Student ID: R05944004

Name: Cheng-Wei Wu

hw 1
Predict PM2.5
Report

- ## Linear Regression Function by Gradient Descent (Figure 7.)

Since I divided my program into several parts, I will introduce the overview of my program first. My program can be divided into 7 parts as follow (codes will be shown in figure 7):

1. Read Training Data
2. Setup Environment Parameters
3. Generate All Samples
4. Create Training & Validation Datasets + Calculate w & b + Validate the RMSE
5. Read Testing Data
6. Create Testing Dataset
7. Calculate for Predicted PM2.5 & Generate CSV output

- ### Read Training Data

At first, I created a list of numbers (size of the list equals to the #observation items of the data) which represent different types of observations from the data for the convenience of feature choosing in the future(e.g., "0" represents the observation of "AMB_TEMP").

Then I use the function "genfromtxt[1]" to load the train data "train.csv" into the data structure "ndarray[2]" with specific parameters setup (e.g., skip the header). Next, I allocated a 2-D list to store the original data from "train.csv"[3].

- ### Setup Environment Parameters

I set several parameters for sampling (e.g., length of hours for each sample, how many folds for cross-validation, best RMSE, etc.) data and cross-validation in this part.

- ### Generate All Samples

I sampled all the possible data and then shuffle them for the coming training dataset sampling and validation set sampling. (to shuffle the data and its corresponding ground-truth correctly, I shuffled them after zipping the data and ground-truth, and finally extracted them)

- ### Create Training & Validation Datasets + Calculate w & b + RMSE

Since I had already shuffled the data, I only need to loop the list from 0 to #fold to fetch the data for the validation (e.g., 12-fold → 1st loop: 0~470 for validation, 471~last for training, 2nd

loop: 471~941 for validation, others for training, and so on). To accelerate the computing process, after assigning the data to training set and validation set, I converted both of them into the type of "ndarray" mentioned in the previous paragraph. Also, I setup several parameters for the iterating process such like weights (also in the type of "ndarray", 1weight for 1 hour of a type, so each type may have maximum 9 weights in each sample data), b, alpha (learning rate), number of iteration, lambda, etc. Finally, in the computing process, I only need to do only 1 inner product of weights and training data and I could get the summation value[4] of all the products of each weight and its corresponding data. Then I could also use the ground-truth array to minus previous summation value and b to get the value (error)[5] for the computing of gradient descent. But because the differential of each weight need to multiply its corresponding data (x), I transposed the data matrix and performed the inner product to the previous matrix to get the final differential value[6] of each weight. After getting all the needed values for computing the new weight and b, I updated the weight and b, and go to next iteration again and again until iteration reach the maximum number of iteration. At last, I calculated the error rate of the validation set and chose the better one in each validation process.

- ## Read Testing Data

This part is almost the same with reading the training data, the only difference between it and reading training data is the input format. I only modified few lines to overcame this difference.

- ## Create Testing Dataset

Same with the previous one, it's almost the same with reading training part. Each test data's type has the same length of the training data.

- ## Calculate for Predicted PM2.5 & Generate CSV output

I performed the inner product of testing data and weight matrix and add b to the results to get the value of predicted PM2.5. Finally, I used "csv.writer[7]" to generate the output cvs file.

• Regularization

To my surprise, after I adding lambda parameter to regularize the computing formula, I found out that the result seems not better than the original one and get even worse …. But the effect is so small that it could be the same without regularization, I will show the experiment result in the end which compares the effect of regularization. After searching relative information[8] about regularization, I thought the reasons may be that:

1. I proposed my function set to be multivariate linear equation, which has lower probability overfitting the training set than quadratic functions
2. The increment of the error rate of the public set may indicate that my model won't overfit the training data or the public set (which means it will be more adaptable to other data)

• Learning Rate

Since I had written normal version, Adagrad[9] version and Adamgrad[10] version for the optimal gradient descent updating process, I set several initial learning rate for each of them and test

different values of them. In the normal version, I could only set it to be roughly about 0.0000000001, this is the max magnitude, and if I multiply 10 to it, the computing process will cause value overflow. Different from the normal version, Adagrad version can afford different learning since it will fix it in each iteration according to the summation of historical values of square of gradient. However, in the Adamgrad version, I could only set it to 0.001 according to the original paper which proposed it (and also the Google's tensorflow opensource file[11]).

