Image Inpainting and sparse matrices

Asaf Shimshovitz

February 2016

One of the stumbles when working with images, is that you can easily find yourself dealing with very large systems. The 2-D nature of image has the potential to turn simple manipulations into huge computations involving millions of variables. This without saying a word on 3-D volumes and videos.

One of the efficient tools, among the methods for dealing with such large computations (such as GPU, parallel computing), is using sparse systems.

As an example I will introduce the image inpainting problem. In image inpainting we have information only on part of the pixels and the goal is to recover the values of the unknown pixels. In order to complicate things a little bit, let's assume that even the known pixels are noisy and may have inaccurate value. Two things are guiding us in restoring the new image. 1) Remain close as possible to the data of the given pixels 2) The smoothness nature of images, that says that two neighboring pixel have a good chance to share the same value.

The first guiding line is translating to data Energy terms:

$$\mathrm{E}_{data} = \sum_i u(i)(I(i) - J(i))^2 \quad (1)$$

Where $J(i)$ is the value of pixel i in the original Image J, $I(i)$ is the value of pixel i in the desired image we want to find I, and $u(i)$ is the penalty for deviation from the original image J. A reasonable choice for u can be 1 for all the pixels that have data in J and 0 for all the others.

The second guide line is translating to the pairwise energy function:

$$\mathrm{E}_{pairwise} = \sum_i (d_x(i))^2 + (d_y(i))^2 \quad (2)$$

where $d_x(i)$ and $d_y(i)$ are the derivative of I in the i^{th} pixel in the x,y directions respectively.

The best image I is the one that minimize the total energy $E = E_{data} + \lambda E_{pairwise}$. Where λ control the weighting between smoothness and the data term.

Surprisingly this minimization problem has a global analytic solution. In order to find it we will perform 2 tricks: First we will write the images I and J as a vectors size Nx1 where N is the number of pixels in the images. Second we will write the energy in a matrix form:

E = (I-J)$^\dagger U(I-J) + \lambda((D_xI)^\dagger(D_xI) + (D_yI)^\dagger(D_yI))$
(3)

Here U is a diagonal matrix size NxN where $U(i,i)$ is the data weight $u(i)$ and D_x , D_y are derivative matrix operator size NxN.

Setting the derivative of E with respect to I to 0, and do some more lines of algebra, we are getting that the desired I satisfied the linear system of equations:

AI = b
(4)

where $A = U + \lambda(Dx^\dagger Dx + Dy^\dagger Dy)$ and $b = UJ$.

So, after a long way (especially if you do the math) we have a recipe for an optimal solution for the inpainting problem, all we need to do is constructing A and b and use some linear solver (like Matlab backslash /).

Nevertheless, it turns out to be not that simple. Naively implementation of the equations above get you into computational problems. Look at the size of the matrices: I and b are just vectors with size N - the number of pixels, but A , U,D_x and D_y are all huge matrices size N by N. That means that for a moderate image size 512 x 512 pixels we need to construct and solve a linear system for matrices with more than 60 Giga entries! An impossible task, both computationally and memory wise.

Very fortunately, the sparse nature of those matrices come to our help. A sparse matrix is a matrix that large portion of its entries are zero. In Matlab, defining such a matrix as sparse results in saving memory storage and in computational effort. U is a great example of sparse matrix since its diagonal. What about Dx and Dy? Let's focus on Dy. When applying Dy on an image I, we replace the ^{ith} entry with I(i) - I(i-1). In order to achieve this behavior Dy has 1 on its diagonal, -1 on its lower secondary diagonal and zero elsewhere. Again a very sparse matrix! From the same considerations Dx has a similar sparse nature.

Taking advantage of this sparse nature we can construct all the involved matrices as sparse (This can be done by using the Matlab commands: sparse, speye and spdiag). As a result the inpainting problem is solved accurately, fast and without any memory issues.

As a footnote we should add, that this formulation can be easily generalized for other linear smoothness terms (high order derivatives) and for cases where each pixel has a different smoothness weight.