Optimization for (sparse) BSS

Christophe Kervazo / Jérôme Bobin
christophe.kervazo@telecom-paris.fr
jerome.bobin@cea.fr

Practical aspects

Project:

- The project subject is on Github!
- You are expected to do it by groups of two
- Defense on the 22/03
- A 3 page report is expected

Practical works:

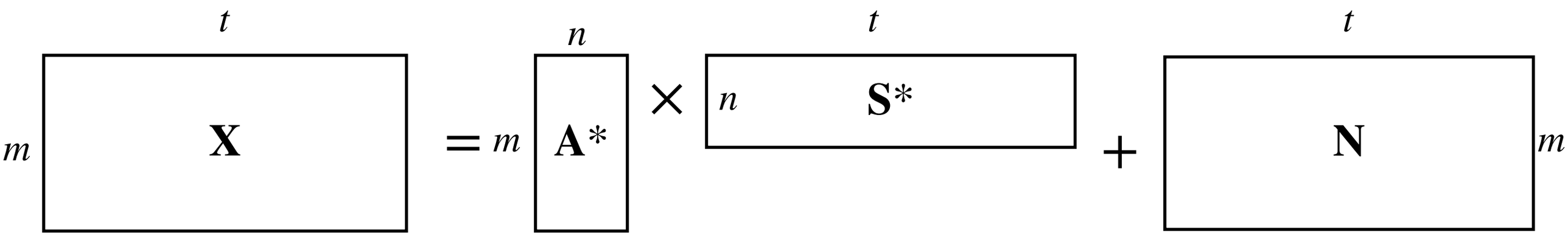
- If possible, please install pytorch on your python distribution
- Otherwise, use google colab (but the data set takes a long time to be uploaded)

Recall : Blind Source separation (BSS)

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

X : m rows observations and t samples columns (m x t)
A* : mixing function (m x n)
S* : sources (n x t)
N : noise and model imperfections (m x t)

Goal : retrieve A*and S*from the sole knowledge of 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 **A*** and **S*** ?

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

Three main families of priors in BSS :

- ➢ Assume the independence of **S** (ICA)
- ➢ Assume the sparsity of **S** (SBSS)
- ➢ Use non-negativity (NMF) of **A** and **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)

Reminder: sparse BSS : cost function

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

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

λ : Regularization parameters

$\mathbf{\Phi_S^T}$: is a sparsifying transform

X : m rows observations and t samples columns (m x t)

A : mixing function (m x n)

S : sources (n x t)

N : noise and model imperfections (m x t)

How to minimize the cost function?

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

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

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

λ : Regularization parameters

$\mathbf{\Phi}_{\mathbf{S}}^{\mathbf{T}}$: is a sparsifying transform

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

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

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

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

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

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

- Good news: the problem is now convex ! (why?)
- It is however still non-smooth because of the $\|.\|_1$ 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 $\mathbf{s} \to h_{\mathbf{A}}(\mathbf{s})$ is smooth and has a Lipschitz gradient
- The $\mathbf{s} \to \mathcal{G}(\mathbf{s})$ is a closed proper convex function

