

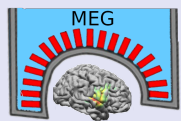
A Journey through Algorithm Unrolling for Inverse Problems

Thomas Moreau
INRIA Saclay - MIND Team

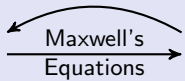


Inverse Problems

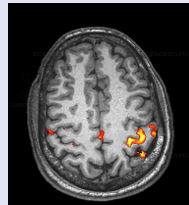
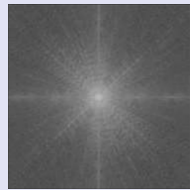
Neuroimaging – M/EEG



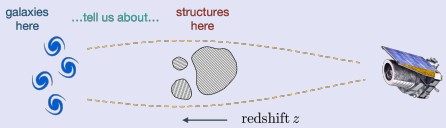
Inverse Problem



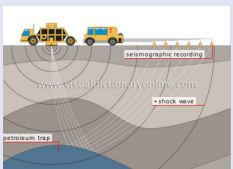
Neuroimaging – MRI



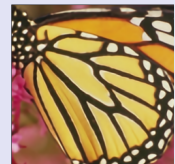
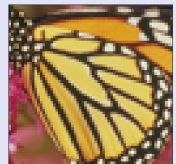
Astrophysics



Seismology – Prospection

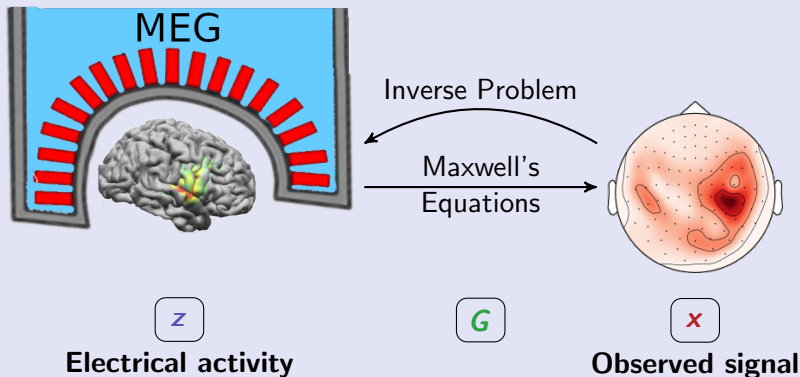


Imaging



Super-Resolution, Inpainting, Deblurring, ...

Inverse Problem: Source Localization for M/EEG



Forward model: $x = Gz + \varepsilon$

Inverse problem: find z from x

- ▶ Noisy problem: need to account for ε
- ▶ Ill-posed problem: many solutions z such that $Gz = x$, need to select one.

MAP estimate as a regularized regression problem

$$z^*(\mathbf{x}; \theta) = \operatorname{argmin}_z \underbrace{\frac{1}{2} \|\mathbf{x} - \mathbf{G}z\|_2^2}_{-\log p(\mathbf{x}|z)} + \underbrace{\mathcal{R}(z; \theta)}_{-\log p(z; \theta)}$$

where \mathcal{R} encodes prior information to select a good/plausible solution.

MAP estimate as a regularized regression problem

$$\mathbf{z}^*(\mathbf{x}; \theta) = \operatorname{argmin}_{\mathbf{z}} \underbrace{\frac{1}{2} \|\mathbf{x} - \mathbf{G}\mathbf{z}\|_2^2}_{-\log p(\mathbf{x}|\mathbf{z})} + \underbrace{\mathcal{R}(\mathbf{z}; \theta)}_{-\log p(\mathbf{z}; \theta)}$$

where \mathcal{R} encodes prior information to select a good/plausible solution.

Common framework:

- ▶ **Efficient solvers:** Forward backward, ADMM, ...
⇒ But might require many iterations to get quality estimate.
- ▶ **Flexible:** Can choose many priors – *handpicked, learned, implicit, ...*
⇒ Quality of the solution depends on the prior's choice $p(\mathbf{z}; \theta)$

Prior learning as a bilevel problem

Evaluate the quality of a solution with \mathcal{L} , and try to find the best prior:

$$\min_{\theta} \mathcal{L}(\mathbf{z}^*(\mathbf{x}; \theta)) \quad \text{s.t.} \quad \mathbf{z}^*(\mathbf{x}; \theta) = \operatorname{argmin}_{\mathbf{z}} \underbrace{-\log p(\mathbf{z}|\mathbf{x}; \theta)}_{F(\mathbf{z}, \theta)}$$

