Introduction to Machine Learning

Pattern: A pattern is an abstraction, represented by a set of measurements describing a "physical" Object.
Pattern can be visual, temporal, sonic, logical, ...

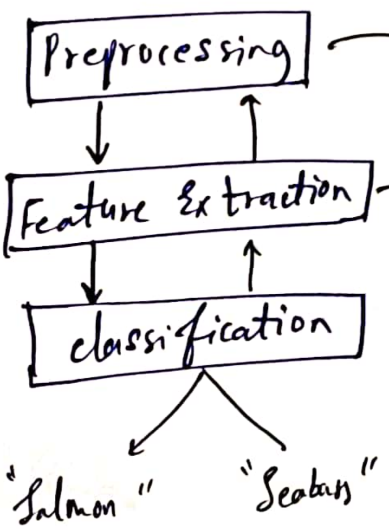
Class or Category:
- A set of pattern sharing common attributes.
- A collection of similar not necessarily identical, objects.

Feature:
- Properties of an object.
- Ideally representative of a specific type (class) of objects.

Feature Extractor: A program that inputs data (images) and extracts feature that can be used in classification

Classifier: A program that inputs feature vector and assigns it to one of a set of designated classes or to the "reject" class.

ML Process:
1. Data acquisition and Sensing
2. Pre-processing
3. Feature Extraction
4. Classification



Preprocessing → Using Segmentation operation to isolate fishes from one another and from background.

Feature Extraction → Represents the data by measuring certain features.

Decision boundary :-

↳ A conceptual line or surface that seperates different classes or categories in a classification problem.

↳ The dividing line that a machine learning model uses to determine which class a new data point belongs to.

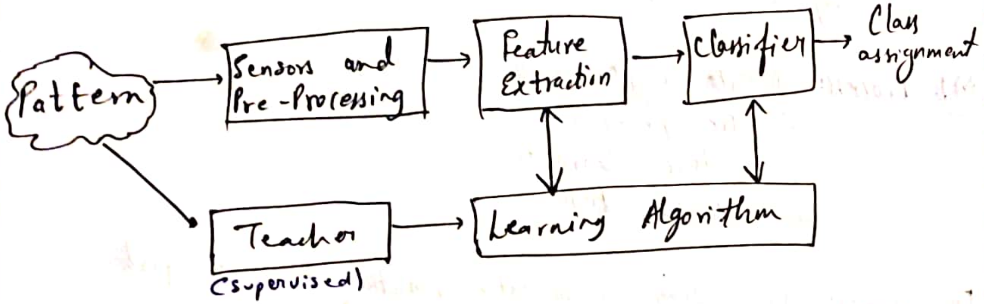
Supervised learning :-

↳ Given input X and label Y learn a mapping from X to Y.

Classifier :

↳ A classifier partitions sample space X into class-labelled regions such that $X = X_1 \cup X_2 \cup \dots \cup X_{|Y|}$ and $X_i \cap X_j = \phi$

- Classification consists of determining to which region a feature vector x belongs to.
- Borders between "Decision Boundaries" are called decision regions.



Example : $\boxed{\text{Linear classifier:}}$

$x \in X$

$$q(x) = \begin{cases} a & \text{if } (w.x) + b \geq 0 \\ b & \text{if } (w.x) + b < 0 \end{cases}$$

Minimum Distance classifier

1. m_j : The mean vector or centroid of class ω_j. Calculated as the average of all feature vectors X belonging to class ω_j

$$\boxed{m_j = \frac{1}{N_j} \sum_{X \in \omega_j} X}$$

where N_j is the number of data points (or pixels) in class ω_j.

2. $D_j(X)$: The Euclidean distance between a data point X and the mean vector m_j of class ω_j :

$$\boxed{D_j(X) = \| X - m_j \|}$$

This measures how far X is from centroid of class j.

3. Classification Rule :
A new data point X is assigned to class ω_i if its distance to the mean vector m_i is smaller than its distance to any other class's mean vector:

$$D_i(X) < D_j(X) ; \; j = 1, 2, \dots, M \; ; \; j \neq i$$

Intuition :-

The minimum Distance classifier assigns a data point to the class whose mean (centroid) it is closest to, based on Euclidean distance.

