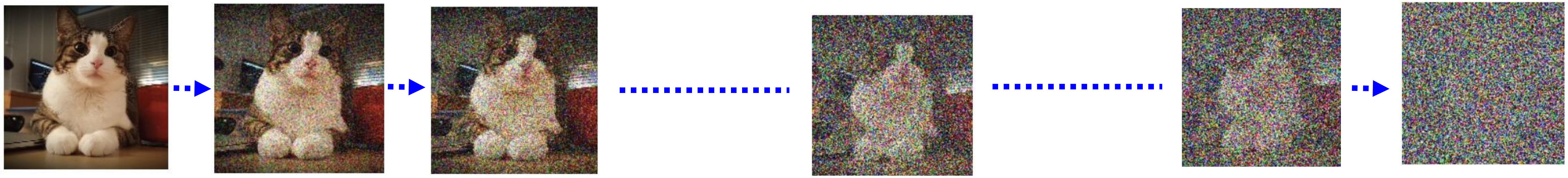
Denoising Diffusion Models (contd)

CS772A: Probabilistic Machine Learning

Piyush Rai

Denoising Diffusion Models

- Consider gradually corrupting an image ($\boldsymbol{z}_0 = \boldsymbol{x}$) till it becomes pure noise ($\boldsymbol{z}_T$)



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

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

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

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

Usually pre-defined but can also be learned

implies

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

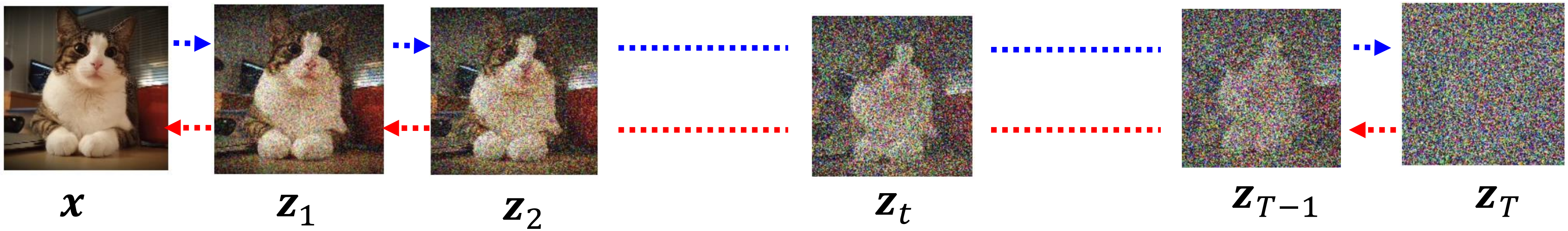
where $\alpha_t = \prod_{\tau=1}^{t}(1-\beta_\tau)$

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

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

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., $q(z_{t-1}|z_t)$

Intractable

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

Tractable but valid only for training images (i.e., need to know $\boldsymbol{x}$)

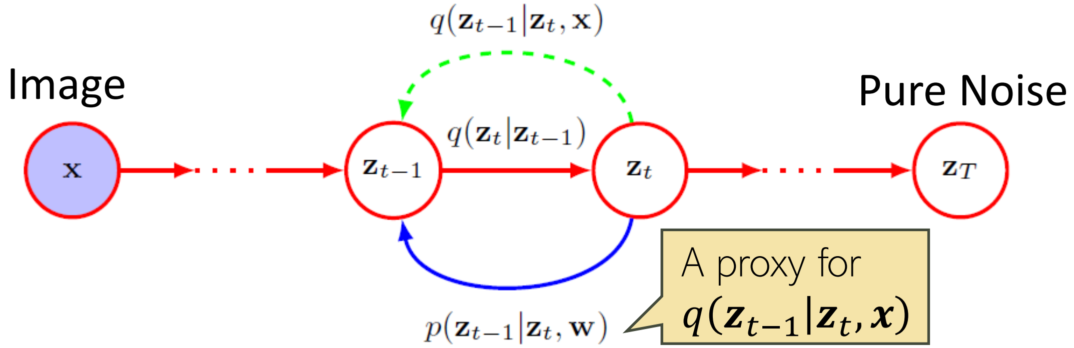
$$q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) = \frac{q(\boldsymbol{z}_{t-1}|\boldsymbol{x})\, q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, \boldsymbol{x})}{q(\boldsymbol{z}_t|\boldsymbol{x})} = \frac{q(\boldsymbol{z}_{t-1}|\boldsymbol{x})\, q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}{q(\boldsymbol{z}_t|\boldsymbol{x})}$$

