Reformulating Nonlinear Projection Equations as Neural Network Learning

Dawen Wu

Centre national de la recherche scientifique (CNRS)
National University of Singapre (NUS)

01/07/2025

Table of Contents

Table of Contents

Problem Definition

Definition (Nonlinear projection equation or NPE)

A nonlinear projection equation is to find a vector $x^* \in \Omega$ that solves

$$P_\Omega(x^* - G(x^*)) = x^*, \tag{1}$$

where $G : \mathbb{R}^n \to \mathbb{R}^n$ is a nonlinear function, and $\Omega \subset \mathbb{R}^n$ is a feasible set. $P_\Omega : \mathbb{R}^n \to \Omega$ is a projection function that maps a point $z \in \mathbb{R}^n$ onto Ω, defined as

$$P_\Omega(z) = \arg\min_{x \in \Omega} \|z - x\|_2. \tag{2}$$

The objective is to find $x^* \in \mathbb{R}^n$ that solves (1).

Assumption

Consider the problem NPE(Ω, G), with the following properties:

- *The feasible set Ω is a* box-constrained, spherical-constrained, affined-constrained *set*
- *The function $G(\cdot)$ is* locally Lipschitz continuous. *on Ω.*
- *Jacobian $\nabla G(x)$ for $x \in \Omega$ is* positive semi-definite.

Why Study NPEs

Many constrained optimization problems can be reformulated as NPEs. [1]

Proposition (NCP, Harker & Pang, 1990)

Let $\Omega \subset \mathbb{R}^n$ be a nonempty closed convex set. Then x^* solves the nonlinear complementarity problem

$$G(x^*) \geq 0, \quad x^* \geq 0, \quad G(x^*)^T x^* = 0. \tag{3}$$

if and only if x^* solves $NPE(\mathbb{R}^n_+, G)$, where $\mathbb{R}^n_+ = \{x \in \mathbb{R}^n | x \geq \mathbf{0}\}$ represents the set of non-negative real vectors.

Proposition (VI, Harker & Pang, 1990)

Let $\Omega \subset \mathbb{R}^n$ be a nonempty closed convex set. Then x^* solves the variational inequality

$$(x - x^*)^T G(x^*) \geq 0, \quad x \in \Omega. \tag{4}$$

if and only if x^* solves $NPE(\Omega, G)$.

[1] Harker, P. T., & Pang, J. S. (1990). Finite-dimensional variational inequality and nonlinear complementarity problems: a survey of theory, algorithms and applications. Mathematical programming, 48(1-3), 161-220.

Example

Example

A standard convex optimization problem is formulated as follows:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \end{aligned} \tag{5}$$

where the objective function $f : \mathbb{R}^n \to \mathbb{R}$ and the constraint functions $g_i : \mathbb{R}^n \to \mathbb{R}$ are convex. This problem can be reformulated as an NPE.

The Karush–Kuhn–Tucker conditions (KKT conditions) are

$$\begin{aligned} \nabla f(x) + u \nabla g(x) = 0, \\ g(x) \leq 0, \quad u \geq 0, \quad ug(x) = 0, \end{aligned} \tag{6}$$

The KKT conditions are equivalent to the following equation system:

$$\begin{aligned} \nabla f(x) + \nabla g(x)^T (u + g(x))^+ = 0 \\ (u + g(x))^+ - u = 0 \end{aligned} \tag{7}$$

Table of Contents

Tool 1: Neurodynamic Optimization

Xia & Feng (2007) proposed the following ODE method to solve NPEs: [2] [3]

ODE system: Models the $NPE(\Omega, G)$ as a first-order autonomous ODE system:

$$\frac{dy}{dt} = -G\left(P_\Omega(y)\right) + P_\Omega(y) - y, \tag{8}$$

where the Ω, $G(\cdot)$ and $P_\Omega(\cdot)$ are consistent with the target NPE problem.
We simplify it as $\frac{dy}{dt} = \Phi(y)$, where

$$\Phi(y) = -G\left(P_\Omega(y)\right) + P_\Omega(y) - y. \tag{9}$$

Solution of ODE:

- Optimization is finding a vector. ODE is finding a function.
- $y : \mathbb{R} \to \mathbb{R}^n$ is called a state solution if its derivative satisfies $\frac{dy}{dt} = \Phi(y)$.
- We denote $y^* \in \mathbb{R}^n$ as an equilibrium point if $\Phi(y^*) = 0$

[2] Hopfield J J, Tank D W. Computing with neural circuits: A model[J]. Science, 1986, 233(4764): 625-633.

