CHAPTER 7

변이형 오토인코더

7.1 VAE와 디코더

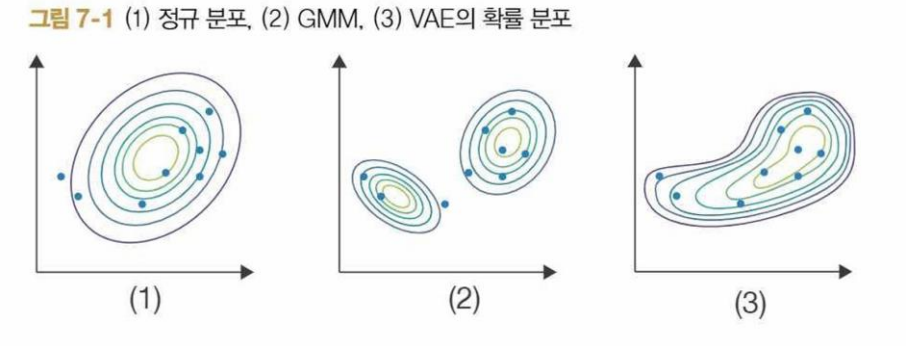


그림 7-1 (1) 정규 분포, (2) GMM, (3) VAE의 확률 분포

② 가우스 혼합 모델(GMM)

- 여러 개의 정규 분포를 혼합하여 표현력을 높인 모델입니다.
- 범주형 분포에서 정규 분포 하나를 선택한 뒤, 데이터를 생성합니다.
- 정규 분포가 여러 개 있기 때문에 표현력이 좀 더 좋아집니다.
- 하지만 여전히 **잠재 변수가 이산형 변수**라 복잡한 데이터 구조를 모두 표현하기 어렵습니다.

(1) 기존 모델들의 한계

① 하나의 정규 분포 모델

- 데이터의 확률 분포를 단순히 하나의 정규 분포로 표현하는 방식입니다.
- 장점: 매우 간단하고 직관적.
- 단점: 너무 간단해서 복잡한 데이터(이미지, 음성 등)는 표현하기 어렵습니다.

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\theta})$$

$$\log p_{\boldsymbol{\theta}}(\mathcal{D}) = \log \left(p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(1)})\ p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(2)})\ \cdots\ p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(N)}) \right)$$

$$\log p_{\boldsymbol{\theta}}(\mathcal{D}) \geq \underbrace{\sum_{n=1}^{N} \sum_{z^{(n)}} q^{(n)}(z^{(n)}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(n)}, z^{(n)})}{q^{(n)}(z^{(n)})}}_{\text{ELBO}}$$

EM 알고리즘은 ELBO를 목적 함수로 설정하고 확률 분포 $q^{(n)}(z)$와 매개변수 θ를 번갈아 갱신합니다. 정확하게는 다음 두 작업을 반복합니다.

1 **E-스텝**: $\{q^{(1)}, q^{(2)} \cdots q^{(N)}\}$ 갱신
각 n에 대해 $q^{(n)}(z) = p_\theta(z \mid \boldsymbol{x}^{(n)})$으로 정한다.

2 **M-스텝**: θ 갱신
ELBO가 최대가 되는 θ를 해석적으로 구한다.

7.1.3 VAE 와 디코더

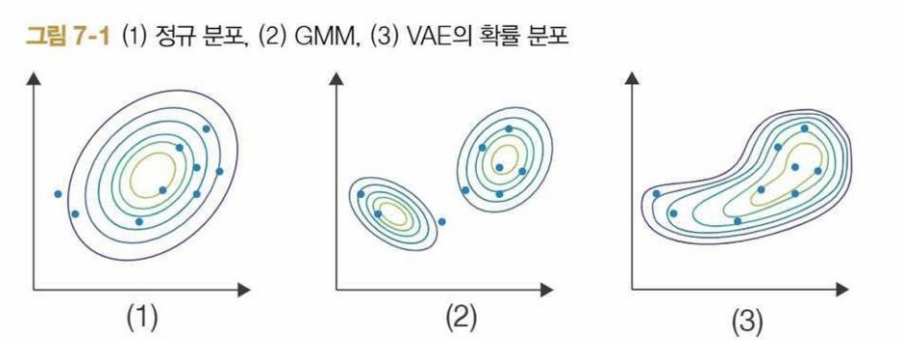
(2) VAE란 무엇인가?

① VAE의 기본 아이디어

- VAE는 신경망을 활용한 **복잡한 데이터 분포를 표현**할 수 있는 **강력한 생성 모델**입니다.
- **연속된 벡터 형태의 잠재 변수(latent variable)를 사용**합니다.
    - 잠재 변수는 데이터의 **추상적이고 압축된 표현**을 의미합니다.

② VAE의 핵심 구성 요소

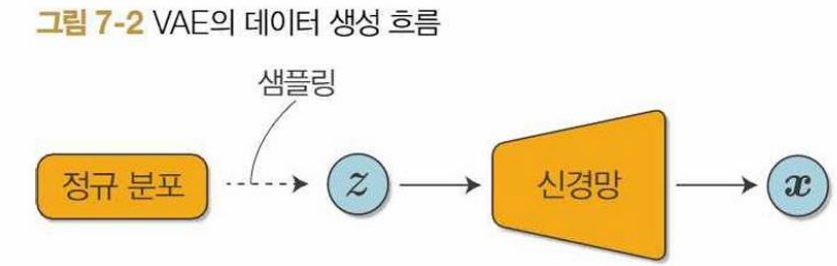
- **잠재 변수 z**: 데이터 생성의 시작점.
- **디코더(Decoder)**: 잠재 변수를 받아 데이터로 변환해 주는 신경망.



$$z = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_H \end{pmatrix}$$

1단계: 잠재 변수 z 샘플링

- **잠재 변수 z** 는 미리 고정된 정규 분포에서 생성됩니다.
- 일반적으로 평균이 0이고 공분산 행렬이 단위행렬(III)인 정규 분포에서 생성됩니다.



2단계: 디코더(Decoder)를 통한 데이터 생성

•**디코더는 신경망으로 구성**되며, 이 신경망이 잠재 변수 z를 실제 관측 가능한 데이터 x로 변환합니다.

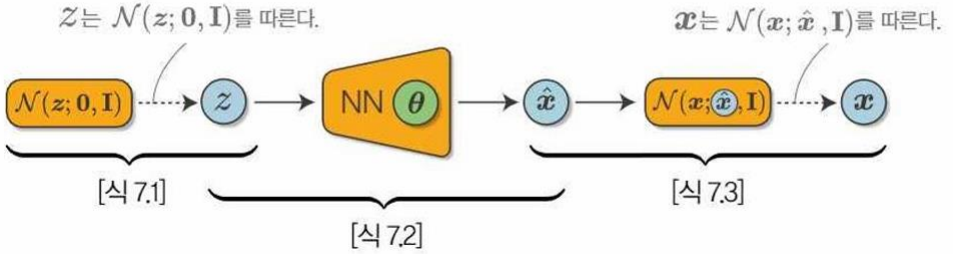
•디코더의 출력은 데이터 분포의 평균벡터(즉, 중심 위치)를 나타냅니다.

