ALL about Activation function

Table of contents


```
# import numpy library
```

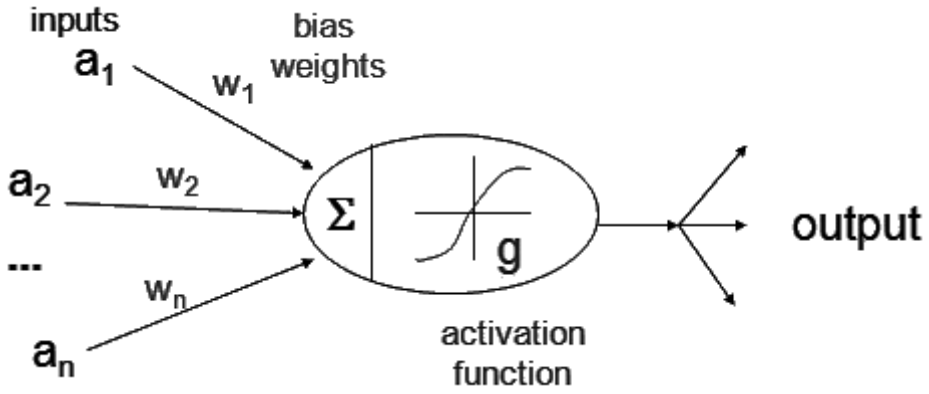
What Is Activation Function?

An activation function is a very important feature of an artificial neural network , they basically decide whether the neuron should be activated or not.

In artificial neural networks, the activation function defines the output of that node given an input or set of inputs.

Important

Use of any activation function is to introduce non-linear properties to our Network.



From above image:

It calculates a "sum of weights" of its input and then adds a bias to it then decides whether the neuron should be "fired" or not.

All inputs X1,X2, X3... Xn are multiplied with the weights W1, W2, W3 ... Wn assigned to each link between inputand node and then summed together along with Bias b

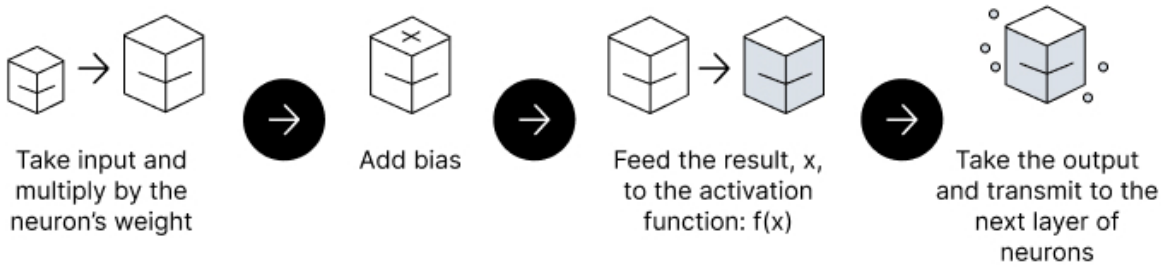
y = summation((Wi*Xi) + b)

Note:

Xi's and Wi's are vectors and b is scalar.

The value of Y can be anything ranging from -inf to +inf. Meaning it has lot of information ,now neuron must know to distinguish between the "useful" and "not -so-useful" information.To build this sense into our network we add 'activation function (f)'.

The activation Function will decide whether the information passed is useful or not based on the result it get fired.

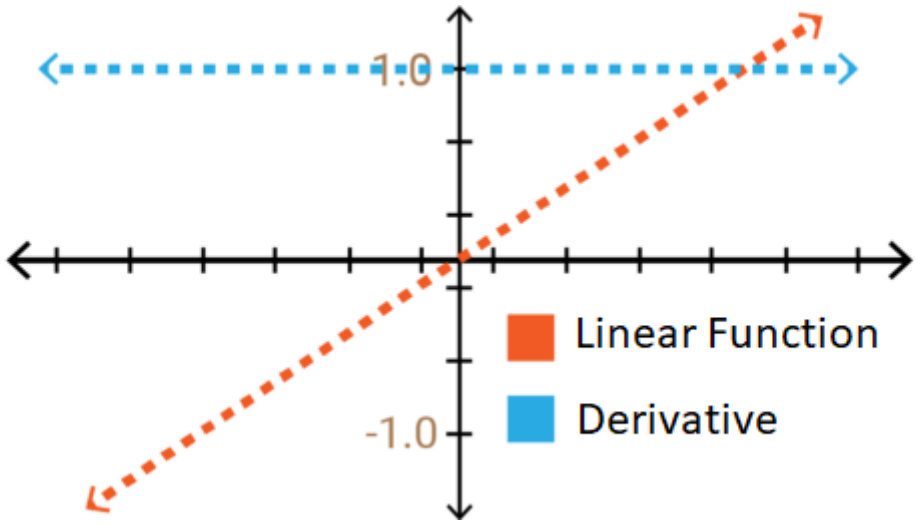


V7 Labs

Activation function Types :

- Linear function
- Binary Step function
- Non-Linear function

Linear Function:



A linear activation function takes the form:

y=mx+c (m is line equation represents W and c is represented as b in neural nets so equation can be modified as y=Wx+b)

It takes the inputs (Xi's), multiplied by the weights(Wi's) for each neuron, and creates an output proportional to the input. In simple term, weighted sum input is proportional to output.

