S. B. JAIN INSTITUTE OF TECHNOLOGY,

MANAGEMENT & RESEARCH, NAGPUR.

Practical No. 02

Aim: Apply the knowledge to study and implement Activation functions in Deep Learning.

Name of Student	**:** Shrutika Pradeep Bagdi
Roll No	**:** CS22130
Semester/Year	**:** VIIth Sem / IVth Year
Academic Session	**:** 2025-2026 [ODD]
Date of Performance	: ______________________________
Date of Submission	: ______________________________

AIM: Apply the knowledge to study and implement Activation functions in deep learning.

OBJECTIVE/EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

- Develop a deeper understanding of the activation functions and its limitations
- Know how to diagnose and apply corrections to some problems with the Activation functionsfound in real data
- Use and understand applications of Activation functions in Neural Network.
- Develop a greater familiarity with a range of techniques and methods through a diverse set of theoretical and applied readings.
- Know where to go to learn more about the techniques in this class and those called for that were not covered in this class.

HARDWARE AND SOFTWARE REQUIREMENTS:

Hardware Requirement:

Software Requirement:

Google Colab

THEORY:

Activation function:

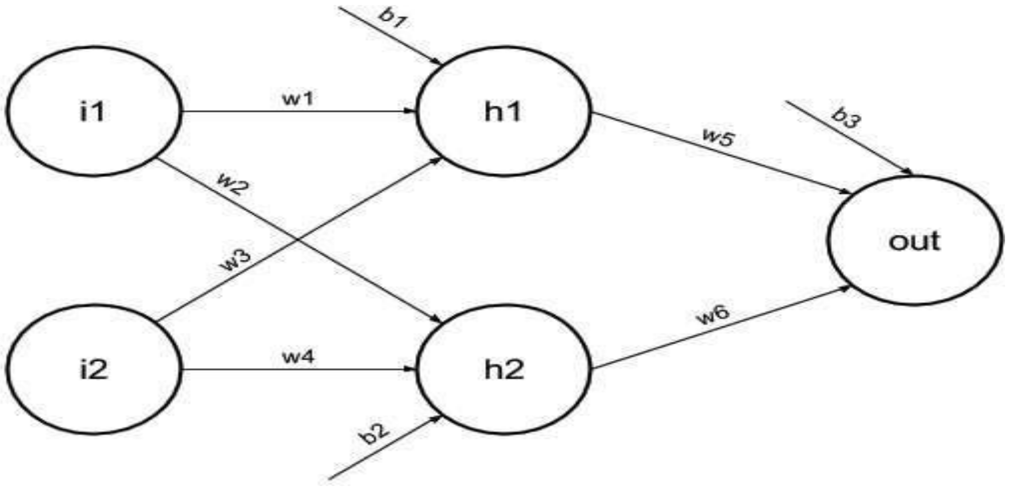
In the process of building a neural network, one of the choices you get to make is what Activation Function to use in the hidden layer as well as at the output layer of the network.

Elements of a Neural Network

Input Layer: This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.

Hidden Layer*:* Nodes of this layer are not exposed to the outer world, they are part of the abstraction provided by any neural network. The hidden layer performs all sorts of computation on the features entered through the input layer and transfers the result to the output layer.

Output Layer: This layer bring up the information learned by the network to the outer world.



