Principled and Efficient Bilevel Optimization for Machine Learning

Riccardo Grazzi, Massimiliano Pontil

Computational Statistics and Machine Learning, Istituto Italiano di Tecnologia
Department of Computer Science, University College London

riccardo.grazzi@iit.it

ISTITUTO ITALIANO
DI TECNOLOGIA

Outline

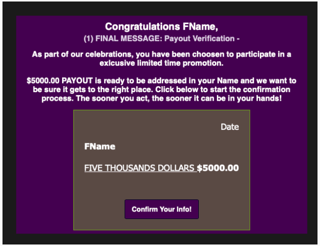
Supervised Learning

Real Estate Pricing

	Quarter	Bedrooms	Size (sq m)
0	Albaro	3	100
1	Boccadasse	2	85
2	Castelletto	4	120
3	Molassana	3	95

	Price (€)
0	350000
1	280000
2	400000
3	320000

Spam Detection



{Spam, Not spam}

Image Recognition

{Elephant, Cat, Dog, ... }

The Supervised Learning Setting

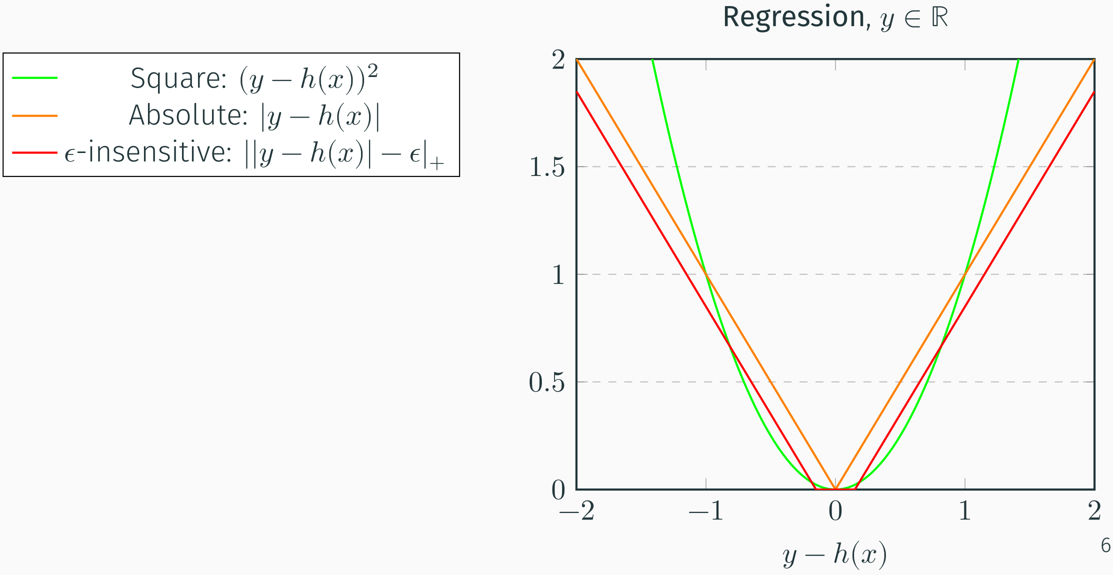
Expected Risk

$$R(h) := \mathbb{E}_{(x,y)\sim\rho}[\mathcal{L}(h(x), y)] = \int_{\mathcal{X}\times\mathcal{Y}} \mathcal{L}(h(x), y) d\rho(x, y)$$

- $\mathcal{X}$ and $\mathcal{Y}$ are the **input** and **output** spaces where the data lives.
- ρ is the unknown **data distribution**.
- $h : \mathcal{X} \to \mathcal{Y}$ is a machine learning **model**.
- The **loss** $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ measures the discrepancy between $h(x)$ and y.

The appropriate choice of $\mathcal{L}$ depends on the problem.

Popular Loss Functions for Regression



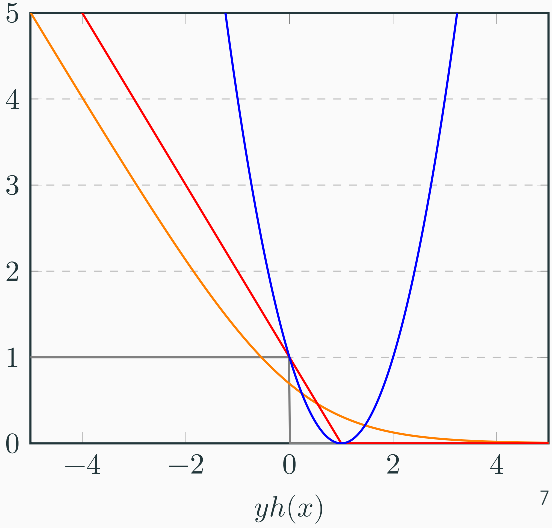
Popular Loss Functions for Classification

Loss	Formula
Zero-one:	$1\{yh(x) < 0\}$
Cross-entropy:	$y \log\left(1 + e^{-yh(x)}\right)$
Hinge:	$\max\left(0, 1 - yh(x)\right)$
Square:	$(1 - yh(x))^2$

Multi-class, $y \in \{0,1\}^c$, $\sum_i y_i = 1$:

- Zero-one:
  $1\{\arg\max_i y_i = \arg\max_i h(x)_i\}$.
- Cross-entropy:
  $-\sum_i 1\{y_i = 1\} \log\left(\frac{e^{-h(x)_i}}{\sum_j e^{-h(x)_j}}\right)$.

Binary, $y \in \{-1, 1\}$



Empirical Risk Minimization

- We cannot compute $h^\star = \arg\min_h \{R(h)\}$ since ρ is unknown.
- **Goal**: find $h_D \approx h^\star$ from $D = \{(x_i, y_i)\}_{i=1}^n$ i.i.d. sampled from ρ

Regularized Empirical Risk Minimization (ERM)

Set **model class** $\mathcal{H}$ and (optional) regularizer $\mathcal{R}_\lambda : \mathcal{H} \to \mathbb{R}$ and compute

$$h_D \in \arg\min_{h \in \mathcal{H}} \{R_D(h) + \underbrace{\mathcal{R}_\lambda(h)}_{\text{Regularizer}}\}, \quad \text{where} \quad \underbrace{R_D(h)}_{\text{Empirical Risk}} := \frac{1}{n}\sum_{i=1}^n \mathcal{L}(h(x_i), y_i)$$

Common choices:

Complexity

- **Linear**: $h(x) = w^\top x$, $\mathcal{H}_{\text{linear}} = \{h(x) : h(x) = w^\top x, w \in \mathbb{R}^d\}$
- **Polynomial**: $h(x) = w^\top \phi(x)$, $\phi(x) =$ polynomial's variables
- **Kernel Methods**: $h(x) = \langle w, \phi(x) \rangle$, $\phi(x)$ may have infinite dimensions.
- **(Deep) Neural Networks (DNNs)**: $h(x) = w^\top \sigma(W_l \sigma(W_{l-1} \cdots \sigma(W_1 x)))$

Overfitting: Complex $\mathcal{H}$ can yield h_D with low empirical but high expected risk.

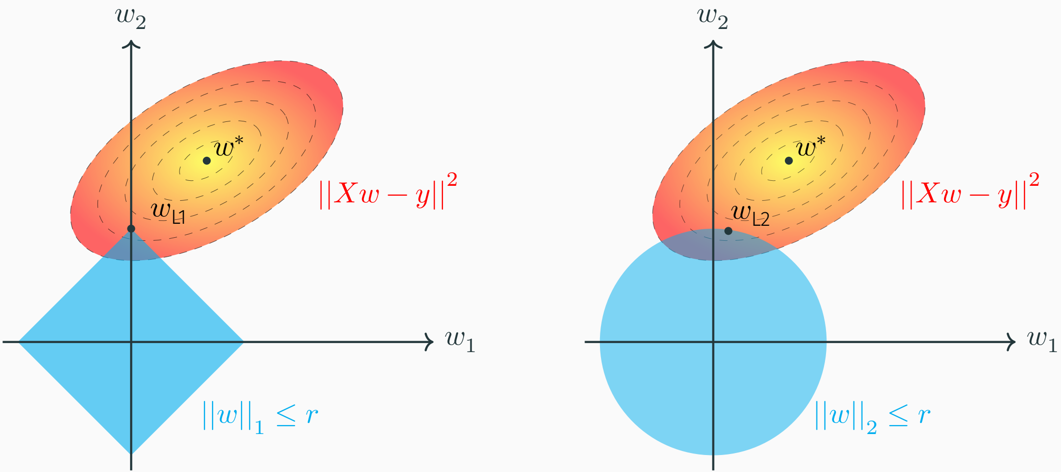
$\implies$ **Regularization:** penalize complex models (Occam's razor).

Choosing a Regularizer

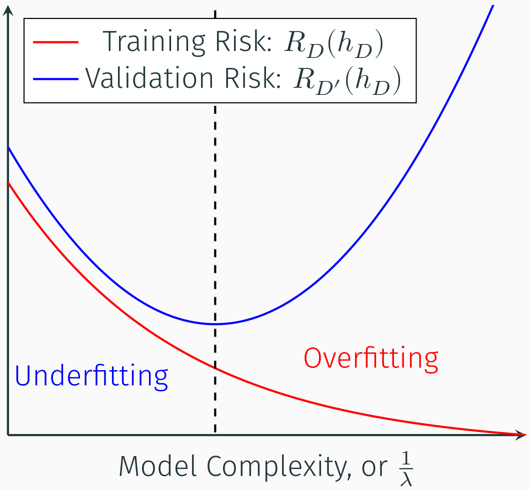
- **Lp Regularization:** $\mathcal{R}_\lambda(h_w) = \lambda \|w\|_p^p$, with $\lambda \in \mathbb{R}_+$, $p \in [0, \infty)$.
- **Elastic Net**: $\mathcal{R}_\lambda(h_w) = \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2^2$
- **Many regularization hyperparameters:** e.g. $\mathcal{R}_\lambda(h_w) = w^\top \text{diag}(\lambda) w$

Tikhonov Regularization		Ivanov Regularization
$\min_{h_w \in \mathcal{H}} \{R_D(h_w) + \lambda \|w\|_p^p\}$	$\Longleftrightarrow$	$\min_{h_w \in \mathcal{H}, \|w\|_p^p \leq r} \{R_D(h)\}$

L1 and L2 regularization: Ivanov Visualization



Choosing the Regularization Hyperparameter



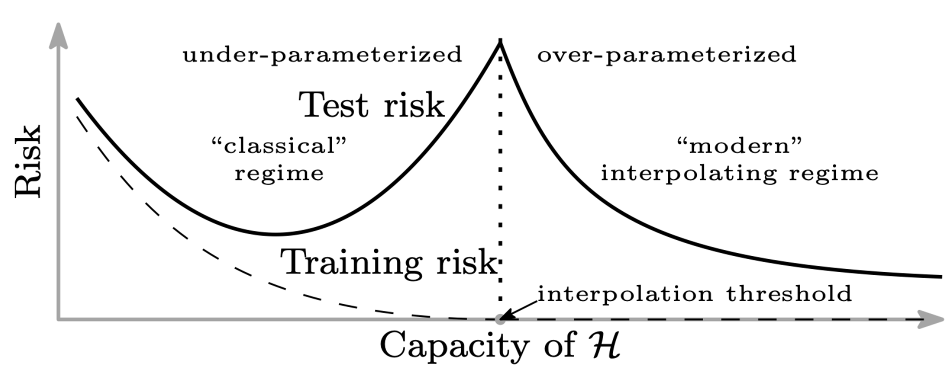
Data split into **train** D and **validation** D'.

$\lambda^* = \arg\min_{\lambda} R_{D'}(h_D)$.

Solved via **grid/random search**.

Problematic for many hyperparameters.

Overparametization and Double Descent ($\lambda = 0$)



Source: Belkin et al. (2019)

Gradient descent select minimum norm solutions $\implies$ implicit regularization.

Computing the Empirical Risk Minimizer

$$\min_{w\in\mathcal{W}}\left\{f(w) := \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(h_w(x_i), y_i) + \mathcal{R}_\lambda(h_w)\right\}$$

where $\mathcal{H} = \{h_w(x) : w \in \mathcal{W}\}$ and w are the **parameters** of h_w.

- f can be **convex** (h_w is linear, polynomial, kernel) or **non-convex** (h_w is a NN).
- f can have multiple minima, e.g. when $\mathcal{W} = \mathbb{R}^d$ and $d > n$.
- f and its derivatives can be costly to evaluate for complex h_w and large n.

Optimization should be efficient and might affect the learning process

$$\min_{w \in \mathbb{R}^d} f(w), \qquad f : \mathbb{R}^d \to \mathbb{R} \text{ differentiable}$$

Stochastic Gradient Descent (SGD):

$$w_{t+1} = w_t - \eta_t \nabla \hat{f}(w_t; \xi), \qquad w_0 \in \mathbb{R}^d.$$

- $\eta_t > 0$ are the **step-sizes** or **learning rates**.
- ξ is a **random variable** (e.g. $\hat{f}(w; \xi) = \mathcal{L}(w^\top x_\xi, y_\xi)$)
- **Unbiased**: $\mathbb{E}_\xi[\nabla \hat{f}(w; \xi)] = \nabla f(w)$
- Gradient Descent (GD): $\nabla \hat{f}(w, \xi) = \nabla f(w)$.

Stochastic Gradient Descent with Momentum (SGDM)

$$
\begin{aligned}
v_{t+1} &= \gamma_t v_t + \nabla \hat{f}(w_t; \xi_t), & v_0 &= 0, \\
w_{t+1} &= w_t - \eta_t v_{t+1}, & w_0 &\in \mathbb{R}^d.
\end{aligned}
$$

- v_t is an **average of past gradients**.
- $\gamma_t > 0$ are the **momentum factors**.
- Gradient Descent with Momenutm (GDM): $\nabla \hat{f}(w; \xi) = \nabla f(w)$.
- Other variants, e.g. Nesterov momentum (GDMN) (Nesterov, 1983).

Coordinate-Wise Step-Sizes

ADAM (Kingma and Ba, 2015): $w_0 \in \mathbb{R}^d$, $v_0 = 0$, $m_0 = 0$,

$$
\begin{aligned}
v_{t+1} &= \gamma_1 v_{t-1} + (1-\gamma_1)\nabla \hat{f}(w_t;\xi_t) \\
m_{t+1} &= \gamma_2 m_{t-1} + (1-\gamma_2)\nabla \hat{f}(w_t;\xi_t)^2 \\
w_{t+1} &= w_t - \eta \frac{\sqrt{1-\gamma_2^t}}{1-\gamma_1^t} \frac{v_t}{\sqrt{m_t + \varepsilon}},
\end{aligned}
$$

- m_t is the **moving average of square gradients**.
- m_t yields **coordinate wise learning rates**.
- Defaults $\gamma_1 = 0.9$, $\gamma_2 = 0.999$ and $\eta = 10^{-3}$ perform well for DNNs.

Advantages of (Stochastic) Gradient-based Methods

- **Effective** even for $n \approx 10^{12}$ or $d \approx 10^{11}$ (i.e. Large Language Models).
- **Implicit regularization**: good expected risk for complex model w/o $\mathcal{R}_\lambda$.
  - Early stopping, i.e. stop before convergence.
  - SGD selects solution with minimal norm at convergence.
- Efficient and automatic $\nabla f(w; \xi)$ via **backpropagation**.
- **Efficiency guarantees in terms of iterations/samples**.

Advantages of (Stochastic) Gradient-based Methods

- **Effective** even for $n \approx 10^{12}$ or $d \approx 10^{11}$ (i.e. Large Language Models).
- **Implicit regularization**: good expected risk for complex model w/o $\mathcal{R}_\lambda$.
  - Early stopping, i.e. stop before convergence.
  - SGD selects solution with minimal norm at convergence.
- Efficient and automatic $\nabla f(w; \xi)$ via **backpropagation**.
- **Efficiency guarantees in terms of iterations/samples**.

Second-order methods, relying on $\nabla^2 f(w; \xi)$, are more expensive and rarely used.