Problem with Linear function

1. We performed the learning process for neurons with the backpropagation algorithm. This algorithm consists of a derivative system. When y = m.x is derived from x, we reach m.

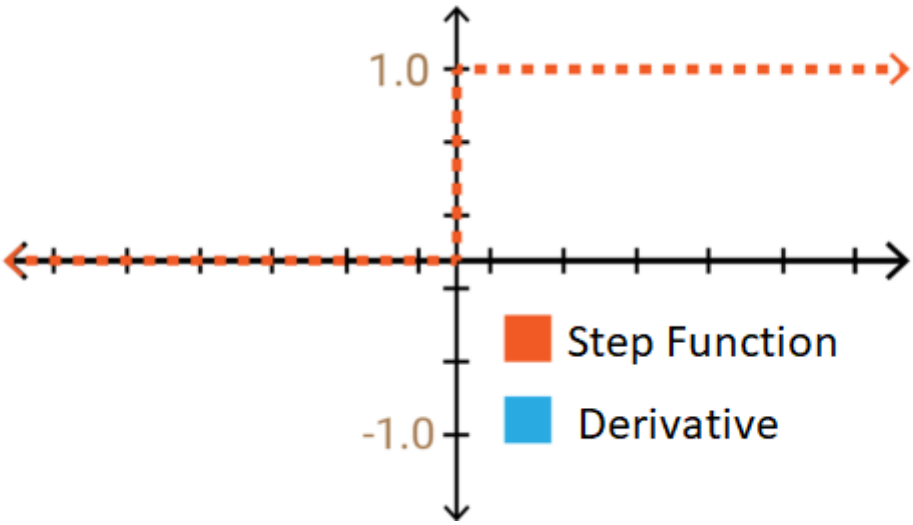
This means that there is no relationship with x. It means that the derivative is always a constant value.

So how can we say that the learning process is taking place. The answer is No!

2. linear activation functions, no matter how many layers in the neural network, the last layer will be a linear function of the first layer — Meaning Output of the first layer is same as the output of the nth layer. 😐

A neural networks with a linear activation function is simply a linear regression model.

Binary Step Function:



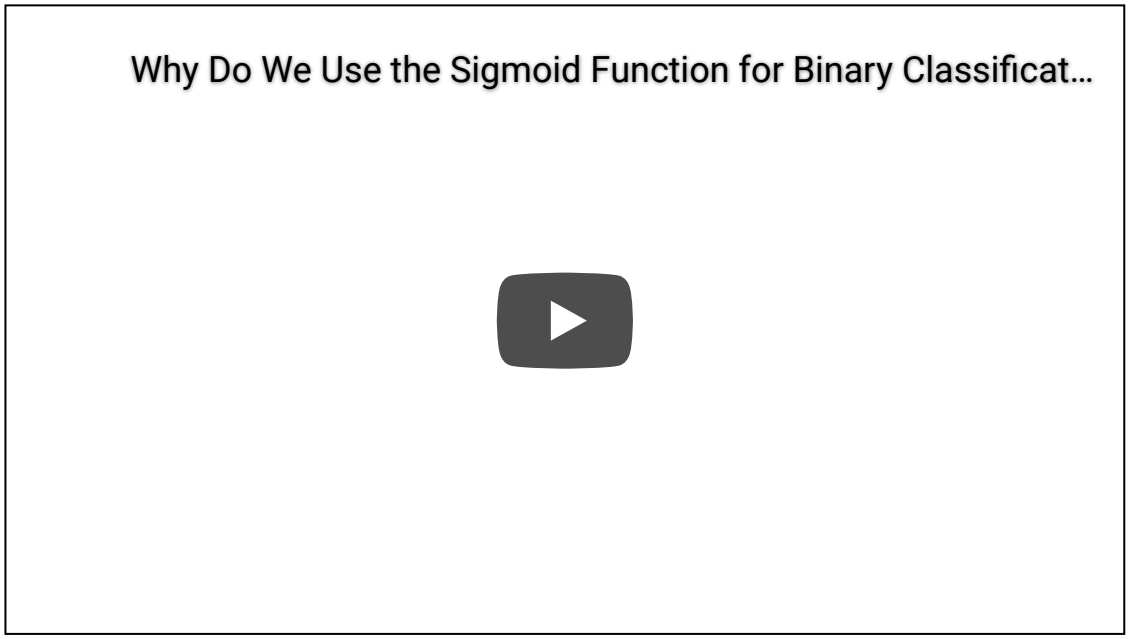
It is a function that takes a binary value and is used as a binary classifier. Binary step function are popular known as " Threshold function". It is very simple function.

