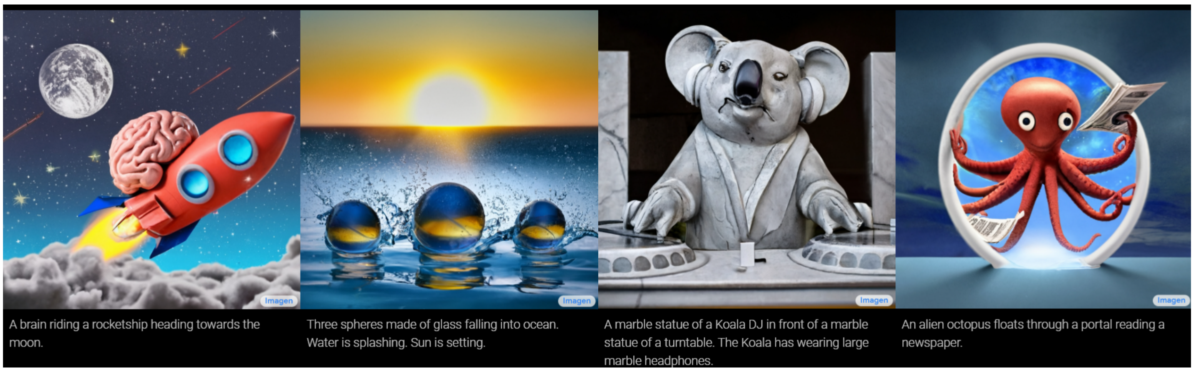
Denoising Diffusion Probabilistic Models (DDPM)

Diffusion Model 基本介绍 吴斌

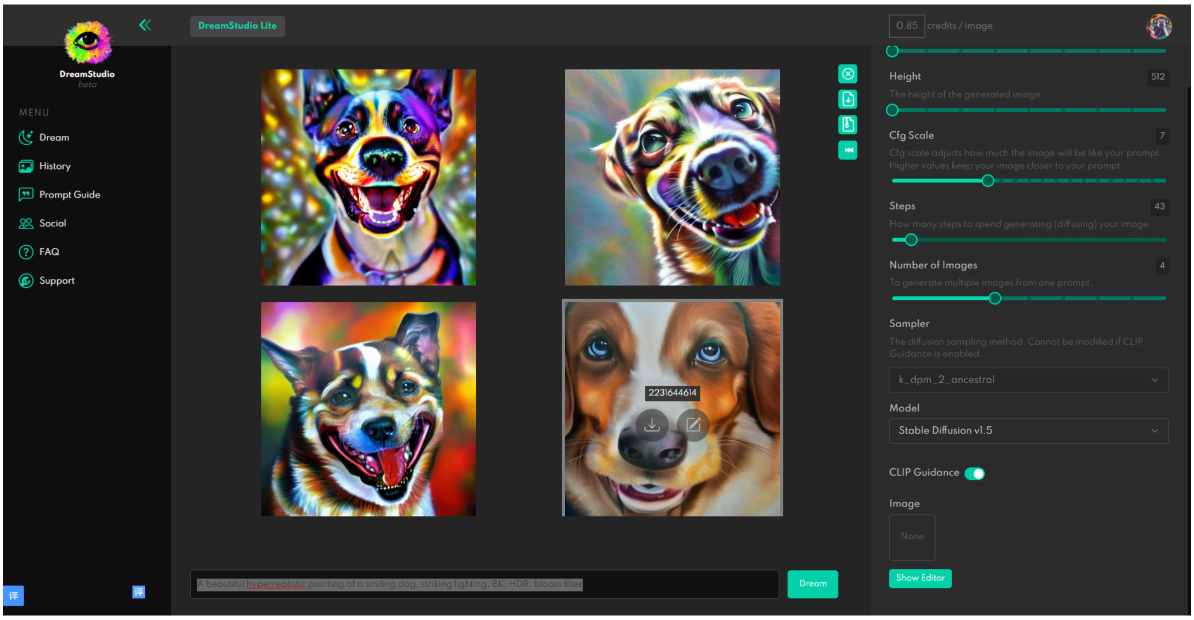
扩散模型（Diffusion Models）[1] 发表以来其实并没有收到太多的关注，因为他不像 GAN 那样简单粗暴好理解。不过最近这几年正在生成模型领域异军突起，当前最先进的两个文本生成图像——OpenAI 的 DALL·E 2 和 Google 的 Imagen，都是基于扩散模型来完成的。stable difussion也是今年异军突起的一个图像绘图模型，它开源且可以在消费级显卡运行。

> Ho J, Jain A, Abbeel P. Denoising diffusion probabilistic models[J]. Advances in Neural Information Processing Systems, 2020, 33: 6840-6851.

