```
#Build a three-layer Artificial Neural Network by implementing the Backpropagation algorithm
import numpy as np
def sigmoid(x):
  return 1/(1+np.exp(-x))
def sigmoid_derivative(x):
  return x*(1-x)
input_size=2
hidden_size=3
output_size=1
learning_rate=0.1
weights_input_hidden=np.random.rand(input_size,hidden_size)
biases_hidden=np.zeros((1,hidden_size))
weights_hidden_output=np.random.rand(hidden_size,output_size)
biases_output=np.zeros((1,output_size))
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([[0],[1],[1],[0]])
epochs=5000
for epoch in range(epochs):
  hidden_layer_input=np.dot(x,weights_input_hidden)+biases_hidden
  hidden_layer_output=sigmoid(hidden_layer_input)
  output_layer_input=np.dot(hidden_layer_output,weights_hidden_output)+biases_output
  predicted_output=sigmoid(output_layer_input)
  loss=0.5*np.mean((y-predicted_output)**2)
  output_error=y-predicted_output
  output_delta=output_error*sigmoid_derivative(predicted_output)
  hidden_layer_error=output_delta.dot(weights_hidden_output.T)
  hidden_layer_delta=hidden_layer_error*sigmoid_derivative(hidden_layer_output)
  weights_hidden_output+=hidden_layer_output.T.dot(output_delta)*learning_rate
  biases_output+=np.sum(output_delta,axis=0,keepdims=True)*learning_rate
  weights_input_hidden+=x.T.dot(hidden_layer_delta)*learning_rate
  biases_hidden+=np.sum(hidden_layer_delta,axis=0,keepdims=True)*learning_rate
  if epoch%1000==0:
    print(f"Epoch{epoch},Loss:{loss}")
test_input=np.array([[0,0],[0,1],[1,0],[1,1]])
hidden_layer_input_test=np.dot(test_input,weights_input_hidden)+biases_hidden
hidden_layer_output_test=sigmoid(hidden_layer_input_test)
output_layer_input_test=np.dot(hidden_layer_output_test,weights_hidden_output)+biases_output
predicted_output_test=sigmoid(output_layer_input_test)
print("\nPredicted Output after Training")
print(predicted_output_test)
```

```
Epoch0,Loss:0.1506096062967821
Epoch1000,Loss:0.12084062789686631
Epoch2000,Loss:0.09730947676475625
Epoch3000,Loss:0.06011101979327276
Epoch4000,Loss:0.01939037392237135

Predicted Output after Training
[[0.11993103]
 [0.88700678]
 [0.88305949]
 [0.13080452]]
```

```
import numpy as np
import pandas as pd
df=pd.read_csv("IRIS.csv")
def sigmoid(x):
  return 1/(1+np.exp(-x))
def sigmoid_derivative(x):
  return x*(1-x)
input_size=5
hidden_size=3
output_size=1
learning_rate=0.1
weights_input_hidden=np.random.rand(input_size,hidden_size)
biases_hidden=np.zeros((1,hidden_size))
weights_hidden_output=np.random.rand(hidden_size,output_size)
biases_output=np.zeros((1,output_size))
x=df.iloc[:, :5].values
y=df.iloc[:, :5].values
print(f"Shape of x: {x.shape}")
print(f"Shape of weights_input_hidden: {weights_input_hidden.shape}")
print(f"Data type of x: {x.dtype}")
print(f"Data type of weights_input_hidden: {weights_input_hidden.dtype}")
df = df.drop('species', axis=1)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['species'] = le.fit_transform(df['species'])
epochs=10000
for epoch in range(epochs):
  hidden_layer_input=np.dot(x,weights_input_hidden)+biases_hidden
  hidden_layer_output=sigmoid(hidden_layer_input)
  output_layer_input=np.dot(hidden_layer_output,weights_hidden_output)+biases_output
  predicted_output=sigmoid(output_layer_input)
  loss=0.5*np.mean((y-predicted_output)**2)
  output_error=y-predicted_output
  output_delta=output_error*sigmoid_derivative(predicted_output)
  hidden_layer_error=output_delta.dot(weights_hidden_output.T)
  hidden_layer_delta=hidden_layer_error*sigmoid_derivative(hidden_layer_output)
  weights_hidden_output+=hidden_layer_output.T.dot(output_delta)*learning_rate
  biases_output+=np.sum(output_delta,axis=0,keepdims=True)*learning_rate
  weights_input_hidden+=x.T.dot(hidden_layer_delta)*learning_rate
  biases_hidden+=np.sum(hidden_layer_delta,axis=0,keepdims=True)*learning_rate
  if epoch%1000==0:
    print(f"Epoch{epoch},Loss:{loss}")
test_input=np.array([[0,0],[0,1],[1,0],[1,1]])
hidden_layer_input_test=np.dot(test_input,weights_input_hidden)+biases_hidden
hidden_layer_output_test=sigmoid(hidden_layer_input_test)
output_layer_input_test=np.dot(hidden_layer_output_test,weights_hidden_output)+biases_output
predicted_output_test=sigmoid(output_layer_input_test)
print("\nPredicted Output after Training")
print(predicted_output_test)
```