Carleton UNIVERSITY

Canada's Capital University

Neural ODEs

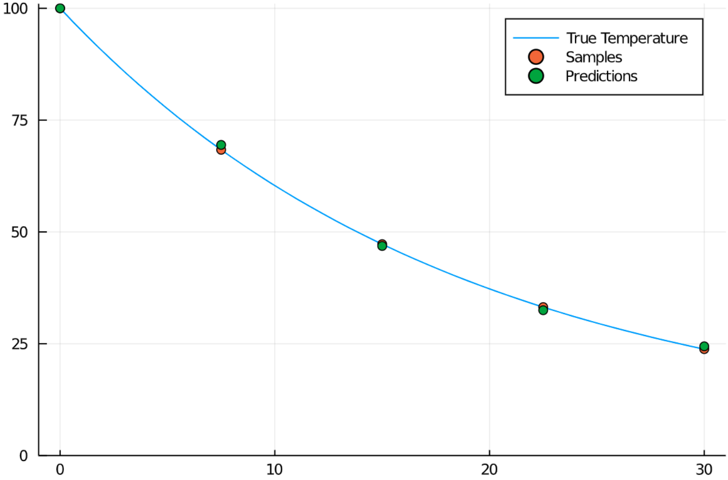
Andriy Drozdyuk, andriy@drozdyuk.com

Nov 29, 2020

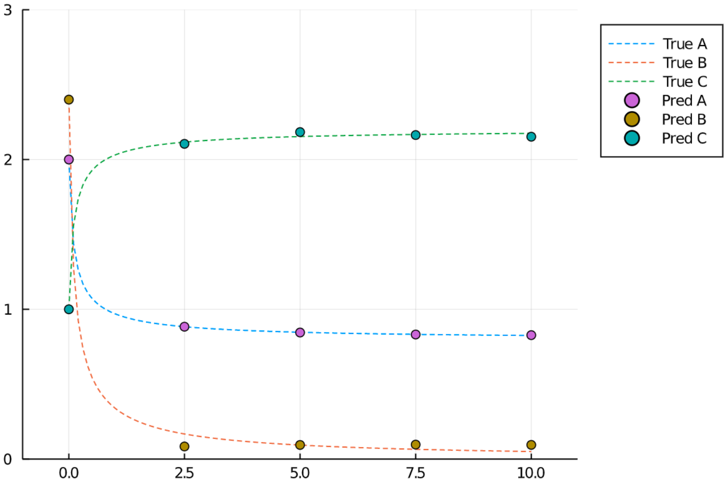
Table of Contents

Some cool results

Newton’s Law of Cooling

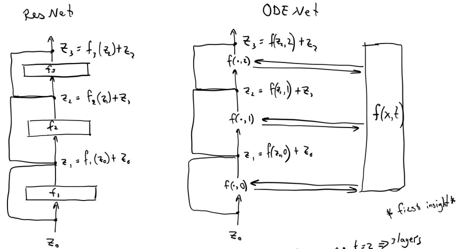


Chemical Reaction Rate Law



ML Part

Neural ODE Presentation



$F(z, \Theta)$ OR
$F(f_1, f_2, f_3, z, \Theta)$

e.g. $F(z_0, \Theta_0)$

num layers -1   e.g. $t_1 = 2 \Rightarrow 3$ layers

$F(z, t_1, \theta)$

OR $F(f, z, t_1, \theta)$

params θ

e.g. $F(f_\theta, z_0, 2, \theta_0)$

We can view z as a func. of t:

$$z(t)$$

then:

$$z_0 = z(0)$$
$$z_1 = z(1)$$
$$z_2 = z(2)$$
$$z_3 = z(3)$$

However, we don't "know" z, it's hidden.

