Dynamical systems-based structured networks

Davide Murari

Department of Applied Mathematics and Theoretical Physics, University of Cambridge.

dm2011@cam.ac.uk

Outline

- Introduction to dynamical systems-based neural networks.
- Time-dependent symplectic networks.
- 1-Lipschitz and $\alpha-$averaged networks.

ResNets as dynamical systems

- Residual Neural Networks (ResNets) are networks of the form $\mathcal{N}_\theta = f_{\theta_L} \circ \ldots \circ f_{\theta_1}$ with

$$f_{\theta_i}(\mathbf{x}) = \mathbf{x} + B_i^\top \sigma\left(A_i \mathbf{x} + \boldsymbol{b}_i\right) \in \mathbb{R}^d,\ \mathbf{x} \in \mathbb{R}^d,$$
$$A_i, B_i \in \mathbb{R}^{h\times d},\ \boldsymbol{b}_i \in \mathbb{R}^h,\ \theta_i = \{A_i, B_i, \boldsymbol{b}_i\}\,.$$

ResNets as dynamical systems

- Residual Neural Networks (ResNets) are networks of the form $\mathcal{N}_\theta = f_{\theta_L} \circ \ldots \circ f_{\theta_1}$ with

$$f_{\theta_i}(\mathbf{x}) = \mathbf{x} + B_i^\top \sigma\left(A_i \mathbf{x} + \boldsymbol{b}_i\right) \in \mathbb{R}^d, \ \mathbf{x} \in \mathbb{R}^d,$$
$$A_i, B_i \in \mathbb{R}^{h \times d}, \ \boldsymbol{b}_i \in \mathbb{R}^h, \ \theta_i = \{A_i, B_i, \boldsymbol{b}_i\}.$$

- The layer

$$f_{\theta_i}(\mathbf{x}) = \mathbf{x} + B_i^\top \sigma\left(A_i \mathbf{x} + \boldsymbol{b}_i\right) = \mathbf{x} + \mathcal{F}_{\theta_i}(\mathbf{x}) \in \mathbb{R}^d$$

is an explicit Euler step of size 1 for the initial value problem

$$\begin{cases} \dot{\mathbf{y}}(t) = B_i^\top \sigma(A_i \mathbf{y}(t) + \boldsymbol{b}_i) = \mathcal{F}_{\theta_i}(\mathbf{y}(t)), \\ \mathbf{y}(0) = \mathbf{x} \end{cases}.$$

ResNet-like archtectures

- We can define ResNet-like neural networks by choosing a family of parametric functions $\mathcal{S}_\Theta = \left\{\mathcal{F}_\theta : \mathbb{R}^d \to \mathbb{R}^d : \ \theta \in \Theta\right\}$ and a numerical method $\Psi^h_{\mathcal{F}}$, like explicit Euler defined as $\Psi^h_{\mathcal{F}}(\mathbf{x}) = \mathbf{x} + h\mathcal{F}(\mathbf{x})$, and set

$$\mathcal{N}_\theta(\mathbf{x}) = \Psi^{h_L}_{\mathcal{F}_{\theta_L}} \circ \cdots \circ \Psi^{h_1}_{\mathcal{F}_{\theta_1}}(\mathbf{x}), \ \mathcal{F}_{\theta_1}, ..., \mathcal{F}_{\theta_L} \in \mathcal{S}_\Theta.$$

- We could also combine these residual blocks with lifting and projection layers, as for usual neural networks.

Structured networks based on dynamical systems

- Choose a property $\mathcal{P}$ that the network has to satisfy, e.g. volume preservation.

Structured networks based on dynamical systems

- Choose a property $\mathcal{P}$ that the network has to satisfy, e.g. volume preservation.
- Choose a family of parametric vector fields $\mathcal{S}_\Theta$ whose solutions satisfy $\mathcal{P}$, e.g.

$$\mathcal{F}_\theta(\mathbf{x}) = \begin{bmatrix} \sigma\left(A_1\mathbf{x}_2 + \boldsymbol{b}_1\right) \\ \sigma\left(A_2\mathbf{x}_1 + \boldsymbol{b}_2\right) \end{bmatrix}, \ \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}.$$

Structured networks based on dynamical systems[1]

- Choose a property $\mathcal{P}$ that the network has to satisfy, e.g. volume preservation.
- Choose a family of parametric vector fields $\mathcal{S}_\Theta$ whose solutions satisfy $\mathcal{P}$, e.g.

$$\mathcal{F}_\theta(\mathbf{x}) = \begin{bmatrix} \sigma\left(A_1\mathbf{x}_2 + \boldsymbol{b}_1\right) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \sigma\left(A_2\mathbf{x}_1 + \boldsymbol{b}_2\right) \end{bmatrix}, \ \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}.$$

- Choose a numerical method $\Psi^h_{\mathcal{F}_\theta}$ that preserves the property $\mathcal{P}$ at a discrete level, e.g.

$$\Psi^h_{\mathcal{F}_\theta}(\mathbf{x}) = \begin{bmatrix} \mathbf{x}_1 + h\sigma\left(A_1\mathbf{x}_2 + \boldsymbol{b}_1\right) =: \widetilde{\mathbf{x}}_1 \\ \mathbf{x}_2 + h\sigma\left(A_2\widetilde{\mathbf{x}}_1 + \boldsymbol{b}_2\right) \end{bmatrix}.$$

- The resulting network $\mathcal{N}_\theta = \Psi^{h_L}_{\mathcal{F}_{\theta_L}} \circ \cdots \circ \Psi^{h_1}_{\mathcal{F}_{\theta_1}}$ will preserve $\mathcal{P}$.

[1] Elena Celledoni et al. "Dynamical Systems—Based Neural Networks". In: *SIAM Journal on Scientific Computing* 45.6 (2023), A3071–A3094.

