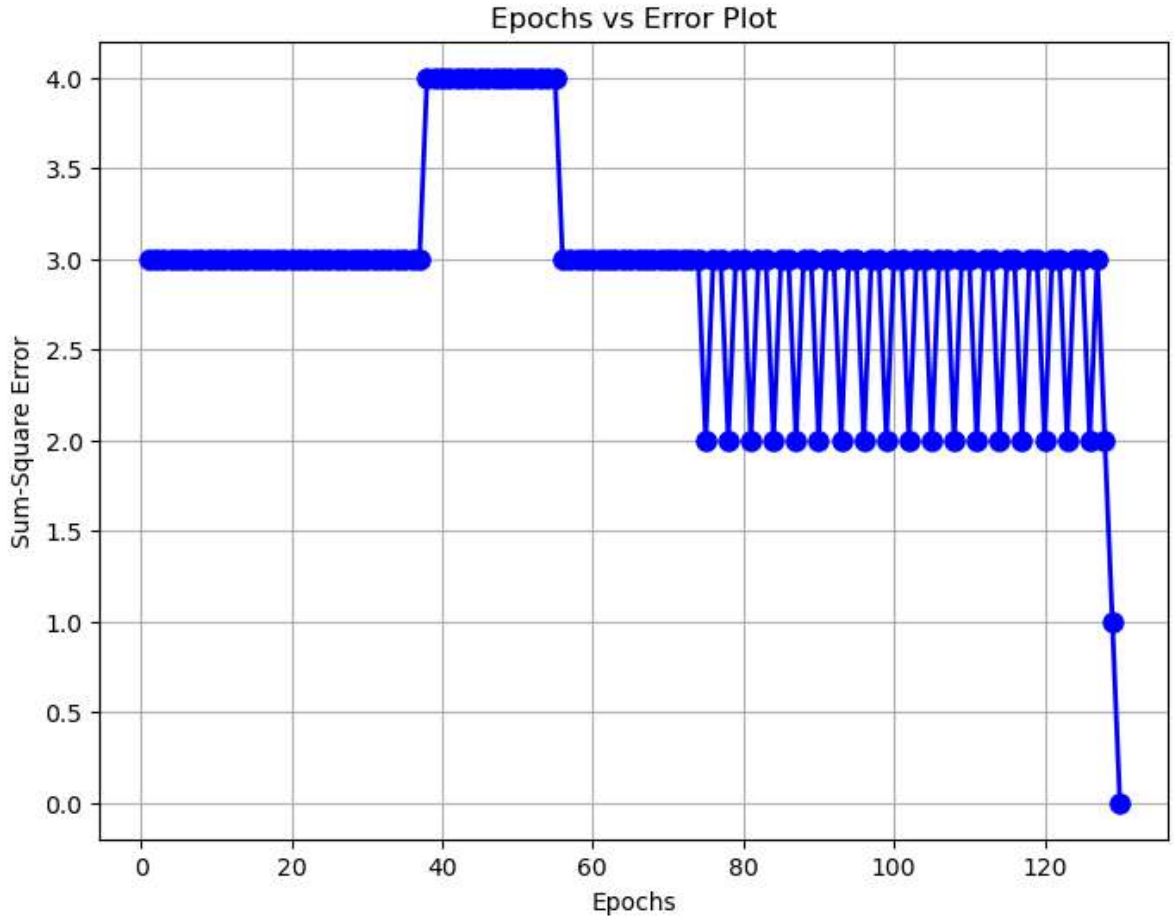
In [9]:

```
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]])
```

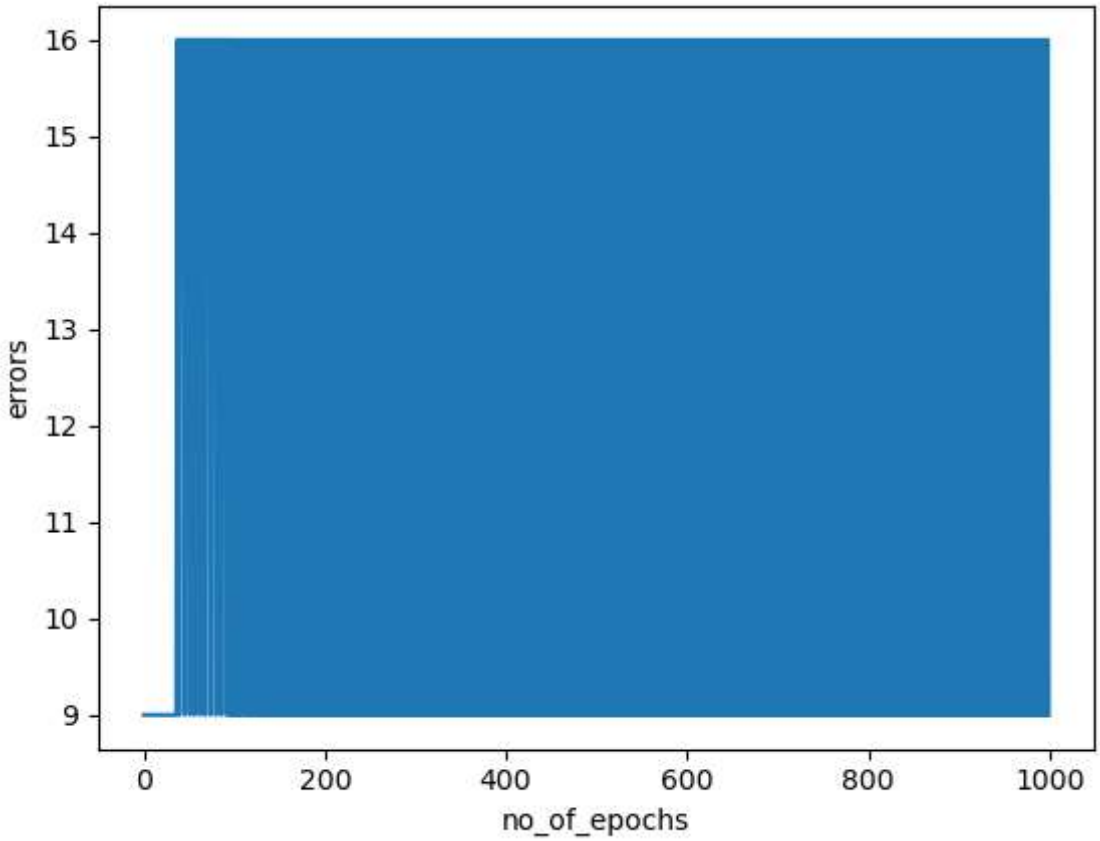
```
print("Predictions:")
for inputs in test_inputs:
    weighted_sum = np.dot(inputs, weights[1:]) + weights[0]
    prediction = step_function(weighted_sum)
    print(f"{inputs} -> {prediction}")
```




```
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
```

