Neural ODE Applications: Lagrangian and Hamiltonian Neural Networks

Lorenzo Liuzzo LORENZOLIUZZO@OUTLOOK.COM
Department of Physics, University of Studies
Milan, Italy
March 22, 2024

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1 Introduction

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2 Ordinary Differential Equations and Neural Networks

An ordinary differential equation (ODE) is an equation that describes the relationship between a function and its total derivatives. A neural network is a composition of L blocks, parameterized by a vector of parameters $\boldsymbol{\theta}$

$$\hat{h} : \mathbb{K}^M \to \mathbb{K}^N \quad \hat{h}(\mathbf{x}; \boldsymbol{\theta}) = h^{(L)} \circ \cdots \circ h^{(1)}(\mathbf{x}; \boldsymbol{\theta}) \quad \boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_L) \in \mathbb{K}^P$$

where each block is a function $h^{(l)} : \mathbb{K}^{M'} \to \mathbb{K}^{N'}$ parameterized by the component $\boldsymbol{\theta}_l \in \mathbb{K}^{P'}$.

2.1 Residual Neural Network

A residual neural network (RNN) uses building blocks of the form $h(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{x} + \mathbf{f}(\mathbf{x}; \boldsymbol{\theta})$, where $\mathbf{f}$ is some differentiable non-linear function of $\mathbf{x}$, parameterized by a vector $\boldsymbol{\theta}$, which preserves the input dimensionality. These blocks are then composed in a sequence of N

layers, as

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \Delta\mathbf{x}_t \qquad \Delta\mathbf{x}_t = \mathbf{f}_t(\mathbf{x}_t;\boldsymbol{\theta}_t) \qquad t = 0,\dots,N-1 \tag{1}$$

The function $\mathbf{f}_t$ is called the residual function of the t-th layer and it is often chosen to be the same for all layers. This process reseambles the discretization of the evolution of a dynamical system, where $\Delta\mathbf{x}_t$ is the increment of the state $\mathbf{x}_t$ at time t.
In particular, one could observe that the equation (**??**) is the first-order Euler's method for solving ordinary differential equations with a fixed step size $\Delta t = 1$. The idea behind Neural ODE network is to extend the residual network to a continuous dynamical system.

2.2 Neural ODE network

Consider a state $\mathbf{x} \in \mathbb{K}^M$ whose dynamic is defined by an initial value problem for a continuous function of the state and optionally some $t \in \mathbb{R}$, parameterized by some parameter vector $\boldsymbol{\theta} \in \mathbb{K}^P$, such as

$$\frac{d}{dt}\mathbf{x} = \mathbf{f}(\mathbf{x}, t; \boldsymbol{\theta}) \quad \text{with} \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{2}$$

The ODE-net transformation $\hat{h} : \mathbb{R} \to \mathbb{K}^M$ is given indirectly as the solution of the IVP:

$$\hat{h}(t; \mathbf{x}_0, \boldsymbol{\theta}) \equiv \mathbf{x}(t; \boldsymbol{\theta}) = \mathbf{x}_0 + \int_{t_0}^{t} d\tau \ \mathbf{f}(\mathbf{x}, \tau; \boldsymbol{\theta}) \tag{3}$$

A continuous transformation of the state would require a RNN to have an infinite number of layers, while a Neural ODE network has a single implicit layer, that employs a black-box solver to perform the integration. In a sense, the amount of steps it takes to solve the ODE could be thought as the depth of the network.
As it is presented by **?**, this black-box approach yields the possibility of choosing adaptive-step integrators, which leads to a trade-off between accuracy and computational cost. Perhaps, one can even train a Neural ODE network with high accuracy and adjust it to a lower accuracy at test time.
Another advantage of Neural ODE networks over residual networks is that they are continuous time-series models and thus can be trained on irregularly sampled data.
The model network architecture is also invertible and the inverse of the transformation h can be computed just by solving the ODE backwards in time. This is useful for tasks such as generative modeling, where the goal is to sample from a distribution over the input space, and normalizing flows, where the goal is to learn a distribution over the input space by transforming a simple base distribution.

