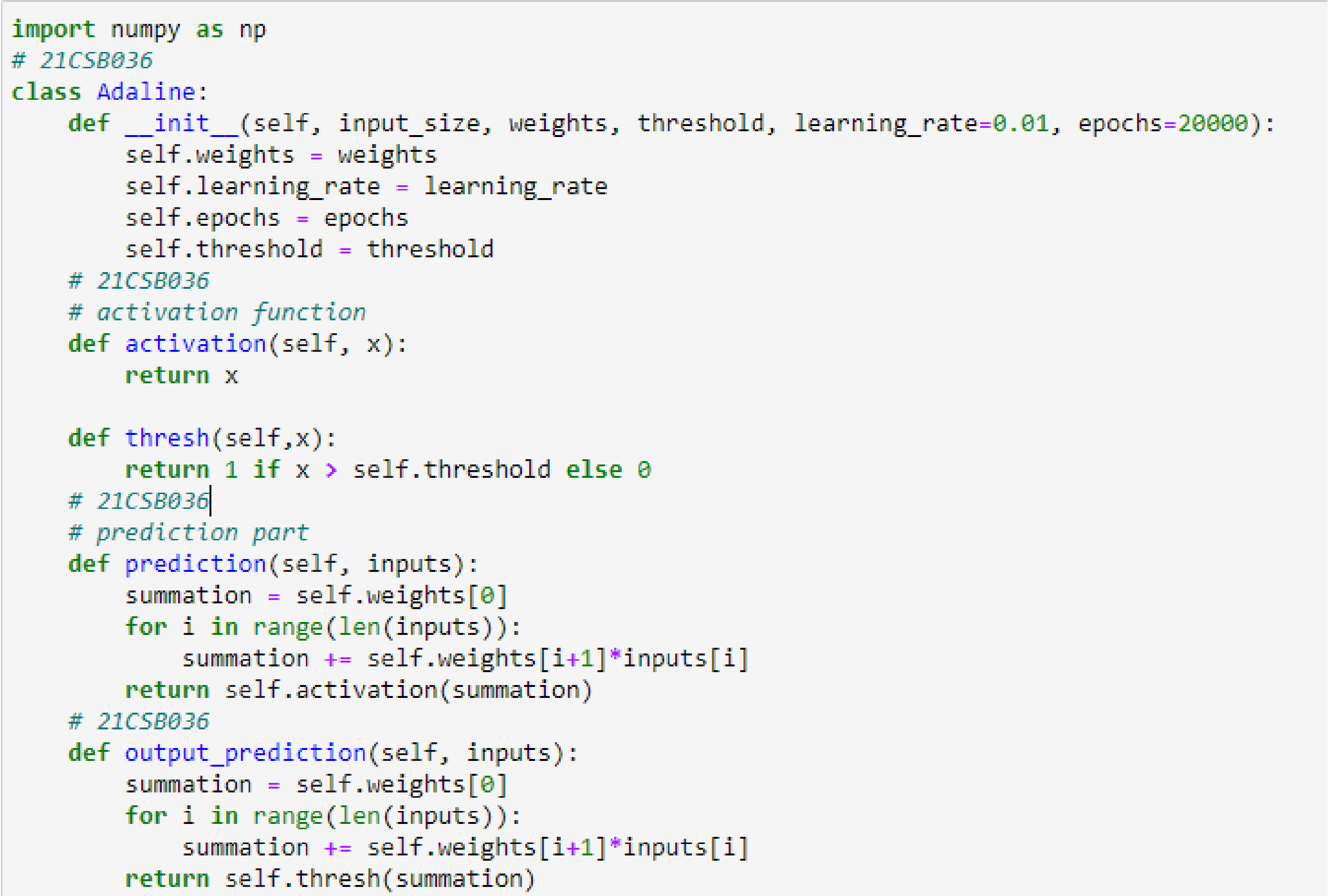
ML LAB ASSIGNMENT 3

Mohammed Junaid Anwar Qader
21CSB0B36

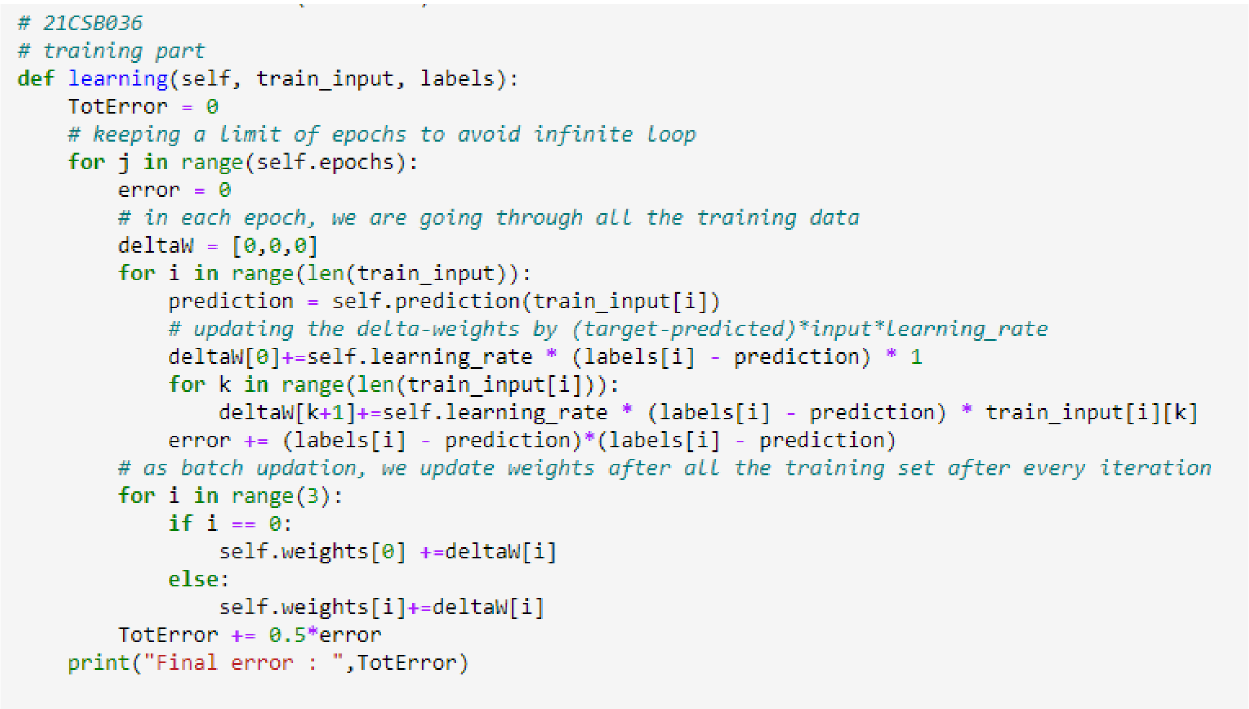
Q.Implement Adaline learning algorithm and attempt to solve two input i) AND gate ii) Or Gate iii) XNOR gate problems.

ADALINE CLASS:



```
import numpy as np
# 21CSB036
class Adaline:
    def __init__(self, input_size, weights, threshold, learning_rate=0.01, epochs=20000):
        self.weights = weights
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.threshold = threshold
    # 21CSB036
    # activation function
    def activation(self, x):
        return x

    def thresh(self,x):
        return 1 if x > self.threshold else 0
    # 21CSB036
    # 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)
    # 21CSB036
    def output_prediction(self, inputs):
        summation = self.weights[0]
        for i in range(len(inputs)):
            summation += self.weights[i+1]*inputs[i]
        return self.thresh(summation)
```

