CS391L Machine Learning: HW2 ICA

Joel Iventosch UTEID: jwi245 Email: joeliven@gmail.com

February 21, 2015

1 Introduction

The goal of this assignment was to learn about Independent Component Analysis (ICA) by performing Blind Source Separation. Although the ICA algorithm can be applied to many different applications of blind source separation, in this particular assignment we focused on implementing ICA to extract the original (source) sound signals from a set of mixed sound signals. ICA can be effectively implemented in many application domains - but it is particularly useful in those domians that are time-series based.

ICA is a very powerful statistical techniqe in that it enables one to extract the hidden information in a set of mixed data. This is accomplished by assuming that the mixed signals are linear combinations of some set of underlying *independent* (or *source*) signals. ICA then utilizes an algorithm to minimize the mutual information contained in the reconstructed source signals, thus maximizing the entropy. For ICA to work, the underlying probability distributions of the source signals must be non-Guassian - because Guassian distributions are unable to capture the higer order statstical information that is needed to effectively perform blind source separation. Additionally, ICA makes the assumption that the source signals are independent, hence the data we recover from them in our reconstruction is independent and identically distributed. As I will discuss in the next section (Methodology), this is a critical assumption of ICA because it drastically reduces the computational complexity of the calculations involved (summarized from class lectures).

2 Methodology

ICA works by minimizing the mutual information contained in the mixed signals as a mechanism for deriving or reconstructing the source signals. Although there are several algorithms that can be used to minimize the mutual information in a data set, we used a gradient descent algorithm for this assignment. More specifically, we accomplished this by maximizing the entropy of the probability density function (pdf) of our underlying (unknown) source signals. Since we don't really know the density function(s) of our source signals, we start by making a reasonable assumption that their cumulative distribution function (cdf) is sigmoidal: $g(x) = 1/(1 + e^{-y})$. We then differentiate our cdf to obtain our density function: $g'(x) = 1/(1 + e^{-y}) * (1 - 1/(1 + e^{-y})) = g(1 - g)$.

Now, recalling the fact that entropy is equal to the negative expectation of the log of the density function, we obtain the useful result that maximizing entropy can be achieved by maximizing the log of the likelihood of our underlying density functions for our source signals (paraphrased from class notes and lectures). Since our underlying probability distributions (of our source signals) are assumed to be independent and identically distributed, we can find their joint probability by simply multiplying each individual probability together - which is critical (as metnioned above) because it implies that the log of their products is addative - which makes their partial differentiation much cheaper computationally. By taking our *unmixing* matrix, W, to be the parameter of our model, we arrive at our ultimate gradient descent (or ascent as the case may be) algorithm for blind source separation, by maximizing the log likelihood of the parameter W in our model, given the observed

set (X in our homework assignment) of mixed signal data.

3 Experiments

To test the accuracy of the ICA algorithm that the homework outlines and that we implemented for our assignment, I ran approximately 70 experiments. I use the term *approximately* because many of them were *partial* runs of the experiment in which I would cut the full experiment short (by simply killing the program/process) due to an obvious indication of poor results. Although terminating an experiment prematurely is never an ideal option in a perfect world, given the time constraints and computational costs of the reality we must operate in, it is sometimes the lesser of two evils - in other words, if you *know* (or are pretty confident you know) that the results of a given experimental run will turn out poorly, it seems like the best choice might be to terminate the run, chalk up the paramters that were used to poor selections, and start a new experimental run using different parameters in order to broaden the field of total visited (tested) parameter values. That said, one of the big drawbacks to this approach is that there could potentially be unexpected occurrences or trends that would have appeared further into a given experiment if I had let them all finish out to completion. The use of the term *completion* brings up another interesting point of discussion - namely, with an experiment like this, how do you know exactly when you've reached *completion*? I will discuss this further in the next section as well. Overall, I was less than satisfied with the outcome of my experiments, in spite of a few individual cases of good results. The average of the results, however, was subpar - at least based on the expectations that had been built up surrounding the potential efficacy the ICA algorithm.

I will now outline the strategy I implemented while performing my experiments. Typically, I first try to perform a short round of smaller exploratory experiments in which I manipulate the various parameters of the model in time-complexity cheap implemenations (i.e. shorter experiments) in order to get a sense for which parameters have the biggest impact and approximately what value range of those parameters needs to be further explored in a more organized, statistical fashion. This was essentially the approach that I utilized for the first homework assignment and it was very effective.

