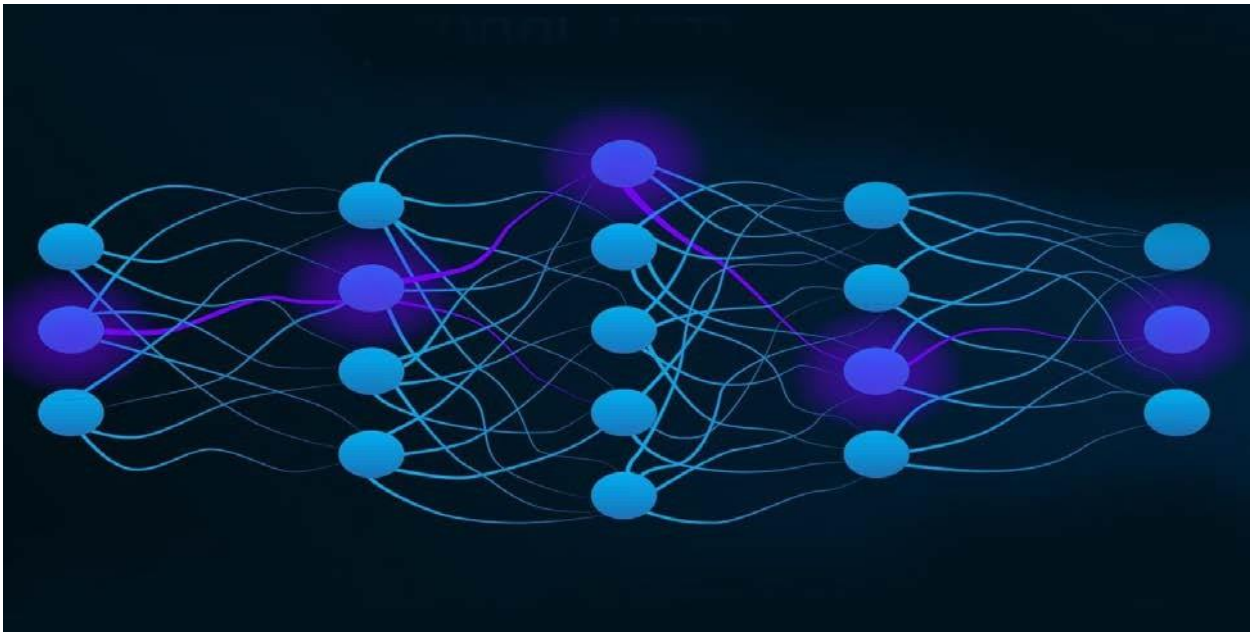
NARSIMHA REDDY ENGINEERING COLLEGE
An Autonomous Institution| Affiliated to JNTUH | Approved by AICTE
Accredited by NBA & NAAC with 'A' Grade

ARTIFICIAL NEURAL NETWORKS



IV BTECH I SEM

SUBJECT NOTES

DEPARTMENT OF ECE,NRCM A.DILEEP ASSISTANT PROFESSOR

UNIT 1

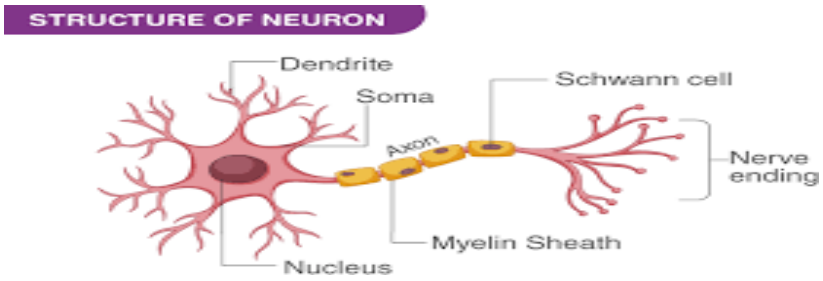
Neural Network, Human Brain, and Models of a Neuron:

The study of neural networks is deeply inspired by the human brain, which processes information through a vast network of interconnected neurons. Understanding neural networks requires exploring their biological inspiration—the human brain—along with different models of a neuron used in artificial intelligence.

