Decision trees

Classification is the process of dividing the dataset into different categories or groups by adding label.

Eg:- classifying spam or not spam mail.

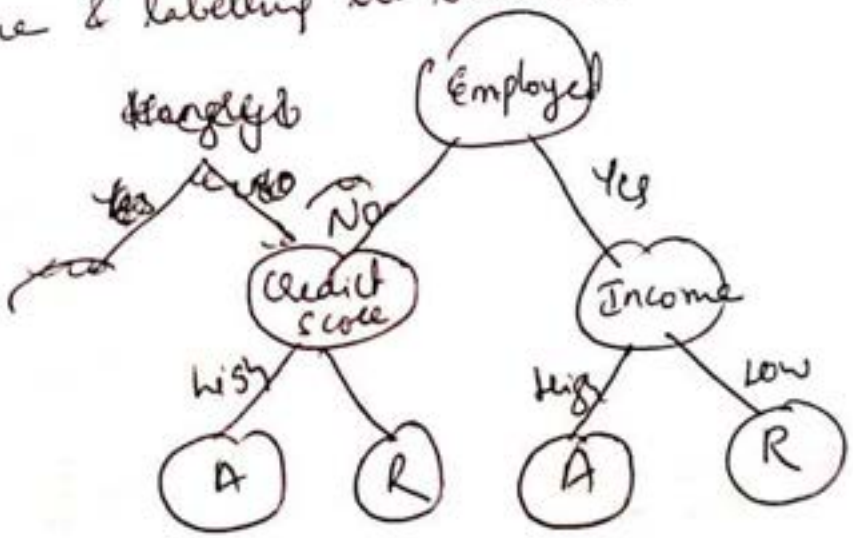
There are different types of classification Algms

1) Decision tree
2) Naive Bayes classifier
3) KNN Classifier
4) Random forest.

A decision tree is a graphical representation of all possible solutions to a decision based on certain conditions.

A decision tree is a hierarchical model for supervised learning. A decision tree is composed of internal decision nodes & terminal leaves. Each decision node m implements a test function with discrete outcome & labelling the branches.

Eg:-
To Sanction loan or not



Decision tree is a tree where each node represents a feature (attribute), each link (branch) represents a decision rule and each leaf represent an outcome.

Given a dataset and we have to make a decision tree that predicts whether Eg:- tennis will be played on the day?

Step1:- Create a root node

How to Choose the root node?

The Attribute that best classifies the training data, use this attribute at the root of the tree.

Step2 How to choose the best attribute

Use the ID_3 Algm

It has two Important concepts.

(a) Entropy:- which calculates amount of uncertainty in dataset [It means number of positive and number of negative examples].

Note:- If we have equal number of positive and negative examples then Entropy=1

If we have only positive example or only negative example then entropy=0

$$\text{Entropy} = \frac{-P}{P+N}\log_2\left(\frac{P}{P+N}\right) - \frac{N}{P+N}\log_2\left(\frac{N}{P+N}\right).$$

Average Information Entropy

$$I(\text{Avg Information Entropy})(\text{attribute}) = \sum \frac{P_i + N_i}{P+N} \text{Entropy}(A)$$

Information gain :- Difference in entropy before & after splitting dataset on attribute A.

$$\text{Gain} = \text{Entropy}(S) - I(\text{attribute}).$$

good quantitative measure that measures how well given attribute seperates the training examples according to their target classification.

Steps:

1) Compute the entropy for dataset Entropy(S)
2) for every attribute/feature
   - (a) Calculate entropy for all other values
   - (b) Take average Information Entropy for current attribute
   - (c) Calculate gain for current attribute.
   - (d) Pick the highest gain attribute.
   - (e) Repeat steps (b)(c)(d) until we get the tree we desired.

ed on the attributes (features) given whether we get profit in the company or not (up or down)

	Age	Competation	Type	profit
1)	Old	Yes	Software	down
2)	Old	NO	Software	down
3)	old	No	Hardware	down
4)	Mild	Yes	Software	down
5)	Mild	Yes	Hardware	down
6)	Mild	NO	Hardware	up
7)	Mild	No	Software	up
8)	New	Yes	Software	up
9)	New	NO	Hardware	up
10)	New	No	Software	up

No of Positive examples → Profit goes up P=5
No of Negative examples → Profit goes down N=5

Entropy of class

$$\text{Entropy}(S) = \frac{-P}{P+N}\log_2\left(\frac{P}{P+N}\right) - \frac{N}{N+N}\log_2\left(\frac{N}{P+N}\right)$$

$$= \frac{-5}{10}\log_2\left(\frac{5}{10}\right) - \frac{5}{10}\log_2(5/10) = 1$$

To find which attribute form the dataset becomes the root node find

for each attribute find

→ Entropy
→ Average Information Entropy
→ Gain.

	Pi	Ni	Entropy
old	0	3	0
New	3	0	0
Mild	2	2	1

Entropy of age
(Average Information Entropy)

$$I = \sum_{i=1}^{3} \frac{P_i + N_i}{P+N} \text{Entropy}(A)$$

$$= \frac{0+3}{5+5}(0) + \frac{3+0}{5+5}(0) + \frac{2+2}{5+5}(1) = 0.4$$

$$\text{Gain} = \text{Entropy}(S) - \text{Entropy}(\text{age})$$
$$= 1 - 0.4 = 0.6.$$

Competation

	P	N	Entropy
Yes	1	3	0.8127
No	4	2	0.9182

$$\frac{-P}{P+N}\log_2\left(\frac{P}{P+N}\right) - \frac{N}{P+N}\log_2\frac{N}{P+N}$$

$$-\frac{1}{4}\log_2(1/4) - \frac{3}{4}\log 3/4$$

$$-\frac{4}{6}\log_2(4/6) - 2/6\log_2(2/6)$$

Entropy Competation

$$= \frac{(1+3)}{(5+5)} \times (0.81127) + \frac{(4+2)}{(5+5)}(0.918295)$$
$$= 0.8754$$

$$\text{Gain} = \text{Class entropy} - \text{Entropy(Competation)}$$
$$= 1 - 0.8754$$