[3] Xia, Youshen, and Gang Feng. "A new neural network for solving nonlinear projection equations." Neural Networks 20.5 (2007): 577-589.

Tool 1: Neurodynamic Optimization

Lemma (State Solution Existence)

If $G(\cdot)$ is locally Lipschitz continuous on the feasible set Ω, then for each initial point $y_0 \in \mathbb{R}^n$ there exists a continuous and unique solution trajectory for the ODE system $\frac{dy}{dt} = \Phi(y)$.

Lemma (Equilibrium Existence)

$\nabla\Phi(y)$ is positive semi-definite for any y.

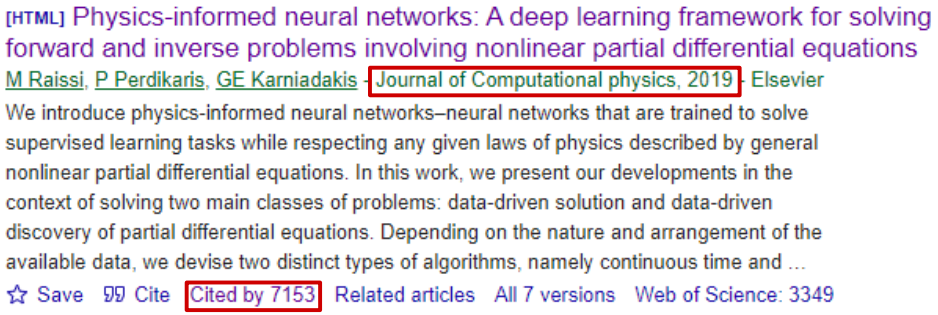
Theorem (Xia & Feng, *Neural Networks*, 2007)

Consider the problem $NPE(\Omega, G)$, with the Assumptions hold. Given any initial condition, $y(t_0) = y_0$, the state solution of the ODE system converges to the optimal solution of $NPE(\Omega, G)$ as time t goes to infinity, i.e,

$$\lim_{t\to\infty} y(t) = x^*, \tag{10}$$

where x^* is an optimal solution of $NPE(\Omega, G)$.

Tool 2: Physics-Informed Neural Network (PINN)


[HTML] Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations

M Raissi, P Perdikaris, GE Karniadakis - Journal of Computational physics, 2019 - Elsevier

We introduce physics-informed neural networks–neural networks that are trained to solve supervised learning tasks while respecting any given laws of physics described by general nonlinear partial differential equations. In this work, we present our developments in the context of solving two main classes of problems: data-driven solution and data-driven discovery of partial differential equations. Depending on the nature and arrangement of the available data, we devise two distinct types of algorithms, namely continuous time and ...

☆ Save 99 Cite Cited by 7153 Related articles All 7 versions Web of Science: 3349

Little story:

- Nov. 2020, Start of Ph.D., 800 citations.
- Nov. 2023, End of Ph.D., 7000 citations.

- Consider PDE(s) in general form:

$$u_t + \mathcal{N}[u] = 0, \quad x \in \Omega \subseteq \mathbb{R}^n, \quad t \in [0, T]; \quad (\cdot)_t = \frac{\partial(\cdot)}{\partial t}, \tag{11}$$

  where $u(t, x)$ denotes the solution, $\mathcal{N}[\cdot]$ is a nonlinear operator.
- The above setup covers a wide range of PDEs in math, physics, engineering. E.g., Burger's equation in 2D

$$\mathcal{N}[u] = \lambda_1 u_{xx} - \lambda_2 u_x; \quad (\cdot)_x = \frac{\partial(\cdot)}{\partial x}, \quad (\cdot)_{xx} = \frac{\partial^2(\cdot)}{\partial x^2} \tag{12}$$

- PINNs aim to use Neural network $\hat{u}_\theta(t, x)$ to solve the PDE

Tool 2: Physics-Informed Neural Network (PINN)

- $\hat{u}_\theta(t, x)$ is a neural network with trainable parameters θ.

