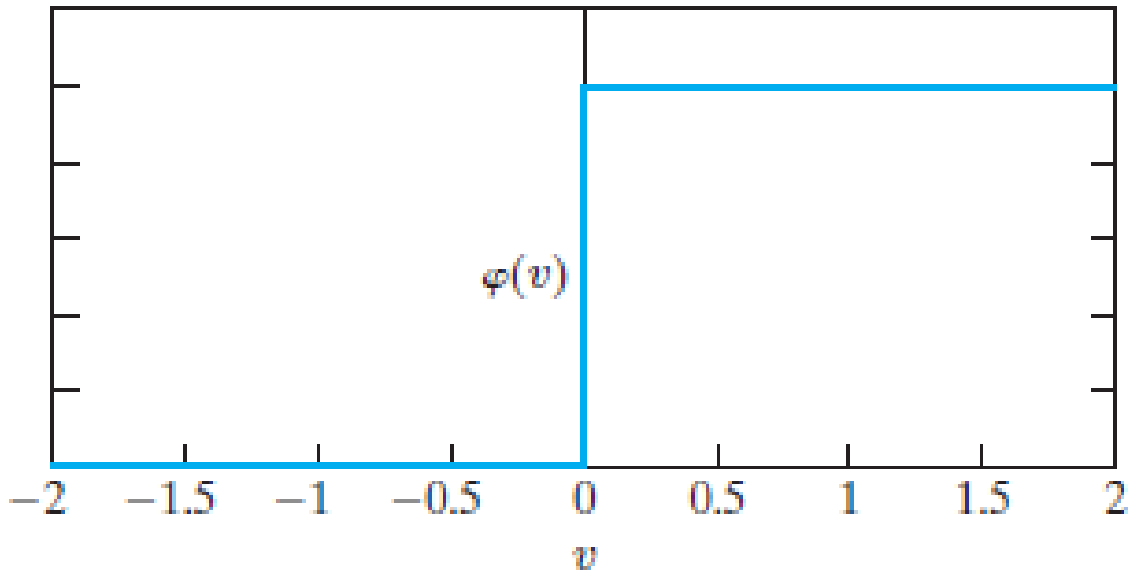
Although these two models are different in appearance, they are mathematically equivalent.

Types of Activation Functions

The activation function, denoted by $\varphi(v)$, defines the output of a neuron in terms of the induced local field v. There are two basic types of activation functions

I. **Threshold Function:** This type activation function is described as

$$\varphi(v) = \begin{cases} 1 & if \quad v \geq 0 \\ 0 & if \quad v < 0 \end{cases}$$



In engineering, this form of a threshold function is commonly referred to as a Heaviside function. Correspondingly, the output of neuron *j* employing such a threshold function is expressed as

$$y_j = \begin{cases} 1 & if \quad v_j \geq 0 \\ 0 & if \quad v_j < 0 \end{cases}$$

$$Where \quad v_j = \sum_{i=1}^{n} w_{ji} x_i + b_j \quad is \quad the \quad local \quad induced \quad field$$

In neural computation, such a neuron is referred to as the McCulloch–Pitts model, in recognition of the pioneering work done by McCulloch and Pitts (1943). In this model, the output of a neuron takes on the value of 1 if the induced local field of that neuron is nonnegative, and 0 otherwise. This statement describes the all-or-none property of the McCulloch–Pitts model.

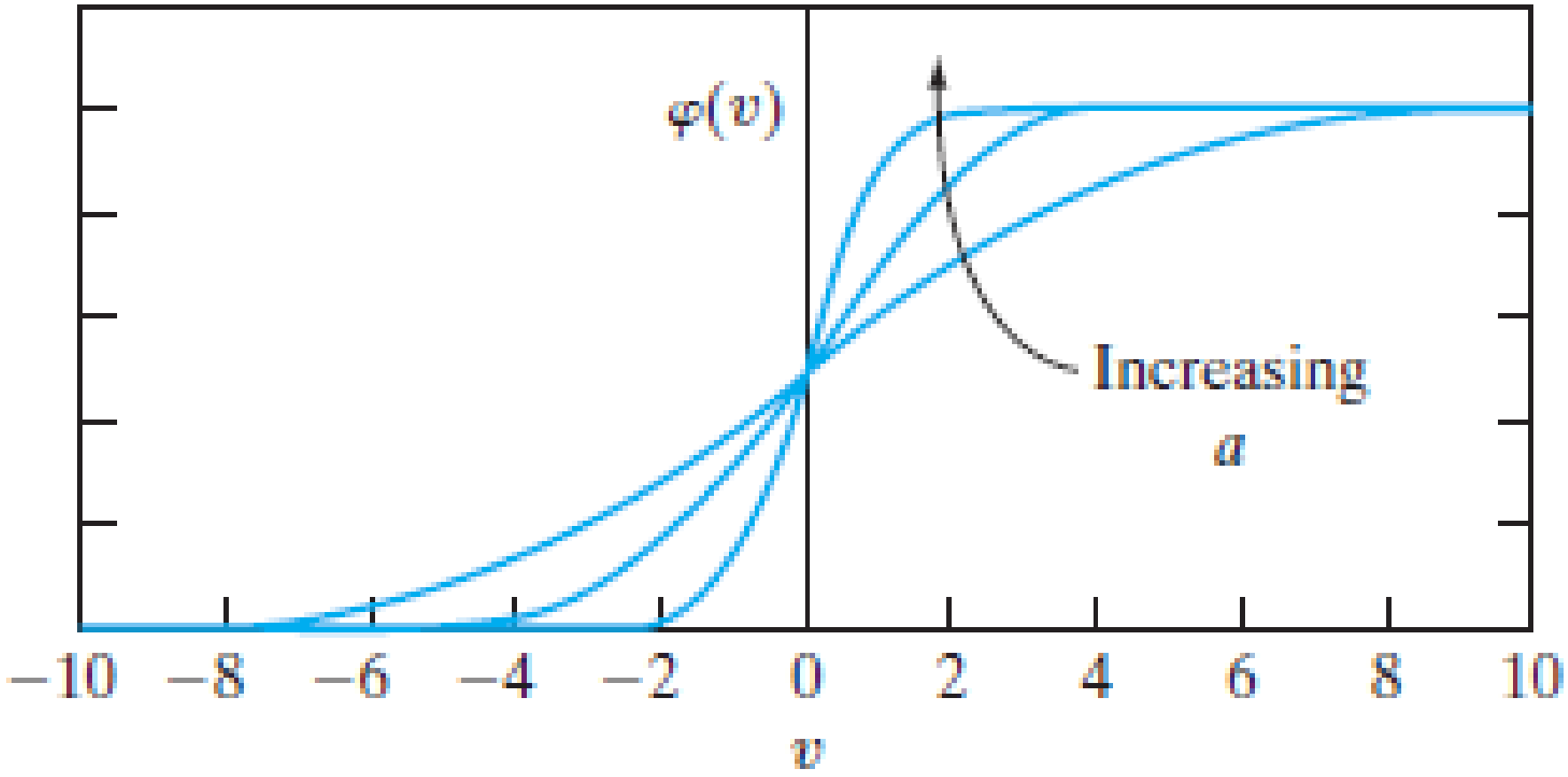
II. **Sigmoid Function**: The sigmoid function graph is "S"-shaped, is by far the most common form of activation function used in the construction of neural networks. It is defined as a strictly increasing function that exhibits a graceful balance between linear and nonlinear behavior. An example of the sigmoid function is the logistic function and is defined by

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$

where a is the slope parameter of the sigmoid function. By varying the parameter a, we obtain sigmoid functions of different slopes. In fact, the slope at the origin equals a/4. In the limit, as the slope parameter approaches infinity, the sigmoid function becomes simply a threshold function.

A threshold function assumes the value of 0 or 1, a sigmoid function assumes a continuous range of values from 0 to 1. Also note that the sigmoid function is differentiable, whereas the threshold function is not

Sigmoid Function Curves With Varying Slope Parameter a



It is sometimes desirable to have the activation function range from -1 to 1, in which case, the activation function is an odd function of the induced local field. The threshold function is now redefined as

$$\varphi(v)=\begin{cases}+1 & if \quad v>0\\ 0 & if \quad v=0\\ -1 & if \quad v<0\end{cases}$$

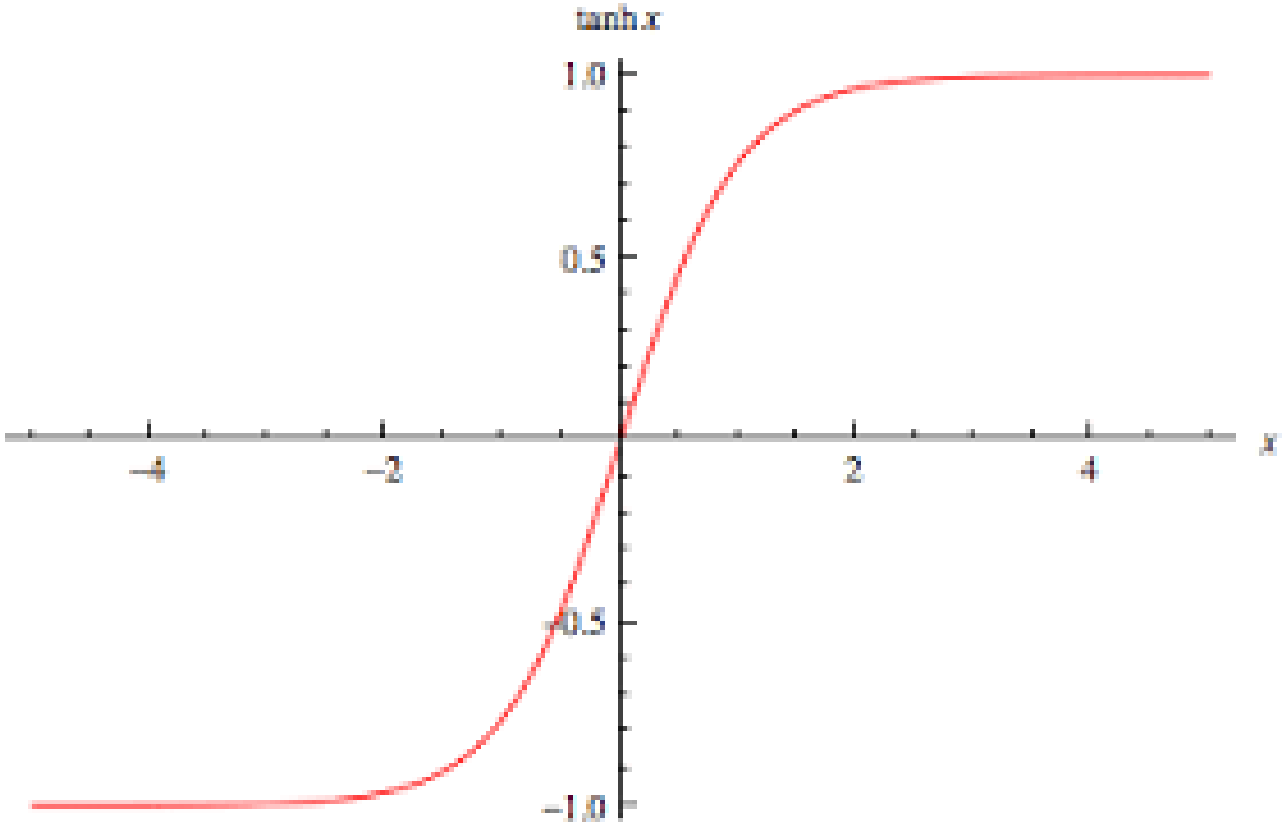
which is commonly referred to as the signum function.

For the corresponding form of a sigmoid function, we may use the hyperbolic tangent function, defined by

$$\varphi(v)=\tanh(v)$$

Allowing an activation function of the sigmoid type to assume negative values may yield practical benefits over the logistic function.

tanh Function Curve



McCulloch-Pitts Model of Neuron(1943)

The early model of an artificial neuron was introduced by Warren McCulloch and Walter Pitts in 1943. This model is also known as linear threshold gate.

- The McCulloch-Pitts Neuron is a discrete time machine. Thus, the McCulloch-Pitts neuron operates on a discrete time scale, t = 0,1,2,3,…
- Here neuron have n number inputs are $I_1, I_2, \ldots, I_n$ and one output y. The linear threshold gate classify a set of input in to two different classes. Thus y is binary.
- The input values I_i^t from the i^{th} pre-synaptic neuron at any instant t may be equal to either **0** or **1** only

- The weights of interconnections w_i are equal to

  +1 for excitatory type connection and

  -1 for inhibitory type connection

- There is an excitation threshold θ associated with the neuron.

- Output y^{t+1} of the neuron at the following instant $t+1$ is defined according to the rule

$$y^{t+1} = 1 \quad \textit{if and only if} \quad S^t = \sum_{i=1}^{n} w_i I_i^t \geq \theta$$

$$\textit{while} \quad w_i > 0 \quad \forall I_i > 0.$$

- The condition "**while $w_i > o \quad \forall \quad I_i \geq o$** " means that activity of a single inhibitory input, i.e. input via a connection with negative weight $w_i = -1$, would absolutely prevents excitation of the neuron at that instant.

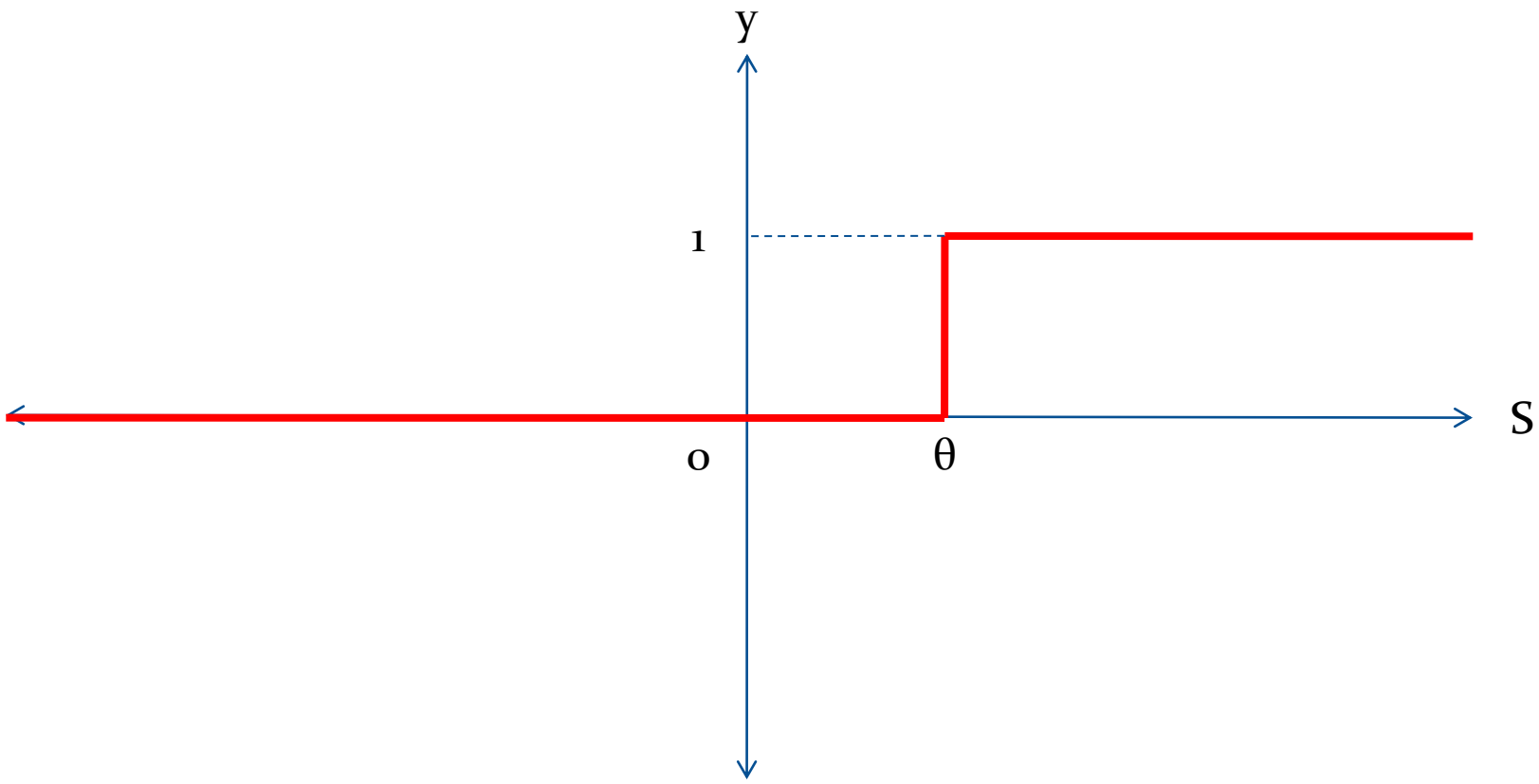
- **Activation function:** The output y^{t+1} is function of the state S^t of the neuron, therefore it also may be written as function of discrete time

$$y^{t+1} = \varphi\left(S^t\right) = \varphi\left(\sum_{i=1}^{n} w_i I_i^t\right)$$

Where $\varphi\left(S^t\right)$ *is threshold activation function*

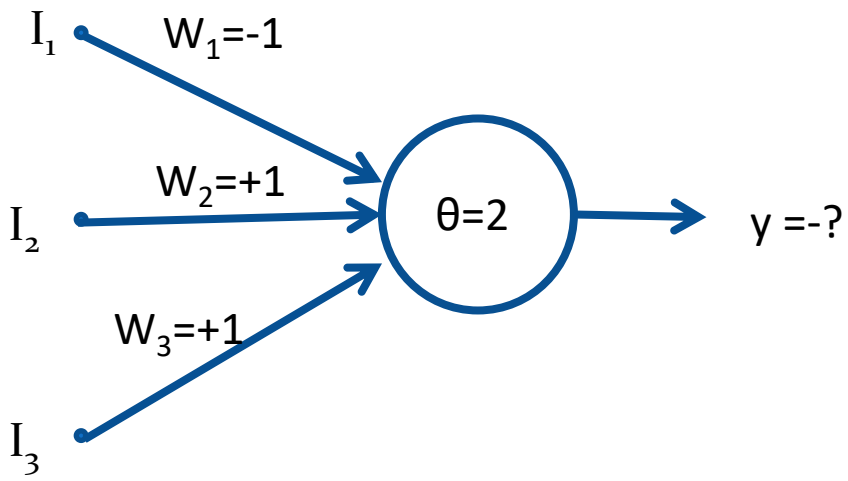
$$\varphi\left(S^t\right) = H\left(S^t - \theta\right) = \begin{cases} 1 & if \quad S^t \geq \theta \\ 0 & if \quad S^t < \theta \end{cases}$$

Linear Threshold Function



MP Neuron Example

- Consider the MP Neuron



Case-I

1) I_1=0, I_2=0, I_3=1
2) All inhibitory connections are silent
3) S = 0×(-1) + 0×1 + 1×1 = 1 < θ
4) S < θ => y=0

Case-II

1) I_1=0, I_2=1, I_3=1
2) All inhibitory connections are silent
3) S = 0×(-1) + 1×1 + 1×1 = 2 = θ
4) S = θ => y=1

Case-III

1) I_1=1, I_2=1, I_3=1
2) There is an inhibitory connection activated
3) y=0

Artificial Neural Networks

Topic-03

Perceptron

Perceptron

This is an early adaptive network which is two layer architecture designed for pattern recognition as well as a research tool for modeling the possible brain mechanism. It was the first algorithmically described neural network

