Title: - Understanding and Visualizing Activation Functions in Neural Networks

Aim/Objective: - To implement and analyze different activation functions used in neural networks, such as Sigmoid, ReLU, Leaky ReLU, and Softmax, along with their graphical representations.

Software Required:

- Python programming environment (Jupyter Notebook, Google Colab, or any Python IDE)
- NumPy library
- Matplotlib library

Hardware Required:

- 4GB RAM
- Intel i3 or higher / AMD equivalent
- GPU for accelerated computations (if using deep learning frameworks)
- 120 GB SSD

Theory:

In neural networks, an **activation function** is a mathematical function applied to each neuron's output to introduce non-linearity. This helps the network learn complex patterns and relationships.

What is Activation Energy?

Activation energy in the context of neural networks is akin to the concept in physics and chemistry, where a system needs a minimum energy threshold to proceed. Similarly, in a neural network, an activation function determines whether a neuron should be "activated" based on its weighted sum and bias.

Common Activation Functions:

1. **Sigmoid Function**
    - Maps any input to a range between 0 and 1.
    - Useful for binary classification problems.
    - Disadvantage: Prone to **vanishing gradient** problem.

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

2. **Tanh (Hyperbolic Tangent) Function**
    - Outputs values between -1 and 1.
    - Helps in **centering data** but still suffers from vanishing gradients.

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

3. **ReLU (Rectified Linear Unit)**
   - Faster convergence and less computation.
   - Helps mitigate vanishing gradient problem.
   - Disadvantage: Can lead to **dead neurons** when inputs are always negative.

$$f(x)=\max(0,x)$$

4. **Leaky ReLU**
   - Variant of ReLU that allows a small gradient for negative inputs.

$$f(x)=\max(0.01x,x)$$

5. **Softmax Function** (For multi-class classification)
   - Converts logits into probability distributions.

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Procedure:

1. Import the required libraries (NumPy and Matplotlib).
2. Define functions for Sigmoid, ReLU, Leaky ReLU, and Softmax.
3. Generate an array of input values from -5 to 5.
4. Compute the output values for each activation function.
5. Plot the activation functions using Matplotlib.
6. Observe and analyze the behavior of each function.

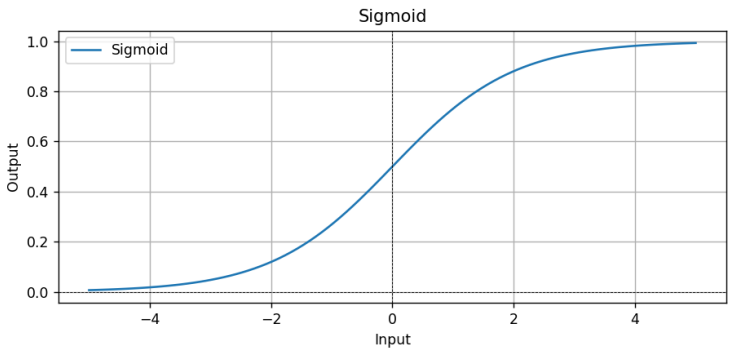
Observations:

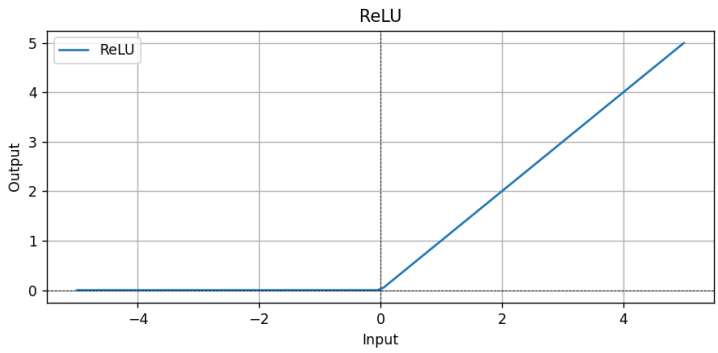
1. Sigmoid outputs values between 0 and 1, making it useful for probability-based predictions.
2. ReLU is zero for negative inputs and linear for positive values, preventing the vanishing gradient problem.
3. Leaky ReLU allows small negative values, improving gradient flow.
4. Softmax converts inputs into a probability distribution, commonly used in classification problems.

