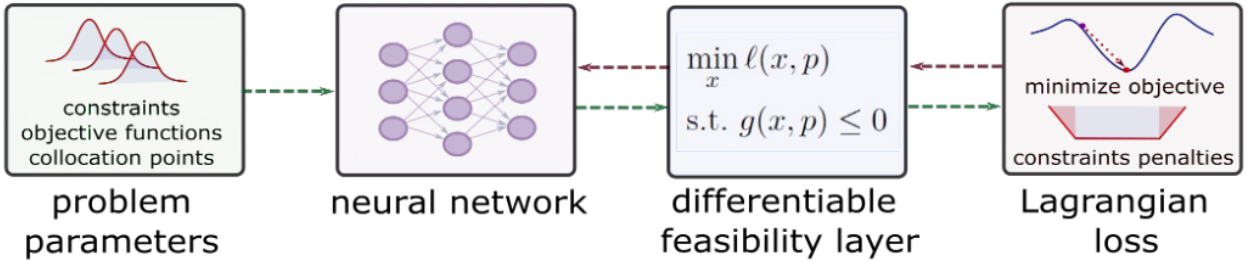


SciML Tutorial: Learning to Optimize (L2O)

Truong X. Nghiem

Associate Professor
Department of Electrical and Computer Engineering
Department of Computer Science (secondary)
School of Modeling, Simulation, and Training (secondary)
College of Engineering and Computer Science
University of Central Florida

8th Annual Learning for Dynamics & Control Conference
University of Southern California, June 17

Objective

The tutorial will show …

- What is Learning to Optimize (L2O)? What problem does it solve?
- What are major L2O approaches?
- Some L2O methods
- Demonstrations of two L2O methods for real optimization examples, using Neuromancer and JAX

L2O learns to solve parametric optimization

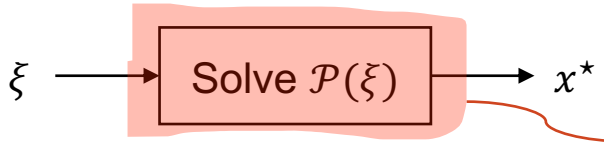
Parametric constrained optimization problem $\mathcal{P}(\xi)$:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x;\xi) \\ \text{subject to} \quad & g(x;\xi) \leq 0 \\ & h(x;\xi) = 0 \end{aligned}$$

$\xi \in \Xi$ is instance parameter → *Many applications in control, planning, scheduling, etc.*

Problem can be convex, non-convex, smooth, non-smooth, continuous, discrete, etc.

Solving $\mathcal{P}(\xi)$ for an optimal solution $x^\star$ is essentially a mapping $\pi^\star: \Xi \to X$ that maps ξ to $x^\star$



Often computationally expensive, esp. for time-critical applications

L2O accelerates solving $\mathcal{P}(\xi)$ – computing the solution mapping.

Two main ways:

- Learn to enhance existing solvers: warm-start, active constraints, penalty parameters, …
- Learn the mapping to predict solution $x^\star$ directly from ξ

How does L2O learn to solve?

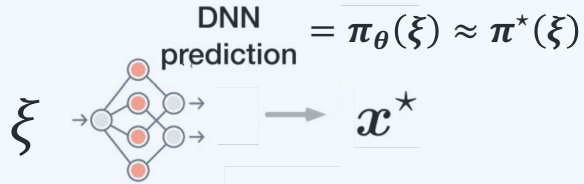
Objective: Approximate true solution mapping $\pi^\star$ with parameterized π_θ such that

$$\hat{x} = \pi_\theta(\xi) \approx \pi^\star(\xi) = x^\star, \qquad \forall \xi \in \Xi$$

while satisfying feasibility constraints $g(\hat{x}; \xi) \leq 0$ and $h(\hat{x}; \xi) = 0$.

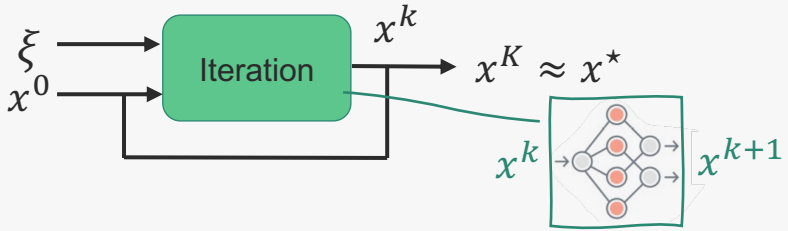
Direct mapping (end-to-end L2O):

Policy $\pi_\theta : \xi \mapsto x$ directly predicts a solution from parameters ξ (amortized optimizer).



