An introduction to ICA followed by: EM Algorithms for ICA

Pierre Ablin
Parietal
Joint work with: F. Bach, JF. Cardoso & A. Gramfort

https://arxiv.org/abs/1805.10054

Parietal presentation, 2018
Introduction to ICA
Source separation: the cocktail party problem
Independent component analysis

Special case of source separation:

- Linear & instantaneous mixture
- “Square problem”: as many sources as sensors

\[
\begin{align*}
x_1 &= 1.1s_1 + 0.9s_2 + 1.2s_3 \\
x_2 &= 0.5s_1 + 0.8s_2 + 2.2s_3 \\
x_3 &= 1.5s_1 + 0.5s_2 - 2.4s_3
\end{align*}
\]
Problem formulation: ICA as a generative model

- We observe $p$ signals $[x_1, \cdots, x_p] = \mathbf{x} \in \mathbb{R}^{p \times 1}$

Key assumption

There are $p$ independent signals $[s_1, \cdots, s_p] = \mathbf{s} \in \mathbb{R}^{p \times 1}$ and $A \in \mathbb{R}^{p \times p}$ invertible such that:
Problem formulation: ICA as a generative model

- We observe $p$ signals $[x_1, \cdots, x_p] = \mathbf{x} \in \mathbb{R}^{p \times 1}$

Key assumption

There are $p$ independent signals $[s_1, \cdots, s_p] = \mathbf{s} \in \mathbb{R}^{p \times 1}$ and $A \in \mathbb{R}^{p \times p}$ invertible such that:

$$\mathbf{x} = A\mathbf{s}$$
Problem formulation: ICA as a generative model

We observe \( p \) signals \([x_1, \cdots, x_p] = \mathbf{x} \in \mathbb{R}^{p \times 1}\)

Key assumption

There are \( p \) independent signals \([s_1, \cdots, s_p] = \mathbf{s} \in \mathbb{R}^{p \times 1}\) and \( A \in \mathbb{R}^{p \times p} \) invertible such that:

\[
\mathbf{x} = A\mathbf{s}
\]
Problem formulation

\[ x = As \]

Given some realizations of \( x \), we want to recover \( A \) and \( s \).
Is it possible?

Standard indeterminations:

- No hope to recover sources scales

Otherwise, the problem is well-posed [Comon '94].
Is it possible?

Standard indeterminations:
  - No hope to recover sources scales
  - Same for the ordering
Is it possible?

Standard indeterminations:

- No hope to recover sources scales
- Same for the ordering
- Impossible to separate two Gaussian signals (rotation invariant)

Otherwise, the problem is well-posed [Comon ’94].
Is it possible?

Standard indeterminations:

- No hope to recover sources scales
- Same for the ordering
- Impossible to separate two Gaussian signals (rotation invariant)

Otherwise, the problem is well-posed [Comon ’94].
A geometric viewpoint

In 2D ($p = 2$). $n = 2000$ points.

Sources

Mixed observed signals
Density matters

Different densities lead to different patterns

Super-Gaussian  Sub-Gaussian  Gaussian
ICA in the real world
A cute example

ECG of a pregnant mother

Recovered ICA sources

[Zarzoso ’97]
ICA on EEG-MEG data

Selected Channels

EOG
EOG
\(\theta\)
\(\alpha\)
ERP
\(\theta\)
\(\alpha\)
ECG
EMG

Time (s)
ICA on fMRI
A matrix factorization problem
Link with dictionary learning

Given \( n \) samples noted in matrix form \( X \in \mathbb{R}^{p \times n} \)

**ICA**: Find \( A \in \mathbb{R}^{p \times p} \) and \( S \in \mathbb{R}^{p \times n} \) such that \( X = AS \).

- Perfect data fit (\( X \equiv AS \))
- Assumption of statistical independence on \( S \)

**Dictionary learning**: Find \( D \in \mathbb{R}^{p \times k} \) and \( R \in \mathbb{R}^{k \times n} \) such that \( X \approx DR \)

- Approximate data fit (introduces a penalty \( ||X - DR||_F \) in the optimization)
- Assumption of sparsity on \( R \)
Given \( n \) samples noted in matrix form \( X \in \mathbb{R}^{p \times n} \)