- The function $\mathcal{G}$ is said to be closed proper convex if its epigraph

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

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 **<u>gradient descent algorithm</u>**:

```
Initialize S^(0)
k = 0
while not converged do:
    S^(k+1) = S^(k) − γ ∇h_A(S^(k))
    k ← k + 1
end
return 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

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

Exercise

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

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

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

$k = 0$

while **not** converged do:

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

$k \leftarrow k+1$

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.

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

with $\|.\|_{\ell_2}$ 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.

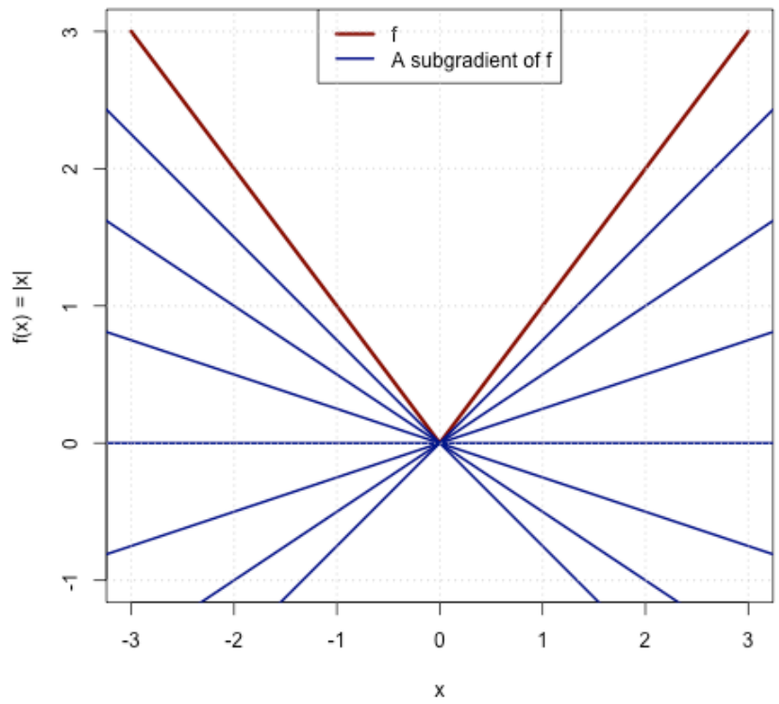
- The regularization by the ℓ_2-norm makes the problem easier to solve

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:

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

=> 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)

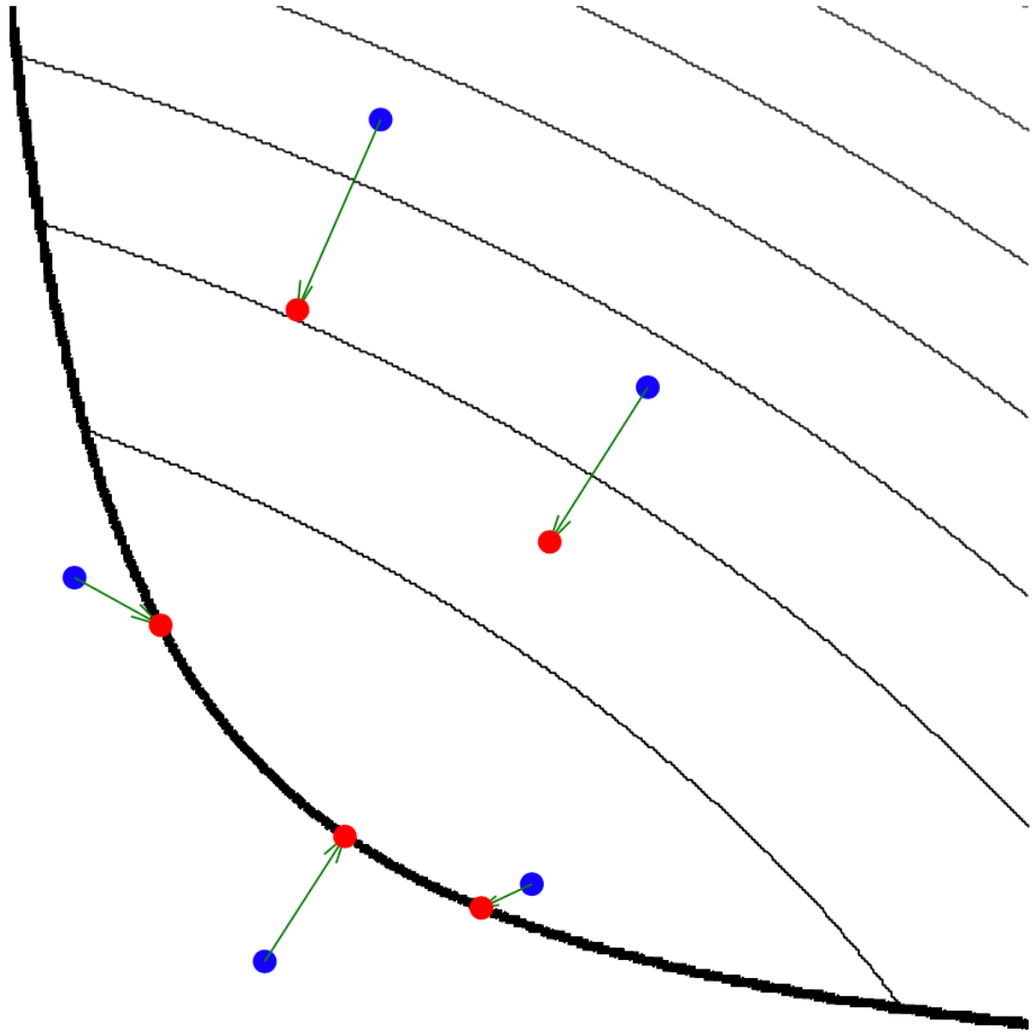
- The proximal operators and the sub-differential are related as follows

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

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



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

Proximal operators : a first property

Separability

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

$$\mathbf{prox}_{\mathcal{G}(.)}(s_1, s_2, \ldots, s_k) = (\mathbf{prox}_{g_1}(s_1), \mathbf{prox}_{g_2}(s_2), \ldots, \mathbf{prox}_{g_k}(s_k))$$

=> 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

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

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

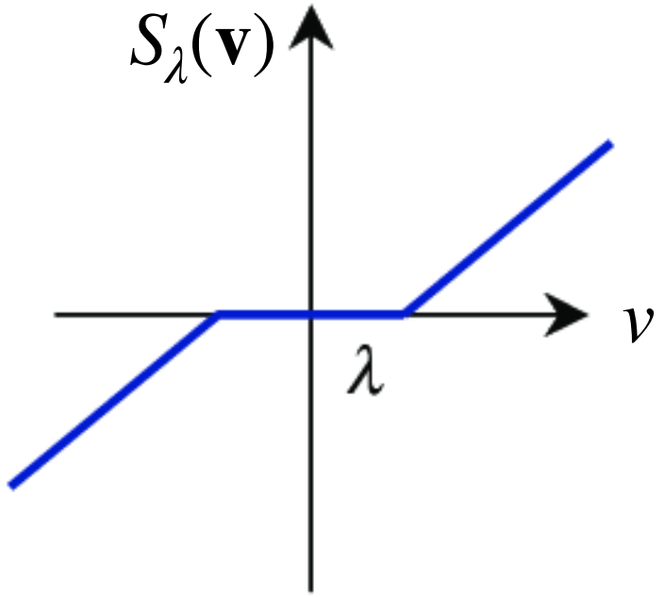
Example of proximal operators (1/3)

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

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

Then:

$$(\mathbf{prox}_{\lambda\|.\|_1}(\mathbf{v}))_i = S_\lambda(\mathbf{v}) = \begin{cases} v_i - \lambda & v_i \geq \lambda \\ 0 & |v_i| \leq \lambda \\ v_i + \lambda & v_i \leq -\lambda. \end{cases}$$



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

Example of proximal operators (2/3)

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

$$I_C(x) = \begin{cases} 0 & x \in C \\ +\infty & x \notin C, \end{cases}$$

Then:

$$\mathbf{prox}_f(\mathbf{s}) = \Pi_C(\mathbf{s}) = \arg\min_{\mathbf{x}\in C} \|\mathbf{x} - \mathbf{s}\|_2$$

The proximal operator is the Euclidean projection onto C

Example of proximal operators (3/3)

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

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

Then:

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

- If $\mathbf{\Phi}$ 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:

$\quad \mathbf{s}^{(k+1)} = \mathbf{prox}_{\mathcal{G}}(\mathbf{s}^{(k)})$

$\quad k \leftarrow k + 1$

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

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

- $h(\,.\,)$ smooth, convex as a function of $\mathbf{S}$

- Gradient descent algorithm:

```
Initialize S^(0)
k = 0
while not converged do:
    S^(k+1) = S^(k+1) − γ ∇h(S^(k))
    k ← k + 1
end
return S^(k)
```

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

- $\mathcal{G}(\,.\,)$ non-smooth, convex

- Proximal operator:

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

- Proximal point method:

```
Initialize S^(0)
k = 0
while not converged do:
    S^(k+1) = prox_{G(.)}(S^(k))
    k ← k + 1
end
return S^(k)
```

Forward-backward splitting (FBS)

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

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

```
Initialize s^(0)
k = 0
while not converged do:
    s^(k+1) = prox_{γG}(s^(k) − γ∇h(s^(k)))
    k ← k + 1
end
return 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

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

With the following assumptions:

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

- Theorem

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

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

- 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 $\left(h_{\mathbf{A}}(\mathbf{S}^{(k)}) + h_{\mathbf{A}}(\mathbf{S}^{(k)})\right)_k$ sequence, which converges to the optimal value, let say C^*, of the criterion.
  - The difference $|h_{\mathbf{A}}(\mathbf{S}^{(k)}) + h_{\mathbf{A}}(\mathbf{S}^{(k)}) - C^*|$ is majored by an arbitrary small constant δ in $\mathcal{O}(1/\delta)$ iterations.

