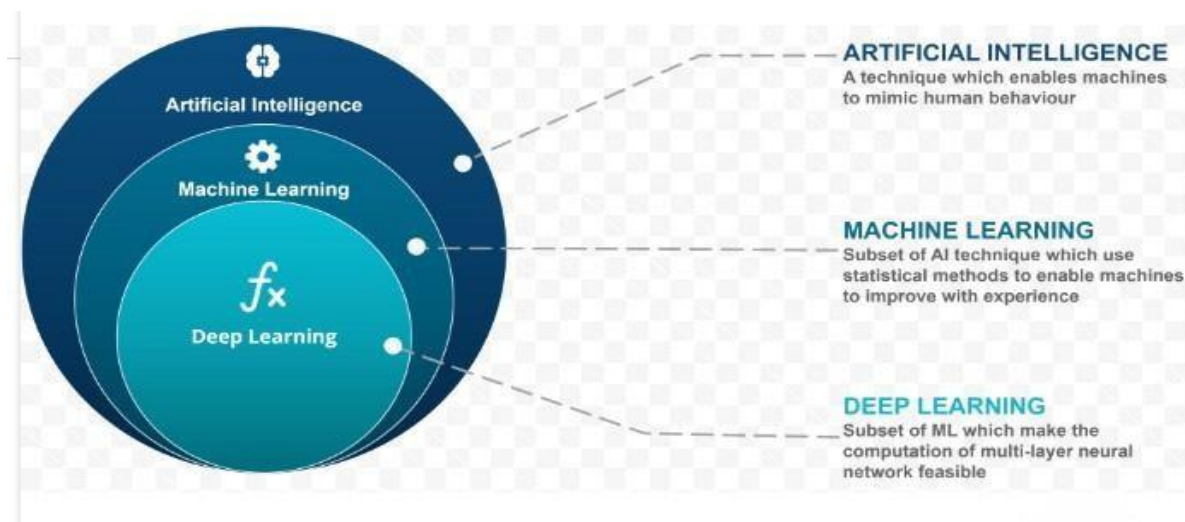


Module 1-notes

Deep Learning (BCA-701)

Module-1

MODULE-1
Introduction: What is a Neural Network?, The Human Brain, Models of a Neuron, Neural Networks Viewed As Directed Graphs, Feedback, Network Architectures, Rosenblatt's Perceptron: Introduction, Perceptron, The Perceptron Convergence Theorem, Relation Between the Perceptron and Bayes Classifier for a Gaussian Environment.



1. INTRODUCTION

1.1 Definition:

Neural Network or Artificial Neural Network (ANN) or Simulated Neural Network (SNN) is derived from Biological neural networks that develop the structure of a human brain.

- Neural networks are inspired by the **human brain**, which is a complex, nonlinear, parallel processing system.
- The brain can organize its structural constituents, known as neurons,
- The brain performs tasks like **pattern recognition** and **motor control** faster than digital computers.

Examples

- **Human vision** which is an information-processing task, it processes complex scenes in 100–200 ms.
- **Sonar**-an active echolocation System, **Bats** use **echolocation** to detect: Distance,

Velocity, Size, Shape, Direction (azimuth and elevation). These complex tasks are performed by a brain as small as a plum, outperforming man-made sonar systems.

- The brain is **structured at birth** but develops over time through **experience**. Most neural development happens in the **first two years**, but continues afterward. The brain exhibits **plasticity**, meaning it can adapt to its environment.

Definition of Neural Network

A **massively parallel distributed processor** made of simple processing units (artificial neurons).

- Has a **natural tendency to store and utilize experiential knowledge**.
- Mimics the brain in two key ways:
 1. **Learning** from the environment.
 - Stores and uses **experiential knowledge**.
 - Learns from the environment through a **learning algorithm**.
 2. **Storing knowledge** via synaptic weights (inter-neuron connections).
 - Knowledge is stored in **synaptic weights** (interconnections).
 - Learning modifies these weights to improve performance.

Like biological neurons, artificial networks can **modify their structure** (e.g., grow or prune connections).

- Aim to adjust weights to meet a **desired objective**.
- Related to concepts in **adaptive filter theory**.

Example:

- A neural network can be used for predicting housing prices.
- Email spam detection
- Used for sentiment analysis in natural language processing.
- Facial recognition.

Benefits of Neural Networks

Neural Network derives its computing power through,

1. Neural networks leverage massively parallel processing, enhancing computational efficiency.
 2. They can learn from data and generalize to unseen inputs, making them powerful in dynamic environments.
- Their combined processing capabilities allow them to approximate solutions to large-scale, intractable problems.
 - Neural networks work best when integrated into a larger system, assigned to specific tasks that match their strengths.
 - *Complex problems are broken into simpler tasks*, which neural networks handle effectively within their design limits.

Neural networks offer the following useful properties and capabilities:

1. Nonlinearity
2. Input–Output Mapping
3. Adaptivity
4. Evidential Response
5. Contextual Information.
6. Fault Tolerance.
7. VLSI Implementability.
8. Uniformity of Analysis and Design
9. Neurobiological Analogy.

1. Nonlinearity

- Neural networks use nonlinear neurons.
- Suitable for modelling complex, real-world nonlinear processes (e.g., speech).

2. Input–Output Mapping (Supervised Learning)

- Learns from labelled data by adjusting synaptic weights.

- Builds input-output mapping without assuming a fixed statistical model.
- Related to nonparametric statistical inference.

3. Adaptivity

- Adapts to environmental changes by updating weights.
- Balances stability vs. plasticity (the stability–plasticity dilemma).
- Useful in dynamic/nonstationary environments.

4. Evidential Response

- Provides confidence levels in decisions.
- Can reject ambiguous inputs to improve reliability.

5. Contextual Information

- Neurons influenced by overall network activity.
- Naturally captures and utilizes contextual knowledge.

6. Fault Tolerance

- Tolerates failures without catastrophic performance drop.
- Information is distributed—small damage has minimal effect.
- Requires robust training algorithms.

7. VLSI Implementability

- Neural networks' parallel structure fits well in VLSI hardware.
- Enables high-speed and efficient processing.

8. Uniformity of Analysis and Design

- Consistent structure (neurons/connections) across applications.
- Supports modularity and algorithm reuse.
- Provides a unified design framework.

9. Neurobiological Analogy

- Inspired by the human brain's parallel and fault-tolerant processing.

- A research tool for both engineers and neurobiologists.

1.2 HUMAN BRAIN

- The human nervous system may be viewed as a three-stage system.

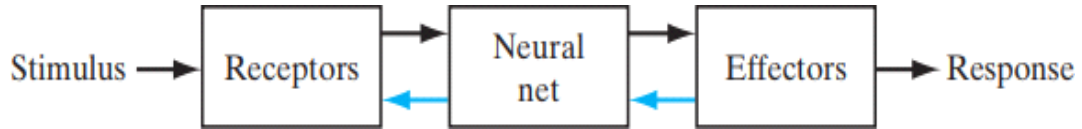


Figure 1: Block diagram representation of nervous system

- Two sets of arrows are shown in the figure.
- Those pointing from **left to right** indicate the *forward transmission of information*-bearing signals through the system.
- The arrows pointing from **right to left** signify the presence of *feedback in the system*.
- The **receptors** convert stimuli from the human body or the external environment into *electrical impulses* that convey information to the neural net (brain).
- Central to the system is the brain, represented by the **neural (nerve) net**, which continually receives information, perceives it, and makes appropriate decisions.
- The **effectors** convert electrical impulses generated by the neural net into discernible responses as system outputs.
- Typically, neurons are five to six orders of magnitude slower than silicon logic gates; events in a silicon chip happen in the nanosecond range, whereas neural events happen in the millisecond range. However, the brain makes up for the relatively slow rate of operation of a neuron by having a truly staggering number of neurons (nerve cells) with massive interconnections between them.
- It is estimated that there are approximately 10 billion neurons in the human cortex, and 60 trillion synapses or connections.