Efficiency of Gradient-based Methods - Assumptions

Let $f : \mathbb{R}^d \to \mathbb{R}$ be **differentiable** on $\mathbb{R}^d$:

- f is L-**Lipschitz smooth** with $L > 0$ if for every $w_1, w_2 \in \mathbb{R}^d$

$$\|\nabla f(w_1) - \nabla f(w_2)\| \leq L\|w_1 - w_2\|$$

- f is μ-**strongly convex** with $\mu > 0$ (**convex** if $\mu = 0$) if for every $w_1, w_2 \in \mathbb{R}^d$

$$f(w_2) \geq f(w_1) + \nabla f(w_1)^\top (w_2 - w_1) + \frac{\mu}{2}\|w_2 - w_1\|^2.$$

$\nabla \hat{f}$ satisfies

$$\mathbb{E}[\nabla \hat{f}(x;\xi)] = \nabla f(x), \qquad \mathrm{Var}[\nabla \hat{f}(x;\xi)] = \mathbb{E}[\|\nabla \hat{f}(x;\xi) - \mathbb{E}[\nabla \hat{f}(x;\xi)]\|^2] \leq \sigma^2$$

Efficiency of Gradient-based Methods - Optimality Measures

$$w^* = \underset{w \in \mathbb{R}^d}{\arg\min} f(w)$$

Optimality Measures (ϵ-accuracy):

- Strongly-Convex: $\mathbb{E}\|w_t - w^*\|^2 \leq \epsilon$
- Convex: $\mathbb{E}[f(w_t) - f(w^*)] \leq \epsilon$
- Non-convex: $\min_{1 \leq s \leq t} \mathbb{E}\|\nabla f(w_s)\|^2 \leq \epsilon$

Efficiency of Gradient-based Methods - Optimality Measures

$$w^* = \arg\min_{w \in \mathbb{R}^d} f(w)$$

Optimality Measures (ϵ-accuracy):

- Strongly-Convex: $\mathbb{E}\|w_t - w^*\|^2 \leq \epsilon$
- Convex: $\mathbb{E}[f(w_t) - f(w^*)] \leq \epsilon$
- Non-convex: $\min_{1 \leq s \leq t} \mathbb{E}\|\nabla f(w_s)\|^2 \leq \epsilon$

How many iterations/samples needed?

Iteration Complexity for Lipschitz Smooth f

	GD	GDMN	SGD
Strongly Convex	$O(\kappa \log(\epsilon^{-1}))$	$O(\sqrt{\kappa} \log(\epsilon^{-1}))$	$O(\kappa\epsilon^{-1})$
Convex	$O(\epsilon^{-1})$	$O(\epsilon^{-1/2})$	$O(\epsilon^{-2})$
non-Convex	$O(\epsilon^{-1})$	$O(\epsilon^{-1})$	$O(\epsilon^{-2})$

# of iterations to reach ϵ-optimality. $\kappa = L/\mu$ with L and μ smoothness and strong convexity constants. GDMN is Nesterov momenutm

- For SGD f can be the **empirical risk** and also the **expected risk**.
- Stochastic Momentum or ADAM do not improve SGD rates.
- GDMN and SGD have **matching lower bounds** in all cases.

Bilevel Problems In Machine Learning

Bilevel Min-Min

$$\min_{\lambda \in \Lambda \subseteq \mathbb{R}^n} f(\lambda) := E(w(\lambda), \lambda)$$

$$w(\lambda) := \arg\min_{w \in \mathbb{R}^d} g(w, \lambda)$$

Bilevel Problem Formulations

Bilevel Min-Min

$$\min_{\lambda \in \Lambda \subseteq \mathbb{R}^n} f(\lambda) := E(w(\lambda), \lambda)$$

$$w(\lambda) := \arg\min_{w \in \mathbb{R}^d} g(w, \lambda)$$

Bilevel Min-Fixed-point

$$\min_{\lambda \in \Lambda \subseteq \mathbb{R}^n} f(\lambda) := E(w(\lambda), \lambda)$$

$$w(\lambda) := \Phi(w(\lambda), \lambda)$$

Bilevel Problem Formulations

Bilevel Min-Min

$$\min_{\lambda \in \Lambda \subseteq \mathbb{R}^n} f(\lambda) := E(w(\lambda), \lambda)$$

$$w(\lambda) := \arg\min_{w \in \mathbb{R}^d} g(w, \lambda)$$

Bilevel Min-Fixed-point

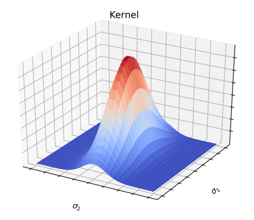
$$\min_{\lambda \in \Lambda \subseteq \mathbb{R}^n} f(\lambda) := E(w(\lambda), \lambda)$$

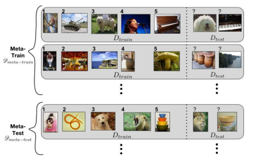
$$w(\lambda) := \Phi(w(\lambda), \lambda)$$

- $w(\lambda)$ is **unique**.
- If g is **convex** and **differentiable** Min-Fixed-point is more general:

$$\Phi(w, \lambda) = w - \nabla_1 g(w, \lambda)$$

Applications in Machine Learning



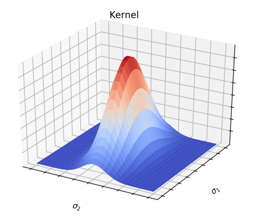


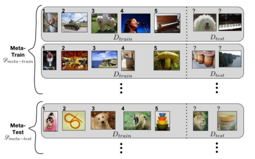
Source: S.Ravi, H. Larochelle (2016).

Min-Min:

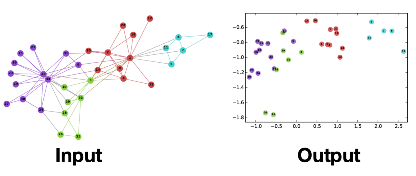
- Hyperparameter optimization (learn the kernel/regularization, ...).
- Meta-learning.
- Data poisoning attacks.
- Others

Applications in Machine Learning





Source: S.Ravi, H. Larochelle (2016).



Source: snap.stanford.edu/proj/embeddings-www

Min-Min:

- Hyperparameter optimization (learn the kernel/regularization, ...).
- Meta-learning.
- Data poisoning attacks.
- Others

Min-Fixed-point:

- Some Graph and Recurrent NNs.
- Equilibrium Models.
- Others

Hyperparameter Optimization (HPO)

Empirical Risk Minimization:

$$\min_{w \in \mathcal{W}} \underbrace{\frac{1}{n} \sum_{(x,y) \in D} \mathcal{L}(h(x; w, \lambda), y)}_{\text{Training risk}} + \mathcal{R}_\lambda(w)$$

- λ are the **hyperparameters** and w are the **parameters** of the model h.
- **hyperparameters**:
  - **Discrete**: # layers, choice of kernel function.
  - **Continuous**: regularization strength ($\mathcal{R}_\lambda(w) = \lambda \|w\|_p^p$), kernel parameters.

Hyperparameter Optimization (HPO)

Empirical Risk Minimization:

$$\min_{w \in \mathcal{W}} \underbrace{\frac{1}{n} \sum_{(x,y)\in D} \mathcal{L}(h(x; w, \lambda), y)}_{\text{Training risk}} + \mathcal{R}_\lambda(w)$$

- λ are the **hyperparameters** and w are the **parameters** of the model h.
- **hyperparameters**:
  - **Discrete**: # layers, choice of kernel function.
  - **Continuous**: regularization strength ($\mathcal{R}_\lambda(w) = \lambda\|w\|_p^p$), kernel parameters.
- How can we set λ?

Hyperparameter Optimization (HPO)

$$\min_{\lambda \in \Lambda} \overbrace{\frac{1}{n'} \sum_{(x,y) \in D'} \mathcal{L}(h(x; w(\lambda), \lambda), y)}^{\text{Validation risk}}$$

$$w(\lambda) = \arg\min_{w \in \mathcal{W}} \underbrace{\frac{1}{n} \sum_{(x,y) \in D} \mathcal{L}(h(x; w, \lambda), y) + \mathcal{R}_\lambda(w)}_{\text{Training risk}}$$

- $D \sim \rho^n$ is the **training** set, $D \sim \rho^{n'}$ is the **validation** set.
- Train-validation split can be replaced by **K-fold cross-validation**.
- Possible **overfitting** on $D' \implies w(\lambda^*)$ is finally tested on **hold-out test** set D''.

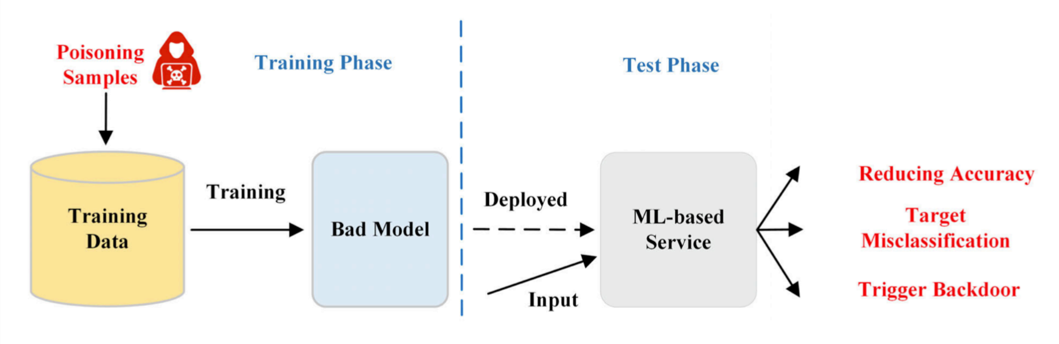
Example: Optimizing the Regularization Hyperparameters in Ridge Regression

$$\min_{\lambda \in \Lambda} \frac{1}{2n'} \|X' w(\lambda) - y'\|_2^2$$

$$w(\lambda) = \arg\min_{w \in \mathbb{R}^d} \left\{ \ell(w, \lambda) := \frac{1}{2n} \|Xw - y\|_2^2 + \underbrace{\frac{1}{2} w^\top B(\lambda) w}_{\text{Regularization}} \right\}$$

- $w(\lambda) = n\big(\frac{1}{n} X^\top X + B(\lambda)\big)^{-1} X y$
- if $\lambda \in \mathbb{R}_{++}$, $B(\lambda) = \lambda I$ we have standard ridge regression.
- if $\lambda \in \mathbb{R}^d$, $B(\lambda) = \text{diag}(\lambda)$ we are selecting/rescaling the features.
- if square-loss is replaced by e.g. cross-entropy, $w(\lambda)$ has **no closed form**.
- Regularization term could be **non-smooth** (e.g. lasso or elastic net)

Data-Poisoning Attacks (DPA)



Source: Liu, Ximeng, et al. "Privacy and security issues in deep learning: A survey." IEEE Access 9 (2020): 4566-4593.

Example: DPA to reduce accuracy

$$\max_{\lambda \in \Lambda \subseteq \mathbb{R}^{n \times d}} \frac{1}{n'} \sum_{(x,y) \in D'} \mathcal{L}(h(x; w(\lambda)), y)$$

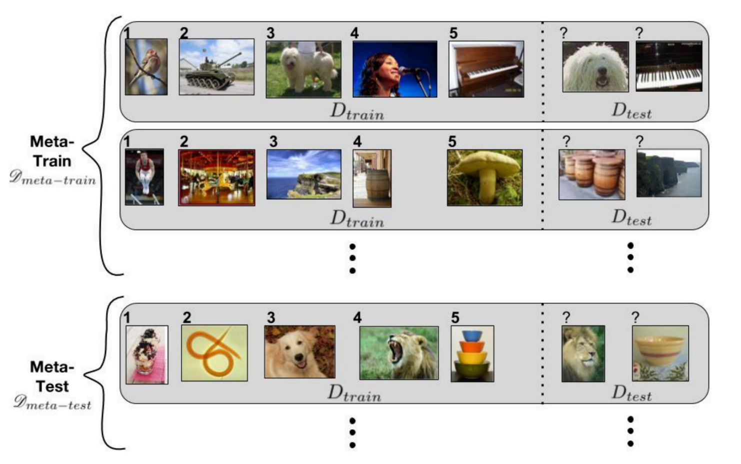
$$w(\lambda) = \arg\min_{w \in \mathbb{R}^{d \times c}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(h(x_i + \lambda_i; w), y_i) + \mathcal{R}_\beta(w)$$

- $\Lambda_i = \mathbf{B}_p(0, \epsilon) = \{\lambda_i \in \mathbb{R}^d : \|\lambda_i\|_p \leq \epsilon\}$ for poisoned samples.
- Smaller ϵ and # of poisoned samples $\implies$ lower chances to be detected.
- Possible **overfitting** on D' $\implies$ $w(\lambda)$ is also tested on **hold-out test** set D''.

Biggio, Battista, et al. ”Poisoning attacks against support vector machines.” ICML 2012.

Muñoz-González, Luis, et al. ”Towards poisoning of deep learning algorithms with back-gradient optimization.” ACM workshop on artificial intelligence and security. 2017.

Meta-learning (MTL): learn meta-model working well on new related tasks



Source: S.Ravi, H. Larochelle (2016).

Example: Meta-training a shared meta-representation

$$\min_{\lambda \in \mathbb{R}^d} \frac{1}{n_{\text{tasks}}} \sum_{i=1}^{n_{\text{tasks}}} \frac{1}{|D^i_{\text{test}}|} \sum_{(x,y)\in D^i_{\text{test}}} \mathcal{L}^i(w(\lambda)_i^\top \phi(x;\lambda), y)$$

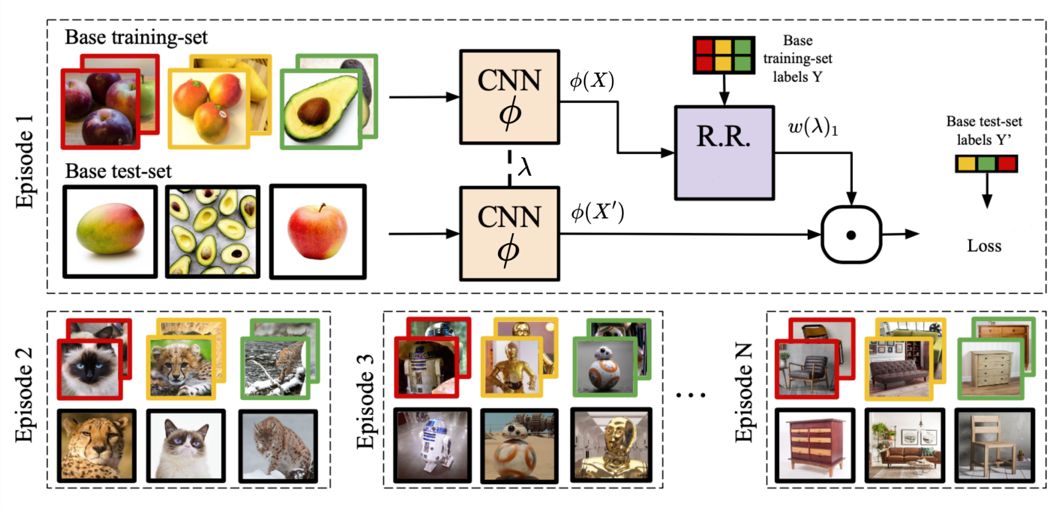
$$w(\lambda)_i = \arg\min_{w\in\mathbb{R}^d} \frac{1}{|D^i_{\text{train}}|} \sum_{(x,y)\in D^i_{\text{train}}} \mathcal{L}^i_{\text{train}}(w^\top \phi(x;\lambda), y)$$

- ϕ is the **meta-representation** (can be a neural network) with parameters λ.
- $w(\lambda)_i$ is the **linear model** adapted to task i (to D^i_{train}).
- n_{tasks} separate lower-level problems.
- λ^* is then tested on hold-out tasks (**meta-test**).

Franceschi, Luca, et al. "Bilevel programming for hyperparameter optimization and meta-learning." ICML 2018.

Bertinetto, L., et al. "Meta-learning with differentiable closed-form solvers." ICLR 2019.

Lee, K., et al. "Meta-learning with differentiable convex optimization." CVPR 2019. APA



Meta-Training

Modified from Bertinetto et al. (2019)

Example: Deep Equilibrium Models (Bai et al. NeurIPS 2019)

Explicit Layer

$$x \longrightarrow \boxed{z_1 = h_\theta(x)} \to \boxed{z_2 = h_\theta(z_1)} \longrightarrow \cdots \longrightarrow \boxed{z_L = h_\theta(z_{L-1})} \longrightarrow z_L \longrightarrow \ell(z_L, y)$$

$$\frac{d}{d\theta}\ell(z_L, y) \xleftarrow{\quad\quad\quad\quad}$$

Backprop

Example: Deep Equilibrium Models (Bai et al. NeurIPS 2019)

Explicit Layer

$$x \longrightarrow \boxed{z_1 = h_\theta(x)} \rightarrow \boxed{z_2 = h_\theta(z_1)} \longrightarrow \cdots \longrightarrow \boxed{z_L = h_\theta(z_{L-1})} \longrightarrow z_L \longrightarrow \ell(z_L, y)$$

$$\frac{d}{d\theta}\ell(z_L, y) \leftarrow$$

Backprop

Equilibrium Layer

$$x \longrightarrow \boxed{z^* = g_\theta(z^*, x)} \longrightarrow z^* \longrightarrow \ell(z^*, y)$$

$$\frac{d}{d\theta}\ell(z^*, y) \leftarrow$$

Implicit Backprop

Example: Deep Equilibrium Models (Bai et al. NeurIPS 2019)

Explicit Layer

$$x \longrightarrow \boxed{z_1 = h_\theta(x)} \rightarrow \boxed{z_2 = h_\theta(z_1)} \longrightarrow \cdots \longrightarrow \boxed{z_L = h_\theta(z_{L-1})} \longrightarrow z_L \longrightarrow \ell(z_L, y)$$

$$\frac{d}{d\theta}\ell(z_L, y) \leftarrow$$

Backprop

Equilibrium Layer

$$x \longrightarrow \boxed{z^* = g_\theta(z^*, x)} \longrightarrow z^* \longrightarrow \ell(z^*, y)$$

$$\frac{d}{d\theta}\ell(z^*, y) \leftarrow$$

Implicit Backprop

E.g. $h_\theta(z) = \sigma(Wz + b)$, $\theta = (W, b)$, $\quad g_\theta(z, x) = \sigma(W_1 z + W_2 x + b)$, $\theta = (W_1, W_2, b)$

Equilibrium Models: Bilevel Formulation

$$\min_{(\theta,w)\in\mathbb{R}^n} \sum_{(x,y)\in D} \mathcal{L}(w^\top z(x,\theta), y) \qquad z(x,\theta) = g_\theta(z(x,\theta), x)$$

- Upper-level is **ERM** on D. (θ^*, w^*) is tested on a hold out set D'.
- One lower-level fixed-point problem per training example (x, y).
- Structure resembles meta-learning: separable lower-level.

Common Traits

- **Large scale**: large # of examples, # of parameters or # of hyperparameters.
- **Simple or no constraints at the lower-level**:
  - ERM is usually an unconstrained minimization problem ($\mathcal{W} = \mathbb{R}^d$).
- f is not the final objective (**overfitting**). Final test on **hold-out data**.
- Analysis often assumes existence and **uniqueness of the lower-level solution**.

Summary of Part 1

Supervised Learning

- Many machine learning problems fit in **supervised Learning**.
- No access to data distribution $\implies$ learning = **ERM**.
- ERM requires **efficient optimization algorithms** (**SGD, SGMD, ADAM**).

Bilevel Applications

- Several machine-learning problems are **bilevel problems**.
- Large scale, simple/no constraints, possible overfitting.

Automatic Differentiation

Differentiating Complex Functions

Let $f : \mathbb{R}^d \to \mathbb{R}^n$ be a **vector-valued** function

and assume we have a **computer program** that outputs $f(x)$ from the input x:

- f can be a **neural network** with several layers.
- $f(x)$ can be the **output of an optimization algorithm**.

The analytic expression for f might be complicated and its derivatives even more.

How can we efficiently differentiate f?

Symbolic Differentiation

Apply the **chain rule to the symbolic expression of** f (e.g. Mathematica).

Issues:

- Resulting expression might be complex and contain redundant parts.
- It doesn't tell you how to compute the derivative efficiently.

Problematic Example:

$$f(x) = \prod_{i=1}^{d} x_i = x_1 x_2 \cdots x_d, \qquad \nabla f(x)_i = \prod_{j \neq i} x_j = x_1 \cdots x_{i-1} x_{i+1} \cdots x_d$$

Finite Differences

Let $\epsilon > 0$, and compute the directional of $f(x)$ along the direction $h \in \mathbb{R}^d$ as:

$$\frac{f(x+\epsilon h) - f(x)}{\epsilon} \qquad \text{or} \qquad \frac{f(x+\epsilon h) - f(x-\epsilon h)}{2\epsilon}$$

- Small $\epsilon \implies$ **cancellation error** due to finite precision arithmetic.
- Large $\epsilon \implies$ **truncation error:** we are further from the derivative definition.
- Can have high error even with optimal ϵ.
- Full derivative requires **at least d function evaluations**.

Automatic Differentiation (AD) (Griewank and Walther, 2008)

Idea: Use the evaluation program of f to derive a program for its derivative.

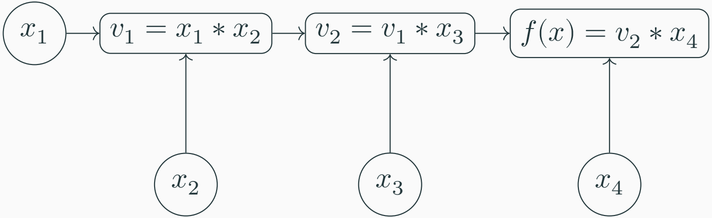
Advantages:

- **Exact derivative** in infinite precision (No **truncation error**).
- Derivative program generated **automatically**.
- $f(x) \in \mathbb{R} \implies \text{Time}(\nabla f(x)) = O(\text{Time}(f(x)))$ for **reverse mode AD**.
- **Open source implementations** (Pytorch, JAX, Tensorflow, ...).

Disadvantages:

- **Implementation overhead**: might be slow for simple functions.
- Reverse mode AD can have a **high memory cost**.

Computational Graph



Graph for $f(x) = \prod_{i=1}^{4}, x_i, \quad v_i = \psi_i(v_{i-1}, v_{-i}) = v_{i-1} * x_{i+1}$.

- **Input variables**: $(v_0, v_{-1}, \dots, v_{1-d}) = (x_1, x_2, \dots, x_d)$.
- **Auxiliary variables**: $v_i = \psi_i(u_i)$, $i > 0$.
- $u_i = (v_j)_{j \prec i}$ are the **parent nodes** of v_i.
- ψ_i are **primitive** operations (e.g. $x + y, x * y, x^{-1}$, $Ax + b$).

Let the **Jacobian** of $f : \mathbb{R}^d \to \mathbb{R}^m$ in x be

$$Df(x) := \begin{bmatrix} \partial_1 f_1(x) & \cdots & \partial_d f_1(x) \\ \vdots & \ddots & \vdots \\ \partial_1 f_m(x) & \cdots & \partial_d f_m(x) \end{bmatrix} \in \mathbb{R}^{m \times d}$$

AD provides **efficient** ways to compute the **Jacobian-vector products**

$$Df(x)\,\dot{x} \quad \text{(Forward AD)}, \qquad Df(x)^\top\,\overline{y} \quad \text{(Reverse AD)}.$$

where $\dot{x} \in \mathbb{R}^d$ and $\bar{y} \in \mathbb{R}^m$.

Relies on efficient **Jacobian-vector products** for the primitives:

$$D\psi_i(u_i)\,\dot{u}_i \quad \text{(Forward AD)}, \qquad D\psi_i(u_i)^\top\,\bar{v}_i \quad \text{(Reverse AD)}.$$

Evaluation Program of $f(x)$

1: $(v_{1-d}, \dots, v_0) = x$
2: **for** $i = 1, \dots, l$ **do**
3: $\quad v_i = \psi_i(u_i) \in \mathbb{R}$ for simplicity
4: **end for**
5: **return** $(v_{l-m}, \dots, v_l)^\top$

Let $w_i := (v_{1-d}, \dots, v_i)^\top$,

$$w_i = g_i(w_{i-1}) := \begin{pmatrix} w_{i-1} \\ \psi_i(u_i) \end{pmatrix} = \begin{pmatrix} w_{i-1} \\ v_i \end{pmatrix} \qquad \forall i \in \{1, \dots, l\}$$

The program can be seen as computing from **right to left**

$$f(x) = P \circ g_l \circ g_{l-1} \circ \cdots \circ g_1(\underbrace{w_0}_{x}) \qquad P w_l = (v_{l-m}, \dots, v_l)^\top$$

Forward Mode AD

The **chain rule** on $f(x) = Pg_l(g_{l-1}(\cdots g_1(w_0)\cdots))$ yields

$$Df(x)\dot{x} = PDg_l(w_{l-1})\cdots Dg_2(w_1)Dg_1(w_0)\underbrace{\dot{w}_0}_{\dot{x}}$$

Forward mode AD computes for $i = 1, \ldots, l$

$$\begin{aligned} \dot{w}_i &= Dg_i(w_{i-1})\dot{w}_{i-1} = (\dot{w}_{i-1}, \dot{v}_i)^\top \\ g_i \text{ stucture} \implies \dot{v}_i &= D\psi_i(u_i)\dot{u}_i = \sum_{j \prec i} \partial_j \psi_i(u_i)\dot{v}_j \end{aligned}$$

where $(\dot{v}_{1-d}, \ldots, \dot{v}_0) := (\dot{x}_1, \ldots, \dot{x}_d)$ and $\dot{u}_i = (\dot{v}_j)_{j \prec i}$

Reverse Mode AD

The **chain rule** on $f(x) = Pg_l(g_{l-1}(\cdots g_1(w_0)\cdots))$ yields

$$Df(x)^\top \bar{y} = Dg_1(w_0)^\top \cdots Dg_{l-1}(w_{l-2})^\top Dg_l(w_{l-1})^\top \underbrace{\bar{w}_l}_{P^\top \bar{y}},$$

Adjoint variables: $(\bar{v}_{1-d}, \ldots \bar{v}_l)$, with $\bar{w}_i = (\bar{v}_{1-d}, \ldots \bar{v}_i)$ and $\bar{u}_i = (\bar{v}_j)_{j \prec i}$.

Reverse Mode AD first sets $\bar{w}_l = P^\top \bar{y} = (0, \ldots, 0, \bar{y}_1, \ldots, \bar{y}_m)$.

Then updates the adjoint variables by computing for $i = l, \ldots, 1$

$$\begin{aligned} & & \bar{w}_{i-1} &= Dg_i(w_{i-1})^\top \bar{w}_i \\ g_i \text{ structure} & \implies & \bar{u}_i &= \bar{u}_i + D\psi_i(u_i)^\top \bar{v}_i \\ & \implies & \bar{v}_j &= \bar{v}_j + \partial_j \psi_i(u_i) \bar{v}_i \quad \forall j \prec i \end{aligned}$$

Algorithm Forward AD

Input: $x,\ \dot{x} \in \mathbb{R}^d$

Output: $f(x),\ Df(x)\,\dot{x}$

1: $(v_0, \dots, v_{1-d}) = x$, $(\dot{v}_0, \dots, \dot{v}_{1-d}) = \dot{x}$
2: **for** $i = 1, \dots, l$ **do** *# forward pass*
3: $\quad v_i = \psi_i(u_i)$, $\dot{v}_i = D\psi_i(u_i)\,\dot{u}_i$
4: **end for**
5: **return** $(v_{l-m}, \dots, v_l)$, $(\dot{v}_{l-m}, \dots, \dot{v}_l)$

Where $u_i = (v_j)_{j \prec i}$, $\dot{u}_i = (\dot{v}_j)_{j \prec i}$.

Algorithm Forward AD

Input: $x,\ \dot{x} \in \mathbb{R}^d$

Output: $f(x),\ Df(x)\,\dot{x}$

$(v_0, \dots, v_{1-d}) = x,\ (\dot{v}_0, \dots, \dot{v}_{1-d}) = \dot{x}$
for $i = 1, \dots, l$ **do** *# forward pass*
  $v_i = \psi_i(u_i),\ \dot{v}_i = D\psi_i(u_i)\,\dot{u}_i$
end for
return $(v_{l-m}, \dots, v_l),\ (\dot{v}_{l-m}, \dots, \dot{v}_l)$

Where $u_i = (v_j)_{j \prec i},\ \dot{u}_i = (\dot{v}_j)_{j \prec i}$.

Algorithm Reverse AD

Input: $x \in \mathbb{R}^d,\ \overline{y} \in \mathbb{R}^m$

Output: $f(x),\ Df(x)^\top\,\overline{y}$

$(v_0, \dots, v_{1-d}) = x$
$(\overline{v}_{1-d}, \dots, \overline{v}_l) = (0, \dots, 0, \overline{y}_1, \dots, \overline{y}_m)$
for $i = 1, \dots, l$ **do** *# forward pass*
  $v_i = \psi_i(u_i)$
end for
for $i = l, \dots, 1$ **do** *# backward pass*
  $\overline{u}_i = \overline{u}_i + D\psi_i(u_i)^\top\,\overline{v}_i$
end for
return $(v_{l-m}, \dots, v_l),\ (\overline{v}_{1-d}, \dots, \overline{v}_0)$

Where $u_i = (v_j)_{j \prec i},\ \overline{u}_i = (\overline{v}_j)_{j \prec i}$.

Time and Space Complexity

For each $\psi_i(u_i)$, **only one Jacobian-vector product** of ψ_i is computed

$+ \quad \text{Time}(D\psi_i(u_i)\dot{u}_i) = O(\text{Time}(\psi_i(u_i))) = \text{Time}(D\psi_i(u_i)^\top \bar{v}_i) \implies$

Time and Space Complexity

For each $\psi_i(u_i)$, **only one Jacobian-vector product** of ψ_i is computed

$+ \quad \mathsf{Time}(D\psi_i(u_i)\dot{u}_i) = O(\mathsf{Time}(\psi_i(u_i))) = \mathsf{Time}(D\psi_i(u_i)^\top \bar{v}_i) \implies$

Time Complexity

$$\mathsf{Time}(Df(x)\dot{x}) = O(\mathsf{Time}(f(x))) = \mathsf{Time}(Df(x)^\top \bar{y})$$

Cheap gradient: $f(x) \in \mathbb{R}, \bar{y} = 1 \implies \mathsf{Time}(\nabla f(x)) = O(\mathsf{Time}(f(x)))$

Time and Space Complexity

For each $\psi_i(u_i)$, **only one Jacobian-vector product** of ψ_i is computed

$+ \quad \text{Time}(D\psi_i(u_i)\dot{u}_i) = O(\text{Time}(\psi_i(u_i))) = \text{Time}(D\psi_i(u_i)^\top \bar{v}_i) \implies$

Time Complexity

$$\text{Time}(Df(x)\dot{x}) = O(\text{Time}(f(x))) = \text{Time}(Df(x)^\top \bar{y})$$

Cheap gradient: $f(x) \in \mathbb{R}, \bar{y} = 1 \implies \text{Time}(\nabla f(x)) = O(\text{Time}(f(x)))$

Space Complexity

- $\text{Mem}(Df(x)\dot{x}) = O(\text{Mem}(f(x)))$ (delete unused v_i-s).
- $\text{Mem}(Df(x)^\top \bar{y}) = O(l)$ **(needs to store all v_i-s)**.

Checkpointing and Second Order Derivatives

Checkpointing. Define higher-level primitives by composing primitives:

- **Saves memory** for reverse AD by reducing the number of auxiliary variables.
- **Increases time** since **reverse mode AD is applied recursively**.

Checkpointing and Second Order Derivatives

Checkpointing. Define higher-level primitives by composing primitives:

- **Saves memory** for reverse AD by reducing the number of auxiliary variables.
- **Increases time** since **reverse mode AD is applied recursively**.

