# *Neural Ordinary Differential Equations.*

Neural Ordinary Differential Equations.pdf

https://seewoo5.tistory.com/12

https://kimjy99.github.io/논문리뷰/neural-ode/

https://kubig-2022-2.tistory.com/45

https://blog.naver.com/qbxlvnf11/222164341856

## Abstract


## Abstract

이 논문은 새로운 유형의 심층 신경망 모델을 소개한다. hidden layers의 이산적인 순서를 지정하지 않는 대신 hidden state의 derivative을 parameterize한다. 이러한 연속적인 깊이를 가지는 모델은 다음과 같은 장점이 있다.

- 일정한 메모리 비용 (상수값)
- 입력에 맞게 평가 전략을 조정
- 속도를 위한 정밀도 - 속도 trade-off

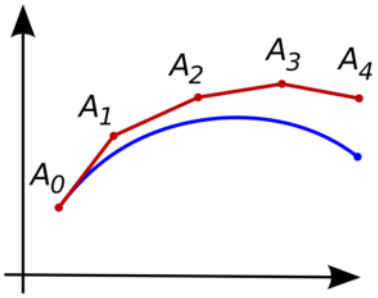

---

## 1. Introduction

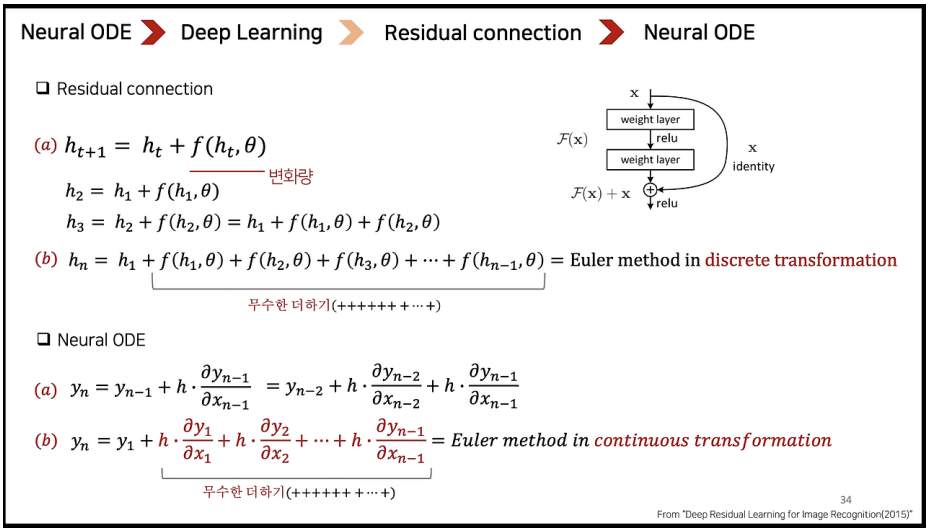
ResNet, RNN 디코더, normalizing flow와 같은 기존 모델들은 여러 개의 hidden layers를 쌓아서 반복적인 업데이트 (discrete) 로 변환한다.

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

이 반복적인 업데이트 (ResNet의 residual connection)는 연속적인 변환의 오일러 방법(Euler's method)과 유사하다.

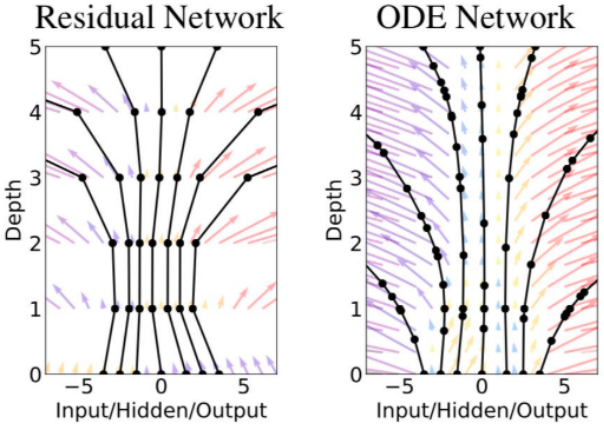



Neural ODE는 ODE solver인 오일러 방법(Euler's method)과 ResNet의 residual connection이 많은 변화량의 더하기로 구성된다는 점에서 유사하다는 것을 착안하여 고안되었다.




그렇다면 더 많은 레이어를 추가하고 더 작은 step을 수행하면 어떻게 될까?

