Bi-level optimization in Machine Learning

Thomas Moreau INRIA Saclay

PARIETAL

Inria informatics mathematics

A classical Machine Learning Pipeline:

- Get some data X and supervised task y,

A classical Machine Learning Pipeline:

- Get some data X and supervised task y,
- Select a class of model $f(\cdot\,;\theta,\lambda)$ to solve the task,

A classical Machine Learning Pipeline:

- Get some data X and supervised task y,
- Select a class of model $f(\cdot\,;\theta,\lambda)$ to solve the task,
- Split the data between a training set X_{tr}, y_{tr} and a validation set X_{val}, y_{val} (*multiple time for CV*),

A classical Machine Learning Pipeline:

- Get some data X and supervised task y,
- Select a class of model $f(\cdot\,; \theta, \lambda)$ to solve the task,
- Split the data between a training set X_{tr}, y_{tr} and a validation set X_{val}, y_{val} (*multiple time for CV*),
- Train the model on X_{tr}, y_{tr} to find the best model parameters θ^*

$$\theta^*(\lambda) = \operatorname*{argmin}_{\theta} G(\lambda, \theta) = \frac{1}{M} \sum_{i=1}^{M} \ell\Big(f(X_{tr,i}, \theta, \lambda), y_{tr,i}\Big) \ ,$$

A classical Machine Learning Pipeline:

- Get some data X and supervised task y,
- Select a class of model $f(\cdot\,; \theta, \lambda)$ to solve the task,
- Split the data between a training set X_{tr}, y_{tr} and a validation set X_{val}, y_{val} (*multiple time for CV*),
- Train the model on X_{tr}, y_{tr} to find the best model parameters θ^*

$$\theta^*(\lambda) = \operatorname*{argmin}_{\theta} G(\lambda, \theta) = \frac{1}{M}\sum_{i=1}^{M} \ell\Big(f(X_{tr,i}, \theta, \lambda), y_{tr,i}\Big) \ ,$$

- Evaluate the performances on X_{val}, y_{val} with accuracy:

$$F(\lambda, \theta) = \frac{1}{N}\sum_{j=1}^{N} \tilde{\ell}\Big(f(X_{val,j}; \theta^*, \lambda), y_{val,j}\Big) \ .$$

A classical Machine Learning Pipeline:

- Get some data X and supervised task y,
- Select a class of model $f(\cdot\ ; \theta, \lambda)$ to solve the task,
- Split the data between a training set X_{tr}, y_{tr} and a validation set X_{val}, y_{val} (*multiple time for CV*),
- Train the model on X_{tr}, y_{tr} to find the best model parameters θ^*

$$\theta^*(\lambda) = \underset{\theta}{\operatorname{argmin}}\, G(\lambda, \theta) = \frac{1}{M} \sum_{i=1}^{M} \ell\Big(f(X_{tr,i}, \theta, \lambda), y_{tr,i}\Big) \ ,$$

- Evaluate the performances on X_{val}, y_{val} with accuracy:

$$F(\lambda, \theta) = \frac{1}{N} \sum_{j=1}^{N} \tilde{\ell}\Big(f(X_{val,j}; \theta^*, \lambda), y_{val,j}\Big) \ .$$

$\Rightarrow$ The 1 000 000€ question: How to select λ, f, X_{tr}, ...?

Hyper-parameter selection λ: the Grid Search

Idea: try many parameters and keep the best one according to the validation loss.

Hyper-parameter selection λ: the Grid Search

Idea: try many parameters and keep the best one according to the validation loss.

- Select a grid of hyper-parameters $\{\lambda_1, \dots, \lambda_K\}$,
- For each λ_k, train the model the best parameters θ_k^*

$$\theta_k^* = \operatorname*{argmin}_{\theta} G(\lambda_k, \theta) = \frac{1}{M} \sum_{i=1}^{M} \ell\Big(f(X_{tr,i}, \theta, \lambda), y_{tr,i}\Big) \ ,$$

- Select the best model performances

$$\lambda^* = \operatorname*{argmin}_{\lambda_k} F(\lambda_k, \theta_k^*) = \frac{1}{N} \sum_{j=1}^{N} \tilde{\ell}\Big(f(X_{val,j}; \theta_k^*, \lambda_k), y_{val,j}\Big) \ .$$

Hyper-parameter selection λ: the Grid Search

Idea: try many parameters and keep the best one according to the validation loss.

