Independent Component Analysis with Neural Networks

Project category: Applications
Raúl Rojas-González
SUNet ID: raulr@stanford.edu

CS230 Final Report

Problem and motivation

The problem was stated in the midterm proposal: given a linear mix of *m* time series, process them with a neural network in order to recover the mixing matrix M. The unmixing matrix W (inverse of M) provides then the reconstructed signals.

The cocktail party effect

When many people talk at the same time in a room, if we record their voices we obtain a superposition of all of them. We can use several microphones, each located at a different position. We can then try to separate the different audio sources taking advantage of the fact that each voice will have a different amplitude relative to each microphone (when the microphones are distributed across the room, or in an array with sufficient separations).This is a classic problem in signal processing: it is called "blind source separation" or "independent component analysis" (ICA).

Fig. 1: The cocktail party problem

As we all know, people are capable of concentrating on one single voice when confronted with a cacophony of many superimposed voices. The acoustic cortex can zoom-in and separate the voices, focusing on just one of them.

The spectrograms below show the mixing of frequencies when one, three or seven additional speakers are added to a recording. The spectrogram data becomes almost impossible to use for a speech recognition system.

My hypothesis with this project is that once a person locks-in on a specific voice (the time series of microphone data), it is possible to "track" the conversation of that single voice by concentrating on the predictability of the next points in the time series.

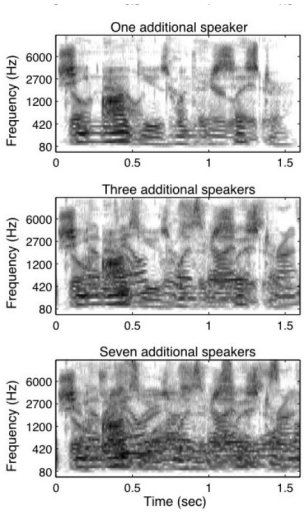
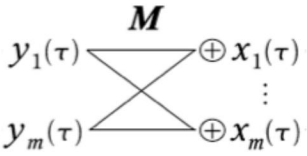


Fig. 2: The spectrogram for several speakers in the same microphone (Karczweski 2014)

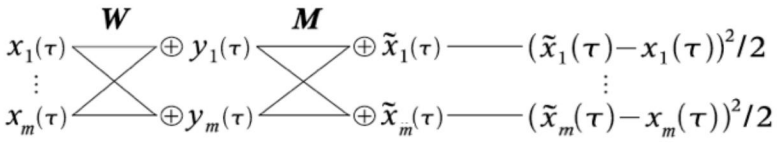
A possible application of this project is to do real-time speech recognition of voice data, even when the signal is corrupted by several people talking in the background (using some kind of microphone array, inside a car, for example).

The mixing-unmixing process

Let us call $x_1(\tau)$ to $x_m(\tau)$ the *m* mixtures of the original time series $y_1(\tau)$ to $y_m(\tau)$ recorded in *m* microphones. It would be possible to have more or less microphones, but in this project, I work with m microphones for *m* time series mixtures. We can conceive of the mixing process as the input to m linear units, whose weights are represented by the mixing matrix M. Each time series $x_i(\tau)$ is in general a different linear combination of the *m* time series $y_i(\tau)$.



Since this is a linear process, we can apply an unmixing matrix W to each point of the *m* time series from the microphones in order to reconstruct the original signals y_1 to y_n (*m*=*n*). The unmixing matrix W is initialized as the identity matrix and is constructed using an iterative approach.
Since M is unknown, what we want to do is to find the matrix M together with the unmixing matrix W (the inverse of M), as shown in the diagram below.



The mixing matrix M mixes the reconstructed signals and delivers an approximation to the original microphone data (i.e., the time series $\tilde{x}_i$ represent an estimation of the "reconstructed" mixtures using the matrix M, plus the computation error). The matrix W is therefore the inverse of the matrix M.

As stated, the problem seems ill-posed, because both M and W are unknown matrices. We need to add some kind of constraint to the type of signals that we want to reconstruct. Our constraint will be "better predictability", that is, if we have a mixture of two voices (a man and a woman, for example), the idea is that a classifier would be able to better predict the time series corresponding to each separate voice. We prefer reconstructed signals *y* which in some sense are as smooth as possible. We will use the criterion that the signals can be predicted by a network where some regularization criterion constrains its complexity. In telephony, Linear Predictive Coding is used to predict the next speech data based on the last *k* speech frames. We adopt this approach here, although any kind of regularized network could be used.

Predicting the time series

I follow the usual approach for predicting time series, which consists in using a sliding window of *k* data points which is used to predict the point *k*+1 in the time series. This is the strategy used in auto-regressive moving-average models, as shown in Fig. 3.