**ICA:** Find \( A \in \mathbb{R}^{p \times p} \) and \( S \in \mathbb{R}^{p \times n} \) such that \( X = AS \).

- Perfect data fit \( (X \equiv AS) \)
- Assumption of statistical independence on \( S \)

**Dictionary learning:** Find \( D \in \mathbb{R}^{p \times k} \) and \( R \in \mathbb{R}^{k \times n} \) such that \( X \simeq DR \)

- Approximate data fit (introduces a penalty \( \|X - DR\|_F \) in the optimization)
- Assumption of sparsity on \( R \)
Inference techniques
Maximum likelihood ICA

- $x = As$: generative model.
- Further assumption: fixed density. $s_i \sim d$

Likelihood:

$$p(x|A) = \frac{1}{|\det(A)|} \prod_{i=1}^{p} d([A^{-1}x]_i)$$
Optimization problem

- Work with the unmixing matrix $W = A^{-1}$
- Cost function $\ell(x, W) = -\log(p(x|W^{-1}))$

$$\ell(x, W) = -\log|\det(W)| - \sum_{i=1}^{p} \log(d([Wx]_i))$$
Optimization problem

- Work with the unmixing matrix $W = A^{-1}$
- Cost function $\ell(x, W) = - \log(p(x|W^{-1}))$

\[
\ell(x, W) = - \log|\det(W)| - \sum_{i=1}^{p} \log(d([Wx]_i))
\]

Expected risk:

\[
\mathcal{L}(W) = \mathbb{E}_x[\ell(x, W)] = - \log|\det(W)| - \sum_{i=1}^{p} \mathbb{E}[\log(d([Wx]_i))]
\]
Optimization problem

- Work with the unmixing matrix $W = A^{-1}$
- Cost function $\ell(x, W) = - \log(p(x|W^{-1}))$

$$\ell(x, W) = - \log|\det(W)| - \sum_{i=1}^{p} \log(d([Wx]_i))$$

**Expected risk:**

$$\mathcal{L}(W) = \mathbb{E}_x[\ell(x, W)] = - \log|\det(W)| - \sum_{i=1}^{p} \mathbb{E}[\log(d([Wx]_i))]$$

**Empirical risk.** Given $n$ samples $[x_1, \cdots, x_n] = X \in \mathbb{R}^{p \times n}$:

$$\mathcal{L}_n(W) = \frac{1}{n} \sum_{j=1}^{n} \ell(x_j, W) = - \log|\det(W)| - \frac{1}{n} \sum_{i=1}^{p} \sum_{j=1}^{n} \log(d([WX]_{ij}))$$
Optimization problem

- Work with the unmixing matrix $W = A^{-1}$
- Cost function $\ell(x, W) = -\log(p(x|W^{-1}))$

$$\ell(x, W) = -\log|\det(W)| - \sum_{i=1}^{p} \log(d([Wx]_i))$$

Expected risk:

$$\mathcal{L}(W) = \mathbb{E}_x[\ell(x, W)] = -\log|\det(W)| - \sum_{i=1}^{p} \mathbb{E}[\log(d([Wx]_i))]$$

Empirical risk. Given $n$ samples $[x_1, \cdots, x_n] = X \in \mathbb{R}^{p \times n}$:

$$\mathcal{L}_n(W) = \frac{1}{n} \sum_{j=1}^{n} \ell(x_j, W) = -\log|\det(W)| - \frac{1}{n} \sum_{i=1}^{p} \sum_{j=1}^{n} \log(d([WX]_{ij}))$$
Objective of maximum-likelihood ICA: find

$$W = \arg \min \mathcal{L}(W)$$

If you have a fixed dataset: find

$$W = \arg \min \mathcal{L}_n(W)$$
Objective of maximum-likelihood ICA: find

$$W = \arg \min \mathcal{L}(W)$$

If you have a fixed dataset: find

$$W = \arg \min \mathcal{L}_n(W)$$

This is the problem solved by Infomax [Bell ’95]
Objective of maximum-likelihood ICA: find

\[ W = \arg \min \mathcal{L}(W) \]

If you have a fixed dataset: find

\[ W = \arg \min \mathcal{L}_n(W) \]

This is the problem solved by Infomax [Bell ’95]
Geometry of the problem

\[ \mathcal{L}_n(W) = -\log|\det(W)| - \frac{1}{n} \sum_{i=1}^{p} \sum_{j=1}^{n} \log(d([WX]_{ij})) \]

