

به نام خدا



دانشگاه تهران

دانشکده مهندسی برق و کامپیووتر

## مدل‌های مولد عمیق

تمرین دوم

محمد طاها مجلسی کوپایی	نام و نام خانوادگی
۸۱۰۱۰۱۵۰۴	شماره دانشجویی
۱۴۰۳/۹/۸	تاریخ ارسال گزارش

## فهرست

۴	.....	سوال اول - Diffusion Model
۴	.....	سوال اول زیر بخش اول
۶	.....	سوال اول زیر بخش دوم
۱۰	.....	سوال اول زیر بخش سوم
۱۱	.....	سوال اول زیر بخش چهارم
۱۳	.....	سوال اول زیر بخش پنجم
۱۶	.....	بخش دوم پیاده سازی
۳۷	.....	سوال دوم - Score-based models
۳۷	.....	زیر بخش اول
۳۶	.....	زیر بخش دوم
۳۹	.....	زیر بخش سوم
۴۱	.....	زیر بخش چهارم
۴۳	.....	زیر بخش پنجم
۴۶	.....	بخش دوم - پیاده سازی
۴۶	.....	زیر بخش اول
۴۸	.....	زیر بخش دوم
۵۵	.....	زیر بخش سوم
۶۰	.....	زیر بخش چهارم
۶۲	.....	زیر بخش پنجم
۶۴	.....	مراجع

## سوال اول - DIFFUSION MODEL

اثبات ریاضی عدم نیاز به افزودن نویز اضافی در مدل‌های احتمالی دیفیوژن (DDPM)

در مقاله‌ی (Denoising Diffusion Probabilistic Models) (DDPM)، مطرح شده است که افزودن نویز به طور اضافی در مسیر پیشرفت فرآیند دیفیوژن ضروری نیست و می‌توان در هر مرحله‌ی میانی، بازسازی‌های بین‌میانی را بدون افزودن نویز اضافی به دست آورد. برای اثبات این ادعا از خاصیت توزیع نرمال گاووسی استفاده می‌کنیم که بیان می‌دارد جمع دو متغیر تصادفی مستقل با توزیع نرمال نیز توزیع نرمال خواهد داشت.

این فرضیات اولیه را برای سوال داریم :

فرض کنیم:

$$x_0 \quad \bullet \quad \text{داده‌ی اصلی (بدون نویز) است.}$$

$x_t \quad t = 1, 2, \dots, T$  است که در هر مرحله  $t$ ، مقدار  $x_t$  با فرآیند دیفیوژن یک زنجیره مارکوفی با مراحل  $t$  (یعنی تصویر مرحله‌ی قبلی) به دست می‌آید:

$x_{t-1} \quad \bullet \quad \text{افزودن نویز گاووسی به}$  (یعنی تصویر مرحله‌ی قبلی) به دست می‌آید:

$$\alpha_t = 1 - \beta_t \quad \text{و} \quad \epsilon_t \sim \mathcal{N}(0, I) \quad \text{که در آن} \quad x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1 - \alpha_t} \epsilon_t \quad \bullet \quad \text{است.}$$

حال به این صورت اثبات می‌کنیم که اضافه کردن نویز در مسر رو به جلو ضروری نخواهد بود :

1. نمایش توزیع  $x_t$ :

با بازگشت به عقب، می‌توان نشان داد که :

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon \quad \bullet$$

که در آن:

$$\bar{\alpha}_t = \prod_{s=1}^t \alpha_s \quad \text{و} \quad \epsilon \sim \mathcal{N}(0, I)$$

که بدان معنا است که با اضافه کردن یک مرحله نویز در هر مرحله میتوان از  $x_0$  به  $x_t$  رسید.

#### .2. استقلال نویزها:

میدانیم که نویزهای  $\epsilon_1, \epsilon_2, \dots, \epsilon_t$  در هر مرحله مستقل از یکدیگر هستند. بنابراین، مجموع آنها نیز توزیع نرمال گاوی خواهد داشت: پس این مجموع هم توضیع گاوی دارد:

$$\sum_{s=1}^t \sqrt{\frac{1-\alpha_s}{\bar{\alpha}_t}} \epsilon_s \sim \mathcal{N}\left(0, \sum_{s=1}^t \frac{1-\alpha_s}{\bar{\alpha}_t}\right) = \mathcal{N}(0, I)$$

$\sum_{s=1}^t \frac{1-\alpha_s}{\bar{\alpha}_t} = 1$  ، نتیجه می‌گیریم که نویز تجمعی نیز گاوی است. که از آنجایی که

#### .3. رسیدن به بازسازی‌های میانی:

حال می‌خواهیم نشان دهیم که می‌توانیم به  $x_{t'}$  برای  $t' < t$  برسیم بدون افزودن نویز اضافی. با استفاده از خاصیت توزیع نرمال گاوی، می‌دانیم که:

$$x_t = \sqrt{\alpha_{t'}} x_{t'} + \sqrt{1 - \alpha_{t'}} \epsilon'$$

$$\epsilon' \sim \mathcal{N}(0, I)$$

که در آن  $\epsilon_{t'}$  مستقل از است.

