Generative Adversarial Networks, Diffusion Models

Milan Straka

May 9, 2023
Generative Models

There are several approaches how to represent a probability distribution $P(x)$. **Likelihood-based models** represent the probability density function directly, often using an unnormalized probabilistic model (also called energy-based model; i.e., specifying a non-zero score or density or logits):

$$P_\theta(x) = \frac{e^{f_\theta(x)}}{Z_\theta}.$$  

However, estimating the normalization constant $Z_\theta = \int e^{f_\theta(x)} \, dx$ is often intractable.

- We can compute $Z_\theta$ by restricting the model architecture (sequence modeling, invertible networks in normalizing flows);
- we can only approximate it (using for example variational inference as in VAE);
- we can use **implicit generative models**, which avoid representing likelihood (like GANs).
Generative Adversarial Networks

We have a **generator** $G(z; \theta_g)$, which given $z \sim P(z)$ generates data $x$.

Then we have a **discriminator** $D(x; \theta_d)$, which given data $x$ generates a probability whether $x$ comes from real data or is generated by a generator.

The discriminator and generator play the following game:

\[
\min_G \max_D \mathbb{E}_{x \sim P_{\text{data}}} \left[ \log D(x) \right] + \mathbb{E}_{z \sim P(z)} \left[ \log(1 - D(G(z))) \right].
\]

https://miro.medium.com/v2/1*-ucVYsbDnwa2NM-f5qm_Yg.png
The generator and discriminator are alternately trained, the discriminator by

$$\arg \max_{\theta_d} \mathbb{E}_{x \sim P_{\text{data}}} \left[ \log D(x) \right] + \mathbb{E}_{z \sim P(z)} \left[ \log (1 - D(G(z))) \right]$$

and the generator by

$$\arg \min_{\theta_g} \mathbb{E}_{z \sim P(z)} \left[ \log (1 - D(G(z))) \right].$$

Basically, the discriminator acts as a trainable loss for the generator.
Generative Adversarial Networks

Because \(\log(1 - D(G(z)))\) can saturate in the beginning of the training, where the discriminator can easily distinguish real and generated samples, the generator can be trained by 

\[
\arg \min_{\theta_g} \mathbb{E}_{z \sim P(z)} [-\log D(G(z))]
\]

instead, which results in the same fixed-point dynamics, but much stronger gradients early in learning.

On top of that, if you train the generator by using “real” as the gold label of the discriminator, you naturally get the above loss (which is the negative log likelihood, contrary to the original formulation).
Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, \( k \), is a hyperparameter. We used \( k = 1 \), the least expensive option, in our experiments.

for number of training iterations do
  for \( k \) steps do
    • Sample minibatch of \( m \) noise samples \( \{z^{(1)}, \ldots, z^{(m)}\} \) from noise prior \( p_g(z) \).
    • Sample minibatch of \( m \) examples \( \{x^{(1)}, \ldots, x^{(m)}\} \) from data generating distribution \( p_{\text{data}}(x) \).
    • Update the discriminator by ascending its stochastic gradient:
      \[
      \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D \left( x^{(i)} \right) + \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right) \right].
      \]
  end for
  • Sample minibatch of \( m \) noise samples \( \{z^{(1)}, \ldots, z^{(m)}\} \) from noise prior \( p_g(z) \).
  • Update the generator by descending its stochastic gradient:
    \[
    \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right).
    \]
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
Generative Adversarial Networks

Figure 2 of "Generative Adversarial Nets", https://arxiv.org/abs/1406.2661
Conditional GAN

Figure 1 of "Conditional Generative Adversarial Nets", https://arxiv.org/abs/1411.1784
In Deep Convolutional GAN, the discriminator is a convolutional network (with batch normalization) and the generator is also a convolutional network, utilizing transposed convolutions.

Figure 1 of "An Online Learning Approach to Generative Adversarial Networks", https://arxiv.org/abs/1706.03269
Deep Convolutional GAN

Figure 1 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434
Deep Convolutional GAN

Figure 3 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434
Deep Convolutional GAN

Figure 4 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434
Deep Convolutional GAN

Figure 7 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434

Results of doing the same arithmetic in pixel space
Deep Convolutional GAN

Figure 7 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434

Results of doing the same arithmetic in pixel space
Deep Convolutional GAN

Figure 8 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434
GANs Training — Training Experience

GAN output in paper

Your GAN output

https://miro.medium.com/max/1400/1*r8cuSIAH5oHUERP01TCTxg.jpeg
GANs Training – Results of In-House BigGAN Training
GANs are Problematic to Train

Unfortunately, alternating SGD steps are not guaranteed to reach even a local optimum of a minimax problem – consider the following one:

$$\min_x \max_y x \cdot y.$$ 

The update rules of $x$ and $y$ for learning rate $\alpha$ are

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & -\alpha \\ \alpha & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix}.$$ 

The update matrix is a rotation matrix multiplied by a constant $\sqrt{1 + \alpha^2} > 1$

$$\begin{bmatrix} 1 & -\alpha \\ \alpha & 1 \end{bmatrix} = \sqrt{1 + \alpha^2} \cdot \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{bmatrix},$$

so the SGD will not converge with arbitrarily small step size.
GANs are Problematic to Train

Fig. 1: Performance of gradient method with fixed step size for Example 2. (a) illustrates the choices of x and y as iteration processes, the red point (0.1, 0.1) is the initial value. (b) illustrates the value of $xy$ as a function of iteration numbers.

