Pacific Northwest NATIONAL LABORATORY

Neuromancer: Scientific Machine Learning Library for Modeling, Optimization, and Control

The 7th Workshop on Autonomous Energy Systems @NREL, Golden, Colorado

September 18, 2024

Ján Drgoňa
Rahul Birmiwal, Bruno Jacob, Draguna Vrabie

U.S. DEPARTMENT OF ENERGY BATTELLE
PNNL is operated by Battelle for the U.S. Department of Energy

NEUROMANCER
Scientific Machine Learning

Scientific Machine Learning (SciML)

What?

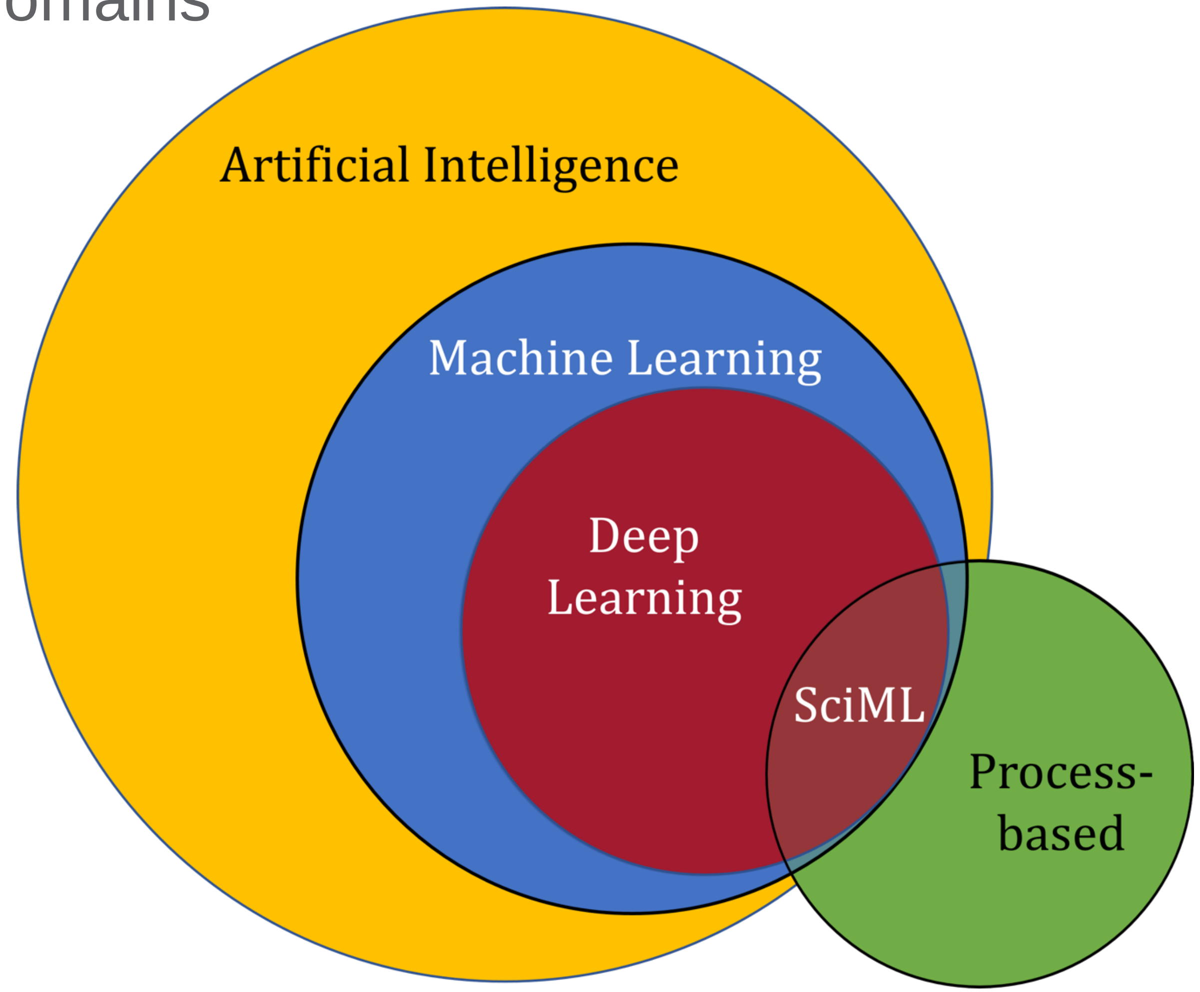
- SciML systematically integrates ML methods with mathematical models and algorithms developed in various scientific and engineering domains

Why?

- Scientific applications are governed by fundamental principles and physical constraints
- Purely data-driven “black box” ML methods cannot satisfy underlying physics

How?