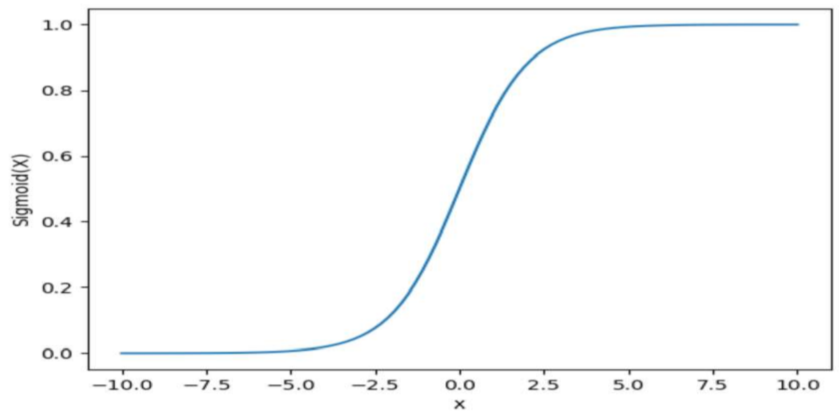
Variants of Activation Function

Linear Function

- **Equation :** Linear function has the equation similar to as of a straight line i.e. **y = x**
- No matter how many layers we have, if all are linear in nature, the final activation function of the last layer is nothing but just a linear function of the input of first layer.
- **Range :** -inf to +inf
- **Uses : Linear activation function** is used at just one place i.e. output layer.
- **Issues :** If we will differentiate linear function to bring non-linearity, result will no more depend on *input "x"* and function will become constant, it won't introduce any ground-breaking behavior to our algorithm.

For example : Calculation of price of a house is a regression problem. House price may have any big/small value, so we can apply linear activation at the output layer. Even in this case a neural net must have any non-linear function at hidden layers.

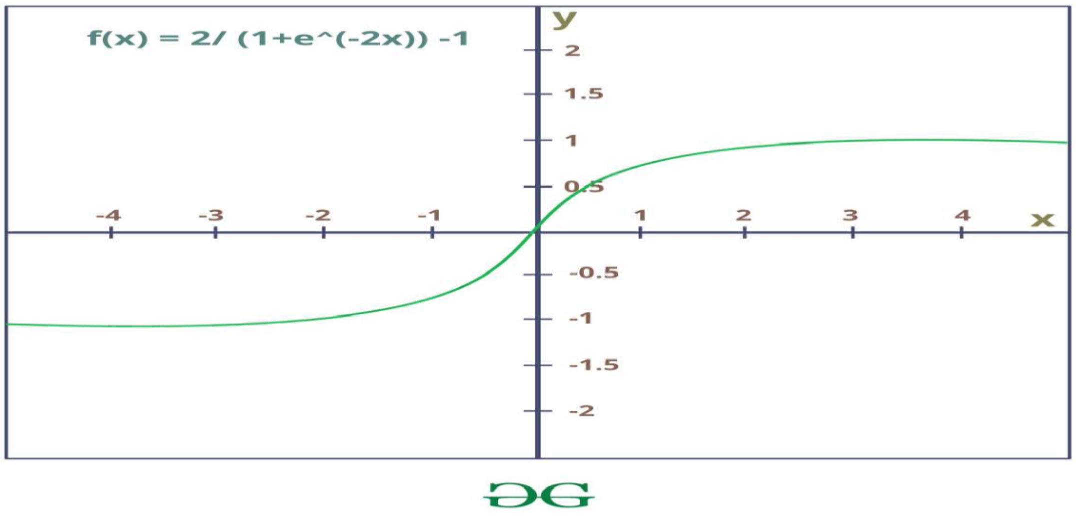
Sigmoid Function



- It is a function which is plotted as **'S'** shaped graph.
- **Equation :** A = 1/(1 + e^{-x})
- **Nature :** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range :** 0 to 1

  Uses : Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.

Tanh Function

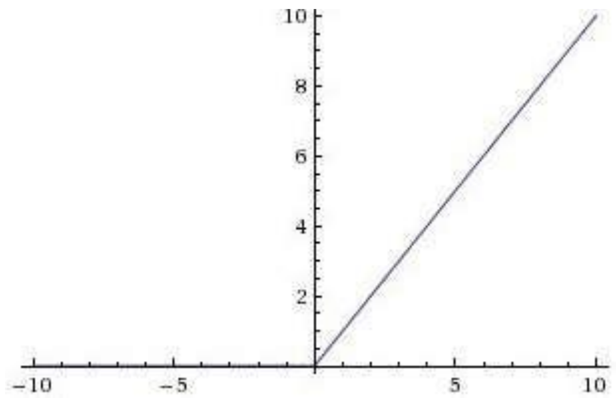


- The activation that works almost always better than sigmoid function is Tanh function also known as **Tangent Hyperbolic function**. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
- **Equation :-**

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

- **Value Range :-** -1 to +1
- **Nature :-** non-linear
- **Uses :-** Usually used in hidden layers of a neural network as it's values lies between **-1 to 1** hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.

