**Activation Functions**

Activation functions are an integral building block of [**neural networks**](https://www.datacamp.com/blog/what-are-neural-networks) that enable them to learn complex patterns in data. They transform the input signal of a node in a neural network into an output signal that is then passed on to the next layer. Without activation functions, neural networks would be restricted to modeling only linear relationships between inputs and outputs.

Activation functions introduce non-linearities, allowing neural networks to learn highly complex mappings between inputs and outputs.

Choosing the right activation function is crucial for training neural networks that generalize well and provide accurate predictions. In this post, we will provide an overview of the most common activation functions, their roles, and how to select suitable activation functions for different use cases.

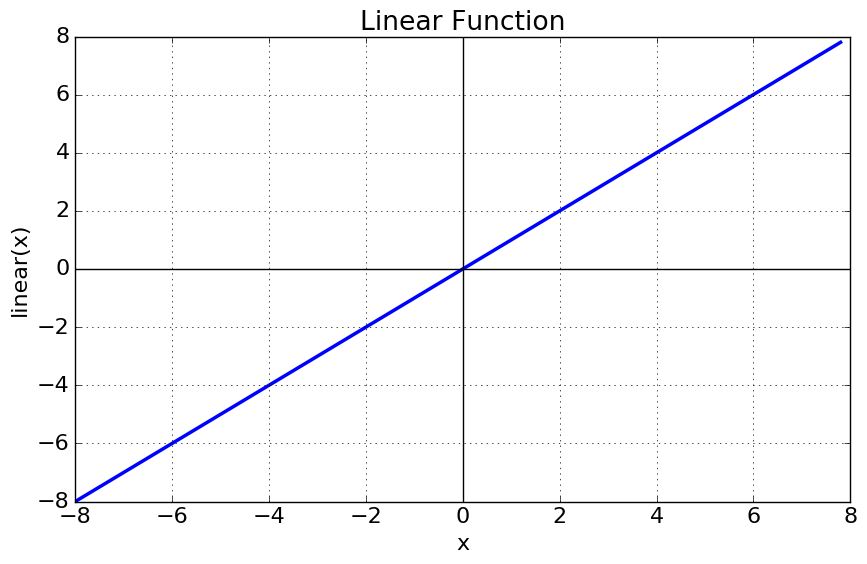
Whether you are just starting out in [**deep learning**](https://www.datacamp.com/courses/introduction-to-deep-learning-in-python) or are a seasoned practitioner, understanding activation functions in depth will build your intuition and improve your application of neural networks.

**Types of Activation Function**

**Linear Activation Function**

linear activation functions, also known as identity functions, when they want the output to be the same as the input signal. Identity is differentiable, and like a train passing through a station without stopping, this activation function doesn’t change the signal in any way, so it’s not used within internal layers of a DL network.

Although, in most cases, this might not sound very useful, it is when you want the outputs of your neural network to be continuous rather than modified or discrete. There is no convergence of data, and nothing decreases either. If you use this activation function for every layer, then it would collapse the layers in a neural network into one. So, not very useful unless that’s exactly what you need or there are different activation functions in the subsequent hidden layers.

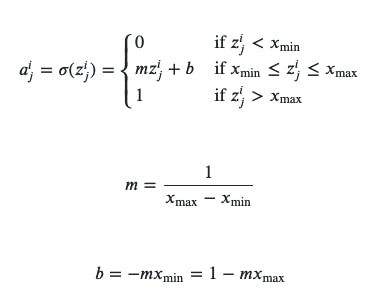
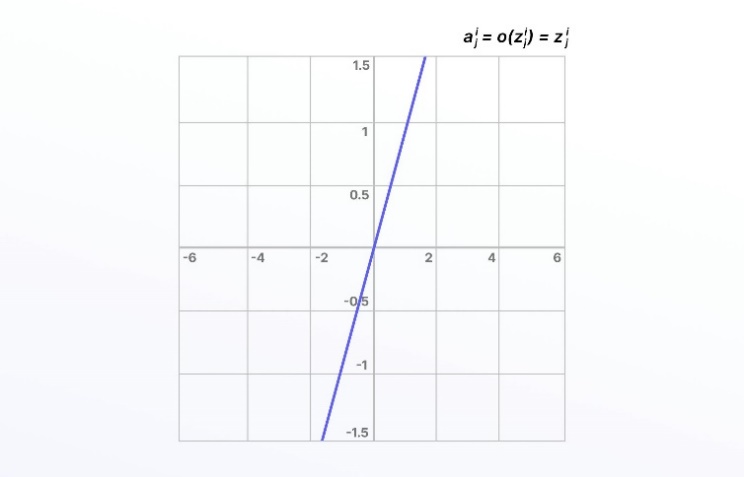


