6 - Neural Ordinary Differential Equations

Yi Tang (yt1433)

June 12, 2019

1 From networks to ODE

Different networks architectures (such as ResNets, some simple recurrent neural nets (RNN) and normalizing flows), transform the hidden states with an iteration of the form

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

for $t = 0, \ldots, T$. By concatenating the parameter vectors $\boldsymbol{\theta}_t$ into a single vector $\boldsymbol{\theta}$, the transformations can be equivalently formulated as

$$\begin{aligned} \mathbf{h}_{t+1} &= \mathbf{h}_t + \mathbf{f}(t, \mathbf{h}_t, \boldsymbol{\theta}) \\ \mathbf{h}_{t+1} - \mathbf{h}_t &= \mathbf{f}(t, \mathbf{h}_t, \boldsymbol{\theta}) \end{aligned}$$

The last equation has the form of a *difference equation*. If the number of hidden states is increased and the step of index t as well as the factor of the increment is decreased accordingly, then the equation becomes

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

for $t = 0, \Delta t, \ldots, T$. This corresponds to an explicit Euler step for the ODE obtained by taking the limit $\Delta t \to 0$:

$$\frac{\mathrm{d}\mathbf{h}(t)}{\mathrm{d}t} = \mathbf{f}(t, \mathbf{h}(t), \boldsymbol{\theta}) \qquad \text{for} \quad t \in [0, T] \tag{2}$$

If one is given the function $\mathbf{f}$ and the parameter $\boldsymbol{\theta}$, the computation of the output $\mathbf{h}(T)$ can be carried out by using any ordinary differential equation numerical solver, which we can considered as blackbox in the following discussion. That is, we think about the blackbox ODE solver as a map

$$\text{ODESolve} : (\mathbf{h}(t_0), \mathbf{f}, t_0, t_1, \boldsymbol{\theta}) \mapsto \mathbf{h}(t_1).$$

where $\mathbf{h}(t)$ is the solution to (2) with initial condition $\mathbf{h}(t_0) = \mathbf{h}_0$; the output $\mathbf{h}(T)$ is then computed as

$$\mathbf{h}(T) = \text{ODESolve}(\mathbf{h}(0), \mathbf{f}, 0, T, \boldsymbol{\theta})$$

2 AutoGrad of ODE

In practice, we need to learn parameter $\boldsymbol{\theta}$. To replace standard networks with (neural) ODEs as neural network modules, the forward / backward propagation should be implemented. As previously stated, the forward propagation is accomplished by the ODE solver. Given input $\mathbf{z}(t_0)$ and internal parameter $\boldsymbol{\theta}$, the output of the neural ODE module (equipped with $\mathbf{f}$ and domain $[t_0, t_1]$) is

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

Given a loss function L, evaluated on $\mathbf{z}(t_1)$, we can easily compute $\frac{\mathrm{d}L}{\mathrm{d}\mathbf{z}(t_1)}$; to train the network, we also need to compute $\frac{\mathrm{d}L}{\mathrm{d}\mathbf{z}(t)}$ (for $t < T$) as well as $\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}}$. The backward propagation can thus be fulfilled using the adjoint method that we discussed in the previous lecture. In the following, we review the adjoint method for this specific case, following the approach presented in the paper (Chen et al., 2018).

As we saw in the last lecture, the adjoint method for ODE constraints, consists in solving an auxiliary (backward) ODE (called adjoint equation) whose solution $\mathbf{a}(t)$ is called adjoint. In fact, the adjoint is exactly $\mathbf{a}(t) = \frac{\mathrm{d}L}{\mathrm{d}\mathbf{z}(t)}$ (we take it here as a row vector, with the same shape as $\mathbf{z}(t)^\top$), for which we can easily evaluate the initial value $\mathbf{a}(t_1) = \frac{\mathrm{d}L}{\mathrm{d}\mathbf{z}(t_1)}$. The adjoint ODE for $\mathbf{a}(t)$ is given by

$$\frac{\mathrm{d}\mathbf{a}(t)}{\mathrm{d}t} = -\mathbf{a}(t)\frac{\partial}{\partial \mathbf{z}(t)} f(\mathbf{z}(t), t, \boldsymbol{\theta}). \tag{3}$$

