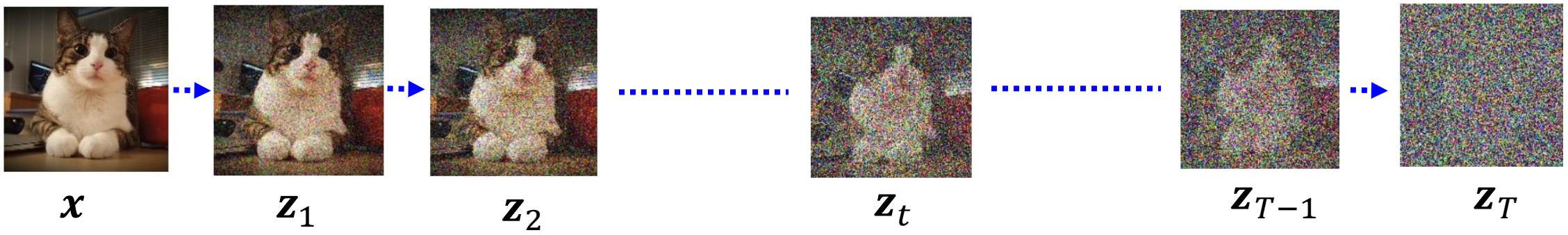
Deep Generative Models (contd)

CS772A: Probabilistic Machine Learning

Piyush Rai

Denoising Diffusion Models

- Consider gradually corrupting an image ($\boldsymbol{z_0 = x}$) till it becomes pure noise ($\boldsymbol{z_T}$)



- Each step $\boldsymbol{z_{t-1} \rightarrow z_t}$ is a pre-defined Gaussian perturbation (forward process)

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t|\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I})$$

$$\mathbf{z}_t = \sqrt{1-\beta_t}\mathbf{z}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon} \quad (\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}))$$

$\beta_t \in (0,1)$ and $\beta_1 < \beta_2 < \cdots \beta_{T-1} < \beta_T$

Usually pre-defined but can also be learned

implies

Imp: Thus we can also compute $\boldsymbol{z_t}$ from $\boldsymbol{x}$ directly in a single step

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

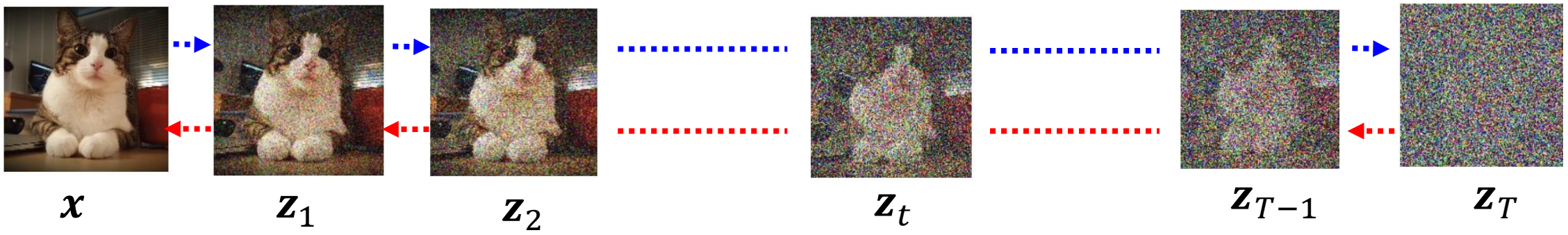
where $\alpha_t = \prod_{\tau=1}^{t}(1-\beta_\tau)$

$$\mathbf{z}_t = \sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}$$

$$q(\mathbf{z}_T|\mathbf{x}) = \mathcal{N}(\mathbf{z}_T|\mathbf{0}, \mathbf{I}) \quad \text{as } T \rightarrow \infty$$

Generating Data by Reversing Diffusion

- Reversing the diffusion (red arrows) would enable generating data from pure noise



- To reverse the diffusion, we need the distribution of z_{t-1} given z_t, i.e., $q(z_{t-1}|z_t)$

