Activation_Functions.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

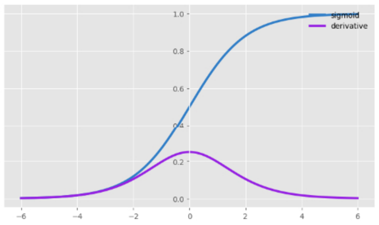
```
Start coding or generate with AI.
```

Sigmoid or Logistic Activation Function

The Sigmoid Function curve looks like a S-shape.

The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output.Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

```python
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()
```

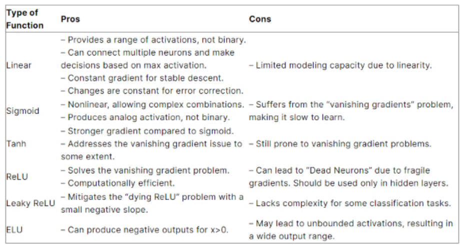


Observations:

The sigmoid function has values between 0 to 1.

The output is not zero-centered.

Sigmoids saturate and kill gradients.



Type of Function	Pros	Cons
Linear	– Provides a range of activations, not binary. – Can connect multiple neurons and make decisions based on max activation. – Constant gradient for stable descent. – Changes are constant for error correction.	– Limited modeling capacity due to linearity.
Sigmoid	– Nonlinear, allowing complex combinations. – Produces analog activation, not binary.	– Suffers from the "vanishing gradients" problem, making it slow to learn.
Tanh	– Stronger gradient compared to sigmoid. – Addresses the vanishing gradient issue to some extent.	– Still prone to vanishing gradient problems.
ReLU	– Solves the vanishing gradient problem. – Computationally efficient.	– Can lead to "Dead Neurons" due to fragile gradients. Should be used only in hidden layers.
Leaky ReLU	– Mitigates the "dying ReLU" problem with a small negative slope.	– Lacks complexity for some classification tasks.
ELU	– Can produce negative outputs for x>0.	– May lead to unbounded activations, resulting in a wide output range.

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.

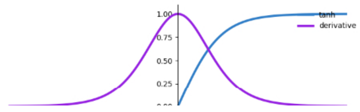
```
#Tanh Activation Function

#Tanh or hyperbolic tangent Activation Function

#tanh is also like logistic sigmoid but better. The range of the tanh function
is from (-1 to 1). tanh is also sigmoidal (s - shaped).
```

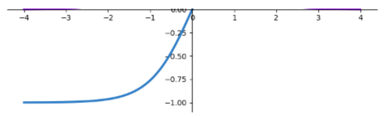
Tanh Activation Function

Tanh or hyperbolic tangent Activation Function

tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).

```python
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()
```





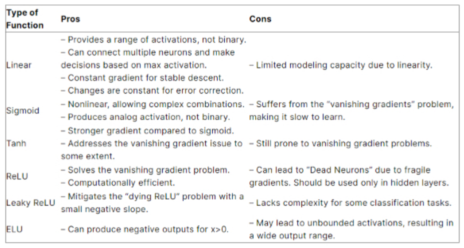
Observations:

Its output is zero-centered because its range is between -1 to 1. i.e. -1 < output < 1.

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

Double-click (or enter) to edit

Pros and Cons of Activation Functions



Type of Function	Pros	Cons
Linear	– Provides a range of activations, not binary. – Can connect multiple neurons and make decisions based on max activation. – Constant gradient for stable descent. – Changes are constant for error correction.	– Limited modeling capacity due to linearity.
Sigmoid	– Nonlinear, allowing complex combinations. – Produces analog activation, not binary.	– Suffers from the "vanishing gradients" problem, making it slow to learn.
Tanh	– Stronger gradient compared to sigmoid. – Addresses the vanishing gradient issue to some extent.	– Still prone to vanishing gradient problems.
ReLU	– Solves the vanishing gradient problem. – Computationally efficient.	– Can lead to "Dead Neurons" due to fragile gradients. Should be used only in hidden layers.
Leaky ReLU	– Mitigates the "dying ReLU" problem with a small negative slope.	– Lacks complexity for some classification tasks.
ELU	– Can produce negative outputs for x>0.	– May lead to unbounded activations, resulting in a wide output range.

Double-click (or enter) to edit

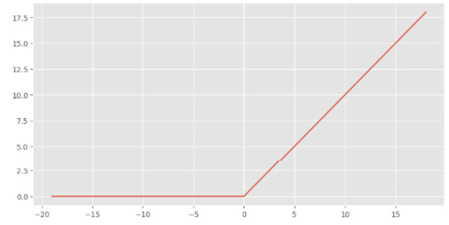
What is ReLU Activation Function?

ReLU stands for rectified linear activation unit and is considered one of the few milestones in the deep learning revolution. It is simple yet really better than its predecessor activation functions such as sigmoid or tanh.

```
[ ] Start coding or generate with AI.
```

```
[5] def ReLU(x):
      if x>0:
        return x
      else:
        return 0
```

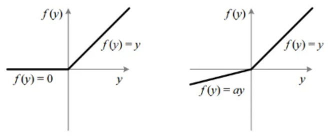
```
[6] from matplotlib import pyplot
    pyplot.style.use('ggplot')
    pyplot.figure(figsize=(10,5))
    # define a series of inputs
    input_series = [x for x in range(-19, 19)]
    # calculate outputs for our inputs
    output_series = [ReLU(x) for x in input_series]
    # line plot of raw inputs to rectified outputs
    pyplot.plot(input_series, output_series)
    pyplot.show()
```



Leaky ReLU activation function

Leaky ReLU function is an improved version of the ReLU activation function. As for the ReLU activation function, the gradient is 0 for all the values of inputs that are less than zero, which would deactivate the neurons in that region and may cause dying ReLU problem.

Leaky ReLU is defined to address this problem. Instead of defining the ReLU activation function as 0 for negative values of inputs(x), we define it as an extremely small linear component of x. Here is the formula for this activation function



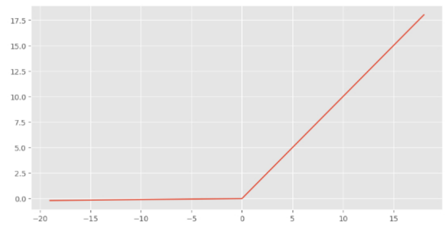
Double-click (or enter) to edit

f(x)=max(0.01*x , x).

This function returns x if it receives any positive input, but for any negative value of x, it returns a really small value which is 0.01 times x. Thus it gives an output for negative values as well. By making this small modification, the gradient of the left side of the graph comes out to be a non zero value. Hence we would no longer encounter dead neurons in that region.

Now like we did for ReLU activation function, we will give values to Leaky ReLU activation functions and plot them.

```
[7] from matplotlib import pyplot
    pyplot.style.use('ggplot')
pyplot.figure(figsize=(10,5))

# rectified linear function
def Leaky_ReLU(x):
  if x>0:
    return x
  else:
    return 0.01*x

# define a series of inputs
input_series = [x for x in range(-19, 19)]
# calculate outputs for our inputs
output_series = [Leaky_ReLU(x) for x in input_series]
# line plot of raw inputs to rectified outputs
pyplot.plot(input_series, output_series)
pyplot.show()
```



If you look carefully at the plot, you would find that the negative values are not zero and there is a slight slope to the line on the left side of the plot.

```
Start coding or generate with AI.
```