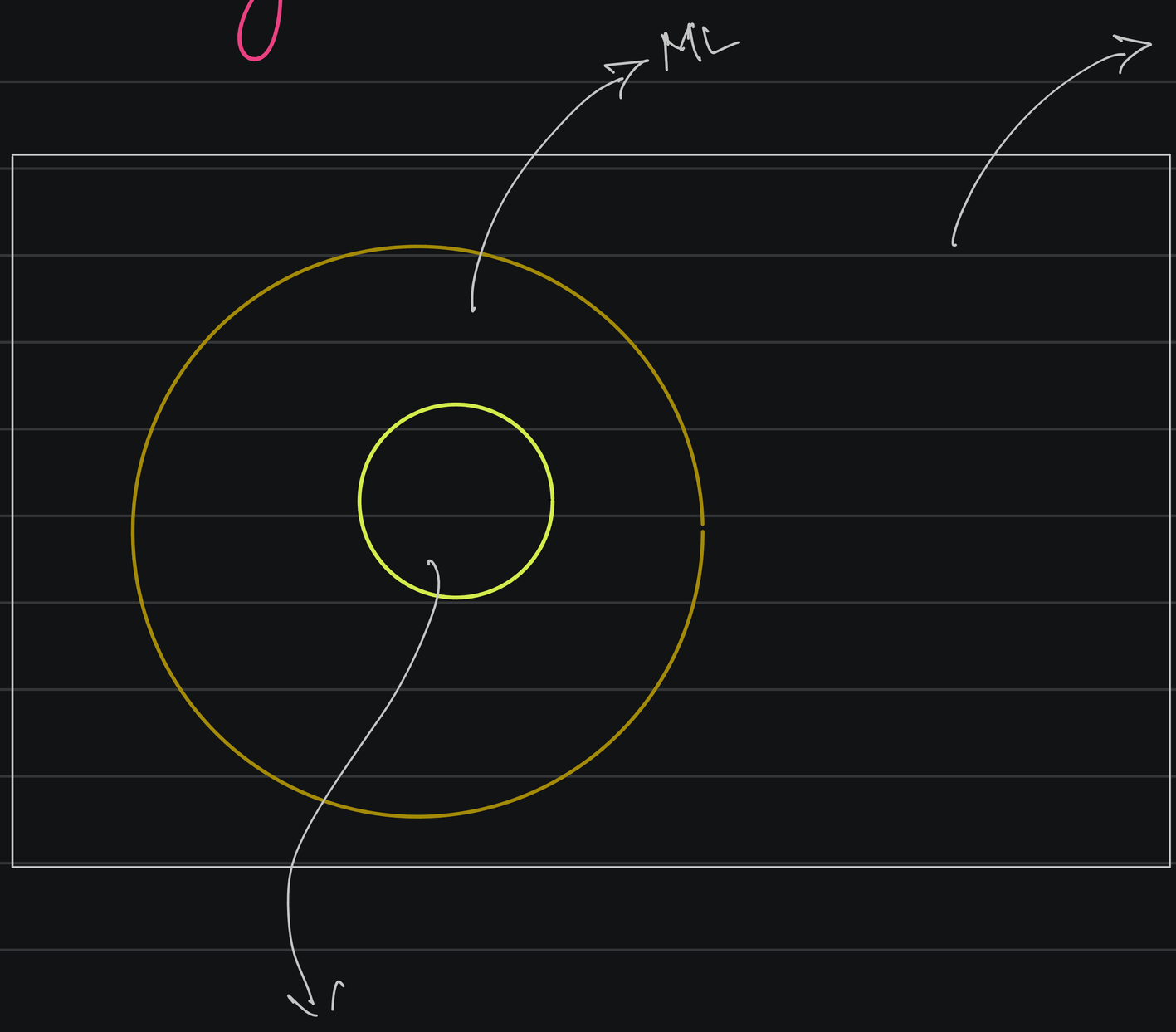
Deep Learning



DL ⟶ Mimicing the human brain.

Using multilayered Neural-Network.

Deep Learning

1. ANN ⟶ Artificial Neural Network.
   - ⟶ Classification.
   - ⟶ Regression.
2. CNN ⟶ Convolutional Neural Network.
   - i/p. ⟶ image, video features.
   - Computer vision.
   - eg. RCNN, Masked RCNN, Detectron, YOLO, YOLO v8

(3.) RNN ⟶ Recurrent Neural Network.

i/p. ↳ Text data, time series data.

eg. LSTM RNN, RNN GRU, Bidirectional LSTM RNN, Encoder Decoder, Transformers, BERT., Attention models

Libraries used : Tensor Flow, PyTorch.

Q Why DL is becoming very popular ?

(2005) ⟶ Facebook, Orkut, } Social Media Platform.

⇓

Data generated exponentially.

⇓

(2011 → BIG DATA ⟶ Storing this data efficiently.
-2016)

+1 HADOOP

↳ Unstructured data.

↳ Structured data.

Companies. ⟶ Cloudera, Hortonworks.

2015. ⟶ What to do with the data ?

↓

Can we use the data to

make the product better.

↓

Data Science.

↓

Deep Learning

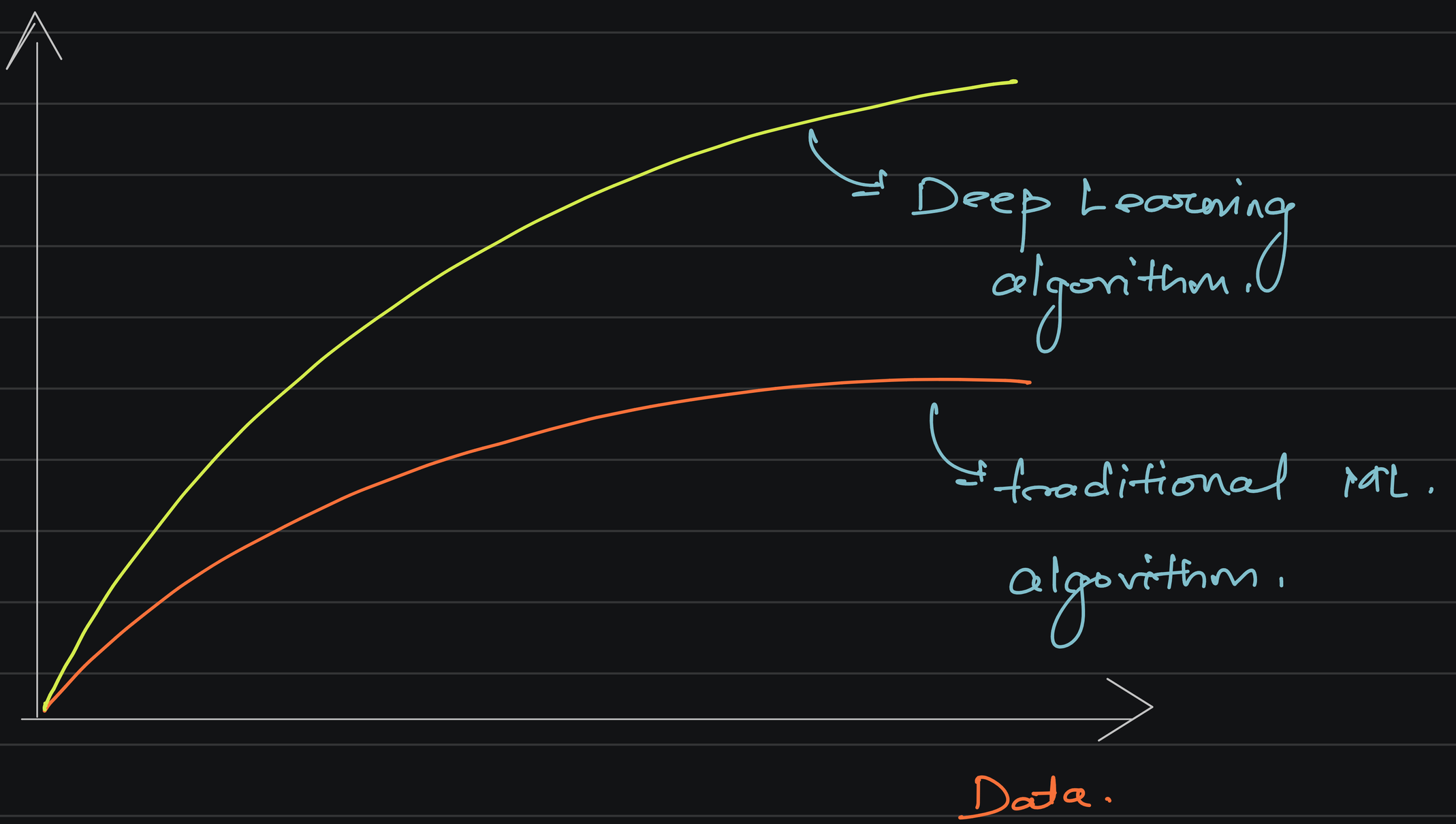
1. Hardware requirement ⇒ GPUs ⇒ Nvidia GPUs

* Price ↓↓↓.

* & Processing power ↑↑↑.

2. Huge amount of data is generated.

↳ Deep Learning model performs well.



(3.) Deep learning is being used in many domains.

(a.) Medical ⟶ Prediction of disease with X-ray scans., bone cracks, lung disease.

(b.) E Commerce

(c.) Retail.

(d.) Logistics.

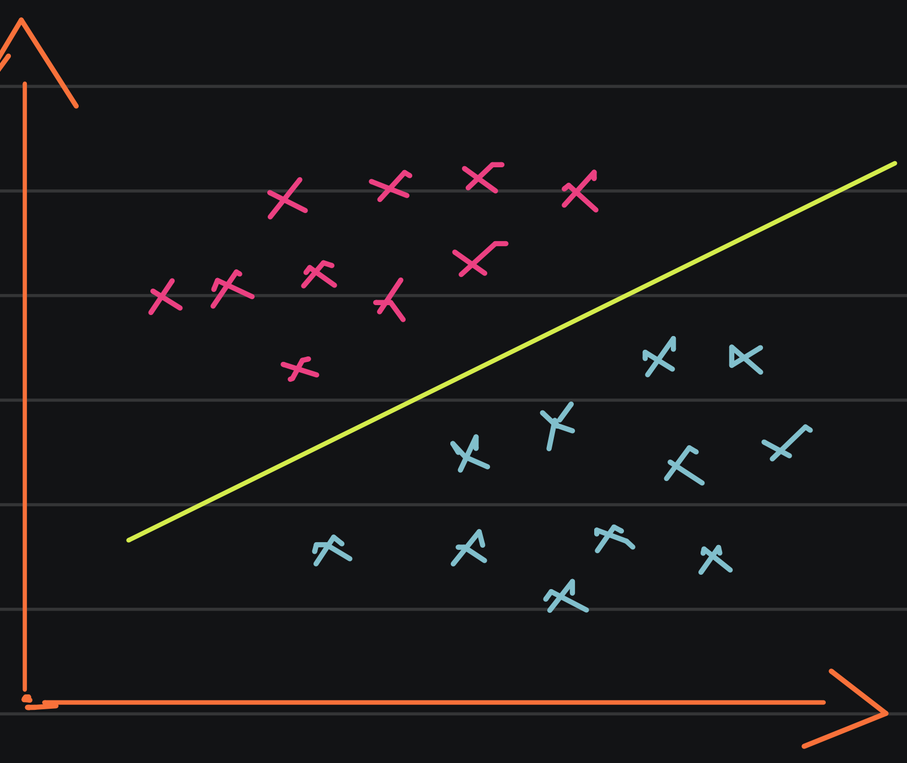
* Perceptron. [Artificial Neuron or Neural Network Unit]

Single layered neural network,

Layers.:

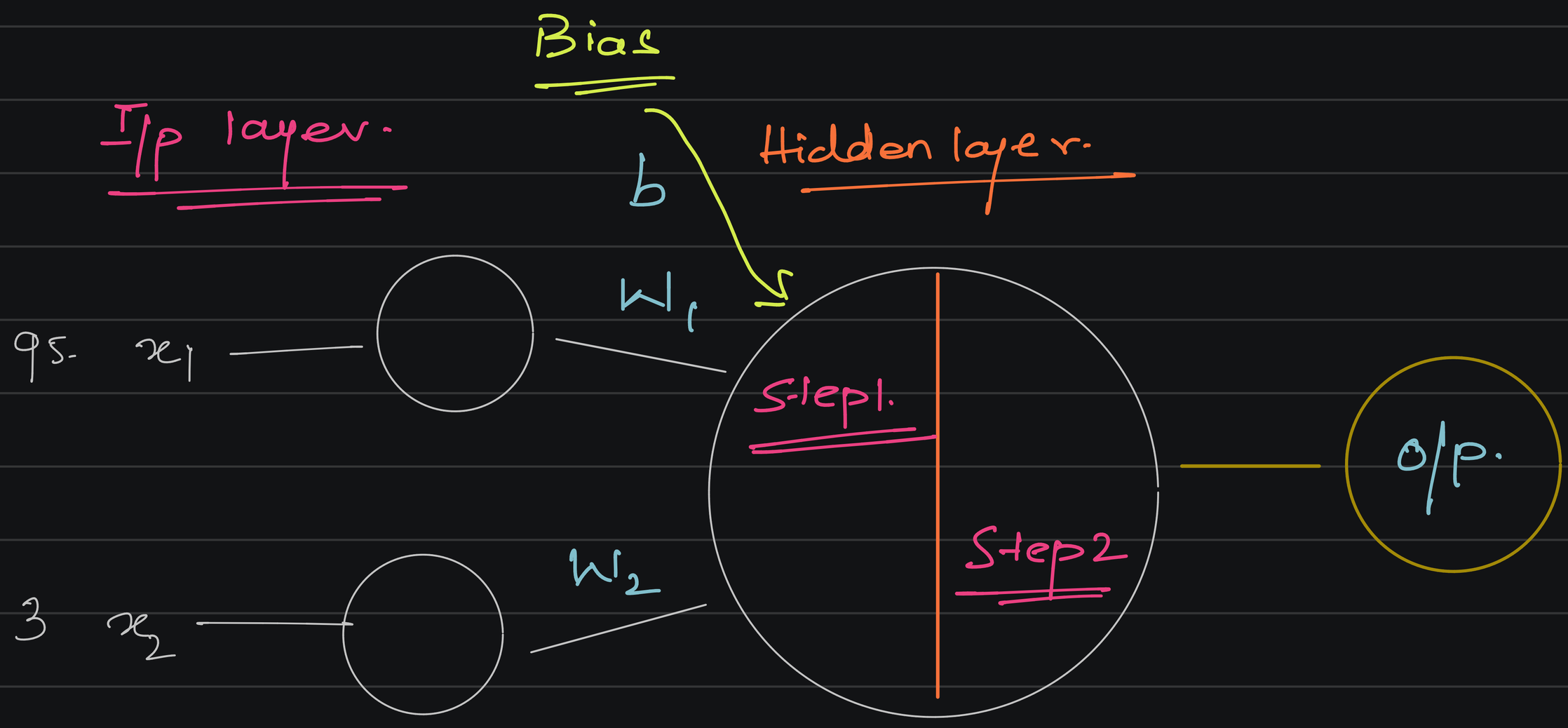
1. I/p layer.
2. Hidden layer.
3. Weights.
4. Activation -function.
5. Bias.
6. O/p layer.

Binary classification.
(linear. separable)



DATASET.

x_1 IQ	x_2 No. of study hrs.	y O/p Pass/Fail.
95.	3	0
110.	4	1.
100	5	1.