Little History

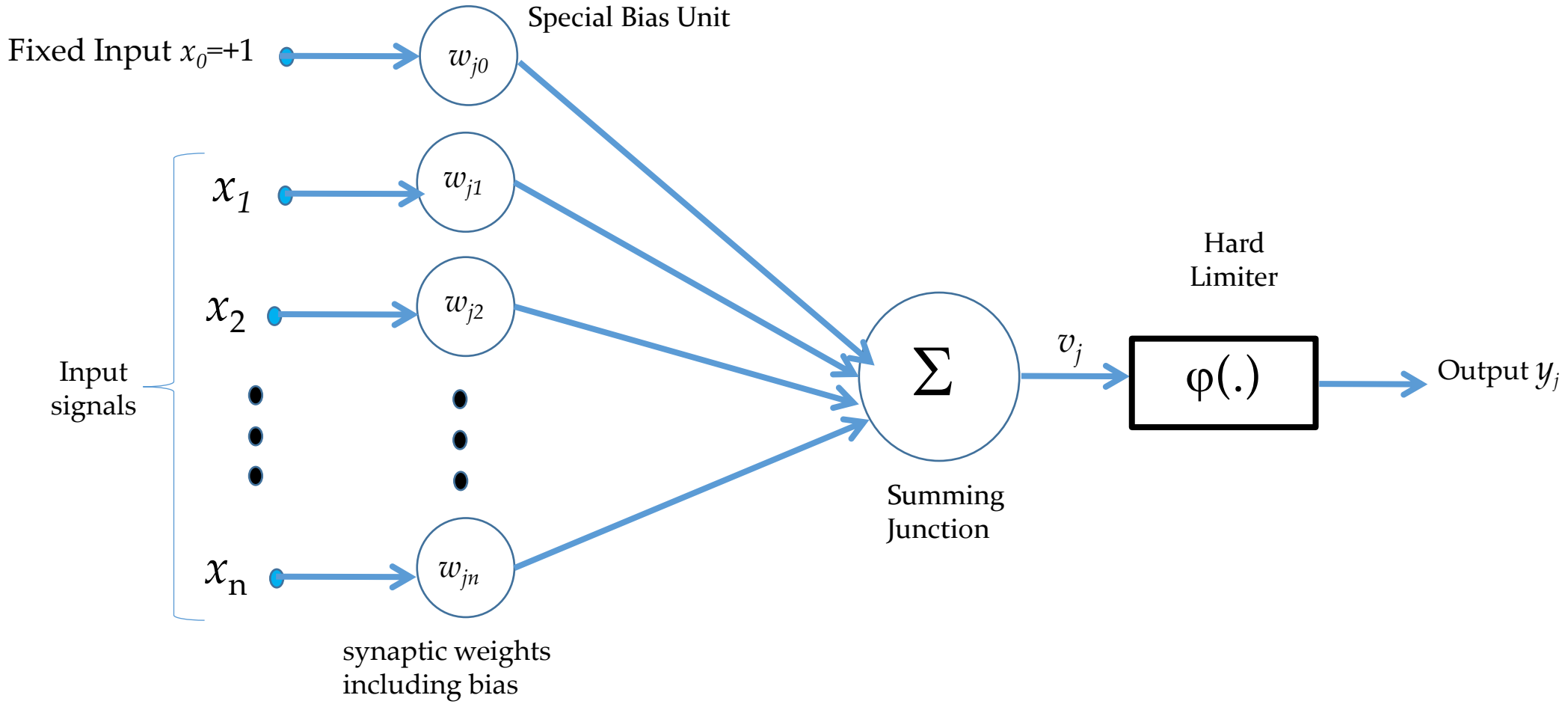
- The first adaptive computing machine was built by Marvin Misnsky and Dean Elmonds in 1951. This machine includes
  - Electric Clutches
  - Control Knobs
  - Gyro Pilot (with memory stored in the position of the control Knob)
- While learning, the machine uses clutch to adjust its own knob. The capabilities of the Minsky / Elmond machine were extremely limited.
- The first meaningful adaptive architecture “The Perceptron” was invented by Frank Rosenblatt in 1958

Basic Architecture

Basically the Perceptron is a pattern mapping architecture learn to classify patterns through supervised learning

1. The pattern it classifies are binary-valued vector
2. The classification categories are also expressed as binary valued vectors
3. The Perceptron is limited to two layers of processing units with a single layer of adaptive weights between the processing units of the output layer and its previous layer(here for two layer architecture it is input layer)
4. Additional hidden layers of processing units may be added but the corresponding layer of interconnection weights are unable to adapt.

Signal-Flow Graph of a Single Processing Unit



- Rosenblatt's Perceptron is built around a nonlinear neuron, consists of a linear combiner followed by a hard limiter(step function).
- The summing node j of the neural model computes a linear combination of the inputs $(x_1, x_2, \ldots, x_n)$ applied to its synaptic weights $(w_{j1}, w_{j2}, \ldots, w_{jn})$ as well as incorporates an externally applied bias(x_0=+1 and w_{j0}=b_j). So the weighted input v_j is defined by

$$v_j = \sum_{i=1}^{n} w_{ji} x_i + b_j = \sum_{i=1}^{n} w_{ji} x_i + w_0 * 1 = \sum_{i=1}^{n} w_{ji} x_i + w_j x_0 = \sum_{i=0}^{n} w_{ji} x_i$$

Where

w_{ji} is the weight associated with the interconnection link to the processing unit j from the processing unit i.

x_i the value of the output by the input unit i

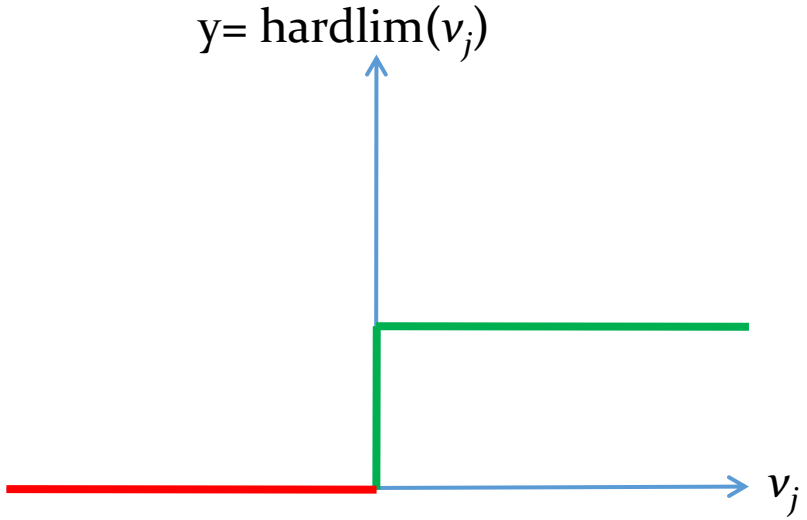
x_0=+1

- The special bias unit behaves as an input with fixed input value of +1. Its connection to the unit j has an interconnection weight w_{j0} which adjust in the same way as all other weights.
- The basic processing unit tests whether the weighted sum v_j is above or below a threshold value using the rule. That is The resulting sum, which is, the induced local field, is applied to a hard limiter(Step Function). Accordingly, the neuron produces an output equal to 1 if the *hard limiter input* v_j is positive, and 0 if it is negative. The hardlim function is defined as

$$y_j = hard\lim(v_j) = \begin{cases} 1 & if \quad v_j \geq 0 \\ 0 & if \quad v_j < 0 \end{cases}$$

- Therefore the inner product of the i^{th} row of the weight matrix with the input vector id greater than or equal to –b, the output will be1, otherwise the output will be 0. thus each neuron in the network divide the input space in to two regions.
- The result of *hardlim*(v_j) is passed along the output line. It effectively separates the input space into two categories by the hyperplane. Where Decision Boundary is defined by all points p for which

$$\sum_{i=1}^{n} w_{ji} x_i + b_j = 0$$



Two-Input and a Single Output Perceptron

$$y = hard\lim(v_1) = hard\lim(W \bullet X + b_1)$$
$$= hard\lim(w_{11}x_1 + w_{12}x_2 + b_1)$$

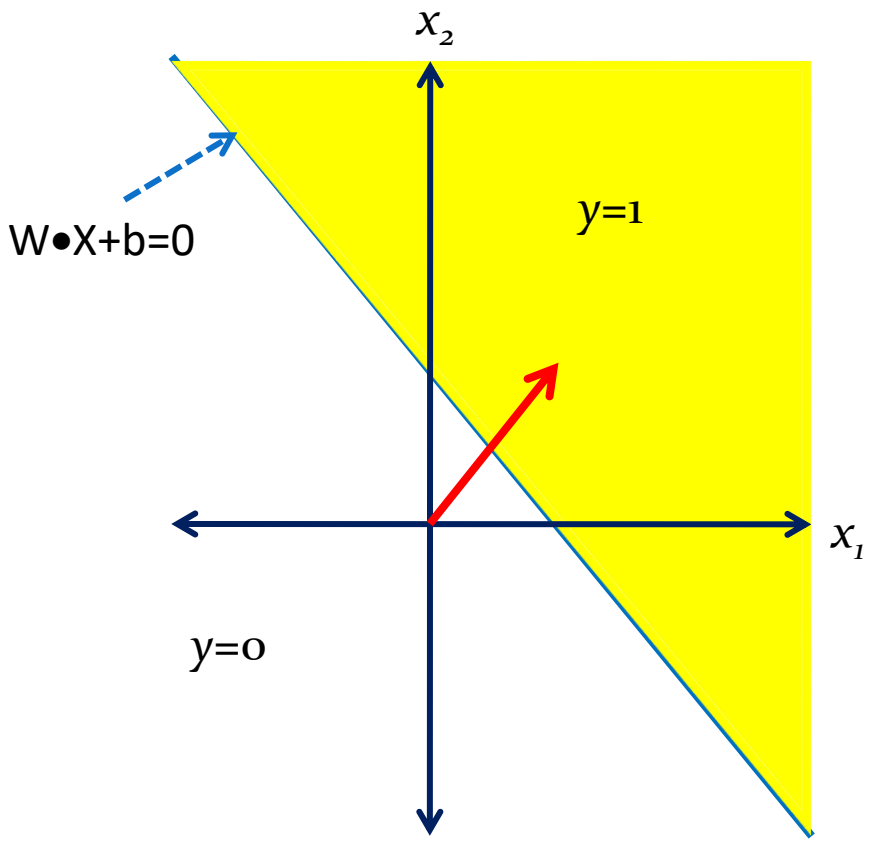
Where W is the weight vector and X input vectorm

- The Decision Boundary is determined by the input vector for which the net input v_1 is zero .
- To make the concept more clear, let us assign the following weights for weights and bias

$$w_{11} = 1 \quad w_{12} = 1 \quad b_1 = -1$$

The Decision Boundarty is then

$$v_1 = W \bullet X + b_1 = w_{11}x_1 + w_{12}x_2 + b_1 = x_1 + x_2 - 1 = 0$$



- The equation defines a line in the input space. On one side of the line, the network output will be 0; on the other side of the line the output will be 1.
- To draw the line, the points where the line intersects with x_1 and x_2 axes to be found.
- To find the x_1 intercepts at the axis $x_2=0$

$$x_1^I = -\frac{b}{w_{12}} = -\frac{-1}{1} = 1 \quad if \quad x_1 = 0$$

- To find the x_2 intercepts at the axis $x_1=0$

$$x_2^I = -\frac{b}{w_{12}} = -\frac{-1}{1} = 1 \quad if \quad x_2 = 0$$

- To find out which side of the boundary corresponds to an output of 1, we need test one point.
- For the point (2,0) the network output will be

$$y = hard\lim(W \bullet X + b) = hard\lim\left(\begin{bmatrix} 1,1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} - 1 \right) = 1$$

- Therefore, the network output will be 1 for the region above and to the right of the decision boundary. This region is indicated by the shaded area in yellow colour.
- It can be found graphically that the boundary is always orthogonal to weight vector W
- For all the points on the boundary, the inner product of the input vector with the weight vector is the same

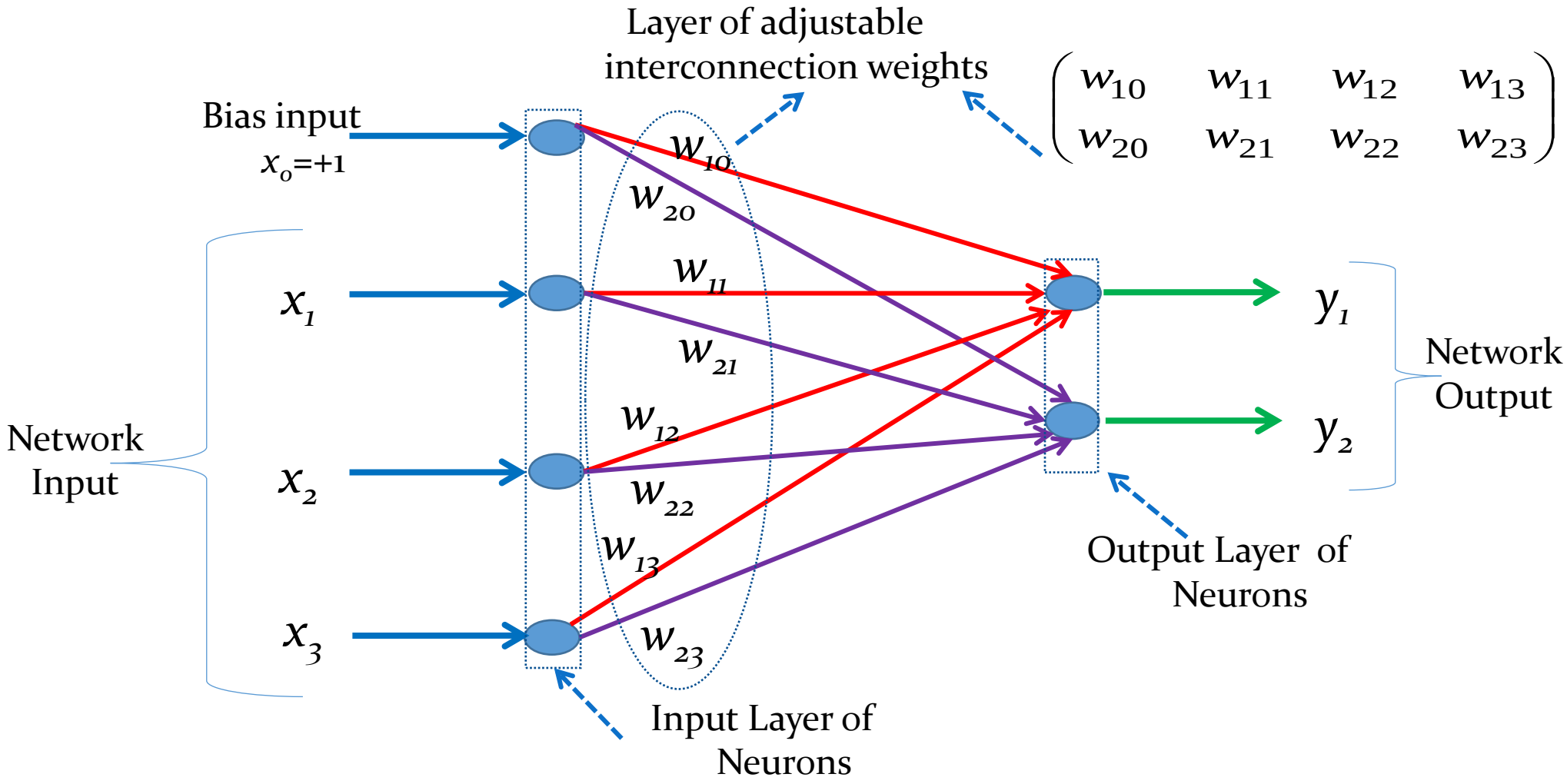
- This implies that these input vectors will have the same projection on to the weight vector,
- so they must lie on the line orthogonal to weight vector.
- Any input vector in the shaded region will have inner product greater than –b
- Any input vectors in the unshaded region will have inner product less than –b
- Therefore the weight vector W will always point towards the region where the neuron output is 1
- After the selection of weight vector with the correct angular orientation, the bias value can be computed by selecting a point on the boundary and satisfying the equation $W \bullet X+b=0$

Two Layer Multiple-Neuron Perceptron

- Two layer Perceptron network has an input layer of processing units(input neurons) and an output layer of processing units(output neurons).
- These two layer are fully interconnected-that means every processing unit in the input layer is connected to all the processing units in the output layer.
- There is a single layer of adjustable interconnection weights
- Each processing unit in the input layer uses its input as its output.
- The processing units in the output layer performs the weighted sum and apply hard limiting function to produce the network outputs

- Perceptron learning rule corrects the interconnection weights so that the network is more likely to produce the desired output.
- Allows only a single layer of weights to be adjusted during learning.
- Each neuron will have its own decision boundary. $WX+ b_j = 0$
- A single neuron can classify input vectors into two categories.
- An *k*-neuron perceptron can potentially classify input vectors into 2^k categories.

Multiple–Neuron Perceptron



Training Perceptron

- Perceptron is trained by using a training set
  - A training set consist of a set of patterns that is presented to the network during the training.
    - Each pattern in the training set is a vector pair consisting of an input (pattern) vector and its corresponding desired output(target) vector
- Training Set: $\{(p_1,t_1), (p_2,t_2), \ldots, (p_m,t_m)\}$
- Example: Input/target pairs for the AND logical gate"

$$\left\{\left(p_1=\begin{bmatrix}0\\0\end{bmatrix},t_1=0\right),\left(p_2=\begin{bmatrix}0\\1\end{bmatrix},t_2=0\right),\left(p_3=\begin{bmatrix}1\\0\end{bmatrix},t_3=0\right),\left(p_4=\begin{bmatrix}1\\1\end{bmatrix},t_4=1\right)\right\}$$

Training Steps:

- During training each pattern in the training set is presented to the network.
- The perceptron processing units then computes the weighted sum of their inputs and uses hard limiting function to produce the network output.
- The network output are compared with the desired output specified in the training set and the difference is computed to readjust the interconnection weights
- The readjustment of weights is done in such a way that the network is more likely to give the desired response next time
- After the training is over the weights will remain fixed and are not readjusted

Updating Weights

- The difference between the target output and the network output is computed as (t_{jp}-y_{jp})
  Where
  t_{jp}-target value of the output unit j for presented input pattern p
  y_{jp}-the output value produced by the output unit j after the presentation of the pattern.
- Since the perceptron uses the binary values (0,1). The result of the difference will be either +1 or -1 only when these values are different

$$\left(t_{jp} - y_{jp}\right) = \begin{cases} 1 & if \quad t_j = 1 \quad and \quad y_j = 0 \\ 0 & if \quad t_j = y_j \\ -1 & if \quad t_j = 0 \quad and \quad y_j = 1 \end{cases}$$

Perceptron Learning Rules

- Learning rules means a procedure for modifying the weights and biases of a network. This procedure may also referred to as training algorithm. A constant is added or subtracted from the appropriate weights during learning

$$w_{ji}^{new} = w_{ji}^{old} + \eta\left(t_j - y_j\right)x_i$$

  - This rule is require a weight to be changed only when the input unit is active and the values of t_j & y_j are different.
  - η is a small fraction called learning rate

Performance Parameter

- During training, each member of the training set is presented to the network individually and upon each presentation, the weights are readjusted. After the entire training set is presented, the set is presented again many times.
- At first the performance improves and stops eventually. At this the network is said to have converged. Following the convergence. Thee are two possibilities
    - The network has learned the training set successfully

      or
    - The network has failrd to learn all of the answers correctly

Learning Curve

- Initially, the adaptable weights all set to small random values. As the weights are adapted during training, the error rate decreases and becomes very low. At this stage the training is stopped. The performance is measured by a root mean-squared (RMS) error value

