Activation Functions and their Derivatives – A Quick & Complete Guide

50+ Exciting Industry Projects to become a Full-Stack Data Scientist 👨‍💻

Download Projects

Lakshmi Panneerselvam — Published On April 14, 2021

Advanced Deep Learning Maths

This article was published as a part of the Data Science Blogathon.

Introduction

In Deep learning, a neural network without an activation function is just a linear regression model as these functions actually do the non-linear computations to the input of a neural network making it capable to learn and perform more complex tasks. Thus, it is quite essential to study the derivatives and implementation of activation functions, also analyze the benefits and downsides for each activation function, for choosing the right type of activation function that could provide non-linearity and accuracy in a specific neural network model.

Table of Contents

Introduction to Activation Functions

What it actually is?

Activation functions are functions used in a neural network to compute the weighted sum of inputs and biases, which is in turn used to decide whether a neuron can be activated or not. It manipulates the presented data and produces an output for the neural network that contains the parameters in the data. The activation functions are also referred to as *transfer functions* in some literature. These can either be linear or nonlinear
depending on the function it represents and is used to control the output of neural networks across different domains.

For a linear model, a linear mapping of an input function to output is performed in the hidden layers before the final prediction for each label is given. The input vector x transformation is given by

$$f(x) = w^T \cdot x + b$$

We use cookies on Analytics Vidhya websites to deliver our services, analyze web traffic, and improve your experience on the site. By using Analytics Vidhya, you agree to our Privacy Policy and Terms of Use. Accept

to convert these linear outputs into non-linear output for further computation, and then to learn the patterns in the data. The output of these models are given by

$y = (w_1 x_1 + w_2 x_2 + ... + w_n x_n + b)$

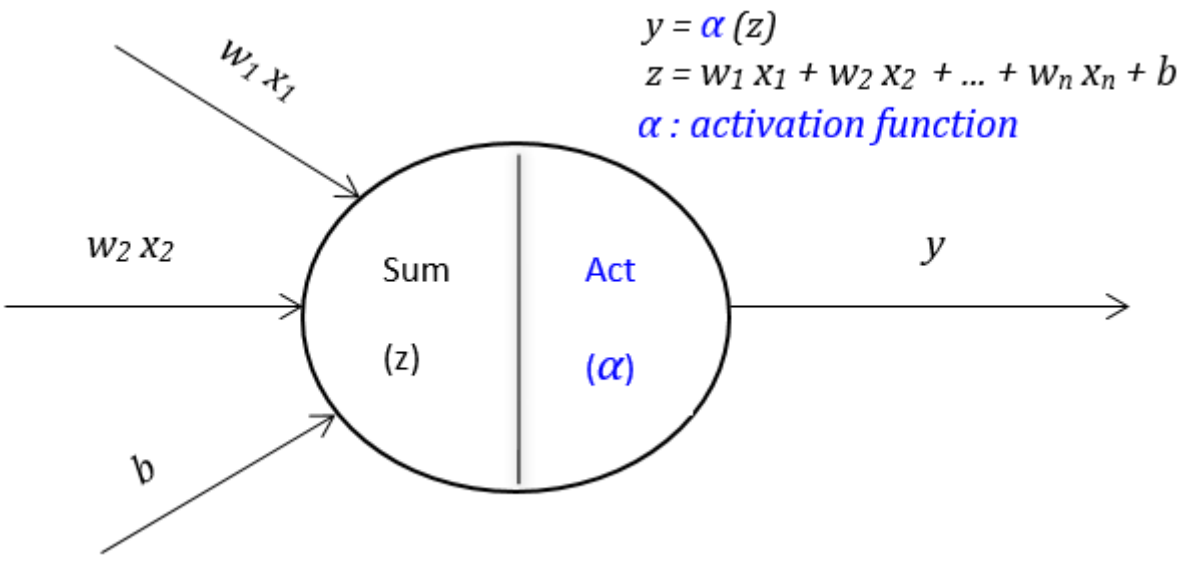
These outputs of each layer are fed into the next subsequent layer for multilayered networks until the final output is obtained, but they are linear by default. The expected output is said to determine the type of activation function that has to be deployed in a given network.

However, since the outputs are linear in nature, the nonlinear activation functions are required to convert these linear inputs to non-linear outputs. These transfer functions, applied to the outputs of the linear models to produce the transformed non-linear outputs are ready for further processing. The non-linear output after the application of the activation function is given by

$y = \alpha\ (w_1 x_1 + w_2 x_2 + ... + w_n x_n + b)$

where α is the activation function.

