CS264: Beyond Worst-Case Analysis
Lecture #9: A Taste Of Compressive Sensing*

Tim Roughgarden†

October 22, 2014

1 Preamble

The last several lectures proved that polynomial-time exact recovery is possible for instances of several NP-hard problems that satisfy some type of stability condition. Lecture #7 showed that the single-link++ algorithm, which searches over a restricted set of feasible solutions, and thus can return a suboptimal solution in worst-case instances, recovers the optimal clustering in stable k-median instances. Last lecture we showed that a linear programming relaxation, whose optimal solution is generally not an integral solution, recovers the optimal multiway cut in stable instances. Today's lecture continues this theme of exact recovery via linear programming, but in a different problem domain: systems of linear equations.

2 Sparse Solutions to Underdetermined Linear Systems

2.1 The Problem

Consider a linear system $Ax = b$, where the number m of rows of A is less than the number n of columns (Figure 1). Such a system is *underdetermined*: if it has any solutions, then it has an entire $(n - m)$-dimensional subspace of solutions. One useful way to think about this is that the unknown vector x has n "degrees of freedom": each (linearly independent) constraint of A pins down one of these degrees, and n constraints are necessary to determine x uniquely. The goal of this lecture is recover the "most meaningful" solution to an underdetermined linear system.

Call a vector $\mathbf{x} \in \mathcal{R}^n$ *k-sparse* if it has at most k non-zero entries — if $|\text{supp}(\mathbf{x})| \leq k$, where $\text{supp}(\mathbf{x})$ denotes the support of $\mathbf{x}$. For this lecture, you might want to think of k as $\sqrt{n}$

†Department of Computer Science, Stanford University, 474 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: tim@cs.stanford.edu.

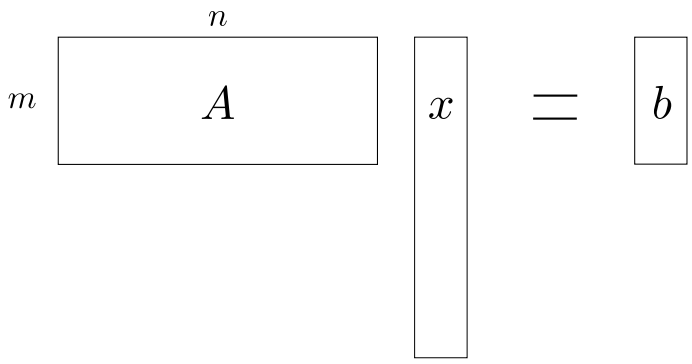

Figure 1: An underdetermined system of linear equations. The $m \times n$ matrix A and vector b are known, while x is unknown.

or $\log n$. We adopt the bias that sparser solutions are "more meaningful." This is a useful bias in several problem domains, at least if we allow two extensions. The first extension is to also favor vectors that are "approximately" k-sparse; for example, vectors $\mathbf{x}$ for which most of the ℓ_1 norm $\sum_{i=1}^{n} |x_i|$ is concentrated on k coordinates. The second extension is to allow vectors to be sparse in some known basis other than the standard one — the Fourier and wavelet bases being common examples. For example, real-world images and signals are typically approximately sparse in a suitable basis. For simplicity, this lecture considers only exact recovery of vectors that are exactly sparse in the standard basis. The results and techniques developed extend to the approximate recovery of vectors that are approximately sparse in some other basis; see also Homework #5.

We study the computational problem of recovering in polynomial time a "ground truth" k-sparse solution $\mathbf{z}$ to a linear system $Ax = b$. This is a fundamental problem in the field of "compressive sensing," which has been studied intensely over the past decade. The typical viewpoint in compressive sensing is to think of the solution $\mathbf{z}$ as a "signal" that one is trying to recover through "linear measurements." If $\mathbf{z}$ is a solution to the system $Ax = b$, then we can think of each row i of the constraint system as telling us something (a "measurement") about $\mathbf{z}$, namely the value (b_i) of its inner product with a vector (A_i). The more linearly independent rows of the system, the more we know about the unknown signal $\mathbf{z}$. We've already noted that with no assumptions about $\mathbf{z}$, n rows/measurements are necessary and sufficient to reconstruct $\mathbf{z}$. What if we assume that $\mathbf{z}$ is not a worst-case signal — that it has additional structure, such as sparsity? Perhaps then we get away with many fewer than n rows without losing the ability to uniquely reconstruct $\mathbf{z}$.

