Active Learning of Sparse Pseudo Boolean Functions with the Walsh Hadamard Transform

Tavor Baharav

12/13/16

1 Background and Motivation

The Walsh Hadamard Transform is a very effective tool for analyzing certain families of signals by representing them in an appropriate basis. Whereas the Fourier transform represents signals in a basis of complex exponentials (harmonics), the Walsh Hadamard Transform represents these signals in a basis of Walsh functions, which are a set of orthonormal vectors with values -1 or 1. The Walsh basis is a natural one for representing pseudo boolean functions, functions that take in n boolean inputs and return a real or complex valued output. This representation can be described as a polynomial with $N = 2^n$ terms, with the resultant function being defined as:

$$f(x) = \sum_{k \in \mathbb{F}_2^n} X[k] P_k(x) \qquad \forall x \in \{-1, 1\}^n$$

where $k := [k[1], ..., k[n]]^T \in \mathbb{F}_2^n$ is the index of the monomial $P_k(x) = \prod_{i=1}^n x_i^{k[i]}$, and $P[k] \in \mathbb{F}$ is the coefficient, where P is K sparse [3]

In the 3 variable case the functions simplifies to $f(x_1, x_2, x_3) = X_0 + X_1 x_1 + X_2 x_2 + X_3 x_1 x_2 + X_4 x_3 + X_5 x_1 x_3 + X_6 x_2 x_3 + X_7 x_1 x_2 x_3$

Many real world problems exhibit sparsity when represented using the Walsh basis, such as graph sketching, decision tree learning, and in pseudo boolean function learning [1]. One such pseudo boolean function example lies in genetics, where given N boolean genes we are trying to predict the output phenotype, eg hair color. It is assumed that very few of the genes' interactions matter. In this case, the system can be described as $f(x_1, x_2, ..., x_n)$, where x_i is a boolean gene that can be classified as either on or off: $x_i \in \{-1, +1\}$, and the real valued output is defined as above. Our motivation for applying active learning to problems like these is that sampling the pseudo boolean involves running a lengthy experiment, and is very costly in both time and resources. In these situations especially, we are willing to sacrifice computation time for a lower sample complexity, as additional computation is insignificant compared to the time needed to collect more samples.

In this work, we will focus on cases where sampling the function is very costly, and we would like to minimize the number of samples needed.