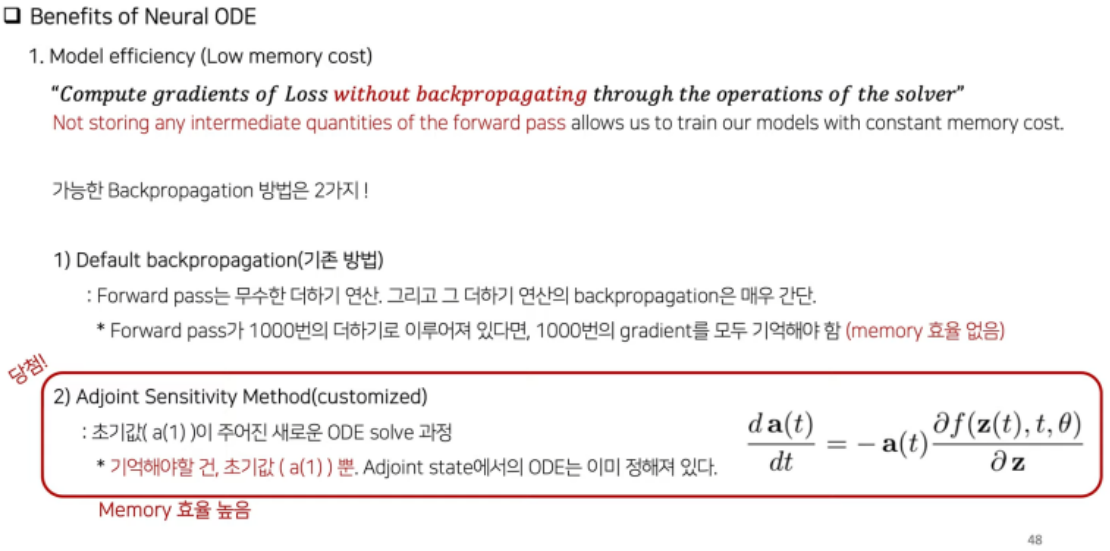
→ residual connection은 ResNet50/152 등 레이어 수로 제한되어 있고, Neural ODE의 경우 오일러 방법(Euler's method)을 이용하기 때문에 이론상 스텝 사이즈 h를 작게 둠으로써 infinite한 레이어를 만들 수 있다. 즉, discrete transformation에서 continuous transformation으로 변경이 가능하다.



ODE solver를 사용하여 모델을 정의하고 평가하면 다음과 같은 이점이 있다

.

1. **메모리 효율성(2챕터):** 본 논문은 solver의 연산을 통해 역전파하지 않고 ODE solver의 모든 입력에 대해 스칼라 값 loss의 기울기를 계산하는 방법을 제시하였다. Forward pass의 중간 값들을 저장하지 않기 때문에 모델 깊이와 상관없이 일정한 메모리 비용으로 모델을 학습시킬 수 있다.




→ Backpropagation 없이 operations of the solver를 통해 loss의 gradients 계산이 가능하다.

모든 forward 과정에서 구한 gradient를 저장하고, chain rule로 풀어주는 과정이 별도로 없기 때문이다.

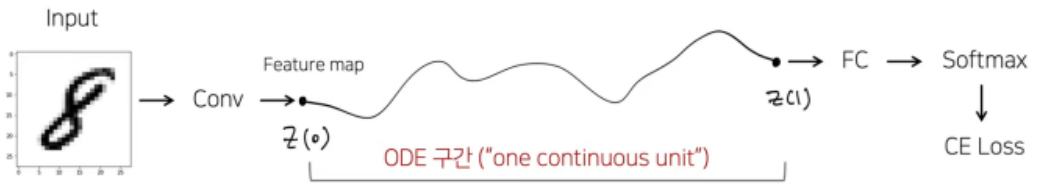
2. **적응형 계산:** Euler's method는 아마도 ODE를 푸는 가장 간단한 방법일 것이다. 그 이후로 효율적이고 정확한 ODE solver가 개발된 지 120년이 넘었다. 최신 ODE solver는 근사 오차의 증가를 보장하고, 오차 수준을 모니터링하며, 평가 전략을 즉시 조정하여 요청된 정확도 수준을 달성한다. 이를 통해 문제의 복잡도에 따라 모델 평가 비용을 확장할 수 있다.
3. **변환가능하고 가역적인 normalizing flow(4챕터):** 연속적인 변환의 예상치 못한 부수적 이점은 change of variables 수식을 계산하기가 더 쉬워진다는 것이다. 이를 통해 normalizing flow의 bottleneck을 피하고 maximum likelihood에 의해 직접 학습할 수 있는 새로운 클래스의 가역 밀도 모델을 구성할 수 있다.
4. **연속적인 시계열 모델(5챕터):** 이산적인 관찰 및 방출 간격이 필요한 RNN과 달리 연속적으로 정의된 역학은 임의의 시간에 도착하는 데이터를 자연스럽게 통합할 수 있다.

---

## 2. Reverse-mode automatic differentiation of ODE solutions

연속적인 깊이의 신경망을 학습할 때 가장 큰 기술적 어려움은 ODE solver를 통해 역방향 미분(역전파)을 수행하는 것이다. Forward pass의 연산을 따라서 다시 역방향 미분은 간단하지만 메모리 비용이 많이 들고 추가적인 수치적 오차가 발생한다.

❑ Neural ODE in supervised-learning (MNIST digit)




❑ ODE 구간 역할 = 변화량을 구하다 = 적분을 풀다

$$Z(1) = Z(0) + \int_0^1 f(z(t), t\,;\,\theta)\,dt$$

Z의 변화량 * Z : ODE의 state = ODE의 hidden vector

Solved by Euler method, Runge kutta, etc.

