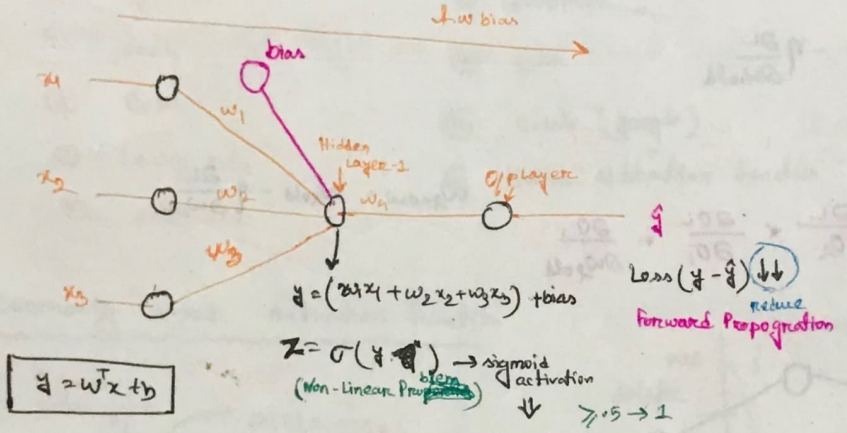
Tutorial-2

Forward Propogration, Loss Functions, Chain Rule of Derivatives

Day-2 : Deep Learning.

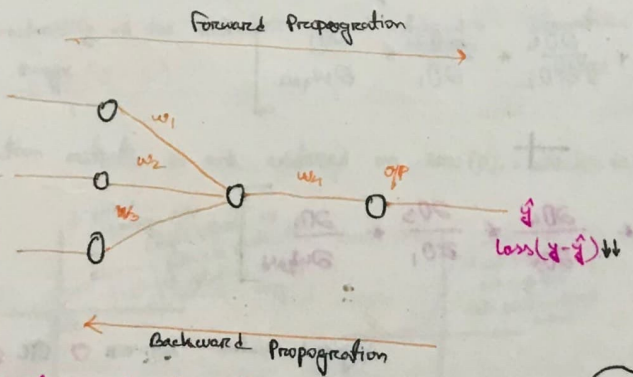
Agenda –

1. Forward Propogration
2. Chain Rule of Derivative
3. Vanishing Gradient Problem
4. Loss function
5. Activation function



Backward Propogration

1. Weight updation formula
2. Chain Rule of Differentiation

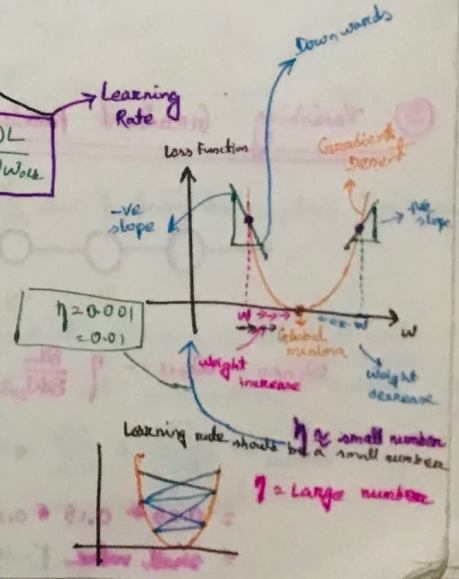


1. **Weight updation formula** –

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

η → Learning Rate

$\frac{\partial L}{\partial W_{old}}$ → slope



$\frac{\partial L}{\partial W_{old}}$ = **-ve slope**

$W_{new} = W_{old} - \eta(-ve)$

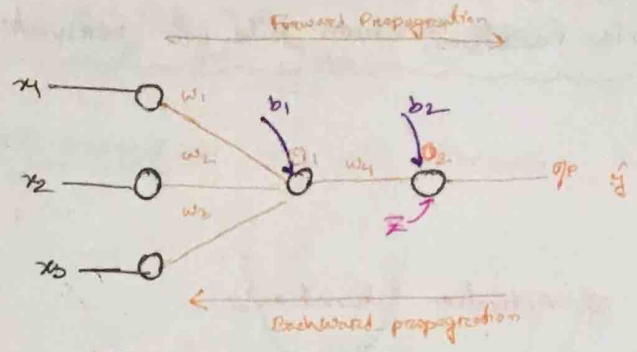
$= W_{old} + \eta(+ve)$

$W_{new} >> W_{old}$

$W_{new} = W_{old} - \eta(+ve)$

$W_{new} << W_{old}$

Chain rule of Differentiation:



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

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

[bias updation, weight updation, learning]

$Z = \sigma(a_1 w_4 + b)$

$$W_{4new} = W_{4old} - \eta \frac{\partial L}{\partial W_{4old}}$$ ← {chain rule of derivative}

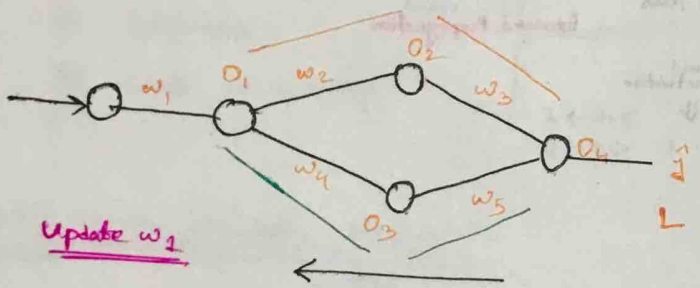
$$\frac{\partial L}{\partial W_{4old}} = \frac{\partial L}{\partial O_2} * \frac{\partial O_2}{\partial W_{4old}}$$

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

$$\frac{\partial L}{\partial W_{1old}} = \frac{\partial L}{\partial O_2} * \frac{\partial O_2}{\partial O_1} * \frac{\partial O_1}{\partial W_{1old}}$$

$$\frac{\partial L}{\partial W_{2old}} = \frac{\partial L}{\partial O_2} * \frac{\partial O_2}{\partial O_1} * \frac{\partial O_1}{\partial W_{2old}}$$

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



Update w_1

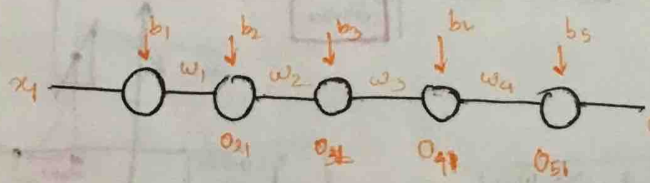
$$W_{1new} = W_{1old} - \eta \frac{\partial L}{\partial W_{1old}}$$

chain rule of derivatives →

$$\frac{\partial L}{\partial W_{1old}} = \left[\frac{\partial L}{\partial O_4} * \frac{\partial O_4}{\partial O_2} * \frac{\partial O_2}{\partial O_1} * \frac{\partial O_1}{\partial W_{1old}}\right] + \left[\frac{\partial L}{\partial O_4} * \frac{\partial O_4}{\partial O_3} * \frac{\partial O_3}{\partial O_1} * \frac{\partial O_1}{\partial W_{1old}}\right]$$

③ Vanishing Gradient Problem:

Sigmoid value between 0 & 1



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

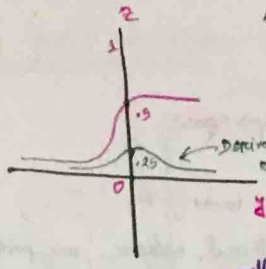
$Loss = \frac{1}{2}(y - \hat{y})^2$ → [MSE (Mean Square Error)]

$O_{51} = \sigma[(O_{41} * W_4) + b_5]$ (σ: sigmoid) 0 to .25

$$\frac{\partial L}{\partial W_{1old}} = \frac{\partial L}{\partial O_{51}} * \frac{\partial O_{51}}{\partial O_{41}} * \frac{\partial O_{41}}{\partial O_{31}} * \frac{\partial O_{31}}{\partial O_{21}} * \frac{\partial O_{21}}{\partial W_{1old}}$$

$= 0.25 * 0.15 * 0.1 * 0.05 * 0.02$ [every hidden neuron derivative loss is decreased]

= small value [multiply with small value result is small]



sigmoid → $a = \frac{1}{1+e^{-z}}$ → 0 or 1

$\geq 0.5 = 1$

$< 0.5 = 0$

Derivative condition

$0 \leq \sigma(z) \leq 0.25$

$z = wx+b$

$z = w^T x + b$

$W_{new} = W_{old} - \eta$ (small number) ← small num.

