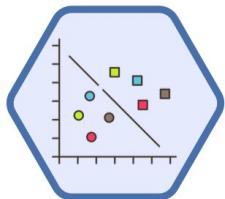


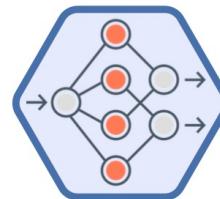
# Physics-Informed Hamiltonian Neural Networks

## Marios Mattheakis

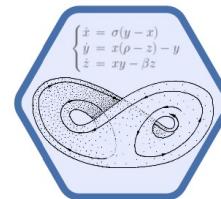
Workshop on Machine Learning in astronomy:  
From classical to physics-informed



Classical ML



Deep Learning Physics-informed ML

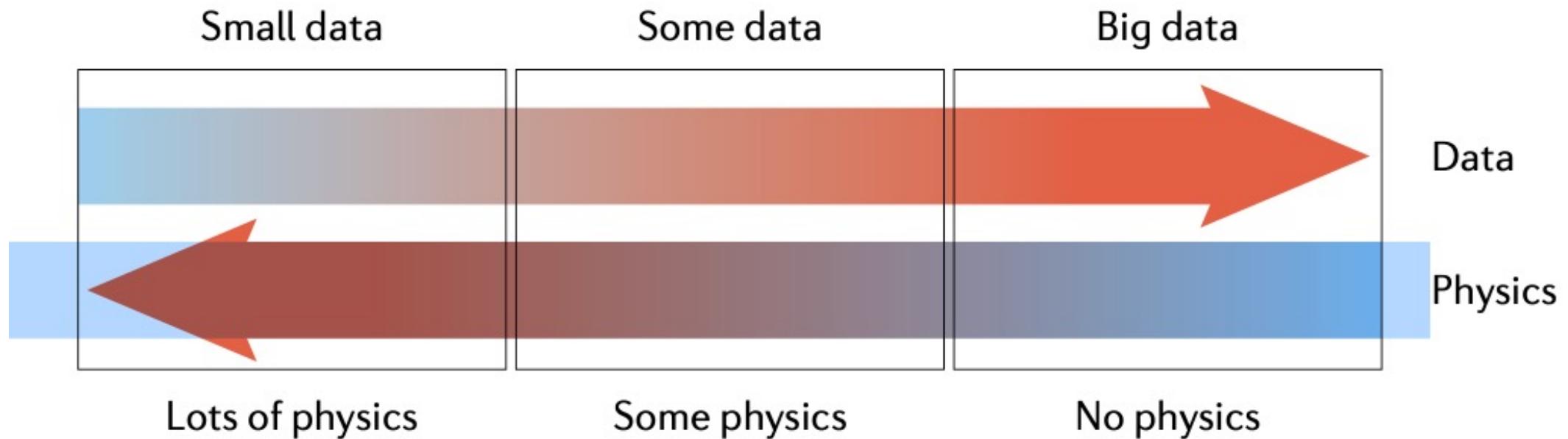


**HARVARD**  
School of Engineering  
and Applied Sciences



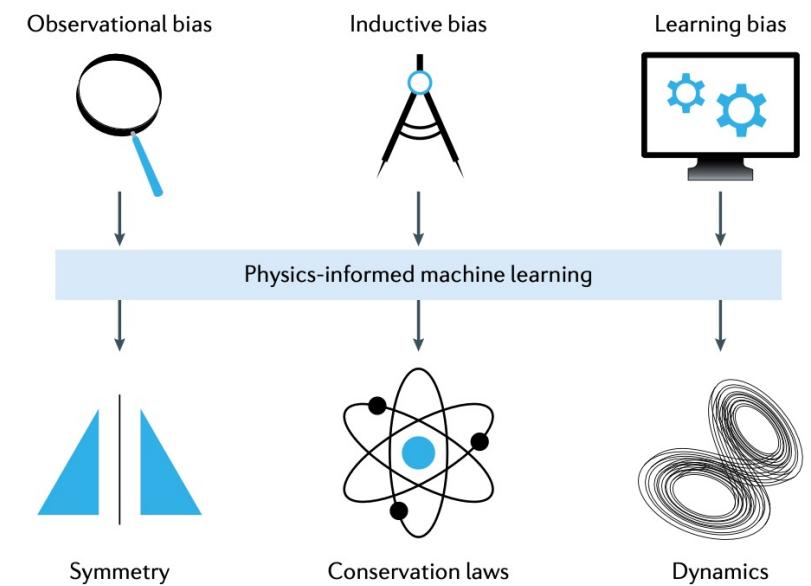
**IACS**  
INSTITUTE FOR APPLIED  
COMPUTATIONAL SCIENCE  
AT HARVARD UNIVERSITY

# From classical to physics-informed ML



# Outline

- Express physics in NN language
- Building physics-informed Neural Networks
- Solving Hamiltonian systems, data-free NN



# Expressing physics

Physical laws are formulated as differential equations (DEs)

An ML model can be informed through the loss function as

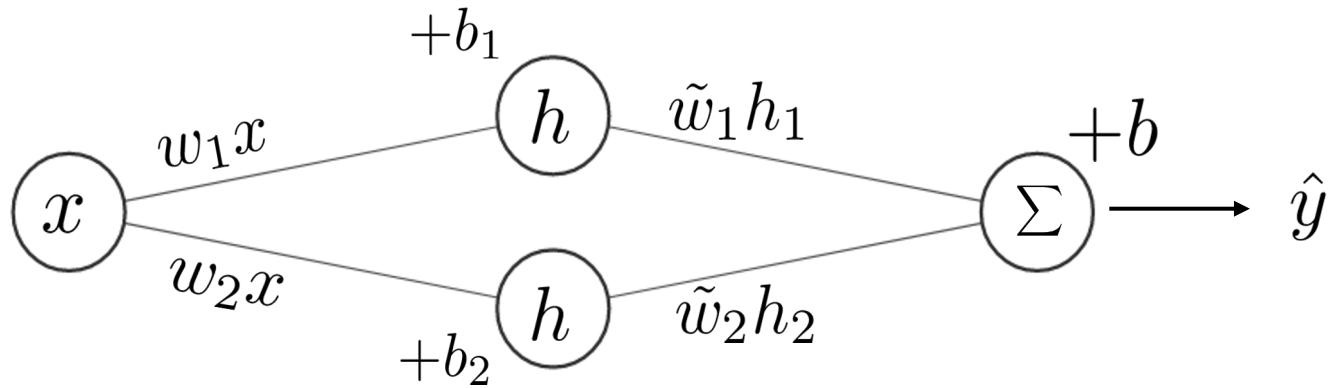
$$\mathcal{L} = \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{DE}}$$

$\mathcal{L}_{\text{data}}$  : Incorporating data

$\mathcal{L}_{\text{DE}}$  : Embedding physics

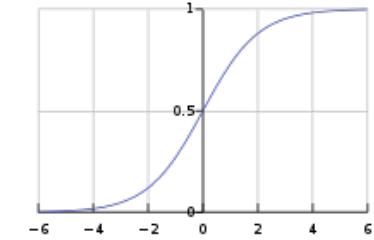
How can we define derivatives and DEs in Neural Networks?

# Review: Fully Connected Feed Forward NN



$$h_j = f(w_jx + b_j)$$

$$\hat{y}(x) = \sum_j \tilde{w}_j h_j + b$$



## Loss Functions Examples

Data Driven model

$$\mathcal{L}_{\text{data}} = \sum_i (\hat{y}_i - y_i)^2$$

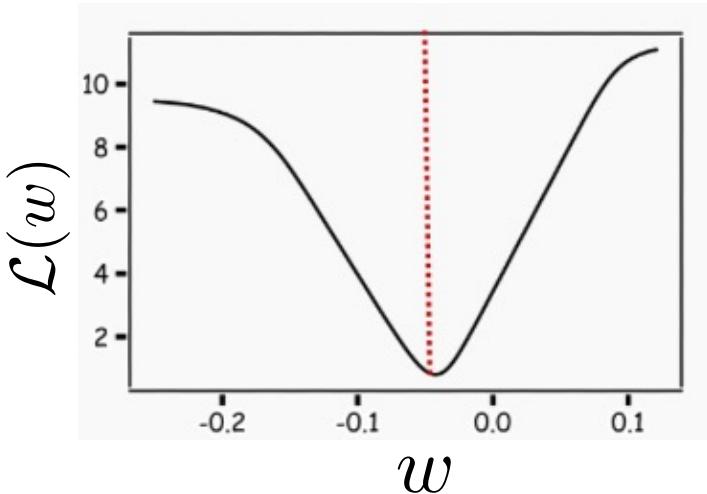
Equation Driven model

$$\mathcal{L}_{\text{DE}} = \sum_i \left( \frac{d\hat{y}_i}{dx} - g(\hat{y}_i) \right)^2$$

Physics-informed

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{DE}}$$

# Network optimization: Gradient descent example

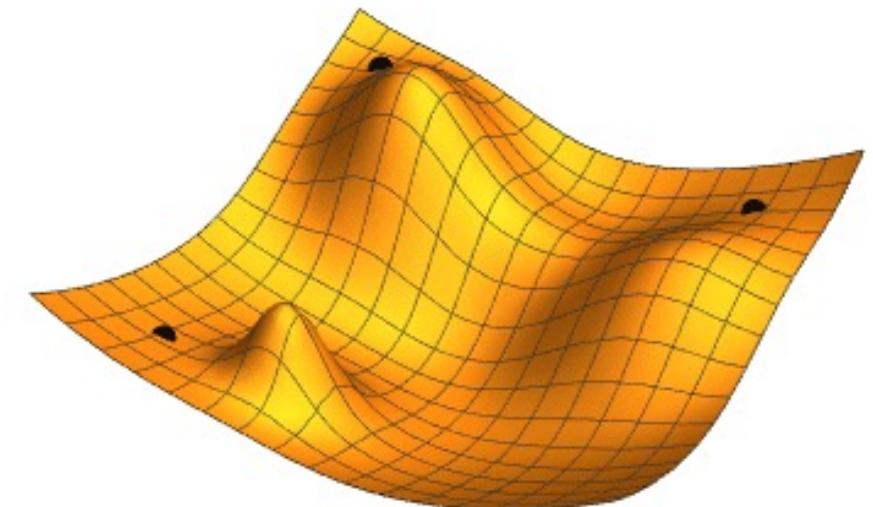
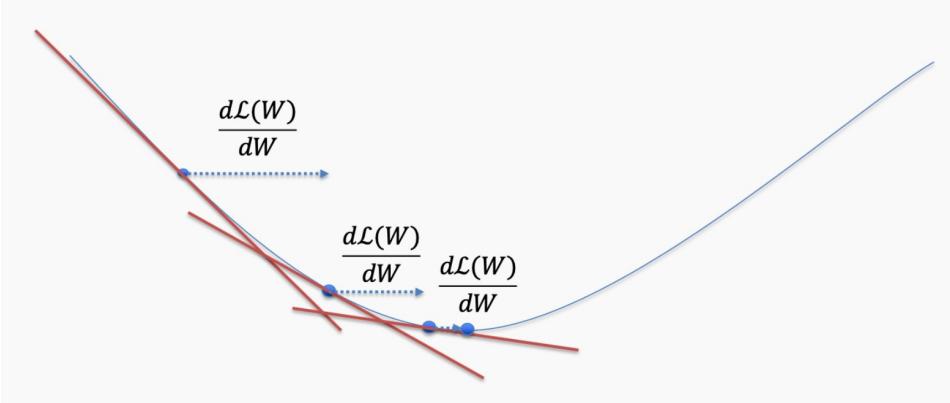


Minimizing the loss function

$$w^* = \operatorname{argmin}_w \mathcal{L}(w)$$

$$\frac{\partial \mathcal{L}}{\partial w} = 0$$

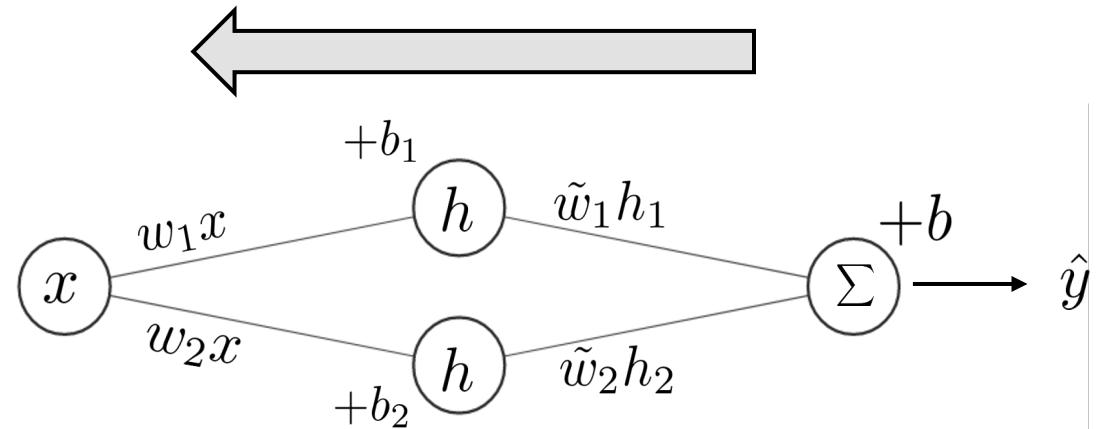
Gradient descent:  $w^{\text{new}} = w^{\text{old}} - \lambda \frac{\partial \mathcal{L}}{\partial w^{\text{old}}}$



# Auto-differentiation and back-propagation

Backpropagation through chain rule

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_1} \frac{\partial h_1}{\partial w_1}$$



