

Physics-informed Machine Learning for Dynamical Systems

Samia Islam

islamsa3@msu.edu

Department of Computer Science and Engineering
Michigan State University

February 10, 2025

Topics Covered

1. Hamiltonian Neural Network (HNN)
2. Physics Informed Neural Network (PINN)
3. Neural Ordinary Differential Equation (Neural ODE)

Dynamical System in Physics

- Physics:
 - The study of matter, energy, motion, and force in the nature
 - Understanding how systems evolve over time and respond to external forces
- Dynamical System:
 - A system where the state changes over time according to specific rules
 - Usually expressed using mathematical models, particularly differential equations
 - Example: motion of planets, waves, gene expression
 - $x(t)$ = state of the system at time t
 - $f(x, t)$ = dynamical system
 - $\frac{dx}{dt}$ = how the state changes over time

$$\frac{dx}{dt} = f(x, t)$$

Physics-informed Machine Learning (PiML)

- Integrates known physical principles into neural networks:
 - Improves robustness
 - More interpretable
 - Efficient with less data
- Physics can be integrated at various ML stages^[1]:
 - **Problem formulation:** Frame variables in terms of physical laws
 - **Data collection:** Use domain knowledge to collect data
 - **Architecture modification:** Embed physics into the model
 - **Loss function design:** Penalize violations of physical laws
 - **Optimization:** Ensure optimization respects physical constraints

Ordinary Differential Equation (ODE) in Physics

- Describes systems evolving over time with respect to a single variable
- Solution to an ODE: State of the system at time t
- Example: Mass-spring system
 - m = mass of the object, q = position of the object, p = momentum, k = spring constant
 - The system evolves according to the law:
$$m \frac{d^2 q}{dt^2} = -kq$$
 - The momentum is defined as: $p = m \frac{dq}{dt} \Rightarrow \frac{dq}{dt} = \frac{p}{m}$
 - $p = m \frac{dq}{dt} \Rightarrow \frac{dp}{dt} = m \frac{dv}{dt} \Rightarrow \frac{dp}{dt} = -m \frac{k}{m} q \Rightarrow \frac{dp}{dt} = -kq$
 - So the system now becomes:

$$\frac{dq}{dt} = \frac{p}{m}, \frac{dp}{dt} = -kq$$

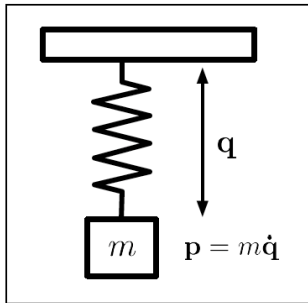


Figure: A simple mass-spring system^[2]

Hamiltonian Mechanics

- Describes the evolution of a system using energy-based principles
- Formulates the system's evolution in phase space:
 - Phase space contains the coordinates q (position) and p (momentum)
 - The state of a dynamical system is recorded as a trajectory in phase space
- The Hamiltonian represents the total energy of the system: $H(q_i, p_i, t) = T + V$
 V = potential energy, T = kinetic energy
- The system dynamics are governed by the Hamiltonian equations:

$$\frac{dq}{dt} = \frac{\delta H}{\delta p}, \frac{dp}{dt} = -\frac{\delta H}{\delta q}$$

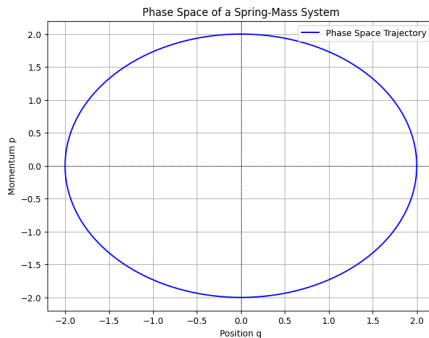
- Ensures energy conservation

Hamiltonian in Mass-spring System

- The system is defined as:
 $\frac{dq}{dt} = \frac{p}{m}, \frac{dp}{dt} = -kq$
- The Hamiltonian for the mass-spring system:

$$\mathcal{H}(q, p) = \frac{p^2}{2m} + \frac{1}{2}kq^2$$

- Hamilton's equations: $\frac{dq}{dt} = \frac{\delta H}{\delta p}, \frac{dp}{dt} = -\frac{\delta H}{\delta q}$
- $\frac{\delta H}{\delta p} = \frac{\delta}{\delta p} \left(\frac{p^2}{2m} + \frac{1}{2}kq^2 \right) = \frac{p}{m}$, thus $\frac{dq}{dt} = \frac{p}{m}$
- $\frac{\delta H}{\delta q} = \frac{\delta}{\delta q} \left(\frac{p^2}{2m} + \frac{1}{2}kq^2 \right) = kq$, thus
 $\frac{dp}{dt} = -kq$



