Compressive Saliency Sensing: sampling near the edges

Scott Sievert
sieve121@umn.edu

Contents

---

1 One Dimension

In one dimension, we do not have a tree like we do in the two dimension case. Instead of having three branches like the 2D case, we have two branches at each node.

1.1 Approximating the Wavelet transform

Lets say we have an n dimensional signal: $x = (x_1, \dots, x_n)^T$. We can take the wavelet transform of this signal, w, using $x = hw$, where h is the wavelet matrix (in our case, the simple Haar matrix).

The terms in w correspond to different frequency components, and the latter terms correspond to higher frequencies. We only care about these terms if we're close to an edge, since that's where high frequency terms are. There's no need to approximate a DC or constant term with high frequency.

So let's say we're only interested in the top m terms of w, since what's specified when we approximate the wavelet. Since we only care about the upper portions of w, we can get rid of the corresponding rows for h.

$$\begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & h_{1,4} \\ h_{2,1} & h_{2,2} & h_{2,3} & h_{2,4} \\ h_{3,1} & h_{3,2} & h_{3,3} & h_{3,4} \\ h_{4,1} & h_{4,2} & h_{4,3} & h_{4,4} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

Since we only care about the first two entries of w ($m = 2$), we can write

$$\begin{bmatrix} h_{1,1} & h_{1,2} \\ h_{2,1} & h_{2,2} \\ h_{3,1} & h_{3,2} \\ h_{4,1} & h_{4,2} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

And, saying we didn't sample at index number 2, we can write

$$\begin{bmatrix} h_{1,1} & h_{1,2} \\ h_{3,1} & h_{3,2} \\ h_{4,1} & h_{4,2} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

Carrying out that matrix multiplication, or solving a linear system of equations, will approximate the wavelet coefficients.

1.2 Going between the wavelet and time indices

We know that $h^{-1}x = w$. Since we know that $h_{i,j} = 0$, we know that x_j won't matter for the ith element of w.

1.3 Scaling the wavelet

We know that samples in spaces where there aren't many samples are much more important. We therefore have to inflate the wavelet coeffecients to stay true to Peserval's Thm.

For the first wavelet coeff., $w(1) = wn/m$ where n the length of our signal, and m is where we sample at. If the signal is 16 components long and we only sample at 4 locations, it has to be inflated by a factor $16/4 = 4$.

But, that breaks down for the lower terms. The lower terms are of different signs. If we sample frequently in the first half of our signal, our term looks something like $[+ \ + \ + \ + \ + \ + \ - \ - \ - \ ]$ – more positives than negatives, pushing our coeff. more positive.

That means we have to scale each term in the Haar matrix. Since the Haar wavelet has positive and negatives of equal scale, we can look to see how many sampled locations lay within that region, inflating by n/m. n is how wide the region is, and m is (again) how many samples lie within that region.

2 Two Dimensions

2.1 Approximating the wavelet transform

2.2 The approximation algorithm

I went through a long, intensive process to find this. But of course, there's a simple mathematical fix to this problem. Let's say you have

$$AXB = C$$

Using the Kroneker product ($\otimes$), that's equivalent to

$$(B^T \otimes A)\text{vec}(X) = \text{vec}(C) = \text{vec}(AXB)$$

This is remarkably similar to the 1D case. You have to delete the rows corresponding to where the finer detail lies and the columns that correspond to where you don't sample.

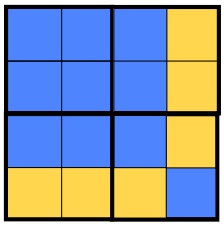

Figure 1: The signs associated with a wavelet coefficient. Blue is positive and orange is negative; both magnitudes are one.

In our case, we have $w = cxr$, where c performs the complete wavelet transform on the columns and r the complete transform on the rows.

2.2.1 Scaling

We are currently saying "we're interested in these coefficients, and don't care where they lie." That means that to get an accurate reconstruction, we have to have scale our function right.

But, we could just approximate our wavelet to some level. Then, when we don't care about terms, they're automatically set to 0 (via the dot product).

2.3 Which indices are important?

Initially, I thought that the wavelet transform had to be recursive. That makes it really tricky for the indices – you would have to make `dwt_ind` functions, and keep doing it.

But after I talked to Ashkay, I learned that you can do the full wavelet transform on each row then each wavelet transform on each column. That makes this function trivial: it's just a matter of indexing.

2.4 The reconstruction

In the wavelet domain, we have a tree that corresponds to the image. An example is in Figure 2. The upper levels of the tree represent the lower frequency terms. Since there's no need to closely sample a low frequency term, we only sample where there are high frequencies.

We know that if any of these "branches" are close enough to zero ($|x| < \lambda$) that all of it's child branches are close enough to 0 as well. Therefore, we only look at where the branches are not close enough to zero ($|x| > \lambda$). As we go further down the branch, we form a better approximation of the wavelet transform.

Where we choose to look (or where we sample) is near the edges. The wavelet transform is zero for a constant: it has no high frequency terms.

2.5 The tree

The idea is simple: where the coefficients in the wavelet tree are nonzero, go further down the tree. But, how do we decide when a coefficient is nonzero? What we currently do is simply see if there nonzero after the *initial* approximation, and this tree is never updated. That part is "easy."

