Investigating Neural ODEs

Jacob Rottenberg

May 8 2024

Introduction

Ordinary differential equations (ODEs) are very prevalent in life ranging from describing object motion to being able to describe some biological processes and functions. In order to use data to model a system we know follows an ODE, we need to be able to approximate it with a computer for more complex ones. This does not only involve learning parameters to systems, but the dynamics themselves.

Background

- Recurrent Neural Networks which were previous used to model ODEs due to their hidden state: $h_{t+1} = h_t + f(h_t, \theta_t)$
- However, this models a discrete system rather than a continuous one, which would be more helpful in general ODE situations.
- As proposed by Chen et al., we can add more layers and take smaller steps and in the limit parameterize the continuous dynamics using an ode: $\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t, \theta)$
- This approach looks towards continuous dynamics, can utilize a black box ODE solver, and can be more memory efficient in deep networks since intermediate quantities on the forward pass would not need to be stored.

Background

- This approach introduces a lot of difficulty in backpropagation. If we consider optimizing the loss as:

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta))$$

- As proven by Chen et al., defining the adjoint $\mathbf{a}(t) = \frac{dL}{d\mathbf{z}(t)}$ gives:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$

and

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(T)^\top \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}} dt$$

Formulation

- Using the Neural ODE approach, I aimed to investigate its capabilities on learning the dynamics of different ODEs.
- I looked at three main ODEs: A simple linear one, the Van Der Pol differential equation, and the Lorenz system.
- Since ODE solvers can only handle first-order ODEs, they had to be reformulated in this way, if not already, which is not a problem since it can be done easily.

Formulation

- Linear ODE (Spiral Sink):

$$\frac{dx}{dt} = \begin{bmatrix} -\frac{1}{10} & 1 \\ -1 & -\frac{1}{10} \end{bmatrix} x^3 \text{ with initial condition: } x(0) = \begin{pmatrix} 1.5 \\ 0 \end{pmatrix}$$

- Van Der Pol:

$$\frac{d^2x}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x = 0 \text{ which maps to the system:}$$

$$\frac{dx}{dt} = y$$

$$\frac{dy}{dt} = \mu(1 - x^2)y - x$$

Formulation

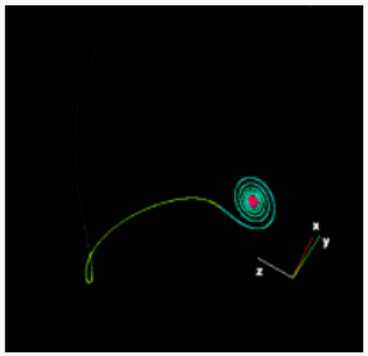
- Lorenz:

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = x(\rho - z) - y$$
$$\frac{dz}{dt} = xy - \beta z$$

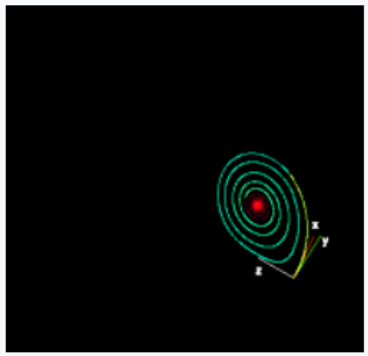
- When ρ is small, the system is stable, but when ρ is large, the system becomes very chaotic.

Formulation

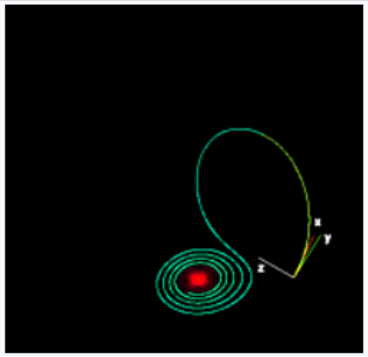
Example solutions of the Lorenz system for different values of ρ



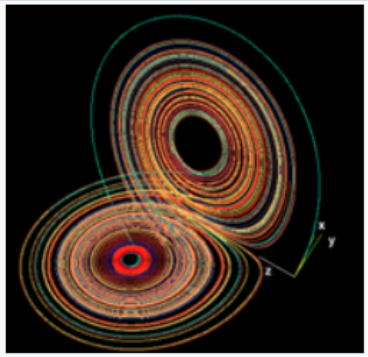
$\rho = 14, \sigma = 10, \beta = \frac{8}{3}$ (Enlarge)



$\rho = 13, \sigma = 10, \beta = \frac{8}{3}$ (Enlarge)



$\rho = 15, \sigma = 10, \beta = \frac{8}{3}$ (Enlarge)

