5) Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Data Initialization

- `X` is a NumPy array containing the input data, with each row having two features (two columns).
- `y` is a NumPy array representing the target output values, where each row contains a single output value. This represents a regression problem.
- To aid training convergence, both the input and output values are normalized to a range between 0 and 1.

Sigmoid Function

- The `sigmoid(x)` function is a mathematical function commonly used in neural networks. It introduces non-linearity to the model and maps any input to a value between 0 and 1.

Derivative of Sigmoid Function

- The `derivatives_sigmoid(x)` function calculates the derivative of the sigmoid function. It plays a crucial role in the backpropagation process, which is used to update the network's weights.

Hyperparameters and Network Architecture

- `epoch` represents the number of training iterations (epochs).
- `lr` is the learning rate, determining how much the network's weights are updated during each iteration.
- `inputlayer_neurons` is the number of input features, which is 2 in this case.
- `hiddenlayer_neurons` is the number of neurons in the hidden layer, with 3 neurons in this example.
- `output_neurons` represents the number of neurons in the output layer, with 1 neuron here.

Weight and Bias Initialization

- Weight matrices `wh` and `wout` are initialized with random values. These weights are essential for the network to learn the mapping from input to output.
- Bias vectors `bh` and `bout` are also initialized with random values.

Training Loop

- The program executes a training loop for a specified number of epochs.
- In each epoch, it conducts forward propagation to compute the predicted output based on the current weights and biases.
- The error, which is the difference between the actual output `y` and the predicted output, is then calculated. This error is used to adjust the network's weights and biases.

Forward Propagation

- `hinp1` computes the weighted sum of inputs to the hidden layer.
- `hinp` adds the bias terms to the weighted sum.
- `hlayer_act` represents the output of the hidden layer after applying the sigmoid activation function.

- `outinp1` and `outinp` follow similar calculations for the output layer, ultimately yielding the final predicted output.

Backpropagation

- The error (`EO`) is determined by finding the difference between the actual output and the predicted output.
- Gradients are computed using the derivative of the sigmoid function. These gradients help assess how much the network's weights contributed to the error.
- The weights are then updated using these gradients to minimize the error.

Printing the Results

- The program prints the original input, actual output (`y`), and the predicted output. The predicted output is the network's estimation of the output values.

In this example, the Artificial Neural Network is trained to predict output values based on input features. After training, it can make predictions for new input data. The predicted output values are close to the actual output values, indicating that the network has learned the relationship between the inputs and outputs.

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X / np.amax(X, axis=0)  # maximum of X array longitudinally
y = y / 100


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


# Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)


# Variable initialization
epoch = 5000  # Setting training iterations
lr = 0.1  # Setting learning rate
inputlayer_neurons = 2  # number of features in data set
hiddenlayer_neurons = 3  # number of hidden layers neurons
output_neurons = 1  # number of neurons at output layer

# weight and bias initialization
wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))
bh = np.random.uniform(size=(1, hiddenlayer_neurons))
wout = np.random.uniform(size=(hiddenlayer_neurons, output_neurons))
bout = np.random.uniform(size=(1, output_neurons))
```

```
# draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
    # Forward Propogation
    hinp1 = np.dot(X, wh)
    hinp = hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1 = np.dot(hlayer_act, wout)
    outinp = outinp1 + bout
    output = sigmoid(outinp)

    # Backpropagation
    EO = y - output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    # how much hidden layer wts contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad

    # dotproduct of nextlayererror and currentlayerop
    wout += hlayer_act.T.dot(d_output) * lr
    wh += X.T.dot(d_hiddenlayer) * lr

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n", output)
```

Output:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89813172]
 [0.87315548]
 [0.89797009]]
```