Blind Signal Decomposition

A study in the separation of unknown signals

(AKA adhoc experimentation in a blind alley)

Alec Rogers
Learning From Data, EE 410

Introduction: Previous Work

The Cocktail Party Effect is a common example of being able to track the source of interest, and not irrelevant information. This body of literature is significant in that it shares a problem with the current research; the background (i.e. what is not desired as a part of the output signal) is often not suitably modeled by a white (gaussian) noise process, as it is composed of other signals which are much like the signal of interest in terms of spectral characteristics. In this paper, I will be limiting the background noise to a single source. Although it would be more difficult to pick out a signal from a more complicated background (as would occur at a cocktail party), it remains quite a difficult task without making assumtions about the signals. The premise in this paper is also more difficult from the cocktail party effect as it is usually set up, as there are not mutiple, spatially seperated, sound receivers.

Another related topic is called Blind Signal Separation, or BSS. In this field, there are multiple sources that pass through a linear mixer, which yields multiple outputs (each of which contains a different amount of each input signal). I read this literature avidly, as much of it seems to pertain to what should be done with a single source. The most popular method in this field seems to be ICA, or Independent Components Analysis (which is a non-linear extension or Principle Components Analysis). The basic idea of ICA is to maximize the entropy of a signal when it is de-mixed. Another way of putting this is that the outputs are uncorrelated with each other after they are seperated, so that they each contain as little redundant information (i.e. information about each other) as possible. Unfortunately, there is not an easy migration from the methodology which assumes multiple inputs and the model I am using, which presumes a single input.

Much of the literature on signal or sound processing utilizes signal blocking, after which the signal can be treated as a series of points in a higher dimensional space. I decided not to pursue this avenue as it violates the property of time invariance. Although this method has shown itself useful for such tasks as compression, a windowed signal will often exibit undesired behavior when subsequent manipulations are performed on this signal, e.g. it will exibit characteristics related to the window insted of the underlying signal. As analysis of the blocked data will represent a given signal in a myriad of number of ways, depending on the way it was blocked, this can be viewed as a form of aliasing. I have decided to stay away from this methodology for these reasons.

Many of the speech-recognition approaches utilize time-frequency methods (e.g. Short Time Fourier Transforms), and I will use a modified form of this approach. Specifically, I will use a series of recursive lowpass filters which divide the signal into ocatve-based frequency bands. In order to preserve time-invariance, a sample at each frequency for each sample will be preserved. I will not say too much about this process of transforming input into this time-frequency domain, the interested reader is referred to [Rogers 99].

Introduction: Problem Formulation

The goal of Blind Signal Decomposition is to decompose a source signal in to its "natural" constituents. It is an unsupervised learning task, in that the correct outputs of the model (the separated sources) are not provided.

This methodology assumes several things about the signal: first, of course, that the signal actually has constituent components (into which the signal may be decomposed once again). Another assumption about the input signal for this experiment is that it is composed of repeating *events*. In other words, there are occurrances of events in the input signal, so that the input signal is not an always occuring phenomenon.

. Further, the signals should not always be present at the same time: there need to be examples in the combined source signal during which only one signal is present.

Throughout this paper, u represents the input to the model, x represents the state vector, y represents the model output, t stands for time, and z is reserved for the delay operator.

The task of Blind Signal Decomposition finds it application (or at least one of them) in the encoding of polyphonic music. In this context, it is assumed that 'sound events' will repeat, will have similar though not identical timbral characteristics, and that they will *sometimes* occur at the same time (thus polyphonic instead of monophonic transcription).

The basic process used in this experiment was as follows:

- First, an event was created (e.g. sine wave which lasted for one second). This sound was less real than it could have been, in that no large-scale time varying aspects were allowd (e.g. no Attack-Decay-Sustain-Release envelope).
- This event was repeatedly mixed into a longer input signal (u1), with intervening periods of silence.
- A second sound event was created, with a different timbre from the first (e.g. a square or sine wave of a different fundamental frequency).
- This event was mixed into a longer signal also (u2), typically in a rhythmic relationship with the first event (eg. it was mixed in only at every other onset, but it lasted twice as long as the first).
- The two constituent signals (u1 + u2) were added together to produce the unified source (u), which would eventually be givent to the model to decompose.
- The input signal (u) thus generated was scaled to the range of -1:1.