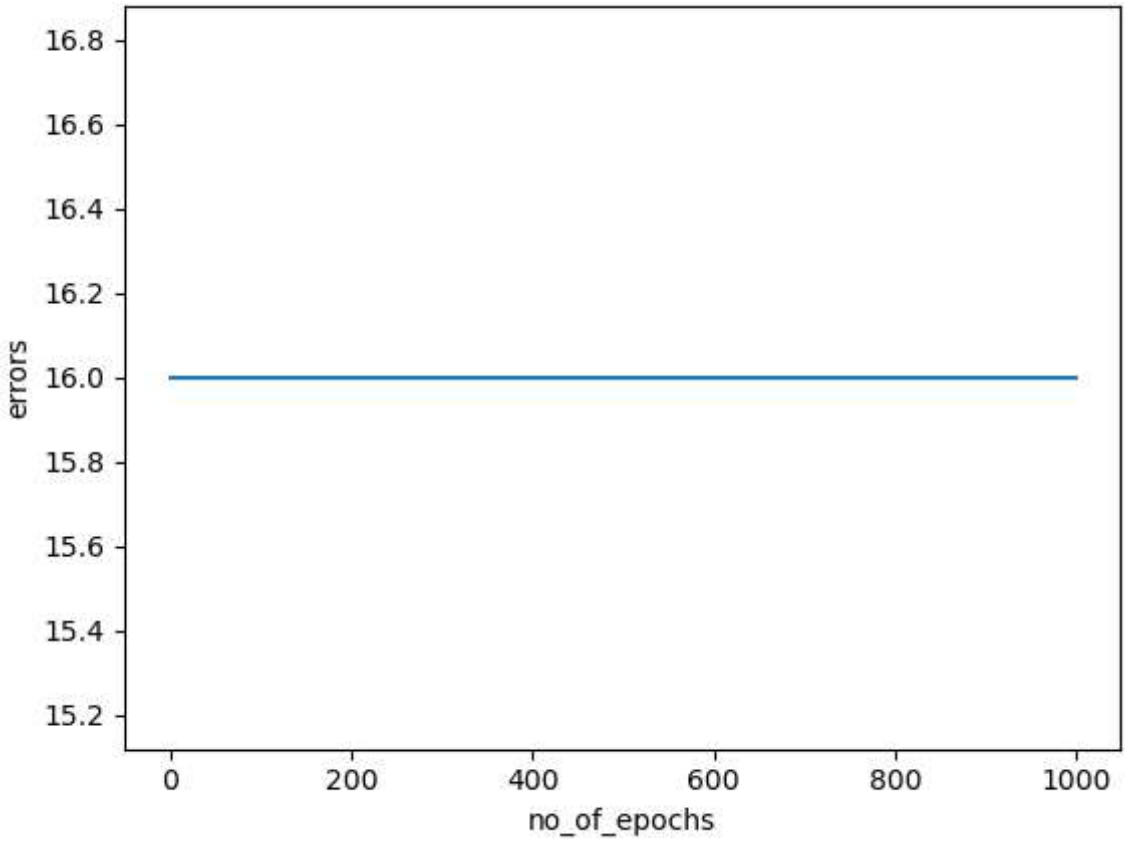
In [10]:
```
#perceptron for AND gate using bipolar step function
import matplotlib.pyplot as plt
import numpy as np
def bipolar(i):
    if i>0:
        return 1
    elif i==0:
        return 0
    else:
        return -1
def AND_function(x,w,learning_rate):
    i=np.dot(w,x)
    s=bipolar(i)
    return s
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([0,0,0,1])
w=np.array([10,0.2,-0.75])
learning_rate=0.05
no_of_epochs=0
errors=[]
while True:
    err_count=0
    for i in range(len(x)):
        X=np.insert(x[i],0,1)
        t=y[i]
        y_in=AND_function(X,w,learning_rate)
        w=w+learning_rate*(t-y_in)*X
        if t!=y_in:
            err_count+=1
    sum_square_error=err_count**2
    errors.append(sum_square_error)
    no_of_epochs+=1
    if(sum_square_error<=0.002 or no_of_epochs>=1000):
        break
print("final weights",w)
print("no.of epochs is ",no_of_epochs)
plt.plot(range(no_of_epochs),errors)
plt.xlabel("no_of_epochs")
plt.ylabel("errors")
plt.show()
```

```
final weights [-0.1   0.1   0.05]
no.of epochs is  1000
```


16
15
14
13
12
11
10
9
errors
0
200
400
600
800
1000
no_of_epochs

In [11]:
```
#perceptron for AND gate using sigmoid function
import matplotlib.pyplot as plt
import math
import numpy as np
def sigmoid(x):
    return 1/(1+math.exp(-x))

def AND_function(x,w,learning_rate):
    i=np.dot(w,x)
    s=sigmoid(i)
    return s
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([0,0,0,1])
w=np.array([10,0.2,-0.75])
learning_rate=0.05
no_of_epochs=0
errors=[]
while True:
    err_count=0
    for i in range(len(x)):
        X=np.insert(x[i],0,1)
        t=y[i]
        y_in=AND_function(X,w,learning_rate)
        w=w+learning_rate*(t-y_in)*X
        if t!=y_in:
            err_count+=1
    sum_square_error=err_count**2
    errors.append(sum_square_error)
    no_of_epochs+=1
    if(sum_square_error<=0.002 or no_of_epochs>=1000):
        break
print("final weights",w)
print("no.of epochs is ",no_of_epochs)
plt.plot(range(no_of_epochs),errors)
plt.xlabel("no_of_epochs")
plt.ylabel("errors")
plt.show()
```

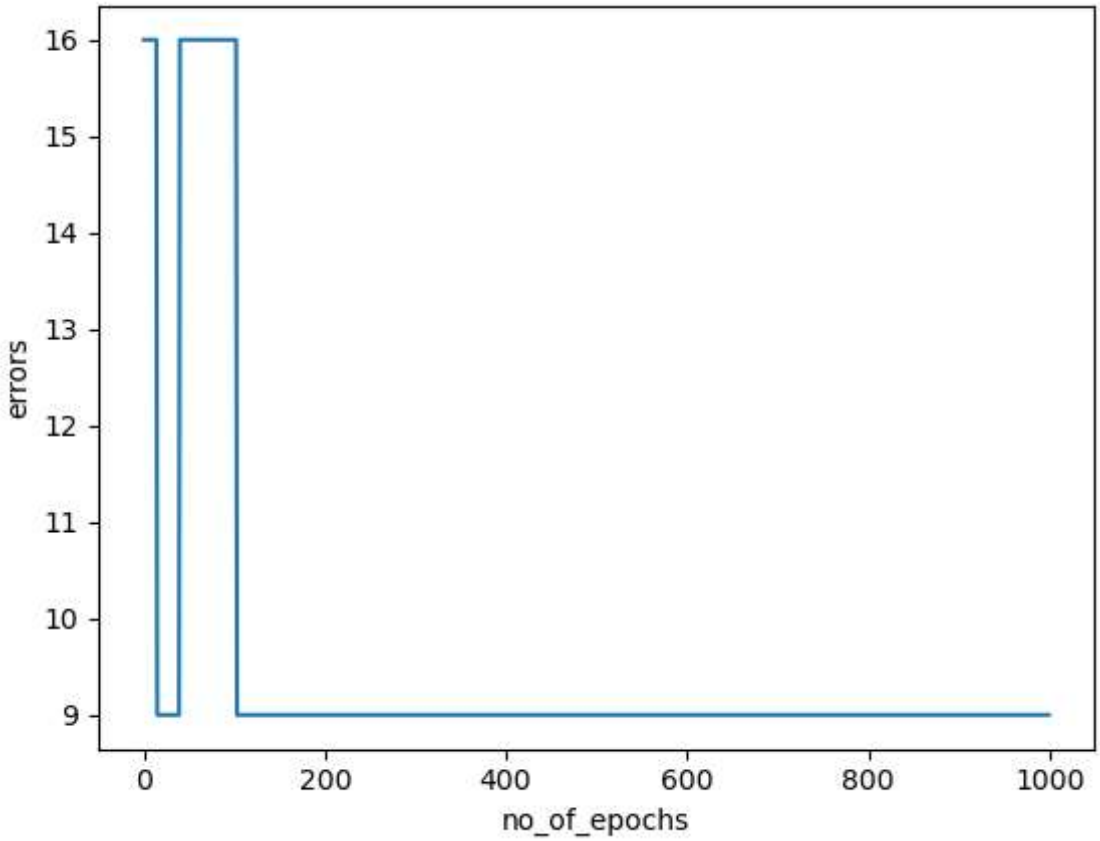
```
final weights [-6.13531308  3.97083321  3.96394835]
no.of epochs is  1000
```


16.8
16.6
16.4
16.2
16.0
15.8
15.6
15.4
15.2
errors
0
200
400
600
800
1000
no_of_epochs

In [12]:

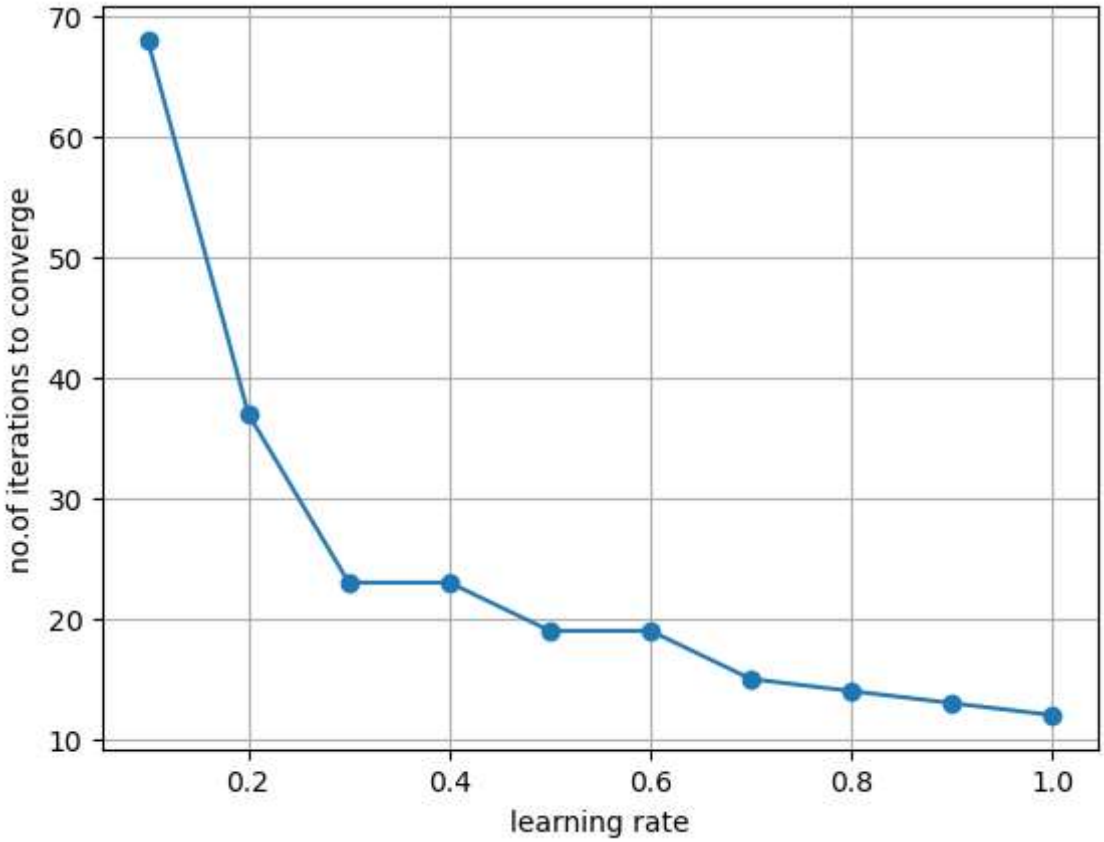
```
#perceptron for AND gate using sigmoid function
import matplotlib.pyplot as plt
import math
import numpy as np
def ReLU(x):
    if x>0:
        return x
    else:
        return 0

def AND_function(x,w,learning_rate):
    i=np.dot(w,x)
    s=ReLU(i)
    return s
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([0,0,0,1])
w=np.array([10,0.2,-0.75])
learning_rate=0.05
no_of_epochs=0
errors=[]
while True:
    err_count=0
    for i in range(len(x)):
        X=np.insert(x[i],0,1)
        t=y[i]
        y_in=AND_function(X,w,learning_rate)
        w=w+learning_rate*(t-y_in)*X
        if t!=y_in:
            err_count+=1
    sum_square_error=err_count**2
    errors.append(sum_square_error)
    no_of_epochs+=1
    if(sum_square_error<=0.002 or no_of_epochs>=1000):
        break
print("final weights",w)
print("no.of epochs is ",no_of_epochs)
plt.plot(range(no_of_epochs),errors)
plt.xlabel("no_of_epochs")
plt.ylabel("errors")
plt.show()
```

```
final weights [-0.99972829  0.99981634  0.99981127]
no.of epochs is  1000
```


16
15
14
13
12
11
10
9
errors
0
200
400
600
800
1000
no_of_epochs

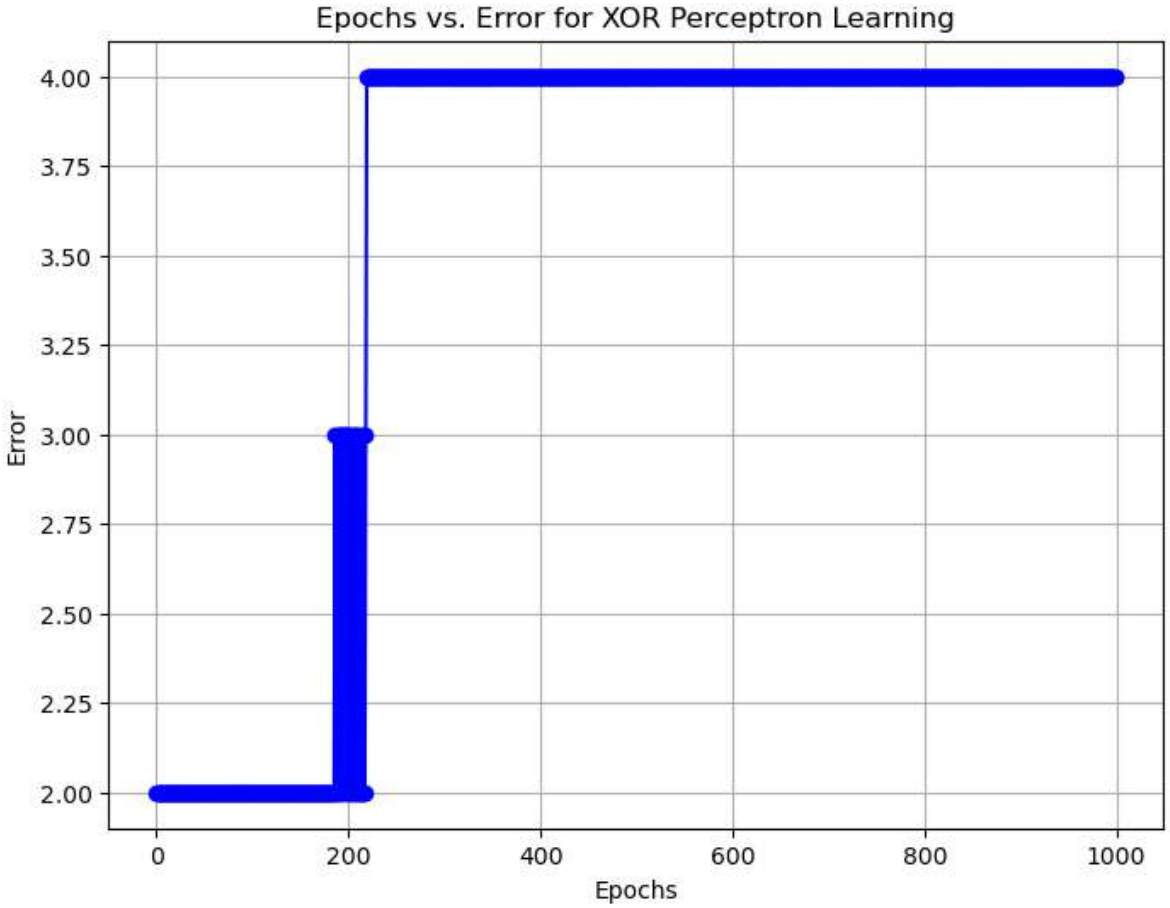
In [2]:
```
#A3 AND GATE
import numpy as np
import matplotlib.pyplot as plt
def step(i):
    if i>=0:
        return 1
    else:
        return 0
def AND_func(x,w,learning_rate):
    i=np.dot(w,x)
    s=step(i)
    return s
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([0,0,0,1])
w=np.array([10,0.2,-0.75])
no_of_epochs=0
learning_rate=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
no_of_iterations=[]
for l in learning_rate:
    W=np.copy(w)
    no_of_epochs=0
    while True:
        error=0
        for i in range(len(x)):
            X=np.insert(x[i],0,1)
            t=y[i]
            y_in=AND_func(X,W,l)
            W=W+l*(t-y_in)*X
            if t!=y_in:
                error=error+1
        sum_square_error=error**2
        no_of_epochs=no_of_epochs+1
        if sum_square_error<=0.002 or no_of_epochs>=1000:
            break
    no_of_iterations.append(no_of_epochs)
for i,l in enumerate(learning_rate):
    print(f"learning rate {l}: no.of iterations to converge={no_of_iterations[
plt.plot(learning_rate,no_of_iterations,marker='o')
plt.xlabel("learning rate")
plt.ylabel("no.of iterations to converge")
plt.grid(True)
plt.show()
```

```
learning rate 0.1: no.of iterations to converge=68
learning rate 0.2: no.of iterations to converge=37
learning rate 0.3: no.of iterations to converge=23
learning rate 0.4: no.of iterations to converge=23
learning rate 0.5: no.of iterations to converge=19
learning rate 0.6: no.of iterations to converge=19
learning rate 0.7: no.of iterations to converge=15
learning rate 0.8: no.of iterations to converge=14
learning rate 0.9: no.of iterations to converge=13
learning rate 1: no.of iterations to converge=12
```


