Day 61 - DIY

Q1. What is Backpropagation?

Backpropagation is a ***supervised learning algorithm*** used for training neural networks. It minimizes the error between predicted and actual outputs by adjusting the weights and biases of the network. Backpropagation uses the **chain rule** of derivatives to compute the gradient of the error with respect to each weight, moving backward from the output layer to the input layer.

Key points:

- Backpropagation calculates how much each weight contributes to the error.
- It is essential for optimizing weights using **gradient descent**.

Q2. How does a Neural Network Learn?

A neural network learns by adjusting its weights and biases to minimize the error in predictions. This process can be broken into three steps:

1. **Forward Propagation**: Compute the outputs based on current weights and biases.
2. **Error Calculation**: Compare the predicted outputs to the target values and calculate the error (e.g., Mean Squared Error).
3. **Backpropagation**: Use the gradients of the error (calculated using the chain rule) to update weights and biases iteratively.

Learning is driven by minimizing the error through ***multiple iterations of forward propagation, error calculation, and backpropagation***.

Q3. What is Weight and Bias in an Artificial Neural Network?

- **Weights**: Parameters that determine the strength and direction of the influence between neurons. They scale the input values, allowing the network to learn which features are important for prediction.
- **Bias**: A value added to the weighted sum of inputs before passing through the activation function. Bias allows the activation function to shift, helping the network fit more complex patterns.

Example: For a neuron, the input-output relationship is:

$$\text{output}=\text{activation}(W \cdot \text{input}+b)$$

Where W is the weight, b is the bias, and activation is a non-linear function (e.g., sigmoid or ReLU).

Q4. How Does Weight Work?

Weights control the contribution of each input to a neuron. During training, weights are updated based on how much they affect the error. If a weight significantly contributes to the error, it is adjusted more.

Example:

1. Inputs $(i1,i2)$ are multiplied by weights $(W1,W2)$.
2. Weighted sum: $weighted\_sum=W1\cdot i1+W2\cdot i2$.
3. Activation: Apply a non-linear function to the weighted sum.

Weights allow the neural network to learn the relationships between inputs and outputs.

Q5. What Will Happen If the Learning Rate Is Set Too Low or Too High?

- **If Learning Rate is Too Low:**
  - The network updates weights very slowly.
  - Convergence to the optimal solution takes a long time, or the network may get stuck in local minima.
  - Training becomes inefficient.
- **If Learning Rate is Too High:**
  - The network may overshoot the optimal solution.
  - It can lead to oscillations or divergence, failing to minimize the error.
  - The model may never learn or become unstable.

Ideal Learning Rate: A balanced learning rate ensures steady convergence without overshooting. Often, learning rates are tuned experimentally or using techniques like learning rate scheduling or adaptive optimizers (e.g., Adam, RMSProp).