

Optimizing the Optimizer: Bilevel Learning and Hypergradient Computation

Thomas Moreau
INRIA Saclay - MIND Team



What is Bilevel optimization?

You can follow this intro with a complementary notebook:

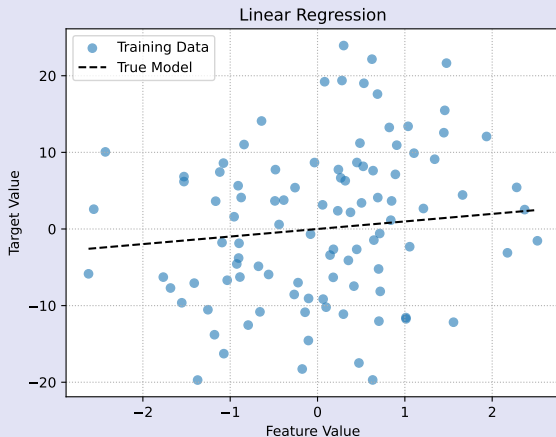
<https://tinyurl.com/bilevel-hpo>



Learning a linear ML model

Setup:

- ▶ Regression task $(X_i, y_i)_{i=1}^N \in \mathbb{R}^p \times \mathbb{R}$
- ▶ Linear model: predict y from X with $\langle \theta, X \rangle$.



Empirical risk minimization with ℓ_2 loss:

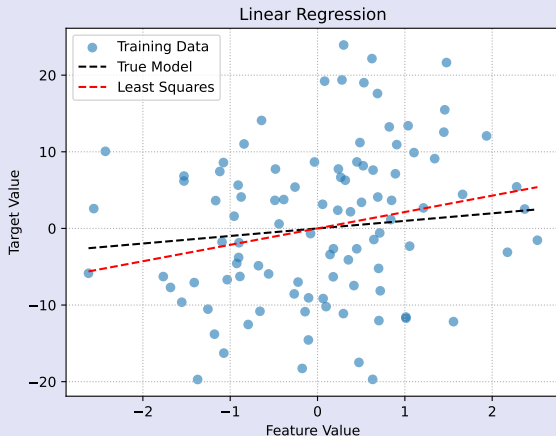
$$G(\theta) = \frac{1}{N} \sum_{i=1}^N (y_i - \langle \theta, X_i \rangle)^2$$

Training the model:

$$\theta^* = \operatorname{argmin}_{\theta} G(\theta)$$

Avoiding overfitting

Here, we don't have many samples and there is noise. The model we learn is slightly off



Avoiding overfitting with a regularization

Regularized ERM with ℓ_2 loss:

$$G(\theta, \lambda) = \frac{1}{N} \sum_{i=1}^N (y_i - \langle \theta, X_i \rangle)^2 + \lambda \|\theta\|_2^2$$

Training the model:

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

Avoiding overfitting with a regularization

Regularized ERM with ℓ_2 loss:

$$G(\theta, \lambda) = \frac{1}{N} \sum_{i=1}^N (y_i - \langle \theta, X_i \rangle)^2 + \lambda \|\theta\|_2^2$$

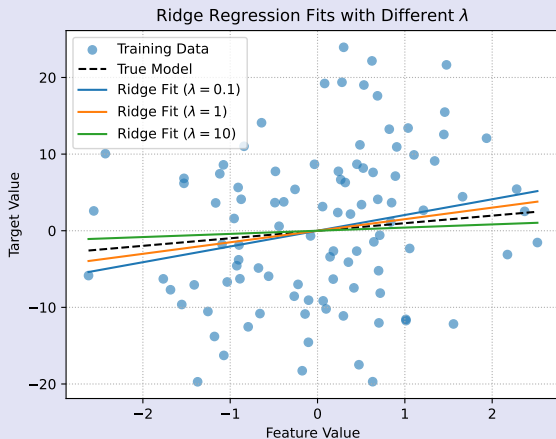
Training the model:

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

\Rightarrow How to choose λ ?

Fitting with regularization

For each regularisation we get a different solution



Evaluating the generalization

We want to find λ that ensure the best *generalization* of $\theta^*(\lambda)$.

Validation loss: use held out data $(X_i^{val}, y_i^{val})_{i=1}^M$

$$F(\theta) = \frac{1}{M} \sum_{i=1}^M (y_i^{val} - \langle \theta, X_i^{val} \rangle)^2$$

Independent estimate of the risk of the model.

Evaluating the generalization

We want to find λ that ensure the best *generalization* of $\theta^*(\lambda)$.

Validation loss: use held out data $(X_i^{val}, y_i^{val})_{i=1}^M$

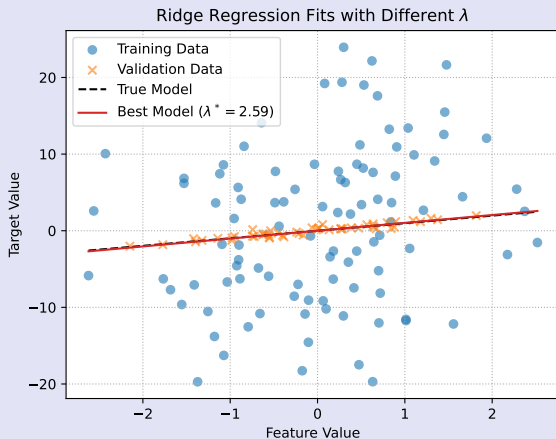
$$F(\theta) = \frac{1}{M} \sum_{i=1}^M (y_i^{val} - \langle \theta, X_i^{val} \rangle)^2$$

Independent estimate of the risk of the model.

\Rightarrow Find λ that gives a model $\theta^*(\lambda)$ with a good validation loss.

Evaluating the generalization

Choose λ that gives the best validation error



The Grid Search

- ▶ Select a grid of parameters $\{\lambda_1, \dots, \lambda_K\}$.
- ▶ Train a model for each parameter λ_k : $\theta^*(\lambda_k)$.
- ▶ Evaluate the performance with the validation loss $F(\theta^*(\lambda_k))$.
- ▶ Keep the value λ_k with the best performance.

The Grid Search

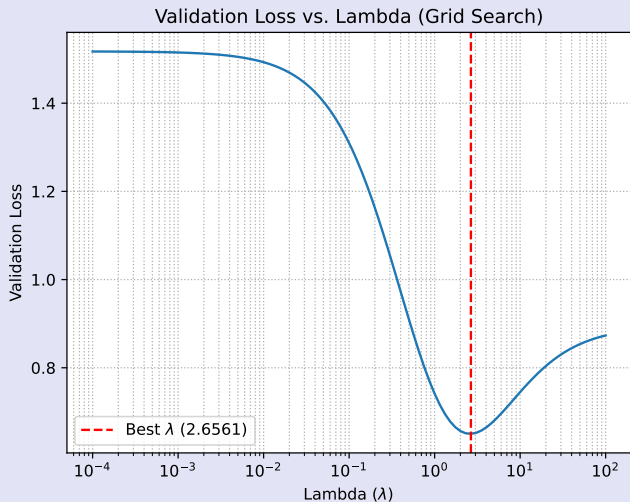
- ▶ Select a grid of parameters $\{\lambda_1, \dots, \lambda_K\}$.
- ▶ Train a model for each parameter λ_k : $\theta^*(\lambda_k)$.
- ▶ Evaluate the performance with the validation loss $F(\theta^*(\lambda_k))$.
- ▶ Keep the value λ_k with the best performance.