The model input signal generated in this manner bears a close resemblance to music (although it did not sound particularly nice). By generating the signal in this way, we produced a signal with the following characteristics:

- There were durations of time when either one source or the other was present (not both).
- There were durations of time during which both signals were present.
- There were also segments during which neither sound was present, although these sections were added primarily to observe what the model would do duing these times (there was no theoretical underpinning for including periods of silence).
- The spectral (fourier) characteristics of the sources were different.

To begin, let us formulate our two source signals, and their combined sum (the model input). Note that the scaling operation is not represented:

1. $u = u1 + u2$

Model: Formulation

Dual Prediction

The most basic formulation of the model was an extention of the standard temporal predictive model to cover two predictions whose sum would come to represent the input signal. In other words, the first condition that our model should obey is the following:

2. $u(t+1) = y1(t) + y2(t)$

This easily translates to an error function for both output values, although their errors are linked to each other:

3. $err(y1, t) = u(t+1) - y2(t)$
4. $err(y2, t) = u(t+1) - y1(t)$

Additionally, we would like not only for the sum of the predictions to be equivalent to the sum of the actual sources, but for each prediction to be identical to one or the other of the sources. Note that it is not important (nor possible) for a given predictor to know "which" of the two source signals that it is supposed to come to represent.

5. $y1 = u1 \quad || \quad y1 = u2$

These basic formulae give us a basic indication of where to start with the error criteria for our predictors. Fomula (2) gives a basic criteria for our models, and (3) will give us an indication of how well our model had parsed the combined input signal. Clearly, as our model is not permitted access to the seperate source signals, (3) cannot serve as an error for our model. Thus begins the project: how to develop a model that will decompose a combined source signal into its constituents without providing those constituents, or detailed information about them.

The prediction of a signal causes the predictor to come to model the signal. It was anticipated that by allowing two predictors to model components of the signal, each predictor would "specialize" in its predictions. I figured that two predictors would predict a signal well if they modelled logically independant aspects of that signal (i.e. they were independant of each other).

Model: The Predictor

In the course of the project I tried many (read: too many) approaches to the basic task. In this section I outline some of the major predictor designs, and the reasons for adopting them.

Nonlinear vs linear models

This project began with nonlinear models: multilayer perceptrons. They used a single hidden layer, most often with a linear output layer. A logistic sigmoid activation function was used in all cases. Various numbers of hidden layer neurons were used, although I most often used eight. There are not hard and fast rules about the design of MLP's, and I think that I did significantly more "parameter twiddling" than I should have.

At first, batch mode training was used, with inclusion of a momentum term. The batch consisted of evaluation of multiple error criteria, after which a single weight adjustment was made. This adjustment contained a fractional part of the last weight adjustment; this was the momentum term.

The use of multiple error surfaces is a dubious practice, in general. If there were a single error surface, the use of batch mode training (as compared with sequential (single-measurement single-update) training) does not guarantee convergence. In order to guarantee as "much convergence as possible" (or to satisfy as many covergence criteria), I eventually stopped using batch mode update, and resorted to updates at each evaluation of the error surface. This also implied that I give give up the use of the momentum term, which I did. These changes did not affect the model convergence to any great degree, as (I'm sorry to say) the model did not usually converge anywhere.

After toying a great deal with various MLP parameters such as these (and even implementing a different learning rule (the covariance hypothesis [Sejnowki 77])), I moved to linear models. In doing this, I assumed that they would be less capable overall in their ability to predict a signal, but much faster. A I was making various tweaks to the model, this was of extreme importance. If I achieved some degree of success, I decided, it would be relatively easy to switch back to a corresponding non-linear model.

Unified vs independent predictors.

