

Optimization for (sparse) BSS

Christophe Kervazo / Jérôme Bobin

christophe.kervazo@telecom-paris.fr

jerome.bobin@cea.fr



Recall : Blind Source separation (BSS)

$$\mathbf{x}_i = \sum_{k=1}^n \mathbf{a}_k^* s_{ki}^* + \mathbf{n}_i \quad \longrightarrow \quad \boxed{\mathbf{X} = \mathbf{A}^* \mathbf{S}^* + \mathbf{N}}$$

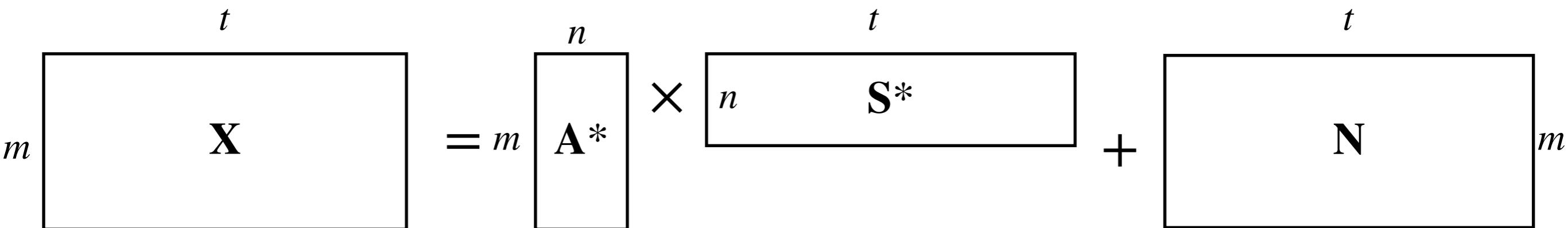
\mathbf{X} : m rows observations and t samples columns ($m \times t$)

\mathbf{A}^* : mixing function ($m \times n$)

\mathbf{S}^* : sources ($n \times t$)

\mathbf{N} : noise and model imperfections ($m \times t$)

Goal : retrieve \mathbf{A}^* and \mathbf{S}^* from the sole knowledge of \mathbf{X}
(or more generally: unmix some signals)



Introducing additional priors

BSS admits an infinite number of solutions

=> How to determine which ones are the true generating factors \mathbf{A}^* and \mathbf{S}^* ?

=> need to introduce additional information, or also additional *priors*, on the sought after factors \mathbf{A}^* , \mathbf{S}^* .

Three main families of priors in BSS :

- Assume the independence of \mathbf{S} (ICA)
- Assume the sparsity of \mathbf{S} (SBSS)
- Use non-negativity (NMF) of \mathbf{A} and \mathbf{S} - **To be done**

=> in addition, it is possible to use deep learning methods

Together, we will see:

- Some **optimization tools** to tackle BSS problems (6h)
(we will mostly focus on *sparse* BSS)
- **Nonnegative matrix factorization** (6h)
(we will in particular put emphasis on *provably robust* algorithms)

Interpretation of sparse BSS (3/3) : statistical view (1/3)

Recovering the sources amounts to back-project the observations on the canonical axes

- **How to do that?**

=> rewrite sparse BSS as a Maximum A Posteriori (MAP) estimation.

$$\arg \max_{\mathbf{A}, \mathbf{S}} P(\mathbf{A}, \mathbf{S} | \mathbf{X})$$

But :

$$P(\mathbf{A}, \mathbf{S} | \mathbf{X}) \propto P(\mathbf{X} | \mathbf{A}, \mathbf{S})P(\mathbf{A}, \mathbf{S}) \quad (\text{Bayes' rule})$$

$$= P(\mathbf{X} | \mathbf{A}, \mathbf{S})P(\mathbf{A})P(\mathbf{S}) \quad (\mathbf{A}, \mathbf{S} \text{ independent})$$

Thus, the MAP writes as

$$\arg \max_{\mathbf{A}, \mathbf{S}} P(\mathbf{X} | \mathbf{A}, \mathbf{S})P(\mathbf{A})P(\mathbf{S})$$

Interpretation of sparse BSS (3/3) : statistical view (2/3)

$$\arg \max_{\mathbf{A}, \mathbf{S}} P(\mathbf{X} | \mathbf{A}, \mathbf{S}) P(\mathbf{A}) P(\mathbf{S})$$

$$= \arg \min_{\mathbf{A}, \mathbf{S}} -\log(P(\mathbf{X} | \mathbf{A}, \mathbf{S})) - \log(P(\mathbf{A})) - \log(P(\mathbf{S})) \quad (\text{rather use } -\log) :$$

Assuming:

- A Gaussian noise: $P(\mathbf{X} | \mathbf{A}, \mathbf{S}) \propto e^{\frac{-\|\mathbf{X} - \mathbf{AS}\|_F^2}{2\sigma^2}}$
- An exponential distribution for $\mathbf{S} = e^{-\beta\|\mathbf{S}\|_1}$
- A uniform distribution for \mathbf{A} ,

we obtain:

$$\arg \min_{\mathbf{A}, \mathbf{S}} \frac{1}{2\sigma^2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \beta \|\mathbf{S}\|_1$$

$$\Rightarrow \arg \min_{\mathbf{A}, \mathbf{S}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1$$

Interpretation of sparse BSS (3/3) : statistical view (3/3)

$$\arg \min_{\mathbf{A}, \mathbf{S}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1$$

Unfortunately, for any \mathbf{A}, \mathbf{S} and $\alpha > 1$,

$$\frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1 > \frac{1}{2} \|\mathbf{X} - \mathbf{A}\alpha\alpha^{-1}\mathbf{S}\|_F^2 + \lambda \|\alpha^{-1}\mathbf{S}\|_1$$

- The solution $\mathbf{S}' = \alpha^{-1}\mathbf{S}$ and $\mathbf{A}' = \alpha\mathbf{A}$ has a lower cost than \mathbf{S}, \mathbf{A}
- The larger α , the better the results
=> this known as *scaling indeterminacy*

More generally, we need to modify the cost function and limit the energy of the columns of \mathbf{A} so that we do not obtain degenerated solutions.

Oblique constraint

We require each column of \mathbf{A} to have a unit energy:

$$l_{\{\forall i \in [1, n]; \|\mathbf{a}_{:,j}\|_{\ell_2}^2 \leq 1\}}(\mathbf{A})$$

Reminder: sparse BSS : cost function

$$\arg \min_{\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1 + \iota_{\{\forall i \in [1, n]; \|\mathbf{a}_{:, i}\|_{\ell_2}^2 \leq 1\}}(\mathbf{A})$$

Data-fidelity **Sparsity** **Oblique constraint**

λ : Regularization parameters

\mathbf{X} : m rows observations and t samples columns ($m \times t$)

\mathbf{A} : mixing function ($m \times n$)

\mathbf{S} : sources ($n \times t$)

\mathbf{N} : noise and model imperfections ($m \times t$)

How to minimize the cost function?

We obtain the following cost-function to minimize to solve the sparse BSS problem

$$\arg \min_{\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1 + \iota_{\{\forall i \in [1, n]; \|\mathbf{a}_{:,i}\|_{\ell_2}^2 \leq 1\}}(\mathbf{A})$$

 **Data-fidelity** **Sparsity** **Oblique constraint**

λ : Regularization parameters

Challenges:

- 1) **Non-smooth** (needs advanced optimization tools: proximal operators)
- 2) **Non-convex** (non-unique minima)

=> **Difficult optimization problem**

Outline

1) Non-smooth optimization for sparse BSS

- a) Reminder on smooth optimization
- b) Non-smooth optimization
- c) Splitting methods

2) Non-convex optimization for sparse BSS

3) Towards learnt optimization for sparse BSS

Optimization problem at hand

$$\arg \min_{\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1 + \iota_{\{\forall i \in [1, n]; \|\mathbf{a}_{:, i}\|_{\ell_2}^2 \leq 1\}}(\mathbf{A})$$

In all this part, we will fix \mathbf{A} . The problem reduces to:

$$\arg \min_{\mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1$$

In addition, we will first work on a single column of \mathbf{X} (and the corresponding column of \mathbf{S}):

$$\arg \min_{\mathbf{s} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{x} - \mathbf{As}\|_F^2 + \lambda \|\mathbf{s}\|_1$$

- Good news: the problem is now **convex** ! (why?)
- It is however still **non-smooth** because of the $\|\cdot\|_1$ term...

Optimization problem at hand

$$\arg \min_{\mathbf{s} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{x} - \mathbf{A}\mathbf{s}\|_F^2 + \lambda \|\mathbf{s}\|_1$$

- More generally, let us consider a problem of the form:

$$\arg \min_{\mathbf{s} \in \mathbb{R}^n} h_{\mathbf{A}}(\mathbf{s}) + \mathcal{G}(\mathbf{s})$$

With the following assumptions:

- The function $\mathbf{s} \rightarrow h_{\mathbf{A}}(\mathbf{s})$ is smooth and has a Lipschitz gradient
 - The $\mathbf{s} \rightarrow \mathcal{G}(\mathbf{s})$ is a closed proper convex function
-
- The function \mathcal{G} is said to be closed proper convex if its epigraph

$$\text{epi } \mathcal{G} = \{(\mathbf{x}, t) \in \mathbb{R}^n \times \mathbb{R} \mid \mathcal{G}(\mathbf{x}) \leq t\}$$

is a non-empty closed convex set

Outline

1) Non-smooth optimization for sparse BSS

- a) Reminder on smooth optimization
- b) Non-smooth optimization
- c) Splitting methods

2) Non-convex optimization for sparse BSS

3) Towards learnt optimization for sparse BSS

Reminder on smooth optimization

If we only have had the simpler (smooth) problem:

$$\arg \min_{\mathbf{s} \in \mathbb{R}^n} h_{\mathbf{A}}(\mathbf{s})$$

- Now easy, we can use a **gradient descent algorithm**:

```
Initialize  $\mathbf{S}^{(0)}$ 
 $k = 0$ 
while not converged do:
     $\mathbf{S}^{(k+1)} = \mathbf{S}^{(k)} - \gamma \nabla h_{\mathbf{A}}(\mathbf{S}^{(k)})$ 
     $k \leftarrow k + 1$ 
end
return  $\mathbf{S}^{(k)}$ 
```

- With $\gamma < \frac{1}{L}$, the Lipschitz constant of $\nabla h_{\mathbf{A}}$

Exercise

- Write the gradient descent algorithm for the BSS data-fidelity term

$$\arg \min_{\mathbf{s} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{x} - \mathbf{As}\|_{\ell_2}^2$$

Exercise

- Write the gradient descent algorithm for the BSS data-fidelity term

$$\arg \min_{\mathbf{s} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{x} - \mathbf{As}\|_{\ell_2}^2$$

Initialize $\mathbf{S}^{(0)}$

$k = 0$

while **not** converged do:

$$\mathbf{s}^{(k+1)} = \mathbf{s}^{(k)} - \frac{1}{\|\mathbf{A}^T \mathbf{A}\|_2} \mathbf{A}^T (\mathbf{As}^{(k)} - \mathbf{x})$$

$k \leftarrow k + 1$

end

return $\mathbf{S}^{(k)}$

Outline

1) Non-smooth optimization for sparse BSS

- a) Reminder on smooth optimization
- b) Non-smooth optimization
- c) Splitting methods

2) Non-convex optimization for sparse BSS

3) Towards learnt optimization for sparse BSS

Non-smooth optimization and proximal operators

Let us now consider the minimization of a non-differentiable cost function:

$$\arg \min_{\mathbf{S} \in \mathbb{R}^{n \times t}} \mathcal{G}(\mathbf{S})$$

(with \mathcal{G} following the above hypotheses)

In this case, a widely used tool is the proximal operator.

$$\text{prox}_{\gamma \mathcal{G}}(\mathbf{s}) = \arg \min_{\mathbf{y} \in \mathbb{R}^n} \mathcal{G}(\mathbf{y}) + \frac{1}{2\gamma} \|\mathbf{s} - \mathbf{y}\|_{\ell_2}^2$$

with $\|\cdot\|_{\ell_2}$ the usual Euclidean norm and $\alpha > 0$.