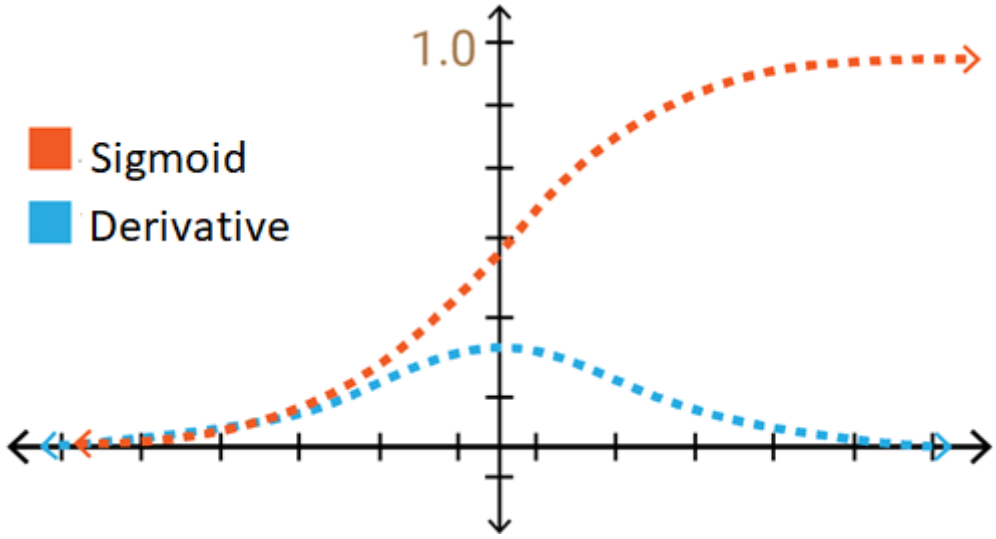
Therefore, it is generally preferred in the output layers. It is not recommended to use it in hidden layers because it does not represent derivative learning value and it will not appear in the future.

Non-Linear function:

Sigmoid function:

```
# just run these cell
# this cell is just for yputube video display in notebook
from IPython.display import HTML

# Youtube
HTML('<iframe width="560" height="315" src="https://www.youtube.com/embed/WsFasV46KgQ" tit
```





Most modern neural network use the non-linear function as their activation function to fire the neuron. It is also derivated because it is different from the step function tha means that learning of neural network can happen.

If we examine the graph x is between -2 and +2, y values change quickly. Small changes in x will be large in y. This means that it can be used as a good classifier.

Another advantage of this function is that it produces a value in the range of (0,1) when encountered with (- infinite, + infinite) as in the linear function. So the activation value does not vanish, this is good news! 😀

Formula

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

```
# define a function named sigmoid with parameter x

  # return the calculated value by the formula 1/(1+e^(-x))


# create a variable named sigmoid_value and store value returned by above sigmoid function

# print the sigmoid_value
```

```
    0.9999999999999065
```

Oop's there is a problem with sigmoid 😑

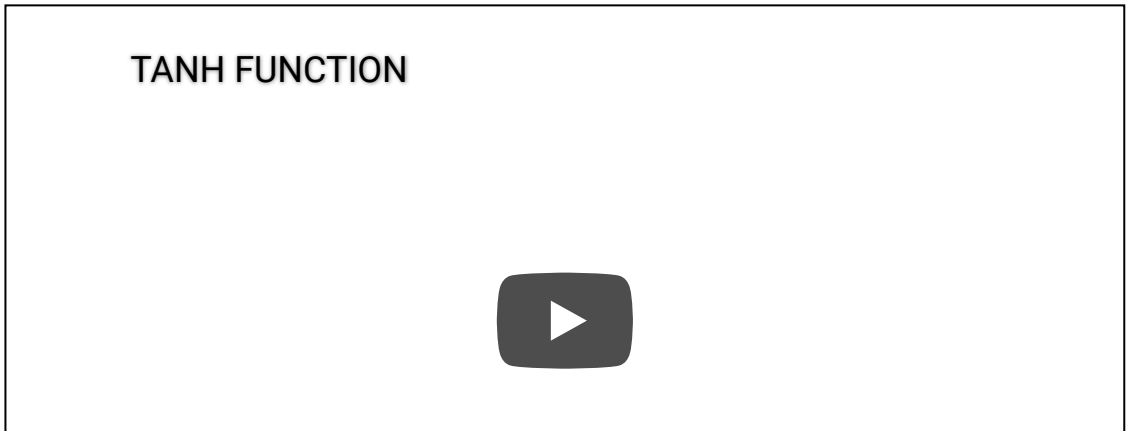
If we look carefully at the graph towards the ends of the function, y values react very little to the changes in x.

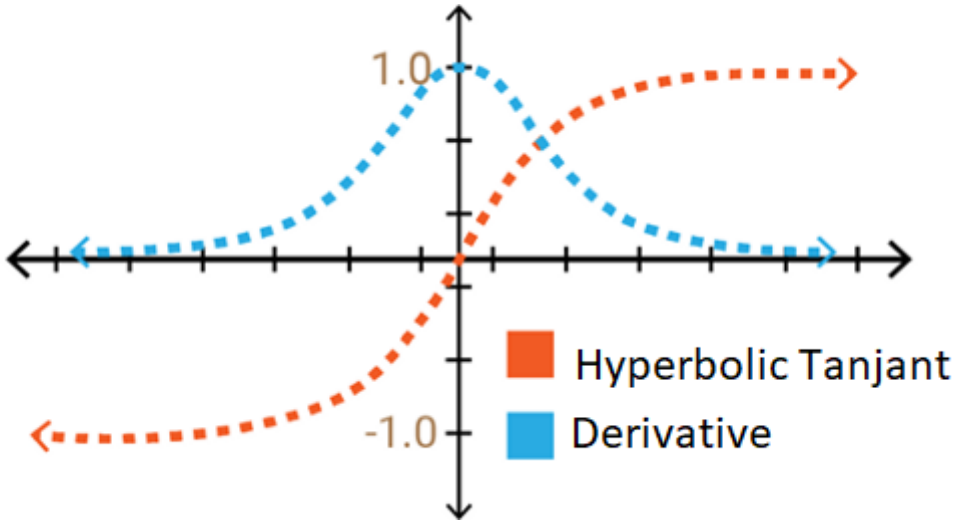
The derivative values in these regions are very small and converge to 0. This is called the "vanishing gradient" and the learning is minimal.

