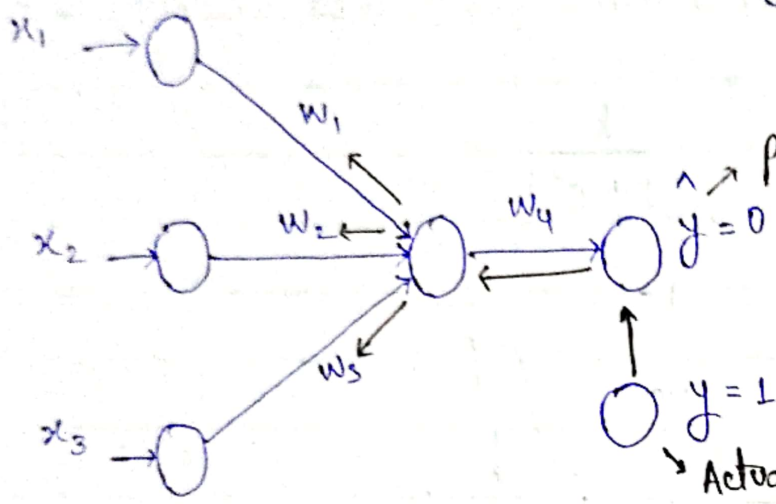
How to train Neural Network with Back Propagation

$$y = [x_1 w_1 + x_2 w_2 + x_3 w_3] + b_1$$

$$z = Act(y)$$

learning rate

$$W_{4New} = W_{4old} - \eta \frac{\partial L}{\partial W_4}$$



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

$$= (1 - 0)^2$$

$$= 1$$

reduced

→ We have to update weights during training phase to decrease loss function.

∴ To minimize loss, Optimizers are used.

→ To update the weights Backpropagation is done.

