Part 2

Learning to optimize with convergence guarantees using nonlinear system theory

[1] Andrea Martin and Luca Furieri, "*Learning to optimize with convergence guarantees using nonlinear system theory*", IEEE LCSS, 2024

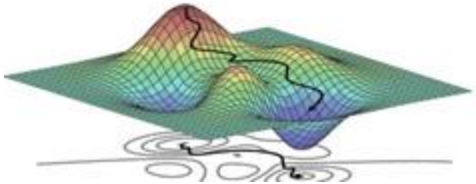
Algorithm design

Non-convex program

$$\boxed{x^\star = \mathrm{argmin}_{x \in \mathbb{R}^d} f(x)}$$

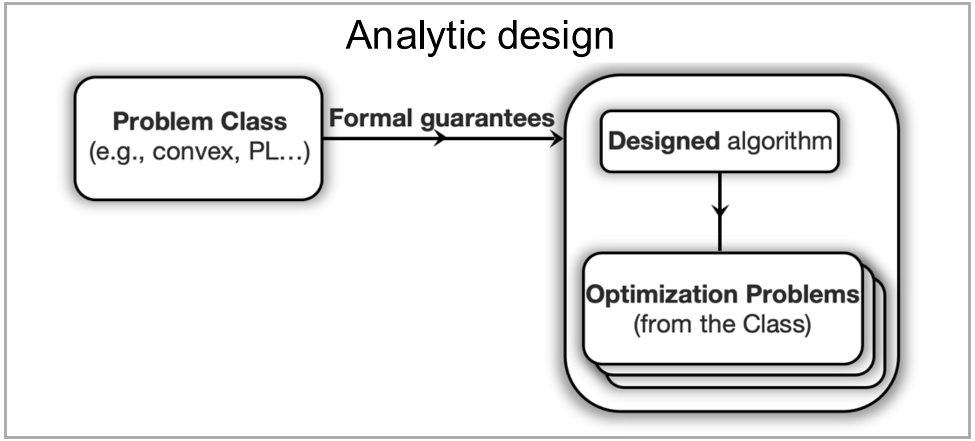
Iterative optimization algorithm

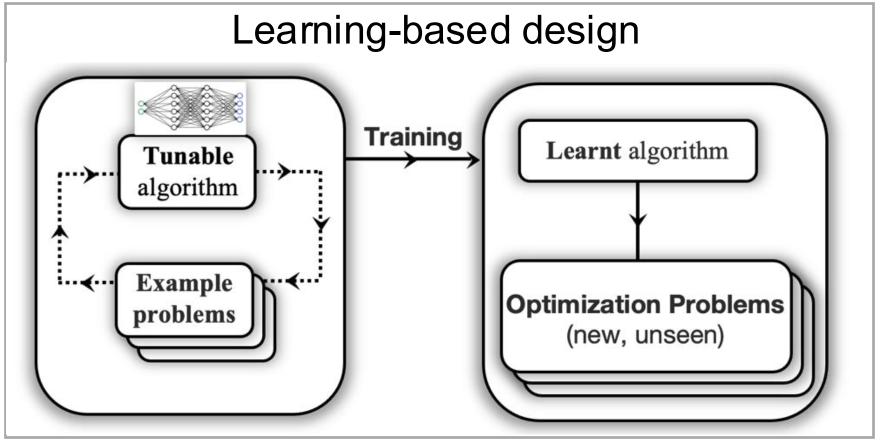
$$x_{t+1} = x_t + \mathrm{update}_t(f, x_t)$$



Algorithm requirements:

1. **Convergence** as $t \to \infty$
2. **Speed**: find stationary point in few steps
3. **Quality**: find low-cost stationary point

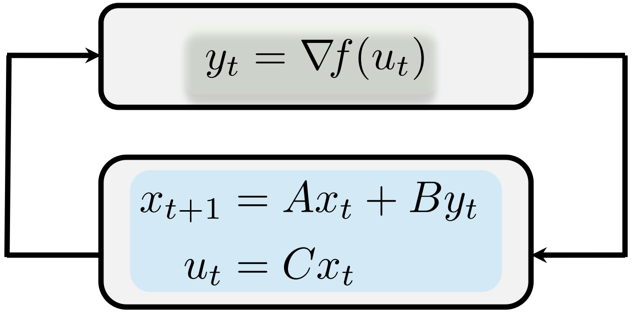




Background: system theory for algorithm design

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

uncertain plant



linear controller with memory

Example:

$$x_{t+1} = x_t - \eta \nabla f(x_t)$$

$$\updownarrow$$

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

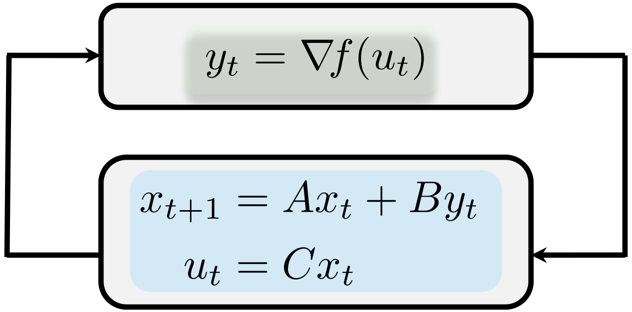
[1] Lessard, Recht, Packard, "*Analysis and design of optimization algorithms via integral quadratic constraints*", SIAM Journal on Optimization, 2016

