Optnet for DDPG Constrained L_2 Projection (QP)

Abhinav Bhatia

October 24, 2018

1 Problem Statement

Given a given node g (which has already been allocated C resources) from the constraints tree, we want to allocate resources to it's k children such that:

$$\sum_i^k z_i = C$$

$$\forall_i^k : 0 \leq \check{C}_i \leq z_i \leq \hat{C}_i \leq 1$$

We are given C and a vector $\vec{y} \in [0,1]^k$. We want to project this vector to the nearest (by L_2 norm) feasible solution $\vec{z}$.

2 The Quadratic Program

$$\begin{aligned} \min_{\vec{z}} \sum_i^k (z_i - y_i)^2 \quad \text{subject to} & \\ \sum_i^k z_i - C = 0 \qquad & \lambda \\ \forall_i^k : z_i - \hat{C}_i \leq 0 \qquad & \alpha_i \\ \forall_i^k : \check{C}_i - z_i \leq 0 \qquad & \beta_i \end{aligned} \tag{1}$$

Here $\lambda, \alpha_i, \beta_i$ are the corresponding Lagrange multipliers.

Since the objective function is *strictly* convex and the constraints are convex too, there exists a unique solution to this QP.

The objective can be rewritten as:

$$f = \sum_i^k z_i^2 - \sum_i^k 2y_i z_i + \sum_i^k y_i^2$$

The last term is a constant and is not really a part of the QP. Whether we include it or not, either way, it does not affect the KKT conditions.

If we want to write the objective function in form $\frac{1}{2} z^T Q^T z + c^T z$, then Q would be $2I$ and $\vec{c}$ would be $-2\vec{y}$.

3 The KKT Conditions

The Langragian is:

$$L(\vec{z},\vec{\alpha},\vec{\beta},\lambda) = \sum_i^k (z_i - y_i)^2 + \lambda(\sum_i^k z_i - C) + \sum_i^k \alpha_i(z_i - \hat{C}_i) + \sum_i^k \beta_i(\check{C}_i - z_i) \tag{2}$$

The KKT conditions (conditions satisfied by the solution $\vec{z}^*, \vec{\alpha}^*, \vec{\beta}^*, \lambda^*$ of the QP 1) is given by

$$\begin{aligned} & \nabla_{\vec{z},\lambda} L && = \vec{0} \\ \forall_i^k : \quad & \alpha_i(z_i - \hat{C}_i) && = 0 \\ \forall_i^k : \quad & \beta_i(\check{C}_i - z_i) && = 0 \end{aligned}$$

which expand to:

$$\begin{cases} & \sum_i^k z_i - C & = 0 \\ \forall_i^k : & 2(z_i - y_i) + \lambda + \alpha_i - \beta_i & = 0 \\ \forall_i^k : & \alpha_i(z_i - \hat{C}_i) & = 0 \\ \forall_i^k : & \beta_i(\check{C}_i - z_i) & = 0 \end{cases} \tag{3}$$

i.e. $3k+1$ equations.

4 Differentiating the KKT conditions

We can differentiate both sides of each equation in set of equations 3 w.r.t to inputs $\vec{y}$ and C.

The partial differential equations w.r.t. input y_j are:

$$\begin{cases} & \sum_i^k \frac{\partial z_i}{\partial y_j} & = 0 \quad (a) \\ \forall_i^k : & 2\frac{\partial z_i}{\partial y_j} - 2\delta_{ij} + \frac{\partial \lambda}{\partial y_j} + \frac{\partial \alpha_i}{\partial y_j} - \frac{\partial \beta_i}{\partial y_j} & = 0 \quad (b) \\ \forall_i^k : & \frac{\partial \alpha_i}{\partial y_j}(z_i - \hat{C}_i) + \alpha_i \frac{\partial z_i}{\partial y_j} & = 0 \quad (c) \\ \forall_i^k : & \frac{\partial \beta_i}{\partial y_j}(-z_i + \check{C}_i) - \beta_i \frac{\partial z_i}{\partial y_j} & = 0 \quad (d) \end{cases} \tag{4}$$

Here δ_{ij} is the Kronecker delta function, which is 1 when $i = j$, and 0 otherwise.

The partial differential equations w.r.t C are:

$$\begin{cases} & \sum_i^k \frac{\partial z_i}{\partial C} - 1 & = 0 \quad (a) \\ \forall_i^k : & 2\frac{\partial z_i}{\partial C} + \frac{\partial \lambda}{\partial C} + \frac{\partial \alpha_i}{\partial C} - \frac{\partial \beta_i}{\partial C} & = 0 \quad (b) \\ \forall_i^k : & \frac{\partial \alpha_i}{\partial C}(z_i - \hat{C}_i) + \alpha_i \frac{\partial z_i}{\partial C} & = 0 \quad (c) \\ \forall_i^k : & \frac{\partial \beta_i}{\partial C}(-z_i + \check{C}_i) - \beta_i \frac{\partial z_i}{\partial C} & = 0 \quad (d) \end{cases} \tag{5}$$

The equations can be solved independently per input y_j and C.

