Artificial Neural Networks (ANN)

→ Neural N/W is used to describe a N/W of biological neurons that are functionally connected to central nervous system of living organisms.

→ ANN is composed of artificial neurons or nodes that are connected together to form a N/W.

→ ANN uses a mathematical or computational model for processing of information.

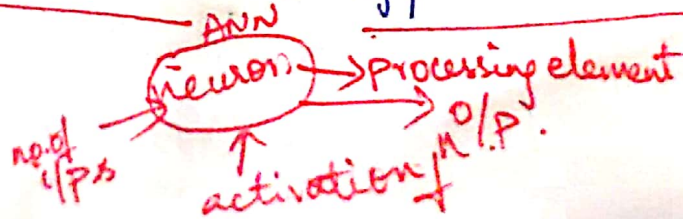
→ ANN is capable of modifying its structure depending on external or internal information being transmitted through system.

→ So ANNs can be non-linear statical data modeling or decision-making tools.

→ We can say ANN is a M/c learning approach that follows the same learning principle employed by brain.

→ ANN is defined as a N/W consisting of a large no. of neurons connected to each other with some weights & is also referred to as connectionist model.

→ Each neuron in ANN is considered to be a processing element that receives a no. of i/ps on which an activation function is applied to obtain the o/p value of neuron.



notes bias and V is called induced
h has effect of applying a transformation to

→ Neurons are trained by using knowledge of domain which is given in form of training examples.

→ Aim of training ANN is to create a generalized n/w that can behave correctly even in case of new instances that have not been experienced or learned previously.

→ ANN may be Specified by following components:

- Neuron model: The information processing unit of ANN.
- Architecture: A set of neurons & links connecting neurons, where each link possesses a weight.
- A learning algorithm: For training ANN by modifying the weights in order to model a particular learning task correctly on training examples.

Neuron Model

Neuron Consists:

- Inputs ✓
- Weights ✓
- Adder ✓
- Activation function ✓

→ It Consists of set links describing neuron i/ps (x_i) with weights $w_1, w_2 \cdots w_m$, and an adder f^n (linear combiner) for Computing weighted Sum (U) of i/ps as

$$U = \sum_{i=1}^{m} w_i * x_i$$

where m denotes the no. of i/ps.

→ It requires an activation f^n φ for limiting the amplitude of neuron o/p. o/p (Y) of neuron is

$$Y = \varphi(U + b) = \varphi(V), \text{ where } V = U + b.$$

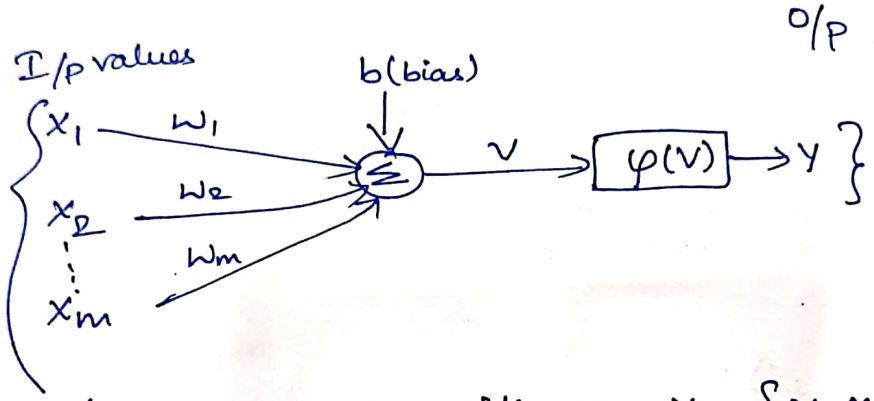
→ b denotes bias and V is called induced field of neurons.

→ bias b has effect of applying a transformation to weighted sum V and is an external parameter of neuron.

The same above expression can be written as

$$V = \sum_{i=0}^{m} w_i * x_i \quad \text{where } w_0 = b \text{ \& } x_0 = 1.$$

Neuron Diagram



Architecture of neuron with m i/ps $\{x_1, x_2, \ldots, x_m\}$ & one o/p Y, obtained after applying activation f^n on weighted sum.

Activation Functions or.

~~Ob. is~~ one of the factors that determine nature of a neuron model, ~~is activation~~ is activation f^n & φ.

Commonly used activation f^n's are

- Step function is also known as Sign f^n.
- Ramp "
- Sigmoid "
- Gaussian "

→ Step function is used for binary classification.

