* * Deep Learning * *

ML V/s DL

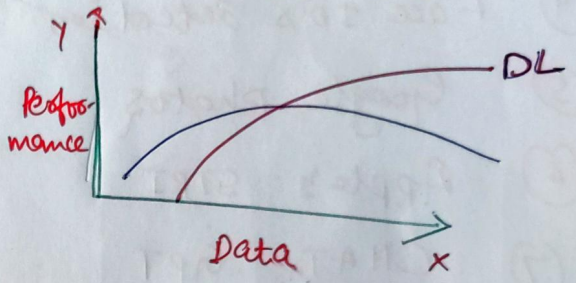
(1) ML → FE (domain Knowledge)
↓
FS

(2) Data increases but Performance not increases at that level.

(3) ML models unable to identify Complex (nonlinear) Pattern.

(1) Automated FE

(2) Data and Performance both increases.



(3) DL adept at identifying Complex Patterns.
→ Complex data

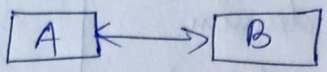
DL → It is a method in AI that teaches computers to process data in a way that is inspired by the human brain.

Need for Deep Learning ⇒

(1) To solve very complex problem.
(2) Data management becomes easier in terms of hardware
(3) Handling Large data sets.
(4)

* Uses cases in DL →

(1) Youtube Recommendation → By youtube history
→ for Experience Top Notch

(2) Google Translate → 

(3) Netflix Recommendation

(4) Face ID's detection

(5) Google photos

(6) Apple's SIRI

(7) CHAT - GPT

* Deep learning ⟷ Neural Network (NN)

→ Any deep neural network called as DL.

* Characterize DL →

(1) Artificial NN
- (1) Tabular Data
- (2) Froud Detection
- (3) Price optimisation
- (4) Convolution NN →
  - (1) Images
  - (2) Object Detection
  - (3) Stable Diffusion
  - (4) Froud Recognition
  - (5) Traffic Violation

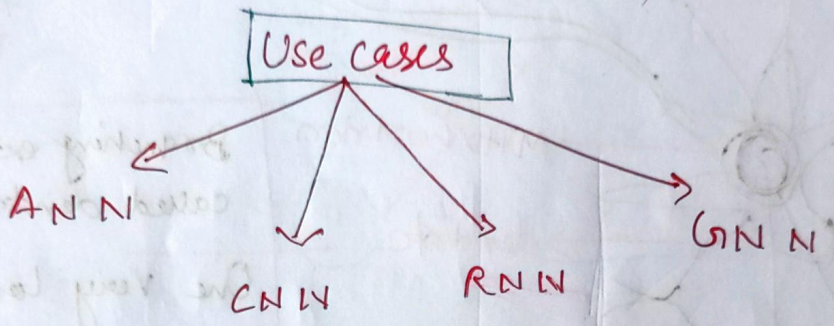
(3) Recurrent NN (RNN)
- (1) Sequential data
  - → Text (Running text)
  - → Videos
  - → Speeches
  - → Trading data
- (2) Speech Recognition
- (2) Translation
- (3) Forecast Prices
- (4) Language Modeling
- (11) Transformers → SEQ.

Transformer — Advanced RNN.

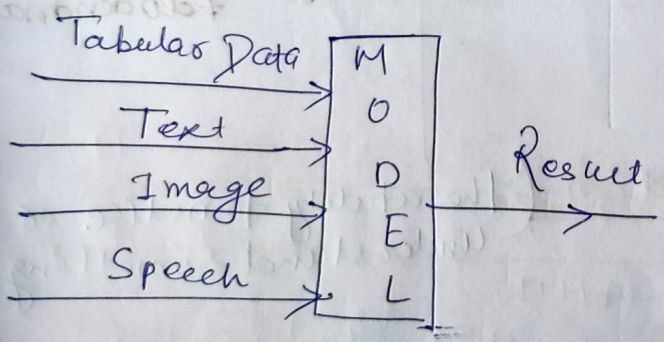
④ Graph NN

→ Social Media Data

→ Molecular structure



* Multi-Model Nature →



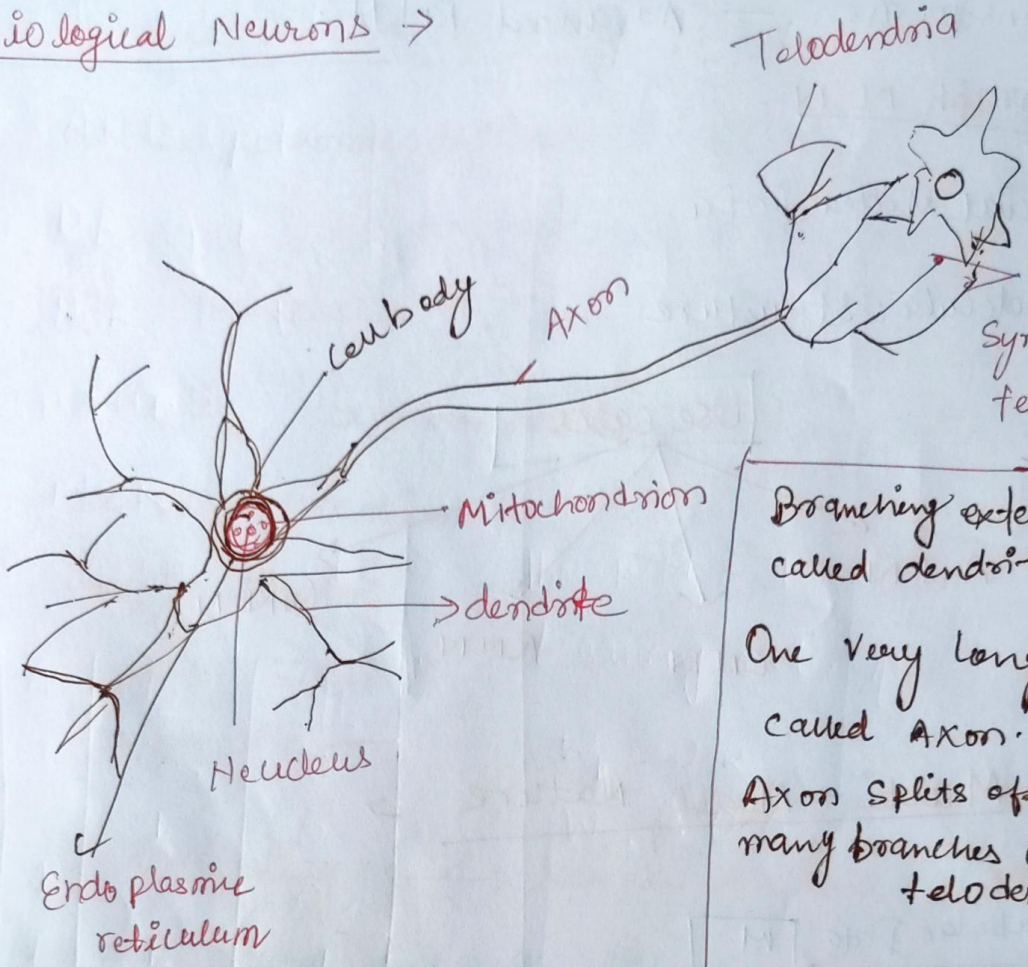
→ here not only tabular data but also all types of data are used.