This perspective is rather different from the usual one in the study of algorithms. Rather than treating the input A and b to the computational problem as being handed down from the sky, we are essentially asking: how can we *design* the constraint matrix A so that we get the properties that we want, namely a small linear system and the ability to efficiently reconstruct the unknown signal $\mathbf{z}$ (from A and $b = A\mathbf{z}$), whatever it may be. There's a useful analogy with error-correcting codes, where the usual goal is to design a code (equivalently for

linear codes, a matrix A) with desirable properties, such as large distance and fast encoding and decoding algorithms.

2.2 Applications

The compressive sensing approach, where the matrix A is designed a priori, is a good match for some applications. One buzzword you can look up and read more about is the "single-pixel camera." The standard approach to taking pictures is to first take a high-resolution picture in the "standard basis" — e.g., a light intensity for each pixel — and then to compress the picture later (via software). Because real-world images are typically sparse in a suitable basis, they can be compressed a lot. The compressive sensing approach asks, then why not just capture the image directly in a compressed form — in a representation where its sparsity shines through? For example, one can store random linear combinations of light intensities (implemented via suitable mirrors) rather than the light intensities themselves. This idea leads to a reduction in the number of pixels needed to capture an image at a given resolution. Another application of compressive sensing is in MRI. Here, decreasing the number of measurements decreases the time necessary for a scan. Since a patient needs to stay motionless during a scan — in some cases, not even breathing — shorter scan times can be a pretty big deal.

2.3 Lines in the Sand

Recovering a target k-sparse solution $\mathbf{z}$ to an underdetermined linear system $Ax = b$ is possible only with assumptions on the system. For example, if the system has only one row — pinning down only one dimension of a solution — then it's hopeless to figure out what $\mathbf{z}$ is. Since $\mathbf{z}$ is k-sparse, one's first hope might be that roughly k rows might be enough to reconstruct $\mathbf{z}$. It turns out, however, that some dependence on the dimension n is also necessary. Intuitively, we need to at least differentiate between the $\binom{n}{k}$ supports that $\mathbf{z}$ could have. Assuming that each row of the linear system gives us only a "constant number of bits" about the unknown sparse solution $\mathbf{z}$, we would expect that at least $\log \binom{n}{k} \approx k \log n$ linearly independent rows are necessary.[1] When k is sublinear in n, this is still a big improvement over the n rows necessary to reconstruct an arbitrary unknown solution.

The matrix A also shouldn't be too sparse. For example, if $k = n^{1/4}$ and A consists of $m \approx k \log n$ (linearly independent) rows each with one "1" and the rest zeroes, then Ax will be a vector of mostly zeroes for most k-sparse vectors x. In this case, there is no way to distinguish between the large set of sparse solutions to the linear system. Thus, in addition to requiring that m be sufficiently large, we need that A is sufficiently dense.

[1] This back-of-the-envelope calculation is roughly correct, with $m = \Theta(k \log \frac{n}{k})$ rows necessary and sufficient for the guaranteed recovery of approximately k-sparse solutions. The proof is non-trivial; see [1].

3 ℓ_1-Minimization via Linear Programming

Consider a constraint matrix A that is sufficiently big and dense, so that a target k-sparse solution can be reconstructed exactly in principle. There remains the question of how to perform this reconstruction efficiently. Indeed, computing the sparsest solution to a linear system is an NP-hard problem [2]. Following last lecture, our goal is to understand when a natural linear programming relaxation for this problem hands us back the correct solution on a silver platter.

Minimizing the support size of a solution — the "ℓ_0 norm" — to a linear system is sometimes called "ℓ_0-minimization." The linear programming relaxation simply replaces the ℓ_0 norm with the ℓ_1 norm, where $\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$. Formally, our constraints are

$$Ax = b$$

together with

$$y_i \geq x_i \quad \text{and} \quad y_i \geq -x_i \tag{1}$$

for $i = 1, 2, \ldots, n$. That is, constraint (1) forces $y_i \geq |x_i|$, and equality will hold for every optimal solution under the objective function

$$\sum_{i=1}^n y_i.$$

This linear program (LP) computes the solution to $Ax = b$ with the smallest ℓ_1 norm.