Gain = 0.124155

	P	N	Entropy
Software	3	3	1
Hardware	2	2	1

Entropy(Type)

Avg Inf $= \frac{3+3}{5+5}(1) + \frac{2+2}{5+5}(1) = 6/10 + 4/10$

$= 10/10 = 1$

Gain = Class Entropy − Entropy(Type)

$= 1 - 1 = 0$

Information Gain:-

Age	0.6
Competitn	0.1245
Type	0

Next Repeat

Age	competition	Type	Profit
mild	Yes	Software	Down
mild	Yes	Hardwa	Down
mild	No	Hardwa	up
mild	No	Softwa	up

} Current dataset

Entropy for the class

P=2 N=2

Entropy(Profit) = 1

ompetition

	P	N	Entropy
Yes	0	2	0
NO	2	0	0

Entropy(Competition)

Avege In $= \frac{2}{4}(0) + 2/4(0) = 0$

Gain = Entropy class − Entropy(Competition)

$= 1 - 0 = 1$

Type

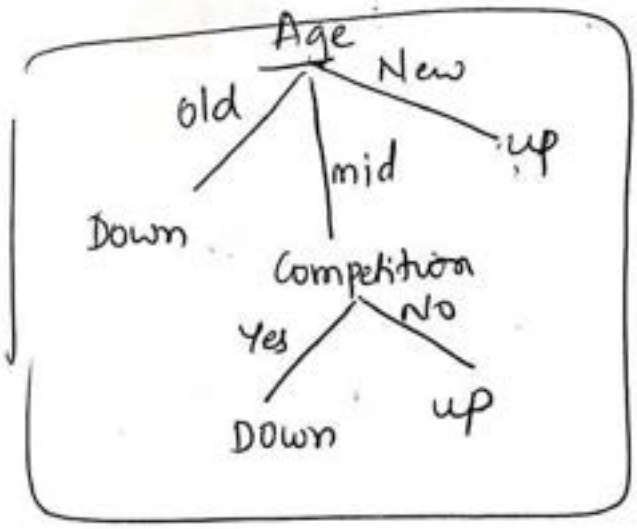
	P	N	Entropy
Soft	1	1	1
Hard	1	1	1

Avege In Entropy(Type)

$= 2/4(1) + 2/4(1) = 2/4 + 2/4 = 1$

Gain

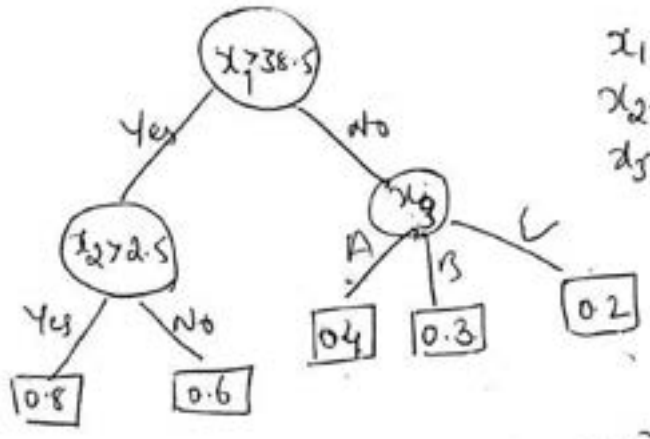
Competition	1
Type	0



Rule Extraction from a Decision tree (1)

One rule is created for each path from the root to a leaf.

- Each attribute-value pair along a path forms conjunction. The leaf holds the class prediction.



x_1 = age
x_2 = years in job
x_3 = job type

Rules

R_1: If (age > 38.5) and (years_in_job > 2.5) then $y = 0.8$

R_2: If (age > 38.5) and (years-in-job ≤ 2.5) then $y = 0.6$

R_3: If (age ≤ 38.5) and (job-type = 'A') then $y = 0.4$

R_4: If (age ≤ 38.5) and (job type = 'B') then $y = 0.3$

R_5: If (age ≤ 38.5) and (job type = 'C' then $y = 0.2$

Interpretability: Another major advantage of decision trees is interpretability.

Learning rules from Data

Sequential covering Algm

→ Rules are learned sequentially, where each rule for a given class will ideally cover many of the class tuples.

Pruning

→ Stop splitting a node if a decision based on too few instances causes variance and thus generalization error. Stopping tree construction early on before is called Prepruning the tree

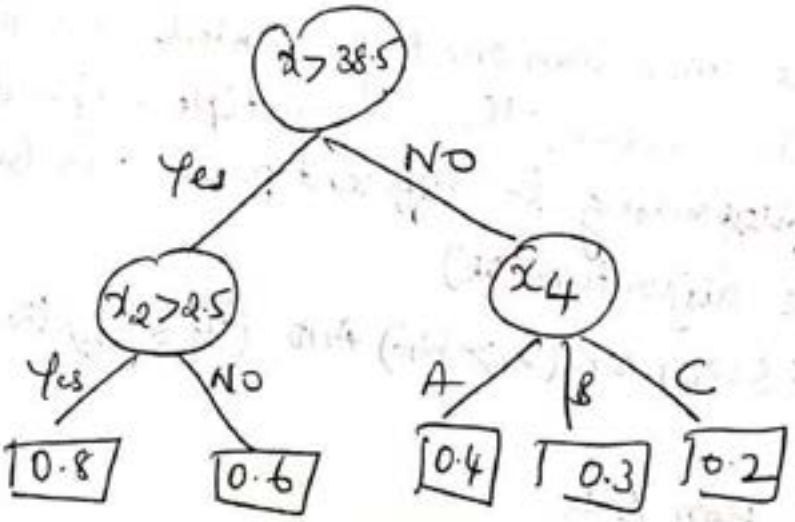
→ Eg: If you have 5 instances and split into 2 & 3 instances is not a significant split.

