School of Electronics and Computer Science
University of Southampton

COMP6247(2019/20): Reinforcement and Online Learning

Recursive Least Squares

Issue	10/02 2020
Deadline	20/02 (10:00 AM)
Feedback by	10/06/2020

Objective

To study online learning using the Recursive Least Squares (RLS) algorithm and its application to solving linear regression problems. We will build on this and address time series analysis problems in the next task.

Introduction

In online estimation of a regression problem defined by $\{\boldsymbol{x}_n, y_n\}_{n=1}^N$ where the inputs $\boldsymbol{x}_n \in \mathcal{R}^p$ and the target y_n is scalar, we consider the arrival of data to be sequential

$$... \ \{\boldsymbol{x}_{n-1}, y_{n-1}\}, \ \{\boldsymbol{x}_n, y_n\}, \ \{\boldsymbol{x}_{n+1}, y_{n+1}\}, \ ...$$

Our task is to update the solution for the parameters $\boldsymbol{w}_{n-1}$ we have at time $(n-1)$ upon the arrival of data $\{\boldsymbol{x}_n, y_n\}$ at time n *without* refering back to data seen in the past: $\{\boldsymbol{x}_i, y_i\}_{i=1}^{n-1}$. What summary information do we carry with us is the basic question underlying this topic. The RLS algorithm achieves this by updating an inverse of a matrix (inverse covariance matrix, P_{n-1}) sequentially. The algorithm is as follows:

$$\begin{aligned}
\boldsymbol{k}_n &= \frac{\frac{1}{\lambda} P_{n-1} \boldsymbol{x}_n}{1 + \frac{1}{\lambda} \boldsymbol{x}_n^T P_{n-1} \boldsymbol{x}_n} \\
e(n) &= y_n - \boldsymbol{w}_{n-1}^T \boldsymbol{x}_n \\
\boldsymbol{w}_n &= \boldsymbol{w}_{n-1} + \boldsymbol{k}_n e(n) \\
P_n &= \frac{1}{\lambda} P_{n-1} - \frac{1}{\lambda} \boldsymbol{k}_n \boldsymbol{x}_n^T P_{n-1}
\end{aligned}$$

Note several aspects of the algorithm:

- There is no matrix inversion at each step, the update of P_{n-1} uses the matrix inversion lemma.
- There is a gain term $\boldsymbol{k}_n$ which gives a structure: the new values of parameters being the old values to which we add a term given by the error and this gain.
- $0 < \lambda < 1$ is a user defined parameter that controls how fast the contribution of data seen in the past decays.

The derivation of the algorithm is given in Appendix.

Tasks

1. Study the derivation of the recursive least Squares algorithm given in Appendix A. This should consist of writing out (pen and paper) all the equations by hand and noting the dimensions of each term you encounter. In particular, you understand how the step $\boldsymbol{k}_n = P_n \boldsymbol{x}_n$ for the gain term was arrived at. State in your report that you have done this.

2. Implement a simple linear regression on a synthetic problem you construct. This can be achieved by generating covariates from a random number generator in a design matrix $X \in \mathcal{R}^{N \times p}$, for suitable values of N and p, defining a set of weights $w_* \in \mathcal{R}^p$, again at random, constructing target data by the product $X\, w_*$. It is usual to corrupt the product by adding a small amount of noise to it to simulate how measurements are made in the real world.

   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. Show in a graph the scatter of predicted and true targets and the learning curves for the gradient descent solutions. Graphs you obtain will be similar to those shown in Figs. 1 and 2.

   With stochastic learning, note that the learning curve looks noisy and does not converge to a fixed answer. In Fig. 2(b), for example, between iterations 250 and 2000, where we decide to stop will return answers that differ in error by about 20. Why does this happen? Implement a method to avoid this and return a more consistent answer.

   Snippets of code are provided in Appendix B. You are encouraged to work on your own without looking at these to avoid any danger of learning bad proramming habits. But if you cannot get started, these snippets may be of help.

3. Implement your own Recursive Least Squares solution and show how error reduces as a function of iteration. Is there a trend / relationship between the speed of convergence and the dimensionality of the problem?

4. Download a dataset from the UCI Machine Learning Repository (`https://archive.ics.uci.edu/ml/index.php`). Compare the learning curves of stochastic gradient descent and recursive least squares on this problem.