if 0, not any learning! When slow learning occurs, the optimization algorithm that minimizes error can be attached to local minimum values and cannot get maximum performance from the artificial neural network model

Hyperbolic Tangent Function (Tanh):

```
# just run these cell
# this cell is just for yputube video display in notebook

# Youtube
HTML('<iframe width="560" height="315" src="https://www.youtube.com/embed/IbeJl3EIJYg" tit
```





It has a structure very similar to Sigmoid function. However, this time the function is defined as (-1, + 1). The advantage over the sigmoid function is that its derivative is more steep, which means it can get more value. This means that it will be more efficient because it has a wider range for faster learning and grading. But again, the problem of gradients at the ends of the function continues

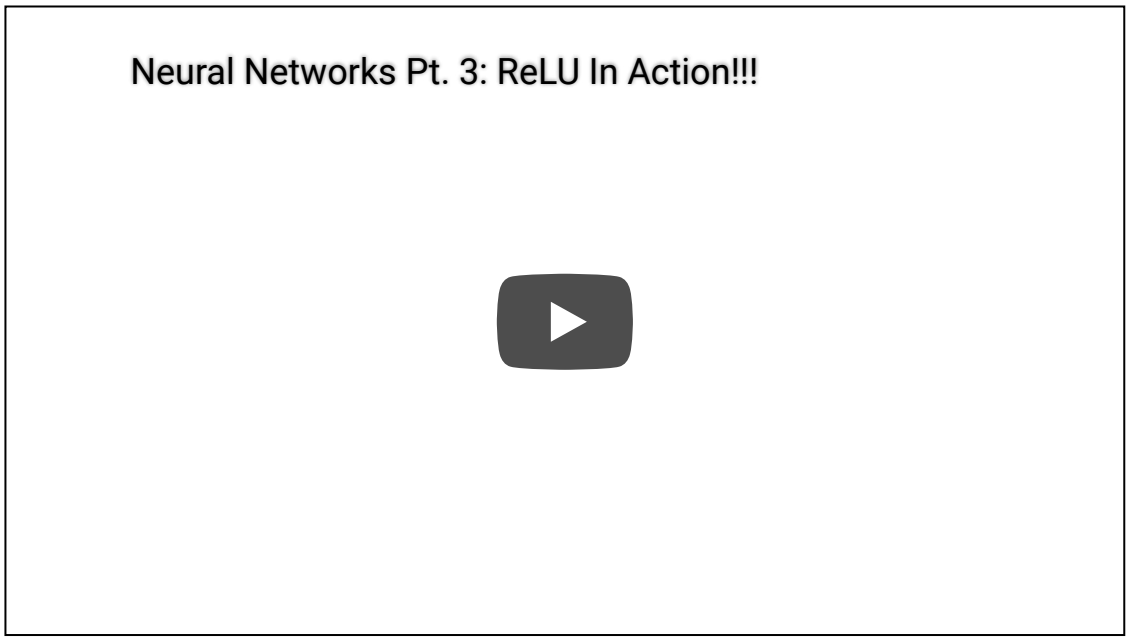
```
# define a function named tanh with parameter x

  # return the calculated value by the formula tanh(x)



# create a variable named tanh_value and store value returned by above tanh function after

# print the sigmoid_value
```

