Name : محمد اسماعيل السعيد الدسوقي العساس

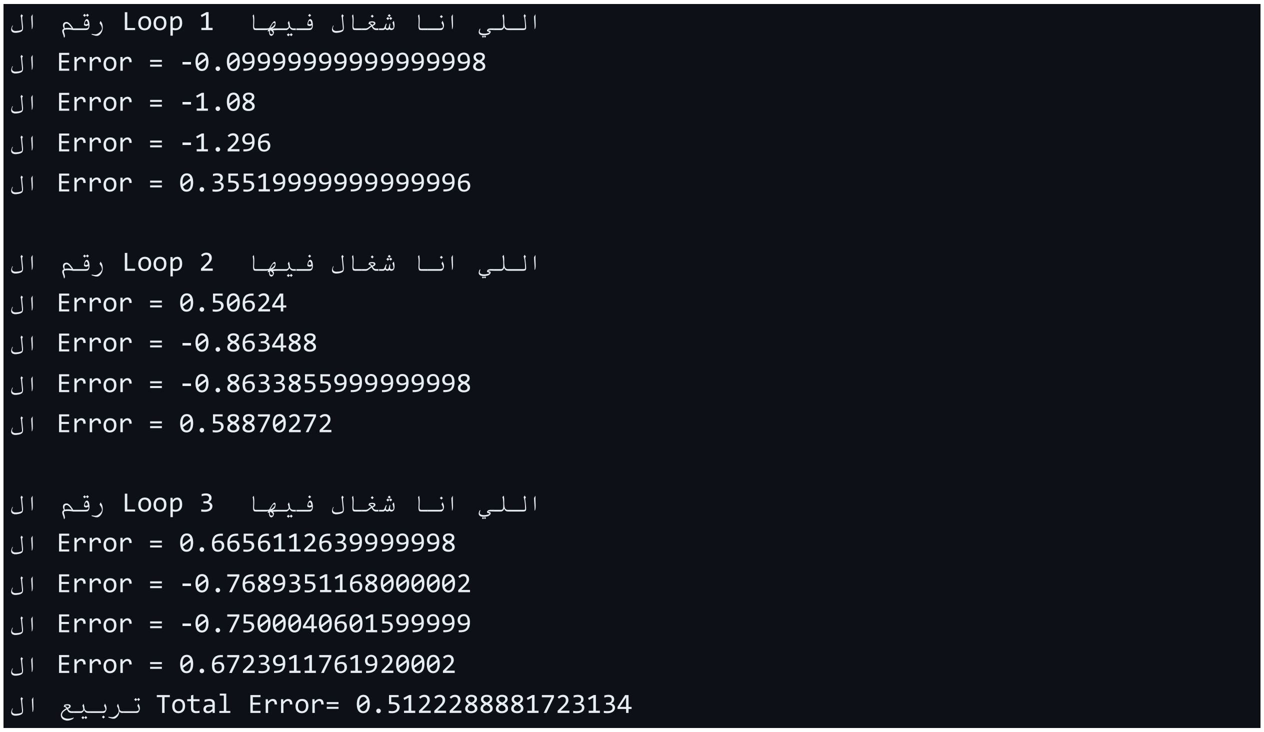
ID : 800167561

==

كود ال Adaline ;)

```
import numpy as np

import sys

sys.stdout.reconfigure(encoding='utf-8')


inputs = np.array([[-1, -1], [-1, 1], [1, -1], [1, 1]])

outputs = np.array([-1, -1, -1, 1])


weight = np.array([0.5, 0.5])

bias = 0.1

learning_rate = 0.2

epoch = 3


for i in range(epoch):

  print("رقم ال Loop ," اللي انا شغال فيها i+1)

  sum_squared_error = 0.0


  for j in range(len(inputs)):

    actual = outputs[j]

    x1, x2 = inputs[j]

    unit = np.dot(np.array([x1, x2]), weight) + bias
    error = actual - unit

    print("ال Error =", error)

    sum_squared_error += error ** 2

    weight[0] += learning_rate * error * x1
    weight[1] += learning_rate * error * x2
    bias += learning_rate * error

print("ال تربيع Total Error=", sum_squared_error / len(inputs), "\n")
```