Biological Neuron Network

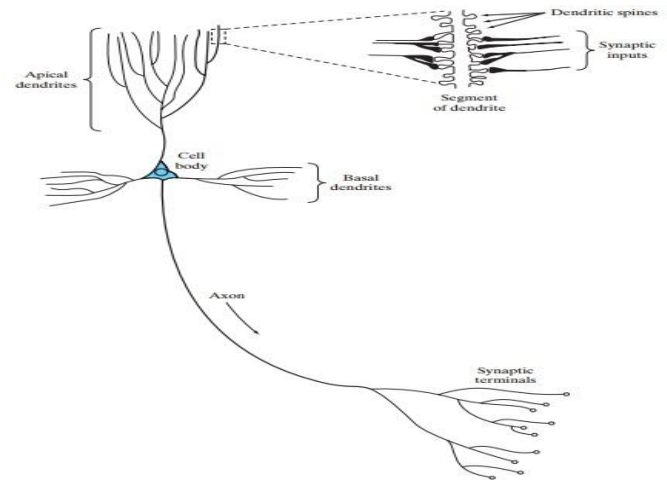
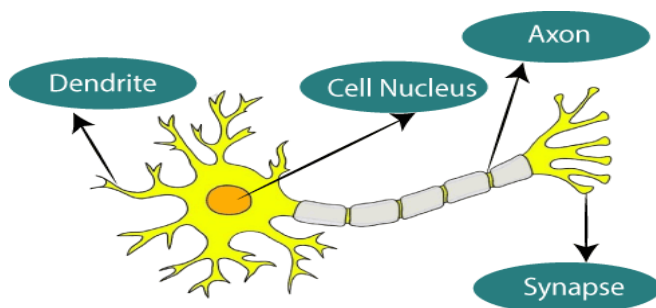


Figure 2: The

pyramidal cell.

As shown in the above diagram, a typical neuron consists of the following four parts, with the help of which we can explain its working –

Dendrites – They are tree-like branches, responsible for receiving the information from other neurons it is connected. In another sense, we can say that they are like the ears of a neuron.

Soma (or Cell Nucleus) – It is the cell body of the neuron and is responsible for processing of information they have received from dendrites.

Axon – It is just like a cable through which neurons send the information.

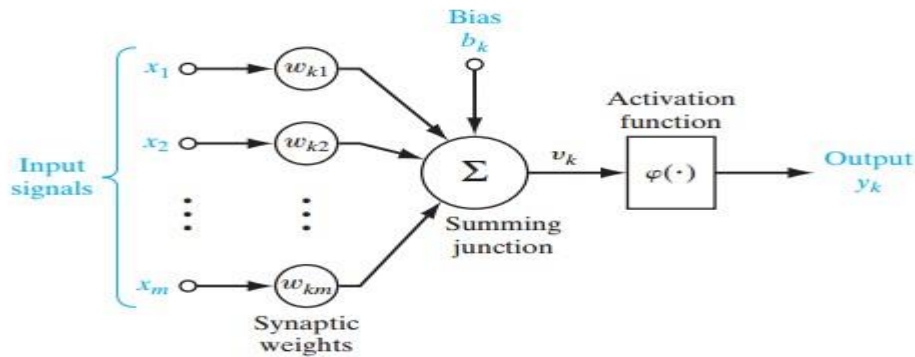
Synapses – It is the connection between the axon and other neuron dendrites. Synapses, or nerve endings, are elementary structural and functional units that mediate **the interactions between neurons**.

The most common kind of **synapse** is a chemical synapse, which operates as follows: A presynaptic process liberates a transmitter substance that diffuses across the synaptic junction between neurons and then acts on a postsynaptic process. Thus, a synapse converts a presynaptic electrical signal into a chemical signal and then back into a postsynaptic electric al signal.

1.3 MODELS OF A NEURON

A neuron is an information-processing unit that is fundamental to the operation of a neural network.

The block diagram of Fig. 3 shows the model of a neuron, which forms. The basis for designing a large family of neural networks.

Figure 3: Nonlinear model of a neuron, labeled k .

The three basic elements of the neural model:

1. synapses, or connecting links
2. adder
3. activation function

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_j at the input of **synapse j** connected to neuron **k** is **multiplied by the synaptic weight w_{kj}** .

In the weight notation w_{kj} :

- The **first subscript (k)** → refers to the **neuron receiving the input**.
- The **second subscript (j)** → refers to the **input source or synapse index**.

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 Sums all input signals after they are **weighted by their respective synaptic strengths**.

Performs a **linear combination** of the form:

$$u_k = \sum_{j=1}^m w_{kj} x_j \quad (1)$$

This summation is **linear** and forms the **net input** to the neuron.

3. An activation function(Squashing Function) for **limiting(squashes)** the amplitude of the neuron's output signal to a **finite range**.

Typically,

$[0, 1] \rightarrow$ e.g., Sigmoid function

$[-1, 1] \rightarrow$ e.g., Tanh function

Introduces **non-linearity**, essential for learning complex patterns.

Output:

$$y_k = \phi(v_k) \quad (2)$$

❖ Bias Term (b_k)

- **Externally applied parameter** to shift activation threshold.
- The **bias b_k** adjusts the **net input** to the activation function.
- It causes an **affine transformation** of the linear combiner output u_k , resulting in a new value called the **induced local field** or **activation potential** v_k

Adds flexibility to the neuron's response:

$$v_k = u_k + b_k \quad (3)$$

Alters the graph of v_k vs u_k , so it **no longer passes through the origin**.

- The bias b_k 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,

$$u_k = \sum_{j=1}^m w_{kj} x_j \quad (\text{Linear combiner})$$

$$v_k = u_k + b_k \quad (\text{Add bias})$$

$$y_k = \phi(v_k) = \phi \left(\sum_{j=1}^m w_{kj} x_j + b_k \right)$$

$$y_k = \phi(u_k + b_k)$$

where

$x_1, x_2, \dots, x_m$: Input signals

$w_{k1}, w_{k2}, \dots, w_{km}$: Synaptic weights of neuron k

u_k : Linear combiner output (sum of weighted inputs)

b_k : Bias of neuron k

$\phi(\cdot)$: Activation (or squashing) function

y_k : Output signal of neuron k

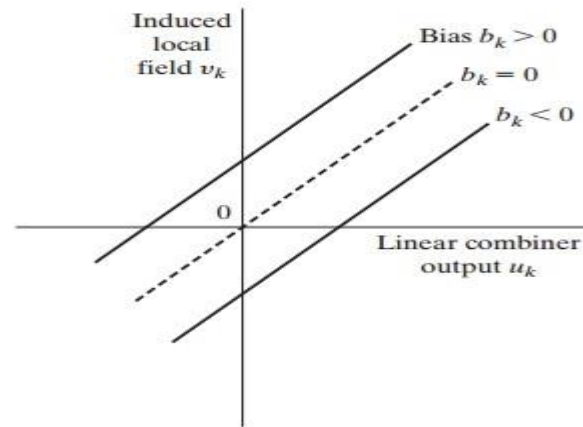


Figure 5 Affine transformation produced by the presence of a bias; note that $v_k = b_k$ at $u_k = 0$.

- If $b_k > 0$, the activation function shifts right; if $b_k < 0$, it shifts left.
- Due to this shift, the graph of v_k vs. u_k does not pass through the origin, as shown in Fig. 6
- This flexibility helps neurons learn complex decision boundaries during training.

Reformulated as:

$$v_k = \sum_{j=0}^m w_{kj} x_j$$

$$y_k = \varphi(v_k) \quad (5)$$

- Graphically and mathematically equivalent to previous model, more convenient in implementation (like in backpropagation).

In Eq. (4), we have added a new synapse. Its input is

$$x_0 = +1 \quad (6)$$

and its weight is

$$w_{k0} = b_k \quad (7)$$

1. When $b_k = 0$:

- $v_k = u_k$

- The output is a 45° diagonal line, shown as a dashed line (passing through the origin).
- This means no shift in the neuron's activation behavior.

2. When $b_k > 0$:

- $v_k = u_k + b_k$

The plot shifts upward, showing a positive shift in the activation potential.
The neuron reaches its threshold sooner, i.e., for smaller values of u_k .

3. When $b_k < 0$:

- $v_k = u_k + b_k$

The plot shifts downward, showing a negative shift.
The neuron requires a larger u_k to activate, i.e., it becomes less sensitive.