[2] Scherer, C., & Ebenbauer, C. «*Convex synthesis of accelerated gradient algorithms*». SIAM Journal on Control and Optimization, 59(6), 4615-4645, 2021

Background: system theory for algorithm design

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

uncertain plant



linear controller with memory

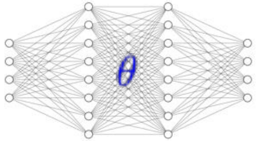
- Design of new algorithms, i.e., matrices (A,B,C)…
  - …leveraging IQCs and robust control theory[1],[2]

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

[1] Lessard, Recht, Packard, "*Analysis and design of optimization algorithms via integral quadratic constraints*", SIAM Journal on Optimization, 2016

[2] Scherer, C., & Ebenbauer, C. «*Convex synthesis of accelerated gradient algorithms*». SIAM Journal on Control and Optimization, 59(6), 4615-4645, 2021

Background: machine learning for algorithm design

Idea: let a neural network guide the algorithm updates $\longrightarrow$ $x_{t+1} = x_t +$ 

Train parameters θ to minimize $\mathbb{E}_{\substack{f\sim\mathcal{F}\\ x_0\sim\mathcal{X}_0}}\left[\underbrace{\sum_{t=0}^{T} \alpha_t\|\nabla f(x_t)\|_2^2 + \gamma_t f(x_t)}_{\texttt{MetaLoss}(f,\mathbf{x})}\right]$

promotes convergence

class of example problems of interest

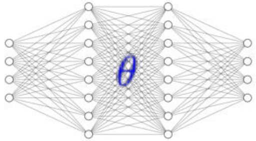
promotes solution quality

✅ Empirical performance and generalization ❌ Lack of formal guarantees

[1] Andrychowicz, M., ... & De Freitas, N. (2016). *Learning to learn by gradient descent by gradient descent*. Advances in neural information processing systems

[2] Li, K., & Malik, J. *Learning to optimize*. International Conference on Learning Representations, 2016

Background: machine learning for algorithm design

Idea: let a neural network guide the algorithm updates $\longrightarrow$ $x_{t+1} = x_t +$ 

convergence

Train paramete

class
proble

$t)$

promotes solution quality

Today's focus

exploit flexibility of learned updates…

…while preserving convergence guarantees

Problem formulation

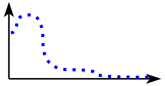
Design of <u>optimal</u> convergent algorithms

Let $\mathcal{S}_\beta$ denote the class of non-convex functions with βLipschitz gradients:

$$\mathbf{x} = (x_0, x_1, x_2, \dots)$$

$$\begin{aligned} \min_{\boldsymbol{\pi}} \quad & \mathbb{E}_{\substack{f\sim\mathcal{F} \\ x_0\sim\mathcal{X}_0}} \left[\texttt{MetaLoss}(f, \mathbf{x})\right] \\ \text{subject to} \quad & x_{t+1} = x_t + \pi_t(f, x_{t:0}), \\ & \boldsymbol{\pi}(f, \mathbf{x}) \text{ converges } \forall f \in \mathcal{S}_\beta, \end{aligned}$$

Algorithm dynamics



$$\boldsymbol{\pi}(f, \mathbf{x}) \in \ell_2, \quad \nabla f(\mathbf{x}) \in \ell_2,$$

Problem formulation

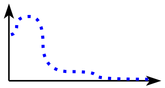
Design of <u>optimal</u> convergent algorithms

Let $\mathcal{S}_\beta$ denote the class of non-convex functions with β-Lipschitz gradients:

$\mathbf{x} = (x_0, x_1, x_2, \dots)$

$$
\begin{aligned}
\min_{\boldsymbol{\pi}} \ & \mathbb{E}_{\substack{f \sim \mathcal{F} \\ x_0 \sim \mathcal{X}_0}} \left[\texttt{MetaLoss}(f, \mathbf{x})\right] \\
\text{subject to } & x_{t+1} = x_t + \pi_t(f, x_{t:0}), \\
& \boldsymbol{\pi}(f, \mathbf{x}) \text{ converges } \forall f \in \mathcal{S}_\beta,
\end{aligned}
$$