*Figure 1 of "Fictitious GAN: Training GANs with Historical Models", https://arxiv.org/abs/1803.08647*
GANs are Problematic to Train

- Mode collapse

  - If the discriminator could see the whole batch, similar samples in it would be candidates for fake images.
    - Batch normalization helps a lot with this.
  - Historical averaging

- Label smoothing of only positive samples helps with the gradient flow.
Comparison of VAEs and GANs

The Variational Autoencoders:

- are theoretically-pleasing;
- also provide an encoder, so apart from generation, they can be used as unsupervised feature extraction (the VAE encoder is used in various modeling architectures);
- the generated samples tend to be blurry, especially with $L^1$ or $L^2$ loss (because of the sampling used in the reconstruction; patch-based discriminator with perceptual loss helps).

The Generative Adversarial Networks:

- offer high sample quality;
- are difficult to train and suffer from mode collapse.

In past few years, GANs saw a big development, improving the sample quality substantially. However, since 2019/2020, VAEs have shown remarkable progress (alleviating the blurriness issue by using perceptual loss and a 2D grid of latent variables), and are being used for generation too. Furthermore, additional approaches (normalizing flows, diffusion models) were also being explored, with diffusion models becoming the most promising approach since Q2 of 2021, surpassing both VAEs and GANs.
Currently (as of May 2023), the best architecture for generating images seems to be the diffusion models.

The diffusion models are deeply connected to **score-based generative models**, which were developed independently. These two approaches are in fact just different perspectives of the same model family, and many recent papers utilize both sides of these models.
Given a data point $x_0$ from a real data distribution $q(x)$, we define a $T$-step diffusion process (or the forward process) which gradually adds Gaussian noise to the input image:

$$q(x_{1:T} | x_0) = \prod_{t=1}^{T} q(x_t | x_{t-1}).$$

Our goal is to reverse the forward process $q(x_t | x_{t-1})$, and generate an image by starting with $x_T \sim \mathcal{N}(0, I)$, and then performing the forward process in reverse. We therefore learn a model $p_{\theta}(x_{t-1} | x_t)$ to approximate the reverse of $q(x_t | x_{t-1})$, and obtain a reverse process:

$$p_{\theta}(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_{\theta}(x_{t-1} | x_t).$$
Diffusion Models – The Diffusion Process

The forward trajectory
\[ q(x_{0:T}) \]

The reverse trajectory
\[ p_\theta(x_{0:T}) \]

The drifting term
\[ \mu_\theta(x_t, t) - x_t \]
Diffusion Models – The Diffusion Process
The $p_\theta(x_{t-1} | x_t)$ is commonly modelled using a UNet architecture with skip connections.

**Training**
During training, we randomly sample a time step $t$, and perform an update of the parameters $\theta$ in order for $p_\theta(x_{t-1} | x_t)$ to better approximate the reverse of $q(x_t | x_{t-1})$.

**Sampling**
In order to sample an image, we start by sampling $x_T \sim \mathcal{N}(0, I)$, and then perform $T$ steps of the reverse process by sampling $x_{t-1} \sim p_\theta(x_{t-1} | x_t)$ for $t$ from $T$ down to 1.
Normal Distribution Reminder

Normal (or Gaussian) distribution is a continuous distribution parametrized by a mean $\mu$ and variance $\sigma^2$:

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp \left( -\frac{(x - \mu)^2}{2\sigma^2} \right)$$

For a $D$-dimensional vector $\mathbf{x}$, the multivariate Gaussian distribution takes the form

$$\mathcal{N}(\mathbf{x}; \mu, \Sigma) \overset{\text{def}}{=} \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp \left( -\frac{1}{2} (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu) \right).$$

The biggest difference compared to the single-dimensional case is the covariance matrix $\Sigma$, which is (in the non-degenerate case, which is the only one considered here) a symmetric positive-definite matrix of size $D \times D$.

However, in this lecture we will only consider isotropic distribution, where $\Sigma = \sigma^2 I$:

$$\mathcal{N}(\mathbf{x}; \mu, \sigma^2 I) = \prod_i \mathcal{N}(x_i; \mu_i, \sigma^2).$$
Normal Distribution Reminder

• A normally-distributed random variable \( x \sim \mathcal{N}(\mu, \sigma^2 I) \) can be written using the reparametrization trick also as

\[
x = \mu + \sigma e, \quad \text{where} \quad e \sim \mathcal{N}(0, I).
\]

• The sum of two independent normally-distributed random variables \( x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2 I) \) and \( x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2 I) \) has normal distribution \( \mathcal{N}(\mu_1 + \mu_2, (\sigma_1^2 + \sigma_2^2)I) \).

Therefore, if we have two standard normal random variables \( e_1, e_2 \sim \mathcal{N}(0, I) \), then

\[
\sigma_1 e_1 + \sigma_2 e_2 = \sqrt{\sigma_1^2 + \sigma_2^2} e
\]

for a standard normal random variable \( e \sim \mathcal{N}(0, I) \).
We now describe Denoising Diffusion Probabilistic Models (DDPM).