DALLE2



stable difussion -> Dream Studio

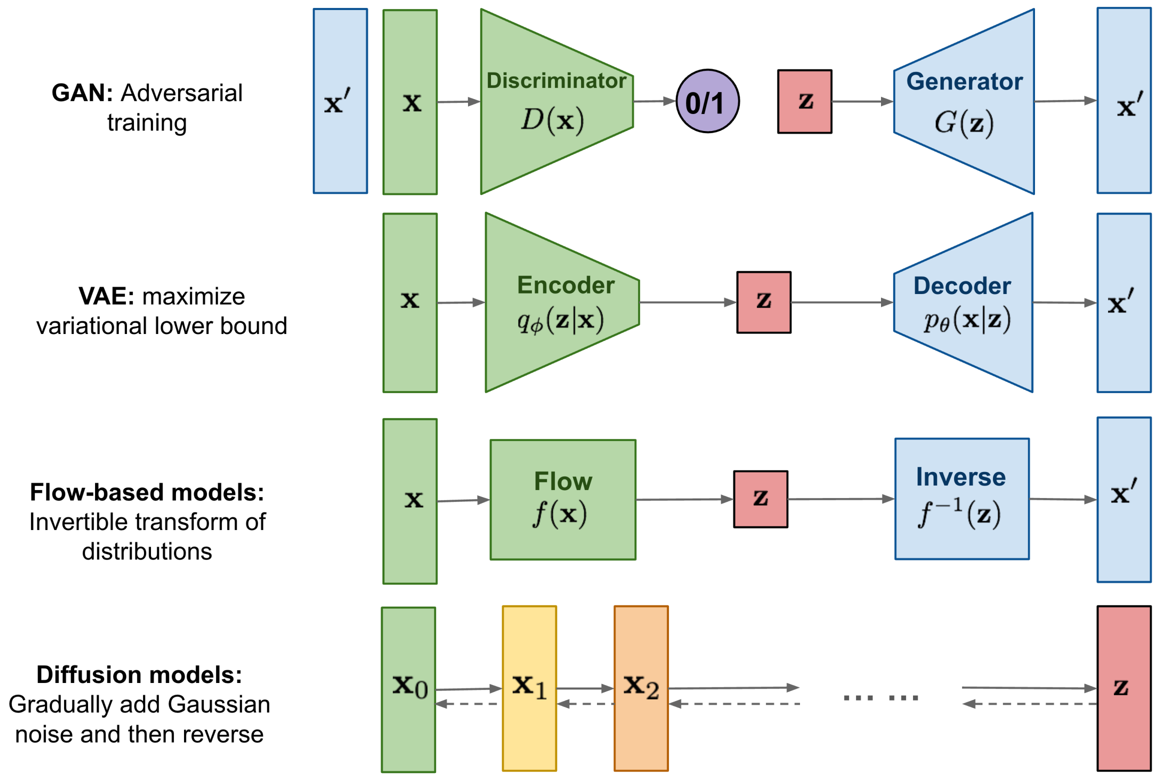


A beautiful hyperrealistic painting of a smiling dog, striking lighting, 8K, HDR, bloom filter

a painting of a space station in the sky, concept art by Peter Elson, cgsociety, space art, sci-fi, redshift, concept art

a dram of the moon falling down on the paddy field.by Victo Ngai.