* Step 1.

$$Z = x_1 w_1 + x_2 w_2 + 1 \times b.$$

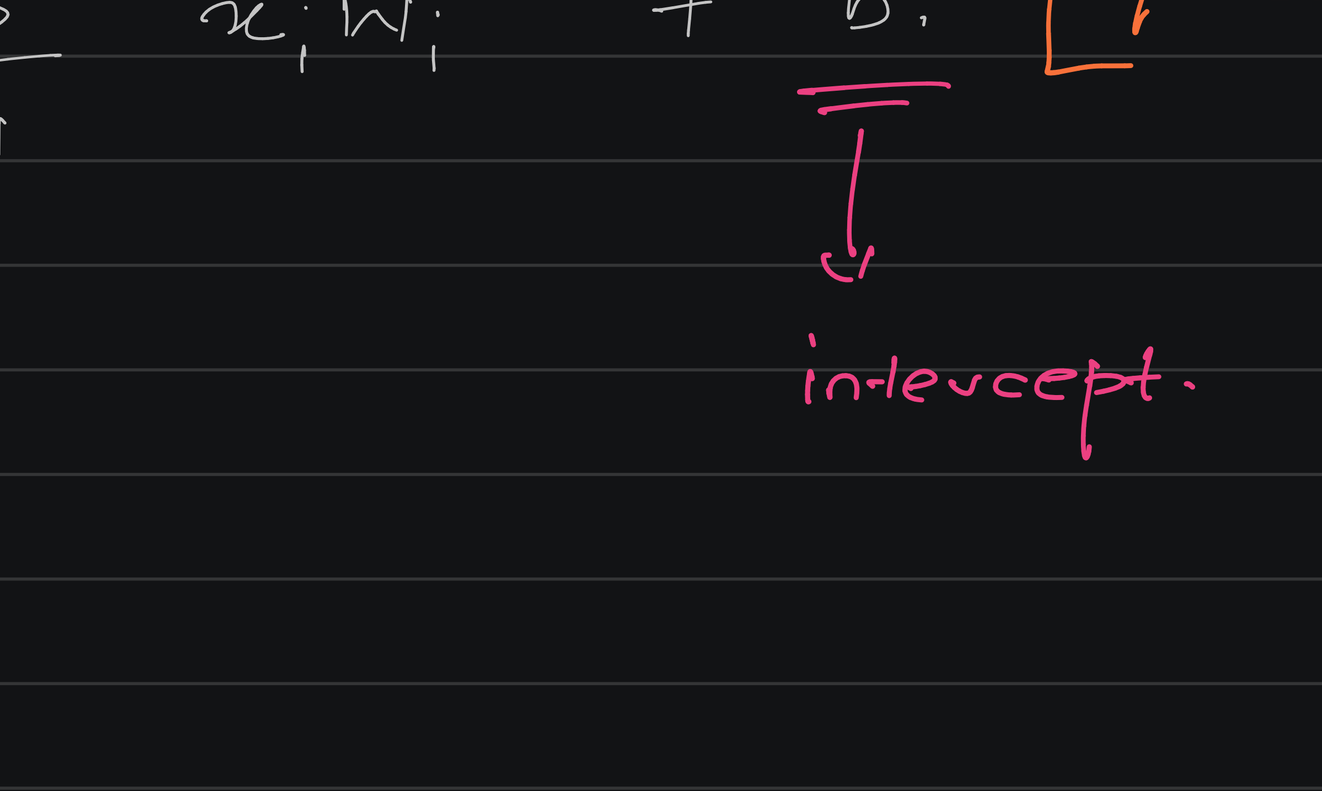
$$Z = \sum_{i=1}^{n} x_i w_i + b. \qquad [y = \sum mx + b]$$

b → intercept.

* Step 2

Apply an activation function on Z - value.

eg. Step function



decide a threshold. = 0 (Let)

if Z = 5, Step = 1.

as 5 > 0.

Z. —step-function→ -transforms into [0 or 1.]

(activation -function)

0 or 1. → binary classification.

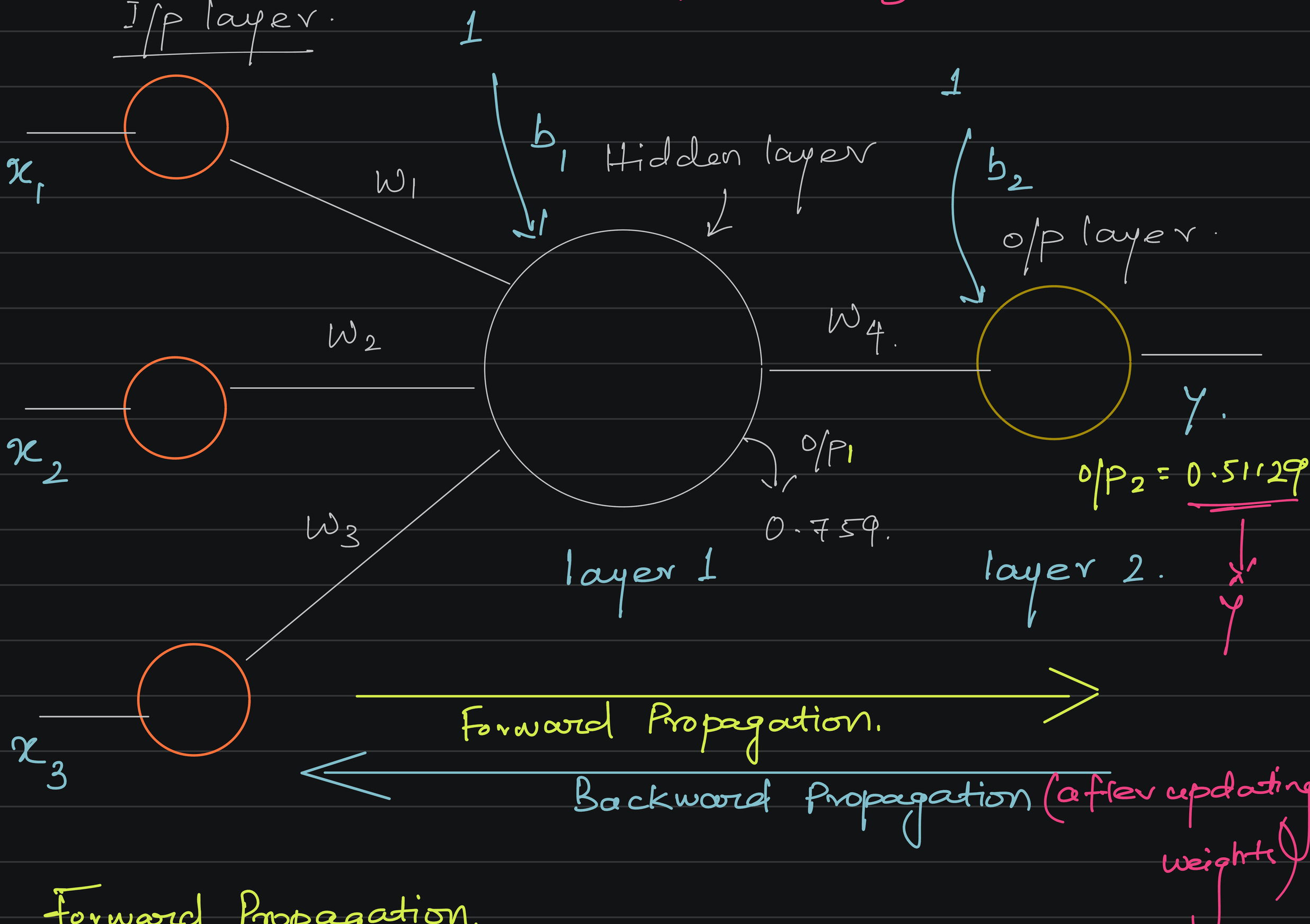
* Multi-layered Perceptron model. (ANN)

1. Forward Propagation.
2. Backward Propagation.
3. Loss -function.
4. Activation function.
5. Optimizers.

2- layered Neural N/W.

Dataset-

x_1 IQ	x_2 Study hrs.	x_3 Play hrs.	y O/P.	$\hat{y}$	Loss. $(y - \hat{y})$
95	4	4	1	0.51129.	0.49.
100	5	2	1		
95	2	7	0		



Forward Propagation.

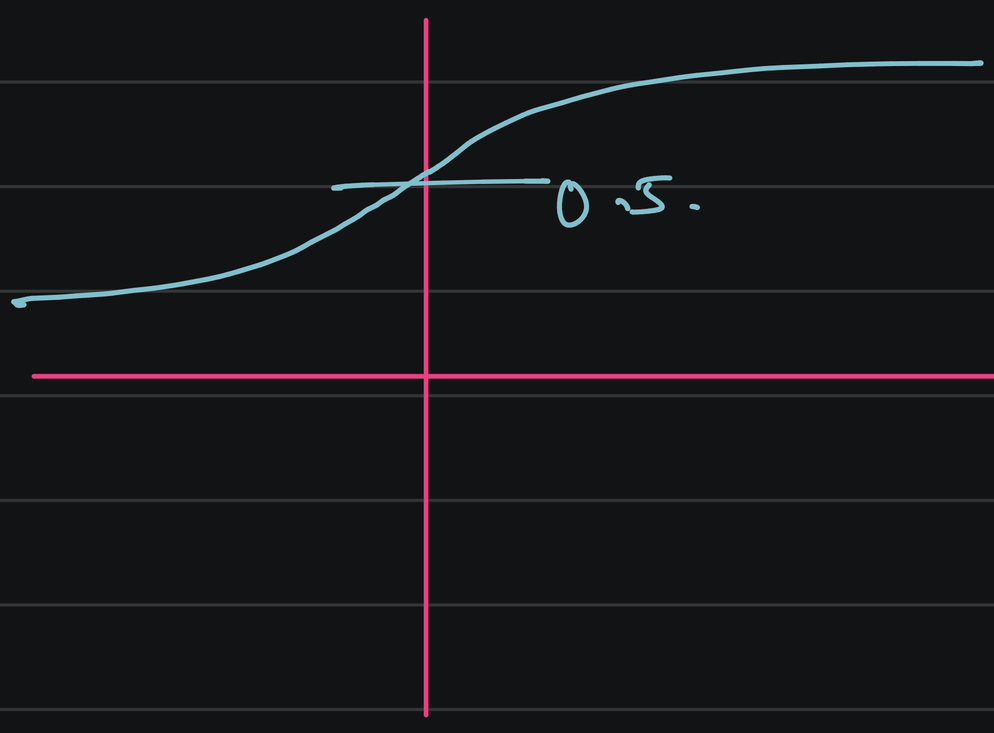
Hidden layer 1.

w_1	w_2	w_3	b_1
0.01	0.02	0.03.	0.01

Step 1 = $95 * 0.01 + 4 * 0.02 + 4 * 0.03 + 0.01$.

$Z = 1.151$.

Step 2 : Activation (Z)



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

$$O/P_1 = \frac{1}{1 + e^{-(1.151)}} = 0.759.$$

Hidden Layer 2 $\quad [w_4 = 0.02] \quad [b_2 = 0.03]$

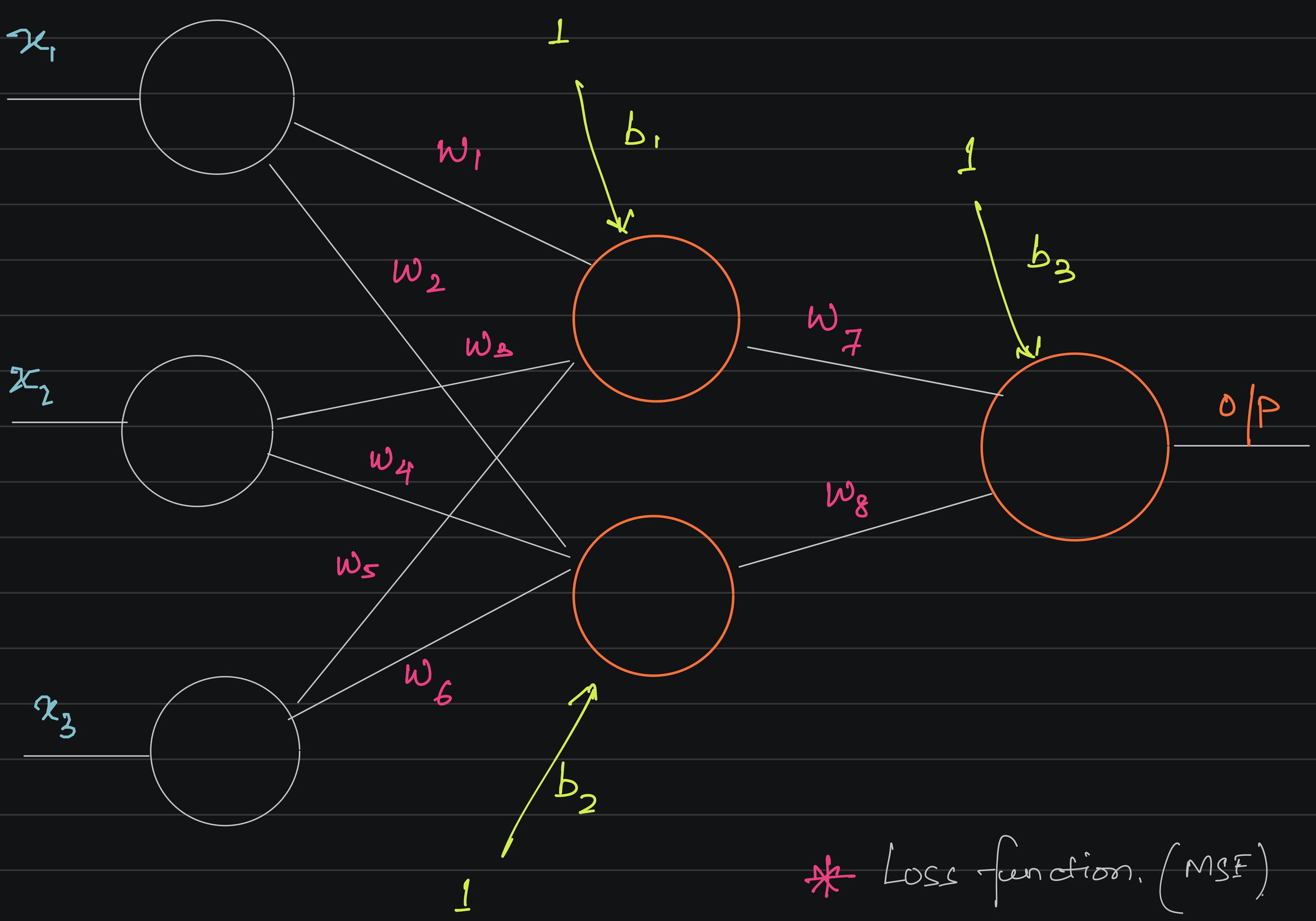
$$Z = 0.759 * 0.02 + 1 * (0.03)$$

$$= 0.04518$$

Step 2 : $O/P_2 = \frac{1}{1 + e^{-0.04518}}$

$$= 0.51129. \Rightarrow \text{o/p layer}$$

Backward Propagation & weight upgradation formula



* Loss function. (MSE)

$$(y - \hat{y})^2$$

* Weight upgradation formula. :

$$w_{7new} = w_{7old} - \eta \left[\frac{\partial h}{\partial w_{7old}} \right] \rightarrow \text{loss.}$$

Learning rate

$$\boxed{w_{new} = w_{old} - \eta \frac{\partial h}{\partial w_{old}}} \Rightarrow \text{Weight updation formula.}$$