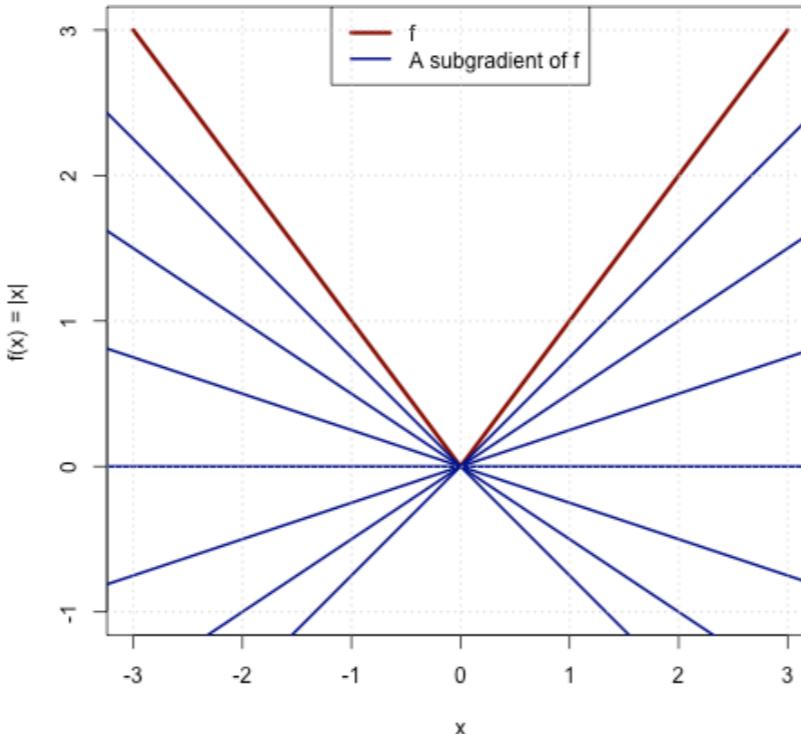
- The function minimized on the right-hand side is strongly convex and not everywhere infinite, so it has a unique minimizer.
- The regularization by the ℓ_2 -norm makes the problem easier to solve

An interpretation of proximal operators

- Let us consider the $\partial\mathcal{G}(\mathbf{s}) \subset \mathbb{R}^n$, the subdifferential of \mathcal{G} at $\mathbf{s} \in \mathbb{R}^n$, defined as:

$$\partial\mathcal{G}(\mathbf{s}) = \{ \mathbf{y} \mid \mathcal{G}(\mathbf{z}) \geq \mathcal{G}(\mathbf{s}) + \mathbf{y}^T(\mathbf{z} - \mathbf{s}) \text{ for all } \mathbf{z} \text{ in the domain of } \mathcal{G} \}$$

=> this is a **set**



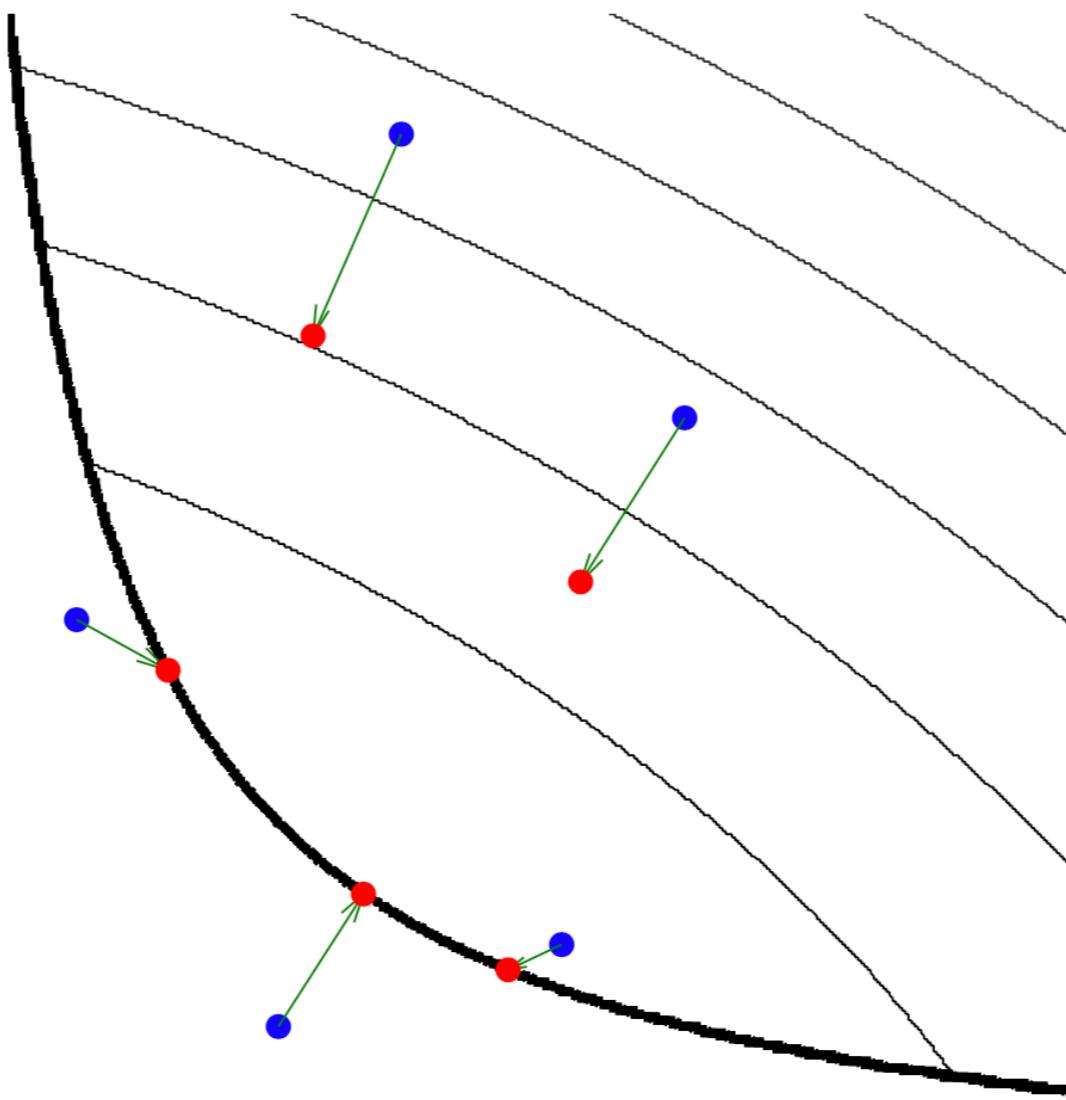
- Specific case: if \mathcal{G} is differentiable in \mathbf{s} , $\partial\mathcal{G}(\mathbf{s}) = \{ \nabla\mathcal{G}(\mathbf{s}) \}$ (singleton)
- The proximal operators and the sub-differential are related as follows

$$\mathbf{prox}_{\lambda\mathcal{G}} = (I + \lambda\partial\mathcal{G})^{-1}$$

N.B.: the equality stems from our assumptions

- The left-hand term is called the resolvent of $\partial\mathcal{G}$

Graphical effect of proximal operators



- $\text{prox}_{\mathcal{G}}(y)$ is a point that compromises between minimizing \mathcal{G} and being close to y (γ weights this trade-off).

Proximal operators : a first property

Separability

If \mathcal{G} is separable across some variables s_1, s_2, \dots, s_k : $\mathcal{G}(s_1, s_2, \dots, s_k) = g_1(s_1) + g_2(s_2) + \dots + g_k(s_k)$, then

$$\text{prox}_{\mathcal{G}(\cdot)}(s_1, s_2, \dots, s_k) = (\text{prox}_{g_1}(s_1), \text{prox}_{g_2}(s_2), \dots, \text{prox}_{g_k}(s_k))$$

=> Said differently, the proximal operator can then be computed component-wise

=> At the basis for parallel computation

Proof : The \mathcal{G} function is separable, as well as the squared ℓ_2 -norm in

$$\text{prox}_{\gamma\mathcal{G}}(\mathbf{s}) = \arg \min_{\mathbf{y} \in \mathbb{R}^n} \mathcal{G}(\mathbf{y}) + \frac{1}{2\gamma} \|\mathbf{s} - \mathbf{y}\|_{\ell_2}^2,$$

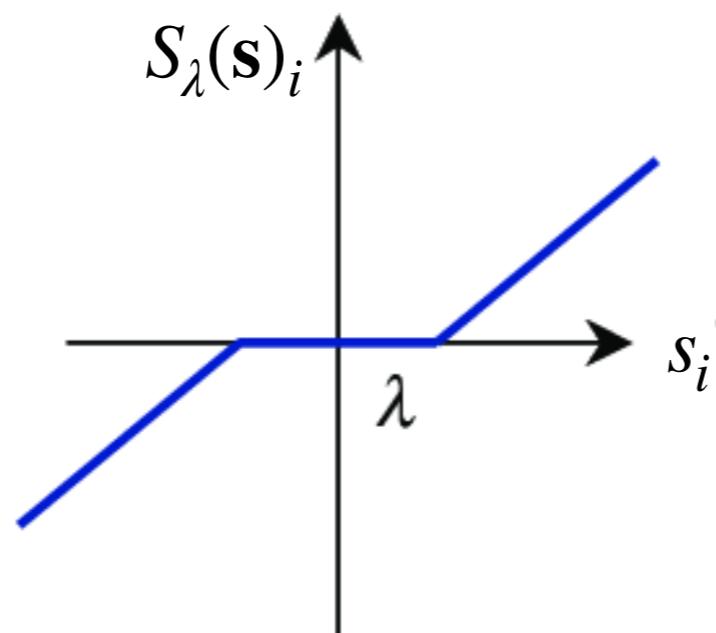
It implies that the problem can be minimized separately over the s_i variables.

Example of proximal operators (1/3)

- If $\mathcal{G}(\cdot)$ is the ℓ_1 norm :

$$\mathcal{G}(\mathbf{s}) = \lambda \|\mathbf{s}\|_1 \text{ for } \mathbf{s} \in \mathbb{R}^n$$

Then: $(\text{prox}_{\lambda \|\cdot\|_1}(\mathbf{s}))_i = \mathcal{S}_\lambda(\mathbf{s})_i = \begin{cases} s_i - \lambda & \text{if } s_i \geq \lambda \\ 0 & \text{if } |s_i| \leq \lambda \\ s_i + \lambda & \text{if } s_i \leq -\lambda \end{cases}$



The proximal operator of the ℓ_1 -norm is the soft-thresholding operator

Example of proximal operators (2/3)

- If $\mathcal{G}(\cdot)$ is the **indicator function** of a closed non-empty convex set C :

$$\iota_C(\mathbf{s}) = \begin{cases} 0 & \text{if } \mathbf{s} \in C \\ +\infty & \text{if } \mathbf{s} \notin C \end{cases}$$

Then:

$$\text{prox}_{\iota_C(\cdot)}(\mathbf{s}) = \Pi_C(\mathbf{s}) = \arg \min_{\mathbf{x} \in C} \|\mathbf{x} - \mathbf{s}\|_2$$

The proximal operator is the Euclidean projection onto C

Example of proximal operators (3/3)

- If $\mathcal{G}(\cdot)$ is the ℓ_1 norm, but the sparsity is enforced into a transformed domain, with Φ an *orthogonal* transform:

$$\mathcal{G}(\mathbf{S}) = \lambda \|\mathbf{S}\Phi\|_1 \text{ for } \mathbf{s} \in \mathbb{R}^n$$

Then:

$$\text{prox}_{\lambda \|\cdot\Phi\|_1}(\mathbf{S}) = S_\lambda(\mathbf{S}\Phi)\Phi^T$$

- If Φ is not orthogonal, the proximal operator is not explicit. This calls for :
 - Either an algorithm to compute it
 - Either to use duality

=> This is beyond the scope of this class

Proximal operators : two key properties

Fixed point property

The point \mathbf{s}^* minimizes $\mathcal{G}(\mathbf{s})$ if and only if

$$\mathbf{prox}_{\mathcal{G}(.)}(\mathbf{s}^*) = \mathbf{s}^*$$

Firm non-expansiveness

Let $x, y \in \mathbb{R}^n$, then

$$\|\mathbf{prox}_{\mathcal{G}}(\mathbf{s}_1) - \mathbf{prox}_{\mathcal{G}}(\mathbf{s}_2)\|_2^2 \leq (\mathbf{s}_1 - \mathbf{s}_2)^T (\mathbf{prox}_{\mathcal{G}}(\mathbf{s}_1) - \mathbf{prox}_{\mathcal{G}}(\mathbf{s}_2))$$

N.B. : stronger than mere non-expansiveness :

$$\|\mathbf{prox}_{\mathcal{G}}(\mathbf{s}_1) - \mathbf{prox}_{\mathcal{G}}(\mathbf{s}_2)\|_2^2 \leq \|\mathbf{s}_1 - \mathbf{s}_2\|_2^2$$

Non-smooth minimization method: the proximal point method

Coming back to

$$\arg \min_{\mathbf{s} \in \mathbb{R}^n} \mathcal{G}(\mathbf{s})$$

How to perform the minimization if we know the proximal operator of \mathcal{G} ?

