Assignment #1 -- Perceptron

題目

1. Use line equation y=mx+*b* with particular parameters *m* and *b* to randomly generate 30 2D data samples. 15 samples in the right of the line are marked as positive samples and the others in the left are used as negative samples. No samples on the line.
2. Implement **Perceptron Learning Algorithm** with your own initial w_0. Discuss if your PLA halts or how many iterations it halts. Generate the data samples three times and calculate the average number of iterations when PLA halts.
3. In Problem 1, generating 1000 positive samples and 1000 negative samples. Implement Pocket Algorithm and compare the execution time to PLA on the same dataset.
4. In Problem 3, mislabel 50 positive and 50 negative samples by incorrect label. Report the accuracy of Pocket Algorithm by this setting and the setting in Problem 3.

Note:
• The assignment should be implemented by Python.
• You need to hand in the python code and the report.
• In your report, it should contain: (請以中文撰寫)
• **Execution description**: steps how to execute your codes.
• **Experimental results**: As specified in the assignment.
• **Conclusion**: The observation from your results.
• **Discussion**: The questions or the difficulties you met during the implementation.
• Assignment format
    • Zip all your files into a single one and upload it to the E-Course2 website.
• Please format the file name as: Student ID_proj1_verNo, ex：602410143_proj1_v1 • No copy! Late policy applies.

我的答案

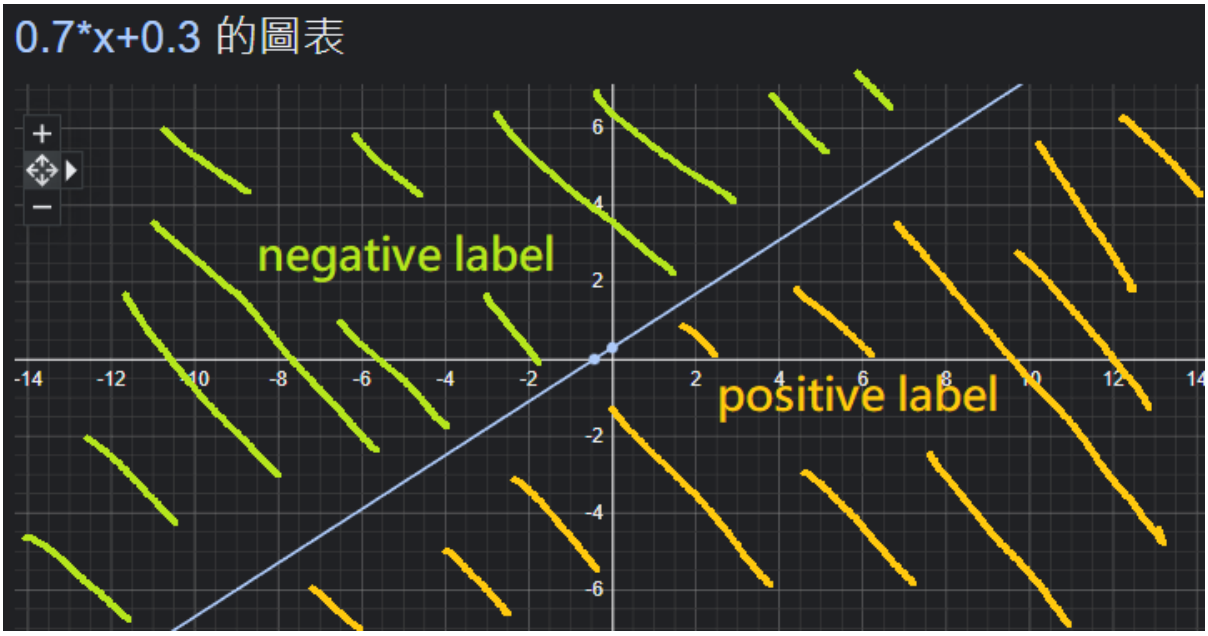
如何執行 (Execution Description)

