Diffusion Models Presentation

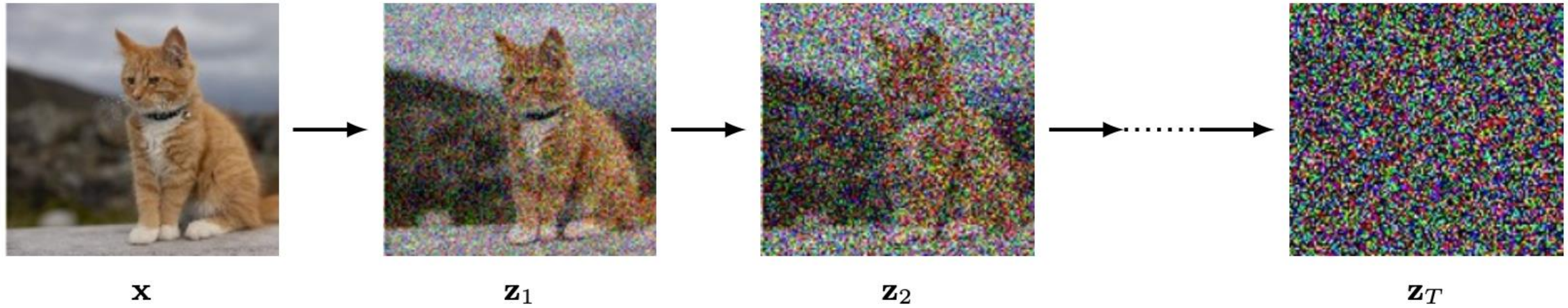
추성재, DGIST

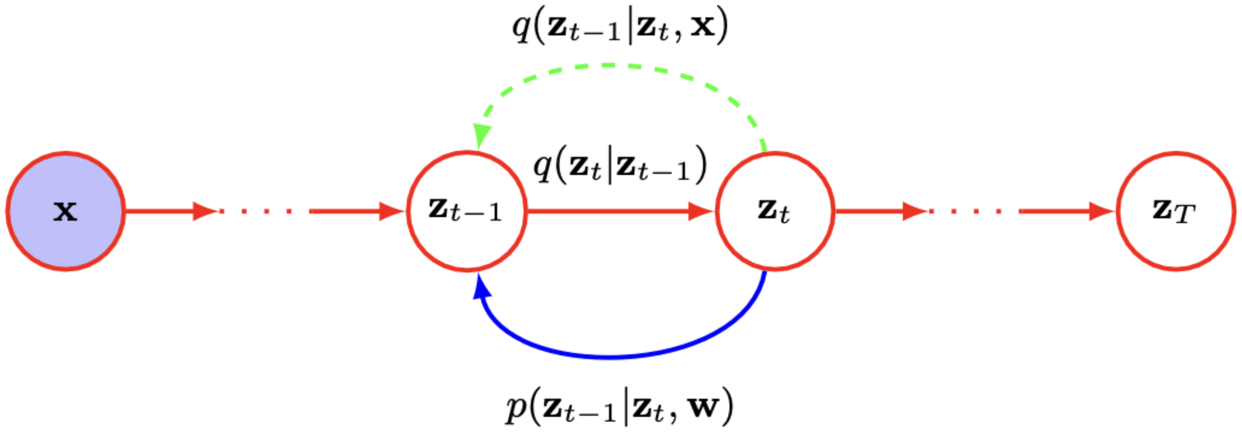
What is Diffusion Model?

