NAME	**Shreya Shetty**
UID	**2019140059**
CLASS	**TE IT**
BATCH	**B**
ACADEMIC YEAR	**2021-22**
SUBJECT	**SC (Soft Computing)**
COURSE CODE	**IT312**
EXPERIMENT NO.	**2**

Aim:

To implement various Transfer/Activation Functions.

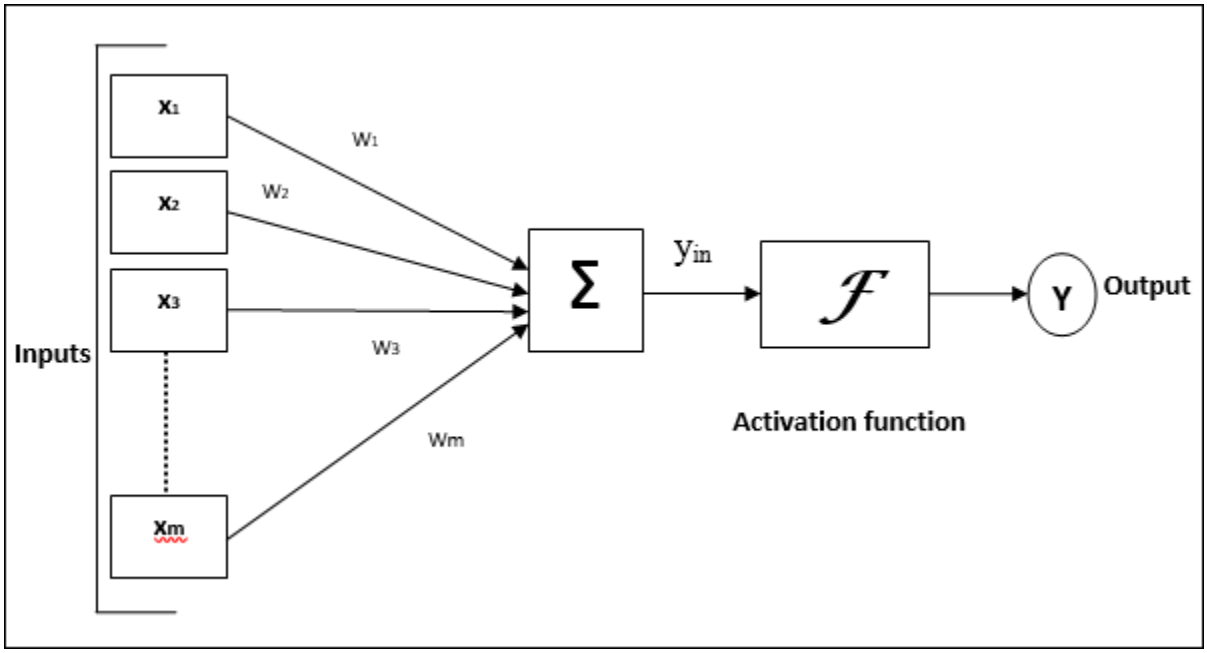
Theory:

Activation Function

- The output from a processing unit is termed as its activation. Activation of a processing unit is a function of the net input to the processing unit.
- The function that maps the net input value to the output signal value, i.e., the activation, is known as the activation function of the unit.
- The activation function is applied over the net input to calculate the output of an ANN.
- The information processing of a processing element can be viewed as consisting of two major parts: input and output.
- An integration function (say f) is associated with the input of a processing element. This function serves to combine activation, information or evidence from an external source or other processing elements into a net input to the processing element.
- The nonlinear activation function is used to ensure that a neuron's response is bounded - that is, the actual response of the neuron is conditioned or dampened as a result of large or small activating stimuli and is thus controllable.
- Certain nonlinear functions are used to achieve the advantages of a multilayer network from a single-layer network.
- When a signal is fed through a multilayer network with linear activation functions, the output obtained remains same as that could be obtained using a single-layer network.
- Due to this reason, nonlinear functions are widely used in multilayer networks compared to linear functions.

Model of Artificial Neural Network

The following diagram represents the general model of ANN followed by its processing.



For the above general model of artificial neural network, the net input can be calculated as follows −

$$y_{in}=x_1.w_1+x_2.w_2+x_3.w_3\ldots x_m.w_m$$

i.e., Net iInput $y_{in}=\sum_i^m x_i.w_i$

The output can be calculated by applying the activation function over the net input.

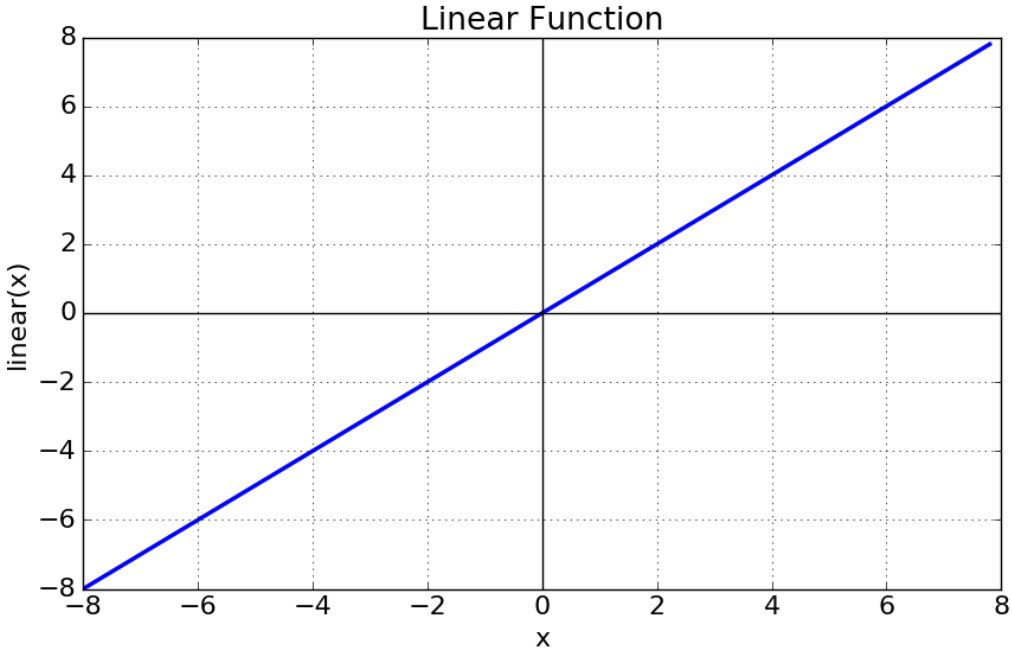
$$Y=F(y_{in})$$

Output = function (net input calculated)

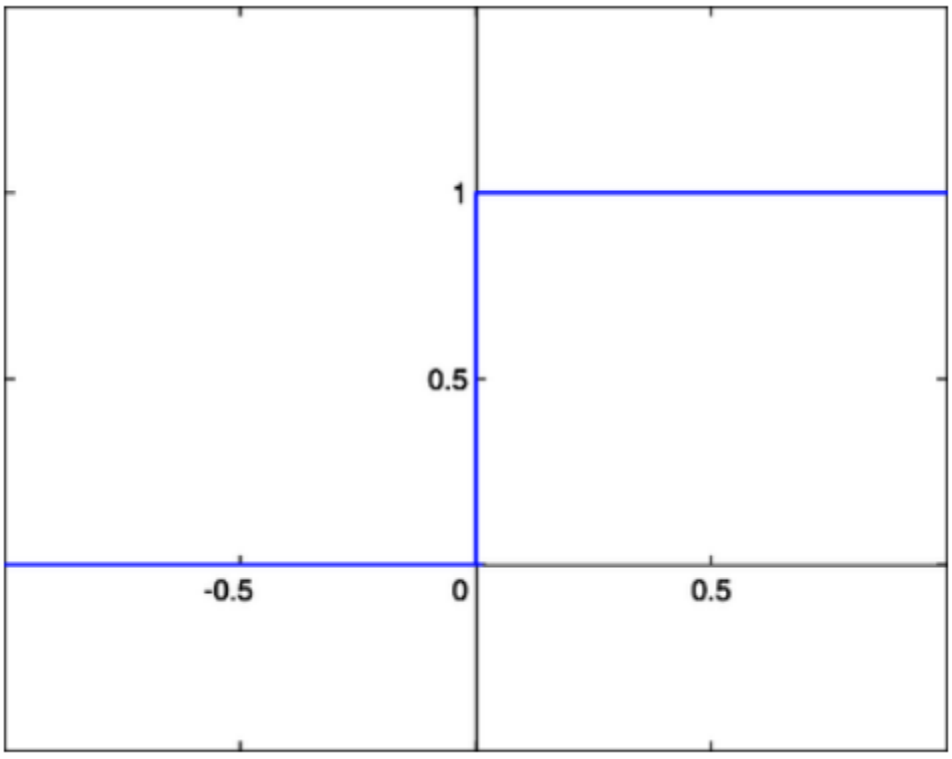
Types of Activation Function

Diagrammatic Representation :

