

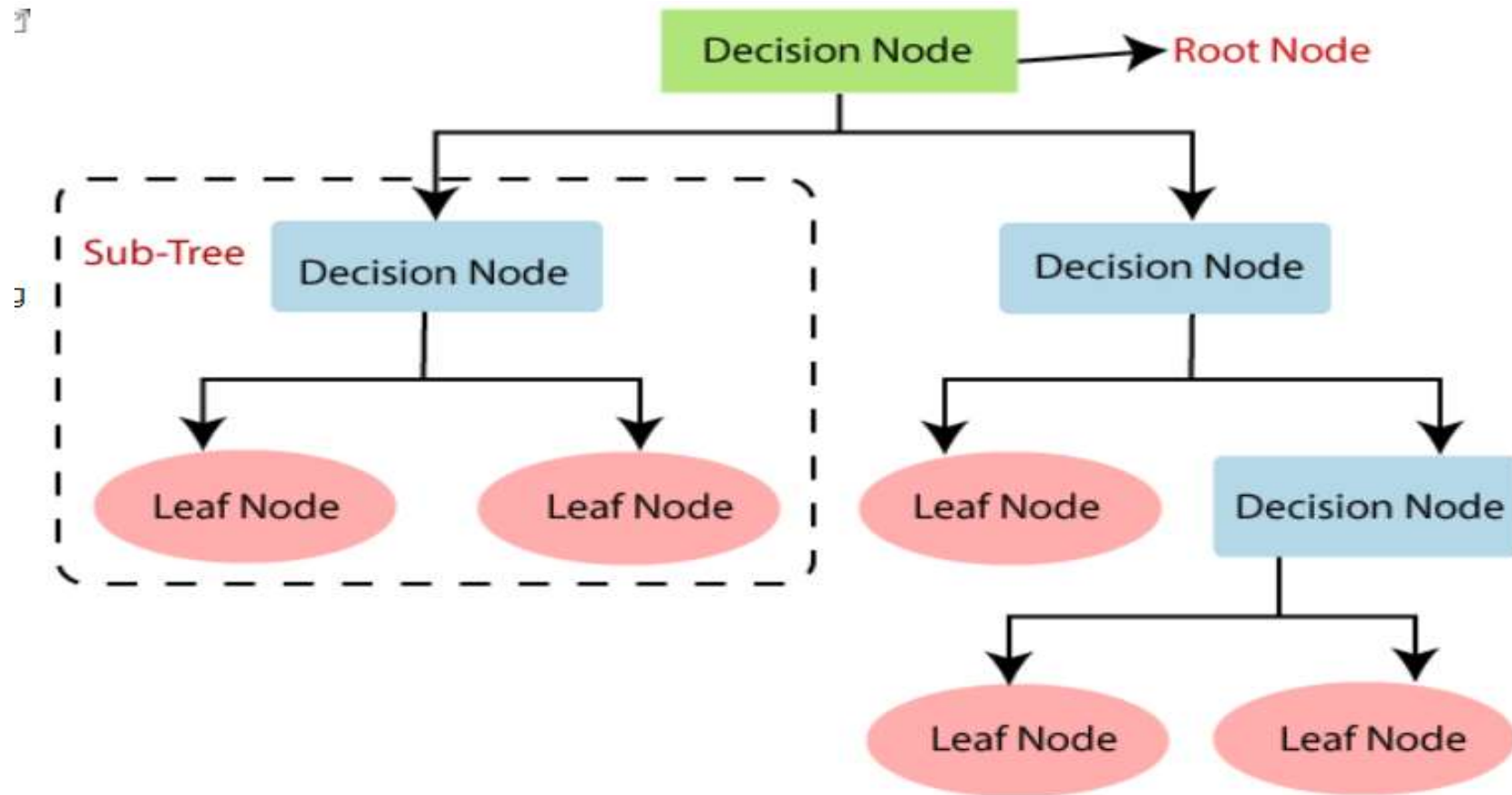
Unit-2

DECISION TREE LEARNING

- A decision tree is a type of supervised learning algorithm that is commonly used in machine learning to model and predict outcomes based on input data. It is a tree-like structure where each internal node tests on attribute, each branch corresponds to attribute value and each leaf node represents the final decision or prediction.
- In a Decision tree, there are two nodes:
- **Decision nodes** are used to make any decision and have multiple branches,
- **Leaf nodes** are the output of those decisions and do not contain any further branches.

- The decisions or the test are performed on the basis of features of the given dataset.
- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.



Why use Decision Trees?

- There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model.
- Below are the two reasons for using the Decision tree:
 - Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
 - The logic behind the decision tree can be easily understood because it shows a tree-like structure.

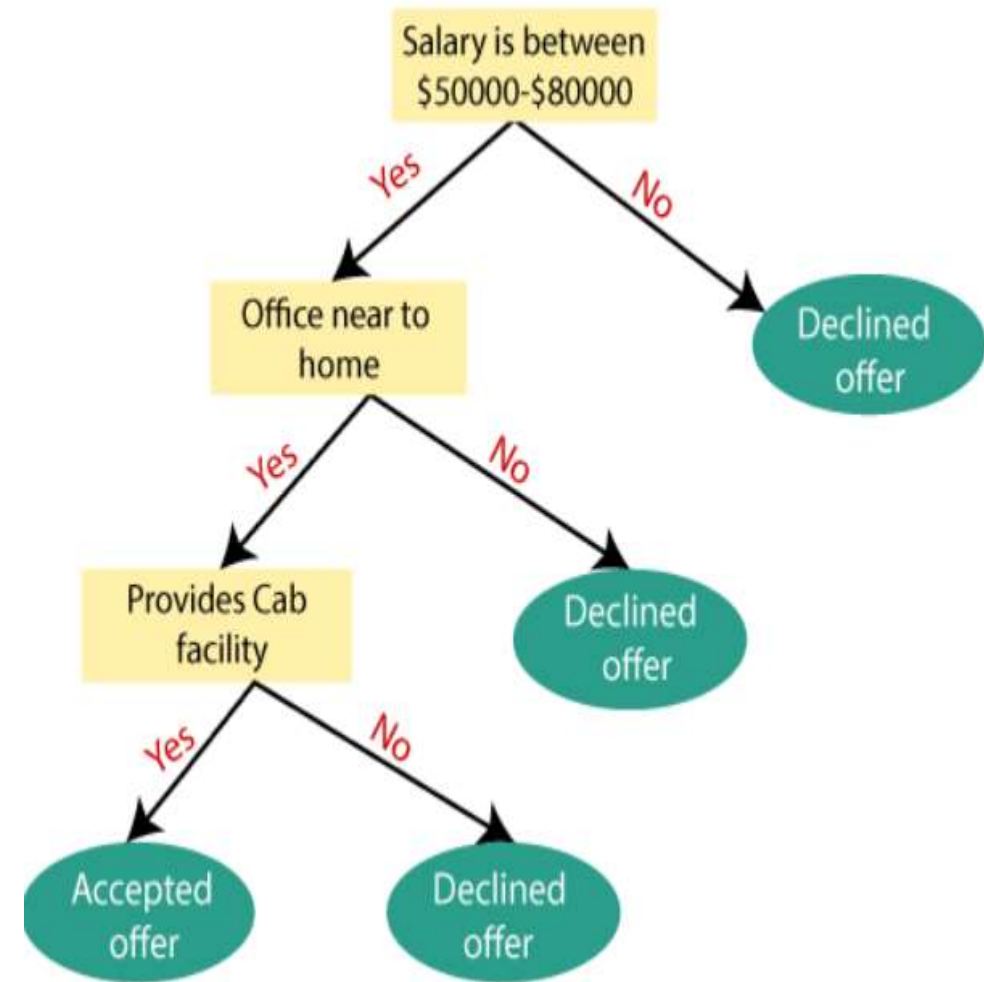
Decision Tree Terminologies

- Root Node:** Root node is from where the decision tree starts. It represents **the entire dataset**, which further gets divided into two or more homogeneous sets.
- Each link represents a decision rule.
- Leaf Node:** Leaf nodes are the **final output node**, and the tree cannot be segregated further after getting a leaf node.
- Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- Branch/Sub Tree:** A tree formed by splitting the tree.
- Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

Decision Tree Steps

- **Step-1:** Begin the tree with the root node, says S , which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM)**.
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

- **Example:** Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM).
- The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node.
- Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures

- While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM**.
- By this measurement, we can easily select the best attribute for the nodes of the tree. There popular technique for ASM, which are:
- Information Gain

Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

- **Disadvantages of the Decision Tree**

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm**.
- For more class labels, the computational complexity of the decision tree may increase.

