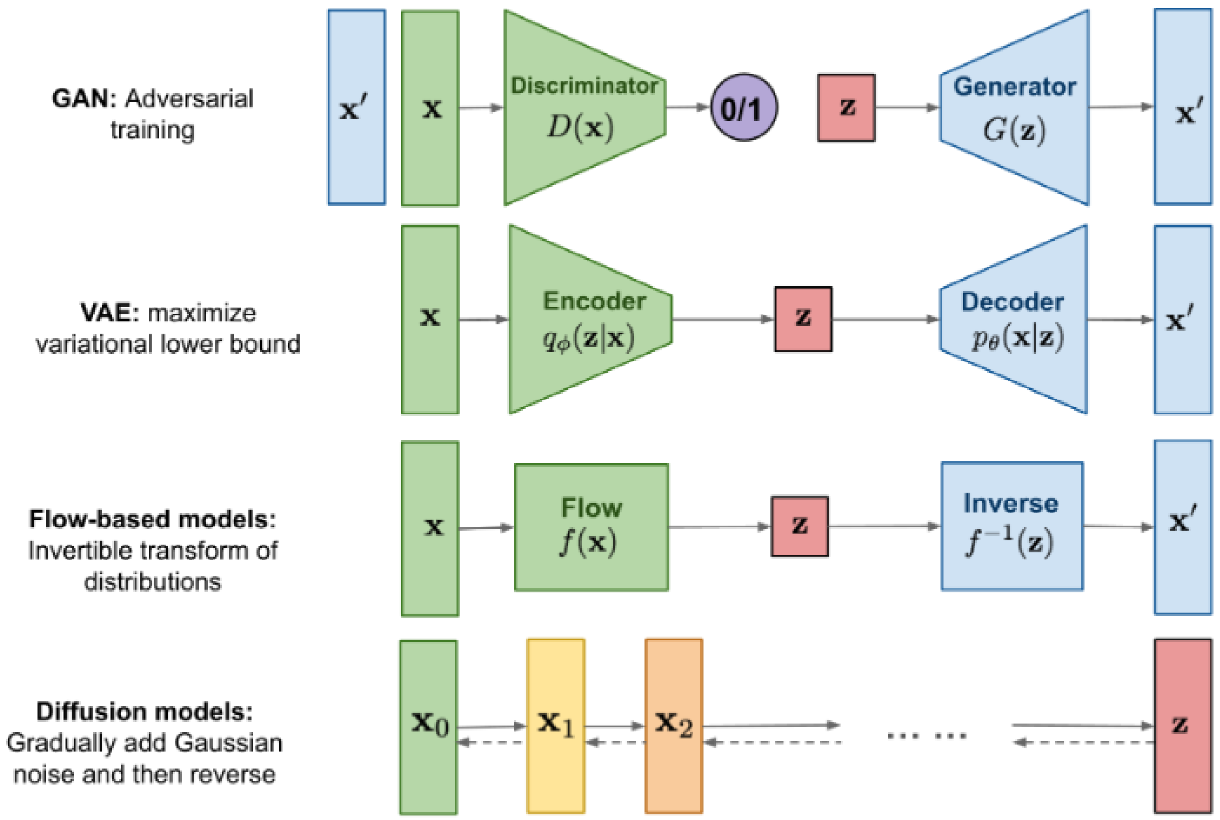
diffusion

1. Diffusion model 背景

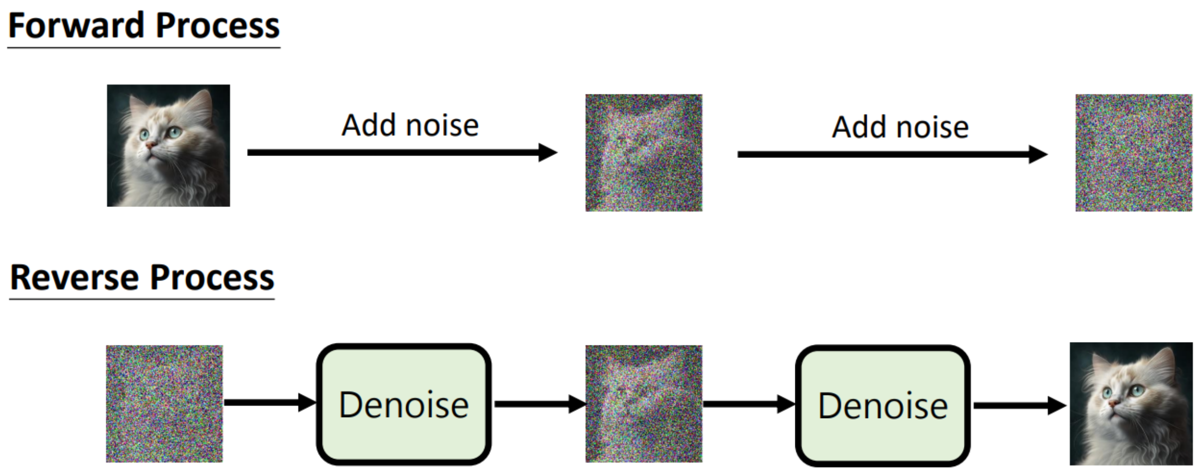
diffusion在生成式模型中大火，在许多地方都可以有使用的地方，本文主要对学习diffusion进行整理总结，同时参考github上的一些代码进行训练讲解，搞懂diffusion是什么内容



首先对于生成式模型而言，之前主要了解是的GAN相关的一些内容， GAN训练的网络是生成器和判别器，生成器用于生成图像，判别器辅助生成器的训练, 而Diffusion 训练的噪声评估网络。 GAN最大的一个问题是GAN的训练可能是不稳定的，容易出现模式崩溃和训练振荡等问题，也就是生成器更容易生成一些平均的数据骗过判别器。而Diffusion 训练loss收敛性好，比较平稳。 同时diffusion model 通过学习噪声通过不断加噪去噪的方式去优化生成的内容，有不断加深网络迭代的感觉。

2. DDPM

本文所理解的diffusion主要基于DDPM(denoising diffusion probabilistic models)这篇论文