⊛ It works well when the classes are compact and well-separated in feature space.

Lazy learners :-

Lazy:- Do not create a model of the training instances in advance.

- When an instance arrives for testing, runs the algorithm to get the class prediction.
  - Example :- K-nearest neighbor classifier (K-NN)

K-NN classifier schematic [Non-parametric, supervised]

For a test instance,

1) Calculate distance from training points.
2) Find K-nearest neighbours (say $K = 3$)
3) Assign class label based on majority.

$$\boxed{dist(X_1, X_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}}$$ Eucleadian.

$$\boxed{v' = \frac{v - min_A}{max_A - min_A}}$$ Data Normalization.

v : original feature value

v' : normalised feature value

min_A : minimum value of feature A.

max_A : maximum value of feature A.

$dist(X_1, X_2)$: Computes the distance between the test instance and all training points using a distance metric.

X_1, X_2 : Feature vectors in n-dimensional space.

$X_1 = [x_{11}, x_{12}, \ldots, x_{1n}]$

$X_2 = [x_{21}, x_{22}, \ldots, x_{2n}]$

$x_{ij} \Rightarrow$ Value of i^{th} feature for j^{th} data point.

Bayesian classifier :-

- Bayes theorem provides a way to calculate the posterior probability of a class given the observed data.

$$\boxed{P(w_i|x) = \frac{P(x|w_i) \cdot P(w_i)}{P(x)}}$$

where;

$P(w_i|x)$: posterior probability of a class w_i given the data x.

$P(x|w_i)$: class-conditional probability (likelihood) of observing data x given class w_i.

$P(w_i)$: Prior probability of class w_i

$P(x)$: Evidence or marginal probability of observing the data x.

$$\boxed{P(x) = \sum_{j=1}^{c} P(x|w_j) \cdot P(w_j)}$$ c : total # of classes.

Bayesian Decision Rule :-

To classify a new data point, assign it to the class with the highest posterior probability :

$$\boxed{Class = \arg \max_{w_i} P(w_i|x)}$$

Bayes Discriminant function for classification.

$$G(x) = \log \frac{P(x|class1) \cdot P(class1)}{P(x|class 2) \cdot P(class 2)}$$

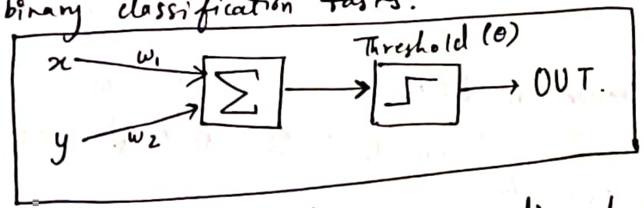
Decision Rule :

- If $G(x) > 0$, classify the data point as class 1.
- If $G(x) \leq 0$, classify the data point as class 2.

Scanned with CamScanner

Neural classifier :-

↳ **Perceptron** :- A perceptron is the simplest type of artificial neural network. It acts as a single neuron connected by weights to a set of inputs and is used for binary classification tasks.



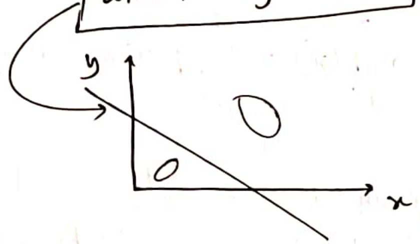
$$z = w_1 x + w_2 y$$

if $z > \theta$, output = 1

Otherwise, output = 0

∴ The equation of the decision boundary becomes;

$$\boxed{w_1 x + w_2 y = \theta}$$



⊛ Learning Rule for Perceptron

- **Learning Process** :- The perceptron learns by adjusting its weights $\boxed{\text{iteratively}}$ until it produces the correct output for all input pattern.