生成模型对比



GAN可能是最常用的生成模型，但是不容易训练，不容易收敛。目前对于GAN能挖掘的都挖掘了，前景有限。Diffusion Model目前起步不久，各项研究正在百花齐放。Diffusion Models 的灵感来自non-equilibrium thermodynamics （非平衡热力学）。理论首先定义扩散步骤的马尔可夫链，以缓慢地将随机噪声添加到数据中，然后学习逆向扩散过程以从噪声中构造所需的数据样本。

具体理论过程

Diffusion Models 既然叫生成模型，这意味着 Diffusion Models 用于生成与训练数据相似的数据。从根本上说，Diffusion Models 的工作原理，是通过连续添加高斯噪声来破坏训练数据，然后通过反转这个噪声过程，来学习恢复数据。

训练后，可以使用 Diffusion Models 将随机采样的噪声传入模型中，通过学习去噪过程来生成数据。也就是下面图中所对应的基本原理，不过这里面的图仍然有点粗。

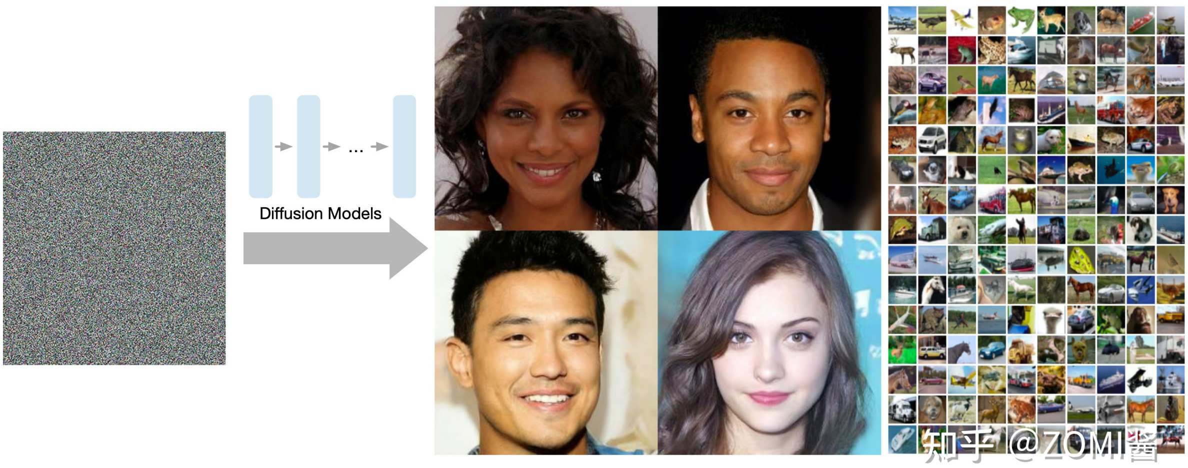


Figure 1: Generated samples on CelebA-HQ 256 × 256 (left) and unconditional CIFAR10 (right)

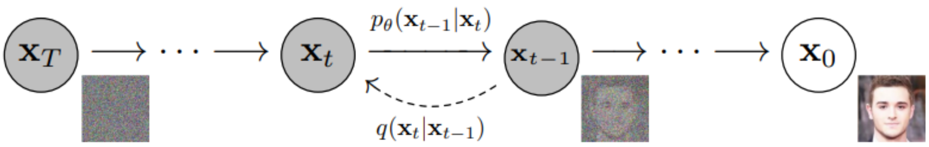


Figure 2: The directed graphical model considered in this work.

扩散过程

扩散过程：

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

首先由$x_t - 1$知道了x_t的高斯分布。另外我们需要在这个分布中采样出具体的x_t，为了过程可导，作者利用了重参数技巧。

即从$x_t \sim N(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \mathrm{I})$采样一个$x_t$，可以写成:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \beta_t \mathrm{I} \odot \varepsilon, \varepsilon \sim N(0, \mathrm{I})$$

由x_0到x_t一步步进行太麻烦，作者推到了一步到位的公式，直接由x_0得出x_t：

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

其中，$\overline{\alpha}_t=\alpha_t\alpha_{t-1}\cdots\alpha_1,\quad \alpha=1-\beta$

一直推导，最终，x_t的采样公式：

$$x_t=\sqrt{\overline{\alpha}_t}x_0+(1-\overline{\alpha}_t)\mathrm{I}\odot\varepsilon,\varepsilon\sim N(0,\mathrm{I})$$

上述公式非常重要

逆扩散过程

$q(x_{t-1}|x_t)$很难求，所以作者求$q(x_{t-1}|x_t,x_0)$。

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

$$\tilde{\mu}_t=\frac{1}{\sqrt{\bar{\alpha}_t}}(x_t-\frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_t),\tilde{\beta}_t=\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

