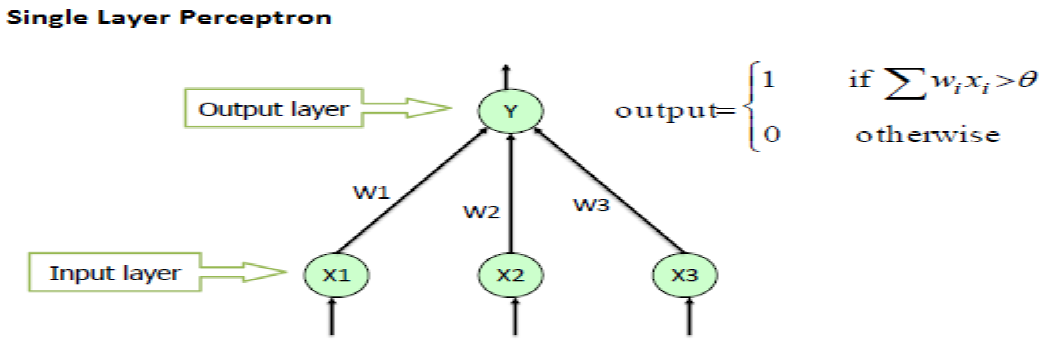
Unit-2

SLP & MLP

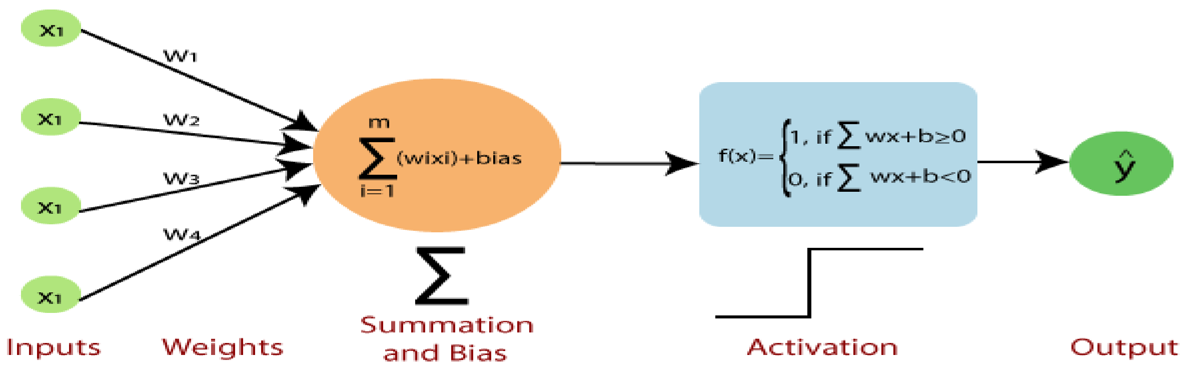
A single layer perceptron

A perceptron is a simple type of neural network that can learn to classify linearly separable patterns. It consists of a single layer of weighted inputs and a binary output. A multi-layer perceptron (MLP) is a more complex type of neural network that can learn to classify non-linearly separable patterns. It consists of multiple layers of perceptrons, each with its own weights and activation function. In this article, you will learn about the advantages and disadvantages of using a single-layer perceptron versus a multi-layer perceptron for different tasks and scenarios.

A single layer perceptron (**SLP**) is a feed-forward network based on a threshold transfer function. SLP is the simplest type of artificial neural networks and can only classify linearly separable cases with a binary target (1 , 0).



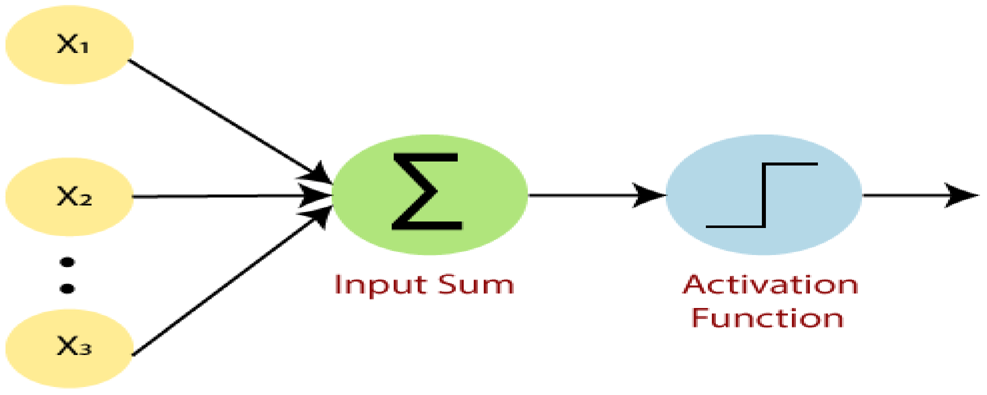
$$\text{output} = \begin{cases} 1 & \text{if } \sum w_i x_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

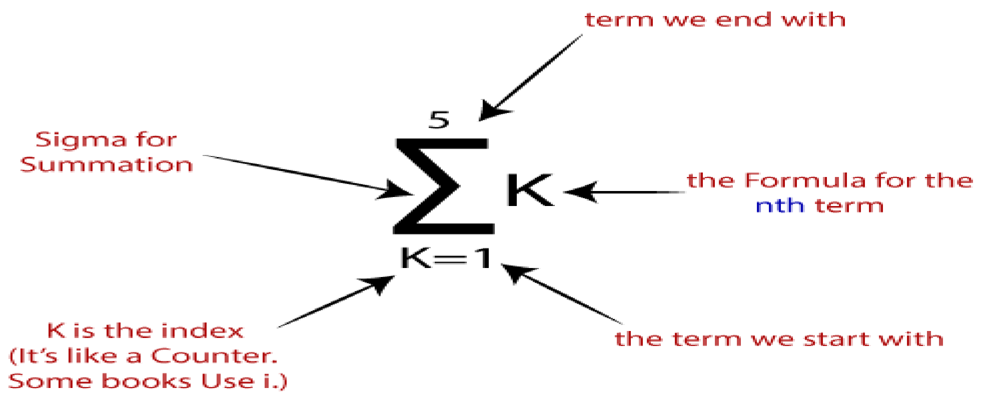


X1
X1
X1
X1
W1
W2
W3
W4
$\sum_{i=1}^{m}$ (wixi)+bias
$f(x)=\begin{cases}1, \text{ if } \sum wx+b \geq 0 \\ 0, \text{ if } \sum wx+b<0\end{cases}$
ŷ
Inputs
Weights
Summation and Bias
Activation
Output

The perceptron consists of 4 parts.

- **Input value or One input layer:** The input layer of the perceptron is made of artificial input neurons and takes the initial data into the system for further processing.
- **Weights and Bias:**
  Weight: It represents the dimension or strength of the connection between units. If the weight to node 1 to node 2 has a higher quantity, then neuron 1 has a more considerable influence on the neuron.
  Bias: It is the same as the intercept added in a linear equation. It is an additional parameter which task is to modify the output along with the weighted sum of the input to the other neuron.
- **Net sum:** It calculates the total sum.
- **Activation Function:** A neuron can be activated or not, is determined by an activation function. The activation function calculates a weighted sum and further adding bias with it to give the result.




term we end with
Sigma for Summation
$\sum_{K=1}^{5} K$
the Formula for the nth term
K is the index (It's like a Counter. Some books Use i.)
the term we start with

There are two types of architecture. These types focus on the functionality of artificial neural networks as follows-

- Single Layer Perceptron
- Multi-Layer Perceptron

Single Layer Perceptron

The single-layer perceptron was the first neural network model, proposed in 1958 by Frank Rosenbluth. It is one of the earliest models for learning. Our goal is to find a linear decision function measured by the weight vector w and the bias parameter b.

To understand the perceptron layer, it is necessary to comprehend artificial neural networks (ANNs).

The artificial neural network (ANN) is an information processing system, whose mechanism is inspired by the functionality of biological neural circuits. An artificial neural network consists of several processing units that are interconnected.

This is the first proposal when the neural model is built. The content of the neuron's local memory contains a vector of weight.

Single-layer perceptron advantages

One of the main advantages of using a single-layer perceptron is its simplicity and efficiency. It is easy to implement, train, and understand. It has a clear geometric interpretation as a hyperplane that separates two classes of data. It can also perform well on problems that are linearly separable, such as logical operations, linear regression, and binary classification.

2Single-layer perceptron disadvantages

One of the main disadvantages of using a single-layer perceptron is its limited expressive power and generalization ability. It cannot learn to classify non-linearly separable patterns, such as XOR, circles, or spirals. It is also prone to overfitting and noise, as it tries to fit a straight line to the data. It does not have any hidden layers or activation functions that can introduce non-linearity and flexibility to the model.

- **Multi-Layer Perceptron**
- An MLP is a type of feedforward artificial neural network with multiple layers, including an input layer, one or more hidden layers, and an output layer. Each layer is fully connected to the next. In this article, we will understand MultiLayer Perceptron Neural Network, an important concept of deep learning and neural networks.

A multilayer perceptron (MLP) Neural network belongs to the feedforward neural network. It is an Artificial Neural Network in which all nodes are interconnected with nodes of different layers.

Frank Rosenblatt first defined the word Perceptron in his perceptron program. Perceptron is a basic unit of an artificial neural network that defines the artificial neuron in the neural network. It is a supervised learning algorithm containing nodes' values, activation functions, inputs, and weights to calculate the output.

The Multilayer Perceptron (MLP) Neural Network works only in the forward direction. All nodes are fully connected to the network. Each node passes its value to the coming node only in the forward direction. The MLP neural network uses a Backpropagation algorithm to increase the accuracy of the training model.

Structure of MultiLayer Perceptron Neural Network

This network has three main layers that combine to form a complete Artificial Neural Network. These layers are as follows:

Input Layer

It is the initial or starting layer of the Multilayer perceptron. It takes input from the training data set and forwards it to the hidden layer. There are n input nodes in the input layer. The number of input nodes depends on the number of dataset features. Each input vector variable is distributed to each of the nodes of the hidden layer.

Hidden Layer

It is the heart of all Artificial neural networks. This layer comprises all computations of the neural network. The edges of the hidden layer have weights multiplied by the node values. This layer uses the activation function.

There can be one or two hidden layers in the model.

Several hidden layer nodes should be accurate as few nodes in the hidden layer make the model unable to work efficiently with complex data. More nodes will result in an overfitting problem.

Output Layer

This layer gives the estimated output of the Neural Network. The number of nodes in the output layer depends on the type of problem. For a single targeted variable, use one node. N classification problem, ANN uses N nodes in the output layer.

Working of MultiLayer Perceptron Neural Network

- The input node represents the feature of the dataset.
- Each input node passes the vector input value to the hidden layer.
- In the hidden layer, each edge has some weight multiplied by the input variable. All the production values from the hidden nodes are summed together. To generate the output
- The activation function is used in the hidden layer to identify the active nodes.
- The output is passed to the output layer.
- Calculate the difference between predicted and actual output at the output layer.
- The model uses backpropagation after calculating the predicted output.

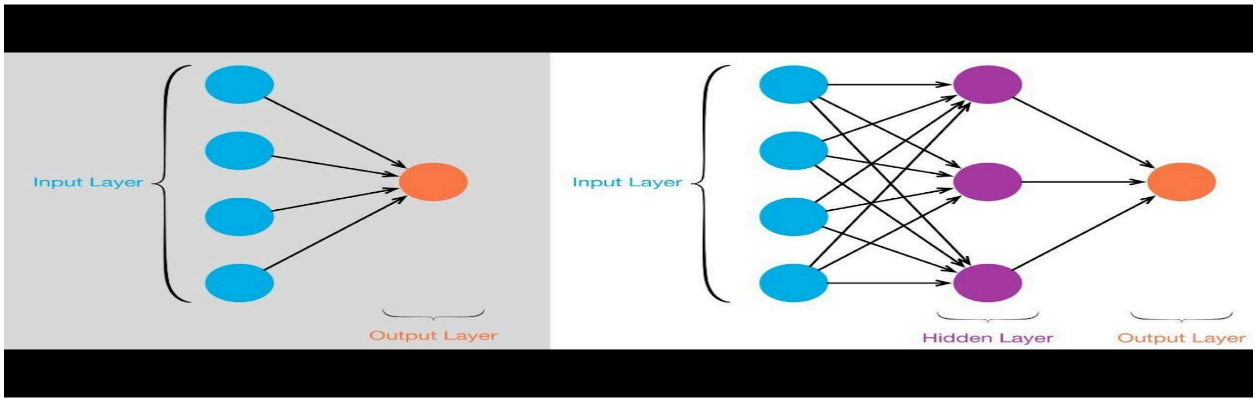
Advantages of MultiLayer Perceptron Neural Network

1. MultiLayer Perceptron Neural Networks can easily work with non-linear problems.
2. It can handle complex problems while dealing with large datasets.
3. Developers use this model to deal with the fitness problem of Neural Networks.
4. It has a higher accuracy rate and reduces prediction error by using backpropagation.