1. 把 `perceptron_code.ipynb` 檔案以 GOOGLE COLAB 開啟。
2. 點選 **執行階段** > **全部執行** 。

實驗結果 (Experimental Results)

- 題目 1 實驗結果：
    - 我實作 `rand_sample_generation` 這個 function ，其有兩個參數 `pos_n` 和 `neg_n` ，分別代表 positive label 的資料數量以及 negative label 的資料數量。
    - `rand_sample_generation` 有定義題目所要求的一直線，如下：

$$y = 0.7x + 0.3$$



- `rand_sample_generation` 會在 `0.7 * x_tmp - y_tmp + 0.3 != 0` 的時候才產生樣本，也就是說不會產生落在線上的樣本。
- `rand_sample_generation` 會產生出指定數量的樣本並且以 `random.shuffle` 打亂順序。
- 使用以下程式碼便可以產生樣本 (樣本附帶 label) ，付上結果截圖：

```
# generate random sample
data = rand_sample_generation(15, 15)
print(data)
```

```
tensor([[-15.3536,   7.4616,  -1.0000],
        [ 18.4288,  31.9370,  -1.0000],
        [-41.9092, -26.4187,  -1.0000],
        [-15.4115, -46.9166,   1.0000],
        [-10.4020,   7.6098,  -1.0000],
        [-43.3205,   1.4967,  -1.0000],
        [-35.1359,  -2.1730,  -1.0000],
        [-20.7053, -31.7365,   1.0000],
        [  1.2750, -28.0854,   1.0000],
        [-39.2225, -32.2348,   1.0000],
        [-16.8428, -26.4752,   1.0000],
        [ 10.0873,  26.9734,  -1.0000],
        [  3.9953, -21.0184,   1.0000],
        [  7.2118, -31.5616,   1.0000],
        [-36.9033, -44.2527,   1.0000],
        [ 12.6171, -48.7071,   1.0000],
        [ 19.9786,  33.7099,  -1.0000],
        [-18.4480,  37.6435,  -1.0000],
        [-43.7406,  36.4861,  -1.0000],
        [ 36.7934,  -7.0908,   1.0000],
        [-14.4023,  -7.6041,  -1.0000],
        [ 41.5265,   8.0071,   1.0000],
        [ 23.1404,  45.1217,  -1.0000],
        [ 40.9922,  33.7690,  -1.0000],
        [ 17.7193,  27.8783,  -1.0000],
        [ 46.1234, -45.9491,   1.0000],
        [ 32.4717, -16.4228,   1.0000],
        [ 41.6170,   6.3581,   1.0000],
        [-13.4445,  17.1738,  -1.0000],
        [ 16.7616,   1.4337,   1.0000]])
```

- 題目 2 實驗結果：
  - 初始化 weight ：

    ```
    tensor([0.8823, 0.9150, 0.3829])
    ```

  - 先執行一次 PLA 的結果，發現 PLA 會在執行 2 次 iteration 之後停止：

    ```
    fin_weight: tensor([ 49.8511, -64.4137,  -1.6171])
    iter_count: 2
    ```

  - 執行 PLA 3 次並記錄 iteration count 結果如下：

```
i: 0 , iter_count: 25
i: 1 , iter_count: 1
i: 2 , iter_count: 6
iter_avg: 10.666666666666666
```

- 題目 3 實驗結果：
  - 先產生出 2000 筆資料：

```
# generate random sample
data = rand_sample_generation(1000, 1000)
print(data)
```

  - 比較 `PLA` 與 `pocket_alg` 的結果：

```
--> PLA: 1540.9111080000005ms
--> POCKET: 16692.379519ms
pocket accuracy: 1.0
```

- 題目 4 實驗結果：
  - 產生錯誤標記數據與正常標記數據：

```
data = rand_sample_generation(1000, 1000)
mis_data = rand_sample_generation_mis50(1000, 1000)
```