FBS: exercise

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

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

Non-smooth minimization method: example

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

- 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:

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

$k \leftarrow k + 1$

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

$k = 1$

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

while **not** converged do:

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

$\mathbf{S}^{(k+1)} = S_{\gamma\mathcal{G}}\left(\mathbf{Z}^{(k+1)} - \gamma\nabla h_{\mathbf{A}}(\mathbf{Z}^{(k+1)})\right)$

$k \leftarrow k + 1$

end

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

- The first update can be seen as an extrapolation term
- $\omega^{(k)}$ can be for instance chosen as $\omega^{(k)} = \dfrac{k}{k+3}$

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 $|h_{\mathbf{A}}(\mathbf{S}^{(k)}) + h_{\mathbf{A}}(\mathbf{S}^{(k)}) - C^*|$ 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 $\left(h_{\mathbf{A}}(\mathbf{S}^{(k)}) + h_{\mathbf{A}}(\mathbf{S}^{(k)})\right)_k$ 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

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

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

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

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

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

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

GFBS: motivation (2/2)

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

- In this case though, both $\|\mathbf{S}\mathbf{\Phi}\|_1$ and $\iota_{\cdot>0}(\mathbf{S})$ 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:

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

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

$k = 0$

while **not** converged do:

  for l = 1..L do:

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

  endfor

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

$k \leftarrow k + 1$

endwhile

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

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

$\gamma \in (0,2L)$ 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?

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

- So far, we have tackled the problem over only a single variable (w.l.o.g., $\mathbf{S}$)

- How to deal with <u>both</u> $\mathbf{A}$ 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?

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

- 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–Lojasiewicz 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 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:

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

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

$k \leftarrow k + 1$

end

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

PALM : application to sparse BSS

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

- 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:

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

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

$k \leftarrow k+1$

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:

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

- 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 $\|.\|_1$ 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 $\mathbf{S}^*$

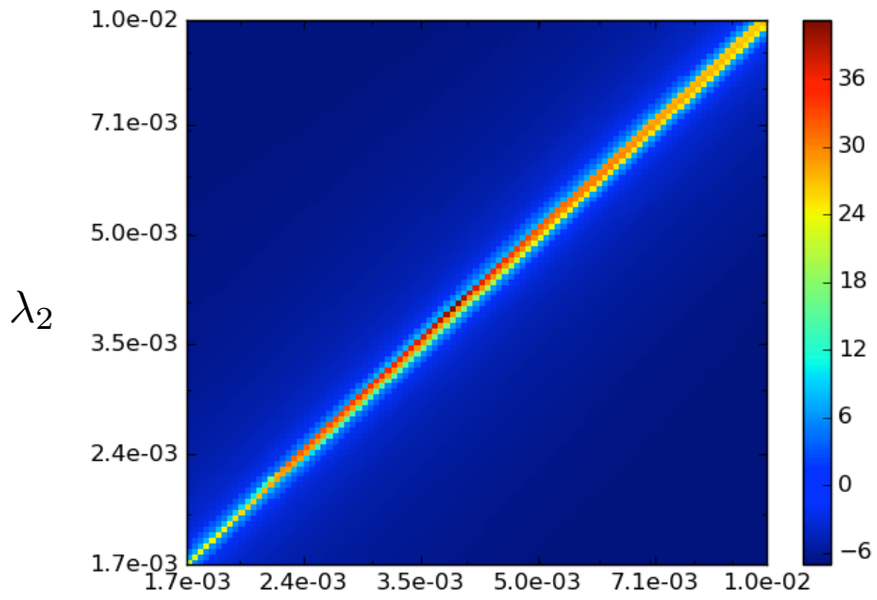
  => in particular, the solution depends on the regularization parameters λ values

Exact algorithms: impact of the regularization parameter choice

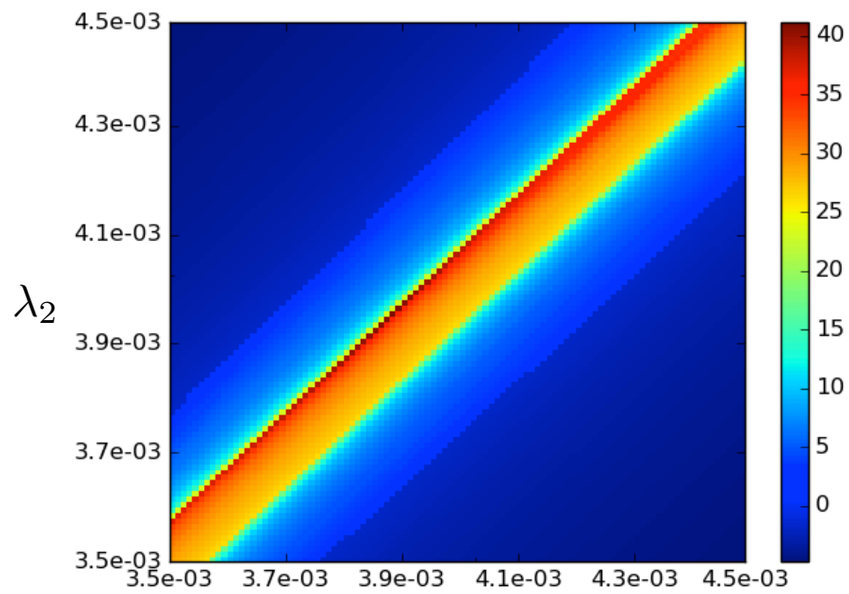
Empirical study: study sensitivity to regularization parameters

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

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



Median of C_A



Zoom on maximum

GMCA: 36 dB

- $3\text{x}10^{-4}$ change => 30 dB drop
- In diagonal, a $2\text{x}10^{-3}$ change => 7 dB drop

Generalized Morphological Component Analysis (GMCA)

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

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

- 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 $\mathbf{\Lambda}$ choice (the $\mathbf{\Lambda}$ 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…)

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

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

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

endfor

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

$k \leftarrow k + 1$

endwhile

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

Thus, $\mathbf{\Lambda} = diag(\lambda_1, \lambda_2, .., \lambda_n)$

- where † stands for the pseudo-inverse.