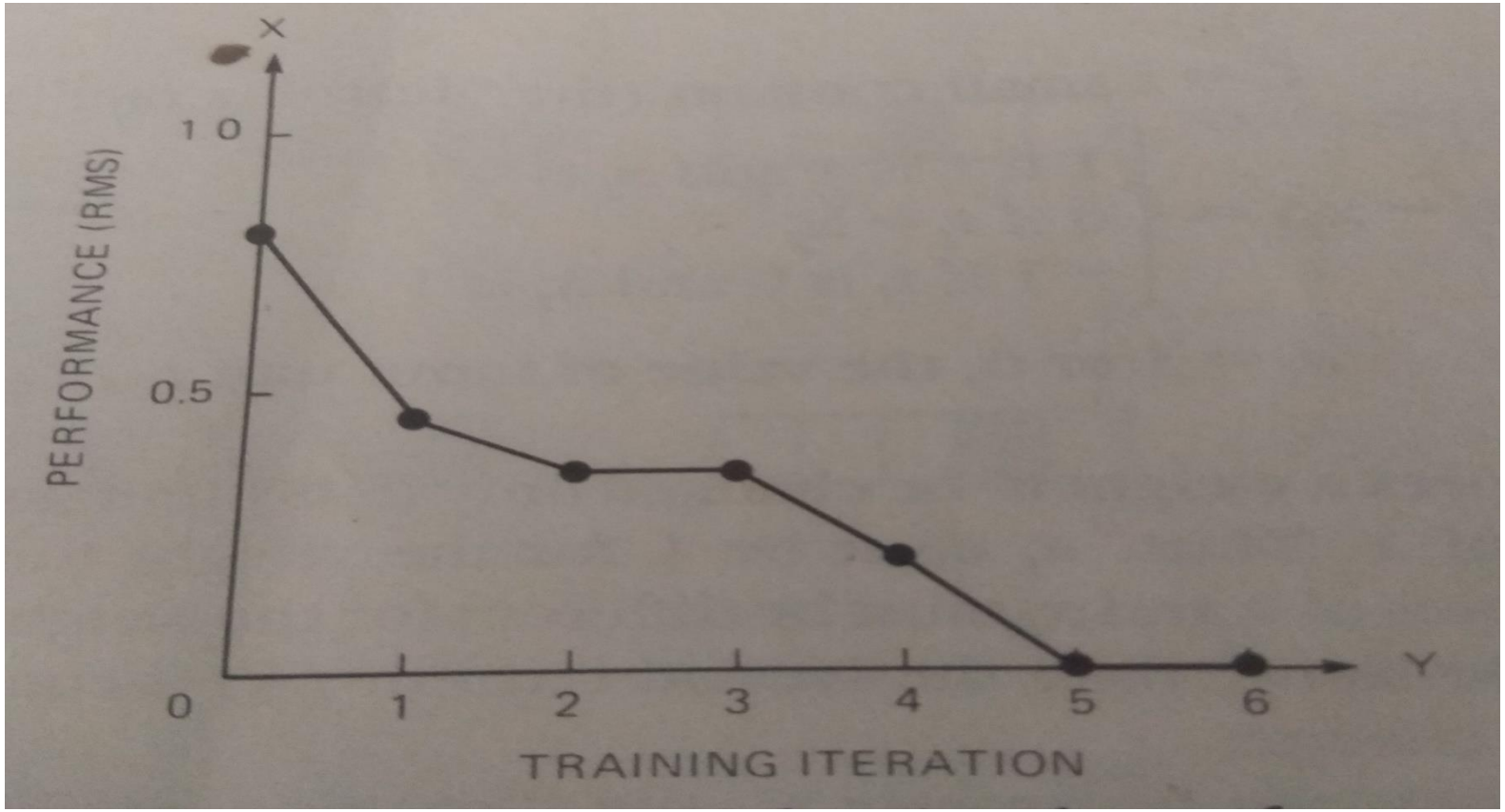
$$\sqrt{\frac{\sum_{P}\sum_{j}\left(t_{jp}-y_{jp}\right)^2}{n_p n_o}}$$

Where

n_p the number of patterns in the training set

n_0 number of units in the output layer

Learning Curve



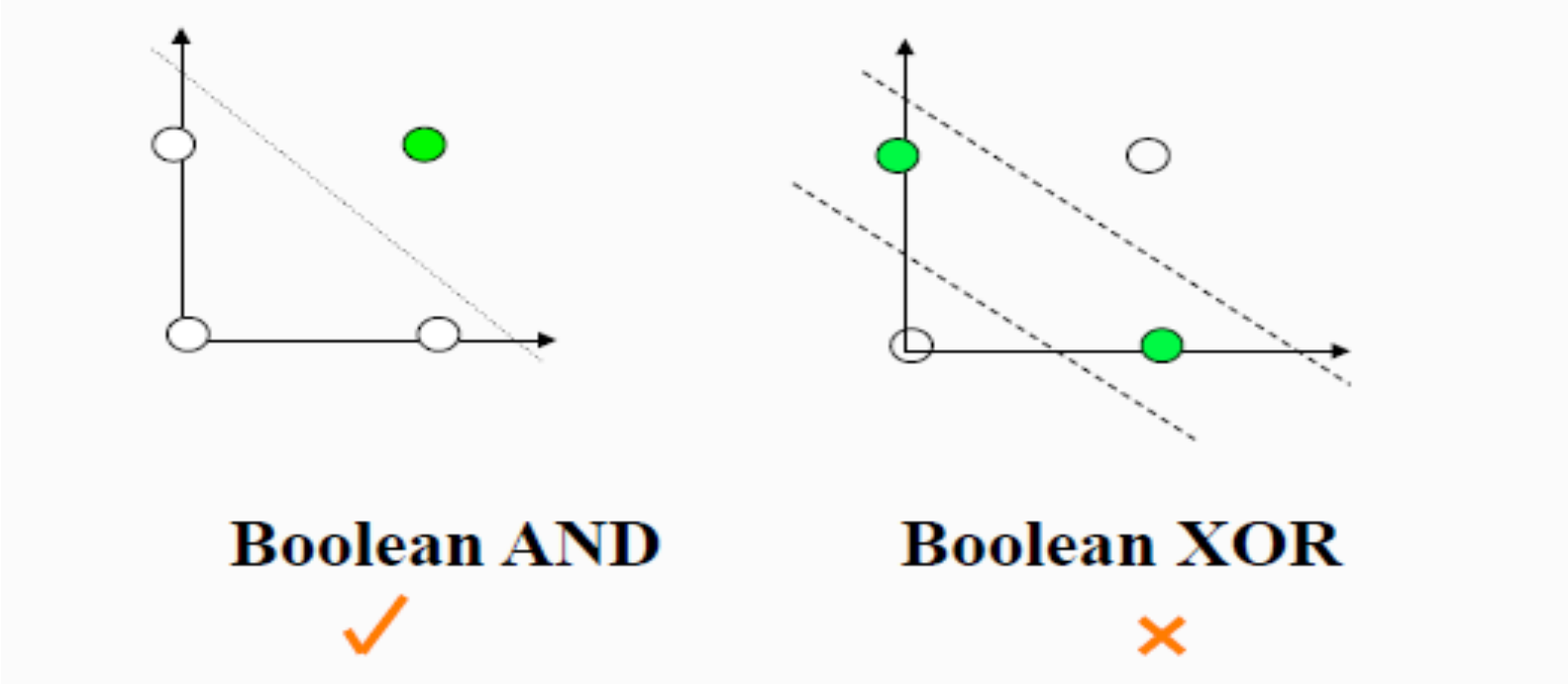
Perceptron Limitations

- The Perceptron is actually capable of learning and performs a variety of classification tasks successfully, but the learning does not always occur and convergence time can be extremely long.
- The crucial limitation of the perceptron is that it does not allow more than one layer of adjustable weights (this is because the early researcher did not find a way of propagating weight correction through multiple layers.
- The chief functional limitation of the perceptron is that an output unit can classify only linearly separable patterns that is the pattern classes can be separated in to two classes by drawing a single line in case of two inputs.
- The concept of linear separability can be extended to three or more dimension where the patterns are divided in to two classes by drawing a plane or hyperplane

Perceptron Limitations

- The Perceptron is actually capable of learning and performs a variety of classification tasks successfully, but the learning does not always occur and convergence time can be extremely long.
- The crucial limitation of the perceptron is that it does not allow more than one layer of adjustable weights (this is because the early researcher did not find a way of propagating weight correction through multiple layers.
- The chief functional limitation of the perceptron is that an output unit can classify only linearly separable patterns that is the pattern classes can be separated in to two classes by drawing a single line in case of two inputs.
- The concept of linear separability can be extended to three or more dimension where the patterns are divided in to two classes by drawing a plane or hyperplane

- A single layer perceptron can only learn linearly separable problems.
  - Boolean AND function is linearly separable,
  - whereas Boolean XOR function is a classic example of prooblem that is not linearly separable.



Artificial Neural Networks
Topic-04

Recurrent Networks

- Jordan Network
- Elman Network
- Hopfield Network
  - Basic structure
  - Updating procedure
  - Convergence
  - Applications: *Associative Memory* and *Optimization Problems*

Recurrent Neural Networks

All the data flow in a feed-forward network do not contain any cycles(when cycles are introduced the network will become recurrent). That is L^{th} layer receive signals from the environment(when $L=1$) or , neurons of the immediate previous $(L-1)^{th}$ layer (when $L>1$) and pass their outputs neuron of the next$(L+1)^{th}$ layer or to the environment (when L is the last layer).

The recurrent neural networks consists of both feed-forward and feedback connections between the layers and neurons forming a complicated dynamic system. The ability of the recurrent network to approximate a continuous or discrete non-linear dynamic system by neural dynamics defined by a system of non-linear differential or difference equations offer the opportunities for applications to adaptive control problems.

From computational point of view, a dynamic neural structure which contains a state feedback may provide more computational advantage than a purely feed-forward structure.

Generalized Delta Rule in Recurrent Networks

The back-propagation learning rule can be easily used for training patterns in recurrent networks. The possibilities can be

- The hidden unit activation values are fed back to an extra set of inputs (Elman Networks).
- The output values are fed back into hidden units(Jordan network).

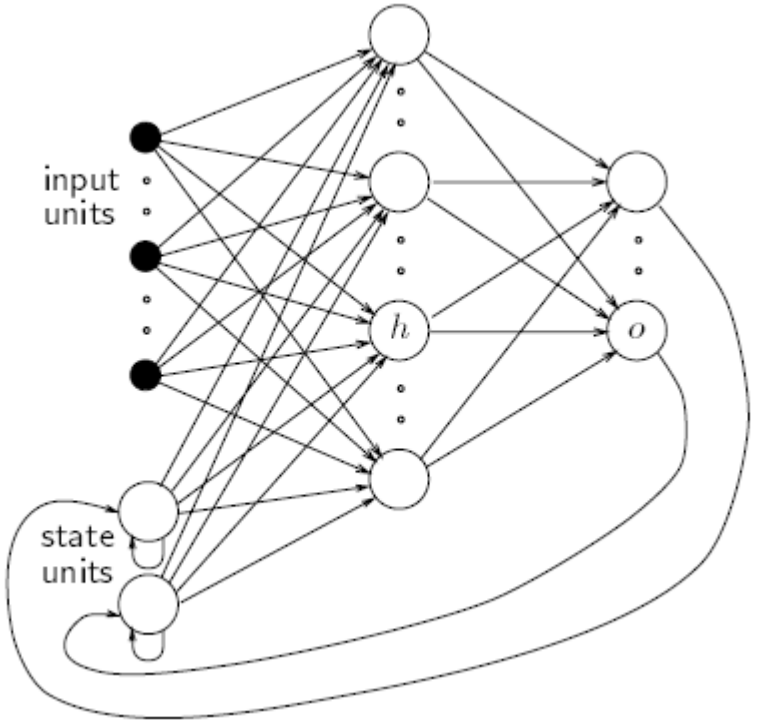
For typical application such as to generate a control command depending on external input which is a time series. Using the feed forward network will result in need of large network and which makes slow and difficult to train

The Jordan Network

- One of the earliest recurrent neural networks was the Jordan network(1986).
- In the Jordan network, the activation values of the output units are fed back into the input layer through a set of extra units called state units.
- There are as many state units as there are output units in the network.
- The connection between the output units and state units have a fixed inter-connection weights of +1.
- The learning take place only in the connections between input and hidden units as well as hidden and output units.
- Thus all the learning rules derived for multi-layer Perceptron can be used to train the Jordan network.

Jordan Network

(The output activation values are fed back to the input layer to a set of extra neurons called the state units)



Elman Network

- The Elman Network was introduced by Elman in 1990.
- In this network a set of context units are introduced, which are the extra input units whose activation values are fed back from the hidden units.
- Thus the network is very similar to the Jordan network except that
  - The hidden units instead of the output units are fed back
  - The extra input units have no self connections.
- The hidden units are connected to the context units with a fixed weight of value +1

Learning of Elman Network

Step-1:The context units are set to 0: t=1

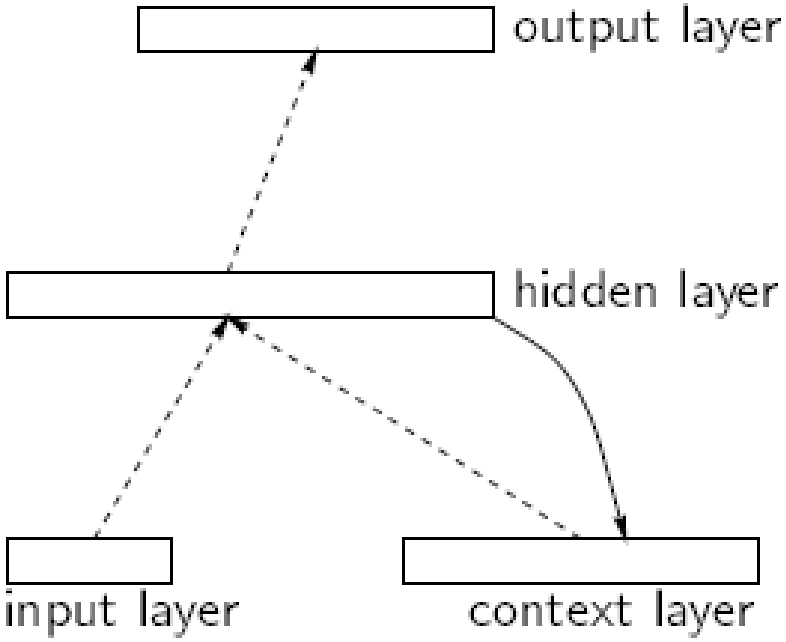
Step-2: The pattern x^t is clamped, the forward calculations are performed once.

Step-3: The back propagation learning rule is applied

Step-4: t← t+1; go to step2

Elman Network

(The hidden units activation values are fed back to the input layer to a set of extra neurons called the context units)



Hopfield Network

- John Hopfield introduced the new single layer architecture in 1982 that has become known as Hopfield Network.
- Hopfield had demonstrated that how more computational capabilities can be incorporated in a network of neuron like components.
  - Associative memory can be implemented with this network.
  - This network can also be used to solve optimization problems.
- The Hopfield network consists of a set of N interconnected neurons, which update their activation values asynchronously and independently of other neurons.
- All neurons are both input and output neurons.
- The activation values are binary.

Basic Structure

- The binary Hopfield network has a single layer of processing units.
- Each processing unit has an *activity value* or *state* *denoted by '**u**'* that is binary
- Originally, Hopfield chose activation values of 0 and 1 but using values -1 and +1 presents some advantages.
    - The advantage of a -1/+1 model over a 0/1 model then is symmetry of the states of the network. For, when some pattern x is stable, its inverse is stable, too,
    - Whereas in the 0/1 model this is not always true (as an example, the pattern 000…00 is always stable, but 111…11 need not be)
    - Similarly, both a pattern and its inverse have the same energy in the -1/+1 model.

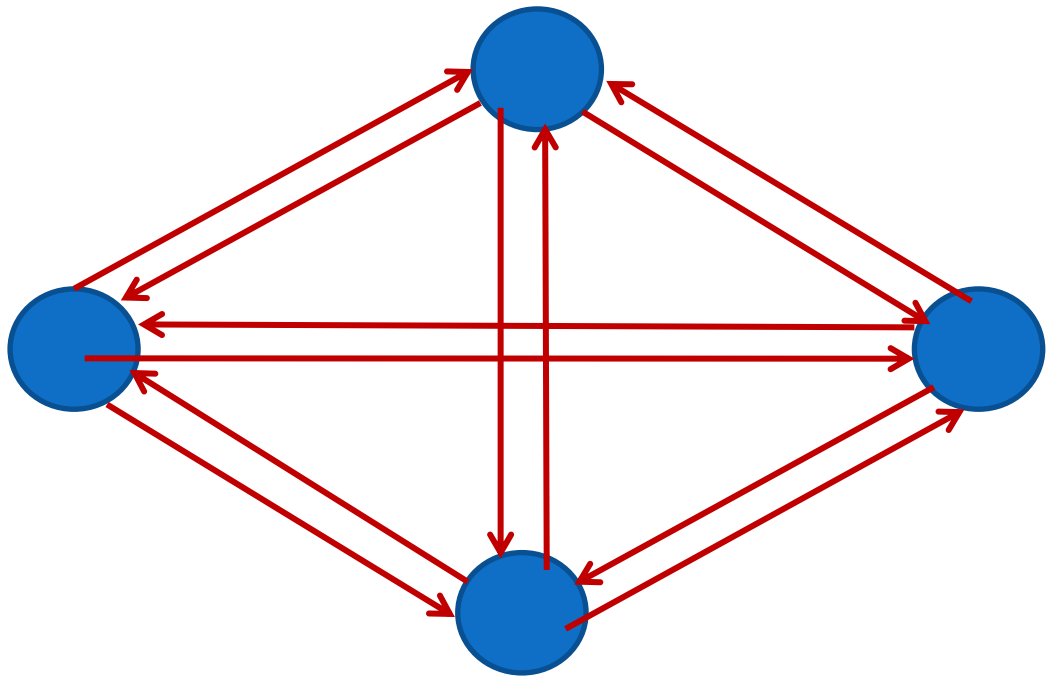
- The entire network is considered to have a state denoted by U at each moment and this state is a vector of 0s and 1s.
- Each entry in the vector corresponds to an individual processing unit in the network.
- Thus at any given moment the state of the network is represented by a state vector.

$$U = (u_1, u_2, \bullet\bullet\bullet, u_n)$$

$$or \quad \text{exp}ressed \quad as$$

$$= (+, -, \bullet\bullet\bullet, +)$$

Example



Network State

U =(u_1,u_2,u_3,u_4)

=(1,1,0,0)

Or

= (+,+,-,-)

- The processing units in the Hopfield network are fully interconnected
    - Each Processing unit is connected to every other units
    - This makes the connections are directed ones
    - Every pair of processing units has a connection in each direction
    - The network with n units will $n(n-1)$ interconnections
- This interconnection topology makes the network *Recursive,* because the out put of each unit is feed in to input of all the other unit in the same layer.
    - This recursive organization will allow the network to relax in to a stable state in the absence of external inputs

- Each interconnection has an associated weight
    - This weight is the scalar value
    - Let T_{ji} denote the weight associated with the interconnection to the j^{th} unit from the i^{th} unit
- In Hopfield network the weights T_{ji} and T_{ij} have same value, that is $T_{ji} = T_{ij}$
- Mathematical Analysis has shown that when this equality is true, the network is able converge, that is it eventually attains a stable state.
- This convergence is necessary in order for it to perform useful computational task such as optimization and associative memory.
- Many networks with unequal weights ($T_{ji} \neq T_{ij}$) are also able to converge successfully.