Time-dependent Symplectic Neural Networks

In collaboration with Priscilla Canizares, Carola-Bibiane Schönlieb, Ferdia Sherry, Zakhar Shumaylov[2]

[2] Priscilla Canizares et al. "Hamiltonian Matching for Symplectic Neural Integrators". In: *arXiv preprint arXiv:2410.18262* (2024).

Canonical Hamiltonian equations

- The equations of motion of canonical Hamiltonian systems write

$$\dot{\mathbf{x}} = \mathbb{J}\nabla H(\mathbf{x}) = X_H(\mathbf{x}) \in \mathbb{R}^{2n}, \quad \mathbb{J} = \begin{bmatrix} 0 & I_n \\ -I_n & 0_n \end{bmatrix} \in \mathbb{R}^{2n\times 2n}. \tag{1}$$

- Denoted with $\phi_{H,t} : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ the exact flow of (1), we have that
  - $$\frac{d}{dt}H(\phi_{H,t}(\mathbf{x}_0)) = \nabla H(\phi_{H,t}(\mathbf{x}_0))^\top \mathbb{J}\nabla H(\phi_{H,t}(\mathbf{x}_0)) = 0,$$

Canonical Hamiltonian equations

- The equations of motion of canonical Hamiltonian systems write

$$\dot{\mathbf{x}} = \mathbb{J}\nabla H(\mathbf{x}) = X_H(\mathbf{x}) \in \mathbb{R}^{2n}, \quad \mathbb{J} = \begin{bmatrix} 0 & I_n \\ -I_n & 0_n \end{bmatrix} \in \mathbb{R}^{2n\times 2n}. \tag{1}$$

- Denoted with $\phi_{H,t} : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ the exact flow of (1), we have that
  - $$\frac{d}{dt}H(\phi_{H,t}(\mathbf{x}_0)) = \nabla H(\phi_{H,t}(\mathbf{x}_0))^\top \mathbb{J}\nabla H(\phi_{H,t}(\mathbf{x}_0)) = 0,$$
  - $$\left(\frac{\partial \phi_{H,t}(\mathbf{x}_0)}{\partial \mathbf{x}_0}\right)^\top \mathbb{J} \left(\frac{\partial \phi_{H,t}(\mathbf{x}_0)}{\partial \mathbf{x}_0}\right) = \mathbb{J},$$

Canonical Hamiltonian equations

- The equations of motion of canonical Hamiltonian systems write

$$\dot{\mathbf{x}} = \mathbb{J}\nabla H(\mathbf{x}) = X_H(\mathbf{x}) \in \mathbb{R}^{2n}, \quad \mathbb{J} = \begin{bmatrix} 0 & I_n \\ -I_n & 0_n \end{bmatrix} \in \mathbb{R}^{2n\times 2n}. \tag{1}$$

- Denoted with $\phi_{H,t} : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ the exact flow of (1), we have that
  - $$\frac{d}{dt}H(\phi_{H,t}(\mathbf{x}_0)) = \nabla H(\phi_{H,t}(\mathbf{x}_0))^\top \mathbb{J}\nabla H(\phi_{H,t}(\mathbf{x}_0)) = 0,$$
  - $$\left(\frac{\partial \phi_{H,t}(\mathbf{x}_0)}{\partial \mathbf{x}_0}\right)^\top \mathbb{J} \left(\frac{\partial \phi_{H,t}(\mathbf{x}_0)}{\partial \mathbf{x}_0}\right) = \mathbb{J},$$
  - the flow preserves the canonical volume form of $\mathbb{R}^{2n}$.

Forward invariant subset of the phase space

- Suppose $\mathbf{x}(t) \in \Omega \subset \mathbb{R}^{2n}$, whenever $\mathbf{x}(0) \in \Omega$, for any $t \geq 0$.

Forward invariant subset of the phase space

- Suppose $\mathbf{x}(t) \in \Omega \subset \mathbb{R}^{2n}$, whenever $\mathbf{x}(0) \in \Omega$, for any $t \geq 0$.
- By the group property of the flow map, we know that

$$\phi_{H,n\Delta t+\delta t} = \phi_{H,\delta t} \circ \underbrace{\phi_{H,\Delta t} \circ \ldots \circ \phi_{H,\Delta t}}_{n \text{ times}}, \quad n \in \mathbb{N},\ \delta t \in (0, \Delta t).$$

As a consequence, to approximate $\phi_{H,t} : \Omega \to \Omega$ for any $t \geq 0$, we only have to approximate it for $t \in [0, \Delta t]$.

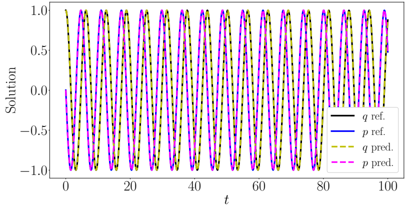


Figure 1: Neural network trained to approximate $\phi_{H,t}$ for $t \in [0, \Delta t = 1]$ and applied up to $T = 100$.

Two learning problems associated to Hamiltonian systems

Unsupervised solution of the Hamiltonian equations

Approximate the flow map $\phi_{H,t} : \Omega \to \Omega$, for any $t \geq 0$, on a compact forward invariant set $\Omega \subset \mathbb{R}^{2n}$, given the Hamiltonian energy $H : \mathbb{R}^{2n} \to \mathbb{R}$.

Two learning problems associated to Hamiltonian systems

Unsupervised solution of the Hamiltonian equations

Approximate the flow map $\phi_{H,t}:\Omega\to\Omega$, for any $t\geq 0$, on a compact forward invariant set $\Omega\subset\mathbb{R}^{2n}$, given the Hamiltonian energy $H:\mathbb{R}^{2n}\to\mathbb{R}$.