Two kinds of predictors were used for signal prediction: a single predictor with two outputs, or two predictors, each with a single output. Single predictors have the advantage of ‘direct knowledge’ of what the other model is doing. In other words, non-linear functions about the inputs formed in the hidden layer would be available to both models. This was assumed to have the effect of decreasing the computation of two seperate prdictors, in that for a predictor to predict only part of an output still requires that it know something about the other part of the output. In general, I used the dual-output model most often. There was some concern on my part, however, that training one of the predictors would have a decremental effect on the performance of the other predictor. In other words, independent predictors have the advantage that training one model will not affect the predictions of the other model. The combined model, on the other hand, will alter the strengths of the hidden layer neurons based on error from both predictors.

Serperate predictors also allow for other types of manipulations in an easier fashion. One example of this is "winner take all" training. In a winner-take-all training paradigm, you are still training the loser to some degree if both of the outputs rely on the same neurons. This fact makes me uneasy; I am not aware of all of the implications of a shared hidden layer for a two-predictor model.

Filter Inputs

Time Domain Analysis

I used a Tapped Delay Line (TDL) for all of the cases of this experiment. Usually I used 16 taps in each direction (i.e. causal and anticausal), and made sure that the sampling rate was sufficiently low that this range would cover a period of the sound sample. This ensured that my filter taps would sufficiently characterize the lowest frequencies of the input signal. This is a fairly standard prediction setup, where the coefficients relating each delayed sample to the prediction of the desired sample are the autoregressive parameters of the input model.

Wavelet Domain analysis

As an alternative to time domain taps, time-frequency (or wavelet) domain domain taps may be used. In models where I performed this kind of analysis, all of the signals (the source and the estimation)

were converted into the wavelet domain. Before visual analysis of the results, everything was transformed back into the time domain.

An advantage of this kind of analysis is that (theoretically) fewer taps may be used. The reason for this is as follows: low frequency predictions (which require taps extending further into the future/past) may be achieved by looking only at low frequency information from the future and past. Because this low-frequency information requires a lower sampling rate to capture its content, high-frequncy (i.e. time-domain) taps are not necessary far into the future/past.

These wavelets (really filters, as I have used them) are also known as convolution kernels.

For more details on the wavelet methodology used in this paper, consult [Rogers 99].

Recursive (stateful) models

In order to augment the information provided by the taps, state information was utilized. This was provided by feeding back the output of the model to a 4 tap TDL which served as additional input. This transfomed the basic MLP design into a NARX (non-linear autoregressive with exogenous inputs) design. This design was used becuase it is fairly simple, and it has been proven to be computationally equivalent to a Turing machine [Haykin 99].

After the move to linear models (from an MLP based methodology), I began exploring state infomation more closely (this is something I have had little exposure to). The basic formulation of an LTI system is as follows:

$x1 = A1x1 + B1u$

$x2 = A2x2 + B2u$

$y1 = C1x1$

$y2 = C2x2$

I moved to this alternative representation as an intermediary step in trying to implement a Dual Extended Kalman Filter. There has been evidence to suggest that this kind of filter is capable of basic signal seperation tasks in the only paper on monaural signal seperation that I could find [Wan]. Unfortunately, I was not able to understand the DEKF well enough to implement it.

Results: Expectation

The first problem with using (2) as an error function is that the two signals are free to diverge from the mean, as long as their sum is equal to the input signal. The question of how to correct this behavior involved the expectation of the value: we do not want to add an error criteria which forces all responces toward zero, we only want to induce the mean (expected) value of the response to be zero.

Using expectation within an error function provides several difficulties. One of them is that the error is no longer due to the immediate error surface; it is due to both the present evaluation of the error surface and the evaluation of past error surfaces. To some degree, this would seem to necessitiate that we keep the weights of previous training trials, so that we can adjust them with respect to their contribution to the error surface at that time. This is a huge burden, and to overcome this obstacle I made a simplifying assumption: that the rate of change of the weights would change much more slowly than the rate of change of the exprected value of the result. With this assumtion in mind, the back propogation of the error can rely on only the surrent weight values. This assumtion worked in practice, but I do not have a songly argued mathematical foundation to support this heuristic.

