Neural Ordinary Differential Equations

Pramook Khungurn

April 24, 2022

This is a note on the paper "Neural Ordinary Differential Equations" by Chen et al.[CRBD18].

1 Introduction

- Many existing neural networks models creates a sequence of hidden states $\mathbf{h}_0$, $\mathbf{h}_1$, $\mathbf{h}_2$, ... $\mathbf{h}_T$ by adding something to the previous state:

  $$\mathbf{h}_{t+1} = \mathbf{h}_t + \mathbf{f}(\mathbf{h}_t, t, \boldsymbol{\theta})$$

  Such models include such as residual networks [HZRS15], recurrent neural networks, and normalizing flows [RM15, DKB14].

- What if we take the limit as the number of time step goes to infinity? We will have a differential equation:

  $$\frac{\mathrm{d}\mathbf{h}(t))}{\mathrm{d}t} = \mathbf{f}(\mathbf{h}(t), t, \boldsymbol{\theta}).$$

- To use the network, we simply say that $\mathbf{h}(0)$ is the input layer, and the output is $\mathbf{h}(T)$ at some time T. The output can be found by solving the initial value problem, and this can be done by any black-box differential equation solver.

2 How to train a neural ODE model

- The problem with the above approach is that it is unclear how to train such a neural ODE model.
  - The computation of the solution can require a lot of time steps. Differentiating through these time steps to compute the gradient would requires saving a lot of information in memory.
- The good news is that there is a method to compute the gradient using constant memory (i.e., does not depend on the number of time steps). This is called the **adjoint sensitivity method**. It requires, however, an ODE solve, which can be done, again, by any ODE solver.
- To derive the method, we first change the notations.
  - The hidden state at time t is now denoted by $\mathbf{z}(t)$.
  - The start and end time is now t_0 and t_1 instead of 0 and T, respectively.
  - With this notation, we have that, for any $t \geq t_0$,

    $$\mathbf{z}(t) = \mathbf{z}(t_0) + \int_{t_0}^{t} \mathbf{f}(\mathbf{z}(u), u, \boldsymbol{\theta})\,\mathrm{d}u.$$

In this case, we view $\mathbf{z}(t)$ as a result of simulating the ODE forward in time from t_0 to t. It is thus probably better to write

$$\mathbf{z}(t) = \mathbf{z}_0 + \int_{t_0}^{t} \mathbf{f}(\mathbf{z}(u), u, \boldsymbol{\theta})\, \mathrm{d}u$$

to emphasize the fact that $\mathbf{z}_0 := \mathbf{z}(t_0)$ is a part of the initial condition.

- We also have that

$$\mathbf{z}(t) = \mathbf{z}(t_1) + \int_{t_1}^{t_0} \mathbf{f}(\mathbf{z}(u), u, t)\, \mathrm{d}u.$$

Here, we view $\mathbf{z}(t)$ as a resulf simulating the ODE *backward* in time from t_1 to t. This view will become important later on. Again, when taking this view, it is better to write

$$\mathbf{z}(t) = \mathbf{z}_1 + \int_{t_1}^{t_0} \mathbf{f}(\mathbf{z}(u), u, t)\, \mathrm{d}u.$$

instead.

- The goodness of the output $\mathbf{z}(t_1)$ is computed by a loss function $L(\cdot)$ as $L(\mathbf{z}(t_1))$.
- Because $\mathbf{z}(t_1)$ is a function of $\mathbf{z}(t_0)$, t_0, and $\boldsymbol{\theta}$, we may define a function

$$\mathcal{L} : (\mathbf{z}(t_0), t_0, \boldsymbol{\theta}) \mapsto L(\mathbf{z}(t_1))$$

that maps the initial condition to the loss.

- Now, for any $t \leq t_1$, it makes sense to talk about

$$\mathcal{L}(\mathbf{z}, t, \boldsymbol{\theta}).$$

This expression gives the loss if we simulate the ODE starting with the hidden state $\mathbf{z}$ at time t until time t_1.