Post Pruning:- Split the entire dataset into training set and testing set.
Train the training set for decision tree and run the testing set on the decision tree, and accuracy is not 100%, because the tree is overfit to the training data.

→ Remove the node one by one from tree
→ Measure the performance on testing set.
→ Remove the node that results in greatest improvement.
→ Repeat until further pruning is harmful.

Rule Extraction from a Decision tree

- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction: the leaf holds the class prediction.
- Rules are mutually exclusive



x_1 = age
x_2 = Years in job
x_3 : job type

R_1: If (age > 38.5) and (Years in job > 2.5) then $y = 0.8$
R_2: If (age > 38.5) AND (Years-in-job ≤ 2.5) then $y = 0.6$
R_3: If (age ≤ 38.5) AND (job-type = 'A') then $y = 0.4$
R_4: If (age ≤ 38.5) AND (job-type = 'B') then $y = 0.3$
R_5: If (age ≤ 38.5) AND (job-type = 'C') then $y = 0.2$

Interpretability:- another main advantage of decision trees is interpretability. The decision nodes carry conditions that are simple to understand. Each path from root to a leaf corresponds to one conjunction of tests, as all those conditions should be satisfied to reach to the leaf. These paths together can be written down as a

Set of IF-THEN rules, called a rule base.
Rule base is a knowledge extraction, which is the model learned from data.

- Rule support indicates the percentage of training data covered by the rule.
- Rule reflects the main characteristics of the dataset. They show the important features and split positions.
- There may be more than one leaf labeled with the same class. In such case, these multiple conjunction expression corresponding to different paths can be combined as disjunction (OR).

If $(x \leq w_{10})$ OR $(x_1 > w_{10})$ AND $(x_2 \leq w_{20})$ then C_1

Learning rules from Data

Sequential Covering Algm :- Rules are learned sequentially, where each rule for a given class will ideally cover many of the class tuples. A covering algm is an algm that develops a cover taking into account only set of positive examples but none of negative examples. It learns one rule at a time and repeat this process to gradually cover the full set of positive examples.

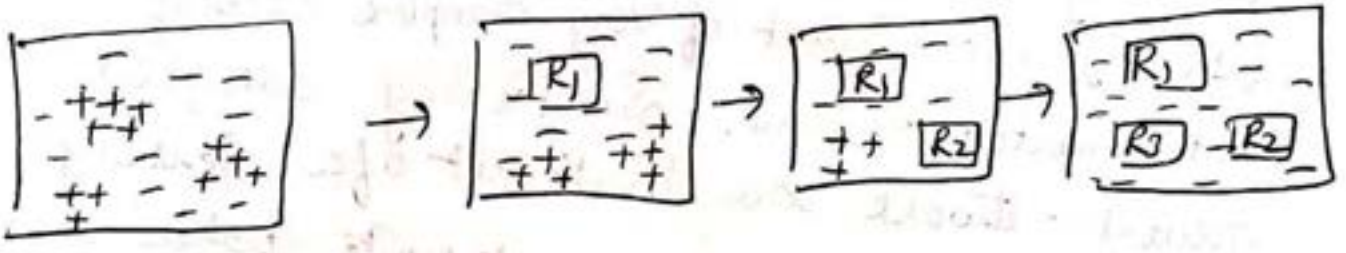
Steps
1) Starts with empty cover (No rules generated)
2) Rules are learned one at a time. Rule is said to cover an example if the example satisfies all the conditions of the rule.
3) Once a rule is obtained, all the training

examples covered by the rule are removed from the training set.

4) Goto step2 until enough rules are added.

Example

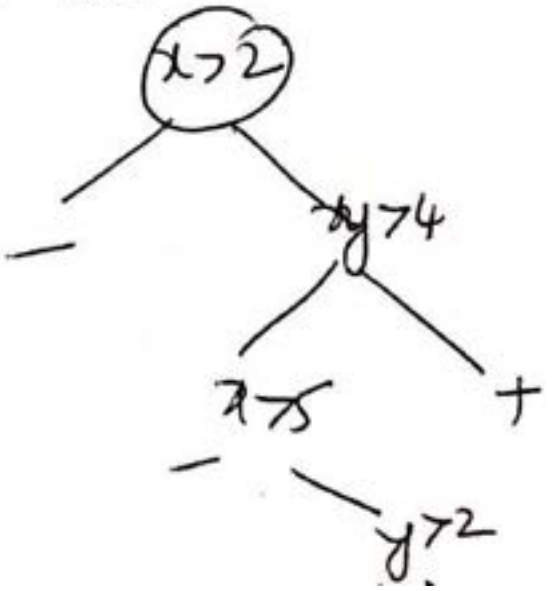
Ripper :- One example of rule induction is Ripper. We start with the case of positive and negative examples. Rules are added to explain positive examples. Other instances which are not covered by the rule is negative examples.



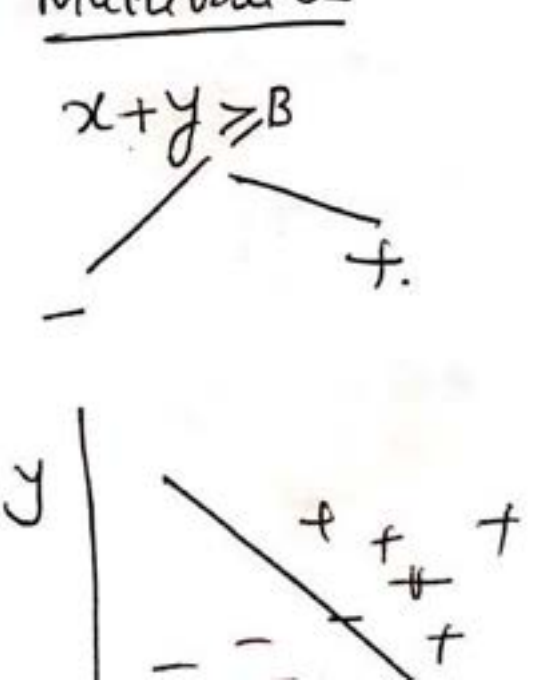
Multivariate trees :-

- Multivariate Decision tree are able to generalize well when dealing with attributes correlation & different from univariate.

