

Learning Dynamical Systems Through ODE Inspired Neural Networks

Emma Hayes, Mathias Heider, and Carrie Vanty

Mentor: Deepanshu Verma
Emory Math REU

June 30, 2022

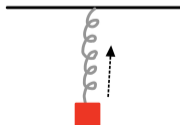
Motivation

Our main goal for this project is to compare Hamiltonian Inspired Neural Networks (HINNs) and Sparse Identification of Nonlinear Dynamics (SINDy), which are different ways to learn dynamics of ODEs from data.

- One example problem is finding the coordinates of a spring mass system at any given time value.
- This is a special type of differential equation, called a Hamiltonian differential equation, and has the property that energy is conserved. It is defined as

$$\frac{dy}{dt} = \frac{\partial \mathcal{H}}{\partial z}, \quad \frac{dz}{dt} = -\frac{\partial \mathcal{H}}{\partial y}.$$

- For the spring mass system, $\mathcal{H}(t, y, z) = \frac{k}{2m}y^2 + \frac{1}{2}z^2$



- Machine Learning is the use of statistical learning and optimization methods that let computers analyze datasets and identify patterns
- Deep learning is a type of machine learning and artificial intelligence (AI) that imitates the way humans gain certain types of knowledge
- Neural Networks are an example of deep learning and are inspired by how the human brain works as they mimic the way neurons signal.

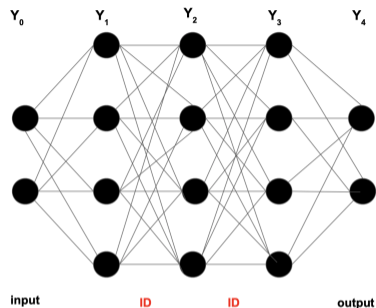
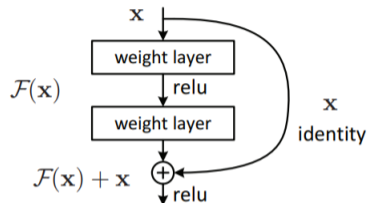


Figure: Residual Neural Network

- It creates an adaptive system that computers use to learn from their mistakes and improve continuously.
- In a neural network you have an input layer, output layer, and a varying number of hidden layers.



Residual Neural Networks (RNN) solve the issue that the deeper you go in neural network the higher the training and test error (vanishing/exploding gradient) [5]

We can write the forward propagation between Y_2 and Y_3 as:

$$Y_2 = Y_1 + \sigma(K_2 Y_1 + b_2)$$

$$Y_3 = Y_2 + \sigma(K_3 Y_2 + b_3)$$

It was discovered that our equations for forward propagation were nearly identical to the discretization of an ODE [5], except for the step size, denoted h

$$Y_2 = Y_1 + \mathbf{h}\sigma(K_2 Y_1 + b_2)$$

$$Y_3 = Y_2 + \mathbf{h}\sigma(K_3 Y_2 + b_3)$$

If we think of this a system, we can create the ODE:

$$Y'(t) = \sigma(K(t)Y(t) + b(t))$$

ODEs in Residual Neural Networks

Our general forward propagation written as the discretization of an ODE [5]:

$$Y_{j+1} = Y_j + h\sigma(Y_j K_j + b_j)$$

Translated into our neural network:

- In every layer, we move one step forward in the discretization of the ODE
- The weights and biases may be different between layers based on the given ODE
- The output of the network creates an ODE that matches the given data set

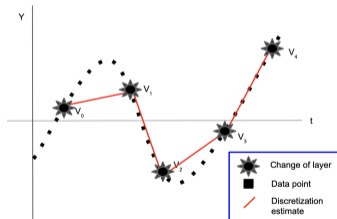


Figure: Discretization of an ODE

Hamiltonian Inspired Neural Networks

Let's again consider the spring mass system. We are going to use a Hamiltonian ODE of the form:

$$Y''(t) = \sigma(K^T(t)Y(t) + b(t))$$
$$Y(0) = Y_0 \text{ and } Y'(0) = Y'_0$$

This can be turned into a system with first order differential equations [5]:

$$Y'(t) = \sigma(K(t)Y(t) + b(t))$$
$$Z'(t) = -\sigma(K^T(t)Y(t) + b(t))$$
$$Y(0) = Y_0 \text{ and } Z(0) = 0$$

