Learning:

Learning is the process by which the free parameters (synaptic weight, bias, activation function, network model) of a neural network are adapted through a continuing process of stimulation (input) by the environment in which the network is embedded (stable, fixed). The type of learning is determined by the manner in which the parameter change takes place.

This definition of the learning process implies the following sequence of events:

1) The neural network is stimulated by the environment (Input signal)
2) The neural network undergoes changes as a result of this stimulation (updates the value of synaptic weight, bias, activation function, network architecture (deleting or adding nodes))
3) The neural network responds in a new way to the environment because of the changes that have occurred to its internal structure.

Learning - The process by which weights and biases are adjusted to achieve some desired network behavior.

Parameters in a Neural Network:

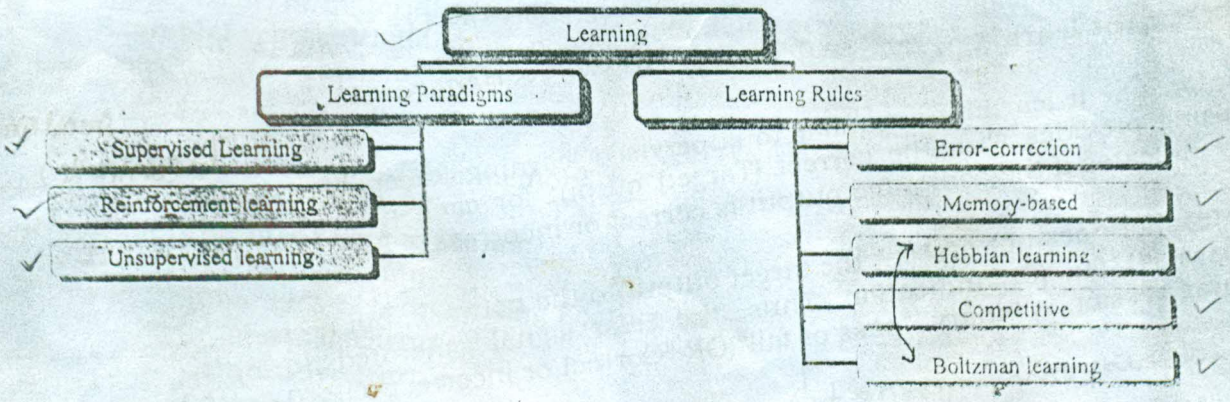
Typically:
- Weights
- Bias coefficients

Less Common:
- The activation function
- Network architecture

Learning deals with three things:

i) **Learning Paradigms:** Refers to a model of the environment in which the neural network operates
ii) **Learning rules:** Methods of deriving the next changes that might be made in a network) OR a procedure for modifying the weights and biases of a network.
iii) **Learning algorithm:** Well structured formula of deriving the next changes that might be made in a network

Learning can be classified as follows:

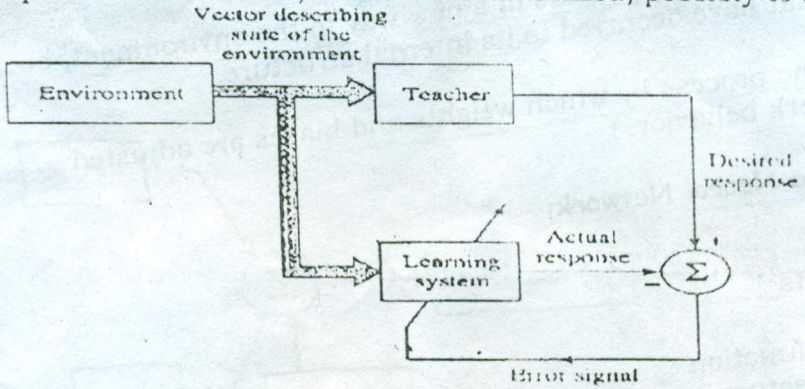


Supervised Learning:

- Every input pattern that is used to train the network is associated with an output pattern or target pattern known as desired output.

