Distributed Optimization using ADMM

Zhenyuan Liu, 26968476

1 Introduction

Despite the superior performance of many state-of-the-art algorithms, solving large-scale optimization problems in serial is infeasible. Distributed optimization has thus become necessary in solving such problems. In other problems, many agents seek to minimize the sum of the objective functions collectively, and they only have access to local objective functions and decision variables. Distributed optimization is a natural way to solve such problems. In the current project, I did a survey on ADMM (Alternating Direction Method of Multipliers)[1] and did some computer experiments.

2 ADMM

2.1 Dual Ascent

In dual ascent, we solve the dual problem using gradient ascent. An important result regarding the sub-gradients of the dual of a closed and convex function is: If f is closed and convex, then:

$$y \in \partial f(x) \leftrightarrow x \in \partial f^*(y) \leftrightarrow x^T y = f(x) + f^*(y) \tag{1}$$

We use this conclusion to find the (sub-)gradient of the dual function of an equality-constrained convex optimization problem:

$$\min_x f(x) \quad s.t.\ Ax = b \tag{2}$$

The Lagragian is $L(x, y) = f(x) + y^T(Ax - b)$. It can be shown easily that the dual function is $g(y) = \inf_x L(x, y) = -f^*(-A^T y) - b^T y$. Assume that f and g are differentiable, if $x^+ \in \arg\min_x f(x) + y^T(Ax - b)$, then $\nabla f(x^+) + A^T y = 0$, i.e. $-A^T y = \nabla f(x^+)$, thus $x^+ = \nabla f^*(-A^T y)$, then $\nabla g(y) = A\nabla f^*(-A^T y) - b = Ax^+ - b$, following from equation (1). Finally, the updates of the dual ascent method are:

$$\begin{aligned} x^{k+1} &:= \arg\min_x L(x, y^k) \\ y^{k+1} &:= y^k + \alpha^k(Ax^{k+1} - b) \end{aligned} \tag{3}$$

2.2 Dual Decomposition

In cases when f is separable (e.g. by features), the optimization problem (2) can be written as:

$$\min_x \sum_{i=1}^n f_i(x_i) \quad s.t.\ \sum_{i=1}^n A_i x_i = b \tag{4}$$

where $x = (x_1, ..., x_n)$, and $A = [A_1 ... A_n]$. In these conditions, the Lagrangian is also decomposable, and the dual ascent updates in equation (3) can be written:

$$\begin{aligned} x_i^{k+1} &:= \arg\min_{x_i} f_i(x_i) + (y^k)^T(A_i x_i - b) \\ y^{k+1} &:= y^k + \alpha^k(Ax^{k+1} - b) \end{aligned} \tag{5}$$

2.3 Augmented Lagrangians and the Method of Multipliers

The augmented Lagrangian is the Lagrangian of the problem:

$$\min_x f(x) + (\rho/2)||Ax - b||_2^2 \;\; s.t. \; Ax = b \tag{6}$$

Problem (6) is obviously equivalent to problem (2), because any feasible x would make the quadratic term vanish. Problem (6) can bring robustness to the dual ascent method, in particular, to yield convergence without assumptions like strict convexity or finiteness of the objective function[1], since the quadratic function is strongly convex. In this case, we write the augmented Lagrangian $L_\rho(x, y) = f(x) + (y)^T(Ax - b) + (\rho/2)||Ax - b||_2^2$, and the dual function $g_\rho(y) := \inf_x L_\rho(x, y)$. Another benefit of the quadratic term is that $g_\rho(y)$ can be shown to be differentiable under rather mild conditions on the original problem[1]. Finally, the updates of the dual ascent method applied to the modified problem can be written:

$$\begin{aligned} x^{k+1} &:= \arg\min_x L_\rho(x, y^k) \\ y^{k+1} &:= y^k + \rho(Ax^{k+1} - b) \end{aligned} \tag{7}$$

This is called the method of multipliers. There are two differences between equation (7) and equation (3). First, the Lagragian $L(x, y^k)$ is replaced by the augmented Lagrangian $L_\rho(x, y^k)$. Second, the step size α^k is replaced by ρ. It can be shown easily that this particular choice of the step size α^k ensures that the iterates (x^{k+1}, y^{k+1}) satisfy the optimality condition[1].