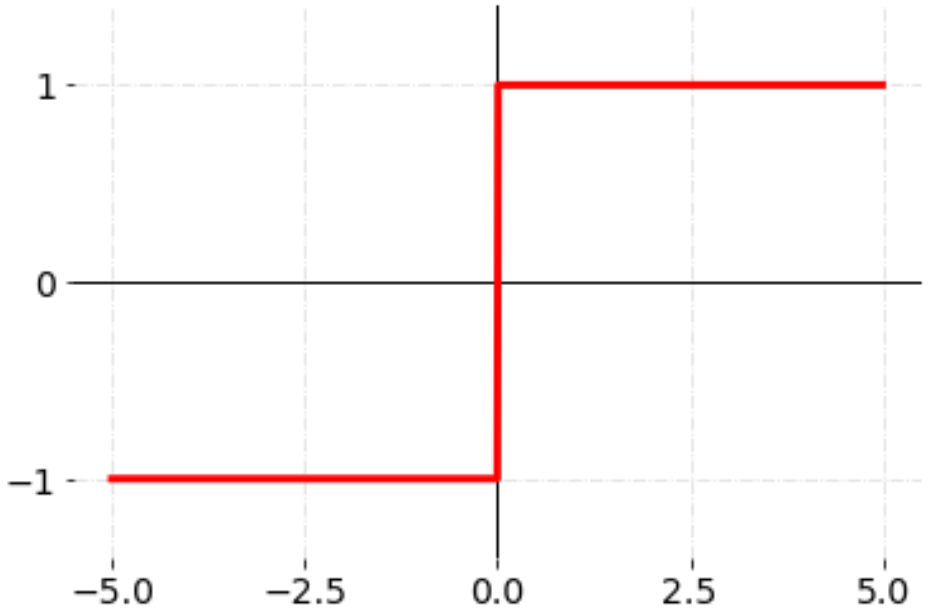
1. Linear/Identity Function



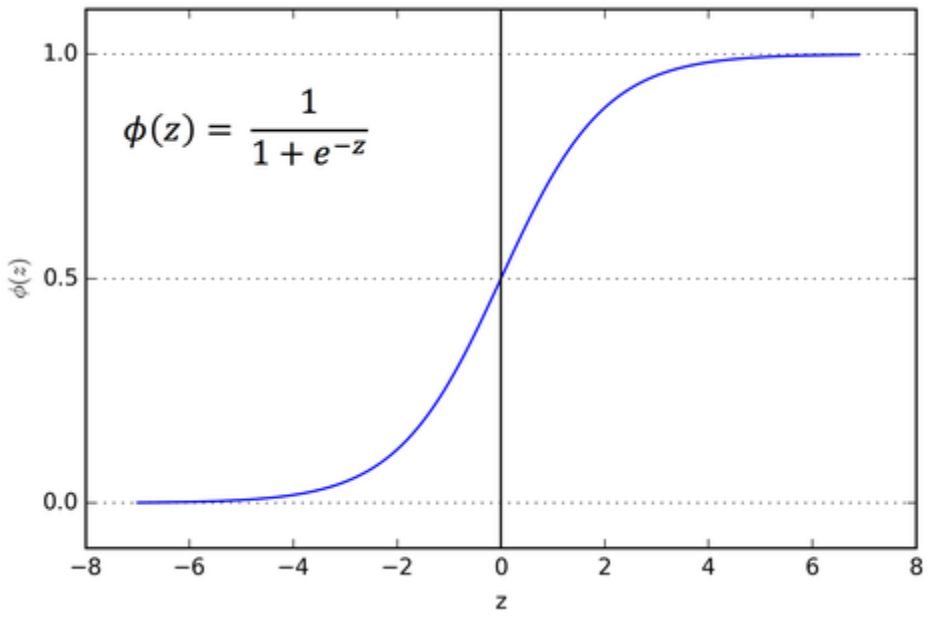
2. Binary Step Function



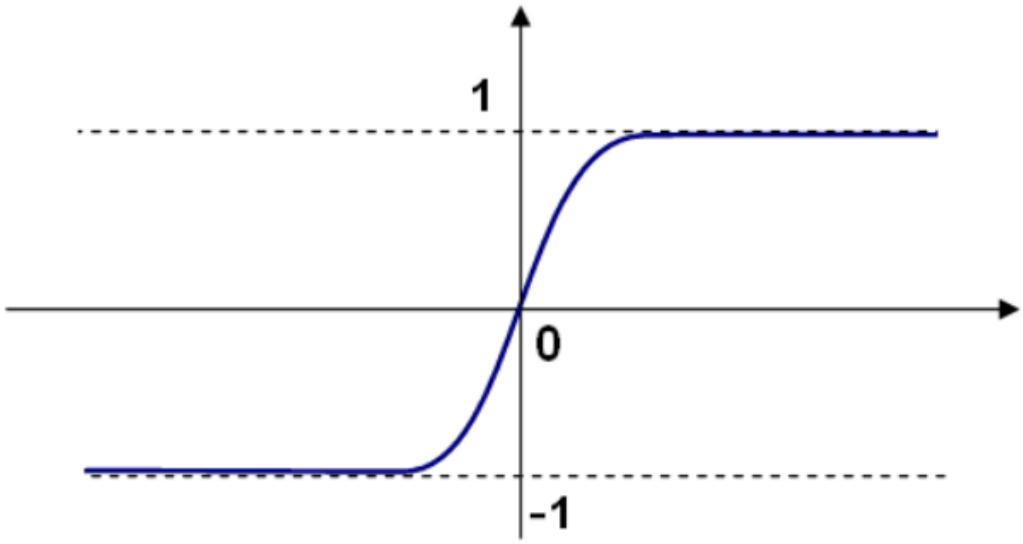
3. Bipolar Step Function



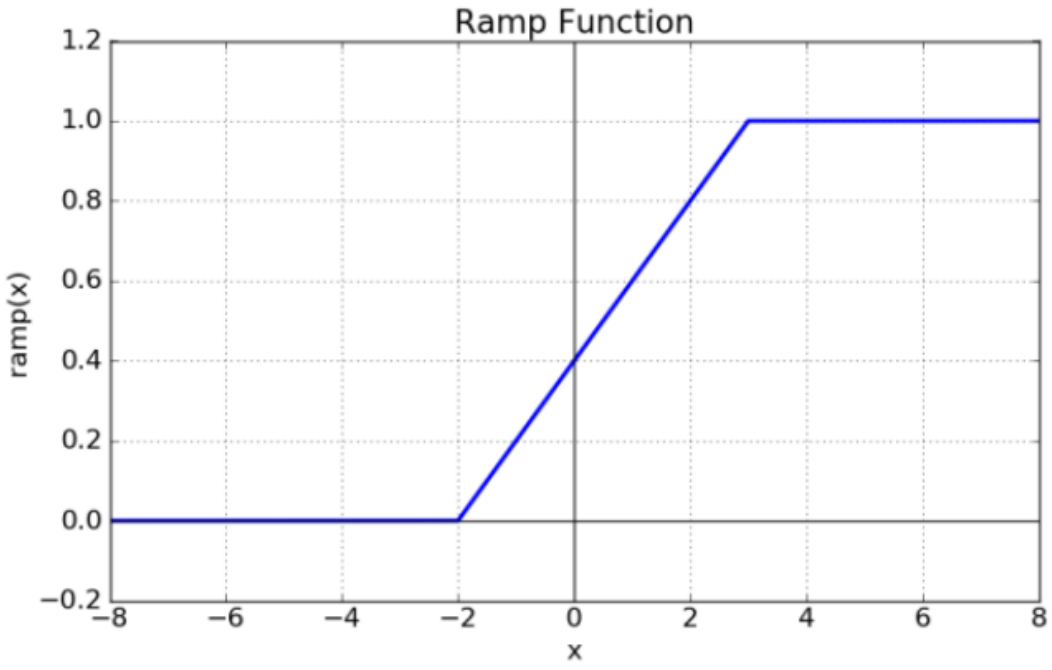
4. Sigmoid Function
   a. Binary Sigmoid Function