5. After training the model, the Multilayer Perceptron Neural Network quickly predicts the output.

Disadvantages of MultiLayer Perceptron Neural Network

1. This Neural Network consists of large computation, which sometimes increases the overall cost of the model.
2. The model will perform well only when it is trained perfectly.
3. Due to this model's tight connections, the number of parameters and node redundancy increases.



Difference between single layer perceptron and multi layer perceptron

Adaptive Filtering Problem

Adaptive filtering is of central importance in many applications of signal processing, such as the modelling, estimation and detection of signals. Adaptive filters also play a crucial role in system modelling and control. These applications are related to

communications, radar, sonar, biomedical
electronics, geophysics, etc.
A general discrete-time filter defines a relationship between an input time sequence {u(n), u(n–1),
…} and an output time sequence {y(n), y(n–1), …}, u(n) and y(n) being either uni or
multidimensional signals. In the following, we consider filters having one input and one output. The
generalization to multidimensional signals is straightforward.
There are two types of filters: (i) transversal filters (termed Finite Impulse Response or FIR filters in

linear filtering) whose outputs are functions of the input signals only; and (ii) recursive filters (termed Infinite Impulse Response or IIR filters in linear filtering) whose outputs are functions both of the input signals and of a delayed version of the output signals. Hence, a transversal filter is defined by:

y(n) =鍶[u(n), u(n-1), ..., u(n-M+1)], (1)

where M is the length of the finite memory of the filter, and a recursive filter is defined by

y(n) =鍶[u(n), u(n-1), ..., u(n-M+1), y(n-1), y(n-2), ..., y(n-N)] (2)

where N is the order of the filter.

The ability of a filter to perform the desired task is expressed by a criterion; this criterion may be either quantitative, e.g., maximizing the signal to noise ratio for spatial filtering [see for instance Applebaum and Chapman 1976], minimizing the bit error rate in data transmission [see for instance Proakis 1983], or qualitative, e.g. listening for speech prediction [see for instance Jayant and Noll 1984]. In practice, the criterion is usually expressed as a weighted sum of squared differences between the output of the filter and the desired output (e.g. LS criterion).

An adaptive filter is a system whose parameters are continually updated, without explicit control by
the user. The interest in adaptive filters stems from two facts: (i) tailoring a filter of given architecture to perform a specific task requires a priori knowledge of the characteristics of the input
signal; since this knowledge may be absent or partial, systems which can learn the characteristics of
the signal are desirable; (ii) filtering nonstationary signals necessitates systems which are capable of tracking the variations of the characteristics of the signal.

The bulk of adaptive filtering theory is devoted to linear adaptive filters, defined by relations (1) and
(2), where 鍉is a linear function. Linear filters have been extensively studied, and are appropriate for
many purposes in signal processing. A family of particularly efficient adaptation algorithms has bee
Adaptive filtering is of central importance in many applications of signal processing, such as the modelling, estimation and detection of signals. Adaptive filters also play a crucial role in system
modelling and control. These applications are related to communications, radar, sonar, biomedical

electronics, geophysics, etc.
A general discrete-time filter defines a relationship between an input time sequence {u(n), u(n–1),
…} and an output time sequence {y(n), y(n–1), …}, u(n) and y(n) being either uni or
multidimensional signals. In the following, we consider filters having one input and one output. The
generalization to multidimensional signals is straightforward.
There are two types of filters: (i) transversal filters (termed Finite Impulse Response or FIR filters in
linear filtering) whose outputs are functions of the input signals only; and (ii) recursive filters

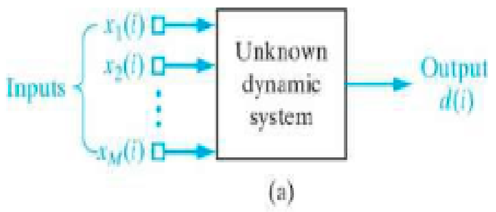
(termed Infinite Impulse Response or IIR filters in linear filtering) whose outputs are functions both
of the input signals and of a delayed version of the output signals. Hence, a transversal filter is
defined by:
y(n) =鎚[u(n), u(n-1), ..., u(n-M+1)], (1)
where M is the length of the finite memory of the filter, and a recursive filter is defined by
y(n) =鎚[u(n), u(n-1), ..., u(n-M+1),
y(n-1), y(n-2), ..., y(n-N)] (2)
where N is the order of the filter.
The ability of a filter to perform the desired task is expressed by a criterion; this criterion may be

either quantitative, e.g., maximizing the signal to noise ratio for spatial filtering [see for instance
Applebaum and Chapman 1976], minimizing the bit error rate in data transmission [see for instance
Proakis 1983], or qualitative, e.g. listening for speech prediction [see for instance Jayant and Noll
1984]. In practice, the criterion is usually expressed as a weighted sum of squared differences
between the output of the filter and the desired output (e.g. LS criterion).
An adaptive filter is a system whose parameters are continually updated, without explicit control by

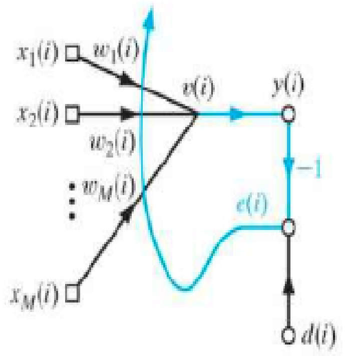
the user. The interest in adaptive filters stems from two facts: (i) tailoring a filter of given architecture to perform a specific task requires a priori knowledge of the characteristics of the input signal; since this knowledge may be absent or partial, systems which can learn the characteristics of the signal are desirable; (ii) filtering nonstationary signals necessitates systems which are capable of tracking the variations of the characteristics of the signal.
The bulk of adaptive filtering theory is devoted to linear adaptive filters, defined by relations (1) and

(2), where 鍉is a linear function. Linear filters have been extensively studied, and are appropriate for
many purposes in signal processing. A family of particularly efficient adaptation algorithms has bee

Consider a dynamical system, the mathematical characterization of which is unknown. All that we have available on the system is a set of labeled input-output data generated by the system at discrete instants of time at some uniform rate. Specifically, when an m-dimensional stimulus x(i) is applied across m input nodes of the system, the system responds by producing a scalar output d(i), where i = 1 , 2, ... n, . . . as depicted in Fig. a. Thus, the external behavior of the system is described by the data set T:{$x(i)$,$d(i)$;i=1,2,…,n…} where $x(i) = [x_1(i), x_2(i),\dots, x_m(i)]^T$,m is the input dimensionality.



(a)



(b)

The stimulus vector x(i) can arise in either way of the following:

Spatial: The m input elements of x originate at different point in space, Snapshot data

Temporal data, x(i) is uniformly spaced in time, The m input elements of x represent the set of present and (m-1) past values of some excitation.

To design a multiple input-single output model of the unknown dynamical system, it is by building it around a single linear neuron.The neuronal model operates under the influence of an algorithm that controls necessary adjustments to the synaptic weights of the neuron.

With the following points in mind:

To design a multiple input-single output model of the unknown dynamical system, it is by building it around a single linear neuron.The neuronal model operates under the influence of an algorithm that controls necessary adjustments to the synaptic weights of the neuron.
With the following points in mind:

To design a multiple input-single output model of the unknown dynamical system, it is by building it around a single linear neuron.The neuronal model operates under the influence of an algorithm that controls necessary adjustments to the synaptic weights of the neuron.

With the following points in mind:

To design a multiple
input-single output model of
the unknown dynamical
system, it is by
building it around a single
linear neuron.The neuronal
model operates under the
influence
of an algorithm that controls
necessary adjustments to
the synaptic weights of the
neuron.
With the following points in
mind:

To design a multiple
input-single output model of
the unknown dynamical
system, it is by
building it around a single
linear neuron.The neuronal
model operates under the
influence
of an algorithm that controls
necessary adjustments to
the synaptic weights of the
neuron.
With the following points in
mind:

To design a multiple
input-single output model of
the unknown dynamical
system, it is by
building it around a single
linear neuron.The neuronal
model operates under the
influence
of an algorithm that controls
necessary adjustments to
the synaptic weights of the
neuron.
With the following points in
mind:

To design a multiple input-single output model of the unknown dynamical system, it is by building it around a single linear neuron.The neuronal model operates under the influence of an algorithm that controls necessary adjustments to the synaptic weights of the neuron.

With the following points in mind:

To design a multiple input-single output model of the unknown dynamical system, it is by building it around a single linear neuron. The neuronal model operates under the influence of an algorithm that controls necessary adjustments to the synaptic weights of the neuron. With the following points in mind:

- Start from an arbitrary setting of the adjustable weights.
- Adjustment of weights are made on continuous basis.
- Computation of adjustments to the weight are completed inside one interval that is one sampling period long.

The neuronal model described is referred to as an adaptive filter.

Figure b shows a signal-flow graph of the adaptive filter. Its operation consists of two continuous processes:

- Filtering process: computation of the output signal y(i) and the error signal e(i).
- Adaptive process: The automatic adjustment of the weights according to the error signal.

Thus, the combination of these two processes working together constitutes a feedback loop acting around the neuron. Since the neuron is linear,

$y(i) = v(i) = \sum_{k=1}^{m} w_k(i)x_k(i)$,

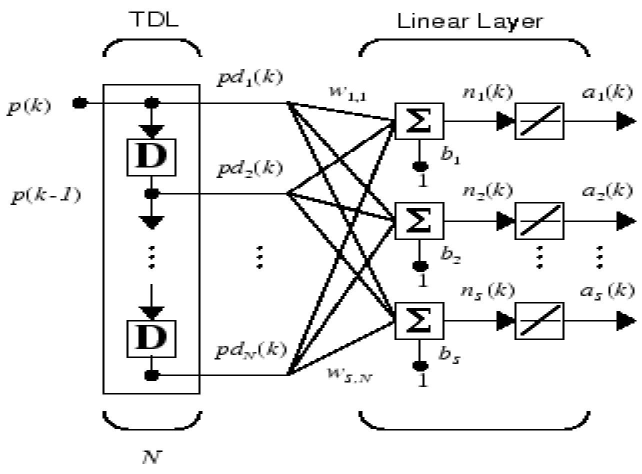
In matrix form: $y(i) = X^T(i)W(i)$

Where, $W(i) = [w_1(i), w_2(i),..., w_m(i)]$ and

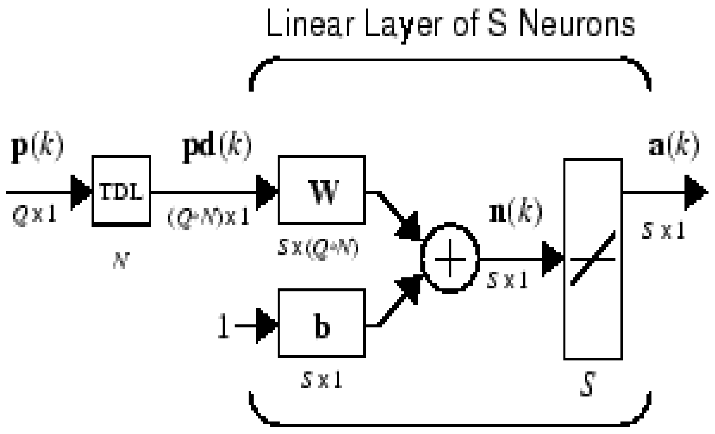
Error, e(i) = d(i) – y(i).

Multiple Neuron Adaptive Filters

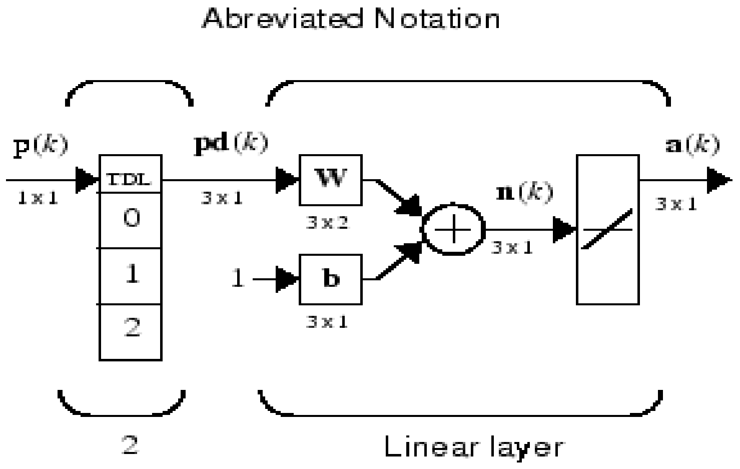
You might want to use more than one neuron in an adaptive system, so you need some additional notation. You can use a tapped delay line with *S* linear neurons, as shown in the next figure.



