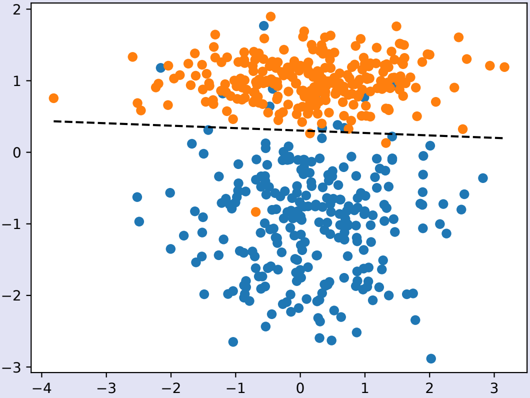
Bi-level optimization in Machine Learning

Thomas Moreau INRIA Saclay

Setup:

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



Learning a linear ML model

Logistic loss:

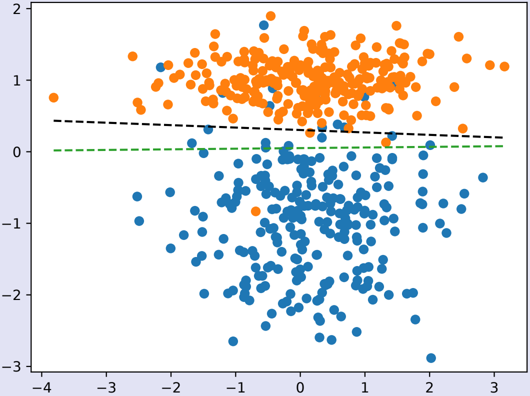
$$G(\theta) = \frac{1}{N}\sum_{i=1}^{N} \log(1 + e^{-y_i\langle\theta, X_i\rangle})$$

Training the model:

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

Avoiding overfitting

Here, the second feature is uninformative,



Avoiding overfitting with a regularization

Regularized **Logistic loss:**

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

Training the model:

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

Avoiding overfitting with a regularization

Regularized **Logistic loss:**

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

Training the model:

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

$\Rightarrow$ **How to choose λ?**

Evaluating the generalization

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

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

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

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 $(X_i^{val}, y_i^{val})_{i=1}^M$

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

Independent estimate of the risk of the model.

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

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 rewritting:

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

The Grid Search with multiple hyperparameters

Regularized Logistic loss:

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

The Grid Search with multiple hyperparameters

Regularized Logistic loss:

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

Grid search is inefficient as the grid increases exponentially with the number of parameters.

Regularized Logistic loss:

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

Grid search is inefficient as the grid increases exponentially with the number of parameters.

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

Bi-level optimization

Bi-level problem: Optimization problem with two levels

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

Outer function

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

Value function

Inner function/Problem

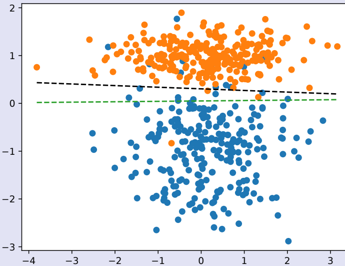
Goal: Optimize the value function h whose value depends on the result of another optimization problem.

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.

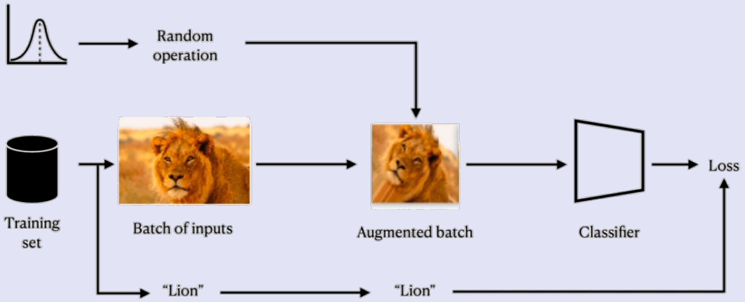
Hyperparameter optimization: λ is a regularization parameter:



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.

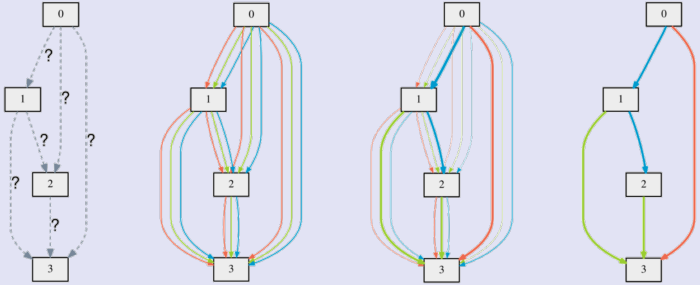
Data augmentation: λ parametrizes the transformations distribution.