5 Solving system of equations 4 and 5

For equations 4, the variables are $\frac{\partial}{\partial y_j}$ of $\alpha_i, \beta_i, \lambda, z_i$. So there are $n = 3k+1$ variables and that many equations. Trying to write equations 4 in matrix form:

$$A^{y_j}_{n \times n} J^{y_j}_{n \times 1} = B^{y_j}_{n \times 1}$$

where

$$J^{y_j} = \frac{\partial}{\partial y_j}[\lambda, \alpha_1, \beta_1, z_1, ..., \alpha_k, \beta_k, z_k]^T$$

and

$$A^{y_j} = \begin{bmatrix} eqn4 & \vdots & \lambda & \alpha_1 & \beta_1 & z_1 & \cdots & \alpha_i & \beta_i & z_i \\ \cdots & \vdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a & \vdots & 0 & 0 & 0 & 1 & \cdots & 0 & 0 & 1 \\ b_1 & \vdots & 1 & 1 & -1 & 2 & \cdots & 0 & 0 & 0 \\ c_1 & \vdots & 0 & z_1 - \hat{C}_1 & 0 & \alpha_1 & \cdots & 0 & 0 & 0 \\ d_1 & \vdots & 0 & 0 & -z_1 + \check{C}_1 & -\beta_1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_k & \vdots & 1 & 0 & 0 & 0 & \cdots & 1 & -1 & 2 \\ c_k & \vdots & 0 & 0 & 0 & 0 & \cdots & z_k - \hat{C}_k & 0 & \alpha_k \\ d_k & \vdots & 0 & 0 & 0 & 0 & \cdots & 0 & -z_k + \check{C}_k & -\beta_k \end{bmatrix}$$

and

$$B^{y_j} = \begin{bmatrix} 0 \\ 2\delta_{1j} \\ 0 \\ 0 \\ \vdots \\ 2\delta_{kj} \\ 0 \\ 0 \end{bmatrix}$$

Basically,

$$A^{y_j}_{rc} = \begin{cases} 1 & r = 1, c = 3m+1 & m = 1, 2, ..., k \\ 1 & r = 3m-1, c = 1 & m = 1, 2, ..., k \\ 1 & r = 3m-1, c = r & m = 1, 2, ..., k \\ -1 & r = 3m-1, c = r+1 & m = 1, 2, ..., k \\ 2 & r = 3m-1, c = r+2 & m = 1, 2, ..., k \\ z_m - \hat{C}_m & r = 3m, c = r-1 & m = 1, 2, ..., k \\ \alpha_m & r = 3m, c = r+1 & m = 1, 2, ..., k \\ -z_m + \check{C}_m & r = 3m+1, c = r-1 & m = 1, 2, ..., k \\ -\beta_m & r = 3m+1, c = r & m = 1, 2, ..., k \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

and

$$B^{y_j}_r = \begin{cases} 2\delta_{mj} & r = 3m-1 \quad m = 1, 2, ..., k \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Thus we can find

$$J^{y_j} = (A^{y_j})^{-1} B^{y_j} \tag{8}$$

Note from equation 6 that A^{y_j} does not depend on j. Thus its inverse need not be computed seperately for each and every j.

Also, It is clear from set of equations 4 and 5 that

$$\forall_j^k : \quad A^C = A^{y_j} \tag{9}$$

Only B^C is different:

$$B_r^C = \begin{cases} 1 & r = 1 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Thus,

$$J^C = (A^C)^{-1} B^C \tag{11}$$

The overall Jacobian would be simply the horizontal contatenation:

$$J_{(3k+1)\times(k+1)} = \begin{bmatrix} J^{y_1} & J^{y_2} & \ldots & J^{y_k} & J^C \end{bmatrix} \tag{12}$$

6 The overall picture

From J, we can extract the rows corresponding to z_i and transpose it and thus write $\nabla_{\vec{y},C}\vec{z}$, which is a $(k+1) \times k$ matrix.

Thus we can get gradient of output $\vec{z}$ w.r.t network parameters $\theta \in \mathbb{R}^p$ using the chain rule as:

$$(\nabla_\theta \vec{z})_{p\times k} = (\nabla_\theta (\vec{y}, c))_{p\times(k+1)} (\nabla_{(\vec{y},C)} \vec{z})_{(k+1)\times k}$$

7 Computation requirements

Let N be the size of the minibatch.

Computation needed to compute A^{-1} will be of order k^3. And luckily it need not be computed for all y_j and C (by 9). $A^{-1}B$ ($\propto k^2$) will be done k times, i.e. k^3. Thus just $\mathcal{O}(k^3)$ per backward pass.

For the entire minibatch, $\mathcal{O}(Nk^3)$. Plus all the matrix operations can be parallelized on a GPU and $\nabla_\theta \vec{z}$ can be computed parallely for each minibatch item. Then $\nabla_\theta \vec{z}$ can be averaged (parallely per cell) across the minibatch in $\mathcal{O}(N)$ time.