COMP6247 Reinforcement and Online Learning - Lab One

DAJI LI
Student ID: 31326234
E-mail: ld5n19@soton.ac.uk

February 2020

1 The derivation of the recursive least Squares algorithm

I had done the derivation of the recursive least Squares (RLS) algorithm given in Appendix A. P is a matrix in size [p, p], $\boldsymbol{k}$ is the gain vector in shape (p, 1). In RLS, $\boldsymbol{k}_n$ is generated by P_{n-1} according to equation (1), while P_n is generated by $\boldsymbol{k}_n$ according to equation (2), and the iteration follows $P_{n-1} -> \boldsymbol{k}_n -> P_n -> \boldsymbol{k}_{n+1} ->...$

$$\boldsymbol{k}_n = \frac{\frac{1}{\lambda} P_{n-1} \boldsymbol{x}_n}{1 + \frac{1}{\lambda} \boldsymbol{x}_n^T P_{n-1} \boldsymbol{x}_n} \tag{1}$$

$$P_n = \frac{1}{\lambda} P_{n-1} - \frac{1}{\lambda} k_n x_n^T P_{n-1} \tag{2}$$

According to equations (3) (4) (5), we can find the relationship of $\boldsymbol{k}_n = P_n x_n$, which we can use it in generating weight $w_n = P_n z_n$ to $w_n = w_{n-1} - k_n \left(x_n^T w_{n-1} - y_n\right)$.

$$\begin{aligned} P_n &= \frac{1}{\lambda} P_{n-1} - \frac{\frac{1}{\lambda^2} P_{n-1} \boldsymbol{x}_n \boldsymbol{x}_n^T P_{n-1}}{1 + \boldsymbol{x}_n^T \frac{1}{\lambda} P_{n-1} \boldsymbol{x}_n} \\ &= \frac{1}{\lambda} P_{n-1} - \frac{1}{\lambda} \boldsymbol{k}_n \boldsymbol{x}_n^T P_{n-1} \end{aligned} \tag{3}$$

$$\boldsymbol{k}_n = \frac{\frac{1}{\lambda} P_{n-1} \boldsymbol{x}_n}{1 + \frac{1}{\lambda} \boldsymbol{x}_n^T P_{n-1} \boldsymbol{x}_n} \tag{4}$$

$$\begin{aligned} \boldsymbol{k}_n &= \left[\frac{1}{\lambda} P_{n-1} - \frac{1}{\lambda} \boldsymbol{k}_n \boldsymbol{x}_n^T P_{n-1}\right] \boldsymbol{x}_n \\ &= P_n \boldsymbol{x}_n \end{aligned} \tag{5}$$

$New = Old - Gain * PredictionError$, is the core thinking pattern of RLS, which the gain vector will update the weights vector with prediction error by iteration. At the same time, because of non-stationary environment, forgetting factor λ will filter the noise, which will lead a good estimated weights closed to true weights compare with Linear Regression by Gradient Descent and Stochastic Gradient Descent on Linear Regression.

2 Linear Regression

2.1 Simple Linear Regression

For the simple linear regression, we can (6) and (7), we can be easy to get Pseudo inverse solution: $\mathbf{w} = \left(X^T X\right)^{-1} X^t \mathbf{y}$

$$E = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 \tag{6}$$

To minimize:

$$\nabla_{\boldsymbol{w}} E = 2X^T(\boldsymbol{y} - X\boldsymbol{w}) \quad = 0 \tag{7}$$

And use this Pseudo inverse solution to predict targets, and the result show in Fig 1.

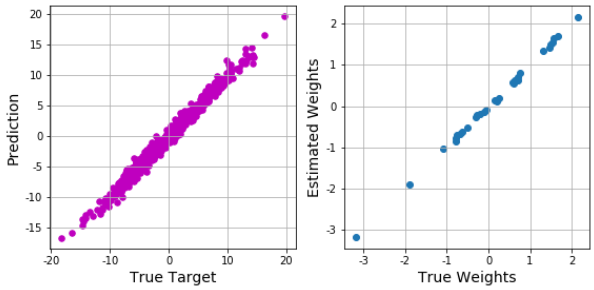


Figure 1: Simple Linear Regression: the results of prediction with target y and weight w

2.2 Linear Regression by Gradient Descent and Stochastic Gradient Descent (SGD) on Linear Regression

On such a synthetic dataset, implement linear regression estimated in closed form, linear regression estimated by gradient descent and linear regression estimated using stochastic gradient descent. And the results of the scatter of predicted and true targets and the learning curves for the gradient descent solutions show in Fig. 2 and Fig. 3.

Refer to Fig. 2 and Fig. 3 (left and center), with the default learning rates (LR), we can find the result of batch training ($LR = 0.0001$) is better than the result of stochastic training ($LR = 0.05$) with a better convergence performance.

