

Open in app ↗



Search

Write

✦ **Jump-start your best year yet:** Become a member and get 25% off the first year

Activation Functions: Sigmoid, Tanh, ReLU, Leaky ReLU, Softmax



Mukesh Chaudhary · Follow

6 min read · Aug 28, 2020



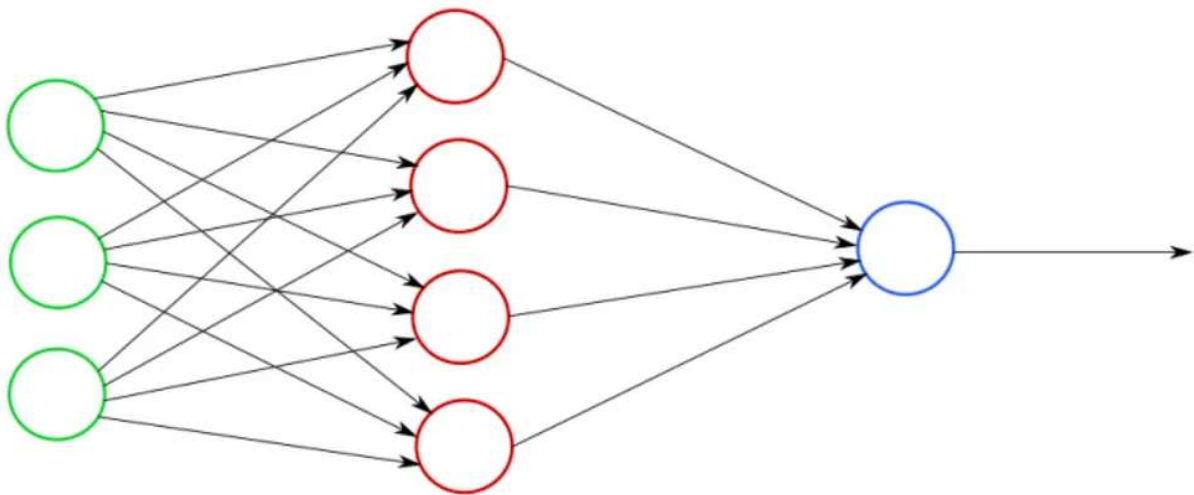
136



2

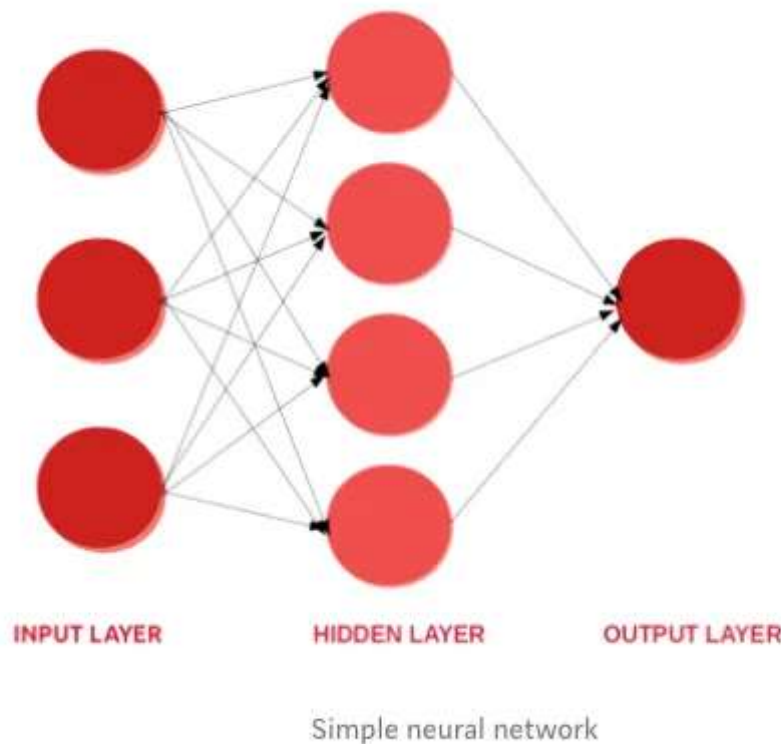


Comparing Sigmoid function with others activation functions and Importance ReLU in Hidden Layer of NN