- No closed form solution. Iterative algorithms
- Optimization on the set of invertible matrices
Geometry of the problem

\[ \mathcal{L}_n(W) = -\log|\det(W)| - \frac{1}{n} \sum_{i=1}^{p} \sum_{j=1}^{n} \log(d([WX]_{ij})) \]

- No closed form solution. Iterative algorithms
- Optimization on the set of invertible matrices
- Invariant by permutation of two rows of \( W \)
Geometry of the problem

\[ \mathcal{L}_n(W) = -\log|\det(W)| - \frac{1}{n} \sum_{i=1}^{p} \sum_{j=1}^{n} \log(d([WX]_{ij})) \]

- No closed form solution. Iterative algorithms
- Optimization on the set of invertible matrices
- Invariant by permutation of two rows of \( W \)
- Non-convex problem
Geometry of the problem

\[ \mathcal{L}_n(W) = -\log|\det(W)| - \frac{1}{n} \sum_{i=1}^p \sum_{j=1}^n \log(d([WX]_{ij})) \]

- No closed form solution. Iterative algorithms
- Optimization on the set of invertible matrices
- Invariant by permutation of two rows of \( W \)
- Non-convex problem
Infomax

Stochastic gradient descent:

\[ W_{t+1} = W_t - \rho \nabla \mathcal{L}_n(W_t) \]

The gradient is computed on a mini-batch of samples.

Issues

- Choosing \( \rho \) is critical and difficult (non-convex problem)
- No safe rule / descent guarantee
- Too small: slow convergence
- Too large: blow-up
- Line-search is hard in a stochastic setting
Infomax

Stochastic gradient descent:

\[ W_{t+1} = W_t - \rho \nabla L_n(W_t) \]

The gradient is computed on a mini-batch of samples.

Issues

- Choosing \( \rho \) is critical and difficult (non-convex problem)
- No safe rule / descent guarantee
- Too small: slow convergence
- Too large: blow-up
- Line-search is hard in a stochastic setting

Advantage: SGD can be much faster than full-batch method, especially for large \( n \).
Infomax

Stochastic gradient descent:

$$W_{t+1} = W_t - \rho \nabla L_n(W_t)$$

The gradient is computed on a mini-batch of samples.

**Issues**

- Choosing $\rho$ is critical and difficult (non-convex problem)
- No safe rule / descent guarantee
- Too small: slow convergence
- Too large: blow-up
- Line-search is hard in a stochastic setting

**Advantage**: SGD can be much faster than full-batch method, especially for large $n$. 

Proposed method

- Stochastic, so fast
- Guaranteed descent at each iteration
- One iteration is as costly as SGD
EM algorithms for ICA
Super-Gaussian densities

- Define $G(y) = - \log(d(y))$.
- $\mathcal{L}_n(W) = - \log|\det(W)| + \frac{1}{n} \sum_{i=1}^{p} \sum_{j=1}^{n} G([WX]_{ij})$

Key assumption: $d$ is super-Gaussian.

$G(\sqrt{\cdot})$ is concave.

- This is the case for most brain sources
Main idea: surrogate functions

$G$ has a quadratic surrogate at each point.

$$G(y) = \min_{u \geq 0} \frac{uy^2}{2} + f(u)$$
Main idea: surrogate functions

$G$ has a quadratic surrogate at each point.

\[ G(y) = \min_{u \geq 0} \frac{uy^2}{2} + f(u) \]

- $f$ is an unimportant function.
- minimum reached for an unique value $u^*(y) = \frac{G''(y)}{y}$. 
Main idea: surrogate functions

$G$ has a quadratic surrogate at each point.

\[ G(y) = \min_{u \geq 0} \frac{uy^2}{2} + f(u) \]

- $f$ is an unimportant function.
- minimum reached for an unique value $u^*(y) = \frac{G''(y)}{y}$. 
Surrogate risk

\[ G(y) = \min_{u \geq 0} \frac{uy^2}{2} + f(u) \]

\[ \ell(x, W) = -\log|\det(W)| + \sum_{i=1}^{p} G([Wx]_i) \]