There is no reason to think that the heuristic of solving (LP) would ever be useful for recovering a sparse solution. Why wouldn't the optimal solution to (LP) be a vector that is non-zero but very small in every single coordinate? It was first observed empirically that ℓ_1-minimization was unreasonably effective at recovering exact solutions, and this motivated the theory of compressive sensing. The goal of this lecture is to prove the following representative result in the area.

Theorem 3.1 *Let $m = \Omega(k \log n)$ and A a "random" $m \times n$ matrix. Then with high probability, for every k-sparse vector $\hat{\mathbf{x}}$ with $b = A\hat{\mathbf{x}}$, $\hat{\mathbf{x}}$ is the unique optimal solution of (LP).*

Theorem 3.1 holds for several different notions of a "random matrix" — for example, entries can be i.i.d. ± 1 or i.i.d. standard Gaussians. The theorem guarantees that, for almost all such random matrices, an arbitrary "ground truth" k-sparse vector can be recovered in polynomial time via linear programming.

4 Proof of Theorem 3.1

The proof of Theorem 3.1, like most of our exact recovery results, has two conceptual parts. First, we develop sufficient conditions under which the outcomes of ℓ_0-minimization and ℓ_1-minimization are the same. Second, we need to show that a random constraint matrix A satisfies these sufficient conditions with high probability. We'll give complete proofs for the first part, and make some comments about the second part.

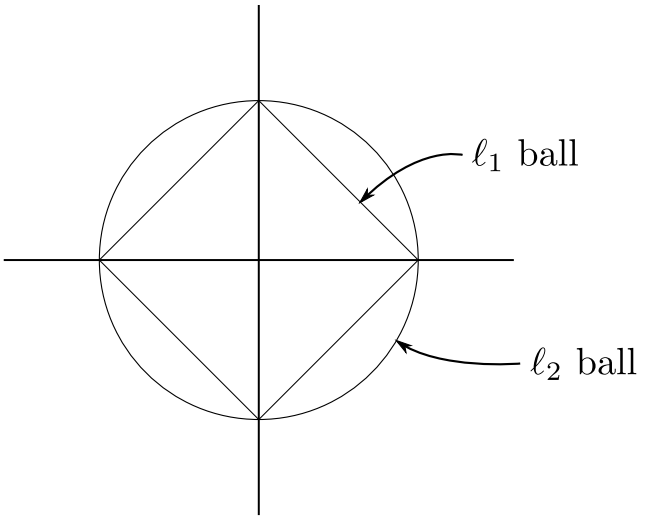

Figure 2: The ℓ_1 and ℓ_2 balls in two dimensions.

4.1 Preliminaries: ℓ_1 vs. ℓ_2 Norms

Before discussing the proof of Theorem 3.1, we need to recall some basics about the geometry of $\mathcal{R}^n$ under different norms. Recall that for $\mathbf{x} \in \mathcal{R}^n$,

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$$

and

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1} x_i^2 \right)^{1/2}.$$

Geometrically, the ℓ_2 ball is a sphere while the ℓ_1 ball is a (high-dimensional analog of a) diamond. See Figure 2. Recall that for every $\mathbf{x} \in \mathcal{R}^n$, we have

$$\frac{1}{\sqrt{n}} \|\mathbf{x}\|_1 \leq \|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1; \tag{2}$$

that is, the ℓ_1 norm can only be larger than the ℓ_2 norm, but by at most a $\sqrt{n}$ factor. The two norms are the same when there is at most one non-zero component (e.g., for the standard basis vectors). The ratio between the two norms is the largest when all of the components of $\mathbf{x}$ are equal.

Ultimately, we're going to want to claim that "most" vectors are more like an all-1s vector than a standard basis vector — i.e., the ratio between the ℓ_1 and ℓ_2 norm of a vector is "typically" close to $\sqrt{n}$. The following definition helps us make this precise.

