Learning to optimize: convergence guarantees from convex to nonconvex landscapes

Luca Furieri

Joint work with Andrea Martin and Ian R. Manchester

[1] A. Martin and L. Furieri, «*Learning to optimize with convergence guarantees using nonlinear system theory*», IEEE Control Systems Letters, 2024.

[2] A. Martin, I. R. Manchester, and L. Furieri «*Learning to optimize with guarantees: a complete characterization of linearly convergent algorithms*», ArXiV 2508.00775

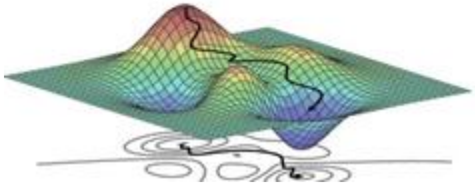
Algorithm design

Optimization program

$$\xi^* = \mathrm{argmin}_{\xi \in \Xi} f(\xi)$$

Iterative optimization algorithm

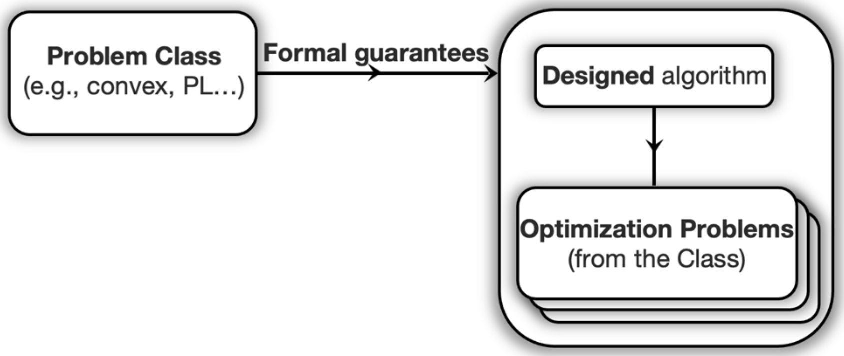
$$\xi_{t+1} = \xi_t + \mathrm{update}_t(f, \xi_t)$$



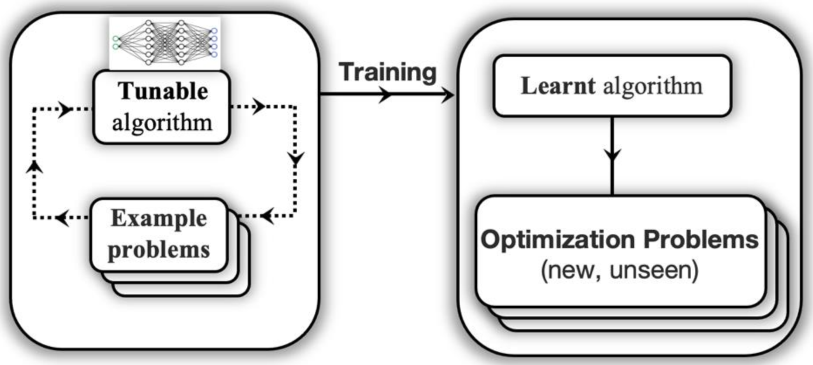
Algorithm requirements:

1. **Convergence** and **feasible iterates**
2. **Speed**: find stationary point in few steps
3. **Quality**: find low-cost stationary point

Analytical design

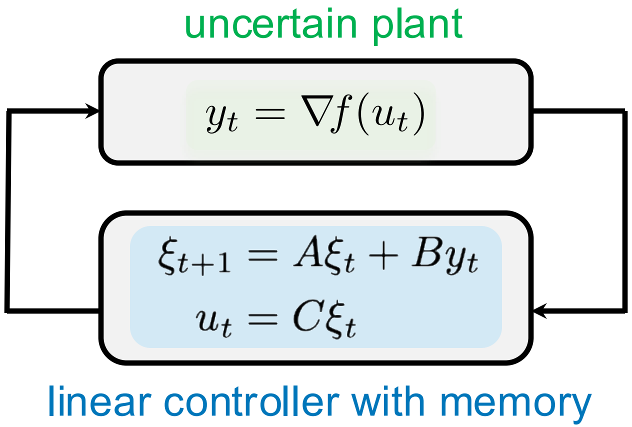


Learning-based design



Systems theory for analytical algorithm design

Classical optimization algorithms (gradient descent, accelerated…) as Lure’s systems



Example:

$$\xi_{t+1} = \xi_t - \eta \nabla f(\xi_t)$$

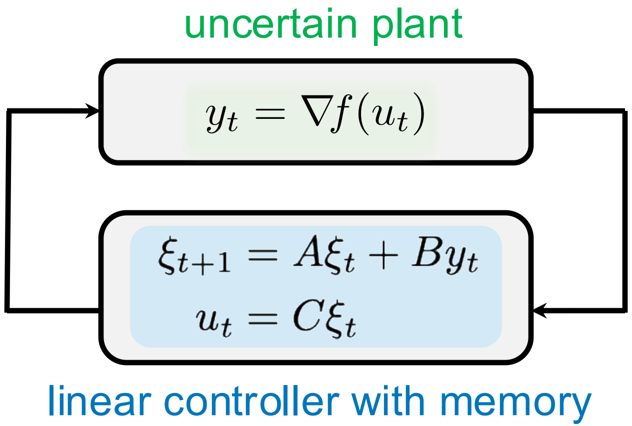
$$\updownarrow$$

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] = \left[\begin{array}{c|c} I_d & -\eta I_d \\ \hline I_d & 0_d \end{array}\right]$$

[1] L. Lessard., B. Recht, A. Packard. «*Analysis and design of optimization algorithms via integral quadratic constraints*». *SIAM Journal on Optimization*, 2016
[2] C. Scherer, C. Ebenbauer. «*Convex synthesis of accelerated gradient algorithms*». *SIAM Journal on Control and Optimization*, *59*(6), 2021