$W_{new} \approx W_{old}$

↓ Vanishing Gradient

No change in weights

w_{new}	$w_{old} - \eta$ (small)
1000	1000 + .001
1000	999.99
10000	10000

$w_{new} \approx w_{old}$

to solve use Another activation function

There are many another activation function

ⓐ Sigmoid
ⓑ tanh
ⓒ Relu
ⓓ Leaky Relu
ⓔ pre-relu
ⓕ softmax
ⓖ Elu
ⓗ Swish (google)
ⓘ Linear activation function

Commonly Used Activation function

a. Sigmoid :

Formula

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

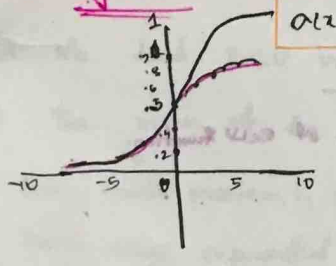


fig - 1.a : Functionality of the activation range (0 - 1)

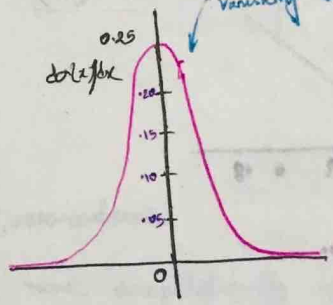
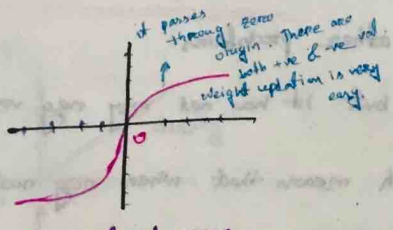
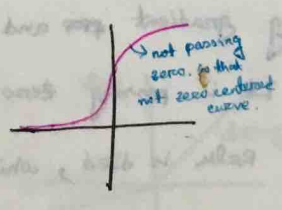


fig - 1.b : Derivatives of activation range (0 - 0.25)

① The function output is not centered on zero (0). which is will reduce the efficiency of weight update.



Zero centered curve.



② The sigmoid function performs exponential operation $\left(\sigma(z) = \frac{1}{1+e^{-z}}\right)$, which is slower for computers. Time increase

Advantages:

① Smooth gradient, preventing 'jumps' in output values.
② Output values bound between 0 and 1. normalizing the output of each neuron.
③ Clear prediction, that is very close to 1 or 0.

Disadvantages:

① Prone to Gradient vanishing. So that doesn't create a DN network. Deep Neural
② Function output is not zero-centered
③ Power operations are relatively time consuming.

★ can't create deep neural network.

b. tanh: hyperbolic

It is a hyperbolic function.

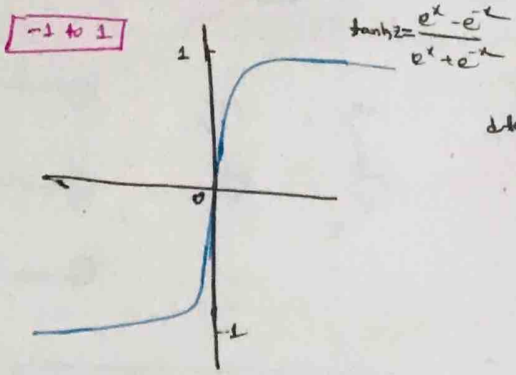


fig- Functionality of tanh

range (-1 to 1)

