Neural Ordinary Differential Equations

Rodrigo Arrieta
18.337 Spring 2023

What are Neural Ordinary Differential Equations (Neural ODEs)?

- Neural ODEs is a class of neural networks that, instead of modeling the data directly, it **models the derivative of the data**.
- An **ODE solver is required** to the predicted data from the modeled derivative.
- They can be understood as the **continuum limit of Recurrent Neural Networks (RNNs)**.

$$\frac{d}{dt}\mathbf{u}(t) = NN(\mathbf{u}, t, \theta)$$

$$\mathbf{u}(t) = \mathrm{ODESolver}(NN, \mathbf{u}_0, t)$$

Why are they useful?

- They are useful to model **continuous time-series data**.
- Neural ODEs do not “learn the data”, which could be complicated, they **“learn” the latent dynamics** of the system. That’s how physics works!
- Useful in modeling physical, financial, and biological systems **when theory-driven models are lacking or non-existent.**

Backpropagation

- We really do not want to backpropagate through the steps of the ODE solver: too expensive!
- Instead, we can use the **adjoint method** to compute the gradient of the cost w/r to weights.

$$\frac{d}{dt}\mathbf{u}(t) = NN(\mathbf{u}, t, \theta)$$

$$\mathbf{u}(t) = \mathrm{ODESolver}(NN, \mathbf{u}_0, t)$$

Cost:

$$G(\mathbf{u}, \theta) = \int_0^T g(\mathbf{u}(t, \theta))\, dt$$

Backpropagation

Cost: $G(\mathbf{u}, \theta) = \int_0^T g(\mathbf{u}(t, \theta))\, dt$

- First, we solve the ODE forward, from 0 to T.
- Second, we solve the adjoint ODE backwards, from T to 0.

$$
\begin{aligned}
\frac{d}{dt}\lambda &= -\frac{\partial NN}{\partial \mathbf{u}}^{\top} \lambda - \frac{dg}{d\mathbf{u}}, \\
\frac{d}{dt}\mu &= -\frac{\partial NN}{\partial \theta}^{\top} \lambda, \\
\lambda(T^+) &= 0, \\
\mu(T^+) &= 0.
\end{aligned}
$$

- The gradient is then:

$$\mu(0^-) = \frac{\partial G}{\partial \theta}$$

My project

- I implemented my **own Neural ODE adjoint solver from scratch**, fully compatible with Flux.jl
- In the next slides I’ll present some examples of interests.