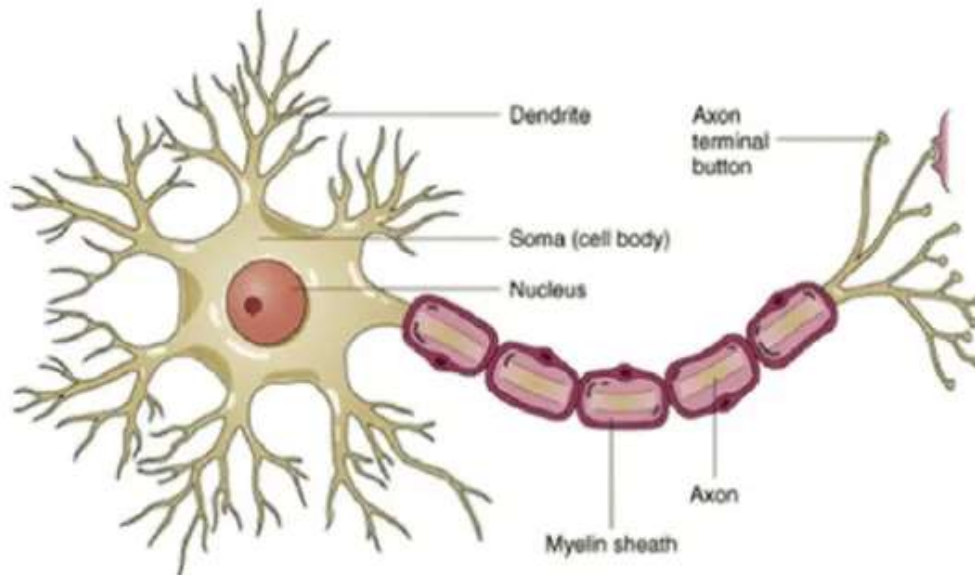


In this blog, I will try to compare and analysis Sigmoid(logistic) activation function with others like Tanh, ReLU, Leaky ReLU, Softmax activation function. In my previous blog, I described on how to work sigmoid function in logistic Regression algorithm. There , I described with mathematical term and python implementation code. If you want to read, you may visit the [link](#). So now i want to analysis more sigmoid function with another activation functions. Let's move . These all are activation function used generally in Neural Network algorithm and deep learning. Here I don't go in depth detail about Neural Network . We try to focus only activation functions. We will discuss Neural Network on another blog.

Basic idea of a Neural Network works-

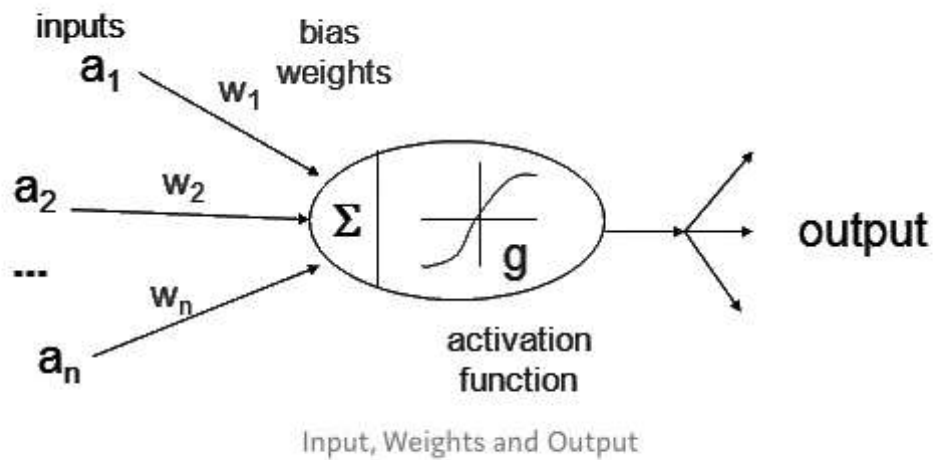


There are many algorithms in the market to solve classification problem . Neural Network is one of them which is very famous for predicting accurate data. However, it takes a lot of computational time. It is inspired by the way biological neural systems process data. *It contains layers of interconnected nodes or neurons arranged in interconnected layers.*



(a) Biological neuron

The information moves from the input layer to the hidden layers. In a simple case of each layer, we just multiply the inputs by the **weights**, add a **bias** and apply an **activation function** to the result and pass the output to the next layer. We keep repeating this process until we reach the last layer.



Activation Function:

Activation functions are generally two types, These are

1. Linear or Identity Activation Function
2. Non-Linear Activation Function.

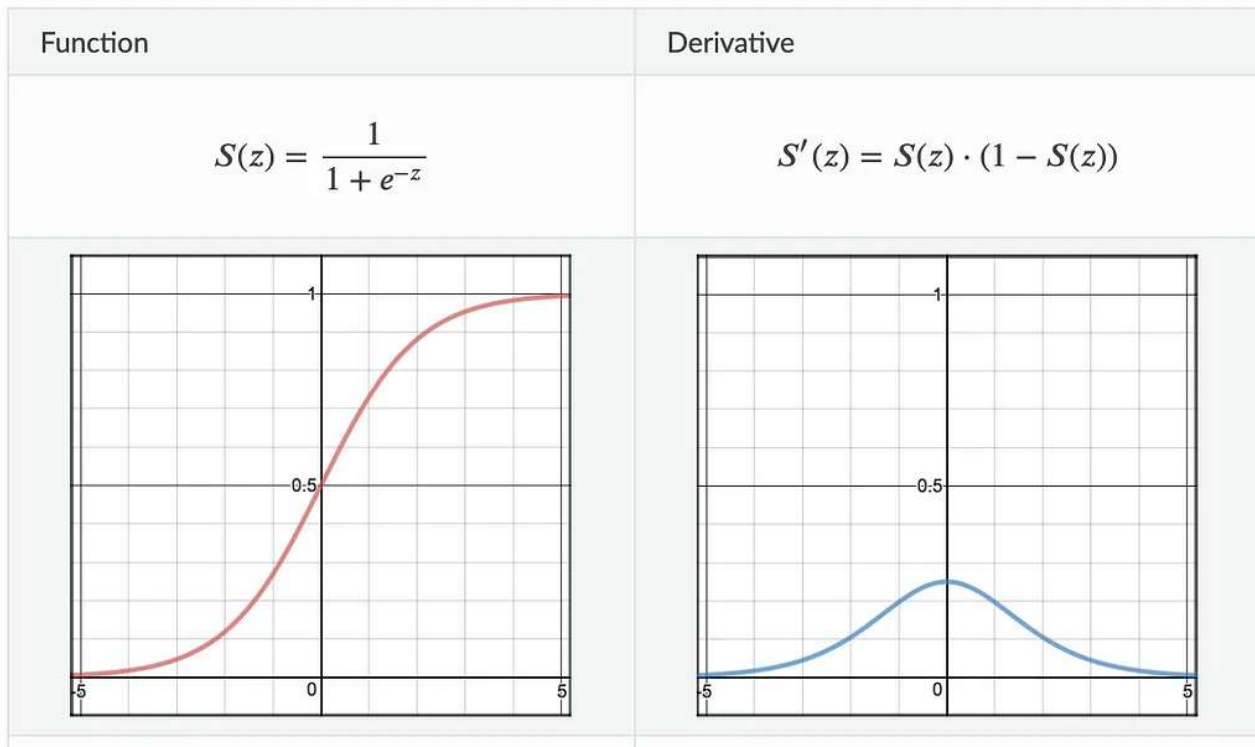
Generally, neural networks use **non-linear activation functions**, which can help the network learn complex data, compute and learn almost any function representing a question, and provide accurate predictions. They **allow back-propagation** because they have a derivative function which is related to the inputs.

Non-linear Activation Functions:

Above listed all activation functions are belong to non-linear activation functions. And we will discuss below more in details.

1. Sigmoid Activation Function:

Sigmoid Activation function is very simple which takes a real value as input and gives probability that 's always between 0 or 1. It looks like 'S' shape.



Sigmoid function and it's derivative

It's non-linear, continuously differentiable, monotonic, and has a fixed output range. Main advantage is simple and good for classifier. But Big disadvantage of the function is that it gives rise to a problem of “vanishing gradients” because Its output isn't zero centered. It makes the gradient updates go too far in different directions. $0 < \text{output} < 1$, and it makes optimization harder. That takes very high computational time in hidden layer of neural network