2.3 Adjoint Sensitivity Method

To train a neural network, one needs to define a cost function and minimize it with respect to the network parameters $\boldsymbol{\theta}$. The cost function $\mathcal{C}$ for a neural ODE network can be defined as a functional acting on some loss function $l : \mathbb{K}^M \times \mathbb{R} \to \mathbb{R}$ over the whole state trajectory

$$\mathcal{C}(\mathbf{x}, t; \mathbf{x}_0, \boldsymbol{\theta}) \equiv \int_{t_0}^{t} d\tau \ l(\hat{h}_{\boldsymbol{\theta}}(\mathbf{x}_0, \tau), \tau) \tag{4}$$

It follows that the initial value problem in (??) can be formulated as an optimization problem with equality constraints for the function $\mathbf{f}$:

$$\mathbf{x}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{C}(\mathbf{x}, t; \mathbf{x}_0, \boldsymbol{\theta}) \quad \text{subject to} \quad \mathbf{g}(\mathbf{x}, t; \boldsymbol{\theta}) \equiv \mathbf{f}(\mathbf{x}, t) - \frac{d}{dt}\mathbf{x} = 0 \tag{5}$$

The problem is then addressed introducing the Lagrangian function $\mathcal{L}$ with a continuous multiplier $\boldsymbol{\lambda} \in \mathbb{K}^M$:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, t; \mathbf{x}_0, \boldsymbol{\theta}) = \mathcal{C}(\mathbf{x}, t; \mathbf{x}_0, \boldsymbol{\theta}) + \int_{t_0}^{t} d\tau \ \boldsymbol{\lambda}^T(\tau)\mathbf{g}(\mathbf{x}, \tau; \boldsymbol{\theta})$$

The sensitivity of $\mathcal{L}$ with respect to the network parameter $\boldsymbol{\theta}$ can be obtained as

$$\begin{aligned}
\frac{d\mathcal{L}}{d\boldsymbol{\theta}} &= \int_{t_0}^{t} d\tau \left[\frac{\partial l}{\partial \boldsymbol{\theta}} + \frac{\partial l}{\partial \mathbf{x}}\frac{d\mathbf{x}}{d\boldsymbol{\theta}} + \boldsymbol{\lambda}^T\left(\frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}} + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\frac{d\mathbf{x}}{d\boldsymbol{\theta}} - \frac{d}{d\tau}\frac{d\mathbf{x}}{d\boldsymbol{\theta}}\right)\right] \\
&= \int_{t_0}^{t} d\tau \left[\frac{\partial l}{\partial \boldsymbol{\theta}} + \boldsymbol{\lambda}^T\frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}} + \left(\frac{\partial l}{\partial \mathbf{x}} + \boldsymbol{\lambda}^T\frac{\partial \mathbf{f}}{\partial \mathbf{x}} - \boldsymbol{\lambda}^T\frac{d}{d\tau}\right)\frac{d\mathbf{x}}{d\boldsymbol{\theta}}\right]
\end{aligned}$$

In the context of conventional neural networks, the application of automatic differentiation facilitates the propagation over the network of the expression $\frac{d\mathbf{x}}{d\boldsymbol{\theta}}$, which represents how the output of the network depends on the parameters. However, in the case of a ODE-net a complexity arises from the usage of a black-box solver to determine the state, rendering it nontrivial and inefficient to backpropagate through.
Integrating by parts, the sensitivity of the Lagrangian $\mathcal{L}$ can be rewritten as

$$\frac{d\mathcal{L}}{d\boldsymbol{\theta}} = \int_{t_0}^{t} d\tau \left[\frac{\partial l}{\partial \boldsymbol{\theta}} + \boldsymbol{\lambda}^T\frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}} + \left(\frac{\partial l}{\partial \mathbf{x}} + \boldsymbol{\lambda}^T\frac{\partial \mathbf{f}}{\partial \mathbf{x}} + \frac{d}{d\tau}\boldsymbol{\lambda}^T\right)\frac{d\mathbf{x}}{d\boldsymbol{\theta}}\right] - \boldsymbol{\lambda}^T\frac{d\mathbf{x}}{d\boldsymbol{\theta}}\Big|_{t_0}^{t}$$