Alternatively, you can represent this same network in abbreviated form.



If you want to show more of the detail of the tapped delay line—and there are not too many delays—you can use the following notation:



Here, a tapped delay line sends to the weight matrix:

- The current signal
- The previous signal
- The signal delayed before that

You could have a longer list, and some delay values could be omitted if desired. The only requirement is that the delays must appears in increasing order as they go from top to bottom.

Unconstrained optimization technique

Consider the cost function E (w) that is continuously differentiable function of unknown weights w. The function E(w) maps the elements of w into real numbers. It is a measure of how to choose the weight (parameter) vector w of an adaptive filtering algorithm so that it behaves in an optimum manner. We want to find an optimal solution w* that satisfies the condition:

$E(w^*) \leq E(w) \Rightarrow$ Minimize E(w) with respect to w.

That is, we need to solve an unconstrained optimization problem, stated as follows:

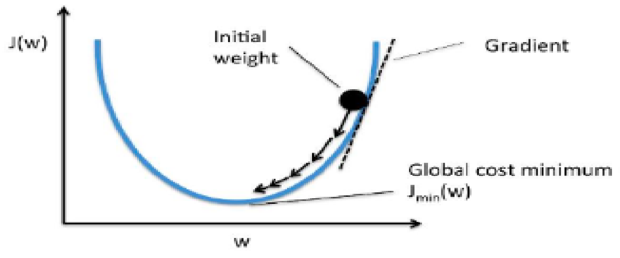
The necessary condition for optimality:

$\nabla E(w^*) = 0 \; \nabla = [\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \dots \frac{\partial}{\partial w_m}]^T \Rightarrow \nabla E(w) = [\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots \frac{\partial E}{\partial w_m}]^T$

A class of unconstrained optimization algorithms that is particularly well suited for the design of adaptive filters is based on the idea of local iterative descent:

Starting with an initial guess denoted by w(O), generate a sequence of weight vectors w(l), w(2), . . ., such that the cost function E(w) is reduced at each iteration of the algorithm, as shown by **E(w(n + 1)) < E(w(n))**, where w(n) is the old value of the weight vector and w(n + 1) is its updated value.

Method of Steepest Descent



The successive adjustments applied to the weight vector w are in the direction of steepest descent; that is in a direction opposite to the gradient of E(W).(In fig. E(W) is considered as J (w))

If g = ∇ E(w) , then the steepest descent algorithm is

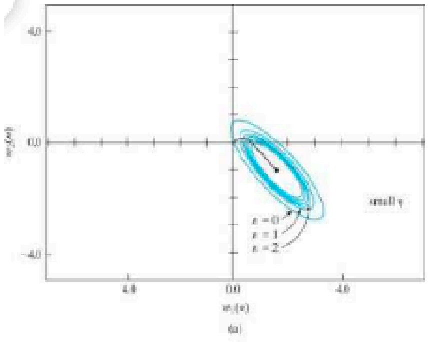
$$w(n+1) = w(n) - \eta g(n)$$

η is a positive constant called the *step size*, or *learning rate* parameter. In each step, the algorithm applies the correction:

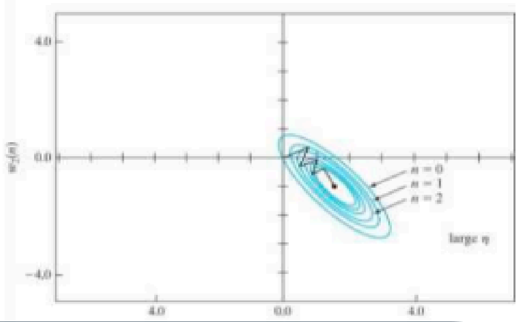
$$\Delta w(n) = w(n+1) - w(n)$$

$$= -\eta g(n)$$

The method of steepest descent converges to the optimal solution w* slowly. Moreover, the learning-rate parameter η has a profound influence on its convergence behaviour:



- When η is small, the transient response of the algorithm is overdamped, i.e., w(n) follows smooth path in the W-plane, as illustrated in Fig. a
- When η is large, the transient response of the algorithm is underdamped, i.e., w(n) follows zig zaging (oscillatory) path, as illustrated in Fig. b.



- When η exceeds critical value, the algorithm becomes unstable(diverges).

Newton method

The basic idea of Newton 's method is to minimize the quadratic approximation of the cost function E(w) around the current point w(n); this minimization is performed at each iteration of the algorithm. Specifically, using a second-order Taylor series expansion of the cost function around the point w(n), we may write

$$E\big(w(n) + \Delta w(n)\big) = E\big(w(n)\big) + E'\big(w(n)\big)\Delta w(n) + \frac{1}{2}\Delta w^T(n) E''\big(w(n)\big)\Delta w(n) + \cdots$$

Note: Taylor series expansion $f(\mathbf{x}_k + \Delta\mathbf{x}) \approx f(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T Q_k \Delta\mathbf{x}_k$

,where $\mathbf{g}_k = \mathbf{g}(\mathbf{x}_k) = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}$

Gradient of cost function i.e., diff. w.r.to $\Delta w(n)$

- $\nabla E\big(w(n) + \Delta w(n)\big) = E'\big(w(n)\big) + \frac{1}{2}.2.H\big(w(n)\big)\Delta w(n) = 0$
- $E'\big(w(n)\big) = g(n)$ =1st derivative

$$\mathbf{H} = \nabla^2 \mathscr{E}(\mathbf{w}) = \begin{bmatrix} \frac{\partial^2 \mathscr{E}}{\partial w_1^2} & \frac{\partial^2 \mathscr{E}}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 \mathscr{E}}{\partial w_1 \partial w_m} \\ \frac{\partial^2 \mathscr{E}}{\partial w_2 \partial w_1} & \frac{\partial^2 \mathscr{E}}{\partial w_2^2} & \cdots & \frac{\partial^2 \mathscr{E}}{\partial w_2 \partial w_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 \mathscr{E}}{\partial w_m \partial w_1} & \frac{\partial^2 \mathscr{E}}{\partial w_m \partial w_2} & \cdots & \frac{\partial^2 \mathscr{E}}{\partial w_m^2} \end{bmatrix}$$

$$\nabla E(w(n) + \Delta w(n)) = E'(w(n)) + \frac{1}{2}.2.H(w(n))\Delta w(n) = 0$$

$$H(w(n))\Delta w(n) = -E'(w(n))$$

$$\Delta w(n) = -H(w(n))^{-1} E'(w(n))$$

$$w(n+1) = w(n) - \eta H(w(n))^{-1} E'(w(n))$$

$$w(n+1) = w(n) - \eta H(w(n))^{-1} E'(w(n))$$

Where $H^{-1} = inverse\ of\ Hessian\ matrix$

Gauss-Newton method

The Gauss-Newton method is applicable to a cost function that is expressed as the sum of error squares. Let $E(w) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$, where the scaling factor 1/2 is included to simplify matters in subsequent analysis. All the error terms in this formula are calculated on the basis of a weight vector w that is fixed over the entire observation interval $1 \leq i \leq n$

e(i) is function of w, which is fixed over observation interval $1 \leq i \leq n$. Given an operating point w(n), linearize the dependence of e(i) on w as

$$e'(i, w) = e(i) + [\frac{\partial e(i)}{\partial w}]^T (w - w(n)),\ i = 1,2,\ldots,n\ \ at\ w = w(n)$$

In matrix form

$$e'(n, w) = e(n) + J(n)(w - w(n))$$

where, e(n) is the error vector , e(n) =[e(1),e(2),...,e(n)]^T and J(n) is the Jacobian matrix, which is defined as $\longrightarrow$

$$\mathbf{J}(n) = \begin{bmatrix} \frac{\partial e(1)}{\partial w_1} & \frac{\partial e(1)}{\partial w_2} & \cdots & \frac{\partial e(1)}{\partial w_m} \\ \frac{\partial e(2)}{\partial w_1} & \frac{\partial e(2)}{\partial w_2} & \cdots & \frac{\partial e(2)}{\partial w_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e(n)}{\partial w_1} & \frac{\partial e(n)}{\partial w_2} & \cdots & \frac{\partial e(n)}{\partial w_m} \end{bmatrix}_{\mathbf{w}=\mathbf{w}(n)}$$

The gradient of error vector

$\nabla e(n) = [\nabla e(1), \nabla e(2), \dots, \nabla e(n)]$ m x n

But, J(n) is the Jacobian matrix of dimension n x m, therefore

J(n) = $\nabla e(n)^T$

The updated weight w(n+1) is defined by

$$w(n+1) = arg \min_{w} \left\{ \frac{1}{2} \|e'(n,w)\|^2 \right\}$$

The squared Euclidean norm of error vector

$$\frac{1}{2}\|e'(n,w)\|^2 = \frac{1}{2}\|e\ (n)\|^2 + e^T(n)J(n)(w - w(n)) + \frac{1}{2}(w - w(n))^T J^T(n)J(n)(w - w(n))$$

Diff. this eq. with w and setting it to zero, we get $J^T(n)e(n) + J^T(n)J(n)(w - w(n)) = 0$

which describes the pure form of the Gauss-Newton method. Unlike Newton's method that requires knowledge of the Hessian matrix of the cost function E(n), the Gauss-Newton method only requires the Jacobian matrix of the error vector e(n). However, for the Gauss-Newton iteration to be computable, the matrix product $J^T(n)J(n)$ must be nonsingular.

$J^T(n)J(n)$ matrix has to be nonsingular, which is *not guaranteed* always. This is solved by adding δI to $J^T(n)J(n)$ matrix . Where δ is a positive constant, added to ensure

$J^T(n)J(n) + \delta I$ to be positive definite for all n. Therefore, the modified weight update equation Gauss-Newton Method as

$$w(n+1) = w(n) - \left(J^T(n)J(n) + \delta I\right)^{-1} J^T(n)e(n)$$

LINEAR LEAST-SQUARES FILTER

As the name implies, a linear least-squares filter has two distinctive characteristics. First, the single neuron around which it is built is linear, Second, the cost function E(w) used to design the filter consists of the sum of error squares. We may express the error vector e(n) as follows: e(n) = d(n) – $[X(1), X(2), \ldots, X(n)]^T$w(n) = d(n) –$X^T$(n)w(n) -------------------- (1)

Where, d(n) = $[d(1),d(2), \ldots ,d(n)]^T$ n x 1 desired response vector

X(n) = [X(1), X(2), … ,X(n)] m x n data matrix w(n) – m x 1 weight matrix.

Case 1: assuming error e = 0 in equation (1)

0 = d(n) – $[X(1), X(2), \ldots ,X(n)]^T$w(n)

=> d(n) = X^T(n)w(n) -------------------- (2)

Multiply with X on both sides

=> $X(n)d(n) = X(n)X^T(n)w(n)$

- ⇒ $w(n) = (X(n)X^T(n))^{-1} X(n)d(n)$
- ⇒ For inverse to exist, $|X(n)X^T(n)|$ is non zero and hence rank of this matrix must be full

Case 2: However, error is not equal to zero always, $e \neq 0$, In such cases, we have to define the cost function as squared error

- ⇒ $E = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$
- ⇒ In matrix form,
- ⇒ $E = \frac{1}{2}e(n)e^T(n)$

- ⇒ Where, $e(n) = [e(1)\ e(2) \ldots e(n)]$

In matrix form,

- ⇒ $E = \frac{1}{2}e(n)e^T(n) = \frac{1}{2}(d(n) - X^T(n)w(n))(d(n) - X^T(n)w(n))^T$
- ⇒ $= \frac{1}{2}(d - X^T w)(d - X^T w)^T$ (neglecting time steps)
- ⇒ $= \frac{1}{2}(dd^T - 2dX^T w + w^T X X^T w)$
- ⇒ To find an optimal weight, diff. the cost function and equate that to zero
- ⇒ $\nabla E = -Xd^T + XX^T w = 0$
- ⇒ $W = (XX^T)^{-1} Xd^T$ => Linear least squares solution

If XX^T is singular, i.e., rank of XX^T is equal to m, then it is customary practice to add diagonal matrix I , which results in

⇒ W = $(XX^T + \delta I)^{-1} Xd^T$---------------------(3)

⇒ Where δ is a positive constant and I is the identity matrix

⇒ Eq(3) is the solution to the cost function

⇒ $E = \frac{1}{2}e(n)e^T(n) + \frac{\delta}{2}\|w^T w\|$

WKT, e(n) = d(n) –X^T(n)w(n)

⇒ $\nabla e(n) = -X(n)$ But Jacobian, J(n) = $\nabla e(n)^T = -X(n)^T$

⇒ W = -$(J^T J)^{-1} J^T$ d

Limitations:

When data size is large, computing inverse matrix would be difficult. In such cases, iterative approach will be better to use.

