School of Electronics and Computer Science
University of Southampton

COMP6247(2020/21): Reinforcement and Online Learning

Recursive Least Squares

Issue	08/02/2021
Deadline	22/02/2021 (10:00 AM)

Objective

To study online learning using the Recursive Least Squares (RLS) algorithm and its application to solving linear regression problems.

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

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

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.

Tasks for Assessment

1. 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: (i) linear regression estimated in closed form; (ii) linear regression estimated by gradient descent; and (iii) 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.

2. 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?

3. 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.

4. Find an implementation of the RLS algorithm in a software toolbox. `MATLAB`'s Signal Processign toolbox, Python packages `statsmodels` and `Padasip` are possibilities. Compare your results on one of the above tasks with that from the implementation in the toolbox.

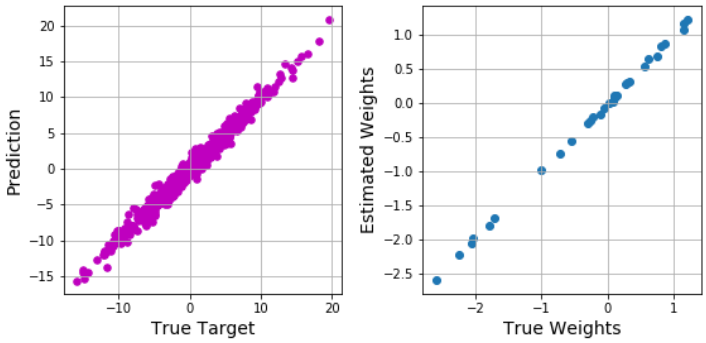
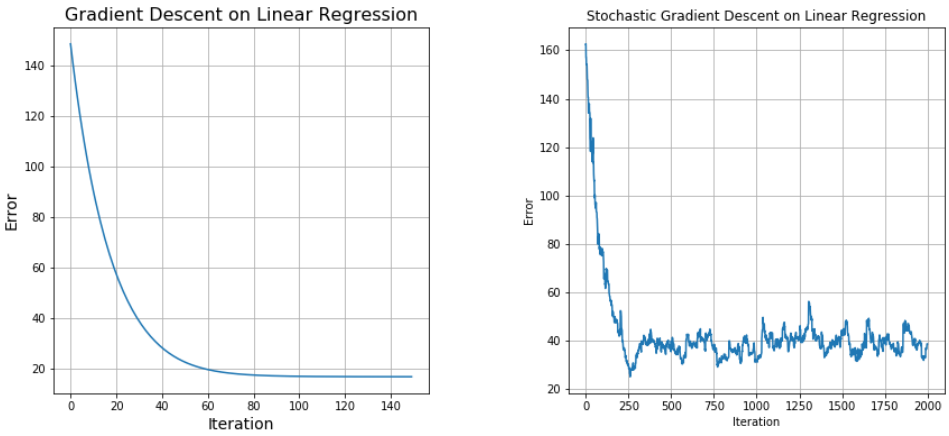


Figure 1: Results of Linear Regression on Synthetic Data. Left: true value of target against prediction from the model; Right: true values of the underlying parameters used to construct the target against values estimated by solving the regression problem.

Optional Task – Not for Assessment

We considered adaptive noise cancellation as an example of online learning. The example illustrated in class used the Least Mean Squares (LMS) algorithm for adapting the weights of



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

Figure 2: Gradient Descent Training of Linear Regression

the filter. Replace the filter with a Recursive Least Squares algorithm and study its properties. You could change the dynamics of the noise, the filter that models the propagation path and compare how the LMS and RLS algorithms adapt in a changing environment.

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 7 points (7/10). You need to "go an extra mile" and show a detectable element of creativity, originality or polished presentation to earn more.

Report

Describe the work you have done as a short report. Upload a *pdf* file **four pages** 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.

Appendix A: 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")
```