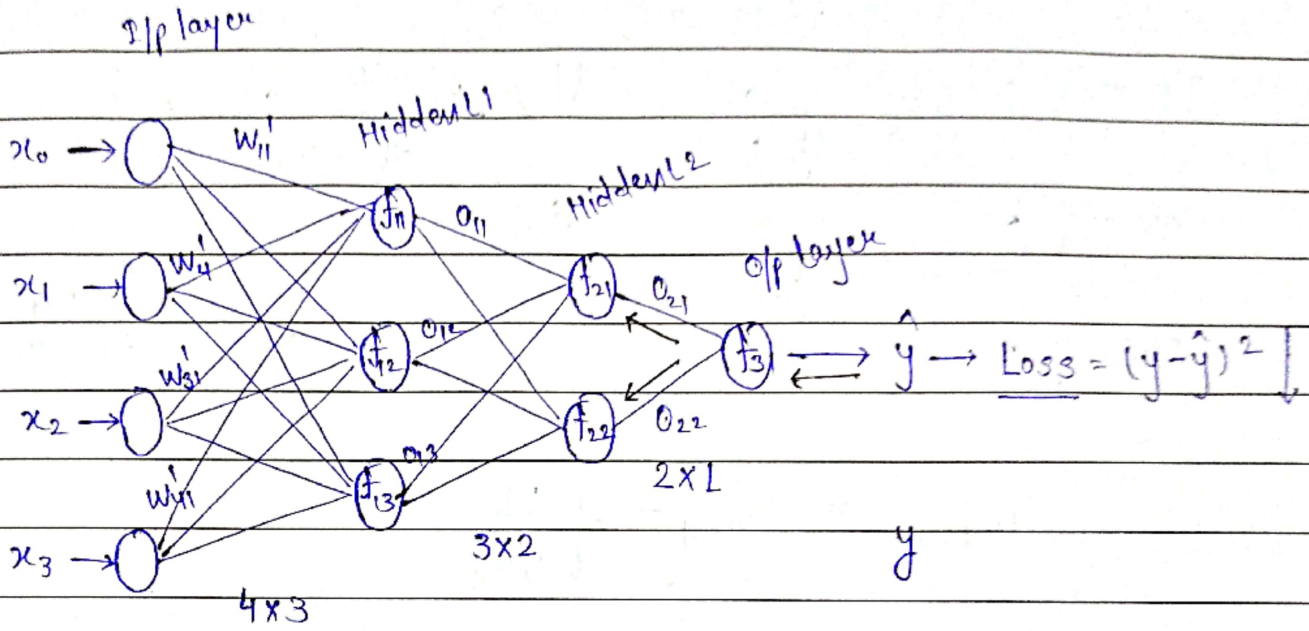
Similarly,

$$W_{3New} = W_{3old} - \eta \frac{\partial L}{\partial W_3}$$

learning rate

POORNIMA

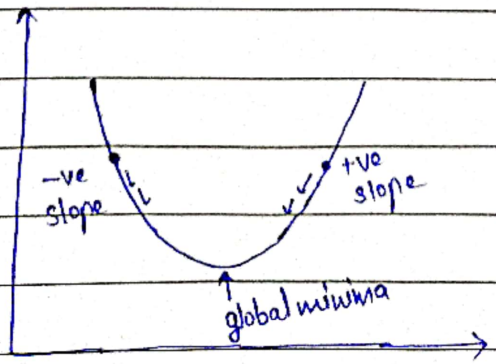
How to train Multilayer Neural Network & Gradient Descent:



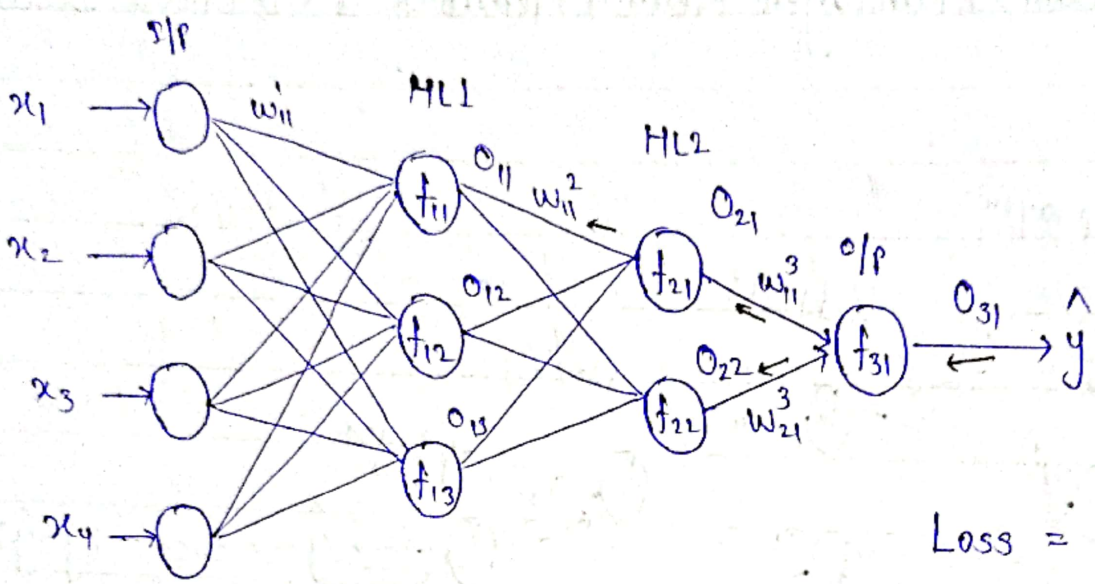
→ Weights Updation: $W_{new} = W_{old} - \eta \frac{\partial L}{\partial w}$

η → learning rate

Gradient Descent:



Chain Rule of Differentiation with Back Propagation:



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

$$w_{11\,new}^3 = w_{11\,old}^3 - \eta \frac{\partial L}{\partial w_{11}^3}$$

$$\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{11}^3}$$

$$\frac{\partial L}{\partial w_{21}^3} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{21}^3}$$

$$\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial w_{11}^2}$$

POORNIMA

Vanishing Gradient Problem:

→ The difficulty of training deep neural networks because the gradients, used to update the networks weights, become extremely small as they are propagated backward from the output layer to the earlier layers.

Root Cause

It typically arises when using sigmoid or tanh activation functions, which squash input into a small range, causing derivatives to be less than 1, and their repeated multiplication causes gradient to shrink exponentially.

