

Pie-Lab 2025年暑期培训 Practice1

徐浩

Pie-Lab

北京理工大学计算机学院
北京市海淀区中关村南大街
526358612@qq.com

Abstract

本文是“实践III：掌握diffusion model”的实验报告。

1 数学原理

1.1 DDPM整体思想

DDPM是一种生成模型，希望从数据中学到其特征分布，使得生成的数据符合原数据集。其大致过程见图1。

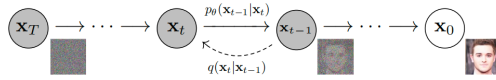


Figure 1: ddpm_overview

从右往左看，对输入的图像 x_0 逐步加噪，最终变成符合标准正态分布的 x_T 。从左往右看，从标准正态分布里面采样，逐步去噪，还原出原始图像。

因此，训练好模型之后，只需要从标准正态分布里面采样，逐步去噪，就能得到一张生成的新图像，它是符合原数据集特征的。

1.2 正向加噪

设 x_{t-1} 加噪得到 x_t 的公式如下，其中 $0 < \alpha_t, \beta_t < 1$ ，加根号是为了待会儿推出的结论和论文的表达一致：

$$\mathbf{x}_t = \sqrt{\alpha_t} \mathbf{x}_{t-1} + \sqrt{\beta_t} \boldsymbol{\varepsilon}_t, \quad \boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

因此，类推得到：

$$\mathbf{x}_t = \sqrt{(\alpha_t \dots \alpha_1)} \mathbf{x}_0 + \sqrt{(\alpha_t \dots \alpha_2) \beta_1} \boldsymbol{\varepsilon}_1 + \sqrt{(\alpha_t \dots \alpha_3) \beta_2} \boldsymbol{\varepsilon}_2 + \dots + \sqrt{\alpha_t \beta_{t-1}} \boldsymbol{\varepsilon}_{t-1} + \sqrt{\beta_t} \boldsymbol{\varepsilon}_t$$

所有含 $\boldsymbol{\varepsilon}$ 的项可以合并为一个均值为 0、方差为系数平方和的正态分布，因此：

$$\mathbf{x}_t = \sqrt{(\alpha_t \dots \alpha_1)} \mathbf{x}_0 + \sqrt{(\alpha_t \dots \alpha_2) \beta_1 + (\alpha_t \dots \alpha_3) \beta_2 + \dots + \alpha_t \beta_{t-1} + \beta_t} \boldsymbol{\varepsilon}_t, \quad \boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

注意，这里的 $\boldsymbol{\varepsilon}_t$ 不是之前 β_t 后的 $\boldsymbol{\varepsilon}_t$ ，而是和 \mathbf{x}_t 相关的量，这么写是为了简化后续的表达。

又因为该系数可以表示为：

$$\begin{aligned}
& (\alpha_t \dots \alpha_1) + (\alpha_t \dots \alpha_2)\beta_1 + (\alpha_t \dots \alpha_3)\beta_2 + \dots + \alpha_t\beta_{t-1} + \beta_t \\
&= (\alpha_t \dots \alpha_2)\alpha_1 + (\alpha_t \dots \alpha_2)\beta_1 + (\alpha_t \dots \alpha_3)\beta_2 + \dots + \alpha_t\beta_{t-1} + \beta_t \\
&= (\alpha_t \dots \alpha_2)(\alpha_1 + \beta_1) + (\alpha_t \dots \alpha_3)\beta_2 + \dots + \alpha_t\beta_{t-1} + \beta_t \\
&= (\alpha_t \dots \alpha_3)(\alpha_2(\alpha_1 + \beta_1) + \beta_2) + \dots + \alpha_t\beta_{t-1} + \beta_t \\
&= \dots \\
&= \alpha_t(\alpha_{t-1}(\dots(\alpha_2(\alpha_1 + \beta_1) + \beta_2) + \dots) + \beta_{t-1}) + \beta_t
\end{aligned}$$

如果约束 $\alpha_t + \beta_t = 1$ ，上式就为 1，再令 $\bar{\alpha}_t = (\alpha_t \dots \alpha_1)$ ，就能得到：

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\varepsilon}_t, \quad \boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad (1)$$

