* Deep Learning

- **Perceptron :-**

1) Single Layer Neural Network

→ This is a type of neural Network that performs binary classification that maps input features to output decision.

→ **Types :**

1) **Single Layer perceptron :-**

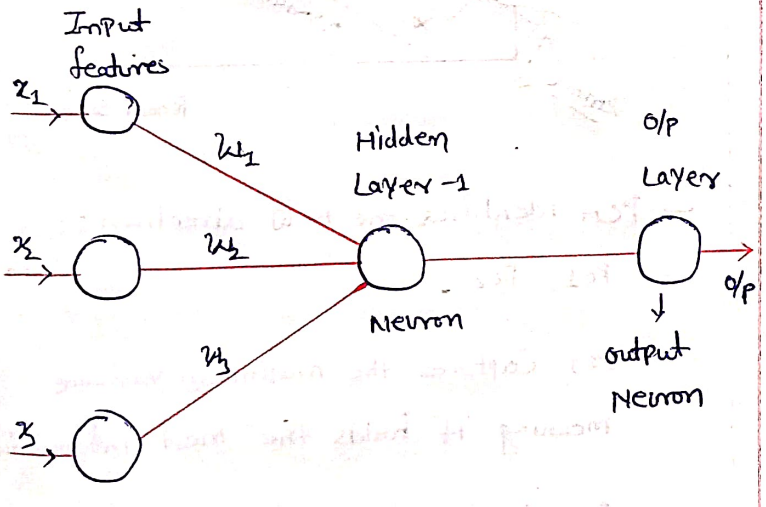
→ This is Limited to learning linearly separable patterns.

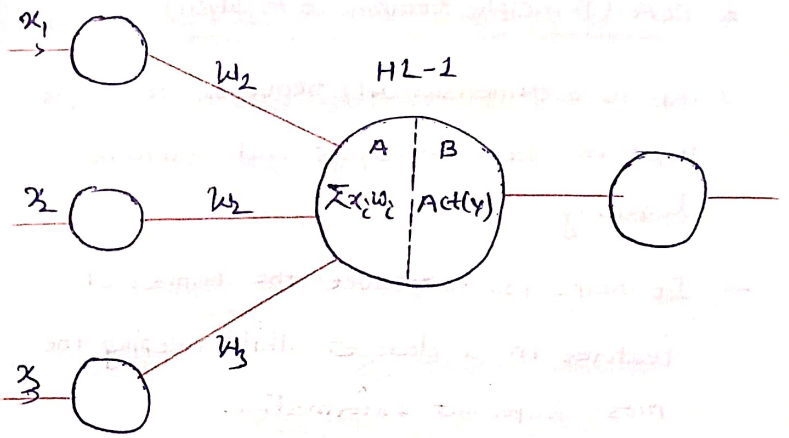
→ It Consists single Layer of i/p Nodes that are fully Connected to a layer of output Nodes.

2) **Multi - Layer perceptron :-**

→ Consists two or more Layers.

→ Adept at handling more Complex patterns.





Weighted sum -

$$z = \Sigma x_i w_i = w_1 x_1 + x_2 w_2 + \dots x_n w_n$$

$$= w^T x$$

→ Now Weighted sum passes through a Activation fn that classify the o/p.

$$h(z) = \begin{cases} 0, & z < \text{Threshold} \\ 1, & z > \text{Threshold} \end{cases}$$

↓

heaviside step fn

→ To Enhance flexibility we use bias that helps perceptron to do adjustment independent of input.

$$z = \Sigma x_i w_i + b$$

↳ bias

∵ for binary classification we Can use sigmoid fn -

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

$$= \frac{1}{1+e^{-(\Sigma x_i w_i + b)}}$$

→ This Complete process # is called forward propagation.

→ during training, perceptron's weights are adjusted to minimize the difference b/w the predicted output and Actual output.

So To update the weights Back propogation happens.