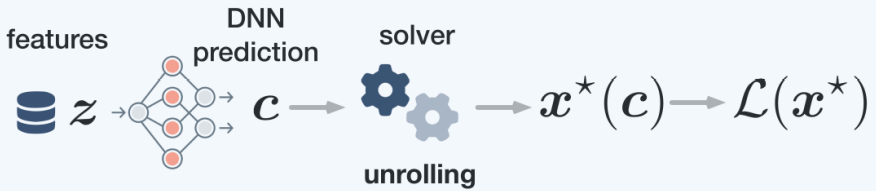
Unrolled optimizer:

Solvers are typically iterative (PGD, ADMM,…). Policy $\pi_\theta : x^k \mapsto x^{k+1}$ approximates the iteration update or an expensive part of it.



Hybrid design:

Combine direct mapping with an unrolled iterative update/refinement procedure.



J. Kotary et al., *Backpropagation of Unrolled Solvers with Folded Optimization*, IJCAI, 2023

Supervised vs Self-Supervised L2O

Supervised L2O minimizes the expected prediction error w.r.t. optimal solution by classical solver.

$$\min_{\theta} \ \mathbb{E}_{\xi\sim\mathcal{D}}\Big[\big\|\pi_\theta(\xi) - x^\star(\xi)\big\|^2\Big],$$

Loss function: $\mathcal{L}_{\text{sup}}(\theta) = \dfrac{1}{M}\sum_{i=1}^{M} \|\pi_\theta(\xi^i) - x^{\star i}\|^2$

Dataset: Labeled pairs $(\xi^i, x^{\star i})$ of true optimal solution, or true iteration update, from a classical solver.

Self-supervised L2O minimizes the expected objective subject to constraints.

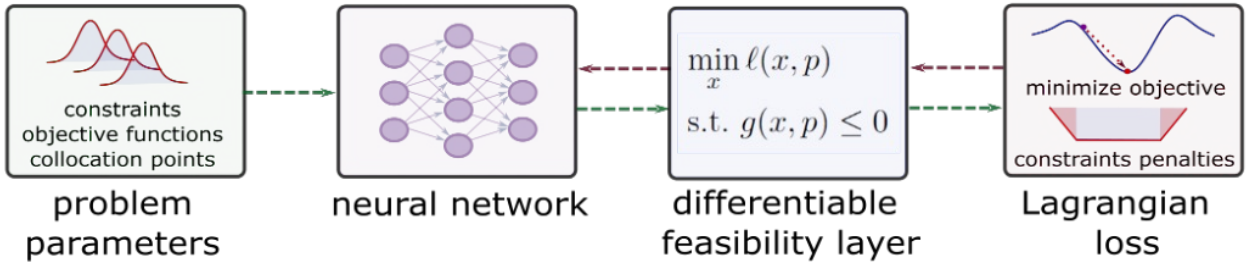
$$\begin{aligned}\min_{\theta} \ & \mathbb{E}_{\xi\sim\mathcal{D}}\Big[f\big(\pi_\theta(\xi);\xi\big)\Big] \\ \text{s.t. } & g\big(\pi_\theta(\xi);\xi\big) \le 0, \quad \text{a.s. } \xi\sim\mathcal{D}, \\ & h\big(\pi_\theta(\xi);\xi\big) = 0, \quad \text{a.s. } \xi\sim\mathcal{D}.\end{aligned}$$

Loss function: $\mathcal{L}_{\text{self}}(\theta) = \dfrac{1}{M}\sum_{i=1}^{M}\big(f(\pi_\theta(\xi^i);\xi^i) + \lambda_1\|g_+(\pi_\theta(\xi^i);\xi^i)\|^2 + \lambda_2\|h(\pi_\theta(\xi^i);\xi^i)\|^2 + \lambda_3\mathcal{R}(\pi_\theta(\xi^i),\theta)\big)$

Dataset: Problem instances ξ^i sampled from the problem parameter distribution; no labels are needed.

Differentiable Optimization

Differentiable optimization layers expose the sensitivity of optimization solutions to parameters, enabling end-to-end learning with an optimization problem inside the computation graph.



