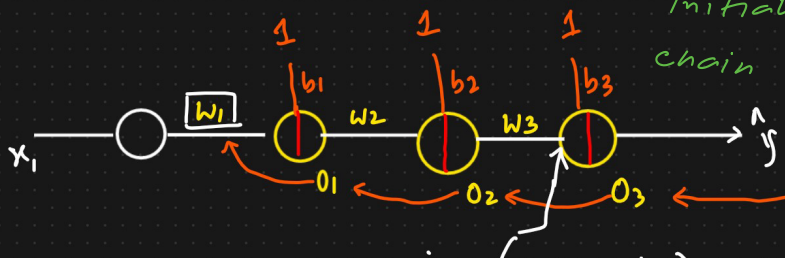
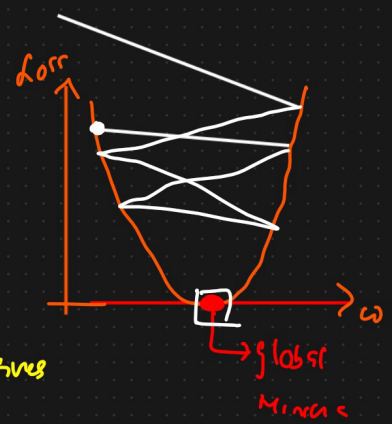
Exploding Gradient Problem ⇒ Weight Initialization Technique



Initializing with high value means chain derivative will give high Wnew

$\rightarrow z = (O_2 * w_3 + b_3) \rightarrow \sigma(z)$

Optimizers



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

$\{ W_{new} >>>> W_{1old} \quad W_{new} << W_{1old} \}$

Chain Rule of Derivatives

$$\frac{\partial h}{\partial W_{1old}} = \frac{\partial h}{\partial O_3} * \frac{\partial O_3}{\partial O_2} * \frac{\partial O_2}{\partial O_1} * \frac{\partial O_1}{\partial W_{1old}}$$

big * big * big * big ⇒ big value

Weight initialization ⇓ Very high value

$$\frac{\partial O_3}{\partial O_2} = \frac{\partial \sigma(z)}{\partial z} * \frac{\partial z}{\partial O_2}$$

$$= [0 - 0.25] * \frac{\partial (O_2 * w_3 + b_3)}{\partial O_2} = [0 - 0.25] * w_3 \Rightarrow 500 - 1000$$

✅ **Solutions:**

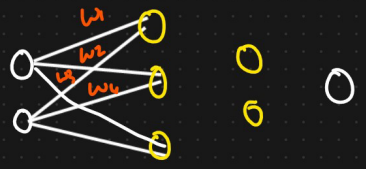
Problem	Solution
Gradients getting too large	Gradient Clipping
Poor initialization	Use He/Xavier Initialization
Very deep network	Use architectures like ResNet (skip connections)
High learning rate	Lower learning rate

Exploding gradients happen when gradients grow too large during backpropagation, often due to **deep networks or bad initialization.** They can break the training process. Fix it with good initialization, gradient clipping, and careful architecture design.

Gradient explode =
- Bad initialization
- Deep networks

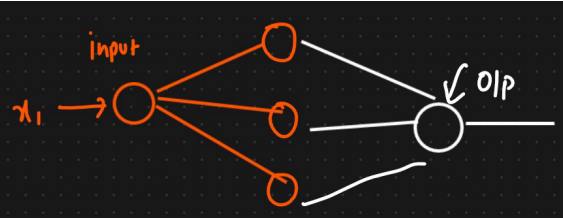
Weight Initializing Techniques

1. Uniform Distribution ✓
2. Xavier / Glorot Initialization ✓
3. Kaiming He Initialization ✓



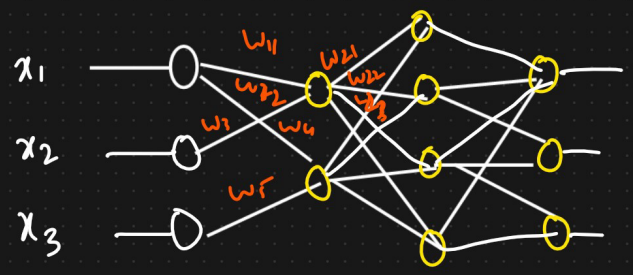
Key Points

1. Weights should be small ✓
2. Weights should not be same ✓
3. Weights should have good variance



input = 1

Output = 1



input = 3

O/p = 3

① Uniform Distribution

$W_{ij} \sim$ Uniform Distribution $\left[\frac{-1}{\sqrt{input}}, \frac{1}{\sqrt{input}}\right]$

$\left[\frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right]$

② Xavier/Glorot Initialization

Researcher $\longrightarrow$ Xavier Glorot

① Xavier Normal Init

$W_{ij} \sim N(0, \sigma)$

$$\sigma = \sqrt{\frac{2}{(input + output)}}$$

② Xavier Uniform

$W_{ij} \sim$ Uniform Distribution $\left[\frac{-\sqrt{6}}{\sqrt{input + Output}}, \frac{\sqrt{6}}{\sqrt{i/p + o/p}}\right]$

③ Kaiming HE Initialization

① He Normal

$$W_{ij} \sim N(0, \sigma)$$

$$\sigma = \sqrt{\frac{2}{\text{input}}}$$

② He uniform

$$W_{ij} \sim \text{Uniform Distribution} \left[-\sqrt{\frac{6}{\text{input}}}, \sqrt{\frac{6}{\text{input}}} \right]$$

CONCLUSION;

Q. why do we need weight Initialization Techniques?

when training a neural network, how we initialize the weights matters a lot - it can make or break our model's ability to learn.

Imp

✅ Key Reasons:

Reason	Explanation
Breaks Symmetry	If all weights are the same (e.g., zeros), neurons learn the same features — no diversity.
Avoids Vanishing/Exploding Gradients	Poor initialization can cause gradients to become too small or too large during backprop.
Speeds Up Convergence	Good initialization helps the model learn faster and more reliably.

✅ Essential Weight Initialization Techniques

Technique	Use When	Activation Function
Xavier (Glorot)	For balanced input/output layers	`tanh`, `sigmoid`
He Initialization	For deep networks with ReLU-based activations	`ReLU`, `Leaky ReLU`
Random Uniform	Simple or shallow networks	Any (but not recommended for deep networks)

🔁 Quick Recap:

- 🎯 **Xavier** → good for smooth activations
- ⚡ **He** → best for ReLU and its variants
- ⚠️ **Avoid** zeros or constant initializations