diffusion主要分为前向过程和反向过程。 前向过程也即通过输入一张原图，通过不断的加噪声，直到最后的图像为完全的噪声。这个在后面推导可以链接，不断加噪声的过程可以直接从 x_0 到 某一个时间t得到 x_t 的图像结果 但是反向过程需要不断的计算迭代，得到从 X_T 到 X_0

先放一张 训练和采样对应的算法框图，后面一点点解释

Algorithm 1 Training

1: **repeat**
2: $\quad \mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $\quad t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4: $\quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: $\quad$ Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

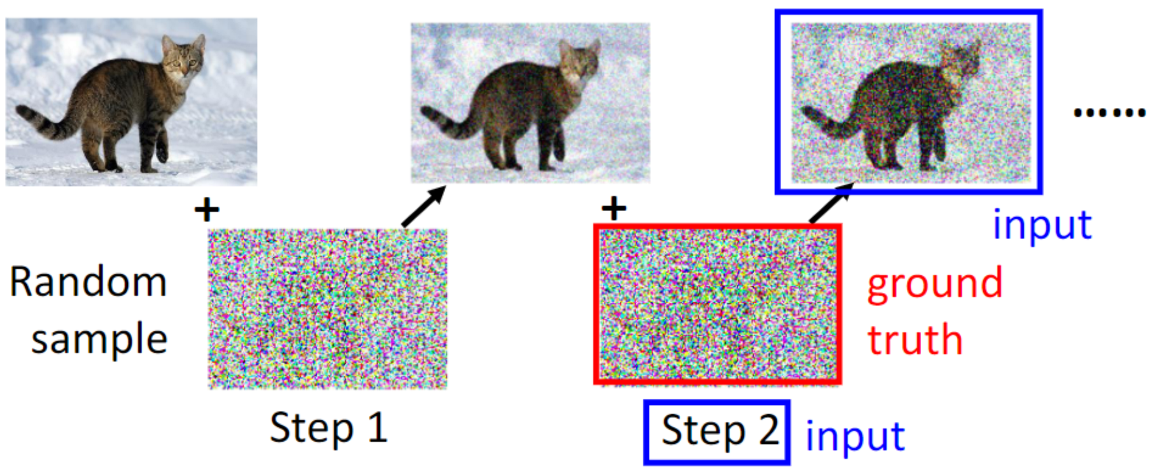
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