- Select a grid of hyper-parameters $\{\lambda_1, \dots, \lambda_K\}$,
- For each λ_k, train the model the best parameters θ_k^*

$$\theta_k^* = \underset{\theta}{\operatorname{argmin}}\, G(\lambda_k, \theta) = \frac{1}{M}\sum_{i=1}^{M} \ell\Big(f(X_{tr,i}, \theta, \lambda), y_{tr,i}\Big) \ ,$$

- Select the best model performances

$$\lambda^* = \underset{\lambda_k}{\operatorname{argmin}}\, F(\lambda_k, \theta_k^*) = \frac{1}{N}\sum_{j=1}^{N} \tilde{\ell}\Big(f(X_{val,j}; \theta_k^*, \lambda_k), y_{val,j}\Big) \ .$$

$\Rightarrow$ This is a bi-level optimization problem.

Bi-level problem: Optimization problem with two levels

$$\min_{x} h(\lambda) = F(\lambda, \theta^*(x))$$

$$s.t. \quad \theta^*(\lambda) = \operatorname*{argmin}_{\theta} G(\lambda, \theta)$$

Outer function

Value function

Inner function/Problem

Goal: Optimize the value function h whose value depends on the result of another optimization problem.

$\Rightarrow$ Challenging to theoretically and practically.

Other bi-level optimization problems: Model selection

Selecting the best model: G is the training loss and θ are the parameters of the model. The goal is to optimize λ to get the best validation loss F,

- *Hyperparameter optimization:* λ are the regularisation parameters, or the number of trees, ...

  [Pedregosa 2016, Lorraine et al. 2020]

- *Automatic Data Augmentation:* λ are the parameters of data augmentation used to train the model.

  [Cubuk et al. 2019; Rommel et al. 2022]

- *Neural Architecture Search:* λ are the parameter of a Neural Network architecture.

  [Liu et al. 2018, Zhang et al. 2021]

Other bi-level optimization problems: Representation Learning

Generative Adversarial Network: G is the discriminator loss, that classify between generated and natural samples. Then $F = -G$ and one aims to solve [Goodfellow et al. 2014]

$$\max_{\lambda} G(\lambda, \theta^*) \quad s.t. \quad \theta^* = \min_{\theta} G(\lambda, \theta)$$

Here θ are the parameter of the discriminator and λ of the generator.

Dictionary Learning: $F = G$ are the reconstruction loss and one looks for the dictionary D that minimizes [Malezieux et al. 2022]

$$\min_{D} \|X - D\theta^*\| \quad s.t. \quad \theta^* = \operatorname*{argmin}_{\theta} \|X - D\theta\| + \lambda\|\theta\|_1$$

Here, θ^* is a sparse representation of the input sample X.

Other bi-level optimization problems: Implicit Deep Learning

Deep Equilibrium Network: G is a fixed point equation that defines the output of a layer and F is the training loss of the network, [Bai et al. 2019]

$$\max_{\lambda} F(\lambda, \theta^*) \quad s.t. \quad G(\lambda, \theta) = \theta - g(\theta, \lambda) = 0$$

These networks micmic infinite depth network as θ^* can be seen as applying the transfer function g infinitly many times if it is contractive.