• Discussion

In fact, I also implemented the mini-batch version for normal version, Adagrad version and Adamgrad version. But after doing the try-and-error test, I finally found out that the original normal version may get the best result if it could run enough iteration. In other words, my conclusions are:

1. Normal version may get best result if it could run enough iteration
2. Adagrad and Adamgrad can get the relative good results if the #iteration is limited
3. Regularization may get even worse result against the one without it

• Figures

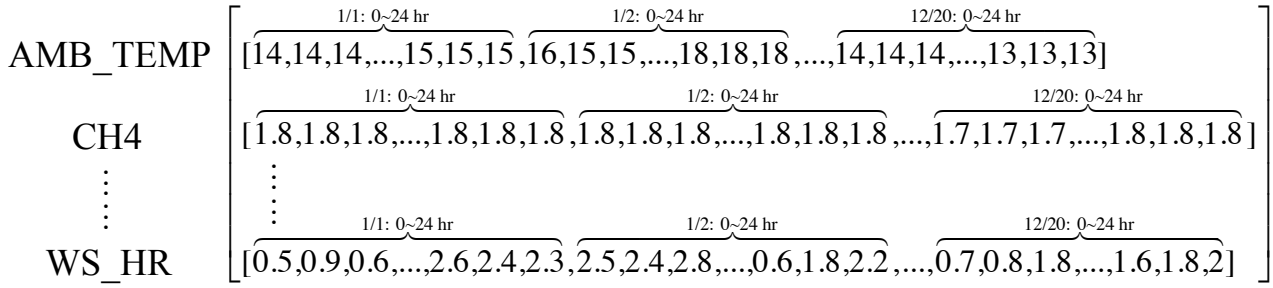

Figure 1.

Data-structure for Storing Training Data (2-D python built-in list)

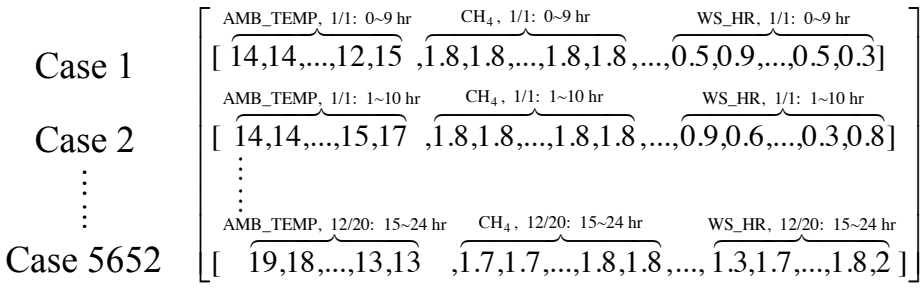

Figure 2.

Data Format for Computing Gradient (***X*** in iteration process, ndarray of numpy)

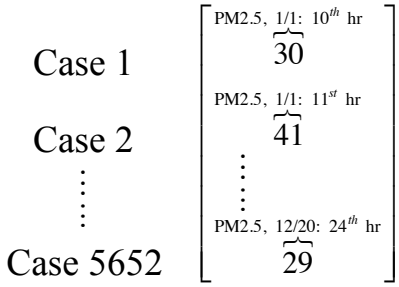

Figure 3.

