ADMM算法理论与应用

前言

交替方向乘子法（Alternating Direction Method of Multipliers，ADMM）是一种解决可分解凸优化问题的简单方法，尤其在解决大规模问题上卓有成效，利用ADMM算法可以将原问题的目标函数等价的分解成若干个可求解的子问题，然后并行求解每一个子问题，最后协调子问题的解得到原问题的全局解。ADMM 最早分别由 Glowinski & Marrocco 及 Gabay & Mercier 于 1975 年和 1976 年提出，并被 Boyd 等人于 2011 年重新综述并证明其适用于大规模分布式优化问题。由于 ADMM 的提出早于大规模分布式计算系统和大规模优化问题的出现，所以在 2011 年以前，这种方法并不广为人知。

对偶上升方法

考虑等式约束的最优化问题如下

$$\min_x f(x)$$

$$st. Ax = b$$

其中$x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $f: \mathbb{R}^n \in \mathbb{R}$是凸函数
原问题的拉格朗日函数为：

$$L(x, y) = f(x) + y^T(Ax - b)$$

那么其对偶函数为：

$$g(y) = inf_x L(x, y) = -f^*(-A^Ty) - b^Ty$$

其中y是拉格朗日乘子，也是对偶变量，f^*是f共轭函数。
假设满足强对偶性，则原问题和对偶问题的最优值相等。我们设原问题最优解为x^*,对偶问题最优解为y^*，则

$$x^* = argmin_x L(x, y^*)$$

在对偶上升方法中，对偶问题是通过梯度上升方法来解，因此对偶上升迭代更新为：

$$x_{k+1} = argmin_x L(x, y^k)$$

$$y^{k+1} = y^k + \alpha_k(Ax^k + 1 - b)$$

其中$\alpha_k > 0$,是梯度上升的步长。

对偶分解性

对偶上升方法中在满足强对偶性条件下，通过梯度上升来逐步调整对偶变量，再通过对偶变量来求解原问题最优解，这样的好处是在有些情况下可以使算法可分解，假设目标函数是可分解的，即，

$$f(x) = \sum_{i=1}^N f_i(x_i)$$

其中$x = (x_1, x_2, \ldots, x_N)$, $x_i \in \mathbb{R}^{n_i}$,划分矩阵A

$$A = [A_1, A_2, \cdots, A_N]$$

所以$Ax = \sum_{i=1}^N A_i x_i$,则拉格函数重写成

$$L(x, y) = \sum_{i=1}^N L_i(x_i, y) = \sum_{i=1}^N (f_i(x_i) + y^T A_i x_i - (\frac{1}{N}) y^T b)$$

对偶上升的迭代更新:

$$x_i^{k+1} = argmin_x L_i(x_i, y^k)$$

$$y^{k+1} = y^k + \alpha^k(Ax^{k+1} - b)$$

增广拉格朗日（乘子法）

为了增加对偶上升方法的鲁棒性和放松目标强凸的要求，引入了增广拉格朗日，增广拉格朗日，形式如下:

$$L_\rho(x, y) = f(x) + y^T(Ax - b) + \frac{\rho}{2}\|Ax - b\|_2^2$$

其中$\rho > 0$，是惩罚参数。
增广拉格朗日相当于在拉格朗日加了一个强凸的惩罚项使得原约束问题变成如下形式

$$\min_x f(x) + \frac{\rho}{2}\|Ax - b\|_2^2$$

$$s.t. Ax = b$$

对偶上升迭代更新:

$$x^{k+1} = argmin_x L_\rho(x, y^k)$$

$$y^{k+1} = y^k + \rho(Ax^{k+1} - b)$$

这就是所谓的乘子法，从迭代形式上看，有两点和对偶上升方法不一样，第一是x^{k+1}更新使用增广拉格朗日，第二是更新的步长用惩罚参数ρ代替了α^k。乘子法虽然可以比对偶上升方法在更一般的条件下收敛，但是增加了二次惩罚项，使得原问题失去可分解性，因而引入了ADMM算法。

交替方向乘子法（ADMM）

假设有如下优化问题:

$$\min_x f(x) + g(z)$$

$$s.t. Ax + Bz = c$$

其中$x \in \mathbb{R}^n, z \in \mathbb{R}^m$, $A \in \mathbb{R}^{p \times n}$, $B \in \mathbb{R}^{p \times m}$, $c \in \mathbb{R}^p$
其增广拉格朗日函数如下：

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2$$

ADMM的更新迭代形式如下:

$$x^{k+1} = argmin_x L_\rho(x, y^k, z^k)$$

$$z^{k+1} = argmin_z L_\rho(x^{k+1}, z, y^k)$$

$$y^{k+1} = y^k + \rho(Ax^{k+1} + Bz^{k+1} - c)$$

其中$\rho > 0$，以上就是最原始的ADMM算法迭代形式

全局变量一致性优化(Global Variable Consensus Optimization)