=> leverage the two above properties

Here comes our first *proximal algorithm*, the **proximal point method**:

```
Initialize  $\mathbf{s}^{(0)}$ 
 $k = 0$ 
while not converged do:
     $\mathbf{s}^{(k+1)} = \text{prox}_{\mathcal{G}}(\mathbf{s}^{(k)})$ 
     $k \leftarrow k + 1$ 
end
return  $\mathbf{s}^{(k)}$ 
```

Outline

1) Non-smooth optimization for sparse BSS

- a) Reminder on smooth optimization
- b) Non-smooth optimization
- c) Splitting methods

2) Non-convex optimization for sparse BSS

3) Towards learnt optimization for sparse BSS

Non-smooth minimization method: summary so far

$$\arg \min_{\mathbf{S} \in \mathbb{R}^{n \times t}} h(\mathbf{A}, \mathbf{S})$$

- $h(\cdot)$ smooth, convex as a function of \mathbf{S}
- Gradient descent algorithm:

```

Initialize  $\mathbf{S}^{(0)}$ 
k = 0
while not converged do:
     $\mathbf{S}^{(k+1)} = \mathbf{S}^{(k+1)} - \gamma \nabla h(\mathbf{S}^{(k)})$ 
    k  $\leftarrow k + 1$ 
end
return  $\mathbf{S}^{(k)}$ 

```

$$\arg \min_{\mathbf{S} \in \mathbb{R}^{n \times t}} \mathcal{G}(\mathbf{S})$$

- $\mathcal{G}(\cdot)$ non-smooth, convex
- Proximal operator:
- Proximal point method:

$$\mathbf{prox}_{\mathcal{G}(\cdot)}(\mathbf{S}) = \arg \min_{\mathbf{Y}} \mathcal{G}(\mathbf{Y}) + \frac{1}{2} \|\mathbf{S} - \mathbf{Y}\|^2$$

```

Initialize  $\mathbf{S}^{(0)}$ 
k = 0
while not converged do:
     $\mathbf{S}^{(k+1)} = \mathbf{prox}_{\mathcal{G}(\cdot)}(\mathbf{S}^{(k)})$ 
    k  $\leftarrow k + 1$ 
end
return  $\mathbf{S}^{(k)}$ 

```

Forward-backward splitting (FBS)

$$\arg \min_{\mathbf{s} \in \mathbb{R}^n} h_A(\mathbf{s}) + \mathcal{G}(\mathbf{s})$$

- $h(\cdot)$ smooth, convex as a function of \mathbf{s} , $\mathcal{G}(\cdot)$ non-smooth, convex (see full hypotheses above)
- Forward-backward splitting method:

Initialize $\mathbf{s}^{(0)}$

$k = 0$

while **not** converged do :

$$\mathbf{s}^{(k+1)} = \text{prox}_{\gamma \mathcal{G}}(\mathbf{s}^{(k)} - \gamma \nabla h(\mathbf{s}^{(k)}))$$

$k \leftarrow k + 1$

end

return $\mathbf{s}^{(k)}$

- The gradient step must γ be chosen such that $\gamma < 1/L$
- Also known as proximal gradient method :
 - Perform a gradient step on the smooth part of the cost function
 - Perform a proximal step on the non-smooth part

FBS as a fixed point iteration

$$\arg \min_{\mathbf{s} \in \mathbb{R}^n} h_A(\mathbf{s}) + \mathcal{G}(\mathbf{s})$$

With the following assumptions:

- The function $\mathbf{s} \rightarrow h_A(\mathbf{s})$ is smooth and has a Lipschitz gradient
- The $\mathbf{s} \rightarrow \mathcal{G}(\mathbf{s})$ is a closed proper convex function

• Theorem

For all $\gamma > 0$, a solution \mathbf{s}^* to the above problem must satisfy:

$$\mathbf{s}^* = \text{prox}_{\gamma \mathcal{G}}(\mathbf{s}^* - \gamma \nabla h_A(\mathbf{s}^*))$$

- Thus, FBS can be seen as a fixed point iteration.

Forward-backward splitting: theoretical results

$$\arg \min_{\mathbf{s} \in \mathbb{R}^n} h_{\mathbf{A}}(\mathbf{s}) + \mathcal{G}(\mathbf{s})$$

- Under the above hypotheses:
 - The $(\mathbf{s}^{(k)})_k$ can be shown to generate a monotonically decreasing $(h_{\mathbf{A}}(\mathbf{s}^{(k)}) + \mathcal{G}(\mathbf{s}^{(k)}))_k$ sequence, which converges to the optimal value, let say C^* , of the criterion.
 - The difference $|h_{\mathbf{A}}(\mathbf{s}^{(k)}) + \mathcal{G}(\mathbf{s}^{(k)}) - C^*|$ is majored by an arbitrary small constant δ in $\mathcal{O}(1/\delta)$ iterations.

FBS: exercise

$$\arg \min_{\mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1$$

- Write the FBS algorithm for this (LASSO) problem
- Forward-backward splitting method = Iterative Shrinkage Thresholding Algorithm (ISTA)

Non-smooth minimization method: example

$$\arg \min_{\mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1$$

- Forward-backward splitting method = Iterative Shrinkage Thresholding Algorithm (ISTA)

Initialize $\mathbf{S}^{(0)}$

$k = 0$

Choose $\gamma = 0.9/\|\mathbf{A}^T \mathbf{A}\|_*$

while **not** converged do:

$$\mathbf{S}^{(k+1)} = \mathcal{S}_{\gamma\lambda} \left(\mathbf{S}^{(k)} - \gamma \mathbf{A}^T (\mathbf{AS}^{(k)} - \mathbf{X}) \right)$$

$k \leftarrow k + 1$

end

return $\mathbf{S}^{(k)}$

Accelerated FBS

$$\arg \min_{\mathbf{s} \in \mathbb{R}^n} h_{\mathbf{A}}(\mathbf{s}) + \mathcal{G}(\mathbf{s})$$

- Fast Iterative Shrinkage Algorithm

Initialize $\mathbf{S}^{(0)}, \mathbf{Z}^{(0)}$ (auxiliary variable)

$k = 1$

Choose $\gamma = 0.9/\|\mathbf{A}^T \mathbf{A}\|_*$

while **not** converged do:

$$\mathbf{Z}^{(k+1)} = \mathbf{S}^{(k)} + \omega^{(k)}(\mathbf{Z}^{(k)} - \mathbf{Z}^{(k-1)})$$

$$\mathbf{S}^{(k+1)} = \text{prox}_{\gamma \mathcal{G}}(\mathbf{Z}^{(k+1)} - \gamma \nabla h_{\mathbf{A}}(\mathbf{Z}^{(k+1)}))$$

$$k \leftarrow k + 1$$

end

return $\mathbf{S}^{(k)}$

- The first update can be seen as an extrapolation term

- $\omega^{(k)}$ can be for instance chosen as $\omega^{(k)} = \frac{k}{k+3}$

Accelerated FBS

- The accelerated FBS converges to the optimal value of the cost function C^* , under the same hypotheses as FBS.
- Accelerated FBS however converges much faster than FBS: it can be shown that the difference $|h_{\mathbf{A}}(\mathbf{s}^{(k)}) + \mathcal{G}(\mathbf{s}^{(k)}) - C^*|$ is majored by an arbitrary small constant δ in $\mathcal{O}\left(1/\sqrt{\delta}\right)$ iterations.
- Nevertheless, contrary to FBS, the $(\mathbf{s}^{(k)})_k$ generated by accelerated FBS does not necessarily generate a monotonically decreasing $(h_{\mathbf{A}}(\mathbf{s}^{(k)}) + \mathcal{G}(\mathbf{s}^{(k)}))_k$ sequence.
- In the case of the LASSO cost function, a well known accelerated FBS algorithm is the *FISTA* algorithm.

Generalized FBS: motivation (1/2)

- So far, we have seen how to solve a problem of the type

$$\arg \min_{\mathbf{s} \in \mathbb{R}^n} h_{\mathbf{A}}(\mathbf{s}) + \mathcal{G}(\mathbf{s})$$

that is, for instance the LASSO / non-blind sparse BSS optimization problem:

$$\arg \min_{\mathbf{s} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1$$

Here, the good news was that the proximal operator of the ℓ_1 -norm was explicit.

- However, $\text{prox}_{\mathcal{G}}$ might not have an explicit form. Consider for instance the following problem, with Φ an orthogonal transform:

$$\arg \min_{\mathbf{s} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1 + \iota_{>0}(\mathbf{S})$$

GFBS: motivation (2/2)

$$\arg \min_{\mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\Phi\|_1 + \iota_{>0}(\mathbf{S})$$

- In this case though, both $\|\mathbf{S}\Phi\|_1$ and $\iota_{>0}(\mathbf{S})$ have explicit proximal operators

=> More generally, it often happens that we have constraints such that

$$\mathcal{G}(\mathbf{s}) = \sum_{i=1}^L \mathcal{G}_i(\mathbf{s})$$

with each of the \mathcal{G}_i having an (simple) explicit proximal operator, but not \mathcal{G} .

- The GFBS enables to still perform easily the minimization of

$$\arg \min_{\mathbf{s} \in \mathbb{R}^n} h_{\mathbf{A}}(\mathbf{s}) + \sum_{i=1}^L \mathcal{G}_i(\mathbf{s})$$

GFBS

- Generalized Forward-backward splitting method:

Initialize $\mathbf{s}^{(0)}$, $\mathbf{z}_{(l)}$ for $l \in \{1..L\}$

$k = 0$

while **not** converged do:

for $l = 1..L$ do:

$$\mathbf{z}_{(l)} = \mathbf{z}_{(l)} + \mu \left(\text{prox}_{\frac{\gamma}{\omega_l} \mathcal{G}_l} \left(2\mathbf{s}^{(k)} - \mathbf{z}_{(l)} - \gamma \nabla h_{\mathbf{A}}(\mathbf{s}^{(k)}) \right) - \mathbf{s}^{(k)} \right)$$

endfor

$$\mathbf{s}^{(k+1)} = \sum_{l=1}^L \omega_l \mathbf{z}_{(l)}$$

$$k \leftarrow k + 1$$

endwhile

return $\mathbf{s}^{(k)}$

$$\boxed{\arg \min_{\mathbf{s} \in \mathbb{R}^n} h_{\mathbf{A}}(\mathbf{s}) + \sum_{i=1}^L \mathcal{G}_i(\mathbf{s})}$$

With: $\forall l \in [1,L]$, $\omega_l \in (0,1)$ and $\sum_{l=1}^L \omega_l = 1$,

$$\gamma \in (0,2L) \text{ and } \mu \in \left(0, \min \left(1.5, 0.5 + \frac{1}{L} \right) \right)$$

Outline

1) Non-smooth optimization for sparse BSS

- a) Reminder on smooth optimization
- b) Non-smooth optimization
- c) Splitting methods

2) Non-convex optimization for sparse BSS

- a) Exact algorithms
- b) Heuristics

3) Towards learnt optimization for sparse BSS

How to minimize the cost function?

$$\boxed{\arg \min_{\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1 + \iota_{\{\forall i \in [1, n]; \|\mathbf{a}_{:,j}\|_{\ell_2}^2 \leq 1\}}(\mathbf{A})}$$

- So far, we have tackled the problem over only a single variable (w.l.o.g., \mathbf{S})
- How to deal with both \mathbf{A} and \mathbf{S} ?

=> Unfortunately, the problem becomes non-convex. In particular, it can have several (local) minima.