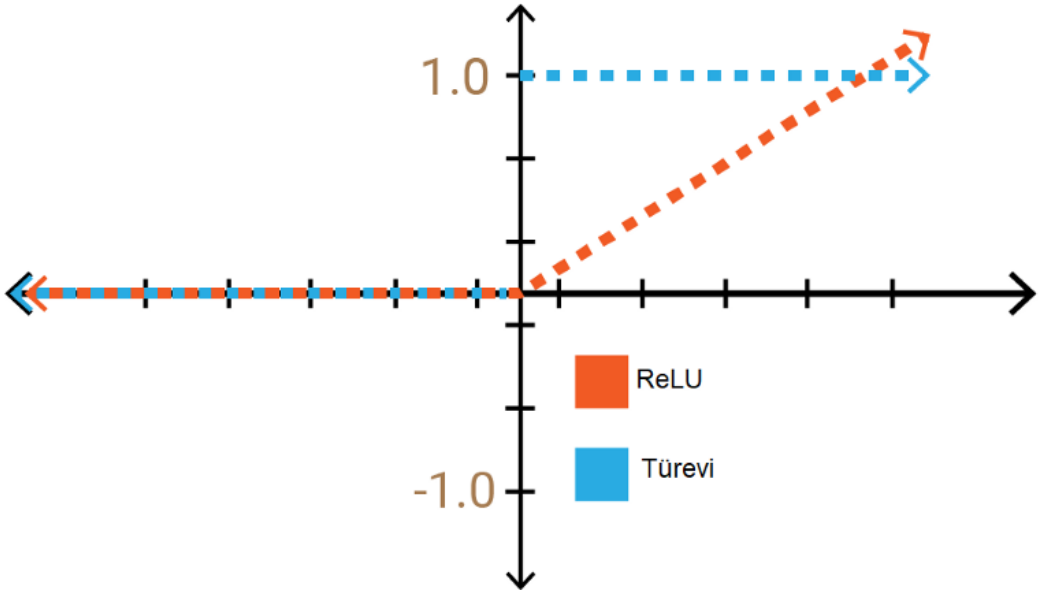
```
    1.0
```

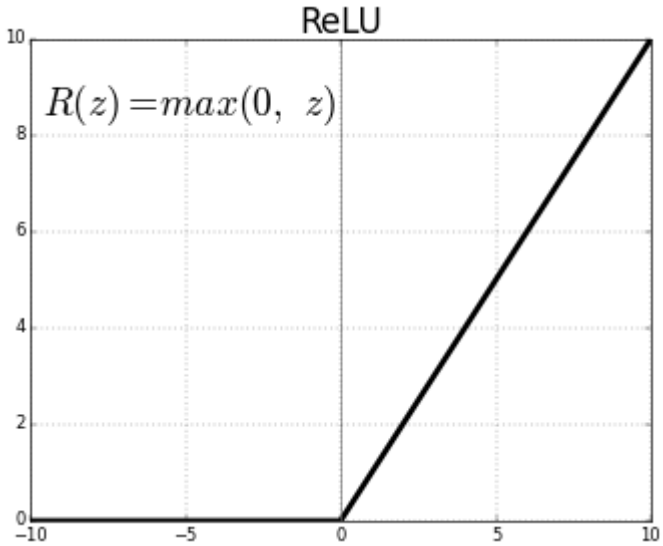
ReLu Activation Function (ReLu — Rectified Linear Units)

```
# just run these cell
# this cell is just for yputube video display in notebook
```

```
# Youtube
HTML('<iframe width="560" height="315" src="https://www.youtube.com/embed/68BZ5f7P94E" tit
```







The ReLU is the most used activation function in the world right now.Since, it is used in almost all the convolutional neural networks or deep learning.

As you can see, the ReLU is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.

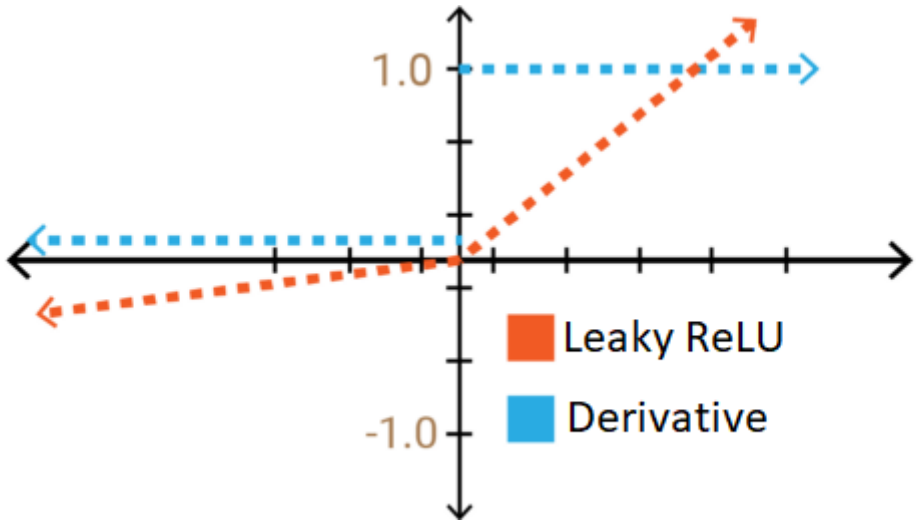
Range: [0 to infinity]

Main advantage of using the ReLU function- It does not activate all the neurons at the same time. It converges very fast It is computaionally efficient

```
# define a function named relu with parameter x

  # return the maximum values in x greater than zero i.e All positive numbers


# create a variable named relu_value and store value returned by above relu function after

# print relu_value
```

```
    0
```

leaky ReLu Activation Function:



Leaky ReLU function is nothing but an improved version of the ReLU function with introduction of "constant slope".

It has a small slope for negative values instead of a flat slope

Leaky ReLU is defined to address problem of dying neuron/dead neuron.

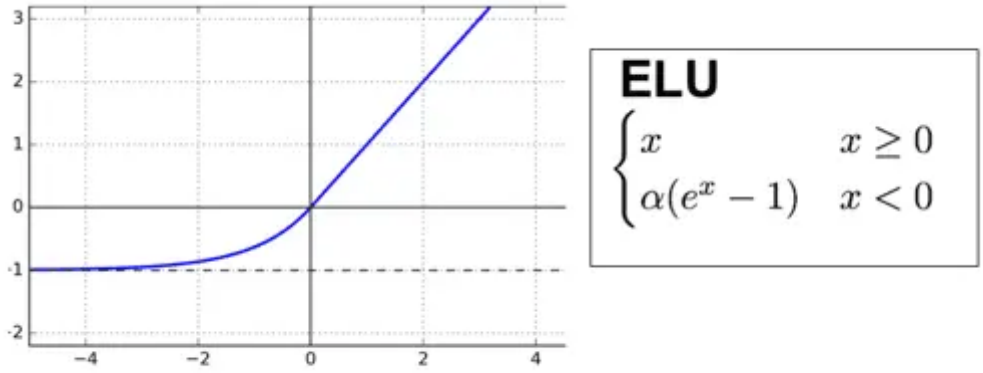
It allows negative value during back propagation