Another problem with error functions that involve expectation is that the computation of the average value is a difficult construct in itself for causal time series. To accomodate this computation, I used a local average, which gave more weight to recent values as opposed to values far in the past. The specific calculation of the average invloved a IIR filter, which may be summaraized as follows:

6. E(y(t)) = Gamma * y(t) + (1-Gamma) * E(y(t-1))

I will refer to this sort of memory structure as a Gamma memory henceforth in this paper, although it differs somewhat from the standard definition of a gamma memory [Haykin 99]. This formulation depends on a coefficient, Gamma, which determines the depth of the memory. If we set gamma to 1, it is clear that we no longer have an expectation; we have only the most recent result, and there is no memory whatsoever. If we set Gamma to a small (non-zero) value, we are extending the memory further into the past. It is also computationally a very simple calculation, as opposed to a sliding rectangular window (in which case we would have to preserve previous values independantly up to the depth of the memory).

Interestingly, if gamma is a harmonically decreasing series (starting with 1.0), we have *exactly* the mean (since the inception of the filter). This implies that the filter (with such a Gamma parameter) is mean ergodic in the MSE sense. Although this is an attractive property to have, I chose to use a small value for gamma to approximate this summation.

As a result of using this Gamma function (with Gamma=0.0001) to calculate the expectation of a prediction, the means of the predictions were able to be reliably constrained to zero mean. Although this formulation primarily reduced means that diverged from zero over the long run, it did seem to contribute to reducing the overall power of the output to a small degree.

7. err(y1, t) += E(y(t))

Results: Power

After implementing the model as described so far, one of the predictors invariable came to model the signal. The other predictor flatlined; it predicted nothing. In order to balance the two filters, a penalty for relative power was added. In this term, if the power of one filter grew much higher relative to the other filter, it's error term was reduced. The error of the other filter increased; specifically, formula (2) was changed so that the contribution of a filter to the total prediction was based on the expectation of its power (i.e. the mean of its squared amplitude).

8. $u(t+1) = (1-p) * y1(t) + p * y2(t)$

Where p is a ratio of the power of one predictor relative to the other:

9. $p1 = E(y1^{**}2)$
10. $p2 = E(y2^{**}2)$
11. $p = p1/(p1 + p2)$

This formulation, as it stands, would (other things being equal) induce the power of one prediction to be equal to the other prediction. In order to let the balance of power be more flexable, I chose a method of altering this ratio so that the balance of power would only take strong effect if the power of one signal grew 'far beyond' the power of the other signal. This was achieved through the use of a squashed exponential function applied to the previous 'p':

12. $p' = (2*(p-0.5))^{**}3 + 0.5$

Results: Correlation

While the balance of power did induce both predictors to respond, both predictors inevitably came to look identical: each came to model ½ of the signal, and the results of the predictors mirrored each other. This was not what I had thought would happen; I thought that the predictors would diverge so that they could better represent the signal. When this did not happen, I began to wonder how to force the predictions apart, without 'telling them' exactly where they should go.

The first idea that I had in this respect was to add an element of 'random jitter' to the weight adjustment. I figured that if I slightly varied the responses, and reinforced results which were beneficial, that it could help transform the MLP (which is good at learning error surfaces) into a mechanism that had some degree of selection, a la 'genetic programming'. Although I took at stab at this, it did not work out, and I was not clear of the theoretical underpinnings, so I abandoned this line of persuit.

At this point I tried to force the decorrelation of the two sequences. I began with the classic definition of correlation (defined in terms of products, E(y1*y2)), but I found the following fomula to express the power of the sum of two signals:

13. E(y**2) = E(y1**2) + E(y2**2) + 2*E(y1)*E(y2)

This was exactly the situation that I had. I wanted to eliminate the last term (the correlation term), so I adopted the following constraint for the model:

14. E(y**2) = E(y1**2) + E(y2**2)

This formulation has the added benefit of balancing the power of the two predictors automatically. Unfortunately, this fomulation is not a linear function on the output, and thus it produces a non-quadratic error surface. I am unfamiliar with a proceedure to handle such an error function, and I could not find one in the literature. I thought that it may be possible to use a second-order network to get around this in order to require a second order term from the network directly, but I decided to drop the idea.

Another way to penalize output correlation was formulated in terms of the non-orthogonality of the weight space (the inner product). This did not have a significant effect on the results.

