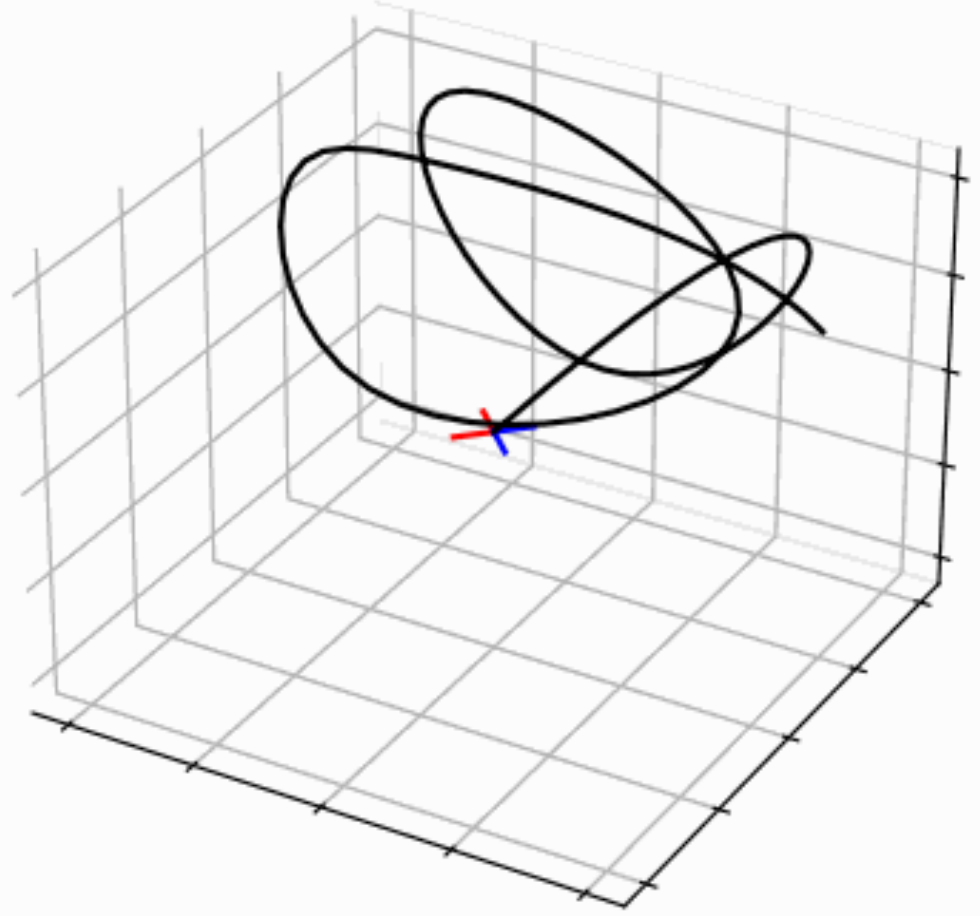
Learning Algorithm Hyperparameters for Fast Parametric Convex Optimization with Certified Robustness

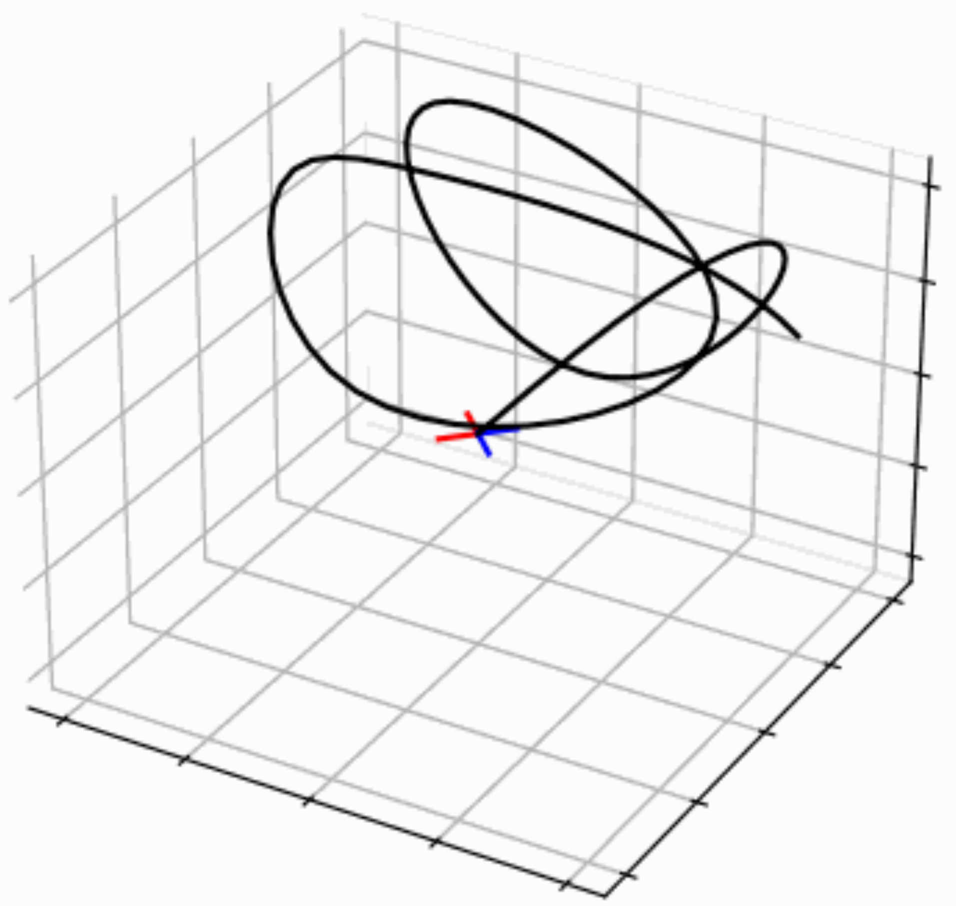
ICCOPT 2025
Rajiv Sambharya

University of Pennsylvania

Tracking a reference trajectory with a quadcopter



Success!
(If given enough time)



Failure: not enough time to solve

Model predictive control
optimize over a smaller horizon (T steps),
implement first control,
repeat

Current state,
reference trajectory

→

Model predictive controller

$$\begin{array}{ll} \text{minimize} & \sum_{t=1}^{T} \|s_t - s_t^{\text{ref}}\|_2^2 \\ \text{subject to} & s_{t+1} = A s_t + B u_t \\ & s_t \in \mathcal{S}, \quad u_t \in \mathcal{U} \\ & s_0 = s_{\text{init}} \end{array}$$

→

Control
inputs

Real-world optimization is parametric

Parameter

$x \longrightarrow$

minimize	$f(z, x)$
subject to	$g(z, x) \leq 0$

f and g convex in z

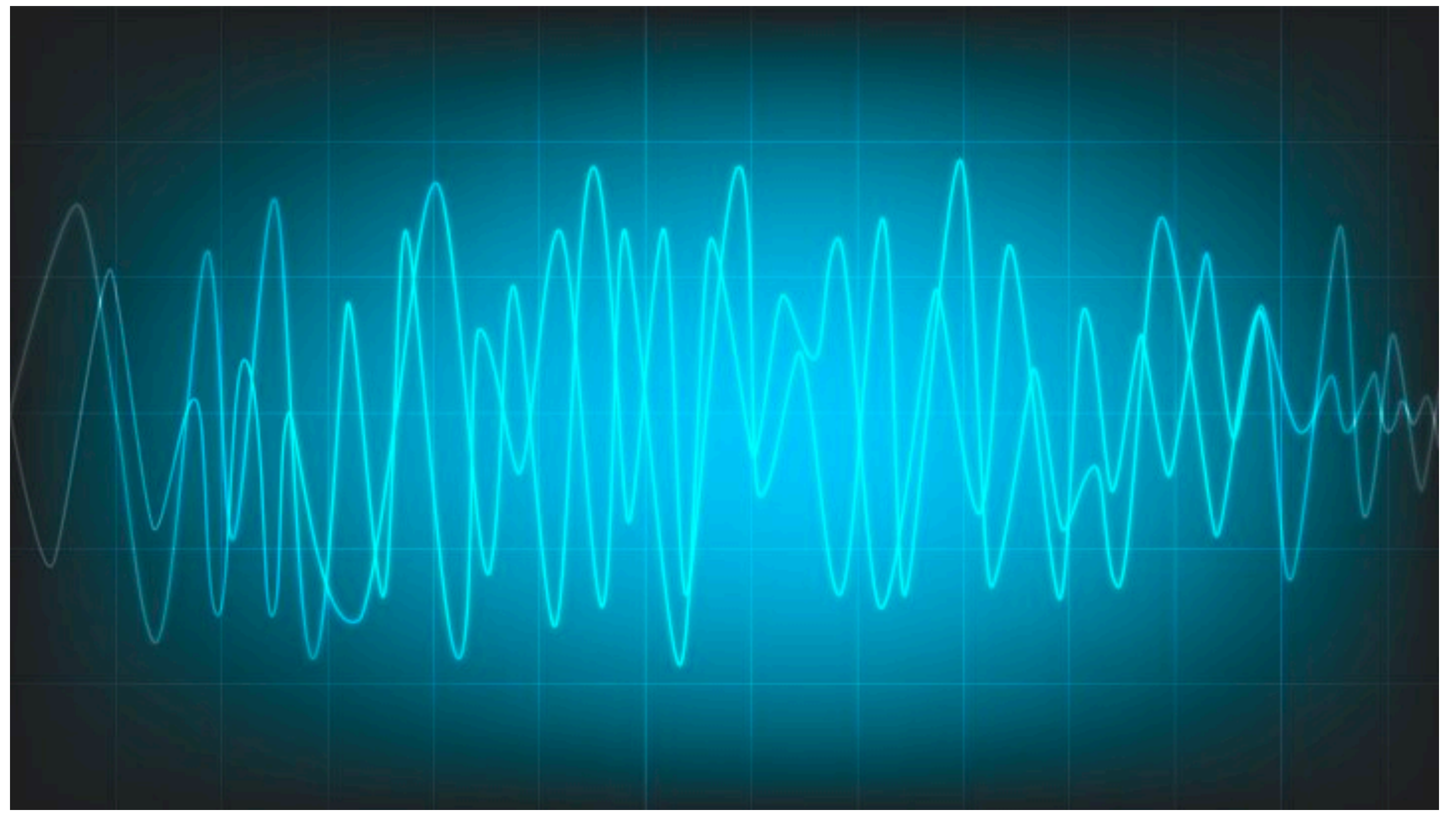
Optimal solution

$\longrightarrow z^\star(x)$

Robotics and control

Energy

Signal processing



First-order methods are widely popular again…

First-order methods use only gradient information

Fixed-point iterations $z^{k+1}(x) = T(z^k(x), x)$

Example: projected gradient descent

$$\begin{array}{lll} \text{minimize} & f(z,x) & \text{convex, smooth} \\ \text{subject to} & z \in \mathcal{C}(x) & \text{convex set} \end{array}$$

