MIT 6.893: Algorithms and Signal Processing Fall 2014

Lecture 1 — September 8, 2014

Prof. Piotr Indyk *Scribe: Christopher Musco*

1 Course Content

This course covers applications of algorithmic tools (e.g. dynamic programming, hashing, randomization, sketching) to problems in signal processing (e.g. Fourier transform, sparse approximation, compressive sensing, Wavelet transform). Specific topics will include:

- Sparse Fourier Transform
- Non-uniform Fourier Transform
- Prony/Matrix Pencil Method
- Other Sparse Approximation Methods (e.g., Wavelet Transform)
- (Model-based) Compressive Sensing
- Distance/Vector Quantization

2 Sparse Approximations

Most of this course will be dedicated to various methods for finding *sparse approximations* of signals. Generally, given a signal $\mathbf{a} \in \mathbb{C}^n$, we will focus on finding a sparse approximate representation of the signal that can be parameterized by just $k \ll n$ coefficients. One high level approach to this problem is *transform coding*. We could imagine approximating $\mathbf{a}$ by simply selecting its largest k entries. However, this approach often fails because $\mathbf{a}$ may not be sparse in its original basis. So, the signal is multiplied by an $n \times n$ orthonormal (i.e. unitary) matrix $\mathbf{T}$ to form $\mathbf{Ta}$. We then compute $\mathbf{H}_k(\mathbf{Ta})$ by zeroing out all but the top k coefficients of $\mathbf{Ta}$. The final sparse approximation is given by $\mathbf{T}^{-1}\mathbf{H}_k(\mathbf{Ta})$. Both the Fourier transform and Wavelet transform are examples of transform coding.

Another approach to sparse approximation is piecewise approximation, which includes histograms (piecewise constant approximation) and piecewise polynomial approximation. We will discuss algorithms for these methods as well.

3 Fourier Transform

The Fourier Tranform maps a signal in the time domain, $\mathbf{a} = [a_0, \ldots, a_{n-1}]$, to a signal in the frequency domain, $\hat{\mathbf{a}} = [\hat{a}_0, \ldots, \hat{a}_{n-1}]$. It is computed by

$$\hat{a}_u = \frac{1}{n} \sum_{j=0}^{n-1} a_j e^{-\frac{2\pi i}{n} uj}. \tag{1}$$

We will denote the n^{th} root of unity as $\omega = \omega_n = e^{\frac{2\pi i}{n}}$. Thus, $\hat{a}_u = \frac{1}{n}\sum_{j=0}^{n-1} a_j \omega^{-uj}$. In matrix form

$$\hat{\mathbf{a}} = \mathbf{F}\mathbf{a}, \tag{2}$$

where

$$\mathbf{F}_{u,j} = \frac{1}{n}\omega^{-uj} \tag{3}$$

is the Fourier matrix. $\mathbf{F}^{-1}_{j,u} = \omega^{uj}$ and naturally $\mathbf{a} = \mathbf{F}^{-1}\hat{\mathbf{a}}$. Note that $\mathbf{F}$ and $\mathbf{F}^{-1}$ are sometimes normalized differently to have entries of $\frac{1}{\sqrt{n}}\omega^{\pm uj}$. This makes both matrices unitary.

4 Sparse Fourier Transform

Suppose we wish to use the Fourier transform for transform coding. The Fast Fourier Transform (FFT) algorithm can be used to compute $\hat{\mathbf{a}} = \mathbf{F}\mathbf{a}$ in $O(n \log n)$ time (it can also be used to compute $\mathbf{a} = \mathbf{F}^{-1}\hat{\mathbf{a}}$ in the same time). $\mathbf{H}_k(\hat{\mathbf{a}})$ can then be computed in $O(n)$ time using standard selection algorithms. While fast with respect to our input size, this approach inherently computes extra information (Fourier coefficients that are discarded). Thus, we might hope for a sub-linear time algorithm that only computes the top coefficients of $\hat{\mathbf{a}}$ in a single shot.

In fact, several exist. The first algorithms of this type were developed in [5, 3, 4]. The most recent *Sparse Fourier Transform* can approximately compute the k largest entries of $\hat{\mathbf{a}}$ in $O(k \log n \log(n/k))$ time [2]. Note that this running time is always not greater than $O(n \log n)$. Some variants run even faster and different versions of the algorithm come with different guarantees and input assumptions.

