COMP6247 Reinforcement and Online Learning Lab 1 Report

MSc.Artificial Intelligence
Student: Wei Lou
wl4u19@soton.ac.uk
Supervisor: Dr Mahesan Niranjan

1 Introduction

THE main contents of this lab are using 2 methods to do the linear regression, one is the Stochastic Learning algorithm and the other is Recursive Least Square algorithm. Besides comparing the performance, this report also shows some improvements for stochastic learning algorithm and some built-in libraries for RLS algorithm.

2 Implementation

2.1 Study the derivation of the recursive least square algorithm.

I have done the derivation and shown on image one. The equation of k_n is:

$$k_n = \frac{\frac{1}{\lambda}P_{n-1}x_n}{1+\frac{1}{\lambda}x_n^T P_{n-1}x_n}$$

The step $k_n = P_{n-1}x_n$ is by multiply k_n on the left with the denominator part: $1+\frac{1}{\lambda}\boldsymbol{x}_n^T P_{n-1}\boldsymbol{x}_n$ on the right of the equation. Then we can get

$$\boldsymbol{k}_n = \left[\frac{1}{\lambda}P_{n-1} - \frac{1}{\lambda}\boldsymbol{k}_n\boldsymbol{x}_n^T P_{n-1}\right]\boldsymbol{x}_n$$

which is actually $k_n = P_{n-1}x_n$

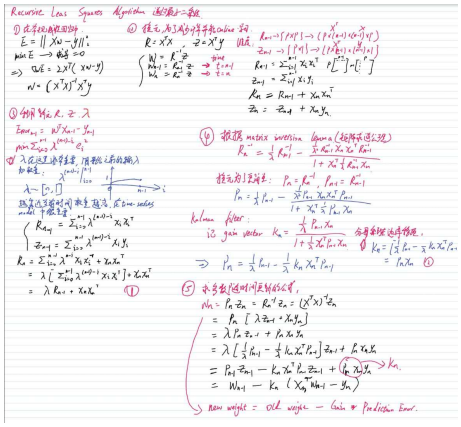


Fig. 1: Derivation of RLS

2.2 Simple Linear Regression

The synthetic data made in this task is a 500*30 matrix with random values in gaussian distribution, and the target values are constructed by function Y = X @ W + noise. W is a 30*1 vector represents the initial value of weights. By using pseudo inverse and the derivation of the linear function, the estimation of weight can be calculated as W = inverse(X.T @ X) @ X.T @ y, the use this weight vector we can get the prediction value and calculate the error. The overall residual error is 17.69 and the prediction graph and weights estimations are as follows:

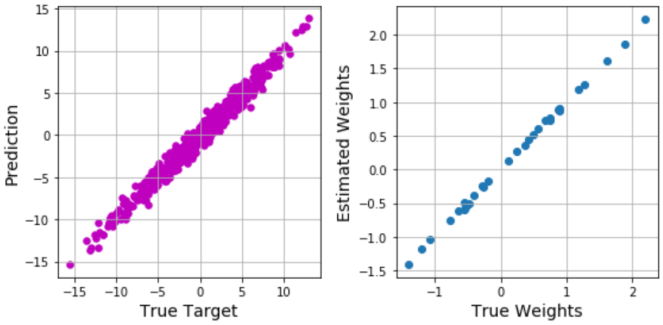


Fig. 2: Prediction and estimate error

2.3 Gradient descent and stochastic gradient descent

By using gradient descent to minimise error is also a good way to update the weights. The basic gradient descent also called batch gradient descent, update the weights by using all the data in every iteration. Because of this, it would not be affected by some noise data or special features so the loss graph is very smoothing but it is very time-consuming. Stochastic gradient descent is a kind of useful method to perform the gradient descent, in every iteration, it only uses one data vector to update the weights which largely reduced the time cost. But the loss graph is fluctuant.