Ex :- univariate



Multivariate



Artificial neuron is called perceptron

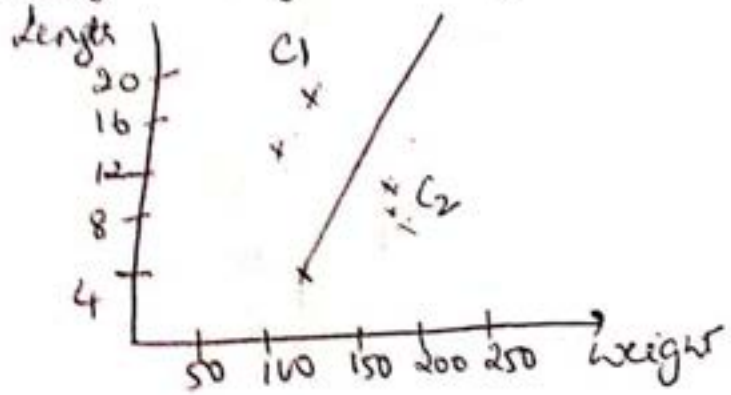
(1) Single Layer perceptron

	weight (grams)	length (cm)
Fruit 1 (class 1)	121	16.8
	114	15.2
Fruit 2 (class 2)	210	9.4
	195	8.2

Sample data obtained from two different fruit.

→ Train a single layer perceptron model using the above parameters to classify the two fruits.

Using the model parameters we can classify the fruit with weight 140 gm & length 17.9cm.



We found a decision boundary which classify the two classes.

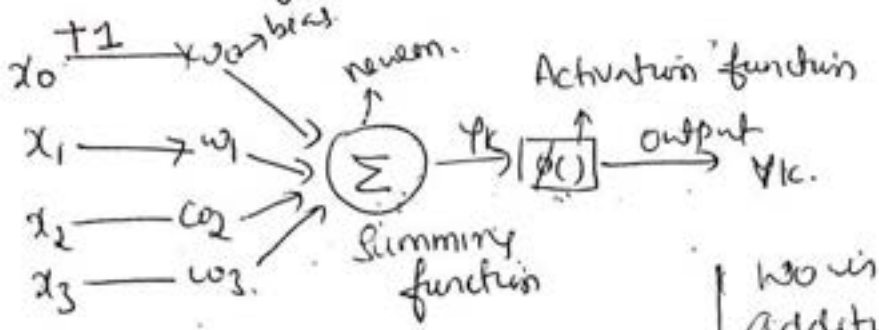
If new weight and height is given, this model is able to predict C_1 and C_2.

How will implement neural network for the above problem is shown in the next page.

Perceptrons [Perceptrons training] Single layer perceptron

(2) (4)

One type of ANN system, is based on a unit called perceptron. A perceptron takes a vector of real valued inputs, calculates the linear combination of these inputs and output.



$$v_k = \sum_{j=0}^{m} w_j x_j + w_0 \quad \text{— (1)}$$

w_0 is the bias is additional parameter which is used to adjust the output. Help the model to best fit the given data.

Activation is a threshold function.

$$\text{Eg:} = \phi v = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases} \text{ or } \begin{cases} 1 & \text{if } v \geq 0 \\ -1 & \text{if } v < 0 \end{cases}$$

where each w_i in eqn (1) is a real valued constant or weight, that determines the contribution of input x_i to perceptron output.

→ Perceptron is the simplest form of a neural network used for the classification of patterns said to be linear seperable.

Training a perceptron

(b)

$x(n) = [+1, x_1(n), x_2(n) \cdots x_m(n)]$

$w(n) = [b(n), w_1(n), w_2(n) \cdots w_m(n)]$

$b(n) \rightarrow$ bias, $y(n) \rightarrow$ actual response, $d(n) \rightarrow$ desired response

$\eta \rightarrow$ learning rate $\rightarrow$ It is degree to which weights are changed at each step

Steps:-

(1) Initialization: Initialize weights & inputs.

(2) Activation - At time step n, activate the perceptron by applying input vector $x(n)$ and desired response $d(n)$

(3) Computation of actual response of the perceptron

$$y(n) = \text{sgn}[w^T(n) \cdot x(n)] \quad -(1)$$

where $\text{sgn}(\cdot)$ is the signum function [Activation function]

$$\text{Sgn}(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases} \quad -(2)$$

(4) update the weight vector of the perceptron.

$$w(n+1) = w(n) + \eta[d(n) - y(n)]x(n) \quad -(3)$$

(5) Goback to step2 [retraining]

Note - (a) One way to learn acceptable weight vector is to begin with random weights then iteratively apply the perceptron to each training examples, modify the perceptron weights whenever it misclassifies an example.

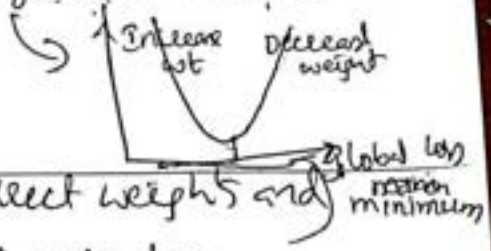
(b) This process is repeated iteratively through the training examples correctly.

(c) Weights are modified at each step according to the perceptron training rule, which revises the weight associated with input x_i

(d) Suppose training example is correctly classified $(d(n) - y(n)) = 0$ then no weights are updated.