70
60
50
40
30
20
10
no.of iterations to converge
0.2
0.4
0.6
0.8
1.0
learning rate

In [9]:

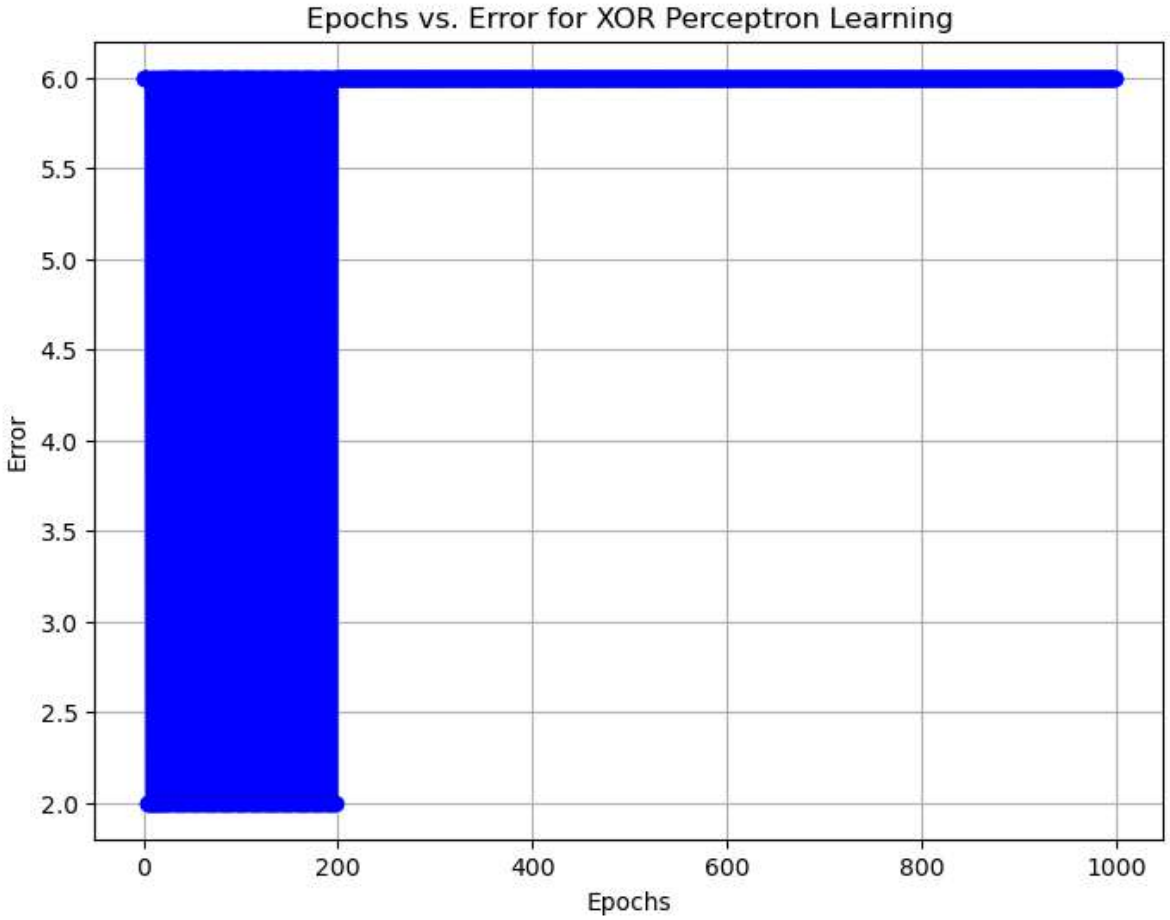
```
import numpy as np
import matplotlib.pyplot as plt

# Initialize weights and bias
w = np.array([10, 0.2, -0.75])
learning_rate = 0.05

# XOR training data
x_train = np.array([[0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1]])  # XOR input
y_train = np.array([0, 1, 1, 0])  # XOR output

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

# Perceptron model
def perceptron(x, weights):
    return step_function(np.dot(x, weights))

# Training the perceptron
epochs = 1000
errors = []

for epoch in range(epochs):
    total_error = 0
    for i in range(len(x_train)):
        prediction = perceptron(x_train[i], w)
        error = y_train[i] - prediction
        total_error += error ** 2
        w += learning_rate * error * x_train[i]  # Weight update rule
    errors.append(total_error)

# Plot epochs against error values
plt.figure(figsize=(8, 6))
plt.plot(range(epochs), errors, marker='o', linestyle='-', color='b')
plt.xlabel('Epochs')
plt.ylabel('Error')
plt.title('Epochs vs. Error for XOR Perceptron Learning')
plt.grid(True)
plt.show()

print("Final weights after training: ", w)
print("Number of epochs ",epochs)
```




```
Final weights after training:  [-7.64666108e-15  1.38777878e-17  9.71445147e-
17]
Number of epochs  1000
```

In [10]:

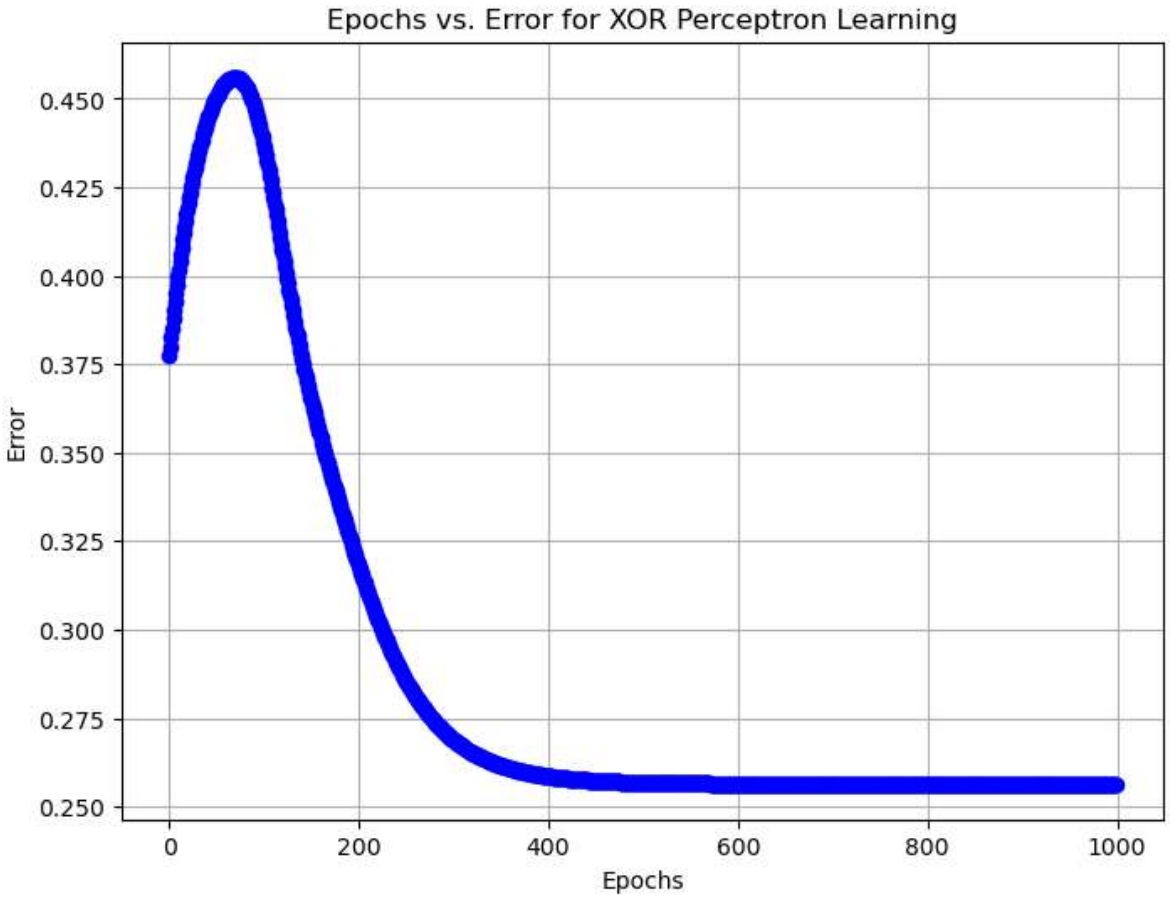
```python
import numpy as np
import matplotlib.pyplot as plt

# Initialize weights and bias
w = np.array([10, 0.2, -0.75])
learning_rate = 0.05

# XOR training data
x_train = np.array([[0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1]])  # XOR input
y_train = np.array([0, 1, 1, 0])  # XOR output

def bipolar_function(i):
    if i>0:
        return 1
    elif i==0:
        return 0
    else:
        return -1

# Perceptron model
def perceptron(x, weights):
    return bipolar_function(np.dot(x, weights))

# Training the perceptron
epochs = 1000
errors = []

for epoch in range(epochs):
    total_error = 0
    for i in range(len(x_train)):
        prediction = perceptron(x_train[i], w)
        error = y_train[i] - prediction
        total_error += error ** 2
        w += learning_rate * error * x_train[i]  # Weight update rule
    errors.append(total_error)

# Plot epochs against error values
plt.figure(figsize=(8, 6))
plt.plot(range(epochs), errors, marker='o', linestyle='-', color='b')
plt.xlabel('Epochs')
plt.ylabel('Error')
plt.title('Epochs vs. Error for XOR Perceptron Learning')
plt.grid(True)
plt.show()

print("Final weights after training: ", w)
print("Number of epochs ",epochs)
```




```
Final weights after training:  [-7.64666108e-15  2.35922393e-16  5.00000000e-
02]
Number of epochs  1000
```

In [17]:

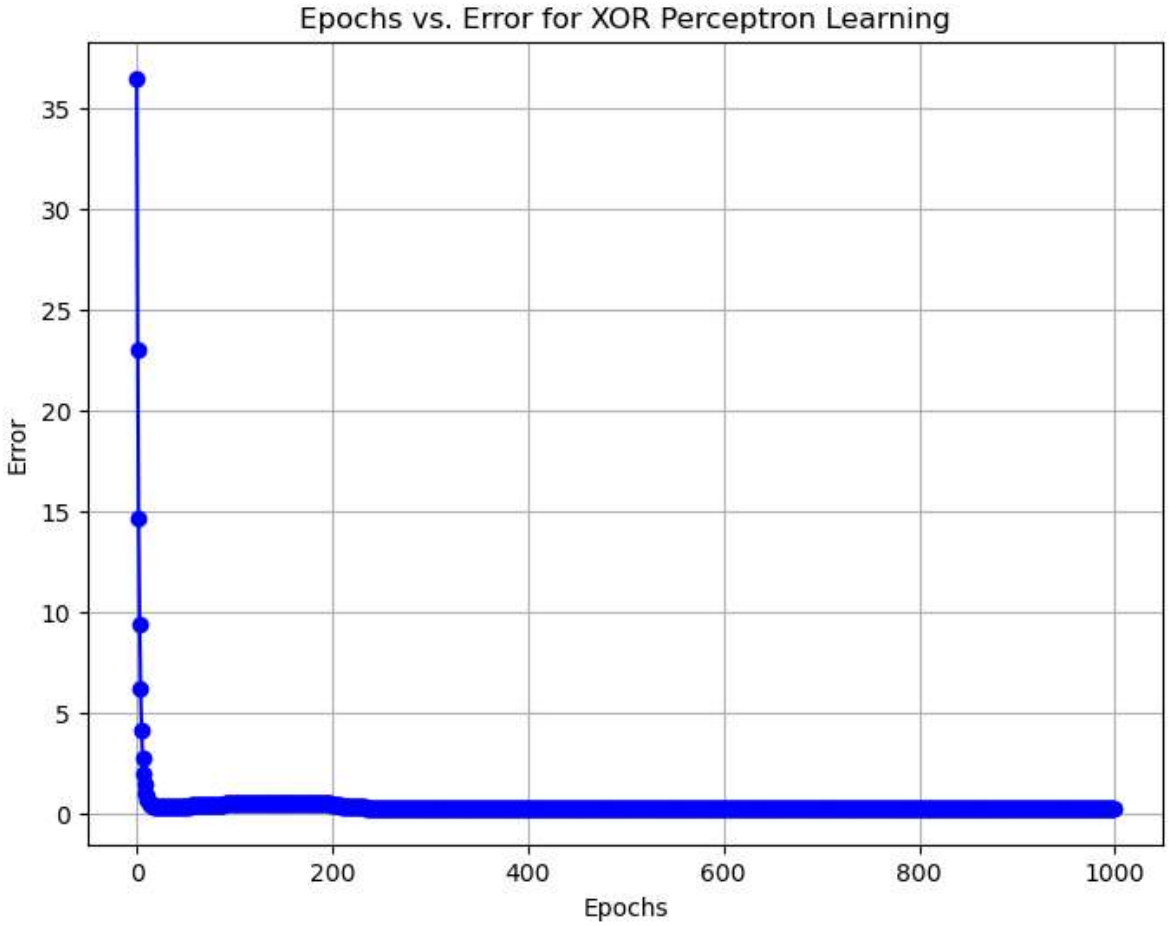
```
import numpy as np
import matplotlib.pyplot as plt
import math

# Initialize weights and bias
w = np.array([10, 0.2, -0.75])
learning_rate = 0.05

# XOR training data
x_train = np.array([[0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1]])  # XOR input
y_train = np.array([0, 1, 1, 0])  # XOR output

def sigmoid_function(x):
    return 1/(1 + np.exp(-x))

# Perceptron model
def perceptron(x, weights):
    return sigmoid_function(np.dot(x, weights))

# Training the perceptron
epochs = 1000
errors = []

for epoch in range(epochs):
    total_error = 0
    for i in range(len(x_train)):
        prediction = perceptron(x_train[i], w)
        error = y_train[i] - prediction
        total_error += error ** 2
        w += learning_rate * error * x_train[i]  # Weight update rule
    mean_error = total_error / len(x_train)
    errors.append(mean_error.item())

# Plot epochs against error values
plt.figure(figsize=(8, 6))
plt.plot(range(epochs), errors, marker='o', linestyle='-', color='b')
plt.xlabel('Epochs')
plt.ylabel('Error')
plt.title('Epochs vs. Error for XOR Perceptron Learning')
plt.grid(True)
plt.show()

print("Final weights after training: ", w)
print("Number of epochs: ", epochs)
```




```
Final weights after training:  [-0.0484434  -0.02320303  0.02272497]
Number of epochs:  1000
```