Black box methods: Take $\{\lambda_k\}_k$ and compute $\min_k h(\lambda_k)$

- Grid-Search
- Random-Search
- Bayesian-Optimization

Solving bi-level optimization

Black box methods: Take $\{\lambda_k\}_k$ and compute $\min_k h(\lambda_k)$

- Grid-Search
- Random-Search
- Bayesian-Optimization

First order methods: Compute the gradient of h

$$\lambda^{t+1} = \lambda^t - \rho^t \underbrace{\frac{\partial F(\lambda, \theta^*(\lambda))}{\partial \lambda}}_{\nabla h(\lambda)}$$

Solving bi-level optimization

Black box methods: Take $\{\lambda_k\}_k$ and compute $\min_k h(\lambda_k)$

- Grid-Search
- Random-Search
- Bayesian-Optimization

First order methods: Compute the gradient of h

$$\lambda^{t+1} = \lambda^t - \rho^t \underbrace{\frac{\partial F(\lambda, \theta^*(\lambda))}{\partial \lambda}}_{\nabla h(\lambda)}$$

- Can we compute the gradient of h?
- Do we need to compute $\theta^*(\lambda)$?
- How to efficiently approximate $\nabla h(\lambda)$?

Implicit Gradient

Computing the gradient of the value function h

References

- Pedregosa, F. (2016). Hyperparameter optimization with approximate gradient. In *International Conference on Machine Learning (ICML)*, pages 737–746, New-York, NY, USA
- Lorraine, J., Vicol, P., and Duvenaud, D. (2020). Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1540–1552. PMLR
- Ramzi, Z., Mannel, F., Bai, S., Starck, J.-L., Ciuciu, P., and Moreau, T. (2022). SHINE: SHaring the INverse Estimate from the forward pass for bi-level optimization and implicit models. In *International Conference on Learning Representations (ICLR)*, online

Value Function's Gradient

First order methods on h needs to compute the gradient of h.

Chain rule:

$$\nabla_\lambda h(\lambda) = \frac{\partial F}{\partial \lambda}(\lambda, \theta^*(\lambda)) + \frac{\partial F}{\partial \theta}(\lambda, \theta^*(\lambda)) \frac{\partial \theta^*}{\partial \lambda}(\lambda)$$

Value Function's Gradient

First order methods on h needs to compute the gradient of h.

Chain rule:

$$\nabla_\lambda h(\lambda) = \frac{\partial F}{\partial \lambda}(\lambda, \theta^*(\lambda)) + \frac{\partial F}{\partial \theta}(\lambda, \theta^*(\lambda)) \frac{\partial \theta^*}{\partial \lambda}(\lambda)$$

Implicit function Theorem: $\theta^*(\lambda)$ verifies the KKT $\frac{\partial G}{\partial \theta}(\lambda, \theta^*(\lambda)) = 0$,

$$\frac{\partial^2 G}{\partial \theta^2}(\lambda, \theta^*(\lambda)) \frac{\partial \theta^*}{\partial \lambda}(\lambda) + \frac{\partial^2 G}{\partial \theta \partial \lambda}(\lambda, \theta^*(\lambda)) = 0,$$

$$\frac{\partial \theta^*}{\partial \lambda}(\lambda) = - {\frac{\partial^2 G}{\partial \theta^2}}^{-1}(\lambda, \theta^*(\lambda)) \frac{\partial^2 G}{\partial \theta \partial \lambda}(\lambda, \theta^*(\lambda)),$$

Value Function's Gradient

First order methods on h needs to compute the gradient of h.

Chain rule:

$$\nabla_\lambda h(\lambda) = \frac{\partial F}{\partial \lambda}(\lambda, \theta^*(\lambda)) + \frac{\partial F}{\partial \theta}(\lambda, \theta^*(\lambda)) \frac{\partial \theta^*}{\partial \lambda}(\lambda)$$

Implicit function Theorem: $\theta^*(\lambda)$ verifies the KKT $\frac{\partial G}{\partial \theta}(\lambda, \theta^*(\lambda)) = 0$,

$$\frac{\partial^2 G}{\partial \theta^2}(\lambda, \theta^*(\lambda)) \frac{\partial \theta^*}{\partial \lambda}(\lambda) + \frac{\partial^2 G}{\partial \theta \partial \lambda}(\lambda, \theta^*(\lambda)) = 0,$$

$$\frac{\partial \theta^*}{\partial \lambda}(\lambda) = - \frac{\partial^2 G}{\partial \theta^2}^{-1}(\lambda, \theta^*(\lambda)) \frac{\partial^2 G}{\partial \theta \partial \lambda}(\lambda, \theta^*(\lambda)),$$

$$\nabla_\lambda h(\lambda) = \frac{\partial F}{\partial \lambda}(\lambda, \theta^*(\lambda)) - \frac{\partial F}{\partial \theta}(\lambda, \theta^*(\lambda)) \frac{\partial^2 G}{\partial \theta^2}^{-1}(\lambda, \theta^*(\lambda)) \frac{\partial^2 G}{\partial \theta \partial \lambda}(\lambda, \theta^*(\lambda))$$

