NeuralODE studying

Ilya Lopatin
Optimization Class Project. MIPT

Introduction

The NeuralODE architecture of was created in paper [1], 2018. This project is devoted to studying of this neural network's architecture and carrying out numerical experiments with it.

The main idea of NeuralODE

We can see that forward propagation in residual neural network looks like Euler's method of numerical solving of ordinary differential equation (1). It turns out that calculating gradient is presented as a solution of the other (*adjoint*) ODE (2):

$$\begin{cases} \frac{dz(t)}{dt} = f(z(t), t, \theta), \\ t \in [t_1; t_2], \\ z(t_1) = x \end{cases}, \tag{1}$$

$$\begin{cases} \frac{da(t)}{dt} = -a^T(t)\frac{\partial f}{z(t)} \\ a(t_2) = \frac{\partial L}{\partial y} \end{cases}, \tag{2}$$

where $f(z(t), t, \theta)$ is representation of neural network, θ is set of trainable parameters and $L = L(y(x))$ is loss function. The euation (3) gives us possibility to calculate gradient $\nabla_\theta L$ and do the procedures of gradient descent.

$$\nabla_\theta L = \int_{t_1}^{t_2} a^T(t)\frac{\partial f}{\partial \theta}(z(t), t, \theta)\, dt. \tag{3}$$

Possible applications

- Replacing Res-block with NeuralODE block. It can be fruitful because we replace Euler's scheme related to Res-block with more powerfull methods likes Runge-Kutta methodes or another methodes of numerical solving ODE.
- We get new method of restoring the hidden dynamics function by given set of observations. Dynamics function is right side of equation $z'(t) = f(z(t), t)$ and for NeuralODE restoring f is standart training procedure.
- Let's have a look at task of modeling or sampling of unknown probability density function by given points. The most common methodes have $O(d^3)$ time asymptotics when d is dimension of space. The following statement turns out to be useful (see [1]):
  Let $z(t)$ be a finite continuous random variable with probability $p(z(t))$ dependent on time. Let $dz/dt = f(z(t), t)$ be a differential equation describing a continuous-in-time transformation of $z(t)$. Assuming that f is uniformly Lipschitz continuous in z and continuous in t, then the change in $\log$ probability also follows a differential equation (4) :

$$\frac{\partial \log p(z(t))}{\partial t} = -tr\left(\frac{\partial f}{\partial z(t)}\right) \tag{4}$$

  Note that calculating matrix trace is $O(d)$. Thanks to (4) we can interpret modeling or sampling of unknown probability density function as restoring the hidden dynamics function, see previous point.

Algorithm

Below is an algorithm from [1] for calculating all the necessary gradients

Algorithm 1 Reverse-mode derivative of an ODE initial value problem

Input: dynamics parameters θ, start time t_0, stop time t_1, final state $\mathbf{z}(t_1)$, loss gradient $\partial L/\partial \mathbf{z}(t_1)$
$s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}_{|\theta|}]$ ▷ Define initial augmented state
def aug_dynamics($[\mathbf{z}(t), \mathbf{a}(t), \cdot], t, \theta$): ▷ Define dynamics on augmented state
  return $[f(\mathbf{z}(t), t, \theta), -\mathbf{a}(t)^\intercal \frac{\partial f}{\partial \mathbf{z}}, -\mathbf{a}(t)^\intercal \frac{\partial f}{\partial \theta}]$ ▷ Compute vector-Jacobian products
$[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}] = \text{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$ ▷ Solve reverse-time ODE
return $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}$ ▷ Return gradients

when f_{aur} is augmented dynamic: $\frac{d[z,\theta,t]}{dt} = f_{aug}([z, \theta, t]) = [f([z, \theta, t]), 0, 1]$
If we have some set of given observations or points of hidden distribution density, the adjoint sensitivity method solves an augmented ODE backwards in time. The augmented system contains both the original state and the sensitivity of the loss with respect to the state. If the loss depends directly on the state at multiple observation times, the adjoint state must be updated in the direction of the partial derivative of the loss with respect to each observation, see figure 1.