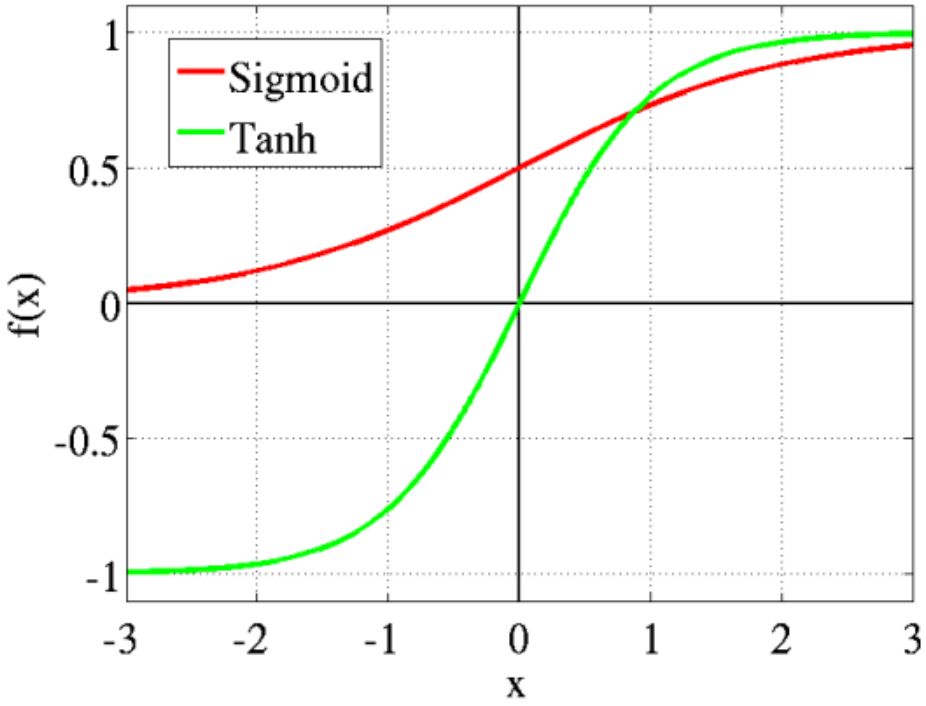
b. Bipolar Sigmoid Function



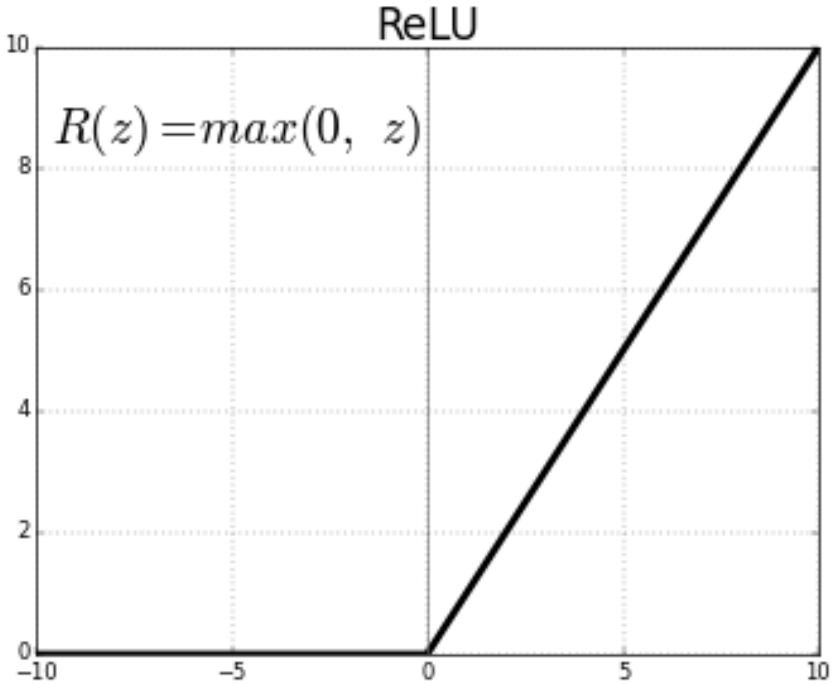
5. Ramp Activation Function



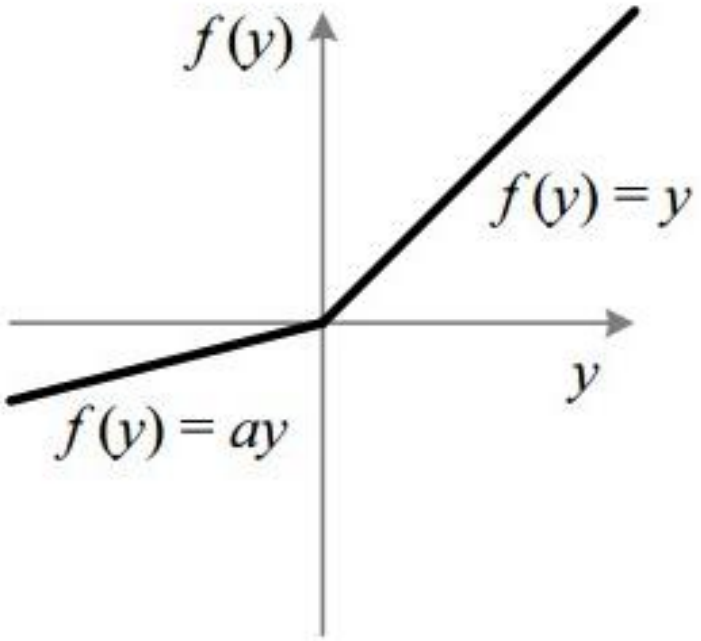
6. Tanh Function



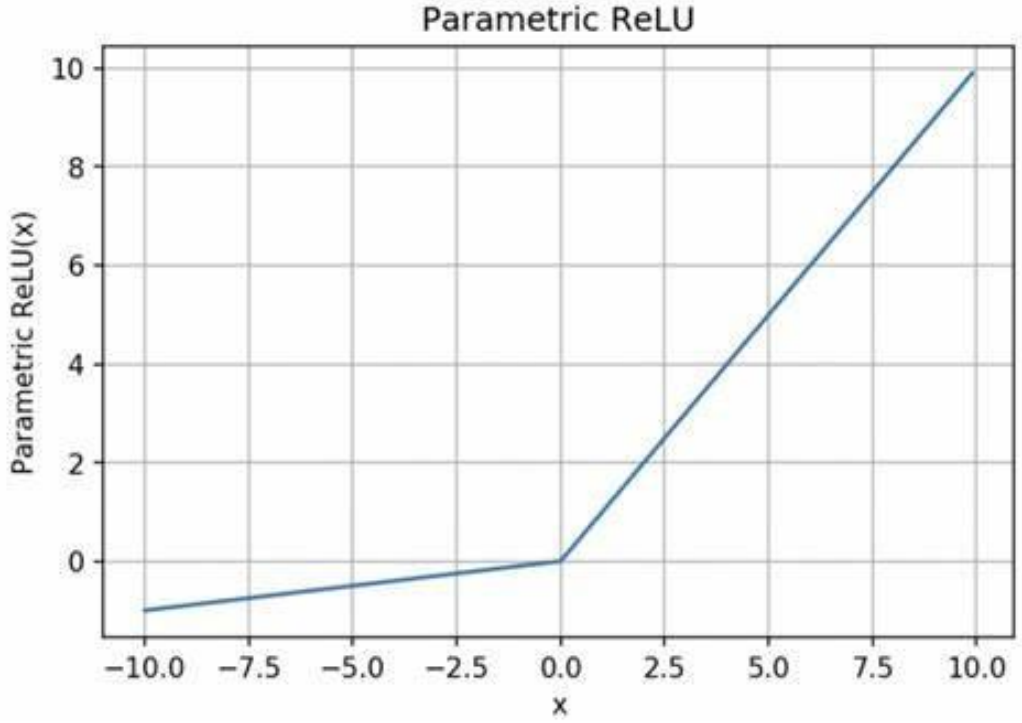
7. ReLU Function



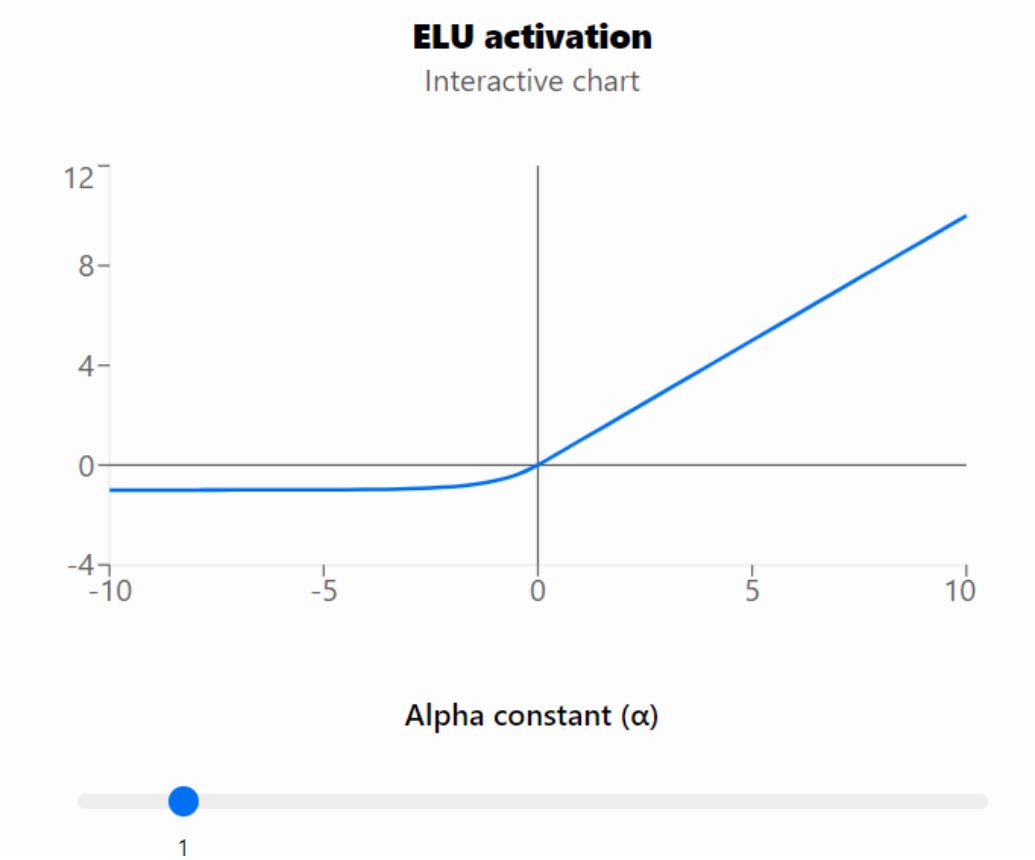
8. Leaky ReLU



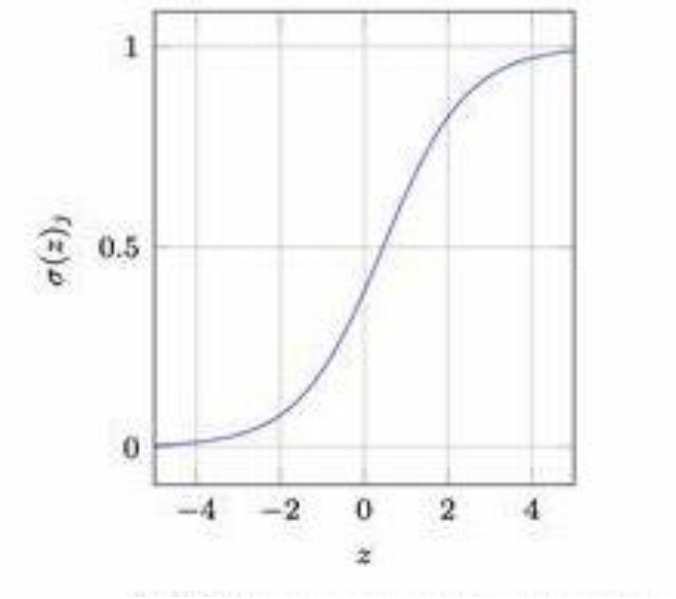
9. Parameterised ReLU



10. Exponential Linear Unit



11. Softmax



Function Name	Formula	Description	Significance	Application
Identity Function	f(x) = x $\forall x$	It is a linear activation function	Output is same as input	Input layer uses Identity activation function
Binary Step Function	f(x) = 1, x >= θ = 0, x < θ where θ represents the threshold value.	The neuron is activated if the input to the activation function is greater than a a threshold, else it is deactivated.	Used as an activation function while creating a binary classifier.	Widely used in single-layer nets to convert the net input to an output that is binary (1 or 0).
Bipolar Step Function	f(x) = 1, x >= θ = -1, x < θ where θ represents the threshold value.	If the value of Y is above the threshold, the output is +1 and if it's less than the threshold then the output is -1	Used to convert the net input to an output that is bipolar (+1 or -1).	Used in single-layer nets
Sigmoid Function	Binary Sigmoid: f(x) = $1/(1+e^{-\lambda x})$ Bipolar Sigmoid: f(x) = $(1-e^{\lambda x}))/(1+e^{-\lambda x})$	Sigmoid Function is not symmetric around origin. So, output of all the neurons will be of the same sign.	Most widely used non-linear activation function.	The sigmoidal functions are used back-propagation nets. However, it has vanishing gradient problem.
Ramp Activation Function	f(x)= 1, if x > 1 = x, if 0 <= x <= 1 = 0, if x < 0	The output of ramp function is either 1, 0 or x itself	The ramp function is a truncated version of the linear function.	