Mathematical rewriting:

$$\begin{cases} \min_{\lambda \in \{\lambda_1, \dots, \lambda_K\}} F(\theta^*(\lambda)) \\ \text{s.t. } \theta^*(\lambda) = \operatorname{argmin}_{\theta} G(\theta, \lambda) \end{cases}$$

The Grid Search illustration

Achieve a trade-off between overfitting and over-simple model



Regularized ERM with ℓ_2 loss:

$$G(\theta, \lambda) = \frac{1}{N} \sum_{i=1}^N (y_i - \langle \theta, X_i \rangle)^2 + \lambda \|\theta\|_2^2$$

Regularized ERM with ℓ_2 loss:

$$G(\theta, \lambda) = \frac{1}{N} \sum_{i=1}^N (y_i - \langle \theta, X_i \rangle)^2 + \lambda \|\theta\|_2^2$$

- ▶ Grid search precision is uniform: many points are “wasted” in non-informative zones.

Regularized ERM with ℓ_2 loss:

$$G(\theta, \lambda) = \frac{1}{N} \sum_{i=1}^N (y_i - \langle \theta, X_i \rangle)^2 + \sum_{k=1}^p \lambda_k \theta_k^2$$

- ▶ Grid search precision is uniform: many points are “wasted” in non-informative zones.
- ▶ Grid search is inefficient in high dimension as the grid size grows exponentially with the number of parameters.

Regularized ERM with ℓ_2 loss:

$$G(\theta, \lambda) = \frac{1}{N} \sum_{i=1}^N (y_i - \langle \theta, X_i \rangle)^2 + \sum_{k=1}^p \lambda_k \theta_k^2$$

- ▶ Grid search precision is uniform: many points are “wasted” in non-informative zones.
- ▶ Grid search is inefficient in high dimension as the grid size grows exponentially with the number of parameters.

⇒ Can we use first-order methods to minimize $h(\lambda) = F(\theta^*(\lambda))$?

Bi-level optimization

Bi-level problem: Optimization problem with two levels

$$\begin{array}{ll} \min_{\lambda} & h(\lambda) = F(\lambda, \theta^*(\lambda)) \quad \leftarrow \text{Outer function} \\ \text{s.t.} & \theta^*(\lambda) = \operatorname{argmin}_{\theta} G(\lambda, \theta) \end{array}$$

Value function \nearrow

\uparrow *Inner function/Problem*

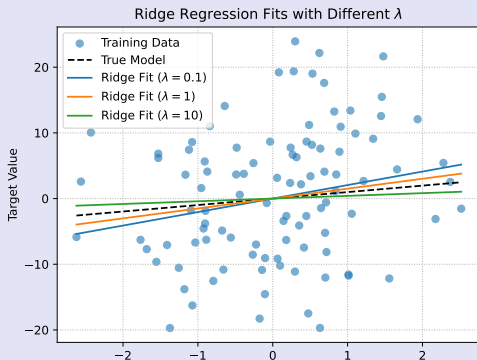
Goal: Optimize the value function h whose value depends on the result of another optimization problem.

Bi-level optimization problems: Model selection

Selecting the best model:

- ▶ G is the training loss and θ are the parameters of the model.
- ▶ Select the hyper-parameter λ to get the best validation loss F .

Hyperparameter optimization: λ is a regularization parameter:

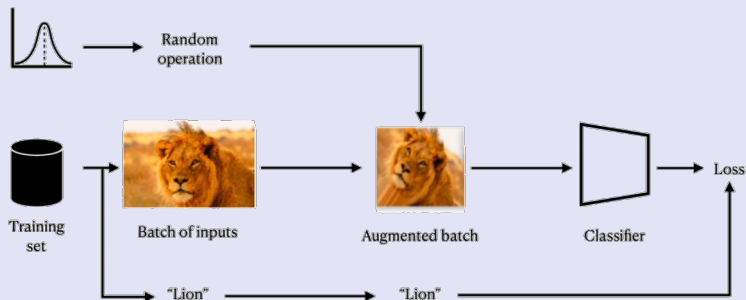


Bi-level optimization problems: Model selection

Selecting the best model:

- ▶ G is the training loss and θ are the parameters of the model.
- ▶ Select the hyper-parameter λ to get the best validation loss F .

Data augmentation: λ parametrizes the transformations distribution.

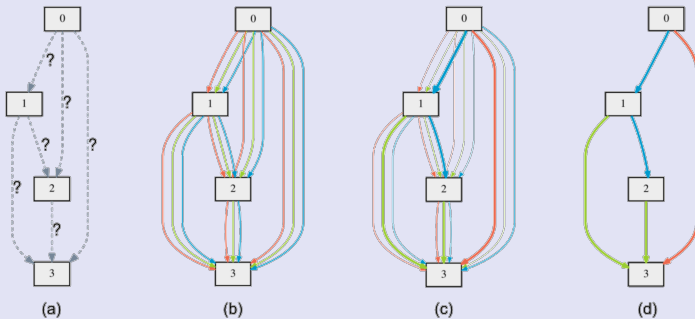


Bi-level optimization problems: Model selection

Selecting the best model:

- ▶ G is the training loss and θ are the parameters of the model.
- ▶ Select the hyper-parameter λ to get the best validation loss F .

Neural Architecture Search: λ parametrizes the architecture.



Bi-level optimization problems: Implicit Deep Learning

Deep Equilibrium Network:

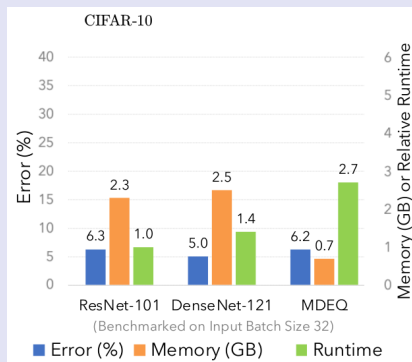
$$\begin{cases} \min_{\lambda} h(\lambda) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y_i, \theta^*(X_i, \lambda)) \\ \text{s.t. } \theta^*(X_i, \lambda) = g_{\lambda}(\theta^*(X_i, \lambda)) \end{cases}$$

Output of the network is the root of $G(\theta, \lambda) = \theta - g_{\lambda}(\theta) = 0$.

- ▶ Mimic infinite depth:

$$\theta^{(t+1)} = g_{\lambda}(\theta^{(t)}) \quad t \rightarrow \infty .$$

- ▶ Efficient memory
- ▶ Slow runtime



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

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

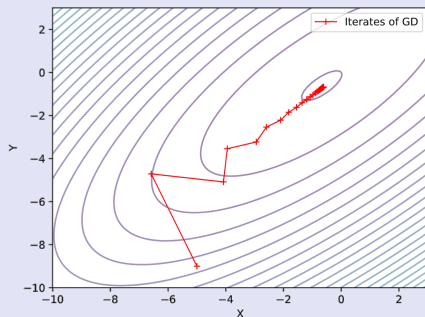
\Rightarrow Do not scale well with the dimension

First order methods: Gradient descent on h

Iterate in the steepest direction:

$$\lambda^{t+1} = \lambda^t - \rho^t \nabla h(\lambda)$$

- ▶ Gradient $\nabla h(\lambda) = \frac{d F(\lambda, \theta^*(\lambda))}{d \lambda}$
- ▶ Step size ρ^t .



\Rightarrow How to compute the hyper-gradient $\nabla h(\lambda)$?