Weights Updating Procedure

- Initially the network is assigned a state for each processing unit
- An updating procedure updates one unit at a time
- The updating procedure affects the state of each unit, sometimes changing it and sometimes leaving it same.
- Updating single unit
    - The net input $S_j(t+1)$ [*the weighted sum inputs*] for a neuron j at time $t+1$ is given by

$$S_j(t+1) = \sum_{\substack{i=1 \\ i \neq j}}^{n} u_i T_{ji}$$

- The new activation value at time *t+1* for 0/1 model

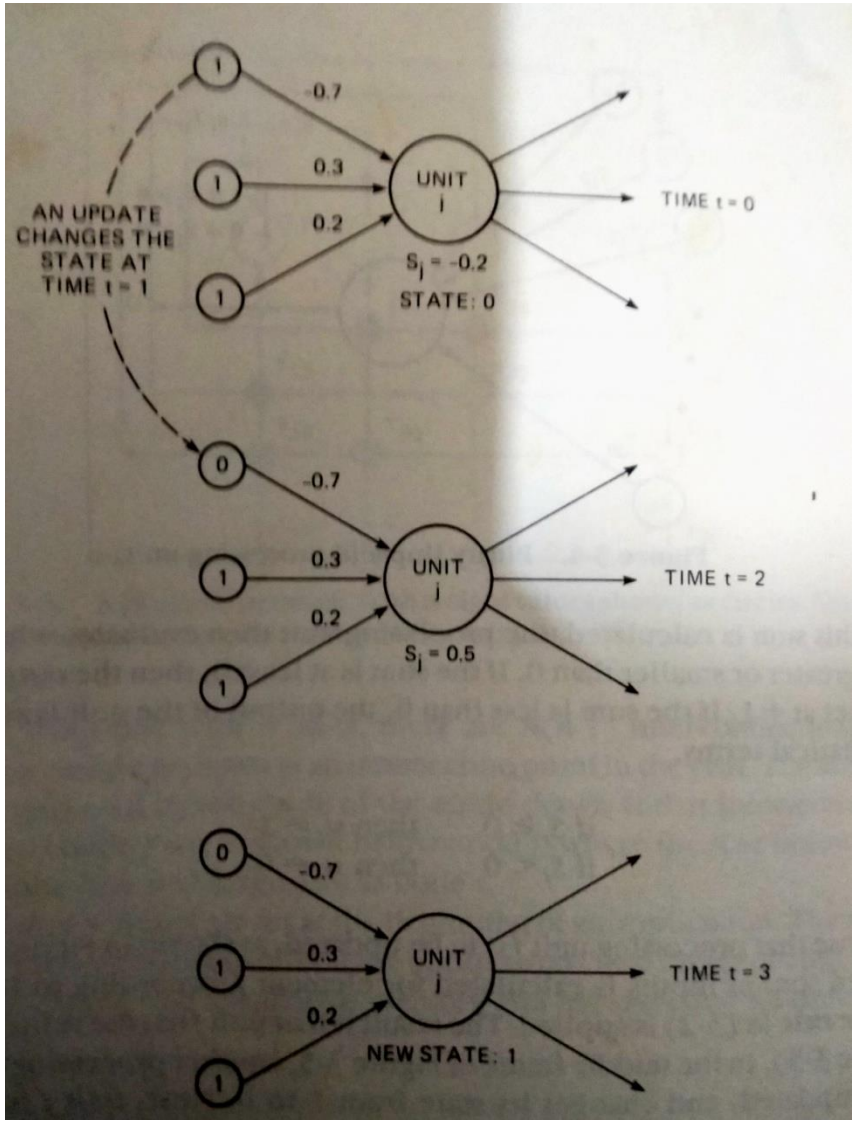
$$u_j(t+1) = \begin{cases} +1 & if \quad S_j \geq 0 \\ 0 & if \quad S_j < 0 \end{cases}$$

- The new activation value at time *t+1* for -1/1 model

$$u_j(t+1) = \begin{cases} +1 & if \quad S_j(t+1) > 0 \\ -1 & if \quad S_j(t+1) < 0 \\ u_j(t) & otherwise \end{cases}$$

- The updating process is to update all the units one by one in a sequence and the repeat the sequence again and again until a stable state is attained

ILLUSTRATION



Convergence

- Each state of the Hopfield network has an associated energy value:

$$E = -\frac{1}{2}\sum_{j}\sum_{i} T_{ji} u_j u_i$$

- This is an objective function to be minimized by the network [This does not represent the real energy of any physical system].
- The successive updating of the Hopfield network provides a convergence procedure whereby the energy of the overall network gets smaller and smaller.
- The network eventually goes in to a stable state, this energy is at minimum and can be either local or global.
- It is possible to prove that each time a processing unit is updated, the energy of the network either stays the same or decreases.

- As a result, this updating procedure will always allow the energy of the network to converge to a minimum.

Proof:

Suppose that the j^{th} unit is the next processing unit to be updated, then the portion of E affected by the processing unit j is given by

$$E_j = -\frac{1}{2}\sum_{\substack{i \\ i \neq j}} T_{ji} u_j u_i$$

$$E_j = -\frac{1}{2} u_j \sum_{\substack{i \\ i \neq j}} T_{ji} u_i$$

- When j^{th} is updated, if there is no change in its sate, then the energy E_j remains the same

- if there is a change in its state, then the difference in energy E_j is

$$\Delta E_j = E_{j\,new} - E_{j\,old} = -\frac{1}{2}\Delta u_j \sum_i T_{ji} u_i$$

$$\Delta u_j = u_{j\,new} - u_{j\,old}$$

$$if \quad u_j \quad changes \quad from \quad 0 \quad to \quad 1$$

$$\Delta u_j = 1 \quad and \quad \sum_i T_{ji} u_i \geq 0 \quad \Rightarrow \Delta E_j \leq 0$$

$$if \quad u_j \quad changes \quad from \quad 1 \quad to \quad 0$$

$$\Delta u_j = -1 \quad and \quad \sum_i T_{ji} u_i < 0 \quad \Rightarrow \Delta E_j < 0$$

$$if \quad u_j \quad changes \quad from \quad 1 \quad to \quad 1 \quad or \quad 0 \quad to \quad 0$$

$$\Delta u_j = 0 \quad and \quad \Delta E_j = 0$$

- The ΔE_j is the product of three quantities and this changes is either negative or zero, no matter what change there is in the state of the unit j upon updating.
- So the network is guaranteed to converge with E taking lower and lower until the network reaches a steady state.

Hopfield Network as Associative Memory

- A primary application of the Hopfield binary network is an associative memory
- Each memory is represented by a binary vector (either in +1/-1 model or a 0/1 model)
- Each memory is a state of the network that corresponds to a minimum in the terrain defined by the network energy.
- When a network start at an initial state, the updating procedure moves to a minimum energy state. That minimum then is expected to corresponds to one of the memories of the network.
- Thus the network may converge to the stored memory that is most similar or most accessible to the initial state.

Setting of interconnection weights

- The number of processing units is equal to the number of entries in the pattern to be stored.
- The weights of the interconnections between the neurons have to be thus set that the state of the system corresponding with the patterns which are to be stored in the network are stable. (these states can be seen as dips in energy space).
- When the network is cued with a noisy or incomplete pattern, it will render the incorrect or missing data by iterating to a stable state which is in some sense near to the cued pattern

- First the notation for the patterns to be stored must be defined.
- A pattern p is denoted by

$$X_p = \left(x_1^p, x_2^p, ..., x_i^p, ..., x_n^p\right)$$

Where

x_i^p *is the* i^{th} *entry in the pattern vector* p

and

There are m *patterns in total* $\left(X_1, X_2, ..., X_p, ..., X_m\right)$

- The Hebb's rule can be used to store m patterns

$$T_{ji} = \begin{cases} \sum_{p=1}^{m} x_i^p x_j^p & if \quad i \neq j \\ o & otherwise \end{cases}$$

$$T_{ji} = \begin{cases} \sum_{p=1}^{m} (2x_i^p - 1)(2x_j^p - 1) & if \quad i \neq j \\ o & otherwise \end{cases}$$

  - The first rule is applicable for +1/-1 model
  - The second rule is applicable for 0/1 model

If x_i^p *and* x_j^p *are equal* T_{ji} *is increased by* 1

If x_i^p *and* x_j^p *are not equal* T_{ji} *is decreased by* 1

- This increment/decrement process is done for all pairs *i* and *j* (i≠ j) in all pattern X_p
- One can add a pattern by doing appropriate increment/ decrement process for the new pattern
- The network gets saturated very quickly and about 0.15 n can only be stored.

Two Problems associated with storing too many patterns:

  - The stored patterns become unstable.
  - Spurious stable state(does not correspond with stored pattern) may appear.
- Both the patterns and its inverse have same energy in the +1/-1 model

Hopfield Network for Solving the Optimization Problems

- An interesting application of the Hopfield network with graded response arises in a heuristic solution to the NP-complete travelling salesman problem (Garey & Johnson, 1979). In this problem, a path of minimal distance must be found between n cities, such that the beginning and ending points are the same.
- Hopfield and Tank (Hopfield & Tank, 1985) use a network with n × n neurons. Each row in the matrix represents a city, whereas each column represents the position in the tour. When the network is settled, each row and each column should have one and only one active neuron, indicating a specific city occupying a specific position in the tour.

- Solving optimization problems with the Hopfield network requires careful and adequate choice of the energy function, i.e. weights T_{ij} . Function E must be determined in such a way that its minima correspond to solutions of the problem considered. So, to ensure a correct solution, the following energy must be minimized

$$E_1 = \frac{A}{2}\sum_{x=1}^{n}\sum_{i=1}^{n}\sum_{\substack{j=1 \\ j\neq i}}^{n} u_{xi}u_{yj} + \frac{B}{2}\sum_{i=1}^{n}\sum_{x=1}^{n}\sum_{\substack{y=1 \\ y\neq x}}^{n} u_{xi}u_{yj} + \frac{C}{2}\left(\sum_{x=1}^{n}\sum_{i=1}^{n} u_{xi} - n\right)^2$$

Where, the activation value u_{xi} =1 indicates that city x occupies the i^{th} place in the tour.

A, B and C are constants

- The first and second terms in the equation are zero if and only if there is a maximum of one active neuron in each row and column, respectively.
    - First term allows only one visit to each city. This term is small when there is only a single 1 in a given row.
    - The second term does not allow the sales person to be in two cities at the same time. This term is small when there is only a single 1 in a column.
- The third term is zero if and only there are exactly *n* active neurons.
    - The third term allows only n cities to appear in the itinerary. This term is small when there is exactly *n* number of 1s in the matrix.

- To minimize the distance of the tour, an extra term fourth term of the expression E_2 is added to the energy E_1 .

$$E_2 = \frac{D}{2} \sum_{x=1}^{n} \sum_{\substack{y=1 \\ y \neq x}}^{n} \sum_{i}^{n} d_{xy} u_{xi} \left(u_{y,i+1} + u_{y,i-1} \right)$$

Where

d_{xy} is the distance between cities x and y

D is a constant

- This expression for distance sum has to be minimized.

- The weights are set as follows

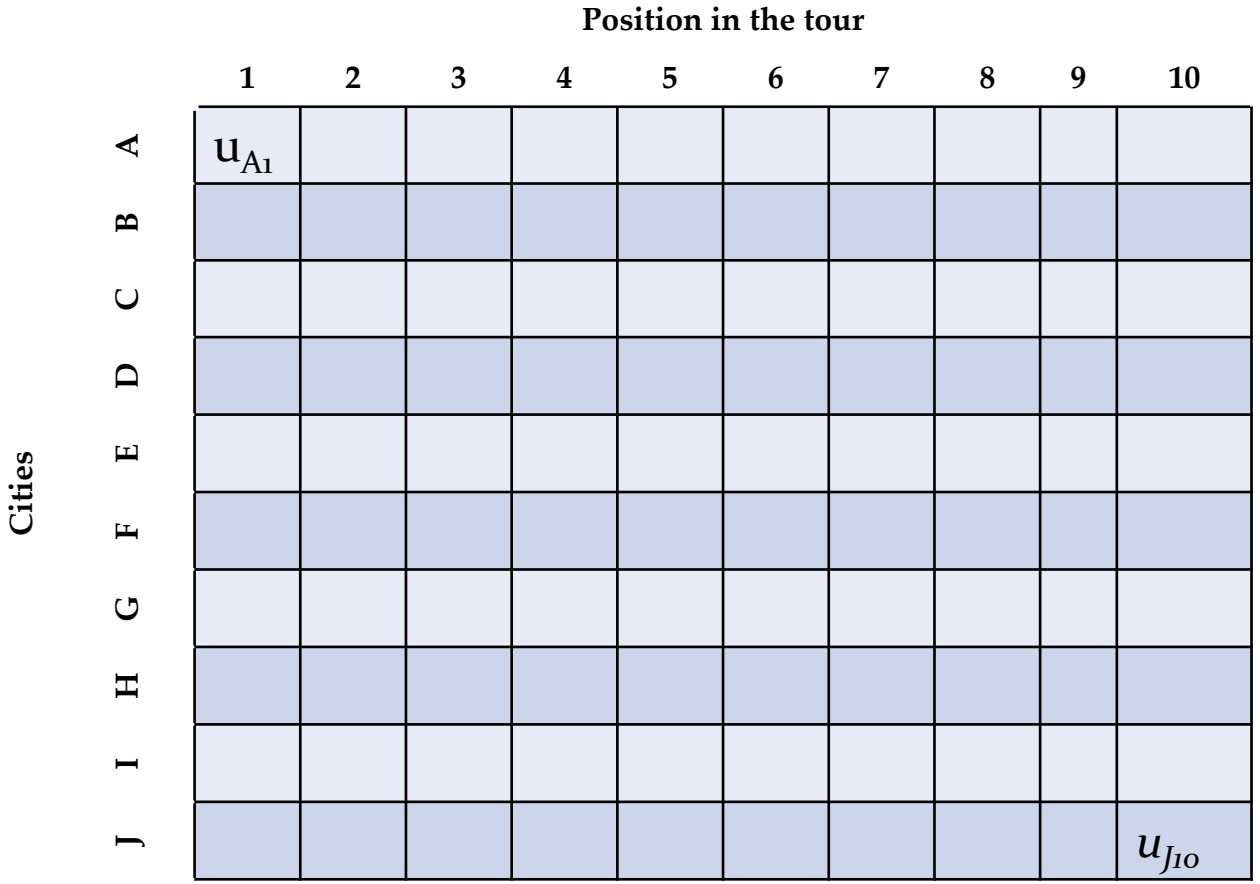
$$
\begin{aligned}
T_{Xi,Yj} = \quad & -A\delta_{XY}\left(1-\delta_{ij}\right) && \textit{inhibitary connections within each row} \\
& -B\delta_{ij}\left(1-\delta_{XY}\right) && \textit{inhibitary connections within each column} \\
& -C && \textit{global inhibition} \\
& -Dd_{XY}\left(\delta_{J,I+1}+\delta_{J,I-1}\right) && \textit{data term}
\end{aligned}
$$

Where

$$
\delta_{ij} = \begin{cases} 1 & \textit{if} \quad i = j \\ 0 & \textit{otherwise} \end{cases}
$$

- Finally, each neuron has an external bias unit Cn

Hopfield network is constructed with the same number of processong as there are entries in the matrix. For example, 10 cities TSP would require a Hopfield net with 100 (10^2) units. Tour can be represented by matrix of 0s and 1s



Artificial Neural Networks
Topic-05

Back-Propagation Networks

Back-Error Propagation Network
Back-Propagation Network
Multi-Layer Perceptron

- It is the most widely used neural net paradigm and has been applied successfully in broad range of application areas [such as pattern recognition, medical diagnosis, speech recognition, etc.]
- The conceptual basis of back-propagation network was first introduced by Paul Werbos in 1974 and independently reinvented by David Parker in1982. it was presented to a wide readership by Rumelhart and McClelland in 1986.

Architecture

- Typically back-propagation network employs three or more layers of processing units.
    - The first layer of units is the (single) input layer-the only units in the network that receive external input. The size of this layer that is the number of input units is equal to the size of the input pattern.
    - The next layers are (multiple) hidden layer, in which the processing units are interconnected to layers above and blow.
    - The last layer is the (single) output layer. The size of this layer that is the number of output units is equal to the size of the desired target pattern.
- The network is fully interconnected: Each processing unit is connected to every unit in the layer above and in the layer below.
- Units are not connected other units in the same layer.
- Back-propagation networks do not have to be fully interconnected.

Figure 5-1: A Typical Three Layered Fully Interconnected Back-propagation Network

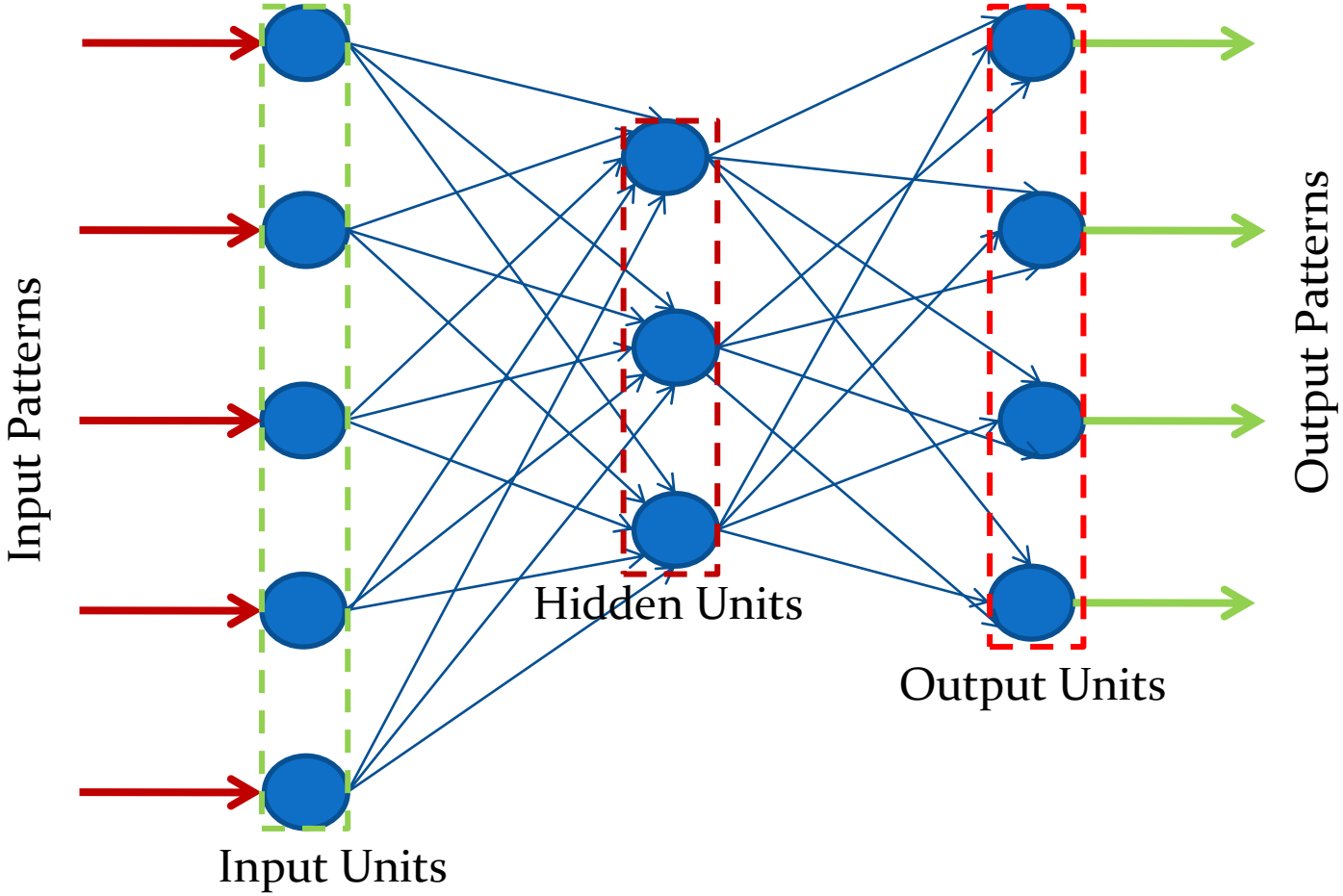
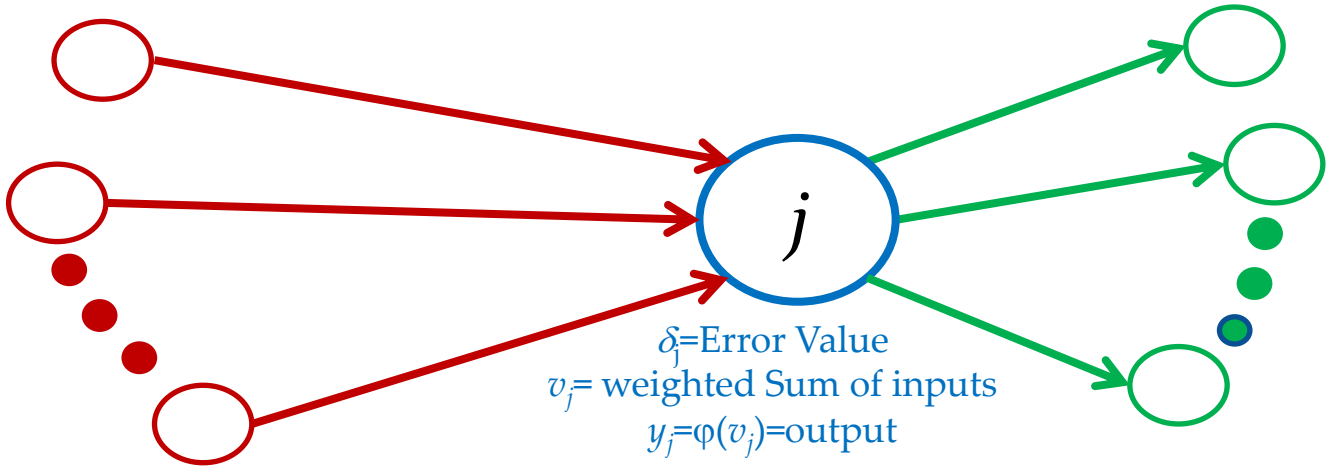


