Sorbonne Université
Master DAC

Reinforcement Learning and Advanced Deep Learning

Neural ODE report

Tan Khiem HUYNH

17 février 2023

The Damped Pendulum Problem

The goal of this section is to use Neural ODE to learn the underlying dynamic of a damped pendulum. Let assume that this dynamic is given by the ODE :

$$\ddot{\theta}_t + \omega_0^2 \sin(\theta_t) + \alpha\dot{\theta}_t = 0 \tag{1}$$

Let $Y_t = (\theta_t, \dot{\theta}_t)$. Then from the equation above we can derive the ODE for Y :

$$\dot{Y}_t = f(Y_t) \tag{2}$$

with

$$\begin{aligned} f \colon \mathbb{R}^2 &\to \mathbb{R}^2 \\ (Y_0, Y_1) &\mapsto (Y_1, -\omega_0^2 \sin Y_0 - \alpha Y_1) \end{aligned}$$

The training data consist of 1000 data points generate by (2), in the time interval from $t = 0$ to $t = 25$, with the initial condition $Y_0 = (2, 0), \omega_0 = \frac{\pi}{6}$ and $\alpha = 0.2$. Each batch of training data consist a sequence of 10 continuous data points.

The test data also consist of sequences of data points generate by (2), but from different Y_0, which is generate randomly but with a constraint on the angular velocity to avoid big value that can cause the sequence diverge from 0.

Visualizing the training data and the test data :

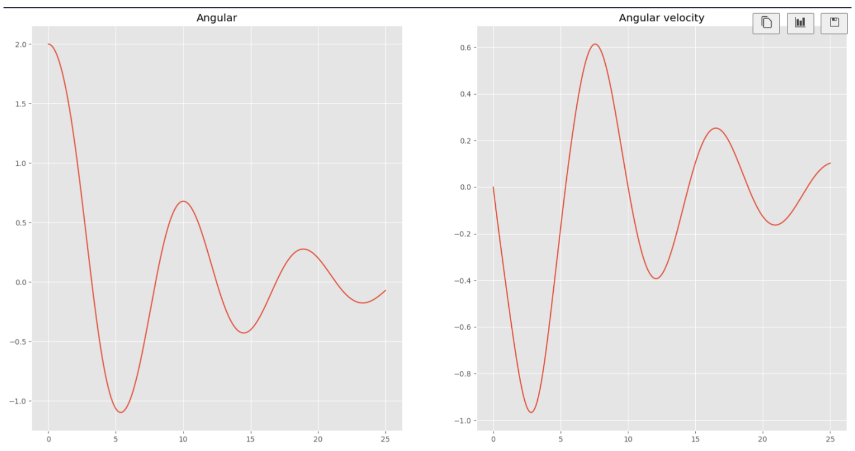

FIGURE 1 – Training data

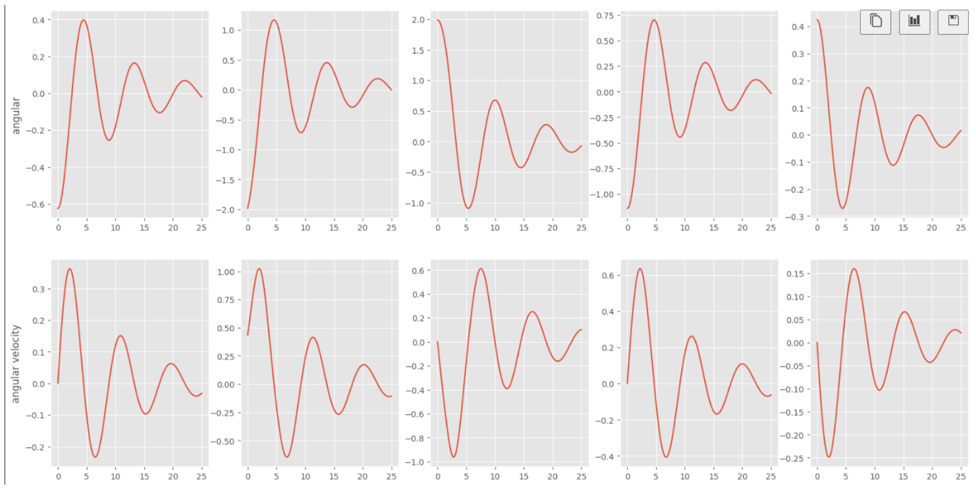

FIGURE 2 – Test data

The we use a simple MLP with 2 hidden layers to parameterize $f(Y)$. We use the RK4 solver from the library **torchdiffeq** to generate the predicted sequence from the parameterized ODE and then use the L1 loss to update the MLP. After 1000 epoch, we can generalize quite well to new random initial condition from the test dataset :

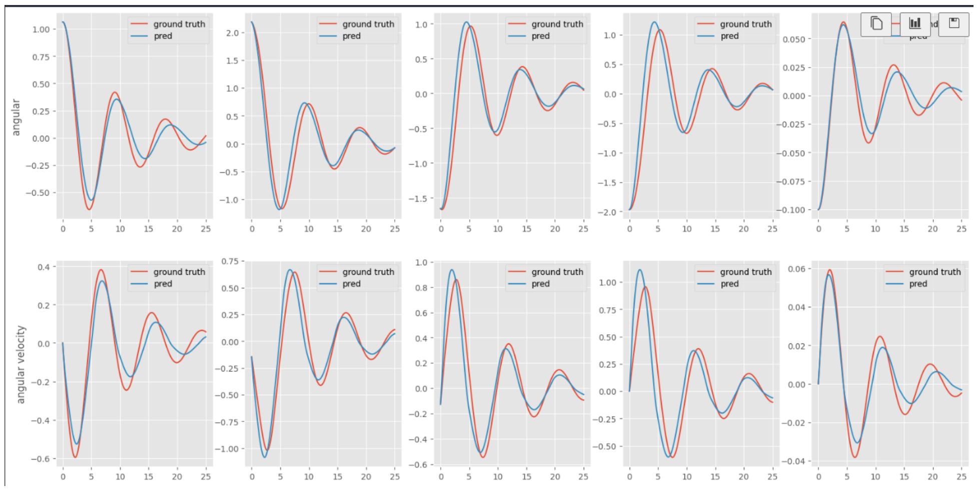

FIGURE 3 – Results on test dataset

Neural ODE as Residual Network for MNIST classification

In this section, we implemented Neural ODE as a substitution for the residuals network to solve the classification problem on the MNIST dataset. In detail, we experiment 3 approaches :

- A small residual network as a baseline
- Neural ODE with adjoint method
- Neural ODE without adjoint method

The input images in all 3 approaches are first down-sampled using the following block :

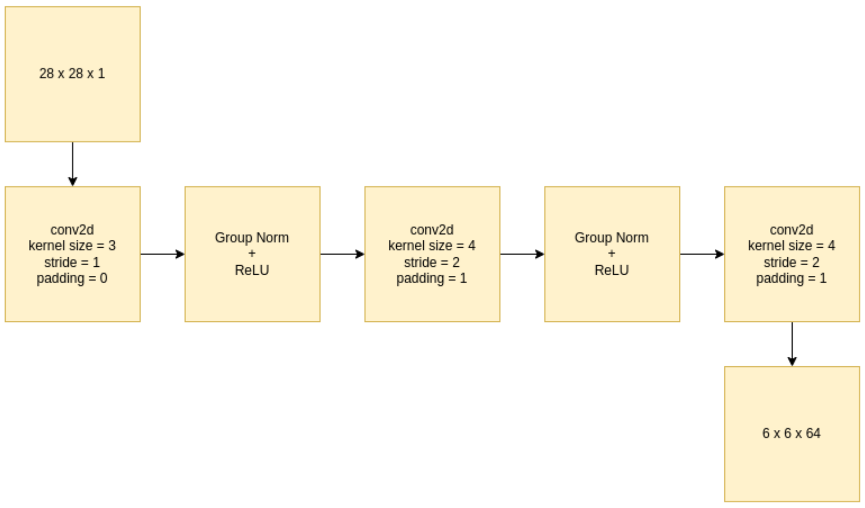
28 x 28 x 1
conv2d kernel size = 3 stride = 1 padding = 0
Group Norm + ReLU
conv2d kernel size = 4 stride = 2 padding = 1
Group Norm + ReLU
conv2d kernel size = 4 stride = 2 padding = 1
6 x 6 x 64

FIGURE 4 – Down-sampling block

Then for the residual network baseline, we use 6 residual block like below as a feature extractor :

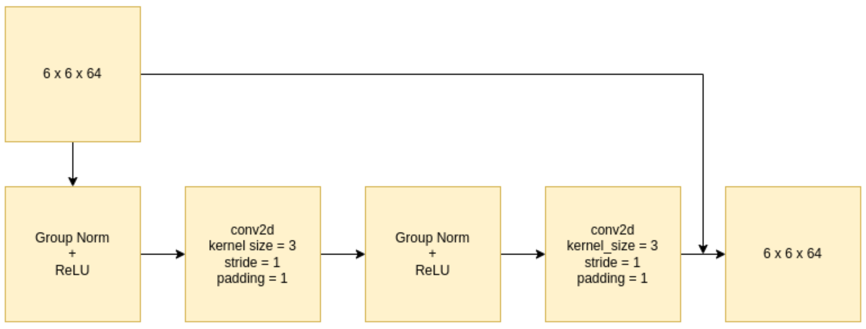
6 x 6 x 64
Group Norm + ReLU
conv2d kernel size = 3 stride = 1 padding = 1
Group Norm + ReLU
conv2d kernel_size = 3 stride = 1 padding = 1
6 x 6 x 64