- **Weight Update Formula** :-

$$w_i(t+1) = w_i(t) + \Delta_i$$

where ; $\Delta_i = \eta \delta x_i$

- η : Learning rate (a small positive constant)
- $\delta = T - A$: Error term (Target Output − Actual Output)
  - Target Output ↓ Desired
  - Actual Output ↓ perceptron calculation
- x_i : input feature
- t : iteration step.

⊛ **Linear Separable** :

↳ If the input patterns are linearly separable, the perceptron algorithm is guaranteed to find a separating hyperplane in a finite number of steps.

- **Convergence** :

↳ The weights will converge to values that correctly classify all training data points if linear separability holds.

⊛ The perceptron is a foundational model in neural networks that classifies data points by learning a linear decision boundary.

↳ It adjusts its weights using a simple learning rule based on errors in predictions.

- **change of weights** in a neural network adjusts the decision boundary to better classify data points.
- Each iteration of training updates the weights to minimize classification errors.

Cascading layers:

- Cascading layers refers to the stacking of multiple layers of neurons in a neural network.
- This stacking allows the network to model increasingly complex patterns and decision boundaries.

Q) Why do we Need Multiple layers:

1.) Single layer (Linear Decision Boundary):

- A single-layer neural network (like a perceptron) can only create linear decision boundaries.

Ex: If you have two classes of data points that can be seperated by a straight line, a single-layer network is sufficient.

2. Two Layers (Convex Regions):

- When data points cannot be seperated by a single straight line but can be seperated by convex shapes (eg. triangles or polygons), two layers are required.
- The second layer combines the outputs of the first layer to form convex regions in the feature space.

3. Three or More Layers (Non-Convex Regions):

- For more complex patterns, where data points are in non-convex regions (eg. irregular or curved shapes), three or more layers are needed.
- Each additional layer adds more complexity, allowing the network to model intricate patterns.

- An example of a single-layer neural network failing and why a two-layer network is needed.

↳ classical XOR problem:

- Consider You want catch a friend when he's telling a lie.

↳ You friend tells lie when:

① His face gets red and he is not smiling.

② His face is not red and he is smiling.

	0	1
0	No red No Smile	No red Smile → lieing
1	red No Smile → lieing	red Smile

AND Logic

	0	1
0	No red No Smile	No red Smile
1	red No Smile	red Smile

Red and Smile

↳ linearly Separable. (one perceptron)

OR Logic

	0	1
0	No red No Smile	No red Smile
1	red No Smile	red Smile

Red or Smile.

↳ linearly Seperable. (one perceptron).

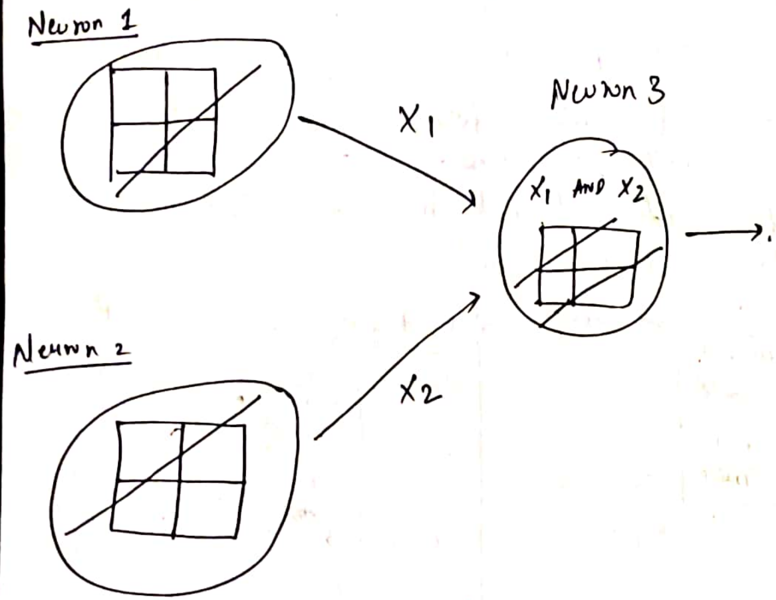
What we want:

	0	1
0	No red No Smile	No red Smile
1	red No Smile	red Smile

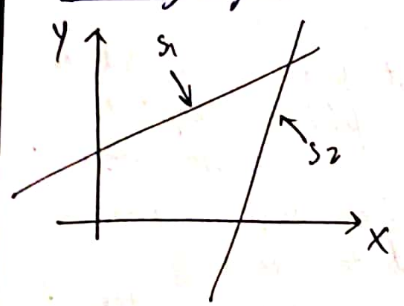
Red XOR Smile

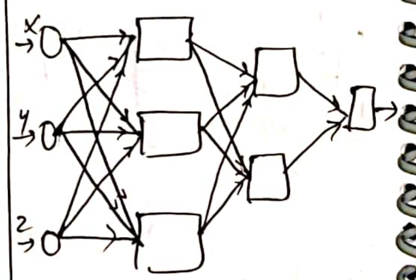
Not linearly Seperable.

- Single line cannot be use to seperate the data
- What we can do is use 2 layer.
- In first layer we use And logic ~~get~~ feed it's output to a different neuron, say neuron ③, take OR logic & feed it's output to neuron ③, Neuron 3 performs intersection of AND & OR logic, i.e. neuron 3 says output 1 when both output of AND and OR logic is 1.

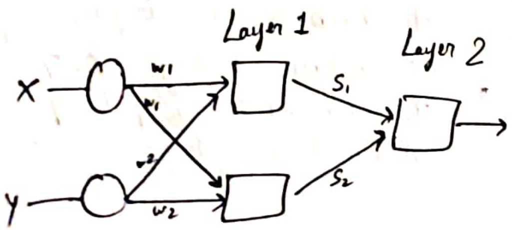


· Cascading layers





Multi layer.



Two Layer

Neuro - Computing

· **Purpose:** To achieve human like performance particularly in pattern recognition and image processing.

· Artificial Neural Network (ANN) or Neural Network (NN) models try to simulate the biological neural network with electronic circuitry.

· Also known as Connectionists Model / Parallel Distributed Processing (PDP).

ANN:

↳ Massively parallel interconnected network of simple processing elements which are intended to interact with the objects of the real world in the same way as biological systems do.

· Computational elements (neurons / nodes / processors) are analogous to neurons (biological)

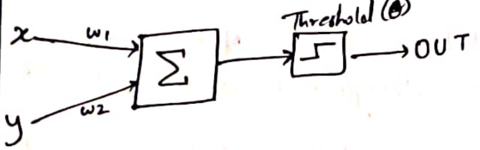
· **Characteristics of Neural Network:**

1. Learn from example - Shown a set of inputs, they self-adjust to produce consistent response.
2. Generalize from previous examples to new ones - Once trained, a network's response is mostly insensitive to variations in input.
3. Abstract essential characteristics from inputs - find the ideal (prototype) from imperfect inputs.

Scanned with CamScanner

Two input perceptron:

Perceptron: A single neuron connected by weights to a set of inputs.



$\begin{cases} w_1 x + w_2 y > \theta \Rightarrow 1 \\ \text{else} \Rightarrow 0. \end{cases}$

Separating line → $w_1 x + w_2 y = \theta$.

Learning: Present a set of input patterns, adjust the weights until the desired output occurs for each of them.

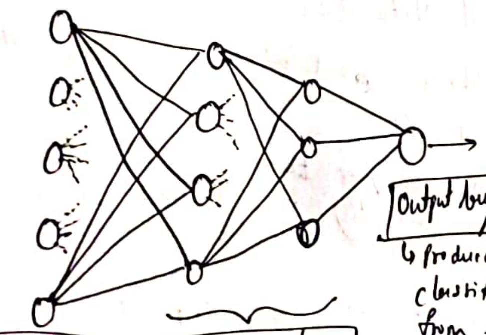
$w_i(t+1) = w_i(t) + \Delta_i$

$\Delta_i = \eta \delta x_i$;

$\delta = T - A$ (Target − actual).

- t : iteration step.
- η : learning rate.
- δ : Error term.
- x_i : input feature.

