Chenyun Yu

Prof. Fern

CS-575 MACHINE LEARNING

Implementation 1 Reporter

Part 0 (20 pts) : Understanding your data + preprocessing.Perform the following steps:

a) **Remove the ID feature. Why do you think it is a bad idea to use this feature in learning?**

   Based on the commonsense, I don't think ID is relevant to the final price.

b) **Split the date feature into three separate numerical features: month, day, and year. Can you think of better ways of using this date feature?**

   Splitting date features concatenate to the end of the original data, showing in the following picture.

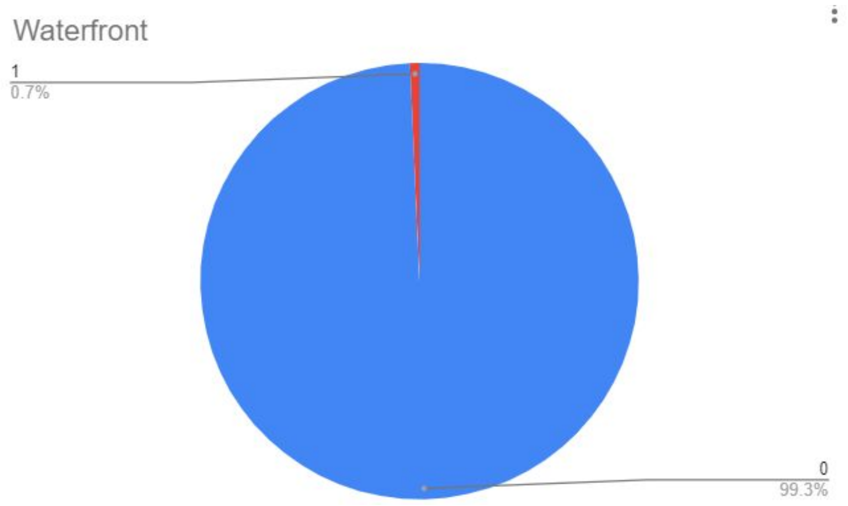
   Splitting the date feature will benefit the training process, using a numerical date feature makes this training have more realistic results. Sometimes the price of houses will periodically change, using the numerical feature can show the price changing in different months and days.

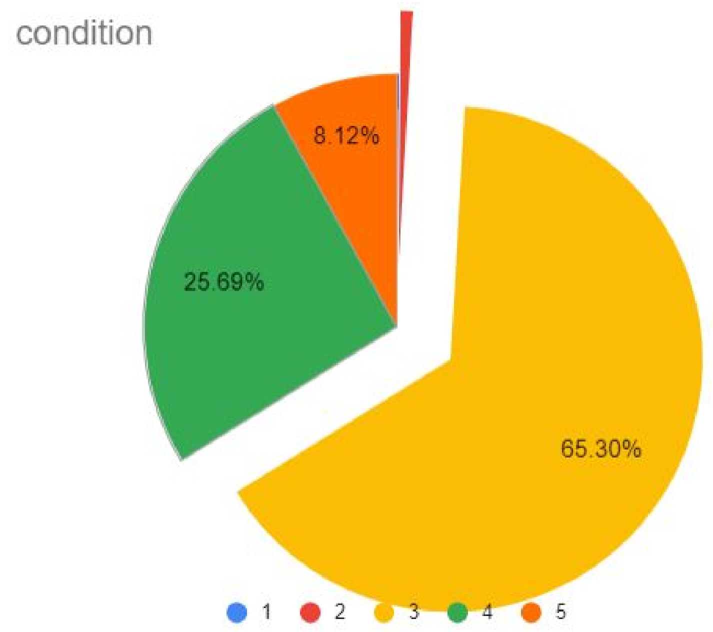
c) **Build a table that reports the statistics for each feature. For numerical features, please report the mean,the standard deviation, and the range. For categorical features such as waterfront, grade, condition(the later two are ordinal), please report the percentage of examples for each category.**