ID3 [Iterative Dichotomiser3]

- ID3 stands for Iterative Dichotomizer3 and is named such because the algorithm iteratively(repeatedly) dichotomizes(divides) features into two or more groups at each step.
- ID3 is an algorithm invented by Ross Quinlan used to generate a decision tree from a dataset and is the most popular algorithms used to constructing trees.
- ID3 is the core algorithm for building a decision tree. It employs a top-down greedy search through the space of all possible branches with no backtracking.
- This algorithm uses information gain and entropy to construct a classification decision tree.

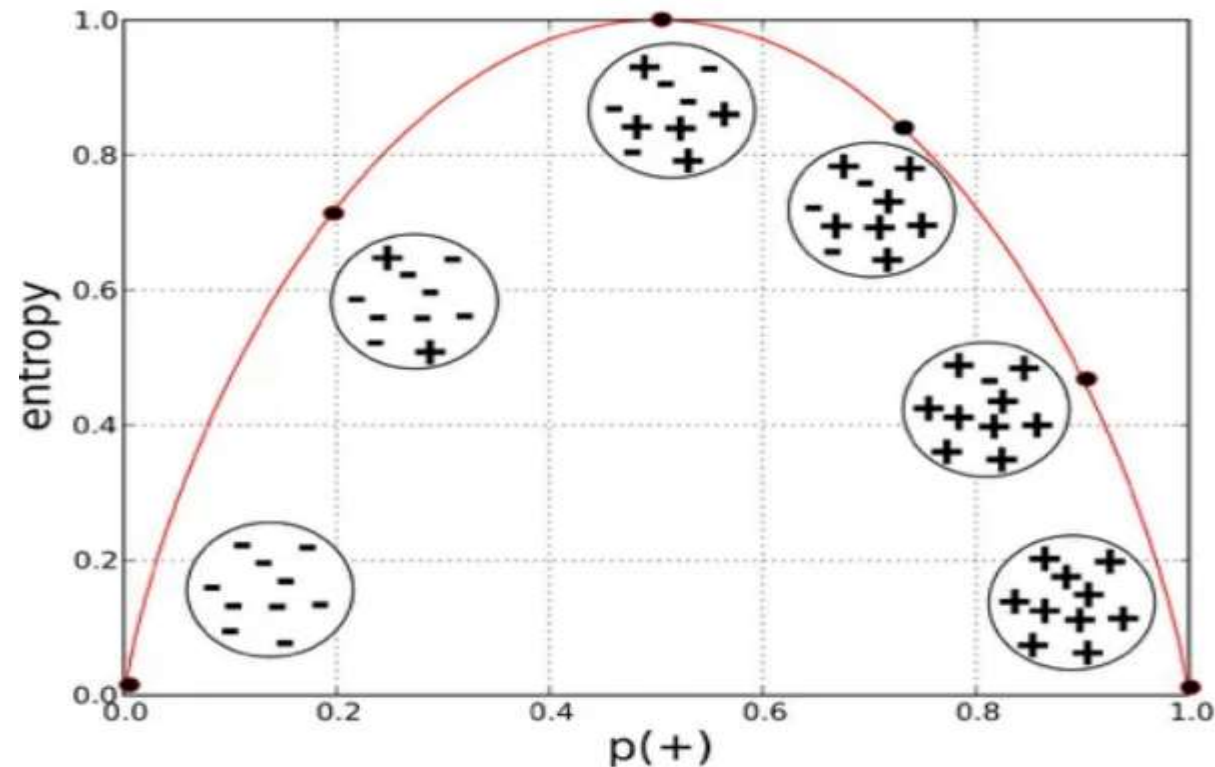
Characteristics of ID3 Algorithm

- ID3 can overfit the training data (to avoid overfitting, smaller decision trees should be preferred over larger ones).
- This algorithm usually produces small trees, but it does not always produce the smallest possible tree.
- ID3 is harder to use on continuous data. i.e. continuous data refers to data that can be measured. This data has values that are not fixed and have an infinite number of possible values.

- **Steps to making Decision Tree**

1. Take the Entire dataset as an input.
2. Calculate the Entropy of the target variable, as well as the predictor attributes.
3. Calculate the information gain of all attributes.
4. Choose the attribute with the highest information gain as the Root Node
5. Repeat the same procedure on every branch until the decision node of each branch is finalized.

- **Entropy**-It is used for checking the impurity or uncertainty present in the data. Entropy is used to evaluate the quality of a split.
- When entropy is zero the sample is completely homogeneous, meaning that each instance belongs to the same class and entropy is one when the sample is equally divided between different classes.



- Entropy = $\sum (P_i + N_i) / (P + N) * IG(P_i * N_i)$

OR

- Entropy = Information Gain * Probability

- **Information Gain-** Information gain indicates how much information a particular feature/ variable give us about the final outcome.

- $IG = [-P / (P + N) \log (P / (P + N))] - [N / (P + N) \log (N / (P + N))]$ with base2.

- Final Gain can be calculated as: $IG - E(A)$

- Example:

Age	Competition	Type	Profit
Old	Yes	Software	Down
Old	No	Software	Down
Old	No	Hardware	Down
Mid	Yes	Software	Down
Mid	Yes	Hardware	Down
Mid	No	Hardware	Up
Mid	No	Software	Up
New	Yes	Software	Up
New	No	Hardware	Up
New	No	Software	Up

- Step-1: Target Attribute → Profit (because it's a decision making attribute)

Step 2: Find the Information Gain of Target Attribute.

- $IG = [-P/(P+N) \log_2 (P/(P+N))] - [N/(P+N) \log_2 (N/(P+N))]$ with base2.

Where, $P = \text{Count}(\text{Down}) = 5$ $N = \text{Count}(\text{Up}) = 5$

IG (Profit) = 1

Step 3: Find the Entropy of each attribute other than Target attribute.

- $\text{Entropy} = \sum (P_i + N_i) / (P + N) * IG(P_i * N_i)$

OR

- $\text{Entropy} = \text{Information Gain} * \text{Probability}$

- i) **Age:** prepare a table for each attribute. Where,
- Rows: represents the value of undertaken attributes.
 - Columns: represents the values of target attribute.

Attributes	Down Value	Up Value
Old	3	0
Mid	2	2
New	0	3

- $\text{Entropy}(\text{Age}) = \text{IG} * \text{Probability}$.
- i.e. we have to find IG of each attribute within Age attribute.

- $IG = [-P/(P+N) \log_2 (P/(P+N))] - [N/(P+N) \log_2 (N/(P+N))]$ with base2.