Ground-truth format for Computing Gradient (***Y*** in iteration process, ndarray of numpy)

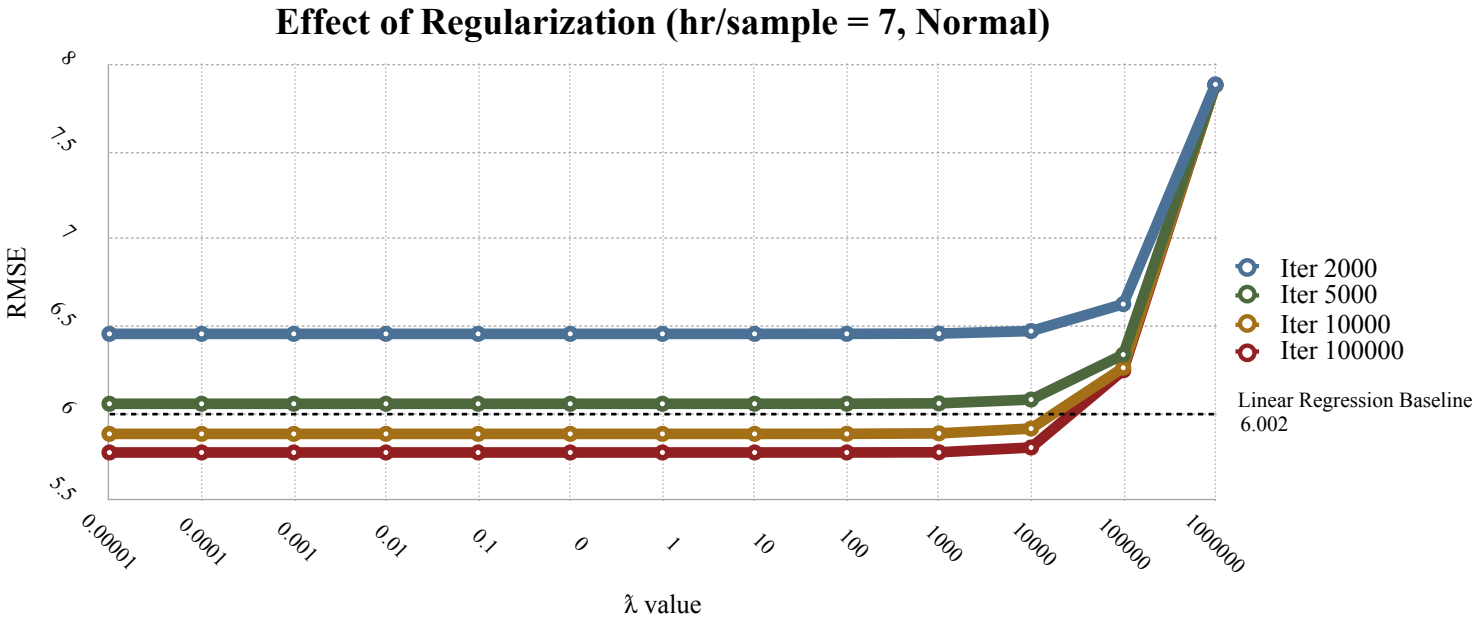

Figure 4.

Experiment results of the effect of regularization (Length of hours/sample = 7, Normal)