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.



(a) (b) (c) (d)

Deep Equilibrium Network:

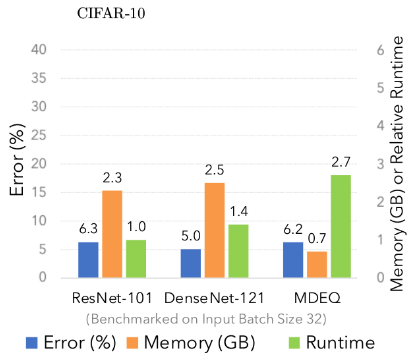
$$\begin{cases} \min_\lambda h(\lambda) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(y_i, \theta^*(X_i,\lambda)) \\ s.t. \quad \theta^*(X_i,\lambda) = g_\lambda(\theta^*(X_i,\lambda)) \end{cases}$$

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

- **Mimic infinite depth:**

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

- **Efficient memory**
- **Slow runtime**



Solving bi-level optimization

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

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

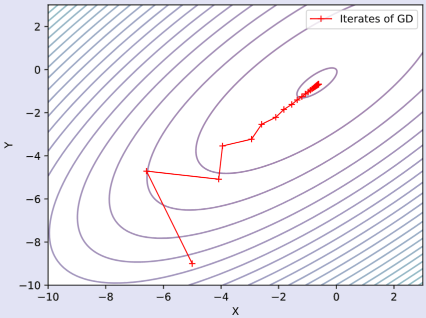
$\Rightarrow$ Do not scale well with the dimension

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.



Computing the gradient of h

Value function definition:

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

Chain rule:

$$\nabla_\lambda h(\lambda) = \nabla_1 F(\lambda, \theta^*(\lambda)) + (d\theta^*(\lambda))^T \nabla_2 F(\lambda, \theta^*(\lambda))$$

Optimality condition for θ^*

$$\nabla_2 G(\lambda, \theta^*(\lambda)) = 0$$

Jacobian of θ^* - implicit differentiation

Optimality condition for θ^*

$$\nabla_2 G(\lambda, \theta^*(\lambda)) = 0$$

Derivating this equation relative to λ gives:

$$\nabla^2_{22} G(\lambda, \theta^*(\lambda)) d\theta^*(\lambda) + \nabla^2_{21} G(\lambda, \theta^*(\lambda)) = 0,$$

Jacobian of θ^* - implicit differentiation

Optimality condition for θ^*

$$\nabla_2 G(\lambda, \theta^*(\lambda)) = 0$$

Derivating this equation relative to λ gives:

$$\nabla^2_{22} G(\lambda, \theta^*(\lambda)) d\theta^*(\lambda) + \nabla^2_{21} G(\lambda, \theta^*(\lambda)) = 0,$$

Implicit function theorem

$$d\theta^*(\lambda) = -\left[\nabla^2_{22} G(\lambda, \theta^*(\lambda))\right]^{-1} \nabla^2_{21} G(\lambda, \theta^*(\lambda)),$$

Computing the gradient of h

Value function gradient:

$$\nabla h(\lambda) = \nabla_1 F(\lambda, \theta^*) - \nabla^2_{21} G(\lambda, \theta^*) \left[\nabla^2_{22} G(\lambda, \theta^*)\right]^{-1} \nabla_2 F(\lambda, \theta^*)$$

Computing the gradient of h

Value function gradient:

$$\nabla h(\lambda) = \nabla_1 F(\lambda, \theta^*) - \nabla^2_{21} G(\lambda, \theta^*)\left[\nabla^2_{22} G(\lambda, \theta^*)\right]^{-1}\nabla_2 F(\lambda, \theta^*)$$

- Need to compute the solution of the inner

Value function gradient:

$$\nabla h(\lambda) = \nabla_1 F(\lambda, \theta^*) - \nabla^2_{21} G(\lambda, \theta^*) \left[\nabla^2_{22} G(\lambda, \theta^*)\right]^{-1} \nabla_2 F(\lambda, \theta^*)$$

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

$$v^*(\lambda) = \left[\nabla^2_{22} G(\lambda, \theta^*)\right]^{-1} \nabla_2 F(\lambda, \theta^*)$$

Approximate bi-level optimization

References

- Pedregosa, F. (2016). Hyperparameter optimization with approximate gradient. In *International Conference on Machine Learning (ICML)*, pages 737–746, New-York, NY, USA

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

Idea: Approximate $\theta^*(\lambda^t)$ and $v^*(\lambda^t) = \left[\nabla^2_{22} G(\lambda^t, \theta^*)\right]^{-1} \nabla_2 F(\lambda^t, \theta^*)$

Hyperparameter optimization with Approximate Gradient HOAG [Pedregosa 2016]

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

Idea: Approximate $\theta^*(\lambda^t)$ and $v^*(\lambda^t) = \left[\nabla^2_{22} G(\lambda^t, \theta^*)\right]^{-1} \nabla_2 F(\lambda^t, \theta^*)$

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

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

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

$$\nabla^2_{22} G(\lambda^t, \theta^t) \approx Id \qquad \nabla^2_{22} G(\lambda^t, \theta^t)^{-1} \approx \sum_k (Id - \nabla^2_{22} G(\lambda^t, \theta^t))^k$$

- Algorithm unrolling

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

SHINE - Sharing the INverse Estimate

References

- 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

Quasi Newton 101:

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

Newton Method

$$\theta^{t+1} = \theta^t - \left[\nabla^2 G(\theta^t)\right]^{-1} \nabla G(\theta^t)$$

Quasi-Newton Method

$$\theta^{t+1} = \theta^t - B_t^{-1} \nabla G(\theta^t)$$

B_t: low-rank approx. of $\nabla^2 G(\theta^t)$.

Inverse with Sherman-Morrison

Quasi Newton 101:

$$\text{Solving } \theta^* = \text{argmin}_\theta\, G(\theta)$$

Newton Method

$$\theta^{t+1} = \theta^t - \left[\nabla^2 G(\theta^t)\right]^{-1} \nabla G(\theta^t)$$

Quasi-Newton Method

$$\theta^{t+1} = \theta^t - B_t^{-1} \nabla G(\theta^t)$$

B_t: low-rank approx. of $\nabla^2 G(\theta^t)$.

Inverse with Sherman-Morrison

$\Rightarrow$ The Hessian for v^* is the same as the one from the inner problem.

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

$$\begin{cases} \tilde{v}_t = B_t^{-1} \nabla_2 F(\lambda, \theta^t) \\ \tilde{\nabla} h(\lambda) = \nabla_1 F(\lambda, \theta^t) + \nabla^2_{12} G(\lambda, \theta^t) \tilde{v}_t \end{cases}$$

Properties of B:

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

$$\begin{cases} \tilde{v}_t = B_t^{-1} \nabla_2 F(\lambda, \theta^t) \\ \tilde{\nabla} h(\lambda) = \nabla_1 F(\lambda, \theta^t) + \nabla^2_{12} G(\lambda, \theta^t) \tilde{v}_t \end{cases}$$

Properties of B:

- It is computed when solving $\theta^* = \operatorname{argmin}_\theta G(\theta)$ using a quasi-Newton method.

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

$$\begin{cases} \tilde{v}_t = B_t^{-1} \nabla_2 F(\lambda, \theta^t) \\ \tilde{\nabla} h(\lambda) = \nabla_1 F(\lambda, \theta^t) + \nabla^2_{12} G(\lambda, \theta^t) \tilde{v}_t \end{cases}$$

Properties of B:

- It is computed when solving $\theta^* = \operatorname{argmin}_\theta G(\theta)$ using a quasi-Newton method.
- It is easily invertible using the Sherman-Morrison formula, because low-rank.

SHINE direction convergence

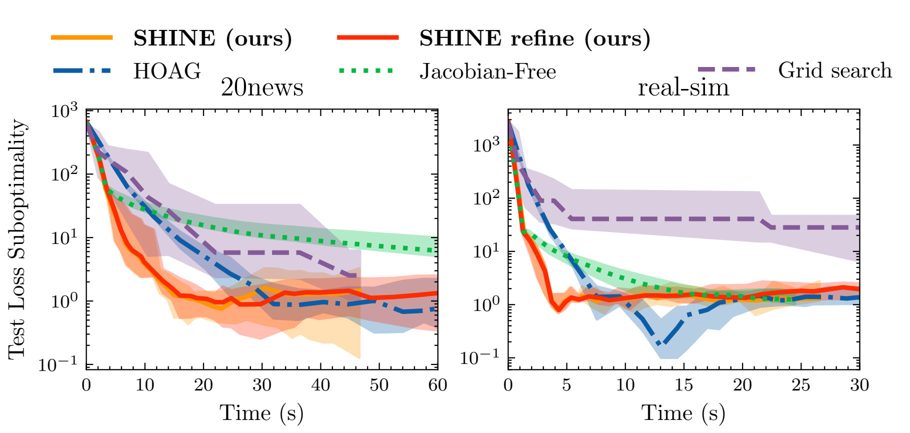
Theorem (Convergence of SHINE to the Hypergradient using ULI)