- Diffusion Model or Denoising Diffusion Probabilistic Model (DDPM)

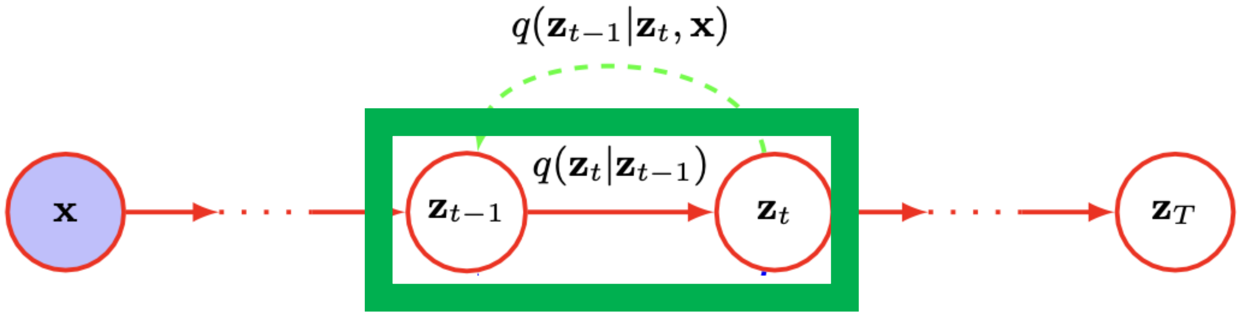
- Forward Encoder
  - corrupt training image using multi-step noising process
  - training image turn into sample of Gaussian distribution
- Reverse Decoder
  - deep neural network is trained to invert noising process
  - trained network can generate new images with new sample of Gaussian distribution

What is Diffusion Model?



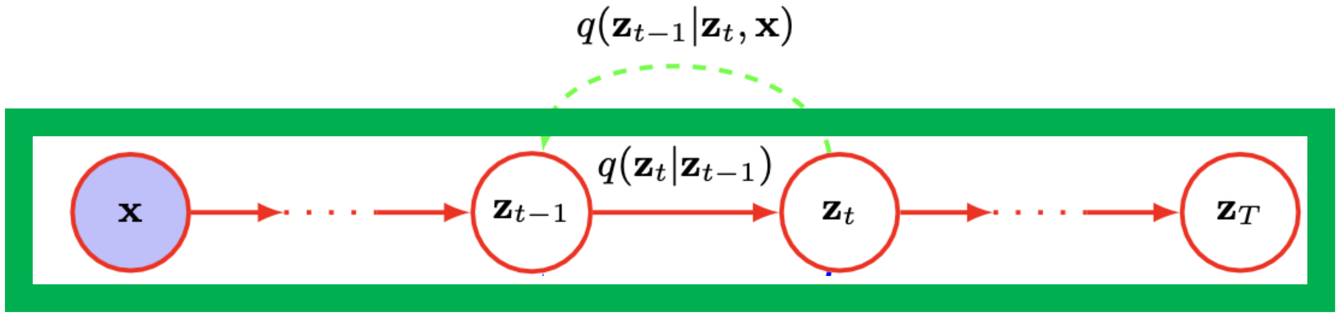


20.1 Forward Encoder



- Forward process step
  - Markov chain – current state only influenced by previous state
  - step: $\mathbf{z}_t = \sqrt{1-\beta_t}\mathbf{z}_{t-1} + \sqrt{\beta_t}\epsilon_t$
    where $\epsilon_t \sim \mathcal{N}(\epsilon_t|\mathbf{0}, \mathbf{I})$, $\beta_t \in (0,1)$, $\beta_1 < \beta_2 < \ldots < \beta_T$
  - can be rewrite in : $q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t|\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I})$

20.1 Forward Encoder – Diffusion kernel



- Diffusion kernel :

$$q(\mathbf{z_t}|\mathbf{x}) = \mathcal{N}(\mathbf{z_t}|\sqrt{\alpha_t}\mathbf{x}, (1-\alpha_t)\mathbf{I}$$

$$\text{where } \alpha_t = \Pi_{\tau=1}^{t}(1-\beta_\tau)$$

$$\text{can be written like } \mathbf{z}_t = \sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\epsilon_t$$

