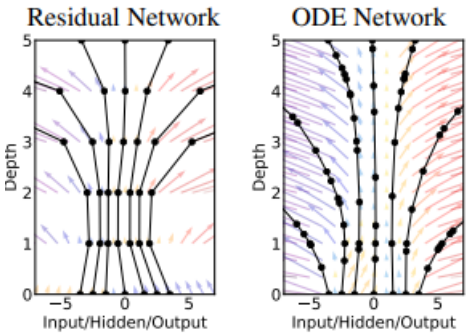
Neural Ordinary Differential Equations

1. Introduction

ResNet, RNN 디코더, normalizing flow 등과 같은 모델들은 변환의 시퀀스를 hidden state로 구성하여 복잡한 변환을 만들어낸다. 그리고 이러한 반복적인 변환은 함수를 아주 작은 step으로 쪼개고 현재 값과 기울기를 이용하여 다음 값을 예측하는 Euler discretization(Euler's method)을 적용할 수 있는데, 이렇게 여러 개의 레이어와 작은 스텝들이 극한까지 가게되면 hidden unit의 연속적인 dynamics를 ODE(상미분 방정식)으로 표현할 수 있게된다.

input layer h(0)에서 시작하여 output layer h(T)를 time step T에서의 ODE의 초기치 문제의 해로서 정의할 수 있다. 원하는 정확도를 맞추기 위해 방정식의 해를 결정할 때마다 hidden unit의 dynamics f를 평가하게 되는데 앞선 값은 이때 사용되는 black-box 미분방정식(ODE solver)의 해로서 계산된다.

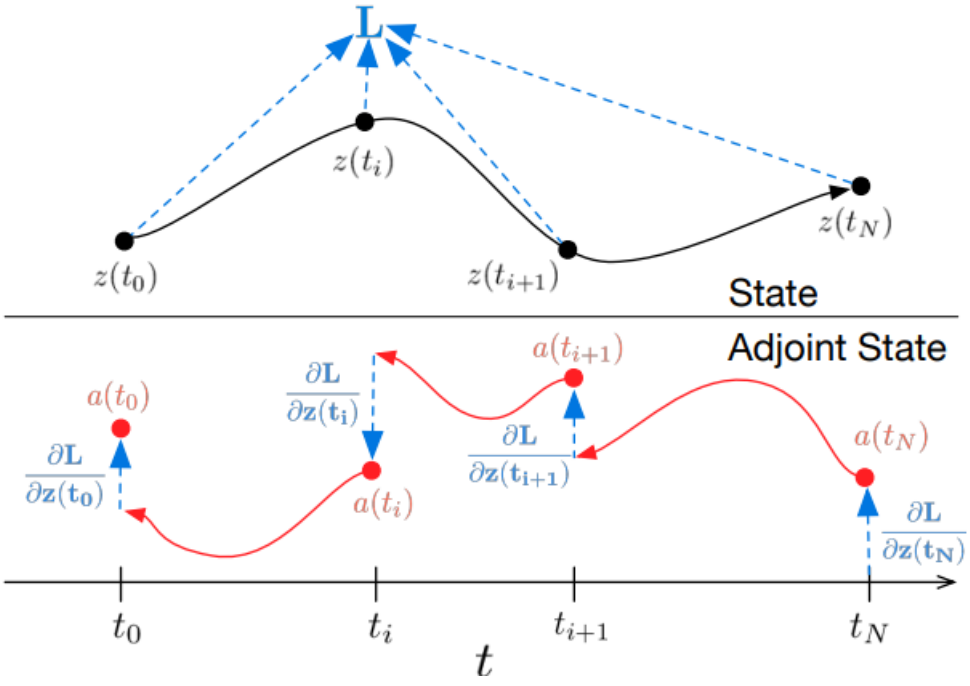


<ODE의 장점>

1. **메모리 효율성 Memory Efficiency** - ODE solver는 모든 입력에 대해 scalar-valued loss의 기울기를 계산하기 때문에 역전파를 사용하지 않는다. 따라서 순전파의 중간 계산들을 메모리에 저장하지 않고 일정한 메모리로 필요한 값들만 저장할 수 있다.
2. **적응형계산 Adaptive Computation** - 최신 ODE solver는 높은 수준의 오차의 근사와 오차수준의 모니터링을 해내고 전략을 즉시 조정하여 원하는 정확도를 얻어낸다. 이것은 모델의 평가비용이 모델의 복잡성과 연결되도록 한다. 학습 후에는 정확도는 실시간으로 낮아질 수 있다.
3. **스케일 가능하고 가역적인 normalizing flows** - change of variables의 식의 계산이 더욱 쉬워지고, normalizing flow의 bottleneck 현상(병목현상-데이터처리능력의 부족)을 피함과 동시에 최대우도로 직접 학습할 수 있는 새로운 클래스의 가역 밀도 모델을 구성할 수 있게된다.
4. **연속적인 시계열 모델** - 이산적인 관찰 및 방출 간격이 필요한 RNN과 달리, 연속적으로 정의된 dynamics는 임의의 시간에 도달하는 데이터를 통합할 수 있다.