Hamiltonian Neural Network (HNN)^[2]

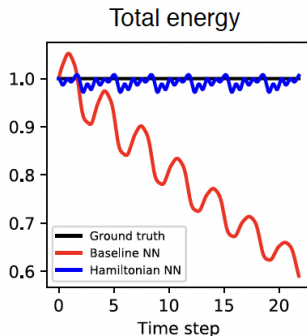
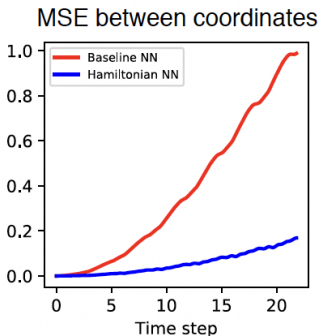
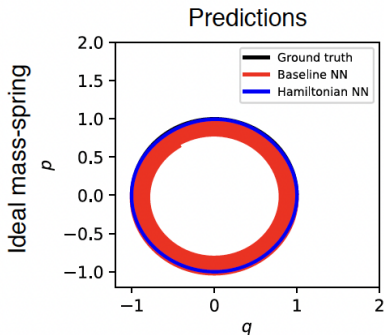
- Learns $H(q, p)$ directly from data
- The loss function:

$$\mathcal{L}_{HNN} = \left\| \frac{\delta H_\theta}{\delta p} - \frac{dq}{dt} \right\|_2 + \left\| \frac{\delta H_\theta}{\delta q} + \frac{dp}{dt} \right\|_2$$

- Penalizes deviations from the Hamiltonian equations: $\frac{dq}{dt} = \frac{\delta H}{\delta p}$, $\frac{dp}{dt} = -\frac{\delta H}{\delta q}$
- Input: Observations of the system's state over time
 - Position: q
 - Momentum: p
- Output: The Hamiltonian H_θ
 - H_θ is used to compute $\frac{d\hat{q}}{dt}$ and $\frac{d\hat{p}}{dt}$
 - From $\frac{d\hat{q}}{dt}$ and $\frac{d\hat{p}}{dt}$, $\hat{p}$ and $\hat{q}$ can be computed by Runge-Kutta method
- $\frac{\delta H_\theta}{\delta p}$ and $\frac{\delta H_\theta}{\delta q}$ for loss function can be computed by automatic differentiation

Performance of HNN^[2]

- Same architecture for HNN and baseline
- HNN uses $\mathcal{L}_{HNN}$
- Baseline directly outputs $\frac{d\hat{q}}{dt}$ and $\frac{d\hat{p}}{dt}$
- HNN outputs H_θ from where we can compute $\frac{d\hat{q}}{dt}$ and $\frac{d\hat{p}}{dt}$



Learning Hamiltonian from Pixels

- Autoencoder can be used
- An encoder encodes pixel data into a latent space $z = (z_q, z_p)$
 - z_q approximates q , z_p approximates p
- The HNN operates on the latent space and learns $H(q, p)$
- Loss functions:
 - $\mathcal{L}_{HNN}$: Hamiltonian loss
 - $\mathcal{L}_{AE}$: reconstruction loss for the autoencoder
 - $\mathcal{L}_{CC} = \|z_p^t - (z_q^t - z_q^{t+1})\|_2$
 - Encourages the latent space variables to behave like canonical coordinates
 - $z_q^t - z_q^{t+1}$ approximates the velocity or the rate of change of position
 - z_p^t corresponds to the velocity
 - $\mathcal{L}_{CC}$ penalizes the difference between the two
- Project the output from HNN back to the pixel space

Ground truth



Hamiltonian NN



Strengths and Limitations of HNN

- **Strengths:**

- Can learn the information of conserving energy
- Reversible dynamics: we can simulate trajectories backward in time
- Learns better than regular neural networks in physics-based tasks

- **Limitations:**

- Not suitable for complex systems with non-conserved forces
- Works Only for Hamiltonian Systems, cannot be directly applied to general PDEs
- Less scalable for complex systems, e.g., three-body problems

How can we address these problems and generalize for any system?