ADALINE -> CLASS
ACTIVATION -> y = x
Threshold -> for these examples taken as 0.5 (all three)
Prediction -> used during learning phase (gives value based on activation)
Output_prediction -> used during output (gives output based on threshold value)

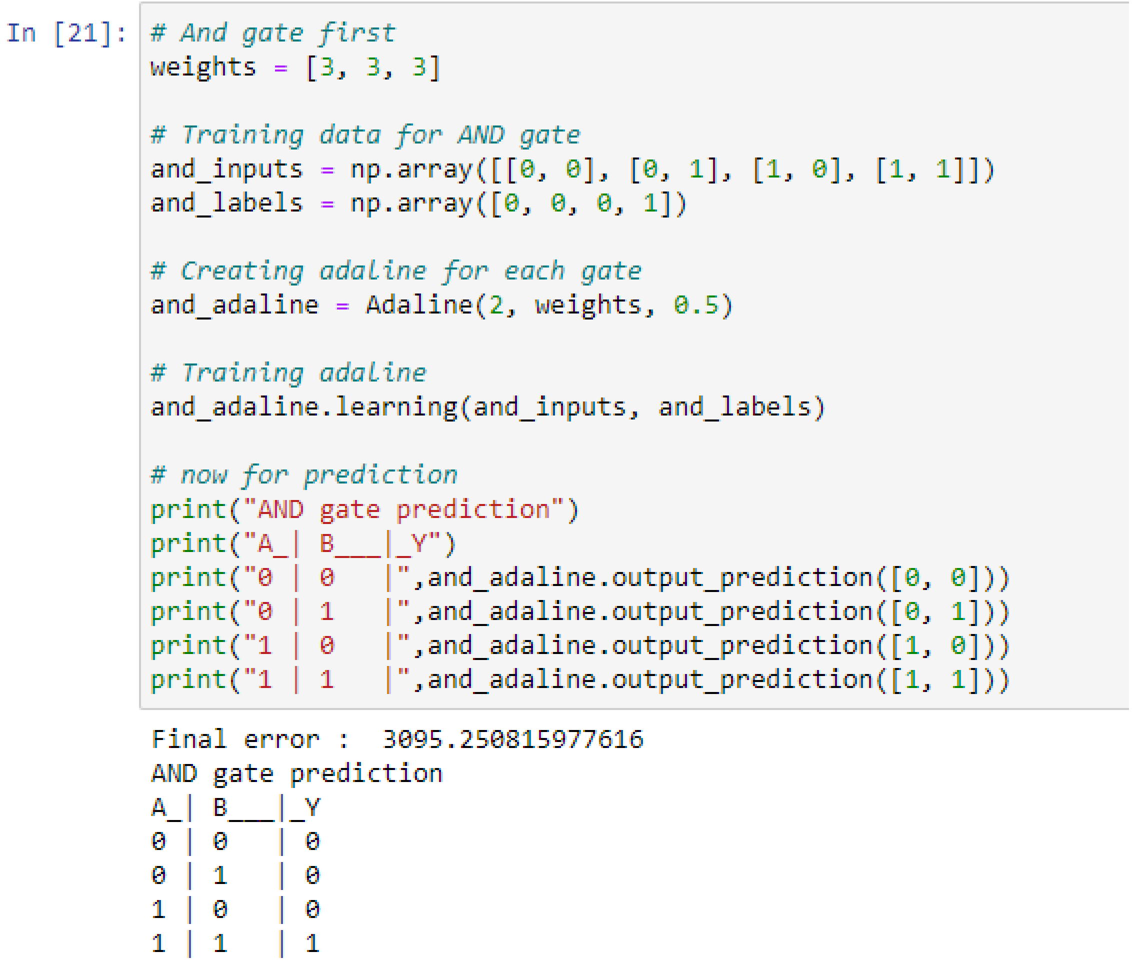


```
# 21CSB036
# training part
def learning(self, train_input, labels):
    TotError = 0
    # keeping a limit of epochs to avoid infinite loop
    for j in range(self.epochs):
        error = 0
        # in each epoch, we are going through all the training data
        deltaW = [0,0,0]
        for i in range(len(train_input)):
            prediction = self.prediction(train_input[i])
            # updating the delta-weights by (target-predicted)*input*learning_rate
            deltaW[0]+=self.learning_rate * (labels[i] - prediction) * 1
            for k in range(len(train_input[i])):
                deltaW[k+1]+=self.learning_rate * (labels[i] - prediction) * train_input[i][k]
            error += (labels[i] - prediction)*(labels[i] - prediction)
        # as batch updation, we update weights after all the training set after every iteration
        for i in range(3):
            if i == 0:
                self.weights[0] +=deltaW[i]
            else:
                self.weights[i]+=deltaW[i]
        TotError += 0.5*error
    print("Final error : ",TotError)
```

