带有图像导数的能量函数的求解

侠之大者

2016 年 3 月 24 日

我们知道，在图像处理中，先验知识是必不可少的。有时候，我们会在能量函数中添加有关图像导数的约束条件。那么这类能量函数如何去求解呢?

1 问题的提出

1.1 导数的定义

对于一幅大小为 $m \times n$ 的图像 X, 其中的每一像素点 $X(i,j)$ 都有两个偏导数。

水平方向的偏导数为

$$\frac{\partial}{\partial h}X(i,j) = X(i+1,j) - X(i,j)$$

竖直方向的偏导数为

$$\frac{\partial}{\partial v}X(i,j) = X(i,j+1) - X(i,j)$$

为了更简单直观的表示，我们使用图像的模板，

$$G_h = \begin{bmatrix} -1 & 1 \end{bmatrix} \quad G_v = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

这里，模板 G_h, G_v 的中心为点 (1,1)，即中心像素点对应的系数为 -1。于是，整个图像的偏导数为

$$X_h = G_h \otimes X \quad X_v = G_v \otimes X$$

其中，$\otimes$ 表示图像的模板操作 (卷积运算)，这也是大家最熟悉的操作。

这里有一个细节问题，图像边缘处的导数如何定义。其实，无论怎么去定义都无关大雅。为了下文表述的方便，我们给出的定义如下：

$$\frac{\partial X(i,n)}{\partial h} = X(i+1,1) - X(i,n) \qquad \frac{\partial X(m,j)}{\partial v} = X(m,1) - X(m,j)$$

其中 $1 \le i \le m, 1 \le j \le n$。（这里 $\frac{\partial X(i,n)}{\partial h}$ 与大家习惯的定义不一样，我们这样定义，只是为了下文推导方便。）

1.2 经典的例子

在这里，给出一个大家都熟悉的 L_0 图像平滑例子来展开讨论。

假设原图像为 Y，大小为 $m \times n$，我们想要得到的光滑图像为 X, 先验约束条件是理想图像 X 中应该有尽可能多的像素点 x 的导数 (x_h, x_v) 为零向量 $(0,0)$。

我们把所有像素点的水平方向偏导数排成一个列向量，记做 X_h, 所有像素点的竖直方向偏导数排成一个列向量，记做 X_v, 它们的大小都是 $N \times 1$，N 表示像素点的总个数，有

$$\min_X ||X-Y||^2 + \alpha ||(X_h, X_v)||_0$$

其中 $||(X_h, X_v)||_0$ 表示图像 X 上所有导数满足条件 $(x_h, x_v) \neq \mathbf{0}$ 的像素点 x 的个数。

如何去求解呢?

我们观察这个能量函数，它只有一个自变量 X。X_h, X_v 是随 X 而变化的，而且被放在比较麻烦的 0-范数之中。

在这种情况下，我们一般采用变量分离法，即添加新的变量 u, t 去分别代替 X_h, X_v 的位置，再添加一项保证 u 与 X_h, t 与 X_v 比较接近，有

$$\min_{X,u,t} ||X-Y||^2 + \alpha ||(u,t)||_0 + \beta(||X_h - u||^2 + ||X_v - t||^2)$$

这样做有什么好处? 在变量分离之后，虽然自变量的数目变多了，但是，在 0-范数里面的内容 (u,t) 已经与变量 X 无关了。这就是变量分离的目的。

接下来，自然是交替迭代法，因为变量分离之后就不止一个自变量了。

首先，固定 X, 求解 u, t,

$$\min_{u,t} \alpha ||(u,v)||_0 + \beta(||X_h - u||^2 + ||X_v - t||^2)$$

这可以通过硬阈值收缩求解。

然后，固定 u, t, 求解 X。这才是我们这一次要讲的主题。如何求解呢，我们留到下一章再说。

2 全局闭合解

现在，我们给出上面的第二个子问题，

$$\min_X ||X-Y||^2 + \beta(||X_h-u||^2 + ||X_v-t||^2) \tag{1}$$

其中 X 未知，Y, u, t 都已知的。

按照我们以前的思路，把未知量当做列向量，将能量函数转换成矩阵乘积的形式。那么 X_h, X_v 怎么表示呢?