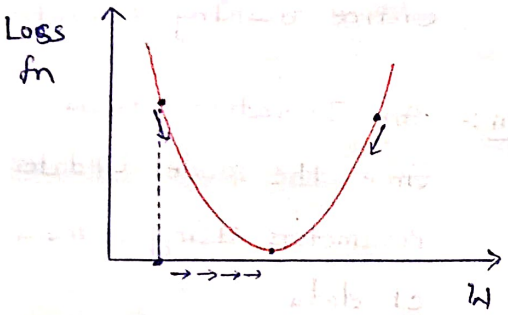
Here we use optimizers.

* Back Propogation –

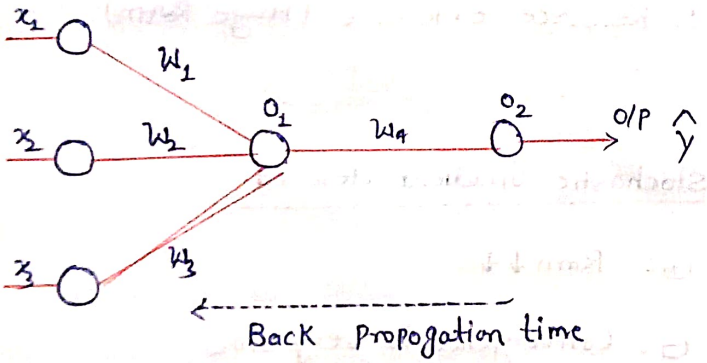
1) Weight updation formula –

$$W_{new} = W_{old} - \eta \frac{dL}{dw_{old}}$$

η → Learning Rate



2) Chain Rule of differentiation:-



$$w_{4new} = w_{4old} - \eta \frac{\partial L}{\partial w_{4old}}$$

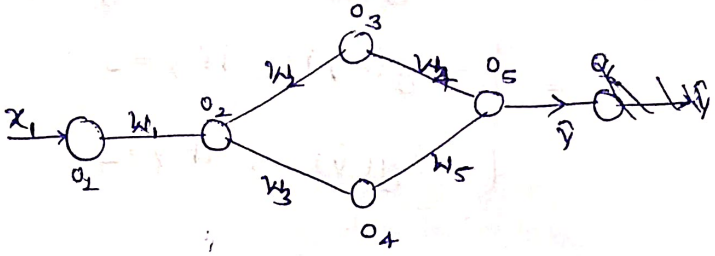
$$\frac{\partial L}{\partial w_{4old}} = \frac{\partial L}{\partial o_2} * \frac{\partial o_2}{\partial w_{4old}}$$

$\because z_{o_2} = o_1 w_4 + b_2$

$$\boxed{b_{2new} = b_{2old} - \eta \frac{\partial L}{\partial b_{2old}}}$$

$$\Rightarrow w_{1new} = w_{1old} - \eta \frac{\partial L}{\partial w_{1old}}$$

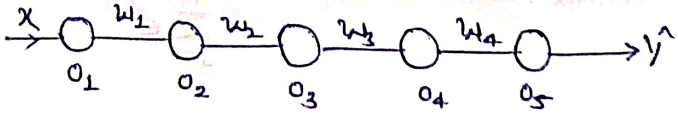
$$\frac{\partial L}{\partial w_{10}} = \frac{\partial L}{\partial o_2} * \frac{\partial o_2}{\partial o_1} * \frac{\partial o_1}{\partial w_{10}}$$



$$\frac{\partial L}{\partial w_{10}} = \left(\frac{\partial L}{\partial o_5} * \frac{\partial o_5}{\partial o_3} * \frac{\partial o_3}{\partial o_2} * \frac{\partial o_2}{\partial w_{10}}\right) + \left(\frac{\partial L}{\partial o_5} * \frac{\partial o_5}{\partial o_4} * \frac{\partial o_4}{\partial o_2} * \frac{\partial o_2}{\partial w_{10}}\right)$$

3) Vanishing Gradient problem :

→ This occurs when gradients become extremely small during back propogation, causing early layers to learn very slowly or stop Learning.