and, given the sensitivity of the initial state $\frac{d\mathbf{x}_0}{d\boldsymbol{\theta}}$, it is possible to write an equivalent system for $\frac{d\mathbf{x}}{d\boldsymbol{\theta}}$ as a terminal value problem for an adjoint state $\boldsymbol{\lambda}$:

$$\frac{d}{d\tau}\boldsymbol{\lambda}^T(\tau) = -\boldsymbol{\lambda}^T(\tau)\frac{\partial \mathbf{f}(\mathbf{x}, \tau)}{\partial \mathbf{x}} - \frac{\partial l(\mathbf{x}, \tau)}{\partial \mathbf{x}} \quad \text{with} \quad \boldsymbol{\lambda}^T(t) = \mathbf{0} \tag{6}$$

Furthermore, the sensitivity of the cost function $\mathcal{C}$ with respect to $\boldsymbol{\theta}$ is obtained from the Lagrangian sensitivity by integrating the adjoint system from the terminal condition $\boldsymbol{\lambda}^T(t) = \mathbf{0}$ backward to $\boldsymbol{\lambda}_0^T \equiv \boldsymbol{\lambda}^T(t_0)$:

$$\frac{d\mathcal{C}(\mathbf{x}; \boldsymbol{\theta})}{d\boldsymbol{\theta}} = \frac{d\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}; \boldsymbol{\theta})}{d\boldsymbol{\theta}} = \boldsymbol{\lambda}_0^T\frac{d\mathbf{x}_0}{d\boldsymbol{\theta}} - \int_{t}^{t_0} d\tau \left(\frac{\partial l}{\partial \boldsymbol{\theta}} + \boldsymbol{\lambda}^T\frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}}\right) \tag{7}$$

This method, known as the adjoint sensitivity method, allows efficient calculations of the sensitivity without storing any intermediate states during the forward pass, making neural ODE networks trainable with a constant memory cost.

3 Lagrangian Neural Network

Since both residual and neural ODEs are based on the evolution of a state, it could be interesting to delve deeper into the connection between these two approaches and the formulation of the evolution of a physical system given by classical mechanics.

3.1 Variational principles

Variational principles aim to globally characterize the trajectory of an object in motion from an initial to a final state using some stationarity property with respect to a family of possible movements.

Definition 1 *(Lagrangian) The Lagrangian $\mathcal{L}$ is a function of the generalized coordinates $\mathbf{q}$, velocities $\dot{\mathbf{q}} \equiv \frac{d}{dt}\mathbf{q}$ and time t:*

$$\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$$

The union $\mathbf{z} = (\mathbf{q}, \dot{\mathbf{q}})$ form a state in the phase space.

Theorem 2 *(Principle of Stationary Action) Consider a Lagrangian system over a fixed time interval $[t_0, t_1]$ and the family of movements $\mathbf{q}(t)$ whose satisfy the boundary conditions $\mathbf{q}(t_0) = \mathbf{q}_0$, $\mathbf{q}(t_1) = \mathbf{q}_1$. The Hamiltonian action $\mathcal{S}$ is a functional defined as the integral of the Lagrangian $\mathcal{L}$ of the system over time:*

$$\mathcal{S}[\mathbf{q}] \equiv \int_{t_0}^{t_1} dt\ \mathcal{L}(\mathbf{q}(t), \dot{\mathbf{q}}(t), t)$$

Natural movements $\tilde{\mathbf{q}}$ are those for which the action has an extremum, such that

$$\delta S[\tilde{\mathbf{q}}] = 0$$

Theorem 3 *(Euler-Lagrange constraints) According to the principle of stationary action, for a natural movement $\mathbf{q}$ over a fixed time interval $[t_0, t_1]$, from $\mathbf{q}(t_0) = \mathbf{q}_0$ to $\mathbf{q}(t_1) = \mathbf{q}_1$, it follows that*

$$\delta S = \int_{t_0}^{t_1} dt \left(\frac{\partial \mathcal{L}}{\partial \mathbf{q}} \delta\mathbf{q} + \frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} \delta\dot{\mathbf{q}} \right) = \int_{t_0}^{t_1} dt \left(\frac{\partial \mathcal{L}}{\partial \mathbf{q}} - \frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} \right) \delta\mathbf{q} + \frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} \delta\mathbf{q} \Big|_{t_0}^{t_1} = 0$$

