The University of Texas at Austin
Department of Electrical and Computer Engineering

EE380L: Data Mining — Spring 2016

Problem Set Two, Part I

Constantine Caramanis — Due: March 7, 2016.

This is Part I of Problem Set 2. This part focuses on Gradient Descent, and a powerful extension called Stochastic Gradient Descent. Part II will focus on probability and on machine learning/statistics.

Gradient Descent

1. Stochastic Gradient Descent (SGD). Recall the basic problem of multivariate regression: we are given n p-dimensional row vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, and n numbers, $\{y_i\}$. We want to find a p-dimensional vector $\beta = (\beta_1, \ldots, \beta_n)$, so that on average, $\sum_j x_{ij}\beta_j - y_i$ is small for all i. That is, we want to find the values of the vector β that provide the least-squares fit, i.e., they minimize:

$$\min : \ \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{p} x_{ij}\beta_j - y_i)^2\right) = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i \cdot \beta - y_i)^2.$$

   In the last four lectures, we have spent time discussing gradient descent. Once the gradient is computed, this algorithm is very simple. It updates the current iterate β, via

$$\beta_{t+1} = \beta_t - \eta \nabla f(\beta_t),$$

   where

$$f(\beta) = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i \cdot \beta - y_i)^2.$$

   Note that in this case, by the linearity of the derivative (the derivative of a sum is the sum of the derivatives), we have

$$\begin{aligned} \nabla f(\beta) &= \frac{1}{n}\sum_{i=1}^{n} \nabla_\beta (\boldsymbol{x}_i \cdot \beta - y_i)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n} 2\boldsymbol{x}_i(\boldsymbol{x}_i \cdot \beta - y_i). \end{aligned}$$

   In other words, *to compute the gradient, we need to touch all the data.* For small problems this may not be an issue. But for large problems, this could be an issue.

   Stochastic Gradient Descent (SGD) is a method designed to deal precisely with this problem, and it is now very widely used in industry. It has in large part been employed for training of deep networks, in part because it is so light weight, but also because it lends itself to parallelization.

The idea comes from the observation (one can prove this – it's not obvious) that if instead of using the gradient of a function, we use a *noisy but unbiased version of the gradient*, then gradient descent still converges. That is, instead of running:

$$\beta_{t+1} = \beta_t - \eta \nabla f(\beta_t),$$

we run

$$\beta_{t+1} = \beta_t - \eta \tilde{g}(\beta_t),$$

where $\tilde{g}(\beta_t)$ is a random variable, whose expectation is equal to $\nabla f(\beta_t)$, then the algorithm still converges.

a. In this part of the problem you will investigate this claim. The problem asks you to minimize a quadratic function using gradient descent, and then to repeat, adding independent Gaussian noise to the computed gradient at each iteration. Note that *in reality you would never do this.* The point of this (part of the) exercise is only to illustrate a point, it is not to suggest a method that you might want to use!

   Consider the quadratic function

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top Q \boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x},$$

   where

$$Q = \begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix}, \qquad \boldsymbol{q} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}.$$

   Start from the point $\boldsymbol{x}_0 = \begin{pmatrix} 10 \\ 10 \end{pmatrix}$, and solve the problem using gradient descent. Make sure to be clear about what step size you choose!

   Then, solve again, but this time at each iteration, replace $\nabla f(\boldsymbol{x})$ by $\tilde{\boldsymbol{g}} = \nabla f(\boldsymbol{x}) + \boldsymbol{w}$, where $\boldsymbol{w} \sim N(0, I/10)$. That is, at each point in time, add independent Gaussian noise $N(0, 1/10)$ to each coordinate of $\nabla f(\boldsymbol{x})$.

   What step size do you need to choose for convergence? Plot the convergence together with the convergence for standard gradient descent. Gradient descent may be faster, but amazingly, the noisy version also converges!