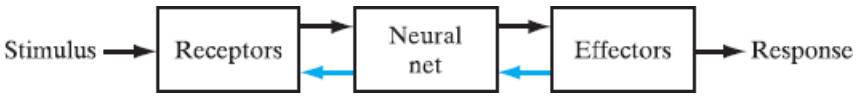
Structure of a Biological Neuron

A **biological neuron** has several key components:

- **Dendrites**: Receive signals from other neurons.
- **Cell body (Soma)**: Processes the received signals.
- **Axon**: Transmits the processed signal to other neurons.
- **Synapse**: A connection between two neurons where neurotransmitters help transmit the signal.



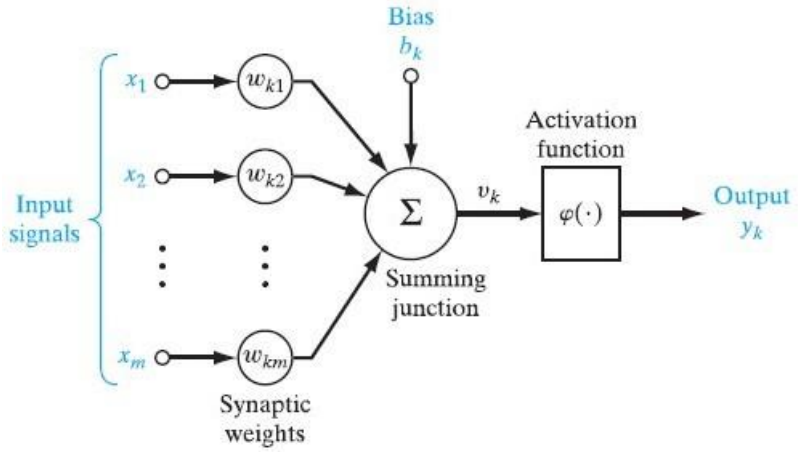
The brain operates using a network of these neurons, forming complex pathways responsible for cognition and perception.



Models of an Artificial Neuron

(a) McCulloch-Pitts Model (1943)

- One of the earliest mathematical models of a neuron.
- Uses **binary inputs** and a simple threshold function.
- If the weighted sum of inputs exceeds a threshold, the neuron activates (outputs 1), otherwise, it remains inactive (outputs 0).



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

$$y_k = \varphi(u_k + b_k) \qquad (2)$$

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

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \qquad (4)$$

Perceptron Model (1958)

- Introduced by **Frank Rosenblatt**.
- Similar to the McCulloch-Pitts model but includes **adjustable weights**.
- Uses a learning algorithm to update weights based on errors.

Sigmoid Neuron Model

- Overcomes the perceptron's limitation by introducing a **smooth activation function**.

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

Neural Networks viewed as Directed Graphs:

Two different types of links may be distmguished:
Synaptic links, whose behavior is governed by a linear input-output relation. Specif-
ically, the node signal x is multiplied by the synaptic weight wy to produce the hode signal yk, as illustrated in Fig. 9a.
Activation links, whose behavior is governed in general by a nonlinear **input-**

output
relation. This form of relationship is illustrated in Fig. 9b, where o(.) is the non-

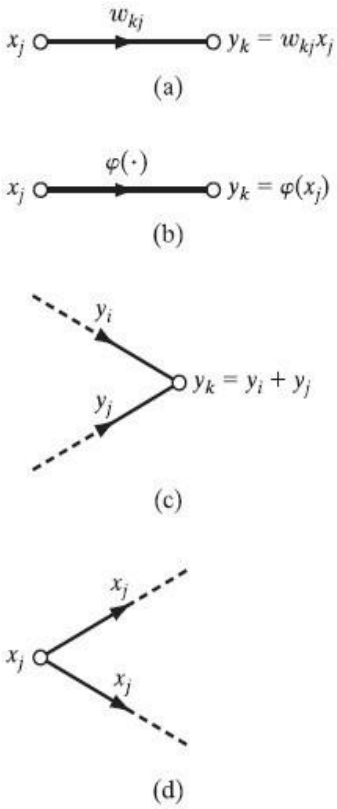


FIGURE 9 Illustrating basic rules for the construction of signal-flow graphs.