How can we address these problems and generalize for any system?

Physics Informed Neural Network (PINN)^[3]

Partial Differential Equation (PDE) in Physics

- Describes how the system evolves over multiple variables (e.g., both time and space)
- General form^[3]: $\frac{\delta u}{\delta t} + \mathcal{N}[u, \lambda] = 0$
- Example: the Burger's equation: a nonlinear PDE commonly used in fluid dynamics

$$\frac{\delta u}{\delta t} + u \frac{\delta u}{\delta x} - \frac{0.01}{\pi} \frac{\delta^2 u}{\delta x^2} = 0; x \in [-1, 1], t \in [0, 1]$$

- $u(t, x)$ = velocity field, ν = viscosity coefficient
- $\frac{\delta u}{\delta t}$ = rate of change of velocity
- $u \frac{\delta u}{\delta x}$ describes how velocity changes in space
- $\frac{0.01}{\pi} \frac{\delta^2 u}{\delta x^2}$ describes how disturbances spread out
- By comparing, $\mathcal{N}[u, \lambda] = u \frac{\delta u}{\delta x} - \frac{0.01}{\pi} \frac{\delta^2 u}{\delta x^2} = 0; x \in [-1, 1], t \in [0, 1]$

mRNA Dynamics as a PDE^[5]

- The dynamics of mRNA in a gene regulatory network:

$$\frac{\delta q}{\delta t} = \alpha - \gamma q + D_q \frac{\delta^2 q}{\delta x^2}$$

- q = mRNA concentration
- α = transcription rate
- γ = degradation rate
- D_q = diffusion coefficient
- $\frac{\delta^2 q}{\delta x^2}$ = the spatial spread of q .

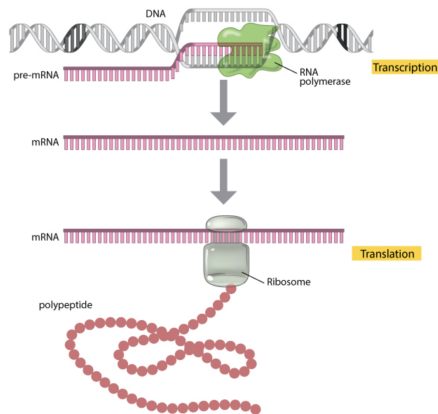


Figure: mRNA life cycle^[4]

Forward Problems of PDE^[3]

- Forward Problem:
 - Given: a known PDE, initial and boundary conditions, and system parameters
 - Want: compute the solution over time or space
- Approximate $u(t, x)$ using a neural network
- Standard PDE: $\frac{\delta u}{\delta t} + \mathcal{N}[u, \lambda] = 0$
- The residual of the PDE: $f = \frac{\delta u}{\delta t} + \mathcal{N}[u, \lambda]$
- By comparing, we get $f = 0$
- Incorporate this information into the loss function: **Physics Informed Neural Network (PINN)**!

Forward Problems of PDE^[3] (cont.)

- A neural network approximates $u(t, x)$
- Automatic differentiation is used to compute $f(t, x)$
- Apply the loss function:
 - Data Constraint (MSE_u):

$$\mathcal{L}_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_i, x_i) - u_i|$$

N_u = number of initial and boundary training data points

- Physics Constraint (MSE_f):

$$\mathcal{L}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_i, x_i)|$$

N_f = number of PDE training data points

- Total Loss: $L = MSE_u + MSE_f$

Solving Burgers' Equation^[3]

- The one-dimensional Burger's equation:

$$\frac{\delta u}{\delta t} + u \frac{\delta u}{\delta x} - \frac{0.01}{\pi} \frac{\delta^2 u}{\delta x^2} = 0; x \in [-1, 1], t \in [0, 1]$$

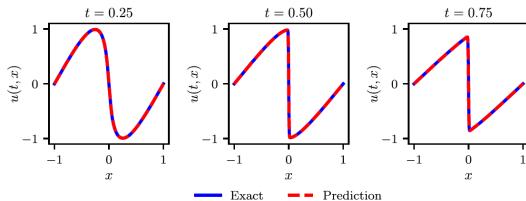
- Given initial condition: $u(0, x) = -\sin \pi x$
- Given boundary condition: $u(t, -1) = u(t, 1) = 0$

- The residual: $f(t, x) = \frac{\delta u}{\delta t} + u \frac{\delta u}{\delta x} - \frac{0.01}{\pi} \frac{\delta^2 u}{\delta x^2}$

- The loss function:

$$\mathcal{L} = \frac{1}{N_x} \sum_{i=1}^{N_x} |u(0, x_i) + \sin \pi x_i| + \frac{1}{N_t} \sum_{i=1}^{N_t} |u(t_i, -1) + u(t_i, 1)| + \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_i, x_i)|$$

- Input: t and x , Output: $u(t, x)$



Inverse Problem of PDE^[3]

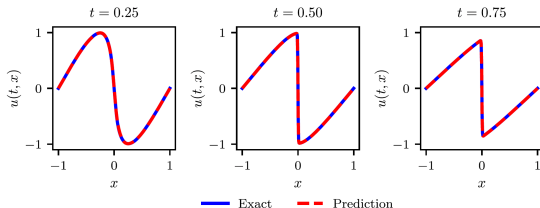
- Inverse Problem:
 - Given: observations of the system's state over time $u(t, x)$, general form of the PDE
 - Want: unknown parameters, the governing equation
- Approximate λ using a neural network:
 - Estimate u_λ and compute the derivative of u_λ with respect to time: $\frac{\delta u_\lambda}{\delta t}$
 - Compute the residual using the derivative and inferred λ : $f(t, x) = \frac{\delta u_\lambda}{\delta t} + \mathcal{N}[u, \lambda]$
 - Enforce the residual during training to respect the physical law described by the PDE
- Same loss function as the forward problem

Estimating Parameters of Burger's Equation^[3]

- Given equation: $\frac{\delta u}{\delta t} + \lambda_1 u \frac{\delta u}{\delta x} - \lambda_2 \frac{\delta^2 u}{\delta x^2} = 0$
- The residual: $f(t, x) = \frac{\delta u}{\delta t} + \lambda_1 u \frac{\delta u}{\delta x} - \lambda_2 \frac{\delta^2 u}{\delta x^2}$
- Try approximating $u(t, x)$ by predicted λ_1 and λ_2 using a deep neural network
- Fit the estimation using the loss function:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N |u(t_u^i, x_u^i) - u^i| + \frac{1}{N} \sum_{i=1}^N |f(t_i, x_i)|$$

- Input: t and x , Output: $u(t, x)$
- λ_1 and λ_2 are learned indirectly through optimization

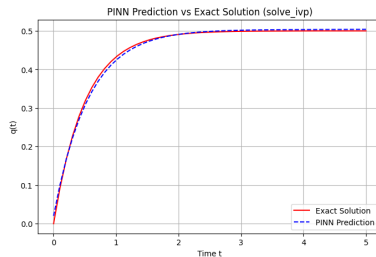


Estimating mRNA Dynamics Parameters

- For simplicity, I considered the diffusion term is negligible: $\frac{\delta q}{\delta t} = \alpha + \gamma q$
- The residual: $f(t, q) = \frac{\delta q}{\delta t} - \alpha - \gamma q$
- Generated toy data q for $\alpha = 1$ and $\gamma = 2$
- Trained a very simple neural network
- Loss function:

$$\mathcal{L}_{mRNA} = \frac{1}{N} \sum_{i=1}^N |q(t_i, \theta) - q_i|^2 + \frac{1}{N} \sum_{i=1}^N \left| \frac{\delta q}{\delta t_i} - \alpha(\theta) + \gamma(\theta)q(t_i, \theta) \right|$$

- Result: $\alpha = 0.892874$, $\gamma = 1.771568$



Strengths and Limitations of PINN

- **Strengths:**

- Incorporates physical laws into machine learning
- Data-efficient: comparatively works with small datasets
- Can solve both forward and inverse problems

- **Limitations:**

- Training with physics constraints and data constraints increases computational overhead
- Relies on collocation points: in higher dimensions the number of points required will grow exponentially
- Works only on loss functions

So far modified only the loss function,
how about the architecture?

So far modified only the loss function,
how about the architecture?