  {P1, T1}, {P2, T2} .....{Pn, Tn} Ex: AND {(0,1), 0}

- During the training session of a neural network, an input stimulus is applied that results in an output response (actual output). The response is compared with the desired output signal or the target response.
- If the actual response differs from the target response, the neural network generates an error signal, which is then used to calculate the adjustment that should be made to network synaptic weights so that the actual output matches the target output. In other words, the error is minimized, possibly to zero.



- The error minimization process requires a special circuit known as a teacher or supervisor, hence the name supervised learning or learning with a teacher.

- **Tasks:**
  - classification
  - pattern recognition
  - regression
- **Supervised learning systems**
  - Perceptrons
  - Adaline
  - Backpropagation
  - support vector machines
  - Radial Basis Function networks

Supervised Learning:
- Input applied, output response generated
- Output response compared to target response.
- Error between target and actual response is used to update synaptic weight.
- Aim: error is minimized (possibly to zero)
- Error minimization requires teacher (special circuit) → controller

Reinforcement learning:

- Reinforcement learning is similar to supervised learning, except that, in stead of being provides with the correct (Target) output for each network input, it only indicates if the computed output is correct or incorrect.
- The teacher does not present the target output to the network, but present only a "pass (1)/fail (0)" indication. Thus, the error signal generated during the training session is binary: "pass or fail" OR "Correct or incorrect" OR "0 or 1".

- Output response generated
- Teacher indicates whether output and target match.
- No match ⇒ adjust network parameters
- Boundaries needed to prevent infinite search

- If the teacher indication is "bad/fail" the network readjusts its parameters and tries again and again until it gets its output response right. But, there is no indication if the output response is moving in the right direction or how close to the correct response it is.

- Reinforcement learning is not one of the popular forms of learning.
- It is most suitable for control system application.

Unsupervised Learning:

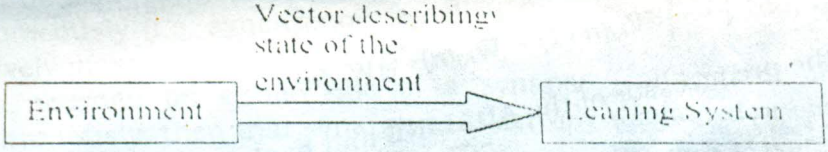
- In Unsupervised Learning or self-organized learning, there is no external teacher to oversee the learning process. That is, no target outputs are available. This is why it is also called learning without a teacher.

At a first glance this might seem to be impractical. How can you train a network if you don't know what is supposed to do?

- Most of these algorithms perform some kind of clustering operation. They learn to categorize the input pattern in to a finite number of classes.

- During the training session, the neural net receives at its input different excitations, or input pattern, and it arbitrary organizes the pattern into categories or classes.

- When stimulus (new input pattern) is later applied, the neural net provides an output response indicating the class to which the stimulus belongs. If a class cannot be found for the input stimulus, a new class is generated.



- Even though Unsupervised Learning does not require a teacher, it requires guidelines to determine how it will form group. Grouping may be based on shape, color, or material consistency or on some other property of the object.
- **Example:** vegetables (lalshak, poishak, patshak), fish(Roi, katal, clis), meat (beef, mutton, chicken)
- **Task:**
  1. clustering
  2. content addressable memory

Unsupervised learning systems:

1. Hebbian learning
2. Kohonen's Self-organising Feature Maps
3. Associative learning
4. Competitive learning
5. Adaptive Resonance Theory (ART).

Learning rate

A training parameter that controls the size of weights and bias changes during learning.

Error Correction Learning

Widrow-Hoff learning rule:

- It is also known as Delta rule or Least Mean Square Rule.
- A learning rule used to trained single-layer linear networks. This rule is based on the idea of continuous adjustment of the value of the weights such that the difference of error (delta) between the desired (or target) output values and the actual output value of a neural network is reduced.
- An error signal, denoted by $e_k(n)$ is given by:

$$e_k(n)=d_k(n)-y_k(n)$$

Where,

- $d_k(n)$—Desired or target output.
- $y_k(n)$—Actual Output.