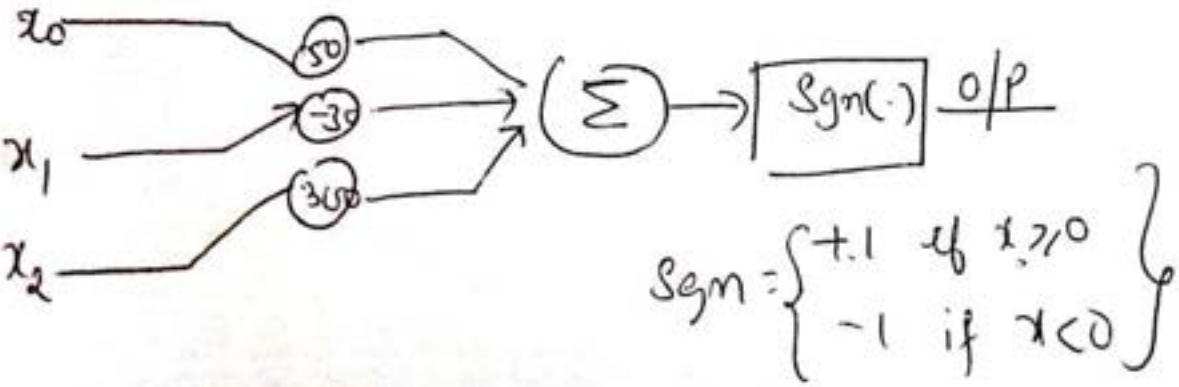
(e) Suppose perceptron outputs -1, when target o/p is $+1$. To make perceptron output $+1$ instead of -1, weights must be altered.
Increasing w_i will bring the perceptron closer to correctly classifying this example.

(f) If $d(n) = -1$ and $y(n) = +1$, then weight will be decreased rather than increased.
This is Also known as stochastic gradient descent.



Example

Intially Let us assume we know the correct weights and bias and see how we get the desired classification



$W_1(0)=30$, $W_2(0)=300$ $b(0)=50$, $\eta=0.01$ (7)

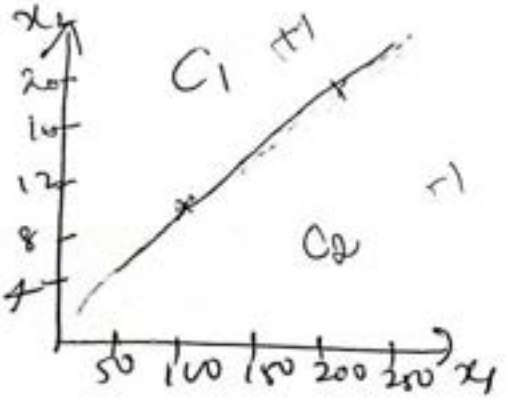
Decision boundary weights for this example is

$W_1x_1+W_2x_2+b=0$

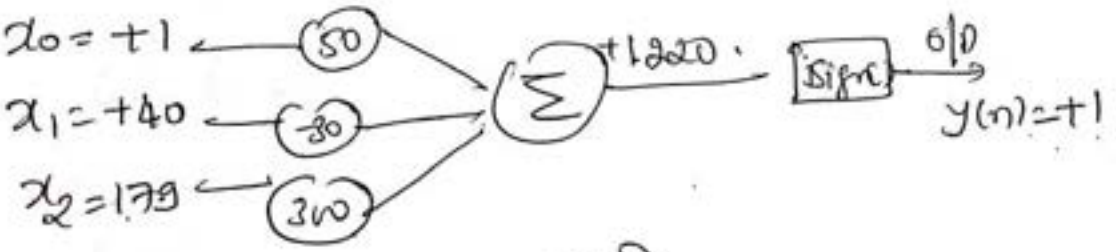
$-30x_1+300x_2+50=0$

$x_1=100 \rightarrow x_2=\frac{30\times100-50}{300}=9.83$

$x_1=200 \rightarrow x_2=\frac{30\times200-50}{300}=19.83$



This decision boundary clearly seperates two classes. So the weights are fixed and decision boundary is obtained. Let us give test input (new unknown fruit) & determine the class.



$x(n)=[+1, +40, +179]$

$W(3)=[50, -30, 300]$ $d(n)=+1$

$y(n)=Sgn\left(W^T(3)\times X(n)\right)=Sgn\left[50\times1-30\times140+300\times179\right)$

$=Sgn(12920)=+1 \Rightarrow$ This unknown fruit belongs to class C_1

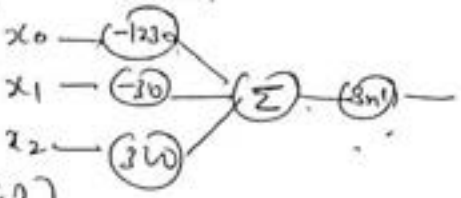
Example 2

Initially let us assume an random weight and bias and train the neural network with the training example.

Train with known fruit

$x(n) = [+1, 114, 15.2]^T$ and $d(1) = +1 \rightarrow$ class 1.

$w(n) = [-1230, -30, 300]$



$y(n) = \text{Sgn}(w^T(n)x(n))$

$= \text{sgn}(-1230 \times 1 - 30 \times 114 + 300 \times 15.2)$

$= \text{sgn}(-90) = -1 \neq d(1)$ class 2

The target class is class 1 but the output class is class 2. So new weight should be recalculated.

$w(n) = [-1230, -30, 300]$, $x(n) = [+1, 114, 15.2]$

$d(1) = +1$ $y(1) = -1$, $\eta = 0.01$

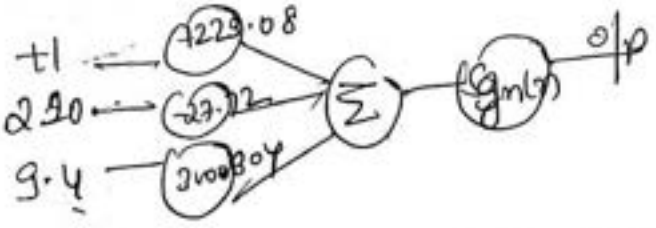
$w(1+1) = w(2) = [-1230, -30, 300] + 0.01[+1 - (-1)][+1, 114, 15.2]$

In general $w(n+1) = w(n) + \eta [d(n) - y(n)] x(n)$

$w(2) = [-1230, -30, 300]^T + [+0.02, 2.28, 0.304]^T$

$w(n) = [-1229.08, -27.72, 300.304]^T$

Train with known fruit



$w(2) = [-1229.08, -27.72, 300.304]$

$x(2) = [+1, 2.10, 9.4]$ and $d(2) = -1 \rightarrow$ class 2

$y(2) = \text{Sgn}(w^T(2)x(2)) =$

$= \text{sgn}(-1229.08 \times 1 - 27.72 \times 210 + 300.304 \times 9.4)$

$= \text{sgn}(-4227.4224) = -1 = d(2)$

Hence no need to recalculate the weights.