2.4 Aternating Direction Method of Multipliers

ADMM blends the decomposability of dual ascent with the superior convergence properties of the method of multipliers[1]. The algorithm solves problem in the following form:

$$\min_x f(x) + g(z) \;\; s.t. \; Ax + Bz = c \tag{8}$$

Note that any constrained convex optimization problem can be rewritten in the ADMM form as in equation (8), shown as follows:

$$\min_x f(x) + g(z) \;\; s.t. \; x - z = 0 \tag{9}$$

where g is the indicator function of the convex set C. The augmented Lagrangian of problem (8) is $L_\rho(x, z, y) = f(x) + g(z) + (y)^T(Ax + Bz - c) + (\rho/2)||Ax + Bz - c||_2^2$. ADMM consists of the updates:

$$\begin{aligned} x^{k+1} &:= \arg\min_x L_\rho(x, z^k, y^k) \\ z^{k+1} &:= \arg\min_z L_\rho(x^{k+1}, z, y^k) \\ y^{k+1} &:= y^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \end{aligned} \tag{10}$$

In ADMM, x and z are updated in an alternating fashion, which accounts for the term alternating direction[1]. Define the residual $r = Ax + Bz - c$ and introduce the scaled dual variable $u = (1/\rho)y$, we can write the updates in equation (10) as

$$\begin{aligned} x^{k+1} &:= \arg\min_x f(x) + (\rho/2)||Ax + Bz^k - c + u^k||_2^2 \\ z^{k+1} &:= \arg\min_z g(z) + (\rho/2)||Ax^{k+1} + Bz - c + u^k||_2^2 \\ u^{k+1} &:= u^k + Ax^{k+1} + Bz^{k+1} - c \end{aligned} \tag{11}$$

2.5 Convergence of ADMM

In[1], a general result applicable to a broad range of optimization problems is given under two relatively mild assumptions.
Assumption 1: The (extended-real-valued) function $f : \mathbf{R}^n \rightarrow \mathbf{R} \cup \{+\infty\}$ and $g : \mathbf{R}^m \rightarrow \mathbf{R} \cup \{+\infty\}$ are closed, proper and convex.
Assumption 2: The augmented Lagrangian L_0 has a saddle point, i.e. strong duality holds and optimum attained.
Under assumptions 1 and 2, the ADMM iterates satisfy the following:

1. *Residual convergence.* $r^k \rightarrow 0$ as $k \rightarrow \infty$, i.e., the iterates approach feasibility.
2. *Obejective convergence.* $f(x^k) + g(z^k) \rightarrow p^*$ as $k \rightarrow \infty$, i.e., the objective function of the iterates approaches the optimal value.
3. *Dual variable convergence.* $y^k \rightarrow y^*$ as $k \rightarrow \infty$, where y^* is the dual optimal point.

Under assumptions 1 and 2, the necessary and sufficient optimality conditions for problem (8) are the KKT conditions, namely