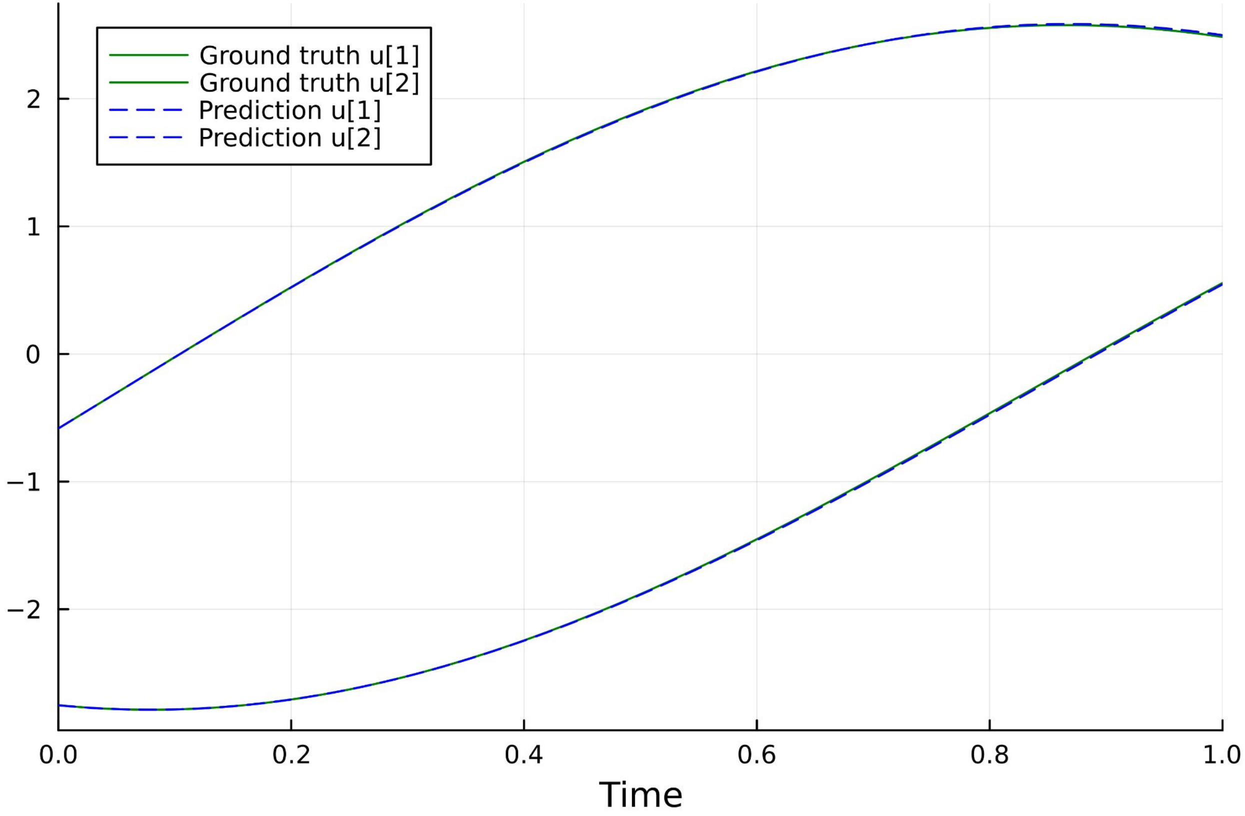
Example 1: Linear system

- Trained the NN for a fixed interval [0,1] and various random initial conditions.
- NN: 1 hidden layer, 64 hidden units

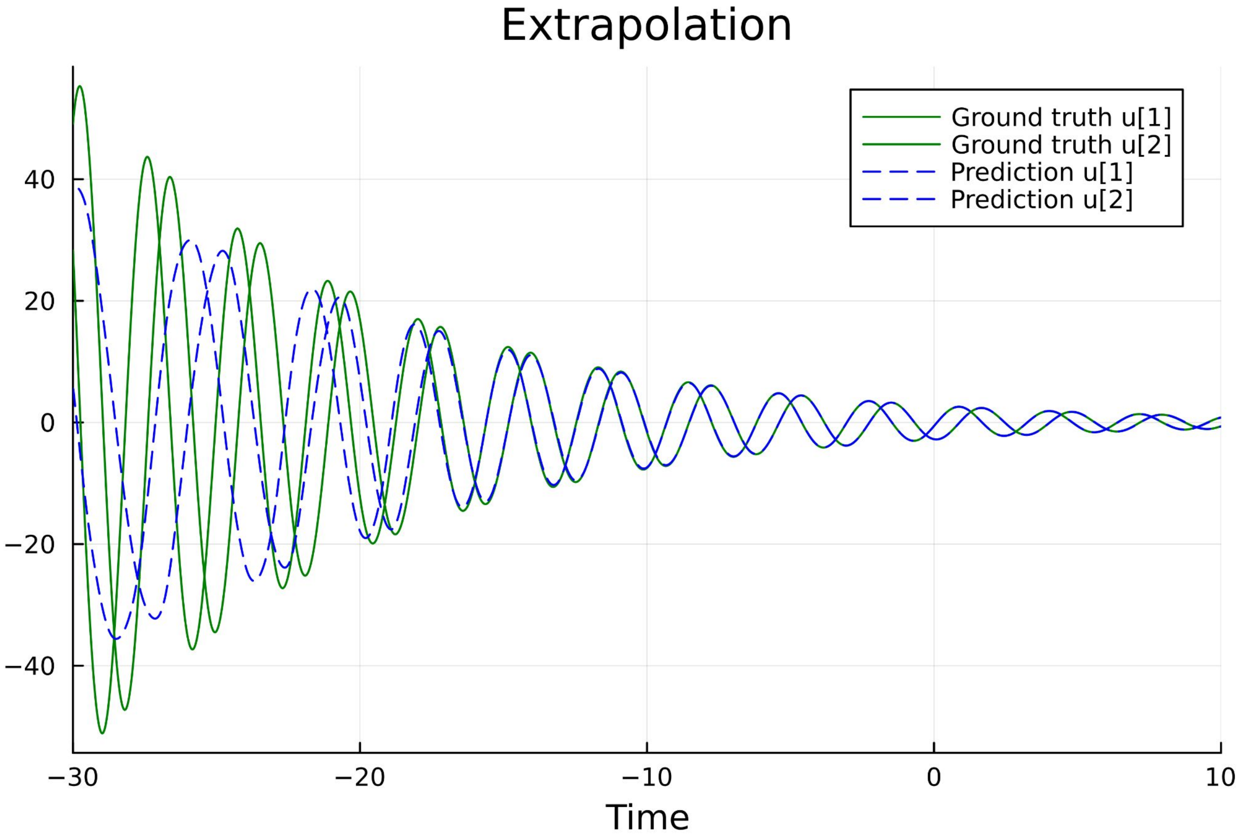
$$\frac{d}{dt}\mathbf{u}(t) = A\mathbf{u}(t),$$

