Stable Diffusion理解与代码解读

2023-03-01 016:55

ref: https://zhuanlan.zhihu.com/p/525106459

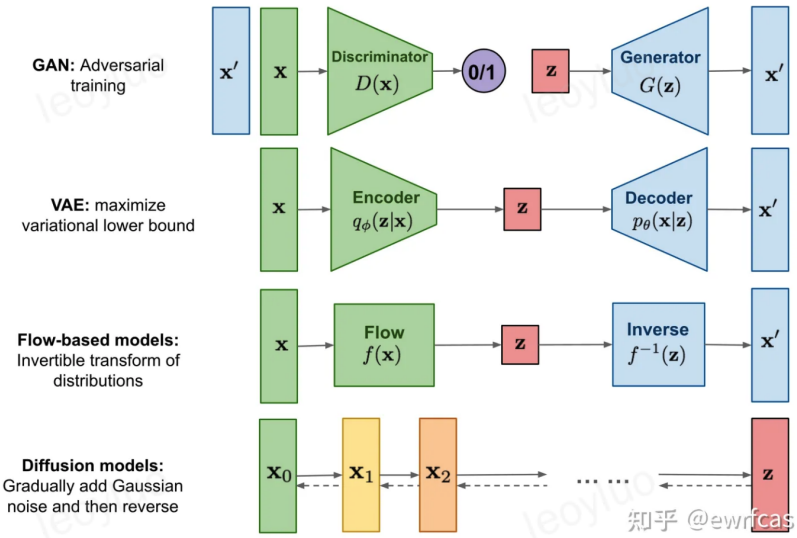


图3. 不同生成模型对比图（来源：Lil博客）

diffusion的latent尺寸和原始尺寸一致

Denoising diffusion probabilistic models (DDPM)

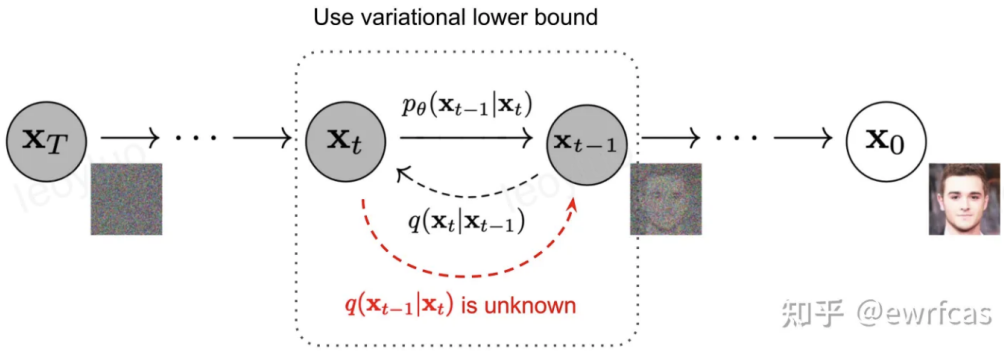


图4. diffusion的前向(q)和逆向(p)过程，来源：DDPM

在所有的时间T

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

Loss推导

Simplified loss

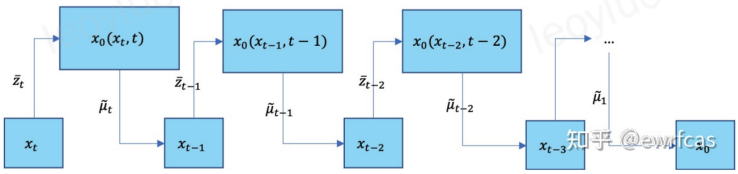
$$L_{\text{simple}}(\theta) = \mathbb{E}_{t,x_0,\epsilon}\left[\left\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\right\|^2\right]$$

逆向过程

1) 每个时间步通过 x_t 和t 来预测高斯噪声$z_\theta(x_t,t)$，随后根据(9)得到均值 $\mu_\theta(x_t,t)$.

2) 得到方差 $\Sigma_\theta(x_t,t)$ ，DDPM中使用untrained $\Sigma_\theta(x_t,t)=\tilde{\beta}_t$ ，且认为 $\tilde{\beta}_t=\beta_t$ 和 $\tilde{\beta}_t=\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\cdot\beta_t$ 结果近似，在GLIDE中则是根据网络预测trainable方差 $\Sigma_\theta(x_t,t)$.

3) 根据(5-2)得到 $q(x_{t-1}|x_t)$ ，利用重参数得到 x_{t-1} .