learning-> batch wise, updated each weights after entire train_set, for every iteration

So weights only get updated, after every epoch

1. AND gate

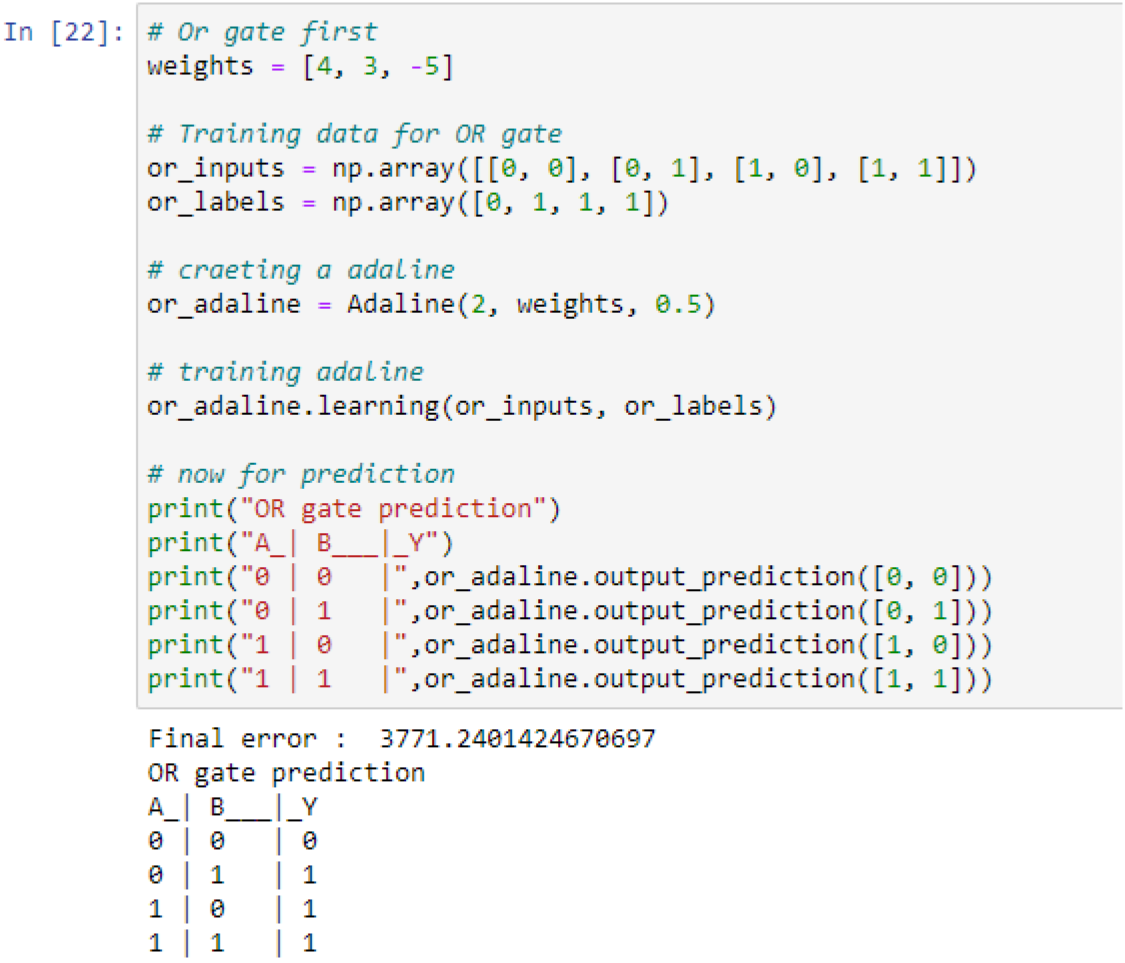


```
In [21]: # And gate first
         weights = [3, 3, 3]

         # 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 adaline for each gate
         and_adaline = Adaline(2, weights, 0.5)

         # Training adaline
         and_adaline.learning(and_inputs, and_labels)

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