Prior learning as a bilevel problem

Evaluate the quality of a solution with \mathcal{L} , and try to find the best prior:

$$\min_{\theta} \mathcal{L}(\mathbf{z}^*(\mathbf{x}; \theta)) \quad \text{s.t.} \quad \mathbf{z}^*(\mathbf{x}; \theta) = \operatorname{argmin}_{\mathbf{z}} \underbrace{-\log p(\mathbf{z}|\mathbf{x}; \theta)}_{F(\mathbf{z}, \theta)}$$

How to solve such problem:

- ▶ *Random search*: sample some θ and keep the "best one".

⇒ Slow for θ in high dimension.

- ▶ *Gradient based method*: use first order information:

$$\frac{d\mathcal{L}(\mathbf{z}^*(\mathbf{x}; \theta))}{d\theta} = \frac{d\mathbf{z}^*(\mathbf{x}; \theta)}{d\theta}^\top \frac{\partial \mathcal{L}}{\partial \mathbf{z}}(\mathbf{z}^*(\mathbf{x}; \theta))$$

⇒ Expensive to compute $\mathbf{z}^*(\mathbf{x}; \theta)$ and its Jacobian.

Unrolling for prior learning

Idea:

- ▶ Replace $z^*(\mathbf{x}; \theta)$ by $z^N(\mathbf{x}; \theta, \psi)$ with hyperparameter ψ .
- ▶ Compute the Jacobian using backpropagation through the network.

Unrolling for prior learning

Idea:

- ▶ Replace $\mathbf{z}^*(\mathbf{x}; \theta)$ by $\mathbf{z}^N(\mathbf{x}; \theta, \psi)$ with hyperparameter ψ .
- ▶ Compute the Jacobian using backpropagation through the network.

⇒ **Why?**

Prior learning: learn θ to get the prior that gives the best reconstruction.

- ▶ *Supervised:* $\mathcal{L}(\theta) = \mathbb{E}_{(\mathbf{x}, \mathbf{z})} \frac{1}{2} \|\mathbf{z} - \mathbf{z}^N(\mathbf{x}; \theta, \psi)\|_2^2$
- ▶ *Unsupervised:* consistency loss, ...

Unrolling for prior learning

Idea:

- ▶ Replace $z^*(\mathbf{x}; \theta)$ by $z^N(\mathbf{x}; \theta, \psi)$ with hyperparameter ψ .
- ▶ Compute the Jacobian using backpropagation through the network.

⇒ **Why?**

Learned solver: To solve with many \mathbf{x} and a single \mathbf{G} , learn ψ

- ▶ Learning algorithm to resolve the original problem faster.
- ▶ With supervised or unsupervised losses.

Unrolling for prior learning

Idea:

- ▶ Replace $z^*(\mathbf{x}; \theta)$ by $z^N(\mathbf{x}; \theta, \psi)$ with hyperparameter ψ .
- ▶ Compute the Jacobian using backpropagation through the network.

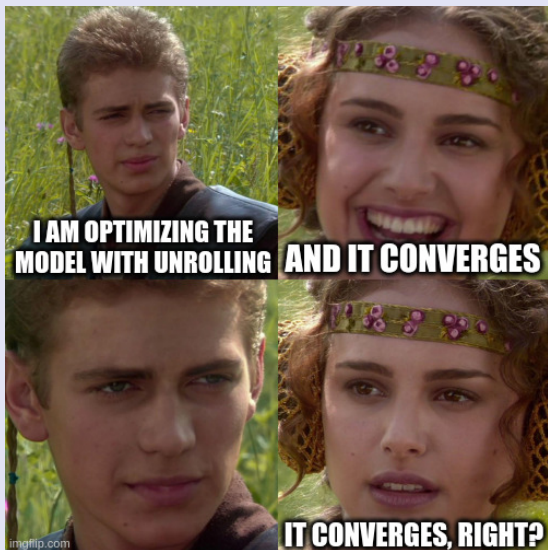
⇒ **Why?**

Learned solver: To solve with many \mathbf{x} and a single \mathbf{G} , learn ψ

- ▶ Learning algorithm to resolve the original problem faster.
- ▶ With supervised or unsupervised losses.

