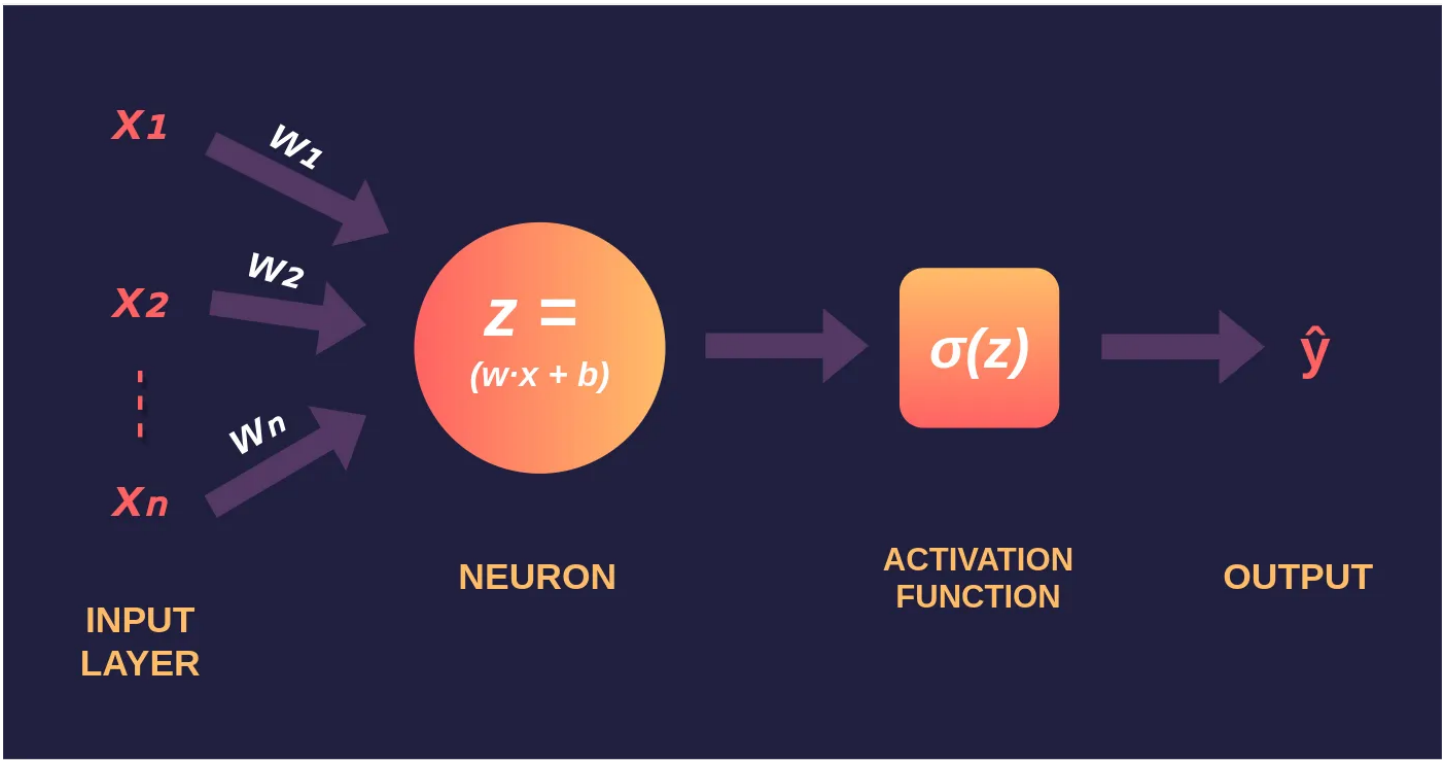

Created by Author using Diagrams.Net

A Gentle Introduction To Math Behind Neural Networks

Let's dive into the Mathematics behind Neural Networks and Deep Learning

Dasaradh S K · Follow
Published in Towards Data Science
6 min read · Oct 7, 2020

Today, with open source machine learning software libraries such as TensorFlow, Keras, or PyTorch we can create a neural network, even with high structural complexity, with just a few lines of code. Having said that, the mathematics behind neural networks is still a mystery to some of us, and having the mathematics knowledge behind neural networks and deep learning can help us understand what's happening inside a neural network. It is

also helpful in architecture selection, fine-tuning of deep learning models, hyperparameters tuning, and optimization.

Introduction

I had ignored understanding the mathematics behind neural networks and deep learning for a long time as I didn't have good knowledge of algebra or differential calculus. A few days ago, I decided to start from scratch and derive the methodology and mathematics behind neural networks and deep learning, to know how and why they work. I also decided to write this article, which would be useful to people like me, who find it difficult to understand these concepts.

Perceptrons

Perceptrons — invented by Frank Rosenblatt in 1958, are the simplest neural network that consists of *n* number of inputs, only one neuron, and one output, where *n* is the number of features of our dataset. The process of passing the data through the neural network is known as forward propagation and the forward propagation carried out in a perceptron is explained in the following three steps.