In [19]:

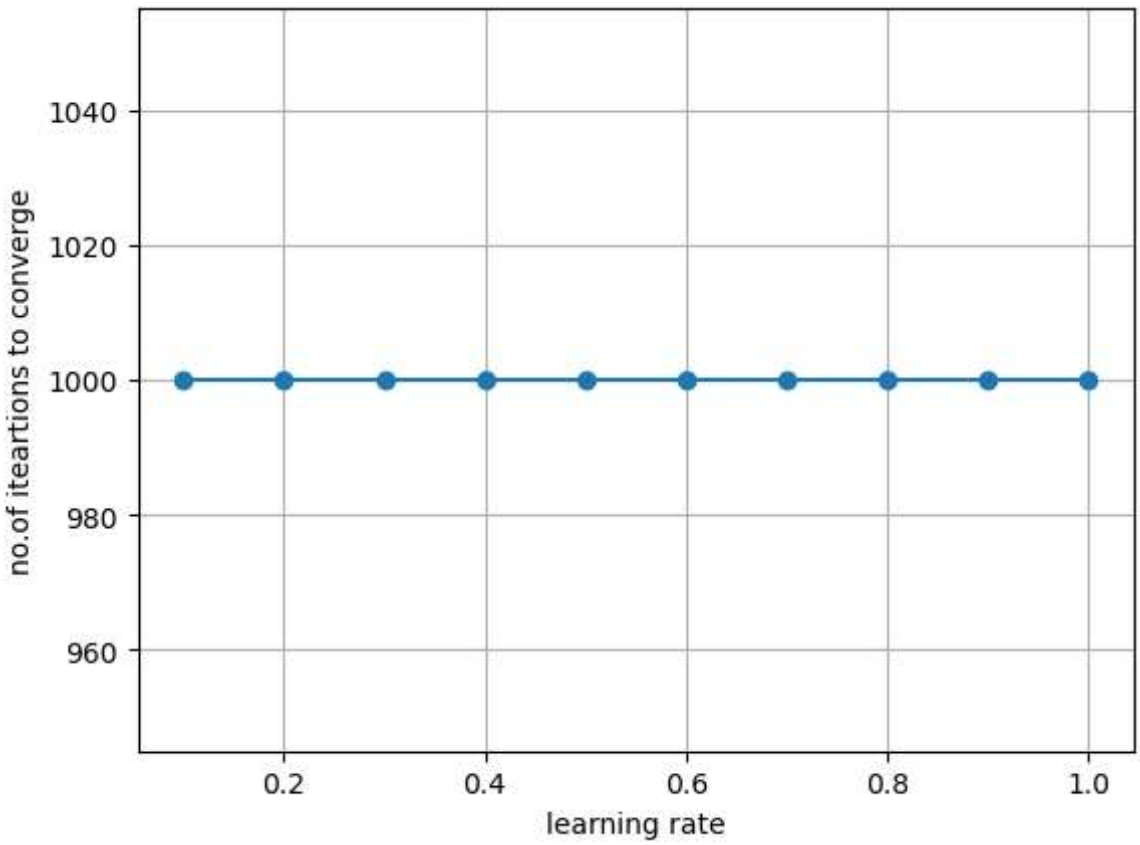
```
import numpy as np
import matplotlib.pyplot as plt
import math

# Initialize weights and bias
w = np.array([10, 0.2, -0.75])
learning_rate = 0.05

# XOR training data
x_train = np.array([[0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1]])  # XOR input
y_train = np.array([0, 1, 1, 0])  # XOR output

def ReLU(x):
    if x>0:
        return x
    else:
        return 0


# Perceptron model
def perceptron(x, weights):
    return ReLU(np.dot(x, weights))

# Training the perceptron
epochs = 1000
errors = []

for epoch in range(epochs):
    total_error = 0
    for i in range(len(x_train)):
        prediction = perceptron(x_train[i], w)
        error = y_train[i] - prediction
        total_error += error ** 2
        w += learning_rate * error * x_train[i]  # Weight update rule
    mean_error = total_error / len(x_train)
    errors.append(mean_error.item())

# Plot epochs against error values
plt.figure(figsize=(8, 6))
plt.plot(range(epochs), errors, marker='o', linestyle='-', color='b')
plt.xlabel('Epochs')
plt.ylabel('Error')
plt.title('Epochs vs. Error for XOR Perceptron Learning')
plt.grid(True)
plt.show()

print("Final weights after training: ", w)
print("Number of epochs: ", epochs)
```




```
Final weights after training:  [-0.05263158 -0.02631579  0.52631579]
Number of epochs:  1000
```

In [1]:
```
#A3(xor function)
import numpy as np
import matplotlib.pyplot as plt
def step(i):
    if i>=0:
        return 1
    else:
        return 0
def XOR_func(x,w,learning_rate):
    i=np.dot(w,x)
    s=step(i)
    return s
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([0,1,1,0])
w=np.array([10,0.2,-0.75])
no_of_epochs=0
learning_rate=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
no_of_iterations=[]
for l in learning_rate:
    W=np.copy(w)
    no_of_epochs=0
    while True:
        error=0
        for i in range(len(x)):
            X=np.insert(x[i],0,1)
            t=y[i]
            y_in=XOR_func(X,W,l)
            W=W+l*(t-y_in)*X
            if t!=y_in:
                error=error+1
        sum_square_error=error**2
        no_of_epochs=no_of_epochs+1
        if sum_square_error<=0.002 or no_of_epochs>=1000:
            break
    no_of_iterations.append(no_of_epochs)
for i,l in enumerate(learning_rate):
    print(f"learning rate {l}: no.of iterations to converge={no_of_iterations[
plt.plot(learning_rate,no_of_iterations,marker='o')
plt.xlabel("learning rate")
plt.ylabel("no.of iteartions to converge")
plt.grid(True)
plt.show()
```

```
learning rate 0.1: no.of iterations to converge=1000
learning rate 0.2: no.of iterations to converge=1000
learning rate 0.3: no.of iterations to converge=1000
learning rate 0.4: no.of iterations to converge=1000
learning rate 0.5: no.of iterations to converge=1000
learning rate 0.6: no.of iterations to converge=1000
learning rate 0.7: no.of iterations to converge=1000
learning rate 0.8: no.of iterations to converge=1000
learning rate 0.9: no.of iterations to converge=1000
learning rate 1: no.of iterations to converge=1000
```


1040
1020
1000
980
960
no.of iteartions to converge
0.2
0.4
0.6
0.8
1.0
learning rate