由于加了约束，则 $\alpha_t = 1 - \beta_t$ ，因此最开始 x_{t-1} 加噪得到 x_t 的公式可以写为：

$$\mathbf{x}_t = \sqrt{1 - \beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\varepsilon}_t, \quad \boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

由该公式可知， \mathbf{x}_t 的分布为：

$$\mathbf{x}_t \sim \mathcal{N}(\sqrt{1 - \beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

随着 t 的增加，如果让超参数 β_t 在 $[a, b]$ 线性增加，那么 α_t 就会线性减小，所以 $\bar{\alpha}_t$ 趋近于 0 的速度会越来越快，由式1可知，当 t 足够大， \mathbf{x}_t 就会约等于 $\boldsymbol{\varepsilon}$ ，即服从标准正态分布。

至此，给定原始图像 \mathbf{x}_0 ，可以通过式1一次性加噪得到符合标准正态分布的 \mathbf{x}_T ，而非最初提到的一步一步地加噪。

1.3 反向去噪

反向去噪即：从标准正态分布采样 \mathbf{x}_T ，经过 T 次去噪，得到类似训练集的图像。

反向去噪的公式推导我只看懂了一点点，简述如下：

数学原理表明，当 β_t 足够小时，每一步“加噪声的逆操作”也满足正态分布，即已知 \mathbf{x}_t ，还原出的 \mathbf{x}_{t-1} 应满足：

$$\mathbf{x}_{t-1} \sim \mathcal{N}(\tilde{\mu}_t, \tilde{\beta}_t\mathbf{I})$$

因此需要求 $\tilde{\mu}_t$ 和 $\tilde{\beta}_t$ 。

由贝叶斯公式可得：

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

其中：

$$\begin{aligned}
q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\mu}_t, \tilde{\beta}_t\mathbf{I}) \\
q(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \\
q(\mathbf{x}_t | \mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I}) \\
q(\mathbf{x}_{t-1} | \mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0, (1 - \bar{\alpha}_{t-1})\mathbf{I})
\end{aligned}$$

解得：

$$\begin{aligned}
\tilde{\mu}_t &= \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \varepsilon_t \right) \\
\tilde{\beta}_t &= \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t \\
\mathbf{x}_{t-1} &= \tilde{\mu}_t + \sqrt{\tilde{\beta}_t} \cdot \varepsilon, \quad \varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})
\end{aligned} \tag{2}$$

其中 ε_t 就是式1里的那个，并且上式中只有它是未知量，其他都是常数。

换言之，只需要学习一个网络，它的输入是 \mathbf{x}_t 和 t ，希望它的输出等于 ε_t ，就可以算出 \mathbf{x}_{t-1} 。

因此，一旦训练好这个网络，就可以从 \mathbf{x}_T 一步一步去噪得到生成的新图像 \mathbf{x}_0 。

1.4 训练与采样

下面将论文的算法和上述理论对应起来。

训练的算法如图2，解释如下：

- 从训练集随机选一个图像 \mathbf{x}_0 。
- 随机取一个时刻 t 。
- 生成符合标准正态分布的噪声 ε 。
- 利用 \mathbf{x}_0 和 ε 生成 \mathbf{x}_t （见式1），将 \mathbf{x}_t 和 t 作为神经网络（U-Net等）的输入，输出 ε_θ ，目标是让 ε 和 ε_θ 的平方损失最小。

Algorithm 1 Training

```

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

```

Figure 2: algorithm_train

生成（采样）的算法如图3，解释如下：

- 从标准正态分布采样 \mathbf{x}_T 。
- 利用训练好的网络输出 ε_t ，利用式2反向逐步计算 \mathbf{x}_{t-1} ，其中 σ_t 就是式2中的 $\sqrt{\tilde{\beta}_t}$ 。
- 最终得到生成的图像 \mathbf{x}_0 。

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:    $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$ 
5: end for
6: return  $\mathbf{x}_0$ 

```

Figure 3: algorithm_sample

2 模型结构与代码说明

在这一部分，我用文字结合代码的方式，把“模型结构”和“实现细节与关键代码”一起阐述。

本实验的设置见图4。其中设置 $\beta \in [0.0001, 0.02]$ ， $T = 1000$ ，并尽可能将所有的系数提前算好。