$$Ax^* + Bz^* - c = 0 \tag{12}$$

$$0 \in \partial f(x^*) + A^T y^* \tag{13}$$

$$0 \in \partial g(z^*) + B^T y^* \tag{14}$$

For (z^k, y^k) generated by (10), the last condition is always satisfied[1]. Thus, obtaining optimality comes down to the first two conditions. The quantity $r^{k+1} = Ax^{k+1} + Bz^{k+1} - c$ is defined as the primal residual, and $s^{k+1} = \rho A^T B(z^{k+1} - z^k)$ is defined as the dual residual, since it can be shown that $\rho A^T B(z^{k+1} - z^k) \in \partial f(x^{k+1}) + A^T y^{k+1}$[1].
It is shown in[1] that the stopping criterion is: $||r^k||_2 \leq \epsilon^{pri}$ and $||s^k||_2 \leq \epsilon^{dual}$, where ϵ^{pri} and ϵ^{dual} are given by $\epsilon^{pri} = \sqrt{p}\epsilon^{abs} + \epsilon^{rel} \max\{||Ax^k||_2, ||Bz^k||_2, ||c||_2\}$, and $\epsilon^{dual} = \sqrt{n}\epsilon^{abs} + \epsilon^{rel}||A^T y^k||_2$, where p is the number of rows of A, n is the length of x, ϵ^{dual} is the absolute tolerance and ϵ^{rel} is the relative tolerance.

3 Computer Experiments

A python implementation of the ADMM algorithm[2] using mpi4py[3] was used to solve two different kinds of convex optimization problems using a node with 32 physical cores at 2.3 GHz. The code used was based on the MATLAB and C code of Boyd[1].

3.1 LASSO

In ADMM form, the LASSO can be written as:

$$\min \quad f(x) + g(z) \quad s.t. \quad x - z = 0$$

where $f(x) = (1/2)||Ax - b||_2^2$, and $g(z) = \lambda||z||_1$. A and b can be partitioned by samples(rows), leading to

$$\min \quad \sum_{i=1}^{N} f_i(x_i) + g(z) \quad s.t. \quad x_i - z = 0, \quad \forall i = 1, ..., N \tag{15}$$

where $f_i(x_i) = (1/2)||A_i x_i - b_i||_2^2$, and $g(z) = \lambda||z||_1$. Following from (11), we have the distributed algorithm:

$$\begin{aligned} x_i^{k+1} &:= \arg\min_{x_i} \; (1/2)||A_i x_i - b_i||_2^2 + (\rho/2)||x_i - z^k + u_i^k||_2^2 \\ z^{k+1} &:= S_{\lambda/\rho N}(\overline{x}^{k+1} + \overline{u}^k) \\ u_i^{k+1} &:= u_i^k + x_i^{k+1} - z^{k+1} \end{aligned} \tag{16}$$

where S is the soft-thresholding operator. The x-minimization step has a closed-form solution:

$$x_i^{k+1} = (A_i^T A_i + \rho I)^{-1}(A_i^T b_i + \rho(z^k - u_i^k))$$

this can be calculated easily without solving an optimization problem locally. Note that we can use the matrix lemma to factor $A_i A_i^T + \rho I$ if necessary. After the x-minimization step, x_i and u_i were collected by the master node to calculate $\overline{x}$ and $\overline{u}$, which were used to calculate z. The dual ascent update has a simple form.

A large dataset was used for the computer experiment. The dataset has $m = 128000$ samples, and $n = 10000$ features, so each of the 32 cores handles 4000 samples. The coefficient matrix is NOT sparse, so the dataset is about 10GB. ADMM can deal with even larger dataset, given that more resources are available. This problem requires a very large memory to be solved using a serial implementation. In the computer experiment, $\rho = 1$, $\epsilon^{abs} = 10^{-4}$, $\epsilon^{rel} = 10^{-2}$, u^0, z^0 and x^0 were initialized as zero vectors. These parameters are not optimized.

The result is plotted in figure 1. On the left panel, how the suboptimality changes vs. iteration is plotted. The optimal value p^* of the LASSO problem was 1168.81, found by another serial code using scikit-learn[4]. Objetive function value found by ADMM is 1169.20, indicating a suboptimality less than 0.4. On the right panel, how the primal residual $||r||_2$ and the dual residual $||s||_2$ changes vs. iteration is plotted, the dashed lines are ϵ^{pri} and ϵ^{dual}, defined in section 2.5. In the current experiment, ADMM converges in 129 steps. The results show that ADMM performs very well with the LASSO problem studied.