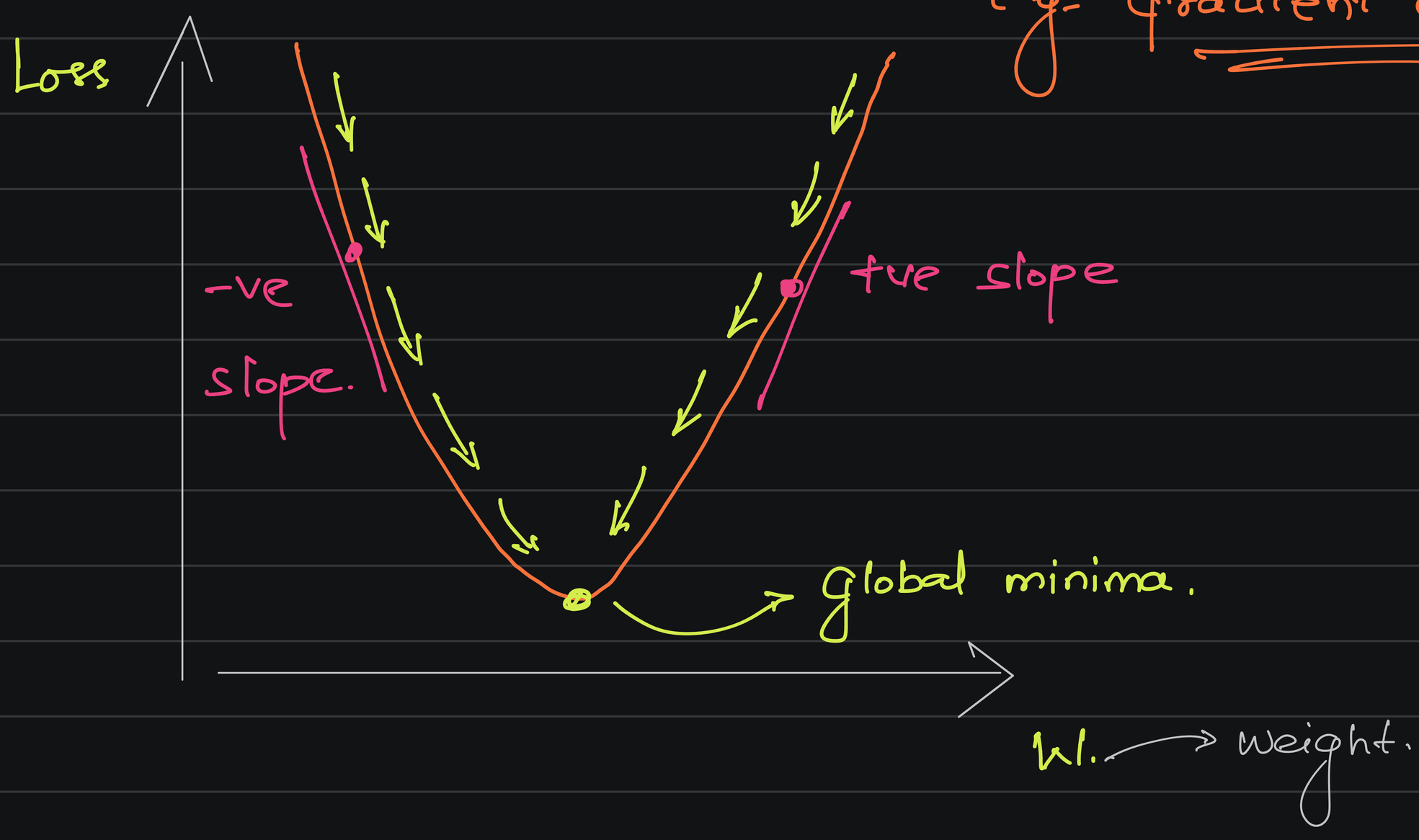
Similarly,

$$b_{new} = b_{old} - \eta \frac{\partial L}{\partial b_{old}}$$ ⇒ Bias updation formula.

* Weight updation is done using **Optimizers**.

↓

eg- Gradient descent.



* Optimizer. → Used to reduce the loss value.

When (-)ve slope, $w_{new} = w_{old} - \eta(-)ve$

$= w_{old} + (+)ve$

$w_{new} >> w_{old}$. ⇒ Moving towards global minima.

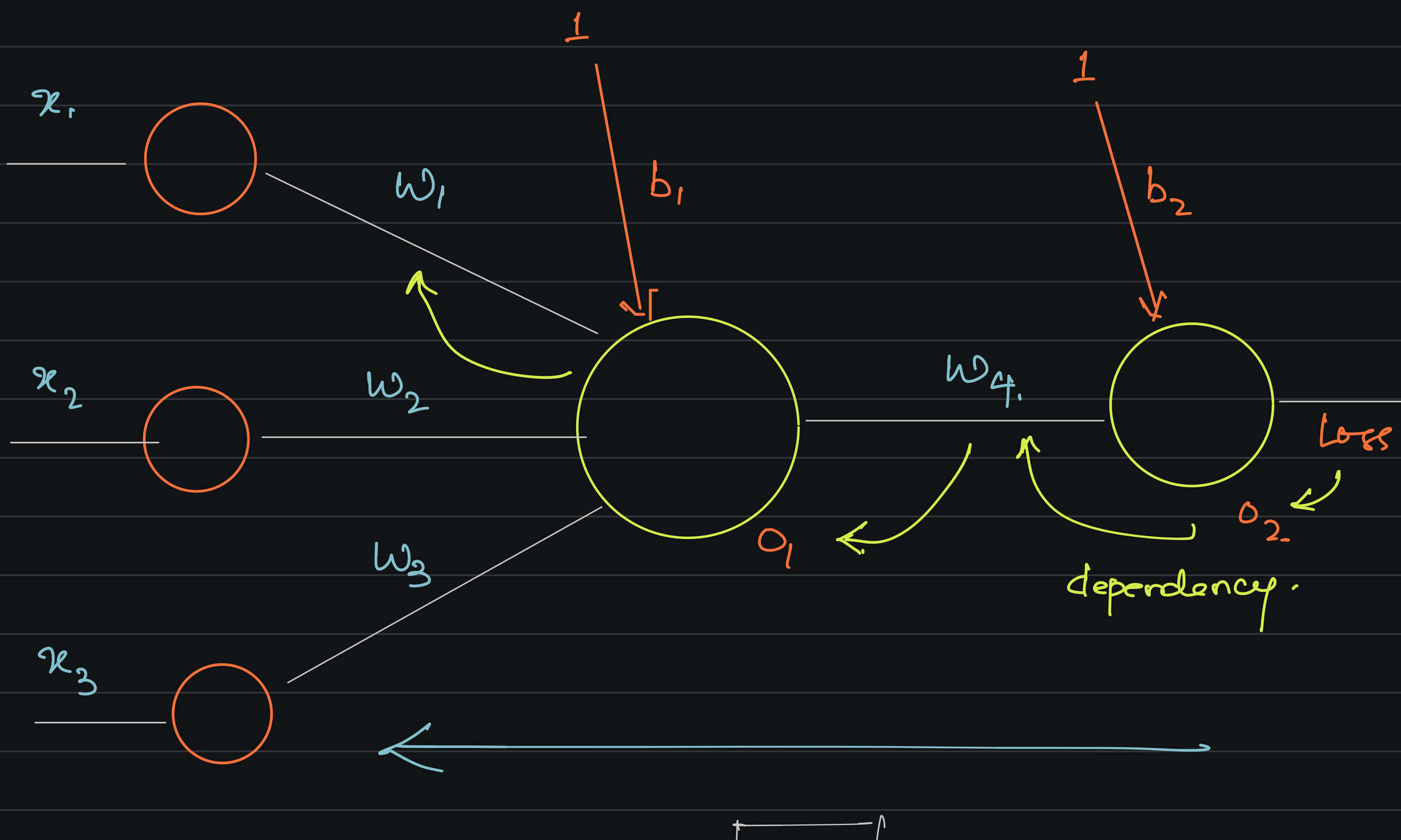
When (+)ve slope, $w_{new} = w_{old} - \eta(+)ve.$

$w_{new} << w_{old}.$

↓

moving toward = global minima.

✱ Chain Rule of Derivative



$$w_{4\,new} = w_{4old} - \eta\left[\frac{\partial L}{\partial w_{4old}}\right] \Rightarrow \text{Slope}$$

$$\therefore \frac{\partial L}{\partial w_{4old}} = \frac{\partial L}{\partial o_2} * \frac{\partial o_2}{\partial w_{4old}} \quad \{\text{Chain Rule}\} \text{ of derivative.}$$

$$w_{1new} = w_{1old} - \eta \left[\frac{\partial L}{\partial w_{1old}} \right]$$

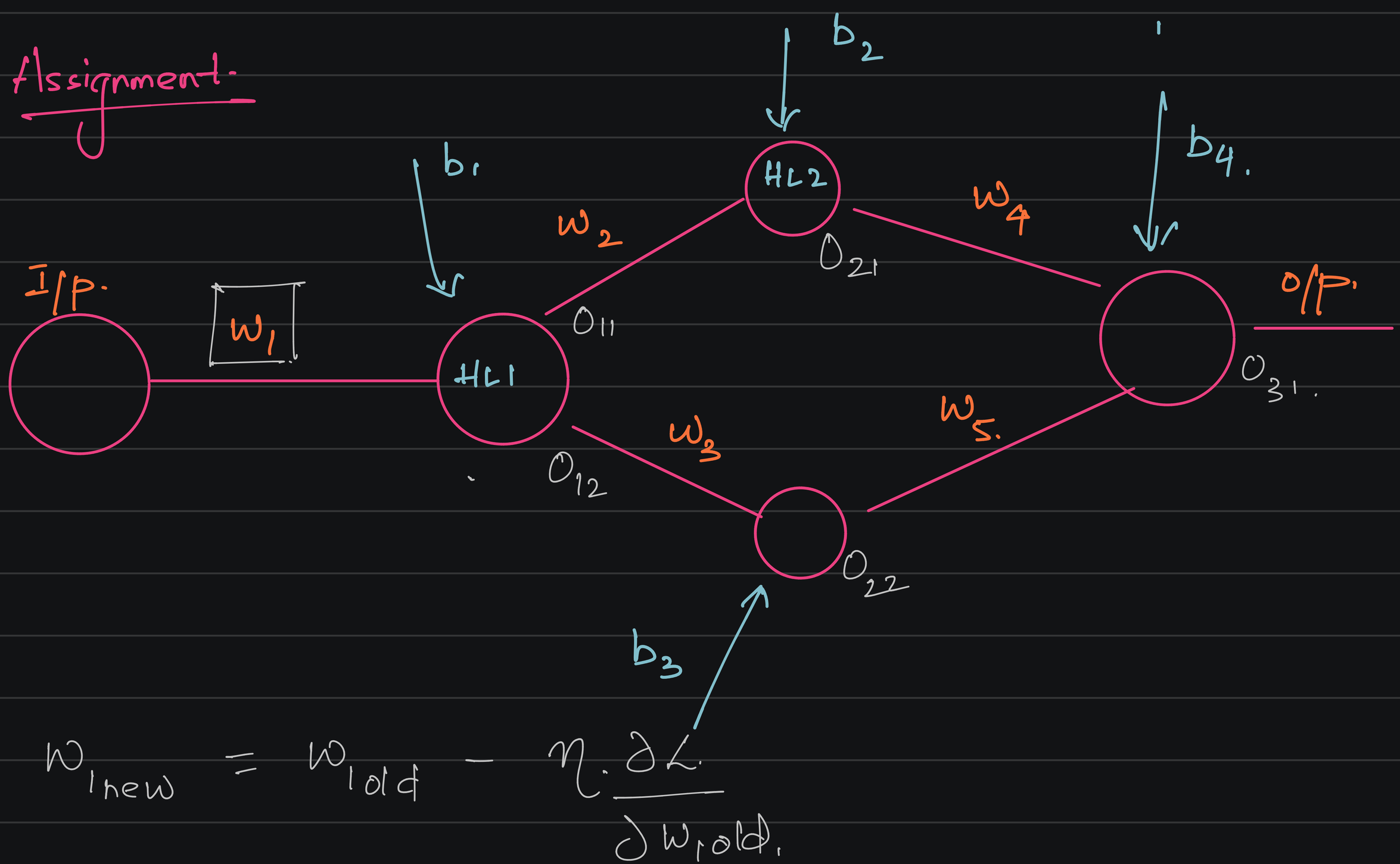
$$\frac{\partial L}{\partial w_{1old}} = \frac{\partial L}{\partial O_2} * \frac{\partial O_2}{\partial w_{4old}} * \frac{\partial w_{4old}}{\partial O_1} * \frac{\partial O_1}{\partial w_{1old}}$$

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

$w_1 \rightarrow O_1 \rightarrow w_4 \rightarrow O_2 \rightarrow L$

dependency.

Assignment:-



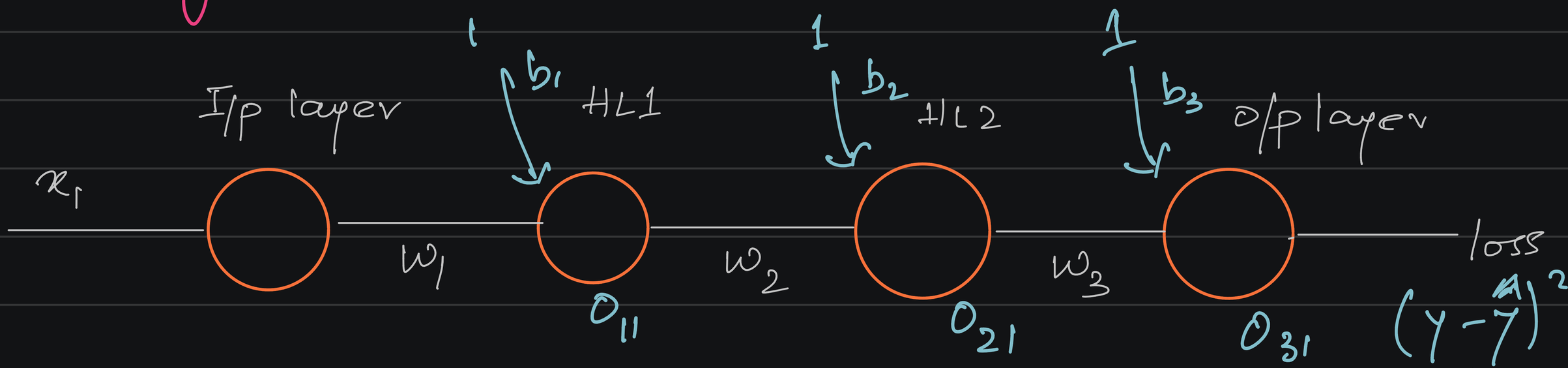
$$w_{1new} = w_{1old} - \eta \cdot \frac{\partial L}{\partial w_{1old}}$$

$$* \; w_{1new} = w_{1old} - \eta \left(\left[\frac{\partial L}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{21}} * \frac{\partial o_{21}}{\partial o_{11}} * \frac{\partial o_{11}}{\partial w_{1old}} \right] \right.$$

$$+$$

$$\left. \left[\frac{\partial L}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{22}} * \frac{\partial o_{22}}{\partial o_{12}} * \frac{\partial o_{12}}{\partial w_{1old}} \right] \right)$$

Vanishing Gradient Problem And Activation functions.



Deep layered Neural Network.