Given the adjoint $\mathbf{a}(t)$, the derivative of L with respect to $\boldsymbol{\theta}$ is given by

$$\begin{aligned} \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}} &= \frac{\partial L}{\partial \boldsymbol{\theta}} + \mathbf{a}(t_0)\frac{\mathrm{d}\mathbf{z}(t_0)}{\mathrm{d}\boldsymbol{\theta}} + \int_{t_0}^{t_1} \mathbf{a}(t)\frac{\partial}{\partial \boldsymbol{\theta}}\mathbf{f}(t, \mathbf{z}(t), \boldsymbol{\theta})\mathrm{d}t \\ &= \mathbf{0} - \int_{t_1}^{t_0} \mathbf{a}(t)\frac{\partial}{\partial \boldsymbol{\theta}}\mathbf{f}(t, \mathbf{z}(t), \boldsymbol{\theta})\mathrm{d}t \end{aligned} \tag{4}$$

In particular, $\frac{dL}{d\boldsymbol{\theta}}$ can be seen as the solution to a second backward ode for a vector $\mathbf{b}(t)$ such that $\mathbf{b}(t_1) = \mathbf{0}$ and

$$\dot{\mathbf{b}}(t) = -\mathbf{a}(t)\frac{\partial}{\partial \boldsymbol{\theta}}\mathbf{f}(t, \mathbf{z}(t), \boldsymbol{\theta})$$

so that $\mathbf{b}(t) = \frac{dL}{d\boldsymbol{\theta}}$ (When L has direct dependency on $\boldsymbol{\theta}$, e.g. L contains regularization terms of $\boldsymbol{\theta}$, the term $\frac{\partial L}{\partial \boldsymbol{\theta}}$ could be non-zero. Nevertheless that would does not add difficulty - one can simply apply $\frac{\partial L}{\partial \boldsymbol{\theta}}$ instead of $\mathbf{0}$ as the initial values to the ODE solver.) To find $\frac{dL}{d\boldsymbol{\theta}}$ and $\frac{dL}{d\mathbf{z}(t_0)}$, we thus need to solve the (augmented) adjoint backward ODE:

$$\frac{d}{dt}\begin{pmatrix}\mathbf{z}(t)\\ \mathbf{a}(t)\\ \mathbf{b}(t)\end{pmatrix} = \begin{pmatrix}\mathbf{f}(t, \mathbf{z}, \boldsymbol{\theta})\\ -\mathbf{a}(t)\frac{\partial}{\partial \mathbf{z}}\mathbf{f}(t, \mathbf{z}, \boldsymbol{\theta})\\ -\mathbf{a}(t)\frac{\partial}{\partial \boldsymbol{\theta}}\mathbf{f}(t, \mathbf{z}, \boldsymbol{\theta})\end{pmatrix} \doteq \mathbf{f}_{aug}\left(t, \begin{pmatrix}\mathbf{z}(t)\\ \mathbf{a}(t)\\ \mathbf{b}(t)\end{pmatrix}, \boldsymbol{\theta}\right)$$

We can therefore find the sought gradients applying

$$\begin{pmatrix}\cdot\\ \frac{\mathrm{d}L}{\mathrm{d}\mathbf{z}(t_0)}\\ \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}}\end{pmatrix} = \text{ODESolve}\left(\begin{pmatrix}\mathbf{z}(t_1)\\ \frac{\mathrm{d}L}{\mathrm{d}\mathbf{z}(t_1)}\\ \mathbf{0}\end{pmatrix}, \mathbf{f}_{aug}, t_1, t_0, \boldsymbol{\theta}\right)$$