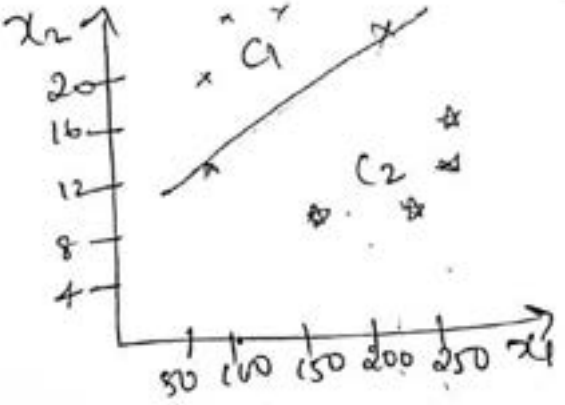
$d(n) - y(n) = 0$

Therefore decision boundary after neural network training is.

$-27.72x_1 + 300.304x_2 - 1229.08 = 0$

$x_1 = 100 \rightarrow x_2 = \frac{-27.72 \times 100 + 1229.08}{300.304} = 13.32$

$x_2 = 200 \rightarrow x_2 = \frac{27.72 \times 200 + 1229.08}{300.08} = 22.55$



Once the decision boundary is built, new test examples can be applied to get to idea which class it belongs.

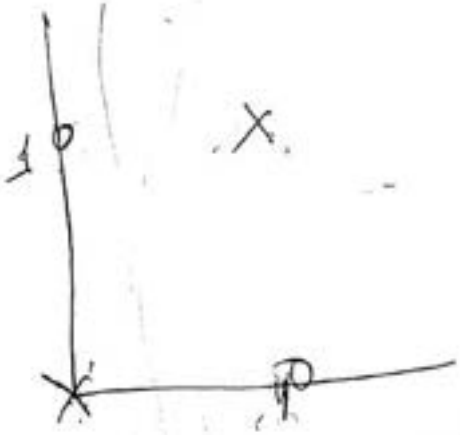
Limitations of Perceptrons (Single layer Perceptrons)

→ It can seperate classes that lie either side of straight line easily.

→ But it in reality decision bpw classes are more complex. For example classical Exclusive (XOR) problem

XOR

x_1	x_2	o/p
0	0	0
0	1	1
1	0	1
1	1	0



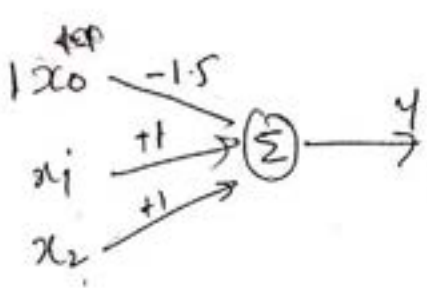
XOR problem is not linearly seperable. We cannot draw a line that seperates X class and circle class.

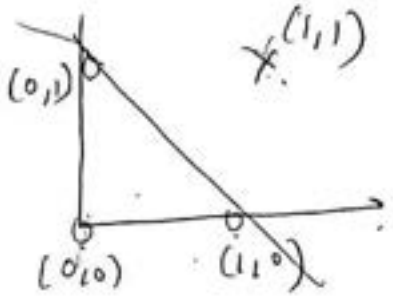
An example of a perceptron that implements AND and its geometric interpretation in two is given below