我们细心观察一下。(这里，注意我们给出的图像边缘处的偏导数定义。)

$$\begin{aligned} X_h = \begin{pmatrix} x_2-x_1 \\ x_3-x_2 \\ \vdots \\ x_{i+1}-x_i \\ \vdots \\ x_N-x_{N-1} \\ x_1-x_N \end{pmatrix} &= \begin{pmatrix} -1 & 1 & & & & & \\ & -1 & 1 & & & & \\ & & \ddots & \ddots & & & \\ & & & -1 & 1 & & \\ & & & & \ddots & \ddots & \\ & & & & & -1 & 1 \\ 1 & & & & & & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_{N-1} \\ x_N \end{pmatrix} \\ &= HX \end{aligned} \tag{2}$$

同理，$X_v = VX$, 其中

$$V = \begin{pmatrix} -1 & \cdots & \cdots & 1 & & & & \\ & \ddots & & & \ddots & & & \\ & & -1 & \cdots & \cdots & 1 & & \\ & & & \ddots & & & \ddots & \\ & & & & -1 & \cdots & \cdots & 1 \\ 1 & & & & & -1 & & \\ & \ddots & & & & & \ddots & \\ & & 1 & & & & & -1 \end{pmatrix}$$

这样，上面问题的矩阵形式为

$$\min_{X} ||X-Y||^2 + \beta||HX-u||^2 + \beta||VX-t||^2$$

对 X 进行求导，并令导数为 0,

$$(X-Y) + \beta H^T(HX-u) + \beta V^T(VX-t) = 0$$

所以 X 的闭合解为:

$$X = (I + \beta H^T H + \beta V^T V)^{-1}(Y + \beta H^T u + \beta V^T t) \tag{3}$$

如果在 matlab 中，上面的闭合解 (3) 可以用一行代码求解。这里，我们想介绍一下 matlab 中稀疏矩阵的表示。

在 matlab 中，使用 sparse 函数生成稀疏矩阵。

$$S = sparse(V, R, C, m, n)$$

其中，V, R, C 都是 $k \times 1$ 的列向量，分别对应于非零元素的值，横坐标和纵坐标，k 是非零元素的个数，m, n 是稀疏矩阵的大小。

例如，有一个稀疏矩阵

$$A = \begin{pmatrix} 9 & & & \\ & & 10 & \\ & 12 & & 28 \end{pmatrix}$$

则有

$$V = \begin{pmatrix} 9 \\ 10 \\ 12 \\ 28 \end{pmatrix} \quad R = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 3 \end{pmatrix} \quad C = \begin{pmatrix} 1 \\ 3 \\ 2 \\ 4 \end{pmatrix}$$

这样，

$$S = sparse(V, R, C, 3, 4)$$

便生成了矩阵 A 对应的稀疏矩阵 S，它与普通矩阵一样，可以进行加减乘除和求逆等运算。

3 梯度下降法

如果我们使用 C++ 编程，也没有使用稀疏矩阵的软件包，那该如何求解能量函数 (1) 呢?

在解类似的问题时，大部分的论文都轻描淡写地说，使用梯度下降法求解。那如何使用梯度下降法求解呢，我们来看一看。

我们令 $f(X) = ||X - Y||^2 + \beta||HX - u||^2 + \beta||VX - t||^2$, 则能量函数式 (1) 为:

$$\min_X f(X)$$

这里，函数 $f(X)$ 是一个可导的凸函数，有全局唯一的最小值点。

梯度下降法的基本形式为:

$$X^{k+1} = X^k - \alpha\nabla f(X^k)$$

其中 α 表示步长，X^k 表示第 k 次迭代时的坐标值。梯度 $\nabla f(X)$ 表示函数 $f(X)$ 上升最快的方向，

$$\nabla f(X) = 2(X - Y + \beta H^T HX - \beta H^T u + \beta V^T VX - \beta V^T t)$$

这个梯度 $\nabla f(X)$ 似乎很难计算。

很多时候，我们遇到的困难并没有我们想的那么大，如果不去试一试，就永远不知道结果。

我们来试一试;

$$\begin{aligned}
H^T H &= \begin{pmatrix} -1 & & & & 1 \\ 1 & -1 & & & \\ & \ddots & \ddots & & \\ & & 1 & -1 & \\ & & & 1 & -1 \end{pmatrix} \begin{pmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \\ 1 & & & & -1 \end{pmatrix} \\
&= \begin{pmatrix} 2 & -1 & & & -1 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ -1 & & & -1 & 2 \end{pmatrix}
\end{aligned}$$