The denoising distribution

Intractable because $q(z_t)$ and $q(z_{t-1})$ are difficult to compute

$$q(z_{t-1}|z_t) = \frac{q(z_{t-1})\,q(z_t|z_{t-1})}{q(z_t)}$$

$$q(z_t) = \int q(z_t|x)p(x)dx$$

Since the true data distribution $p(x)$ is not known, we can't compute this integral

Towards a Tractable Reverse Diffusion

- Although $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$ isn't tractable, the following distribution is tractable

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

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

- Reason: $q(\boldsymbol{z}_{t-1}|\boldsymbol{x})$ and $q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$ are Gaussians, so $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$ is Gaussian

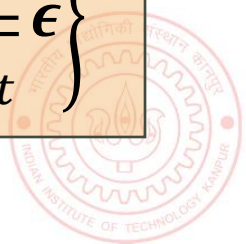
$$q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}_{t-1}|m(\boldsymbol{x}, \boldsymbol{z}_t), \sigma_t^2 \mathbf{I})$$

$$m(\boldsymbol{x}, \boldsymbol{z}_t) = \frac{(1-\alpha_{t-1})\sqrt{1-\beta_t}\boldsymbol{z}_t + \sqrt{\alpha_{t-1}}\beta_t \boldsymbol{x}}{1-\alpha_t}$$

$$\sigma_t^2 = \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t}$$

Using $\boldsymbol{x} = \frac{1}{\sqrt{\alpha_t}}\boldsymbol{z}_t - \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}}\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

$$= \frac{1}{\sqrt{1-\beta_t}}\left\{\boldsymbol{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}\right\}$$



Towards a Tractable Reverse Diffusion

- We saw that the reverse diffusion distribution is the Gaussian

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

where $$m(\boldsymbol{x}, \boldsymbol{z}_t) = \frac{1}{\sqrt{1-\beta_t}}\left\{\boldsymbol{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}\right\}$$

Issue: At generation time, we don't have $\boldsymbol{x}$ (the goal is to generate $\boldsymbol{x}$ which is only available for training data) so we can't use $m(\boldsymbol{x}, \boldsymbol{z}_t)$ at generation time since it depends on $\boldsymbol{x}$

- Let's approximate $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$ by another Gaussian that doesn't depend on $\boldsymbol{x}$

$$p(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{w}) = \mathcal{N}(\boldsymbol{z}_{t-1}|\mu(\boldsymbol{z}_t, \boldsymbol{w}, t), \Sigma(\boldsymbol{z}_t, \boldsymbol{w}, t))$$

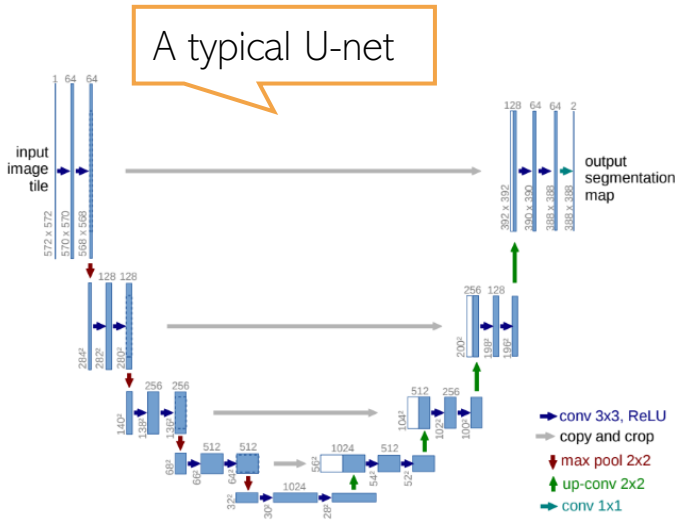
- Usually, $\Sigma(\boldsymbol{z}_t, \boldsymbol{w}, t)$ is chosen to be spherical. A popular choice: $\Sigma(\boldsymbol{z}_t, \boldsymbol{w}, t) = \beta_t \mathbf{I}$
- The mean $\mu(\boldsymbol{z}_t, \boldsymbol{w}, t)$ is defined to mimic the form of $m(\boldsymbol{x}, \boldsymbol{z}_t)$