* Biological Neuron <——> Artificial Neuron

→ Artificial neuron is based on Biological neuron, structure.

→ Biological neuron is also called as Spiking neuron.

Biological Neurons →



Branching extensions called dendrites.

One very long extensions called Axon.

Axon splits off into many branches called telodendria.

(Fran Rosenblatt 1957) **Perception** (The ability to notice or understand something)

↓

→ It is building block of ANN.

→ It works as linear classifier to make linear boundary.

↓

(Stepping stone to solve complex Problems)

→ $O(X_1, X_2, X_3, X_4 \cdots)$ = $(W_1X_1 + W_2X_2 + \cdots + W_b)$

I/P - Perceptron

O/P → linear combination (LC)

↓

Compare with threshold

↓

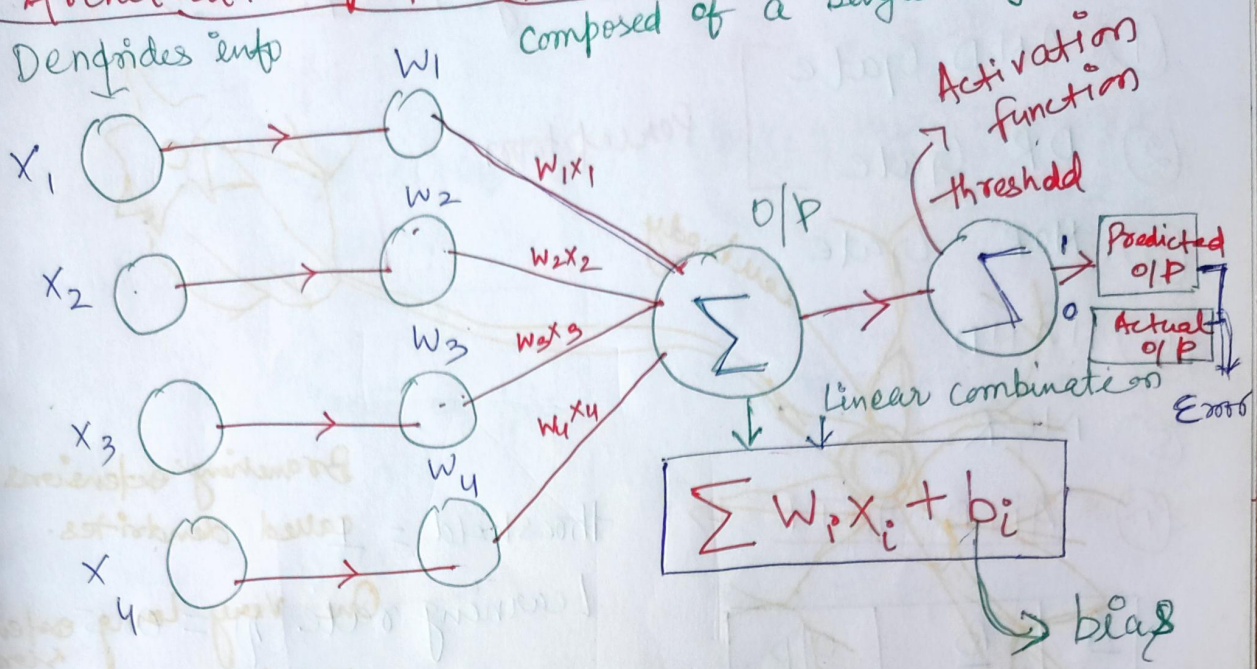
1

LC > 1 → O/P → 1

LC < 0 → O/P → 0

Fully connected layer also called dense layer.

Architecture of Perceptron ⇒ A perceptron is simply composed of a single layer of TLUs.



⇒ Difference between Predicted O/P & Actual O/P = Error.

⇒ If LC < threshold or 0 ⇒ Weight is updated by ~~data~~ ~~to~~ error.

* **Weight Updation Formulae ⇒ [Training]**

$$W_{new} = W_{old} + \eta (Target\ O/P - Pred.\ O/P) * X_i$$

$$Error = Target\ O/P - Pred.\ O/P$$

η → Learning rate [0 to -1]

X_i → Input

* Perceptron also can be multilayer.

* It can be also called threshold logic unit.

⇒ Logical Gates solved by Perceptron ⇒

1. AND Gate
2. OR Gate
3. NOR Gate
4. NAND
5. ~~XNOR~~

Perceptron

(1) AND GATE →

threshold = 1

Learning rate (η) = 0.5

$W_1 = 0.2$

$W_2 = 0.6$

	A	B	O/P
Case 1	0	0	0
" 2	0	1	0
" 3	1	0	0
" 4	1	1	1

(1) case I :- A = B = O/P = 0 (Actual)

$W_1X_1 + W_2X_2 > 1 \longrightarrow 1$

$\searrow 0$

$X_1 = A = 0$

$X_2 = B = 0$

$W_1X_1 + W_2X_2 = 0 < 1 \Rightarrow O/P = 0$ (Predicted)

Actual O/P = 0

error = 0 − 0 = 0

$W_{new} = W_{old} - \eta(0) * X_i$

$W_{new} = W_{old}$.