RELU Function

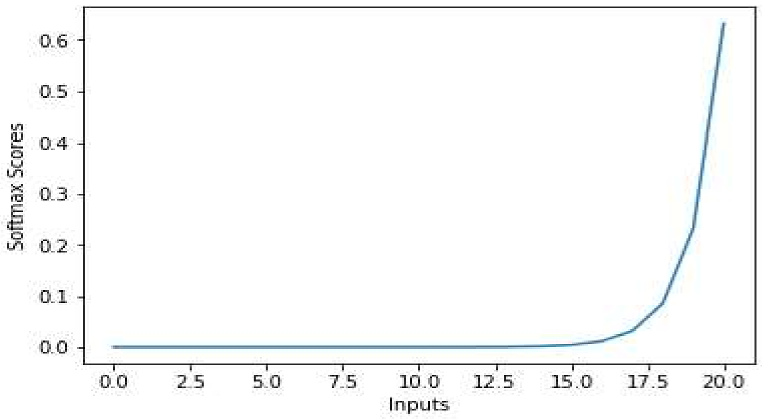


- It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.
- **Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range :-** [0, inf)
- **Nature :-** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- **Uses :-** ReLu is less computationally expensive than tanh and sigmoid because it involves

simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

In simple words, RELU learns *much faster* than sigmoid and Tanh function.

Softmax Function



The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi- class classification problems.

- **Nature :-** non-linear
- **Uses :-** Usually used when trying to handle multiple classes. the softmax function was commonly found in the output layer of image classification problems.The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
- **Output:-** The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.
- The basic rule of thumb is if you really don't know what activation function to use, then simply use *RELU* as it is a general activation function in hidden layers and is used in most cases these days.
- If your output is for binary classification then, *sigmoid function* is very natural choice for output layer.
- If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each classes.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('seaborn')
plt.figure(figsize=(8,4))
def SigmoidBinary(t):
    return 1/(1+np.exp(-t))
t = np.linspace(-5, 5)
plt.plot(t, SigmoidBinary(t))
plt.title('Binary Sigmoid Activation Function')
plt.show()
```

Code:

1. **Binary step function :**
   f(x) = 1, x >= 0
   f(x) = 0, x < 0

```
def binary_step(x):
  if x<0:
    return 0
  else:
    return 1
```

2. **Linear Function :**
   f(x)=ax

```
def linear_function(x):
  return 4*x
linear_function(4), linear_function(-2), linear_function(3)
```

3. **Sigmoid Activation Function:**
   f(x) = 1/(1+e^-x)

```
import numpy as np
def sigmoid_function(x):
  z = (1/(1 + np.exp(-x)))
  return z
sigmoid_function(7), sigmoid_function(-22), sigmoid_function(0)
```

4. **Tanh Activation Function :**
   tanh(x)=2 sigmoid(2x)-1
   tanh(x)=2/(1+e^(2x))-1

```
import numpy as np
def tanh_function(x):
  z = (2/(1 + np.exp(-2*x)))-1
  return z
tanh_function(0.5), tanh_function(-1), tanh_function(0)
```

5. **ReLU Activation Function :**
   f(x)=max(0,x)

```
def relu_function(x):
  if x<0:
    return 0
  else:
    return x
relu_function(7), relu_function(-7)
```

6. **Leaky ReLU :**
   f(x)=0.01x, x < 0,
   f(x)=x, x >0

```
def leaky_relu_function(x):
  if x<0:
    return 0.01*x
  else:
    return x
leaky_relu_function(7), leaky_relu_function(-7)
```

7. **Parameterised relu_function :**
   f(x) = x, x >= 0 ,
   f(x)=ax, x < 0

