Forward and Backward Propogation

Forward Propogation

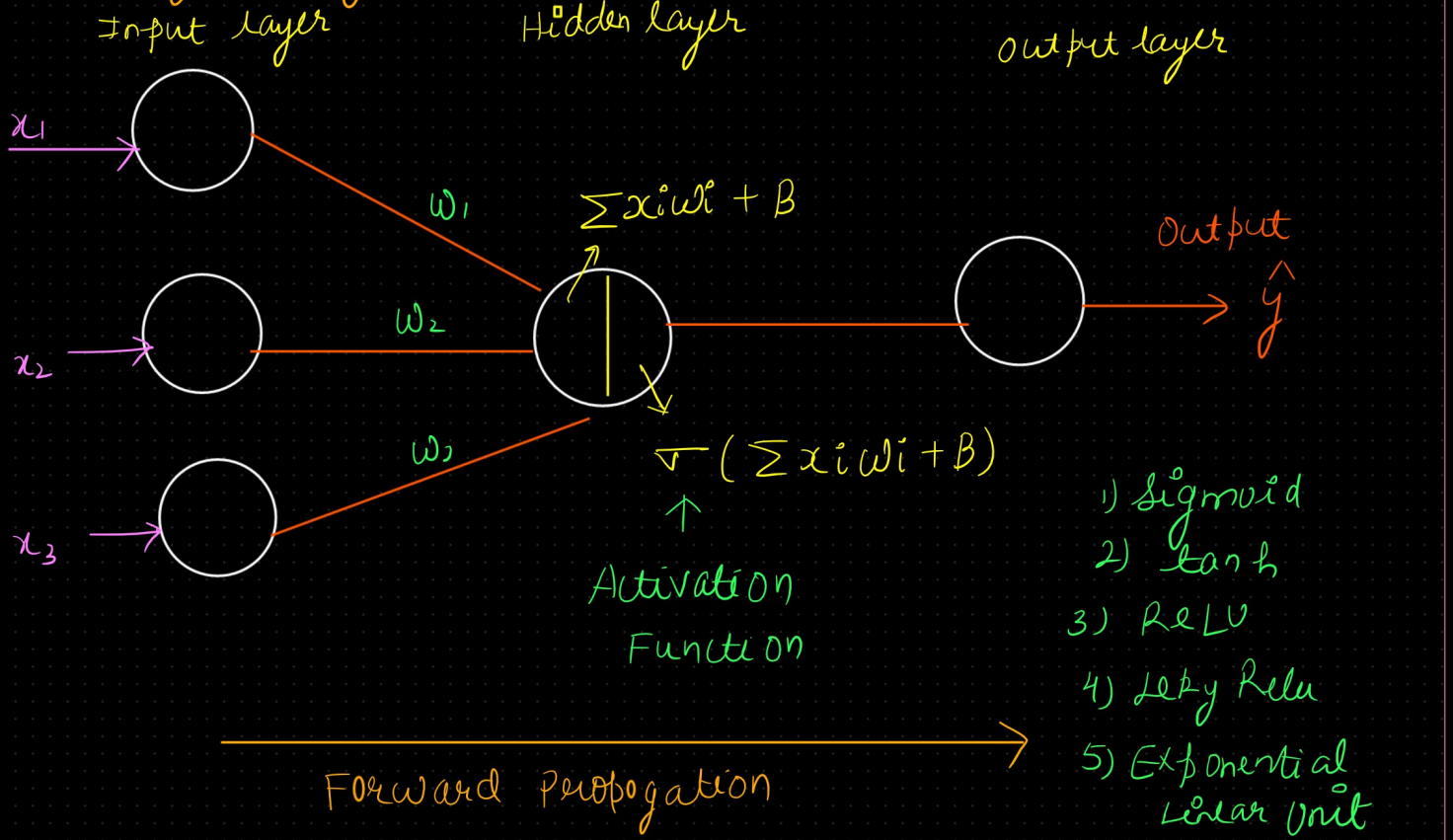
→ Forward propagation is the process of moving from the input layer through the hidden layers to the output layer of a neural network to make predictions.

