Experimental study of Neural ODE training with adaptive solver for dynamical systems modeling

Hannah Plath, Thiago Petrilli Maffei Dardis, Alexandre Allauzen

ESPCI Paris - PSL

MILES team – Université Paris Dauphine

December 1st, 2022

Outlines

Introduction

From ResNet to Neural ODE

A deep Network is a cascade of K transformations/layers , for the kth layer

$$h_k = \underbrace{h_{k-1} \;+}_{\text{Residual connexion}} \;\underbrace{f_{\theta_k}(h_{k-1})}_{\text{Neural-Net Layer}}$$

- The residual connexion allows the model to be very deep
- Help for the vanishing gradient issue

If $K \longrightarrow \infty$

$$\frac{dh(t)}{dt} = f(h(t), t, \theta), \text{ the NNet is now a model of the dynamics}$$

Neural ODEs

- We can parameterize the continuous dynamics of hidden states in Neural Networks with an ODE:

$$\frac{dh(t)}{dt} = f(h(t), t, \boldsymbol{\theta}) \tag{1}$$

- Starting from the input layer $h(0)$, we can define the output layer $h(T)$ to be the solution to this ODE initial value problem at some time T.

- From the hidden state $z(t_0)$ we can define the loss function at $z(t_1)$ as:

$$L(z(t_1)) = L(z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta))dt \tag{2}$$

Neural ODEs

- We can parameterize the continuous dynamics of hidden states in Neural Networks with an ODE:

$$\frac{dh(t)}{dt} = f(h(t), t, \boldsymbol{\theta}) \tag{1}$$

- Starting from the input layer $h(0)$, we can define the output layer $h(T)$ to be the solution to this ODE initial value problem at some time T.

- From the hidden state $z(t_0)$ we can define the loss function at $z(t_1)$ as:

$$L(z(t_1)) = L(z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta))dt \tag{2}$$

Neural ODEs

- We can parameterize the continuous dynamics of hidden states in Neural Networks with an ODE:

$$\frac{dh(t)}{dt} = f(h(t), t, \boldsymbol{\theta}) \tag{1}$$

- Starting from the input layer $h(0)$, we can define the output layer $h(T)$ to be the solution to this ODE initial value problem at some time T.

- From the hidden state $z(t_0)$ we can define the loss function at $z(t_1)$ as:

$$L(z(t_1)) = L(z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \boldsymbol{\theta}))dt \tag{2}$$

Lorenz'63

- This butterfly attractor is broadly used as a benchmark for time series modeling
- Consider a point $x \in \mathcal{R}^3$ with its three coordinates x_1, x_2, x_3. The Lorenz'63 system consists of three coupled nonlinear ODEs:

$$
\begin{aligned}
\dot{x_1} &= \frac{dx_1}{dt} = \sigma(x_2 - x_1), \\
\dot{x_2} &= x_1(\rho - x_3) - x_1, \\
\dot{x_3} &= x_1 x_2 - \beta x_3
\end{aligned}
\tag{3}
$$

- In this work we consider the standard setting ($\beta = 8/3$, $\sigma = 10$, $\rho = 28$), such that the solution exhibits a *chaotic* regime.

Lorenz'63

- This butterfly attractor is broadly used as a benchmark for time series modeling
- Consider a point $x \in \mathcal{R}^3$ with its three coordinates x_1, x_2, x_3. The Lorenz'63 system consists of three coupled nonlinear ODEs:

$$\begin{aligned} \dot{x_1} &= \frac{dx_1}{dt} = \sigma(x_2 - x_1), \\ \dot{x_2} &= x_1(\rho - x_3) - x_1, \\ \dot{x_3} &= x_1 x_2 - \beta x_3 \end{aligned} \tag{3}$$

- In this work we consider the standard setting ($\beta = 8/3$, $\sigma = 10$, $\rho = 28$), such that the solution exhibits a *chaotic* regime.