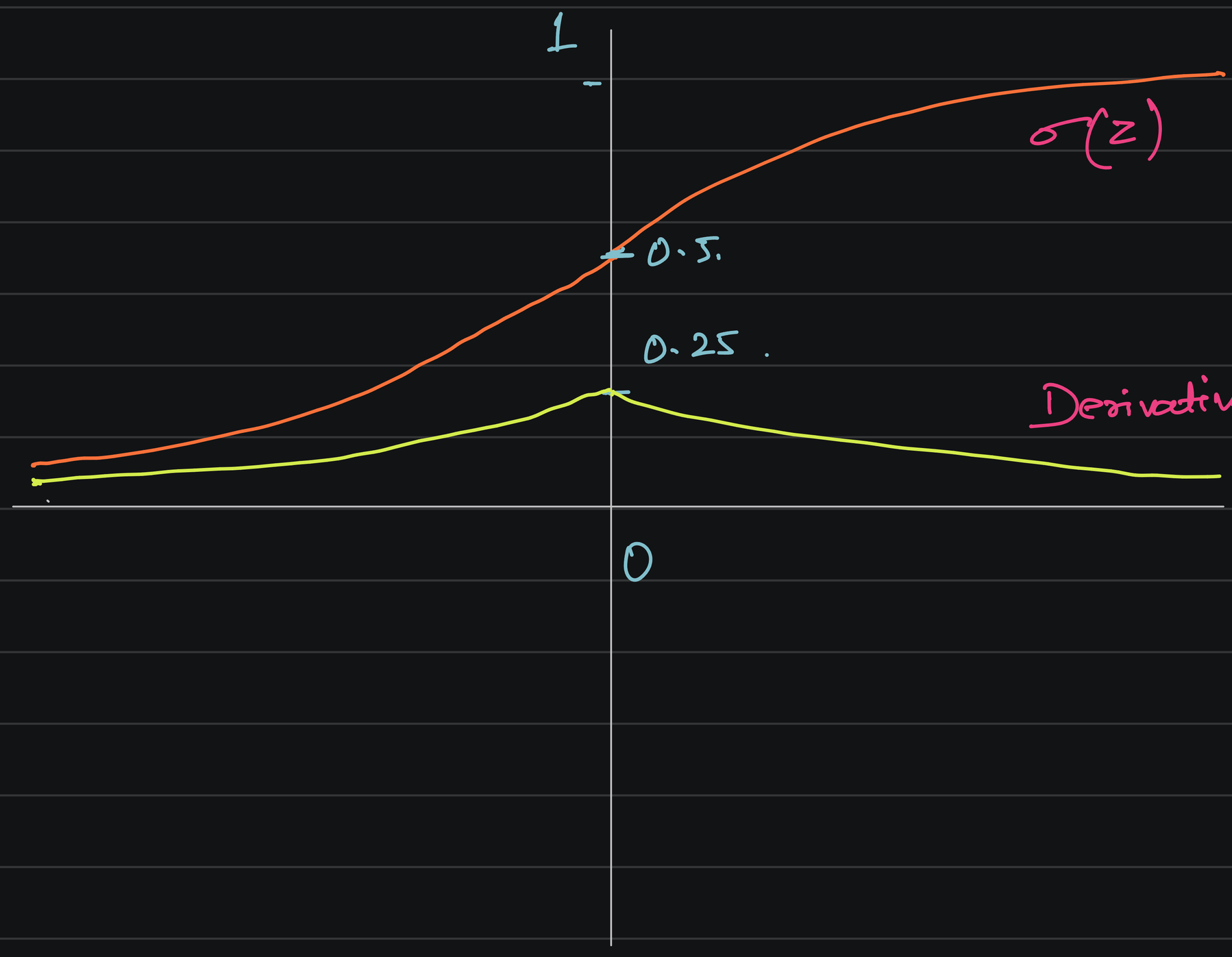
Sigmoid Activation.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \Rightarrow [0 \text{ to } 1]$$

$$\text{So, } [0 \leq \sigma(z) \leq 1]$$

Derivative of $\sigma(z)$: $0 \leq \sigma(z) \leq 0.25$.

* In backward propagation, we take derivatives.



If we want to update weights,

$$w_{1new} = w_{1old} - \eta \frac{dL}{dw_{1old}}$$

$$\frac{dL}{dw_{1old}} = \frac{dL}{do_{31}} \times \frac{do_{31}}{do_{21}} \times \frac{do_{21}}{do_{11}} \times \frac{do_{11}}{dw_{1old}}$$

$$o_{31} = \sigma(w_3 \times o_{21} + b_3) \quad z.$$

$$\text{or} \quad o_{31} = \sigma(z)$$

Now, $\frac{\partial O_{31}}{\partial O_{21}} = \frac{\partial(\sigma(z))}{\partial(z)} \times \frac{\partial(z)}{\partial O_{21}}$

1 ——— 1

→ chain rule of derivatives was applied.

$= [0 \text{ to } 0.25] \times w_3$.

⇓

Small value.

$\therefore \frac{\partial(w_3 \times O_{21} + b_3)}{\partial O_{21}} = w_3$.

($w_3 \times O_{21} + b_3$ → z.)

So,

$\frac{\partial L}{\partial w_{1old}} = \frac{\partial L}{\partial O_{31}} \times \frac{\partial O_{31}}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial w_{1old}}$

Small value | Small value | Small value | Small value.

Result : Bigger the chain, smaller is the derivative of loss. value.

Then.

$w_{1new} = w_{1old} - \left[\eta \frac{\partial L}{\partial w_{1old}}\right]$ → small.

So, $w_{new} \approx w_{old}$.

↓

Resulting in gradient descent to not work properly.

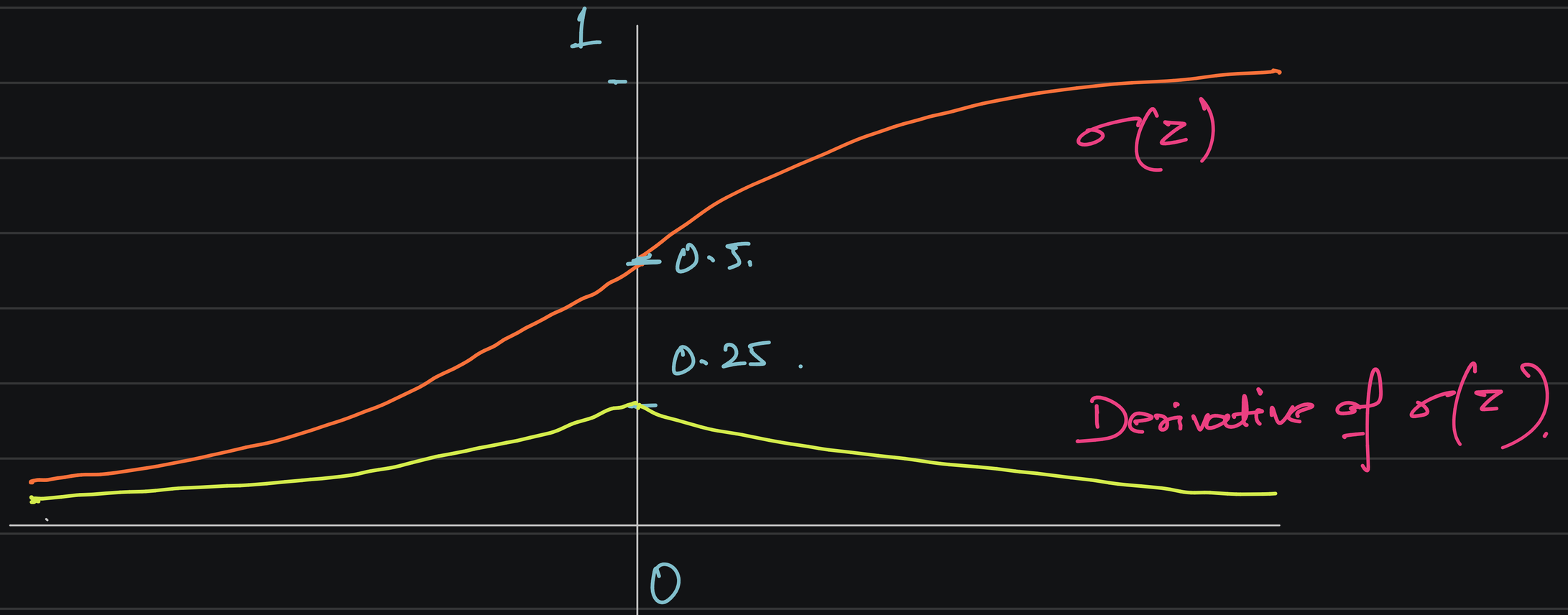
So, in deep layeved neural network., sigmoid activation function will not work as its derivative ranges between 0 - 0.25.

To fix this, use other activation functions :

1. Tanh.
2. Relu.
3. Prelu.
4. Elu.
5. Swiss.

(*) Activation functions :

1. Sigmoid activation function.



Advantages.

1. Clear prediction. (1 or 0)

Disadvantages.

1. Prone to vanishing gradient problem.
2. Function output is not zero centered.

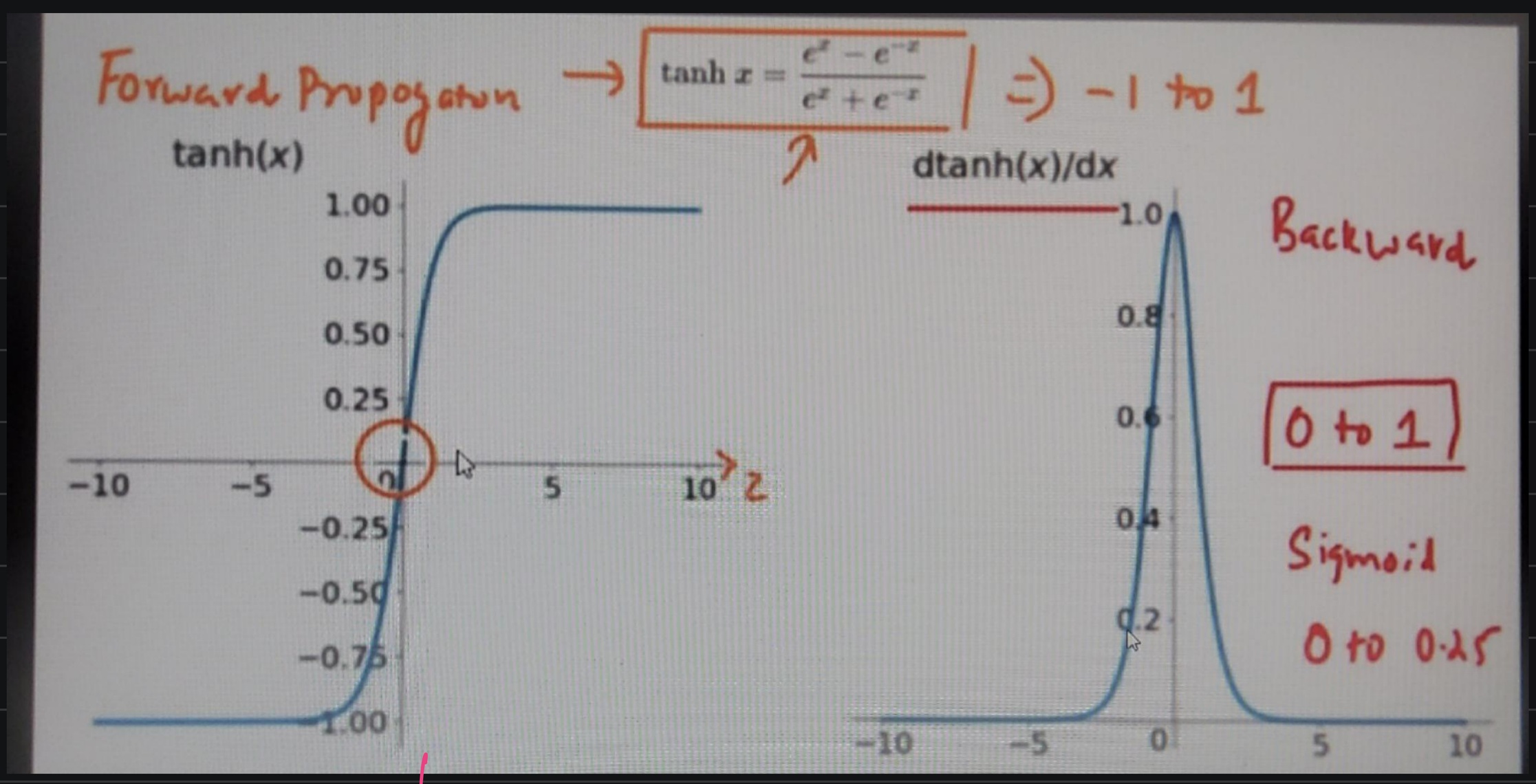
↓

Efficient weight updation.

Zero centered. ⟶ Smoothens the weight updation.

2. Tanh Activation function.

Forward Propogation → $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ ⇒ -1 to 1



Zero centered.

Advantages:

(1.) Zero centered → Efficient weight updation.

Disadvantage:

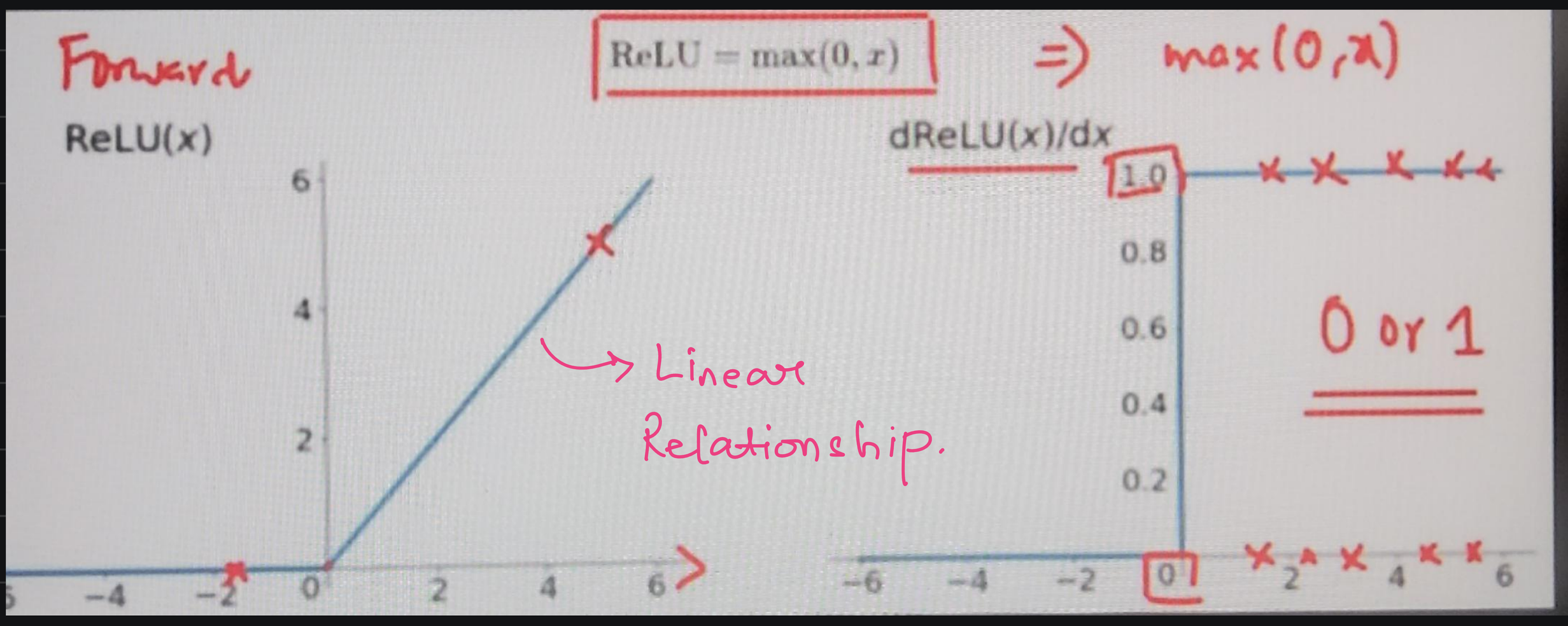
(1.) Time complexity is move.

(2.) Possibility of vanishing gradient problem for a very deep neural network.

(3.) Relu Activation Function.

Forward

$\mathbf{ReLU} = \max(0, x)$ ⇒ max(0,x)



→ Linear Relationship.

0 or 1

Advantages.

1. Solves vanishing gradient problem.
2. Time complexity — Fast.

Disadvantage.

1. If the derivative is 0, the neuron becomes a dead neuron.
2. Not zero centric.

4. Leaky Relu or Parametric Relu. Activation Function.

→ Leaky Relu.