Supervised approximation of an unknown Hamiltonian flow map

Approximate the flow map $\phi_{H,t}:\Omega\to\Omega$, for any $t\geq 0$, on a compact forward invariant set $\Omega\subset\mathbb{R}^{2n}$, given trajectory segments $\{(\mathbf{x}_0^n,\mathbf{y}_1^n,\dots,\mathbf{y}_M^n)\}_{n=1}^N$, $\mathbf{y}_m^n\approx\phi_{H,t_m^n}(\mathbf{x}_0^n)$.

Two learning problems associated to Hamiltonian systems

Unsupervised solution of the Hamiltonian equations

Approximate the flow map $\phi_{H,t} : \Omega \to \Omega$, for any $t \geq 0$, on a compact forward invariant set $\Omega \subset \mathbb{R}^{2n}$, given the Hamiltonian energy $H : \mathbb{R}^{2n} \to \mathbb{R}$.

Supervised approximation of an unknown Hamiltonian flow map

Approximate the flow map $\phi_{H,t} : \Omega \to \Omega$, for any $t \geq 0$, on a compact forward invariant set $\Omega \subset \mathbb{R}^{2n}$, given trajectory segments $\{(\mathbf{x}_0^n, \mathbf{y}_1^n, \dots, \mathbf{y}_M^n)\}_{n=1}^N$, $\mathbf{y}_m^n \approx \phi_{H,t_m^n}(\mathbf{x}_0^n)$.

Remark: Given the several known qualitative properties of $\phi_{H,t}$, we want to exploit them when designing the approximating map.

The SympFlow

- We now build a neural network that approximates $\phi_{H,t} : \Omega \to \Omega$ for a forward invariant set $\Omega \subset \mathbb{R}^{2n}$, and $t \in [0, \Delta t]$, while reproducing the qualitative properties of $\phi_{H,t}$.

The SympFlow

- We now build a neural network that approximates $\phi_{H,t} : \Omega \to \Omega$ for a forward invariant set $\Omega \subset \mathbb{R}^{2n}$, and $t \in [0, \Delta t]$, while reproducing the qualitative properties of $\phi_{H,t}$.
- We rely on two building blocks, which applied to $(\mathbf{q}, \mathbf{p}) \in \mathbb{R}^{2n}$ write:

$$\phi_{\mathbf{p},t}((\mathbf{q}, \mathbf{p})) = \begin{bmatrix} \mathbf{q} \\ \mathbf{p} - (\nabla_{\mathbf{q}} V(t, \mathbf{q}) - \nabla_{\mathbf{q}} V(0, \mathbf{q})) \end{bmatrix},$$

$$\phi_{\mathbf{q},t}((\mathbf{q}, \mathbf{p})) = \begin{bmatrix} \mathbf{q} + (\nabla_{\mathbf{p}} K(t, \mathbf{p}) - \nabla_{\mathbf{p}} K(0, \mathbf{p})) \\ \mathbf{p} \end{bmatrix}.$$

The SympFlow

- We now build a neural network that approximates $\phi_{H,t} : \Omega \to \Omega$ for a forward invariant set $\Omega \subset \mathbb{R}^{2n}$, and $t \in [0, \Delta t]$, while reproducing the qualitative properties of $\phi_{H,t}$.
- We rely on two building blocks, which applied to $(\mathbf{q}, \mathbf{p}) \in \mathbb{R}^{2n}$ write:

$$\phi_{\mathbf{p},t}((\mathbf{q},\mathbf{p})) = \begin{bmatrix} \mathbf{q} \\ \mathbf{p} - (\nabla_{\mathbf{q}} V(t,\mathbf{q}) - \nabla_{\mathbf{q}} V(0,\mathbf{q})) \end{bmatrix},$$

$$\phi_{\mathbf{q},t}((\mathbf{q},\mathbf{p})) = \begin{bmatrix} \mathbf{q} + (\nabla_{\mathbf{p}} K(t,\mathbf{p}) - \nabla_{\mathbf{p}} K(0,\mathbf{p})) \\ \mathbf{p} \end{bmatrix}.$$

- The SympFlow architecture is defined as

$$\mathcal{N}_\theta(t, (\mathbf{q}_0, \mathbf{p}_0)) = \phi^L_{\mathbf{p},t} \circ \phi^L_{\mathbf{q},t} \circ \cdots \circ \phi^1_{\mathbf{p},t} \circ \phi^1_{\mathbf{q},t}((\mathbf{q}_0, \mathbf{p}_0)).$$

Properties of the SympFlow

- The SympFlow is symplectic for every time $t \in \mathbb{R}$. The building blocks we compose are exact flows of time-dependent Hamiltonian systems:

$$
\begin{aligned}
\phi^i_{\mathbf{p},t}((\mathbf{q},\mathbf{p})) &= \begin{bmatrix} \mathbf{q} \\ \mathbf{p} - \left(\nabla_{\mathbf{q}} V^i(t,\mathbf{q}) - \nabla_{\mathbf{q}} V^i(0,\mathbf{q})\right) \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{q} \\ \mathbf{p} - \nabla_{\mathbf{q}} \left(\int_0^t \partial_s V^i(s,\mathbf{q}) ds \right) \end{bmatrix} = \phi_{\widetilde{V}^i,t}((\mathbf{q},\mathbf{p})),
\end{aligned}
$$

with $\widetilde{V}^i(t,(\mathbf{q},\mathbf{p})) = \partial_t V^i(t,\mathbf{q})$.

Properties of the SympFlow

- The SympFlow is symplectic for every time $t \in \mathbb{R}$. The building blocks we compose are exact flows of time-dependent Hamiltonian systems:

$$\phi^i_{\mathbf{p},t}((\mathbf{q},\mathbf{p})) = \begin{bmatrix} \mathbf{q} \\ \mathbf{p} - \left(\nabla_{\mathbf{q}} V^i(t,\mathbf{q}) - \nabla_{\mathbf{q}} V^i(0,\mathbf{q})\right) \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{q} \\ \mathbf{p} - \nabla_{\mathbf{q}} \left(\int_0^t \partial_s V^i(s,\mathbf{q}) ds \right) \end{bmatrix} = \phi_{\widetilde{V}^i,t}((\mathbf{q},\mathbf{p})),$$

  with $\widetilde{V}^i(t,(\mathbf{q},\mathbf{p})) = \partial_t V^i(t,\mathbf{q})$.

- The SympFlow is volume preserving.

Properties of the SympFlow

- The SympFlow is symplectic for every time $t \in \mathbb{R}$. The building blocks we compose are exact flows of time-dependent Hamiltonian systems:

$$
\begin{aligned}
\phi^i_{\mathbf{p},t}((\mathbf{q},\mathbf{p})) &= \begin{bmatrix} \mathbf{q} \\ \mathbf{p} - \left(\nabla_{\mathbf{q}} V^i(t,\mathbf{q}) - \nabla_{\mathbf{q}} V^i(0,\mathbf{q})\right) \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{q} \\ \mathbf{p} - \nabla_{\mathbf{q}} \left(\int_0^t \partial_s V^i(s,\mathbf{q}) ds\right) \end{bmatrix} = \phi_{\widetilde{V}^i,t}((\mathbf{q},\mathbf{p})),
\end{aligned}
$$

  with $\widetilde{V}^i(t,(\mathbf{q},\mathbf{p})) = \partial_t V^i(t,\mathbf{q})$.
- The SympFlow is volume preserving.
- The SympFlow is the exact solution of a time-dependent Hamiltonian system.

Composition of Hamiltonian flows[3]

Theorem (The Hamiltonian flows are closed under composition)

Let $H^1, H^2 : \mathbb{R} \times \mathbb{R}^{2n} \to \mathbb{R}$ be continuously differentiable functions. Then, the map $\phi_{H^2,t} \circ \phi_{H^1,t} : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ is the exact flow of the time-dependent Hamiltonian system defined by the Hamiltonian function

$$H^3(t, \mathbf{x}) = H^2(t, \mathbf{x}) + H^1\left(t, \phi^{-1}_{H^2,t}(\mathbf{x})\right).$$

- This theorem implies that there is a Hamiltonian function $\mathcal{H}(\mathcal{N}_\theta) : \mathbb{R} \times \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ such that
$$\mathcal{N}_\theta\,(t, \mathbf{x}) = \phi_{\mathcal{H}(\mathcal{N}_\theta),t}(\mathbf{x})$$
for every $t \geq 0$ and $\mathbf{x} \in \mathbb{R}^{2n}$.

[3] Leonid Polterovich. *The Geometry of the Group of Symplectic Diffeomorphisms*. Lectures in Mathematics ETH Zürich. Basel: Springer Basel AG, 2001. ISBN: 978-3-7643-6432-8.

Training of the SympFlow to solve $\dot{\mathbf{x}}(t) = X_H(\mathbf{x}(t))$

- The SympFlow is based on modelling the scalar-valued potentials $\widetilde{V}^i, \widetilde{K}^i : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ with feed-forward neural networks.

- To train the overall model $\mathcal{N}_\theta$ we minimise the loss function

$$\mathcal{L}(\theta) = \underbrace{\frac{1}{N_r}\sum_{i=1}^{N_r}\left\| \frac{d}{dt}\mathcal{N}_\theta\left(t, \mathbf{x}_0^i\right)\Big|_{t=t_i} - \mathbb{J}\nabla H\left(\mathcal{N}_\theta\left(t_i, \mathbf{x}_0^i\right)\right)\right\|_2^2}_{\text{Residual term}}$$
$$+ \underbrace{\frac{1}{N_m}\sum_{j=1}^{N_m}\left(\mathcal{H}(\mathcal{N}_\theta)(t_j, \mathbf{x}^j) - H(\mathbf{x}^j)\right)^2}_{\text{Hamiltonian matching}},$$

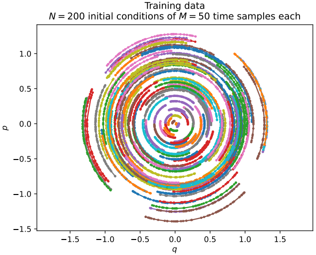
where we sample $t_i, t_j \in [0, \Delta t]$, and $\mathbf{x}_0^i, \mathbf{x}^i \in \Omega \subset \mathbb{R}^{2n}$.

Supervised training of the SympFlow to approximate $\phi_{H,t}$

- The SympFlow is based on modelling the scalar-valued potentials $\widetilde{V}^i, \widetilde{K}^i : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ with feed-forward neural networks.
- To train the overall model $\mathcal{N}_\theta$ we minimise the loss function

$$\mathcal{L}(\theta) = \frac{1}{NM}\sum_{n=1}^{N}\sum_{m=1}^{M}\left\|\mathcal{N}_\theta\left(t_m^n, \mathbf{x}_0^n\right) - \mathbf{y}_m^n\right\|_2^2,$$

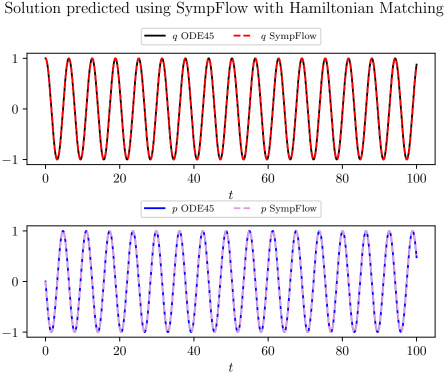
where $\mathbf{x}_0^n \in \Omega \subset \mathbb{R}^{2n}$, and $\mathbf{y}_m^n \approx \phi_{H,t_m^n}(\mathbf{x}_0^n)$.