![Diagram of the forward process](image-url)

Given a data point $x_0$ from a real data distribution $q(x)$, we define a $T$-step diffusion process (or the forward process) which gradually adds Gaussian noise according to some variance schedule $\beta_1, \ldots, \beta_T$:

$$q(x_{1:T} | x_0) = \prod_{t=1}^{T} q(x_t | x_{t-1}),$$

$$q(x_t | x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t} x_{t-1}, \beta_t I),$$

$$= \sqrt{1 - \beta_t} x_{t-1} + \sqrt{\beta_t} e \text{ for } e \sim \mathcal{N}(0, I).$$

More noise gets gradually added to the original image $x_0$, converging to pure Gaussian noise.
Let \( \alpha_t = 1 - \beta_t \) and \( \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i \). Then we have

\[
x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1 - \alpha_t} e_t
\]

\[
= \sqrt{\alpha_t} \left( \sqrt{\alpha_{t-1}} x_{t-2} + \sqrt{1 - \alpha_{t-1}} e_{t-1} \right) + \sqrt{1 - \alpha_t} e_t
\]

\[
= \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{\alpha_t (1 - \alpha_{t-1}) + (1 - \alpha_t) \bar{e}_{t-1}}
\]

\[
= \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{1 - \alpha_t \alpha_{t-1}} \bar{e}_{t-1}
\]

\[
= \sqrt{\alpha_t \alpha_{t-1}} \alpha_{t-2} x_{t-3} + \sqrt{1 - \alpha_t \alpha_{t-1} \alpha_{t-2}} \bar{e}_{t-2}
\]

\[
= \ldots
\]

\[
= \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \bar{e}_0
\]

for standard normal random variables \( e_i \) and \( \bar{e}_i \).

In other words, we have shown that

\[
q(x_t | x_0) = \mathcal{N} \left( \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t) I \right)
\]

Therefore, if \( \bar{\alpha}_t \rightarrow 0 \) as \( t \rightarrow \infty \), the \( x_t \) converges to \( \mathcal{N}(0, I) \) as \( t \rightarrow \infty \).
DDPM – Noise Schedule

Originally, linearly increasing sequence of noise variations $\beta_1 = 0.0001, \ldots, \beta_T = 0.04$ was used.

However, the resulting sequence $\bar{\alpha}_t$ was not ideal (nearly the whole second half of the diffusion process was mostly just random noise), so later a cosine schedule was proposed:

$$\bar{\alpha}_t = \frac{1}{2} \left( \cos \left( \frac{t}{T} \cdot \pi \right) + 1 \right),$$

and now it is dominantly used.

In practice, we want to avoid both the values of 0 and 1, and keep $\alpha_t$ in $[\varepsilon, 1 - \varepsilon]$ range.

Figure 5. $\bar{\alpha}_t$ throughout diffusion in the linear schedule and our proposed cosine schedule.

Figure 5 of "Improved Denoising Diffusion Probabilistic Models", https://arxiv.org/abs/2102.09672
We assume the images $x_0$ have zero mean and unit variance (we normalize them to achieve that). Then every

$$q(x_t|x_0) = \sqrt{\alpha_t}x_0 + \sqrt{1 - \alpha_t}e$$

has also zero mean and unit variance.

The $\sqrt{\alpha_t}$ and $\sqrt{1 - \alpha_t}$ can be considered as the signal rate and the noise rate.

Because $\sqrt{\alpha_t}^2 + \sqrt{1 - \alpha_t}^2 = 1$, the signal rate and the noise rate form a circular arc. The proposed cosine schedule

$$\sqrt{\alpha_t} = \cos(t/T \cdot \pi/2),$$

$$\sqrt{1 - \alpha_t} = \sin(t/T \cdot \pi/2),$$

corresponds to an uniform movement on this arc.
In order to be able to generate images, we therefore learn a model \( p_{\theta}(x_{t-1} | x_t) \) to approximate the reverse of \( q(x_t | x_{t-1}) \).

When \( \beta_t \) is small, this reverse is nearly Gaussian, so we represent \( p_{\theta} \) as

\[
p_{\theta}(x_{t-1} | x_t) = \mathcal{N}(x_{t-1}; \mu_{\theta}(x_t, t), \sigma_t^2 I)
\]

for some fixed sequence of \( \sigma_1, \ldots, \sigma_T \).

The whole reverse process is then

\[
p_{\theta}(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_{\theta}(x_{t-1} | x_t).
\]
We now want to derive the loss. First note that the reverse of $q(x_t|\tilde{x}_{t-1})$ is actually tractable when conditioning on $x_0$:

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I),$$

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\alpha_{t-1}} \beta_t}{1 - \bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_t)}{1 - \bar{\alpha}_t} x_t,$$

$$\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t.$$ 

We present the proof on the next slide for completeness.
Starting with the Bayes' rule, we get

\[ q(x_{t-1}|x_t, x_0) = q(x_t|x_{t-1}, x_0) \frac{q(x_{t-1}|x_0)}{q(x_t|x_0)} \]

\[ \propto \exp\left(-\frac{1}{2}\left(\frac{\left(x_t - \sqrt{\alpha_t} x_{t-1}\right)^2}{\beta_t} + \frac{\left(x_{t-1} - \sqrt{\alpha_{t-1}} x_0\right)^2}{1 - \bar{\alpha}_{t-1}} - \frac{\left(x_t - \sqrt{\bar{\alpha}_t} x_0\right)^2}{1 - \bar{\alpha}_t}\right)\right) \]

\[ = \exp\left(-\frac{1}{2}\left(\frac{x_t^2 - 2\sqrt{\alpha_t} x_t x_{t-1} + \alpha_t x_{t-1}^2}{\beta_t} + \frac{x_{t-1}^2 - 2\sqrt{\alpha_{t-1}} x_t x_{t-1} x_0 + \alpha_{t-1} x_0^2}{1 - \bar{\alpha}_{t-1}} + \ldots\right)\right) \]