ال output 😉



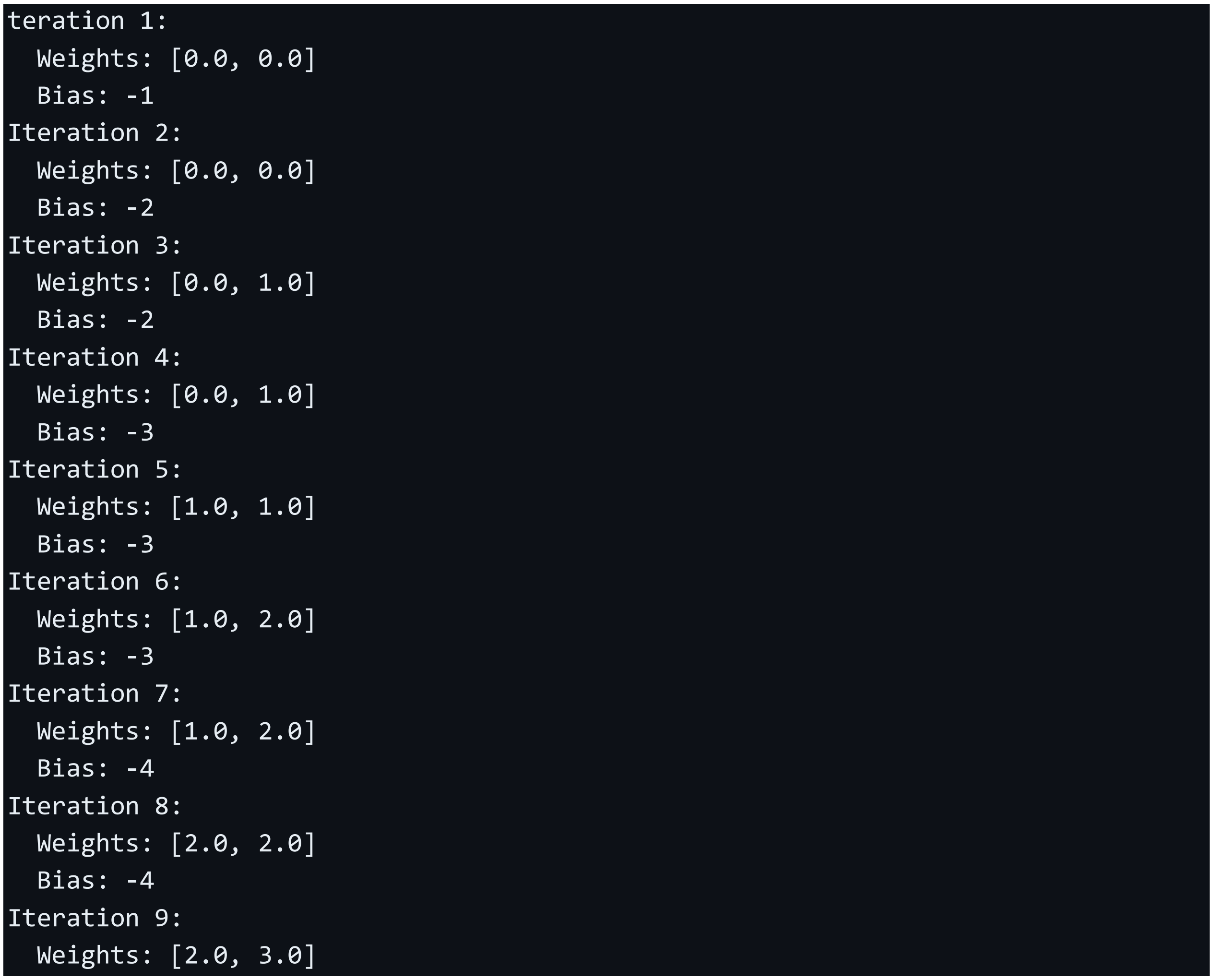
```
ال  رقم Loop 1  فيها شغال انا اللي
ال  Error = -0.09999999999999998
ال  Error = -1.08
ال  Error = -1.296
ال  Error = 0.35519999999999996

ال  رقم Loop 2  فيها شغال انا اللي
ال  Error = 0.50624
ال  Error = -0.863488
ال  Error = -0.8633855999999998
ال  Error = 0.58870272

ال  رقم Loop 3  فيها شغال انا اللي
ال  Error = 0.6656112639999998
ال  Error = -0.7689351168000002
ال  Error = -0.7500040601599999
ال  Error = 0.6723911761920002
ال  تربيع Total Error= 0.5122288881723134
```

كود ال preceptron :)