Now to discuss how this idea is used in practice. Consider a simple setting where we have only 4 data points:

$$\text{Data} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), (\boldsymbol{x}_3, y_3), (\boldsymbol{x}_4, y_4)\}.$$

Then the function we wish to minimize is

$$f(\beta) = \frac{1}{4}\left((\boldsymbol{x}_1 \cdot \beta - y_1)^2 + (\boldsymbol{x}_2 \cdot \beta - y_2)^2 + (\boldsymbol{x}_3 \cdot \beta - y_3)^2 + (\boldsymbol{x}_4 \cdot \beta - y_4)^2\right).$$

The gradient of $f(\beta)$ at the point β is given by:

$$\nabla f(\beta) = \frac{1}{4}\left(\nabla(\boldsymbol{x}_1 \cdot \beta - y_1)^2 + \nabla(\boldsymbol{x}_2 \cdot \beta - y_2)^2 + \nabla(\boldsymbol{x}_3 \cdot \beta - y_3)^2 + \nabla(\boldsymbol{x}_4 \cdot \beta - y_4)^2\right).$$

Now let's design our stochastic gradient. Suppose that $\tilde{\boldsymbol{g}}$ is a random variable. With probability 1/4 it is equal to $\nabla(\boldsymbol{x}_1 \cdot \beta - y_1)^2 = 2\boldsymbol{x}_1(\boldsymbol{x}_1 \cdot \beta - y_1)$. With probability 1/4 it is equal to $\nabla(\boldsymbol{x}_2 \cdot \beta - y_2)^2 = 2\boldsymbol{x}_1(\boldsymbol{x}_2 \cdot \beta - y_2)$. With probability 1/4 it is equal to $\nabla(\boldsymbol{x}_3 \cdot \beta - y_3)^2 = 2\boldsymbol{x}_3(\boldsymbol{x}_3 \cdot \beta - y_3)$, and with probability 1/4 it is equal to $\nabla(\boldsymbol{x}_4 \cdot \beta - y_4)^2 = 2\boldsymbol{x}_4(\boldsymbol{x}_4 \cdot \beta - y_4)$.

b. Show that the expected value of $\tilde{\boldsymbol{g}}$ is equal to the full gradient.

Now back to the general case, where we have n data points. The SGD algorithm, therefore, is as follows: Initialize at some $\boldsymbol{\beta}_0$. Then at each step, to update from β_t to β_{t+1}, select an index $i \in \{1, \ldots, n\}$ uniformly at random (i.e., select a data point $(\boldsymbol{x}_i, y_i)$ uniformly at random from the given data), and update via

$$\beta_{t+1} = \beta_t - \eta \nabla [(\boldsymbol{x}_i \cdot \beta - y_i)^2].$$

Implement this algorithm, and plot its rate of convergence on a synthetic (randomly generated) regression problem. Specifically: generate a random $n \times p$ matrix X, with $n = 10,000$ and $p = 100$, and $X_{i,j}$ $N(0, 1)$. Select β^* to be the $p \times 1$ vector of all ones. Generate $\boldsymbol{y}$ by $\boldsymbol{y} = X\beta + w$, where w is the $n \times 1$ vector with $w_i \sim N(0, 0.1)$.

c. Solve the problem you created using gradient descent. How did you choose your step size?

d. Solve the problem using your implementation of the SGD algorithm. Again, take care in the choice of your step size.

You should see that gradient descent is much faster. So it is natural to ask why anyone would ever use SGD. The answer has several parts. First, as mentioned above, each step is extremely inexpensive, and requires looking at only a single piece of data. Where as in the problem above the entire data set can easily fit into memory, if you have a much larger problem, that may be impossible, and scanning through the entire data set for each iteration can be too expensive. Second, there are ways to accelerate SGD considerably (see below). Finally, SGD presents significant opportunities for parallelization, which on multi-core machines can again significantly speed up the computation.

2. (Bonus^2) Here is a paper (and there have been many followups) on ideas for parallelizing SGD for various problems in machine learning, and on how this would be implemented: `https://www.eecs.berkeley.edu/~brecht/papers/hogwildTR.pdf`