AND

$x = [1, x_1, x_2]$ and $w = [-1.5, 1, 1]$

x_1	x_2	o/p
0	0	0
0	1	0
1	0	0
1	1	1





Discriminate function $wx_1+w_2x_2+b=0$

$y = \text{Sgn}(x_1 + x_2 - 1.5)$

Activation function

$$\begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$

$x_1 = 1, x_2 = 0$

$y = \text{Sgn}(1 + 0 - 1.5)$

$y = \text{Sgn}(-0.5)$

$y = 0$

$x_1 = 1, x_2 = 1$

$y = \text{Sgn}(1 + 1 - 1.5)$

$y = \text{Sgn}(0.5)$

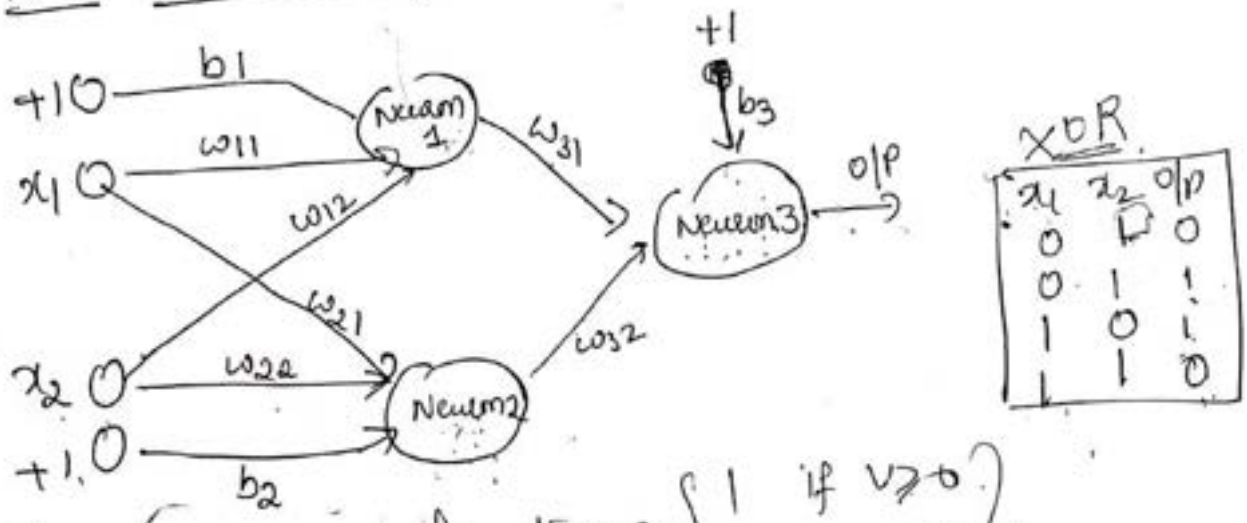
$y = 1$

Multilayer Perceptron

A perceptron that has a single layer of weights can only approximate linear function of inputs and cannot solve problems like XOR, where the discriminant to be estimated is nonlinear. Similarly a perceptron cannot be used for nonlinear regression.

Multilayer perceptron can be used for classification which can implement nonlinear discriminants.

XOR in multilayer perceptron



XOR

x_1	x_2	o/p
0	0	0
0	1	1
1	0	1
1	1	0

$\phi(v)$ = Activation function = $\begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$

(i) $w_{11} = w_{12} = w_{21} = w_{22} = w_{31} = w_{32} = +1$

$w_{31} = -2$, $b_1 = -1.5$, $b_2 = b_3 = -0.5$

For (0,1) input $W_H(n) = \begin{bmatrix} -1.5 & -0.5 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$ $x(n) = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$

$$y^H(n) = \phi\left[W^T(n)\, x(n)\right] = \phi\left[\begin{bmatrix} -1.5 & -0.5 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}^T \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}\right]$$

$$= \phi\left[\begin{bmatrix} -1.5 & 1 & 1 \\ -0.5 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}\right] = \phi\begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} y_1^H \\ y_2^H \end{bmatrix}$$

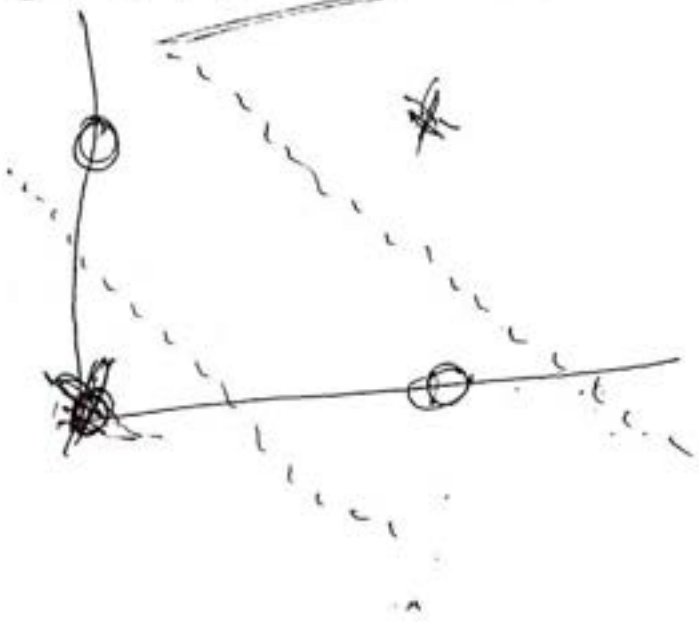
$$y^H(n) = \phi\left[w^T(n)\,x(n)\right] = \phi\left[\begin{bmatrix} b_3 \\ w_{31} \\ w_{32} \end{bmatrix}^T \begin{bmatrix} 1 \\ y_1^H \\ y_2^H \end{bmatrix}\right]$$

see the activation function

$$= \phi\left[\begin{bmatrix} -0.5 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}\right] = \phi[0.5] = \boxed{1}$$

For input [0, 1] we got = o/p as 1

Similarly, it is done for input [1, 0), [0, 0) & [1, 1].

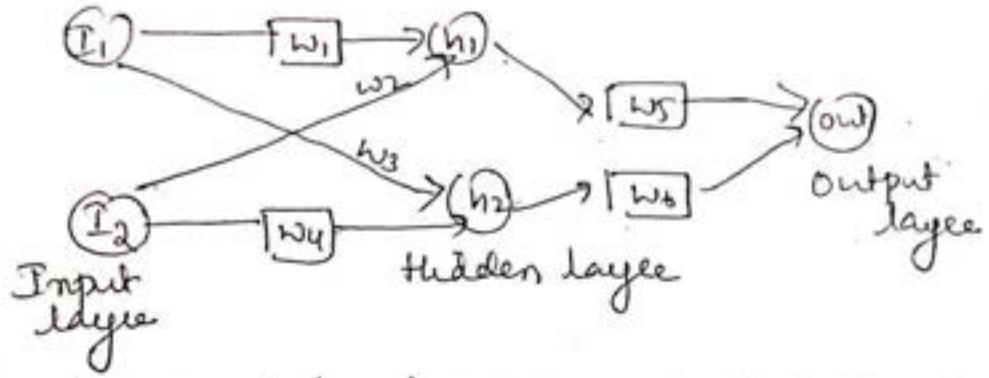


Backpropagation Algm

What is Backpropagation?

