Machine Learning Report

HW1 Linear Regression

姓名：許義宏 學號：B02901053

1. Linear regression function by Gradient Descent

由於實際的implementation牽涉到對於資料的格式，所以我將由資料開始說明起我的function implementation

資料處理：我將所有的feature都當成data，所以每一筆input資料都是一個18*9的矩陣（18個feature*9個小時），而我將一筆input data 壓成162*1的矩陣進行運算。因此weight相對應的就令成一個1*162的矩陣。

參考上課講義，將Gradient Descent的公式以向量表達則公式如下：

$$L(w,b) = \sum (y-(b+\overline{w}\cdot\bar{x})^2 ; \frac{\partial L}{\partial w} = \sum (2*(y-(b+\overline{w}\cdot\bar{x})*(-\bar{x}^T) ; \frac{\partial L}{\partial b} = \sum (-2*(y-(b+\overline{w}\cdot\bar{x})))$$

Funtion Implementation:

```
1  for i in range(iteration):
2      print "iteration  " + str(i)
3      loss = 0
4      diff_w = 0
5      diff_b = 0
6      for m in range(12):
7          bi = np.zeros((1,471))
8          bi.fill(bias)
9          lo = data[m][9,9:480].reshape(1,471) - bi - np.dot(weight,inputdata[m])
10         loss = loss+loss_unit(lo)
11         diff_w = diff_w+diff_Wunit(lo,inputdata[m])
12         diff_b = diff_b+diff_Bunit(lo)
```

```
1  def loss_unit(loss):
2      ans = np.square(loss)
3      return np.sum(ans)
4
5  def diff_Wunit(loss, data):
6      xT = np.transpose(data)
7      return (-2)*(np.dot(loss,xT))
8
9  def diff_Bunit(loss):
10     ans = (-2)*loss
11     return np.sum(ans)
```

對於每一個iteration來說，都要將每一個inputdata造成的值給累加起來，而爲了加速計算，我將每一個月的data轉換成矩陣形式（擴增成471維，爲何是471維見第二題）直接計算，再將12個月的值加起來就完成累加。

以每個月爲單位細部來說明，上圖第7&8行在進行的是data的轉換，而第九行lo的值相當於 $(y-(b+\overline{w}\cdot\bar{x})$，lo是一個1*471的矩陣，裏面的每個element都是 $(y-(b+\overline{w}\cdot\bar{x})$，因此做square sum即完成471筆data的loss加總(即function loss_unit 進行的工作);再來則是計算 $\frac{\partial L}{\partial w}$:function diff_Wunit做的事即是完成 $(2*(y-(b+\overline{w}\cdot\bar{x})*(-\bar{x}^T)$ 算式的計算(因爲lo = $(y-(b+\overline{w}\cdot\bar{x})$); 同理可知 $\frac{\partial L}{\partial b}$ 的計算方法。

再完成了值的計算後，每個iteration最重要的事是完成參數w與b的update，如右圖。

```
weight = weight - learning_r*diff_w
bias = bias - learning_r*diff_b
```

以上即是Linear regression function by Gradient Descent 的implementation。

2. Describe your method. 因為我們沒限制你該怎麼做,所以請詳述方法

1) 針對資料選取：我試過僅使用PM2.5去進行預測，預測出來的training loss也能達到大約5.75左右，然而若選用全部的Data去進行預測，效果還是有比較好，因此我選用全部的feature(18個）

2) 爲了讓資料能夠有效利用，每連續的10小時Data皆拿來利用，因此在一個月的連續20天內，共480個小時，共能夠選取471筆連續10小時的Data拿來做Training
3) 爲了加速learning所需花費的時間，使用Adagrad進行adaptive learning rate的調整，然而我發現到當iteration到比較後面時，用很小的constant learning rate會收斂到比較好的解。
4) 嘗試使用regularizaion看看performance有沒有比較好，發現沒有
5) 針對Data，我瞭解到在測量PM2.5時，量出來的值都是整數，因此我在進行預測時，把最後將linear regression預測的值進行進位（試過進位到整數位與小數點下1位），發現有比較好的performance。
6) initial point的嘗試：使用random的方法放置initial point，但測試後發現：initial point(w)太大的話，要花很久的時間去收斂，因此initial point random的值設在[0,0.01]

3. Discussion on regularization.

針對一次的linear regression model , iteration = 50000, constant learning rate = 10^{-10}

λ	0	1	10	100	1000	100000
training loss	6.026126	6.026139	6.026258	6.027451	6.039327	6.942990
testing loss	6.960102	6.960103	6.960167	6.960809	6.967222	7.634764