Compute **exact** derivatives with respect to the inputs

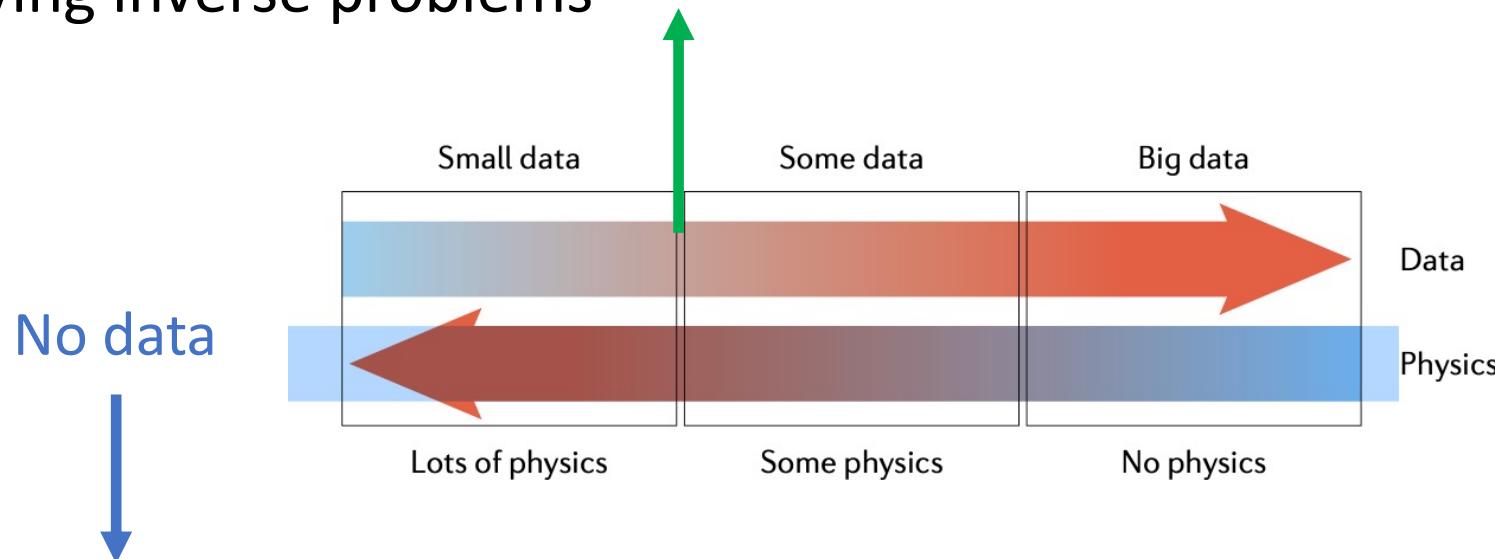
$$\frac{\partial \hat{y}}{\partial x} = \frac{\partial \hat{y}}{\partial h_1} \frac{\partial h_1}{\partial x}$$

This is the **key** in constructing loss functions that contain differential equations

Similarly for higher order derivatives and for deeper networks

# Physics-Informed NNs applications

- Learn solutions that fit data and respect physics (satisfying DEs)
- Discover hidden physics by learning DEs and physical quantities from data
- Robust training of deep networks with sparse and noisy data
- Solving inverse problems



- Solving DEs; unsupervised data-free learning

# Hamiltonian Neural Networks

Physics-informed NNs for solving the equations of motion for energy conserving dynamical systems

No data limit:  
A data-free unsupervised learning methods

# Hamiltonian formulation: Review

Hamiltonian

$$\mathcal{H}(q_i, p_i) = K(p_i) + V(q_i)$$

Hamilton's Equations

$$\dot{q}_i = \frac{\partial \mathcal{H}}{\partial p_i}, \quad \dot{p}_i = -\frac{\partial \mathcal{H}}{\partial q_i}$$

Symplectic notation yields energy conservation (Liouville's theorem)

$$\dot{\mathbf{z}} = \mathbf{J} \cdot \frac{\partial \mathcal{H}}{\partial \mathbf{z}}$$

$$\mathbf{J} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

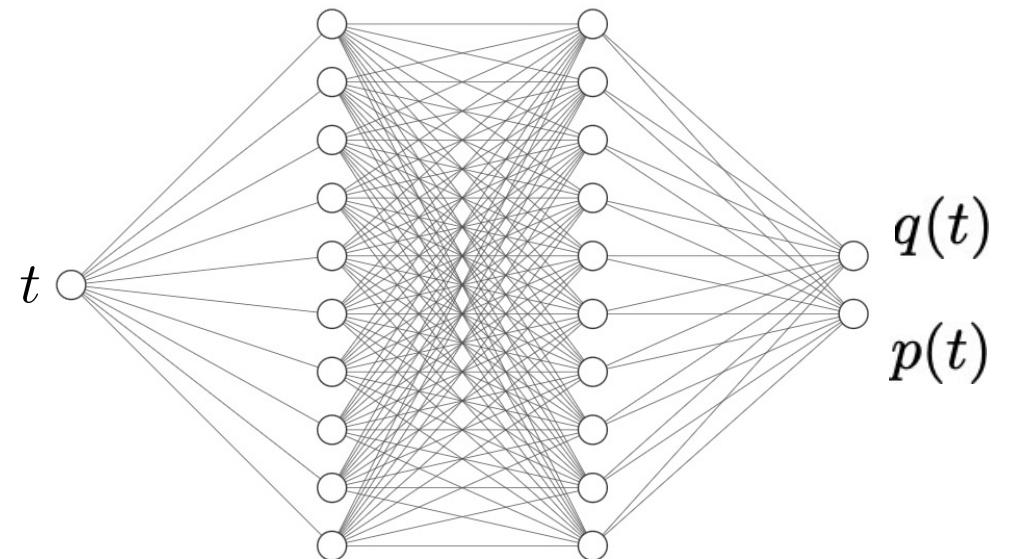
$$\mathbf{z} = (q_1, \dots, q_\nu, p_1, \dots, p_\nu)^T$$

# Problem statement

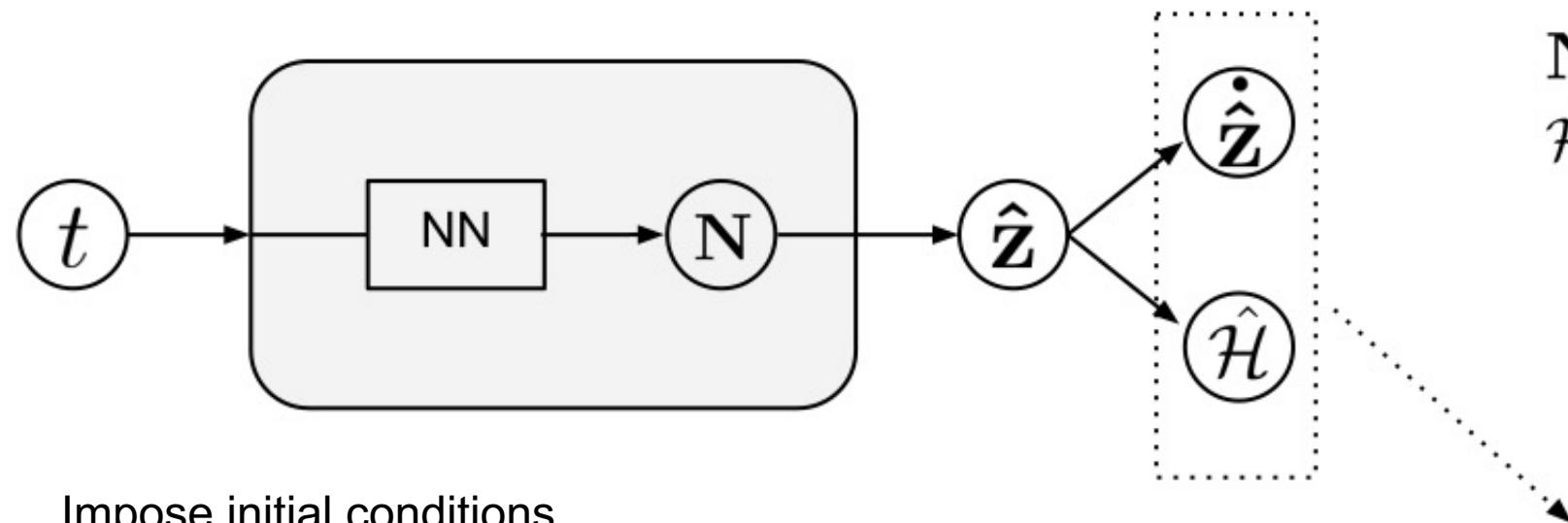
For a given Hamiltonian, calculate the phase space trajectories for a certain initial state

## Pipeline

1. Define a training interval and sample time points. Random sampling in every forward pass improves the training
2. Define the Hamiltonian function and ICs
3. The loss function is defined by Hamilton's equations (physics informed part)
4. Impose initial conditions by constructing a parametric solution for  $q, p$ .



# Hamiltonian Neural Networks



$$\mathbf{N} \in \mathbb{R}^D$$
$$\hat{\mathcal{H}} \equiv \mathcal{H}(\hat{\mathbf{z}})$$

Impose initial conditions

$$\hat{\mathbf{z}}(t) = \mathbf{z}(0) + f(t)\mathbf{N}(t)$$

$$f(t) = 1 - e^{-t}$$

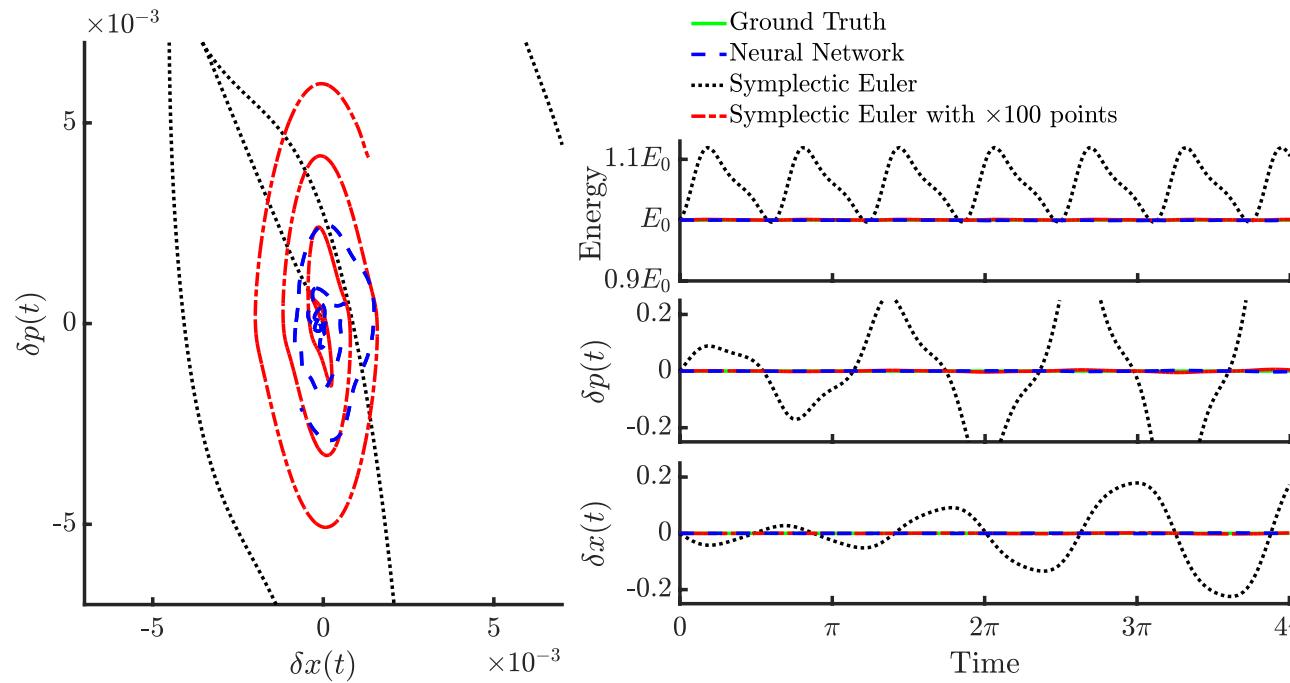
$$L = \frac{1}{K} \sum_{n=1}^K \left( \dot{\hat{\mathbf{z}}}^{(n)} - \mathbf{J} \cdot \nabla_{\hat{\mathbf{z}}^{(n)}} \mathcal{H}(\hat{\mathbf{z}}^{(n)}) \right)^2$$

Derivatives are calculated with auto-grad.