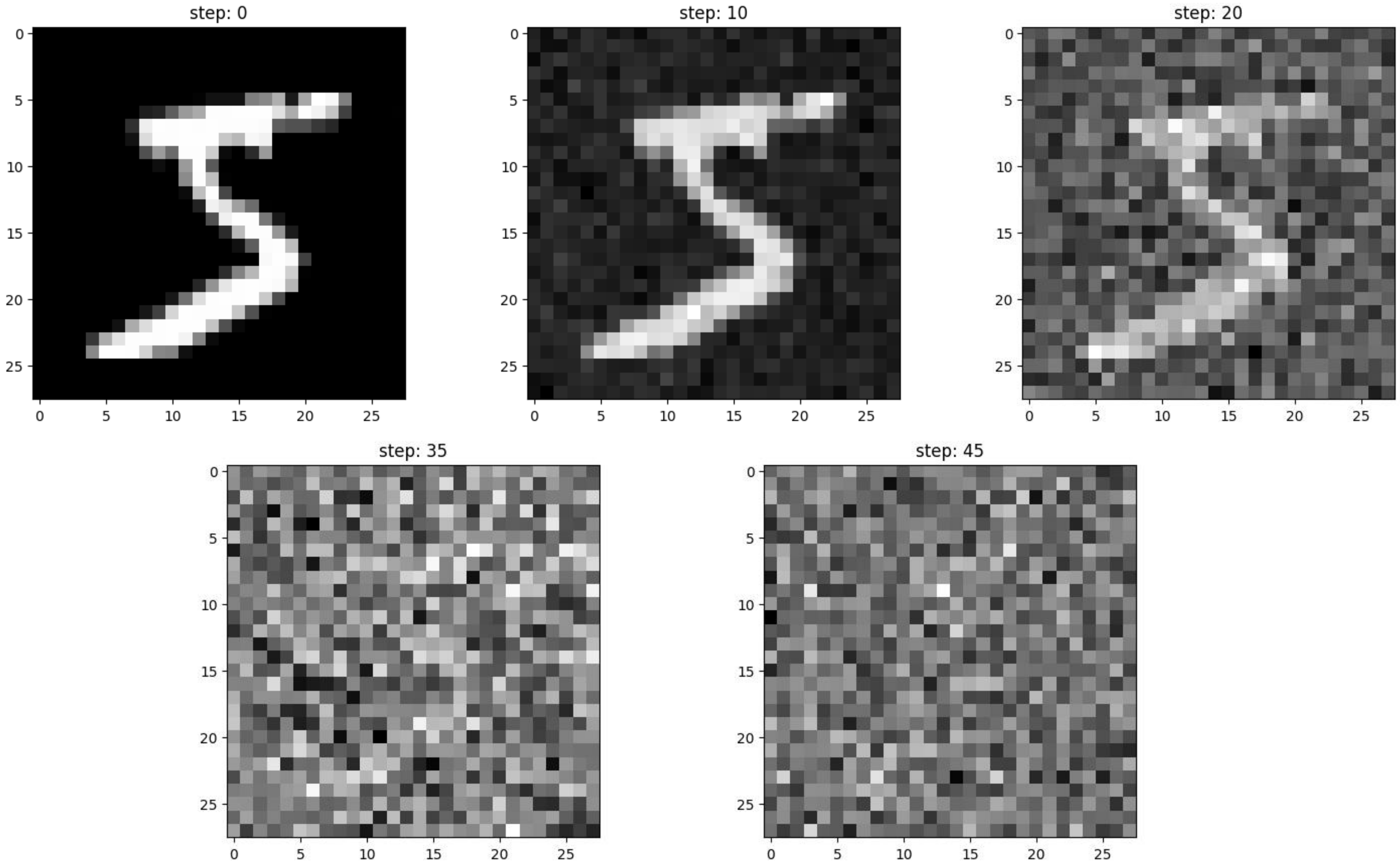
$$\text{if } T \to \infty,\ q(\mathbf{z_t}|\mathbf{x}) = \mathcal{N}(\mathbf{z}_T|\mathbf{0}, \mathbf{I})$$