Step 1: For each input, multiply the input value $\mathbf{x_i}$ with weights $\mathbf{w_i}$ and sum all the multiplied values. Weights — represent the strength of the connection between neurons and decides how much influence the given input will have on the neuron's output. If the weight w_1 has a higher value than the weight w_2, then the input x_1 will have a higher influence on the output than w_2.

$$\sum = (x_1 \times w_1) + (x_2 \times w_2) + \cdots + (x_n \times w_n)$$

The row vectors of the inputs and weights are $x = [x_1, x_2, \ldots, x_n]$ and $w = [w_1, w_2, \ldots, w_n]$ respectively and their *dot product* is given by

$$x.w = (x_1 \times w_1) + (x_2 \times w_2) + \cdots + (x_n \times w_n)$$

Hence, the summation is equal to the *dot product* of the vectors *x* and *w*

$$\sum = x.w$$

Step 2: Add bias **b** to the summation of multiplied values and let's call this *z*. Bias — also known as the offset is necessary in most of the cases, to move the entire activation function to the left or right to generate the required output values.

$$z = x.w + b$$

Step 3: Pass the value of *z* to a non-linear activation function. Activation functions — are used to introduce non-linearity into the output of the neurons, without which the neural network will just be a linear function. Moreover, they have a significant impact on the learning speed of the neural network. Perceptrons have *binary step function* as their activation function. However, we shall use s*igmoid* — also known as *logistic* function as our activation function.

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

where **σ** denotes the s*igmoid* activation function and the output we get after the forward prorogation is known as the *predicted value* **ŷ**.

Learning Algorithm

The learning algorithm consists of two parts — backpropagation and optimization.

Backpropagation: Backpropagation, short for *backward propagation of errors,* refers to the algorithm for computing the gradient of the loss function with respect to the weights. However, the term is often used to refer to the entire learning algorithm. The backpropagation carried out in a perceptron is explained in the following two steps.

Step 1: To know an estimation of how far are we from our desired solution a *loss function* is used. Generally, *mean squared error* is chosen as the loss function for regression problems and *cross entropy* for classification problems. Let's take a regression problem and its loss function be mean squared error, which squares the difference between *actual* (y_i) and *predicted value* ($\hat{y}_i$).

$$MSE_i = (y_i - \hat{y}_i)^2$$

Loss function is calculated for the entire training dataset and their average is called the *Cost function* **C**.

$$C = MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Step 2: In order to find the best weights and bias for our Perceptron, we need to know how the cost function changes in relation to weights and bias. This is done with the help of the *gradient*s (*rate of change)* — how one quantity changes in relation to another quantity. In our case, we need to find the gradient of the cost function with respect to the weights and bias.

Let's calculate the gradient of cost function **C** with respect to the weight $\mathbf{w_i}$ using *partial derivation.* Since the cost function is not directly related to the weight w_i, let's use the chain rule.

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w_i}$$

Now we need to find the following three gradients

$$\frac{\partial C}{\partial \hat{y}} = ? \qquad \frac{\partial \hat{y}}{\partial z} = ? \qquad \frac{\partial z}{\partial w_1} = ?$$

Let's start with the gradient of the c*ost function (C)* with respect to the *predicted value* ($\hat{y}$)

$$\frac{\partial C}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = 2 \times \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$

Let $y = [y_1 , y_2 , \dots y_n]$ and $\hat{y} = [\hat{y}_1 , \hat{y}_2 , \dots \hat{y}_n]$ be the row vectors of actual and predicted values. Hence the above equation is simplified as

$$\frac{\partial C}{\partial \hat{y}} = \frac{2}{n} \times sum(y - \hat{y})$$

Now let's find the gradient of the *predicted value* with respect to the z. This will be a bit lengthy.

$$\begin{aligned}
\frac{\partial \hat{y}}{\partial z} &= \frac{\partial}{\partial z}\sigma(z) \\
&= \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1}{(1+e^{-z})} \times \frac{e^{-z}}{(1+e^{-z})} \\
&= \frac{1}{(1+e^{-z})} \times \left(1 - \frac{1}{(1+e^{-z})}\right) \\
&= \sigma(z) \times (1 - \sigma(z))
\end{aligned}$$

The gradient of z with respect to the weight w_i is

$$\begin{aligned}
\frac{\partial z}{\partial w_i} &= \frac{\partial}{\partial w_i}(z) \\
&= \frac{\partial}{\partial w_i} \sum_{i=1}^{n} (x_i.w_i + b) \\
&= x_i
\end{aligned}$$

Therefore we get,

$$\frac{\partial C}{\partial w_i} = \frac{2}{n} \times sum(y - \hat{y}) \times \sigma(z) \times (1 - \sigma(z)) \times x_i$$

What about Bias? — Bias is theoretically considered to have an input of constant value *1*. Hence,

$$\frac{\partial C}{\partial b} = \frac{2}{n} \times sum(y - \hat{y}) \times \sigma(z) \times (1 - \sigma(z))$$

Optimization: Optimization is the selection of the best element from some set of available alternatives, which in our case, is the selection of best weights and bias of the perceptron. Let's choose *gradient descent* as our optimization algorithm, which changes the *weights* and *bias,* proportional to the negative of the gradient of the cost function with respect to the corresponding weight or bias. *Learning rate* (α) is a hyperparameter which is used to control how much the weights and bias are changed.

The weights and bias are updated as follows and the backpropagation and gradient descent is repeated until convergence.

$$w_i = w_i - \left(\alpha \times \frac{\partial C}{\partial w_i}\right)$$

$$b = b - \left(\alpha \times \frac{\partial C}{\partial b}\right)$$

Final Thoughts

I hope that you've found this article useful and understood the mathematics behind the neural networks and deep learning. I have explained the working of a single neuron in this article. However, these basic concepts are applicable to all kinds of neural networks with some modifications.

Data Science | Artificial Intelligence | Machine Learning | Programming | Technology