Systems theory for analytical algorithm design

Classical optimization algorithms (gradient descent, accelerated…) as Lure's systems



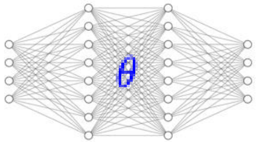
- Design of new algorithms, i.e., matrices (A,B,C)…
  - …leveraging IQCs and robust control theory[1],[2]

✅ Optimal worst-case convergence rates ❌ Limited to convex objective functions

[1] L. Lessard., B. Recht, A. Packard. «*Analysis and design of optimization algorithms via integral quadratic constraints*». *SIAM Journal on Optimization*, 2016

[2] C. Scherer, C. Ebenbauer. «*Convex synthesis of accelerated gradient algorithms*». *SIAM Journal on Control and Optimization*, *59*(6), 2021

Machine learning for algorithm design

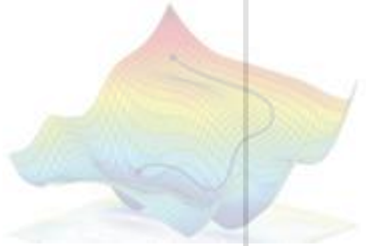
Idea: let a neural network guide the algorithm updates → $\xi_{t+1} = \xi_t +$ 

Train parameters θ to minimize $\mathbb{E}_{f \in \text{Examples}} \left[\sum_{t=0}^{T} \alpha |\nabla f(\xi_t)|^2 + \gamma f(\xi_t) \right]$

promotes convergence

class of example problems of interest

promotes solution quality

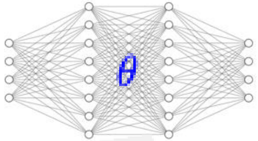


✔ Empirical performance and generalization ✘ Lack of formal guarantees

[1] M. Andrychowiz..., N. De Freitas. «*Learning to learn by gradient descent by gradient descent*». NeurIPS, 2016.

[2] K. Li. and J. Malik. «*Learning to optimize*». ICLR, 2016

Machine learning for algorithm design

Idea: let a neural network guide the algorithm updates ⟶ $\xi_{t+1} = \xi_t +$ 

Train parameters θ

class

proble

Today's focus

exploit flexibility of learned updates…

…while preserving convergence and feasibility guarantees

promotes solution quality

✅ Empirical performance and generalization

❌ Lack of formal guarantees

[1] M. Andrychowiz..., N. De Freitas. «*Learning to learn by gradient descent by gradient descent*». NeurIPS, 2016.

[2] K. Li. and J. Malik. «*Learning to optimize*». ICLR, 2016

Problem Formulation

- Let $\mathcal{F}$ be a family of objective functions (convex, smooth, PL…)
- Let π be *a legacy algorithm* $\xi_{t+1} = \pi(\xi_{t:0})$ to optimize any function $f \in \mathcal{F}$

Problem Formulation

- Let $\mathcal{F}$ be a family of objective functions (convex, smooth, PL…)
- Let π be *a legacy algorithm* $\xi_{t+1} = \pi(\xi_{t:0})$ to optimize any function $f \in \mathcal{F}$

- *Some objective functions are more frequent than others…* e.g. MPC

$$\min_{u_0,\dots,u_{N-1}} \sum_{k=0}^{N-1} x_k^\top Q x_k + u_k^\top R u_k + x_N^\top Q x_N$$
$$\text{subject to } x_0 = x_t\,,\ x_{k+1} = Ax_k + Bu_k$$
$$x_k \in \mathcal{X}\,,\ u_k \in \mathcal{U}\,,\ x_N \in \mathcal{X}_f$$

$\xi = (u_0, u_1, \dots, u_N)$ ⇒

$$\min_{\xi} \ \xi^\top G \xi + b^\top(x_0)\xi$$
$$\text{subject to } \ \xi \in \mathcal{C}(x_0)$$

- In general, $f \in \mathcal{F}$ is drawn from a distribution $f \sim \mathbb{D}_{\mathcal{F}}$

Problem Formulation

Goal: Evolve the performance of legacy algorithm π over instances $f \sim \mathbb{D}_{\mathcal{F}}$...

...without losing worst-case guarantees over the entire family $\mathcal{F}$.

Problem Formulation

Goal: Evolve the performance of legacy algorithm π over instances $f \sim \mathbb{D}_{\mathcal{F}}$…

…without losing worst-case guarantees over the entire family $\mathcal{F}$.

- We design *evolved algorithms* in the form

$$\xi_{t+1} = \pi(\xi_t) + v(\xi_{t:0})$$

legacy algorithm ensures convergence/feasibility over $\mathcal{F}$ ($\pi(\xi_t)$)

enhancement term to be designed ($v(\xi_{t:0})$)

- Algorithm performance for $f \sim \mathbb{D}_{\mathcal{F}}$ measured as $\mathbb{E}_{f\sim \mathbb{D}_{\mathcal{F}}}\left[\sum_{t=0}^{T} \alpha |\nabla f(\xi_t)|^2 + \gamma f(\xi_t)\right]$

Scenarios we consider

Scenario A[1]: smooth nonconvex landscapes

$$\min_{\xi \in \mathbb{R}^d} f(\xi)$$

problem class:

f is β-smooth

legacy algorithm:

gradient descent

convergence guarantee:

asymptotic convergence to stationary point

Scenario B[2]: composite convex landscapes

$$\min_{\xi \in \mathbb{R}^d} f(\xi) + g(\xi)$$

problem class:

f, g are convex

g is nonsmooth

legacy algorithm:

accelerated methods (e.g., heavy-ball, Nesterov…)

convergence guarantee:

linear convergence

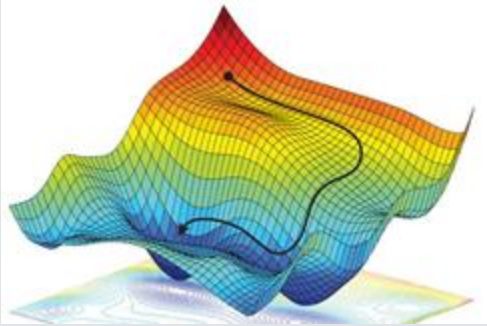
$$|\xi_{t+1} - \xi^\star| \leq p(t)\gamma^t |\xi_0 - \xi^\star|$$

[1] A. Martin and L. Furieri, «*Learning to optimize with convergence guarantees using nonlinear system theory*», IEEE Control Systems Letters, 2024.

