# Machine Learning HW2

R04942070吳冠融

## 1. Logistic regression function.

1. **for** i **in** range(200000):
2. # forward
3. y\_pred\_rows = special.expit(np.dot(x\_data\_rows, w))  # shape = (x\_row\_num, 1)
4. y\_diff\_rows = (y\_data\_rows - y\_pred\_rows)  # shape = (x\_row\_num, 1)
5. w\_grad = -np.dot(x\_data\_rows\_trans, y\_diff\_rows)
7. # write model
8. **if** total\_epoch % 10000 == 0:
9. train\_error\_count = np.sum(np.abs(to\_bool(y\_pred\_rows) - y\_data\_rows))
10. train\_accuracy = float(x\_row\_num - train\_error\_count) / x\_row\_num
12. model\_file\_info = "epo" + str(total\_epoch) + "\_acc" + str(train\_accuracy)[:6]
13. **print**(model\_file\_info)
14. np.savez(MODEL\_FILE\_PATH, w=w, vw=vw, lr=lr, total\_epoch=total\_epoch, x\_train\_means=x\_train\_means, x\_train\_stds=x\_train\_stds)
16. # update weights. using momentum
17. vw = lr \* w\_grad + gamma \* vw
18. w = w - vw
20. total\_epoch += 1

Note: I have appended a ‘1’ in each data row, so I don’t need a constant b here.

## 2. Describe your another method, and which one is best.

### Method 2: Deep Neural Network

* 在Method 2裡，我實作了很多功能：
  + 資料預處理
    - Normalization
    - Selecting Data：
      * 我嘗試了兩種版本：一種使用data的全部column ，另一種是只用與output相關性最高的32個column（相關性判斷依據是 logistic regression 訓練出來的model的weight）。
  + 任意形狀的Neural Network
    - 初期我是把層數寫死。到後期會了方便測試，我用一個list（layers = [x\_col\_num, 30, 30, 30, 30, 1]）來設定NN的結構，要變動結構時只要改寫這個list再重新train即可。
  + Dropout
    - 在train時，Hidden layer每一層都會都會根據我設定的dropout rate而使部分node失去作用，以避免overfitting。
    - dropout rate我嘗試過0.5與0.75兩個數值，結果是：0.75的training accuracy上升得較快（收斂速度快），而當兩者的training accuracy一樣時，testing accuracy通常是0.5的略高一點。
    - 因為我有充足的時間train，所以後來我都採用 0.5。
    - 在上傳的code中，為了在10分鐘內達到最好的訓練效果，我將dropout rate設為0.75。

### Experements on DNN structures

**3 layers, dropout rate = 0.5, using 32 columns of data**

|  |  |  |  |
| --- | --- | --- | --- |
| Number of Neurons | 30 \* 3 | 40 \* 3 | 50 \* 3 |
| Train Accuracy | 0.9317 | 0.9653 | 0.9725 |
| Test Accuracy (Public) | 0.93 | 0.95 | 0.93333 |

可以看出在同為三層的情況下，neurons = 40\*3效果最好，到50\*3就overfit了。

**4 layers, dropout rate = 0.5, using all columns of data**

|  |  |  |
| --- | --- | --- |
| Number of Neurons | 30 \* 4 | 40 \* 4 |
| Train Accuracy | 0.9780 | 0.9810 |
| Test Accuracy  (Public) | 0.95333 | 0.93667 |

在同為四層的情況下，30 \* 4剛好能達到最高的accuracy，而40\*4就overfit了。

### Results

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Logistic Regression | Final Submission 1  DNN  using 32 columns  neurons = 40 \* 3 | Final Submission 2  DNN  using all columns  neurons = 30 \* 4 |
| Public  Score | 0.92667 | 0.95000 | 0.95333 |
| Private  Score | 0.91667 | 0.94000 | 0.92333 |
| Average  Score | 0.92167 | 0.945 | 0.93833 |

* 毫無疑問的，DNN的準確度比Logistic Regression高。
* 令我意外的是 40\*3的test accuracy比30\*4的高，可能是因為前者我使用的是32 column的資料，後者是用全部的資料，所以後者稍微overfit。（也可能只是test data太少筆的關係）
* Final Result:
  + Scored 0.94 and ranked 27/276 in private leaderboard.