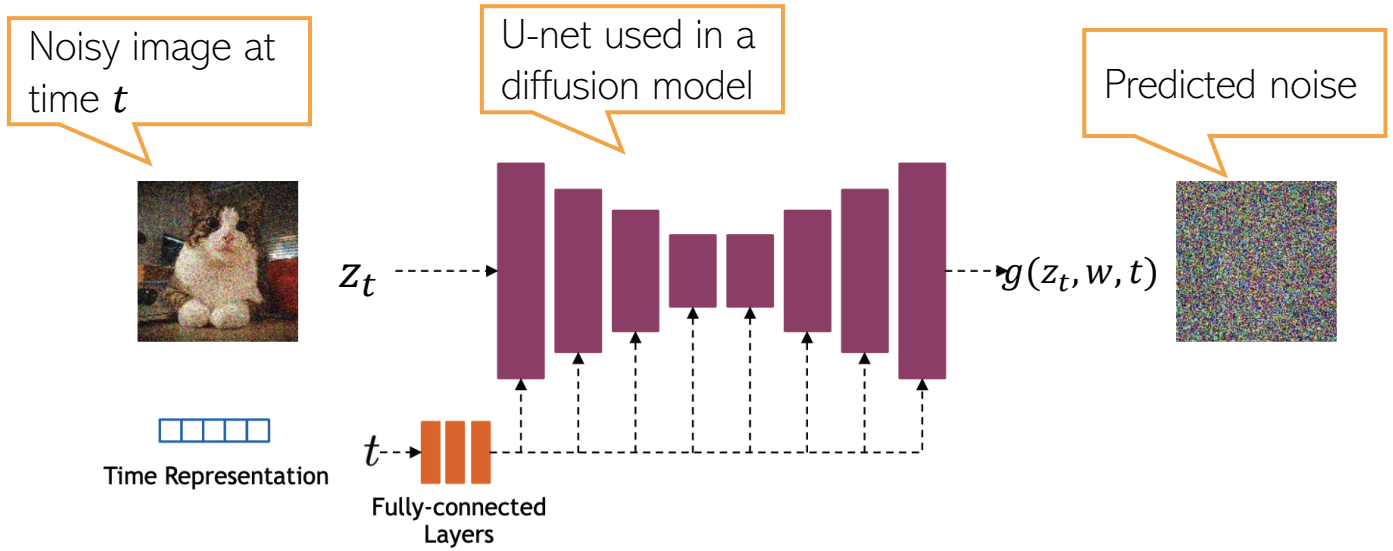
$$\mu(\boldsymbol{z}_t, \boldsymbol{w}, t) = \frac{1}{\sqrt{1-\beta_t}}\left\{\boldsymbol{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} g(\boldsymbol{z}_t, \boldsymbol{w}, t)\right\}$$

Our goal will be to model $g()$ as a neural network which predicts the noise $\boldsymbol{\epsilon}$ which was added to $\boldsymbol{x}$ to get its noisy version $\boldsymbol{z}_t$

Noise Predictor Network

- A "U-net" model (a neural net) is commonly used as the noise predictor network





- An embedding (positional embedding) of the time-step t is fed into the residual blocks of the U-net architecture

Figure source: Arash Vahdat

Denoising Diffusion Model: The Training Algo

- The overall training algo is as follows

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 $\mathbf{w}$

Pseudo-code from Bishop & Bishop (2023)

Denoising Diffusion Model: Generation

- Using the training model, we can now generate data as follows

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 $\mathbf{x}$

Generation can be slow because it requires several steps

Reducing the number of steps is an active area of research

One such approach is DDIM (denoising diffusion implicit model) which relaxes the Markov assumption in the noise process