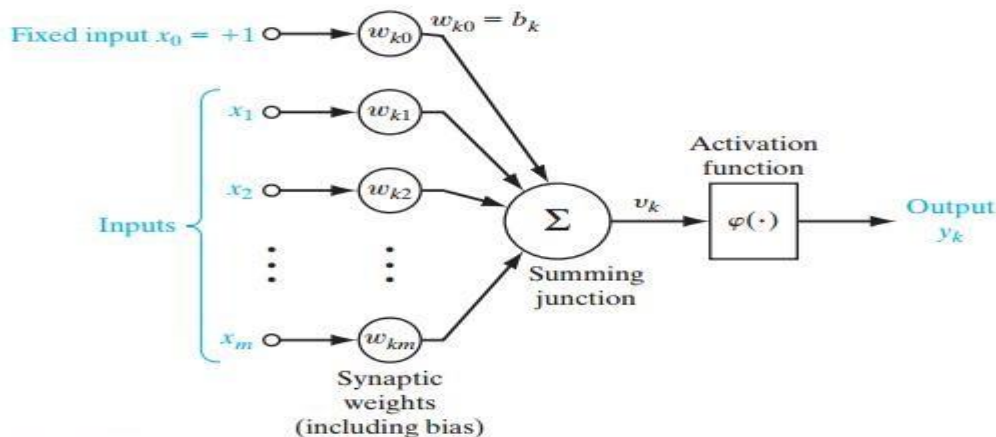


FIGURE 7 Another nonlinear model of a neuron; w_{k0} accounts for the bias b_k .

Types of Activation Function

The activation function, denoted by $\varphi(v)$, defines the output of a neuron in terms of the induced local field v . In what follows, we identify two basic types of activation functions:



1. Threshold Function. For this type of activation function, described in Fig. 8a, we have

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases} \quad (8)$$

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

$$y_k = \begin{cases} 1 & \text{if } v_k \geq 0 \\ 0 & \text{if } v_k < 0 \end{cases} \quad (9)$$

where v_k is the induced local field of the neuron; that is,

$$v_k = \sum_{j=1}^m w_{kj} x_j + b_k \quad (10)$$

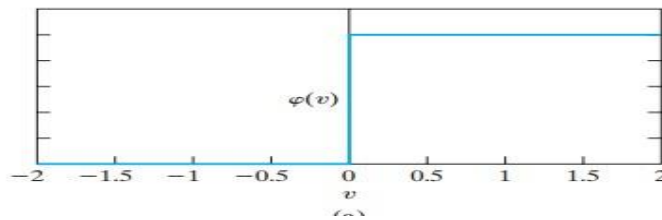


FIGURE 8 (a) Threshold function. (b) Sigmoid function for varying slope parameter a .

2. Sigmoid Function

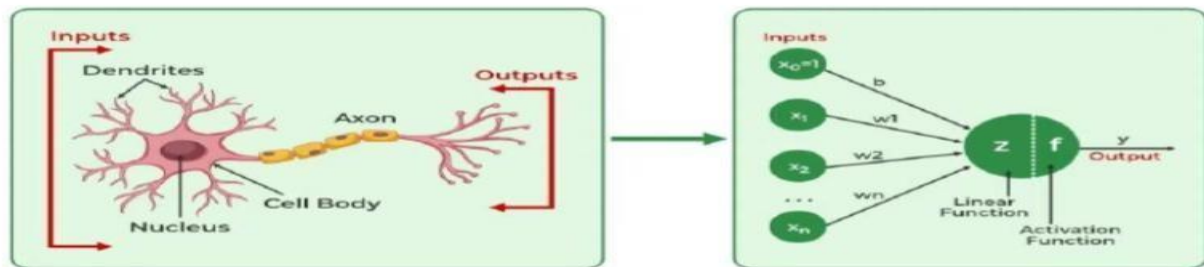
The sigmoid function is the most common activation function in neural networks, known for its S-shaped curve.

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

Where, a : **Slope parameter**, controls the steepness of the curve.

v : Induced local field (input to activation function).

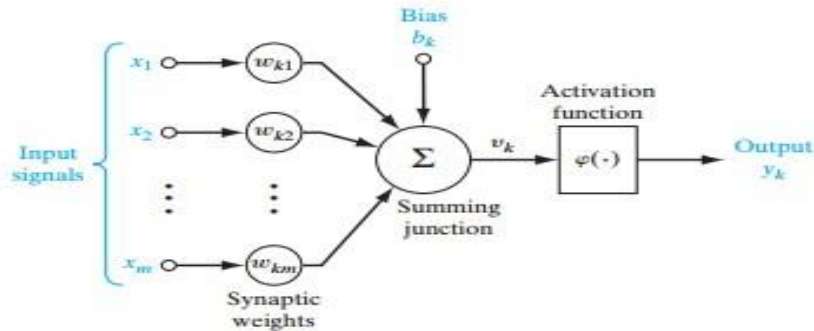
State how artificial neural networks are related with biological neural networks.



Each biological neuron has three main parts: the cell body, dendrites, and an axon. Dendrites receive incoming signals, and the axon sends signals out. In the same way Input layer acted as dendrites to receive incoming signals or data, Hidden layer is used to process data like cell body of biological neural networks and output layer is used to produce output like Axon

1.4 NEURAL NETWORKS VIEWED AS DIRECTED GRAPHS

The various elements that constitute the model of an artificial neuron.



Purpose:

- To **simplify the visual representation** of neural networks using **signal-flow graphs** while preserving all functional aspects of neuron models.

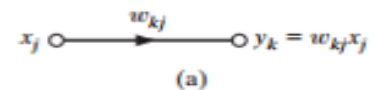
Signal-Flow Graphs: Key Concepts

- A **signal-flow graph** is a directed graph made up of:
 - **Nodes:** Represent variables or signals (e.g., x_j, y_k).
 - **Directed links (edges):** Represent relationships or transformations between nodes.

Types of Links:

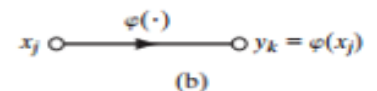
• Synaptic Links (Linear)

- Represent **weighted connections**.
- **Output:** $y_k = w_{kj} \cdot x_j$
- Fig (a) Synaptic weight applied to input.



• Activation Links (Nonlinear)

- Represent **activation function** (e.g., sigmoid, tanh).
- **Output:** $y_k = \varphi(x_j)$
- Fig (b) Nonlinear transformation.



Rules of Signal-Flow Graphs:

• Rule 1 – Directionality:

Rule 1. A signal flows along a link only in the direction defined by the arrow on the link.

Signal flows **only in the direction of the arrow**.

• Rule 2 – Node Summation:

Rule 2. A node signal equals the algebraic sum of all signals entering the pertinent node via the incoming links.

The signal at a node is the **sum of all incoming signals**.

→ Represents **synaptic convergence** (fan-in).

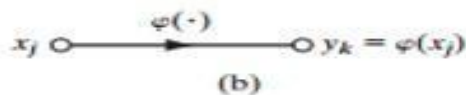
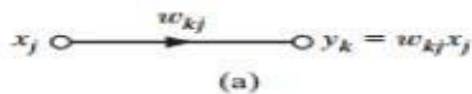
• Rule 3 – Fan-out:

Rule 3. The signal at a node is transmitted to each outgoing link originating from that node, with the transmission being entirely independent of the transfer functions of the outgoing links.

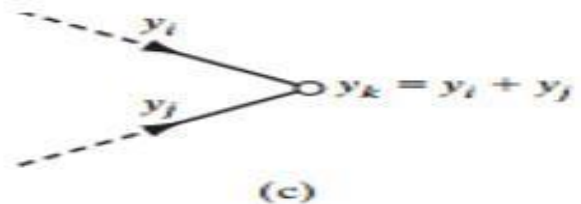
Signal from a node goes to **all outgoing links, independently** of their weights or functions.

→ Represents **synaptic divergence** (fan-out).

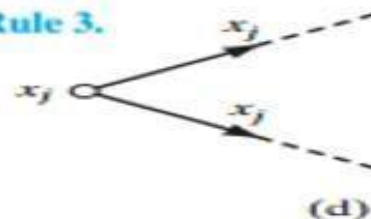
Rule 1.