Note:

Leaky ReLU does not provide consistent predictions for negative input values.

```
# define a function named relu with parameter x

  # return the maximum values between x and 0.01*x


# create a variable named leaky_relu_value and store value returned by above leakyRelu fun

# print leaky_relu_value


     -0.1
```

ELU (Exponential Linear Units) Activation Function:



ELU is very similiar to RELU except negative inputs. They are both in 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.

It is a function that tend to converge cost to zero faster and produce more accurate results

Note:

With values higher values above 0 the Elu function blows up the activation beacuse the number can lie between [0,infinity]

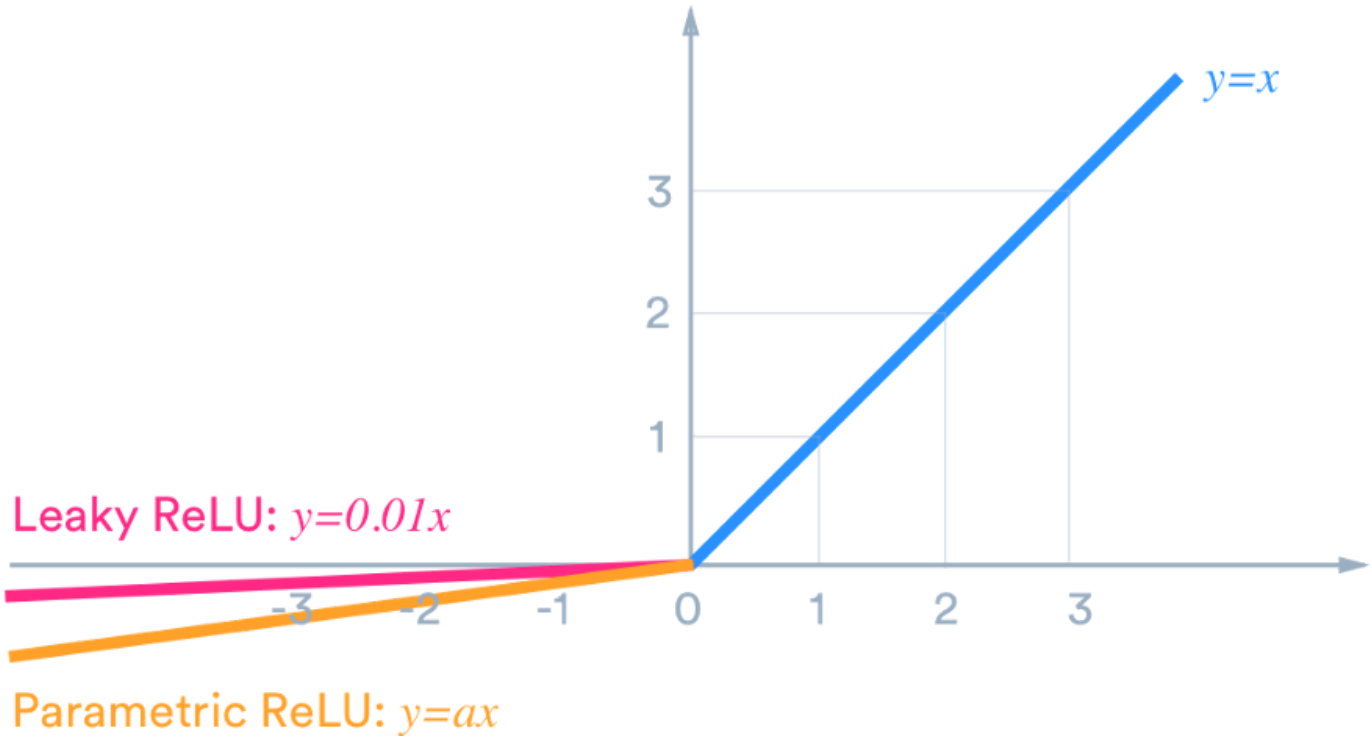
Similar to Leaky ReLU, although theoretically better than ReLU, there is currently no good evidence in practice that ELU is always better than ReLU.

```
# define a function named elu with parameter x

  # return the maximum values between x and 0.01*((e^x) - 1)


# create a variable named elu_value and store value returned by above elu function after p
```

```
# print elu_value
```

```
    -0.009999546000702375
```

P ReLu (Parametric ReLU) Activation Function:



A Parametric Rectified Linear Unit, or PReLU, is an activation function that generalizes the traditional rectified unit with a slope for negative values.

Whenever the x value will be greater than zero itwill give a output value as x, but whenever the x value is below zero it will give the value as aplha*x

Pay Attention

1. If alpha value is 0.01 then Prelu function will become Leaky Relu function
2. If alpha value is 0 then Prelu will become Relu function
3. also aplha can be any value we want.

Thus the alpha here is named as "learning parameter" which will dynamicaly change based on the this particularactivation function.