However, it was much more difficult to implement that strategy for this assignment, because the results of my initial exploratory micro-experimental runs were so inconclusive that they provided very little basis on which to create a well structured design of experiments (DOE). As such, my experiments were based more so on intuition and curiosity of testing out different parameter settings, than a well defined, statistically valid experimental design (this is admittedly a short-coming of my results for this assignment). That said, I did attempt to focus my experiments on a particular set of parameters, namely:

- learning rate, lr, of the gradient descent algorithm
- rate of variability for the learning rate (constant, linearly increasing/decreasing, or exponentially decreasing)
- value range for randomly initializing the mixing matrix, A
- value range for randomly initializing the unmixing matrix, W
- number of source signals used, n
- number of mixed signals generated to start with, m (determined by adjusting the number of rows in A)
- randomly generated values of a Beta parameter vector, b in $nx1$, used in the sigmoidal density function to provide some level of variability within the underlying probability distributions of the source signals: $1/(1 + e^{-b_i * y_{i,j}})$

LEARNING RATE: Among these various parameters, I spent the most time manipulating and trying to fine-tune the learning rate parameter, lr. My results varied drastically. Many of my better experiments utilized very small learning rates (i.e. < 0.01), however a few of my best results used larger

learning rates. All said, I ended up using learning rates ranging all the way from $0.001 - 0.9$. One trend that became apparent was that the larger value range I used to initialize A and W, the smaller the learning rate I had to use in order to avoid getting OverflowErrors in Python. The issue seemed to be that when the combination of A (and hence X, since $X = A * U$), W and lr was too big, the values of Y, our ongoing estimation of the source signal matrix U, became overly small (i.e. their absolute value became too big), triggering an OverflowErrorr in the next step of the algorithm, during the calculation of Z. This occurred because when the values of Y became overly negative, the program tried to compute $e^{-y_{i,j}}$ in the calculation of Z, for some very negative values of Y. As we all know, the value of e^y (in general) explodes very rapidly at as y grows. Thus, symetrically, the value of of e^{-y} blows up very quickly as y gets negative. And it doesn't take that negative of a y value to trigger the error.

While on the topic of learning rates, it is worthwhile to mention that it seems very counter-intuitive to me that using very small learning rates would produce better results - my intuition tells me that too small of a learning rate would inhibit the ability of W to climb the *gradient ascent* as needed, and would cause our algorithm to stagnate and possibly even get caught in a local minima that we might have been better able to avoid (or *jump* over) if utilizng a higher learning rate. For instance, in my past experience with neural network experiments, I typically used an annealed learning rate starting at $0.9 - .95$ and annealing down to the $0.05 - 0.1$ range. Whatever the case, based on the results of my experiments, I was not able to find a particular learning rate for this algorithm that I would consider optimal. I found very little correlation between the learning rate and the quality of my results (as measured by error correlation). Figure 1 and 2 plot error vs starting and ending learning rate, respectively. As you can see from the figures, there appears to be very little sensible correlation between the two.

LEARNING RATE ANNEALING: A second (related) parameter that I spent a fair amount of time building the code for and then manipulating in my experiments was the annealing of the learning rate parameter. I started my model with just a simple constant learning rate, but then shortly after decided to add in the option to pick a starting learning rate and an ending learning rate and coded in the logic to have the learning rate adjust linearly between the start and end value, with the rate of the annealing based on the number of max iterations allowed for the particular run (max iterations was also a parameter that I intended to tune initially - but so few of my experiments survived to max iteration count, either because they reached the indicated convergence threshhold first or I terminated them manually first - that max iterations ended up being a somewhat neglible parameter). I anticipated that this would have a hugely positive impact on the quality of the results, because it seemed like a great way to speed up the ascent in the ealry learning phases (and possibly avoid local minima), while allowing the algorithm to settle down and perform fine-tuned learning in the later stages of an experimental run. Although this seemed to work at times, my results were inconsisten with regard to linear annealing - thus it's hard to say whether that had a positive, negative, or no net impact on the outcome of the experiments overall. Although not statiscally of much significance given that it was not a result that I was able to consistenly reproduce, one in interesting outcome is that my overall best experimental results occurred when I used linearly annealing learning rate - in the postive direction! Most (if not all) of the time we think of an annealed learning rate, we think of a *decreasing* learning rate. However, in a few of my experiments I tried a linearly *increasing* learning rate - and one of them just so happened to be my best overall result. The others, however, were no better than average, statistically, so it is pretty difficult to make any solid inferences out of the one-time occurrence (after all, it could have just so happened to coincide with a very nicely seeded random initial W matrix, since W was a new randomly generated matrix when initialized with each new experiment).

