Denoising Diffusion Models (contd)

CS772A: Probabilistic Machine Learning

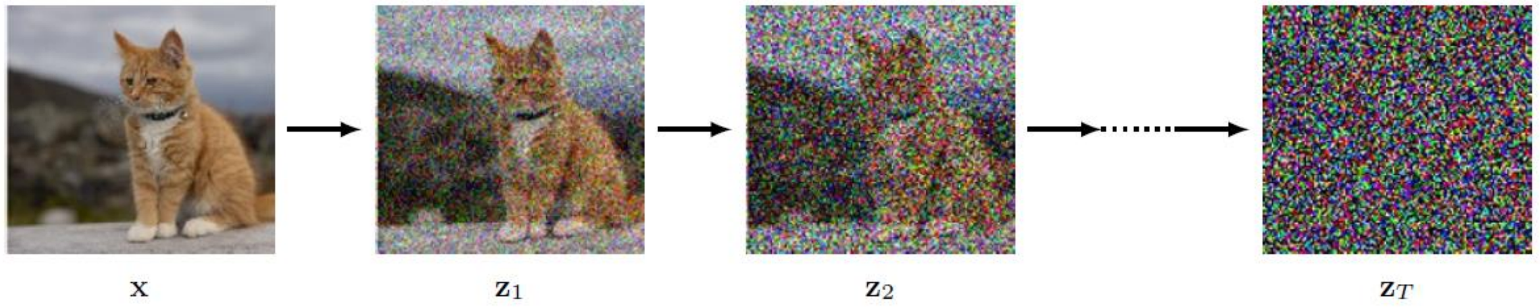
Piyush Rai

Plan today

- Recap of denoising diffusion models
- Another perspective of diffusion using score based generative models
- Guided/conditional generation with diffusion models
  - Classifier guidance
  - Classifier-free guidance
- Latent diffusion models

Denoising Diffusion Models

- Based on a forward process (adding noise) and a reverse process (denoising)
- The forward process as is follows $q(\mathbf{z}_1, \ldots, \mathbf{z}_t|\mathbf{x}) = q(\mathbf{z}_1|\mathbf{x}) \prod_{\tau=2}^{t} q(\mathbf{z}_\tau|\mathbf{z}_{\tau-1})$



$$\mathbf{z}_1 = \sqrt{1-\beta_1}\mathbf{x} + \sqrt{\beta_1}\boldsymbol{\epsilon}_1$$

$$q(\mathbf{z}_1|\mathbf{x}) = \mathcal{N}(\mathbf{z}_1|\sqrt{1-\beta_1}\mathbf{x}, \beta_1\mathbf{I})$$

$$\mathbf{z}_t = \sqrt{1-\beta_t}\mathbf{z}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}_t$$

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

$$\beta_t \in (0,1)$$

Typically set by hand

$$\beta_1 < \beta_2 < \ldots < \beta_T.$$

Called the "diffusion kernel"

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

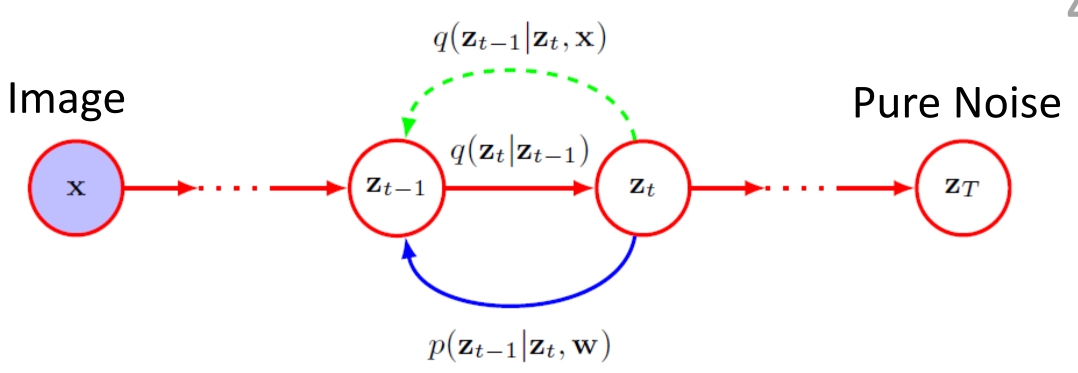
$$\alpha_t = \prod_{\tau=1}^{t}(1-\beta_\tau)$$

As $T \to \infty$

$$q(\mathbf{z}_T|\mathbf{x}) = \mathcal{N}(\mathbf{z}_T|\mathbf{0}, \mathbf{I})$$