2. Reverse-mode automatic differentiation(backpropagation) of ODE solutions

ODE solver의 순전파-역전파 계산이 많은 메모리 비용과 오차를 발생시킨다는 문제점을 해결하기 위해 ODE solver를 black box로 생각하는 방법이 고안된다. 이때 기울기는 adjoint sensitivity method를 사용하여 계산되는데 이 접근법은 다음 ODE를 시간에 대해 거꾸로 기울기 계산을 행하며 모든 ODE solver에 적용이 가능하다는 특징을 갖는다. 이는 문제의 복잡도에 따라 선형적으로 확장될 수 있고 메모리 비용과 오차를 제어할 수 있다.



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

$$\frac{da(t)}{dt} = -a(t)^\top \frac{\partial f(z(t), t, \theta)}{\partial z}$$

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

스칼라 값인 loss function L을 최적화하기 위해 (1)과 같은 식을 정의했고, 이를 구하기 위해선 theta에 대한 기울기가 필요하다.

1. loss의 기울기가 각 순간의 hidden state z(t)에 얼마나 의존하는지를 결정한다.
   ⇒ adjoint :
   $a(t) = \partial L / \partial z(t)$

   ⇒ adjoint의 dynamics는 chain rule의 순간적인 변화량으로 생각할 수 있는 다른 ODE에 의해 제공된다
2. a(t0)를 구하기 위해 a(t1)을 이용할 수 있는데, 이때 $a(t1) = \partial L / \partial z(t1)$의 초기값부터 시작하여 backward로 실행된다.
   * ODE를 풀기위해선 전체 궤적을 따라서 z(t)의 값을 알아야 하지만, 최종값을 z(t1)으로 설정하여 adjoint 계산으로 시간에 대해 backward를 실행한다.
3. theta에 대한 기울기를 계산하기 위해 z(t)와 a(t)를 포함하는 적분을 실행한다.

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

- z와 a를 풀기위한 모든 적분 계산은 state, adjoint, 편도함수를 하나의 벡터로 concatenate한 뒤 ODE에 대해 계산한다.
- Loss가 중간 state에서 달라지더라도 역전파 과정은 각각의 연속적인 출력 시간 쌍(time step pair) 사이에서 별도의 solve 시퀀스로 나뉘게 된다.

3. Replacing residual networks with ODEs for supervised learning

ODE의 초기값 문제를 풀기위해 Adams Method를 사용한다.

<MNIST 성능 비교>

	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	$\mathcal{O}(1)$	$\mathcal{O}(\tilde{L})$

ResNet - input을 두 번 다운샘플링하고 6개의 standard residual block을 적용
ODE-Net - 같은 모델 구성에 ODE solver을 사용 *
implicit method

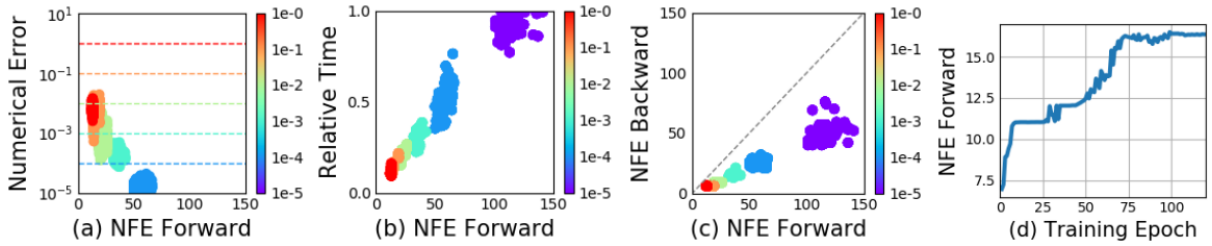
RK-Net - 같은 모델 구성에 역전파 과정에서 Runge-Kutta integrator을 사용 ***explicit method**

