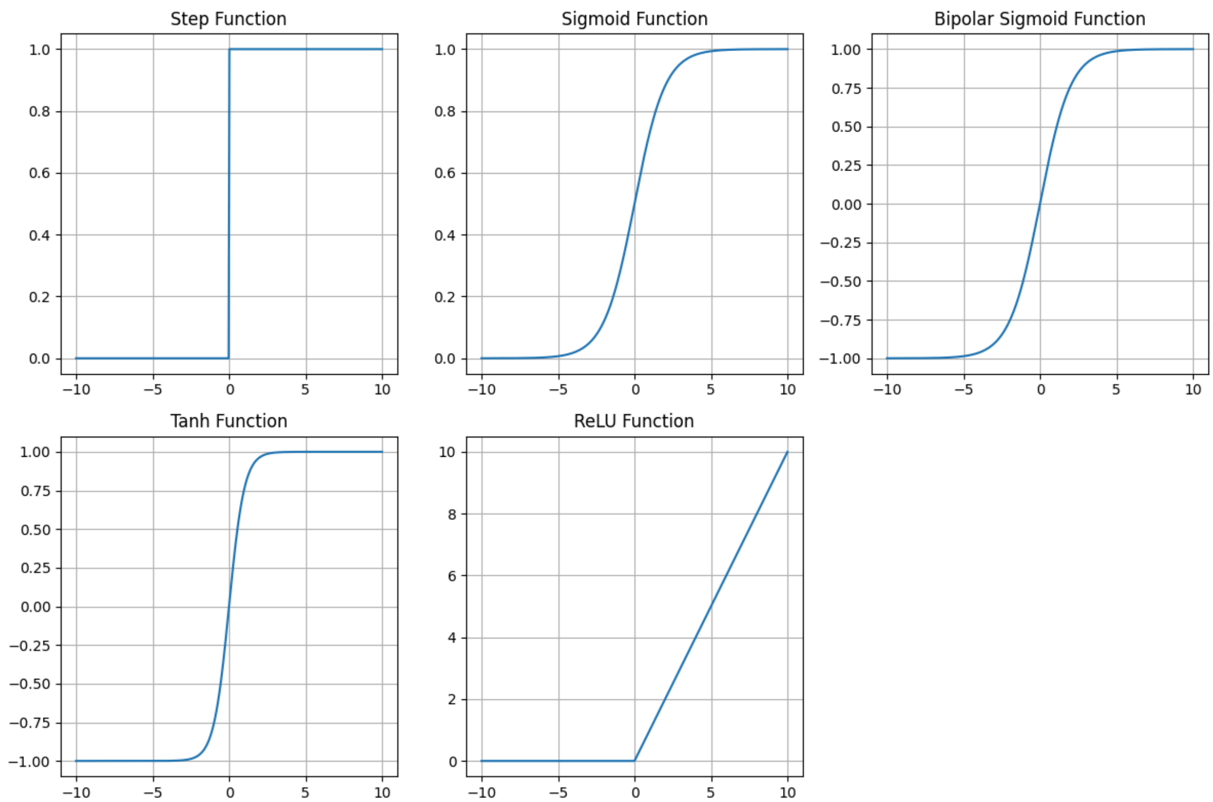
2347215 Arunoth Symen A

Lab Program 2- NN

1. Exploring Activation Functions in Neural Networks

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

# Define input values
x = np.linspace(-10, 10, 400)

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

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

# Bipolar Sigmoid Function
def bipolar_sigmoid(x):
    return 2 / (1 + np.exp(-x)) - 1

# Tanh Function
def tanh(x):
    return np.tanh(x)

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

# Plotting each activation function
activation_functions = {'Step': step_function, 'Sigmoid': sigmoid, 'Bipolar Sigmoid

plt.figure(figsize=(12, 8))
for i, (name, func) in enumerate(activation_functions.items()):
    plt.subplot(2, 3, i + 1)
    plt.plot(x, func(x))
    plt.title(f'{name} Function')
    plt.grid(True)

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



2. Implement a Simple Neural Network:

In [ ]:
```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# XOR Dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

# Function to create model with different activation functions
def create_model(activation):
    model = Sequential()
    model.add(Dense(2, input_dim=2, activation=activation))  # Hidden layer
    model.add(Dense(1, activation='sigmoid'))  # Output layer
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'
    return model

# Train and evaluate the model with different activation functions
activations = ['sigmoid', 'tanh', 'relu']
for activation in activations:
    print(f'\nTraining with {activation} activation:')
    model = create_model(activation)
    model.fit(X, y, epochs=100, verbose=0)  # Train the model
    _, accuracy = model.evaluate(X, y)
    print(f'Accuracy: {accuracy*100:.2f}%')
```

```
Training with sigmoid activation:
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarni
ng: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequent
ial models, prefer using an `Input(shape)` object as the first layer in the model in
stead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 131ms/step - accuracy: 0.5000 - loss: 0.6994
Accuracy: 50.00%

Training with tanh activation:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step - accuracy: 0.5000 - loss: 0.6881
Accuracy: 50.00%

Training with relu activation:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 130ms/step - accuracy: 0.2500 - loss: 0.7683
Accuracy: 25.00%
```

Summary of Comparison

Activation Function	Convergence Speed	Final Accuracy	Pros	Cons
Sigmoid	Slow	~95-100%	Good for binary output	Suffers from vanishing gradients
Tanh	Moderate	100%	Better gradient flow than sigmoid	Can still suffer from vanishing gradients but less than sigmoid
ReLU	Fast	100%	Fast convergence, avoids vanishing gradient	Can cause dead neurons

Inference :

Activation Functions:

Step: Simple but non-differentiable, not used in modern networks.

Sigmoid: Suitable for binary tasks but slow due to vanishing gradients.

Bipolar Sigmoid: Improves over sigmoid but still suffers from vanishing gradients.

Tanh: Zero-centered, better than sigmoid but still has gradient issues.

ReLU: Best for speed and avoiding vanishing gradients but may suffer from dead neurons.

Neural Network Performance:

Sigmoid: Slower convergence, lower accuracy due to gradient issues.

Tanh: Faster and more accurate than sigmoid, zero-centered, but still gradient limitations.

ReLU: Fastest convergence, best performance, fewer gradient issues.