# Nonlinear Oscillator

$$\mathcal{H} = \frac{p^2}{2} + \frac{x^2}{2} + \frac{x^4}{4}$$

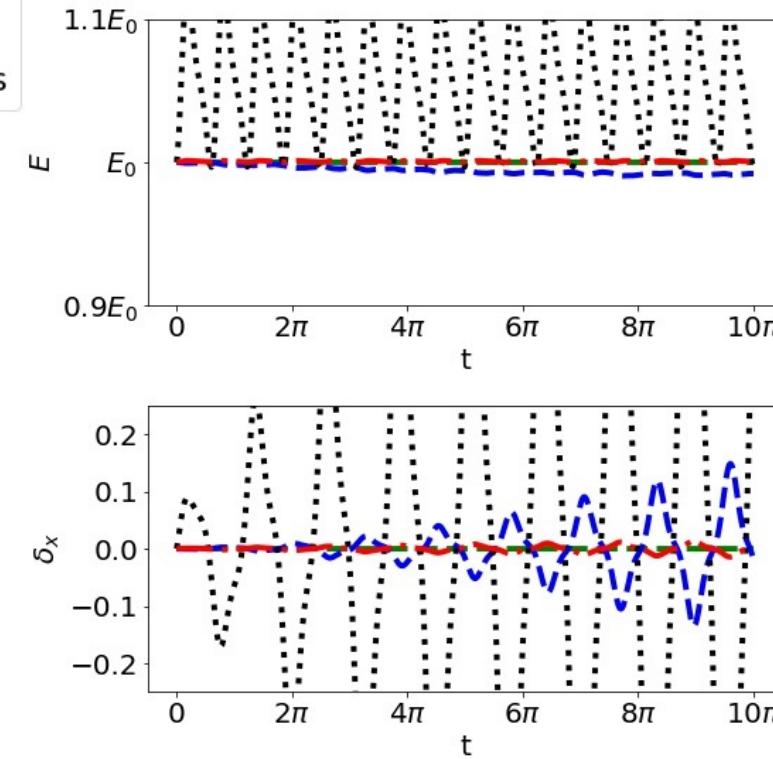
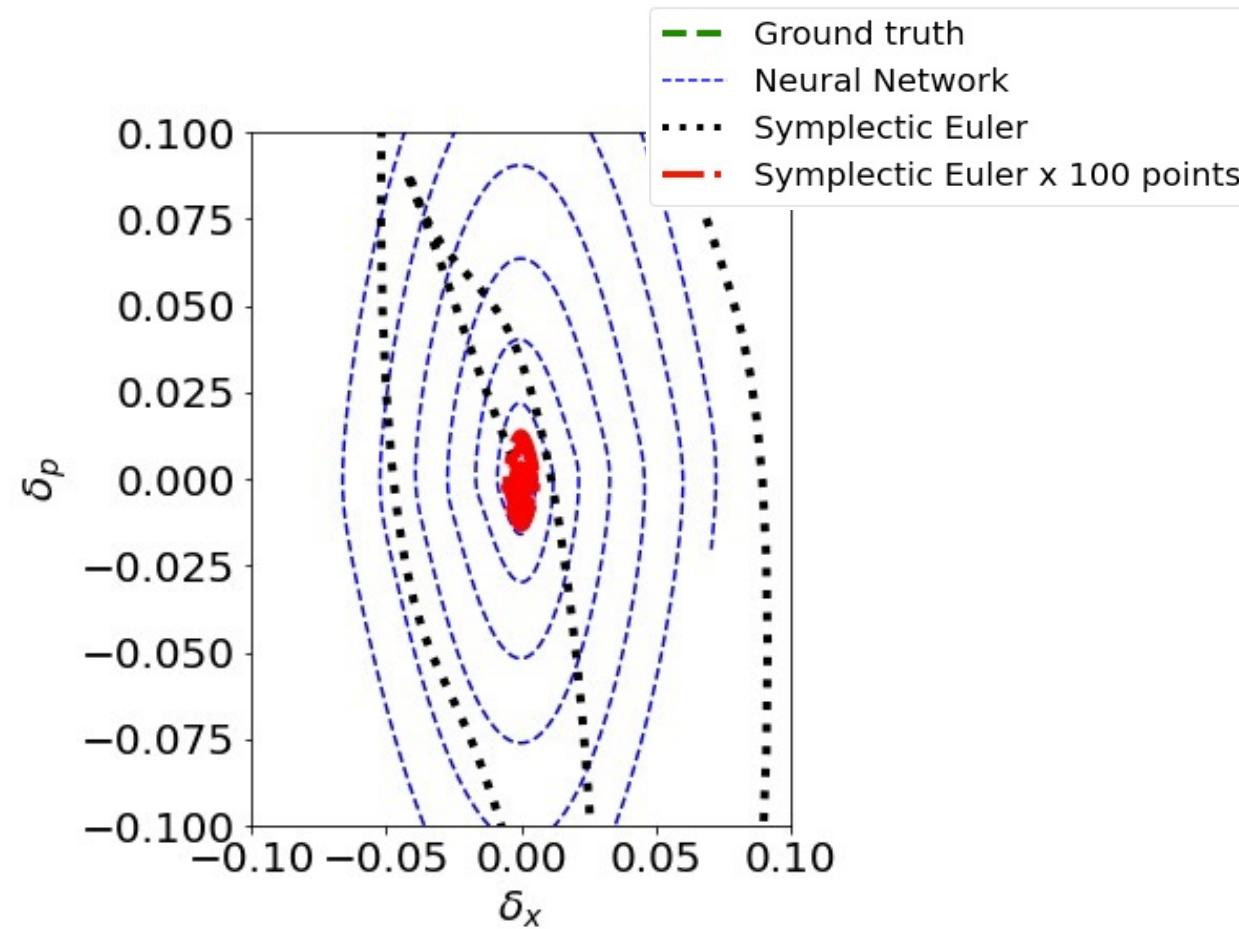
$$\mathcal{L} = \frac{1}{K} \sum_{n=1}^K \left[ \left( \dot{\hat{x}}^{(n)} - \hat{p}^{(n)} \right)^2 + \left( \dot{\hat{p}}^{(n)} + \hat{x}^{(n)} + \left( \hat{x}^{(n)} \right)^3 \right)^2 \right]$$



Due to the efficient sampling, NN needs less points than symplectic Euler method

# Long times: Leaking of Energy

The NN tends to forget the initial state jumping to solutions with different energy



# Energy conservation through Regularization

Any extra information assists the NN to reach the correct solutions

Physics-informed Loss function

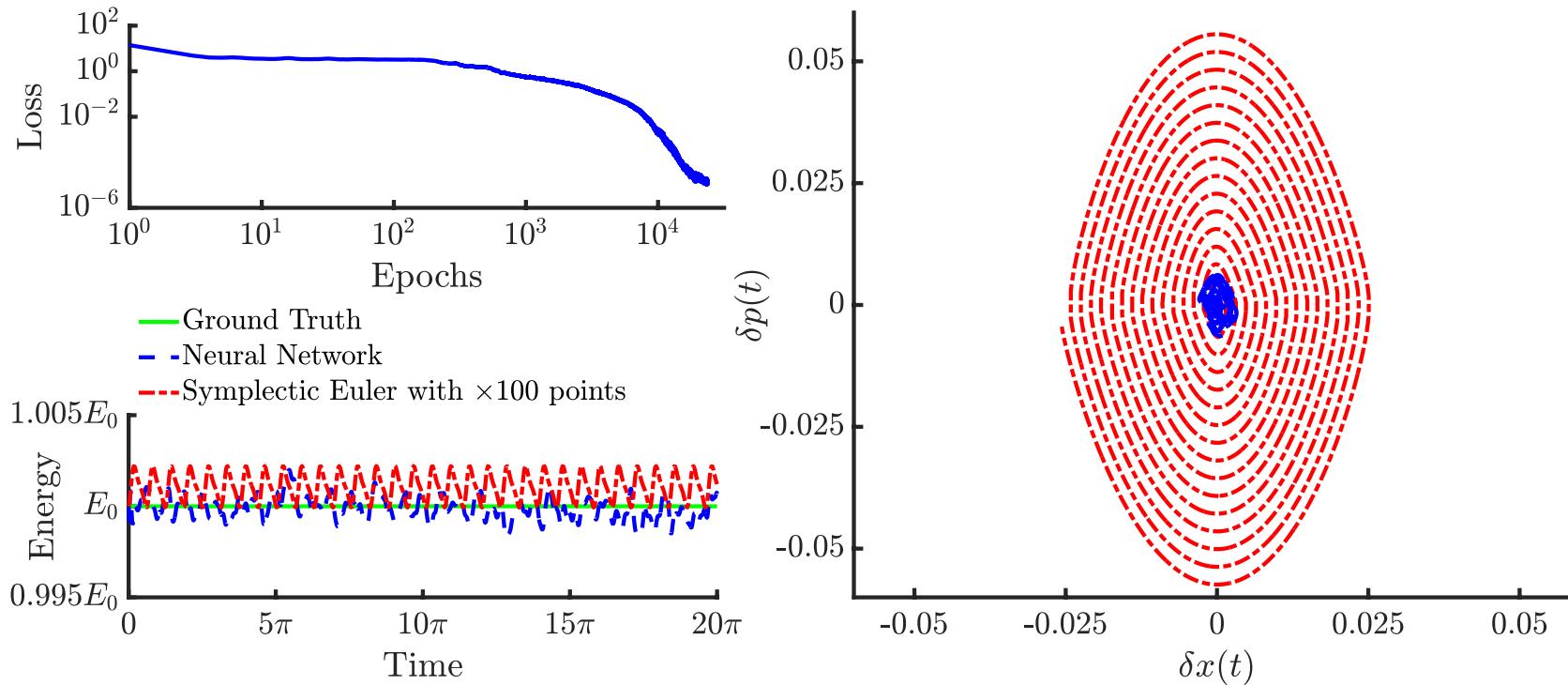
$$\mathcal{L} = \frac{1}{K} \sum_{n=1}^K \left( \dot{\hat{\mathbf{z}}}^{(n)} - \mathbf{J} \cdot \nabla_{\hat{\mathbf{z}}^{(n)}} \mathcal{H} \left( \hat{\mathbf{z}}^{(n)} \right) \right)^2 + \lambda L_{\text{reg}}$$


Penalty (regularization) term

Energy penalty stabilizes the trajectory for long-time solutions

$$L_{\text{reg}} = \frac{1}{K} \sum_{n=1}^K \left[ \left( \mathcal{H} \left( \hat{\mathbf{z}}^{(n)} \right) - E_0 \right)^2 \right]$$

# Not accumulating numerical error



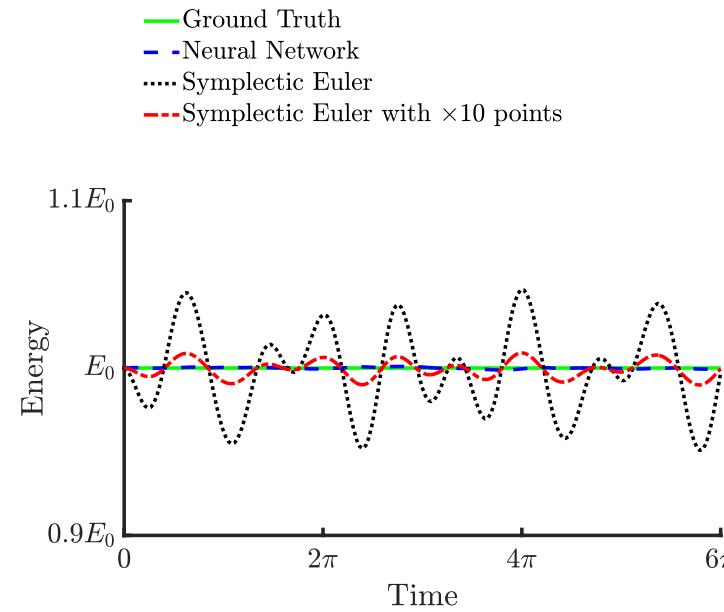
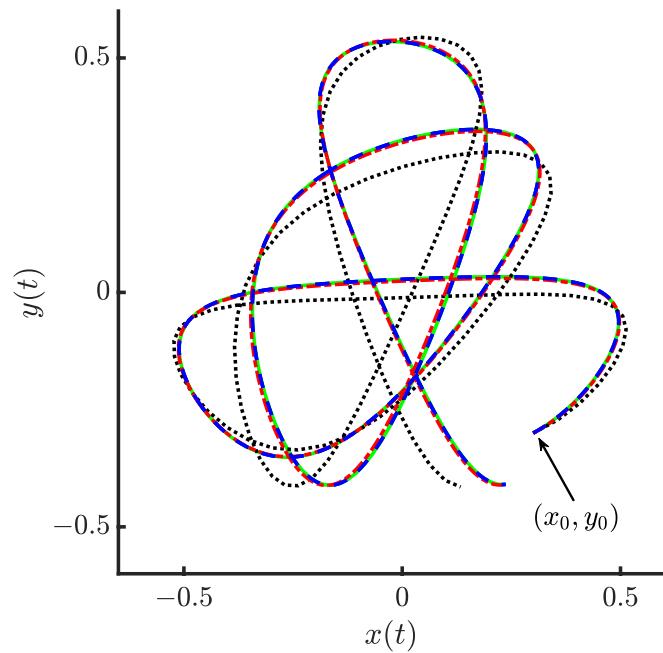
Satisfy the equations at each point independently (avoiding accumulation of numerical error)

Satisfy all the equations of system simultaneously (energy conservation)

The maximum numerical error can be pre-defined by the minimum value of the loss function

# Henon-Heiles: A chaotic system (irregular motion)

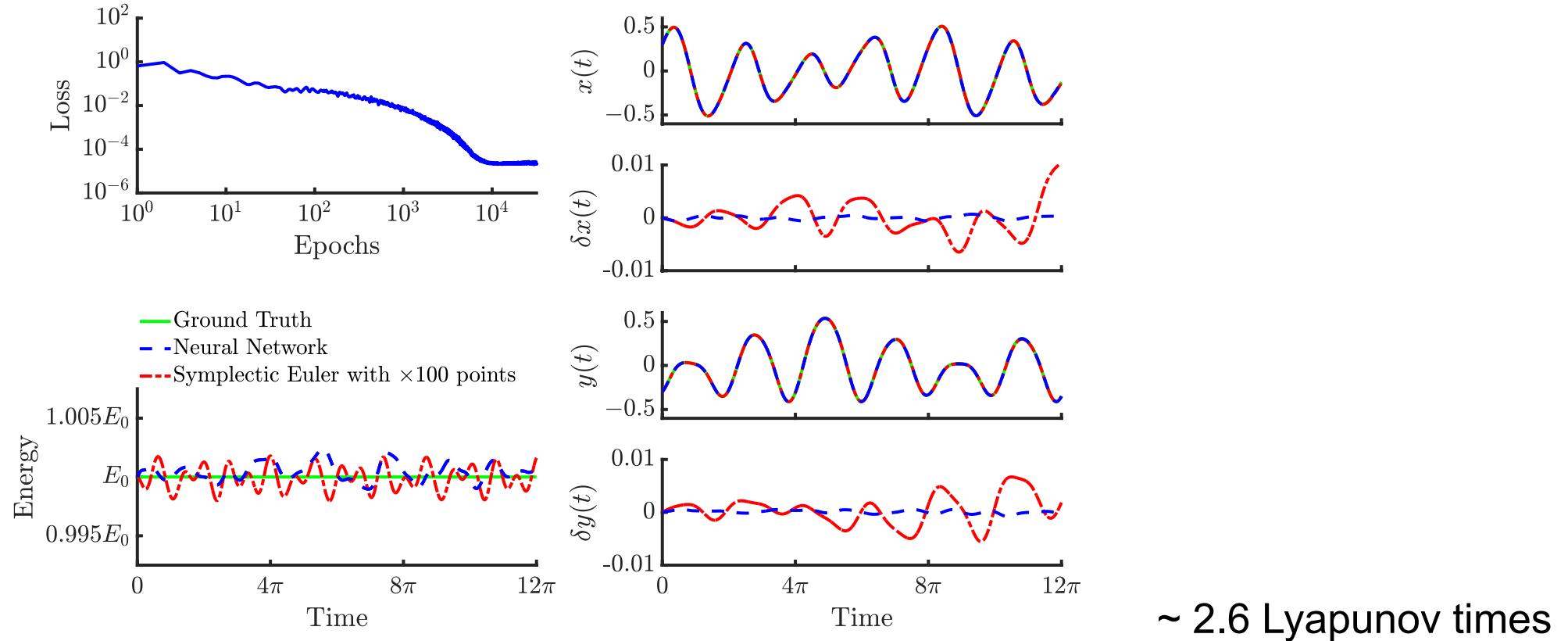
$$\mathcal{H} = \frac{p_x^2 + p_y^2}{2} + \frac{x^2 + y^2}{2} + x^2y - \frac{y^3}{3}$$