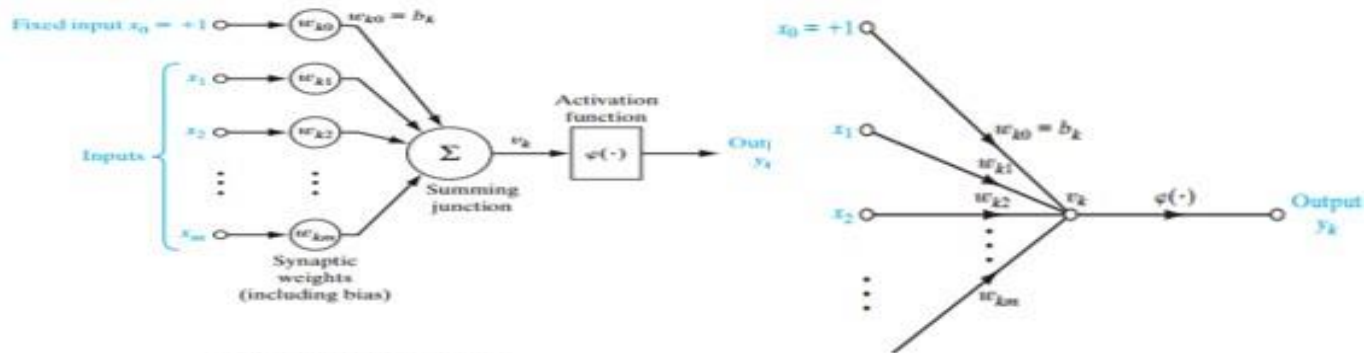
Rule 2.



Rule 3.



Neuron Model via Signal-Flow Graph



This graph includes:

- Inputs $x_0 = +1$ (bias), $x_1, x_2, \dots, x_m$
- Weights $w_{k0} = b_k, w_{k1}, \dots, w_{km}$
- Output y_k through activation function

Neural Network Definition as a Directed Graph:

A neural network is a directed graph consisting of nodes with interconnecting synaptic and activation links.

Or

A directed graph of nodes connected by synaptic (linear) and activation (possibly nonlinear) links.

Four properties characterize it:

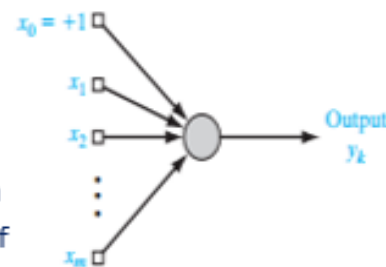
1. Each neuron is represented by a set of *linear synaptic links*, an externally applied *bias*, and a nonlinear *activation link*. The bias is represented by a synaptic link connected to an input fixed at +1.
2. Input signals are weighted by synaptic links.
3. The weighted sum of the input signals defines the induced local field of the neuron.
4. The activation link squashes the induced local field of the neuron to produce a final output.

Partially Complete Graph (Architectural Graph)

When **internal details of neurons** are ignored and the focus is only on **signal flow between neurons**, then this type of graph is said to be **partially complete**.

It is characterized as follows:

1. Source nodes(input layer) supply input signals to the graph.
 2. Each neuron is represented by a single node called a computation **nodes** (neurons)
 3. The *communication links* interconnecting the source and computation nodes of the graph carry no weight; they merely provide directions of signal flow in the graph.
- Shows **unweighted connections** for signal flow only



1.5 FEEDBACK

1.6 NETWORK ARCHITECTURE

There are several different architectures for ANNs, each with their own strengths and weaknesses. Some of the most common architectures include:

Feedforward Neural Networks: This is the simplest type of ANN architecture, where the information flows in **one direction from input to output**. The layers are fully connected, meaning each neuron in a layer is connected to all the neurons in the next layer.

There are three fundamental classes of ANN architectures:

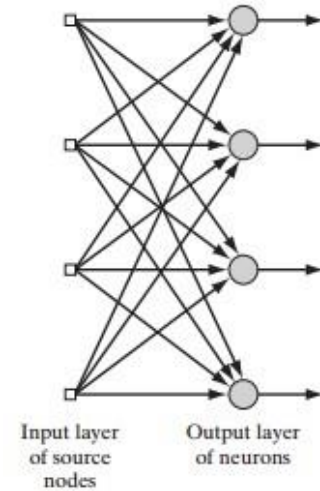
1. Single layer feed forward architecture
2. Multilayer feed forward architecture
3. Recurrent networks architecture

Single-layer Feedforward Networks:

The simplest kind of neural network is a **single-layer Feed-forward network**, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights.

In this way, it can be considered the simplest kind of feed-forward network. Layer is formed by taking processing elements and combining it with other processing elements.

- Input and output are linked with each other.
- Inputs are connected to the processing nodes with various weights, resulting in series of outputs per node.
- We do not count the input layer of source nodes because no computation is performed there.
- The sum of the products of the weights and the inputs is calculated in each node, and if the value is above some threshold (typically 0), the neuron fires and takes the activated value (typically 1); otherwise, it takes the deactivated value (typically -1).
- Neurons with this kind of activation function are also called *artificial neurons* or *linear threshold units*.
- In the literature, the term *perceptron* often refers to networks consisting of just one of these units.
- Warren McCulloch and Walter Pitts described a similar neuron in the 1940s.



Multilayer Feedforward Networks:

A **Multilayer feedforward neural network**, as its name suggests, is made up of **multiple layers**.

These architectures typically include:

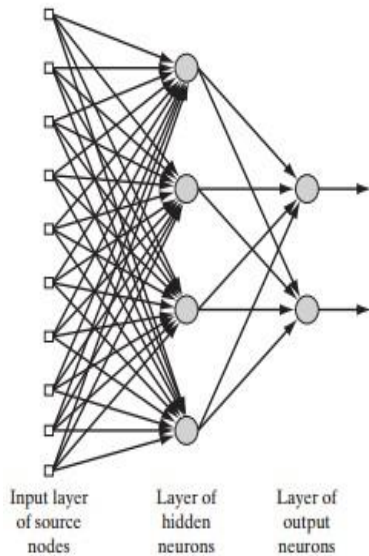
An **input layer**, which receives the raw data.

- One or more **intermediary (hidden) layers**, and
- An **output layer**, which produces the final prediction or result.

Architectures of this class not only process inputs and outputs but also include **hidden layers** intermediary layers between input and output.

- The neurons in these layers are called **hidden neurons** or **hidden units**, so named because they are not directly seen from either the input or the output of the network.

- These **hidden neurons perform useful intermediary computations**, allowing the network to transform and extract meaningful patterns from the input before passing information to the output layer.



- Adding one or more hidden layers enables the network to **extract higher-order features** from the input and gain a **global perspective** despite the typically local connectivity of neurons.

- It does not have any loops, no feedback, and no signal moves in backward directions.

- The **source nodes** in the input layer supply the **activation pattern** (input vector).

- These signals are passed forward to the **first hidden layer**.

- The **outputs of each hidden layer serve as inputs to the next layer**, continuing this pattern until the **output layer** is reached.

- The **final output layer provides the overall response** of the network to the given input.

Connectivity of the Network

- A network is called **fully connected** when **every node in a layer is connected to every node in the adjacent forward layer**.
- If some of these synaptic connections are missing, the network is considered **partially connected**.

Example Architectures, A network described as **10–4–2** has:

- 10 input (source) nodes,
- 4 hidden neurons in a single hidden layer,
- 2 output neurons.

More complex networks with multiple hidden layers are **m–h1–h2–q**, where:

- m = number of input nodes,
- h1 = number of neurons in the first hidden layer,
- h2 = number of neurons in the second hidden layer,
- q = number of output neurons.

Recurrent Neural Network (RNNs) Architecture:

Unlike **feedforward networks**, **Recurrent Neural Networks (RNNs)** include at least one **feedback loop**.

An **RNN** may consist of a **single layer of neurons** where each neuron's output is **fed back to the inputs of all other neurons** in the layer. This setup enables the network to maintain a form of memory,

where the **output from one time step influences subsequent time steps**. **Self-feedback** can also

occur, where

a neuron's output is fed back into its own input.

- This structure shows an RNN where there are no **self-feedback loops**—meaning that each neuron's

output is fed into the other neurons but not back into itself.

Self-feedback occurs when a neuron's output is fed back into its own input, a characteristic sometimes found in more complex.

RNN architectures.

- It has no **hidden neurons**.
- It only involves feedback between the output neurons, emphasizing the difference in structure compared to feedforward networks, which typically have hidden layers for intermediate computation.