Under the Uniform Linear Independence (ULI) assumption and some additional smoothness and convexity assumptions, for a given parameter λ*,* (θ^t) *converges q-superlinearly to* $\theta^\star$ *and*

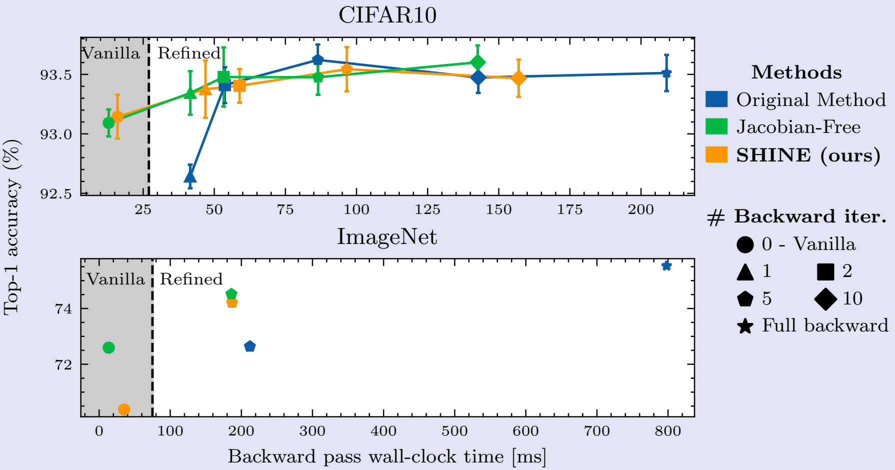
$$\lim_{t \to \infty} \nabla_1 F(\lambda, \theta^t) + \nabla_{12}^2 G(\lambda, \theta^t) \tilde{v}_t = \nabla h(\lambda).$$

SHINE - Hyper-parameter optimization [Ramzi et al. 2022]

Logistic Regression with ℓ_2-regularisation on 2 datasets:



Multiscale DEQ on CIFAR10:



Stochastic Bi-level Optimization

A framework for linear updates

References

- Dagréou, M., Ablin, P., Vaiter, S., and Moreau, T. (2022). A framework for bilevel optimization that enables stochastic and global variance reduction algorithms. *preprint ArXiv*, 2201.13409

Classical ML setting:

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

Empirical Risk minimization

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.

Aside: Stochastic optimization for single level problems

Single level problem:

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

Aside: Stochastic optimization for single level problems

Single level problem:

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

First order stochastic optimization:

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

Aside: Stochastic optimization for single level problems

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 Monro 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

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

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

Bilevel optimization case

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

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

Problem:

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

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

1. Take for θ^t an approximation of $\theta^*(\lambda^t)$
2. Take for v^t an approximation of $\left[\nabla^2_{22} G(\lambda^t, \theta^t)\right]^{-1} \nabla_2 F(\lambda^t, \theta^t)$
3. Set
$$p^t = \underbrace{\nabla_1 F(\lambda^t, \theta^t) - \nabla^2_{12} G(\lambda^t, \theta^t) v^t}_{\approx \nabla h(\lambda^t)}$$
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 $v^t = \left[\nabla^2_{22} G(\lambda^t, \theta^t)\right]^{-1} \nabla_2 F(\lambda^t, \theta^t)$ with:

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

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

What about the linear system?

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

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

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

- Stochastic Gradient Descent [Grazzi et al. 2021] since

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

One loop algorithms

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

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

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

Neumann approximation

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

Main idea

Three variables to maintain:

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

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

Directions:

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

Motivation of the framework

Directions:

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

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

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

Directions:

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

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

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

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

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

Directions:

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

$$D_v(\theta, v, \lambda) = \frac{1}{n}\sum_{i=1}^{n} \nabla^2_{22} G_i(\lambda, \theta) v + \frac{1}{m}\sum_{j=1}^{m} \nabla_2 F_j(\lambda, \theta)$$