WKT, e(n) = d(n) –X^T(n)w(n)

$$\nabla e(n) = -X(n)$$

But Jacobian, J(n) = $\nabla e(n)^T = -X(n)^T$, Wkt, by Gauss Newton method

$$w(n+1) = w(n) - (J^T(n)J(n))^{-1} J^T(n)e(n)$$

$$= w(n) - ((X(n)X^T(n))^{-1} X(n))(d(n) - X^T(n)w(n))$$

$= (X(n)X^T(n))^{-1} X(n)$d(n)

$w(n+1) = (X(n)X^T(n))^{-1} X(n)$d(n)

Let $X^+ = (X(n)X^T(n))^{-1} X(n) = pseudo$ inverse.

$w(n+1) = X^+(n)d(n)$.

This formula represents a convenient way of saying: "The weight vector w(n + 1) solves the linear least-squares problem defined over an observation interval of duration n.”

Least Mean Square Algorithm

The ADALINE (adaptive linear neuron) networks discussed in this topic are similar to the perceptron, but their transfer function is linear rather than hard-limiting. This allows their outputs to take on any value, whereas the perceptron output is limited to either 0 or 1. Both the ADALINE and the perceptron can solve only linearly separable problems

The LMS algorithm adjusts the weights and biases of the ADALINE so as to minimize this mean square error.

Fortunately, the mean square error performance index for the ADALINE network is a quadratic function. Thus, the performance index will either have one global minimum, a weak minimum, or no minimum, depending on the characteristics of the input vectors. Specifically, the characteristics of the input vectors determine whether or not a unique solution exists.

LMS Algorithm (learnwh)

Adaptive networks will use the LMS algorithm or Widrow-Hoff learning algorithm based on an approximate steepest descent procedure. Here again, adaptive linear networks are trained on examples of correct behavior.

The LMS algorithm, shown here, is discussed in detail in Linear Neural Networks.

$$\mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha\mathbf{e}(k)\mathbf{p}^T(k)$$

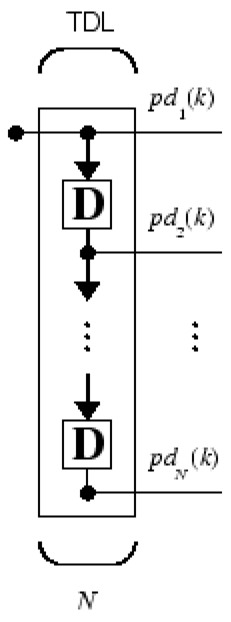
$$\mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha\mathbf{e}(k)$$

Adaptive Filtering (adapt)

The ADALINE network, much like the perceptron, can only solve linearly separable problems. It is, however, one of the most widely used neural networks found in practical applications. Adaptive filtering is one of its major application areas.

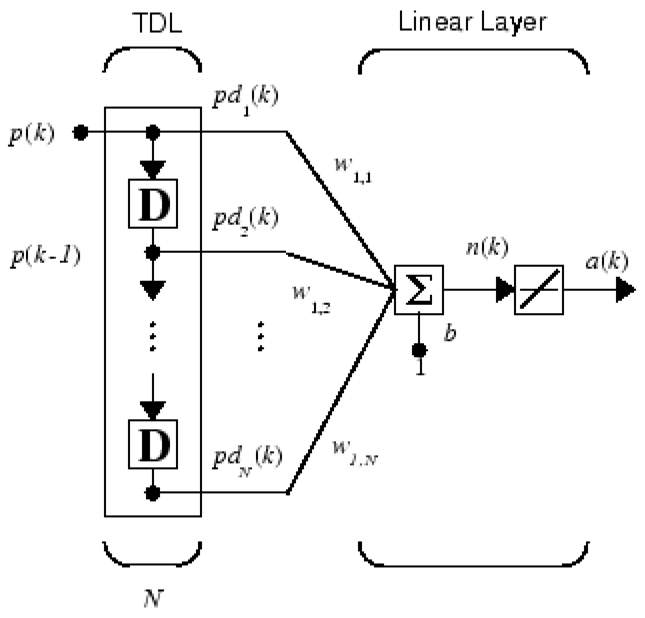
Tapped Delay Line

You need a new component, the tapped delay line, to make full use of the ADALINE network. Such a delay line is shown in the next figure. The input signal enters from the left and passes through *N*-1 delays. The output of the tapped delay line (TDL) is an *N*-dimensional vector, made up of the input signal at the current time, the previous input signal, etc.



Adaptive Filter

You can combine a tapped delay line with an ADALINE network to create the *adaptive filter* shown in the next figure.



The output of the filter is given by

$$\alpha(k)=purelin(\mathbf{Wp}+b)={}_{R}\sum_{i=1} w_{1,i}\alpha(k-i+1)+b$$

In digital signal processing, this network is referred to as a *finite impulse response (FIR)* filter [WiSt85]. Take a look at the code used to generate and simulate such an adaptive network.

LEAST-MEANS-SQUARE (LMS) ALGORITHM

The least-mean-square (LMS) algorithm is based on the use of instantaneous values for the cost function, namely, $E(w) = \frac{1}{2}e^2(n)$. Where e(n) is the error signal measured at time n.

Differentiation of E (w), with respect to w, yields:

$$\frac{\partial E(w)}{\partial w} = e(n)\frac{\partial e(n)}{\partial w}$$

As with the least-square filters, the LMS operates on linear neuron, we can write:

$$e(n) = d(n) - X^T(n)w(n)$$

$$\frac{\partial e(n)}{\partial w} = -X(n)$$

$$g(n) = \frac{\partial E(w)}{\partial w} = -e(n)X(n)$$

Weight update equation in steepest descent algorithm

$$w(n+1) = w(n) - \eta g(n)$$

Therefore, $w(n+1) = w(n) + \eta e(n)X(n)$. Where η is the learning rate parameter. The feedback loop around the weight vector w(n) in the LMS algorithm behaves like a low-pass filter, passing the low frequency components of the error signal and attenuating its high frequency components (Haykin, 1996). The average time constant of this filtering action is inversely proportional to the learning-rate parameter η. Hence, by assigning a small value to η, the adaptive process will progress slowly. More of the past data are then remembered by the LMS algorithm, resulting in a more accurate filtering action. In the steepest-descent algorithm the weight vector w(n) follows a well-defined trajectory in the weight space for a prescribed η. In contrast, in the LMS algorithm, the weight vector w(n) traces a random trajectory. For this reason, the LMS algorithm is sometimes referred to as "stochastic gradient algorithm."

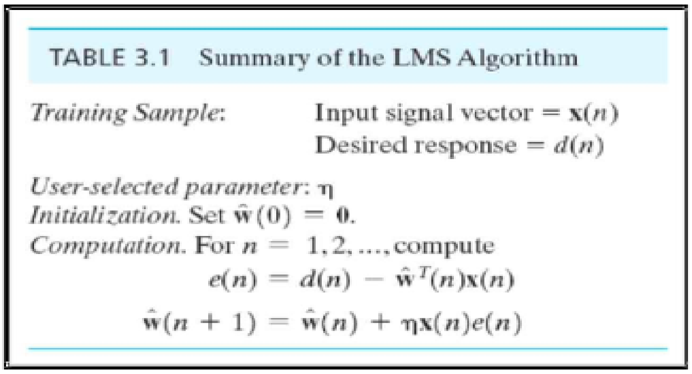


TABLE 3.1 Summary of the LMS Algorithm

Training Sample:	Input signal vector = $\mathbf{x}(n)$
	Desired response = $d(n)$

User-selected parameter: η

Initialization. Set $\hat{\mathbf{w}}(0) = \mathbf{0}$.

Computation. For $n = 1, 2, \ldots,$ compute

$$e(n) = d(n) - \hat{\mathbf{w}}^T(n)\mathbf{x}(n)$$

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

Unlike the steepest-descent, the LMS algorithm does not require knowledge of the statistics of the environment. It produces an instantaneous estimate of the weight vector. A summary of the LMS algorithm is presented in Table 3.1, which clearly illustrates the simplicity of the algorithm. As indicated in this table, for the initialization of the algorithm, it is customary to set the initial value of the weight vector in the algorithm equal to zero.

Convergence of the LMS Algorithm:

Convergence is influenced by the statistical characteristics of the input vector x(n) and the value assigned to the learning-rate parameter η.

By invoking the elements of independence theory and assuming the learning- rate parameter η is sufficiently small, it is shown in Haykin (1996) that the LMS is convergent in the mean square provided that η satisfies the condition

$$0 < \eta < \frac{2}{\lambda_{max}}$$

Where, λ_{max} is the largest eigenvalue of the correlation matrix R_x,

In typical applications of the LMS algorithm, knowledge of λ_{max} is not available. To overcome this difficulty, the trace of R_x, may be taken as a conservative estimate forλ_{max}, the condition is reformulated as

$$0 < \eta < \frac{2}{tr[R_x]}$$

Where, λ_{max} is the largest eigenvalue of the correlation matrix R_x,

By definition, the trace of a square matrix is equal to the sum of its diagonal elements. Since each diagonal element of the correlation matrix R_x equals the mean-square value of the corresponding sensor input

$$0 < \eta < \frac{2}{\text{sum of mean-square values of the sensor inputs}}$$

Virtues and Limitation of the LMS Algorithm:

Computational Simplicity and Efficiency:

- The Algorithm is very simple to code, only two or three line of code.
- The computational complexity of the algorithm is linear in the adjustable parameters.

Robustness:

Since the LMS is model independent, therefore it is robust with respect to disturbance, (small model uncertainty and small disturbances (i.e., disturbances with small energy) can only result in small estimation errors (error signals)).

Factors Limiting the LMS Performance:

The primary limitations of the LMS algorithm are:

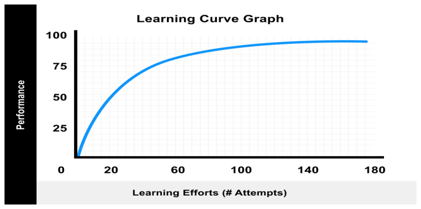
- Its **slow rate** of convergence (which become serious when the dimensionality of the input space becomes **high**)
- Its **sensitivity** to variation in the eigen structure of the input. (it typically requires a number of iterations equal to about 10 times the dimensionality of the input data space for it to converge to a stable solution)

- The sensitivity to changes in environment become particularly acute when the condition number of the LMS algorithm is high. The condition number,
- $\chi(R) = {}^{\lambda_{max}}/_{\lambda_{min}}$, Where λ_{max} and λ_{min} are the maximum and minimum eigenvalues of the correlation matrix, R_x.

Learning Curves

A learning curve is a correlation between a learner's performance on a task and the number of attempts or time required to complete the task; this can be represented as a direct proportion on a graph

The learning curve theory proposes that a learner's efficiency in a task improves over time the more the learner performs the task



Bias:
It is basically nothing but the difference between the average prediction of a model and the correct value of the prediction. Models with high bias make a lot of assumptions about the training data. This leads to over-simplification of the model and may cause a high error on both the training and testing sets. However, this also makes the model faster to learn and easy to understand. Generally, linear model algorithms like Linear Regression have a high bias.
Variance:
It is the amount a model's prediction will change if the training data is changed. Ideally, a machine learning model should not vary too much with a change in training sets i.e., the algorithm should be good at picking up important details about the data, regardless of the data itself. Example of algorithms with high variance is Decision Trees, Support Vector Machines (SVM)
The Artificial Neural Network literature has used the term to show the diverging behavior of in and out-of-sample performance as a function of the number of training iterations for a given number of training examples.
General Machine Learning uses learning curves to show the predictive generalization performance as a function of the number of training examples. Both graphs in Figure 3 are examples of such learning curves.

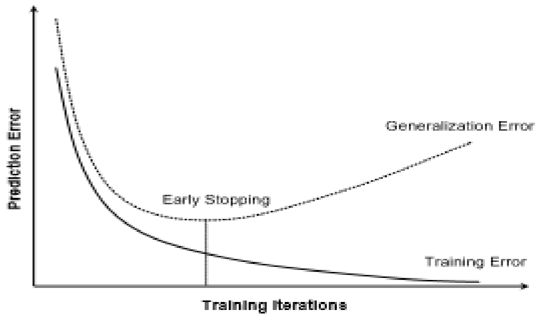


Figure 2: Learning curve for an artificial neural network.

Learning curve formula:- $Y = aX^b$

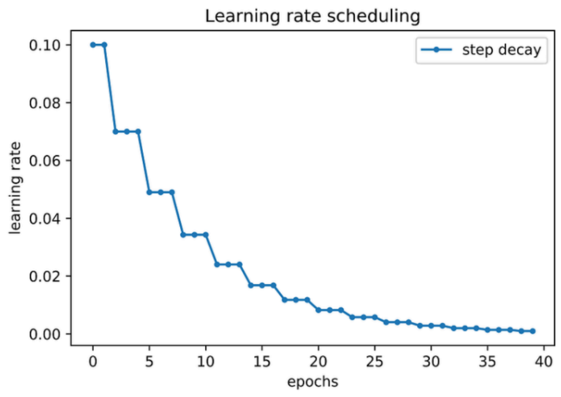
Where:
Y is the average time over the measured duration
a represents the time to complete the task the first time

X represents the total amount of attempts completed
^{b} represents the slope of the function

The formula can be used as a prediction tool to forecast future performance.

