Vanishing Gradient Problem in ANN

💡 **IMP Interview Question**

- The **vanishing gradient problem** occurs when gradients become **extremely small** during backpropagation, causing early layers in a neural network to learn **very slowly or not at all**.
- This is a major issue in **deep neural networks** (especially those with sigmoid/tanh activations).
    - You face this problem when there are many **(8 to 10) hidden layers.**

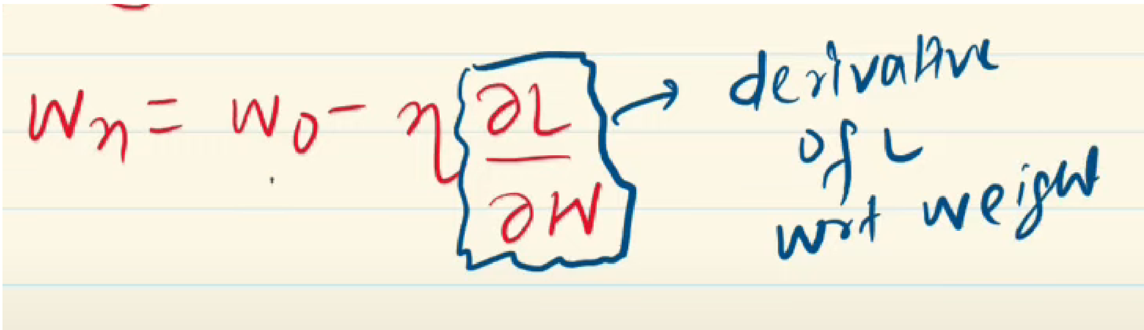
> **Occurs only in case of sigmoid & tanh activation functions.**

Why Does the Vanishing Gradient Occur?

Chain Rule

- During backpropagation, gradients are calculated using the **chain rule**

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_L} \cdot \frac{\partial z_L}{\partial a_{L-1}} \cdots \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial W_1}$$



- If many of these terms are **< 1**, their product **shrinks exponentially**.
- If **activation functions** like **Sigmoid** or **Tanh** are used, their derivatives are **between 0 and 1**.
- When these small derivatives are multiplied in multiple layers, they **shrink exponentially**, leading to **vanishing gradients**.

Activation Functions Matter

Activation	Gradient Behavior
Sigmoid	Max gradient = **0.25** (vanishes for extreme inputs).
Tanh	Max gradient = **1** (still vanishes for large inputs).
ReLU	Gradient = **0** (if input < 0) or **1** (if input > 0).

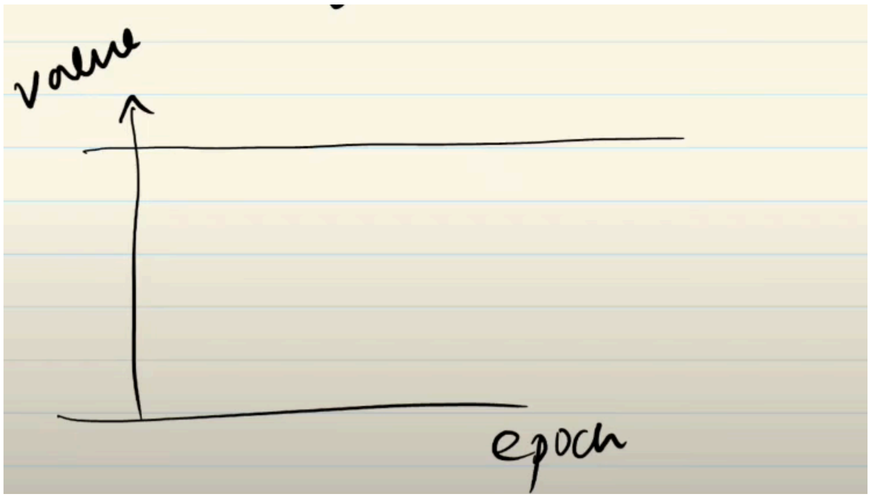
Deep Networks Suffer More

- In a 10-layer network with sigmoid:
- $TotalGradient \approx (0.25)^{10} = Nearlyzero!$

How to recognize Vanishing Gradient Problem?

1. Pay attention at the **loss**
    - If the loss does not reduce, it's an indication of vanishing gradient problem
2. Plot graphs of weight

- Plot Epoch vs Value graph
- If the graph is a straight line, it means the weight is not changing indicating vanishing gradient problem.



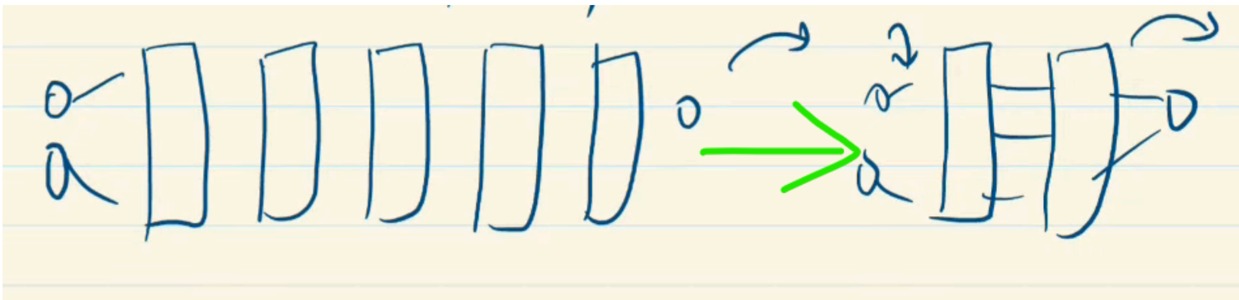
How to Solve the Vanishing Gradient Problem?

✅ 1. Use Different Activation Functions

Replace Sigmoid/Tanh with:

Activation	Why It Helps
ReLU (Rectified Linear Unit)	Avoids small gradients by keeping positive values unchanged $f(x) = max(0, x)$
Leaky ReLU	Fixes ReLU's zero-gradient issue for negative values
ELU (Exponential Linear Unit)	Further smoothens gradient flow

✅ 2. Reduce the hidden layers



- **Major con: You lose the complex patterns.**
- **Not effective for most of the times.**

✅ 3. Weight Initialization

- Use **He Initialization** (for ReLU) or **Glorot/Xavier** (for sigmoid/tanh):

  - **Xavier/Glorot Initialization** (for Sigmoid, Tanh):

  $$W \sim \mathcal{N}(0, \frac{1}{\text{fan\_in} + \text{fan\_out}})$$

  - **He Initialization** (for ReLU):

  $$W \sim \mathcal{N}(0, \frac{2}{\text{fan\_in}})$$

✅ 4. Batch Normalization

- Normalizes layer outputs to **mean=0, std=1**, keeping gradients stable.

```
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation('relu'))
```

✅ 5. Use Residual Connections (Skip Connections)

- It's a building block
- **ResNet (Residual Networks)** solve vanishing gradients by adding **shortcuts**:

$$x_{l+1} = f(W_l x_l) + x_l$$

- This helps gradients **flow directly** to earlier layers.

```
# Residual block in Keras
x = Dense(128)(inputs)
x = BatchNormalization()(x)
x = Activation('relu')(x)
outputs = Add()([x, inputs])  # Skip connection
```

Exploding Gradient Problem

- When you multiply numbers greater than 1, you get a larger numberA

10, 10, 10, 10 → 10000

- It's opposite of vanishing gradient problem
- If gradients grow too large (common in RNNs), training becomes unstable.
- **Fix**: Use **gradient clipping**.