=> Generally speaking, there is no perfect solution for solving general non-convex optimization problems

What can we say about the non-convexity?

$$\arg \min_{\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1 + \iota_{\{\forall i \in [1, n]; \|\mathbf{a}_{:, i}\|_{\ell_2}^2 \leq 1\}}(\mathbf{A})$$

- Although the problem is non-convex, we can still obtain a few results since our cost function is particular. It is of the form

$$\arg \min_{\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{S} \in \mathbb{R}^{n \times t}} h(\mathbf{A}, \mathbf{S}) + \mathcal{J}(\mathbf{A}) + \mathcal{G}(\mathbf{S})$$

with (there might be specific - weaker - conditions for the different algorithms we will see - we use general ones, which fits our BSS context - but you can look at the Kurdyka–Łojasiewicz property if interested)

- $h(\mathbf{A}, \mathbf{S})$ multi-convex (convex in \mathbf{A} and \mathbf{S} separately) and differentiable. We assume that $h_{\mathbf{A}}$ and $h_{\mathbf{B}}$ are strongly convex and that their gradients are Lipschitz continuous.
 - \mathcal{J} and \mathcal{G} are extended-valued closed proper functions
-
- Under the above hypotheses, some algorithms exist which ensure the convergence to a critical point of the cost function (i.e. local min. / max. or saddle point). The proofs are however quite involved, and will not be studied here.
 - The different algorithms might converge to different critical points due to the non-convexity.

Outline

1) Non-smooth optimization for sparse BSS

- a) Reminder on smooth optimization
- b) Non-smooth optimization
- c) Splitting methods

2) Non-convex optimization for sparse BSS

- a) Exact algorithms
- b) Heuristics

3) Towards learnt optimization for sparse BSS

Block-coordinate descent (BCD)

$$\arg \min_{\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{S} \in \mathbb{R}^{n \times t}} h(\mathbf{A}, \mathbf{S}) + \mathcal{J}(\mathbf{A}) + \mathcal{G}(\mathbf{S})$$

- BCD was the first studied algorithm
- Main idea: alternate between the two variables, for which the problem is convex, and solve exactly each subproblem with one of the convex algorithms we saw
- Block-coordinate descent (BCD)

Initialize $\mathbf{S}^{(0)}, \mathbf{A}^{(0)}$

$k = 0$

while **not** converged do :

$$\mathbf{S}^{(k+1)} = \arg \min_{\mathbf{S} \in \mathbb{R}^{n \times t}} h(\mathbf{A}^{(k)}, \mathbf{S}) + \mathcal{G}(\mathbf{S})$$

$$\mathbf{A}^{(k+1)} = \arg \min_{\mathbf{A} \in \mathbb{R}^{m \times n}} h(\mathbf{A}, \mathbf{S}^{(k+1)}) + \mathcal{J}(\mathbf{A})$$

$k \leftarrow k + 1$

end

return $\mathbf{S}^{(k)}, \mathbf{A}^{(k)}$

- It is for instance possible to use an accelerated FBS to solve each subproblem

Proximal Alternating Linear Minimization (PALM)

$$\arg \min_{\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{S} \in \mathbb{R}^{n \times t}} h(\mathbf{A}, \mathbf{S}) + \mathcal{J}(\mathbf{A}) + \mathcal{G}(\mathbf{S})$$

- However, it might be slower to exactly minimize each sub-problem at each iterations
- The PALM algorithm alternates proximal-gradient steps over \mathbf{A} and \mathbf{S} variables
- Proximal Alternating Linear Minimization (PALM)

Initialize $\mathbf{S}^{(0)}, \mathbf{A}^{(0)}$

$k = 0$

while **not** converged do:

$$\mathbf{S}^{(k+1)} = \text{prox}_{\gamma \mathcal{G}(\cdot)}(\mathbf{S}^{(k)} - \gamma \nabla h(\mathbf{A}^{(k)}, \mathbf{S}^{(k)}))$$

$$\mathbf{A}^{(k+1)} = \text{prox}_{\eta \mathcal{J}(\cdot)}(\mathbf{A}^{(k)} - \eta \nabla h(\mathbf{A}^{(k)}, \mathbf{S}^{(k+1)}))$$

$$k \leftarrow k + 1$$

end

return $\mathbf{S}^{(k)}, \mathbf{A}^{(k)}$

PALM : application to sparse BSS

$$\arg \min_{\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1 + \iota_{\{\forall i \in [1, n]; \|\mathbf{a}_{:, i}\|_{\ell_2}^2 \leq 1\}}(\mathbf{A})$$

- Proximal Alternating Linear Minimization (PALM)

PALM : application to sparse BSS

$$\arg \min_{\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1 + \iota_{\{\forall i \in [1, n]; \|\mathbf{a}_{:,j}\|_{\ell_2}^2 \leq 1\}}(\mathbf{A})$$

- Proximal Alternating Linear Minimization (PALM)

Initialize $\mathbf{S}^{(0)}, \mathbf{A}^{(0)}$

$k = 0$

while **not** converged do:

$$\mathbf{S}^{(k+1)} = S_{\gamma\lambda} \left(\mathbf{S}^{(k)} + \frac{0.9}{\|\mathbf{A}^{(k)T} \mathbf{A}^{(k)}\|_*} \mathbf{A}^{(k)T} (\mathbf{X} - \mathbf{A}^{(k)} \mathbf{S}^{(k)}) \right)$$

$$\mathbf{A}^{(k+1)} = \Pi_{\|\cdot\|_2 \leq 1} \left(\mathbf{A}^{(k)} + \frac{0.9}{\|\mathbf{S}^{(k+1)} \mathbf{S}^{(k+1)T}\|_*} (\mathbf{X} - \mathbf{A}^{(k)} \mathbf{S}^{(k+1)}) \mathbf{S}^{(k+1)T} \right)$$

$k \leftarrow k + 1$

end

return $\mathbf{S}^{(k)}, \mathbf{A}^{(k)}$

Outline

1) Non-smooth optimization for sparse BSS

- a) Reminder on smooth optimization
- b) Non-smooth optimization
- c) Splitting methods

2) Non-convex optimization for sparse BSS

- a) Exact algorithms
- b) Heuristics

3) Towards learnt optimization for sparse BSS

Exact algorithms

- From now on, we will focus on the case of sparse BSS:

$$\arg \min_{\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1 + \iota_{\{\forall i \in [1, n]; \|\mathbf{a}_{:, i}\|_{\ell_2}^2 \leq 1\}}(\mathbf{A})$$

- The algorithms that we saw so far are « exact » algorithms (guaranteed to converge to a critical point of the cost function).
However, it does not mean that they recover the ground-truth factors

$$\mathbf{X} = \mathbf{A}^* \mathbf{S}^* + \mathbf{N}$$

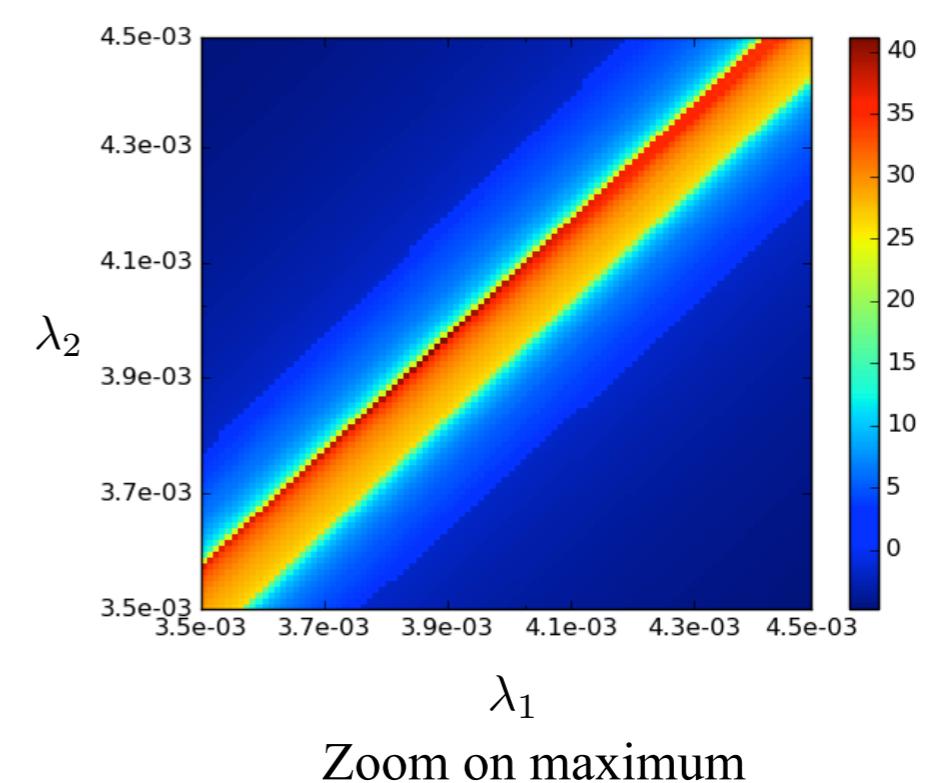
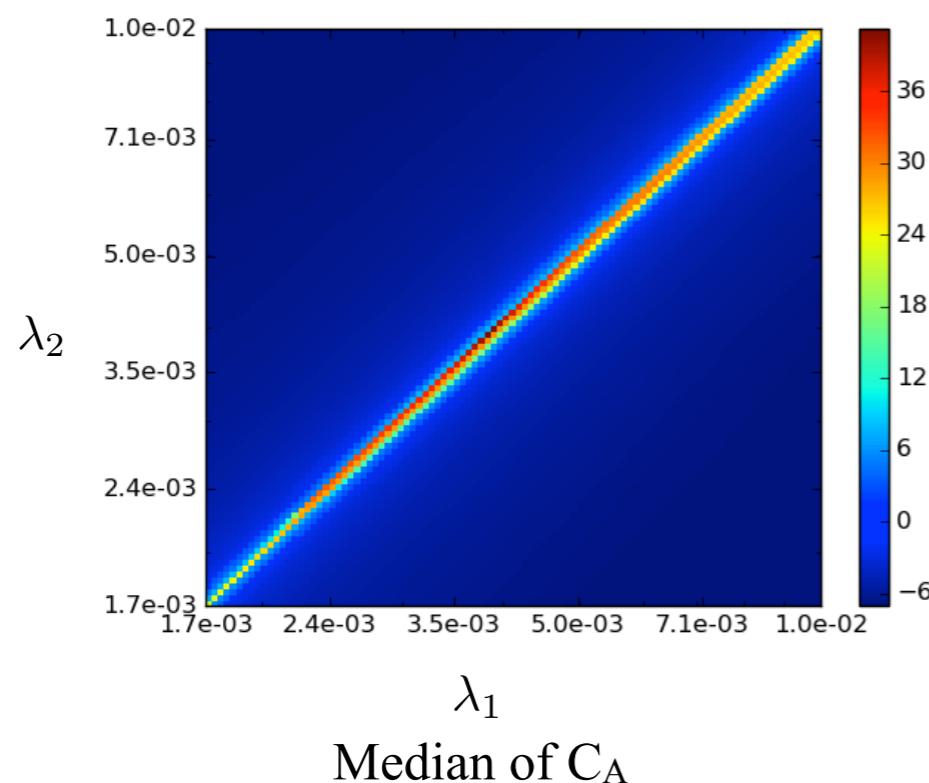
- Several reasons for that:
 - Using the $\|\cdot\|_1$ norm sparsity constraint introduces a bias in the algorithm estimates
 - The algorithms are not guaranteed to converge to the global minimum of the cost function (impact of the initialization)
 - The global minimum of the cost function does not necessarily correspond to the true \mathbf{A}^* and \mathbf{S}^*
- => in particular, the solution depends on the regularization parameters λ values

Exact algorithms: impact of the regularization parameter choice

Empirical study: study sensitivity to regularization parameters

Let us first assume one parameter per source: $\Lambda = \begin{bmatrix} \lambda_1 & \lambda_1 \dots \lambda_1 \\ \lambda_2 & \lambda_2 \dots \lambda_2 \\ \dots \\ \lambda_n & \lambda_n \dots \lambda_n \end{bmatrix}$

- Simulated $n = 2$ sources (dynamic range = 0.6)
- Compute $C_A = -10 \log(\text{mean}(|\mathbf{P}\hat{\mathbf{A}}^\dagger \mathbf{A}^* - \mathbf{I}_d|)))$, \mathbf{P} : indeterminacy correction



- 3×10^{-4} change \Rightarrow 30 dB drop
- In diagonal, a 2×10^{-3} change \Rightarrow 7 dB drop

GMCA: 36 dB

Generalized Morphological Component Analysis (GMCA)

- GMCA can be seen as an heuristic strategy enabling to approximately minimize

$$\arg \min_{\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \|\Lambda \odot \mathbf{S}\|_1 + \iota_{\{\forall i \in [1, n]; \|\mathbf{a}_{:, i}\|_{\ell_2}^2 \leq 1\}}(\mathbf{A})$$

- It has several practical interest:
 - It is empirically robust to the initialization (low sensitivity to spurious critical points)
 - It comes with an automatic regularization parameter Λ choice (the Λ matrix is assumed to be diagonal)
- However, although in practice GMCA often stabilizes, it cannot be proved to converge (and sometimes diverges)
- Note that several extensions of GMCA have been proposed, to tackle more general problems (nonnegativity constraints, partial correlations, large-scale problems...)

