**Unit 6  
Chain Rule in Backpropagation**

**6.1 Concept of Backpropagation and Chain Rule**

Backpropagation is the algorithm used to train multilayer neural networks. It calculates gradients of the loss function with respect to each weight by applying the **chain rule of differentiation**.

**Why Chain Rule?**

* Neural networks are compositions of functions:

Input  →  Weighted  Sum  →  Activation  →  OutputInput \; \rightarrow \; Weighted\;Sum \; \rightarrow \; Activation \; \rightarrow \; Output

* To compute how a weight affects the final loss, we must differentiate through all intermediate functions.
* The chain rule provides a systematic way to break down derivatives into smaller, manageable parts.

**6.2 Working of Backpropagation with Chain Rule**

1. **Forward Pass**
   * Inputs (x₁, x₂, x₃, x₄) flow through multiple hidden layers (H1, H2) and reach the output layer.
   * At each layer, weighted sums and activations are computed.
2. **Loss Calculation**
   * Loss is measured as the difference between actual and predicted output. For regression:

L=∑(y−y^)2L = \sum (y - ŷ)^2

1. **Gradient Calculation with Chain Rule**
   * To update weight w113w\_{11}^3 in the third layer, the gradient is computed as:

∂L∂w113=∂L∂O31⋅∂O31∂w113\frac{∂L}{∂w\_{11}^3} = \frac{∂L}{∂O\_{31}} \cdot \frac{∂O\_{31}}{∂w\_{11}^3}

* + Here, O31O\_{31} is the output of neuron f₃₁ in the last hidden layer.
  + Each term is computed step by step, moving backward through the network.

1. **Weight Update**
   * Once gradients are calculated, weights are updated using gradient descent:

wnew=wold−η∂L∂ww\_{new} = w\_{old} - η \frac{∂L}{∂w}

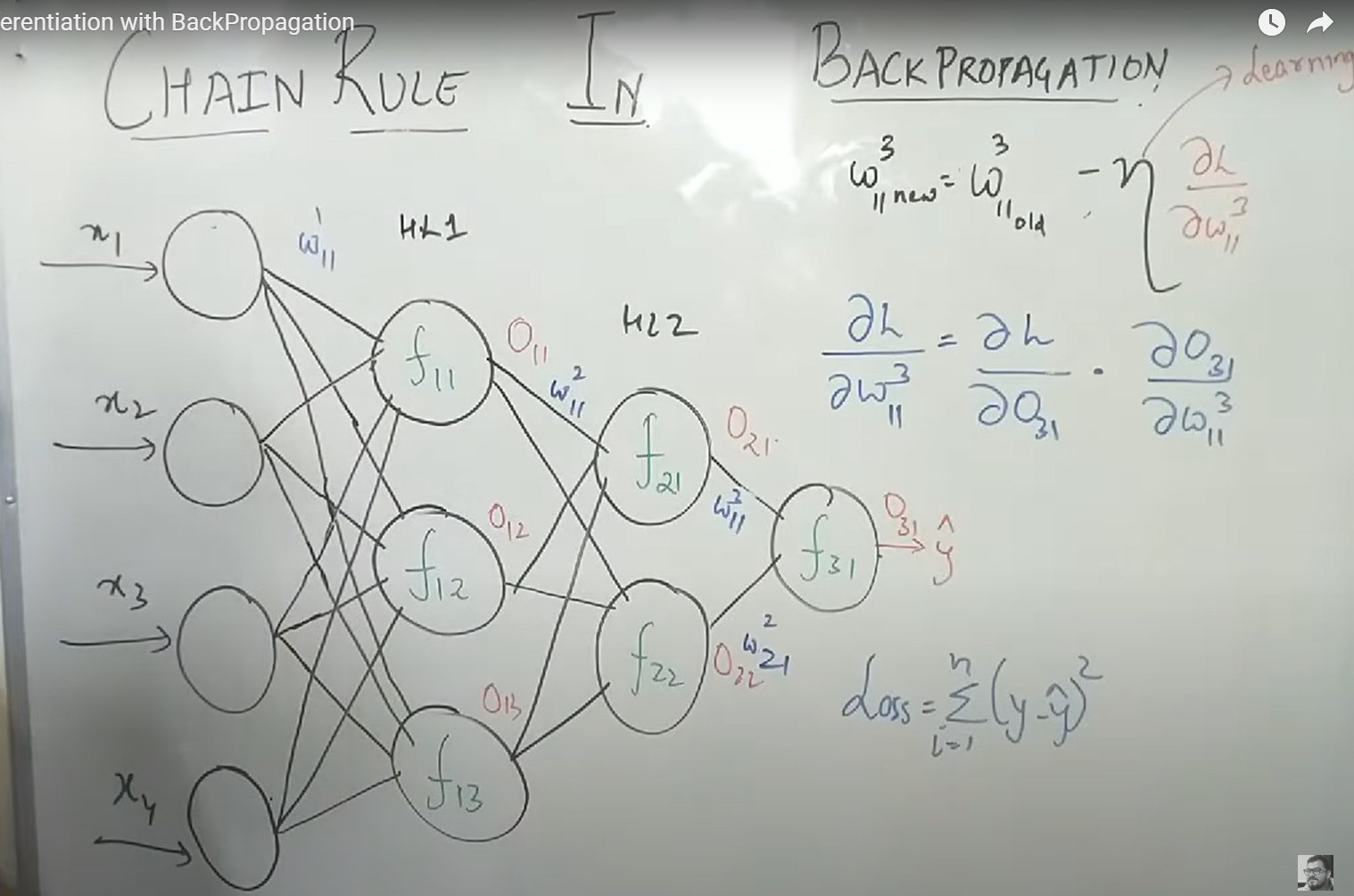
* + η (learning rate) controls the size of the adjustment.