$$A = \begin{pmatrix} -0.1 & 2 \\ -2 & -0.1 \end{pmatrix}$$

Example 1: Linear system



Example 1: Linear system

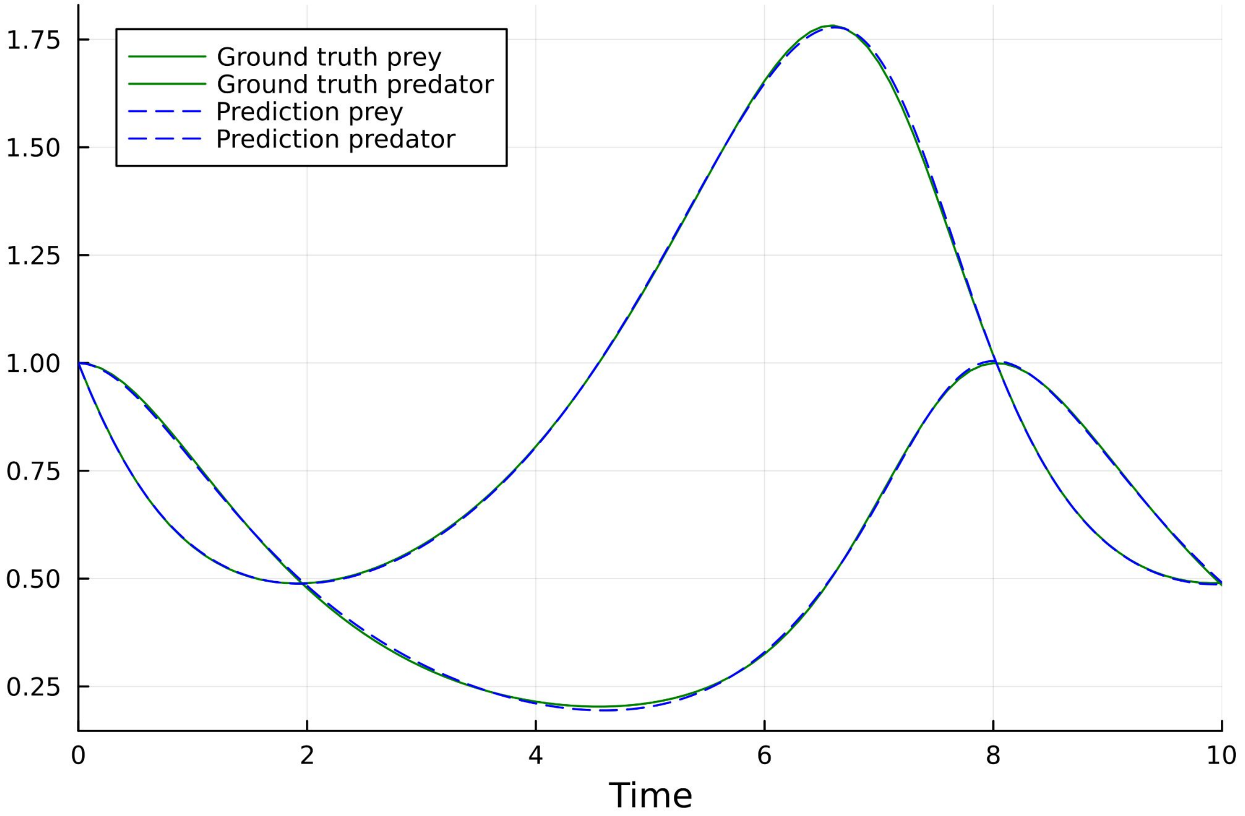


Example: Lotka-Volterra eqs.