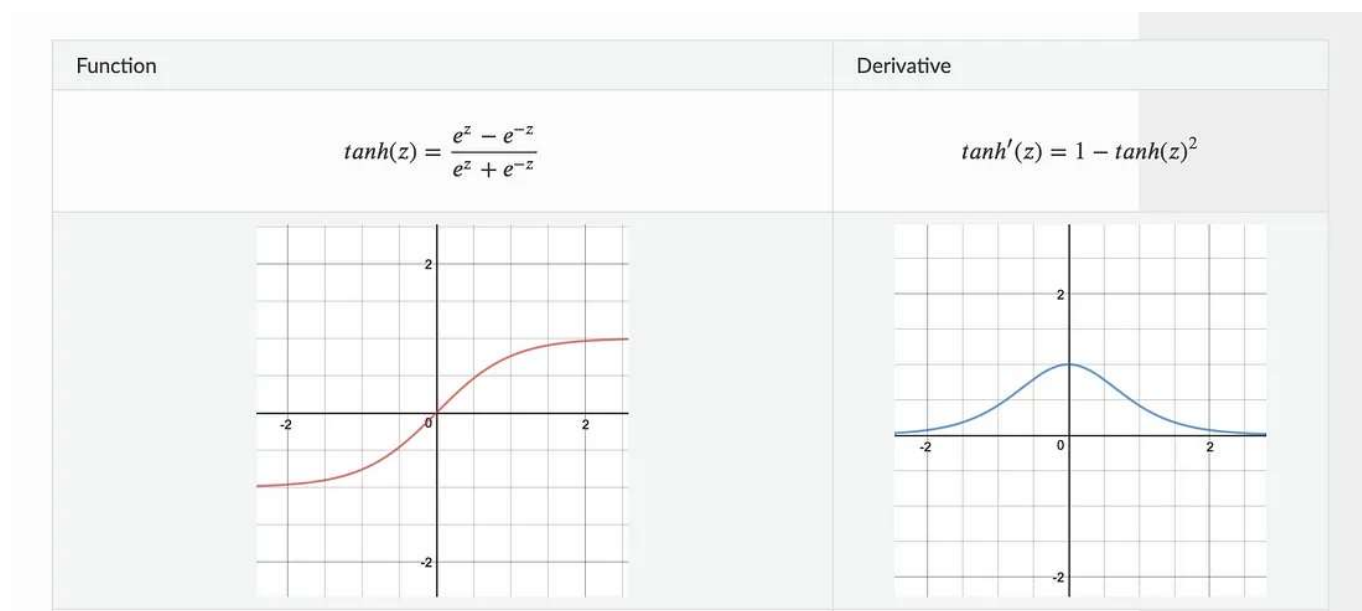
```
# sigmoid function
def sigmoid(z):
    return 1.0 / (1 + np.exp(-z))

# Derivative of sigmoid function
def sigmoid_prime(z):
```

```
return sigmoid(z) * (1-sigmoid(z))
```

2. Tanh or Hyperbolic tangent:

Tanh help to solve non zero centered problem of sigmoid function. Tanh squashes a real-valued number to the range $[-1, 1]$. It's non-linear too.



Derivative function give us almost same as sigmoid's derivative function.

It solve sigmoid's drawback but it still can't remove the vanishing gradient problem completely.

When we compare tanh activation function with sigmoid , this picture give you clear idea.