*implicit method : 음해법 (현재 알고 있는 시스템과 미래의 시스템의 상태로부터 미래 시간의 상태를 계산하는 방법) $y_{k+1} = y_k + f(y_{k+1}, t_{k+1})$

*explicit method : 양해법 (현재 알고 있는 시스템의 상태로부터 미래의 시스템의 상태를 계산하는 방법) $y_{k+1} = y_k + f(y_k, t_k)$ ⇒ 기울기가 직접 역전파됨

⇒ RK와 ODE 모두 ResNet과 비슷한 성능을 보임

<Error Control in ODE-Nets>



(a) - forward pass의 함수평가횟수(NFE)가 늘어날 수록 오차가 제어됨

(b) - NFE에 따라 forward call의 상대적인 시간이 증가함 ⇒ 시간비용과 정확도의 tradeoff

(c) - backward pass의 함수평가횟수가 forward pass의 함수평가횟수의 반으로 줄어듦 ⇒ direct한 역전파보다 계산적 효율성을 가짐

4. Continuous Normalizing Flows

> CNF의 기본 아이디어와 장점, 그리고 실험 결과에 대한 소개

이산화된 방정식 $ht + 1 = ht + f(ht, \theta t)$ 은 normalizing flows와 NICE framework에서도 존재한다. 두 방법은 샘플이 전단사 함수를 통해 변환될 때 the change of variables theorem를 사용하여 확률의 변화를 정확하게 계산한다.

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

ex) the planar normalizing flow

$$\mathbf{z}(t+1) = \mathbf{z}(t) + uh(w^\mathsf{T}\mathbf{z}(t) + b), \quad \log p(\mathbf{z}(t+1)) = \log p(\mathbf{z}(t)) - \log \left| 1 + u^\mathsf{T} \frac{\partial h}{\partial \mathbf{z}} \right|$$

- **<change of variables 공식에서 한계점>**

일반적으로 Jacobian $\partial f / \partial z$ 의 determinant을 계산하는 것이며. 계산 비용은 z의 차원에서 hidden unit수의 세제곱에 비례한다. 최근연구에서는 normalizing flow layer의 표현력과 비용간의 trade-off를 연구하고있다.

- **<개선 방법>**

이산적인 레이어의 집합에서 연속 변환으로 이동하면 normalizing 상수의 변화 계산이 단순화된다.

$z(t)$를 시간에 따른 확률 $p(z(t))$를 갖는 유한하고 연속적인 확률 변수라고 하자. 그리고$dz/dt = f(z(t), t)$ 를 $z(t)$의 시간에 대해 연속적인 변환을 설명하는 미분 방정식이라고 하자. f가 z에서 균일하게 Lipschitz 연속이고 t에서 연속이라고 가정하면, 로그 확률의 변화는 다음과 같은 미분 방정식을 따른다.

$$\frac{\partial \log p(z(t))}{\partial t} = -\mathrm{tr}\left(\frac{df}{dz(t)} \right)$$

- **<개선점>**

1. Jacobian의 dterminant 계산이 대각합연산으로 대체되었고, 미분방정식 f는 전단사일 필요가 없으므로 매우 단순화 되었다.
2. 초기 분포 $p(z(0))$가 주어지면 $p(z(t))$에서 샘플링하고 이 결합된 ODE를 풀어 밀도를 평가할 수 있다.

Using multiple hidden units with linear cost

Determinant는 선형 함수가 아니지만 trace 함수는 선형 함수이다.

⇒ $tr(\sum_n Jn) = \sum_n tr(Jn)$

따라서 역학이 함수의 합으로 주어지면 로그 밀도에 대한 미분 방정식도 합이 된다.

$$\frac{dz(t)}{dt} = \sum_{n=1}^{M} f_n(z(t)) \quad \Longrightarrow \quad \frac{d\log p(z(t))}{dt} = \sum_{n=1}^{M} \mathrm{tr}\left(\frac{df_n}{dz(t)}\right)$$

이는 hidden unit이 많은 flow model을 낮은 비용으로 평가할 수 있음을 의미하며 비용은 hidden unit 수 MM에 선형이다. 표준 normalizing flow를 사용하여 이러한 '넓은' flow layer를 평가하는 데는 O(M3의 비용이 든다. 이는 표준 normalizing flow 아키텍처가 하나의 hidden unit의 여러 레이어를 사용한다는 것을 의미한다.

