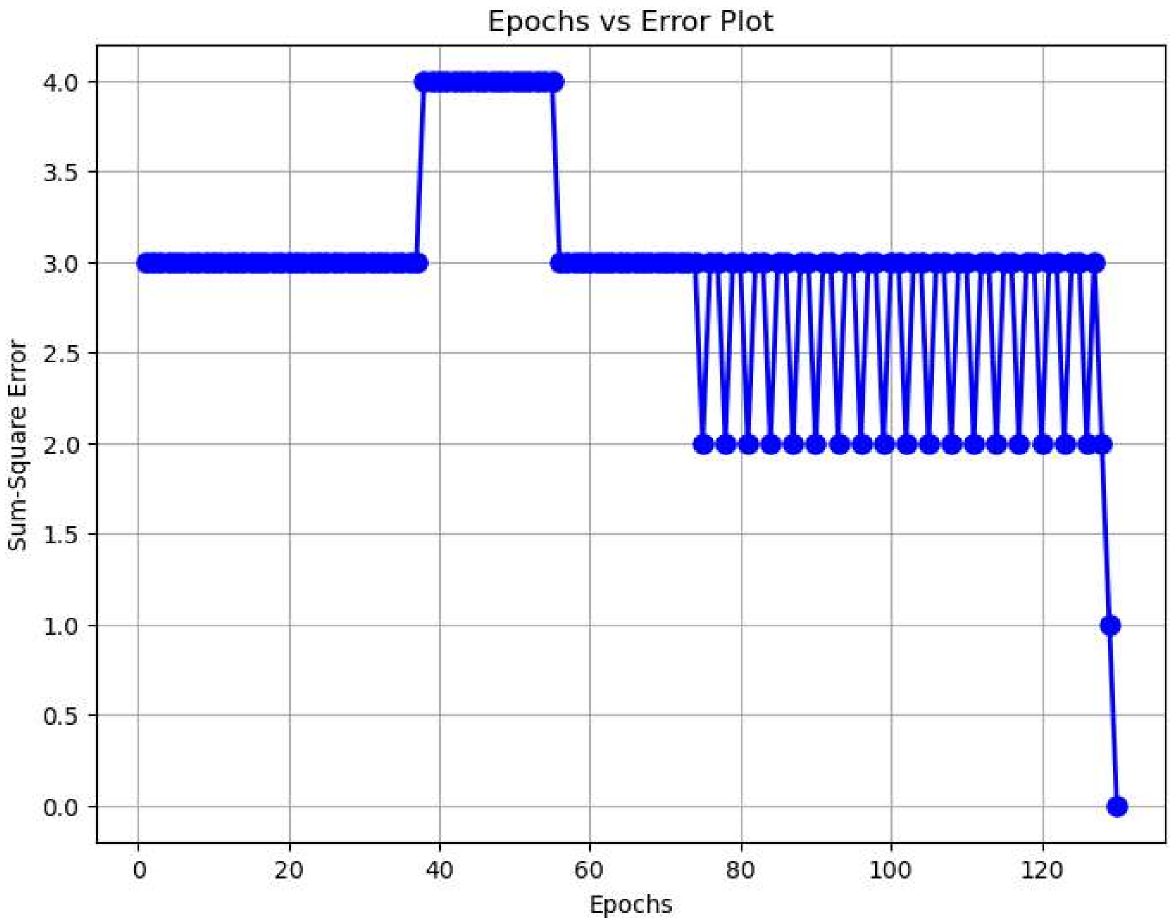
In [2]:

```
import matplotlib.pyplot as plt
import numpy as np
# Initial weights and learning rate
weights = np.array([10, 0.2, -0.75])
learning_rate = 0.05

# AND gate inputs and targets
and_inputs = np.array([[0, 0],
                       [0, 1],
                       [1, 0],
                       [1, 1]])

and_targets = np.array([0, 0, 0, 1])

# Step activation function
def step_function(x):
    return 1 if x >= 0 else 0

# Training the perceptron and recording errors
epochs = 0
errors_list = []
while True:
    errors = 0
    for inputs, target in zip(and_inputs, and_targets):
        # Compute weighted sum
        weighted_sum = np.dot(inputs, weights[1:]) + weights[0]
        # Apply step activation function
        prediction = step_function(weighted_sum)
        # Compute error
        error = target - prediction
        # Update weights
        weights[1:] += learning_rate * error * inputs
        weights[0] += learning_rate * error
        errors += error**2
    errors_list.append(errors)
    epochs += 1
    if errors == 0:
        break

# Plotting epochs against error values
plt.figure(figsize=(8, 6))
plt.plot(range(1, epochs + 1), errors_list, marker='o', color='b', linestyle='
plt.xlabel('Epochs')
plt.ylabel('Sum-Square Error')
plt.title('Epochs vs Error Plot')
plt.grid(True)
plt.show()

# Test the trained perceptron
print("Trained Weights:", weights)
print("Number of epochs needed for convergence:", epochs)

# Test the trained perceptron
test_inputs = np.array([[0, 0],
                        [0, 1],
                        [1, 0],
                        [1, 1]])
```

```python
print("Predictions:")
for inputs in test_inputs:
    weighted_sum = np.dot(inputs, weights[1:]) + weights[0]
    prediction = step_function(weighted_sum)
    print(f"{inputs} -> {prediction}")
```




```
Trained Weights: [-0.1   0.1   0.05]
Number of epochs needed for convergence: 130
Predictions:
[0 0] -> 0
[0 1] -> 0
[1 0] -> 0
[1 1] -> 1
```