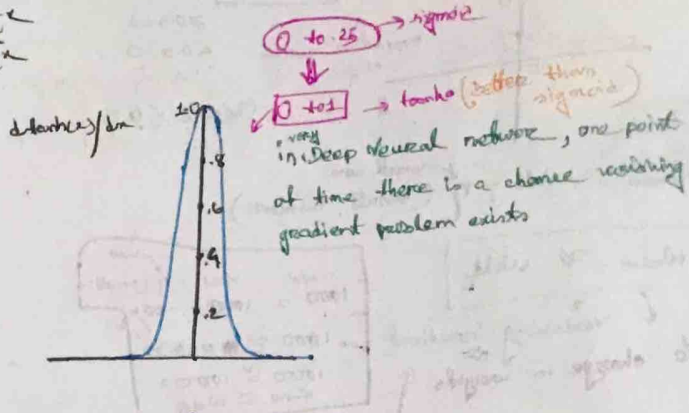


fig: Derivatives of tanh

range (0 to 1)

① Zero centric function, hyperbolic function.

② In binary classification, the tanh function is used the hidden layers and sigmoid is used for the output.

c. ReLU function:

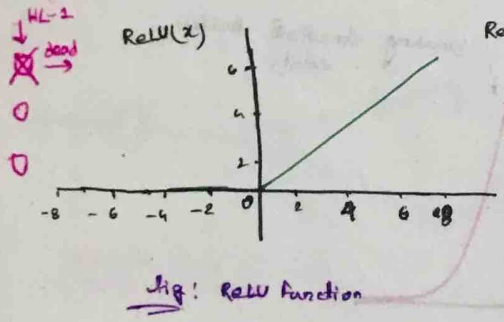


fig: ReLU function

ReLU = max(0,x) → quicker

$\frac{\partial L}{\partial W_{old}} = 0$ → $W_{new} \approx W_{old}$ → vanishing gradient problem

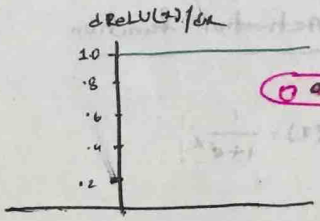


fig: derivatives of ReLU function

① If x is neg the value becomes zero(0).

② When $\frac{\partial L}{\partial W_{old}} = 0$ than $\frac{\partial L}{\partial W_{new}} \approx \frac{\partial L}{\partial W_{old}}$ which is called vanishing gradient problem

③ It is better than sigmoid & tanh. It is pretty much quicker

④ It is obviously solving vanishing gradient ~~prob~~ and other problems

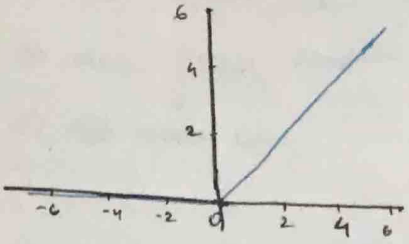
Disadvantage:

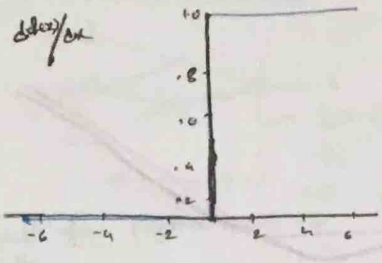
⑤ It is not Zero Centric. It is passing zero but it has not any neg value.

⑥ When input is negative the Relu is died, which means that when neg number is entered the output val is zero.

4. Leaky - ReLU Function:

$f(x) = \max(.01x, x)$



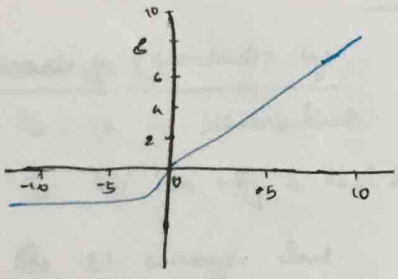


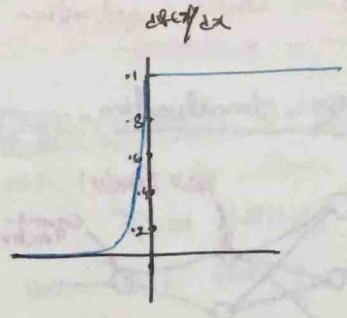
① Solves dead neuron problem (ReLU), because it has small neg number

② ~~It~~ has been not fully proved.