- $IG(Old) = 0$

- $Entropy(Old) = 0 * (3/10) = 0$

- $IG(Mid) = 1$

- $Entropy(Mid) = 1 * (4/10) = 0.4$

- $IG(New) = 0$

- $Entropy(New) = 0 * (3/10) = 0$

- Entropy (Age) = E(Old) + E(Mid) + E(New)

$$\text{Entropy (Age)} = 0 + 0.4 + 0$$

$$\text{Entropy (Age)} = 0.4$$

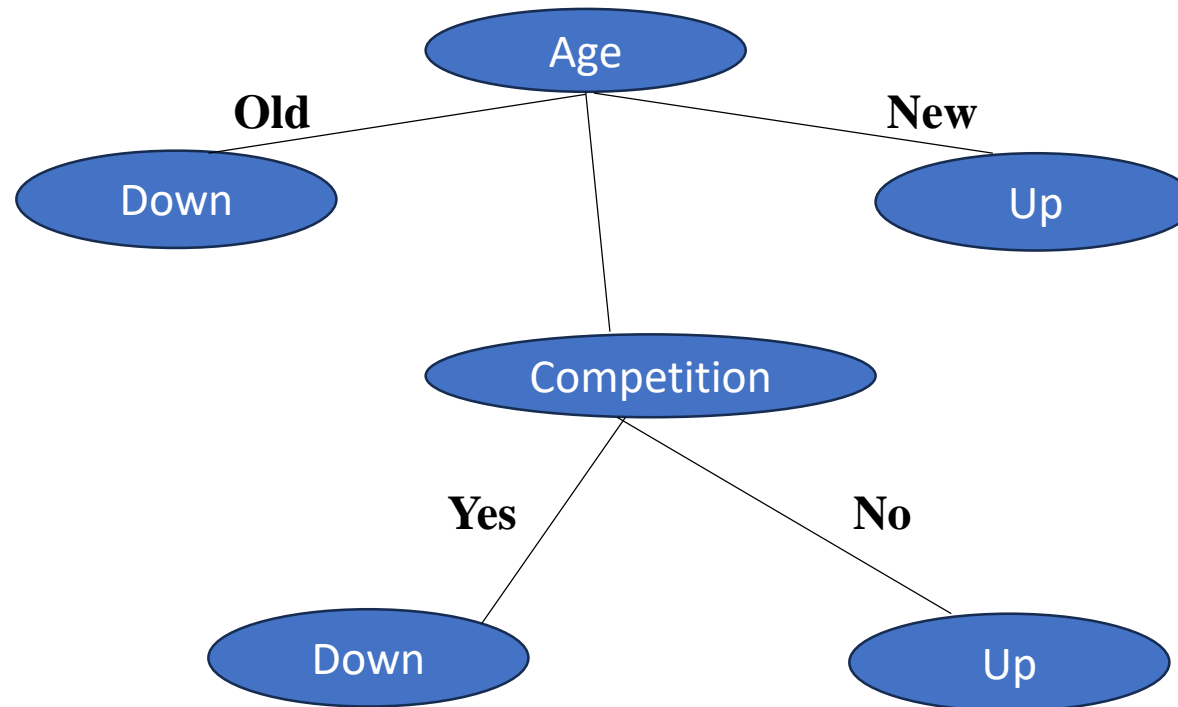
Step 4: Now, calculate the final Gain corresponding to the Target Attribute.

$$\text{IG(Age)} = \text{IG (Target Attribute)} - \text{Entropy (Age)}$$

$$\text{IG(Age)} = 1 - 0.4$$

$$\text{Information Gain (Age)} = 0.6$$

- Similarly we have to find $IG(\text{Type})$ and $IG(\text{Competition})$.
- **Information Gain (Age) = 0.6**
- **Information Gain (Type) = 0**
- **Information Gain (Competition) = 0.124**
- **The attribute having highest IG will be the root node in Decision Tree Learning.**



Q. Calculate data set entropy and information gain.

DAY	Outlook	Temperature	Humidity	Sun light	Play cricket
D1	Rainy	Hot	High	Weak	No
D2	Rainy	Hot	High	strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Sunny	Mild	High	Weak	Yes
D5	Sunny	Cool	Normal	Weak	Yes

Appropriate problems for DTL

DT algorithm best suited to problems with the following characteristics:

1. Instances are represented by attributes value pairs.
2. The target function has discrete output value pairs.
3. Training data can have errors.
4. May contain missing attributes values also.

Issues in DTL

1. Determining how deeply to grow the decision tree.
2. Handling continuous attributes.
3. Choosing an appropriate attribute selection measure.
4. Handling training data with missing attribute values.
5. Handling attributes with differing costs.
6. Improving computational efficiency.

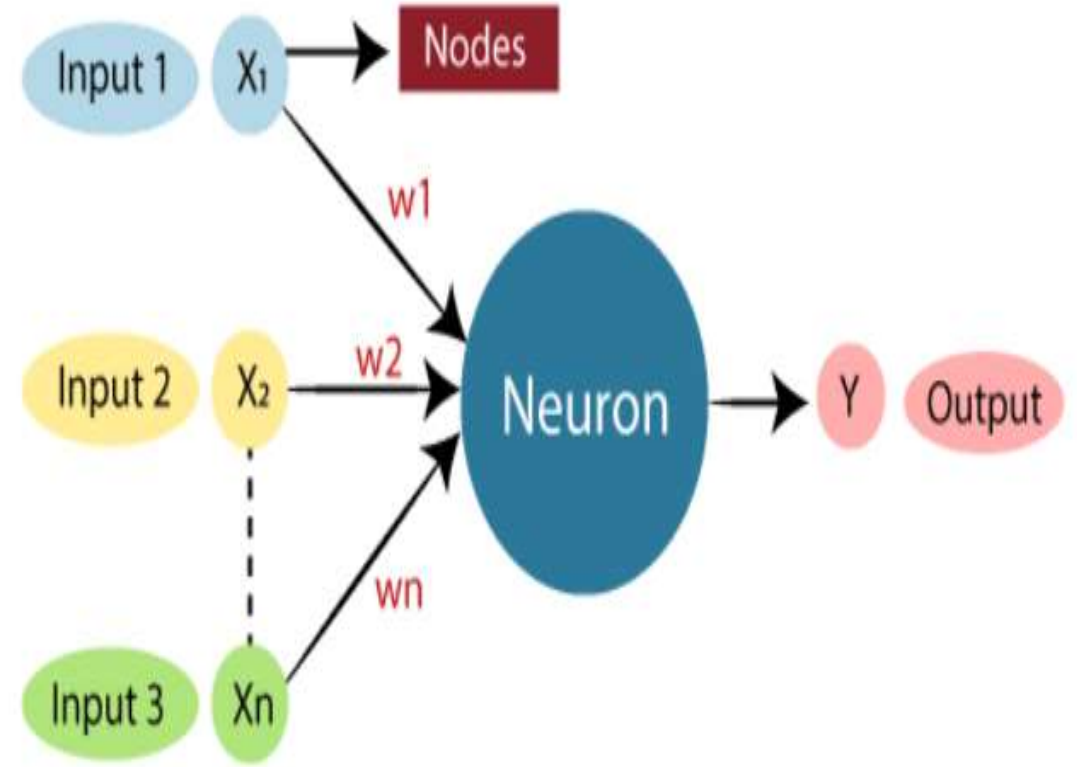
Inductive Bias in DTL

- It consists of describing the basis of which ID3 chooses i.e. 1 consistent decision tree.
- Selection strategy:
 1. Select in favor of shorten trees over longer ones.
 2. Select element with highest IG as root attribute over lowest IG.
- Types of Inductive Bias:
 1. Restrictive Bias: based on conditions. example candidate elimination algo
 2. Preference Bias: based on priorities. example. ID3

