**Problem Statement:**

Apply Madeline algorithm for Bipolar XOR gates. (Weights are randomly initialized, v0, v1, v2 = 0.5, bias values are set to 1, learning rate = 0.5, Network topology: 2-2-1).

**Data Description:**

Here Dataset is Bipolar XOR gate with 4 rows as the input of the network.  
**Procedure:**

Step 1: Initialize V0, V, V1 with 0.5 and other weights w0, w1, w2, w01, w02 by small random values. All bias inputs are set to 1.

Step 2: Set the learning rate h to a suitable value.

Step 3: For each bipolar training pair (s : t), do Steps 4-6.

Step 4: Activate the input units: x1 = s1, x2 = s2, all biases are set to 1 permanently.

Step 5:

Propagate the input signals through the net to the output unit Y.

5.1 Compute net inputs to the hidden units.

z\_in1 = 1 × w0 + x1 × w1 + x2 × w2

z\_in2 = 1 × w01 + x1 × w02 + x2 × w02

5.2 Compute activations of the hidden units z\_out using the bipolar step function

If z\_in ≥ 0, then z\_out = 1

If z\_in < 0, then z\_out = -1

5.3 Compute net input to the output unit

y\_in = 1 × v0 + z\_out1 × v1 + z\_out2 × v2

5.4 Find the activation of the output unit y\_out using the same activation function as in Step 5.2

If y\_in ≥ 0, then y\_out = 1

If y\_in < 0, then y\_out = -1

Step 6:

Adjust the weights of the hidden units, if required, according to the following rules:

(i) If y\_out = t, then the net yields the expected result. Weights need not be updated.

(ii) If y\_out ≠ t, then apply one of the following rules whichever is applicable.

Case I: t = 1

Find the hidden unit z whose net input z\_in is closest to 0. Adjust the weights attached to z according to the formula:

w\_j(new) = w\_j(old) + h × (1 - z\_in) × x\_j, for all j

Case II: t = -1

Adjust the weights attached to those hidden units z that have positive net input.

w\_j(new) = w\_j(old) + h × (-1 - z\_in) × x\_j, for all j

Step 7:

Test for stopping condition. It can be any one of the following:

(i) No change of weight occurs in Step 6.

(ii) The weight adjustments have reached an acceptable level.

(iii) A predefined number of iterations have been carried out.

If the stopping condition is satisfied, then stop. Otherwise, go to Step 3.

**Source Code:**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

"""Madaline procedure"""

class Madaline:

# All methods of the Madaline algorithm

def \_\_init\_\_(self,

dataset: pd.DataFrame,

input\_layer\_size: int,

hidden\_layer\_size: int,

output\_layer\_size: int,

v0: float = 0.5,

v1: float = 0.5,

v2: float = 0.5,

bias: list = [1, 1, 1],

learning\_rate: float = 0.5):

"""

Initialize the Madaline network parameters.

Args:

dataset: Input dataset in the form of a Pandas DataFrame.

input\_layer\_size: Size of the input layer.

hidden\_layer\_size: Size of the hidden layer.

output\_layer\_size: Size of the output layer.

v0, v1, v2: Bias values for each layer.

bias: List of bias terms for each hidden unit.

learning\_rate: Learning rate for weight updates.

"""

self.v0 = v0

self.v1 = v1

self.v2 = v2

self.bias = bias

self.dataset = dataset

self.learning\_rate = learning\_rate

# Ensure learning rate is within the correct range

if learning\_rate > 1:

print('Learning rate exceeds 1')

exit()

self.stop = 0

# Initialize weights randomly between 0 and 1

self.input\_weights = np.random.uniform(0, 1, (input\_layer\_size, hidden\_layer\_size))

def train(self):

"""

Train the Madaline network on the dataset.

"""

epoch = 0

while not self.stop:

print('Epoch:', epoch)

epoch += 1

print('Input weights:', self.input\_weights)

self.y = [] # Array to store the output for each data point

# Iterate through each data point in the dataset

for i in range(len(self.dataset)):

print('iteration:', i,'----------------------------')

# Step 5.1: Calculate net input to the hidden units

self.z\_in = np.matmul(self.dataset.iloc[i, :-1].to\_numpy(), self.input\_weights)

print("Net input to hidden units (z\_in):", self.z\_in)

# Step 5.2: Apply the bipolar step function to get hidden layer activations

z\_out = self.\_\_sign\_activation\_function(self.z\_in)

print("Hidden layer activations (z\_out):", z\_out)

# Step 5.3: Compute net input to the output unit

y\_in = self.v0 + np.dot(z\_out, [self.v1,self.v2])

print("Net input to output unit (y\_in):", y\_in)

# Step 5.4: Apply activation function to output unit

self.y\_out = self.\_\_sign\_activation\_function(y\_in)

print("Output unit activation (y\_out):", self.y\_out)

# Store the output of the current data point

self.y.append(self.y\_out)

# Step 6: Update weights if the output is incorrect

if self.y\_out != self.dataset.iloc[i, -1]: # Check if output matches the expected output

print("Output not matched!!")

self.\_\_update\_weight\_matrix(i)

# Check for convergence (if all outputs match target outputs)

self.stop = self.\_\_stop\_function()

def \_\_sign\_activation\_function(self, x):

"""

Bipolar step activation function.