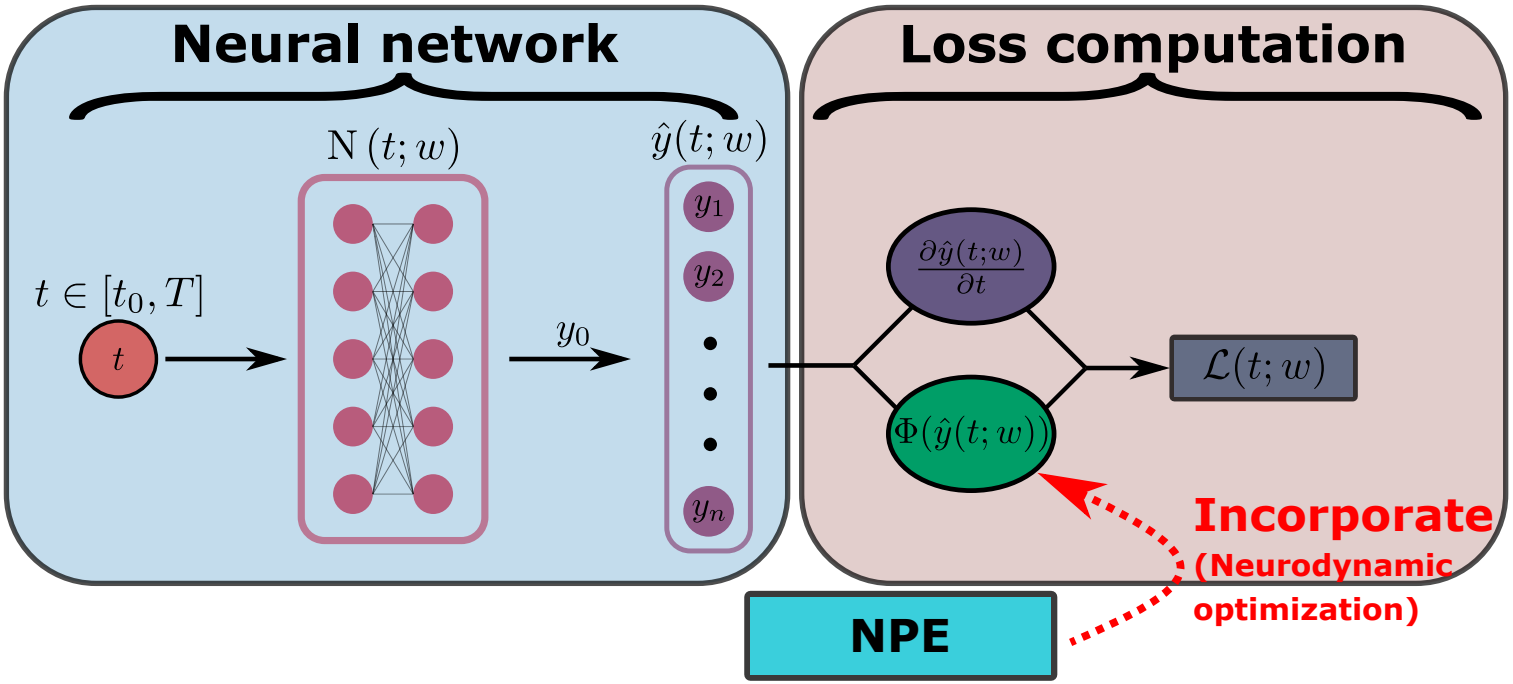
$$\mathcal{L} = \mathcal{L}_u + \mathcal{L}_f \tag{13}$$

$$\mathcal{L}_u = \frac{1}{N_u} \sum_{i=1}^{N_u} \left(\hat{u}_\theta(t_u^i, x_u^i) - u^i \right)^2, \quad \mathcal{L}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left(\hat{u}_t(t_f^i, x_f^i) + \mathcal{N}[\hat{u}](t_f^i, x_f^i) \right)^2 \tag{14}$$