20.1 Forward Encoder – Pytorch example

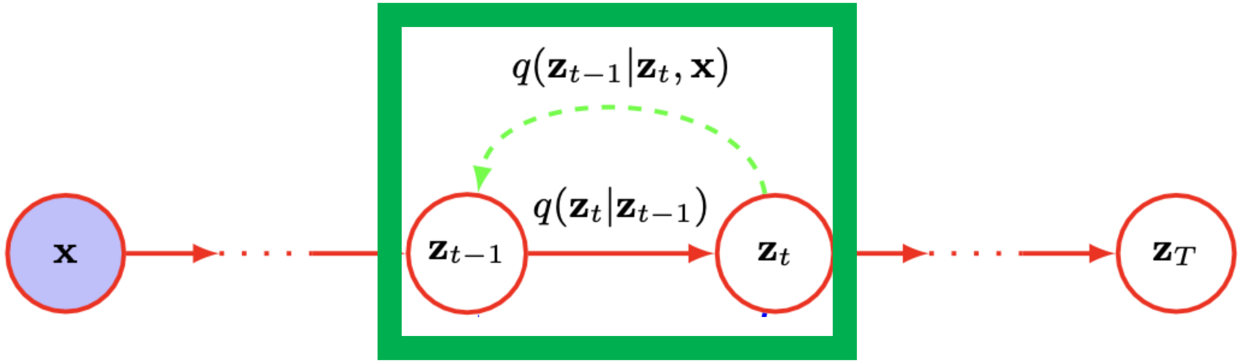
```
def cal_alpha(beta, step):
    result = torch.tensor(1).float()

    for i in range(0, step+1):
        tmp = beta[0, i]
        result *= (1-tmp)

    return result
```

```
# 점진적 forward 과정

for t in range(0, terminal_step, 5):
    noise_t = torch.randn(28,28)
    alpha_t = cal_alpha(beta, t)

    # forward 과정
    z_t = forward_image * torch.sqrt(alpha_t) + noise_t * (1-torch.sqrt(alpha_t))
    plt.figure()
    plt.imshow(z_t, cmap='gray')
```

20.1 Forward Encoder – Pytorch example



20.1 Forward Encoder – Conditional distribution



$$q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}) = q(\mathbf{z}_t|\mathbf{z}_{t-1})$$

exponential of quadratic form

-> can be turn into normal distirbution

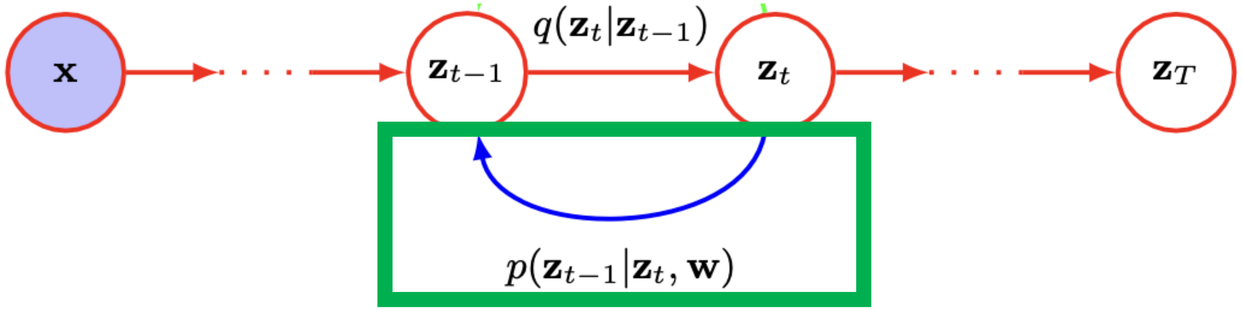
$$q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) = \frac{q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x})\, q(\mathbf{z}_{t-1}|\mathbf{x})}{\cancel{q(\mathbf{z}_t|\mathbf{x})}}$$

can be ignore

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{z}) = \mathcal{N}(\mathbf{z}_{t-1}|\mathbf{m}(\mathbf{x}, \mathbf{z}_t), \sigma_t^2\mathbf{I})$$