→ Each neuron in a neural network layer receives inputs from the previous layer, applies weights and biases, and then passes the result through an activation function to produce an output.

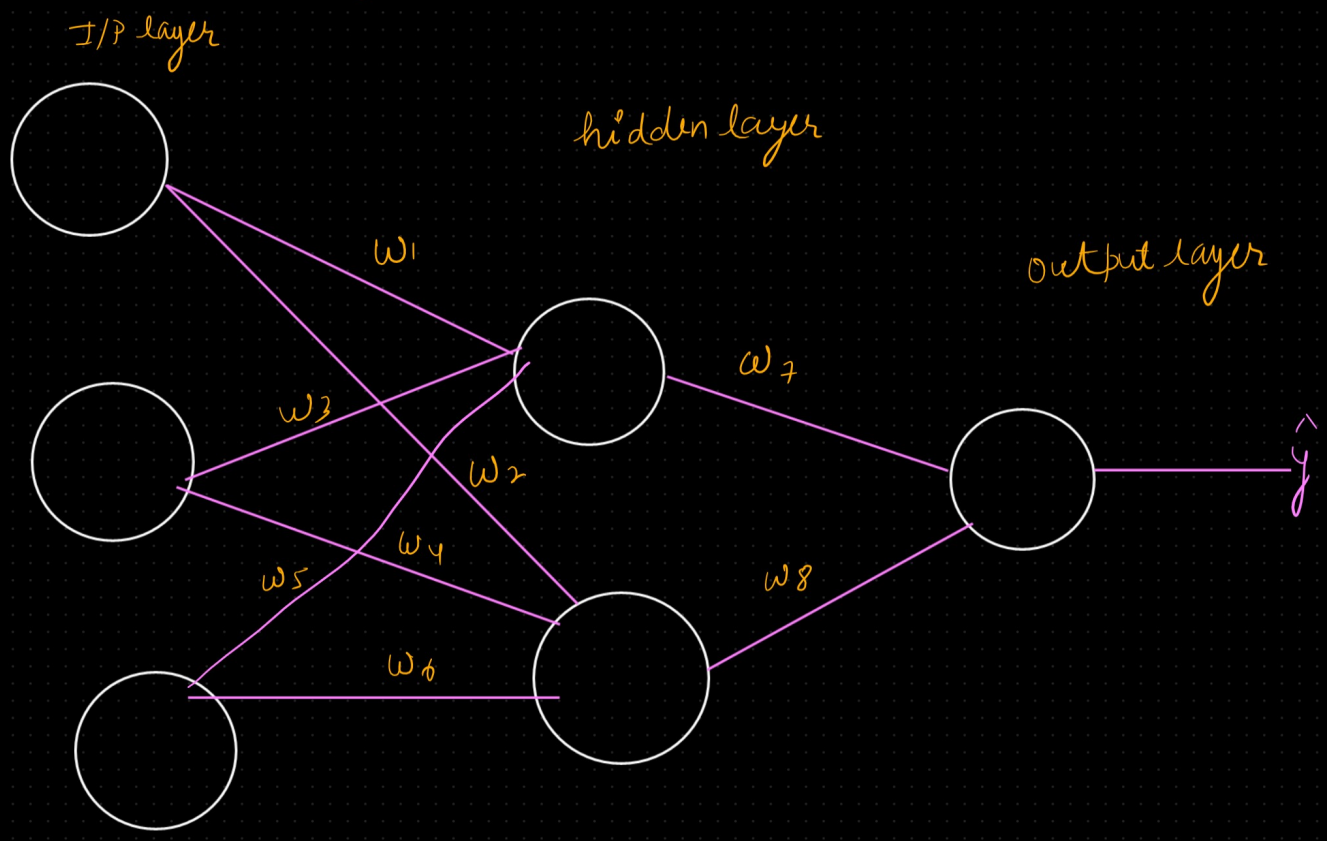
$$y = \sigma(\sum x_i w_i + b)$$

→ This process is repeated layer by layer until the output layer is reached, and the final predictions are obtained.