Generalized Morphological Component Analysis (GMCA)

- Generalized Morphological Component Analysis (GMCA)

Initialize $\mathbf{A}^{(0)}$

$k = 0$

while **not** *stabilized* do:

$$\mathbf{S}^{(k+1/2)} = \mathbf{A}^{(k)\dagger} \mathbf{X}$$

for i = 1..n

$$\lambda_i = 3 \text{ mad} \left(\mathbf{s}_i^{(k+1/2)} \right)$$

$$\mathbf{s}_i^{(k+1)} = S_{\lambda_i} \left(\mathbf{s}_i^{(k+1/2)} \right)$$

endfor

$$\mathbf{A}^{(k+1)} = \Pi_{\|\cdot\|_2 \leq 1} \left(\mathbf{X} \mathbf{S}^{(k+1)\dagger} \right)$$

$$k \leftarrow k + 1$$

endwhile

return $\mathbf{S}^{(k)}, \mathbf{A}^{(k)}$

Thus, $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$

- where \dagger stands for the pseudo-inverse.

GMCA automatic hyper-parameter choice

$$\underset{\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{S} \in \mathbb{R}^{n \times t}}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \|\mathbf{RS} \odot (\mathbf{S}\Phi_{\mathbf{S}}^T)\|_1 + \iota_{\{\forall i \in [1, n]; \|\mathbf{A}^i\|_{\ell_2}^2 = 1\}}(\mathbf{A})$$

with $\mathbf{R}_{\mathbf{S}} = \begin{bmatrix} \lambda_1 & \lambda_1 & \dots & \lambda_1 \\ \lambda_2 & \lambda_2 & \dots & \lambda_2 \\ \dots & \dots & \dots & \dots \\ \lambda_n & \lambda_n & \dots & \lambda_n \end{bmatrix}$

- In GMCA, use of **MAD of the current source estimation** (new value at each iteration):

$$\lambda_i = \kappa \text{MAD}[\hat{\mathbf{A}}^\dagger \mathbf{X}]_i \quad \text{with } \text{MAD}(s) \simeq 1.48 \operatorname{median}(|s - \operatorname{median}(s)|)$$
- MAD: robust STD estimator. Insensible to *sparse* perturbations: $\text{MAD}(\mathbf{S}^*)$ small
- Such a choice is motivated by a fixed point argument: assume that at a given iteration, GMCA has perfectly estimated \mathbf{A}^* , then the source \mathbf{S} update is given by:

$$\begin{aligned} \mathbf{S} \leftarrow \mathcal{S}_{\mathbf{R}_{\mathbf{S}}}(\mathbf{A}^{*\dagger} \mathbf{X}) &= \mathcal{S}_{\mathbf{R}_{\mathbf{S}}}(\mathbf{A}^{*\dagger} \mathbf{A}^* \mathbf{S}^* + \mathbf{A}^{*\dagger} \mathbf{N}) \\ &= \mathcal{S}_{\mathbf{R}_{\mathbf{S}}}(\mathbf{S}^* + \mathbf{A}^{*\dagger} \mathbf{N}) \end{aligned}$$

- Therefore, to make $\mathbf{S} \leftarrow \mathbf{S}^*$, the soft-thresholding should remove the Gaussian noise $\mathbf{A}^{*\dagger} \mathbf{N}$

\Rightarrow by thresholding with $\lambda_i = 3 \times \text{std}[\mathbf{A}^{*\dagger} \mathbf{N}]_i$, 99% of the noise is removed

\Rightarrow here, $\text{std}[\mathbf{A}^{*\dagger} \mathbf{N}]_i$ is estimated using $\text{MAD}(\hat{\mathbf{A}}^\dagger \mathbf{X})$

Interpretation of GMCA

- GMCA as an approximation of the BCD algorithm
 - In the BCD algorithm, each subproblem over \mathbf{A} and \mathbf{S} is minimized exactly
 - In contrast, GMCA uses projected least squares, which can be considered as an approximate solution of the subproblem
 - For instance, for $\arg \min_{\mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{AS}\|_F^2 + \lambda \|\mathbf{S}\|_1$, the update

$$\mathbf{S} = S_\lambda(\mathbf{A}^\dagger \mathbf{X})$$

corresponds to first minimize the data-fidelity term and then apply the proximal operator

- GMCA as a PALM for specific problems
 - If you consider $\mathbf{A}^{(k)}$ to be orthogonal and choose $\gamma = 1$ in PALM iterations,
- You obtain

$$\mathbf{S}^{(k+1)} = S_\lambda(\mathbf{A}^T \mathbf{X}) = S_\lambda(\mathbf{A}^\dagger \mathbf{X})$$

which corresponds to GMCA update

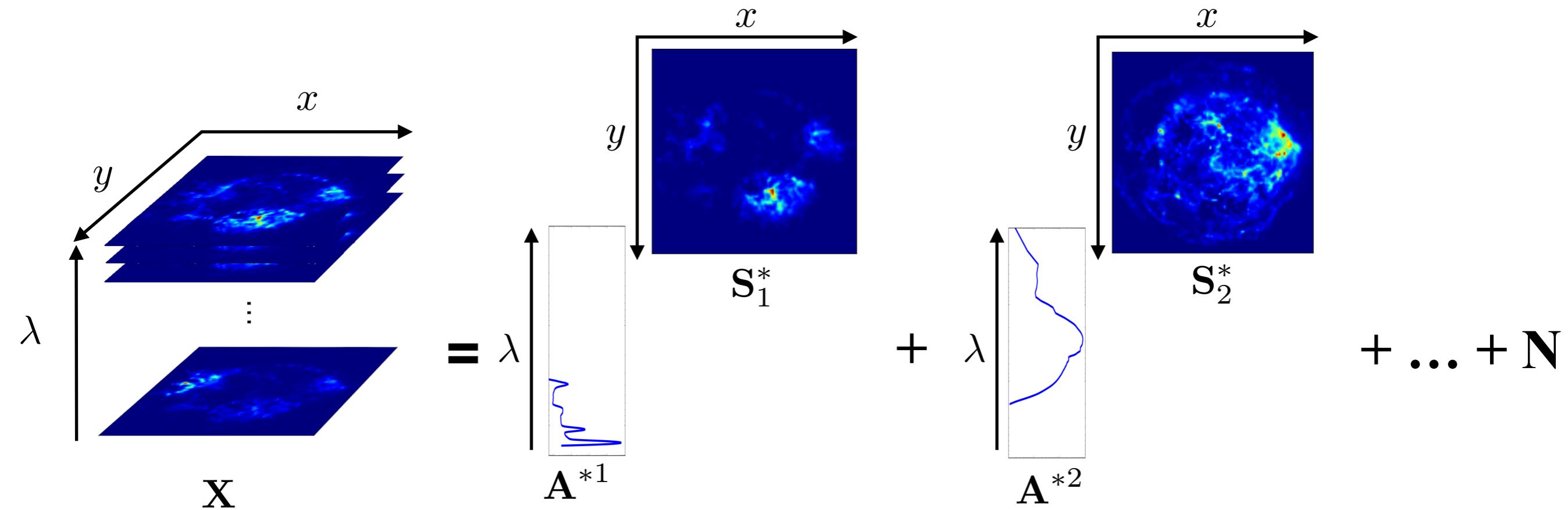
Two-step strategies

	BCD	GMCA	PALM
Reference	[Tseng01]	[Bobin07]	[Bolte14]
Fast	✗	✓✓	✓
Automatic parameter choice	✗	✓	✗
Robust to spurious min.	✗	✓	✗
Convergence	✓	✗	✓
Finds a critical point ?	✓	✗	✓

Easy to use
]
 Mathematically sound

- It is possible to use GMCA as a warm-up stage for PALM:
 - Gives a good starting point to PALM
 - Enable to derive good regularization parameters for PALM from the first guess

Exemple of application: Chandra dataset



$$\mathbf{X} = \mathbf{A}^* \mathbf{S}^* + \mathbf{N}$$

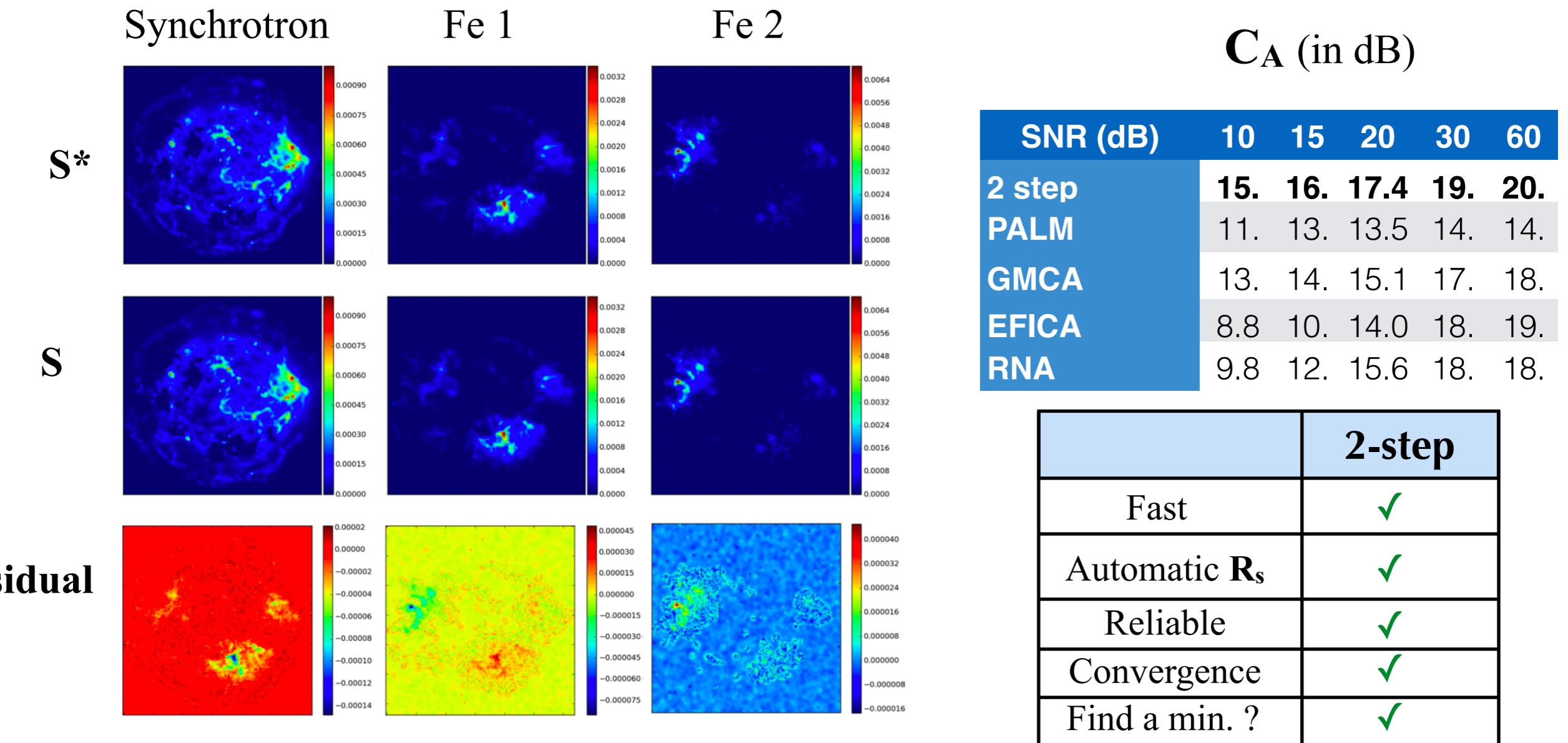
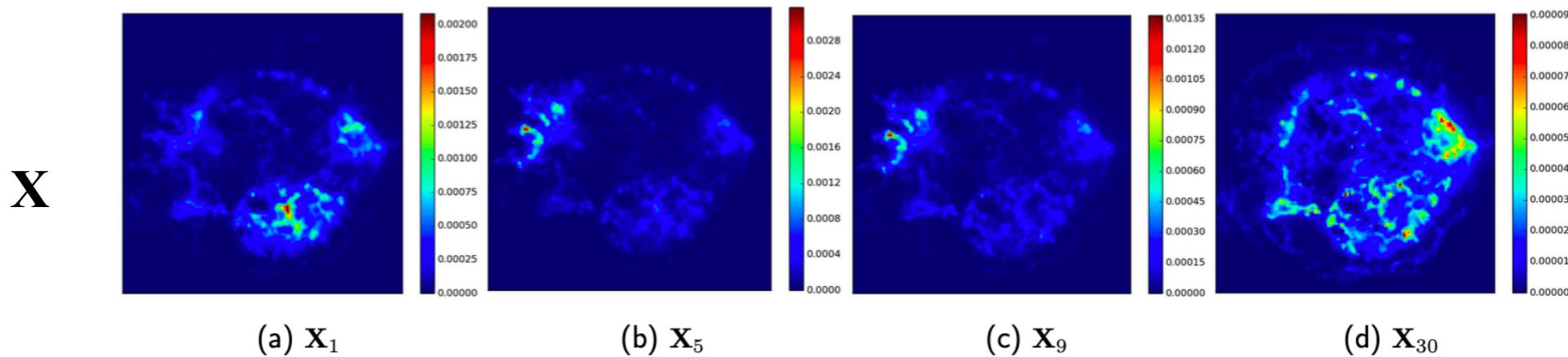
\mathbf{X} ($m = 12 \times n = 128^2$) : m rows observations, each corresponding to an image of the scene captured in a given spectral band and t columns, corresponding to the number of pixels in the original image

