Compressed sensing and single-pixel cameras

13 April, 2007 in expository, math.NA, non-technical | Tags: compressed sensing, linear algebra, single-pixel camera

I've had a number of people ask me (especially in light of some recent publicity) exactly what "compressed sensing" means, and how a "single pixel camera" could possibly work (and how it might be advantageous over traditional cameras in certain circumstances). There is a large literature on the subject, but as the field is relatively recent, there does not yet appear to be a good non-technical introduction to the subject. So here's my stab at the topic, which should hopefully be accessible to a non-mathematical audience.

For sake of concreteness I'll primarily discuss the camera application, although compressed sensing is a more general measurement paradigm which is applicable to other contexts than imaging (e.g. astronomy, MRI, statistical selection, etc.), as I'll briefly remark upon at the end of this post.

The purpose of a camera is, of course, to record images. To simplify the discussion, let us think of an image as a rectangular array, e.g. a 1024 x 2048 array of pixels (thus there are 2 megapixels in all). To ignore the (minor) issue of colour, let us assume that we are just taking a black-and-white picture, so that each pixel is measured in grayscale as an integer (e.g. an 8-bit integer from 0 to 255, or a 16-bit integer from 0 to 65535) which signifies the intensity of each pixel.

Now, to oversimplify things quite a bit, a traditional digital camera would take one measurement of intensity for each of its pixels (so, about 2 million measurements in the above example), resulting in a relatively large image file (2MB if one uses 8-bit grayscale, or 4MB if one uses 16-bit grayscale). Mathematically, this file can be represented by a very high-dimensional vector of numbers (in this example, the dimension is about 2 million).

Before I get to the new story of "compressed sensing", I have to first quickly review the somewhat older story of plain old "compression". (Those who already know how image compression works can skip forward a few paragraphs.)

The images described above can take up a lot of disk space on the camera (or on some computer where the images are later uploaded), and also take a non-trivial amount of time (and energy) to transfer from one medium to another. So, it is common practice to get the camera to *compress* the image, from an initial large size (e.g. 2MB) to a much smaller size (e.g. 200KB, which is 10% of the size). The thing is that while the space of *all* images has 2MB worth of "degrees of freedom" or "entropy", the space of all *interesting* images is much smaller, and can be stored using much less space, especially if one is willing to throw away some of the quality of the image. (Indeed, if one generates an image at random, one will almost certainly not get an interesting image; instead, one will just get random noise looking much like the static one can get on TV screens.)

How can one compress an image? There are many ways, some of which are rather technical, but let me try to give a non-technical (and slightly inaccurate) sketch of how it is done. It is quite typical for an image to have a large featureless component – for instance, in a landscape, up to half of the picture might be taken up by a monochromatic sky background. Suppose for instance that we locate a large square, say 100×100 pixels, which are all exactly the same colour – e.g. all white. Without compression, this square would take 10,000 bytes to store (using 8-bit grayscale); however, instead, one can simply record the dimensions and location of the square, and note a single colour with which to paint the entire square; this will require only four or five bytes in all to record, leading to a massive space saving. Now in practice, we don't get such an impressive gain in

compression, because even apparently featureless regions have some small colour variation between them. So, given a featureless square, what one can do is record the *average* colour of that square, and then subtract that average off from the image, leaving a small residual error. One can then locate more squares where the average colour is significant, and subtract those off as well. If one does this a couple times, eventually the only stuff left will be very small in magnitude (intensity), and not noticeable to the human eye. So we can throw away the rest of the image and record only the size, location, and intensity of the "significant" squares of the image. We can then reverse this process later and reconstruct a slightly lower-quality replica of the original image, which uses much less space.

Now, the above algorithm is not all that effective in practice, as it does not cope well with sharp transitions from one colour to another. It turns out to be better to work not with average colours in squares, but rather with average colour *imbalances* in squares – the extent to which the intensity on (say) the right half of the square is higher on average than the intensity on the left. One can formalise this by using the (two-dimensional) Haar wavelet system. It then turns out that one can work with "smoother" wavelet systems which are less susceptible to artefacts, but this is a technicality which we will not discuss here. But all of these systems lead to similar schemes: one represents the original image as a linear superposition of various "wavelets" (the analogues of the coloured squares in the preceding paragraph), stores all the significant (large magnitude) wavelet coefficients, and throws away (or "thresholds") all the rest. This type of "hard wavelet coefficient thresholding" compression algorithm is not nearly as sophisticated as the ones actually used in practice (for instance in the JPEG 2000 standard) but it is somewhat illustrative of the general principles in compression.