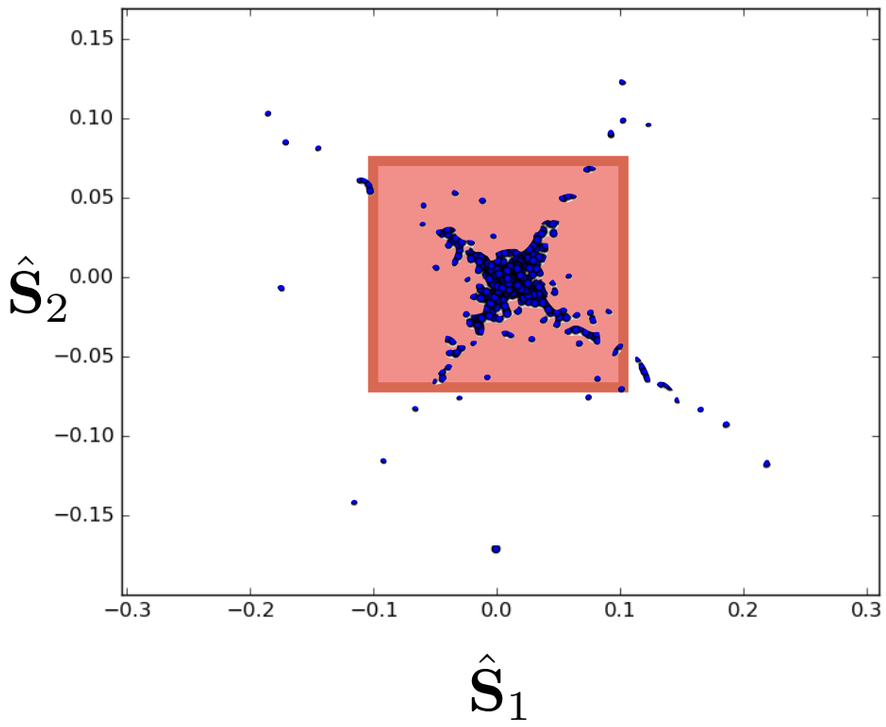
More about GMCA automatic hyper-parameter choice

- Use of **MAD[1] of the current source estimation** (new value at each iteration):

$$\lambda_i^{(l)} = \kappa \mathrm{MAD}[\tilde{\mathbf{S}}]_i = \kappa \mathrm{MAD}[\hat{\mathbf{A}}^{(l-1)\dagger}\mathbf{X}]_i$$

- MAD: robust STD estimator. Insensible to *sparse* perturbations: $\mathrm{MAD}(\mathbf{S}^*)$ small

- Straightforwardly, $\mathrm{MAD}[\hat{\mathbf{A}}^{(l-1)\dagger}\mathbf{X}] = \mathrm{MAD}[\underbrace{\hat{\mathbf{A}}^{(l-1)\dagger}\mathbf{A}^*\mathbf{S}^*}_{\textbf{Interferences}} + \underbrace{\hat{\mathbf{A}}^{(l-1)\dagger}\mathbf{N}}_{\textbf{Noise}}]$



If stabilization close to good

During first iterations, high λ_i

During the iterations, λ_i **decreases**

=> **interpretation in terms of signal recovery**

$$\lambda_i \simeq \mathrm{MAD}[\mathbf{A}^{*\dagger}\mathbf{N}]$$

[1] $\mathrm{MAD}(\mathbf{s}) \simeq 1.48\ \mathtt{median}(|\mathbf{s} - \mathtt{median}(\mathbf{s})|)$