Introduce dual variables \( u \in \mathbb{R}^{p \times 1} \):

\[ \tilde{\ell}(x, W, u) = -\log|\det(W)| + \frac{1}{2} \sum_{i=1}^{p} u_i [Wx]_i^2 + \sum_{i=1}^{p} f(u_i) \]

- Much simpler dependence in \( W \)!
Surrogate loss

$$G(y) = \min_{u \geq 0} \frac{u y^2}{2} + f(u)$$

$$\mathcal{L}_n(W) = - \log |\det(W)| + \frac{1}{n} \sum_{i=1}^{p} \sum_{j=1}^{n} G([WX]_{i,j})$$

Introduce dual variables $U \in \mathbb{R}^{p \times n}$:

$$\tilde{\mathcal{L}}_n(W, U) = - \log |\det(W)| + \frac{1}{2n} \sum_{i=1}^{p} \sum_{j=1}^{n} U_{ij} [WX]_{i,j}^2 + \frac{1}{n} \sum_{i=1}^{p} \sum_{j=1}^{n} f(U_{i,j})$$
Majorization properties

\[ \mathcal{L}_n(W) = - \log |\det(W)| + \frac{1}{n} \sum_{i=1}^{p} \sum_{j=1}^{n} G([WX]_{ij}) \]

\[ \tilde{\mathcal{L}}_n(W, U) = - \log |\det(W)| + \frac{1}{2n} \sum_{i=1}^{p} \sum_{j=1}^{n} U_{ij} [WX]_{ij}^2 + \frac{1}{n} \sum_{i=1}^{p} \sum_{j=1}^{n} f(U_{ij}) \]

- \( \mathcal{L}_n(W) \leq \tilde{\mathcal{L}}_n(W, U) \), with equality iif \( U = u^*(WX) \)
- \( W \) minimizes \( \mathcal{L}_n \) if and only if \( (W, u^*(WX)) \) minimizes \( \tilde{\mathcal{L}}_n \).
Alternate minimization

Idea:

- For a fixed $U$, minimize $\tilde{\mathcal{L}}_n(W, U)$ w.r.t. $W$
- For a fixed $W$, minimize $\tilde{\mathcal{L}}_n(W, U)$ w.r.t. $U$
Minimization in $W$

$$\tilde{\mathcal{L}}_n(W, U) = -\log|\det(W)| + \frac{1}{2n} \sum_{i=1}^{p} \sum_{j=1}^{n} U_{ij} [WX]^2_{ij} + \cdots$$

Quadratic function in the rows of $W$:

$$\tilde{\mathcal{L}}_n(W, U) = -\log|\det(W)| + \frac{1}{2} \sum_{i=1}^{p} W_i: A^i W_{i:}^\top + \cdots$$
Minimization in $W$

$$\tilde{\mathcal{L}}_n(W, U) = -\log|\det(W)| + \frac{1}{2n} \sum_{i=1}^{p} \sum_{j=1}^{n} U_{ij} [WX]_{ij}^2 + \cdots$$

Quadratic function in the rows of $W$:

$$\tilde{\mathcal{L}}_n(W, U) = -\log|\det(W)| + \frac{1}{2} \sum_{i=1}^{p} W_{i:} A_i W_{i:}^\top + \cdots$$

Sufficient statistics:

$$A_{kl}^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} X_{kj} X_{lj}$$
Minimization in $W$

$$
\tilde{\mathcal{L}}_n(W, U) = -\log|\det(W)| + \frac{1}{2n} \sum_{i=1}^{p} \sum_{j=1}^{n} U_{ij} [WX]_{ij}^2 + \cdots
$$

Quadratic function in the rows of $W$:

$$
\tilde{\mathcal{L}}_n(W, U) = -\log|\det(W)| + \frac{1}{2} \sum_{i=1}^{p} W_i: A^i W_i^\top + \cdots
$$

Sufficient statistics:

$$
A^i_{kl} = \frac{1}{n} \sum_{j=1}^{n} U_{ij} X_{kj} X_{lj}
$$
\[ \tilde{\mathcal{L}}_n(W, U) = -\log|\det(W)| + \frac{1}{2} \sum_{i=1}^{p} W_i: A^i W_i^\top + \cdots \]

Minimization possible w.r.t. a multiplicative update of \( W_i: \)
\[ W \leftarrow MW \] where \( M \) is identity except for its \( i \)-th row which equals \( m \in \mathbb{R}^p \).

