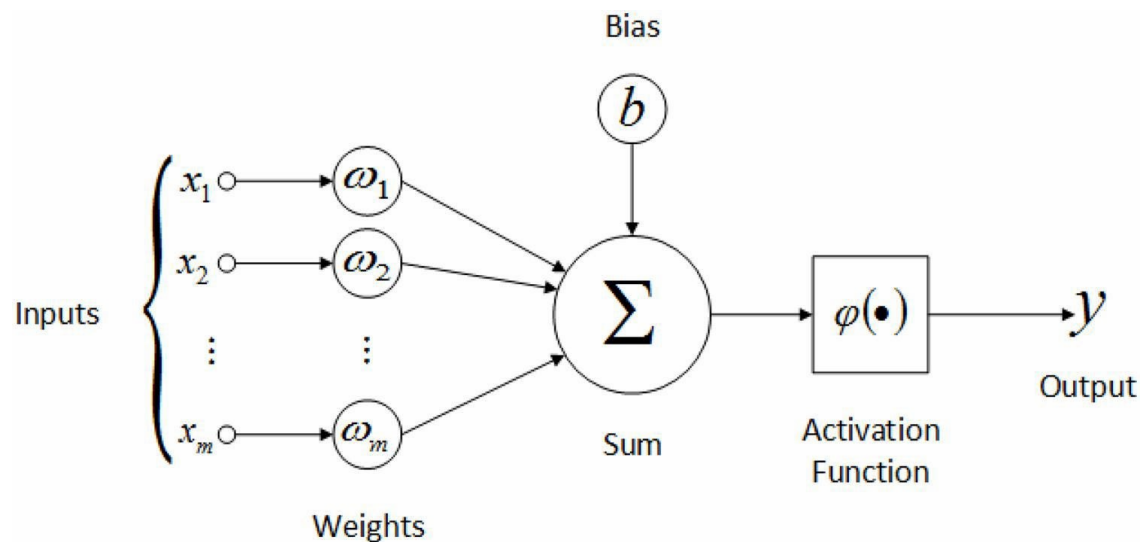


# Forward and Backward Propagation in Multilayered Neural Networks: A Deep Dive



Forward propagation is a fundamental process in neural networks, where inputs are passed through the network to produce an output. Understanding this mechanism is crucial for anyone looking to delve into the world of deep learning and artificial intelligence.

## What is Forward Propagation?

Forward propagation, often simply referred to as “forward pass,” is the process of passing input data through a neural network’s layers to produce an output. In a multilayered neural network, also known as a deep neural network, this involves several intermediate layers between the input and output layers.

## Components of a Multilayered Neural Network

Before diving into the mechanics of forward propagation, let’s review the key components of a multilayered neural network:

1. **Input Layer:** This layer receives the raw data, such as pixel values from an image or features from a dataset.
2. **Hidden Layers:** These intermediate layers process the input data. Each hidden layer consists of neurons (or nodes) that apply transformations to the data.
3. **Output Layer:** This final layer produces the network’s output, which could be a classification, regression value, or any other desired result.

## The Forward Propagation Process

The forward propagation process can be broken down into several steps:

1. **Input Data:** The process begins with the input layer receiving the raw data. Each feature of the input data corresponds to a neuron in the input layer.
2. **Weighted Sum:** For each neuron in a hidden layer, the input data is multiplied by a set of weights and added together. A bias term is also included to adjust the weighted sum. Mathematically, this can be represented as:

$$z = \sum (w_i \cdot x_i) + b$$

where  $w_i$  are the weights,  $x_i$  are the input values, and  $b$  is the bias.

3. **Activation Function:** The weighted sum  $z$  is passed through an activation function to introduce non-linearity into the model. Common activation functions include ReLU (Rectified Linear Unit), Sigmoid, and Tanh. For example, the ReLU activation function is defined as:

$$\text{ReLU}(z) = \max(0, z)$$

4. **Propagation Through Layers:** The output of the activation function becomes the input for the next layer. This process repeats for each hidden layer in the network.