→ Solutions include using ReLU, proper weight initialization, batch normalization, and Residual connections.

Exploding Gradient Problem:

→ Gradients grow exponentially large during backpropagation in deep neural networks

Root Cause

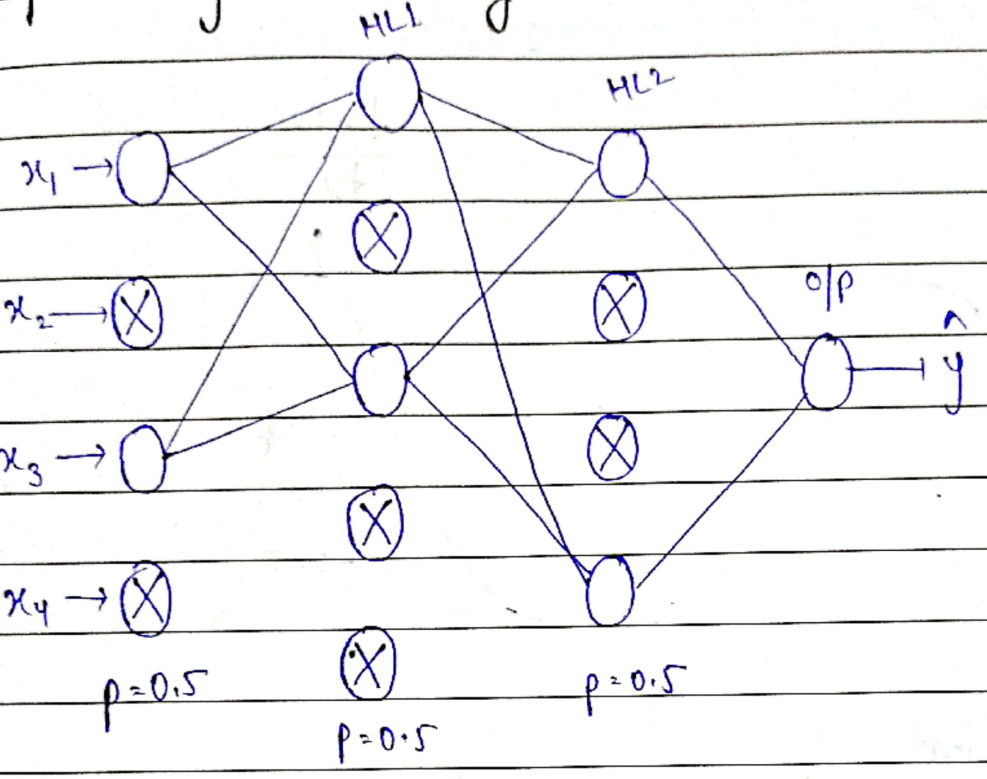
→ Caused by repeatedly multiplying large weights, especially in deep networks or RNNs.

→ The exploding gradient problem leads to unstable training by causing excessively large weight updates, often resulting in divergence or NaN values.

By applying techniques like Gradient clipping, proper weight initialization and normalization, this issue can be effectively managed, ensuring stable and successful model training.

POORNIMA

Dropout layers and Regularization:



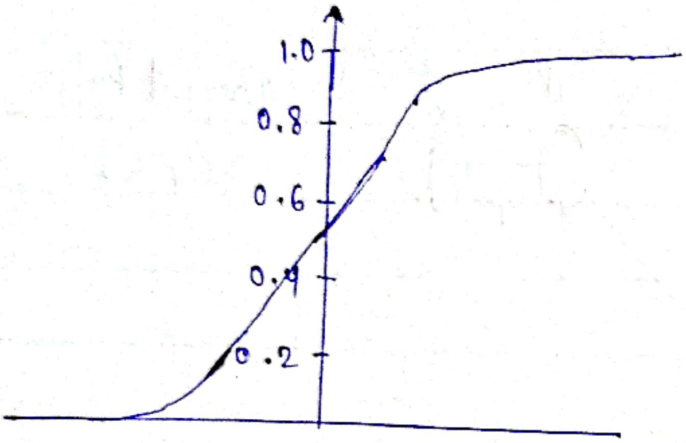
→ Dropout is a regularization technique used in neural networks to prevent overfitting.