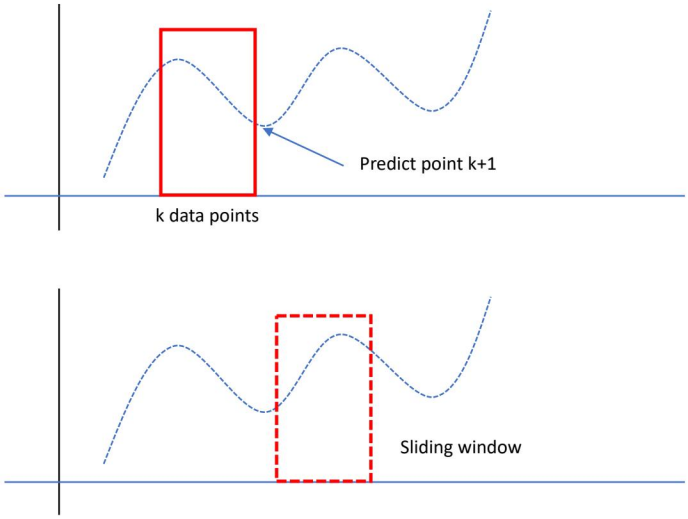


Fig. 3: A sliding window is used to predict the time series

The sum of all prediction errors, for all the k-point windows, is the total prediction error for one signal. If we think of the sliding window as weights of a NN-layer, this would correspond to a

convolutional network acting on input of dimension 1 (the totality of data points in the time series).

Network architecture

What we do is then construct a network which contains all these elements. Fig. 4 shows the composite network. The first parts is the decoding-encoding part, where the microphone data is unmixed by the matrix of weights W, and mixed again by the mixing matrix M (I show only the process for a single time step τ). The error represents the goodness of the decoding-encoding submatrix over all time-steps.

For each reconstructed time series *y* we have a convolution using a sliding window of weights that have to be fitted (in each NN box) and which produces a prediction error for all time steps.

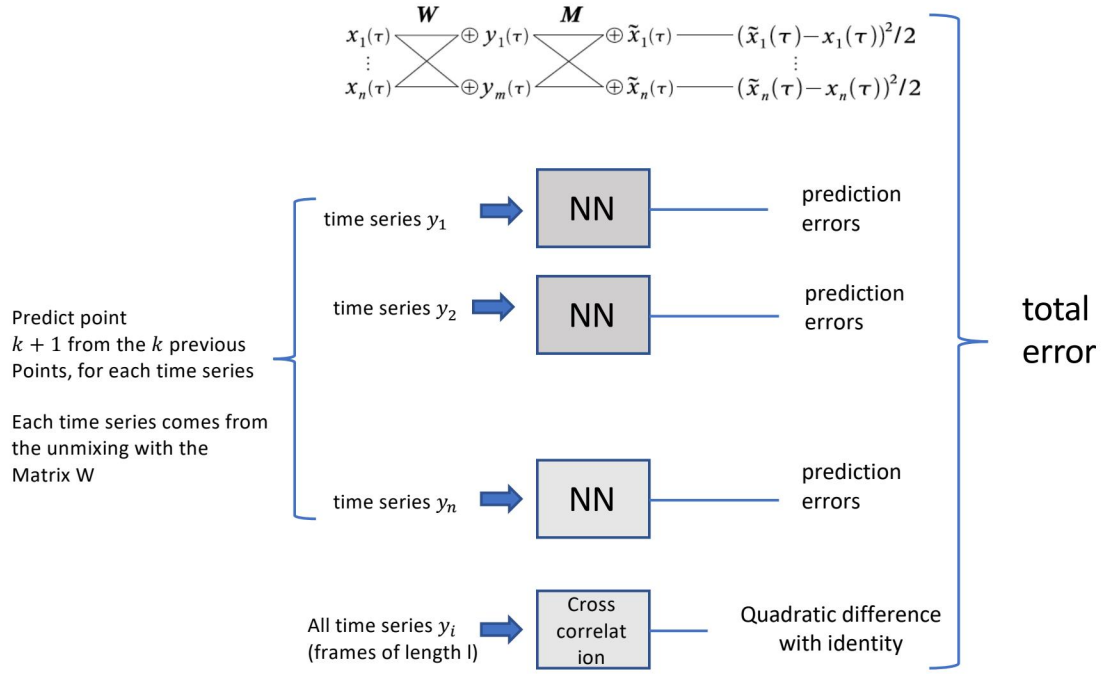


Fig. 4: The architecture of the network. It has an decoding-encoding "bottleneck" and one convolutional layer for each reconstructed signal *y*, as well as a computation of the cross-correlation between signals.

Finally, we would like to have some control on the cross-correlation of the reconstructed signals, without giving this part an excessive weight (each portion of the produced errors is weighted with a hyperparameter). If for example, we only use decoding-encoding matrix any pair of invertible matrices M and W would be a solution. By increasing the weight of the prediction error for the reconstructed signals *y* we avoid this flaw.