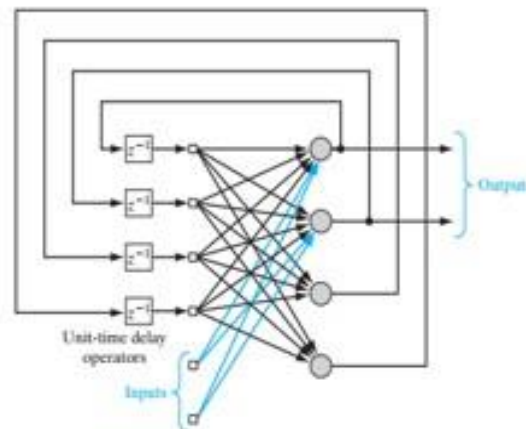
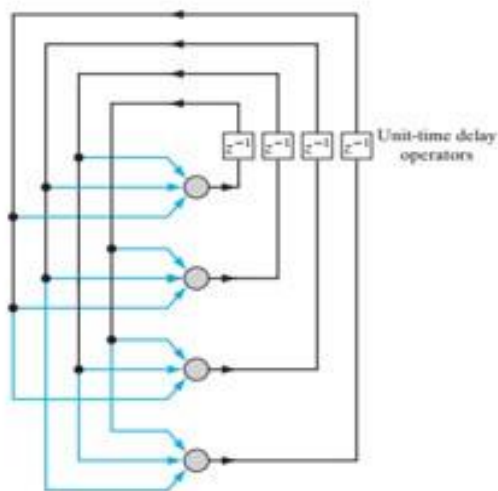


FIGURE 18 Recurrent network with hidden neurons.

- In more complex **RNNs**, as shown there are **hidden neurons**.
- These hidden neurons, like the output neurons, contribute to the feedback connections.
- The feedback in this case originates from both the **hidden neurons** and the **output neurons**, which further enriches the network's capability to store and propagate information.
- The **presence of feedback loops** profoundly **impacts the learning capability and performance** of the network.
- Feedback allows the network to maintain a form of memory of past inputs, which is crucial for tasks such as time-series prediction, speech recognition, and sequence modeling.
- These feedback loops incorporate elements like **unit-time delay elements** (denoted by z^{-1}).
- These delay elements introduce **nonlinear dynamic behaviour** into the network,

especially when the neural network contains **nonlinear units**.

- This dynamic behaviour helps the network learn more complex patterns, unlike the static nature of feedforward networks.

1.7 The Perceptron: A Basic Pattern Classifier

The **Perceptron** is the simplest form of a neural network and is used for **classifying linearly separable patterns**—i.e., patterns that can be separated into two classes by a hyperplane.

- The **Perceptron** consists of a **single neuron** with **adjustable synaptic weights** and a **bias**.
- The basic perceptron with a single neuron can only handle **binary classification**—two classes.

Multi-Class Classification-----However, by expanding the **output layer** of the perceptron to include more neurons, it can handle multi-class classification. Each neuron would correspond to a different class, but for the perceptron to work effectively, the classes still need to be linearly separable.

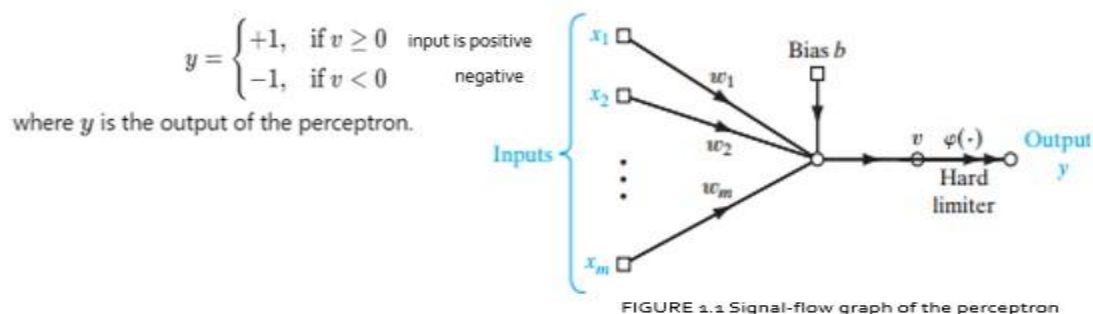
1.7.1 Neural Model of the Perceptron

The perceptron is based on a **McCulloch-Pitts model of a neuron**, which combines:

1. A **linear combiner** that computes the weighted sum of inputs.
2. A **hard limiter** (the sigum function) after calculating the weighted sum of inputs, producing an output of +1 or -1, depending on whether the summed value is positive or negative.

This is illustrated in **Fig. 1.1**, where the inputs are multiplied by their respective weights, summed together, and then passed through the hard limiter to generate the output.

The perceptron then passes this weighted sum through a hard limiter function:



Mathematical Representation:

- Inputs to the perceptron are represented by $x_1, x_2, \dots, x_m$ (with m being the number of input features).
- Each input has an associated **synaptic weight** $w_1, w_2, \dots, w_m$
- The **bias** term b is applied to shift the decision boundary.
- The perceptron computes a weighted sum of the inputs:

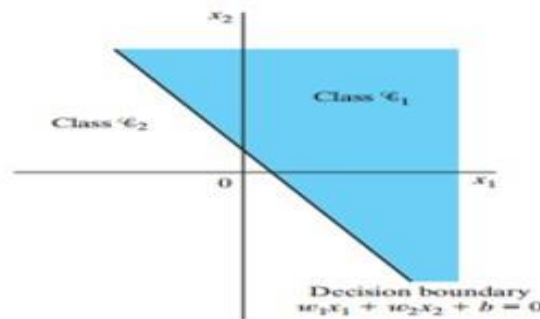
$$v = \sum_{i=1}^m w_i x_i + b$$

where:

- v is the induced local field (weighted sum),
- x_i are the input features,
- w_i are the weights associated with each input, and
- b is the bias.

Classification Decision:

- The perceptron classifies inputs into **two classes**:
 - If the output y is $+1$, the input belongs to class c_1 .
 - If the output y is -1 , the input belongs to class c_2 .

**Decision Boundary:**

- In the case of two input variables x_1 and x_2
- Defines a **hyperplane** that separates the two classes.
- The decision boundary is given by:

$$w_1x_1 + w_2x_2 + b = 0$$

This boundary can be visualized as a **line** in the two-dimensional case, as shown in **Fig. 1.2**.

Points above the line are classified as class C_1 and points below the line are classified as class C_2 .

- The **bias** b simply shifts the decision boundary along the axis, without affecting the slope.

Limitation of the Perceptron

- The perceptron is limited to solving classification problems where the classes are **linearly separable**.
- If the data is not linearly separable (i.e., no hyperplane can separate the classes), the perceptron will fail to converge, and no solution will be found.

1.8. Perceptron Convergence Theorem

This theorem **proves convergence** of the perceptron for linearly separable data. If training patterns are from **two linearly separable classes**, the algorithm **converges**.

- It positions a **decision surface (hyperplane)** between the classes.
- It adjusts the free parameters (weights and bias) based on training data.

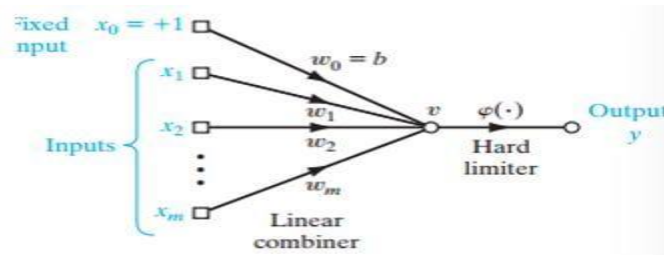


FIGURE 1.3 Equivalent signal-flow graph of the perceptron

Neuron Architecture:

1. Linear Combiner:

- Computes a **weighted sum** of inputs.
- **Bias** is added to this sum.

2. Induced Local Field:

- The **total sum** (input-weight products + bias) is called the **induced local field**.

3. Hard Limiter (Signum Function):

- The **hard limiter** applies a **nonlinear decision rule** to the linear combination.
- The **induced local field** is the **input to the hard limiter**.
- Acts on the induced local field.
- Produces a binary output (**+1 or -1**, typically).

$$v = \sum_{j=1}^m w_j x_j + b$$

where:

- v is the **induced local field**.
- w_j are the **weights**.
- x_j are the **inputs**.
- b is the **bias**.