Figure source: Bishop & Bishop (2023)

Reversing the Diffusion



- Reversing the diffusion is like denoising
- Can use it to generate data from pure noise
- Reverse process will need $q(z_{t-1}|z_t)$ but computing it is hard in general because

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t) = \frac{q(\mathbf{z}_t|\mathbf{z}_{t-1})q(\mathbf{z}_{t-1})}{q(\mathbf{z}_t)} \quad \text{where} \quad q(\mathbf{z}_{t-1}) = \int q(\mathbf{z}_{t-1}|\mathbf{x})p(\mathbf{x})\,d\mathbf{x}$$

Requires integrating over the data distribution $p(x)$ which is not easy

- Conditioning <u>also</u> on $\boldsymbol{x}$ makes computing the reverse process distribution tractable

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

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

But this denoising only holds for training images (because it is conditioned on $\boldsymbol{x}$)

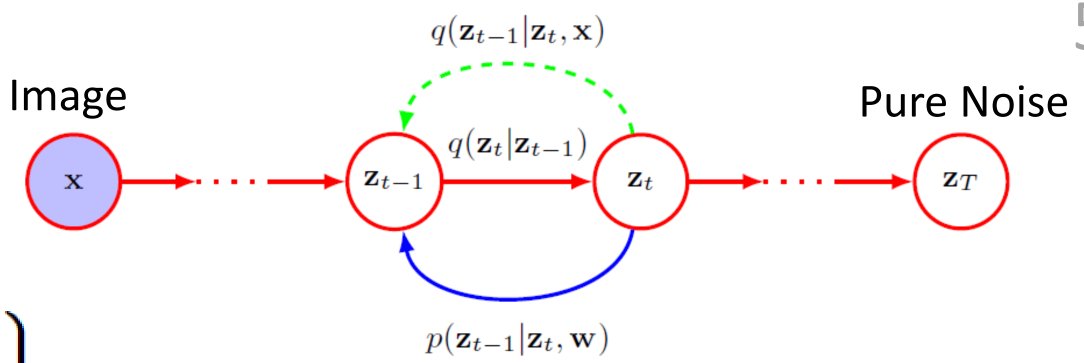
Thus we will also learn "parallel" distributions of the form $p(z_{t-1}|z_t, \boldsymbol{w})$ which approximate $q(z_{t-1}|z_t, \boldsymbol{x})$

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

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

If we can now estimate $\boldsymbol{w}$ from training data, we can use $p(z_{t-1}|z_t, \boldsymbol{w})$ to generate new synthetic data starting with pure noise

Reversing the Diffusion



- The joint distribution of data and latents

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

- Let's assume $p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w}) = \mathcal{N}(\mathbf{z}_{t-1}|\boldsymbol{\mu}(\mathbf{z}_t,\mathbf{w},t),\beta_t\mathbf{I})$

Note that μ represents the denoising model (e.g., a neural net) which denoises z_t to produce z_{t-1}

- The true joint distribution of the latents given $\boldsymbol{x}$

$$q(z_1,z_2,\ldots,z_T|x) = q(z_1|x)\prod_{t=2}^{T} q(z_t|z_{t-1},x)$$

- To estimate $\boldsymbol{w}$, we can maximize the ELBO defined as

This term is just like the VAE reconstruction error term (can approximate it using samples of z_1 from $q(z_1|x)$

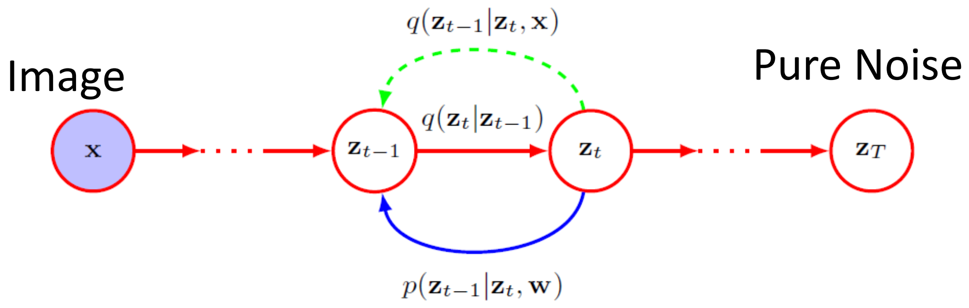
$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_q\left[\ln\frac{p(\mathbf{z}_T)\left\{\prod_{t=2}^{T} p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})\right\} p(\mathbf{x}|\mathbf{z}_1,\mathbf{w})}{q(\mathbf{z}_1|\mathbf{x})\prod_{t=2}^{T} q(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{x})}\right] = \mathbb{E}_q\left[\ln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

