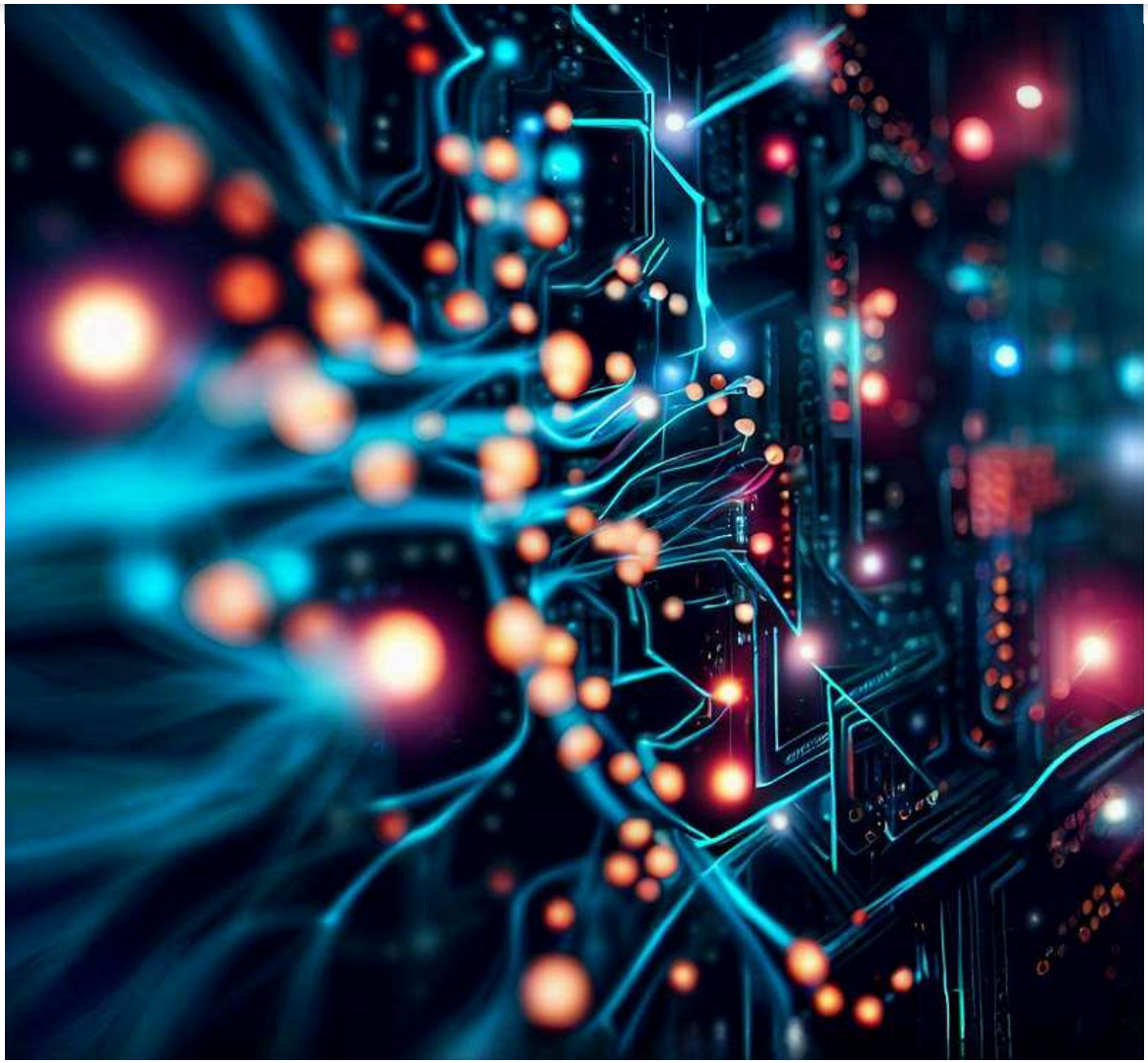
Machine Learning in Plain English

Summary

The provided web content describes the process of training a neural

Deep Learning Course — Lesson 5: Forward and Backward Propagation



Forward Propagation

the input data is fed through the network to generate a prediction.

Input Data

In the case of a feedforward neural network, the data flows in one direction — from input to output. The input layer consists of the data that you feed into the network. This could be any type of data suitable for your problem, like images, text, or numerical data. The data is often vectorized or turned into arrays that the network can process.

Weights and Bias

Weights and biases are the learnable parameters of a neural network. The weights determine the strength of the influence of a given input neuron on the output, while biases allow the output to be shifted by a constant value.