$\sim 1.3$  Lyapunov times

Lyapunov time is the characteristic timescale on which a dynamical system is chaotic

# Henon Heiles: Longer time



Symplectic Euler shows similar order of numerical error in the energy with x100 more points , but accumulates more error in the solutions

# Discussion

- Computational Cost: It is expensive to train a network but:
  - GPUs are getting better and better and can accelerate the calculations
- Fail to extrapolate outside the training range:
  - RNNs and attention models hold a promise
- A closed-form, analytical, and differentiable numerical solution
  - The solution is available everywhere and can be inverted
  - Efficient storage of solutions: Store just a budge of numbers (weights & biases)
- Parametrization and regularization
  - Initial conditions are identically satisfied
  - Physical laws can be embedded through regularization
- Available data can be easily incorporated to improve the network performance
- Same method can be used to solve PDEs and eigenvalue problems

# Conclusion

- Data-free network for solving differential equations
- Analytical and differentiable numerical solutions
- Structured mesh is not needed, random sampling in every iteration
- Satisfy identically the initial conditions
- Encoded physical principles



**HARVARD**

School of Engineering  
and Applied Sciences



**INSTITUTE FOR APPLIED  
COMPUTATIONAL SCIENCE**  
AT HARVARD UNIVERSITY

*Thank you*

mariosmat@g.harvard.edu

[https://scholar.harvard.edu/marios\\_matthaiakis](https://scholar.harvard.edu/marios_matthaiakis)

# References

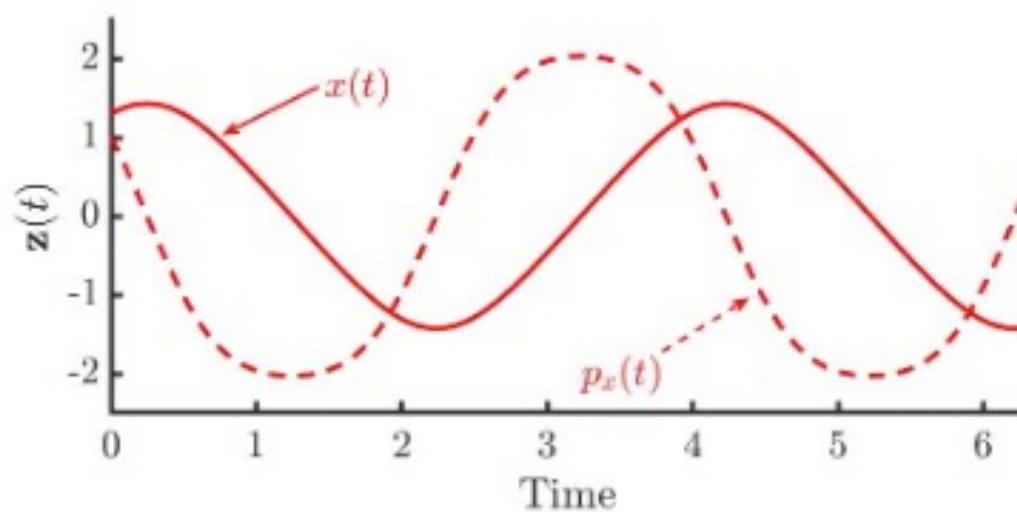
- Physics-informed machine learning, G.E. Karniadakis et al. *Nature Reviews Physics* 2021
- Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, L. Lu et al. *Nature machine intelligence*, 2021
- Semi-supervised Neural Networks solve an inverse problem for modeling Covid-19 spread, A. Paticchio et al. *NeurIPS workshop* 2020
- Unsupervised Neural Networks for Quantum Eigenvalue Problems, H. Jin et al. *NeurIPS workshop* 2020
- NeuroDiffEq: A Python package for solving differential equations with neural networks, F. Chen, *JOSS* 2020
- Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, M. Raissi et al. *J. Comp. Physics* 2019
- Hamiltonian Neural Networks, S. Greydanus et al, *NeurIPS* 2019
- Hamiltonian Neural Networks for Solving Differential Equations, M. Mattheakis et. *arXiv* 2020
- Port-Hamiltonian Neural Networks for Learning Explicit Time-Dependent Dynamical Systems, S. Desai et al. *arXiv* 2021
- Variational Integrator Graph Networks for Learning Energy Conserving Dynamical Systems, S. Desai et al. *arXiv* 2021

# Supplementary Material

# Nonlinear Oscillator

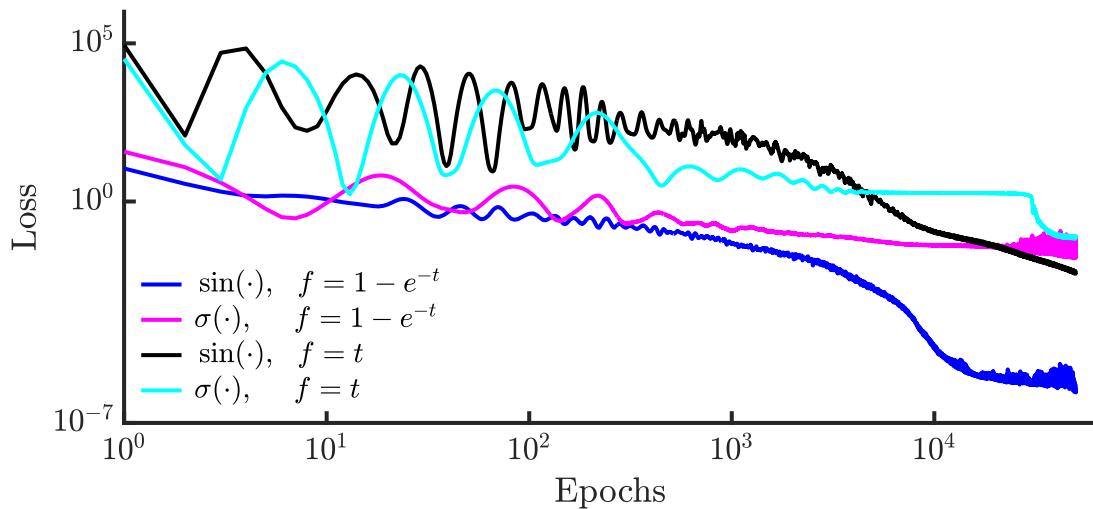
$$\mathcal{H}(q, p) = \frac{p^2}{2} + \frac{x^2}{2} + \frac{x^4}{4}$$

$$\begin{aligned}\dot{x} &= p \\ \dot{p} &= -(x + x^3)\end{aligned}$$

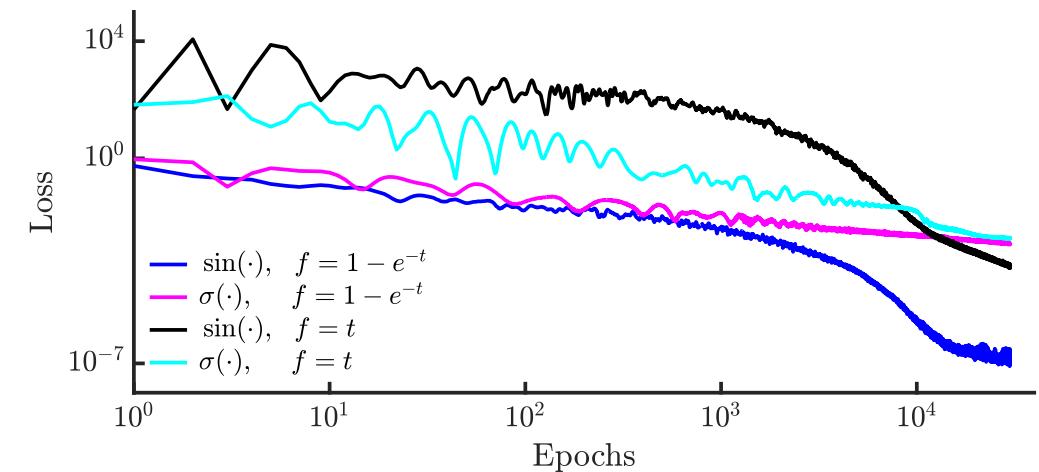


# Network convergence for different activation and parametric functions

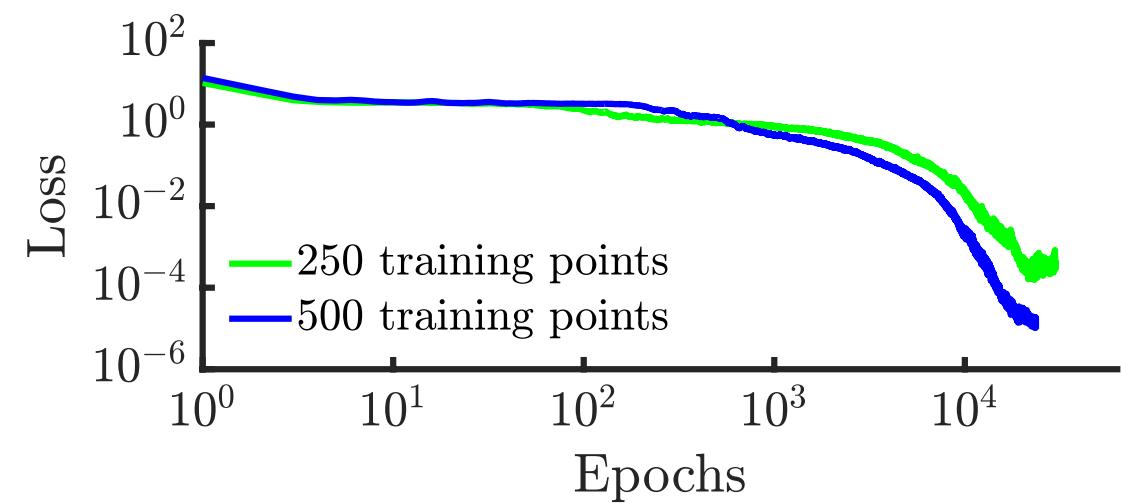
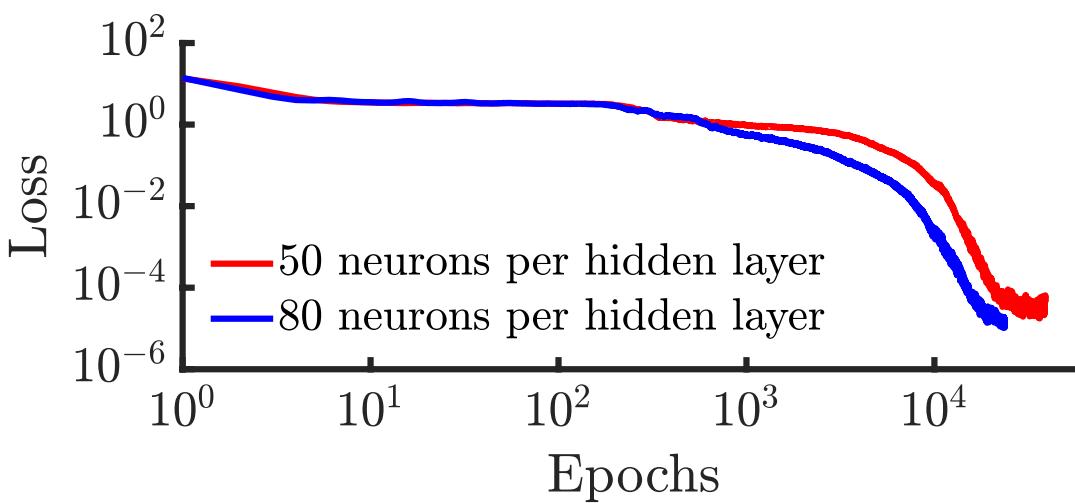
Nonlinear oscillator



