Block Splitting for Large-Scale Distributed Learning

Neal Parikh
Department of Computer Science
Stanford University
Stanford, CA 94305
npparikh@cs.stanford.edu

Stephen Boyd
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
boyd@stanford.edu

Abstract

Machine learning and statistics with very large datasets is now a topic of widespread interest, both in academia and industry. Many such tasks can be posed as convex optimization problems, so algorithms for distributed convex optimization serve as a powerful, general-purpose mechanism for training a wide class of models on datasets too large to process on a single machine. In previous work, it has been shown how to solve such problems in such a way that each machine only looks at either a subset of training examples or a subset of features. In this paper, we extend these algorithms by showing how to split problems by both examples and features simultaneously, which is necessary to deal with datasets that are very large in both dimensions. We present some experiments with these algorithms run on Amazon's Elastic Compute Cloud.

1 Introduction

Many modern datasets now share three key characteristics: they contain a large number of training examples, they contain a large number of features, and they are stored (and in some cases even collected) in a distributed or decentralized fashion. Training machine learning models in this regime requires the use of algorithms amenable to distributed computation, and ideally, one should be able to fit a global model over such distributed datasets without having to transfer any portions of the dataset over the network. Since many such problems reduce to solving a convex optimization problem to estimate model parameters from data, the distributed optimization algorithm we present here can be used to train a wide variety of machine learning models in a distributed setting.

While there has been work on training models in the presence of either a large number of examples or a large number of features, there has been no general purpose method for training models on datasets that are large (and possibly dense) in both dimensions, to the best of our knowledge. We present such an algorithm here. We first discuss theoretical motivation for the method, then the algorithm and some implementation details, and finally some initial experiments.

Related work. This paper directly builds on the recent paper by Boyd et al. [2], which in turn draws from, among other areas, the literature on proximal methods [13, 15, 16], monotone operator splitting methods [5, 12], (dual) decomposition [4, 7], and augmented Lagrangians [10, 9, 15, 8]. For some examples of recent related work in the machine learning literature, see, *e.g.*, [6, 17, 11]. We omit a detailed review of related work for space reasons.

2 Problem statement

Let $\mathcal{D}$ be a dataset consisting of m examples each with n features, collected in a feature matrix $A \in \mathbf{R}^{m \times n}$. For simplicity of notation in the sequel, we consider the responses, or outputs, as being included as a column in A. Many learning problems can then be posed as (regularized) loss minimization problems of the form

$$\begin{array}{ll} \text{minimize} & l(y) + r(x) \\ \text{subject to} & y = Ax \end{array} \tag{1}$$

with variables $x \in \mathbf{R}^n$ (the model parameters) and $y \in \mathbf{R}^m$, convex loss function $l : \mathbf{R}^m \to \mathbf{R} \cup \{+\infty\}$, and convex regularization function $r : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$. (We assume that the component of x corresponding to the responses is fixed as -1.) The reason for expressing the problem in this particular way (with the auxiliary variable y) is to make it more amenable to decomposition, for reasons that will become more clear later.

For example, if $l = (1/2)\|\cdot\|_2^2$ and $r = \lambda\|\cdot\|_1$, this is exactly the lasso [18], and if l is hinge loss and $r = \lambda\|\cdot\|_2^2$, this is the primal form of a support vector machine. Generalized linear and additive models also follow this form, in addition to a variety of other model families. The regularization function r can take the form of a squared ℓ_2 penalty (ridge or Tikhonov regularization), ℓ_1, sum-of-norms or group lasso [14, 20], and many other more sophisticated variants (*e.g.*, [21]). Because l and r can be extended-real-valued, they can also encode constraints [3, §3.1.2].

We partition A into MN blocks $A_{ij} \in \mathbf{R}^{m_i \times n_j}$ and x conformably into N subvectors $x_j \in \mathbf{R}^{n_j}$, so $\sum_{i=1}^M m_i = m$ and $\sum_{j=1}^N n_j = n$. Thus $y = Ax$ can be expressed as $y_i = \sum_{j=1}^N A_{ij}x_j$ for $i = 1, \ldots, M$. We assume that l and r are *block separable, i.e.*, $l(y) = \sum_{i=1}^M l_i(y_i)$ and $r(x) = \sum_{j=1}^N r_j(x_j)$. As a convention, i will index block rows and j will index block columns.

The goal is to solve this problem in a way that (a) allows for each block A_{ij} to be handled by a separate processor or machine, and (b) does not involve transfer of the A_{ij} over the network.

3 Operator splitting

The algorithm we use is an operator splitting method called the *alternating direction method of multipliers* (ADMM), or equivalently, *Douglas-Rachford splitting*. The algorithm itself is discussed in detail in [2], so we omit background here for space reasons; for general background on monotone operator theory and operator splitting, see [1].