From this formulation, we can derive that \( q(x_{t-1}|x_t, x_0) = \mathcal{N}\left(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I\right) \) for

\[ \tilde{\beta}_t = 1/\left(\frac{\alpha_t}{\beta_t} + \frac{1}{1 - \alpha_{t-1}}\right) = 1/\left(\frac{\alpha_t(1 - \bar{\alpha}_{t-1}) + \beta_t}{\beta_t(1 - \alpha_{t-1})}\right) = 1/\left(\frac{\alpha_t + \beta_t - \bar{\alpha}_t}{\beta_t(1 - \alpha_{t-1})}\right) = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t, \]

\[ \tilde{\mu}_t(x_t, x_0) = \left(\frac{\sqrt{\alpha_t}}{\beta_t} x_t + \frac{\sqrt{\alpha_{t-1}}}{1 - \alpha_{t-1}} x_0\right) \frac{1 - \alpha_{t-1}}{1 - \alpha_t} \beta_t = \frac{\sqrt{\alpha_{t-1}}}{1 - \bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}}{1 - \bar{\alpha}_t} x_t. \]
DDPM – Deriving Loss using Jensen's Inequality

\[-\mathbb{E}_{q(x_0)} \left[ \log p_\theta(x_0) \right] = -\mathbb{E}_{q(x_0)} \left[ \log \mathbb{E}_{p_\theta(x_{1:T})} \left[ p_\theta(x_0) \right] \right] \]

\[= -\mathbb{E}_{q(x_0)} \left[ \log \mathbb{E}_{q(x_{1:T} \mid x_0)} \left[ \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)} \right] \right] \]

\[\leq -\mathbb{E}_{q(x_{0:T})} \left[ \log \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)} \right] = \mathbb{E}_{q(x_{0:T})} \left[ \log \frac{q(x_{1:T} \mid x_0)}{p_\theta(x_{0:T})} \right] \]

\[= \mathbb{E}_{q(x_{0:T})} \left[ -\log p_\theta(x_T) + \sum_{t=2}^{T} \log \frac{q(x_t \mid x_{t-1})}{p_\theta(x_{t-1} \mid x_t)} + \log \frac{q(x_1 \mid x_0)}{p_\theta(x_0 \mid x_1)} \right] \]

\[= \mathbb{E}_{q(x_{0:T})} \left[ -\log p_\theta(x_T) + \sum_{t=2}^{T} \log \left( \frac{q(x_{t-1} \mid x_t, x_0)}{p_\theta(x_{t-1} \mid x_t)} \frac{q(x_t \mid x_0)}{q(x_{t-1} \mid x_0)} \right) + \log \frac{q(x_1 \mid x_0)}{p_\theta(x_0 \mid x_1)} \right] \]

\[= \mathbb{E}_{q(x_{0:T})} \left[ -\log p_\theta(x_T) + \sum_{t=2}^{T} \log \frac{q(x_{t-1} \mid x_t, x_0)}{p_\theta(x_{t-1} \mid x_t)} + \log \frac{q(x_T \mid x_0)}{q(x_1 \mid x_0)} + \log \frac{q(x_1 \mid x_0)}{p_\theta(x_0 \mid x_1)} \right] \]

\[= \mathbb{E}_{q(x_{0:T})} \left[ \log \frac{q(x_T \mid x_0)}{p_\theta(x_T)} + \sum_{t=2}^{T} \log \frac{q(x_{t-1} \mid x_t, x_0)}{p_\theta(x_{t-1} \mid x_t)} - \log p_\theta(x_0 \mid x_1) \right] \]

\[= \mathbb{E}_{q(x_{0:T})} \left[ \text{KL}(q(x_T \mid x_0) \mid \mid p_\theta(x_T)) + \sum_{t=2}^{T} \text{KL}(q(x_{t-1} \mid x_t, x_0) \mid \mid p_\theta(x_{t-1} \mid x_t)) - \log p_\theta(x_0 \mid x_1) \right] \]
DDPM – Deriving Loss using Jensen's Inequality

The whole loss is therefore composed of the following components:

- \( L_T = D_{KL} \left( q(x_T | x_0) || p_\theta(x_T) \right) \) is constant with respect to \( \theta \) and can be ignored,

- \( L_t = D_{KL} \left( q(x_{t-1} | x_t, x_0) || p_\theta(x_{t-1} | x_t) \right) \) is KL divergence between two Gaussians, so it can be computed explicitly as

\[
L_t = \mathbb{E} \left[ \frac{1}{2\|\sigma_t I\|^2} \left\| \tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t) \right\|^2 \right],
\]

- \( L_0 = -\log p_\theta(x_0 | x_1) \) can be used to generate discrete \( x_0 \) from the continuous \( x_1 \); we will ignore it in the slides for simplicity.
Recall that \( q(x_{t-1} | x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I) \) for

\[
\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\alpha_{t-1}} \beta_t}{1 - \alpha_t} x_0 + \frac{\sqrt{\alpha_t} (1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t,
\]

\[
\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t.
\]

Because \( x_t = \sqrt{\alpha_t} x_0 + \sqrt{1 - \bar{\alpha}_t} e_t \), we get \( x_0 = \frac{1}{\sqrt{\alpha_t}} (x_t - \sqrt{1 - \bar{\alpha}_t} e_t) \).

Substituting \( x_0 \) to \( \tilde{\mu}_t \), we get

\[
\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\alpha_{t-1}} \beta_t}{1 - \alpha_t} \frac{1}{\sqrt{\alpha_t}} (x_t - \sqrt{1 - \bar{\alpha}_t} e_t) + \frac{\sqrt{\alpha_t} (1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t
\]

\[
= \left( \frac{\sqrt{\alpha_{t-1}} \beta_t}{1 - \alpha_t} \frac{1}{\sqrt{\alpha_t}} + \frac{\sqrt{\alpha_t} (1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \right) x_t - \left( \frac{\sqrt{\alpha_{t-1}} \beta_t}{1 - \alpha_t} \frac{\sqrt{1 - \bar{\alpha}_t}}{\sqrt{\alpha_t}} \right) e_t
\]

\[
= \frac{\beta_t + \alpha_t (1 - \bar{\alpha}_{t-1})}{(1 - \bar{\alpha}_t) \sqrt{\alpha_t}} x_t - \left( \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t} \sqrt{\alpha_t}} \right) e_t = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} e_t \right).
\]
We change our model to predict $\mathbf{e}_\theta(x_t, t)$ instead of $\mu_\theta(x_t, t)$. The loss $L_t$ then becomes

$$L_t = \mathbb{E} \left[ \frac{1}{2\|\sigma_t I\|^2} \left\| \hat{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t) \right\|^2 \right]$$

$$= \mathbb{E} \left[ \frac{1}{2\|\sigma_t I\|^2} \left\| \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \mathbf{e}_t \right) - \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \mathbf{e}_\theta(x_t, t) \right) \right\|^2 \right]$$

$$= \mathbb{E} \left[ \frac{(1 - \alpha_t)^2}{2\alpha_t(1 - \bar{\alpha}_t)\|\sigma_t I\|^2} \left\| \mathbf{e}_t - \mathbf{e}_\theta(x_t, t) \right\|^2 \right]$$

$$= \mathbb{E} \left[ \frac{(1 - \alpha_t)^2}{2\alpha_t(1 - \bar{\alpha}_t)\|\sigma_t I\|^2} \left\| \mathbf{e}_t - \mathbf{e}_\theta(\sqrt{\alpha_t}x_0 + \sqrt{1 - \bar{\alpha}_t} \mathbf{e}_t, t) \right\|^2 \right].$$

The authors found that training without the weighting term performs better, so the final loss is

$$L_t^{\text{simple}} = \mathbb{E}_{t \in \{1..T\}, x_0, \mathbf{e}_t} \left[ \left\| \mathbf{e}_t - \mathbf{e}_\theta(\sqrt{\alpha_t}x_0 + \sqrt{1 - \bar{\alpha}_t} \mathbf{e}_t, t) \right\|^2 \right].$$

Note that both losses have the same optimum if we used independent $\mathbf{e}_{\theta_t}$ for every $t$. 
DDPM – Training and Sampling Algorithms

Algorithm 1 Training

1: repeat
2: \( x_0 \sim q(x_0) \)
3: \( t \sim \text{Uniform}\{1, \ldots, T\} \)
4: \( \epsilon \sim \mathcal{N}(0, I) \)
5: Take gradient descent step on
\[
\nabla_\theta \left\| \epsilon - \epsilon_\theta \left( \sqrt{\alpha_t} x_0 + \sqrt{1 - \alpha_t} \epsilon, t \right) \right\|^2
\]
6: until converged

Algorithm 2 Sampling

1: \( x_T \sim \mathcal{N}(0, I) \)
2: for \( t = T, \ldots, 1 \) do
3: \( z \sim \mathcal{N}(0, I) \) if \( t > 1 \), else \( z = 0 \)
4: \( x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \alpha_t}} \epsilon_\theta(x_t, t) \right) + \sigma_t z \)
5: end for
6: return \( x_0 \)

Sampling using the proposed algorithm is slow – it is common to use \( T = 1000 \) steps during sampling.

The value of \( \sigma_t^2 \) is chosen to be either \( \beta_t \) or \( \tilde{\beta}_t \), or any value in between (it can be proven that these values correspond to upper and lower bounds on the reverse process entropy).

Both of these issues will be alleviated later, when we present DDIM providing an updated sampling algorithm, which runs in several tens of steps and does not use \( \sigma_t^2 \).
The DDPM models the noise prediction $\varepsilon_{\theta}(x_t, t)$ using a UNet architecture with pre-activated ResNet blocks.

- The current time step is represented using the Transformer sinusoidal embeddings and added “in the middle” of every residual block (after the first convolution).
- Additionally, on several lower-resolution levels, a self-attention block (an adaptation of the Transformer self-attention, which considers the 2D grid of features as a sequence of feature vectors) is commonly used.