最终，x_t-1的采样公式：

$$x_{t-1}=\frac{1}{\sqrt{\bar{\alpha}_t}}(x_t-\frac{1-\alpha_t}{1-\bar{\alpha}_t}\varepsilon_\theta(x_t,t))+\sigma_t\mathrm{z},\mathrm{z}\sim(0,\mathrm{I})$$

上述公式非常重要

具体流程伪代码：

Algorithm 1 Training

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

Algorithm 2 Sampling

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

代码细节

定义参数

class GaussianDiffusion(nn.Module)

```
# 定义需要使用的相关参数
alphas = 1.0 - betas # beta 0.0001->0.02
alphas_cumprod = np.cumprod(alphas)

to_torch = partial(torch.tensor, dtype=torch.float32)

self.register_buffer("betas", to_torch(betas))
self.register_buffer("alphas", to_torch(alphas))
self.register_buffer("alphas_cumprod", to_torch(alphas_cumprod))

self.register_buffer("sqrt_alphas_cumprod", to_torch(np.sqrt(alphas_cumprod)))
```

```
self.register_buffer("sqrt_one_minus_alphas_cumprod", to_torch(np.sqrt(1 -
alphas_cumprod)))
self.register_buffer("reciprocal_sqrt_alphas", to_torch(np.sqrt(1 / alphas)))

self.register_buffer("remove_noise_coeff", to_torch(betas / np.sqrt(1 -
alphas_cumprod)))
self.register_buffer("sigma", to_torch(np.sqrt(betas)))

# 使用时根据需要索引相应t对应的参数
```

```
betas = generate_linear_schedule(
            args.num_timesteps,
            args.schedule_low * 1000 / args.num_timesteps,
            args.schedule_high * 1000 / args.num_timesteps,
        )
```

训练

```
def forward(self, x, y=None):
    b, c, h, w = x.shape
    device = x.device

    if h != self.img_size[0]:
        raise ValueError("image height does not match diffusion parameters")
    if w != self.img_size[0]:
        raise ValueError("image width does not match diffusion parameters")

     t = torch.randint(0, self.num_timesteps, (b,), device=device)
    # 为每一张图片分配一个t,t属于(1,T)
     return self.get_losses(x, t, y)

def get_losses(self, x, t, y):
noise = torch.randn_like(x)

perturbed_x = self.perturb_x(x, t, noise)  # xt
estimated_noise = self.model(perturbed_x, t, y)

if self.loss_type == "l1":
loss = F.l1_loss(estimated_noise, noise)
elif self.loss_type == "l2":
loss = F.mse_loss(estimated_noise, noise)

return loss

def perturb_x(self, x, t, noise):
    return (
        extract(self.sqrt_alphas_cumprod, t, x.shape) * x +
        extract(self.sqrt_one_minus_alphas_cumprod, t, x.shape) * noise
    )
```

采样过程

```
    @torch.no_grad()
    def remove_noise(self, x, t, y, use_ema=True):
        if use_ema:
            return (
                (x - extract(self.remove_noise_coeff, t, x.shape) *
self.ema_model(x, t, y)) * extract(self.reciprocal_sqrt_alphas, t, x.shape)
            )
        else:
            return (
                (x - extract(self.remove_noise_coeff, t, x.shape) *
self.model(x, t, y))                    *
extract(self.reciprocal_sqrt_alphas, t, x.shape)
            )

    @torch.no_grad()
    def sample(self, batch_size, device, y=None, use_ema=True):
        if y is not None and batch_size != len(y):
            raise ValueError("sample batch size different from length of given
y")

        x = torch.randn(batch_size, self.img_channels, *self.img_size,
device=device)

        for t in range(self.num_timesteps - 1, -1, -1):
            t_batch = torch.tensor([t], device=device).repeat(batch_size)
            x = self.remove_noise(x, t_batch, y, use_ema)

            if t > 0:
                x += extract(self.sigma, t_batch, x.shape) * torch.randn_like(x)

        return x.cpu().detach()
```

UNet

```
    def forward(self, x, time=None, y=None):
        ip = self.initial_pad
        if ip != 0:
            x = F.pad(x, (ip,) * 4)

        if self.time_mlp is not None:
            if time is None:
                raise ValueError("time conditioning was specified but tim is not
passed")

            time_emb = self.time_mlp(time)
        else:
            time_emb = None

        if self.num_classes is not None and y is None:
            raise ValueError("class conditioning was specified but y is not
passed")
```

```
        x = self.init_conv(x)

        skips = [x]

        for layer in self.downs:
            x = layer(x, time_emb, y)
            skips.append(x)

        for layer in self.mid:
            x = layer(x, time_emb, y)

        for layer in self.ups:
            if isinstance(layer, ResidualBlock):
                x = torch.cat([x, skips.pop()], dim=1)
            x = layer(x, time_emb, y)

        x = self.activation(self.out_norm(x))
        x = self.out_conv(x)

        if self.initial_pad != 0:
            return x[:, :, ip:-ip, ip:-ip]
        else:
            return x
```

```
self.time_mlp = nn.Sequential(
            PositionalEmbedding(base_channels, time_emb_scale),
            nn.Linear(base_channels, time_emb_dim),
            nn.SiLU(),
            nn.Linear(time_emb_dim, time_emb_dim),
        ) if time_emb_dim is not None else None
```

class ResidualBlock(nn.Module):

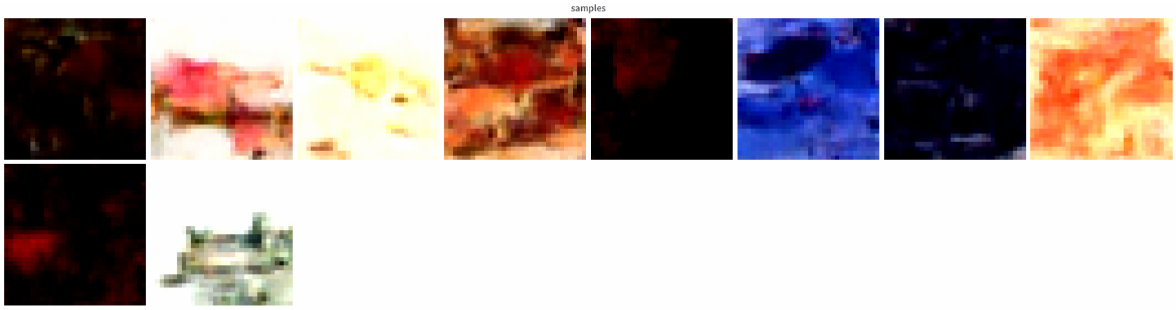
```
def forward(self, x, time_emb=None, y=None):
    out = self.activation(self.norm_1(x))
    out = self.conv_1(out)

    if self.time_bias is not None:
        if time_emb is None:
            raise ValueError("time conditioning was specified but time_emb is
not passed")
        out += self.time_bias(self.activation(time_emb))[:, :, None, None]
        # out 128 128 32 32 time_emb 128 128 1 1

    if self.class_bias is not None:
        if y is None:
            raise ValueError("class conditioning was specified but y is not
passed")

        out += self.class_bias(y)[:, :, None, None]

    out = self.activation(self.norm_2(out))
    out = self.conv_2(out) + self.residual_connection(x)
    out = self.attention(out)
```

```
    return out
```

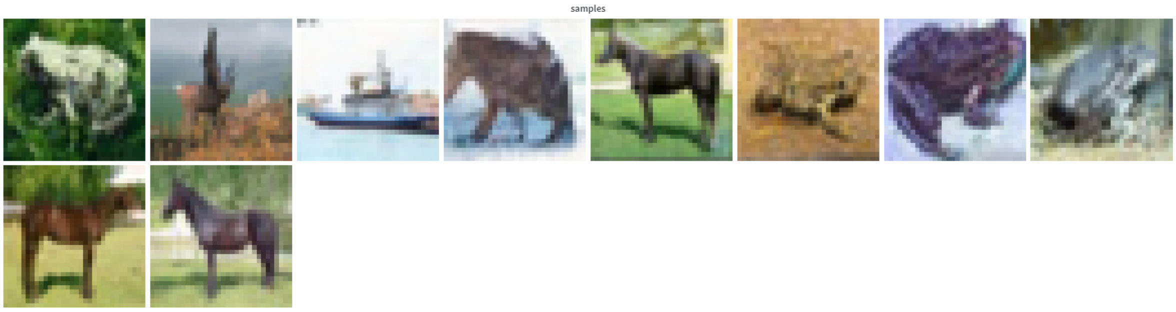
实验效果展示

[训练展示网址]

step 0



step 500



train_loss & test_loss