Definition 4.1 A vector $\mathbf{x} \in \mathcal{R}^n$ is *c-spread out* if

$$\|\mathbf{x}\|_2 \leq \frac{c}{\sqrt{n}}\|\mathbf{x}\|_1.$$

Note that c is somewhere between 1 (if all components are equal) and $\sqrt{n}$ (for a standard basis vector). For a set of c-spread out vectors with c small, the ℓ_1 and ℓ_2 norms are basically the same, up to a scaling factor of $\sqrt{n}$. We'll be interested in the case where $c \approx \sqrt{n/k}$, with $\|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1/\sqrt{k}$.

Intuitively, if $\mathbf{x}$ is c-spread out for reasonably small c, then the ℓ_1 weight of $\mathbf{x}$ cannot be concentrated on just a few coordinates. The following lemma makes this intuition precise.

Lemma 4.2 *Let $\mathbf{x} \in \mathcal{R}^n$ be c-spread out and $I \subseteq \{1, 2, \ldots, n\}$. Then*

$$\|\mathbf{x}_I\|_1 \leq c\sqrt{\frac{|I|}{n}} \cdot \|\mathbf{x}\|_1, \tag{3}$$

where $\mathbf{x}_I$ denotes $\mathbf{x}$ with all components outside I zeroed out.

Proof: Project to the coordinates in I, apply (2) and then the definition of c-spread out to derive

$$\|\mathbf{x}_I\|_1 \leq \sqrt{|I|} \cdot \|\mathbf{x}_I\|_2 \leq \sqrt{|I|} \cdot \|\mathbf{x}\|_2 \leq c\sqrt{\frac{|I|}{n}} \cdot \|\mathbf{x}\|_1.$$

■

For example, if $|I| = o(n)$ and $c = O(1)$, then Lemma 4.2 states that a sublinear number of coordinates cannot capture a constant fraction of the ℓ_1 weight of $\mathbf{x}$. For us, the relevant parameters are $|I| = k$ and $c \approx \frac{1}{4}\sqrt{n/k}$, where Lemma 4.2 states that no k coordinates capture a large fraction of the ℓ_1 weight.

4.2 The Sufficient Condition

We next state the key property of constraint matrices that is sufficient for the conclusion of Theorem 3.1.

Definition 4.3 An $m \times n$ matrix A is *c-good* if every non-zero vector of the kernel $\ker A = \{\mathbf{y} : A\mathbf{y} = \mathbf{0}\}$ is c-spread out.

Alternative terminology for Definition 4.3 is that $\ker A$ should be an "almost Euclidean subspace." Geometrically, Definition 4.3 requires that $\ker A$ is not too close to any basis vector, or any linear combination of a small number of basis vectors.

If you're familiar with error-correcting codes (see also Lectures #11 and #12), then the following analogy should be helpful. Recall that linear codes can be defined as the kernel of a matrix (e.g., for binary codes, over the field $\mathbb{F}_2$). A good code has large Hamming distance between every pair of code words, and for linear codes this is equivalent to the

property that every non-zero codeword has large (i.e., "spread out") Hamming weight. Thus the "almost Euclidean subspace property" can be viewed as a strengthening of the "large distance property" — every non-zero vector in $\ker A$ should not only have its ℓ_0 weight spread out over a large number of coordinates, but also its ℓ_1 weight.

We state without proof the following fact.

Fact 4.4 *For $m = \Omega(k \log n)$, a random matrix A is $\frac{1}{4}\sqrt{\frac{n}{k}}$-good with high probability.*

The matrix A is "random" in the same sense as in Theorem 3.1 — for example, with i.i.d. ± 1 or standard Gaussian entries. Homework #5 asks you to prove a much easier but still illuminating version of Fact 4.4, in the context of error-correcting codes. Fact 4.4 can be proved without resorting to overly heavy machinery — see the blog posts [4, 3] for an accessible proof — but doing so is outside the scope of these notes.

