Reviewer : 김정학

<탄생배경>

다음의 residual network 를 살펴보자.

$$h_{t+1} = h_t + f(h_t, \theta_t)$$

이 식을 간단하게 되면 Euler method과 상당히 유사한 것을 확인할 수 있다.

만약 우리가 hidden layer을 무수히 많이 추가하면 어떻게 될까?

그러면 위 식의 $f(h_t, \theta_t)$을 h의 변화량으로 볼 수도 있다. 즉,

$$\frac{dh(t)}{dt} = f(h(t), t, \theta) \text{ 로 볼 수 있을 것이다.}$$

이는 만약 우리가 $h(T)$을 알고 싶으면 $h(0)$에 '더하기' 연산만 계속 해서 $h(T)$을 얻을 수 있다는 것을 의미한다.

<특징>

① memory efficiency

기존의 backtracking이 아닌 방식으로 학습을 진행하게 되는데/

이 방식은 또 다른 덧셈의 장점이라 forward에 대한 정보를

계속 가질 필요가 없어 메모리관점에서 효율적이다. ↓ 참고

https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/supervised_learning/backpropagation

(기존 딥러닝이 메모리 사용이 많은 이유)

② adaptive computation

forward 혹은 backward을 정밀을 계산을 할 것이다. 그런데 정밀을

얼마나 최소단위를 어떻게 할까? 매우 작은 수를 고정을 하는 것은 비효율적

이다. 따라서 정밀과정(함수)의 변화에 따라서 step-size을 정하는

(adaptive) 방식의 ODE solver을 사용한다 (e.g dopri)

③ Scalable and invertible normalizing flows

데이터가 function을 통해서 변할 때 분포의 변화를

계산하는 과정이 간단해진다. (정확하는 잘...)

④ Continuous time-series models

히든벡터가 연속적으로 구해질 수 있어서 시계열 모델에 적용이 될 수 있다.

< backward >

forward는 간단한 적분임을 알았다. 이 순간에서 주로 다루는 내용은 backward를 어떻게 할지이다.

우선 우리는 다음 식을 통해서 ODE layer을 통과한 결과를 알 수 있다.

$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta)\,dt$$

$$= \mathrm{ODESolver}(z(t_0), f, t_0, t_1, \theta)$$

(ODESolver: 적분기, $z(t_0)$: 초기값, f: ODEFunc, t_0, t_1: 적분구간, θ: parameters)

이로 Loss을 구하면

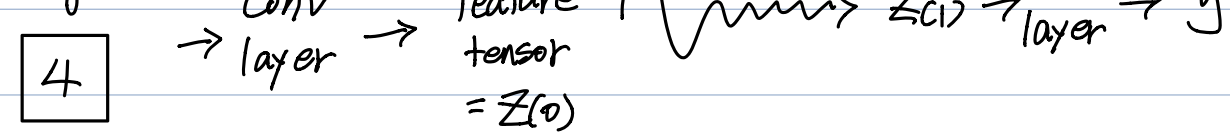
$$L(z(t_1)) = L\left(z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta)\,dt\right)$$

$$= L(\mathrm{ODESolver}(z(t_0), f, t_0, t_1, \theta))$$

여기서 우리가 학습하기 위해 필요한 미분값을 보자

우선 ODE network는 다음과 같다.

MNIST 예제



우리는 $z(0)$을 뽑아주는, 즉 feature을 뽑아주는 network (여기에서는 Conv layer)을 학습하기 위해

① $\frac{dL}{dz(0)}$ 가 필요하다. 그리고 ODE layer을 학습하기위해

② $\frac{dL}{d\theta}$ 가 필요하다.

이를 구하기 위해 adjoint $a(t) = \frac{dL}{dz(t)}$ 를 도입한다.

이를 수식으로 이리저리 해보면 (논문의 Appendix를 참고)

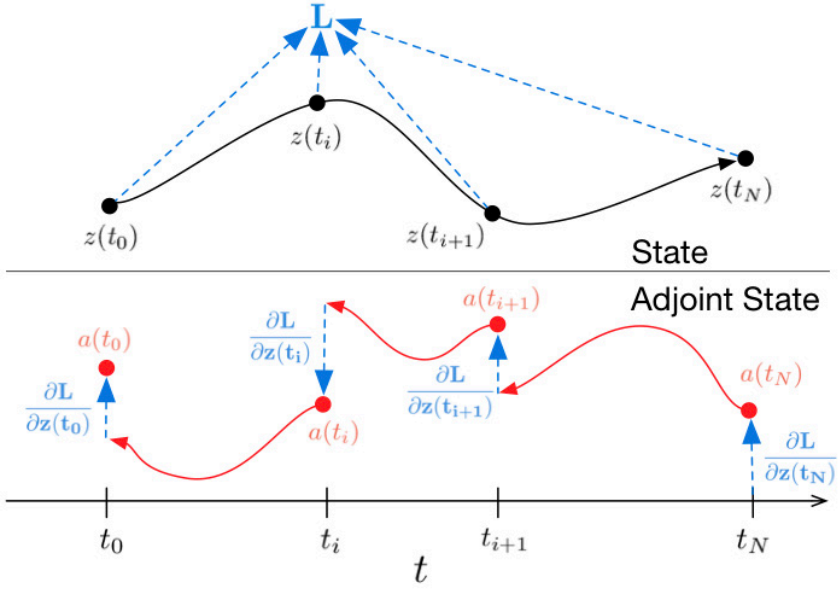
$\frac{da(t)}{dt} = -a(t)^T \frac{df(z(t), t, \theta)}{dz}$ 을 얻을 수 있다.

자 생각해보자 어라? ① $a(t_0) = \frac{dL}{dz(t_0)}$ 를 구하기 위해서

$\frac{da(t)}{dt}$ 을 이용해 또다른 ODESolver을 계산해야 된다.

그리고 또 한번 계산하면 ② $\frac{dL}{d\theta} = -\int_{t_1}^{t_0} a(t)^T \frac{df(z(t), t, \theta)}{d\theta} dt$

을 얻을 수 있다.



이는 다음을 나타낸 도식이다.

- State에서는 단순한 ODESolver을 계산한다.
- adjoint state에서는 augmented dynamic을 이용한다. 시간점 사이사이에서는 ODESolver로 계산하고 각 시간점에서 업데이트를 한다.

이제가 backpropagation을 위한 알고리즘이다.

Algorithm 1 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 $\partial L / \partial \mathbf{z}(t_1)$

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

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

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

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

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

초기상태

feature를 뽑아내는 network 학습

ODE layer 학습.

역방향으로 적분.