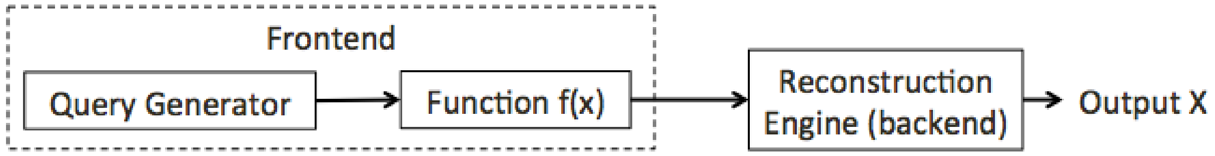
2 SPRIGHT algorithm

In X. Li and K. Ramchandran's paper they develop an algorithm called SPRIGHT ($\mathbf{SP}arse\ \mathbf{R}obust\ \mathbf{I}terative\ \mathbf{G}raph-based\ \mathbf{H}adamard\ \mathbf{T}ransform$) which computes the Walsh Hadamard Transform of a length N vector that has a K sparse Walsh Spectrum, K = O(N^δ) [1]. The advantage this algorithm offers is in its speed and sample complexity: while the classic Fast Walsh Hadamard Transform looks at all O(N) samples and operates in O(NlogN) time, the SPRIGHT algorithm operates in O(KlogN) time using O(K) samples in the noiseless case, and in O(Klog^2N) time using O(KlogN) samples in the noisy case. It does this by inducing a "friendly" aliasing pattern by subsampling in such a manner that we are able to generate a Low-Density Parity-Check code (LDPC code), between our aliased observations and the k sparse coefficients. This generates measurement bins, which we deterministically know which coefficients mapped into, based on the subsampling pattern. Because of the LDPC code we induced, we now have a sparse bipartite graph with K nonzero left nodes (variable nodes, the coefficients), and CB right nodes (check nodes, our bin measurements), where C is the number of subsampling stages we choose to use and B is the number of bins per group. After we complete the sampling stage, we move on to the reconstruction engine. Here, we have a peeling based decoder where we iterate over our check nodes, and one by one classify them as a zero-ton (no nonzero coefficients hash to that bin), single-ton (exactly one nonzero coefficient hashes to that bin), or a multi-ton (more than one nonzero coefficient hashes to that bin). If this bin is a zero-ton, it is irrelevant and we can ignore it, as it has no information to give us. If it is a multi-ton, we can't determine any specific coefficient from it, and so save it for later. If it is a single-ton, then we can use a ratio test based off of the logN + 1 delays we took to determine what frequency it corresponds to, and thus solve one of the left variable nodes (one of the k nonzero coefficients). Now that we have solved this variable node, we can subtract its contribution from all the right check nodes it hashes to, peeling that frequency off of the graph. The idea behind this peeling decoder is that we will have enough single-tons, and an appropriate density, such that we are always able to find a single-ton and peel at each stage in the process, eventually recovering all k nonzero coefficients.

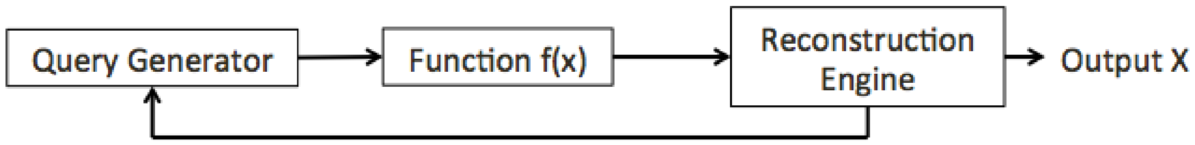
We recently finished writing a testing platform for the SPRIGHT algorithm, which can be found on the UCBASiCS GitHub (https://github.com/UCBASiCS).

3 Active adaptation

The SPRIGHT algorithm works by predetermining a set of queries, getting those measurements, then running the backend peeling decoder to solve for the sparse signal. This sequential nature can be seen below.



The idea behind active learning is to minimize the number of queries and computation time required, given that we can make sequential queries, that we are able to decide what our next query should be given the results of all previous queries. This is implemented by updating the reconstruction engine with new information as we query the function, have the reconstruction engine send its state back to the query generator, and the query generator determining what is the most informative next query to make. In this sense, we have a feedback loop as modeled below.



We propose an active learning framework in which we still utilize the SPRIGHT algorithms subsampling pattern and peeling decoder, but make use of our active querying ability to minimize the number of bins we need to measure. To do this, we take the measurements required for a bin in a group, but go sequentially through the bins, with the goal being to stop after as few bins as possible. We check if the bin is a single-ton after we take its corresponding measurements and perform a short Walsh Hadamard Transform on its elements. If it is, we iterate over all the bins this single-ton's frequency would map to: if we have not yet needed to measure this bin, we make a note that this single-ton has been found, and to subtract it from that bin once we measure it. If we have already measured this bin, then we subtract the spectrum of this frequency, the impact it would have on the current bin, and check to see if this updated bin is a single-ton. If so, we keep peeling. We peel until all exposed single-tons have been peeled, ensuring that if at any point we have taken enough bin measurements to be able to determine the signal, we don't take any more.