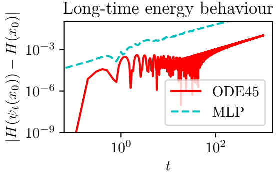


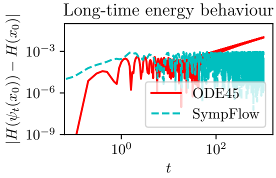
Simple Harmonic Oscillator (unsupervised)

Equations of motion

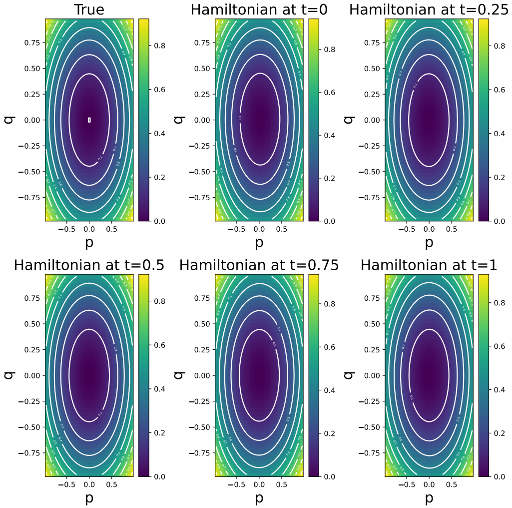
$$\dot{x} = p, \quad \dot{p} = -x.$$

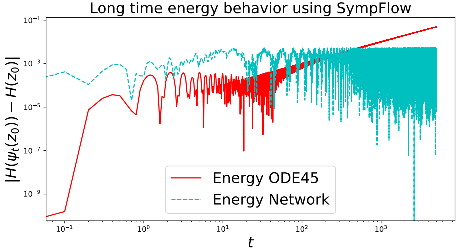


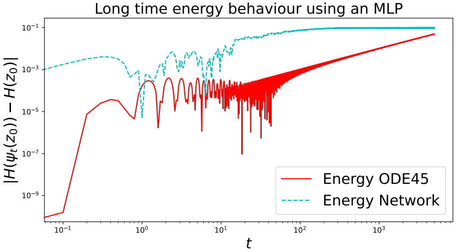




Simple Harmonic Oscillator (supervised)







1-Lipschitz and $\alpha-$averaged neural networks

In collaboration with Elena Celledoni, Matthias J. Ehrhardt, Brynjulf Owren, Carola-Bibiane Schönlieb, Ferdia Sherry[4]

[4] Ferdia Sherry et al. "Designing stable neural networks using convex analysis and odes". In: *Physica D: Nonlinear Phenomena* 463 (2024), p. 134159.

Why do we care about these constraints?

Adversarial robustness

Constraining the Lipschitz constant allows the reduction/control of the sensitivity of the network to perturbations in the input space.

Convergent Plug-and-Play algorithms

They can help in Plug-and-Play algorithms to ensure the convergence of the iteration

$$\mathbf{x}_{k+1} = \mathcal{N}_\theta(\mathbf{x}_k - \tau \nabla F(\mathbf{x}_k)).$$

This iteration converges if $\mathcal{N}_\theta$ is $\alpha-$averaged and the sequence has a fixed point[a].

[a] Pravin Nair, Ruturaj G Gavaskar, and Kunal Narayan Chaudhury. "Fixed-Point and Objective Convergence of Plug-and-Play Algorithms". In: *IEEE Transactions on Computational Imaging* 7 (2021), pp. 337–348.

1-Lipschitz neural networks

Non-expansive dynamical systems

$$\text{Dynamics: } \dot{\mathbf{x}} = \mathcal{F}_\theta(\mathbf{x}) = -A^\top \sigma\left(A\mathbf{x} + \boldsymbol{b}\right).$$

$$\text{Explicit Euler approximation: } \Psi^h_{\mathcal{F}_\theta}(\mathbf{x}) = \mathbf{x} - hA^\top \sigma\left(A\mathbf{x} + \boldsymbol{b}\right).$$

$$\text{1-Lipschitz map: } \left\|\Psi^h_{\mathcal{F}_\theta}(\mathbf{y}) - \Psi^h_{\mathcal{F}_\theta}(\mathbf{x})\right\|_2 \leq \|\mathbf{y} - \mathbf{x}\|_2,$$

if $h \leq 2/\|A\|_2^2$ and σ is $1-$Lipschitz.

We define the $1-$Lipschitz neural network

$$\mathcal{N}_\theta = \Psi^{h_L}_{\mathcal{F}_{\theta_L}} \circ \ldots \circ \Psi^{h_L}_{\mathcal{F}_{\theta_L}} : \mathbb{R}^d \to \mathbb{R}^d,$$

where $h_1, \ldots, h_L$ are adjusted during training to ensure $h_i \leq 2/\|A_i\|_2^2$.

$\alpha-$averaged neural networks

$\alpha-$averaged maps

A map $F : \mathbb{R}^d \to \mathbb{R}^d$ is $\alpha-$averaged, $\alpha \in (0, 1)$, if there is a $1-$Lipschitz map $T : \mathbb{R}^d \to \mathbb{R}^d$ such that

$$F = (1 - \alpha)\mathrm{id} + \alpha T.$$

If F is continuously differentiable and has symmetric Jacobian, then it is $\alpha-$averaged if and only if $\mathrm{spectrum}(F'(\mathbf{x})) \subset [1 - 2\alpha, 1]$. Composition of averaged maps is averaged.