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

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

    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,

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

  - You obtain

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

    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 ?	✓	✗	✓

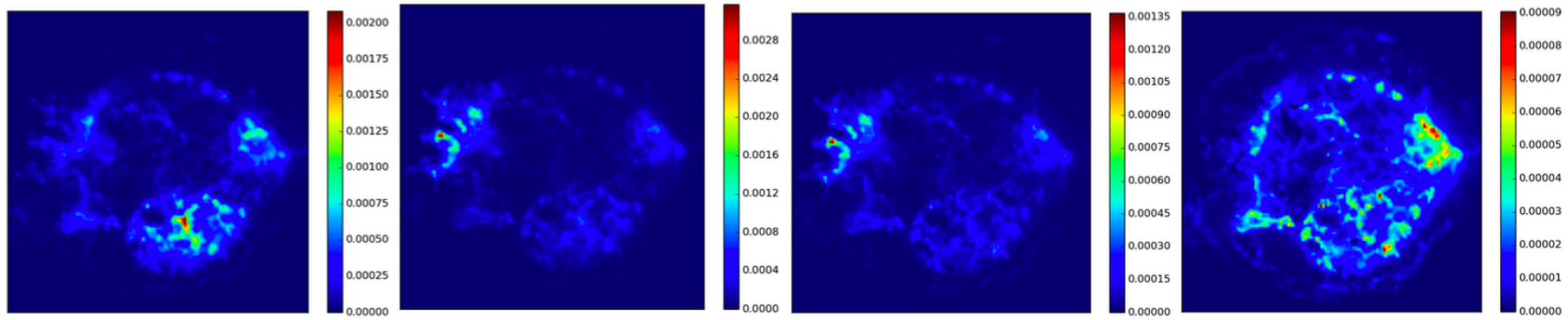
Easy to use

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 data

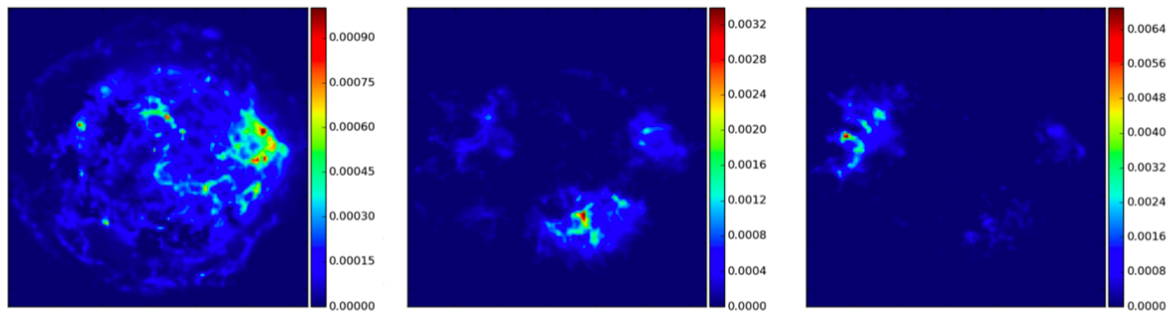
X



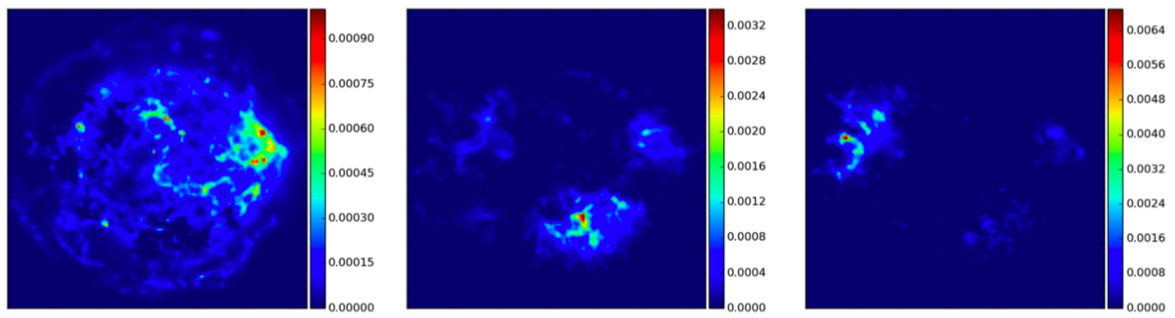
(a) $\mathbf{x}_1$ (b) $\mathbf{x}_5$ (c) $\mathbf{x}_9$ (d) $\mathbf{x}_{30}$

Synchrotron Fe 1 Fe 2

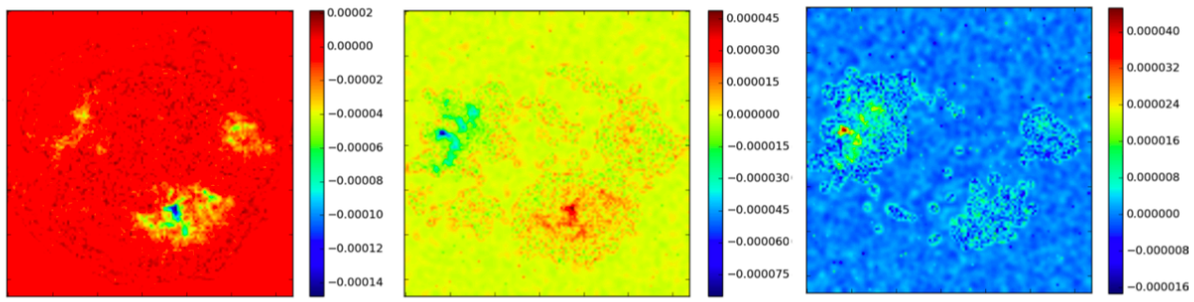
S*



S



Residual



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 $\|\mathbf{S}^* - \mathbf{P}\hat{\mathbf{S}}\|_F$ (what $\mathbf{P}$ mights stand for?)

- 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 **<u>algorithm unrolling (AU).</u>**

- In brief, **algorithm unrolling**

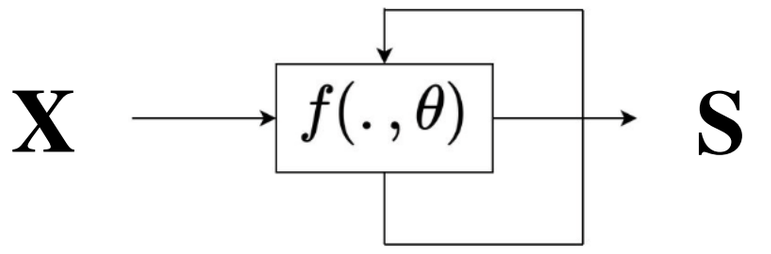
  => is a principled methodology to propose new neural network architectures for solving inverse problems, based on classical iterative algorithms

=> enables to bypass the cumbersome hyper-parameter choice

=> yields interpretable neural networks

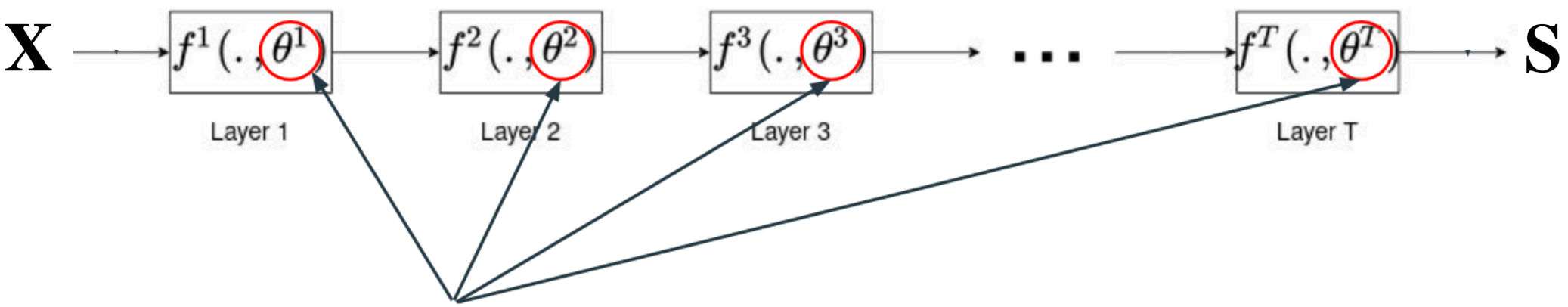
Algorithm unrolling: methodology

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



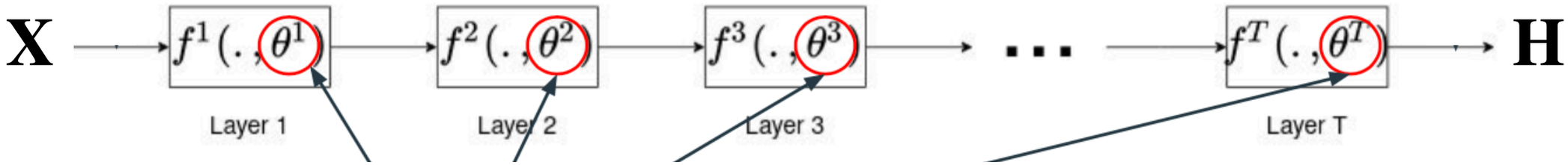
with θ the algorithm parameters (regularization, gradient step sizes…)