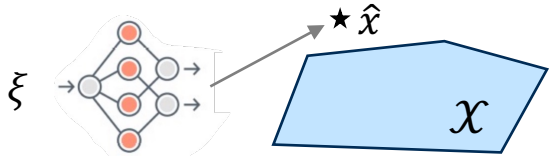
Two ways of differentiating through optimization program:

1. **Fixed-point differentiation**, based on either *unrolling iterative solvers* (backpropagation through time) or applying the *implicit function theorem (IFT)* at a fixed point.
2. **KKT-based differentiation**, which differentiates the *Karush–Kuhn–Tucker (KKT)* optimality system directly using *IFT.*

Brandon Amos, J. Zico Kolter, OptNet: Differentiable Optimization as a Layer in Neural Networks, ICML, 2017
Agrawal, A. and Amos, B. and Barratt, S. and Boyd, S. and Diamond, S. and Kolter, Z., Differentiable Convex Optimization Layers, NeurIPS 2019
James Kotary, My H. Dinh, Ferdinando Fioretto, Backpropagation of Unrolled Solvers with Folded Optimization, IJCAI, 2023
Quill Healey, Parth Nobel, Stephen Boyd, Differentiating Through a Quadratic Cone Program, 2025
https://implicit-layers-tutorial.org/differentiable_optimization/

End-to-end L2O: Feasibility

End-to-end (direct mapping) L2O often returns infeasible solution that violates constraints.



Feasibility Restoration Layer
restores feasibility by an operator $x^\star = \mathcal{F}(\hat{x})$
that maps infeasible $\hat{x} \notin \mathcal{X}$ to a feasible or approximately feasible $x^\star \in \mathcal{X}$.

The operator $\mathcal{F}$ may be realized through one of the following mechanisms:

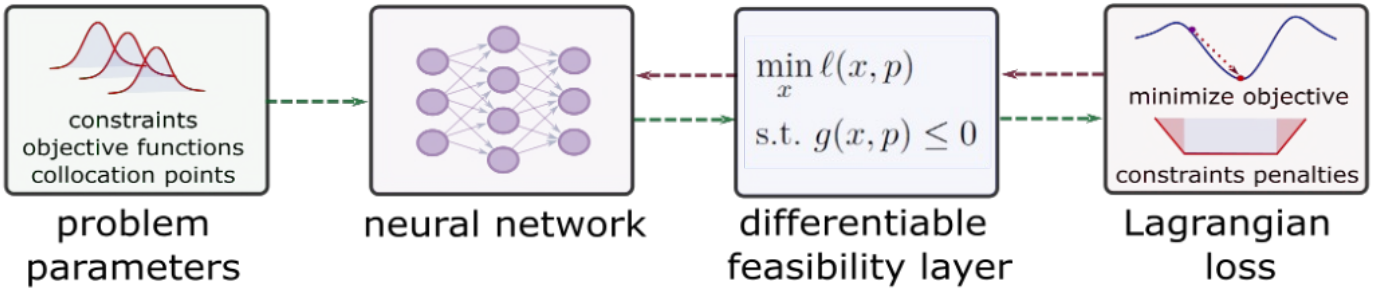
(i) **Projection operator:** Enforces hard feasibility by mapping $\hat{x}$ to the closest point $x^* \in \mathcal{X}$ that satisfies all constraints, i.e., the orthogonal projection onto the feasible set.

(ii) **Proximal operator:** $x^* = \arg\min_x \Psi(x) + \frac{1}{2}\|x - \hat{x}\|^2$, where $\Psi(x)$ is a potential encoding feasibility through indicator, penalty, barrier, or Lagrangian terms.

(iii) **Algebraic completion:** $x^* = r(\hat{x})$ such that $h(x^*) = 0$, where feasibility is restored by explicit algebraic substitution or variable completion without optimization.

(iv) **Implicit equilibrium:** $F(x^*, \hat{x}) = 0$, where F defines feasibility through the equilibrium of a fixed-point solver or KKT conditions of an optimization solver.

L2O Example in NeuroMANCER SciML Library



https://colab.research.google.com/github/SciML-L4DC2026/SciML-L4DC2026/blob/main/notebooks/L2O/Example_1_DC3/Example_1_DC3.ipynb

sciml-l4dc2026.github.io/SciML-L4DC2026

github.com/pnnl/neuromancer

NEUROMANCER
Scientific Machine Learning

Basic Unrolled L2O for ADMM

Parametric convex optimization problem, where $f(x;\xi)$ and $\mathcal{C}(\xi)$ are convex for all ξ:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x;\xi) \\ \text{subject to} \quad & x \in \mathcal{C}(\xi) \end{aligned}$$