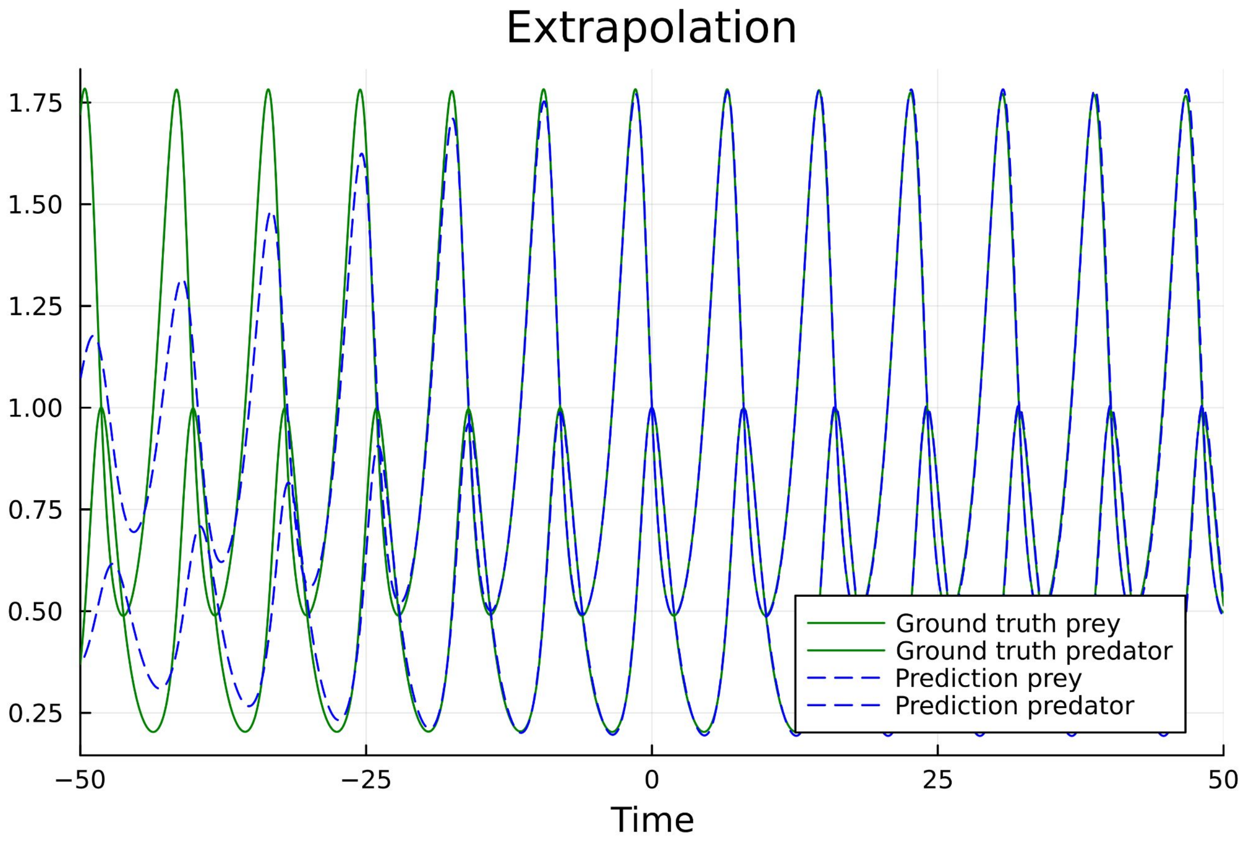
- Trained the NN for a fixed interval and initial cond.
- This is a **nonlinear** system with **periodic** behavior.

$$
\begin{aligned}
\frac{dx}{dt} &= \alpha x - \beta x y, \\
\frac{dy}{dt} &= \delta x y - \gamma y,
\end{aligned}
$$

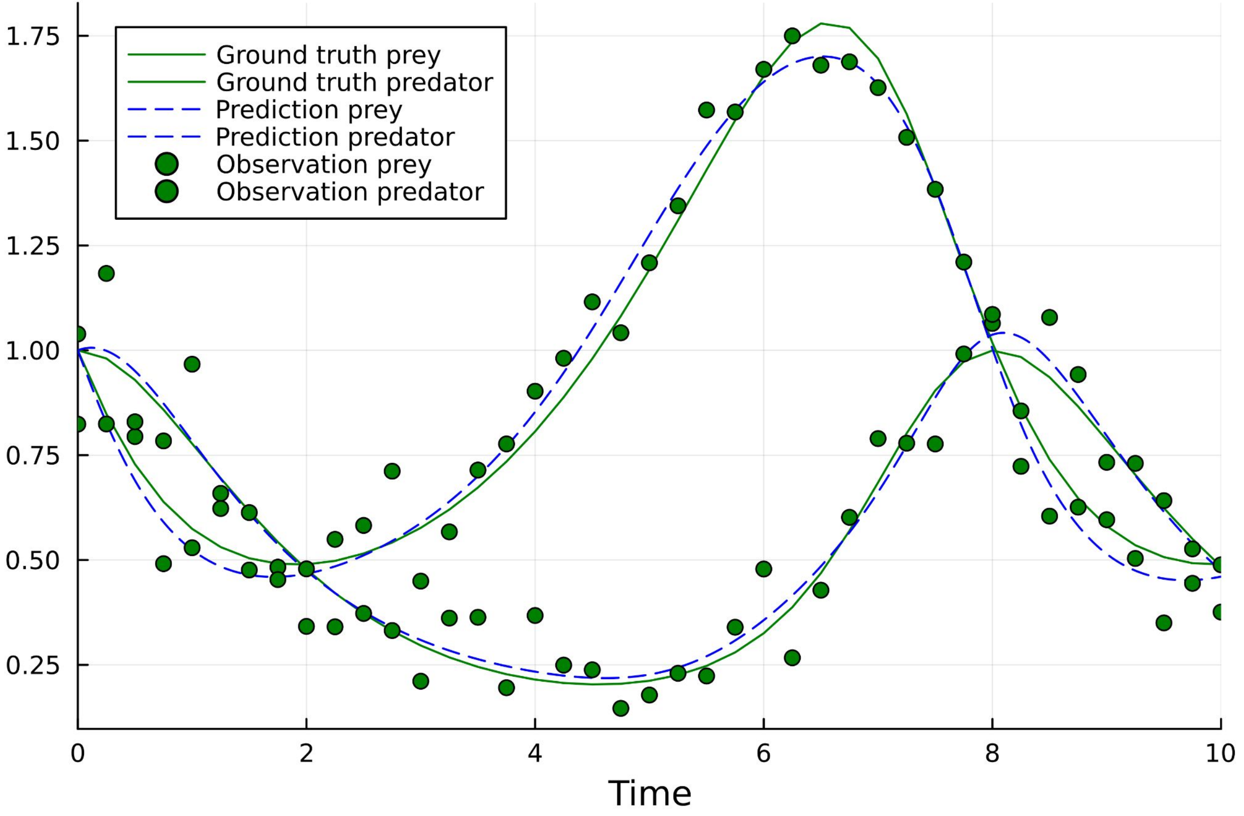
Example: Lotka-Volterra eqs.



