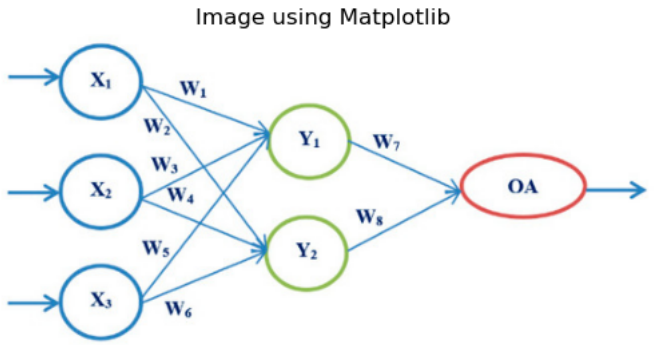
Activation Function

- An **activation function** is a mathematical function used in neural networks to decide whether a neuron should be activated or not.
- It **introduces non-linearity** into the model, helping the network learn complex patterns and relationships in data.
- Activation functions decide whether a neuron should be activated based on the weighted sum of inputs and a bias term. They also make backpropagation possible by providing gradients for weight updates.

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# Give path
img = mpimg.imread("WhatsApp Image 2025-10-05 at 8.33.49 PM.jpeg")

# Display
plt.imshow(img)
plt.title("Image using Matplotlib")
plt.axis("off")
plt.show()
```



1- Sigmoid Function:

The sigmoid activation function, also known as the logistic function. It transforms an input value into a range between 0 and 1.Use in binary classification problems.Suffers from Gradient vanishing problem.

The sigmoid function has an S-shaped curve and is defined as $f(x) = \frac{1}{1+e^{-x}}$

where f(x) represents the sigmoid output for a given input x and e is the mathematical constant approximately equal to 2.71828.
$f'(x) = f(x).(1 - f(x))$

```
import math

def sigmoid(x):
  return 1 / (1 + math.exp(-x))
sigmoid(0.8)
```

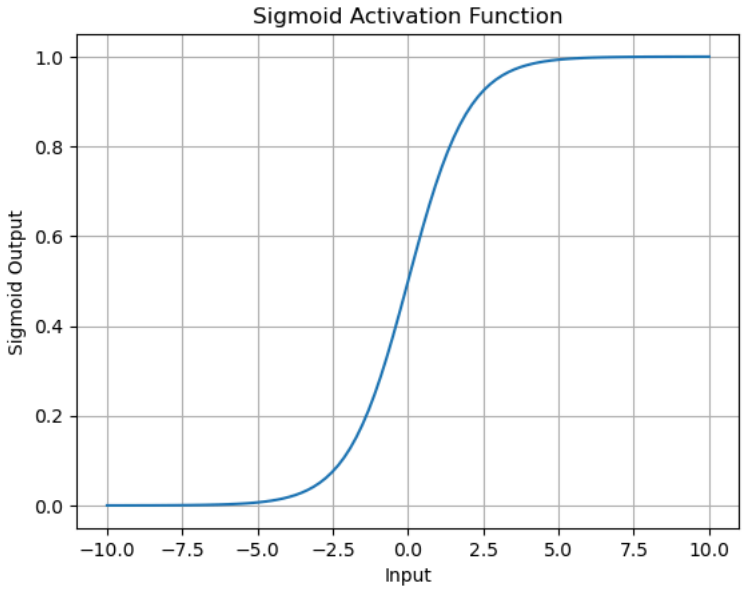
```
0.6899744811276125
```

Python code to plot Sigmoid:

```
def plot_sigmoid():
    x = np.linspace(-10, 10, 100)  # Generate 100 equally spaced values from -10 to 10
    y = 1 / (1 + np.exp(-x))  # Compute the sigmoid function values

    plt.plot(x, y)
    plt.xlabel('Input')
    plt.ylabel('Sigmoid Output')
    plt.title('Sigmoid Activation Function')
    plt.grid(True)
    plt.show()
```

