An Introduction to Neural Differential Equations

Seminar II

Jyotirmaya Shivottam

23226001

School of Computer Sciences

NISER, HBNI

September 09, 2024

Outline

- Neural Differential Equations (NDEs)
- Differential Equations $\longleftrightarrow$ Deep Learning
- NODE $\leftrightarrow$ ResNet: Performance
- Examples: Hamiltonian & Lagrangian Neural Networks
- *Aside*: Physics-Informed Neural Networks
- Universal Differential Equations (UDEs)
- Manifold Hypothesis
- Universal Approximation
- Practical Considerations
- Limitations
- Conclusion
- References

Neural Differential Equations (NDEs)

- Differential equations, where the vector field is parameterized by a neural network, e.g. a **neural ODE (NODE)** [2]:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = f_\theta(t, \mathbf{x}(t)),\ \mathbf{x}(0) = \mathbf{x}_0$$

- The vector field, $f_\theta : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$ is any neural network, e.g., $\text{FeedForward}, \text{Conv}$ etc.
- The parameters are optimized by backpropagating through the ODE solve.

$$\boxed{\mathcal{L}(\mathbf{x}(t)) = \mathcal{L}(\text{ODESolve}\,(\mathbf{x}(0), t, f_\theta))}$$

- Here, $\mathcal{L}$ is a typical loss function, e.g., MSE, RMSE etc.
- NDEs include types like **Neural ODEs** (above), **Neural CDEs**, **Neural SDEs**, and more, each using a different type of differential equation.
- NDEs bridge Deep Learning (DL) & Differential Equations (DE), allowing us to *leverage the rich history of numerical DE solvers for DL tasks*.

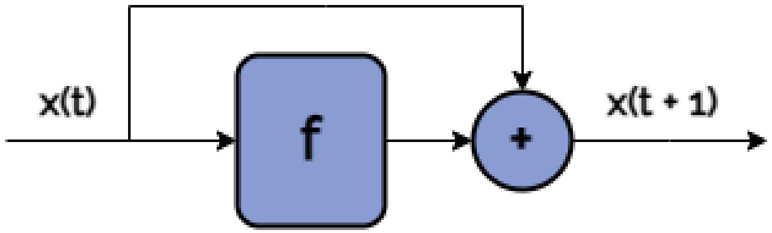
DE $\leftrightarrow$ DL

- A specific example from the DE side — the **SIR model** for infectious diseases:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} S(t) \\ I(t) \\ R(t) \end{pmatrix} = \begin{pmatrix} -\beta S I \\ \beta S I - \gamma I \\ \gamma I \end{pmatrix} \qquad \longleftrightarrow \qquad \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = f_\theta(t, \mathbf{x}(t)),\ \mathbf{x}(0) = \mathbf{x}_0$$

- From the Deep Learning side — the **ResNet** [He et al., 2015]:

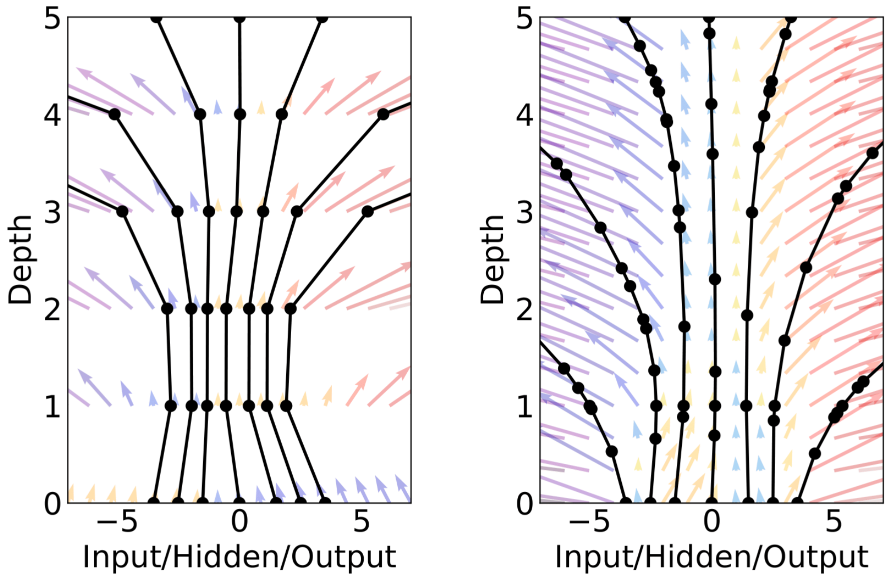
$$\mathbf{x}_{t+1} = \mathbf{x}_t + f_{\theta_t}(\mathbf{x}_t), \quad \text{where } t \in \{0, \ldots, T\} \text{ and } \mathbf{x}_t \in \mathbb{R}^d$$



- Residual Networks (ResNets) use **skip connections** to enable training ***very deep networks***. The core idea is to learn the **residual** function, f_{θ_t}.
- These represent a **Euler discretization** of a **continuous dynamical system** or an **Initial Value Problem** with a unit step size.