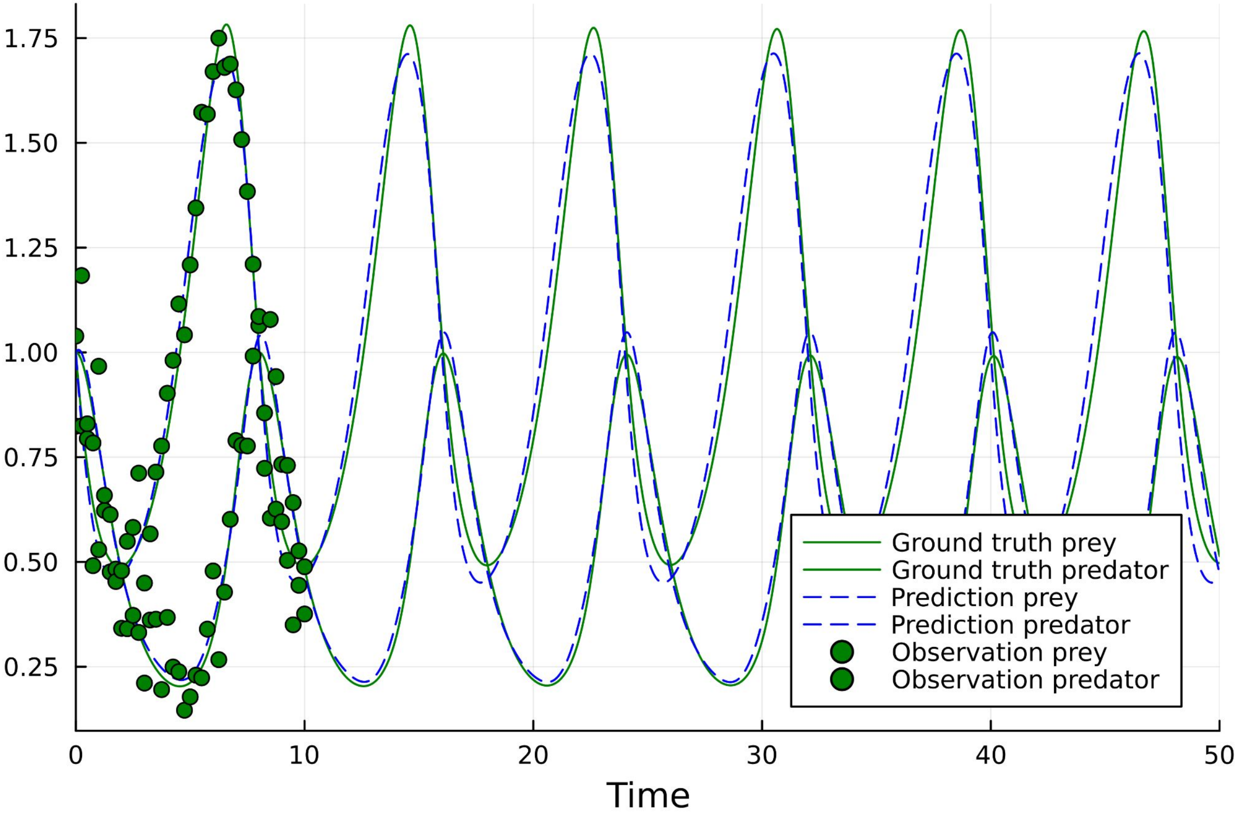
Example: Lotka-Volterra eqs.



