ML2016 HW1

B02902041 徐朝駿

1.(1%) Linear regression function by Gradient Descent

```
while not converge:
    cost = 0.0
    # Run SGD randomly
    for i in np.random.randint(m , size = m):
        # Hypothesis of answer
        hypothesis = np.dot(x[i].reshape((1, 162)), theta) + bias
        # Here is the derivation of loss function , the regularization will add in gradient step
        loss = hypothesis - y[i]
        # Sum up all the cost in an epoch
        cost += loss**2/m
        # Gradient with regularization lamda = 2
        gradient = x[i].reshape((162, 1))*loss + 4*theta
        # Update the theta and bias
        theta = theta - alpha*gradient
        bias  = bias - alpha*loss

    print("Iteration %d | Cost: %f" % (it, cost))
    if cost < min_cost:
        min_cost = cost
        best_theta = theta
    # Stop condition
    if cost <= 31:
        converge = True
    it += 1
    if it >= numberofiter:
        converge = True
```

x 為 example , y 為 target

2.(1%) Describe your method.

根據 test.csv 的檔案，我把連續九小時的所有 feature 壓成一維陣列。處理 feature 比較需要注意的是跨月的地方，不小心直接接起來會產生時間的不相連 data。之後進入 Training 的部份，我使用的是 SGD 配上 Regularization。然後再取 training data 是 randomly 抽樣，最後再把所有的 cost 加總得到這一個 epoch 的 cost。最後是終止條件，除了道了指定的 iteration 後會停止並採用最小 cost 的 theta 外，根據觀察 training 時的 cost 我也設了一個根據 cost 的終止條件。最後得到的 theta 帶入 test set 就可以得到預測的答案。

另外我想討論 randomly 抽取 example 的好壞。先說他的缺點，就是因為每次抽取道的資料是不同組的關係，所以 cost 始終不會 converge，需要特別給定一個觀察後的低點作為結束的訊號。優點則是隨機取樣可能可以抽取到較好的 example，所以有時候可能會得到好於乖乖整組做完時獲得的結果。

3. (1%) Discussion on regularization.

實驗中發現加上 regularization 後可以使 cost 下降的幅度趨緩，這在做 sgd 時有很大的幫助，如果是尚未加上 regularization 前在即將收斂時會有比較巨大幅度的震盪，使得較難找到最佳 theta。

4.(1%) Discussion on learning rate.

發現調整 learning rate 的重要性是這次作業最大的收穫。剛開始使用 0.001 不過 learning rate 會突然暴升導致 overflow。根據老師上課所說這應該是太大導致他近不去 local minimum。之後才逐漸調整道 1e-7 才可以進入 baseline。對於 learning rate 的問題還可以使用 adagrad，不過我在使用 adagrad 時出現了最後幾乎 learning rate 趨近為零時，卻還沒到達 minimum 的情況。我猜可能是我在每個 epoch 就增加了 5750 次，下次要使用的話可能一個 epoch 更新一次 adagrad 就足夠了。