AI for Optimization: Learning Algorithm Hyperparameters for Fast Parametric Convex Optimization

University of Delaware RGSO-CARS Seminar
Fall Seminar Sept 18, 2026
Rajiv Sambharya

TEXAS A&M UNIVERSITY®

Optimization is a powerful, flexible tool for decision-making

$$\begin{array}{ll} \text{minimize} & f(z) \\ \text{subject to} & z \in \Omega \end{array}$$

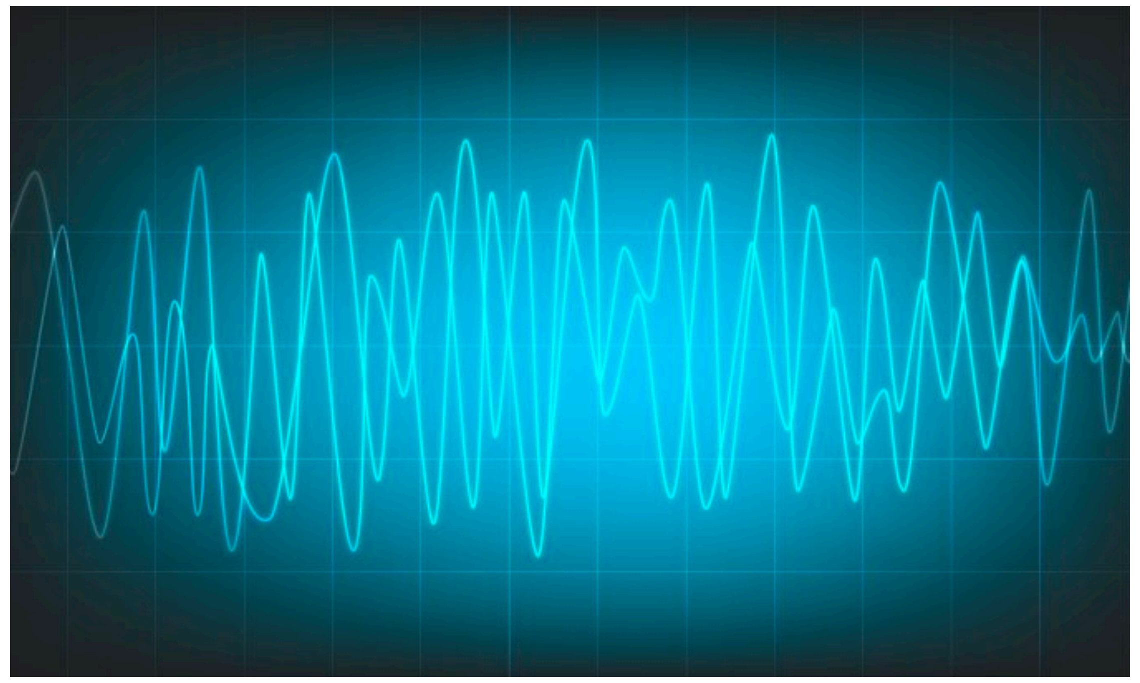
robotics + control

thrusts, torques

energy

power flow

signal processing

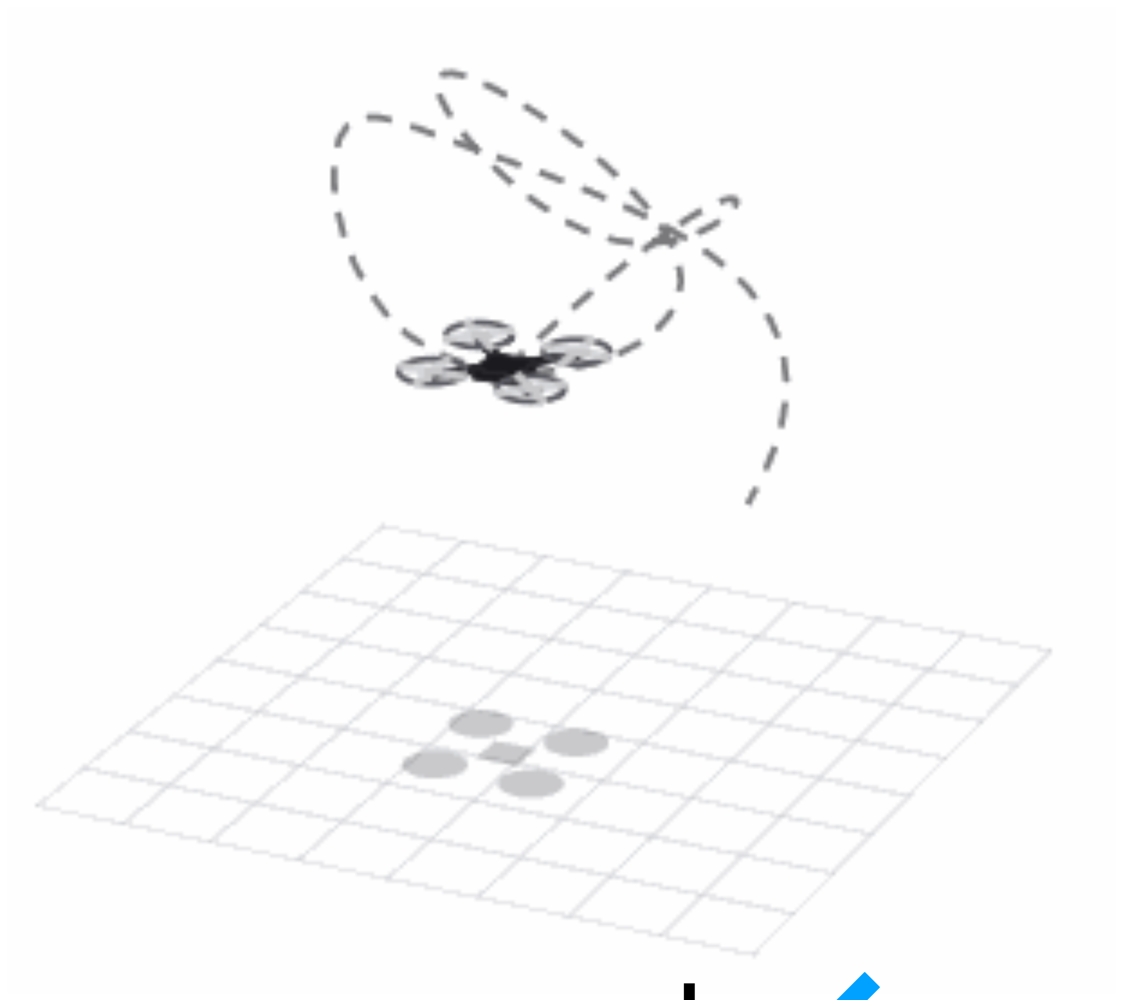


underlying signal

decision variable z

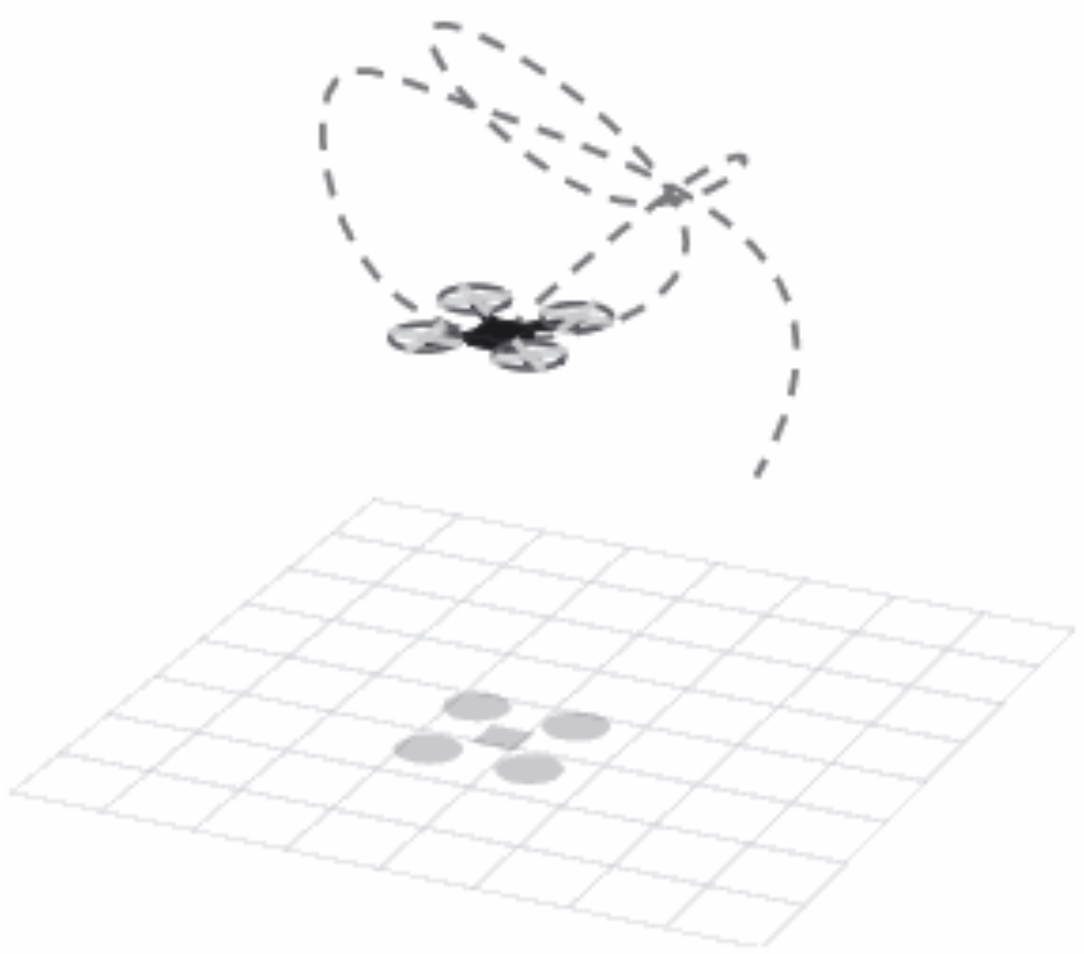
running example

Quadcopter trajectory-tracking with optimization



success!

(if given enough time)



failure!

not enough time to solve

Model predictive control

optimize over a smaller horizon (T steps),
implement first action,
repeat

current state,
reference trajectory →

optimization

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

→ actions

we need good algorithms to enable real-time optimization!

Algorithms are the computational engine for optimization

$$\begin{array}{ll} \text{minimize} & f(z) \\ \text{subject to} & z \in \Omega \end{array}$$

how can we design and analyze optimization algorithms?

we want the fastest algorithms in practice

we want the strongest guarantees for those algorithms

traditional viewpoint

optimization algorithms are **general-purpose**

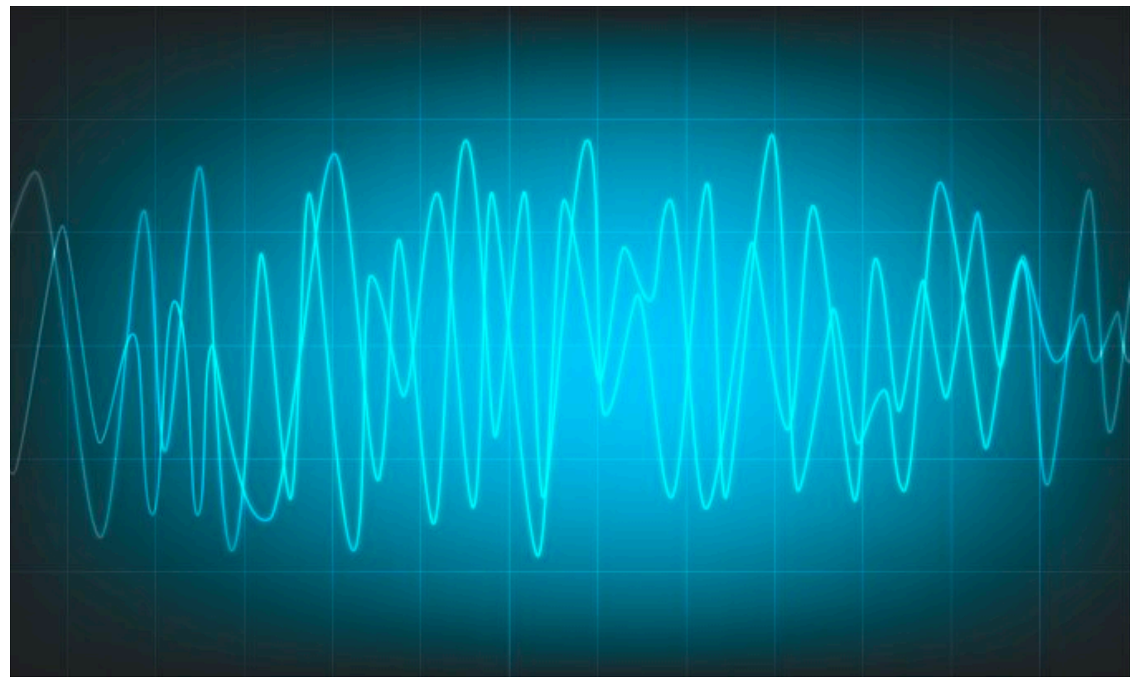
problem data $\longrightarrow$ $\boxed{\text{solver}}$ $\longrightarrow$ solution

Real-world optimization is parametric

problem parameter x →

$$\boxed{\begin{array}{ll} \text{minimize} & f(z,x) \\ \text{subject to} & z \in \Omega(x) \end{array}}$$

→ optimal solution $z^\star(x)$

	robotics + control	**energy**	**signal processing**
			
problem parameter x	initial state	demand, renewable generation	noisy measurement
	↓	↓	↓
decision variable z	thrusts, torques	power flow	underlying signal

Parametric optimization presents new opportunities

problem parameter x →

$$
\begin{array}{ll}
\text{minimize} & f(z, x) \\
\text{subject to} & z \in \Omega(x)
\end{array}
$$

→ optimal solution $z^\star(x)$

new opportunity

we have **data**

$\{x_i\}_{i=1}^N$

↓

new viewpoint

Van Hentenryck, Yin, Amos, and others

design and analysis of optimization algorithms

can we exploit the parametric structure to…

make algorithms task-specific?

and develop tighter guarantees?