The map

$$\Psi^h_{\mathcal{F}_\theta}(\mathbf{x}) = \mathbf{x} - hA^\top \sigma\left(A\mathbf{x} + \boldsymbol{b}\right) = \nabla\left(\frac{\|\mathbf{x}\|_2^2}{2} - h1^T\gamma(A\mathbf{x} + b)\right),\ \gamma' = \sigma,$$

is averaged if $h \leq 2/\|A\|_2^2$.

Comparison of learned denoisers

$$\Gamma_{\text{Euler}} := \mathcal{P} \circ \mathcal{N}_\theta \circ \mathcal{L},$$

$$\mathcal{L}(x_1, x_2, x_3) = (x_1, x_3, x_3, 0, \dots, 0) \in \mathbb{R}^{64}, \ \mathcal{P}(x_1, \dots, x_{64}) = (x_1, x_2, x_3) \in \mathbb{R}^3.$$

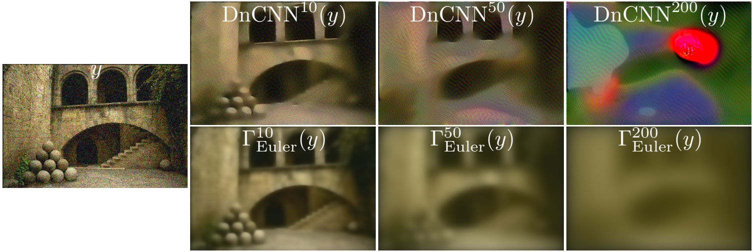


Figure 2: Repeated application of the unconstrained denoiser DnCNN[5] and the constrained denoiser Γ_{Euler} to a given input image.

[5] Kai Zhang et al. "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising". In: *IEEE transactions on image processing* 26.7 (2017), pp. 3142–3155.

Plug-and-Play for a deblurring task

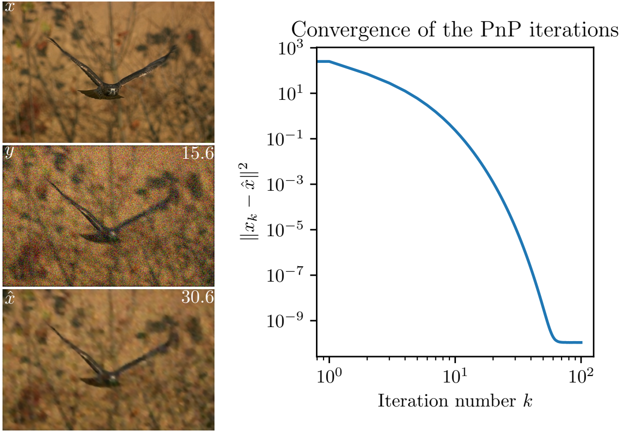


Figure 3: Using the learned Euler denoiser to solve an ill-posed inverse problem (deblurring) in a PnP fashion, with convergence guarantee. The numbers in the top right corner of each image are the PSNRs (in dB) relative to the ground truth x.

- Canizares, Priscilla et al. “Hamiltonian Matching for Symplectic Neural Integrators”. In: *arXiv preprint arXiv:2410.18262* (2024).
- Celledoni, Elena et al. “Dynamical Systems—Based Neural Networks”. In: *SIAM Journal on Scientific Computing* 45.6 (2023), A3071–A3094.
- Nair, Pravin, Ruturaj G Gavaskar, and Kunal Narayan Chaudhury. “Fixed-Point and Objective Convergence of Plug-and-Play Algorithms”. In: *IEEE Transactions on Computational Imaging* 7 (2021), pp. 337–348.
- Polterovich, Leonid. *The Geometry of the Group of Symplectic Diffeomorphisms*. Lectures in Mathematics ETH Zürich. Basel: Springer Basel AG, 2001. ISBN: 978-3-7643-6432-8.
- Sherry, Ferdia et al. “Designing stable neural networks using convex analysis and odes”. In: *Physica D: Nonlinear Phenomena* 463 (2024), p. 134159.
- Zhang, Kai et al. “Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising”. In: *IEEE transactions on image processing* 26.7 (2017), pp. 3142–3155.

THANK YOU FOR THE ATTENTION

Physics-informed neural networks

- We introduce a parametric map $\mathcal{N}_\theta(\cdot, \mathbf{x}_0) : [0, T] \to \mathbb{R}^d$ such that $\mathcal{N}_\theta(0, \mathbf{x}_0) = \mathbf{x}_0$, and choose its weights so that

$$\mathcal{L}(\theta) := \frac{1}{C} \sum_{c=1}^{C} \left\| \frac{d}{dt} \mathcal{N}_\theta(t, \mathbf{x}_0) \Big|_{t=t_c} - \mathcal{F}\left(\mathcal{N}_\theta(t_c, \mathbf{x}_0)\right) \right\|_2^2 \to \min$$

  for some collocation points $t_1, \ldots, t_C \in [0, T]$.

Physics-informed neural networks

- We introduce a parametric map $\mathcal{N}_\theta(\cdot, \mathbf{x}_0) : [0, T] \to \mathbb{R}^d$ such that $\mathcal{N}_\theta(0, \mathbf{x}_0) = \mathbf{x}_0$, and choose its weights so that

$$\mathcal{L}(\theta) := \frac{1}{C} \sum_{c=1}^{C} \left\| \frac{d}{dt} \mathcal{N}_\theta(t, \mathbf{x}_0) \Big|_{t=t_c} - \mathcal{F}(\mathcal{N}_\theta(t_c, \mathbf{x}_0)) \right\|_2^2 \to \min$$

  for some collocation points $t_1, \ldots, t_C \in [0, T]$.