2.1 Extra: Derivation of Adjoint Method

Equations (3) and (4) are two crucial equations for the adjoint method, which give us a way to back-propagate. In the following we derive these equations going through the proof in (Chen et al., 2018). Define the *ODE solver* operator

$$T_s^{\mathbf{f},t}(\mathbf{z}(t)) = \tilde{\mathbf{z}}(t+s)$$

where $\tilde{\mathbf{z}}(\tau)$ is the solution to the initial value problem

$$\begin{aligned}\frac{\mathrm{d}\tilde{\mathbf{z}}}{\mathrm{d}\tau} &= \mathbf{f}(\tilde{\mathbf{z}}(\tau)) \\ \tilde{\mathbf{z}}(t) &= \mathbf{z}(t)\end{aligned}$$

For sake of simplicity, we write $T_s^{\mathbf{f},t}$ as T_s since $\mathbf{f}$ and t can be clearly read from context. Then, by defining $\mathbf{a}(t) = \frac{\mathrm{d}L}{\mathrm{d}\mathbf{z}(t)}$, we have

$$\mathbf{a}(t) = \frac{\mathrm{d}L}{\mathrm{d}\mathbf{z}(t)} = \frac{\mathrm{d}L}{\mathrm{d}\mathbf{z}(t+s)}\frac{\mathrm{d}\mathbf{z}(t+s)}{\mathrm{d}\mathbf{z}(t)} = \mathbf{a}(t+s)\frac{\mathrm{d}T_s(\mathbf{z}(t))}{\mathrm{d}\mathbf{z}(t)}.$$

Consequently,

$$\begin{aligned}\frac{\mathrm{d}\mathbf{a}(t)}{\mathrm{d}t} &= \lim_{s\to 0}\frac{\mathbf{a}(t+s)-\mathbf{a}(t)}{s} \\ &= \lim_{s\to 0}\frac{\mathbf{a}(t+s)}{s}\left[\mathbf{I} - \frac{\mathrm{d}T_s(\mathbf{z}(t))}{\mathrm{d}\mathbf{z}(t)}\right] \\ &= \lim_{s\to 0}\frac{\mathbf{a}(t+s)}{s}\left[\mathbf{I} - \frac{\mathrm{d}}{\mathrm{d}\mathbf{z}(t)}\left(\mathbf{z}(t) + \int_t^{t+s} f(\mathbf{z}(\tau))\mathrm{d}\tau\right)\right] \\ &= -\lim_{s\to 0}\frac{\mathbf{a}(t+s)}{s}\frac{\mathrm{d}}{\mathrm{d}\mathbf{z}(t)}\int_t^{t+s} f(\mathbf{z}(\tau))\mathrm{d}\tau \\ &= -\lim_{s\to 0}\frac{\mathbf{a}(t+s)}{s}\frac{\mathrm{d}}{\mathrm{d}\mathbf{z}(t)}\left[s\cdot f(\mathbf{z}(t)) + o(s^2)\right] \\ &= -\lim_{s\to 0}\frac{\mathbf{a}(t+s)}{s}\left[s\cdot\frac{\partial}{\partial\mathbf{z}(t)}f(\mathbf{z}(t)) + o(s^2)\right] \\ &= -\mathbf{a}(t)\frac{\partial}{\partial\mathbf{z}(t)}f(\mathbf{z}(t)).\end{aligned}$$

The last expression is indeed the adjoint equation (3). By defining $\boldsymbol{\theta}(t)$ as the solution to $\boldsymbol{\theta}(t_0) = \boldsymbol{\theta}$, $\dot{\boldsymbol{\theta}}(t) = 0$, and following the same steps of above (with the augmented function $\mathbf{f}_{aug} = \begin{pmatrix}\mathbf{f}\\ \mathbf{0}\end{pmatrix}$ and augmented adjoint $\mathbf{a}_{aug}(t) = \begin{pmatrix}\frac{\mathrm{d}L}{\mathrm{d}\mathbf{z}(t)} & \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}(t)}\end{pmatrix}$), it can be derived that

