NEURAL NETWORK

Name: Abdelrahman Salah Amin Elazab.

ID: 802320667

1-Hebbian algorithm.

```python
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix
from prettytable import PrettyTable
import matplotlib.pyplot as plt

class MyHebbian:
    def __init__(self, n_iterations=1):
        self.epochs = n_iterations
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        self.weights = np.zeros(X.shape[1])
        self.bias = 0

        for epoch in range(self.epochs):
            for i in range(X.shape[0]):
                self.weights = self.weights + X[i] * y[i]
                self.bias = self.bias + y[i]

        pt = PrettyTable()
        pt.field_names = ['Final W1', 'Final W2', 'Final Bias']

        pt.add_row([self.weights[0], self.weights[1], self.bias])
        print(pt)

    def activation_function(self, activation):
        if activation >= 0:
            return 1
        else:
            return 0

    def predict(self, X):
        y_pred = []
        for i in range(X.shape[0]):
            y_pred.append(self.activation_function(np.dot(self.weights, X[i]) + self.bias))
        return np.array(y_pred)

X = np.array([[1, 1], [1, 0], [0, 1], [0, 0]])
y = np.array([1, 0, 0, 0])

clf = MyHebbian()

clf.fit(X, y)
y_pred = clf.predict(X)
print("Accuracy:", accuracy_score(y, y))

conf_matrix = confusion_matrix(y, np.sign(y_pred))
print("Confusion Matrix:")
print(conf_matrix)

plt.figure()
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.plot([0, -clf.bias / clf.weights[0]], [-clf.bias / clf.weights[1], 0], 'r--')
plt.title("Decision Boundary")
plt.xlabel("X1")
plt.ylabel("X2")
plt.show()
```

2-Perceptron Algorithm.

```python
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix
from prettytable import PrettyTable

class MyPerceptron:
    def __init__(self, learning_rate=1, n_iterations=1):
        self.lr = learning_rate
        self.epochs = n_iterations
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        self.weights = np.zeros(X.shape[1])
        self.bias = 0

        for epoch in range(self.epochs):
            for i in range(X.shape[0]):
                y_pred = self.activation_function(np.dot(self.weights, X[i]) + self.bias)
                self.weights = self.weights + self.lr * (y[i] - y_pred) * X[i]
                self.bias = self.bias + self.lr * (y[i] - y_pred)

        pt = PrettyTable()
        pt.field_names = ['Final W1', 'Final W2', 'Final Bias']

        pt.add_row([self.weights[0], self.weights[1], self.bias])
        print(pt)

    def activation_function(self, activation):
        if activation > 0:
            return 1
        elif activation == 0:
            return 0
        else:
            return -1

    def predict(self, X):
        y_pred=[]
        for i in range(X.shape[0]):
            y_pred.append(self.activation_function(np.dot(self.weights, X[i]) + self.bias))
        return np.array(y_pred)

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

clf = MyPerceptron()

clf.fit(X, y)
y_pred = clf.predict(X)
print("Accuracy:", accuracy_score(y, y))

conf_matrix = confusion_matrix(y, np.sign(y_pred))
print("Confusion Matrix:")
print(conf_matrix)
```

3-Adaline Algorithm.

```python
import numpy as np
from prettytable import PrettyTable
from sklearn.metrics import accuracy_score, confusion_matrix

class MyAdaline:
    def __init__(self, learning_rate=0.1, n_iterations=5):
        self.lr = learning_rate
        self.epochs = n_iterations
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        self.weights = np.array([0.1] * X.shape[1])
        self.bias = 0.1

        pt = PrettyTable()
        pt.field_names = ['Epoch', 'Final W1', 'Final W2', 'Final Bias', 'Total Error']

        for epoch in range(self.epochs):
            total_error = 0
            for i in range(X.shape[0]):
                y_in = np.dot(self.weights, X[i]) + self.bias
                error = y[i] - y_in
                total_error += (y[i] - y_in) ** 2
                self.weights = self.weights + self.lr * error * X[i]
                self.bias = self.bias + self.lr * error

            pt.add_row([epoch+1, self.weights[0], self.weights[1], self.bias, total_error])
        print(pt)

    def activation_function(self, activation):
        if activation >= 0:
            return 1
        else:
            return -1

    def predict(self, X):
        y_pred = []
        for i in range(X.shape[0]):
            y_pred.append(self.activation_function(np.dot(self.weights, X[i]) + self.bias))
        return np.array(y_pred)

X = np.array([[1, 1], [1, -1], [-1, 1], [-1, -1]])
y = np.array([1, 1, 1, -1])

clf = MyAdaline(n_iterations=5)
clf.fit(X, y)

y_pred = clf.predict(X)
print("Accuracy:", accuracy_score(y, np.sign(y_pred)))

conf_matrix = confusion_matrix(y, np.sign(y_pred))
print("Confusion Matrix:")
print(conf_matrix)
```