- Then, $t \mapsto \mathcal{N}_\theta(t, \mathbf{x}_0)$ will solve a different IVP

$$\begin{cases} \dot{\mathbf{y}}(t) = \mathcal{F}(\mathbf{y}(t)) + \left(\frac{d}{dt} \mathcal{N}_\theta(t, \mathbf{x}_0) \big|_{t=t} - \mathcal{F}(\mathbf{y}(t)) \right) \in \mathbb{R}^d, \\ \mathbf{y}(0) = \mathbf{x}_0 \in \mathbb{R}^d, \end{cases}$$

  where hopefully the residual $\frac{d}{dt} \mathcal{N}_\theta(t, \mathbf{x}_0) \big|_{t=t} - \mathcal{F}(\mathbf{y}(t))$ is small in some sense.

Training issues with neural network

- Solving a single IVP on $[0, T]$ with a neural network can take long training time.
- The obtained solution can not be used to solve the same ordinary differential equation with a different initial condition.

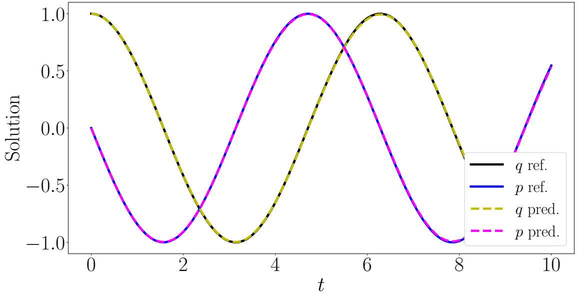
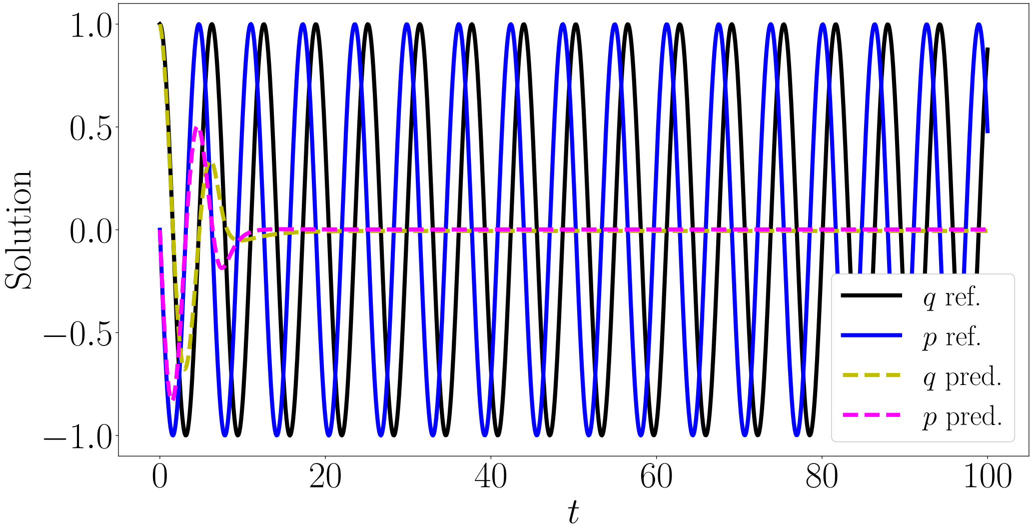


Figure 4: Solution comparison after reaching a loss value of 10^{-5}. The training time is of 87 seconds (7500 epochs with 1000 new collocation points randomly sampled at each of them).

Training issues with neural network

- It is hard to solve initial value problems over long time intervals.



Extension of the SympFlow outside of $[0, \Delta t]$

- Once we have trained $\mathcal{N}_\theta$ to be reliable for $t \in [0, \Delta t]$, we extend it for longer times as

$$\psi(t, \mathbf{x}_0) := \bar{\psi}_{t-\Delta t\lfloor t/\Delta t\rfloor} \circ \left(\bar{\psi}_{\Delta t}\right)^{\lfloor t/\Delta t\rfloor}(\mathbf{x}_0),$$

for $t \in [0, +\infty)$ and $\mathbf{x}_0 \in \Omega \subset \mathbb{R}^{2n}$, where

$$\bar{\psi}_s(\mathbf{x}_0) := \mathcal{N}_\theta(s, \mathbf{x}_0), \;\; s \in [0, \Delta t),$$
$$\left(\bar{\psi}_{\Delta t}\right)^k := \underbrace{\bar{\psi}_{\Delta t} \circ \cdots \circ \bar{\psi}_{\Delta t}}_{k \text{ times}}, \;\; k \in \mathbb{N}.$$

Extension of the SympFlow outside of $[0, \Delta t]$

- Once we have trained $\mathcal{N}_\theta$ to be reliable for $t \in [0, \Delta t]$, we extend it for longer times as

$$\psi(t, \mathbf{x}_0) := \bar{\psi}_{t-\Delta t\lfloor t/\Delta t\rfloor} \circ \left(\bar{\psi}_{\Delta t}\right)^{\lfloor t/\Delta t\rfloor} (\mathbf{x}_0),$$

  for $t \in [0, +\infty)$ and $\mathbf{x}_0 \in \Omega \subset \mathbb{R}^{2n}$, where

$$\bar{\psi}_s(\mathbf{x}_0) := \mathcal{N}_\theta\left(s, \mathbf{x}_0\right), \;\; s \in [0, \Delta t),$$
$$\left(\bar{\psi}_{\Delta t}\right)^k := \underbrace{\bar{\psi}_{\Delta t} \circ \cdots \circ \bar{\psi}_{\Delta t}}_{k \text{ times}}, \;\; k \in \mathbb{N}.$$