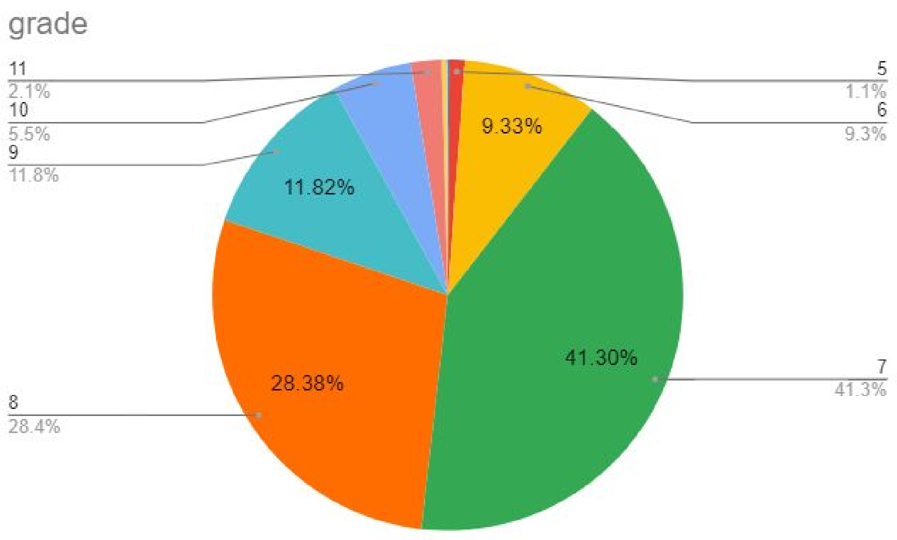
   Statistical data shows in the following table., and when you run part0.py, you will find a file named *part0_numerical_statistic.csv*, you also can find the data. Based on my understanding, I believe that the zip code, is not numerical features, because they are too discrete and it is very difficult to say the standard deviation is really meaningful for zip code.

feature	mean	std	max	min
bedrooms	3.3752	0.9431993215	1	33
bathrooms	2.118875	0.7650898538	0.5	7.75
sqft_living	2080.2232	911.2887904	370	9890
sqft_lot	15089.2014	41201.83467	572	1651359
floors	1.5037	0.5426198577	1	3.5
view	0.2294	0.7558939344	0	4
sqft_above	1793.0993	830.8238901	370	8860
sqft_basement	287.1239	434.9835127	0	2720
yr_built	1971.1249	29.47911973	1900	2015
yr_renovated	81.2267	394.3600846	0	2015
zipcode	98078.2931	53.51571538	98001	98199
lat	47.55981419	0.1386436568	47.1559	47.7776
long	-122.2132865	0.1413976146	-122.514	-121.319
sqft_living15	1994.3261	691.8657051	460	6110
sqft_lot15	12746.3234	28239.83095	660	871200

The percentages of categorical features show in the following charts, and data can be found in *part0_categorical _percentage.csv* after tun part0.py.







d) Based on the statistics and your understanding of these features/housing prices, which set of features do you expect to be useful for this task? Why?

Based on my commonsense, I don't think the latitude and longitude play an important role in the final price, because this data set comes from the same city/town on my observation.

Meanwhile, I don't think zip code is a very important feature for this training, actually, zip code can represent the quality of community surrendering house, but there is no such relationship that, for example, a large zip code number means a better community and environment.

I think the grade and living space and condition are more useful features.

e) **Normalize all numerical features (excluding the housing prices y) to the range 0 and 1 using the training data.**