[2] A. Martin, I. R. Manchester, and L. Furieri «*Learning to optimize with guarantees: a complete characterization of linearly convergent algorithms*», ArXiV 2508.00775

Scenario A

Learning to optimize for smooth nonconvex landscapes



[1] A. Martin and L. Furieri, «*Learning to optimize with convergence guarantees using nonlinear system theory*», IEEE Control Systems Letters, 2024.

Main result 1: a separation principle for algorithms

Consider the iterations: $\xi_{t+1} = \xi_t - \eta \nabla f(\xi_t) + v_t$

legacy algorithm: gradient descent

enhancement term

Main result 1: a separation principle for algorithms

Consider the iterations: $\xi_{t+1} = \xi_t - \eta \nabla f(\xi_t) + v_t$

legacy algorithm: gradient descent

enhancement term

If $0 < \eta < \beta^{-1}$, and $\sum_{t=0}^{\infty} |v_t|^2 < \infty$, then $\sum_{t=0}^{\infty} |\nabla f(\xi_t)|^2 < \infty$

⇨ Evolve gradient descent by designing a finite-energy sequence v_t

Main result 1: a separation principle for algorithms

Consider the iterations: $\xi_{t+1} = \xi_t - \eta \nabla f(\xi_t) + v_t$

legacy algorithm: gradient descent

enhancement term

If $0 < \eta < \beta^{-1}$, and $\sum_{t=0}^{\infty} |v_t|^2 < \infty$, then $\sum_{t=0}^{\infty} |\nabla f(\xi_t)|^2 < \infty$

⇨ Evolve gradient descent by designing a finite-energy sequence v_t

❌ **Needs proof**: exponential stability with $v_t = 0$ may not imply stability when $\sum_{t=0}^{\infty} |v_t|^2 < \infty$[1]

[1] H. K. Khalil and J. W. Grizzle. «*Nonlinear systems* (Vol. 3)» Upper Saddle River, NJ: Prentice hall, 2002

Main result 2: universality

Take *any* target algorithm $\xi_{t+1} = \sigma_t(\xi_{t:0})$ converging to a stationary point for all $f \in \mathcal{F}_{smooth}$

The target algorithm $\xi_{t+1} = \sigma_t(\xi_{t:0})$ is equivalent to

$$\xi_{t+1} = \xi_t - \eta \nabla f(\xi_t) + v_t(f, \xi_0)$$

for some sequence $v_t(f, \xi_0)$ with finite energy.

$$\left(\sum_{t=0}^{\infty} |v_t(f, \xi_0)|^2 < \infty \right)$$

Proof insight

1. Construct the update rule $v_t(f, \xi_0)$ matching the algorithm

2. Prove that $\sum_{t=0}^{\infty} |v_t(f, \xi_0)|^2 < \infty$

Akin to Youla and System Level Synthesis (SLS) for algorithm design

Implications

Evolve gradient descent using automatic differentiation while preserving convergence

Samples from $f \in \mathbb{D}_{\mathcal{F}}$

$$\min_{\theta \in \mathbb{R}^n} \sum_{f \in \texttt{Examples}} \left[\sum_{t=0}^{T} \alpha \|\nabla f(\xi_t)\|_2^2 + \gamma f(\xi_t) \right]$$

$$\text{subject to } \xi_{t+1} = \xi_t - \eta \nabla f(\xi_t) + v_t(f, \xi_{t:0}, \theta)$$

Neural-network parametrizations + PyTorch

How to evolve your convergent algorithm with neural networks

- Factorize $v_t(\xi_{t:0}, \theta)$ using two neural networks:

neural network 2: direction

$$v_t(f, \xi_{t:0}) = r_t(\xi_0, \theta) d_t(\xi_{t:0}, f(\xi_{t:0}), \nabla f(\xi_{t:0}), \theta)$$

neural network 1: vanishing radius[1],[2]

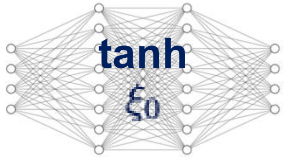
⇒ trainable NN
stable nonlinear dynamics

- We prove: **the factorization above preserves universality**

[1] M. Revay, R. Wang, and I.R. Manchester, «*Recurrent equilibrium networks: Flexible dynamic models with guaranteed stability and robustness*». *IEEE Transactions on Automatic Control*, *2023*
[2] A. Orvieto, S. L. Smith, A. Gu, A. Fernando, C. Gulcehre, R. Pascanu, S. De, «*Resurrecting Recurrent Neural Networks for Long Sequences", ICML, 2024*

Experiment: training a perceptron for image classification

data and labels



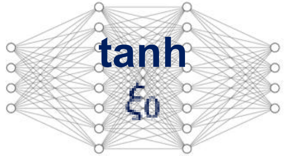
classifier prediction $f(\xi_0)$

Experiment: training a perceptron for image classification

1) train the perceptron with an algorithm (fixed θ)

data and labels

→



→

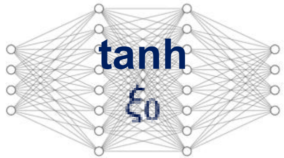
classifier prediction $f(\xi_t)$

$$\xi_{t+1} = \xi_t - \eta \nabla f(\xi_t) + r_t(\xi_0, \theta) d_t(\xi_{t:0}, f(\xi_{t:0}), \nabla f(\xi_{t:0}), \theta)$$

