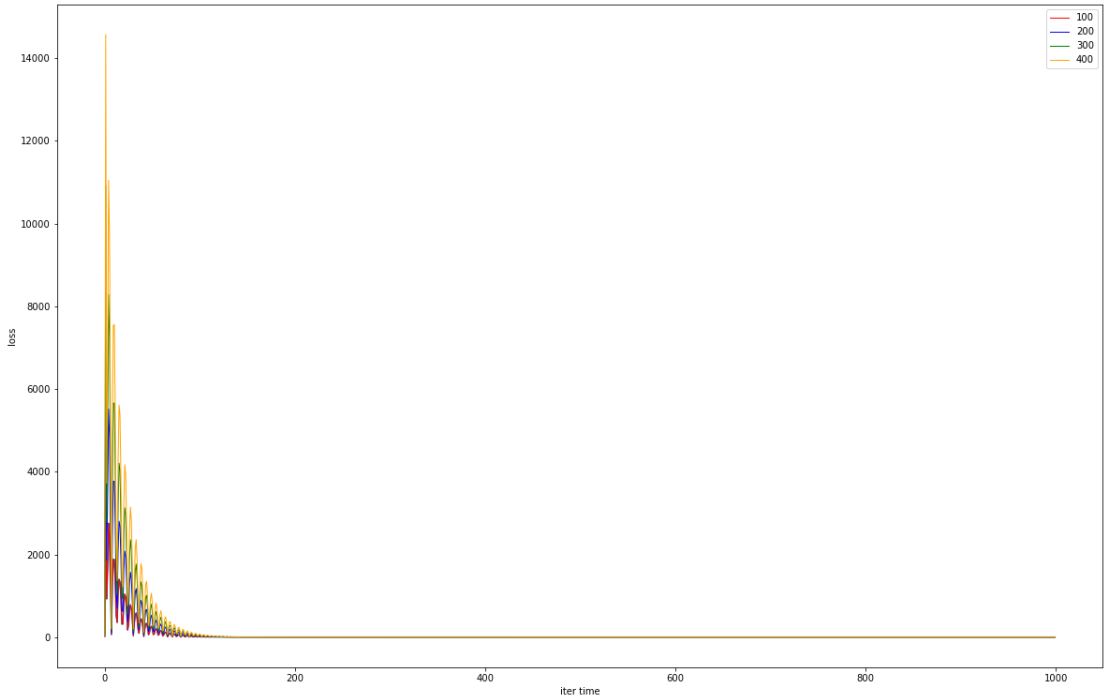
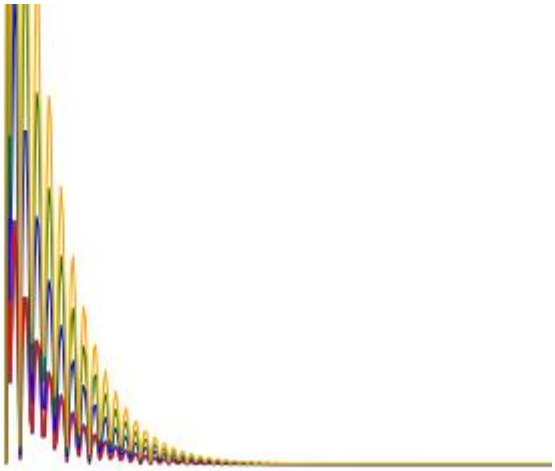
學號：B06705027 系級：資管三　　姓名：黃柏叡

1. (2%) 使用四種不同的 learning rate 進行 training (其他參數需一致)，作圖並討論其收斂過程（橫軸為 iteration 次數，縱軸為 loss 的大小，四種 learning rate 的收斂線請以不同顏色呈現在一張圖裡做比較）。



A:　圖中分別為初始 learning rate 在 100，200，300，400 下的作圖。首先從這張圖中我們可以看見 learning rate 的特色之一: learning rate 是呈鋸齒狀收斂的，原因是因為進行 gradient descent 時，我們雖然是朝著最低點的方向前進，但我們無法決定怎樣的步伐大小能夠準確到達最低點；而是只能透過過程中不段的下修以求能夠靠近它。

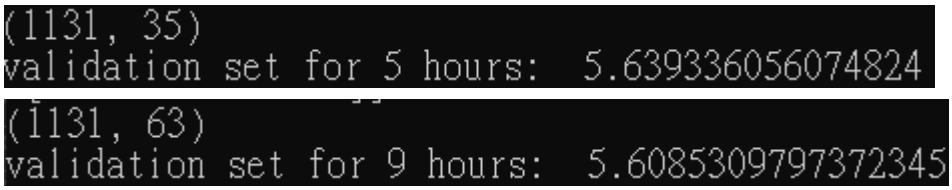
在圖中，我們能看到 4 個不同大小的 learning rate，其最後收斂所需的 iteration time 皆在 100 次左右:



然而放大一點來看的話，還是能夠看出些許差異：初始 learning rate 較大的情況其每次收斂的幅度都是相對高的，而原因則如同前面所述(步伐較大)。

2. (1%) 比較取前 5 hrs 和前 9 hrs 的資料（5*18 + 1 v.s 9*18 + 1）在 validation set 上預測的結果，並說明造成的可能原因（1. 因為 testing set 預測結果要上傳 Kaggle 後才能得知，所以在報告中並不要求同學們呈現 testing set 的結果，至於什麼是 validation set 請參考：https://youtu.be/D_S6y0Jm6dQ?t=1949 2. 9hr:取前 9 小時預測第 10 小時的 PM2.5；5hr:在前面的那些 features 中，以 5~9hr 預測第 10 小時的 PM2.5。這樣兩者在相同的 validation set 比例下，會有一樣筆數的資料）。