Standard ADMM iterations:

$$\begin{aligned} x^{k+1} &= \arg\min f(x;\xi) + \frac{\rho}{2}\|z^k - x + \rho^{-1}y^k\|_2^2 \\ z^{k+1} &= \Pi_{\mathcal{C}(\xi)}(x^{k+1} - \rho^{-1}y^k) \\ y^{k+1} &= y^k + \rho(z^{k+1} - x^{k+1}) \end{aligned}$$

Primal update is often expensive: learn a surrogate $\pi_\theta(z^k, y^k, \xi) = \hat{x}^{k+1} \approx x^{k+1}$

Or learn a surrogate for both x- and z-updates $\pi_\theta(z^k, y^k, \xi) = (\hat{x}^{k+1}, \hat{z}^{k+1}) \approx (x^{k+1}, z^{k+1})$

ADMM-unrolled L2O:

$$\begin{aligned} \hat{x}^{k+1} &= \pi_\theta(z^k, y^k, \xi) \\ z^{k+1} &= \Pi_{\mathcal{C}(\xi)}(\hat{x}^{k+1} - \rho^{-1}y^k) \\ y^{k+1} &= y^k + \rho(z^{k+1} - x^{k+1}) \end{aligned}$$

Advantages:

- ✓ Learning an update step (proximal operator) is simpler than learning the optimal solution.
- ✓ Feasibility via z-update

Challenges: convergence, error propagation

Learning Moreau Envelope Framework (LEAF)

Parametric convex optimization problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & x \in \mathcal{C}(\xi) \end{aligned}$$

Primal update in ADMM-unrolled L2O:

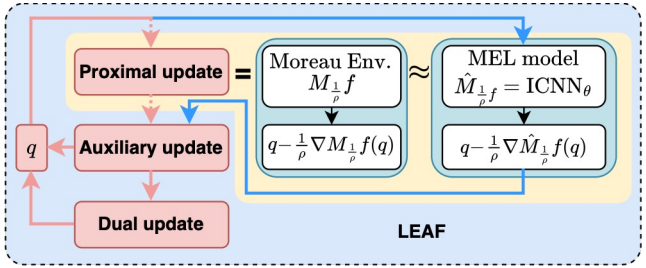
$$x^{k+1} = \arg\min f(x) + \frac{\rho}{2}\|z^k - x + \rho^{-1}y^k\|_2^2$$

Moreau envelope = optimal objective:

$$M_{1/\rho}f(z) = \min_x \left\{ f(x) + \frac{\rho}{2}\|z - x\|_2^2 \right\}$$

$M_{1/\rho}f$ is convex & bounded by f
$\nabla M_{1/\rho}f$ is ρ-Lipschitz continuous

$$x^{k+1} = (z^k + \rho^{-1}y^k) - \frac{1}{\rho}\nabla M_{1/\rho}f(z^k + \rho^{-1}y^k)$$



$$\hat{x}^{k+1} = \pi_\theta(z^k + \rho^{-1}y^k)$$

vector output, no structure

Learn surrogate $\pi_\theta^M(z) \approx M_{1/\rho}f(z)$

- Scalar output
- ICNN to enforce convexity & Lipschitz
- Boundedness

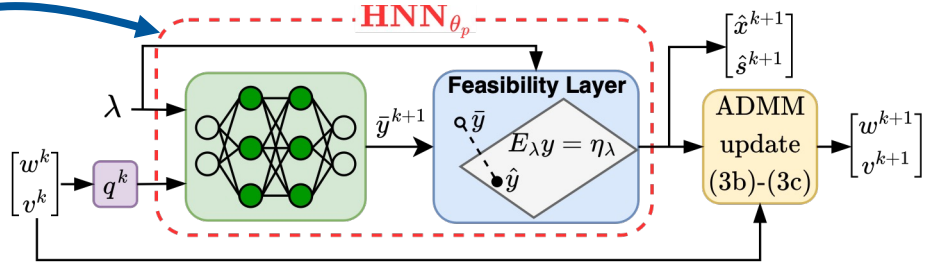
$$\hat{x}^{k+1} = (z^k + \rho^{-1}y^k) - \frac{1}{\rho}\nabla \pi_\theta^M(z^k + \rho^{-1}y^k)$$