Figure 2 of "Self-Attention Generative Adversarial Networks", https://arxiv.org/abs/1805.08318

*NPFL114, Lecture 13*
Diffusion Models Architecture – ImaGen

Figure A.30 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", https://arxiv.org/abs/2205.11487

Figure A.27 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", https://arxiv.org/abs/2205.11487
There are just minor differences in the ImaGen architecture – for example the place where the time sinusoidal embeddings are added.
Conditional Models, Classifier-Free Guidance

In many cases we want the generative model to be conditional. We have already seen how to condition it on the current time step. Additionally, we might consider also conditioning on

- an image (e.g., for super-resolution): the image is then resized and concatenated with the input noised image (and optionally in other places, like after every resolution change);
- a text: the usual approach is to encode the text using some pre-trained encoder, and then to introduce an “image-text” attention layer (usually after the self-attention layers).

To make the effect of conditioning stronger during sampling, we might also employ classifier-free guidance:

- During training, we sometimes train $\varepsilon_\theta(x_t, t, y)$ with the conditioning $y$, and sometimes we train $\varepsilon_\theta(x_t, t, \emptyset)$ without the conditioning.
- During sampling, we pronounce the effect of the conditioning by taking the unconditioned noise and adding the difference between conditioned and unconditioned noise weighted by the weight $w$ (Stable Diffusion uses $w = 7.5$):

$$
\varepsilon_\theta(x_t, t, \emptyset) + w(\varepsilon_\theta(x_t, t, y) - \varepsilon_\theta(x_t, t, \emptyset)).
$$
We now describe *Denoising Diffusion Implicit Models (DDIM)*, which utilize a different forward process.

This forward process is designed to:

- allow faster sampling,
- have the same "marginals" $q(x_t|x_0) = \mathcal{N}(\sqrt{\alpha_t}x_0, (1 - \bar{\alpha}_t)I)$.

The second condition will allow us to use the same loss as in DDPM – therefore, the training algorithm is exactly identical to DDPM, only the sampling algorithm is different.

Note that in the slides, only a special case of DDIM is described; the original paper describes a more general forward process. However, the special case presented here is almost exclusively used.
The forward process of DDIM can be described using

\[ q_0(x_{1:T}|x_0) = q_0(x_T|x_0) \prod_{t=2}^{T} q_0(x_{t-1}|x_t, x_0), \]

where

- \( q_0(x_T|x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_T}x_0, (1 - \bar{\alpha}_T)I) \),
- \( q_0(x_{t-1}|x_t, x_0) = \mathcal{N}\left( \sqrt{\bar{\alpha}_{t-1}}x_0 + \sqrt{1 - \bar{\alpha}_{t-1}}\left( \frac{x_t - \sqrt{\bar{\alpha}_t}x_0}{\sqrt{1 - \bar{\alpha}_t}} \right), 0 \cdot I \right) \).

With these definitions, we can prove by induction that \( q_0(x_t|x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}x_0, (1 - \bar{\alpha}_t)I) \):

\[
\begin{align*}
x_{t-1} &= \sqrt{\bar{\alpha}_{t-1}}x_0 + \sqrt{1 - \bar{\alpha}_{t-1}}\left( \frac{x_t - \sqrt{\bar{\alpha}_t}x_0}{\sqrt{1 - \bar{\alpha}_t}} \right) \\
&= \sqrt{\bar{\alpha}_{t-1}}x_0 + \sqrt{1 - \bar{\alpha}_{t-1}}\left( \frac{\sqrt{\bar{\alpha}_t}x_0 \sqrt{1 - \bar{\alpha}_t} + \sqrt{1 - \bar{\alpha}_t}e_t - \sqrt{\bar{\alpha}_t}x_0}{\sqrt{1 - \bar{\alpha}_t}} \right) \\
&= \sqrt{\bar{\alpha}_{t-1}}x_0 + \sqrt{1 - \bar{\alpha}_{t-1}}e_t.
\end{align*}
\]

The real “forward” \( q_0(x_t|x_{t-1}, x_0) \) can be expressed using Bayes’ theorem using the above definition, but we do not actually need it.
The definition of $q_0(x_{t-1}|x_t, x_0)$ provides us also with a sampling algorithm – after sampling the initial noise $x_T \sim \mathcal{N}(0, I)$, we perform the following for $t$ from $T$ down to 1:

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}} x_0 + \sqrt{1 - \bar{\alpha}_{t-1}} \varepsilon_\theta(x_t, t)$$

$$= \sqrt{\bar{\alpha}_{t-1}} \left( \frac{x_t - \sqrt{1 - \bar{\alpha}_{t}} \varepsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}} \right) + \sqrt{1 - \bar{\alpha}_{t-1}} \varepsilon_\theta(x_t, t).$$

An important property of $q_0$ is that it can also model several steps at once:

$$q_0(x_{t'}|x_t, x_0) = \mathcal{N} \left( \sqrt{\bar{\alpha}_{t'}} x_0 + \sqrt{1 - \bar{\alpha}_{t'}} \left( \frac{x_t - \sqrt{1 - \bar{\alpha}_t} x_0}{\sqrt{1 - \bar{\alpha}_t}} \right), 0 \right).$$
We base our accelerated sampling algorithm on the “multistep” \( q_0(x_t' | x_t, x_0) \).