40

*** ODE 구간만 주의해서 어떻게 처리 되는지 확인하면 된다. ***

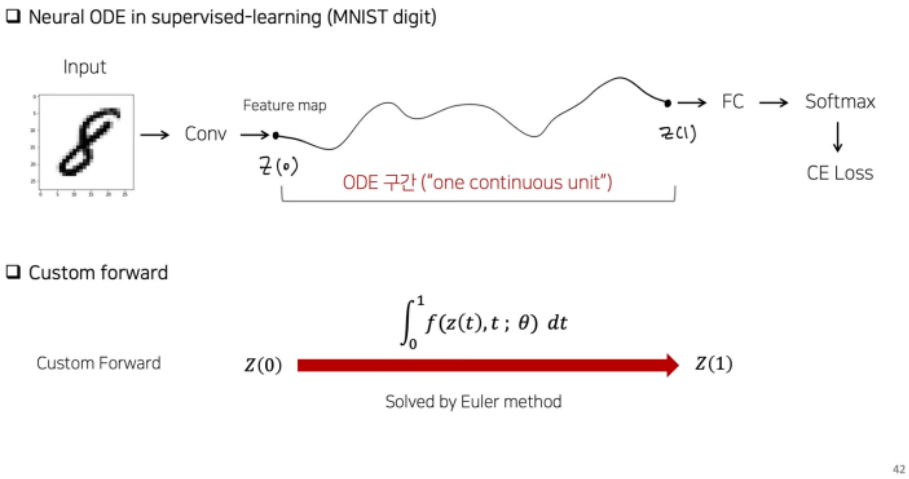
❑ Neural ODE in supervised-learning (MNIST digit)


❑ Custom forward, backward(?)

| | |
|---|---|
| Custom Forward | Euler method |
| Custom Backward | Adjoint Sensitivity method |

41

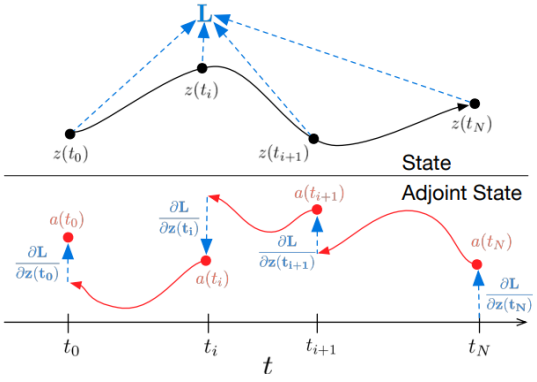
Neural ODE의 Forward에서는 다음과 같이 기존 forward 프로세스가 진행되고 ODE 구간에서만 initial state에서 final state까지 변화량을 더해주는 Euler method을 통해 적분해주면 된다.




그렇다면 역전파 과정은 어떻게 진행을 할까?

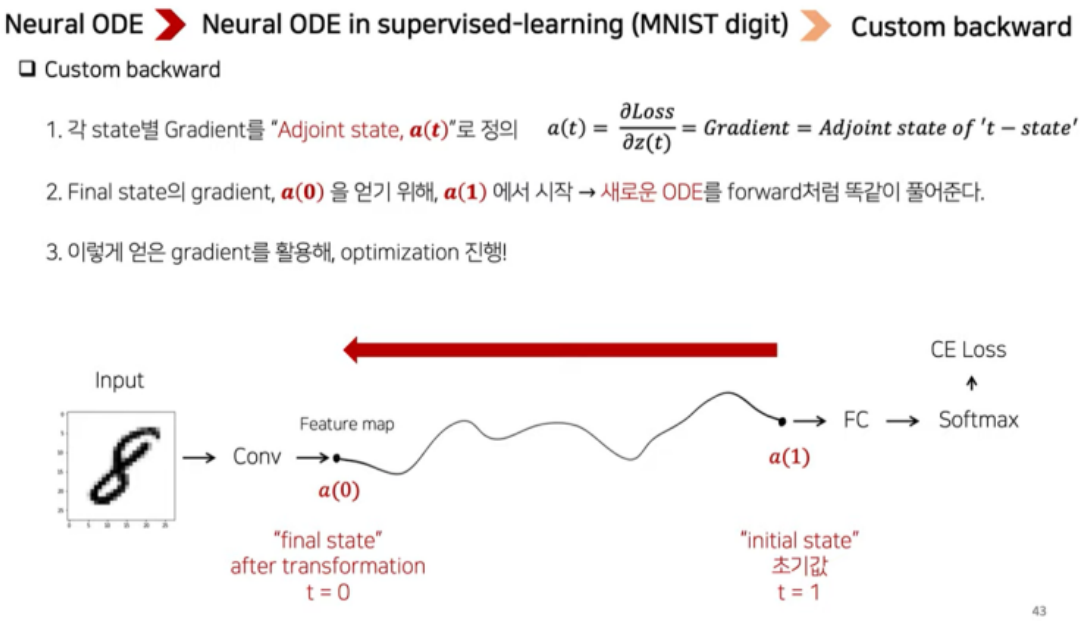
ODE solver를 블랙박스로 취급하고 adjoint sensitivity method (본질적으로 Euler method와 동일)를 사용하여 기울기를 계산한다. 이 접근 방식은 새로운 두 번째 ODE를 시간에 대하여 거꾸로 풀어서 기울기를 계산하며 모든 ODE solver에 적용 가능하다. 또한, 문제 크기에 따라 선형적으로 확장되고 메모리 비용이 낮으며 수치적 오차를 명시적으로 제어한다.