Hessian-vector products. $f : \mathbb{R}^d \to \mathbb{R}$

- RR: **(1)** $\nabla f(x)$ (reverse) **(2)** $\partial_x(x \mapsto \nabla f(x))^\top v = \nabla^2 f(x)\, v$ (reverse)
- RF: **(1)** $\nabla f(x)^\top v$ (forward) **(2)** $\partial_x(x \mapsto \nabla f(x)\, v) = \nabla^2 f(x)\, v$ (reverse)
- FR: **(1)** $\nabla f(x)$ (reverse) **(2)** $\partial_x(x \mapsto \nabla f(x))\, v = \nabla^2 f(x)\, v$ (forward)
- $\text{Time}(\nabla^2 f(x)^\top v) = O(\text{Time}(f(x)))$, $\quad \text{Mem}(RR) \geq \text{Mem}(RF) \geq \text{Mem}(FR)$.

Checkpointing and Second Order Derivatives

Checkpointing. Define higher-level primitives by composing primitives:

- **Saves memory** for reverse AD by reducing the number of auxiliary variables.
- **Increases time** since **reverse mode AD is applied recursively**.

Hessian-vector products. $f : \mathbb{R}^d \to \mathbb{R}$

- RR: **(1)** $\nabla f(x)$ (reverse) **(2)** $\partial_x (x \mapsto \nabla f(x))^\top v = \nabla^2 f(x)\, v$ (reverse)
- RF: **(1)** $\nabla f(x)^\top v$ (forward) **(2)** $\partial_x (x \mapsto \nabla f(x)\, v) = \nabla^2 f(x)\, v$ (reverse)
- FR: **(1)** $\nabla f(x)$ (reverse) **(2)** $\partial_x (x \mapsto \nabla f(x))\, v = \nabla^2 f(x)\, v$ (forward)
- $\text{Time}(\nabla^2 f(x)^\top v) = O(\text{Time}(f(x)))$, $\quad \text{Mem}(RR) \geq \text{Mem}(RF) \geq \text{Mem}(FR)$.

f **defined implicitly?** (Griewank and Walther, 2008, Chap. 15) and hypergradients.

The Hypergradient

Strategies to Solve the Bilevel Problem

$$\min_{\lambda \in \Lambda \subseteq \mathbb{R}^m} f(\lambda) := E(w(\lambda), \lambda) \quad \text{(UL)}$$

$$\overbrace{w(\lambda) := \Phi(w(\lambda), \lambda)}^{\text{Min-Fixed-point}} \quad \text{or} \quad \overbrace{w(\lambda) := \arg\min_{w \in \mathbb{R}^d} g(w, \lambda)}^{\text{Min-Min}} \quad \text{(LL)}$$

How can we solve it?

- Black-box methods (random/grid search, Bayesian optimization, ...).

Strategies to Solve the Bilevel Problem

$$\min_{\lambda \in \Lambda \subseteq \mathbb{R}^m} f(\lambda) := E(w(\lambda), \lambda) \quad \textbf{(UL)}$$

$$\overbrace{w(\lambda) := \Phi(w(\lambda), \lambda)}^{\text{Min-Fixed-point}} \quad \text{or} \quad \overbrace{w(\lambda) := \arg\min_{w \in \mathbb{R}^d} g(w, \lambda)}^{\text{Min-Min}} \quad \textbf{(LL)}$$

How can we solve it?

- Black-box methods (random/grid search, Bayesian optimization, ...).
- Gradient-based methods exploiting the ***hypergradient*** $\nabla f(\lambda)$.
  Work best when m is large and f is smooth.

Strategies to Solve the Bilevel Problem

$$\min_{\lambda \in \Lambda \subseteq \mathbb{R}^m} f(\lambda) := E(w(\lambda), \lambda) \quad \textbf{(UL)}$$

$$\overbrace{w(\lambda) := \Phi(w(\lambda), \lambda)}^{\text{Min-Fixed-point}} \quad \text{or} \quad \overbrace{w(\lambda) := \arg\min_{w \in \mathbb{R}^d} g(w, \lambda)}^{\text{Min-Min}} \quad \textbf{(LL)}$$

How can we solve it?

- Black-box methods (random/grid search, Bayesian optimization, ...).
- Gradient-based methods exploiting the ***hypergradient*** $\nabla f(\lambda)$.
  Work best when m is large and f is smooth.
- Value-function approaches (no need for $\nabla f(\lambda)$).

Differentiability of f

Assumptions

- E, Φ (or $\nabla_1 g$) are **cont. differentiable** in an open set containing $\Lambda \times \mathbb{R}^d$.
- $I - \partial_1 \Phi(w(\lambda), \lambda)$ (or $\nabla_1^2 g(w(\lambda), \lambda)$) is **invertible** for every $\lambda \in \Lambda$.

$\implies$ f is **differentiable** thanks to the implicit function theorem.

From the **chain rule** we obtain

$$\nabla f(\lambda) = \frac{d}{d\lambda} E(w(\lambda), \lambda) = \nabla_2 E(w(\lambda), \lambda) + w'(\lambda)^\top \nabla_1 E(w(\lambda), \lambda)$$

Min-Fixed-point $w'(\lambda)$ Derivation

$$
\begin{aligned}
w(\lambda) &= \Phi(w(\lambda), \lambda) \\
&\Big\Downarrow \text{Differentiate both sides using the chain rule} \\
w'(\lambda) &= \partial_1 \Phi(w(\lambda), \lambda) w'(\lambda) + \partial_2 \Phi(w(\lambda), \lambda) \\
&\Big\Downarrow \text{Rearrange terms} \\
(I - \partial_1 \Phi(w(\lambda), \lambda)) w'(\lambda) &= \partial_2 \Phi(w(\lambda), \lambda) \\
&\Big\Downarrow (I - \partial_1 \Phi(w(\lambda), \lambda)) \text{ is invertible} \\
w'(\lambda) &= (I - \partial_1 \Phi(w(\lambda), \lambda))^{-1} \partial_2 \Phi(w(\lambda), \lambda)
\end{aligned}
$$

Min-Min $w'(\lambda)$ Derivation

$$w(\lambda) = \arg\min_{w \in \mathbb{R}^d} g(w, \lambda)$$

$\Downarrow$ g is differentiable in w

$$0 = \nabla_1 g(w(\lambda), \lambda)$$

$\Downarrow$ Differentiating both sides using the chain rule

$$0 = \nabla_1^2 g(w(\lambda), \lambda) w'(\lambda) + \nabla_{12} g(w(\lambda), \lambda)$$

$\Downarrow$ $\nabla_1^2 g(w(\lambda), \lambda)$ is invertible

$$w'(\lambda) = -\nabla_1^2 g(w(\lambda), \lambda)^{-1} \nabla_{12} g(w(\lambda), \lambda)$$

An Implicit Expression for the hypergradient

$$\nabla f(\lambda) = \nabla_2 E(w(\lambda), \lambda) + w'(\lambda)^\top \nabla_1 E(w(\lambda), \lambda)$$

$$w'(\lambda) = (I - \partial_1 \Phi(w(\lambda), \lambda))^{-1} \partial_2 \Phi(w(\lambda), \lambda).$$

$$w'(\lambda) = -\nabla_1^2 g(w(\lambda), \lambda)^{-1} \nabla_{12} g(w(\lambda), \lambda)$$

Min-Min $w'(\lambda)$ $=$ Min-Fixed-point $w'(\lambda)$ + $\Phi(w, \lambda) = w - \nabla_1 g(w, \lambda).$

An Implicit Expression for the hypergradient

$$\nabla f(\lambda) = \nabla_2 E(w(\lambda), \lambda) + w'(\lambda)^\top \nabla_1 E(w(\lambda), \lambda)$$

$$w'(\lambda) = (I - \partial_1 \Phi(w(\lambda), \lambda))^{-1} \partial_2 \Phi(w(\lambda), \lambda).$$

$$w'(\lambda) = -\nabla_1^2 g(w(\lambda), \lambda)^{-1} \nabla_{12} g(w(\lambda), \lambda)$$

Min-Min $w'(\lambda)$ = **Min-Fixed-point** $w'(\lambda)$ + $\Phi(w, \lambda) = w - \nabla_1 g(w, \lambda).$

$$\nabla f(\lambda) = \underbrace{\nabla_2 E(w(\lambda), \lambda)}_{n} + \underbrace{\partial_2 \Phi(w(\lambda), \lambda)^\top}_{n \times d} (I - \underbrace{\partial_1 \Phi(w(\lambda), \lambda)^\top}_{d \times d})^{-1} \underbrace{\nabla_1 E(w(\lambda), \lambda)}_{d}$$

$$\nabla f(\lambda) = \underbrace{\nabla_2 E(w(\lambda), \lambda)}_{n} - \underbrace{\nabla_{12} g(w(\lambda), \lambda)^\top}_{n \times d} \underbrace{\nabla_1^2 g(w(\lambda), \lambda)^{-1}}_{d \times d} \underbrace{\nabla_1 E(w(\lambda), \lambda)}_{d}$$

An Implicit Expression for the hypergradient

$$\nabla f(\lambda) = \nabla_2 E(w(\lambda), \lambda) + w'(\lambda)^\top \nabla_1 E(w(\lambda), \lambda)$$

$$w'(\lambda) = (I - \partial_1 \Phi(w(\lambda), \lambda))^{-1} \partial_2 \Phi(w(\lambda), \lambda).$$

$$w'(\lambda) = -\nabla_1^2 g(w(\lambda), \lambda)^{-1} \nabla_{12} g(w(\lambda), \lambda)$$

Min-Min $w'(\lambda)$ = **Min-Fixed-point** $w'(\lambda)$ + $\Phi(w, \lambda) = w - \nabla_1 g(w, \lambda).$

$$\nabla f(\lambda) = \underbrace{\nabla_2 E(w(\lambda), \lambda)}_{n} + \underbrace{\partial_2 \Phi(w(\lambda), \lambda)^\top}_{n \times d} (I - \underbrace{\partial_1 \Phi(w(\lambda), \lambda)^\top}_{d \times d})^{-1} \underbrace{\nabla_1 E(w(\lambda), \lambda)}_{d}$$

$$\nabla f(\lambda) = \underbrace{\nabla_2 E(w(\lambda), \lambda)}_{n} - \underbrace{\nabla_{12} g(w(\lambda), \lambda)^\top}_{n \times d} \underbrace{\nabla_1^2 g(w(\lambda), \lambda)^{-1}}_{d \times d} \underbrace{\nabla_1 E(w(\lambda), \lambda)}_{d}$$

With d or m large, $\nabla f(\lambda)$ can be expensive to compute even when $w(\lambda)$ is given

Hypegradient for the Ridge Regression Hyperparameter

$$E(w,\lambda) = \frac{1}{2n'}\|X'w - y'\|^2, \quad g(w,\lambda) = \frac{1}{2n}\|Xw - y\|^2 + \frac{1}{2}\lambda\|w\|^2,$$

Hypegradient for the Ridge Regression Hyperparameter

$$E(w,\lambda) = \frac{1}{2n'}\|X'w - y'\|^2, \quad g(w,\lambda) = \frac{1}{2n}\|Xw - y\|^2 + \frac{1}{2}\lambda\|w\|^2,$$

$$\nabla_1 E(w,\lambda) = \frac{1}{n'}X'^{\top}(X'w - y'), \quad \nabla_2 E(w,\lambda) = 0,$$

$$\nabla_1 g(w,\lambda) = \frac{1}{n}X^{\top}(Xw - y) + \lambda w$$

$$\nabla_1^2 g(w,\lambda) = \frac{1}{n}X^{\top}X + \lambda I, \qquad \nabla_{1,2} g(w,\lambda) = w$$

Hypegradient for the Ridge Regression Hyperparameter

$$E(w,\lambda) = \frac{1}{2n'}\|X'w - y'\|^2, \quad g(w,\lambda) = \frac{1}{2n}\|Xw - y\|^2 + \frac{1}{2}\lambda\|w\|^2,$$

$$\nabla_1 E(w,\lambda) = \frac{1}{n'}X'^\top(X'w - y'), \quad \nabla_2 E(w,\lambda) = 0,$$

$$\nabla_1 g(w,\lambda) = \frac{1}{n}X^\top(Xw - y) + \lambda w$$

$$\nabla_1^2 g(w,\lambda) = \frac{1}{n}X^\top X + \lambda I, \qquad \nabla_{1,2} g(w,\lambda) = w$$

$$\nabla f(\lambda) = -w(\lambda)^\top\left(\frac{1}{n}X^\top X + \lambda I\right)^{-1}\frac{1}{n'}X'^\top(X'w(\lambda) - y')$$

$$w(\lambda) = n\left(\frac{1}{n}X^\top X + \lambda I\right)^{-1}Xy$$

Discussion on The Assumptions

Uniqueness of $w(\lambda)$:

- If $g(\cdot, \lambda)$ is strongly convex: many L2-regularized problems.
- $\Phi(\cdot, \lambda)$ is a contraction: GD on strongly convex Lip. smooth objectives.
- Not true for Overparametrized models and NNs without regularization.

Differentiability of E, Φ, and twice for g: smooth loss, regularizer and model.

Invertibility of $I - \partial_1 \Phi(w(\lambda), \lambda)$ or $\nabla_1^2 g(w(\lambda), \lambda)$:

- When $\Phi(\cdot, \lambda)$ is a **differentiable contraction** near $w(\lambda)$.
- When $g(\cdot, \lambda)$ is **twice differentiable, strongly convex, Lip. smooth** near $w(\lambda)$.

Hypergradient Approximation

Approximating the Lower-level

LL solution $w(\lambda)$ can be **expensive** to evaluate accurately (e.g. ERM with large n)

Idea: Replace $w(\lambda)$ with

$$w_t(\lambda) := \mathcal{A}_t(w_0, \lambda), \qquad \lim_{t \to \infty} w_t(\lambda) = w(\lambda),$$

$\mathcal{A}_t : \mathbb{R}^d \times \Lambda \to \mathbb{R}^d$ is the iterative **lower-level solver** which starts from $w_0(\lambda) = w_0$.

LL solution $w(\lambda)$ can be **expensive** to evaluate accurately (e.g. ERM with large n)

Idea: Replace $w(\lambda)$ with

$$w_t(\lambda) := \mathcal{A}_t(w_0, \lambda), \qquad \lim_{t\to\infty} w_t(\lambda) = w(\lambda),$$

$\mathcal{A}_t : \mathbb{R}^d \times \Lambda \to \mathbb{R}^d$ is the iterative **lower-level solver** which starts from $w_0(\lambda) = w_0$.

Examples:

$$\textbf{(FP)} \qquad w_t(\lambda) = \Phi(w_{t-1}(\lambda), \lambda)$$

$$\textbf{(GD)} \qquad w_t(\lambda) = w_{t-1}(\lambda) - \eta_t \nabla_1 g(w_{t-1}(\lambda), \lambda)$$

or **SGD, SGMD, ADAM, ...**

Approximate Bilevel Problem

Exact Bilevel		Approximate Bilevel
$\min_{\lambda \in \Lambda} f(\lambda) := E(w(\lambda), \lambda)$	$\Rightarrow$	$\min_{\lambda \in \Lambda} f_t(\lambda) := E(w_t(\lambda), \lambda)$
$w(\lambda) = \Phi(w(\lambda), \lambda)$		$w_t(\lambda) = \mathcal{A}_t(w_0, \lambda)$

- E, $\mathcal{A}_t$ differentiable $\implies$ f_t differentiable.
- Mild assumptions: $\arg\min_{\lambda \in \Lambda} f_t(\lambda) \underset{t \to \infty}{\to} \arg\min_{\lambda \in \Lambda} f(\lambda)$ (Franceschi et al., 2018).
- Allows to **optimize solver's parameters** like learning rate, t or w_0.
- **They are both valid frameworks for machine learning problems.**

Iterative Differentiation (ITD)

1. $w_t(\lambda) = \mathcal{A}_t(w_0, \lambda)$, where $\mathcal{A}_t$ differentiable
2. $f_t(\lambda) = E(w_t(\lambda), \lambda)$
3. Get $\nabla f_t(\lambda)$ *efficiently* using **backpropagation** (reverse mode AD).

1. $w_t(\lambda) = \mathcal{A}_t(w_0, \lambda)$, where $\mathcal{A}_t$ differentiable
2. $f_t(\lambda) = E(w_t(\lambda), \lambda)$
3. Get $\nabla f_t(\lambda)$ *efficiently* using **backpropagation** (reverse mode AD).

- Also called **Unrolling**: the compuation in $\mathcal{A}_t$ need to be saved in memory.
- **Checkpoining**: store only $(w_i(\lambda))_{i=0}^t$ for the **backward pass**.
- if $\lambda \in \mathbb{R}^m$ with small m, might use **forward mode AD** to save memory.

Iterative Differentiation In Pytorch

```
lmbd = randn(..., requires_grad=True)
w0 = zeros(..., requires_grad=True)
t = ...

wt = ll_alg(t, w0, lmbd, phi)  # solve the lower-level

ft = E(wt, lmbd)
ft.backward()  # hypergradient in lmbd.grad
```

AID Preliminaries: The Linear System

$$\nabla f(\lambda) = \nabla_2 E(w(\lambda),\lambda) + \partial_2 \Phi(w(\lambda),\lambda)^\top \underbrace{(I - \partial_1 \Phi(w(\lambda),\lambda)^\top)^{-1} \nabla_1 E(w(\lambda),\lambda)}_{v(w(\lambda),\lambda)}$$

where $v(w,\lambda)$ is the solution of the **linear system**

$$\underbrace{I - \partial_1 \Phi(w,\lambda)^\top}_{A \in \mathbb{R}^{d\times d}} x = \underbrace{\nabla_1 E(w,\lambda)}_{b \in \mathbb{R}^d}$$

Recall d is the number of lower-level parameters

Solving the Linear System

$v(w, \lambda)$ can be **expensive** to compute accurately for large d:

- storing A costs $\Theta(d^2)$ in memory, which might be prohibitive
- with A in memory: time is usually $O(d^3)$ (product $A^\top v$ is $O(d^2)$).

Idea: Replace $v(w, \lambda)$ with

$$v_k(\lambda) = \mathcal{B}_k(v_0, w, \lambda), \qquad \lim_{k \to \infty} v_k(\lambda) = v(w, \lambda)$$

$\mathcal{B}_k$ is a **solver for the linear system** starting from v_0: E.g. Conjugate Gradient.

Approximate Implict Differentiation (AID)

1. $w_t(\lambda) = \mathcal{A}_t(w_0, \lambda)$
2. $v_k(\lambda) = \mathcal{B}_k(v_0, w_t(\lambda), \lambda)$. $\mathcal{B}_k$ is a solver for the **linear system**

$$(I - \partial_1 \Phi(w_t(\lambda), \lambda)^\top) v = \nabla_1 E(w_t(\lambda), \lambda)$$

3. $\hat{\nabla} f(\lambda) = \nabla_2 E(w_t(\lambda), \lambda) + \partial_2 \Phi(w_t(\lambda), \lambda)^\top v_k(\lambda)$

Approximate Implict Differentiation (AID)

1. $w_t(\lambda) = \mathcal{A}_t(w_0, \lambda)$
2. $v_k(\lambda) = \mathcal{B}_k(v_0, w_t(\lambda), \lambda)$. $\mathcal{B}_k$ is a solver for the **linear system**

$$(I - \partial_1 \Phi(w_t(\lambda), \lambda)^\top) v = \nabla_1 E(w_t(\lambda), \lambda)$$

3. $\hat{\nabla} f(\lambda) = \nabla_2 E(w_t(\lambda), \lambda) + \partial_2 \Phi(w_t(\lambda), \lambda)^\top v_k(\lambda)$

- Only requires to save the last iterate $w_t(\lambda)$.
- $\mathcal{B}_k$ is efficient if uses **AD** for the Jacobian-vector products

$$\partial_1 \Phi(w_t(\lambda), \lambda)^\top v$$

$\implies$ Time$(\partial_1 \Phi(w_t(\lambda), \lambda)^\top v) = O($Time$(\Phi(w_t(\lambda), \lambda)))$.

```python
from torch.autograd import grad

wt = ll_alg(t, w0, lmbd, phi)  # solve the lower-level
wt = wt.detach().requires_grad_()  # avoid AD through wt

# gradient of E
grad_1_E, grad_2_E = grad(outputs=E(wt, lmbd), inputs=[wt, lmbd])

w_up = phi(wt, lmbd)
def jac_1_phi_v_f(v):  # reverse AD Jacobian-vector products
    return grad(w_up, grad_outputs=v, inputs=wt, retain_graph=True)[0]

vk = ls_alg(k, v0, jac_1_phi_v_f, grad_1_E)  # solve the linear-system

# compute the hypergradient approximation
jac_2_phi_vk = grad(w_up, grad_outputs=vk, inputs=lmbd)[0]
hypergrad = grad_2_E + jac_2_phi_vk
```

Comparison between ITD and AID

ITD

- Ignores the bilevel structure.

AID

- Can use any lower-level solver.

Comparison between ITD and AID

ITD

- Ignores the bilevel structure.
- Cost in time: $O(\text{Time}(f_t(\lambda)))$

AID

- Can use any lower-level solver.
- Cost in time $(k = t)$: $O(\text{Time}(f_t(\lambda)))$.

Comparison between ITD and AID

ITD

- Ignores the bilevel structure.
- Cost in time: $O(\text{Time}(f_t(\lambda)))$
- Cost in memory: $O(td)$.

AID

- Can use any lower-level solver.
- Cost in time $(k = t)$: $O(\text{Time}(f_t(\lambda)))$.
- Cost in memory: $O(d)$.

ITD and AID in Applications: Popular Works

		Work	UL variable λ	LL variable w
HPO	ITD	Maclaurin et al. (2015)	lr, mu	NN weights
		Franceschi et al. (2017)	lr, mu, others	NN weights
	AID	Pedregosa (2016)	reg params	linear params
MTL	ITD	MAML (Finn et al., 2017)	NN init.	NN weights
		Franceschi et al. (2018)	NN representation	last linear layer
	AID	iMAML (Rajeswaran et al., 2019)	biased reg	NN weights
		Lee et al. (2019); Bertinetto et al. (2019)	NN representation	last layer
DPA	ITD	Muñoz-González et al. (2017)	examples noise	NN weights

ITD and AID in Applications: Popular Works

		Work	UL variable λ	LL variable w
HPO	ITD	Maclaurin et al. (2015)	lr, mu	NN weights
		Franceschi et al. (2017)	lr, mu, others	NN weights
	AID	Pedregosa (2016)	reg params	linear params
MTL	ITD	MAML (Finn et al., 2017)	NN init.	NN weights
		Franceschi et al. (2018)	NN representation	last linear layer
	AID	iMAML (Rajeswaran et al., 2019)	biased reg	NN weights
		Lee et al. (2019); Bertinetto et al. (2019)	NN representation	last layer
DPA	ITD	Muñoz-González et al. (2017)	examples noise	NN weights

ITD/AID Code:

(Grazzi et al., 2020) *https://github.com/prolearner/hypertorch*

(Choe et al., 2023) *https://github.com/leopard-ai/betty*

(Blondel et al., 2021) *https://github.com/google/jaxopt*

Summary - Part 2

Automatic Differentiation

- Efficient program for $f(x) \implies$ efficient programs for $\partial f(x)\dot{x}$, $\partial f(x)^\top \bar{y}$
- Forward mode AD: Little overhead, fast $\partial f(x)$ if $x \in \mathbb{R}$.
- Reverse mode AD (backpropagation): fast $\nabla f(x)$ with high memory cost.

Hypergradient Approximation

- **ITD:** backpropagates over t steps of the LL solver.
- **AID:** solves the LS in the **implicit expression** of ∇f exploiting (reverse) AD for fast Jacobian-vector products.
- Both efficient but **ITD has a memory cost proportional to** t.

Questions on AID and ITD

- Can we control the approximation error of the hypergradient?
- Convergence guarantees on the bilevel problem?
- Stochastic approaches?

Error Rates for AID and ITD

Preliminary on Norms

For any $d, m \in \mathbb{N}$, $v \in \mathbb{R}^d$, $A \in \mathbb{R}^{m \times d}$

- $\|v\|$, $\|Av\|$ are two **vector norms**.
- $\|A\|$ denotes the corresponding **operator norm**:

$$\|A\| := \sup \left\{ \frac{\|Au\|}{\|u\|} \; : \; u \neq 0, u \in \mathbb{R}^d \right\}$$

$\implies$ sub-multiplicativity: $\|Av\| \leq \|A\| \|v\|$ for any $v \in \mathbb{R}^d$

The Contraction Assumption

Assumption

LL fixed point map $\Phi(\cdot, \lambda) : \mathbb{R}^d \to \mathbb{R}^d$ is a q_λ-**contraction**, i.e. for all $w_1, w_2 \in \mathbb{R}^d$

$$\|\Phi(w_1, \lambda) - \Phi(w_2, \lambda)\| \leq q_\lambda \|w_1 - w_2\|, \qquad q_\lambda < 1,$$

or equivalently, if Φ is differentiable $\|\partial_1 \Phi(w, \lambda)\| \leq q_\lambda < 1, \quad \forall w \in \mathbb{R}^d$.

The Contraction Assumption

Assumption

LL fixed point map $\Phi(\cdot, \lambda) : \mathbb{R}^d \to \mathbb{R}^d$ is a q_λ-**contraction**, i.e. for all $w_1, w_2 \in \mathbb{R}^d$

$$\|\Phi(w_1, \lambda) - \Phi(w_2, \lambda)\| \leq q_\lambda \|w_1 - w_2\|, \qquad q_\lambda < 1,$$

or equivalently, if Φ is differentiable $\|\partial_1 \Phi(w, \lambda)\| \leq q_\lambda < 1, \quad \forall w \in \mathbb{R}^d$.

$\implies$ $\Phi(\cdot, \lambda)$ has a **unique fixed point** $w(\lambda)$ (Banach, 1922).

$\implies$ **lower-level linear convergence** for $w_t(\lambda) = \Phi(w_{t-1}(\lambda), \lambda)$:

$$\|w(\lambda) - w_t(\lambda)\| \leq q_\lambda^t \|w(\lambda) - w_0(\lambda)\|$$

The Contraction Assumption

Assumption

LL fixed point map $\Phi(\cdot, \lambda) : \mathbb{R}^d \to \mathbb{R}^d$ is a q_λ-**contraction**, i.e. for all $w_1, w_2 \in \mathbb{R}^d$

$$\|\Phi(w_1, \lambda) - \Phi(w_2, \lambda)\| \leq q_\lambda \|w_1 - w_2\|, \qquad q_\lambda < 1,$$

or equivalently, if Φ is differentiable $\|\partial_1 \Phi(w, \lambda)\| \leq q_\lambda < 1, \quad \forall w \in \mathbb{R}^d$.

$\implies$ $\Phi(\cdot, \lambda)$ has a **unique fixed point** $w(\lambda)$ (Banach, 1922).

$\implies$ **lower-level linear convergence** for $w_t(\lambda) = \Phi(w_{t-1}(\lambda), \lambda)$:

$$\|w(\lambda) - w_t(\lambda)\| \leq q_\lambda^t \|w(\lambda) - w_0(\lambda)\|$$

Proof step: $\|w(\lambda) - w_t(\lambda)\| = \|\Phi(w(\lambda), \lambda) - \Phi(w_{t-1}(\lambda), \lambda)\| \leq q_\lambda \|w(\lambda) - w_{t-1}(\lambda)\|$

Examples of Contractions

Equilibrium models:

$$\Phi(w, \underbrace{(A, B, c)}_{\lambda}) = \tanh(Aw + Bx + c), \qquad \|A\| < 1$$

Bilevel Min-Min with $\underbrace{g(\cdot, \lambda)}_{\text{LL objective}}$ is μ-**strongly convex** and L-**Lipschitz smooth**:

- **Gradient Descent reformulation**:

$$\Phi(w, \lambda) = w - \eta \nabla_1 g(w, \lambda), \qquad \eta < 2/L.$$

- **Heavy-ball momentum reformulation when $g(\cdot, \lambda)$ is quadratic.**
  $\implies$ w contains also the momentum variable.

ITD Error

We want to control the hypergradient approximation error $\|\nabla f(\lambda) - \nabla f_t(\lambda)\|$.

$$\nabla f(\lambda) = \nabla_2 E(w(\lambda), \lambda) + w'(\lambda)^\top \nabla_1 E(w(\lambda), \lambda)$$
$$w'(\lambda) = \partial_1 \Phi(w(\lambda), \lambda) w'(\lambda) + \partial_2 \Phi(w(\lambda), \lambda).$$

$\nabla f_t(\lambda)$ is given by **ITD** with the LL solver

$$w_t(\lambda) = \Phi(w_{t-1}(\lambda), \lambda), \quad w_0(\lambda) = 0.$$

Using the **chain rule** on $f_t(\lambda) = E(w_t(\lambda), \lambda)$ we get

$$\nabla f_t(\lambda) = \nabla_2 E(w_t(\lambda), \lambda) + w_t'(\lambda)^\top \nabla_1 E(w_t(\lambda), \lambda)$$
$$w_t'(\lambda) = \partial_1 \Phi(w_{t-1}(\lambda), \lambda) w_{t-1}'(\lambda) + \partial_2 \Phi(w_{t-1}(\lambda), \lambda).$$

Useful Proof Step

Goal: Upper bound $\|Q - Q'\|$

with $Q = A + BC, Q' = A' + B'C'$ with all quantities being vectors or matrices.

$$
\begin{aligned}
& \|Q - Q'\| = \|A - A' + BC \mp BC' - B'C'\| \\
\text{Triangle inequality} \implies \quad & \leq \|A - A'\| + \|BC - BC'\| + \|BC' - B'C'\| \\
\text{Sub-multiplicativity} \implies \quad & \leq \|A - A'\| + \|B\|\|C - C'\| + \|C'\|\|B - B'\|.
\end{aligned}
$$

Then it only remains to bound $\|X - X'\|$ with $X \in \{A, B, C\}$, $\|B\|$ and $\|C'\|$.

ITD Error Rate - Bounding $\|\nabla f(\lambda) - \nabla f_t(\lambda)\|$

$$\underbrace{\nabla f(\lambda)}_{Q} = \underbrace{\nabla_2 E(w(\lambda), \lambda)}_{A} + \underbrace{w'(\lambda)^\top}_{B} \underbrace{\nabla_1 E(w(\lambda), \lambda)}_{C},$$

$$\underbrace{\nabla f_t(\lambda)}_{Q'} = \underbrace{\nabla_2 E(w_t(\lambda), \lambda)}_{A'} + \underbrace{w_t'(\lambda)^\top}_{B'} \underbrace{\nabla_1 E(w_t(\lambda), \lambda)}_{C'}$$

We need to bound $\|B\|$, $\|C'\|$ and $\|X - X'\|$ with $X \in \{A, B, C\}$:

- $\|w'(\lambda)\|$ is constant since λ is fixed.
- $w_0(\lambda) = 0$ + contraction $\implies \|w_t(\lambda)\| \leq 2\|w(\lambda)\| \implies \|\nabla_1 E(w_t(\lambda), \lambda)\|$ bounded.
- $\nabla_i E(\cdot, \lambda)$ L_E-Lipschitz + contraction $\implies$

$$\|\nabla_i E(w(\lambda), \lambda) - \nabla_i E(w_t(\lambda), \lambda)\| \leq L_E \|w(\lambda) - w_t(\lambda)\| \leq L_E q_\lambda^t \|w(\lambda)\|$$

- Next we bound $\|w_t'(\lambda) - w'(\lambda)\|$?

ITD Error Rate - Bounding $\|w_t'(\lambda) - w(\lambda)\|$

$$\underbrace{w_t'(\lambda)}_{Q} = \underbrace{\partial_2\Phi(w_{t-1}(\lambda), \lambda)}_{A} + \underbrace{\partial_1\Phi(w_{t-1}(\lambda), \lambda)}_{B}\,\underbrace{w_{t-1}'(\lambda)}_{C}$$

$$\underbrace{w'(\lambda)}_{Q'} = \underbrace{\partial_2\Phi(w(\lambda), \lambda)}_{A'} + \underbrace{\partial_1\Phi(w(\lambda), \lambda)}_{B'}\,\underbrace{w'(\lambda)}_{C'}$$

We need to bound $\|B\|$, $\|C'\|$ and $\|X - X'\|$ with $X \in \{A, B, C\}$:

- $\|w'(\lambda)\|$ is constant since λ is fixed.
- $\Phi(\cdot, \lambda)$ is a differentiable q_λ-contraction $\implies \|\partial_1\Phi(w_{t-1}(\lambda), \lambda)\| \leq q_\lambda$
- $\partial_i\Phi(\cdot, \lambda)$ Lipschitz $\implies \|\partial_i\Phi(w_t(\lambda), \lambda) - \partial_i\Phi(w(\lambda), \lambda)\| \leq L_\Phi\|w_t(\lambda) - w(\lambda)\|$.
- we **unroll the recursion** to control $\|w_{t-1}'(\lambda) - w'(\lambda)\|$

ITD Error Rate - Unrolling $\|w'_t(\lambda) - w'(\lambda)\|$

Denote $\Delta_t = \|w_t(\lambda) - w(\lambda)\|$, $\Delta'_t = \|w'_t(\lambda) - w'(\lambda)\|$, $c_\lambda = L_\Phi(\|w'(\lambda)\| + 1)$

$$
\begin{aligned}
\Delta'_t &\leq c_\lambda \Delta_{t-1} + q_\lambda \Delta'_{t-1} \\
&\leq c_\lambda \Delta_{t-1} + q_\lambda c_\lambda \Delta_{t-2} + q_\lambda^2 \Delta'_{t-2} \\
&\leq c_\lambda \sum_{i=1}^{t} q_\lambda^{i-1} \Delta_{t-i} + q_\lambda^t \|w'(\lambda)\| \\
&\leq c_\lambda \Delta_0 \sum_{i=1}^{t} q_\lambda^{i-1+t-i} + q_\lambda^t \|w'(\lambda)\| \qquad (\Delta_{t-i} \leq q_\lambda^{t-i} \Delta_0) \\
&\leq t q_\lambda^{t-1} c_\lambda \|w(\lambda)\| + q_\lambda^t \|w'(\lambda)\|
\end{aligned}
$$

ITD Error Rate

Theorem *(ITD error upper bound)* (Grazzi et al., 2020)

If $\Phi(\cdot, \lambda)$ is a q_λ-contraction and $\nabla E(\cdot, \lambda)$, $\partial_i \Phi(\cdot, \lambda)$ are Lipschitz, then ITD with $w_t(\lambda) = \Phi(w_{t-1}, \lambda)$, $w_0(\lambda) = 0$ satisfies

$$\|\nabla f_t(\lambda) - \nabla f(\lambda)\| \leq \left(c_1(\lambda) + \frac{c_2(\lambda)}{q_\lambda} t + c_3(\lambda)\right) q_\lambda^t,$$

ITD **converges linearly** to $\nabla f(\lambda)$ with rate smaller than q_λ^t.

AID Error Rate

We want to control the hypergradient approximation error $\|\nabla f(\lambda) - \hat{\nabla} f(\lambda)\|$

$$\nabla f(\lambda) = \nabla_2 E(w(\lambda), \lambda) + \partial_2 \Phi(w(\lambda), \lambda)^\top v(\lambda),$$
$$v(\lambda) = (I - \partial_1 \Phi(w(\lambda), \lambda)^\top)^{-1} \nabla_1 E(w(\lambda), \lambda)$$

Recall that for **AID**:

$$\hat{\nabla} f(\lambda) = \nabla_2 E(w_t(\lambda), \lambda) + \partial_2 \Phi(w_t(\lambda), \lambda)^\top v_k(\lambda),$$

$w_t(\lambda)$ is is the output after t steps of the LL solver.

$v_k(\lambda)$ is the output after k steps of the LS solver for the LS with solution

$$\hat{v}(\lambda) := (I - \partial_1 \Phi(w_t(\lambda), \lambda)^\top)^{-1} \nabla_1 E(w_t(\lambda), \lambda)$$

AID Error Rate - Part 2

Using similar techniques as for ITD we get

Theorem *(AID error upper bound)* (Grazzi et al., 2020)

Assume that $\nabla_i E$ and $\partial_i \Phi$ are Lipschitz, $\Phi(\cdot, \lambda)$ is a q_λ-contraction and

- $\|w_t(\lambda) - w(\lambda)\| \leq \rho_\lambda(t) \|w(\lambda)\|$ (LL rate)
- $\|v_k(\lambda) - \hat{v}(\lambda)\| \leq \sigma_\lambda(k) \|\hat{v}(\lambda)\|$ (LS rate)

Then,

$$\|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\| \leq \left(c_1(\lambda) + \frac{c_2(\lambda)}{1 - q_\lambda}\right) \rho_\lambda(t) + c_3(\lambda) \sigma_\lambda(k).$$

Solving the Linear System

Fixed point method (FP)

Let $v_0(\lambda) = 0$, assume $w_t(\lambda)$, $\nabla_1 E(w_t(\lambda), \lambda)$ given and compute

$$v_k(\lambda) = \Psi(v_{k-1}(\lambda), \lambda) := \partial_1 \Phi(w_t(\lambda), \lambda)^\top v_{k-1}(\lambda) + \nabla_1 E(w_t(\lambda), \lambda)$$

Efficient evaluation with AD: only one jacobian-vector product per step.

Solving the Linear System

Fixed point method (FP)

Let $v_0(\lambda) = 0$, assume $w_t(\lambda)$, $\nabla_1 E(w_t(\lambda), \lambda)$ given and compute

$$v_k(\lambda) = \Psi(v_{k-1}(\lambda), \lambda) := \partial_1 \Phi(w_t(\lambda), \lambda)^\top v_{k-1}(\lambda) + \nabla_1 E(w_t(\lambda), \lambda)$$

Efficient evaluation with AD: only one jacobian-vector product per step.

$\Psi(\cdot, \lambda)$ **is an affine map and when** $\Phi(\cdot, \lambda)$ **is a** q_λ**-contraction:**

- Is a q_λ-**contraction** since $\|\partial_1 \Psi(v, \lambda)\| = \|\partial_1 \Phi(w_t(\lambda), \lambda)\| \leq q_\lambda$.
- Its **fixed point** is $\hat{v}(\lambda)$
- $\implies$ **linear convergence**: $\|\hat{v}(\lambda) - v_k(\lambda)\| \leq q_\lambda^k \|\hat{v}(\lambda) - v_0(\lambda)\|$

Solving the Linear System - Part 2

$$\underbrace{I - \partial_1 \Phi(w_t(\lambda), \lambda)}_{A \in \mathbb{R}^{d \times d}} x = \underbrace{\nabla_1 E(w_t(\lambda), \lambda)}_{b \in \mathbb{R}^d}$$

Conjugate Gradient (CG)

- A needs to be **symmetric positive-definite** (Bilevel Min-Min).
- Only one matrix-vector (in our case Jacobian-vector) product per step.
- Converges **exactly in d iterations**.
- Converges **linearly** with rate $p_\lambda^k < q_\lambda^k$

Refined Analysis for the Fixed Point Method

Let $u_0(\lambda) := 0$ $u_k(\lambda) := \partial_1 \Phi(w_t(\lambda), \lambda) u_{k-1}(\lambda) + \partial_2 \Phi(w_t(\lambda), \lambda)$ and note that

$$\partial_2 \Phi(w_t(\lambda), \lambda)^\top v_k(\lambda) = u_k(\lambda)^\top \nabla_1 E(w_t(\lambda), \lambda)$$

Proof Let $A = \partial_2 \Phi(w_t(\lambda), \lambda)^\top$, $B = \partial_1 \Phi(w_t(\lambda), \lambda)^\top$, $c = \nabla_1 E(w_t(\lambda), \lambda)$

$$v_k(\lambda) = B v_{k-1}(\lambda) + c = B^k v_0(\lambda) + \sum_{i=1}^{k} B^{i-1} c = \sum_{i=1}^{k} B^{i-1} c$$

$$u_k^\top(\lambda) = u_{k-1}^\top(\lambda) B + A = B^k u_0^\top(\lambda) + A \sum_{i=1}^{k} B^{i-1} = A \sum_{i=1}^{k} B^{i-1}.$$

Therefore $\quad A v_k(\lambda) = A \sum_{i=1}^{k} B^{i-1} c = u_k^\top(\lambda) c$

Refined Analysis for the Fixed Point Method - Part 2

Denote $\Delta'_k = \|w'(\lambda) - u_k(\lambda)\|$, $\Delta_t = \|w(\lambda) - w_t(\lambda)\|$, $c_\lambda = L_\Phi(\|w'(\lambda)\| + 1)$

$$
\begin{aligned}
\Delta'_k &\leq c_\lambda \Delta_t + q_\lambda \Delta'_{k-1} \\
&\leq \sum_{i=1}^{k} q_\lambda^{i-1} c_\lambda \Delta_t + q_\lambda^k \|w'(\lambda)\| \\
&\leq \sum_{i=1}^{k} q_\lambda^{t+i-1} c_\lambda \Delta_0 + q_\lambda^k \|w'(\lambda)\| \\
&\leq q_\lambda^t \sum_{i=0}^{k-1} q_\lambda^{i} c_\lambda \Delta_0 + q_\lambda^k \|w'(\lambda)\| \\
&\leq q_\lambda^t \frac{1 - q_\lambda^k}{1 - q_\lambda} c_\lambda \|w(\lambda)\| + q_\lambda^k \|w'(\lambda)\|
\end{aligned}
$$

AID Fixed Point and Conjugate gradient Rates

Theorem *(CG and FP error upper bounds)* (Grazzi et al., 2020)

If $\Phi(\cdot, \lambda)$ is a q_λ-contraction and $\nabla E(\cdot, \lambda)$, $\partial_i \Phi(\cdot, \lambda)$ are Lipschitz, and set the LL solver to $w_t(\lambda) = \Phi(w_{t-1}, \lambda)$, $w_0(\lambda) = 0$, then

$$\text{(AID-FP)} \qquad \|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\| \leq \left(c_1(\lambda) + c_2(\lambda)\frac{1 - q_\lambda^k}{1 - q_\lambda}\right) q_\lambda^t + c_3(\lambda) q_\lambda^k.$$

Moreover, when $\partial_1 \Phi(w_t(\lambda), \lambda)$ is symmetric,

$$\text{(AID-CG)} \qquad \|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\| \leq \left(c_1(\lambda) + \frac{c_2(\lambda)}{1 - q_\lambda}\right) q_\lambda^t + c_3(\lambda) \hat{c}(\lambda) p_\lambda^k,$$

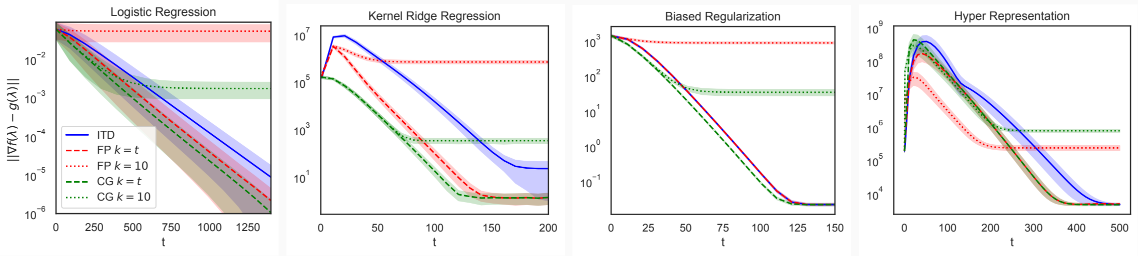
where $p_\lambda < q_\lambda$.

Which method has the best approximation error?

$$\text{ITD} = O(tq_\lambda^t), \quad \text{AID-FP} = O(q_\lambda^t + q_\lambda^k), \quad \text{AID-CG} = O(q_\lambda^t + p_\lambda^k).$$

- ITD, AID-CG and AID-FP **converge linearly** (in t and k) to $\nabla f(\lambda)$.
- AID-FP bound $<$ ITD bound for every t, when $k = t$.
- AID-CG bound $<$ AID-FP bound for large k and $\partial_1 \Phi(w_t(\lambda), \lambda)$ symmetric.

Hypergradient Approximation on Synthetic Data



Hypergradient approximation errors (mean/std on randomly drawn values of λ). $g(\lambda)$ is equal to $\nabla f_t(\lambda)$ for ITD and to $\hat{\nabla} f(\lambda)$ for CG and FP. In all settings $\Phi(\cdot, \lambda)$ is a contraction and $\partial_1 \Phi(w, \lambda)$ is symmetric.

- After a while the error decreases linearly for all methods.
- Methods with lower error bounds have lower error on average.

Inexact Hypegradient Descent

Inexact Hypegradient Descent (IHD)

Inexact Hypergradient Descent

For $s = 0, 1, 2, \dots S$

$$\lambda_{s+1} = \lambda_s - \alpha_s \hat{\nabla} f(\lambda_s)$$

where **AID/ITD** gives $\hat{\nabla} f(\lambda_s)$

- **Unconstrained** ($\Lambda = \mathbb{R}^m$), similar results also for **Projected** IHD ($\Lambda \subset \mathbb{R}^m$).
- The bilevel objective f is usually **non-convex**.
- The errors $\|\hat{\nabla} f(\lambda_s) - \nabla f(\lambda_s)\|$ can be controlled by setting t and k.

Convergence to stationary points

Assume f is L_f-smooth and $\|\cdot\|$ be the euclidean norm.

Non-convex optimality measure:

$$\min_{s \leq S} \|\nabla f(\lambda_s)\|^2 \leq \frac{1}{S} \sum_{s=1}^{S} \|\nabla f(\lambda_s)\|^2$$

Theorem (IHD Bound with Errors)

IHD with learning rate $\alpha_s = \alpha$, $0 < \alpha \leq 1/L_f$ yields

$$\frac{1}{S} \sum_{s=0}^{S-1} \|\nabla f(\lambda_s)\|^2 \leq \frac{1}{S} \left[\frac{2\Delta_f}{\alpha} + \sum_{s=0}^{S-1} \delta_s^2 \right].$$

$$\Delta_f := f(\lambda_0) - \min_{\lambda} f(\lambda), \qquad \delta_s = \|\hat{\nabla} f(\lambda_s) - \nabla f(\lambda_s)\|.$$

Proof

$$
\begin{aligned}
f\ L\text{-smooth} \implies f(\lambda_{s+1}) &\leq f(\lambda_s) + \nabla f(\lambda_s)^\top(\lambda_{s+1} - \lambda_s) + \frac{L_f}{2}\|\lambda_{s+1} - \lambda_s\|^2 \\
\lambda_{s+1} - \lambda_s = -\alpha\hat{\nabla} f(\lambda_s) \implies &= f(\lambda_s) + \alpha\big(-\nabla f(\lambda_s)^\top \hat{\nabla} f(\lambda_s) + \frac{L_f\alpha}{2}\|\hat{\nabla} f(\lambda_s)\|^2\big) \\
0 \leq \alpha \leq 1/L_f \implies &\leq f(\lambda_s) + \alpha\big(-\nabla f(\lambda_s)^\top \hat{\nabla} f(\lambda_s) + \frac{1}{2}\|\hat{\nabla} f(\lambda_s)\|^2\big) \\
\|b\|^2 - 2a^\top b = \|a-b\|^2 - \|a\|^2 \implies &= f(\lambda_s) + \alpha\left(\frac{1}{2}\underbrace{\|\nabla f(\lambda_s) - \hat{\nabla} f(\lambda_s)\|^2}_{\delta_s^2} - \frac{1}{2}\|\nabla f(\lambda_s)\|^2\right)
\end{aligned}
$$

Proof

$$f\ L\text{-smooth} \implies f(\lambda_{s+1}) \leq f(\lambda_s) + \nabla f(\lambda_s)^\top(\lambda_{s+1} - \lambda_s) + \frac{L_f}{2}\|\lambda_{s+1} - \lambda_s\|^2$$

$$\lambda_{s+1} - \lambda_s = -\alpha\hat{\nabla} f(\lambda_s) \implies = f(\lambda_s) + \alpha\Big(-\nabla f(\lambda_s)^\top \hat{\nabla} f(\lambda_s) + \frac{L_f\alpha}{2}\|\hat{\nabla} f(\lambda_s)\|^2\Big)$$

$$0 \leq \alpha \leq 1/L_f \implies \leq f(\lambda_s) + \alpha\Big(-\nabla f(\lambda_s)^\top \hat{\nabla} f(\lambda_s) + \frac{1}{2}\|\hat{\nabla} f(\lambda_s)\|^2\Big)$$

$$\|b\|^2 - 2a^\top b = \|a-b\|^2 - \|a\|^2 \implies = f(\lambda_s) + \alpha\left(\frac{1}{2}\underbrace{\|\nabla f(\lambda_s) - \hat{\nabla} f(\lambda_s)\|^2}_{\delta_s^2} - \frac{1}{2}\|\nabla f(\lambda_s)\|^2\right)$$

$$\text{Rearranging} \implies \|\nabla f(\lambda_s)\|^2 \leq \frac{2}{\alpha}(f(\lambda_s) - f(\lambda_{s-1})) + \delta_s^2$$

$$\text{Telescoping} \implies \frac{1}{S}\sum_{s=0}^{S-1}\|\nabla f(\lambda_s)\|^2 \leq \frac{2}{\alpha S}\underbrace{(f(\lambda_0) - f(\lambda_{s+1}))}_{\Delta_f} + \frac{1}{S}\sum_{s=0}^{S-1}\delta_s^2$$

Additional Assumptions

Assumption C

- Previous constants depending on λ need to be uniformly bounded over Λ.
- In particular $\Phi(\cdot, \lambda)$ is a q-contraction for all $\lambda \in \Lambda$.
- $\nabla_1 E(w^*, \cdot), \nabla_2 E(w^*, \cdot)$, $\partial_1 \Phi(w^*, \cdot)$, $\partial_2 \Phi(w^*, \cdot)$ are Lipschitz continuous $\forall w^* \in \{w(\lambda) : \lambda \in \Lambda\}$.