```
# Call the function to plot the sigmoid function
plot_sigmoid()
```



2-Tanh Function:

The hyperbolic tangent (tanh) activation function is a mathematical function commonly used in neural networks. This Function is zero centered.

It is an extension of the sigmoid function but with better gradient properties and also maps the input to a range between -1 and 1.

$f(x) = \tanh(x)$

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

$f'(x) = 1 - (f(x))^2$

```
def tanh(x):
  return (math.exp(x) - math.exp(-x)) / (math.exp(x) + math.exp(-x))
tanh(1)
```

```
0.7615941559557649
```

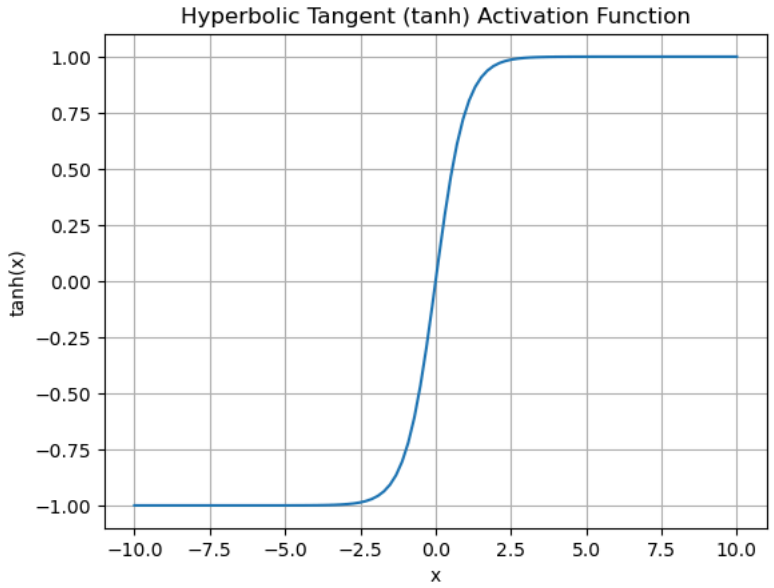
Python code to plot Tanh:

```
def plot_tanh():
    # Generate values for x
    x = np.linspace(-10, 10, 100)

    # Compute tanh values for corresponding x
    tanh = np.tanh(x)

    # Plot the tanh function
    plt.plot(x, tanh)
    plt.title("Hyperbolic Tangent (tanh) Activation Function")
    plt.xlabel("x")
    plt.ylabel("tanh(x)")
    plt.grid(True)
    plt.show()

# Call the function to plot the tanh function
plot_tanh()
```



3-Relu Function:

The **ReLU** (Rectified Linear Unit) is one of the most widely used activation functions in deep learning because of its simplicity and efficiency. It applies a simple thresholding operation to the input values, transforming negative values to zero and leaving positive values unchanged.used in hidden layers to introduce non linearity while keeping computation efficient.

Mathematically, the ReLU function can be defined as $f(x) = \max(0, x)$

where f(x) represents the ReLU output for a given input x.

f(x) =

$$\begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

Derivative of ReLU

The derivative (used during backpropagation) is given by:

f'(x) =

$$\begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