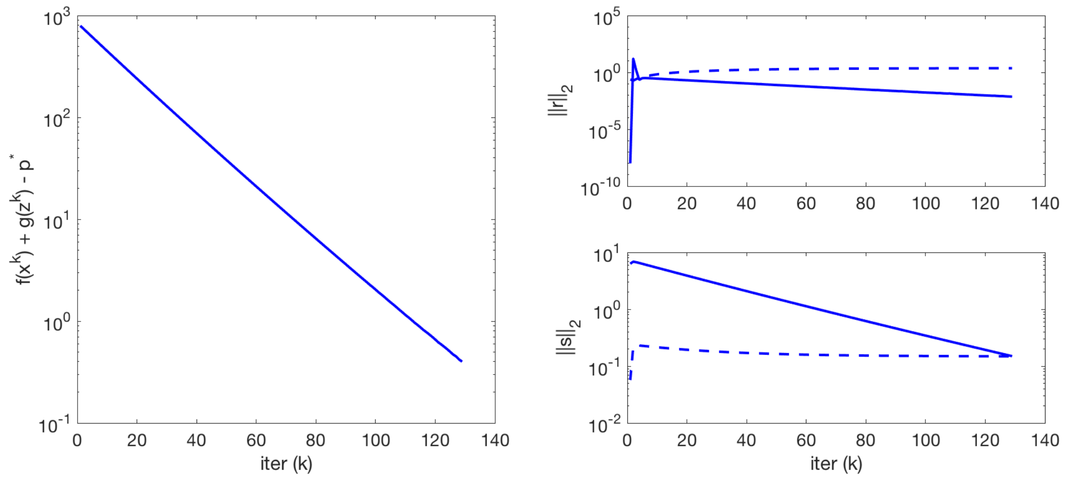

Figure 1: Objective value, norms of primal and dual residual vs. iteration for the LASSO problem

3.2 Soft-margin SVM

Consider the soft-margin SVM problem with tuning parameter $C = 1$:

$$\min \quad (1/2)||w||_2^2 + \sum_{i=1}^{m} \max(0,\ 1 - y_i(w^T x_i + b))$$

In distributed form, the ADMM algorithm have the following updates:

$$\begin{aligned} w_i^{k+1} &:= \underset{w_i}{\arg\min} \ \left(1^T(A_i w_i + 1)_+ + (\rho/2)||w_i - z^k + u_i^k||_2^2\right) \\ z^{k+1} &:= \frac{\rho}{(1/\lambda) + N\rho}(\overline{w}^{k+1} + \overline{u}^k) \\ u_i^{k+1} &:= u_i^k + w_i^{k+1} - z^{k+1} \end{aligned} \tag{17}$$

where A is a matrix whose row is given by $[y_j x_j^T - y_j]$, where x_j^T is the j-th row of the data matrix, N is the number of sub-processes, in our case $N = 32$. Unlike the LASSO problem, the x-minimization step doesn't have a closed-form solution and requires solving a non-trivial optimization problem locally. A closer look reveals that the x-minimization step is actually itself a SVM problem with an offset in the quadratic term. CVXPY[5] was used for the x-minimization step. After that, x_i and u_i were collected by the master node to calculate $\overline{x}$ and $\overline{u}$, which were used to calculate z. The dual ascent update has a simple form.
Solving a large-scale SVM problem is much more expensive than LASSO. So the dataset used is much smaller. The dataset has $m = 128000$ samples, and $n = 100$ features, again each of the 32 cores handles 4000 samples. The dataset is about 100MB. In the computer experiment, $\rho = 1$, $\epsilon^{abs} = 10^{-4}$, $\epsilon^{rel} = 10^{-3}$, and u^0, z^0 and x^0 were initialized as zero vectors.