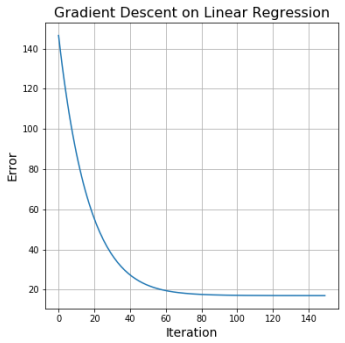


Fig. 3

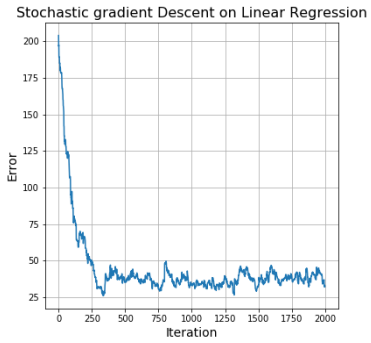


Fig. 4: Gradient descent and SGD

It's clear that the loss curve of SGD is very fluctuant because it updates the weight on one data vector which may not the right descent direction on global space. And it also can not converge on a fixed value.

2.3.1 Improvement

To fix this problem, I used 3 different methods. The simplest way is just decrease the learning rate, because this would reduce the fluctuation when approach the minimum value in gradient space.

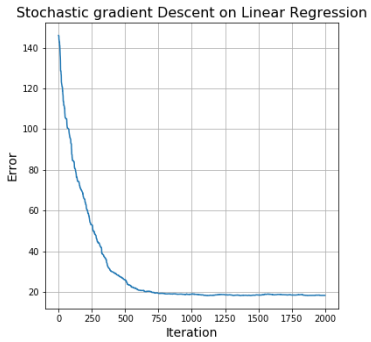


Fig. 5: SGD with smaller learning rate

And another intuitive way is to save the best model which is the model with minimum error. Every time detect the error is lower than the minimum error in the past, save the weights and the error, so the curve will be very flat at the end of the training process.

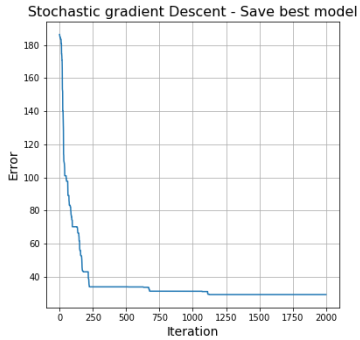


Fig. 6: Save the best model

The other method is Mini-Batch gradient descent, this method combines the advantages of SGD and BGD, in every iteration, update the weights by using a certain amount of data, find the balance of time-cost and the robustness. In this case, I choose the batch size as 16 and get the error curse in figure 6. We can see the error curve actually becomes much smooth than SGD and can converge to a fixed number.

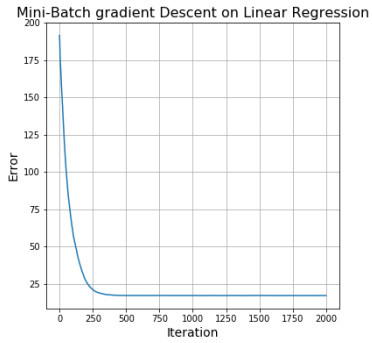


Fig. 7: Mini-Batch gradient descent

2.4 Recursive Least Squares solution

Implement the Recursive Least Squares solution and compare the performance between RLS and SGD. In this part, the training data is the same as the data used in the gradient descent algorithm. By using recursive least square methods, we can avoid using large matrix inverse operations which largely reduced the time and space cost. And in RLS, the time series is considered as a key fact in the training process, so there is a parameter lambda represents the contribution of past data. In this part, the lambda is 0.99, means the closer data will have more contribution.
The result of linear regression on synthetic data shows on figure7. The error is around 26.40 which is slightly higher than the SGD algorithm.

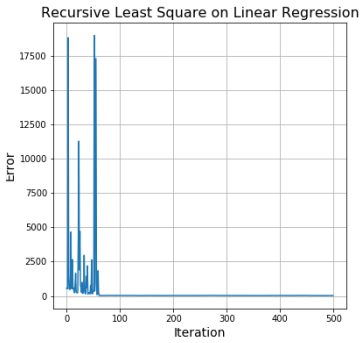


Fig. 8: Recursive Least Squares error curve-30

And by changing the dimensions from 30 to 100, it's become longer for the algorithm to converge, because when using 100-dimension features, the algorithm uses around 200 iterations to converge which is much more than using 30-dimension features.High dimension data increase the sparseness and will have more bias and variance for the model, then make it harder to converge.