먼저 역전파 미분 과정을 전체적으로 살펴보자면, 각 state(initial state와 final state 사이의 continuous한 state들)의 gradient를 **adjoint state** a(t)로 정의한다.




그 후에는 a(1)이 주어졌을 때 얻은 a(0) 즉, forward 기준으로 initial state가 되는 gradient을 구한다.

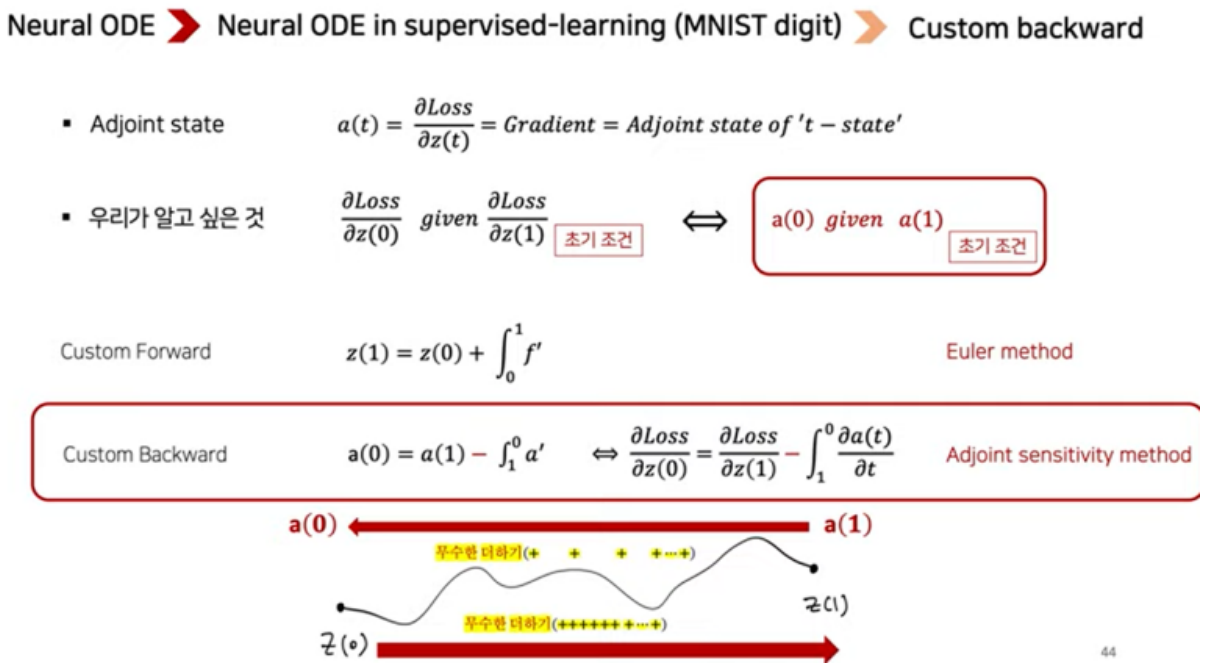
다음에는 이렇게 얻은 gradient로 optimization을 진행한다.

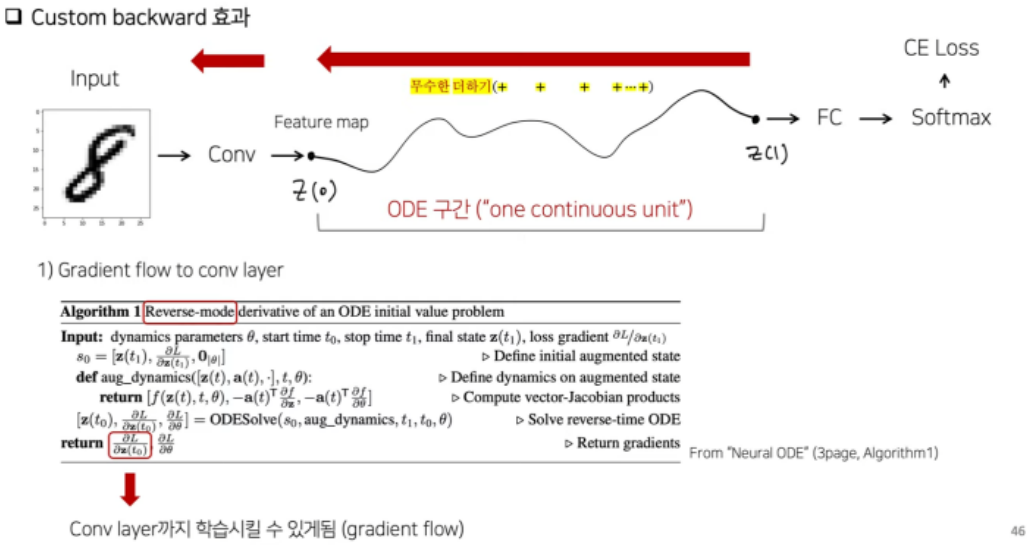



.

Backward에서 구하는 gradient는 ODE solver인 **adjoint sensitivity method**를 통해 구해진다.

Backward이기 때문에 1에서 0으로 방향을 반대로 가기 위하여 부호가 마이너스로 바뀐다.






Algorithm 1은 필요한 역학을 구성하고 ODE solver를 call하여 모든 기울기를 한 번에 계산하는 방법을 보여준다.

