**A neural network** is a machine learning algorithm, or model, that makes decisions in a manner similar to the human brain, by using processes that mimic the way biological nurons work together to identify phenomena, weigh options and make decision.

**An artificial neuron** (perceptron) or neural node is a mathematical model. In most cases, it computes the weighted average of its input and then applies a bias to it. Post that, it passes this resultant term through an activation function.**This activation function is a nonlinear function such as the sigmoid function that accepts a linear input and gives a nonlinear output.**

A typical neural network consists of layers of neurons called neural nodes. These layers are of the following three types:

1. input layer (single)
2. hidden layer (one or more than one)
3. output layer (single)

Each neural node is connected to another and is characterized by its weight and a threshold. It gets an input on which it does some transformation and post that, it sends an output. If the output of any individual node is above the specified threshold value, that node gets activated. Then, it sends data to the next layer of the network. Otherwise, it remains dormant and thus doesn’t transmit any data to the next layer of the network.

**Activation Function**: An activation function is a mathematical function applied to the output of a neuron. It introduces non-linearity into the model, allowing the network to learn and represent complex patterns in the data. Without this non-linearity feature, a neural network would behave like a linear regression model, no matter how many layers it has.

Choosing the right activation function is crucial for training neural networks that generalize well and provide accurate predictions.

**Backpropagation:**

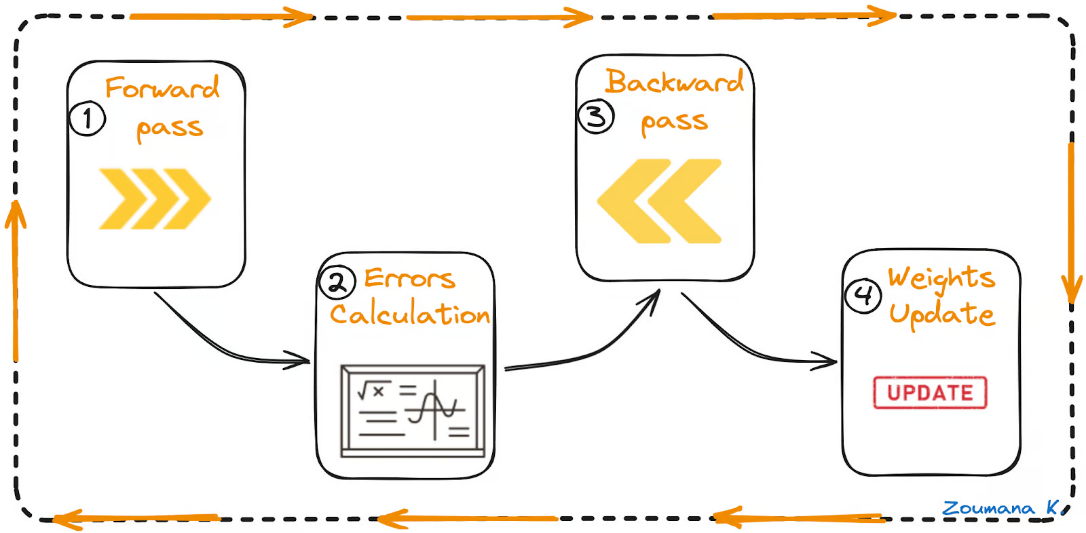
Backpropagation is a machine learning technique essential to the optimization of artificial neural networks. It facilitates the use of gradient descent algorithms to update network weights, which is how the deep learning models driving modern artificial intelligence (AI) “learn.”

backpropagation is an elegant method to calculate how changes to any of the weights or biases of a neural network will affect the accuracy of model predictions. It’s essential to the use of supervised learning, semi-supervised learning or self-supervised learning to train neural networks.

How Does backpropagation work:

Backpropagation algorithm comprises four main step which are

* Forward pass
* Errors calculation
* Backward pass
* Weights update



Explanation: Considering we have two input value x1 and x2

Forward pass

This is the first step of the backpropagation process, and it’s illustrated below:

* The data (inputs X1 and X2) is fed to the input layer
* Then, each input is multiplied by its corresponding weight, and the results are passed to the neurons N1X and N2X of the hidden layers.
* Those neurons apply an activation function to the weighted inputs they receive, and the result passes to the next layer.

Errors calculation

* The process continues until the output layer generates the final output (o/p).
* The output of the network is then compared to the ground truth (desired output), and the difference is calculated, resulting in an error value.

Backward pass

This is an actual backpropagation step, and cannot be performed without the above forward and error calculation steps. Here is how it works:

* The error value obtained previously is used to calculate the gradient of the loss function.
* The gradient of the error is propagated back through the network, starting from the output layer to the hidden layers.
* As the error gradient propagates back, the weights (represented by the lines connecting the nodes) are updated according to their contribution to the error. This involves taking the derivative of the error with respect to each weight, which indicates how much a change in the weight would change the error.
* The learning rate determines the size of the weight updates. A smaller learning rate means than the weights are updated by a smaller amount, and vice-versa.

Weights update

* The weights are updated in the opposite direction of the gradient, leading to the name “gradient descent.” It aims to reduce the error in the next forward pass.
* This process of forward pass, error calculation, backward pass, and weights update continues for multiple epochs until the network performance reaches a satisfactory level or stops improving significantly.

