Deep Learning Lab - Activation Functions

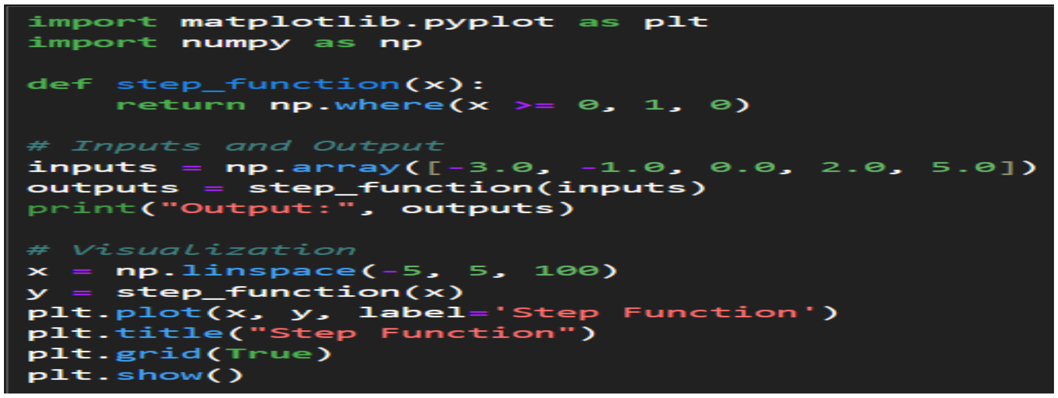
Task 1: Step Function

Goal: Implement a step function that outputs 1 if input ≥ 0, else 0.

Input Values: [-3.0, -1.0, 0.0, 2.0, 5.0]

Expected Output: [0, 0, 1, 1, 1]

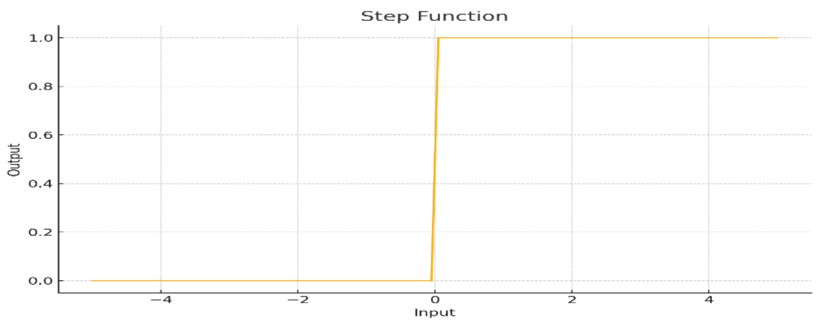
CODE:



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

def step_function(x):
    return np.where(x >= 0, 1, 0)

# Inputs and Output
inputs = np.array([-3.0, -1.0, 0.0, 2.0, 5.0])
outputs = step_function(inputs)
print("Output:", outputs)

# Visualization
x = np.linspace(-5, 5, 100)
y = step_function(x)
plt.plot(x, y, label='Step Function')
plt.title("Step Function")
plt.grid(True)
plt.show()
```

OUTPUT:

Output: [0 0 1 1 1]



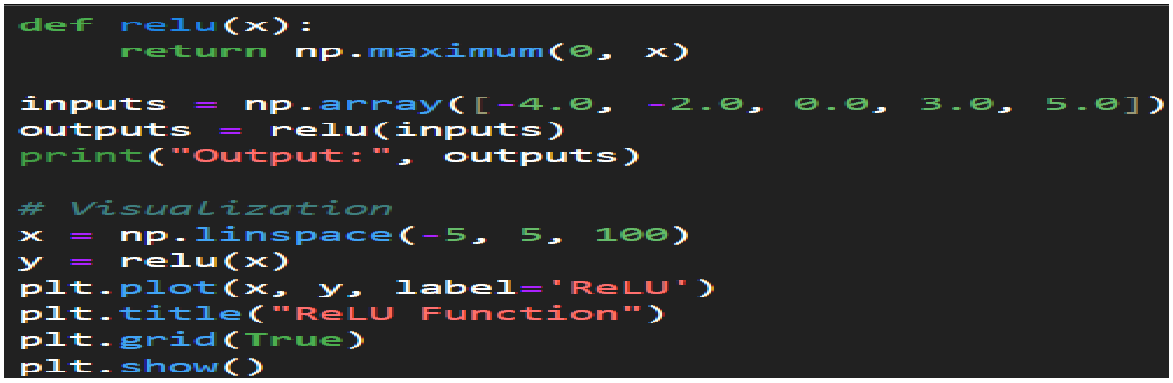
Task 2: ReLU

Goal: Implement ReLU: f(x) = max(0, x).

Input Values: [-4.0, -2.0, 0.0, 3.0, 5.0]

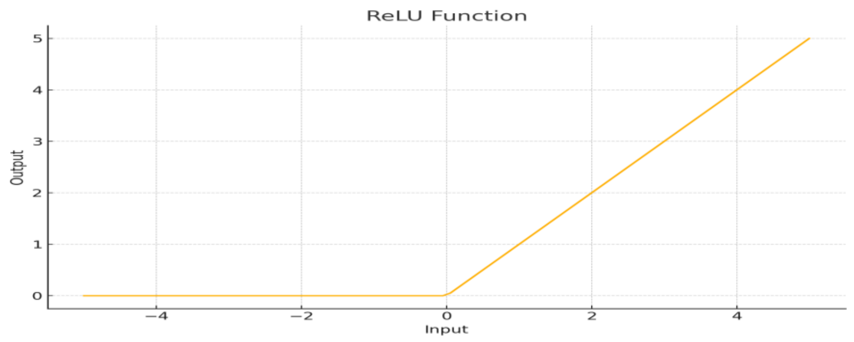
Expected Output: [0, 0, 0, 3.0, 5.0]

CODE:



```python
def relu(x):
    return np.maximum(0, x)

inputs = np.array([-4.0, -2.0, 0.0, 3.0, 5.0])
outputs = relu(inputs)
print("Output:", outputs)

# Visualization
x = np.linspace(-5, 5, 100)
y = relu(x)
plt.plot(x, y, label='ReLU')
plt.title("ReLU Function")
plt.grid(True)
plt.show()
```

Output: [0. 0. 0. 3. 5.]



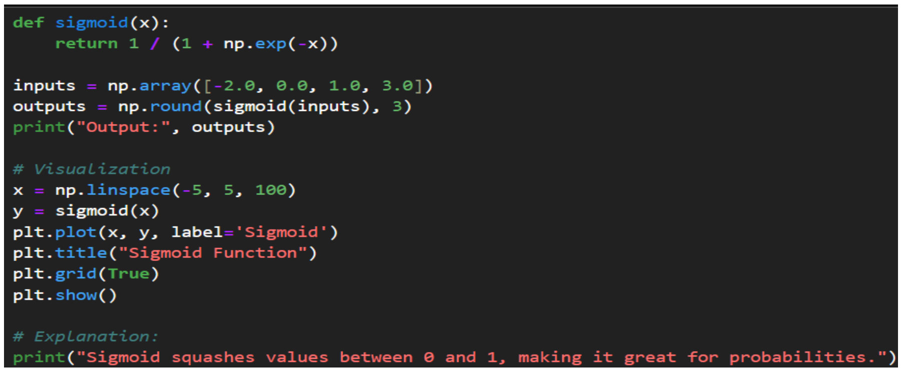
Task 3: Sigmoid

Goal: Implement Sigmoid: f(x) = 1 / (1 + e^{-x}).

Input Values: [-2.0, 0.0, 1.0, 3.0]

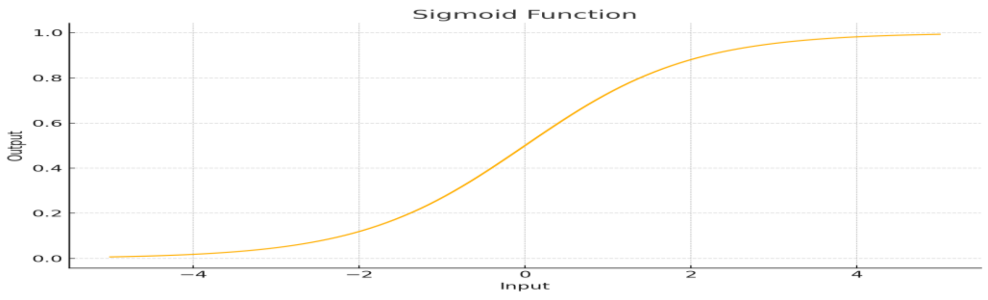
Expected Output: [0.119, 0.5, 0.731, 0.953]

CODE:



```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

inputs = np.array([-2.0, 0.0, 1.0, 3.0])
outputs = np.round(sigmoid(inputs), 3)
print("Output:", outputs)

# Visualization
x = np.linspace(-5, 5, 100)
y = sigmoid(x)
plt.plot(x, y, label='Sigmoid')
plt.title("Sigmoid Function")
plt.grid(True)
plt.show()