linear activation function.
Rule 2. A node signal equals the algebraic sum of all signals entering the pertinentnode via the incoming links.
This second rule is illustrated in Fig. 9c for the case of synaptic convergence, or fan-in.
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.

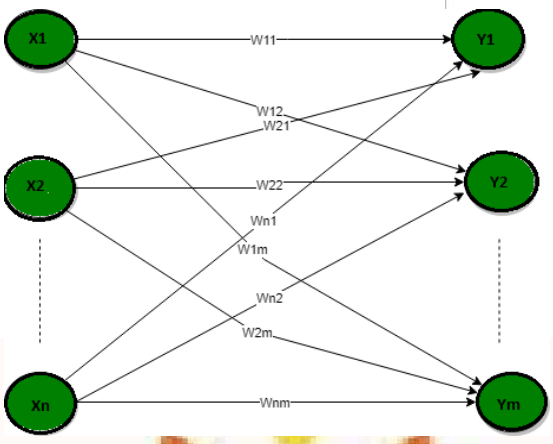
Network Architectures:

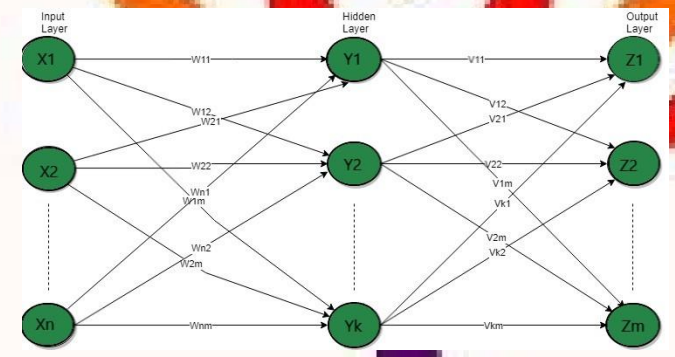
There exist five basic types of neuron connection architecture :

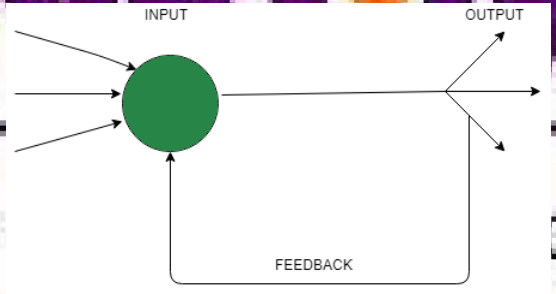
1. Single-layer feed-forward network
2. Multilayer feed-forward network
3. Single node with its own feedback
4. Single-layer recurrent networks
5. Multilayer recurrent network

Single-layer feed-forward network:

In this type of network, we have only two layers input layer and the input layer does not count because no computation is performed in this layer. The output layer is formed when different weights are applied to input nodes and the cumulative effect per node is taken. After this, the neurons collectively give the output layer to compute the output signals.







Single-layer recurrent network:

The above network is a single-layer network with a feedback connection in which the processing element's output can be directed back to itself or

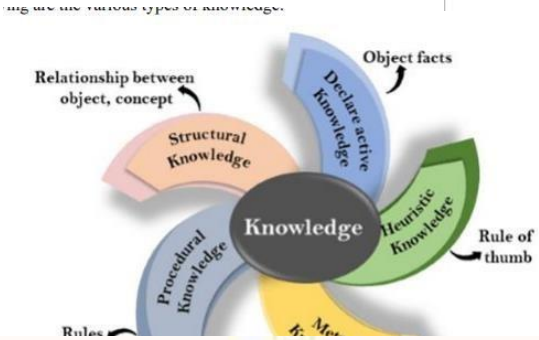
to another processing element or both. A recurrent neural network is a class of artificial neural networks where connections between nodes form a directed graph along a sequence. This allows it to exhibit

Events: Events are the actions which occur in our world.