We can see that this method is a strict improvement on the current scheme: at worst it will take the same number of measurements, but in most cases it will take many fewer. This is especially true in the lower sparsity regimes, as in these scenarios, we have very few balls that we are throwing into our bins, leaving us with a significantly larger proportion of single-tons than we would otherwise have. Because of this, with high probability, we will not need to use all the bins we designed to determine all the nonzero frequencies, and thus stop needing additional measurements earlier. We can see this difference in experimental simulations, running on a signal of length 4096, with 2 chains of delays, using maximum likelihood detection, with a SNR of 15. By running 1000 iterations with a sparsity of k = 20 nonzero coefficients, we get a truncated normal distribution centered around 35 (see figure 1). While we appear to have made a significant improvement in terms of number of bins used, going from 48 down to an average of 35.7, we only reduce the number of measurements we need to take from 696 down to 641, less than a 10% improvement. This is due to the fact that by the time that we have reached the later bins, we have already taken most of the time domain measurements necessary to construct them, meaning that the percentage of bins we improve by is always going to be better than the percentage of samples we improve by. When we run the same simulation, but with k = 10 nonzero coefficients, we get a normal distribution centered around 27 (see figure 1). This can be seen as a significant improvement on the current standard, as we only use 26.3 bins on average out of the 48 total, and decrease the number of measurements from 696 to 526, meaning we use 25% fewer measurements in the process. One interesting similarity between these 2 graphs is that they both have a characteristic dip at 32 bins used. This means that the last bin in the second stage gives us very little new information that's immediately usable. We do not yet have an explanation for why this phenomenon occurs.

4 Future work

The majority of the work thus far on this project was spent getting the SPRIGHT codebase up and running. From this, many directions further exploration have been unearthed.

4.1 Classical Active Learning

We can try and model the probability distribution of X, the k sparse coefficients vector, and go about this problem from the more classical active learning perspective. This entails making the query at each step that will minimize the maximum entropy of the resulting possible probability distributions, based on the results of the query. With the addition of noise, this becomes more difficult, but still doable. A good toy example to show the motivation behind adaptively sampling is in the case of estimating a step function [4]. To be more concrete, we are given a function f defined on the interval [0,1] which is fully characterized by its step parameter θ, which is uniformly distributed on the interval [0,1], st $f_\theta(x) = \mathbf{1}\{x \in [\theta, 1], 0 \text{ else}\}$. Given a function f of this form with an unknown parameter θ, our goal is to estimate θ as precisely as possible from n queries to $f_\theta(x)$. We take the set of observations, $\{Y_i\}_{i=1}^n$, to be exact queries of the function $f_\theta(x)$, take at our sampling locations $\{X_i\}_{i=1}^n$, where $Y_i = f_\theta(X_i)$. We assume no noise for simplicity's sake, but the noisy case yields similar results, and can be seen in [4].

We start with non-adaptive sampling, meaning that our sample locations X_i are predetermined and are independent of the results of our other function queries, $\{Y_j\}_{j=1,j\neq i}^n$. In this case, the best we can do is effectively a grid search: have equally spaced X_i, such that we minimize the maximum distance between the step and the closest sample on either side. This means that $X_i = \frac{i}{n+1}$, and our backend computation to estimate the parameter θ from the $\{Y_i\}_{i=1}^n$ will simply entail guessing the midpoint between X_i = max i st. $Y_i = 0$ and X_j = min i st. $Y_i = 1$. These two measurements will be adjacent, eg j = i+1. Since we assumed a uniform prior on θ, our expected error will be $\frac{1}{2(n+1)}$, which we can see scales with $\frac{1}{n}$

In the case where we employ adaptive sampling, we are able to define a strategy for choosing our i^{th} observation, based on $\{X_i\}_{j=1}^{i-1}$ and $\{Y_i\}_{j=1}^{i-1}$. In this case, we logically want to perform binary search, as we know that with each measurement we take, we should be able to double our accuracy, as we can be making more refined queries than in our grid search non-adaptive previous method. We can see that this strategy is indeed optimal, as we end up with an error at most $.5 * 2^{-n}$. We can see that this is optimal, as each measurement is encoding a bits worth of accuracy, and thus with n bits, we can do at most a factor of 2^n better than we could previously. The reason that we're able to attain such a higher level of accuracy is that with each measurement we make, we divide the realm of possibilities in exactly half. More formally, this means that we make a query that divides probability distribution in 2 at its median each time, as this is the 1d case. With each query, we want to gain the most information possible, and thus we want to find a query that bisects our probability distribution. This technique of active learning can be used to help learn sparse boolean functions both within and beyond the SPRIGHT framework.