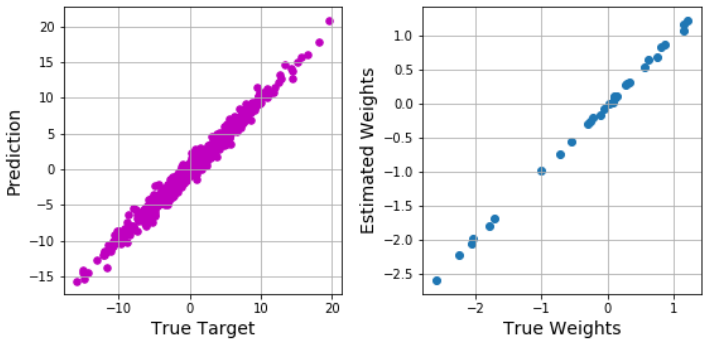
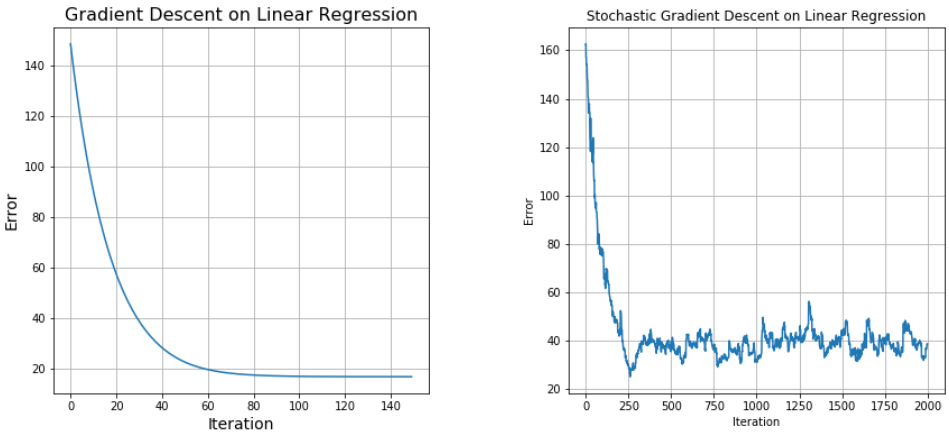


Figure 1: Results of Linear Regression on Synthetic Data. Left: true value of target against prediction from the model; Right: true values of the underlying parameters used to construct the target against values estimated by solving the regression problem.



(a) Batch Training **(b)** Stochastic Training

Figure 2: Gradient Descent Training of Linear Regression

Marking Scheme

This assignment counts for 10% of assessed work for this module and should be seen as *Laboratory Work* as described in the module specification. Attempting *all* the tasks diligently and correctly earns seven points (7/10). You need to "go an extra mile" and show a detectable element of creativity, originality or polished presentation to earn anything more. The possible marks obtainable are in the set {0, 7, 10}.

Report

Describe the work you have done as a short report. Upload a *pdf* file **no longer than four pages**[1] using the ECS handin system: `http://handin.ecs.soton.ac.uk`. Please use LaTeX to typeset your report. Please make sure your name and email are included in the report you upload.

[1]The page limit is strictly enforced because I will be printing the work. Hence no cover pages, appendices etc.

Appendix A: Derivation of the Recursive Least Squares Algorithm

In linear regression, we define the error as sum over all the data:

$$\begin{aligned} E &= \sum_{i=1}^{N} e_i^2 \\ &= \sum_{i=1}^{N} \left(\boldsymbol{x}_i^T \, \boldsymbol{w} - y_i \right)^2 \end{aligned}$$

We saw how to write this in matrix form:

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

where the rows of matrix X are transposes of the input data vectors (covariates) $\boldsymbol{x}_i^T$ and the elements of vextor $\boldsymbol{y}$ are the targets y_i (the responses).
At this point you should be clear what the dimensions of X, $\boldsymbol{w}$ and $\boldsymbol{y}$ are.
We solve the linear regression problem by minimising E, with respect to $\boldsymbol{w}$. We differentiate (*i.e.* calculate the gradient) and set it to zero. Note, for the linear model and sum squared error, E is a quadratic function of the parameters $\boldsymbol{w}$. The problem we solve is:

$$\min_{\boldsymbol{w}} ||X \, \boldsymbol{w} - \boldsymbol{y}||_2^2.$$

The derivative:

$$\nabla_{\boldsymbol{w}} E = 2 \, X^T \, (X \, \boldsymbol{w} - \boldsymbol{y})$$

Setting it to zero

$$\begin{aligned} \left(X^T \, X \right) \boldsymbol{w} &= X \, \boldsymbol{y} \\ \boldsymbol{w} &= \left(X^T \, X \right)^{-1} X^T \, \boldsymbol{y} \end{aligned}$$