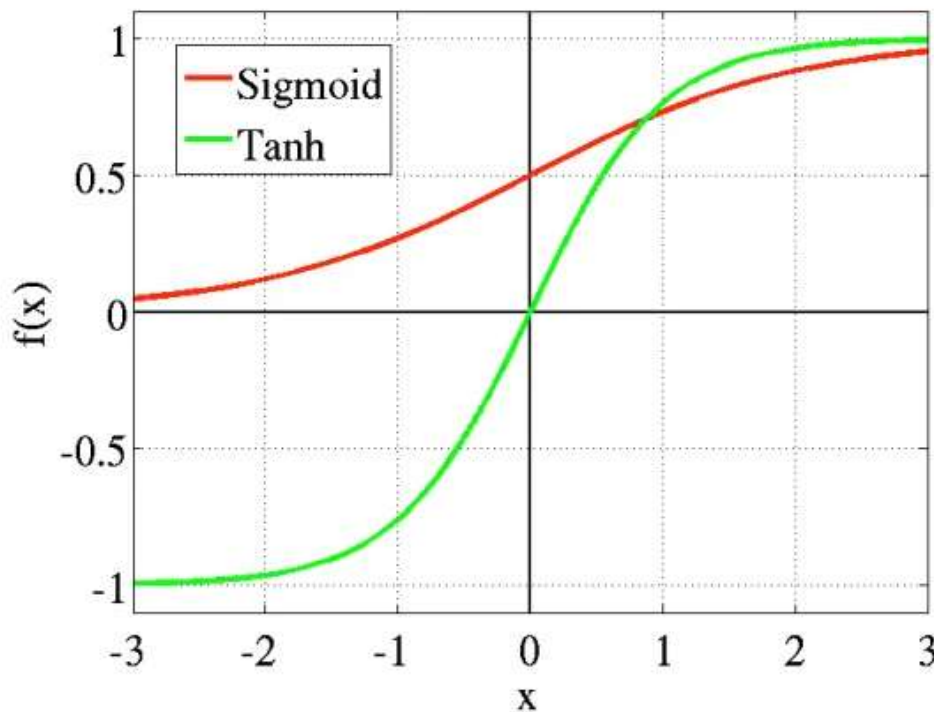


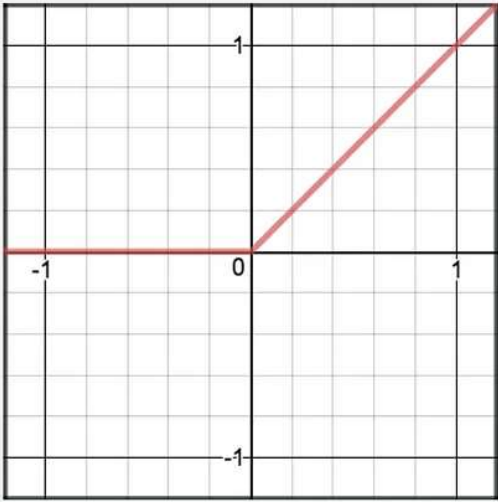
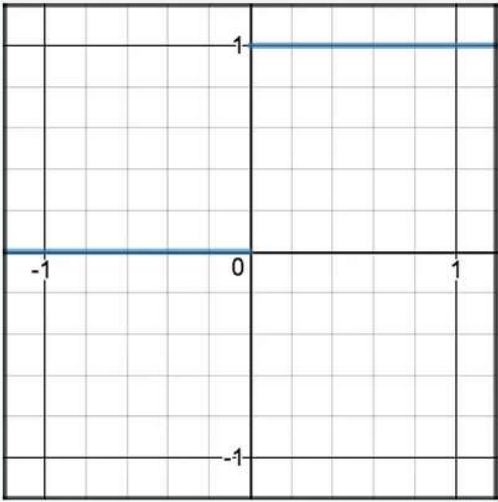
Fig: tanh v/s Logistic Sigmoid

```
# tanh activation function
def tanh(z):
    return (np.exp(z) - np.exp(-z)) / (np.exp(z) + np.exp(-z))

# Derivative of Tanh Activation Function
def tanh_prime(z):
    return 1 - np.power(tanh(z), 2)
```

3. ReLU (Rectified Linear Unit):

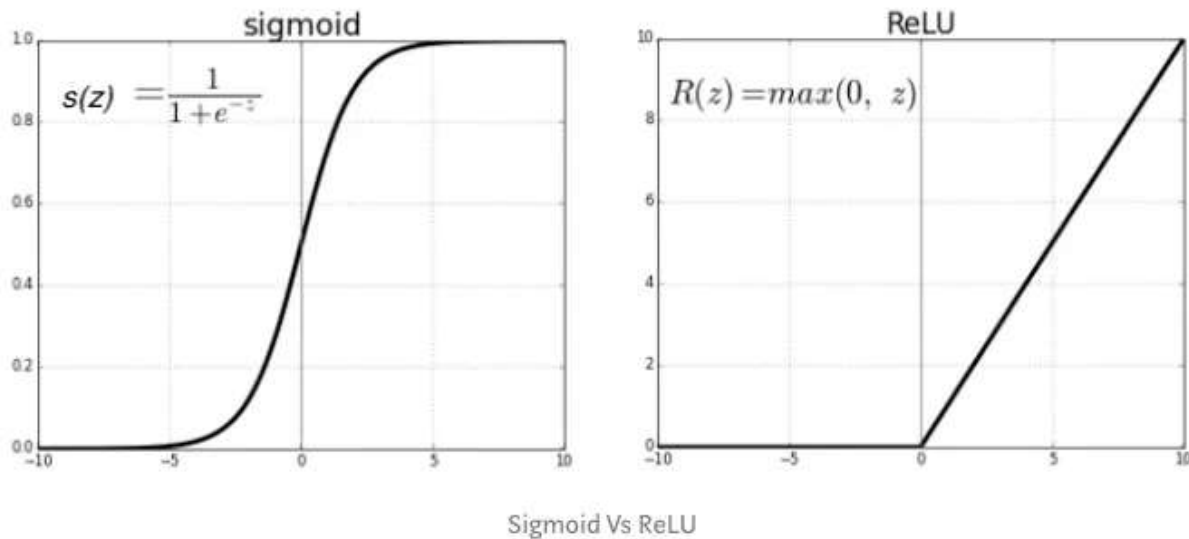
This is most popular activation function which is used in hidden layer of NN. The formula is deceptively simple: $\max(0, z)$. Despite its name and appearance, it's not linear and provides the same benefits as Sigmoid but with better performance.

Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ 0 & z \leq 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$
	

ReLU Activation Function and It's derivative

It's main advantage is that it avoids and rectifies vanishing gradient problem and less computationally expensive than tanh and sigmoid. But it has also some draw back . Sometime some gradients can be fragile during training and can die. That leads to dead neurons. In another words, for activations in the region ($x < 0$) of ReLU, gradient will be 0 because of which the weights will not get adjusted during descent. That means, those neurons which go into that state will stop responding to variations in error/ input (simply because gradient is 0, nothing changes). So We should be very carefully to choose activation function , and activation function should be as per business requirement.