發現在使用regularization的情況下，training loss隨着lambda的權重越來越重而增加，符合預期；然而在testing loss上，並沒有跟預期一樣隨着lambda上升而下降，因次我額外再做了將regularization 應用在二次的model上的實驗

二次的linear regression model , iteration = 50000, adagrad learning rate = 0.001

λ	0	1	10	100
training loss	6.79170499	6.791706132	6.79171640	6.79181887
testing loss	7.98328985	7.98329046	7.98329591	7.983350389

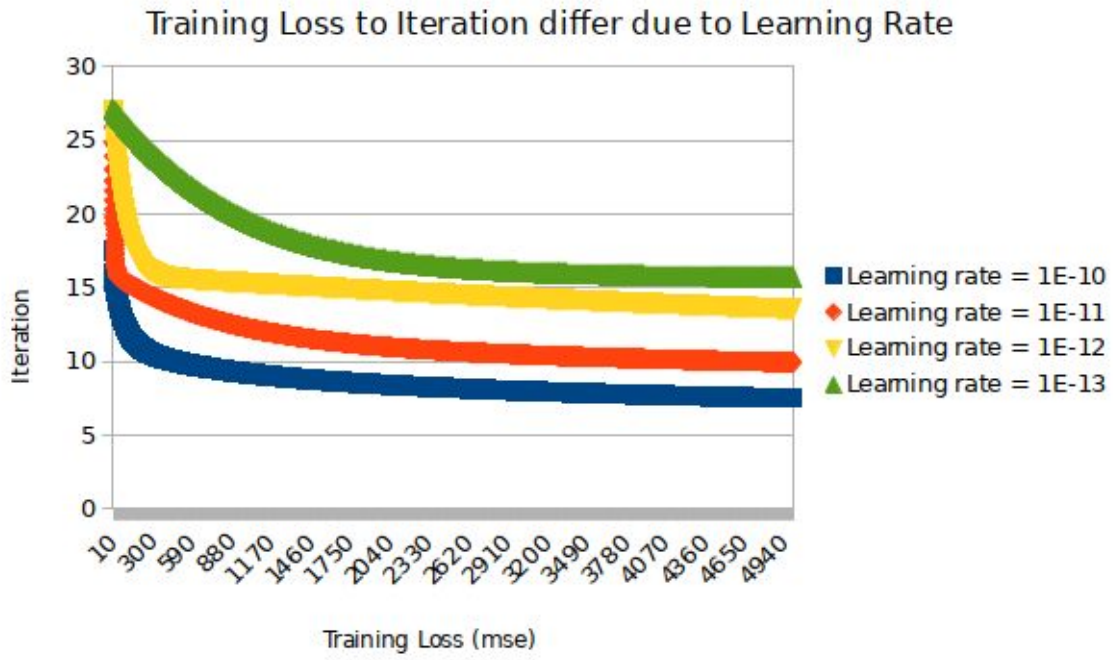
發現仍然有一樣的情形。甚至試三次的model (iteration = 50000 adagrad rate = 0.001)

λ	1	100
training loss	9.04150816938	9.04150823369
testing loss	14.4170376658	14.4170377128

儘管下降不多，但仍然沒有預期的好效果

4. Discussion on learning rate.

如圖，可知隨着learning rate的下降，要train到相同的training loss所需要的iteration數越多；但learning rate也不是越高越好，像是在這個情況下，將learning rate在調高一個數量級（即達到1E-9）則會在一百個iteration內即會因爲training loss 過大而超出電腦計算量。



5. TA depend on your other discussion and detail.

除了第3.第4題所調整的learning rate的改變與regularization的測試以外，我還有嘗試過用更高階的model，亦即若有feature x_1，我將其高次項$x_1{}^2$...等也加入weight的計算範圍，以二次爲例：inputdata就變成1*324的矩陣，實驗結果如下：

次數	Iteration	Adagrad rate	lambda	Training loss	Testing loss
1	50000	0.01	1	5.710029	6.52692
2	50000	0.01	1	6.79529017	7.9975384
3	50000	0.01	1	9.04150817	14.4170377

結果不甚理想，理論上即使testing loss沒有下降，Training Loss也應該要能下修，有懷疑過是不是因爲還沒有收斂的問題，但在以200000爲界，我仍然沒能用高次的model訓練出更好的結果，而加入regularization在高次的model裡，也沒能增進performance(詳見第三題）。

除了使用高次的model，我還有試過針對一階的模型採取不同的initial point的測試，預期結果是由於只有一階的模型，所以只會有一個谷（global minimum），所以應該在不同的initial point仍能收斂到同一個值，可是在實際上我們選用不同的initial point去對“testing data"進行測試，仍可以透過不同的initial point的選取，而有更好的結果。