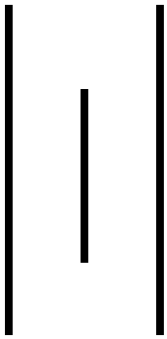
TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PURWANCHAL CAMPUS

DHARAN



ARTIFICIAL INTELLIGENCE

Lab Report V

Submitted by:

Gokarna Baskota
PUR075BEI013

Submitted To:

Department of Electronics and Computer Engineering

Lab 5: Implementation of Hebbian Learning and Perceptron

Hebbian Learning: Hebbian Learning is based upon Hebb's postulate of learning "If two interconnected neurons are both "on" at the same time, then the weight between them should be increased!" It is a single layer neural network, i.e. it has one input layer and one output layer. The input layer can have many units, say n. The output layer only has one unit.

Hebbian rule works by updating the weights between neurons in the neural network for each training sample.

Learning Rule

Step 0: Initialize all weights to 0
Step 1: Given a training input, s, with its target output, t, set the activation of the input units: xi = si
Step 2: Set the activation of the output unit to the target value: y = t
Step 3: Adjust the weights: wi (new) = wi (old) + xi y
Step 4: Adjust the bias (just like the weights): b(new) = b(old) + y

In this lab, we will implement Hebbian Learning on a network, which works just like an AND function.

X1	X2	b	y
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

Program:

```
import numpy as np
import matplotlib.pyplot as plt

x = [[1, 1], [1, -1], [-1, 1], [-1, -1]]

y = [1, -1, -1, -1]

b = 0
theta  = [0, 0]

for i in range(len(x)):
    for j in range(len(theta)):
        theta[j] = theta[j] + x[i][j]*y[i]
    b = b + y[i]

print("weights : {}".format(theta))
print("bias : {}".format(b))

def activation(val):
    if(val > 0):
        return 1
    else:
        return -1

example = [1, -1]
```

```
out = example[0]*theta[0] + example[1]*theta[1] + b

print("Output for x{} is : {}".format(example, activation(out)))
```

Output:

```
gokarna@gokarna-pc:~/Downloads/Study/AI_lab/lab-5-machine_learning$ python3 hebbian_learning.py
weights : [2, 2]
bias : -1
Output for x[-1, 1] is : -1
```

Perceptron: The perceptron was first introduced by American psychologist, Frank Rosenblatt in 1957 at Cornell Aeronautical Laboratory. Rosenblatt was heavily inspired by the biological neuron and its ability to learn. Rosenblatt's perceptron consists of one or more inputs, a processor, and only one output.

Perceptron is an algorithm for Supervised Learning of single layer binary linear classifier. Optimal weight coefficients are automatically learned. Weights are multiplied with the input features and decision is made if the neuron is fired or not.

Activation function applies a step rule to check if the output of the weighting function is greater than zero. Linear decision boundary is drawn enabling the distinction between the two linearly separable classes +1 and -1. If the sum of the input signals exceeds a certain threshold, it outputs a signal; otherwise, there is no output.

Algorithm

1. Set initial weights W1, W2,....Wn and bias to random numbers in the range [-0.5, 0.5]
2. Activate the perceptron by applying inputs and calculate the y_pred.
$$y\_pred = step(\Sigma X_i * W_i + b)$$
3. Update the weights of the perceptron as $W_i(p+1) = W_i(p) + alpha * X_i * e(p)$
4. Increase iteration p by one, go back to step 2 and repeat until convergence.

In this lab, we will implement a Perceptron that will work just like an OR function.

X1	X2	b	y
1	1	1	1
1	-1	1	1
-1	1	1	1
-1	-1	1	-1

Program:

```
from random import random
import numpy as np
import matplotlib.pyplot as plt

def step(val):
   if val > 0:
      return 1
   else:
      return -1

def pred(X, thetas):
   sum = X[0]*thetas[0][0] + X[1]*thetas[1][0] + X[2]*thetas[2][0]
   print("sum =", sum)
   return step(sum)

x = [[1, 1, 1], [1, 1, -1], [1, -1, 1], [1, -1, -1]]

y = [1, 1, 1, -1]
```