(2) Case - II

A = 0, B = 1, O/P = 0

$W_1X_1 + W_2X_2 = 0 \times 0.2 + 1 \times 0.6$

$= 0.6 < 1$

Predicted O/P = 0

Error = 0

$\therefore W_{new} = W_{old}$

③ Case 4 :-

A = B = 1 = O/p = 1 ← Actual O/P

$w_1x_1 + w_2x_2 = 1.2 \times 1 + 0.6 \times 1$

$= 1.8 > 1$

↓

Predicted O/p = 1

Error = 1 − 1 = 0

∴ $w_{new} = w_{old}$

---

here in case ③

↓

w_1 new → 1.2 → 0.7

④ Case 3 :

A = 1, B = 0, O/P = 0

$w_1x_1 + w_2x_2 = 1.2 \times 1 + 0.6 \times 0$

$= 1.2 > 1$

↓

O/P = 1

Error = 0 − 1 = −1

↓

Update weight

$w_{new} = w_1 + \eta(-1) * 1$

w_1 new $= 1.2 + 0.5(-1) * 1$

$= 1.2 - 0.5$

w_1 new = 0.7

w_2 new $= 0.6 + 0.5 \times -1 \times 0$

w_2 new = 0.6

Advantages

- Single Layer Perceptrons can learn only linearly separable patterns.
- Easy to implement
- Low computation Power
- Perceptrons can implement Logic Gates like AND, OR

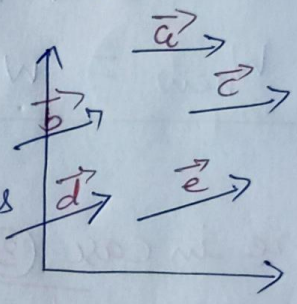
Disadvantages

- Perceptron networks have several limitations.
- the output of a Perceptron can take on only one of two values (0 or 1) due to the hard-limit activation function.
- Here we just use the step function as an activation
- Perceptrons can only classify linearly Separable set of vectors.

Mathematical concepts :-

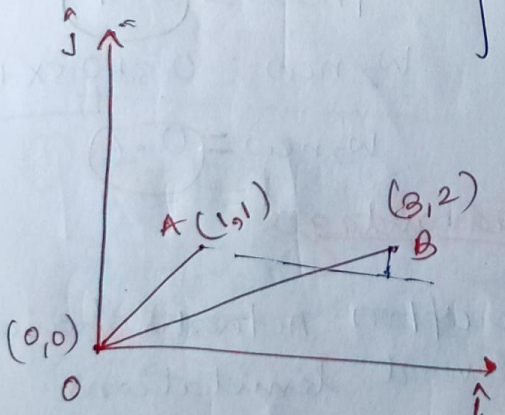
1. Vectors
2. Differentiation
3. Partial Differentiation
4. Gradient
5. Maxima and Minima

Vectors :→ Any object which have two characteristic one of magnitude and 2nd is Direction called Vectors.



→ if magnitude & direction of any vectors are same then those are equal.

$\vec{d} = \vec{e}$ } Magnitude & dirn same.



$\overrightarrow{OA} = 1\hat{i} + 1\hat{j}$

$\overrightarrow{OB} = 3\hat{i} + 2\hat{j}$

$\overrightarrow{AB} = \overrightarrow{OB} - \overrightarrow{OA}$

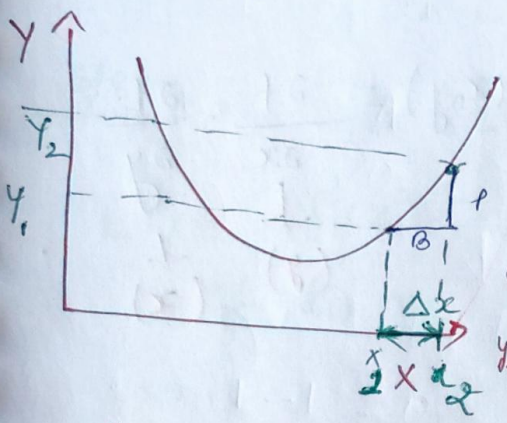
$(3\hat{i}+2\hat{j}) - (1\hat{i}+1\hat{j})$

$\boxed{\overrightarrow{AB} = 2\hat{i} + 1\hat{j}}$

$|\overrightarrow{AB}|$ Magnitude $= \sqrt{2^2+1^2} = \sqrt{5}$

Scaler	Vector
① Only magnitude exists	① Magnitude & direction both exists.
Ex :- Speed	Ex :- Velocity, Gravity Displacement etc.

* Differentiation ⇒ Rate of change of variable in any function.



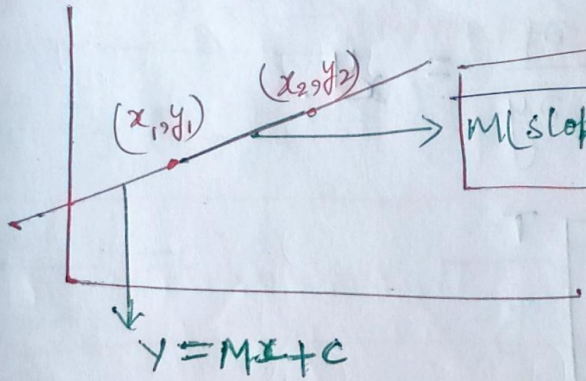
$y_1 = f(x_1)$

$y_2 = f(x_2)$

$$\frac{\text{Perpendicular}}{\text{Base}} = \tan\theta$$

(1) $\frac{\Delta y}{\Delta x} = \frac{dy}{dx}$

(2) $\tan\theta = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_2}{x_2 - x_1}$



$M(\text{slope}) = \frac{y_2 - y_1}{x_2 - x_1}$

(3) $\frac{f(x_2) - f(x_1)}{\Delta x}$

(4) $\lim_{x \to 0} \frac{f(x_1 + \Delta x) - f(x_1)}{\Delta x}$

$[x_2 = x_1 + \Delta x]$

$\therefore$ Differentiation $= \lim_{x \to 0} \frac{f(x_1 + \Delta x) - f(x_1)}{\Delta x}$

$y = x^5$

$\frac{dy}{dx} = ?$

(1) Use Power Rule

$$y = x^n, \quad \frac{dy}{dx} = nx^{n-1}$$

Ex: $\frac{dy}{dx} = 5x^4$

(2) Product Rule

$$f(x)\,g(x) = f'(x)g(x) + f(x)\cdot g'(x)$$

Ex: $f(x) = (x^4 + 2)\cdot\cos x$

$\frac{dy}{dx} = 4x^3\cos x + (x^4 + 2)(-\sin x)$

$= 4x^3\cos x - x^4\sin x - 2\sin x$

③ Partial Differentiation ⇒

$f(x,y,z) = \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$\downarrow$ $\quad$ $\downarrow$ $\quad$ $\downarrow$ $\quad$ $\downarrow$

$(y,z$ const$)$ $\quad$ (x,z) $\quad$ (x,y)

$f(x,y) = \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}$

$\downarrow$ $\quad$ $\downarrow$

(y) $\quad$ (x)

Ex :- $f(x,y) = x^4 y$

① $\frac{\partial f}{\partial x} = 4x^3 \cdot y$. ② $\frac{\partial f}{\partial y} = x^4$.

④ Gradient ⇒

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix}^T$$

* Partial derivatives are written in vector format.

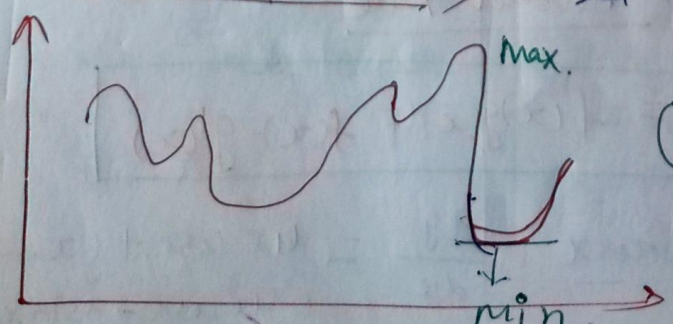
$$\nabla f = \begin{bmatrix} 4x^3 y & x^4 \end{bmatrix}^T = \begin{bmatrix} 4x^3 y \\ x^4 \end{bmatrix}$$

$f(x,y) = 2x^2 + 4y$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 4x \\ 4 \end{bmatrix}$$

⑤ Maxima & Minima ⇒ Used by → Error → Min

Profit → Max.



① $y = f(x)$ → Univariate

step 1 : $f'(x) = 0$

step 2 : find critical points (a), (b)

step 3 : $f(x) = x^5$

$f'(x) = 5x^4$

$f''(x) = 20x^3$

case 1: $f''(x) > 0 \rightarrow$ Minima

case 2: $f''(x) < 0 \rightarrow$ Maxima

case 3: $f''(x) = 0 \Rightarrow$ Point of inflection

* If $f(x) \rightarrow$ Bivariante $\rightarrow$

$y = f(x, y)$

(1) $\frac{\partial f}{\partial x} = 0 \quad , \quad \frac{\partial f}{\partial y} = 0$ $\rightarrow$ find critical points

$f(x)$ $\qquad$ $f(y)$

(2) f_{xx} $\qquad$ f_{yy}

$\frac{\partial^2 f}{\partial x^2} = f_{xy} \quad \frac{\partial^2 y}{\partial y^2} = \rightarrow \left(\frac{\partial f}{\partial x}\right) \rightarrow (x, y) \rightarrow \frac{\partial f}{\partial y}$ new

$f_{xx} = $

(3) Critical Point $(a, b) \rightarrow f_{xx}, f_{yy}, f_{xy}$ $\rightarrow$ Check Minima or Maxima.

case 1 $\rightarrow$ $rt - s^2 > 0 \rightarrow$ Maxima/Minima

$r > 0 \rightarrow$ Local min. $r < 0$ Local Maxi

case 2 $\rightarrow$ $rt - s^2 = 0 \rightarrow$ Point of reflection ~~inflection~~

Case-3 $\rightarrow$ $rt - s^2 < 0 \rightarrow$ neither max nor min saddle Pt.

$r > 0 \rightarrow$ Local Minima
$r < 0 \rightarrow$ Local Maxima

* **Activation Functions** ⇒

The purpose of the activation function is to introduce non-linearity into the output of a neuron. the activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Ex :- XOR operation

* **Activation functions are:** There are most used activation functions.

(1) Sigmoid
(2) RELU (Rectified Linear unit)
(3) Leaky RELU
(4) tanh (Hyperbolic Tangent)
(5) Soft max
(6) Parametic RELU

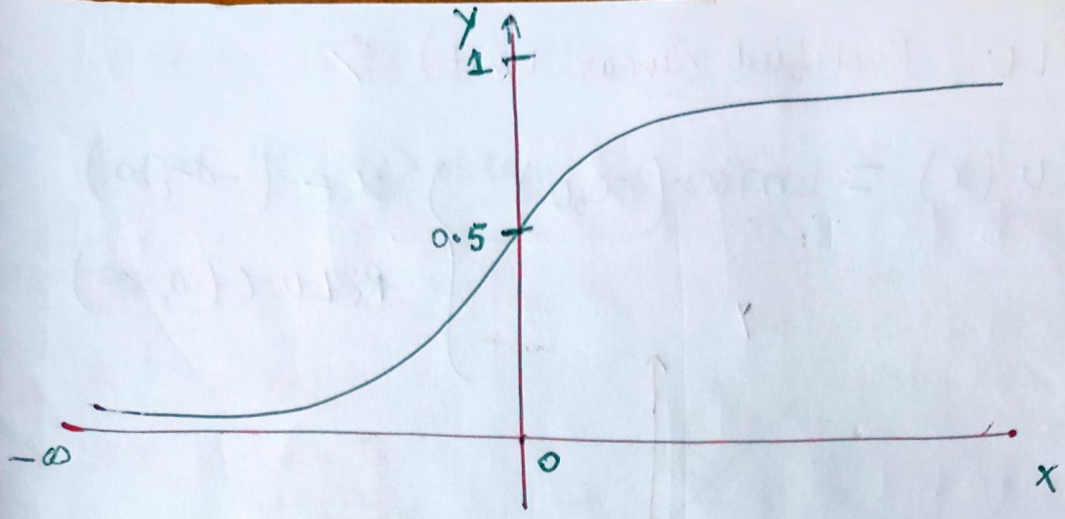
$W_i(x) + b(1)\, a(1)$

(1) **Sigmoid function** ⇒ It is used in binary classification

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad ; \quad x \in (-\infty, \infty)$$

$$\sigma(x) \in (0,1)$$

In this function, output belongs to $[0,1]$



* We would need to understand the derivative of every activation function, Because derivative helps our NN models to optimise Weights for our NN.

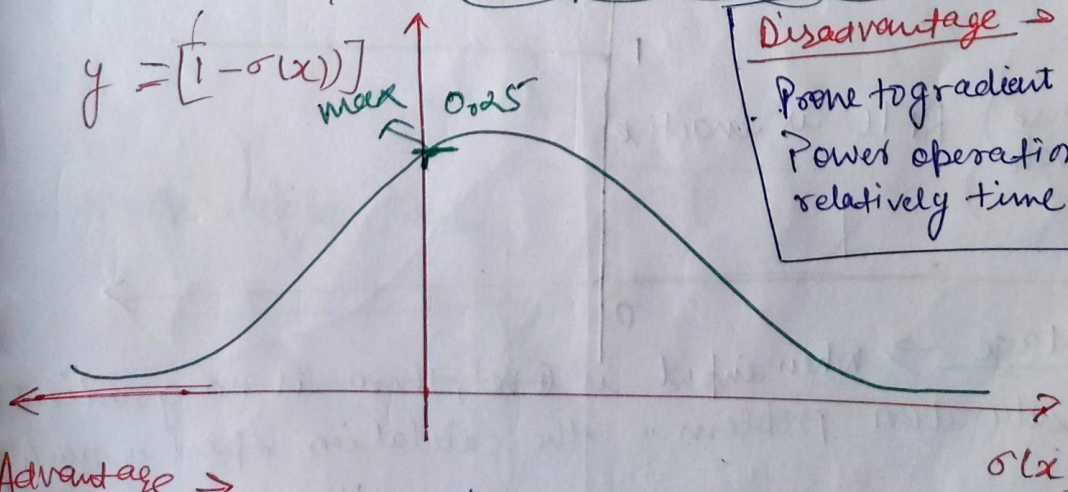
Derivative of sigmoid function: →

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

$$\sigma'(x) = \sigma(x)\,(1-\sigma(x))$$ $\left(\frac{u}{v}\right)' = \frac{u'v - uv'}{v^2}$

Derivative {0,1} Graphical Structure {0,1} y

$y = [1-\sigma(x)]$ max 0.25



Disadvantage →
- Prone to gradient vanishing
- Power operations are relatively time consuming

Advantage → derivative of sigmoid

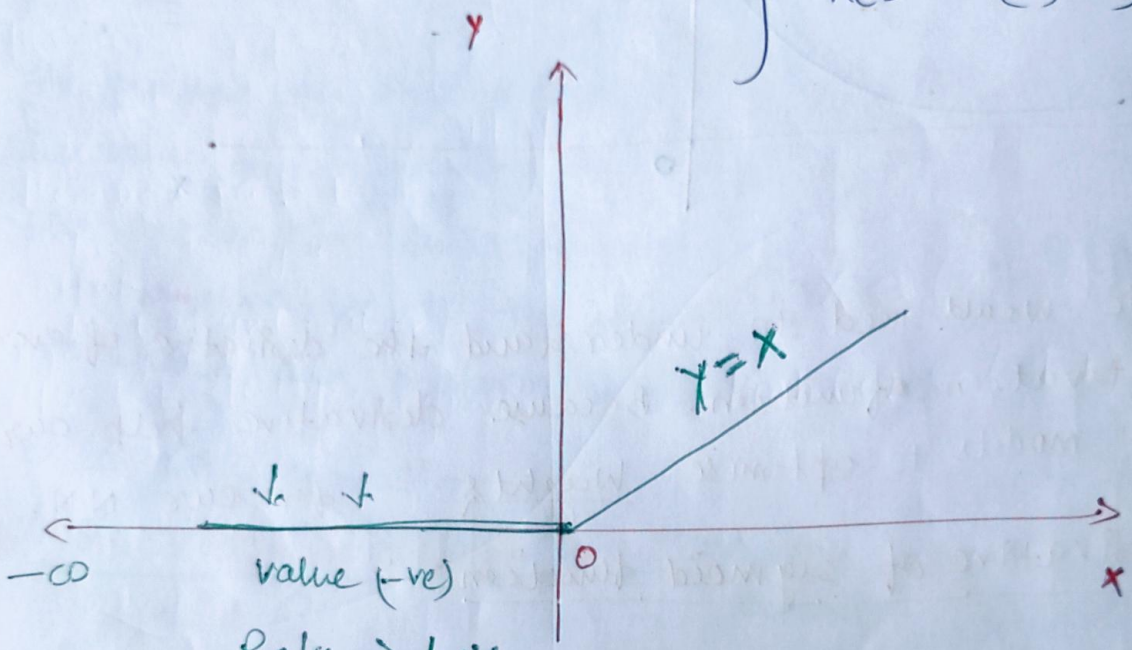
→ Smooth Gradient, Preventing "jumps" in output values.

(2) RELU (Rectified Linear Unit) ⇒

$$Relu(x) = \max(x, 0)$$

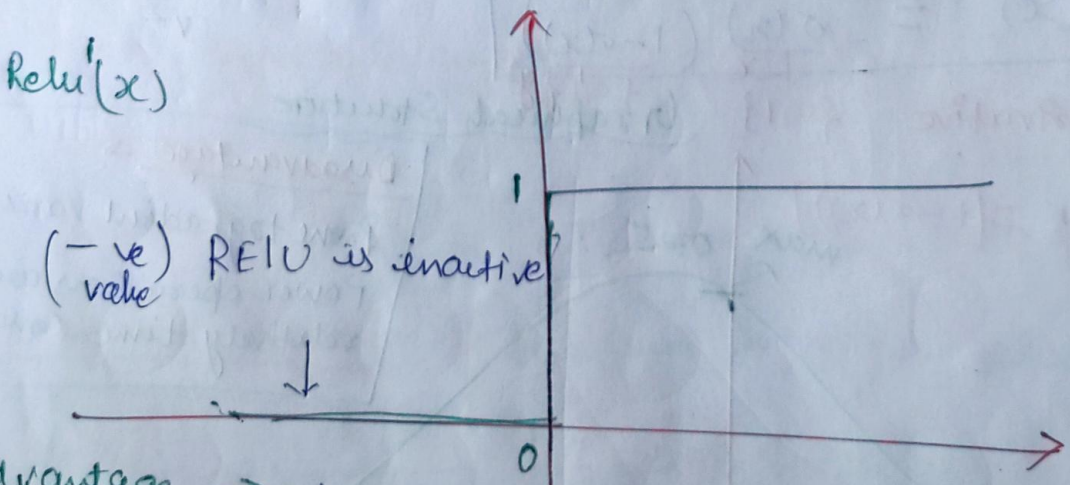
$x \in (-\infty, \infty)$

$ReLU \in (0, \infty)$



Relu → fail or Dead (saturation is not found)

Derivative of Relu →



Advantage → When input is (+ve), there is no gradient saturation problem. the calculation speed is much faster. (than Sigmoid & tanh)

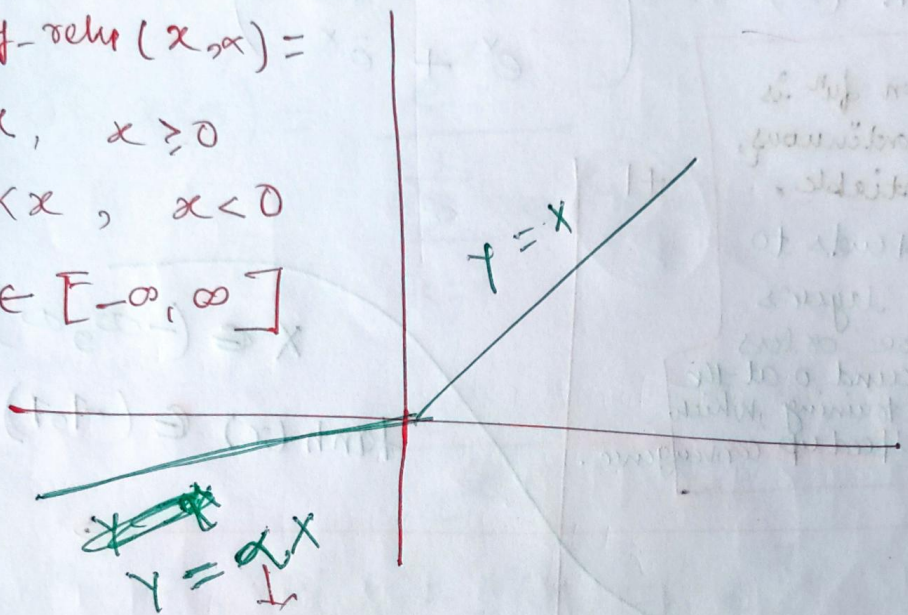
Disadvantage → When the input is −ve, Relu is completely inactive

(3) LEAKY RELU → It is special case of Parametric RELU.

→ Dead RELU problem is solved.

$$\text{Leaky\_relu}(x, \alpha) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$$

$x \in [-\infty, \infty]$

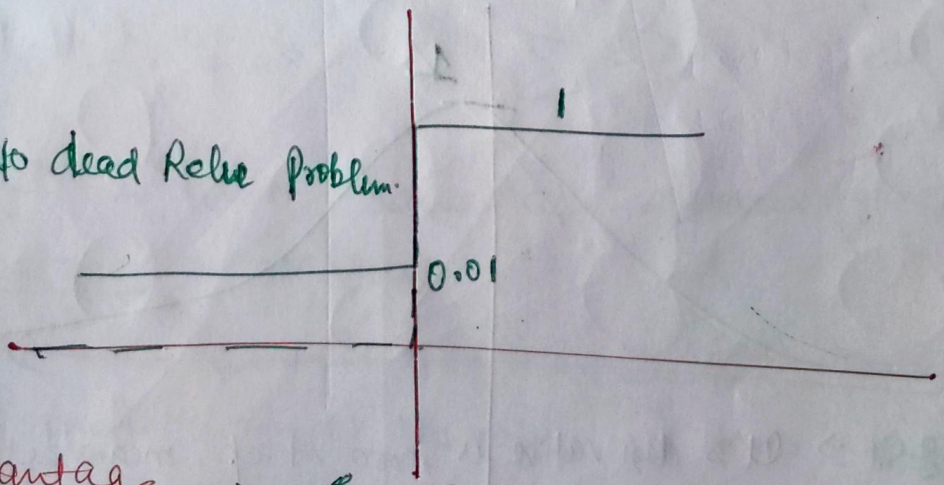


If $\alpha = 0.01$ → Leaky Relu

if $\alpha \in$ (other than 0.01) → Randomized Relu

if α = Learnable → Parametric Relu.

Derivative of Leaky Relu →

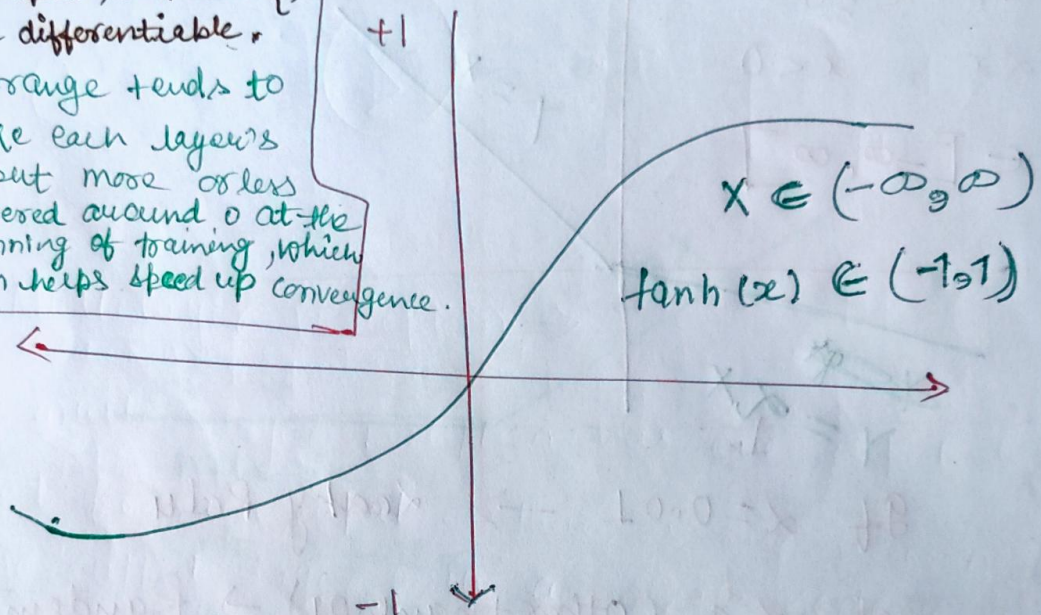


Advantage → Gradient vanishing problem is not present.

(4) Hyperbolic tangent activation function (tanh)

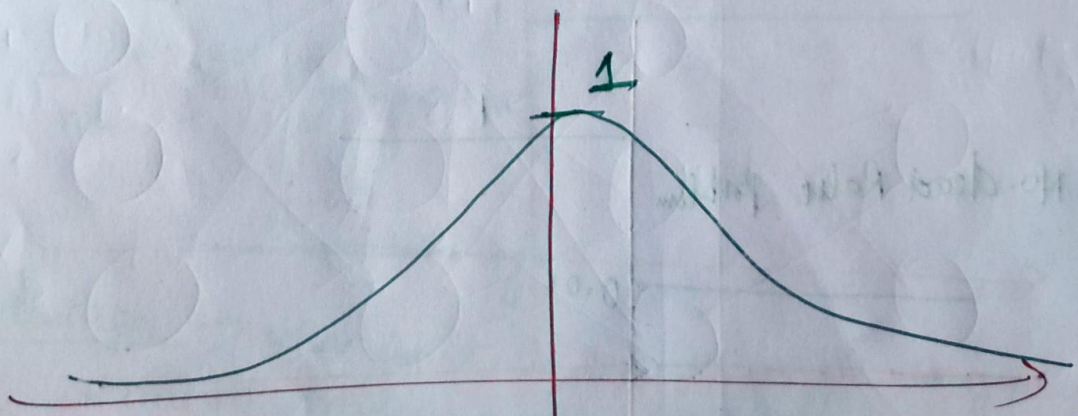
$$\tanh(x) = \left(\frac{e^x - e^{-x}}{e^x + e^{-x}} \right)$$

This activation fun is S-shaped, continuous, and differentiable. The range tends to make each layer's output more or less centered around 0 at the beginning of training, which often helps speed up convergence.



→ Now steep curve is found than Sigmoid fun.

$$\tanh'(x) = 1 - \tanh^2(x)$$



Advantage → It's derivative is more steep, means it gets more value.
→ It has a wider range for faster learning and grading

Disadvantage → More computation expensive.

(5) Soft Max activation function :→

* → It is used mostly in multiclass problem.

$$S(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$$

If $K = 1$,

$$\frac{e^x}{e^x} = 1$$

$K = 2$, $P(X = 1/x) = \frac{e^{x_1}}{e^{x_1} + e^{x_2}}$

$$= \frac{1}{1 + e^{-(x_2 - x_1)}}$$

If multiple class →

Case 1 :

$$\frac{e^{x_1}}{e^{x_1} + e^{x_2} + e^{x_3}}$$

Case 2 :

$$\frac{e^{x_2}}{\sum e^{x_i}}$$

Case 3 :

$$\frac{e^{x_3}}{\sum e^{x_3}}$$

→ It is also same as sigmoid function.

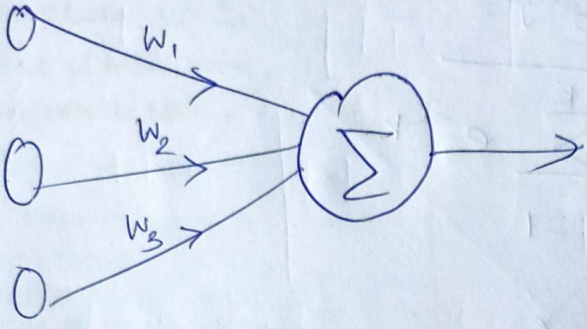
Advantage →

the main advantage of using Softmax is the output Probabilities range. the range will 0 to 1, and the sum of all the probabilities will be equal to one.

Disadvantage → Vanishing gradient problem

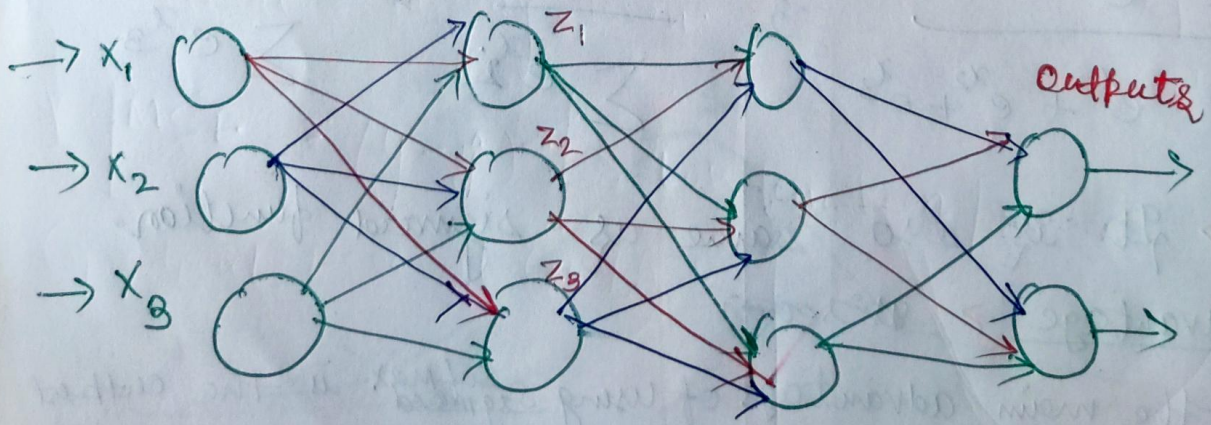
* Forward Propagation →

Information is fed through a network, in a forward direction, to generate an output.



* Multilayer Perceptron →

A multi-layer Perceptron has one input layer and for each input, there is one neuron, it has one output layer with a single node for each output and it can have any no. of hidden layers and each hidden layer can have any no. of nodes.



$x_1 \rightarrow (w_{a1}x_1 + w_{b1}x_2 + w_{c1}x_3)$

$\downarrow$

z_1

$z_1 \rightarrow w_{a1}z_1 + w_{b2}z_2 + w_{c1}z_3$

Backward Propagation ⇒ It is able to compute the gradient of the network's error with regard to every single model parameter.

Back propagation requires a known, desired output for each input value in order to calculate the loss function gradient - that how a prediction differs from actual results - (as a type of supervised ML).

Chain Rule ⇒

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

Let $p = -x$ $\quad P(x)$

$q = 1 + e^{-x} \quad = q(p)$

$q = 1 + e^{p}$ ⟵ $f(q) = \frac{1}{q}$

$\downarrow$

$f(q(p(x)) \rightarrow$ Sigmoid function

$$\frac{df}{dq} = q^{-1} = (-1) \times q^{-1-1} = \frac{-1}{q^2}$$

$$\frac{dq}{dp} = 0 + e^{p} = e^{p}$$

$$dp/dx = -1$$

$$\frac{df}{dq} \times \frac{dq}{dp} \times \frac{dp}{dx} = \frac{df}{dx}$$

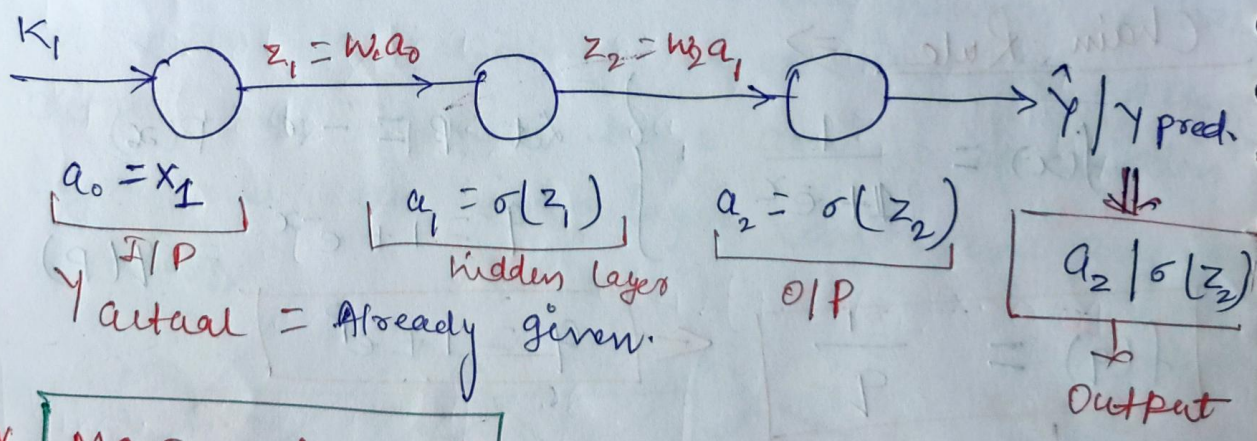
$$= \frac{e^p}{q^2} = \frac{e^{-x}}{[1+e^{-x}]}$$

$$\Rightarrow \boxed{1 + e^{-x} = \frac{1}{1+e^{-x}}\left[1 - \frac{1}{[1+e^{-x}]}\right]}$$

$$\sigma(x) = \sigma(x)\,[1 - \sigma(x)]$$

* **Back Propagation** ⇒ using Chain Rule

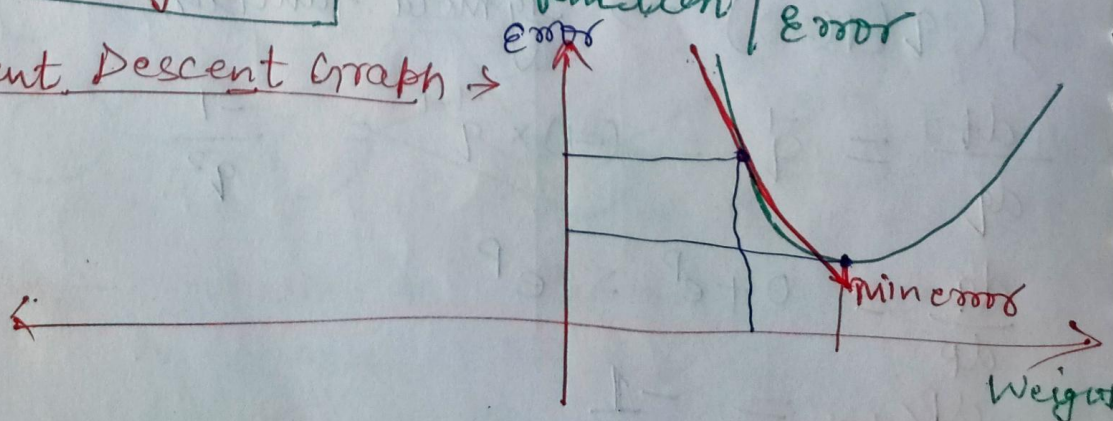
In Simple NN ⇒



Y_{actual} = Already given.

* $\boxed{MSE = (y - \hat{y})^2}$ ⇒ Loss function / Error

Gradient Descent Graph →

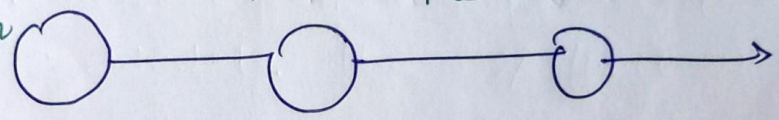


$$W_{new} = W_{old} - \eta \frac{\partial E}{\partial w}$$

Gradient Descent weight formulae.

$\eta \rightarrow$ learning rate
$\in (0, 1)$

Now → We will go backward and find weight
→ Back propagation Algo first makes a prediction (forward pass) and measures the algo, then goes through each layer in reverse to measure the error contribution from each connection and finally tweaks the connection weights to reduce the error.



W_2 update :

$$W_{2\,new} = W_{2\,old} - \eta \frac{\partial E}{\partial W_2}$$

$$\frac{\partial E}{\partial W_2} = \frac{\partial E}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial W_2}$$

① $E = (y - \hat{y})^2$

$$= -2(y - a_2)(\sigma(z_2)(1 - \sigma(z_2)) \times a_1$$

$$W_{2\,new} = W_{2\,old} - \eta \frac{\partial E}{\partial W_2}$$ → find

~~$W_{2\,new} = W_{2\,old} - \eta \times$~~

① $Error = (y - \hat{y})^2$

② $= -2(y - a_2)$
$\hat{y} = a_2 = \sigma(z_2)$
$\sigma(z_2)(1 - \sigma(z_2))$
$\hat{y} = \sigma(z_2)$
$\hat{y} = \sigma(W_2 a_1)$

$Error = (y - a_2)^2$

$Error = f(a_2)$

$Error = f(z_2)$

③ $z_2 = W_2 a_1 \rightarrow f(W_2)$
a_1

w_1 update →

(1) Error $= (y-\hat{y})^2 = (y-a_2)^2 \rightarrow f(a_2)$

(2) $z_2 = w_2 a_1 \Rightarrow f(a_1)$

(3) $a_1 = \sigma(z_1) = f(z_1)$

(4) $a_2 = \sigma(z_2) \Rightarrow f(z_2)$

(5) $z_1 = w_1 a_0 = f(w_1)$

(6)

$$w_{inew} = w_{iold} - \eta \frac{\partial E}{\partial w_1}$$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial w_1}$$

$$= \frac{\partial E}{\partial w_2} \times w_2 \times \sigma(z_1)(1-\sigma(z_1)) * 0.01$$

$$\boxed{w_{inew} = w_{old} - \eta \frac{\partial E}{\partial w_1}}$$