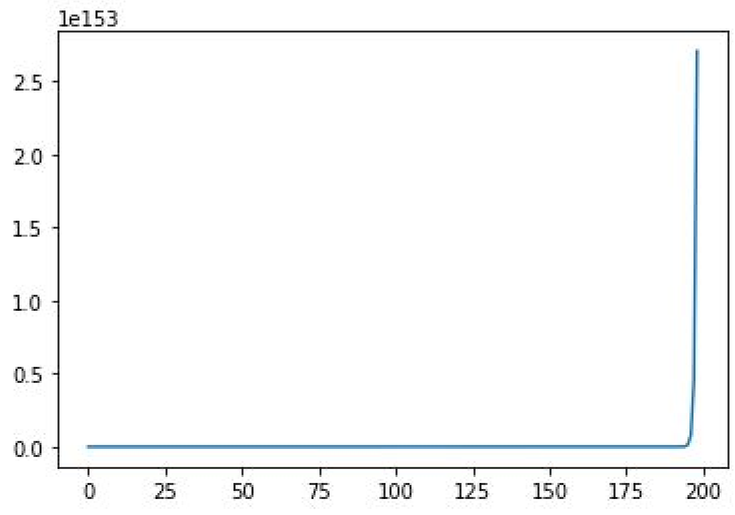
Normalized data will be stored in the *part0_nomalizaed_data.csv* file after the first run part0.py

Part 1 (40 pts). Complete the implementation and explore different learning rates for batch gradient descent. For this part, you will work with the preprocessed and normalized data, and consider at least the following values for the learning rate: 10^0, 10^−1, 10^−2, 10^−3, 10^−4, 10^−5, 10^−6, 10^−7.

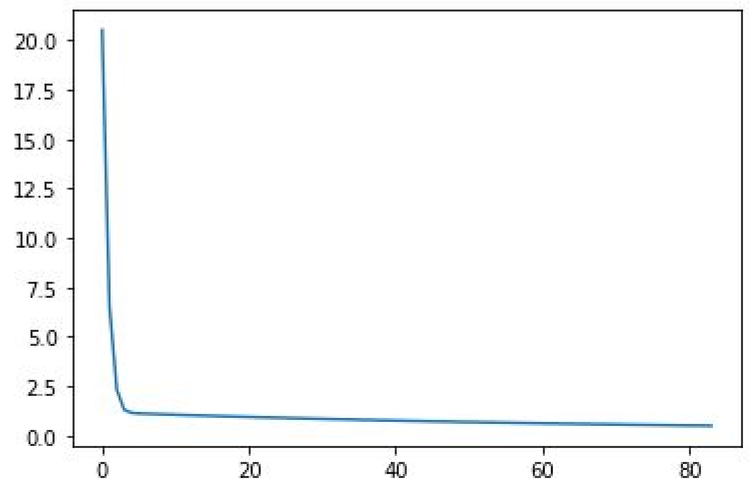
a. Which learning rate or learning rates did you observe to be good for this particular dataset? What learning rates (if any) make gradient descent diverge? Report your observations together with some example curves showing the training MSE as a function of training iterations and its convergence or non-convergence behaviors.

Based on my observations, the convergence starts when learning small or equal to 10^−1. Meanwhile, I think a reasonable learning rate should be pow(10, -1), because the smaller learning will dramatically increase the burden of calculation.

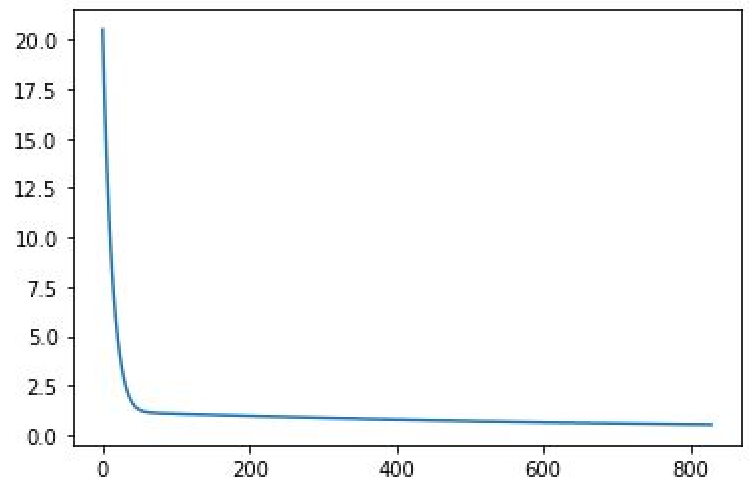
Learning rate = pow(10,0)