⇒ What can we say about the learned procedure?
Convergence toward $z^*(\mathbf{x}; \theta)$?

Unrolling for prior learning



A bilevel view on prior learning with unrolling

References

- ▶ Ablin, P., Peyré, G., and **TM** (2020). Super-efficiency of automatic differentiation for functions defined as a minimum, In *ICML*
- ▶ Malézieux, B., **TM**, and Kowalski, M. (2022). Understanding approximate and Unrolled Dictionary Learning for Pattern Recovery, In *ICLR*

Unrolling with min-min problems

Bi-level formulation:

$$\min_{\theta \in \mathcal{C}} h(\theta) \triangleq F(\theta, \mathbf{z}^*(\theta)) \quad \text{s.t.} \quad \mathbf{z}^*(\theta) = \underset{z}{\operatorname{argmin}} F(\theta, z) .$$

Optimization problem in D solved with projected gradient descent.

\Rightarrow How to estimate the gradient $g^*(\theta) = \nabla h(\theta)$ efficiently?

Unrolling with min-min problems

Bi-level formulation:

$$\min_{\theta \in \mathcal{C}} h(\theta) \triangleq F(\theta, \mathbf{z}^*(\theta)) \quad \text{s.t.} \quad \mathbf{z}^*(\theta) = \underset{z}{\operatorname{argmin}} F(\theta, z) .$$

Optimization problem in D solved with projected gradient descent.

\Rightarrow How to estimate the gradient $g^*(\theta) = \nabla h(\theta)$ efficiently?

Danskin Theorem:

[Danskin, 1967]

$$g^*(\theta) = \nabla_1 F(\theta, \mathbf{z}^*(\theta))$$

This is due to the fact that “ $\nabla_2 F(\theta, \mathbf{z}^(\theta)) = 0$ ”.*

Unrolling with min-min problems

Bi-level formulation:

$$\min_{\theta \in \mathcal{C}} h(\theta) \triangleq F(\theta, \mathbf{z}^*(\theta)) \quad \text{s.t.} \quad \mathbf{z}^*(\theta) = \underset{z}{\operatorname{argmin}} F(\theta, z) .$$

Optimization problem in D solved with projected gradient descent.

\Rightarrow How to estimate the gradient $g^*(\theta) = \nabla h(\theta)$ efficiently?

Danskin Theorem:

[Danskin, 1967]

$$g^*(\theta) = \nabla_1 F(\theta, \mathbf{z}^*(\theta))$$

This is due to the fact that “ $\nabla_2 F(\theta, \mathbf{z}^(\theta)) = 0$ ”.*

Issue: computing $\mathbf{z}^*(\theta)$ is computationally expensive.

Unrolling with min-min problems

Unrolled formulation:

$$\min_{\theta \in \mathcal{C}} h_N(\theta) \triangleq F(\theta, \mathbf{z}^N(\theta)) .$$

The gradient estimate becomes:

$$\mathbf{g}_N^2(\theta) = \nabla_1 F(\theta, \mathbf{z}^N(\theta)) + J_N^\top \nabla_2 F(\theta, \mathbf{z}^N(\theta))$$

Estimate the jacobian $J_N = \frac{d\mathbf{z}^N}{d\theta}$ with back-propagation.

Unrolling with min-min problems

Unrolled formulation:

$$\min_{\theta \in \mathcal{C}} h_N(\theta) \triangleq F(\theta, \mathbf{z}^N(\theta)) .$$

The gradient estimate becomes:

$$\mathbf{g}_N^2(\theta) = \nabla_1 F(\theta, \mathbf{z}^N(\theta)) + J_N^\top \nabla_2 F(\theta, \mathbf{z}^N(\theta))$$

Estimate the jacobian $J_N = \frac{d\mathbf{z}^N}{d\theta}$ with back-propagation.

Question: More efficient to use unrolling than classic AM?