$\Rightarrow$ Need to compute $\theta^*(\lambda)$ and an inverse hvp.

Do we need to compute them precisely?

Idea: Approximate $\theta^*(\lambda^t)$ and $v^*(\lambda^t) = -\frac{\partial^2 G}{\partial \theta^2}^{-1} \frac{\partial F}{\partial \theta}(\lambda^t, \theta^*(\lambda^t))$

HOAG - Approximating $\theta^*(\lambda^t)$ [Pedregosa 2016]

Do we need to compute them precisely?

Idea: Approximate $\theta^*(\lambda^t)$ and $v^*(\lambda^t) = -\frac{\partial^2 G}{\partial \theta^2}^{-1} \frac{\partial F}{\partial \theta}(\lambda^t, \theta^*(\lambda^t))$

- Compute θ^t such that $\|\theta^t - \theta^*(\lambda^t)\|_2 \leq \epsilon_t$,
  iterative solver *e.g.* L-BFGS
- Compute v^t such that $\|\frac{\partial^2 G}{\partial \theta^2}(\lambda^t, \theta^t)v^t + \frac{\partial F}{\partial \theta}(\lambda^t, \theta^t)\|_2 \leq \epsilon_t$,
  L-BFGS or CG
- Compute the approximate gradient $g_t = \frac{\partial F}{\partial \lambda}(\lambda^t, \theta^t) + \frac{\partial^2 G}{\partial \theta \partial \lambda}(\lambda^t, \theta^t)v^t$
- Update the outer variable $\lambda^{t+1} = \lambda^t - \rho^t g^t$

Do we need to compute them precisely?

Idea: Approximate $\theta^*(\lambda^t)$ and $v^*(\lambda^t) = -\frac{\partial^2 G}{\partial \theta^2}^{-1} \frac{\partial F}{\partial \theta}(\lambda^t, \theta^*(\lambda^t))$

- Compute θ^t such that $\|\theta^t - \theta^*(\lambda^t)\|_2 \leq \epsilon_t$,
  iterative solver *e.g.* L-BFGS
- Compute v^t such that $\|\frac{\partial^2 G}{\partial \theta^2}(\lambda^t, \theta^t)v^t + \frac{\partial F}{\partial \theta}(\lambda^t, \theta^t)\|_2 \leq \epsilon_t$,
  L-BFGS or CG
- Compute the approximate gradient $g_t = \frac{\partial F}{\partial \lambda}(\lambda^t, \theta^t) + \frac{\partial^2 G}{\partial \theta \partial \lambda}(\lambda^t, \theta^t)v^t$
- Update the outer variable $\lambda^{t+1} = \lambda^t - \rho^t g^t$

Theorem: If $\sum_t \epsilon_t < \infty$ and the step are chosen appropriatly, then the algorithm converges to a stationary point *i.e.*

$$\|\nabla h(\lambda^t)\|_2 \to 0 \ .$$

Further approximation of the Inverse Hvp

Solving the linear system for $v^*(\lambda^t)$,

- Core idea is to not inverse the hessian $\frac{\partial^2 G}{\partial \theta^2}(\lambda^t, \theta^t)$,

  We are only interested in one direction.

- Only rely on Hessian-vector product (Hvp).