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathbf{I}) \quad [식\ 7.1]$$

$$\hat{\boldsymbol{x}} = \text{NeuralNet}(\boldsymbol{z}; \boldsymbol{\theta}) \quad [식\ 7.2]$$

$$p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}; \hat{\boldsymbol{x}}, \mathbf{I}) \quad [식\ 7.3]$$

그림 7-3 VAE의 디코더가 수행하는 처리(NN = 신경망)



(4) GMM과 VAE의 핵심 차이

	GMM	VAE
잠재 변수의 형태	이산형 변수(범주형)	연속형 벡터
표현 능력	제한적 (정규 분포 몇 개의 혼합)	매우 유연하고 강력한 표현력
사용 모델	단순 혼합 정규 분포	**인공 신경망**을 사용하여 훨씬 복잡한 변환 가능

(5) EM 알고리즘의 한계와 VAE 도입 이유

• 이론상 EM 알고리즘으로 VAE 모델의 매개변수를 학습할 수 있습니다.

• 하지만 VAE의 잠재 변수 z가 다차원 벡터 형태이기 때문에, EM 알고리즘에서 사용하는 사후 분포(conditional distribution)의 **적분 계산이 거의 불가능한 수준으로 어려워집니다**.

• 이를 극복하기 위해 VAE는 **변분 베이즈(Variational Bayes)** 접근법을 도입하게 됩니다. (다음 절에서 다룰 내용)

(6) 요약

• **정규 분포, GMM**은 복잡한 데이터 표현에 한계가 있습니다.

• **VAE**는 **연속적인 잠재 변수를 가진 신경망 모델**로 매우 복잡한 데이터 분포를 표현할 수 있습니다.

• 데이터를 생성할 때는 간단한 정규 분포에서 샘플링한 잠재 변수를 **디코더(신경망)**로 복잡한 데이터 공간으로 변환합니다.

•잠재 변수가 연속형 벡터이기 때문에 복잡하고 유연한 표현이 가능합니다.

7.2 VAE와 인코더

•기존 **EM 알고리즘**의 한계를 짚어보고, 이를 극복하기 위해 VAE가 어떻게 접근하는지 소개합니다.

•VAE의 학습 목표인 **ELBO**를 다시 정의하고, 이를 최적화하는 방법인 **변분 베이즈**와 **인코더 신경망**을 명확히 설명합니다.

(1) EM 알고리즘에서 VAE로

◆ EM 알고리즘 복습

•잠재 변수가 있는 모델은 EM 알고리즘으로 학습할 수 있습니다.

•EM 알고리즘의 E-step에서는 잠재 변수의 사후분포 p(z|x)를 계산합니다.

•하지만 VAE의 잠재 변수는 연속적이고 다차원이기 때문에, EM 알고리즘의 핵심 계산인 사후 분포 p(z|x) 의 적분이 매우 복잡하거나 불가능해집니다.

$$
\begin{aligned}
&\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \\
&= \left(\int q(\boldsymbol{z})d\boldsymbol{z}\right)\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) && \left(\int q(\boldsymbol{z})d\boldsymbol{z}=1\text{을 곱한다.}\right) \\
&= \int q(\boldsymbol{z})\log p_{\boldsymbol{\theta}}(\boldsymbol{x})d\boldsymbol{z} && \left(\log p_{\boldsymbol{\theta}}(\boldsymbol{x})\text{를 }\int\text{ 안으로}\right) \\
&= \int q(\boldsymbol{z})\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}\mid\boldsymbol{x})}d\boldsymbol{z} && \text{(확률의 곱셈 정리)} \\
&= \int q(\boldsymbol{z})\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}\mid\boldsymbol{x})}\frac{q(\boldsymbol{z})}{q(\boldsymbol{z})}d\boldsymbol{z} && \left(\frac{q(\boldsymbol{z})}{q(\boldsymbol{z})}=1\text{을 곱한다.}\right) \\
&= \int q(\boldsymbol{z})\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z})}d\boldsymbol{z} + \underbrace{\int q(\boldsymbol{z})\log \frac{q(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}\mid\boldsymbol{x})}d\boldsymbol{z}}_{\text{KL 발산}}
\end{aligned}
$$

$$
\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \int q(\boldsymbol{z})\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z})}d\boldsymbol{z} + D_{\mathrm{KL}}(q(\boldsymbol{z}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) \\
&\geq \underbrace{\int q(\boldsymbol{z})\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z})}d\boldsymbol{z}}_{\text{ELBO}}
\end{aligned}
\qquad \text{[식 7.4]}
$$

이 식에서 $\mathrm{ELBO}(x; q, \theta)$가 더 커지도록 매개변수를 갱신하면 $\log p_\theta(x)$는 그 이상의 값이 될 것임을 알 수 있습니다. 따라서 감당할 수 없는 $\log p_\theta(x)$ 대신에 $\mathrm{ELBO}(x; q, \theta)$를 최적화 대상으로 삼습니다.

EM 알고리즘에서는 $q(\boldsymbol{z})$와 $\boldsymbol{\theta}$를 번갈아 갱신합니다. $q(\boldsymbol{z})$를 갱신할 때는 $q(\boldsymbol{z}) = p_\theta(\boldsymbol{z} \mid \boldsymbol{x})$라는 사후 분포를 구합니다. $\boldsymbol{\theta}$를 갱신할 때는 ELBO를 최대화합니다. 그런데 VAE에서는 사후 분포를 직접 구하기가 어렵기 때문에 다음 방식을 이용합니다.

1 $q(\boldsymbol{z})$를 간단한 확률 분포(예: 정규 분포)로 제한한다.

2 한정된 확률 분포 내에서 ELBO를 최대화한다.

◆ **VAE의 아이디어**

•정확한 사후 분포 p(z|x) 를 직접 계산하지 않고, **간단한 확률분포로 근사**합니다.

•이를 **변분 베이즈(Variational Bayes)** 혹은 **변분 근사**라고 합니다.

1 $q(z)$를 간단한 확률 분포(예: 정규 분포)로 제한한다.

2 한정된 확률 분포 내에서 ELBO를 최대화한다.

$$q_\psi(z) = \mathcal{N}(z; \mu, \Sigma)$$

그러면 [식 7.4]를 다음 식으로 표현할 수 있습니다.

$$\log p_\theta(x) = \underbrace{\int q_\psi(z) \log \frac{p_\theta(x,z)}{q_\psi(z)} dz}_{\text{ELBO}} + D_{\text{KL}}(q_\psi(z) \parallel p_\theta(z \mid x)) \quad \text{[식 7.5]}$$

(2) 변분 베이즈(Variational Bayes)의 기본 아이디어

•변분 베이즈는 원래의 복잡한 확률분포를 다루기 쉬운 간단한 확률분포로 근사합니다.

•이 간단한 확률분포를 보통 정규 분포로 설정하며, 이를 $q_\Psi(z \mid x)$ 로 표현합니다.

•VAE의 학습은 바로 이 **근사된 분포** $q_\Psi(z \mid x)$를 잘 추정하는 과정입니다.