When we compare with sigmoid activation function, It's look like

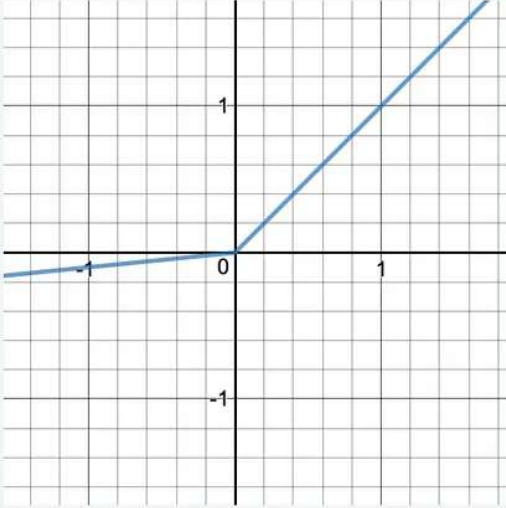
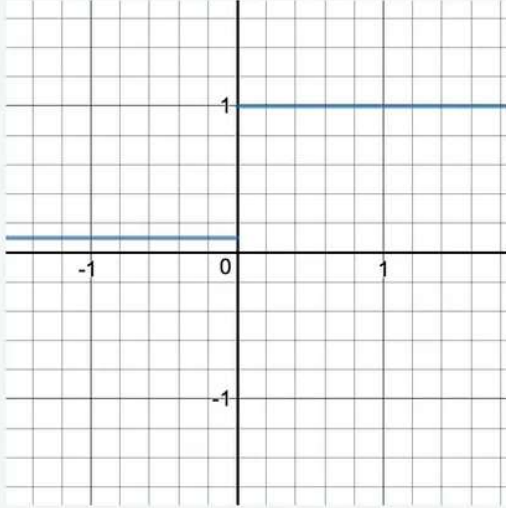


```
# ReLU activation function
def relu(z):
    return max(0, z)

# Derivative of ReLU Activation Function
def relu_prime(z):
    return 1 if z > 0 else 0
```

4. Leaky ReLU

It prevents dying ReLU problem. This variation of ReLU has a small positive slope in the negative area, so it does enable back-propagation, even for negative input values

Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ \alpha z & z \leq 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ \alpha & z < 0 \end{cases}$
	

Leaky ReLU does not provide consistent predictions for negative input values. During the front propagation if the learning rate is set very high it will overshoot killing the neuron.

The idea of leaky ReLU can be extended even further. Instead of multiplying x with a constant term we can multiply it with a hyper-parameter which seems to work better the leaky ReLU. This extension to leaky ReLU is known as **Parametric ReLU**.

While we compare Leaky-ReLU with ReLU, then It shows clear concept of difference between them.