```
def parameterised_relu_function(x):
  if x<0:
    return 0.2*x
  else:
    return x
parameterised_relu_function(7),parameterised_relu_function(-7)
```

8. **Exponential Linear Function:**
   f(x) = x, x >= 0 ,
   f(x) = a(e^x-1), x<0

```
def elu_function(x,a):
  if x<0:
    return a*(np.exp(x)-1)
  else:
    return x
elu_function(5, 0.1), elu_function(-5,0.1)
```

9. Swish :

f(x)= x * sigmoid(x)
f(x) = x/(1/e^-x)

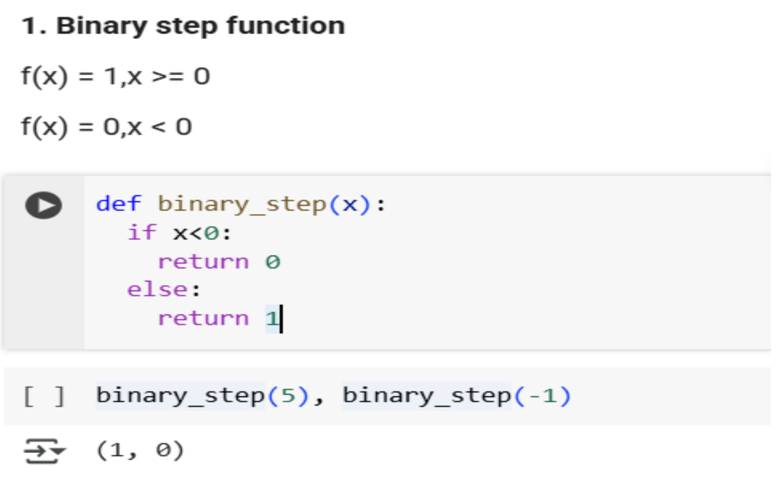
```
def swish_function(x):
  return x/(1 + np.exp(-x))
swish_function(5), swish_function(-5)
```

10. Softmax :

$$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

```
def softmax_function(x):
  z = np.exp(x)
  z_ = z/z.sum()
  return z_
softmax_function([0.8,1.2,3.1])
```

OUTPUT (SCREENSHOT):



1. Binary step function

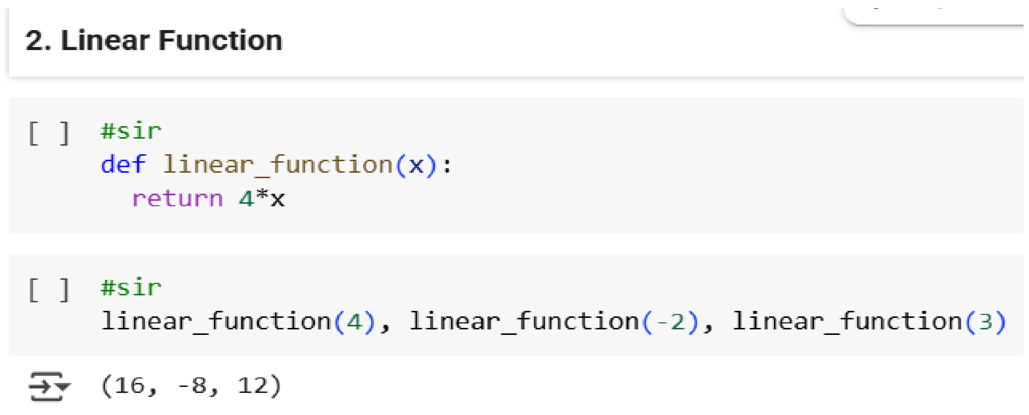
f(x) = 1,x >= 0

f(x) = 0,x < 0