•근사 분포를 정규 분포로 제한함으로써 문제를 매우 간단하게 만들어줍니다.

그림 7-4 ELBO를 ψ에 대해 최대화할 때의 변화(z가 1차원 데이터라고 가정)

(3) ELBO(Evidence Lower Bound)의 재등장

•VAE가 실제로 최적화하는 대상은 로그 가능도($\log p_\theta(x)$)가 아니라 **ELBO**입니다.

•ELBO는 로그 가능도의 하한으로, 이를 최대화하면 간접적으로 원래 로그 가능도를 증가시킵니다.

(5) 전체 데이터셋으로 확장된 ELBO

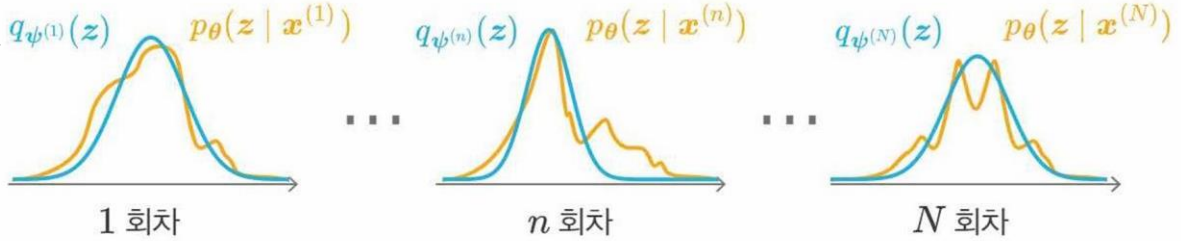
데이터가 여러 개$(x^{\{(1)\}}, x^{\{(2)\}}, \ldots, x^{\{(N)\}})$있다면, 전체 ELBO는 다음과 같이 표현됩니다:

•문제점: 데이터가 많아질수록 각 데이터마다 근사 분포의 매개변수를 일일이 준비해야 합니다.

(데이터셋 이 크다면 예컨대 데이터가 1억 개라면 매개변수도 1억 개 필요, 비현실적)

$$\sum_{n=1}^{N} \mathrm{ELBO}(\boldsymbol{x}^{(n)}; \boldsymbol{\theta}, \boldsymbol{\psi}^{(n)}) = \sum_{n=1}^{N} \int q_{\boldsymbol{\psi}^{(n)}}(\boldsymbol{z}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(n)}, \boldsymbol{z})}{q_{\boldsymbol{\psi}^{(n)}}(\boldsymbol{z})} d\boldsymbol{z} \quad \text{[식 7.7]}$$

그림 7-6 각 데이터 $\boldsymbol{x}^{(n)}$마다 매개변수 $\boldsymbol{\psi}^{(n)}$을 준비한다.



(6) 인코더(Encoder)의 등장

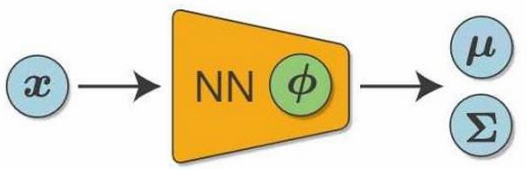
•위 문제(각 데이터마다 매개변수가 너무 많아짐)를 해결하기 위해 **하나의 신경망(인코더)**을 사용합니다.

•인코더는 **데이터 x** 를 입력으로 받아서, 그 데이터에 적합한 근사 분포의 매개변수(평균과 공분산)를 출력합니다.

즉, 인코더 신경망의 역할은 다음과 같습니다:

$$(\mu, \sigma) = \mathrm{NeuralNet}(x; \phi)$$

그림 7-7 데이터로부터 정규 분포의 매개변수를 출력하는 신경망



•여기서 μ, σ는 잠재 변수 분포의 매개변수입니다.

이러한 방법을 **분할 상환 추론(Amortized inference)**이라 부릅니다.

즉, 모든 데이터를 한 번에 처리하는 '하나의 신경망'으로 효율성을 높입니다.

https://ricoshin.tistory.com/3

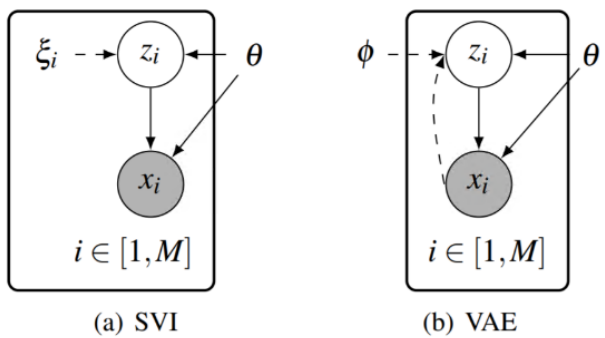
Bayesian neural network를 variational inference 사용하여 학습하는 상황을 가정해보자.

관찰 포인트가 여러 개라고 하더라도 상응되는 global latent variable은 하나만 존재하도록 모델링할 수도 있다.
(일반적으로 weight의 approximate posterior에 사용되는 variational parameter는 모든 데이터에 대해 공통)

그런데 만약 Observable variables x_i에 대응하는 local latent variable z_i이 각각 존재한다고 한다면?
(VAE가 대표적인 케이스)

두 가지 문제가 생긴다.
1. 데이터가 늘어날 수록 그에 대응하는 latent variable이 계속해서 선형적으로 늘어나야 한다.
2. 새로운 데이터 x^*가 들어오면 z^*에 대하여 다시 피팅을 해야한다.

이 문제를 해결하기 위한 방법이 **amortized inference**이다.
결론부터 말하면 네트워크가 x_i를 입력받아 z_i를 만들어내도록 하는 방법이다.
VAE에서 encoder가 데이터 포인트마다 다른 latent code를 만들어 내는 것과 같다.



(a) SVI (b) VAE

(a) Standard Variational Inferecne (b) Amortized Vritional Inference (그림 출처: https://arxiv.org/pdf/1711.05597)

1 SVI(Standard Variational Inference, 표준 변분추론)

- 각 데이터 x_i 마다 별도의 변분 파라미터 ξ_i를 만들어 개별적으로 최적화(optimize)합니다.
- 데이터의 개수 M 만큼 매번 별도의 파라미터를 찾는 작업을 반복합니다.
- 이는 데이터를 추가하거나 바꿀 때마다 매번 최적화 작업을 수행해야 하므로 비용이 큽니다.

쉽게 말하면: 데이터 하나하나마다 별도의 최적화를 진행하는 방식으로, 마치 매 문제마다 처음부터 새로 문제를 푸는 방식입니다.

---

2 VAE(Amortized Variational Inference, 변분 오토인코더 예시)

- 이 방식에서는 데이터마다 매번 새로 파라미터를 찾는 것이 아니라, 하나의 공유되는 파라미터 ϕ (흔히 신경망의 가중치)를 학습합니다.
- 이렇게 미리 학습된 파라미터를 이용해 새로운 데이터가 들어오면 곧바로(feed-forward) 숨겨진 잠재변수(latent variable) z_i를 쉽게 추론할 수 있습니다.
- 즉, 매 데이터마다 다시 최적화를 하지 않고, 이전에 학습한 지식을 재활용하는 방식입니다.