DE $\leftrightarrow$ DL: Residual Networks

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = f_\theta(t, \mathbf{x}), \quad \mathbf{x}(0) = \mathbf{x}_0 \implies \boxed{\mathbf{x}_{t+1} = x_t + \int_t^{t+1} f_{\theta_\tau}(\mathbf{x}(\tau))\mathrm{d}\tau \simeq \mathbf{x}_{t+1} = \mathbf{x}_t + f_{\theta_t}(\mathbf{x}_t)}$$



Source: NODE, Chen et al. [2]

- ‼ *If a basic Euler integrator can perform so well, a more advanced scheme with a more accurate approximation might perform better* $\longrightarrow$ also motivates continuous DL models.

NODE $\leftrightarrow$ ResNet: Performance

Table 1: Performance on MNIST. †From LeCun et al. (1998).

	Test Error	# Params	Memory	Time
1-Layer MLP†	1.60%	0.24 M	-	-
ResNet	0.41%	0.60 M	$\mathcal{O}(L)$	$\mathcal{O}(L)$
RK-Net	0.47%	0.22 M	$\mathcal{O}(\tilde{L})$	$\mathcal{O}(\tilde{L})$
ODE-Net	0.42%	0.22 M	$\boldsymbol{\mathcal{O}(1)}$	$\mathcal{O}(\tilde{L})$

Source: NODE, Chen et al. [2]

Example: Hamiltonian Neural Networks

- Proposed by Greydanus et al. [4], **Hamiltonian Neural Networks (HNNs)** are a specific case of NDEs.
- **Assumption**: Observed dynamics follow a Hamiltonian, $H(\mathbf{q}(t), \mathbf{p}(t))$ (e.g., physical **conservation** laws).

$$\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} = \frac{\partial H}{\partial \mathbf{p}} \qquad \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = -\frac{\partial H}{\partial \mathbf{q}}$$

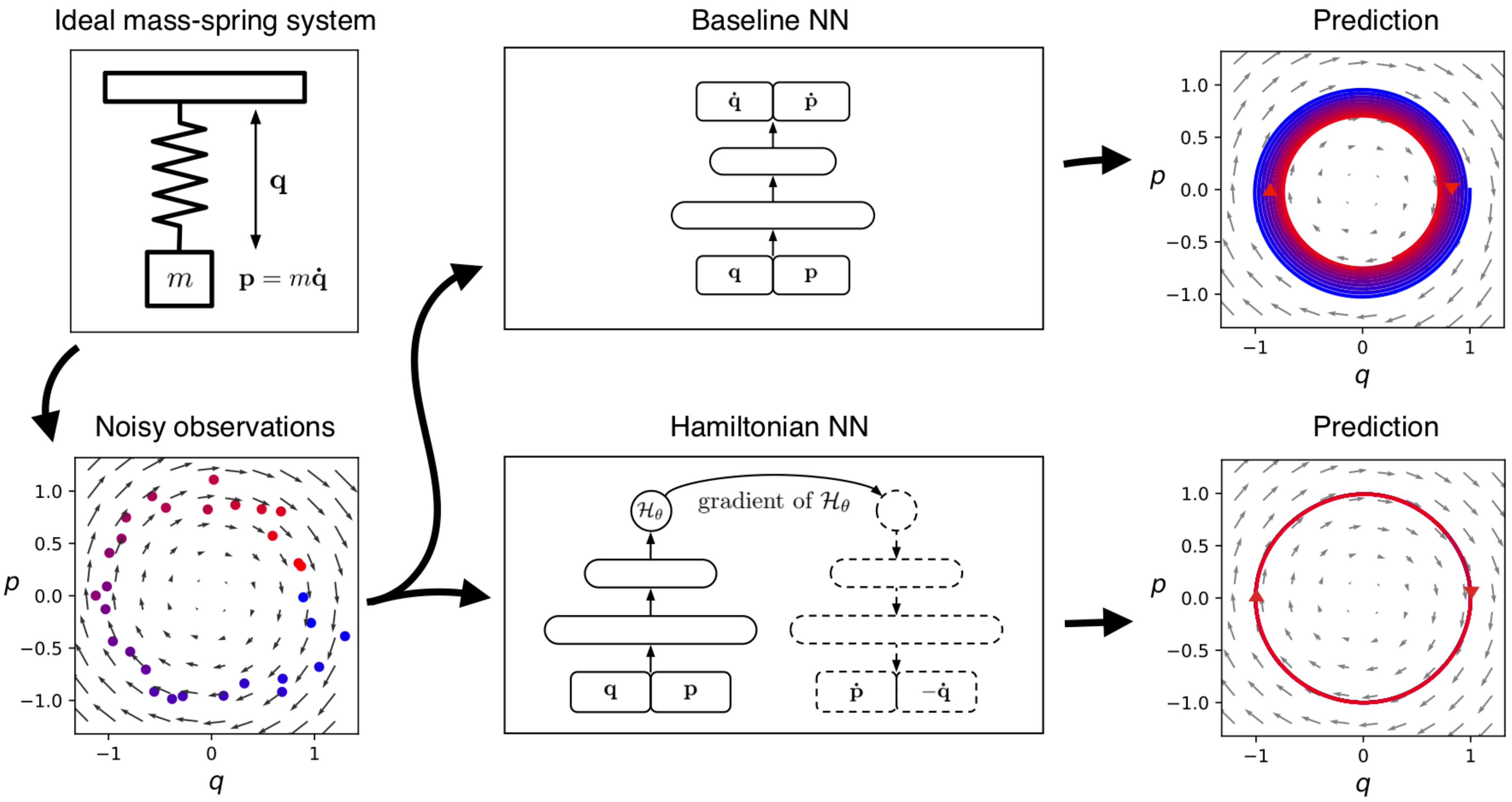
- Parameterize and learn H using a NN, $H_\theta(\mathbf{q}, \mathbf{p})$, in **canonical coordinates**.
- Can be extended by component-wise parameterization into the Mass matrix, M_θ, and Potential Energy, V_θ.