In [2]:

```python
#A5
#A5
import pandas as pd
import numpy as np
data=[["C_1",20,6,2,386,"Yes"],
      ["C_2",16,3,6,289,"Yes"],
      ["C_3",27,6,2,393,"Yes"],
      ["C_4",19,1,2,110,"No"],
      ["C_5",24,4,2,280,"Yes"],
      ["C_6",22,1,5,167,"No"],
      ["C_7",15,4,2,271,"Yes"],
      ["C_8",18,4,2,274,"Yes"],
      ["C_9",21,1,4,148,"No"],
      ["C_10",16,2,4,198,"No"]]

cols=["Customer","Candies","Mangoes (kg)","Milk Packets (#)","Payment (Rs)","H
data=np.array(data)
x=data[:,1:-1].astype(float)
y=(data[:,-1]=="Yes").astype(int)
x=x/x.max(axis=0)
np.random.seed(0)
w=np.random.rand(x.shape[1])
b=np.random.rand()
def sigmoid(i):
    return 1/(1+np.exp(-i))
learning_rate=0.05
no_of_epochs=1000
for _ in range(no_of_epochs):
    for i in range(len(x)):
        X=x[i]
        t=y[i]
        z=np.dot(X,w)+b
        y_in=sigmoid(z)
        error=t-y_in
        w=w+learning_rate*(t-y_in)*X
        b=b+learning_rate*(t-y_in)
def predict(j):
    z=np.dot(j,w)+b
    y_in=sigmoid(z)
    return round(y_in)
for i in range(len(x)):
    res=predict(x[i])
    actual_output=y[i]
    print(f"Sample {i + 1}: Predicted={res}, Actual={actual_output}")
```

```
Sample 1: Predicted=1, Actual=1
Sample 2: Predicted=1, Actual=1
Sample 3: Predicted=1, Actual=1
Sample 4: Predicted=0, Actual=0
Sample 5: Predicted=1, Actual=1
Sample 6: Predicted=0, Actual=0
Sample 7: Predicted=1, Actual=1
Sample 8: Predicted=1, Actual=1
Sample 9: Predicted=0, Actual=0
Sample 10: Predicted=0, Actual=0
```

In [3]:
```
#A7
import numpy as np
def sigmoid(i):
    return 1/(1+np.exp(-i))
def sigmoid_derivative(i):
    return i*(1-i)
inputlayer=2
outputlayer=1
hiddenlayer=2
np.random.seed(0)
weights_hidden=2*np.random.rand(inputlayer,hiddenlayer)-1
weights_output=2*np.random.rand(hiddenlayer,outputlayer)-1
bias_hidden=np.random.rand(1,hiddenlayer)
bias_output=np.random.rand(1,outputlayer)
learning_rate=0.05
no_of_epochs=1000
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([0,0,0,1])
for epochs in range(no_of_epochs):
    input_layer=x
    input_hidden=np.dot(input_layer,weights_hidden)+bias_hidden
    output_hidden=sigmoid(input_hidden)
    input_output=np.dot(output_hidden,weights_output)+bias_output
    output_output=sigmoid(input_output)
    error=y.reshape(-1,1)-output_output
    if np.mean(np.abs(error))<=0.002:
        print(f"convergence attained after {epochs+1} epochs")
        break
    output=error*sigmoid_derivative(output_output)
    error_hidden=output.dot(weights_output.T)
    hidden=error_hidden*sigmoid_derivative(output_hidden)
    weights_output+=output_hidden.T.dot(output)*learning_rate
    bias_output+=np.sum(output,axis=0,keepdims=True)*learning_rate
    weights_hidden+=input_layer.T.dot(hidden)*learning_rate
    bias_hidden+=np.sum(hidden,axis=0,keepdims=True)*learning_rate
test_data=np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
predicted_output=sigmoid(np.dot(sigmoid(np.dot(test_data, weights_hidden)+bias
print("Predicted Output is")
print(predicted_output)
print("Weights Hidden Layer:")
print(weights_hidden)
print("Weights Output Layer:")
print(weights_output)
```

```
Predicted Output is
[[0.21309172]
 [0.26559309]
 [0.26868945]
 [0.3239306 ]]
Weights Hidden Layer:
[[-0.87389445  0.11483407]
 [-0.77522389 -0.19685031]]
Weights Output Layer:
[[-1.4620038 ]
 [-0.31175157]]
```

In [4]:
```
#A8#A8 neural network for XOR gate
import numpy as np
def sigmoid(i):
    return 1/(1+np.exp(-i))
def sigmoid_derivative(i):
    return i*(1-i)
inputlayer=2
outputlayer=1
hiddenlayer=2
np.random.seed(0)
weights_hidden=2*np.random.rand(inputlayer,hiddenlayer)-1
weights_output=2*np.random.rand(hiddenlayer,outputlayer)-1
bias_hidden=np.random.rand(1,hiddenlayer)
bias_output=np.random.rand(1,outputlayer)
learning_rate=0.05
no_of_epochs=1000
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([0,1,1,0])
for epochs in range(no_of_epochs):
    input_layer=x
    input_hidden=np.dot(input_layer,weights_hidden)+bias_hidden
    output_hidden=sigmoid(input_hidden)
    input_output=np.dot(output_hidden,weights_output)+bias_output
    output_output=sigmoid(input_output)
    error=y.reshape(-1,1)-output_output
    if np.mean(np.abs(error))<=0.002:
        print(f"convergence attained after {epochs+1} epochs")
        break
    output=error*sigmoid_derivative(output_output)
    error_hidden=output.dot(weights_output.T)
    hidden=error_hidden*sigmoid_derivative(output_hidden)
    weights_output+=output_hidden.T.dot(output)*learning_rate
    bias_output+=np.sum(output,axis=0,keepdims=True)*learning_rate
    weights_hidden+=input_layer.T.dot(hidden)*learning_rate
    bias_hidden+=np.sum(hidden,axis=0,keepdims=True)*learning_rate
test_data=np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
predicted_output=sigmoid(np.dot(sigmoid(np.dot(test_data, weights_hidden)+bias
print("Predicted Output is")
print(predicted_output)
```

```
Predicted Output is
[[0.50528112]
 [0.49927189]
 [0.50121227]
 [0.49537714]]
```

In [5]:

```
#A9
import numpy as np
def sigmoid(i):
    return 1/(1+np.exp(-i))
def sigmoid_derivative(i):
    return i*(1-i)
inputlayer=2
outputlayer=2
hiddenlayer=2
np.random.seed(0)
weights_inputhidden=2*np.random.rand(inputlayer,hiddenlayer)-1
weights_hiddenoutput=2*np.random.rand(hiddenlayer,outputlayer)-1
bias_hidden=np.random.rand(1,hiddenlayer)
bias_output=np.random.rand(1,outputlayer)
learning_rate=0.05
no_of_epochs=1000
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([[1,0],[1,0],[1,0],[0,1]])
for epochs in range(no_of_epochs):
    input_hidden=np.dot(x,weights_inputhidden)+bias_hidden
    output_hidden=sigmoid(input_hidden)
    input_output=np.dot(output_hidden,weights_hiddenoutput)+bias_output
    output_output=sigmoid(input_output)
    error=y-output_output
    if np.mean(np.abs(error))<=0.002:
        print(f"convergence attained after {epochs+1} epochs")
        break
    output=error*sigmoid_derivative(output_output)
    hidden=output*sigmoid_derivative(output_hidden)
    weights_hiddenoutput+=output_hidden.T.dot(output)*learning_rate
    bias_output+=np.sum(output,axis=0,keepdims=True)*learning_rate
    weights_inputhidden+=x.T.dot(hidden)*learning_rate
    bias_hidden+=np.sum(hidden,axis=0,keepdims=True)*learning_rate
print("predicted outputs are ")
print(output_output)
```

```
predicted outputs are
[[0.79381813 0.19044883]
 [0.72964327 0.27463307]
 [0.70894676 0.29802031]
 [0.63972411 0.39737433]]
```

In [6]:
```python
#A10
from sklearn.neural_network import MLPClassifier
import numpy as np

# Define the input data (X) and corresponding labels (y) for AND gate
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 0, 0, 1])

# Create an MLPClassifier with two hidden layers, each containing 4 neurons
mlp = MLPClassifier(hidden_layer_sizes=(4, 4), max_iter=2000, random_state=42)

# Train the model
mlp.fit(X, y)

# Test the model with some examples
test_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
predictions = mlp.predict(test_data)

# Print the predictions
for i in range(len(test_data)):
    print(f"Input: {test_data[i]}, Predicted Output: {predictions[i]}")
```

```
Input: [0 0], Predicted Output: 0
Input: [0 1], Predicted Output: 0
Input: [1 0], Predicted Output: 0
Input: [1 1], Predicted Output: 1
```