Implicit Gradient

Computing the gradient of the value function h

References

- ▶ Pedregosa, F. (2016). [Hyperparameter optimization with approximate gradient](#). In *ICML*
- ▶ Lorraine, J., Vicol, P., and Duvenaud, D. (2020). [Optimizing millions of hyperparameters by implicit differentiation](#). In *AISTATS*

Value function definition:

$$h(\lambda) = F(\lambda, \theta^*(\lambda))$$

Chain rule:

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

Optimality condition for θ^*

$$\nabla_{\theta} G(\lambda, \theta^*(\lambda)) = 0$$

Optimality condition for θ^*

$$\nabla_{\theta} G(\lambda, \theta^*(\lambda)) = 0$$

Derivating this equation relative to λ gives:

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

Optimality condition for θ^*

$$\nabla_{\theta} G(\lambda, \theta^*(\lambda)) = 0$$

Derivating this equation relative to λ gives:

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

Implicit function theorem

$$d\theta^*(\lambda) = -[\nabla_{\theta\theta}^2 G(\lambda, \theta^*(\lambda))]^{-1} \nabla_{\theta\lambda}^2 G(\lambda, \theta^*(\lambda)),$$

Value function gradient:

$$\nabla h(\lambda) = \nabla_{\lambda} F(\lambda, \theta^*) - \nabla_{\theta\lambda}^2 G(\lambda, \theta^*) [\nabla_{\theta\theta}^2 G(\lambda, \theta^*)]^{-1} \nabla_{\theta} F(\lambda, \theta^*)$$

Value function gradient:

$$\nabla h(\lambda) = \nabla_{\lambda} F(\lambda, \theta^*) - \nabla_{\theta\lambda}^2 G(\lambda, \theta^*) [\nabla_{\theta\theta}^2 G(\lambda, \theta^*)]^{-1} \nabla_{\theta} F(\lambda, \theta^*)$$

- Need to compute the solution of the inner

Value function gradient:

$$\nabla h(\lambda) = \nabla_{\lambda} F(\lambda, \theta^*) - \nabla_{\theta\lambda}^2 G(\lambda, \theta^*) [\nabla_{\theta\theta}^2 G(\lambda, \theta^*)]^{-1} \nabla_{\theta} F(\lambda, \theta^*)$$

- ▶ Need to compute the solution of the inner
- ▶ Need to solve a $p \times p$ linear system

$$v^*(\lambda) = [\nabla_{\theta\theta}^2 G(\lambda, \theta^*)]^{-1} \nabla_{\theta} F(\lambda, \theta^*)$$

Computing the Hessian matrix and inverting it is prohibitive for large p .

In practice, we don't need to compute the full Hessian inverse, only the product with a vector $\nabla_{\theta} F(\lambda, \theta^*)$.

[inverse HVP]

And this can be done using an iterative method that only requires Hessian-vector products (HVP)!

[Conjugate Gradient]

Computing the Hessian matrix and inverting it is prohibitive for large p .

In practice, we don't need to compute the full Hessian inverse, only the product with a vector $\nabla_{\theta} F(\lambda, \theta^*)$.

[inverse HVP]

And this can be done using an iterative method that only requires Hessian-vector products (HVP)!

[Conjugate Gradient]

\Rightarrow HVP can be computed efficiently using Pearlmutter's trick

[Pearlmutter 1994]

Use automatic differentiation over automatic differentiation.

A quick detour by automatic differentiation

Automatic differentiation (AD) allows to compute derivatives of functions which is the composition of elementary operations implemented as a computer program.

$$\underbrace{\frac{\partial f}{\partial \theta}(\theta)}_{p \times d} = \frac{\partial z_n}{\partial \theta} = \frac{\partial z_n}{\partial z_1} \frac{\partial z_1}{\partial \theta} = \dots = \underbrace{\frac{\partial z_n}{\partial z_{n-1}}}_{p \times m_{n-1}} \underbrace{\frac{\partial z_{n-1}}{\partial z_{n-2}}}_{m_{n-1} \times m_{n-2}} \dots \underbrace{\frac{\partial z_1}{\partial \theta}}_{m_1 \times d} .$$

A quick detour by automatic differentiation

Automatic differentiation (AD) allows to compute derivatives of functions which is the composition of elementary operations implemented as a computer program.

$$\underbrace{\frac{\partial f}{\partial \theta}(\theta)}_{p \times d} = \frac{\partial z_n}{\partial \theta} = \frac{\partial z_n}{\partial z_1} \frac{\partial z_1}{\partial \theta} = \dots = \underbrace{\frac{\partial z_n}{\partial z_{n-1}}}_{p \times m_{n-1}} \underbrace{\frac{\partial z_{n-1}}{\partial z_{n-2}}}_{m_{n-1} \times m_{n-2}} \dots \underbrace{\frac{\partial z_1}{\partial \theta}}_{m_1 \times d}.$$

Forward mode AD: compute the Jacobian-vector product (JVP) with a vector $v \in \mathbb{R}^d$:

$$\frac{\partial f}{\partial \theta}(\theta) \times v = \frac{\partial z_n}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \dots \frac{\partial z_1}{\partial \theta} v.$$

A quick detour by automatic differentiation

Automatic differentiation (AD) allows to compute derivatives of functions which is the composition of elementary operations implemented as a computer program.

$$\underbrace{\frac{\partial f}{\partial \theta}(\theta)}_{p \times d} = \frac{\partial z_n}{\partial \theta} = \frac{\partial z_n}{\partial z_1} \frac{\partial z_1}{\partial \theta} = \dots = \underbrace{\frac{\partial z_n}{\partial z_{n-1}}}_{p \times m_{n-1}} \underbrace{\frac{\partial z_{n-1}}{\partial z_{n-2}}}_{m_{n-1} \times m_{n-2}} \dots \underbrace{\frac{\partial z_1}{\partial \theta}}_{m_1 \times d} .$$

Forward mode AD: compute the Jacobian-vector product (JVP) with a vector $v \in \mathbb{R}^d$:

$$\frac{\partial f}{\partial \theta}(\theta) \times v = \frac{\partial z_n}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \dots \frac{\partial z_1}{\partial \theta} v .$$

Reverse mode AD: compute the vector-Jacobian product (VJP) with a vector $u \in \mathbb{R}^p$:

$$u^\top \frac{\partial f}{\partial \theta}(\theta) = u^\top \frac{\partial z_n}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \dots \frac{\partial z_1}{\partial \theta} .$$

A quick detour by automatic differentiation

Automatic differentiation (AD) allows to compute derivatives of functions which is the composition of elementary operations implemented as a computer program.

$$\underbrace{\frac{\partial f}{\partial \theta}}_{p \times d}(\theta) = \frac{\partial z_n}{\partial \theta} \frac{\partial z_n}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \dots \frac{\partial z_1}{\partial \theta}.$$

For more details on AD, see

*The Elements of Differentiable
Programming*

by V. Roulet & M. Blondel

Forward mode
vector $v \in \mathbb{R}^p$

) with a

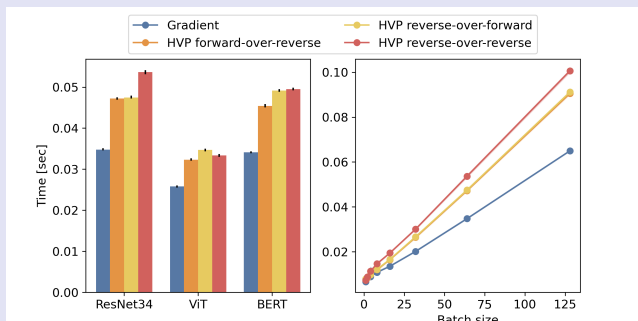
Reverse mode AD: compute the vector-Jacobian product (VJP) with a vector $u \in \mathbb{R}^p$:

$$u^\top \frac{\partial f}{\partial \theta}(\theta) = u^\top \frac{\partial z_n}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \dots \frac{\partial z_1}{\partial \theta}.$$

Computing a HVP is a simple differentiation over a function that involves a JVP:

$$\nabla^2 f(\theta)v = \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} [\nabla f(\theta + \epsilon v) - \nabla f(\theta)] = \nabla[\langle \nabla f(.), v \rangle](\theta) .$$

Can be computed using AD, with reverse over reverse, forward over reverse or reverse over forward mode.



<https://tinyurl.com/bilevel-hvp>



Hyperparameter optimization with Approximate Gradient HOAG

[Pedregosa 2016]

Do we need to compute θ^ and v^* precisely?*

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

Hyperparameter optimization with Approximate Gradient HOAG

[Pedregosa 2016]

Do we need to compute θ^ and v^* precisely?*

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

- ▶ 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: (*informal*) If $\sum_t \epsilon_t < \infty$ and the step-sizes are chosen appropriately, then the algorithm converges to a stationary point *i.e.*

$$\|\nabla h(\lambda^t)\|_2 \rightarrow 0 \text{ .}$$

Further linear system approximation v^*

Linear system solution $v^*(\lambda^t)$ is a by product.

\Rightarrow Avoid computing it as much as possible.

Proposed Methods:

► L-BFGS

► Conjugate Gradient

► Jacobian-Free method

► Neumann iterations

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

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

► SHINE

[Ramzi et al. 2022]

► Algorithm unrolling

[Shaban et al. 2019]

Jacobian estimation 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*

Differentiable unrolling of θ^N

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

Differentiable unrolling of θ^N

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

For the gradient descent algorithm:

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

The Jacobian reads,

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

Differentiable unrolling of θ^N

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

For the gradient descent algorithm:

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

The Jacobian reads,

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

\Rightarrow Under smoothness conditions, if θ^N converges to θ^* , this converges toward $\frac{\partial \theta^*}{\partial \lambda}(\lambda) = \frac{\partial^2 F}{\partial \theta^2}(\lambda, \theta^*)^{-1} \frac{\partial^2 F}{\partial \mathbf{x} \partial \lambda}(\lambda, \theta^*)$

We consider the 3 gradient estimates:

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

We consider the 3 gradient estimates:

► $g_1^N = \nabla_{\lambda} F(\lambda, \theta^N)$

Analysis

► $g_2^N = \nabla_{\lambda} F(\lambda, \theta^N) + \frac{\partial \theta^N}{\partial \lambda}^{\top} \nabla_{\mathbf{x}} F(\lambda, \theta^N)$

Automatic

► $g_3^N = \nabla_{\lambda} F(\lambda, \theta^N) - \frac{\partial^2 G}{\partial \mathbf{x} \partial \lambda}(\lambda, \theta^N) \frac{\partial^2 G}{\partial \theta^2}^{-1}(\lambda, \theta^N) \nabla_z F(\lambda, \theta^N)$

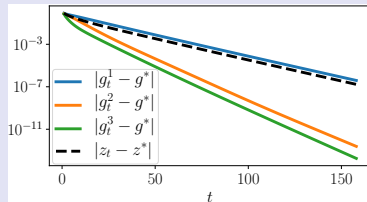
Implicit

Convergence rates: For G strongly convex in z ,

$$|g_1^N(x) - g^*(x)| = O\left(|\theta^N(\lambda) - \theta^*(\lambda)|\right),$$

$$|g_t^N(x) - g^*(x)| = o\left(|\theta^N(\lambda) - \theta^*(\lambda)|\right),$$

$$|g_3^N(x) - g^*(x)| = O\left(|\theta^N(\lambda) - \theta^*(\lambda)|^2\right).$$



What about non-smooth problem?

Very common in inverse problem.

What about non-smooth problem?

Very common in inverse problem.

\Rightarrow Here, we consider the case of the Lasso:

$$\theta^* = \operatorname{argmin}_{\theta} \|\mathbf{y} - \mathbf{A}D\theta\|_2^2 + \lambda \|\theta\|_1$$

with $\lambda = D$

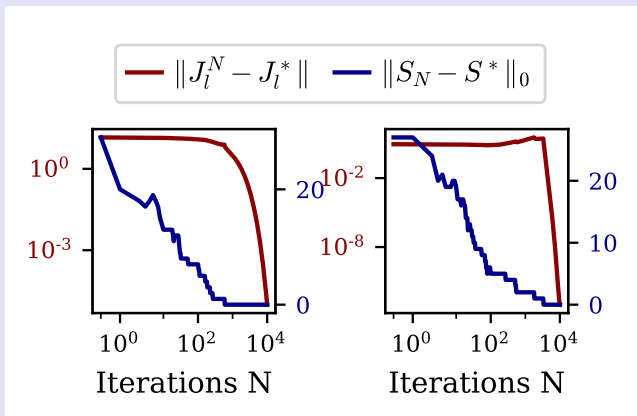
Convergence of the Jacobian

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

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

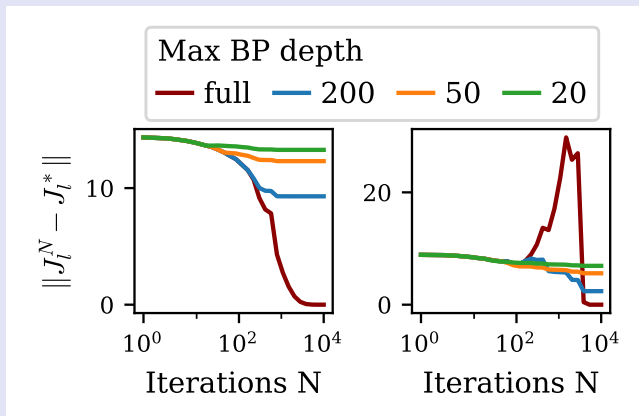
- ▶ On the support, the jacobian converges linearly
- ▶ Before reaching the support, B_N is an error term that can accumulate
- ▶ B_N can be attenuated with truncated back-propagation

Empirical evaluation



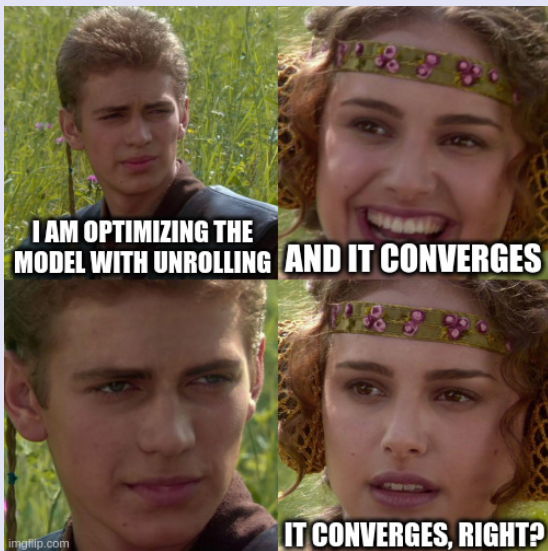
- Linear convergence once the support S^* is reached
- Possible explosion before reaching S^*

Empirical evaluation



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

Learning with unrolled models



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:

$$\min_{\lambda} \mathcal{L}(\theta^N(\lambda))$$

with $\theta^N(\lambda)$ an unrolled algorithm with N steps

- ▶ But we are not optimizing for θ^*

⇒ We are not independent of how we obtain θ^N

[Ramzi et al. 2023]

Stochastic Bi-level Optimization

A framework for linear updates

References

- ▶ Dagr  ou, M., Ablin, P., Vaiter, S., and **TM** (2022). [A framework for bilevel optimization that enables stochastic and global variance reduction algorithms.](#) In *NeurIPS*
- ▶ Dagr  ou, M., Ablin, P., Vaiter, S., and **TM** (2024). [How to compute Hessian-vector products?](#) In *The Third Blogpost Track at ICLR 2024*

Classical ML setting:

$$F(\lambda, \theta) = \frac{1}{m} \sum_{j=1}^m F_j(\lambda, \theta), \quad G(\lambda, \theta) = \frac{1}{n} \sum_{i=1}^n G_i(\lambda, \theta)$$

Classical ML setting:

$$F(\lambda, \theta) = \frac{1}{m} \sum_{j=1}^m F_j(\lambda, \theta), \quad G(\lambda, \theta) = \frac{1}{n} \sum_{i=1}^n G_i(\lambda, \theta)$$

Consequence: For large m and n , any single derivative is cumbersome to compute.