$$H_\theta(\mathbf{q}, \mathbf{p}) = \frac{1}{2}\mathbf{p}^T M_\theta^{-1} \mathbf{p} + V_\theta(\mathbf{q})$$

- We can even include additional control terms (β) in the dynamics:

$$\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} = \frac{\partial H}{\partial \mathbf{p}} \qquad \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = -\frac{\partial H}{\partial \mathbf{q}} + g_\theta(\mathbf{q})\beta(\mathbf{q})$$

Example: Hamiltonian Neural Networks



Source: HNN, Greydanus - https://greydanus.github.io/2019/05/15/hamiltonian-nns/

Example: Lagrangian Neural Networks

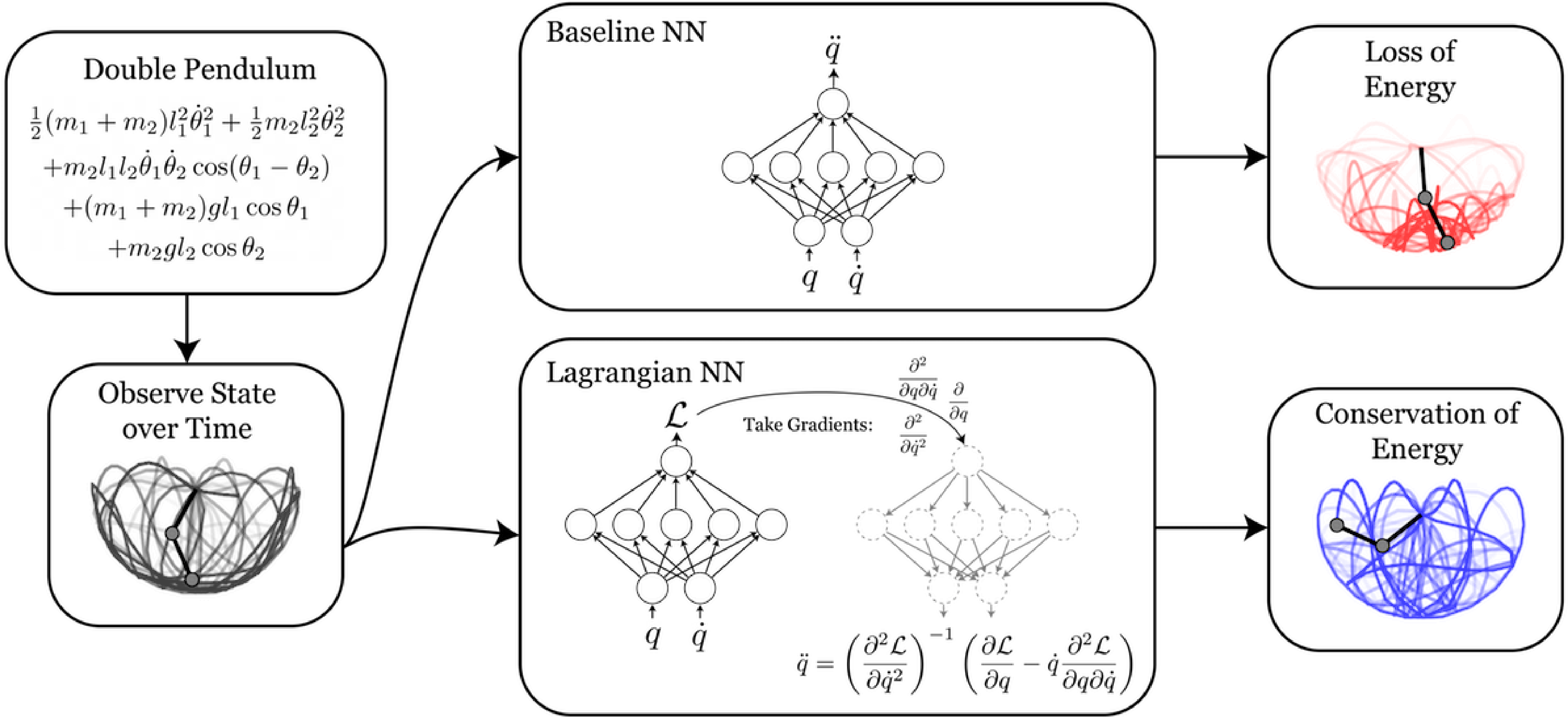
- ⚠️ Observed data may not possess a "*Hamiltonian structure*" (**Symplectomorphism**).
- Proposed by Cranmer et al. [5], **Lagrangian Neural Networks** (LNNs) relax the need for such a structure.
- Here, the Lagrangian, $\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}})$, is parameterized by a NN, $\mathcal{L}_\theta(\mathbf{q}, \dot{\mathbf{q}})$.