쉽게 말하면: 처음에 비용을 내고 하나의 일반적인 규칙(네트워크)을 만들어 놓으면, 이후에 들어오는 데이터는 미리 만들어 놓은 규칙을 바로 적용하여 빠르고 효율적으로 처리하는 방식입니다.

🎯 "Amortize"(나누어 지불한다)의 의미:

- 초기에는 비용이 들어가지만, 나중에 추가적으로 들어오는 데이터마다 발생하는 추가 비용이 크지 않도록 처음 비용을 분산시켜 미리 지불한 것과 같습니다.
- 매번 새롭게 최적화를 하는 것이 아니라, 이미 만들어 놓은 시스템을 사용해 빠르게 결과를 얻는 효율적인 전략을 의미합니다.

📌 요약 비교:

방식	개념	장단점 비교
SVI (표준 변분추론)	데이터마다 매번 새로 최적화	더 정확할 수 있으나 매번 비용이 큼
VAE (Amortized VI)	한 번 학습하고 여러 번 사용	초기 비용이 있지만 이후 효율적으로 추론 가능

(7) 인코더를 통한 최종 근사 분포 표현

•실제 구현상 효율성을 위해 **공분산 행렬을 대각 행렬로 한정**합니다.

•즉, 근사 분포는 다음과 같습니다:

$$q_\phi(z|x) = N(z; \mu, \sigma^2 I)$$

•여기서 μ, σ는 모두 신경망(인코더)의 출력으로부터 얻습니다.

$$\Sigma = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_H^2 \end{pmatrix} \qquad \mu = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_H \end{pmatrix} \qquad \sigma = \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_H \end{pmatrix}$$

이상의 내용을 정리하면 VAE의 인코더가 수행하는 일은 다음 식으로 표현할 수 있습니다.

$$\boldsymbol{\mu}, \boldsymbol{\sigma} = \mathrm{NeuralNet}(\boldsymbol{x}; \boldsymbol{\phi}) \quad \text{[식 7.8]}$$

$$q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I}) \quad \text{[식 7.9]}$$

여기서 $\boldsymbol{\sigma}^2\mathbf{I}$는 다음과 같은 대각 공분산 행렬을 뜻합니다.

$$\boldsymbol{\sigma}^2\mathbf{I} = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_H^2 \end{pmatrix}$$

요약

•EM 알고리즘은 VAE의 잠재 변수의 사후 분포를 계산하기 어렵습니다. (적분의 어려움)

•VAE는 **변분 베이즈(Variational Bayes)** 접근법을 이용하여 사후 분포를 간단한 정규 분포로 근사합니다.

•이 정규 분포의 매개변수는 **인코더 신경망**을 통해 데이터로부터 효율적으로 얻습니다.

•ELBO를 최대화하면, 근사 분포가 실제 사후 분포와 가까워지도록 학습이 진행됩니다.

비유를 통한 이해

•EM 알고리즘: 정확한 답을 찾으려 했지만 계산이 너무 어려워 실패.

•VAE(변분 베이즈): 정확한 답 대신 "쉽게 다룰 수 있는 비슷한 답"으로 문제를 바꿔 해결.

•인코더 신경망: 여러 개의 문제를 하나의 기계(신경망)로 효율적으로 처리하여 매개변수를 자동으로 계산하는 역할 수행.

7.3 ELBO 최적화

VAE의 손실 함수(ELBO)를 최적화하기 위한 실제적인 방법과 기술적 문제를 해결하는 핵심 아이디어인
재매개변수화 트릭(Reparameterization Trick)에 대해 자세히 설명합니다.

- **ELBO**를 명확히 정의하고, VAE가 실제로 최적화하는 손실 함수로 변형합니다.
- 손실 함수의 각 항(**재구성 항, KL 발산 항**)의 의미를 명확히 설명합니다.
- VAE 학습 시 발생하는 기술적 문제(샘플링 과정의 미분 불가능성)를 **재매개변수화 트릭**으로 해결하는 방법을 소개합니다.

지금까지의 이야기를 정리해보겠습니다. 7.1절에서는 디코딩을 수행하는 신경망이 등장했습니다. 수식으로는 다음처럼 표현할 수 있습니다.

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathbf{I}) \quad \text{[식 7.1]}$$

$$\hat{\boldsymbol{x}} = \text{NeuralNet}(\boldsymbol{z}; \boldsymbol{\theta}) \quad \text{[식 7.2]}$$

$$p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}; \hat{\boldsymbol{x}}, \mathbf{I}) \quad \text{[식 7.3]}$$

다음으로 7.2절에서 소개한 인코딩 신경망은 다음 식으로 표현할 수 있습니다.

$$\boldsymbol{\mu}, \boldsymbol{\sigma} = \text{NeuralNet}(\boldsymbol{x}; \boldsymbol{\phi}) \quad \text{[식 7.8]}$$

$$q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I}) \quad \text{[식 7.9]}$$

우리의 목표는 이 두 신경망을 사용한 모델에서 로그 가능도를 최대화하는 매개변수 $\boldsymbol{\theta}, \boldsymbol{\phi}$를 찾는 것입니다. 하지만 로그 가능도 자체를 직접 처리할 수는 없으므로, 대신 다음의 ELBO를 최대화합니다.

$$\text{ELBO}(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\phi}) = \int q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})} d\boldsymbol{z}$$

이 식은 하나의 데이터 $\boldsymbol{x}$에 대한 ELBO입니다. $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)} \cdots \boldsymbol{x}^{(N)}\}$처럼 데이터가 N개라면 다음처럼 ELBO들의 합이 최대화 대상이 됩니다.

$$\sum_{n=1}^{N} \text{ELBO}(\boldsymbol{x}^{(n)}; \boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{n=1}^{N} \int q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}^{(n)}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(n)}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}^{(n)})} d\boldsymbol{z}$$

(1) ELBO 다시 정리하기

앞선 내용을 다시 한 번 정리하면, VAE가 최적화하는 함수인 ELBO는 다음과 같이 나타낼 수 있습니다:

$$
\begin{aligned}
&\mathrm{ELBO}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) \\
&= \int q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid\boldsymbol{x}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid\boldsymbol{x})} d\boldsymbol{z} \\
&= \int q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid\boldsymbol{x}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}\mid\boldsymbol{z})p(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid\boldsymbol{x})} d\boldsymbol{z} \quad ❶ \\
&= \int q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid\boldsymbol{x}) \log p_{\boldsymbol{\theta}}(\boldsymbol{x}\mid\boldsymbol{z}) d\boldsymbol{z} + \int q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid\boldsymbol{x}) \log \frac{p(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid\boldsymbol{x})} d\boldsymbol{z} \quad ❷ \\
&= \int q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid\boldsymbol{x}) \log p_{\boldsymbol{\theta}}(\boldsymbol{x}\mid\boldsymbol{z}) d\boldsymbol{z} - \int q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid\boldsymbol{x}) \log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid\boldsymbol{x})}{p(\boldsymbol{z})} d\boldsymbol{z} \quad ❸
\end{aligned}
$$

식을 전개하면서 다음 공식을 적용했습니다.