Single level problem:

$$\min_{\theta} f(\theta) = \frac{1}{n} \sum_{i=1}^n f_i(\theta)$$

Single level problem:

$$\min_{\theta} f(\theta) = \frac{1}{n} \sum_{i=1}^n f_i(\theta)$$

First order stochastic optimization:

$$\theta^{t+1} = \theta^t - \rho^t g^t, \quad \mathbb{E}[g^t | \theta^t] = \nabla f(\theta^t)$$

Single level problem:

$$\min_{\theta} f(\theta) = \frac{1}{n} \sum_{i=1}^n f_i(\theta)$$

First order stochastic optimization:

$$\theta^{t+1} = \theta^t - \rho^t g^t, \quad \mathbb{E}[g^t | \theta^t] = \nabla f(\theta^t)$$

Example: stochastic gradient descent [Robbins and Monroe 1951] :

$$\theta^{t+1} = \theta^t - \rho^t \nabla f_i(\theta^t), \quad i \sim \mathcal{U}(\{1, \dots, n\})$$

Bilevel optimization case

$$F(\lambda, \theta) = \frac{1}{m} \sum_{j=1}^m F_j(\lambda, \theta), \quad G(\lambda, \theta) = \frac{1}{n} \sum_{i=1}^n G_i(\lambda, \theta)$$

Bilevel optimization case

$$F(\lambda, \theta) = \frac{1}{m} \sum_{j=1}^m F_j(\lambda, \theta), \quad G(\lambda, \theta) = \frac{1}{n} \sum_{i=1}^n G_i(\lambda, \theta)$$

$$\nabla h(\lambda) = \nabla_1 F(\lambda, \theta^*(\lambda)) - \nabla_{12}^2 G(\lambda, \theta^*(\lambda)) [\nabla_{22}^2 G(\lambda, \theta^*(\lambda))]^{-1} \nabla_2 F(\lambda, \theta^*(\lambda))$$

Bilevel optimization case

$$F(\lambda, \theta) = \frac{1}{m} \sum_{j=1}^m F_j(\lambda, \theta), \quad G(\lambda, \theta) = \frac{1}{n} \sum_{i=1}^n G_i(\lambda, \theta)$$

$$\nabla h(\lambda) = \nabla_1 F(\lambda, \theta^*(\lambda)) - \nabla_{12}^2 G(\lambda, \theta^*(\lambda)) [\nabla_{22}^2 G(\lambda, \theta^*(\lambda))]^{-1} \nabla_2 F(\lambda, \theta^*(\lambda))$$

Problem:

$$\left[\sum_{i=1}^n \nabla_{22}^2 G_i(\lambda, \theta^*(\lambda)) \right]^{-1} \neq \sum_{i=1}^n [\nabla_{22}^2 G_i(\lambda, \theta^*(\lambda))]^{-1}$$

General algorithm

1 **for** $t = 1, \dots, T$ **do**

1. Take for θ^t an approximation of $\theta^*(\lambda^t)$

2. Take for v^t an approximation of $[\nabla_{22}^2 G(\lambda^t, \theta^t)]^{-1} \nabla_2 F(\lambda^t, \theta^t)$

3. Set

$$p^t = \underbrace{\nabla_1 F(\lambda^t, \theta^t) - \nabla_{12}^2 G(\lambda^t, \theta^t) v^t}_{\approx \nabla h(\lambda^t)}$$

4. Update the outer variable

$$\lambda^{t+1} = \lambda^t - \gamma^t p^t$$

Two loops algorithms

Two loops [[Ghadimi et al. 2018](#)]: $\theta^*(\lambda^t)$ is approximated by output of K steps of SGD:

$$\theta^{t,k+1} = \theta^{t,k} - \rho^t \nabla_2 G_i(\lambda^t, \theta^{t,k})$$

Warm start strategy [[Ji et al. 2021](#), [Arbel and Mairal 2022](#)]: Initialize the inner SGD by the previous iterate θ^{t-1} .

What about the linear system?

Approximate $v^t = [\nabla_{22}^2 G(\lambda^t, \theta^t)]^{-1} \nabla_2 F(\lambda^t, \theta^t)$ with:

- Neumann approximations [[Ghadimi et al. 2018](#), [Ji et al. 2021](#)]:

$$v^t \approx \eta \sum_{q=0}^Q \prod_{k=0}^q (I - \eta \nabla_{22}^2 G_{i_k}(\lambda^t, \theta^t)) \nabla_1 F_j(\lambda^t, \theta^t)$$

What about the linear system?

Approximate $v^t = [\nabla_{22}^2 G(\lambda^t, \theta^t)]^{-1} \nabla_2 F(\lambda^t, \theta^t)$ with:

- ▶ Neumann approximations [[Ghadimi et al. 2018](#), [Ji et al. 2021](#)]:

$$v^t \approx \eta \sum_{q=0}^Q \prod_{k=0}^q (I - \eta \nabla_{22}^2 G_{i_k}(\lambda^t, \theta^t)) \nabla_1 F_j(\lambda^t, \theta^t)$$

- ▶ Stochastic Gradient Descent [[Grazzi et al. 2021](#)] since

$$v^t \in \operatorname{argmin}_{v \in \mathbb{R}^p} \frac{1}{2} \langle \nabla_{22}^2 G(\lambda^t, \theta^t) v, v \rangle + \langle \nabla_2 F(\lambda^t, \theta^t), v \rangle$$

One loop algorithms

Alternate steps in θ and λ [Hong et al. 2020, Yang et al. 2021]:

$$\theta^{t+1} = \theta^t - \rho^t \nabla_2 G_i(\lambda^t, \theta^t) \quad \text{SGD step}$$

$$v^{t+1} = \eta \sum_{q=1}^Q \prod_{k=0}^q (I - \eta \nabla_{22}^2 G_{i_k}(\lambda^t, \theta^{t+1})) \nabla_2 F_j(\lambda^t, \theta^{t+1})$$

Neumann approximation

$$\lambda^{t+1} = \lambda^t - \gamma^t \underbrace{(\nabla_1 F_j(\lambda^t, \theta^{t+1}) - \nabla_{12}^2 G_i(\lambda^t, \theta^{t+1}) v^{t+1})}_{\approx \nabla h(\lambda^t)}$$

Three variables to maintain:

- ▶ $\theta \rightarrow$ inner optimization problem
- ▶ $v \rightarrow$ linear system
- ▶ $\lambda \rightarrow$ outer optimization problem

Idea: evolve in θ , v and λ at the same time following well chosen directions.