Performance: It describe behavior which involves knowledge about how to do things. Meta-knowledge: It is knowledge about what we know.

Facts: Facts are the truths about the real world and what we represent.

Knowledge-Base: The central component of the knowledge-based agents is the knowledge base. It is represented as KB.



Characteristics of BNN and ANN

Characteristics	Biological Neural Network	Artificial Neural Network
Speed	Processes information at a slower rate. Response time is measured in milliseconds.	Information is processed at a faster rate. The response time is measured in nanoseconds.
Processing	Massively parallel processing.	Serial processing.
Size & Complexity	An extremely intricate and dense network of linked neurons of the order of 1011 neurons and 1015 interconnections.	Size and complexity are reduced. It is incapable of performing sophisticated pattern recognition tasks.
Storage	An extremely intricate and dense network of linked neurons with 1015 interconnections, including neurons on the order of 1011.	The term "replaceable information storage" refers to the practice of replacing fresh data with old data.
Fault tolerance	The fact that information storage is flexible means that new information may be added by altering the connectivity strengths without deleting existing information.	Intolerant of faults. In the event of a system failure, corrupt data cannot be recovered.
Control Mechanism	There is no unique control mechanism outside of the computational task.	Controlling computer activity is handled by a control unit.

your roots to success...

- A core concept is that neural networks are powerful function approximators.
- Given an input x, a neural network aims to learn a function f(x) that approximates the desired output y.
- This can be represented as

$$y \approx f(x; w)$$

Where:

- x is the input data.
- y is the desired output.
- f is the neural network function.
- w represents the network's weights and biases, which are the parameters that the network learns.

Learning through Gradient Descent:

- Neural networks learn by adjusting their weights w to minimize a loss function L, which measures the difference between the predicted output f(x; w) and the actual output y.
- The most common learning algorithm is gradient descent, which updates the weights iteratively:

$$w(t+1) = w(t) - \eta \nabla L(w(t))$$

Where:

- $w(t)$ are the weights at iteration t.
- η is the learning rate, which controls the step size.
- $\nabla L(w(t))$ is the gradient of the loss function with respect to the weights.

Activation Functions:

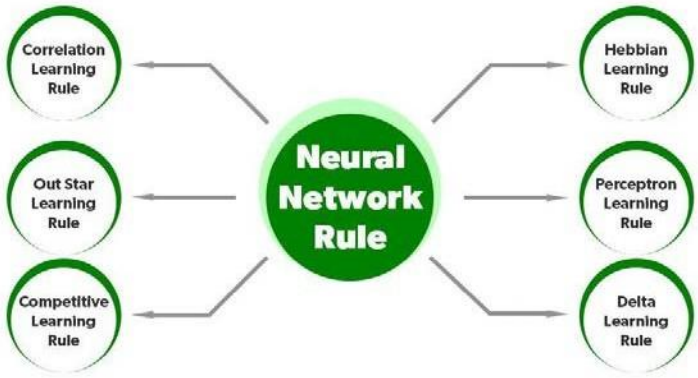
Activation functions introduce non-linearity into the network, allowing it to learn complex patterns.

Common activation functions include:

- Sigmoid: $\sigma(x) = 1 / (1 + e^{-x})$
- ReLU: $ReLU(x) = max(0, x)$
- Tanh: $\tanh(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x})$

LEARNING PROCESS:

The learning rule enhances the Artificial Neural Network's performance by applying this rule over the network. Learning in ANNs involves adjusting the weights of the connections between neurons to improve performance on a given task. This adjustment is governed by learning rules, which define how weights are updated based on input data, output, and error.