Neural Ordinary Differential Equation
(Neural ODE)^[6]

ODE Solvers

- ODE: describes how a variable $y(t)$ evolves over time based on a $f(y, t)$:
 $\frac{dy}{dt} = f(y, t)$
- The solution $y(t)$ is found by integrating this equation over time
- Initial value problem:
 - Given: initial condition $y(t_0) = y_0$, dynamics $\frac{dy}{dt} = f(y, t)$
 - Want: $y(t)$ where $t > t_0$
- Numerical ODE Solvers: solve the initial value problem numerically
- Euler's Method, Runge-Kutta Method etc.

Euler Discretization

- A numerical method for solving ODEs
- Approximates a solution at discrete time steps:

$$h_{t+1} = h_t + \Delta t \cdot f(h(t), t)$$

- h_t = solution at time step t
- $f(h(t), t)$ = the function defining the system's dynamics
- Δt = time step

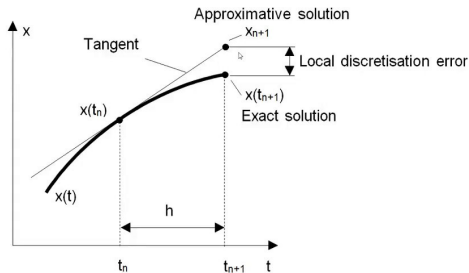


Figure: Euler Discretization^[7]

Residual Network (ResNet)^[8]

- A type of deep neural network that incorporate skip connections
- The hidden state is updated by:

$$h_{t+1} = h_t + f(h_t, \theta_t)$$

- The equation is structurally identical to the Euler discretization formula, where $\Delta t = 1$
- ResNets can be interpreted as a discretized approximation of a continuous dynamical system

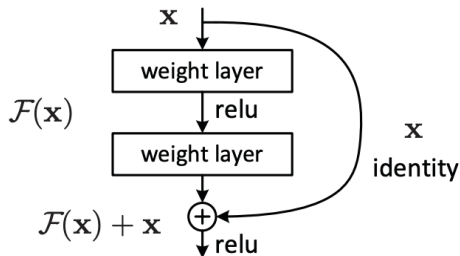


Figure: A Residual Building Block^[8]

What if we use a continuous time model instead of a discretized approximation?

Neural Ordinary Differential Equation (Neural ODE)^[6]

- Replaces discrete layers of ResNet with a continuous-time system
- Treats hidden states as solutions of an ODE

$$\frac{dh}{dt} = f(h(t), t, \theta)$$

- $h(t)$ = hidden state as a continuous function of time t
- $f(h(t), t, \theta)$ = a neural network that defines the dynamics of the system
- θ = the parameters of f
- ODE solvers are treated as black-boxes

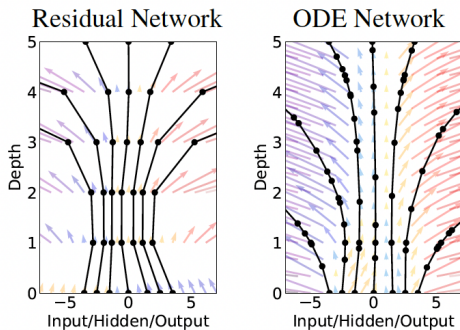


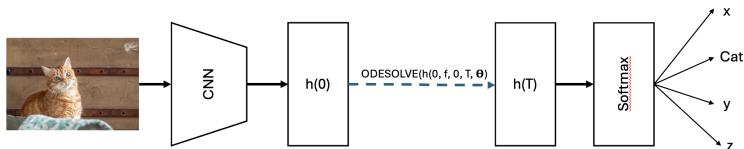
Figure: ResNet treats the system as a discrete model and Neural ODE treats the system as a continuous model^[6]

Neural Ordinary Differential Equation (Neural ODE)^[6] contd.

- Starts with an initial state $h(t_0)$ (may come from the input data)
- $f(h(t), t, \theta)$: a neural network determines the rate of change of the hidden state over time
- A numerical ODE solver integrates $f(h(t), t, \theta)$ to compute the state $h(T)$ at the final time T

$$h(T) = h(t_0) + \int_{t_0}^T f(h(t), t, \theta) dt$$

- $h(T) = \text{ODESolve}(h(t_0), f, t_0, T, \theta)$ [black box]
- The final state $h(T)$ is the output of the network



Backpropagation in Neural ODE^[6]

- Backpropagation through an ODE solver is memory-intensive
- Requires storing intermediate states: high memory consumption
- Solution?