Directions:

$$D_{\theta}(\theta, \nu, \lambda) = \nabla_2 G(\lambda, \theta) \quad \text{gradient step toward } \theta^*(\lambda)$$

Directions:

$$D_{\theta}(\theta, \nu, \lambda) = \nabla_2 G(\lambda, \theta) \quad \text{gradient step toward } \theta^*(\lambda)$$

$$D_{\nu}(\theta, \nu, \lambda) = \nabla_{22}^2 G(\lambda, \theta) \nu + \nabla_2 F(\lambda, \theta)$$

$$\text{gradient step toward } - [\nabla_{11}^2 G(\lambda, \theta)]^{-1} \nabla_2 F(\lambda, \theta)$$

Directions:

$$D_{\theta}(\theta, \nu, \lambda) = \nabla_2 G(\lambda, \theta) \quad \text{gradient step toward } \theta^*(\lambda)$$

$$D_{\nu}(\theta, \nu, \lambda) = \nabla_{22}^2 G(\lambda, \theta) \nu + \nabla_2 F(\lambda, \theta)$$

$$\text{gradient step toward } - [\nabla_{11}^2 G(\lambda, \theta)]^{-1} \nabla_2 F(\lambda, \theta)$$

$$D_{\lambda}(\theta, \nu, \lambda) = \nabla_{12}^2 G(\lambda, \theta) \nu + \nabla_1 F(\lambda, \theta)$$

$$\text{gradient step toward } \lambda^*$$

Directions:

$$D_{\theta}(\theta, \nu, \lambda) = \frac{1}{n} \sum_{i=1}^n \nabla_2 G_i(\lambda, \theta)$$

$$D_{\nu}(\theta, \nu, \lambda) = \frac{1}{n} \sum_{i=1}^n \nabla_{22}^2 G_i(\lambda, \theta) \nu + \frac{1}{m} \sum_{j=1}^m \nabla_2 F_j(\lambda, \theta)$$

$$D_{\lambda}(\theta, \nu, \lambda) = \frac{1}{n} \sum_{i=1}^n \nabla_{12}^2 G_i(\lambda, \theta) \nu + \frac{1}{m} \sum_{j=1}^m \nabla_1 F_j(\lambda, \theta)$$

1 for $t = 1, \dots, T$ do

1. Update θ

$$\theta^{t+1} = \theta^t - \rho^t D_{\theta}^t$$

2. Update v

$$v^{t+1} = v^t - \rho^t D_v^t$$

3. Update λ

$$\lambda^{t+1} = \lambda^t - \gamma^t D_{\lambda}^t$$

with D_{θ}^t , D_v^t , D_{λ}^t stochastic estimators of $D_{\theta}(\theta^t, v^t, \lambda^t)$, $D_v(\theta^t, v^t, \lambda^t)$ and $D_{\lambda}(\theta^t, v^t, \lambda^t)$.

SOBA (StOchastic Bilevel Algorithm) directions

Pick $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, m\}$ and take

$$D_{\theta}^t = \nabla_2 G_i(\lambda^t, \theta^t)$$

$$D_v^t = \nabla_{22}^2 G_i(\lambda^t, \theta^t) v^t + \nabla_2 F_j(\lambda^t, \theta^t)$$

$$D_{\lambda}^t = \nabla_{12}^2 G_i(\lambda^t, \theta^t) v^t + \nabla_1 F_j(\lambda^t, \theta^t)$$

SOBA (Stochastic Bilevel Algorithm) directions

$$\mathbb{E}_{i,j}[D_{\theta}^t] = \frac{1}{n} \sum_{i=1}^n \nabla_2 G_i(\lambda^t, \theta^t) = D_{\theta}(\theta^t, v^t, \lambda^t)$$

$$\mathbb{E}_{i,j}[D_v^t] = \frac{1}{n} \sum_{i=1}^n \nabla_{22}^2 G_i(\lambda^t, \theta^t) v^t + \frac{1}{m} \sum_{j=1}^m \nabla_2 F_j(\lambda^t, \theta^t) = D_v(\theta^t, v^t, \lambda^t)$$

$$\mathbb{E}_{i,j}[D_{\lambda}^t] = \frac{1}{n} \sum_{i=1}^n \nabla_{12}^2 G_i(\lambda^t, \theta^t) v^t + \frac{1}{m} \sum_{j=1}^m \nabla_1 F_j(\lambda^t, \theta^t) = D_{\lambda}(\theta^t, v^t, \lambda^t)$$

\Rightarrow As the direction are linear combinations, we can use classical variance reduction techniques.

[SABA, SRBA, ...]

\Rightarrow We prove it converges as fast as single level counter parts

Hyperparameter selection on ℓ^2 regularized logistic regression

Setting:

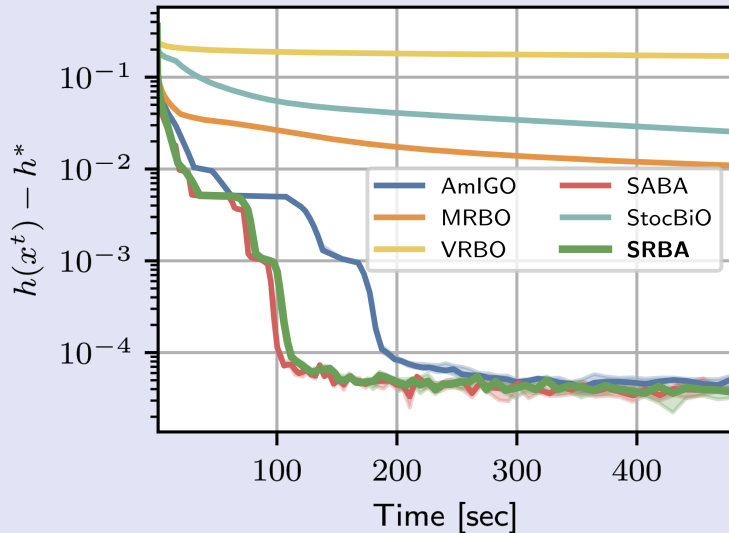
- ▶ Task: binary classification
- ▶ IJCNN1 dataset: 49 990 training samples, 91 701 validation samples, 22 features
- ▶ Training loss:

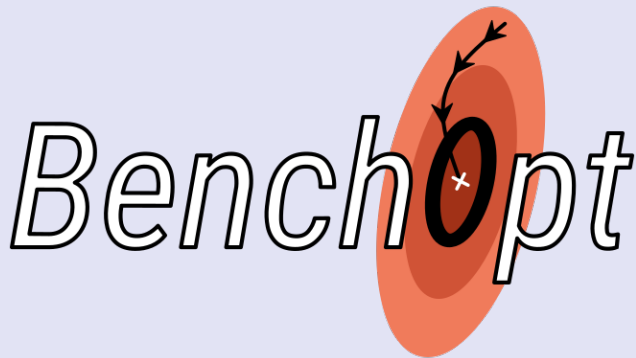
$$G(\theta, \lambda) = \frac{1}{n} \sum_{i=1}^n \log(1 + \exp(-y_i \langle x_i, \theta \rangle)) + \frac{1}{2} \sum_{k=1}^p e^{\lambda_k} \theta_k^2$$

- ▶ Validation loss: logistic loss

$$F(\theta, \lambda) = \frac{1}{m} \sum_{j=1}^m \log(1 + \exp(-y_i^{val} \langle x_i^{val}, \theta \rangle))$$

Hyperparameter selection on ℓ^2 regularized logistic regression





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 sprint in Paris July 2024.