Henon Heiles



# Network convergence for different number of points and number of neurons



# Solving Differential Equations

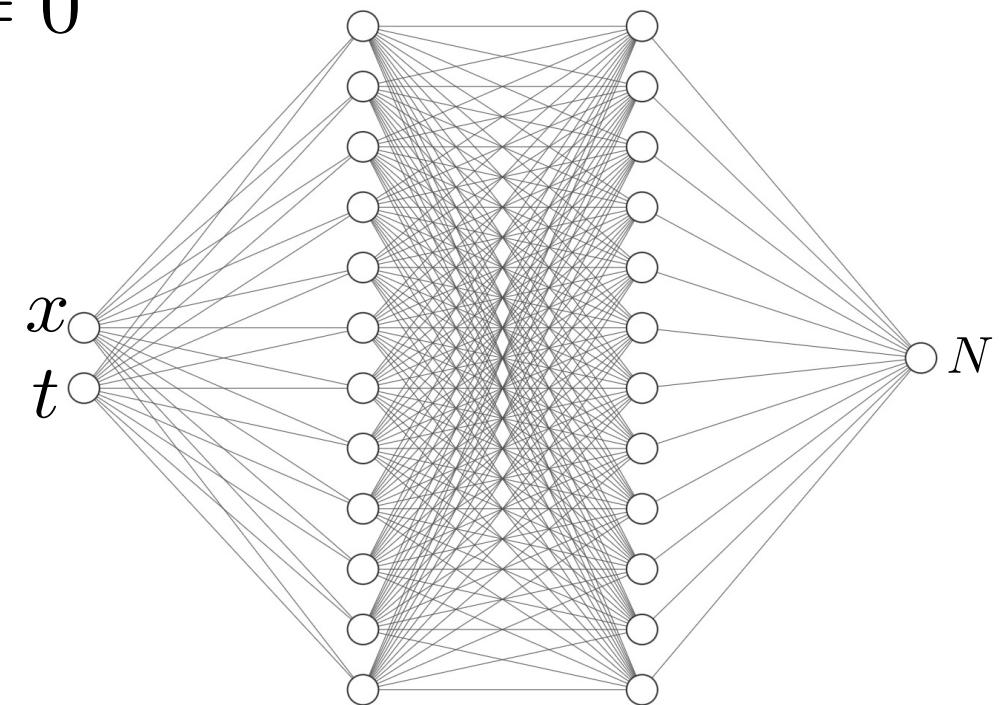
E. Lagaris et. al. IEEE Neural Nets 1998

Consider the general (1+1) dimensional PDE form

$$\mathcal{G}y(x, t) - f(x, t) = 0$$

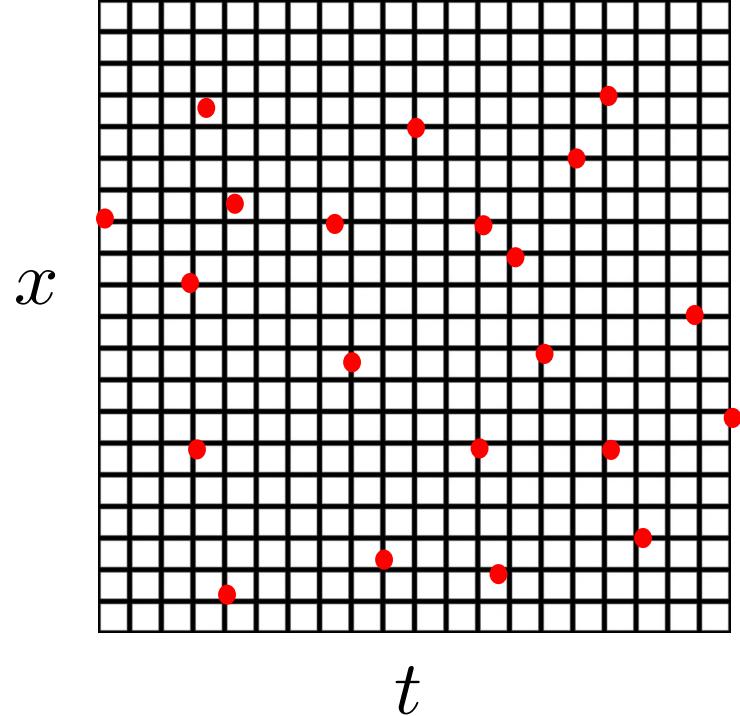
Train a network  $N(x, t; w)$  by minimizing  
the loss function

$$\operatorname{argmin}_w (\mathcal{G}N(x, t; w) - f(x, t))^2$$

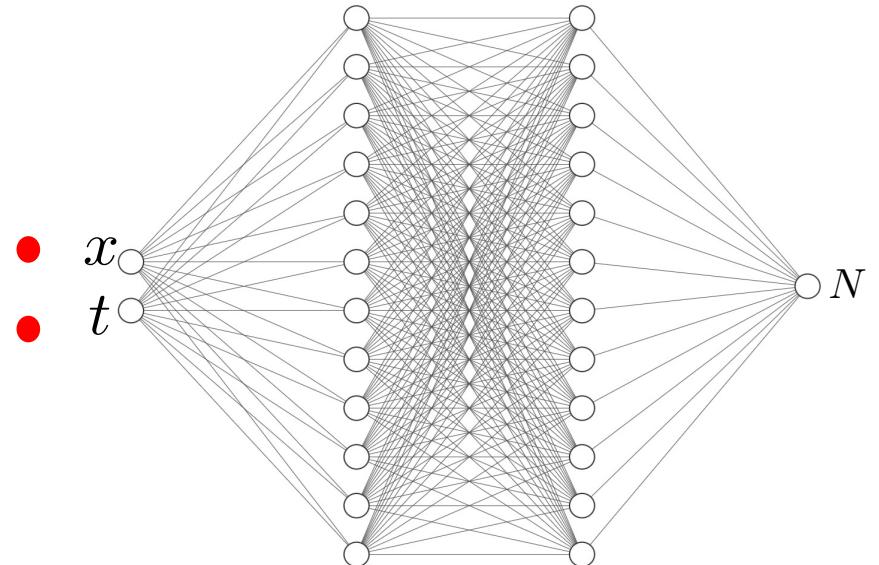


Unsupervised and data-free learning method

# Structured mesh is not required



$$\mathcal{G}y(x, t) - f(x, t) = 0$$



Random sampling in every iteration: Less points than usual  
Promising for a weaker “dimensionality curse”  
No distinguish between dimensions

# Imposing Conditions through regularization

$$\operatorname{argmin}_w \left[ (\mathcal{G}N(x, t) - f)^2 + \underbrace{(N(x, t_0) - y_0(x))^2 + (N(x_b, t) - y_b(t))^2}_{\text{Initial Condition}} + \underbrace{\dots}_{\text{Boundary Condition}} \right]$$

The regularization approximates the conditions

Any other known information can be encoded in the loss function (e.g. conservation laws)

# Imposing Conditions through parametrization

Parametric solution

$$\hat{y}(x, t) = A(x, t) + F(x, t, N)$$

A(.) contains the conditions

F(.) contains the NN

Parametric function

$$F(x_b, t, N) = F(x, t_0, N) = 0$$

The loss function contains only the equation

$$\operatorname{argmin}_w \left[ (\mathcal{G}\hat{y} - f)^2 \right]$$

The conditions are identically satisfied, but a parametric form is not always accessible

# Neural Network Solvers

- A general method for solving ODEs & PDEs
- Obtain analytical and differentiable numerical solutions (evaluated everywhere)
- Structured mesh is not needed, random sampling in every iteration (less points)
- Satisfy identically the conditions when a parametric function is possible
- The solver does not distinguish between the dimensions
- Efficient storage of the solutions
- We learn analytic general solutions

# Implementations of NN solvers

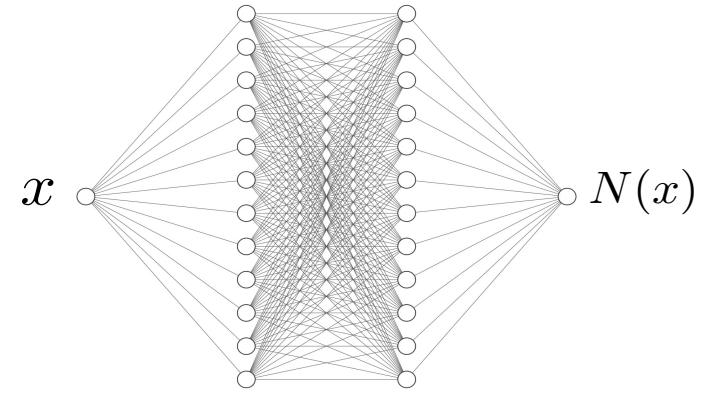
- Hamiltonian Neural Networks
- Eigenvalue problems
- Inverse problems

# Example: linear ODE

$$y'(x) + y(x) = 0, \quad y(0) = y_0$$

Regularization method

$$\mathcal{L} = \sum_{i=1}^M (N'(x_i) + N(x_i))^2 + (N(0) - y_0)^2$$



Parametrization method

$$\hat{y}(x) = y_0 + f(x)N(x)$$

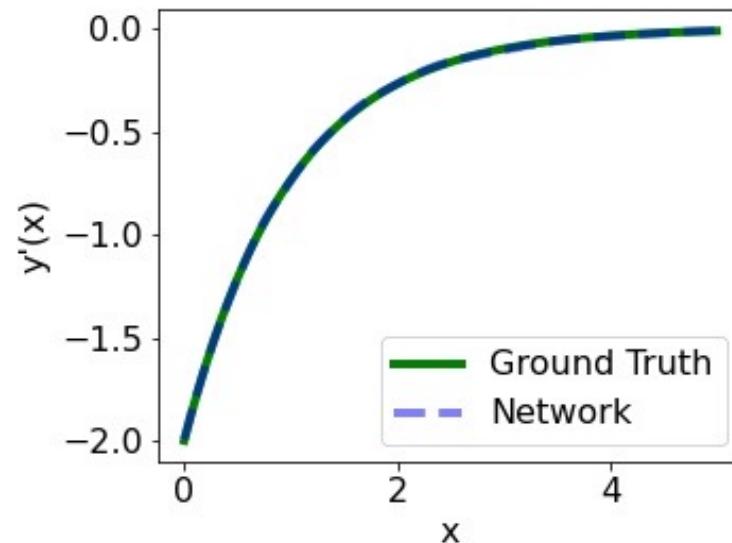
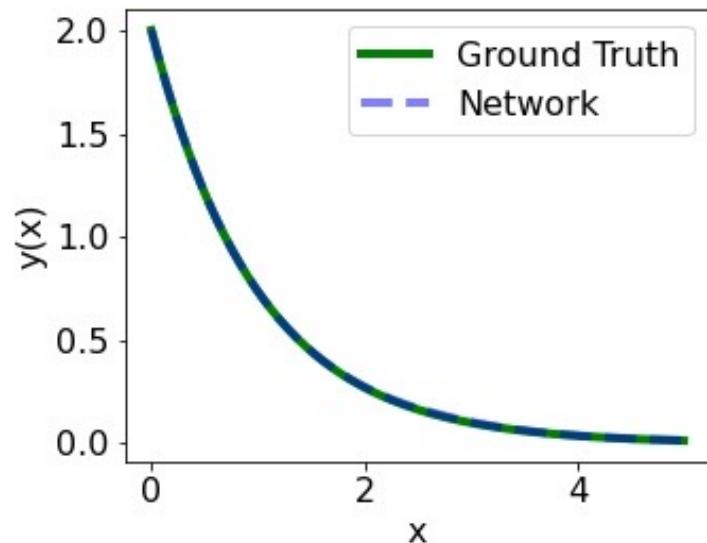
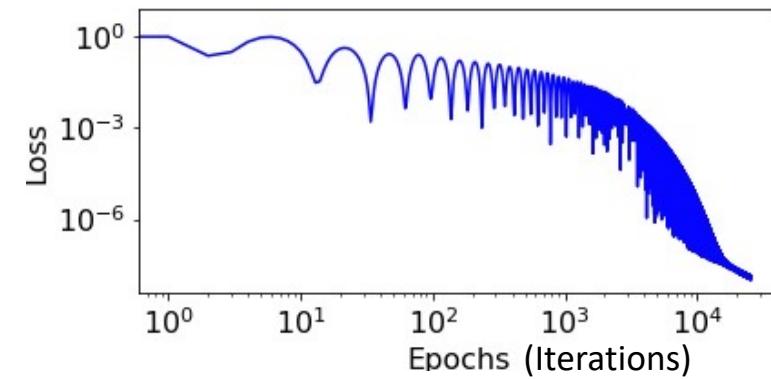
any function  $f(x)$  with the constraint  $f(0)=0$

$$\mathcal{L} = \sum_{i=1}^M (\hat{y}'(x_i) + \hat{y}(x_i))^2$$

$$f(x) = x, \quad f(x) = 1 - e^{-x}$$

# Example: linear ODE

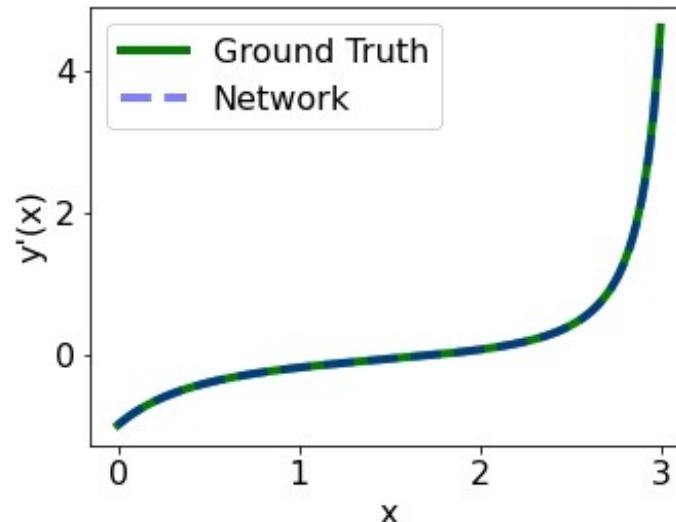
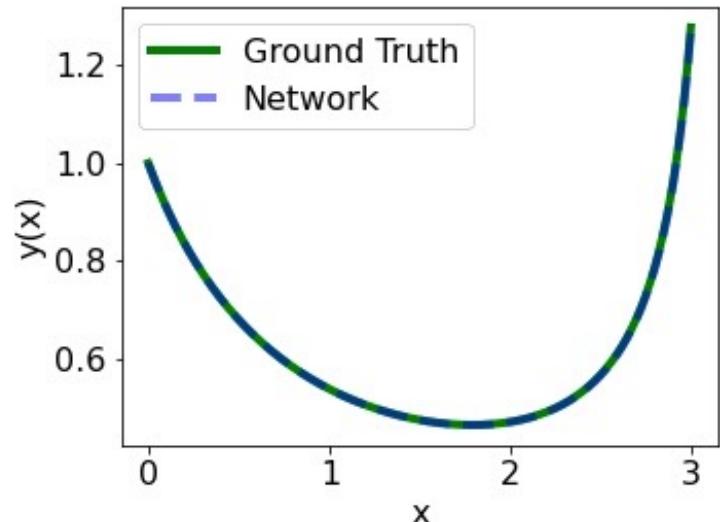
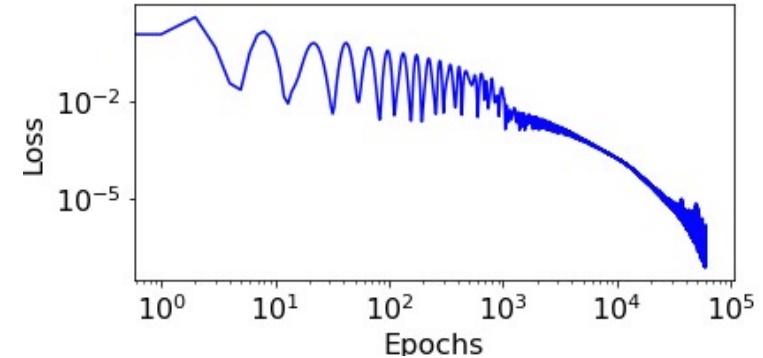
$$\mathcal{L} = \sum_{i=1}^M (\hat{y}'(x_i) + \hat{y}(x_i))^2$$