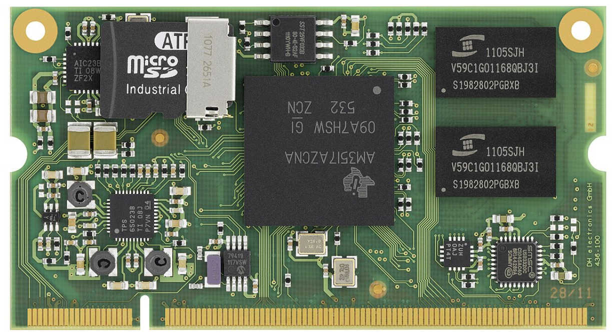
$$z^{k+1}(x) = \Pi_{\mathcal{C}(x)}(\underbrace{z^k(x) - \alpha \nabla f(z^k(x), x)}_{\text{gradient step}})$$

projection

Benefits of first-order methods

cheap iterations ✔

embedded optimization



large-scale optimization

Many general-purpose solvers…

PDLP — Applegate et al. 2021

OSQP — Stellato et al. 2020

SCS SPLITTING CONIC SOLVER — O'Donoghue et al. 2016

…But general-purpose first-order methods can converge slowly

Initialize	$z^0(x) = 0$
Algorithm steps	$z^{k+1}(x) = T(z^k(x), x)$
Terminate when	$\|z^{k+1}(x) - z^k(x)\|_2 \leq \epsilon$

Fixed point

$$z^\star(x) = T(z^\star(x), x)$$

Optimal solution

Problem!

In many applications, we have a **budget** of iterations
(*e.g.*, I only have the time to run 20 fixed-point steps)

Can machine learning speed up convex parametric optimization?

Goal: Do mapping quickly and accurately

Parameter $x \longrightarrow$

$$\begin{array}{ll} \text{minimize} & f(z,x) \\ \text{subject to} & g(z,x) \leq 0 \end{array}$$

Optimal solution $\longrightarrow z^\star(x)$

$x \longrightarrow$ Only Optimization $\longrightarrow \hat{z}^{\mathrm{Opt}}(x)$

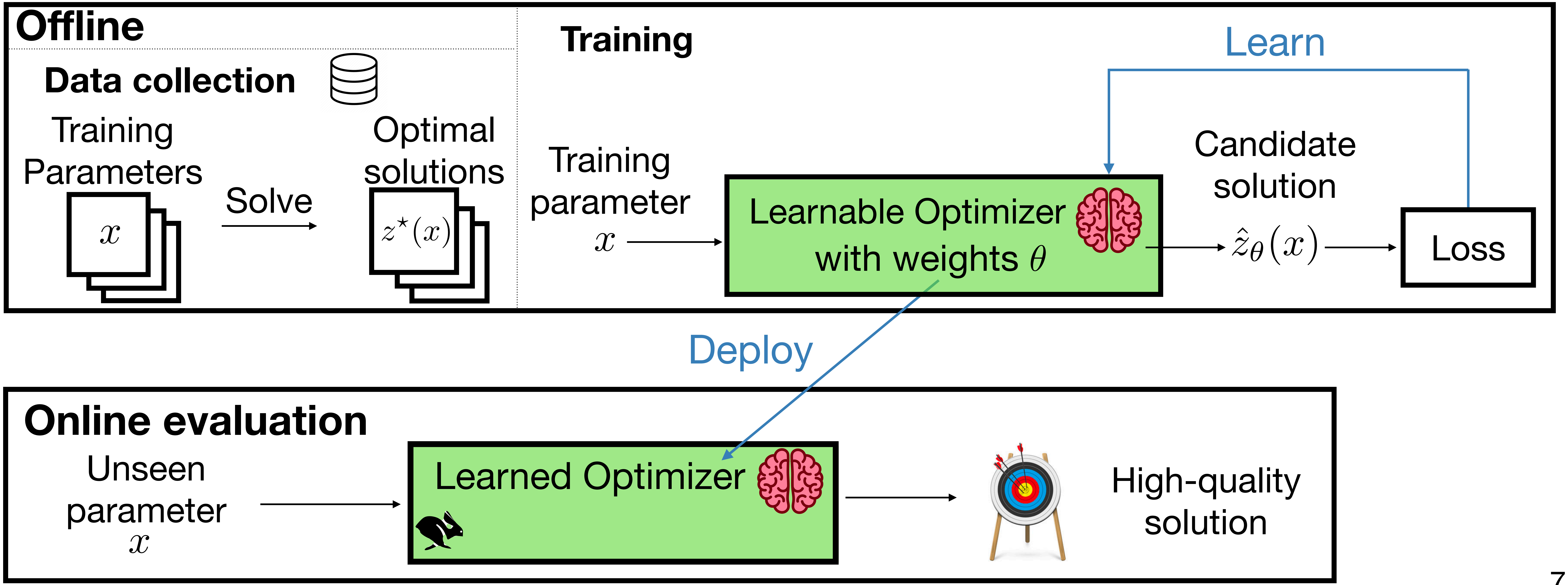
$x \longrightarrow$ Only Machine Learning $\longrightarrow \hat{z}^{\mathrm{ML}}(x)$

$x \longrightarrow$ Optimization Machine Learning $\longrightarrow \hat{z}^{\mathrm{Opt+ML}}(x)$

The learning to optimize paradigm

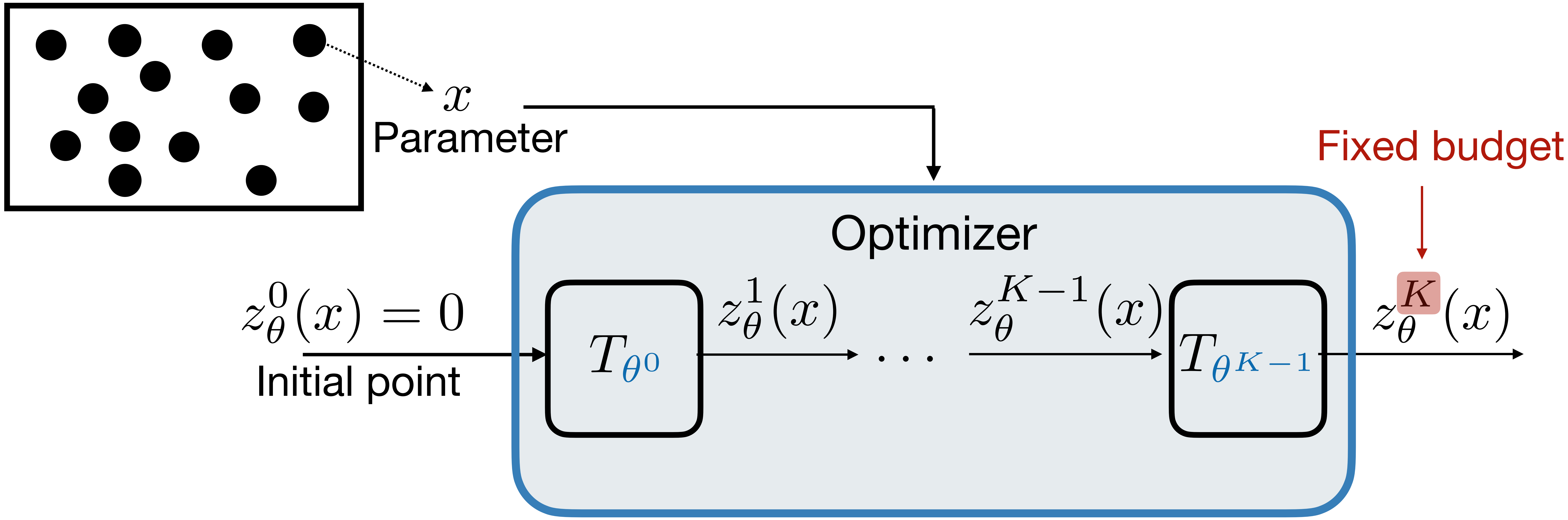
Goal: solve the parametric optimization problem fast

$$\begin{array}{ll} \text{minimize} & f(z, x) \\ \text{subject to} & g(z, x) \leq 0 \end{array}$$



Learning Algorithm Hyperparameters

First-order methods as fixed-length computational graphs



Algorithm Hyperparameters (*e.g.*, step sizes)