❶ $p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}\mid\boldsymbol{z})\ p(\boldsymbol{z})$

❷ $\log AB = \log A + \log B$

❸ $\log \frac{A}{B} = -\log \frac{B}{A}$

$$
\mathrm{ELBO}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) = \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}\mid\boldsymbol{z})]}_{J_1} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid\boldsymbol{x}) \parallel p(\boldsymbol{z}))}_{J_2} \quad \text{[식 7.10]}
$$

$\boldsymbol{J_1}$: 데이터 x를 잘 복원하는 정도를 측정 (**재구성 항, Reconstruction Term**).

$\boldsymbol{J_2}$: 근사 사후 분포 qϕ(z|x) 와 잠재변수의 사전분포 p(z)가 얼마나 가까운지 측정
(**정규화 항, Regularization Term**).

① 재구성 항(Reconstruction Term)

•목적: **데이터를 얼마나 잘 재구성하는지 측정**

•구체적으로는 디코더 신경망이 잠재 변수 z를 통해 원래 데이터를 얼마나 잘 복원하는지를 평가합니다.

•쉽게 말해, 이 값이 클수록 원래 데이터에 더 가까운 데이터를 생성합니다.

•보통 **평균제곱오차(MSE)** 또는 **교차 엔트로피**를 사용하여 계산합니다.

② KL 발산 항(KL Divergence Term)

•목적: 근사 사후 분포 qϕ(z|x) 를 **잠재변수의 사전분포 p(z)** 에 가깝게 유지하여, 잠재 공간의 구조를 일정하게 유지합니다.

•쉽게 말해, 이 값이 작을수록 잠재 공간이 잘 정리됩니다.

•정규 분포 간의 KL 발산은 해석적으로 구할 수 있어서 쉽게 계산됩니다.

즉, VAE는 "잘 복원하면서도 잠재 공간을 규칙적으로 유지"하도록 두 가지를 균형있게 학습합니다.

(2) ELBO의 두 가지 핵심 항

VAE의 ELBO는 다음 두 항으로 나누어집니다:

$$\mathrm{ELBO}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) = \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})]}_{J_1} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p(\boldsymbol{z}))}_{J_2} \quad \text{[식 7.10]}$$

① 재구성 항(Reconstruction Term)

•목적: **데이터를 얼마나 잘 재구성하는지 측정**

•구체적으로는 디코더 신경망이 잠재 변수 z를 통해 원래 데이터를 얼마나 잘 복원하는지를 평가합니다.

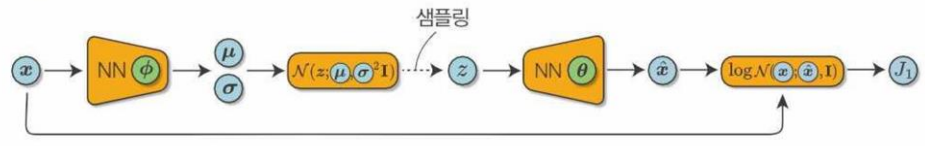
•쉽게 말해, 이 값이 클수록 원래 데이터에 더 가까운 데이터를 생성합니다.

•보통 **평균제곱오차(MSE)** 또는 **교차 엔트로피**를 사용하여 계산합니다.

J_1은 기댓값이므로 몬테카를로 방법으로 근삿값을 구할 수 있습니다. 구체적으로 $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$에서 몇 개의 무작위 수를 생성하고, 생성된 무작위 수를 이용하여 $\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$의 평균을 구합니다. VAE에서는 샘플이 하나뿐이어도 잘 작동하는 경우가 많습니다. 그래서 샘플 크기를 1이라고 가정하면, $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$에 따르는 $\boldsymbol{z}$를 한 개 샘플링하고 $\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$를 계산하여 [식 7.10]의 J_1을 근사할 수 있습니다. 이로부터 J_1을 다음과 같이 계산할 수 있습니다.

$$\begin{aligned}
\boldsymbol{\mu}, \boldsymbol{\sigma} &= \mathrm{NeuralNet}(\boldsymbol{x};\boldsymbol{\phi}) && \text{[식 7.8]}\\
\boldsymbol{z} &\sim \mathcal{N}(\boldsymbol{z};\boldsymbol{\mu},\boldsymbol{\sigma}^2\mathbf{I})\\
\hat{\boldsymbol{x}} &= \mathrm{NeuralNet}(\boldsymbol{z};\boldsymbol{\theta}) && \text{[식 7.2]}\\
J_1 &\approx \log \mathcal{N}(\boldsymbol{x};\hat{\boldsymbol{x}},\mathbf{I})
\end{aligned}$$

그림 7-9 J_1의 계산 흐름



여기서 하는 계산은 이렇습니다. 먼저 인코더인 $\mathrm{NeuralNet}(\boldsymbol{x};\boldsymbol{\phi})$가 입력 데이터 $\boldsymbol{x}$로부터 $\boldsymbol{z}$를 샘플링합니다. 이어서 디코더인 $\mathrm{NeuralNet}(\boldsymbol{z};\boldsymbol{\theta})$에 따라 데이터 $\hat{\boldsymbol{x}}$이 다시 생성됩니다. 다시 생성된 데이터 $\hat{\boldsymbol{x}}$이 원래의 입력 데이터 $\boldsymbol{x}$에 가까울수록 J_1의 값이 커집니다. 그래서 J_1을 **재구성 오차**reconstruction error 혹은 **재구성 오류**라고 합니다.

계속해서 $J_1 \approx \log \mathcal{N}(\boldsymbol{x};\hat{\boldsymbol{x}},\mathbf{I})$ 계산을 더 진행해봅시다.

$$\begin{aligned}
J_1 &\approx \log \mathcal{N}(\boldsymbol{x};\hat{\boldsymbol{x}},\mathbf{I})\\
&= \log\left(\frac{1}{\sqrt{(2\pi)^D|\mathbf{I}|}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\hat{\boldsymbol{x}})^\top\mathbf{I}^{-1}(\boldsymbol{x}-\hat{\boldsymbol{x}})\right)\right)\\
&= -\frac{1}{2}(\boldsymbol{x}-\hat{\boldsymbol{x}})^\top(\boldsymbol{x}-\hat{\boldsymbol{x}}) + \underbrace{\log\frac{1}{\sqrt{(2\pi)^D}}}_{\text{상수}}\\
&= -\frac{1}{2}\sum_{d=1}^{D}(x_d-\hat{x}_d)^2 + \mathrm{const}
\end{aligned}$$

② KL 발산 항(KL Divergence Term)

•목적: 근사 사후 분포 qϕ(z|x) 를 **잠재변수의 사전분포 p(z)** 에 가깝게 유지하여, 잠재 공간의 구조를 일정하게 유지합니다.

•쉽게 말해, 이 값이 작을수록 잠재 공간이 잘 정리됩니다.

•정규 분포 간의 KL 발산은 해석적으로 구할 수 있어서 쉽게 계산됩니다.

$$\mathrm{ELBO}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) = \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})]}_{J_1} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{z}))}_{J_2} \quad \text{[식 7.10]}$$

$D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{z}))$는 다음 두 정규 분포에 대한 KL 발산입니다.