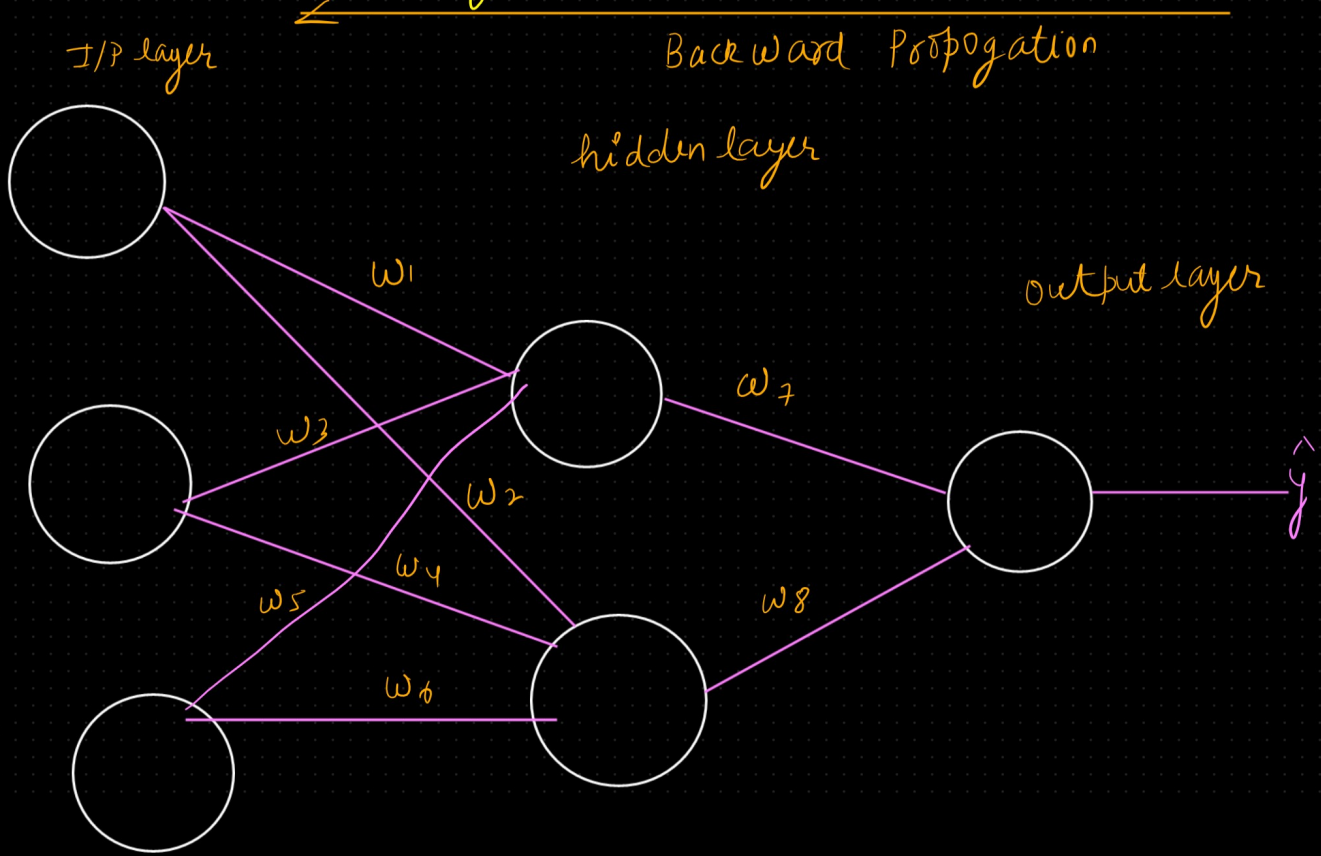
Single-layered Neural Network



Multi-layered Neural Network



Backward Propogation



① Error → Diff b/w predicted output and actual target output

$$\text{Error} = \hat{y} - y$$

① Weights update

② loss func^n ↓ ↓

→ Backward propagation, also known as backpropagation, is the process of updating the weights of the neural network based on the error between the predicted output and the actual target output

Step 2:

diff. → rate of change

It involves calculating the gradient of the loss function with respect to each weight and bias in the network, and then using this gradient to update the weights and biases in the opposite direction of the gradient to minimize the loss.