Example: Lotka-Volterra eqs., with **noise**



Example: Lotka-Volterra eqs., with **noise**

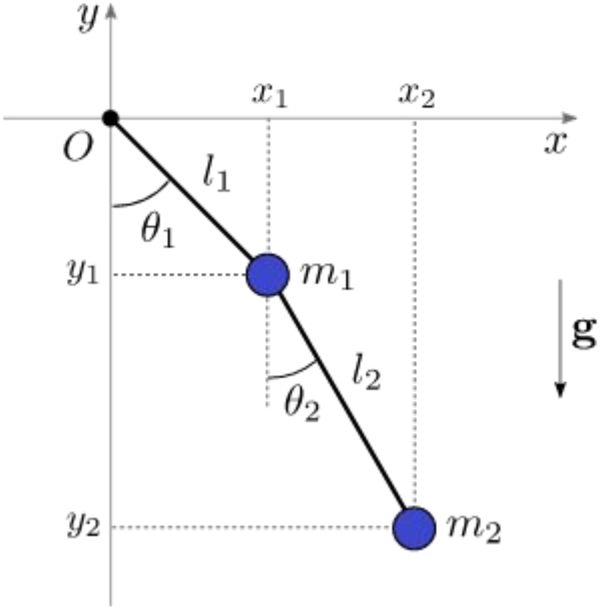


Example: double pendulum

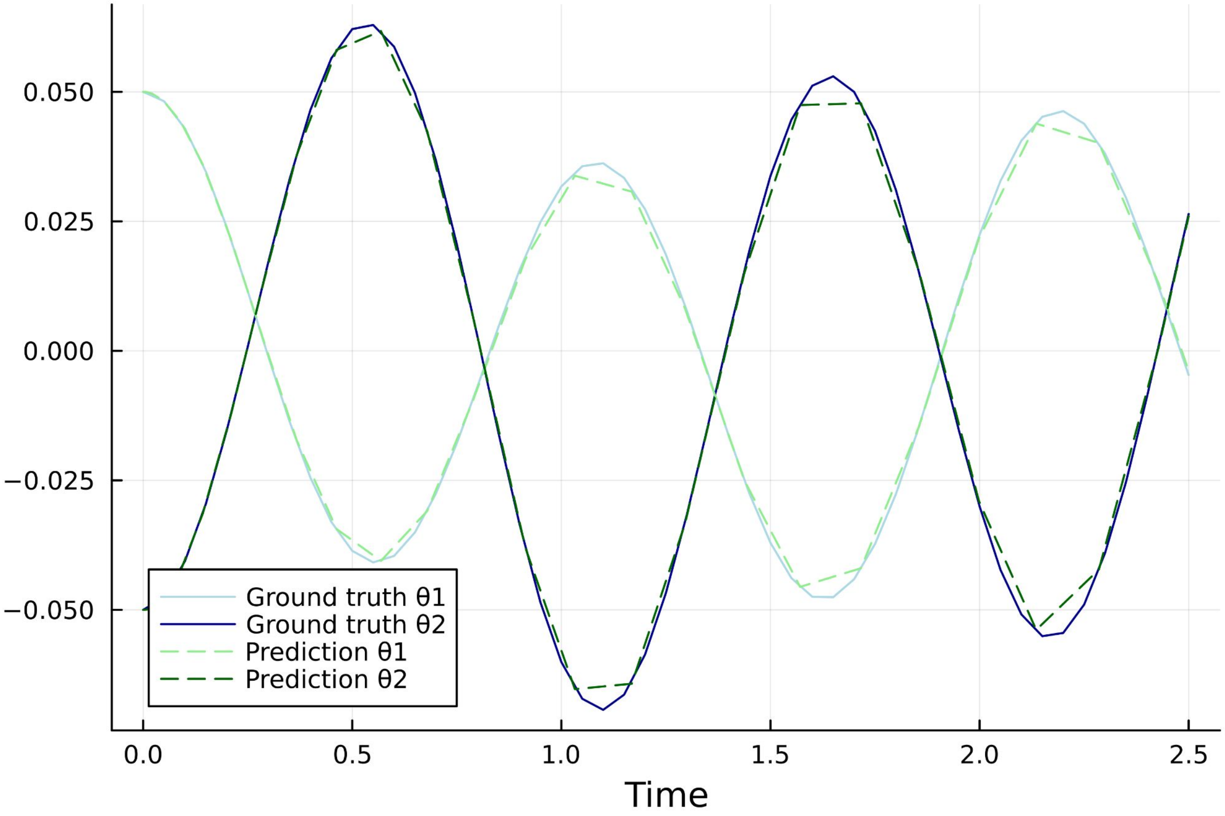
- Trained the NN for a fixed interval and initial cond.
- This is a **nonlinear** system with **aperiodic** behavior.

$$H = \frac{m_2 l_2^2 p_{\theta_1}^2 + (m_1 + m_2) l_1^2 p_{\theta_2}^2 - 2 m_2 l_1 l_2 p_{\theta_1} p_{\theta_2} \cos(\theta_1 - \theta_2)}{2 m_2 l_1^2 l_2^2 \left[m_1 + m_2 \sin^2(\theta_1 - \theta_2)\right]}$$
$$-(m_1 + m_2) g l_1 \cos\theta_1 - m_2 g l_2 \cos\theta_2$$