In [*]:

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

# Initial weights and learning rate
weights = np.array([10, 0.2, -0.75])
learning_rate = 0.05

# AND gate inputs and targets
and_inputs = np.array([[0, 0],
                       [0, 1],
                       [1, 0],
                       [1, 1]])

and_targets = np.array([0, 0, 0, 1])

#  activation function
def bipolar_step_function(x):
    if x>0:
        return 1
    elif (x==0):
        return 0
    else:
        return -1

# Training the perceptron and recording errors
epochs = 0
errors_list = []
while True:
    errors = 0
    for inputs, target in zip(and_inputs, and_targets):
        # Compute weighted sum
        weighted_sum = np.dot(inputs, weights[1:]) + weights[0]
        # Apply step activation function
        prediction = bipolar_step_function(weighted_sum)
        # Compute error
        error = target - prediction
        # Update weights
        weights[1:] += learning_rate * error * inputs
        weights[0] += learning_rate * error
        errors += error**2
    errors_list.append(errors)
    epochs += 1
    if errors == 0:
        break

# Plotting epochs against error values
plt.figure(figsize=(8, 6))
plt.plot(range(1, epochs + 1), errors_list, marker='o', color='b', linestyle='
plt.xlabel('Epochs')
plt.ylabel('Sum-Square Error')
plt.title('Epochs vs Error Plot')
plt.grid(True)
plt.show()

# Test the trained perceptron
print("Trained Weights:", weights)
print("Number of epochs needed for convergence:", epochs)
```

```python
# Test the trained perceptron
test_inputs = np.array([[0, 0],
                        [0, 1],
                        [1, 0],
                        [1, 1]])

print("Predictions:")
for inputs in test_inputs:
    weighted_sum = np.dot(inputs, weights[1:]) + weights[0]
    prediction = step_function(weighted_sum)
    print(f"{inputs} -> {prediction}")
```

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

# Initial weights and learning rate
weights = np.array([10, 0.2, -0.75])
learning_rate = 0.05

# AND gate inputs and targets
and_inputs = np.array([[0, 0],
                       [0, 1],
                       [1, 0],
                       [1, 1]])

and_targets = np.array([0, 0, 0, 1])

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

# Training the perceptron and recording errors
epochs = 0
errors_list = []
while True:
    errors = 0
    for inputs, target in zip(and_inputs, and_targets):
        # Compute weighted sum
        weighted_sum = np.dot(inputs, weights[1:]) + weights[0]
        # Apply step activation function
        prediction = Sigmoid_function(weighted_sum)
        # Compute error
        error = target - prediction
        # Update weights
        weights[1:] += learning_rate * error * inputs
        weights[0] += learning_rate * error
        errors += error**2
    errors_list.append(errors)
    epochs += 1
    if errors == 0:
        break

# Plotting epochs against error values
plt.figure(figsize=(8, 6))
plt.plot(range(1, epochs + 1), errors_list, marker='o', color='b', linestyle='
plt.xlabel('Epochs')
plt.ylabel('Sum-Square Error')
plt.title('Epochs vs Error Plot')
plt.grid(True)
plt.show()

# Test the trained perceptron
print("Trained Weights:", weights)
print("Number of epochs needed for convergence:", epochs)

# Test the trained perceptron
test_inputs = np.array([[0, 0],
                        [0, 1],
                        [1, 0],
```

```
                        [1, 1]])

print("Predictions:")
for inputs in test_inputs:
    weighted_sum = np.dot(inputs, weights[1:]) + weights[0]
    prediction = step_function(weighted_sum)
    print(f"{inputs} -> {prediction}")
```

In [*]:

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

# Initial weights and learning rate
weights = np.array([10, 0.2, -0.75])
learning_rate = 0.05

# AND gate inputs and targets
and_inputs = np.array([[0, 0],
                       [0, 1],
                       [1, 0],
                       [1, 1]])

and_targets = np.array([0, 0, 0, 1])

#  activation function
def ReLU_function(x):
    if x>0:
        return x
    else:
        return 0

# Training the perceptron and recording errors
epochs = 0
errors_list = []
while True:
    errors = 0
    for inputs, target in zip(and_inputs, and_targets):
        # Compute weighted sum
        weighted_sum = np.dot(inputs, weights[1:]) + weights[0]
        # Apply step activation function
        prediction = ReLU_function(weighted_sum)
        # Compute error
        error = target - prediction
        # Update weights
        weights[1:] += learning_rate * error * inputs
        weights[0] += learning_rate * error
        errors += error**2
    errors_list.append(errors)
    epochs += 1
    if errors == 0:
        break

# Plotting epochs against error values
plt.figure(figsize=(8, 6))
plt.plot(range(1, epochs + 1), errors_list, marker='o', color='b', linestyle='
plt.xlabel('Epochs')
plt.ylabel('Sum-Square Error')
plt.title('Epochs vs Error Plot')
plt.grid(True)
plt.show()

# Test the trained perceptron
print("Trained Weights:", weights)
print("Number of epochs needed for convergence:", epochs)

# Test the trained perceptron
```

```
test_inputs = np.array([[0, 0],
                        [0, 1],
                        [1, 0],
                        [1, 1]])

print("Predictions:")
for inputs in test_inputs:
    weighted_sum = np.dot(inputs, weights[1:]) + weights[0]
    prediction = step_function(weighted_sum)
    print(f"{inputs} -> {prediction}")
```

In [ ]:

```
```