Input Vector:

$$\mathbf{x}(n) = [+1, x_1(n), x_2(n), \dots, x_m(n)]^T$$

n is the time step (iteration number).

- The leading **+1** accounts for the **bias** in a unified vector form.

Weight Vector:

$$\mathbf{w}(n) = [b, w_1(n), w_2(n), \dots, w_m(n)]^T$$

The first component $w_0(n) = b$, represents the **bias weight**.

Linear Combiner Output (Induced Local Field)

$$v(n) = \sum_{i=0}^m w_i(n)x_i(n) = \mathbf{w}^T(n)\mathbf{x}(n) \quad (1.3)$$

This simplifies the computation of the neuron's pre-activation value.

Decision Surface:

- The equation $\mathbf{w}^T \mathbf{x} = 0$ defines a hyperplane in m-dimensional space.
- This hyperplane acts as the decision boundary between two classes.
- Classes are well separated = a straight line (2D) can divide them.

Fig. 1.4(a):

c_1 and c_2 are well apart.

You can draw a **straight line** between them → perceptron works.

Fig. 1.4(b):

The two classes are **too close or overlapping**.

The boundary needed to separate them is **curved or more complex** → single perceptron fails.

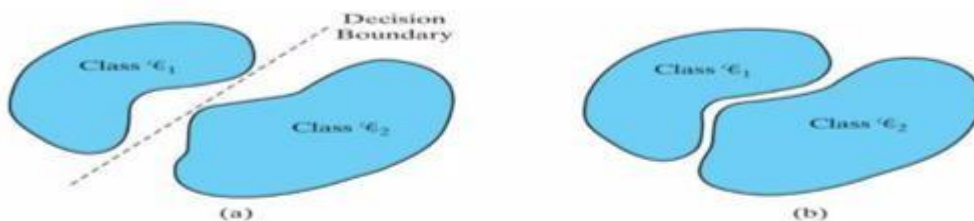


Figure 1.4 (a) A pair of linearly separable patterns. (b) A pair of non-linearly separable.

- Two sets of points (patterns) are **linearly separable** if there exists **at least one hyperplane** that divides them so that:
 - All points of class c_1 lie on one side.
 - All points of class c_2 lie on the other side.
- In 2D, this hyperplane is just a **straight line**.
- In 3D, it's a flat **plane**.
- In general, for m -dimensional data, it's an $(m - 1)$ -dimensional hyperplane.

Start with considering,

1. The perceptron's input variables come from **two linearly separable classes**.
2. Let $\mathcal{H}_1$ = subspace of training vectors
 $\mathbf{x}_1(1), \mathbf{x}_1(2), \dots$ belonging to class C_1 .
3. Let $\mathcal{H}_2$ = subspace of training vectors
 $\mathbf{x}_2(1), \mathbf{x}_2(2), \dots$ belonging to class C_2 .
4. The union of $\mathcal{H}_1$ and $\mathcal{H}_2$ forms the **complete space** $\mathcal{H}$.
5. The training process:
 - Uses vectors from $\mathcal{H}_1$ and $\mathcal{H}_2$ to train the classifier.
 - Adjusts the **weight vector** $\mathbf{w}$ so that C_1 and C_2 become **linearly separable**.
6. This means there exists a **weight vector** $\mathbf{w}$ such that the separation condition can be stated mathematically.

$$\mathbf{w}^T \mathbf{x} > 0 \quad \text{for all } x \in c_1$$

$$\mathbf{w}^T \mathbf{x} \leq 0 \quad \text{for all } x \in c_2$$

The perceptron updates weights only when **misclassification** occurs.

- **Case 1: Correct classification** → **No update** *(No correction is made to the weight vector of the perceptron)*

$$\mathbf{w}(n+1) = \mathbf{w}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathcal{C}_1$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathcal{C}_2 \quad \text{-----Eqn1}$$

- **Case 2: Misclassification** *(The weight vector of the perceptron is updated)*

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathcal{C}_2$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathcal{C}_1 \quad \text{-----Eqn2}$$

Where,

The learning-rate parameter $\eta(n)$ controls the adjustment applied to the weight vector at iteration n

- $\eta(n) > 0$ is the learning rate.
- For fixed-increment learning, $\eta(n) = 1$.

Proof of the Perceptron Convergence Algorithm (Rosenblatt, 1962)

Initial Conditions

1. Assume $w(0) = 0$.
2. Suppose $\mathbf{w}^T(n)\mathbf{x}(n) < 0$ for $n = 1, 2, \dots$ when $\mathbf{x}(n)$ belongs to class $\mathcal{H}_1$ (incorrect classification).

Weight Update Rule

3. For $\mathbf{x}(n) \in \mathcal{H}_1$, update:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{x}(n) \quad \text{-----Eqn3}$$

4. With $w(0) = 0$, iterative updates give:

$$\mathbf{w}(n+1) = \mathbf{x}(1) + \mathbf{x}(2) + \dots + \mathbf{x}(n) \quad \text{-----Eqn4}$$

Existence of a Solution

5. Since classes are linearly separable, there exists a solution vector $\mathbf{w}_o$ such that:

$$\mathbf{w}_o^T \mathbf{x}(n) > 0 \quad \forall \mathbf{x}(n) \in \mathcal{H}_1 \quad \text{-----Eqn5}$$

6. Define:

$$\alpha = \min_{\mathbf{x}(n) \in \mathcal{H}_1} \mathbf{w}_o^T \mathbf{x}(n) > 0 \quad \text{-----Eqn6}$$

Lower Bound on Norm

7. Multiply update equation by $\mathbf{w}_o^T$ to get: $\mathbf{w}(n+1) = \mathbf{x}(1) + \mathbf{x}(2) + \dots + \mathbf{x}(n)$

$$\mathbf{w}_o^T \mathbf{w}(n+1) \geq n\alpha \quad \text{-----Eqn7}$$

Next we make use of an inequality known as the Cauchy-Schwarz inequality. Given two vectors $\mathbf{w}_o$ and $\mathbf{w}(n+1)$, the *Cauchy-Schwarz inequality* states that

$$\|\mathbf{w}_o\| \cdot \|\mathbf{w}(n+1)\| \geq n\alpha \quad \text{-----Eqn8}$$

$$\|\mathbf{w}_o\|^2 \|\mathbf{w}(n+1)\|^2 \geq [\mathbf{w}_o^T \mathbf{w}(n+1)]^2$$

$$\|\mathbf{w}_o\|^2 \|\mathbf{w}(n+1)\|^2 \geq n^2 \alpha^2$$

Therefore, we can write,

$$\|\mathbf{w}(n+1)\|^2 \geq \frac{n^2 \alpha^2}{\|\mathbf{w}_o\|^2} \quad \text{-----Eqn9}$$

Upper Bound on Norm

Eqn3, we have

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{x}(n)$$

start from the update (for $k = 1, \dots, n$ and $x(k) \in \mathcal{H}_1$)

$$w(k+1) = w(k) + x(k) \quad \text{-----Eqn10}$$

Take the squared Euclidean norm of both sidesThe squared norm of a vector v is $\|v\|^2 = v^T v$. So

$$\|w(k+1)\|^2 = \|w(k) + x(k)\|^2 = (w(k) + x(k))^T (w(k) + x(k))$$

Expand the right-hand side:

$$\|w(k+1)\|^2 = w(k)^T w(k) + x(k)^T x(k) + 2w(k)^T x(k), \text{ but considering } w^T(k)x(k) \leq 0$$

$$\|w(k+1)\|^2 \leq \|w(k)\|^2 + \|x(k)\|^2. \quad \text{-----Eqn11}$$

From Eqn 3 we have,

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{x}(n)$$

Squaring on both sides,

$$\mathbf{w}(n+1)^2 = \mathbf{w}(n)^2 + \mathbf{x}(n)^2$$

If, $\|\mathbf{w}(0)\|^2 = 0$ gives

$$\|\mathbf{w}(n+1)\|^2 \leq \sum_{k=1}^n \|\mathbf{x}(k)\|^2. \quad \text{-----Eqn11}$$

Define,

$$\beta = \max_k \|\mathbf{x}(k)\|^2$$

$$\|\mathbf{w}(n+1)\|^2 \leq \sum_{k=1}^n \|\mathbf{x}(k)\|^2 \leq n\beta$$

$$\|w(n+1)\|^2 \leq n\beta. \quad \text{-----Eqn12}$$

From eqn 9, we have,

$$\|\mathbf{w}(n+1)\|^2 \geq \frac{n^2 \alpha^2}{\|\mathbf{w}_o\|^2} \quad \text{We can write, } \frac{n^2 \alpha^2}{\|\mathbf{w}_o\|^2} \leq n\beta \quad \text{-----Eqn13}$$

$$\frac{n^2 \alpha^2}{\|\mathbf{w}_o\|^2} \leq n\beta$$

$$n \leq n_{\max} = \frac{\beta \|\mathbf{w}_o\|^2}{\alpha^2}$$

Therefor, we have,

$$n_{\max} = \frac{\beta \|\mathbf{w}_o\|^2}{\alpha^2}$$

The solution vector $\mathbf{w}(n)$ becomes constant after $n_0 \leq n_{\max}$ iterations.