```
Final error :  3095.250815977616
AND gate prediction
A_| B___|_Y
0 | 0   | 0
0 | 1   | 0
1 | 0   | 0
1 | 1   | 1
```

Weights can be taken as anything initially
Inputs and labels given, and given for learning
Then prediction.

2. OR gate



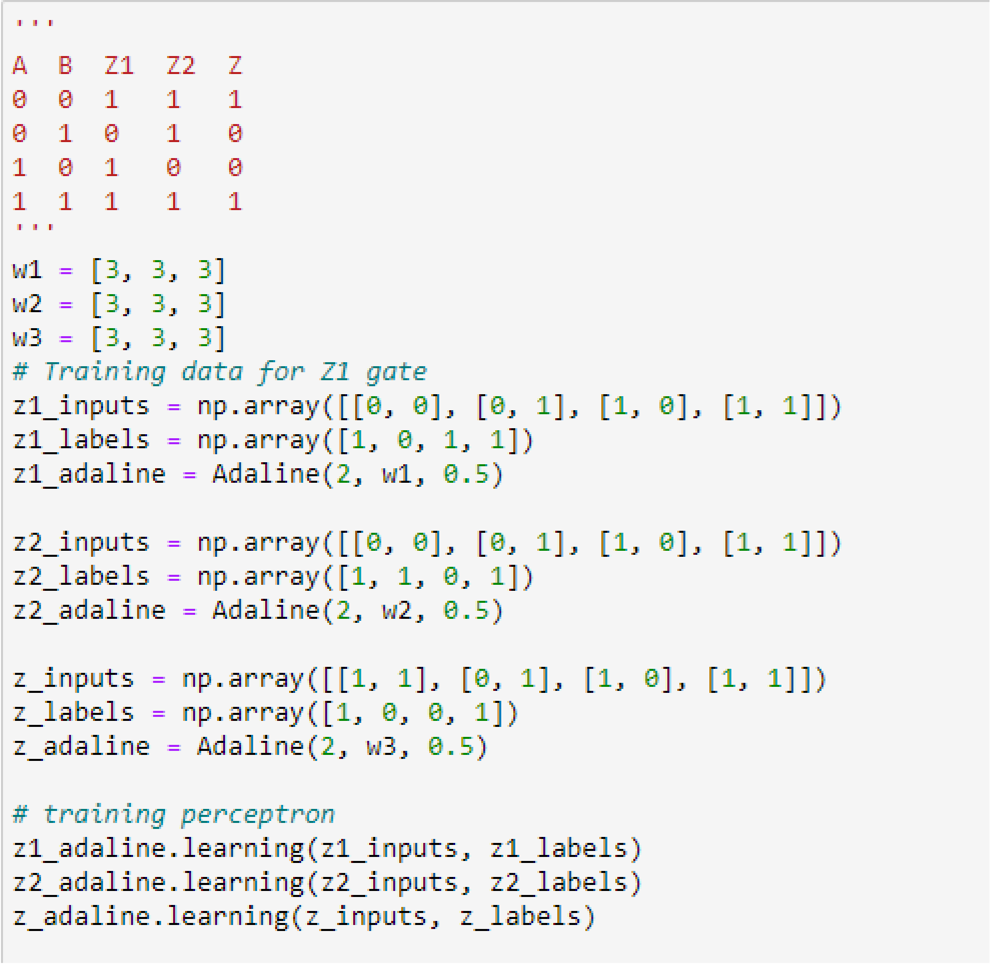
```
In [22]: # Or gate first
         weights = [4, 3, -5]