$(\theta^0, \dots, \theta^{K-1})$

Example: projected gradient descent

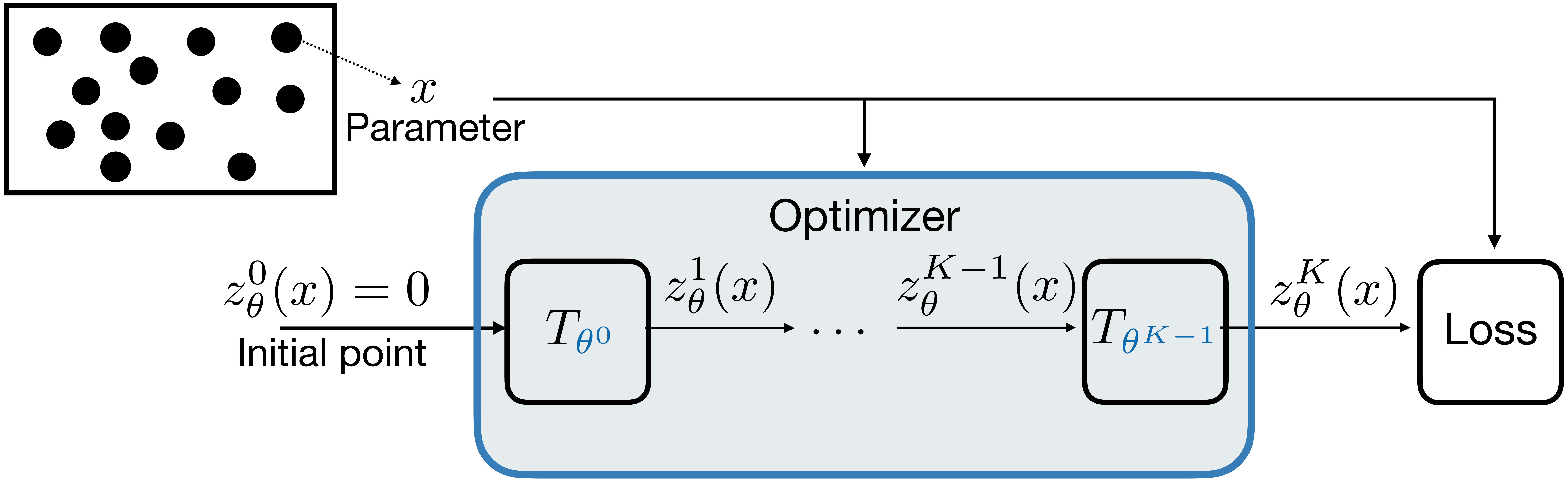
$$z_\theta^{k+1}(x) = \Pi_{\mathcal{C}(x)}(z_\theta^k(x) - \theta^k \nabla_z f(z_\theta^k(x), x))$$

- Conventional wisdom: use a constant step size
- Recent advances: vary the step size!

Altschuler et. al 2023, Grimmer 2023, Bok et. al 2024

What if we **learn** the step sizes?

Learning the algorithm hyperparameters framework



Provided N training instances: $\{(x_i, z^\star(x_i))\}_{i=1}^N$

Training problem

$$
\begin{array}{ll}
\text{minimize} & (1/N)\sum_{i=1}^N \|z_\theta^K(x_i) - z^\star(x_i)\|_2^2 \\
\text{subject to} & z_\theta^{k+1}(x_i) = T_{\theta^k}(z_\theta^k(x_i)) \\
 & z_\theta^0(x_i) = 0
\end{array}
$$

Optimize θ with gradient-based methods

Learned hyperparameters

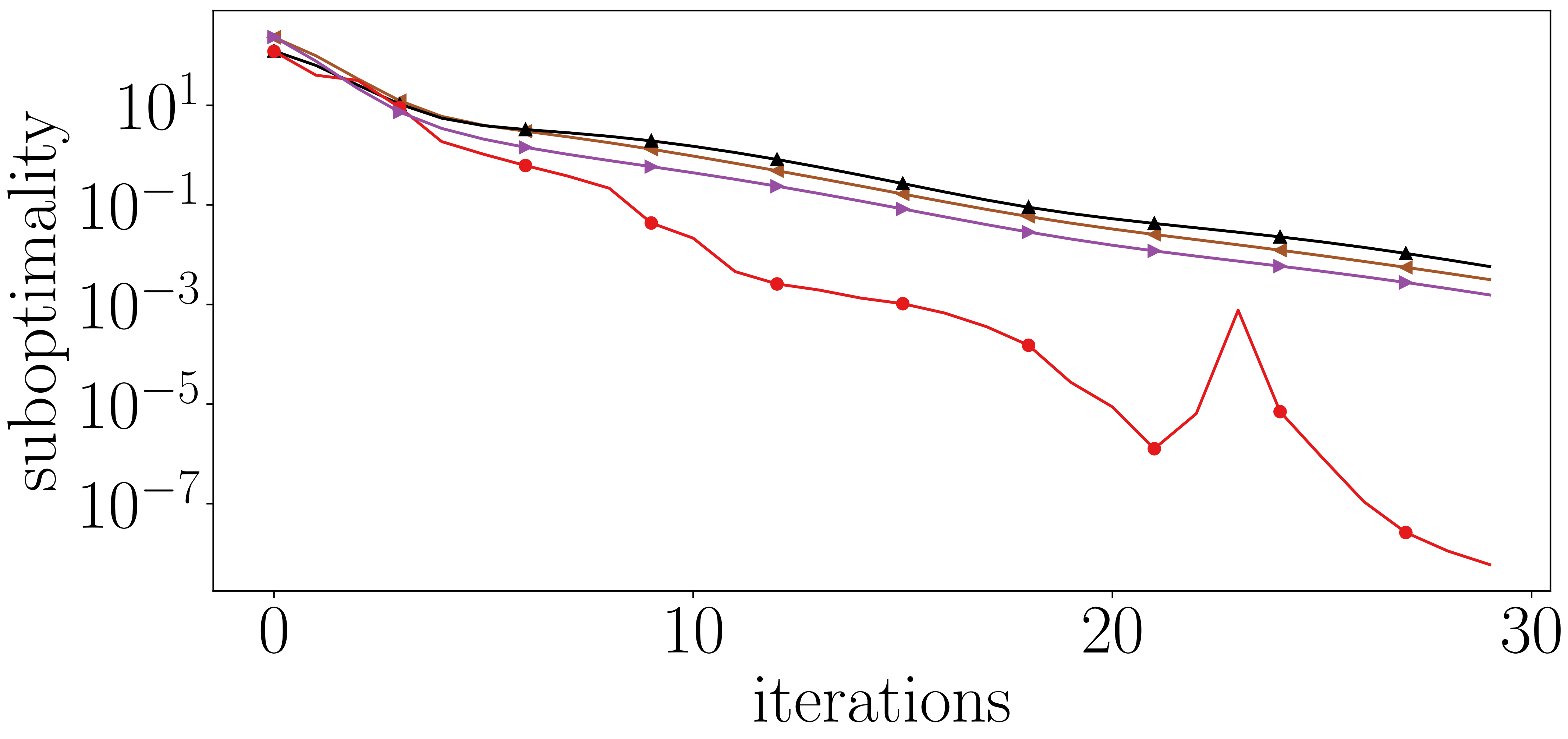
- shared across problem instances
- differ across iterations

Learning step sizes for non-negative least squares

Parameter
x

$$\begin{array}{ll} \text{minimize} & (1/2)\|Az - x\|_2^2 \\ \text{subject to} & z \geq 0 \end{array}$$

Solution
$z^\star(x)$



5 orders
of magnitude

Learning step sizes can be powerful

This is a highly data-efficient approach

Multi-step descent phenomenon

Nesterov

Nearest neighbor warm start

Learned warm starts

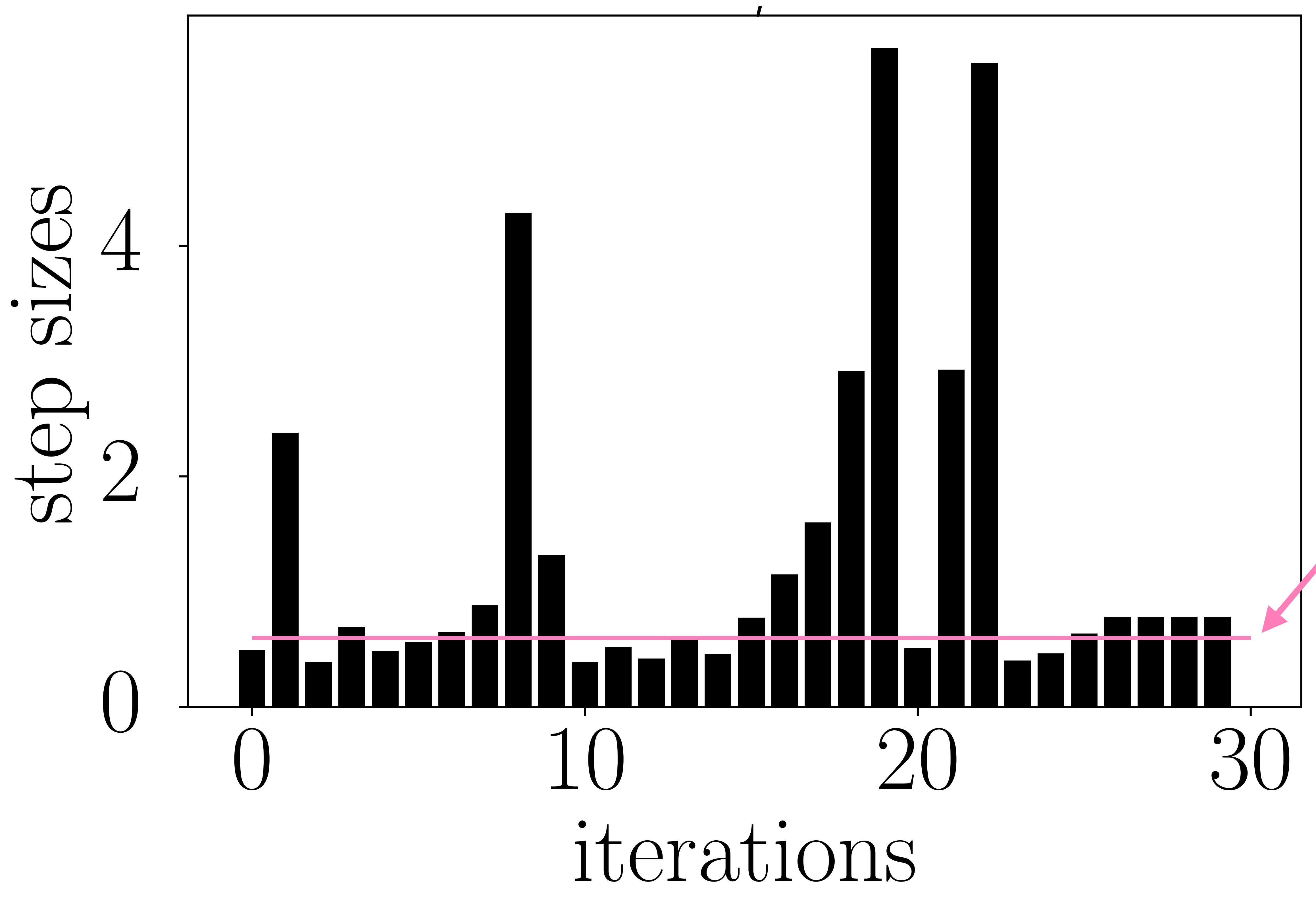
Sambharya et. al 2024

Learned step sizes

10000 training instances

10 training instances

We learn long steps!



