Vishwakarma Institute of Technology, Pune *Artificial Neural Networks Lab*

Name: Tanishq Thuse *PRN-* 12310237 *Rollno.* 52 *Class- TY CSAI- B*

Experiment Number: 01

Title: Write a program to visualize popular activation functions used in neural networks, aiding in understanding their behaviour and suitability for different tasks

Title of Experimentation	CO Mapping	CO-Statements	PO Mapping
Write a program to visualize popular activation functions used in neural networks, aiding in understanding their behaviour and suitability for different tasks	CO1	Understand the role of activation functions in neural networks.	PO1 PO2 PO3

Objective

To write a program that visualizes commonly used activation functions in neural networks, enabling better understanding of their mathematical behaviour, graphical representation, and suitability for different learning tasks.

Theory

Activation functions introduce non-linearity in neural networks and help them learn complex mappings. Common activation functions include:

1. **Step Function**
   - Outputs 0 or 1 depending on threshold.
   - Suitable for simple binary classification.
2. **Sigmoid Function**
   - Formula: f(x)=11+e−xf(x) = \frac{1}{1 + e^{-x}}f(x)=1+e−x1
   - Range: (0, 1)
   - Used in early neural networks, but suffers from vanishing gradient.
3. **Hyperbolic Tangent (tanh)**

- Formula: f(x)=tanh(x)f(x) = \tanh(x)f(x)=tanh(x)
- Range: (-1, 1)
- Zero-centered, better than sigmoid.

4. **ReLU (Rectified Linear Unit)**
    - Formula: f(x)=max(0,x)f(x) = \max(0, x)f(x)=max(0,x)
    - Range: [0, ∞)
    - Fast, widely used, but suffers from "dying ReLU" problem.
5. **Leaky ReLU**
    - Formula: f(x)=xf(x) = xf(x)=x if x>0x > 0x>0, else αx\alpha xαx
    - Prevents neurons from dying by allowing small negative slope.
6. **Softmax**
    - Converts logits into probabilities.
    - Used in the output layer of multi-class classification.

Algorithm

1. Define mathematical formulas for activation functions.
2. Generate a range of input values (e.g., -10 to +10).
3. Compute the corresponding output values using activation functions.
4. Plot the graphs of these functions.
5. Compare their behaviour and suitability.

Program (Python - Visualization)

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

```
In [2]: # Define activation functions
        def step_function(x):
            return np.where(x >= 0, 1, 0)
```

```
In [3]: def sigmoid(x):
            return 1 / (1 + np.exp(-x))
```

```
In [4]: def tanh(x):
            return np.tanh(x)
```

```
In [5]: def relu(x):
            return np.maximum(0, x)
```

```
In [6]: def leaky_relu(x, alpha=0.01):
            return np.where(x > 0, x, alpha * x)
```

```
In [7]: # Generate input values
        x = np.linspace(-10, 10, 400)
```

```
In [8]: # Plotting
        plt.figure(figsize=(12, 8))
```

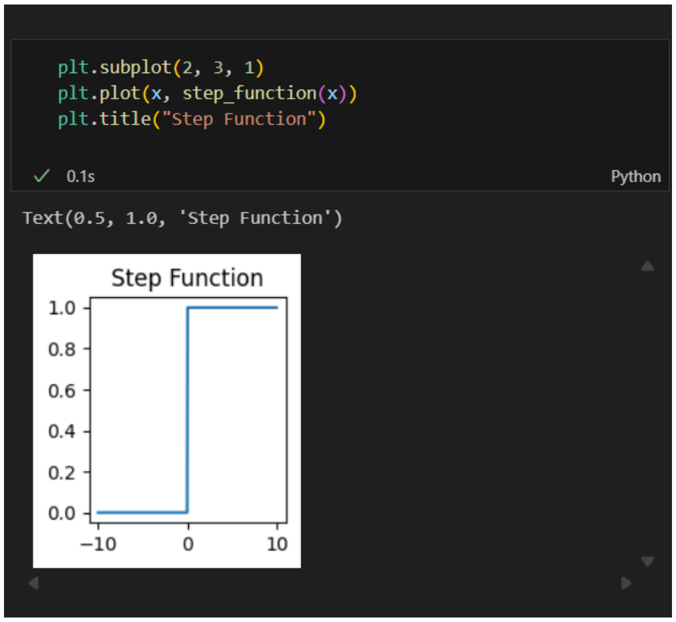
```
Out[8]: <Figure size 1200x800 with 0 Axes>

       <Figure size 1200x800 with 0 Axes>
```