- $\mathcal{L}_u$ enforces initial and boundary conditions, while $\mathcal{L}_f$ imposes the PDE.
- $\mathcal{L}_u$ considers **initial and boundary training data** $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$.
- $\mathcal{L}_f$ considers **collocation points** $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$. $\{t_f^i, x_f^i\}_{i=1}^{N_f}$.
- $\hat{u}_\theta$ is trained toward minimizing $\mathcal{L}$ to become a solution of the PDE.

Table of Contents

Methodology: Overview



Three steps:

- Reformulate the NPE as an ODE system by the neurodynamic approach (2007).
- Build a NN model with the initial condition (t_0, y_0).
- Train the NN toward solving the ODE system.

The NPE is not entered directly in the NN, but is incorporated in the loss calculation.

Methodology: Model Details

- **NN model:**

$$\hat{y}(t; w) = \underbrace{y_0 + \left(1 - e^{-(t-t_0)}\right)}_{\text{modifies the NN output}} N(t; w), \quad t \in [t_0, T] \tag{15}$$

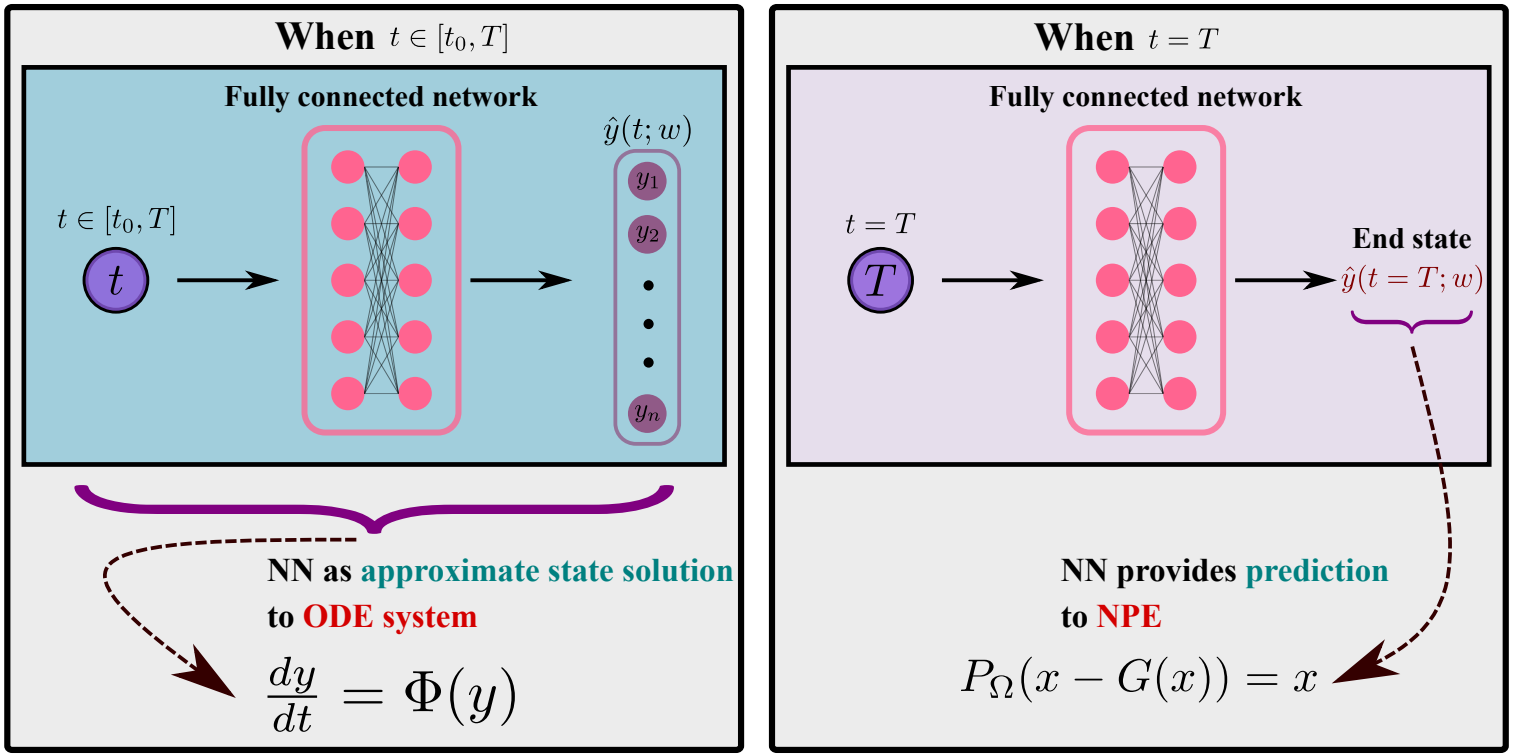
  - $t \in \mathbb{R}$ is the input, $\hat{y}(t; w)$ is the predicted state at t.
  - $N(\cdot; w)$ is the FNN with parameters w.
  - (t_0, y_0) is the initial point, $[t_0, T]$ is the time range.
  - $y_0 = \hat{y}(t_0; w)$, the initial condition is always satisfied regardless w.

