通訊所 碩一 107064522 李曼鈴

Computer Assignment 5-1

- 語言格式 : Python
- 程式流程
  1. 建立 Iris data
  2. 分別將 sigmoid function、weight initialization 以及 Forward/Backward Propagation 寫成 function
  3. 實作 neural network learning
     (1) 對每筆 example 做 forward propagation
     (2) 假設 output 的 y 與 target 不同，透過 backpropagation 去 update weight，並計算 MSE
     (3) 對下一組 example 重複(1)(2)，並累加 MSE
     (4) 完成所有 example(在 Iris 中為 150 筆)，進入下一個 epoch 重複上述步驟獲得新的累加 MSE
     (5) 比較兩次 epoch 的 MSE 是否符合 Termination Criterion，若不符合則重複新的 epoch 直到符合為止。

- 程式說明與截圖輸出
  1. Python 本身內建作業要求的 Iris dataset，透過 import scikit-learn 資料集來獲取。讀入 data 做為輸入的 x；target 做為分類 data 的依據。

```
# Processing Iris Dataset
iris = datasets.load_iris()
data = iris['data']
target = iris['target'] # target
```

  2. **sigmoid()** function:輸入 sigmoid function 的定義

```
# Sigmoid function
def sigmoid(z):
    f = 1/(1+exp(-z))
    return f
```

  3. **initial_weight()** function: Random 設定範圍[-0.1, 0.1]的 weight 初始值，參考下行 network 示意圖，總共有三組 weight。
     Input **x | w3 |** hidden layer **h2 | w2 |** hidden layer **h1 | w1 |** Output **y**

```
# Initial Weights
def initial_weight():
    w1 = np.zeros((4, 3))
    w2 = np.zeros((4, 4))
    w3 = np.zeros((4, 4))
    for i in range(4):
        for j in range(4):
            if j != 3:
                w1[i, j] = random.uniform(-0.1, 0.1)
            w2[i, j] = random.uniform(-0.1, 0.1)
            w3[i, j] = random.uniform(-0.1, 0.1)
    return w1, w2, w3
```

4. **fwd_propagation()** function: 從 input x 到 output y 順向的計算，其中用到第 2 點的 sigmoid function，最後 return hidden layers 以及 output y 的值。

```
# Forward propagation
def fwd_propagation(x, w1, w2, w3):
    for i in range(4):
        h2[i] = sigmoid((w3[:, i]*x).sum(axis = 0))
        h1[i] = sigmoid((w2[:, i]*h2).sum(axis = 0))
    for i in range(3):
        y[i] = sigmoid((w1[:, i]*h1).sum(axis = 0))
    return h1, h2, y
```

5. **backpropagation()** function: 從 output y 到 input x 方向的 error propagation，第一層透過 output y 及期望 target 的差距來計算 delta，再層層回算 delta 值，利用算出的 delta1, 2, 3 來 update weights，最後 return update 過後的 weights。

```
# Backpropagation
def backpropagation(x, y, t, h1, h2, w1, w2, w3, eta):
    delta1 = np.zeros((3))
    Delta1 = np.zeros((4))
    delta2 = np.zeros((4))
    Delta2 = np.zeros((4))
    delta3 = np.zeros((4))

    for i in range(3):
        delta1[i] = y[i]*(1-y[i])*(t[i]-y[i])
    for i in range(4):
        Delta1[i] = (delta1*w1[i, :]).sum(axis = 0)
        delta2[i] = h1[i]*(1-h1[i])*Delta1[i]
    for i in range(4):
        Delta2[i] = (delta2*w2[i, :]).sum(axis = 0)
        delta3[i] = h2[i]*(1-h2[i])*Delta2[i]

    # Update Weights
    for i in range(4):
        for j in range(3):
            w1[i, j] += eta*delta1[j]*h1[i]
    for i in range(4):
        for j in range(4):
            w2[i, j] += eta*delta2[j]*h2[i]
    for i in range(4):
        for j in range(4):
            w3[i, j] += eta*delta3[j]*x[i]
    return w1, w2, w3
```

6. 在跑主要流程之前先輸入一些初始數值:

```
# main
eta = 0.1
w1, w2, w3 = initial_weight()
MSE = 0
MSE_new = 0
epoch = -1
MSE_diff = 0
```

7. 利用 while 迴圈來跑流程，當 termination criterion 達成時跳出迴圈
   (1) 下面跑完一次 for loop 表示完成一組 example，總共會跑 **len(data)**，也就是 150 次，總共 150 次跑完即完成一次 epoch。
   (2) for loop 中會先讀取每組 example 的 target，設為一個三維的 array，方便與 output y 做比較，接著利用前面建立的 **fwd_propagation()** function 做 forward propagation。
   (3) 比較當 output y 與 target 不同時，進行 backpropagation，並回傳 update 後的 weight，接著計算 MSE 的值。
   (4) MSE 的值在每次做完一組 example 時做累加(不取平均)，而在完成 150 組 examples(一個 epoch)之後歸零重新計算。
   (5) 檢查 termination criterion，依題目要求比對前後兩組 epoch 的 MSE 值，低於 threshold 即終止，印出累計 epoch 數以及兩組 epoch 的 MSE 差值，然後跳出 while loop 結束計算。

```
while MSE == 0 or MSE_diff > 10**-4:
    epoch += 1
    MSE = MSE_new
    MSE_new = 0.0

    for l in range(len(data)):
        x = data[l]
        y = np.zeros((3))
        t = np.zeros((3))
        h1 = np.zeros((4))
        h2 = np.zeros((4))

        # Target
        if target[l] == 0:
            t[0] = 1
        elif target[l] == 1:
            t[1] = 1
        elif target[l] == 2:
            t[2] = 1

        # Forward Propagation
        h1, h2, y = fwd_propagation(x, w1, w2, w3)

        if list(y) != list(t):
            # Backrpopagation
            w1, w2, w3 = backpropagation(x, y, t, h1, h2, w1, w2, w3, eta)
            temp = 0.0
            for i in range(3):
                temp += (t[i]-y[i])**2
                MSE_new += temp/3

    # Termination Criterion
    if MSE == 0:
        MSE_diff = 0
    else:
        MSE_diff = (abs(MSE_new-MSE)/MSE)

    print('Epoch', epoch,'MSE\n\t', MSE_new/len(data))
```

8. **Output 結果**: 以題目要求設定 learning 為 0.1，總共跑了 9 個 epochs。

```
Epoch 0 MSE
   0.4508184982337941
Epoch 1 MSE
   0.4353437272246347
Epoch 2 MSE
   0.4377370343887559
Epoch 3 MSE
   0.4397969206128885
Epoch 4 MSE
   0.44086225173362315
Epoch 5 MSE
   0.44136924943171846
Epoch 6 MSE
   0.44160743378769873
Epoch 7 MSE
   0.4417214027471722
Epoch 8 MSE
   0.44177859943045167
Epoch 9 MSE
   0.4418097756247651


Number of epochs :  9
MSE Difference :  7.056972509216296e-05
```

Computer Assignment 5-2

- 語言格式: Python
- 程式流程

  大致與 5-1 相同，唯第三點「實作 neural network learning」的部分用 for loop 跑了五組不同的 learning rate。

- 程式說明與截圖輸出

  1. 在**# main** 的部分首先輸入指定 learning rate 的數值，在每次 for loop 設定 **eta** 為不同的 learning rate 值，記錄不同 learning rate 下，所需收斂的 epoch 數。

```
# main
learning_rate = np.array([0.1, 0.2, 0.3, 0.4, 0.5])

for LR in range(len(learning_rate)):
    eta = learning_rate[LR]
```

  2. **Output 結果**: 下頁為執行 4 次的結果，透過不同 learning rate 下執行的結果可以發現，在 learning rate 為 0.1 時，收斂次數較穩定，大約 9 個 epoch；然而其他組 learning rate 的執行結果較浮動，但 epoch 數並沒有因為 learning rate 的增加而嚴格增加，這是因為當 learning rate 較大時，有可能 update 時不小心跨過 minimum (或 local minimum)，也可能很幸運走幾步就剛好踏入 minimum，因此收斂速度較浮動；而使用小的 learning rate 時，比較不會有不小心跨過 minimum 的問題，因此收斂速度較穩定。

```
Learning Rate :  0.1
Number of epochs :  9
MSE Difference :  7.86513420023267e-05


Learning Rate :  0.2
Number of epochs :  47
MSE Difference :  9.377759445410522e-05


Learning Rate :  0.3
Number of epochs :  1135
MSE Difference :  9.64839656422026e-05


Learning Rate :  0.4
Number of epochs :  905
MSE Difference :  5.281766103187012e-05


Learning Rate :  0.5
Number of epochs :  1040
MSE Difference :  9.409008864215842e-05
```

```
Learning Rate :  0.1
Number of epochs :  9
MSE Difference :  8.299261873743144e-05


Learning Rate :  0.2
Number of epochs :  35
MSE Difference :  9.480644289862744e-05


Learning Rate :  0.3
Number of epochs :  1176
MSE Difference :  1.0248231221264043e-05


Learning Rate :  0.4
Number of epochs :  14
MSE Difference :  7.687188095382592e-05


Learning Rate :  0.5
Number of epochs :  130
MSE Difference :  9.784946091156863e-05
```

```
Learning Rate :  0.1
Number of epochs :  9
MSE Difference :  8.907187427401057e-05


Learning Rate :  0.2
Number of epochs :  16
MSE Difference :  9.80345994424854e-05


Learning Rate :  0.3
Number of epochs :  14
MSE Difference :  4.812940196810442e-05


Learning Rate :  0.4
Number of epochs :  120
MSE Difference :  4.215902819406782e-05


Learning Rate :  0.5
Number of epochs :  7
MSE Difference :  2.4379427255139713e-05
```

```
Learning Rate :  0.1
Number of epochs :  9
MSE Difference :  7.950803130551332e-05


Learning Rate :  0.2
Number of epochs :  51
MSE Difference :  6.064755898501432e-05


Learning Rate :  0.3
Number of epochs :  151
MSE Difference :  9.262180830401724e-05


Learning Rate :  0.4
Number of epochs :  222
MSE Difference :  1.7723370159995357e-05


Learning Rate :  0.5
Number of epochs :  40
MSE Difference :  1.6718019469139057e-05
```