Machine Learning
Lab Assignment 2

Mohammed Junaid Anwar Qader
21CSB0B36

2. Implement perceptron learning algorithm and attempt to solve two input i) AND gate ii) Or Gate iii) EXOR gate problems.

-> AND and OR gate can be solved with a single layer perceptron, as they are linearly separable

Below is the implementation of AND and OR gate using a perceptron

```python
import numpy as np

class Perceptron:
    def __init__(self, input_size, weights, learning_rate=0.0001, epochs=200000000):
        self.weights = weights
        self.learning_rate = learning_rate
        self.epochs = epochs

    # activation function
    def activation(self, x):
        return 1 if x > 0 else 0

    # prediction part
    def prediction(self, inputs):
        summation = self.weights[0]
        for i in range(len(inputs)):
            summation += self.weights[i+1]*inputs[i]
        return self.activation(summation)

    # training part
    def learning(self, train_input, labels):
        # keeping a limit of epochs to avoid infinite loop
        for j in range(self.epochs):
            error = 0
            # in each epoch, we are going through all the training data
            for i in range(len(train_input)):
                prediction = self.prediction(train_input[i])
                # updating the wweights by (target-predicted)*input*learning_rate
                self.weights[0] += self.learning_rate * (labels[i] - prediction) * 1
                for k in range(len(train_input[i])):
                    self.weights[k+1] += self.learning_rate * (labels[i] - prediction) * train_input[i][k]
                error += abs(labels[i] - prediction)
            if error == 0:
                print("Learning done at iteration:", j," (as error = 0)")
                break
```

Input and learning for AND gate ->

```
# And gate first
weights = [4, 3, -5]

# Training data for AND gate
and_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
and_labels = np.array([0, 0, 0, 1])

# Creating perceptrons for each gate
and_perceptron = Perceptron(2, weights)

# Training perceptrons
and_perceptron.learning(and_inputs, and_labels)

# now for prediction
print("AND gate prediction")
print("A_| B___|_Y")
print("0 | 0   |",and_perceptron.prediction([0, 0]))
print("0 | 1   |",and_perceptron.prediction([0, 1]))
print("1 | 0   |",and_perceptron.prediction([1, 0]))
print("1 | 1   |",and_perceptron.prediction([1, 1]))
```

Output for AND GATE ->

```
Learning done at iteration: 68334  (as error = 0)
AND gate prediction
A_| B___|_Y
0 | 0   | 0
0 | 1   | 0
1 | 0   | 0
1 | 1   | 1
```

Input and training for OR gate ->

```
# Or gate first
weights = [4, 3, -5]

# Training data for OR gate
or_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
or_labels = np.array([0, 1, 1, 1])

# craeting a perceptron
or_perceptron = Perceptron(2, weights)

# training perceptron
or_perceptron.learning(or_inputs, or_labels)

# now for prediction
print("OR gate prediction")
print("A_| B___|_Y")
print("0 | 0   |",or_perceptron.prediction([0, 0]))
print("0 | 1   |",or_perceptron.prediction([0, 1]))
print("1 | 0   |",or_perceptron.prediction([1, 0]))
print("1 | 1   |",or_perceptron.prediction([1, 1]))
```

Output for OR gate ->

```
Learning done at iteration: 90001  (as error = 0)
OR gate prediction
A_| B___|_Y
0 | 0   | 0
0 | 1   | 1
1 | 0   | 1
1 | 1   | 1
```

2. Now for the XOR gate part, this is not a linearly separable problem so we divide into 2 layers -> see below for explanantion

```
# Q : create a Multi Level perceptron for solving XOR problem
# XOR problem is not linearly separable
# hence we need to use a multi level perceptron to solve this

# A XOR B = A'B + AB' (X' -> NOT X)
# A XOR B = (A OR B) AND (A NAND B)


# so technically the smaller operations in it use AND and OR gate, which are lineraly separable
# using both AND and OR gate perceptrons we can solve the XOR problem
'''
Approach:

let Z1 = A'B  weights used here are : w1
let Z2 = AB'  weights used here are : w2

let Z = Z1 + Z2 weights used here are : w3

now lets see the truth table
A_|__B__|_Z1__|_Z2_|__Z
0 |  0   | 0   | 0  |  0
0 |  1   | 1   | 0  |  1
1 |  0   | 0   | 1  |  1
1 |  1   | 0   | 0  |  0

also you can verify by drawing graphs that Z1, Z2, Z all are linearly separable
so all can be a 1d perceptron, but overall its a 2 layer one by combing all

Steps :
1. train weights w1 using the table and a 1d perceptron
2. train similarly for w2 and w3
3. now when anyone gives input, first evaluate z1,z2 then out in w3 to get z value.
4. for now activation im using the normal activation of sgn(x) 1 : x > 0 ,0 : else

'''
```