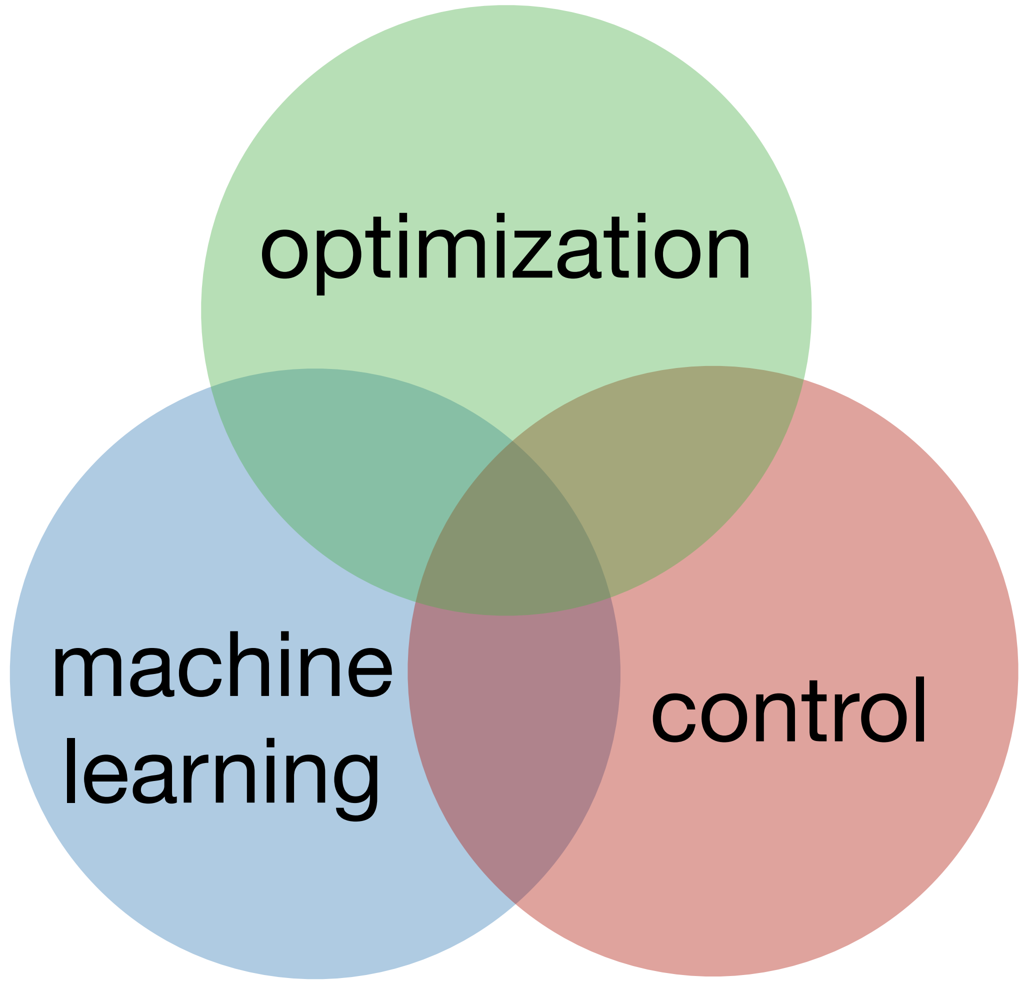
My research in a nutshell: AI for Optimization

problem parameter x $\longrightarrow$

$$\begin{array}{ll} \text{minimize} & f(z, x) \\ \text{subject to} & z \in \Omega(x) \end{array}$$

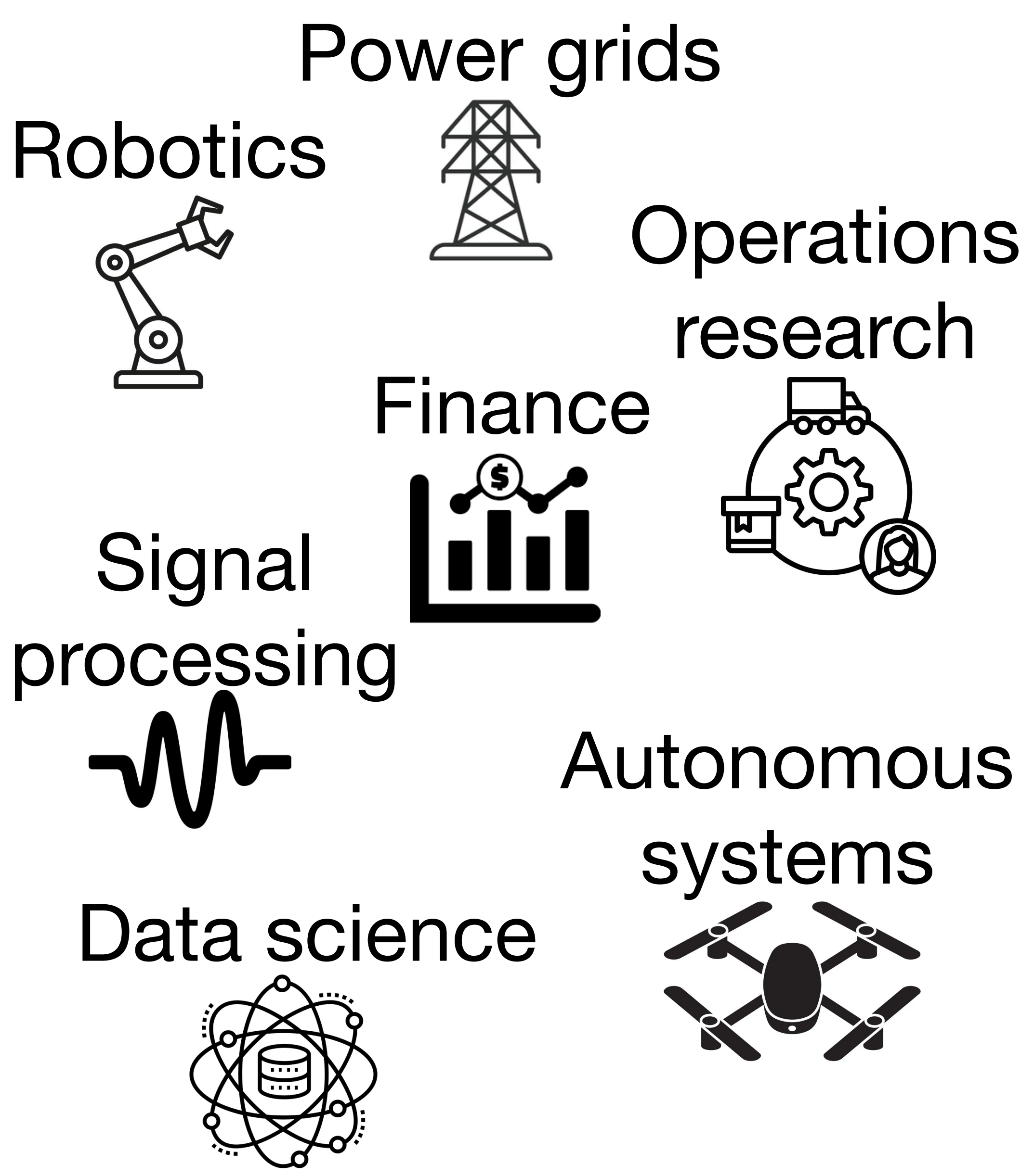
$\longrightarrow$ optimal solution $z^\star(x)$

Research areas



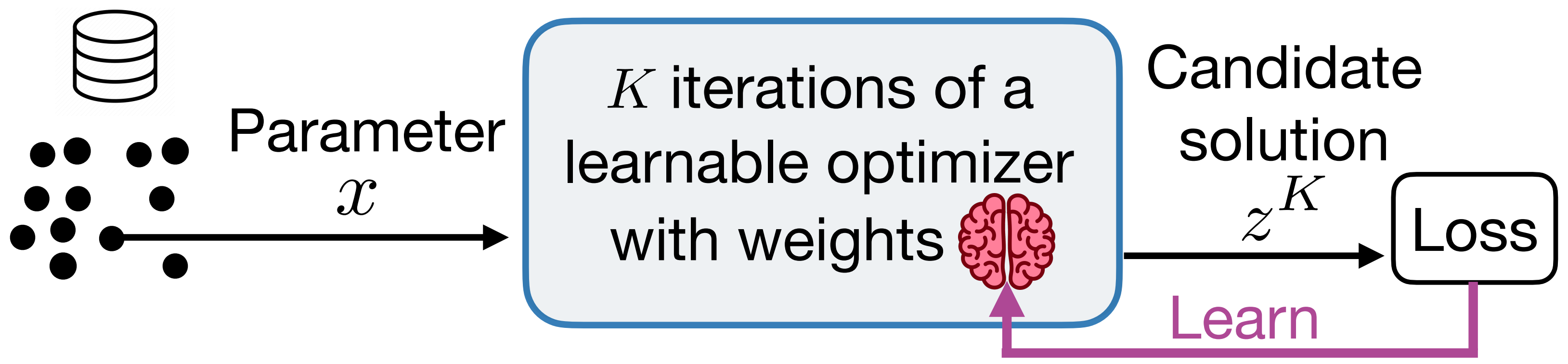
$\longleftrightarrow$ how can AI and data enable real-time optimization that is reliable? $\longleftrightarrow$

Applications



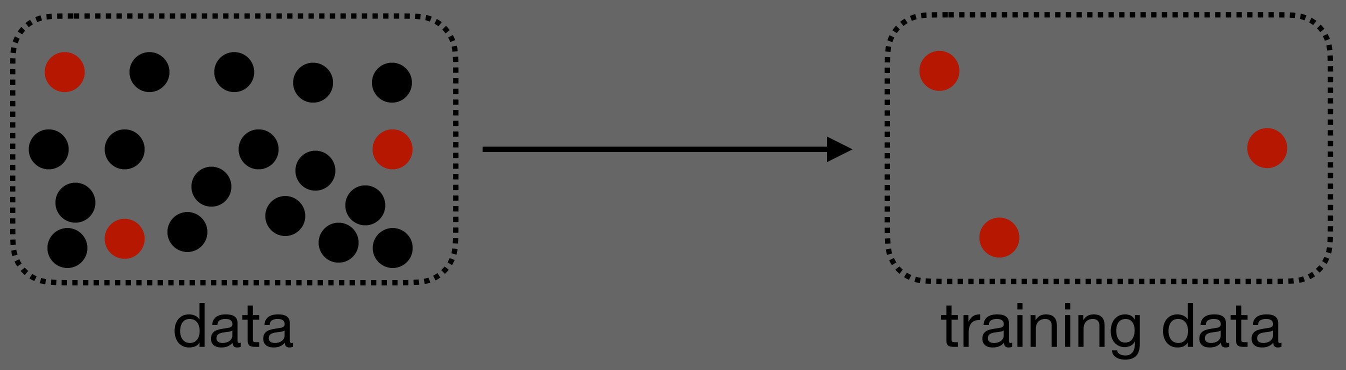
Overview of talk: AI for Optimization

a brief tutorial on learning to optimize



learning algorithm hyperparameters

can we learn powerful optimizers with scarce training data?



Learning Algorithm Hyperparameters for Fast Parametric Convex Optimization

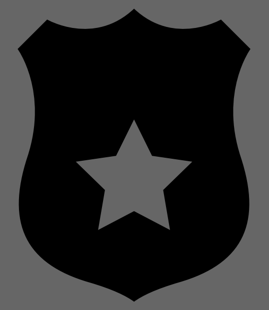
R. Sambharya, B. Stellato

SIAM Journal on the Mathematics of Data Science, 2026

https://github.com/stellatogrp/learning_algorithm_hyperparameters

learning to optimize with guarantees

can we learn powerful optimizers with finite-iteration worst-case guarantees?