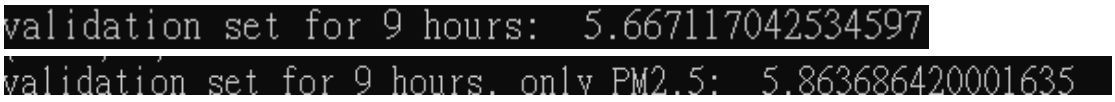
A: 我們透過將原本的 x set 切成 80%的 training set 跟 20%的 validation set 去進行比較，得到的結果如下圖(圖中數字為 loss)：



由圖可以得知：9 小時的預測相比 5 小時的預測結果更好。至於原因為何，我認為是在 training 以及 validation 中所取資料多寡的差異：對於 9 小時而言，其在預測每一筆 PM2.5 的資料都相對 5 小時擁有更多資訊，亦能在 test 的時候對於某些極端情況的資料加以稀釋差異，進而使其對結果影響較低。

3. (1%) 比較只取前 9 hrs 的 PM2.5 和取所有前 9 hrs 的 features（9*1 + 1 vs. 9*18 + 1）在 validation set 上預測的結果，並說明造成的可能原因。

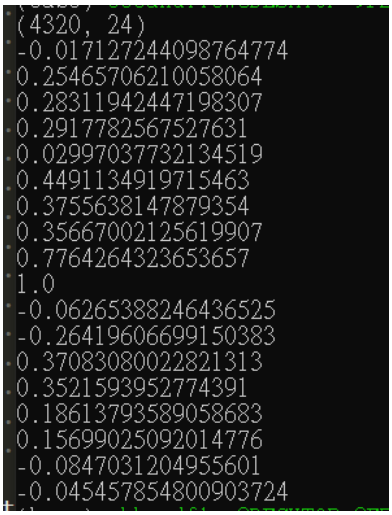
A: 我們透過將原本的 x set 切成 80%的 training set 跟 20%的 validation set 去進行比較，得到的結果如下圖(圖中數字為 loss)：



由圖可以得知，在擁有更多的 features 去 train model 的情況下，最後得到的 loss 會比單單只有 PM2.5 這項單一 feature 去測的結果還要好：猜想原因是 PM2.5 的這些懸浮微粒可能受到周圍環境的影響會滯留更久，或是更快消散；再者，PM2.5 的部分成因是由硫、氮的氧化物轉化而成，因此這些 feature 極可能和 PM2.5 具有某種程度上的相關性，也能夠更加準確預測出結果，降低 loss。

4. (2%) 請說明你超越 baseline 的 model(最後選擇在 Kaggle 上提交的) 是如何實作的（例如：怎麼進行 feature selection, 有沒有做 pre-processing、learning rate 的調整、advanced gradient descent 技術、不同的 model 等等）。

A: 首先是 feature selection，在這個部分，我透過簡單的相關係數公式，得之 18 項 feature 中，那些和 pm2.5 的相關性較高：



在反覆利用切割的 training set 進行模擬後，我認為和 PM2.5 |相關係數| > 0.3 為判斷是否具有影響力的指標，因此選出除了 PM2.5 之外的其他 6 項 feature，並將其餘 feature 刪除。

```
mean_x = np.mean(x, axis = 0) #18 * 9
print(mean_x)
std_x = np.std(x, axis = 0) #18 * 9
for i in range(len(x)): #12 * 471
    for j in range(len(x[0])): #18 * 9
        if std_x[j] != 0:
            x[i][j] = (x[i][j] - mean_x[j]) / std_x[j]
#x
```

在 pre-processing 中，我只有將 x 資料做簡單的 normalization，這麼做的目的是確保進行 gradient descent 時，能夠更有效率的走到最低點(loss 低點處)

```
93  #adam
94  beta1 = 0.9
95  beta2 = 0.999
96  learning_rate = 200
97  eps = 0.0000000001
98  iter_time = 1000
99  m_t = 0
100 v_t = 0
101 for t in range(iter_time):
102     gradient = 2*np.dot(x.transpose(), np.dot(x, w) - y) #dim*1
103     m_t = beta1 * m_t + (1-beta1) * gradient
104     v_t = beta2 * v_t + (1-beta2) * gradient * gradient
105     m_head = m_t / (1 - np.power(beta1, t+1))
106     v_head = v_t / (1 - np.power(beta2, t+1))
107     w = w - learning_rate* m_head /(np.sqrt(v_head)+eps)
108
109 print(w)
```

我使用了 adam 取代助教 code 中的 adagrad(如上圖)，理由是 adam 結合了 adagrad、RMSprop 及 momentum 的優點，考慮更多面向、表現進而更穩定。在調整 adam 的 learning rate 時，透過反覆不斷的 learning rate 在 validation 下的測試，我認為 200 是一個好的 learning rate，也透過做圖能看出其能成功完成收斂。

透過以上調整所產出的 submit.csv，在 kaggle 上的 loss 為 5.44224，成功通過 strong baseline。