  Can be computed efficiently

Proposed Methods:

- L-BFGS
- Conjugate Gradient
- Jacobian-Free method

$$\frac{\partial^2 G}{\partial \theta^2}(\lambda^t, \theta^t) \approx Id$$

- Neumann iterations

$$\frac{\partial^2 G}{\partial \theta^2}(\lambda^t, \theta^t)^{-1} \approx \sum_k (Id - \frac{\partial^2 G}{\partial \theta^2}(\lambda^t, \theta^t))^k$$

[Pedregosa 2016, Lorraine et al. 2020, Luketina et al. 2016]

Idea: reuse the approximation of the Hessian computed by L-BFGS for the inner problem.

Idea: reuse the approximation of the Hessian computed by L-BFGS for the inner problem.

Quasi Newton 101:

Solving $\text{argmin}_x f(x)$

Newton Method

$$x^{t+1} = x^t - \frac{\partial^2 f}{\partial x^2}(x^t)^{-1}\frac{\partial f}{\partial x}(x^t)$$

Quasi-Newton Method

$$x^{t+1} = x^t - B_t^{-1}\frac{\partial f}{\partial x}(x^t)$$

B_n: low-rank approx. of the Hessian.

Inverse with Sherman-Morrison

Idea: reuse the approximation of the Hessian computed by L-BFGS for the inner problem.

Quasi Newton 101:

Solving $\text{argmin}_x f(x)$

Newton Method

$$x^{t+1} = x^t - \frac{\partial^2 f}{\partial x^2}(x^t)^{-1}\frac{\partial f}{\partial x}(x^t)$$

Quasi-Newton Method

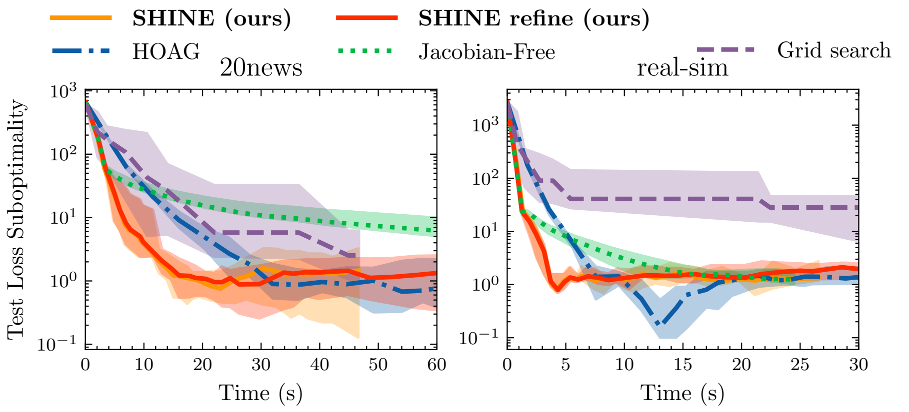
$$x^{t+1} = x^t - B_t^{-1}\frac{\partial f}{\partial x}(x^t)$$

B_n: low-rank approx. of the Hessian.

Inverse with Sherman-Morrison

$\Rightarrow$ Use B_n^{-1} as the inverse of the Hessian for v^*

Logistic Regression with ℓ_2-regularisation on 2 datasets:



$\Rightarrow$ Theoretically grounded, can be further refined, large scale experiments on DEQs.

Algorithm Unrolling

Differentiable inner problem solvers

References

- Shaban, A., Cheng, C.-A., Hatch, N., and Boots, B. (2019). Truncated Back-propagation for Bilevel Optimization. In *Artificial Intelligence and Statistics (AISTAT)*, pages 1723–1732, Okinawa, Japan
- Ablin, P., Peyré, G., and Moreau, T. (2020). Super-efficiency of automatic differentiation for functions defined as a minimum. In *International Conference on Machine Learning (ICML)*
- Malézieux, B., Moreau, T., and Kowalski, M. (2022). Understanding approximate and Unrolled Dictionary Learning for Pattern Recovery. In *International Conference on Learning Representations (ICLR)*, online

Differentiable unrolling of θ^t

Idea: Compute $\frac{\partial \theta^{\boldsymbol{t}}}{\partial \lambda}(\lambda) \approx \frac{\partial \theta^*}{\partial \lambda}(\lambda)$ using automatic differentiation through an iterative algorithm.