$$o_5 = \sigma[(o_4 * w_4) + b]$$

↓

Sigmoid fn

$\because \quad 0 < \sigma(y) < 1$

$0 < \sigma'(y) < 0.25$

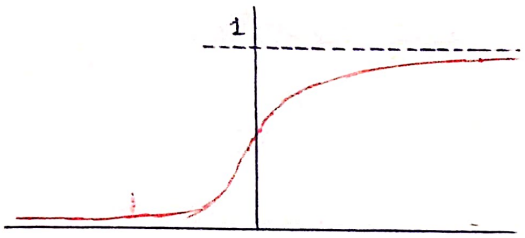
$\frac{\partial L}{\partial W_{1n}} \longrightarrow 0$

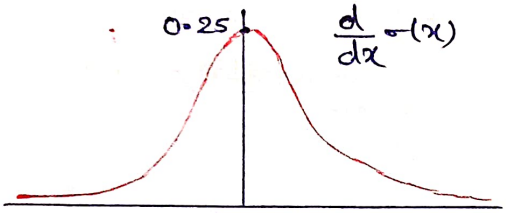
$W_{1n} \approx W_{10}$

. Commonly used Activation fn :-

1) sigmoid function

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





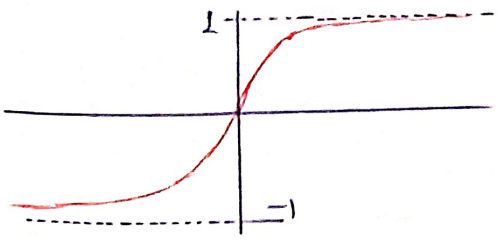
⊖: 1) Prone to gradient ~~des~~ vanishing

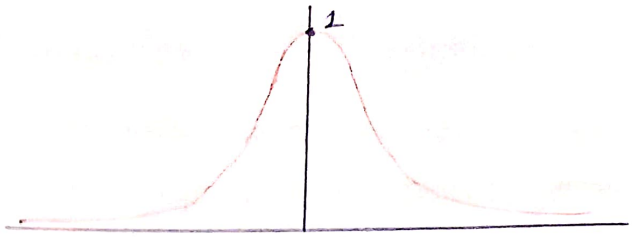
2) Function o/p is not Zero centered

3) Power operation are relatively time Consumming

2) tanh function :-

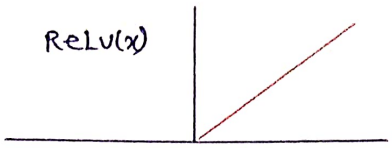
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

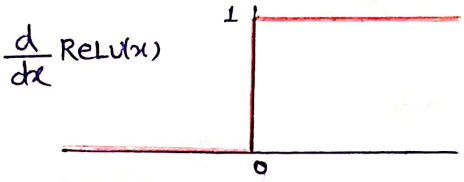




3) ReLU function :- (Rectified Linear unit)

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





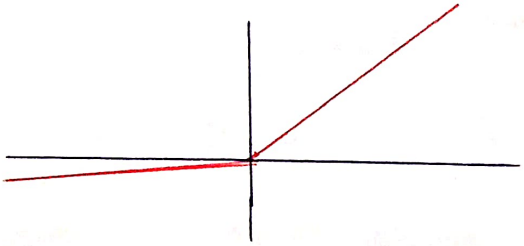
dead Neuron Problem

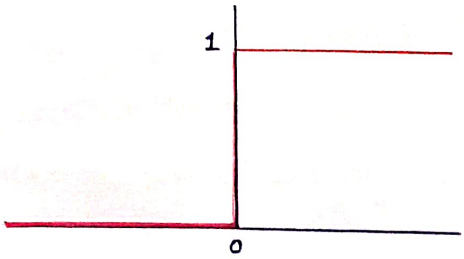
⊖: 1) Not a zero centric

2) When the input is -ive, ReLU is Completely Inactive.