4.1.1 Within SPRIGHT

Within the SPRIGHT framework, we can use active learning to try and optimize the order in which we measure bins: eg, if we know that a bin is a zero-ton, then we know that all the frequencies that map to that bin have coefficient zero. With this information, we can choose to measure future bins that have these known zero coefficient frequencies map to them, in an effort to induce more single-tons in less sparse signals. Alternatively, in lower sparsity regimes, given this information, we can try and pick bins that have as few of these known zero frequencies as possible map to them, so that we gain information about unknown coefficients faster. This would be the simplest way to apply active learning techniques to the SPRIGHT framework. Another way of utilizing our previous queries would be to change our subsampling pattern after i queries, upon realizing that a different pattern would perform better, and that we want 8 bins per stage instead of 16, for example.

4.1.2 Outside of SPRIGHT

Outside of the SPRIGHT framework, we encounter a much more challenging and theoretically interesting problem than optimizing bin order. Applying adaptive sampling to learn sparse boolean functions is a multifaceted problem, that begins with modeling the probability distribution of the k sparse coefficient vector. Assuming we only draw coefficients from a finite alphabet, we are still dealing with a N dimensional probability distribution. The main difficulty in this, however, will be in determining how a (query, measurement) tuple will impact this probability distribution. With the SPRIGHT framework, we know deterministically what frequencies comprise a given bin, and thus we either know something is a single-ton in which case we know a new frequency, a zero-ton in which case we know all the frequencies mapping to it are 0, or a multi-ton in which case we know multiple frequencies mapping to it are non zero. With this active learning framework, the issue that arises is that each measurement depends on all the frequencies, not on just a select few as in SPRIGHT by construction. This means that with each query, we need to determine which of the possibilities within the probability distribution are eliminated, as we are dealing with a finite alphabet, and which become less likely. Furthermore, with this evolving probability distribution, we need to determine the median query, or be able to get a good approximation of it, which while in the 1 dimensional case was just a point, in this N dimensional case is a hyperplane. In the end, because this is a discrete distribution, we are interested in querying it until it degenerates to a delta function, which will be the K sparse coefficient vector.

4.2 Nonuniform sparsity

One of the issues with the SPRIGHT framework is that it has a uniform sparsity assumption, which does not hold in most cases. With boolean functions especially, coefficients are much more likely to be nonzero for lower order terms [2]. This is because in general, natural phenomena are not unnecessarily complicated. Thinking about this as a series of terms in Disjunctive Normal Form, we are applying the or operation (adding) a series of anded terms (multiplication with 1s and 0s). This non uniform sparsity assumption is saying that the terms with fewer variables in them are more likely to have nonzero coefficients than terms with many variables, or that lower order monomials are more likely. This assumption generally holds true for most real world phenomena.

Because SPRIGHT currently assumes uniform sparsity, it works well on signals with a randomly generated support, but doesn't perform as well when the sparsity starts clustering towards lower order terms. This means that our bipartite graph analysis for the LDPC will not work out as neatly, as we currently assume that each of the k balls is equally likely to be thrown into any bin. When all the balls are thrown uniformly at random, our aliasing pattern induces an appropriate distribution on the degree of the right hand bins. However, with a non-uniform sparsity assumption, the balls are no longer thrown at random, and it is easy to imagine that with our current scheme, there will end up being stages where most of the nonzero coefficients will end up hashing into the same bin, as they correspond to majoritively lower order terms.