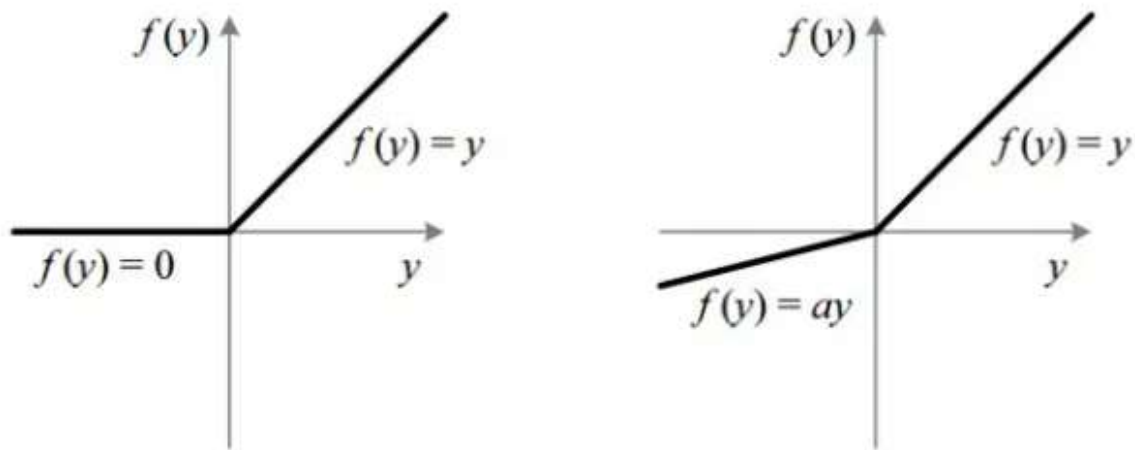


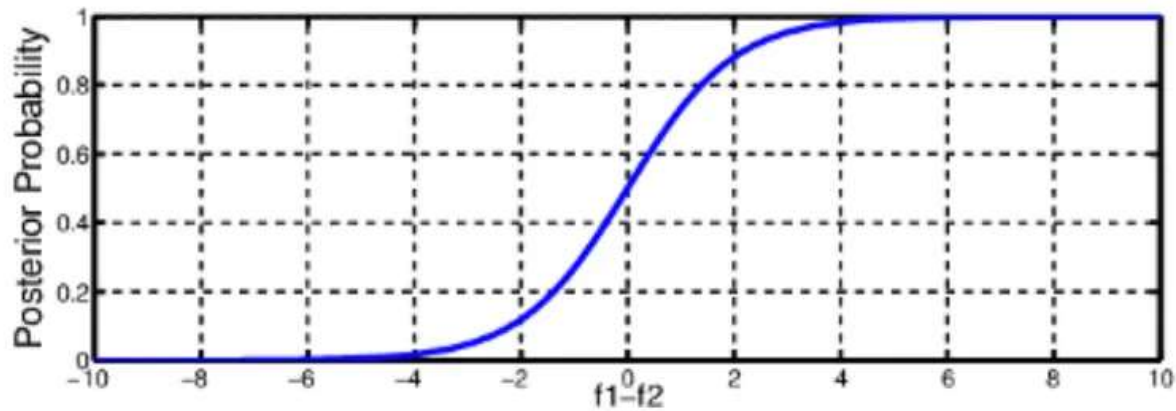
Fig : ReLU v/s Leaky ReLU

```
# Leaky_ReLU activation function
def leakyrelu(z, alpha):
    return max(alpha * z, z)

# Derivative of leaky_ReLU Activation Function
def leakyrelu_prime(z, alpha):
    return 1 if z > 0 else alpha
```

5. Softmax

Generally, we use the function at last layer of neural network which calculates the probabilities distribution of the event over 'n' different events. The main advantage of the function is able to handle multiple classes.



when we compare the sigmoid and softmax activation functions , they produce different results.

$$f(x) = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid input values: -0.5, 1.2, -0.1, 2.4

Sigmoid output values: 0.37, 0.77, 0.48, 0.91

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \text{ for } j = 1, \dots, K$$

SoftMax input values: -0.5, 1.2, -0.1, 2.4

SoftMax output values: 0.04, 0.21, 0.05, 0.70

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \text{ for } j = 1, \dots, K$$

Sigmoid's probabilities produced by a Sigmoid are independent.

Furthermore, they are *not* constrained to sum to one: $0.37 + 0.77 + 0.48 + 0.91 = 2.53$. The reason for this is because the Sigmoid looks at each raw output value separately. Whereas **Softmax's** the outputs are interrelated. The Softmax probabilities will always sum to one by design: $0.04 + 0.21 + 0.05 + 0.70 = 1.00$. In this case, if we want to increase the likelihood of one class, the other has to decrease by an equal amount.

Conclusion:

In conclusion, we can see advantage and disadvantage of all activation functions. As per our business requirement, we can choose our required activation function. Generally, we use ReLU in hidden layer to avoid vanishing gradient problem and better computation performance, and Softmax function use in last output layer.

References

<https://keras.io/api/layers/activations/>

[https://en.wikipedia.org/wiki/Activation_function#:~:text=In%20artificial%20neural%20networks%2C%20the,%20\)%2C%20depending%20on%20input.](https://en.wikipedia.org/wiki/Activation_function#:~:text=In%20artificial%20neural%20networks%2C%20the,%20)%2C%20depending%20on%20input.)

<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

<https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f>

Data Science

Neural Networks



Written by Mukesh Chaudhary

Follow



150 Followers

More from Mukesh Chaudhary



M Mukesh Chaudhary

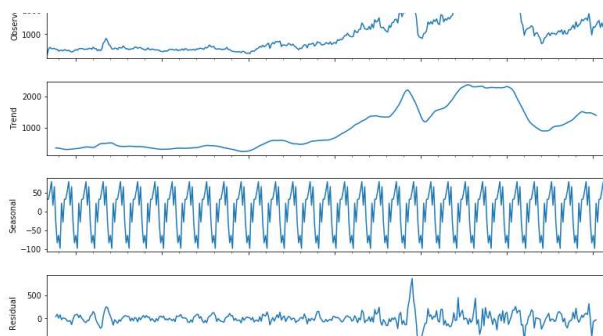
TF-IDF Vectorizer scikit-learn

Deep understanding TfidfVectorizer by customizing parameter

6 min read · Apr 24, 2020

👏 530 💬 6

🔖 ...

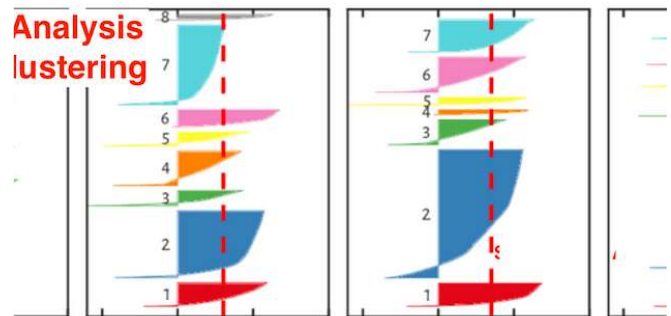


M Mukesh Chaudhary

Why is Augmented Dickey–Fuller test (ADF Test) so important in...