接下来讨论如何将其应用到机器学习算法分布式计算中，但在这之前我们先讨论一个比较广义的优化问题如下:

$$\min_x \sum_{i=1}^N f_i(x_i)$$

$$s.t. x_i - z = 0, i = 0, 1, \ldots, N$$

以上问题形式就是全局一致性问题，x_i局部变量，z是全局一致变量。从限制约束条件来看，每一个局部变量需要保持一致。
其增广拉格朗日函数为:

$$L_\rho(x_1, x_2, \ldots, x_N, z, y) = \sum_{i=1}^N f_i(x_i) + y_i^T(x_i - z) + \frac{\rho}{2}\|x_i - z\|_2^2$$

ADMM迭代更新形式:

$$x_i^{k+1} = argmin_x f_i(x_i) + y_i^{Tk}(x_i - z^k) + \frac{\rho}{2}\|x_i - z^k\|_2^2$$

$$z^{k+1} = \frac{1}{N}\sum_{i=1}^N (x_i^{k+1} + \frac{1}{\rho}y_i^k)$$

$$y_i^{k+1} = y_i^k + \rho(x_i^{k+1} - z^{k+1})$$

我们可以更进一步将上述问题推广更广义的优化问题，等正则的全局一致性优化问题，具体问题如下:

$$\min_x \sum_{i=1}^N f_i(x_i) + g(z)$$

$$s.t. x_i - z = 0, i = 0, 1, \ldots, N$$

其ADMM迭代更新形式如下:

$$x_i^{k+1} = argmin_x (f_i(x_i) + y_i^{Tk}(x_i - z^k) + \frac{\rho}{2}\|x_i - z^k\|_2^2)$$

$$z^{k+1} = argmin_z (g(z) + \sum_{i=1}^N (-y_i^{Tk} + \frac{\rho}{2}\|x_i^{k+1} - z\|_2^2))$$

$$y_i^{k+1} = y_i^k + \rho(x_i^{k+1} - z_i^{k+1})$$

##ADMM算法的应用
以上讨论了一些的理论的东西。接下来就尝试一下如何去应用ADMM算法去解决一些机器学习问题。实际上大多数分布式机器学习问题可以表达成全局一致性优化问题：

$$\min_x \sum_{i=1}^N f_i(x_i) + g(z)$$

$$s.t. x_i - z = 0, i = 0, 1, \ldots, N,$$

$f_i(x_i)$表示划分到每一个节点上的目标函数，x_i表示局部模型参数，z表示全局一致性变量。我们把整个数据集划分到每一节点，各节点独立并行训练，通过迭代更新，最终收敛到一个一致的全局解。

##Lasso问题
具体问题形式如下:

$$\min_x \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1$$

其中,$A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$.
更换成全局一致性优化问题：

$$\min_{x_i} \sum_{i=1}^N \frac{1}{2}\|Ax - b\|^2 + \lambda\|z\|_1$$

$$s.t. x_i - z = 0, i = 0, 1, \ldots, N$$

ADMM迭代更新形式如下:

$$x_i^{k+1} = argmin_{x_i}(\frac{1}{2}\|A_i x_i - b_i\|^2 + y_i^{Tk}(x_i - z^k) + \frac{\rho}{2}\|x_i - z^k\|_2^2)$$

$$z^{k+1} = argmin_z(\lambda\|z\|_1 + \sum_{i=1}^N(-y_i^{Tk}z + \frac{\rho}{2}\|x_i^{k+1} - z\|_2^2))$$

$$y_i^{k+1} = y_i^k + \rho(x_i^{k+1} - z^{k+1})$$

对于x^{k+1}的更新，右边括号内的函数对x求可导得(这里可以有更高效的算法求解):

$$A_i^T(A_i x_i - b_i) + y_i^k + \rho(x_i - z^k) = 0$$

即可得:

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

对于z的更新如下:

$$z^{k+1} = argmin_z(\lambda\|z\|_1 + \frac{\rho N}{2}\|z - \frac{1}{\rho N}\sum_{i=1}^N(\rho x_i^{k+1} + y_i^k)\|_2^2)$$

因此z的更新可以使用softthreshold方法如下所示：

$$z^{k+1} = \mathcal{S}_{\lambda/\rho N}(\frac{1}{\rho N}\sum_{i=1}^N(\rho x_i^{k+1} + y_i^k))$$

其中

$$\mathcal{S}_\kappa(a) = \begin{cases} a - \kappa, & a > \kappa \\ 0, & |a| \le \kappa \\ a + \kappa, & a < -\kappa \end{cases}$$