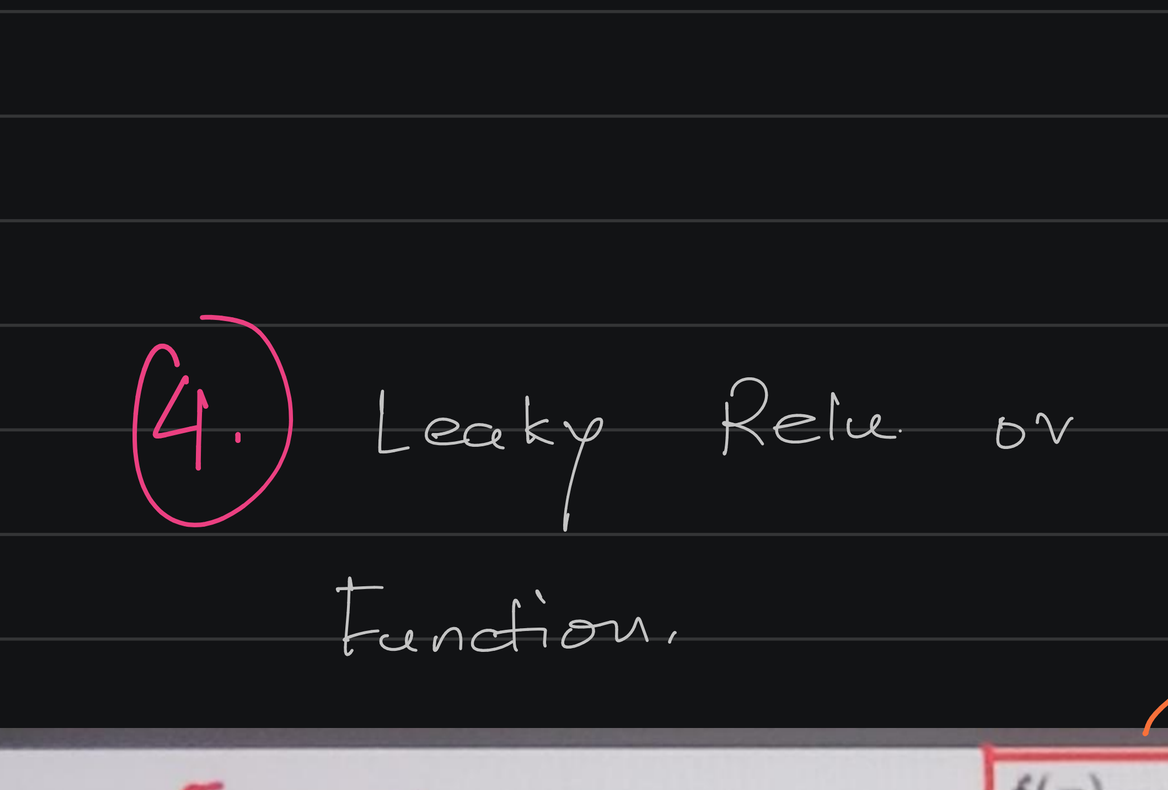
Forward

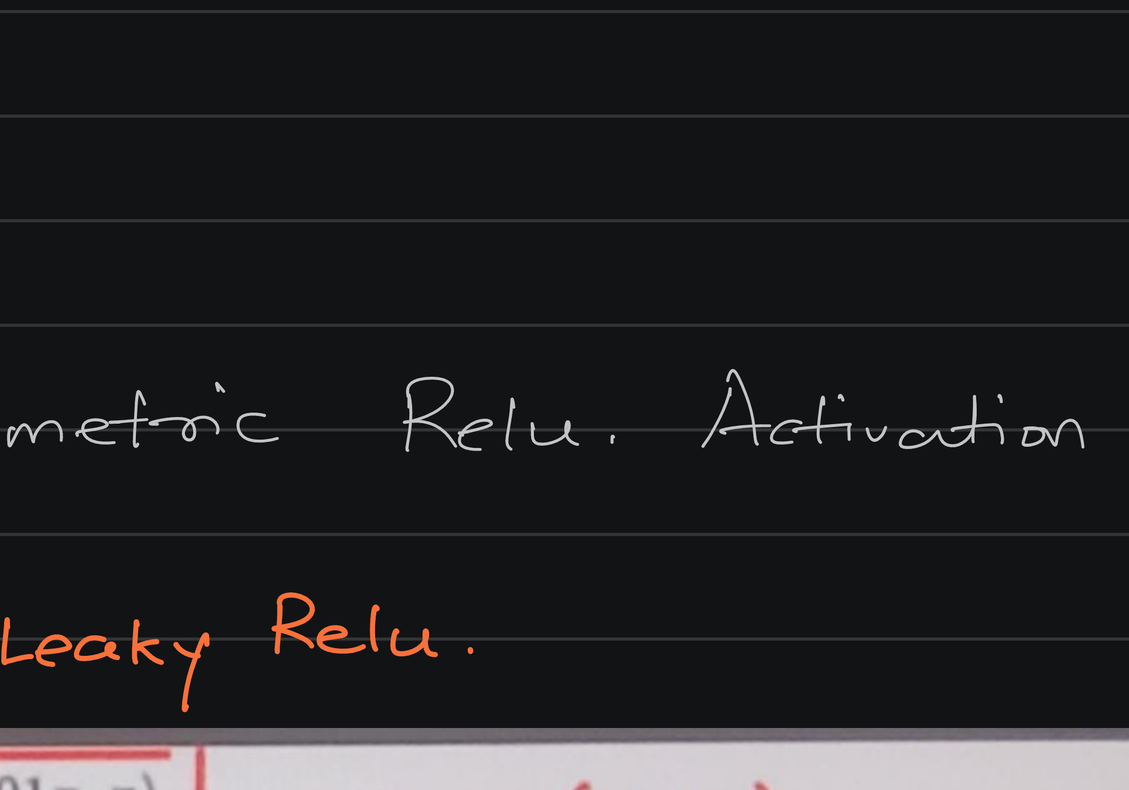
f(x)

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

max(0, x)

df(x)/dx





Back propogation

Parametric Relu.

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

α → hyperparameter.

Advantage.

1. Prevents the dead neuron, due to the small value of $f(x)$. Derivative never becomes 0.

Disadvantage.

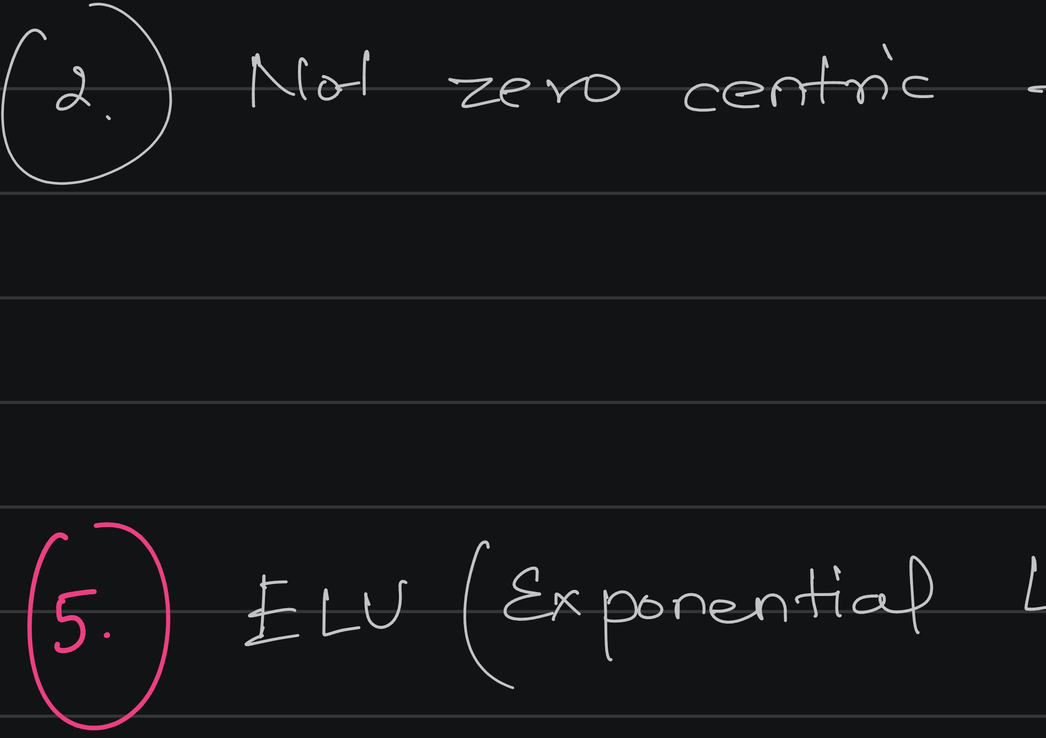
2. Not zero centric — Weight updation not efficient.

5. ELU (Exponential Linear Unit)

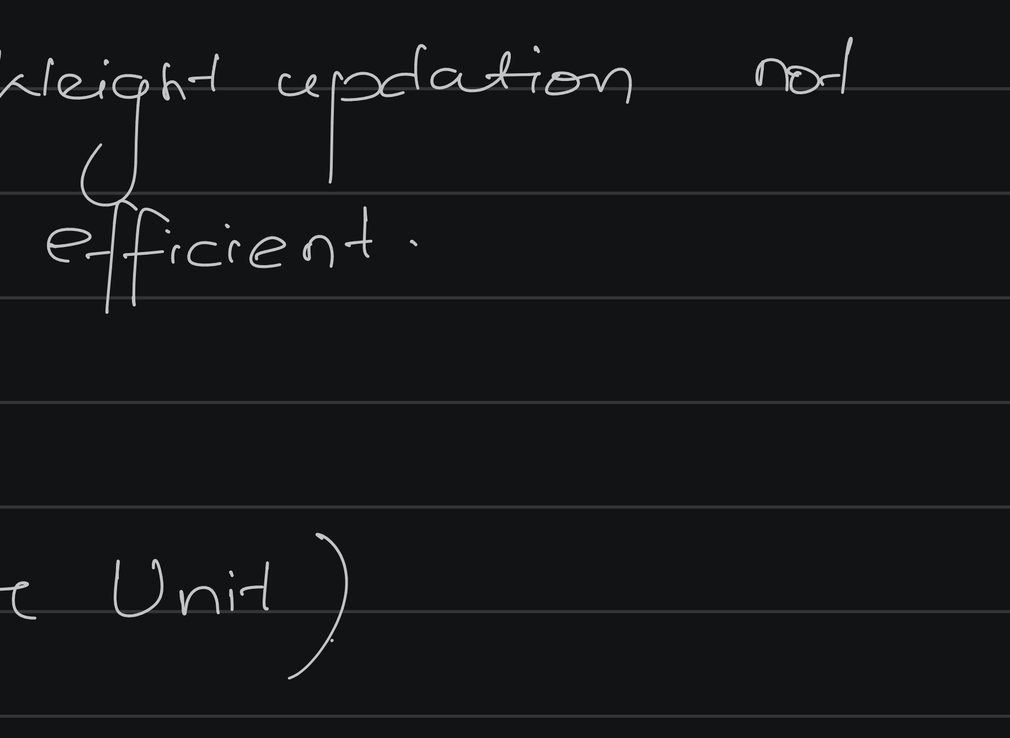
Forward propogation

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

hyperparameter



Backward propagation



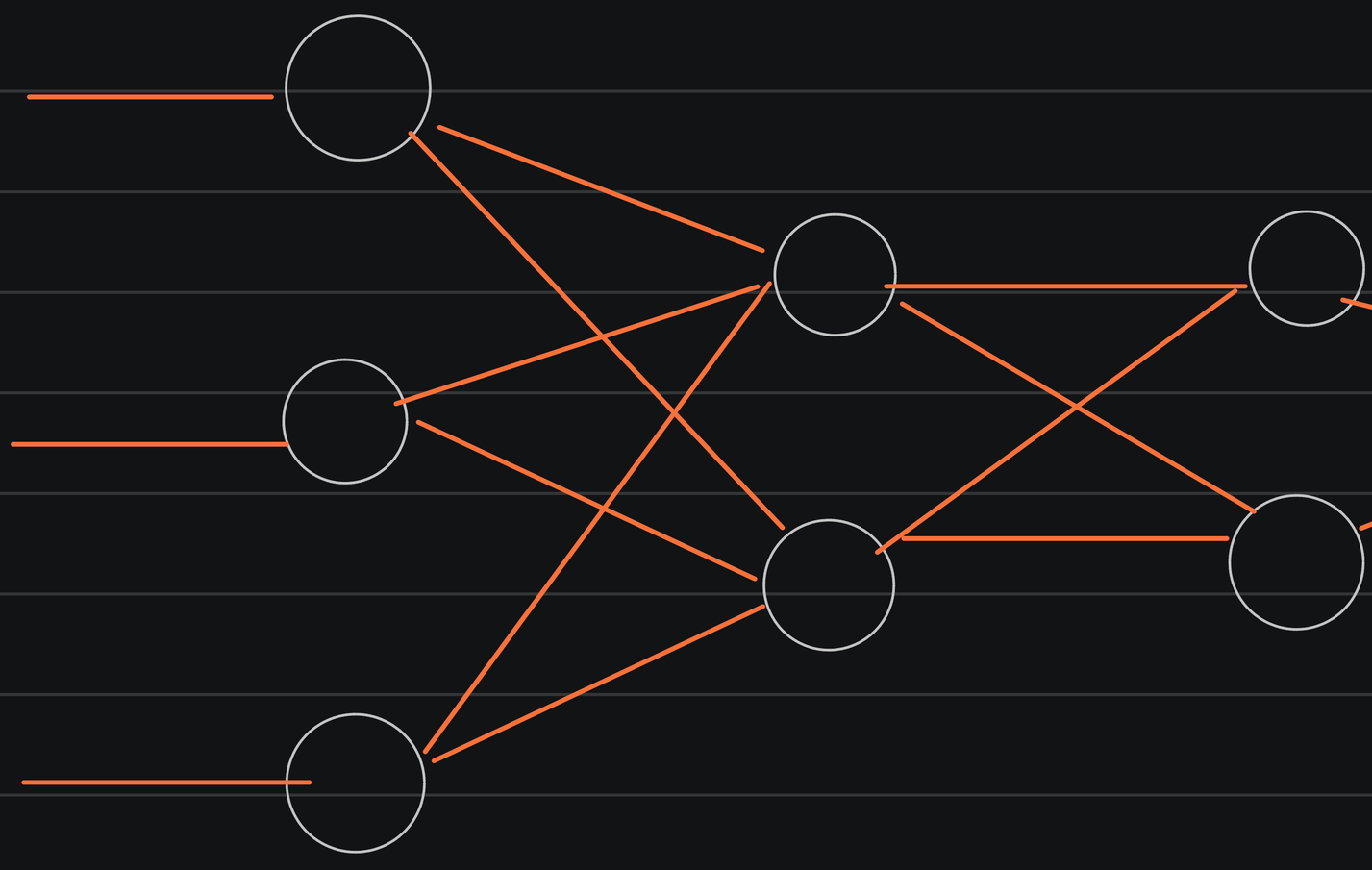
Advantage.

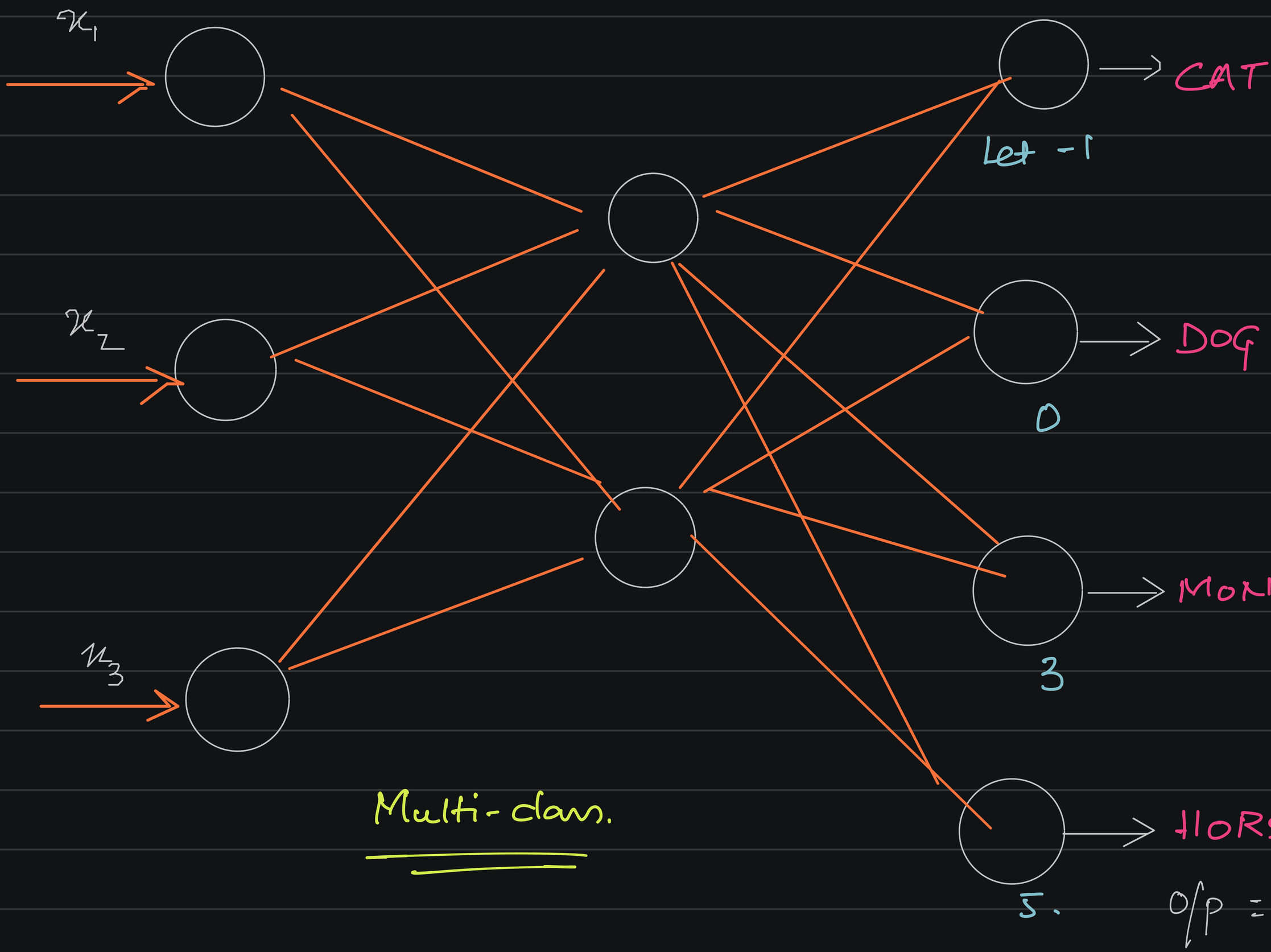
1. Zero centric.
2. Prevents dead neurons.
3. Linear relationship.