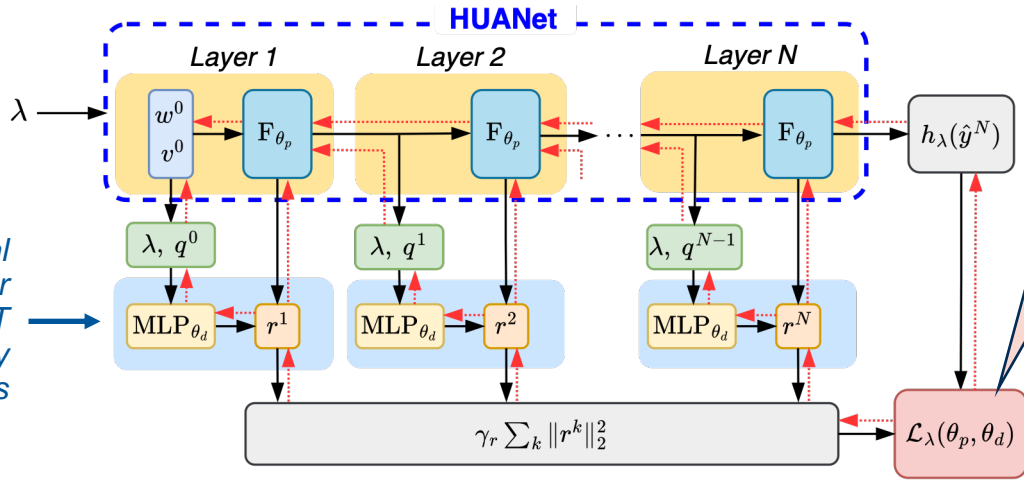
B. Nguyen et al., *"LEAF: A learning-enabled ADMM framework for accelerated convex optimization,"* 2026

Hard-Constrained Unrolled ADMM (HUANet)

Unroll ADMM iterations into NN with KKT optimality conditions, trained with self-supervised learning.

$$x^{k+1}, s^{k+1} = \arg\min_{x,s} \; f_\lambda(x) + \frac{\rho}{2}\|s - w^k + \rho^{-1}v^k\|_2^2,$$
$$\text{s.t.} \quad A_\lambda x = b_\lambda,$$
$$C_\lambda x + s = d_\lambda,$$
$$w^{k+1} = \Pi_{\geq 0}(s^{k+1} + \rho^{-1}v^k)$$
$$v^{k+1} = v^k + \rho(s^{k+1} - w^{k+1})$$





Self-supervised training loss:

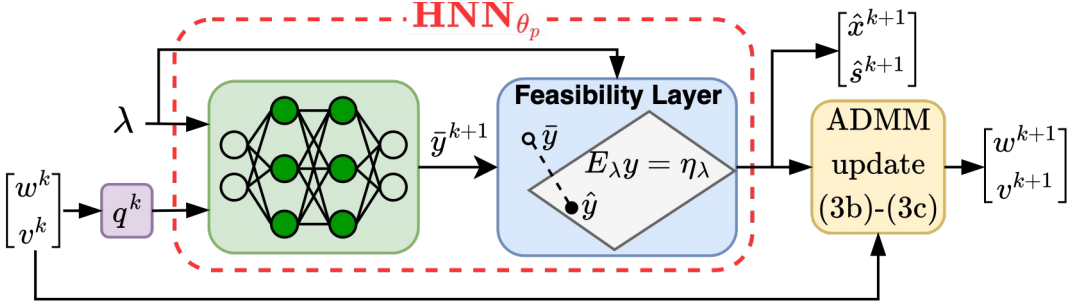
$$L_\lambda(\theta_p, \theta_d) = \frac{1}{S}\sum_{i=1}^{S}\left(h_\lambda(\hat{y}_i^N) + \gamma_r \sum_{k=1}^{N}\|r_i^k\|_2^2\right)$$

Optimal objective

Primal KKT residuals

T. Tran, et al., *"HUANet: Hard-Constrained Unrolled ADMM for Constrained Convex Optimization,"* 2026.

HUANet Example in Python + JAX



https://colab.research.google.com/github/SciML-L4DC2026/SciML-L4DC2026/blob/main/notebooks/L2O/Example_2_HUANet/HUANet_entropy.ipynb

Learning to Optimize (L2O) Summary

L2O via learning entire or part of solving optimization problem

- Learns *direct mapping* to predict solution from parameters, with differentiable feasibility restoration layer.
- *Unrolls optimizer* to learn iteration or an expensive part of it.
- Supervise & self-supervised training.
- Shifts computational burden offline, enabling fast optimization online.

- **Challenges**
  - Feasibility & feasibility restoration
  - Optimality gap
  - Convergence (for unrolled L2O)
- **Links:**
  - More L2O examples: https://github.com/pnnl/neuromancer
  - PIMLXplore: https://pimlxplore.nxtlab.org/
- **Hiring in SciML!!!** nxt@ucf.edu https://nxtlab.org/