- ▶ Work for smooth problems. [Ablin et al., ICML 2020]
- ▶ Improved performances for supervised learning. [Monga et al., 2021]

Alternate Minimization

No Jacobian estimation

$$g_N^1(\theta) = \nabla_1 F(\theta, \mathbf{z}^N(\theta))$$

Unrolling

Account for Jacobian of \mathbf{z}^N

$$g_N^2(\theta) = \nabla_1 F(\theta, \mathbf{z}^N(\theta)) \\ + J_N^\top \nabla_2 F(\theta, \mathbf{z}^N(\theta))$$

Alternate Minimization

No Jacobian estimation

$$g_N^1(\theta) = \nabla_1 F(\theta, \mathbf{z}^N(\theta))$$

Converges as fast as \mathbf{z}^N

$$\|g_N^1 - g^*\|_2 \leq L_1 \|\mathbf{z}^N - \mathbf{z}^*\|_2$$

Unrolling

Account for Jacobian of \mathbf{z}^N

$$g_N^2(\theta) = \nabla_1 F(\theta, \mathbf{z}^N(\theta)) \\ + J_N^\top \nabla_2 F(\theta, \mathbf{z}^N(\theta))$$

Alternate Minimization

No Jacobian estimation

$$g_N^1(\theta) = \nabla_1 F(\theta, \mathbf{z}^N(\theta))$$

Converges as fast as \mathbf{z}^N

$$\|g_N^1 - g^*\|_2 \leq L_1 \|\mathbf{z}^N - \mathbf{z}^*\|_2$$

Unrolling

Account for Jacobian of \mathbf{z}^N

$$g_N^2(\theta) = \nabla_1 F(\theta, \mathbf{z}^N(\theta)) \\ + J_N^\top \nabla_2 F(\theta, \mathbf{z}^N(\theta))$$

May converge faster than \mathbf{z}^N

$$\|g_N^2 - g^*\| \leq L \|J_N - J^*\|_2 \|\mathbf{z}^N - \mathbf{z}^*\|_2 \\ + L_2 \|\mathbf{z}^N - \mathbf{z}^*\|_2^2$$

\Rightarrow Need to study $\|J_N - J^*\|_2$.

Differentiable unrolling of z^N

Idea: Compute $J_N = \frac{\partial z^N}{\partial \theta}(\theta) \approx \frac{\partial z^*}{\partial \theta}(\theta)$ using automatic differentiation through an iterative algorithm.

Differentiable unrolling of \mathbf{z}^N

Idea: Compute $J_N = \frac{\partial \mathbf{z}^N}{\partial \theta}(\theta) \approx \frac{\partial \mathbf{z}^*}{\partial \theta}(\theta)$ using automatic differentiation through an iterative algorithm.

For the gradient descent algorithm:

$$\mathbf{z}^{N+1} = \mathbf{z}^N - \rho \frac{\partial F}{\partial \mathbf{z}}(\theta, \mathbf{z}^N)$$

The Jacobian reads,

$$\frac{\partial \mathbf{z}^{N+1}}{\partial \theta}(\theta) = \left(Id - \rho \frac{\partial^2 F}{\partial \mathbf{z}^2}(\theta, \mathbf{z}^N) \right) \frac{\partial \mathbf{z}^N}{\partial \theta}(\theta) - \rho \frac{\partial^2 F}{\partial \mathbf{z} \partial \theta}(\theta, \mathbf{z}^N)$$

Differentiable unrolling of \mathbf{z}^N

Idea: Compute $J_N = \frac{\partial \mathbf{z}^N}{\partial \theta}(\theta) \approx \frac{\partial \mathbf{z}^*}{\partial \theta}(\theta)$ using automatic differentiation through an iterative algorithm.

For the gradient descent algorithm:

$$\mathbf{z}^{N+1} = \mathbf{z}^N - \rho \frac{\partial F}{\partial \mathbf{z}}(\theta, \mathbf{z}^N)$$

The Jacobian reads,

$$\frac{\partial \mathbf{z}^{N+1}}{\partial \theta}(\theta) = \left(Id - \rho \frac{\partial^2 F}{\partial \mathbf{z}^2}(\theta, \mathbf{z}^N) \right) \frac{\partial \mathbf{z}^N}{\partial \theta}(\theta) - \rho \frac{\partial^2 F}{\partial \mathbf{z} \partial \theta}(\theta, \mathbf{z}^N)$$

\Rightarrow Under smoothness conditions, if \mathbf{z}^N converges to \mathbf{z}^* ,
this converges toward $\frac{\partial \mathbf{z}^*}{\partial \theta}(\theta)$

We consider the 3 gradient estimates:

- ▶ $g_1^N = \nabla_{\theta} F(\theta, \mathbf{z}^N)$ Analysis
- ▶ $g_2^N = \nabla_{\theta} F(\theta, \mathbf{z}^N) + \frac{\partial \mathbf{z}^N}{\partial \theta}^{\top} \nabla_{\mathbf{z}} F(\theta, \mathbf{z}^N)$ Automatic
- ▶ $g_3^N = \nabla_{\theta} F(\theta, \mathbf{z}^N) - \frac{\partial^2 G}{\partial \mathbf{z} \partial \theta}(\theta, \mathbf{z}^N) \frac{\partial^2 G}{\partial \mathbf{z}^2}^{-1}(\theta, \mathbf{z}^N) \nabla_{\mathbf{z}} F(\theta, \mathbf{z}^N)$ Implicit

We consider the 3 gradient estimates:

$$\blacktriangleright g_1^N = \nabla_{\theta} F(\theta, \mathbf{z}^N)$$

Analysis

$$\blacktriangleright g_2^N = \nabla_{\theta} F(\theta, \mathbf{z}^N) + \frac{\partial \mathbf{z}^N}{\partial \theta}^{\top} \nabla_{\mathbf{z}} F(\theta, \mathbf{z}^N)$$

Automatic

$$\blacktriangleright g_3^N = \nabla_{\theta} F(\theta, \mathbf{z}^N) - \frac{\partial^2 G}{\partial \mathbf{z} \partial \theta}(\theta, \mathbf{z}^N) \frac{\partial^2 G}{\partial \mathbf{z}^2}^{-1}(\theta, \mathbf{z}^N) \nabla_{\mathbf{z}} F(\theta, \mathbf{z}^N)$$

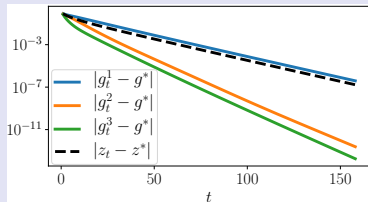
Implicit

Convergence rates: For G strongly convex in \mathbf{z} ,

$$|g_1^N(x) - g^*(x)| = O\left(|\mathbf{z}^N(\theta) - \mathbf{z}^*(\theta)|\right),$$

$$|g_t^N(x) - g^*(x)| = o\left(|\mathbf{z}^N(\theta) - \mathbf{z}^*(\theta)|\right),$$

$$|g_3^N(x) - g^*(x)| = O\left(|\mathbf{z}^N(\theta) - \mathbf{z}^*(\theta)|^2\right).$$



What about non-smooth problem?

Very common in inverse problem.

What about non-smooth problem?

Very common in inverse problem.

⇒ Here, we consider the case of the Lasso.

$$z^* = \operatorname{argmin} \|x - GDz\|_2^2 + \lambda \|z\|_1$$

with $\theta = D$.

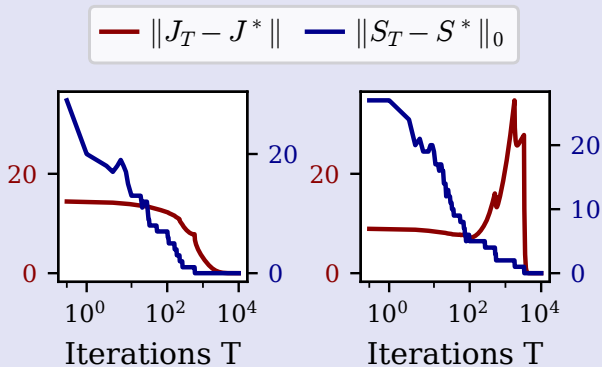
Convergence of the Jacobian

$$\|J_N - J^*\|_2 \leq A_N + B_N .$$

A_N converges linearly towards 0, B_N is an error term which may increase for large N and vanishes on the support of z^* .

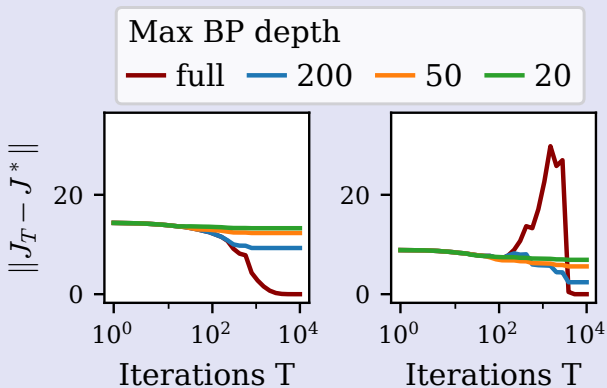
- ▶ On the support, the jacobian converges linearly.
- ▶ Before reaching the support, B_N is an error term that can accumulate.
- ▶ B_N can be attenuated with truncated back-propagation.