Threshold to
guarantee convergence
for constant step size
$2 \,/\, L$

An extension: we can also learn momentum sizes

Composite convex optimization

$$\text{minimize} \quad f(z, x) + g(z, x)$$

Convex, smooth   Convex

Proximal operator

$$\mathbf{prox}_g(v) = \arg\min_u g(u) + (1/2)\|u - v\|_2^2$$

Nesterov's acceleration

$$y^{k+1}(x) = \mathbf{prox}_{\alpha^k g}(z^k(x) - \alpha^k \nabla f(z^k(x), x))$$

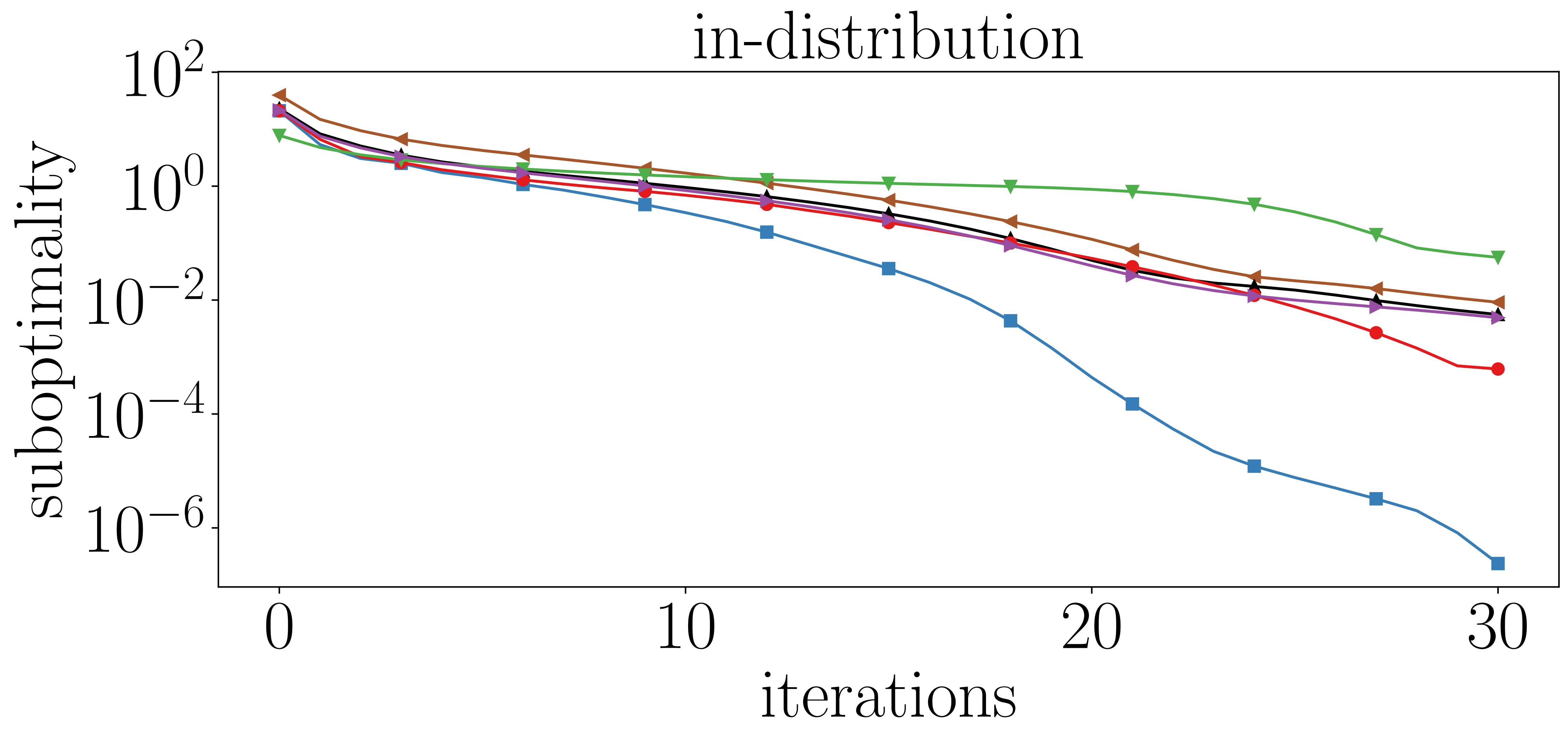
$$z^{k+1}(x) = y^{k+1}(x) + \beta^k (y^{k+1}(x) - y^k(x))$$

Learn $\theta^k = (\alpha^k, \beta^k)$

Beck and Teboulle 2009

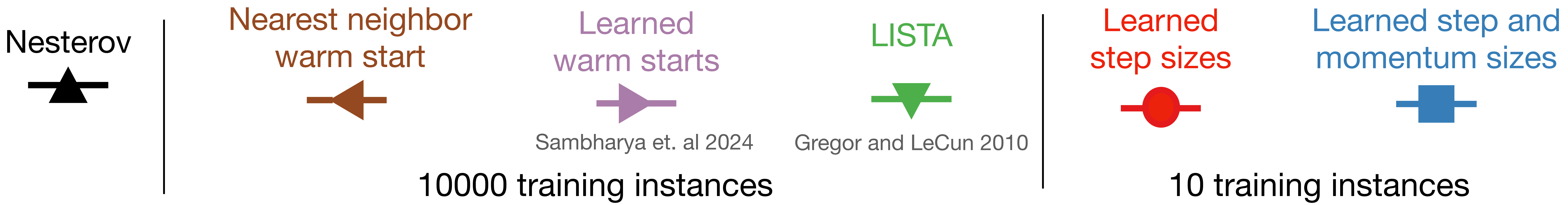
Example: learned hyperparameters for sparse coding

Parameter x → **Signal reconstruction** minimize $(1/2)\|Az - x\|_2^2 + \lambda\|z\|_1$ → Reconstructed signal $z^\star(x)$



Learning momentum steps can sometimes help significantly

But what about worst-case guarantees?



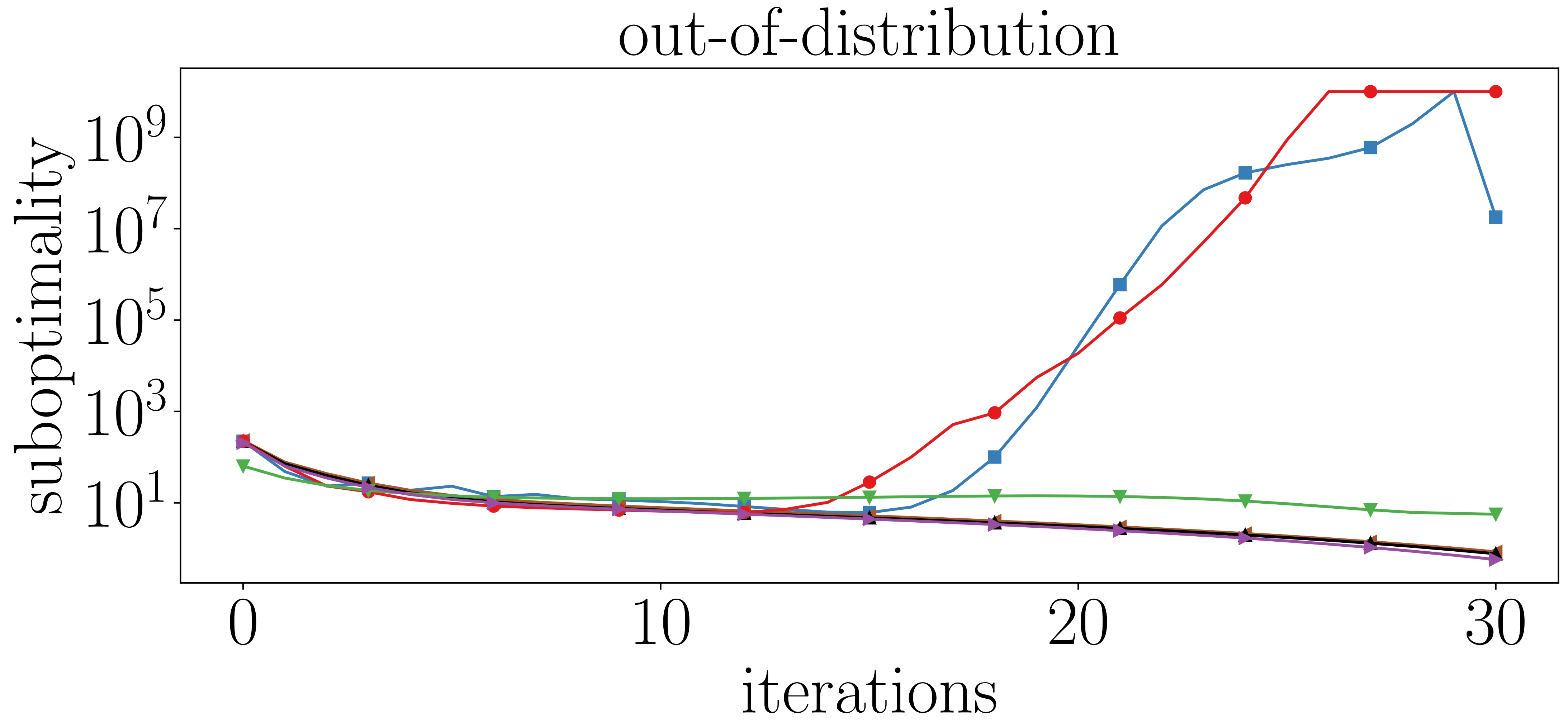
This approach lacks worst-case guarantees

Parameter
x

Signal reconstruction

$$\text{minimize} \quad (1/2)\|Az - x\|_2^2 + \lambda\|z\|_1$$

Reconstructed signal
$z^\star(x)$



Failures on out-of-distribution instances

Can we learn hyperparameters that are robust?