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z};\boldsymbol{\mu},\boldsymbol{\sigma}^2\mathbf{I})$$
$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z};\mathbf{0},\mathbf{I})$$

두 정규 분포 사이의 KL 발산

두 정규 분포 사이의 KL 발산은 해석적으로 구할 수 있습니다. 예컨대 $q(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z};\boldsymbol{\mu}_1,\boldsymbol{\sigma}_1^2\mathbf{I})$ 이고 $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z};\boldsymbol{\mu}_2,\boldsymbol{\sigma}_2^2\mathbf{I})$라고 한다면, KL 발산은 다음 식으로 표현합니다(H는 $\boldsymbol{z}$의 차원 수).

$$D_{\mathrm{KL}}(q \parallel p) = -\frac{1}{2}\sum_{h=1}^{H}\left(1 + \log\frac{\sigma_{1,h}^2}{\sigma_{2,h}^2} - \frac{(\mu_{1,h}-\mu_{2,h})^2}{\sigma_{2,h}^2} - \frac{\sigma_{1,h}^2}{\sigma_{2,h}^2}\right)$$

따라서 이번 KL 발산은 다음과 같이 계산할 수 있습니다.

$$\begin{aligned} J_2 &= D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(z \mid x) \parallel p(z)) \\ &= -\frac{1}{2}\sum_{h=1}^{H}\left(1 + \log\sigma_h^2 - \mu_h^2 - \sigma_h^2\right) \end{aligned}$$

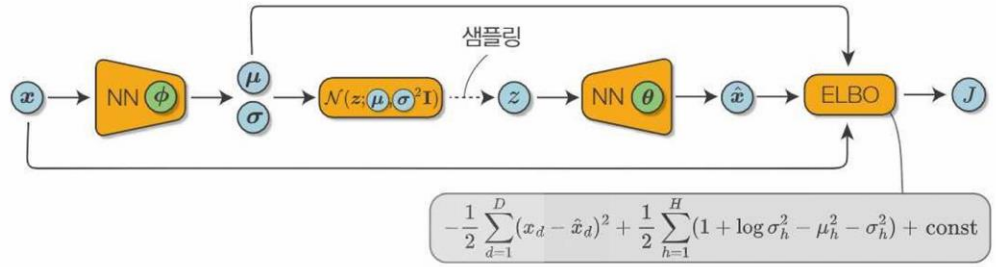
③ 최종 ELBO 계산

$$\mathrm{ELBO}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) \approx -\frac{1}{2}\sum_{d=1}^{D}(x_d-\hat{x}_d)^2+\frac{1}{2}\sum_{h=1}^{H}\left(1+\log\sigma_h^2-\mu_h^2-\sigma_h^2\right)+\mathrm{const}$$

참고로 ELBO를 계산하는 흐름은 그림 7-10 과 같습니다.

이 그림처럼 인코더와 디코더를 사용하는 모습이 예로부터 사용되어 온 오토인코더를 연상시키기 때문에 **VAE를 변이형 오토 인코더** 라고 부릅니 다.

그림 7-10 ELBO를 계산하는 흐름



ELBO 최적화와 매개변수 갱신 (θ, ϕ)

• VAE의 목표는 ELBO라는 값을 최대화하는 것입니다.

• 이를 위해 VAE는 두 가지 매개변수(θ, ϕ)를 최적화합니다.

• θ: 디코더 신경망의 매개변수 (데이터를 생성하는 부분)

• ϕ: 인코더 신경망의 매개변수 (잠재변수로 변환하는 부분)

• 이 두 매개변수는 신경망의 매개변수이므로, 경사 하강법(역전파)을 이용해 동시에 최적화할 수 있습니다.

문제 발생: 샘플링 부분의 미분 불가능성

• 그런데 문제가 하나 있습니다. VAE에서 잠재변수 z 를 샘플링하는 과정은 무작위적(random)인 연산입니다.

• 무작위 샘플링은 미분이 불가능하므로, 이 부분에서 역전파(Backpropagation)가 끊어지는 문제가 발생합니다.

해결 방법: 재매개변수화 트릭 (Reparameterization Trick)

• 이 문제를 해결하는 방법이 바로 **재매개변수화 트릭**입니다.

• 재매개변수화 트릭을 사용하면 샘플링 과정을 미분 가능한 형태로 바꿀 수 있습니다.

• 덕분에 역전파가 가능해져서 θ,ϕ 매개변수를 효과적으로 학습할 수 있게 됩니다.

(3) 재매개변수화 트릭(Reparameterization Trick)

재매개변수화 트릭은 다음과 같은 방식으로 잠재변수 z를 표현하여 샘플링 과정을 미분 가능한 형태로 변형합니다:

원래 방식 (미분 불가능):

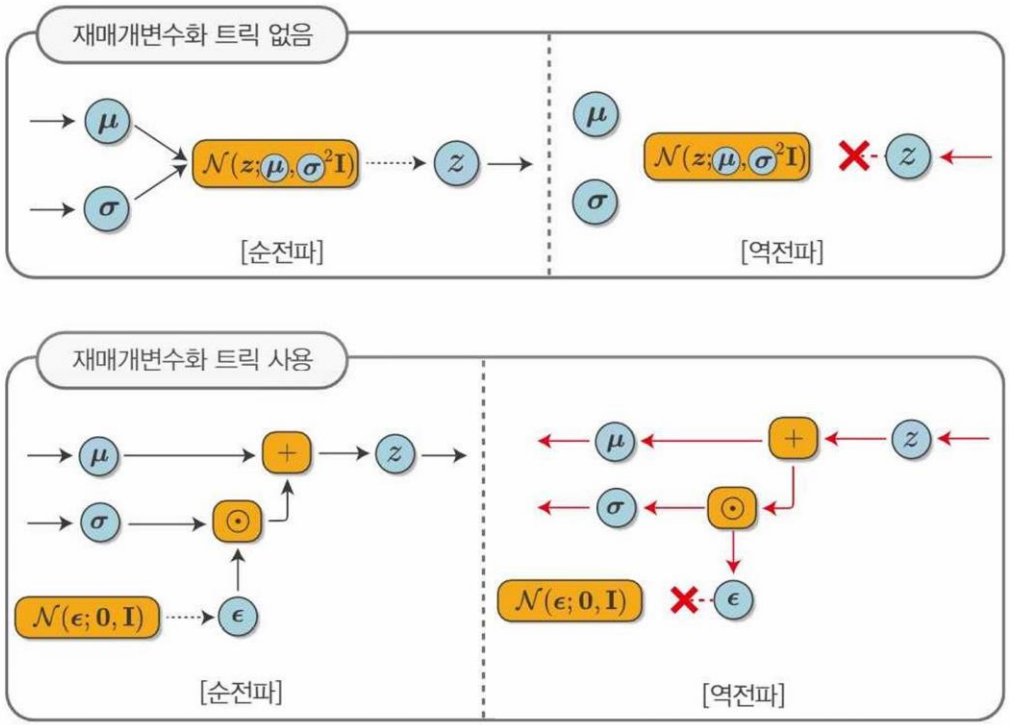
- $z \sim N(\mu, \sigma^2 I)$

재매개변수화 방식 (미분 가능):

- $\epsilon \sim N(0, I)$, 즉 표준 정규분포에서 샘플링 후
- $z = \mu + \sigma \cdot \epsilon$

이렇게 표현하면, μ, σ 에 대한 역전파가 가능해져서 신경망 매개변수를 효율적으로 학습할 수 있습니다.

