Learning Efficient and Robust Ordinary Differential Equations via Invertible Neural Networks

Weiming Zhi[1], Tin Lai[1], Lionel Ott[2], Edwin V. Bonilla[3], Fabio Ramos[1,4]

[1] School of Computer Science, the University of Sydney

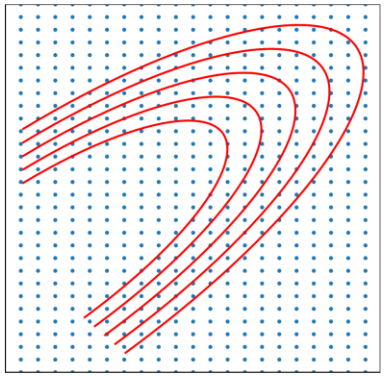
[2] Autonomous Systems Lab, ETH Zurich

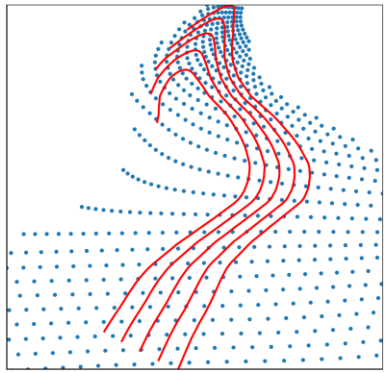
[3] Data61, CSIRO

[4] NVIDIA

Introduction

- Motivations of our work
  - Integrating long Neural ODE trajectories with numerical integrators can be slow and prone to error
  - There is no straightforward way to constrain the asymptotic stability of Neural ODEs
- We introduce an alternative approach to learning ODEs, by viewing the target ODE to be connected to a "simpler" ODE. Intuitively, we can use an invertible mapping to "morph" the simpler ODE into the desired target.





ODE Learning with Neural Network Dynamics

- Chen et al., 2018 introduced Neural ODEs, in which the adjoint method is used to enable tractable training of dynamics that are parameterised by a neural network. The integration is performed via a numerical integrator

$$\mathbf{y}'(t) = f(\mathbf{y}(t), t), \qquad \mathbf{y}(0) = \mathbf{y}_0,$$

Parameterised by NN

$$\begin{aligned}\mathbf{y}(t_e) =& \mathbf{y}_{t_0} + \int_{t_0}^{t_e} f_\omega(\mathbf{y}(t))\mathrm{d}t \\ =& \mathrm{ODESolve}(f_\omega, \mathbf{y}_{t_0}, t_e - t_0).\end{aligned}$$

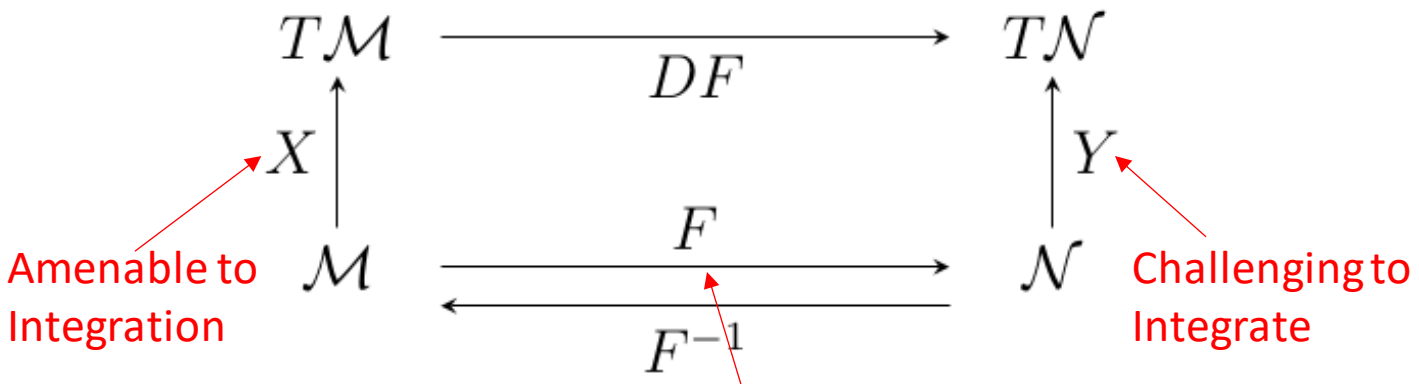
Solved with a numerical integrator

$$\ell(\omega) = \mathrm{Loss}(\{\mathbf{y}_{t_i}^{obs}\}_{i=1}^{n_t}, \{\mathbf{y}(t_i)\}_{i=1}^{n_t})$$

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, David K. Duvenaud. Neural Ordinary Differential Equations. Advances in Neural Information Processing Systems. 2018.

ODE Learning as Related Vector Fields

- We view the target ODE as a vector field that is *related* to an alternative vector field that is more amenable to integration. A coupling-based Invertible Neural Network is used to learn the mapping between integrals of the two vector fields



$$\mathcal{N} \xrightarrow{F^{-1}} \mathcal{M} \xrightarrow{X} T\mathcal{M} \xrightarrow{DF} T\mathcal{N}$$

Algorithm 1 Efficient integration of learned ODEs

Input: $F_\theta, g_\varphi, \mathbf{y}_0, t_1, \ldots, t_{end}$
Output: $\mathbf{y}(t_1), \ldots, \mathbf{y}(t_{end})$
$\mathbf{x}_0 \leftarrow F_\theta^{-1}(\mathbf{y}_0)$
$\mathbf{x}(t_i) \leftarrow \mathbf{x}_0 + \int_0^{t_i} g_\varphi(\mathbf{x}(t))\mathrm{d}t$, for $i = 1, \ldots, end$
The integral is easier to solve.
$\mathbf{y}(t_1), \ldots, \mathbf{y}(t_{end}) \leftarrow F_\theta(\mathbf{x}(t_1), \ldots, \mathbf{x}(t_{end}))$
Batched pass through INN can be efficiently computed on GPUs.