5. **Output Layer:** Finally, the processed data reaches the output layer, where another set of weights and biases is applied, followed by

an activation function if necessary. The resulting values are the network's predictions.

## Example of Forward Propagation

Consider a simple neural network with one hidden layer:

1. **Input Layer:** Let's assume we have two input features  $x_1$  and  $x_2$ .
2. **Hidden Layer:** This layer has two neurons. Each neuron will perform the weighted sum and apply an activation function:

$$z_1 = w_{11} \cdot x_1 + w_{12} \cdot x_2 + b_1$$

$$a_1 = \text{ReLU}(z_1)$$

$$z_2 = w_{21} \cdot x_1 + w_{22} \cdot x_2 + b_2$$

$$a_2 = \text{ReLU}(z_2)$$

3. **Output Layer:** This layer has one neuron producing the final output:

$$z_{out} = w_{31} \cdot a_1 + w_{32} \cdot a_2 + b_{out}$$

$$\text{output} = \text{Sigmoid}(z_{out})$$

## Significance of Forward Propagation

Forward propagation is critical for several reasons:

- **Prediction:** It is the mechanism by which neural networks make predictions. By processing input data through multiple layers, the network can learn complex patterns and relationships.
- **Training:** Forward propagation is used during the training phase to calculate the network's output, which is then compared to the actual target to compute the loss. This loss is used to update the weights and biases through backward propagation.
- **Efficiency:** Efficient forward propagation ensures that the network can handle large datasets and complex tasks without excessive computational costs.

## Challenges and Optimization

While forward propagation is straightforward in concept, optimizing it for large and complex networks can be challenging. Techniques such as vectorization (using matrix operations), efficient weight initialization,

and batch normalization can significantly improve the efficiency and performance of forward propagation.

Backward propagation, or backpropagation, is a core concept in the training of neural networks. It's the process that allows these networks to learn from data by adjusting their weights and biases to minimize the prediction error.

## What is Backpropagation?

Backpropagation is a supervised learning algorithm used for training neural networks. It calculates the gradient of the loss function with respect to each weight by the chain rule, iteratively updating the weights to minimize the error. This process allows the network to improve its performance over time.

## Components of Backpropagation

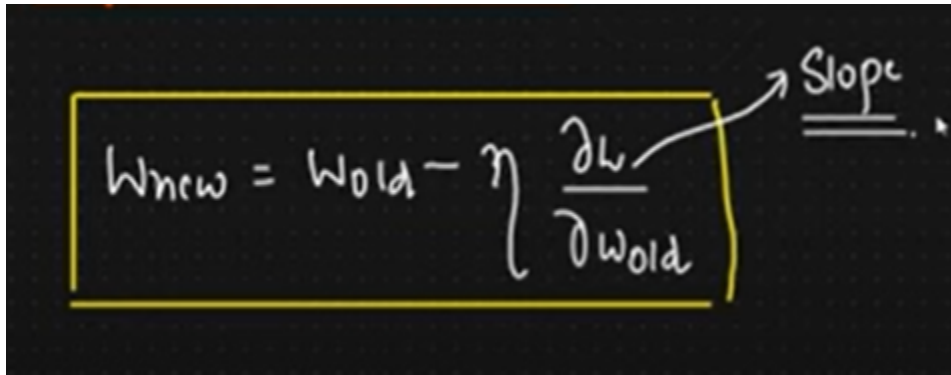
Before diving into the steps of backpropagation, it's essential to understand the key components involved:

1. **Loss Function:** A function that measures the difference between the network's prediction and the actual target value. Common loss functions include Mean Squared Error (MSE) for regression tasks and Cross-Entropy Loss for classification tasks.
2. **Weights and Biases:** Parameters of the network that are adjusted during training to minimize the loss.

3. **Learning Rate:** A hyperparameter that determines the size of the steps taken to update the weights. It controls how quickly or slowly the model learns.

