Machine Learning HW2 report

R04922108 林俊佑

1. Logistic Regression

i. 首先，先擷取 feature 和 label 將其放到 x, y 裡面另外 initial 一個 weight 矩陣，經過測試，我選擇使用前 56 個 feature

```
for row in train_data:
    row_l = re.sub('\n|\r', '', row).split(',')
    x.append(row_l[1 : 1 + 56])
    y.append(row_l[len(row_l) - 1])
x = np.asarray(x, dtype = np.float32)
bias = np.ones(shape = (x.shape[0], 1))
x = np.concatenate((bias, x), axis = 1)
w = np.random.normal(0., 0.01, (x.shape[1]))
y = np.asarray(y, dtype = np.float32)
```

ii. 這次我實作了兩種 optimization, adagrad 和 adam，最後選擇使用 adam，在 training 前先 init 一些 adam 會需要用到的 variable

```
t = 0
BETA1 = 0.9
BETA2 = 0.999
E = 10**(-8)
m = np.zeros_like(w)
v = np.zeros_like(w)
```

iii. 最後就是 training 了，每個 iteration 我都會計算一次目前預測 training data 的 accuracy, L 就是 cross entrophy，下面那個 y_[i] -= 0.00000000001 是為了怕 y_[i]出來是 1 造成 ln0 的情況

```
while True:
    t += 1
    y_ = 1. / (1. + np.exp(((-1.) * np.dot(x, w))))
    acc = 0.
    for i in range(y_.shape[0]):
        if (y_[i] > 0.5 and y[i] == 1) or (y_[i] <= 0.5 and y[i] == 0):
            acc += 1
        if y_[i] == 1:
            y_[i] -= 0.000000000001
    L = (1. / k) * np.sum((-y) * np.log(y_) - (1. - y) * np.log(1. - y_))
```

```
    m = BETA1 * m + (1 - BETA1) * gw
    v = BETA2 * v + (1 - BETA2) * gw**2
    m_hat = m / (1 - BETA1**t)
    v_hat = v / (1 - BETA2**t)
    w -= Learning_rate * m_hat / (v_hat**0.5 + E)
    #  gw_mean = np.mean(np.abs(gw))
    #  w -= (Learning_rate / np.sum(gw_his, axis = 0)**0.5) * gw
    if t == 4000:
        w_file_name = sys.argv[2]
        w_file = open(w_file_name, 'w')
        for i in range(len(w)):
            w_file.write(str(w[i]))
            w_file.write(',')
        w_file.close()
        break
    k += 1
```

最後，就是更新 weight，使用 adam 來做更新，初始的 learning rate 選擇 0.005，因為這個 model 比較簡單，大約只能 train 到 93%左右，有點 underfitting

```
iter 4000
Learning_rate 0.005
cross entrophy 0.0995099671057
mean of gradient 0.122116490102
acc 0.929517620595
```

2. Another Function(Neural Network)

i. 由於第一個方法 underfitting，為了減少 bias，我選用 neural network 來實作，首先擷取 feature label ,定義 network 的架構, initial weight,與 initial optimize(adam, adagrad)需要的變數, L 代表 layer 數, s 代表每一層的 node 數，在經過各種測試後,我最終選用一個 48 node hidden layer 的 model, DELTA 代表每一個 weight 的 gradient, m, v 都是 adam 需要的變數

```
train_data = open(sys.argv[1], 'r')
x_ = []
y = []
for row in train_data:
    row_l = re.sub('\n|\r', '', row).split(',')
    x_.append(row_l[1 : 1 + 56])
    y.append(row_l[len(row_l) - 1])

x_ = np.asarray(x_, dtype = np.float32)
y = np.asarray(y, dtype = np.float32)
y = y.reshape(y.shape[0], 1)
L = 3
s = [x_.shape[1], 48, 1]

w = [[] for i in range(L - 1)]
DELTA = [[] for i in range(L - 1)]
DELTA_his = [[] for i in range(L - 1)]
m = [[] for i in range(L - 1)]
v = [[] for i in range(L - 1)]
for l in range(len(w)):
    #  w[l] = np.random.normal(0., 0.01, (s[l] + 1, s[l + 1]))
    w[l] = np.random.randn(s[l] + 1, s[l + 1]) * (2. / (s[l] + 1))**0.5
for l in range(len(DELTA)):
    DELTA[l] = np.zeros_like(w[l])
    DELTA_his[l] = np.zeros_like(w[l])
    m[l] = np.zeros_like(w[l])
    v[l] = np.zeros_like(w[l])
a = [[] for i in range(L)]
a_ = [[] for i in range(L)]
delta = [[] for i in range(L)]
```