\mathbf{A}^* ($m \times n = 4$) : each column is the spectrum of a specific element

\mathbf{S}^* ($n \times t$) : abundance (= concentration) of each element in each pixel

\mathbf{N} ($m \times t$) : noise and model imperfections

Exemple of application : Chandra data



Outline

1) Non-smooth optimization for sparse BSS

- a) Reminder on smooth optimization
- b) Non-smooth optimization
- c) Splitting methods

2) Non-convex optimization for sparse BSS

- a) Exact algorithms
- b) Heuristics

3) Towards learnt optimization for sparse BSS

Motivation

- Exact algorithms (such as PALM) can obtain **high accuracy separations**
- On the other hand, they are really **difficult to tune**
- Usually they are tuned by cross-validation :
 - Either on the reconstruction error $\|\mathbf{X} - \mathbf{AS}\|_F$ (why is it a bad idea ?)
 - Either, when ground truth exists, on the estimation error $\|\mathbf{S}^* - \hat{\mathbf{S}}\|_F$ (with \mathbf{P} a permutation matrix)

- On the other hand, let us assume that you have examples of the type of factors \mathbf{A}^* and \mathbf{S}^* , or merely several data sets of the same type that the one you want to predict \mathbf{X} :

Why not use machine learning to infer the best parameters from the data you have ?

- Here, we will consider **algorithm unrolling (AU)**.

Motivation

- Here, we will consider **algorithm unrolling (AU)**.

- In brief, **algorithm unrolling**

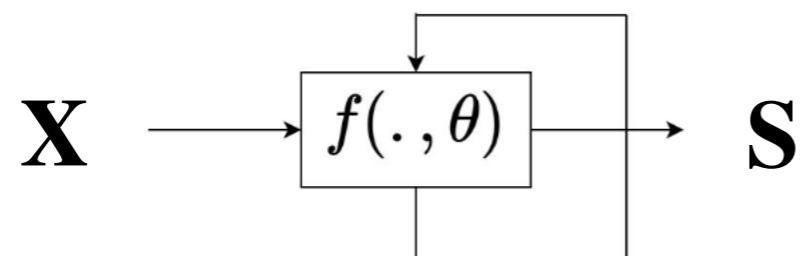
=> is a principled methodology to propose new neural network architectures for solving inverse problems, based on classical iterative algorithms

=> enables to **bypass the cumbersome hyper-parameter choice**

=> yields **interpretable neural networks**

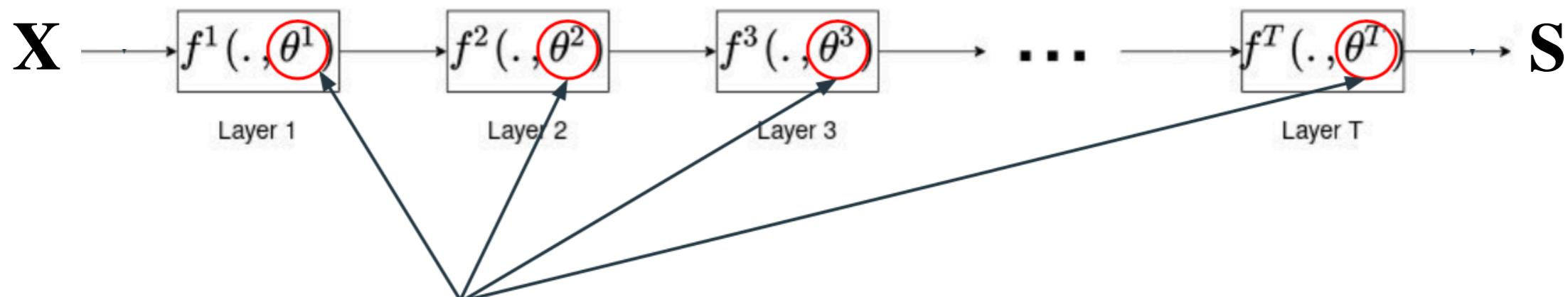
Algorithm unrolling: methodology

- The iterative algorithms write in the form of a loop refining at each step the solution:



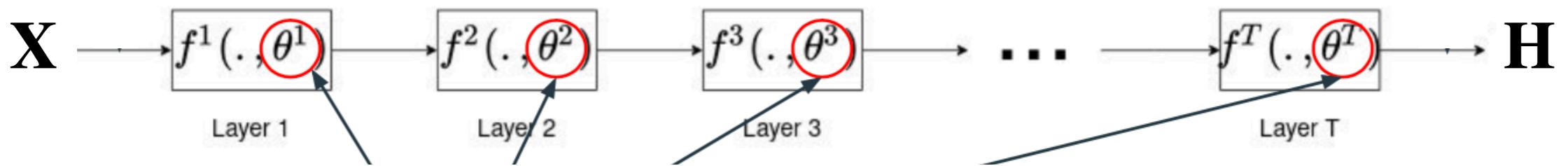
with θ the algorithm parameters
(regularization, gradient step sizes...)

- Algorithm unrolling rewrites this scheme in the form of a neural network:



- The algorithms parameters becomes *trainable* on a *training set* (i.e. they becomes the weights of the neural network)
- The number of iterations T is usually much smaller than in the original algorithm

Algorithm unfolding vs classical optimization



Classical iterative algorithm

Unrolled algorithm

1 iteration

=

1 layer

Hand-set parameters

Learned parameters = neural network weights

Requires tenth, hundreds iterations (or even more)

10-25 layers is often enough

Data free (general)

Requires a training data base (problem-specific)

General unsupervised cost function

Mostly supervised loss on a training set

Algorithm unrolling, first example on NBSS (1/2)

- Let us consider non-blind source separation (NBSS), in which we only want to estimate \mathbf{S}

$$\arg \min_{\mathbf{S} \in \mathbb{R}^{n \times t}} \frac{1}{2} \|\mathbf{X} - \mathbf{A}^* \mathbf{S}\|_F^2 + \lambda \|\mathbf{S}\|_1$$

- The corresponding ISTA update is:

$$\mathbf{S}^{(k+1)} = \mathcal{S}_{\gamma\lambda} \left(\mathbf{S}^{(k)} - \gamma \mathbf{A}^{*T} (\mathbf{A}^* \mathbf{S}^{(k)} - \mathbf{X}) \right)$$

- Let us further assume that :

- We know \mathbf{A}^* (NBSS)
- That we have several training datasets ${}^1\mathbf{X}, {}^2\mathbf{X}, \dots, {}^{t_{train}}\mathbf{X}$ such that

$${}^1\mathbf{X} = \mathbf{A}^* {}^1\mathbf{S}^* + \mathbf{N}$$

$${}^2\mathbf{X} = \mathbf{A}^* {}^2\mathbf{S}^* + \mathbf{N}$$

...

$${}^{t_{train}}\mathbf{X} = \mathbf{A}^* {}^{t_{train}}\mathbf{S}^* + \mathbf{N}$$

- That for each training dataset ${}^i\mathbf{X}$, we have access to the corresponding source ${}^i\mathbf{S}^*$
- In a second phase, we will be willing to predict, from a new testing dataset ${}^{test}\mathbf{X}$, the sources ${}^{test}\mathbf{X}$ where

$${}^{test}\mathbf{X} = \mathbf{A}^* {}^{test}\mathbf{S}^* + \mathbf{N}$$

Algorithm unrolling, first example on NBSS (2/2)

- Now, we can infer the λ and γ ISTA hyperparameters from the training set by minimizing a loss between ISTA estimates ${}^i\hat{\mathbf{S}}$ and the ground truth ${}^i\mathbf{S}^*$, for $i = 1 \dots t_{train}$:

$$\underline{\lambda}^*, \underline{\gamma}^* = \arg \min_{\lambda, \gamma} \sum_{i=1}^{t_{train}} \| {}^i\hat{\mathbf{S}} - {}^i\mathbf{S}^* \|_F^2$$

- $\Theta = \{\lambda^*, \gamma^*\}$ are then learnt by back-propagation in the ISTA algorithm.

- At test time, the new « optimized » algorithm is given by

Initialize $\mathbf{S}^{(0)}$, choose $\gamma = 0.9/\|\mathbf{A}^T \mathbf{A}\|_*$

$k = 0$

while **not** converged do:

$$\mathbf{S}^{(k+1)} = \mathcal{S}_{\underline{\gamma}^* \underline{\lambda}^*} \left(\mathbf{S}^{(k)} - \underline{\gamma}^* \mathbf{A}^T (\mathbf{A} \mathbf{S}^{(k)} - \mathbf{X}) \right)$$

$k \leftarrow k + 1$

end

return $\mathbf{S}^{(k)}$

Learned ISTA (LISTA)

- Beyond merely inferring the λ and γ , LISTA further proposes to re-parametrize the ISTA update:

$$\mathbf{S}^{(k+1)} = \mathcal{S}_{\gamma\lambda} \left(\mathbf{S}^{(k)} - \gamma \mathbf{A}^{*T} (\mathbf{A}^* \mathbf{S}^{(k)} - \mathbf{X}) \right) \quad \text{ISTA}$$
$$\mathbf{S}^{(k+1)} = \mathcal{S}_{\underline{\theta}} \left(\underline{\mathbf{W}}_1 \mathbf{S}^{(k)} + \underline{\mathbf{W}}_2 \mathbf{X} \right) \quad \text{LISTA}$$

- But rather than using the explicit forms of $\underline{\mathbf{W}}_1$ and $\underline{\mathbf{W}}_2$, all the $\Theta = \{\underline{\theta}, \underline{\mathbf{W}}_1, \underline{\mathbf{W}}_2\}$ parameters are learnt.
- In principle,

$$\underline{\theta} = \lambda\gamma, \underline{\mathbf{W}}_1 = I_d - \gamma \mathbf{A}^{*T} \mathbf{A}^*, \underline{\mathbf{W}}_2 = \gamma \mathbf{A}^{*T} \mathbf{A}^*$$

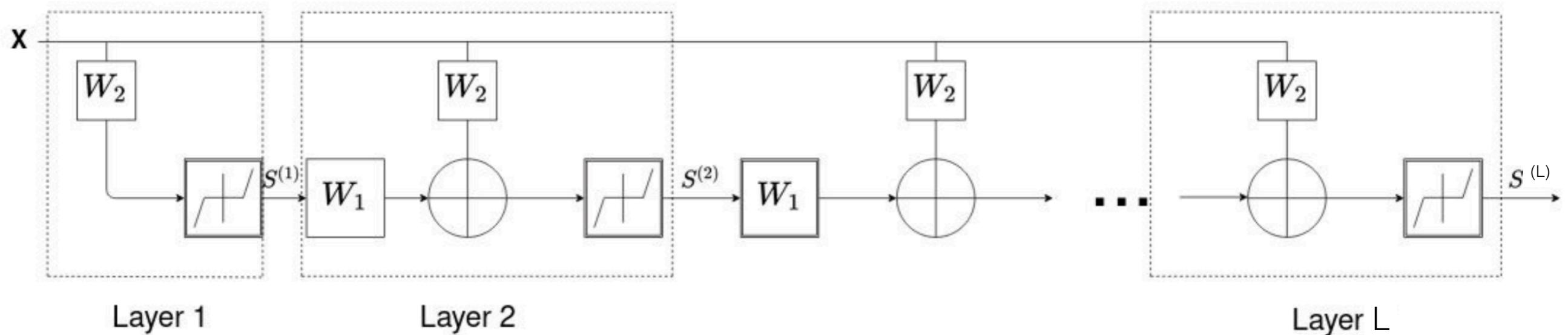
=> but this is not exploited *at all*, and LISTA learns these parameters without using any potential shape they should have.