```
# 超参数
PNOISE = 10
GEN_EPOCHS = 5
FID_SAMPLES = 100 # FID评估的样本数量
IMG_SIZE = 32
CHANNELS = 3
DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
T = 1000 # 总步数
BETA_MIN = 1e-4
BETA_MAX = 0.02
BETAS = torch.linspace(BETA_MIN, BETA_MAX, T).to(DEVICE) # 所有下标t都是: [0, T-1]
ALPHAS = 1. - BETAS
ALPHAS_BAR = torch.cumprod(ALPHAS, dim=0) # alpha累积
MODIFIED_ALPHAS_BAR = torch.cat([torch.tensor([1.0], device=DEVICE), ALPHAS_BAR[1:]]) # 去掉ALPHAS_BAR第一个, 前缀1
SQRT_ALPHAS_BAR = torch.sqrt(ALPHAS_BAR) # train
SQRT_ONE_MINUS_ALPHAS_BAR = torch.sqrt(1. - ALPHAS_BAR) # train
SQRT_REVERSE_ALPHAS = torch.sqrt(1. / ALPHAS) # sample
STD = torch.sqrt(BETAS * (1. - MODIFIED_ALPHAS_BAR) / (1. - ALPHAS_BAR)) # sample, 初始值为0. (variance方差, std标准差)
# 固定参数
BATCH_SIZE = 128
L1 = 1e-4
```

Figure 4: code_config

模型训练的实现见图5，首先输入原始图像 x 和步数 t ，通过加噪公式得到 x_t 和 noise ，然后将 x_t 和 noise 作为网络的输入，得到预测的 noise_θ ，利用均方损失计算 noise 和 noise_θ 的差距即可。

```
def train(model, train_loader, optimizer, criterion):
    model.train()
    total_loss, count = 0, 0
    for imgs, _ in train_loader:
        imgs = imgs.to(DEVICE)
        t = torch.randint(0, T, (imgs.size(0),), device=DEVICE).long() # t: [0, T-1]
        x_t, noise = add_noise(imgs, t) # 一批图像, 每个图像的t随机, noise也随机
        predicted_noise = model(x_t, t)
        loss = criterion(predicted_noise, noise)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item() * imgs.size(0)
        count += imgs.size(0)
    return total_loss / count
```

Figure 5: code_train

其中加噪的实现见图6，传入一个批量的 x 和随机生成的 t ，然后生成一个批量的 noise ，利用式1即可得到 x_t ，然后返回 x_t 和 noise ，再用去训练网络。

```
def add_noise(x, t):
    noise = torch.randn_like(x) # 每个x_1的噪声都不同
    sqrt_alphas_bar = SQRT_ALPHAS_BAR[t][:, None, None, None] # 扩展为(BATCH_SIZE, 1, 1, 1)
    sqrt_one_minus_alphas_bar = SQRT_ONE_MINUS_ALPHAS_BAR[t][:, None, None, None]
    return sqrt_alphas_bar * x + sqrt_one_minus_alphas_bar * noise, noise
```

Figure 6: code_add_noise

生成的实现见图7，从 $T-1$ 步到 0 步，利用训练好的网络生成 noise ，通过式2一步一步去噪，最后得到生成的图像。这里需要一个小设计，将 $\text{STD}[0]$ 设置为 0 ，就可以规避算法3最后一步的特殊处理。

```
@torch.no_grad() # 禁用梯度计算
def sample(model, n=16):
    model.eval()
    x = torch.randn(n, CHANNELS, IMG_SIZE, IMG_SIZE).to(DEVICE)
    for t in reversed(range(0, T)): # t: [T-1, ..., 0]
        time_tensor = torch.full((n,), t, device=DEVICE, dtype=torch.long)
        predicted_noise = model(x, time_tensor)
        beta = BETAS[t]
        sqrt_reverse_alpha = SQRT_REVERSE_ALPHAS[t]
        sqrt_one_minus_alpha_bar = SQRT_ONE_MINUS_ALPHAS_BAR[t]
        mean = sqrt_reverse_alpha * (x - beta / sqrt_one_minus_alpha_bar * predicted_noise) # 均值
        noise = torch.randn_like(x)
        x = mean + STD[t] * noise # 设置: STD[0]=0
    x = torch.clamp(x, -1., 1.) # 通过截断, 限制数据在[-1, 1]范围内
    return x
```

Figure 7: code_sample

神经网络的实现见图8，整体结构参考 UNet。时间步 t 首先经过正弦余弦嵌入映射为高维向量，然后经过线性层映射到每个像素的维度，就得到了 t 的嵌入向量，再把它加到每个像素上，就完成了时间步和输入图像的融合。当然这里面有很多的细节需要学习，这里由于时间紧张，没有完全看懂网络结构，重点还是放在 diffusion 的原理上。