Empirical evaluation



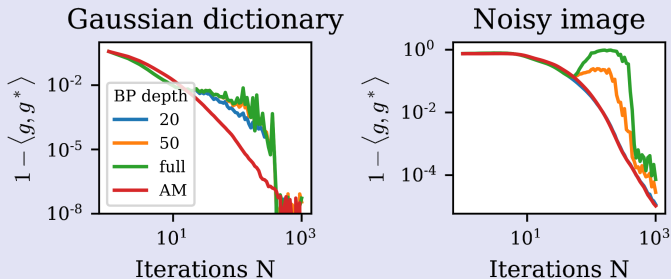
- ▶ Linear convergence once the support S^* is reached.
- ▶ Possible explosion before reaching S^* .

Empirical evaluation



- ▶ Truncated backpropagation (BP) reduces the explosion.
- ▶ Less precise when the support is reached.

Numerical experiments on gradient



- ▶ **First iterations:** Stable behavior.
- ▶ **Too many iterations:** Numerical instabilities due to the accumulation of errors. Truncated back-propagation reduces the errors.
- ▶ **On the support:** Convergence towards g^* .

Unrolling for Jacobian estimation

Not the expected performance boost in the non-smooth case.

- ▶ Jacobian estimate stable only for a very low number of iteration.

⇒ What does this mean for unrolling?

- ▶ Still interesting to solve the problem:

$$\min_{\theta} \mathcal{L}(\mathbf{z}^N(\mathbf{x}; \theta, \psi))$$

with $\mathbf{z}^N(\mathbf{x}; \theta, \psi)$ an unrolled algorithm with N steps.

- ▶ But we are not optimizing for \mathbf{z}^* .

⇒ We are not independent of how we obtain \mathbf{z}^N .

Iteration overfitting with unrolled optimization

References

- ▶ Ramzi, Z., Ablin, P., Peyré, G., and **TM** (2023). [Test like you Train in Implicit Deep Learning](#)

Deqs – Deep Equilibrium Networks

Consider the DEqs framework (*more general than bilevel*)

$$\min_{\theta} \mathcal{L}(\mathbf{z}^*(\theta)) \quad \text{s.t.} \quad \mathbf{z}^*(\theta) = f_{\theta}(\mathbf{z}^*(\theta))$$

Deqs – Deep Equilibrium Networks

Consider the DEqs framework (*more general than bilevel*)

$$\min_{\theta} \mathcal{L}(\mathbf{z}^*(\theta)) \quad \text{s.t.} \quad \mathbf{z}^*(\theta) = f_{\theta}(\mathbf{z}^*(\theta))$$

In practice, solved as

$$\theta^{*,N} = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\mathbf{z}^N(\theta))$$

with $\mathbf{z}^N(\theta)$ obtained through N iterations of an algorithm.

The promise of these models: you can use $M > N$ during test time to get performance boost.

Deqs – Deep Equilibrium Networks

Consider the DEqs framework (*more general than bilevel*)

$$\min_{\theta} \mathcal{L}(\mathbf{z}^*(\theta)) \quad \text{s.t.} \quad \mathbf{z}^*(\theta) = f_{\theta}(\mathbf{z}^*(\theta))$$

In practice, solved as

$$\theta^{*,N} = \operatorname{argmin}_{\theta} \mathcal{L}(\mathbf{z}^N(\theta))$$

with $\mathbf{z}^N(\theta)$ obtained through N iterations of an algorithm.

The promise of these models: you can use $M > N$ during test time to get performance boost.

⇒ Is this really true?

If we learn $\theta^{*,N}$ with a given N , what can you say about $\mathcal{L}(z^{N+\Delta N}(\theta^{*,N}))$?

If we learn $\theta^{*,N}$ with a given N , what can you say about $\mathcal{L}(z^{N+\Delta N}(\theta^{*,N}))$?

