Optnet for DDPG Constrained L_2 Projection (QP)

Abhinav Bhatia

November 8, 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:

$$f = \sum_{i}^{k} z_i^2 - \sum_{i}^{k} 2 y_i z_i + \sum_{i}^{k} y_i^2$$

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 $\frac{1}{2} z^T Q z + c^T z$, 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:

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

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:

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

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:

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

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

$$A^{y_j} = \begin{bmatrix} eqn4 & \vdots & \lambda & \alpha_1 & \beta_1 & z_1 & \cdots & \alpha_i & \beta_i & z_i \\ \cdots & \vdots & \multicolumn{8}{c}{\dotfill} \\ a & \vdots & 0 & 0 & 0 & 1 & \cdots & 0 & 0 & 1 \\ b_1 & \vdots & 1 & 1 & -1 & 2 & \cdots & 0 & 0 & 0 \\ c_1 & \vdots & 0 & z_1 - \hat{C}_1 & 0 & \alpha_1 & \cdots & 0 & 0 & 0 \\ d_1 & \vdots & 0 & 0 & -z_1 + \check{C}_1 & -\beta_1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_k & \vdots & 1 & 0 & 0 & 0 & \cdots & 1 & -1 & 2 \\ c_k & \vdots & 0 & 0 & 0 & 0 & \cdots & z_k - \hat{C}_k & 0 & \alpha_k \\ d_k & \vdots & 0 & 0 & 0 & 0 & \cdots & 0 & -z_k + \check{C}_k & -\beta_k \end{bmatrix}$$

and

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

Basically,

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

and

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

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:

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

6 Issues

There can be two main issues:

1. Zero gradients
   This can happen when for some i, $z_i = \hat{C}_i$ or $z_i = \check{C}_i$, irrespective of the exact value of $\vec{y}$. In that case, $\frac{\partial z_i}{\partial y_j} = 0, \forall j$, which means that irrespective of the loss function, z_i, would not change at this y. But this is not problem in itself, since y can continue to change through $\frac{\partial z_{\neq i}}{\partial y_j}$, and hopefully for some other value of y, z_i will start changing. But still, the whole thing can get stuck in a local minimum, if the gradient of the loss function becomes zero w.r.t all components of y.
   In practice, it is not getting stuck in any local minimum, but this is really an issue of a landscape in which one has to go *around* lots of hills, making the convergence slow. A hack is to make $\vec{y}$ have a range between $\vec{\check{C}}$ and $\vec{\hat{C}}$ by say preprocessing it with a scaled *tanh* layer. This *tanh* layer can cause gradient vanishing gradients. A hack is to scale it to between $\vec{\check{C}} - \epsilon$ and $\vec{\hat{C}} + \epsilon$. This way the gradients due to *tanh* become better; and although the original issue can appear again, it becomes way less severe (there are no long distances to travel around hills).

2. Non differentiable points
   Happens when matrix A has a zero row. This is possible if both $z_i - \hat{C}_i = \alpha_i = 0$ for some i. This would happen if the $y_i \leq \hat{C}_i$ (making $\alpha = 0$) and gets projected to the boundary ($z_i = \hat{C}_i$). An example would be that y was a feasible point and atleast one of the components was *exactly on* the boundary. In that case, $z = y$, and for those boundary components, $z_i - \hat{C}_i = \alpha_i = 0$.

7 Batched Problem

Now we want to solve the problem for the entire batch simultaneously. Let the batch size be N.

For sample s, given node g_s (which has already been allocated C_s resources) from the constraints tree, we want to allocate resources to it's k children such that:

$$\sum_{i}^{k} z_{si} = C_s$$

$$\forall i \in 1, ..., k : 0 \leq \check{C_{si}} \leq z_{si} \leq \hat{C_{si}} \leq 1$$

i.e. for all $s \in 1, ..., N$, we are given C_s and a vector $\vec{y_s} \in [0, 1]^k$. We want to project this vector to the nearest (by L_2 norm) feasible solution $\vec{z_s}$.