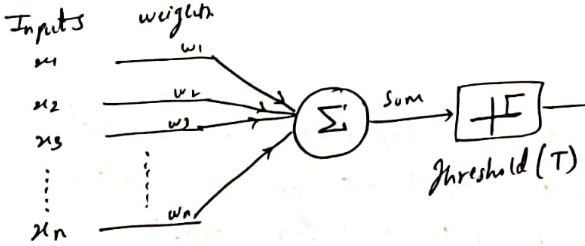
- Neural Networks are composed of networks of computational models of neurons called perceptrons.



Output layer.
↳ Produce final predictions or classification based on inputs from last hidden layer.

Input layer
↳ Simply pass input features ($x_1, x_2, \ldots$) to next layer

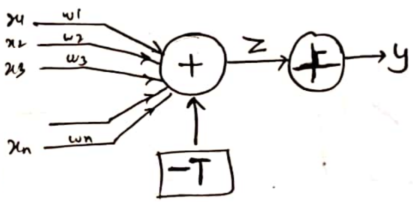
Hidden layer
↳ Compute weighted sums of inputs from previous layer & apply activation function.



$y = \begin{cases} 1; \text{ if } \sum_i w_i x_i \geq T \\ 0; \text{ otherwise} \end{cases}$

- Neuron fires if the weighted sum of inputs exceeds a threshold.

A better version of Neuron:



; $z = \sum_i w_i x_i - T$

$y = \begin{cases} 1 & ; z \geq 0 \\ 0 & ; \text{otherwise.} \end{cases}$

Key difference between the two representation of Neurons:

- In first, the threshold T is explicitly shown as seperate parameter.
- The perceptron fires if $\sum_i w_i x_i \geq T$
- For the second representation we are using the same equation $\sum w_i x_i \geq T$, just modified:

$$\sum w_i x_i - T \geq T - T \quad \left[\text{subtracting } T \text{ on both side}\right].$$

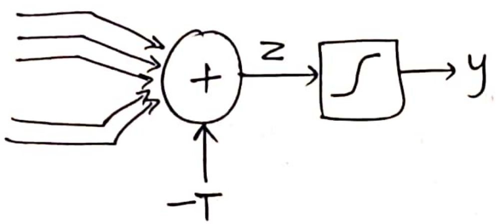
$$\boxed{\underbrace{\sum w_i x_i - T}_{z} \geq 0}$$

if Σ value is greater than or equal to Threshold, Z will be non-negative.

In this equation we call the (−T) as bias term (b) and is added to $\sum w_i x_i \Rightarrow (b = -T)$

The "soft" perceptron (logistic)

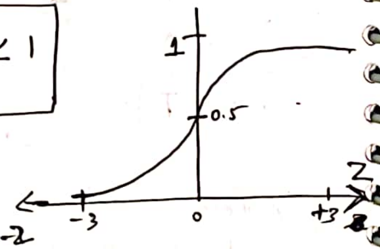
- A "squashing" function, instead of a threshold at the output:-
  - ↳ The sigmoid "activation" replaces the threshold
- Activation: The function that acts on the weighted combination of inputs and threshold.



$$z = \sum_i w_i x_i - T$$

$$y = \frac{1}{1+e^{-z}}$$

$$\boxed{y = g(z) = \frac{1}{1+e^{-z}} \; ; \; 0 < g(z) < 1}$$



When (i) z = very large

$$g(z) = \frac{1}{1+e^{-\text{very large}}}$$

$$g(z) = \frac{1}{1+\text{very small}}$$

$$g(z) \approx 1$$

When (ii) z = 0 ; $g(z) = \frac{1}{1+e^{0}} = \frac{1}{2} = 0.5$

when (iii) z = very small (some −ve) ↳ $-\infty$

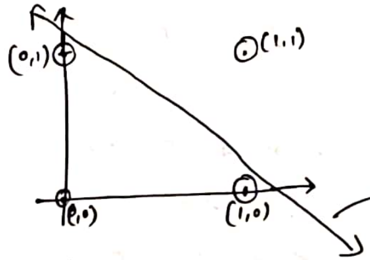
$$g(z) = \frac{1}{1+e^{-(-\infty)}} = \frac{1}{1+e^{\infty}} = \frac{1}{1+\text{very large}} \approx 0$$

Training for logic AND with a single neuron :

- One neuron can be trained to realize a linear function.