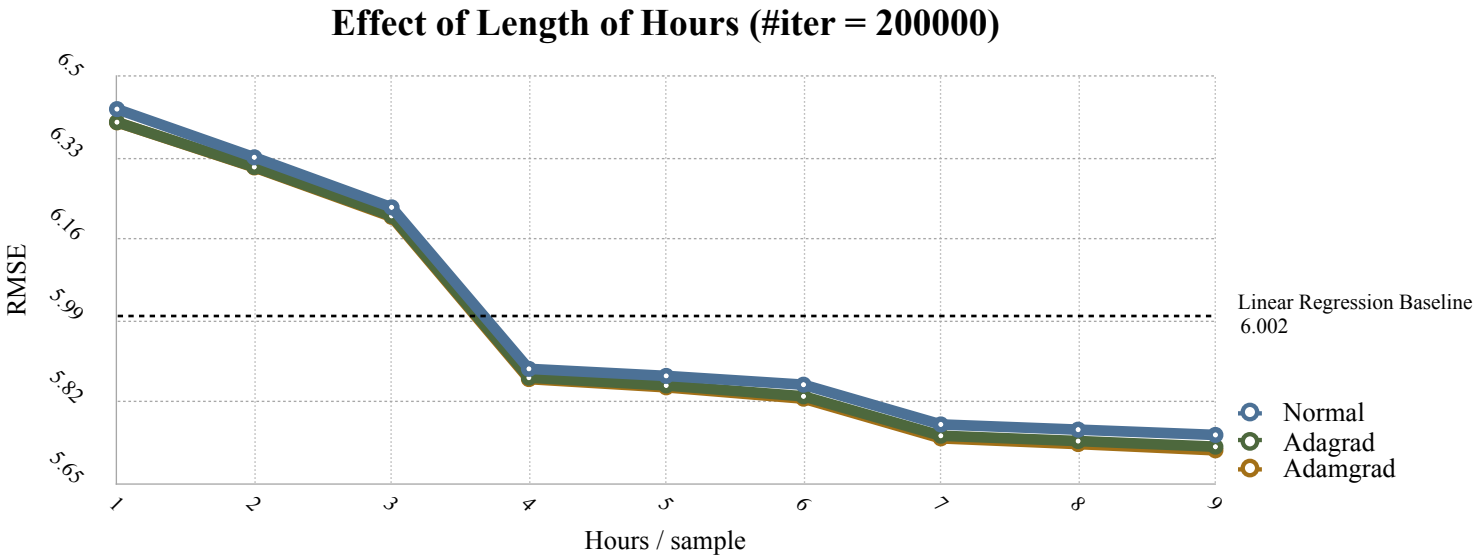

Figure 5.

Experiment results of the effect of length of hours (iteration = 200000)

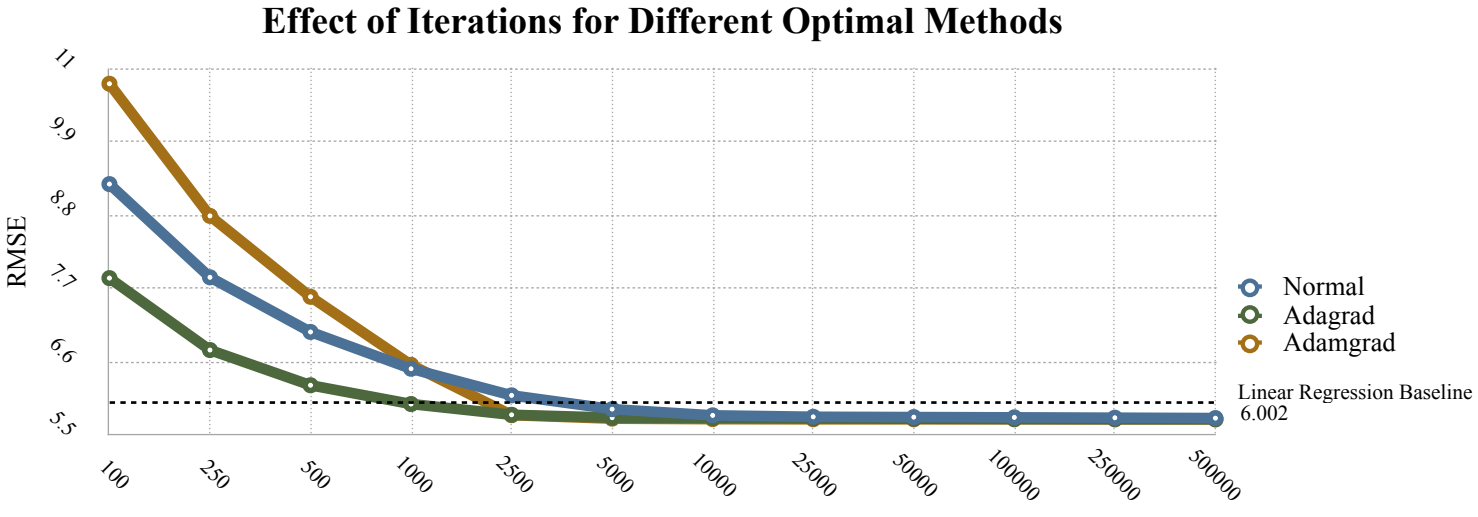

Figure 6.