For the generic constrained convex problem

$$\begin{array}{ll} \text{minimize} & f(z) \\ \text{subject to} & z \in \mathcal{C}, \end{array}$$

where $\mathcal{C}$ is convex, one form of the algorithm is, for $k = 0, 1, 2, \ldots$,

$$\begin{aligned} z^{k+1/2} &:= \mathbf{prox}_f(z^k - \tilde{z}^k) \\ z^{k+1} &:= \Pi_{\mathcal{C}}(z^{k+1/2} + \tilde{z}^k) \\ \tilde{z}^{k+1} &:= \tilde{z}^k + z^{k+1/2} - z^{k+1}. \end{aligned}$$

Here k is an iteration counter, $\Pi_{\mathcal{C}}$ denotes (Euclidean) projection onto $\mathcal{C}$, and

$$\mathbf{prox}_f(v) = \operatorname*{argmin}_x \left(f(x) + (\rho/2)\|x - v\|_2^2\right)$$

is called the *proximal operator* of f with parameter $\rho > 0$ [13]. Some of the proximal operators that will arise here have simple closed form expressions; for instance, the operators

$$S_{\lambda/\rho}(v) = (v - \lambda/\rho)_+ - (-v - \lambda/\rho)_+, \qquad T_\rho(v) = \left(\frac{\rho}{1+\rho}\right) v,$$

are the proximal operators of $\lambda\|\cdot\|_1$ (here, $(\cdot)_+ = \max(0, \cdot)$) and $(1/2)\|\cdot\|_2^2$, respectively. Note that both of these proximal operators are very efficient to evaluate. The operator $S_{\lambda/\rho}$ is known as *soft thresholding*.

4 Block splitting algorithm

The final algorithm is obtained by carrying out an appropriate problem transformation of (1), applying ADMM to it, and simplifying. For brevity, we show the problem transformation and then present the simplified algorithm directly.

Problem transformation. Introducing MN new variables $x_{ij} \in \mathbf{R}^{n_j}$ and $y_{ij} \in \mathbf{R}^{m_i}$, the problem becomes

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^M l_i(y_i) + \sum_{j=1}^N r_j(x_j) & \\ \text{subject to} & x_{ij} = x_j, & i = 1, \ldots, M \\ & y_i = \sum_{j=1}^N y_{ij}, & i = 1, \ldots, M \\ & y_{ij} = A_{ij}x_{ij}, & i = 1, \ldots, M, \quad j = 1, \ldots, N, \end{array}$$

with variables x_j, y_i, x_{ij}, and y_{ij}. Here, x_{ij} can be viewed as the 'local opinion' of the value of x_j only using data block A_{ij}, and y_{ij} can be viewed as the 'partial' responses only using the local estimate x_{ij} and local data A_{ij}.

We now move the partial response constraints $y_{ij} = A_{ij}x_{ij}$ into the objective, writing the problem as

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^M l_i(y_i) + \sum_{j=1}^N r_j(x_j) + \sum_{i=1}^M \sum_{j=1}^N I_{ij}(y_{ij}, x_{ij}) \\ \text{subject to} & x_{ij} = x_j, \quad i = 1, \ldots, M \\ & y_i = \sum_{j=1}^M y_{ij}, \quad i = 1, \ldots, M, \end{array} \tag{2}$$

where I_{ij} is the indicator function [3, §3.1.2] of the convex set $\mathcal{C}_{ij} = \{(y_{ij}, x_{ij}) \mid y_{ij} = A_{ij}x_{ij}\}$. The three objective terms now involve distinct sets of variables, and the two sets of constraints involve distinct sets of variables, which will simplify a number of computations in the sequel.

Algorithm. Applying the form of ADMM above to (2) and simplifying yields the algorithm

$$y_i^{k+1/2} := \mathbf{prox}_{l_i}(y_i^k - \tilde{y}_i^k) \tag{3}$$

$$x_j^{k+1/2} := \mathbf{prox}_{r_j}(x_j^k - \tilde{x}_j^k) \tag{4}$$

$$(y_{ij}^{k+1/2}, x_{ij}^{k+1/2}) := \Pi_{ij}(y_{ij}^k + \tilde{y}_i^k, x_j^k - \tilde{x}_{ij}^k) \tag{5}$$

$$x_j^{k+1} := \mathbf{avg}(x_j^{k+1/2}, \{x_{ij}^{k+1/2}\}_{i=1}^M) \tag{6}$$

$$(y_i^{k+1}, \{y_{ij}^{k+1}\}_{j=1}^N) := \mathbf{exch}(y_i^{k+1/2}, \{y_{ij}^{k+1/2}\}_{j=1}^N) \tag{7}$$

$$\tilde{x}_j^{k+1} := \tilde{x}_j^k + x_j^{k+1/2} - x_j^{k+1} \tag{8}$$

$$\tilde{y}_i^{k+1} := \tilde{y}_i^k + y_i^{k+1/2} - y_i^{k+1} \tag{9}$$

$$\tilde{x}_{ij}^{k+1} := \tilde{x}_{ij}^k + x_{ij}^{k+1/2} - x_j^{k+1}. \tag{10}$$

