ECSE 444: Microprocessor

Hints on FastICA

Winter 2019

One of the important aspects of signal processing applications is unmixing of signals that have been generated and collected through different sources. Blind Source Separation (BSS), also known as Blind Signal Separation, is the process of separating a set of source signals from a set of mixed signals without the aid of information about the source signals or the mixing process.

In this project, BSS deploys Fast Independent Component Analysis (FastICA) to unmix the signals. This document provides some guidance for implementing FastICA algorithm on Keil uVision IDE using CMSIS_DSP.

If we consider that $s_1(t)$ and $s_2(t)$ are the source signals, $x_1(t)$ and $x_2(t)$ are the recorded mixed signals, and A is the mixing matrix, there is a linear relation between the source and the mixed signals as follows:

$$\begin{bmatrix} x_1(t) \\ x_1(t) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} . \begin{bmatrix} s_1(t) \\ s_2(t) \end{bmatrix} \rightarrow X = AS$$

FastICA is going to be performed to find W which is the inverse of mixing matrix A:

$$A^{-1}X = A^{-1}AS \rightarrow A^{-1}X = S$$
$$W = A^{-1} \rightarrow WX = S$$

The objective of FastICA algorithm is to separate a mixed signal into its original components. It should be noted that FastICA only estimates the original signals, so the unmixed signals may be slightly different from the original signals. For implementing FastICA algorithm, different functions of the CMSIS-DSP (see here) library may be used.

The CMSIS-DSP library contains many of the core data processing features needed, and it is being designed for Cortex-M processors, so that it ensures the code is optimized for the development platform. As an example, for generating *sine* waves, one can use `arm_sin_f32()` function of CMSIS-DSP library.

Different Steps of BSS

BSS is mainly composed of different steps which will be described abstractly in the following paragraphs. It is assumed that you have a clear understanding of BSS algorithms, and are familiar with different steps and functions (which can be acquired from here).

1. The first step of BSS, which mainly consists of preprocessing the signals, is **centering** the signals. In centering process, you need to calculate the *mean* (μ_1, μ_2) of the signals and subtract the mean from each single value in order to center the values. By performing these calculations, you assure that the mixed signal matrix has zero-mean values.

2. You need to multiply the mixed samples by its transpose in order to obtain the variance-covariance matrix of the mixed samples.
3. Then, you need to find eigenvalues (**E**) and eigenvectors (**D**) of the variance-covariance matrix. This step requires solving a quadratic equation and a pair of simultaneous linear equations.
4. The result of the previous step is a diagonal matrix (D) and the matrix of eigenvalues (E) which can be saved and used as instances of the *arm_mat_f32* structure which are provided by the CMSIS-DSP library to make use of the matrix operations.
5. Then, you need to obtain ***whitening*** and ***de-whitening*** matrices. These matrices are calculated as follows:

$$whitening\ matrix = \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{bmatrix}^{-1} \times E^T$$

$$de - whitening\ matrix = E \times \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{bmatrix}$$

   where λ_1 and λ_2 are the eigenvalues of the variance-covariance matrix.

   For implementing the previous equations, you may find the following functions useful:

   - *arm_mat_mult_f32()* → implements matrix multiplications
   - *arm_mat_inverse_f32()* → implements matrix transpose
   - *arm_mat_trans_f32()* → implements matrix inverse

   The whitening matrix is later used to calculates the white signal by multiplying it with the mixed signal.

6. The final step to retrieve the original signals is to pass the mixed signals through the ICA filter and add the mean value back in:

$$\begin{bmatrix} s_1'(t) \\ s_2'(t) \end{bmatrix} = W \times \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + W \times \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$$

   where $s_1'(t)$ and $s_2'(t)$ are the approximations of the original signals $s_1(t)$ and $s_2(t)$ that we started with, and $x_1(t)$ and $x_2(t)$ are the mixed signals.

Some More Notes to Consider

There are different considerations that may help you in correctly implementation of FastICA algorithm:

- You need to use matrix components of the CMSIS-DSP library for implementing functionality needed to perform sine wave mixing and FastICA. It should be noted that you may face some situations where certain arm matrix functions do not return correct results in the event that the same matrix is passing as both the input and outputs. For instance, *ARM* inverse and transpose functions need to have separate source and destination matrices, otherwise the input values are altered when the relevant calculations are performed.
- Moreover, it is important to note that the eigenvector need to be normalized for the algorithm to work.