2.1 Diffusion 前向过程

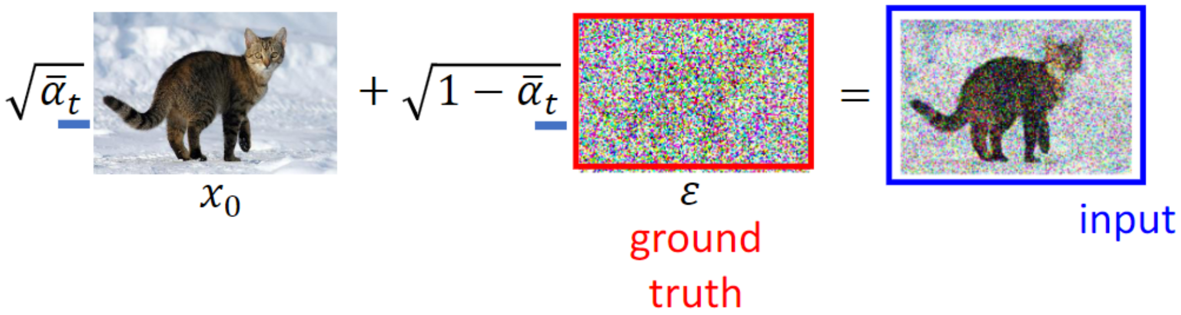
对于diffusion 的前向过程，是不断加噪声， 同时构建训练噪声GT的过程。

给定真实图像 x_0 diffusion 前向过程通过T次不断添加高斯噪声，得到 $x_1, x_2, \ldots, x_T$, 其中 x_0 为原图， x_T 为不断添加噪声的结果， 可以认为是一个完全的噪声。

想象中，不断对原图添加噪声的过程是先给原图添加噪声，然后得到第一步得到的带噪声的step1 图像，然后再对第二张图添加噪声。



实际上，给定需要添加的噪声步数，后可以直接添加噪声得到最后的噪声图像



下面看一个DDPM论文中的一个公式：

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}), q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}).$$

这个共识描述的就是，从 x_{t-1} 的图像如何得到 x_t 的数学表达，这个过程称为q过程。这里需要给定一系列的高斯分布方差的超参数 $\{\beta_t \in (0,1)\}_{t=1}^{T}$. 前向过程由于每个时刻t只与前一个时刻t-1有关，所以也可以看做马尔科夫过程： 随着t的增大， x_t 越来越接近噪声， 当 $T \to \infty$ 时 x_T 就是完全的高斯噪声。 公式中的 β_T 是一个超参数，控制噪声添加的大小， 且训练中是逐渐变大的 $\beta_1 < \beta_2 < \ldots < \beta_T$ 这里可以理解为开始让网络学习比较小的噪声，后面再不断增大。

2.1.1 重参数

对于diffusion而言，指的是， 将一个高斯分布 $z \sim \mathcal{N}(z; \mu_\theta, \sigma_\theta^2\mathbf{I})$ 采样一个 z ,我们可以写成：

$z = \mu_\theta + \sigma_\theta \odot \epsilon, \epsilon \sim \mathcal{N}(0, \mathbf{I})$

上面的 z 依然就有随机性， 且满足均值为 μ 方差为 σ_θ 的分布， 且均值和方差可以有神经网络学习到， 这个过程中，梯度依旧可导， 随机性被转嫁到 ϵ 上了。

2.1.2 任意时刻的 x_t 表示

前面提到， 实际上的采样过程可以直接由 x_0 通过一次采样得到 x_t , 首先假设 $\alpha_t = 1 - \beta_t$, 并且 $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$,
对 x_t 通过一层层的展开可以得到：

$$\begin{aligned}
x_t &= \sqrt{a_t}x_{t-1} + \sqrt{1-\alpha_t}z_1 \quad \text{where} \quad z_1, z_2, \ldots \sim \mathcal{N}(0, \mathbf{I}); \\
&= \sqrt{a_t}\left(\sqrt{a_{t-1}}x_{t-2} + \sqrt{1-\alpha_{t-1}}z_2\right) + \sqrt{1-\alpha_t}z_1 \\
&= \sqrt{a_t a_{t-1}}x_{t-2} + \left(\sqrt{a_t(1-\alpha_{t-1})}z_2 + \sqrt{1-\alpha_t}z_1\right) \\
&= \sqrt{a_t a_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{z}_2 \quad \text{where} \quad \bar{z}_2 \sim \mathcal{N}(0, \mathbf{I}); \\
&= \ldots \\
&= \sqrt{\bar{\alpha}}x_0 + \sqrt{1-\bar{\alpha}_t}\bar{z}_t.
\end{aligned}$$