→ When i/p data is continuous then the remaining 3 f^n's may be applied as activation f^n's for classification & regression problems.

formulas

Step function

$$\varphi(v) = \begin{cases} a, & \text{if } v < c \\ b, & \text{if } v \geq c \end{cases}$$

Ramp f^n

$$\varphi(v) = \begin{cases} a, & \text{if } v < c \\ b, & \text{if } v > d \\ a + ((v-c) * (b-a)/(d-c)), & \text{otherwise} \end{cases}$$

Sigmoid f^n with Z, X, Y parameters

$$\varphi(v) = Z + \frac{1}{1 + \exp(-x * v + y)}$$

Gaussian f^n:

Simplest form

$$\varphi(v) = \exp\left(-\frac{(v-c)^2}{\sigma^2}\right)$$

c represents center & σ denotes spread of radius.

Neural N/W Architectures

→ Architecture of a neural N/W is associated with learning algorithm used to train it.

→ There are three different classes of N/W architectures.

- Single-layer feed-forward N/W
- Multi-layer " " "
- Recurrent N/W.

Single - layer Feed - Forward N/ws.

→ Feed - forward Neural N/ws were first & simplest of ANNs to be developed.

→ A Single - layer feed - forward Neural N/w is defined as N/w where connections b/w various nodes donot form a directed cycle such that the information in this N/w moves only in the forward direction. i.e., from i/p nodes to o/p nodes, through a series of weights. no hidden layers.

→ It contains only i/p & o/p layer.

→ Each i/p node is connected to each o/p node with a link having weight.

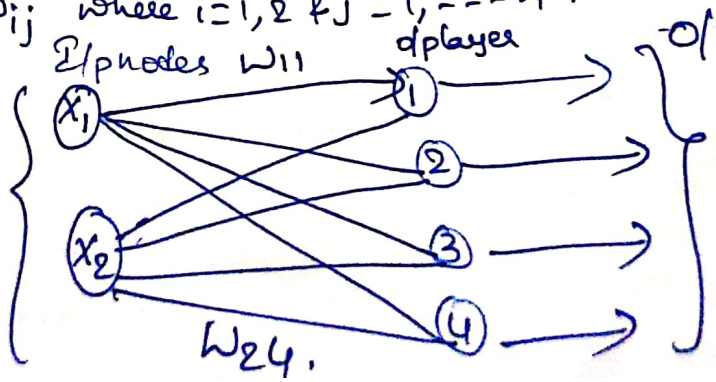
→ I/p to each o/p node is weighted sum of i/p parameters.

→ So, Sum of products of weights & i/p parameters is calculated at each node.

→ If Sum is above certain threshold, the neuron fires and takes the activated value, else it takes the deactivated value.

→ Single - layer - feed - forward n/w with 2 i/p nodes $\{x_1, x_2\}$ & 4 o/p nodes

w_{ij} where $i = 1, 2$ & $j = 1, \ldots 4$.



Perceptron : Neuron model

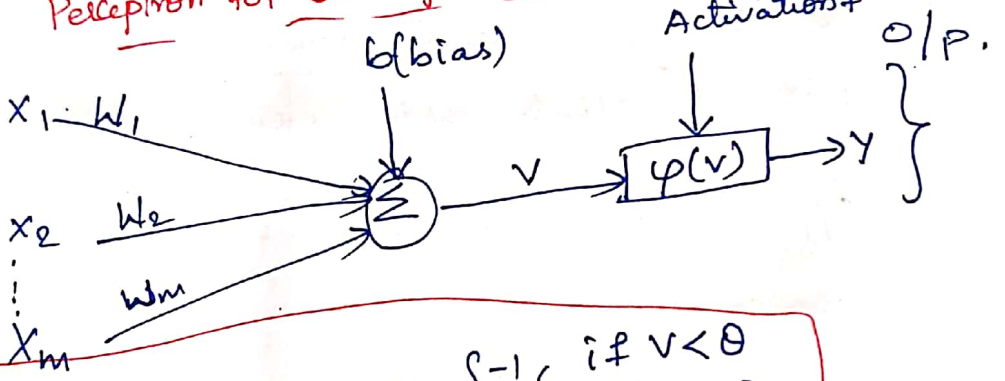
→ is a special form of single-layer feed-forward n/w & was 1st proposed by Rosenblatt in 1958.

→ It is simple neuron model primarily for binary classification i.e., it classifies i/p into one of 2 given categories.

→ It has only one o/p node.

→ Activation f^n used in case of a perceptron is a Step f^n which returns the value 1 if the weighted sum of its i/p is greater than or equal to a defined threshold (θ), else it returns the value −1.

