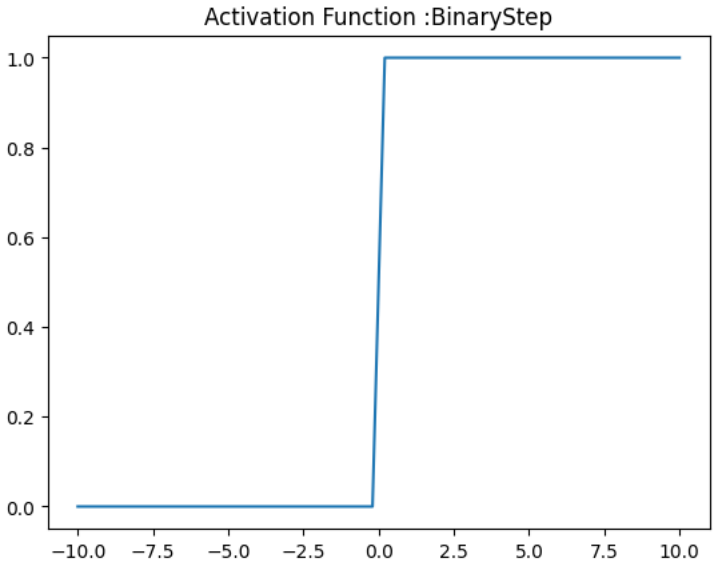
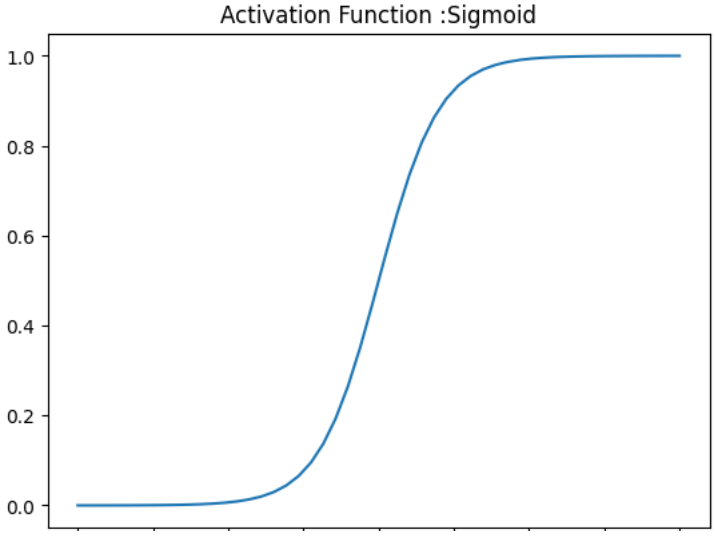
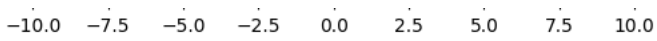
```
import numpy as np
import matplotlib.pyplot as plt
import numpy as np


# Binary Step Activation Function
def binaryStep(x):
  #It returns '0' is the input is less then zero otherwise it returns one '''
  return np.heaviside(x,1)
x = np.linspace(-10, 10)
plt.plot(x, binaryStep(x))
plt.axis('tight')
plt.title('Activation Function :BinaryStep')
plt.show()
```

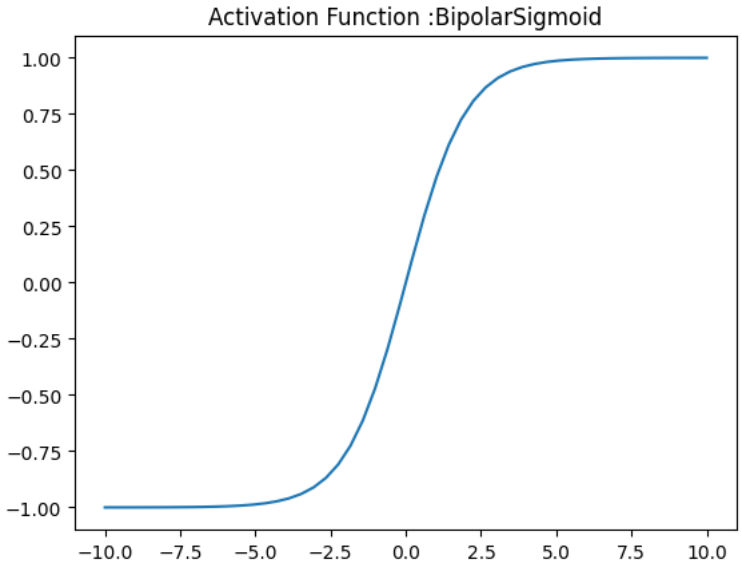



```
# Linear Activation Function
def linear(x):
  #''' y = f(x) It returns the input as it is'''
  return x
  x = np.linspace(-10, 10)
  plt.plot(x, linear(x))
  plt.axis('tight')
  plt.title('Activation Function :Linear')
  plt.show()


# Sigmoid Activation Function
def sigmoid(x):
  #''' It returns 1/(1+exp(-x)). where the values lies between zero and one '''
  return 1/(1+np.exp(-x))
x = np.linspace(-10, 10)
plt.plot(x, sigmoid(x))
plt.axis('tight')
plt.title('Activation Function :Sigmoid')
plt.show()
```