         # 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 adaline
         or_adaline = Adaline(2, weights, 0.5)

         # training adaline
         or_adaline.learning(or_inputs, or_labels)

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

```
Final error :  3771.2401424670697
OR gate prediction
A_| B___|_Y
0 | 0    | 0
0 | 1    | 1
1 | 0    | 1
1 | 1    | 1
```

Weights can be taken as anything initially
Inputs and labels given, and given for learning
Then prediction.

3. XNOR gate

For this normal adaline cannot solve it as it is not linearly separable, so we use the same strategy of solving non-linear perceptron for adaline



```
'''
A  B  Z1  Z2  Z
0  0  1   1   1
0  1  0   1   0
1  0  1   0   0
1  1  1   1   1
'''
w1 = [3, 3, 3]
w2 = [3, 3, 3]
w3 = [3, 3, 3]
# Training data for Z1 gate
z1_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
z1_labels = np.array([1, 0, 1, 1])
z1_adaline = Adaline(2, w1, 0.5)

z2_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
z2_labels = np.array([1, 1, 0, 1])
z2_adaline = Adaline(2, w2, 0.5)

z_inputs = np.array([[1, 1], [0, 1], [1, 0], [1, 1]])
z_labels = np.array([1, 0, 0, 1])
z_adaline = Adaline(2, w3, 0.5)

# training perceptron
z1_adaline.learning(z1_inputs, z1_labels)
z2_adaline.learning(z2_inputs, z2_labels)
z_adaline.learning(z_inputs, z_labels)
```