$$\begin{aligned}\frac{\mathrm{d}\mathbf{a}_{aug}(t)}{\mathrm{d}t} &= -\mathbf{a}_{aug}(t)\begin{pmatrix}\frac{\partial}{\partial\mathbf{z}(t)}\mathbf{f}_{aug}(\mathbf{z}(t),\boldsymbol{\theta}(t)) & \frac{\partial}{\partial\boldsymbol{\theta}(t)}\mathbf{f}_{aug}(\mathbf{z}(t),\boldsymbol{\theta}(t))\end{pmatrix} \\ &= -\mathbf{a}_{aug}(t)\begin{pmatrix}\frac{\partial}{\partial\mathbf{z}(t)}\mathbf{f}(\mathbf{z}(t),\boldsymbol{\theta}(t)) & \frac{\partial}{\partial\boldsymbol{\theta}(t)}\mathbf{f}(\mathbf{z}(t),\boldsymbol{\theta}(t)) \\ \mathbf{0} & \mathbf{0}\end{pmatrix} \\ &= -\mathbf{a}(t)\begin{pmatrix}\frac{\partial}{\partial\mathbf{z}(t)}\mathbf{f}(\mathbf{z}(t),\boldsymbol{\theta}(t)) & \frac{\partial}{\partial\boldsymbol{\theta}(t)}\mathbf{f}(\mathbf{z}(t),\boldsymbol{\theta}(t))\end{pmatrix} \\ &= -\mathbf{a}(t)\begin{pmatrix}\frac{\partial}{\partial\mathbf{z}(t)}\mathbf{f}(\mathbf{z}(t),\boldsymbol{\theta}) & \frac{\partial}{\partial\boldsymbol{\theta}}\mathbf{f}(\mathbf{z}(t),\boldsymbol{\theta})\end{pmatrix}\end{aligned}$$

By integrating over t, we get

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}} = \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}(t_0)} = \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}(t_1)} - \int_{t_1}^{t_0} \mathbf{a}(t)\frac{\partial}{\partial\boldsymbol{\theta}}\mathbf{f}(\mathbf{z}(t),\boldsymbol{\theta})\mathrm{d}t = \mathbf{0} - \int_{t_1}^{t_0} \mathbf{a}(t)\frac{\partial}{\partial\boldsymbol{\theta}}\mathbf{f}(\mathbf{z}(t),\boldsymbol{\theta})\mathrm{d}t$$

that is equation (4).

3 Distribution Transformation by ODE

In normalizing flows, the iterative transformations of hidden states (1) is often to transport an initial input distribution. According to the change of variables formula, if

$$\mathbf{z}(0) \sim p(\mathbf{z}(0))$$

then

$$\mathbf{z}(t) \sim p(\mathbf{z}(t)) = p(\mathbf{z}(0))\left|\det\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}\mathbf{z}(0)}\right|^{-1}$$

Taking the logarithm, the relationship becomes

$$\log p(\mathbf{z}(t)) = \log p(\mathbf{z}(0)) - \log\left|\det\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}\mathbf{z}(0)}\right|.$$

To efficiently compute the probabilities, the Jacobian $\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}\mathbf{z}(0)}$ is required to be **nonsingular** and **efficiently computable** (or at least have efficiently computable determinant; generally, the computation of determinant has a cubic time complexity and often leads to a main bottleneck). Surprisingly, moving from the discrete case of normalizing flows to the continuous case of neural ODEs simplifies the computation. The dynamics of the log-probability can be computed as