- The adjustment applied to the synaptic weight W_{kj} at time step n is expressed in the general form:

$$\Delta W_{kj}(n)=\eta . e_k(n)\, x_j(n))$$

Where, η = Learning rate parameter. $\eta < 1$

- Having computed the synaptic adjustment $\Delta W_{kj}(n)$, the updated value of synaptic weight $W_{kj}(n)$ is determined by:

$$W_{kj}(n+1) = W_{kj}(n)+ \Delta W_{kj}(n)$$

- This rule is the predecessor of the **ADALINE** and **backpropagation** rule.

Gradient descent

- The process of making changes to weights and biases, where the changes are proportional to the derivatives of network error with respect to those weights and biases.

Suppose we have a function $f(x)$ and we want to change the value of x to minimize $f(x)$. What we need to do depends on the derivative of $f(x)$. There are three cases to consider: This is done to minimize network error.

If $\frac{df}{dx} > 0$ then $f(x)$ increases as x increases so we should decrease x

If $\frac{df}{dx} < 0$ then $f(x)$ decreases as x increases so we should increase x

If $\frac{df}{dx} = 0$ then $f(x)$ is at a maximum or minimum so we should not change x

In summary, we can decrease $f(x)$ by changing x by the amount:

$$\Delta x = x_{new} - x_{old} = -\eta \frac{\partial f}{\partial x}$$

- An error signal, denoted by $e_k(n)$ is given by:

$$e_k(n) = d_k(n) - y_k(n)$$

- The adjustment applied to the synaptic weight W_{kj} at time step n is expressed in the general form:

$$\Delta w_{k,j}(n) = -\eta \frac{\partial e_k(n)}{\partial w_{k,j}(n)}$$

Hebbian Learning

This rule was proposed by Donald Hebb (1949)

Hebb's postulate:

"When an axon of cell is near enough to excite a cell B and repeatedly or persistently takes part firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased" – Hebb (1949)

- *We may expand and reshape the Hebb's rule as a twopart rule:*

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

ii) If two neurons on either side of a synapse (connection) are activated asynchronously, then that synapse is selectively weakened or eliminated.

Mathematical model of Hbbian modification:

- To formulate Hebbian learning in mathematical terms, consider a synaptic weight W_{kj} of neuron k with presynaptic and postsynaptic signals denoted by X_j and Y_k, respectively.
- The adjustment applied to the synaptic weight W_{kj} at time step n is expressed in the general form:

$$\Delta W_{kj}(n) = F(y_k(n), x_j(n))$$

Where, F(.,.) is a function of both presynaptic and postsynaptic signals.

Hebb's hypothesis:

The simplest form of Hebbian learning is described by:

$$\Delta W_{kj}(n) = \eta\, y_k(n)\, x_j(n))$$

Where, η = Learning rate parameter, $\eta < 1$

Learning rate

A training parameter that controls the size of weights and bias changes during learning.

Error Correction Learning

Widrow-Hoff learning rule:

- It is also known as Delta rule or Least Mean Square Rule.
- A learning rule used to trained single-layer linear networks. This rule is based on the idea of continuous adjustment of the value of the weights such that the difference of error (delta) between the desired (or target) output values and the actual output value of a neural network is reduced.
- An error signal, denoted by $e_k(n)$ is given by:

$$e_k(n) = d_k(n) - y_k(n)$$

Where,

- $d_k(n)$—Desired or target output.
- $y_k(n)$—Actual Output.

- The adjustment applied to the synaptic weight W_{kj} at time step n is expressed in the general form:

$$\Delta W_{kj}(n) = \eta . e_k(n)\, x_j(n)$$

Where, η = Learning rate parameter. $\eta < 1$

- Having computed the synaptic adjustment $\Delta W_{kj}(n)$, the updated value of synaptic weight $W_{kj}(n)$ is determined by:

$$W_{kj}(n+1) = W_{kj}(n) + \Delta W_{kj}(n)$$

- This rule is the predecessor of the **ADALINE** and **backpropagation** rule.

Gradient descent

- The process of making changes to weights and biases, where the changes are proportional to the derivatives of network error with respect to those weights and biases.

Suppose we have a function $f(x)$ and we want to change the value of x to minimize $f(x)$. What we need to do depends on the derivative of $f(x)$. There are three cases to consider: This is done to minimize network error.