Pseudo-code from Bishop & Bishop (2023)

Score based deep generative models

- For a probability distribution $p(\boldsymbol{x})$ its score function is defined as

$$s(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$$

Note: Here, this gradient is w.r.t. $\boldsymbol{x}$ and not w.r.t. the parameters of the distribution

- Assuming $p(\boldsymbol{x})$ as a target distribution, we can use SGLD to generate data samples as

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \frac{\delta}{2} \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}_t) + \sqrt{\delta} \epsilon_t \quad \text{where } \epsilon_t \sim \mathcal{N}(0, I)$$

- But doing so requires the score function $s(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$
- Since $p(\boldsymbol{x})$ itself is not known, how do get the score function $s(\boldsymbol{x})$?
- We can train a neural network to model the score function
- The score based approach is also helpful in "guided" or conditional generation
  - Example: Want to generate $\boldsymbol{x}$ while conditioning on some signal $\boldsymbol{c}$ (e.g., class label or texual description of the input to be generated). Score function: $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}|\boldsymbol{c})$

Diffusion Models via Score Matching

- Learning the score function will enable learning the distribution $p(x)$
- We can parameterize the score function as $s(x) = s(x, w)$ and define a loss function

Can define it as a neural network

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

- The distribution $p(x)$ isn't known but we only have a dataset $\mathcal{D}$ of N samples

However, this is non-differentiable function of x and thus can't use it in the minimization of $J(w)$

A discrete distribution represented by the N samples from the dataset

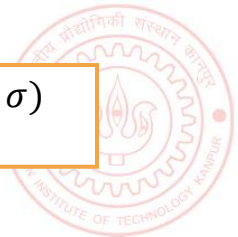
$$p_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N}\delta(\mathbf{x} - \mathbf{x}_n)$$

- Let's define a "smoothened" version of the distribution $p(x)$

Basically a convolution of $p(x)$ by a kernel $q(z|x,\sigma)$

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

One option is to define $q(z|x,\sigma)$ as a Gaussian $\mathcal{N}(z|x,\sigma^2 I)$



Score Matching

- Using this $q_\sigma(z)$ instead of $p(x)$, we can define the "score loss" function as

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

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

- Using the N samples from the dataset, the empirical loss will be

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\int \|\mathbf{s}(\mathbf{z},\mathbf{w}) - \nabla_{\mathbf{z}} \ln q(\mathbf{z}|\mathbf{x}_n,\sigma)\|^2 q(\mathbf{z}|\mathbf{x}_n,\sigma)\,\mathrm{d}\mathbf{z} + \text{const}$$

Thus the score function $s(z,w)$ is also like a noise predictor

- Choosing $q(z|x,\sigma) = \mathcal{N}(z|x,\sigma^2 I)$, we get $\nabla_{\mathbf{z}} \ln q(\mathbf{z}|\mathbf{x},\sigma) = -\frac{1}{\sigma}\epsilon$ where $\epsilon = x - z$
- Note the similarity with denoising diffusion model where

$$q(z_t|x) = \mathcal{N}(z|\sqrt{\alpha_t}x, (1-\alpha_t)I) \quad \Rightarrow \quad \nabla_{\mathbf{z}} \ln q(\mathbf{z}|\mathbf{x},\sigma) = -\frac{1}{\sqrt{1-\alpha_t}}\epsilon$$

- Thus the score function $s(z,w)$ plays a similar role as noise predictor $g(z,w,t)$

Guided Diffusion

- Often we want to generate data based on some "reference" conditioning signal, e.g.,
  - Images of a specific class (class-conditional generation)
  - Images based on some textual description
  - High resolution image using a low-resolution image (image "super-resolution")

Conditioning signal: "stained glass window of a panda eating bamboo"

Conditioning signal: Low-res image on the left

- Denoting the data as $\boldsymbol{x}$ and the conditioning signal as $\boldsymbol{c}$, we want to learn $\boldsymbol{p(x|c)}$

