Reimplementation of AMICA in Julia

Alexander LULKIN[1]

[1]University of Stuttgart, Universitätsstraße 38, D–70569 Stuttgart
st108415@stud.uni-stuttgart.de

Abstract – Independent Component Analysis is commonly used on electroencephalography data for blind source separation. Various ICA algorithms exist, among them Adaptive Mixture Independent Component Analysis. AMICA is widely used in EEG due to being one of the best performing ICA algorithms. However, the current implementations are not transparent and not perfectly suited for further development. This thesis aims to produce an implementation in the scientific programming language Julia. The implementation will feature a modular structure, which will make it easy to replace certain components of the algorithm to find better performing variations.

1 Introduction

Electroencephalography (EEG) is a method to record electric signals in the brain. This is achieved by placing electrodes along the scalp. The non-invasiveness of this method is it's biggest advantage, but it also comes with some drawbacks. The signals might mix and interfere with each other, making it harder to identify different sources in the brain. Noise also poses an issue. For example, discharges from neuromuscular activity might get recorded alongside the brain activity and need to be filtered out with post-processing techniques. One technique, that can be used to tackle both of these issues, is Independent Component Analysis (ICA) [4]. ICA algorithms model the data as a linear combination of the source signals. Different variations of ICA exist, one of which is Adaptive Mixture Independent Component Analysis (AMICA) [6]. AMICA is a multi-layered mixing network which combines multiple ICA mixture models with mixture of generalized Gaussian distributions. The method has been successfully implemented and used in different programming languages, such as Matlab or Fortran. These implementations however come with their own disadvantages. For example, the Fortran implementation is hard to maintain, inspect and further develop. The Matlab implementation on the other hand is easier to develop, but suffers from worse performance. The main goal of this thesis is to implement the algorithm in Julia, a scientific programming language which was designed as a new approach to numerical computing [2]. This new implementation is supposed to ease further development of the algorithm by having a modular structure.

2 Mandatory and Optional Goals

The following goals are mandatory:

1. Implementation of the adaptive source density models
2. Implementation of the ICA mixing network

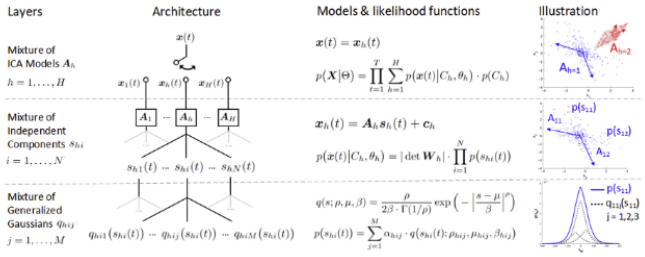

Figure 1: The three layers of AMICA (Image by Sheng-Hsiou Hsu et al., 2018)

3. Evaluation of the correctness of the algorithm
4. Measurement of the computation time

The following changes could be made and benchmarked as optional goals:

1. Replacing manual calculations of gradient/hessian with automatic differentiation
2. Replacing Newton Method with an estimation based method
3. Evaluating different Rejection Schedules

3 Methods and Approach

To achieve the first goal of creating a functioning implementation of AMICA in Julia, the following methods will be used. The algorithm will be implemented as it is described in the paper by Sheng-Hsiou Hsu et al. [6]. Furthermore the already existing implementations in Fortran and Matlab will serve as guidelines for how a working implementation might look like. *GitHub* will be used to allow for collaboration between the student and the supervisor. The Julia implementation will make use of packages such as Optim.jl, ForwardDiff.jl, MultivariateStats.jl and GaussianMixtures.jl.

3.1 The AMICA Algorithm

Blind source separation is a problem, in which multiple receivers receive a different mix of signals from different sources [3]. The goal is to identify the independent sources without knowing anything about the structure of the mix. Independent Component Analysis (Fig. 2) achieves this by modeling the data as a linear combination of the source signals. Due to the *Central Limit Theorem*, mixed independent variables can be assumed to at least approximately have a Gaussian joint distribution. Therefore the signals can be considered unmixed if the joint distribution becomes non-Gaussian. ICA achieves this by iteratively improving an unmixing matrix [4]. Adaptive Mixture Independent Component Analysis (see Fig. 1) takes this one step further. It models the data with multiple ICA models at once by assuming that a different model might be active for a different sample [6]. These models can be learned simultaneously and components can even be shared between those models. Another core feature of AMICA are the adaptive source density models. While other ICA mixture models might assume pre-defined probability density functions for the sources, AMICA adaptively learns the PDFs for each source. This is achieved with a mixture of generalized Gaussian distributions.