Perceptron for Binary classification



Step f^n used is $\varphi(v) = \begin{cases} -1, & \text{if } v < \theta \\ +1, & \text{if } v \geq \theta. \end{cases}$

→ Before the perceptron is used for binary classification, it needs to be trained.

→ This is achieved by determining appropriate values of weights such that training examples may be classified correctly.

→ If there exists a hyper plane that separates examples of two classes C_1 & C_2, then such classes are called linearly separable classes.

→ Since perceptron is a binary classifier, it can only model linearly separable classes.

→ In the case where classes are not linearly separable, it is desired to find a linear separator that minimizes the mean squared error.

→ Perceptron is trained with training examples of classes C_1 & C_2 in such a way that if its o/p is +1, then i/p is assigned to class C_1 & if its o/p is −1, then i/p is " " class C_2.

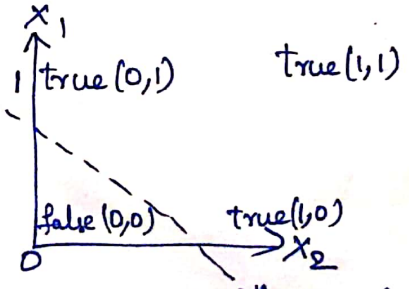
→ Boolean f^{n}s AND, OR, COMPLEMENT are linearly Separable f^{n}s whereas XOR is non-linearly Separable f^n.

→ Linearly Separable f^{n}s are plotted on a 2 dimensional graph, a single straight line can be drawn such that it separates the values in 2 parts.

→ Consider OR f^n. for concept of linearly Seperable Classes.

OR f^n.

Input		output
x_1	x_2	x_1 OR x_2
0	0	0
0	1	1
1	0	1
1	1	1



OR f^n Graph.

Learning Algorithm for Perceptron

→ To ensure perceptron gives correct results for each of its training examples, it is continuously trained. The following algorithmic steps are used in training process:

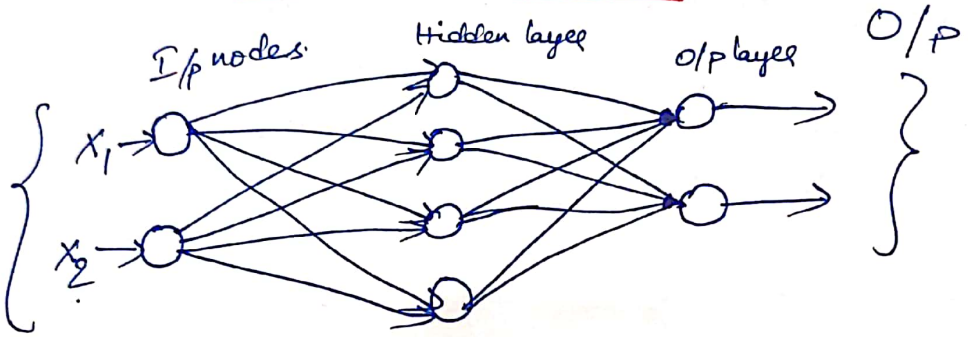
- Initially random weights are assigned to i/p variables in range −0.5 and +0.5
- Training data is presented to perceptron & its o/p is observed.
- If o/p is incorrect, the weights are adjusted accordingly using following formula.

  $W_i = W_i + (A * X_i * E)$, where E is the error produced & $A (0 < A < 1)$ is learning rate.
- The learning rate A may be defined as follows:
  - if o/p is correct, then A = 0
  - if o/p is too low, then A = some +ve no.
  - " " " too high, then A = some −ve no.
- After modification of weights, the next training data is applied with the modified weights & weights are further adjusted accordingly
- Once all training data have been applied, process starts again until all weights are correct & all errors are zero
- Each iteration of this process is known as an epoch

Multilayer Feed-Forward N/ws

- Multilayer feed forward neural N/w used to handle the data that is not linearly separable or not separable by hyper-plane.
- These contain hidden layers b/w i/p & o/p layers, which do not directly receive i/ps or send o/ps to external environment.

Architecture of an FFNN



→ Multi-layer-feed-forward N/ws continously evolve with help of learning techniques.

→ Most crucial learning techniques is known as back-propagation method.

It works:

- The o/p values obtained for a known set of i/ps is compared with correct answer in order to determine the value of some pre-defined error f^n.
- This error is then fed back to the N/w using back propagation method
- Algorithm utilizes this information & then adjusts the weights of all connections so that the corresponding error f^n is reduced to some extent.
- This procedure is repeated for large no. of training cycles, so n/w contains obtains a state where error becomes negligible. We can say the n/w has learned the target f^n.