  - 實驗結果如下：

```
fin_weight: tensor([ 658.8668, -943.4258,  289.5936]) , err_rate: 0.0
pocket accuracy: 1.0
fin_weight_mis: tensor([ 71.0908, -29.7757,  13.7411]) , err_rate_mis: 0.1525
pocket accuracy (mis) : 0.8475
```

結果觀察 (Conclusion)

- 題目 1 結果觀察：
  - 30 筆正確標記的隨機數據。
- 題目 2 結果觀察：
  - `PLA` 會停止。
  - 平均下來會經過大約 `10.67` 次之後停止。
- 題目 3 結果觀察：

    - 發現 pocket_alg 的 max iteration 上限值夠大時，其會花比較多時間執行。

- 題目 4 結果觀察：
    - 如果把數據混入 mislabeled data ，那 pocket_alg 得出來的準確率會降低。

相關討論 (Disscussion)

- PLA 何時需要更新？
    - 相同 label 但是不同邊或者是不同 label 同一邊時要更新，可以理解成用相乘為負值時需要更新。
    - 實作 need_update function ：

```
def need_update(v1, v2, label):
    res = label * torch.dot(v1, v2)
    if res < 0:
        return True
    else:
        return False
```

- pocket_alg 有時候會比 PLA 快？
    - 如果 pocket_alg 的 max iteration 值設得不夠大時有可能會發生，而且 iteration 太低可能會影響準確率，故我多次嘗試把 iteration 調整到 200 次，目前這個調整可以顯示出預期的結果。

程式碼 (Code)