$\implies$ f is L_f-smooth.

IHD iteration complexity

Theorem (deterministic iteration complexity) (Grazzi et al., 2022)

Set the learning rate $\alpha_s = 1/L_f$ and use **AID-FP** with

$$t_s = k_s = \lceil \kappa \log(s+1) \rceil, \qquad \kappa := \frac{1}{1-q},$$

we have

$$\frac{1}{S} \sum_{s=0}^{S-1} \|\nabla f(\lambda_s)\|^2 \leq \frac{2 L_f \Delta_f + C}{S},$$

ϵ-accuracy in $O(\epsilon^{-1} \log(\epsilon^{-1}))$ total LL and LS iterations, optimal up to log factors.

IHD iteration complexity

Theorem (deterministic iteration complexity) (Grazzi et al., 2022)

Set the learning rate $\alpha_s = 1/L_f$ and use **AID-FP** with

$$t_s = k_s = \lceil \kappa \log(s+1) \rceil, \qquad \kappa := \frac{1}{1-q},$$

we have

$$\frac{1}{S} \sum_{s=0}^{S-1} \|\nabla f(\lambda_s)\|^2 \leq \frac{2L_f \Delta_f + C}{S},$$

ϵ-accuracy in $O(\epsilon^{-1} \log(\epsilon^{-1}))$ total LL and LS iterations, optimal up to log factors.

- Bilevel Min-Min: κ is the LL **condition number**.
- $t_s = \lceil (s+1)^{1/4}/2 \rceil \implies O\left(\epsilon^{-5/4}\right)$ (Ghadimi and Wang, 2018)

Stochastic Bilevel Optimization

Stochastic Bilevel Optimization Problem

$$\min_{\lambda \in \Lambda \subseteq \mathbb{R}^m} f(\lambda) := E(w(\lambda), \lambda) = \mathbb{E}_\xi[\hat{E}(w(\lambda), \lambda, \xi)] \quad \text{upper-level (UL)}$$

$$w(\lambda) := \Phi(w(\lambda), \lambda) = \mathbb{E}_\zeta[\hat{\Phi}(w(\lambda), \lambda, \zeta)] \quad \text{lower-level (LL)}$$

- E, Φ are too expensive to evaluate, we use $\hat{\Phi}, \hat{E}$ instead.

Stochastic Bilevel Optimization Problem

$$\min_{\lambda \in \Lambda \subseteq \mathbb{R}^m} f(\lambda) := E(w(\lambda), \lambda) = \mathbb{E}_{\xi}[\hat{E}(w(\lambda), \lambda, \xi)] \quad \text{upper-level (UL)}$$

$$w(\lambda) := \Phi(w(\lambda), \lambda) = \mathbb{E}_{\zeta}[\hat{\Phi}(w(\lambda), \lambda, \zeta)] \quad \text{lower-level (LL)}$$

- E, Φ are too expensive to evaluate, we use $\hat{\Phi}, \hat{E}$ instead.
- E.g. $\Phi(w, \lambda) = \frac{1}{n} \sum_{i=1}^{n} \hat{\Phi}(w, \lambda, i)$, $E(w, \lambda) = \frac{1}{n} \sum_{i=1}^{n} \hat{E}(w, \lambda, i)$ (large n).
- E.g. $\hat{\Phi}$ is the SGD map and $\hat{E}$ is the loss on a few examples.
- **Large scale** hyperparameter optimization and Meta-learning.

Stochastic Implicit Differentiation (SID)

1. $w_t(\lambda)$ by t steps of a **stochastic solver** approximating $w(\lambda)$.
2. $$\nabla \bar{E}(w_t(\lambda), \lambda) = \tfrac{1}{J} \textstyle\sum_{j=1}^{J} \nabla \hat{E}(w_t(\lambda), \lambda, \xi_j)$$
3. $v_k(\lambda)$ by k steps of a **stochastic solver** for the linear system
   $$(I - \partial_1 \Phi(w_t(\lambda), \lambda)^\top) v = \nabla_1 \bar{E}(w_t(\lambda), \lambda).$$
   with solution $\bar{v}(\lambda)$.
4. $$\hat{\nabla} f(\lambda) := \nabla_2 \bar{E}(w_t(\lambda), \lambda) + \partial_2 \bar{\Phi}(w_t(\lambda), \lambda)^\top v_k(\lambda).$$
   where $\partial_2 \bar{\Phi}(w_t(\lambda), \lambda) = \frac{1}{J} \sum_{j=1}^{J} \partial_2 \hat{\Phi}(w_t(\lambda), \lambda, \zeta'_j)$

Stochastic Implicit Differentiation (SID)

1. $w_t(\lambda)$ by t steps of a **stochastic solver** approximating $w(\lambda)$.
2. $$\nabla \bar{E}(w_t(\lambda), \lambda) = \tfrac{1}{J} \textstyle\sum_{j=1}^{J} \nabla \hat{E}(w_t(\lambda), \lambda, \xi_j)$$
3. $v_k(\lambda)$ by k steps of a **stochastic solver** for the linear system
$$(I - \partial_1 \Phi(w_t(\lambda), \lambda)^\top) v = \nabla_1 \bar{E}(w_t(\lambda), \lambda).$$
with solution $\bar{v}(\lambda)$.
4. $$\hat{\nabla} f(\lambda) := \nabla_2 \bar{E}(w_t(\lambda), \lambda) + \partial_2 \bar{\Phi}(w_t(\lambda), \lambda)^\top v_k(\lambda).$$
where $\partial_2 \bar{\Phi}(w_t(\lambda), \lambda) = \tfrac{1}{J} \sum_{j=1}^{J} \partial_2 \hat{\Phi}(w_t(\lambda), \lambda, \zeta'_j)$

- ∇E, $\partial_2 \Phi$ are estimated with **mini-batches** of size J.
- LL solver will use $\hat{\Phi}(w, \lambda, \xi_t)$, LS solver will use $\partial_1 \hat{\Phi}(w_t(\lambda), \lambda, \hat{\xi}_k)^\top v_k$.

SID Mean Squared Error (MSE): Preliminaries

‘

Assumption A

$\forall \lambda \in \Lambda$:

- $\Phi(\cdot, \lambda)$ is a q_λ-**contraction** with $q_\lambda < 1$.
- $E(\cdot, \lambda)$, $\partial_1 \Phi(\cdot, \lambda)$, $\partial_2 \Phi(\cdot, \lambda)$, $\nabla_1 E(\cdot, \lambda)$ and $\nabla_2 E(\cdot, \lambda)$ are Lipschitz continuous

Assumption B

$\forall \lambda \in \Lambda, w \in \mathbb{R}^d$

- $\mathbb{V}[\hat{\Phi}(w, \lambda, \zeta)] \leq \sigma_{\lambda,1} + \sigma_{\lambda,2} \|\Phi(w, \lambda) - w\|^2$ for $\sigma_{\lambda,1}, \sigma_{\lambda,2} \geq 0$
- $\mathbb{V}[\partial_1 \hat{\Phi}(w, \lambda, \zeta)]$, $\mathbb{V}[\partial_2 \hat{\Phi}(w, \lambda, \zeta)]$, $\mathbb{V}[\nabla_1 \hat{E}(w, \lambda, \xi)]$, $\mathbb{V}[\nabla_2 \hat{E}(w, \lambda, \xi)]$ are bounded.

Theorem *(SID error upper bound)* (Grazzi et al., 2021, 2022)

Assume that $\forall \lambda \in \Lambda, w \in \mathbb{R}^d$

$$\mathbb{E}[\|w_t(\lambda) - w(\lambda)\|^2] \leq \rho_\lambda(t) \qquad \text{(LL rate)}$$

$$\mathbb{E}[\|v_k(\lambda) - \bar{v}(\lambda)\|^2] \leq \sigma_\lambda(k) \qquad \text{(LS rate)}$$

Then,

$$\mathbb{E}[\|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\|^2] \leq \frac{c_{0,\lambda}}{J} + c_{1,\lambda}\rho_\lambda(t) + c_{2,\lambda}\sigma_\lambda(k) + c_{3,\lambda}\rho_\lambda(t)\sigma_\lambda(k).$$

Theorem *(SID error upper bound)* (Grazzi et al., 2021, 2022)

Assume that $\forall \lambda \in \Lambda, w \in \mathbb{R}^d$

$$\mathbb{E}[\|w_t(\lambda) - w(\lambda)\|^2] \leq \rho_\lambda(t) \qquad \text{(LL rate)}$$

$$\mathbb{E}[\|v_k(\lambda) - \bar{v}(\lambda)\|^2] \leq \sigma_\lambda(k) \qquad \text{(LS rate)}$$

Then,

$$\mathbb{E}[\|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\|^2] \leq \frac{c_{0,\lambda}}{J} + c_{1,\lambda}\rho_\lambda(t) + c_{2,\lambda}\sigma_\lambda(k) + c_{3,\lambda}\rho_\lambda(t)\sigma_\lambda(k).$$

How to solve the LL and LS?

- Stochastic Fixed-point (SGD rates):

$$\rho_\lambda(t) = O(1/t), \sigma_\lambda(k) = O(1/k) \qquad \text{(decreasing step sizes).}$$

- Conjugate Gradient does not have a stochastic variant.

Stochastic Fixed-point Iterations

$$w_{t+1} = w_t + \eta_t(\hat{T}(w_t, \zeta_t) - w_t).$$

- **LL map**: $\hat{T}(w, \zeta) = \hat{\Phi}(w, \lambda, \zeta)$.
- **LS map**: $\hat{T}(v, \zeta) = \partial_1 \hat{\Phi}(w_t(\lambda), \lambda, \zeta)^\top v + \nabla_1 \bar{E}(w_t(\lambda), \lambda)$.
- $\mathbb{E}[\hat{T}(\cdot, \zeta)]$ is a q_λ-contraction for both: we could use the same η_t.

- If $\hat{T}(w, \zeta) = w - \gamma \nabla_1 \hat{g}(w, \zeta)$ we recover SGD with learning rate $\gamma \eta_t$.
- $\mathbb{E}[\hat{T}(\cdot, \zeta)]$ is a contraction $\implies$ SGD rates on strongly convex and Lip. smooth.

Stochastic Fixed-point Rates (Inspired by Bottou et al. (2018))

Assumptions

- $w \to \mathbb{E}[\hat{T}(w, \zeta)]$ is a q-**contraction** with $q < 1$.
- $\forall w \in \mathbb{R}^d$, $\mathbb{V}[\hat{T}(w, \zeta)] \leq \sigma_1 + \sigma_2 \|T(w) - w\|^2$.

Theorem (Stochstic Fixed Point Rates) (Grazzi et al., 2021)

If $\eta_t = \eta \leq \frac{1}{1+\sigma_2}$. then

$$\mathbb{E}[\|w_t - w^*\|^2] \leq (1 - \eta(1 - q^2))^t \left(\mathbb{E}[\|w_0 - w^*\|^2] - \frac{\eta \sigma_1}{1 - q^2} \right) + \frac{\eta \sigma_1}{1 - q^2}.$$

If $\eta_t = \beta/(\gamma + t)$ with $\beta > 1/(1 - q^2)$ and $\gamma \geq \beta(1 + \sigma_2)$, then

$$\mathbb{E}[\|w_t - w^*\|^2] \leq \frac{c}{\gamma + t}.$$

Final Combined Rate

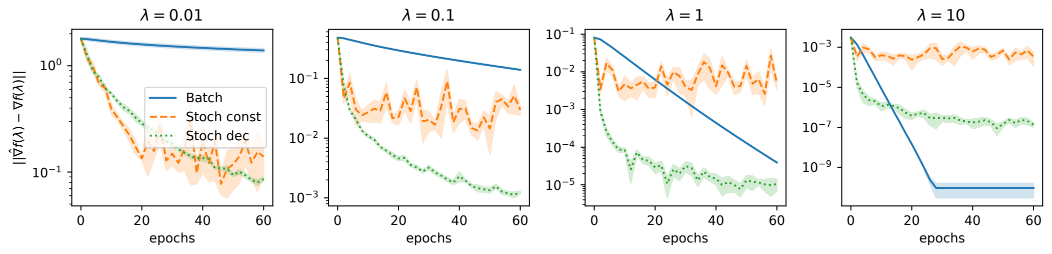
SID with decreasing step sizes for both the LL and LS achieves

$$\underbrace{\mathbb{E}[\|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\|^2]}_{\text{MSE}} = O\left(\frac{1}{J} + \frac{1}{t} + \frac{1}{k}\right)$$

$$\text{MSE} = \underbrace{\|\mathbb{E}[\hat{\nabla} f(\lambda)] - \nabla f(\lambda)\|^2}_{\text{Bias}} + \underbrace{\mathbb{E}[\|\hat{\nabla} f(\lambda) - \mathbb{E}[\hat{\nabla} f(\lambda)]\|^2]}_{\text{Variance}}$$

- With constant step sizes for the LS and decreasing for the LL we can control the **bias** but not the **variance** (Ghadimi and Wang, 2018).
- No results for **stochastic ITD**: same samples for $w_t(\lambda)$ and $w'_t(\lambda)$.

Regularized Logistic Regression on MNIST Odd vs Even $\lambda \in \mathbb{R}_{++}$



one regularization parameter: $\mathcal{R}_\lambda(w) = \frac{\lambda}{2}\|w\|^2$, $\lambda \in \mathbb{R}_{++}$

Stochastic Inexact Hypergradient Descent

Stochastic Inexact Hypergradient Descent (SIHD)

For $s = 0, 1, 2, \ldots S$

$$\lambda_{s+1} = \lambda_s - \alpha_s \hat{\nabla} f(\lambda_s), \qquad \text{SID gives } \hat{\nabla} f(\lambda_s)$$

- **Biased gradients:** $\mathbb{E}[\hat{\nabla} f(\lambda)] \neq \nabla f(\lambda) \implies$ can't apply SGD/GD analysis.
- We can control the **MSE** $\mathbb{E}\|\hat{\nabla} f(\lambda) - \nabla f(\lambda)\|^2$ or the **bias** $\|\mathbb{E}[\hat{\nabla} f(\lambda)] - \nabla f(\lambda)\|^2$.

IHD analysis + Decreasing MSE $\implies$

Theorem (Stochastic sample complexity) (Grazzi et al., 2022)

SIGD with UL learning rate $\alpha_s = 1/L_f$ and

$$\text{decreasing } (\eta_{s,j})_{j\in\mathbb{N}}, \quad t_s = k_s = J_s = \lceil c_3 S \rceil, \quad c_3 > 0$$

yileds

$$\frac{1}{S}\sum_{s=0}^{S-1} \mathbb{E}\|\nabla f(\lambda_s)\|^2 \leq \frac{1}{S}\left[2L_f\Delta_f + \frac{C}{c_3}\right],$$

ϵ-accuracy after $O(\epsilon^{-2})$ samples, which is optimal (Arjevani et al., 2023).