Backpropagation in Neural ODE^[6]

- Backpropagation through an ODE solver is memory-intensive
- Requires storing intermediate states: high memory consumption
- Solution?
- Use Adjoint Sensitivity Method

Adjoint Sensitivity Method^[9]

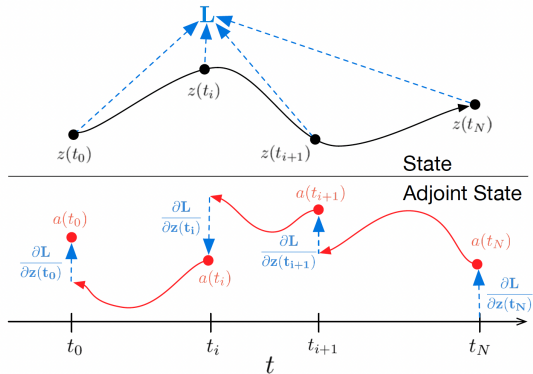
- A technique used to efficiently calculate the sensitivity of a system's output to changes in its input parameters
- In stead of storing all intermediate states, define an adjoint variable:

$$a(t) = \frac{\delta L}{\delta h(t)}$$

- Tracks how the loss depends on the hidden state $h(t)$ at time t (sensitivity)
- Compute how the adjoint state evolves backward in time $\frac{da}{dt} = -a(t)^T \frac{\delta f(h(t), t, \theta)}{\delta h}$
- Solve this ODE backward in time to compute gradients
- Ultimately the parameter gradients: $\frac{\delta \mathcal{L}}{\delta \theta} = \int_{t_0}^{t_1} a(t)^T \frac{\delta f(h(t), t, \theta)}{\delta \theta}$

Adjoint Sensitivity Method in Neural ODE^[6]

- Solve the ODE from t_0 to t_1 to get the final state $h(t_1)$
- Define the adjoint state $a(t)$
- Solve the adjoint ODE backward in time from t_1 to t_0
- Compute gradients using the integral $\frac{\partial \mathcal{L}}{\partial \theta} = \int_{t_1}^{t_0} a(t)^T \frac{\partial f(h(t), t, \theta)}{\partial \theta}$



Performance of Neural ODE^[6]

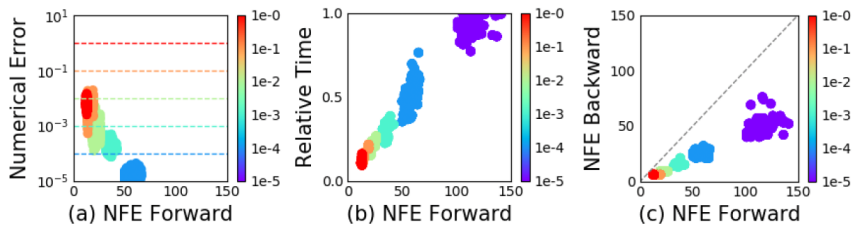


Figure: Statistics of Neural ODE (NFE: number of function evaluations)

Recurrent Neural Network (RNN) and Variational Autoencoder (VAE)

- Recurrent Neural Network (RNN): Previous outputs influence future predictions

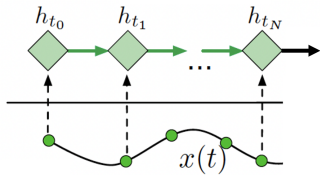


Figure: Recurrent Neural Network^[6]

- Variational Autoencoder: Learns a probabilistic latent representation of data

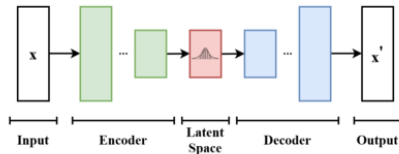
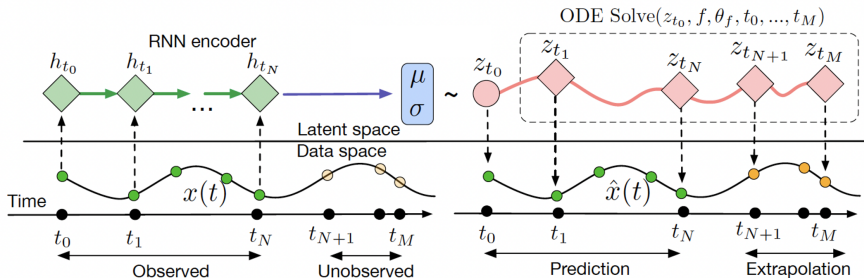