=> learning more parameters than θ and λ enables the algorithm to have **more flexibility** to adapt to the training set.

LISTA as a neural network

$$\mathbf{S}^{(k+1)} = \mathcal{S}_\theta \left(\underline{\mathbf{W}}_1 \mathbf{S}^{(k)} + \underline{\mathbf{W}}_2 \mathbf{X} \right) \quad \text{LISTA}$$

- LISTA updates typically have the shape of a neural network (NN).



- Such a neural network is trained end-to-end using the training set.
- In addition, it is further possible to untie the parameters and make them layer-dependent (non-shared parameters):

$$\mathbf{W}_1^{(1)}, \mathbf{W}_1^{(2)}, \dots, \mathbf{W}_1^{(L)}$$

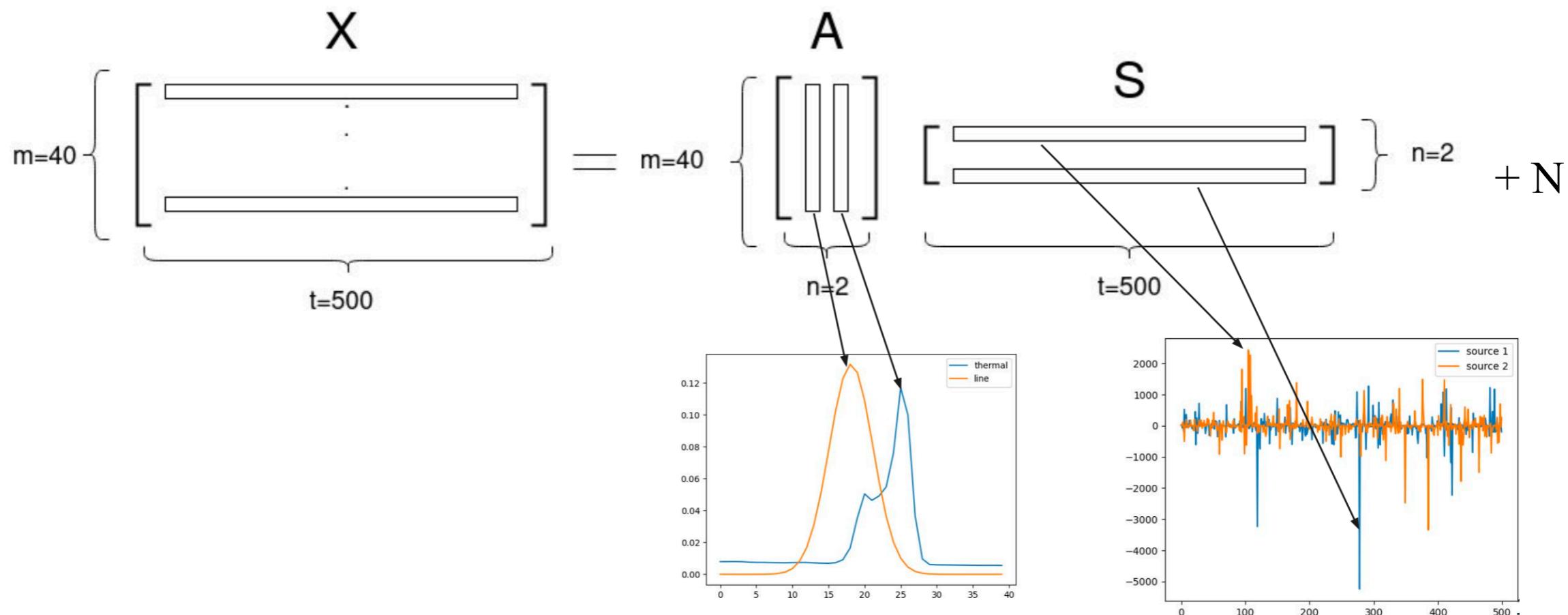
$$\mathbf{W}_2^{(1)}, \mathbf{W}_2^{(2)}, \dots, \mathbf{W}_2^{(L)}$$

$$\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(L)}$$

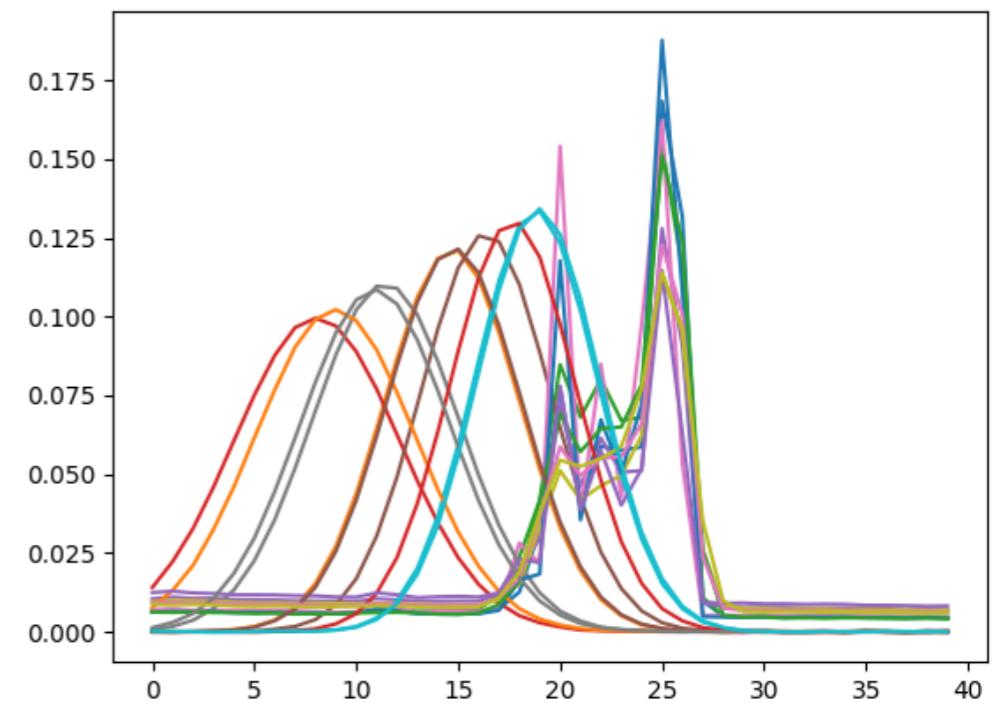
LISTA limitations

- LISTA was originally proposed in the sparse coding context. However, in sparse BSS:
 - The \mathbf{A}^* operator is not fixed for the whole data base (both in training and test sets)
 - Even worse, \mathbf{A}^* is unknown
- Beyond the above specific to NBSS issue, LISTA does not use at all the explicit form of the ISTA re-parametrization.
- As such, it might be enhanced by better using the theoretical expressions.

Numerical illustration of LISTA : dataset

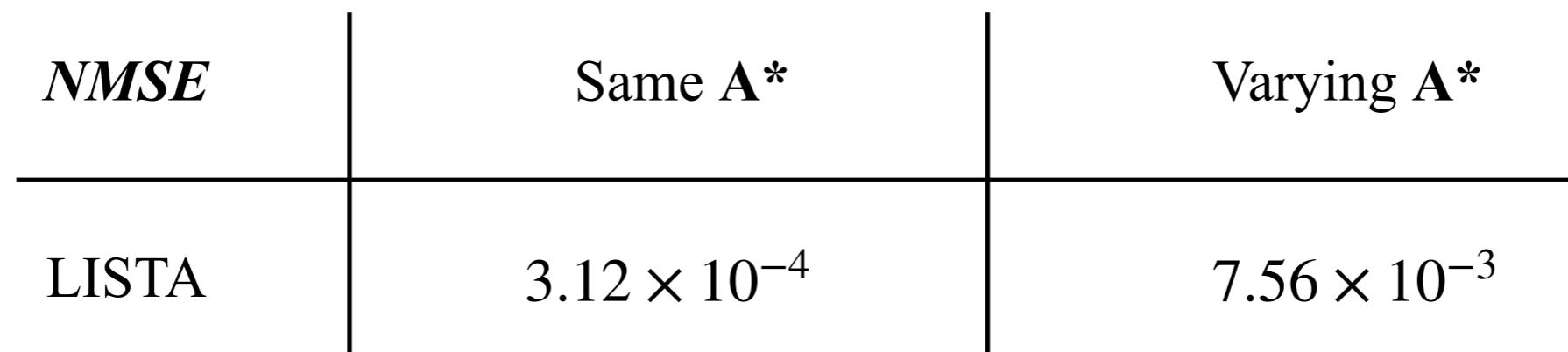


- Study case in astrophysics, with $n = 2$ spectra:
 - Thermal
 - Line (redshift)
- So far, the sources are simulated as generalized Gaussians of parameter $\beta = 0.3$
- The noise is assumed to be Gaussian
- There are $p = 10000$ samples for the data bases (split in 80% - 20%)



Numerical illustration of LISTA: non-fixed A* issue

- A LISTA is implemented to solve the NBSS problem on our data set
- We assessed how much the variations of \mathbf{A}^* matters by trying to estimate \mathbf{S} in 2 experimental settings:
 - Fix \mathbf{A}^* (use a single \mathbf{A}^* for both the training and testing data sets)
 - Use varying \mathbf{A}^* in both sets



=> it means that even with our gentle variations of \mathbf{A}^* , the results of LISTA are 24 times worse !

What is the issue of using LISTA for source separation?

- In a classical ISTA update, the information about \mathbf{A}^* is fully used to update \mathbf{S}

$$\mathbf{S}^{(k+1)} = \mathcal{S}_{\frac{\lambda}{L}} \left(\mathbf{S}^{(k)} - \frac{1}{L} \mathbf{A}^T (\mathbf{A}\mathbf{S} - \mathbf{X}) \right)$$

- In contrast, in LISTA

$$\mathbf{S}^{(k+1)} = \mathcal{S}_{\underline{\theta}} \left(\underline{\mathbf{W}}_1 \mathbf{S}^{(k)} + \underline{\mathbf{W}}_2 \mathbf{X} \right)$$

=> \mathbf{A}^* is not directly used, it only appears implicitly in

$$\mathbf{W}_1 = \mathbf{I} - \frac{1}{L} \mathbf{A}^T \mathbf{A} \quad \mathbf{W}_2 = \frac{1}{L} \mathbf{A}^T$$

- However, \mathbf{W}_1 and \mathbf{W}_2 are the same for the *whole* training set.
- Therefore, they correspond to a kind of *marginalization* of \mathbf{A}^* on the whole training set, deteriorating the results by not taking into account the variability

LISTA-CP

- Therefore, it seems important to have an update which better takes into account the variability of \mathbf{A}^*
- [chen18] have used the coupling of

$$\mathbf{W}_1 = \mathbf{I} - \frac{1}{L} \mathbf{A}^T \mathbf{A} \quad \text{and} \quad \mathbf{W}_2 = \frac{1}{L} \mathbf{A}^T$$

to propose the LISTA-CP algorithm, having as update:

$$\mathbf{S}^{(k+1)} = \mathcal{S}_{\underline{\theta}^{(k)}} \left(\mathbf{S}^{(k)} - \underline{\mathbf{W}}^T (\mathbf{A}^* \mathbf{S}^{(k)} - \mathbf{X}) \right)$$

- In contrast to LISTA, LISTA-CP do not use only a marginalization on the whole training set of \mathbf{A}

<i>NMSE</i>	Same \mathbf{A}	Varying \mathbf{A}
LISTA	3.12×10^{-4}	7.56×10^{-3}
LISTA-CP	3.12×10^{-4}	5.18×10^{-4}

- There is a huge gain in using LISTA-CP for non-blind source separation !