```
def binary_step(x):
  if x<0:
    return 0
  else:
    return 1
```

```
binary_step(5), binary_step(-1)
```

```
(1, 0)
```

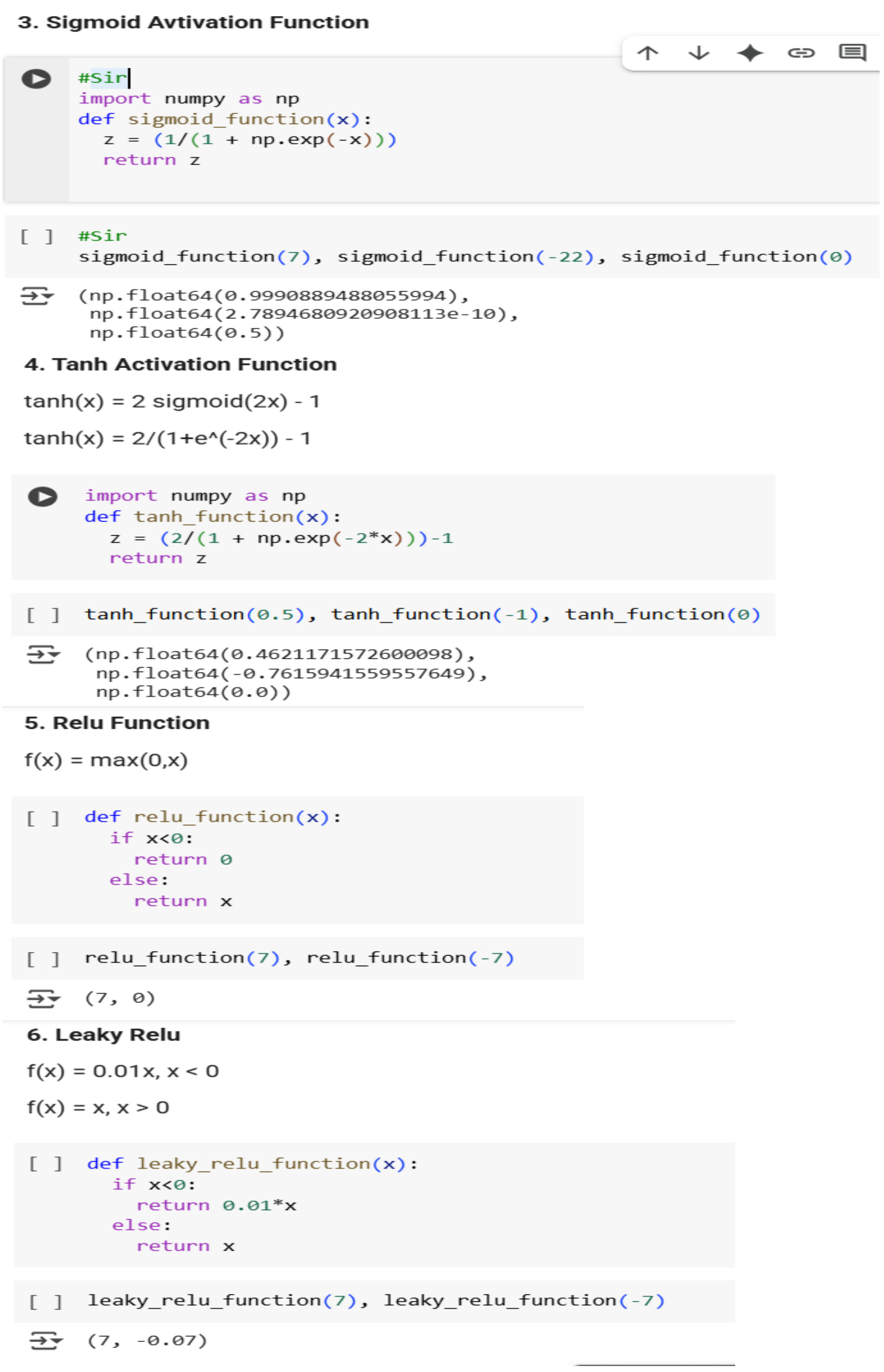


2. Linear Function

```
#sir
def linear_function(x):
  return 4*x
```

```
#sir
linear_function(4), linear_function(-2), linear_function(3)
```

```
(16, -8, 12)
```



3. Sigmoid Avtivation Function

```
#Sir
import numpy as np
def sigmoid_function(x):
  z = (1/(1 + np.exp(-x)))
  return z
```