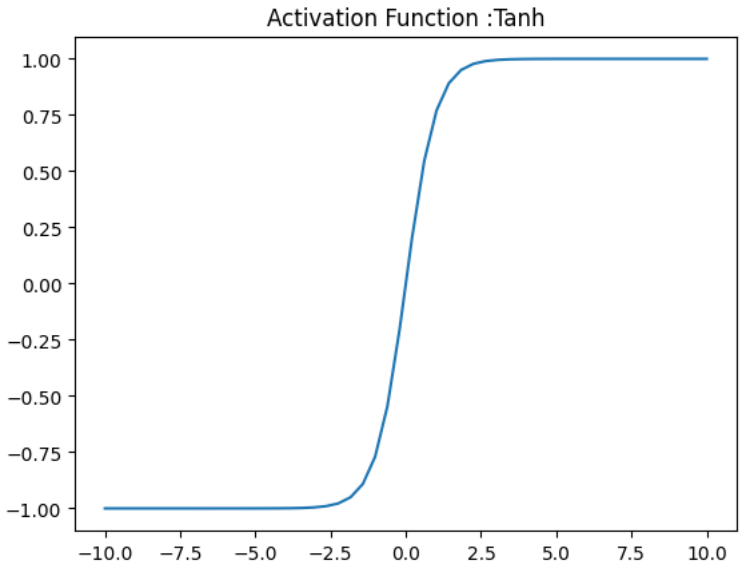





```
#Bipolar Sigmoid Activation Function
def bipolarsigmoid(x):
  #''' It returns -1+2/(1+exp(-x)). where the values lies between zero and one '''
  return -1+2/(1+np.exp(-x))
x = np.linspace(-10, 10)
plt.plot(x, bipolarsigmoid(x))
plt.axis('tight')
plt.title('Activation Function :BipolarSigmoid')
plt.show()
```

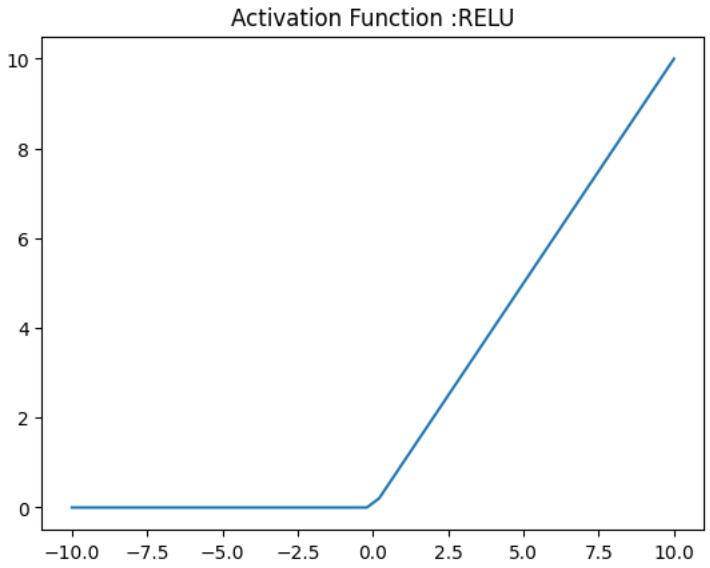



```
# Tanh Activation Function
def tanh(x):
  #''' It returns the value (1-exp(-2x))/(1+exp(-2x)) and the value returned will be lies in between -1 to 1.'''
  return np.tanh(x)
x = np.linspace(-10, 10)
plt.plot(x, tanh(x))
plt.axis('tight')
plt.title('Activation Function :Tanh')
plt.show()
```

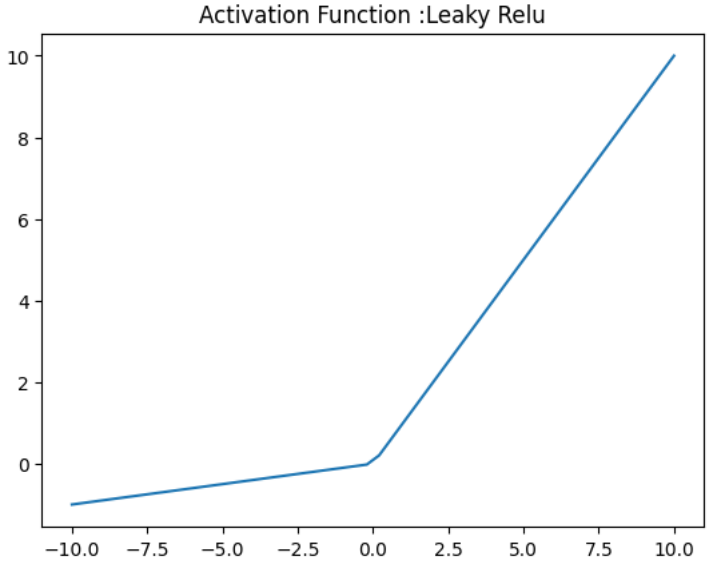



```
# ReLU Activation Function
def RELU(x):
  #''' It returns zero if the input is less than zero otherwise it returns the given input. '''
  x1=[]
  for i in x:
    if i<0:
      x1.append(0)
    else:
      x1.append(i)
  return x1
```

```
x = np.linspace(-10, 10)
plt.plot(x, RELU(x))
plt.axis('tight')
plt.title('Activation Function :RELU')
plt.show()
```




```
#Leaky ReLU Activation Function
def leaky_relu(x):
  alpha = 0.1
  return np.maximum(alpha*x, x)
x = np.linspace(-10, 10)
plt.plot(x, leaky_relu(x))
plt.axis('tight')
plt.title('Activation Function :Leaky Relu')
plt.show()
```



```
# Softmax Activation Function
def softmax(x):
  #''' Compute softmax values for each sets of scores in x. '''
  return np.exp(x) / np.sum(np.exp(x), axis=0)
x = np.linspace(-10, 10)
plt.plot(x, softmax(x))
plt.axis('tight')
plt.title('Activation Function :Softmax')
plt.show()
```