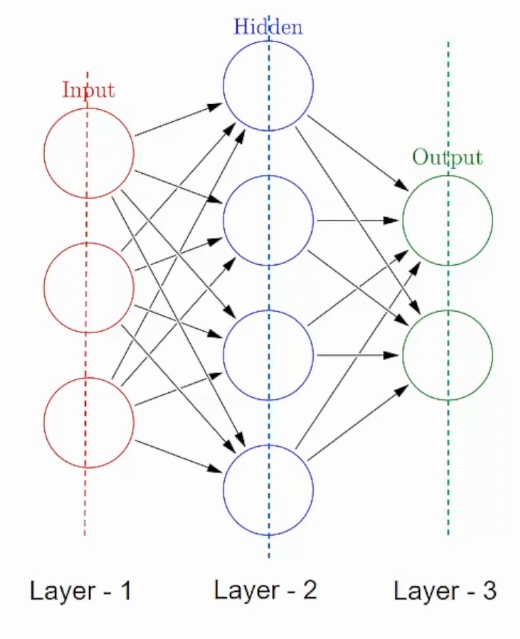
**Neural Network Layer**

A neural network is made up of vertically stacked components called Layers. Each dotted line in the image represents a layer. There are three types of layers in a Neural Network

Input layer

Hidden layer

Output layer



**Input Layer**– First is the input layer. This layer will accept the data and pass it to the rest of the network.

**Hidden Layer**– The second type of layer is called the hidden layer. Hidden layers are either one or more in number for a neural network. In the above case, the number is 1. Hidden layers are the ones that are actually responsible for the excellent performance and complexity of neural networks. They perform multiple functions at the same time such as data transformation, automatic feature creation, etc.

**Output layer**– The last type of layer is the output layer. The output layer holds the result or the output of the problem. Raw images get passed to the input layer and we receive output in the output layer.

**Weights**

Weightscontrol the signal (or the strength of the connection) between two neurons. In other words, a weight decides how much influence the input will have on the output.

**Biases**:

Biases, which are constant, are an additional input into the next layer that will always have the value of 1. Bias units are not influenced by the previous layer (they do not have any incoming connections) but they do have outgoing connections with their own weights. The bias unit guarantees that even when all the inputs are zeros there will still be an activation in the neuron, i.e., the neuron can still produce a non-zero output.

Weights and Biases effect in neural network

Higher Weights shows Stronger influence of input on output.

Lower Weights shows Weaker influence of input.

Bias Allows flexibility in fitting the model even if inputs are minimal or zero.

**Overfitting in Neural Network:**

Overfitting occur when the model tries to learn too many details in the training data along with the noise from the training data. As a result, the model performance is very poor on unseen or test datasets. Therefore, the network fails to generalize the features or patterns present in the training dataset.

Overfitting during training can be spotted when the error on training data decreases to a very small value but the error on the new data or test data increases to a large value.

Ways to mitigate Overfitting

Add more data to the training set if available

Reducing Complexity of the model by removing some layers from the model, or reducing the number of neurons in the layers.

Introduce Drop out.

**ReLU Activation function**

The **Rectified Linear Unit (ReLU)** is one of the most popular activation functions used in neural networks, especially in deep learning models. It has become the default choice in many architectures due to its simplicity and efficiency. The ReLU function is a piecewise linear function that outputs the input directly if it is positive; otherwise, it outputs zero.

Mathematical Formular of ReLU activation

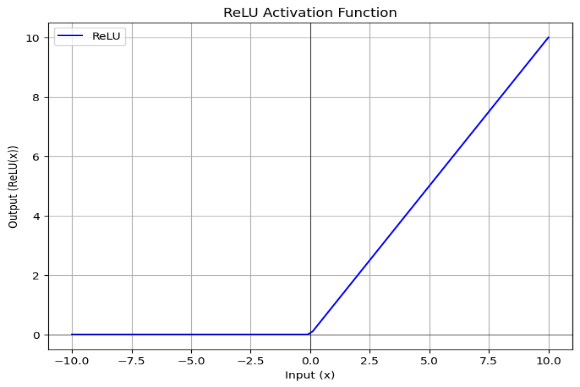
f(x)=max (0, x)

Where:

* f (x)=x if x>0
* f (x)=0 if x≤0

Graphical Representation

On a graph, ReLU appears as a straight line for x>0 (slope = 1) and as a flat line along the x-axis for x≤0



Where and why, it is used

ReLU is the preferred choice for hidden layers in most deep learning models due to its computational efficiency and effectiveness. However, it might not always be the best choice for every problem. In cases where your model suffers from the "dying ReLU" problem or unstable gradients, trying alternative functions like Leaky ReLU, PReLU, or ELU could yield better results.

ReLU is widely used in convolutional neural networks (CNNs), recurrent neural networks (RNNs), and feedforward neural networks.

Advantages of ReLU

* Faster Convergence: ReLU accelerates the training process compared to sigmoid or tanh because it mitigates the vanishing gradient problem (gradients do not shrink to zero for positive inputs).
* Simplicity: ReLU is simple to implement and computationally inexpensive.
* Effective in Deep Networks: Deep networks with ReLU often converge faster and perform better than those using other activation functions.

Disadvantages of ReLU

1. Dying ReLU Problem

* When the input to a neuron is consistently negative, the gradient is zero, and the neuron stops updating. This leads to “dead neurons” that no longer contribute to learning.
* In extreme cases, a large portion of the network may become inactive.

1. Unbounded Output

* ReLU can output arbitrarily large values for high inputs, which can lead to instability during training. This is sometimes mitigated with weight regularization or batch normalization.

Real World Application of Relu

The following are real world application of Relu

Computer Vission

Natuaral Language Processing

Speech Recognition

Recommendation System

Autonomous Systems and Robotics

**Generative Models (GANs and VAEs)**