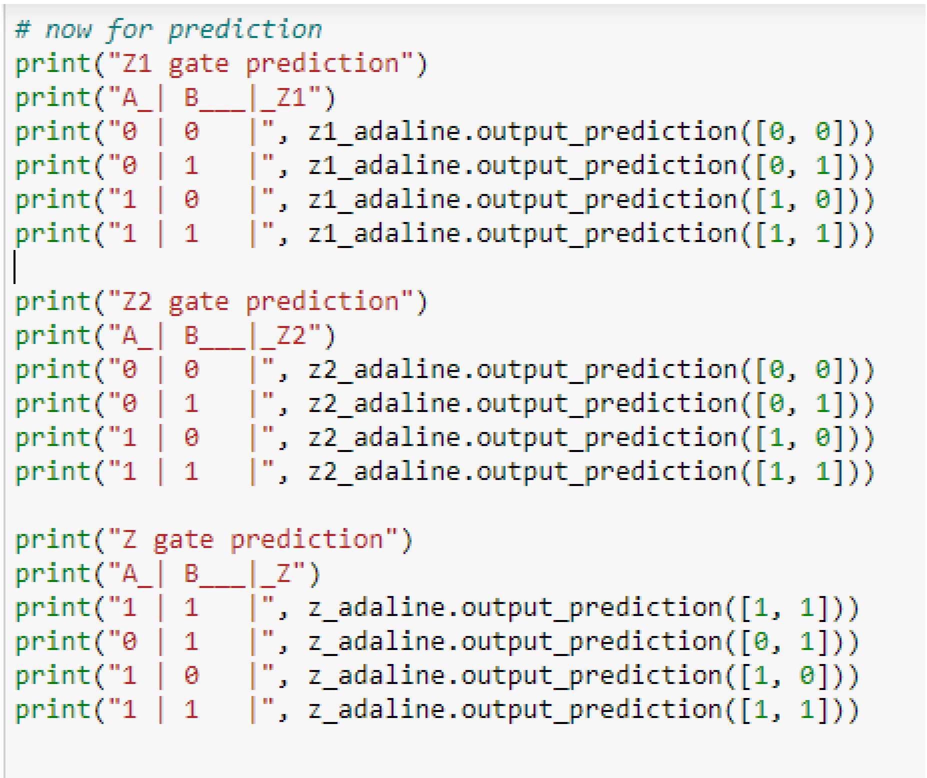
As we can see, we train Z1, Z2, zZ

Z = Z1 ^ Z2

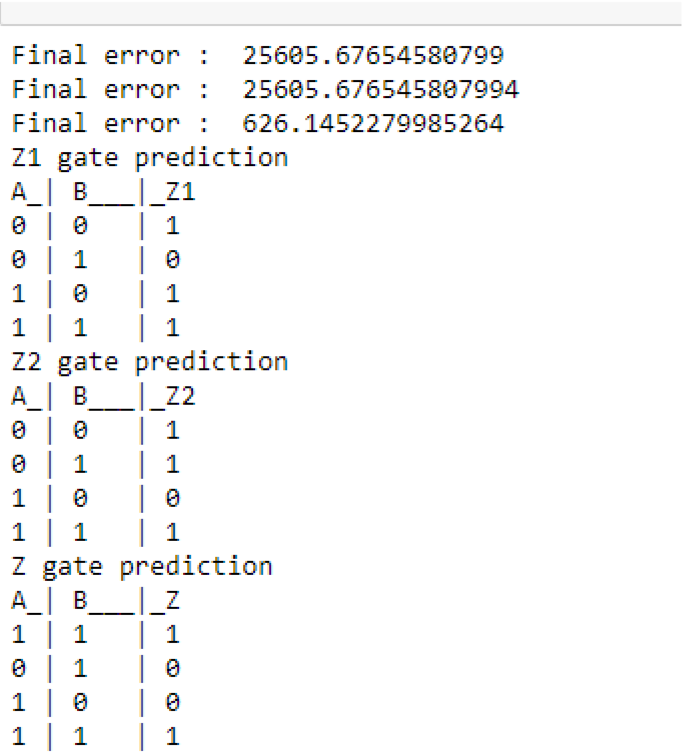
Z1 = A’ + B

Z2 = A + B’