Logic AND

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1



$\to w_1x_1 + w_2x_2 \geq 0$

	Σ	Logic value
$x_1=0$ + $x_2=0$ →	0 ⇒	0
$x_1=0$ + $x_2=1$ →	1 ⇒	0
$x_1=1$ + $x_2=0$ →	1 ⇒	0
$x_1=1$ + $x_2=1$ →	2 ⇒	1 ⇒ We see logic is $\boxed{1}$ when sum is 2.

we can decide value of w_1, w_2 & θ

↳ $w_1 = 1$, $w_2 = 1$ & $\theta = -1.5$ → subtraction 1.5 from 2 still will be > 0.

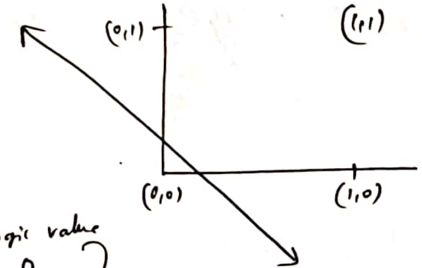
~~$x_1 + x$~~ $\quad 1.x_1 + 1.x_2 \geq 1.5$

or

$$\boxed{x_1 + x_2 - 1.5 \geq 0}$$

Logic OR

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1



	Σ	Logic value
$x_1=0$ + $x_2=0$ →	0 →	0
$x_1=0$ + $x_2=1$ →	1 →	1
$x_1=1$ + $x_2=0$ →	1 →	1
$x_1=1$ + $x_2=1$ →	2 →	1

$w_1 = 1$, $w_2 = 1$, $\theta = -0.5$

$$\boxed{x_1 + x_2 - 0.5 \geq 0}$$

Logic NOT

x	y
0	1
1	0

Logic value.

$x_1 = 0 \rightarrow 1$

$x_1 = 1 \rightarrow 0$

$w_1 = -1$, $\theta = 0.5$

When $x_1 = 0 \Rightarrow$ $w_1 x_1 + 0.5 \geq 0$

$-1.0 + 0.5 \geq 0$

$0 + 0.5 \geq 0$

$0.5 \geq 0 \longrightarrow$ True $\rightarrow$ logic becoms 1.

When $x_1 = 1 \Rightarrow$ $w_1 x_1 + 0.5 \geq 0$

$-1.(1) + 0.5 \geq 0$

$-1 + 0.5 \geq 0$

$-0.5 \geq 0 \rightarrow$ False $\rightarrow$ logic becomes 0

Logic XOR [Linearly Not Seperable].

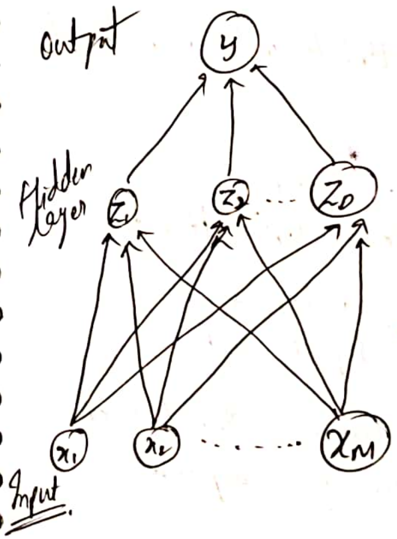
↳ Not possible using single perceptron.

↳ we need Multilayer perceptron.

Artificial Neural Network [Multilayer perceptron - MLP]

↳ Training of a neural network is to get the right weights and biases such that the error across the training data is minimized.

- Inputs are fed simultaneously into the input layer
- The weighted outputs of these units are fed into hidden layer.
- The weighted outputs of the last hidden layer are inputs to units making up the output layer.



