CS4278 Neural Computing

Lab 1
The Perceptron (MLP)

WEEK 4
Autumn Semester 2024/25

Exercises

1. Implement a perceptron with fixed weights w1 = w2 =1, and bias -0.5 to compute the OR logical gate.
2. Implement a perceptron that uses the perceptron training rule to learn the logical AND gate.
3. Implement a perceptron that attempts to learn the logical XOR gate

Table1. XOR

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

4. Implement a perceptron to learn to classify breast cancer data set using python's Scikit-Learn library.
5. How does learning rate impact the error of the perceptron implemented in 4?

Solutions Overleaf …………

Solution to Exercise 1

```
#https://towardsdatascience.com/perceptrons-logical-functions-and-the-xor-problem-37ca5025790a

# define Unit Step Function
def unitStep(v):
   if v >= 0:
    return 1
   else:
     return 0

# design Perceptron Model
def perceptronModel(x, w, b):
   v = np.dot(w, x) + b
   y = unitStep(v)
   return y

# OR Logic Function
def logicFunction(x):
   w = np.array([1, 1])
   b = -0.5
   return perceptronModel(x, w, b)

# testing the Perceptron Model
test1 = np.array([0, 0])
test2 = np.array([0, 1])
test3 = np.array([1, 0])
test4 = np.array([1, 1])
print("OR({}, {}) = {}".format(0, 0, logicFunction(test1)))
print("OR({}, {}) = {}".format(0, 1, logicFunction(test2)))
print("OR({}, {}) = {}".format(1, 0, logicFunction(test3)))
print("OR({}, {}) = {}".format(1, 1, logicFunction(test3)))
```

PARTIAL Solution to Exercise 2 (and 3): code has a bug – implement and investigate.

```
#https://medium.com/analytics-vidhya/implementing-perceptron-learning-algorithm-to-solve-and-in-python-903516300b2f

import numpy as np

w = np.random.rand(1,3) * 10
w_1 = np.round(w[0][0], 1)
w_2 = np.round(w[0][1], 1)
theta = np.round(w[0][2], 1)

#Inputs
x = [ [0,0], [0,1], [1,0], [1,1]]
x_array = np.asarray(x)

# expected outputs for AND/OR/NOT/XOR
out = np.array([0,0,0,1])

#step function
def step (net):
   if net >= 0:
      return 1
   else:
      return 0
   #the error vector
```

```
    error = np.array([0,0,0,0])
    for i in range(len(x)):
        f_net = step(np.dot(np.asarray([w_1, w_2]) , x[i]) + theta)
        error[i] = out[i] - f_net
    E = np.sum(error)

    max_it = 1000
    t = 1
    learning_rate=0.1
    vals = [[w_1, w_2, theta]]

    while t < max_it and E!= 0:
        for i in range(len(x)):
            f_net = step(np.dot(np.asarray([w_1, w_2]) , x[i]) + theta)
            error[i] = out[i] - f_net
            w_1 = w_1 + learning_rate * error[i] * x[i][0]
            w_2 = w_2 + learning_rate * error[i] * x[i][1]
            theta = theta + learning_rate*error[i]
        vals.append([w_1, w_2, theta])
        E = np.sum(error)
        print(' sum of errors', E)
        t = t+1

print("w_1 ", w_1)
print("w_2 ",w_2)
test_LogicGate_00 = step(np.dot(np.asarray([w_1, w_2]) , np.array([0,0])) + theta)
test_LogicGate_01 = step(np.dot(np.asarray([w_1, w_2]) , np.array([0,1])) + theta)
test_LogicGate_10 = step(np.dot(np.asarray([w_1, w_2]) , np.array([1,0])) + theta)
test_LogicGate_11 = step(np.dot(np.asarray([w_1, w_2]) , np.array([1,1])) + theta)
print("LogicGate [0,0]",test_LogicGate_00)
print("LogicGate [0,1]",test_LogicGate_01)
print("LogicGate [1,0]",test_LogicGate_10)
print("LogicGate [1,1]",test_LogicGate_11)
```

Solution to Exercise 4

```
#From
#https://dzone.com/articles/perceptron-explained-using-python-example-data-ana

import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Perceptron implementation

class CustomPerceptron(object):
    def __init__(self, n_iterations=1000, random_state=1, learning_rate=0.001):
        self.n_iterations = n_iterations
        self.random_state = random_state
        self.learning_rate = learning_rate
        self.testScore = []
        self.trainScore = []


'''
Stochastic Gradient Descent
1. Weights are updated based on each training examples.
2. Learning of weights can continue for multiple iterations
```

```
3. Learning rate needs to be defined
'''
     def fit(self, X, y):
         rgen = np.random.RandomState(self.random_state)
         self.coef_ = rgen.normal(loc=0.0, scale=0.01, size=1 + X.shape[1])
         for _ in range(self.n_iterations):
              for xi, expected_value in zip(X, y):
                         predicted_value = self.predict(xi)
                         self.coef_[1:] = self.coef_[1:] + self.learning_rate * (expected_value - predicted_value) * xi
                         self.coef_[0] = self.coef_[0] + self.learning_rate * (expected_value - predicted_value) * 1
         prcptrn.score(X_test, y_test, "test")
         prcptrn.score(X_train, y_train,"train")

    '''
    Net Input is sum of weighted input signals
    '''
    def net_input(self, X):
             weighted_sum = np.dot(X, self.coef_[1:]) + self.coef_[0]
             return weighted_sum

    '''
    Activation function is fed the net input and the unit step function
    is executed to determine the output.
    '''
    def activation_function(self, X):
        weighted_sum = self.net_input(X)
        return np.where(weighted_sum >= 0.0, 1, 0)

    '''
    Prediction is made on the basis of output of activation function
    '''
    def predict(self, X):
    return self.activation_function(X)

    '''
    Model score is calculated based on comparison of
    expected value and predicted value
    '''
    def score(self, X, y, switch):
         misclassified_data_count = 0
         for xi, target in zip(X, y):
              output = self.predict(xi)
               if(target != output):
                   misclassified_data_count += 1
         total_data_count = len(X)
         self.score_ = (total_data_count - misclassified_data_count)/total_data_count
        if(switch == "train"):
             self.trainScore.append(self.score_)
        else:
              self.testScore.append(self.score_)
         #print(" self.score = ", self.score_)
        return self.score_

 #
# Load the data set
#
bc = datasets.load_breast_cancer()
X = bc.data
y = bc.target
```

```
#
# Info on the dataset
#
print(bc.DESCR)
print(list(bc.target_names)) # how many classes
print(len(bc['feature_names'])) # num features

# Create training and test split
#
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

#
# Instantiate CustomPerceptron
#
prcptrn = CustomPerceptron()

#
# Fit the model
#
prcptrn.fit(X_train, y_train)

#
# Score the model
#
print(" test error",prcptrn.score(X_test, y_test, "test"))
print(" train error", prcptrn.score(X_train, y_train,"train"))

#
# Visualise learning
#
plt.plot(range(1, len(prcptrn.testScore) + 1), prcptrn.testScore, marker='o',label="test")
plt.plot(range(1, len(prcptrn.trainScore) + 1), prcptrn.trainScore, marker='+', label="train")
plt.legend(loc="lower right")
plt.xlabel('Epochs')
plt.ylabel('Error')
plt.show()
```