The boundary term vanishes to satisfy boundary conditions, whilst the integral vanishing for any variation $\delta\mathbf{q}$, as per Euler's theorem, implies that movements must satisfy the differential equations, known as Euler-Lagrange equations, given by

$$\frac{\partial \mathcal{L}}{\partial \mathbf{q}} - \frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} = 0$$

3.2 LNN architecture

What the Principle of Stationary Action is claiming is that, for a generic system, Nature herself optimizes some cost function of the system and, according to the definition of the cost function for a continuous system (**??**), the Hamiltonian Action $\mathcal{S}$ could be indeed thought as the most natural cost function of a system, with the Lagrangian $\mathcal{L}$ as loss function. That is the basic definition of a Lagrangian Neural Network (LNN), a Neural ODE network build upon a parameterized Lagrangian $\mathcal{L}_{\boldsymbol{\theta}}$ that satisfies the Euler-Lagrange constraints.
In fact, Euler-Lagrange equations can be rewritten as a second order differential equation in the generalized coordinates $\mathbf{q}$ by expanding the total derivative using the chain rule:

$$\frac{d}{dt} \nabla_{\dot{\mathbf{q}}} \mathcal{L} = (\nabla_{\dot{\mathbf{q}}} \nabla_{\dot{\mathbf{q}}}^T \mathcal{L}) \ddot{\mathbf{q}} + (\nabla_{\mathbf{q}} \nabla_{\dot{\mathbf{q}}}^T \mathcal{L}) \dot{\mathbf{q}} = \nabla_{\mathbf{q}} \mathcal{L}$$

If the Hessian matrix $\nabla_{\dot{\mathbf{q}}}\nabla_{\dot{\mathbf{q}}}^T\mathcal{L}$ is invertible, a condition that always holds for natural Lagrangian systems, then the second order differential equation can be solved for $\ddot{\mathbf{q}}$.
It follows that, given an initial phase state $\mathbf{x}_0 \equiv (\mathbf{q}_0, \dot{\mathbf{q}}_0)$, the ODE defined in (**??**) for a Neural ODE network can be expanded for a Lagrangian Neural Network with a parameterized Lagrangian $\mathcal{L}_{\boldsymbol{\theta}}$ of the system as

$$\frac{d}{dt}\mathbf{x} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) = (\dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \left(\dot{\mathbf{q}}, (\nabla_{\dot{\mathbf{q}}}\nabla_{\dot{\mathbf{q}}}^T\mathcal{L}_{\boldsymbol{\theta}})^{-1}\left[\nabla_{\mathbf{q}}\mathcal{L}_{\boldsymbol{\theta}} - (\nabla_{\mathbf{q}}\nabla_{\dot{\mathbf{q}}}^T\mathcal{L}_{\boldsymbol{\theta}})\dot{\mathbf{q}}\right]\right) \quad \text{with} \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{8}$$

3.3 Training LNNs

3.4 Hamiltonian Neural ODE

4 Conclusion

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

References

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Jack Cai. Vectorized adjoint sensitivity method for graph convolutional neural ordinary differential equations, 2022.

Yang Cao, Shengtai Li, Linda Petzold, and Radu Serban. Adjoint sensitivity analysis for differential-algebraic equations: The adjoint dae system and its numerical solution. *SIAM Journal on Scientific Computing*, 24(3):1076–1089, 2003.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks, 2020.

Steven Johnson. Notes on adjoint methods for 18.336, 2007.

Hongzhou Lin and Stefanie Jegelka. Resnet with one-neuron hidden layers is a universal approximator, 2018.

Michael Lutter, Christian Ritter, and Jan Peters. Deep lagrangian networks: Using physics as model prior for deep learning, 2019.

Takashi Matsubara, Yuto Miyatake, and Takaharu Yaguchi. Symplectic adjoint method for exact gradient of neural ode with minimal memory. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 20772–20784. Curran Associates, Inc., 2021.

Han Zhang, Xi Gao, Jacob Unterman, and Tom Arodz. Approximation capabilities of neural odes and invertible residual networks, 2020.