Exploring a Stochastic Quasi-Newton Method
Ahmed Bou-Rabee
EE364b: Convex Optimization II Class Project

Abstract

In this project we present an open-source Python implementation of a stochastic quasi-Newton method (SQN), test it, and discuss possible extensions.

Background

Stochastic optimization methods are useful when the optimization problem is large. For instance, steepest descent is computationally intensive as each iteration requires the evaluation of the gradient on the entire set of data. In stochastic gradient descent (SGD), we approximate the gradient by evaluating it on a small subset of the data, which is much less expensive. However, both of these ignore curvature:

\[ x^{k+1} = x^k - \alpha^k H_x^k \nabla F(x^k), \]

where \( \alpha^k \) is the step size, \( H_x^k \) is an approximation of the inverse Hessian matrix, and \( x^k \) is the optimization variable.

Code Demonstration

Here is a demonstration of running the code on a logistic regression problem where \( \omega \) is a random instance consisting of a data and label pair, reducing our minimization problem to

\[ \min \sum_{i=1}^{N} \ell(A_i, b_i, x). \]

where \( \ell \) is the logistic regression cost function and \( A \in \mathbb{R}^{p \times N} \) is a matrix of random data samples with labels \( b \in \{0, 1\}^N \).

We must hard-code three functions: the objective, gradient oracle, and then Hessian-vector oracle, to pass into the algorithm. These functions have a fixed standard input/output form, detailed in the code source. We form an “expression” object from these functions and the data.

\[ \text{exp} = \text{Expression(\text{logisticObjective}, \text{logisticGradient}, \text{logisticHessianVec}, \text{data, numFeatures})} \]

Then, we can form a problem from the expression and additional constraints, however, for this example there are no constraints so we just call

\[ \text{prob} = \text{Problem}(\text{exp}, \text{constraints} = []) \]

Using this problem object we can minimize the given expression. Right now we have stochastic gradient descent and stochastic quasi-Newton implemented. We can call the stochastic quasi-Newton solver by running the following:

\[ \text{prob = sqnsolve(K = 100, gradientBatchSize = 300, hessianBatchSize = 300)} \]

where \( x_{\text{final}} \) is the value of \( x \) at the optimal point and optval is the value of the objective at that point.

Convergence Rate

Here we test SQN on data taken from the HP spam database. The following graph compares the convergence rate of SGD with gradientBatchSize = 50 and SQN with gradientBatchSize = 10, hessianBatchSize = 300, M = 10, and L = 20. Note the poor convergence of SQN in this case.

Sensitivity to Parameter Choice

The convergence rate, iteration speed, and optimal value of SQN are heavily dependent upon its parameters. In the table below we compare an average of these values relative to those of the SGD of Figure 2.

<table>
<thead>
<tr>
<th>Grad B.S. Hess B.S. M L</th>
<th>Rel. Objective</th>
<th>NumIters</th>
<th>Rel. Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>300</td>
<td>10</td>
<td>20.0</td>
</tr>
<tr>
<td>30</td>
<td>300</td>
<td>10</td>
<td>20.1</td>
</tr>
<tr>
<td>30</td>
<td>300</td>
<td>20</td>
<td>20.15</td>
</tr>
<tr>
<td>30</td>
<td>300</td>
<td>100</td>
<td>20.15</td>
</tr>
<tr>
<td>30</td>
<td>300</td>
<td>10</td>
<td>20.15</td>
</tr>
<tr>
<td>30</td>
<td>300</td>
<td>100</td>
<td>20.15</td>
</tr>
<tr>
<td>30</td>
<td>300</td>
<td>10</td>
<td>20.15</td>
</tr>
<tr>
<td>30</td>
<td>300</td>
<td>100</td>
<td>20.15</td>
</tr>
<tr>
<td>30</td>
<td>300</td>
<td>10</td>
<td>20.15</td>
</tr>
<tr>
<td>30</td>
<td>300</td>
<td>100</td>
<td>20.15</td>
</tr>
<tr>
<td>30</td>
<td>300</td>
<td>10</td>
<td>20.15</td>
</tr>
<tr>
<td>30</td>
<td>300</td>
<td>100</td>
<td>20.15</td>
</tr>
</tbody>
</table>

Figure 3: Varying Parameters

Constrained Optimization

A natural way to extend SQN to allow constraints is to project the current estimate of the optimization variable onto the constraints in every iteration. There are existing randomized methods that increase the speed of this procedure when the number of constraints is large.

Limitations and Future Work

As SQN is very sensitive to parameter choice it is often very time-consuming to come up with the right choice of parameters for a particular problem. In some ways this defeats the purpose of having a software package, since non-experts will not be able to fully utilize SQN. One possible way to mitigate this is to automate parameter choices based on the input data.

Another weakness of the current implementation is that it is very cumbersome to hand-code the objective, gradient, and stochastic sub-gradient. Existing versions of stochastic gradient descent provide built in choices of loss functions, which makes the coding much easier, but lacks customizability. A compromise between the two would be to provide both a collection of built-in loss functions along with the ability to pass in custom functions.

Lastly, to make SQN robust, we must regularize.

Acknowledgements

I would like to thank Ernest Ryu for direction, mentorship, and helpful advice. I would also like to thank the authors of A Stochastic Quasi-Newton Method for Large-Scale Optimization for coming up with the algorithm. I am also grateful for helpful email exchanges with author Jorge Nocedal and his graduate student Stefan.