4.1 $k = 1$, exactly sparse case

We begin by considering the simplest case, $k = 1$, where we are only interested in finding $\hat{\mathbf{a}}$'s top coefficient. Actually, to start off, lets make the further assumption that $\hat{\mathbf{a}}$ is *exactly* 1-sparse. In other words $\mathbf{a}$ has only a single non-zero frequency component. If $\hat{a}_u$ is the non-zero entry in $\hat{\mathbf{a}}$, our goal is to find both u and the entry's magnitude. This can be done using a simple deterministic 2 point sampling scheme. Select a_0 and a_1 and note that

$$a_0 = F^{-1}_{0,*}\hat{\mathbf{a}} = \omega^0 \hat{a}_u = \hat{a}_u, \tag{4}$$

$$a_1 = F^{-1}_{1,*}\hat{\mathbf{a}} = \omega^u \hat{a}_u. \tag{5}$$

Combining the two equations, we see that $\omega^u = a_1/a_0$. So we can just compute the ratio of our samples and solve for u based on the ratio's angle in the complex plane. Note that, when $\hat{\mathbf{a}}$ is exactly 1 sparse, our Fourier transform inherently cannot be symmetric. Thus, in general, our signal $\mathbf{a}$ will not be real and the ratio a_1/a_0 likely has a complex component.

Obviously this method is very fast (constant time), but unfortunately it relies heavily on the assumption that $\hat{\mathbf{a}}$ is exactly 1-sparse.

4.2 $k = 1$, noisy case

When $\hat{\mathbf{a}}$ is not exactly sparse, we will have to apply a more robust method. Furthermore, it no longer makes sense to ask for the index u exactly, especially when we move on to higher k – we

may be able to obtain a good approximation to $\mathbf{a}$ without returning the indices of exactly $\hat{\mathbf{a}}$'s k largest components (as long as we return k large ones).

Thus, we introduce another measure of success, the ℓ_2/ℓ_2 guarantee. Suppose $\hat{\mathbf{a}}^*$ is the best k-sparse approximation to $\hat{\mathbf{a}}$ – i.e. the vector containing $\hat{\mathbf{a}}$'s largest k entries and thus minimizing $\|\hat{\mathbf{a}} - \hat{\mathbf{a}}^*\|_2$. We want to return some k-sparse vector $\hat{\mathbf{a}}'$ such that

$$\|\hat{\mathbf{a}} - \hat{\mathbf{a}}'\|_2 \leq C\|\hat{\mathbf{a}} - \hat{\mathbf{a}}^*\|_2, \tag{6}$$

for some approximation factor C. Recall that for complex vectors, $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^*\mathbf{x}} = \sqrt{\sum_i x_i \bar{x}_i}$.

This guarantee is of course only meaningful when $C\|\hat{\mathbf{a}} - \hat{\mathbf{a}}^*\|_2 < \|\hat{\mathbf{a}}\|_2$, or else we could just return the all zeros vector. Equivalently, we will require that $\sum_{u' \neq u} |\hat{a}_{u'}|^2 < \epsilon |\hat{a}_u|^2$ for a suitable ϵ.

We describe an algorithm (from [1]) for the ℓ_2/ℓ_2 guarantee that is more robust than the 2 point sampling algorithm. However, for now, again assume that our vector is exactly 1-sparse. I.e. assume $a_j = \hat{a}_u \omega^{uj}$. Noise is dealt with later on. We will find u bit-by-bit, from its lowest order to highest order bit. Its lowest order bit, b, is simply ($u \bmod 2$) – $u = 2v + b$ for some v. To find the bit, note that

$$a_0 = \hat{a}_u, \tag{7}$$

$$a_{n/2} = \hat{a}_u \omega^{un/2} = \hat{a}_u \omega^{2vn/2 + bn/2} = \hat{a}_u (-1)^b. \tag{8}$$