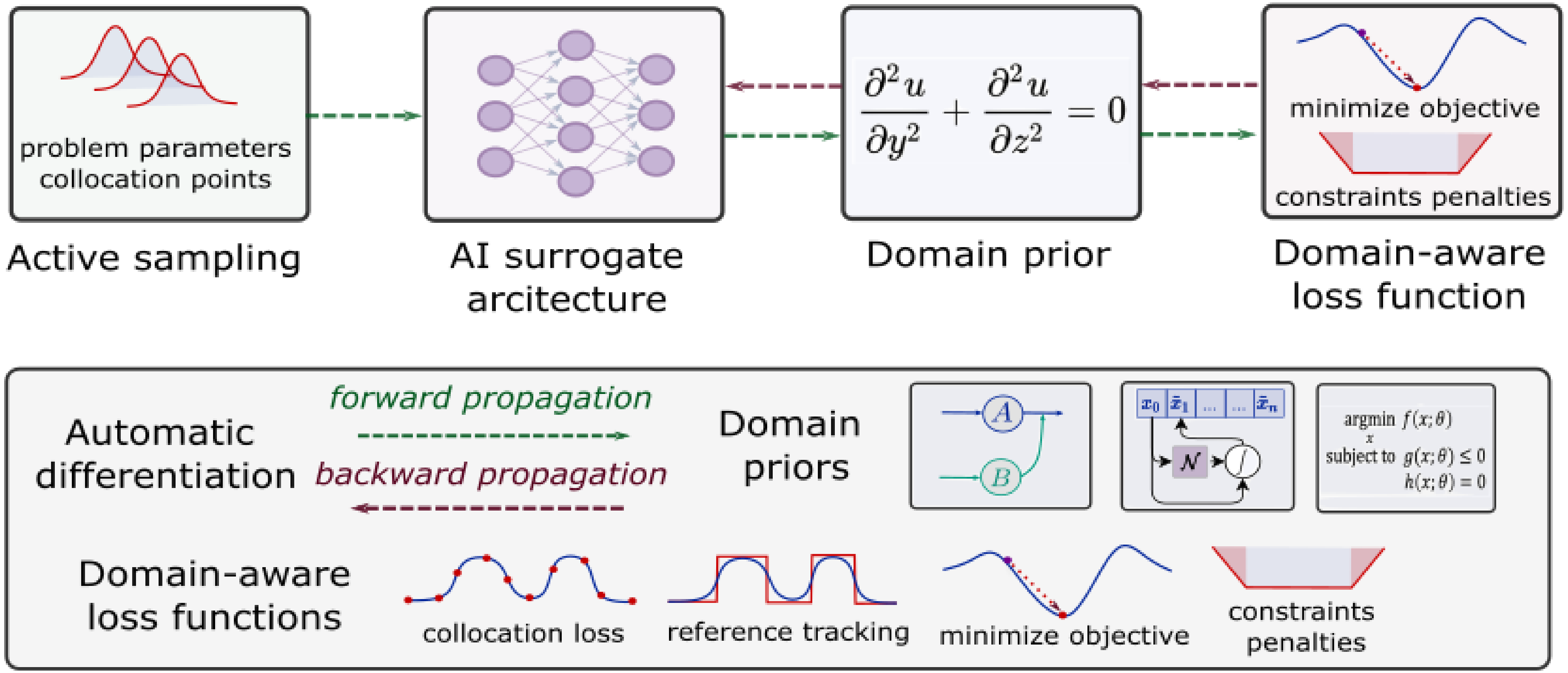
- Leverage **automatic differentiation** used in learning for modeling, optimization, and control



Karniadakis, G.E., Kevrekidis, I.G., Lu, L. et al. Physics-informed machine learning. Nat Rev Phys 3, 422–440, 2021.

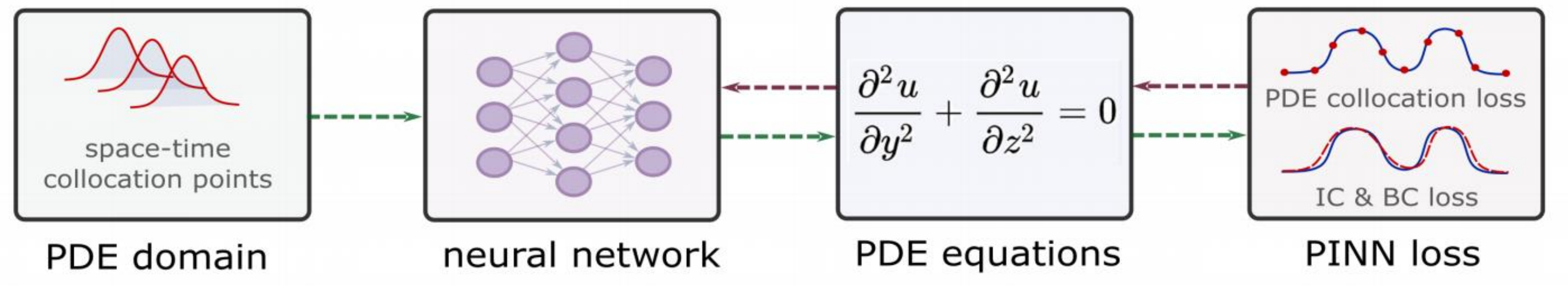
Image source: https://sciml.wur.nl/reviews/sciml/sciml.html

Landscape of SciML Methods

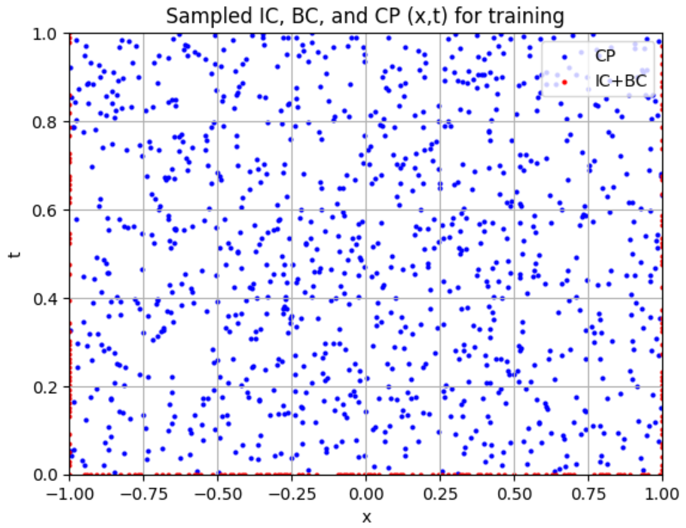


Karniadakis, G.E., Kevrekidis, I.G., Lu, L. et al. Physics-informed machine learning. Nat Rev Phys 3, 2021.
Thiyagalingam, J., Shankar, M., Fox, G. et al. Scientific machine learning benchmarks. Nature Reviews Physics 4, 413–420, 2022.
Nghiem T., Drgona J., et al. Physics-Informed Machine Learning for Modeling and Control of Dynamical Systems, ACC, 2023.