From ELBO definition $\mathbb{E}_q\left[\log\frac{p(X,Z)}{q(Z)}\right]$

Also note that unlike VI, here we aren't estimating the q distribution

First and third terms don't contain $\boldsymbol{w}$ so can be ignored when maximizing the ELBO

ELBO (contd)



- Recall the ELBO for the denoising diffusion model

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

- Ignoring terms that don't depend on $\boldsymbol{w}$ and using $q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}) = \dfrac{q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) q(\mathbf{z}_t|\mathbf{x})}{q(\mathbf{z}_{t-1}|\mathbf{x})}$

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

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

- Since both distributions in the KL divergence term are Gaussians, it becomes

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

Predicting the noise

- The KL terms in the ELBO are of the form

Network which gives the mean of the denoised z_{t-1}

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

- Note that

From the definition of $m_t(x, z_t)$

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

- Instead of learning $\mu(z_t, w, t)$, we will learn a noise predictor $g(z_t, w, t)$ s.t.

Using the same form as m_t with $g(z_t, w, t)$ trying to predict ϵ_t

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

- Therefore

$$\begin{aligned}&\mathrm{KL}(q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})||p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w}))\\ &= \frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}\left\|\mathbf{g}(\mathbf{z}_t,\mathbf{w},t) - \boldsymbol{\epsilon}_t\right\|^2 + \mathrm{const}\\ &= \frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}\left\|\mathbf{g}(\sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t,\mathbf{w},t) - \boldsymbol{\epsilon}_t\right\|^2 + \mathrm{const}\end{aligned}$$

Basically, we are now just predicting the noise ϵ_t using the neural network $g(z_t, w, t)$

Predicting the noise

- We basically had the following

$$\mathrm{KL}(q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})\|p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})) = \frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}\left\|\mathbf{g}(\sqrt{\alpha_t}\mathbf{x}+\sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t,\mathbf{w},t)-\boldsymbol{\epsilon}_t\right\|^2 + \text{const}$$

- The reconstruction error part in the ELBO can also be written as noise prediction

$$\ln p(\mathbf{x}|\mathbf{z}_1,\mathbf{w}) = -\frac{1}{2\beta_1}\|\mathbf{x}-\boldsymbol{\mu}(\mathbf{z}_1,\mathbf{w},1)\|^2 + \text{const.} = -\frac{1}{2(1-\beta_t)}\|\mathbf{g}(\mathbf{z}_1,\mathbf{w},1)-\boldsymbol{\epsilon}_1\|^2 + \text{const}$$

- Ignoring the constants in front of the squared error terms above, the ELBO becomes

Empirically found to give improved performance

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

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

Pick an example x randomly, generate a corruption z_t by sampling ϵ_t and make a gradient based update to w

Can optimize using stochastic optimization

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

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

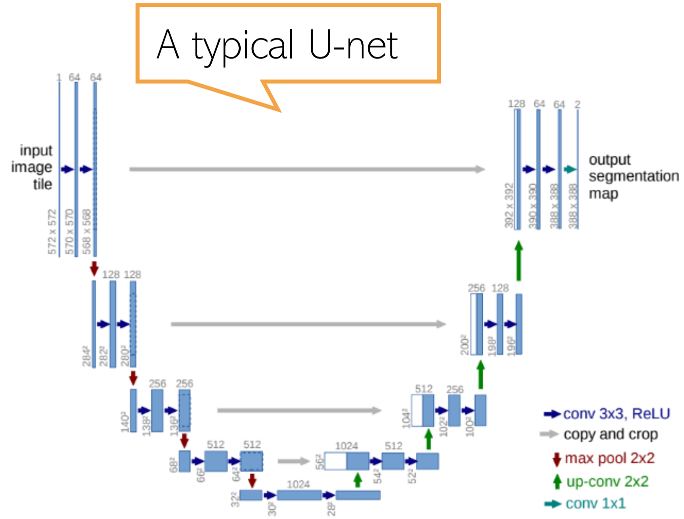
Input: Trained denoising network $\mathbf{g}(\mathbf{z}, \mathbf{w}, t)$
Noise schedule $\{\beta_1, \ldots, \beta_T\}$
Output: Sample vector $\mathbf{x}$ in data space