Output

The program generates plots showing different activation functions:

- Step → Binary output

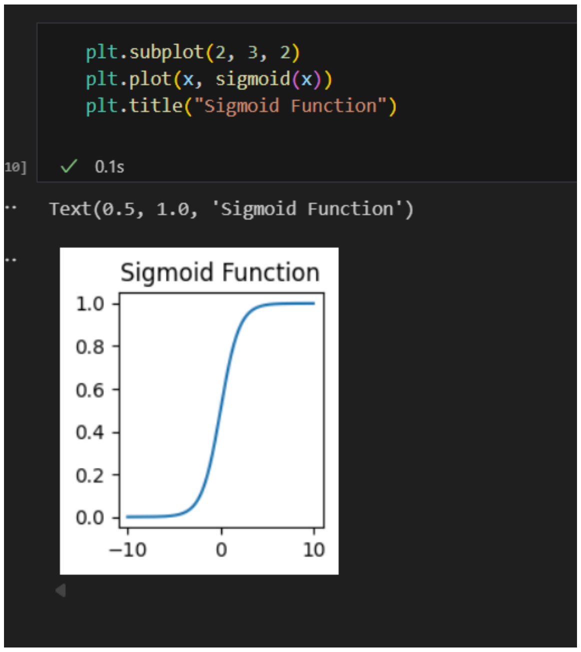


```
plt.subplot(2, 3, 1)
plt.plot(x, step_function(x))
plt.title("Step Function")
```

✓ 0.1s Python

```
Text(0.5, 1.0, 'Step Function')
```

- Sigmoid → Smooth curve (0 to 1)

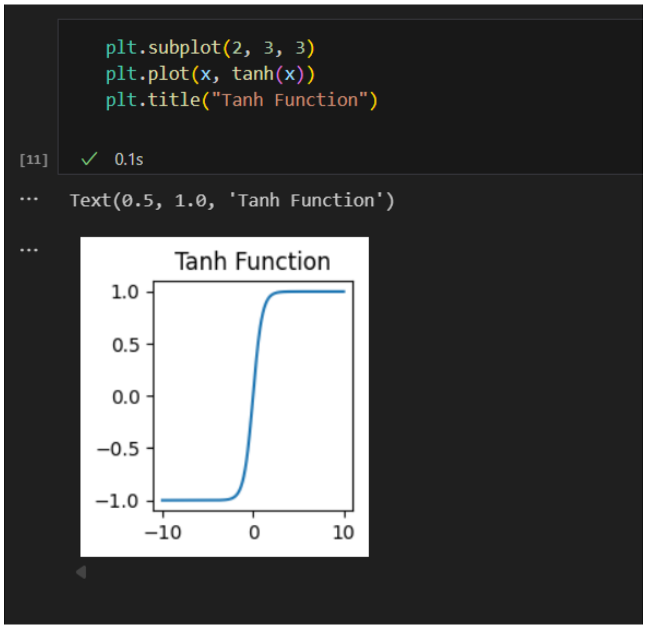


```
plt.subplot(2, 3, 2)
plt.plot(x, sigmoid(x))
plt.title("Sigmoid Function")
```