이렇게 adjoint sensitivity method는 final value z(t_1)로 부터 시작하여 adjoint와 함께 z(t)를 적절한 시간 안으로 간단히 재계산할 수 있다.

즉, adjoint sensitivity method의 경우 ODE와 초기값만 기억하면 되기 때문에 다음과 같이 메모리 효율성이 O(1)이 된다.

---

## 3. Replacing residual networks with ODEs for supervised learning

**지도학습을 위한 neural ODE 훈련 알아보기**

Software

- Implicit Adams Method
  - ODE 초기값 문제 해결 위해 사용된다.
  - LSODE, VODE ODE solver에서 구현되고 spicy.integrate 패키지를 통해 호출할 수 있다.
  - Runge-Kutta Explicit method vs. implicit Adams method
    - Runge-Kutta Explicit: 현재 시점 함수값들로 다음 시점의 상태 계산. 이때 비선형 방정식을 직접 푸는 것은 아니지만, 비선형 ODE의 해를 계산함으로써 비선형 문제를 해결할 수 있음
    - implicit Adams method: 다음 시점 상태를 계산하기 위해 비선형 방정식을 직접 풀어야 함. explicit method보다 더 안정적이나, 매 단계에서 비선형 최적화 문제 풀어야 해 계산 비용 높음
- adjoint sensitivity Method
  - implicit method → ODE solver가 직접 backpropagation 수행 시  계산 복잡, 메모리 사용량 등 문제 발생

- 위 문제를 해결하기 위해 adjoint variable 사용해 시간 역방향으로 역전파 수행한다.
  - adjoint variable: 미분 방정식 사용해 상태 변화 모델링할 때, 손실 함수와 관련된 민감도 계산하기 위해 도입되는 변수로, 역전파를 효율적으로 수행하기 위해 사용됨
- 본 실험에서는 hidden state dynamics 에 대한 미분을 평가하기 위해 autograd 프레임워크로 구현했다.

Model Architectures

- ODE-Net
  - ResNet에서 사용된 6개의 Residual Block을 ODE Solver로 대체한다.
    - input을 두번 downsampling 한 후 6개의 standard residual block을 적용하는데, 이때 residual block 대신 ODE 솔버 사용
- RK-Net
  - Runge-Kutta 통합기를 통해 역전파를 직접 수행한다.

Table 1: Performance on MNIST. †From LeCun et al. (1998).

| | Test Error | # Params | Memory | Time |
|---|---|---|---|---|
| 1-Layer MLP† | 1.60% | 0.24 M | - | - |
| ResNet | 0.41% | 0.60 M | $\mathcal{O}(L)$ | $\mathcal{O}(L)$ |
| RK-Net | 0.47% | 0.22 M | $\mathcal{O}(\tilde{L})$ | $\mathcal{O}(\tilde{L})$ |
| ODE-Net | 0.42% | 0.22 M | $\mathcal{O}(1)$ | $\mathcal{O}(\tilde{L})$ |

L: Resnet에서 사용된 layer 수
L hat: ODE solver가 single forward pass 동안 요청하는 함수 평가 수 (layer 수로 해석 가능)

⇒ ODE-Nets & RK-Nets가 ResNet과 동일한 성능 달성.
파라미터 1/3 수준, 메모리 효율성 O(1) 달성

Error Control in ODE-Nets

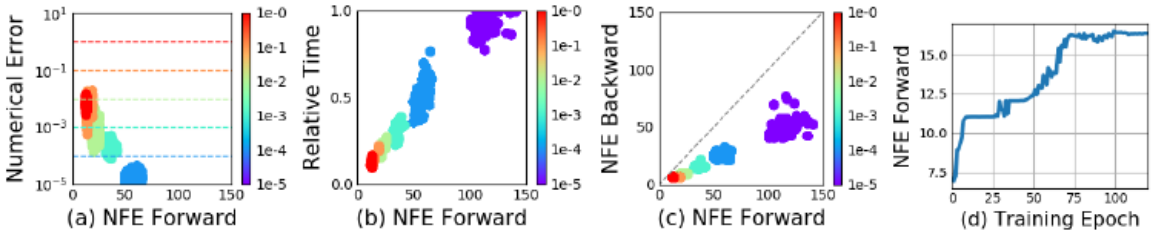



Figure 3: Statistics of a trained ODE-Net. (NFE = number of function evaluations.)

- Figure 3 (a)
  - ODE solver가 오류 제어가 가능함을 보여준다.
  - 허용 오차 조정 시 정확도를 변화한다.

- Figure 3 (b)
  - 순방향에 소요되는 시간은 함수 평가 횟수와 비례한다.
  - 허용 오차 조정해서 정확도 vs. 계산 비용 간 trade-off 가 존재한다.
- Figure 3 (c)
  - 역방향 계산 시 순방향보다 함수 평가가 절반 정도만 필요하다. → adjoint sensitivity가 memory efficient + computationally efficient
- Figure 3 (d)
  - 학습 진행될수록 함수 평가 횟수가 증가한다.
  - 즉, 모델의 복잡도 증가에 따라 계산량도 증가한다.