ii. 再來是 training，使用 backpropagation 來做 training，

i. Forward propagate，算出 output，這邊的 a 就是每一層的 output 通過 sigmoid 後的值 a_則是我為了 validate 哪個架構比較好而用來在每一個 iteration 計算 training 跟 validation set 的 accuracy 用的，loss 和 gradient 則用來累加各層的 loss 和 gradient，用來監控 training 有沒有好好執行

```
while(True):
    t += 1
    lr_t = lr * ((1. - BETA2**t) / (1. - BETA1**t))**0.5
    a[0] = x_[: k]
    biasa = np.ones(shape = (a[0].shape[0], 1))
    a[0] = np.concatenate((biasa, a[0]), axis = 1)
    a_[0] = x_
    biasa = np.ones(shape = (a_[0].shape[0], 1))
    a_[0] = np.concatenate((biasa, a_[0]), axis = 1)
    loss = 0.
    gradient = 0.
    for l in range(1, L):
        a[l] = sigmoid(np.dot(a[l - 1], w[l - 1]))
        a_[l] = sigmoid(np.dot(a_[l - 1], w[l - 1]))
        if l != L - 1:
            biasa = np.ones(shape = (a[l].shape[0], 1))
            a[l] = np.concatenate((biasa, a[l]), axis = 1)
            biasa = np.ones(shape = (a_[l].shape[0], 1))
            a_[l] = np.concatenate((biasa, a_[l]), axis = 1)
```

ii. Backward propagate，算出 gradient，首先先計算小 delta，代表的是每一個 node 的 error，接下來再計算大 DELTA，為最後 weight 的 gradient

```
delta[L - 1] = a[L - 1] - y[: k]
loss += np.mean(np.abs(delta[L - 1]))
for l in range(L - 2, 0, -1):
    if l != L - 2:
        delta[l] = np.dot(delta[l + 1][:, 1:], w[l].transpose()) * a[l] * (1 - a[l])
    else:
        delta[l] = np.dot(delta[l + 1], w[l].transpose()) * a[l] * (1 - a[l])
    loss += np.mean(np.abs(delta[l]))
for l in range(L - 1):
    if l != L - 2:
        DELTA[l] = (1. / k) * np.dot(a[l].transpose(), delta[l + 1][:, 1:])
        DELTA[l][1:] += (1. / k) * LAMBDA * w[l][1:]
    else:
        DELTA[l] = (1. / k) * np.dot(a[l].transpose(), delta[l + 1]) + LAMBDA * w[l]
```

iii. 一層一層更新 weight(使用 adam)

```
        m[l] = BETA1 * m[l] + (1 - BETA1) * DELTA[l]
        v[l] = BETA2 * v[l] + (1 - BETA2) * DELTA[l]**2
        m_hat = m[l] / (1 - BETA1**t)
        v_hat = v[l] / (1 - BETA2**t)
        w[l] -= lr_t * m_hat / (v_hat**0.5 + E)
```

iv. Performance，這個方法比第一個好, kaggle 95.333，第一個方法

kaggle 92.667

3. Other discussion

i. Initialization

在一開始時做的時候，其實我使用的是全部 weight initial 成一個 constant，但每次 training 到後面都會卡住 train 不下去，我想說怎麼可能這麼多 node 沒辦法 fit training set 呢，於是我上網查了很多資料發現，大家建議使用 normal distribution 來做 initial，我試過後發現，可以非常容易的 fit training set，接下來就只剩下選擇 model 以及解決 overfitting 的問題了

ii. Network Structure

在經過各種測試發現，其實只要一個 hidden layer 就可以輕易的 fit training set，最後選擇使用一個 hidden layer 48 個 node，在使用 early stopping 和 regularization 來降低 overfitting

iii. Overfitting

i. Regularization

在沒有使用 regularization 或 early stopping 的情況下，可以讓 training set 99.8% fit，加入 regularization 後就可以避免 overfitting，下圖是使用最後 1000 筆資料當作 validation set 並比較有無 regularization 的結果，可以發現有使用 regularization 的 model 雖然在 training set 上有較差的 accuracy 但在 validation set 上卻有較高的 accuracy