Algorithm dynamics



$\boldsymbol{\pi}(f, \mathbf{x}) \in \ell_2, \ \nabla f(\mathbf{x}) \in \ell_2,$

Example: the gradient descent algorithm $\boldsymbol{\pi}(f, \mathbf{x}) = -\eta \nabla f(\mathbf{x})$ converges $\forall f \in \mathcal{S}_\beta$ if $0 < \eta < \beta^{-1}$

[1] Bertsekas, D. P., & Tsitsiklis, J. N. «*Gradient convergence in gradient methods with errors. SIAM Journal on Optimization*», *10*(3), 627-642, 2000

Main result 1: a separation principle for algorithms

Consider the algorithm: $\boldsymbol{\pi}(f, \mathbf{x}) = -\eta \nabla f(\mathbf{x}) + \mathbf{v}$

gradient descent
ensures convergence

enhancement term to be designed
without ruining convergence

If $0 < \eta < \beta^{-1}$, $\boldsymbol{\pi}(f, \mathbf{x})$ converges $\forall f \in \mathcal{S}_\beta$ for any $\mathbf{v} \in \ell_2$

Needs proof: exponential stability with $\mathbf{v} = 0$ generally does not imply stability when $\mathbf{v} \in \ell_2$[1]

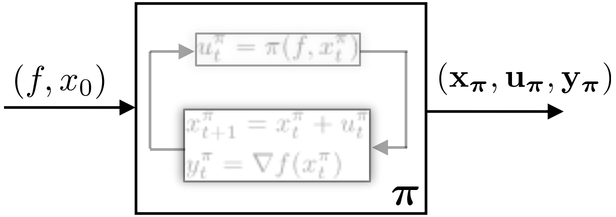
[1]Khalil, H. K. (2002). *Nonlinear systems.*

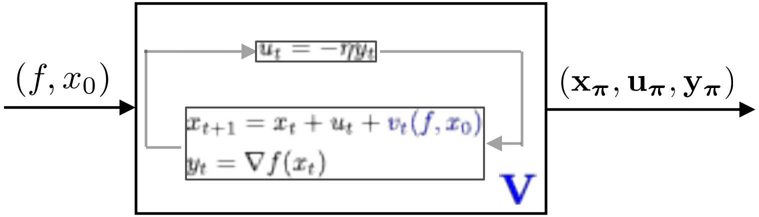
Main result 2: completeness

Take any $\boldsymbol{\pi}(f, \mathbf{x})$ that converges $\forall f \in \mathcal{S}_\beta$:

> There exist $\mathbf{V} \in \mathcal{L}_2$ such that $-\eta \nabla f(\mathbf{x}) + \mathbf{V}(f, x_0)$ and
>
> $\boldsymbol{\pi}(f, \mathbf{x})$ yield the same trajectories $\forall x_0$, $\forall f \in \mathcal{S}_\beta$

Proof insight: design a **stable closed-loop map**, rather than an update rule

Equivalence of 

and 

by picking $\mathbf{V}(f, x_0) = \eta \nabla f(\mathbf{x}_\pi(x_0)) + \mathbf{u}_\pi(x_0)$.

Implications

Learning convergent algorithms using automatic differentiation

$$\begin{aligned}\min_{\boldsymbol{\pi}} \ & \mathbb{E}_{\substack{f\sim\mathcal{F}\\ x_0\sim\mathcal{X}_0}} [\texttt{MetaLoss}(f,\mathbf{x})] \\ \text{subject to } & x_{t+1} = x_t + \pi_t(f, x_{t:0}), \\ & \boldsymbol{\pi}(f,\mathbf{x}) \text{ converges } \forall f \in \mathcal{S}_\beta,\end{aligned}$$

$$\equiv$$

$$\begin{aligned}\min_{\mathbf{V}\in\mathcal{L}_2} \ & \mathbb{E}_{\substack{f\sim\mathcal{F}\\ x_0\sim\mathcal{X}_0}} [\texttt{MetaLoss}(f,\mathbf{x})] \\ \text{subject to } & x_{t+1} = x_t - \eta\nabla f(x_t) + V_t(f, x_0),\end{aligned}$$

how to search over these operators?

Implications

Learning convergent algorithms using automatic differentiation

$$\min_{\boldsymbol{\pi}} \mathbb{E}_{\substack{f\sim\mathcal{F}\\ x_0\sim\mathcal{X}_0}} \left[\mathtt{MetaLoss}(f,\mathbf{x})\right]$$
$$\text{subject to } x_{t+1} = x_t + \pi_t(f, x_{t:0}),$$
$$\boldsymbol{\pi}(f,\mathbf{x}) \text{ converges } \forall f \in \mathcal{S}_\beta,$$

$=$

$$\min_{\theta\in\mathbb{R}^n} \mathbb{E}_{\substack{f\sim\mathcal{F}\\ x_0\sim\mathcal{X}_0}} \left[\mathtt{MetaLoss}(f,\mathbf{x})\right]$$
$$\text{subject to } x_{t+1} = x_t - \eta\nabla f(x_t) + V_t(f, x_0, \theta),$$

✓ Unconstrained parametrizations of $\mathcal{L}_2$ operators (**Part 1**) $+$ PyTorch

Implications

Learning convergent algorithms using automati

$\min_{\pi}$

subjec

$\pi($

Challenges for effective L2O:

1. Generalization: learning the map $x_0 \in \mathbb{R}^d \mapsto \mathbf{v}^\star \in \mathbb{R}^\infty$ is hard…
2. Implementation: how to deal with partial gradient information?