# Example: nonlinear ODE

$$y' + y = \frac{6}{5}xy^2, \quad y(0) = 1, \quad x \in [0, 3]$$

$$\hat{y}(x) = 1 + (1 - e^{-x})N(x)$$



# Example: Second Order ODE

$$y''(x) + y(x) = g(x), \quad y(0) = y_0, \quad y'(0) = v_0$$

Regularization method

$$\mathcal{L} = \sum_{i=1}^M \left( N''(x_i) + \hat{N}(x_i) - g(x_i) \right)^2 + \left( N(0) - y_0 \right)^2 + \left( N'(0) - v_0 \right)^2$$

Parametrization method

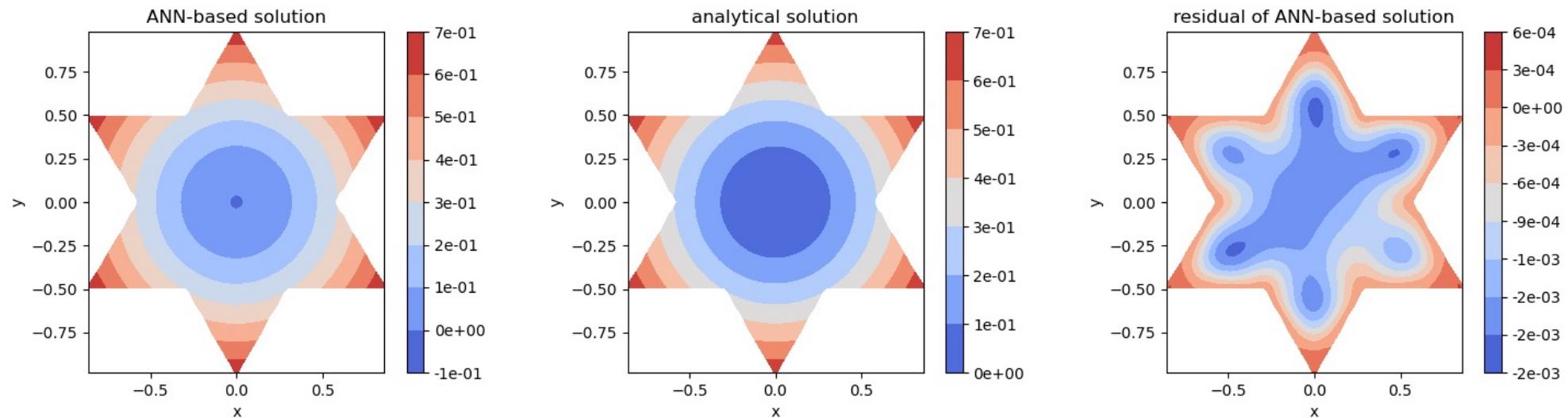
$$\hat{y}(x) = y_0 + xv_0 + x^2 N(x)$$

$$\mathcal{L} = \sum_{i=1}^M \left( \hat{y}''(x_i) + \hat{y}(x_i) - g(x_i) \right)^2$$

# Nonlinear PDE with irregular boundaries

$$\nabla^2 u(x, y) + e^{u(x, y)} = 1 + x^2 + y^2 + \frac{4}{(1 + x^2 + y^2)^2}$$

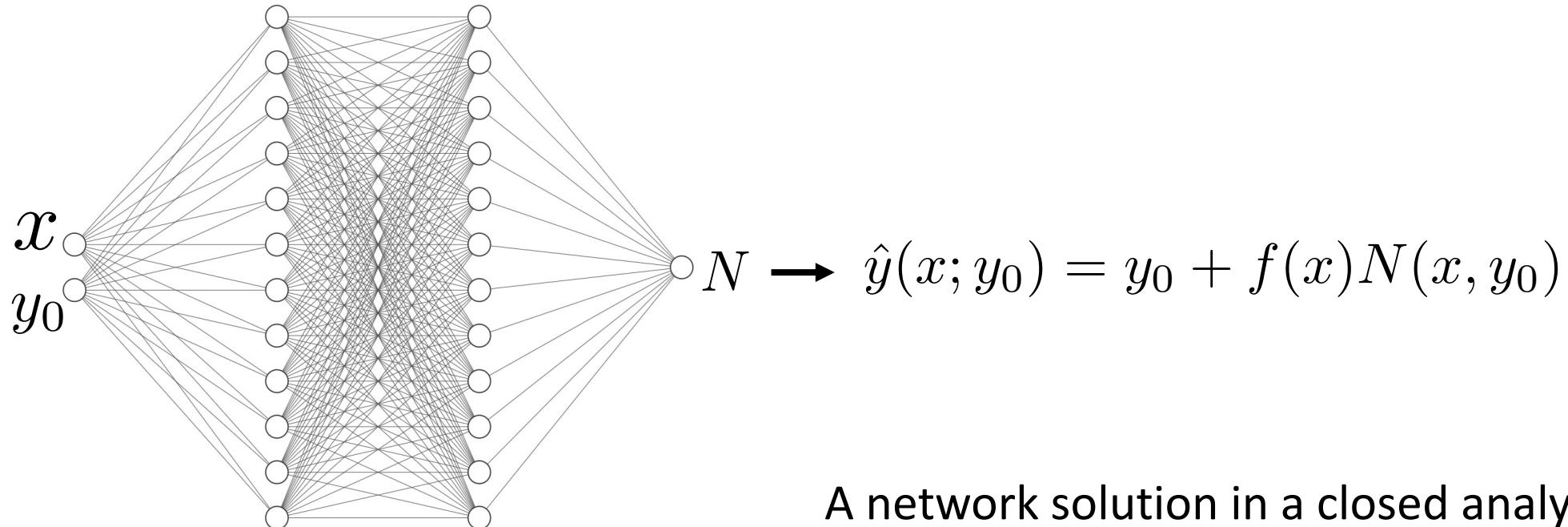
Consider Dirichlet condition along a star-shaped boundary



# Learning the general solutions

Train a network with different initial conditions

A single network learns a family of solutions



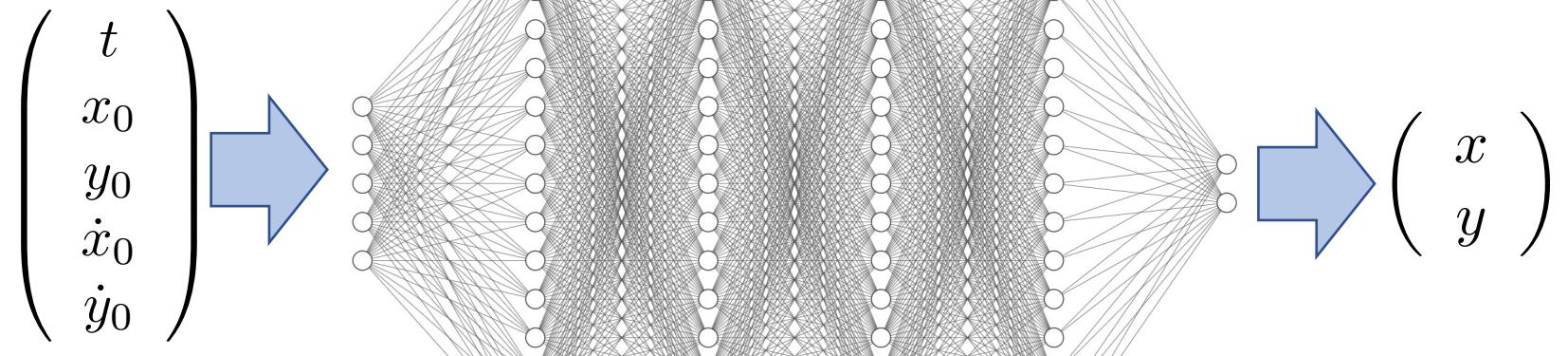
A network solution in a closed analytical form  
that is differentiable in all the inputs

# Planar circular restricted three-body problem

$$\ddot{x} = x - \mu + 2\dot{y} - \left[ \frac{\mu(x-1)}{((x-1)^2 + y^2)^{3/2}} + \frac{(1-\mu)x}{(x^2 + y^2)^{3/2}} \right]$$
$$\ddot{y} = y - 2v_x - \left[ \frac{\mu y}{((x-1)^2 + y^2)^{3/2}} + \frac{(1-\mu)y}{(x^2 + y^2)^{3/2}} \right]$$

$$\mu = \frac{m_2}{m_1 + m_2}$$

$$m_3 \approx 0$$



# Neural network solution bundles

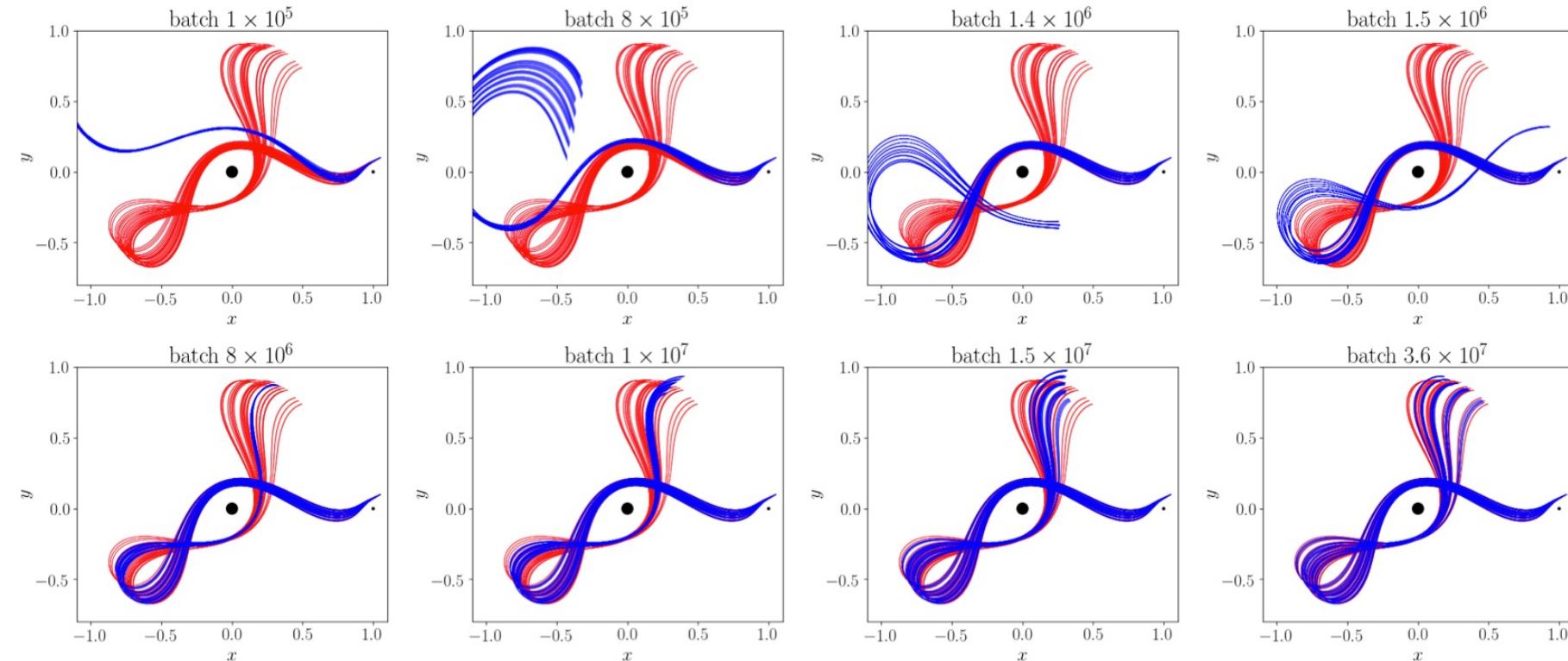


Figure 1: Plots of a few trajectories from the neural network solution bundle at various points in the training. Red trajectories are calculated with fourth-order Runge-Kutta, and the neural network solutions are shown in blue.