Unrolling : summary so far

- So far, we have seen the principle of algorithm unrolling: leverage a dataset to learn (a part) of an update in a classical optimization algorithm such as ISTA
- LISTA, which is the first historical unrolled algorithm, nevertheless have limitations in the context of NBSS
- We saw that using LISTA-CP enables to reduce the issue of variabilities over \mathbf{A}^* in the dataset.
- However, in blind source separation, \mathbf{A}^* is unknown => how to deal with that ?

Solving the BLIND source separation problem

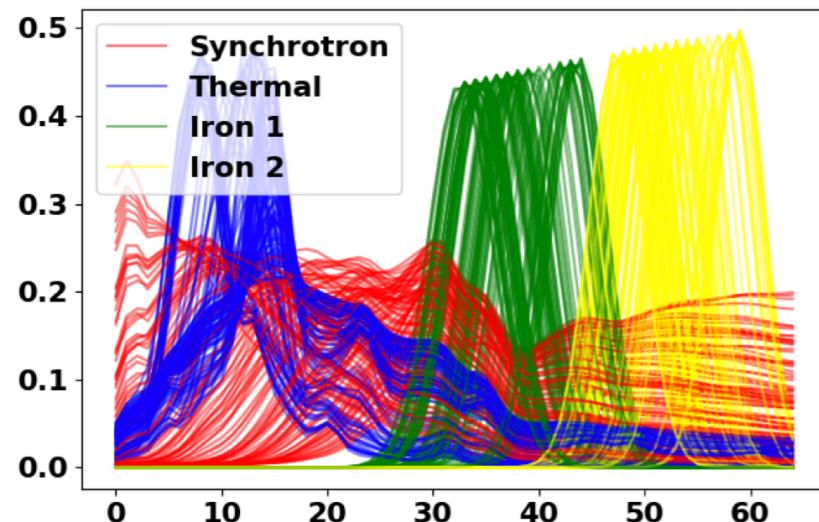
- So far, we knew either one of the factors
- In reality, both are unknown
- We can unroll PALM to give the **L-PALM** algorithm
- The **S** update has already been considered, and we can use a LISTA-CP update
- For **A**, there are much less variables to predict, so it might not be as useful to use a complicated update

- This yields the L-PALM updates:

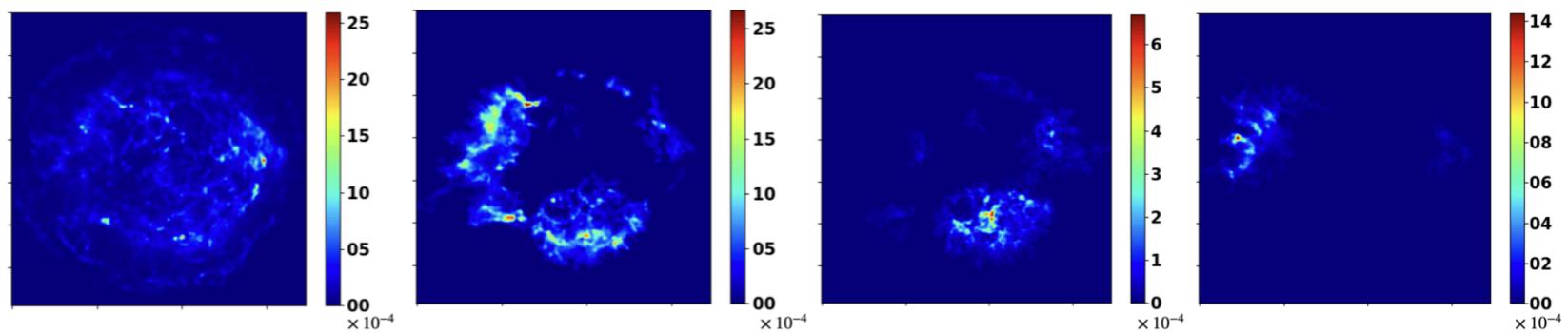
$$\mathbf{S}^{(k+1)} = \mathcal{S}_{\underline{\theta}} \left(\mathbf{S}^{(k)} - \underline{\mathbf{W}}^T (\mathbf{A}^{(k)} \mathbf{S}^{(k)} - \mathbf{X}) \right) \quad (\text{LISTA-CP update})$$

$$\mathbf{A}^{(k+1)} = \Pi_{\|\cdot\| \leq 1} \left(\mathbf{A}^{(k)} + \frac{1}{\underline{L}_A} (\mathbf{X} - \mathbf{A}^{(k)} \mathbf{S}^{(k+1)}) \mathbf{S}^{(k+1)T} \right) \quad (\text{learning step-size})$$

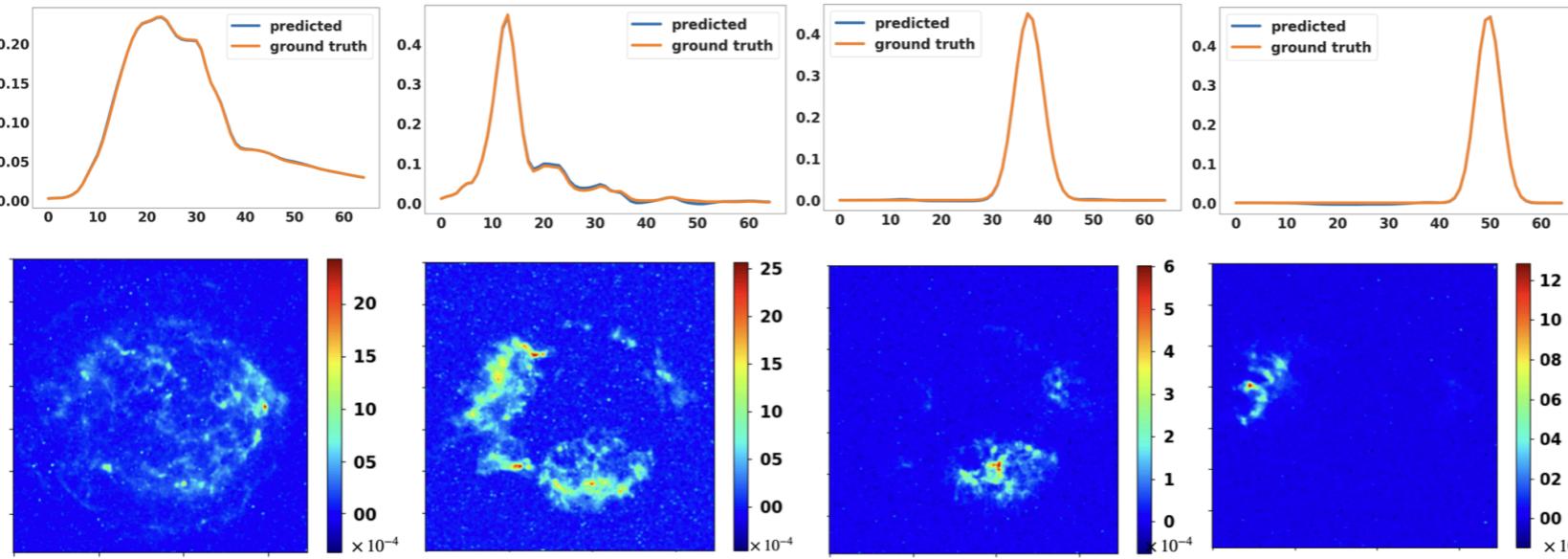
LPALM in practice



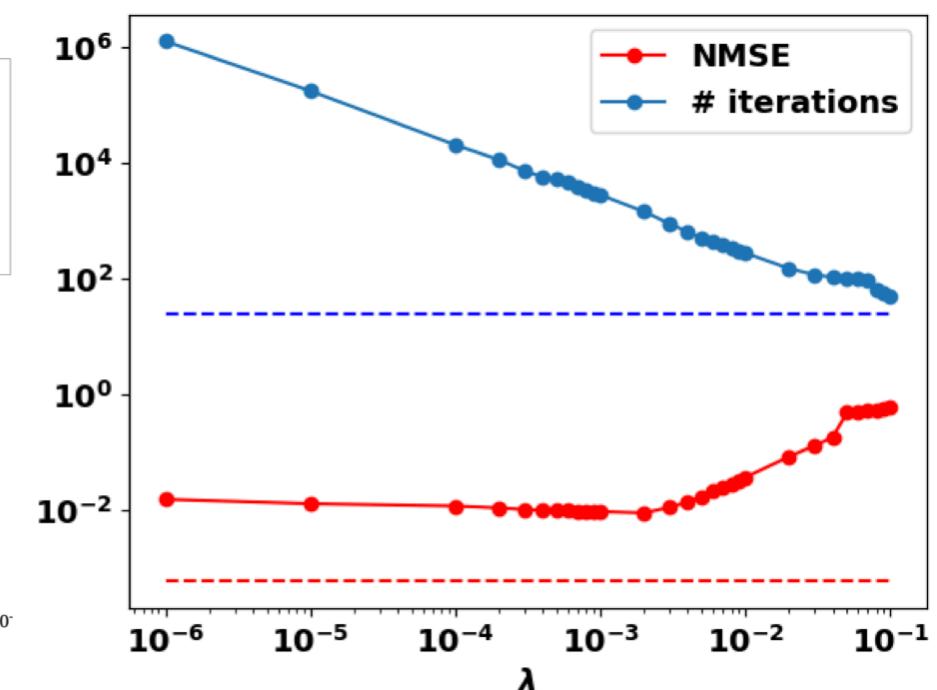
Datasets A*
(750 training, 150 testing)



Dataset S*
(1 for testing, trained on Generalized Gaussians)



Qualitative results



Quantitative results

Conclusion on algorithm unrolling

- Algorithm unrolling can enable :
 - A huge acceleration of conventional algorithms
 - Large amelioration in the separation quality
- But this comes at a cost :
 - Requires a training set
 - Once the unrolled algorithm is learnt, it is specific to a given data type
- There exists other way to do unrolling (such as learning the norm used - see the SNMF net, known in hyperspectral unmixing)
- Beyond unrolling, other methods exists to leverage learning for optimization (such as Plug-and-Play).

A few references

- Basic convex optimization:

BOYD, Stephen, BOYD, Stephen P., et VANDENBERGHE, Lieven. *Convex optimization*. Cambridge university press, 2004.

- Proximal algorithms:

Parikh, N., & Boyd, S. (2014). Proximal algorithms. *Foundations and Trends in optimization*, 1(3), 127-239.

Raguet, H., Fadili, J., & Peyré, G. (2013). A generalized forward-backward splitting. *SIAM Journal on Imaging Sciences*, 6(3), 1199-1226.

- Multi-convex optimization:

Bolte, J., Sabach, S., & Teboulle, M. (2014). Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *Mathematical Programming*, 146(1), 459-494.

Xu, Y., & Yin, W. (2013). A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on imaging sciences*, 6(3), 1758-1789.

- Heuristics in sparse BSS:

Bobin, J., Starck, J. L., Fadili, J., & Moudden, Y. (2007). Sparsity and morphological diversity in blind source separation. *IEEE Transactions on Image Processing*, 16(11), 2662-2674.

Kervazo, C., Bobin, J., Chenot, C., & Sureau, F. (2020). Use of PALM for ℓ_1 sparse matrix factorization: Difficulty and rationalization of a two-step approach. *Digital Signal Processing*, 97, 102611.

- Algorithm unrolling:

Gregor, K., & LeCun, Y. (2010, June). Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning* (pp. 399-406).

Chen, X., Liu, J., Wang, Z., & Yin, W. (2018). Theoretical linear convergence of unfolded ISTA and its practical weights and thresholds. *Advances in Neural Information Processing Systems*, 31.

Fahes, M., Kervazo, C., Bobin, J., & Tupin, F. (2021). unrolling palm for sparse semi-blind source separation. *Accepted at ICLR 2022*.