- **Loss function:**

$$\mathcal{L}(\mathbb{T}, w) = \frac{1}{|\mathbb{T}|} \sum_{t \in \mathbb{T}} \left\| \frac{\partial \hat{y}(t; w)}{\partial t} - \Phi(\hat{y}(t; w)) \right\|, \tag{16}$$

  - $\mathbb{T} = \{t_i\}$ is the collocation time points. Each point is sampled by $U(t_0, T)$.
  - $\frac{\partial \hat{y}(t;w)}{\partial t}$ is the derivatives of the model output w.r.t. the input, computed by automatic differentiation, e.g., Pytorch.
  - $\Phi(\cdot)$ is the ODE system corresponding to the NPE to be solved.

Methodology: NN Prediction to NPE



- Consider $t \in [t_0, T]$ as a variable (LHS):
  - the NN itself is an approximate state solution to the ODE system.
- Fix $t = T$ at the end time (RHS) :
  - the NN end state $\hat{y}(t = T; w) \in \mathbb{R}^n$ is a prediction to the NPE.

Table of Contents

Example 1: VI with 4 Variables

- Consider the following variational inequities with 4 variables:

$$(x - x^*)^T G(x^*) \geq 0, \quad x \in \Omega, \tag{17}$$

$$\mathrm{G(x)} = \begin{bmatrix} x_1 - \frac{2}{x_1+0.8} + 5x_2 - 13 \\ 1.2x_1 + 7x_2 \\ 3x_3 + 8x_4 \\ 1x_3 + 2x_4 - \frac{4}{x_4+2} - 12 \end{bmatrix}, \quad \begin{aligned} \Omega = \{x \in \mathbb{R}^4 \,|1 \leq x_1 \leq 100, -3 \leq x_2 \leq 100, \\ -10 \leq x_3 \leq 100, 1 \leq x_4 \leq 100\}. \end{aligned} \tag{18}$$

- By Prop. 2.3 in Harker & Pang (1990), the VI is reformulated as $NPE(G, \Omega)$.
- Mean square error evaluates how well the NN solves the ODE system

$$\mathcal{L}(\mathbb{T}, w) = \frac{1}{|\mathbb{T}|} \sum_{t \in \mathbb{T}} \left\| \frac{\partial \hat{y}(t; w)}{\partial t} - \Phi(\hat{y}(t; w)) \right\|. \tag{19}$$

- NPE error evaluates how well the NN solves the NPE

$$NPE\_error(x_{\text{pred}}) = \| P_\Omega(x_{\text{pred}} - G(x_{\text{pred}})) - x_{\text{pred}} \|, \tag{20}$$

where $x_{\text{pred}} \in \mathbb{R}^4$ is the NN prediction.