# Eigenvalue problems

Unsupervised data-free NN for learning pairs of eigenfunctions and eigenvalues

Time-independent Schrödinger equation (*single nonrelativistic particle*)

$$\left[ \frac{-\hbar^2}{2m} \nabla^2 + V(\mathbf{r}) \right] \Psi(\mathbf{r}) = E\Psi(\mathbf{r})$$

# Eigenvalue Problems

$$\mathcal{L}f(x) = \lambda f(x), \quad \text{with BCs} \quad f(x_L) = f(x_R) = f_b$$

Parametric solution

$$f(x, \lambda) = f_b + g(x)N(x, \lambda),$$

parametric function

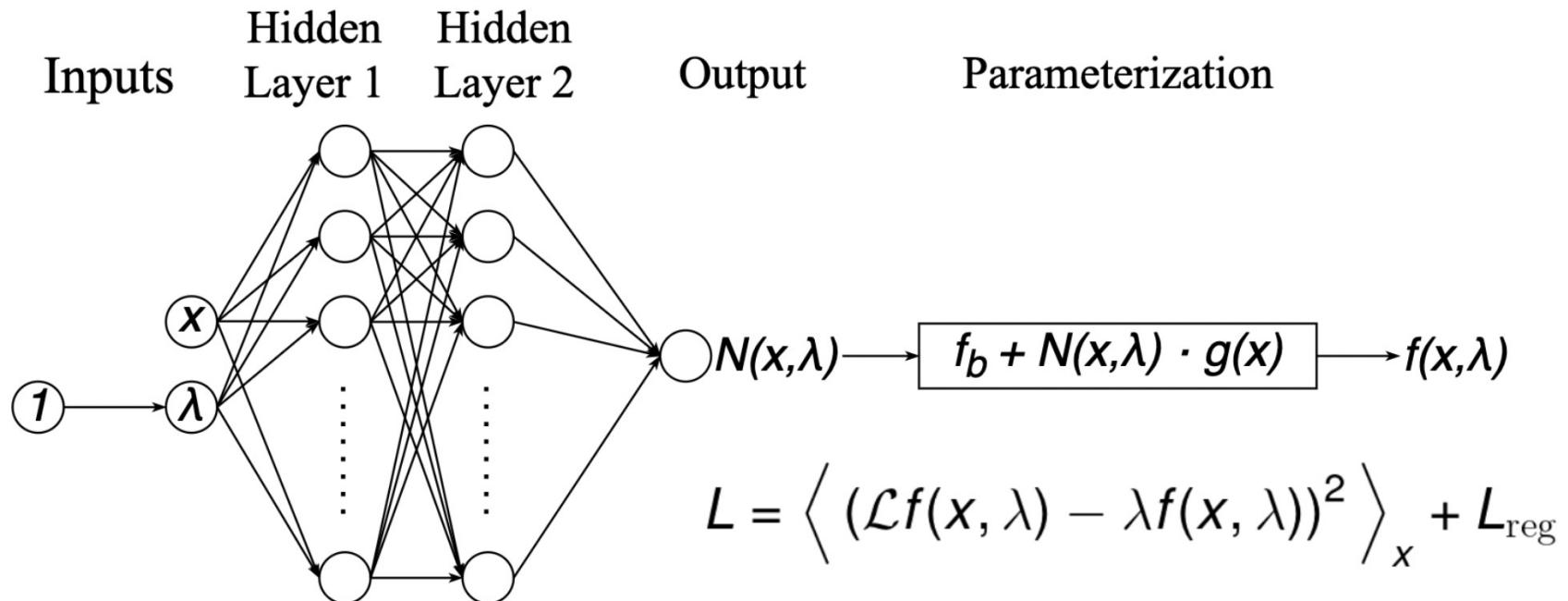
$$g(x) = \left(1 - e^{-(x-x_L)}\right) \left(1 - e^{-(x-x_R)}\right)$$

Loss function

$$\begin{aligned} L &= L_{\text{DE}} + L_{\text{reg}} \\ &= \left\langle (\mathcal{L}f(x, \lambda) - \lambda f(x, \lambda))^2 \right\rangle_x + L_{\text{reg}}, \end{aligned}$$

The regularization loss assists the network to avoid trivial solutions and to search for higher eigen-values

# Network architecture



# Regularization loss function components

Avoid trivial solutions and zero eigenvalues

$$L_f = \frac{1}{f(x, \lambda)^2}, \quad L_\lambda = \frac{1}{\lambda^2}$$

Embed a scanning mechanism that motivates the NN to search for eigenvalues/eigenfunctions

$$L_{\text{drive}} = e^{-\lambda+c} \quad (c \geq 0)$$

Once a pair is found (loss is converged), the “moving wall”  $c$  increases encouraging the NN to search for a larger eigenvalue

$$e^{-\lambda} \leq e^{-\lambda+c}$$

# Infinite square quantum well

$$\left[ -\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} + V(x) \right] \psi(x) = E\psi(x)$$

$$V(x) = \begin{cases} 0 & 0 \leq x \leq \ell \\ \infty & \text{otherwise} \end{cases}$$

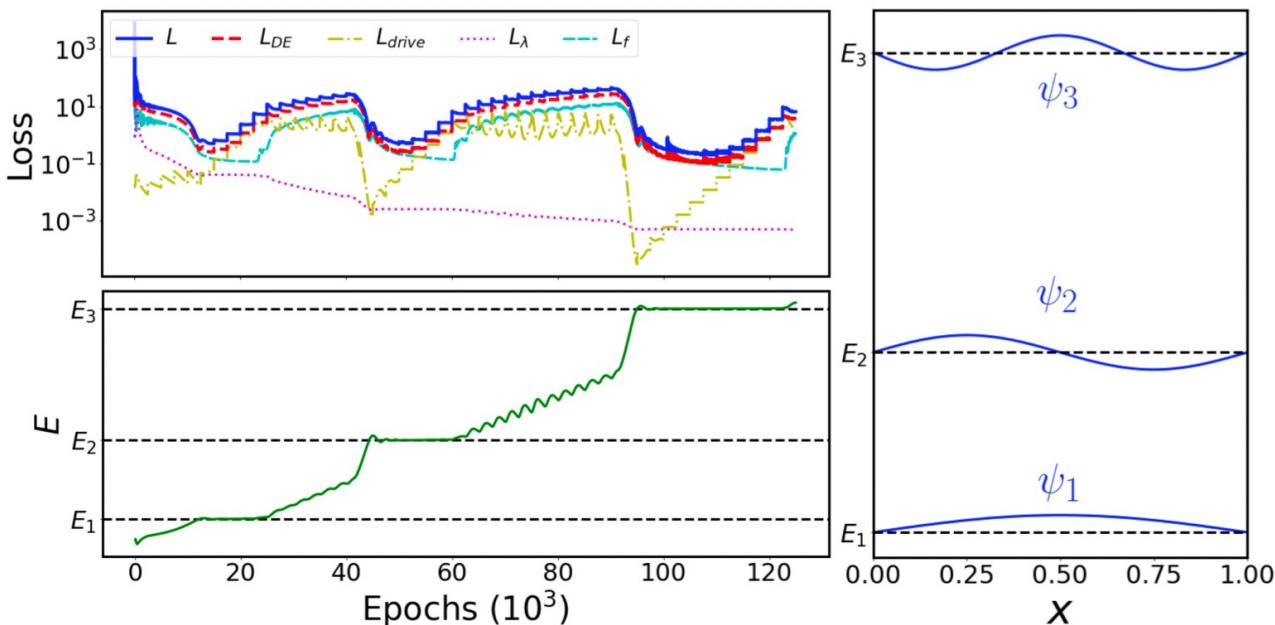


Figure 2: Infinite square well: Left panel shows the loss functions and the predicted energy during the training; dashed lines indicate the exact energy levels. Right plot outlines the predicted eigenfunctions (blue) and eigenvalues (dashed black). The errors are of the order  $10^{-3}$  and  $10^{-4}$  for  $\psi$  and  $E$ .

# Quantum harmonic oscillator

$$\left[ -\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} + V(x) \right] \psi(x) = E\psi(x)$$

$$V(x) = \frac{1}{2}kx^2$$

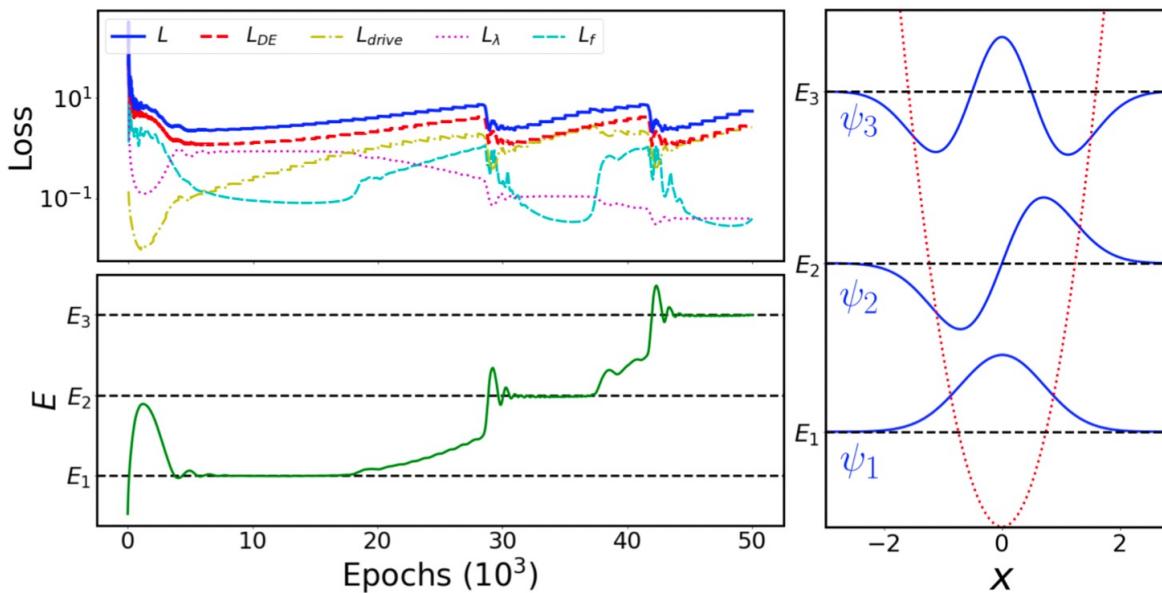
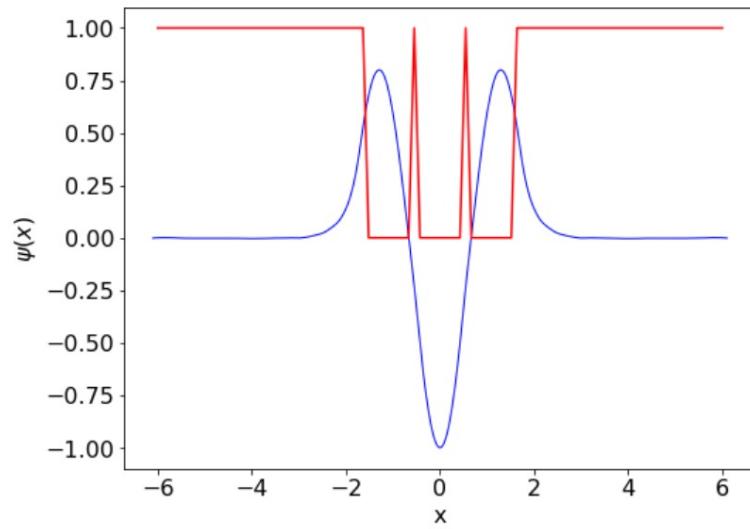


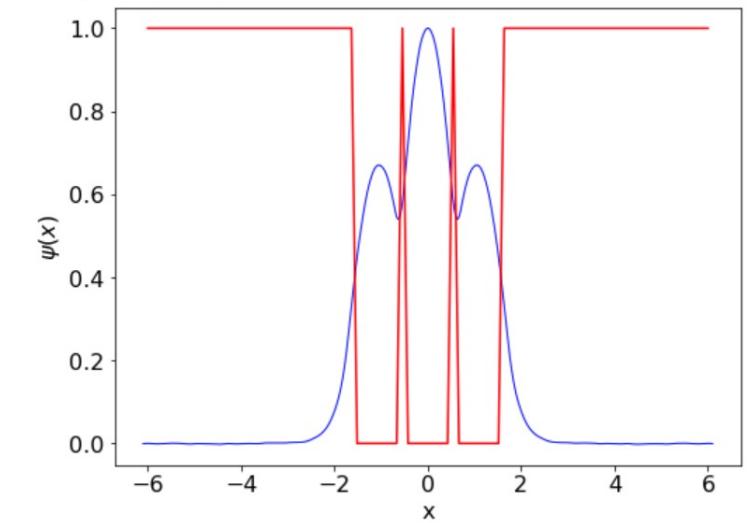
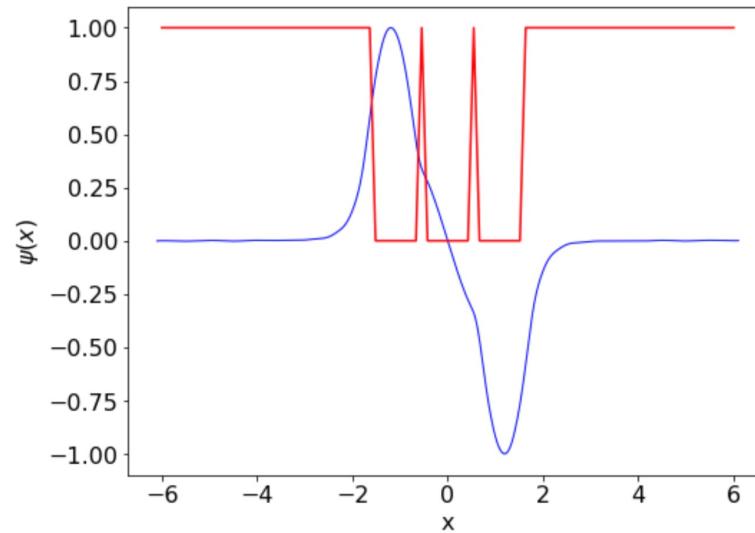
Figure 3: Quantum harmonic oscillator: Top left shows the various loss terms. Bottom left plots the history of the predicted energy. Right plot shows the eigenfunctions found by the model; the red dotted line outlines the potential. The order of magnitude of errors are  $10^{-2}$  for  $\psi$  and  $10^{-2}$  for  $E$ .