我们来观察 H^THX, 它是不是下面的图像模板 G_H 与图像 X 进行卷积运算的结果，即 $H^THX = G_H \otimes X$, 其中

$$G_H = \begin{bmatrix} -1 & 2 & -1 \end{bmatrix}$$

这里，等式左边的 X 表示列向量, 等式右边的 X 表示矩阵。(注意我们边缘处像素点的偏导数定义)

接着，我们发现 $H^Tu = B_H \otimes u$, 其中

$$B_H = \begin{bmatrix} 1 & -1 \end{bmatrix}$$

这里，模板 B_H 中心为点 (1,2)。

同理，大家可以计算一下，$V^TVX = G_V \otimes X, V^Tt = B_V \otimes t$, 其中

$$G_V = \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix} \quad B_V = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

这里，模板 B_V 的中心为点 (2,1)。

再进一步合并上面的结果，有

$$\begin{aligned} \nabla f(X) &= 2((I + \beta G_H + \beta G_V) \otimes X - \beta B_H \otimes u - \beta B_V \otimes t - Y) \\ &= 2(T \otimes X - \beta B_H \otimes u - \beta B_V \otimes t - Y) \end{aligned}$$

其中模板 T 为

$$T = \begin{bmatrix} & -\beta & \\ -\beta & 4\beta + 1 & -\beta \\ & \beta & \end{bmatrix}$$

这样，我们可以通过模板运算快速地计算出梯度 $\nabla f(X)$。

这里补充一点：如果定义图像最右边像素点水平偏导数为

$$\frac{\partial X(i,n)}{\partial h} = X(i,1) - X(i,n)$$

那上述一切的推导都是成立的，只是式 (2) 中矩阵 H 的元素分布稍微复杂一点。

4 二次型能量函数问题的再思考

还记得我们讲过的二次型能量函数吗?

$$\min_X \sum_i (X_i - Y_i)^2 + \alpha \sum_i \sum_{j \in N(i)} (X_i - X_j)^2$$

对应的矩阵形式:

$$\min_X (X-Y)^T(X-Y)+\alpha X^T HX$$

其中 $H=(H_{ij})_{N\times N}$，H_{ij} 是矩阵 H 的第 i 行第 j 列的元素。

$$H_{ij}=\begin{cases} 8 & if \quad i=j \\ -2 & if \quad i\neq j, \quad i\in N(j) \\ 0 & else \end{cases}$$

这里我们能不能使用梯度下降法去求解呢?

令 $f(X)=(X-Y)^T(X-Y)+\alpha X^T HX$, 则 $f(X)$ 的梯度为:

$$\nabla f(X)=2((I+\alpha H)X-Y)$$

我们再来看矩阵 H 的形式，这不正是标准的模板操作吗?

$$\nabla f(X)=2(G\otimes X-Y)$$

$$G=\begin{bmatrix} & -2\alpha & \\ -2\alpha & 8\alpha+1 & -2\alpha \\ & -2\alpha & \end{bmatrix}$$

哈哈，当时怎么就没有看出来呢?

5 傅里叶变换

在 L_0 图像平滑的论文中，采用了快速傅里叶变换的方法求解。我想给出我的看法。

首先，如果使用 matlab 编程，使用稀疏矩阵很方便，一行代码就给出了最优解。毕竟，把复杂的数学运算留给机器，花更多的精力去思考问题，这是非常明智的行为。

其次，如果使用 C++ 编程，梯度下降法非常方便，写个循环迭代一下就行。我以前试过将 L_0 图像平滑的 matlab 代码改写为 C++ 代码，发现使用快速傅里叶变换库 fftw 比较麻烦。

最后，我不明白为什么这里可以用傅里叶变换，嘿嘿，毕竟能力有限嘛。教材上说，图像的卷积运算可以转换成在傅里叶变换域中的乘法运算;傅里叶变换可以加速问题的求解。好吧，我就知道这么多了。大家如果有兴趣，可以去了解一下。