Learning to Solve Differential Equations with Physics-Informed Neural Networks (PINNs)



Dataset: collocation points in the spatio-temporal coordinates.



Architecture: PDE equations solved with **neural network** via automatic differentiation.

$$\hat{y} = NN_\theta(x,t)$$

$$f_{\mathrm{PINN}}(t,x) = \left(\frac{\partial NN_\theta}{\partial t} - \frac{\partial^2 NN_\theta}{\partial x^2}\right) + e^{-t}(sin(\pi x) - \pi^2 sin(\pi x))$$

Loss function: minimizing PDE equation, initial and boundary condition residuals.

$$\ell_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f_{\mathrm{PINN}}(t_f^i, x_f^i)|^2$$

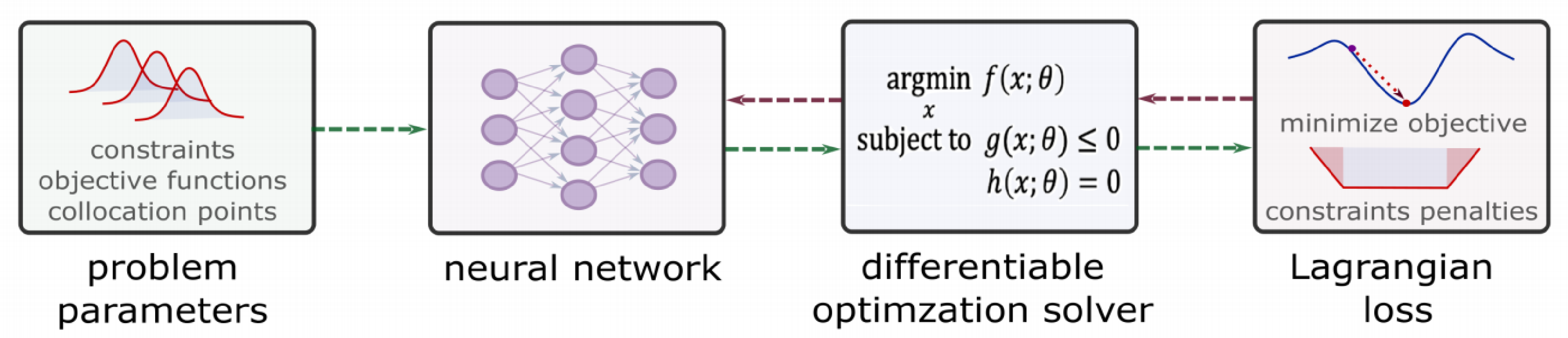
$$\ell_u = \frac{1}{N_u}\sum_{i=1}^{N_u} |y(t_u^i, x_u^i) - NN_\theta(t_u^i, x_u^i)|^2$$

$$\ell_{\mathrm{PINN}} = \ell_f + \ell_u$$

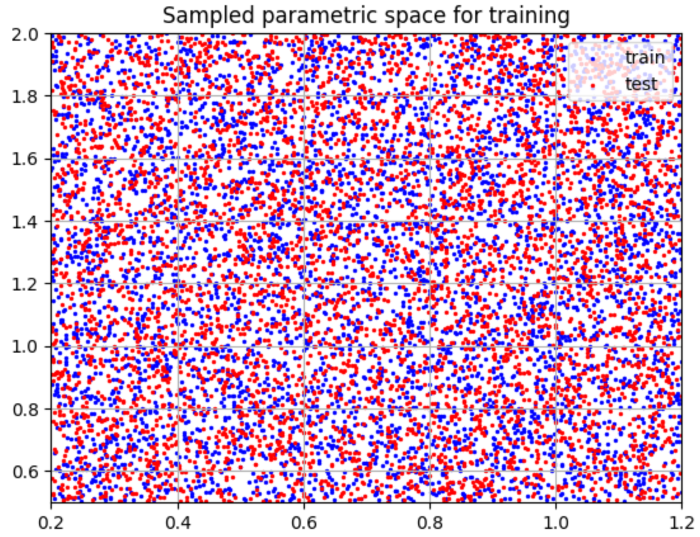
https://github.com/pnnl/neuromancer/blob/master/examples/PDEs/Part_2_PINN_BurgersEquation.ipynb

M. Raissi, et al., *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics*, 2019

Learning to Optimize (L2O) with Constraints



Dataset: collocation points in the parametric space.



Architecture: **differentiable optimization solver** with **neural network** surrogate.

$$
\begin{aligned}
\text{minimize}_{\theta} \quad & f(x,\xi) \\
\text{subject to} \quad & g(x,\xi) \leq 0 \\
& x = NN_{\theta}(\xi)
\end{aligned}
$$

$$\hat{x} = \text{proj}_{g(x,\xi)\leq 0}(x,\xi)$$

Loss function: minimizing objective function and constraints penalties.