4.3 Completing the Proof

Given Fact 4.4, we only need to prove that the $\frac{1}{4}\sqrt{\frac{n}{k}}$-good property is a sufficient condition for the linear programming relaxation (LP) to recover a k-sparse vector. This is equivalent to showing that, if $\hat{\mathbf{x}}$ is k-sparse with $A\hat{\mathbf{x}} = b$, and also $A\mathbf{w} = b$, then $\|\mathbf{w}\|_1 > \|\hat{\mathbf{x}}\|_1$. In this event, $\hat{\mathbf{x}}$ minimizes the ℓ_1 norm over all solutions to $Ax = b$, as desired.

The derivation below formalizes the following intuition. Every solution $\mathbf{w}$ to $Ax = b$ differs from $\hat{\mathbf{x}}$ by an element $(\mathbf{w} - \hat{\mathbf{x}})$ in the kernel of A. Since A is good, all such vectors are spread out. Since $\hat{\mathbf{x}}$ is k-sparse, not much of the ℓ_1 weight of $\hat{\mathbf{x}}$ and $(\mathbf{w} - \hat{\mathbf{x}})$ can cancel out. This leaves $\mathbf{w}$ with a lot of ℓ_1 weight, completing the proof.

Formally, write $\mathbf{w} = \hat{\mathbf{x}} + \mathbf{y}$, with $\mathbf{y} \in \ker A$ and hence $\frac{1}{4}\sqrt{\frac{n}{k}}$-spread out. (This is where we use that A is $\frac{1}{4}\sqrt{\frac{n}{k}}$-good.) Let I denote the support of $\hat{\mathbf{x}}$ (so $|I| \leq k$) and J the other components. As before, $\mathbf{z}_I$ is the vector obtained from $\mathbf{z}$ be zeroing out all coordinates other than those in I. We can write

$$\|\mathbf{w}\|_1 = \|(\hat{\mathbf{x}} + \mathbf{y})_I\|_1 + \underbrace{\|(\hat{\mathbf{x}} + \mathbf{y})_J\|_1}_{=\|\mathbf{y}_J\|_1}.$$

By the Triangle Inequality of the ℓ_1 norm (i.e., $\|a + b\|_1 \leq \|a\|_1 + \|b\|_1$), we have

$$\|(\hat{\mathbf{x}} + \mathbf{y})_I\|_1 \geq \underbrace{\|\hat{\mathbf{x}}_I\|_1}_{=\|\hat{\mathbf{x}}\|_1} - \underbrace{\|\mathbf{y}_I\|_1}_{\text{small since } \mathbf{y} \text{ spread out}},$$

with equality holding when $\hat{\mathbf{x}}, \mathbf{y}$ have opposite signs in I and $\hat{\mathbf{x}}_I$ cancels out all of the ℓ_1 weight of $\mathbf{y}_I$. (The lower bound $\|\mathbf{y}_I\|_1 - \|\hat{\mathbf{x}}_I\|_1$ also holds, but we won't need it.) Thus,

$$\|\mathbf{w}\|_1 \geq \|\hat{\mathbf{x}}_I\|_1 - \|\mathbf{y}_I\|_1 + \underbrace{\|\mathbf{y}_J\|_1}_{=\|\mathbf{y}\|_1 - \|\mathbf{y}_I\|_1} = \|\hat{\mathbf{x}}\|_1 - 2\|\mathbf{y}_I\|_1 + \|\mathbf{y}\|_1.$$

By Lemma 4.2,

$$\|\mathbf{y}_I\|_1 \leq \frac{1}{4}\sqrt{\frac{n}{k}} \cdot \sqrt{frac|I|n} \cdot \|\mathbf{y}\|_1 \leq \frac{1}{4}\|\mathbf{y}\|_1,$$

and hence $\|\mathbf{w}\|_1 > \|\hat{\mathbf{x}}\|_1$, as desired. This completes the proof of Theorem 3.1.

References

[1] K. Do Ba, P. Indyk, E. Price, and D. P. Woodruff. Lower bounds for sparse recovery. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1190–1197, 2010.

[2] L. Khachiyan. On the complexity of approximating extremal determinants in matrices. *Journal of Complexity*, 11(1):138–153, 1995.

[3] J. R. Lee. Kernels of random sign matrices. TCS Math Blog. May 8, 2008.

[4] J. R. Lee. The pseudorandom subspace problem. TCS Math Blog. May 4, 2008.