Activation function

Lucas C. França

2022-02-22

Contents

Neural networks have become a powerful tool in various scientific research and industrial applications domains. The first step in using neural networks is to determine a suitable topology, optimal if possible (number of inputs, outputs; the number of hidden layers; and neurons in each layer) and optimal internal parameters(weights, biases, and activation functions). The choice of activation functions significantly affects the training dynamics and task performance. Common activation functions include rectified linear unit(ReLU), tanh, and logistic functions like Sigmoid and Softmax. This article will provide a general explanation of activation functions as well as specifics about each of the types;

1 Introduction

In artificial neural networks, the **activation function** of a node defines the output of that node given an input or set of inputs. A standard integrated circuit can be seen as a digital network of activation functions that can be "ON"(1) or "OFF"(0), depending on the input.

The activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input.

2 Classification of activation functions

The most common activation functions can be basically divided into two categories: Linear and Nonlinear functions.

2.1 Linear activation functions

2.1.1 Linear activation:

The linear activation function is also called "identity" (multiplied by 1.0) or "no activation."

This is because the linear activation function does not change the weighted sum of the input in any way and instead returns the value directly.

The simplest activation function is referred to as the linear activation, where no transform is applied at all. A network comprised of only linear activation functions is very easy to train, but cannot learn complex mapping functions. Linear activation functions are still used in the output layer for networks that predict a quantity (e.g. regression problems).

The function can be defined by:

$$\phi(\mathbf{v}) = \alpha + \mathbf{v}'\mathbf{b}$$

The linear activation function can be obtained with the following code, in R:

```
linear <- function(x){
   return(x)
}
```

We can get an intuition for the shape of this function with the worked example below.

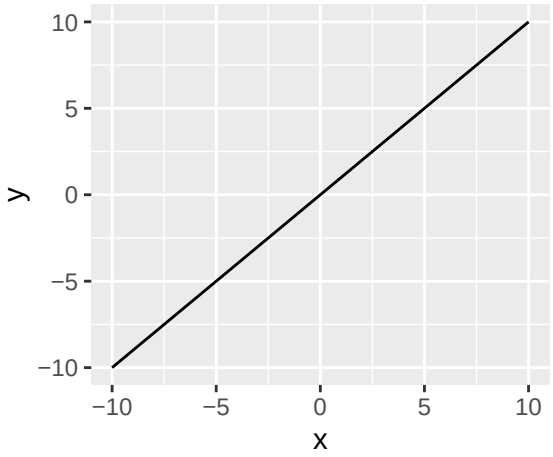

Figure 1: Plot of Inputs vs. Outputs for the Linear Activation Function.

2.1.2 Heaviside activation (Binary step):

Heaviside (Binary step, 0 or 1, high or low) step function is typically only useful within single-layer perceptrons, an early type of neural networks that can be used for classification in cases where the input data is linearly separable. These functions are useful for binary classification tasks. The output is a certain value, A1, if the input sum is above a certain threshold and A0 if the input sum is below a certain threshold. The values used by the Perceptron were A1 = 1 and A0 = 0

The binary step function can be used as an activation function while creating a binary classifier. But the function is not really helpful when there are multiple classes to deal with. Moreover, the gradient of the step function is zero which causes a hindrance in the backpropagation process. That is if you calculate the derivative of f(x) with respect to x, it comes out to be 0.

The heaviside activation function can be obtained with the following code, in R:

```
heaviside <- function(x){
  ifelse(x < 0, 0, 1)
```

We can get an intuition for the shape of this function with the worked example below.

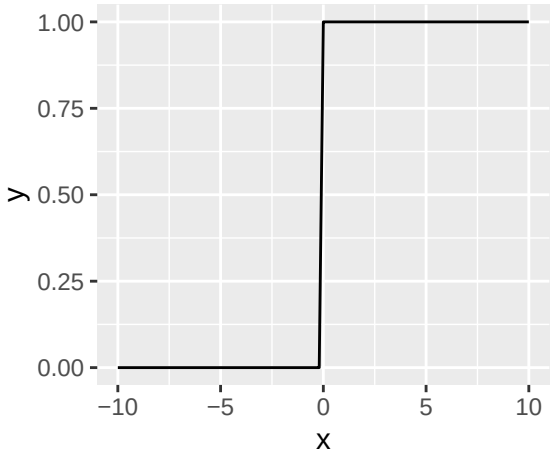

Figure 2: Plot of Inputs vs. Outputs for the Heaviside Activation Function.

2.2 Nonlinear activation functions

Nonlinear activation functions are preferred as they allow the nodes to learn more complex structures in the data. Traditionally, two widely used nonlinear activation functions are the sigmoid and hyperbolic tangent activation functions.