Then we can write F as a cont. func. of t:

$z_0 = z(0)$ ⟵ we are given this as input

So that F is no longer a func. of z,
but we only need the input/'initial' value of z_0

$$F(f, z, t, \theta) = F(f, z(0), t, \theta)$$

ML Territory

Dynamical Systems Land

t_0 - "initial time"
implicitly zero for ns
before

t_1 - final time, "number of
layers" in some sense

$$\text{ODESolve}(z(0), f, t_0, t_1, \theta)$$

true output

our estimate

$$\underbrace{\hat{z}(t_1) \quad - \quad z(t_1)}_{\text{Loss}}$$

Problem: How to backprop through ODESolve? * second insight *

Dynamical Systems Part

Dynamical Systems (and ODEs)

Dynamical system is:

1) Some "state" that changes with time, e.g.

$$z(t)$$

2) Some "rule" for how that state changes:

$$\frac{dz}{dt} = f(\underset{\text{time}}{t}, \underset{\text{state}}{z}, \underset{\text{parameters}}{p})$$

Usually this is a system of ODEs
(ordinary diff. equations)

"Solution" to an ODE is a function

$$z(t)$$

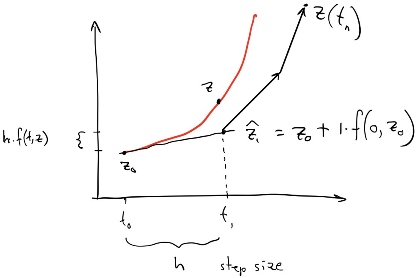
E.g. So we can find $z(5.5)$ directly, without the "rule" $\frac{dz}{dt}$.

Note: we are always given $z(0)$ – initial value
(important later)

Finding a solution to a dynamical system

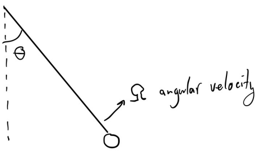
Given some dynamical system $f(t, z, p)$ with an initial condition z_0, find the solution $z(t)$.

Euler method: $z_{t+1} = z_t + h\, f(t, z_t)$



This way we can find $z(t)$ for any t.

Example : Pendulum




```
def pendulum(y, t, b, c):
    θ, Ω = y          ← state
    dydt = [Ω, -bΩ - c sin(θ)]
    return dydt
```

Assume

$b = 0.25$

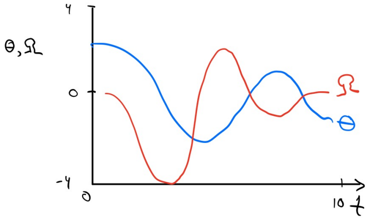
$c = 5.0$

```
y0 = [π-0.1, 0.0]
```
Initially pendulum is nearly vertical

```
t = np.linspace(0, 10, 101)
```
Solve for 101 points in $0 \le t \le 10$

```
from scipy.integrate import odeint
sol = odeint(pendulum, y0, t, args=(b, c))
```

Let's Plot sol[:,0], sol[:,1]



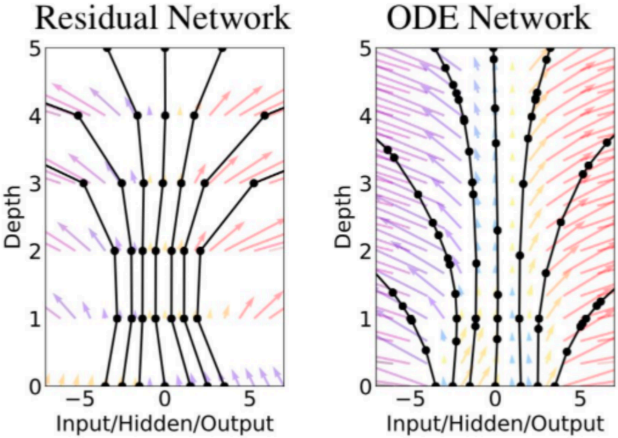
The solution

Adjoint method

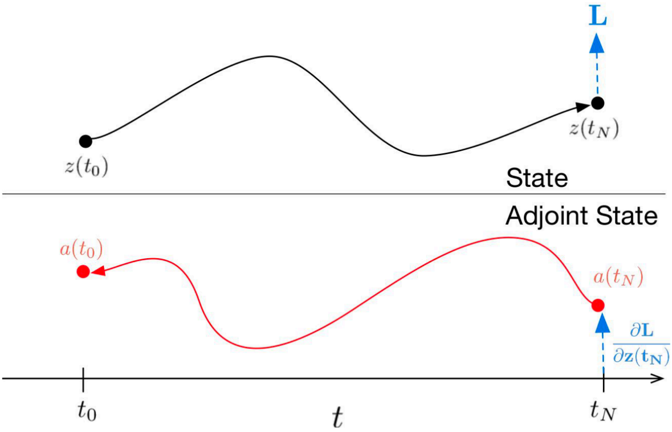
From ResNET to ODENet

Starting from the input $z(0)$ layer we can define output layer $z(T)$ to be the solution to an ODE initial value problem:

$$\frac{dz(t)}{dt} = f(z(t), t, \theta)$$



Backprop through ODE solver



Continuous time backprop

Continuous-time Backpropagation

Residual network. $a_t := \frac{\partial L}{\partial z_t}$

Forward: $z_{t+h} = z_t + hf(z_t)$

Backward: $a_t = a_{t+h} + ha_{t+h}\frac{\partial f(z_t)}{\partial z_t}$

Params: $\frac{\partial L}{\partial \theta} = ha_{t+h}\frac{\partial f(z(t),\theta)}{\partial \theta}$

Adjoint method. Define: $a(t) := \frac{\partial L}{\partial z(t)}$

Forward: $z(t+1) = z(t) + \int_t^{t+1} f(z(t))\,dt$

Backward: $\underbrace{a(t)}_{\text{adjoint state}} = a(t+1) + \int_{t+1}^{t} \underbrace{a(t)\frac{\partial f(z(t))}{\partial z(t)}}_{(1)?}dt$

Params: $\underbrace{\frac{\partial L}{\partial \theta}}_{(2)?} = \int_t^{t+1} a(t)\frac{\partial f(z(t),\theta)}{\partial \theta}\,dt$

We will show how to obtain adjoint differential equation (1) and gradients wr.t. θ (2) next!

Adjoint Method proof

Let $\mathbf{z}(t)$ follow the differential equation $\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t, \theta)$, where θ are the parameters. We will prove that if we define an adjoint state

$$\mathbf{a}(t) = \frac{dL}{d\mathbf{z}(t)} \tag{34}$$

then it follows the differential equation

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)\frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} \tag{35}$$

The adjoint state is the gradient with respect to the hidden state at a specified time t. In standard neural networks, the gradient of a hidden layer $\mathbf{h}_t$ depends on the gradient from the next layer $\mathbf{h}_{t+1}$ by chain rule

$$\frac{dL}{d\mathbf{h}_t} = \frac{dL}{d\mathbf{h}_{t+1}} \frac{d\mathbf{h}_{t+1}}{d\mathbf{h}_t}. \tag{36}$$

With a continuous hidden state, we can write the transformation after an ε change in time as

$$\mathbf{z}(t+\varepsilon) = \int_t^{t+\varepsilon} f(\mathbf{z}(t), t, \theta) dt + \mathbf{z}(t) = T_\varepsilon(\mathbf{z}(t), t) \tag{37}$$

and chain rule can also be applied

$$\frac{dL}{\partial \mathbf{z}(t)} = \frac{dL}{d\mathbf{z}(t+\varepsilon)} \frac{d\mathbf{z}(t+\varepsilon)}{d\mathbf{z}(t)} \qquad \text{or} \qquad \mathbf{a}(t) = \mathbf{a}(t+\varepsilon) \frac{\partial T_\varepsilon(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)} \tag{38}$$

The proof of (35) follows from the definition of derivative:

$$\frac{d\mathbf{a}(t)}{dt} = \lim_{\varepsilon \to 0^+} \frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t)}{\varepsilon} \tag{39}$$

$$= \lim_{\varepsilon \to 0^+} \frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t+\varepsilon)\frac{\partial}{\partial \mathbf{z}(t)} T_\varepsilon(\mathbf{z}(t))}{\varepsilon} \qquad \text{(by Eq 38)} \tag{40}$$

$$= \lim_{\varepsilon \to 0^+} \frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t+\varepsilon)\frac{\partial}{\partial \mathbf{z}(t)} \left(\mathbf{z}(t) + \varepsilon f(\mathbf{z}(t), t, \theta) + \mathcal{O}(\varepsilon^2)\right)}{\varepsilon} \qquad \text{(Taylor series around } \mathbf{z}(t)\text{)} \tag{41}$$

$$= \lim_{\varepsilon \to 0^+} \frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t+\varepsilon)\left(I + \varepsilon \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} + \mathcal{O}(\varepsilon^2)\right)}{\varepsilon} \tag{42}$$

$$= \lim_{\varepsilon \to 0^+} \frac{-\varepsilon \mathbf{a}(t+\varepsilon)\frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} + \mathcal{O}(\varepsilon^2)}{\varepsilon} \tag{43}$$

$$= \lim_{\varepsilon \to 0^+} -\mathbf{a}(t+\varepsilon)\frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} + \mathcal{O}(\varepsilon) \tag{44}$$

$$= -\mathbf{a}(t)\frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} \tag{45}$$