→ Gradient descent, is applied in case of non-linear classification for proper adjustments of weights.

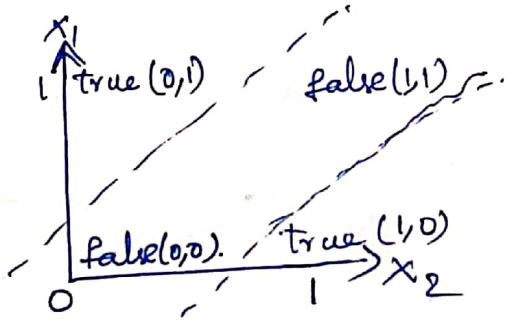
→ The derivative of error f^n with respect to network weights is determined, & weights are then changed accordingly to decrease margin of error.

→ Back-propagation can be applied only those n/ws that have differentiable activation f^ns.

Example of non-linearly Separable f^n XOR f^n.

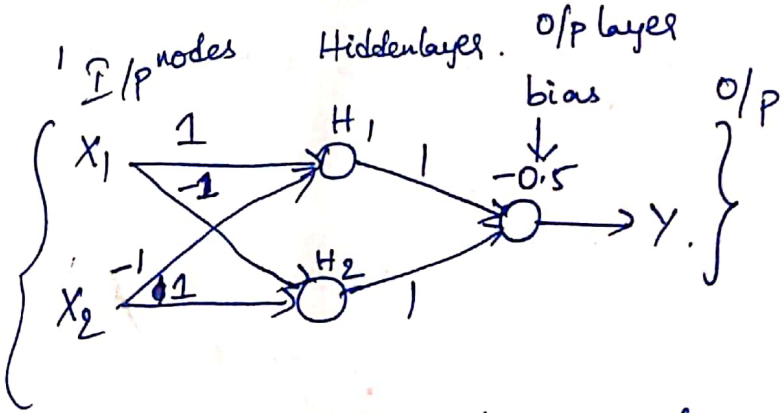
XOR f^n

I/P		O/P
x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0



Graph for XOR f^n.

FFNN for this XOR.



→ Connections from i/p nodes to 2 hidden nodes are weighted with weight vectors: (1, −1) & (−1, 1) indicating weights from (X_1 to H_1, X_1 to H_2) & (X_2 to H_1, X_2 to H_2)

Working of an FFNN for XOR

Inputs		Outputs of Hidden Nodes		Output Node	X_1 XOR X_2
X_1	X_2	H_1	H_2		
0	0	0	0	$-0.5 \rightarrow 0$	0
0	1	$-1 \rightarrow 0$	1	$0.5 \rightarrow 1$	1
1	0	1	$-1 \rightarrow 0$	$0.5 \rightarrow 1$	1
1	1	0	0	$-0.5 \rightarrow 0$	0

Back Propagation Training Algorithm for FFNN

→ Commonly used training algorithm for FFNN is based on gradient descent method.

→ The derivative of error f^n with respect to n/w weights is determined,

Back Propagation Algorithm

→ Using set of training examples (also called training set) it determines those weights for which total error of network is minimum.

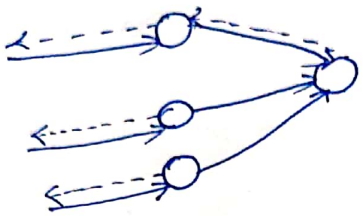
→ Back propagation consists of repeated appl^n of following two passes:

Forward Pass: The network is activated for one example & error of each neuron at o/p layer is computed.

Backward Pass: N/w error is used for updating the weights & and propagated from last node to start nodes.

→ Beginning at o/p layer, error is propagated backwards through the n/w, layer by layer, by recursively computing local gradient of each neuron.

Back propagation Algorithm



→ N/w activation Forward Pass

⇢ Error Propagation Backward Pass.

* Refine the weight values:

- Run a set of i/p variable values through the n/w using a random set of weights.
- Compute the difference b/w the computed & actual target values for this case
- Average the error over entire set of training cases
- Propagate the error backward through the n/w & compute the gradient of change in error with respect to changes in weight values
- Make adjustments to weights to reduce the error
- Each Cycle is called an epoch.

This method is called backward Propagation.

- Consider an example of a n/w with 3 layers.
- i represents nodes in i/p layer, j represents nodes in hidden layer, k represents nodes in o/p layer.

→ w_{ij} refer to weight of connection b/w a node in i/p layer i & a node in hidden layer j.