Therefore depending on the parameter the function changes to Relu, Leaky Relu, Elu, etc

```
# define a function named PRelu with parameter x, alpha
```

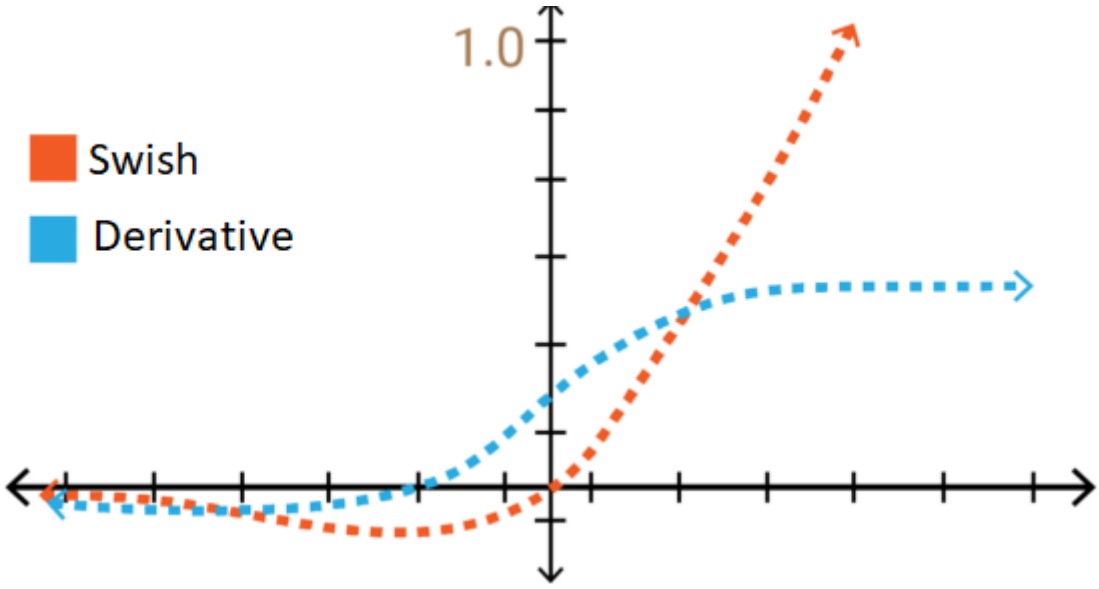
```
  # return the maximum values between x and alpha*x
```

```
# create a variable named p_relu_value and store value returned by above PRelu function af

# print p_relu_value
```

```
-0.1
```

Swish (A Self-Gated) Activation Function:(Sigmoid Linear Unit)

```
HTML('<iframe width="560" height="315" src="https://www.youtube.com/embed/2BCxzWji1rQ" tit
```



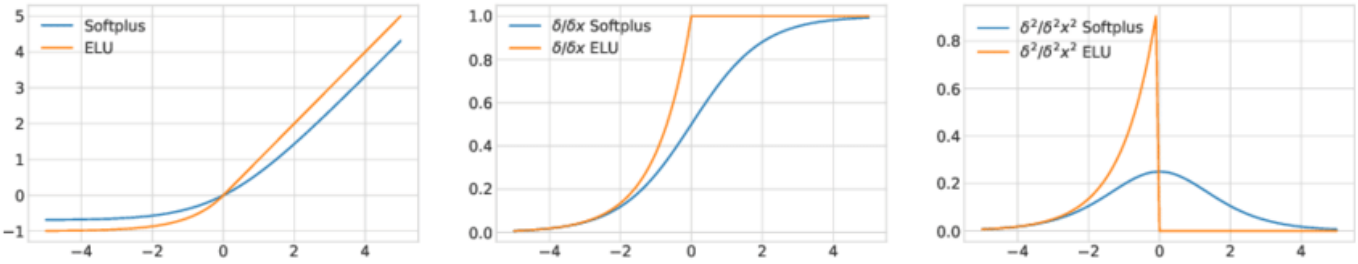
formula:

y = x * sigmoid(x)

Swish is a smooth continuous function, unlike ReLU which is a piecewise linear function. Swish allows a small number of negative weights to be propagated through, while ReLU thresholds all negative weights to zero. This is an extremely important property and is crucial in the success of non-monotonic smooth activation functions, like that of Swish, when used in increasingly deep neural networks. Lastly, the trainable parameter allows to better tune the activation function to maximize information propagation and push for smoother gradients, which makes the landscape easier to optimize, thus generalizing better and faster. Swish is also a self-gating activation function since it modulates the input by using it as a gate to multiply with the sigmoid of itself, a concept first introduced in Long Short Term Memory (LSTMs)

```
# define a function named swish with parameter x

  # return the calculated value by the formula x * 1/(1+e^(-x))


# create a variable named swish_value and store value returned by above swish function aft

# print swish_value


    29.999999999997197
```

Softplus



Formula:

y = ln(1+e^x)

Softplus activation function is a smooth version of ReLU. The name ‘softplus’ is used because of the smoothed or softened version of ReLU.

There is no need to find derivative of 0 as compared to Relu

It enhances the stability and performance of a Deep Neural Network architecture because of smoothness and non-zero gradient.