diffusion kernel: normal distribution

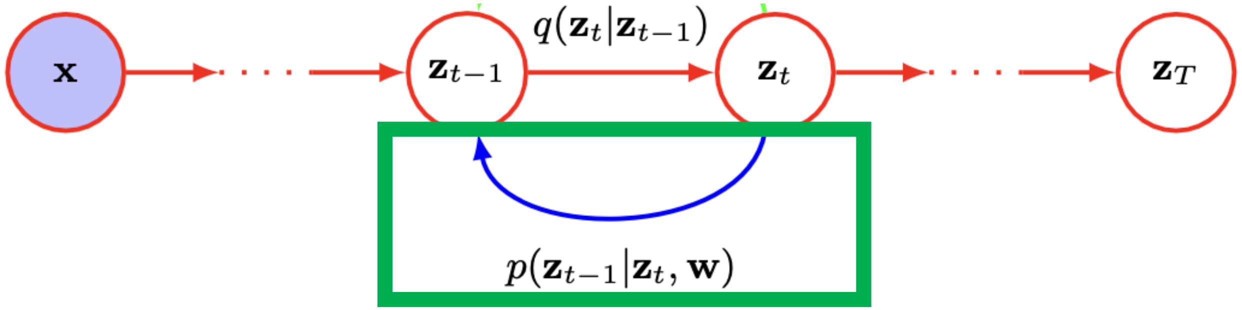
20.2 Reverse Decoder



- Instead of solving $q(\mathbf{z}_{t-1}|\mathbf{z}_t)$, we learn approximation of reverse distribution: $p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w}) = \mathcal{N}(\mathbf{z}_{t-1}|\mu(\mathbf{z}_t, \mathbf{w}, t), \beta_t \mathbf{I})$
- $\mu(\mathbf{z}_t, \mathbf{w}, t)$ is deep neural network with parameter $\mathbf{w}$
- Main assumption :

$$\beta_t << 1 \text{ so that } q(\mathbf{z}_{t-1}|\mathbf{z}_t)$$
$$\text{approximately Gaussian distribution over } \mathbf{z}_{t-1}$$

20.2 Reverse Decoder



- $\mu(\mathbf{z}_t, \mathbf{w}, t)$ can invert total step of forward process
- input dimension and output dimension of $\mu(\mathbf{z}_t, \mathbf{w}, t)$ have to be same, so U-net architecture is commonly used.
- total denoising process:

$$p(\mathbf{x}, \mathbf{z}_1, \ldots, \mathbf{z}_T|\mathbf{w}) = p(\mathbf{z}_T)\left\{\prod_{t=2}^{T} p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})\right\} p(\mathbf{x}|\mathbf{z}_1, \mathbf{w}). \qquad (20.19)$$

20.2 Reverse Decoder – Evidence Lower Bound (ELBO)

- Total denoising process is intractable: process involve integrating over the neural network functions
- We use Evidence Lower Bound (ELBO)
  - for all distribution $q(\mathbf{z})$, following relation always holds

$$\ln p(\mathbf{x}|\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \mathrm{KL}\left(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x},\mathbf{w})\right)$$

  - Where Kullback-Leibler divergence term is always bigger than 0, we can get following relation

$$\ln p(\mathbf{x}|\mathbf{w}) \geq \mathcal{L}(\mathbf{w})$$

  - Why KL-divergence is always bigger than 0?

$$p(\mathbf{z}|\mathbf{x},\mathbf{w}) \text{ and } q(z) \text{ is different.}$$
$$\text{so } \mathrm{KL}(q(z)||p(\mathbf{z}|\mathbf{x},\mathbf{w})) \geq 0$$

- Our objective : train neural network to maximize ELBO

20.2 Reverse Decoder – Evidence Lower Bound (ELBO)

Evidence Lower Bound (ELBO)는 계산이 용이한 임의의 가우시안 분포 $q(z)$를 가지고 계산이 불가능한 $\log p_\theta(x)$(Evidence)의 lower bound를 최대화하는 방식으로 $\log p_\theta(x)$을 근사하는 변분 추론 방법입니다.