If $\frac{df}{dx} > 0$ then $f(x)$ increases as x increases so we should decrease x

If $\frac{df}{dx} < 0$ then $f(x)$ decreases as x increases so we should increase x

If $\frac{df}{dx} = 0$ then $f(x)$ is at a maximum or minimum so we should not change x

In summary, we can decrease $f(x)$ by changing x by the amount:

$$\Delta x = x_{new} - x_{old} = -\eta \frac{\partial f}{\partial x}$$

- An error signal, denoted by $e_k(n)$ is given by:

$$e_k(n) = d_k(n) - y_k(n)$$

- The adjustment applied to the synaptic weight W_{kj} at time step n is expressed in the general form:

$$\Delta w_{k,j}(n) = -\eta \frac{\partial e_k(n)}{\partial w_{k,j}(n)})$$

Hebbian Learning

This rule was proposed by Donald Hebb (1949)

Hebb's postulate:

"When an axon of cell is near enough to excite a cell B and repeatedly or persistently takes part firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased" – Hebb (1949)

- *We may expand and reshape the Hebb's rule as a twopart rule:*

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

ii) If two neurons on either side of a synapse (connection) are activated asynchronously, then that synapse is selectively weakened or eliminated.

Mathematical model of Hbbian modification:

- To formulate Hebbian learning in mathematical terms, consider a synaptic weight W_{kj} of neuron k with presynaptic and postsynaptic signals denoted by X_j and Y_k, respectively.
- The adjustment applied to the synaptic weight W_{kj} at time step n is expressed in the general form:

$$\Delta W_{kj}(n) = F(y_k(n), x_j(n))$$

Where, F(.,.) is a function of both presynaptic and postsynaptic signals.

Hebb's hypothesis:

The simplest form of Hebbian learning is described by:

$$\Delta W_{kj}(n) = \eta\, y_k(n)\, x_j(n))$$

Where, η = Learning rate parameter, $\eta < 1$

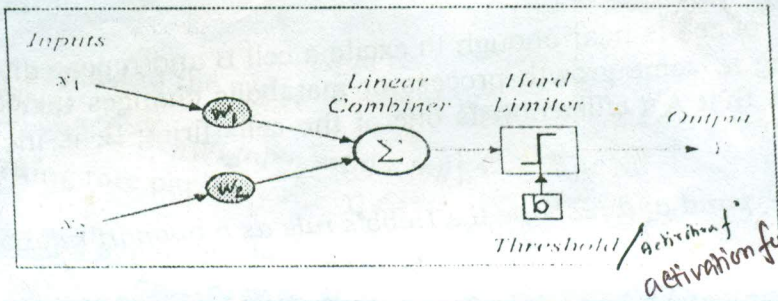
Perceptron:

The perceptron is the simplest form of a neural network used for the classification of pattern said to be *linearly separable* (i.e. patterns that lie on opposite sides of a hyperplane). A single perceptron classifies input patterns, x, into **two classes**.

A perceptron is based on the McCulloch-Pitts neuron. It consists of a single neuron with *adjustable synaptic weights* and *bias*. *They have n inputs and 1 output.*

In 1958-1960, **Frank Rosenblatt** introduced a training algorithm that provided the first procedure for training a simple ANN: a **perceptron**.

The operation of Rosenblatt's perceptron is based on the **McCulloch and Pitt's neuron model**. The model consists of a linear combiner followed by a hard limiter. The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to +1 if its input is positive and −1 if it is negative.



Single-layer two-input perceptron

How does the perceptron learn its classification tasks?

The perceptron learning rule was first proposed by **Rosenblatt** in 1960. Using this rule we can derive the perceptron training algorithm for classification tasks.

This is done by making small adjustments in the weights to reduce the difference between the actual and desired outputs of the perceptron. The initial weights are randomly assigned, usually in the range [−1, 1], and then updated to obtain the output consistent with the training examples.