- It also makes sense to talk about three gradient functions

$$\nabla_1 \mathcal{L}(\mathbf{z}, t, \boldsymbol{\theta}) = \nabla_{\mathbf{z}} \mathcal{L}(\mathbf{z}, t, \boldsymbol{\theta}) = \left.\frac{\partial \mathcal{L}}{\partial \mathbf{z}}\right|_{(\mathbf{z}, t, \boldsymbol{\theta})},$$
$$\nabla_2 \mathcal{L}(\mathbf{z}, t, \boldsymbol{\theta}) = \nabla_t \mathcal{L}(\mathbf{z}, t, \boldsymbol{\theta}) = \left.\frac{\partial \mathcal{L}}{\partial t}\right|_{(\mathbf{z}, t, \boldsymbol{\theta})},$$
$$\nabla_3 \mathcal{L}(\mathbf{z}, t, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{z}, t, \boldsymbol{\theta}) = \left.\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\right|_{(\mathbf{z}, t, \boldsymbol{\theta})}.$$

These functions output linear transformations that can turn a perturbation on their associated variables into perturbation of the value of $\mathcal{L}$. In particular, we have that

$$\mathcal{L}(\mathbf{z} + \Delta\mathbf{z}, t, \boldsymbol{\theta}) - \mathcal{L}(\mathbf{z}, t, \boldsymbol{\theta}) \approx \nabla_1 \mathcal{L}(\mathbf{z}, t, \boldsymbol{\theta}) \Delta\mathbf{z}.$$

Similar equations can written for $\nabla_2 \mathcal{L}$ and $\nabla_3 \mathcal{L}$.

- We use the convention that, for a vector function $\mathbf{f}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$, the gradient $\nabla_{\mathbf{x}} \mathbf{f} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ would be an $m \times n$ matrix.
  - As a result, $\nabla_{\mathbf{z}} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}}$ is a row vector in $\mathbb{R}^{1 \times d}$.

- Note that our end goal is to compute

$$\nabla_3 \mathcal{L}(\mathbf{z}_0, t_0, \boldsymbol{\theta}) = \left.\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\right|_{(\mathbf{z}_0, t_0, \boldsymbol{\theta})}.$$

  This is the gradient that we will use to update $\boldsymbol{\theta}$ in any SGD algorithm.

- To derive the adjoint sensitivity method, we focus on a trajectory of the hidden variable $\mathbf{z}$ during the time interval $[t_0, t_1]$. To avoid confusion, let us denote this fixed trajectory by $\mathbf{z}^*(t)$, and we say that it is determined by simulating the neural ODE backward in time from t_1. In other words,

$$\mathbf{z}^*(t) = \mathbf{z}_1^* + \int_{t_1}^{t} \mathbf{f}(\mathbf{z}^*(u), u, \boldsymbol{\theta})\, \mathrm{d}u.$$

  We note that $\mathbf{z}^*(t)$ is a function of t, $\mathbf{z}_1^*$, t_1, and $\boldsymbol{\theta}$.

- We define the **adjoint function**

$$\mathbf{a} : (t, \boldsymbol{\theta}) \mapsto \nabla_1 \mathcal{L}(\mathbf{z}^*(t), t, \boldsymbol{\theta}).$$

  In other words,

$$\mathbf{a}(t, \boldsymbol{\theta}) = \left.\frac{\partial \mathcal{L}}{\partial \mathbf{z}}\right|_{(\mathbf{z}^*(t), t, \boldsymbol{\theta})}.$$

- We adjoint function has the following property.

  Theorem 1.

$$\left.\frac{\partial \mathbf{a}}{\partial t}\right|_{(t, \boldsymbol{\theta})} = -\mathbf{a}(t, \boldsymbol{\theta}) \left.\frac{\partial \mathbf{f}}{\partial \mathbf{z}}\right|_{(\mathbf{z}^*(t), t, \boldsymbol{\theta})}$$

  Proof. By definition,

$$\left.\frac{\partial \mathbf{a}}{\partial t}\right|_{(t, \boldsymbol{\theta})} = \lim_{\varepsilon \to 0^+} \frac{\mathbf{a}(t+\varepsilon, \boldsymbol{\theta}) - \mathbf{a}(t, \boldsymbol{\theta})}{\varepsilon}.$$

  To evalute the above expression, we shall rewrite $\mathbf{a}(t, \boldsymbol{\theta})$ in terms of $\mathbf{a}(t+\varepsilon, \boldsymbol{\theta})$.