그림 7-11 재매개변수화 트릭에 의해 달라지는 계산 그래프(역전파의 빨간 화살표는 기울기 전파를 나타냄)



7.4 VAE 구현

•VAE를 실제 코드로 어떻게 구현하는지 소개합니다.
•VAE의 핵심인 **인코더(Encoder)**와 **디코더(Decoder)** 신경망을 명확히 정의합니다.
•실제 MNIST 데이터셋을 사용해 VAE를 학습하고, 학습 후 새로운 데이터를 생성하는 방법을 설명합니다.

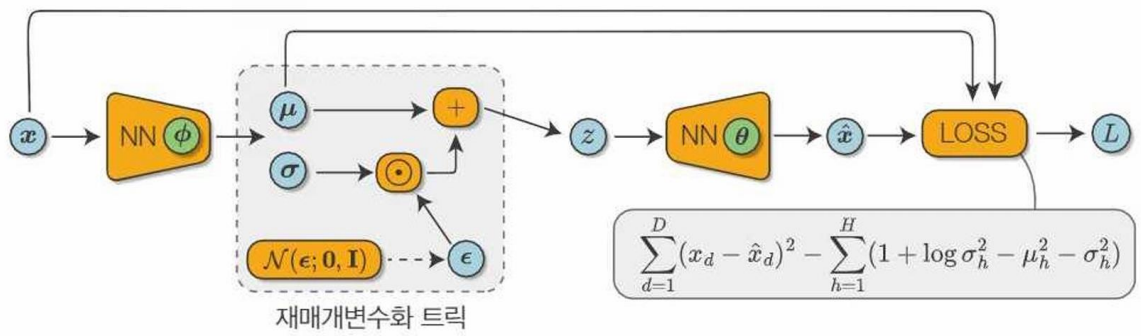
(1) VAE 구현 전략

VAE의 손실 함수(ELBO)는 다음과 같이 정의됩니다:

$$\mathrm{ELBO}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) \approx -\frac{1}{2}\sum_{d=1}^{D}(x_d-\hat{x}_d)^2+\frac{1}{2}\sum_{h=1}^{H}(1+\log\sigma_h^2-\mu_h^2-\sigma_h^2)+\mathrm{const}$$

$$\mathrm{Loss}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) \approx \sum_{d=1}^{D}(x_d-\hat{x}_d)^2-\sum_{h=1}^{H}(1+\log\sigma_h^2-\mu_h^2-\sigma_h^2)$$

그림 7-12 VAE 학습 시의 계산 그래프(손실 함수의 출력을 L로 가정)



- 이 계산 그래프를 이제부터 파이토치로 구현합니다. 사용하는 데이터셋은 MNIST입니다.
- MNIST는 1채널 (그레이스케일) 28X28 크기의 이미지 데이터입니다. 하지만 여기서는 이미지 데이터를 한줄로 정렬하여 크기가 784 (=1X28X28) 인 1차원 배열로 취급합니다.

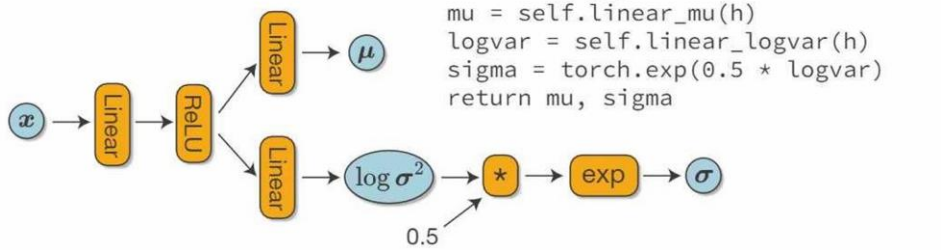
(2) 인코더와 디코더의 구조

```
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
from torchvision import datasets, transforms

# hyperparameters
input_dim = 784  # x dimension
hidden_dim = 200  # neurons in hidden layers
latent_dim = 20  # z dimension
epochs = 30
learning_rate = 3e-4
batch_size = 32

class Encoder(nn.Module):
    def __init__(self, input_dim, hidden_dim, latent_dim):
        super().__init__()
        self.linear = nn.Linear(input_dim, hidden_dim)
        self.linear_mu = nn.Linear(hidden_dim, latent_dim)
        self.linear_logvar = nn.Linear(hidden_dim,
latent_dim)

    def forward(self, x):
        h = self.linear(x)
        h = F.relu(h)
        mu = self.linear_mu(h)
        logvar = self.linear_logvar(h)
        sigma = torch.exp(0.5 * logvar)
        return mu, sigma

class Decoder(nn.Module):
    def __init__(self, latent_dim, hidden_dim, output_dim):
        super().__init__()
        self.linear1 = nn.Linear(latent_dim, hidden_dim)
        self.linear2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, z):
        h = self.linear1(z)
        h = F.relu(h)
        h = self.linear2(h)
        x_hat = F.sigmoid(h)
        return x_hat
```

인코더(Encoder)

- 입력 데이터 x를 받아서 잠재변수 z의 분포를 정의하는 두 매개변수 **평균(μ)**과 **표준편차(σ)**를 출력합니다.
- 실제 구현 시에는 신경망으로 다음을 출력합니다:
    - μ (잠재변수 평균)
    - $\log\sigma^2$ → σ (잠재변수 표준편차)

그림 7-13 인코더의 순전파 계산 그래프와 코드

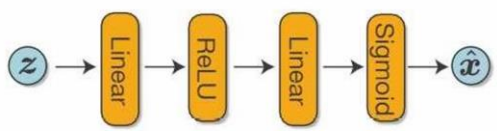



```
def forward(self, x):
    h = self.linear(x)
    h = F.relu(h)
    mu = self.linear_mu(h)
    logvar = self.linear_logvar(h)
    sigma = torch.exp(0.5 * logvar)
    return mu, sigma
```

디코더(Decoder)

- 인코더에서 얻은 잠재변수 z를 받아 원본 데이터 크기의 데이터($\hat{x}$)를 생성합니다.
- 출력 데이터는 일반적으로 Sigmoid 활성화 함수를 사용해 [0,1] 범위로 출력됩니다. (MNIST 이미지 [0,1] 정규화)

그림 7-14 디코더의 순전파 계산 그래프와 코드



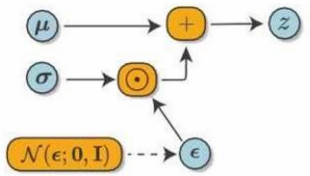

```
def forward(self, z):
    h = self.linear1(z)
    h = F.relu(h)
    h = self.linear2(h)
    x_hat = F.sigmoid(h)
    return x_hat
```