Gradients w.r.t θ

We can generalize (35) to obtain gradients with respect to θ–a constant wrt. t–and and the initial and end times, t_0 and t_N. We view θ and t as states with constant differential equations and write

$$\frac{\partial \theta(t)}{\partial t} = \mathbf{0} \qquad \frac{dt(t)}{dt} = 1 \tag{47}$$

We can then combine these with z to form an augmented state[1] with corresponding differential equation and adjoint state,

$$\frac{d}{dt}\begin{bmatrix}\mathbf{z}\\ \theta\\ t\end{bmatrix}(t) = f_{aug}([\mathbf{z}, \theta, t]) := \begin{bmatrix} f([\mathbf{z}, \theta, t]) \\ \mathbf{0} \\ 1 \end{bmatrix},\ \mathbf{a}_{aug} := \begin{bmatrix}\mathbf{a}\\ \mathbf{a}_\theta \\ \mathbf{a}_t\end{bmatrix},\ \mathbf{a}_\theta(t) := \frac{dL}{d\theta(t)},\ \mathbf{a}_t(t) := \frac{dL}{dt(t)} \tag{48}$$

Note this formulates the augmented ODE as an autonomous (time-invariant) ODE, but the derivations in the previous section still hold as this is a special case of a time-variant ODE. The Jacobian of f has the form

$$\frac{\partial f_{aug}}{\partial[\mathbf{z}, \theta, t]} = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{z}} & \frac{\partial f}{\partial \theta} & \frac{\partial f}{\partial t} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} (t) \tag{49}$$

where each $\mathbf{0}$ is a matrix of zeros with the appropriate dimensions. We plug this into (35) to obtain

$$\frac{d\mathbf{a}_{aug}(t)}{dt} = -\begin{bmatrix} \mathbf{a}(t) & \mathbf{a}_\theta(t) & \mathbf{a}_t(t) \end{bmatrix} \frac{\partial f_{aug}}{\partial[\mathbf{z}, \theta, t]}(t) = -\begin{bmatrix} \mathbf{a}\frac{\partial f}{\partial \mathbf{z}} & \mathbf{a}\frac{\partial f}{\partial \theta} & \mathbf{a}\frac{\partial f}{\partial t} \end{bmatrix}(t) \tag{50}$$

Full backprop algorithm

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

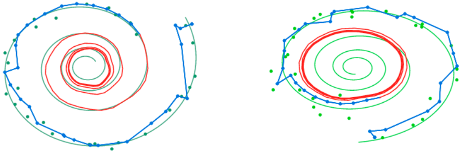
def aug_dynamics($[\mathbf{z}(t), \mathbf{a}(t), \cdot, \cdot], t, \theta$): ▷ Define dynamics on augmented state

  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

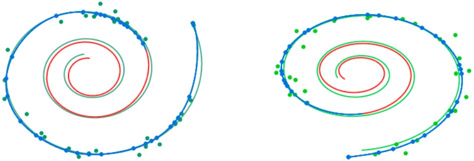
$[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial t_0}] = \text{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$ ▷ Solve reverse-time ODE

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

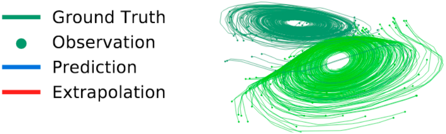
The End



(a) Recurrent Neural Network



(b) Latent Neural Ordinary Differential Equation



References

Neural Ordinary Differential Equations, Best Paper NIPS 2018. Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, David Duvenaud

DiffEqFlux.jl - differential equations with machine learning library.

Neural Ordinary Differential Equations, UofT Lecture Notes by Ricky T. Q. Chen

The idea of a dynamical system, Math Insight.