Disadvantage

1. Time complexity — high. (because of formula).

6. Softmax Activation Function. (Used for multi-class Classification).





Multi-class.

$$\text{Softmax} = \frac{e^{y_i}}{\sum_{k=0}^{n} e^{y_k}}$$

So, $$\text{Cat} = \frac{e^{-1}}{\left(e^{-1} + e^{0} + e^{3} + e^{5}\right)} = 0.00033$$

$$\text{Dog} = \frac{e^{0}}{\left(e^{-1}+e^{0}+e^{3}+e^{5}\right)} = 0.0024.$$

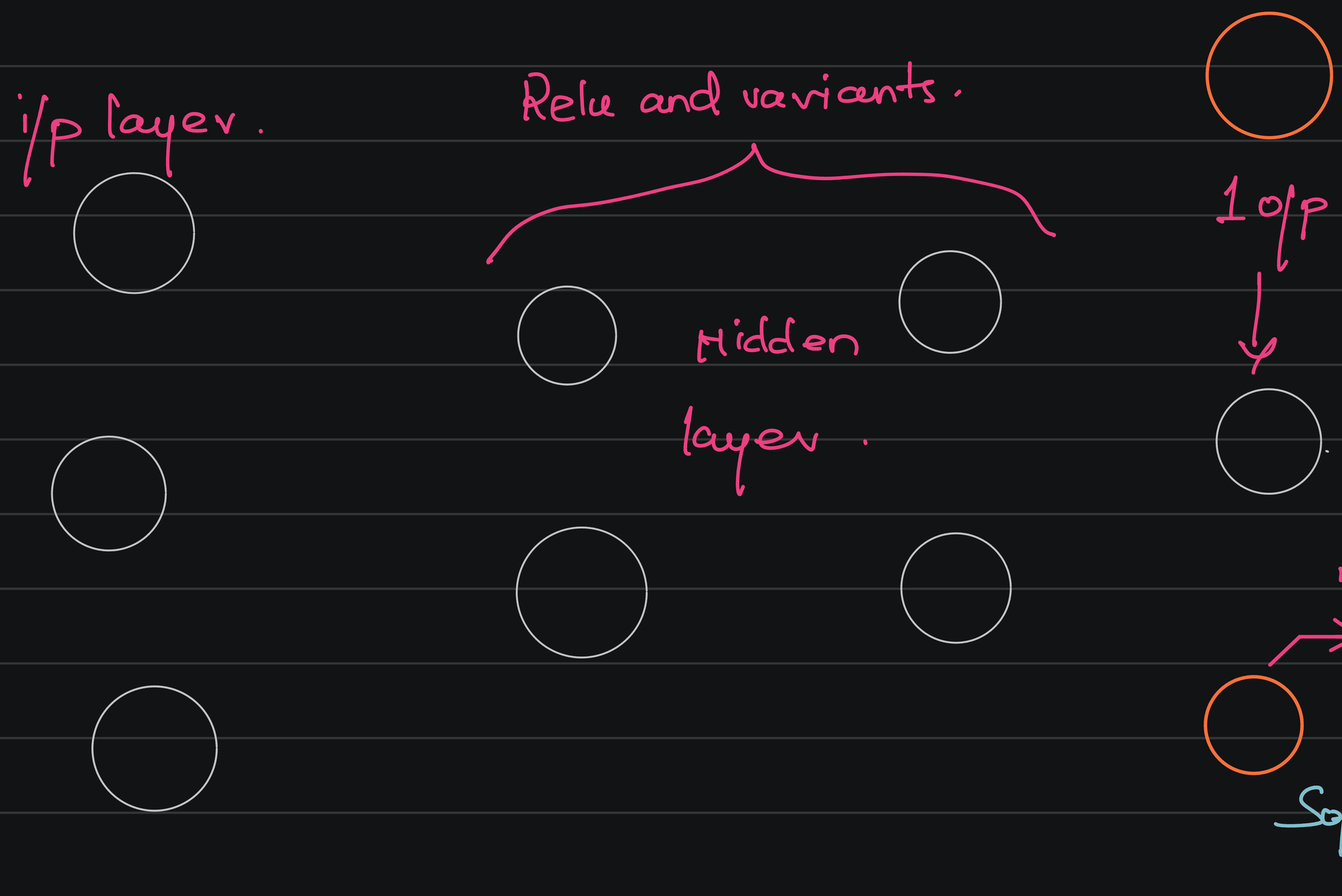
$$\text{Monkey} = \frac{e^{3}}{\left(e^{-1}+e^{0}+e^{3}+e^{5}\right)} = 0.0183.$$

$$\text{Horse} = \frac{e^{5}}{\left(e^{-1}+e^{0}+e^{3}+e^{5}\right)} = 0.1353.$$

$$Pr(\text{Horse}) = \frac{0.1353}{0.00033 + 0.0024 + 0.0183 + 0.1353}$$

$$= 0.86 \approx 86\%$$

Which activation function to use when.



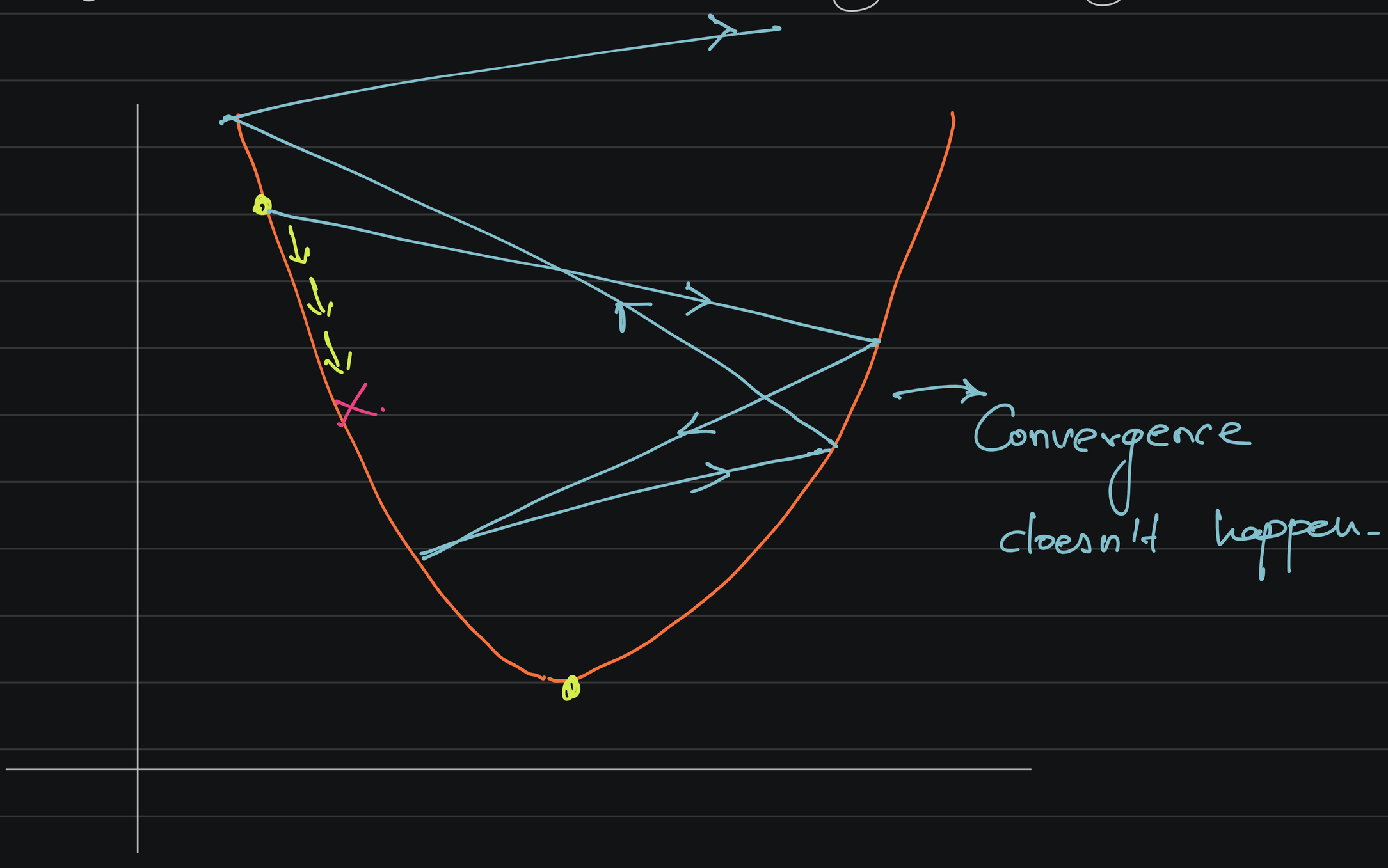
So, Binary classification in o/p → Sigmoid.

Multiclass classification in o/p → Softmax.

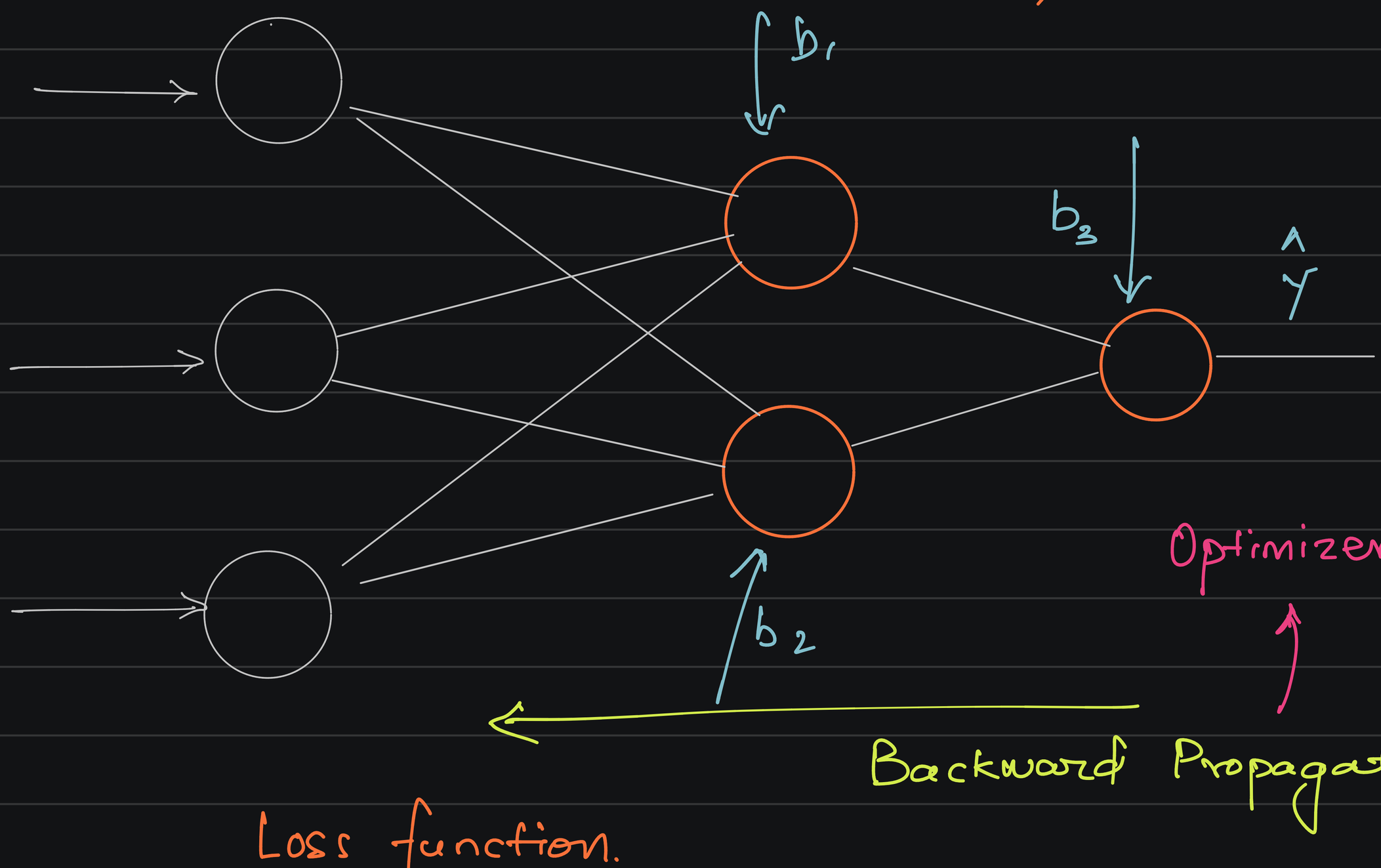
* Regression problem → Linear activation function.

* Exploding Gradient Problem. → When does this happen?

eg Weight initialization → Higher weights.



(1.) Loss Function & Cost function.

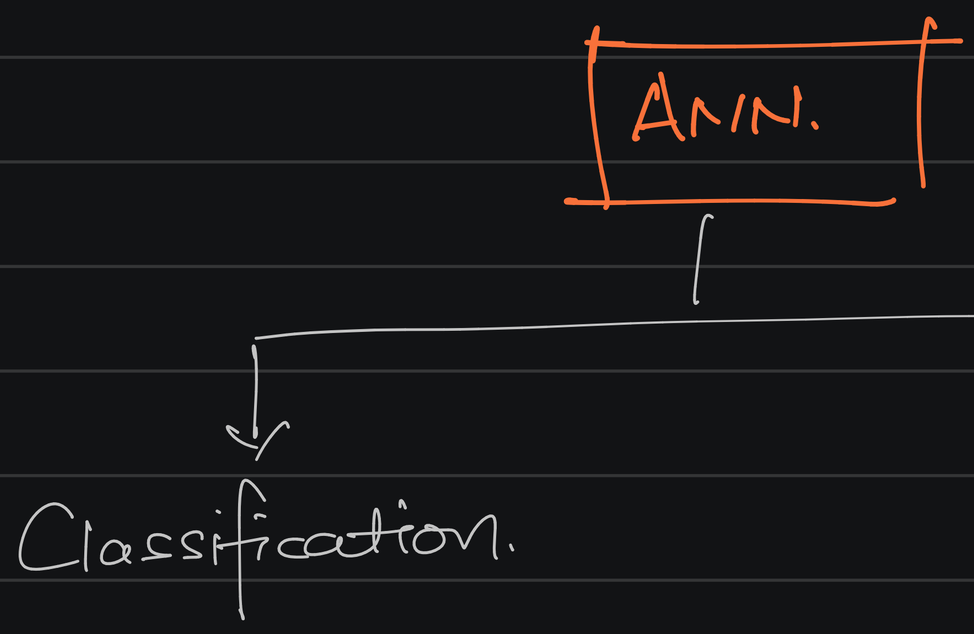


Loss function.

$$= (y - \hat{y})^2$$ → for single datapoint.