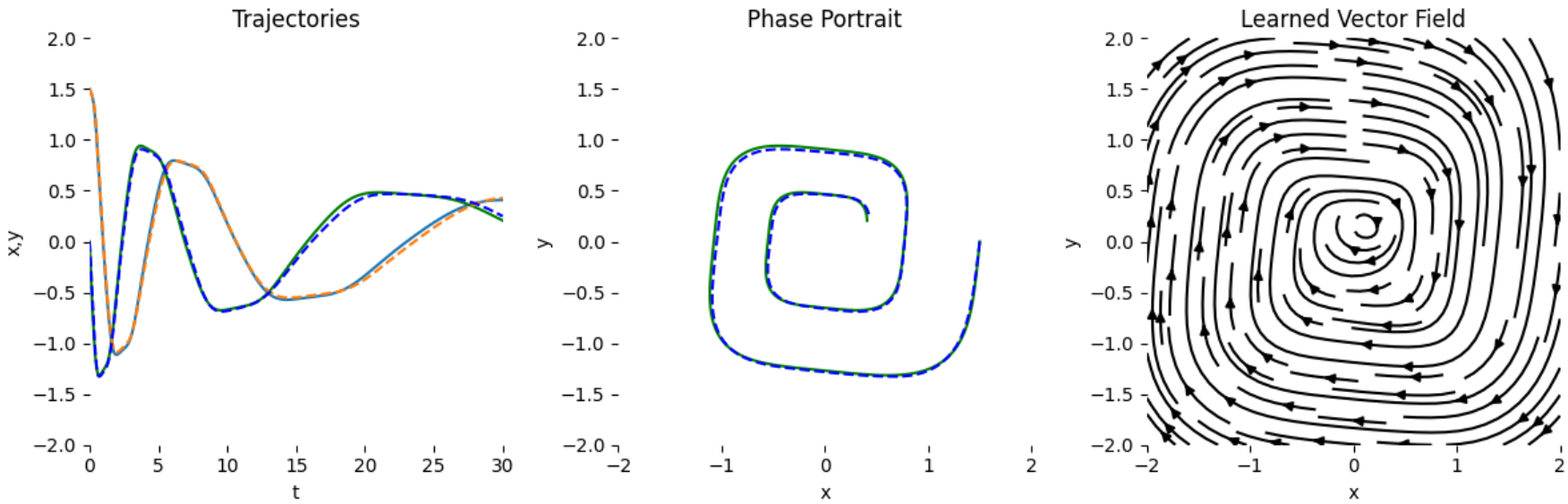


$\rho = 28, \sigma = 10, \beta = \frac{8}{3}$ (Enlarge)

For small values of ρ, the system is stable and evolves to one of two fixed point attractors. When $\rho > 24.74$, the fixed points become repulsors and the trajectory is repelled by them in a very complex way.