To summarise (and to oversimplify somewhat), the original 1024×2048 image may have two million degrees of freedom, and in particular if one wants to express this image in terms of wavelets then one would thus need two million different wavelets in order to reconstruct all images perfectly. However, the typical *interesting*image is very *sparse* or *compressible* in the wavelet basis: perhaps only a hundred thousand of the wavelets already capture all the notable features of the image, with the remaining 1.9 million wavelets only contributing a very small amount of "random noise" which is largely invisible to most observers. (This is not always the case: heavily *textured* images – e.g. images containing hair, fur, etc. – are not particularly compressible in the wavelet basis, and pose a challenge for image compression algorithms. But that is another story.)

Now, if we (or the camera) knew in advance which hundred thousand of the 2 million wavelet coefficients are going to be the important ones, then the camera could just measure those coefficients and not even bother trying to measure the rest. (It is possible to measure a single coefficient by applying a suitable "filter" or "mask" to the image, and making a single intensity measurement to what comes out.) However, the camera does not know which of the coefficients are going to be the key ones, so it must instead measure all 2 million pixels, convert the image to a wavelet basis, locate the hundred thousand dominant wavelet coefficients to keep, and throw away the rest. (This is of course only a caricature of how the image compression algorithm really works, but we will use it for sake of discussion.)

Now, of course, modern digital cameras work pretty well, and why should we try to improve on something which isn't obviously broken? Indeed, the above algorithm, in which one collects an enormous amount of data but only saves a fraction of it, works just fine for consumer photography. Furthermore, with data storage becoming quite cheap, it is now often feasible to use modern cameras to take many images with no compression whatsoever. Also, the computing power required to perform the compression is manageable, even if it does contribute to the notoriously battery-draining energy consumption level of these cameras. However, there are

non-consumer imaging applications in which this type of data collection paradigm is infeasible, most notably in sensor networks. If one wants to collect data using thousands of sensors, which each need to stay *in situ* for long periods of time such as months, then it becomes necessary to make the sensors as cheap and as low-power as possible – which in particular rules out the use of devices which require heavy computer processing power at the sensor end (although – and this is important – we are still allowed the luxury of all the computer power that modern technology affords us at the *receiver* end, where all the data is collected and processed). For these types of applications, one needs a data collection paradigm which is as "dumb" as possible (and which is also robust with respect to, say, the loss of 10% of the sensors, or with respect to various types of noise or data corruption).

This is where *compressed sensing* comes in. The main philosophy is this: if one only needs a 100,000 components to recover most of the image, why not just take a 100,000 measurements instead of 2 million? (In practice, we would allow a safety margin, e.g. taking 300,000 measurements, to allow for all sorts of issues, ranging from noise to aliasing to breakdown of the recovery algorithm.) In principle, this could lead to a power consumption saving of up to an order of magnitude, which may not mean much for consumer photography but can be of real importance in sensor networks.

But, as I said before, the camera does not know in advance which hundred thousand of the two million wavelet coefficients are the important ones that one needs to save. What if the camera selects a completely different set of 100,000 (or 300,000) wavelets, and thus loses all the interesting information in the image?

The solution to this problem is both simple and unintuitive. It is to make 300,000 measurements which are *totally unrelated* to the wavelet basis – despite all that I have said above regarding how this is the best basis in which to view and compress images. In fact, the best types of measurements to make are (pseudo-)*random*measurements – generating, say, 300,000 random "mask" images and measuring the extent to which the actual image resembles each of the masks. Now, these measurements (or "correlations") between the image and the masks are likely to be all very small, and very random. But – and this is the key point – each one of the 2 million possible wavelets which comprise the image will generate their own distinctive "signature" inside these random measurements, as they will correlate positively against some of the masks, negatively against others, and be uncorrelated with yet more masks. But (with overwhelming probability) each of the 2 million signatures will be distinct; furthermore, it turns out that arbitrary linear combinations of up to a hundred thousand of these signatures will still be distinct from each other (from a linear algebra perspective, this is because two randomly chosen 100,000-dimensional subspaces of a 300,000 dimensional ambient space will be almost certainly disjoint from each other). Because of this, it is possible *in principle* to recover the image (or at least the 100,000 most important components of the image) from these 300,000 random measurements. In short, we are constructing a linear algebra analogue of a hash function.

There are however two technical problems with this approach. Firstly, there is the issue of noise: an image is not perfectly the sum of 100,000 wavelet coefficients, but also has small contributions from the other 1.9 million coefficients also. These small contributions could conceivably disguise the contribution of the 100,000 wavelet signatures as coming from a completely unrelated set of 100,000 wavelet signatures; this is a type of "aliasing" problem. The second problem is how to use the 300,000 measurements obtained to recover the image.