$$\ell_f = \frac{1}{m}\sum_{i=1}^{m} |f(x^i,\xi^i)|^2$$

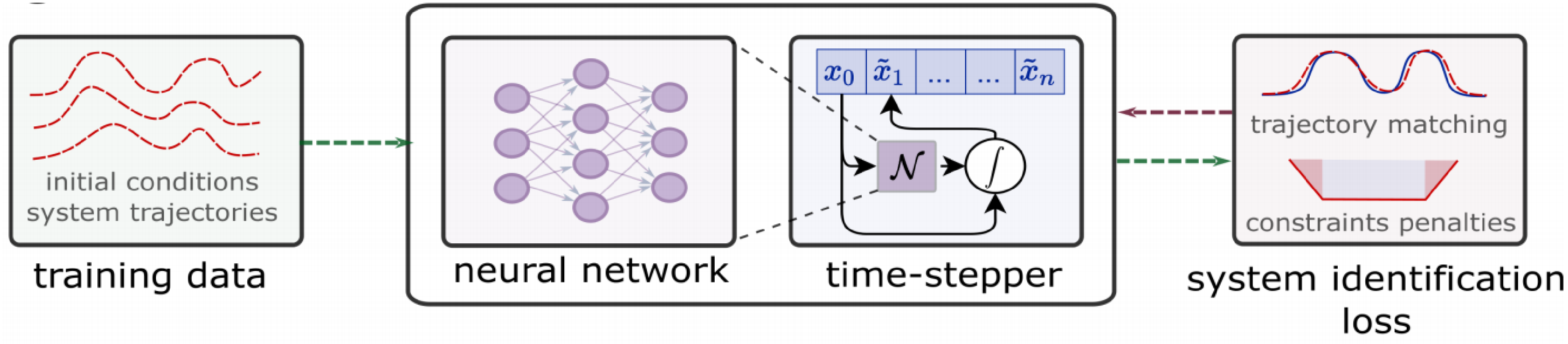
$$\ell_g = \frac{1}{m}\sum_{i=1}^{m} |\text{RELU}(g(x^i,\xi^i))|^2$$

$$\ell_{L2O} = \ell_f + \ell_g$$

https://github.com/pnnl/neuromancer/blob/master/examples/parametric_programming/Part_1_basics.ipynb

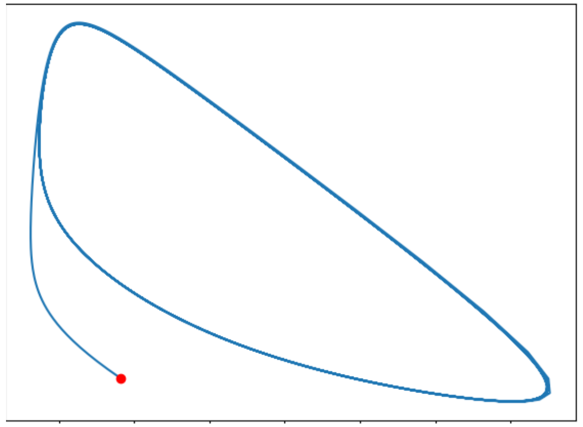
A. Agrawal, et al., *Differentiable Convex Optimization Layers*, 2019
P. Donti, et al., *DC3: A learning method for optimization with hard constraints*, 2021

Learning to Model (L2M) with Neural Operators



Dataset: time-series of states, inputs, and disturbances tuples.

$$\hat{X} = [\hat{x}_0^i, \ldots, \hat{x}_N^i], \; i \in [1, \ldots, m]$$



Architecture: differentiable ODE solver with **neural network** model.

$$x_{k+1} = \text{ODESolve}(NN_\theta(x_k))$$

Architecture: Koopman operator with **neural network** basis functions.

$$y_k = NN_\theta(x_k)$$
$$y_{k+1} = K_\theta(y_k)$$
$$x_{k+1} = NN_\theta^{-1}(y_{k+1})$$

Loss function: trajectory matching, regularizations, and constraints penalties.

$$\ell_1 = \sum_{i=1}^{m} \sum_{k=1}^{N} Q_x ||x_k^i - \hat{x}_k^i||_2^2$$

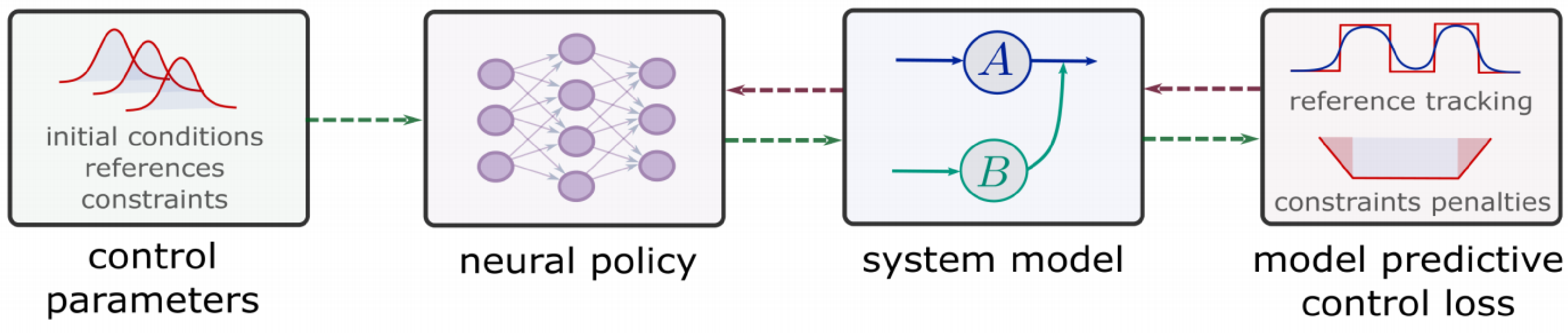
$$\ell_2 = \sum_{i=1}^{m} \sum_{k=1}^{N-1} Q_{dx} ||\Delta x_k^i - \Delta \hat{x}_k^i||_2^2$$

$$\ell_{L2M} = \ell_1 + \ell_2$$

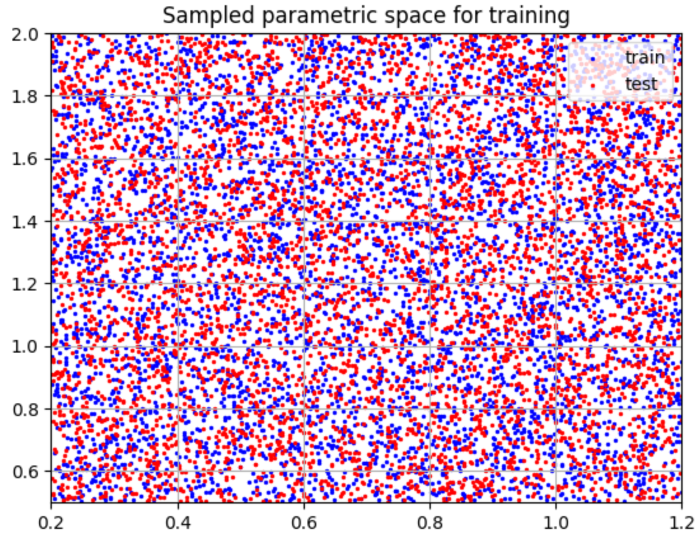
https://github.com/pnnl/neuromancer/blob/master/examples/ODEs/Part_1_NODE.ipynb