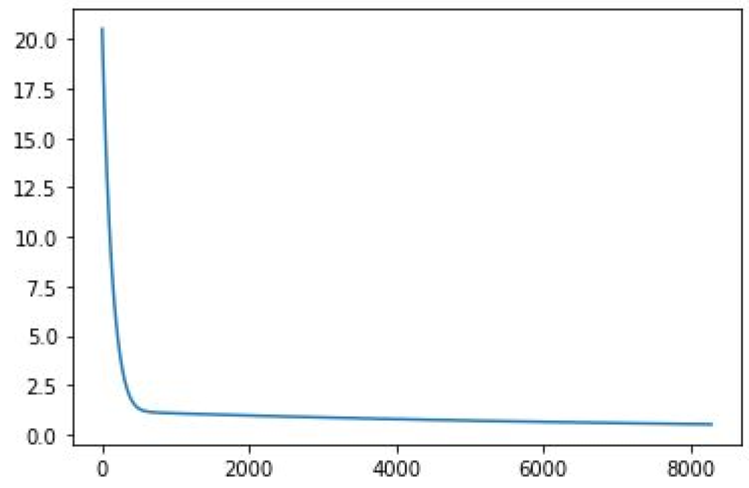
Learning rate = pow(10,-1):



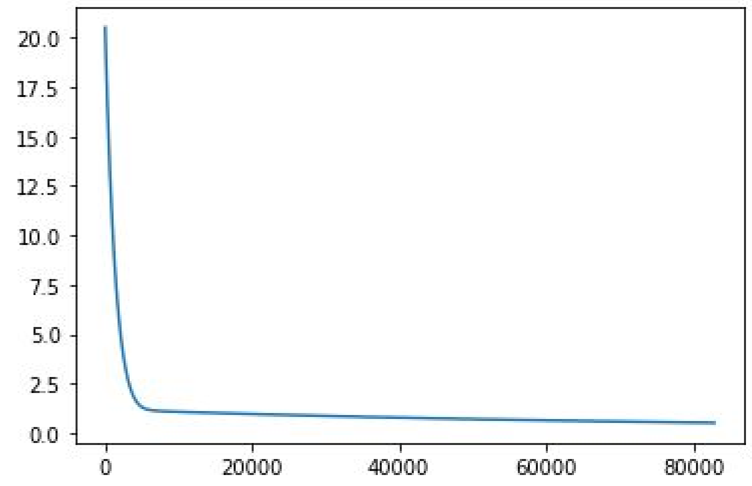
Learning rate = pow(10,-2):



Learning rate = pow(10,-3):



Learning rate = pow(10,-4):



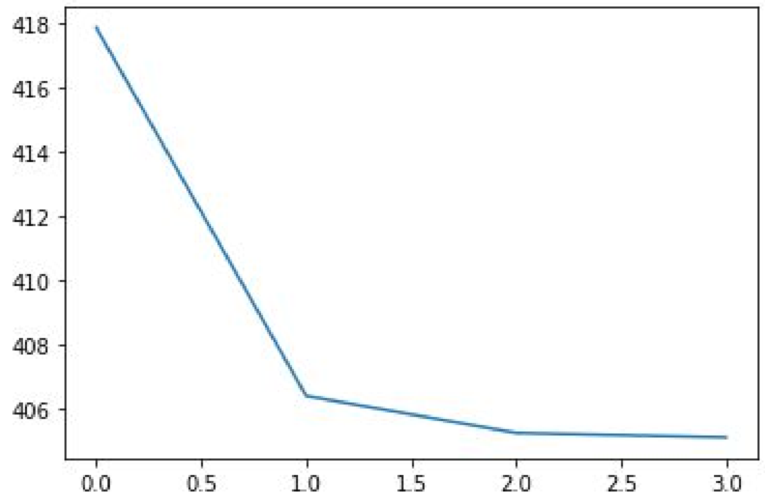
Not experiment data after pow(10,-4), It took almost an hour to test pow(10,-4). According to the following analysis of the convergence speed, the times used to wait for pow(10,-5), pow(10,-6), and pow(10,-7). are estimated to be 10 hours, 100 hours, and 1000 hours, respectively, which is obviously undesirable. (10 hours may be considerable, but during the test, I disconnected to school servers twice, so I gave up. If maybe I can use the DGX server next time.)