Let \( t_S = T, t_{S-1}, \ldots, t_1 \) be a subsequence of the process steps (usually, a uniform subsequence of \( T, \ldots, 1 \) is used). Starting from the initial noise \( x_T \sim \mathcal{N}(0, I) \), we perform \( S \) sampling steps for \( i \) from \( S \) down to 1:

\[
\begin{align*}
x_{t_{i-1}} &= \sqrt{\bar{\alpha}_{t_{i-1}}} \left( \frac{x_{t_i} - \sqrt{1 - \bar{\alpha}_{t_i}} \epsilon_\theta(x_{t_i}, t_i)}{\sqrt{\bar{\alpha}_{t_i}}} \right) + \sqrt{1 - \bar{\alpha}_{t_{i-1}}} \epsilon_\theta(x_{t_i}, t_i).
\end{align*}
\]

The sampling procedure can be described in words as follows:

- using the current time step \( t_i \), we compute the estimated noise \( \epsilon_\theta(x_{t_i}, t_i) \);
- by utilizing the current signal rate \( \sqrt{\alpha_{t_i}} \) and noise rate \( \sqrt{1 - \alpha_{t_i}} \), we estimate \( x_0 \);
- we obtain \( x_{t_{i-1}} \) by combining the estimated signal \( x_0 \) and noise \( \epsilon_\theta(x_{t_i}, t_i) \) using the signal and noise rates of the time step \( t_{i-1} \).
DDIM – Accelerated Sampling Examples

Figure 3 of “Denoising Diffusion Implicit Models”, https://arxiv.org/abs/2010.02502

Figure 5 of “Denoising Diffusion Implicit Models”, https://arxiv.org/abs/2010.02502
DDIM – Samples from Model Trained in Practicals
DDIM – Conditional Samples from Model Trained in Practicals
Stable Diffusion – Semantic and Perceptual Compression

Figure 2 of "High-Resolution Image Synthesis with Latent Diffusion Models", https://arxiv.org/abs/2112.10752

Semantic Compression

→ Generative Model: Latent Diffusion Model (LDM)

Perceptual Compression

→ Autoencoder+GAN
Figure 3 of "High-Resolution Image Synthesis with Latent Diffusion Models", https://arxiv.org/abs/2112.10752
Score Matching

Recall that loglikelihood-based models explicitly represent the density function, commonly using an unnormalized probabilistic model

\[ p_{\theta}(x) = \frac{e^{f_{\theta}(x)}}{Z_{\theta}}, \]

and it is troublesome to ensure the tractability of the normalization constant \( Z_{\theta} \).

One way to avoid the normalization is to avoid the explicit density \( p_{\theta}(x) \), and represent a score function instead, where the score function is the gradient of the log density:

\[ s_{\theta}(x) = \nabla_x \log p_{\theta}(x), \]

because

\[ s_{\theta}(x) = \nabla_x \log p_{\theta}(x) = \nabla_x \log \frac{e^{f_{\theta}(x)}}{Z_{\theta}} = \nabla_x f_{\theta}(x) - \nabla_x \log Z_{\theta} = \nabla_x f_{\theta}(x). \]
When we have a score function $\nabla_x \log p_\theta(x)$, we can use it to perform sampling from the distribution $p_\theta(x)$ by using **Langevin dynamics**, which is an algorithm akin to SGD, but performing sampling instead of optimum finding. Starting with $x_0$, we iteratively set

$$x_{i+1} \leftarrow x_i + \epsilon \nabla_x \log p_\theta(x_i) + \sqrt{2\epsilon} z_i, \quad \text{where } z_i \sim \mathcal{N}(0, I).$$

When $\epsilon \to 0$ and $K \to \infty$, $x_K$ obtained by the Langevin dynamics converges to a sample from the distribution $p_\theta(x)$. 

When we have a score function $\nabla_x \log p_\theta(x)$, we can use it to perform sampling from the distribution $p_\theta(x)$ by using **Langevin dynamics**, which is an algorithm akin to SGD, but performing sampling instead of optimum finding. Starting with $x_0$, we iteratively set

$$x_{i+1} \leftarrow x_i + \epsilon \nabla_x \log p_\theta(x_i) + \sqrt{2\epsilon} z_i, \quad \text{where } z_i \sim \mathcal{N}(0, I).$$

When $\epsilon \to 0$ and $K \to \infty$, $x_K$ obtained by the Langevin dynamics converges to a sample from the distribution $p_\theta(x)$. 

When we have a score function $\nabla_x \log p_\theta(x)$, we can use it to perform sampling from the distribution $p_\theta(x)$ by using **Langevin dynamics**, which is an algorithm akin to SGD, but performing sampling instead of optimum finding. Starting with $x_0$, we iteratively set

$$x_{i+1} \leftarrow x_i + \epsilon \nabla_x \log p_\theta(x_i) + \sqrt{2\epsilon} z_i, \quad \text{where } z_i \sim \mathcal{N}(0, I).$$

When $\epsilon \to 0$ and $K \to \infty$, $x_K$ obtained by the Langevin dynamics converges to a sample from the distribution $p_\theta(x)$. 

When we have a score function $\nabla_x \log p_\theta(x)$, we can use it to perform sampling from the distribution $p_\theta(x)$ by using **Langevin dynamics**, which is an algorithm akin to SGD, but performing sampling instead of optimum finding. Starting with $x_0$, we iteratively set

$$x_{i+1} \leftarrow x_i + \epsilon \nabla_x \log p_\theta(x_i) + \sqrt{2\epsilon} z_i, \quad \text{where } z_i \sim \mathcal{N}(0, I).$$

When $\epsilon \to 0$ and $K \to \infty$, $x_K$ obtained by the Langevin dynamics converges to a sample from the distribution $p_\theta(x)$.
Score-Based Generative Modeling

Data samples
\( \{x_1, x_2, \ldots, x_N\} \overset{i.i.d.}{\sim} p(x) \)

Scores
\( s_\theta(x) \approx \nabla_x \log p(x) \)

Langevin dynamics

New samples

https://yang-song.net/assets/img/score/smld.jpg
However, estimating the score function from data is inaccurate in low-density regions.