Differentiable unrolling of θ^t

Idea: Compute $\frac{\partial \theta^t}{\partial \lambda}(\lambda) \approx \frac{\partial \theta^*}{\partial \lambda}(\lambda)$ using automatic differentiation through an iterative algorithm.

For the gradient descent algorithm:

$$\theta^{t+1} = \theta^t - \rho \frac{\partial G}{\partial \theta}(\lambda, \theta^t)$$

The Jacobian reads,

$$\frac{\partial \theta^{t+1}}{\partial \lambda}(\lambda) = \left(Id - \rho \frac{\partial^2 G}{\partial \theta^2}(\lambda, \theta^t) \right) \frac{\partial \theta^t}{\partial \lambda}(\lambda) - \rho \frac{\partial^2 G}{\partial \theta \partial \lambda}(\lambda, \theta^t)$$

Differentiable unrolling of θ^t

Idea: Compute $\frac{\partial \theta^t}{\partial \lambda}(\lambda) \approx \frac{\partial \theta^*}{\partial \lambda}(\lambda)$ using automatic differentiation through an iterative algorithm.

For the gradient descent algorithm:

$$\theta^{t+1} = \theta^t - \rho \frac{\partial G}{\partial \theta}(\lambda, \theta^t)$$

The Jacobian reads,

$$\frac{\partial \theta^{t+1}}{\partial \lambda}(\lambda) = \left(Id - \rho \frac{\partial^2 G}{\partial \theta^2}(\lambda, \theta^t) \right) \frac{\partial \theta^t}{\partial \lambda}(\lambda) - \rho \frac{\partial^2 G}{\partial \theta \partial \lambda}(\lambda, \theta^t)$$

$\Rightarrow$ Under smoothness conditions, if θ^t converges to θ^*, this converges toward $\frac{\partial \theta^*}{\partial \lambda}(\lambda)$

Context: min-min problems where $F = G$

$$\Rightarrow \text{Here, } \frac{\partial F}{\partial \theta}(\lambda, \theta^*) = 0$$

Context: min-min problems where $F = G$

$$\Rightarrow \text{Here, } \frac{\partial F}{\partial \theta}(\lambda, \theta^*) = 0$$

We consider the 3 gradient estimates:

- $g_1 = \frac{\partial G}{\partial \lambda}(\lambda, \theta^t)$ — Analysis
- $g_2 = \frac{\partial G}{\partial \lambda}(\lambda, \theta^t) + \frac{\partial G}{\partial \theta}(\lambda, \theta^t)\frac{\partial \theta^t}{\partial \lambda}$ — Automatic
- $g_3 = \frac{\partial G}{\partial \lambda}(\lambda, \theta^t) - \frac{\partial G}{\partial \theta}(\lambda, \theta^t)\frac{\partial^2 G}{\partial \theta^2}^{-1}(\lambda, \theta^t)\frac{\partial^2 G}{\partial \theta \partial \lambda}(\lambda, \theta^t)$ — Implicit

Context: min-min problems where $F = G$

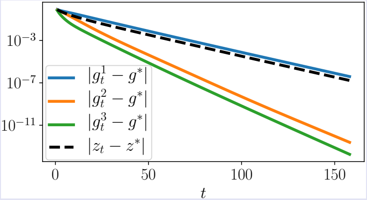
$$\Rightarrow \text{Here, } \frac{\partial F}{\partial \theta}(\lambda, \theta^*) = 0$$

We consider the 3 gradient estimates:

- $g_1 = \frac{\partial G}{\partial \lambda}(\lambda, \theta^t)$ Analysis
- $g_2 = \frac{\partial G}{\partial \lambda}(\lambda, \theta^t) + \frac{\partial G}{\partial \theta}(\lambda, \theta^t)\frac{\partial \theta^t}{\partial \lambda}$ Automatic
- $g_3 = \frac{\partial G}{\partial \lambda}(\lambda, \theta^t) - \frac{\partial G}{\partial \theta}(\lambda, \theta^t)\frac{\partial^2 G}{\partial \theta^2}^{-1}(\lambda, \theta^t)\frac{\partial^2 G}{\partial \theta \partial \lambda}(\lambda, \theta^t)$ Implicit