Nesterov | Nearest neighbor warm start | Learned warm starts | LISTA | Learned step sizes | Learned step and momentum sizes

Certifying robustness of algorithms with learned hyperparameters

Can we learn hyperparameters that are robust?

A strong form of robustness—worst-case guarantees for all parameters in a set $\mathcal{X}$

$$r(z_\theta^K(x), x) \leq \gamma(\theta)\|z^0(x) - z^\star(x)\|^2 \quad \forall x \in \mathcal{X}$$

Performance metric: *e.g.,*

$$r(z_\theta^K(x), x) = \|z_\theta^K(x) - z^\star(x)\|_2^2$$

$$r(z_\theta^K(x), x) = f(z_\theta^K(x), x) - f(z^\star(x), x)$$

Level of robustness

A provided set of interest

Ideally, learn θ as before
but constrain $\gamma(\theta) \leq \gamma^{\text{target}}$

🤔

But how can we evaluate $\gamma(\theta)$?

Certified robustness for all parameters in a set

Definition: $(f, \mathcal{X})$ is $\mathcal{F}_{\mu,L}$-parametrized if $f(\cdot, x) \in \mathcal{F}_{\mu,L} \quad \forall x \in \mathcal{X}$

μ-strongly convex, L-smooth

Example:

$$\text{minimize} \quad \underbrace{(1/2)\|Az - x\|_2^2}_{f(z,x)} + \underbrace{\lambda\|z\|_1}_{g(z,x)} \qquad \mathcal{X} = \mathbf{R}^d$$

$(f, \mathcal{X})$ is $\mathcal{F}_{\mu,L}$-parametrized

min and max eigenvalues of $A^T A$

$(g, \mathcal{X})$ is $\mathcal{F}_{0,\infty}$-parametrized

Worst-case guarantees over function class imply worst-case guarantees over set

Can leverage Performance Estimation Problem Analysis

The Performance Estimation Problem (PEP) Framework can help us

$$
\begin{array}{lll}
\text{maximize} & \text{(performance metric)} & r(z^K) \\
\text{subject to} & \text{(initial point)} & z^0 = y^0, \|z^0 - z^\star\|_2^2 \leq 1 \\
& \text{(optimality)} & \nabla f(z^\star) + \partial g(z^\star) = 0 \\
& \text{(algorithm update)} & y^{k+1} = \mathbf{prox}_{\alpha^k g}(z^k - \alpha^k \nabla f(z^k)) \\
& & z^{k+1} = y^{k+1} + \beta^k (y^{k+1} - y^k) \\
& \text{(function class)} & f \in \mathcal{F}_{\mu,L}, g \in \mathcal{F}_{0,\infty}.
\end{array}
$$

$\updownarrow$

PEP: Tight convex SDP formulation using gram matrix G

Drori, Teboulle, Hendrickx, Glineur, Taylor, Ryu, Grimmer, and many more

$$
\theta \longrightarrow
\boxed{
\begin{array}{ll}
\text{maximize} & \mathbf{tr}(A_0 G) \\
\text{subject to} & \mathbf{tr}(A_i(\theta) G) \leq b_i, i = 1, \ldots, m \\
& G \succeq 0
\end{array}
}
\longrightarrow \gamma(\theta)
$$

Level of robustness $\gamma(\theta)$

✔ Solve an SDP to **evaluate** $\gamma(\theta)$

🤔 But how can we **train** for robustness?

Robust training of hyperparameters

PEP-regularized training problem

$$
\begin{array}{ll}
\text{minimize} & (1/N)\sum_{i=1}^{N} \ell(z_\theta^K(x_i), x_i) + \lambda((\gamma(\theta) - \gamma^{\text{target}})_+)^2 \\
\text{subject to} & y_\theta^{k+1}(x_i) = \mathbf{prox}_{\alpha^k}(z_\theta^k(x_i) - \alpha^k \nabla f(z_\theta^k(x_i), x_i)) \\
& z_\theta^{k+1}(x_i) = y_\theta^{k+1}(x_i) + \beta^k(y_\theta^{k+1}(x_i) - y_\theta^k(x_i)) \\
& z_\theta^0(x_i) = 0, y_\theta^0(x_i) = 0
\end{array}
$$

Penalty formulation

differentiable optimization to compute $\dfrac{\partial \gamma(\theta)}{\partial \theta}$

CVXpy

$$
\begin{array}{rl}
x^\star(\theta) = \underset{x}{\text{argmin}} & f(x;\theta) \\
\text{subject to} & g(x;\theta) \leq 0 \\
& h(x;\theta) = 0
\end{array}
$$