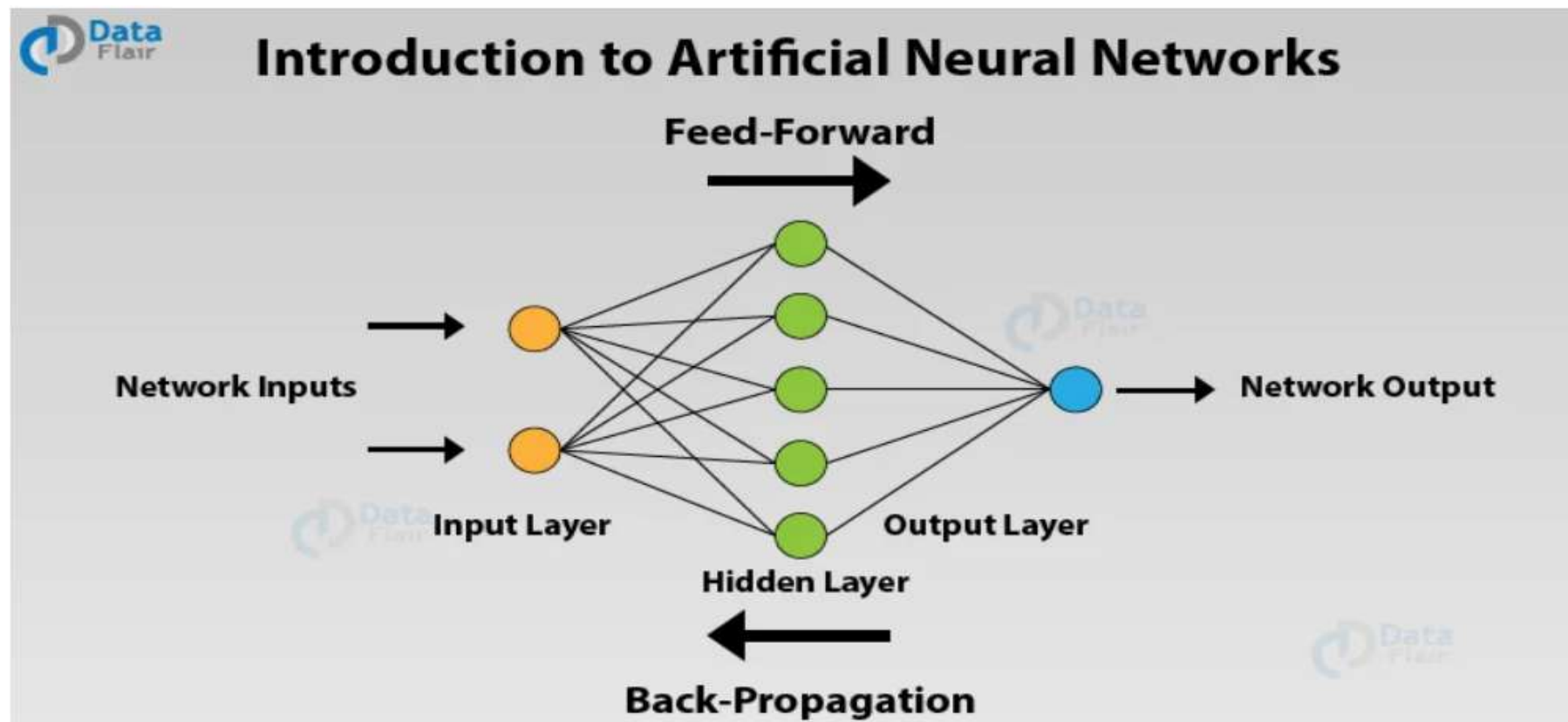
Artificial Neural Network

- An Artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain.
- Similar to a human brain has neurons interconnected to each other, artificial neural networks also have neurons that are linked to each other in various layers of the networks.
- These neurons are known as nodes.

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

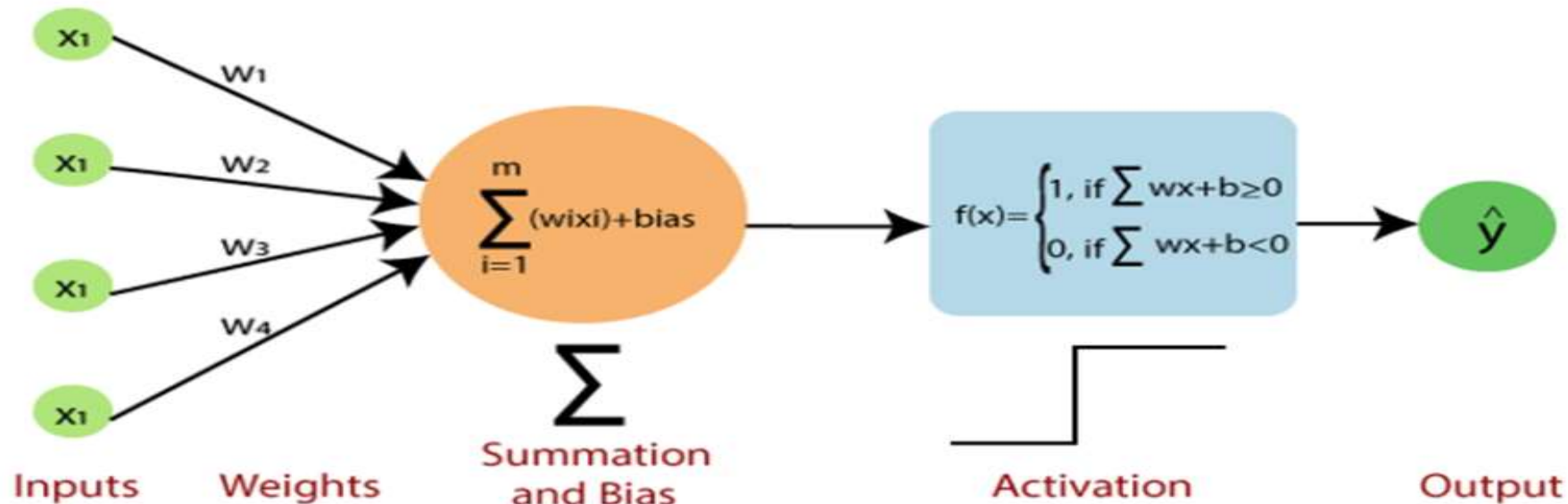


- **ANNs are nonlinear statistical models** which display a complex relationship between the inputs and outputs to discover a new pattern.
- A variety of tasks such as image recognition, speech recognition, machine translation as well as medical diagnosis makes use of these artificial neural networks.



- In a neural network, there are multiple parameters and hyperparameters that affect the performance of the model. The output of ANNs is mostly dependent on these parameters.
- Some of these parameters are weights, biases, learning rate, batch size etc. Each node in the ANN has some weight.
- Each node in the network has some weights assigned to it. A transfer function is used for calculating the weighted sum of the inputs and the bias.
- After the transfer function has calculated the sum, **the activation function obtains the result.** Based on the output received, the activation functions fire the appropriate result from the node. For example, if the output received is above 0.5, the activation function fires a 1 otherwise it remains 0.

- Based on the value that the node has fired, we obtain the final output. Then, using the error functions, we calculate the discrepancies between the predicted output and resulting output and adjust the weights of the neural network through a process known as *backpropagation*.

