Interpolating coarse-grained data of physical systems with Neural ODE

Junyi Guo, Rajit Rajpal

[1]UC Berkeley

Introduction

We would often like to learn the dynamics of physical systems but presented with data that is discrete (coarse-grained) in real-world applications. There exist many methods to interpolate between them like cubic spline interpolation. We thus propose using an autoencoder-decoder architecture and have a Euler or Runge-Kutta-4 solver in the hidden layer.

Problem Setup

Let's say we have w_0 as input, then we would like to predict w_5. In the latent space, we use ode step as 5. The learning process should be: $w_0, z_0, z_1, z_2, z_3, z_4, z_5, w_5$. The $rk4$ in latent space is given by:

$$z_{k+1} = z_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where $h = \frac{\Delta t_5}{odesteps}$

When you do the evaluation, let's say you want to predict w_2. Then keep all other parameters same but change your Δt_5 to Δt_5 (which is corresponding to your trajectory time). Inside the latent space, you still have 5 ode steps. But now your inferred result should be . And your new h^* should be

$$h^* = \frac{\Delta t_2}{odesteps}$$

The learning process will be: $w_0, z_0, z_{0.4}, z_{0.8}, ...z_2, w_2$ Then for $\hat{w_3}$.

We try to solve the following question: Is state function $F(z_k; ode)$ time invariant?

Architecture

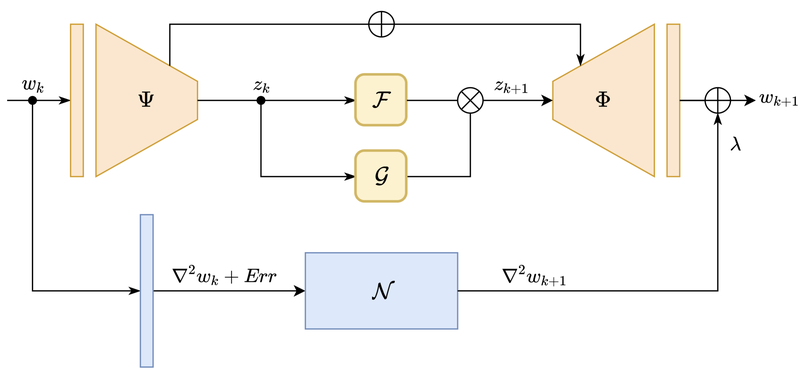

Figure 1. Sketch of the architecture

Method

There are 4 major components in the proposed architecture: auto-encoder denoted by Ψ and Φ, neural ODE block F, the gating block G and the physics inspired block N. The skipped connections are made between encoder and decoder to enable stable training. Gating functions G are used to deal with multi-scale effect.

Euler's Method

Note that in the theory of ODE and numerical methods, the state function F have to be differentiable. Therefore the choice of activation function should be differentiable such as tanh. However, as reported in literature, ReLU also works fine.

In our case, we first map the initial condition wk from RN×N to the latent space $\mathbb{R}^{M»M}$ and obtain z_k, where $zk = \Psi(w_k; \theta_E)$. We pass z_k to the state function F obtain:

$$\frac{dz(t)}{dt} = F(z(t), t, \theta_{ODE})$$

To solve this ode, we seek numerical integration methods.

$$z_{k+1} = z_k + \int_{t_k}^{t_{k+1}} F(z(s), s, \theta_{ODE}) ds$$

For one step forward Euler: $z_{k+1} = z_k + \Delta h F(z_k, t_k; \theta_{ode})$, where $\Delta h = t_{k+1} - t_k$

For N steps forward Euler:

$$z_{\frac{k+1}{N}} = z_k + \frac{\Delta h}{N} F(z_k, t_k; \theta_{ode})$$

...

$$z_{k+1} = z_{k+\frac{N-1}{N}} + \frac{\Delta h}{N} F(z_{k+\frac{N-1}{N}}, t_k + \Delta h \frac{N-1}{N}; \theta_{ode})$$

We omit the RK-4 and backward Euler procedure for space.

Time as an extra channel

In our example, the latent space data z has shape (Batch, Channel, Height, Width) = (32, 128, 8, 8). To match the input dimension requirement for nn.Linear(), the input shape to ode function should be (B,H*W,C). Consider a simplified case: batch size $B = 3$. Given the initial conditions $w(t = 0), w(t = 100), w(t = 200)$, our goal is to predict $w(t = 5), w(t = 105), w(t = 205)$. Note that t here represents global time which corresponds to the time in ground truth data. As we map the batch of initial conditions to the latent space, we obtain the corresponding $z(t = 0), z(t = 100), z(t = 200)$. Where $\Psi(x; \theta) = z$. We can plug this into our ODE function F and use the integration scheme defined earlier.

Experiment Setup

We apply our method on the vorticity equation in 2D which follows the following governing equations:

$$\delta_t w(x,t) + u.\nabla w(x,t) = v\nabla^2 w(x,t)$$

$$\delta_t w_1 = v\nabla^2 w = N(w_k, t; \theta_{phy})$$

$$\delta_t w_2 = u\nabla w = N(w_k, t; \theta_{phy} = F(z_k, t; \theta_{ODE})$$

The physics term of interest $\nabla^2 w_k$ is extracted by gradient-free convolutional filters. The weights in kernel K are determined by central difference scheme where the numerical error is reduced by a following neural network N.

$$\nabla^2 w(x,y) = \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2}$$

$$\frac{\partial^2 w}{\partial x^2} = \frac{w(x+\Delta x, y) - 2w(x,y) + w(x-\Delta x, y)}{\Delta x^2}$$

$$\frac{\partial^2 w}{\partial y^2} = \frac{w(x, y+\Delta y) - 2w(x,y) + w(x, y-\Delta y)}{\Delta x^2}$$

$$K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Training: Given any paired snapshots at time k and $k+1$: (w_k, w_{k+1}), our model interpolates N snapshots in between (w_k, w_{k+1}) yielding $w_{\frac{k+1}{N}}, w_{\frac{k+2}{N}}, ..., w_{k+\frac{N-1}{N}}, w_{k+1}$. The training objective is then $min \|w_{k+1} - \hat{w}_{k+1}\|$.

Test: We measure the fit by calculating the relative error based on intermediate results.

$$err = \Sigma_{k=1}^{M} \frac{\|(\hat{w}_{\frac{k+1}{N}} - w_{\frac{k+1}{N}}) + ... + (\hat{w}_{\frac{k+1}{N}} - w_{k+\frac{N-1}{N}})\|}{\|w_{\frac{k+1}{N}} + ... + w_{k+\frac{N-1}{N}}\|}$$

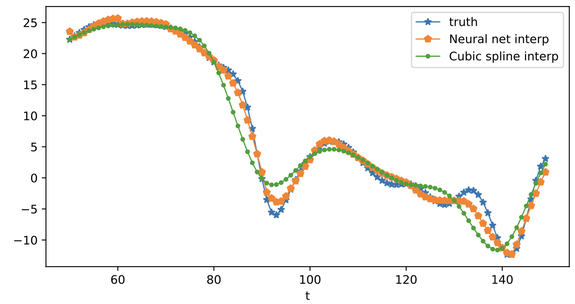

Figure 2. Neural network vs Cublic spline interpolation. The interpolation steps N = 10. Relative Error for traditional model is 0.23 while for ours it is 0.14