If at iteration n, the actual output is Y(n) and the desired output is d (n), then the error is given by:

$$e(n) = d(n) - Y(n) \quad \text{where } n = 1, 2, 3, \ldots$$

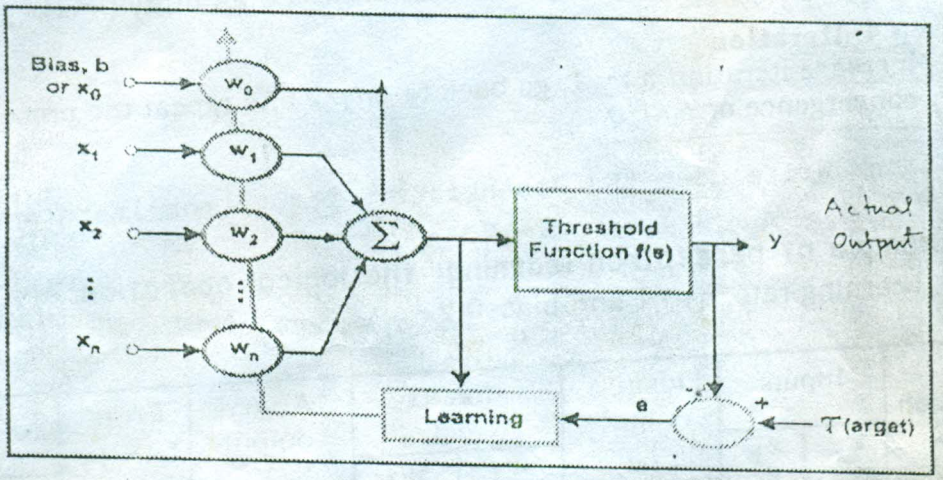
Iteration n here refers to the nth training example presented to the perceptron.

If the error, $e(n)$ is positive, we need to increase perceptron output $Y(p)$, but if it is negative, we need to decrease $Y(p)$.

$$w(n+1)=w(n)+\eta.[d(n) - Y(n)].x(n)$$

$$w(n+1)=w(n)+\eta.e(n).x(n) \qquad \text{where, } n = 1, 2, 3, \ldots .$$

η is the **learning rate**, a positive constant less than unity ($\eta<1$).



Perception's training algorithm

Step 1: Initialisation

Set initial weights $w1$, $w2$,..., wn and threshold or bias b_k to random numbers in the range [-1, 1].

If the error, e(n), is positive, we need to increase perceptron output $Y(n)$, but if it is negative, we need to decrease $Y(n)$.

Step 2: Activation

Activate the perceptron by applying inputs $x1(n)$, $x2(n)$,..., $x_m(n)$ and desired output $d(n)$.

Calculate the actual output at iteration n = 1

$$Y(n) = hard\lim\left(\sum_{i=1}^{m} x_i(n).w_i(n) + b_k\right)$$

where, m is the number of the perceptron inputs, and *hardlim* is the hard limiter activation function

Step 3: Weight training

Update the weights of the perceptron

$$W_i(n+1)=w_i(n)+\Delta w_i(n) \quad \text{---(i)}$$

Where, $\Delta w_i(n)$ is the weight correction at iteration n. The weight correction is computed by the delta rule:

$$\Delta w_i(n) = \eta \,.\, [d_i(n) - Y_i(n)] \,.\, x_i(n) \quad \text{---(ii)}$$

$$\Delta w_i(n) = \eta . e_i(n) . x_i(n)$$

η is the **learning rate**, a positive constant less than unity ($\eta<1$).

Step 4: Iteration

Increase iteration n by 1, go back to *Step 2* and repeat the process until convergence or $e(n)=0$

Question: Write down the Rosenblatt's perceptron learning algorithm

Example of perceptron learning: the logical operation AND

Here, Learning rate, η=0.1 and bias=0.2 , $b_K = -0.2$

Epoch n	Inputs x_1	Inputs x_2	Desired output Y_d	Initial weights w_1	Initial weights w_2	Actual output Y	Error $Y_d - y = e$	Final weights w_1	Final weights w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	−1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	−1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	−1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

$Y = \text{hardlim}\left([w_1\ w_2] * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b_K\right)$

Final weight $= [w_1'\ w_2'] = [w_1\ w_2] + \eta . e . [x_1\ x_2]$

N.B. – Calculation [illegible]