Another thing that I find interesting and confusing is we know that the energy in the transform remains the same. In mathematical terms, it's $||x|| = ||w||$. This norm can either be the $||\cdot||_2^2$ or the Fourbineas (yeah, spelling) norm.

But we're not interested in x. We interested in sampling x, meaning Sx, where S is just a $m \times n$ matrix with one 1 per row. That means that the total energy is off by a factor of n^2/m (m being the locations we sample at, n being the width/height of our image). But, that only corresponds to each term being off by *exactly* n^2/m.

What we ideally want is to reconstruct the wavelet term *exactly* (correctly scaled) at each level. Using the tree case, that means that the percentage of the original value increases as you go down the tree. The lower terms are closer to the original values while the higher terms are reduced at a much lower rate. The top term samples at 2^n locations while some term k levels down samples at 2^k locations.

Does that mean we just have to normalize each wavelet coefficient by k/m (k where that term samples at, m how many samples you have there). I'll ask Akshay to be sure before I try (hello Akshay!).

References

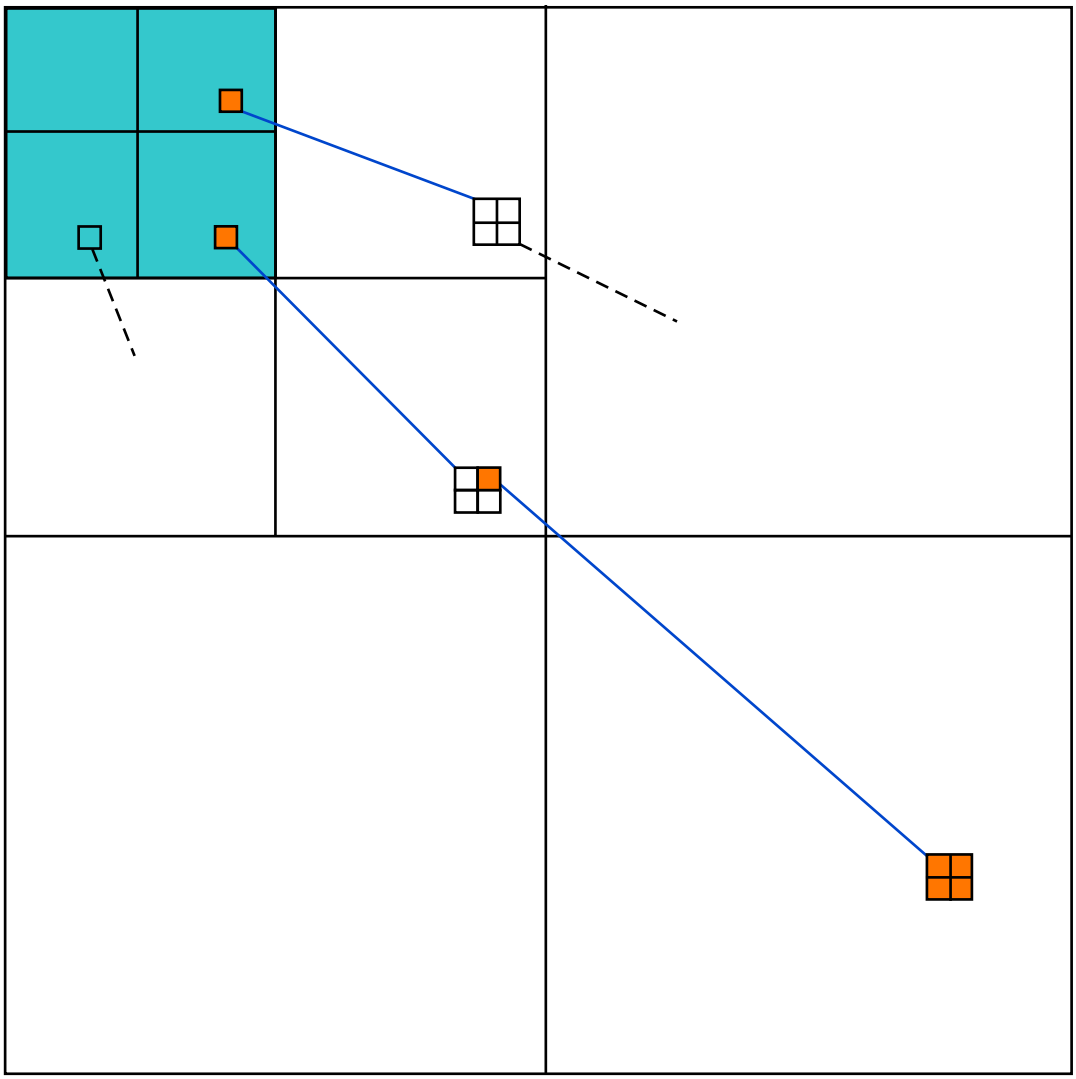

Figure 2: A visualization of our tree. The blue square is our current node. Only two of it's components are non-zero, meaning that there's more detail at lower levels we need to look at closer. We then build a better and better approximation as we need too.