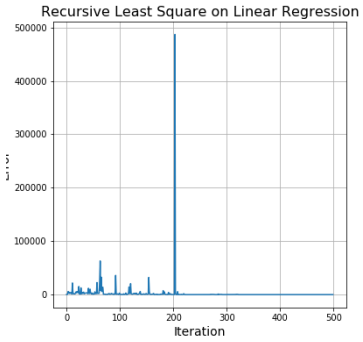
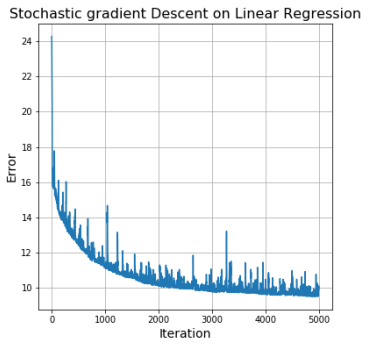


Fig. 9: Recursive Least Squares error curve-100

2.5 Linear regression on real data

In this part, the SGD, Minibatch gradient descent and RLS algorithm are compared on a UCI dataset called "Wine" datasets. The shape of the training data is 4898 * 11, and the target data is 4898 * 1, the iterations for SGD is 10000 and learning rate is 0.001. And the iterations for Mini-Batch gradient descent is 2000, the learning rate is the same. The preprocess for the dataset is the min-max normalization.
The final errors for these 2 methods are 10.18 and 13.0.



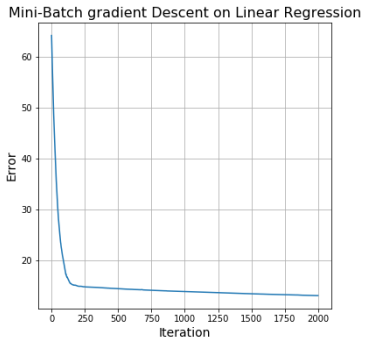


Fig. 10: SGD and mini-Batch Gradient descent

And for the RLS algorithm, as first I encounter a singular matrix problem which means the initial P can not be inversed. So I initialize the P with an identity matrix multiply a small constant number. The final error for RLS is about 19.

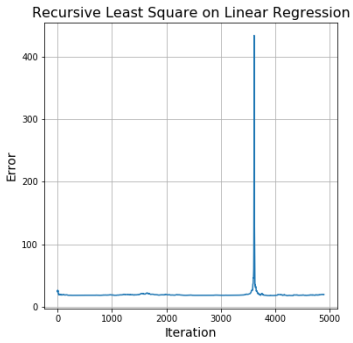


Fig. 11: RLS algorithm

2.6 Experiment with some build-in library

Besides perform the RLS ourselves, for comparison, I chose a machine learning library called 'padasip', by using it's filiter.FilterRLS, set the n as the dimension of feature which is 11 and mu is 0.99. Print its prediction and target are on figure11 and the overall error is 15.25 which is slightly better than our codes.

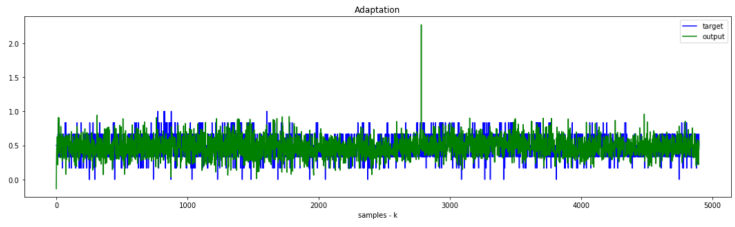


Fig. 12: Comparison of target and prediction

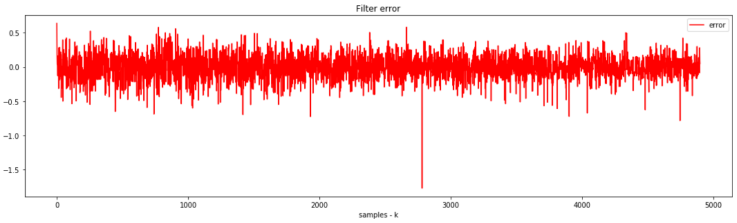


Fig. 13: Error on each data sample