⑤ ELU (Exponential Linear Units) Function:

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





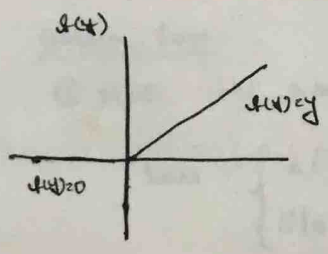
① No dead ReLU issue

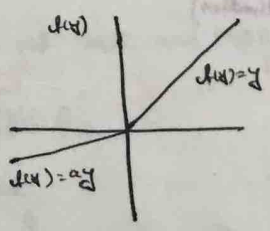
② The mean of the output is closed to 0, zero-centered

③ One small problem is that it is slightly more computationally intensive. Similar to Leaky ReLU. Here using exponential value.

⑥ Softmax

⑦ PRelu (Parametric Relu) Function:





ReLU vs PReLU. For PReLu the coefficient of the negative part is not constant and is adaptively learned.

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i < 0 \end{cases}$$

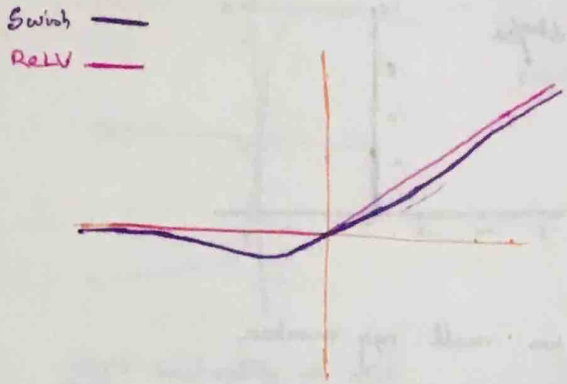
① If $a_i = 0$ ↓ becomes ReLU

② If $a_i > 0$, ↓ becomes Leaky ReLU

③ ~~If $a_i \neq 0$, becomes PReLU~~ If a_i is a learnable parameter, ↓ becomes PReLU

(h) **Swish :**

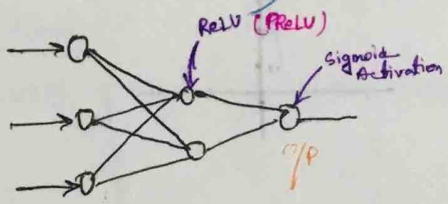
A self gated function. This is brought by google itself.



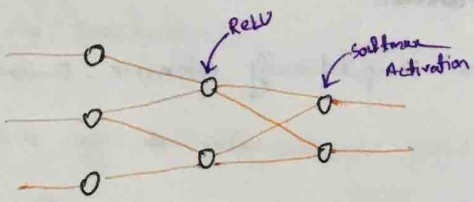
There is a problem in that we cannot find out the derivative of zero.

Technique — which activation function we should use :

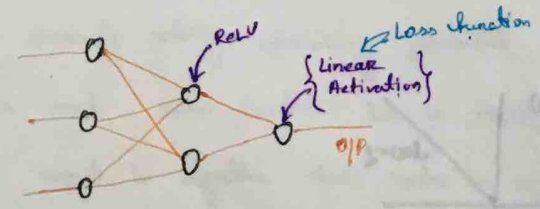
(i) **Binary classification** → convergence not happening so that change ReLU to PReLU (FLU)



(ii) **Multiclass classification :**



(iii) **Regression classification :** it means will be happening continuous value.



(4) **Loss Function :**

Deep Learning (ANN)

Loss Function

Regression

Data :

Exp	Degree	Salary (O/P)
10	PHD	—
—	—	—
—	—	—

continuous value

Regression Problem

Classification

Data :

Play	Study	Pass/Fail
10	2	Fail
4	3	Fail
5	5	maybe Pass

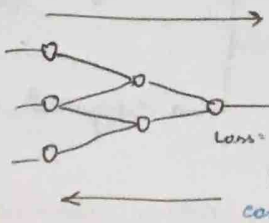
① Loss Func in Regression:

ⓐ MSE (Mean Square Error)

ⓑ MAE (Mean Absolute Error)

ⓒ ~~Huber~~ Huber Loss

Loss & cost function



Data = 100 records → 10 record (pass)

$$\text{cost} = \frac{1}{2n}\sum_{i=1}^{n}(y-\hat{y})^2$$

in cost func provide batch of data.