Network Depth

- ODE 솔루션에서 네트워크의 깊이(≈층)에 대한 정의는 불분명하다.
- 하지만 관련된 개념으로 HIDDEN STATE DYNAMICS를 평가하기 위한 함수 평가 횟수가 존재한다.
- 훈련이 진행될수록 모델 복잡성 증가하고 함수 평가의 수가 증가하는 것이다.

---

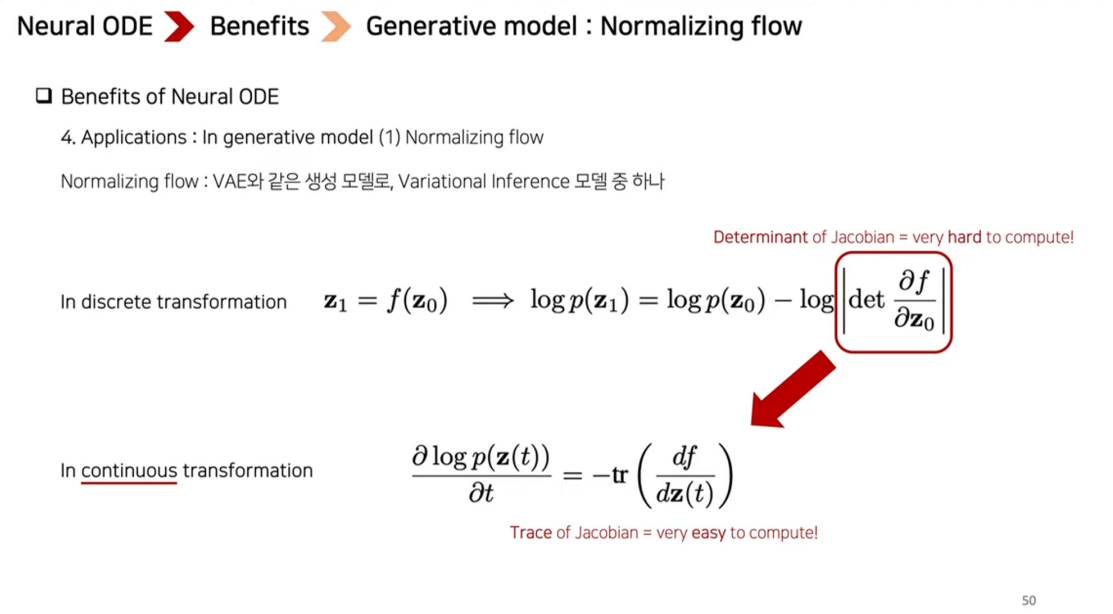
## 4. Continuous Normalizing Flows

이산화된 방정식은 normalizing flow와 NICE에도 나타나는데, 샘플들이 전단사 함수(일대일 대응 함수)를 통해 변환될 경우 change of variables theorem을 사용해 확률의 정확한 변화를 계산할 수 있다.

$$\mathbf{z}_1 = f(\mathbf{z}_0) \implies \log p(\mathbf{z}_1) = \log p(\mathbf{z}_0) - \log \left| \det \frac{\partial f}{\partial \mathbf{z}_0} \right|$$

change of variables의 한계는 비용 문제다.

- Jacobian의 determinant 계산이, hidden unit 수의 세제곱에 비례한 비용을 유발한다.
- 기존 NF의 연산량이 많은 부분인 Determinant of Jacobian가 Trace of Jacobian으로 변경되어 linear cost로 변경된다.




Theorem 1 (Instantaneous Change of Variables)

- 이산 layer 집합 → 연속 변환 시 nomalizing 상수 변화 계산이 단순화된다.

Using multiple hidden units with linear cost

- 선형 trace를 이용해 여러 개의 hidden unit을 사용하는 dynamics를 정의할 수 있다.
- linear cost로 평가가 가능하다. (↔ cubic cost에 비해 효율성 증대)
- wide flow model(폭 넓음 = 많은 hidden unit 포함하는 layer) 효율적으로 평가할 수 있다.

Time-dependent dynamics

- 시간에 따라 parameter가 변화한다.
- hypernetwork(network의 parameter 자체를 학습 가능)와 유사
- gating 매커니즘 도입: 특정 시간에 어떤 dynamics가 활성화될지 결정

⇒ 이러한 모델을 CNF(continuous normalizing flows)라 부른다.

## 4.1 Experiments with Continuous Normalizing Flows

연속 / 이산 평면 flow의 알려진 분포에서 추출한 샘플에 대한 학습을 비교했다.

M개의 은닉 유닛을 사용하는 planar CNF가, M개의 층을 사용하는 Planar NF와 동일하거나 혹은 더 높은 표현력을 가질 수 있다.

**Experiments with CNF : 1.** Density matching

CNF 모델 vs. NF 모델, target 분포에서 샘플 생성 능력을 비교하기 위해 수행된 실험이다.