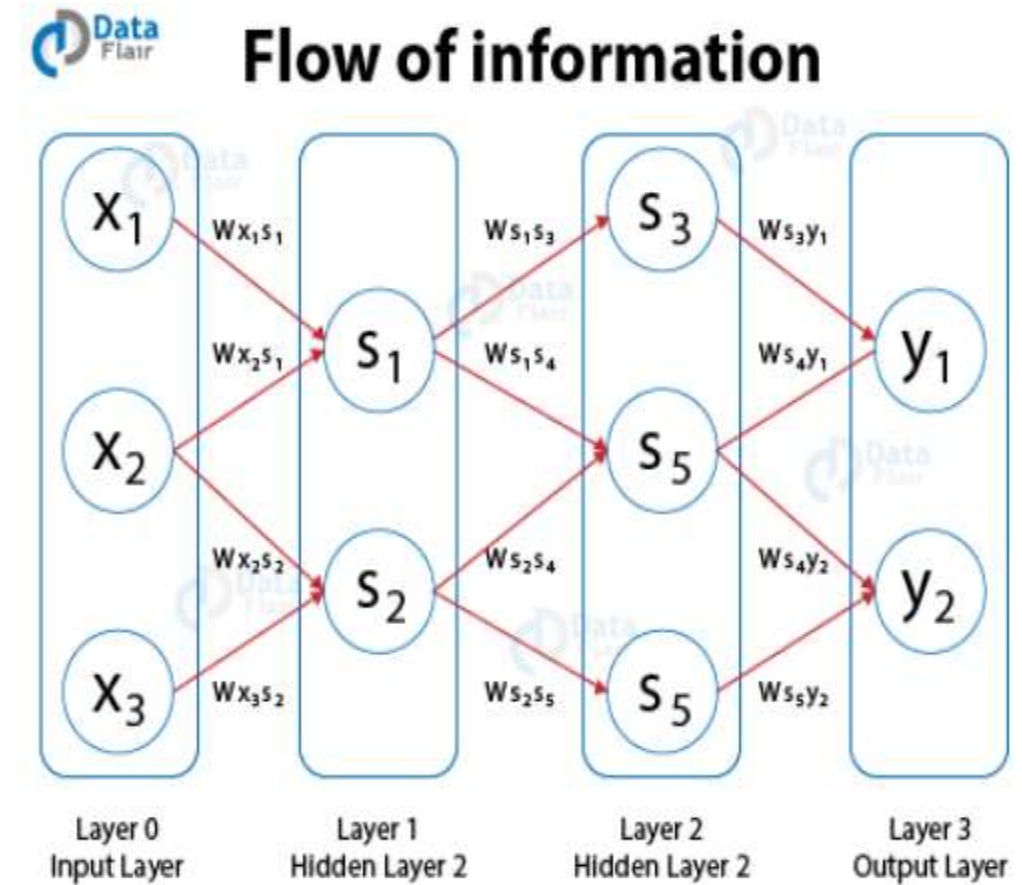


- **Perceptron** are a type of artificial neural network that can be used for classification and regression.
- They are supervised learning algorithms, meaning they need labeled input data in order to learn. how to map inputs to outputs.
- Perceptrons require at least one input and one output.
- They only learn linear functions, they can't learn nonlinear functions, they can't be connected with each other (unless using something like the backprop algorithm or dynamic programming), and they require labeled input data.

- Types of Artificial Neural Networks

1. FeedForward Artificial Neural Networks

- **In the feedforward ANNs, the flow of information takes place only in one direction.** That is, the flow of information is from the input layer to the hidden layer and finally to the output.
- There are no feedback loops present in this neural network.
- These type of neural networks are mostly used in *supervised learning* for instances such as classification, image recognition etc.
- Can be used in cases where the data is not sequential in nature.



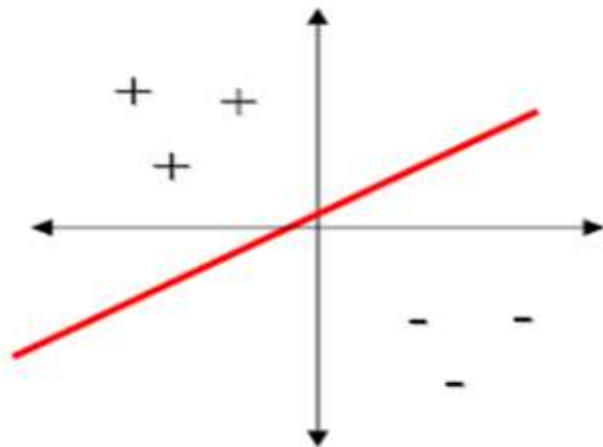
2. Feedback Artificial Neural Networks

- In the feedback ANNs, the feedback loops are a part of it.
- Such type of neural networks are mainly for memory retention such as in the case of recurrent neural networks.
- These types of networks are most suited for areas where the data is sequential or time-dependent.

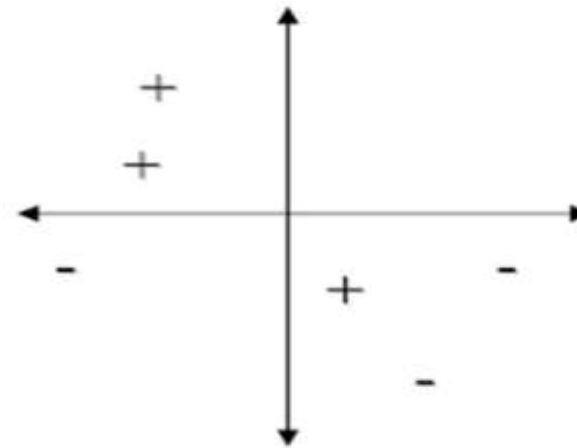
Multilayer Artificial Neural Network

- To be accurate a fully connected Multi-Layered Neural Network is known as Multi-Layer Perceptron.
- A Multi-Layered Neural Network consists of multiple layers of artificial neurons or nodes. Unlike Single-Layer Neural networks.
- A Multi-layered Neural Network is a typical example of the Feed Forward Neural Network.
- The number of neurons and the number of layers consists of the hyperparameters of Neural Networks which need tuning.
- Using the Back-Propagation technique, weight adjustment training is carried out.

- The perceptron rule is applicable when the training examples are linearly separable, it fail to converge if the examples are not linearly separable.
- **The delta rule**, is designed to overcome this difficulty. It converges toward a best-fit approximation to the target concept.
- A set of data points are said to be **linearly separable if the data can be divided into two classes using a straight line**. If the data is not divided into two classes using a straight line, such data points are said to be called non-linearly separable data.



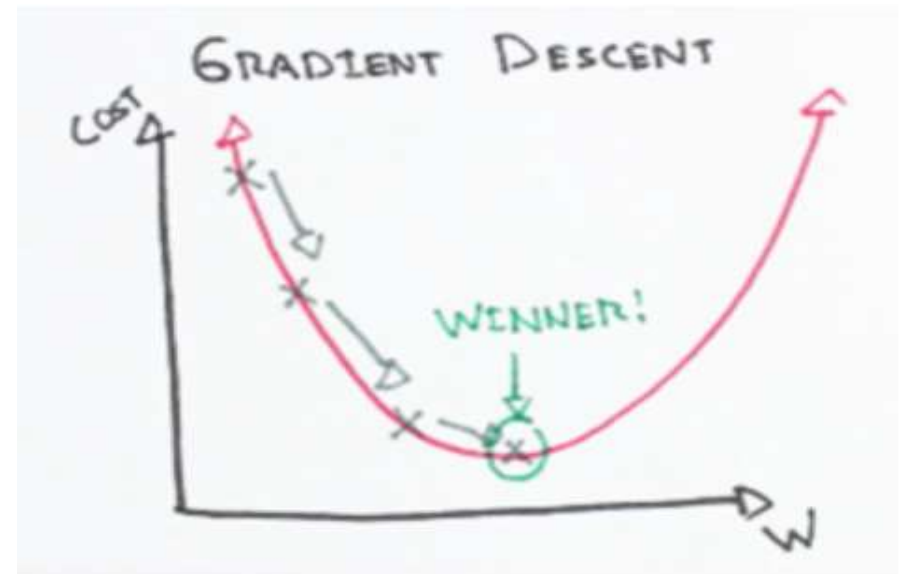
Linearly separable



Non-linearly separable

Delta Rule and Gradient Descent

- The key idea behind the delta rule is to use gradient descent to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.
- This rule is important because gradient descent provides the basis for the BACKPROPAGATION algorithm, which can learn networks with many interconnected units.
- **A gradient simply measures the change in all weights with regard to the change in error.**



Derivation of Delta Rule

- Considering the task of training an thresholder perceptron; that is, a linear unit for which the output o is given by:

$$o = w_0 + w_1x_1 + \cdots + w_nx_n$$
$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

- Let's start by providing a measure for the training error of a hypothesis(weight vector) relative to the training instances in order to build a weight learning algorithm for linear units.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- Where D is the set of training examples, 'td' is the target output for training example 'd', and o_d is the output of the linear unit for training example 'd'.
- The direction of steepest descent along the error surface can be found by computing the derivative of E with respect to each component of the vector w . This vector derivative is called the gradient of E with respect to w , written as:

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

- The gradient specifies the direction of steepest increase of E , the training rule for gradient descent is:

$$\vec{w} \leftarrow \vec{w} + \Delta \vec{w}$$

where

$$\Delta \vec{w} = -\eta \nabla E(\vec{w})$$

- Here η is a positive constant called the learning rate, which determines the step size in the gradient descent search.
- The negative sign is present because we want to move the weight vector in the direction that decreases E . Also be written as:

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

For training example d , x_{id} signifies the single input component x_i .