TABLE 1.1 Summary of the Perceptron Convergence Algorithm

Variables and Parameters:

$\mathbf{x}(n)$ = $(m + 1)$ -by-1 input vector
 $= [+1, x_1(n), x_2(n), \dots, x_m(n)]^T$

$\mathbf{w}(n)$ = $(m + 1)$ -by-1 weight vector
 $= [b, w_1(n), w_2(n), \dots, w_m(n)]^T$

b = bias

$y(n)$ = actual response (quantized)

$d(n)$ = desired response

η = learning-rate parameter, a positive constant less than unity

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \dots$

2. *Activation.* At time-step n , activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.

3. *Computation of Actual Response.* Compute the actual response of the perceptron as

$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

where $\text{sgn}(\cdot)$ is the signum function.

4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain

$$\mathbf{w}(n + 1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

where

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathcal{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathcal{C}_2 \end{cases}$$

5. *Continuation.* Increment time step n by one and go back to step 2.

1.9 Relation between the perceptron and Bayes classifier for a Gaussian environment

The perceptron bears a certain relationship to a classical pattern classifier known as the Bayes classifier. When the environment is Gaussian, the Bayes classifier reduces to a linear classifier. This is the same form taken by the perceptron. However, the linear nature of the perceptron is *not* contingent on the assumption of Gaussianity.

Bayes Classifier

In the *Bayes classifier*, or *Bayes hypothesis testing procedure*, we minimize the *average risk*, denoted by

$$\mathcal{R} = \sum_{i=1}^2 \sum_{j=1}^2 c_{ij} p_j \int_{h_i} p_X(x|C_j) dx \quad \text{-----Eq(1)}$$

For a two-class problem, represented by classes c_1 and c_2 , the average risk is defined by Van Trees (1968) as

$$\begin{aligned} \mathcal{R} = & c_{11}p_1 \int_{\mathcal{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1) d\mathbf{x} + c_{22}p_2 \int_{\mathcal{X}_2} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2) d\mathbf{x} \\ & + c_{21}p_1 \int_{\mathcal{X}_2} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1) d\mathbf{x} + c_{12}p_2 \int_{\mathcal{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2) d\mathbf{x} \end{aligned} \quad \text{-----Eq(2)}$$

The first two terms on the right-hand side of Eq.(2) represent correct decisions (i.e., correct classifications). The last two terms represent incorrect decisions (i.e., misclassifications).

Each decision is weighted by the product of two factors:

1. The cost involved in making the decision
2. The relative frequency (i.e., prior probability) with which it occurs

- **The intention is to determine a strategy for the minimum average risk.**
- Because we require that a decision be made, each observation vector $\mathbf{x}$ must be assigned in the overall observation space $\mathcal{X}$ to either $\mathcal{X}_1$ or $\mathcal{X}_2$.

$$\mathcal{X} = \mathcal{X}_1 + \mathcal{X}_2$$

Accordingly, we may rewrite Eq. (2) in the equivalent form

$$\begin{aligned} \mathcal{R} = & c_{11}p_1 \int_{\mathcal{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1) d\mathbf{x} + c_{22}p_2 \int_{\mathcal{X}-\mathcal{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2) d\mathbf{x} \\ & + c_{21}p_1 \int_{\mathcal{X}-\mathcal{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1) d\mathbf{x} + c_{12}p_2 \int_{\mathcal{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2) d\mathbf{x} \end{aligned} \quad \text{-----Eq(3)}$$

The Bayes rule says:

Assign x to the class with the higher posterior probability $p(C_i|x)$, adjusted for the cost.

where $c_{11} < c_{21}$ and $c_{22} < c_{12}$. We now observe the fact that

$$\int_{\mathcal{X}} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1) d\mathbf{x} = \int_{\mathcal{X}} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2) d\mathbf{x} = 1$$

Hence, Eq. (3) reduces to

$$\begin{aligned} \mathcal{R} = & c_{21}p_1 + c_{22}p_2 \\ & + \int_{\mathcal{X}_1} [p_2(c_{12} - c_{22}) p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2) - p_1(c_{21} - c_{11}) p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1)] d\mathbf{x} \end{aligned} \quad \text{-----Eq(4)}$$

The first two terms on the right-hand side of Eq.(4) represent a fixed cost.

$$r = r_{\text{fixed}} + \int_{x_1} [p_2(c_{12} - c_{22})p_X(x|C_2) - p_1(c_{21} - c_{11})p_X(x|C_1)] dx$$

Here, the expression inside [] is the **integrand**.

- If the integrand is **negative** → assigning x to x_1 (class C_1) *reduces* risk, so put it in C_1 .
- If the integrand is **positive** → assigning x to x_1 *increases* risk, so assign it to C_2 .
- If the integrand is zero → no effect, assign arbitrarily.

-----Eq(5)

On this basis, we may now formulate the Bayes classifier as follows:

If the condition

$$p_1(c_{21} - c_{11}) p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1) > p_2(c_{12} - c_{22}) p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2)$$

holds, assign the observation vector $\mathbf{x}$ to subspace $\mathcal{X}_1$ (i.e., class $\mathcal{C}_1$). Otherwise assign $\mathbf{x}$ to $\mathcal{X}_2$ (i.e., class $\mathcal{C}_2$).

If costs are equal ($c_{12} = c_{21}$, $c_{11} = c_{22} = 0$), the threshold reduces to:

$$\frac{p_X(x|C_1)}{p_X(x|C_2)} \geq \frac{p_2}{p_1}$$

which is the standard **Bayes classifier**.

Likelihood ratio,

-----Eq(6)

Threshold of the test

$$\xi = \frac{p_2(c_{12} - c_{22})}{p_1(c_{21} - c_{11})}$$

$$\Lambda(\mathbf{x}) = \frac{p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1)}{p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2)} \quad \text{-----Eq(7)}$$

The quantity $\Lambda(\mathbf{x})$, the ratio of two conditional probability density functions, is called the **likelihood ratio**. The quantity ξ is called the **threshold** of the test. Note that both $\Lambda(\mathbf{x})$ and ξ are always positive. In terms of these two quantities, we may now reformulate the Bayes classifier by stating the following

If, for an observation vector $\mathbf{x}$, the likelihood ratio $\Lambda(\mathbf{x})$ is greater than the threshold ξ , assign $\mathbf{x}$ to class $\mathcal{C}_1$. Otherwise, assign it to class $\mathcal{C}_2$.

Figure 1.5a depicts a block-diagram representation of the Bayes classifier. The important points in this block diagram are twofold:

1. The data processing involved in designing the Bayes classifier is confined entirely to the computation of the likelihood ratio $\Lambda(\mathbf{x})$.