W.r.t \( m \), \( \tilde{\mathcal{L}}_n(MW, U) \) is of the form
\[ -\log(|m_i|) + \frac{1}{2} m K m^\top, \]
where \( K = WA^i W^\top \in \mathbb{R}^{p \times p} \).
\[ \tilde{\mathcal{L}}_n(W, U) = -\log|\det(W)| + \frac{1}{2} \sum_{i=1}^{p} W_i: A^i W_i^\top + \cdots \]

Minimization possible w.r.t. a multiplicative update of \( W_i: \)

\[ W \leftarrow MW \text{ where } M \text{ is identity except for its } i\text{-th row which equals } m \in \mathbb{R}^p. \]

W.r.t \( m, \tilde{\mathcal{L}}_n(MW, U) \text{ is of the form } \sum_{i=1}^{p} W_i: A^i W_i^\top \in \mathbb{R}^{p \times p}. \] Minimization in closed form:

\[ m = \frac{K^{-1}}{\sqrt{(K^{-1})_{ii}}} \]
\[ \tilde{\mathcal{L}}_n(W, U) = -\log|\det(W)| + \frac{1}{2} \sum_{i=1}^{p} W_i : A^i W_i^\top + \cdots \]

Minimization possible w.r.t. a *multiplicative* update of \( W_i \):
\[ W \leftarrow MW \text{ where } M \text{ is identity except for its } i\text{-th row which equals } m \in \mathbb{R}^p. \]

W.r.t \( m \), \( \tilde{\mathcal{L}}_n(MW, U) \) is of the form \( -\log(|m_i|) + \frac{1}{2} mKm^\top \),
\[ K = WA^i W^\top \in \mathbb{R}^{p \times p}. \] Minimization in **closed form**: 
\[
m = \frac{K^{-1}_{i:}}{\sqrt{(K^{-1})_{ii}}} \]
We only need the $A^i$'s to minimize in $W$.

\[
A^i_{kl} = \frac{1}{n} \sum_{j=1}^{n} U_{ij} X_{kj} X_{lj}
\]

\[
A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} x_j x_j^T
\]

- Accumulate the $A^i$'s (in a stochastic way)
We only need the $A^i$’s to minimize in $W$.

$$A^i_{kl} = \frac{1}{n} \sum_{j=1}^{n} U_{ij} X_{kj} X_{lj}$$

$$A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} x_j x_j^\top$$

- Accumulate the $A^i$’s (in a stochastic way)
Incremental algorithm

Finite sum setting: \( n \) fixed, minimize \( \tilde{\mathcal{L}}_n \).

\[
A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} x_j x_j^\top
\]

Need a memory \( U^{\text{mem}} \in \mathbb{R}^{p \times n} \)
Incremental algorithm

Finite sum setting: \( n \) fixed, minimize \( \tilde{\mathcal{L}}_n \).

\[
A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} x_j x_j^\top
\]

Need a memory \( U^{\text{mem}} \in \mathbb{R}^{p \times n} \)

- Take a sample \( x_j \) at random
Incremental algorithm

Finite sum setting: \( n \) fixed, minimize \( \tilde{\mathcal{L}}_n \).

\[
A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} x_j x_j^T
\]

Need a memory \( U^{\text{mem}} \in \mathbb{R}^{p \times n} \)

- Take a sample \( x_j \) at random
- Compute \( U^{\text{new}}_{ij} = u^*(W x_j) \)
Incremental algorithm

Finite sum setting: \( n \) fixed, minimize \( \tilde{\mathcal{L}}_n \).

\[
A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} x_j x_j^\top
\]

Need a memory \( U^{\text{mem}} \in \mathbb{R}^{p \times n} \)

- Take a sample \( x_j \) at random
- Compute \( U_{ij}^{\text{new}} = u^* (W x_j) \)
- Update \( A^i \leftarrow A^i + \frac{1}{n} (U_{ij}^{\text{new}} - U_{ij}^{\text{mem}}) x_j x_j^\top \)
Incremental algorithm

Finite sum setting: \( n \) fixed, minimize \( \tilde{L}_n \).

\[
A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} x_j x_j^\top
\]

Need a memory \( U^{\text{mem}} \in \mathbb{R}^{p \times n} \)