1. Hebbian Learning Rule

Principle: *This rule is based on the biological concept that "neurons that fire together, wire together.*

Hebbian Learning Rule is an unsupervised learning algorithm used in neural networks to adjust the weights between nodes. It is based on the principle that the connection strength between two neurons should change depending on their activity patterns.

The rule can be summarized as follows:

- When two neighboring neurons operate in the same phase at the same time, the weight between them increases.
- If the neurons operate in opposite phases, the weight between them decreases.
- When there is no signal correlation between the neurons, the weight remains unchanged.

The sign of the weight between two nodes is determined by the sign of their input signals:

- If both nodes receive inputs that are either positive or negative, the resulting weight is strongly positive.
- If one node's input is positive while the other's is negative, the resulting weight is strongly negative.

Mathematical Formulation:

$$\delta w = \alpha x_i y$$

Here:

- δw is the change in weight.
- α is the learning rate.
- x_i represents the input vector.
- y is the output.

2. **Perceptron Learning Rule**

Principle: *This rule adjusts weights to minimize classification errors.*

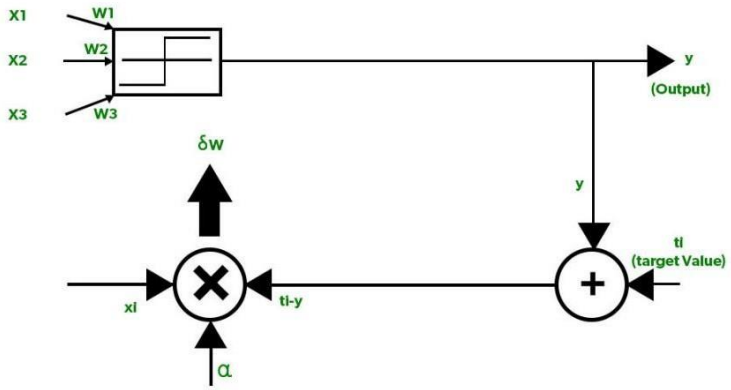
Perceptron Learning Rule is an error-correcting algorithm designed for single-layer feedforward networks. It is a supervised learning approach that adjusts weights based on the error calculated between the desired and actual outputs. Weight adjustments are made only when an error is present. The process is computed as follows:

- $(x_1, x_2, x_3, \ldots, x_n)$: Set of input vectors
- $(w_1, w_2, w_3, \ldots, w_n)$: Set of weights
- y: Actual output
- w_0: Initial weight
- w_{new}: New weight
- δw: Change in weight
- α: Learning rate

3. **Output Calculation:** The final output is based on the net input and the activation function applied to it:

$$y = \begin{cases} 1, & \text{if net input} \geq \theta \\ 0, & \text{if net input} < \theta \end{cases}$$

This rule provides the foundation for learning in perceptrons, enabling them to make adjustments and improve performance iteratively.



3. Delta Learning Rule

Principle: *Minimizes the error between the actual output and the desired output using gradient descent.*

Delta Learning Rule is a supervised learning algorithm that uses a continuous activation function. Also known as the **Least Mean Square (LMS) method**, it aims to minimize the error across all training patterns. This rule is based on the **gradient descent approach**, which iteratively reduces error by updating the weights of the network.

Computed as follows:

- $(x_1, x_2, x_3, \ldots, x_n)$: Set of input vectors
- $(w_1, w_2, w_3, \ldots, w_n)$: Set of weights
- y: Actual output
- w_0: Initial weight
- $w$new: Updated weight
- δw: Change in weight
- Error $= t_i - y$: Difference between the target (t_i) and actual output (y)

4. Correlation Learning Rule

Principle: *Updates weights based on the correlation between input and output signals.*

Correlation Learning Rule is based on principles similar to the Hebbian Learning Rule. Specifically, it states that:

- If two neighboring neurons operate in the same phase at the same time, the weight between them becomes more positive.
- Conversely, if the neurons operate in opposite phases, the weight becomes more negative.