After implementing the linearly annealing learning rate without much sucess - in terms of improved experimental results - I decided to implement the ability to select an exponentially decaying annealed learning rate. I coded my program to accept a base

parameter for the exponential decay and a power of the time paramter - namely c and k in $lr = c * (lr_{start} - lr_{end})^{t^k} + lr_{end}$. Although this equation worked well in terms of implementing an exponentially decaying learning rate with an asymptotically bounded lower limit - it did not seem to have any conclusive impact on the results of my experiment.

VALUE RANGE OF INITIAL W AND A: I also focused on manipulating the range of values with which A, our "God-like mixing matrix", was randomly generated with. Similarly, I manipulated the range of values with whicih W, our "unmixing matrix", was randomly initializd with. To start out I used the interval $(0, 1)$ to generate both A and W (using $1 * numpy.random.random(dims)$). However, it quickly became apparent that when using this large of an upper-bound limit for W, my algorithm triggered an OverflowError in Python nearly everytime. At first I thought this might be a bug, but after very careful review, extensive testing (using many print statements to debug), and even recoding the algorithm a few times, I was convinced that this was not a bug, but simply a consequence of having too large of a value range used to generate W. Accordingly, I then tried lowering the value range used to generate the initial W to $(0, 0.1)$ (using $0.1 * numpy.random.random(dims)$). This seemed to help significantly and reduced the frequency with which the OverflowErrors were occurring - although it did not entirely eliminate them.

Throughout the various experiments, I continued to alter the value ranges used to generat W and A, trying different combinations of $(0, 0.5)$, $(0, 0.1)$, $(0, 0.05)$, and $(0, 0.01)$. Although no single combination stood out as dominant in terms of yielding clearly superior results, a very interesting discovery did result from this - namely, manipulating the value range used to seed the matrices had a significant impact on the appropriate value that should be used for the starting learning rate in order to produce sensible results (and avoid OverflowErrors). The relationship appeared to be inverse and roughly proportional. The higher the seeding value range was, the lower the learning rate had to be - and the same vise versa. In hindsight this is a logical result, given the form of the algorithm used to optimize W - nevertheless, I was surprised by the rigidity with which it impacted the experiments.

NUMBER OF SOURCE AND MIXED SIGNALS: In an ideal scenario I would have first determined what was roughly the "best" learning rate to use by fine-tuning that parameter with all other parameters fixed - including the number of source and mixed signals used. Then, once an optimal learning rate (or several top contenders) had been established, I would have created a DOE to generate stastically valid results for the numerous values of the *number of source signals* and *number of mixed signals* parameters. However, given the difficulties surrounding producing consistently quality results, this was not possible. I was never able to determine an optimal learning rate (*optimal* in a statistically valid manner, at least - I could conjecture as to what "appeared" to be optimal, but my results were sporadic enough that statical optimality would be difficult to claim given the lack of reproducibility), hence my manipulation of the parameters controlling the number of source and mixed singals used was more of a random/exploratory-based search method than a well-thought-out DOE. Nevertheless, two conlusions became apparent regarding these parameters:

1. The more source signals used, the harder it became to achieve quality results (as measured by the correlation between the original source and reconstructed source signals).

2. Increasing the number of mixed signals dramatically increased the time-cost of running an experiment, yet had an inconclusive impact on the results.

The first conclusion is very intuitive and was expected - the more source signals you are dealing with, the more difficult it will be to extract those signals from the mixed signals in the reconstruction. This makes sense because of the algorithm design and assumptions that ICA is based on - namely, that the source signals are entirely independent and thus any correlation between mixed signals is due to the notion that they are linear combinations of the independent source signals. However, this rests on the *assump-*

tion that the original source signals are, in fact, completely *independent*. In reality, as we add more and more original source signals, the odds of them being *completely independent* decreases significantly - hence the increased difficulty in accurate blind source separation as the number of original sources increases.