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

https://github.com/rajivsambharya/learn_accel_steps_robust

First-order methods are widely popular again…

first-order methods use
only (sub)-gradient information

fixed-point iterations

$$\underbrace{z^{k+1}}_{\text{iterates}}(x) = \underbrace{T}_{\text{algorithm update}}(z^k(x), \underbrace{x}_{\text{parameter}})$$

example: projected gradient descent

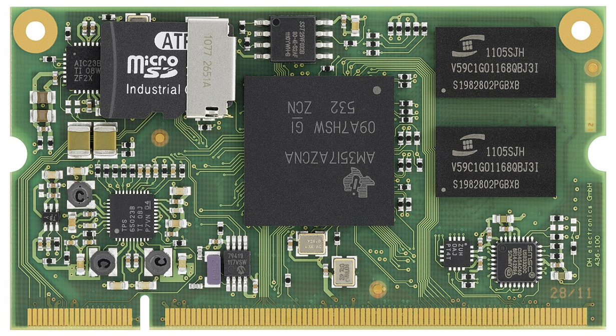
$$\begin{array}{ll} \text{minimize} & f(z,x) \\ \text{subject to} & z \in \Omega(x) \end{array}$$

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

benefits of first-order methods

cheap iterations ✔

embedded optimization



large-scale optimization

many general-purpose convex 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 time to run 20 fixed-point steps)

recall quadcopter: only a **few milliseconds** to solve each problem

Can machine learning speed up convex parametric optimization?

goal: do mapping quickly and accurately

parameter $x \longrightarrow$

$$\begin{array}{ll} \text{minimize} & f(z,x) \\ \text{subject to} & z \in \Omega(x) \end{array}$$

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

$x \longrightarrow$ only optimization $\longrightarrow \hat{z}^{\text{Opt}}(x)$

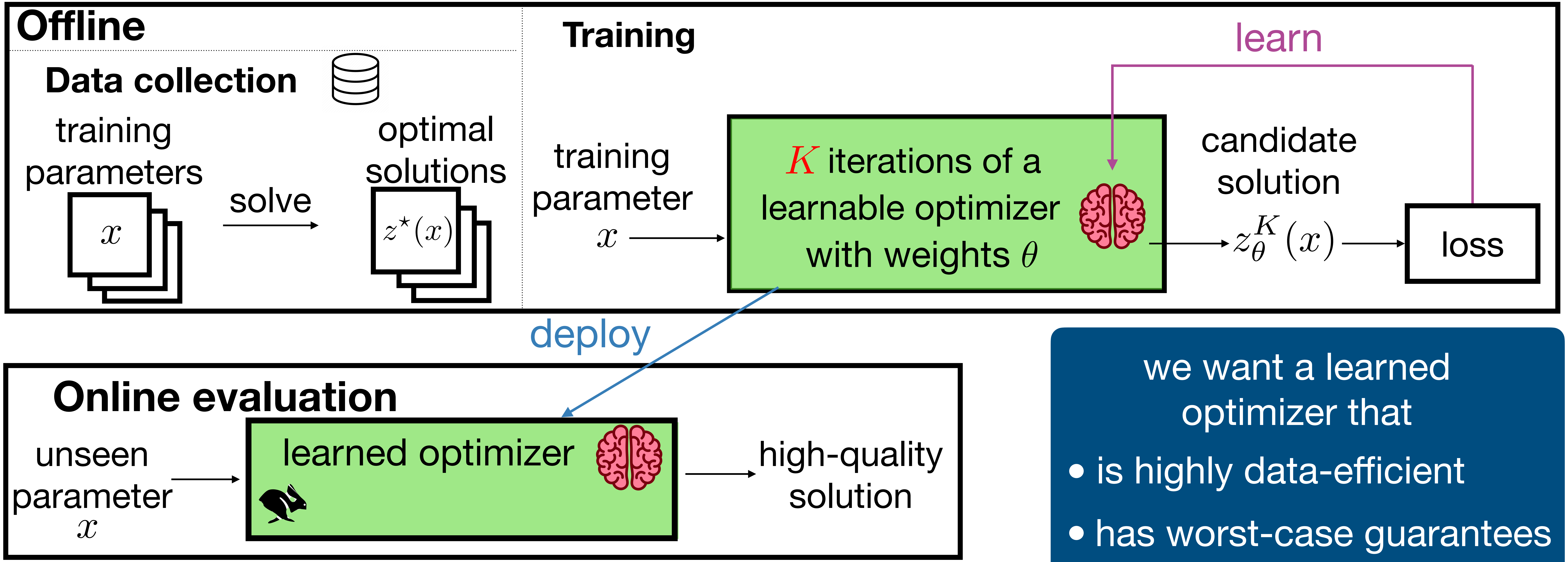
$x \longrightarrow$ only machine learning $\longrightarrow \hat{z}^{\text{ML}}(x)$

$x \longrightarrow$ optimization 🤝 machine learning $\longrightarrow \hat{z}^{\text{Opt+ML}}(x)$

The learning to optimize paradigm

Goal: solve the problem within a **budget** of K iterations

$$\begin{array}{ll} \text{minimize} & f(z, x) \\ \text{subject to} & z \in \Omega(x) \end{array}$$

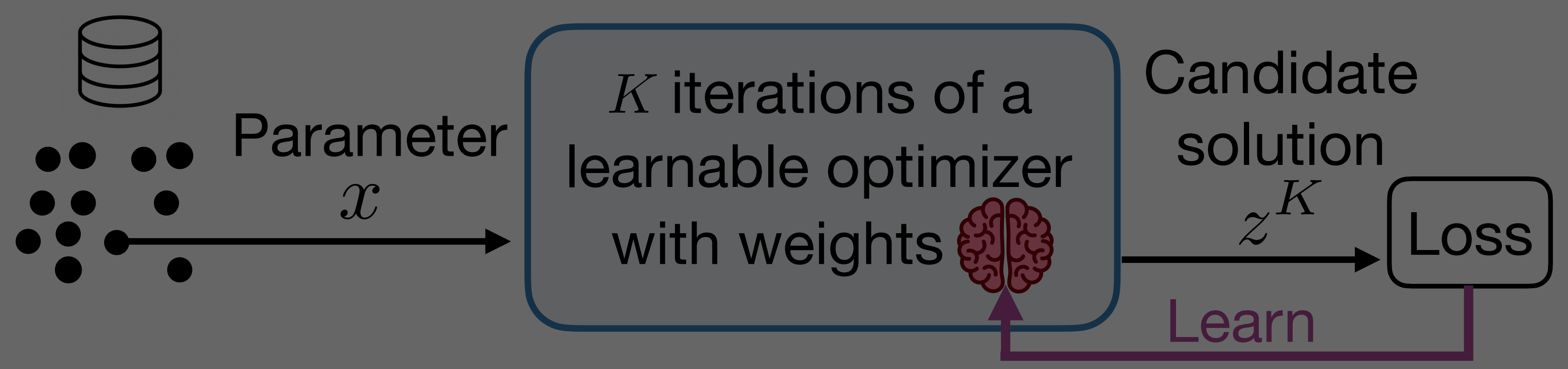


we want a learned optimizer that

- is highly data-efficient
- has worst-case guarantees

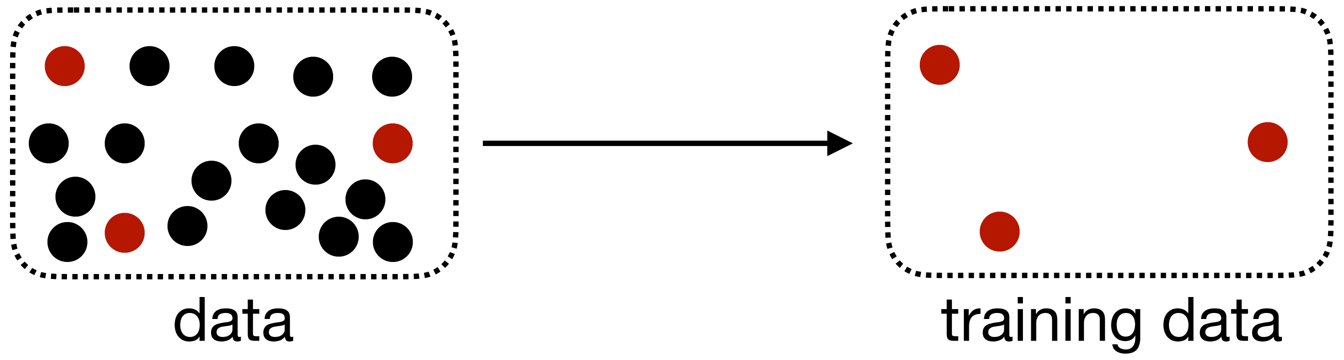
Overview of talk: AI for Optimization

a brief tutorial on learning to optimize



learning algorithm hyperparameters

can we learn powerful optimizers with **scarce training data?**



Learning Algorithm Hyperparameters for Fast Parametric Convex Optimization
R. Sambharya, B. Stellato
SIAM Journal on the Mathematics of Data Science, 2026
https://github.com/stellatogrp/learning_algorithm_hyperparameters

learning to optimize with guarantees

can we learn powerful optimizers with **finite-iteration worst-case guarantees?**

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
https://github.com/rajivsambharya/learn_accel_steps_robust

Research question

parameter size

$x \in \mathbf{R}^d \longrightarrow$

$$\begin{array}{ll} \text{minimize} & f(z, x) \\ \text{subject to} & z \in \Omega(x) \end{array}$$

$\longrightarrow z^\star(x) \in \mathbf{R}^n$

decision variable size

training parameters

x

size of training dataset

$\{x_i\}_{i=1}^N$

low-data regime is common

$N << d$ and / or $N << n$

- data can be expensive (*e.g.*, robotics)
- often have high-dimensional problems

(problems in control, power grids, data science, can have 10,000+ variables)

can we learn powerful optimizers with scarce training data? → just $N = 10$ samples!

Other methods: learning to optimize in the convex setting

our question: can we learn powerful optimizers with scarce training data?

learning warm starts

learn a high-quality initial point from data

Baker 2019; Finn et al. 2017; Chen et al. 2018; Chen et al. 2022; Sharony et al. 2024; **Sambharya et al. 2024**

learning algorithm steps

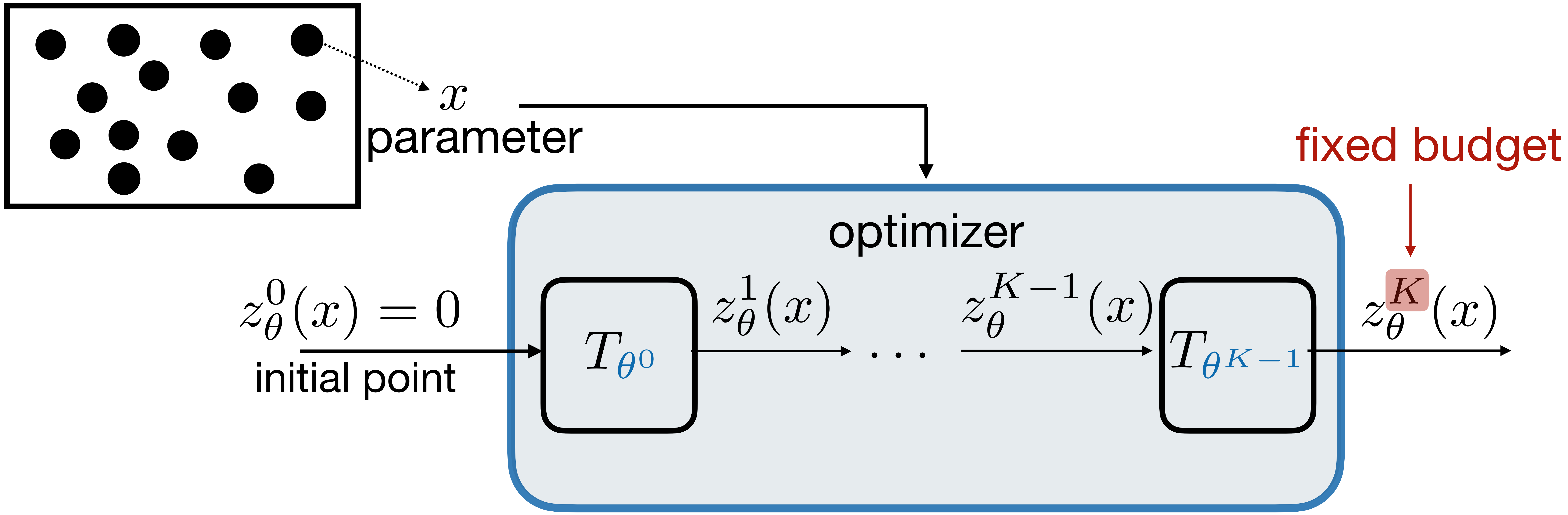
learn the update function from data

Gregor and LeCun 2010; Ichnowski 2021; Venkataraman and Amos 2021; Banert et al. 2024; Saravanos et al. 2024; King et al. 2024

limitation: these methods are data-hungry since they require learning many weights

can we develop a learned optimizer with a very low number of weights?