2.2.1 Logistic activation (Sigmoid):

The sigmoid activation function is also called the logistic function. It is the same function used in the logistic regression classification algorithm. The function takes any real value as input and outputs values in the range 0 to 1. The larger the input, the closer the output will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0.

The function is mostly used for binary classification problems.

The sigmoid activation function is calculated as follows:

$$\phi(\mathbf{v}) = (1 + exp(-\alpha - \mathbf{v}'\mathbf{b})^{-}1$$

Where e is a mathematical constant, which is the base of the natural logarithm.

The sigmoid activation function can be obtained with the following code, in R:

```
sigmoid <- function(x){

    x = 1/ (1 + exp(-x))
    return(x)

}
```

We can get an intuition for the shape of this function with the worked example below.

2.2.2 Hyperbolic tangent function (tanh)

Tanh function is very similar to the sigmoid/logistic activation function, data is centered around zero, so the derivatives will be higher. Tanh quickly converges than sigmoid and logistic activation functions.

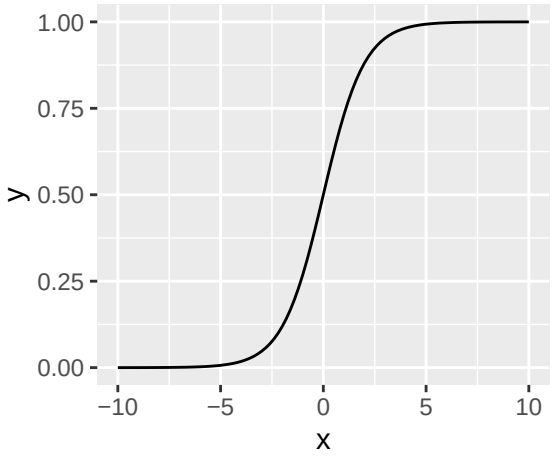

Figure 3: Plot of Inputs vs. Outputs for the Sigmoid Activation Function.

The tanh activation function is calculated as follows:

$$\phi(\mathbf{v}) = \frac{\mathbf{e}^{x} - \mathbf{e}^{-x}}{\mathbf{e}^{x} + \mathbf{e}^{-x}}$$

The tanh activation function can be obtained with the following code, in R:

```
tanh <- function(x){

    x = (exp(x) - exp(-x))/(exp(x) + exp(-x))
    return(x)

}
```

We can get an intuition for the shape of this function with the worked example below.

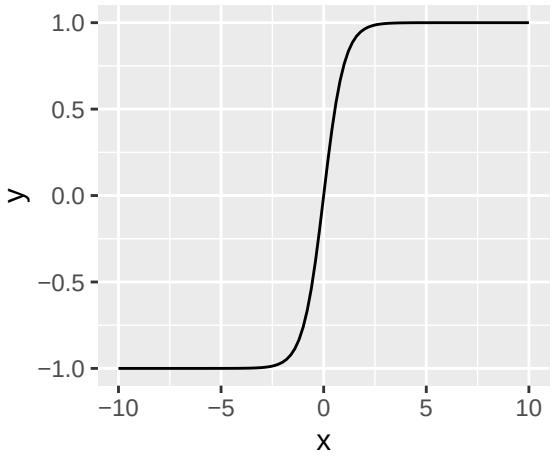

Figure 4: Plot of Inputs vs. Outputs for the Tahn Activation Function.

A general problem with both the sigmoid and tanh functions is that they saturate. This means that large values snap to 1.0 and small values snap to -1 or 0 for tanh and sigmoid respectively. Further, the functions

are only really sensitive to changes around their mid-point of their input, such as 0.5 for sigmoid and 0.0 for tanh.

The limited sensitivity and saturation of the function happen regardless of whether the summed activation from the node provided as input contains useful information or not. Once saturated, it becomes challenging for the learning algorithm to continue to adapt the weights to improve the performance of the model.

Finally, as the capability of hardware increased through GPUs' very deep neural networks using sigmoid and tanh activation functions could not easily be trained.

Layers deep in large networks using these nonlinear activation functions fail to receive useful gradient information. Error is backpropagated through the network and used to update the weights. The amount of error decreases dramatically with each additional layer through which it is propagated, given the derivative of the chosen activation function. This is called the vanishing gradient problem and prevents deep (multi-layered) networks from learning effectively.

Although the use of nonlinear activation functions allows neural networks to learn complex mapping functions, they effectively prevent the learning algorithm from working with deep networks.

2.2.3 ReLU activation:

In order to use stochastic gradient descent with backpropagation of errors to train deep neural networks, an activation function is needed that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned.