  First, we note that

$$\mathcal{L}(\mathbf{z}^*(t), t, \boldsymbol{\theta}) = \mathcal{L}(\mathbf{z}^*(t+\varepsilon), t+\varepsilon, \boldsymbol{\theta})$$

  for any $0 \le \varepsilon \le t_1 - t$. This is because the loss should be the same if we start at any point on the trajectory $\{\mathbf{z}^*(t) : t_0 \le t \le t_1\}$. Hence, we can consider the process of computing $\mathcal{L}(\mathbf{z}^*(t), t, \boldsymbol{\theta})$ by first sending $(\mathbf{z}^*(t), t, \boldsymbol{\theta})$ to $(\mathbf{z}^*(t+\varepsilon), t+\varepsilon, \boldsymbol{\theta})$ and then slapping $\mathcal{L}$ on $(\mathbf{z}^*(t+\varepsilon), t+\varepsilon, \boldsymbol{\theta})$. Let $\mathbf{e}_\varepsilon$ be the function $(\mathbf{z}^*(t), t, \boldsymbol{\theta})$ to $(\mathbf{z}^*(t+\varepsilon), t+\varepsilon, \boldsymbol{\theta})$. It follows that

$$\mathbf{e}_\varepsilon \left(\begin{bmatrix} \mathbf{z} \\ t \\ \boldsymbol{\theta} \end{bmatrix} \right) = \begin{bmatrix} \mathbf{z} \\ t \\ \boldsymbol{\theta} \end{bmatrix} + \varepsilon \begin{bmatrix} \mathbf{f}(\mathbf{z}, t, \boldsymbol{\theta}) \\ 1 \\ \mathbf{0} \end{bmatrix} + O(\varepsilon^2).$$

  So,

$$\left.\frac{\partial \mathbf{e}_\varepsilon}{\partial (\mathbf{z}, t, \boldsymbol{\theta})}\right|_{(\mathbf{z}^*(t), t, \boldsymbol{\theta})} = I + \varepsilon \begin{bmatrix} \left.\frac{\partial \mathbf{f}}{\partial \mathbf{z}}\right|_{(\mathbf{z}^*(t), t, \boldsymbol{\theta})} & \left.\frac{\partial \mathbf{f}}{\partial t}\right|_{(\mathbf{z}^*(t), t, \boldsymbol{\theta})} & \left.\frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}}\right|_{(\mathbf{z}^*(t), t, \boldsymbol{\theta})} \\ \mathbf{0} & 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} + O(\varepsilon^2).$$

Now, by the chain rule, we have that

$$\left.\frac{\partial \mathcal{L}}{\partial(\mathbf{z}, t, \boldsymbol{\theta})}\right|_{(\mathbf{z}^*(t), t, \boldsymbol{\theta})} = \frac{\partial \mathcal{L}}{\partial \mathbf{e}_\varepsilon(\mathbf{z}(t), t, \boldsymbol{\theta})} \frac{\partial \mathbf{e}_\varepsilon(\mathbf{z}(t), t, \boldsymbol{\theta})}{\partial(\mathbf{z}(t), t, \boldsymbol{\theta})}$$

$$\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{z}(t)} & \frac{\partial \mathcal{L}}{\partial t} & \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{z}(t+\epsilon)} & \frac{\partial \mathcal{L}}{\partial (t+\epsilon)} & \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \end{bmatrix} \left(I + \varepsilon \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial \mathbf{z}(t)} & \frac{\partial \mathbf{f}}{\partial t} & \frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}} \\ \mathbf{0} & 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} + O(\varepsilon^2) \right).$$

By multipling the matrices out, it follows that

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}(t)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}(t+\varepsilon)} + \varepsilon \frac{\partial \mathcal{L}}{\partial \mathbf{z}(t+\varepsilon)} \frac{\partial \mathbf{f}}{\partial \mathbf{z}(t)} + O(\varepsilon^2).$$

In other words,

$$\left.\frac{\partial \mathcal{L}}{\partial \mathbf{z}}\right|_{(\mathbf{z}, t, \boldsymbol{\theta})} = \left.\frac{\partial \mathcal{L}}{\partial \mathbf{z}}\right|_{(\mathbf{z}+\varepsilon, t+\varepsilon, \boldsymbol{\theta})} \left.\frac{\partial \mathbf{f}}{\partial \mathbf{z}}\right|_{(\mathbf{z}(t), t, \boldsymbol{\theta})}$$

References

[CRBD18] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[DKB14] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation, 2014.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[RM15] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows, 2015.