The second conclusion was somewhat surprising. I would have anticipated that increasing the number of mixed signals would have significantly improved the accuracy of the blind source separation, given that it would have provided much more correlated data for the algorithm to use when extracting the hidden factors (i.e. source signals) from the mixed signals. Although at times this did appear to be the case, the results were not conclusive in this regard.

BETA: Lastly, I implemented a randomly generated beta parameter, b, to provide some level of variability within the underlying probability distributions of the source signals: $1/(1 + e^{-b_i * y_{i,j}})$. One of the fundamental issues that I see with the ICA algorithm, is that by simply picking a sigmoid as the density function for *all* of our source signals, it assumes that they all have exactly the same underlying probability distributions - which seems pretty far fetched in reality. Although introducing the beta parameter does not fundamentally change the functional form of the density function, it does provide some variability by "squeezing" or "stretching" the sigmoid for that particular source singal (in my experiments beta was an $nx1$ column vector so that each source signal would have its own beta value in its sigmoidal density function). While I conceptually like the idea of using this beta paramter, my implementation had two primary drawbacks:

1. I implemented the beta paramter later on in the project, thus many of the experiments did not utilize beta.

2. I used randomly generated initial values in the interval $(0, 1)$ for beta - but I was unable (due to project scope and time constraints) to implement an "outer loop" in my algorithm in order to optimize beta.

Although the beta parameter would have been much more effective if properly optimized, I still think it was worthwhile since it helps offset the unlikely assumption that all of the source signals have the exact same underlying probability distributions. That said, without optimization, it's impact is hard to quantify.

4 Results and Discussion

In this section I will summarize the results of my various experiments, and suggest some possible explanations for these results. All plots, figures, and tables appear at the end of the report.

BEST RESULTS: Of the 70 experiments I ran, 40 of them I ran through to completion. Of these 40, the top 10% produced fairly good results with respective errors of 0.25%, 1.15%, 2.23%, and 2.62% (see figure 10 for further details...top 10% of results are highlighted in green). A few interesting things to note about my top results: they all used only 3 source signals, they used a varying number of mixed signals, and each used a different starting learning rate - ranging all the way from 0.001 to 0.5. Additionally, it is worth noting that they each ran for 2500 iterations or less. These top results were satisfactory and resulted in very clear source separation, both from a visual standpoint when comparing the plots of the original source signals to the reconstructed signals, and from an auditory standpoint when listening to the actual signals (see figures 4, 5, and 6 for visuals of original source signals plotted next to reconstructed source signals).

AVERAGE RESULTS: In spite of achieving satisfactory top results, the avarage of my 40 experiments was significantly less than satisfactory. The overall average error correlation for all 40 experiments was 16.49%, utilizing an average starting learning rate of 0.113, an average ending learning rate of 0.075, and an average of approximately 7500 iterations. I was aniticipating much better average results than what I achieved. I think this was caused by several factors:

- W and A were both reinitialized with random values at the start of every experiment; this shouldn't impact the outcome, but if I'd used a constant initial W throughout (which I did breifly at the end, but it was too little, too late),

it might have helped in determing which learning rates are the most effective *relatively* - this optimized learning rate then could have been used with a variably initialized W in later experiments.

- Getting caught in local maxima - I think this was probably the biggest overall factor, because the algorithm seemed to very randomly get caught at different levels of correlation that it just didn't seem to be able to get past...and there was very little consistency in terms of where these local maximas occurred. Ultimatley I think an a combination of an optimized initial random W and a very carefully fine-tuned learning rate are the key to avoiding some of the local maxima, but much of the time I just wasn't able to accomplish that.

- Algorithm design - I think that since we were using a variant of gradient descent (since we post-multiplied by W^TW to avoid having to compute the inverse) it might have been subject to more variability and less accuracy at times, than if we'd been using the original algorithm. That said, it would obviously come at a cost of computational complexity.

- More iterations - I think my average results would have improved modestly by simply allowing each experiment to run for 2-3 times as many iterations.

The portion of figure 10 highlighted in yellow shows more detailed information for the median 10% of my experimental results. Interesting to note, they too each used a different learning rate and ran for a varying number of iterations.