In Fig 3 (center), between iterations 250 and 2000, where we decide to stop will return answers that differ in error by about 20. The LR represents the step size of each overshooting. A too high learning rate will make the learning jump over minima but a too low learning rate will either take too long to converge or get stuck in an undesirable local minimum. If we modify the LR to 0.005, the convergence will become smooth, which refers to Fig. 3 (right), but the converging speed also will decrease. **In order to solve this problem, we also can use mini batch with k-mean to reduce the effects of learning rates.**

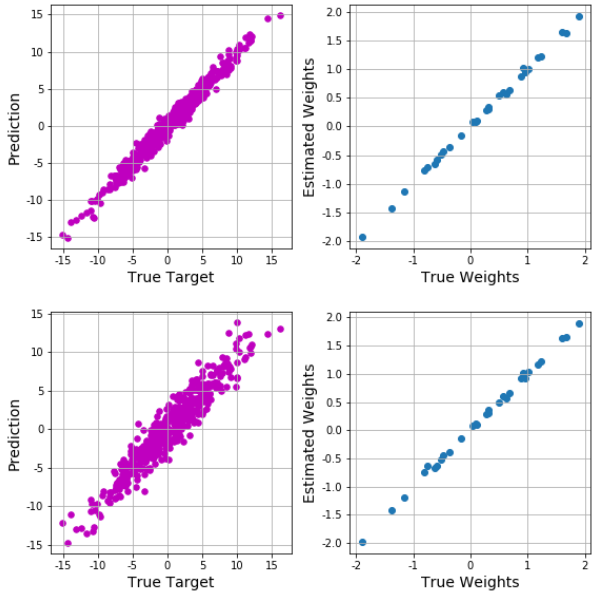


Figure 2: Linear Regression by Gradient Descent ($LR = 0.0001$) (top) Stochastic Gradient Descent on Linear Regression ($LR = 0.05$) (down): the results of prediction with target y and weight w

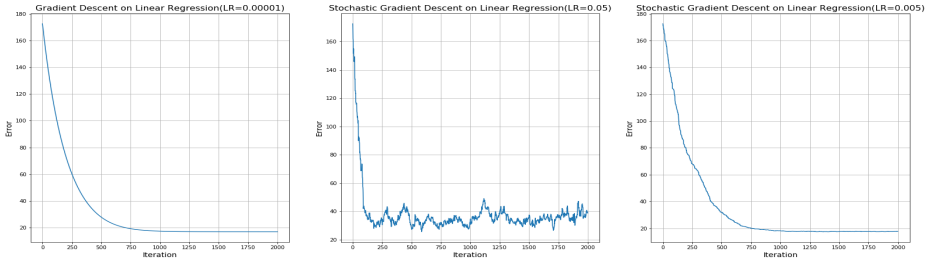


Figure 3: Gradient Descent Training of Linear Regression: Regression by Gradient Descent ($LR = 0.0001$) (left), Stochastic Gradient Descent on Linear Regression ($LR = 0.05$) (center) and Stochastic Gradient Descent on Linear Regression ($LR = 0.005$) (right)

3 Recursive Least Squares solution

According to section 1, implement Recursive Least Squares solution and show error reduces as a function of iteration in Fig. 4. We can find the learning curve of recursive least squares decrease rapidly at first.

There is relationship between the speed of convergence and the dimensionality of the problem. According to equation (4), if dimensionality $\mathcal{R}$ increases, the value of denominator will increase at the same time, it will lead the gain vector $\boldsymbol{k}_n$ become smaller, which will lead each iteration become smaller and samller.

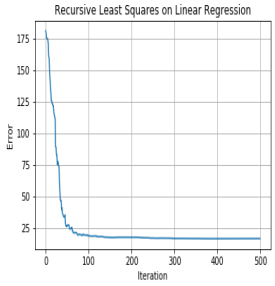
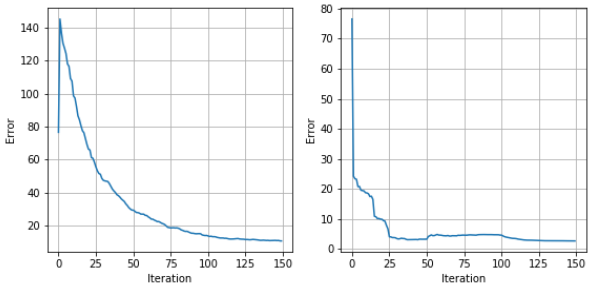


Figure 4: Recursive Least Squares solution: Gradient Descent Training of Linear Regression

4 UCI dataset solution

Implement stochastic gradient descent and recursive least squares for Iris dataset from the UCI Machine Learning Repository, the learning curves show in Fig. 5. In SGD and RLS algorithm, the weight of linear model begin in a same random weight. We can find the results of RLS have a better performance of residual error after convergence. That is because recursive least squares can filter the noise via forgetting factor λ.




```
Initial: 76.54104240783336
converaged(SGD): 10.919027214753157
converaged(RSL): 2.686982857800611
```

Figure 5: Gradient Descent Training of Linear Regression: stochastic gradient descent($LR = 0.001$) (left) and recursive least squares (right)