- Randomly drops out a fraction of the neurons during training.
- During each training step, selected neurons are temporarily ignored.
- At test time, all neurons are used, but their outputs are scaled to account for the dropout.

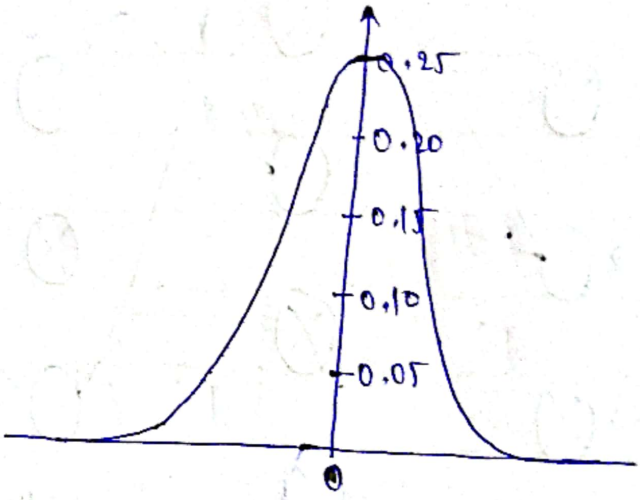
Activation Functions:

1. Sigmoid Function : $\sigma(x) = \frac{1}{1+e^{-x}}$

$\sigma(x)$



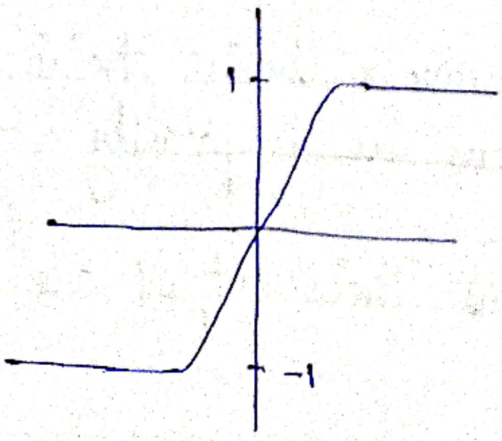
$\frac{d\,\sigma(x)}{dx}$



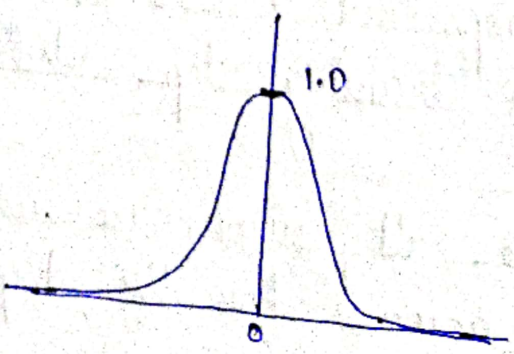
2. tanh Function:

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

$\tanh(x)$



$\frac{d\,\tanh(x)}{dx}$

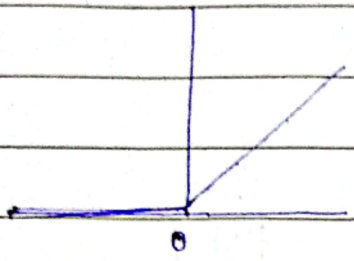


POORNIMA

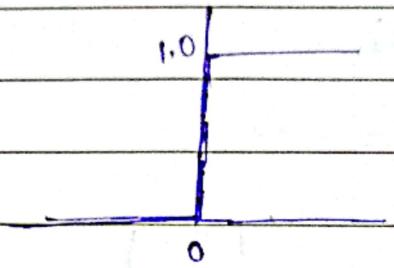
3. ReLU Function:

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

ReLU(x)



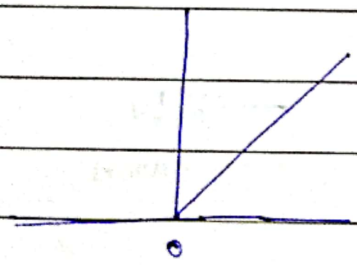
$\frac{d\,ReLU(x)}{dx}$



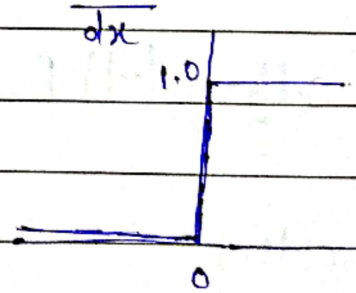
4. Leaky ReLU function:

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

f(x)



$\frac{df(x)}{dx}$

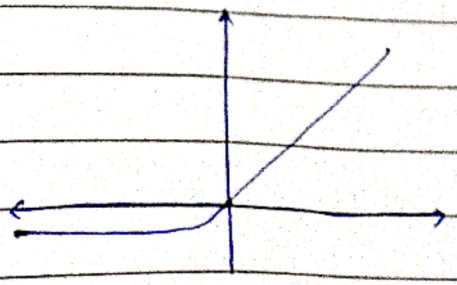


5. ELU (Exponential Linear Units) function:

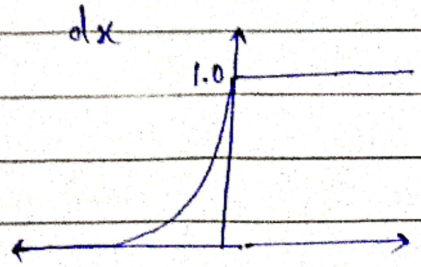
$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & \text{otherwise} \end{cases}$$

learning parameter

f(x)



$\frac{df(x)}{dx}$

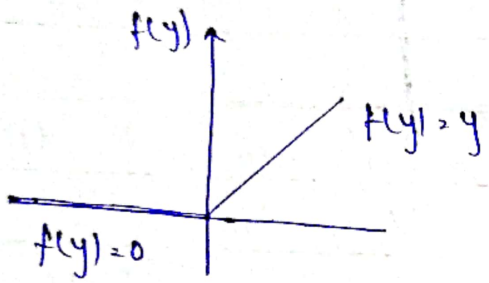


6. PReLU (Parametric ReLU):

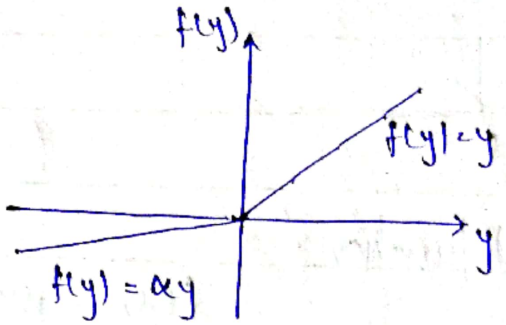
learning parameter

$$f(x) = \begin{cases} x & , \quad x > 0 \\ \alpha * x & , \quad x \leq 0 \end{cases}$$