To solve this, we are given the t -values and want to find the coordinates, y, z . We would then:

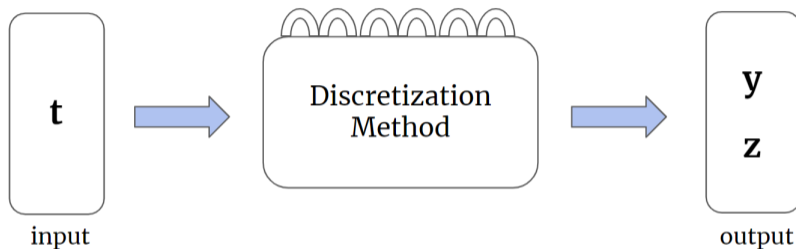
- discretize the given network ODE
- set the discretization as our forward propagation

Forward propagation can be thought of as discretization of an ODE, but there are many different discretization techniques with different strengths which can improve training.

For Hamiltonian ODEs, [symplectic](#) methods work the best.

- They capture features of Hamiltonian
- They are structure preserving [6]

Implementation



- One Input: Time
- Two Neural Nets: RNN, HINN
- Discretization Methods: Euler's Method, Runge-Kutta Method, Verlet Method
- Two Outputs: y and z

Euler's Method

Our initial ResNet was built with forward Euler.

$$Y_{j+1} = Y_j + h\sigma(Y_j K_j + b_j)$$

Euler is a first order technique with local truncation error in $O(h^2)$. Euler is an explicit method that is not symplectic, so it has some limitations.

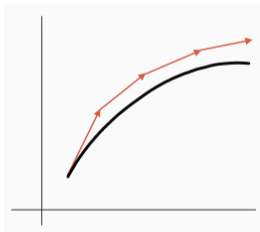


Figure: Simple Euler's method depiction

Runge-Kutta Method

RK4 uses a series of slopes at a few points and takes their weighted average to find the solution at a future time step ([1], [2]).

$$Y_{j+1} = Y_j + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

RK4 is an explicit method which is not symplectic. RK4 uses Euler's method and is three orders of magnitude more accurate than Euler [3].

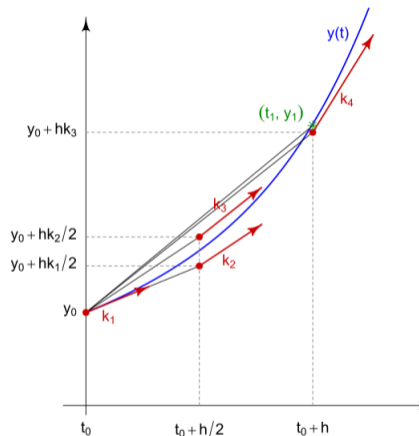


Figure: RK4 Method on a simple curve, each k_i slope is shown [8]

Verlet Method

Verlet is symplectic and second order, so it works especially well with our Hamiltonians which are second order and conserve energy ([1], [2]).

$$\begin{aligned}z_{j+\frac{1}{2}} &= z_{j-\frac{1}{2}} - h\sigma(K_j^T y_j + b_j) \\ y_{j+1} &= y_j + h\sigma(K_j z_{j+\frac{1}{2}} + b_j)\end{aligned}$$

The Verlet method allows us to integrate a second order ODE without needing to calculate the first derivative [5]. Applications from physics may look familiar.