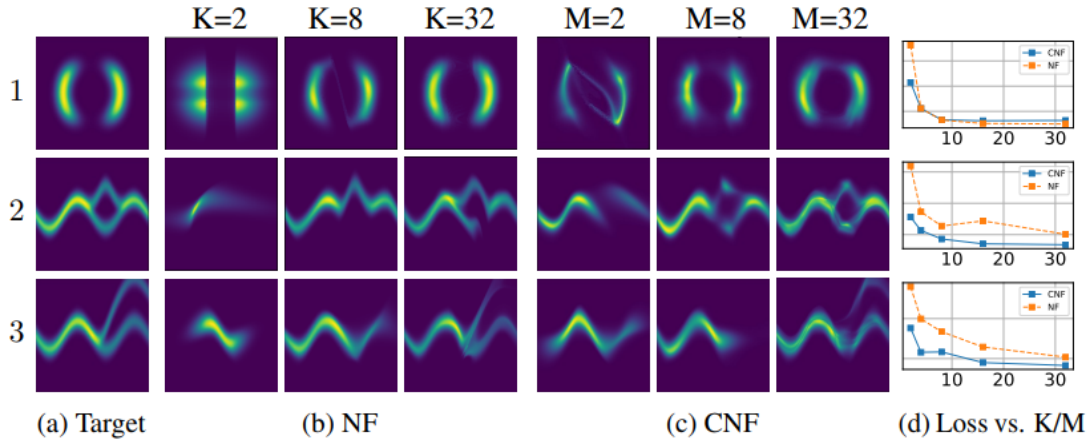
Time-dependent dynamics[Permalink]

저자들은 flow의 파라미터를 t의 함수로 지정하여 미분 방정식 $f(z(t), t)$를 t에 따라 변경할 수 있도록 하였다. 이는 일종의 hypernetwork이다.

또한 각 hidden unit에 대한 gating 메커니즘인 $dz/dt = \sum_n \sigma_n(t) fn(z)$ 를 도입하였다. 여기서 $\sigma_n(t) \in (0, 1)$ 은 역학 $f_n(z)$ 가 언제 적용되어야 하는지 학습하는 신경망이다. 이러한 모델을 continuous normalizing flow (CNF)라고 부른다.

Experiments with Continuous Normalizing Flows

- Density matching



<1> CNF를 구성하고 Adam을 사용하여 1만번의 iteration으로 학습시킴

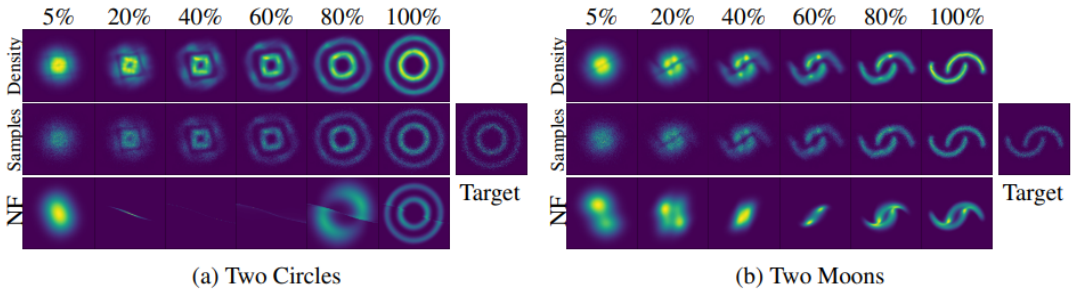
<2> NF는 RMSprop을 사용하여 50만 번의 iteration으로 학습됨

Flow model을 q라고 하면, 타겟 밀도 $p(\cdot)$가 평가될 수 있는 loss function인 $KL(q(x)\|p(x))$

을 최소화한다.

⇒ CNF가 일반적으로 더 낮은 loss를 달성한다.