Why Activation Functions?



The need for these activation functions includes converting the linear input signals and models into non-linear output signals, which aids the learning of high order polynomials for deeper networks.

How to use it?

In a neural network every neuron will do two computations:

- *Linear summation of inputs:* In the above diagram, it has two inputs x_1, x_2 with weights $w_1, w_2,$ and bias b. And the linear sum $z = w_1 x_1 + w_2 x_2 + ... + w_n x_n + b$
- *Activation computation:* This computation decides, whether a neuron should be activated or not, by calculating the weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

Most neural networks begin by computing the weighted sum of the inputs. Each node in the layer can have its own unique weighting. However, the activation function is the same across all nodes in the layer. They are typical of a fixed form whereas the weights are considered to be the learning parameters.

What is a Good Activation Function?

We use cookies on Analytics Vidhya websites to deliver our services, analyze web traffic, and improve your experience on the site. By using Analytics Vidhya, you agree to our Privacy Policy and Terms of Use. Accept

activation functions must be monotonic, differentiable, and quickly converging with respect to the weights for optimization purposes.

Types of Activation Functions

The different kinds of activation functions include:

1) Linear Activation Functions

A linear function is also known as a straight-line function where the activation is proportional to the input i.e. the weighted sum from neurons. It has a simple function with the equation:

$f(x) = ax + c$

The problem with this activation is that it cannot be defined in a specific range. Applying this function in all the nodes makes the activation function work like linear regression. The final layer of the Neural Network will be working as a linear function of the first layer. Another issue is the gradient descent when differentiation is done, it has a constant output which is not good because during backpropagation the rate of change of error is constant that can ruin the output and the logic of backpropagation.

2) Non-Linear Activation Functions

The non-linear functions are known to be the most used activation functions. It makes it easy for a neural network model to adapt with a variety of data and to differentiate between the outcomes.

These functions are mainly divided basis on their range or curves:

a) Sigmoid Activation Functions

Sigmoid takes a real value as the input and outputs another value between 0 and 1. The sigmoid activation function translates the input ranged in $(-\infty,\infty)$ to the range in (0,1)

b) Tanh Activation Functions

The tanh function is just another possible function that can be used as a non-linear activation function between layers of a neural network. It shares a few things in common with the sigmoid activation function. Unlike a sigmoid function that will map input values between 0 and 1, the Tanh will map values between -1 and 1. Similar to the sigmoid function, one of the interesting properties of the tanh function is that the derivative of tanh can be expressed in terms of the function itself.

c) ReLU Activation Functions

The formula is deceptively simple: *max(0,z)*. Despite its name, Rectified Linear Units, it's not linear and provides the same benefits as Sigmoid but with better performance.

(i) Leaky Relu

Leaky Relu is a variant of ReLU. Instead of being 0 when $z<0$, a leaky ReLU allows a small, non-zero, constant gradient α (normally, $\alpha=0.01$). However, the consistency of the benefit across tasks is presently unclear. Leaky ReLUs attempt to fix the "dying ReLU" problem.

(ii) Parametric Relu

We use cookies on Analytics Vidhya websites to deliver our services, analyze web traffic, and improve your experience on the site. By using Analytics Vidhya, you agree to our Privacy Policy and Terms of Use. Accept

The Maxout activation is a generalization of the ReLU and the leaky ReLU functions. It is a piecewise linear function that returns the maximum of inputs, designed to be used in conjunction with the dropout regularization technique. Both ReLU and leaky ReLU are special cases of Maxout. The Maxout neuron, therefore, enjoys all the benefits of a ReLU unit and does not have any drawbacks like dying ReLU. However, it doubles the total number of parameters for each neuron, and hence, a higher total number of parameters need to be trained.

e) ELU

The Exponential Linear Unit or ELU is a function that tends to converge faster and produce more accurate results. Unlike other activation functions, ELU has an extra alpha constant which should be a positive number. ELU is very similar to ReLU except for negative inputs. They are both in the identity function form for non-negative inputs. On the other hand, ELU becomes smooth slowly until its output equal to -α whereas ReLU sharply smoothes.

f) Softmax Activation Functions

Softmax function calculates the probabilities distribution of the event over 'n' different events. In a general way, this function will calculate the probabilities of each target class over all possible target classes. Later the calculated probabilities will help determine the target class for the given inputs.

When to use which Activation Function in a Neural Network?