First-order methods as fixed-length computational graphs



algorithm hyperparameters (*e.g.*, step sizes)

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

example: projected gradient descent

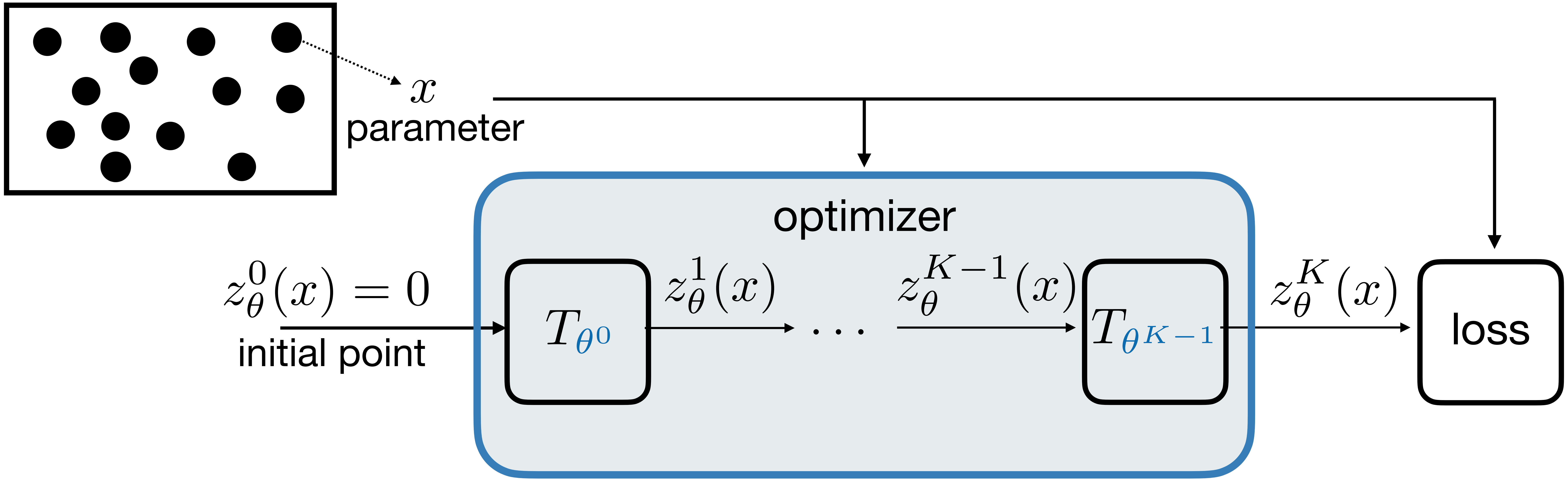
$$z_\theta^{k+1}(x) = \Pi_{\Omega(x)}(z_\theta^k(x) - \theta^k \nabla_z f(z_\theta^k(x), x))$$

- conventional wisdom: use a constant step size
- recent advances: vary the step size!

Altschuler and Parrilo 2023, Grimmer 2023, Bok and Altschuler 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

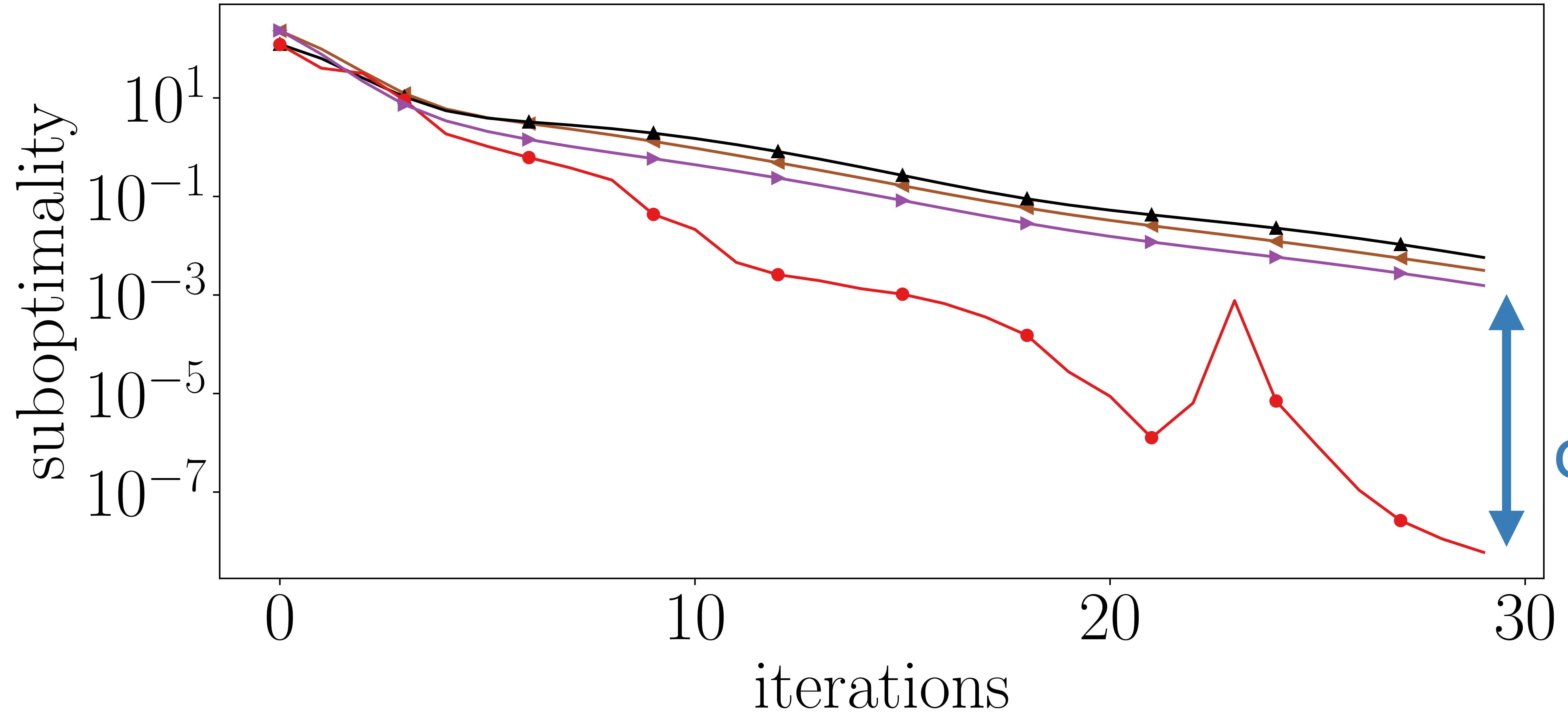
- shared across problem instances
- differ across iterations
- low number of weights

Learning step sizes for non-negative least squares

parameter
$x \in \mathbf{R}^{700}$

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

solution
$z^\star(x) \in \mathbf{R}^{500}$



5 orders
of magnitude

Nesterov

Nearest neighbor
warm start

Learned
warm starts

Sambharya et al. 2024

10000 training instances

Learned
step sizes

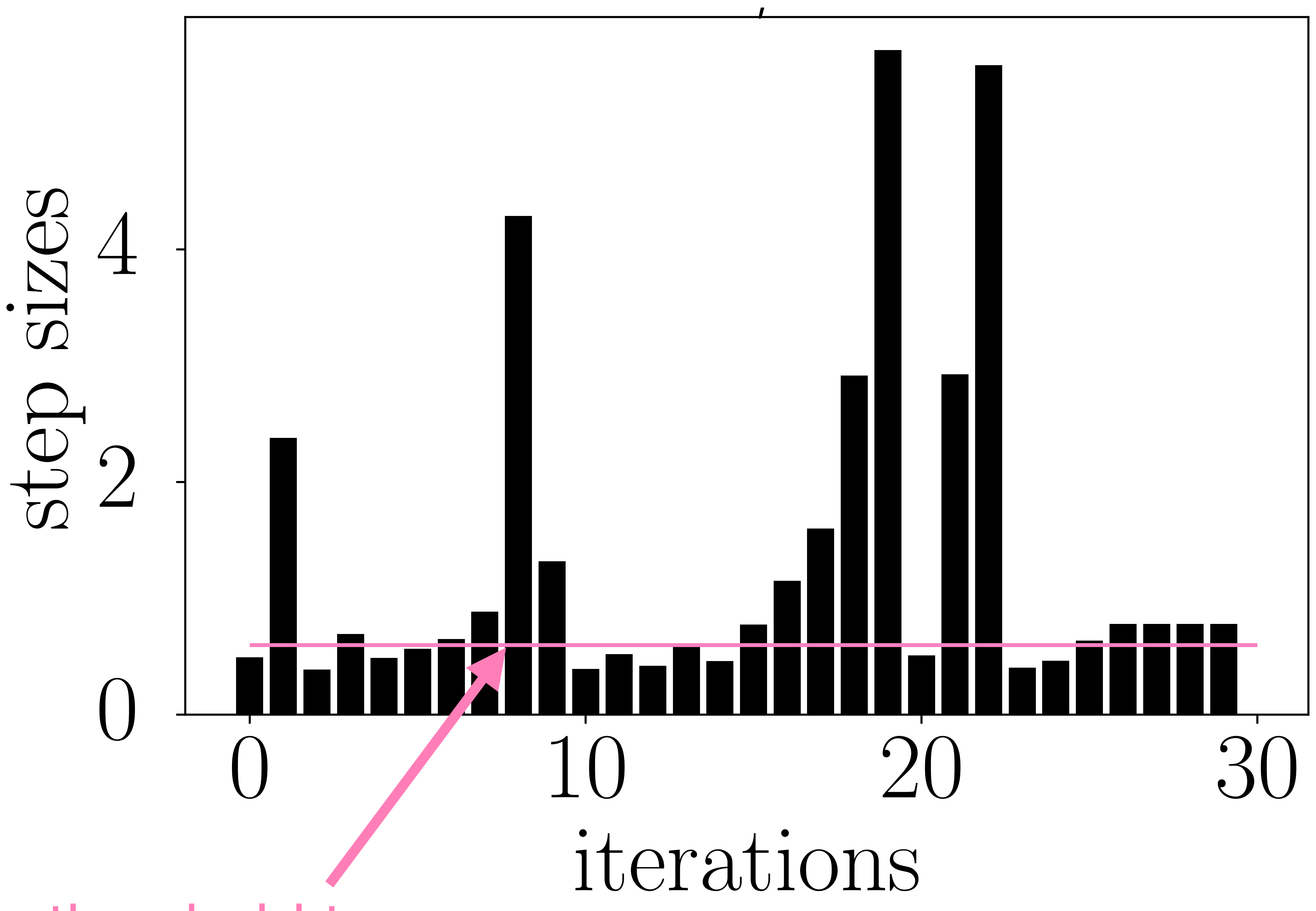
10 training instances

learning step sizes
can be powerful

this is a highly
data-efficient approach

multi-step descent
phenomenon

We learn a spiky step size schedule!



threshold to guarantee convergence for constant step size

similar to recent results on spiking (silver) step sizes

Altschuler, Parrilo, Grimmer

we exploit the parametric structure to learn aggressive step sizes

- focus: family of parameters
- similar opt landscapes
- same initial point

our learned optimizer is highly interpretable

→ no neural networks!