(3) 재매개변수화 트릭 코드 구현

샘플링을 미분 가능하게 만드는 **재매개변수화 트릭**은 다음처럼 구현합니다:

```
def reparameterize(mu, sigma):
    eps = torch.randn_like(sigma)
    z = mu + eps * sigma
    return z
```

그림 7-15 재매개변수화 트릭의 계산 그래프와 코드




```
def reparameterize(mu, sigma):
    eps = torch.randn_like(sigma)
    z = mu + eps * sigma
    return z
```

(4) 전체 손실 함수 코드 구현

```
class VAE(nn.Module):
    def __init__(self, input_dim, hidden_dim, latent_dim):
        super().__init__()
        self.encoder = Encoder(input_dim, hidden_dim, latent_dim)
        self.decoder = Decoder(latent_dim, hidden_dim, input_dim)

    def get_loss(self, x):
        mu, sigma = self.encoder(x)
        z = reparameterize(mu, sigma)
        x_hat = self.decoder(z)

        batch_size = len(x)
        L1 = F.mse_loss(x_hat, x, reduction='sum')        #1
        L2 = - torch.sum(1 + torch.log(sigma ** 2) - mu ** 2 - sigma ** 2) #2
        return (L1 + L2) / batch_size      #3
```

$$\text{Loss}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) \approx \underbrace{\sum_{d=1}^{D}(x_d-\hat{x}_d)^2}_{L_1} - \underbrace{\sum_{h=1}^{H}(1+\log\sigma_h^2-\mu_h^2-\sigma_h^2)}_{L_2}$$

이 식은 데이터 하나에 대한 손실 함수입니다. 여러 데이터를 미니배치로 처리한다면 이 식의 평균을 구해야 합니다. 코드에서는 ❶ F.mse_loss()의 인수로 reduction='sum'을 지정하여 L_1 항의 제곱 오차의 총합sum을 구했습니다. ❷ L_2 항에 대해서도 계산식에 따라 총합을 구합니다. ❸ 이렇게 손실 함수의 총합을 구하고 마지막으로 batch_size로 나누면 손실 함수의 평균을 구할 수 있습니다.

```
#1 데이터셋
# datasets
transform = transforms.Compose([
                transforms.ToTensor(),
                transforms.Lambda(torch.flatten) # falatten
            ])
dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
dataloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size, shuffle=True)

#2 모델과 옵티마이저
model = VAE(input_dim, hidden_dim, latent_dim)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
losses = []

#3 학습 루프
for epoch in range(epochs):
    loss_sum = 0.0
    cnt = 0

    for x, label in dataloader:
        optimizer.zero_grad()
        loss = model.get_loss(x)
        loss.backward()
        optimizer.step()

        loss_sum += loss.item()
        cnt += 1

    loss_avg = loss_sum / cnt
    print(loss_avg)
    losses.append(loss_avg)

# plot losses
epochs = list(range(1, epochs + 1))
plt.plot(epochs, losses, marker='o', linestyle='-')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```

(5) VAE 전체 학습 과정

VAE를 실제로 학습하는 과정은 다음과 같습니다:

1. MNIST 데이터 불러오기 (이미지 데이터는 [0,1]로 정규화 및 1차원 벡터로 변환)
2. 인코더와 디코더 모델 정의 및 옵티마이저(Adam) 설정
3. 에포크(epoch)마다 데이터 로더에서 미니배치를 추출해 학습 진행
4. 각 미니배치에 대해 손실을 계산 → 역전파 → 매개변수 갱신 반복
5. 손실을 지속적으로 모니터링하면서 학습 상태 확인

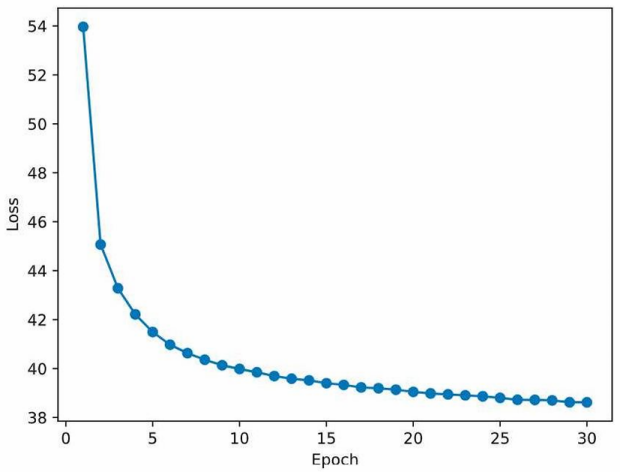
❶ 먼저 이미지 전처리 과정을 transform으로 정의합니다. 이미지를 텐서로 변환하고 다시 1차원 배열로 평탄화합니다. transforms.ToTensor()가 MNIST 데이터를 텐서로 변환하는데, 이때 각 원소는 0.0에서 1.0 사이로 정규화됩니다.

❷ 이어서 VAE 클래스의 인스턴스를 만들고 Adam 옵티마이저를 설정합니다.

❸에서는 지정한 에포크 수만큼 학습을 반복합니다. 에포크마다 데이터 로더에서 데이터를 일괄로 가져옵니다. 그리고 옵티마이저의 기울기를 초기화하고 VAE의 손실 함수를 계산하여 역전파로 기울기를 구합니다. 마지막으로 옵티마이저를 통해 모델의 매개변수를 갱신합니다.

이 코드를 실행하면 VAE가 학습되면서 에포크별 손실이 출력됩니다. 추이는 [그림 7-16]과 같습니다. 에포크가 진행될수록 손실이 작아짐을 알 수 있습니다.

그림 7-16 에포크별 손실 추이



(6) 학습 후 이미지 생성하기

이 코드는 VAE(변분 오토인코더)의 디코더를 사용하여 새로운 이미지를 생성하고, 이를 그리드 형태로 화면에 출력합니다.

1. 먼저 torch.no_grad()를 사용하여 기울기 계산을 비활성화합니다. 이 과정으로 메모리 사용량을 절약할 수 있습니다.
2. 지정된 잠재 공간(latent dimension)의 크기에 맞춰 64개의 무작위 잠재변수(z)를 생성합니다.
3. 생성된 잠재변수(z)를 모델의 디코더(decoder)에 입력하여 이미지 데이터(x)를 생성합니다.
4. 생성된 이미지 데이터(x)를 (64, 1, 28, 28)의 형태로 변환하여, 64장의 1채널(그레이스케일), 28×28 픽셀 이미지를 얻습니다.
5. 마지막으로 이미지를 그리드 형태로 정렬하여 화면에 출력합니다. 코드 실행 시 그리드 형태의 생성된 이미지가 나타납니다.

```
# generate new images
with torch.no_grad():
    sample_size = 64
    z = torch.randn(sample_size, latent_dim)
    x = model.decoder(z)
    generated_images = x.view(sample_size, 1, 28, 28)

grid_img = torchvision.utils.make_grid(generated_images, nrow=8, padding=2,
normalize=True)
plt.imshow(grid_img.permute(1, 2, 0))
plt.axis('off')
plt.show()
```