Specifically, it depends on the problem type and the value range of the expected output. For example, to predict values that are larger than 1, tanh or sigmoid are not suitable to be used in the output layer, instead, ReLU can be used. On the other hand, if the output values have to be in the range (0,1) or (-1, 1) then ReLU is not a good choice, and sigmoid or tanh can be used here. While performing a classification task and using the neural network to predict a probability distribution over the mutually exclusive class labels, the softmax activation function should be used in the last layer. However, regarding the hidden layers, as a rule of thumb, use ReLU as an activation for these layers.

In the case of a binary classifier, the Sigmoid activation function should be used. The sigmoid activation function and the tanh activation function work terribly for the hidden layer. For hidden layers, ReLU or its better version leaky ReLU should be used. For a multiclass classifier, Softmax is the best-used activation function. Though there are more activation functions known, these are known to be the most used activation functions.

Activation Functions and their Derivatives

We use cookies on Analytics Vidhya websites to deliver our services, analyze web traffic, and improve your experience on the site. By using Analytics Vidhya, you agree to our Privacy Policy and Terms of Use. Accept

Linear	$f(x) = ax + c$	$f'(x) = a$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)\,(1 - f(x))$
TanH	$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ReLU	$f(x) = \begin{cases} 0 \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$
Parametric ReLU	$f(x) = \begin{cases} \alpha x \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$
ELU	$f(x) = \begin{cases} \alpha(e^x - 1) \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$

Implementation using Python

Having learned the types and significance of each activation function, it is also essential to implement some basic (non-linear) activation

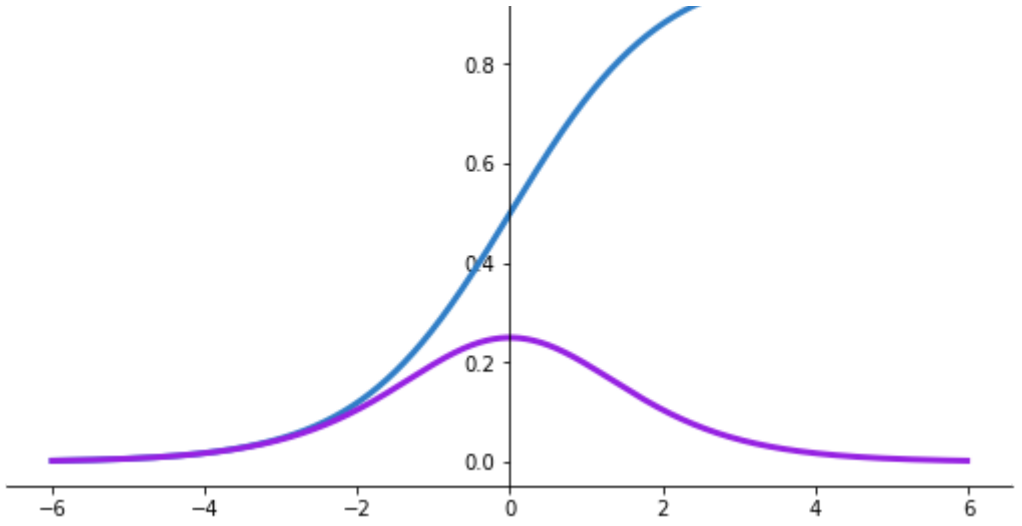
functions using python code and observe the output for more clear understanding of the concepts:

Sigmoid Activation Function

```
import matplotlib.pyplot as plt
import numpy as np
def sigmoid(x):
    s=1/(1+np.exp(-x))
    ds=s*(1-s)
    return s,ds
x=np.arange(-6,6,0.01)
sigmoid(x)
fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
ax.plot(x,sigmoid(x)[0], color="#307EC7", linewidth=3, label="sigmoid")
ax.plot(x,sigmoid(x)[1], color="#9621E2", linewidth=3, label="derivative")
ax.legend(loc="upper right", frameon=False)
fig.show()
```

We use cookies on Analytics Vidhya websites to deliver our services, analyze web traffic, and improve your experience on the site. By using Analytics Vidhya, you agree to our Privacy Policy and Terms of Use. Accept