Tanh Function	tanh(x) = $(1-e^{-2x}))/(1+e^{-2x})$	Very similar to the sigmoid function. Only difference is that it is symmetric around the origin.	Usually tanh is preferred over the sigmoid function since it is zero centered and the gradients are not restricted to move in a certain direction.	Widely used activation function in neural networks
ReLU Function	f(x) = x , x >= 0 = 0, x < 0	The ReLu function is non-linear activation function	It does not activate all the neurons at the same time. Hence, ReLU function is far	It has gained popularity in Deep learning models

			more computationally efficient When compared to the sigmoid and tanh function. It avoids vanishing gradient problems.	
Leaky ReLU	$f(x) = x , x >= 0$ $= ax, x < 0$	Leaky ReLU is an improved version of ReLU Function	Here, the gradient of the left side of the graph comes out to be a non zero value. Hence, no dead neurons problem in that region.	Used in Deep Learning Models
Parameterised ReLU	$f(x) = x , x >= 0$ $= 0.01x, x < 0$	This is another variant of ReLU that aims to solve the problem of gradient's becoming zero for the left half of the axis.	Introduces a new parameter as a slope of the negative part of the function. If a = 0.01 then it becomes Leaky ReLU	The parameterized ReLU function is used when the leaky ReLU function still fails to solve the problem of dead neurons and the relevant information is not successfully passed to the next layer.
Exponential Linear Unit	$f(x) = x , x >= 0$ $= a(e^x - 1), x < 0$	Variant of Rectified Linear Unit (ReLU) that modifies the slope of the negative part of the function.	Unlike the leaky ReLU and parametric ReLU functions, instead of a straight line, ELU uses a log curve for defining the negative values	Used to solve dead neuron problem
Softmax	$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$	Softmax function is often described as a combination of multiple sigmoids.	Used for multiclass classification problems. This function returns the probability for	Used for multiclass classification

			a datapoint belonging to each individual class.	