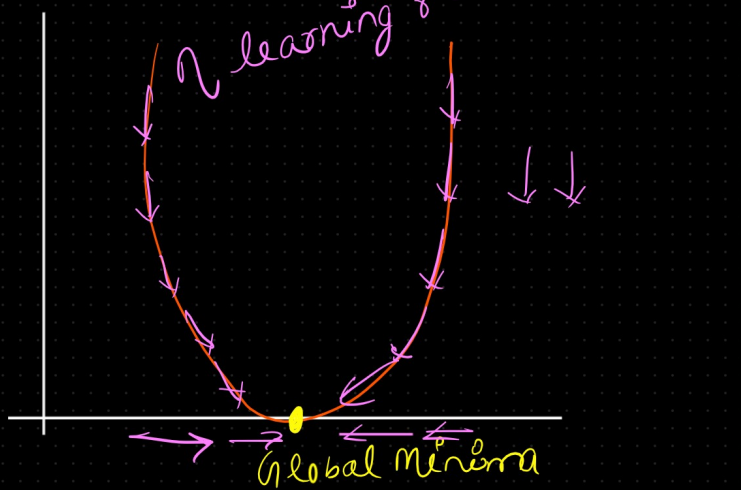
single data point

loss function ↔ Regression → MSE
Classification → Log loss / Cross Entropy

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

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

batches data points



Step 3:

The gradient of the loss function with respect to the weights and biases is calculated using the chain rule of calculus, which allows us to propagate the error backward through the network.

Chain Rule

Derivative → Rate of change

$$f(x) = y$$

$$f(x) = x^2$$

∴ Power Rule

$$f'(x) = nx^{n-1}$$

n = power
x = variable

$$f(x) = x^2$$

$$f'(x) = 2x^{2-1} = 2x$$

$$f(x) = 4x^3$$

$$f'(x) = 12x^2$$

Partial derivative

$f(x)$

$f(x, y)$

w.r.t x

$\frac{\partial f}{\partial x}$ taking y as constant

Similarly $\frac{\partial f}{\partial y}$ w.r.t y taking x as constant

$\therefore f(x, y) = x^2 + 2xy + y^2$

$\therefore \frac{\partial f}{\partial x} = \frac{\partial}{\partial x}(x^2 + 2xy + y^2)$

$= 2x + 2y$

$\frac{\partial f}{\partial y} = \frac{\partial}{\partial y}(x^2 + 2xy + y^2)$

$= 2x + 2y$

Chain Rule

$f(x)$ $\quad$ $g(x)$

$\rightarrow f(g(x))$

The chain rule says that to find the rate of change of the outer function with respect to the inner function, you multiply the rate of change of the outer function with respect to its input by the rate of change of the inner function with respect to its input

$\frac{dy}{dx}$ = Derivative of y w.r.t x

$f(g(x)) \Rightarrow \frac{dy}{dx}$ $\quad$ $f(x) = y$ $\quad$ $\therefore \frac{dy}{dx}$

$y = f(u)$ and $u = g(x)$

$\therefore$ Chain Rule

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

$\therefore f(u) = u^2$ and $g(x) = 3x$

$\therefore \frac{dy}{du} = 2u$ $\quad$ $\therefore \frac{du}{dx} = 3$

$\therefore \frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} = 2u \cdot 3 = 6u$

$= 6(3x) = 18x$

Step4: loss function $\rightarrow L$

Weight $\rightarrow w$

$z \rightarrow \hat{y} - y$

By using chain Rule

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w}$$

$\Downarrow$

Gradient of loss function

$\rightarrow$ Step 5

This process is repeated layer by layer in the reverse order until the gradients for all weights and biases are calculated.

