Projection onto the capped simplex

Weiran Wang
Toyota Technological Institute at Chicago
weiranwang@ttic.edu

Canyi Lu
National University of Singapore
canyilu@gmail.com

March 2, 2015

Abstract

We provide a simple and efficient algorithm for computing the Euclidean projection of a point onto the capped simplex, formally defined as

$$\min_{\mathbf{x}\in\mathbb{R}^D} \quad \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2 \qquad \text{s.t.} \quad \mathbf{x}^\top\mathbf{1}=s, \quad \mathbf{0}\le\mathbf{x}\le\mathbf{1},$$

together with an elementary proof. Both the MATLAB and `C++` implementations of the proposed algorithm can be downloaded at `https://eng.ucmerced.edu/people/wwang5`.

1 The Problem

In this report, we consider the following optimization problem

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^D} \quad & \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2 && \text{(1a)}\\ \text{s.t.} \quad & \mathbf{x}^\top\mathbf{1}=s && \text{(1b)}\\ & \mathbf{0}\le\mathbf{x}\le\mathbf{1}, && \text{(1c)} \end{aligned}$$

where $s\in[0,D]$ is a parameter of the problem, $\mathbf{0}$ and $\mathbf{1}$ are vectors of 0's and 1's respectively, and $\le$ means elementwise comparison. The feasible set of this problem is the intersection of the unit cube and a hyperplane with normal $\mathbf{1}$. Alternatively, the feasible set is the simplex $\{\mathbf{x}: \ \mathbf{x}\ge\mathbf{0},\ \mathbf{x}^\top\mathbf{1}=s\}$ with an additional capping constraints $\mathbf{x}\le\mathbf{1}$, so we call it the *capped simplex*. Problem 1 is a quadratic program and the objective function is strictly convex, so there is a unique solution which we denote by $\mathbf{x}=[x_1,\dots,x_D]^\top$ with a slight abuse of notation.

Remark 1.1. This problem is a slight generalization of the projection onto the probability simplex (see Duchi et al., 2008; Wang and Carreira-Perpiñán, 2013 and the references therein), which is a special case of (1) by setting $s=1$ and can be solved exactly with $\mathcal{O}(D\log D)$ time complexity. An elementary proof of the corresponding algorithm can be found in Wang and Carreira-Perpiñán (2013) and the cost mainly comes from sorting the dimensions of $\mathbf{y}$. Our solution to (1) in this report is derived using a similar idea.

Remark 1.2. The constraint $\mathbf{0}\le\mathbf{x}\le\mathbf{1}$ in (1) can be generalized to $\mathbf{0}\le\mathbf{x}\le t\mathbf{1}$ where t is any positive number. We only need to solve the following instance of (1):

$$\min_{\hat{\mathbf{x}}} \ \frac{1}{2}\|\hat{\mathbf{x}}-\mathbf{y}/t\|^2, \quad \text{s.t.} \quad \hat{\mathbf{x}}^\top\mathbf{1}=s/t, \quad \mathbf{0}\le\hat{\mathbf{x}}\le\mathbf{1},$$

and then scale its solution $\hat{\mathbf{x}}$ by t to obtain the solution of the original problem.

2 The algorithm

We provide an $\mathcal{O}(D^2)$ algorithm for solving (1) in Algorithm 1.

Algorithm 1 Euclidean projection of a vector onto the section of cube.

```
Input: $\mathbf{y} \in \mathbb{R}^D$ is sorted in ascending order: $y_1 \le y_2 \le \dots \le y_D$.
 1: Set $y_0 = -\infty$ and $y_{D+1} = \infty$, compute partial sums $T_0 = 0$, and $T_k = \sum_{j=1}^{k} y_k$, $k = 1, \dots, D$.
 2: for $a = 0, 1, \dots, D$ do
 3:   if $(s == D - a)$ && $(y_{a+1} - y_a \ge 1)$ then
 4:     Set b=a.
 5:     break
 6:   end if
 7:   for $b = a + 1, \dots, D$ do
 8:     Compute $\gamma = \frac{s+b-n+T_a-T_b}{b-a}$.
 9:     if $(y_a + \gamma \le 0)$ && $(y_{a+1} + \gamma > 0)$ && $(y_b + \gamma < 1)$ && $(y_{b+1} \ge 1)$ then
10:       break
11:     end if
12:   end for
13: end for
Output: $\mathbf{x} = [0, \dots, 0, y_{a+1} + \gamma, \dots, y_b + \gamma, 1, \dots, 1]$.
```