FIGURE 5 – Residual block

For the Neural ODE variant, we use the following block to replace the residual block as a continuous parameterization of the dynamic of hidden units :

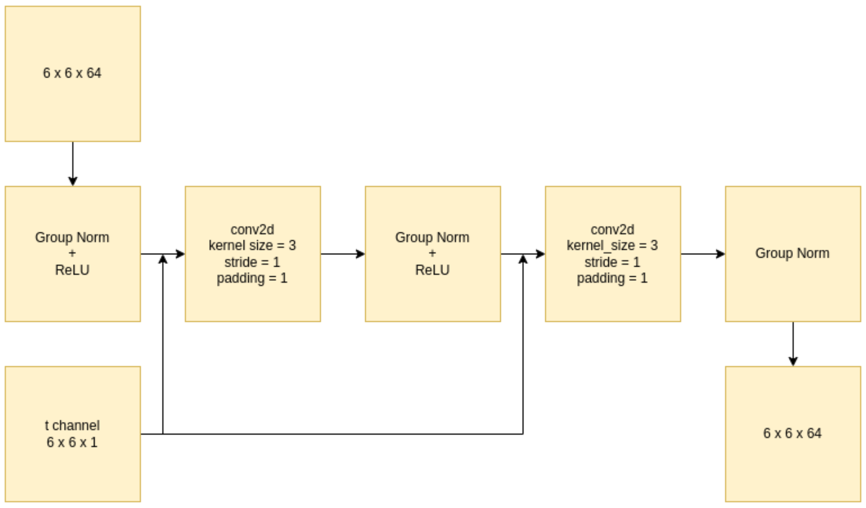
6 x 6 x 64
Group Norm + ReLU
conv2d kernel size = 3 stride = 1 padding = 1
Group Norm + ReLU
conv2d kernel_size = 3 stride = 1 padding = 1
Group Norm
t channel 6 x 6 x 1
6 x 6 x 64

FIGURE 6 – ODE block

Then starting from $t = 0$, a ODE solver is used to extract the final feature at $t = 1$ using the above block.

The straightforward method for calculating the gradient of the loss with respect to the parameters of the ODE block is that we store the intermediate hidden states of the solver in the forward pass, and then in the backward pass we differentiate directly through these states. However, this incurs a high memory cost and additional numerical errors, especially for complex solver.

The adjoint method allow us not to store the intermediate states in the forward pass yet we will still be able to calculate the gradient. In general, this method consider the hidden states, as well as the gradient of the loss with respect to the ODE parameters, as an "augmented dynamic". This augmented dynamic can also be modeled by another ODE, so we could use another solver that run backward in time to compute it.

Finally, a common fully connected block is used to produce the suitable output for the classification problem :

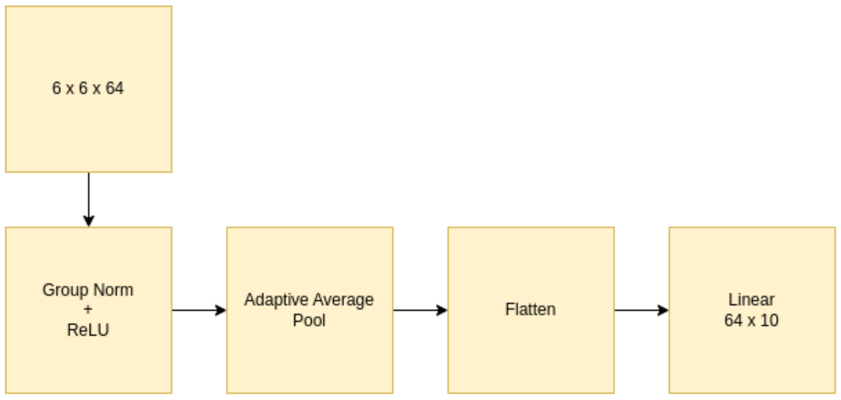

FIGURE 7 – FC block

We run the 3 approaches for 10 epochs on the MNIST dataset, with a batch size of 128, learning rate 0.1 and SGD optimizer. The results show that the Neural ODE variant can achieve around the same performance as the famous residual architecture :