Here, Π_{ij} is projection onto $\mathcal{C}_{ij}$, the averaging operator $\mathbf{avg}$ is given by

$$\mathbf{avg}(c, c_1, \ldots, c_M) = \frac{1}{M+1}\left(c + \sum_{i=1}^M c_i\right),$$

and the exchange projection $\mathbf{exch}(c, \{c_j\}_{j=1}^N)$ is given by

$$y_{ij}^{k+1} := c_j + \frac{1}{N+1}\left(c - \sum_{j=1}^N c_j\right), \qquad y_i^{k+1} := c - \frac{1}{N+1}\left(c - \sum_{j=1}^N c_j\right).$$

This can be viewed as a kind of de-meaning operation. We call this an exchange projection since it enforces a constraint very similar to the one in the optimal exchange problem described in [2, §7.3.2].

The iterate $(x_1^{k+1/2}, \ldots, x_N^{k+1/2})$ converges to optimality (see [2, §3.3] for details).

5 Implementation

Computing Π_{ij}. Explicitly, the projection $\Pi_{ij}(c, d)$ is given by

$$\begin{aligned} y_{ij}^{k+1/2} &:= (I + A_{ij}A_{ij}^T)^{-1}(A_{ij}d + A_{ij}A_{ij}^T c) \\ x_{ij}^{k+1/2} &:= d + A_{ij}^T(c - y_{ij}^{k+1/2}). \end{aligned}$$

(The $y_{ij}^{k+1/2}$ update can be adjusted in a straightforward manner for the block that contains the column of responses.)

Thus the work reduces to solving MN linear systems (in parallel) with coefficient matrices $I + A_{ij}A_{ij}^T$, assumed to be of modest size. The key fact we can exploit is that these matrices stay fixed throughout the runtime of the algorithm. This means, for instance, that we can form $I + A_{ij}A_{ij}^T$ and its Cholesky factorization $L_{ij}L_{ij}^T$ (assuming A_{ij} is dense) once and cache them, then simply reuse these in all subsequent iterations, giving a very large speed improvement. There are a number of ways of solving linear systems of this form, some of which can be superior to the simple method described here, but we omit a further discussion for space reasons. (In particular, an iterative method like conjugate gradient or LSQR is likely best when A is sparse.) See [2, §4.2–4.3] for related material.

Parallelism. Several of the steps in the algorithm be carried out independently in parallel. The first three steps can all be performed in parallel: Each of the M $y_i^{k+1/2}$'s, the N $x_j^{k+1/2}$'s, and the MN ($x_{ij}^{k+1/2}$, $y_{ij}^{k+1/2}$) pairs can all be updated separately. Similarly, each of the N averaging and M exchange operations can be carried out independently in parallel, and the final three steps can also be carried out independently in parallel. Overall, the algorithm thus involves three distinct stages. Intuitively, the **avg** and **exch** operations are the mechanism by which the local variables coordinate to move towards the global solution.

Communication. We use a total of MN processes split across some number of machines. For example, we may want each process on a separate machine, in which case MN machines are required; alternatively, each process may use a distinct core, in which case multiple processes would run on the same machine. Only the averaging and exchange steps require communication. The averaging step communicates within each block column (but not across columns), and the exchange step communicates within block rows (but not across rows). The main collaborative component in computing both is summing a set of vectors, so both can be implemented via *reduce* operations in parallel programming frameworks like MPI; averaging involves N reduce steps run in parallel and exchange involves M reduce steps.

Separating data from problems. Note that the data A_{ij} and the objective terms l_i and r_j never appear together; the objective terms appear in their own proximal step and the data only appears in the Π_{ij} projection step. This implies that if one wishes to train multiple models on the same dataset, the fitting problems can be solved simultaneously, and a large amount of computational work can be shared and reused across the problems (such as the cached factorizations mentioned above). Roughly speaking, one can fit multiple models for the price of one. Obtaining an algorithm for a specific problem only requires working out the proximal operators of l_i and r_j. Though this is not the main use case of the algorithm, it is an interesting and unusual side benefit.