Experiment results of the effect of iterations for different optimal methods (hr/sample = 7)

- Codes

```
83      for i in range(epoch): # iteration (default = 50000)
84          t += 1 # time step for admagrad
85          WX_TRAIN = np.dot(X_TRAIN, W) # inner product of weight & data
86          ERR = Y_TRAIN - (b + WX_TRAIN) # error of predicted result (y-(b+wx))
87          X_TRAIN_T = X_TRAIN.T # transpose data for next inner product
88          DW = -2 * np.dot(X_TRAIN_T, ERR) # multiply -X to error formula
89          DB = -2 * np.sum(ERR) # sum the error
90
91          # Compute Loss & Print
92          J = np.sum(ERR ** 2)
93          # print "Epoch %s | Loss: %.7f" % (i, J)
94
95          # Regularization
96  #         DW += Lambda * 2 * W
97
98          # Normal
99  #         W = W - nor_alpha * DW # / X_TRAIN.shape[0]
100 #         b = b - nor_alpha * DB # / X_TRAIN.shape[0]
101
102         # Adagrad
103 #         SUM_SQDW += np.square(DW)
104 #         SUM_SQDB += np.square(DB)
105 #         W = W - ada_alpha/np.sqrt(SUM_SQDW) * DW # / X_TRAIN.shape[0]
106 #         b = b - ada_alpha/np.sqrt(SUM_SQDB) * DB # / X_TRAIN.shape[0]
107
108         # Adamgrad
109         Wmt = beta1 * Wmt + (1-beta1) * DW
110         Wvt = beta2 * Wvt + (1-beta2) * np.square(DW)
111         Wmthat = Wmt / (1-np.power(beta1, t))
112         Wvthat = Wvt / (1-np.power(beta2, t))
113         Bmt = beta1 * Bmt + (1-beta1) * DB
114         Bvt = beta2 * Bvt + (1-beta2) * np.square(DB)
115         Bmthat = Bmt / (1-np.power(beta1, t))
116         Bvthat = Bvt / (1-np.power(beta2, t))
117         W = W - (adam_alpha*Wmthat) / (np.sqrt(Wvthat) + eps)
118         b = b - (adam_alpha*Bmthat) / (np.sqrt(Bvthat) + eps)
```

Figure 7.

Codes for iteration process (cost computing→line-92, updating parameters→line-99~118)

---

[1] Numpy.genfromtxt: http://docs.scipy.org/doc/numpy/reference/generated/numpy.genfromtxt.html

[2] Numpy.ndarray: http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html

[3] 2-D List for storing original data: Figure 1.

[4] Summation value: $\sum_{k}^{K} w_k \cdot x_k^n$ (k = number of weights)

[5] Ground-truth minus the summation value: $\sum_{n=1}^{N} 2\left(\hat{y}^n - \left(b + w_k \cdot x_k^n\right)\right)$

[6] Differential value of each weight: $\frac{\partial L}{\partial w_k} = \sum_{n=1}^{N} 2\left(\hat{y}^n - \left(b + w_k \cdot x_k^n\right)\right)\left(-x_k^n\right)$

[7] CSV file reading and writing: https://docs.python.org/2/library/csv.html

[8] Regularization: http://cpmarkchang.logdown.com/posts/193261-machine-learning-overfitting-and-regularization

[9] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 2121–2159, http://jmlr.org/papers/v12/duchi11a.html

[10] Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13, https://arxiv.org/pdf/1412.6980.pdf

[11] adam.py: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/training/adam.py