Unconstrained parametrizations of $\mathcal{L}_2$ operators (**Part 1**) + PyTorch

Main result 2★: learning with input features

$$\min_{\theta \in \mathbb{R}^n} \mathbb{E}_{\substack{f \sim \mathcal{F} \\ x_0 \sim \mathcal{X}_0}} [\mathtt{MetaLoss}(f, \mathbf{x})]$$

$$\text{subject to } x_{t+1} = x_t - \eta \nabla f(x_t) + V_t(f, x_0, \theta),$$

replace with $r_t(x_0, \theta) d_t(x_{t:0}, f(x_{t:0}), \nabla f(x_{t:0}), \theta)$

unit direction vector

ℓ_2 radius vector

Main result 2★: learning with input features

$$\min_{\theta \in \mathbb{R}^n} \mathbb{E}_{\substack{f \sim \mathcal{F} \\ x_0 \sim \mathcal{X}_0}} [\texttt{MetaLoss}(f, \mathbf{x})]$$

$$\text{subject to } x_{t+1} = x_t - \eta \nabla f(x_t) + V_t(f, x_0, \theta),$$

replace with $\underbrace{r_t(x_0, \theta)}_{\ell_2 \text{ radius vector}} \underbrace{d_t(x_{t:0}, f(x_{t:0}), \nabla f(x_{t:0}), \theta)}_{\text{unit direction vector}}$

…preserves one-to-one parametrization!

- Proof of sufficiency: by construction
- Proof of necessity: reduces to the previous case by writing $V_t(f, x_0)$ in polar coordinates

Main result 3: the case of gradients with errors

Typical scenario in machine learning: $f(x) = \sum_{i \in \text{batches}} f_i(x) \longrightarrow$ access to $\nabla f_i(x)$ only

If $\boldsymbol{\eta} \in \ell_2$ and $\|v_t\| \leq \eta_t \|\nabla f_t(x_t)\|$, $\pi_t(x_t) = -\eta_t \nabla f_t(x_t) + v_t$ converges asymptotically

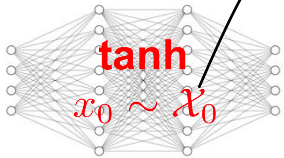
instantaneous bounds $\longrightarrow$ sufficient condition consistent with SGD

Compatibility with machine learning tasks with batch data

Experiment: training a perceptron for image classification

Training of the learned algorithm

classifier initial weights distribution



classifier prediction

train the classifier with the update

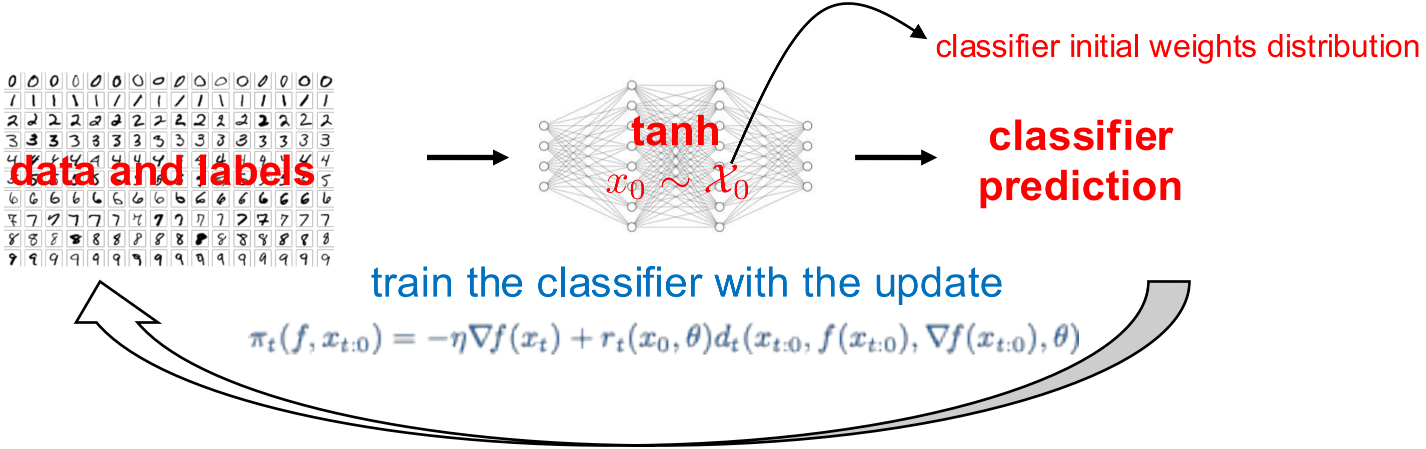
$$\pi_t(f, x_{t:0}) = -\eta\nabla f(x_t) + r_t(x_0, \theta) d_t(x_{t:0}, f(x_{t:0}), \nabla f(x_{t:0}), \theta)$$

classifier training loss

current algorithm parameters

Experiment: training a perceptron for image classification

Training of the learned algorithm

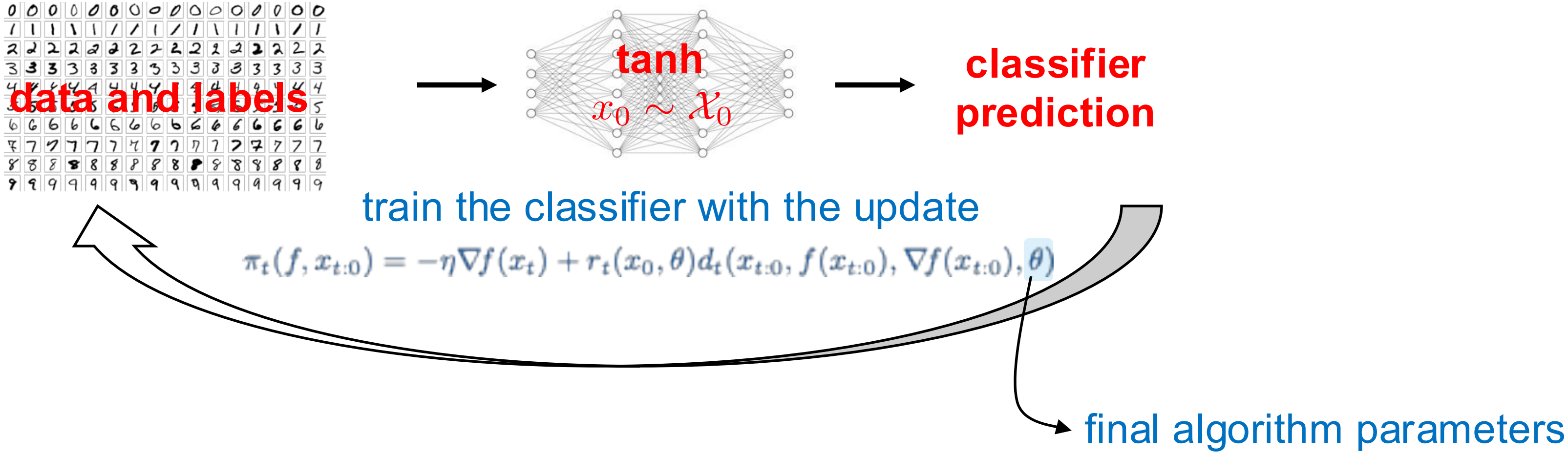


After training the classifier, we get one evaluation of $\texttt{MetaLoss}(f, \mathbf{x}, \theta)$ …

…repeat several times to approximate $\mathbb{E}_{\substack{f \sim \mathcal{F} \\ x_0 \sim \mathcal{X}_0}} [\texttt{MetaLoss}(f, \mathbf{x}, \theta)]$, then update θ

Experiment: training a perceptron for image classification

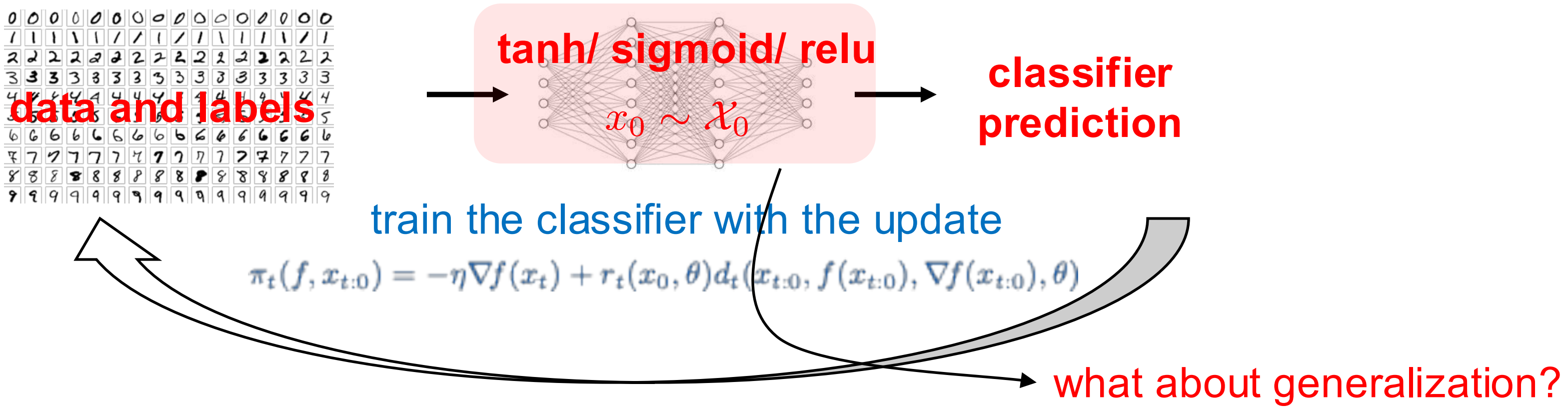
Testing of the learned algorithm: θ is kept frozen