PyTorch

TensorFlow

Amos et. al 2017, Agrawal et. al 2019

Learning robust hyperparameters for sparse coding

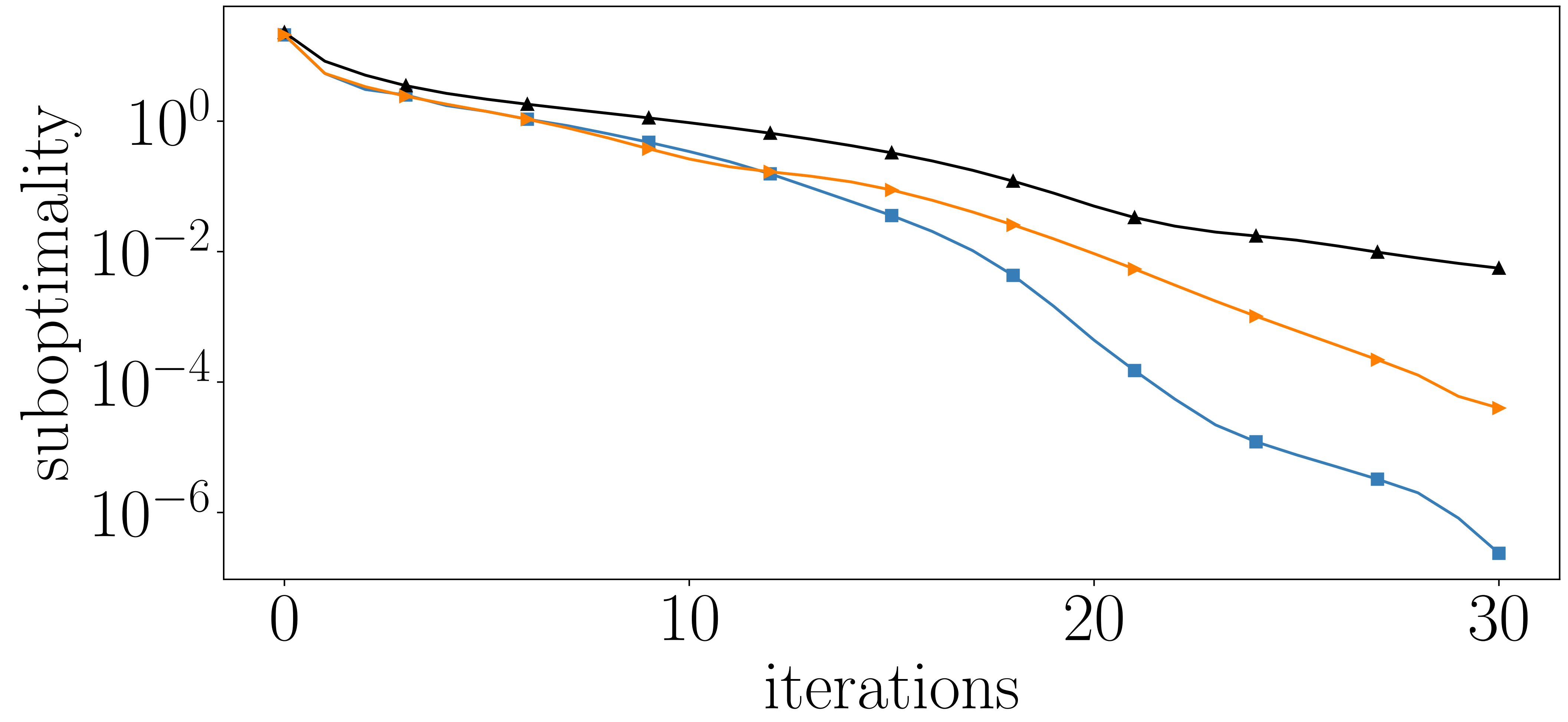
Parameter
x

Signal reconstruction

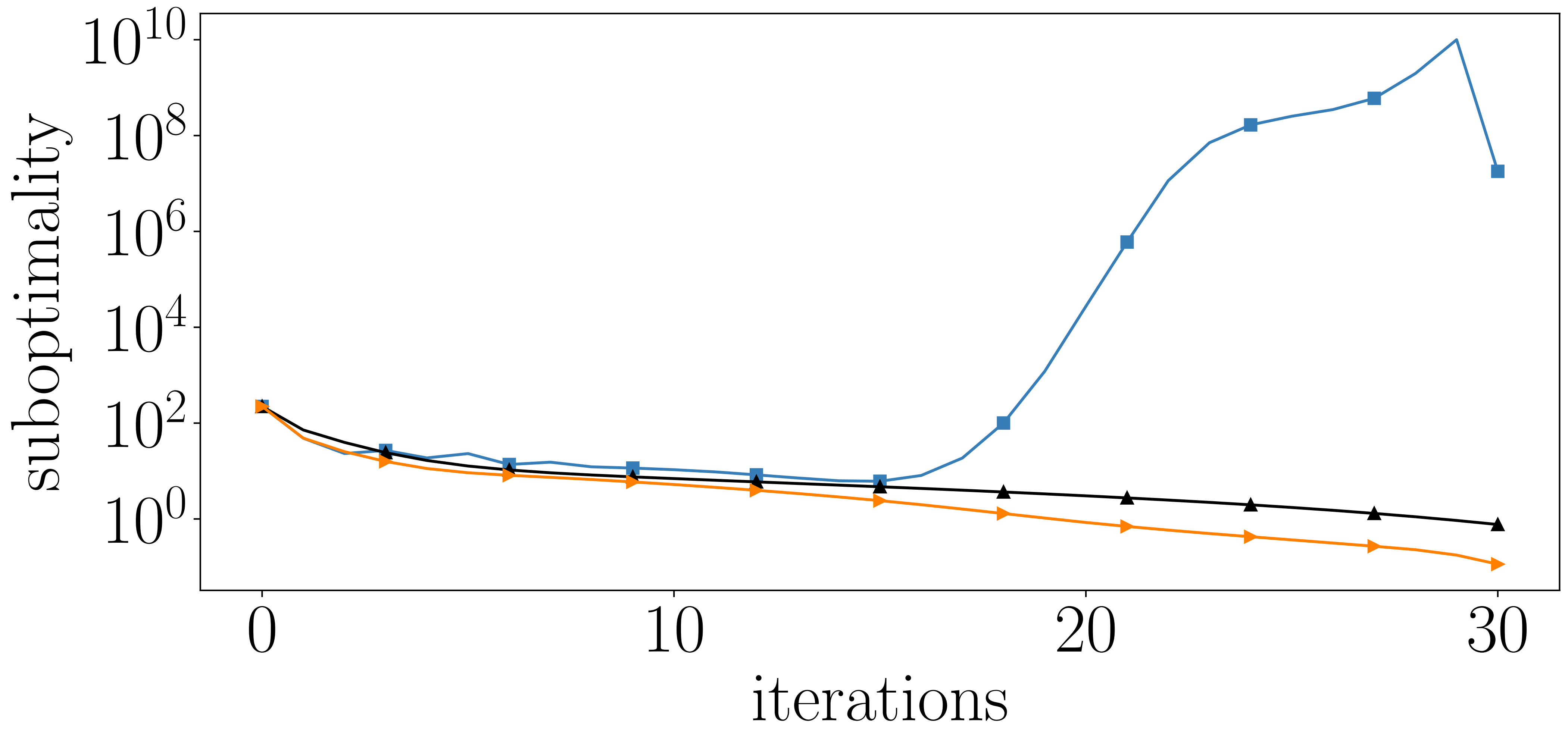
$$\text{minimize} \quad (1/2)\|Az - x\|_2^2 + \lambda\|z\|_1$$

Reconstructed signal
$z^\star(x)$

In-distribution



Out-of-distribution



Nesterov	Learned step and momentum sizes - robust	Learned step and momentum sizes
$\gamma = 0.01$	$\gamma = 0.10$	$\gamma = \infty$

We can train and maintain robustnesss

Guarantee holds for any $x \in \mathbf{R}^d$

Learning hyperparameters for the alternating direction method of multipliers (ADMM)

We learn hyperparameters for accelerated ADMM also

Two popular ADMM-based solvers

OSQP

Stellato et al. 2020

SCS

SPLITTING CONIC SOLVER

O'Donoghue et al. 2016

Conic problems

$$\begin{array}{ll} \min & (1/2)w^T P w + c^T w \\ \text{s.t.} & Aw + s = b \\ & s \in \mathcal{K} \end{array}$$

Convex cone

$$\text{with} \quad x = (P, A, c, b)$$

Accelerated Splitting Conic Solver

$$\text{solve} \begin{bmatrix} P + \sigma I & A^T \\ -A & \rho I \end{bmatrix} \tilde{u}^{k+1} = z^k - \begin{bmatrix} c \\ b \end{bmatrix}$$

$$u^{k+1} = \Pi_{\mathbf{R}^q \times \mathcal{K}^*}(2\tilde{u}^{k+1} - z^k)$$

$$y^{k+1} = z^k + \alpha^k(u^{k+1} - \tilde{u}^{k+1})$$

$$z^{k+1} = y^{k+1} + \beta^k(y^{k+1} - y^k)$$

Time-varying hyperparameters (α^k, β^k)

Time-invariant hyperparameters (σ, ρ)

Why time-invariant?

1. Amenable to PEP
2. Computational advantages—reuse matrix factorization

Model predictive control of a quadcopter

Current state,
reference trajectory

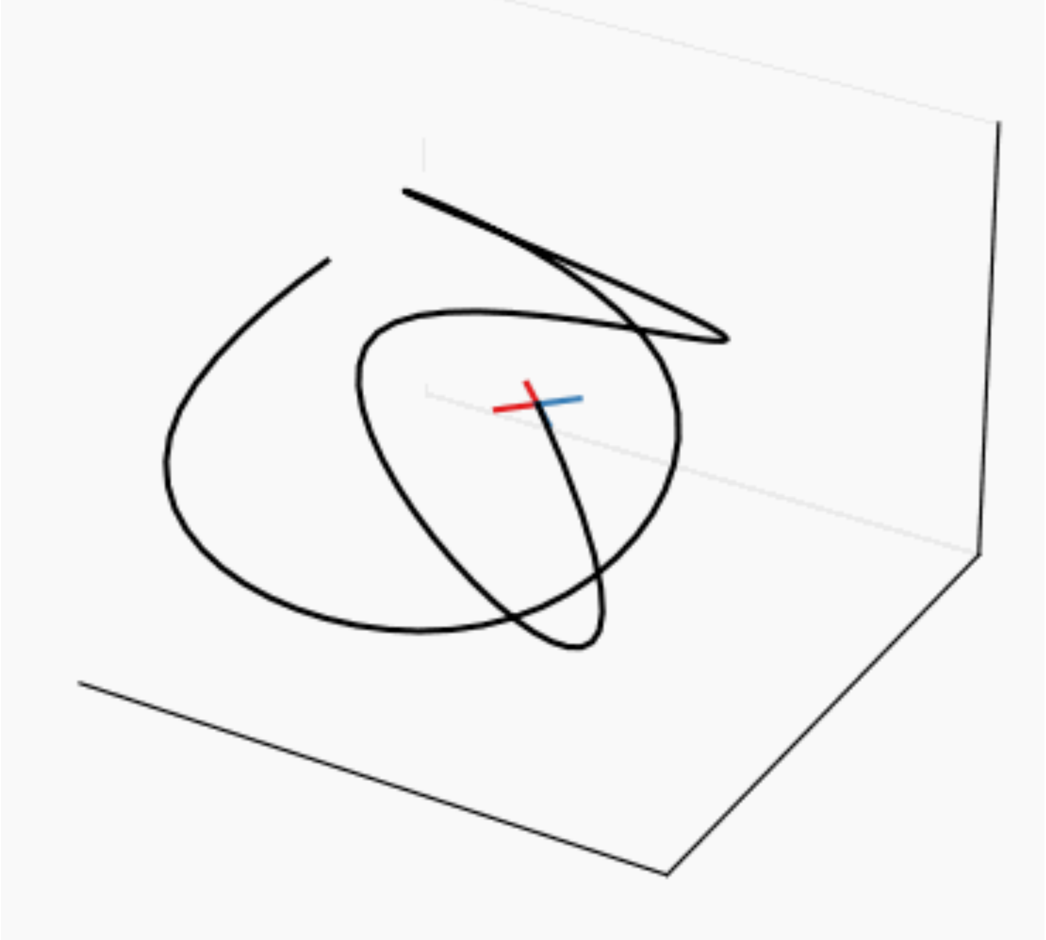
→

Quadratic program

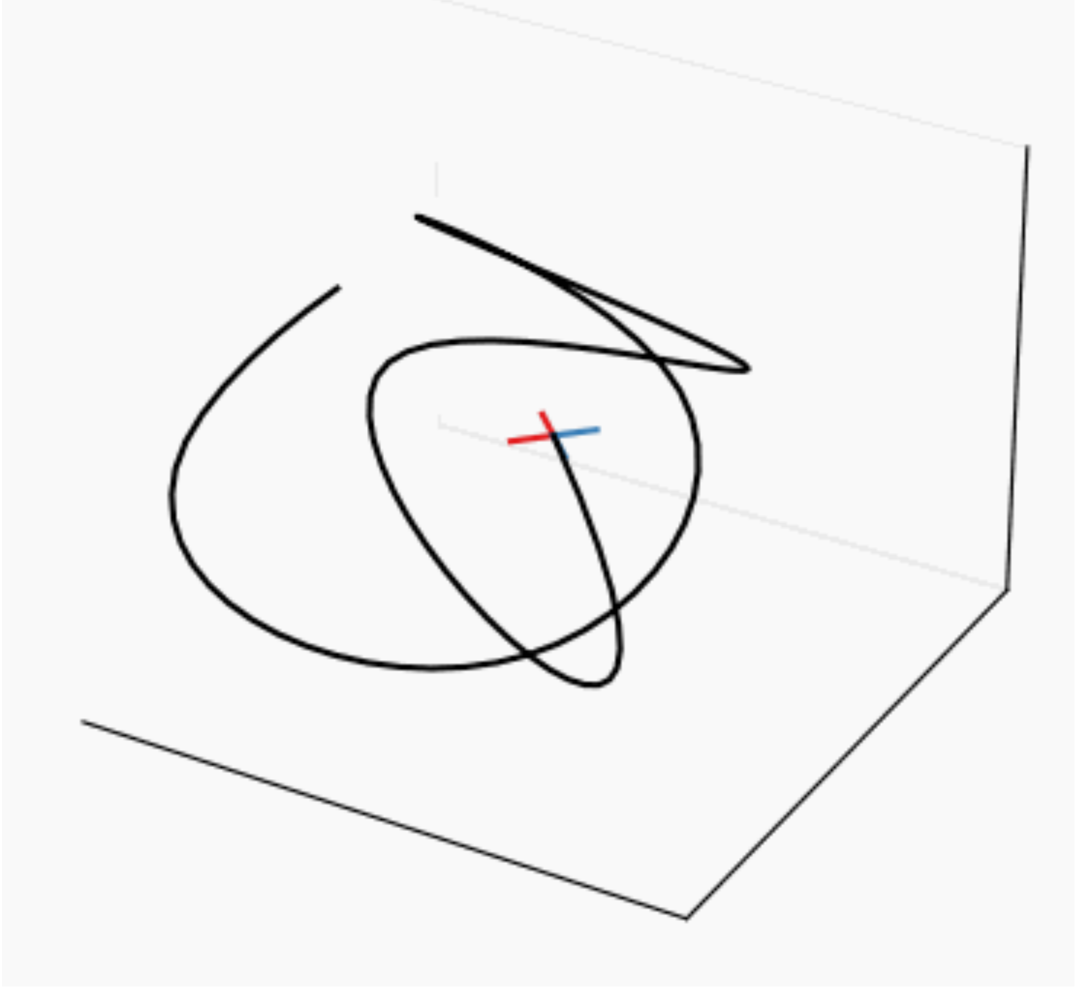
$$\begin{array}{ll} \text{minimize} & \sum_{t=1}^{T} \|s_t - s_t^{\text{ref}}\|_2^2 \\ \text{subject to} & s_{t+1} = A s_t + B u_t \\ & s_t \in \mathcal{S}, \quad u_t \in \mathcal{U} \\ & s_0 = s_{\text{init}} \end{array}$$

