Blind Source Separation Via ICA: Implementation*

Mark Eastaway
John Steinbauer
Angela Qian
Akshay Dayal

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 2.0†

Abstract

This module details the implementation of a blind source separation system using fast ICA.

1 Blind Source Separation via ICA:

1.1 Implementation

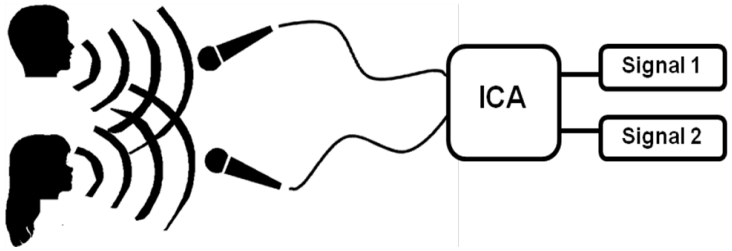

Figure 1

In order to implement our design to enable the separation of two audio signals, we require two microphones and a processing computer with audio output capabilities. When two microphones are present in an environment with two sources, then they will record a mixing of both these signals, weighted by coefficients base on distance away from the microphone (remember that as a signal is further away from a recording source, the quieter it will be).

*Version 1.2: Dec 19, 2007 8:39 pm -0600
†http://creativecommons.org/licenses/by/2.0/

The mixed signal inputs of these microphones must then be imported into a processing computer that is able to run code based in C or MatLab (or potentially any other language, but the most efficient algorithm is run in MatLab and C).

The model for these mixed signals can be represented in matrix notation by:

$\mathbf{x} = \mathrm{A}\mathbf{s}$

As we can see, the signal vector is multiplied by some **mixing matrix** A. Therefore to isolate the signals **s** based on the mixed signals **x** we must find an inverse matrix A-1. To do this we implement a piece of Matlab code known as **FastICA**. The math behind the algorithm is explained elsewhere but the most important piece of code, the iteration to find the inverse mixing matrix, is displayed here:

```
% Take a random initial vector of length 1 and orthogonalize it
% with respect to the other vectors.
if initialStateMode == 0
w = randn (vectorSize, 1);
elseif initialStateMode == 1
w=whiteningMatrix*guess(:,round);
end
w = w - B * B' * w;
w = w / norm(w);

wOld = zeros(size(w));
wOld2 = zeros(size(w));

% This is the actual fixed-point iteration loop.
% for i = 1 :  maxNumIterations + 1
i = 1;
gabba = 1;
while i <= maxNumIterations + gabba
if (usedDisplay > 0)
drawnow;
end

% Project the vector into the space orthogonal to the space
% spanned by the earlier found basis vectors.  Note that we can do
% the projection with matrix B, since the zero entries do not
% contribute to the projection.
w = w - B * B' * w;
w = w / norm(w);

if notFine
if i == maxNumIterations + 1
 if b_verbose
 fprintf('\nComponent number %d did not converge in %d iterations.\n', round, maxNumIterations);
 end
 round = round - 1;
 numFailures = numFailures + 1;
 if numFailures > failureLimit
 if b_verbose
 fprintf('Too many failures to converge (%d).  Giving up.\n', numFailures);
 end
 if round == 0
 A=[];
```

```
W=[];
end
return;
end
% numFailures > failurelimit
break;
```

This iteration will guess a row of the **demixing** matrix (w in the code) and then run through a loop until it finds a projection that agrees with the statistical analysis behind the decoding.

After running on the supplied mixed signals, the program will output what it thinks are the two original sources and the **demixing** matrix. We can then use these outputs to complete a variety of tasks such as removing noise from the original signals, matching the original signals with other signals to detect base elements of the mixed signal, or even just simply outputting the original signals through a speaker system.