#### .4. استفاده از خاصیت جمع نرمال‌ها:

چون  $x_{t'}$  نیز به صورت ترکیبی خطی از  $x_0$  و نویز گاوی تعریف شده است:

$$x_{t'} = \sqrt{\bar{\alpha}_{t'}} x_0 + \sqrt{1 - \bar{\alpha}_{t'}} \epsilon_{t'}$$

پس:

$$x_t = \sqrt{\alpha_{t'}} (\sqrt{\bar{\alpha}_{t'}} x_0 + \sqrt{1 - \bar{\alpha}_{t'}} \epsilon_{t'}) + \sqrt{1 - \alpha_{t'}} \epsilon'$$

که این بازنویسی نشان می‌دهد که  $x_t$  می‌تواند به صورت ترکیب خطی مناسب از  $x_0$  و نویز جدید تعریف شود بدون

نیاز به افزودن نویز اضافی.

## 5. نتیجه‌گیری:

با توجه به اینکه  $x_{t|x_t}$  به صورت ترکیبی خطی از  $\{x_t\}$  و نویز گاووسی تعریف می‌شود و از خاصیت جمع متغیرهای گاووسی مستقل استفاده کردیم، نتیجه می‌گیریم که می‌توانیم به بازسازی‌های میانی در هر مرحله دست یابیم بدون نیاز به افزودن نویز اضافی به فرآیند دیفیوژن.

## نتیجه‌گیری

با بهره‌گیری از خاصیت جمع متغیرهای تصادفی نرمال گاووسی مستقل، اثبات شد که در مدل‌های احتمالی دیفیوژن (DDPM)، افزودن نویز به طور اضافی در مسیر پیشرفت فرآیند ضروری نیست و امکان دستیابی به بازسازی‌های میانی در هر مرحله بدون نیاز به افزودن نویز جدید وجود دارد. این امر به ساده‌سازی فرآیند و بهبود کارایی مدل کمک می‌کند.

## زیر سوال دوم :

در مدل‌های دیفیوژن، یک فرض کلیدی در مسیر روبه‌عقب (reverse path) آن است که توزیع شرطی گاووسی باشد. اما این فرض تحت چه شرایطی صادق است و چرا با داشتن این توزیع به صورت  $q(x_{t-1} | x_t, x_0)$  گاووسی، مسیر روبه‌عقب عملأً توزیع  $q(x_{t-1} | x_t, x_0)$  را تقریب می‌زند؟

### پیش‌زمینه

1. **فرآیند روبه‌جلو (Forward Process)**: در مدل‌های دیفیوژن، فرآیند پیشروی (forward) با افزودن نویز گاووسی در هر گام تعریف می‌شود. به عبارت دیگر:

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

با تعریف  $\alpha_t = 1 - \beta_t$  و  $\beta_t$  به عنوان واریانس‌های کوچک افزوده شده در هر مرحله.

از آنجا که در هر گام نویزی گاووسی با میانگین خطی از مرحله قبل اضافه می‌شود، می‌توان نشان داد:

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

$$\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$$

که در آن

نتیجه این است که اگرچه  $q(x_t | x_0)$  گاوی است، اما داده‌ی اولیه  $x_0$  الزاماً از توزیع گاوی پیروی نمی‌کند. از این رو  $q(x_0)$  می‌تواند توزیع پیچیده‌ای باشد.

. 2. توزیع روبه‌عقب مشروط به داده‌ی اصلی: اگر  $x_0$  را بشناسیم، آن‌گاه می‌توان توزیع شرطی زیر را به دست آورد:

$$q(x_{t-1} | x_t, x_0)$$

چون  $q(x_t | x_0)$  گاوی است و تبدیل آن بر اساس روابط خطی و افزوده‌شدن نویز گاوی صورت می‌گیرد، توزیع شرطی  $q(x_{t-1} | x_t, x_0)$  نیز به صورت تحلیلی گاوی خواهد بود:

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

این توزیع گاوی را می‌توان از روابط بسته‌ای که در مقاله ارائه شده استخراج کرد.