Results: Two Alternative Forced Choice

At this point in the project I became discouraged, so I decided to at least achieve a simpler task. Using two-alternative forced choice could split a temporally uncorrelated signal, although this is not at all the task that I wanted to achieve. Two AFC involves choosing one model to be the sole predictor for a given instant, and then using only its prediction.

The determination of which model should be the predictor (and thus be trained) was formulated in terms of prediction error; the better predictor was trained, and the other was not. Similarly, only the output of this better predictor was used. The error functions for controlling the mean and balancing the power between the two predictors were used, in addition to the basic error which resulted from the distance between the chosen predictor and the value to be predicted.

After this was implemented, it became clear that the predictors would alternate (in terms of prediction period) at relatively random intervals. In other words, there was no correlation between the onset/offset times of the signals and the choice of the predictor. In order to avoid this situation, I used knowledge about the signal; namely, given that the signal occurred for a brief instant, it would be present for at least a little while longer. This knowledge of the signal was translated as a monotonically decreasing exponential potentiation upon switching from one predictor to the next.

Results: A Better Forced Choice

The previous method kept each predictor 'active' for a reasonable amount of time, but it did nothing for the fact that predictor onset was uncorrelated with event/signal onset. Thiking about this, it became clear why: it is trivial for event a random predictor to be closer to a given target value once in a while. This, of course, does not mean that the random predictor is any good at predicting the signal. In order to get a better measure of this, it became necessary to evaluate the performance of the predictor over a range of time. This was achieved with a gamma memory of past performance, which was set up as follows:

15. ChoiceErr(y1(t)) = .1*Error(y1(t)) + .9*ChoiceErr (y1(t-1))

This model was *almost* right, however it did not take into consideration that one of the sources might be harder to predict in general than the other signal. In order to compensate for this effect, I compared the short-term accuracy of the predictor to the long term accuracy of the signal. The ratio between these accuracies was then compared to determine which of the two predictors would be chosen as the next predictor. The ratio of error between the two alternatives was thus computed as follows:

16. ShortTermErr(y1(t)) = .1*Err(y1(t)) + .9*Err(y1(t))
17. LongTermErr(y1(t)) = .001*Err(y1(t)) + .999*Err(y1(t))
18. LocalErr(y1(t)) = ShortTermErr / LongTermErr
19. ErrRatio(t) = LocalErr(y1(t)) / (LocalErr(y1(t)) + LocalErr(y2(t)))

Note that we have the case of two alternative forced choice when we set up an if-then rule using the following boolean criterion:

20. ErrRatio(t) < 0.5.

Using this criteria for model selection was able to separate the signal fairly well. Although 2AFC was not the desired outcome of this project, at least I achived a lesser goal, upon which I hoped to build slowly.

Results: Fuzzy Choice

We can make the prvious two-alternative choice fuzzy by using the degree of the error ratio to determine *how much* of a given models prediction we should use, instead of using it as a binary choice. This gives us the following formulation for prediction:

21. y = y1*(1-ErrRatio) + y2*ErrRatio

This was not effective, so the ErrRatio was modified by a logistic sigmoid function which would allow us to smoothly move between 2AFC and a linear combination of the results in proportion to model accuracy by altering the slope of the sigmoid at the origin. Clearly, at one end is the Heaviside function, which represents two alternative gforce choice. At the other end is a linear function, which (as explored previously) did not work.

22. y = Sig(y1*(1-ErrRatio)) + Sig(y2*ErrRatio)

The results of varying this parameter did not help to decompose the signal. Altering the slope of the sigmoid resulted in fairly chaotic and incorrect behavior.

Conclusion: Future Directions

There are two fairly simple formulae which I would like to set up a network to solve:

2. u(t+1) = y1(t) + y2(t)
14. E(y**2) = E(y1**2) + E(y2**2)

Unfortunately, I could not.

I have attempted to implement a Neural Dual Extended Kalman Filter, following Wan [] who achieved some success with splitting a monaural signal with this method. Unfortunately, I barely understand the Kalman filter, much less the "neural dual extended' Kalman filter. Very frustrating, this.