(E) Output (sigmoid)

$$y = \frac{1}{1 + e^{-b}}$$

↑

(D) Output (linear)

$$b = \sum_{j=0}^{D} \beta_j z_j$$

↑

(C) Hidden (sigmoid)

$$z_j = \frac{1}{1 + e^{-a_j}}, \forall j$$

↑

(B) Hidden (linear)

$$a_j = \sum_{i=0}^{M} \alpha_j x_i, \forall j$$

↑

(A) Input

Given $x_i, \forall i$

A Multilayer Feed Forward Network:

- The units in the hidden layers and output layer are sometimes referred to as neurodes, due to their symbolic biological basis, or as output units.

- A network containing two hidden layers is called a three-layer neural network, and so on...

- The network is feed-forward in that none of the weights cycle back to an input unit or to an output unit of a previous layer.

- **INPUT**:- Records without class attribute with normalized attribute values.

- **INPUT VECTOR**:- $X = \{x_1, x_2, \ldots, x_n\}$
  ↳ Where n is the number of (non class) attributes.

- **INPUT LAYER**:- there are as many nodes as non-class attributes i.e. the length of the input vector.

- **HIDDEN LAYER**:- the number of nodes in the hidden layer and the number of hidden layers depends on implementation.

OUTPUT LAYER:- Corresponds to the class attribute. There are as many nodes as classes (value of class attribute).

$\boxed{O_k}$ $k = 1, 2, \ldots, \#\text{classes}$.

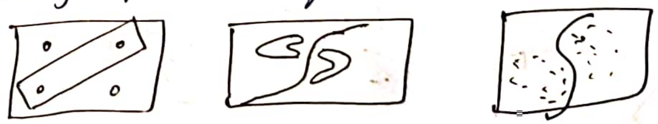
- Network is fully connected, i.e. each unit provides input to each unit in the next forward layer.

- **Decision Boundary**:-
  ↳ A decision boundary is a hypersurface that partitions the input space of a classification model into regions, each corresponding to a specific class.
  ↳ It represents the points in the input space where the model's predictions change from one class to another.

- ~~1 hidden layer ::~~
  ~~↳ Boundary of convex region (open or closed)~~

- 0 Hidden layer : linear classifier
  ↳ Hyperplanes.

- 1 Hidden layer
  ↳ Boundary of convex region (open or closed)



- 2 Hidden layer
  ↳ Combinations of convex regions.

• Multi-Layer Perceptrons (MLPs)

↳ MLPs can represent any Boolean functions, since they can emulate individual gates.

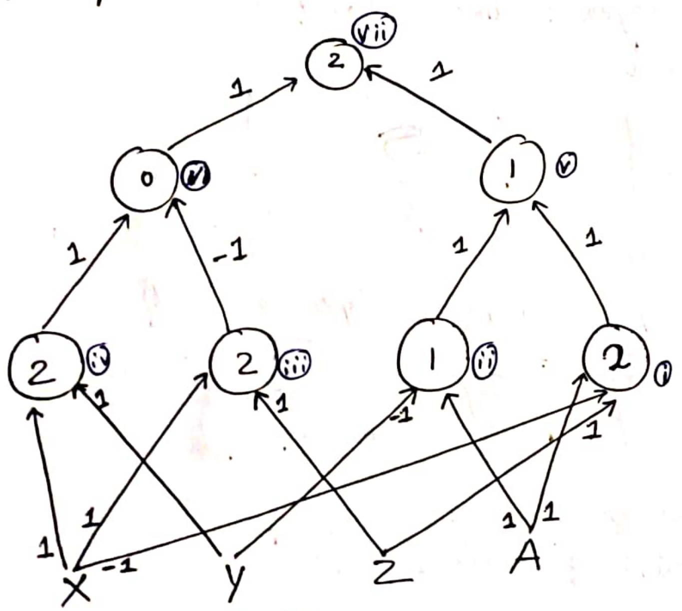
↳ MLPs are universal Boolean functions.