Learning Rate Annealing Techniques

- Changing the learning rate for your stochastic gradient descent optimization technique can improve performance while also cutting down on training time.
- This is also known as adaptable learning rates or learning rate annealing.
- This method is referred to as a learning rate schedule since the default schedule updates network weights at a constant rate for each training period.
- These have the advantage of making big modifications at the start of the training procedure when larger learning rate values are employed and decreasing the learning rate later in the training procedure when a smaller rate and hence smaller training updates are made to weights.



Methods of Learning Rate Annealing

1A Learning Rate Decay

2Adaptive Learning Rate

3Learning-Rate Warmup

Learning-Rate Decay

A schedule defines how things will change over time. In general, learning rate scheduling specifies a certain learning rate for each epoch and batch. There are two types of methods for scheduling global learning rates: the decay, and the cyclical one. The most preferred method is the learning rate annealing that is scheduled to gradually decay the learning rate during the training process

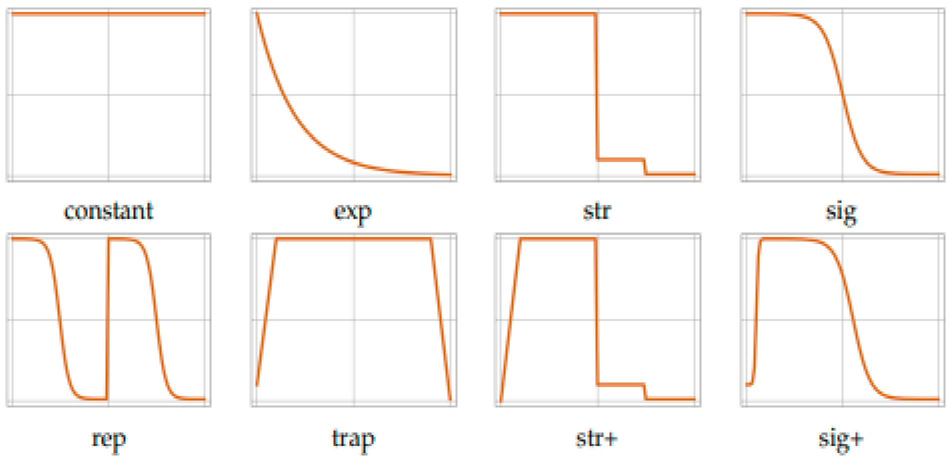
Adaptive Learning Rate

In the gradient-based optimization, it is desirable to determine the step-size automatically based on the loss gradient that reflects the convergence of each of the unknown parameters

However, the adaptive method is usually inferior to SGD in accuracy for unknown data in supervised learning, such as the image classification with conventional shallow model

Learning-Rate Warmup

The learning rate warmup, is a recent approach that uses a relatively small step size at the beginning of the training. The learning rate is increased linearly or non-linearly to a specific value in the first few epochs, and then shrinks to zero. The observations behind the warmup are that: the model parameters are initialized using a random distribution, and thus, the initial model is far from the ideal one; thus, an overly large learning rate causes numerical instability; and training a initial model carefully in the first few epochs may enable us to apply a larger learning rate in the middle stage of the training, resulting in a better regularization The bottom row of Figure provides the learning rate schedules by the conventional annealing methods with warmup.



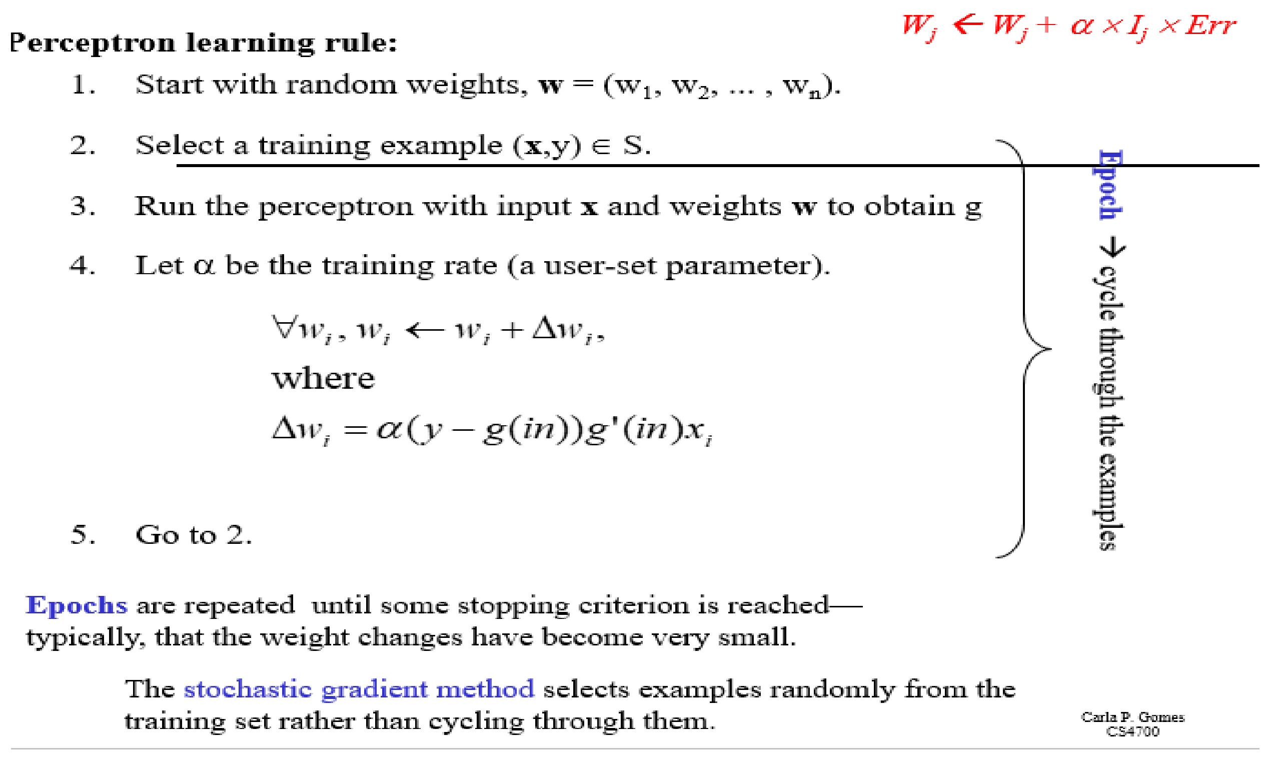
Perceptron –Convergence Theorem

Perceptron Learning Algorithm

```
Initialization: Examples $\{(\mathbf{x}_g, y_g)\}_{g=1}^{N}$, initial weights $w_i$ set to small random
values, learning rate parameter $\eta = 0.1$
Repeat
    for each training example $(\mathbf{x}_g, y_g)$
        - calculate the output: $o = Threshold(\Sigma_{i=0}^{d} w_i x_{ig})$
        - if the Perceptron does not respond correctly update the weights:
            $w_i = w_i + \eta (y_g - o_g) x_{ig}$ // this is the Perceptron Rule
    until termination condition is satisfied.
```

where: y_g is the desired output, o_g is the output generated by the Perceptron,

w_i is the weight associated with the i-th connection.



$$W_j \leftarrow W_j + \alpha \times I_j \times Err$$

Perceptron learning rule:

1. Start with random weights, $\mathbf{w} = (w_1, w_2, \ldots, w_n)$.
2. Select a training example $(\mathbf{x}, y) \in S$.
3. Run the perceptron with input $\mathbf{x}$ and weights $\mathbf{w}$ to obtain g
4. Let α be the training rate (a user-set parameter).

$$\forall w_i,\ w_i \leftarrow w_i + \Delta w_i,$$

where

$$\Delta w_i = \alpha (y - g(in)) g'(in) x_i$$

5. Go to 2.

Epoch → cycle through the examples

Epochs are repeated until some stopping criterion is reached—typically, that the weight changes have become very small.

The stochastic gradient method selects examples randomly from the training set rather than cycling through them.

Carla P. Gomes
CS4700

The role of the learning parameter is to moderate the degree to which weights are changed at each step of the learning algorithm.
During learning the decision boundary defined by the Perceptron moves, and some points that have been previously misclassified will become correctly classified so that the set of examples that contribute to the weighted sum change.
2.2. Perceptron Convergence Theorem
The Perceptron convergence theorem states that for any data set which is linearly separable the Perceptron learning rule is guaranteed to find a solution in a finite number of steps.
In other words, the Perceptron learning rule is guaranteed to converge to a weight vector that correctly classifies the examples provided the training examples are linearly separable.
A function is said to be linearly separable when its outputs can be discriminated by a function which is a linear combination of features, that is we can discriminate its outputs by a line or a hyperplane.
Example: Suppose an example of perceptron which accepts two inputs $x_1 = 2$ and $x_2 = 1$, with weights $w_1 = 0.5$ and $w_2 = 0.3$ and $w_0 = -1$.
The output of the perceptron is :

$$O = 2 * 0.5 + 1 * 0.3 - 1 = 0.3$$

Therefore the output is 1. If the correct output however is 0, the weights will be adjusted according to the Perceptron rule as follows:

$$w_1 = 0.5 + (0 - 1) * 2 = - 1.5$$

$$w_2 = 0.3 + (0 - 1) * 1 = - 0.7$$

$$w_0 = - 1 + (0 - 1) * 1 = - 2$$

Relation Between Perceptron and Bayes Classifier for a Gaussian Environment

- Bayesian decision theory is a fundamental statistical approach to the problem of classification as for pattern recognition.
- It makes the assumption that the decision problem is posed in probabilistic term, and all of the relevant probability values are known.
- To minimize the error probability in classification problem, one must choose the state of nature that maximizes the posterior probability.
- Classification techniques employ a learning algorithm to identify a model that best fits the relationship between attribute set and class label for the input data.
- Its clear that the Bayesian Decision Rule(BDR) has great role in statistical data analysis for various directions in our live, specially in stochastic processes.
- This can be done when one needs to make classification among some classes come from several populations, to return to their origin.
- This criteria can be done by using what is so called (Perceptron)

Back Propagation Algorithm

A neural network is a group of connected I/O units where each connection has a weight associated with its computer programs.

It helps you to build predictive models from large databases.

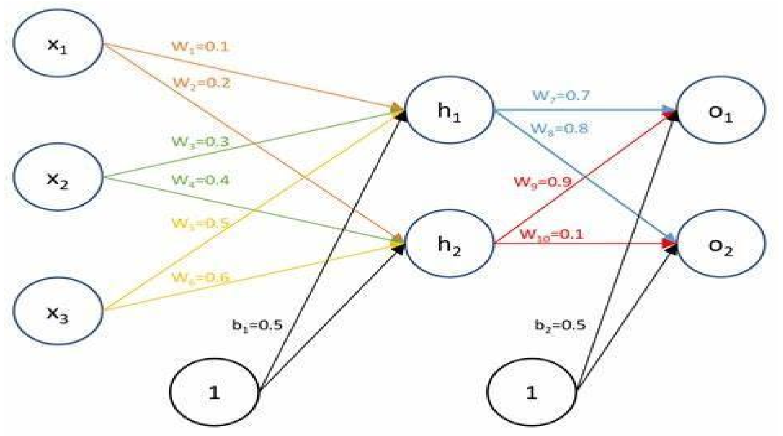
This model builds upon the human nervous system.

Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration).

Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalizationBackpropagation in neural network is a short form for "backward propagation of errors."

It is a standard method of training artificial neural networks.

This method helps calculate the gradient of a loss function with respect to all the weights in the network



Among various logical gates, the XOR or also known as the "exclusive or" problem is one of the logical operations when performed on binary inputs that yield output for different combinations of input, and for the same combination of input no output is produced.

The outputs generated by the XOR logic arenot linearly separable in the hyperplane.