$$\mathbf{y}'(t) = J_{F_\theta}(F_\theta^{-1}(\mathbf{y}(t)))g_\varphi(F_\theta^{-1}(\mathbf{y}(t)))$$

$$\mathbf{y}(t) = F_\theta(F_\theta^{-1}(\mathbf{y}_0) + \int_0^t g_\varphi(\mathbf{x}(t))\mathrm{d}t)$$

"Simpler " base ODEs

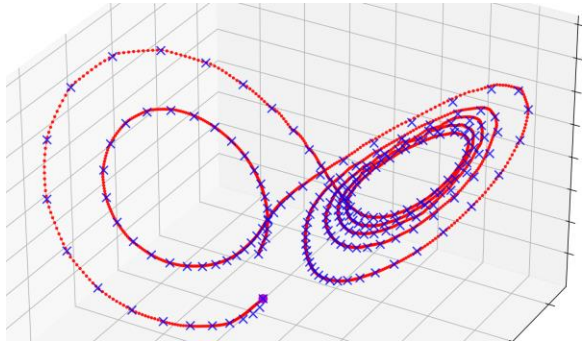
- Linear ODE as the base
  - We can achieve speed-ups of over two orders of magnitude for reasonably length trajectories, by using the matrix exponential method to solve the ODE

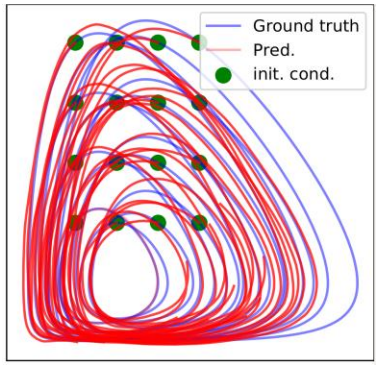
$$\mathbf{x}(t) = \sum_{k=1}^{n} (\mathbf{l}_k \cdot \mathbf{x}_0)\mathbf{r}_k \exp(\lambda_k t)$$

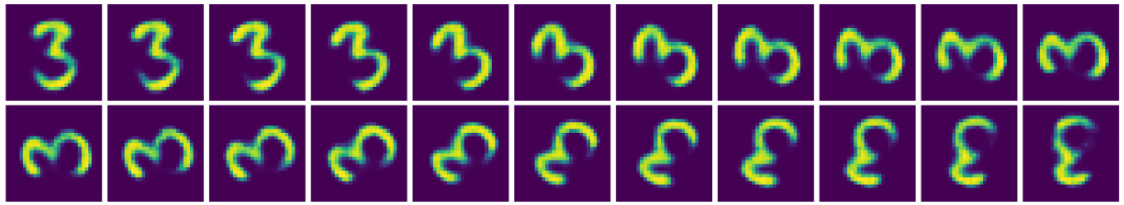
  - We can also ensure that the learned ODE will be asymptotically stable, by constraining the eigenvalues of the base ODE to be negative
  - However, the speed-up comes at a cost of flexibility
- Neural Network dynamics as the base
  - We can offload learning capacity to both the invertible mapping and base ODE, for added flexibility
  - The base ODE can often be smaller than if we directly parameterise the dynamics of the target ODE with a neural network. Note that the invertible neural network can be queried in parallel on a GPU, while a numerical integrator is largely sequential

Experimental Results

- When a linear base ODE is used, we can significantly speed up the integration of trajectories for a wide range of tested problems
- When a neural network base ODE is used, we observe improvements in learning difficult ODEs, while being even slightly faster







	3D Lotka-Volterra			Imitation S		Imitation cube pick		Imitation C	
	MSE (I)	MSE (G)	Time (ms)	MSE (G)	Time (ms)	MSE (G)	Time (ms)	MSE (G)	Time (ms)
Ours (Lin)	0.14± 0.1	1.5± 0.1	9.3± 0.4	6.1±1.2	6.6± 0.2	18.6± 6.2	7.1 ± 1.6	8.1± 1.6	7.5± 0.8
Euler	4.5± 0.3	4.6± 0.1	385.6± 14.4	10.3±2.9	724.7± 8.3	14.9± 1.4	728.4± 9.5	7.3± 2.0	753.9± 1.4
Midpoint	0.38± 0.05	5.51± 0.1	670.4± 31.3	10.9±3.3	581.6± 13.3	12.9± 1.3	1267.2± 13.6	6.9± 2.2	1305.4± 14.7
RK4	0.35± 0.005	5.6± 0.2	1316.1± 30.8	10.3±3.0	2501.7± 18.9	15.9± 0.9	2522.8± 23.1	7.6± 2.7	1292.3± 22.0
DOPRI5	0.93± 0.05	5.19± 0.5	264.7± 17.0	10.8±2.8	1277.7± 14.3	14.9± 0.9	504.0± 12.3	7.1± 1.9	623.4± 15.6

	Lorenz		ROBER	
	MAE	Time(ms)	MAE	Time(ms)
Ours	0.20	230±9	0.01	157±8
Euler	10.99	456± 13	0.22	201±6
Midpoint	6.60	805± 43	0.044	340±11
RK4	6.81	1761± 206	0.041	660±18
DOPRI5	7.55	632± 83	0.039	189±7

Conclusion

We present an alternative approach to learning ODEs: instead of directly parameterising the system dynamics by a neural network, we view the target ODE as related to a simpler base ODE. We can integrate trajectories of the base ODE and find the corresponding trajectories on the target ODE. We study both using a linear base, and a neural network base -- this allows us to obtain significant speed-ups or added flexibility.