Example: $((A \& \bar{X} \& Z) \mid (A \& \bar{Y})) \& ((X \& Y) \mid \overline{(X \& Z)})$

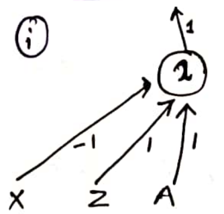
↳ $[(A \wedge \bar{X} \wedge Z) \vee (A \wedge \bar{Y})] \wedge [(X \wedge Y) \vee \overline{(X \wedge Z)}]$

• Inputs to the network are the Boolean variables: X, Y, Z, A.

• Each input can either be 1 (True) or -1 (False).



(i)



$\Rightarrow$ (S) Sum $= -1(X) + 1(Z) + 1(A)$

Threshold $T = 2$

Sum (S) $\geq 1 \Rightarrow$ output 1.

$(\bar{X} \wedge Z \wedge A) \Rightarrow$ output will be 1 when $Z = 1,\ A = 1,\ X = 0$

Sum $= -1(0) + 1(1) + 1(1) = 2 \geq 2$

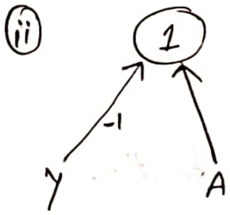
Threshold $T = 2$

Sum (S) $< 0 \rightarrow$ Output 0

$(\bar{X} \wedge Z \wedge A) \Rightarrow$ Output will be 0 when $Z = 1,\ A = 1,\ X = 1$

Sum $= -1(1) + 1(1) + 1(1) = -1 + 2 = 1 < 2$

(ii)



$\Rightarrow (\bar{Y} \wedge A) \Rightarrow$ Sum (S) $= -1(Y) + 1(A)$

Y A	Sum	Logic $(\bar{Y} \wedge A)$
$-1(0) + 1(0)$	0	0
$-1(0) + 1(1)$	1	1
$-1(1) + 1(0)$	-1	0
$-1(1) + 1(1)$	0	0

Threshold ↳ $\boxed{1}$

⋮

and so on all other logics are implemented.

Important points:

- Nodes of two different consecutive layers are connected by links or weights.
- There is no connection among the elements of the same layer.
- The layer where the inputs are presented is known as input layer.
- Output producing layer $\rightarrow$ Output layer.
- Input $\rightarrow$ Hidden $\rightarrow$ Output.
- The total (I_i) input to the i^{th} unit

  ↳ $$I_i = \sum_j w_{ij} O_j$$

  ↳ O_j : output of j^{th} neuron

- Output of node i is obtained as

  ↳ $O_i = f(I_i)$, f is activation function.

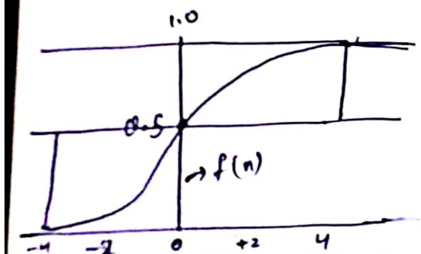
- Mostly the activation function is sigmoidal/squashing, with the form (smooth, non-linear, differentiable & saturating).

$$f(x) = \frac{1}{1 + e^{-\frac{(x-\theta)}{\theta_0}}}$$

} θ = Threshold value

θ_0 = Controls the steepness of sigmoid curve.

A larger θ_0 makes transition smoother.



- For learning (training) we present the input pattern $X = \{x_i\}$, and ask the network to adjust its set of weights/biases in the connecting links such that the desired output $T = \{t_i\}$ is obtained at output layer.
- Then another pair of X and T is presented for learning.
- Learning tries to find a simple set of weights and biases that will be able to discriminate among all the input/output pairs presented to it.
- The output $\{O_i\}$ will not be the same as the target $\{t_i\}$.

  Error is : $$\boxed{E = \frac{1}{2} \sum_i (t_i - O_i)^2}$$

  ↳ Mean squared error.

- For learning the correct set of weights error is E is reduced as rapidly as possible.
- Use gradient descent technique.