Example 1: VI with 4 Variables

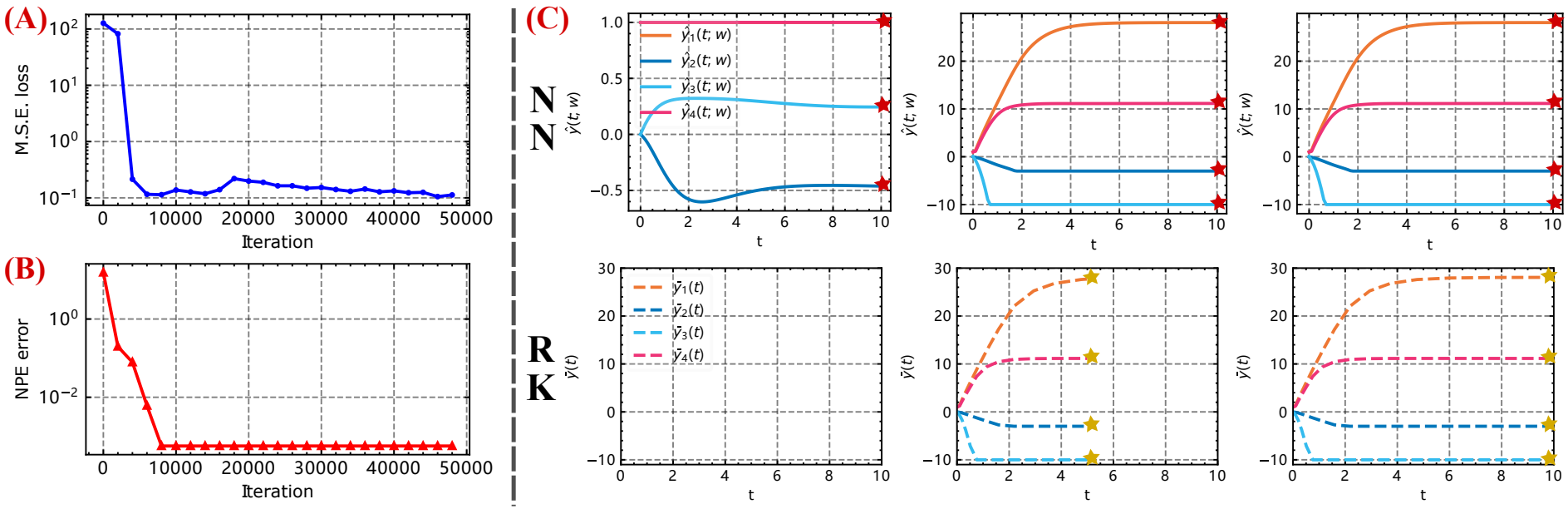


Figure: (A) MSE loss. (B) NPE error. (C) NN and RK (ODE solver) solution procedures.

- 8000 iterations: M.S.E loss $126.99 \rightarrow 0.1$, NPE error $15.42 \rightarrow 0.01$.
- In each iteration:
  - the NN improves the predicted state solution over the entire time range.
  - the RK advances a collocation point to a new end state.

Example 2: NPE with 1000 variables

- Consider the following NPE problem:

$$\left(x_i^* - \frac{1}{2\sqrt{(Mx^*)_i + q_i}} \right)^+ = x_i^*, \quad i = 1, 2, \ldots, 1000. \tag{21}$$

- $M \in \mathbb{R}^{1000\times 1000}$ and $q \in \mathbb{R}^{1000}$ are problem data.
- The objective is to find an optimal solution $x^* = [x_1^*, x_2^*, \ldots, x_{1000}^*] \in \mathbb{R}^{1000}$ that solves (21).
- We generate a problem set with ten different (M, q) [4], which correspond to ten different NPE instances.