$$\log p_\theta(x) \geq \sum_z q(z) \log \frac{p_\theta(x,z)}{q(z)}$$

$$= \sum_z q(z) \log p_\theta(x,z) - \sum_z q(z) \log q(z)$$

$$= \sum_z q(z) \log p_\theta(x,z) + H(q)$$

$$\therefore ELBO(q) = \sum_z q(z) \log p_\theta(x,z) + H(q)$$

출처 : 생성모델 입문서 - WikiDocs

20.2 Reverse Decoder – Evidence Lower Bound (ELBO)

KL-Divergence

임의의 두 확률 분포 $p(x)$와 $q(x)$에 대해 KL-Divergence는 다음 성질을 만족합니다.

$$D_{KL}(p||q) = \sum_{x} p(x) log \frac{p(x)}{q(x)}$$

- $D_{KL}(p||q) \geq 0$: 두 분포간 차이는 존재해야 합니다 ($\not< 0$)
- $D_{KL}(p||q) = 0$ if $p = q$: 두 분포가 같다면 KLD는 0입니다.
- $D_{KL}(p||q) \neq D_{KL}(q||p)$: 교환법칙은 성립하지 않습니다.

출처 : [생성모델 입문서 - WikiDocs]

20.2 Reverse Decoder – Evidence Lower Bound (ELBO)

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_q\left[\ln \frac{p(\mathbf{z}_T)\left\{\prod_{t=2}^{T} p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})\right\} p(\mathbf{x}|\mathbf{z}_1,\mathbf{w})}{q(\mathbf{z}_1|\mathbf{x})\prod_{t=2}^{T} q(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{x})}\right]$$

can be ignored:
has no $\mathbf{w}$

$$= \mathbb{E}_q\left[\cancel{\ln p(\mathbf{z}_T)} + \sum_{t=2}^{T} \ln \frac{p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})}{q(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{x})} - \cancel{\ln q(\mathbf{z}_1|\mathbf{x})} + \ln p(\mathbf{x}|\mathbf{z}_1,\mathbf{w})\right] \qquad (20.26)$$

can be ignored:
has no $\mathbf{w}$

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_q\left[\sum_{t=2}^{T} \ln \frac{p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})}{q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})} + \ln p(\mathbf{x}|\mathbf{z}_1,\mathbf{w})\right]$$

$$\mathcal{L}(\mathbf{w}) = \underbrace{\int q(\mathbf{z}_1|\mathbf{x}) \ln p(\mathbf{x}|\mathbf{z}_1,\mathbf{w})\,\mathrm{d}\mathbf{z}_1}_{\text{reconstruction term}} \underbrace{- \sum_{t=2}^{T} \int \mathrm{KL}(q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x}) \| p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})) q(\mathbf{z}_t|\mathbf{x})\,\mathrm{d}\mathbf{z}_t}_{\text{consistency terms}} \qquad (20.32)$$

20.2 Reverse Decoder – Evidence Lower Bound (ELBO)

$$\mathcal{L}(\mathbf{w}) = \underbrace{\int q(\mathbf{z}_1|\mathbf{x}) \ln p(\mathbf{x}|\mathbf{z}_1, \mathbf{w})\, \mathrm{d}\mathbf{z}_1}_{\text{reconstruction term}}$$

$$\underbrace{- \sum_{t=2}^{T} \int \mathrm{KL}(q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) \| p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})) q(\mathbf{z}_t|\mathbf{x})\, \mathrm{d}\mathbf{z}_t}_{\text{consistency terms}} \qquad (20.32)$$

$$\mathrm{KL}(q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) \| p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w}))$$
$$= \frac{1}{2\beta_t} \|\mathbf{m}_t(\mathbf{x}, \mathbf{z}_t) - \boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t)\|^2 + \mathrm{const}$$

- KL-divergence term acts like squared-loss funtion
- minimizing squared error to maximize ELBO