Working of Back Propagation Algorithm

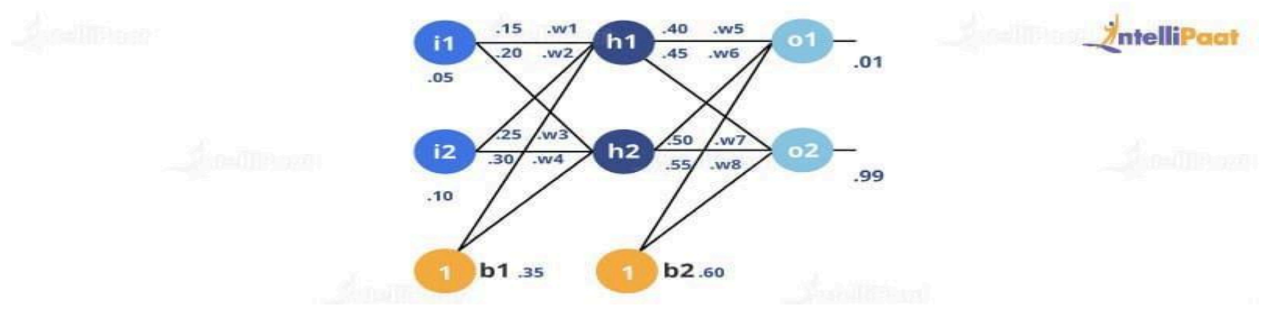
The goal of the back propagation algorithm is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs. Here, we will understand the complete scenario of back propagation in neural networks with the help of a single training set.

In order to have some numbers to work with, here are initial weights, biases, and training input and output.

```
Inputs(i1): 0.05          Output (o1): 0.01

Inputs(i2): 0.10          Output(o2):0.99
```

Step 1: The Forward Pass:



The total net input for h1: The net input for h1 (the next layer) is calculated as the sum of the product of each weight value and the corresponding input value and, finally, a bias value added to it.

$$\text{net } h1 = w_1 * i_1 + w_2 * i_2 + b_1 * 1 \ldots\ldots\ldots\ldots\ldots\ldots \text{(Equation 1)}$$

$$\text{net } h1 = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

The output for h1: The output for h1 is calculated by applying a sigmoid function to the net input Of h1.

The sigmoid function pumps the values for which it is used in the range of 0 to 1.

It is used for models where we have to predict the probability. Since the probability of any event lies between 0 and 1, the sigmoid function is the right choice.

$$\text{out } h1 = 1/(1 + e^{-\text{net } h1}) = 1/(1 + e^{-0.3775}) = 0.593269992 \ldots\ldots\ldots\ldots\ldots\ldots \text{(Equation 2)}$$

Carrying out the same process for h2

```
out h2 = 0.596884378
```

The output for o1 is:

$$\text{net } o1 = w_5 * \text{out } h1 + w_6 * \text{out } h2 + b_2 * 1 \text{ .... (Equation 3)}$$

$$\text{net } o1 = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$\text{out } o1 = 1/(1 + e^{-\text{net } o1}) = 1/(1 + e^{-1.105905967}) = 0.75136507 \text{ .... (Equation 4)}$$

Carrying out the same process for o2:

```
out o2 = 0.772928465
```

Calculating the Total Error:

We can now calculate the error for each output neuron using the squared error function and sum them up to get the total error: E total = Σ1/2(target – output)2

The target output for o1 is 0.01, but the neural network output is 0.75136507; therefore, its error is:

E o1 = 1/2(target o1 - out o1)2 = 1/2(0.01 - 0.75136507)2 = 0.27481108 ……………..……………. (Equat

By repeating this process for o2 (remembering that the target is 0.99), we get:

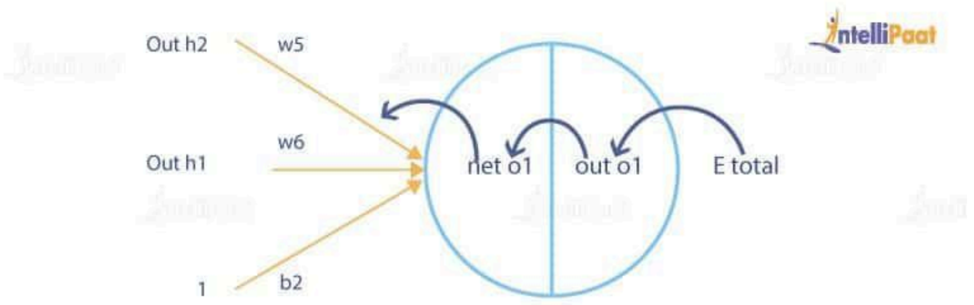
E o2 = 0.023560026

Then, the total error for the neural network is the sum of these errors:

E total = E o1 + E o2 = 0.274811083 + 0.023560026 = 0.298371109

Step 2: Backward Propagation:

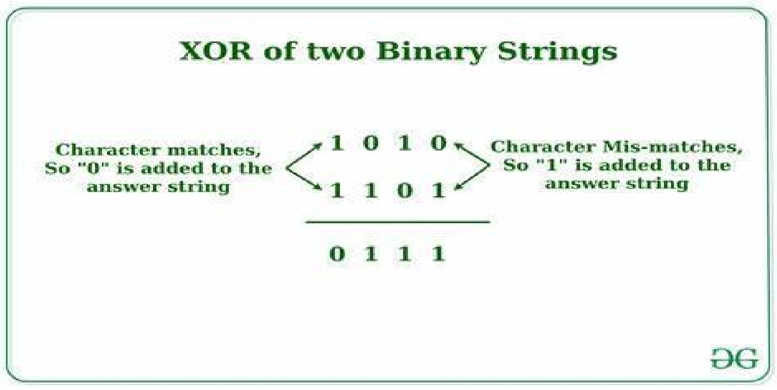
Our goal with the backward propagation algorithm is to update each weight in the network so that the actual output is closer to the target output, thereby minimizing the error for each neuron and the network as a whole.

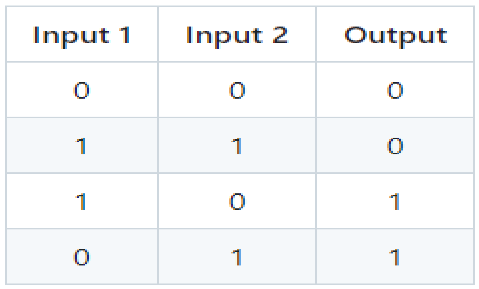


XOR Problem

The XOR logic is used to and how to integrate the XOR logic using neural networks.

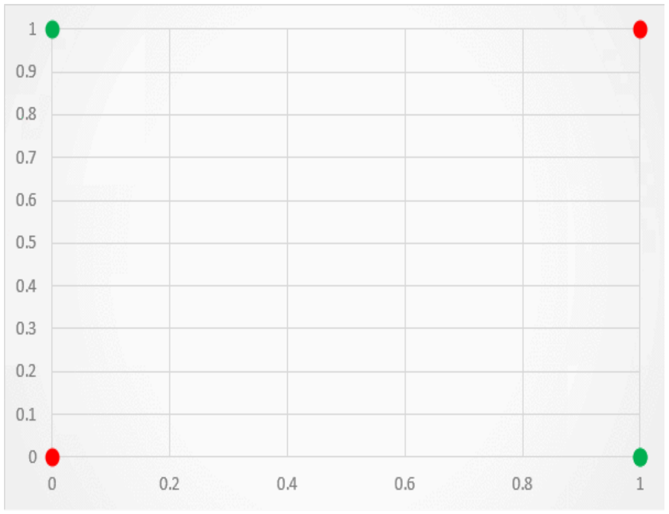
XOR or Exclusive OR is a classic problem in Artificial Neural Network Research. An XOR function takes two binary inputs (0 or 1) & returns True if both inputs are different & False if both inputs are same.


XOR of two Binary Strings
Character matches, So "0" is added to the answer string
1 0 1 0
1 1 0 1
0 1 1 1
Character Mis-matches, So "1" is added to the answer string
GG



Input 1	Input 2	Output
0	0	0
1	1	0
1	0	1
0	1	1

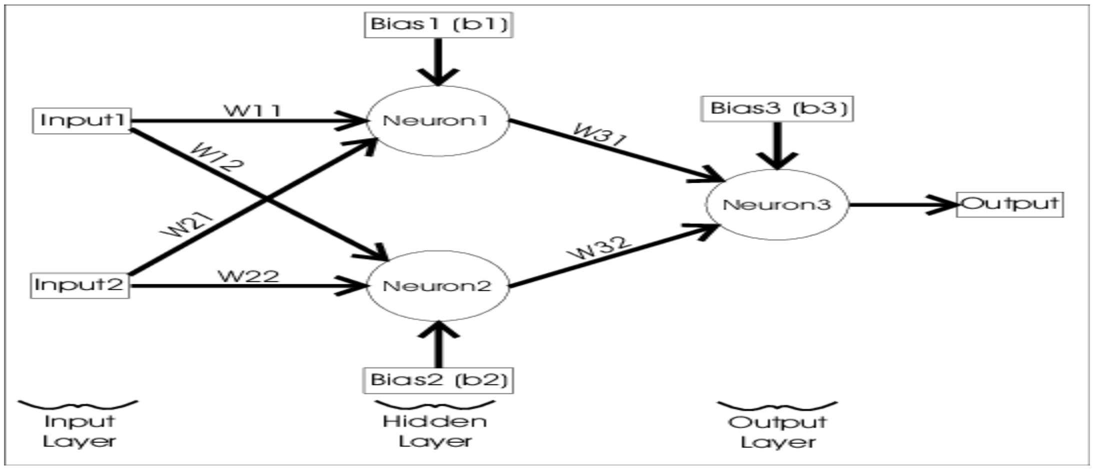
On the surface, XOR appears to be a very simple problem, however, Minksy and Papert (1969) showed that this was a big problem for neural network architectures of the 1960s, known as perceptrons. A limitation of this architecture is that it is only capable of separating data points with a single line. This is unfortunate because the XOR inputs are not linearly separable. This is particularly visible if you plot the XOR input values to a graph. As shown in the figure, there is no way to separate the 1 and 0 predictions with a single classification line.



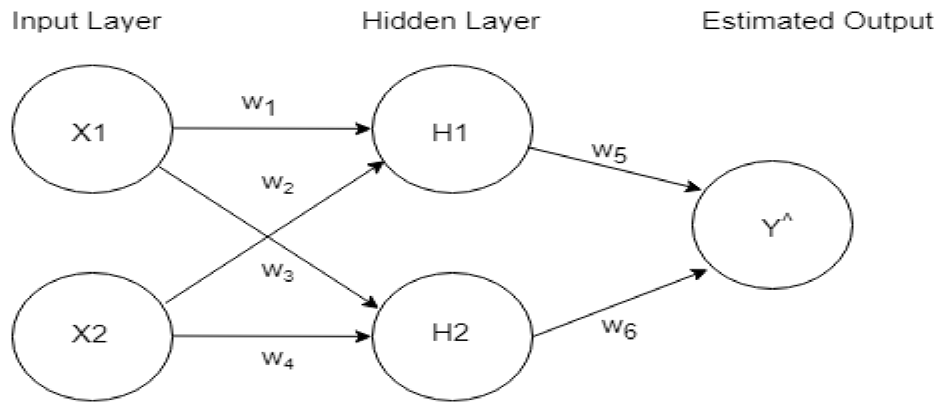
Solution

The backpropagation algorithm begins by comparing the actual value output by the forward propagation process to the expected value and then moves backward through the network, slightly adjusting each of the weights in a direction that reduces the size of the error by a small degree. Both forward and back propagation are re-run thousands of times on each input combination until the network can accurately predict the expected output of the possible inputs using forward propagation.

As mentioned before, the neural network needs to produce two different decision planes to linearly separate the input data based on the output patterns. This is achieved by using the concept of *hidden layers*. The neural network will consist of one input layer with two nodes (X1,X2); one hidden layer with two nodes (since two decision planes are needed); and one output layer with one node (Y). Hence, the neural network looks like this:



Model



Inputs

$$
\begin{aligned}
\mathbf{x_1} &= [1,1]^T, \quad y_1 = +1\\
\mathbf{x_2} &= [0,0]^T, \quad y_2 = +1\\
\mathbf{x_3} &= [1,0]^T, \quad y_3 = -1\\
\mathbf{x_4} &= [0,1]^T, \quad y_4 = -1
\end{aligned}
$$

2 hidden neurons are used, each takes two inputs with different weights. After each forward pass, the error is back propogated. I have used sigmoid as the activation function at the hidden layer.

At hidden layer:

$$H_1 = x_1w_1 + x_2w_2$$

$$H_2 = x_1w_3 + x_2w_4$$

At output layer:

$$Y^{\wedge} = \sigma(H_1)w_5 + \sigma(H_2)w_6$$

here, σ represents sigmoid function.

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

Loss function:

$$\frac{1}{2}(Y - Y^{\wedge})^2$$

Heuristics

- Heuristics are strategies often used to find a solution that is not perfect, but is within an acceptable degree of accuracy for the needs of the process.