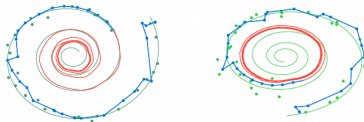
Figure: Variational Autoencoder^[10]

Latent Neural ODE^[6]

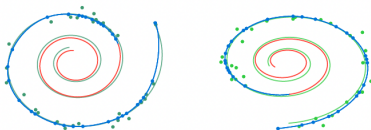
- Observations $x_1, x_2, \dots, x_N$ are encoded into a latent initial state z_{t_0}
- Use an RNN or MLP to approximate z_{t_0}
- Latent ODE solver evolves the latent state over time:
 $z_{t_1}, z_{t_2}, \dots, z_{t_M} = \text{ODESolve}(z_{t_0}, f, \theta_f, t_0, \dots, t_M)$
- Maps latent states back to observations: $\hat{x}_t \sim p(\hat{x}_t | z_t)$



Performance of Latent ODE^[6]

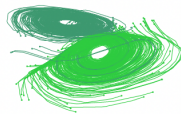


(a) Recurrent Neural Network



(b) Latent Neural Ordinary Differential Equation

— Ground Truth
• Observation
— Prediction
— Extrapolation



(c) Latent Trajectories

Strengths and Limitations of Neural ODE

- **Strengths:**

- Models data evolution continuously, ideal for time-series, and dynamical systems
- Uses the adjoint method for backpropagation, solving an ODE backward instead of storing all intermediate activations. This reduces memory cost
- Can work well with irregular time-series data, particularly useful in biomedical applications

- **Limitations:**

- Requires iterative numerical integration: slower
- Different ODE solvers (Euler, Runge-Kutta) produce varying results
- Without explicit physics constraints (e.g., PINNs) or conservation laws (e.g., HNNs), Neural ODEs may not generalize well to physical systems

Comparative Analysis of PiML Frameworks

Features	HNN	PINN	Neural ODE
Physics Integration	Embedded in the loss function	Embedded in the loss function	Continuous-time evolution of hidden states
Use Case	Energy-conserving systems	PDE (or ODE?) constrained problems	Continuous-time modeling, time-series
Computational Cost	Moderate (efficient, but needs structured problems)	High (many collocation points needed)	Expensive (solving ODEs iteratively)
Advantage	Conserves energy	Ensures physics constraints	Leverages ODE solvers in NN framework

Future Work

- Combining Hamiltonian and Lagrangian mechanics into a unified framework
 - Both energy conservation and motion constraints
- Combine physics constraints with continuous-time representations
 - Combining Neural ODE with PINN or Neural ODE with HNN would be a very strong modeling tool
- In cell biology:
 - PINNs can be used to model reaction-diffusion systems underlying gene expressions
 - HNNs can simulate intracellular dynamics where energy conservation plays a role
 - Neural ODEs with their adaptability to time-series data can track changes in gene expression or cell states

References I

- [1] Steve Brunton. Physics informed machine learning: High level overview of ai and ml in science and engineering, 2024. URL <https://www.youtube.com/watch?v=JoFW2uSd3Uo&list=PLMrJAKhIeNNQ0BaKuBKY43k4xMo6NSbBa&index=1&t=1s>.
- [2] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.
- [3] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.
- [4] Suzanne Clancy and William Brown. Translation: Dna to mrna to protein. *Nature Education*, 1(1): 101, 2008.
- [5] Ting Chen, Hongyu L He, and George M Church. Modeling gene expression with differential equations. In *Biocomputing'99*, pages 29–40. World Scientific, 1999.
- [6] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.
- [7] Engineering Made Easy. Euler discretization, 2019. URL <https://www.youtube.com/watch?v=rjeSqqHowTg>.

References II

- [8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [9] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. The mathematical theory of optimal processes. Interscience Publishers, 1962.
- [10] Variational autoencoder. Wikipedia. URL https://en.wikipedia.org/wiki/Variational_autoencoder.

Thank You