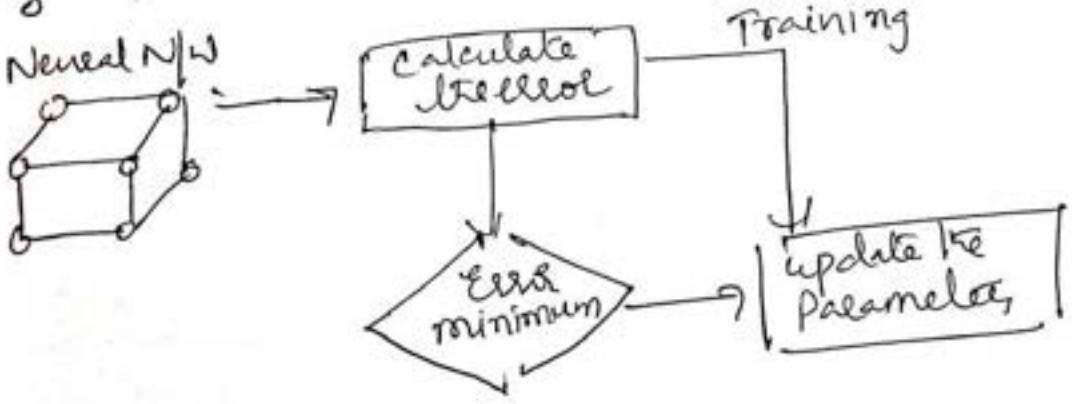
Backpropagation is a supervised learning algm, for training neural networks

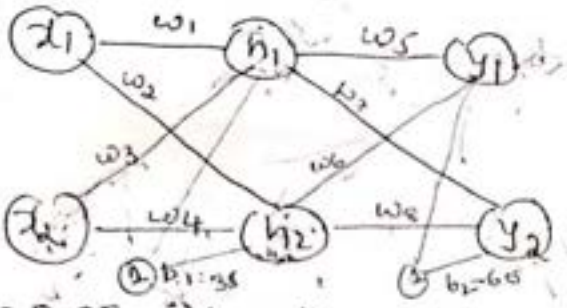
Why we need Backpropagation?



When we need to design neural network, we need to initialize the weights with random value.
There is a least probability with whatever weights assigned will lead to correct output.

If error = 0, if Prediction output = actual o/p.
If error is high we need to change the parameters (weights) so that error becomes minimum.





$x_1 = 0.05$, $x_2 = .10$

$w_1 = 0.15$, $w_2 = .20$, $w_3 = .25$, $w_4 = .30$, $b_1 = 0.3$

$w_5 = .4$, $w_6 = .45$, $w_7 = .5$, $w_8 = .55$, $b_2 = .6$

$T_1 = .01$, $T_2 = .99$. $\eta = 0.5$

Foowald pass :- Using the initialized random weights, we first compute the predicted output.
Then find the error between predicted output and actual output.
TO reduce the error we update the weights from Backward using gradient descent, so that error is minimum.
Again we go forward pass, calculate the difference between predicted and actual output. TO minimize the error update the weights through backward pass. This is repeated until the error is minimum.

Note: A complete pass over all the patterns in the training set is called an epoch.

$net_{h1} = w_1 x_1 + w_3 x_2 + b_1 \times 1$

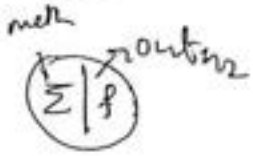
$net_{h1} = 0.15 \times 0.05 + 0.2 \times 0.1 + .35 \times 1 = 0.3775$

Apply the activation function $= \frac{1}{1+e^{-x}}$

(a) $out_{h1} = \frac{1}{1+e^{-h1}} = \frac{1}{1+e^{-0.3775}} = 0.593269$

Similarly.

$out_{h2} = 0.59688$



Calculating y_1

$y_1 = out_{h1} \times w_5 + out_{h2} \times w_6 + b_2$

$y_1 = 0.5932 \times .4 + 0.45 \times .5968 + 0.6$

$y_1 = 1.1059059$

(b) $Out\ y_1 = \frac{1}{1+e^{-y_1}} = \frac{1}{1+e^{1.105505}} = 0.751361$

In the same way.,

$Out\ y_2 = 0.77292$

Calculating total error

$E_{total} = \Sigma \frac{1}{2} (Prediction - actual.)^2$

$= \frac{1}{2} (out_{y_1} - T_1)^2 + \frac{1}{2} (out_{y_2} - T_2)^2$

$= \frac{1}{2} (.07513 - 0.01)^2 + \frac{1}{2} (.075136 - .99)^2$

$E_{total} = .298$

To reduce the error we will backpropate and update the weights. (16)

Consider w_5

Error at $w_5 = \frac{\delta E_{total}}{\delta w_5}$

$$\frac{\delta E_{total}}{\delta w_5} = \frac{\delta E_{total}}{\delta out_{y_1}} \times \frac{\delta out_{y_1}}{\delta y_1} \times \frac{\delta y_1}{\delta w_5}$$

$$\frac{\delta E_{total}}{\delta out_{y_1}} = \frac{1}{2}(T_1 - out_{y_1})^2 + \frac{1}{2}(T_2 - out_2)^2$$

$$= 2 \cdot \frac{1}{2}(T_1 - out_{y_1}) \times -1 + 0$$

$$= -(T_1 - out_{y_1})$$

$$= -(0.01 - 0.75136) = 0.74136$$

$$\frac{\delta out_{y_1}}{\delta y_1} = out_{y_1}(1 - out_{y_1})$$

$$= 0.75136(1 - 0.75136)$$

$$= 0.186815$$

Note: $out_{y_1} = \frac{1}{1 - e^{y_1}}$

$$\frac{\delta y_1}{\delta w_5} = 1 \times out_{h_1} \times w_5^{(1-1)} + 0$$

$$= out_{h_1}$$

$[y_1 = out_{h_1} \times w_5 + out_{h_2} \times w_6 + b_2]$

$$\frac{\delta y_1}{\delta w_5} = 0.59326$$

$$\frac{\delta E_{total}}{\delta w_5} = \frac{\delta E_{total}}{\delta out_{y_1}} \times \frac{\delta out_{y_1}}{\delta y_1} \times \frac{\delta y_1}{\delta w_5}$$

$$= 0.74136 \times 0.1868 \times 0.59326$$

$$= 0.082167$$

Updating w_5

$$w_5 = w_5 - \eta \times \frac{\delta E_{total}}{\delta w_5}$$

$$w_5 = 0.4 - 0.5 \times 0.082167$$

$$= 0.358916$$