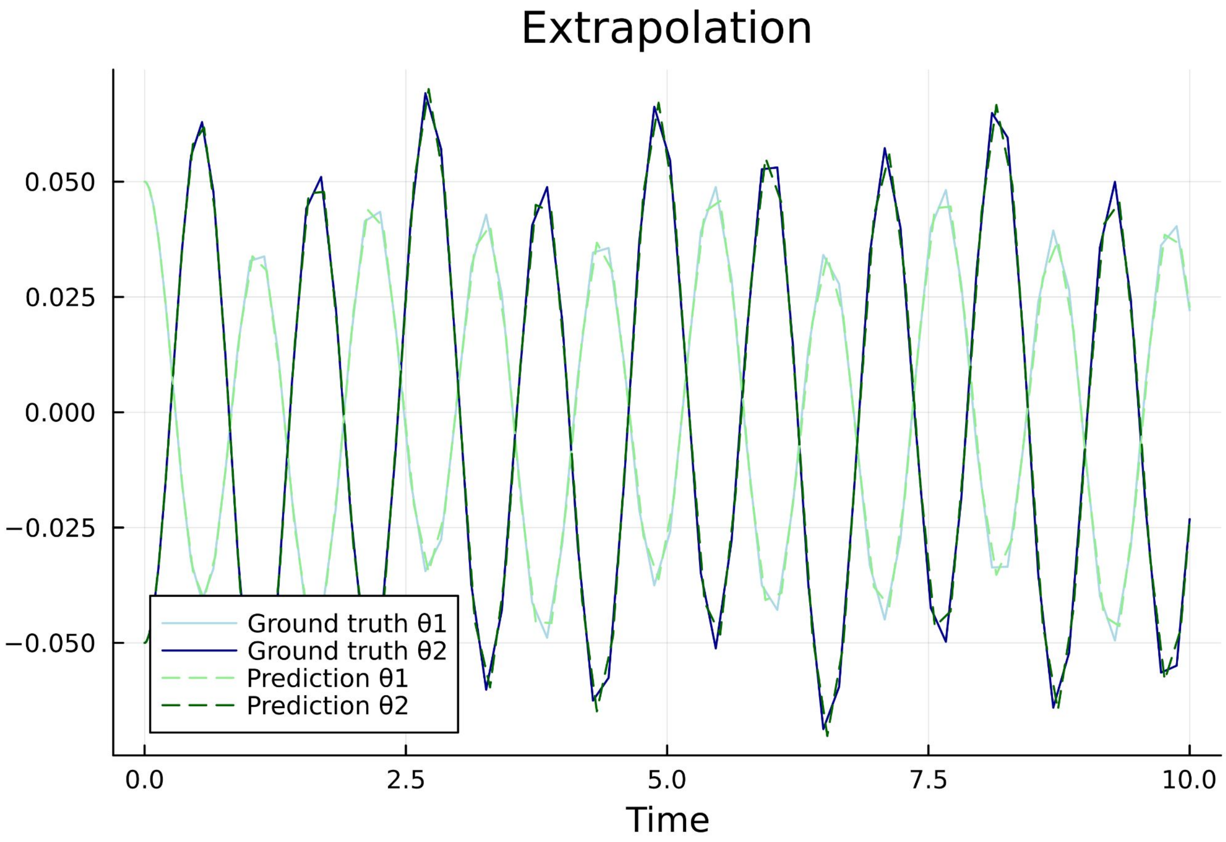
$$\frac{d\mathbf{q}}{dt} = \frac{\partial H}{\partial \mathbf{p}},$$
$$\frac{d\mathbf{p}}{dt} = -\frac{\partial H}{\partial \mathbf{q}}.$$