For the convolutions of the time series (NN boxes), the points $y_1(t-1)$ to $y_1(t-k)$ are used to predict the point $y_1(t)$. The sliding window is of length *k* running over the time series. The signals

predicted by the linear units are called $\widetilde{y_l}$ and their squared difference to y_l is the prediction error.

Backpropagation

I apply backpropagation to the composite network in order to minimize the total error. BP then tries to optimize the predictability of each reconstructed signal, but at the same time, the reconstructed signals have to mix well in order to reproduce the microphone signals. As explained before, the intuition is that the lowest error will be achieved by independent components.
I applied standard backpropagation to the weights in the network, including M and W. A sanity check for the result of the computation is testing whether M is the inverse of W or not.

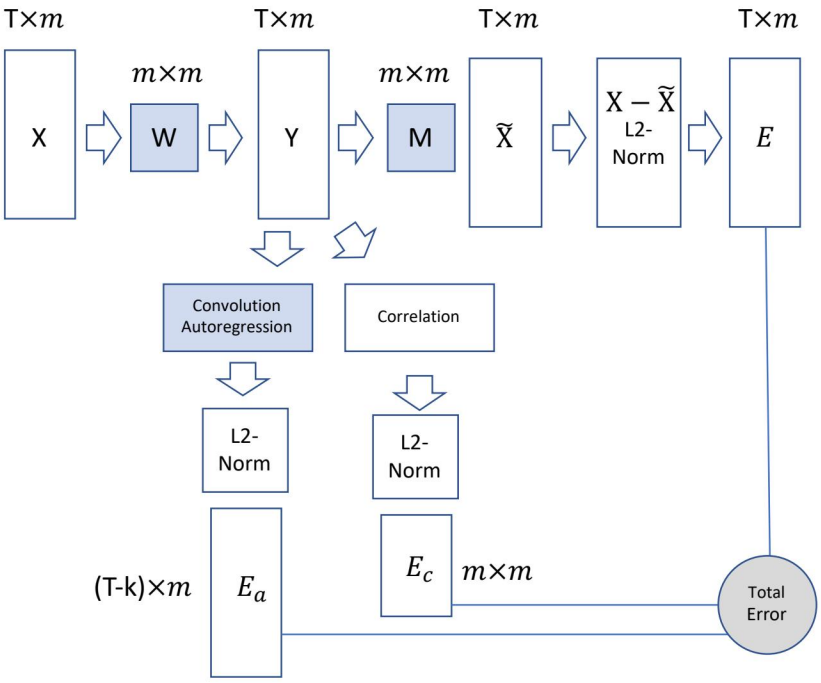


Fig. 5: The flow of tensors in the network

Fig. 5 shows the flow of tensors in the network. The m microphone signals are contained in the $T \times m$ matrix X (each time series is a column of X, T is the total number of time steps). Multiplying X with the unmixing matrix W produces the $T \times m$ matrix Y of reconstructed independent signals. The product of Y with the mixing matrix M produces the estimation $\tilde{X}$ of rederived microphone signals. The quadratic norm of the individual differences for the entries of the matrix $X - \tilde{X}$ is the matrix *E* (of size $T \times m$).

Along a different computation path, we perform a predictive convolution of the *m* time series in the matrix Y (each time series is a column of the matrix) using a window of size k. We can predict $(T - k)$ points in each time series. The squares of the prediction errors are saved in the matrix

E_a. The squared differences of the autocorrelation matrix compared to the identity matrix (for the *m* reconstructed time series) are stored in the $m \times m$ matrix E_c. The sum of all squared errors is the total error of the network. The shaded boxes represent layers of the network with parameters (weights) that have to be adjusted by backpropagation. The desired weighted mixing of the three sources of errors is provided by hyperparameters not shown in the diagram. As it is clear from the diagram, the network is represented by a non-cyclic graph in which the backpropagation algorithm can be applied.

The backpropagation corrections, for example, for the upper part of the network (the decoding-encoding part), can be found easily. We have the following equations:

$$XW = Y$$
$$YM = \tilde{X}$$
$$E = q(\tilde{X} - X)$$

Where *q* is a function which computes the element-wise squared differences of $\tilde{X} - X$. If we call G the matrix of the element-wise differences $\left(\tilde{X} - X\right)$, the BP corrections to the matrix M are given by

$$\Delta M = -\gamma Y^T G$$

where γ is the learning rate. The backpropagated error up to after the layer M is given by MG^T so that the corrections for the matrix W are given by

$$\Delta W = -\gamma X^T G M^T$$

These corrections, in isolation, would force the matrices M and W to be inverses of each other, i.e. any pair of inverses (assuming that the original signals and the mixed time series are non-degenerate). But there are two other paths in the network leading back to the original input and going through shaded boxes (with trainable parameters). While the corrections for the matrix M go through one single path, the corrections for the unmixing matrix W profit from the constraint that we prefer better predictability of the reconstructed signals. W is the matrix really being adjusted, M is being forced to be the inverse of W. However, the computation of M is very important in order to avoid the trivial result of predicting a matrix Y full of zeros. Keeping M prevents the degeneracy of the reconstructed signals in Y.