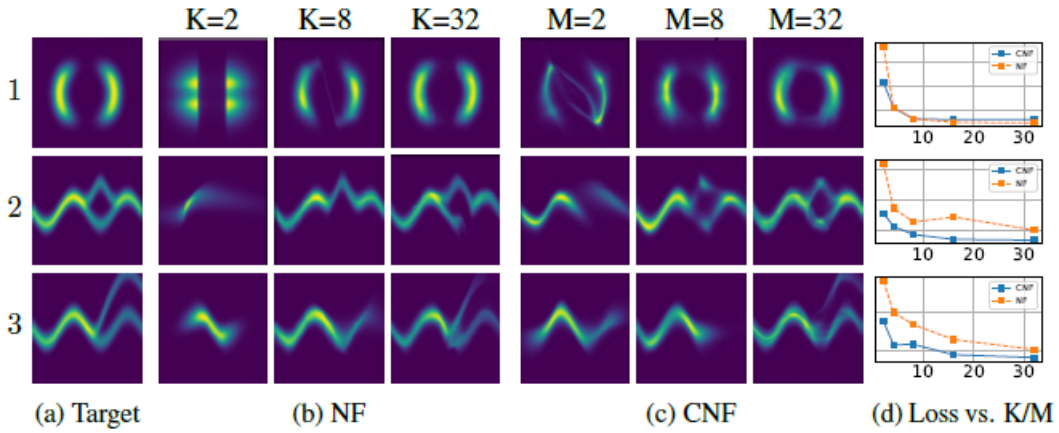
설정

CNF 모델: 10,000 반복 + Adam optimizer
NF 모델: 500,000 반복 + RMSprop optimizer
Loss function: KL 발산 최소화

- KL Divergence:  사후 분포와 사전 분포가 얼마나 다른지를 측정하는 방법

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

결과: CNF 손실 < NF 손실 ⇒ CNF가 효과적이다.




**Experiments with CNF : 2. Maximum Likelihood Training**

CNF VS. N, 밀도 추정 성능 평가를 위한 실험이다.

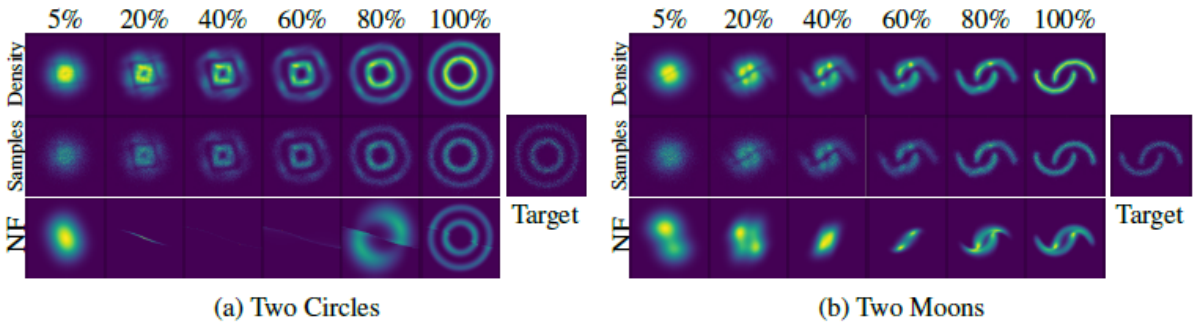
순방향 pass와 거의 동일한 비용으로 역변환 계산이 가능하다. (↔ NF에서 불가)

실험 설정
CNF: hidden units 64개
NF: 64개의 stackked된 one-hidden-unit으로 구성된 layer 사용

결과

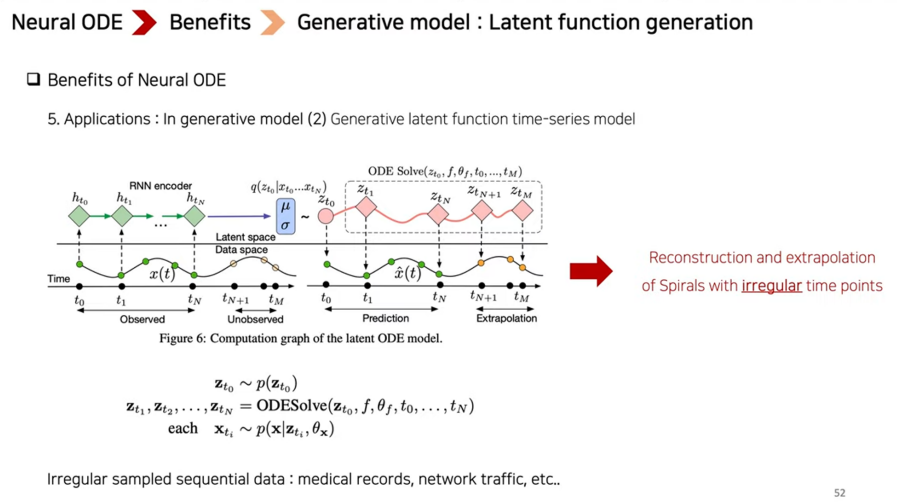
CNF는 학습된 변환을 통해 초기 planar flow를 회전시켜 입자를 원형으로 균등하게 퍼뜨린다.
⇒ smooth, interpretable

NF는 비직관적이고, 학습이 어렵다.