Compare training curves $f(\mathbf{x})$ with fine-tuned classical optimizers

Experiment: training a perceptron for image classification

Testing of the learned algorithm: θ is kept frozen

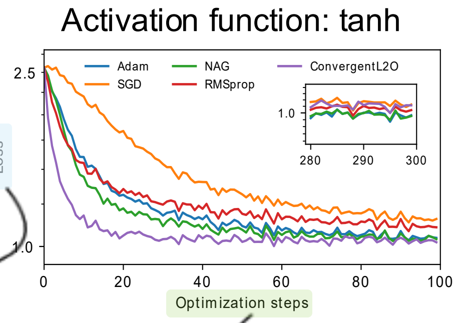


Compare training curves $f(\mathbf{x})$ with fine-tuned classical optimizers

Experiment: training a perceptron for image classification

Classifier training curves

$f(x_t)$

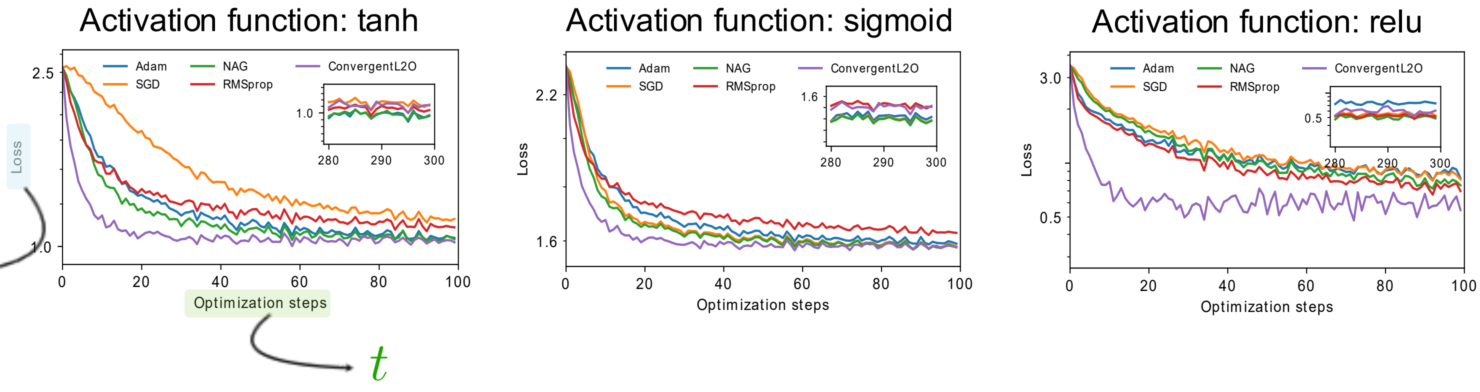


t

Experiment: training a perceptron for image classification

Classifier training curves

$f(x_t)$

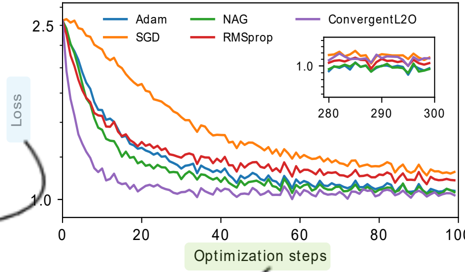


Experiment: training a perceptron for image classification

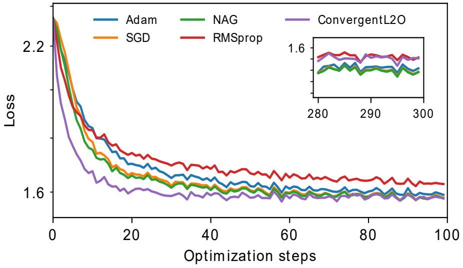
Classifier training curves

$f(x_t)$

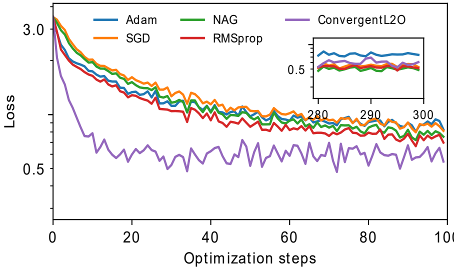
Activation function: tanh



Activation function: sigmoid



Activation function: relu



t

transient performance

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%

Step t = 300	tanh	sigmoid	ReLU
Adam	89.5 ± 0.5%	89.6 ± 0.3%	70.3 ± 12.2%
SGD	87.4 ± 0.4%	89.3 ± 0.3%	80.6 ± 8.1%
NAG	89.4 ± 0.2%	89.4 ± 0.2%	82.2 ± 7.6%
RMSprop	87.6 ± 2.1%	88.5 ± 0.4%	81.5 ± 7.5%