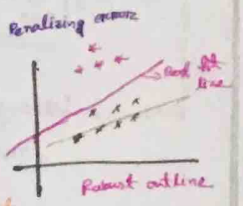
ⓐ MSE (Mean Square Error):

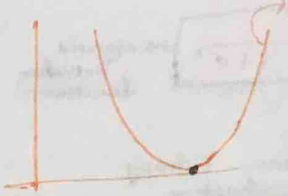
Loss function : $\frac{1}{2n}(y-\hat{y})^2$ ⇒ Quardratic Equation

cost function : $\frac{1}{2n}\sum_{i=1}^{n}(y-\hat{y})^2$

$(a-b)^2 = a^2 - 2ab + b^2$ [quardratic eqn]

⇒ $ax^2 + bx + c$





Advantages (quardratic eqn):

ⓘ it is differentiable

ⓘⓘ it has only 1 local or global minima.

ⓘⓘⓘ It converges fast

Disadvantages (quardratic equation):

ⓘ Not Robust to outlines (Line shifting)

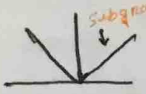
Here line shifting is huge.

ⓑ MAE (Mean Absolute Error):

Loss function = $\frac{1}{2}(y-\hat{y})$ cost function = $\frac{1}{2}\sum_{i=1}^{n}(y-\hat{y})$

Advantages:

ⓘ Robust to outline



Time consuming ↑↑

Disadvantages:

ⓘ Time consuming

ⓒ Huber Loss:

ⓘ MSE and ⓘⓘ MAE (combination of MSE and MAE)

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

outliers are not present

hyper parameter

when outliers is present

Loss Function in classification:

cross entropy → Binary cross entropy → Binary classification
cross entropy → Categorical cross entropy → Multiclass classification

ⓐ Binary cross Entropy:

(Binary classification problem)

$$Loss = -y \log(\hat{y}) - (1-y)\log(1-\hat{y})$$ ∴ logistic regression

$$Loss = \begin{cases} -\log(1-\hat{y}) & \text{if } y=0 \\ -\log \hat{y} & \text{if } y=1 \end{cases}$$ → Binary classification

$$\hat{y} = \frac{1}{1+e^{-x}}$$ → sigmoid Activation function

ⓑ Categorical cross Entropy:

(multiclass classification Problem)

f_1	f_2	f_3	opp
2	3	4	Good
5	6	7	Bad
8	9	10	Neutral

number of categories

	Good (y_{i_1})	Bad (y_{i_2})	Neutral (y_{i_3})
	1	0	0
	0	1	0
	0	0	1

i = row
j = column

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

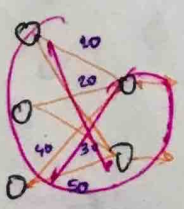
c = number of categories

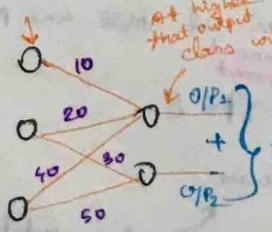
$$y_{i_1} = [y_{i_1}, y_{i_2}, y_{i_3} \dots\dots y_{ic}]$$

$$y_{ij} = \begin{cases} 1 & \text{if the element is in class} \\ 0 & \text{otherwise} \end{cases}$$

$\hat{y}_{ij}$ = Softmax Activation function ← O/P Layer

$$\sigma(z) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$ → softmax activation





At higher probability that output class will be coming up

$$= \frac{e^{10}}{e^{10+20+30+40+50}}$$

probability

0.4, 0.5, 0.6 =

$O/P_1 + O/P_2 = 1$ → at higher probability

Conclusion:

ReLU, Softmax → Multiclass → categorical cross entropy
ReLU, Sigmoid → Binary → Binary classification → Binary cross entropy

Linear Regression

ReLU, Linear Activation (Loss → MSE, MAE, Huber loss)