Notes on Neural Ordinary Differential Equations

Hui Wei

February 2020

1 Ordinary Differential Equations

1.1 Definition

Mathematically, Ordinary Differential Equations (ODE) initial value problem is represented as follows:

$$\begin{aligned} \frac{d\mathbf{z}(t)}{dt} &= f(\mathbf{z}(t), t) \\ \mathbf{z}(t_0) &= \mathbf{z}_0 \end{aligned} \tag{1}$$

The problem is to find function $\mathbf{z}(t)$. We set the derivative to be dependent on both $\mathbf{z}(t)$ and t for flexibility. Note that $f(\mathbf{z}(t), t)$ and $\mathbf{z}_0$ are known. The ODE describes the derivative or gradient at each point $(t, \mathbf{z}(t))$, and the initial value $\mathbf{z}(t) = \mathbf{z}_0$ specifies the constant in the final solution.

1.2 ODE for Classification and Regression Problems

We can use initial value problem to solve classification and regression problems. In those problems, we want to find the relationship between the input space $\mathcal{X}$ and the output space $\mathcal{Y}$. We can define formally these problems based on ODE as follows:

$$\begin{aligned} \frac{d\mathbf{z}(t)}{dt} &= f(\mathbf{z}(t), t, \theta) \\ \mathbf{z}(t_0) &= \mathbf{z_0} \\ \mathbf{z}(t_1) &= \mathbf{z_1} \end{aligned} \tag{2}$$

f is the derivative or gradient over $t \sim \mathbf{z}(t)$ plane. For flexibility, we use f_2 to encode input feature $\mathbf{x}$ as $\mathbf{z_0} = f_2(\mathbf{x})$. Since the solution of ODE $\mathbf{z}_1$ at t_1 has the same dimension as the encoded input $\mathbf{z}_0$ which is probably different from that of the groundtruth $\mathbf{y}$, so f_3 is used to decode $\mathbf{z}_1$ and get the output $\hat{\mathbf{y}} = f_3(\mathbf{z}_1)$. Different from (1), gradient f here is unknown and parameterized by θ. We can use some predefined function as f and some loss function $\mathcal{L}$, such as cross entropy loss for classification and mean square root loss for regression, to get the optimal parameter θ and get the best approximation of the relationship between $\mathcal{X}$ and $\mathcal{Y}$ with training samples $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$.

In the paper, authors stated that they considered t_0 and t_1 also as parameters and trained them as well. However, suggested by Xitian Han that training those two parameters does not work, so just treat them as the predefined parameters. No matter what t_0 and t_1 are set, the derivative learnt by machine learning models are equivalent.

1.3 Intuitive Understanding

We can think of the gradient as the flow direction (please see Fig.1) in the $t \sim \mathbf{z}(t)$ plane, which is like in the river. Our boat will go directly along the gradient direction. We want to take the boat from the start point $(t_0, \mathbf{z}(t_0))$ to the end point $(t_1, \mathbf{z}(t_1))$, and we can change the flow direction at any point in the river to achieve this goal. Given several start and end points, how do we change the flow so that all actual destinations are closest to corresponding desired destinations.

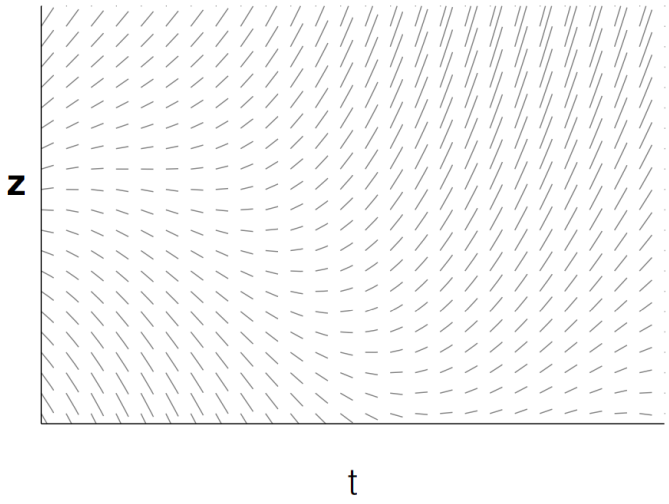

Figure 1: Visualization of gradients defined over $t \sim \mathbf{z}$ plane.