SGD analysis + Decreasing Bias + $J_s = 1 \implies O(\epsilon^{-3})$ (Ghadimi and Wang, 2018).

Summary Part 3

Deterministic Setting

- ITD and AID converge linearly (in t and k) to $\nabla f(\lambda)$.
- AID is generally faster than ITD and requires less memory.
- IHD using AID has an almost optimal complexity of $\tilde{O}(\epsilon^{-1})$.

Stochastic Setting

- SID $=$ AID with stochastic LL/LS solvers and minibatches of size J for ∇E, $\partial_2 \Phi$.
- MSE of SID converges as $O(1/t + 1/k + 1/J)$.
- Stochastic IHD has an optimal complexity of $O(\epsilon^{-2})$

Recent Advances

Non-smooth Bilevel (Smooth Almost Everywhere)

Examples:

- LASSO / Elastic Net

$$\min_{\lambda \in \Lambda} \frac{1}{n'} \sum_{i=1}^{n'} \mathcal{L}(w^\top x_{n+i}, y_{n+i})$$

$$w(\lambda) \in \arg\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(w^\top x_i, y_i) + \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2$$

- **Non-smooth Loss function:** e.g. hinge-loss in SVMs.
- **Neural Networks with non-smooth activations.**

Non-smooth Bilevel - Part 2

Issues:

- Might have more than one solution.
- Lower-level **rates are more difficult to obtain**.
- Requires **generalized derivatives** (e.g. Clarke): **no chain rule**.

Approaches:

- Smooth Algorithm + ITD (Ochs et al., 2015, 2016; Frecon et al., 2018)
- AID/ITD extension to weak derivatives (Bertrand et al., 2020, 2022)
- Make the **chain rule** work (Bolte et al., 2021)

Non-smooth Guarantees

Deterministic Hypergradient Approximation

- In certain situations ITD/AID converges linearly after **support identification** (Bertrand et al., 2020, 2022).
- ITD "converges" linearly for a broad class of functions (Bolte et al., 2022)

Non-smooth Guarantees

Deterministic Hypergradient Approximation

- In certain situations ITD/AID converges linearly after **support identification** (Bertrand et al., 2020, 2022).
- ITD "converges" linearly for a broad class of functions (Bolte et al., 2022)

No guarantees for bilevel optimization or stochastic approaches.

Warm-start

Inexact Gradient Descent

For $s = 0, 1, 2, \dots S$

$$\lambda_{s+1} = \lambda_s - \alpha_s \hat{\nabla} f(\lambda_s))$$

ITD/AID/SID yields $\hat{\nabla} f(\lambda_s)$ using solvers for the LL and LS.

Idea: start solvers with $w_t(\lambda_{s-1})$ and/or $v_k(\lambda_{s-1})$.

Warm-start

Inexact Gradient Descent

For $s = 0, 1, 2, \dots S$

$$\lambda_{s+1} = \lambda_s - \alpha_s \hat{\nabla} f(\lambda_s))$$

ITD/AID/SID yields $\hat{\nabla} f(\lambda_s)$ using solvers for the LL and LS.

Idea: start solvers with $w_t(\lambda_{s-1})$ and/or $v_k(\lambda_{s-1})$.

$\implies$ Often **great performance improvements** in practice.

$\implies$ Bilevel ϵ-accuracy with **constant number of LL and LS steps**.

Warm-start Guarantees for Bilevel Min-Min

Deterministic Total Iteration Complexity:

- AID: $\tilde{O}(\epsilon^{-1}) \implies O(\epsilon^{-1})$, ITD: $\tilde{O}(\epsilon^{-1})$ (Ji et al., 2021).

Stochastic Sample Complexity:

- $O(\epsilon^{-2})$ (Arbel and Mairal, 2021).
- $\tilde{O}(\epsilon^{-2})$ for **single-loop** (1 LL steps) (Chen et al., 2021).

Warm-start Guarantees for Bilevel Min-Min

Deterministic Total Iteration Complexity:

- **AID**: $\tilde{O}(\epsilon^{-1}) \implies O(\epsilon^{-1})$, **ITD**: $\tilde{O}(\epsilon^{-1})$ (Ji et al., 2021).

Stochastic Sample Complexity:

- $O(\epsilon^{-2})$ (Arbel and Mairal, 2021).
- $\tilde{O}(\epsilon^{-2})$ for **single-loop** (1 LL steps) (Chen et al., 2021).

Comments:

- More complex Analysis **(couples the upper and lower levels)**.
- No improvement in ϵ in the stochastic setting.
- Better Constant: $\underbrace{\max_{\lambda} \|w(\lambda) - w_0(\lambda)\|}_{\text{no warm-start}} \implies \underbrace{\|w(\lambda_0) - w_0(\lambda_0)\|}_{\text{warm-start}}$.

Value Function Methods

Optimistic Bilevel		Value Function		Lagrangian
$\min_{\lambda\in\Lambda, w(\lambda)\in S_\lambda} E(w(\lambda),\lambda)$	$\Rightarrow$	$\min_{\lambda\in\Lambda, w\in\mathbb{R}^d} E(w,\lambda)$	$\Rightarrow$	$L_\beta(w,\lambda) := E(w,\lambda)$
$S_\lambda \in \arg\min_{w\in\mathbb{R}^d} g(w,\lambda)$		s.t. $g(w,\lambda) - g(w(\lambda),\lambda) \leq 0$		$+\beta(g(w,\lambda) - g(w(\lambda),\lambda))$

$$\nabla_2 L_\beta(w,\lambda) = \nabla_2 E(w,\lambda) + \beta(\nabla_2 g(w,\lambda) - \nabla_2 g(w(\lambda),\lambda) + w'(\lambda)^\top \underbrace{\nabla_1 g(w(\lambda),\lambda)}_{=0})$$

Value Function Methods

Optimistic Bilevel

$$\min_{\lambda\in\Lambda, w(\lambda)\in S_\lambda} E(w(\lambda),\lambda)$$

$$S_\lambda \in \arg\min_{w\in\mathbb{R}^d} g(w,\lambda)$$

$\Rightarrow$

Value Function

$$\min_{\lambda\in\Lambda, w\in\mathbb{R}^d} E(w,\lambda)$$

s.t. $\quad g(w,\lambda) - g(w(\lambda),\lambda) \leq 0$

$\Rightarrow$

Lagrangian

$$L_\beta(w,\lambda) := E(w,\lambda)$$
$$+\beta(g(w,\lambda) - g(w(\lambda),\lambda))$$

$$\nabla_2 L_\beta(w,\lambda) = \nabla_2 E(w,\lambda) + \beta(\nabla_2 g(w,\lambda) - \nabla_2 g(w(\lambda),\lambda)) \implies$$ no $w'(\lambda)$ needed.

Value Function Methods

Optimistic Bilevel

$$\min_{\lambda \in \Lambda, w(\lambda) \in S_\lambda} E(w(\lambda), \lambda)$$

$$S_\lambda \in \arg\min_{w \in \mathbb{R}^d} g(w, \lambda)$$

$\Rightarrow$

Value Function

$$\min_{\lambda \in \Lambda, w \in \mathbb{R}^d} E(w, \lambda)$$

s.t. $\quad g(w, \lambda) - g(w(\lambda), \lambda) \leq 0$

$\Rightarrow$

Lagrangian

$$L_\beta(w, \lambda) := E(w, \lambda)$$
$$+\beta(g(w, \lambda) - g(w(\lambda), \lambda))$$

$$\nabla_2 L_\beta(w, \lambda) = \nabla_2 E(w, \lambda) + \beta(\nabla_2 g(w, \lambda) - \nabla_2 g(w(\lambda), \lambda)) \implies$$ no $w'(\lambda)$ needed.

- $g(\cdot, \lambda)$ **(local) PL** $\implies$ **Deterministic Rates** (Liu et al., 2022)
- $g(\cdot, \lambda)$ **strongly-convex** $\implies$ **Stochastic Rates** (Kwon et al., 2023; Chen et al., 2023c,a)

Other Advances

- **Variance Reduction** (Li et al., 2022; Khanduri et al., 2021; Yang et al., 2021)
    - **Finite-sums Objectives** (Dagréou et al., 2022, 2023)
- **Federated/Decentralized** (Tarzanagh et al., 2022; Yang et al., 2022; Chen et al., 2023b)
- **Generalization** (Bao et al., 2021)
- **LL Multiple Minima.** (Arbel and Mairal, 2022; Sow et al., 2022)

Little known with a NN at the lower-level.

References

Arbel, M. and Mairal, J. (2021). Amortized implicit differentiation for stochastic bilevel optimization. In *International Conference on Learning Representations*.

Arbel, M. and Mairal, J. (2022). Non-convex bilevel games with critical point selection maps. *arXiv preprint arXiv:2207.04888*.

Arjevani, Y., Carmon, Y., Duchi, J. C., Foster, D. J., Srebro, N., and Woodworth, B. (2023). Lower bounds for non-convex stochastic optimization. *Mathematical Programming*, 199(1-2):165–214.

Banach, S. (1922). Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta mathematicae*, 3(1):133–181.

Bao, F., Wu, G., Li, C., Zhu, J., and Zhang, B. (2021). Stability and generalization of bilevel programming in hyperparameter optimization. *Advances in Neural Information Processing Systems*, 34:4529–4541.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854.

Bertinetto, L., Henriques, J. F., Torr, P. H., and Vedaldi, A. (2019). Meta-learning with differentiable closed-form solvers. *ICLR*.

Bertrand, Q., Klopfenstein, Q., Blondel, M., Vaiter, S., Gramfort, A., and Salmon, J. (2020). Implicit differentiation of lasso-type models for hyperparameter optimization. In *International Conference on Machine Learning*, pages 810–821. PMLR.

Bertrand, Q., Klopfenstein, Q., Massias, M., Blondel, M., Vaiter, S., Gramfort, A., Salmon, J., Chevalier, J., Nguyen, T., Thirion, B., et al. (2022). Implicit differentiation for fast hyperparameter selection in non-smooth convex learning. *Journal of Machine Learning Research*, 23(149):1–43.

Blondel, M., Berthet, Q., Cuturi, M., Frostig, R., Hoyer, S., Llinares-López, F., Pedregosa, F., and Vert, J.-P. (2021). Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*.

Bolte, J., Le, T., Pauwels, E., and Silveti-Falls, T. (2021). Nonsmooth implicit differentiation for machine-learning and optimization. *Advances in neural information processing systems*, 34:13537–13549.

Bolte, J., Pauwels, E., and Vaiter, S. (2022). Automatic differentiation of nonsmooth iterative algorithms. *Advances in Neural Information Processing Systems*, 35:26404–26417.

Bottou, L., Curtis, F. E., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311.

Chen, L., Ma, Y., and Zhang, J. (2023a). Near-optimal fully first-order algorithms for finding stationary points in bilevel optimization. *arXiv preprint arXiv:2306.14853*.

Chen, T., Sun, Y., and Yin, W. (2021). Tighter analysis of alternating stochastic gradient method for stochastic nested problems. *arXiv preprint arXiv:2106.13781*.

Chen, X., Huang, M., Ma, S., and Balasubramanian, K. (2023b). Decentralized stochastic bilevel optimization with improved per-iteration complexity. In *International Conference on Machine Learning*, pages 4641–4671. PMLR.

Chen, X., Xiao, T., and Balasubramanian, K. (2023c). Optimal algorithms for stochastic bilevel optimization under relaxed smoothness conditions. *arXiv preprint arXiv:2306.12067*.

Choe, S. K., Neiswanger, W., Xie, P., and Xing, E. (2023). Betty: An automatic differentiation library for multilevel optimization. In *The Eleventh International Conference on Learning Representations*.

Dagréou, M., Ablin, P., Vaiter, S., and Moreau, T. (2022). A framework for bilevel optimization that enables stochastic and global variance reduction algorithms. *Advances in Neural Information Processing Systems*, 35:26698–26710.

Dagréou, M., Moreau, T., Vaiter, S., and Ablin, P. (2023). A lower bound and a near-optimal algorithm for bilevel empirical risk minimization. *arXiv e-prints*, pages arXiv–2302.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org.

Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. (2017). Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1165–1173. JMLR. org.

Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., and Pontil, M. (2018). Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1563–1572.

Frecon, J., Salzo, S., and Pontil, M. (2018). Bilevel learning of the group lasso structure. *Advances in neural information processing systems*, 31.

Ghadimi, S. and Wang, M. (2018). Approximation methods for bilevel programming. *arXiv preprint arXiv:1802.02246*.

Grazzi, R., Franceschi, L., Pontil, M., and Salzo, S. (2020). On the iteration complexity of hypergradient computation. In *International Conference on Machine Learning*, pages 3748–3758. PMLR.

Grazzi, R., Pontil, M., and Salzo, S. (2021). Convergence properties of stochastic hypergradients. In *International Conference on Artificial Intelligence and Statistics*, pages 3826–3834. PMLR.

Grazzi, R., Pontil, M., and Salzo, S. (2022). Bilevel optimization with a lower-level contraction: Optimal sample complexity without warm-start. *arXiv preprint arXiv:2202.03397*.

Griewank, A. and Walther, A. (2008). *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. Siam.

Ji, K., Yang, J., and Liang, Y. (2021). Bilevel optimization: Convergence analysis and enhanced design. In *International Conference on Machine Learning*, pages 4882–4892. PMLR.

Khanduri, P., Zeng, S., Hong, M., Wai, H.-T., Wang, Z., and Yang, Z. (2021). A momentum-assisted single-timescale stochastic approximation algorithm for bilevel optimization. *arXiv preprint arXiv:2102.07367*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR (Poster)*.

Kwon, J., Kwon, D., Wright, S., and Nowak, R. D. (2023). A fully first-order method for stochastic bilevel optimization. In *International Conference on Machine Learning*, pages 18083–18113. PMLR.

Lee, K., Maji, S., Ravichandran, A., and Soatto, S. (2019). Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10657–10665.

Li, J., Gu, B., and Huang, H. (2022). A fully single loop algorithm for bilevel optimization without hessian inverse. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7426–7434.

Liu, B., Ye, M., Wright, S., Stone, P., and Liu, Q. (2022). Bome! bilevel optimization made easy: A simple first-order approach. *Advances in Neural Information Processing Systems*, 35:17248–17262.

Maclaurin, D., Duvenaud, D., and Adams, R. (2015). Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122.

Muñoz-González, L., Biggio, B., Demontis, A., Paudice, A., Wongrassamee, V., Lupu, E. C., and Roli, F. (2017). Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38.

Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate o (1/k^ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547.

Ochs, P., Ranftl, R., Brox, T., and Pock, T. (2015). Bilevel optimization with nonsmooth lower level problems. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 654–665. Springer.

Ochs, P., Ranftl, R., Brox, T., and Pock, T. (2016). Techniques for gradient-based bilevel optimization with non-smooth lower level problems. *Journal of Mathematical Imaging and Vision*, 56:175–194.

Pedregosa, F. (2016). Hyperparameter optimization with approximate gradient. In *International Conference on Machine Learning*, pages 737–746.

Rajeswaran, A., Finn, C., Kakade, S. M., and Levine, S. (2019). Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*, pages 113–124.

Sow, D., Ji, K., Guan, Z., and Liang, Y. (2022). A primal-dual approach to bilevel optimization with multiple inner minima. *arXiv preprint arXiv:2203.01123*.

Tarzanagh, D. A., Li, M., Thrampoulidis, C., and Oymak, S. (2022). Fednest: Federated bilevel, minimax, and compositional optimization. In *International Conference on Machine Learning*, pages 21146–21179. PMLR.

Yang, J., Ji, K., and Liang, Y. (2021). Provably faster algorithms for bilevel optimization. *Advances in Neural Information Processing Systems*, 34:13670–13682.

Yang, S., Zhang, X., and Wang, M. (2022). Decentralized gossip-based stochastic bilevel optimization over communication networks. *Advances in Neural Information Processing Systems*, 35:238–252.