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}_\theta}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial \mathcal{L}_\theta}{\partial \mathbf{q}} = 0 \implies \boxed{\ddot{\mathbf{q}} = \left(\frac{\partial^2 \mathcal{L}_\theta}{\partial^2 \dot{\mathbf{q}}}\right)^{-1} \left(\frac{\partial \mathcal{L}_\theta}{\partial \mathbf{q}} - \frac{\partial^2 \mathcal{L}_\theta}{\partial \mathbf{q} \partial \dot{\mathbf{q}}} \dot{\mathbf{q}}\right)} \quad \text{(generalized coordinates)}$$

- *LNNs turn out to be easier to work with than HNNs, which is inverse to the case in Physics.*
- HNNs & LNNs and several other ***NDEs inculcate inductive biases (e.g., physical)***.
- ❗**BUT:** We are not limited to these two examples; many other types of NDEs exist, each with its own assumptions and constraints.

$$\begin{aligned} \text{RNN/GRU/LSTM} &\longleftrightarrow \text{Neural Controlled DEs (NCDEs)} \\ \text{Continuous VAE/GAN} &\longleftrightarrow \text{Neural Latent DEs / Stochastic DEs (NSDEs)} \\ \text{Invertible networks, Normalizing Flows} &\longleftrightarrow \text{Reversible DEs} \end{aligned}$$

Example: Lagrangian Neural Networks



Source: LNN, Greydanus - https://greydanus.github.io/2020/03/10/lagrangian-nns/

Aside: Physics-Informed Neural Networks

- Physics-Informed Neural Networks (PINNs) regularize the training of NNs by incorporating known physical laws.
- They also numerically approximate the same ODE as NODEs, but by representing the solution ($\mathbf{x}(t)$) **as a NN**...

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = f(t, \mathbf{x}), \ \ f \text{ known}$$

- ...and by minimizing a loss function with a **physics-based term** that enforces the solution to satisfy the given ODE.

$$\min_\theta \frac{1}{N} \sum_{i=1}^{N} \left|\left| \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}(t_i) - f(t_i, \mathbf{x}_\theta(t_i)) \right|\right|$$

- ‼ This is ***distinct*** from NDEs. NDEs use neural networks to define DEs $\longleftrightarrow$ PINNs uses neural networks to solve pre-defined DEs with physics-based constraints $\longrightarrow$ **different goals**.

Universal Differential Equations (UDEs)

- *Known* aspects of the system are **hard-coded** as **prior structural information** to guide learning.

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = f(t, \mathbf{x}, U_\theta)$$

- Example: **Lotka-Volterra** model of predator-prey dynamics, where the interaction terms are known. $x(t),\ y(t) \in \mathbb{R}$ are the prey and predator populations, respectively.

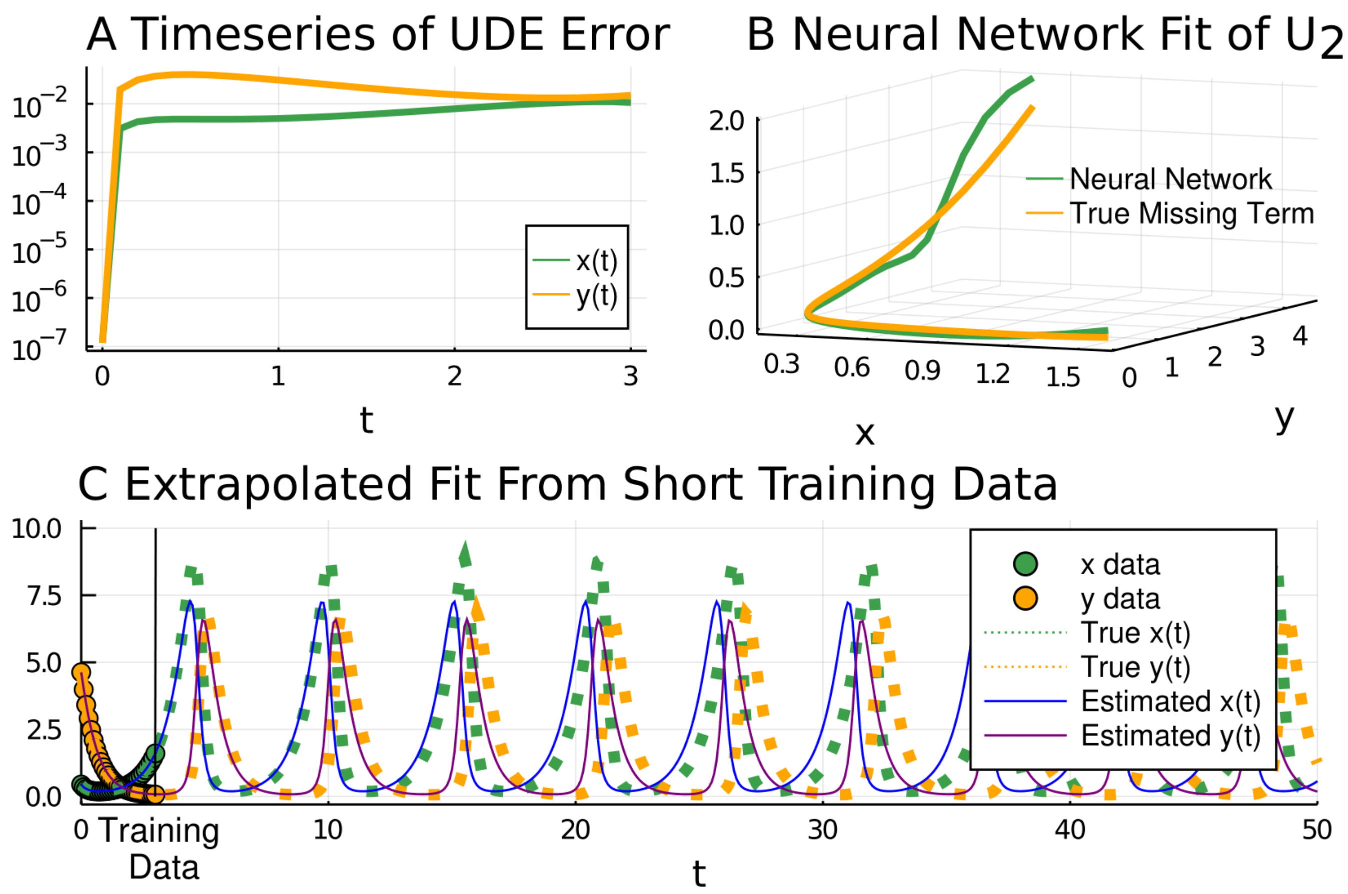
$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha x - \beta xy \quad \frac{\mathrm{d}y}{\mathrm{d}t} = \delta xy - \gamma y, \quad \text{where } \alpha, \beta, \gamma, \delta \text{ are known.}$$