We can also learn momentum hyperparameters

composite convex optimization

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

convex, smooth   convex, non-smooth

e.g., lasso: $\text{minimize} \quad (1/2)\|Az - x\|_2^2 + \lambda\|z\|_1$

proximal operator

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

Nesterov's acceleration

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

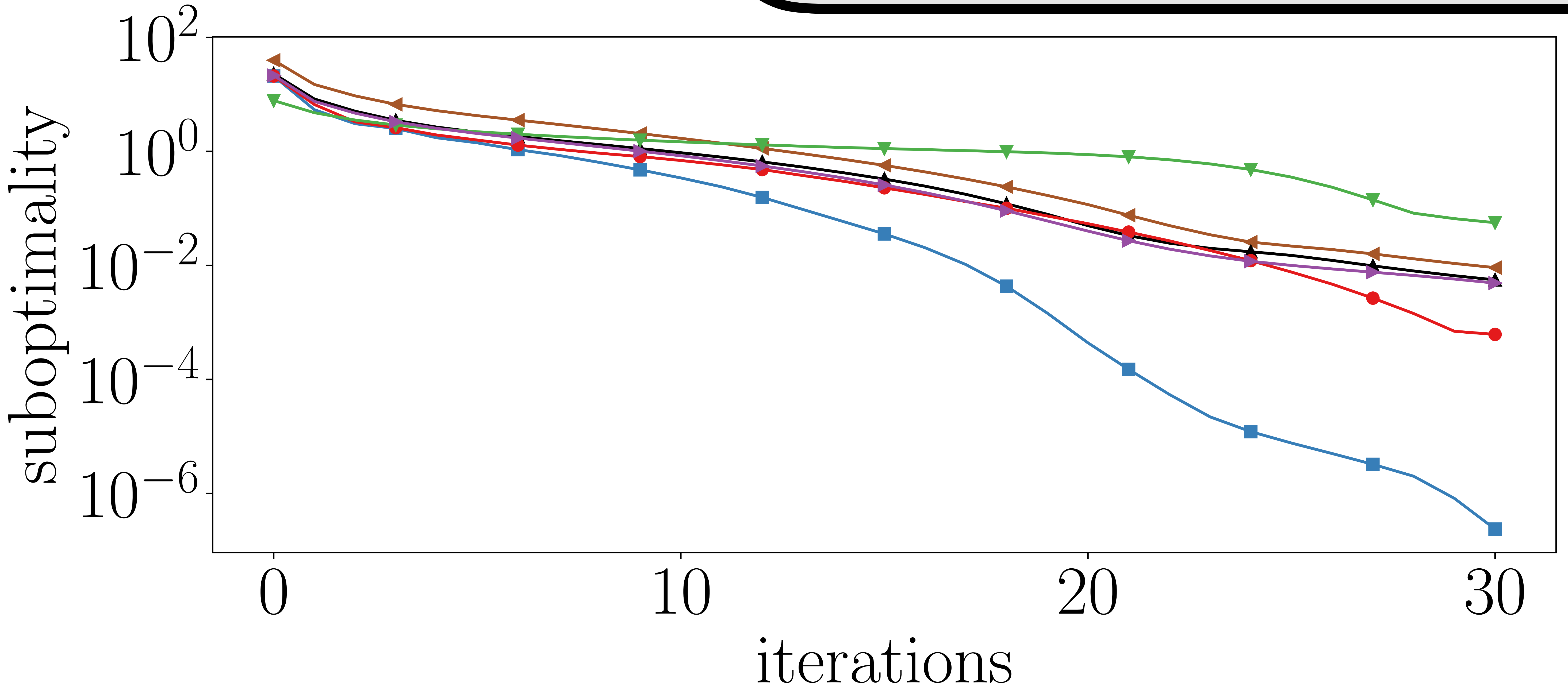
$$z^{k+1}(x) = y^{k+1}(x) + \beta^k (y^{k+1}(x) - y^k(x))$$

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

Nesterov 83, Beck and Teboulle 09

Example: learned hyperparameters for sparse coding

noisy measurement $x \in \mathbf{R}^{1000}$ → **signal reconstruction** minimize $(1/2)\|Az - x\|_2^2 + \lambda\|z\|_1$ → reconstructed signal $z^\star(x) \in \mathbf{R}^{2000}$



learning momentum hyperparameters yields significant benefits

Nesterov

Nearest neighbor warm start

Learned warm starts

Sambharya et al. 2024

LISTA

Gregor and LeCun 2010

10000 training instances

Learned step sizes

Learned step and momentum sizes

10 training instances

Learning hyperparameters for general convex optimization

convex optimization

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

convex cone

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

projected gradient descent not well-suited

- no closed-form projection
- instead, the alternating direction method of multipliers (ADMM) is well-suited

Douglas and Rachford 1956, Gabay and Mercier 1976, Boyd et al 2011

accelerated ADMM

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

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

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

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

hyperparameters

- time-varying (α^k, β^k)
- time-invariant (σ, ρ)

(computational advantages in time-invariance)

with minor changes, we learn-to-optimize more general convex problems

Model predictive control of a quadcopter

$x \in \mathbf{R}^{44}$

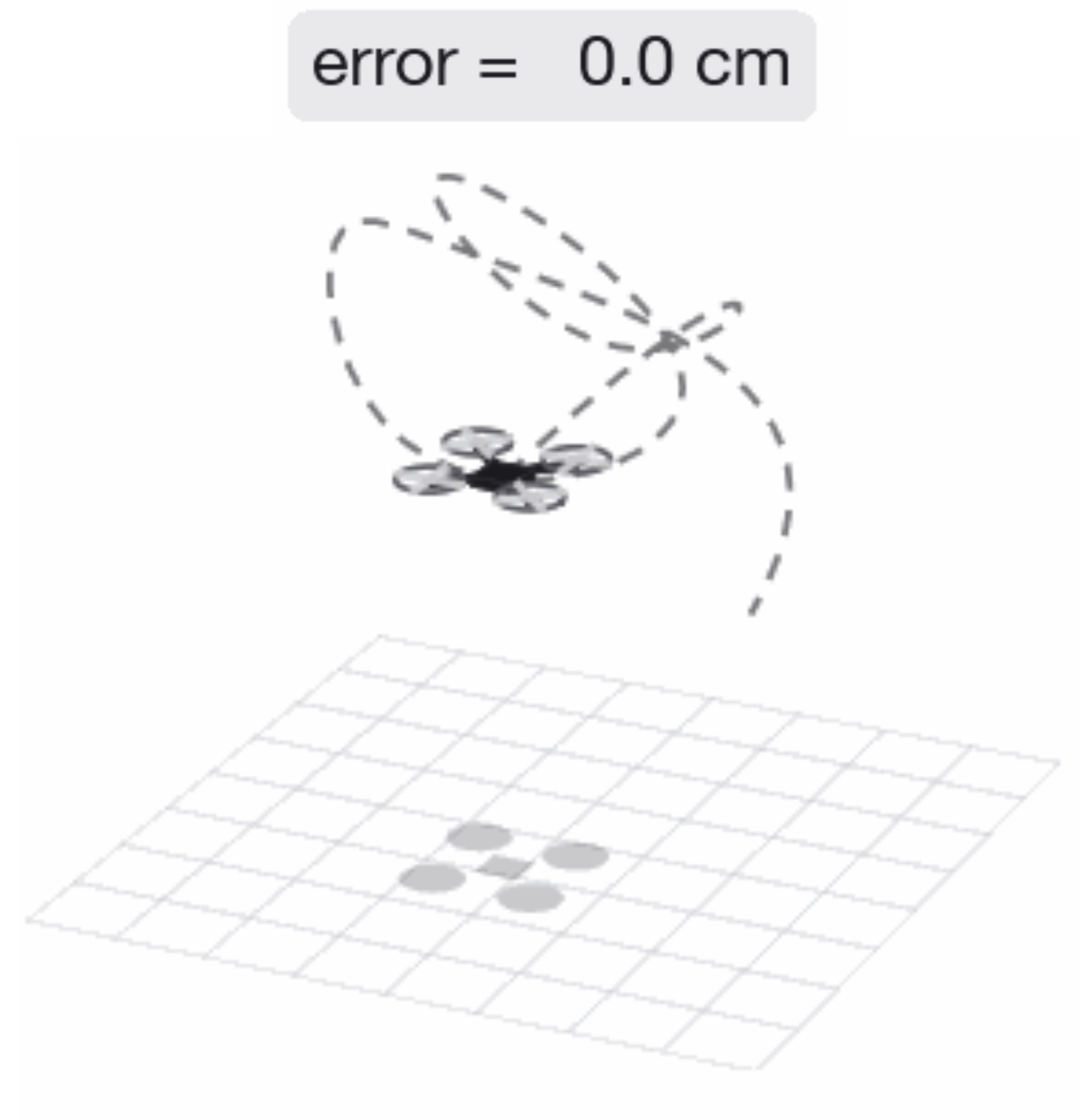
current state,
reference trajectory

quadratic program

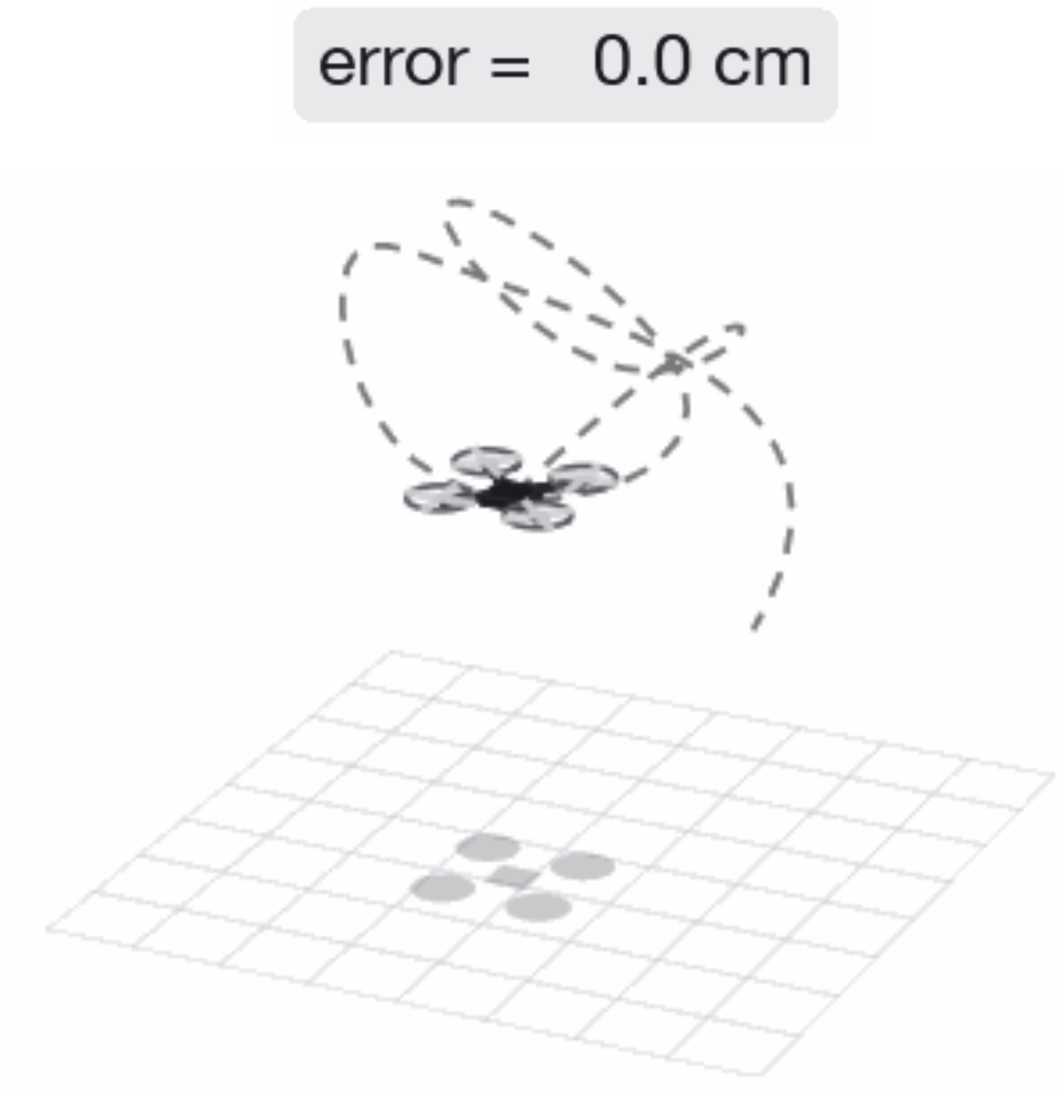
$$
\begin{array}{ll}
\text{minimize} & \sum_{t=1}^{T} \|s_t - s_t^{\text{ref}}\|_2^2 \\
\text{subject to} & s_{t+1} = As_t + Bu_t \\
& s_t \in \mathcal{S}, \quad u_t \in \mathcal{U} \\
& s_0 = s_{\text{init}}
\end{array}
$$