Theorem 1 – Iteration overfitting

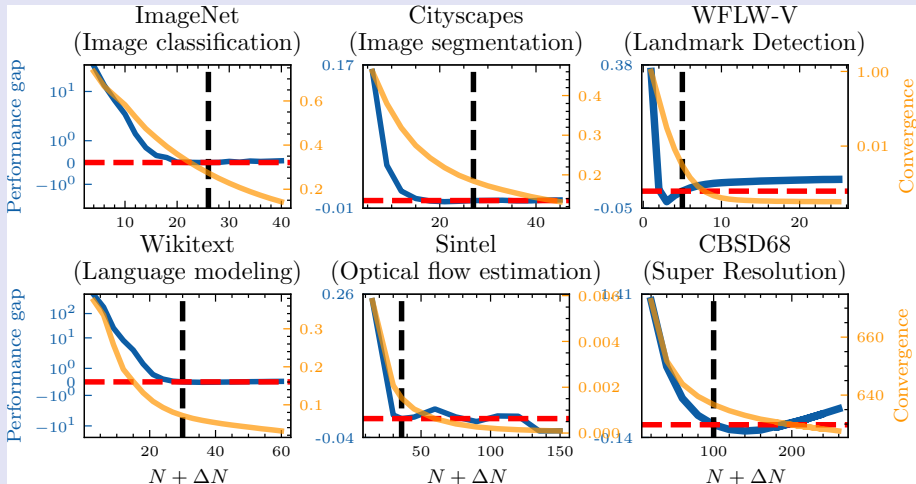
Under simplifying hypothesis (linear DEqs), if f_θ is overparametrized, we have for all ΔN :

$$\mathcal{L}(z^{N+\Delta N}(\theta^{*,N})) \geq \mathcal{L}(z^N(\theta^{*,N})), \quad (1)$$

We also show that the closer to overparametrized f_θ is, the less we expect to see improvement with $N + \Delta N$.

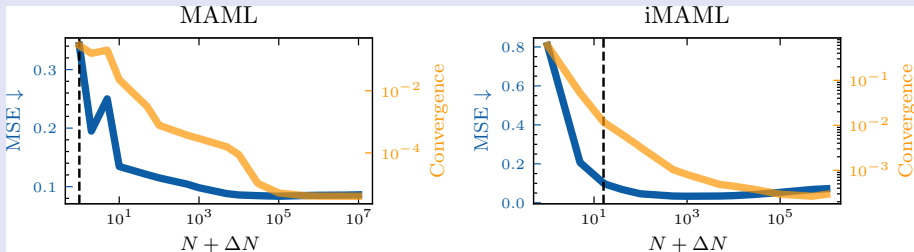
What happens in practice?

Context: Overparametrized DEQs.



What happens in practice?


Context: Underparametrized Meta-learning.



Take-home message

- ▶ Unrolled networks work well for smooth minimization
- ▶ **For non-smooth problems, the jacobian estimate is unstable**
- ▶ When training with fixed number of iterations, it makes sense to use the same number of iterations at test time.

*Bench***o***pt*

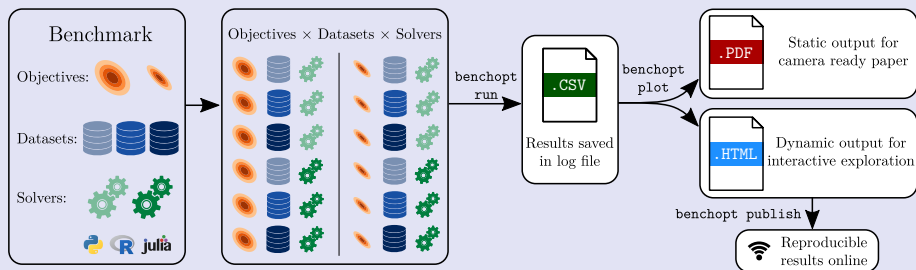


Reproducing a scientific comparison from an article can be as easy as:

```
git clone https://github.com/benchopt/benchmark_bilevel
benchopt run ./benchmark_bilevel
```



Benchopt: principle



⇒ Each object can be parametrized so multiple scenario can be tested.

Making tedious tasks easy:

- ▶ Sharing code
- ▶ Adding methods
- ▶ Exploring results
- ▶ Varying hyperparameters
- ▶ Running in Parallel
- ▶ Caching
- ▶ ...

Join us!



Benchopt sprint in Paris last July.

⇒ Next sprint in June, stay tuned!

Thanks for your attention!

Slides are on my web page:



tommoral.github.io



[@tomamoral](https://twitter.com/tomamoral)