BOTTOM 10% OF RESULTS: The portion of figure 10 highlighted in red shows more detailed information for bottom 10% of my experimental results. Interesting highlights (or lowlights as the case may be), include: they all used 4 or 5 original source signals and at least 6 mixed signals; they each used a very low learning rate: $0.001, 0.001, 0.001$, and 0.01 respectively for the bottom four runs. Additionally, one them even ran for 100000 iterations - far more than most of the other experiments - and yet still attained subpar results of 28.73% error.

WHICH SIGNALS SEPARATED THE BEST: I tracked my error rate by averaging the correlation of each original source signal with its best matching reconstructed source signal (and then subtracting that average from 1). As such, I didn't explicity track the error rates for each signal individually as I experimented - although this data was obviously calculated as part of the total error measurement. Part way through running my experiments I started tracking the individual error rates for each signal as well, however this was after the majority of the experiments had already been run. Accordingly, my correlation data on individual source signals is incomplete. That said, based on this partial data and visual and auditory inspection, I do have a fairly good sense of which original source signals separated the best. Specifically, I believe the 3rd source signal (if using 1-based indexing) - the clapping crowd - separated the best, followed closely by the 4th source signal (the high pitched laughing) and then the 1st source signal (the TV show sound clip). The 2nd and 5th source signals (the vaacum and static noise, respectively) were by far the most difficult to separate. I utilized the 2nd source signal extensively, but only included the 5th signal in a limited number of runs - so I don't think I can accurately classify which was more difficult to separate between the two. However, based on the data I collected and visual and auditory inspection, they were clearly the two hardest among the five signals. Intuitively this also makes sense since they are by far the most "subtle" and generic signals - thus they likely have a higher inherent correlation with the other source signals. Similarly, it makes sense that the 4th source signal (high pitch laughing) was easy to separate. I was somewhat surprised by how well the clapping/applause signal separated - and also surprised that the 1st signal (TV show clip) did *not* separate better.

HOW TO MEASURE CONVERGENCE: Throughout my experiments I used the Frobenius norm of the *deltaW* matrix to determine the level of convergence of the algorithm. It was suggested that this might not be a good measurement of convergence since the Frobenius Norm (Euclidean Norm

for matrices) could increase temporarily at different points before continuing to decrease, due to the algorithm adjusting its learning as it moves past localities. While this might be a valid claim, I would counter the point with the suggestion that it is the best viable measurement to use that is available to us. While it would certainly be more effective to use the change in the average signal correlation from iteration to iteration as our benchmark for convergence, this seems "unfair" given that in a real-world application of blind source separation we likely would not have knowledge of the exact structure of our original source data (after all, if we did, why would we need to perform blind source separation in the first place?). Whatever the case, the exact parameter to use for determining convergence is an interesting point fo discussion and for future exploration.

FUTURE WORK: The results of my experiments for this assignment leave much to be desired in terms of accurcay. Accordingly, the potential for future work is expansive. Among the many additional techniqes that could be experimented on, here are a few that are top of mind for me:

- Additional tuning of the learning rate parameter
- Adding an outer loop to optimze the beta parameter - and possibly designing an alternating loop structure (as opposed to a strictly inner and outer loop) that optimizes W to an extent while holding beta constant, and then switches and optimizes beta, while holding W constant - and continues to alternate in this fashion until some convergence criteria or max iterations is reached
- Using neuroevolution to find the optimal learning rate parameter
- Using the gradient ascent algorithm that includes the calculation of W^{-1}
- Using an alternative to the sigmoid function in our algorithm

Overall, this was a challenging and enjoyable assignement and I learned a lot about ICA, in spite of subpar experimental results. My figures, plots, and tables are below:

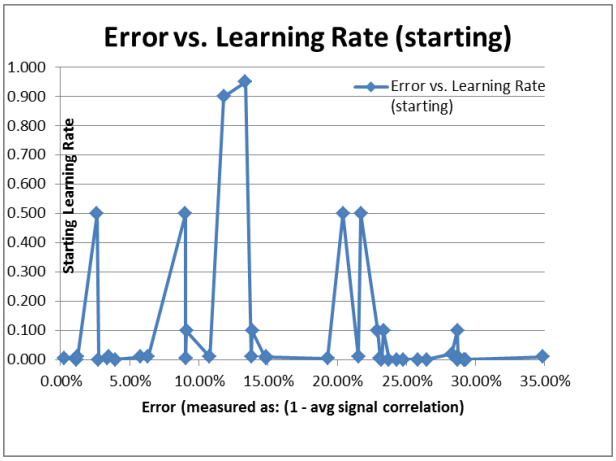

Figure 1: There is no discernable correlation between Error and Starting Learning Rate