Activation Functions and their Derivatives – A Quick & Complete Guide

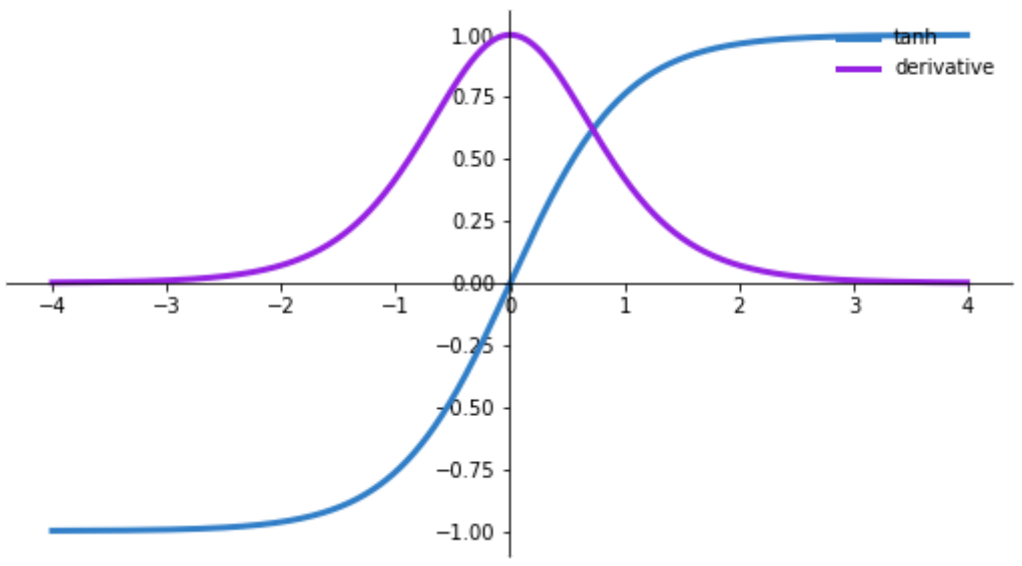


Observations:

- The sigmoid function has values between 0 to 1.
- The output is not zero-centered.
- Sigmoids saturate and kill gradients.
- At the top and bottom level of sigmoid functions, the curve changes slowly, the derivative curve above shows that the slope or gradient it is zero.

Tanh Activation Function

```
import matplotlib.pyplot as plt
import numpy as np
def tanh(x):
    t=(np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
    dt=1-t**2
    return t,dt
z=np.arange(-4,4,0.01)
tanh(z)[0].size,tanh(z)[1].size
fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position('center')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
ax.plot(z,tanh(z)[0], color="#307EC7", linewidth=3, label="tanh")
ax.plot(z,tanh(z)[1], color="#9621E2", linewidth=3, label="derivative")
ax.legend(loc="upper right", frameon=False)
fig.show()
```



We use cookies on Analytics Vidhya websites to deliver our services, analyze web traffic, and improve your experience on the site. By using Analytics Vidhya, you agree to our Privacy Policy and Terms of Use. Accept

- Optimization is easier in this method hence in practice it is always preferred over the Sigmoid function.

Pros and Cons of Activation Functions

Type of Function	*Pros*	*Cons*
Linear	• It gives a range of activations, so it is not binary activation. • It can definitely connect a few neurons together and if more than 1 fire, take the max and decide based on that.	• It is a constant gradient and the descent is going to be on a constant gradient. • If there is an error in prediction, the changes made by backpropagation are constant and not depending on the change in input.
Sigmoid	• It is nonlinear in nature. Combinations of this function are also nonlinear. • It will give an analog activation, unlike the step function.	• Sigmoids saturate and kill gradients. It gives rise to a problem of "vanishing gradients" • The network refuses to learn further or is drastically slow.
Tanh	• The gradient is stronger for tanh than sigmoid i.e. derivatives are steeper.	• Tanh also has a vanishing gradient problem.
ReLU	• It avoids and rectifies the vanishing gradient problem. • ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations.	• It should only be used within hidden layers of a Neural Network Model. • Some gradients can be fragile during training and can die. It can cause a weight update which will make it never activate on any data point again. Thus, ReLu could even result in Dead Neurons.
Leaky ReLU	• Leaky ReLUs is one attempt to fix the "dying ReLU" problem by having a small negative slope	• As it possesses linearity, it can't be used for complex Classification. It lags behind the Sigmoid and Tanh for some of the use cases.
ELU	• Unlike ReLU, ELU can produce negative outputs.	• For x>0, it can blow up the activation with the output range of [0,

We use cookies on Analytics Vidhya websites to deliver our services, analyze web traffic, and improve your experience on the site. By using Analytics Vidhya, you agree to our Privacy Policy and Terms of Use. Accept

to the function, which could be a neural network. A neural network, without the activation functions, might perform solely linear mappings from the inputs to the outputs, and also the mathematical operation throughout the forward propagation would be the dot-products between an input vector and a weight matrix.