Cost function.

$$= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$ → for batch datapoints. (dataset).



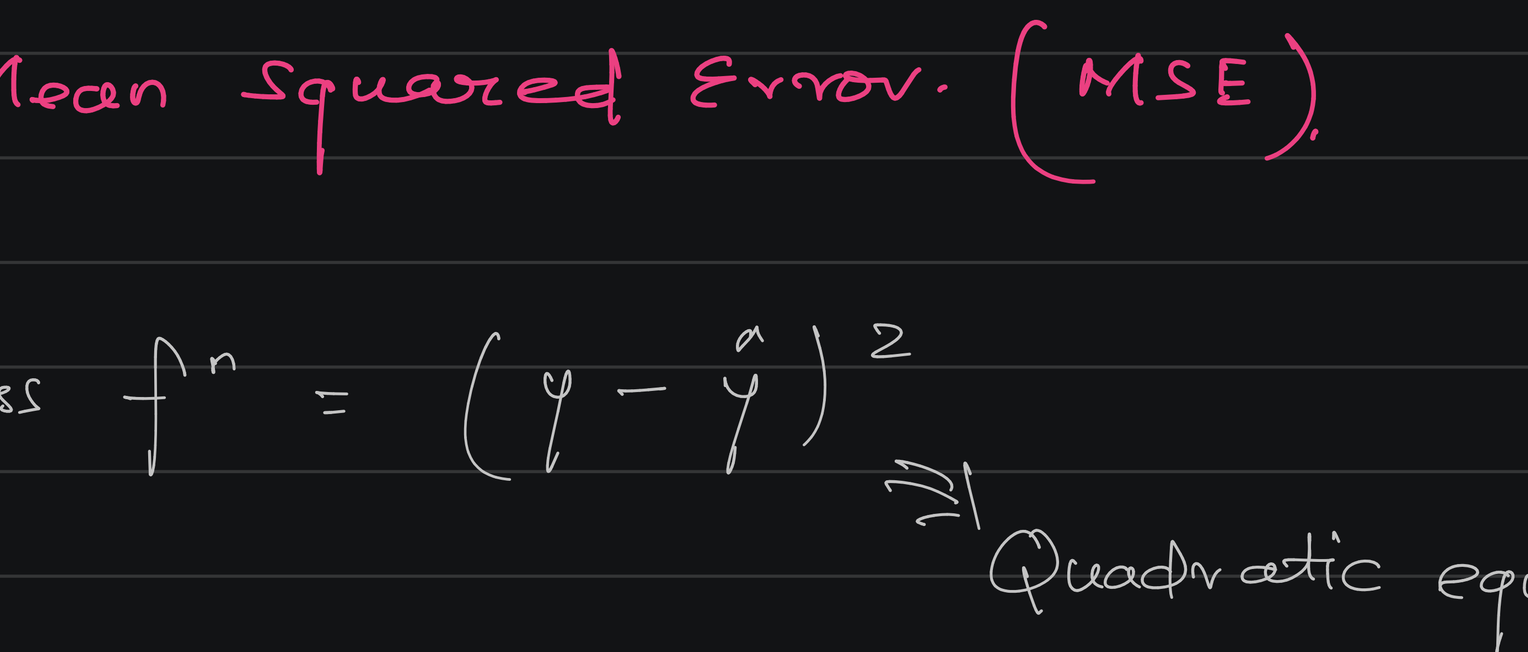
- MSE (Mean squared error)
- MAE (Mean absolute error)
- Huber loss.
- RMSE (Root mean square error).

(1.) Mean Squared Error. (MSE).

$$\text{loss } f^n = (y - \hat{y})^2$$

⇒ Quadratic equation.

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



Advantage

1. MSE is differentiable.
2. It has 1 local or 1 global minima.
3. It converges faster.

Disadvantages.

1. Not robust to outliers.

2. Mean Absolute Error.

$$\text{Loss fn} = |y - \hat{y}|$$

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

Advantage

1. Robust to outliers.

Disadvantages.

1. Convergence is slow.

(3.) Huber loss.

Combination of MSE & MAE.

$$\text{Cost-fn} = \begin{cases} \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, & \text{if } |y-\hat{y}| \le \boxed{\delta} \text{ hyperparameter} \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2, & \text{otherwise.} \end{cases}$$

$\Rightarrow$ MSE.

$\Rightarrow$ MAE

Advantage:

(1.) Takes care of both the issues for MSE and MAE.

(*) Loss & Cost function for classification problems.

CROSS ENTROPY
- → BINARY CROSS ENTROPY {BINARY}.
- → CATEGORICAL " " {Multiclass}.
- → Sparse categorical cross. {Multiclass} entropy.

1. Binary Cross Entropy.

$$\Rightarrow \text{Log loss function.}$$

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

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

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

$\rightarrow$ Sigmoid activation function.

(as binary classificat^n)

2. Categorical Cross Entropy.

(Multiclass classification)

Initialize

i	f_1	f_2	f_3	o/p	(j=1) Good	(j=2) Bad	(j=3) Neutral
1	2	3	4	Good	1	0	0
2	5	6	7	Bad	0	1	0
3	8	9	10	Neutral	0	0	1

C = No. of categories. = 3 (Initialize).

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

Or

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

** $\hat{y}_{ij}$ => Soft-max activation = Soft(z).

$$= \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} = \frac{e^{z_i}}{e^1 + e^2 + \dots + e^k}$$

(3.) Sparse Categorical Cross Entropy.

↳ Similar to categorical cross entropy.

but,

Probabilities for Sof(z) = [0.2, 0.3, 0.5]

o/p. ⇐ Highest prob.

Ignore all others and just give the o/p as the one with highest probability.

e.g. Used in image classification.

Disadvantage

→ Losing all other information.

(F) Right Combination

Activation applied ⇓

	Hidden Layers	O/p Layer	Problem Statement	Loss function
(1)	Relu or its Variants	Sigmoid	Binary classification	Binary Cross Entropy
(2)	Relu or its Variants	Softmax	Multi class	Categorical or Sparse CE
(3)	Relu or its Variants	linear	Regression	MSE, MAE, Huber loss, RMSE

(2.) Optimizers.

↳ used for backward propagation.

(1.) Gradient Descent.

(2.) Stochastic Gradient Descent (SGD).

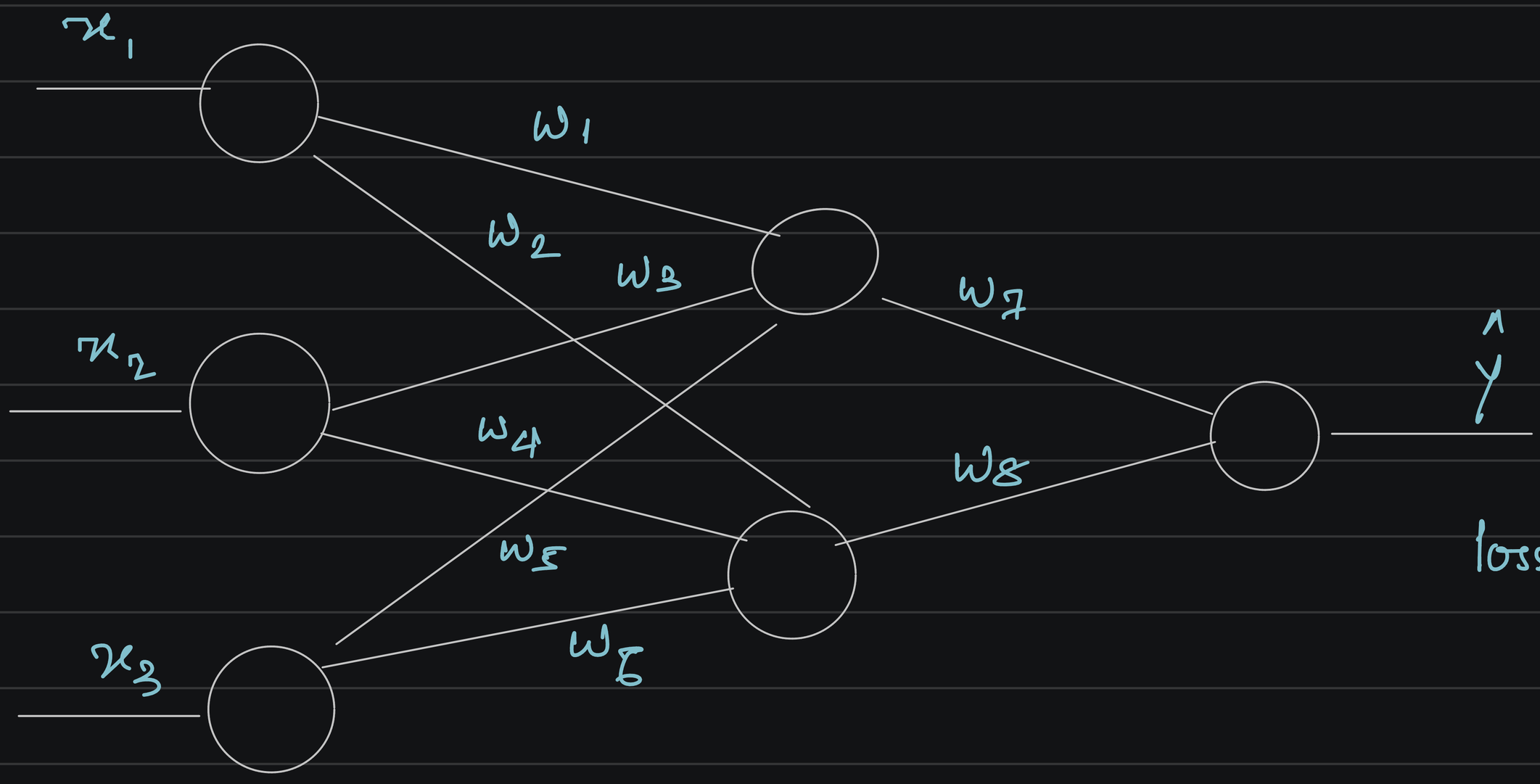
(3.) Mini batch SGD.

(4.) SGD with momentum.

(5.) Adagrad and RMS Prop.

(6.) Adam Optimizers.

(1.) Gradient Descent. Optimizer



Weight Updation formula.

$$w_{new} = w_{old} - \eta \left[\frac{\partial L}{\partial w_{old}} \right] \Rightarrow \text{slope}$$

!!

Chain rule of derivative.

In MSE,

$$\text{loss} = (y - \hat{y})^2$$

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

Epochs vs Iteration.

[Dataset — 1000 clodapoints]

(F.P) 1000 datapoints → $\hat{y}_i \Rightarrow$ Cost

weights updated (B.P.) ←

} EPOCH 1.

(F.P) 1000 data → $\hat{y}_1 \Rightarrow$ Cost.

weights updated. (B.P) ←

} EPOCH 2

⋮

Until Cost ↓↓↓

Cost doesn't change in further EPOCHs. ⇒ Model is ready.

Advantage.

* Convergence will happen.

Disadvantage.

* for huge datasets,

(Huge RAM & GPU required)

⇓ Parallel procesing required.

Resource intensive.

(2.) Stochastic Gradient Descent. (SGD)

If dataset ≡ 1000 datapoint.

Epoch. 1

Iteration 1 ⇄ 1 datapoint ⇄ Loss

⋮

Iteration 1000 ⇄ 1 datapoint ⇄ loss.

→ Not very resource intensive.

Epoch 2

⋮

Epoch 100.

Total 100 × 1000 iterations

!!

Time complexity ↑.

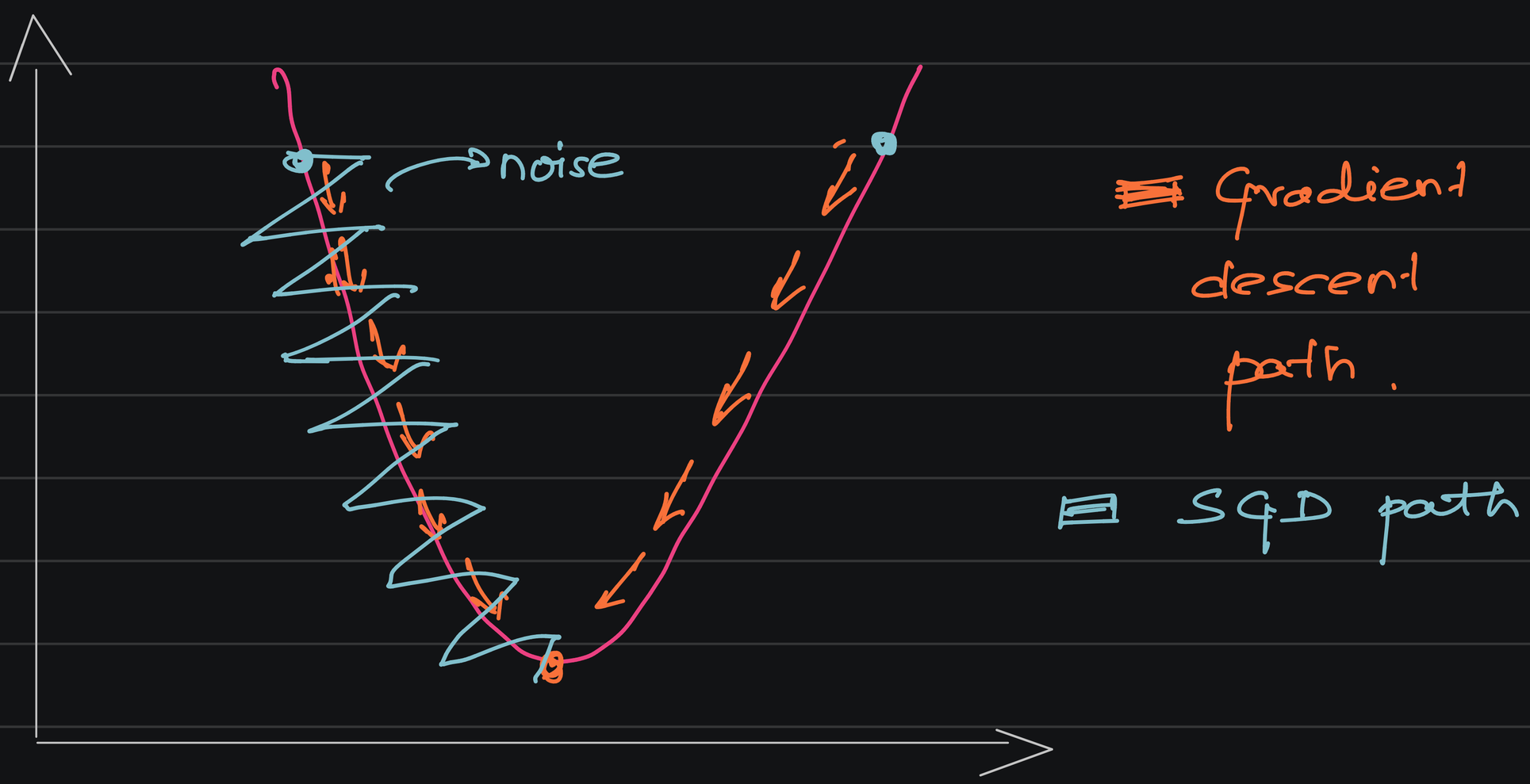
Advantage.

* Solves resource issue.

Disadvantage.

* Convergence will take time.