由于独立高斯的分布可加性， $\mathcal{N}(0, \sigma_1^2\mathbf{I}) + \mathcal{N}(0, \sigma_2^2\mathbf{I}) \sim \mathcal{N}(0, (\sigma_1^2 + \sigma_2^2)\mathbf{I})$

对上面的式子可以进行一个化简：

$\sqrt{a_t(1-\alpha_{t-1})}z_2 \sim \mathcal{N}(0, a_t(1-\alpha_{t-1})\mathbf{I})$
$\sqrt{1-\alpha_t}z_1 \sim \mathcal{N}(0,(1-\alpha_t)\mathbf{I})$
$\sqrt{a_t(1-\alpha_{t-1})}z_2 + \sqrt{1-\alpha_t}z_1 \sim \mathcal{N}(0,[\alpha_t(1-\alpha_{t-1})+(1-\alpha_t)]\mathbf{I})$
$= \mathcal{N}(0,(1-\alpha_t\alpha_{t-1})\mathbf{I})$.

这里需要强调的是，z_1, z_2, z_3, z_4 都是独立的高斯分布， 对于任意时刻的 x_t 都满足 $q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\bar{a}_t}x_0, (1-\bar{a}_t)\mathbf{I})$

2.2 Diffusion 反向过程

反向过程就是diffusion模型从最后的噪声 x_t 通过不断的去噪，得到最后的 x_0 的过程。 过程表示而言指的是前向过程: $q(x_t \mid x_{t-1})$, 逆向过程 $q(x_{t-1} \mid x_t)$

现在的问题转化为，加入 x_t 成为输入（后面代码可以看到，网络推理的时候输入为噪声）， 怎么反向求解得到一张清晰的图像 x_0 .

$$p_\theta(X_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)$$
$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

虽然我们无法得到逆转后的分布 $q(x_{t-1} \mid x_t)$, 但是假设我们知道了 x_0 的分布(前面提到 x_t 和 x_0 的转换关系)就可以通过贝叶斯公式进行一个转换。

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0)\frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)}$$

对于上面这个贝叶斯公式中的三个式子，起始我们都是可以通过前面前向的求解展开的

$$\begin{aligned}
&q(\mathbf{x}_{t-1} \mid \mathbf{x}_0) \quad \sqrt{\bar{a}_t}x_0 + \sqrt{1-\bar{a}_t-1}z \quad \sim \mathcal{N}(\sqrt{\bar{a}_t-1}x_0, 1-\bar{a}_{t-1})\\
&q(\mathbf{x}_t \mid \mathbf{x}_0) \quad \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}z \quad \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}x_0, 1-\bar{\alpha}_t)\\
&q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0) = \sqrt{a_t}x_{t-1} + \sqrt{1-\alpha_t}z \quad \sim \mathcal{N}(\sqrt{a_t}x_{t-1}, 1-\alpha_t)
\end{aligned}$$

将上面的式子带入就可以得到 $q(x_{t-1} \mid x_t, x_0)$ 的分布为： $q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t\mathbf{I})$

具体的推理过程为：

$$\begin{aligned}
q(x_{t-1}|x_t, x_0) &= q(x_t|x_{t-1}, x_0)\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)}\\
&\propto \exp\left(-\frac{1}{2}\left(\frac{(x_t-\sqrt{\alpha_t}x_{t-1})^2}{\beta_t} + \frac{(x_{t-1}-\sqrt{\bar{\alpha}_{t-1}}x_0)^2}{1-\bar{a}_{t-1}} - \frac{(x_t-\sqrt{\bar{\alpha}_t}x_0)^2}{1-\bar{a}_t}\right)\right)\\
&= \exp\\
&\left(-\frac{1}{2}\left(\underbrace{(\frac{\alpha_t}{\beta_t}+\frac{1}{1-\bar{\alpha}_{t-1}})x_{t-1}^2}_{x_{t-1}\text{方差}} - \underbrace{(\frac{2\sqrt{\alpha_t}}{\beta_t}x_t + \frac{2\sqrt{\bar{a}_{t-1}}}{1-\bar{\alpha}_{t-1}}x_0)x_{t-1}}_{x_{t-1}\text{均值}} + \underbrace{C(x_t, x_0)}_{\text{与}x_{t-1}\text{无关}}\right)\right).
\end{aligned}$$