⇒ Next sprint in April, stay tuned!

Conclusion

- ▶ Bi-level optimization is intrinsic in many ML problems.
- ▶ Classical optimization method can be used once we know how to compute the gradient - requires approximating θ^* and v^* .
- ▶ There are various ways to compute the hyper-gradient.


Conclusion

- ▶ Bi-level optimization is intrinsic in many ML problems.
- ▶ Classical optimization method can be used once we know how to compute the gradient - requires approximating θ^* and v^* .
- ▶ There are various ways to compute the hyper-gradient.

Thank you for your attention!

Advertising: we have an open postdoc position to work on bilevel optimization with Pierre Ablin. Contact me if interested!

Slides will be on my web page:

 [tommor.github.io](https://github.com/tommor)

in   @tommor

Iteration overfitting with unrolled optimization

References

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

Implicit deep learning

Consider the Deep Equilibrium Networks (*more general than bilevel*)

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

Implicit deep learning

Consider the Deep Equilibrium Networks (*more general than bilevel*)

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

In practice, solved as

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

with $\mathbf{x}^N(\theta)$ obtained through N iterations of a fixed-point algorithm

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

Implicit deep learning

Consider the Deep Equilibrium Networks (*more general than bilevel*)

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

In practice, solved as

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

with $\mathbf{x}^N(\theta)$ obtained through N iterations of a fixed-point algorithm

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

\Rightarrow Is this true for all models?

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

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

Theorem 1 – Iteration overfitting

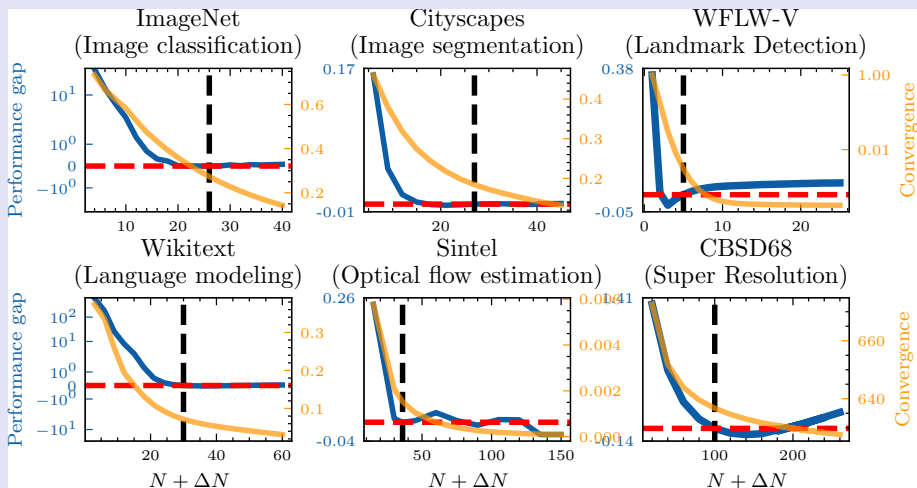
Under simplifying hypothesis (linear DEqs), if f_θ is overparametrized, we have for all ΔN :

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

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

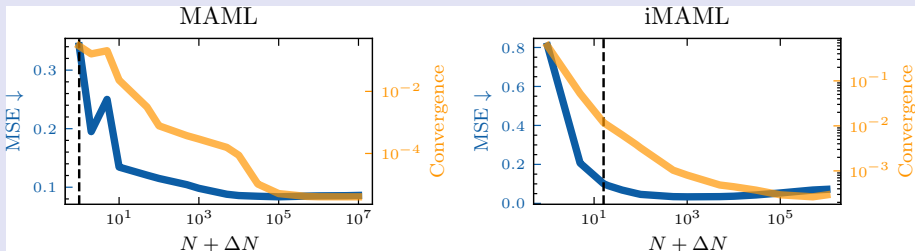
What happens in practice?

Context: Overparametrized DEQs.



What happens in practice?

Context: Underparametrized Meta-learning.



Theorem (Convergence of SOBA)

Under some regularity assumptions on F and G , if h is bounded, then for decreasing step sizes that verify $\rho^t = \alpha t^{-\frac{2}{5}}$ and $\gamma^t = \beta t^{-\frac{3}{5}}$ for some $\alpha, \beta > 0$, the iterates $(\lambda^t)_{1 \leq t \leq T}$ of SOBA verify

$$\inf_{t \leq T} \mathbb{E}[\|\nabla h(\lambda^t)\|^2] = \mathcal{O}(T^{-\frac{2}{5}}) .$$

Single level problem:

$$\min_{\theta \in \mathbb{R}^p} f(\theta) = \frac{1}{n} \sum_{i=1}^n f_i(\theta)$$

Single level problem:

$$\min_{\theta \in \mathbb{R}^p} f(\theta) = \frac{1}{n} \sum_{i=1}^n f_i(\theta)$$

Initialisation: Compute and store $m[i] = \nabla f_i(\theta^0)$ for any $i \in \{1, \dots, n\}$ and $S[m] = \frac{1}{n} \sum_{i=1}^n m[i]$.

Single level problem:

$$\min_{\theta \in \mathbb{R}^p} f(\theta) = \frac{1}{n} \sum_{i=1}^n f_i(\theta)$$

Initialisation: Compute and store $m[i] = \nabla f_i(\theta^0)$ for any $i \in \{1, \dots, n\}$ and $S[m] = \frac{1}{n} \sum_{i=1}^n m[i]$.

At iteration t :

1. Pick $i \in \{1, \dots, n\}$

2. Update θ

$$\theta^{t+1} = \theta^t - \rho(\nabla f_i(\theta^t) \underbrace{- m[i] + S[m]}_{\text{variance reduction}})$$

3. Update the memory

$$m[i] \leftarrow \nabla f_i(\theta^t)$$

Bilevel case: SABA (Stochastic Average Bilevel Algorithm)

To estimate

$$D_{\theta}(\theta^t, v^t, \lambda^t) = \nabla_2 G(\lambda^t, \theta^t)$$

$$D_v(\theta^t, v^t, \lambda^t) = \nabla_{22}^2 G(\lambda^t, \theta^t) v^t + \nabla_2 F(\lambda^t, \theta^t)$$

$$D_{\lambda}(\theta^t, v^t, \lambda^t) = \nabla_{12}^2 G(\lambda^t, \theta^t) v^t + \nabla_1 F(\lambda^t, \theta^t)$$

we have 5 quantities to estimate on the principle of SAGA:

$$\begin{aligned} \nabla_2 G(\lambda^t, \theta^t), \quad \nabla_2 F(\lambda^t, \theta^t), \quad \nabla_1 F(\lambda^t, \theta^t) \\ \nabla_{12}^2 G(\lambda^t, \theta^t) v^t, \quad \nabla_{22}^2 G(\lambda^t, \theta^t) v^t \end{aligned}$$

D_{θ}^t , D_v^t and D_{λ}^t given using these estimates = **SABA directions**

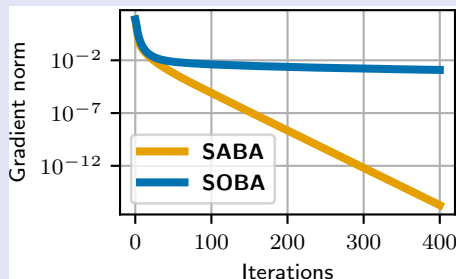
Theorem (Convergence of SABA)