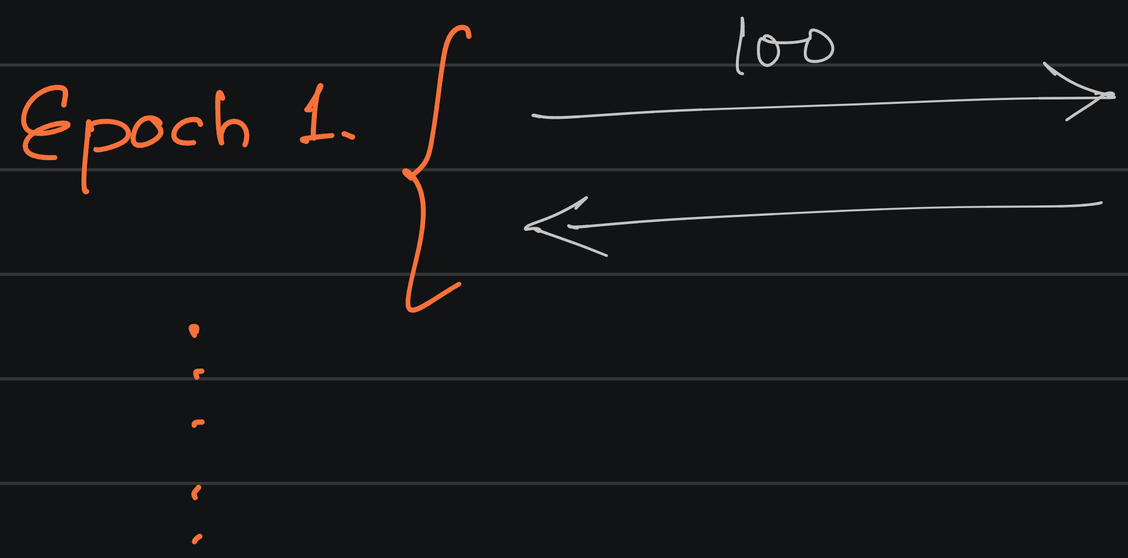
* Noise will get introduced due to procesing of each datapoint individually.



3. Mini-batch SGD.

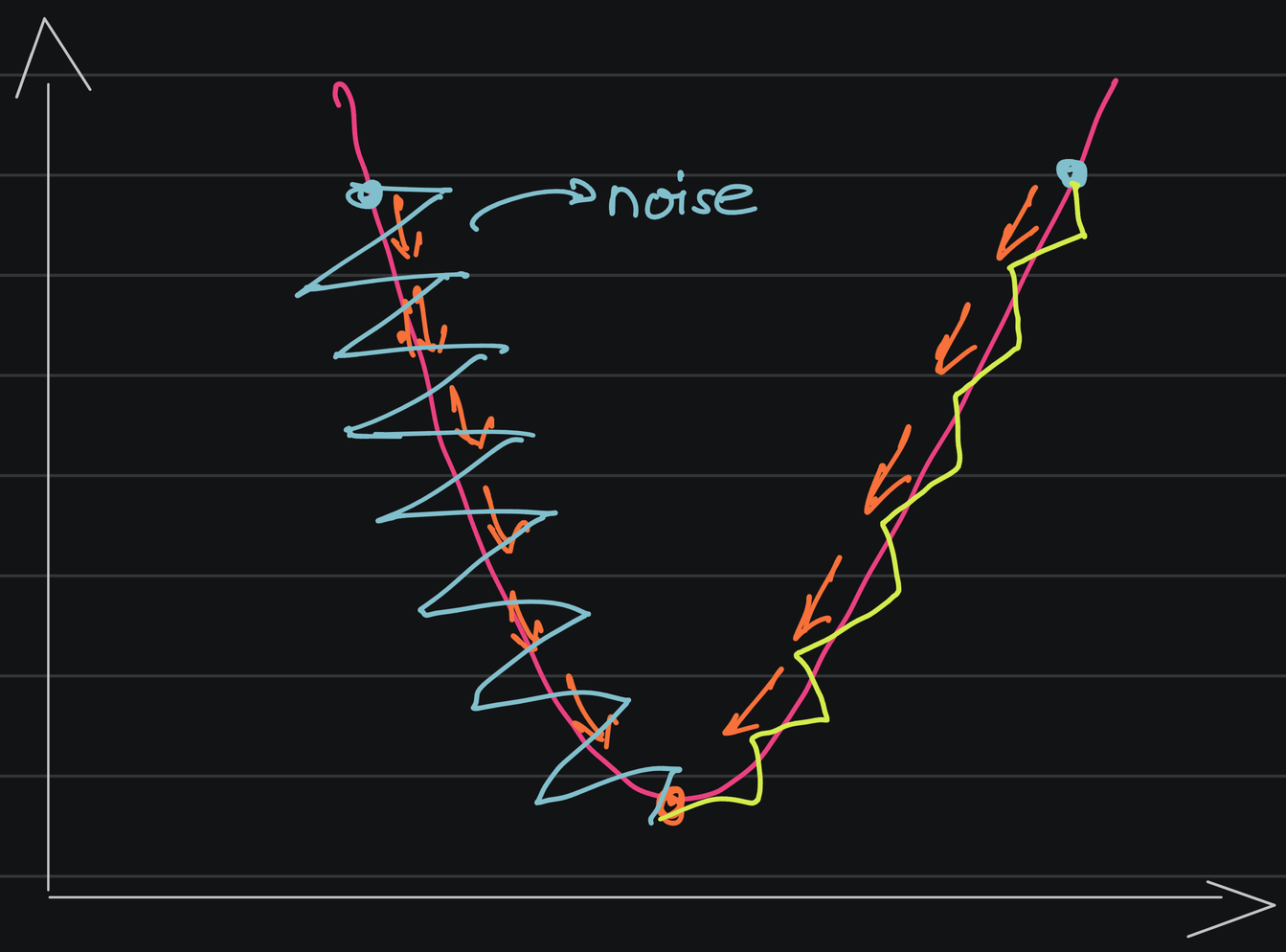
Epoch, Iteration, Batch size

If datapoints = 10000, batch size = 100



Advantage.

* Solves resource issue by taking batches of smaller numbers.
* Noise gets reduced as batch of datapoints is taken instead of individual datapoints.
* Convergence speed ↑.



Gradient descent path.

SGD path

Mini batch SGD path.

Disadvantage

* Noise still exists.

4. SGD with Momentum.

$$w_{new} = w_{old} - \eta \frac{\partial h}{\partial w_{old}}.$$

$$b_{new} = b_{old} - \eta \frac{\partial h}{\partial b_{old}}.$$

$$w_t = w_{t-1} - \eta \frac{\partial h}{\partial w_{t-1}}$$

→ Apply exponential weighted average.

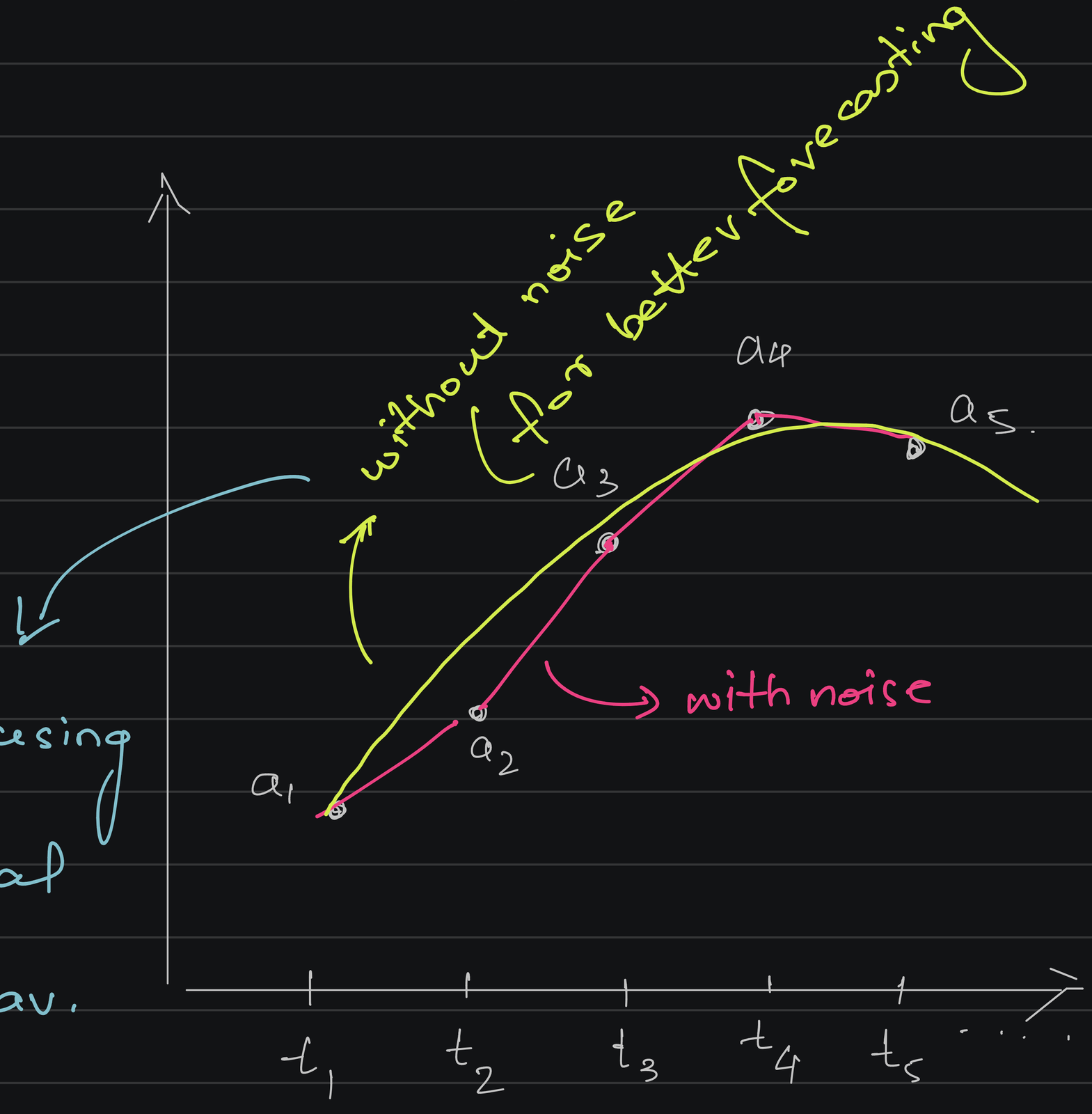
Exponential Weight Average:

Time t_1 t_2 t_3 t_4 ..... t_n.

Value. a_1 a_2 a_3 a_4 ..... a_n.

$V_{t_1} = a_1$

Created using exponential weight av.



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

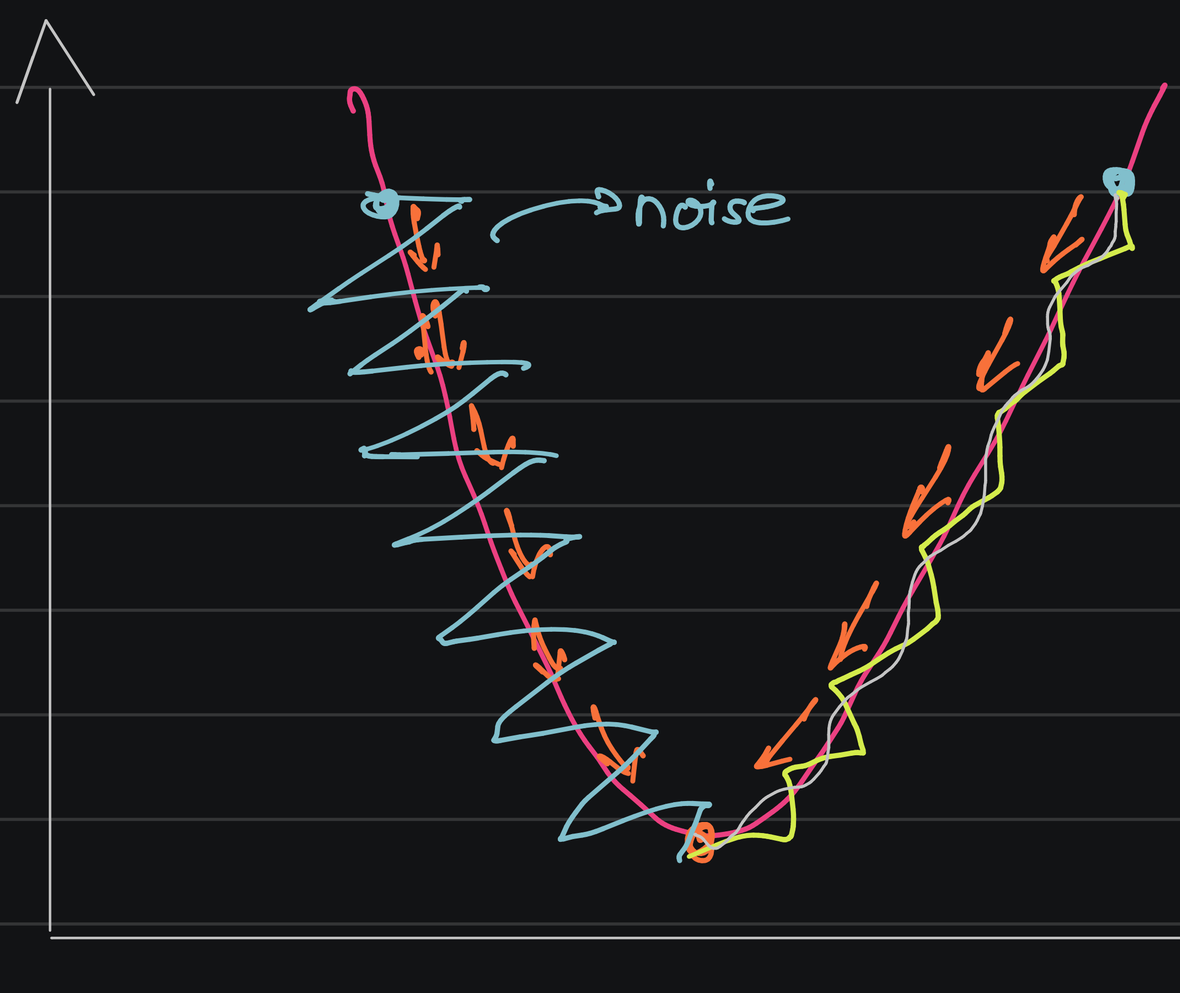
$= 0.95 * a_1 + 0.05 * a_2$ [Let $\beta = 0.95$]

has more control.

$V_{t_3} = \beta * V_{t_2} + (1-\beta) * a_3$.

$$= \quad 0.95 * (v_{t_2}) + 0.05(a_3)$$

more control of the curve.



Gradient descent path.

SGD path

Mini batch SGD path.

SGD with momentum path. (Smoother path).

Advantage:

* Reduces the noise
* Smoothens the noise
* Quick convergence.

(5.) Adagrad. { Adaptive Gradient Descent }

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

↓↓.

$$w_t = w_{t-1} - \eta' \frac{\partial L}{\partial w_{t-1}}.$$

$$\eta' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$$

epsilon = small value.

- to avoid dividing by 0 condition when $\alpha_t = 0$.

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

↓

with each iteration.

$\alpha_t \uparrow \equiv \eta' \downarrow$

So, η' keeps getting smaller by the time of convergence.

→ Dynamic learning rate.

↓

keeps decreasing as it reaches the global minima.

Disadvantage

$$\text{if} \quad \alpha_t = \sum_{i=1}^{t} \left(\frac{\partial \mathcal{L}}{\partial w_t} \right)^2 \Rightarrow \text{Large value.}$$

(in deep layered neural networks)

$$w_t \approx w_{t-1}.$$

↓

Vanishing gradient problem.

* So, η' = Possibility to become a small value.

≈ 0.

6. Adadelta and RMS Prop.

$$\eta' = \frac{\eta}{\sqrt{Sdwt + \epsilon}}$$

↓ instead of α_t

Exponential weighted average.

(smoothening concept is used.)

Initially, $Sdwt = 0$

$$Sdwt = \beta * Sdw_{t-1} + (1-\beta)\left(\frac{\partial \mathcal{L}}{\partial w_t}\right)^2$$

7. Adam Optimizer → Best so far.

⇓

SGD with momentum + RMS Prop [Dynamic Learning Rate]

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

V_{dw}.

$$V_{dwt} = \beta * v_{dw_{t-1}} - (1-\beta)\frac{\partial \mathcal{L}}{\partial w_{t-1}}$$

$$\eta' = \frac{\eta}{\sqrt{s_{dw_t} + \epsilon}}$$

$$b_t = b_{t-1} - \eta' V_{db}.$$

$$V_{db} = \beta * V_{dbt-1} - (1-\beta)\frac{\partial \mathcal{L}}{\partial b_{t-1}}$$

$$\eta' = \frac{\eta}{\sqrt{s_{db_t} + \epsilon}}$$