Figure 5-2: The Basic Back-Propagation Processing Unit

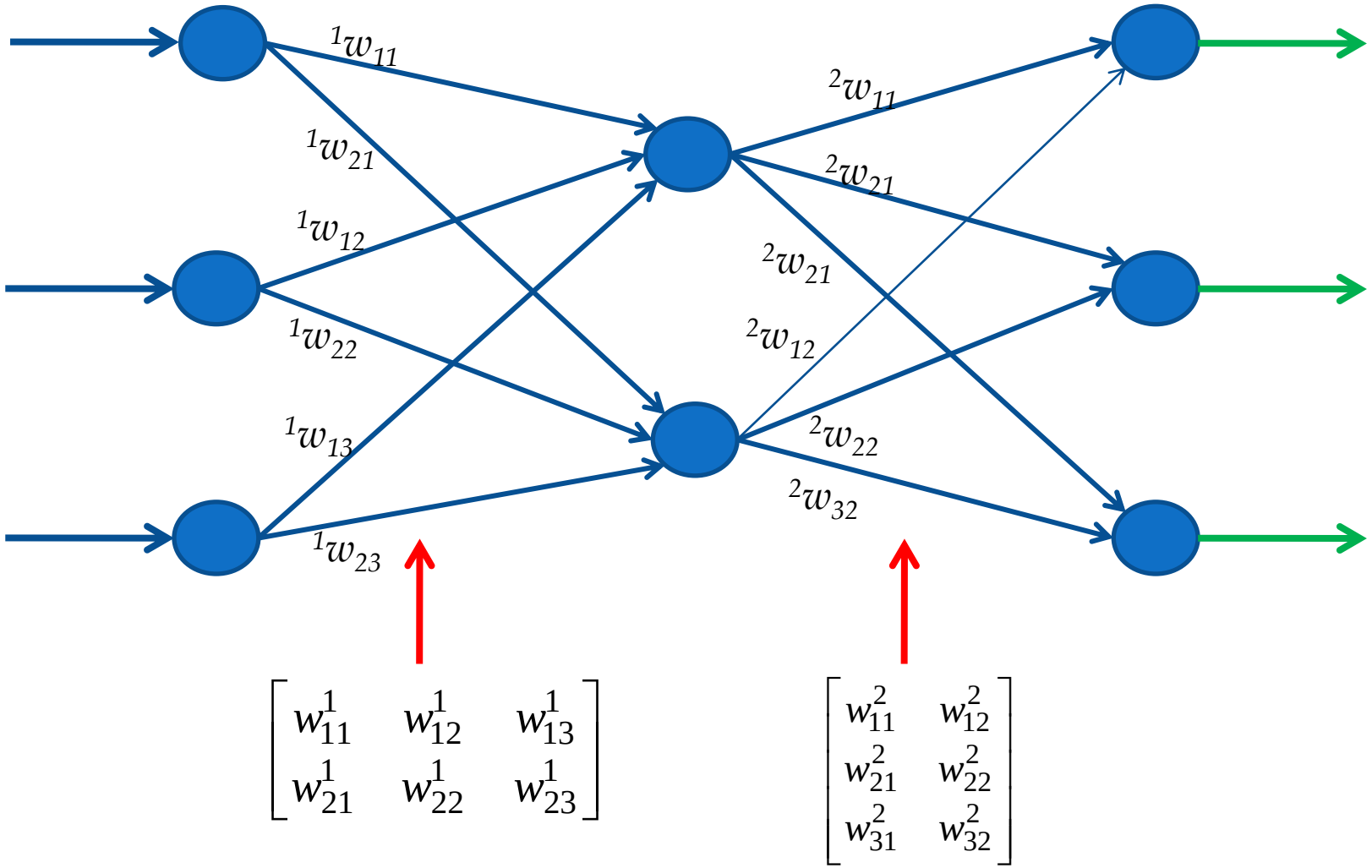


- Inputs are shown at the left and the units at the right side are receiving outputs from the processing unit *j* at the center.
- The each processing unit has associated with three values :
    1. Weighted sum of its inputs
    2. An output value
    3. An error value *that is used during the weight adjustment*.

Matrix Representation of Interconnection Weight Values

- The weights associated with each interconnection are adjusted during learning and are fixed after the learning is completed.
- The weight to the unit j from the unit i is denoted as w_{ji}
- There is matrix of weights values that corresponds to each layer of interconnections
- These matrices are with superscripts to distinguish weights in different layers

Figure 5-3: Weight Matrix of Three Layered Back-Propagation System



Training

- A back-propagation neural network is trained by supervising learning
- During learning, the network is presented with pairs of pattern (an input pattern paired with a desired target pattern)
- Upon presentation of each input pattern, weights are adjusted to decrease the difference between the network's output and the target output.
- The network is trained with training set(a set of input/target pattern pairs) which is presented to the network many times.
- One epoch is entire presentation of a complete training set.
- After training is stopped , the performance of the network is tested.
- A back-propagation learning involves a forward-propagation step followed by backward-propagation step.
- Both the forward-propagation and the backward-propagation steps are done for each pattern presentation during training/learning.

Forward-Propagation

- The forward propagation step is initiated when a input pattern is presented to the network.
- The input layer of units is a special case where each input unit corresponds to an entry in the input pattern vector and takes on the value of this entry (neither perform weighted sum nor use the activation function)
- After the activation levels for the first layer of units is set, the remaining layers perform a forward propagation step, which determine the activation levels of the other layers of units.
- In the figure 5-4, incoming connections to the unit j are at the left and originate at the units in the layer below. The activation values of these units arrive at unit j and summed by

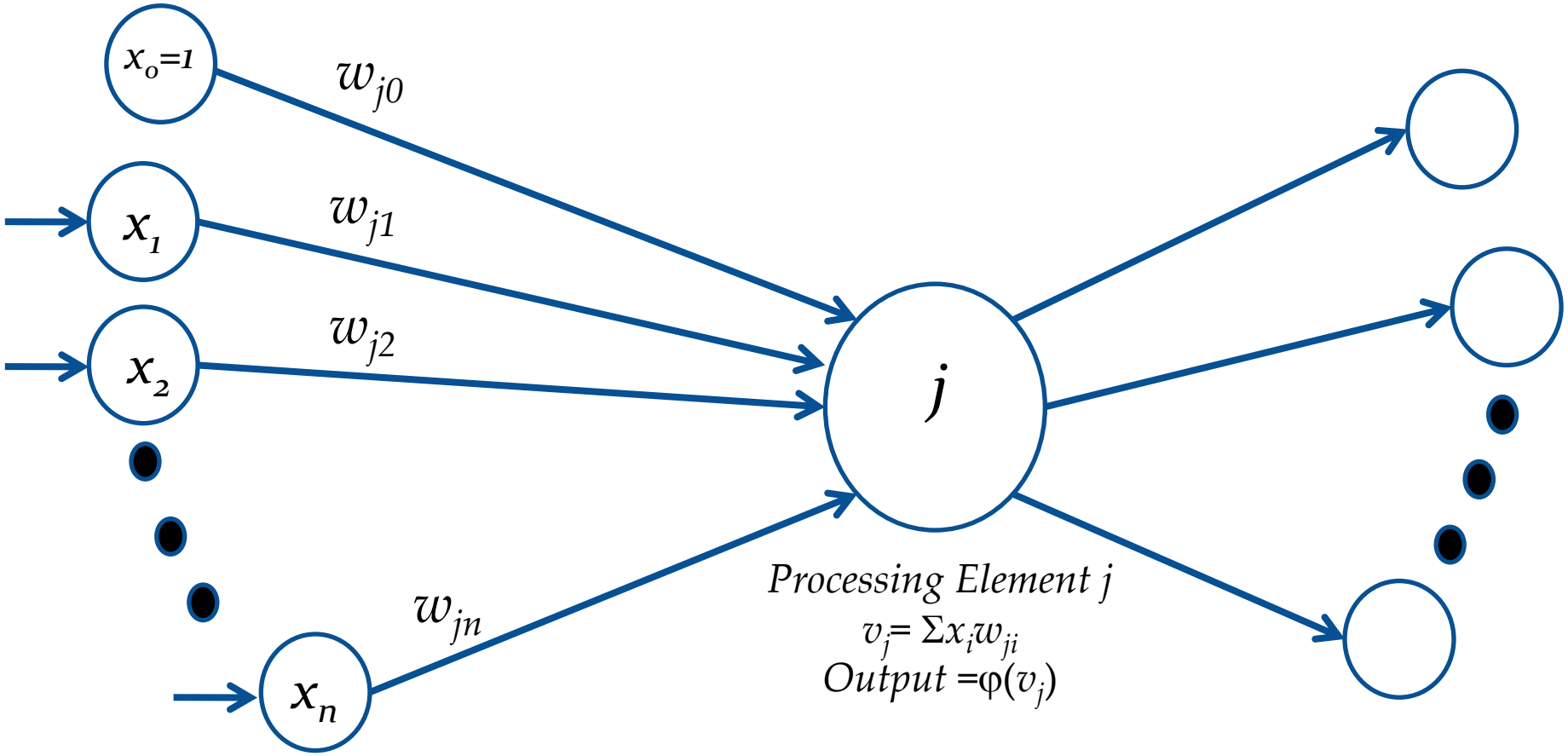
$$v_j = \sum_i x_i w_{ji}$$

Where

x_i the activation level of unit i

w_{ji} is the weight from unit i to unit j

Figure 5-4: The forward-propagation step



Sigmoid Activation Function

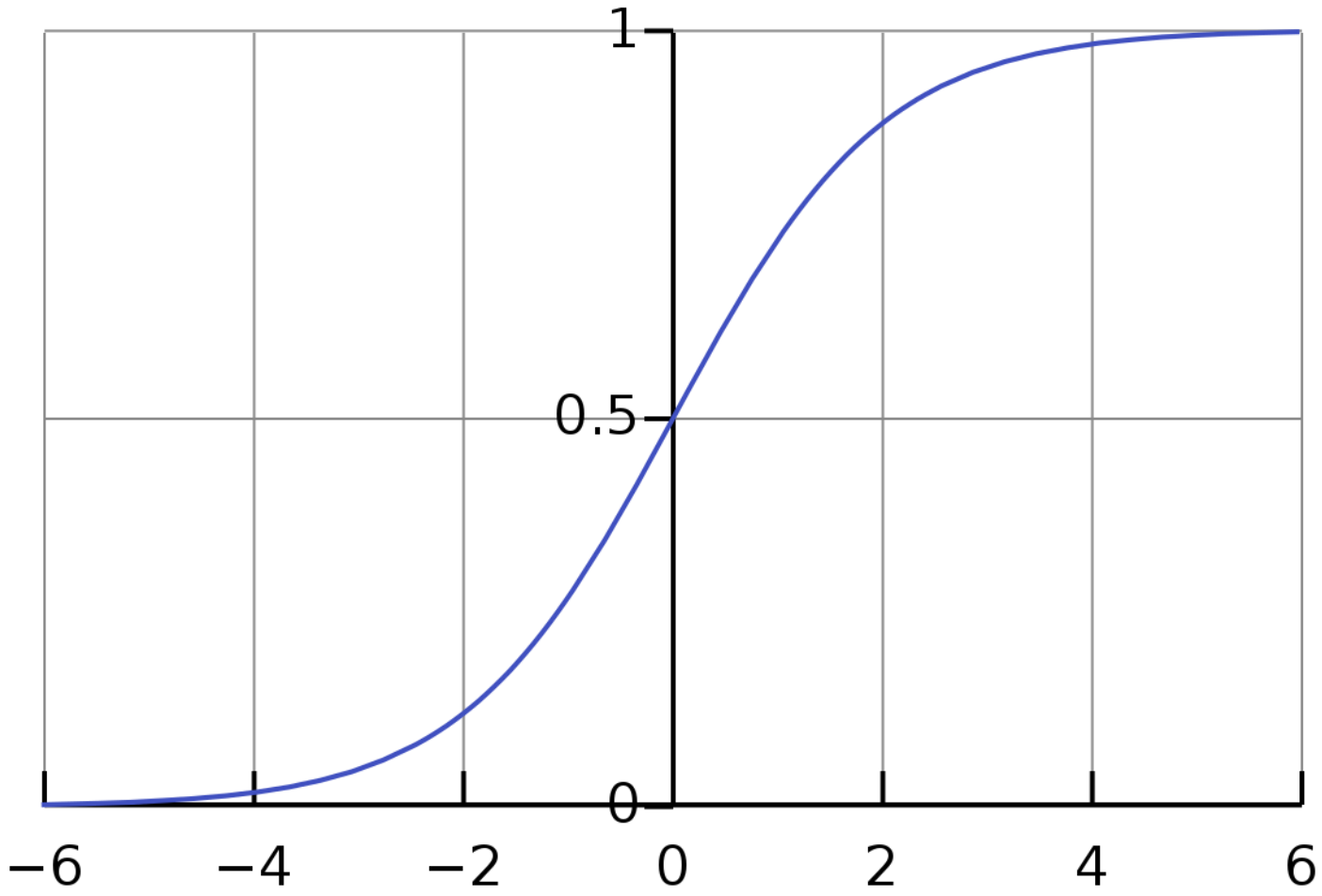
- After computing the weighted sum v_j, an activation function is used to compute $\varphi(v_j)$. Here the activation function φ is a sigmoid curve.
- The sigmoid curve is relatively flat at both ends and has a rapid rise in the middle.
    - When v_j is less than -3 it is close to 0 that is $\varphi(v_j)$ approached 1 asymptotically as v_j gets larger
    - When v_j is greater than +3 it is close to 1 that is $\varphi(v_j)$ approached 0 asymptotically as v_j gets greater negative value.
    - There is a transition from 0 to 1 takes place when $-3 < v_j < +3$

- The sigmoid function performs a soft threshold.

$$\varphi(v_j) = \frac{1}{1+e^{-v_j}} = \frac{1}{1+e^{-\sum x_i w_{ji}}}$$

- After the sigmoid function is computed on v_j, the resulting value becomes the activation level of the unit j
- This activation value of unit j is sent along the output interconnections (the same output value is sent along the all of the output interconnections).

Figure 5-5: Sigmoid Curve

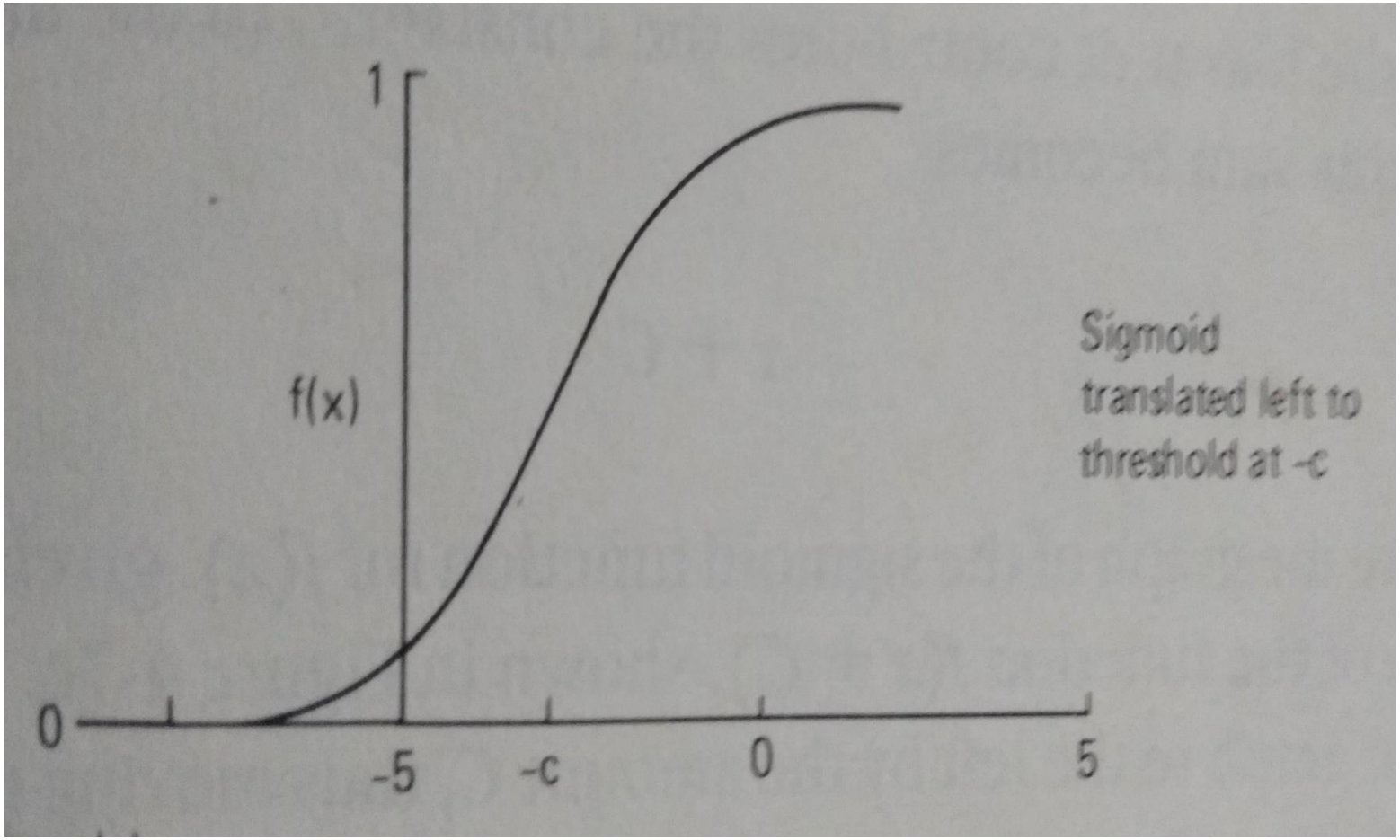


Bias Units and its Role:

- Some back propagation neural networks employ a bias unit as a part of every layer but output layer (some times improve the convergence properties of the network).
- This bias unit always has a constant activation value of x_0=+1.
- Each bias unit connected to all the units in the next layer and its weight to them are adjusted during the learning process and provides a constant term $C_j = w_{j0}$ to the weighted sum of units in the next layer.
- This constant term results in threshold effect on each of its targets by translating the sigmoid curve to the left (if constant term C_j is positive) or right (if constant term C_j is negative)

$$C_j = w_{j0} \quad , \quad u_j = \sum_{i=1}^{n} x_i w_{ji} \quad and \quad v_j = u_j + C_j$$

Figure 5-6 : The sigma curve moved C_j units to the left with threshold at $-C_j$



Backward-Propagation

- The backward propagation step, error correction step take place after a pattern is presented at input layer and the forward step is complete. During this step, the δ values are calculated for all processing units (except input units) and weight changes are calculated for all interconnections. This calculation first begins at output layer and progress backward through the network to the input layer.
- Each processing unit in the output layer produce a single real number for it output, which is compared to target output specified in the training set.
- Based on this difference, an error value is calculated for each unit in the output layer. then the weight are adjusted for all inter connection that go into the output layer.

- After an error value is calculated for all of the units in the hidden layer that is just below the output layer. Then the weight are adjusted for all inter connection that go into the hidden layer. This process is continued until the last layer weights has been adjusted.