Experiment: training a perceptron for image classification

1) train the perceptron with an algorithm (fixed θ)

data and labels



classifier prediction $f(\xi_t)$

$$\xi_{t+1} = \xi_t - \eta \nabla f(\xi_t) + r_t(\xi_0, \theta) d_t(\xi_{t:0}, f(\xi_{t:0}), \nabla f(\xi_{t:0}), \theta)$$

2) train the algorithm itself (train θ)

After training the classifier, evaluate $\texttt{AlgPerf}(\theta) = \sum_{t=0}^{T} \alpha |\nabla f(\xi_t)|^2 + \gamma f(\xi_t)$ …

… backpropagate through θ

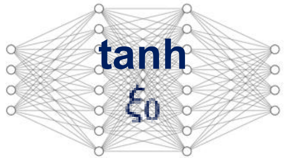
... then update θ

Experiment: training a perceptron for image classification

1) train the perceptron with an algorithm (fixed θ)

data and labels

→



→

classifier prediction $f(\xi_t)$

$$\xi_{t+1} = \xi_t - \eta \nabla f(\xi_t) + r_t(\xi_0, \theta) d_t(\xi_{t:0}, f(\xi_{t:0}), \nabla f(\xi_{t:0}), \theta)$$

2) train the algorithm itself (train θ)

After training the classifier, evaluate $\texttt{AlgPerf}(\theta) = \sum_{t=0}^{T} \alpha |\nabla f(\xi_t)|^2 + \gamma f(\xi_t)$ …

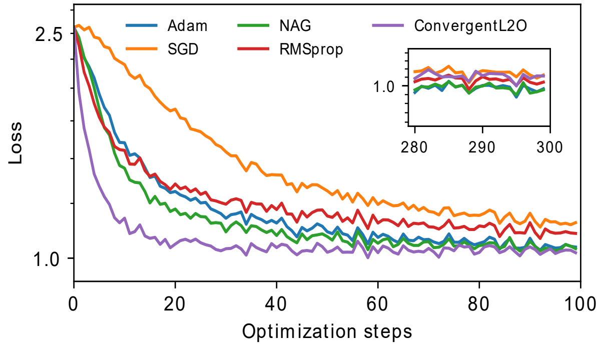
… backpropagate through θ

... then update θ

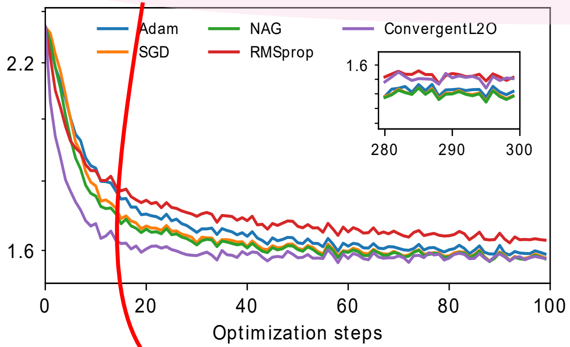
3) after training θ, compare with classical optimizers

Experiment: training a perceptron for image classification

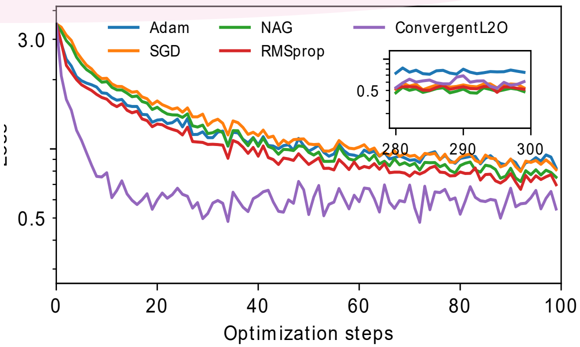
Activation function: tanh



Activation function: sigmoid

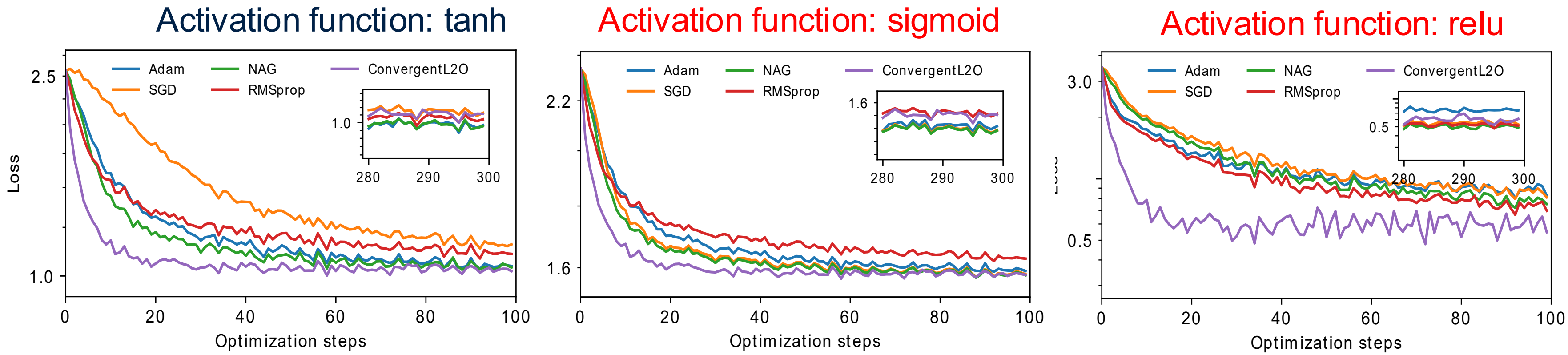


Activation function: relu



Out-of-sample generalization!