- Take a sample \( x_j \) at random
- Compute \( U^{\text{new}}_{:j} = u^* (W x_j) \)
- Update \( A^i \leftarrow A^i + \frac{1}{n} (U^{\text{new}}_{ij} - U^{\text{mem}}_{ij}) x_j x_j^\top \)
- Update the memory: \( U^{\text{mem}}_{:j} = U^{\text{new}}_{:j} \)

Enforces \( A^i = \frac{1}{n} \sum_{j=1}^{n} U^{\text{mem}}_{ij} x_j x_j^\top \) at all time.
Incremental algorithm

Finite sum setting: \( n \) fixed, minimize \( \tilde{\mathcal{L}}_n \).

\[
A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} x_j x_j^\top
\]

Need a memory \( U_{\text{mem}} \in \mathbb{R}^{p \times n} \)

- Take a sample \( x_j \) at random
- Compute \( U_{\text{new}}^{i:j} = u^* (W x_j) \)
- Update \( A^i \leftarrow A^i + \frac{1}{n} (U_{\text{new}}^{i:j} - U_{ij}^{\text{mem}}) x_j x_j^\top \)
- Update the memory: \( U_{\text{mem}}^{i:j} = U_{\text{new}}^{i:j} \)

Enforces \( A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij}^{\text{mem}} x_j x_j^\top \) at all time.
Online algorithm

Streaming setting: you receive samples one at a time. You can only use a sample once. $n$ is not fixed.

$$A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} x_j x_j^\top$$

No more memory
Online algorithm

Streaming setting: you recieve samples one at a time. You can only use a sample once. $n$ is not fixed.

$$A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} x_j x_j^\top$$

No more memory

- Fetch a sample $x$
Online algorithm

Streaming setting: you receive samples one at a time. You can only use a sample once. $n$ is not fixed.

$$A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} x_j x_j^\top$$

No more memory

- Fetch a sample $x$
- Compute $u = u^*(Wx)$
Streaming setting: you receive samples one at a time. You can only use a sample once. \( n \) is not fixed.

\[
A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij}x_jx_j^T
\]

No more memory

- Fetch a sample \( x \)
- Compute \( u = u^*(Wx) \)
- Update \( A^i \leftarrow (1 - \rho(n))A^i + \rho(n)u_i xx^T \)
Online algorithm

Streaming setting: you receive samples one at a time. You can only use a sample once. $n$ is not fixed.

$$A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} x_j x_j^T$$

No more memory

- Fetch a sample $x$
- Compute $u = u^*(Wx)$
- Update $A^i \leftarrow (1 - \rho(n))A^i + \rho(n)u_i x x^T$
- Choose $\rho(n) = \frac{1}{n^\alpha}, \alpha \in [\frac{1}{2}, 1]$
Online algorithm

Streaming setting: you recieve samples one at a time. You can only use a sample once. $n$ is not fixed.

$$A^i = \frac{1}{n} \sum_{j=1}^{n} U_{ij} x_j x_j^\top$$

No more memory

- Fetch a sample $x$
- Compute $u = u^*(Wx)$
- Update $A^i \leftarrow (1 - \rho(n)) A^i + \rho(n) u_i x x^\top$
- Choose $\rho(n) = \frac{1}{n^\alpha}$, $\alpha \in \left[\frac{1}{2}, 1\right]$
So far...

- Stochastic, so fast
So far...

- Stochastic, so fast
- Guaranteed descent at each iteration
So far...

- Stochastic, so fast
- Guaranteed descent at each iteration
- One iteration is as costly as SGD
So far...

- Stochastic, so fast
- Guaranteed descent at each iteration
- One iteration is as costly as SGD
**SGD**: Computing the gradient costs $p^2$ operations /sample

**So far**: Updating one matrix $A^i$ costs $\frac{p(p+1)}{2}$ operations/sample

\[ \rightarrow \frac{p^2(p+1)}{2} \text{ operations/sample} \]
SGD: Computing the gradient costs $p^2$ operations /sample

So far: Updating one matrix $A^i$ costs $\frac{p(p+1)}{2}$ operations/sample

$\rightarrow \frac{p^2(p+1)}{2}$ operations/sample

Idea: only update $q < p$ matrices per sample.
Computation cost

**SGD**: Computing the gradient costs $p^2$ operations /sample

**So far**: Updating one matrix $A^i$ costs $\frac{p(p+1)}{2}$ operations/sample

$\rightarrow \frac{p^2(p+1)}{2}$ operations/sample

Idea: only update $q < p$ matrices per sample.
Diminishing the computation cost

Update $q < p$ matrices $A^i$ per sample.