**Piecewise Linear (PL)**

Piecewise linear is an iteration on the above, except involving an [**affine function**](https://en.wikipedia.org/wiki/Affine_transformation), so it is also known as piecewise affine. It’s defined using a bound or unbound sequence of numbers, either compact, finite, or locally finite, and is not differentiable due to threshold points, so it only propagates signals in the slope region.

Piecewise linear is calculated using a range of numbers required for the particular equation, anything less than the range is 0, and anything greater is 1. Between 0 and 1, the signals going from one layer to the next are linearly-interpolated.

Here is the mathematical representation:



Linear activation functions don’t allow neural networks or deep learning networks to develop complex mapping and algorithmic interpretation between inputs and outputs.

**Non-Linear Activation Functions**

Non-linear activation functions solve the limitations and drawbacks of simpler activation functions, such as the vanishing gradient problem. Non-linear functions, such as **Sigmoid, Tanh, Rectified Linear Unit (ReLU)**, and numerous others.

There are several advantages to using non-linear activation functions, as they can facilitate backpropagation and stacking. Non-linear combinations and functions used throughout a network mean that data scientists and machine learning teams creating and training a model can adjust weights and biases, and outputs are represented as a functional computation.

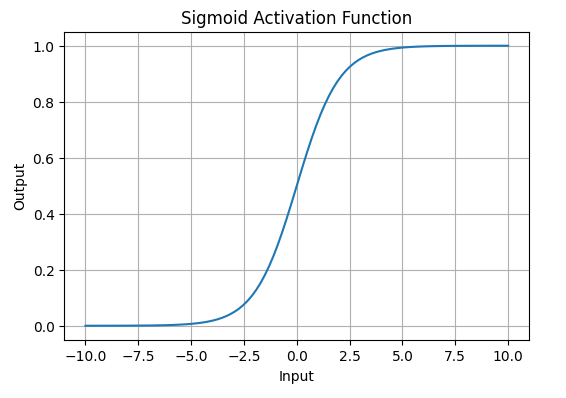
In other words, everything going into, through, and out of a neural network can be measured more effectively when non-linear activation functions are used, and therefore, the equations are adjusted until the right outputs are achieved.

**1. Sigmoid Function**

[**Sigmoid Activation Function**](https://www.geeksforgeeks.org/derivative-of-the-sigmoid-function/) is characterized by ‘S’ shape. It is mathematically defined asA=11+e−x*A*=1+*e*−*x*1​​. This formula ensures a smooth and continuous output that is essential for gradient-based optimization methods.

* It allows neural networks to handle and model complex patterns that linear equations cannot.

1. The output ranges between 0 and 1, hence useful for binary classification.
2. The function exhibits a steep gradient when x values are between -2 and 2. This sensitivity means that small changes in input x can cause significant changes in output y, which is critical during the training process.



*Sigmoid or Logistic Activation Function Graph*

**2. Tanh Activation Function**

Tanh function or hyperbolic tangent function**,** is a shifted version of the sigmoid, allowing it to stretch across the y-axis. It is defined as:

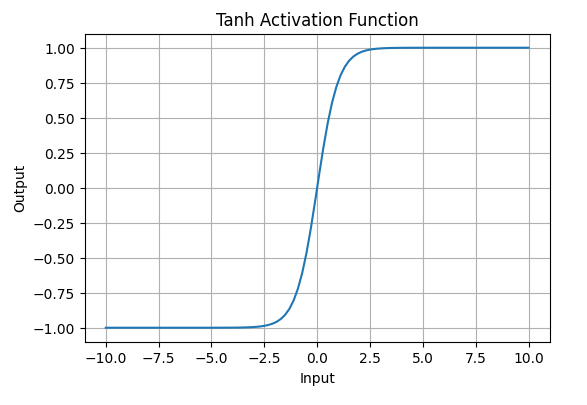
f(x)=tanh⁡(x)=21+e−2x–1.*f*(*x*)=tanh(*x*)=1+*e*−2*x*2​–1.