- This theory is often imperfect, with gaps between predictions and observations. To address this, we can introduce neural networks $f_\theta,\ g_\theta : \mathbb{R}^2 \to \mathbb{R}$:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha x - \beta xy + f_\theta(x, y) \quad \frac{\mathrm{d}y}{\mathrm{d}t} = \delta xy - \gamma y + g_\theta(x, y)$$

- $f_\theta,\ g_\theta$ are trained to capture the unknown dynamics of the system $\longrightarrow$ **Universal** DEs.

Universal Differential Equations (UDEs)



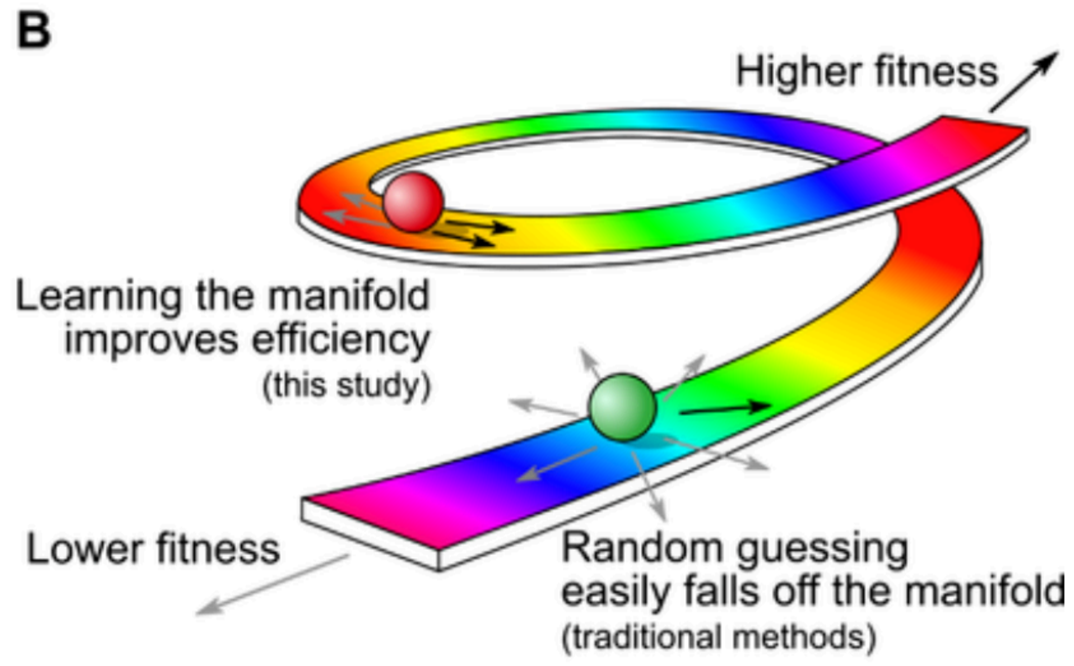
Source: UDE, Rackauckas et al. [3]

Universal Differential Equations (UDEs)

- ‼ Learnt theoretical parameters *may not correspond* to physical quantities $\longrightarrow$ First, fit the theoretical model fixing these parameters, then train only the NN.
- There are also other ways to handle this, like **norm regularization**, **scheduling**, etc.
- **Flexibility-Constraint trade-off**: More flexible the model, the less constrained it is by the theory, and vice-versa.
- UDEs provide a natural approach when modeling complex, *poorly understood behavior* with **sufficient data** ($<<$ what is required for a purely data-driven approach).
- UDEs utilize different solver choices to ensure stability particularly while solving PDEs.
- They are also applicable to SDEs and other types of differential equations $\leftrightarrow$ a **generalization** of the NDE framework.
- *Aside*: Since only a part is learned, UDEs are more interpretable than purely data-driven models. *Using Symbolic Regression (or SINDy), it is possible to extract the learned equations*.

Manifold Hypothesis

- The **Manifold Hypothesis** asserts that real-world data exists on or near a **low-dimensional manifold within a high-dimensional space**, a concept crucial for (non-linear) dimensionality reduction and other machine learning techniques.