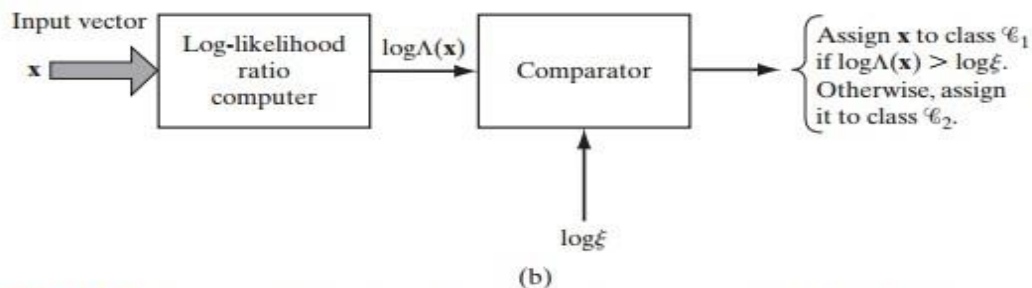
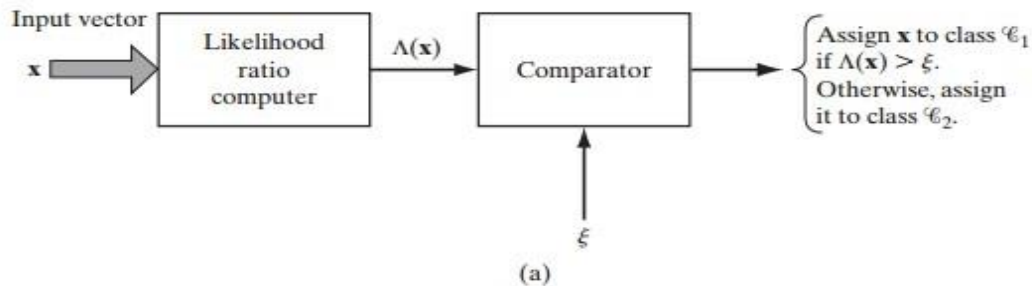


FIGURE 1.5 Two equivalent implementations of the Bayes classifier: (a) Likelihood ratio test, (b) Log-likelihood ratio test.

2. This computation is completely invariant to the values assigned to the prior probabilities and costs involved in the decision-making process. These quantities merely affect the value of the threshold ξ .

Bayes Classifier for a Gaussian Distribution

Consider now the special case of a two-class problem, for which the underlying distribution is Gaussian. The random vector $\mathbf{X}$ has a mean value that depends on whether it belongs to class $\mathcal{C}_1$ or class $\mathcal{C}_2$, but the covariance matrix of $\mathbf{X}$ is the same for both classes. That is to say,

$$\begin{aligned} \text{Class } \mathcal{C}_1: \quad & \mathbb{E}[\mathbf{X}] = \boldsymbol{\mu}_1 \\ & \mathbb{E}[(\mathbf{X} - \boldsymbol{\mu}_1)(\mathbf{X} - \boldsymbol{\mu}_1)^T] = \mathbf{C} \\ \text{Class } \mathcal{C}_2: \quad & \mathbb{E}[\mathbf{X}] = \boldsymbol{\mu}_2 \\ & \mathbb{E}[(\mathbf{X} - \boldsymbol{\mu}_2)(\mathbf{X} - \boldsymbol{\mu}_2)^T] = \mathbf{C} \end{aligned}$$

The covariance matrix $\mathbf{C}$ is nondiagonal, which means that the samples drawn from classes $\mathcal{C}_1$ and $\mathcal{C}_2$ are *correlated*. It is assumed that $\mathbf{C}$ is nonsingular, so that its inverse matrix $\mathbf{C}^{-1}$ exists.

With this background, we may express the conditional probability density function of $\mathbf{X}$ as the multivariate Gaussian distribution

$$p_{\mathbf{x}}(\mathbf{x}|\mathcal{C}_i) = \frac{1}{(2\pi)^{m/2}(\det(\mathbf{C}))^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \mathbf{C}^{-1}(\mathbf{x} - \boldsymbol{\mu}_i)\right), \quad i = 1, 2$$

where m is the dimensionality of the observation vector $\mathbf{x}$.

-----Eq(8)

Assume the following:

1. The two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ are equiprobable:

$$p_1 = p_2 = \frac{1}{2} \quad (1.31)$$

2. Misclassifications carry the same cost, and no cost is incurred on correct classifications:

$$c_{21} = c_{12} \quad \text{and} \quad c_{11} = c_{22} = 0 \quad (1.32)$$

Specifically, by substituting Eq. 8 into 6 and taking the natural logarithm, we get

$$\begin{aligned} \log \Lambda(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_1)^T \mathbf{C}^{-1}(\mathbf{x} - \boldsymbol{\mu}_1) + \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_2)^T \mathbf{C}^{-1}(\mathbf{x} - \boldsymbol{\mu}_2) \\ &= (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \mathbf{C}^{-1} \mathbf{x} + \frac{1}{2}(\boldsymbol{\mu}_2^T \mathbf{C}^{-1} \boldsymbol{\mu}_2 - \boldsymbol{\mu}_1^T \mathbf{C}^{-1} \boldsymbol{\mu}_1) \end{aligned}$$

If above equation is linear classifier, as described by the relation

$$y = \mathbf{w}^T \mathbf{x} + b$$

where

$$\mathbf{y} = \log \Lambda(\mathbf{x})$$

$$\mathbf{w} = \mathbf{C}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

$$b = \frac{1}{2}(\boldsymbol{\mu}_2^T \mathbf{C}^{-1} \boldsymbol{\mu}_2 - \boldsymbol{\mu}_1^T \mathbf{C}^{-1} \boldsymbol{\mu}_1)$$

If the output y of the linear combiner (including the bias b) is positive, assign the observation vector $\mathbf{x}$ to class $\mathcal{C}_1$. Otherwise, assign it to class $\mathcal{C}_2$.

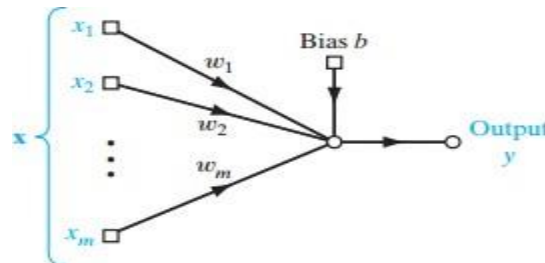
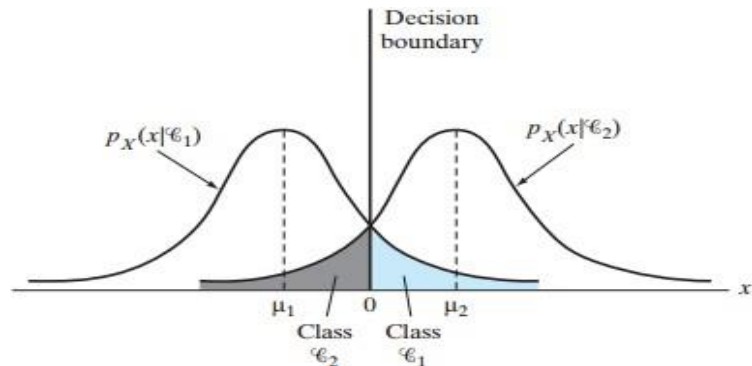


Figure : Signal-flow graph of Gaussian classifier.

- The perceptron operates on the premise that the patterns to be classified are *linearly separable*. The Gaussian distributions of the two patterns assumed in the derivation of the Bayes classifier certainly do *overlap* each other and are therefore *not* separable. The extent of the overlap is determined by the mean vectors

FIGURE 1.7 Two overlapping, one-dimensional Gaussian distributions.



μ_1 and μ_2 and the covariance matrix $\mathbf{C}$. The nature of this overlap is illustrated in Fig. 1.7 for the special case of a scalar random variable (i.e., dimensionality $m = 1$). When the inputs are nonseparable and their distributions overlap as illustrated, the perceptron convergence algorithm develops a problem because decision boundaries between the different classes may oscillate continuously.

- The Bayes classifier minimizes the probability of classification error. This minimization is independent of the overlap between the underlying Gaussian distributions of the two classes. For example, in the special case illustrated in Fig. 1.7, the Bayes classifier always positions the decision boundary at the point where the Gaussian distributions for the two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ cross each other.
- The perceptron convergence algorithm is *nonparametric* in the sense that it makes no assumptions concerning the form of the underlying distributions. It operates by concentrating on errors that occur where the distributions overlap. It may therefore work well when the inputs are generated by nonlinear physical mechanisms and when their distributions are heavily skewed and non-Gaussian. In contrast, the Bayes classifier is *parametric*; its derivation is contingent on the assumption that the underlying distributions be Gaussian, which may limit its area of application.
- The perceptron convergence algorithm is both adaptive and simple to implement; its storage requirement is confined to the set of synaptic weights and bias. On the other hand, the design of the Bayes classifier is fixed; it can be made adaptive, but at the expense of increased storage requirements and more complex computations.