3 The proof

As mentioned earlier, (1) has a unique solution which is characterized by its KKT system (Nocedal and Wright, 2006). The Lagrangian function of the problem is

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \gamma) = \frac{1}{2} \|\mathbf{x} - \mathbf{y}\|^2 - \boldsymbol{\alpha}^\top \mathbf{x} - \boldsymbol{\beta}^\top (\mathbf{1} - \mathbf{x}) - \gamma(\mathbf{1}^\top \mathbf{x} - s)$$

where $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_D]^\top$ and $\boldsymbol{\beta} = [\beta_1, \dots, \beta_D]^\top$ are the Lagrange multipliers for the inequality constraints $\mathbf{x} \ge \mathbf{0}$ and $\mathbf{1} - \mathbf{x} \ge \mathbf{0}$ respectively, and γ is the Lagrange multiplier for the equality constraint. At the optimal solution $\mathbf{x}$ the following KKT conditions hold:

$$x_i - y_i - \alpha_i + \beta_i - \gamma = 0, \qquad i = 1, \dots, D \tag{2a}$$
$$x_i \ge 0, \qquad i = 1, \dots, D \tag{2b}$$
$$x_i \le 1, \qquad i = 1, \dots, D \tag{2c}$$
$$\alpha_i \ge 0, \qquad i = 1, \dots, D \tag{2d}$$
$$\beta_i \ge 0, \qquad i = 1, \dots, D \tag{2e}$$
$$\sum_{i=1}^{D} x_i = s, \tag{2f}$$
$$\alpha_i x_i = 0, \qquad i = 1, \dots, D \tag{2g}$$
$$\beta_i(1 - x_i) = 0, \qquad i = 1, \dots, D, \tag{2h}$$

where (2g) and (2g) are complementary slackness (CS) conditions.

Without loss of generality, we assume the components of the optimal solution $\mathbf{x}$ are in ascending order:

$$0 = x_1 = \dots = x_a < x_{a+1} \le \dots \le x_b < x_{b+1} = \dots x_D = 1, \tag{3}$$

where a is the number of 0's in the solution while $D - b$ is the number of 1's in the solution. The valid ranges for (a, b) are $0 \le a \le D$ and $a \le b \le D$. The KKT conditions can be simplified for different set of dimensions of the solution:

(i) For $i = 1, \dots, a$, the CS condition (2h) indicates $\beta_i = 0$, and thus

$$0 = x_i = y_i + \alpha_i + \gamma \ge y_i + \gamma, \tag{4}$$

where the last inequality uses the fact that $\alpha_i \ge 0$.

(ii) For $j = b+1, \dots, D$, the CS condition (2g) indicates $\alpha_j = 0$, and thus

$$1 = x_j = y_j - \beta_j + \gamma \leq y_j + \gamma, \tag{5}$$

where the last inequality uses the fact that $\beta_j \geq 0$.

(iii) For $k = a+1, \dots, b$, the CS conditions indicate $\alpha_k = \beta_k = 0$, and thus

$$0 < x_k = y_k + \gamma < 1. \tag{6}$$

It is then clear that for any $1 \leq i \leq a$, $b+1 \leq j \leq D$ and $a+1 \leq k \leq b$, we have

$$y_i \leq -\gamma < y_k < 1 - \gamma \leq y_j. \tag{7}$$

In other words, if the dimensions of $\mathbf{y}$ are sorted in ascending order, the corresponding dimensions of the solution $\mathbf{x}$ is also in ascending order. Therefore, the first step of our algorithm is to sort dimensions of $\mathbf{y}$ into ascending order. And all that is left is to find (a, b), the partition of $\mathbf{x}$ into the three segments. The only KKT condition we have not used so far is the sum constraint (2f), which now reduces to

$$\sum_{i=1}^{D} x_i = a \cdot 0 + \sum_{k=a+1}^{b} (y_k + \gamma) + (n - b) \cdot 1 = s. \tag{8}$$

This means that if we know (a, b) for the solution $\mathbf{x}$, we must have

$$\gamma = \frac{s + b - n - \sum_{k=a+1}^{b} y_k}{b - a}. \tag{9}$$

Since there are only $\frac{(D+1)(D+2)}{2}$ possible combinations for the indices (a, b), we could test each combination and compute the hypothesized γ value using (9). With the hypothesized γ, the tests we need for (a, b, γ) to produce the optimal $\mathbf{x}$ are the following:

$$y_a + \gamma \leq 0, \quad y_{a+1} + \gamma > 0, \quad y_b + \gamma < 1, \quad y_{b+1} + \gamma \geq 1. \tag{10}$$

It is easy to verify that the (a, b, γ) combination that passes the above test leads to $(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \gamma)$ that satisfy all the KKT conditions, where $\{\alpha_i\}_{i=1}^{a}$ and $\{\beta_j\}_{j=b+1}^{D}$ can be retrieved using (4) and (5) respectively.