Perceptron class -> (same as before) just combing the multiple weights manually

```python
import numpy as np


class Perceptron:
    def __init__(self, input_size, weights, learning_rate=0.0001, epochs=200000000):
        self.weights = weights
        self.learning_rate = learning_rate
        self.epochs = epochs

    # activation function
    def activation(self, x):
        return 1 if x > 0 else 0

    # prediction part
    def prediction(self, inputs):
        summation = self.weights[0]
        for i in range(len(inputs)):
            summation += self.weights[i+1]*inputs[i]
        return self.activation(summation)

    # training part
    def learning(self, train_input, labels):
        # keeping a limit of epochs to avoid infinite loop
        for j in range(self.epochs):
            error = 0
            # in each epoch, we are going through all the training data
            for i in range(len(train_input)):
                prediction = self.prediction(train_input[i])
                # updating the wweights by (target-predicted)*input*learning_rate
                self.weights[0] += self.learning_rate * (labels[i] - prediction) * 1
                for k in range(len(train_input[i])):
                    self.weights[k+1] += self.learning_rate * (labels[i] - prediction) * train_input[i][k]
                error += abs(labels[i] - prediction)
            if error == 0:
                print("Learning done at iteration:", j," (as error = 0)")
                break
```

Training, learning and outputs for z1, z2, and z

```
# neew to train 3 things z1, z2, z
# z1 first
w1 = [1, 1, 1]
w2 = [1, 1, 1]
w3 = [1, 1, 1]

z1_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
z1_labels = np.array([0, 1, 0, 0])

z2_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
z2_labels = np.array([0, 0, 1, 0])

z_inputs = np.array([[0, 0], [1, 0], [0, 1], [0, 0]])
z_labels = np.array([0, 1, 1, 0])

z1_perceptron = Perceptron(2, w1)
z2_perceptron = Perceptron(2, w2)
z_perceptron = Perceptron(2, w3)

z1_perceptron.learning(z1_inputs, z1_labels)
# now for prediction of Z1
print("Z1 gate prediction")
print("A_| B___|_Z1")
print("0 | 0   |",z1_perceptron.prediction([0, 0]))
print("0 | 1   |",z1_perceptron.prediction([0, 1]))
print("1 | 0   |",z1_perceptron.prediction([1, 0]))
print("1 | 1   |",z1_perceptron.prediction([1, 1]))

z2_perceptron.learning(z2_inputs, z2_labels)
# now for prediction of Z1
print("Z2 gate prediction")
print("A_| B___|_Z2")
print("0 | 0   |",z2_perceptron.prediction([0, 0]))
print("0 | 1   |",z2_perceptron.prediction([0, 1]))
print("1 | 0   |",z2_perceptron.prediction([1, 0]))
print("1 | 1   |",z2_perceptron.prediction([1, 1]))

z_perceptron.learning(z_inputs, z_labels)
# now for prediction of Z1
print("Z gate prediction")
print("A_| B___|_Z")
print("0 | 0   |",z_perceptron.prediction([0, 0]))
print("0 | 1   |",z_perceptron.prediction([0, 1]))
print("1 | 0   |",z_perceptron.prediction([1, 0]))
print("0 | 0   |",z_perceptron.prediction([0, 0]))
```

Outputs ->

```
Learning done at iteration: 6111  (as error = 0)
Z1 gate prediction
A_| B___|_Z1
0 | 0   | 0
0 | 1   | 1
1 | 0   | 0
1 | 1   | 0
Learning done at iteration: 6111  (as error = 0)
Z2 gate prediction
A_| B___|_Z2
0 | 0   | 0
0 | 1   | 0
1 | 0   | 1
1 | 1   | 0
Learning done at iteration: 5001  (as error = 0)
Z gate prediction
A_| B___|_Z
0 | 0   | 0
0 | 1   | 1
1 | 0   | 1
0 | 0   | 0
```

Now combining all z1,z2 and z to get the final output for XOR gate ->

```
# now as all z1, z2, z all are trained,
# using the weights of z1, z2, z we can get the XOR gate
# we need to now combine all these to get the XOR gate

def XOR(a, b):
    z1_output = z1_perceptron.prediction([a,b])
    z2_output = z2_perceptron.prediction([a,b])
    z_output = z_perceptron.prediction([z1_output,z2_output])
    return z_output

print("XOR gate prediction")
print("A_| B___|_Y")
print("0 | 0   |",XOR(0 ,0))
print("0 | 1   |",XOR(0 ,1))
print("1 | 0   |",XOR(1 ,0))
print("1 | 1   |",XOR(1 ,1))
```

✓ 0.0s

```
XOR gate prediction
A_| B___|_Y
0 | 0   | 0
0 | 1   | 1
1 | 0   | 1
1 | 1   | 0
```

THE END