Finally,

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\ \frac{\partial E}{\partial w_i} &= \sum_{d \in D} (t_d - o_d) (-x_{id})\end{aligned}$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

$$w_i \leftarrow w_i + \Delta w_i$$

GRADIENT-DESCENT(*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where \vec{x} is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
 - Initialize each Δw_i to zero.
 - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
 - Input the instance \vec{x} to the unit and compute the output o
 - For each linear unit weight w_i , Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i \quad (\text{T4.1})$$

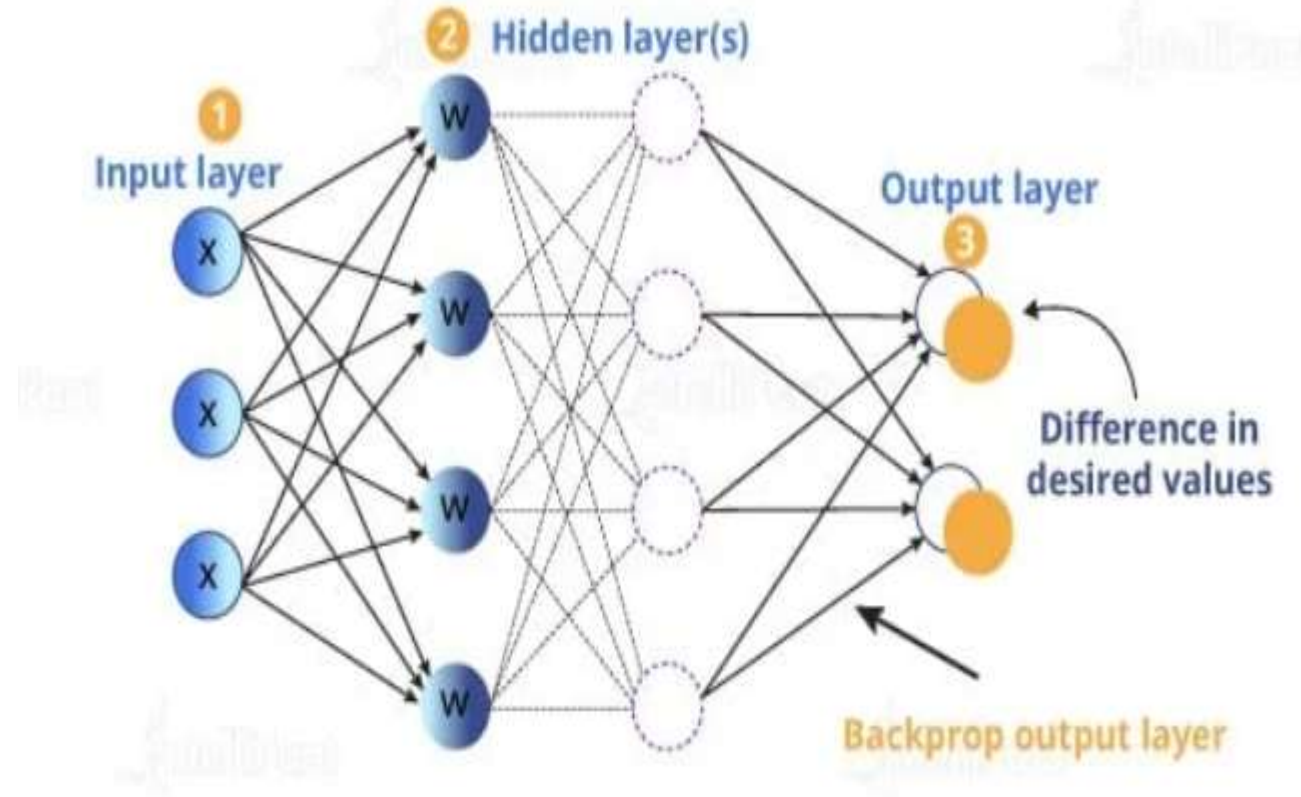
- For each linear unit weight w_i , Do

$$w_i \leftarrow w_i + \Delta w_i \quad (\text{T4.2})$$

- By adding to each weight W_i , you may update it and then repeat the procedure.

Backpropagation Algorithm in Neural Network

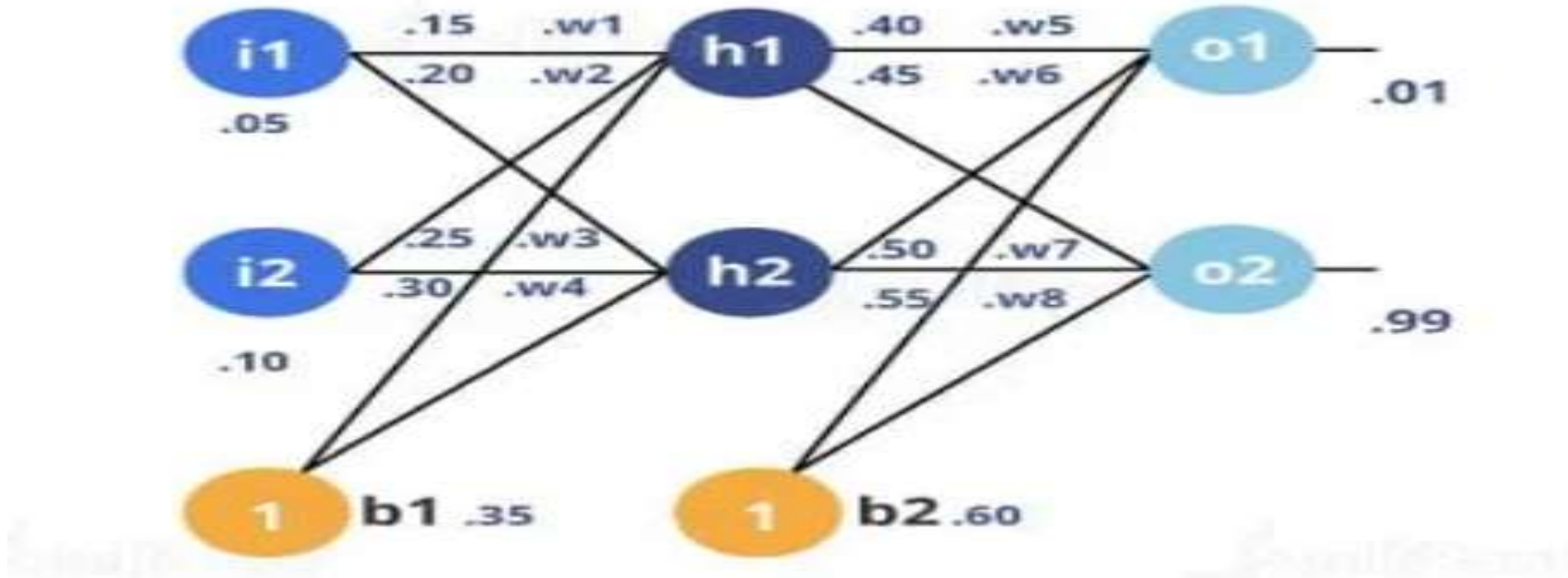
- The goal of the backpropagation algorithm is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.



Steps to be followed

1. Inputs X , arrive through the preconnected path
2. Input is modeled using real weights W . The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs

Step 1: The Forward Pass:



Inputs(**i1**): 0.05

Output (**o1**): 0.01

Inputs(**i2**): 0.10

Output(**o2**):0.99

- 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.