# Explanation:
print("Sigmoid squashes values between 0 and 1, making it great for probabilities.")
```

OUTPUT:



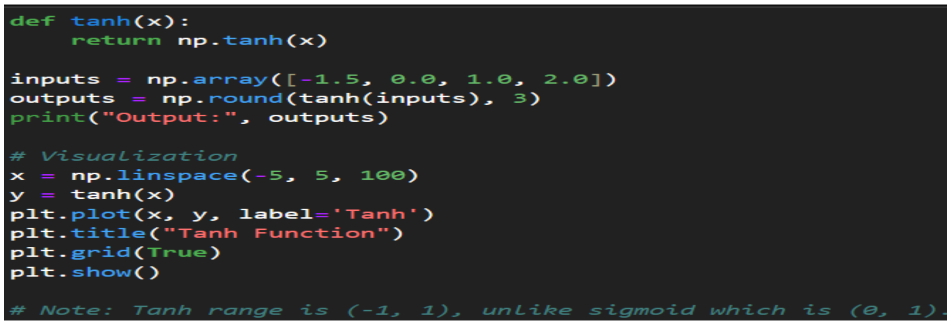
Task 4: Tanh

Goal: Implement Tanh: f(x) = (e^x - e^{-x}) / (e^x + e^{-x}).

Input Values: [-1.5, 0.0, 1.0, 2.0]

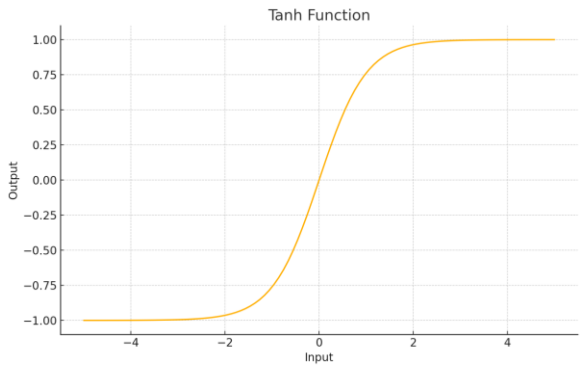
Expected Output: [-0.905, 0.0, 0.762, 0.964]

CODE:



```
def tanh(x):
    return np.tanh(x)

inputs = np.array([-1.5, 0.0, 1.0, 2.0])
outputs = np.round(tanh(inputs), 3)
print("Output:", outputs)

# Visualization
x = np.linspace(-5, 5, 100)
y = tanh(x)
plt.plot(x, y, label='Tanh')
plt.title("Tanh Function")
plt.grid(True)
plt.show()

# Note: Tanh range is (-1, 1), unlike sigmoid which is (0, 1).
```

OUTPUT:



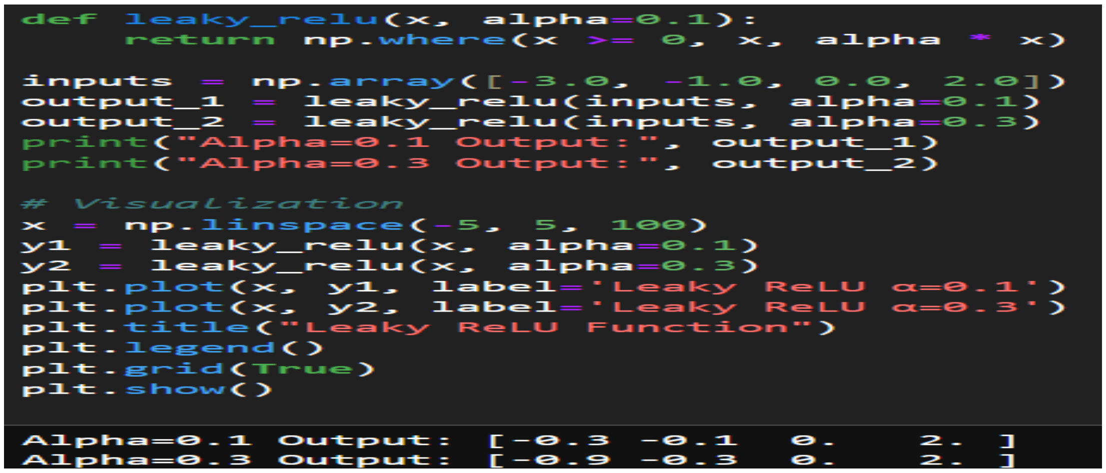
Task 5: Leaky ReLU

Goal: Implement Leaky ReLU: f(x) = x if x ≥ 0 else α*x.

Input Values: [-3.0, -1.0, 0.0, 2.0] | α = 0.1

Expected Output: [-0.30000000000000004, -0.1, 0.0, 2.0]

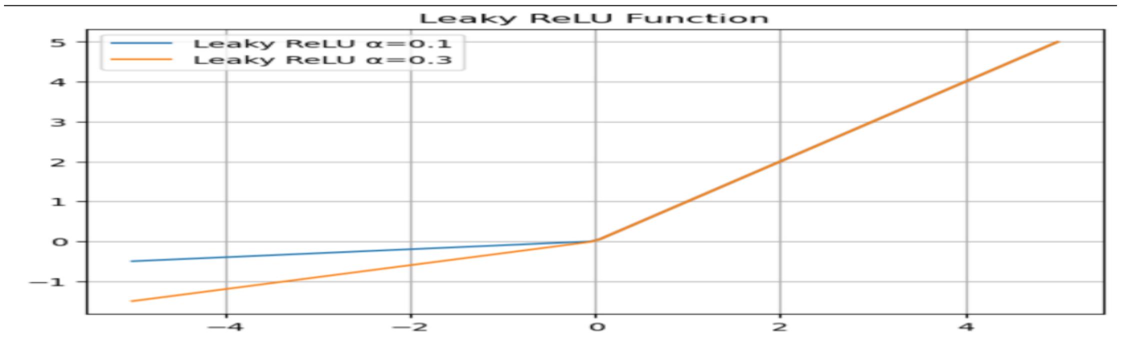
CODE:



```
def leaky_relu(x, alpha=0.1):
    return np.where(x >= 0, x, alpha * x)

inputs = np.array([-3.0, -1.0, 0.0, 2.0])
output_1 = leaky_relu(inputs, alpha=0.1)
output_2 = leaky_relu(inputs, alpha=0.3)
print("Alpha=0.1 Output:", output_1)
print("Alpha=0.3 Output:", output_2)

# Visualization
x = np.linspace(-5, 5, 100)
y1 = leaky_relu(x, alpha=0.1)
y2 = leaky_relu(x, alpha=0.3)
plt.plot(x, y1, label='Leaky ReLU α=0.1')
plt.plot(x, y2, label='Leaky ReLU α=0.3')
plt.title("Leaky ReLU Function")
plt.legend()
plt.grid(True)
plt.show()
```

```
Alpha=0.1 Output: [-0.3 -0.1  0.   2. ]
Alpha=0.3 Output: [-0.9 -0.3  0.   2. ]
```

OUTPUT:

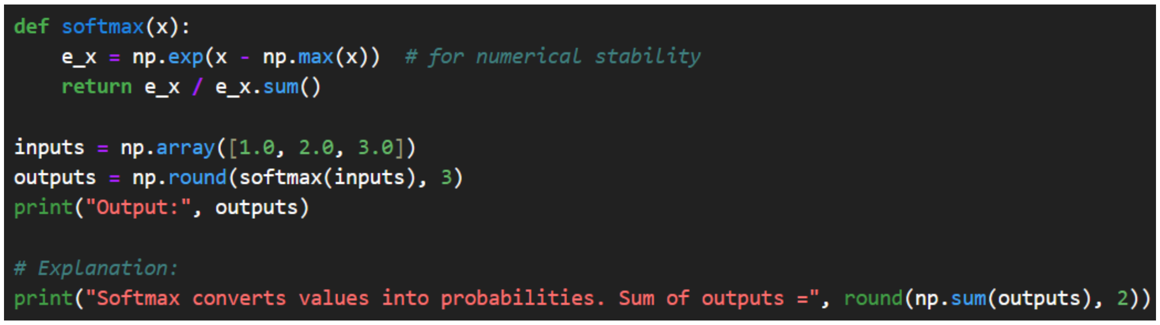


Task 6: Softmax

Goal: Implement Softmax to convert inputs

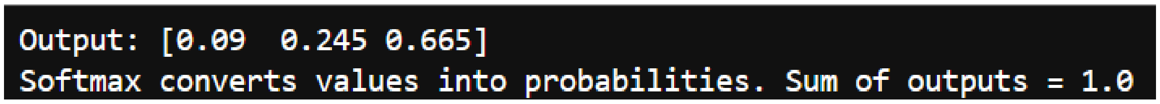
Explanation: Softmax outputs are probabilities that sum to 1.0, making it suitable for multiclass classification.

CODE:



```python
def softmax(x):
    e_x = np.exp(x - np.max(x))  # for numerical stability
    return e_x / e_x.sum()

inputs = np.array([1.0, 2.0, 3.0])
outputs = np.round(softmax(inputs), 3)
print("Output:", outputs)

# Explanation:
print("Softmax converts values into probabilities. Sum of outputs =", round(np.sum(outputs), 2))
```

OUTPUT:



```
Output: [0.09  0.245 0.665]
Softmax converts values into probabilities. Sum of outputs = 1.0
```