Convergence rates: For G strongly convex in θ,

$$|g_t^1(x) - g^*(x)| = O\left(|\theta^t(\lambda) - \theta^*(\lambda)|\right),$$
$$|g_t^2(x) - g^*(x)| = o\left(|\theta^t(\lambda) - \theta^*(\lambda)|\right),$$
$$|g_t^3(x) - g^*(x)| = O\left(|\theta^t(\lambda) - \theta^*(\lambda)|^2\right).$$



Analysis for non-smooth min-min problems [Malezieux et al. 2022]

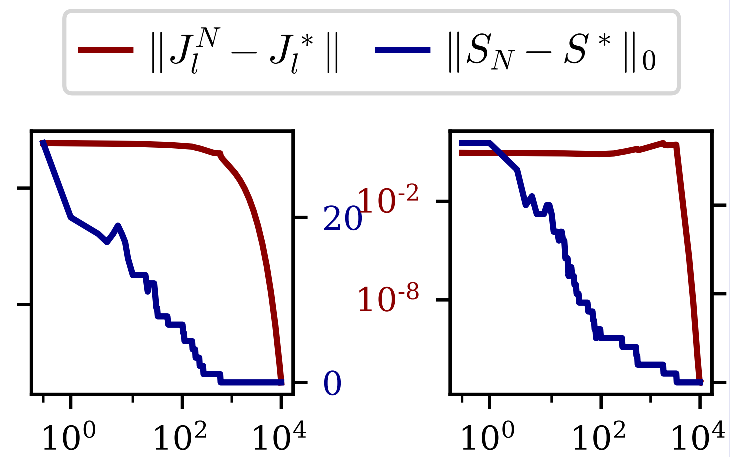
Context: dictionary learning, $F = G$ with an ℓ_1-regularization for θ.

Issue: The implicit gradient quality mostly depends on the support identifiaction,

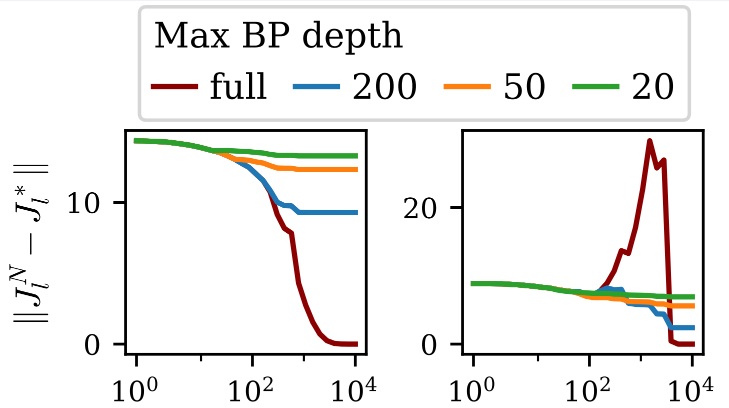
$$\left(\frac{\partial \theta^*}{\partial D_l}\right)_{S^*} = -(D_{:,S^*}^\top D_{:,S^*})^{-1}(D_l \theta^{*\top} + (D_l^\top \theta^* - y_l) Id_n)_{S^*} \ ,$$

$\Rightarrow$ Is the autodiff approach better than the analytic one?

On the support, the function is smooth and we recover the same convergence.



Outside of the support, errors can accumulate and the gradient can blow up.



Conclusion

- Bi-level optimization is intrinsic in many ML problems.
- Classical optimization method can be used once we know how to compute the gradient.
- The gradient can be computed either using implicit function theorem or algorithm unrolling.
- No clear winner, this depends on the problem at end!

Current work:

- Efficient and stochastic bi-level solvers,
- Application of bi-level solvers to Data-Augmentation problems for EEG.

$\Rightarrow$ Stay tuned!

Slides will be on my web page:

tommoral.github.io @tomamoral

Thanks to all my bi-level collaborators!