**Incremental algorithm**

- Compute the dual gap associated with each update:

  $$\text{gap}(W, U_{ij}^{\text{old}}) = \frac{1}{2} U_{ij}^{\text{old}} [WX]_{ij}^2 + f(U_{ij}^{\text{old}}) - G([WX]_{ij})$$
Diminishing the computation cost

Update $q < p$ matrices $A^i$ per sample.

Incremental algorithm

- Compute the *dual gap* associated with each update:

$$\text{gap}(W, U_{ij}^{\text{old}}) = \frac{1}{2} U_{ij}^{\text{old}} \ [WX]_{ij}^2 + f(U_{ij}^{\text{old}}) - G([WX]_{ij})$$

- Measures the decrease of $\tilde{L}_n$ associated with the updating to the $i$-th matrix
Diminishing the computation cost

Update $q < p$ matrices $A^i$ per sample.

**Incremental algorithm**

- Compute the *dual gap* associated with each update:

  $$\text{gap}(W, U_{ij}^{\text{old}}) = \frac{1}{2} U_{ij}^{\text{old}} [WX]_{ij}^2 + f(U_{ij}^{\text{old}}) - G([WX]_{ij})$$

- Measures the decrease of $\tilde{\mathcal{L}}_n$ associated with the updating to the $i$-th matrix

  - Update the $q$ matrix associated with the largest decreases
Diminishing the computation cost

Update $q < p$ matrices $A^i$ per sample.

Incremental algorithm

- Compute the dual gap associated with each update:
  \[
  \text{gap}(W, U_{ij}^{\text{old}}) = \frac{1}{2} U_{ij}^{\text{old}} \left[WX\right]_{ij}^2 + f(U_{ij}^{\text{old}}) - G([WX]_{ij})
  \]

  - Measures the decrease of $\tilde{L}_n$ associated with the updating to the $i$-th matrix
  - Update the $q$ matrix associated with the largest decreases

Online algorithm

- Update $q$ matrices at random
Diminishing the computation cost

Update $q < p$ matrices $A^i$ per sample.

**Incremental algorithm**

- Compute the *dual gap* associated with each update:

  $$\text{gap}(W, U_{ij}^{\text{old}}) = \frac{1}{2} U_{ij}^{\text{old}} [WX]_{ij}^2 + f(U_{ij}^{\text{old}}) - G([WX]_{ij})$$

- Measures the decrease of $\tilde{\mathcal{L}}_n$ associated with the updating to the $i$-th matrix
- Update the $q$ matrix associated with the largest decreases

**Online algorithm**

- Update $q$ matrices at random
All good!

- Stochastic, so fast
- Guaranteed descent at each iteration
- One iteration is as costly as SGD (with $q = 2$)
Results
Convergence measures

- **Loss on left-out data**
  - **Amari distance** Requires that the true mixing matrix $A$ is available. For a matrix $W$, compute $R = WA$ and
  \[
  d = \sum_{i=1}^{p} \left( \sum_{j=1}^{p} \frac{R_{ij}^2}{\max_l R_{il}^2} - 1 \right) + \sum_{i=1}^{p} \left( \sum_{j=1}^{p} \frac{R_{ji}^2}{\max_l R_{ij}^2} - 1 \right).
  \]
  Canceled if $W^{-1}$ and $A$ are equal up to permutation and scale.
Convergence measures

- **Loss on left-out data**
- **Amari distance** Requires that the true mixing matrix $A$ is available. For a matrix $W$, compute $R = WA$ and
  \[ d = \sum_{i=1}^{p} \left( \sum_{j=1}^{p} \frac{R_{ij}^2}{\max_l R_{il}^2} - 1 \right) + \sum_{i=1}^{p} \left( \sum_{j=1}^{p} \frac{R_{ji}^2}{\max_l R_{lj}^2} - 1 \right). \]
  Cancels iif $W^{-1}$ and $A$ are equal up to permutation and scale.
- **Gradient norm**: gradient of $\tilde{\mathcal{L}}_n$. Only meaningful for the finite-sum setting.
Convergence measures