Args:

x: Input array.

Returns:

Array with elements set to 1 if >= 0, else -1.

"""

return np.where(x >= 0, 1, -1)

def \_\_update\_weight\_matrix(self, i):

"""

Update the weight matrix based on error correction.

Args:

i: Index of the current data point.

"""

# Case I: Desired output (target) is +1

if self.dataset.iloc[i, -1] == 1:

# Find hidden unit closest to zero net input and adjust weights

j = np.argmin(np.abs(self.z\_in))

# Update weight for the chosen hidden unit

self.input\_weights[:,j] = self.input\_weights[:,j] + \

self.learning\_rate \* (self.bias[j] - self.z\_in[j]) \* \

self.dataset.iloc[i, :-1].to\_numpy()

print('Updated weights are: ',self.input\_weights)

else:

# Case II: Desired output is -1

indices = np.where(self.z\_in >= 0)[0] # Find hidden units with positive net input

for j in indices:

# Update weights for each of those units

self.input\_weights[:,j] = self.input\_weights[:,j] + \

self.learning\_rate \* (-self.bias[j] - self.z\_in[j]) \* \

self.dataset.iloc[i, :-1].to\_numpy()

print('Updated weights are: ',self.input\_weights)

def \_\_stop\_function(self):

"""

Check if the network has converged by comparing all outputs to targets.

Returns:

1 if all outputs match targets, 0 otherwise.

"""

for i in range(len(self.dataset)):

if self.y[i] != self.dataset.iloc[i, -1]: # If any output is incorrect, do not stop

return 0

return 1 # Stop if all outputs are correct

# Main execution

if \_\_name\_\_ == '\_\_main\_\_':

np.random.seed(7)

dataset = pd.read\_csv('Bipolar XOR.csv')

train\_data, test\_data = train\_test\_split(dataset, test\_size=0.2,random\_state=40)

print("Training Data:")

print(train\_data)

print("\nTesting Data:")

print(test\_data)

mad = Madaline(train\_data,input\_layer\_size=3,hidden\_layer\_size=2,output\_layer\_size=1)

mad.train()

**Output:**

Training Data:

x0 x1 x2 t

0 1 1 1 -1

1 1 1 -1 1

2 1 -1 1 1

Testing Data:

x0 x1 x2 t

3 1 -1 -1 -1

Epoch: 0

Input weights: [[0.07630829 0.77991879]

[0.43840923 0.72346518]

[0.97798951 0.53849587]]

iteration: 0 ----------------------------

Net input to hidden units (z\_in): [1.49270703 2.04187984]

Hidden layer activations (z\_out): [1 1]

Net input to output unit (y\_in): 1.5

Output unit activation (y\_out): 1

Output not matched!!

Updated weights are: [[-1.17004523 0.77991879]

[-0.80794428 0.72346518]

[-0.268364 0.53849587]]

Updated weights are: [[-1.17004523 -0.74102113]

[-0.80794428 -0.79747474]

[-0.268364 -0.98244405]]

iteration: 1 ----------------------------

Net input to hidden units (z\_in): [-1.70962551 -0.55605182]

Hidden layer activations (z\_out): [-1 -1]

Net input to output unit (y\_in): -0.5

Output unit activation (y\_out): -1

Output not matched!!

Updated weights are: [[-1.17004523 0.03700478]

[-0.80794428 -0.01944883]

[-0.268364 -1.76046996]]

iteration: 2 ----------------------------

Net input to hidden units (z\_in): [-0.63046495 -1.70401635]

Hidden layer activations (z\_out): [-1 -1]

Net input to output unit (y\_in): -0.5

Output unit activation (y\_out): -1

Output not matched!!

Updated weights are: [[-0.35481275 0.03700478]

[-1.62317676 -0.01944883]

[ 0.54686847 -1.76046996]]

Epoch: 1

Input weights: [[-0.35481275 0.03700478]

[-1.62317676 -0.01944883]

[ 0.54686847 -1.76046996]]

iteration: 0 ----------------------------

Net input to hidden units (z\_in): [-1.43112104 -1.74291401]

Hidden layer activations (z\_out): [-1 -1]

Net input to output unit (y\_in): -0.5

Output unit activation (y\_out): -1

iteration: 1 ----------------------------

Net input to hidden units (z\_in): [-2.52485798 1.77802591]

Hidden layer activations (z\_out): [-1 1]

Net input to output unit (y\_in): 0.5

Output unit activation (y\_out): 1

iteration: 2 ----------------------------

Net input to hidden units (z\_in): [ 1.81523247 -1.70401635]

Hidden layer activations (z\_out): [ 1 -1]

Net input to output unit (y\_in): 0.5

Output unit activation (y\_out): 1

**Discussion:**

This code implements a Madaline (Multiple Adaptive Linear Element) neural network that is trained to learn from a dataset. The Madaline network uses a simple architecture consisting of an input layer, hidden layer, and output layer. The training process involves iterating through the data points, calculating the net input to the hidden and output layers, applying activation functions (bipolar step function), and updating the weights based on errors using the Perceptron learning rule.