### Weight Updation Formula

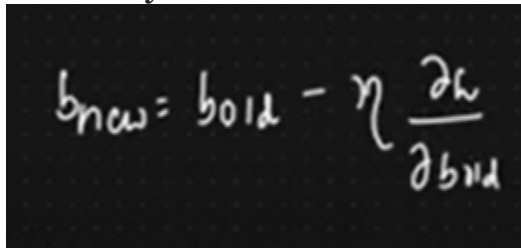
Backward propogation uses this Weight Updation Formula to find the best global minima from the gradient descent which is our final weight value.


$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$$

The handwritten formula is enclosed in a yellow rectangular box. An arrow points from the fraction  $\frac{\partial L}{\partial W_{old}}$  to the word "Slope" which is underlined.

Here lambda is the learning rate

Similarly the formula for Bias Updation becomes


$$b_{new} = b_{old} - \eta \frac{\partial L}{\partial b_{old}}$$

### Significance of Backpropagation

Backpropagation is critical for several reasons:

- **Learning:** It enables neural networks to learn from data by iteratively adjusting weights and biases to minimize the error.
- **Efficiency:** Backpropagation leverages the chain rule to efficiently compute gradients, making it feasible to train deep networks with many layers.
- **Generalization:** By minimizing the loss, backpropagation helps the network generalize well to unseen data, improving its predictive performance.

## Challenges and Optimization

While backpropagation is powerful, it can face challenges such as vanishing gradients, where gradients become very small, slowing down learning. Techniques like gradient clipping, batch normalization, and advanced optimization algorithms (e.g., Adam, RMSprop) can help mitigate these issues and improve training efficiency.

## Chain Rule of Derivatives in Backpropagation

Backpropagation is the key algorithm for training neural networks, and at its heart lies the chain rule of derivatives. This fundamental concept from calculus enables the efficient computation of gradients, allowing neural networks to learn from data.

### What is the Chain Rule?

The chain rule is a formula for computing the derivative of the composition of two or more functions. If we have two functions  $f$  and  $g$ ,



and we define a new function  $h$  as the composition of  $f$  and  $g$  (i.e.,  $h(x)=f(g(x))$ ), the chain rule states that the derivative of  $h$  with respect to  $x$  is given by:

$$\frac{dh}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}$$

## The Chain Rule in Backpropagation

In the context of neural networks, the chain rule is used to compute the gradients of the loss function with respect to each weight and bias in the network. This process is essential for updating the parameters to minimize the loss during training. Here's how it works:

1. **Forward Pass:** Compute the output of the network by passing the input data through each layer.
2. **Loss Calculation:** Calculate the loss by comparing the network's output to the actual target values.
3. **Backward Pass (Backpropagation):** Compute the gradients of the loss with respect to each parameter using the chain rule.

## Applying the Chain Rule in Backpropagation

Consider a simple neural network with one hidden layer:

### 1. Forward Pass:

- Input layer:  $x$
- Hidden layer:  $z^1 = W^1x + b^1, a^1 = \sigma(z^1)$
- Output layer:  $z^2 = W^2a^1 + b^2, a^2 = \sigma(z^2)$

### 2. Loss Calculation:

- Loss:  $L = \text{Loss}(a^2, y)$

To minimize the loss, we need to compute the gradients of  $L$  with respect to  $W^1$ ,  $b^1$ ,  $W^2$ , and  $b^2$ .

#### Gradient of the Loss with Respect to the Output Layer

Using the chain rule, we can find the gradient of the loss with respect to the weights and biases in the output layer:

$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial a^2} \cdot \frac{\partial a^2}{\partial z^2} \cdot \frac{\partial z^2}{\partial W^2}$$

## Importance of the Chain Rule in Backpropagation

The chain rule is essential for backpropagation because it allows for the efficient computation of gradients in a multilayered neural network.

Without the chain rule, calculating these gradients would be computationally infeasible, especially for deep networks with many layers. By systematically applying the chain rule, we can ensure that each parameter in the network is adjusted in the direction that minimizes the loss, enabling the network to learn effectively from data.