Demystifying the mathematics behind Neural ODEs

Harsh Gazula and Sergey Plis

Tri-institutional Center for Translational Research in Neuroimaging and Data Science (TReNDS):
Georgia State University/Georgia Institute of Technology/Emory University, Atlanta, GA 30303

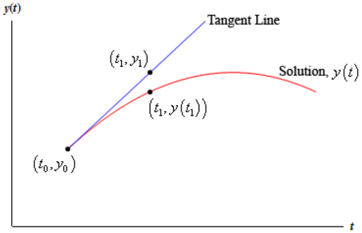
December 6, 2019

ODEs & Euler Method

$$\frac{dy}{dt} = f(t, y) \qquad y(t_0) = y_0 \tag{1}$$

$$\left.\frac{dy}{dt}\right|_{t=t_0} = f(t_0, y_0) \tag{2}$$

$$y_1 = y_0 + f(t_0, y_0)(t_1 - t_0) \tag{3}$$



In general, $y_t = y_0 + \int_0^t f(t, y)dt = \text{ODESolve}(y_0, f, t_0, t_1)$

Reference: Link

Residual Network (building block)

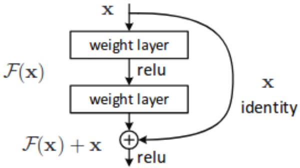


Figure 2. Residual learning: a building block.

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, W_i) + \mathbf{x} \tag{4}$$

In general (for any hidden state):

$$\mathbf{h}_{t+1} = F(\mathbf{h}_t, \theta_t) + \mathbf{h}_t \tag{5}$$

$$\mathbf{h}_{t+1} = \frac{\Delta t}{\Delta t} F(\mathbf{h}_\mathbf{t}, \theta_\mathbf{t}) + \mathbf{h}_t \tag{6}$$

$$\mathbf{h}_{t+1} = \Delta t G(\mathbf{h}_\mathbf{t}, \theta_\mathbf{t}) + \mathbf{h}_t \tag{7}$$

Reference: Kevin Gibson, [1]

From ResNet to ODENet via Euler method

$$y = y_0 + f(t_0, y_0)(t - t_0) \tag{8}$$

$$\mathbf{h}_{t+1} = \mathbf{h}_t + G(\mathbf{h_t}, \theta_\mathbf{t})\Delta t \tag{9}$$

The key takeaway is:

$$\frac{d\mathbf{h}(t)}{dt} = G(\mathbf{h}(t), t, \theta) \tag{10}$$

$$\mathbf{h}(t_1) = \text{ODESolve}(\mathbf{h}(t_0), G, t_0, t_1, \theta) \tag{11}$$

Let's carefully switch **h** with **z**, **G** with **f** and we have

$$\mathbf{z}(t_1) = \text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta) \tag{12}$$

(I made this switch to sync with the paper)

So far..the main idea

- A chain of residual blocks in a neural network is basically a solution of the ODE with the Euler method.

$$\mathbf{z}_{t+1} = F(\mathbf{z}_t, \theta_t) + \mathbf{z}_t \tag{13}$$

- Euler method is too primitive to solve an ODE. So, let's replace ResNet/EulerSolverNet with some abstract concept as ODESolveNet,

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t, \theta) \tag{14}$$

- ODE is then solved with black box solver and the output state $\mathbf{h_t}$ $(\mathbf{z_t})$ is then used to compute some loss L i.e.

$$L\Big(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\Big) \tag{15}$$

This is forward mode integration aka forward propagation

Reference: Honchar

Backpropagation

- The goal in backpropagation is to find the gradients $\frac{\partial L}{\partial \theta}$
- But, how do we do it?
- Let's go back to the ResNets again

$$\text{Consider} \quad \frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t, \theta) \tag{16}$$

$$\text{i.e.} \quad \frac{d\mathbf{z}(t)}{dt} = \mathbf{NN}(\mathbf{z}(t), t, \theta) \tag{17}$$

- The gradient of loss can be computed easily with existing methods
- However, in the case of ODENet, there are memory issues and other issues such as infeasibility or non-differentiability of the solvers
- We will take a slight detour to understand the adjoint method

Reference: Krzysztof Kolasinski

Adjoint Method: Introduction

Let's say we have a system of equations:

$$\mathbf{f}(\mathbf{u}, \mathbf{p}) = 0 \tag{18}$$

whose solution is $\mathbf{u} = \mathbf{F}(\mathbf{p})$ and, we want to find the values of the parameters $\mathbf{p}$ that minimize (or maximize) a given (scalar) function $\mathbf{g}(\mathbf{u})$. So, we need

$$\frac{\partial \mathbf{g}}{\partial \mathbf{p}} = \frac{\partial \mathbf{g}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{p}} \tag{19}$$