- **Maximum Likelihood Training**

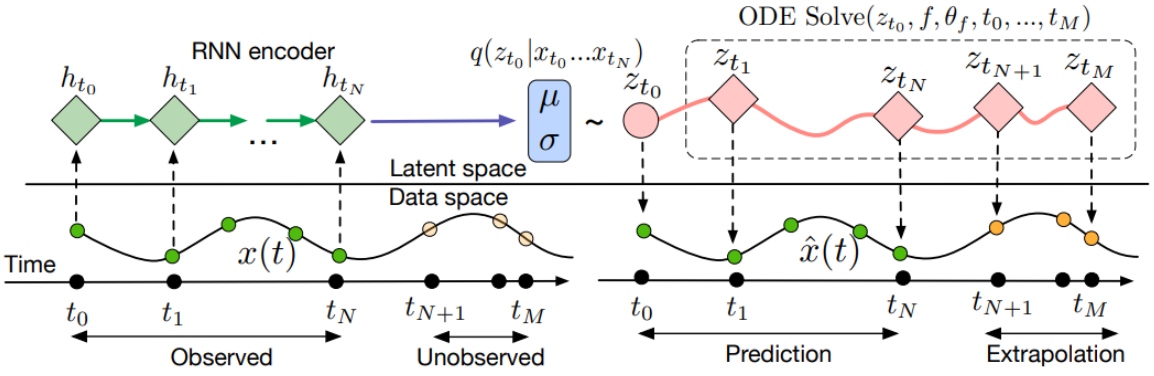


*noise에서 데이터로 변환하는 과정을 시각화한 것

*예시로 64개의 hidden unit으로 CNF를 구성하였고, hidden unit이 1개인 레이어 64개를 쌓아 NF를 구성함

CNF의 유용한 속성은 forward pass와 거의 동일한 비용으로 역변환을 계산할 수 있다는 것이다. 이는 NF는 불가능하다. 이를 통해 maximum likelihood estimation (MLE)을 수행하여 밀도 추정 task에 대한 flow model을 학습시킬 수 있다. 이는 적절히 change of variables 정리를 사용하여 $q(\cdot)$를 계산하고 $E_p(x)[q(x)]$를 최대화한 다음 CNF를 역전시켜 $q(x)$에서 랜덤 샘플을 생성한다.

5. A generative latent function time-series model



불규칙하게 샘플링된 데이터(ex. network traffic)에 신경망을 적용하는 것은 어렵다. 일반적으로 관측값은 고정된 기간의 저장소에 저장되고 latent의 역학은 동일한 방식으로 이산화되기 때문에, 이로 인해 누락된 데이터와 잘못 정의된 latent 변수로 인해 어려움이 발생한다.

본 연구에서는 시계열 모델링에 대한 연속적이고 생성적인 접근 방식을 제시하였다. 본 논문의 모델은 latent 궤적을 통해 각 시계열을 나타낸다. 각 궤적은 로컬한 초기 상태 zt0와 모든 시계열에서 공유되는 (Global) latent 역학 집합에서 결정된다. 관찰 시간 t0,t1,...,tN과 초기 상태 zt0가 주어지면 ODE solver는 zt1,...ztN을 생성하며, 이는 각 관찰에서 latent 상태를 설명한다. 저자들은 다음과 같은 샘플링 절차를 통해 이 생성 모델을 정의하였다.

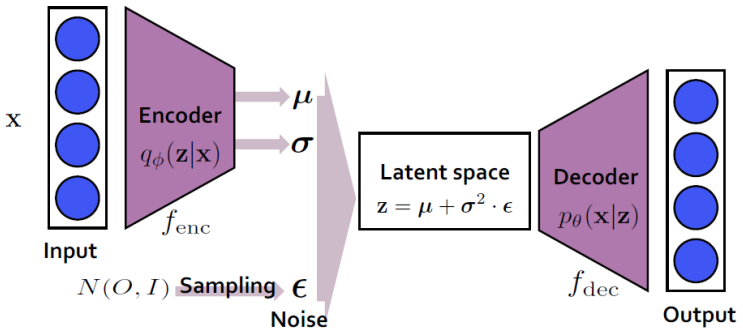
$$
\begin{aligned}
z_{t_0} &\sim p(z_{t_0}) \\
z_{t_1}, \ldots, z_{t_N} &= \mathrm{ODESolve}(z_{t_0}, f, \theta, t_0, \ldots, t_N) \\
\text{each} \quad x_{t_i} &\sim p(x|z_{t_i}, \theta_x)
\end{aligned}
$$

함수 f는 현재 timestep에서 z를 통해 기울기를 출력하는 time-invariant 함수이다. 저자는 신경망을 사용하여 이 함수를 parameterize하였다. f는 time-invariant 이므로 latent 상태 z(t) 가 주어지면 전체 latent 궤적이 정의된다.

Training and Prediction

이 latent-variable 모델은 variational autoencoder를 통해 학습시킬 수 있다.