b. **For each learning rate that worked for you, Report the MSE on the training data and the validation data respectively, and the number of iterations needed to achieve the convergence condition for training. What do you observe? Between different convergent learning rates, how should we choose one of the MSE is nearly identical?**
According to the abscissa form figure that the learning rate is pow(10, -1) to figure that the learning rate is pow(10, -4), I guessed that the amount of calculation required to reach the convergence condition is 10 times the previous one, whenever the learning rate is reduced to 1/10. This shows that the learning rate can not as small as possible.

c. **Use the validation data to pick the best-converged solution, and report the learned weights for each feature. Which features are the most important in deciding the house prices according to the learned weights? Compare them to your pre-analysis results (Part 0 (d)).**

I used pow(10, -3) as the learning rate, this learning rate neither makes the running time too long, but also keep a good accuracy. Fellowing chart shows the mean of prediction error of validation data

$$\frac{1}{N}\sum_{n=1}^{N}(y_i - wx_i)$$



After processing more than 5000 data in validation file , the train based on the 10^-1 learning rate is only 3% more error than 10^-4 learning rate, and the speed of former is 1000 times faster.

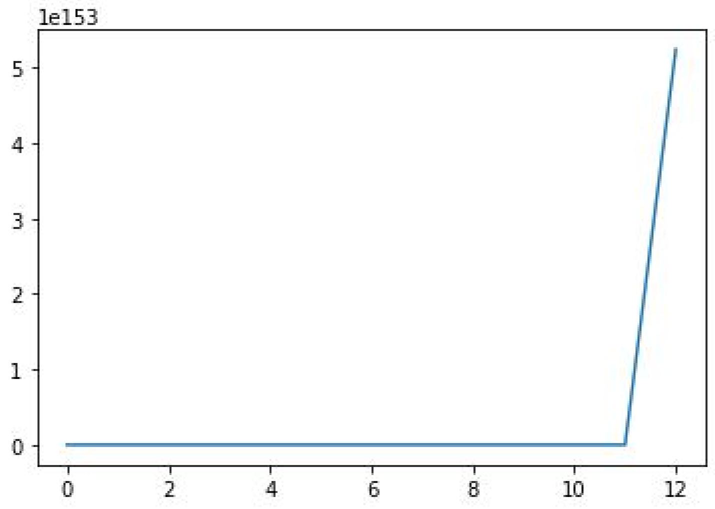
The following data represents the weight of each features

dummy	0.20356677
month	-0.06643192
day	-0.32637033
year	0.10530324
bedrooms	0.30282227
bathrooms	1.82093995
sqft_living15	2.4076606
sqft_lot15	0.08629617
floors	1.41366055
waterfront	0.79943872
view	2.5916738

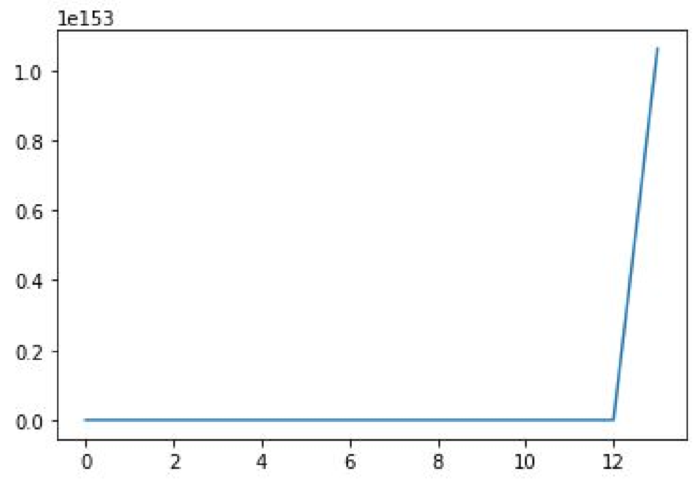
condition	0.67648862
grade	3.20328518
sqft_above	2.15223913
sqft_basement	1.70897747
yr_built	-0.51538807
yr_renovated	0.71775132
zipcode	-0.33819132
lat	2.47111328
long	-0.1214471
sqft_living15	2.54389685
sqft_lot15	0.09525956