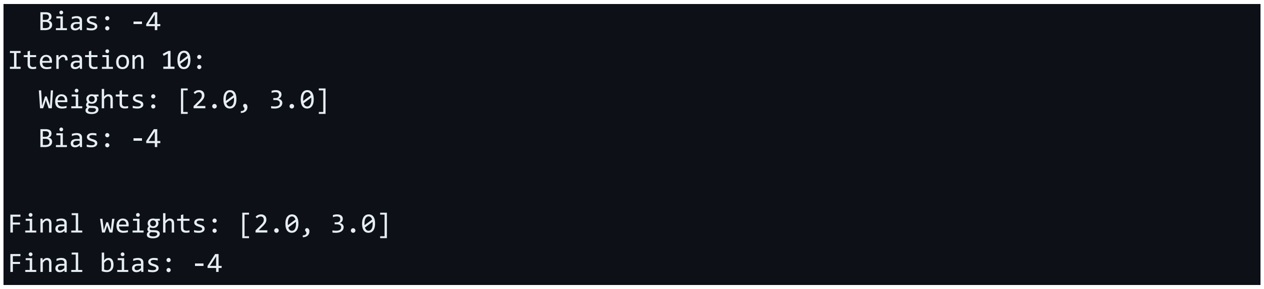
```
import numpy as np

import sys

sys.stdout.reconfigure(encoding='utf-8')


class Perceptron:

  def _init_(self, learning_rate=1):

    self.lr = learning_rate

    self.weights = np.zeros(2)

    self.bias = 0


  def _activation(self, x):

    return np.sign(x)


  def predict(self, X):

    linear_output = np.dot(X, self.weights) + self.bias

    return self._activation(linear_output)



  def fit(self, X, y):

    converged = False

    iteration = 0

    while not converged:
            weights_old = np.copy(self.weights)
            bias_old = self.bias
            for i in range(len(X)):
                x_i = X[i]
                y_in = np.dot(x_i, self.weights) + self.bias
                y = self._activation(y_in)


                if y != y_train[i]:
                    self.weights += self.lr * y_train[i] * x_i
                    self.bias += self.lr * y_train[i]

            iteration += 1

            print(f"Iteration {iteration}:")
            w = list(reversed(self.weights))
            print("  Weights:", w)
            print("  Bias:", self.bias)


            if np.array_equal(weights_old, self.weights) and bias_old == self.bias:
                converged = True

perceptron = Perceptron()

X_train = np.array([[1, 1], [0, 1], [1, 0], [-1, 0]])
y_train = np.array([1, -1, -1, -1])
```

```
perceptron.fit(X_train, y_train)


x = list(reversed(perceptron.weights))

print("\nFinal weights:", x)

print("Final bias:", perceptron.bias)


test_data = np.array([[1, 1], [0, 1], [1, 0], [-1, 0]])

predictions = perceptron.predict(test_data)
```

ال output :)



```
teration 1:
  Weights: [0.0, 0.0]
  Bias: -1
Iteration 2:
  Weights: [0.0, 0.0]
  Bias: -2
Iteration 3:
  Weights: [0.0, 1.0]
  Bias: -2
Iteration 4:
  Weights: [0.0, 1.0]
  Bias: -3
Iteration 5:
  Weights: [1.0, 1.0]
  Bias: -3
Iteration 6:
  Weights: [1.0, 2.0]
  Bias: -3
Iteration 7:
  Weights: [1.0, 2.0]
  Bias: -4
Iteration 8:
  Weights: [2.0, 2.0]
  Bias: -4
Iteration 9:
  Weights: [2.0, 3.0]
```



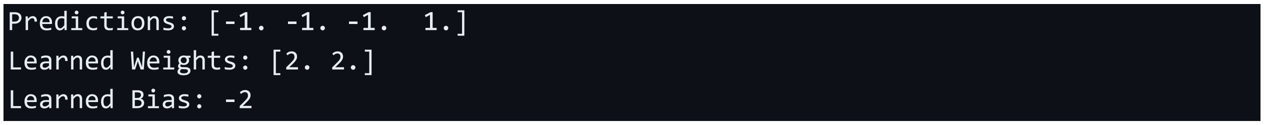
```
  Bias: -4
Iteration 10:
  Weights: [2.0, 3.0]
  Bias: -4

Final weights: [2.0, 3.0]
Final bias: -4
```

كود ال Hebb ;)

```
import numpy as np

import sys

 sys.stdout.reconfigure(encoding='utf-8')


class HebbianLearning:

    def _init_(self, num_inputs):

        self.weights = np.zeros(num_inputs)

        self.bias = 0


    def train(self, input_data, desired_output):

        for input_vector, target_output in zip(input_data, desired_output):

            activations = input_vector

            output = target_output

            self.weights += np.multiply(activations, output)

            self.bias += output


    def predict(self, input_vector):

        output = np.dot(input_vector, self.weights) + self.bias

        return np.sign(output)
```

```
if _name_ == "_main_":
    input_data = np.array([[-1, -1], [-1, 1], [1, -1], [1, 1]])
    desired_output = np.array([-1, -1, -1, 1])
    hebb_net = HebbianLearning(num_inputs=input_data.shape[1])
    hebb_net.train(input_data, desired_output)
    predictions = hebb_net.predict(input_data)
    print("Predictions:", predictions)
    print("Learned Weights:", hebb_net.weights)
    print("Learned Bias:", hebb_net.bias)
```

ال output ;)



```
Predictions: [-1. -1. -1.  1.]
Learned Weights: [2. 2.]
Learned Bias: -2
```