在x0和xt反复横跳的diffusion逆向过程

Algorithm 1 Training

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

Algorithm 2 Sampling

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

DDPM训练测试算法流程图（来源：DDPM）

DDPM简单训练代码

ref:

https://github.com/w86763777/pytorch-ddpm.git.

https://github.com/hojonathanho/diffusion

https://nn.labml.ai/diffusion/ddpm/index.html

- **模型**
  - Unet with
    - time embedding (t→embedding by sin/cos(t*pos) and linear)
    - downSample + Resblock(attentBlock) + upsample
  - 保证输入输出一致

- Gaussian Diffusion Trainer
  - beta_1, beta_T, T: 初始beta/最后beta/时间stepT
  - beta在1~T中插值, alpha=1−alpha
  - alpha_bar = cumprod(alpha), 求出sqrt(a_bar) / sqrt(1−a_bar)

```python
def forward(self, x_0):
    """
    Algorithm 1.
    """
    t = torch.randint(self.T, size=(x_0.shape[0], ), device=x_0.device)
    noise = torch.randn_like(x_0)
    x_t = (
        extract(self.sqrt_alphas_bar, t, x_0.shape) * x_0 +
        extract(self.sqrt_one_minus_alphas_bar, t, x_0.shape) * noise)
    loss = F.mse_loss(self.model(x_t, t), noise, reduction='none')
    return loss
```

- extract就是获取对应的time t
- 每一步的训练是随机找一个t, 求出 $X_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\bar{z}_t$, 判断与noise的区别

- Gaussian Diffusion Sampler
  - 从Xt(高斯nosie) 恢复成为X0(原图)的逆过程
  - $$\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)$$
  - ```python
    def forward(self, x_T):
        """
        Algorithm 2.
        """
        x_t = x_T
        for time_step in reversed(range(self.T)):
            t = x_t.new_ones([x_T.shape[0], ], dtype=torch.long) * time_step
            mean, log_var = self.p_mean_variance(x_t=x_t, t=t)
            # no noise when t == 0
            if time_step > 0:
                noise = torch.randn_like(x_t)
            else:
                noise = 0
            x_t = mean + torch.exp(0.5 * log_var) * noise
        x_0 = x_t
        return torch.clip(x_0, -1, 1)
    ```
- ddpm操作步数多, ddim优化了它

==========

Stable diffusion

= VAE + latent space diffusion + conditioning

ref:

https://zhuanlan.zhihu.com/p/581225163

https://zhuanlan.zhihu.com/p/583124756

https://www.high-flyer.cn/blog/stable-diffusion/

https://zhuanlan.zhihu.com/p/591432516

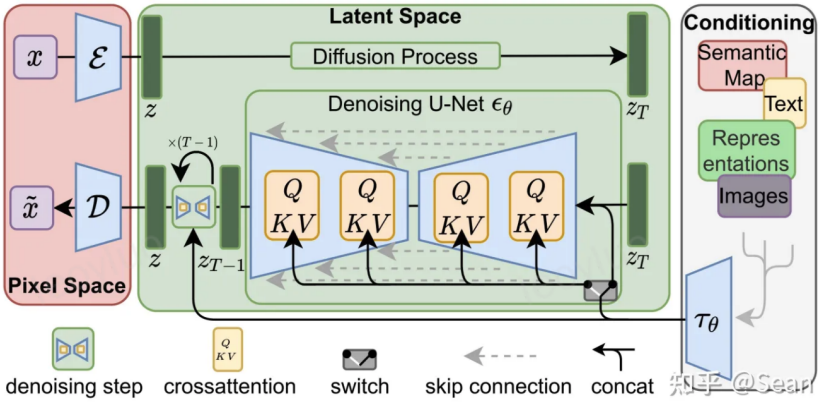
- 原本的DDPM是一个Image to Image的模型, 通过noise生成结果. sd是加入condtional input, 跨模态进行输出


    ```

- 在latent space进行diffusion可以大大降低迭代的step数量 (LDM)
- 通过引入一个领域内的编码器, 将图片进行condition的生成(Unet的每一层都加入了condition)
- 输入的模态可以有 image, segement, depth, text等等

- 首先要训练一个vae模型, 在latent space上进行diffusion操作(比如在imagenet上训练)
- diffusion变成了如何从noise生成latent code, 然后通过解码器恢复图像

核心是一个latent diffusion model



- 核心的context融合, 通过将latent space的z和condition的z通过attention进行融合, 然后在进行去噪声的过程

==========

ControlNet

ref:

https://zhuanlan.zhihu.com/p/609075353

https://zhuanlan.zhihu.com/p/605761756

- 通过另一个网络实现对原本sd模型的微调
- 相当于把text prompt的condition 替换成了其他depth/seg/edge的引导. 最重要的是统一了所有不同的输入