**6.3 Importance of the Chain Rule**

* Ensures **precise gradient flow** across multiple layers.
* Makes it possible to train deep networks efficiently.
* Without the chain rule, calculating derivatives across complex neural architectures would be infeasible.

📝 **Summary**

* Backpropagation uses the **chain rule** to compute how each weight contributes to the final error.
* The chain rule decomposes gradients into stepwise parts, making multi-layer training possible.
* Final weight updates depend on both forward activations and backward error signals.
* This mechanism enables neural networks to learn by minimizing the loss function iteratively.



**Unit 7  
Vanishing Gradient Problem**

**7.1 Sigmoid Activation and Gradient Range**

* The **sigmoid activation function** outputs values between 0 and 1.
* Its derivative (gradient) lies in the range **0 ≤ σ′(z) ≤ 0.25**.
* This means the gradient values are always small (maximum of 0.25).
* When used in deep networks, repeated multiplication of such small gradients during backpropagation makes gradients shrink layer by layer.

**7.2 Effect in Deep Networks**

* As the number of layers increases, the gradient becomes smaller at each layer due to repeated multiplication.
* Eventually, the gradient approaches **zero**, making it ineffective in updating weights.
* In such cases, the weight update rule:

wnew=wold−η∂L∂ww\_{new} = w\_{old} - η \frac{∂L}{∂w}

becomes negligible, and effectively:

wnew≈woldw\_{new} \approx w\_{old}

* This slows down or completely stops learning, especially in early layers of the network.

**7.3 Solution – ReLU Activation**

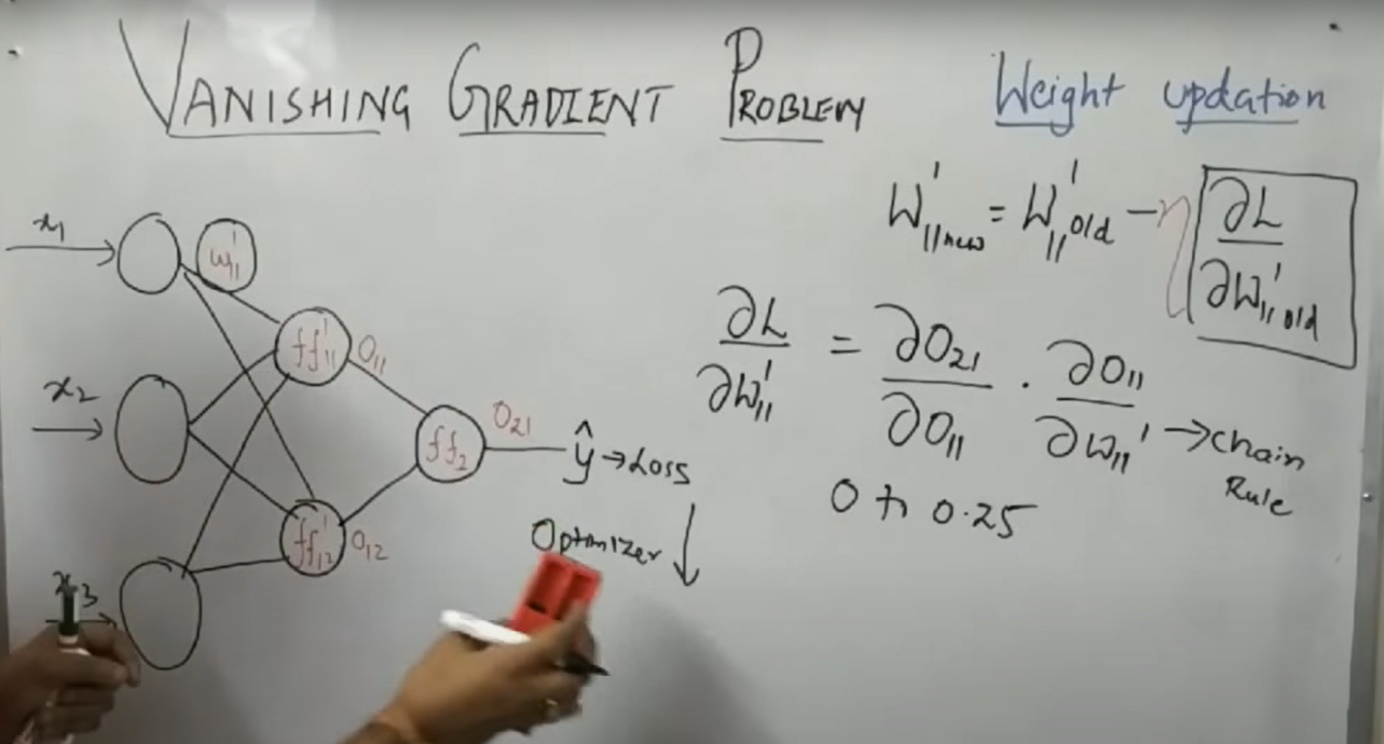
* **ReLU (Rectified Linear Unit)** is widely used to solve the vanishing gradient problem.
* ReLU outputs:

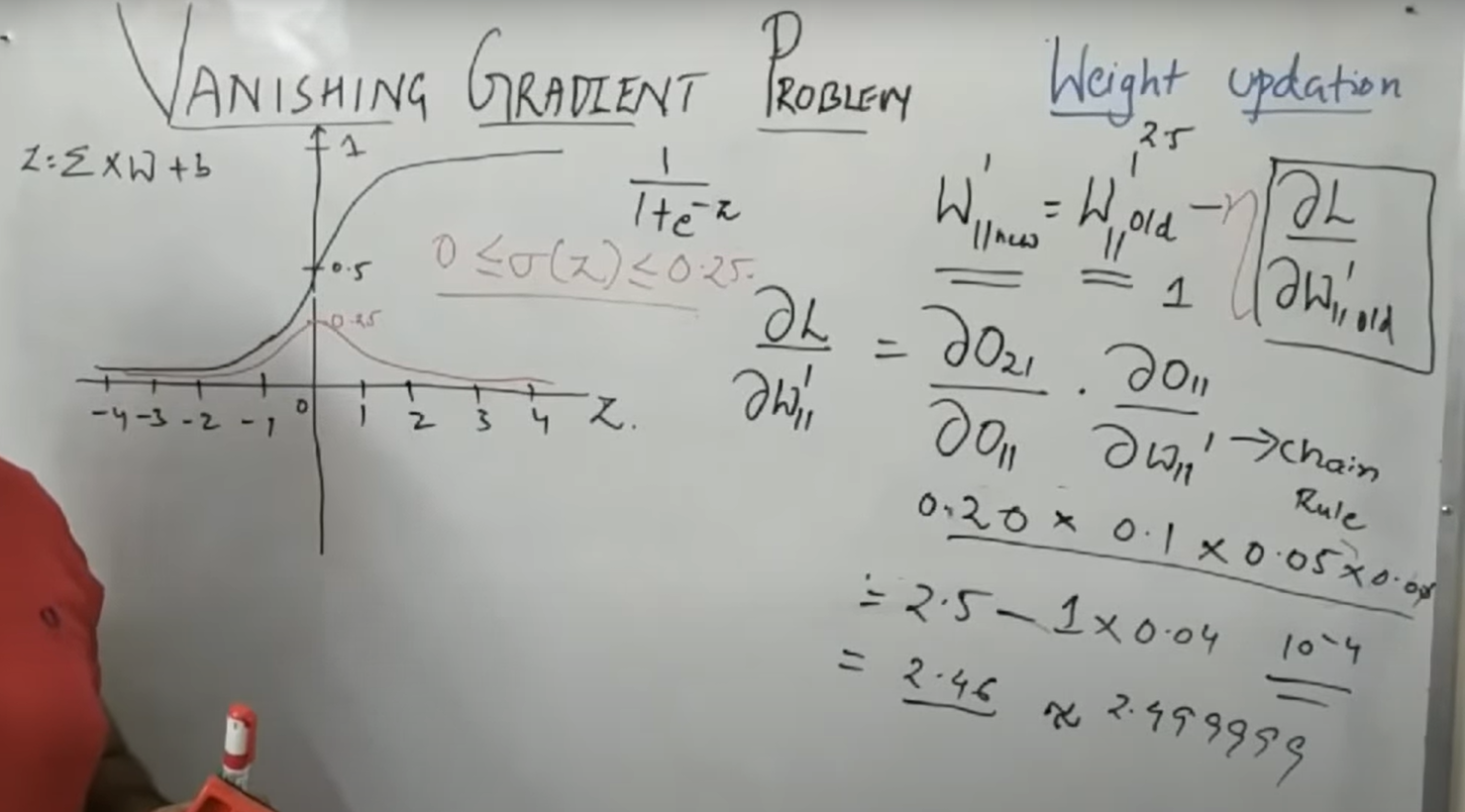
f(x)=max(0,x)f(x) = max(0, x)

* Its gradient is either 0 (for negative inputs) or 1 (for positive inputs).
* Since the gradient does not saturate between 0 and 1 like the sigmoid, ReLU avoids exponential shrinkage.
* Benefits:  
  • Faster training.  
  • Reduced vanishing gradient issues.  
  • Better performance in deep networks.

📝 **Summary**

* Sigmoid’s gradient values lie between 0 and 0.25, causing shrinking gradients.
* In deep networks, this results in almost no learning (weights stay unchanged).
* ReLU is the standard solution, as it provides stronger and non-saturating gradients.





**Unit 8  
Exploding Gradient Problem**

**8.1 When Does Exploding Gradient Occur?**

* The **exploding gradient problem** happens when weights in a neural network are **very large**.
* During backpropagation, gradients are repeatedly multiplied by these large weights.
* As a result, gradient values grow exponentially instead of shrinking.
* This leads to unstable training where parameter updates overshoot and diverge.

**8.2 Importance of Weight Initialization**

* Proper **weight initialization** is critical to prevent exploding (and vanishing) gradients.
* If weights are initialized with excessively high values, forward activations and backward gradients both become disproportionately large.
* Modern initialization strategies such as **Xavier Initialization** or **He Initialization** set weights based on the number of input and output units.
* These methods keep variance stable across layers, preventing gradients from exploding during training.

**8.3 Impact on Cost Function**

* Exploding gradients cause the **cost (loss) function** to spike to very high values.
* Instead of decreasing smoothly, the loss oscillates or diverges.
* This prevents the optimizer from converging toward the global minima.
* Training becomes unstable, and the model fails to learn meaningful patterns.

📝 **Summary**

* Exploding gradient occurs when large weights amplify gradients during backpropagation.
* Leads to very high cost values and unstable training.
* Proper weight initialization (Xavier, He) is essential to stabilize learning.
* Managing gradient magnitudes is crucial for deep network convergence.