> 也可以參考 perceptron_code.ipynb 檔案。

```
import torch
from torch import nn
## torch version
print(torch.__version__)
```

```
import random

# generate samples (a, b)
# 0 <= a <= 1, 0 <= b <= 1
# right ( positive ) --> 15
# left ( negative ) --> 15

torch.manual_seed(42)

def rand_sample_generation(pos_n, neg_n):

    # y = 0.7x + b
    # --> 0.7x - y + b = 0
```

```
    # --> 0.7x1 - x2 + b = 0
    m = 0.7
    b = 0.3

    pos = []
    neg = []
    while True:
        x_tmp = random.random() * 100 - 50
        y_tmp = random.random() * 100 - 50
        # +-1 for label
        if m * x_tmp - y_tmp + b > 0 and len(pos) < pos_n:
            pos.append([x_tmp, y_tmp, 1])
        elif m * x_tmp - y_tmp + b < 0 and len(neg) < neg_n:
            neg.append([x_tmp, y_tmp, -1])

        if len(pos) == pos_n and len(neg) == neg_n:
            break

    # print("pos :", len(pos), pos)
    # print("neg :", len(neg), neg)

    sample = pos + neg
    # print("sample :", len(sample), sample)

    # shuffle pos neg
    random.shuffle(sample)

    # return sample in tensor
    sample_tensor = torch.tensor(sample, dtype=torch.float32)
    return sample_tensor
```

```
def need_update(v1, v2, label):
    res = label * torch.dot(v1, v2)
    if res < 0:
        return True
    else:
        return False

def PLA(data, weight, max_iter=1000, lr=1):
    iter_count = 0
    for i in range(max_iter):
        update_count = 0
        for j in range(len(data)):
            data_vec = torch.cat((
                data[j][:2],
                torch.tensor([1], dtype=torch.float32)
            ))
            label = data[j][2:3]
            if need_update(data_vec, weight, label):
                weight = weight + lr * label * data_vec
                update_count = update_count + 1
```

```
        # bread when we don't neet to update
        if update_count == 0:
            break

        iter_count = iter_count + 1

    # return final weights and total iterations
    return weight, iter_count
```

```
# initialize weight
weight = torch.rand(3)
print(weight)
```

```
# generate random sample
data = rand_sample_generation(15, 15)
print(data)
```

```
fin_weight, iter_count = PLA(data, weight)
print("fin_weight:", fin_weight)
print("iter_count:", iter_count)
```

```
# three time to calculate average
iter_avg = 0
for i in range(3):
    data = rand_sample_generation(15, 15)
    fin_weight, iter_count = PLA(data, weight)
    print("i:", i, ", iter_count:", iter_count)
    iter_avg = iter_avg + iter_count
iter_avg = iter_avg / 3
print("iter_avg:", iter_avg)
```

```
# generate random sample
data = rand_sample_generation(1000, 1000)
print(data)
```

```
# pocket algorithm
# random weight
def pocket_alg(data, max_iter=1000, lr=1):
```

```
    weight = torch.rand(3)

    # declare final weight
    fin_weight = weight

    # declare err count min
    err_count_min = len(data) + 1

    for i in range(max_iter):
        err_count = 0
        for j in range(len(data)):
            data_vec = torch.cat((
                data[j][:2],
                torch.tensor([1], dtype=torch.float32)
            ))
            label = data[j][2:3]
            if need_update(data_vec, weight, label):
                weight = weight + lr * label * data_vec
                err_count = err_count + 1
        if err_count_min > err_count:
            err_count_min = err_count
            fin_weight = weight
        # print("fin_weight:", fin_weight, ", err_count_min:", err_count_min)

    err_rate = err_count / len(data)

    # return fin_weight, err_rate
    return fin_weight, err_rate
```

```
import time

# initialize weight
weight = torch.rand(3)
print(weight)
```

```
# try PLA
tic = time.process_time()
fin_weight, iter_count = PLA(data, weight)
toc = time.process_time()
print ("--> PLA: " + str(1000*(toc - tic)) + "ms")

# try POCKET
tic = time.process_time()
fin_weight, err_rate = pocket_alg(data, max_iter=200)
toc = time.process_time()
print ("--> POCKET: " + str(1000*(toc - tic)) + "ms")
print("pocket accuracy:", 1-err_rate)
```

```python
# mislabeled
def rand_sample_generation_mis50(pos_n, neg_n):

    # y = 0.7x + b
    # --> 0.7x - y + b = 0
    # --> 0.7x1 - x2 + b = 0
    m = 0.7
    b = 0.3

    pos = []
    neg = []
    while True:
        x_tmp = random.random() * 100 - 50
        y_tmp = random.random() * 100 - 50
        # +-1 for label
        if m * x_tmp - y_tmp + b > 0 and len(pos) < pos_n - 50:
            pos.append([x_tmp, y_tmp, 1])
        elif m * x_tmp - y_tmp + b < 0 and len(neg) < neg_n - 50:
            neg.append([x_tmp, y_tmp, -1])

        if len(pos) == pos_n - 50 and len(neg) == neg_n - 50:
            break

    # mislabeled data
    while True:
        x_tmp = random.random() * 100 - 50
        y_tmp = random.random() * 100 - 50
        # +-1 for label
        if m * x_tmp - y_tmp + b > 0 and len(pos) < pos_n:
            pos.append([x_tmp, y_tmp, -1])
        elif m * x_tmp - y_tmp + b < 0 and len(neg) < neg_n:
            neg.append([x_tmp, y_tmp, 1])

        if len(pos) == pos_n and len(neg) == neg_n:
            break

    # print("pos :", len(pos))
    # print("neg :", len(neg))

    sample = pos + neg
    # print("sample :", len(sample), sample)

    # shuffle pos neg
    random.shuffle(sample)

    # return sample in tensor
    sample_tensor = torch.tensor(sample, dtype=torch.float32)
    return sample_tensor
```

```python
data = rand_sample_generation(1000, 1000)
mis_data = rand_sample_generation_mis50(1000, 1000)
```

```
# compare with mislabeled sample

fin_weight, err_rate = pocket_alg(data, max_iter=200)
print("fin_weight:", fin_weight, ", err_rate:", err_rate)
print("pocket accuracy:", 1-err_rate)

fin_weight_mis, err_rate_mis = pocket_alg(mis_data, max_iter=200)
print("fin_weight_mis:", fin_weight_mis, ", err_rate_mis:", err_rate_mis)
print("pocket accuracy (mis) :", 1-err_rate_mis)
```