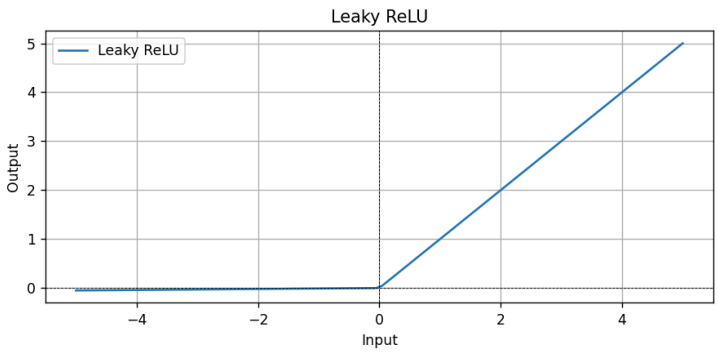
Code:

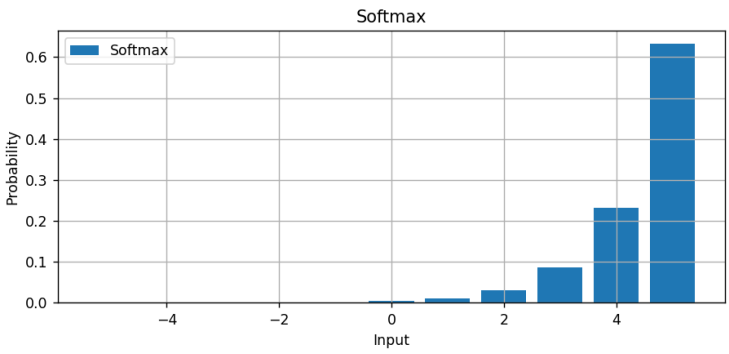
```
import numpy as np
import matplotlib.pyplot as plt

# Sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# ReLU activation function
def relu(x):
    return np.maximum(0, x)

# Leaky ReLU activation function
def leaky_relu(x, alpha=0.01):
    return np.maximum(alpha * x, x)

# Softmax activation function
def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=-1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=-1, keepdims=True)

# Generate input values
x = np.linspace(-5, 5, 100)

# Compute activation values
y_sigmoid = sigmoid(x)
y_relu = relu(x)
y_leaky_relu = leaky_relu(x)

def plot_activation(x, y, title):
    plt.plot(x, y, label=title)
    plt.axhline(0, color='black', linewidth=0.5, linestyle='dashed')
    plt.axvline(0, color='black', linewidth=0.5, linestyle='dashed')
    plt.title(title)
    plt.xlabel('Input')
    plt.ylabel('Output')
    plt.legend()
    plt.grid()

# Plot activation functions
plt.figure(figsize=(10, 6))

plt.subplot(2, 2, 1)
plot_activation(x, y_sigmoid, 'Sigmoid')

plt.subplot(2, 2, 2)
plot_activation(x, y_relu, 'ReLU')
```

```
plt.subplot(2, 2, 3)
plot_activation(x, y_leaky_relu, 'Leaky ReLU')

# Softmax requires a different approach since it's multi-dimensional
x_softmax = np.array([[i] for i in range(-5, 6)])  # 1D input for softmax
y_softmax = softmax(x_softmax.T).T  # Compute softmax

plt.subplot(2, 2, 4)
plt.bar(range(-5, 6), y_softmax.flatten(), label='Softmax')
plt.title('Softmax')
plt.xlabel('Input')
plt.ylabel('Probability')
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()
```









Conclusion:

Hence, we learned that Activation functions play a crucial role in neural networks by introducing non-linearity and improving model learning. Each function has its use case depending on the problem at hand. The plotted graphs provide a clear visual understanding of how each activation function transforms inputs, helping in selecting the right function for specific applications. This conclude successfully implementation of activation functions.