Figure source: DLFC (Bishop and Bishop, 2023)

Classifier Guidance

- Assume we have an already trained a classifier of the form $p(c|x)$
- We can then define the score function of a conditional diffusion model as

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

- We can also control the contribution of the classifier by defining the score function as

Score function of unconditional diffusion model

Classifier guidance term

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

- Large λ will encourage generation of x which respects the conditioning signal c
- However, this approach requires a classifier trained on noisy images

Classifier-free Guidance

- Recall the score function in classifier guidance method

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

- To eliminate the classifier term $\nabla_x \log p(c|x)$, use the fact that

$$\begin{aligned}\nabla_{\mathbf{x}} \ln p(\mathbf{x}|\mathbf{c}) &= \nabla_{\mathbf{x}} \ln \left\{ \frac{p(\mathbf{c}|\mathbf{x}) p(\mathbf{x})}{p(\mathbf{c})} \right\} \\ &= \nabla_{\mathbf{x}} \ln p(\mathbf{x}) + \nabla_{\mathbf{x}} \ln p(\mathbf{c}|\mathbf{x})\end{aligned}$$

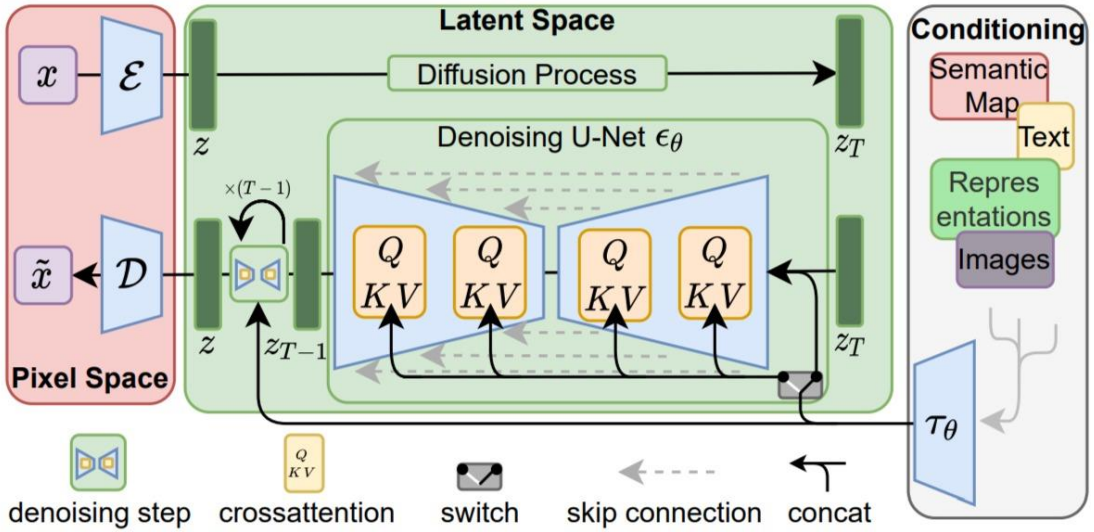
- Thus we can rewrite the score function as follows

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

- No need to train a separate classifier $p(c|x)$
- Also, no need to train both $p(x)$ and $p(x|c)$
  - Just train $p(x|c)$ using a score-function based approach and use $p(x|c = 0) = p(x)$

Latent Diffusion Models (LDM)

- Defines diffusion process in a latent space instead of in data (e.g., pixel) space
- The popular "Stable Diffusion" is based on LDM
- Diffusion process in a low-dim latent space is also more efficient computationally



- Can also condition the generation on other modalities such as text

*High-Resolution Image Synthesis with Latent Diffusion Models (Rombach et al, 2022)

Summary

- Diffusion Models (denoising diffusion models, score based models, etc) are currently the best performing methods
- A lot of ongoing work on diffusion models, e.g.,
  - Improving quality of generation
  - Speeding-up generation
  - Combining them with other generative models (e.g., large language models)