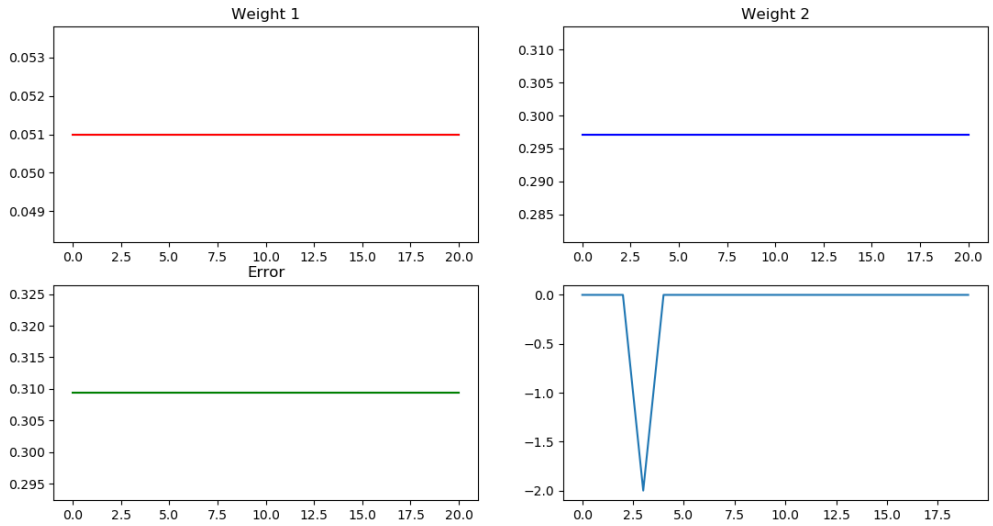
```
theta = np.random.uniform(size=(3, 1))*0.5

print("Initial theta: {}".format(theta))

alpha = 0.1
errors = []
theta0 = []
theta1 = []
theta2 = []

theta0.append(theta[0])
theta1.append(theta[1])
theta2.append(theta[2])

# print("Theta: {}, {}, {}".format(theta0, theta1, theta2))

for k in range(5):
    for i in range(len(x)):
        val = theta[0] + x[i][1]*theta[1] +x[i][2]*theta[2]
        y_hat = step(val)
        error = y[i] - y_hat
        errors.append(error)

        for j in range(3):
            theta[j] = theta[j] + alpha * x[i][j] * error

        theta0.append(theta[0])
        theta1.append(theta[1])
        theta2.append(theta[2])

def predict(X, theta):
    val = theta[0]
    for i in range(len(theta)-1):
        val = val + X[i]*theta[i+1]

    return step(val)

print("Final theta: {}".format(theta))

example = [-1, 1]
out = predict(example, theta)

print("\nPrediction")
print("Input: {}\nOutput: {}".format(example, out))

fig, ax = plt.subplots(2, 2, figsize=(13, 9))
ax[0,0].plot(theta0)
ax[0,1].plot(theta1)
ax[1,0].plot(theta2)
ax[1,1].plot(errors)
```

```
ax[0,0].title.set_text("Weight 1")
ax[0,1].title.set_text("Weight 2")
ax[1,0].title.set_text("Weight 3")
ax[1,0].title.set_text("Error")
ax[0,0].plot(theta0, "r-")
ax[0,1].plot(theta1, "b-")
ax[1,0].plot(theta2, "g-")
plt.show()
```

Output:

```
gokarna@gokarna-pc:~/Downloads/Study/AI_lab/lab-5-machine_learning$ python3 perceptron.py
Initial theta: [[0.25099877]
 [0.09712247]
 [0.10940438]]
Final theta: [[0.05099877]
 [0.29712247]
 [0.30940438]]

Prediction
Input: [-1, 1]
Output: 1
```

Graphs:



Iris Flower Classification Using Perceptron

Program:

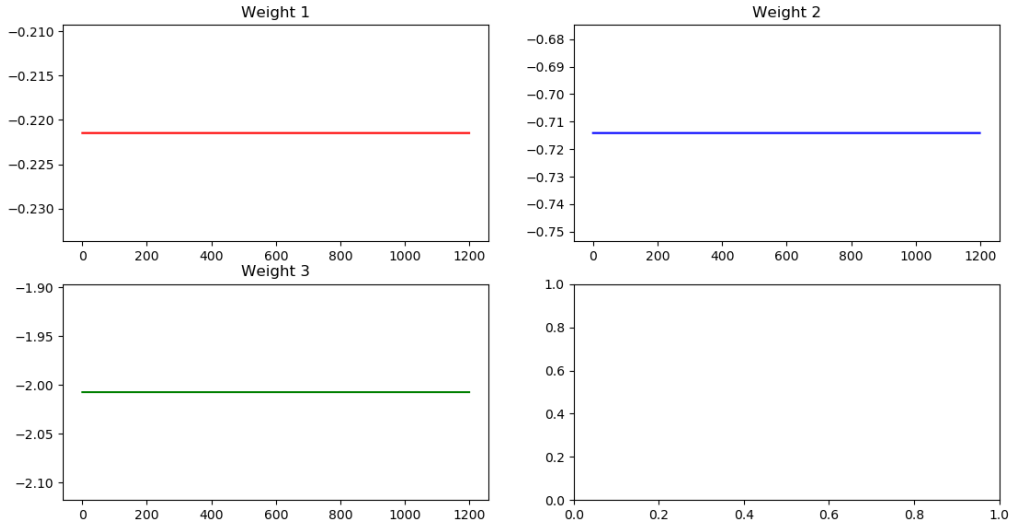
```
import sklearn.datasets as sk
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
# from random import shuffle
from sklearn.utils import shuffle

dataset = sk.load_iris()

dataset.data, dataset.target = shuffle(dataset.data, dataset.target)

index = 0
new_dataset = []
new_target = []

new_target_names = ['setosa', 'versicolor']

for data in dataset.data:

    if dataset.target[index] == 0 or dataset.target[index] == 1:
        new_dataset.append(data)
        new_target.append(dataset.target[index])
        # print(data)
    index += 1

new_dataset = np.array(new_dataset)
new_target = np.array(new_target)
new_target_names = ['setosa', 'versicolor']

# new_dataset.shape

X_train, X_test, y_train, y_test = train_test_split(new_dataset,
new_target, random_state=0, train_size = .6)
# X_train.shape

def sigmoid(z):
    # sigmoid activation
    return 1/(1 + np.exp(-z))

rand = np.random.RandomState(0)
theta = rand.uniform(size=(5, 1))*0.5

alpha = 0.1

theta0 = []
theta1 = []
theta2 = []
theta3 = []

for i in range(20):
    # print("Number of Iterations:", i)
    for j in range(X_train.shape[0]):

        val = 0

        length = theta.shape[0]-1
        for k in range(length):
            val = val + X_train[j][k]*theta[k+1]
```

```
        y_hat = sigmoid(val + theta[0])
        error = y_train[j] - y_hat

        for k in range(length):
            theta[k+1] = theta[k+1] + alpha*X_train[j][k]*error

        theta[0] = theta[0] + alpha*error

        theta0.append(theta[0])
        theta1.append(theta[1])
        theta2.append(theta[2])
        theta3.append(theta[3])


fig, ax = plt.subplots(2, 2, figsize=(13, 9))
ax[0,0].title.set_text("Weight 1")
ax[0,1].title.set_text("Weight 2")
ax[1,0].title.set_text("Weight 3")
ax[0,0].plot(theta0, "r-")
ax[0,1].plot(theta1, "b-")
ax[1,0].plot(theta2, "g-")
plt.show()

def predict(X, theta):
    val = theta[0]
    for i in range(len(theta)-1):
        val = val + X[i]*theta[i+1]

    return sigmoid(val)

index = 10
print("Target value from test set is
{}".format(new_target_names[y_test[index]]))

pred = predict(X_test[index],theta)

if pred > 0.5:
    pred = 1
else:
    pred = 0

print("Predicted target value is {}".format(new_target_names[pred]))
```

Output:

```
gokarna@gokarna-pc:~/Downloads/Study/AI_lab/lab-5-machine_learning$ python3 iris_classification.py
Target value from test set is versicolor
Predicted target value is versicolor
```

Graphs:


Weight 1
−0.210
−0.215
−0.220
−0.225
−0.230
0
200
400
600
800
1000
1200
Weight 2
−0.68
−0.69
−0.70
−0.71
−0.72
−0.73
−0.74
−0.75
0
200
400
600
800
1000
1200
Weight 3
−1.90
−1.95
−2.00
−2.05
−2.10
0
200
400
600
800
1000
1200
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0