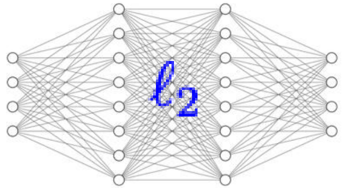
Experiment: training a perceptron for image classification



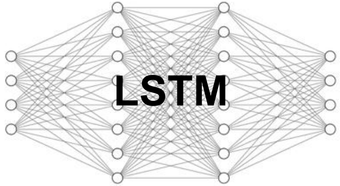
Classifier test accuracy

Step t = 20	tanh	sigmoid	ReLU
Adam	71.7 ± 5.1%	76.1 ± 3.1%	52.7 ± 11.1%
SGD	44.9 ± 4.2%	79.7 ± 1.9%	49.8 ± 9.3%
NAG	79.7 ± 1.4%	81.1 ± 1.5%	52.7 ± 10.2%
RMSprop	69.4 ± 2.9%	72.8 ± 2.3%	61.1 ± 8.9%
ConvergentL2O	87.0 ± 0.5%	86.8 ± 0.6%	86.3 ± 0.6%
LSTM	82.2 ± 0.1%	83.3 ± 0.1%	88.3 ± 0.0%

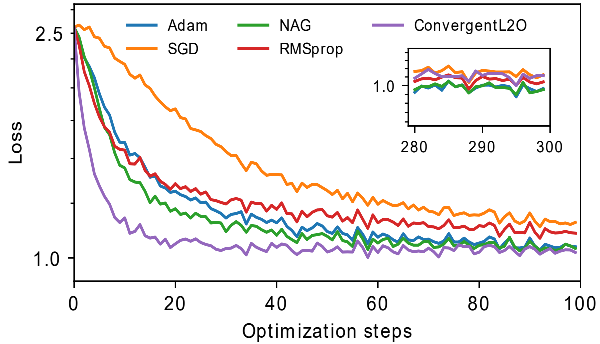
$\xi_{t+1} = \xi_t - \eta \nabla f(\xi_t) +$

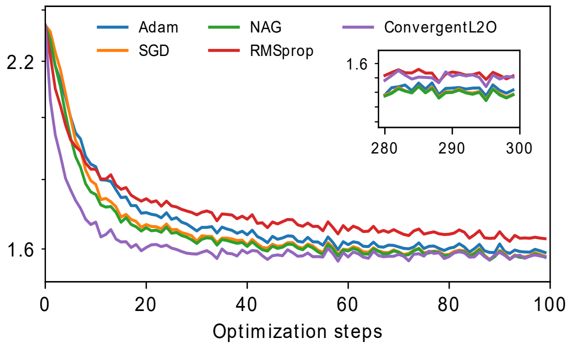


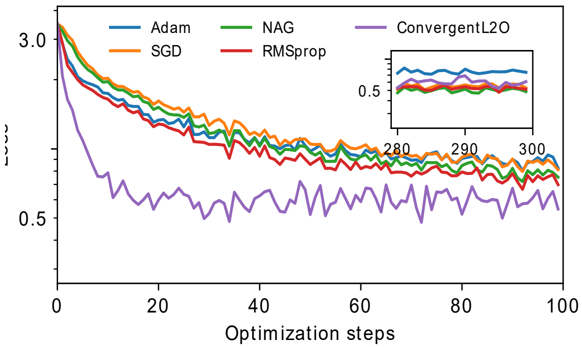
$\xi_{t+1} = \xi_t +$



Experiment: training a perceptron for image classification

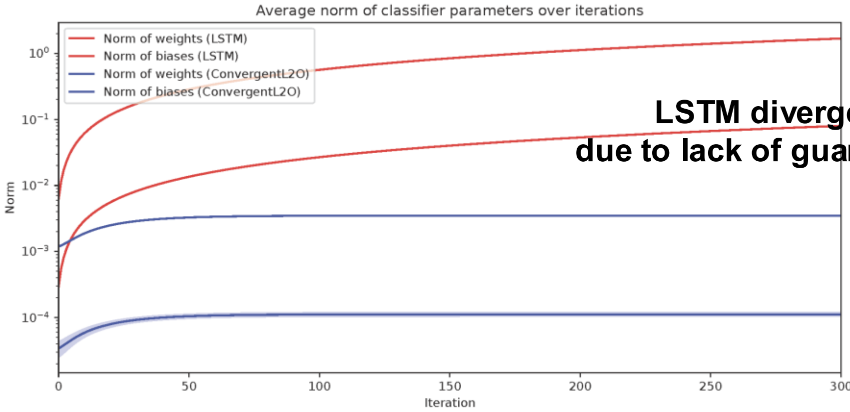






Classifier test accuracy

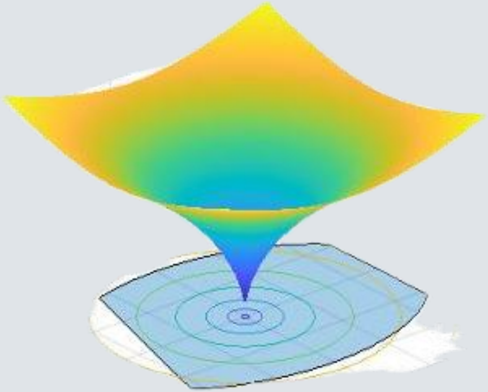
Step t = 20	tanh	sigmoid	ReLU
Adam	71.7 ± 5.1%	76.1 ± 3.1%	52.7 ± 11.1%
SGD	44.9 ± 4.2%	79.7 ± 1.9%	49.8 ± 9.3%
NAG	79.7 ± 1.4%	81.1 ± 1.5%	52.7 ± 10.2%
RMSprop	69.4 ± 2.9%	72.8 ± 2.3%	61.1 ± 8.9%
ConvergentL2O	87.0 ± 0.5%	86.8 ± 0.6%	86.3 ± 0.6%
LSTM	82.2 ± 0.1%	83.3 ± 0.1%	88.3 ± 0.0%