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

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

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

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

- Carrying out the same process for h2:

$$h2(in) = 0.3925$$

$$h2(out) = 0.5968$$

The output for o1 is:

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

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

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

- Carrying out the same process for O2:

$$O2(in) = 1.22484$$

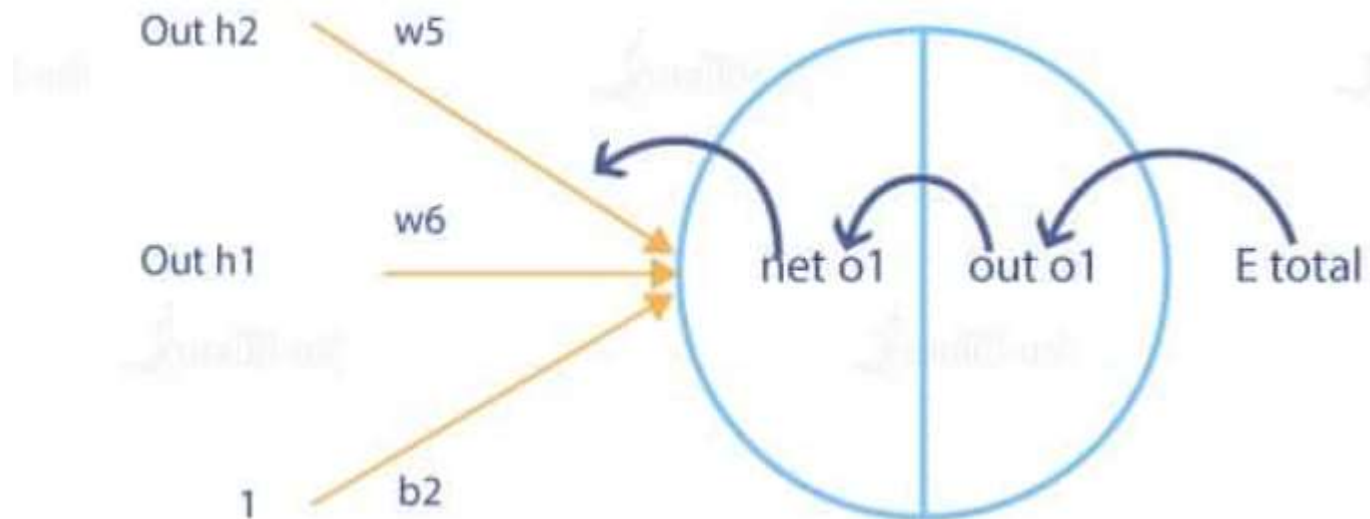
$$O2(out) = 0.7729 \text{ // Actually it must be 0.99 as target}$$

- **Calculating the Total Error:**

- The target output for o1 is 0.01, but the neural network output is 0.75136507; therefore, its error is:
- $E(o1) = 1/2(\text{target } o1 - \text{out } o1)^2 = 1/2(0.01 - 0.75136507)^2 = 0.27481108 \dots (\text{Equation 5})$
- $E(o2) = 0.023560026$
- $E(\text{total}) = E(o1) + E(o2) = 0.274811083 + 0.023560026 = 0.298371109$

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.



Step 2 Calculating Backward Propagation error
(Output layer \rightarrow hidden layer)

i.e. W_5, W_6, W_7 and W_8

Consider W_5 : calculate the rate of change of error w.r.t the change in the weight W_5

$$W_5^* = W_5 - \eta \frac{\partial E_{\text{total}}}{\partial W_5}$$

$$\boxed{\eta = 0.5}$$

$$\boxed{\text{net } O_1 = 0.1}$$

$$\frac{\partial E_{\text{total}}}{\partial W_5} = \underbrace{\frac{\partial E_{\text{total}}}{\partial \text{out } O_1}}_{(i)} * \underbrace{\frac{\partial \text{out } O_1}{\partial \text{net } O_1}}_{(ii)} * \underbrace{\frac{\partial \text{net } O_1}{\partial W_5}}_{(iii)}$$

$$\begin{aligned} (i) \quad \frac{\partial E_{\text{total}}}{\partial \text{out } O_1} &= \text{out } O_1 - \text{target } O_1 \\ &= 0.751365 - 0.01 \\ &= 0.74136507 \end{aligned}$$

$$\begin{aligned} (ii) \quad \frac{\partial \text{out } O_1}{\partial \text{net } O_1} &= \text{out } O_1 (1 - \text{out } O_1) \\ &= 0.75136507 (1 - 0.75136507) \\ &= 0.186815602 \end{aligned}$$

$$(iii) \quad \frac{\partial \text{net } O_1}{\partial W_5} = \text{out } h_1 = 0.593269992$$

$$\begin{aligned}\partial E_{\text{total}} &= 0.413565 * 0.186815602 * 0.59326992 \\ &= 0.08216704\end{aligned}$$

$$W_5^* = W_5 - \eta \frac{\partial E_{\text{total}}}{\partial W_5}$$

$$= 0.4 - 0.5 * 0.082167041$$

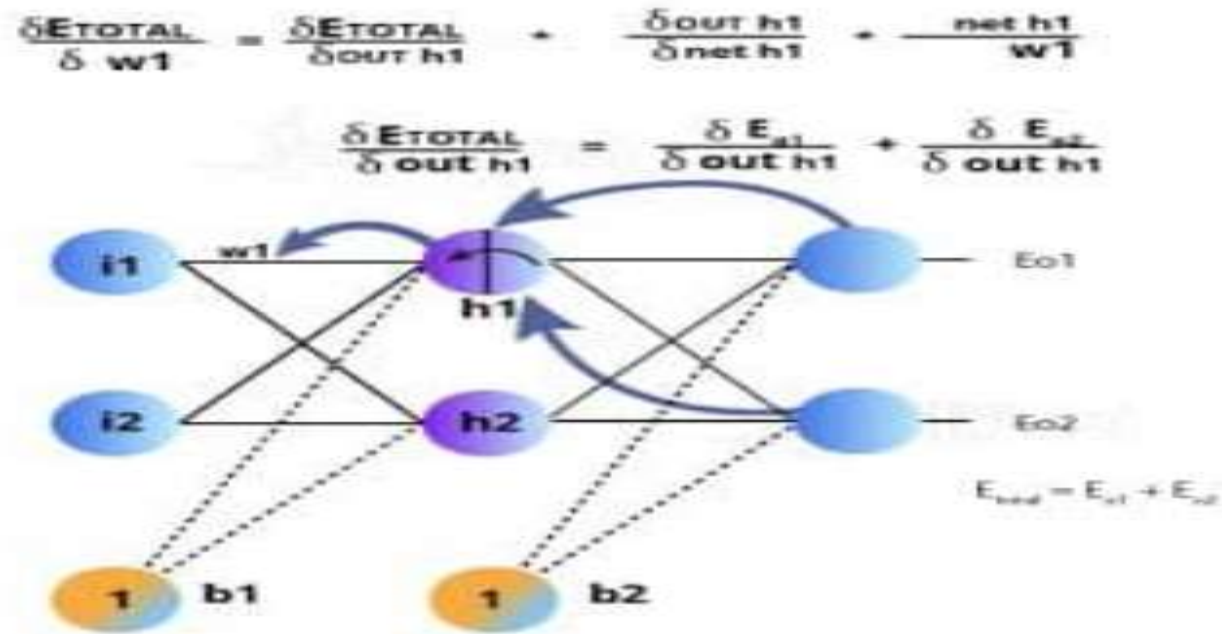
$$W_5 = 0.35891648 \text{ (This is gradient descent)}$$

$$W_6^* = 0.408666186$$

$$W_7^* = 0.511301270$$

$$W_8^* = 0.561370121$$

- Continue the backward pass by calculating new values for w1, w2, w3, and w4:



step 3. Calculating Backward propagation of error
(hidden \rightarrow input layer)
(w_1, w_2, w_3, w_4)