- 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 ${}^1\mathbf{X}, {}^2\mathbf{X}, \ldots, {}^{t_{train}}\mathbf{X}$ such that

$$\begin{aligned} {}^1\mathbf{X} &= \mathbf{A}^* \, {}^1\mathbf{S}^* + \mathbf{N} \\ {}^2\mathbf{X} &= \mathbf{A}^* \, {}^2\mathbf{S}^* + \mathbf{N} \\ &\cdots \\ {}^{t_{train}}\mathbf{X} &= \mathbf{A}^* \, {}^{t_{train}}\mathbf{S}^* + \mathbf{N} \end{aligned}$$

  - 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 $i = 1 \,..\, t_{train}$:

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

- $\Theta = \{\lambda^{*}, \gamma^{*}\}$ 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:

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

ISTA → **LISTA**

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

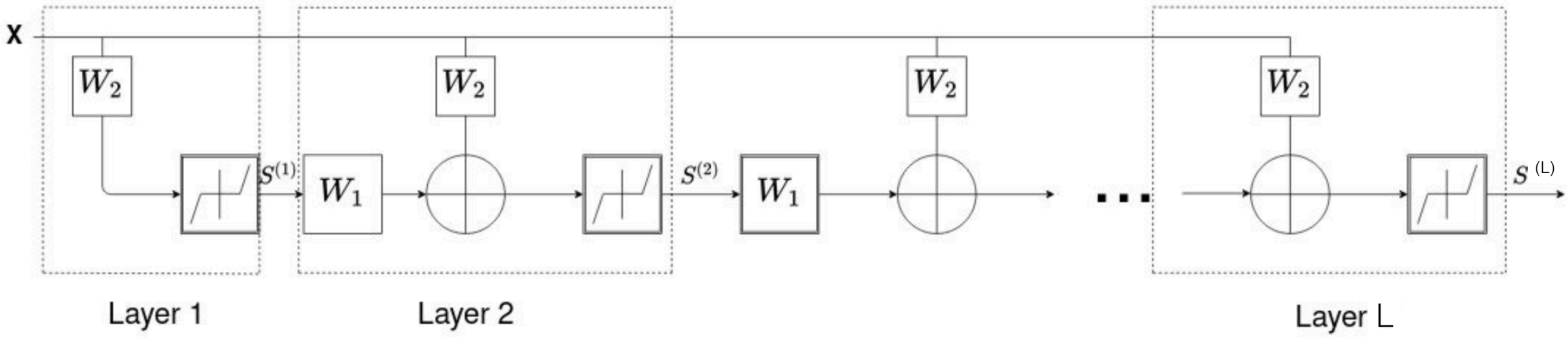
$$\underline{\theta} = \lambda\gamma,\ \underline{\mathbf{W}_1} = I_d - \gamma\mathbf{A}^{*T}\mathbf{A}^*,\ \underline{\mathbf{W}_2} = \gamma\mathbf{A}^{*T}\mathbf{A}^*$$

=> 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

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

- 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):

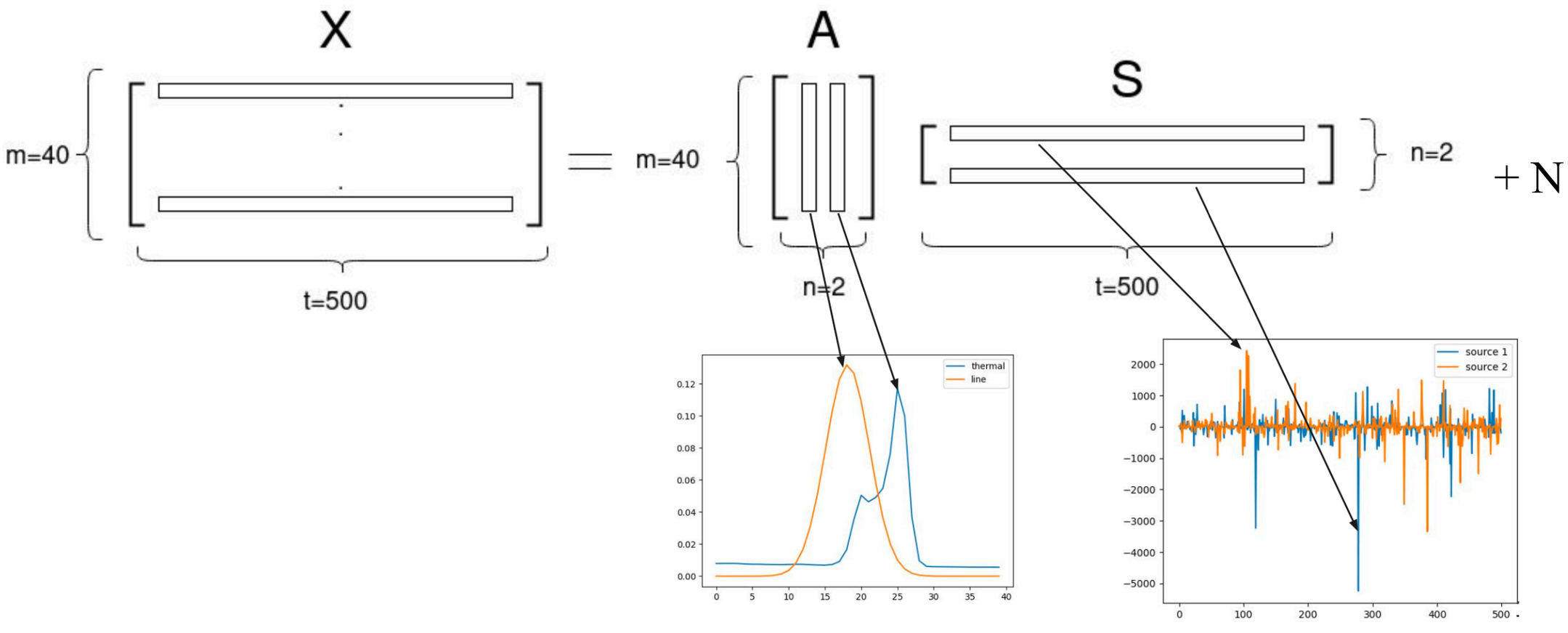
$$\mathbf{W}_1^{(1)}, \mathbf{W}_1^{(1)}, \ldots, \mathbf{W}_1^{(L)}$$
$$\mathbf{W}_2^{(1)}, \mathbf{W}_2^{(2)}, \ldots, \mathbf{W}_2^{(L)}$$
$$\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(L)}$$