→

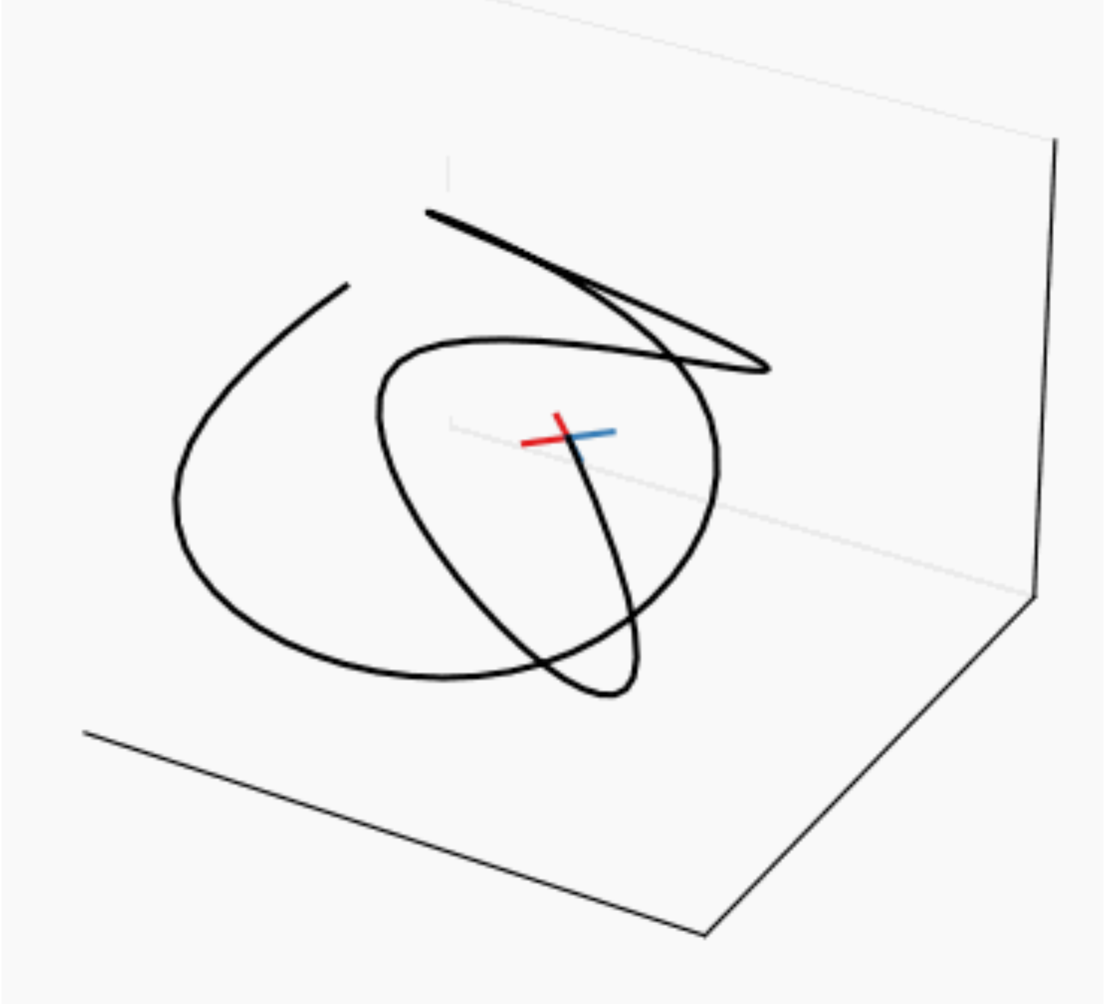
Control
inputs



Nearest neighbor
80 iterations



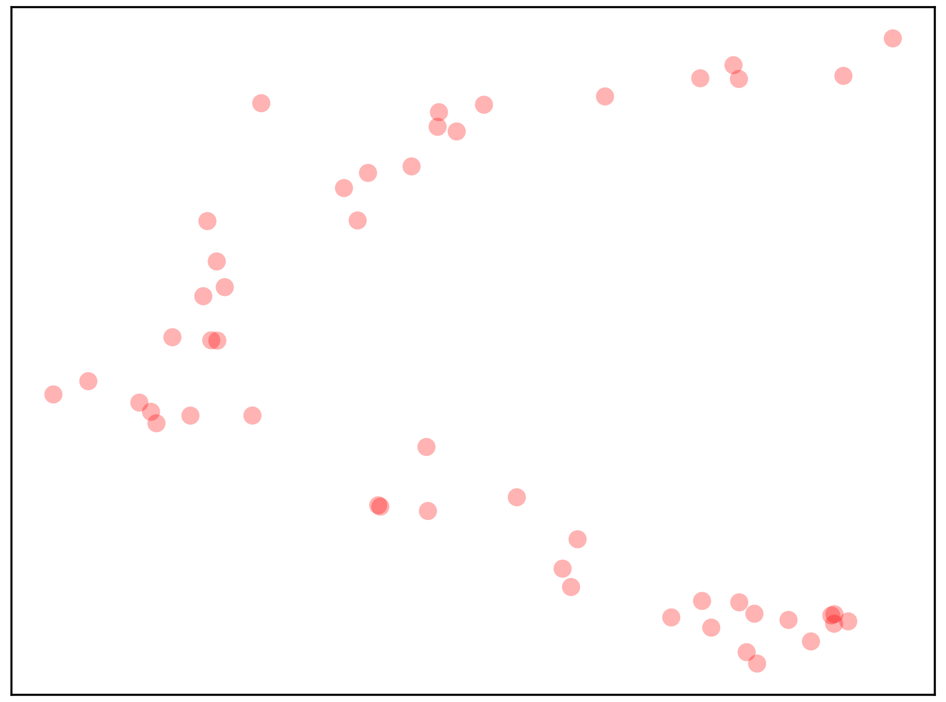
Previous solution
80 iterations



Learned accel + robust
20 iterations

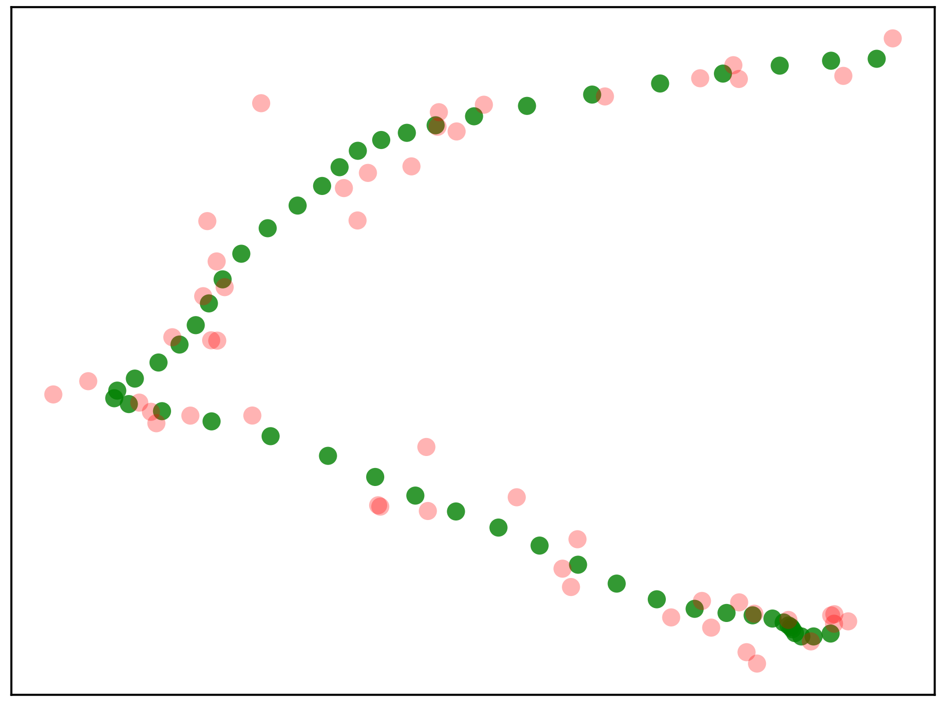
With learning, we can track the trajectory well