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

Proposed framework

for $t = 1, \ldots, T$ **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 $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$ and take

$$\begin{aligned}
D_\theta^t &= \nabla_2 G_i(\lambda^t, \theta^t) \\
D_v^t &= \nabla_{22}^2 G_i(\lambda^t, \theta^t) v^t + \nabla_2 F_j(\lambda^t, \theta^t) \\
D_\lambda^t &= \nabla_{12}^2 G_i(\lambda^t, \theta^t) v^t + \nabla_1 F_j(\lambda^t, \theta^t)
\end{aligned}$$

SOBA (StOchastic Bilevel Algorithm) directions

$$\mathbb{E}_{i,j}[D_\theta^t] = \frac{1}{n}\sum_{i=1}^{n} \nabla_2 G_i(\lambda^t, \theta^t) = D_\theta(\theta^t, v^t, \lambda^t)$$
$$\mathbb{E}_{i,j}[D_v^t] = \frac{1}{n}\sum_{i=1}^{n} \nabla_{22}^2 G_i(\lambda^t, \theta^t) v^t + \frac{1}{m}\sum_{j=1}^{m} \nabla_2 F_j(\lambda^t, \theta^t) = D_v(\theta^t, v^t, \lambda^t)$$
$$\mathbb{E}_{i,j}[D_\lambda^t] = \frac{1}{n}\sum_{i=1}^{n} \nabla_{12}^2 G_i(\lambda^t, \theta^t) v^t + \frac{1}{m}\sum_{j=1}^{m} \nabla_1 F_j(\lambda^t, \theta^t) = D_\lambda(\theta^t, v^t, \lambda^t)$$

Theoretical guarantees of SOBA

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

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

Aside: SAGA for single level problems [Defazio et al. 2014]

Single level problem:

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

Aside: SAGA for single level problems [Defazio et al. 2014]

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 $i \in \{1, \ldots, n\}$ and $S[m] = \frac{1}{n} \sum_{i=1}^{n} m[i]$.

Aside: SAGA for single level problems [Defazio et al. 2014]

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 $i \in \{1, \ldots, n\}$ and $S[m] = \frac{1}{n} \sum_{i=1}^{n} m[i]$.
At iteration t:

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

$$\begin{aligned} D_\theta(\theta^t, v^t, \lambda^t) &= \nabla_2 G(\lambda^t, \theta^t) \\ D_v(\theta^t, v^t, \lambda^t) &= \nabla^2_{22} G(\lambda^t, \theta^t) v^t + \nabla_2 F(\lambda^t, \theta^t) \\ D_\lambda(\theta^t, v^t, \lambda^t) &= \nabla^2_{12} G(\lambda^t, \theta^t) v^t + \nabla_1 F(\lambda^t, \theta^t) \end{aligned}$$

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

$$\nabla_2 G(\lambda^t, \theta^t), \quad \nabla_2 F(\lambda^t, \theta^t), \quad \nabla_1 F(\lambda^t, \theta^t)$$
$$\nabla^2_{12} G(\lambda^t, \theta^t) v^t, \quad \nabla^2_{22} G(\lambda^t, \theta^t) v^t$$

D^t_θ, D^t_v and D^t_λ given using these estimates = **SABA directions**

Theoretical guarantees

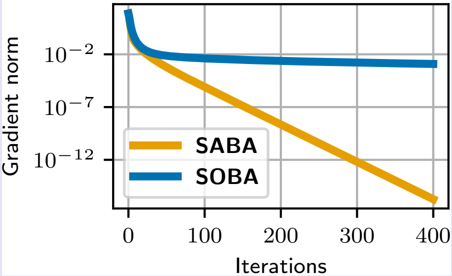
Theorem (Convergence of SABA)

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



Complexity

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

amIGO	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

Hyperparameter selection on ℓ^2 regularized logistic regression

Setting:

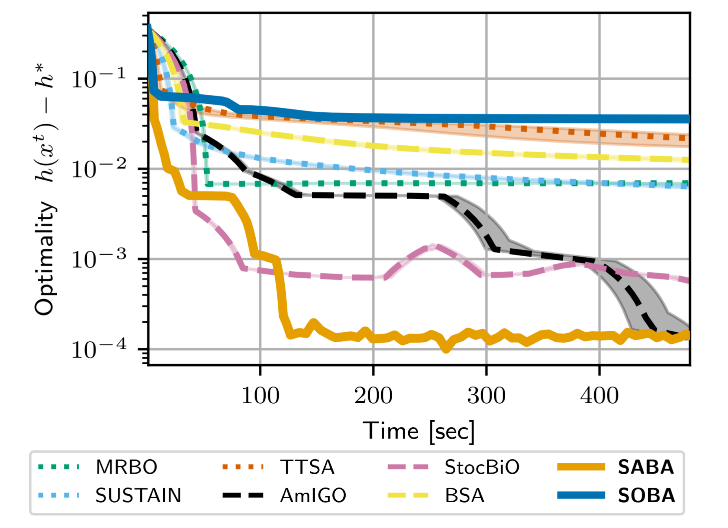
- Task: binary classification
- IJCNN1 dataset: 49 990 training samples, 91 701 validation samples, 22 features
- Training loss:

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

- 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



Stochastic bi-level framework

- It is possible to adapt any kind of single level stochastic optimizer to our framework.
- As in single level optimization, variance reduction allows to get convergence rate that matches rates of full batch gradient descent.

Conclusion

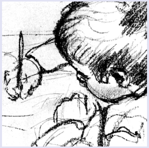
- Bi-level optimization is intrinsic in many ML problems.
- Classical optimization method can be used once we know how to compute the gradient - requires approximating θ^* and v^*.
- Maybe linear dynamic is the solution (1-loop vs 2-loops)

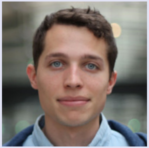
Slides will be on my web page:

`tommoral.github.io` @tomamoral

Thanks to all my bi-level collaborators!

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 $F = G$

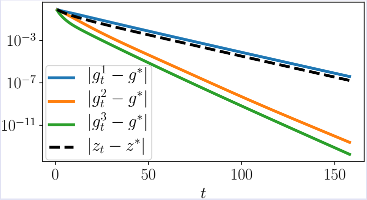
$$\Rightarrow \text{Here, } \frac{\partial F}{\partial \theta}(\lambda, \theta^*) = 0$$

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,

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



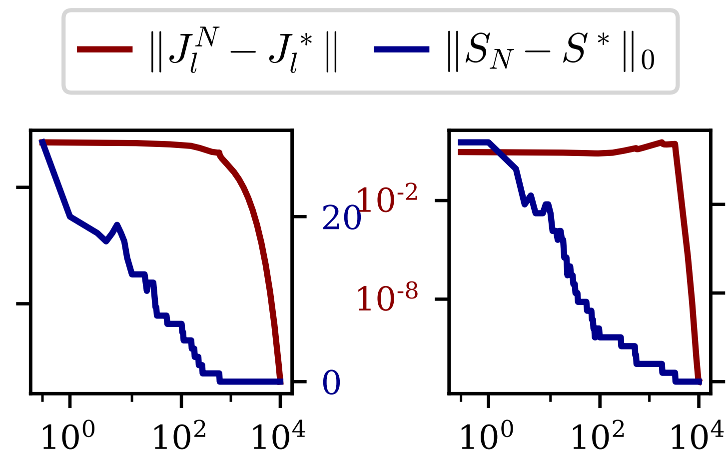
Context: dictionary learning, $F = G$ with an ℓ_1-regularization for θ.

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

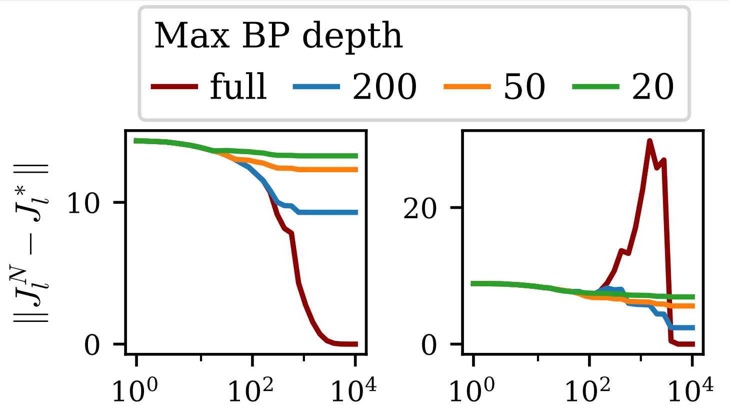
$$\left(\frac{\partial \theta^*}{\partial D_l}\right)_{S^*} = -(D_{:,S^*}^\top D_{:,S^*})^{-1}(D_l {\theta^*}^\top + (D_l^\top \theta^* - y_l) Id_n)_{S^*} \enspace ,$$

$\Rightarrow$ 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.



Hypergradient computation

References

- 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

Linear system approximation v^*

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]