```
def relu(x):
    return max(0,x)
relu(-78)
```

```
0
```

```
relu(78)
```

```
78
```

Python code to plot ReLu:

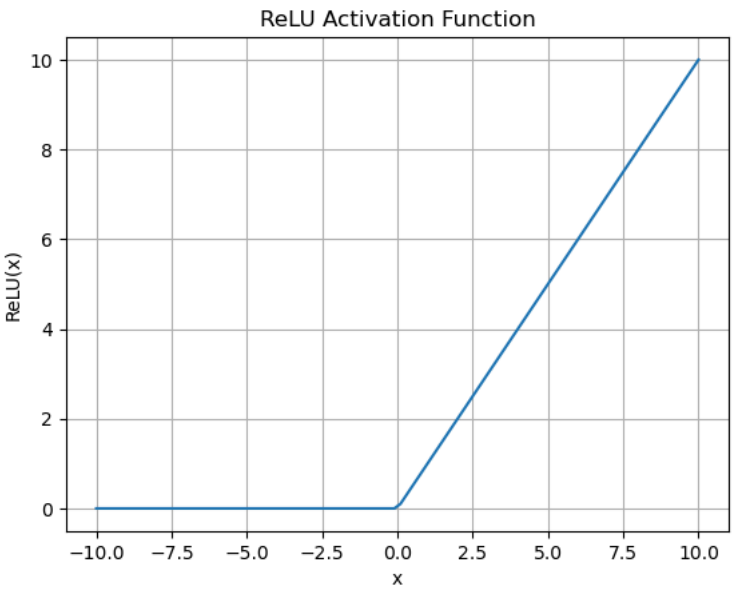
```
def plot_relu():
    # Generate values for x
    x = np.linspace(-10, 10, 100)

    # Compute ReLU values for corresponding x
    relu = np.maximum(0, x)

    # Plot the ReLU function
    plt.plot(x, relu)
    plt.title("ReLU Activation Function")
    plt.xlabel("x")
    plt.ylabel("ReLU(x)")
    plt.grid(True)
```

```
    plt.show()

# Call the function to plot the ReLU function
plot_relu()
```



4-Leaky Relu(Parametric Relu):

The Leaky ReLU activation function is a variation of the Rectified Linear Unit (ReLU) activation function.

Similar to ReLU, it applies a thresholding operation to the input values, setting negative values to zero.

However, unlike ReLU, the Leaky ReLU allows a small, non-zero gradient for negative inputs, which helps address the "dying ReLU" problem where neurons can become non-responsive.

Mathematically, the Leaky ReLU function can be defined as

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.01x, & \text{if } x \leq 0 \end{cases}$$

where f(x) represents the Leaky ReLU output for a given input x,

and a is a small positive constant (typically around 0.1).

```
# Define the leaky ReLU function
def leaky_relu(x, alpha=0.1):
    return max(x, alpha * x)

# Test with input -100
output = leaky_relu(-89)
print(output)
```

```
-8.9
```

```
leaky_relu(100)
```

```
100
```

Python Code to Plot Leaky_ReLu

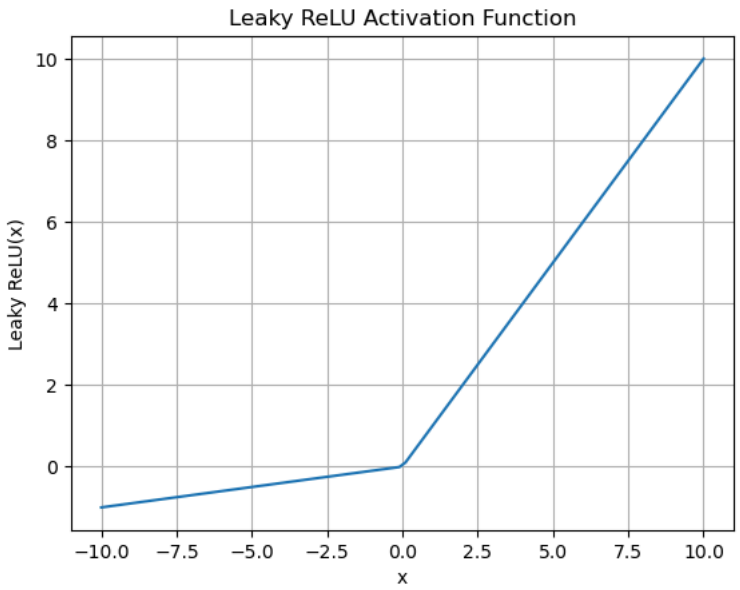
```
def plot_leaky_relu():
    # Generate values for x
    x = np.linspace(-10, 10, 100)