ADF Test

7 min read · Apr 9, 2020



M Mukesh Chaudhary

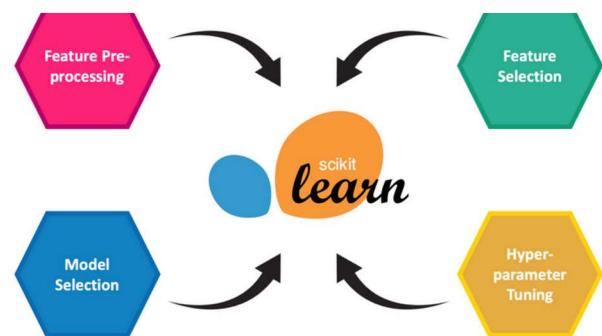
Silhouette Analysis in K-means Clustering

Cluster Evaluation: silhouette coefficient

5 min read · Jun 5, 2020

👏 49 💬

🔖 ...



M Mukesh Chaudhary

SKlearn: Pipeline & GridSearchCV

It makes so easy to fit data into model.

3 min read · Sep 4, 2020

48



...



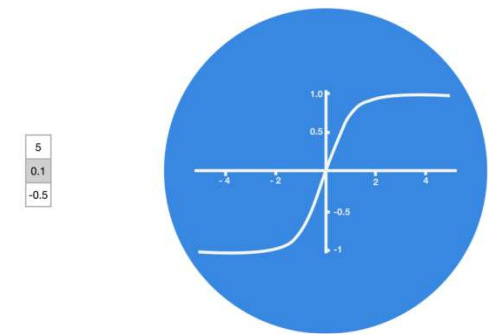
23



...

See all from Mukesh Chaudhary

Recommended from Medium



Everton Gomed, PhD

Activation Functions and Loss Functions in Deep Learning:...

Introduction

5 min read · Oct 8, 2023

100



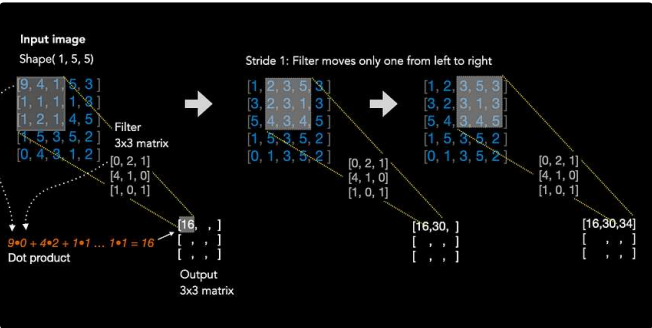
...



133



...



Frederik vl in Advanced Deep Learning

Understanding the Convolutional Filter Operation in CNN's.

5 min read · Aug 18, 2023

Lists



Predictive Modeling w/ Python

20 stories · 828 saves



Coding & Development

11 stories · 399 saves



Practical Guides to Machine Learning

10 stories · 956 saves



data science and AI

39 stories · 49 saves



Prudhviraju Srivatsavaya

Flatten Layer—Implementation, Advantage and Disadvantages

The Flatten layer is a crucial component in neural network architectures, especially whe...

4 min read · Oct 4, 2023



Puneeth Venugopal

The Chain Rule of Calculus: The Backbone of Deep Learning...

Introduction

4 min read · Oct 15, 2023



58

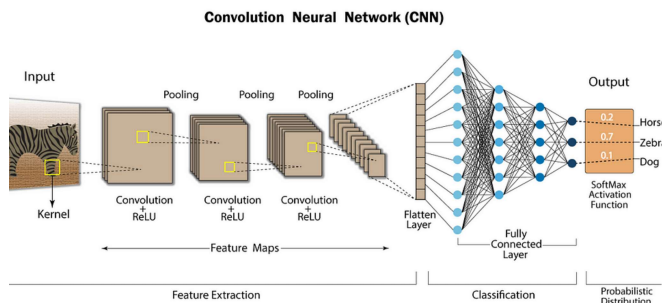


1



Shivam Baldha

Binary Classification with PyTorch



Koushik

Understanding Convolutional Neural Networks (CNNs) in Depth

In the realm of machine learning, binary classification is a fundamental task that...

5 min read · Oct 1, 2023



Convolutional Neural Networks skillfully capturing and extracting patterns from data,...

12 min read · Nov 28, 2023



See more recommendations