6 Experiments

We solve three instances of the lasso (*i.e.*, sparse regression), which involves solving

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda\|x\|_1,$$

where A is the feature matrix and b is the vector of outputs or responses. In the form (1), this involves setting $l_i = (1/2)\|\cdot\|_2^2$ and $r_j = \lambda\|\cdot\|_1$, so the proximal operators for l_i and r_j are T_ρ and $S_{\lambda/\rho}$, respectively, as described in §3. These operators are so efficient to evaluate (both are simple elementwise vector operations) that the bulk of the work is in computing Π_{ij}; this is often the case for many other problems as well.

To get a reasonably large problem, we generate synthetic data in a similar manner as [2, §11.1]. We implemented the algorithm above as written (with factorization caching) in C using MPI and the GNU Scientific Library (linked against ATLAS [19]); the computations were done on Amazon EC2. Below, the 'factorization step' refers to forming $A_{ij}A_{ij}^T$ and factoring $I + A_{ij}A_{ij}^T$ once, and the 'main loop' refers to all the iterations after the factorization has been cached.

We take the A_{ij} to be dense 3000×5000 blocks and then set M and N as needed to produce problems at different scales. For example, if $M = 4$ and $N = 2$, the total A matrix is 12000×10000 and contains 120 million nonzero entries. We solved such a problem in parallel on a single 8-core machine; the factorization step took around 15 seconds and the main loop (90 iterations) took around 10 seconds, yielding a total solve time of 28 seconds. Since each iteration takes only 0.10–0.15 seconds (including all communication), running for more iterations costs little in total runtime.

Second, we solved a problem with $M = 8$ and $N = 5$ (600 million nonzero entries in total) across 5 machines (40 cores total). The Cholesky factorization step still takes around 15 seconds, since it is carried out in parallel. The main loop ran for 230 iterations, taking around 27 seconds, yielding a total solve time of 50 seconds.

Finally, we solved a problem with $M = 8$ and $N = 10$ (1.2 billion nonzero entries and over 10 GB of data) across 10 machines (80 cores total). Again, the only difference was that the main loop ran for 490 iterations, taking around 60 seconds, yielding a total solve time of 80 seconds. Note that each iteration still takes only 0.05–0.15 seconds, despite the larger problem size. In general, larger problems do not necessarily require more iterations to solve, so in some cases, it is possible that larger problems can be solved with no increase in total solve time.

Acknowledgements

We thank Eric Chu and Yang Wang for helpful discussions.

References

[1] H. H. Bauschke and P. L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer-Verlag, 2011.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[4] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[5] J. Douglas and H. H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American Mathematical Society*, 82:421–439, 1956.

[6] J. C. Duchi and Y. Singer. Efficient learning using forward-backward splitting. *Advances in Neural Information Processing Systems*, 22:495–503, 2009.

[7] H. Everett. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, 11(3):399–417, 1963.

[8] M. Fortin and R. Glowinski. *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems*. North-Holland: Amsterdam, 1983.

[9] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximations. *Computers and Mathematics with Applications*, 2:17–40, 1976.

[10] R. Glowinski and A. Marrocco. Sur l'approximation, par elements finis d'ordre un, et la resolution, par penalisation-dualité, d'une classe de problems de Dirichlet non lineares. *Revue Française d'Automatique, Informatique, et Recherche Opérationelle*, 9:41–76, 1975.

[11] J. Langford, A. Strehl, and L. Li. Vowpal Wabbit, 2007. `https://github.com/JohnLangford/vowpal_wabbit`.

[12] P. L. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16:964–979, 1979.

[13] J.-J. Moreau. Fonctions convexes duales et points proximaux dans un espace Hilbertien. *Reports of the Paris Academy of Sciences, Series A*, 255:2897–2899, 1962.

[14] H. Ohlsson, L. Ljung, and S. Boyd. Segmentation of ARX-models using sum-of-norms regularization. *Automatica*, 46(6):1107–1111, 2010.

[15] R. T. Rockafellar. Augmented Lagrangians and applications of the proximal point algorithm in convex programming. *Mathematics of Operations Research*, 1:97–116, 1976.

[16] R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14:877, 1976.

[17] C. H. Teo, S. V. N. Vishwanathan, A. J. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 11:311–365, 2010.

[18] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58(1):267–288, 1996.

[19] R. C. Whaley and J. J. Dongarra. Automatically tuned linear algebra software. In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing (CDROM)*, pages 1–27, 1998.

[20] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[21] P. Zhao, G. Rocha, and B. Yu. The composite absolute penalties family for grouped and hierarchical variable selection. *Annals of Statistics*, 37(6A):3468–3497, 2009.