Y_j of node j:

$$Y_j = \frac{1}{1+e^{-x_i}}, \text{ where } X_i = \sum_{i=1}^{n} X_i * w_{ij} - \theta_j$$

θ_j is threshold for node j & n is no. of i/p to node j.

Algorithm in 457.

Weight Update Rule

→ Weight updates in back-propagation using gradient descent method.

→ The values of weights are adjusted by an amount proportional to 1st derivative of error b/w actual o/p & computed o/p.

→ Weight W_{ji} is updated by.

$$W_{ji} = W_{ji} + \Delta W_{ji}, \text{ where } \Delta W_{ji} = -\eta * [\delta E / \delta W_{ji}],$$

η is learning rate in range [0.1, 0.9]

→ Iteration of back-propagation algorithm is usually terminated when sum of squares of errors of o/p values for all training data in an epoch is less than some pre-defined threshold such as 0.01.

→ Goal is to decrease error fn by reaching global min & avoiding local minima.

→ The weights are updated only after all examples have been processed using the formula.

$$W_{ji} = W_{ji} + \sum^{x} \Delta W_{ji}^{x}$$ x denotes a training example.

Stopping Criterions

- Total mean squared error change — back propagation is considered to have converged when absolute rate of change in avg squared error per epoch is sufficiently small say in range [0.1, 0.01]
- Generalization based criterion

↓

After each epoch FFNN is tested for generalization, if performance is adequate, then process is stopped.

Delta Rule for Error Minimization (least mean square)

→ adjustment of weights in such a manner that difference of error b/w actual & computed o/p is reduced.

→ To minimize the mean squared error, so it is known as least mean square method.

Pg - 458.

$$E_{av} = \sum_{i-1}^{N} E(i)N..$$

Recurrent N/w

→ A FFN represents an acyclic n/w since data can pass from i/p to o/p nodes but not vice versa.

→ Once FFNN is trained, its state gets fixed & doesnot modify when new data is presented to it & it has no memory

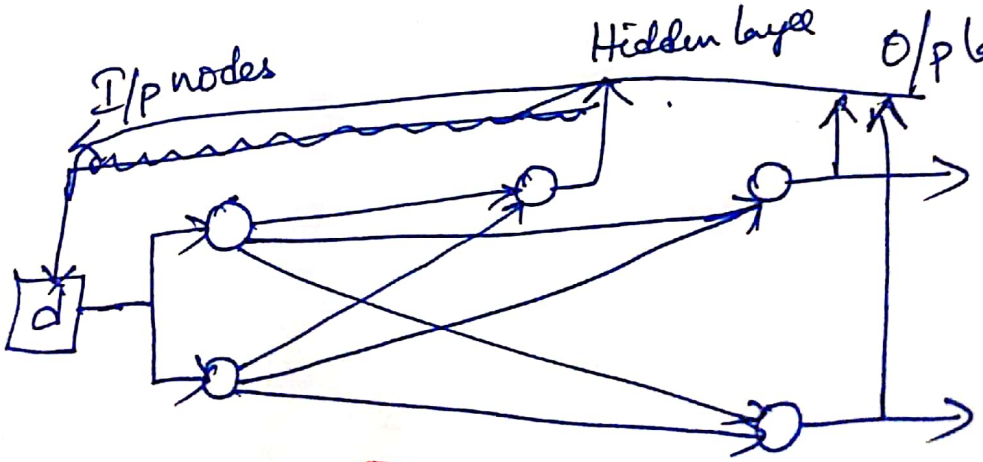
→ These are resolved by recurrent n/w.

→ These n/w can have connections going back from o/p to i/p nodes & infact, can have arbitrary connections between any nodes.

→ Learning in a recurrent n/w involves feeding i/ps through n/w, which includes feeding data back from o/ps to i/ps.

→ So, process of feeding back is repeated until values of o/ps stop changing.

This state is called equilibrium or Stability.



Recurrent N/w.

Support Vector M/cs

→ SVM performs classification of data by constructing an N-dimensional hyper-plane that optimally separates the data pts in to two categories.

→ SVM is also used for regression & is related to Supervised learning method.

→ SVM constructs an (N-1) dimensional hyper-plane in an N-dimensional space that separates data pts in such a manner that there exists max separation margin b/w two data sets. Such an SVM is called linear classifier.

→

Limitations of Perceptron

→ It can only model linear Separable f^ns.

→ When we deal with two classes that are not linearly Separable, we need to obtain a linear Separator that minimizes mean Squared error.

→ ~~Peope~~ Perceptron can't model XOR f^n.