The function must also provide more sensitivity to the activation sum input and avoid easy saturation.

The solution is to use the rectified linear activation function, or ReLU for short.

The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

The function can be defined by:

$$\phi(\mathbf{v}) = \max\ (0, \alpha + \mathbf{v}'\mathbf{b})$$

The ReLU activation function can be obtained with the following code, in R:

```
relu <- function(x){
  ifelse(x < 0, 0, x)

}
```

Below are a few examples of inputs and outputs of the rectified linear activation function.

The function is linear for values greater than zero, meaning it has a lot of the desirable properties of a linear activation function when training a neural network using backpropagation. Yet, it is a nonlinear function as negative values are always output as zero.

But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

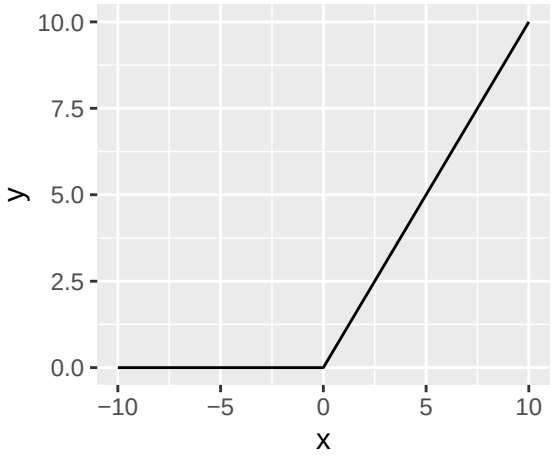

Figure 5: Plot of Inputs vs. Outputs for the ReLU Activation Function.

2.2.4 Leaky rectified linear unit (Leaky ReLU)

It is an attempt to solve the dying ReLU problem, the function has a small slope for negative values instead of a flat slope. The slope coefficient is determined before training. This type of activation function is popular in tasks where we we may suffer from sparse gradients, for example training generative adversarial networks.

The leaky rectifier allows for a small, non-zero gradient when the unit is saturated and not active.

Mathematically, it is defined as follows:

$$f(x) = \begin{cases} 0.01x, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$$

The leaky relu activation function can be obtained with the following code, in R:

```
leakyrelu <- function(x){
  ifelse(x < 0, 0.1*x, x)

}
```

Below are a few examples of inputs and outputs of the rectified linear activation function.

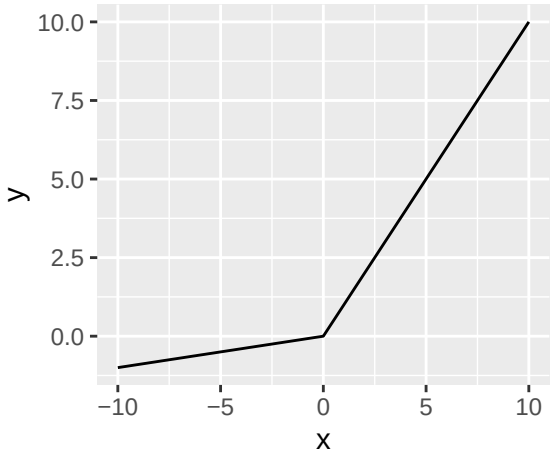

Figure 6: Plot of Inputs vs. Outputs for the Leaky ReLU Activation Function.

If you compare this with the image for traditional ReLU above, you'll see that for all $x < 0$, the outputs are slightly descending. The thesis is that these small numbers reduce the death of ReLU activated neurons. This way, you'll have to worry less about the initialization of your neural network and the normalization of your data. Although these topics remain important, they are slightly less critical.

3 Conclusion and Discussion

In this article, we have discussed about neural networks and activation functions in brief, its uses and its disadvantages.

4 References

1. Activation function. (2022, february 22). In Wikipedia. https://en.wikipedia.org/wiki/Activation_function
2. Activation Functions in Neural Networks.(2022, february 22). In Towardsdatascience. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
3. 12 Types of Neural Network Activation Functions: How to Choose? (2022, february 22). In v7labs. https://www.v7labs.com/blog/neural-networks-activation-functions
4. How to Choose an Activation Function for Deep Learning. (2022, february 22). In Machine Learning Mastery. https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/
5. A Gentle Introduction to the Rectified Linear Unit (ReLU). (2022, february 22).In Machine Learning Mastery.https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
6. Secret Sauce behind the beauty of Deep Learning: Beginners guide to Activation Functions.(2022, february 22). In Towardsdatascience. https://towardsdatascience.com/secret-sauce-behind-the-beauty-of-deep-learning-beginners-guide-to-activation-functions-a8e23a57d046