$f(x)$

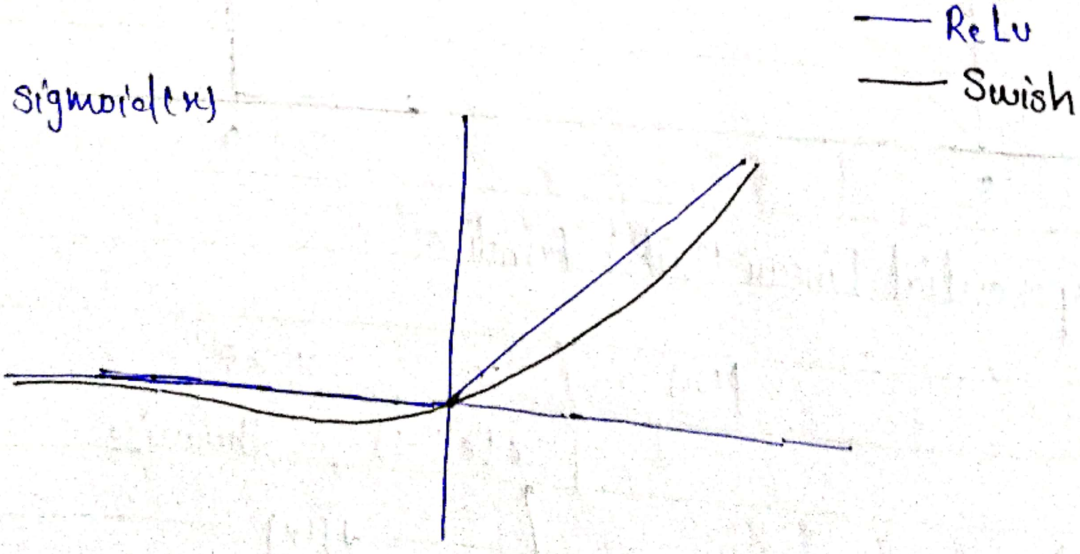


$\frac{\partial f(x)}{\partial x}$



7. Swish (A self-gated) function:

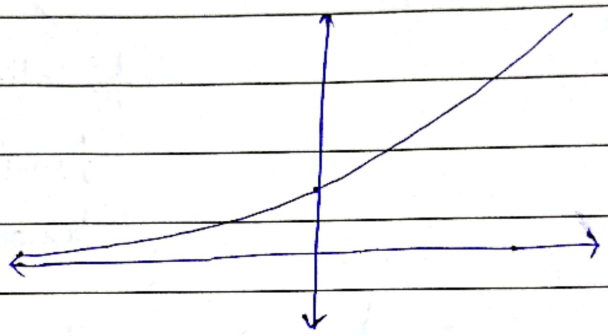
$y = x * \text{sigmoid}(x)$



POORNIMA

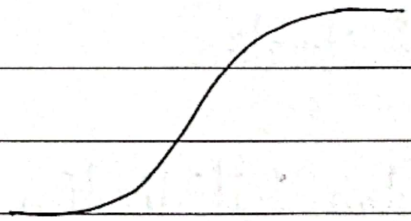
8. Softplus Function:

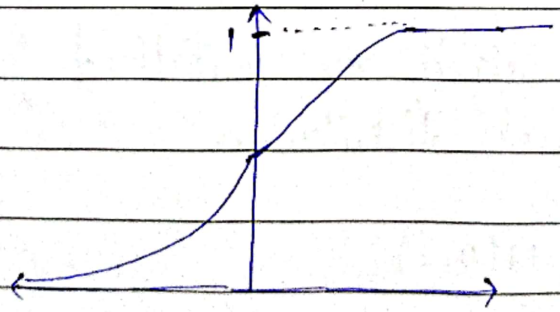
$$f(x) = \ln(1 + \exp x)$$



9. Softmax Function:

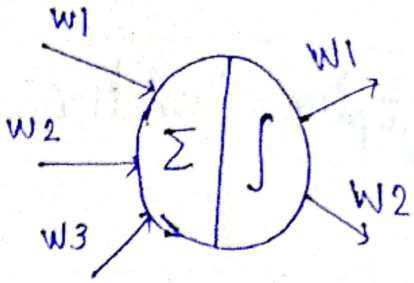
$$\frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$





Weight Initialization in Neural Networks:

- Weights should be small
- Weights should not be same
- Weights should have good variance.



for above neuron,

fan_in = 3

fan_out = 2

1. Zero Initialization:

All the weights are assigned zero as the initial value. is zero initialization. This kind of initialization is highly ineffective as neurons learn the same feature during each iteration.

2. Random Initialization:

a) Random Normal: The weights are initialized from values in a normal distribution.

$$W_i \sim N(0,1)$$

b) Random Uniform: The weights are initialized from values in a uniform distribution.

$$W_i \sim N(0,1)$$

POORNIMA

3. Xavier / Glorot Initialization:

The weights are assigned from values of a uniform distribution as follows:

$$w_i \sim U\left[-\sqrt{\frac{\sigma}{fan\_in + fan\_out}}, \sqrt{\frac{\sigma}{fan\_in + fan\_out}}\right]$$

→ Xavier / Glorot Initialization often termed as Xavier Uniform Initialization, is suitable for layers where activation function used is Sigmoid.

4. Normalized Xavier / Glorot Initialization:

$$w_i \sim N(0, \sigma)$$

$$\sigma = \sqrt{\frac{6}{fan\_in + fan\_out}}$$

5. He Uniform Initialization:

$$w_i \sim U\left[-\sqrt{\frac{6}{fan\_in}}, \sqrt{\frac{6}{fan\_out}}\right]$$

→ He Uniform Initialization is suitable for layers where ReLU is used.