1.4 Advantages of ODE for classification and regression

It turns out that using ODE for solving these problems can save the number of parameters and the storing memory. We use the example in this blog here to explain it: in linear regression, the simplest predefined function is $y = ax + b$ and there are two parameters a and b and need to store gradients for both of them. If we define the same linear function using its gradient $\frac{dy}{dx} = a$, the number of parameters is just a. In ODE, there is no need to compute the constant b since the solver will give the integral solution numerically instead of analytically.

2 Numerical Solvers for ODE

2.1 Euler's Method

The analytical solution for problem (1) is:

$$\begin{aligned} \mathbf{z}(t_1) &= \mathbf{z}(t_0) + \int_{t_0}^{t_1} \frac{d\mathbf{z}(t)}{dt} dt \\ &= \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta) dt \end{aligned} \tag{3}$$

θ is optional since it is independent on t and is just the parameter to depict the gradient function.

Since most of ODE problem (1) cannot be solved analytically, several numerical methods was invented and Euler's method is the simplest one. It is derived from the definition of the derivative:

$$f(\mathbf{z}(t), t, \theta) = \frac{d\mathbf{z}(t)}{dt} = \lim_{\Delta t \to 0} \frac{\Delta \mathbf{z}(t)}{\Delta t} = \lim_{\Delta t \to 0} \frac{\mathbf{z}(t + \Delta t) - \mathbf{z}(t)}{\Delta t} \tag{4}$$

From (4), we can get

$$\mathbf{z}(t + \Delta t) = \mathbf{z}(t) + \Delta t f(\mathbf{z}(t), t, \theta) \tag{5}$$

This is Euler's Method: from t_0, it uses a fixed step Δt to get the next evaluation variable $t_0 + \Delta t$. Meanwhile, it moves from the initial value $\mathbf{z}(t_0)$ along the derivative $f(\mathbf{z}(t_0), t_0, \theta)$ to add $\Delta t f(\mathbf{z}(t_0), t_0, \theta)$ on it. Then replace t_0 with $t_0 + \Delta t$ and iterate above steps until the final evaluation variable t_1.

2.2 ResNet

The residual block in ResNet is:

$$\mathbf{h}(t + 1) = \mathbf{h}(t) + f(\mathbf{h}(t), t, \theta) \tag{6}$$

Equation (6) has the same format with (5) when $\Delta t = 1$. Each layer of the block defines the gradient $f(\mathbf{h}(t), t, \theta)$ at that depth (or time, which is consistent with the paper) t and can be trained through backpropagation. Therefore, residual block is actually Euler's Methods for ODE problem. Note that residual block represents the gradient and is a ODE solver at the same time.

For solving problem (3), Euler's Method or residual block has two shortcomings: 1) it uses the fixed update step, which is inflexible and always leads to larger approximation error, 2) bottleneck block can only control the gradient function at each **discrete** depth t. Since there are so many more advanced ODE solvers which can take adaptive evaluate steps according to the problem difficulty, we need to assume the depth (or time) as continuous instead of discrete like ResNet, so that the gradient can be controlled everywhere on $t \sim \mathbf{z}$ plane.

2.3 Neural ODE

The core question to solve is Equation (2) for both regression and classification problems. Since neural networks are the universe function approximator, it can be used to define the gradient function $f(\mathbf{z}(t), t, \theta)$ everywhere on $t \sim \mathbf{z}$ plane. Also, we won't use residual block (Euler's Method) to solve the ODE problem numerically, instead, more advanced solvers are used.

The visualized difference between Neural ODE and ResNet can be seen in Fig.2 and Fig.1 in the original paper.

2.3.1 Forward Pass

Before any forward pass, what we have right now is (1) a neural network which represents gradient $f(t, \mathbf{z}(t), \theta)$ over $t \sim \mathbf{z}(t)$ plane, (2) Training samples $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$. Therefore, we have the following initial problem:

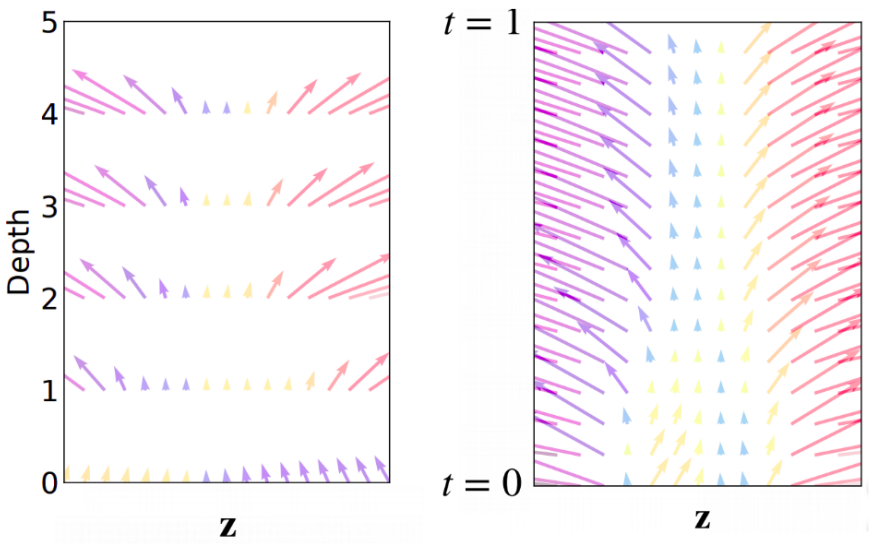

Figure 2: Gradient over $t \sim \mathbf{z}$ defined by residual block (Left) and Neural ODE. The former one is discrete and the latter one is continuous.

$$\begin{aligned} \frac{d\mathbf{z}(t)}{dt} &= f(\mathbf{z}(t), t, \theta) \\ \mathbf{z}(t_0) &= \mathbf{x} \end{aligned} \tag{7}$$

We can call ODESolver$(f, \mathbf{z}(t_0), t_0, t_1)$ to get the output $\hat{\mathbf{y}} = \mathbf{z}(t_1)$ along the evaluation trajectory by some solver. For each training sample, we use the same solver to solve problem (7) and we can compute the corresponding loss function based on the ground-truth $\mathbf{y}$ and output $\hat{\mathbf{y}}$. f_2 and f_3 is ignored here for simplicity.

2.3.2 Backward Pass

Since we use neural networks to define the gradient function $f(\mathbf{z}(t), t, \theta)$, we need to use the gradient w.r.t.θ and the loss function L to update its parameters using backpropagation. However, backpropagation directly from ODESolver leads to high memory and high error, so the paper computes the gradient via adjoint sensitivity method.

From the paper, the gradient is

$$\frac{dL}{d\theta} = -\int_{t_0}^{t_1} \mathbf{a}^T(t) \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt \tag{8}$$

This is an ODE initial value problem:

$$\frac{(\frac{dL}{d\theta})}{dt} = -\mathbf{a}^T(t) \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta}, \quad \frac{dL}{d\theta}\Big|_{t_1} = \mathbf{0}_{|\theta|} \tag{9}$$

In Eq.(8), $\mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{z}(t)}$. From (8), we can see to compute $\frac{dL}{d\theta}$, it also needs to know $\mathbf{a}(t)$, $\mathbf{z}(t)$ at both t_0 and t_1. Note that like backpropagation, the flow to compute the gradient is reversed against the forward pass, so the integral is from t_1 to t_0 and the initial value is at t_1. Also, from (8), we can see the initial value is zeros since there is no item before the integral, unlike (3). Please see more details in Appendix. B of the paper.

It is easy to know $\mathbf{z}(t_1)$ from the forward pass and $\mathbf{a}(t_1)$ directly from the loss function. To get $\mathbf{z}(t_0)$ and $\mathbf{a}(t_0)$, two ODE problems needs to be solved:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}^T(t)\frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}},\ \mathbf{a}(t_1) = \frac{\partial L}{\partial \mathbf{z}(t_1)} \tag{10}$$

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t, \theta),\ \mathbf{z}(t_1) = \mathbf{z}_1 \tag{11}$$

To solve ODE problems (9)(10)(11), we can just call ODESolver once using augmented dynamics $[f(\mathbf{z}, t, \theta), -\mathbf{a}^T\frac{\partial f(\mathbf{z},t,\theta)}{\partial \mathbf{z}}, -\mathbf{a}^T\frac{\partial f(\mathbf{z},t,\theta)}{\partial \theta}]$. Then it can compute those values at the same intermediate evaluation times t_i. Once we get $\frac{dL}{d\theta}$, optimizer such like SGD and Adam can be used to update neural network parameters which represent the gradient (flow direction) over $t \sim \mathbf{z}$ plane.