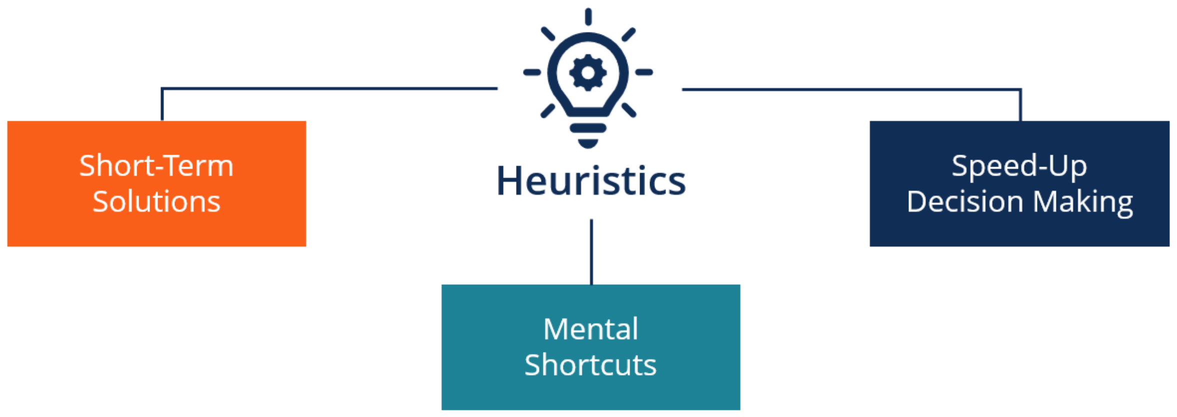
In computing, heuristics are especially useful when finding an optimal solution to a problem is impractical because of slow speed or processing power limitations

The heuristic ANN design approach is made up of the following steps: knowledge-based selection of input values, selection of a learning method, and design of the hidden layers (quantity and nodes per layer).

A heuristic technique is a problem specific approach that employs a practical method that often provides sufficient accuracy for the immediate goals. From: Numerical Methods

Types of Heuristics

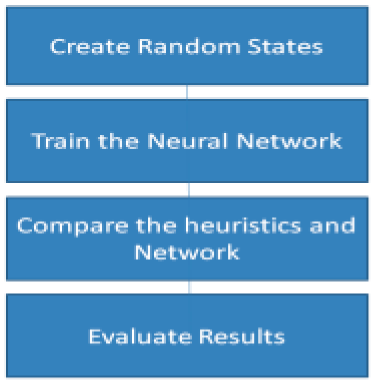
- Affect Heuristics. Affect heuristics are based on positive and negative feelings that are associated with a certain stimulus. ...
- Availability Heuristics. ...
- Representative Heuristics.



Heuristic algorithms are used in a variety of industries, including transportation, logistics, and finance. For example, heuristic algorithms can be used to optimize delivery routes for packages or to minimize risk in financial investments

Heuristic evaluations are certainly useful in some instances and can provide crucial insights into how your site is meeting its objectives without the time, expense and potential problems of real user evaluation. It can. However, be risky to rely on it as the sole means of testing your concept and product.

the neural network's predictions are responsible for Combining heuristics with neural networks



Output Representation and Decision Rule

- To balance the accuracy of neural networks and the interpretability of decision rules, we propose a hybrid technique called rule-constrained networks.

Namely, neural networks that make decisions by selecting decision rules from a given ruleset

learning about a new classifier is what kind of decision boundaries can this classifier learn. We have explored this question in HW1 for the case of 1-KNN and decision trees, and showed that both of them can vary their decision boundary either on a data-driven way for 1-KNN or based on the size of the tree for decision trees. In this question we will explore these issues for the case of a 2-layer Neural Network (NN). Recall from class that the input aj for a node j is given by:

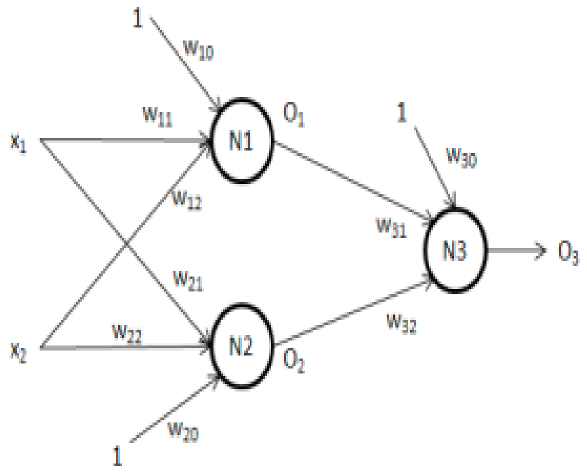
$$a_j = \sum_i w_{ji} o_i$$

Where, w_{ji} is the weight from unit i to unit j, and o_i is the activation/output of unit i. The activation of unit i is the output of a logistic function in this problem (although any differentiable function is allowed):

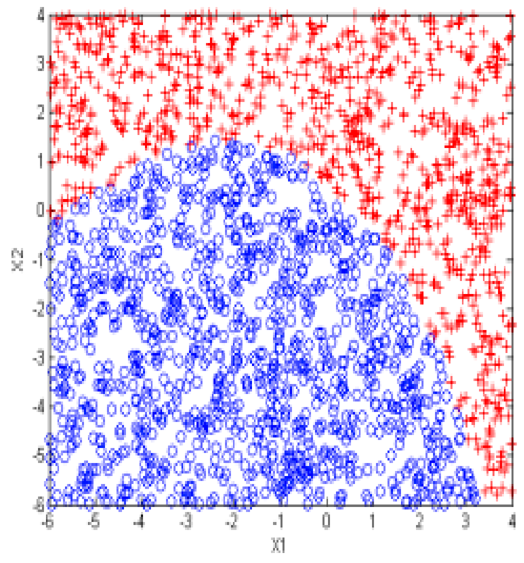
$$o_i = \sigma(a_i) = \frac{1}{1 + \exp^{(-a_i)}}$$

1.1 Decision Boundary

Consider the classification task shown in figure 1 where '+' and 'o' denotes positive and negative classes, respectively. This data is available in the file data.mat, also please read NN readme.txt which contains a simple code to draw the figure below. For this part of the problem, you might want to write really a few lines of matlab code to evaluate the network and get the necessary plots. Consider the 2-layer network in Figure 1. This network has 9 weights and a logistic activation function for both the hidden and output layers.



(a)



(b)

Figure 1: (a) 2-layer NN with logistic activation functions at both the hidden and output layers.

(b) A 2-class dataset: '+' and 'o' marks positive and negative labels respectively. (1) [2 points] For each of the following classifiers, state with a one-line explanation whether or not they can learn the decision boundary illustrated in Figure 1: 1-KNN, decision tree, Naive 1 Bayes, and logistic regression. Solution: NB and LR can only learn linear decision boundaries thus they can not learn the decision boundary in Figure 1. 1-KNN can learn that boundary. For decision trees, you can argue either way, if you consider the approach we discussed in HW1, then if you don't constrain the depth of the tree, then definitely DT can learn that boundary with a very large tree (which is not practical), thus you can argue that using a bounded-size tree, the decision boundary in Figure 1 can not be learned.

(2) **[1 point]** Express o_1 and o_2 in terms of $x_1, x_2, w_{10}, w_{11}, w_{12}, w_{20}, w_{21}, w_{22}$.

Solution: $o1 = \sigma(w_{10} + w_{11}x_1 + w_{12}x_2)$ $o_2 = \sigma(w_{20} + w_{21}x_1 + w_{22}x_2)$

(3) **[1 point]** Write down the decision rule for this 2-layer NN classifier.

Solution: If $o_3 >= .5$ then predict class one otherwise predict class 0;

(4) Consider the following set of weights: $w_{10} = -.8; w_{11} = .8; w_{12} = .1; w_{20} = .3; w_{21} = .3; w_{22} = -.4; w_{31} = 1; w_{32} = -1; w_{30} = .2;$

(a) **[2 points]** Draw the representation of the data after the hidden layer, i.e. your dimensions will be o_1 and o_2, and you should label each point with its ground-truth label. State your observation.

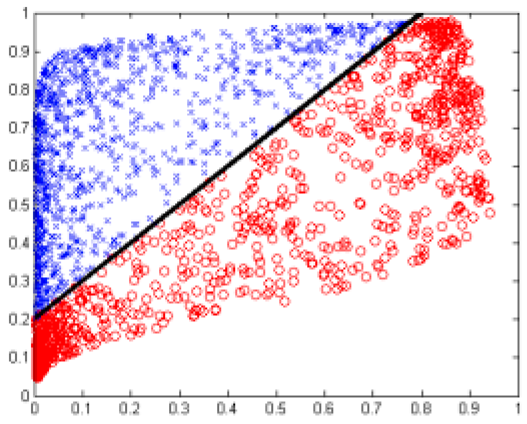


Figure 2:

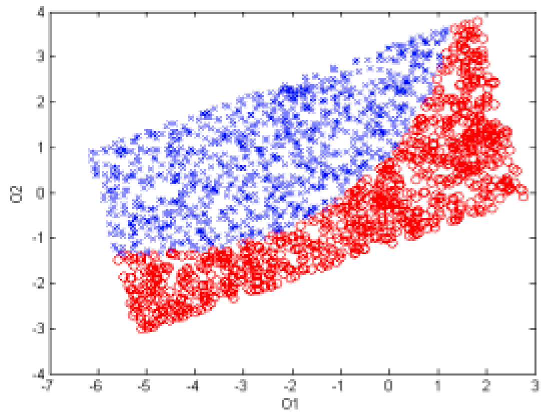
The data becomes linearly separable in this space.

(b) **[2 points]** Draw the classification result for this 2-layer NN, i.e. re-draw the scatter plot in Figure 1 but label each point as classified by the 2-layer NN. Compare to the ground-truth and state your observations.

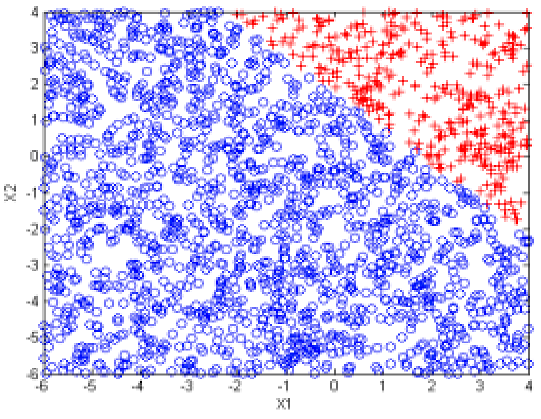
Solution: same as in Figure 1.b. Since the final layer is just a logistic regression classifier whose input is (o_1, o_2), the correct classifier can be learned. The boundary in terms

of the input space (x1, x2) is non-linear though. (c) [2 points] Overlay the decision boundary explicitly over the curve your drew in part (a) in terms of o1 and o2. State your observations.

Solution: Note that the last layer is a logistic regression classifier over the input (o1, o2), thus the decision rule is: w31o1 + w32o2 + w30 = 0. Which you can easily draw as in Figure 2. (e) Lets assume that we removed the logistic function form the hidden layer ONLY and instead used an identity function, i.e o1 = a1 and o2 = a2, while maintaining the same weight values. (i) [2 points] repeat (a) and (b) and state your observations.



(a)



(b)

Figure 3: (a) After first layer. (b) prediction made by the modified NN

Solution:
See figure 3. The data is not linearly separable in terms of (o_1, o_2) thus the final logistic unit can not learn the correct decision boundary. In terms of the input space (x_1, x_2), as shown in figure 3, we got the wrong decision boundary. In fact, this network, as we will show in the next part, can only learn a linear decision boundary over (x_1, x_2) and as shown in figure 3.b, the network learned a liner decision boundary (which is not correct). Note that this is not the best linear boundary that this network can learn, in other words, you can optimize the weights to get a better linear decision boundary, but the network can not still learn the correct decision boundary, nor learn a linear decision boundary that does a good job on this dataset (in terms of classification accuracy)

(ii) [2 points] Can you tweak the weights in this case to learn the correct decision boundary? If yes, then find such weights and repeat (i), if NO, then re-express the network in this case using a simpler network (or classifier) and argue why it can not learn this decision boundary.