The unboundedness in upper limit and boundedness in the lower limit helps the network to avoid saturation and induce regularization respectively.

```
# define a function named softPlus with parameter x
```

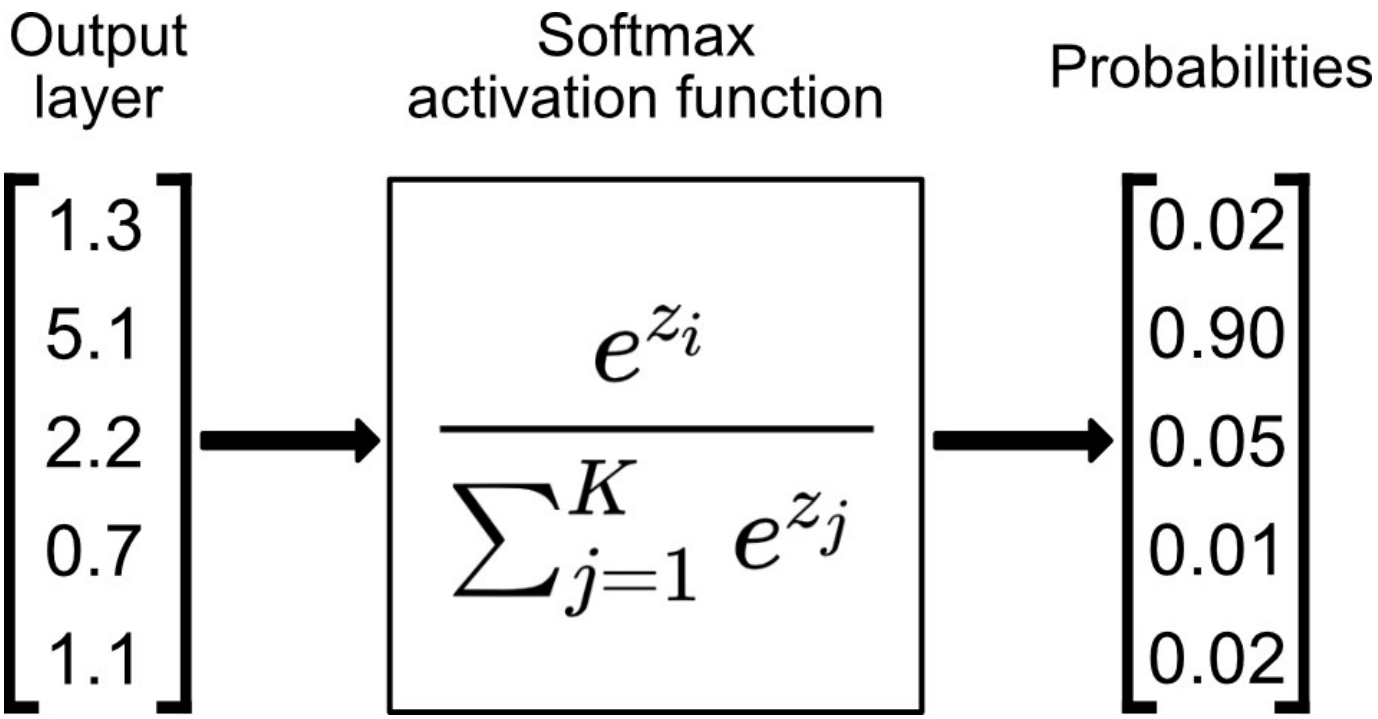
```
  # return the calculated value by the formula ln(1+e^(x))

# create a variable named softplus_value and store value returned by above swish function

# print softplus_value
```

```
    4.5398899216870535e-05
```

Softmax or normalized exponential function:

```
# just run these cell
# this cell is just for yputube video display in notebook

HTML('<iframe width="560" height="315" src="https://www.youtube.com/embed/o6HrH2EMD-w" tit
```

Formula:



Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector.

The most common use of the softmax function in applied machine learning is in its use as an activation function in a neural network model. Specifically, the network is configured to output N values, one for each class in the classification task, and the softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the softmax function is interpreted as the probability of membership for each class

```
# define a function named softmax with parameter x


  # create empty list named output

  # create a loop in range of length of input list

    # create a empty list named exp_values to store exponential values of each element in

    # create a loop in range of length of input list

      # create a variable named exp_value to store current iterated elemnts exponential va

      # append exp_value to the empty list created above named exp_values


    # Append the result of calculation e^xi/sum(e^xi) to output list created at the top
```

```
  # Convert the list to array using np.array


# create a list named input with random values and any length you want.

# create a variable named softmax_value and store value returned by above softmax function

# print softmax_value
```

```
[0.02019046 0.90253769 0.04966053 0.01108076 0.01653055]
```

The above output shows the probailities of the the input array passed to it, with highest probaility of number 5.1 as 90%

Assignment Summary

When creating a neural network we must select a apropriate activation function accordingly:

following are the activation function discussed in this assignment

1. Sigmoid
2. Tanh
3. Relu
4. Leaky Relu
5. Elu
6. PRelu
7. Swish
8. Softplus
9. Softmax

Relu is used in most of the neural network models

we have learned all the activation function used in neural network and their properties with code

Please fill the below feedback form about this assignment

https://forms.zohopublic.in/cloudyml/form/CloudyMLDeepLearningFeedbackForm/formperma/VCFbldnXAnbcgAIl0lWv2blgHdSldheO4RfktMdgK7s