$z^\star(x) \in \mathbf{R}^{1750}$

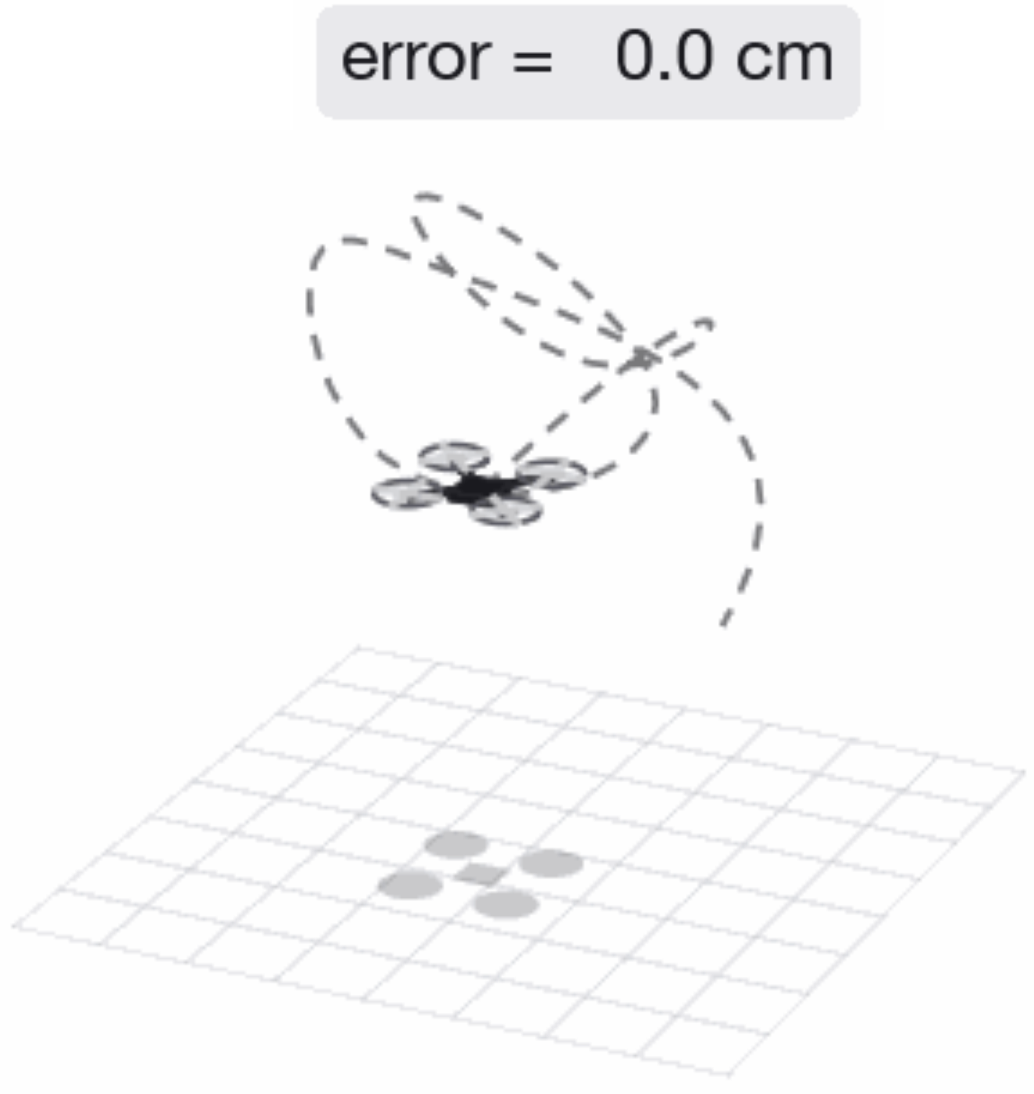
actions



nearest neighbor
80 iterations



previous solution
80 iterations



learned
20 iterations

time = 0.00 s

with learning, we can track the trajectory well

Summary: learning algorithm hyperparameters

question: can we learn powerful optimizers with scarce training data?

approach: learn a shared sequence of hyperparameters

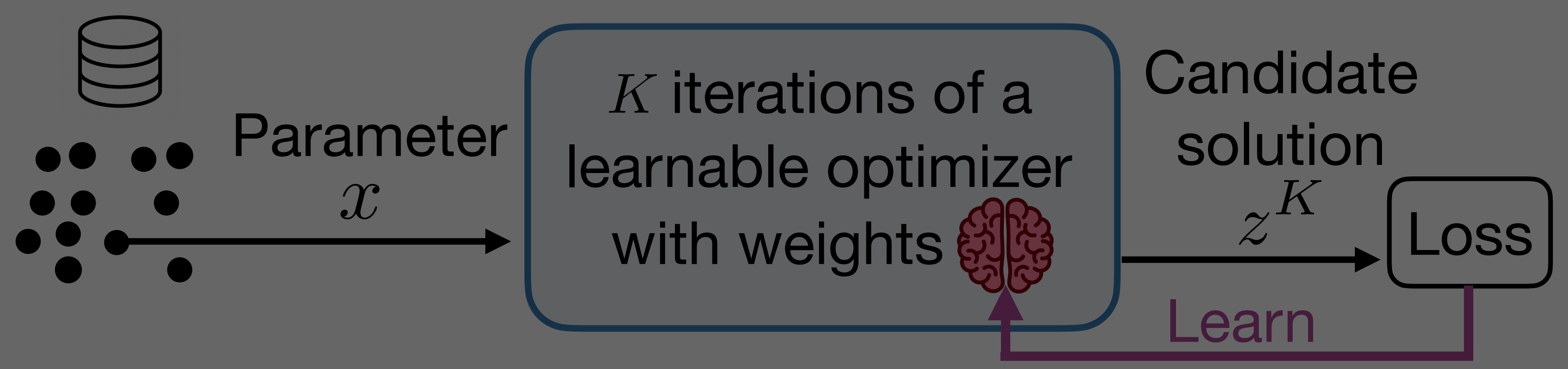
takeaways:

- solution quality improved by 5+ orders of magnitude
- only 10 training instances used, when problem dimensions are much larger
- learn spiky step size schedules (progressive training and closed-form solutions)

Learning Algorithm Hyperparameters for Fast Parametric Convex Optimization
R. Sambharya, B. Stellato
SIAM Journal on the Mathematics of Data Science, 2026
https://arxiv.org/pdf/2411.15717
https://github.com/stellatogrp/learning_algorithm_hyperparameters

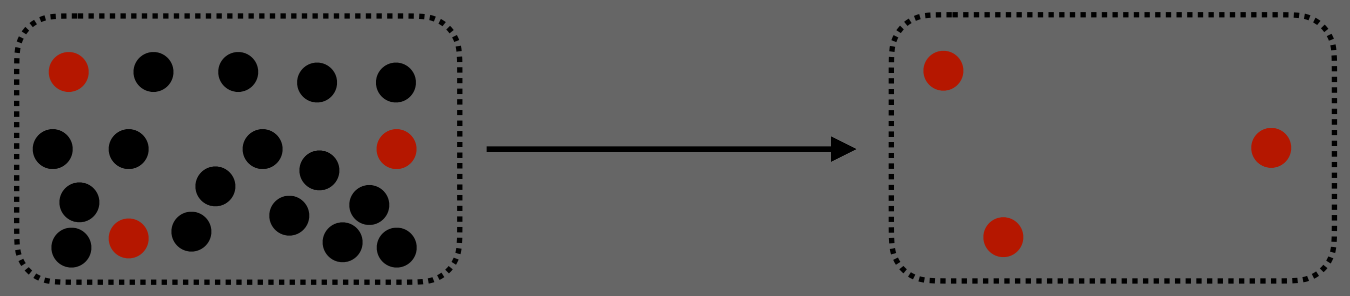
Overview of talk: AI for Optimization

a brief tutorial on learning to optimize



learning algorithm hyperparameters

can we learn powerful optimizers with **scarce training data?**



Learning Algorithm Hyperparameters for Fast Parametric Convex Optimization

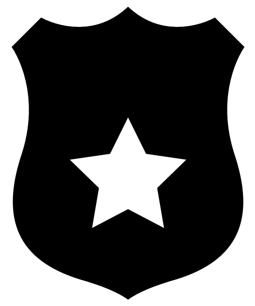
R. Sambharya, B. Stellato

SIAM Journal on the Mathematics of Data Science, 2026

https://github.com/stellatogrp/learning_algorithm_hyperparameters

learning to optimize with guarantees

can we learn powerful optimizers with **finite-iteration worst-case guarantees?**



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

https://github.com/rajivsambharya/learn_accel_steps_robust

Research question

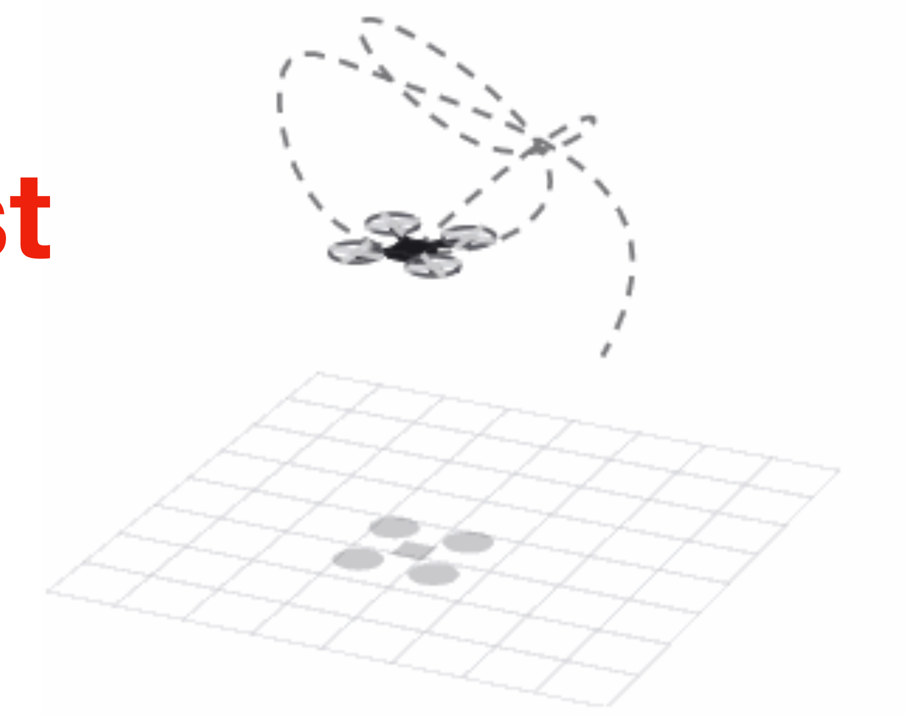
(no learning component)

worst-case guarantees are a hallmark of classical algorithms for convex optimization

learned optimizers
excel in practice…

…but **lack worst-case guarantees**
within a budget of iterations!

guarantees are a must
if we want to deploy
in the real world!



can we learn powerful optimizers that have
finite-iteration worst-case guarantees?

Other types of guarantees in learning to optimize

our question: can we learn powerful optimizers that have finite-iteration worst-case guarantees?

convergence guarantees

asymptotic convergence as iterations $\to \infty$

Fahy, Golbabaee, Ehrhardt 2024; Heaton et al. 2023; Banert et al. 2024; Martin and Furieri 2024; **Sambharya et al. 2024**

✗ limitation: no guarantee on finite-iteration performance

generalization guarantees

under distributional model $x \sim \mathcal{D}$, we get finite-iteration probabilistic guarantees

Sucker, Fadlii, Ochs 2025; Balcan et al. 2021; Saravanos et al. 2024; **Sambharya and Stellato 2025**

✗ limitation: relies on i.i.d. assumption, often unrealistic (e.g., sequential settings); no out-of-distribution guarantees

Preview of the approach

our question: can we learn powerful optimizers that have finite-iteration worst-case guarantees?

approach:

starting point

learning algorithm hyperparameters

(previous part of the talk)

evaluation

evaluate worst-case guarantees

→ use performance estimation problems (PEP)

Drori, Teboulle, Hendrickx, Glineur, Taylor

training

regularize training for robustness

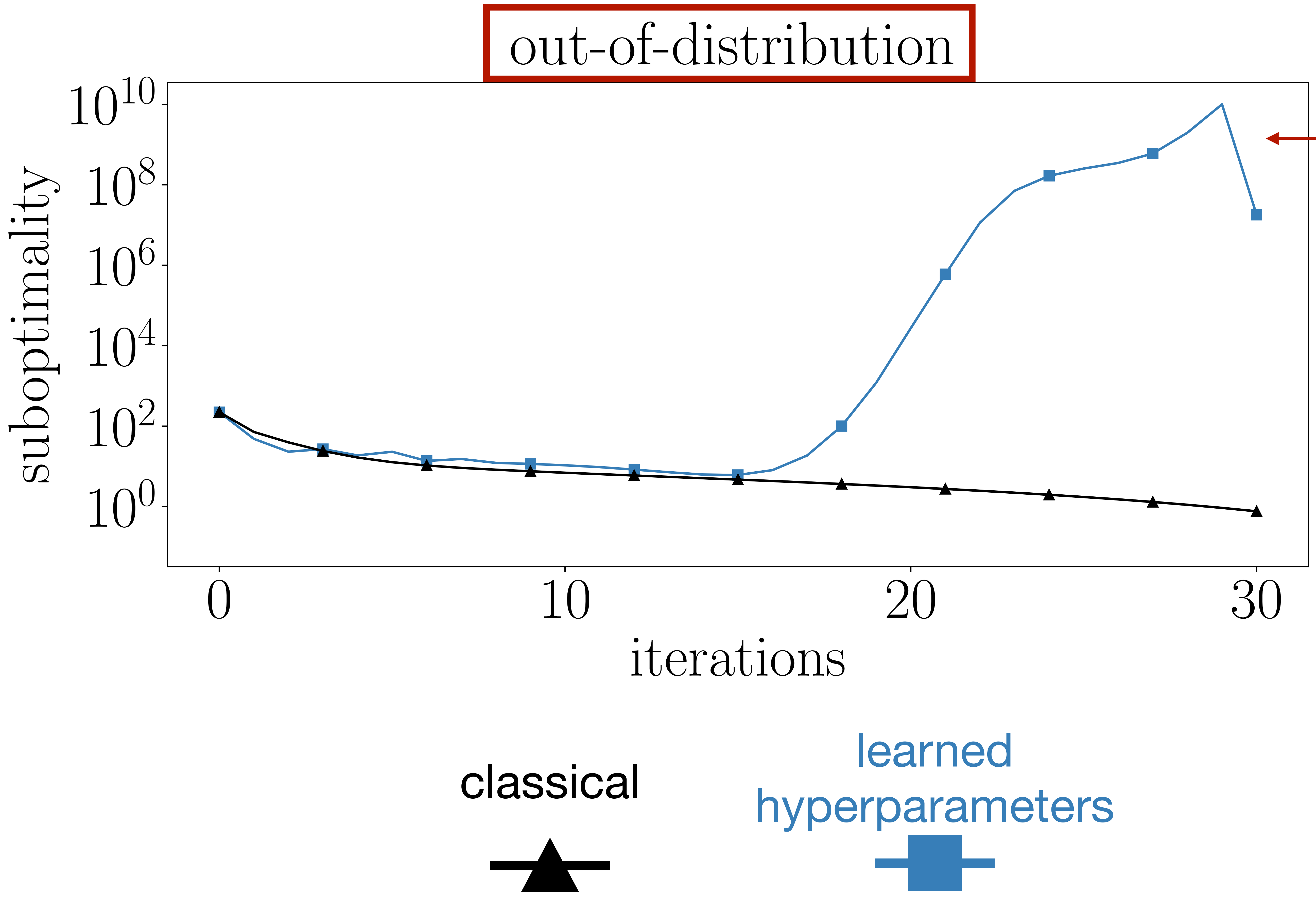
What can go wrong? Learned optimizers lack worst-case guarantees

noisy measurement $x \in \mathbf{R}^{1000}$ $\longrightarrow$

signal reconstruction

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

$\longrightarrow$ reconstructed signal $z^\star(x) \in \mathbf{R}^{2000}$



failures on out-of-distribution instances

can we learn hyperparameters that are robust?