ConvergentL2O	88.5 ± 0.2%	88.4 ± 0.3%	87.7 ± 0.2%
LSTM	81.4 ± 0.0%	81.4 ± 0.0%	88.3 ± 0.0%

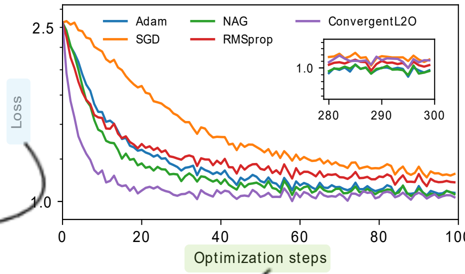
performance upon convergence

Experiment: training a perceptron for image classification

Classifier training curves

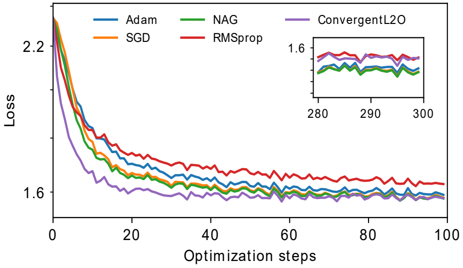
$f(x_t)$

Activation function: tanh

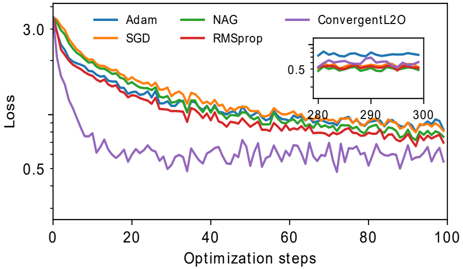


t

Activation function: sigmoid



Activation function: relu



transient performance

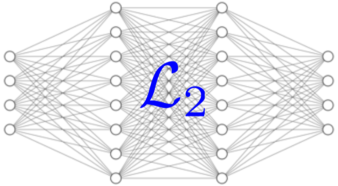
Classifier test accuracy

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%

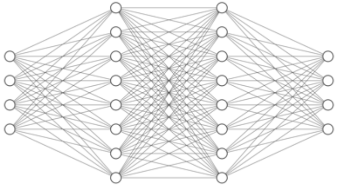
Step t = 300	tanh	sigmoid	ReLU
Adam	89.5 ± 0.5%	89.6 ± 0.3%	70.3 ± 12.2%
SGD	87.4 ± 0.4%	89.3 ± 0.3%	80.6 ± 8.1%
NAG	89.4 ± 0.2%	89.4 ± 0.2%	82.2 ± 7.6%
RMSprop	87.6 ± 2.1%	88.5 ± 0.4%	81.5 ± 7.5%
ConvergentL2O	88.5 ± 0.2%	88.4 ± 0.3%	87.7 ± 0.2%
LSTM	81.4 ± 0.0%	81.4 ± 0.0%	88.3 ± 0.0%

performance upon convergence

$$x_{t+1} = x_t - \eta\nabla f(x_t) +$$



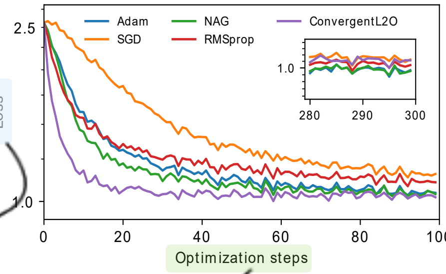
$$x_{t+1} = x_t +$$



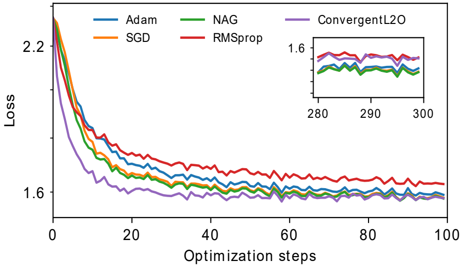
Experiment: training a perceptron for image classification