Under some regularity assumptions on F and G , with constant and small enough step sizes, the iterates $(\lambda^t)_{1 \leq t \leq T}$ of SABA verify

$$\frac{1}{T} \sum_{i=1}^T \mathbb{E}[\|\nabla h(\lambda^t)\|^2] = \mathcal{O}(T^{-1}) .$$

Remarks

- ▶ We match the convergence rate of gradient descent
- ▶ SABA converges with fixed step sizes
- ▶ Faster than SOBA



Number of calls to oracle to get an ϵ -stationary solution.

amlGO	stoBiO	TTSA	MRBO	SUSTAIN	SOBA	SABA
$\mathcal{O}(\epsilon^{-2})$	$\tilde{\mathcal{O}}(\epsilon^{-2})$	$\tilde{\mathcal{O}}(\epsilon^{-5/2})$	$\tilde{\mathcal{O}}(\epsilon^{-3/2})$	$\mathcal{O}(\epsilon^{-3/2})$	$\mathcal{O}(\epsilon^{-5/2})$	$\mathcal{O}(\epsilon^{-1})$

SABA achieves SOTA complexity

The role of warm-starting for unrolled optimization

References



Proximal Gradient Descent: Iterate

$$\mathbf{x}^{k+1} = \text{prox}_{\rho\mathcal{R}}(\mathbf{x}^k - \rho\nabla f(\mathbf{x}^k))$$

for the original problem $\mathbf{x}^*(\mathbf{y}; \theta) = \underset{\mathbf{x}}{\operatorname{argmin}} \underbrace{\frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2}_{f(\mathbf{x})} + \mathcal{R}(\mathbf{x}; \theta)$

Proximal Gradient Descent: Iterate

$$\mathbf{x}^{k+1} = \text{prox}_{\rho\mathcal{R}}(\mathbf{x}^k - \rho\nabla f(\mathbf{x}^k))$$

for the original problem $\mathbf{x}^*(\mathbf{y}; \theta) = \underset{\mathbf{x}}{\text{argmin}} \underbrace{\frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2}_{f(\mathbf{x})} + \mathcal{R}(\mathbf{x}; \theta)$

However, computing the proximal operator can be expensive

requires a sub-routine

Dictionary-based denoisers: take \mathcal{D} as

$$D(\underset{z}{\text{argmin}} \|\mathbf{x} - Dz\|_2^2 + \lambda\|z\|_1) \quad \text{or} \quad \underset{u}{\text{argmin}} \|\mathbf{x} - u\|_2^2 + \lambda\|\Gamma u\|_1$$

These denoisers are proximal operators, so the PnP algorithm converges

Proximal Gradient Descent: Iterate

$$\mathbf{x}^{k+1} = \text{prox}_{\rho\mathcal{R}}(\mathbf{x}^k - \rho\nabla f(\mathbf{x}^k))$$

for the original problem $\mathbf{x}^*(\mathbf{y}; \theta) = \underset{\mathbf{x}}{\text{argmin}} \underbrace{\frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2}_{f(\mathbf{x})} + \mathcal{R}(\mathbf{x}; \theta)$

However, computing the proximal operator can be expensive

requires a sub-routine

Dictionary-based denoisers: take \mathcal{D} as

$$D(\underset{z}{\text{argmin}} \|\mathbf{x} - Dz\|_2^2 + \lambda\|z\|_1) \quad \text{or} \quad \underset{u}{\text{argmin}} \|\mathbf{x} - u\|_2^2 + \lambda\|\Gamma u\|_1$$

These denoisers are proximal operators, so the PnP algorithm converges

\Rightarrow Replace the proximal operator by an unrolled model with convergence?

Replace computing the proximal operator with L steps of a solver for the proximal operator with warm-starting

$$\begin{cases} \mathbf{x}^{k+1/2} = \mathbf{x}^k - \rho \nabla f(\mathbf{x}^k) \\ \mathbf{x}^{k+1}, u^{k+1} = T^L(\mathbf{x}^{k+1/2}, u^k) \end{cases}$$

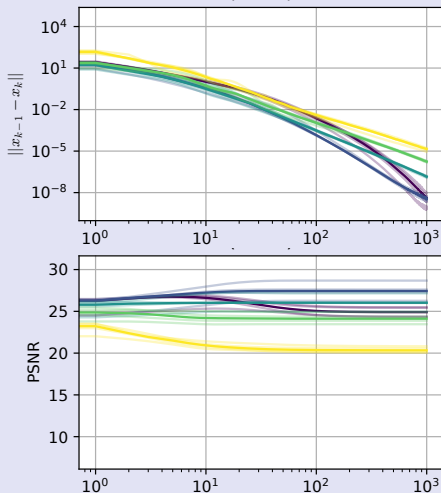
- ▶ We show that for $L \rightarrow \infty$, the PnP algorithm converges
- ▶ For $L = 1$, the PnP algorithm also converges to the same solution
- ▶ For intermediate L , it is conjectured that it converges.

Only able to show convergence for a smoothed version of the problem

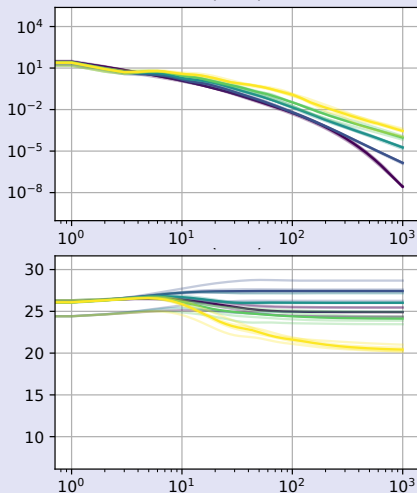
\Rightarrow Warm-starting is key for the convergence here!

Stability of the unrolled algorithm

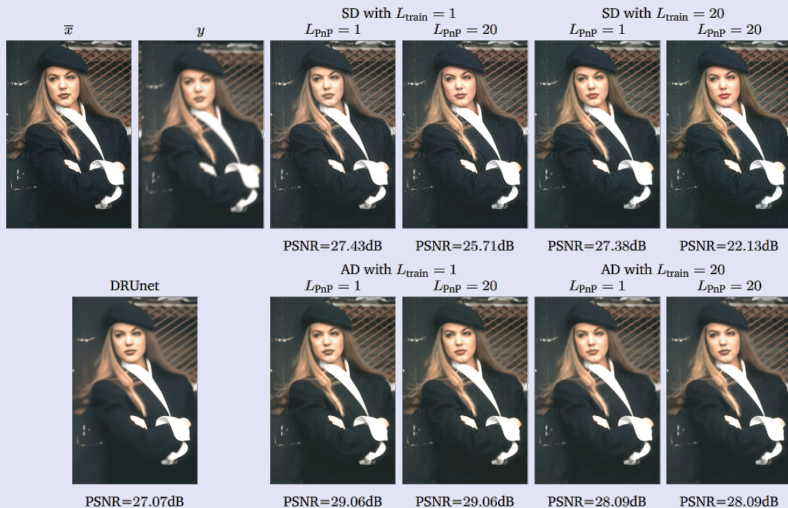
PNP-AD ($L_{train}/PnP = 20/20$)
(31.0s)



PNP-AD ($L_{train}/PnP = 20/1$)
(1.3s)



Dictionary-based denoisers in PnP



Take-home message

- ▶ Unrolled networks can be good approximations of the argmin for smooth problems
- ▶ **For non-smooth problems, the jacobian estimate is unstable**
- ▶ Beware that training with fixed number of iterations leads to iteration overfitting
- ▶ Warm-starting can be a key to get convergence with unrolled algorithms