Source: Hie et al. - https://www.nature.com/articles/s41587-023-01763-2

Manifold Hypothesis

- This *heuristic* helps bypass the curse of dimensionality, enabling a more compact data representation, with ***fewer samples required for learning*** $\longrightarrow$ **continuous modeling** and **structure learning**.
- A model's ability to generalize often depends on how well it can **interpolate** or **extrapolate** on the data's **underlying structure or manifold**.
- NODEs learn well on data that lies on a differentiable manifold $\longrightarrow$ NODEs learn **diffeomorphisms**.
- This provides a way to introduce various **symmetries** and **constraints** into the model.
- ❓ Does this mean NDEs are ***always the better choice***? More concretely, are they **more expressive**?
- ‼️ Turns out, **not necessarily**. Discrete models (ResNets) are generally more expressive. Not to mention, they are easier to train.

Universal Approximation

- *Basic NDEs like NODEs are not universal approximators* $\longrightarrow$ NODEs cannot represent a large class of (non-linear) functions:

$$g(\mathbf{x}) = \begin{cases} -1 & \text{if } ||\mathbf{x}|| < r_1 \\ 1 & \text{if } r_2 < ||\mathbf{x}|| < r_3 \end{cases}$$

- Here, $g : \mathbb{R}^d \to \mathbb{R}$ is a parity-like function and $0 < r_1 < r_2 < r_3$.
- The feature mapping is homeomorphic, i.e., topology-preserving. As phase-space trajectories for an ODE cannot intersect (via Uniqueness Theorem & recall: NODEs learn Diffeomorphisms), linear separation of the input space is impossible.
- **Solution**: Augmented Neural ODEs (ANODEs) solve in $\mathbb{R}^{d+p}$, lifting points into additional dimensions, where they can be separated.

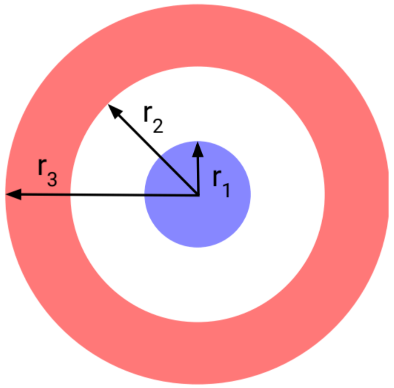
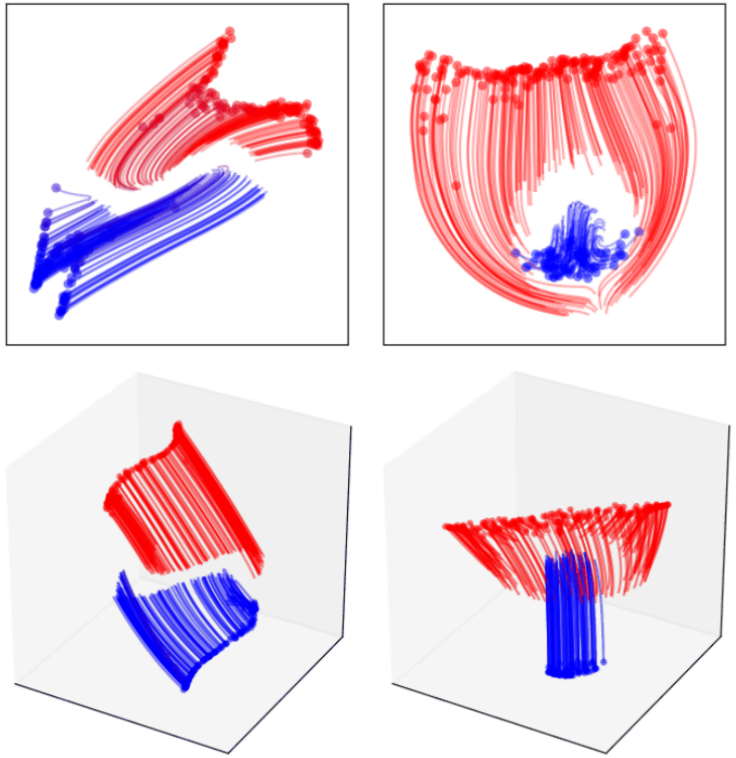


Figure 4: Diagram of $g(\mathbf{x})$ for $d = 2$.