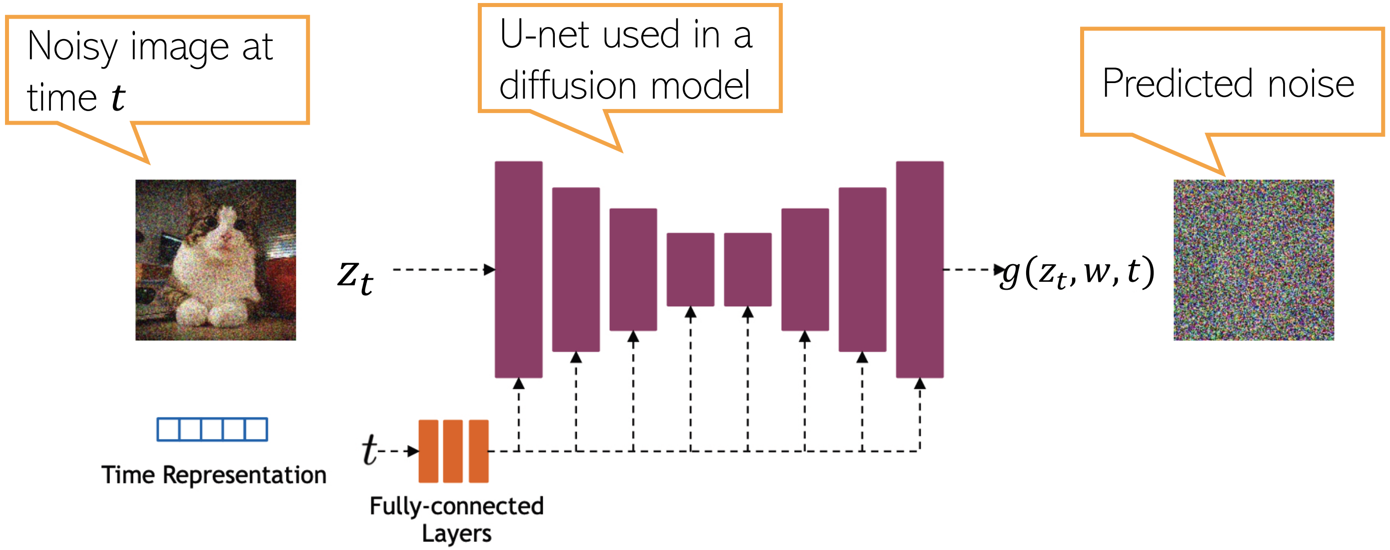
$\mathbf{z}_T \sim \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ `// Sample from final latent space`
for $t \in T, \ldots, 2$ **do**
  $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1 - \beta_\tau)$ `// Calculate alpha`
  `// Evaluate network output`
  $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) \leftarrow \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)\right\}$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ `// 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 $\mathbf{x}$

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

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

- But doing so requires the score function $\boldsymbol{s(x)} = \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$
- Since $\boldsymbol{p(x)}$ itself is not known, how do get the score function $\boldsymbol{s(x)}$?
- 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

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

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

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

Score Matching

- Recall that the score loss function is

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

- 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

Noise Variance in Score Matching

- Success of score matching depends on the estimate of score function $s(\boldsymbol{x}) = \nabla_x \log p(\boldsymbol{x})$

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

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

- In regions where $p(x)$ is small/zero, the estimate $s(z,w)$ may not be reliable
- Recall that, in score matching, we typically use $q(z|x,\sigma) = \mathcal{N}(z|x,\sigma^2 I)$
- Using the appropriate σ^2 is critical
  - Using large σ^2 means we won't have small/zero values for $q(z|x)$ but also high distortion
  - Very small σ^2 means $q(z|x)$ is close to $p(x)$
  - We can choose a series of variances $\sigma_1^2 < \sigma_2^2 < \cdots < \sigma_L^2$ and use the following loss function

This gives us L score matching based diffusion models with different variances

$\lambda(i)$ is the weighting coefficient for model i

$$\frac{1}{2}\sum_{i=1}^{L} \lambda(i) \int \|\mathbf{s}(\mathbf{z},\mathbf{w},\sigma_i^2) - \nabla_{\mathbf{z}} \ln q(\mathbf{z}|\mathbf{x}_n,\sigma_i)\|^2 q(\mathbf{z}|\mathbf{x}_n,\sigma_i)\,\mathrm{d}\mathbf{z}$$

We can run SGLD where we use a few steps of each in a sequences $L, L-1, L-2, \ldots, 2,1$