LSTM diverges due to lack of guarantees

Scenario B

Learning to evolve linearly convergent algorithms



[2] A. Martin, I. R. Manchester, and L. Furieri «*Learning to optimize with guarantees: a complete characterization of linearly convergent algorithms*», ArXiV 2508.00775

Main result 1: evolving a contraction

Consider the iterations: $\xi_{t+1} = \pi(f, \xi_t) + v_t$

monotonically linearly convergent:

$$|\xi_{t+1} - \xi^\star| \leq \gamma^t |\xi_0 - \xi^\star|$$

If $|v_t| \leq p(t)\gamma^t$, then $|\xi_{t+1} - \xi^\star| \leq q(t)\gamma^t |\xi_0 - \xi^\star|$

⇨ Evolve contracting algorithms by designing exponentially decaying v_t

Main result 1: evolving a contraction

Consider the iterations: $\xi_{t+1} = \pi(f, \xi_t) + v_t$

monotonically linearly convergent:

$$|\xi_{t+1} - \xi^\star| \leq \gamma^t |\xi_0 - \xi^\star|$$

If $|v_t| \leq p(t)\gamma^t$, then $|\xi_{t+1} - \xi^\star| \leq q(t)\gamma^t |\xi_0 - \xi^\star|$

⇨ Evolve contracting algorithms by designing exponentially decaying v_t

Main proof idea: study a perturbed scalar linear system

$$|\xi_t - \xi^\star| = \delta_t \leq \gamma^t \delta_0 + \sum_{k=0}^{t-1} \gamma^k |v_{t-1-k}| \leq \gamma^t \delta_0 + \sum_{k=0}^{t-1} \gamma^k p(t-1-k)\gamma^{t-1-k}$$

$$\leq \gamma^t \left(\delta_0 + \frac{1}{\gamma} \sum_{k=0}^{t-1} p(k) \right) = \boxed{\gamma^t q(t)}$$

Same rate γ , degree of $p(t)$ +1

Main result 1: evolving non-monotonic accelerated algorithms

Consider the iterations: $\xi_{t+1} = \pi(f, \xi_t) + v_t$

non-monotonically linearly convergent:

$$|\xi_{t+1} - \xi^\star| \leq r(t)\gamma^t |\xi_0 - \xi^\star|$$

(e.g., Nesterov for strongly convex)

Let $N \in \mathbb{N}$ be large enough to satisfy $r(N)\gamma^N < 1$.
If $|v_t| \leq p(t)\gamma^t$ is applied once every N steps, then:

$$|\xi_{t+1} - \xi^\star| \leq q(t)\left(\sqrt[N]{r(N)\gamma}\right)^t |\xi_0 - \xi^\star|$$

Main result 1: evolving non-monotonic accelerated algorithms

Consider the iterations: $\xi_{t+1} = \pi(f, \xi_t) + v_t$

non-monotonically linearly convergent:

$$|\xi_{t+1} - \xi^\star| \leq r(t)\gamma^t |\xi_0 - \xi^\star|$$

(e.g., Nesterov for strongly convex)

Let $N \in \mathbb{N}$ be large enough to satisfy $r(N)\gamma^N < 1$.
If $|v_t| \leq p(t)\gamma^t$ is applied once every N steps, then:

$$|\xi_{t+1} - \xi^\star| \leq q(t)\left(\sqrt[N]{r(N)\gamma}\right)^t |\xi_0 - \xi^\star|$$

trade-off: how often we inject v_t vs worst-case convergence rate $\sqrt[N]{r(N)\gamma}$

Main proof idea: the repeated legacy algorithm $\xi_{t+1} = \pi^N(f, \xi_t)$ remains monotonic...

Main result 2: universality

Take *any* linearly convergent target algorithm $\xi_{t+1} = \sigma_t(\xi_{t:0})$ with rate γ

The target algorithm $\xi_{t+1} = \sigma_t(\xi_{t:0})$ is equivalent to

$$\xi_{t+1} = \pi(f, \xi_t) + v_t(f, \xi_{t:0})$$

for some sequence $v_t(f, \xi_{t:0})$ with $|v_t| \leq p(t)\gamma^t$ if $\pi(f, \xi_t)$ is monotonic and Lipschitz wrt ξ_t

Main result 2: universality

Take *any* linearly convergent target algorithm $\xi_{t+1} = \sigma_t(\xi_{t:0})$ with rate γ

The target algorithm $\xi_{t+1} = \sigma_t(\xi_{t:0})$ is equivalent to

$$\xi_{t+1} = \pi(f, \xi_t) + v_t(f, \xi_{t:0})$$

for some sequence $v_t(f, \xi_{t:0})$ with $|v_t| \leq p(t)\gamma^t$ if $\pi(f, \xi_t)$ is monotonic and Lipschitz wrt ξ_t

Proof sketch

- The sequence v_t achieving te same iterations as $\xi_{t+1} = \sigma_t(\xi_{t:0})$ is

$$v_t = -\pi(f, \xi_t) + \sigma_t(\xi_{t:0}) = -(\pi(f, \xi_t) - \xi_t) + (\sigma_t(\xi_{t:0}) - \xi_t)$$

Main result 2: universality

Take *any* linearly convergent target algorithm $\xi_{t+1} = \sigma_t(\xi_{t:0})$ with rate γ

The target algorithm $\xi_{t+1} = \sigma_t(\xi_{t:0})$ is equivalent to

$$\xi_{t+1} = \pi(f, \xi_t) + v_t(f, \xi_{t:0})$$

for some sequence $v_t(f, \xi_{t:0})$ with $|v_t| \leq p(t)\gamma^t$ if $\pi(f, \xi_t)$ is monotonic and Lipschitz wrt ξ_t