R. T. Q. Chen, et al., *Neural Ordinary Differential Equations*, 2019
B. Lusch, et al., *Deep learning for universal linear embeddings of nonlinear dynamics*, 2018

Learning to Control (L2C) with Differentiable System Models



Dataset: collocation points in the control parametric space.



Architecture: **differentiable model** with **neural network** control policy.

$$x_{k+1} = \text{ODESolve}(f(x_k, u_k))$$
$$u_k = NN_\theta(x_k, \xi_k)$$
$$g(x_k, u_k, \xi_k) \leq 0$$
$$x_0 \sim \mathcal{P}_{x_0}$$
$$\xi_k \sim \mathcal{P}_\xi$$

Loss function: reference tracking, constraints and terminal penalties.

$$\ell_1 = \sum_{i=1}^{m} \sum_{k=1}^{N-1} Q_x ||x_k^i - r_k^i||_2^2$$

$$\ell_2 = \sum_{i=1}^{m} \sum_{k=1}^{N-1} Q_g ||\text{RELU}(g(x_k^i, u_k^i, \xi_k^i)||_2^2$$

$$\ell_{L2C} = \ell_1 + \ell_2$$

https://github.com/pnnl/neuromancer/blob/master/examples/control/Part_3_ref_tracking_ODE.ipynb

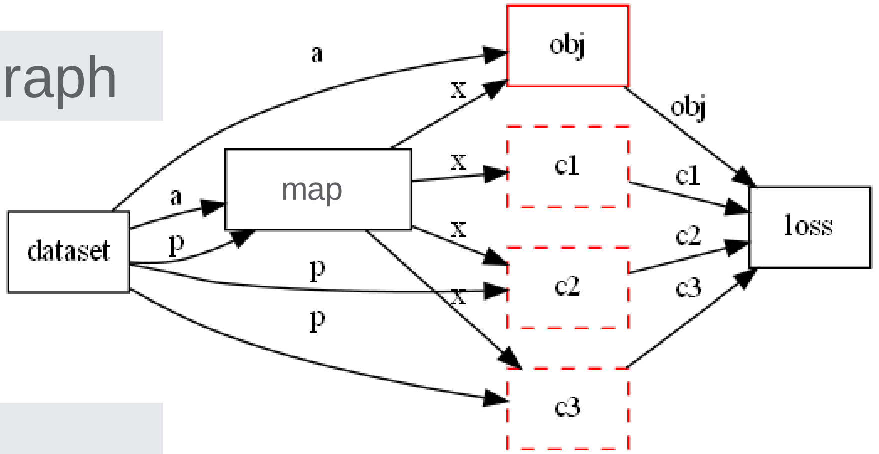
Jan Drgona, et al., *Differentiable Predictive Control: An MPC Alternative for Unknown Nonlinear Systems using Constrained Deep Learning*, Journal of Process Control, 2022

NeuroMANCER Scientific Machine Learning Library

1. Mathematical formulation

$$\min_{\Theta}\ (1-\mathbf{x})^2 + \boldsymbol{p}(\mathbf{y}-\mathbf{x}^2)^2$$
$$\text{s.t. } (\boldsymbol{p}/2)^2 \leq \mathbf{x}^2 + \mathbf{y}^2 \leq \boldsymbol{p}^2,\ \ \mathbf{x} \geq \mathbf{y}$$
$$\mathbf{x} = \boldsymbol{\pi}_{\Theta}(\boldsymbol{p})$$

3. Problem graph

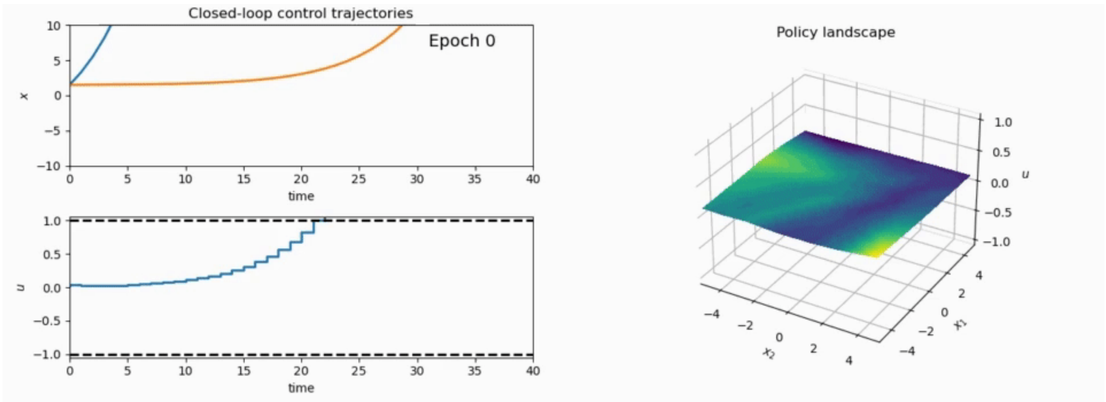


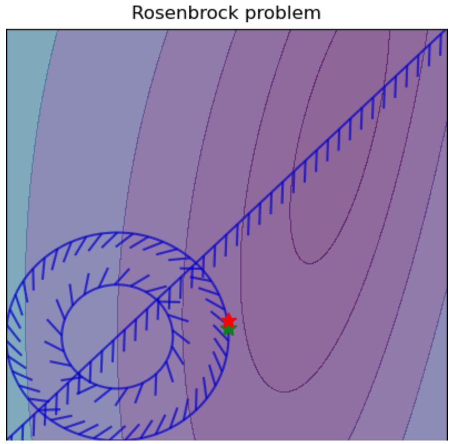
2. Python code interface

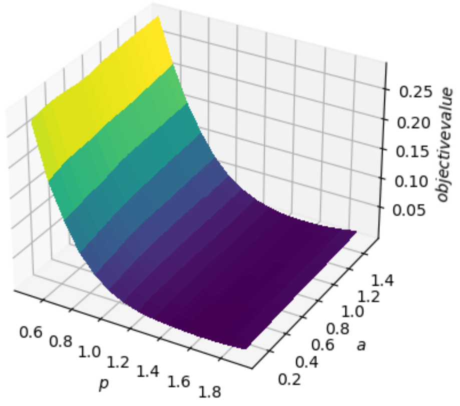
```
import neuromancer as nm

p = nm.variable(‘p’)
x = nm.variable(‘x’)
y = nm.variable('y’)

obj = ((1-x)**2 + p*(y-x**2)**2).minimize(weight=1.0, name='obj’)
c1 = (p/2)**2 <= x**2 + y**2
c2 = x**2 + y**2 <= p**2
c3 = x >= y

net = nm.MLP(insize=2, outsize=2, hsizes=[80]*4)
map = nm.Node(net, input_keys=['p’], output_keys=[‘x’,‘y’])

loss = nm.PenaltyLoss([obj], [c1, c2, c3])
problem = nm.Problem([map], loss)
optimizer = torch.optim.AdamW(problem.parameters())
trainer = nm.Trainer(problem,data,optimizer)
best_model = trainer.train()
```

PyTorch

4. Results







NeuroMANCER Scientific Machine Learning Library

- **Open-source scientific machine learning (SciML)** toolbox in Pytorch integrating deep learning, constrained optimization, and physics-based modeling
  - Learning to optimize
  - Learning to control
  - Nonlinear system identification
  - Physics-informed neural networks

github.com/pnnl/neuromancer

www.youtube.com/@neuromancer_SciML

drgona.github.io

Metric Learning to Accelerate Convergence of Operator Splitting Methods

Parametric programming setting:

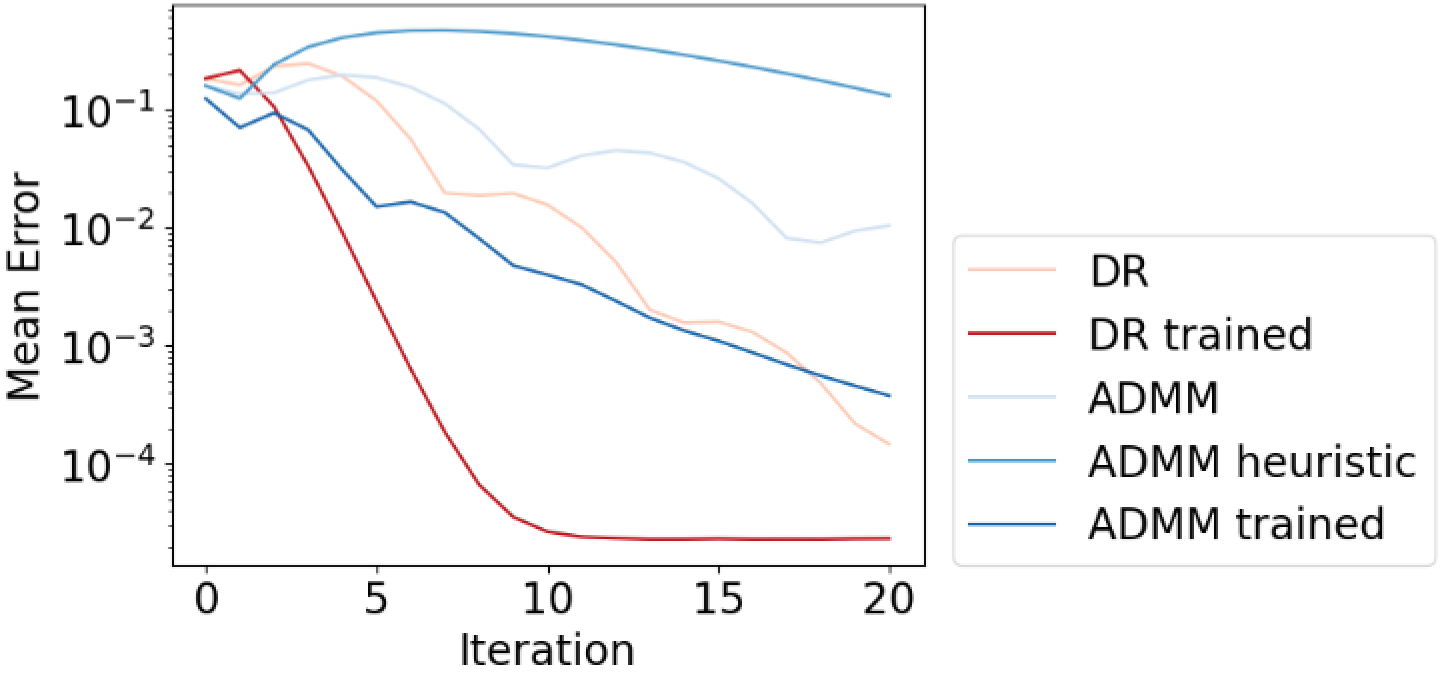
$$x^\star(p) = \underset{x \in \mathbb{R}^n}{\arg\min}\ f_p(x) + g_p(x)$$