- (참고) VAE 구조

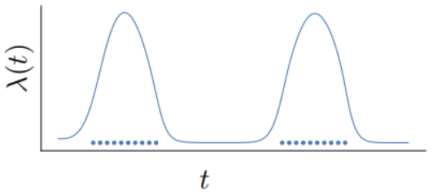


이때, Encoder로 RNN을 사용하고, ODE를 생성 모델로 사용하면 연속적인 타임라인에서 임의의 시점 t1,...,tM 에 대한 예측을 할 수 있다.

Poisson Process likelihoods

이벤트 비율은 latent 상태의 함수인 p(event at time t|z(t))=λ(z(t)) 의 함수로 parameterize될 수 있다. 이 함수가 주어지면 간격 내 일련의 독립적인 관측 시간의 likelihood는 다음과 같은 비균질 포아송 프로세스에 의해 제공된다.

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



관측 시간에 대한 포아송 프로세스 likelihood는 데이터 likelihood와 결합되어 모든 관측과 관측이 이루어진 시간을 공동으로 모델링할 수 있다.

5.1 Time-series Latent ODE Experiments

저자들은 latent ODE 모델이 시계열을 추정하는 능력을 조사하였다. Encoder는 25개의 hidden unit이 있는 RNN이며, 4차원의 latent space를 사용한다. Decoder는 20개의 hidden unit이 있는 하나의 hidden layer가 있는 또 다른 신경망이다.

Time series with irregular time points

저자들은 각각 다른 지점에서 시작하고 100개의 동일한 간격의 timestep으로 샘플링된 1000개의 2차원 나선 데이터셋을 생성했다.

이때, 불규칙한 타임스탬프를 생성하기 위해 교체 없이 각 궤적에서 에서 포인트를 무작위로 샘플링한다 (n= {30,50,100}). 저자들은 학습에 사용된 시점을 넘어서는 100개 시점에 대한 예측 RMSE(평균 제곱근 오차)를 측정하였다. 위 표는 latent ODE의 예측 RMSE가 상당히 낮다는 것을 보여준다.

# Observations	30/100	50/100	100/100
RNN	0.3937	0.3202	0.1813
Latent ODE	**0.1642**	**0.1502**	**0.1346**

Latent space interpolation

궤적은 두 개의 개별 궤적 클러스터를 형성한다. 하나는 시계 방향 나선으로 디코딩되고 다른 하나는 시계 반대 방향으로 디코딩된다.

위 그림은 latent 궤적이 시계 방향에서 시계 반대 방향 나선형으로 전환하면서 초기 지점 z(t0)z(t0)의 함수로 부드럽게 변하는 것을 보여준다.

6. Scope and Limitations

1. **minibatching**
   - 미니배치를 사용하는 것은 standard neural networks보다 덜 직관적.
   - ODE solver를 통해 각 배치 요소의 상태를 함께 연결하여 평가
     - 이를 통해 차원이 D × K인 결합된 ODE 생성
   - 모든 배치 요소의 오류를 제어하려면 각 시스템을 개별적으로 해결할 때보다 결합 시스템을 K배 더 자주 평가해야 함.
     - 그러나 실제로는 미니배치를 사용할 때 평가 횟수가 크게 증가하지 않았음
2. **uniqueness**
   - Picard의 Existence theorem는 초기값 문제의 해가 존재하고 고유하다는 것을 보장함.
   - 이 정리는 신경망이 유한한 가중치를 가지며 tanh 또는 relu와 같은 Lipschitz nonlinearities을 사용할 때 우리 모델에 적용됨.
3. **setting tolerances**
   - 이 프레임워크는 속도와 정확성을 교환할 수 있게 해주지만, 사용자가 훈련 중 순방향 및 역방향 패스의 오류 허용 오차를 선택해야 함.
4. **Reconstructing forward trajectories**
   - 역학을 역으로 실행하여 상태 경로를 재구성하면, 원래 경로와 재구성된 경로가 다를 경우 추가적인 수치적 오류가 발생할 수 있음.
     - 이 문제는 checkpointing을 통해 해결 가능.
       - 체크포인트는 순방향 패스의 중간 값을 저장하고, 해당 지점에서 다시 통합(integrating)하여 정확한 순방향 경로를 재구성하는 방법

7. Related Work

1. Adjoint Method
2. Adaptive computation
3. Constant memory backprop through reversibility
4. Learning differential equations
5. Differentiating through ODE solvers

8. Conclusion

- black-box ODE solver를 모델 구성 요소로 사용하는 방법을 연구함.
    - 시계열 모델링, 지도 학습, 밀도 추정
- 대규모 레이어 크기로 확장 가능한 continuous-time normalizing flows를 도출