Consider Inputs : $x_1 = -2.8$, $x_2 = -3.23$, $x_3 = -4.34$

Consider Weights : $w_1 = 0.3$, $w_2 = 0.1$, $w_3 = 0.4$

Net Input y_{in} = $x_1w + x_2w_2 + x_3w_3$

= (-2.8)*0.3 + (-3.23)*0.1 + (-4.34)*0.4

y_{in} = -2.899

Sr. No.	Function Name	Formula	Input to activation function (y_{in})	Output
1	Identity Function	f(x) = x ∀ x	-2.899	-2.899
2	Binary Step Function	f(x) = 1, x >= Θ = 0, x < Θ	-2.899	0
3	Bipolar Step Function	f(x) = 1, x >= Θ = -1, x < Θ	-2.899	-1
4	Binary Sigmoid Function	f(x) = 1/(1+e^- x)	-2.899	0.0522
	Bipolar Sigmoid Function	f(x) = (1-e^- x)/(1+e^- x)	-2.899	-0.8955
5	Ramp Activation Function	f(x)= 1, if x > 1 = x, if 0 <= x <= 1 = 0, if x < 0	-2.899	0
6	Tanh Function	tanh(x) = (1-e^{-2x})) /(1+e^{-2x})	-2.899	-0.994
7	ReLU Function	f(x) = x , x >= 0 = 0, x < 0	-2.899	0
8	Leaky ReLU	f(x) = x , x >= 0 = ax, x < 0	-2.899	-0.289
9	Parameterised ReLU	f(x) = x , x >= 0 = 0.01x, x < 0	-2.899	-0.0298
10	Exponential Linear Unit	f(x) = x , x >= 0 = $a(e^x - 1)$, $x < 0$	-2.899	-0.945 (for a=1)
11	Softmax	$\sigma(\underset{z}{\rightarrow})_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$	[-1.345, 0.45, 1.34]	[0.046, 0.278, 0.676]

Tool/Language:

Programming language: Python

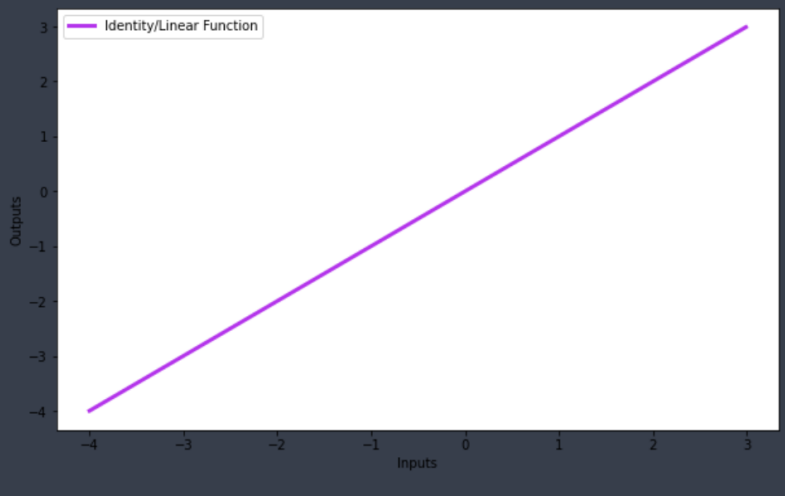
Libraries Used: numpy

Code with Output:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

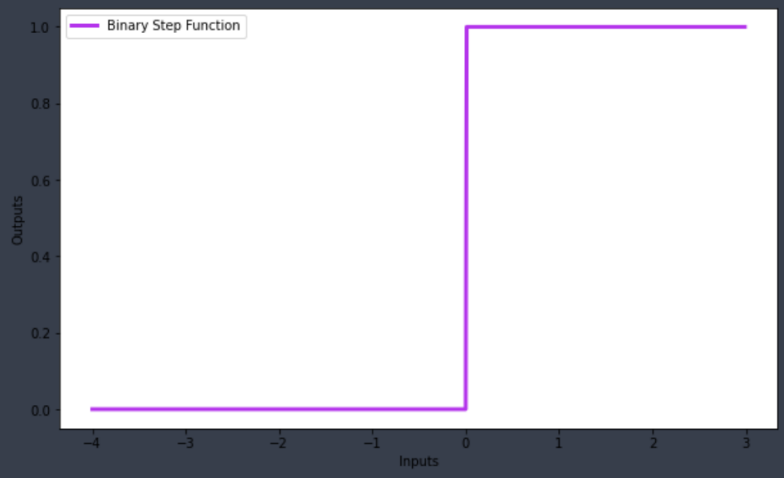
IDENTITY FUNCTION:

In [2]:
```
#Identity Function
def identity(x):
  return x
x= np.arange(-4,3,0.01)
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(x,identity(x), color="#b434eb", linewidth=3, label="Identity/Linear Function")
ax.legend(loc="upper left")
plt.ylabel("Outputs")
plt.xlabel("Inputs")
fig.show()
```



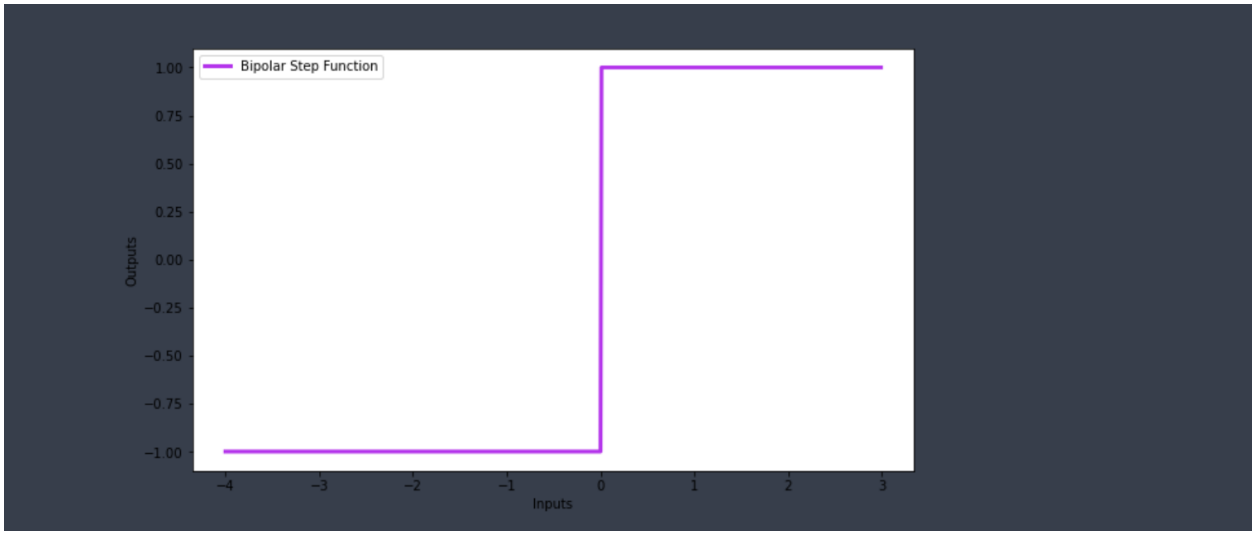
BINARY STEP FUNCTION:

In [3]:
```
#Binary Step Function
def binary_step(x,t):
  if (x<t):
      return 0
  else:
      return 1

x= np.arange(-4,3,0.01)
binary_step= np.vectorize(binary_step)

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(x,binary_step(x,0), color="#b434eb", linewidth=3, label="Binary Step Function")
ax.legend(loc="upper left" )
plt.ylabel("Outputs")
plt.xlabel("Inputs")
fig.show()
```