```
[ ] #Sir
    sigmoid_function(7), sigmoid_function(-22), sigmoid_function(0)
```

```
(np.float64(0.9990889488055994),
 np.float64(2.7894680920908113e-10),
 np.float64(0.5))
```

4. Tanh Activation Function

tanh(x) = 2 sigmoid(2x) - 1

tanh(x) = 2/(1+e^(-2x)) - 1

```
import numpy as np
def tanh_function(x):
  z = (2/(1 + np.exp(-2*x)))-1
  return z
```

```
[ ] tanh_function(0.5), tanh_function(-1), tanh_function(0)
```

```
(np.float64(0.4621171572600098),
 np.float64(-0.7615941559557649),
 np.float64(0.0))
```

5. Relu Function

f(x) = max(0,x)

```
[ ] def relu_function(x):
      if x<0:
        return 0
      else:
        return x
```

```
[ ] relu_function(7), relu_function(-7)
```

```
(7, 0)
```

6. Leaky Relu

f(x) = 0.01x, x < 0

f(x) = x, x > 0

```
[ ] def leaky_relu_function(x):
      if x<0:
        return 0.01*x
      else:
        return x
```

```
[ ] leaky_relu_function(7), leaky_relu_function(-7)
```

```
(7, -0.07)
```

7. Parameterised Relu

f(x) = x, x >= 0

f(x) = ax , x < 0

```
[ ]  def parameterised_relu_function(x):
       if x<0:
         return 0.2*x
       else:
         return x
```

```
[ ]  parameterised_relu_function(7),parameterised_relu_function(-7)
```

```
     (7, -1.4000000000000001)
```

8. Exponential Linear Function

f(x) = x, x >= 0

f(x) = a(e^x-1),x < 0

```
[ ]  def elu_function(x,a):
       if x<0:
         return a*(np.exp(x)-1)
       else:
         return x
```

```
[ ]  elu_function(5, 0.1), elu_function(-5,0.1)
```

```
     (5, np.float64(-0.09932620530009145))
```

9. Swish

Discoverd by researchers at Google. Swish is as computationally efficient as ReLu ans shows better performance than ReLu on deeper models. The values for swish rand=ges from negative infinity to infinity. The function iws defined by:

f(x) = x*sigmoid(x)

f(x) x/(1+e^-x)

```
[ ]  def swish_function(x):
       return x/(1 + np.exp(-x))
```

```
[ ]  swish_function(5), swish_function(-5)
```

```
     (np.float64(4.966535745378576), np.float64(-0.03346425462142428))
```

10. SoftMax

```
[ ]  def softmax_function(x):
       z = np.exp(x)
       z_ = z/z.sum()
       return z_
```

```
[ ]  softmax_function([0.8,1.2,3.1])
```

```
     array([0.08021815, 0.11967141, 0.80011044])
```

CONCLUSION:

Thus Successfully apply the knowledge to study and implement Activation functions in Deep Learning.

DISCUSSION AND VIVA VOCE:

- What is the Activation function?
- How many types of Activation functions?
- Can you briefly talk about Activation Function?
- What are the applications of Activation Function?
- Why do we use the Activation function in Neural Network?

REFERENCE:

- https://www.google.com/search?q=linear+regression+machine+learning&oq=Linear+Regression&aqs=chrome.1.0i433i512j0i512j69i59j0i131i433i512l2j69i60l3.4688j0j7&sourceid=chrome&ie=UTF-8
- https://pubs.acs.org/doi/10.1021/acsomega.2c00362
- https://colab.research.google.com/drive/1gbyjW-RgdgVkUNyRkHAWmjMr7RUCUrYX