The calculation of an error value for the Output-Layer Units:

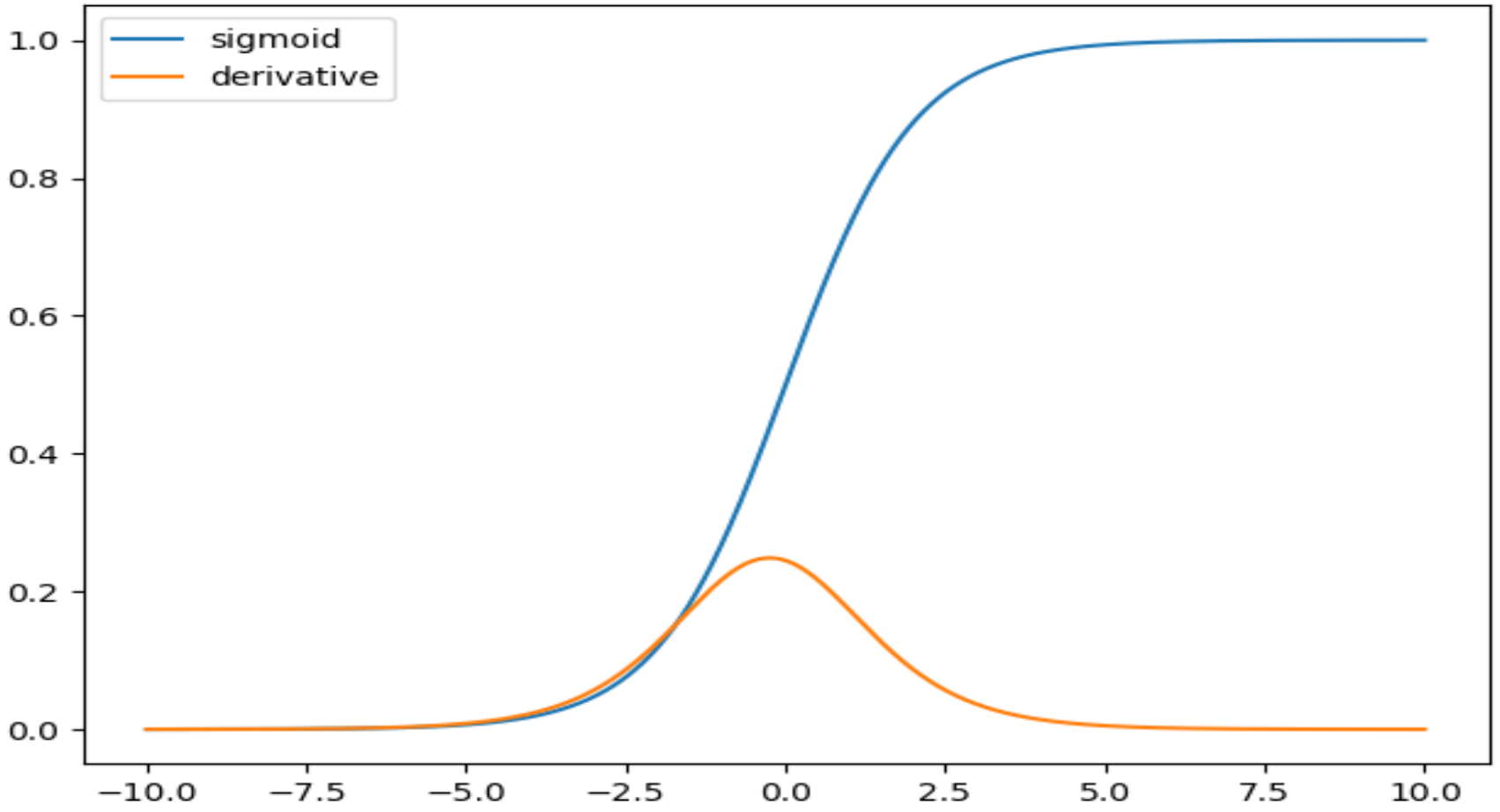
- The calculation of an error value, denoted by δ , is computed for the output layer as follows.

$$\delta_j = (t_j - x_j) f'(v_j)$$

Where

- t_j the target value for the output unit j
- x_j the out value of the unit j
- $f'(v_j)$ the derivative of the sigmoid function
- v_j is the weighted sum of inputs to the unit j.

Figure 5-7 : Sigmoid Function and its Derivative



- The quantity $(t_j\text{-}x_j)$reflects the amount of error. The f' part of the term scales the error to force a stronger correction when the sum v_j is near the rapid rise in the sigmoid curve.

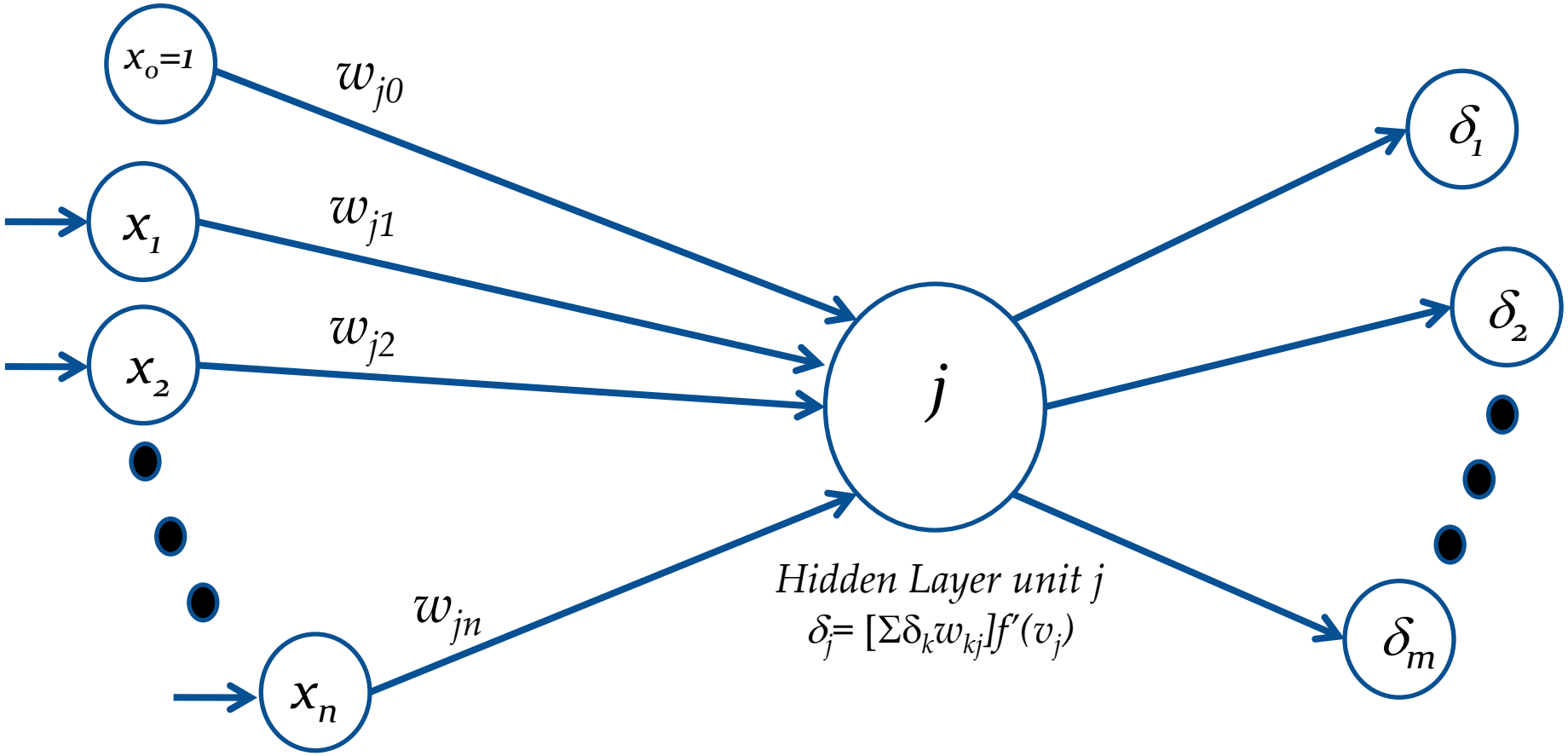
The calculation of an error value for the Hidden-Layer Units

- The calculation of an error value, denoted by δ, is computed for the output layer as follows.

$$\delta_j = \left[\sum_k \delta_k w_{kj}\right] f'(v_j)$$

Here a weighted sum is taken of the value of all units that receive output from the hidden unit j. the f' again serves to scale this output by emphasizing the region of rapid rise of the sigmoid function

Figure 5-8: The Processing unit in a hidden -Layer



Interconnection Weight Adjustment Using δ Value:

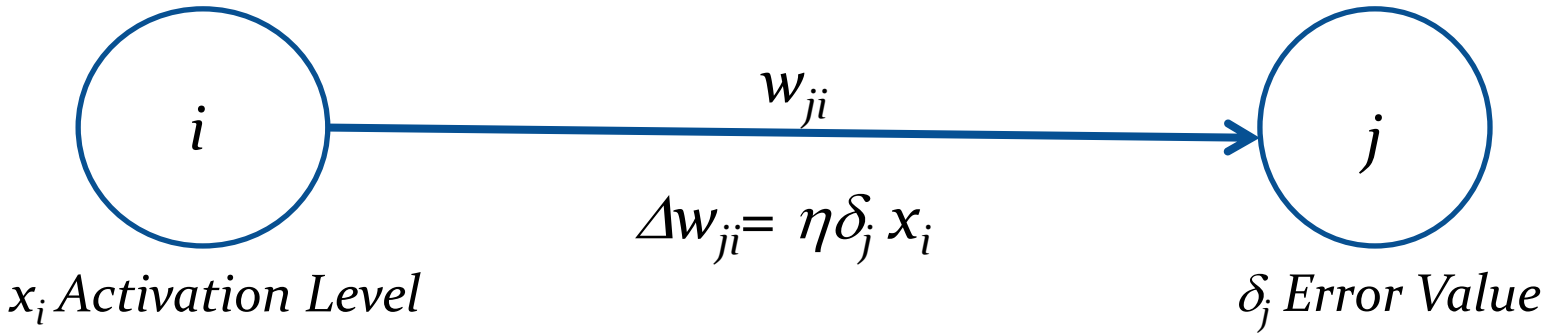
- The adjustment of interconnection weights is done using δ values of the processing unit. That is each interconnection weight is adjusted by taking into account of the δ value that receives input from the interconnection.
- The formula for weight adjustment is

$$\Delta w_{ji} = \eta \delta_j x_i$$

This weight adjustment equation is known as the generalized δ rule

- The size of the weight adjustment is proportional to δ, the error value of the target unit. Thus the larger error value for the unit *j* results in larger adjustment to its incoming weights.
- The weight adjustment is also proportional to x_i, the output value from the originating unit. If the output value is small, then the weight adjustment is small. If the output value is larger, then the weight adjustment is larger. Thus a higher activation value from incoming unit *i* results in a lrger adjustment to its outgoing weights.

Figure 5-9 : Updating Weight



- Thee variable η in the weight adjustment equation is the learning rat and reflects the rate of learning of the network. Its value is commonly ranges between 0.25 and 0.75.
- The larger value of η can lead to instability in the network and unsatisfactory learning
- The much smaller value of η can lead to excessively slow learning.
- Sometime the learning rate η is varied in an attempt to produce more efficient learning of the network
- Allowing the value of η to begin at high value and decrease during the learning session can sometimes produce better learning performance.

Artificial Neural Networks

Topic-06

Unsupervised Learning

Unsupervised Learning

- There exist problems where the training data consist of only input patterns and the desired output pairs are not available.
- Where the only information is provided by a set of input patterns.
- In these cases the relevant information has to be found within the redundant training samples .

Example Problems

- **Clustering:** Here the input data are to be grouped in to clusters and the data processing system has to find these inherent clusters pertaining to the input data. The output of the system should give the cluster label of the input pattern (discrete output)
- **Vector Quantization:** This problem occurs when a continuous space has to be discretized. The input of the system is the n-dimensional vector x. The output is a discrete representation of the input space, The system has to find optimal discretization of the input space.
- **Dimensionality Reduction:** The input data are grouped in a subspace which has lower dimensionality than the dimensionality of the data. The system has to learn an optimal mapping such that most of the variance in the input data is preserved in the output data.
- **Feature Extraction:** The system has to extract features from the input signal. This often means a dimensionality reduction as described above.

Neuro-Computational Solution

There is a special classes of ANN known as are self organizing networks that are suitable for solving these kind of problems. In these networks the training is done without the presence of an external teacher using unsupervised weight adapting algorithms. These algorithms are usually based on some form of global competition between the neurons.

- One of the most basic schemes is competitive learning as proposed by Rumelhart and Zipser
- A very similar network but with different emergent properties is the topology conserving map devised by Kohonen
- Other self-organising networks are ART proposed by Carpenter and Grossberg, and Fukushima's cognitron by Fukushima.

Competitive Learning Network

In competitive learning network all the output neurons are connected to every input neuron with associated interconnection weights. When an input pattern is presented , the output neurons of the network compete among themselves to be activated with the result only one output neuron is ***ON*** at any one time. The output neuron that wins the competition is called winner-takes all-neuron or simply a winning neuron.

There are two methods for the determination of the winner.

Winner selection Using *Dot Product*

Here both the input vectors and weight vectors are normalized to unit length.

Step-1: Each output unit calculates its activation according to the dot product of input and weight vector. (this nothing but the weighted sum)

Step-2: The output neuron *'k'* with maximum activation is selected as a winning neuron.

Step-3: Activation of the output neurons are reset such that the winning neuron will have the value of ***+1*** and other neurons will have the value of ***0*** for the given input.

Winner Selection Using *Euclidean Distance*

The dot product method would fail if un-normalised vectors were to be used.

Naturally one would like to accommodate the algorithm for un-normalized input data.

For this end winning neuron '*k*' is selected with its weight vector closest to the input pattern using the Euclidean distance measure.

Self Organizing Map

In a self-organizing map (SOM), the neurons are placed at the nodes of a lattice that is usually one or two dimensional.

The neurons are selectively tuned to various input patterns.

SOM is characterized by the formation of a topological map of input patterns in which the spatial location of a neurons in the lattice are indicative of intrinsic statistical feature contained in the input pattern. Hence the name self-organizing map.

Feature Mapping Models

Motivation

The development of self organizing map as neural model is motivated by distinct feature of the human brain: the brain is organized in many places in such a way that different sensory inputs are mapped on to corresponding areas in a cerebral cortex in a topologically ordered fashion. Thus the computational map constitutes a basic building block in the information processing infrastructure of the nervous system.

Properties of Brain Computational Map

- At each stage of representation each incoming piece of information is kept in its proper context.
- Neurons dealing with closely related pieces of information are close together sot hat they can interact via short synaptic connections.
- The principle of artificial topological map states that " the spatial location of an output neuron is topological map corresponds to a particular domain or feature of data drawn from the input space".

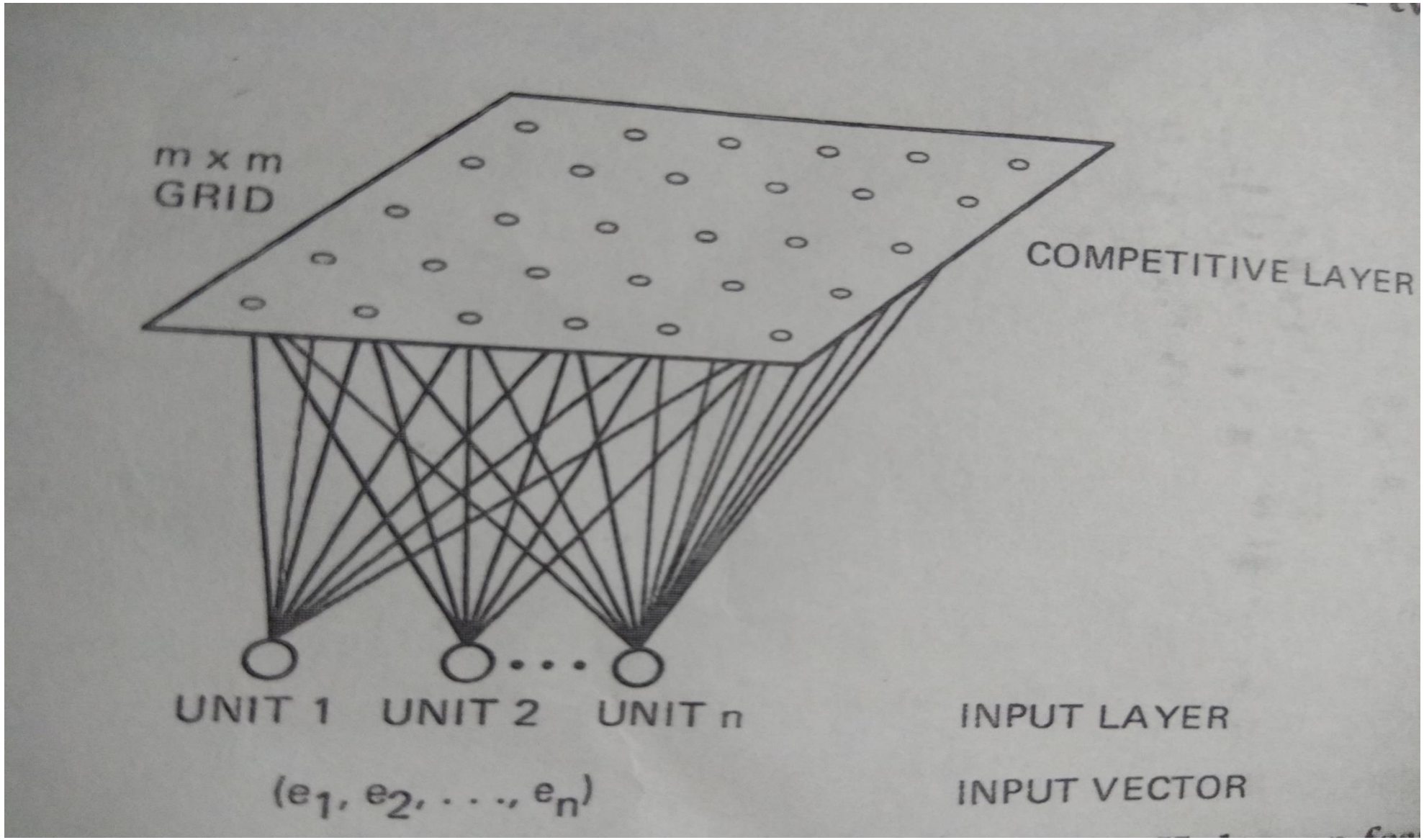
Kohonen Feature map

Kohanen's self-organizing feature map is a two layered network that can organize a topological map from random starting point. The resulting map shows the natural relationship among the pattern that are given to the network. This network combines the input layer with a competitive layer of processing unit and trained by unsupervised learning

The Kohonen feature map finds organization of relationship among patterns.