```
class UNet(nn.Module):
    """U-Net, 编码器-解码器，中间加连接，跳跃连接上采样，输出噪声预测"""
    def __init__(self, img_channels=3, base_channels=64, time_emb_dim=256):
        super().__init__()
        self.time_emb = nn.Sequential(
            nn.Linear(time_emb_dim, time_emb_dim * 4),
            nn.SiLU(),
            nn.Linear(time_emb_dim * 4, time_emb_dim)
        )
        self.time_emb_dim = time_emb_dim
        self.input_conv = nn.Conv2d(img_channels, base_channels, kernel_size=3, padding=1)
        self.down1 = DownBlock(base_channels, base_channels * 2, time_emb_dim)
        self.down2 = DownBlock(base_channels * 2, base_channels * 4, time_emb_dim)
        self.mid1 = ResidualBlock(base_channels * 4, base_channels * 4, time_emb_dim)
        self.mid2 = ResidualBlock(base_channels * 4, base_channels * 4, time_emb_dim)
        self.up1 = UpBlock(in_channels=base_channels * 4, skip_channels=base_channels * 4, out_channels=base_channels * 2, time_emb_dim=time_emb_dim)
        self.up2 = UpBlock(in_channels=base_channels * 2, skip_channels=base_channels * 2, out_channels=base_channels, time_emb_dim=time_emb_dim)
        self.output_conv = nn.Sequential(
            nn.GroupNorm(8, base_channels),
            nn.SiLU(),
            nn.Conv2d(base_channels, img_channels, kernel_size=3, padding=1)
        )