Datasets and Results

The results obtained with synthetically generated time series are very good. I generated independent sets of waveforms combining sinusoidal signals which were a mixture of various phases, frequencies and amplitudes. The error is negligible for these synthetically generated waveforms (Fig. 5).

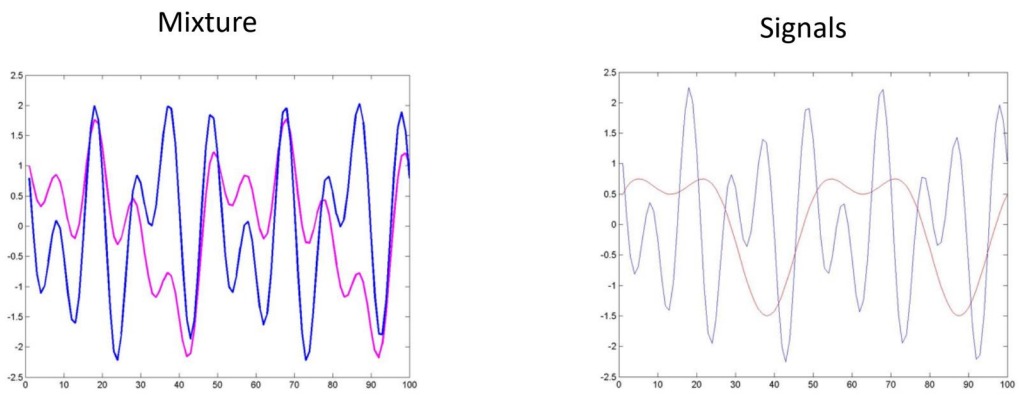


Fig. 5: Recovery of the original signals from two mixtures

I also did some low-scale testing with audio data. Fig. 6 shows and example of two recordings, one is a voice, the other is applause.

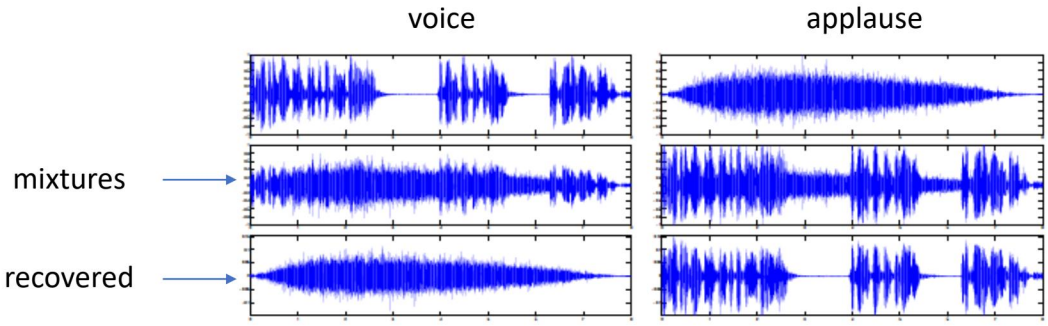


Fig. 6: Recovery of two acoustic data tracks from two mixtures (middle row)

Conclusions

The small-scale experiments I conducted proved that the approach is sound and can be used to separate linear mixtures of complex signals. I would have liked to do tests with a more challenging and larger data set, but I ran out of time due to my work (I was traveling in three countries in late November and the first half of December). My business travels made it also impossible to join other two persons to form a group of three.

Synthetic data is actually easy to separate and the network converges in a few seconds. Voice data is more challenging and processing more than two voices would possibly require a more complex convolutional layer.

Substituting the linear units by more general networks, could allow capturing non-linear mixing effects. I would like to extend this approach to the deconvolution of nonlinearly mixed time series. Another thing that could be considered in the future is using recurrent units, so that the autoregressive process is more powerful and can cover long pauses in the input in one or more time series.

A recorded PowerPoint presentation of this report and the poster can be found at:
https://www.youtube.com/watch?v=pvsSFVYxpmY&feature=youtu.be

Future work

I would like to make the network more complex and capable of modeling more complicated signals. I would like to test with a different number of microphones in realistic settings (inside a car or in a bar).

References

- Aapo Hyvärinen, Erkki Oja und Juha Karhunen, *Independent Component Analysis*, Wiley, 2002.
- Konrad Karczweski et al., „Coherent Functional Modules Improve Transcription Factor Target Identification", *PLOS Genetics*, Feb. 2014, V. 10, N. 2
- Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, MIT Press, November 2016.