Consider w_1 :

$$w_1^* = w_1 - \eta \frac{\partial E_{total}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1}$$

where,

$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{O1}}{\partial out_{h_1}} + \frac{\partial E_{O2}}{\partial out_{h_1}}$$

$$\begin{aligned}
 \frac{\partial E_{O1}}{\partial \text{outh}_1} &= \frac{\partial E_{O1}}{\partial \text{net} O_1} \times \frac{\partial \text{net} O_1}{\partial \text{outh}_1} \\
 &= \left[\frac{\partial E_{O1}}{\partial \text{out} O_1} \times \frac{\partial \text{out} O_1}{\partial \text{net} O_1} \right] \times \frac{\partial \text{net} O_1}{\partial \text{outh}_1} \\
 &= 0.138498562 \times 0.40 \\
 &= 0.055399425
 \end{aligned}$$

$$\begin{aligned}
 \frac{\partial E_{O2}}{\partial \text{outh}_1} &= \frac{\partial E_{O2}}{\partial \text{net} O_2} \times \frac{\partial \text{net} O_2}{\partial \text{outh}_1} \\
 &= \left[\frac{\partial E_{O2}}{\partial \text{out} O_2} \times \frac{\partial \text{out} O_2}{\partial \text{net} O_2} \right] \times \frac{\partial \text{net} O_2}{\partial \text{outh}_1} \\
 &= -0.019049119
 \end{aligned}$$

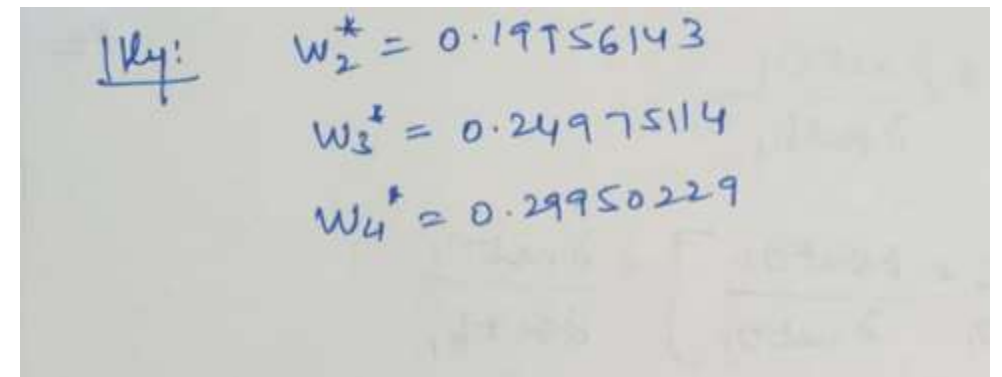
$$\begin{aligned}
 \therefore \frac{\partial E_{\text{total}}}{\partial \text{outh}_1} &= 0.055399425 + (-0.019049119) \\
 &= 0.036350306
 \end{aligned}$$

Putting in eq.ⁿ

$$\begin{aligned}
 \frac{\partial E_{\text{total}}}{\partial w_1} &= 0.036350306 \times 0.241380707 \times 0.0.5 \\
 &= 0.000438568
 \end{aligned}$$

$$\therefore w_1^k = w_1 - \eta \frac{\partial E_{\text{total}}}{\partial w_1} = 0.15 - 0.5 \times 0.000438568 = 0.149780716$$

- When we originally fed forward 0.05 and 0.1 inputs, the error on the network was 0.298371109.
- After the first round of backpropagation, the total error is now down to 0.291027924.

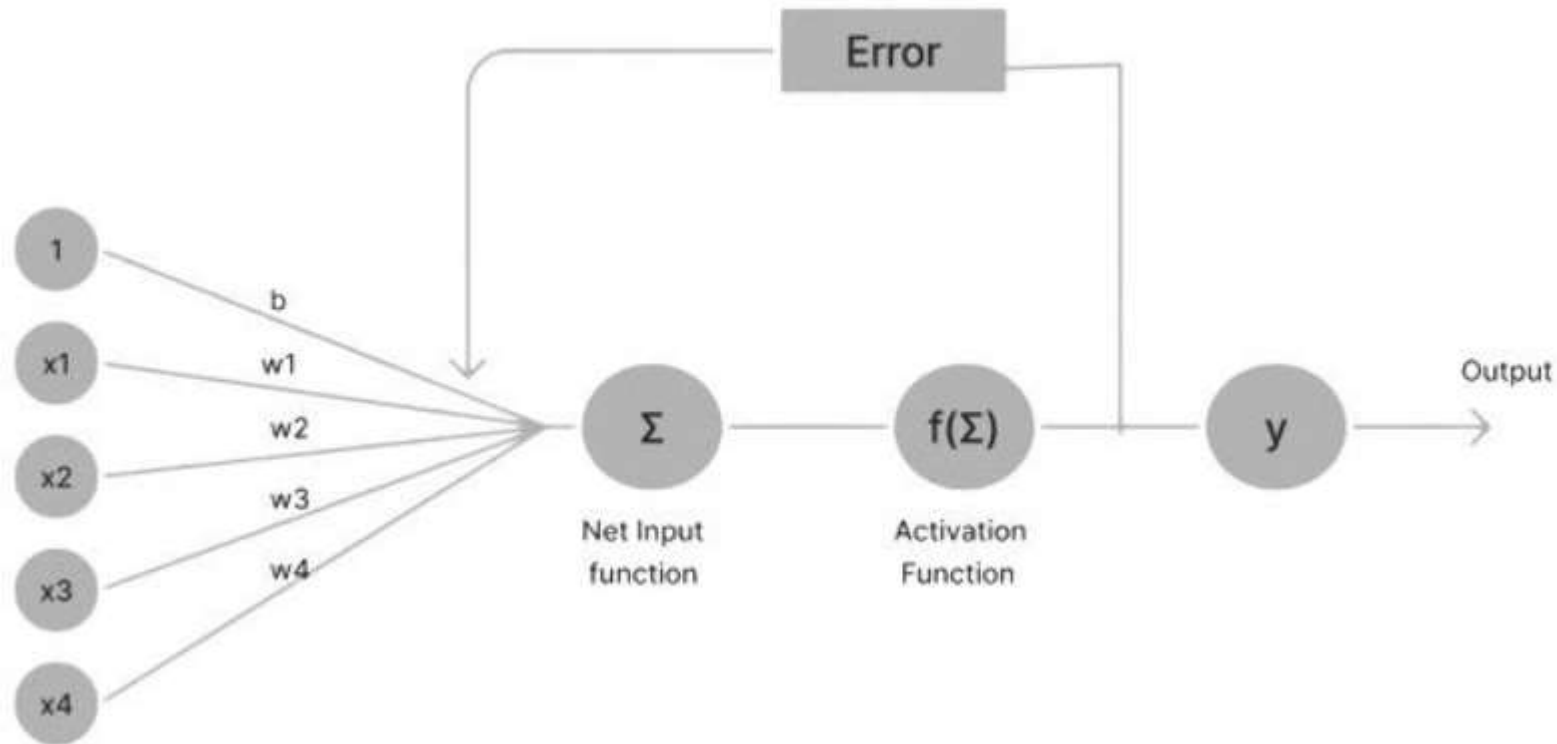


Handwritten notes showing updated weights:

$$\begin{aligned} W_2^* &= 0.19756143 \\ W_3^* &= 0.24975114 \\ W_4^* &= 0.29950229 \end{aligned}$$

- After repeating this process 10,000 times, for example, the error plummets to 0.0000351085. At this point, when we feedforward 0.05 and 0.1, the two output neurons will generate 0.015912196 (vs. 0.01 target) and 0.984065734 (vs. 0.99 target).

- **Adaline (Adaptive Linear Neural) :**
- A network with a single linear unit is called Adaline (Adaptive Linear Neural). A unit with a linear activation function is called a linear unit.
- In Adaline, there is only one output unit and output values are bipolar (+1,-1). Weights between the input unit and output unit are adjustable. It uses the delta rule.



- **Madaline (Multiple Adaptive Linear Neuron) :**
- The Madaline(supervised Learning) model consists of many Adaline in parallel with a single output unit.
- The Adaline layer is present between the input layer and the Madaline layer hence Adaline layer is a hidden layer.
- The weights between the input layer and the hidden layer are adjusted, and the weight between the hidden layer and the output layer is fixed.
- Adaline and Madaline layer neurons have a bias of '1' connected to them.