In order to accurately estimate the score function in low-density regions, we perturb the data distribution by isotropic Gaussian noise with various noise rates $\sigma_t$:

$$q_{\sigma_t}(\tilde{x}) \overset{\text{def}}{=} \mathbb{E}_{x \sim p(x)} \left[ \mathcal{N}(\tilde{x}; x, \sigma_t^2 I) \right],$$

where the noise distribution $q_{\sigma_t}(\tilde{x}|x) = \mathcal{N}(\tilde{x}; x, \sigma_t^2 I)$ as analogous to the forward process in the diffusion models.
Noise Conditional Score Network

To train the score function \( s_\theta(x, \sigma_t) = \nabla_x \log q_{\sigma_t}(x) \), we need to minimize the following objective:

\[
\mathbb{E}_{t, \tilde{x} \sim q_{\sigma_t}} \left[ \| s_\theta(\tilde{x}, \sigma_t) - \nabla_{\tilde{x}} \log q_{\sigma_t}(\tilde{x}) \|^2 \right].
\]

It can be shown (see P. Vincent: A connection between score matching and denoising autoencoders) that it is equivalent to minimize the denoising score matching objective:

\[
\mathbb{E}_{t, x \sim p(x), \tilde{x} \sim q_{\sigma_t}(\tilde{x} | x)} \left[ \| s_\theta(\tilde{x}, \sigma_t) - \nabla_{\tilde{x}} \log q_{\sigma_t}(\tilde{x} | x) \|^2 \right].
\]

In our case, \( \nabla_{\tilde{x}} \log q_{\sigma_t}(\tilde{x} | x) = \nabla_{\tilde{x}} - \frac{\| \tilde{x} - x \|^2}{2\sigma^2_t} = -\frac{\tilde{x} - x}{\sigma^2_t} \). Because \( \tilde{x} = x + \sigma_t e \) for standard normal random variable \( e \sim \mathcal{N}(0, I) \), we can rewrite the objective to

\[
\mathbb{E}_{t, x \sim p(x), e \sim \mathcal{N}(0, I)} \left[ \| s_\theta(x + \sigma_t e, \sigma_t) - \frac{e}{\sigma_t} \|^2 \right],
\]

so the score function basically estimates the noise given a noised image.
Once we have trained the score function for various noise rates $\sigma_t$, we can sample using annealed Langevin dynamics, where we utilize using gradually smaller noise rates $\sigma_t$.

![Image](https://yang-song.net/assets/img/score/multi_scale.jpg)

**Algorithm 1** Annealed Langevin dynamics.

```
Require: $\{\sigma_i\}_{i=1}^L$, $\epsilon$, $T$.
1: Initialize $\tilde{x}_0$
2: for $i \leftarrow 1$ to $L$ do
3:     $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$ \quad $\triangleright$ $\alpha_i$ is the step size.
4:     for $t \leftarrow 1$ to $T$ do
5:         Draw $z_t \sim \mathcal{N}(0, I)$
6:         $\tilde{x}_t \leftarrow \tilde{x}_{t-1} + \frac{\alpha_i}{2} \partial_\theta (\tilde{x}_{t-1}, \sigma_i) + \sqrt{\alpha_i} z_t$
7:     end for
8:     $\tilde{x}_0 \leftarrow \tilde{x}_T$
9: end for
return $\tilde{x}_T$
```

Such a procedure is reminiscent to the reverse diffusion process sampling.

Development of GANs

- Martin Arjovsky, Soumith Chintala, Léon Bottou: Wasserstein GAN
  https://arxiv.org/abs/1701.07875
- Andrew Brock, Jeff Donahue, Karen Simonyan: Large Scale GAN Training for High Fidelity Natural Image Synthesis https://arxiv.org/abs/1809.11096
Figure 1 of "Large Scale GAN Training for High Fidelity Natural Image Synthesis", https://arxiv.org/abs/1809.11096

Figure 2 of "Large Scale GAN Training for High Fidelity Natural Image Synthesis", https://arxiv.org/abs/1809.11096
Figure 7 of "Large Scale GAN Training for High Fidelity Natural Image Synthesis", https://arxiv.org/abs/1809.11096
Development of VAEs


• Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J. Fleet, M. Norouzi: 
  Image Super-Resolution via Iterative Refinement  
  https://arxiv.org/abs/2104.07636

Figure 1 of "GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models", [https://arxiv.org/abs/2112.10741](https://arxiv.org/abs/2112.10741)
Diffusion-Based Text-Conditional Image Generation

Figure 2 of "GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models", https://arxiv.org/abs/2112.10741
  
  [Link to paper](https://arxiv.org/abs/2205.11487)

**Figure 1 of “Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding”, [Link](https://arxiv.org/abs/2205.11487)**
Normalizing Flows


*Figure 1 of "Glow: Generative Flow with Invertible 1x1 Convolutions", [https://arxiv.org/abs/1807.03039](https://arxiv.org/abs/1807.03039)*