[1]*Graphic courtesy of Wikipedia

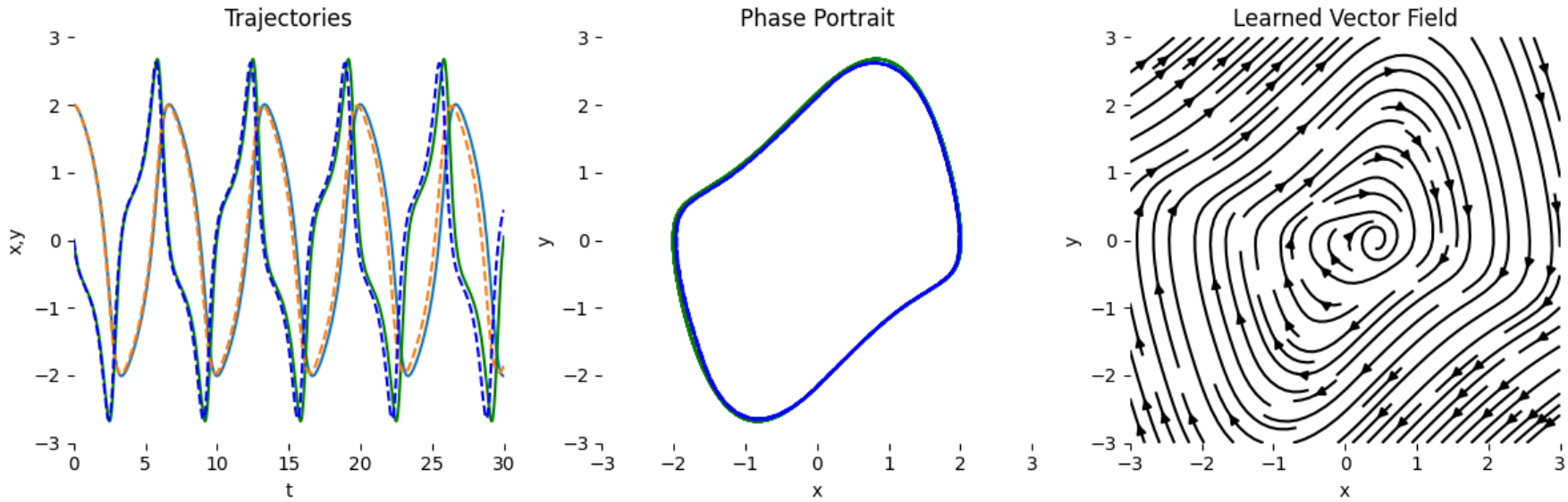
Results



Simple ODE with initial condition $\begin{pmatrix} 1.5 \\ 0 \end{pmatrix}$.

Network architecture:
$Linear(2, 64) \rightarrow LeakyReLU \rightarrow Linear(64, 64) \rightarrow LeakyReLU \rightarrow Linear(64, 2)$

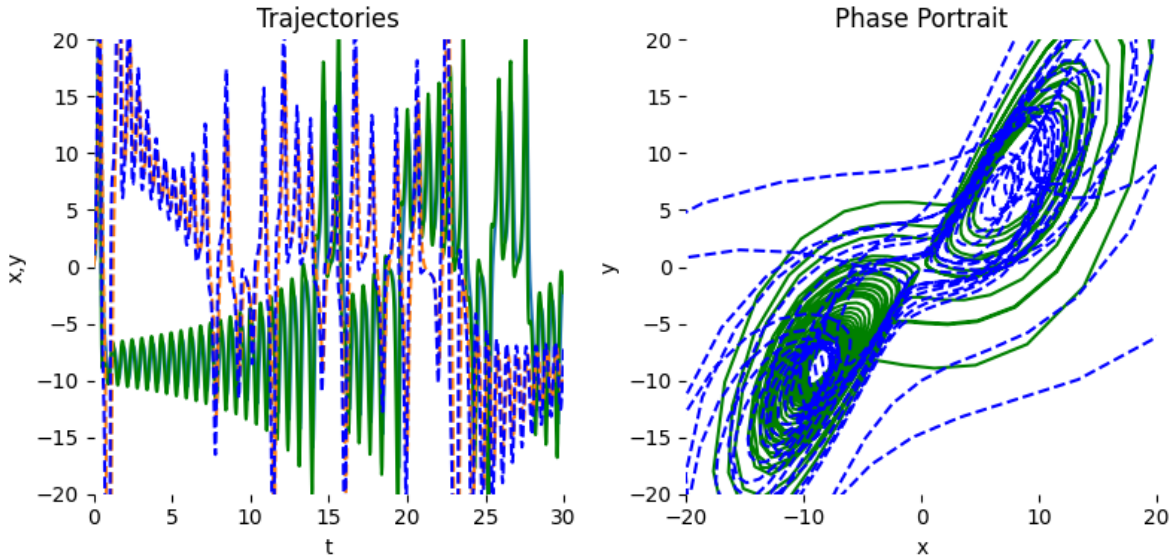
Results



Van der Pol with $\mu = 1$, initial condition $\begin{pmatrix} 2 \\ 0 \end{pmatrix}$.

Network architecture:
$Linear(2, 64) \rightarrow ReLU \rightarrow Linear(64, 64) \rightarrow ReLU \rightarrow Linear(64, 2)$

Results



Lorenz with $\rho = 28, \sigma = 10, \beta = \frac{8}{3}$ initial condition $\begin{pmatrix} 0 \\ 1 \\ 1.05 \end{pmatrix}$.

Network architecture:
$Linear(3, 64) \to LeakyReLU \to Linear(64, 64) \to LeakyReLU \to Linear(64, 64) \to LeakyReLU \to Linear(64, 3)$

Conclusion

To conclude, the Neural ODE does learn dynamical systems, though it can struggle to learn the exact parameters to follow the trajectories. However, it is clear from the initial experiments that overall dynamics. The Van Der Pol limit cycle is learned with the correct geometry, and with the Lorenz system, the network learns the chaotic nature when $\rho = 28$, but not necessarily when to switch "lobes."
This project is not quite finished and in the report I will include details on the loss between trajectories as well as different system parameters for the Van Der Pol equation and Lorenz system and more initial conditions.