# Triple Quantum Well

```
tensor([3.0014], grad_fn=<SelectBackward>)
Text(0.5, 0, 'x')
```



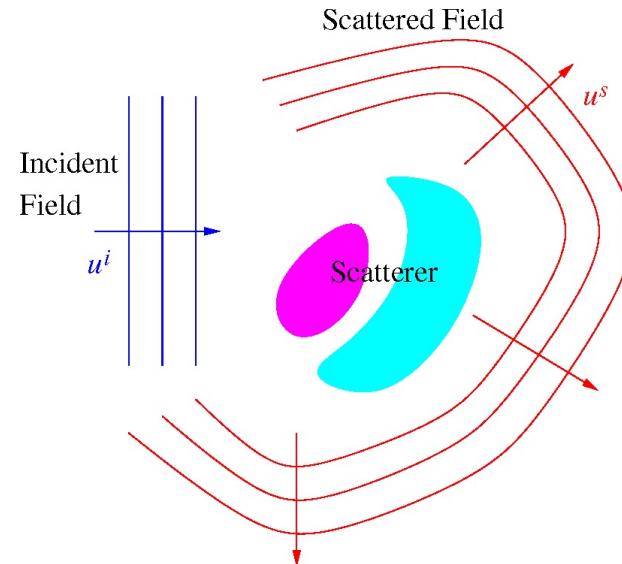
```
tensor([1.6660], grad_fn=<SelectBackward>)
Text(0.5, 0, 'x')
```



# Inverse problems

Having a general solution, which is differentiable in all the inputs, we can solve the inverse problem.

An inverse problem is the process of calculating from a set of observations the causal factors or the equations that produced them



# Solving inverse problems

- In addition to the initial conditions, we can learn general solutions for the equation's parameters.
- Once the network is trained, we set a target such as ground-truth data and optimize for the parameters/conditions to fit the target.
- The predictions fit the data and respect the underlying dynamics

# Semi-supervised networks solve inverse problems

Consider a dynamical system of differential equations

Train a network for many initial conditions  $z_0$  and model parameters  $\theta$

Gradient descent optimization determines  $z_0, \theta$  yielding solutions that fit ground truth data

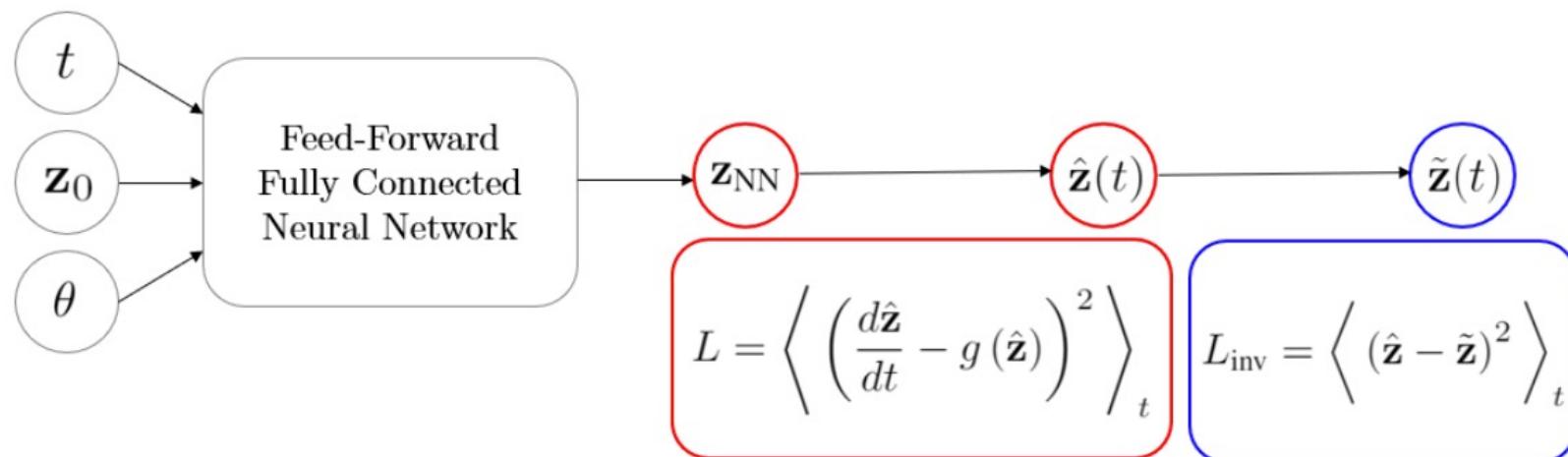


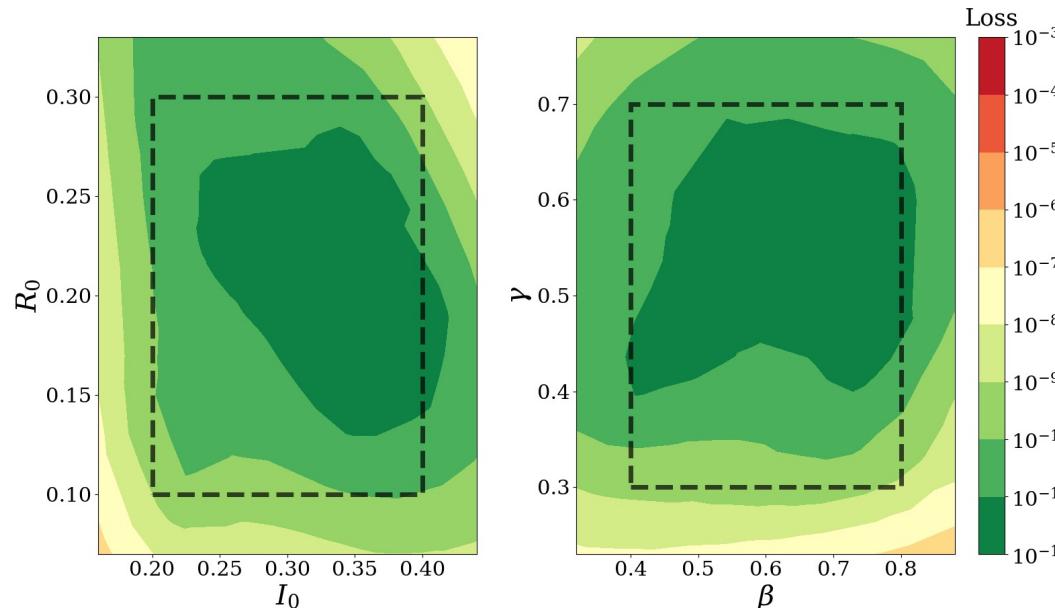
Fig. 1: Semi-supervised neural network architecture. Red and blue indicate, respectively, the unsupervised and supervised learning parts.

# SIRP epidemic model

Loss function

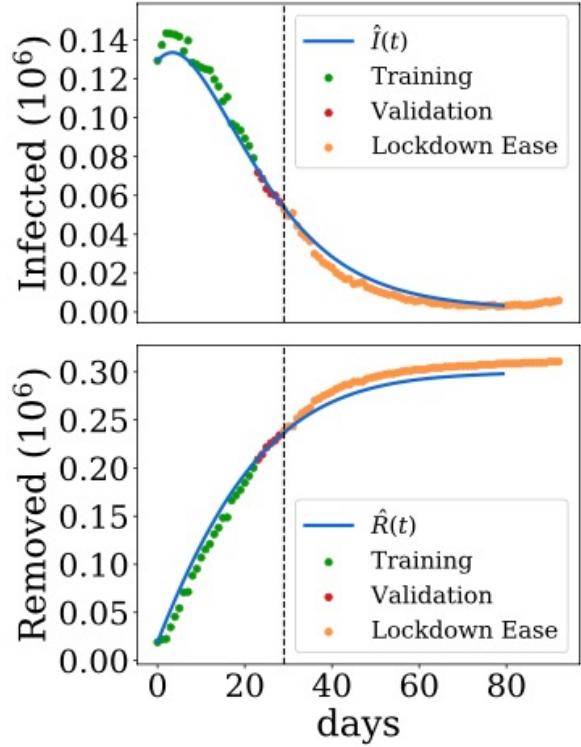
$$L = \left\langle \left( \frac{d\hat{S}}{dt} + \beta \hat{S}\hat{I} \right)^2 + \left( \frac{d\hat{I}}{dt} - \beta \hat{S}\hat{I} + \gamma \hat{I} \right)^2 + \left( \frac{d\hat{R}}{dt} - \gamma \hat{I} \right)^2 + \left( \frac{d\hat{P}}{dt} \right)^2 \right\rangle_t$$

Error analysis: Smaller validation loss in the training bundle (dashed box)

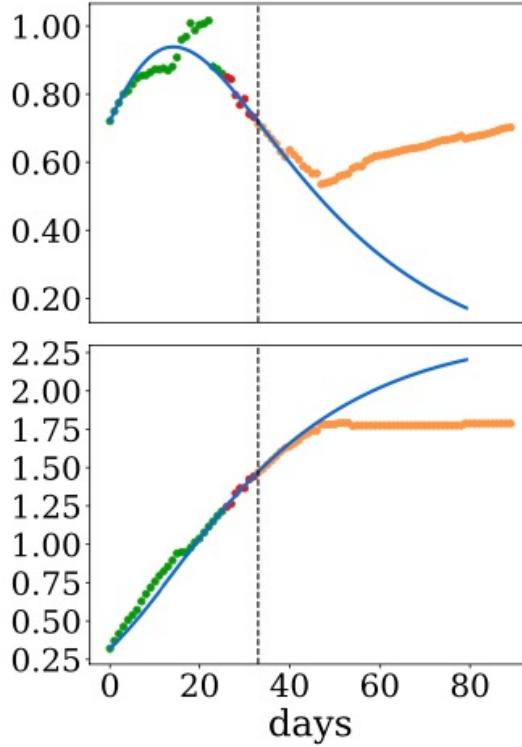


The error gradually increases outside the bundle, but the solution is still reliable

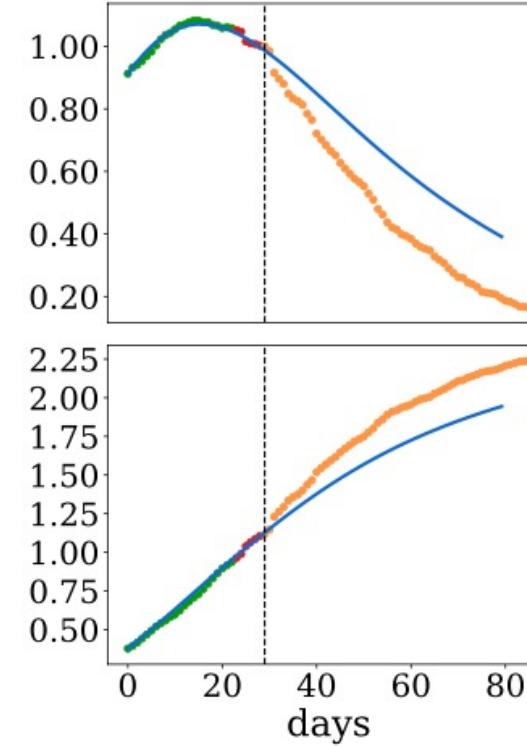
# Modeling Covid-19



Switzerland

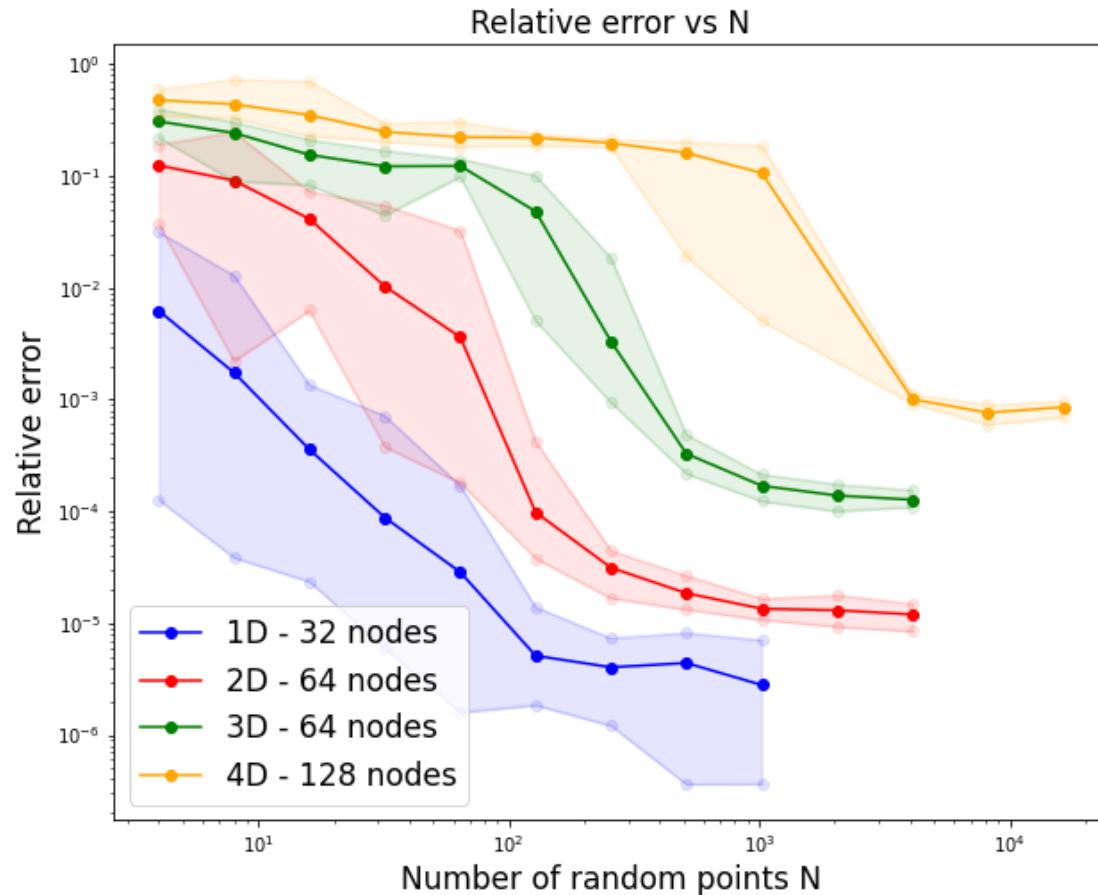


Spain



Italy

# Preliminary results for high-dimensional PDEs



# Results of our novel NN architectures