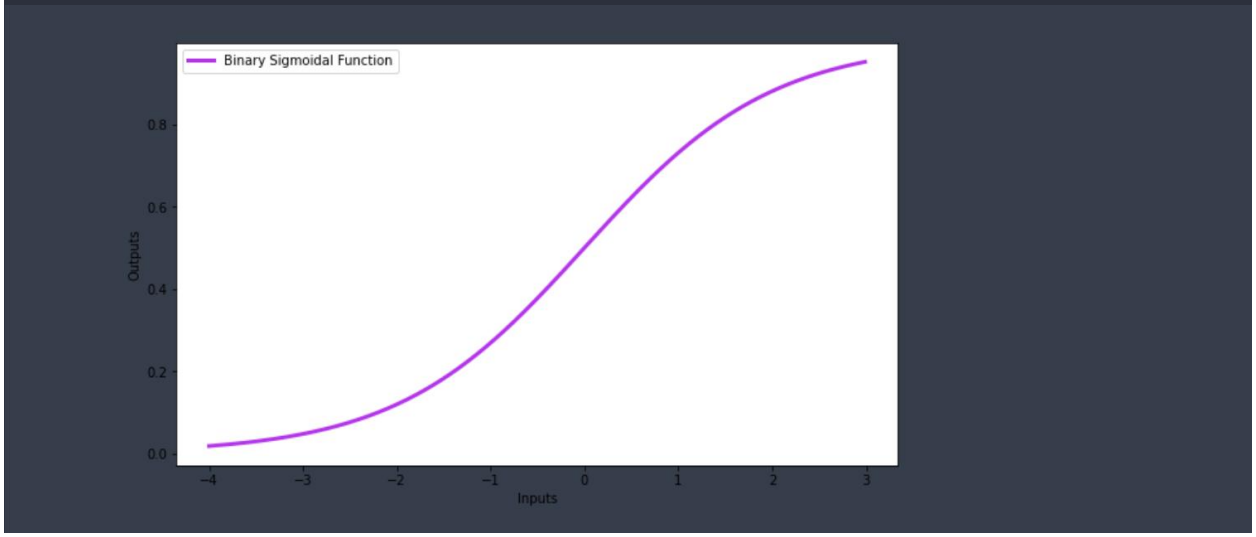
BIPOLAR STEP FUNCTION:

In [4]:
```
#Bipolar Step Function
def bipolar_step(x : int, t : float) ->int:
  if x>=t:
    return 1
  else:
    return -1

x= np.arange(-4,3,0.01)
bipolar_step= np.vectorize(bipolar_step)

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(x,bipolar_step(x,0), color="#b434eb", linewidth=3, label="Bipolar Step Function")
ax.legend(loc="upper left" )
plt.ylabel("Outputs")
plt.xlabel("Inputs")
fig.show()
```



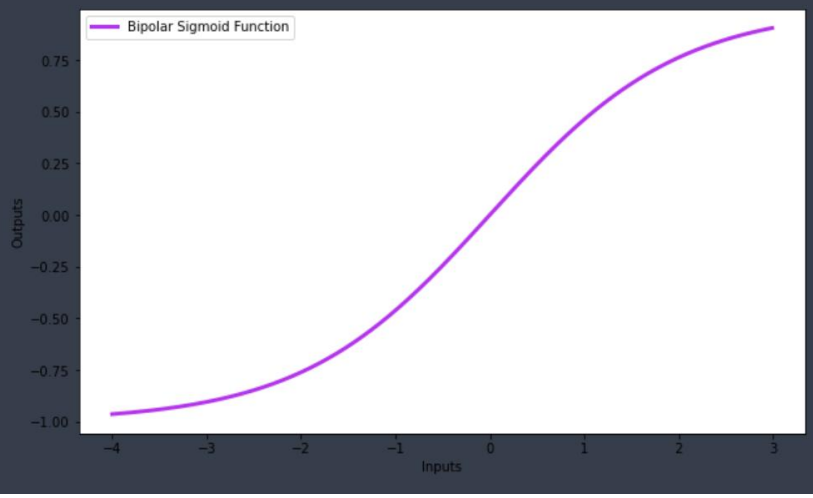
BINARY SIGMOIDAL FUNCTION:

In [5]:
```
#Binary Sigmoidal Function
def binary_sigmoid(x):
    return 1/(1+np.exp(-x))

x= np.arange(-4,3,0.01)
binary_sigmoid= np.vectorize(binary_sigmoid)

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(x,binary_sigmoid(x), color="#b434eb", linewidth=3, label="Binary Sigmoidal Function")
ax.legend(loc="upper left" )
plt.ylabel("Outputs")
plt.xlabel("Inputs")
fig.show()
```



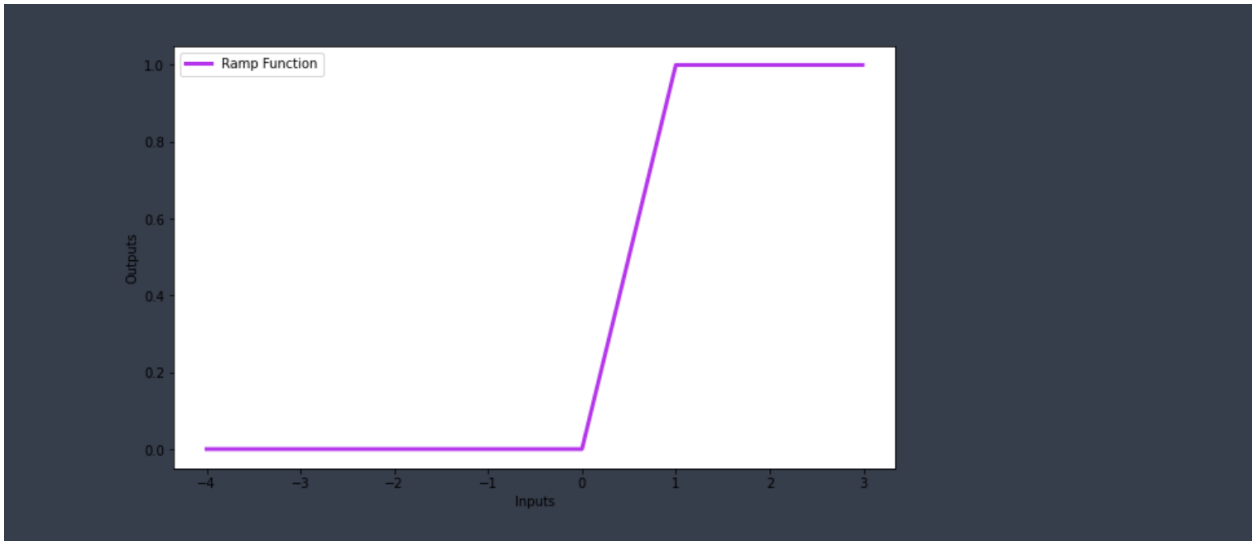
BIPOLAR SIGMOIDAL FUNCTION:

In [6]:
```
#Bipolar Sigmoid Function
def bipolar_sigmoid(x):
    return (1-np.exp(-x))/(1+np.exp(-x))

x= np.arange(-4,3,0.01)

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(x,bipolar_sigmoid(x), color="#b434eb", linewidth=3, label="Bipolar Sigmoid Function")
ax.legend(loc="upper left" )
plt.ylabel("Outputs")
plt.xlabel("Inputs")
fig.show()
```



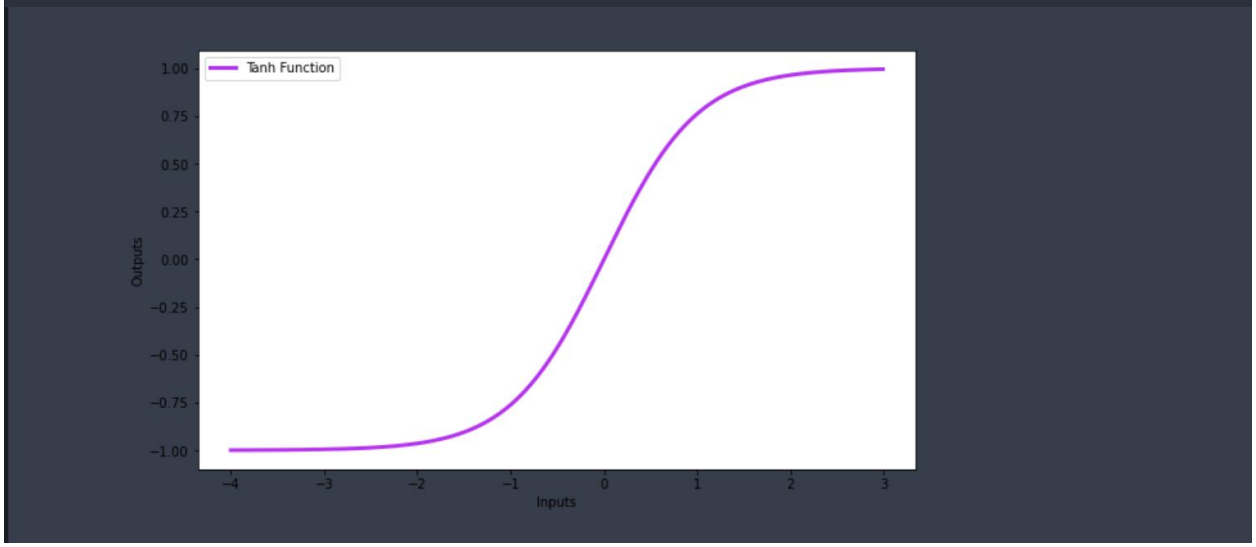
RAMP FUNCTION:

In [7]:
```
#Ramp Function
def ramp(x):
   if x>1:
     return 1
   elif x>=0 and x<=1:
     return x
   else:
     return 0
x= np.arange(-4,3,0.01)
ramp= np.vectorize(ramp,otypes=[np.float64])
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(x,ramp(x), color="#b434eb", linewidth=3, label="Ramp Function")
ax.legend(loc="upper left" )
plt.ylabel("Outputs")
plt.xlabel("Inputs")
fig.show()
```



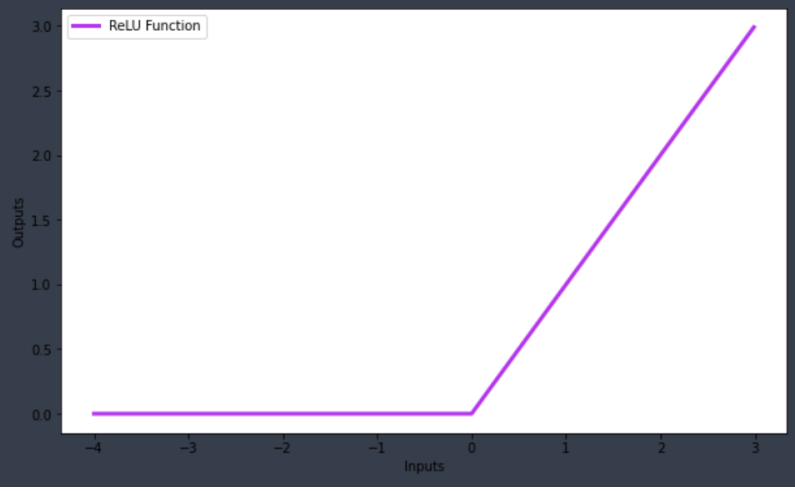
TANH FUNCTION:

```
In [8]: #Tanh Function
def tanh(x):
    return (1-np.exp(-2*x))/(1+np.exp(-2*x))

x= np.arange(-4,3,0.01)

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(x,tanh(x), color="#b434eb", linewidth=3, label="Tanh Function")
ax.legend(loc="upper left" )
plt.ylabel("Outputs")
plt.xlabel("Inputs")
fig.show()
```



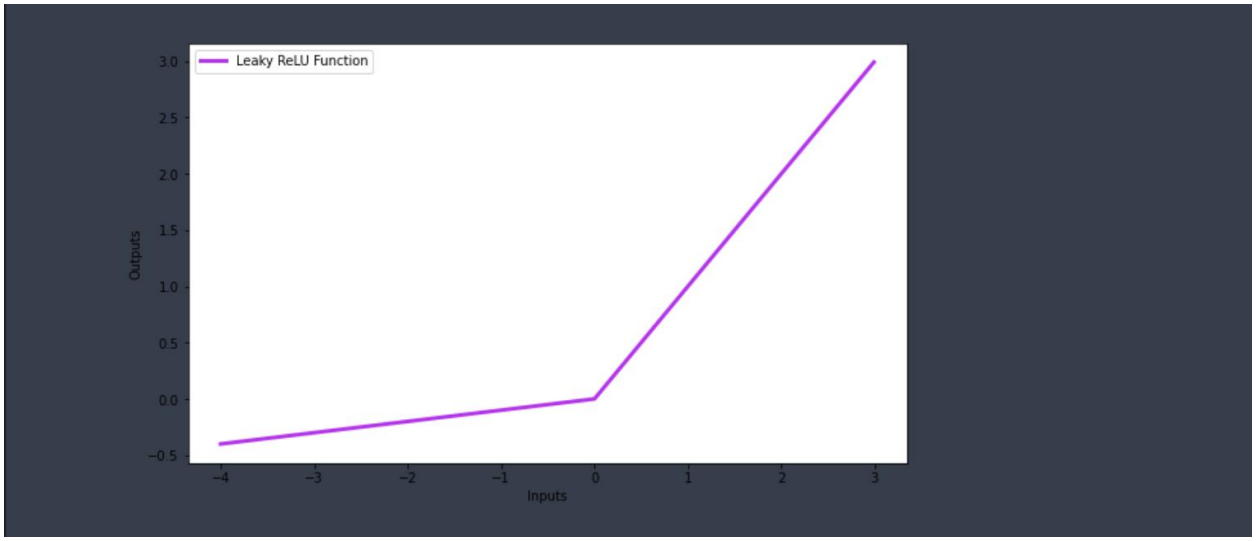
RELU FUNCTION:

In [9]:
```
#ReLU Function
def relu(x):
    return np.maximum(0,x)

x= np.arange(-4,3,0.01)

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(x,relu(x), color="#b434eb", linewidth=3, label="ReLU Function")
ax.legend(loc="upper left" )
plt.ylabel("Outputs")
plt.xlabel("Inputs")
fig.show()
```



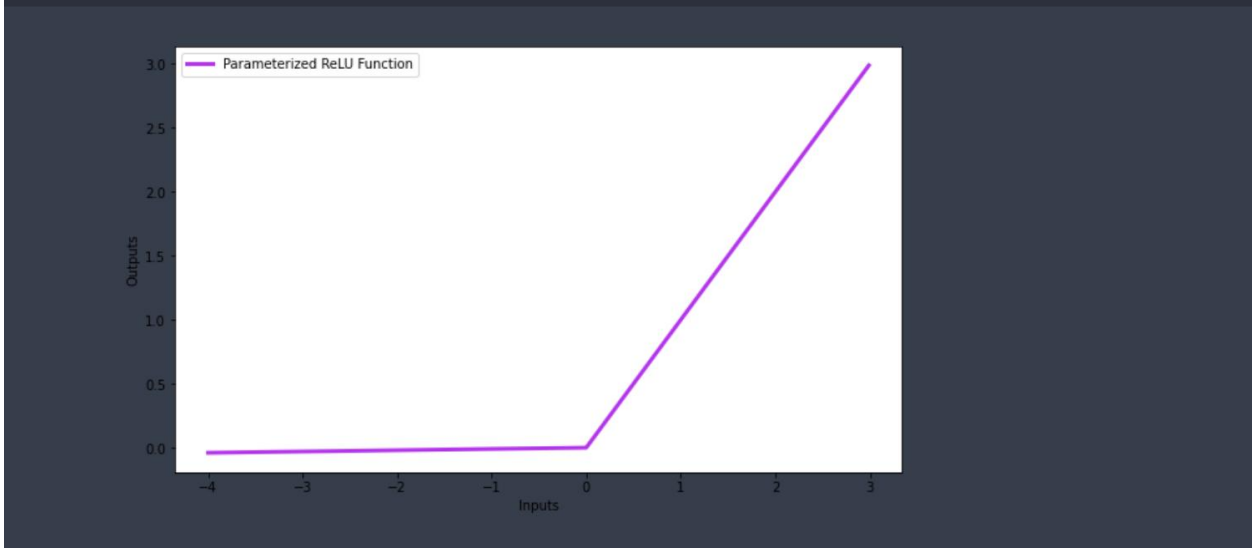
LEAKY RELU FUNCTION:

In [10]:
```
#Leaky ReLU Function
def lrelu(x):
    return np.maximum(0.1*x,x)

x= np.arange(-4,3,0.01)

fig, ax = plt.subplots(figsize=(10, 6))
lrelu = np.vectorize(lrelu,otypes=[np.float64])
ax.plot(x,lrelu(x), color="#b434eb", linewidth=3, label="Leaky ReLU Function")
ax.legend(loc="upper left" )
plt.ylabel("Outputs")
plt.xlabel("Inputs")
fig.show()
```



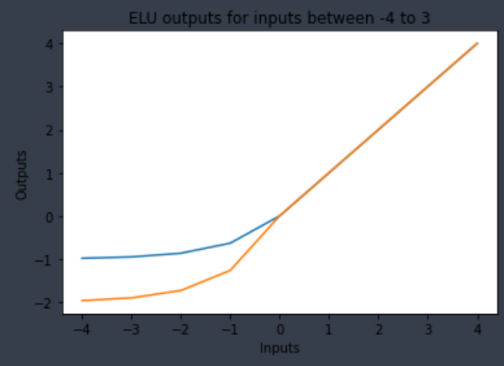
PARAMETERIZED RELU FUNCTION:

In [11]:
```
#Parameterized ReLU Function
#Leaky ReLU Function
def prelu(x):
    return np.maximum(0.01*x,x)
x= np.arange(-4,3,0.01)
fig, ax = plt.subplots(figsize=(10, 6))
prelu = np.vectorize(prelu,otypes=[np.float64])
ax.plot(x,prelu(x), color="#b434eb", linewidth=3, label="Parameterized ReLU Function")
ax.legend(loc="upper left" )
plt.ylabel("Outputs")
plt.xlabel("Inputs")
fig.show()
```



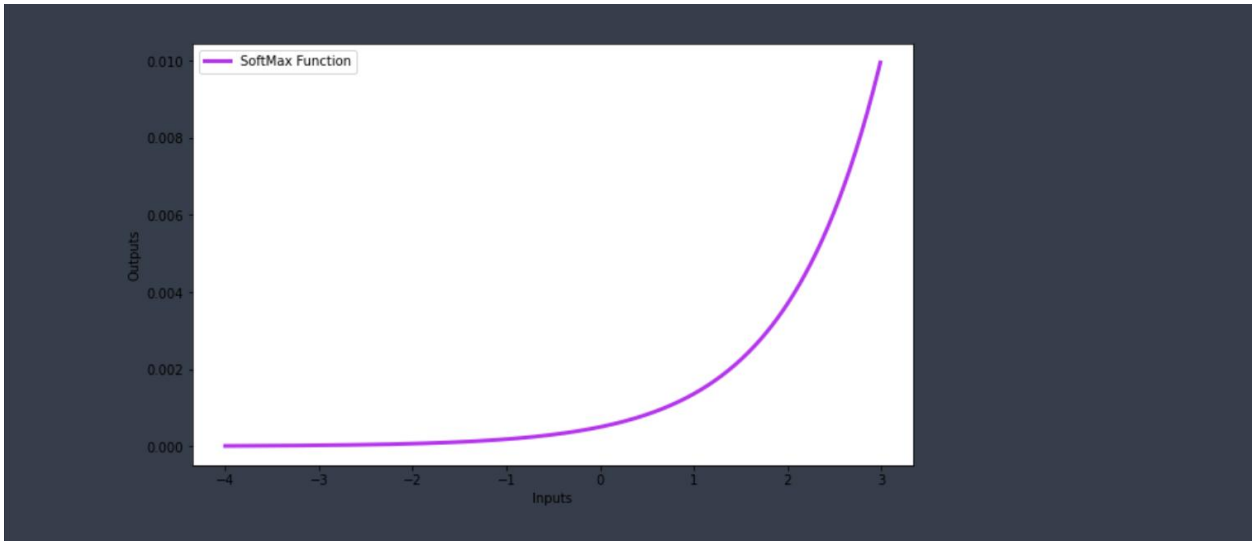
EXPONENTIAL LINEAR UNIT FUNCTION:

In [12]:
```
# Exponential Linear Unit

def elu(x, a):
    if x >= 0:
        return x
    else:
        return a*(np.exp(x) - 1)

for a in range(1, 3):
    inputs = [x for x in range(-4, 5)]
    outputs = [elu(x, a) for x in inputs]
    plt.plot(inputs, outputs)

plt.title("ELU outputs for inputs between -4 to 3")
plt.ylabel("Outputs")
plt.xlabel("Inputs")
plt.show()
```



SOFTMAX FUNCTION:

In [13]:
```
#SoftMax
def softmax(x):
    return np.exp(x)/np.sum(np.exp(x))

x= np.arange(-4,3,0.01)
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(x,softmax(x), color="#b434eb", linewidth=3, label="SoftMax Function")
ax.legend(loc="upper left" )
plt.ylabel("Outputs")
plt.xlabel("Inputs")
fig.show()
```



PRINTING Y_{IN} FOR EACH ACTIVATION FUNCTION AND ITS OUTPUT:

```
inputs = np.array([-2.8, -3.23,-4.34])
wt=np.array([0.3, 0.1, 0.4])
y_in = inputs @ wt
print('Input = ',inputs)
print('Weights = ',wt)
print('y_in = w1*x1+w2*x2+x3*w3 = ',y_in)
print('-------------------------------------------------------------------------------------------')
print('| Function Name \t| Input to Activation Function  |\t\tOutput\t\t|')
print('-------------------------------------------------------------------------------------------')
print('| Identity  \t\t|\t',y_in,' \t\t|\t',identity(y_in),'\t\t|')
print('| Binary Step  \t\t|\t',y_in,' \t\t|\t',binary_step(y_in,0),'\t\t\t|')
print('| Bipolar Step  \t|\t',y_in,' \t\t|\t',bipolar_step(y_in,0),'\t\t\t|')
print('| Binary Sigmoid\t|\t',y_in,' \t\t|\t',np.round(binary_sigmoid(y_in),3),'\t\t\t|')
print('| Bipolar Sigmoid\t|\t',y_in,' \t\t|\t',np.round(bipolar_sigmoid(y_in),3),'\t\t|')
print('| Ramp  \t\t|\t',y_in,' \t\t|\t',ramp(y_in),'\t\t\t|')
print('| Tanh  \t\t|\t',y_in,' \t\t|\t',np.round(tanh(y_in),3),'\t\t|')
print('| ReLU  \t\t|\t',y_in,' \t\t|\t',relu(y_in),'\t\t\t|')
print('| Leaky ReLU  \t\t|\t',y_in,' \t\t|\t',lrelu(y_in),'\t\t|')
print('| Parameterized ReLU  \t|\t',y_in,' \t\t|\t',np.round(prelu(y_in),3),'\t\t|')
print('| ELU Function \t\t|\t',y_in,' \t\t|\t',np.round(elu(y_in,1),3),'\t\t|')
print('| Softmax  \t\t|  ',np.array([-1.345,  0.45, 1.34]),'\t|\t',np.round(softmax(np.array([-1.345,  0.45
print('-------------------------------------------------------------------------------------------')
```

```
Input =  [-2.8  -3.23 -4.34]
Weights =  [0.3 0.1 0.4]
y_in = w1*x1+w2*x2+x3*w3 =  -2.899
```

Function Name	Input to Activation Function	Output
Identity	-2.899	-2.899
Binary Step	-2.899	0
Bipolar Step	-2.899	-1
Binary Sigmoid	-2.899	0.052
Bipolar Sigmoid	-2.899	-0.896
Ramp	-2.899	0.0
Tanh	-2.899	-0.994
ReLU	-2.899	0.0
Leaky ReLU	-2.899	-0.2899
Parameterized ReLU	-2.899	-0.029
ELU Function	-2.899	-0.945
Softmax	[-1.345 0.45 1.34]	[0.046 0.278 0.676]

Conclusion:

In this experiment, I learnt about various activation functions namely Identity Function, Binary Step Function, Bipolar Step Function, Binary Sigmoid Function, Bipolar Sigmoid Function, Ramp Activation Function, Tanh Function, ReLU Function, Leaky ReLU, Parameterised ReLU, Exponential Linear Unit and Softmax and implemented their plots and found out output values using Python.