AND Logic Gate

1. Description:

Implement Perceptron Neural Network for **AND** Logical Function.
The Step function is the activation function using:

- threshold = -0.2
- learning rate = 0.1
- number of iterations =100

For the following data:

Training set input= [[0, 0],
[1, 1],
[1, 0],
[0, 1]]
Output = [0,1, 0, 0]

2. Introduction:

Neural network algorithm:

It is algorithm that is designed to make the machine learn and visualize like the human's brain, It starts by making the machine train on some data with inputs and outputs and then make it think in another data inputs and predict the output by thinking in what it learns.

It's applications:

- ❖ Data visualization and analysis.
- ❖ Data Compression.
- ❖ Face or voice Recognition and verification.

2. The algorithm:

Neural network

1. The main steps of the algorithm

- Step 1: Initialize random weights.
- Step 2: Determine pattern and target output.
- Step 3: Calculate expected output by applying the step of the activation function: expected output = Step $(\sum_{i=1}^{n} wi\ xi\ )$
- Step 4: Calculate error and update weights:
  Error = Expected output – Real output
  New weight i = Old weight i + (Error * Learning rate * Xi)
- Step 5: Repeat from Step 2 until accepted error.

2. The implementation of the algorithm:

```
class NeuralNetwork():

    def __init__(self, learning_rate, threshold):
        self.learning_rate = learning_rate
        self.threshold = threshold
        np.random.seed(1)
        self.synaptic_weights = 2 * np.random.random((2, 1)) - 1

    def step(self, x):
        if x > float(self.threshold):
            return 1
        else:
            return 0
        pass

    def train(self, training_inputs, training_outputs, training_iterations):
        for iteration in range(training_iterations):
            output = self.think(training_inputs)
            error = training_outputs - output
            self.synaptic_weights += np.dot(training_inputs.T, error * self.learning_rate)
        pass

    def think(self, inputs):
        inputs = inputs.astype(float)
        output = self.step(np.sum(np.dot(inputs, self.synaptic_weights)))
        return output
        pass
```

3. Sample run (the output)

```
Beginning Randomly Generated Weights:
[[-0.16595599]
 [ 0.44064899]]
Ending Weights After Training:
[[-0.36595599]
 [ 0.24064899]]
Considering New Situation:  1 1 New Output data:  1
Wow, we did it!
```

3. Discussion:

Neural network algorithm:

- In NeuralNetwork Class there are 4 main functions:
    1. The initialize function which take two parameters the learning rate ant the threshold and initialize them in the class's variables and initialize number of random weights equal to the inputs number.
    2. Step Function which compare the output of the summation with the threshold if greater than return 1 else return 0.
    3. The think function which takes the inputs as a parameter and returns the expected output by applying the step of the activation function.
    4. The train function which takes three parameters the training inputs that the machine will train on them and their outputs which are the training outputs and the number of iterations used to get an accepted error, then loop by the number of iteration and in every iteration call the think function to get the expected output and then calculates the error and update the weights.