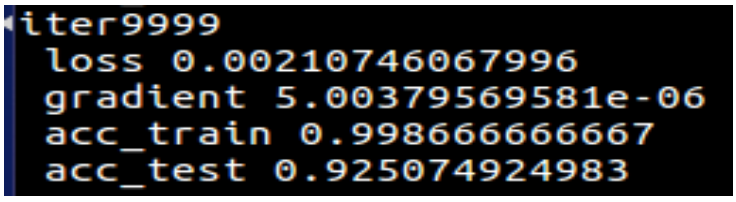


Figure 1 無 regularization

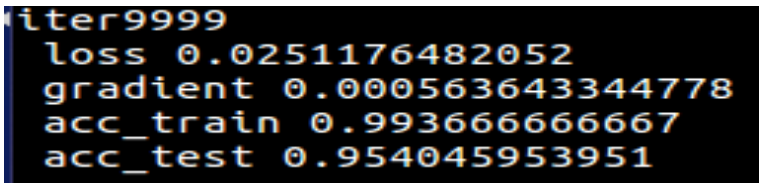


Figure 2 有 regularization(Lambda:0.002)

ii. Early stopping

在 training 的過程中我發現到早點結束 training 可以有效避免 overfitting，從下圖可以發現，其實在 iteration5000 的時候的 test accuracy 比 10000 的時候大很多，原因是在某一個點開始就會 overfitting 了，但其實大多時間 loss 函數是鋸齒狀的，所以很難決定到底要多早結束，我最後的 model 決定使用 training 2000 圈外加 regularization lambda 0.002 來避免 overfitting

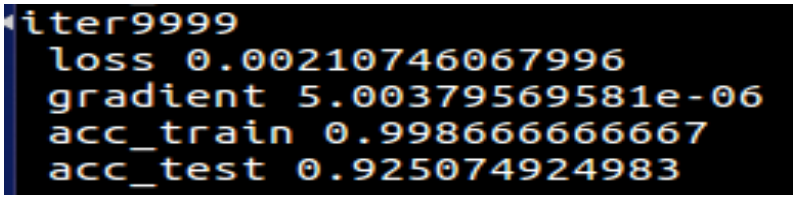


Figure 3 無 early stopping

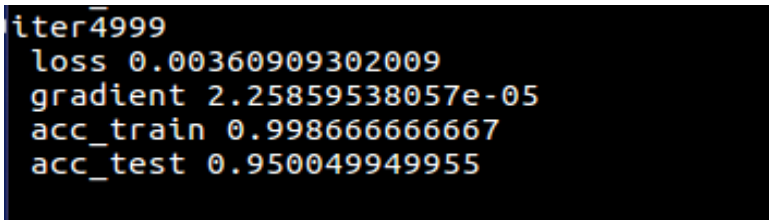
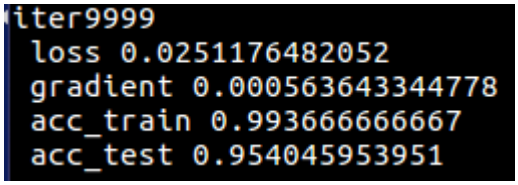


Figure 2 有 early stopping



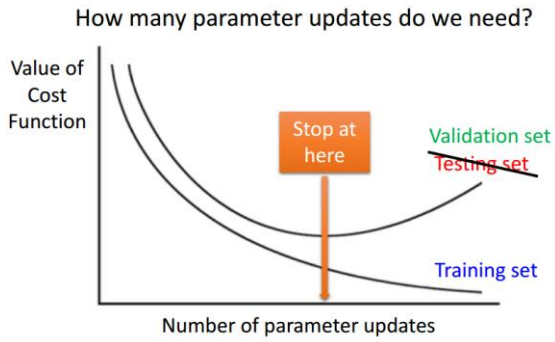


Figure 4 參考老師去年上課投影片

http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Deep%20More%20(v2).pdf

iii. Dropout

這次作業其實我原本有實作 Dropout 想說降低 overfitting，我使用的 dropout rate 是 0.5，但是因為在使用 dropout 後, training 變得極不穩定，最後就選擇用 early stopping + regularization 了