However, unlike the Hebbian rule, the Correlation Rule is **supervised**. This means it uses the target response for calculating the change in weight.

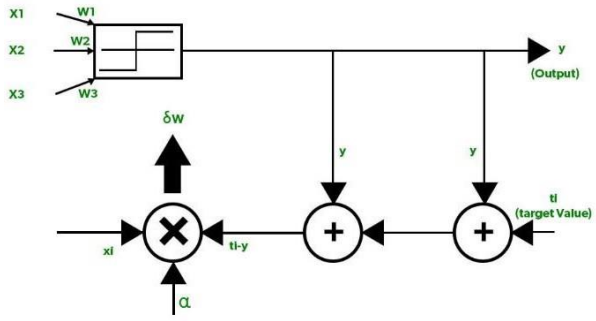
The change in weight (δw) is calculated as:

$$\delta w = \alpha x_i t_j$$

where:

- δw: Change in weight
- α: Learning rate
- x_i: Input vector
- t_j: Target value

This supervised approach allows for more precise adjustments based on the desired output, improving the learning process.



5. **Boltzmann Learning RulePrinciple**: *Inspired by statistical mechanics, this rule uses probabilities to adjust weights.*

Boltzmann Learning Rule is a probabilistic learning method commonly used in stochastic networks such as Boltzmann Machines. It utilizes principles from statistical mechanics to find the optimal weight configuration that minimizes energy in the network.

- **Probabilistic Framework:** Learning is governed by the probability of a node's state, influenced by the system's energy.
- **Energy Minimization:** Weights are updated to minimize the network's energy, ensuring a stable and efficient configuration.

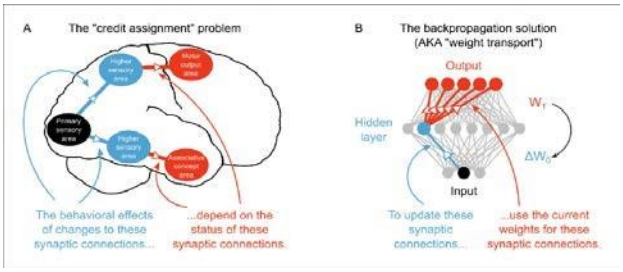
The change in weight (δw) is calculated using the correlation between the states of the input and output nodes:

$$\delta w = \alpha(P\text{data} - P\text{model})$$

Where:

- δw: Change in weight
- α: Learning rate
- $P\text{data}$: Probability of the data
- $P\text{model}$: Probability of the model's predictions

Credit Assignment Problem:



The "credit assignment problem" is a fundamental challenge in neural networks, particularly in deep learning and reinforcement learning. In essence, it's the difficulty of determining which neurons or actions are responsible for a particular outcome. Here's a breakdown:

Core Concept:

- **Attributing Responsibility:**
  - When a neural network produces a result (whether correct or incorrect), the credit assignment problem asks: "Which parts of the network contributed to that result, and to what degree?"
  - This is crucial for learning, as we need to know which connections to strengthen or weaken to improve performance.

Challenges:

- **Hierarchical Networks:**
  - In deep neural networks with multiple layers, it's hard to trace the impact of a neuron in an early layer on the final output.
  - The effect of a single neuron's activity is propagated through

many layers, making it difficult to isolate its contribution.

- **Temporal Delays:**
  - In reinforcement learning, there can be a significant delay between an action and its resulting reward.
  - The agent needs to figure out which of its past actions led to the reward, even if those actions occurred many steps earlier.

Solutions and Approaches:

- **Backpropagation:**
  - This is the most widely used technique in supervised deep learning.
  - It calculates the gradient of the error with respect to each weight in the network, allowing us to determine how much each weight contributed to the error.
- **Reinforcement Learning Techniques:**
  - **Temporal Difference (TD) learning:**
    - These methods learn from predictions and updates based on the difference between predicted and actual rewards.
  - **Monte Carlo methods:**
    - These methods learn from complete episodes of experience, assigning credit to actions based on the total reward received.
  - **Eligibility traces:**
    - These techniques keep track of the recent history of actions, allowing the agent to assign credit to actions that occurred earlier in a sequence.