Universal Approximation – ANODEs

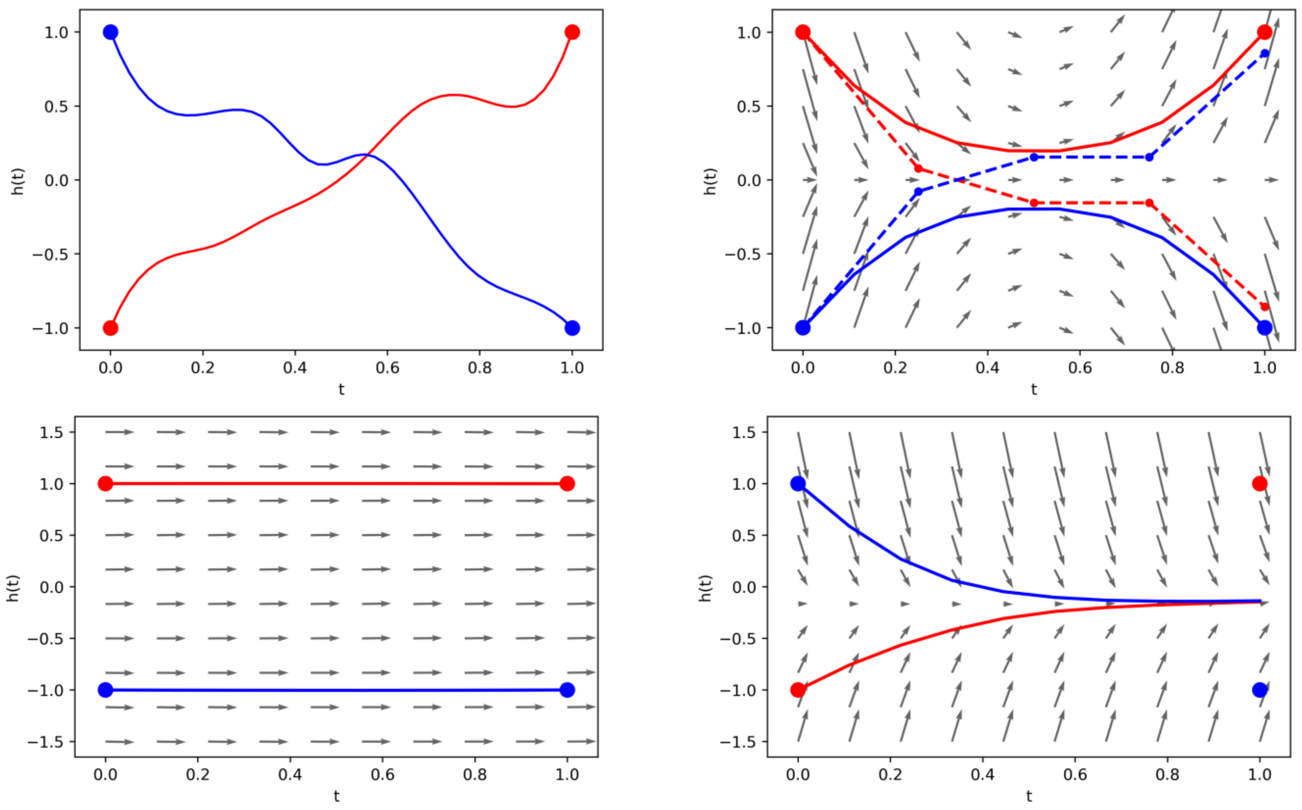
- ANODEs avoid the issue of non-intersecting trajectories, making them universal approximators. Alternatively, use Second Order NODEs (SONODE) (arxiv:2006.07220).



Source: ANODE, Dupont et al. [6]

Universal Approximation – The ResNet Conundrum

- ‼ *ResNets are so expressive, because the Euler integrator accumulates error*.



Source: ANODE, Dupont et al. [6]

Practical Considerations

- **Choice of NN (f_θ)**: To align with structure learning, f_θ can incorporate inductive biases based on the data, like translation-invariance (CNNs) or permutation-equivariance (GNNs). To ensure well-posedness, f_θ should be Lipschitz continuous.
- **ODE Solver**: The choice of solver impacts performance, especially for stiff systems. Fortunately, many options are available in DE literature.
- **Initialization**: Both random and zero-initializations for f_θ work, but they can influence convergence speed and generalization.
- **Activation Functions**: For BPTT through the ODE solver, activation functions should be Lipschitz continuous (e.g., $\tanh$, σ). While $\mathrm{ReLU}(\cdot)$ is not, it works in practice. Alternatives like $\mathrm{Swish}(\cdot)$ can also be used.
- Moreover, there are different schemes such as discretize-then-optimize, optimize-then-discretize to manage training efficiency and stability.
- Given the use of DE solvers, the tolerance (error-budget) can be specified to control the trade-off between accuracy and computation time.