The result is plotted in figure 2. On the left panel, how the suboptimality changes vs. iteration is plotted. The optimal value p^* of the same SVM problem was found by by another serial code using scikit-learn[4]. On the right panel, how the primal residual $||r||_2$ and the dual residual $||s||_2$ changes vs. iteration is plotted, the dashed lines are ϵ^{pri} and ϵ^{dual}, defined in section 2.5. In the

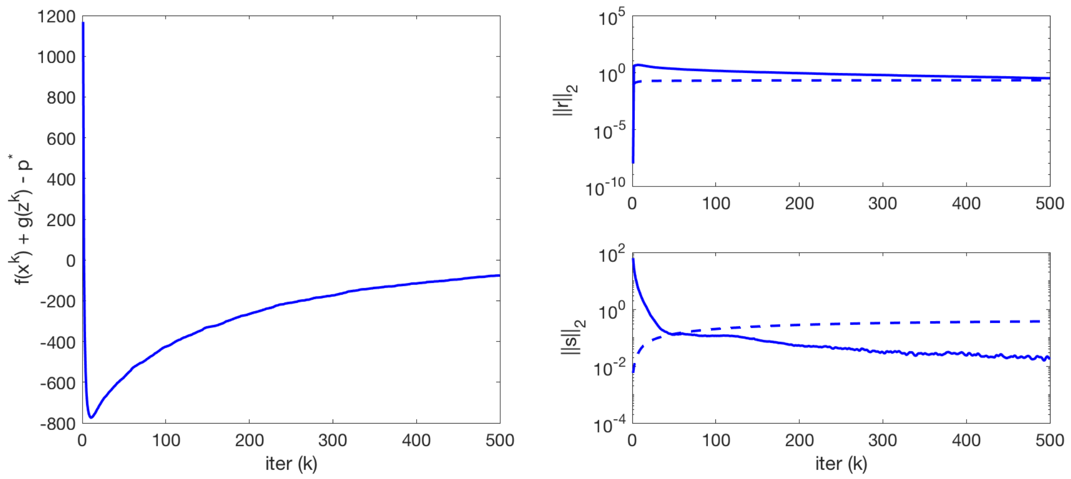

Figure 2: Objective value, norms of primal and dual residual vs. iteration for the SVM problem

current experiment, ADMM didn't converge after 500 steps. As shown by the plot, the norm of the primal residual $||r||_2$ decreases very slowly. After 500 iteration, the objective function value is 1150, whereas p^* found by Scikit-learn is 1225. More advanced techniques or special treatments are required to solve the SVM problem efficiently using ADMM.

4 Conclusions

ADMM can solve very large-scale problem in a distributed way. Computer experiments were conducted using ADMM to solve large-scale LASSO and SVM problem. The LASSO problem solved has a coefficient matrix of 10GB. The distributed ADMM worked very well in solving the LASSO problem, taking less than 130 steps to bring the sub-optimality down to less than 0.4 (p^* is in the order of 1000). For a problem with a coefficient matrix of 10GB, it's possible to use a serial code like the one in Scikit-learn with a high-end computer, for purpose of comparing results, however, it's almost impossible to solve a problem with a coefficient matrix of 100GB, 1TB or even larger using a serial code. Thus distributed ADMM would be very helpful in solving very large-scale LASSO problems. As for the SVM problem with a coefficient matrix about 100MB, it took a much longer time because it needs to solve non-trivial convex optimization problem locally in each iteration. Moreover, the distributed ADMM didn't converge after 500 iterations in the computer experiment. More advanced techniques and special treatments are required to solve SVM in large-scale more efficiently.

In the computer experiments, parameter values and update schemes were not optimized and were selected as suggested by[1]. An exciting future work be to explore how changing ρ, using different augmenting functions, using over/under relaxation in the z/y update steps, using warm-start etc affect the convergence of ADMM.

Overall, it has been a very interesting project for me, it strengthened my understanding of the relationship between the (sub-)gradient of the primal and that of the dual, the conjugate dual, the Lagrange and the augmented Lagrangian, KKT conditions, convergence analysis of the ADMM algorithm, soft-margin SVM, uses of Scikit-learn[4], CVXPY[5] and mpi4py[3].

References

[1] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[2] https://github.com/zhenyuan666/admm.

[3] http://mpi4py.scipy.org/docs/.

[4] http://scikit-learn.org/stable/.

[5] http://www.cvxpy.org/en/latest/.