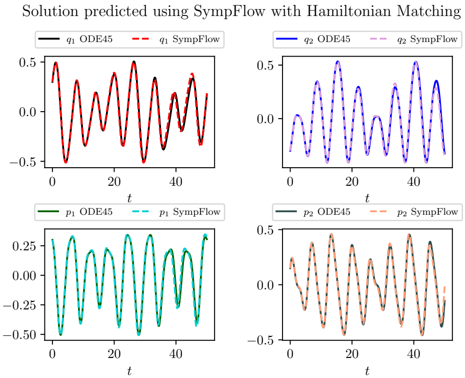
- $\psi(t, \cdot) = \phi_{H,t}(\cdot)$ for the piecewise continuous Hamiltonian

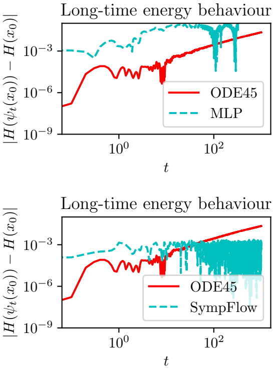
$$H(t, \mathbf{x}) := \mathcal{H}(\mathcal{N}_\theta)\left(t - \Delta t\lfloor t/\Delta t\rfloor, \mathbf{x}\right).$$

Hénon–Heiles (unsupervised)

Equations of motion

$$\dot{x} = p_x, \;\; \dot{y} = p_y, \;\; \dot{p}_x = -x - 2xy, \;\; \dot{p}_y = -y - (x^2 - y^2).$$





Imposing structure over a neural network

- To build networks satisfying a desired property, we can either restrict the parametrisation $\mathcal{N}_\theta$ or modify the loss function.

Imposing structure over a neural network

- To build networks satisfying a desired property, we can either restrict the parametrisation $\mathcal{N}_\theta$ or modify the loss function.

- **Restrict the architecture**:

$$\mathcal{N}_\theta(\mathbf{x}) = \frac{\widetilde{\mathcal{N}}_\theta(\mathbf{x})}{\left\|\widetilde{\mathcal{N}}_\theta(\mathbf{x})\right\|_2} \|\mathbf{x}\|_2 .$$

- **Modify the loss function**:

$$\widetilde{\mathcal{L}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \|\mathcal{N}_\theta(\mathbf{x}_i) - \mathbf{y}_i\|_2^2 + \underbrace{\frac{1}{N}\sum_{i=1}^{N} \left(\|\mathbf{x}_i\|_2 - \|\mathcal{N}_\theta(\mathbf{x}_i)\|_2\right)^2}_{\text{regulariser}} .$$

Imposing structure over a neural network

- To build networks satisfying a desired property, we can either restrict the parametrisation $\mathcal{N}_\theta$ or modify the loss function.

- **Restrict the architecture**:

$$\mathcal{N}_\theta(\mathbf{x}) = \frac{\widetilde{\mathcal{N}}_\theta(\mathbf{x})}{\left\|\widetilde{\mathcal{N}}_\theta(\mathbf{x})\right\|_2} \|\mathbf{x}\|_2 \,.$$

- **Modify the loss function**:

$$\widetilde{\mathcal{L}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \|\mathcal{N}_\theta(\mathbf{x}_i) - \mathbf{y}_i\|_2^2 + \underbrace{\frac{1}{N}\sum_{i=1}^{N} \left(\|\mathbf{x}_i\|_2 - \|\mathcal{N}_\theta(\mathbf{x}_i)\|_2\right)^2}_{\text{regulariser}} \,.$$

- Not all restrictions are equally effective, e.g. $\mathcal{N}_R(\mathbf{x}) = R\mathbf{x}$, $R^\top R = I_d$, is norm-preserving but probably not expressive enough.

Approximation properties

- The inductive bias provided by modelling the network starting from dynamical systems, allows us to study these models using the theory of numerical analysis and dynamical systems.

Universal approximation theorem

Let $F : \Omega \subset \mathbb{R}^d \to \mathbb{R}^d$ be a continuous function, with $\Omega \subset \mathbb{R}^d$ a compact set. Then, for every $\varepsilon > 0$, there exists a finite set of gradient vector fields $\nabla V^1, \cdots, \nabla V^L$, sphere-preserving vector fields $X_S^1, \cdots, X_S^L$, and time steps $h_1, \cdots, h_L \in \mathbb{R}$ such that

$$\left\| F - \Psi_{\nabla V^L}^{h_L} \circ \Psi_{X_S^L}^{h_L} \circ \ldots \circ \Psi_{\nabla V^1}^{h_1} \circ \Psi_{X_S^1}^{h_1} \right\|_{L^p(\Omega)} < \varepsilon.$$

Idea of the proof

Let $F : \Omega \subset \mathbb{R}^d \to \mathbb{R}^d$ be a continuous function, with $\Omega \subset \mathbb{R}^d$ a compact set. Then, for every $\varepsilon > 0$, there exists a finite set of $\mathcal{C}^1$ vector fields $X^1, ..., X^L$, and time steps $h_1, \cdots, h_L \in \mathbb{R}$ such that[a]

$$\left\| F - \Psi^{h_L}_{X^L} \circ \ldots \circ \Psi^{h_1}_{X^1} \right\|_{L^p(\Omega)} < \varepsilon.$$

[a] Qianxiao Li, Ting Lin, and Zuowei Shen. "Deep learning via dynamical systems: An approximation perspective". In: Journal of the European Mathematical Society 25.5 (2022), pp. 1671–1709

Presnov decomposition

For any $X \in \mathcal{C}^1(\mathbb{R}^d, \mathbb{R}^d)$ there is a unique function $U : \mathbb{R}^d \to \mathbb{R}$ with $U(0) = 0$, and a unique sphere-preserving vector field $X_S : \mathbb{R}^d \to \mathbb{R}^d$ such that

$$X(\mathbf{x}) = \nabla U(\mathbf{x}) + X_S(\mathbf{x}), \ \forall \mathbf{x} \in \mathbb{R}^d.$$