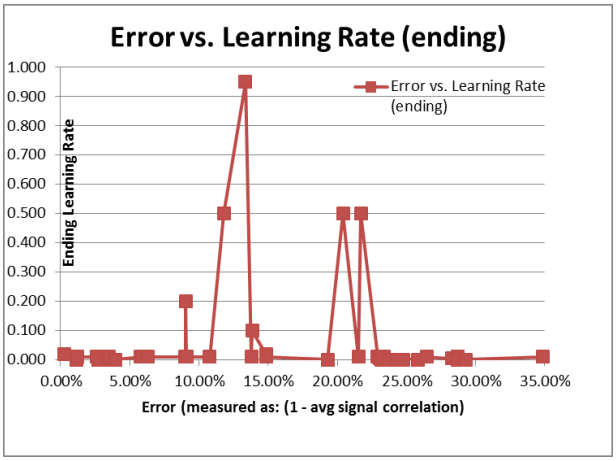

Figure 2: There is no discernable correlation between Error and Ending Learning Rate

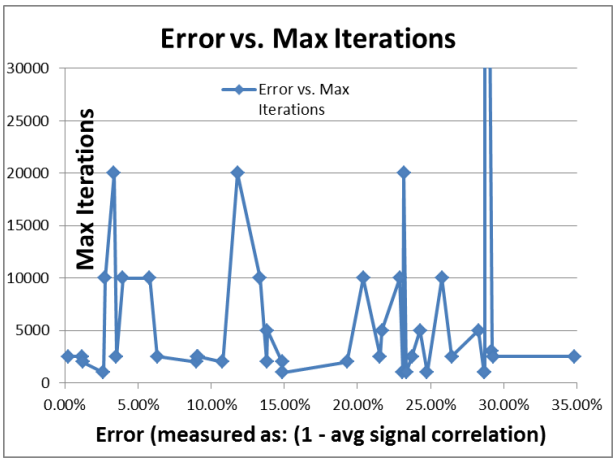

Figure 3: There is no discernable correlation between Error and Max Iterations

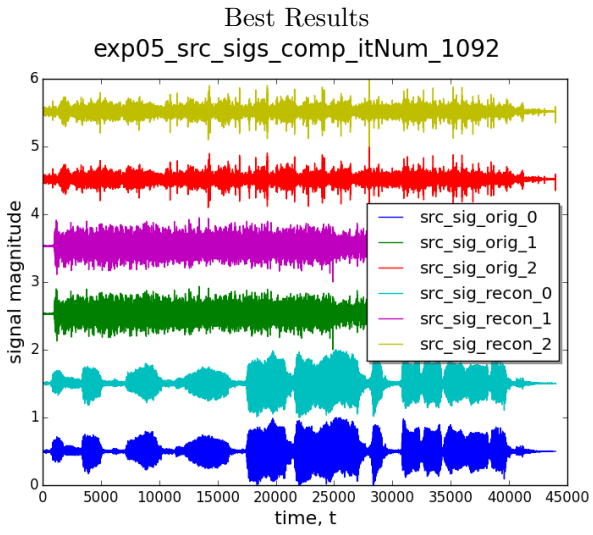

Figure 4: Avg Signal Correlation = 99.75 pct

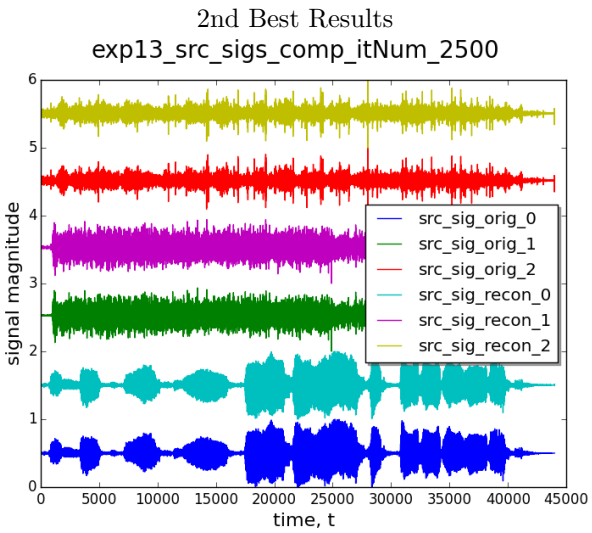

Figure 5: Avg Signal Correlation = 98.85 pct

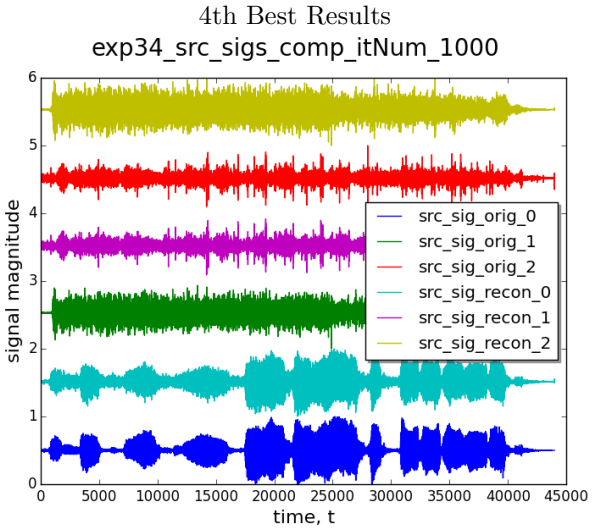

Figure 6: Avg Signal Correlation = 97.38 pct

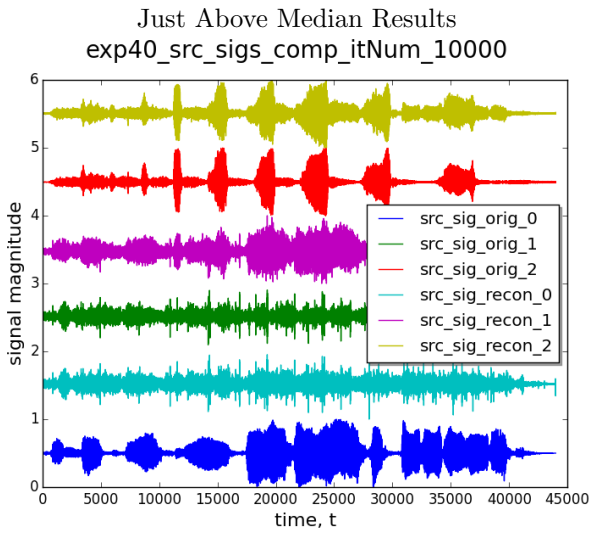

Figure 7: Avg Signal Correlation = 86.66 pct

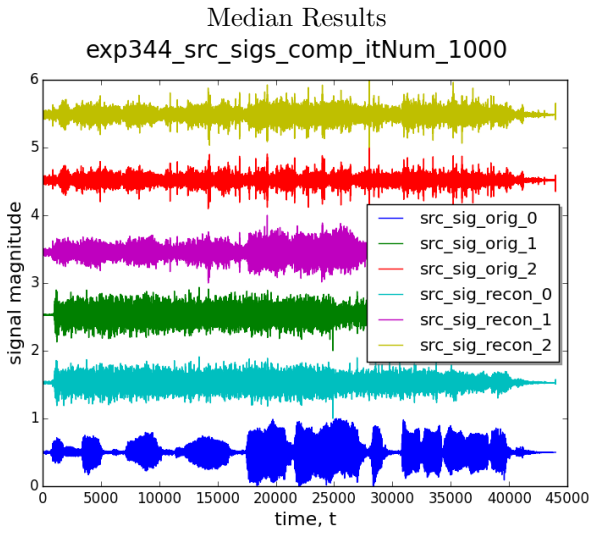

Figure 8: Avg Signal Correlation = 85.11 pct

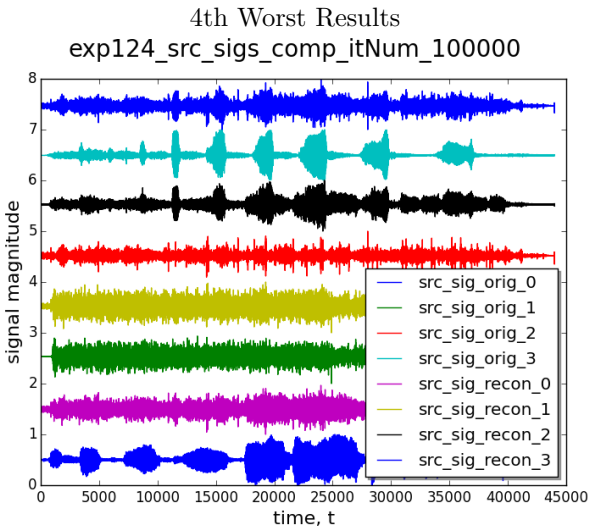

Figure 9: Avg Signal Correlation = 71.27 pct

Error	src_sigs	n	m	lr_str	lr_end	anneal	max_itr
0.25%	[0 1 2]	3	6	0.005	0.020	linear	2500