Remark 3.1. A special case is when $a = b$, for which (9) is ill-defined. In this case, the optimal solution consists of a 0's and $n - a$ 1's. For this to happen, we must have $s = D - a$ and then (10) reduces to $y_{a+1} + \gamma \geq 1$ and $y_a + \gamma \leq 0$, and so $y_{a+1} - y_a \geq 1$.

Remark 3.2. The reason why the computational complexity of our algorithm is $\mathcal{O}(D^2)$ for (1) as opposed to $\mathcal{O}(D \log D)$ for projection onto probability simplex is due to the constraint $\mathbf{x} \leq \mathbf{1}$. This constraint is automatically satisfied for the projection onto probability simplex problem where $s = 1$, in which case we only need to figure out a—the number of zero dimensions in the solution.

Remark 3.3. In view of the previous remark, another way of solving (1) is to alternatively project the estimate onto the (scaled) probability simplex $\{\mathbf{x} : \mathbf{x} \geq \mathbf{0}, \mathbf{x}^\top \mathbf{1} = s\}$ and the set $\{\mathbf{x} : \mathbf{x} \leq \mathbf{1}\}$ for which the projection is trivial to compute (we simply threshold the dimensions that are greater than 1 to 1). And yet another approach is to apply the Alternating Direction Method of Multipliers (Boyd et al., 2011), which introduces another copy of the variables $\mathbf{x}$ and alternately optimize each copy with simple steps while encouraging the two copies to agree. We note that these approaches are iterative and the number of iterations depends on the desired accuracy. On the contrary, our method finds the exact solution within a fixed number of steps.

4 Experiment

In this section, we demonstrate the effectiveness of our proposed method in Algorithm 1 for solving (1). We compare our method with another two solvers. The first one is the CVX package (Grant and Boyd,

Table 1: Running time (in seconds) of different solvers for various sizes of $\mathbf{y}$.

Methods	$D = 50$	100	500	1000	2000	5000	10000	20000	100000
`lsqlin`	0.049	0.11	8.18	59.10	360.20	5078.00	-	-	-
CVX	0.73	1.01	5.03	7.41	14.55	40.94	-	-	-
Ours - MATLAB	0.0005	0.002	0.023	0.083	0.44	2.02	11.30	27.09	870.39
Ours - `C++`	**0.00002**	**0.00003**	**0.0003**	**0.0009**	**0.005**	**0.021**	**0.11**	**0.27**	**8.78**

2012), a general convex program solver which transforms the problem into a semi-definite program and then applies interior point method. And the second one is the MATLAB command `lsqlin` for solving constrained linear least squares. We implement Algorithm 1 in both MATLAB and `C++` and conduct all experiments in MATLAB (the `C++` code is compiled within MATLAB and the mex-file is used).

All experiments are run on a PC with an Intel Core 2 Quad CPU Q9550 of frequency 2.83GH and 8GB main memory, under Windows 7 and MATLAB version 8.0. We generate $\mathbf{y}$ and s using the MATLAB commands

```
y=rand(D,1)-0.5;
s=round(rand*D);
```

and record the running time of each method using `tic` and `toc`. We choose the dimension D of $\mathbf{y}$ from $\{50, 100, 500, 1000, 2000, 5000, 10000, 20000, 100000\}$. For each choice of D, the experiments are repeated 20 times and the average running times are reported for comparison.

The results are shown in Table 1. It can be seen that our proposed method is always faster than CVX and `lsqlin` for different dimensions D of $\mathbf{y}$. And as D increases, the improvement of our method over the others becomes more significant. If D is relatively large, the compared solvers may run out of memory (denoted as '$-$' in Table 1). These results confirm that it is beneficiary to explore the special structures of our problem rather than using general convex program solvers. Furthermore, the `C++` version of our method is by two orders of magnitude faster than the MATLAB version; this improvement is important for projections in very high dimensions.

References

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.

J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In A. McCallum and S. Roweis, editors, *Proc. of the 25th Int. Conf. Machine Learning (ICML'08)*, pages 272–279, Helsinki, Finland, July 5–9 2008.

M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.0 beta. `http://cvxr.com/cvx`, Sept. 2012.

J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006.

W. Wang and M. Á. Carreira-Perpiñán. Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application. arXiv:1309.1541, Sept. 3 2013.