一般的高斯概率密度函数的指数部分应该写为 $\exp(-\frac{(x-\mu)^2}{2\sigma^2}) = \exp(-\frac{1}{2}(\frac{1}{\sigma^2}x^2 - \frac{2\mu}{\sigma^2}x + \frac{\mu^2}{\sigma^2}))$ 因此稍加整理我们可以

$$\frac{1}{\sigma^2} = \frac{1}{\tilde{\beta}_t} = (\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}}); \quad \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t$$

$$\frac{2\mu}{\sigma^2} = \frac{2\tilde{\mu}_t(x_t, x_0)}{\tilde{\beta}_t} = (\frac{2\sqrt{\alpha_t}}{\beta_t}x_t + \frac{2\sqrt{\bar{a}_{t-1}}}{1-\bar{\alpha}_{t-1}}x_0);$$

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

得

根据2.1.2提到的， $x_0 = \frac{1}{\sqrt{\bar{a}_t}}(x_t - \sqrt{1-\bar{a}_t}\bar{z}_t)$，带入到上面式子可以得到 $\tilde{\mu}_t = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{a}_t}}\bar{z}_t)$

其中 $\bar{z}_t$ 就是模型需要预测的噪声， 可看做为 $z_\theta(x_t, t)$，可以得到：

ParseError: KaTeX parse error: Can't use function '$' in math mode at position 1: $_ \mu_\theta\left…
总结一下sample也即diffusion 反向推理过程

1. 根据时间 x_t 和t 通过模型，遇到高斯噪声 $z_\theta(x_t, t)$，从而得到均值 $\mu_\theta(x_t, t)$
2. DDPM中使用的是非训练的方式获取方差 $\tilde{\beta}_t$， $\tilde{\beta}_t = \beta_t$ 和 $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t$ 得到的结果近似。 但是在其他的一些论文中也有通过网络学习得到的方差（GLIDE）
3. 不断迭代，得到最后的结果

2.3 Diffusion 训练过程

diffusion的训练过程通过训练得到一个合理的 $\mu_\theta(x_t, t)$ 和 $\Sigma_\theta(x_t, t)$，通过对真实数据分布下，最大化模型预测分布的对数似然， 即优化在 $x_0 \sim q(x_0)$ 下的 $p_\theta(x_0)$ 的交叉熵（熵越大，信息量越大）

下面是一些推导公式：

$$\begin{aligned}
-\log p_\theta(x_0) &\le -\log p_\theta(x_0) + D_{KL}(q(x_{1:T}|x_0)||p_\theta(x_{1:T}|x_0)) \\
&= -\log p_\theta(x_0) + \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})/p_\theta(x_0)}\right]; \quad \text{where} \quad p_\theta(x_{1:T}|x_0) = \frac{p_\theta(x_{0:T})}{p_\theta(x_0)} \\
&= -\log p_\theta(x_0) + \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})} + \underbrace{\log p_\theta(x_0)}_{\text{与}q\text{无关}}\right] \\
&= \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}\right].
\end{aligned}$$

对上式子左右去期望，利用到重积分中的Fubini定理：

$$\mathcal{L}_{VLB} = \underbrace{\mathbb{E}_{q(x_0)}\left(\mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}\right]\right) = \mathbb{E}_{q(x_{0:T})}\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}\right]}_{Fubini\text{定理}} \ge \mathbb{E}_{q(x_0)}[-\log p_\theta(x_0)].$$