## 5. A generative latent function time-series model

- 논문에서는 RNN을 encoder로, Neural ODE를 decoder로 가지는 VAE를 만든 뒤 이를 이용해서 generative model을 만들었다.
  - 관측시간 t0, ...., tN과 초기상태 zt0이 주어지면 ODE solver는 zt1, zt2, .....,ztN을 생성

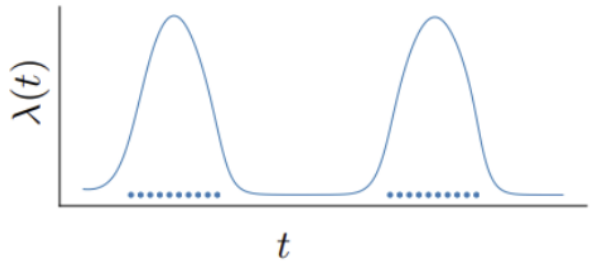



- 기존 RNN의 경우 discrete 하게 구성이 되어 있기 때문에, regular 한 입력이 들어가야만 성능이 잘 나온다.
- 반면, 위 이미지에서의 generative model은 continuous한 특성덕분에 irregular 한 입력이 들어가도 전체 함수를 근사해나갈수 있기 때문에 ( = 그 입력 데이터의 잠재함수를 더욱더 잘 찾아낼수 있기 때문에) 기존 성능이 잘 유지된다.

**Poisson Process**

: 일정 시간 동안 특정 사건일 발생하는 횟수를 모델링하는 데 사용되는 확률 과정

$$\log p(t_1, \ldots, t_N \mid t_{\text{start}}, t_{\text{end}}) = \sum_{i=1}^{N} \log \lambda(z(t_i)) - \int_{t_{\text{start}}}^{t_{\text{end}}} \lambda(z(t))dt$$

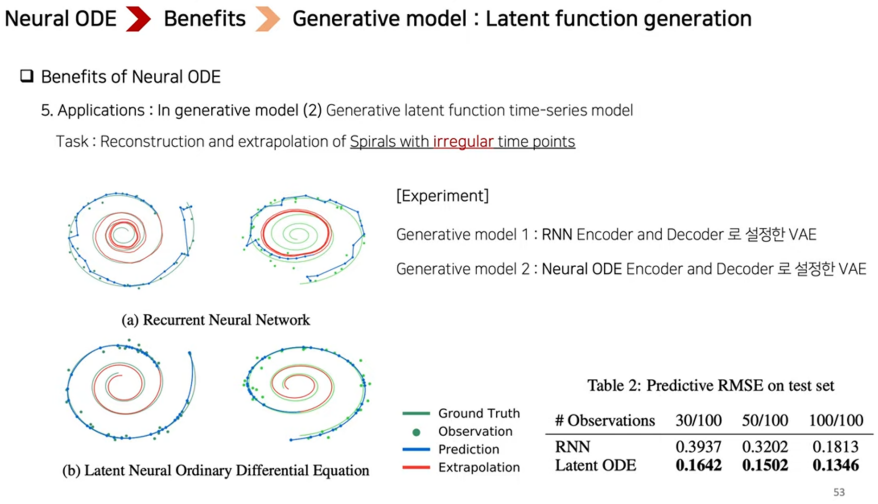



- 특정 시간 범위 내에서 사건이 발생할 확률을 추정함. → λ(z(t)) 학습
- ODE 솔버는 잠재 상태의 궤적을 계산하는 데 사용되며, **Poisson Process Likelihood**는 그 궤적에 따른 이벤트 발생 가능성을 평가하는 데 사용.

⇒ 불규칙한 시계열 데이터를 예측하는 데 유용함, 이를 통해 모델이 모든 관찰 시간과 이벤트 발생 시간을 효과적으로 모델링

- 이 결과는 Extrapolation (시계열 상 관측된 포인트들을 너머의 시점까지 정확히 생성해내는지) 실험으로 자세히 알 수 있다.
- 실험과정
  - 실험은 spiral 모양의 해를 가지는 간단한 ODE를 만든 뒤, 이 ODE의 초기 timestamp에서의 몇개의 값을 바탕으로 이후의 값을 예측하는 task 수행
- 실험 결과
  - discrete하게 모델링을 하는 RNN에 비해서 Neural ODE의 결과가 노이즈 없이 잘 진행이 되었고 RMSE 값이 더 작은 것을 확인하였다.

즉, latent space(잠재 공간 [차원축소에서 자주 나온 그 개념])에 대한 모델링이 매우 잘 되어 있다. ( = latent function을 정확하게 해석해냈다)



## 6. Scope and Limitations

1. **미니배치 (Minibatching)**: Mini-batch의 사용이 일반적인 신경망보다 덜 간단하다. 경우에 따라서는, 모든 batch 요소의 오차를 함께 제어하려면 이렇게 결합된 시스템이 각 시스템을 개별적으로 해결하는 경우보다 K 배 더 평가해야 할 수도 있다.그러나 실제로는 (in practice) mini-batch를 사용하면 평가 횟수가 크게 증가하지 않는다
2. **유일성 (Uniqueness)**
3. **오차 허용치 설정 (Setting tolerances)**: 사용자가 속도와 정밀도를 절충할 수 있지만 학습 중에 forward 및 backward pass 모두에서 오차 허용치를 선택해야 한다.
4. **Forward 궤적 재구성 (Reconstructing forward trajectories)**: 역학을 거꾸로 실행하여 상태 궤적을 재구성하면 재구성된 궤적이 원본에서 벗어나는 경우 수치적 오차가 추가로 발생할 수 있다. 이 문제는 체크포인트를 통해 해결될 수 있다.