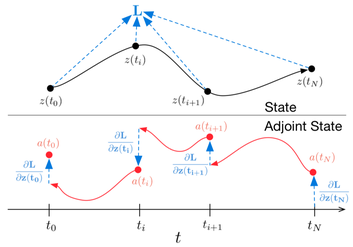


Figure 1

MNIST test

The first experiments were devoted to classic MNIST test of recognizing numbers. In figure 2 we can see result of this expiriment.

	Test Error	# Params	Memory	Time
1-Layer MLP†	1.60%	0.24 M	-	-
ResNet	0.41%	0.60 M	$\mathcal{O}(L)$	$\mathcal{O}(L)$
RK-Net	0.47%	0.22 M	$\mathcal{O}(\tilde{L})$	$\mathcal{O}(\tilde{L})$
ODE-Net	0.42%	0.22 M	$\boldsymbol{\mathcal{O}(1)}$	$\mathcal{O}(\tilde{L})$

Figure 2

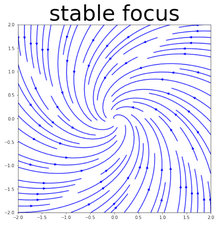
L denotes the number of layers in the ResNet, and $\tilde{L}$ is the number of function evaluations that the ODE solver requests in a single forward pass, which can be interpreted as an implicit number of layers.

Our experiments demonstrate that it takes 4 times longer for NeuralODE to achieve equivalent results with ResNet, but at the same time NeuralODE requires 3 times less trainable parameters and a constant quantity of memory. The runs were carried out under the same conditions in Google Coollaboratory with Google Compute Engine (GPU) support.

Restoring the hidden dynamics function

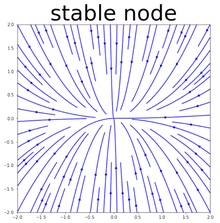
We've carried out experiments in case when hidden dynamics is constant matrix, that is , the equation of dynamics is $z'(t) = Az$

stable focus



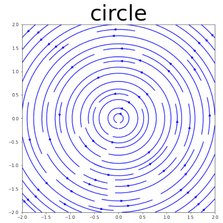
$$A = \begin{bmatrix} -1 & -1 \\ 1 & -1 \end{bmatrix}$$

stable node



$$A = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}$$

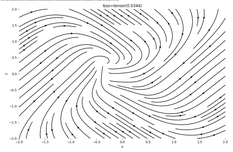
circle



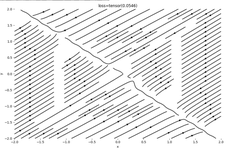
$$A = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Below is the process of learning Neural CODE on a set of random points for each of the three presented systems.

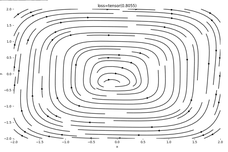
Stable Focus



Stable Node



Circle



Note that during the experiments, the following problem was found. If the matrix A has an eigenvalue with a positive real part, which means that the system is unstable, then the program crashes.

Author's comment: *This underflow error indicates that the system is too stiff for an explicit method to solve. The step size needs to be extremely small in order to satisfy the desired tolerance. Ricky Chen*

Conclusion

During our first experiments, we partially tested some of the advantages and possible applications of Neural CODE indicated in the first two paragraphs of the section "Possible applications". We got positive results on the MNIST test and in the task of restoring hidden dynamics in the case of a stable system.

Acknowledgements

This material is based on paper [1]. The source code of the neural network was taken from [2]. I would like to express my special gratitude to Danya Merkulov for him initiative to make this project, references and advices.

References

[1] Ricky T., Q. Chen, Yulia Rubanova, Jesse Bettencourt. *Neural Ordinary Differential Equations*. University of Toronto, Vector Institute, 2018.

[2] https://github.com/rtqichen/torchdiffeq

[3] https://inlnk.ru/Jj2l88 my github repository with theexperiments