Given the above equations, we can test whether $b = 0$ by just checking that:

$$|a_0 - a_{n/2}| < |a_0 + a_{n/2}|, \tag{9}$$

which always holds when a_0 and $a_{n/2}$ have the same sign. Actually, we're going to do something slightly different, the reason for which will become clear later when we deal with noise. Instead of comparing a_0 and $a_{n/2}$, lets compare two signal samples that are still $n/2$ apart, but are randomly shifted. I.e. choose r uniformly in $[0, \ldots, n-1]$ and again note that

$$a_r = \hat{a}_u \omega^{ur}, \tag{10}$$

$$a_{r+n/2} = \hat{a}_u \omega^{u(r+n/2)} = \hat{a}_u \omega^{un/2} \omega^{ur} = \hat{a}_u (-1)^b \omega^{ur}. \tag{11}$$

Here $r + n/2$ really denotes ($r + n/2 \bmod n$). So now we just set $b = 0$ whenever

$$|a_r - a_{r+n/2}| < |a_r + a_{r+n/2}|. \tag{12}$$

How about finding the next bit? It turns out that we can always reduce to the case when $b = 0$. The *time shift theorem* (which should be clear from the definition of the Fourier transform) says that, if we adjust $\mathbf{a}$ so that $a_j \to a_j \omega^{bj}$, then $\hat{\mathbf{a}}$ adjusts so that $\hat{a}_u \to \hat{a}_{u-b}$. Thus, if b is 1, we simply replace our signal with $\mathbf{a}$ such that $a_j \to a_j \omega^j$ and proceed (u will now have a final bit of 0).

Okay, so now $u = 2v$ for some v and our goal becomes to find the lowest order bit of v.

$$a_j = \hat{a}_u \omega^{uj} = \hat{a}_{2v} \omega^{2vj} = \hat{a}'_v e^{\frac{2\pi i}{n/2} vj}, \tag{13}$$

Well now this should just look like an inverse Fourier transform of length $n/2$. Specifically, consider the vector $\mathbf{a}' = \left[a_0, \ldots, a_{n/2-1}\right]$. Finding the lowest order bit of v is exactly equivalent to finding u for $\mathbf{a}'$. Thus, we can repeat our methodology for the first bit to recover another. Our test is

$$|a_r - a_{r+n/4}| < |a_r + a_{r+n/4}|. \tag{14}$$

Combining this iteration with the adjustments we make to $\mathbf{a}$ as we go (to assume the prior bit is 0) our final set of tests are:

- Bit $b_0 \to 0$ if $|a_r - a_{r+n/2}| < |a_r + a_{r+n/2}|$
- Bit $b_1 \to 0$ if $|a_r\omega^{rb_0} - a_{r+n/4}\omega^{(r+n/4)b_0}| < |a_r\omega^{rb_0} + a_{r+n/4}\omega^{(r+n/4)b_0}|$
- Bit $b_2 \to 0$ if $|a_r\omega^{r(b_0+2b_1)} - a_{r+n/8}\omega^{(r+n/8)(b_0+2b_1)}| < |a_r\omega^{r(b_0+2b_1)} + a_{r+n/8}\omega^{(r+n/8)(b_0+2b_1)}|$
- Bit $b_3 \ldots$

In total, we require $O(\log n)$ samples to identify u and our running time is $O(\log n)$. Note that we can reuse our random r at each level of the recovery process.

Now all we have left to do is show that the recursive algorithm is robust to noise. When $\mathbf{a}$ isn't exactly 1-sparse, all of the equations we used above change to

$$a_j = \hat{a}_u\omega^{uj} + \sum_{u' \neq u} \hat{a}_{u'}\omega^{u'j}. \tag{15}$$

Denote the noise term for entry a_j as $\mu_j = \sum_{u' \neq u} \hat{a}_{u'}\omega^{u'j}$ and the entire noise vector as $\boldsymbol{\mu}$. $\boldsymbol{\mu} = \mathbf{F}^{-1}\hat{\mathbf{a}}_{u=0}$ where $\hat{\mathbf{a}}_{u=0}$ is $\hat{\mathbf{a}}$ with the u^{th} entry zeroed out. Under our chosen scaling, the inverse Fourier transform is a unitary matrix scaled by $\sqrt{n}$. As such, $\|\boldsymbol{\mu}\|_2^2 = n\|\hat{\boldsymbol{a}}_{u=0}\|_2^2$.