20.2 Reverse Decoder – Predicting the noise

- We introduce new neural network $\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)$, that predicts total noise added to $\mathbf{x}$ to generate $\mathbf{z}_t$

$$\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) = \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)\right\}$$

- with several process, we can get

$$\mathcal{L}(\mathbf{w}) = -\sum_{t=1}^{T}\left\|\mathbf{g}(\sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}_t\right\|^2$$

Algorithm 20.1: Training a denoising diffusion probabilistic model

Input: Training data $\mathcal{D} = \{\mathbf{x}_n\}$
Noise schedule $\{\beta_1, \ldots, \beta_T\}$
Output: Network parameters $\mathbf{w}$

for $t \in \{1, \ldots, T\}$ **do**
  $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1-\beta_\tau)$ `// Calculate alphas from betas`
end for
repeat
  $\mathbf{x} \sim \mathcal{D}$ `// Sample a data point`
  $t \sim \{1, \ldots, T\}$ `// Sample a point along the Markov chain`
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ `// Sample a noise vector`
  $\mathbf{z}_t \leftarrow \sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}$ `// Evaluate noisy latent variable`
  $\mathcal{L}(\mathbf{w}) \leftarrow \|\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}\|^2$ `// Compute loss term`
  Take optimizer step
until converged
return $\mathbf{w}$

20.2 Reverse Decoder – Generating new sample

Algorithm 20.2: Sampling from a denoising diffusion probabilistic model

Input: Trained denoising network $\mathbf{g}(\mathbf{z}, \mathbf{w}, t)$
Noise schedule $\{\beta_1, \ldots, \beta_T\}$
Output: Sample vector $\mathbf{x}$ in data space

$\mathbf{z}_T \sim \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ `// Sample from final latent space`
for $t \in T, \ldots, 2$ **do**
  $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1-\beta_\tau)$ `// Calculate alpha`
  `// Evaluate network output`
  $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) \leftarrow \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)\right\}$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ `// Sample a noise vector`
  $\mathbf{z}_{t-1} \leftarrow \boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) + \sqrt{\beta_t}\boldsymbol{\epsilon}$ `// Add scaled noise`
end for
$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\left\{\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}}\mathbf{g}(\mathbf{z}_1, \mathbf{w}, t)\right\}$ `// Final denoising step`
return $\mathbf{x}$

20.3 Score Matching

- Diffusion model is highly related with "score matching" deep generative model

- "score matching" model uses "score function" or "Stein score"

$$\mathbf{s}(\mathbf{x}) = \nabla_{\mathbf{x}} \ln p(\mathbf{x})$$

- $\mathbf{s}(\mathbf{x})$ has same dimensionality as $\mathbf{x}$

20.3 Score Matching – score loss function

- loss function to learn $\mathbf{s}(\mathbf{x}, \mathbf{w})$

$$J(\mathbf{w}) = \frac{1}{2} \int \|\mathbf{s}(\mathbf{x}, \mathbf{w}) - \nabla_{\mathbf{x}} \ln p(\mathbf{x})\|^2 \, p(\mathbf{x}) \, \mathrm{d}\mathbf{x}$$

- Problem of this loss function is we do not know $\nabla_{\mathbf{x}} \ln p(\mathbf{x})$
- So, we need to use data points “smear out” from the noise kernel (Parzen estimator)

$$q_{\sigma}(\mathbf{z}) = \int q(\mathbf{z}|\mathbf{x}, \sigma) p(\mathbf{x}) \, \mathrm{d}\mathbf{x} \qquad (20.47)$$

where $q(\mathbf{z}|\mathbf{x}, \sigma)$ is the *noise kernel*. A common choice of kernel is the Gaussian

$$q(\mathbf{z}|\mathbf{x}, \sigma) = \mathcal{N}(\mathbf{z}|\mathbf{x}, \sigma^2 \mathbf{I}). \qquad (20.48)$$