Lorenz'63

- This butterfly attractor is broadly used as a benchmark for time series modeling
- Consider a point $x \in \mathcal{R}^3$ with its three coordinates x_1, x_2, x_3. The Lorenz'63 system consists of three coupled nonlinear ODEs:

$$
\begin{aligned}
\dot{x_1} &= \frac{dx_1}{dt} = \sigma(x_2 - x_1), \\
\dot{x_2} &= x_1(\rho - x_3) - x_1, \\
\dot{x_3} &= x_1 x_2 - \beta x_3
\end{aligned}
\tag{3}
$$

- In this work we consider the standard setting ($\beta = 8/3$, $\sigma = 10$, $\rho = 28$), such that the solution exhibits a *chaotic* regime.

Fehlberg's Method

Evaluations of f_θ

- The method requires three evaluations of f_θ for a fixed step-size h:

$$f_1 = f_\theta(x_i),\ f_2 = f_\theta(x_i + hf_i),\ f_3 = f_\theta(x_i + \frac{h}{4}[f_1 + f_2]) \tag{4}$$

- Then, we can compute two approximations for the next point:

$$A_1 = x_i + \frac{h}{2}[f_1 + f_2]\ (RK2\ method), and A_2 = x_i + \frac{h}{6}[f_1 + f_2 + 4f_3]\ (RK3) \tag{5}$$

- Then, we can estimate the following error:

$$r = \frac{|A_1 - A_2|}{h} \simeq Kh^2 \tag{6}$$

Evaluations of f_θ

- The method requires three evaluations of f_θ for a fixed step-size h:

$$f_1 = f_\theta(x_i),\ f_2 = f_\theta(x_i + hf_i),\ f_3 = f_\theta(x_i + \frac{h}{4}[f_1 + f_2]) \tag{4}$$

- Then, we can compute two approximations for the next point:

$$A_1 = x_i + \frac{h}{2}[f_1 + f_2]\ (RK2\ method), and A_2 = x_i + \frac{h}{6}[f_1 + f_2 + 4f_3]\ (RK3) \tag{5}$$

- Then, we can estimate the following error:

$$r = \frac{|A_1 - A_2|}{h} \simeq Kh^2 \tag{6}$$

Evaluations of f_θ

- The method requires three evaluations of f_θ for a fixed step-size h:

$$f_1 = f_\theta(x_i),\ f_2 = f_\theta(x_i + hf_i),\ f_3 = f_\theta(x_i + \frac{h}{4}[f_1 + f_2]) \tag{4}$$

- Then, we can compute two approximations for the next point:

$$A_1 = x_i + \frac{h}{2}[f_1 + f_2]\ (RK2\ method), and A_2 = x_i + \frac{h}{6}[f_1 + f_2 + 4f_3]\ (RK3) \tag{5}$$

- Then, we can estimate the following error:

$$r = \frac{|A_1 - A_2|}{h} \simeq Kh^2 \tag{6}$$

Evaluations of f_θ

- If $r > \epsilon$, then A_2 is rejected and we need to restart the computation with

$$h' = S \times h\sqrt{\epsilon/r} \tag{7}$$

- We've used the following default values: h=1, S=0.9 and ϵ=0.1.

Evaluations of f_θ

- If $r > \epsilon$, then A_2 is rejected and we need to restart the computation with

$$h' = S \times h\sqrt{\epsilon / r} \tag{7}$$

- We've used the following default values: h=1, S=0.9 and ϵ=0.1.