Solution: The answer is NO, since you can simplify the 2-layer NN into a singlelayer NN with a logistic unit (i.e. a logistic regression classifier) which can only a linear decision boundary over (x1, x2). To see this:

$$\begin{aligned} o_3 &= \sigma(w_{30} + w_{31}o_1 + w_{32}o_2) \\ &= \sigma\Big(w_{30} + w_{31}(w_{10} + w_{11}x_1 + w_{12}x_2) + w_{32}(w_{20} + w_{21}x_1 + w_{22}x_2)\Big) \\ &= \sigma\Big(w_{30} + w_{31}w_{10} + w_{32}w_{20} + (w_{31}w_{11} + w_{32}w_{21})x_1 + (w_{31}w_{12} + w_{32}w_{22})x_2\Big) \end{aligned}$$

Which is just a logistic regression classifier that can learn ONLY a linear decision boundary over the space (x_1, x_2)

Computer experiment

Nowadays there is a great interest in artificial neural networks, but the materials on this topic are either incomprehensible for high school students or offer ready-made solutions using high-level functions such as TensorFlow. The purpose of this work: the development of a computer program that allows conducting a series of experiments using mathematical modeling methods, as a tool for studying the features of neural networks functioning with a reasonable combination of mathematics and practice in order to be apprehensible for high school students. The work includes the following steps: Developing a list of more than 20 parameters and metrics used to control the neural network. Some of them were developed by the author of this work. Developing a computer program in Python with a plan of 12 different experiments. Experimenting with the neural network the user can evaluate how various parameters of the neural network affect the efficiency of its work, assess the impact of the training sample size on the training of the neural network, etc. It is possible to compare the neural network and the naive Bayes classifier. The program can work autonomously and in the Google Colaboratory environment. The results of the program are displayed in form of graphs, tables and images of matrices. Animation of the images improves perception of the dynamics of weights matrix changes

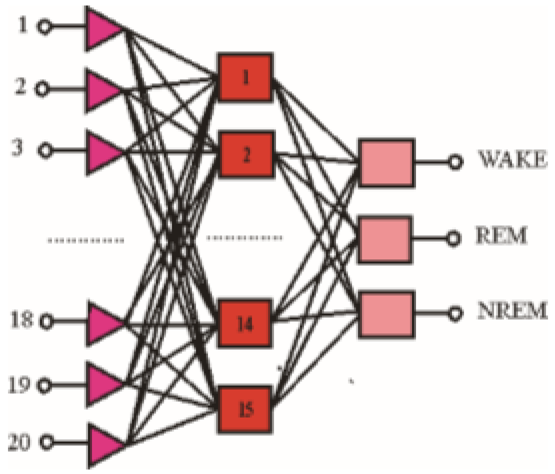


Figure 2. Artificial neural network architecture used in computer experiment

Feature Detection

- Feature detection is a low-level image processing operation.
- That is, it is usually performed as the first operation on an image, and examines every pixel to see if there is a feature present at that pixel.
- If this is part of a larger algorithm, then the algorithm will typically only examine the image in the region of the features.
- Feature detection is a process by which the nervous system sorts or filters complex natural stimuli in order to extract behaviorally relevant cues that have a high probability of being associated with important objects or organisms in their environment, as opposed to irrelevant background or noise.

ASSIGNMENT-2

SHORT ANSWERS(1 MARK)

1. Write a short note on perceptron.
2. Briefly describe about Learning Curves.
3. Write a short note on Heuristics.
4. Briefly describe about Output Representation and Decision Rule.
5. Write a short note on Computer Experiment.
6. Write a short note on Feature Detection.

LONG ANSWERS(5 MARKS)

1. Explain about Adaptive Filtering Problem in detail.
2. Explain in depth about Least Mean Square Algorithm.
3. Explain about Learning Rate Annealing Techniques in detail.

4. Explain in depth about Perceptron –Convergence Theorem.
5. Explain about Back Propagation Algorithm XOR Problem in detail.
6. Explain in depth about Unconstrained Organization Techniques.
7. Explain about Linear Least Square Filters in detail.
8. Explain in depth about Relation Between Perceptron and Bayes Classifier for a Gaussian Environment Multilayer Perceptron.
9. Explain about SLP and MLP in detail.

ACTIVATION FUCTION:

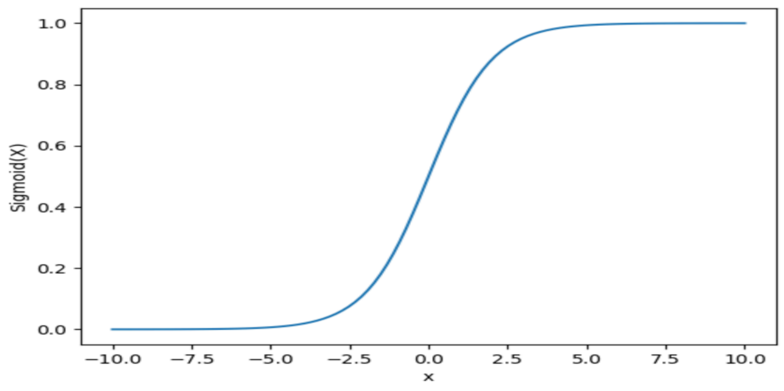
The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

Variants of Activation Function

Linear Function

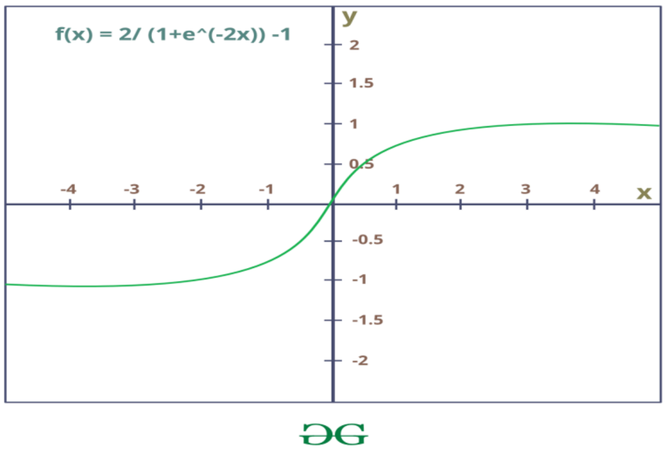
- **Equation :** Linear function has the equation similar to as of a straight line i.e. **y = x**
- No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
- **Range :** -inf to +inf
- **Uses : Linear activation function** is used at just one place i.e. output layer.
- **Issues :** If we will differentiate linear function to bring non-linearity, result will no more depend on *input "x"* and function will become constant, it won't introduce any ground-breaking behavior to our algorithm.

Sigmoid Function



- It is a function which is plotted as **'S'** shaped graph.
- **Equation :** A = 1/(1 + e^{-x})
- **Nature :** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range :** 0 to 1
- **Uses :** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.

Tanh Function



- The activation that works almost always better than sigmoid func
- tion is Tanh function also known as **Tangent Hyperbolic function**. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.

- **Value Range :-** -1 to +1
- **Nature :-** non-linear

- **Uses :-** Usually used in hidden layers of a neural network as it's values lies between **-1 to 1** hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.

RELU Function

- It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.
- **Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range :-** [0, inf)
- **Nature :-** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- **Uses :-** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

In simple words, RELU learns *much faster* than sigmoid and Tanh function.

Softmax Function

- It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.
- **Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range :-** [0, inf)
- **Nature :-** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- **Uses :-** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

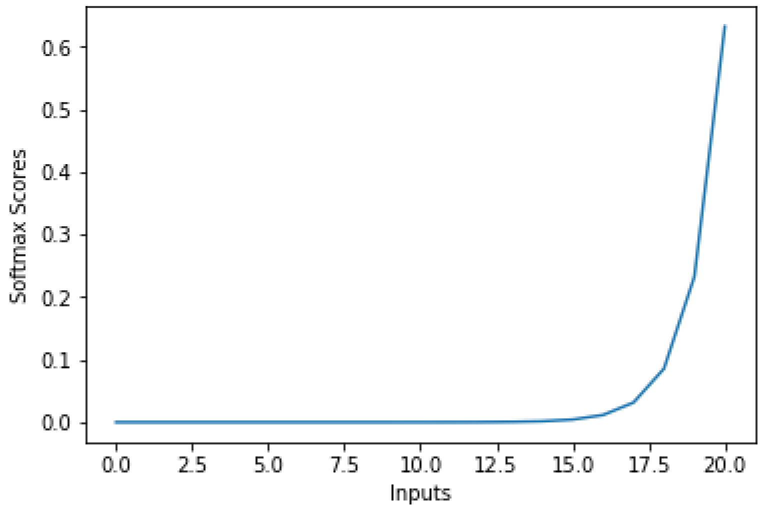
In simple words, RELU learns *much faster* than sigmoid and Tanh function.

Softmax Function

- It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.
- **Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range :-** [0, inf)
- **Nature :-** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- **Uses :-** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

In simple words, RELU learns *much faster* than sigmoid and Tanh function.

Softmax Function



The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi- class classification problems.

- **Nature :-** non-linear
- **Uses :-** Usually used when trying to handle multiple classes. the softmax function was commonly found in the output layer of image classification problems.The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
- **Output:-** The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.
- The basic rule of thumb is if you really don't know what activation function to use, then simply use *RELU* as it is a general activation function in hidden layers and is used in most cases these days.
- If your output is for binary classification then, *sigmoid function* is very natural choice for output layer.
- If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each classes.

LEARNING

An action a learner can engage in on their own to acquire a piece of knowledge or master a skill.

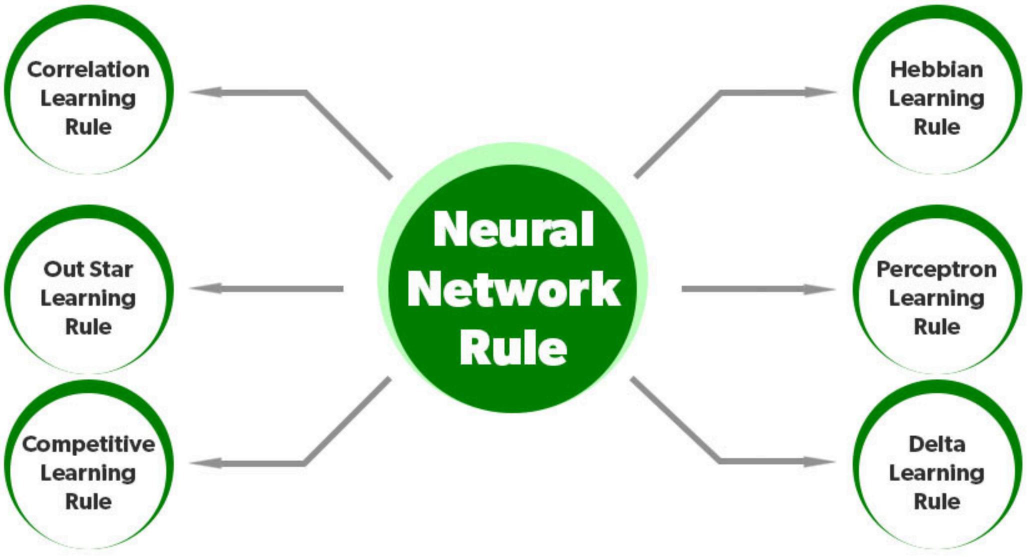
To summarize, neural networks, inspired by the human brain, learn patterns and make predictions based on data through interconnected neurons in input, hidden, and output layers. They learn by adjusting weights and biases using forward propagation, loss computation, backpropagation, and parameter updates.

Types of Learning in Neural Networks

- Supervised Learning :
- Unsupervised Learning :
- Reinforcement Learning :

Types Of Learning Rules in ANN

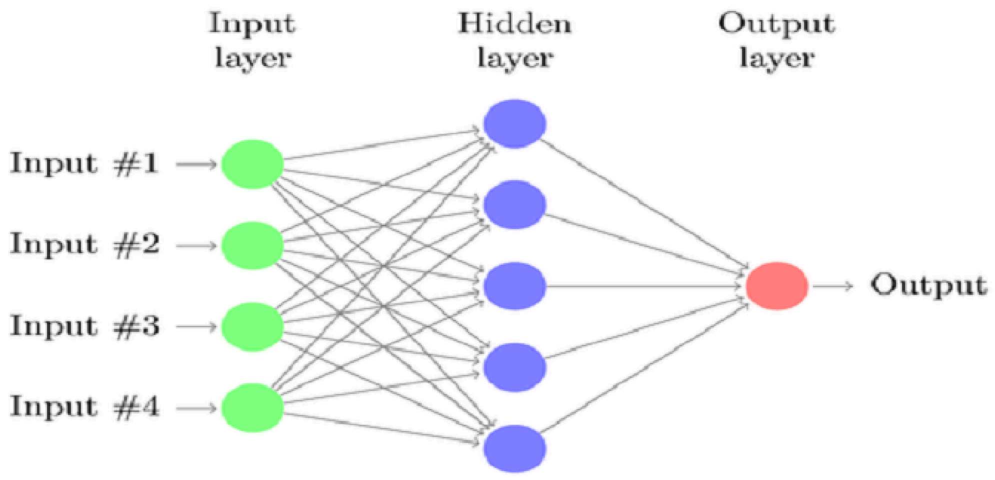
Learning rule enhances the Artificial Neural Network's performance by applying this rule over the network. Thus learning rule updates the weights and bias levels of a network when certain conditions are met in the training process. it is a crucial part of the development of the Neural Network.



TRAINING

Neural network training is the process of teaching a neural network to perform a task. Neural networks learn by initially processing several large sets of labeled or unlabeled data. By using these examples, they can then process unknown inputs more accurately
Key Steps for Training a Neural Network

- Pick a neural network architecture. ...
- Random Initialization of Weights: The weights are randomly intialized to value in between 0 and 1, or rather, very close to zero.


Input layer
Hidden layer
Output layer
Input #1
Input #2
Input #3
Input #4
Output