Douglas-Rachford splitting (DR) algorithm:

$$y_k = \mathrm{prox}_\gamma g(x_k)\,,$$
$$z_k = \mathrm{prox}_\gamma f(2y_k - x_k)\,,$$
$$x_{k+1} = x_k + z_k - y_k\,,$$

$$\mathrm{prox}_\gamma f(x) = \underset{z \in \mathbb{R}^n}{\arg\min}\ f(z) + \frac{1}{\gamma}||x - z||_M^2$$

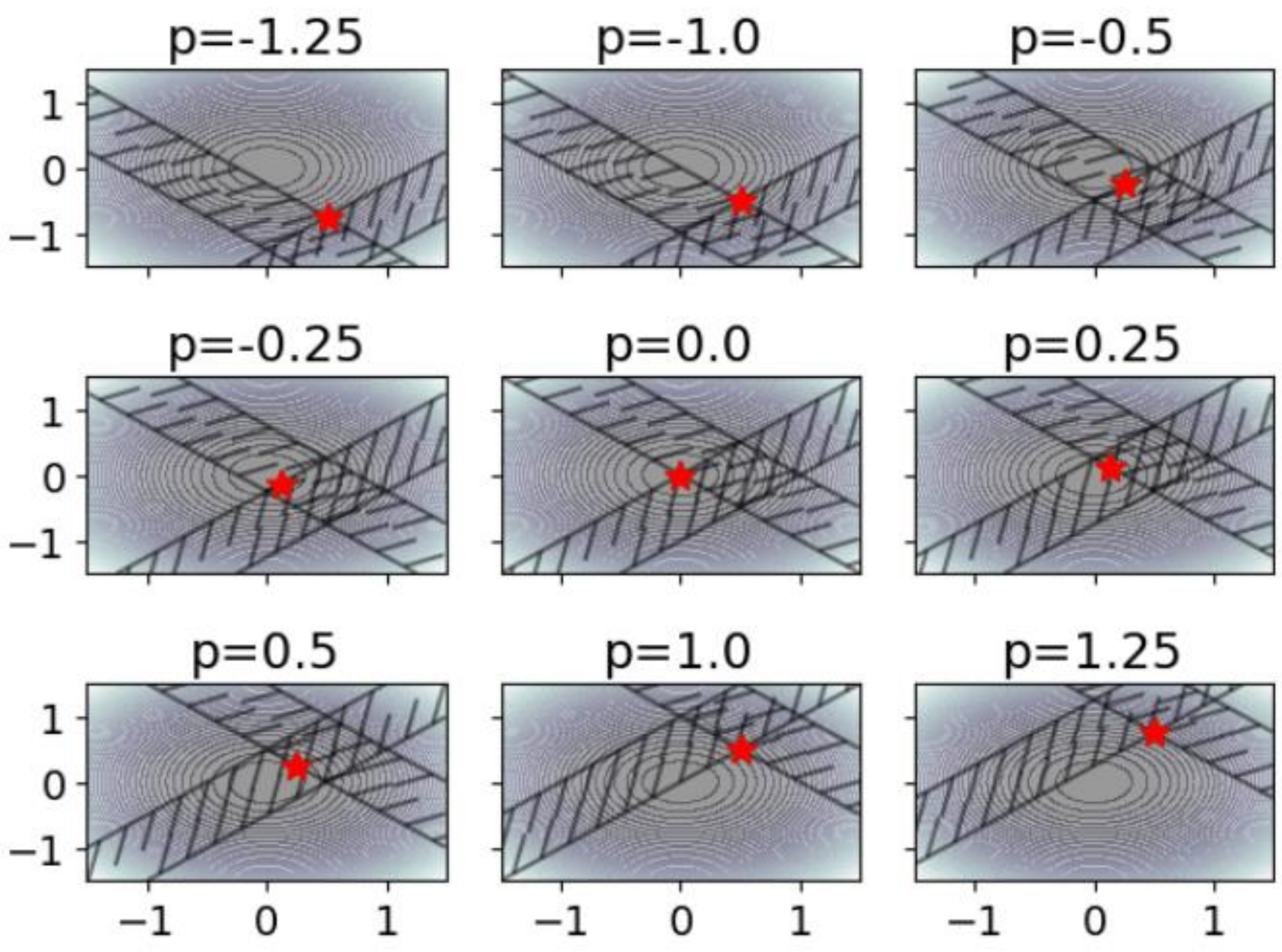
Idea: Train neural network to optimize the metric as a function of problem parameters: $M = \mathcal{N}_\omega(p)$.