Can we learn hyperparameters θ that are robust?

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

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

performance metric

level of robustness

a provided set of all possible parameters

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

an acceptable target robustness level

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

for every $x \in \mathcal{X}$, $f(\cdot, x)$ is μ-strongly convex and 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

PEP main idea: given a function class and algorithm, PEP computes the worst-case finite-iteration performance

hyperparameters θ → **semidefinite program**
- dimension-independent
- exact guarantee

→ level of robustness $\gamma(\theta)$

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

✔ solve an SDP to evaluate $\gamma(\theta)$

🤔 but how can we train for robustness?

Robust training of hyperparameters

penalty formulation

PEP-regularized training problem

$$
\begin{array}{lll}
\text{minimize} & \text{(objective)} & (1/N)\sum_{i=1}^{N} \|z_\theta^K(x_i) - z^\star(x)\|_2^2 + \boxed{\lambda((\gamma(\theta) - \gamma^{\text{target}})_+)^2} \\
\text{subject to} & \text{(algorithm steps)} & z_\theta^{k+1}(x_i) = T(z_\theta^k(x_i), x_i) \quad k = 0, \dots, K-1, \forall i \\
& \text{(initial point)} & z_\theta^0(x_i) = 0
\end{array}
$$

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

CVXpy

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

PyTorch

TensorFlow

Amos et. al 2017, Agrawal et. al 2019

we use gradient-based methods to solve the PEP-regularized training problem

approximately satisfy robustness constraint $\gamma(\theta) \leq \gamma^{\text{target}}$

Learning robust hyperparameters for sparse coding

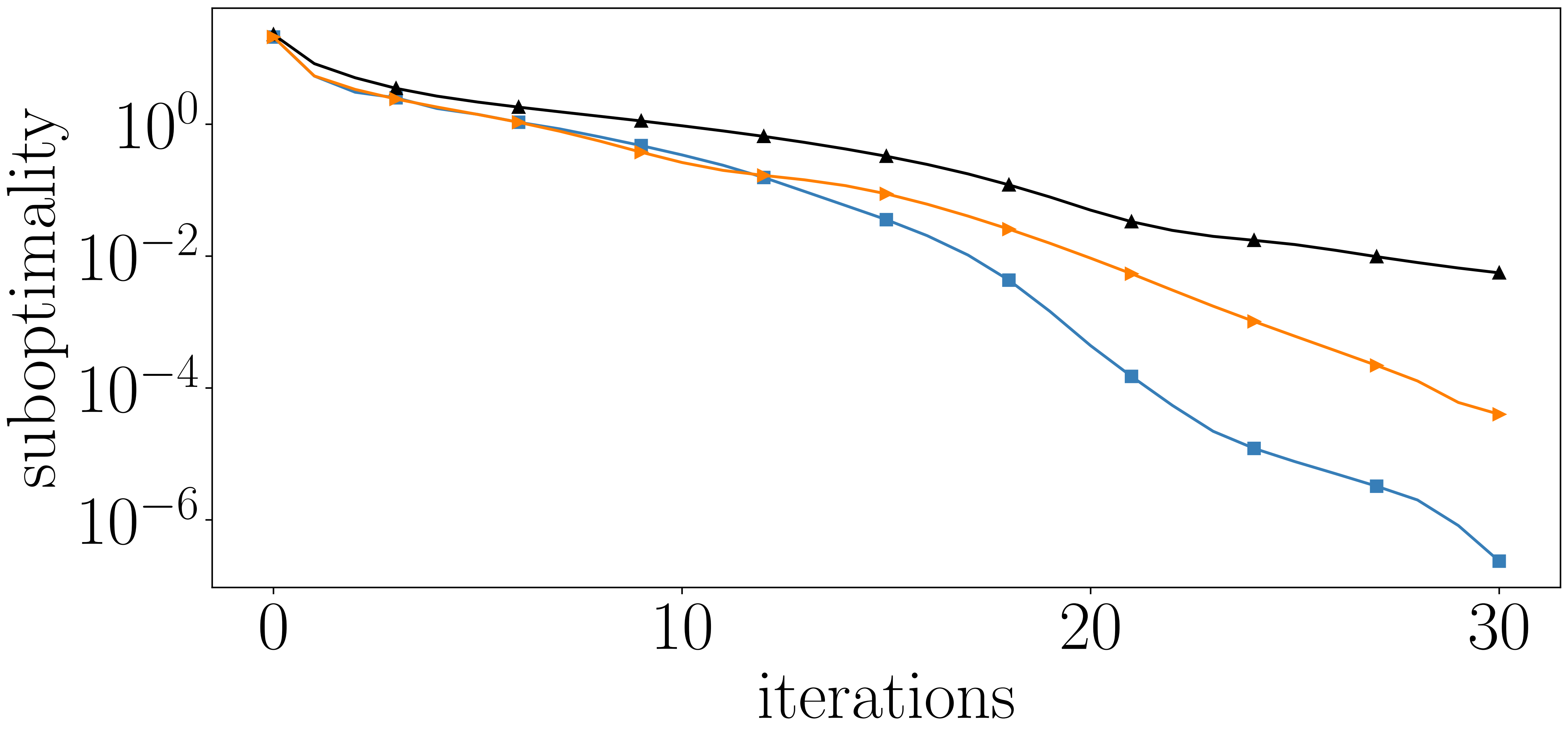
noisy measurement
$x \in \mathbf{R}^{1000}$ $\longrightarrow$

signal reconstruction

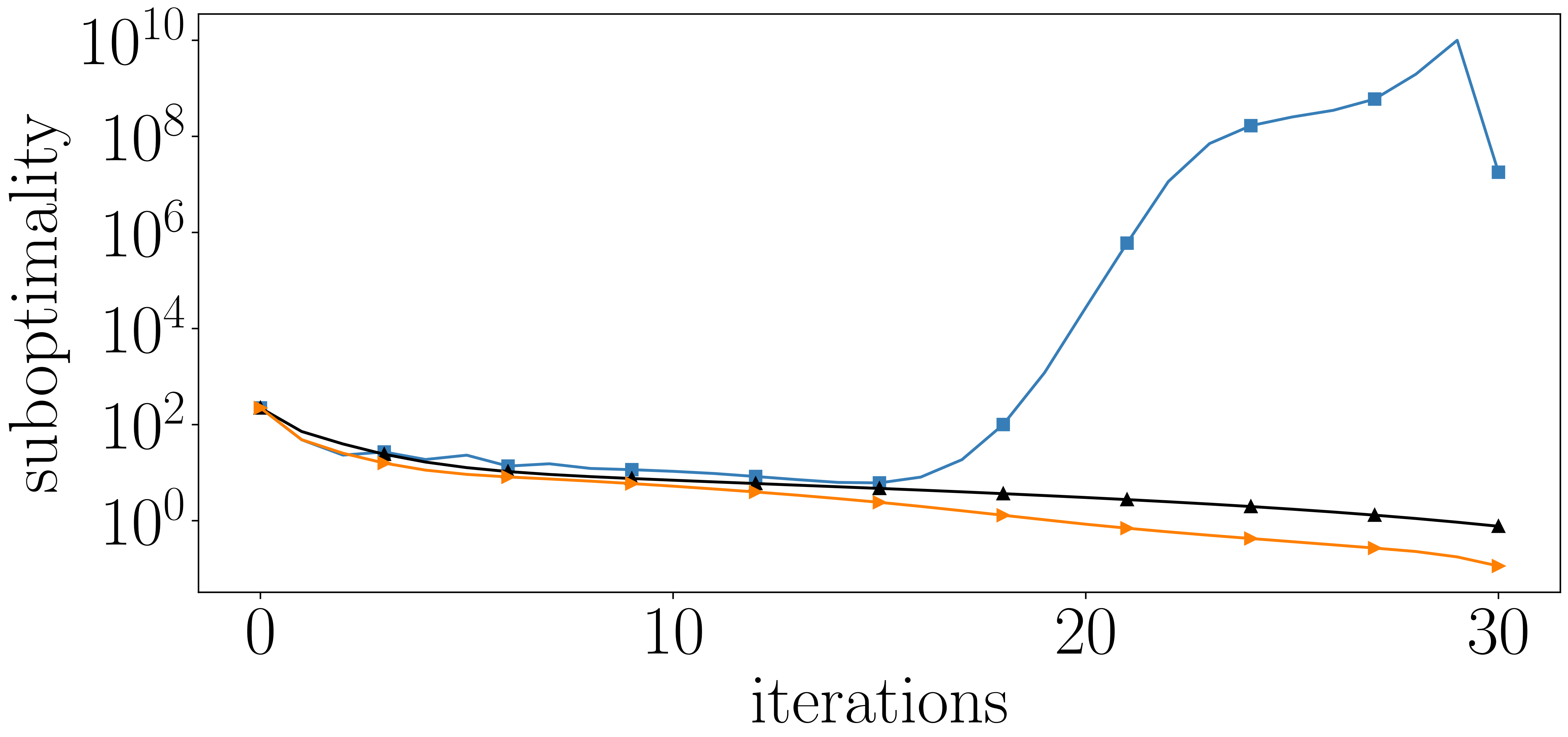
$$\text{minimize} \quad (1/2)\|Az - x\|_2^2 + \lambda\|z\|_1$$

$\longrightarrow$ reconstructed signal
$z^\star(x) \in \mathbf{R}^{2000}$

in-distribution



out-of-distribution

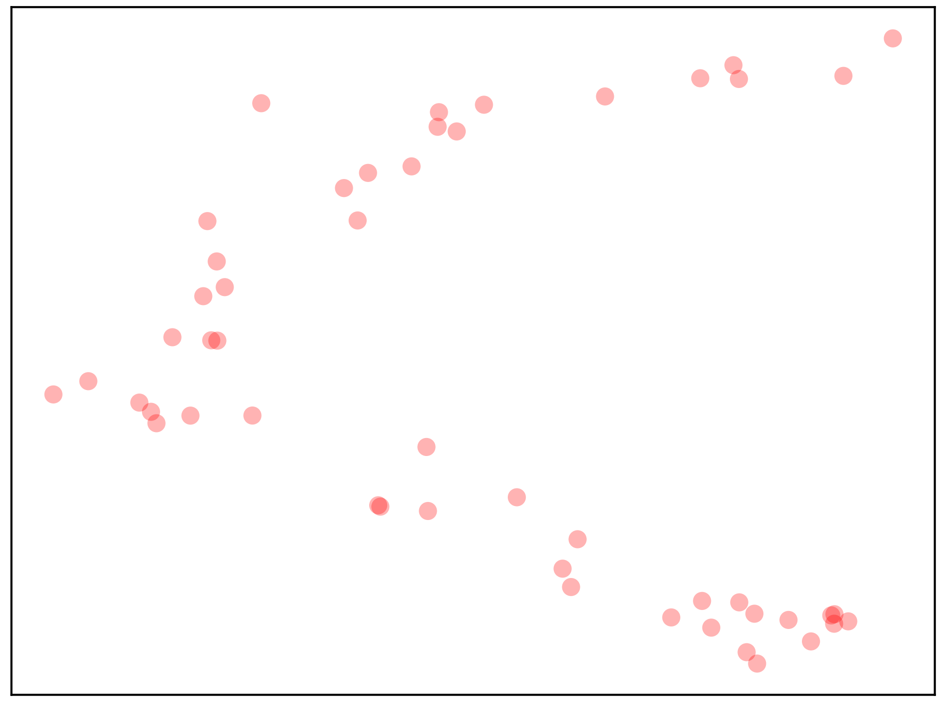


Nesterov	Learned: robust	Learned: not robust
$\gamma = 0.01$	$\gamma = 0.10$	$\gamma = \infty$
(most robust)		(least robust)