Memory, Adaption:

We know that, during ANN learning, to change the input/output behavior, we need to adjust the weights. Hence, a method is required with the help of which the weights can be modified. These methods are called Learning rules, which are simply algorithms or equations. Following are some learning rules for the neural network –**Hebbian Learning Rule**This rule, one of the oldest and simplest, was introduced by Donald Hebb in his book *The Organization of Behavior* in 1949. It is a kind of feed-forward, unsupervised learning.

Mathematical Formulation − According to Hebbian learning rule, following is the formula to increase the weight of connection at every time step.

$$\Delta w_{ji}(t) = \alpha x_i(t).y_j(t)$$

Here, $\Delta w_{ji}(t)$ = increment by which the weight of connection increases at time step **t**

α = the positive and constant learning rate

$x_i(t)$ = the input value from pre-synaptic neuron at time step **t**

$y_i(t)$ = the output of pre-synaptic neuron at same time step **t**

Perceptron Learning Rule

This rule is an error correcting the supervised learning algorithm of single layer feedforward networks with linear activation function, introduced by Rosenblatt.

Basic Concept – as being supervised in nature, to calculate the error, there would be a comparison between the desired/target output and the actual output. If there is any difference found, then a change must be made to the weights of connection.

Mathematical Formulation – To explain its mathematical formulation, suppose we have 'n' number of finite input vectors, xn, along with its desired/target output vector tn, where n = 1 to N.

Now the output 'y' can be calculated, as explained earlier on the basis of the net input, and activation function being applied over that net input can be expressed as follows –

$$y = f(y_{in}) = \begin{cases} 1, & y_{in} > \theta \\ 0, & y_{in} \leqslant \theta \end{cases}$$

Where **θ** is threshold.

The updating of weight can be done in the following two cases –

Case I – when **t ≠ y**, then

$$w(new) = w(old) + tx$$

Case II – when **t = y**, then

No change in weight

Delta Learning Rule

Widrow–HoffRuleWidrow–HoffRule

It is introduced by Bernard Widrow and Marcian Hoff, also called Least Mean Square LMSLMS method, to minimize the error over all training patterns. It is kind of supervised learning algorithm with having continuous activation function.

Basic Concept – The base of this rule is gradient-descent approach, which continues forever. Delta rule updates the synaptic weights so as to minimize the net input to the output unit and the target value.

Mathematical Formulation – to update the synaptic weights, delta rule is given by

$$\Delta w_i = \alpha . x_i . e_j$$

Here Δw_i = weight change for i^{th} pattern;

α = the positive and constant learning rate;

x_i = the input value from pre-synaptic neuron;

e_j = $(t - y_{in})$, the difference between the desired/target output and the actual output y_{in}

The above delta rule is for a single output unit only. The updating of weight can be done in the following two cases –

Case-I – when **t** ≠ **y**, then

$$w(new) = w(old) + \Delta w$$

Case-II – when **t** = **y**, then

No change in weight

Condition to be a winner – Suppose if a neuron y_k wants to be the winner then there would be the following condition –

$$y_k = \begin{cases} 1 & if\ v_k > v_j\ for\ all\ j,\ j \neq k \\ 0 & otherwise \end{cases}$$

t means that if any neuron, say yk, wants to win, then its induced local field the output of summation unit the output of summation unit, say vk, must be the largest among all the other neurons in the network.