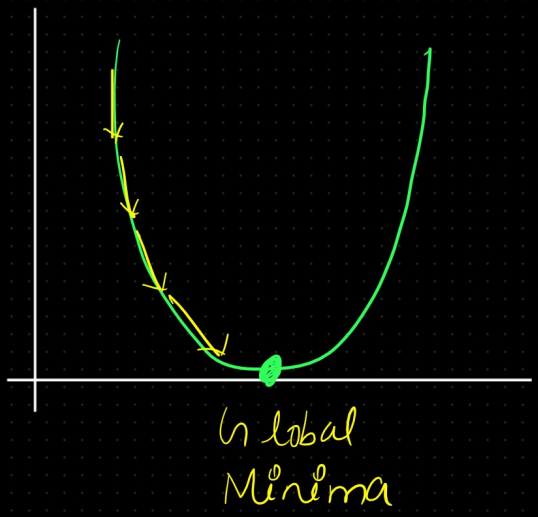
$\rightarrow$ Step 6

Once the gradients are obtained, the weights and biases are updated using an optimization algorithm such as gradient descent:

Weight Updation Formula

$$W_{new} = W_{old} - \alpha \cdot \frac{\partial L}{\partial w}$$

α = learning rate $\Rightarrow$ step size



The intuition behind backward propagation is that it adjusts the weights and biases in such a way that the error between the predicted output and the actual output is minimized. It does this by iteratively updating the parameters in the direction that reduces the error, thus improving the network's ability to make accurate predictions.

α – learning rate

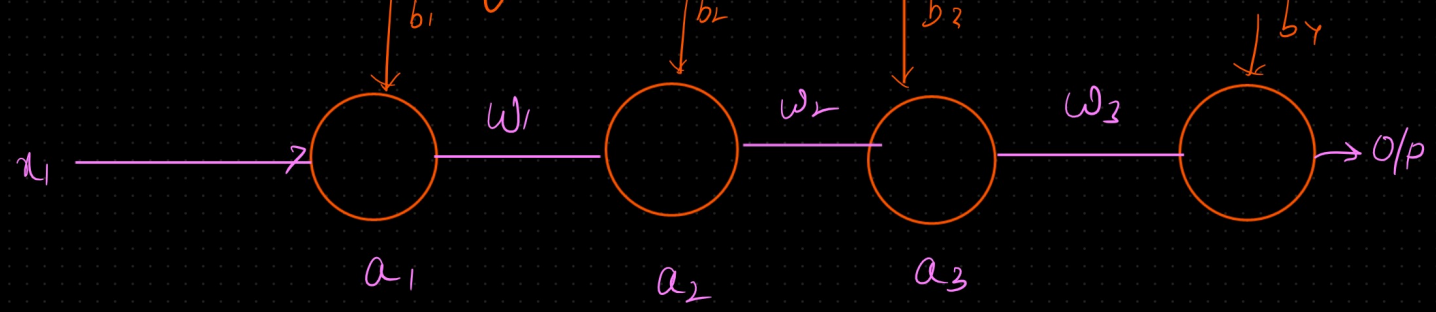
hyperparameter

↓

step size

The learning rate is a hyperparameter that determines the step size at which the weights of a neural network are updated during training. In the context of training a neural network using optimization algorithms like gradient descent, the learning rate controls how much we are adjusting the weights of our network with respect to the loss gradient.

Vanishing Gradient Problem



σ → sigmoid

$[0, 1)$ Funcn

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

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a_1} \times \frac{\partial a_1}{\partial a_2} \times \frac{\partial a_2}{\partial a_3} \approx \text{small number}$$

$$\therefore w_{new} = w_{old} - \eta (\text{small number})$$

$$\boxed{w_{new} \simeq w_{old}} \Rightarrow \text{vanishing gradient problem}$$

$\Downarrow$

No change in weight

Activation Func^n

$\downarrow$

Relu

$\hat{y} = \text{Relu}(a^{[1]})$

$=$