LISTA limitations

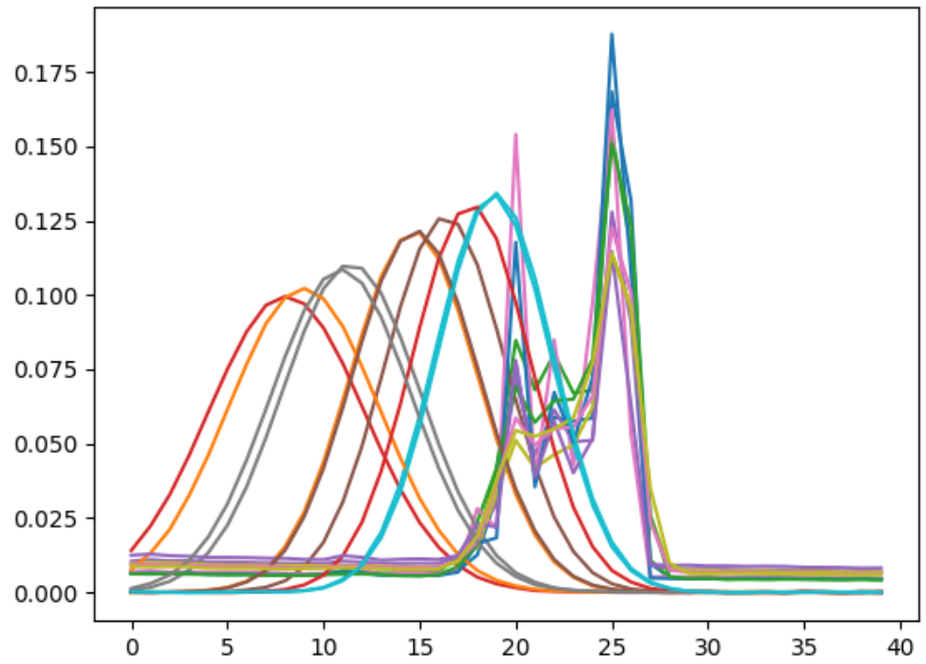
- LISTA was originally proposed in the sparse coding context. However, in sparse BSS:
  - The **A*** operator is not fixed for the whole data base (both in training and test sets)
  - Even worse, **A*** is unknown

- 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 **A*** matters by trying to estimate **S** in 2 experimental settings:
  - Fix **A*** (use a single **A*** for both the training and testing data sets)
  - Use varying **A*** in both sets

NMSE	Same **A***	Varying **A***
LISTA	3.12×10^{-4}	7.56×10^{-3}

=> it means that even with our gentle variations of **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 **A*** is fully used to update **S**

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

- In contrast, in LISTA

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

=> **A*** is not directly used, it only appears implicitly in

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

- However, $\mathbf{W}_1$ and $\mathbf{W}_2$ are the same for the *whole* training set.
- Therefore, they correspond to a kind of *marginalisation* of **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 **A***
- [chen18] have used the coupling of

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

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 **A**

NMSE	Same **A**	Varying **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

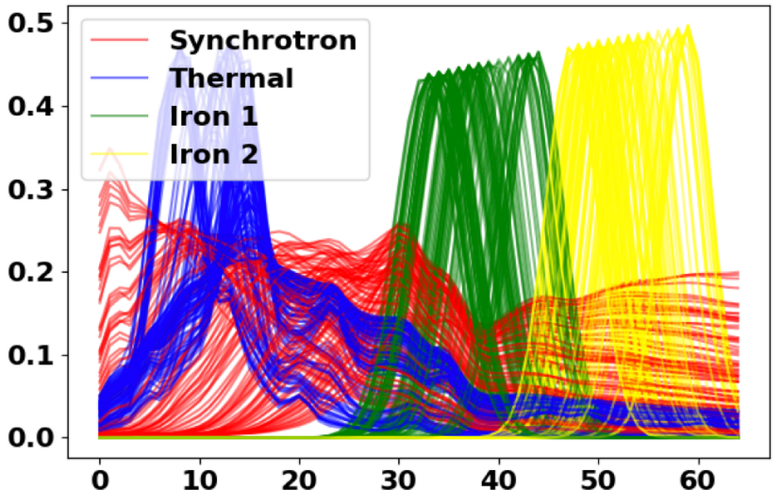
- So far, we knew either one of the factors
- In reality, both are unknown
- We can unroll PALM to give the **L-PALM** algorithm
- The **S** update has already been considered, and we can use a LISTA-CP update
- For **A**, there are much less variables to predict, so it might not be as useful to use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

- This yields the L-PALM updates:

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

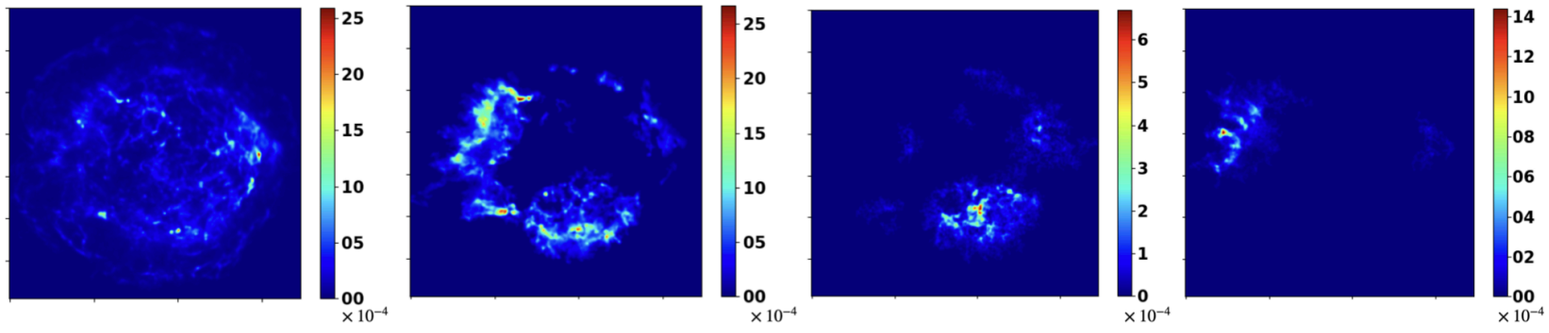
$$\mathbf{A}^{(k+1)} = \Pi_{\|.\|\leq 1}\left(\mathbf{A}^{(k)} + \frac{1}{\underline{L_A}}(\mathbf{X} - \mathbf{A}^{(k)}\mathbf{S}^{(k+1)})\mathbf{S}^{(k+1)T}\right) \qquad \text{(learning step-size)}$$

LPALM in practice



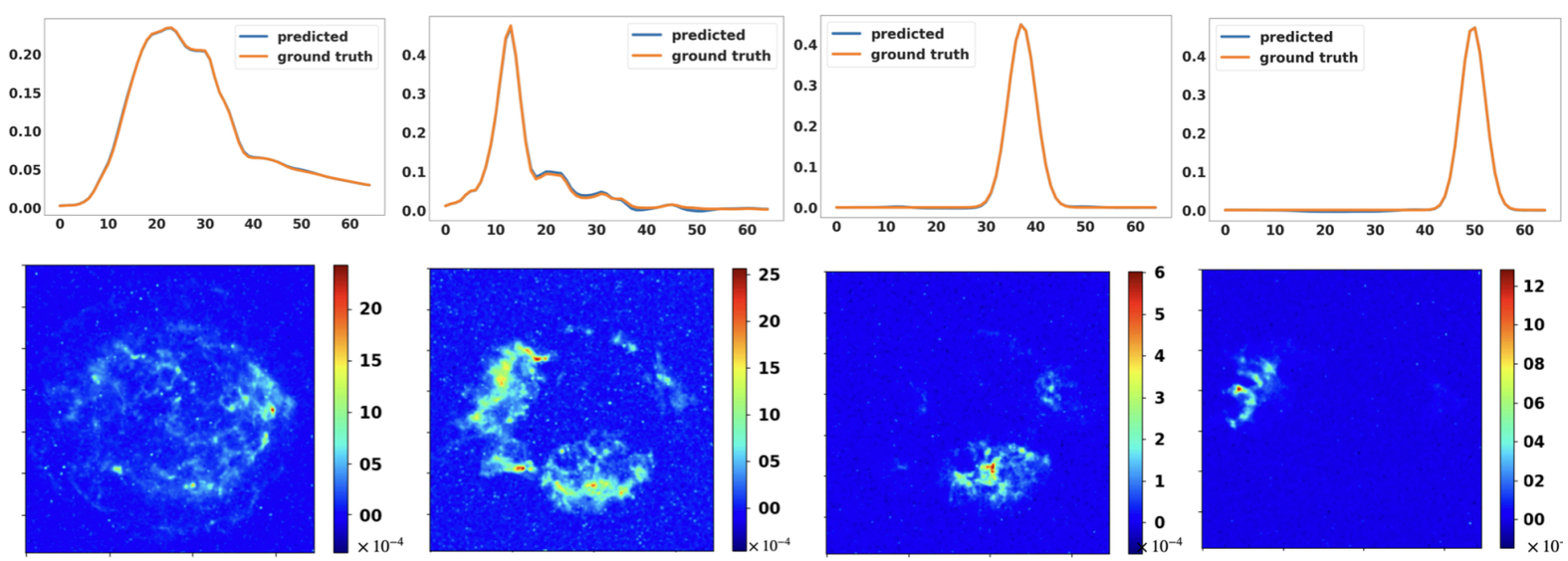
Datasets **A***

(750 training, 150 testing)

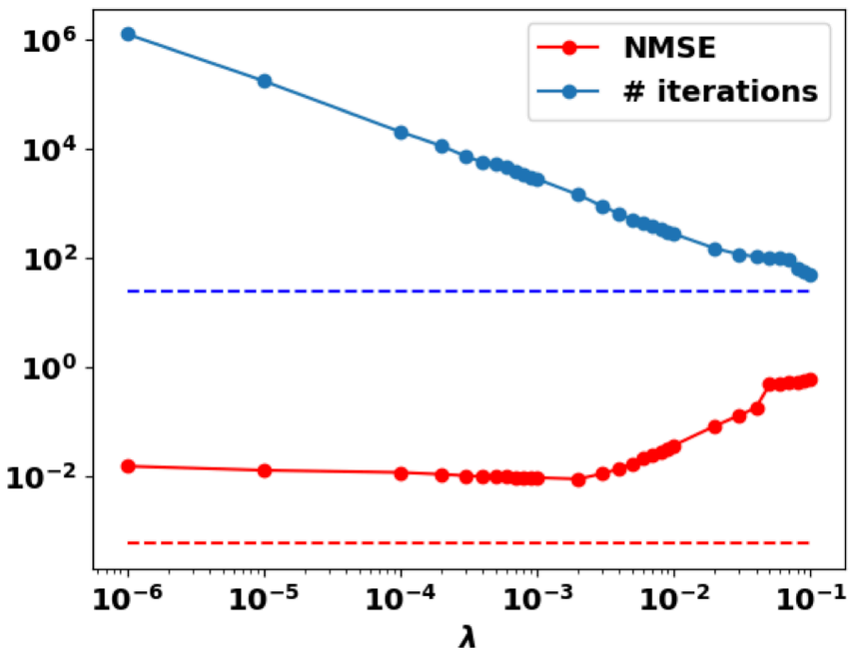


Dataset **S***

(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 prior), but it is still ongoing research.
- 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 ℓ1 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*.