- **Weights:** Each connection between neurons is assigned a weight, which is a measure of its relative importance. When an input is passed through a neuron, it gets multiplied by this weight. Weights can be both positive (excitatory) or negative (inhibitory), and they are initialized randomly at the start of training.
- **Bias:** Bias is an extra input to neurons, and it is always 1, and has its own connection weight. This bias weight gives the ability to shift the activation function to the left or right, which may be critical for successful learning.

Neurons and Layers

organized into multiple layers — an input layer, hidden layers, and an output layer.

- **Input Layer:** This is where the network receives input from your dataset. It's often transformed into a suitable form for the network.
- **Hidden Layer(s):** These are layers in between the input and output layers. They perform transformations on the input data with the goal of learning features that are useful for the task at hand. The term "deep" in deep learning refers to having multiple hidden layers in the network.
- **Output Layer:** This is the final layer. It provides the predictions or classifications that the network has been trained to produce.

Linear Transformation

The first step in forward propagation involves calculating a weighted sum of the inputs and the associated weights, and then adding the bias. This is also known as a linear transformation. The equation is usually as follows: `z = weights * inputs + bias`.

Activation Functions

A neural network is composed of layers of nodes (also known as neurons). Each node in a layer receives input from multiple nodes from the previous layer. Each of these inputs is multiplied by a weight, which is a value that the network learns during training. So, for each node, we sum up the product of

bias term which is analogous to the intercept in linear regression models. This sum is also known as the “weighted sum”. We can write it as:

Weighted Sum = (input_1 * weight1) + (input2 * weight2) + … + (inputN * weightN) + bias

This operation is a linear operation, meaning that the output is directly proportional to the input. If we only used this in our network, our model would only be able to learn **linear relationships** between the inputs and the output.

Here’s where the activation function comes in. After the weighted sum is computed, it’s passed through an activation function before it’s sent as output or as input to the next layer. They are used to introduce **non-linearity** to the neural network, allowing it to learn from complex, non-linear relationships in the data.

Without an activation function, a neural network would just be a simple linear regression model, which is limited in its complexity. With an activation function, we’re able to model and learn from the complexities and nuances in our data, thereby making better predictions or classifications.

Without non-linearity, no matter how many layers the network has, it would behave just like a single layer network because summing these layers would give another linear function. Non-linear activation functions allow the

accordingly, so as to improve the model during training.

In other words, the activation function's job is to decide how much signal to pass onto the next layer (i.e., how "active" it is), based on the summed weighted input it receives. This can be viewed as the node making a decision based on its input: if the input is relevant for the model's prediction, it gets activated and sends the signal forward. If it's not relevant, it doesn't get activated.

The activation function in a neural network serves as a gatekeeper. Each neuron in the network receives an input, processes it, and decides whether it should be activated and pass the information on to the next layer. The neuron does this based on the output of the activation function.

Consider a simple scenario where we're using the ReLU (Rectified Linear Unit) activation function. This function simply outputs the input directly if it's positive; otherwise, it outputs zero. In this context, you can think of the neuron as making a decision: if the weighted sum of the inputs it receives (plus a bias term) is positive, it considers this input significant enough and passes the value forward. If the weighted sum is not positive (i.e., it's zero or negative), it's deemed insignificant, and the neuron outputs zero — effectively blocking the signal from going forward.

To understand why this is important, consider the problem we're trying to solve. Neural networks learn to map complex input patterns (like the pixels in an image) to outputs (like image labels). For this mapping to be accurate, the

predictions and which are not.

That's where activation functions come in. By deciding whether and how much of a signal to pass forward based on the input it receives, each neuron can learn to recognize specific patterns in the input data and propagate that information forward when it sees them. This is a crucial part of the network's ability to learn from and make accurate predictions about the data.

Commonly Used Activation Functions

Let's look at some of the most common types of activation functions used in deep learning:

1. Sigmoid Function

The sigmoid function is an activation function that is shaped like an "S." It is a smooth, continuously differentiable function that restricts the output to a range between 0 and 1. This makes it useful for models where we need to predict the probability as an output.

However, it's not used as much in practice due to two major drawbacks. First, the sigmoid function saturates and kills gradients. Second, the sigmoid function is not zero-centered.

function "saturates" when its input is a large positive or negative number. In these cases, the function's output gets very close to 1 or 0 respectively. The issue with this is when it comes to backpropagation. During backpropagation, we calculate gradients or derivatives of the function. The gradient of the sigmoid function near 0 and 1 is almost 0. This means that during backpropagation, the weight updates would be very small, effectively making it seem like learning has stopped. This phenomenon is called "vanishing gradients". This is a problem because it slows down learning or makes it stop altogether early in the process.