Basic Structure

- The Kohonen feature map is a two-layered network.
- The first layer of the network is the input layer.
- The second layer is the competitive layer and is organized as two-dimensional grid
- All connections go from the first layer to the second layer and these two layers are fully interconnected [each input neuron is connected to all the neurons in the competitive layer]
- Each interconnection has an associated weight value
- The typical initial weights are set by adding a small random number to the average entries in the input patterns.
- These weight values get updated during the training of the network.


m x m
GRID
COMPETITIVE LAYER
UNIT 1
UNIT 2
UNIT n
INPUT LAYER
$(e_1, e_2, \ldots, e_n)$
INPUT VECTOR

Training

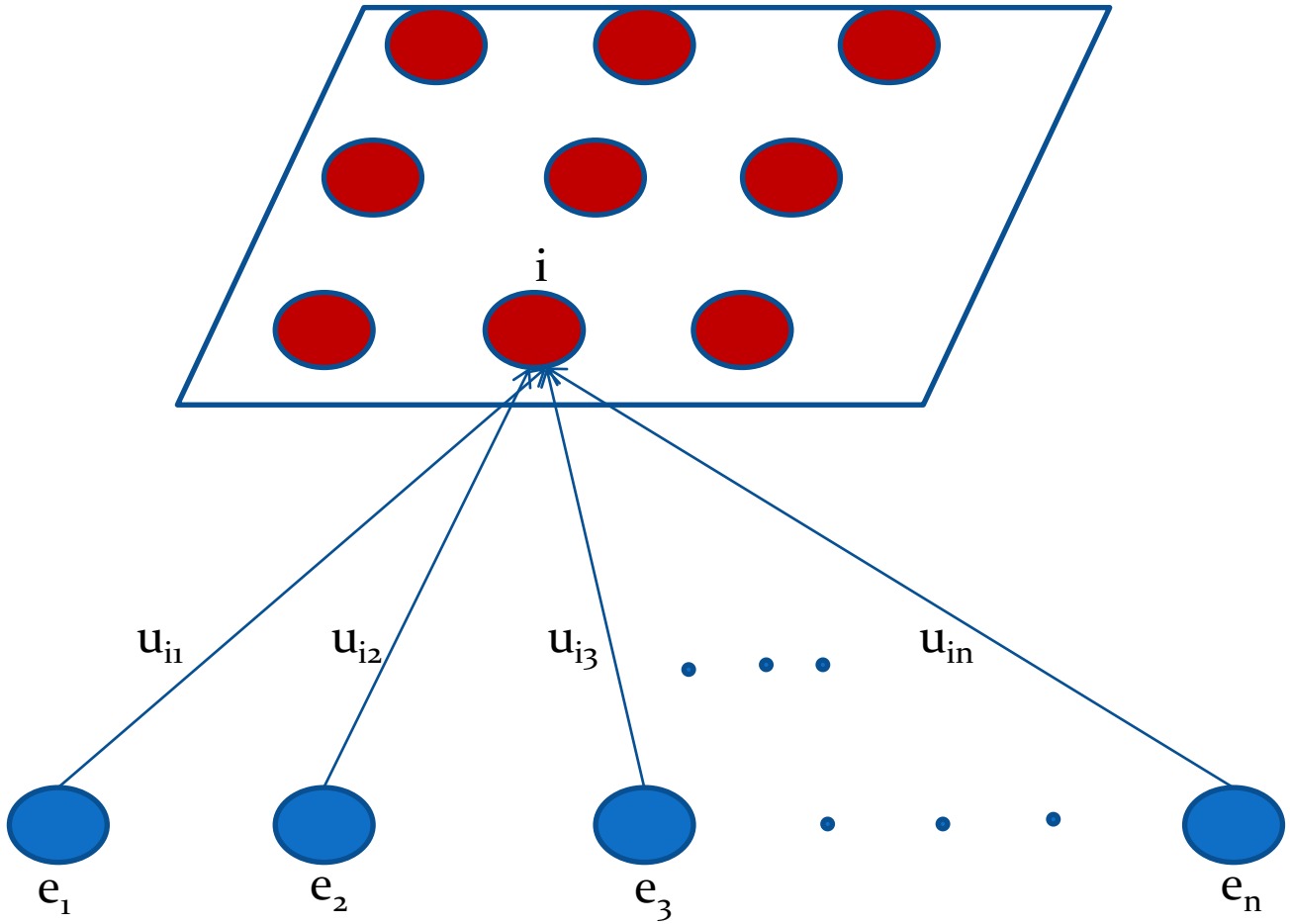
- When an input pattern is presented, each unit in the first layer takes on the value of the corresponding entry in the input pattern. This input pattern is denoted by

$$\boldsymbol{E} = [e_1, e_2, \ldots, e_n]$$

- The weights associated with connections from the input units to single unit (say i^{th} unit, here we identify each unit in the competitive layer by a single index, even though there is a two-dimensional grid of units in this layer) in the competitive layer are given by

$$\boldsymbol{U}_i = [\, u_{i1}, u_{i2}, \ldots, u_{in}]$$

Connections From the Input Vector to Single Unit in the Competitive Layer



Training Steps

The first step is to compute matching value for each unit in the competitive layer. This value measure the extent to which the weights each unit match the corresponding values of the input pattern. The matching value for the unit i is calculated by the following equation

$$\|E - U_i\| = \sqrt{\sum_j (e_j - u_{ij})^2}$$

which is the distance between the vectors E and U_i

The unit with lowest matching value(the best match) wins the competition. Hence the unit with the best match is denoted as unit k and k is chosen such that

$$\|E - U_k\| = \min_i \left(\|E - U_i\| \right)$$

where the minimum is taken over all units i in the competitive layer.

If two or more units have same matching value then by convention, the unit with lower index value i is chosen

The next step after identifying the winning unit is to identify the neighborhood around it. The neighborhood consist of the units that are close to winner in the competitive layer grid. Here the neighborhood consists of a set of units that are within a square that is centered around the winning unit ***k*** and denoted by $N_{k..}$ The of the neighborhood can vary depending up on the distance between ***k*** to the edge of the neighborhood and denoted by ***d.*** The weights are updated for all the units that are in the neighborhood of the winning unit $N_{k.}$ by using the equations

$$\Delta u_{ij} = \begin{cases} \alpha\left(e_j - u_{ij}\right) & if \quad unit \quad i \quad is \quad in \quad the \quad neighborhood \quad N_k \\ 0 & otherwise \end{cases}$$

$$u_{ij}^{new} = u_{ij}^{old} + \Delta u_{ij}$$

This adjustment results in the winning unit and its neighbors having their weights modified , becoming more like input pattern. Now the winner become more likely to win the competition when the same or a similar input pattern will be presented subsequently.

Here two parameter need to be specified:

- The value of α, the learning rate parameter in the weight adjustment equation
- The size of the neighborhood N_k

Setting the Value for α and N_c

- Initially the learning rate α takes relative larger value and being decreased over the span of many iterations.
- This initial value of α is set by choice and is denoted by α_0 (the typical choices are in the range0.2 to 0.5).
- The value of α for t^{th} is denoted by α_t determined by the equation

$$\alpha_t = \alpha_0\left(1 - \frac{t}{T}\right)$$

where t → is the current training iteration

T → is the total number of training iteration to be done,

The value begins with α_0 and decreases until it reaches zero

- The initial neighborhood width is relatively large, and is decreases over the progress of the training iterations. The center of the neighborhood is the winning unit k and at the position (x_k, y_k). Let d be the distance from k to the edge of the neighborhood. The neighborhood is then all (x, y) such that

  $k - d < x < k + d$ and $k - d < y < k + d$

- Sometimes the calculated neighborhood goes outside the grid of units in the competitive layer; in this case the actual neighborhood will be cut off at the edge.
- Initially d is assigned with a chosen value denoted by d_0 (the typical value for d_o can be a half or a third of the width of the competitive layer of processing units. The value of d will be decreased according to the equation

$$d = \left\lceil d_0 \left(1 - \frac{t}{T} \right) \right\rceil$$

where t $\rightarrow$ is the current training iteration and

T $\rightarrow$ is the total number of training iteration

This process assures a gradual linear decrease in $\boldsymbol{d}$ starting with $\boldsymbol{d_o}$ and going down to 1

Summary

1. Locate the unit in the competitive layer whose weights best match the input pattern.
2. Increase matching at this unit and its neighbors by adjusting their weights
3. Gradually decrease the size of the neighborhood and the amount of change to the weights as learning iteration progress

Artificial Neural Networks

Topic-07: Radial Basis Function Neural Networks

Radial Basis Function Neural Networks

- The Radial-Basis function was first introduced in the solution of the real multivariate interpolation problems
- The RBFNN first performs non-liner transformation from given input space into higher dimension hidden space followed by linear transformation from hidden space to output space.
- *A pattern classification problem cast in a higher dimensional space is more likely to be linearly separable than in a lower dimensional space*- this is the reason for frequently making the dimension of the hidden space of RBF network high.
- Another important point is that *the dimension of the hidden space is directly related to the capacity of the network to approximate a smooth input-output mapping*. So the higher the dimension of the hidden space, the more accurate the approximation will be.

Basic Architecture of RBFNN

The construction of a RBFNN in its most basic form involves three layers with entirely different roles

- **Input Layer**: It contains *n* input (sensory/source) neurons that connects to the network to its external environment
- **Hidden Layer**: This is **only one** hidden layer. Hidden units provide a set of radial-basis function performs a non-linear transformation from the input space to the hidden space. In most applications the hidden space is of high dimensionality than input space.
- **Output Layer**: It contains *m* output neurons and each of which combines in a linear way the activations of the hidden layer. Supplies the response of the network for the activation pattern applied to the input layer

- The connection between input layer and hidden layer have no associated weights.
- The selection of an appropriate RBF depends on the type of problem to be solved by RBFNN.

$$RBF \quad \varphi_k = \varphi\left(\left\| X - c_k \right\|\right) = \varphi_k(r)$$

$$\varphi(r) = r \quad a \quad linear \quad radial \quad function$$

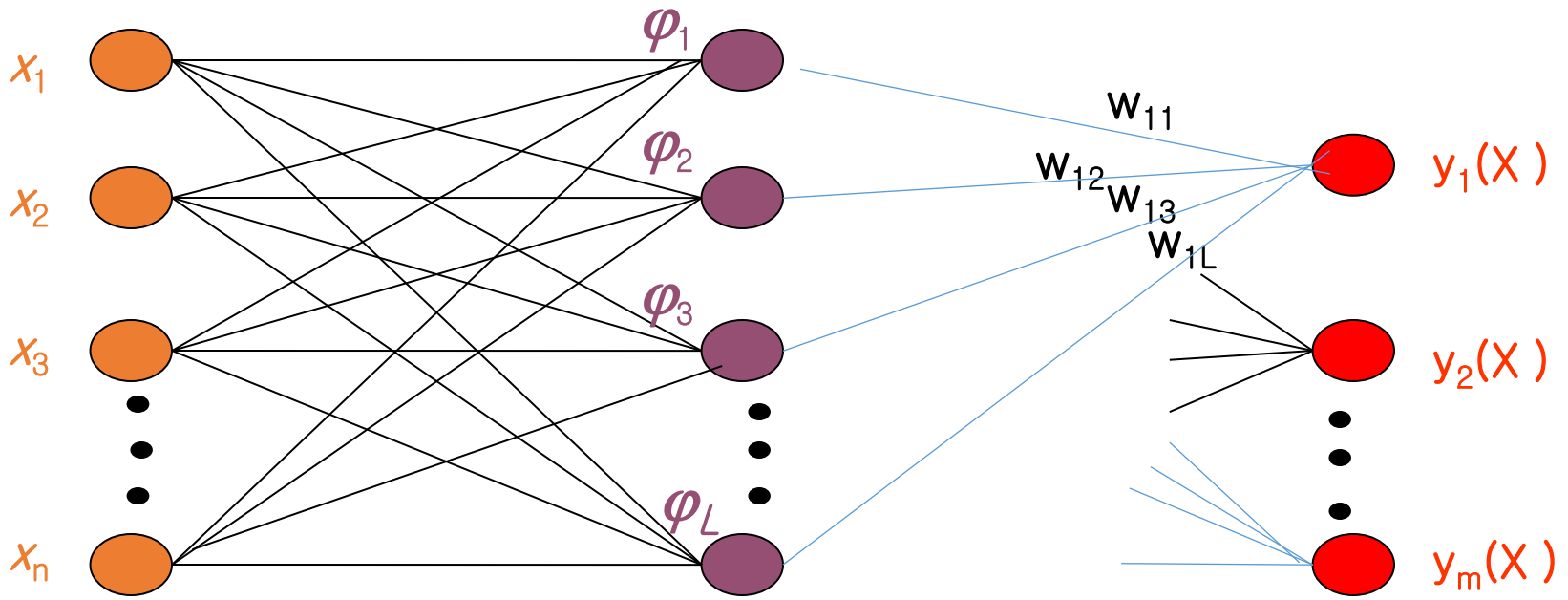
$$\varphi(r) = r^2 a \quad quadratic \quad function$$

$$\varphi(r) = \exp\left(-r^2 / b^2\right) \quad a \quad gaussian \quad function$$

$$\varphi(r) = r^2 \log(r) \quad a \quad thin-plate \quad spline \quad function$$

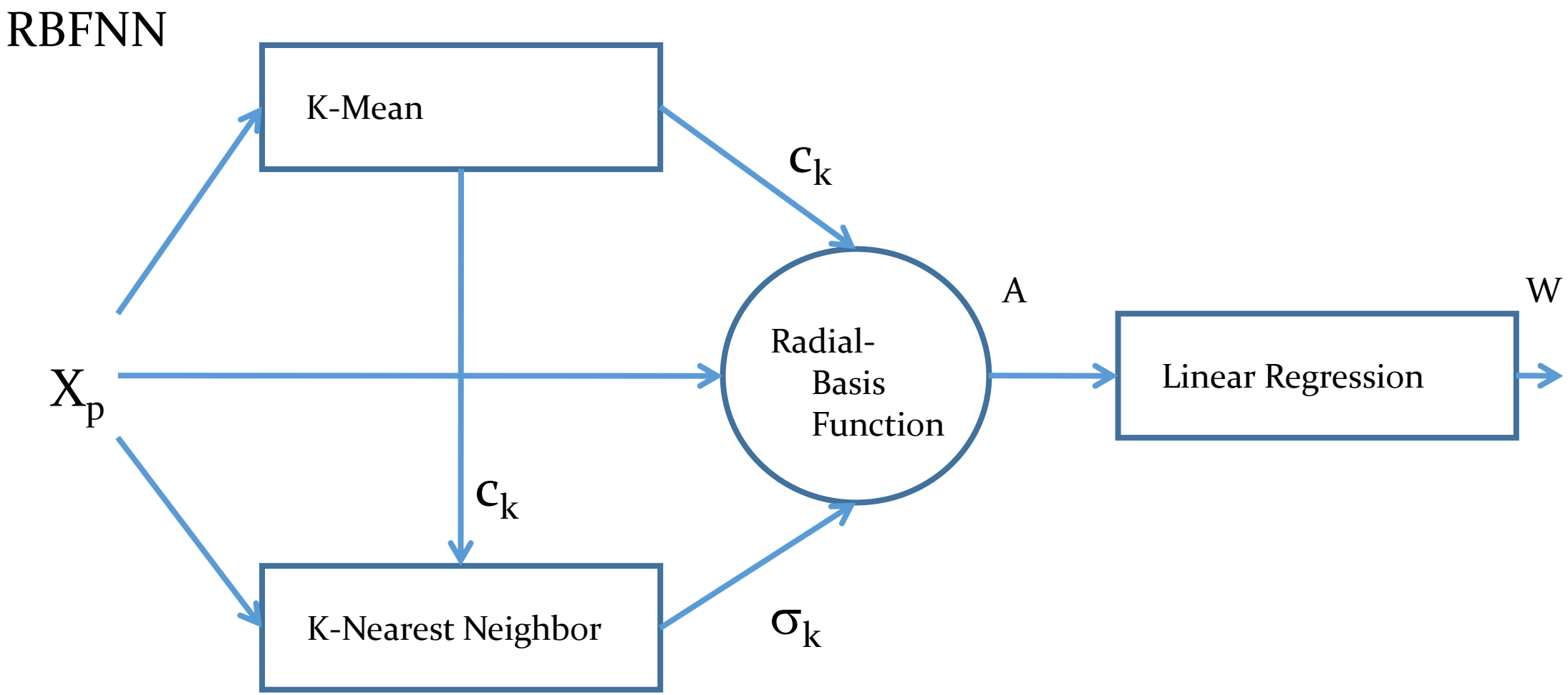
$$\varphi(r) = \sqrt{\left(r^2 - b^2\right)} \quad a \quad multiquadratic \quad function$$

RBFNN Architecture



Input layer Hidden layer Output layer

Learning Process of RBFNN



- RBF is a kind of **supervised** neural networks
- Design of NN as *curve-fitting* problem
- **Learning**: find surface in multidimensional space best fit to training data by determining $\boldsymbol{w_i}$, $\boldsymbol{\sigma_i}$ and $\mathbf{c}_i$ separately
  - RBF networks solve this problem by dividing the learning into two independent processes.
    - Center and spread learning (or determination)
    - Output layer Weights Learning
- **Generalization**: Use of this multidimensional surface to interpolate the test data

The response characteristics of the k^{th} hidden unit is given by

$$\varphi_k(X) = \varphi\left(\frac{\|X - c_k\|}{\sigma_k^2}\right)$$

- Where $\varphi_k(.)$ is strictly positive radial symmetric function with a unique maximum at k^{th} center c_k and which drop off rapidly to zero away from the center.
- The parameter is the width of the receptive field in the input space for the unit k.
- In other words functions σ_k are defined in areas of the corresponding points c_k which causes their sensitive receptive field parameter σ_k that defines the geometric size of the k^{th} receptive field in the input space for unit k.

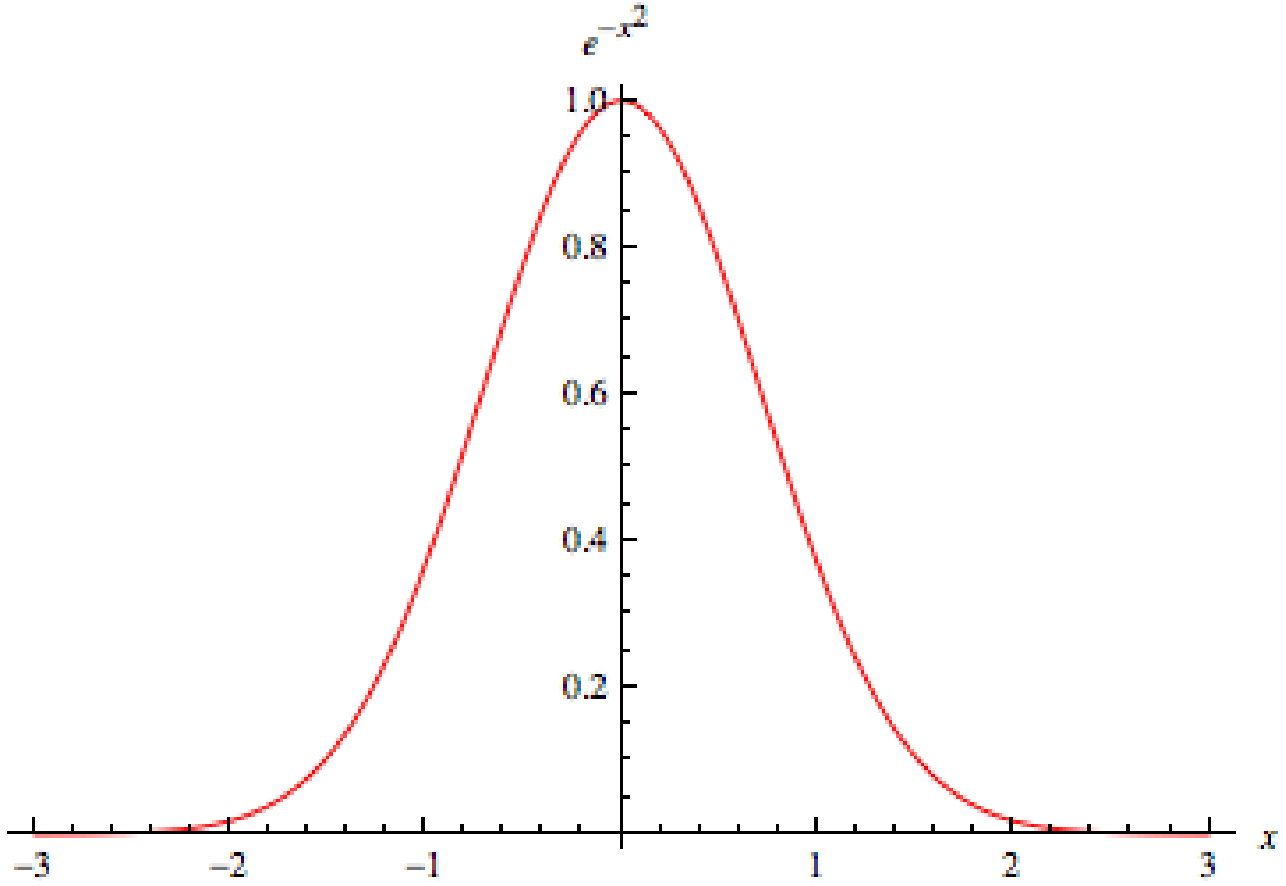
- BFNN performs the mapping of n-dimensional input vector ***X*** to m-dimensional output vector Y ($R^n \rightarrow R^m$).
- The node function φ_k (k= 1,2,3, . . . L) of the hidden layer represent the RBF transformation of the input vector $X = (x_1, x_2, ..., x_i, . . ., x_n)$ the output of the RBFNN is the m-dimensional activity vector $Y=(y_1, y_2, y_3, . . . , y_m)$, where j^{th} component is given by

$$y_j(X) = \sum_{k=1}^{L} W_{jk}\varphi_k(X)$$

$$\varphi_k = \exp\left(-\sum_{i=1}^{n}\frac{(x_i - c_{ki})}{\sigma_{ki}^2}\right) \textit{if we choose Gaussian basis function}$$

$$y_j(X) = \sum_{k=1}^{L} W_{jk}\exp\left(-\sum_{i=1}^{n}\frac{(x_i - c_{ki})}{\sigma_{ki}^2}\right)$$

Gaussian Function



Finding the Weight

- c_i can be find by using k-means algorithm
- The width σ can be find by using k- nearest neighbor rule
- The weights can be determined as follows

$$\begin{bmatrix} y_1(X) \\ y_2(X) \\ \bullet \\ \bullet \\ \bullet \\ y_m(X) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \bullet & \bullet & \bullet & a_{1L} \\ a_{21} & a_{22} & \bullet & \bullet & \bullet & a_{2L} \\ \bullet & \bullet & \bullet & & & \bullet \\ \bullet & \bullet & & \bullet & & \bullet \\ \bullet & \bullet & & & \bullet & \bullet \\ a_{m1} & a_{m2} & \bullet & \bullet & \bullet & a_{mL} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \bullet \\ \bullet \\ \bullet \\ w_L \end{bmatrix}$$

$$Y = AW$$

$$W = A^{-1}Y$$

Finding c_ks by Using k-means Algorithm

Step1: K initial clusters are chosen randomly from the samples to form K groups.