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

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

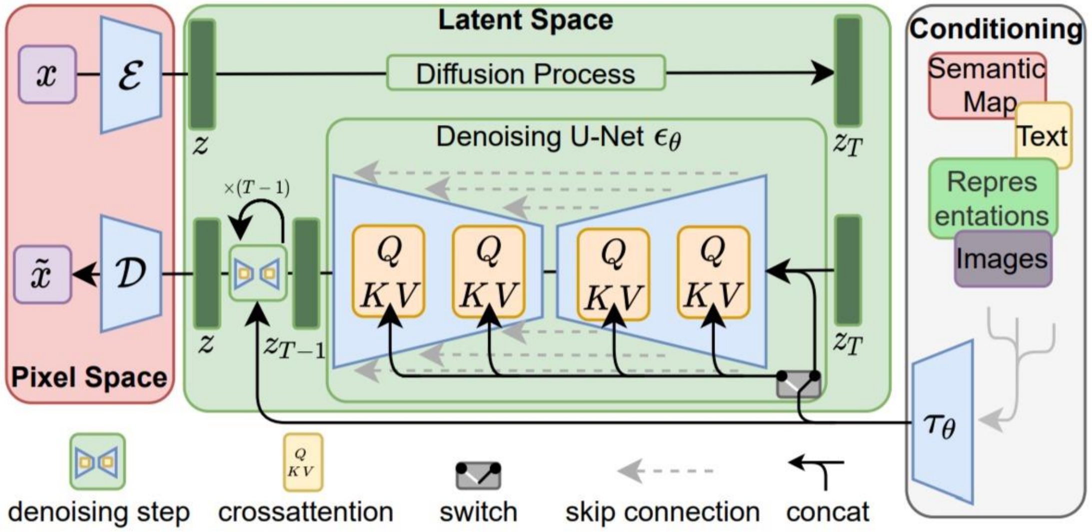
- Thus we can rewrite the score function as follows

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

- 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

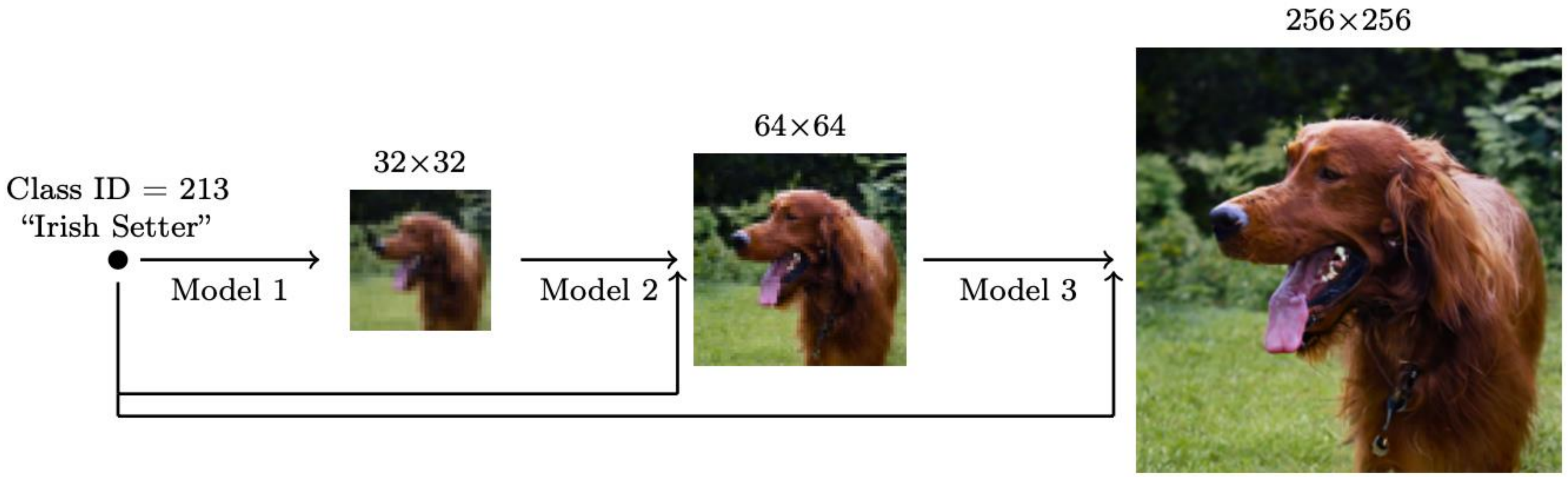


- Can also condition the generation of other modalities such as text

*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)