Alternatively, it can be expressed using the sigmoid function:

tanh⁡(x)=2×sigmoid(2x)–1tanh(*x*)=2×sigmoid(2*x*)–1

* **Value Range**: Outputs values from -1 to +1.

1. **Non-linear**: Enables modeling of complex data patterns.
2. **Use in Hidden Layers**: Commonly used in hidden layers due to its zero-centered output, facilitating easier learning for subsequent layers.



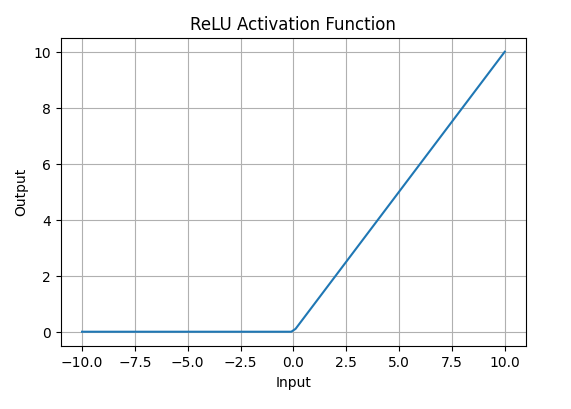
*Tanh Activation Function*

**3. ReLU (Rectified Linear Unit) Function**

[**ReLU activation**](https://www.geeksforgeeks.org/relu-activation-function-in-deep-learning/) is defined by A(x)=max⁡(0,x)*A*(*x*)=max(0,*x*), this means that if the input x is positive, ReLU returns x, if the input is negative, it returns 0.

* **Value Range**: [0,∞)[0,∞), meaning the function only outputs non-negative values.

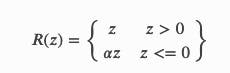
1. **Nature**: It is a **non-linear** activation function, allowing neural networks to learn complex patterns and making backpropagation more efficient.
2. **Advantage over other Activation:**ReLU is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

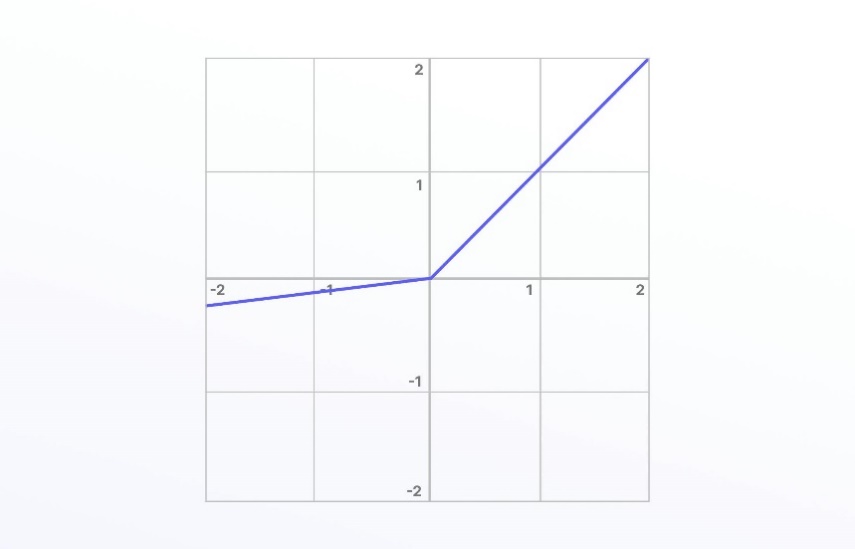


*ReLU Activation Function*

**Leaky ReLU Function**

One solution to the “dying ReLU” problem is a variation on this known as the Leaky ReLU activation function. With the Leaky ReLU, instead of being 0 when *𝑧*<0, a leaky ReLU allows a small, non-zero, constant gradient *𝛼*(Normally, *𝛼*=0.01).

Here is the mathematical representation: 



Leaky ReLU has been shown to perform better than the traditional ReLU activation function. However, because it possesses linearity it can’t be used for more complex classification tasks and lags behind more advanced activation functions such as Sigmoid and Tanh.