It might be nice to explore the possibility of using a combinatorial number of predictors, w.r.t the number of input vectors. For example, S1 + S2 can result in the following possible combinations: {S1+S2}, {S1}, {S2}, {0}. Brief exploration of this idea failed.

The problem that I set out to achieve was substantially harder than I imagined. I found my knowledge of the area (especially linear models) severely lacking. Specifically, I think my general experimental model suffered from two HUGE problems, which would prevent any model of this kind from succeeding: lack of stationarity not understanding Higer Order Statistics.

Stationarity is a problem in that correlational filters require a stationary time series which to model. The event structure which uderlies this experiment is not stationary, and this problem only gets worse when two signals are comingled. This question is directly linked with the convergence time of an adaptive filter relative to the length of an event, during which the signal can at least be assumed to be semi-stationary.

Even more serious than this is the problem of only using second-order statistics (correlation) to capture information about the signal. I do not know enough about the theoretical underpinnings of Higher Order Statistics, but it seems from the literature that I have read on BSS that second order statistics are not sufficient to seperate a signal into its constituents.

Appendix A : Formula Reference

1. u = u1 + u2
2. u(t+1) = y1(t) + y2(t)
3. err(y1, t) = u(t+1) - y2(t)
4. err(y2, t) = u(t+1) - y1(t)
5. y1 = u1 || y1 = u2
6. E(y(t)) = Gamma * y(t) + (1-Gamma) * E(y(t-1))
7. err(y1, t) += E(y(t))
8. u(t+1) = (1-p) * y1(t) + p * y2(t)
9. p1 = E(y1**2)
10. p2 = E(y2**2)
11. p = p1/(p1 + p2)
12. p' = (2*(p-0.5))**3 + 0.5
13. E(y**2) = E(y1**2) + E(y2**2) + 2*E(y1)*E(y2)
14. E(y**2) = E(y1**2) + E(y2**2)
15. ChoiceErr(y1(t)) = .1*Error(y1(t)) + .9*ChoiceErr (y1(t-1))
16. ShortTermErr(y1(t)) = .1*Err(y1(t)) + .9*Err(y1(t))
17. LongTermErr(y1(t)) = .001*Err(y1(t)) + .999*Err(y1(t))
18. LocalErr(y1(t)) = ShortTermErr / LongTermErr
19. ErrRatio(t) = LocalErr(y1(t)) / (LocalErr(y1(t)) + LocalErr(y2(t)))
20. ErrRatio(t) < 0.5.
21. y = y1*(1-ErrRatio) + y2*ErrRatio
22. y = Sig(y1*(1-ErrRatio)) + Sig(y2*ErrRatio)

Bibliography

Bell, Anthony, and Sejnowski, Terrence: Learning the Higher-Order Structure of a Natural Sound, 1996

Bell, Anthony, and Sejnowski, Terrence: An Information-maximization Approach to Blind Seperation and Blind Deconvolution, 1995

Chan, D.C.B, Godsil, S.J. Rayner, P.J.W.: Multi-Channel MUlti-Tap Signal Seperation by Output Decorrelation, 1996

Haykin, Simon: Adaptive Filter Theory (third edition), Prentice Hall, 1996

Haykin, Simon: Neural Networks: A Comprehensive Foundation (second edition), Prentice Hall, 1999

Kaarhunen, Juha: Neural Approaches to Independant Component Analysis and Source Seperation, 1992

Lee, Te-Won & Girolllami, Mark: A Unifying Information-Theoretic Framework for Independant Component Analysis, 1998

Luo, Fa-Long & Unbehauen, Rolf: Applied Neural Networks for Signal Processing, Cambridge University Press, 1998

Mittal, Udal & Phamdo, Nam: Signal/Noise KLT Based Approach for Enhancing Speech Degraded by Colored Noise, IEEE Transactions on Speech and Audio Processing, Vol 8 No 2 2000

Sutton, Richard: Learning to Predict by the Method of Temporal Differences, Machine Learning 3: 9-44, 1988

Wan, Eric & Nelson, Alex: Neural Dual Extended Kalman Filtering: Applications in Speech Enhancement and Monaural Blind Signal Seperation

Wan, Eric: Time Series Prediction by Using a Connectionist Network with Internal Delay Lines