In [7]:
```python
#A10
from sklearn.neural_network import MLPClassifier
import numpy as np

# Define the input data (X) and corresponding labels (y) for XOR gate
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])  # XOR outputs

# Create an MLPClassifier with one hidden layer containing 4 neurons
mlp = MLPClassifier(hidden_layer_sizes=(4,4), max_iter=2000, random_state=42)

# Train the model
mlp.fit(X, y)

# Test the model with some examples
test_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
predictions = mlp.predict(test_data)

# Print the predictions
for i in range(len(test_data)):
    print(f"Input: {test_data[i]}, Predicted Output: {predictions[i]}")
```

```
Input: [0 0], Predicted Output: 0
Input: [0 1], Predicted Output: 1
Input: [1 0], Predicted Output: 1
Input: [1 1], Predicted Output: 0
```

In [4]:

```
#A11
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score,classification_report

data = pd.read_csv('train_data.csv')

data['headline'].fillna('',inplace=True)
data['written_by'].fillna('',inplace=True)
data['news'].fillna('',inplace=True)
x=data.drop(columns=['label'])
y=data['label']
label_encoder=LabelEncoder()
categorical_cols=['headline','written_by','news']
for col in categorical_cols:
    x[col]=label_encoder.fit_transform(x[col])
tfidf_vectorizer=TfidfVectorizer(max_features=2000)
x_text=tfidf_vectorizer.fit_transform(data['headline']+' '+data['written_by']+
x=pd.concat([x.drop(columns=['headline','written_by','news']),pd.DataFrame(x_t
x.columns=x.columns.astype(str)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=
mlp_classifier=MLPClassifier(hidden_layer_sizes=(100,),activation='relu',max_i
mlp_classifier.fit(x_train,y_train)
y_in=mlp_classifier.predict(x_test)
accuracy=accuracy_score(y_test,y_in)
report=classification_report(y_test,y_in)
print(f"accuracy:{accuracy}")
print("classification report")
print(report)
```

```
---------------------------------------------------------------------------
ParserError                               Traceback (most recent call last)
Cell In[4], line 13
     10 from sklearn.feature_extraction.text import TfidfVectorizer
     11 from sklearn.metrics import accuracy_score,classification_report
---> 13 data = pd.read_csv('train_data.csv')
     15 data['headline'].fillna('',inplace=True)
     16 data['written_by'].fillna('',inplace=True)

File ~\anaconda3\Lib\site-packages\pandas\util\_decorators.py:211, in deprecate_kwarg.<locals>._deprecate_kwarg.<locals>.wrapper(*args, **kwargs)
    209     else:
    210         kwargs[new_arg_name] = new_arg_value
--> 211 return func(*args, **kwargs)

File ~\anaconda3\Lib\site-packages\pandas\util\_decorators.py:331, in deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:950, in read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, encoding_errors, dialect, error_bad_lines, warn_bad_lines, on_bad_lines, delim_whitespace, low_memory, memory_map, float_precision, storage_options)
    935 kwds_defaults = _refine_defaults_read(
    936     dialect,
    937     delimiter,
   (...)
    946     defaults={"delimiter": ","},
    947 )
    948 kwds.update(kwds_defaults)
--> 950 return _read(filepath_or_buffer, kwds)

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:611, in _read(filepath_or_buffer, kwds)
    608     return parser
    610 with parser:
--> 611     return parser.read(nrows)

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:1778, in TextFileReader.read(self, nrows)
   1771 nrows = validate_integer("nrows", nrows)
   1772 try:
   1773     # error: "ParserBase" has no attribute "read"
   1774     (
   1775         index,
```

```
   1776         columns,
   1777         col_dict,
-> 1778     ) = self._engine.read(  # type: ignore[attr-defined]
   1779         nrows
   1780     )
   1781 except Exception:
   1782     self.close()

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\c_parser_wrapper.py:230,
in CParserWrapper.read(self, nrows)
    228 try:
    229     if self.low_memory:
--> 230         chunks = self._reader.read_low_memory(nrows)
    231         # destructive to chunks
    232         data = _concatenate_chunks(chunks)

File ~\anaconda3\Lib\site-packages\pandas\_libs\parsers.pyx:808, in pandas._l
ibs.parsers.TextReader.read_low_memory()

File ~\anaconda3\Lib\site-packages\pandas\_libs\parsers.pyx:866, in pandas._l
ibs.parsers.TextReader._read_rows()

File ~\anaconda3\Lib\site-packages\pandas\_libs\parsers.pyx:852, in pandas._l
ibs.parsers.TextReader._tokenize_rows()

File ~\anaconda3\Lib\site-packages\pandas\_libs\parsers.pyx:1973, in pandas._
libs.parsers.raise_parser_error()

ParserError: Error tokenizing data. C error: Calling read(nbytes) on source f
ailed. Try engine='python'.
```

In [5]:

```
#A10
#MLP classifier for AND gate using sigmoid function
import numpy as np
from sklearn.neural_network import MLPClassifier
x=[[0,0],[0,1],[1,0],[1,1]]
y=[0,0,0,1]
weights=np.array([[0.2,-0.75]])
bias=np.array([10.0,10.0])
mlp_and=MLPClassifier(hidden_layer_sizes=(2,),activation='logistic',max_iter=1
mlp_and.coefs_=[weights.T,np.array([[1],[1]])]
mlp_and.intercepts_=[bias,np.array([-10.0])]
mlp_and.fit(x,y)
predicted_output=mlp_and.predict(x)
print("predicted outputs are ",predicted_output)
```

```
predicted outputs are  [0 0 0 1]
```

In [6]:
```
#MLP classifier for AND gate using ReLU function
from sklearn.neural_network import MLPClassifier
x=[[0,0],[0,1],[1,0],[1,1]]
y=[0,0,0,1]
weights=np.array([[0.2,-0.75]])
bias=np.array([10.0,10.0])
mlp_and=MLPClassifier(hidden_layer_sizes=(2,),activation='relu',max_iter=1000,
mlp_and.coefs_=[weights.T,np.array([[1],[1]])]
mlp_and.intercepts_=[bias,np.array([-10.0])]
mlp_and.fit(x,y)
predicted_output=mlp_and.predict(x)
print("predicted outputs are ",predicted_output)
```

```
predicted outputs are  [0 0 0 0]
```

In [7]:
```
#MLP classifier for XOR gate using ReLU function
from sklearn.neural_network import MLPClassifier
x=[[0,0],[0,1],[1,0],[1,1]]
y=[0,1,1,0]
weights=np.array([[0.2,-0.75]])
bias=np.array([10.0,10.0])
mlp_xor=MLPClassifier(hidden_layer_sizes=(2,),activation='relu',max_iter=1000,
mlp_xor.coefs_=[weights.T,np.array([[1],[1]])]
mlp_xor.intercepts_=[bias,np.array([-10.0])]
mlp_xor.fit(x,y)
predicted_output=mlp_xor.predict(x)
print("predicted outputs are ",predicted_output)
```

```
predicted outputs are  [1 0 0 0]
```

In [8]:
```
#MLP classifier for XOR gate using sigmoid function
from sklearn.neural_network import MLPClassifier
x=[[0,0],[0,1],[1,0],[1,1]]
y=[0,1,1,0]
weights=np.array([[0.2,-0.75]])
bias=np.array([10.0,10.0])
mlp_xor=MLPClassifier(hidden_layer_sizes=(2,),activation='logistic',max_iter=1
mlp_xor.coefs_=[weights.T,np.array([[1],[1]])]
mlp_xor.intercepts_=[bias,np.array([-10.0])]
mlp_xor.fit(x,y)
predicted_output=mlp_xor.predict(x)
print("predicted outputs are ",predicted_output)
```

```
predicted outputs are  [0 0 1 1]
```

In [ ]: