1. Explain the concept of forward propagation in a neural network

Forward propagation is the process by which input data is passed through a neural network to compute the output (predictions). It involves matrix multiplications, adding biases, and applying activation functions at each layer.

Steps in Forward Propagation

1. **Input Layer**: The network receives input features XXX.
2. **Hidden Layers**:
    - Each neuron computes a weighted sum of inputs: $Z=W \cdot X+bZ = W \cdot X + bZ=W \cdot X+b$
    - An activation function (e.g., ReLU, Sigmoid) is applied to introduce non-linearity: $A=\sigma(Z)A = \sigma(Z)A=\sigma(Z)$
3. **Output Layer**: The final transformed output is computed using another activation function (e.g., Softmax for classification, Linear for regression).

Forward propagation allows the network to make predictions, which are then compared with actual labels to compute loss during training.

---

2. What is the purpose of the activation function in forward propagation?

Activation functions introduce **non-linearity** into the neural network, allowing it to model complex patterns and relationships in data. Without activation functions, the network would behave like a simple linear transformation and be unable to learn intricate mappings.

Common Activation Functions & Their Purpose

- **ReLU (Rectified Linear Unit)**: Helps in deep networks by avoiding the vanishing gradient problem.
- **Sigmoid**: Converts outputs to probabilities (used in binary classification).
- **Tanh**: Similar to sigmoid but centered around zero, reducing bias issues.
- **Softmax**: Converts output values into a probability distribution for multi-class classification.

---

3. Describe the steps involved in the backward propagation (backpropagation) algorithm

Backpropagation is the process of computing gradients of the loss function with respect to the model parameters (weights and biases) to update them via gradient descent.

Steps in Backpropagation

1. **Compute Loss**: Compare predicted output with actual labels using a loss function (e.g., MSE, Cross-Entropy).
2. **Calculate Gradients (Partial Derivatives)**: Use the chain rule to compute the derivative of the loss with respect to each weight and bias.
3. **Propagate Gradients Backward**:
    - Compute gradients at the output layer.
    - Use chain rule to compute gradients at hidden layers.
    - Continue until gradients are obtained for all layers.
4. **Update Weights and Biases**:
    - Adjust parameters using an optimization algorithm like **Stochastic Gradient Descent (SGD)** or **Adam**: W=W−η∂L∂WW = W - \eta \frac{\partial L}{\partial W}W=W−η∂W∂L where η\etaη is the learning rate.

This process is repeated iteratively until the network reaches optimal performance.

---

4. What is the purpose of the chain rule in backpropagation?

The **chain rule** is used in backpropagation to compute the gradients of the loss function with respect to each weight and bias efficiently. Since neural networks consist of multiple layers, direct differentiation is not feasible.

Chain Rule Formula

If we have a function f(g(x))f(g(x))f(g(x)), the derivative is:

dfdx=dfdg⋅dgdx\frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}dxdf=dgdf⋅dxdg

In backpropagation, this allows us to compute gradients layer by layer:

∂L∂W=∂L∂A⋅∂A∂Z⋅∂Z∂W\frac{\partial L}{\partial W} = \frac{\partial L}{\partial A} \cdot \frac{\partial A}{\partial Z} \cdot \frac{\partial Z}{\partial W}∂W∂L=∂A∂L⋅∂Z∂A⋅∂W∂Z

where:

- AAA is the activation,
- ZZZ is the pre-activation output,
- WWW is the weight.

The chain rule ensures efficient computation by reusing previously computed gradients instead of recalculating them for every layer.