We will also introduce matrix $R = X^T \, X$ and vector $\boldsymbol{z} = X^T \, \boldsymbol{y}$, giving $R \, \boldsymbol{w} = \boldsymbol{z}$ as the equation to solve for the unknowns.
Our task now is to solve such a problem sequentially, as data arrive one at a time. Hence at time $(n-1)$, we will be writing

$$\boldsymbol{w}_{n-1} = R_{n-1}^{-1} \, \boldsymbol{z}_{n-1}$$

and at time n

$$\boldsymbol{w}_n = R_n^{-1} \, \boldsymbol{z}_n$$

Note the dimensions: R_{n-1} is a $p \times p$ matrix, obtained by multiplying $X_{n-1}^T \, X_{n-1}$ ($p \times (n-1) \times ((n-1) \times p)$ and $\boldsymbol{z}_{n-1}$ is a vector obtained by multiplying a matrix and a vector ($p \times (n-1) \times ((n-1) \times 1)$.
Also note:

$$\begin{aligned} R_{n-1} &= X_{n-1}^T \, X_{n-1} \\ &= \sum_{i=1}^{n-1} \boldsymbol{x}_i \, \boldsymbol{x}_i^T \\ \boldsymbol{z}_{n-1} &= \sum_{i=1}^{n-1} \boldsymbol{x}_i \, y_i \end{aligned}$$

and

$$\begin{aligned} R_n &= \sum_{i=1}^{n-1} \boldsymbol{x}_i \, \boldsymbol{x}_i^T \\ &= R_{n-1} + \boldsymbol{x}_n \, \boldsymbol{x}_n^T \\ \boldsymbol{z}_n &= \sum_{i=1}^{n-1} \boldsymbol{x}_i \, y_i \\ &= \boldsymbol{z}_{n-1} + \boldsymbol{x}_n \, y_n \end{aligned}$$

$$e_{n-1} = \boldsymbol{w}^T \boldsymbol{x}_{n-1} - y_{n-1}$$

Forgetting the past: We introduce a forgetting factor λ and weight the contributions of past inputs so that the distant past has less influence than recent past. This is not necessary in the context of a static regression problem we are solving in this exercise, but can be crucial in time series modelling if the underlying environment is non-stationary; *i.e.* the statistical properties of the environment change with time.

$$\min \sum_{i=0}^{n-1} \lambda^{(n-1)-i} \, e_i^2$$

$$\begin{aligned} R_{n-1} &= \sum_{i=0}^{n-1} \lambda^{(n-1)-i} \, \boldsymbol{x}_i \, \boldsymbol{x}_i^T \\ \boldsymbol{z}_{n-1} &= \sum_{i=0}^{n-1} \lambda^{(n-1)-i} \, \boldsymbol{x}_i \, y_i \end{aligned}$$

$$\begin{aligned} R_n &= \sum_{i=0}^{n-1} \lambda^{n-i} \, \boldsymbol{x}_i \, \boldsymbol{x}_i^T + \boldsymbol{x}_n \, \boldsymbol{x}_n^T \\ &= \lambda \left[\sum_{i=0}^{n-1} \lambda^{(n-1)-i} \, \boldsymbol{x}_i \, \boldsymbol{x}_i^T \right] + \boldsymbol{x}_n \, \boldsymbol{x}_n^T \\ &= \lambda \, R_{n-1} + \boldsymbol{x}_n \, \boldsymbol{x}_n^T \end{aligned}$$

It follows from matrix inversion lemma for rank one update of inverse:

$$R_n^{-1} = \frac{1}{\lambda} R_{n-1}^{-1} - \frac{\frac{1}{\lambda^2} R_{n-1}^{-1} \boldsymbol{x}_n \, \boldsymbol{x}_n^T \, R_{n-1}^{-1}}{1 + \boldsymbol{x}_n^T \frac{1}{\lambda} R_{n-1}^{-1} \boldsymbol{x}_n}$$

For convenience, we define: $P_n = R_n^{-1}$, $P_{n-1} = R_{n-1}^{-1}$ and write

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

where, we have introduced a *gain vector* $\boldsymbol{k}_n$:

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

$$
\begin{aligned}
\boldsymbol{z}_n &= \sum_{i=0}^{n} \lambda^{n-i}\, \boldsymbol{x}_i\, y_i \\
&= \lambda\, \boldsymbol{z}_{n-1} + \boldsymbol{x}_n\, y_n
\end{aligned}
$$

Rearranging the terms of the expression for $\boldsymbol{k}_n$, we also have:

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

The solution for the parameter vector at time n is

$$
\begin{aligned}
\boldsymbol{w}_n &= P_n\, \boldsymbol{z}_n \\
&= P_n\, [\lambda \boldsymbol{z}_{n-1} + \boldsymbol{x}_n\, y_n] \\
&= \lambda\, P_n\, \boldsymbol{z}_{n-1} + P_n\, \boldsymbol{x}_n\, y_n \\
&= \lambda \left[\frac{1}{\lambda} P_{n-1} - \frac{1}{\lambda} \boldsymbol{k}_n\, \boldsymbol{x}_n^T\, P_{n-1}\right] \boldsymbol{z}_{n-1} + P_n\, \boldsymbol{x}_n\, y_n \\
&= P_{n-1}\, \boldsymbol{z}_{n-1} - \boldsymbol{k}_n\, \boldsymbol{x}_n^T\, P_{n-1}\, \boldsymbol{z}_{n-1} + P_n\, \boldsymbol{x}_n\, y_n \\
&= \boldsymbol{w}_{n-1} - \boldsymbol{k}_n \left(\boldsymbol{x}_n^T\, \boldsymbol{w}_{n-1} - y_n\right)
\end{aligned}
$$

New = Old - Gain X Prediction Error

Appendix B: Snippets of Code

Simple linear regression

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

# Number of data and dimensions
#
N, p = 500, 30

# Input data (covariates)
#
X = np.random.randn(N, p)

# True parameters
#
wTrue = np.random.randn(p, 1)

# Set up targets (response)
#
yTarget = X @ wTrue + 0.8*np.random.randn(N,1)

# Estimate the weights by pseudo inverse
#
wEst = np.linalg.inv(X.T @ X) @ X.T @ yTarget

# Predict from the model
#
yEst = X @ wEst

# Scatter plot of predictions against truth
#
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8,4))
ax[0].scatter(yTarget, yEst, c='m', s=30)
ax[0].grid(True)
ax[0].set_xlabel("True Target", fontsize=14)
ax[0].set_ylabel("Prediction", fontsize=14)

ax[1].scatter(wTrue, wEst)
ax[1].grid(True)
ax[1].set_xlabel("True Weights", fontsize=14)
ax[1].set_ylabel("Estimated Weights", fontsize=14)
plt.tight_layout()

# Error from the model
#
print("Residual Error: %3.2f" %(np.linalg.norm(yEst - yTarget)))
```

Solving Linear Regression by Gradient Descent

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

# Set up synthetic data
#
N, p = 500, 30
X = np.random.randn(N, p)
wTrue = np.random.randn(p, 1)
yTarget = X @ wTrue + 0.8*np.random.randn(N,1)

# Initial guess and error
#
w0 = np.random.randn(p,1)
E0 = np.linalg.norm(yTarget - X @ w0)

# Parameters for gradient descent
#
MaxIter = 150
lRate = 0.0001
Eplot = np.zeros((MaxIter, 1))

wIter= w0
for iter in range(MaxIter):
    wIter = wIter - lRate * X.T @ (X @ wIter - yTarget)
    Eplot[iter] = np.linalg.norm(X @ wIter - yTarget)

fig, ax = plt.subplots(figsize=(6,6))
ax.plot(Eplot)
ax.set_xlabel("Iteration", fontsize=14)
ax.set_ylabel("Error", fontsize=14)
ax.grid(True)
ax.set_title("Gradient Descent on Linear Regression",
             fontsize=16)

print("Residual Error (Initial)  : %3.2f" %(E0))
print("Residual Error (Converged): %3.2f"
                %(np.linalg.norm(X @ wIter  - yTarget)))
plt.savefig("GDLearningCurve.png")
```

Stochastic Gradient Descent on Linear Regression

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

# Number of data and dimensions
#
N, p = 500, 30

# Input data (covariates)
#
X = np.random.randn(N, p)

# True parameters
#
wTrue = np.random.randn(p, 1)

# Set up targets (response)
#
yTarget = X @ wTrue + 0.8*np.random.randn(N,1)

w0 = np.random.randn(p,1)
E0 = np.linalg.norm(yTarget - X @ w0)
print(E0)

MaxIter = 2000
lRate = 0.05
Eplot = np.zeros((MaxIter, 1))

wIter= w0
for iter in range(MaxIter):
    j = np.floor(np.random.rand()*N).astype(int)
    xj = X[j,:]
    xj = np.array([X[j,:]]).T
    yj = yTarget[j,:]
    yPred = xj.T @ wIter
    wIter = wIter - lRate * (yPred - yj) * xj
    Eplot[iter] = np.linalg.norm(yTarget - X @ wIter)

print(np.linalg.norm(yTarget - X @ wIter))
fig, ax = plt.subplots()
ax.plot(Eplot)
ax.set_xlabel("Iteration")
ax.set_ylabel("Error")
ax.grid(True)
ax.set_title("Stochastic Gradient Descent on Linear Regression")
plt.savefig("SGDLearningCurve.png")
```