- **Loss on left-out data**

- **Amari distance** Requires that the true mixing matrix $A$ is available. For a matrix $W$, compute $\mathbf{R} = WA$ and
  
  $$d = \sum_{i=1}^{p}(\sum_{j=1}^{p} \frac{R_{ij}^2}{\max_l R_{il}^2} - 1) + \sum_{i=1}^{p}(\sum_{j=1}^{p} \frac{R_{ji}^2}{\max_l R_{lj}^2} - 1).$$

  Cancels iif $W^{-1}$ and $A$ are equal up to permutation and scale.

- **Gradient norm**: gradient of $\tilde{\mathcal{L}}_n$. Only meaningful for the finite-sum setting.
Other algorithms

- **SGD** (i.e. Infomax). Step size $\rho = \frac{\beta}{t^\alpha}$ hand tuned to get the best convergence.

- **Variance reduced methods** (i.e. SAG/ SAGA/ SVRG...). Step size $\rho = \frac{\beta}{t^\alpha}$ hand tuned to get the best convergence.
**Other algorithms**

- **SGD** (i.e. Infomax). Step size $\rho = \frac{\beta}{t^\alpha}$ hand tuned to get the best convergence.

- **Variance reduced methods** (i.e. SAG/ SAGA/ SVRG...). Step size $\rho = \frac{\beta}{t^\alpha}$ hand tuned to get the best convergence.

- **Full batch second order methods** (i.e. Picard !) Works with a line search technique.
Other algorithms

- **SGD** (i.e. Infomax). Step size $\rho = \frac{\beta}{t^\alpha}$ hand tuned to get the best convergence.

- **Variance reduced methods** (i.e. SAG/ SAGA/ SVRG...). Step size $\rho = \frac{\beta}{t^\alpha}$ hand tuned to get the best convergence.

- **Full batch second order methods** (i.e. Picard !) Works with a line search technique.

- **Full batch EM**
Other algorithms

- **SGD** (i.e. Infomax). Step size $\rho = \frac{\beta}{t^\alpha}$ hand tuned to get the best convergence.

- **Variance reduced methods** (i.e. SAG/ SAGA/ SVRG...). Step size $\rho = \frac{\beta}{t^\alpha}$ hand tuned to get the best convergence.

- **Full batch second order methods** (i.e. Picard !) Works with a line search technique.

- **Full batch EM**
Simulated data

$p = 10$, $n = 10^6$ in the finite sum setting, $10^7$ in the online setting. $S \in \mathbb{R}^{p \times n}$ generated with density $d(x) = \frac{1}{2} \exp(-|x|)$. $X = AS$
EEG data

\[ p = 30, \; n = 10^6 \]
Future work

- Find an efficient way to code the algorithm (right now I have to take pretty big mini-batches to be competitive with SGD)
- Find a better policy to choose which matrices $A^i$ to update in the streaming setting

\[ K = W A^i W^\top \in \mathbb{R}^{p \times p}, \quad m = K - 1 \]

Can we make it faster by accumulating the $(A^i - 1)$ instead of the $A^i$?
Future work

- Find an efficient way to code the algorithm (right now I have to take pretty big mini-batches to be competitive with SGD)
- Find a better policy to choose which matrices $A^i$ to update in the streaming setting
- The M-step is costly: compute $K = W A^i W^T \in \mathbb{R}^{p \times p}$, and $m = \frac{K^{-1}_{ii}}{\sqrt{(K^{-1})_{ii}}}$. Can we make it faster by accumulating the $(A^i)^{-1}$ instead of the $A^i$?
Future work

- Find an efficient way to code the algorithm (right now I have to take pretty big mini-batches to be competitive with SGD)
- Find a better policy to choose which matrices $A_i$ to update in the streaming setting
- The M-step is costly: compute $K = WA_i^iW^\top \in \mathbb{R}^{p \times p}$, and $m = \frac{K_{ii}^{-1}}{\sqrt{(K^{-1})_{ii}}}$. Can we make it faster by accumulating the $(A_i^i)^{-1}$ instead of the $A_i^i$?
Thanks for your attention!