$$\begin{aligned}
\frac{\mathrm{d}\log p(\mathbf{z}(t))}{\mathrm{d}t} &= \lim_{s\to 0}\frac{\log p(\mathbf{z}(t+s)) - \log p(\mathbf{z}(t))}{s} \\
&= -\lim_{s\to 0}\frac{1}{s}\log\left|\det\frac{\mathrm{d}T_s(\mathbf{z}(t))}{\mathrm{d}\mathbf{z}(t)}\right| \\
&= -\lim_{s\to 0}\left|\det\frac{\mathrm{d}T_s(\mathbf{z}(t))}{\mathrm{d}\mathbf{z}(t)}\right|^{-1}\frac{\partial}{\partial s}\left|\det\frac{\mathrm{d}T_s(\mathbf{z}(t))}{\mathrm{d}\mathbf{z}(t)}\right| \\
&= -\lim_{s\to 0}\frac{\partial}{\partial s}\det\left(\frac{\mathrm{d}T_s(\mathbf{z}(t))}{\mathrm{d}\mathbf{z}(t)}\right) \\
&= -\lim_{s\to 0}\mathrm{tr}\left(\mathrm{adj}\left(\frac{\mathrm{d}T_s(\mathbf{z}(t))}{\mathrm{d}\mathbf{z}(t)}\right)\left(\frac{\partial}{\partial s}\frac{\mathrm{d}T_s(\mathbf{z}(t))}{\mathrm{d}\mathbf{z}(t)}\right)\right) \\
&= -\lim_{s\to 0}\mathrm{tr}\left(\frac{\partial}{\partial s}\frac{\mathrm{d}T_s(\mathbf{z}(t))}{\mathrm{d}\mathbf{z}(t)}\right) \\
&= -\lim_{s\to 0}\mathrm{tr}\left(\frac{\partial}{\partial s}\left[\mathbf{I} + s\cdot\frac{\partial}{\partial\mathbf{z}(t)}\mathbf{f}(\mathbf{z}(t)) + o(s^2)\right]\right) \\
&= -\mathrm{tr}\left(\frac{\partial\mathbf{f}(\mathbf{z}(t))}{\partial\mathbf{z}(t)}\right)
\end{aligned}$$

where we used Jacobi formula. Therefore, moving to a continuous-time formulation also allows us to move from a complicated non-linear operator (i.e. the determinant) to a linear one (i.e. the trace).

3.1 Example

As an example of the computational advantages of neural ODEs for normalizing flows, consider the planar normalizing flow:

$$\mathbf{z}(t+1) = \mathbf{z}(t) + \mathbf{u}\rho(\mathbf{w}^\top \mathbf{z}(t) + b)$$

The determinant of the Jacobian can be computed as

$$\det\left(\mathbf{I} + \mathbf{u}\dot{\rho}(\dots)\mathbf{w}^\top\right) = 1 + \dot{\rho}(\dots)\mathbf{w}^\top\mathbf{u}$$

A main bottleneck here is that to make use of the above formula, we relied on the choice of a one-neuron layer layer transformation, which notably limits the approximation capabilities of the final network. The corresponding neural ODE is

$$\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = \mathbf{u}\rho(\mathbf{w}^\top \mathbf{z}(t) + b)$$

The trace of the Jacobian is given by

$$\mathrm{tr}\left(\mathbf{u}\dot{\rho}(\dots)\mathbf{w}^\top\right) = \dot{\rho}(\dots)\mathbf{w}^\top\mathbf{u}$$

As the trace is linear operator, this easily extends from the one-layer transformation $\mathbf{u}\rho(\mathbf{w}^\top\mathbf{z}(t)+b)$ to a sum $\sum_{i=1}^{n}\mathbf{u}_i\rho(\mathbf{w}_i^\top\mathbf{z}(t)+b_i)$:

$$\mathrm{tr}\left(\sum_{i=0}^{n}\mathbf{u}_i\dot{\rho}(\dots_i)\mathbf{w}_i^\top\right) = \sum_{i=0}^{n}\dot{\rho}(\dots_i)\mathbf{w}_i^\top\mathbf{u}_i$$

This comes with no added extra computational complexity and allows for a network architecture with an higher adaptability.

References

Chen, T. Q., Y. Rubanova, J. Bettencourt, and D. K. Duvenaud
2018. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, Pp. 6571–6583.