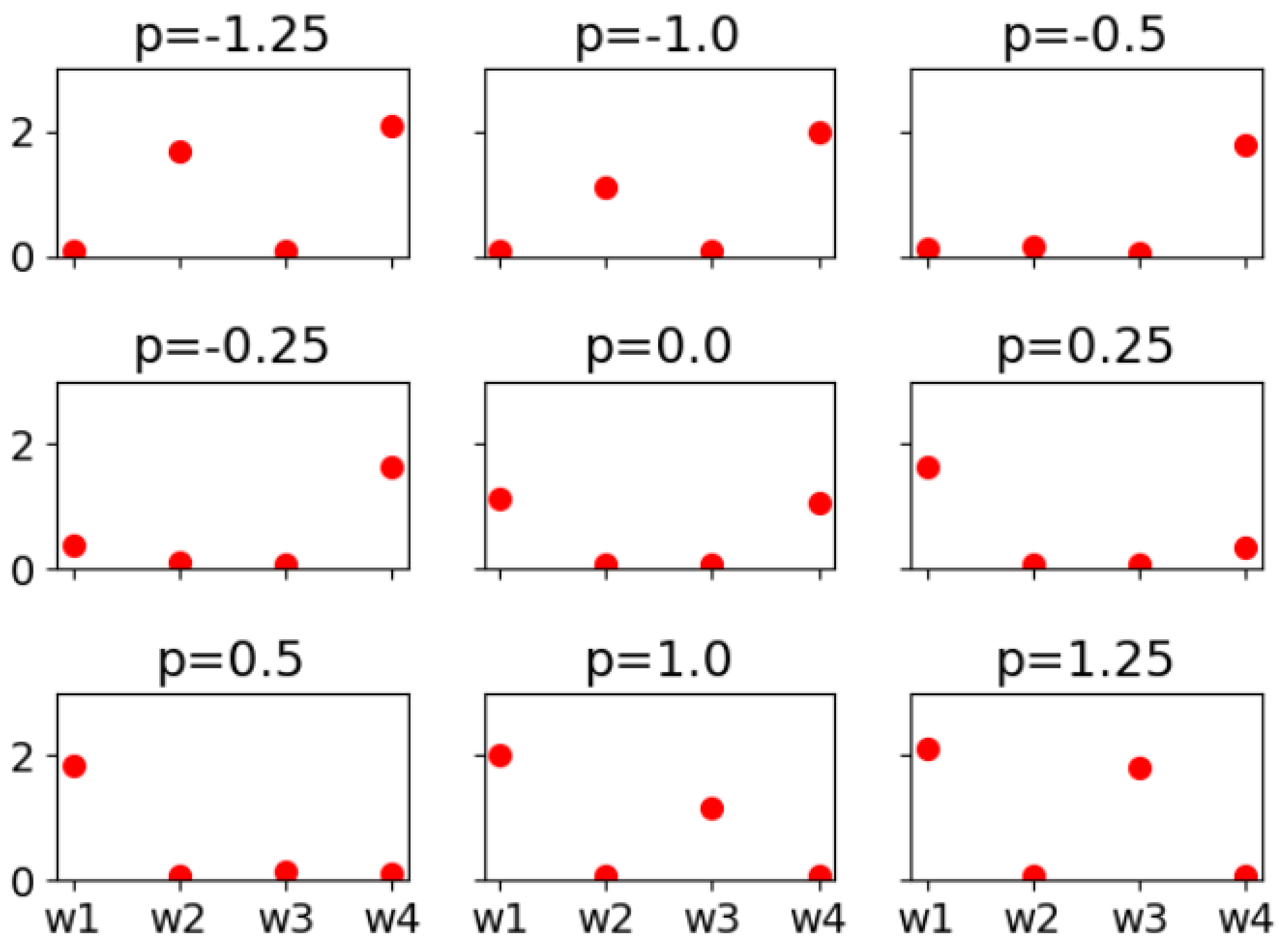
We can accelerate convergence of DR and ADMM algorithms via end-to-end metric learning.

Ethan King, James Kotary, Ferdinando Fioretto, Jan Drgona, **Metric Learning to Accelerate Convergence of Operator Splitting Methods for Differentiable Parametric Programming**, CDC 2024.

Metric Learning is a Form of Active Set Prediction



(a) Optimal Solution



(b) Metric weight for each slack variable

Ethan King, James Kotary, Ferdinando Fioretto, Jan Drgona, **Metric Learning to Accelerate Convergence of Operator Splitting Methods for Differentiable Parametric Programming**, CDC 2024.

NeuroMANCER Scientific Machine Learning Library

github.com/pnnl/neuromancer

- ☆ PyTorch SciML toolkit
- ☆ User-friendly API
- ☆ Interactive tutorials
- ☆ Community engagement

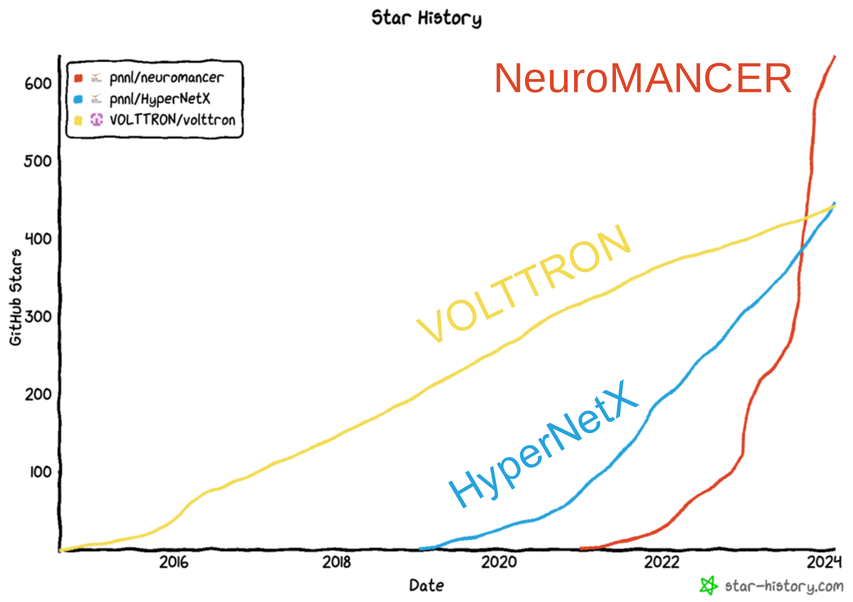
HOME » PUBLICATIONS & REPORTS

February 20, 2024 | Article

NeuroMANCER Shoots for the Stars

This deep learning framework is PNNL's most popular repository on GitHub

Sarah Wong, PNNL



Active Core Team Members


Ján Drgoňa


Rahul Birmiwal


Bruno Jacob


Draguna Vrabie

Past Core Team Members


Aaron Tuor


Madelyn Shapiro


James Koch

Hiring!

Starting a new group at the **Department of Civil & Systems Engineering at Johns Hopkins University**

- Multiple PhD and postdoc opportunities
- Starting January 2025
- Focus on Scientific Machine Learning for dynamics, optimization, and control
- Applications in energy systems

drgona.github.io