Step2: Each new sample is added to the group whose mean is the closest to this sample.

Step3: Adjust the mean of the group to take account of the new points.

Step4: Repeat step2 until the distance between the old means and the new means of all clusters is smaller than a predefined tolerance.

Outcome: There are K clusters with means representing the centroid of each clusters.

Advantages: (1) A fast and simple algorithm.

(2) Reduce the effects of noisy samples.

Finding the RBF function width σ by Using *K* Nearest Neighbor Rule

- **The objective is to cover the training points so that a smooth fit of the training samples can be achieved**

$$\sigma_i = \sqrt{\frac{1}{K}\sum_{k=1}^{K}\|c_k - c_i\|^2}$$

k^{th} nearest neighbor of c_i

Conclusion

- The objective is to cover the training points so that a smooth fit of the training samples can be achieved
- The hidden layer RBFNN does not have corresponding weights and threshold.

Artificial Neural Networks
Topic-08: Hebbian Learning

What is Neural Networks Learning Process

- The significant property of neural networks is its ability to learn from its environment to improve its performance.
- The neural networks learn about its environment through an interactive process of adjustment applied to its synaptic weight and bias level.
- Therefore the neural networks store the knowledge from its environment in terms of synaptic weight and bias level and gains knowledge about its environment after each iteration of the learning process

What is Learning in the Context of Neural Networks

- Learning is a process by which free parameters of a neural networks are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which these parameter change take place.
    - The neural network is stimulated by its environment
    - The neural networks undergoes changes in free parameters as the result of this stimulation
    - The neural network responds in a new way to the environment because of the changes that have occurred in its internal structure,
- A prescribed set of well-defined rules for solution of a learning problem is called learning algorithm

Hebbian Learning

Canadian neuropsychologist Donald O Hebb presented a theory of behaviour based as much as possible on the physiology of the nervous system in his book " The organization of behviour: in 1949.

Hebb's Rule: Hebb's Postulate on Cell Assembly Theory

Hebb reduced the type of physiological evidence in to two categories

1. Existence and properties of continuous cerebral activity
2. The nature of synaptic transmission in the central nervous system.

Hebb combined these two principles to develop a theory of how learning occurs within an organism. He proposed that repeated stimulation of specific receptors leads slowly to the formation of cell assembly, which act as a closed system after stimulation has ceased.

The most important concept to emerge from Hebb's work was his formal statement of how learning could occur. Learning is based on the modification synaptic connections between neurons.

When an axon of cell A is near enough to excite a cell b and repeatedly or persistently takes part in firing it, some growth or metabolic change take place in one or both cells such that A's efficiency as one cell firing B is increased.

This theory often summarized as cells that fire together are wired together". Hebbian learning can be described as a time dependent local, highly interactive mechanism that increases synaptic efficiency as a function of pre and post synaptic activity.

The Hebbian Learning stated in neurobiological context may be expanded and rephrased as two-part rules:

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

II. If two neurons on either side of a synapse are activated asynchronously, then the synapse is selectively weakened or eliminated.

Such a synapse is called Hebbian synapse.

Four key Mechanisms that characterized a Hebbian Synapse

- Time dependant Mechanism: This mechanism refers to the fact that the modifications in a Hebbian synapse depends exact time occurrence of pre-synaptic and post-synaptic signals.
- Local Mechanism: By its very nature , a synapse is the transmission site where the information bearing signals are in spatiotemporal contiguity. This locally available information is used by Hebbian synapse to produce a local modification that is input specific.
- Interaction Mechanism: The occurrence of a change on both sides of the synapse. That is a Hebbian form of learning depends on a true interaction between pre-synaptic and post synaptic signals.
- Conjunctional or correlational Mechanism: One interpretation of Hebb's postulate of learning is that the condition for a change in synaptic efficiency is the conjunction of presynaptic and postsynaptic signals. Thus according to this interpretation , the co-occurrence of presynaptic and postsynaptic signals is sufficient to produce the synaptic modification

Artificial Neural Networks

Topic: *Unsupervised Learning*

Unsupervised Learning

- There exist problems where the training data consist of only input patterns and the desired output pairs are not available.
- Where the only information is provided by a set of input patterns.
- In these cases the relevant information has to be found within the redundant training samples .

Examples

- **Clustering:** Here the input data are to be grouped in to clusters and the data processing system has to find these inherent clusters pertaining to the input data. The output of the system should give the cluster label of the input pattern (discrete output)
- **Vector Quantization:** This problem occurs when a continuous space has to be discretized. The input of the system is the n-dimensional vector x. The output is a discrete representation of the input space, The system has to find optimal discretization of the input space.
- **Dimensionality Reduction:** The input data are grouped in a subspace which has lower dimensionality than the dimensionality of the data. The system has to learn an optimal mapping such that most of the variance in the input data is preserved in the output data.
- **Feature Extraction:** The system has to extract features from the input signal. This often means a dimensionality reduction as described above.

Neuro-Computational Solution

There is a special classes of ANN known as are self organizing networks that are suitable for solving these kind of problems. In these networks the training is done without the presence of an external teacher using unsupervised weight adapting algorithms. These algorithms are usually based on some form of global competition between the neurons.

- One of the most basic schemes is competitive learning as proposed by Rumelhart and Zipser
- A very similar network but with different emergent properties is the topology conserving map devised by Kohonen
- Other self-organising networks are ART proposed by Carpenter and Grossberg, and Fukushima's cognitron by Fukushima.

Competitive Learning Network

In competitive learning network all the output neurons are connected to every input neuron with associated interconnection weights. When an input pattern is presented , the output neurons of the network compete among themselves to be activated with the result only one output neuron is ***ON*** at any one time. The output neuron that wins the competition is called winner-takes all-neuron or simply a winning neuron.

There are two methods for the determination of the winner.

Winner selection Using *Dot Product*

Here both the input vectors and weight vectors are normalized to unit length.

Step-1: Each output unit calculates its activation according to the dot product of input and weight vector. (this nothing but the weighted sum)

Step-2: The output neuron '***k***' with maximum activation is selected as a winning neuron.

Step-3: Activation of the output neurons are reset such that the winning neuron will have the value of ***+1*** and other neurons will have the value of ***0*** for the given input.

Winner Selection Using *Euclidean Distance*

The dot product method would fail if un-normalised vectors were to be used.

Naturally one would like to accommodate the algorithm for un-normalized input data.

For this end winning neuron '***k***' is selected with its weight vector closest to the input pattern using the Euclidean distance measure.

Self Organizing Map

In a self-organizing map (SOM), the neurons are placed at the nodes of a lattice that is usually one or two dimensional.

The neurons are selectively tuned to various input patterns.

SOM is characterized by the formation of a topological map of input patterns in which the spatial location of a neurons in the lattice are indicative of intrinsic statistical feature contained in the input pattern. Hence the name self-organizing map.

Feature Mapping Models

Motivation

The development of self organizing map as neural model is motivated by distinct feature of the human brain: the brain is organized in many places in such a way that different sensory inputs are mapped on to corresponding areas in a cerebral cortex in a topologically ordered fashion. Thus the computational map constitutes a basic building block in the information processing infrastructure of the nervous system.

Properties of brain computational Map

- At each stage of representation each incoming piece of information is kept in its proper context.
- Neurons dealing with closely related pieces of information are close together sot hat they can interact via short synaptic connections.
- The principle of artificial topological map states that " the spatial location of an output neuron is topological map corresponds to a particular domain or feature of data drawn from the input space".

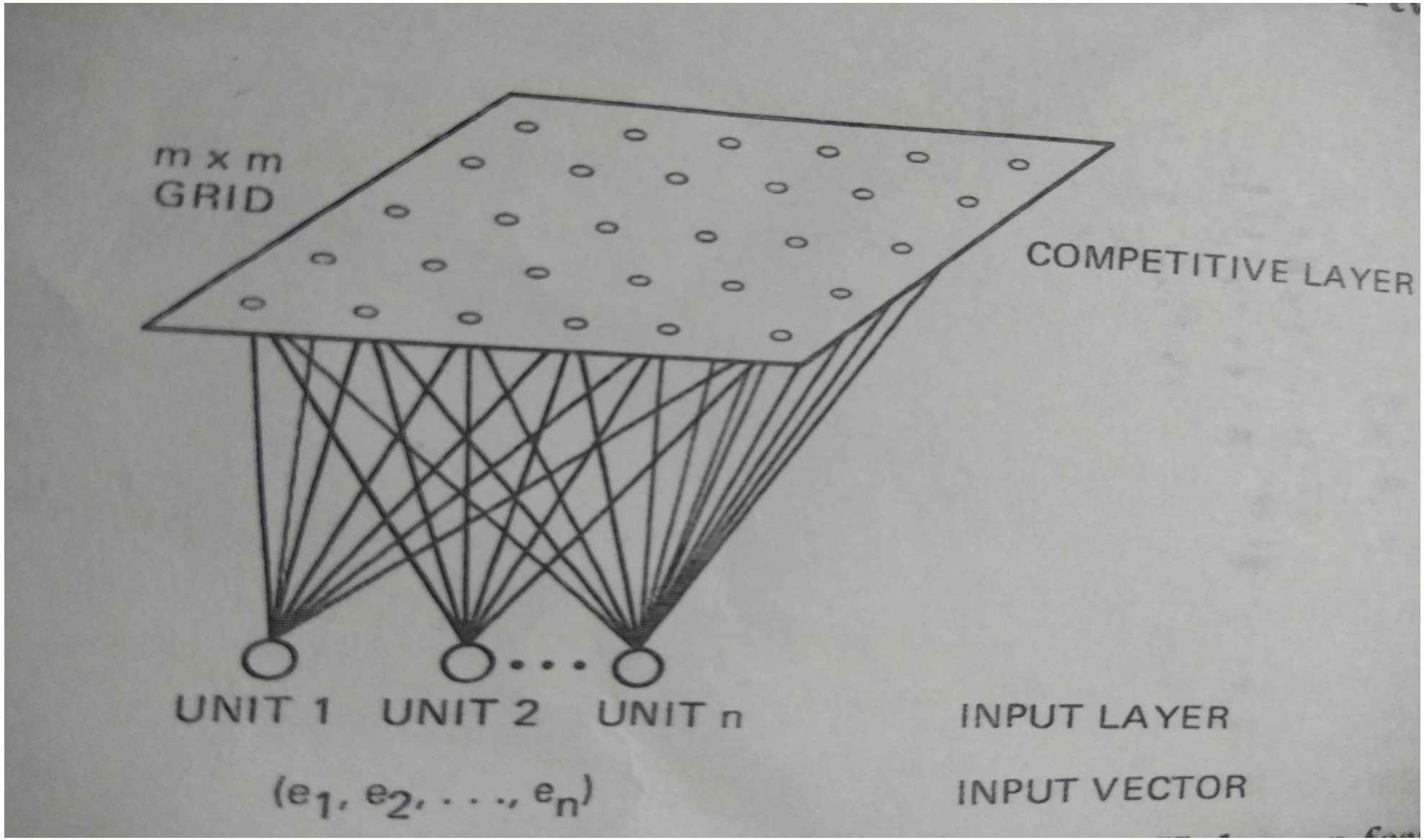
Kohonen Feature map

Kohanen's self-organizing feature map is a two layered network that can organize a topological map from random starting point. The resulting map shows the natural relationship among the pattern that are given to the network. This network combines the input layer with a competitive layer of processing unit and trained by unsupervised learning

The kohonen feature map finds organization of relationship among patterns.

Basic Structure

- The Kohonen feature map is a two-layered network.
- The first layer of the network is the input layer.
- The second layer is the competitive layer and is organized as two-dimensional grid
- All connections go from the first layer to the second layer and these two layers are fully interconnected [each input neuron is connected to all the neurons in the competitive layer]
- Each interconnection has an associated weight value
- The typical initial weights are set by adding a small random number to the average entries in the input patterns.
- These weight values get updated during the training of the network.


m x m
GRID
COMPETITIVE LAYER
UNIT 1
UNIT 2
UNIT n
INPUT LAYER
$(e_1, e_2, \ldots, e_n)$
INPUT VECTOR

Training

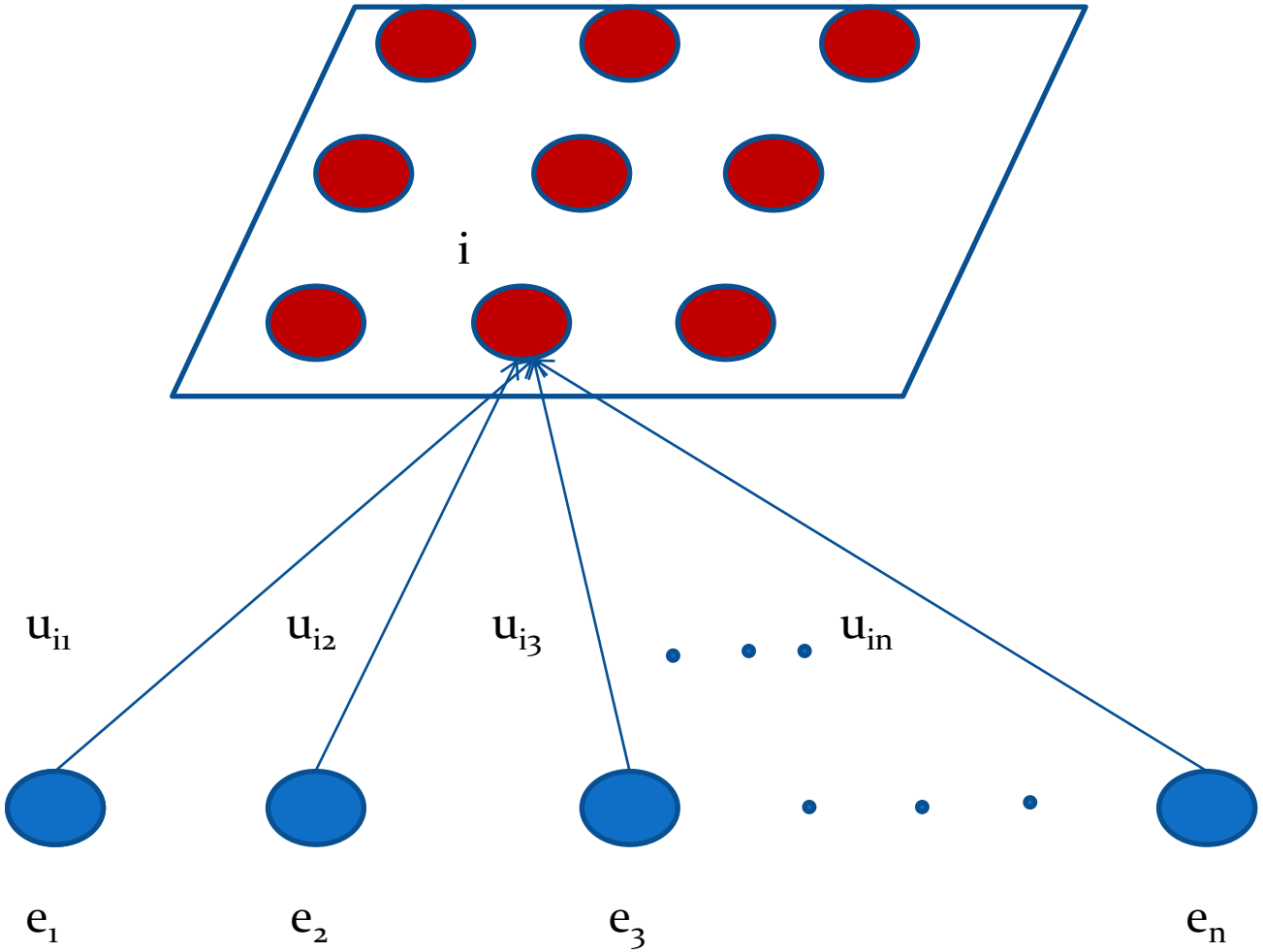
- When an input pattern is presented, each unit in the first layer takes on the value of the corresponding entry in the input pattern. This input pattern is denoted by

$$\boldsymbol{E} = [e_1, e_2, \ldots, e_n]$$

- The weights associated with connections from the input units to single unit (say i^{th} unit, here we identify each unit in the competitive layer by a single index, even though there is a two-dimensional grid of units in this layer) in the competitive layer are given by

$$\boldsymbol{U}_i = [\, u_{i1}, u_{i2}, \ldots, u_{in}]$$

Connections from the input vector to single unit in the competitive layer



Training Steps

The first step is to compute matching value for each unit in the competitive layer. This value measure the extent to which the weights each unit match the corresponding values of the input pattern. The matching value for the unit *i* is calculated by the following equation

$$\|E - U_i\| = \sqrt{\sum_j (e_j - u_{ij})^2}$$

which is the distance between the vectors $\boldsymbol{E}$ and $\boldsymbol{U_i}$

The unit with lowest matching value(the best match) wins the competition. Hence the unit with the best match is denoted as unit *k* and *k* is chosen such that

$$\|E - U_k\| = \min_i \left(\|E - U_i\| \right)$$

where the minimum is taken over all units *i* in the competitive layer.

If two or more units have same matching value then by convention, the unit with lower index value *i* is chosen

The next step after identifying the winning unit is to identify the neighborhood around it. The neighborhood consist of the units that are close to winner in the competitive layer grid. Here the neighborhood consists of a set of units that are within a square that is centered around the winning unit ***k*** and denoted by N_k.. The of the neighborhood can vary depending up on the distance between ***k*** to the edge of the neighborhood and denoted by ***d***. The weights are updated for all the units that are in the neighborhood of the winning unit N_k. by using the equations

$$\Delta u_{ij} = \begin{cases} \alpha\left(e_j - u_{ij}\right) & \text{if unit } i \text{ is in the neighborhood } N_k \\ 0 & \text{otherwise} \end{cases}$$

$$u_{ij}^{new} = u_{ij}^{old} + \Delta u_{ij}$$

This adjustment results in the winning unit and its neighbors having their weights modified , becoming more like input pattern. Now the winner become more likely to win the competition when the same or a similar input pattern will be presented subsequently.

Here two parameter need to be specified:

- The value of α, the learning rate parameter in the weight adjustment equation
- The size of the neighborhood N_k

Setting the Value for α and N_c

- Initially the learning rate α takes relative larger value and being decreased over the span of many iterations.
- This initial value of α is set by choice and is denoted by α_o (the typical choices are in the rangeo.2 to 0.5).
- The value of α for t^{th} is denoted by α_t determined by the equation

$$\alpha_t = \alpha_0\left(1-\frac{t}{T}\right)$$

where t → is the current training iteration

T → is the total number of training iteration to be done,

The value begins with α_o and decreases until it reaches zero

- The initial neighborhood width is relatively large, and is decreases over the progress of the training iterations. The center of the neighborhood is the winning unit k and at the position (x_k, y_k). Let d be the distance from k to the edge of the neighborhood. The neighborhood is then all (x, y) such that $k - d < x < k + d$ and $k - d < y < k + d$
- Sometimes the calculated neighborhood goes outside the grid of units in the competitive layer; in this case the actual neighborhood will be cut off at the edge.
- Initially d is assigned with a chosen value denoted by d_o (the typical value for d_o can be a half or a third of the width of the competitive layer of processing units. The value of d will be decreased according to the equation $$d = \left\lceil d_0 \left(1 - \frac{t}{T}\right) \right\rceil$$

where t → is the current training iteration and

T → is the total number of training iteration

This process assures a gradual linear decrease in d starting with d_0 and going down to 1

Summary

1. **Locate the unit in the competitive layer whose weights best match the input pattern.**
2. **Increase matching at this unit and its neighbors by adjusting their weights**
3. **Gradually decrease the size of the neighborhood and the amount of change to the weights as learning iteration progress**