Classifier training curves

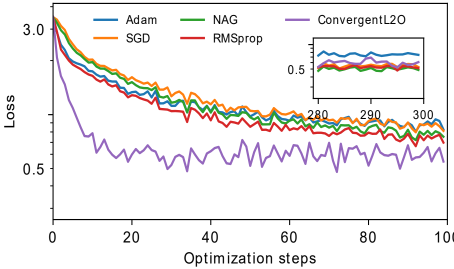
Activation function: tanh



Activation function: sigmoid



Activation function: relu



$f(x_t)$

t

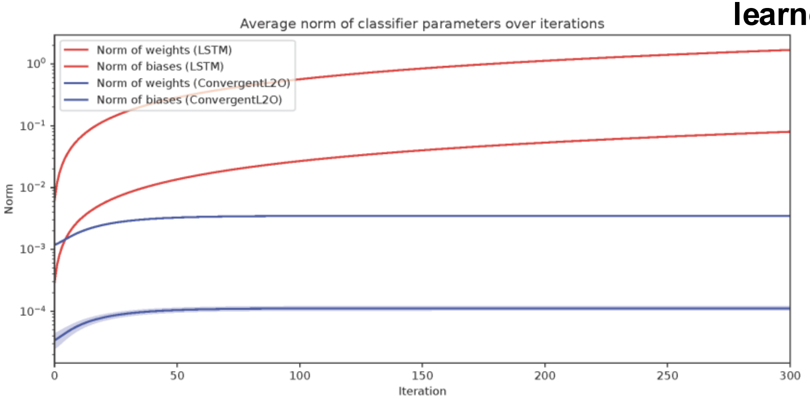
transient performance

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%

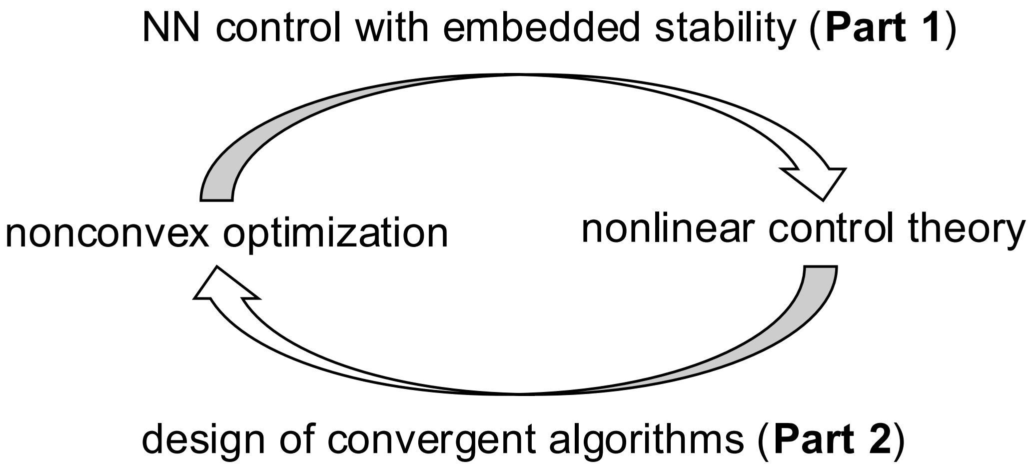
Step t = 300	tanh	sigmoid	ReLU
Adam	89.5 ± 0.5%	89.6 ± 0.3%	70.3 ± 12.2%
SGD	87.4 ± 0.4%	89.3 ± 0.3%	80.6 ± 8.1%
NAG	89.4 ± 0.2%	89.4 ± 0.2%	82.2 ± 7.6%
RMSprop	87.6 ± 2.1%	88.5 ± 0.4%	81.5 ± 7.5%
ConvergentL2O	88.5 ± 0.2%	88.4 ± 0.3%	87.7 ± 0.2%
LSTM	81.4 ± 0.0%	81.4 ± 0.0%	88.3 ± 0.0%

performance upon convergence



Without guarantees learned optimizers diverge!

Conclusions



Unified method

Use NNs to *design the closed-loop behavior directly*… not to parametrize a policy

Future work

Control

- Develop a «new» RL based on learning over stable closed-loop maps, not over policies
- *Lessons from AlphaZero*[1]*:* online NMPC *combined with* learnt feedback policy

Thank you

Optimization

- Learning to optimize with *linear/superlinear* convergence guarantees
  - Exploit monotone operator theory, e.g., strongly convex, PL, ADMM…
- Learning to optimize with constraints
- Formal transfer learning analysis[2]

$$u^{\star}(x_t) = \underset{u_0,\ldots,u_{N-1}}{\arg\min} \sum_{k=0}^{N-1} \ell\left(x_k, u_k\right) + \ell_f\left(x_N\right)$$

$$\text{subject to } x_0 = x_t\,,\ x_{k+1} = g\left(x_k, u_k\right)$$

$$x_k \in \mathcal{X}\,,\ u_k \in \mathcal{U}\,,\ x_N \in \mathcal{X}_f$$

[1] D. Bertsekas, *Lessons from AlphaZero for optimal, model predictive, and adaptive control*, Athena Scientific, 2022

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