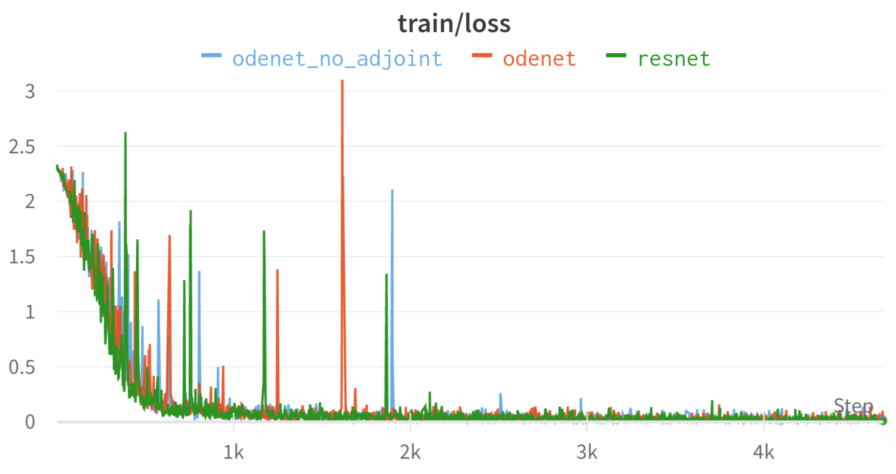

FIGURE 8 – Train loss

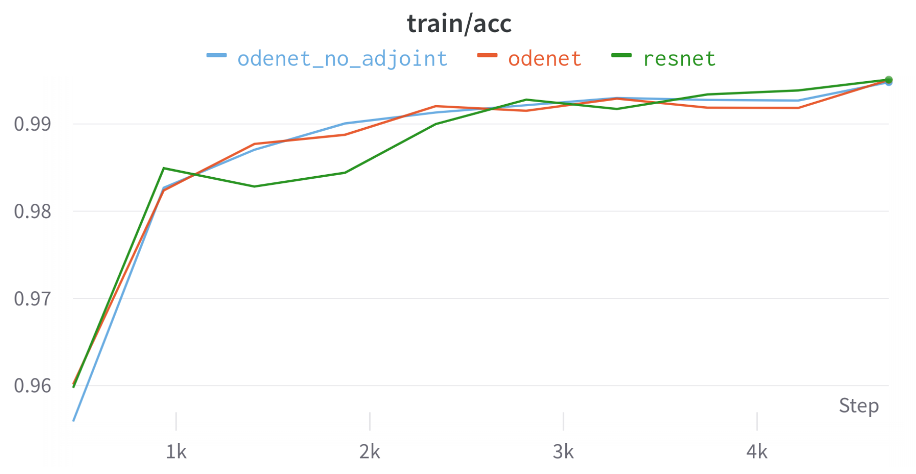

FIGURE 9 – Train accuracy

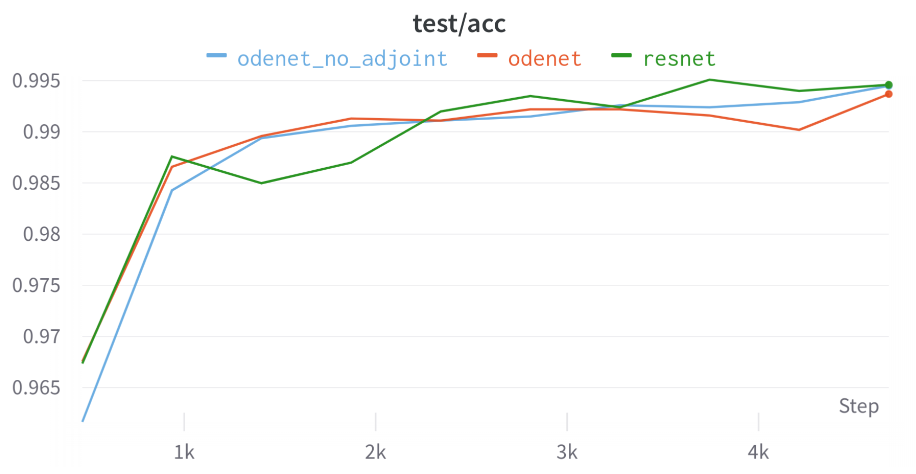

FIGURE 10 – Test accuracy

However, the running time of the NeuralODEs is quite slow in comparison with the residual net, especially with the adjoint method, which use an adaptive solver in the backward pass.

For this simple problem, the adjoint method show no clear advantage in term of memory complexity and performance over the straightforward method. However, with a more complicated integration scheme the memory will blow up if we don't use the adjoint method.

Furthermore, more clear benchmarks on more complicated dataset are still needed to evaluate the advantage of the NeuralODEs over the common residual architecture that is used widely nowadays.