1.15%	[0 1 2]	3	5	0.001	0.001	none	2500
1.23%	[0 1 2]	3	7	0.010	0.010	none	2000
2.62%	[0 1 2]	3	7	0.500	0.010	decay	1000
2.75%	[0 1 2]	3	5	0.001	0.001	none	10000
3.35%	[0 1 2]	3	4	0.005	0.001	linear	20000
3.51%	[0 1 2]	3	6	0.010	0.010	none	2500
3.95%	[0 1 2]	3	3	0.001	0.001	none	10000
5.80%	[0 1 2]	3	6	0.010	0.010	none	10000
6.33%	[0 1 2]	3	8	0.010	0.010	none	2500
9.03%	[0 1 2]	3	7	0.500	0.010	decay	2000
9.05%	[0 1 2]	3	8	0.005	0.200	linear	2500
9.10%	[0 1 2]	3	5	0.100	0.010	linear	2500
10.77%	[0 1 2]	3	5	0.010	0.010	none	2000
11.81%	[0 2 4]	3	5	0.900	0.500	decay	20000
13.34%	[0 2 3]	3	5	0.950	0.950	none	10000
13.80%	[0 1 2]	3	5	0.010	0.010	none	2000
13.84%	[0 2 3]	3	4	0.100	0.100	none	5000
14.86%	[0 1 2]	3	6	0.005	0.020	linear	2000
14.89%	[0 1 2]	3	5	0.010	0.010	none	1000
19.31%	[0 1 2]	3	4	0.005	0.001	linear	2000
20.42%	[0 3 4]	3	7	0.500	0.500	none	10000
21.54%	[0 1 2]	3	3	0.010	0.010	none	2500