[4] the problem data is stored in https://github.com/wuwudawen/IJNME_data_2023/blob/main/Q.npy

Example 2: NPE with 1000 variables

Iteration	M.S.E. loss (Mean ± STD)	CPU time (Mean ± STD)	NPE error (Mean ± STD)
0	470.29 ± 92.68	0.00 ± 0.00	4839.33 ± 673.10
100	45.85 ± 39.83	2.36 ± 0.60	15.34 ± 16.46
500	32.10 ± 26.70	11.32 ± 0.68	0.67 ± 0.89
1000	26.81 ± 11.53	22.57 ± 1.16	0.31 ± 0.18
3000	15.20 ± 11.23	67.82 ± 1.40	0.19 ± 0.06
5000	12.33 ± 10.44	112.91 ± 2.20	0.13 ± 0.07
7000	8.16 ± 8.13	157.98 ± 2.86	0.10 ± 0.06
10000	2.93 ± 3.38	225.47 ± 3.82	0.05 ± 0.04
30000	0.73 ± 1.18	670.38 ± 5.82	0.01 ± 0.01

Table: The NN method

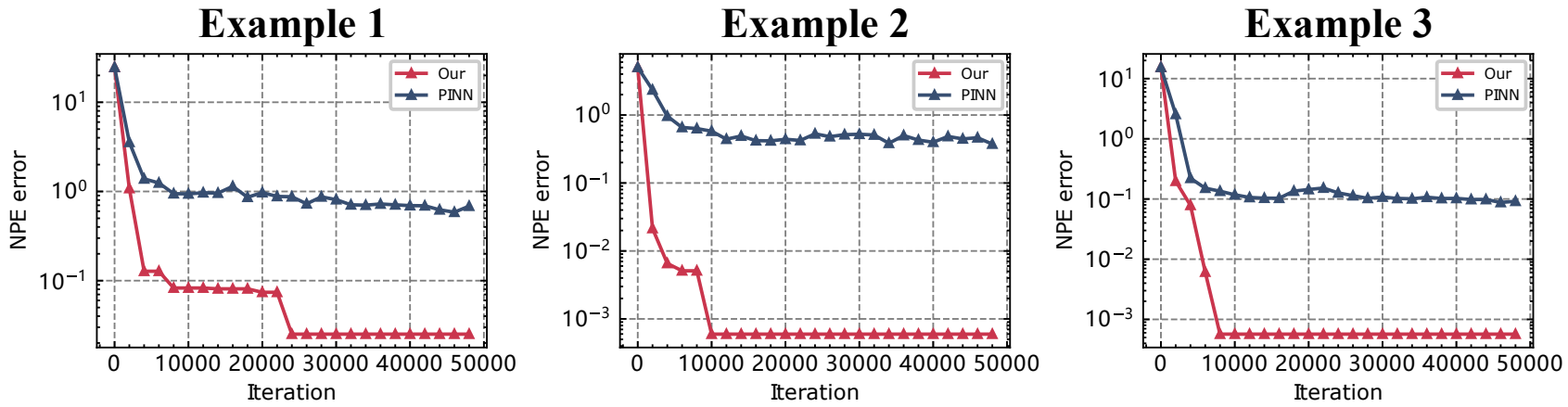
Time range	CPU time (Mean ± STD)	NPE error (Mean ± STD)
$[0, 2]$	369.33 ± 44.27	1.87 ± 0.16
$[0, 4]$	547.93 ± 74.83	0.48 ± 0.06
$[0, 6]$	739.74 ± 88.02	0.14 ± 0.02
$[0, 8]$	1048.92 ± 88.20	0.04 ± 0.01
$[0, 10]$	1542.07 ± 205.43	0.01 ± 0.01

Table: The RK method

The NN method:

- Finds the optimal solution x^* for the 1000-dimensional NPE.
- Outperforms the neurodynamic approach (Xia & Feng, 2007) with the RK solver.
- Very efficient when the accuracy requirement is relaxed.
  - The NN: 20s to reach NPE error 0.31
  - The RK: 500s to reach NPE error 0.48.