4) Leaky ReLU function-

$$f(x) = \max(0.01x, x)$$

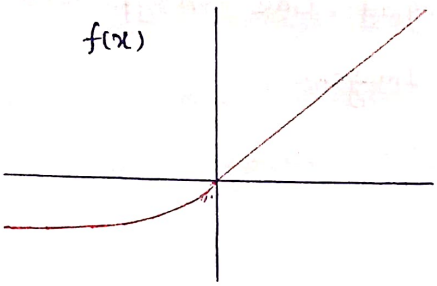


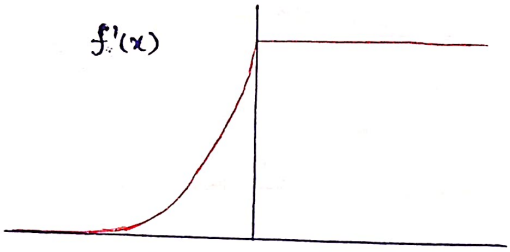


→ Solve the dead ReLU Problem.

5) ELU (Exponential Linear units)

$$f(x) = \begin{cases} x & , \text{ if } x > 0 \\ \alpha(e^x - 1) & , \text{ otherwise} \end{cases}$$



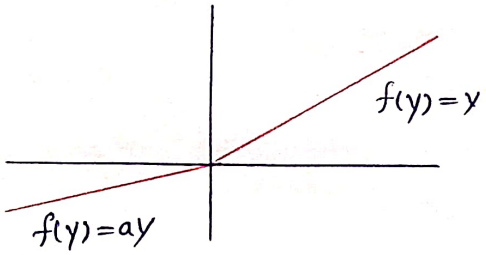


→ No dead ReLU issue

→ Zero centered

6) PReLU (Parametric ReLU)

$$f(y) = \begin{cases} y & , \quad y > 0 \\ ay & , \quad y \leq 0 \end{cases}$$



if $a = 0 \Rightarrow$ ReLU

if $a > 0 \Rightarrow$ Leaky ReLU

if a is a learnable parameter,
f becomes PReLU

In Hidden Layer We use –

1) ReLU
2) PReLU
3) ELU

In output Layer We Can use –

1) sigmoid } Binary classification
2) softmax }

3) Linear Activation fn } Regression

* Loss function :-

A) Regression :-

1) mean squared Err (MSE)

$$\text{Cost fn} = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

⊕ : 1) diffrentiable fn
2) It has only one local / Global minima
3) Converge faster

⊖ : 1) Not Robust to outliers.

2) mean Absolute Err (MAE)

$$\text{Cost fn} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

⊕ : 1) Robust to outliers.

⊖ : 1) Not diffrentiable

3) Huber Loss

$$\text{Loss} = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & , \text{if } |y-\hat{y}| \le \delta \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2 & , \text{otherwise} \end{cases}$$

B) Classification :-

1) Cross Entropy :-

- Binary Cross Entropy -
  ↳ for Binary Classification

$$\text{Loss} = -y * \log(\hat{y}) - (1-y) * \log(1-\hat{y})$$

$$= \begin{cases} -\log(1-\hat{y}) & , \text{if } y=0 \\ -\log(\hat{y}) & , \text{if } y=1 \end{cases}$$

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

- Categorical Cross Entropy -
  ↳ for multiclass Classification

$$y_i = [y_{i1}, y_{i2}, y_{i3} \ldots\ldots y_{ic}]$$

$$\text{Loss} = -\sum_{j=1}^{c} y_{ij} * \ln(\hat{y}_{ij})$$

Where — $\hat{y}_{ij}$ = Softmax Activation
↳ o/p Layer

Softmax Activation $\sigma(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

* optimizers :-

1) Gradient descent

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

- Batch :- A batch refers to a subset of the entire training dataset that is processed at once during a single forward and backward pass through the neural network.

- Epoch :- An Epoch signifies the Completion of one full cycle through the entire training dataset.

- Iteration :- An Iteration occurs each time the model updates its parameters using a mini-batch of data.