    def forward(self, x, t):
        t_emb = sinusoidal_embedding(t, self.time_emb_dim)
        x = self.input_conv(x)
        x1, skip1 = self.down1(x, t_emb)
        x2, skip2 = self.down2(x1, t_emb)
        x_mid = self.mid1(x2, t_emb)
        x_mid = self.mid2(x_mid, t_emb)
        x = self.up1(x_mid, skip2, t_emb)
        x = self.up2(x, skip1, t_emb)
        return self.output_conv(x)
```

Figure 8: code_net

3 实验结果与分析

训练损失见图9，大概第 15 个周期开始就训不动了，loss 一直在 0.035 左右。

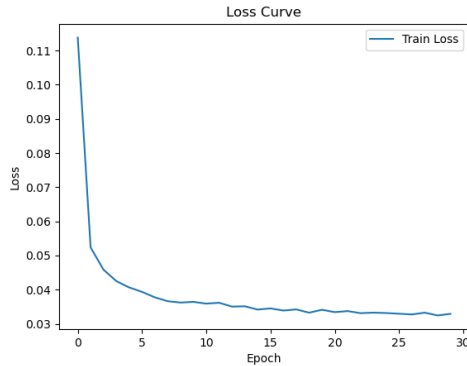


Figure 9: train_loss

评估采用 FID，它通过比较真实数据和生成数据的特征分布（用高斯分布来近似），来衡量二者的相似度，数学原理大致如下。

分别采样真实图像 x 、生成图像 y 若干，然后使用预训练的 Inception 模型 f 将图像转为特征向量 $f(x)$ 、 $f(y)$ 。

分别计算真实图像特征向量、生成图像特征向量的均值和协方差：

$$\begin{aligned}\mu_1 &= \frac{1}{N} \sum_{i=1}^N f(x_i), \\ \mu_2 &= \frac{1}{M} \sum_{i=1}^M f(y_i), \\ \Sigma_1 &= \frac{1}{N-1} \sum_{i=1}^N (f(x_i) - \mu_1)(f(x_i) - \mu_1)^T, \\ \Sigma_2 &= \frac{1}{M-1} \sum_{i=1}^M (f(y_i) - \mu_2)(f(y_i) - \mu_2)^T.\end{aligned}$$

最后计算 FID 分数如下：

$$\text{FID}(X, Y) = \|\mu_1 - \mu_2\|_2^2 + \text{Tr}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1 \Sigma_2)^{1/2})$$

FID 的值越小，说明二者的均值、协方差越接近，也就是特征分布越接近。

概要输出信息如下（完整输出见<https://github.com/xuhao00/pie-lab>）。

其中 FID 值有下降的趋势，说明生成的图像还是有在趋向于原始数据集。但最低也就 205，说明生成效果并不好，和原数据差距较大。

FID score at epoch 5: 257.68

FID score at epoch 10: 214.59

FID score at epoch 15: 246.37

FID score at epoch 20: 211.25

FID score at epoch 25: 205.01

FID score at epoch 30: 228.97

Training time: 19m 45s

生成的图像见10，效果不怎么好，也看不出来是个啥，因此它的 FID 在 200 以上也是理所应当。个人认为，diffusion 算法是固定的、超参数的影响也没这么大，所以肯定是神经网络还不够合适。因此 diffusion 算法更像是一种思路，它通过数学推导给出了可行的训练方案，但需要搭配合适的网络的架构，才能发挥出它的作用。

4 疑惑

本实验存在的疑惑如下：

- **色系** 生成一个批次的图像时，初始采样的 x 都不一样，但为什么最后得到的生成图像都是一个色系的，比如图10，都是蓝色系；比如图11，都是棕色系。个人猜测这是网络的局限性，由于这个神经网络并不合适、训练得不够好，导致它学到的特征分布很单一，比如都往一个色系去输出。
- **预测噪声** 为什么神经网络输入要包含 t ？我查到的资料都是说不同时间步加的噪声程度不同，所以需要传入 t 来告诉网络该噪声属于哪个阶段。但从算法2可以看出，需要被预测的噪声是随机采样出来的，它本身并没有程度之分，也和 t 一点关系都没有。如果硬要说加噪声的程度，那也应该是1.2节第一个公式的超参数 β 来控制的。所以我想知道这个问题能否从数学原理的角度来解释，为什么神经网络输入要包含 t ？为什么理论上给出 x_t 和 t ，就能预测出一个噪声（并且这个噪声还是随机采样出来的）？

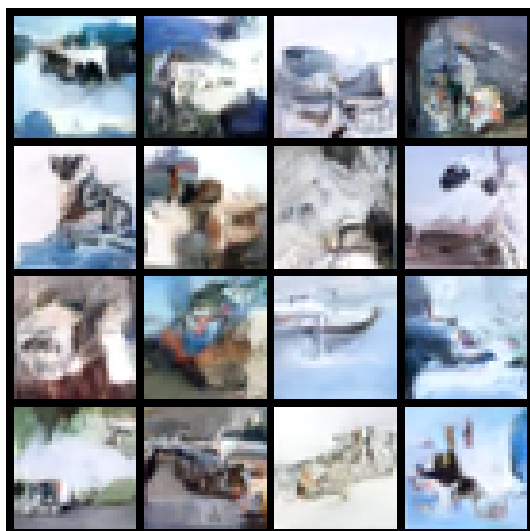


Figure 10: gen_img_e25

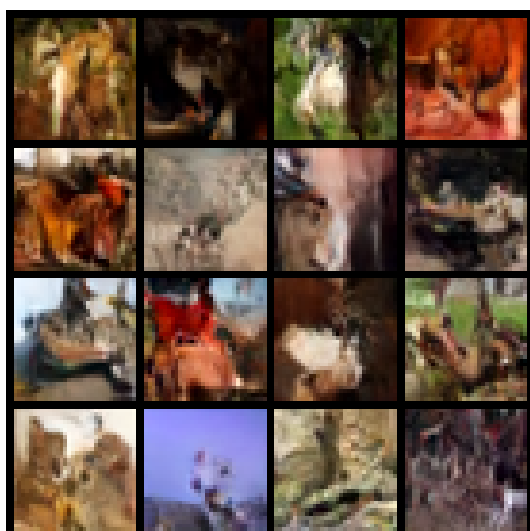


Figure 11: gen_img_e30