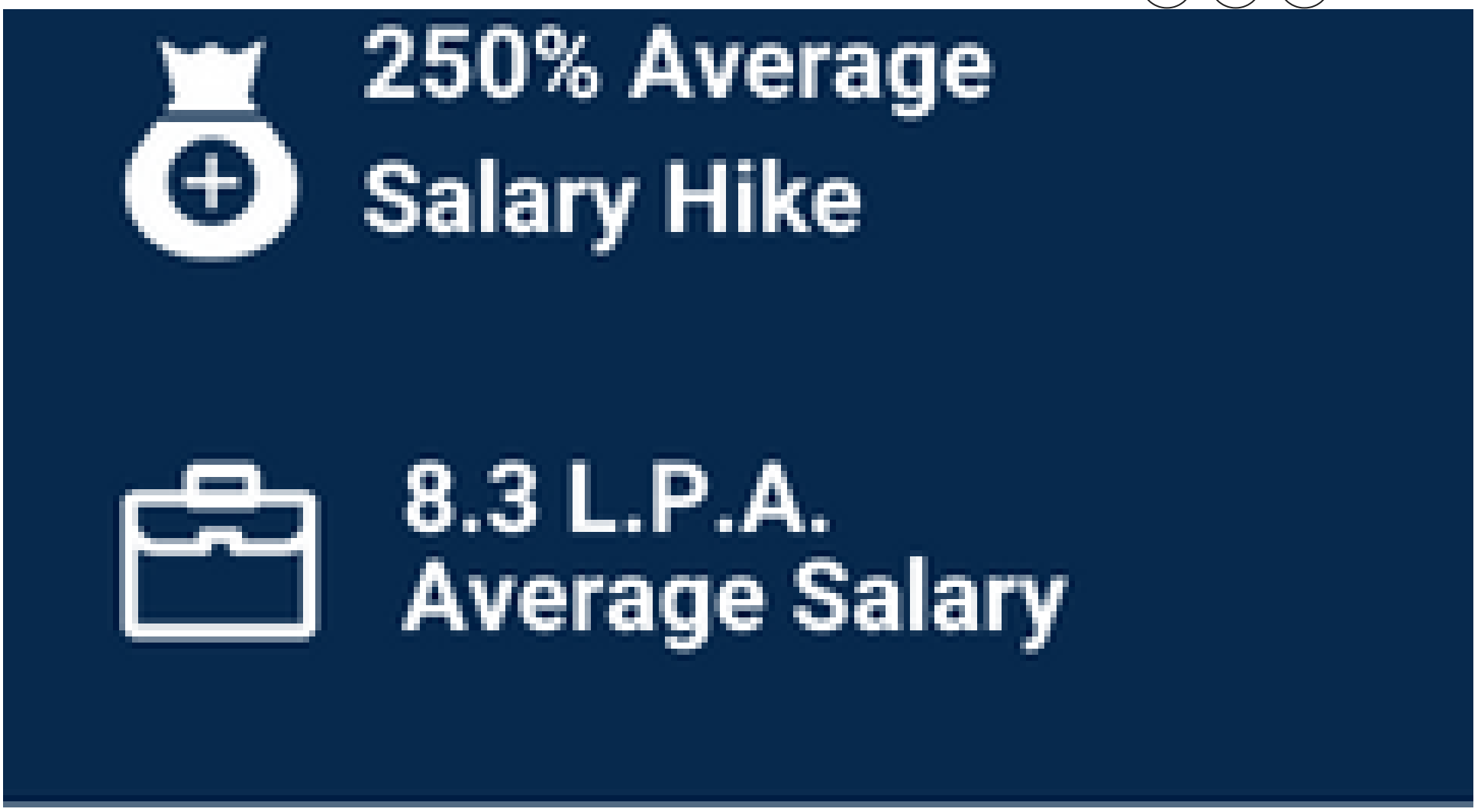
Since one dot product could be a linear operation, sequent dot products would be nothing more than multiple linear operations repeated one after another. And sequent linear operations may be thought of as mutually single learn operations. To be able to work out extremely attention-grabbing stuff, the neural networks should be able to approximate the nonlinear relations from input features to the output labels.

The more complicated the information, the more non-linear the mapping of features to the bottom truth label will usually be. If there is no activation function in a neural network, the network would in turn not be able to understand such complicated mappings mathematically and wouldn't be able to solve tasks that the network is really meant to resolve.