Experiments

Trajectories prediction

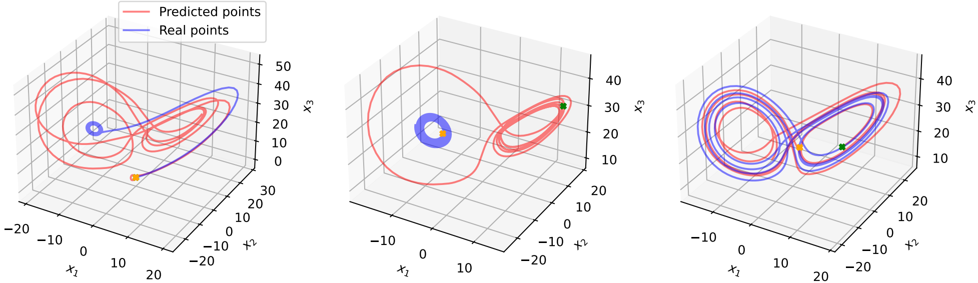

Figure: Each figure depicts a different time slice of the generated trajectory and of the original training data: from 0 to 600, 600 to 1200 and 2000 to 2600.

Performance

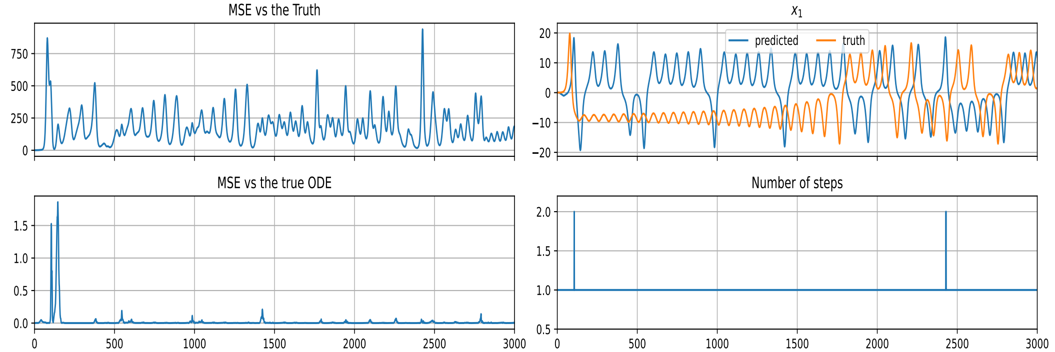

Figure: Time evolution of (from left to right and top to bottom): the MSE, the evolution of x_1, the MSE w.r.t the true ODE of Lorenz'63, and the number of steps.

Fehlberg's Training

Results

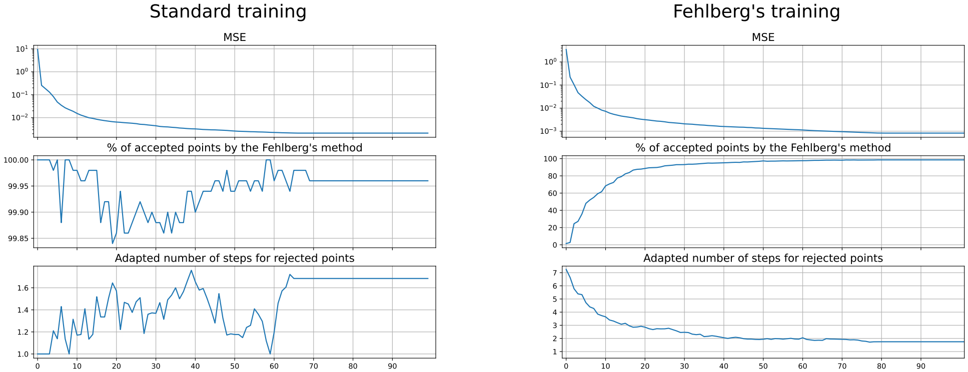

Figure: Time evolution for two training conditions of: the MSE loss; the percentage of accepted hypotheses A_2; the new number of steps for the rejected hypotheses (before rounding).

Conclusion

Take-home message: watch your step size !

- Neural **ODE** relies on solvers for inference and training.
- Adaptive solvers cannot be seamlessly leveraged as a black-box:

Our contributions:

→ For most of the numerical methods, the step size is adapted given the self-estimated error of the model.

→ Neural ODE model is always self-confident, and the "adaptive" ability is under-used.

→ A simple workaround:

Use the supervision within the solver.

Thank you for your attention!