So, if we choose r at random from $[0, \ldots n-1]$,

$$\mathop{\mathbb{E}}_{r}\left[\mu_r^2\right] = \|\hat{\boldsymbol{a}}_{u=0}\|_2^2 = \sum_{u' \neq u} |\hat{a}_{u'}|^2. \tag{16}$$

Since $\sum_{u' \neq u} |\hat{a}_{u'}|^2 < \epsilon|\hat{a}_u|^2$, it follows that $\mathbb{E}_r\left[\mu_r^2\right] < \epsilon|\hat{a}_u|^2$. We use this bound, combined with the Markov inequality, to show that our bit tests succeed with good probability, even in the presence of noise. We claim that the test $|a_r - a_{r+n/2}| < |a_r + a_{r+n/2}|$ is unchanged from the noiseless case whenever $2(|\mu_r| + |\mu_{r+n/2}|) < 2|\hat{a}_u|$. This follows from noting that, in the noiseless case, $|a_r - a_{r+n/2}| = 0$ and $|a_r + a_{r+n/2}| = 2|\hat{a}_u|$ when $b = 0$ (and vice versa when $b = 1$).

From the Markov inequality, $\mathbb{P}_r\left[|\mu_r| > \frac{|\hat{a}_u|}{2}\right] = \mathbb{P}_r\left[|\mu_r|^2 > \frac{|\hat{a}_u|^2}{4}\right] \leq 4\epsilon$. Setting $4\epsilon < 1/6$ ensures that each bit test succeeds with probability $2/3$. By repeating our entire algorithm multiple times, it is possible to boost this probability of correctness by taking b to be the most commonly chosen of 0 and 1. We need to union bound over $\log n$ bit tests, so we can use $O(\log\log n)$ independent tests to push the probability of failure below $O(1/\log n)$ (using the Chernoff bound). This will give a constant success probability after union bounding. Each test takes $O(\log n)$ time to compute every bit of u, so this gives an algorithm for the $k = 1$ noisy case with time complexity $O(\log n \log\log n)$.

In addition to correctly selecting u, to satisfy the ℓ_2/ℓ_2 guarantee we also need an accurate estimate of $\hat{a}_u$. Recalling that $a_j = \hat{a}_u\omega^{uj} + \mu_j$, just take $a_r\omega^{-ur}$ as an estimate for any randomly selected r. The difference between this value and the true value of $\hat{a}_u$ is $\mu_r\omega^{-uj}$. Now, $\|\mathbf{a} - \mathbf{a}^*\|_2^2 = \sum_{u' \neq u} |\hat{a}_{u'}|^2 = \mathbb{E}_r\left[\mu_r^2\right]$. So, by the Markov inequality,

$$\mathbb{P}\left[\mu_r > 2\|\mathbf{a} - \mathbf{a}^*\|_2\right] = \mathbb{P}\left[\mu_r^2 > 4\|\mathbf{a} - \mathbf{a}^*\|_2^2\right] < 1/4. \tag{17}$$

Thus, $|\hat{a}_u - a_r\omega^{-ur}| < 2\|\mathbf{a} - \mathbf{a}^*\|_2$ with probability $> 3/4$. In total, $\|\mathbf{a} - \mathbf{a}'\|_2 < \|\mathbf{a} - \mathbf{a}^*\|_2 + |\hat{a}_u - a_r\omega^{-ur}| < 3\|\mathbf{a} - \mathbf{a}^*\|_2$, giving the ℓ_2/ℓ_2 guarantee with constant probability.

References

[1] A. Gilbert, S. Guha, P. Indyk, M. Muthukrishnan, and M. Strauss. Near-optimal sparse Fourier representations via sampling. *STOC*, 2002.

[2] H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Near-optimal algorithm for sparse Fourier transform. *STOC*, 2012.

[3] E. Kushilevitz and Y. Mansour. Learning decision trees using the fourier spectrum. 1991.

[4] Y. Mansour. Randomized interpolation and approximation of sparse polynomials. *ICALP*, 1992.

[5] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. *STOC*, 1989.