Condition of sum total of weight – Another constraint over the competitive learning rule is, the sum total of weights to a particular output neuron is going to be 1. For example, if we consider neuron **k** then –

$$\sum_j w_{kj} = 1 \qquad for\ all\ k$$

- **Change of weight for winner** – If a neuron does not respond to the input pattern, then no learning takes place in that neuron. However, if a particular neuron wins, then the corresponding weights are adjusted as follows

$$\Delta w_{kj} = \begin{cases} -\alpha(x_j - w_{kj}), & if\ neuron\ k\ wins \\ 0, & if\ neuron\ k\ losses \end{cases}$$

Here α is the learning rate.

This clearly shows that we are favoring the winning neuron by adjusting its weight and if there is a neuron loss, then we need not bother to re-adjust its weight.

Outstar Learning Rule

This rule, introduced by Grasberg, is concerned with supervised learning because the desired outputs are known. It is also called Grasberg learning.

Basic Concept – this rule is applied over the neurons arranged in a layer. It is specially designed to produce a desired output **d** of the layer of **p** neurons.

Mathematical Formulation – The weight adjustments in this rule are computed as follows

$$\Delta w_j = \alpha\,(d - w_j)$$

Here **d** is the desired neuron output and α is the learning rate

Statistical Nature of the Learning Process: This analysis focuses on the deviation between a target function f(x) and the actual function F(x,w) derived by the NN, where x denotes the input signal. By examining this deviation, we can gain insights into the effectiveness of the NN and identify areas for improvement .

Environment Setting

Let's consider a scenario with N realization of a random vector X, denoted by: {xi}i=1N{*xi*}*i*=1*N* and corresponding set of random scalar D denoted by: {di}i=1N{*di*}*i*=1*N*.
These measurements constitute the training sample that is denoted by:

$$\mathcal{T} = \{(x_i, d_i)\}_{i=1}^{N}$$

We assume the regressive model:
D=f(X) +ϵ *D*=*f*(*X*) +ϵ
Where

- f(X) is the deterministic function of its argument vector
- ϵis the random

expectational error

Properties of Regressive Model

Regressive model has two properties:

1. Zero Mean Error

- The mean value of expectation error is zero: E[ϵ |x]=0E[ϵ|*x*]=0 where, E is statistical expectation operator.
- Based on this property, we can say regression function f(x) is the conditional mean of model output D, given that input X=x shown by: f(x)=E[D|x]*f*(*x*)=E[*D*|*x*]

2. Orthogonality Principle

- The expectation error is uncorrelated with regression function f(X): E[ϵ f(X)]=0E[ϵ*f*(*X*)]=0
- This property is called principle of orthogonality which states that all the information about D available to us through input X has been encoded into regression function f(X).

Neural Network Response

The actual response *Y* of the NN to input *X* is:
Y=F(X, w) *Y*=*F*(*X*, *w*)
Where F(X, w) is the NN's input-output function. Cost function

The cost function *ε*(*w*) is the squared difference between the desired response and the actual response *y* of the N, averaged over the training data T:

E(w) -1Er [(d-F(=,T))=]
ICy is average operator taken over training sample T.
By adding and subtracting f(x) to the argument (d - F(e,T)) . we can write it as:
d-F(z,T) = (d-f(=))+(f(=) -F(z,T)) =< +(f(=) - F(z,T))
Substituting this equation in cost function:

The last expectation term on the right hand side is zero for two reasons:
So, the equation is reduced to:

where. {Er[e*] represents intrinsic error because it is independent of weight vector w.

Bias-Variance Trade-Off

Observations:

1. The term B(w) is the bias of the average value of the approximating function F(x,T), measured with respect to the regression function f(x)=E[D|X=x](*x*)=E[*D*|*X*=*x*]. This term represents the inability of the neural network defined by the function F(x,w)to accurately approximate the regression function f(x)=E[D|X=x]*f*(*x*)=E[*D*|*X*=*x*]. We may therefore view the bias B(w) as an approximation error.
2. The term V(w) is the variance of the approximating function *F*(*x*,*w*), measured over the entire training sample T. This second term represents the inadequacy of the information contained in the training sample T about the regression function f(x). We may therefore view the variance V(w) as the manifestation of an estimation error.