Experiments - Comparison with PINNs



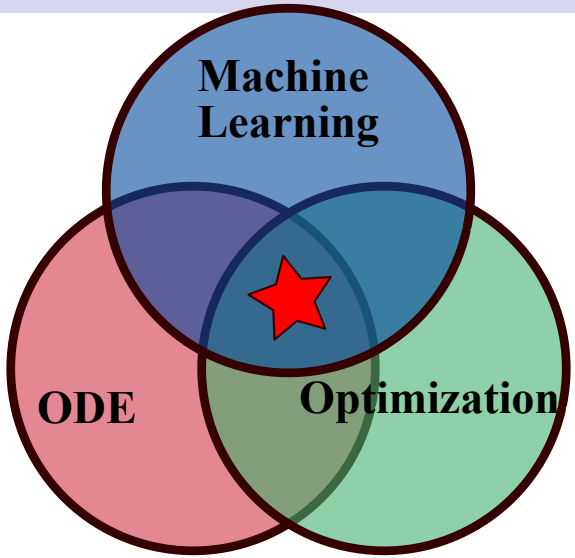
Remark

Take advantage of the problem structure!

- Our NN method focuses on improving the terminal state, $t = T$.
- The baseline method considers equally the entire input space, $[t_0, T]$.

Table of Contents

Concluding remarks



- An optimization problem is first modeled as an ODE system. Its convergence is proven using a Lyapunov function and LaSalle's invariance principle.
- PINN is employed to solve the ODE system. The algorithm and the neural network are specifically tailored to better solve the target problem.

- We reformulate an optimization problem as a neural network training task. This allows various AI tools, such as CUDA for acceleration, to be directly applied to solve these optimization problems.
- This opens the door for many AI methods, such as transfer learning and multi-task learning, to be leveraged for solving optimization problems.

Future works

This thesis is only the beginning of a series of works.

- Large scale problem:
  - Can the proposed NN solves problems with $>$ 1 million variables?
- Convergence: **(Most asked!!!)**
  - Can the proposed NN has a theoretical guarantee?
  - Can we derive an error bound w.r.t. training iteration?
- Solve more optimization problems
  - Can we solves some nonconvex problems ,e.g., Pseudoconvex, Biconvex.
  - How to take advantage of the problem structure?
- What about Operator Learning (DeepONet)?
  - PINN finds the solution of a PDE, DeepONet finds the PDE operator.
  - What happen if we replace PINN with DeepONet?

References

- **Wu, D.**, & Lisser, A. (2023). Neuro-PINN: A Hybrid Framework for Efficient Nonlinear Projection Equation Solutions. Neural Networks. **International Journal for Numerical Methods in Engineering (IJNME)**.
- **Wu, D.**, & Chamoin, L. & Lisser, A. (2025). Solving Large-Scale Variational Inequalities with Dynamically Adjusting Initial Condition in Physics-Informed Neural Networks. **Computer Methods in Applied Mechanics and Engineering (CMAME)**.
- **Wu, D.**, & Lisser, A. (2023). Parallel Solution of Nonlinear Projection Equations in a Multi-Task Learning Framework. **IEEE Transactions on Neural Networks and Learning Systems (TNNLS)**.
- **Wu, D.**, & Lisser, A. (2025). Error Bound Analysis of Physics-Informed Neural Networks for Solving Nonlinear Projection Equations. **Journal of Optimization Theory and Applications (JOTA)**.
- Zi-Yu Khoo, **Dawen Wu**, Jonathan Sze Choong Low, Stéphane Bressan (2024). Separable Hamiltonian Neural Networks. **Physical Review E (PRE)**.
- **Dawen Wu**, Ludovic, Chamoin, Stéphane Bressan (2025). Neural Triangular Map for Density Estimation and Sampling with Application to Bayesian Inference. **Journal of Computational Physics (JCP)**.

- Lagaris I E, Likas A, Fotiadis D I(1998). Artificial neural networks for solving ordinary and partial differential equations. **IEEE transactions on neural networks**
- Raissi M, Perdikaris P, Karniadakis G E (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. **Journal of Computational physics.**
- T De Ryck, S Mishra (2025). Numerical analysis of physics-informed neural networks and related models in physics-informed machine learning. **Acta Numerica**.
- Shi Z, Hu Z, Lin M, et al. Stochastic Taylor Derivative Estimator: Efficient amortization for arbitrary differential operators. **NeuraIPS 2024**.
- Xia Y, Liu Q, Wang J, et al. A survey of neurodynamic optimization. **IEEE Transactions on Emerging Topics in Computational Intelligence, 2024.**.

Thank you!

Thank you for coming!!!!