But $\frac{\partial \mathbf{u}}{\partial \mathbf{p}}$ can only be determined from

$$\frac{\partial \mathbf{f}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{p}} + \frac{\partial \mathbf{f}}{\partial \mathbf{p}} = 0 \tag{20}$$

Reference: [2]

Adjoint Method (cont..)

$$\frac{\partial \mathbf{u}}{\partial \mathbf{p}} = -\frac{\partial \mathbf{f}}{\partial \mathbf{u}}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{p}} \tag{21}$$

Putting equation 15 in equation 13

$$\frac{\partial \mathbf{g}}{\partial \mathbf{p}} = \frac{\partial \mathbf{g}}{\partial \mathbf{u}} \left(-\frac{\partial \mathbf{f}}{\partial \mathbf{u}}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{p}} \right) \tag{22}$$

$$(1 \times m) = (1 \times n) \times (n \times n) \times (n \times m) \tag{23}$$

How about we calculate this instead?

$$\frac{\partial \mathbf{g}}{\partial \mathbf{p}} = -\left(\frac{\partial \mathbf{g}}{\partial \mathbf{u}} \frac{\partial \mathbf{f}}{\partial \mathbf{u}}^{-1} \right) \frac{\partial \mathbf{f}}{\partial \mathbf{p}} \tag{24}$$

Reference: [2]

Adjoint Method (cont...)

Let

$$\lambda^T = \frac{\partial \mathbf{g}}{\partial \mathbf{u}} \frac{\partial \mathbf{f}}{\partial \mathbf{u}}^{-1} \tag{25}$$

$$\implies \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \lambda = \frac{\partial \mathbf{g}}{\partial \mathbf{u}}^T \tag{26}$$

Essentially, we are solving the above system of equations (which is more stable than matrix inversion).

- Oddly enough, the name adjoint comes from the fact that we are taking a transpose of a matrix.
- $\frac{\partial g}{\partial p}$ is the sensitivity of the function $\mathbf{g}$ to the changes in the parameters $\mathbf{p}$
- You now know *adjoint sensitivity*
- The adjoint method can be understood as a continuous version of the chain rule.

Reference: [2]

Examples

Example 1:

Suppose $\mathbf{u} = \begin{bmatrix} p_1^2 + p_2 \\ p_1 p_2 \end{bmatrix}$ and $g(\mathbf{u}) = u_1 + u_2^2$. We want to find $\dfrac{\partial g}{\partial \mathbf{p}} = \begin{bmatrix} \frac{\partial g}{\partial p_1} \\ \frac{\partial g}{\partial p_2} \end{bmatrix}$.

Example 2:

Suppose

$$f_1(u_1, u_2, p_1, p_2) = u_1 + u_2 + p_1$$
$$f_2(u_1, u_2, p_1, p_2) = u_1^3 - u_2 + p_2$$
$$g(u_1, u_2) = u_1^2 + u_2^2$$

If you are wondering, what this is all about..

Think of $\mathbf{g}$ as L, $\mathbf{u}$ as $\mathbf{z}$ and $\mathbf{p}$ as θ

Reference: [2]

Optimization view

The simple one

$$\begin{aligned} \underset{\mathbf{p}}{\text{Minimize}} \quad & \mathbf{g}(\mathbf{u}) \\ \text{subject to} \quad & \mathbf{f}(\mathbf{u}, \mathbf{p}) = 0 \end{aligned} \tag{27}$$

The advanced one

$$\begin{aligned} \underset{\theta}{\text{Minimize}} \quad & L\Big(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\Big) \\ \text{subject to} \quad & \mathbf{f}(\mathbf{z}(t), t_0, t_1, \theta) = 0 \\ & \mathbf{z}(\mathbf{0}) = \mathbf{x} \end{aligned} \tag{28}$$

Notice

$$L\Big(\text{ODESolve}(z(t_0), f, t_0, t_1, \theta)\Big) = L\Big(z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta) dt\Big) \tag{29}$$

Backpropagation (cont...)

Notice how L is a function of $\mathbf{z_t}$. We compute the gradient of L w.r.t input state using chain rule as follows:

$$\frac{\partial L}{\partial \mathbf{z_t}} = \frac{\partial L}{\partial \mathbf{z_{t+1}}} \frac{\partial \mathbf{z_{t+1}}}{\partial \mathbf{z_t}} \tag{30}$$

We are interested in infinitesimal (continuous) change in hidden state, i.e.

$$\mathbf{z}(t+\varepsilon) = \int_{t}^{t+\varepsilon} f(\mathbf{z}(t), t, \theta)dt + \mathbf{z}(t) = T_{\varepsilon}(\mathbf{z}(t), t) \tag{31}$$

Applying the chain rule

$$\frac{dL}{\partial \mathbf{z}(t)} = \frac{dL}{d\mathbf{z}(t+\varepsilon)} \frac{d\mathbf{z}(t+\varepsilon)}{d\mathbf{z}(t)} \tag{32}$$

Backpropagation (cont...)

We let

$$a(t) = -\frac{\partial L}{\partial z(t)} \tag{33}$$

$a(t)$ is called adjoint, its dynamics is given by another ODE,

$$\frac{da(t)}{dt} = -a(t)\frac{\partial f(z(t), t, \theta)}{\partial z} \tag{34}$$

The proof for the above equation is in Appendix B.1 of the NODE paper.

So, how is the adjoint state helpful in backpropagation?

Backpropagation (cont...)

Not to lose track of the goal in backpropagation which is finding:

$$\frac{\partial L}{\partial \mathbf{z}(\mathbf{t_0})} \text{ aka } a(t_0) \qquad \& \qquad \frac{\partial L}{\partial \theta} \tag{35}$$

To get $a(t_0)$ we need to solve another ODE *backwards in time* and the equations are given as follows :

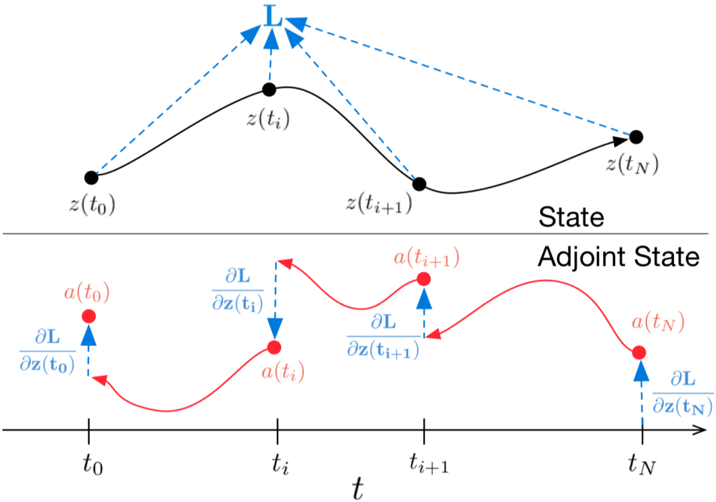
$$a(t_0) = a(t_N) - \int_{t_N}^{t_0} a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial z} dt \tag{36}$$

The initial condition is as follows:

$$a(t_N) = \frac{\partial L}{\partial z(t_N)} \tag{37}$$

which is just a gradient of loss w.r.t. final hidden state

Visually...



Reference: [3]

Backpropagation (cont...)

If you are really paying attention to these slides, I left out the most important derivation/discussion. Any guesses?

Algorithm 2 Complete 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 ${}^{\partial L}/_{\partial \mathbf{z}(t_1)}$

$\frac{\partial L}{\partial t_1} = \frac{\partial L}{\partial \mathbf{z}(t_1)}^{\mathsf{T}} f(\mathbf{z}(t_1), t_1, \theta)$ ▷ Compute gradient w.r.t. t_1

$s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}_{|\theta|}, -\frac{\partial L}{\partial t_1}]$ ▷ Define initial augmented state

def aug_dynamics($[\mathbf{z}(t), \mathbf{a}(t), \cdot, \cdot], t, \theta$): ▷ Define dynamics on augmented state

  return $[f(\mathbf{z}(t), t, \theta), -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f}{\partial \mathbf{z}}, -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f}{\partial \theta}, -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f}{\partial t}]$ ▷ Compute vector-Jacobian products

$[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial t_0}] = \text{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$ ▷ Solve reverse-time ODE

return $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial t_0}, \frac{\partial L}{\partial t_1}$ ▷ Return all gradients

Reference: [3]

References

K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

G. D. Granzow, "A tutorial on adjoint methods and their use for data assimilation in glaciology," *Journal of Glaciology*, vol. 60, no. 221, pp. 440–446, 2014.

R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," 2018.