10] ✓ 0.1s

```
Text(0.5, 1.0, 'Sigmoid Function')
```

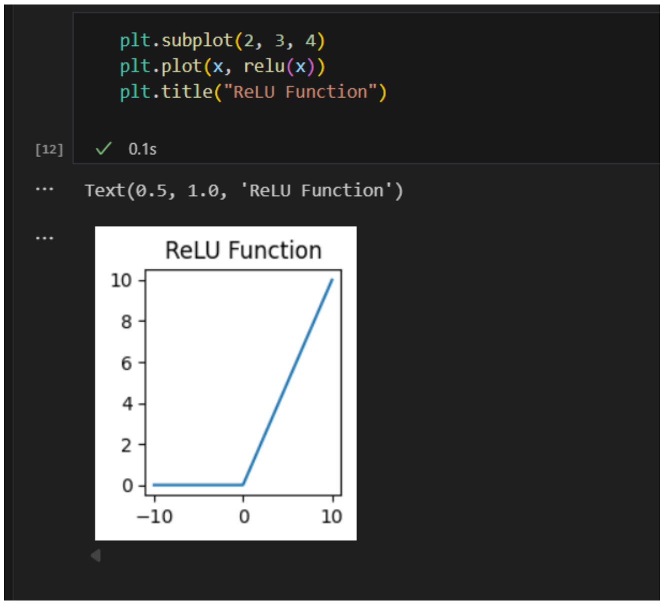
- Tanh → S-shaped (-1 to 1)



```
plt.subplot(2, 3, 3)
plt.plot(x, tanh(x))
plt.title("Tanh Function")
```

```
[11] ✓ 0.1s
Text(0.5, 1.0, 'Tanh Function')
```

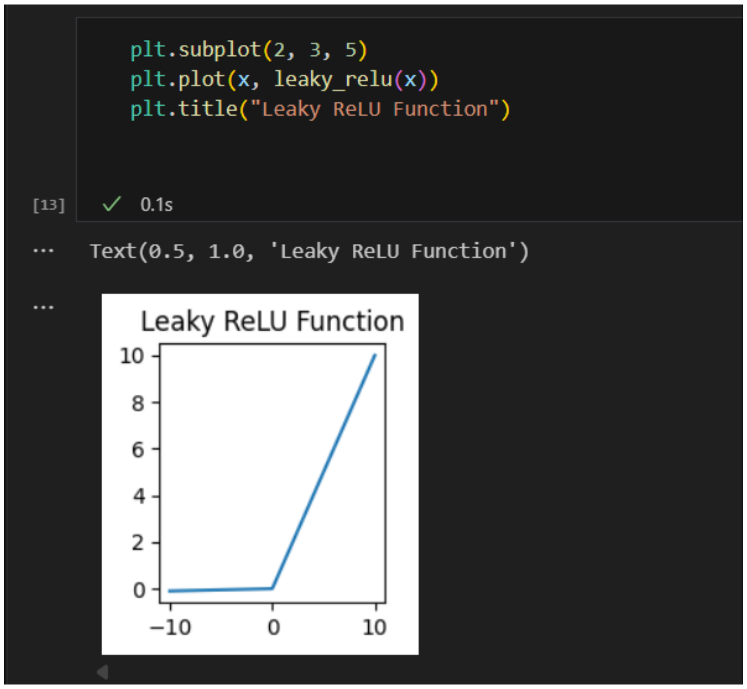
- ReLU → Linear for positive inputs



```
plt.subplot(2, 3, 4)
plt.plot(x, relu(x))
plt.title("ReLU Function")
```

```
[12] ✓ 0.1s
Text(0.5, 1.0, 'ReLU Function')
```

- Leaky ReLU → Allows small negative slope



```
plt.subplot(2, 3, 5)
plt.plot(x, leaky_relu(x))
plt.title("Leaky ReLU Function")
```

[13] ✓ 0.1s

```
Text(0.5, 1.0, 'Leaky ReLU Function')
```

Conclusion

- Activation functions are crucial for introducing non-linearity in neural networks.
- Sigmoid and Tanh are suitable for small networks but suffer from vanishing gradients.
- ReLU is widely used for deep networks due to efficiency.
- Leaky ReLU solves the dying ReLU issue.
- Visualization aids in understanding their differences and appropriate usage.