20.3 Score Matching – score loss function

- Change the loss function:

$$J(\mathbf{w}) = \frac{1}{2}\iint \left\|\mathbf{s}(\mathbf{z},\mathbf{w}) - \nabla_{\mathbf{z}} \ln q(\mathbf{z}|\mathbf{x},\sigma)\right\|^2 q(\mathbf{z}|\mathbf{x},\sigma)p(\mathbf{x})\,\mathrm{d}\mathbf{z}\,\mathrm{d}\mathbf{x} + \text{const.}$$

- If we choose noise kernel as Gaussian kernel, $\nabla_{\mathbf{z}} \ln q(\mathbf{z}|\mathbf{x},\sigma) = -\frac{1}{\sigma}\epsilon$
- Then, loss function measures difference between neural network and noise
- Therefore, this loss function has same minimum as diffusion model's loss function : $\mathbf{s}(\mathbf{x},\mathbf{w}) \sim -\frac{1}{\sigma}\mathbf{g}(\mathbf{z},\mathbf{w})$
- Langevin dynamics is used to sample from score function

20.3 Score Matching – Noise variance

- Score function has three problems
  - If probability density is zero, score function is undefined
  - If data density is low, estimation of score function is inaccurate
  - If data distribution is mixture of disjoint distribution, Langevin procedure might not sample properly
- All problems can be solved with choosing large noise variance σ^2, but this might cause distortion of original distribution
- This can be handled with sequence of noise variance which follows

$$\sigma_1^2 < \sigma_2^2 < \ldots < \sigma_T^2$$

- score network is modified to get noise variance

$$\mathbf{s}(\mathbf{x}, \mathbf{w}, \sigma^2)$$

20.4 Guided Diffusion

- In many applications, we want diffusion models to generate image with certain "conditions"
  - conditions could be label or text description (prompt)

- most simple approach is to add conditioning variable $\mathbf{c}$ inside denoising neural network $\mathbf{g}(\mathbf{z}, \mathbf{w}, t, \mathbf{c})$ and train network with $\{\mathbf{x}_n, \mathbf{c}_n\}$

- In this approach, we need "guidance" which is weight given to conditioning variable

20.4 Guided Diffusion – Classifier guidance

- If there are trained classifier $p(\mathbf{c}|\mathbf{x})$, and there are diffusion model using score function, there are relation that

$$\nabla_{\mathbf{x}} \ln p(\mathbf{x}|\mathbf{c}) = \nabla_{\mathbf{x}} \ln \left\{ \frac{p(\mathbf{c}|\mathbf{x})p(\mathbf{x})}{\cancel{p(\mathbf{c})}} \right\}$$

$$\nabla_{\mathbf{x}} \ln p(\mathbf{c}) = 0$$

$$= \nabla_{\mathbf{x}} \ln p(\mathbf{x}) + \nabla_{\mathbf{x}} \ln p(\mathbf{c}|\mathbf{x})$$

diffusion model
using score function

makes denoising process
maximize probability of $\mathbf{c}$

- add hyperparameter λ (guidance scale), score function becomes

$$\text{score}(\mathbf{x}, \mathbf{c}, \lambda) = \nabla_{\mathbf{x}} \ln p(\mathbf{x}) + \lambda \nabla_{\mathbf{x}} \ln p(\mathbf{c}|\mathbf{x}).$$

- drawback: need to train separate classifier $p(\mathbf{c}|\mathbf{x})$

20.4 Guided Diffusion – Classifier-free guidance

- If we change score function of "Classifier guidance"

$$\mathrm{score}(\mathbf{x}, \mathbf{c}, \lambda) = \lambda \nabla_{\mathbf{x}} \ln p(\mathbf{x}|\mathbf{c}) + (1-\lambda) \nabla_{\mathbf{x}} \ln p(\mathbf{x})$$

- We can avoid training separate networks. ($p(\mathbf{x}) = p(\mathbf{x}|\mathbf{c} = \mathbf{0})$)

- This approach is used in many guided diffusion models
  - Text-guided diffusion model : using prompt (text sequence) as conditioning variable
  - image super-resolution : denoising high-resolution image with using low-resolution image as conditioning variable

20.4 Guided Diffusion – Stable Diffusion