8 Batched QP

The objective of the combined QP should be just the sum of objectives of the individual QPs.

$$\begin{aligned} \min_{\boldsymbol{z}} \sum_{s}^{N} \sum_{i}^{k} (z_{si} - y_{si})^2 \quad & \text{subject to} \\ \forall s = 1..N : \sum_{i}^{k} z_{si} - C_s \quad &= 0 \qquad \lambda_s \\ \forall s = 1..N : \forall i = 1..k : z_{si} - \hat{C_{si}} \quad &\leq 0 \qquad \alpha_{si} \\ \forall s = 1..N : \forall i = 1..k : \; - z_{si} + \check{C_{si}} \quad &\leq 0 \qquad \beta_{si} \end{aligned} \tag{13}$$

Here $\lambda_s, \alpha_{si}, \beta_{si}$ 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:

$$f = \sum_{s}^{N} \sum_{i}^{k} z_{si}^2 - \sum_{s}^{N} \sum_{i}^{k} 2 y_{si} z_{si} + \sum_{s}^{N} \sum_{i}^{k} y_{si}^2$$

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.

For inputs to programs like cplex, if we want to write the objective function in form $\frac{1}{2} z^T Q z + c^T z$, (here z is a vector of Nk dimensions, i.e. flattened version of $\boldsymbol{z}$) then Q would be $2I$ and $\vec{c}$ would be $-2\vec{y}$ (here $\vec{y}$ is flattened version of $\boldsymbol{y}$)

9 Batched QP KKT Conditions

The Langragian is:

$$L(\boldsymbol{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \vec{\lambda}) = \sum_{s}^{N} \sum_{i}^{k} (z_{si} - y_{si})^2 + \sum_{s}^{N} \lambda_s (\sum_{i}^{k} z_{si} - C_s) + \sum_{s}^{N} \sum_{i}^{k} \alpha_{si} (z_{si} - \hat{C_{si}}) + \sum_{s}^{N} \sum_{i}^{k} \beta_{si} (\check{C_{si}} - z_{si}) \tag{14}$$

The KKT conditions (conditions satisfied by the solution $\boldsymbol{z}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^* \vec{\lambda}^*$ of the QP 13) is given by

$$\begin{aligned} \nabla_{\boldsymbol{z}, \vec{\lambda}} L \quad & = \mathbf{0} \\ \forall s = 1..N : \forall i = 1..k : \quad \alpha_{si} (z_{si} - \hat{C_{si}}) \quad & = 0 \\ \forall s = 1..N : \forall i = 1..k : \quad \beta_{si} (-z_{si} + \check{C_{si}}) \quad & = 0 \end{aligned}$$

which expand to:

$$\begin{cases} \forall s = 1..N : \sum_{i}^{k} z_{si} - C_s & = 0 \\ \forall s = 1..N : \forall i = 1..k \quad : 2(z_{si} - y_{si}) + \lambda_s + \alpha_{si} & = 0 \\ \forall s = 1..N : \forall i = 1..k \quad : \alpha_{si}(z_{si} - \hat{C_{si}}) & = 0 \\ \forall s = 1..N : \forall i = 1..k \quad : \beta_{si}(-z_{si} + \check{C_{si}}) & = 0 \end{cases} \tag{15}$$

i.e. $N(3k + 1)$ equations.

10 Differentiating the Batch KKT conditions

We can differentiate both sides of each equation in set of equations 15 w.r.t to inputs $\boldsymbol{y}$ and $\vec{C}$.

We already know that the Jacobian of $\vec{z_s}$ w.r.t $\vec{y_t}$ will be zero when $t \neq s$, since samples do not affect each other during the forward pass. Thus we need to compute individual Jacobians only.

Let the subscript s be implicit in all the text from this point. i.e. we are talking about sample s.

The partial differential equations are exactly same as 4 and 5, which can be solved independently per input y_j and C.

11 Solving system of equations 4 and 5

Exact same procedure as in section 5 can be followed to calculate the Jacobian per sample $s = 1..N$.

12 Closed form solution

TODO

13 Computation requirements

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, $\mathcal{O}(Nk^3)$. Computation per minibatch can be done parallely. Plus all the matrix operations can be parallelized on a GPU. So the complexity can be dialed back to less than $\mathcal{O}(k^3)$.

14 Implementation

We can define new ops in tensorflow. for backward pass, the matrix calculations can be done using the tensorflow function, i.e. it can be done on a GPU.