While this generally harms SPRIGHT performance, we should be able to leverage this nonuniform prior to perform better than we currently do: being given a nonuniform prior gives us more information than a uniform one, and should thus help our algorithm perform better. Looking at this from the perspective of having a probability distribution for the k sparse coefficient vector, a uniform prior gives this distribution maximal initial entropy. Any additional information we know will reduce this entropy, and thus should make it easier for us to make queries to reduce the entropy to 0. The framework we use should be able to take in prior knowledge of the sparsity, or make common assumptions about its non-uniformity, to improve its performance.

5 Figures

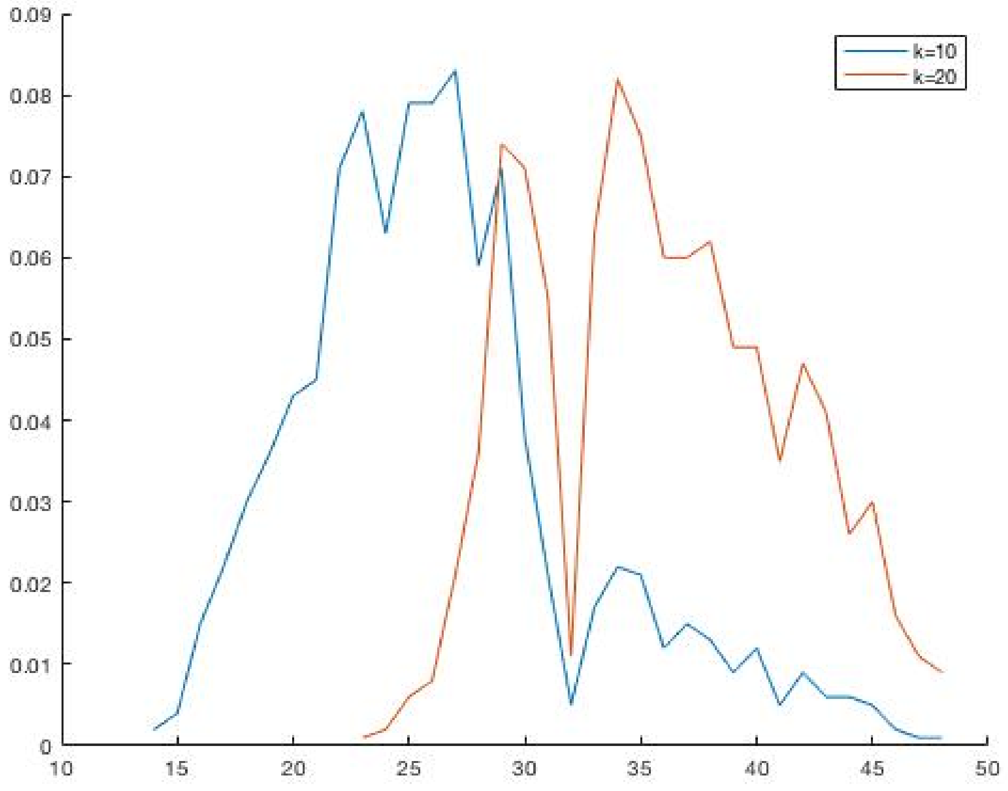


Figure 1: Experimental Results

References

[1] Li, Xiao, et al. “SPRIGHT: A Fast and Robust Framework for Sparse Walsh-Hadamard Transform.” arXiv preprint arXiv:1508.06336 (2015).

[2] Xue, Yexiang, et al. “Variable Elimination in Fourier Domain.” arXiv preprint arXiv:1508.04032 (2015).

[3] Li, Xiao, and Kannan Ramchandran. “An Active Learning Framework using Sparse-Graph Codes for Sparse Polynomials and Graph Sketching.”Advances in Neural Information Processing Systems. 2015.

[4] Castro, Rui, and Robert Nowak. ”Active sensing and learning.” Foundations and Applications of Sensor Management (2009): 177-200.