Part 2 (20 pts). Training with non-normalized data Use the preprocessed data but skip the normalization. Consider at least the following values for learning rate: 100, 10, 10^−1, 10^−2, 10^−3, 10^−4. For each value, train up to 10000 iterations (Fix the number of iterations for this part). If training is clearly diverging, you can terminate early. Plot the training MSE and validation MSE respectively as a function of the number of iterations. What do you observe? Specify the learning rate value (if any) that prevents the gradient descent from exploding? Compare between using the normalized and the non-normalized versions of the data. Which one is easier to train and why?

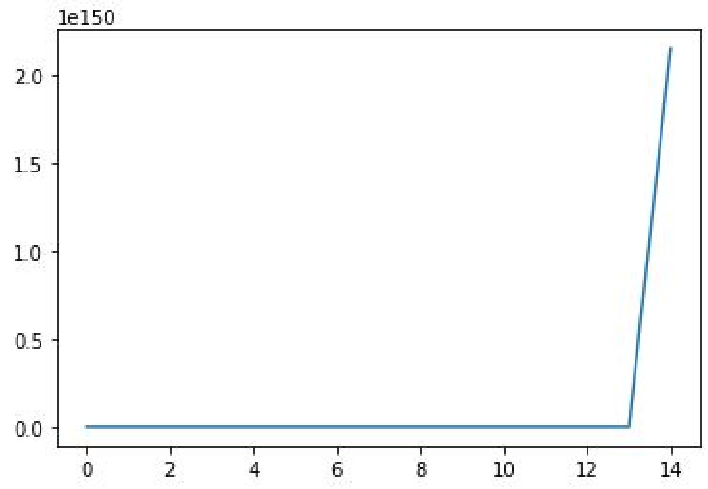
Learning rate = 100



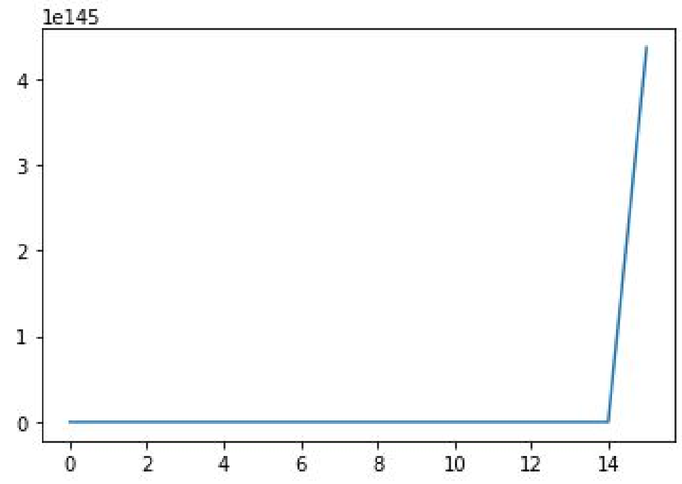
Learning rate = 10



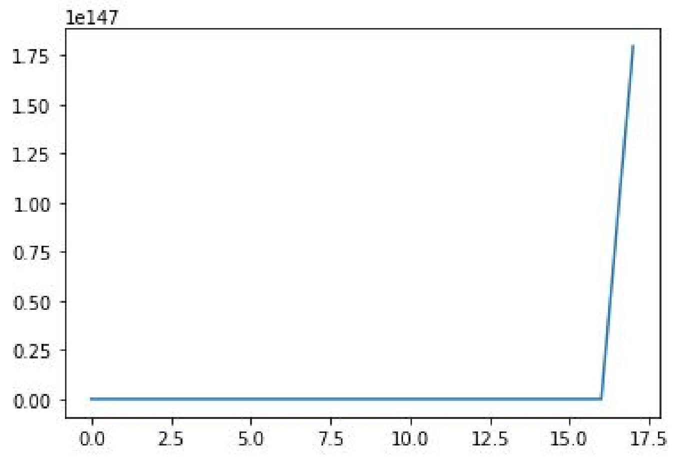
Learning rate = 1



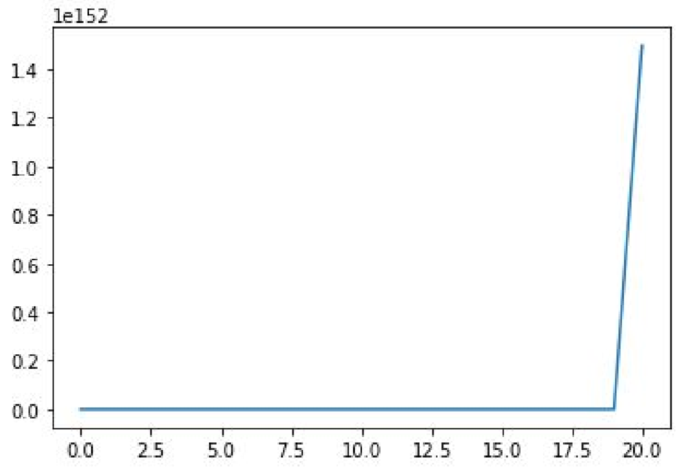
Learning rate = 10^-1



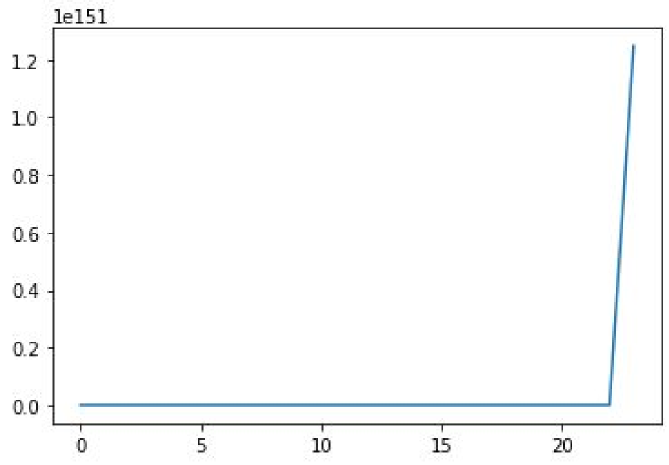
Learning rate = 10^-2



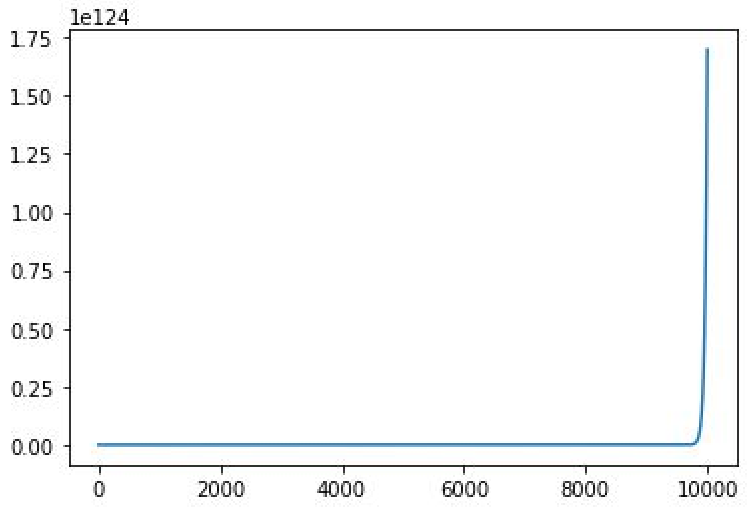
Learning rate = 10^-3:



Learning rate = 10^-4:



Learning rate = 10^-10 with max iteration 10000:



It is clear that using non-normalized data makes extremely slow the speed of convergence, I can't observed the trendency of convergence even when I used 10^-10 as the learning rate.

Part 3 (10 pts).