All these separately are linearly separable



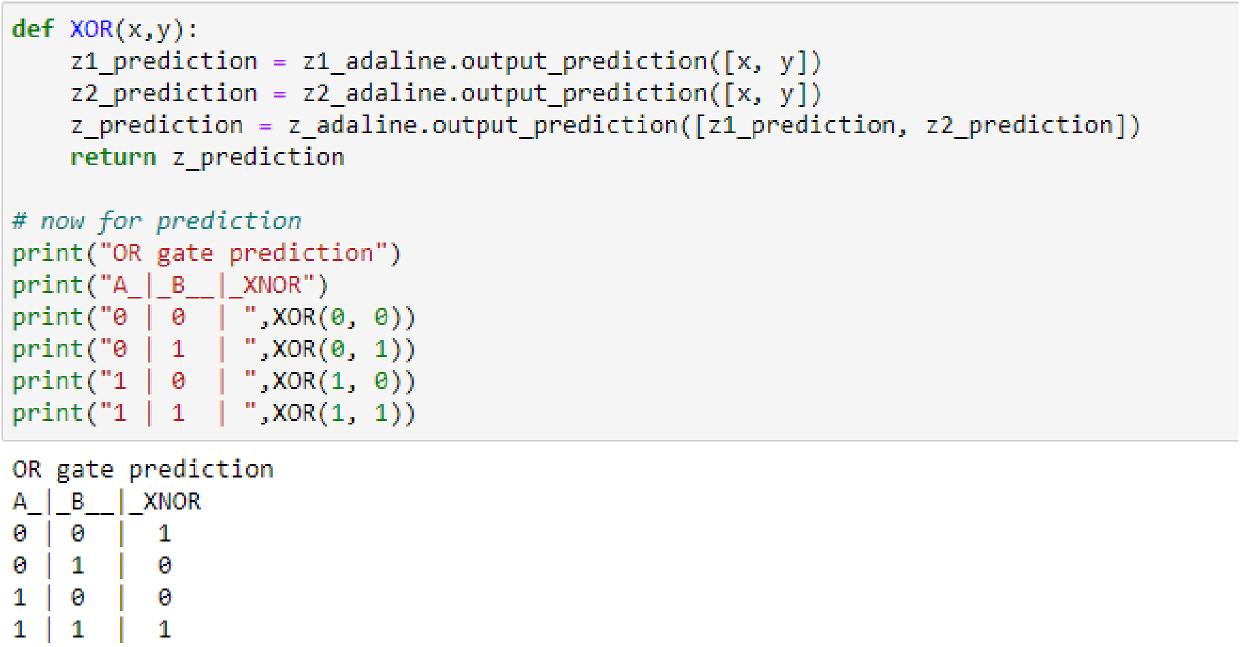
```
# now for prediction
print("Z1 gate prediction")
print("A_| B___|_Z1")
print("0 | 0   |", z1_adaline.output_prediction([0, 0]))
print("0 | 1   |", z1_adaline.output_prediction([0, 1]))
print("1 | 0   |", z1_adaline.output_prediction([1, 0]))
print("1 | 1   |", z1_adaline.output_prediction([1, 1]))

print("Z2 gate prediction")
print("A_| B___|_Z2")
print("0 | 0   |", z2_adaline.output_prediction([0, 0]))
print("0 | 1   |", z2_adaline.output_prediction([0, 1]))
print("1 | 0   |", z2_adaline.output_prediction([1, 0]))
print("1 | 1   |", z2_adaline.output_prediction([1, 1]))

print("Z gate prediction")
print("A_| B___|_Z")
print("1 | 1   |", z_adaline.output_prediction([1, 1]))
print("0 | 1   |", z_adaline.output_prediction([0, 1]))
print("1 | 0   |", z_adaline.output_prediction([1, 0]))
print("1 | 1   |", z_adaline.output_prediction([1, 1]))
```



```
Final error :  25605.67654580799
Final error :  25605.676545807994
Final error :  626.1452279985264
Z1 gate prediction
A_| B___|_Z1
0 | 0   | 1
0 | 1   | 0
1 | 0   | 1
1 | 1   | 1
Z2 gate prediction
A_| B___|_Z2
0 | 0   | 1
0 | 1   | 1
1 | 0   | 0
1 | 1   | 1
Z gate prediction
A_| B___|_Z
1 | 1   | 1
0 | 1   | 0
1 | 0   | 0
1 | 1   | 1
```

We can see the outputs for z1,z2, z respectively

Finally for XNOR gate :



```
def XOR(x,y):
    z1_prediction = z1_adaline.output_prediction([x, y])
    z2_prediction = z2_adaline.output_prediction([x, y])
    z_prediction = z_adaline.output_prediction([z1_prediction, z2_prediction])
    return z_prediction

# now for prediction
print("OR gate prediction")
print("A_|_B__|_XNOR")
print("0 | 0  | ",XOR(0, 0))
print("0 | 1  | ",XOR(0, 1))
print("1 | 0  | ",XOR(1, 0))
print("1 | 1  | ",XOR(1, 1))
```

```
OR gate prediction
A_|_B__|_XNOR
0 | 0  |  1
0 | 1  |  0
1 | 0  |  0
1 | 1  |  1
```

And we solved for XNOR gate as well…

THE END