→ disadvantage of Gradient descent

1) Resource Extensive (Huge RAM)

2) Stochastic Gradient descent

⊕ : Ram ↓↓

⊖ : Convergence is very slow

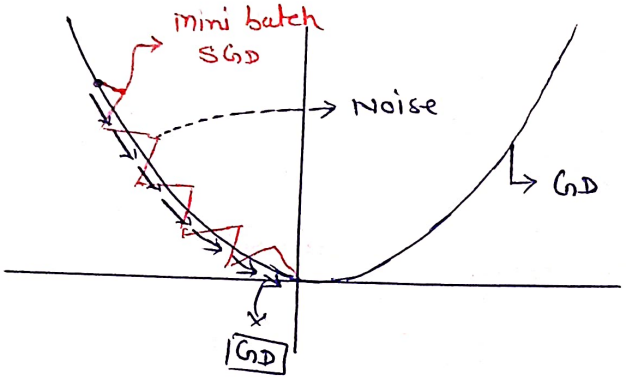
→ In this We train model ~~by~~ ~~the~~ With the 1 ~~batch~~ ~~of~~ records

— Time Complexity ↑↑

3) Mini Batch SGD (Stochastic Gradient descent)

→ In this we select batch size and train model by batches.

⊕ : 1) Resource Intensive
2) Convergence will be better
3) Time Complexity will Improve.



4) SGD With momentum :-

$$W_n = W_0 - \eta \frac{\partial L}{\partial W_0}$$

$$\Rightarrow W_t = W_{t-1} - \eta \frac{\partial L}{\partial W_{t-1}}$$

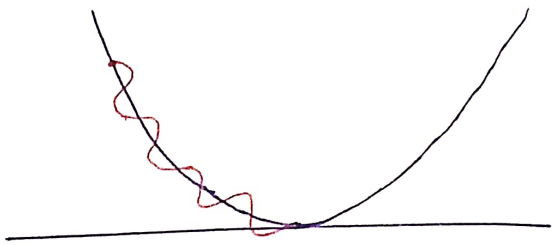
Exponential Weighted Average -

t_1	t_2	t_3	-----	t_n
a_1	a_2	a_3	----	a_n

$$V_{t_1} = a$$

$$V_{t_2} = \beta * V_{t_1} + (1-\beta) * a_2$$

β is a Hyper Parameter that tells the importance of current value



$$W_t = W_{t-1} - V_{dw} * \eta$$

$$V_{dw} = \beta * V_{dw_{t-1}}$$

$$\therefore V_{dw_t} = \beta * V_{dw_{t-1}} + (1-\beta) * \frac{\partial L}{\partial W_{t-1}}$$

⊕ : Reduce Noise and quick Convergence

5) Adagrad : (Adaptive Gradient descent) -

↳ η is not fixed, It is adaptive

$$W_t = W_{t-1} - \eta' \frac{\partial L}{\partial W_{t-1}}$$

$$\eta' = \frac{\eta}{\sqrt{L_{t+\epsilon}}}$$

ϵ : Small Number

$$L_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial W_t}\right)^2$$

→ towards Global minima, η will decrease.

6) Adadelta and Rms prop

$$\eta' = \frac{\eta}{\sqrt{S_{dw}+\epsilon}}$$

$$S_{dw_t} = \beta S_{dw_{t-1}} + (1-\beta)\left(\frac{\partial L}{\partial w_{t-1}}\right)^2$$

Increase by smaller value.

7) Adam optimizer

Momentum + Rmsprop

$$w_t = w_{t-1} - \eta' V_{dw}$$

$$\eta' = \frac{\eta}{\sqrt{S_{dw}+\epsilon}}$$

$$V_{dw_t} = \beta * V_{dw_{t-1}} + (1-\beta)\frac{\partial L}{\partial w_{t-1}}$$

⊕: 1) Smoothening

2) Learning Rate becomes Adaptive

* CNN (Convolutional Neural Network)

↳ It is a deep Learning models designed to process data with a grid-like topology such as Images.