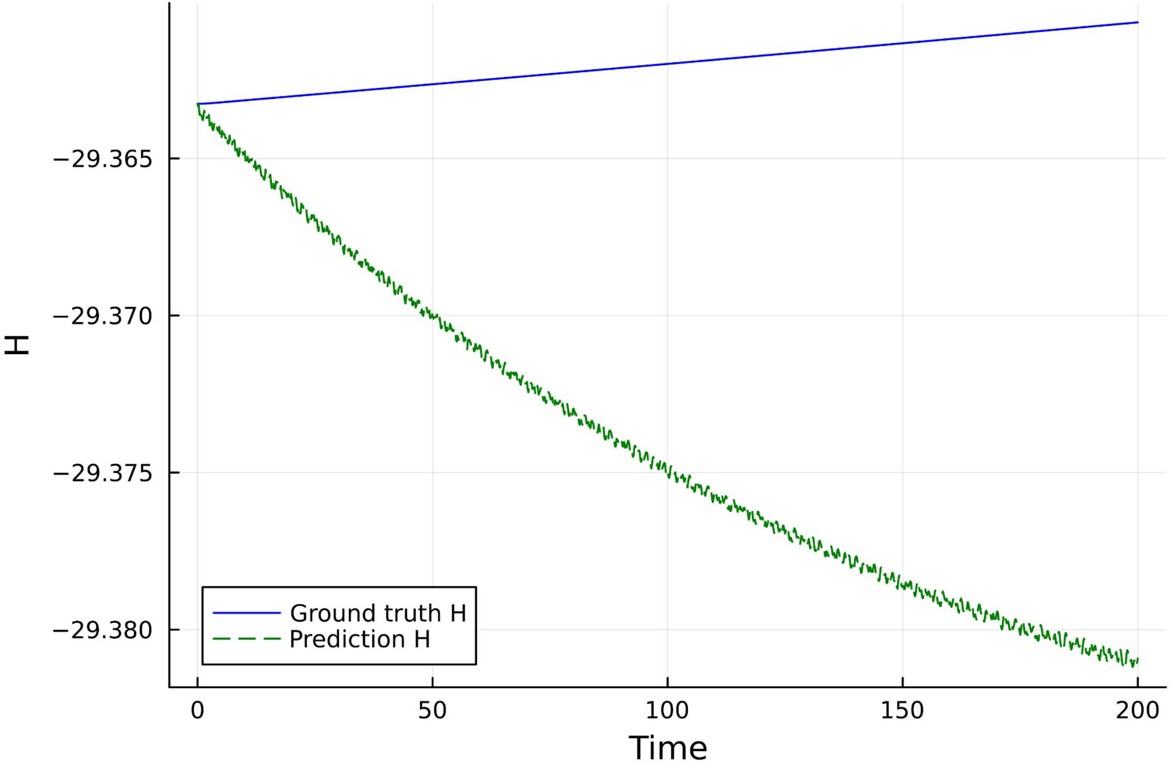
Example: double pendulum



Example: double pendulum



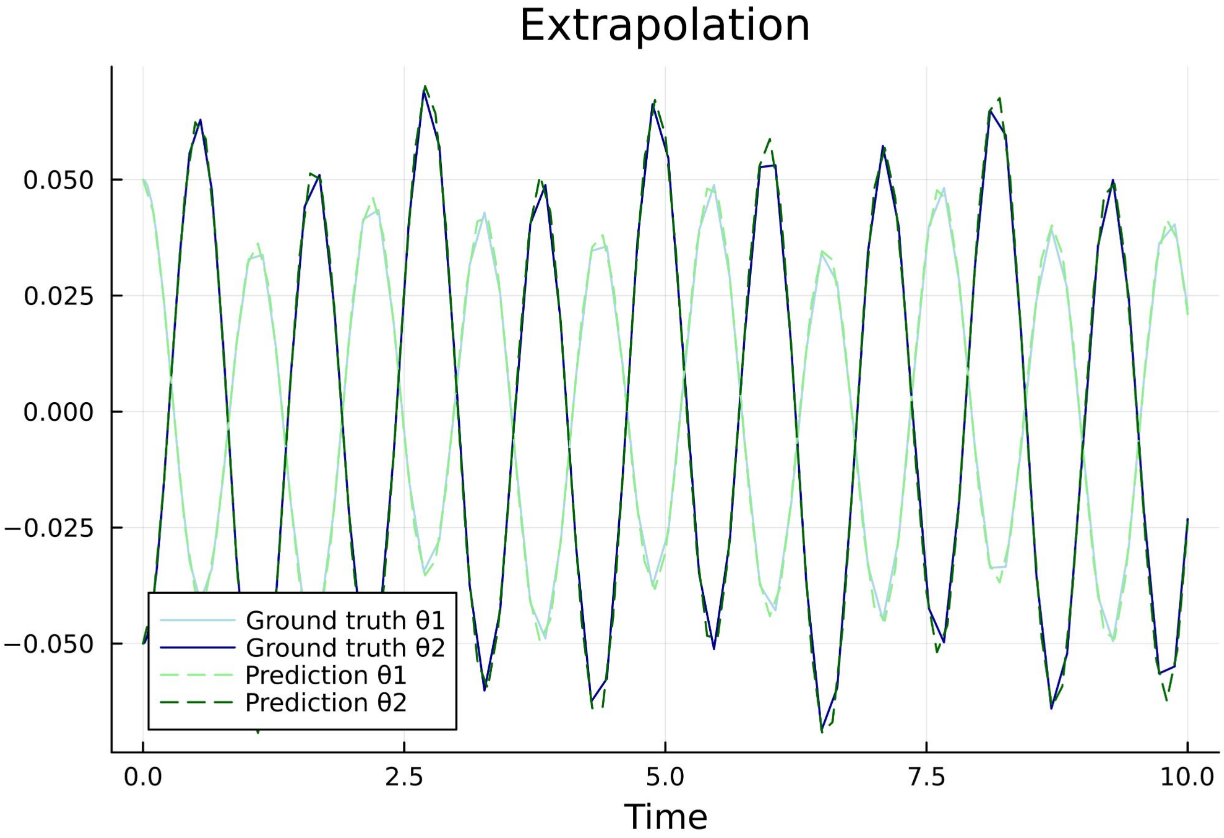
Example: double pendulum



Solution: Hamiltonian Neural Networks (HNNs)!

- The NN models the Hamiltonian instead of the derivative of the system.
- I implemented HNNs using DiffEqFlux.jl.

Example: double pendulum with HMMn



Example: double pendulum with HMMn