Proof sketch

- The sequence v_t achieving te same iterations as $\xi_{t+1} = \sigma_t(\xi_{t:0})$ is

$$v_t = -\pi(f, \xi_t) + \sigma_t(\xi_{t:0}) = -(\pi(f, \xi_t) - \xi_t) + (\sigma_t(\xi_{t:0}) - \xi_t)$$

$$|\pi(f, \xi_t) - \xi_t| \leq (L_\pi + 1)|\xi_t - \xi^\star|$$

vanishes with γ^t

Main result 2: universality

Take *any* linearly convergent target algorithm $\xi_{t+1} = \sigma_t(\xi_{t:0})$ with rate γ

The target algorithm $\xi_{t+1} = \sigma_t(\xi_{t:0})$ is equivalent to

$$\xi_{t+1} = \pi(f, \xi_t) + v_t(f, \xi_{t:0})$$

for some sequence $v_t(f, \xi_{t:0})$ with $|v_t| \leq p(t)\gamma^t$ if $\pi(f, \xi_t)$ is monotonic and Lipschitz wrt ξ_t

Proof sketch

- The sequence v_t achieving te same iterations as $\xi_{t+1} = \sigma_t(\xi_{t:0})$ is

$$v_t = -\pi(f, \xi_t) + \sigma_t(\xi_{t:0}) = -(\pi(f, \xi_t) - \xi_t) + (\sigma_t(\xi_{t:0}) - \xi_t)$$

$$|\pi(f, \xi_t) - \xi_t| \leq (L_\pi + 1)|\xi_t - \xi^\star| \qquad \xi_{t+1} - \xi_t$$

vanishes with γ^t — vanishes with γ^t by assumption

Main result 2: universality

Take *any* linearly convergent target algorithm $\xi_{t+1} = \sigma_t(\xi_{t:0})$ with rate γ

The target algorithm $\xi_{t+1} = \sigma_t(\xi_{t:0})$ is equivalent to

$$\xi_{t+1} = \pi(f, \xi_t) + v_t(f, \xi \ldots$$

for some sequence $v_t(f, \xi \ldots$ … nic and Lipschitz wrt ξ_t

Many legacy algorithms are Lipschitz wrt ξ_t

- Gradient descent…
- Nesterov, heavy-ball, triple-momentum…
- IQC-based (Lessard, Scherer…)

- The sequence v_t …

$$v_t = -\pi(f, \ldots \ldots - \xi_t) + (\sigma_t(\xi_{t:0}) - \xi_t)$$

$$|\pi(f, \xi_t) - \xi_t| \leq (L_\pi + 1)|\xi_t - \xi^*|$$

vanishes with γ^t

$$\xi_{t+1} - \xi_t$$

vanishes with γ^t by assumption

Examples of compatible problems: unconstrained

$$\min_{\xi \in \mathbb{R}^d} f(\xi)$$

problem class:	**legacy algorithm:**	**convergence guarantees:**
f is strongly convex f is smooth	heavy-ball, Nesterov, accelerated methods of [1], [2]	preserves linear convergence $\xi_{t+1} = \pi(f, \xi_t) + v_t$

[1] L. Lessard., B. Recht, A. Packard. «*Analysis and design of optimization algorithms via integral quadratic constraints*». *SIAM Journal on Optimization*, 2016

[2] C. Scherer, C. Ebenbauer. «*Convex synthesis of accelerated gradient algorithms*». *SIAM Journal on Control and Optimization*, *59*(6), 2021

Examples of compatible problems: unconstrained

$$\min_{\xi \in \mathbb{R}^d} f(\xi)$$

problem class:

f is strongly convex
f is smooth

legacy algorithm:

heavy-ball, Nesterov,
accelerated methods of [1], [2]

convergence guarantees:

preserves linear convergence

$$\xi_{t+1} = \pi(f, \xi_t) + v_t$$

$$\min_{\xi \in \mathbb{R}^d} f(\xi) + g(\xi)$$

problem class:

f is strongly convex
g is convex, nonsmooth

legacy algorithm:

proximal gradient descent

$$\pi(f, \xi_t) = \mathrm{prox}_g(\xi_t - \eta \nabla f(\xi_t))$$

convergence guarantees:

all linearly convergent algorithms

$$\xi_{t+1} = \pi(f, \xi_t) + v_t(f, \xi_0)$$

[1] L. Lessard., B. Recht, A. Packard. «*Analysis and design of optimization algorithms via integral quadratic constraints*». *SIAM Journal on Optimization*, 2016
[2] C. Scherer, C. Ebenbauer. «*Convex synthesis of accelerated gradient algorithms*». *SIAM Journal on Control and Optimization*, *59*(6), 2021

Examples of compatible problems: constrained

$$\min_{\xi \in \mathbb{R}^d} \; f(\xi)$$

$$\text{subject to } \; A\xi \leq b$$

problem class:

f is strongly convex

legacy algorithm:

proximal gradient descent

$$\pi(f, \xi_t) = \text{proj}_{\Xi}(\xi_t - \eta \nabla f(\xi_t))$$

convergence guarantees:

all linearly convergent algorithms

$$\xi_{t+1} = \pi(f, \xi_t) + v_t(f, \xi_0)$$

feasibility only upon convergence…

Examples of compatible problems: constrained

$$\min_{\xi \in \mathbb{R}^d} f(\xi)$$

$$\text{subject to } A\xi \leq b$$

problem class:

f is strongly convex

legacy algorithm:

proximal gradient descent

$$\pi(f, \xi_t) = \text{proj}_\Xi(\xi_t - \eta \nabla f(\xi_t))$$

convergence guarantees:

all linearly convergent algorithms

$$\xi_{t+1} = \pi(f, \xi_t) + v_t(f, \xi_0)$$

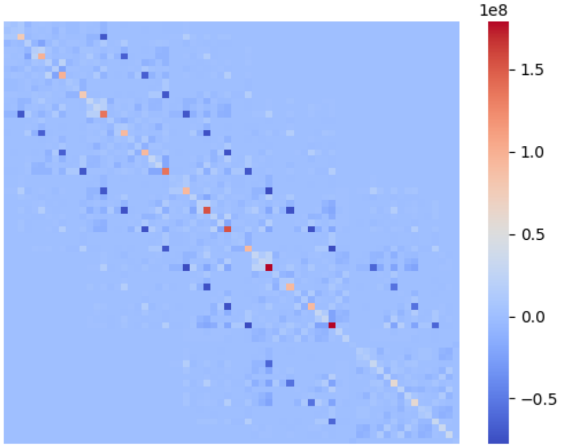
guarantees if $Av_t \leq 0$:

all linearly convergent algorithms

$$\xi_{t+1} = \pi(f, \xi_t) + v_t(f, \xi_0)$$

with feasible iterates $\xi_t \in \Xi$

Experiment: solving hard systems of linear equations

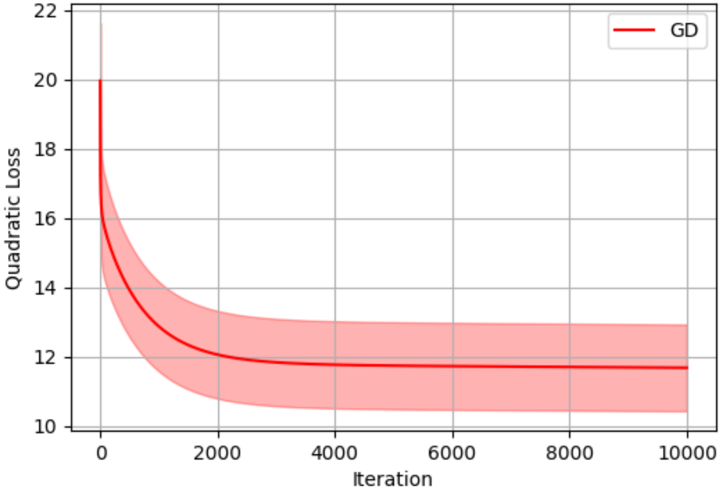


$$\min_{\xi} \ |A\xi - b_i|^2$$

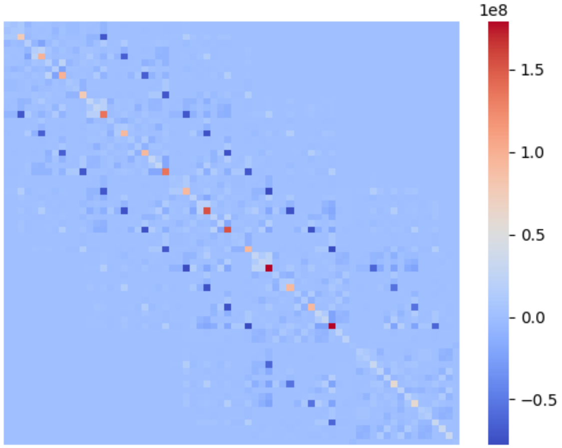
$(A, b_i) \sim \texttt{Gaussian}(0.5, 0.04)$

slow convergence speed when $\kappa(A^\top A)$ is large

…in this example $\kappa(A^\top A) \sim 18.7\,\mathrm{M}$



Experiment: solving hard systems of linear equations



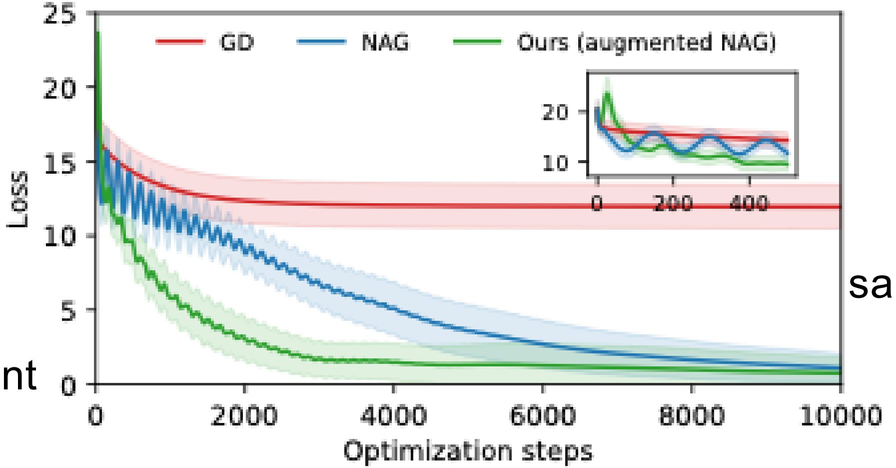
$$\min_{\xi} \; |A\xi - b_i|^2$$

$(A, b_i) \sim \texttt{Gaussian}(0.5, 0.04)$

slow convergence speed when $\kappa(A^\top A)$ is large

…in this example $\kappa(A^\top A) \sim 18.7\,\mathrm{M}$

Train $v_t(\xi_{t:0})$ to evolve NAG…



improved transient behavior

same asymptotic rate as NAG

Conclusions

- A characterization of all *asymptotically (A)* and *linearly convergent (B)* algorithms
  - legacy algorithm as a base policy + nonlinear dynamic updates

Neural-network based evolution of classical algorithms

Conclusions

- A characterization of all *asymptotically (A)* and *linearly convergent (B)* algorithms
  - legacy algorithm as a base policy + nonlinear dynamic updates

Neural-network based evolution of classical algorithms

Future work

- Performance generalization guarantees[1]
- Impact on Model Predictive Control (e.g., evolve IPOPT, OSQP…)
- Inverse design, *e.g.: «for which control cost is NAG optimal?*

[1] R. Sambharya, B. Stellato, "*Data-Driven Performance Guarantees for Classical and Learned Optimizers*", [ArXiV, 2024]