6. He Normal Initialization:

$$W_i \sim N[0, \sigma]$$

$$\sigma = \sqrt{\frac{2}{fan\_in}}$$

→ Suitable where ReLU is used.

POORNIMA

$\frac{\partial L}{\partial W_{old}}$ → n data points → Gradient Descent

Stochastic Gradient Descent

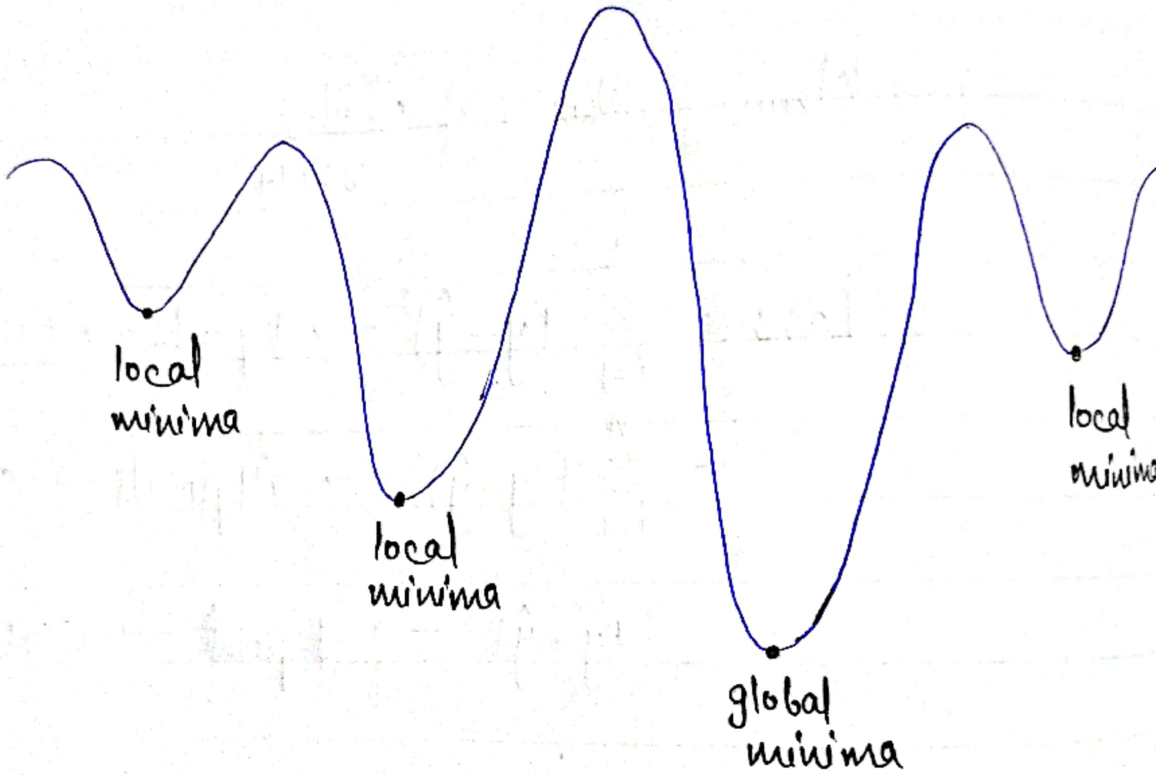
$$W_{new} = W_{old} - \eta \times \frac{\partial L}{\partial W_{old}}$$

$$\text{Loss} = \sum_{i=1}^{k} (y - \hat{y})^2 \longrightarrow k \text{ points} \longrightarrow \text{Mini Batch SGD}$$

$$= \sum_{i=1}^{n} (y - \hat{y})^2 \longrightarrow \text{all points} \longrightarrow \text{GD}$$

$$= (y - \hat{y})^2 \longrightarrow 1 \text{ point} \longrightarrow \text{SGD}$$

Global Minima vs Local Minima:



- A global minimum is the absolute lowest point on the loss surface.
- A local minimum is a point where the loss is lower than neighboring points but not the lowest overall.