The media shown in this article are not owned by Analytics Vidhya and is used at the Author's discretion.

activation function blogathon

We use cookies on Analytics Vidhya websites to deliver our services, analyze web traffic, and improve your experience on the site. By using Analytics Vidhya, you agree to our Privacy Policy and Terms of Use. Accept

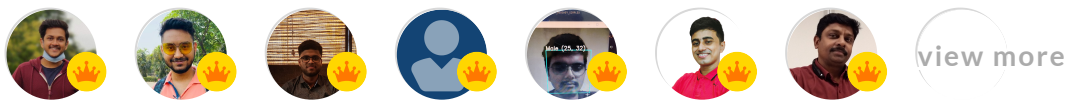


Enroll Now

About the Author

Lakshmi Panneerselvam

Our Top Authors



Download

Analytics Vidhya App for the Latest blog/Article

Previous Post

Machine Learning in Building Automation

Next Post

How Aviation Industry Uses Data Science?

We use cookies on Analytics Vidhya websites to deliver our services, analyze web traffic, and improve your experience on the site. By using Analytics Vidhya, you agree to our Privacy Policy and Terms of Use. Accept

Activation Functions and their Derivatives – A Quick & Complete Guide

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

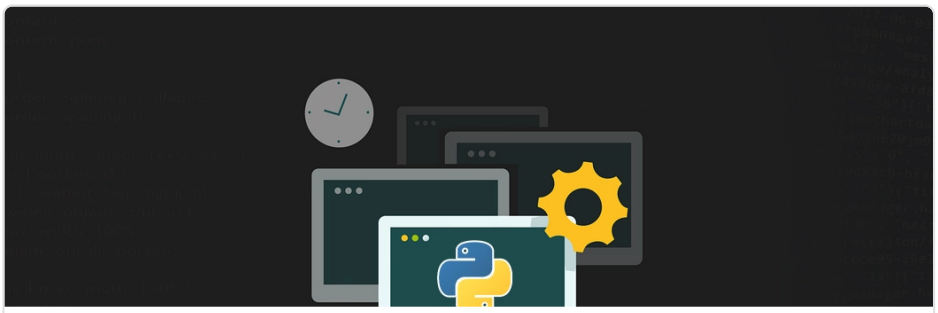
Website

☑ Notify me of follow-up comments by email.

☑ Notify me of new posts by email.

Submit

Top Resources



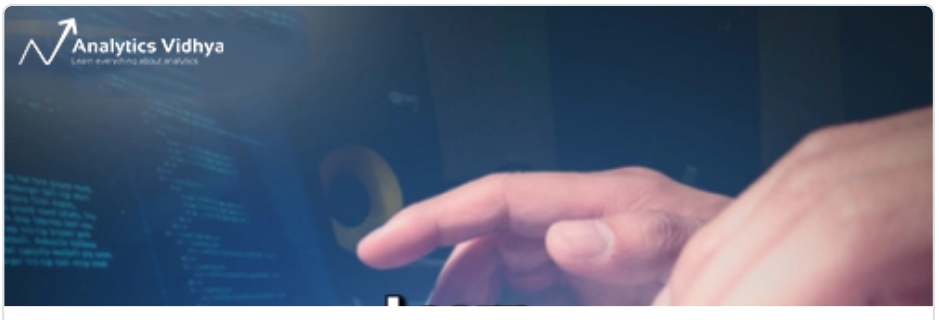
Python Tutorial: Working with CSV file for Data Science

Harika Bonthu - AUG 21, 2021

The Most Comprehensive Guide to K-Means Clustering You'll Ever Need

Pulkit Sharma - AUG 19, 2019

Understanding Random Forest



Understanding Support Vector Machine(SVM) algorithm from examples (along with code)

We use cookies on Analytics Vidhya websites to deliver our services, analyze web traffic, and improve your experience on the site. By using Analytics Vidhya, you agree to our Privacy Policy and Terms of Use. Accept

Activation Functions and their Derivatives – A Quick & Complete Guide

Download App

Analytics Vidhya

About Us

Our Team

Careers

Contact us

Companies

Post Jobs

Trainings

Hiring Hackathons

Advertising

Data Scientists

Blog

Hackathon

Discussions

Apply Jobs

Visit us

© Copyright 2013-2022 Analytics Vidhya.

Privacy Policy Terms of Use Refund Policy

We use cookies on Analytics Vidhya websites to deliver our services, analyze web traffic, and improve your experience on the site. By using Analytics Vidhya, you agree to our Privacy Policy and Terms of Use. Accept