tradeoff between performance and robustness

training with robustness protects in out-of-distribution cases

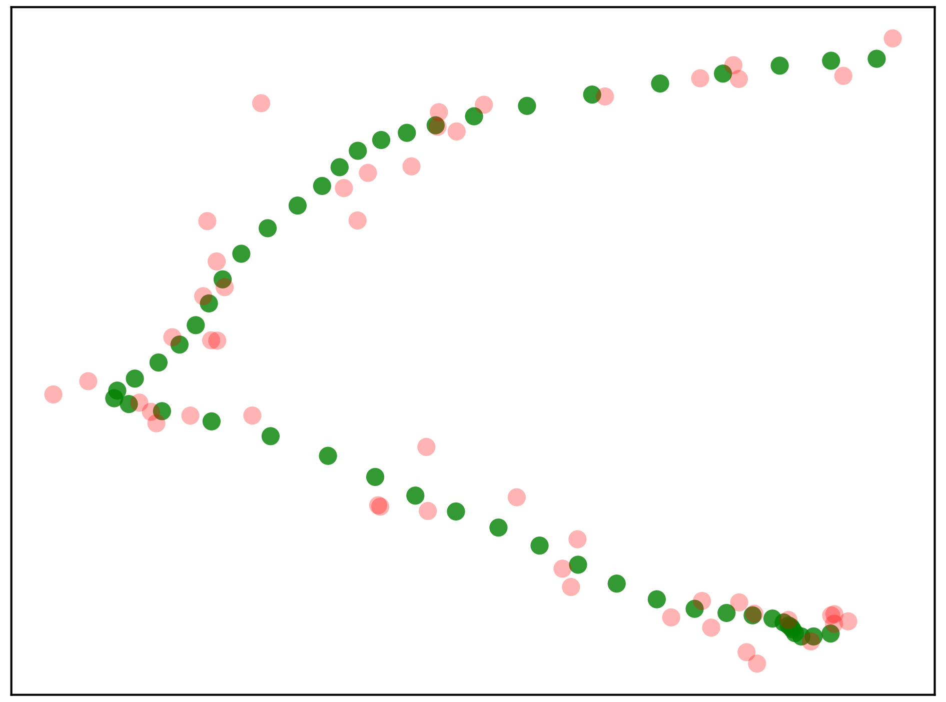
Optimization-based Kalman filtering



second-order cone program

Huber loss

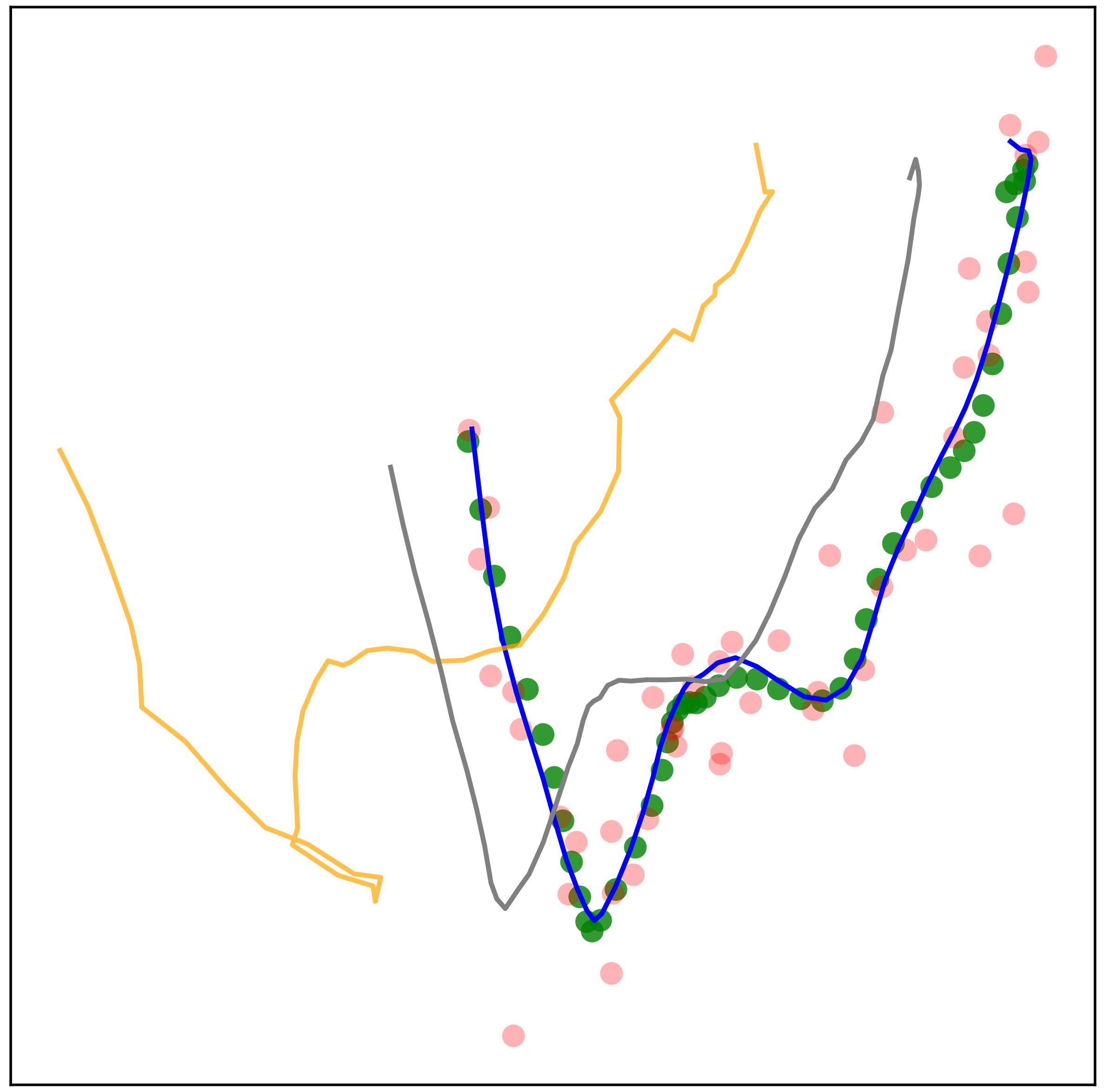
$$\begin{array}{ll} \text{minimize} & \sum_{t=0}^{T-1} \|w_t\|_2^2 + \boxed{\psi_\rho}(v_t) \\ \text{subject to} & s_{t+1} = As_t + Bu_t \\ & y_t = Cs_t + v_t \end{array}$$



noisy trajectory

optimal solution

out-of-distribution



5 iterations

no learning

learned

learned + robust

learning w/ robustness performs best out-of-distribution

Summary: learning to optimize with finite-iteration worst-case guarantees

question: can we learn powerful optimizers with finite-iteration worst-case guarantees?

approach:

- adapt the learning algorithm hyperparameters approach
- use PEP and differentiable optimization for regularized training

takeaways:

- finite-iteration worst-case guarantees obtained
- dramatic performance gains in nominal settings and more robust out-of-distribution

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

https://github.com/rajivsambharya/learn_accel_steps_robust

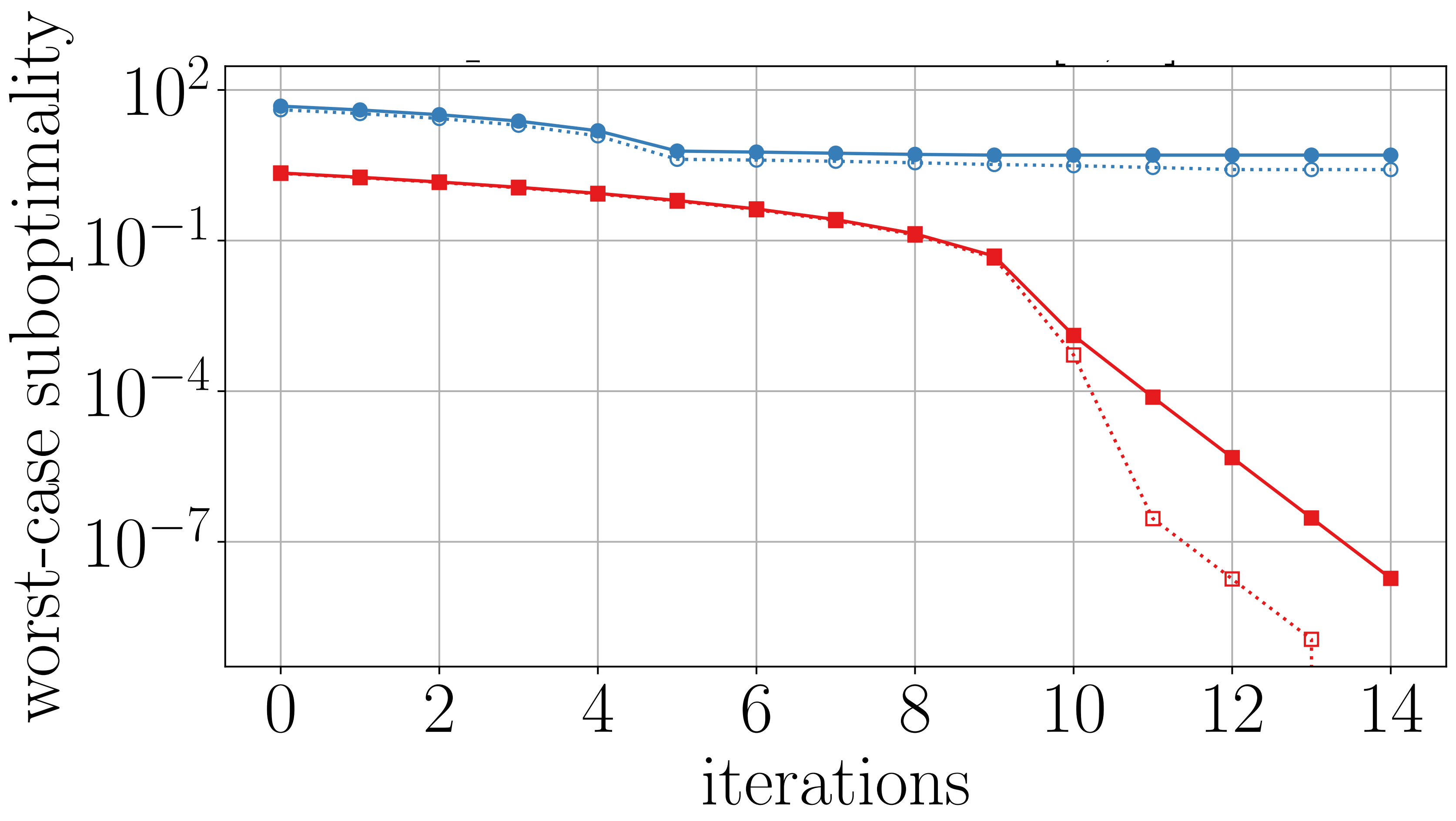
We can also verify optimization algorithms in parametric optimization

we can use optimization to **verify** the worst-case performance of algorithms!

- for sequential convex programming: where guarantees don't exist
  - we obtain exact worst-case guarantees

$$
\begin{array}{lll}
\text{maximize} & \text{(suboptimality)} & f(z^K, x) - f(z^\star, x) \\
\text{subject to} & \text{(algorithm steps)} & z^{k+1} \in T(z^k, x) \\
 & \text{(parameter)} & x \in \mathcal{X} \\
 & \text{(initial point)} & z^0 = 0 \\
 & \text{(optimality)} & z^\star \in \arg\min_z \; f(z, x) \\
 & & \qquad \text{s.t.} \quad z \in \Omega(x)
\end{array}
$$

box-constrained non-convex quadratic



	warm-start	cold-start
verified result		
sample maximum		

can certify global optimality (or not)!

initial point matters

Verification of Sequential Convex Programming for Parametric Non-convex Optimization

R. Sambharya, N. Matni, G. Pappas

https://arxiv.org/pdf/2511.10622

Conclusion: AI for Optimization

- real-world optimization is **parametric**
- we can **learn powerful optimizers** that
  - are highly **data-efficient**
  - have **finite-iteration worst-case guarantees**
- we should **rethink optimization algorithms**

traditional view

- one-size fits all
- too slow for real-time opt
- limited guarantees

new data-driven view

- task-specific + trainable
- enables real-time opt
- tight data-driven guarantees

I am actively looking for PhD and undergraduate students to join my group!

rajivsambharya.github.io
rajivsambharya@tamu.edu