Limitations

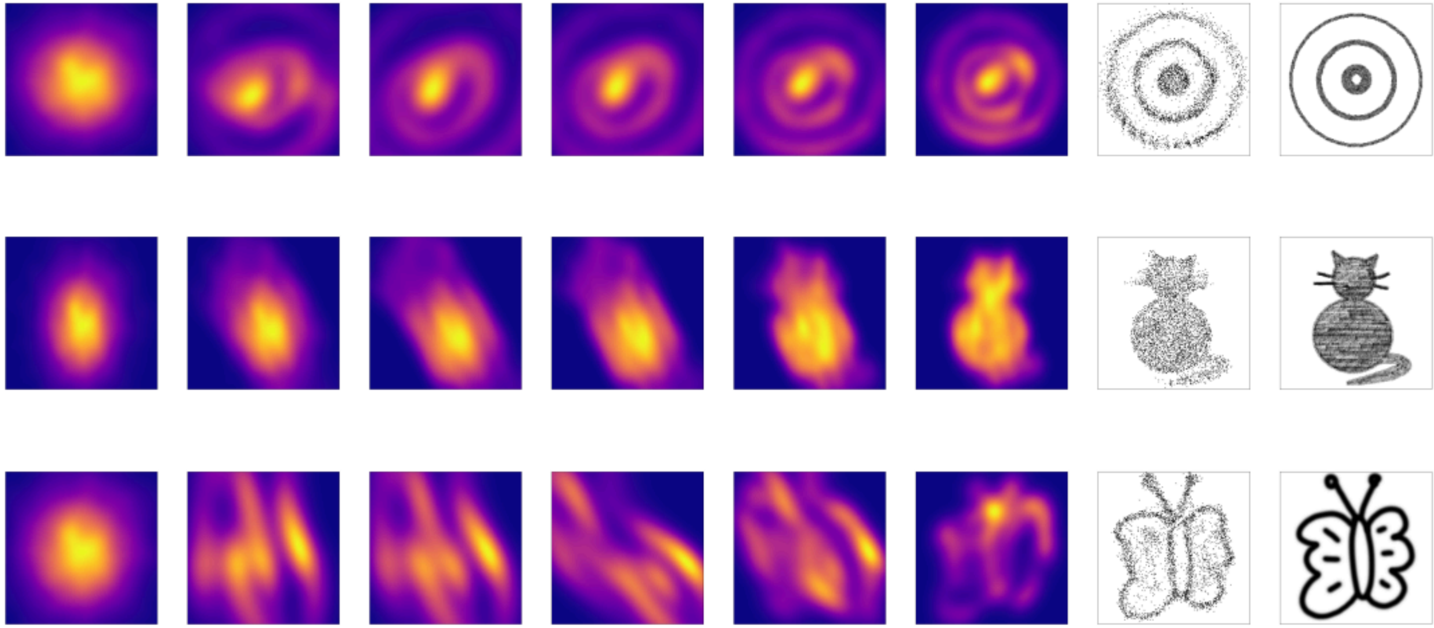
- NDEs are a powerful framework, *but they are not a panacea*.
- For most explicitly discrete tasks like classification on Vision datasets, discrete models are easier to train, though less interpretable.
- Training NDEs at scale can be memory-intensive[‡‡], and error accumulation is a concern $\longrightarrow$ Use **reversible DEs** when possible.
- Matching an error / tolerance budget often requires trial and error.
- **"Depth"** is undefined[‡‡] in basic NDEs, though depth-awareness can be introduced.
- In time-series modeling, NODEs struggle[‡‡] with irregularly spaced data $\longrightarrow$ Extensions like **NCDEs** are needed.
- For modeling dynamic interactions, extensions like **NSDEs** or **Latent ODEs** are **required**, as NODEs only randomize the initial state, ***missing dynamic interactions***.
- ‡‡ **Caveat**: These issues are mostly for basic NDEs like NODEs. Many have since been resolved to a great degree.

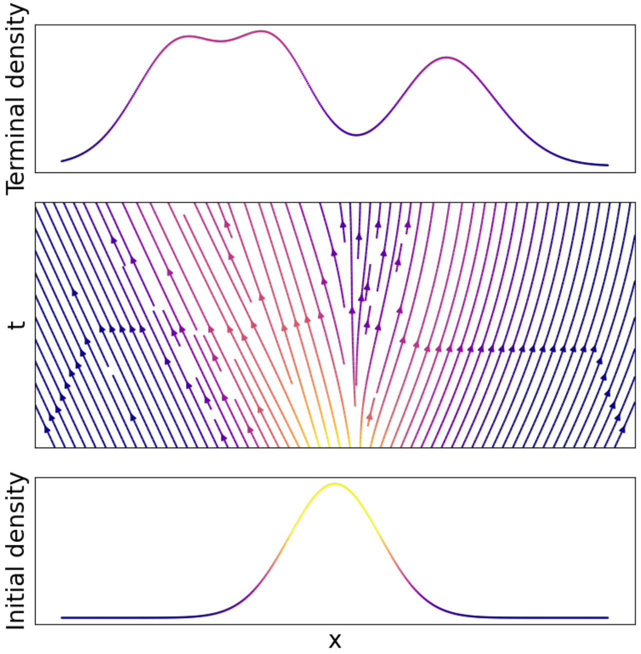
Conclusion

- NDEs are a powerful & efficient tool for **imbuing domain knowledge** in neural networks, enabling them to **learn from less data** and **generalize better**.
- They allow us to use the rich repertoire of numerical DE solvers for DL tasks, with features such as **constant memory cost**, **continuos-depth models**, **adaptive computation** and a means to **quantify uncertainties and approximation errors**.
- ! NDEs provide a **natural basis** for **manifold learning**.
- ! Augmented NDEs are **universal approximators** and are strictly **more expressive** than (Euler-)discretized ResNets.
- ! Basic NDEs that learn diffeomorphisms are suited to **non-intersecting manifold learning** (e.g., shapes. See PointFlow - arxiv:1906.12320)
- NDEs find use in a variety of fields, from **sequence modeling** to **generative modeling** → **Continuous Normalizing Flows** are a prime use-case.

Conclusion

- $\mathcal{O}(d^2)$ complexity for CNFs vs $\mathcal{O}(d^3)$ complexity for NFs.





Source: On NDEs, Kidger P. [1]

References

1. On Neural Differential Equations
2. Neural Ordinary Differential Equations
3. Universal Differential Equations for Scientific Machine Learning
4. Hamiltonian Neural Networks
5. Lagrangian Neural Networks
6. Augmented Neural ODEs

Additional Material

1. Dissecting Neural ODEs - depth-aware NODEs.
2. Deep Equilibrium Models - implicit; uses fixed points of deep ResNets.
3. YouTube - On Neural Differential Equations - Patrick Kidger
4. YouTube - Follow up Neural ODEs @ NeurIPS 2019
5. YouTube - Neural ODEs - Steve Brunton

Thank you! Any questions?

_js.01