Let us focus on the latter problem first. If we knew which 100,000 of the 2 million wavelets involved were, then we could use standard linear algebra methods (Gaussian elimination, least squares, etc.) to recover the signal. (Indeed, this is one of the great advantages of linear encodings – they are much easier to invert than nonlinear

ones. Most hash functions are practically impossible to invert – which is an advantage in cryptography, but not in signal recovery.) However, as stated before, we don't know in advance which wavelets are involved. How can we find out? A naive least-squares approach gives horrible results which involve all 2 million coefficients and thus lead to very noisy and grainy images. One could perform a brute-force search instead, applying linear algebra once for each of the possible set of 100,000 key coefficients, but this turns out to take an insanely impractical amount of time (there are roughly $10^{170,000}$ combinations to consider!) and in any case this type of brute-force search turns out to be NP-complete in general (it contains problems such as subset-sum as a special case). Fortunately, however, there are two much more feasible ways to recover the data:

Matching pursuit: locate a wavelet whose signature seems to correlate with the data collected; remove all traces of that signature from the data; and repeat until we have totally "explained" the data collected in terms of wavelet signatures.

Basis pursuit (or l^1 minimisation): Out of all the possible combinations of wavelets which would fit the data collected, find the one which is "sparsest" in the sense that the total sum of the magnitudes of all the coefficients is as small as possible. (It turns out that this particular minimisation tends to force most of the coefficients to vanish.) This type of minimisation can be computed in reasonable time via convex optimisation methods such as the simplex method.

Note that these image recovery algorithms do require a non-trivial (though not ridiculous) amount of computer processing power, but this is not a problem for applications such as sensor networks since this recovery is done on the receiver end (which has access to powerful computers) rather than the sensor end (which does not).

There are now rigorous results which show that these approaches can reconstruct the original signals perfectly or almost-perfectly with very high probability of success, given various compressibility or sparsity hypotheses on the original image. The matching pursuit algorithm tends to be somewhat faster, but the basis pursuit algorithm seems to be more robust with respect to noise. Exploring the exact range of applicability of these methods is still a highly active current area of research. (Sadly, there does not seem to be an application to $P \neq NP$; the type of sparse recovery problems which are NP-complete are the total opposite (as far as the measurement matrix is concerned) with the type of sparse recovery problems which can be treated by the above methods.)

As compressed sensing is still a fairly new field (especially regarding the rigorous mathematical results), it is still a bit premature to expect developments here to appear in actual sensors. However, there are proof-of-concept prototypes already, most notably the single-pixel camera developed at Rice.

Finally, I should remark that compressed sensing, being an abstract mathematical idea rather than a specific concrete recipe, can be applied to many other types of contexts than just imaging. Some examples include:

Magnetic resonance imaging (MRI). In medicine, MRI attempts to recover an image (in this case, the water density distribution in a human body) by taking a large but finite number of measurements (basically taking a discretised Radon transform (or x-ray transform) of the body), and then reprocessing the data. Because of the large number of measurements needed, the procedure is lengthy for the patient. Compressed sensing techniques can reduce the number of measurements required significantly, leading to faster imaging (possibly even to real-time imaging, i.e. MRI videos rather than static MRI). Furthermore, one can trade off the number of measurements against the quality of the image, so that by using the same number of measurements as one traditionally does, one may be able to get much finer scales of resolution.

Astronomy. Many astronomical phenomena (e.g. pulsars) have various frequency oscillation behaviours which make them very sparse or compressible in the frequency domain. Compressed sensing techniques then allow one to measure these phenomena in the time domain (i.e. by recording telescope data) and being able to reconstruct the original signal accurately even from incomplete and noisy data (e.g. if weather, lack of telescope time, or simply the rotation of the earth prevents a complete time-series of data).

Linear coding. Compressed sensing also gives a simple way for multiple transmitters to combine their output in an error-correcting way, so that even if a significant fraction of the output is lost or corrupted, the original transmission can still be recovered. For instance, one can transmit 1000 bits of information by encoding them using a random linear code into a stream of 3000 bits; and then it will turn out that even if, say, 300 of the bits (chosen adversarially) are then corrupted, the original message can be reconstructed perfectly with essentially no chance of error. The relationship with compressed sensing arises by viewing the corruption itself as the sparse signal (it is only concentrated on 300 of the 3000 bits).

Many of these applications are still only theoretical, but nevertheless the potential of these algorithms to impact so many types of measurement and signal processing is rather exciting. From a personal viewpoint, it is particularly satisfying to see work arising from pure mathematics (e.g. estimates on the determinant or singular values of Fourier minors) end up having potential application to the real world.