通过最小化 L_{VLB} 就可以达到最大化熵的目的

$$
\begin{aligned}
L_{\mathrm{VLB}} &= \mathbb{E}_{q(\mathbf{x}_{0:T})}\Big[\log \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\Big] \\
&= \mathbb{E}_q\Big[\log \frac{\prod_{t=1}^T q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_T)\prod_{t=1}^T p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=1}^T \log \frac{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)} + \log \frac{q(\mathbf{x}_1 \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \Big(\frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)} \cdot \frac{q(\mathbf{x}_t \mid \mathbf{x}_0)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}\Big) + \log \frac{q(\mathbf{x}_1 \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)} + \sum_{t=2}^T \log \frac{q(\mathbf{x}_t \mid \mathbf{x}_0)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1 \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)} + \log \frac{q(\mathbf{x}_T \mid \mathbf{x}_0)}{q(\mathbf{x}_1 \mid \mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1 \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[\log \frac{q(\mathbf{x}_T \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_T)} + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)} - \log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)\Big] \\
&= \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T \mid \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^T \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)) - \log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)]}_{L_{t-1}}
\end{aligned}
$$

也可写为：

$$
\begin{aligned}
&\mathcal{L}_{VLB} = L_T + L_{T-1} + \ldots + L_0 \\
&L_T = D_{KL}(q(x_T|x_0)||p_\theta(x_T)) \\
&L_t = D_{KL}(q(x_t|x_{t+1}, x_0)||p_\theta(x_t|x_{t+1})); \qquad 1 \leq t \leq T-1 \\
&L_0 = -\log p_\theta(x_0|x_1).
\end{aligned}
$$

这一部分为了文章的完整性，截图了一些内容

由于前向 q 没有可学习参数，而 x_T 则是纯高斯噪声， L_T 可以当做常量忽略。而 L_t 则可以看做拉近2个高斯分布 $q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t \mathbf{I})$ 和 $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta)$ ，根据多元高斯分布的KL散度求解：

$$
L_t = \mathbb{E}_q\left[\frac{1}{2||\Sigma_\theta(x_t, t)||_2^2}||\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)||^2\right] + C,
$$

其中C为常数，把 $\tilde{\mu}_t(x_t, x_0)$ 和 $\mu_\theta(x_t, t)$ 带入可得：

$$= \mathrm{E}_{x_0,\bar{z}_t}\left[\frac{1}{2\,\|\Sigma_\theta(x_t,t)\|_2^2}\,\|\tilde{\mu}_t(x_t,x_0)-\mu_\theta(x_t,t)\|^2\right]$$

$$= \mathrm{E}_{x_0,\bar{z}_t}\left[\frac{1}{2\,\|\Sigma_\theta(x_t,t)\|_2^2}\,\|\frac{1}{\sqrt{a_t}}(x_t-\frac{\beta_t}{\sqrt{1-\bar{a}_t}}\bar{z}_t)-\frac{1}{\sqrt{a_t}}(x_t-\frac{\beta_t}{\sqrt{1-\bar{a}_t}}z_\theta(x_t,t)\right.$$

$$= \mathrm{E}_{x_0,\bar{z}_t}\left[\frac{\beta_t^2}{2\alpha_t(1-\bar{\alpha}_t\,\|\Sigma_\theta\|_2^2)}\,\|\bar{z}_t-z_\theta(x_t,t)\|^2\right]$$

$$= \mathrm{E}_{x_0,\bar{z}_t}\left[\frac{\beta_t^2}{2\alpha_t(1-\bar{\alpha}_t\,\|\Sigma_\theta\|_2^2)}\left\|\bar{z}_t-z_\theta(\sqrt{\bar{\alpha}_t}x_0+\sqrt{1-\bar{\alpha}_t}\bar{z}_t,t)\right\|^2\right]$$

从上面推导可以看出diffusion训练的核心就是取学习高斯噪声 $\bar{z}_t, z_\theta$ 之间的MSE。

DDPM将loss进一步简化为:

$$L_t^{simple} = \mathbb{E}_{x_0,\bar{z}_t}\left[||\bar{z}_t - z_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\bar{z}_t, t)||^2\right].$$

因此训练过程可总结为：

1. 获取输入 x_0 ，从1...T随机采样一个t.
2. 从标准高斯分布采样一个噪声 $\bar{z}_t \sim \mathrm{N}(0, \mathbf{I})$
3. 最小化 $\left\|\bar{z}_t - z_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\bar{z}_t, t)\right\|$ (这里需要解释的是)