    # Define the leaky ReLU function
    def leaky_relu(x, alpha=0.1):
        return np.where(x >= 0, x, alpha * x)

    # Compute leaky ReLU values for corresponding x
    leaky_relu_values = leaky_relu(x)
```

```
    # Plot the leaky ReLU function
    plt.plot(x, leaky_relu_values)
    plt.title("Leaky ReLU Activation Function")
    plt.xlabel("x")
    plt.ylabel("Leaky ReLU(x)")
    plt.grid(True)
    plt.show()

# Call the function to plot the Leaky ReLU function
plot_leaky_relu()
```



5-Softmax:

The softmax function is a mathematical function used in machine learning and neural networks.used fot multi-class classification.

$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$

It takes a vector of real numbers as input and outputs a probability distribution over multiple classes.

The softmax function normalizes the input values and transforms them into probabilities that sum up to 1.

It is commonly used as the activation function in the output layer of a neural network for multi-class classification tasks.

The softmax function exponentiates each input value, divides it by the sum of all exponentiated values, and produces a probability value for each class.

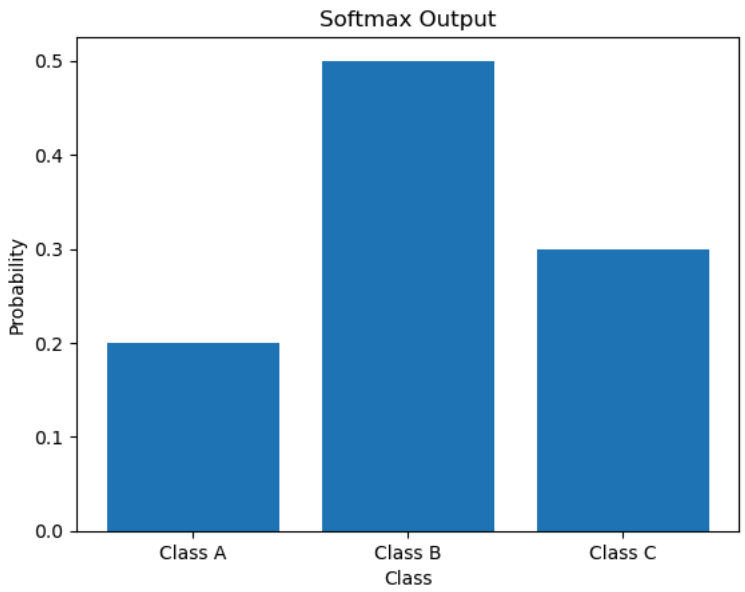
```
import numpy as np

def softmax(x):
    # Subtracting the maximum value for numerical stability
    e_x = np.exp(x - np.max(x))
    return e_x / np.sum(e_x, axis=0)

# Example usage:
x = np.array([8, 7, 3])
result = softmax(x)
print(result)
```

```
[0.72747516 0.26762315 0.00490169]
```

```
def plot_softmax(probabilities, class_labels):
    plt.bar(class_labels, probabilities)
    plt.xlabel("Class")
    plt.ylabel("Probability")
    plt.title("Softmax Output")
    plt.show()
# Example usage:
class_labels = ["Class A", "Class B", "Class C"]
probabilities = np.array([0.2, 0.5, 0.3])
plot_softmax(probabilities, class_labels)
```



Lab Tasks

Task 1

Write Python functions for Sigmoid, ReLU, Tanh, and Leaky ReLU activation functions. Plot graphs to visualize how each activation function transforms input values.

Task 2

1. Which activation function is most suitable for **hidden layers** and why?
2. What is the **vanishing gradient problem** and which activation functions suffer from it?
3. What issue does **ReLU** face and how does **Leaky ReLU** solve it?
4. Why is **Softmax** preferred in output layers for classification?

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