Generating Data by Reversing Diffusion

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

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

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



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

A parametric model to reverse the diffusion: get $\boldsymbol{z}_{t-1}$ given $\boldsymbol{z}_t$

Just need to estimate the parameters $\boldsymbol{w}$

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

Assume a form that mimics the form of $m(\boldsymbol{x},\boldsymbol{z}_t)$

- ELBO maximization reduces to minimization of $||m(\boldsymbol{x},\boldsymbol{z}_t) - \mu(\boldsymbol{z}_t,\boldsymbol{w},t)||^2$
- Equivalent to minimization of $||g(\boldsymbol{z}_t,\boldsymbol{w},t) - \boldsymbol{\epsilon}||^2$ which we can do using SGD

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 $\mathbf{w}$

Can think of $g(\boldsymbol{z}_t, \boldsymbol{w}, t)$ as a noise predictor network

Pseudo-code from Bishop & Bishop (2023)

Denoising Diffusion Model: Generation

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

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

Input: Trained denoising network $\mathbf{g}(\mathbf{z}, \mathbf{w}, t)$

Noise schedule $\{\beta_1, \dots, \beta_T\}$

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 $t \in T, \dots, 2$ **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

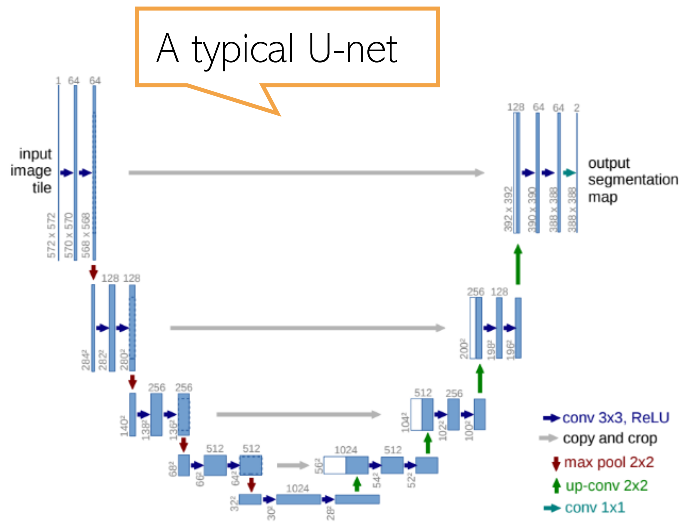
$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\left\{\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}}\mathbf{g}(\mathbf{z}_1, \mathbf{w}, t)\right\}$ `// Final denoising step`

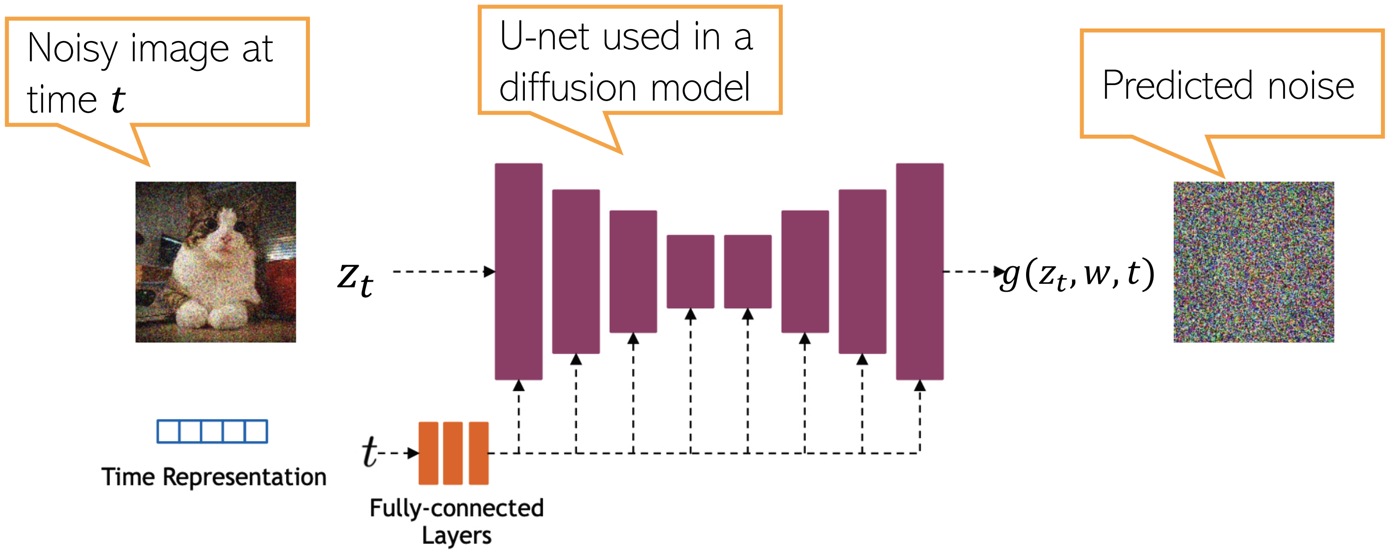
return $\mathbf{x}$

Pseudo-code from Bishop & Bishop (2023)

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

Score based deep generative models

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

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

- But doing so requires the score function $s(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$
- Since $\boldsymbol{p}(\boldsymbol{x})$ 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)

Diffusion Models via Score Matching

- For a probability distribution $\boldsymbol{p(x)}$ 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

- Learning this score function is equivalent to learning the distribution $\boldsymbol{p(x)}$
- We can parameterize the score function as $\boldsymbol{s(x) = s(x,w)}$ and define a loss function

Can define it as a neural network

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

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

A discrete distribution represented by the N samples from the dataset

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

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

Score Matching

- Instead of $p_{\mathcal{D}}(x)$, we define a smooth distribution

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

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

- Using this $q_\sigma(z)$ instead of $p_{\mathcal{D}}(x)$, 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

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

Score Matching

- Recall that the score loss function is

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

- Choosing $q(z|x, \sigma) = \mathcal{N}(z|x, \sigma^2 I)$, we get

$$\nabla_{\mathbf{z}} \ln q(\mathbf{z}|\mathbf{x}, \sigma) = -\frac{1}{\sigma}\boldsymbol{\epsilon} \qquad \text{where} \qquad \epsilon = x - z$$

- Note the similarity with denoising diffusion model where

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

- The score loss function measures the difference b/w predicted score $s(z, w)$ and noise
- Note that the score function $s(z, w)$ plays a similar role as noise predictor $g(z, w, t)$ in denoising diffusion model we saw earlier
- Careful selection of the noise variance σ^2 is important in this approach

Diffusion Models and SDE

- Stochastic Differential Equations (SDE) define a continuous-time process
- Denoising diffusion model and score matching models are like discretization of the continuous-time SDE
- The forward SDE is written as

$$\mathrm{d}\mathbf{z} = \underbrace{\mathbf{f}(\mathbf{z}, t)\,\mathrm{d}t}_{\text{drift}} + \underbrace{g(t)\,\mathrm{d}\mathbf{v}}_{\text{diffusion}}$$

- The corresponding reverse SDE can be written as

This is like the score function

$$\mathrm{d}\mathbf{z} = \left\{\mathbf{f}(\mathbf{z}, t) - g^2(t)\nabla_{\mathbf{z}} \ln p(\mathbf{z})\right\}\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{v}$$

The corresponding ODE for the SDE reverse process

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{f}(\mathbf{z}, t) - \frac{1}{2}g^2(t)\nabla_{\mathbf{z}} \ln p(\mathbf{z})$$

- We can solve SDE by discretizing time
  - For equal-size time steps, we get the Langevin dynamics based equations for the updates
- SDE connection is helpful in designing fast reverse process for diffusion models
  - For example, we can leverage the ODE corresponding to the SDE for faster sampling

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 $p(\boldsymbol{x}|\boldsymbol{c})$

Figure source: DLFC (Bishop and Bishop, 2023)

Classifier Guidance

- Assume we have an already training 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

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

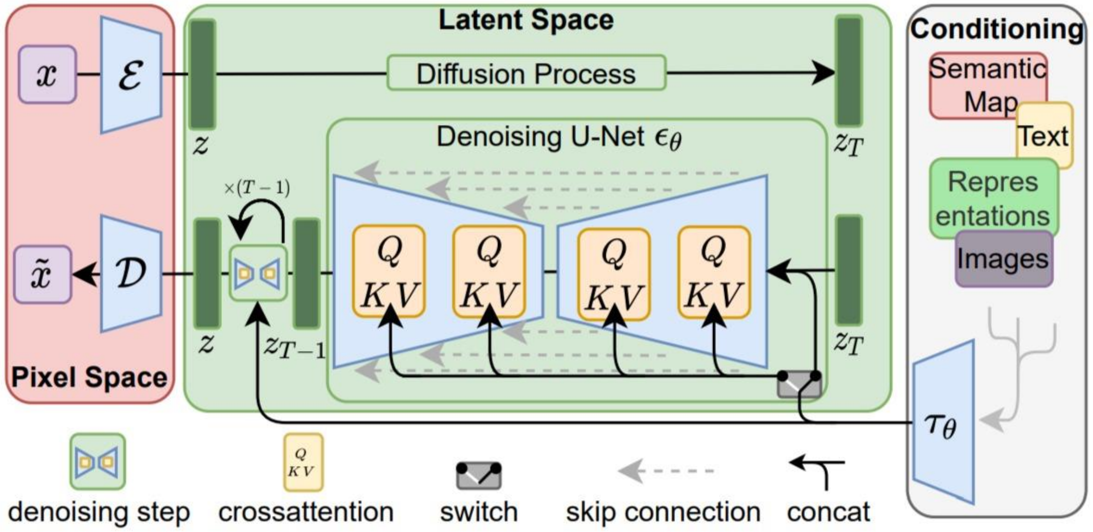
- Thus we can rewrite the score function as follows

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

- 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

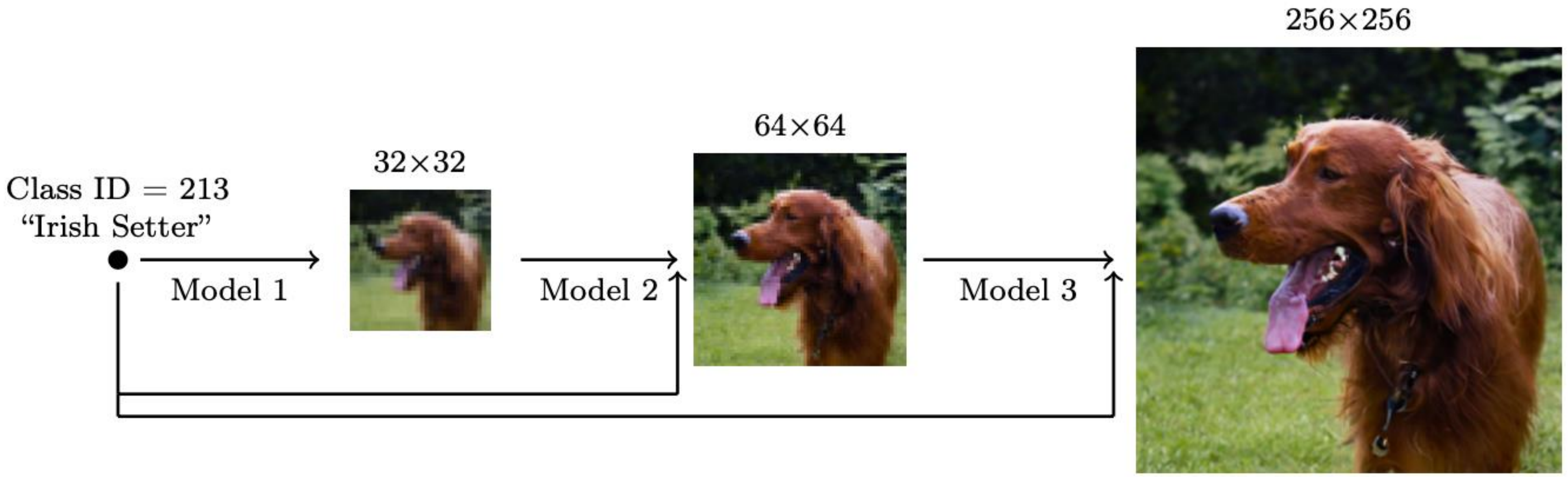


- Can also condition the generation on other modalities such as text

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

Cascaded Diffusion Models

- Useful for generating high-resolution images using conditioning



- Cascaded approach is usually better than a direct generation of high-resolution image
  - Smaller model size
  - Learning gradual transformations is easier than a direct transformation

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)