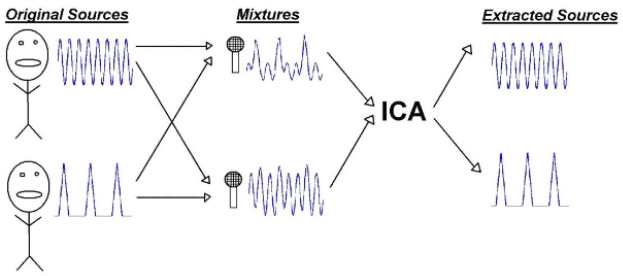

Figure 2: ICA can be used to extract sources from mixtures (Image by Dominic Langlois et al., 2010)

3.2 Benchmarking

After implementation is complete, the very first test should check if the algorithm produces the same results as the previous implementations. This will be done on simulated data. If the results match, then the next step is an evaluation via the ICA evaluation framework developed by Gwenevere Frank et al. [5]. The framework provides different metrics to evaluate the performance of an ICA algorithm: *Pairwise Mutual Information*, *Mutual Information Reduction* and *Equivalent dipole modeling*. In later stages of the project, the framework can also be used to compare different variations of the implementation with each other. Furthermore, the execution time of the algorithm will be measured. This will be done for both total time and time per step. All those metrics plus the computed log likelihood functions will be compared with the previous implementations in Matlab and Fortran.

3.3 Optional Goals

There are a lot options to explore after the basic implementation of AMICA has been coded, debugged and benchmarked. For example, since Julia supports Automatic Differentiation [8], some calculations could be replaced with autodiff functions and the performance could be compared to manually coded functions. This is not expected to increase the performance, but if there is no significant performance loss it would offer a valuable option for faster coding.

Another option would be to replace the Newton method, which is used in AMICA and requires the calculation of a Hessian matrix, with a method that does not use the Hessian [1] and again, benchmark the impact on performance.

It would also be interesting to measure the efficacy of different rejection schedules. AMICA allows to exclude outliers if their deviation from the average log likelihood passes a certain threshold. The threshold is a multiple of the standard deviation and can be set by the user. This comes with certain issues. The standard deviation itself gets inflated by the outliers, because it is calculated before any cleaning takes place. A possible improvement to this would be to replace the standard deviation with a robust method, like median absolute deviation [7]. Different rejection schedules can be compared with each other by running them on data with added noise.

4 Intended Outcomes

The intended outcome of this project is to produce an AMICA implementation which is performant and has a modular structure. This is supposed to make the replacement of certain components of the algorithm much easier in the future.

If the optional goals can be achieved, then another intended outcome is to produce benchmarks of different variations of the implementation.

5 Schedule with Milestones

Figure 3 shows a rough schedule for the project. The goal during the first month is to start working on the AMICA implementation in Julia, which will be the first and most important milestone of this project. Before the practical work on the implementation can start, some additional research might be necessary to get comfortable with the material. This includes tutorials for the Julia programming language and should take maximally one or two weeks. In order to leave enough time for optimization and benchmarking, the basic implementation of the algorithm should be completed by the end of May. During the final weeks of the implementation, research on possible optimization methods should start. This will ensure a smooth transition to the work on the second milestone, which consists of an optimized version of the algorithm and/or code and benchmarks, which compare it to the previous implementations. The third and final milestone of the project will be the written thesis.

The writing will start alongside the coding right at the beginning and will be finished by the very end of the project. The thesis should be complete content-wise by the end of July, leaving enough time for revision before it is submitted. The final month also serves as a buffer, since issues are guaranteed to arise during a project of this scope.

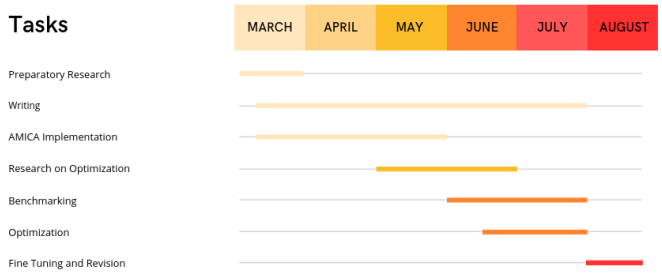

Figure 3: Project Schedule (Chart created on Canva.com)

References

[1] Pierre Ablin, Jean-François Cardoso, and Alexandre Gramfort. Faster independent component analysis by preconditioning with hessian approximations. *IEEE Transactions on Signal Processing*, 66(15):4040–4049, 2018.

[2] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[3] Xi-Ren Cao and Ruey-wen Liu. General approach to blind source separation. *IEEE Transactions on signal Processing*, 44(3):562–571, 1996.

[4] Pierre Comon. Independent component analysis, a new concept? *Signal processing*, 36(3):287–314, 1994.

[5] Gwenevere Frank, Scott Makeig, and Arnaud Delorme. A framework to evaluate independent component analysis applied to eeg signal: testing on the picard algorithm. In *2022 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2009–2016. IEEE, 2022.

[6] Sheng-Hsiou Hsu, Luca Pion-Tonachini, Jason Palmer, Makoto Miyakoshi, Scott Makeig, and Tzyy-Ping Jung. Modeling brain dynamic state changes with adaptive mixture independent component analysis. *NeuroImage*, 183:47–61, 2018.

[7] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of experimental social psychology*, 49(4):764–766, 2013.

[8] Jarrett Revels, Miles Lubin, and Theodore Papamarkou. Forward-mode automatic differentiation in julia. *arXiv preprint arXiv:1607.07892*, 2016.