Sigmoid function is not zero-centered: Ideally, we would like our activation functions to be zero-centered, meaning the output of the activation function should be able to take on both positive and negative values evenly around 0. This helps the gradients to also be zero-centered and helps the optimization algorithm (like gradient descent) converge faster. However, the sigmoid function always outputs a positive value (between 0 and 1). This means that during backpropagation, all the neurons in the network will either all go up or all go down together, instead of having the ability to adjust in a more balanced way. This makes optimization harder and can slow down training.

2. Hyperbolic Tangent (tanh) Function

The tanh function is like the sigmoid function, but it scales the output to range between -1 and 1. This zero-centered nature makes it preferred over the sigmoid function.

making it unsuitable for deep networks.

3. Rectified Linear Unit (ReLU) Function

ReLU is the most commonly used activation function in the field of deep learning. It takes the input and thresholds it at zero (replaces negative values with zero). The function returns 0 if it receives any negative input, but for any positive value x, it returns that value back.

ReLU has become very popular because it has been found to greatly accelerate the convergence of stochastic gradient descent compared to sigmoid/tanh functions. It's also cheaper to compute. However, ReLU units can be fragile during training and can "die".

4. Softmax Function

The softmax function is often used in the output layer of a classifier model where we need to represent the outputs as probabilities. It squashes a K-dimensional vector of arbitrary real values into a K-dimensional vector of real values in the range (0, 1) that add up to 1.

> Each of these activation functions has its advantages and disadvantages, and the choice of which to use depends on the specific requirements of the model and the nature of the data it will learn from.

neural network using a method called backpropagation in combination with a process called gradient descent. Let me explain this further:

Initialization of Weights and Biases

Before training a neural network, we need to initialize the weights and biases for each neuron. Initially, these values are set randomly. This is crucial as it breaks symmetry between different neurons and allows them to learn different features during training. If all the weights were initialized to the same value, all neurons in a layer would learn the same features which is not what we want.

Training: Backpropagation and Gradient Descent

The process of training a neural network involves passing our input data through the network (this is called forward propagation), then updating our weights and biases based on the error of our network's output (this is called backpropagation).

- **Forward Propagation:** We pass the input data through the network, and each neuron in the hidden layers calculates the weighted sum of its inputs and applies an activation function to it. This output is then passed onto the next layer and this process repeats till we reach the output layer and get our final output (predicted value).

training data). The difference between the expected and the actual output gives us the error that our network made in its prediction.

- **Backpropagation:** Now, this error is passed backward through the network. This is where the "learning" happens. Using this error, we calculate the gradients of the error with respect to our weights and biases (basically, we figure out how much changing each weight and bias would change the error). Backpropagation is a method used to calculate the gradient of the loss function with respect to the weights in the network.
- **Gradient Descent:** The cost function gives a measure of how far off our predictions are from the actual values. To improve our model, we need to minimize this cost function, and we do this using an optimization algorithm known as gradient descent. Gradient descent iteratively adjusts the parameters, nudging them in the direction that reduces the cost function until it finds the parameters with the lowest possible cost.
- This cycle (forward propagation, error calculation, backpropagation, and weights & biases update) is repeated for many **iterations (or epochs)** during the training process, gradually improving the accuracy of the network's predictions.

Let's test your understanding

- What is forward propagation in the context of neural networks?

- Why do we initialize weights randomly in a neural network?
- What is the vanishing gradient problem?
- What is the primary difference between the sigmoid function and the tanh function?
- What does the bias term in a neuron do?
- What is the issue with the Sigmoid function as an activation function in neural networks?
- Why is the ReLU function commonly used as an activation function in deep learning?
- How does a neuron decide whether to pass the signal to the next layer?
- Why do we need non-linearity in a neural network?

Deep Learning | Forward Propagation | Backpropagation | Activation Functions | Gradient Descent

---

Recommended from ReadMedium

Jorgecardete

14 min read

Vyacheslav Efimov

Understanding Deep Learning Optimizers: Momentum, AdaGrad, RMSProp & Adam

Gain intuition behind acceleration training techniques in neural networks

8 min read

LM

LM Po

Understanding LLM Decoding Strategies

The emergence of ChatGPT at the end of 2022 revolutionized the world with its ability to generate human-like text responses. At its core...

11 min read

Mikel

Pinterest ML Internship Summer 2025

Looking for a tech internship for the Summer 2025. I share my recent Interview experience, Questions, Solutions and tips.

6 min read

J

Jo Wang

Neural Network Structure

4 min read

Abdur Rahman

Python is No More The King of Data Science

5 Reasons Why Python is Losing Its Crown

7 min read