21.70%	[0 2 3]	3	4	0.500	0.500	none	5000
22.91%	[0 1 2 3]	4	6	0.100	0.010	linear	10000
23.10%	[0 1 2]	3	5	0.005	0.005	none	1000
23.19%	[0 1 2]	3	5	0.001	0.001	none	20000
23.34%	[0 1 2]	3	5	0.100	0.010	linear	1000
23.73%	[0 1 2]	3	3	0.001	0.001	none	2500
24.28%	[0 1 2]	3	5	0.001	0.001	none	5000
24.76%	[0 1 2]	3	6	0.001	0.001	none	1000
25.80%	[0 1 2 3 4]	5	7	0.001	0.001	none	10000
26.47%	[0 1 2 3]	4	8	0.001	0.010	linear	2500
28.27%	[0 1 2]	3	5	0.020	0.005	linear	5000
28.67%	[0 1 2 3 4]	5	15	0.100	0.010	linear	1000
28.71%	[0 1 2]	3	6	0.010	0.010	none	1000
28.73%	[0 1 2 3]	4	6	0.001	0.001	none	100000
29.19%	[0 1 2 3]	4	6	0.001	0.001	none	3000
29.30%	[0 1 2 3 4]	5	9	0.001	0.001	none	2500
34.84%	[0 1 2 3 4]	5	7	0.010	0.010	none	2500
16.49%		3.3	5.9	0.113	0.075		7487.50

Summary of Results

Figure 10: Results of 40 experiments that ran through to completion.