Robust Kalman filtering



Second-order cone program

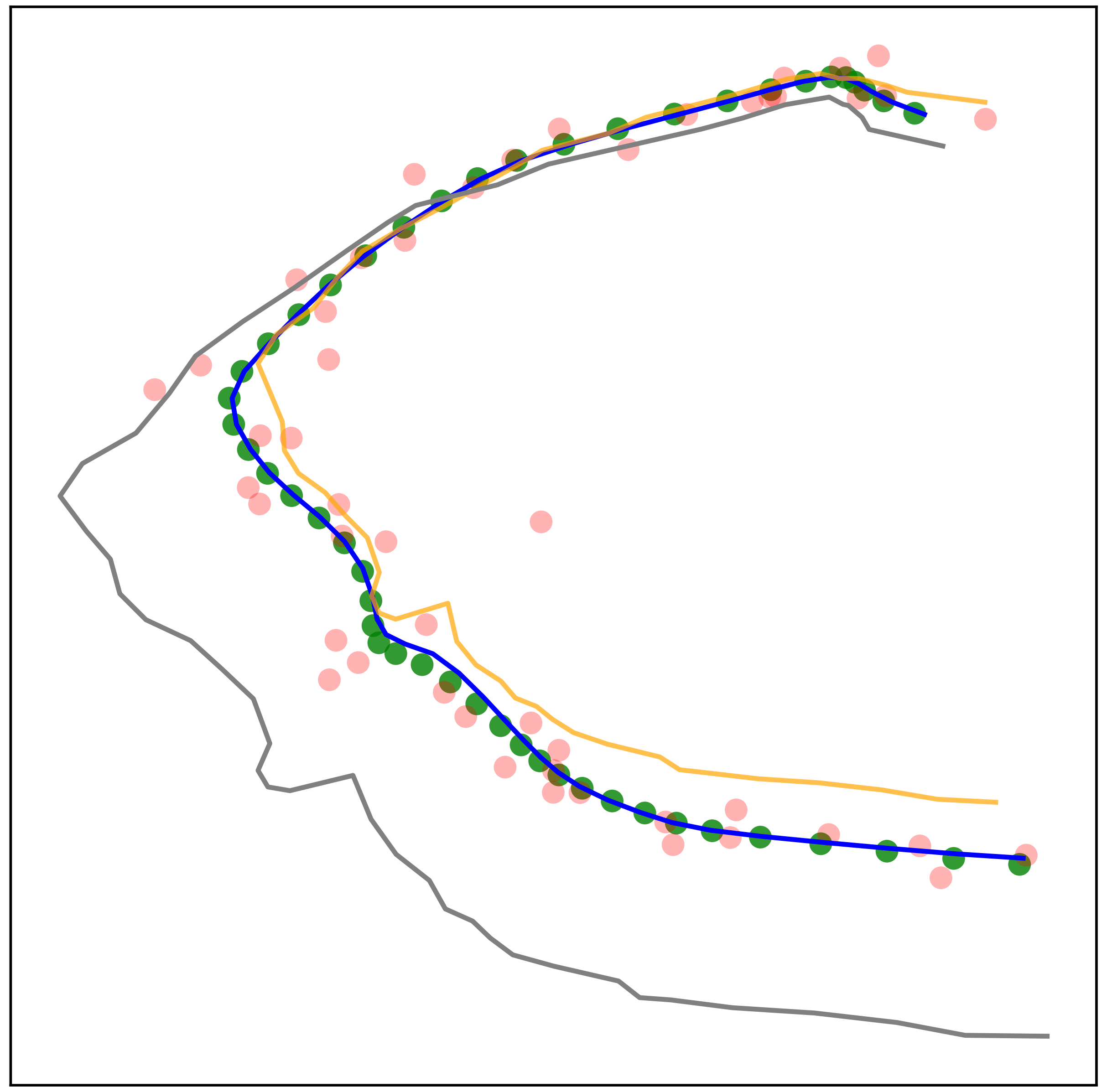
$$
\begin{array}{ll}
\text{minimize} & \sum_{t=0}^{T-1} \|w_t\|_2^2 + \overbrace{\psi_\rho}^{\text{Huber loss}}(v_t) \\
\text{subject to} & s_{t+1} = As_t + Bu_t \\
& y_t = Cs_t + v_t
\end{array}
$$



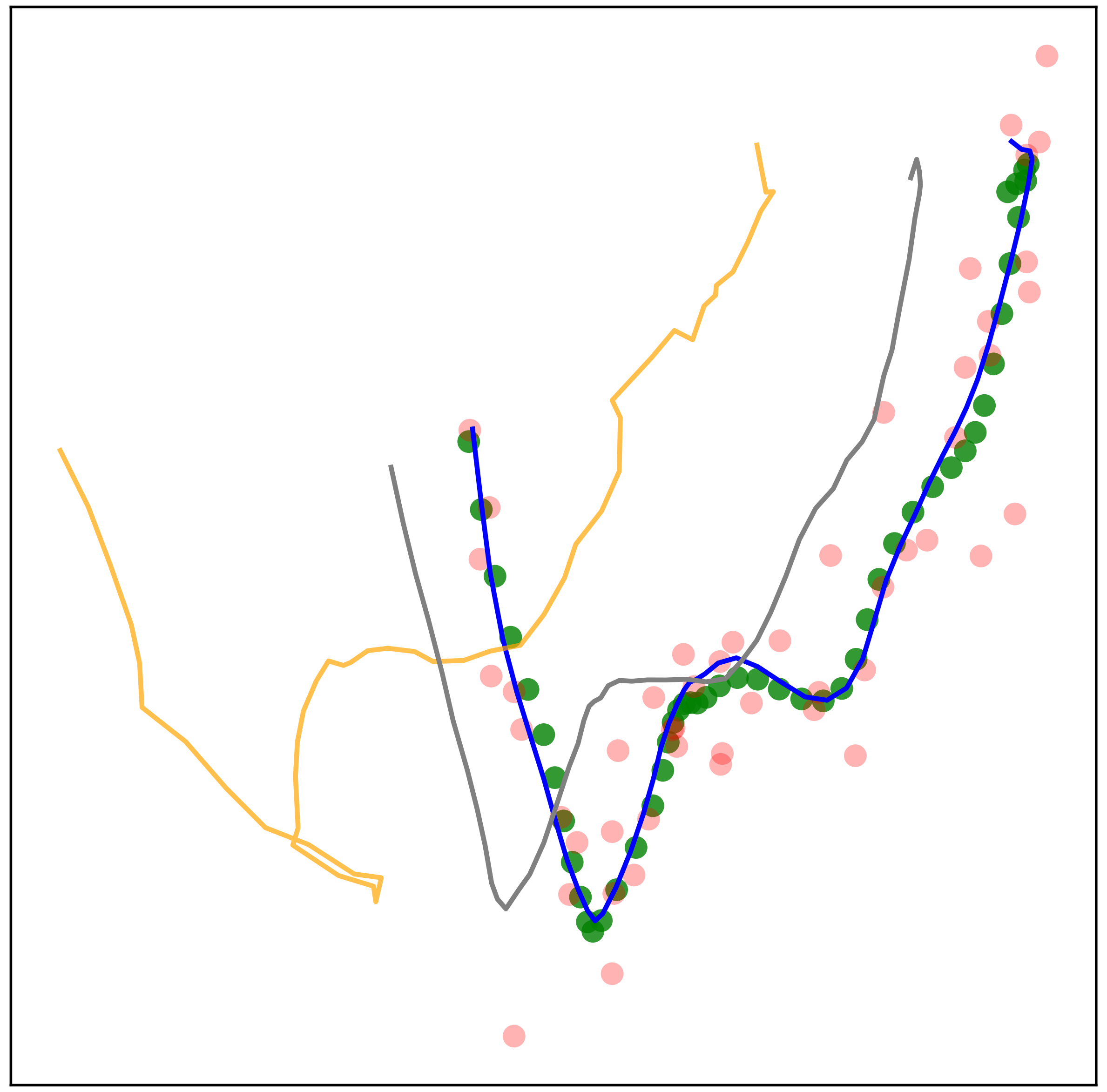
Noisy trajectory

Optimal solution

In-distribution



Out-of-distribution



5 iterations

No learning

Learned hyperparameters

Learned acceleration + Robust

Learning acceleration algorithms w/ robustness tracks the optimal solution

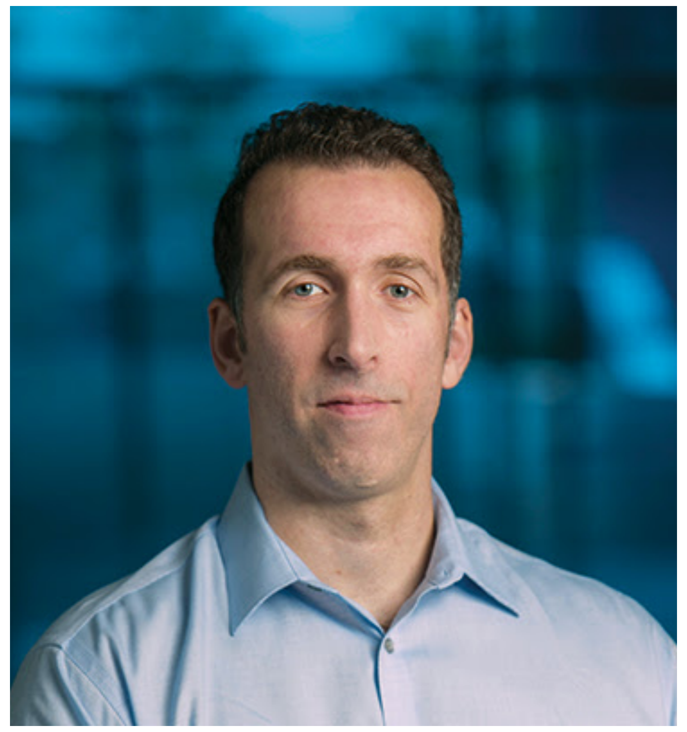
Acknowledgements

Bartolomeo Stellato

PRINCETON UNIVERSITY

Jinho Bok

Penn UNIVERSITY of PENNSYLVANIA



Nikolai Matni

Penn UNIVERSITY of PENNSYLVANIA

George Pappas

Penn UNIVERSITY of PENNSYLVANIA

Learning Algorithm Hyperparameters for Fast Parametric Convex Optimization
R. Sambharya, B. Stellato
https://arxiv.org/pdf/2411.15717

Learning Acceleration Algorithms for Fast Parametric Convex Optimization with Certified Robustness
R. Sambharya, J. Bok, N. Matni, G. Pappas
https://arxiv.org/pdf/2507.16264

Conclusion

Traditional view

- General purpose
- One-size fits all
- With guarantees

Learning to optimize →

New view

- Task-specific
- Trainable
- With guarantees

Takeaways from this talk specifically

- Only **learning** the **hyperparameter sequence** dramatically improves performance
- Very **low amount of training data** needed
- We evaluate and train for **robustness** using PEP

sambhar9@seas.upenn.edu

rajivsambharya.github.io