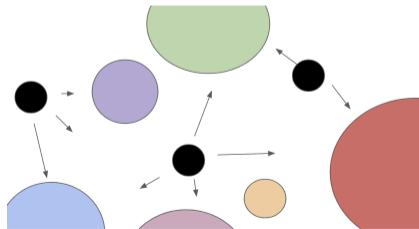
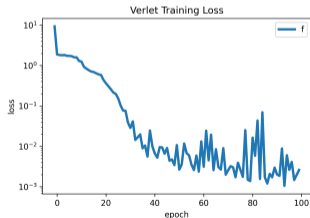
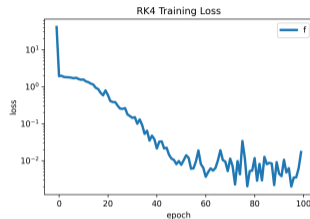
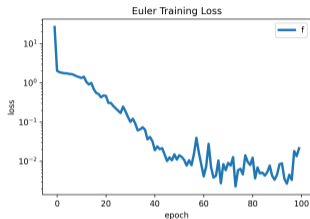


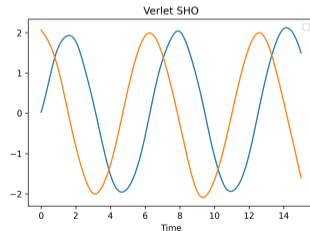
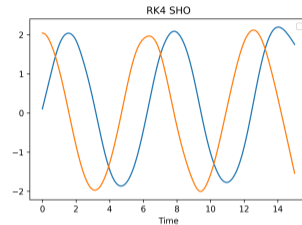
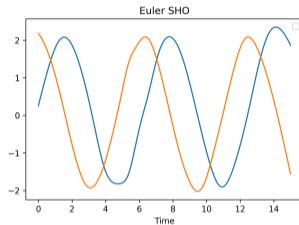
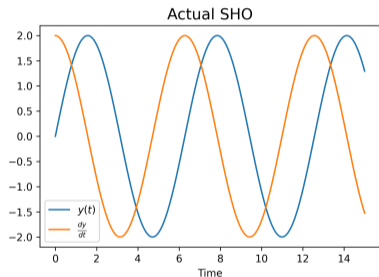
Figure: An object moving through a field of masses.

Method Comparison



As expected, Euler performs the worse than RK4, RK4 worse than Verlet. We have yet to adjust step sizes and will delve into more optimizations to improve the performance of our Verlet method, which we expect to be the most accurate [5].

Method Comparison



Euler Test Loss: 1.4824×10^{-2} , RK4 Test Loss: 9.5902×10^{-3} , Verlet Test Loss: 6.6160×10^{-4}

Summary and Next Steps

Progress:

- We've implemented 3 total Neural Networks: 2 Residual and 1 Hamiltonian Networks.
- We've identified the Verlet method for discretization as the most promising for our Hamiltonian Network, but clearly more optimizations and regularization are needed.

Next Steps:

- We will be working on having our neural nets learn the \mathcal{H} of our Hamiltonian, and also look at chaotic Hamiltonians.
- We will implement the other method, SINDy ([4], [7]), and use both methods to examine more Hamiltonian-based examples.

Questions?



U. M. Ascher.

Numerical methods for evolutionary differential equations, volume 5 of *Computational Science & Engineering*.

Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2008.



U. M. Ascher and C. Greif.

A First Course on Numerical Methods.




SIAM. SIAM, 2011.



J. R. Brannan.

Differential Equations: An Introduction to Modern Methods and Applications, volume 3.

John Wiley Sons, 2015.

-  B. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. Kutz, and S. Brunton.
Pysindy: A python package for the sparse identification of nonlinear dynamical systems from data.
Journal of Open Source Software, 5(49):2104, 2020.
-  E. Haber and L. Ruthotto.
Stable architectures for deep neural networks.
Inverse Problems, 34(1):014004, 22, 2018.
-  E. Hairer, C. Lubich, and G. Wanner.
Geometric numerical integration: Structure preserving algorithms for ordinary differential equations.
2004.

 A. A. Kaptanoglu, B. M. de Silva, U. Fasel, K. Kaheman, A. J. Goldschmidt, J. Callaham, C. B. Delahunt, Z. G. Nicolaou, K. Champion, J.-C. Loiseau, J. N. Kutz, and S. L. Brunton.

Pysindy: A comprehensive python package for robust sparse system identification.
Journal of Open Source Software, 7(69):3994, 2022.

 H. Traum.

Slopes used by the classical runge-kutta method.