代码讲解

参考代码：https://github.com/dome272/Diffusion-Models-pytorch/blob/main/ddpm.py 代码比较简单，对应上面的讲解主要分为三个部分 前向：

```
class Diffusion:
    def __init__(self, noise_steps=1000, beta_start=1e-4, beta_end=0.02, img_size=256,
device="cuda"):
        self.noise_steps = noise_steps
        self.beta_start = beta_start
        self.beta_end = beta_end
        self.img_size = img_size
        self.device = device

        self.beta = self.prepare_noise_schedule().to(device)
        self.alpha = 1. - self.beta
        self.alpha_hat = torch.cumprod(self.alpha, dim=0)

    def prepare_noise_schedule(self):
        return torch.linspace(self.beta_start, self.beta_end, self.noise_steps)

    def noise_images(self, x, t):
        sqrt_alpha_hat = torch.sqrt(self.alpha_hat[t])[:, None, None, None]
        sqrt_one_minus_alpha_hat = torch.sqrt(1 - self.alpha_hat[t])[:, None, None, None]
        Ɛ = torch.randn_like(x)
        return sqrt_alpha_hat * x + sqrt_one_minus_alpha_hat * Ɛ, Ɛ

    def noise_images(self, x, t): # 前向过程
        sqrt_alpha_hat = torch.sqrt(self.alpha_hat[t])[:, None, None, None]
```

```
        sqrt_one_minus_alpha_hat = torch.sqrt(1 - self.alpha_hat[t])[:, None, None, None]
        Ɛ = torch.randn_like(x)
        return sqrt_alpha_hat * x + sqrt_one_minus_alpha_hat * Ɛ, Ɛ
```

后向过程:

```
    def sample(self, model, n):
        logging.info(f"Sampling {n} new images....")
        model.eval()
        with torch.no_grad():
            x = torch.randn((n, 3, self.img_size, self.img_size)).to(self.device)
            for i in tqdm(reversed(range(1, self.noise_steps)), position=0):
                t = (torch.ones(n) * i).long().to(self.device)
                predicted_noise = model(x, t)
                alpha = self.alpha[t][:, None, None, None]
                alpha_hat = self.alpha_hat[t][:, None, None, None]
                beta = self.beta[t][:, None, None, None]
                if i > 1:
                    noise = torch.randn_like(x)
                else:
                    noise = torch.zeros_like(x)
                x = 1 / torch.sqrt(alpha) * (x - ((1 - alpha) / (torch.sqrt(1 - alpha_hat))) *
predicted_noise) + torch.sqrt(beta) * noise
        model.train()
        x = (x.clamp(-1, 1) + 1) / 2
        x = (x * 255).type(torch.uint8)
        return x
```

训练过程:

```
def train(args):
    setup_logging(args.run_name)
    device = args.device
    dataloader = get_data(args)
    model = UNet().to(device)
    optimizer = optim.AdamW(model.parameters(), lr=args.lr)
    mse = nn.MSELoss()
    diffusion = Diffusion(img_size=args.image_size, device=device)
    logger = SummaryWriter(os.path.join("runs", args.run_name))
    l = len(dataloader)

    for epoch in range(args.epochs):
        logging.info(f"Starting epoch {epoch}:")
        pbar = tqdm(dataloader)
        for i, (images, _) in enumerate(pbar):
            images = images.to(device)
            t = diffusion.sample_timesteps(images.shape[0]).to(device)
            x_t, noise = diffusion.noise_images(images, t)
            predicted_noise = model(x_t, t)
            loss = mse(noise, predicted_noise)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            pbar.set_postfix(MSE=loss.item())
            logger.add_scalar("MSE", loss.item(), global_step=epoch * l + i)

        sampled_images = diffusion.sample(model, n=images.shape[0])
        save_images(sampled_images, os.path.join("results", args.run_name, f"{epoch}.jpg"))
        torch.save(model.state_dict(), os.path.join("models", args.run_name, f"ckpt.pt"))
```

可以重点看一下loss = mse(noise, predicted_noise) 计算的过程

总结

大致diffusion过程有了更直观的了解，后面由使用的机会继续完善这个专题

参考资料：

由浅入深了解Diffusion Model

What are Diffusion Models?

参考代码

李宏毅 PPT

DDPM论文

Understanding Diffusion Model