شرایطی که باعث گاوی بودن  $q(x_{t-1} | x_t)$  می‌شود

حالا به اصل سؤال بپردازیم:

• توزیع  $q(x_{t-1} | x_t)$  در واقع یک توزیع (mixture) روی تمام داده‌های ممکن  $x_0$  است که به این

صورت می‌توانیم بیانش کنیم :

$$q(x_{t-1} | x_t) = \int q(x_{t-1}, x_0 | x_t) dx_0 = \int q(x_{t-1} | x_t, x_0) q(x_0 | x_t) dx_0$$

از آنجایی که  $q(x_0 | x_t)$  خود توزیعی پیچیده (و معمولاً غیرگاوی) از داده‌های اصلی است، این انتگرال

یک ترکیب (mixture) از توزیع‌های گاوی با وزن‌های ناشی از خواهد بود. ترکیب خطی از چند توزیع گاوی با وزن‌های مختلف، الزاماً گاوی خالص نیست. بدین معنا که اگر

$q(x_{t-1} | x_t)$  گاوی نباشد (و معمولاً در مسائل واقعی نیست)، آنگاه به صورت تحلیلی یک توزیع گاوی خالص نخواهد بود؛ بلکه یک مخلوط گاوی پیچیده است.

- زمانی که گاوی باشد، اگر داده‌های اصلی نیز از توزیع گاوی تبعیت کنند، در این صورت

نیز گاوی خواهد بود. زیرا در این حالت ترکیب چندین گاوی با هم (در شرایط خطی بودن تبدیل) همچنان گاوی باقی می‌ماند. اما این فرض در عمل بسیار محدود کننده است و معمولاً داده‌های واقعی توزیع گاوی ساده ندارند.

تقریب مسیر روبه‌عقب با استفاده از  $q(x_{t-1} | x_t, x_0)$ :

با وجود اینکه  $q(x_{t-1} | x_t)$  عملی یک مخلوط پیچیده است، اما ما می‌دانیم:

$$q(x_{t-1} | x_t, x_0)$$

گاوی است و به صورت بسته محاسبه‌پذیر است. ایده‌ی مدل‌های دیفیوژن در مسیر معکوس (Sampling) این است که با

مدلسازی و تقریب  $q(x_{t-1} | x_t, x_0)$ ، که یک توزیع گاوی ساده است، به طور غیرمستقیم

$q(x_{t-1} | x_t)$  را تقریب بزنیم. در عمل چون  $x_0$  ناشناخته است، یک شبکه عصبی پارامتری

آموخته داده می‌شود تا به بهترین شکل ممکن این توزیع را تخمین بزند. از آنجایی که شکل

تحلیلی  $q(x_{t-1} | x_t, x_0)$  معلوم است، می‌توان از آن به عنوان راهنمای (Guidance) برای آموخته مدل معکوس استفاده کرد.

در مدل‌های **Diffusion Probabilistic (DDPM)**، فرضیه این بود که توزیع  $q(x_{t-1} | x_t)$  در مسیر معکوس گویی است. این فرض زمانی معتبر است که:

1. **فرایند انتشار خطی و گویی باشد:** یعنی در هر مرحله انتشار، نویز گویی به داده‌ها اضافه می‌شود. در این صورت، مسیر معکوس نیز به طور طبیعی گویی خواهد بود.

2. **فرضیه مارکوف:** فرض می‌شود که وضعیت  $x_t$  تنها به وضعیت قبلی  $x_{t-1}$  وابسته است و این ساده‌سازی باعث می‌شود که مدل بتواند مسیر معکوس را به صورت گویی مدل کند.

**چرا مسیر معکوس تقریباً  $q(x_{t-1} | x_t)$  را نشان می‌دهد؟**

در DDPM، مسیر معکوس  $q(x_{t-1} | x_t)$  به گویی بودن خود ادامه می‌دهد چرا که مدل یاد می‌گیرد که نویز گویی را از بین ببرد تا به داده‌ی اصلی  $x_0$  برسد. این مسیر معکوس، به طور تقریبی  $q(x_{t-1} | x_t)$  را نشان می‌دهد چرا که:

- فرایند انتشار گویی است، بنابراین مسیر معکوس نیز گویی خواهد بود.
- مدل، به دلیل پیچیدگی محاسباتی، به طور تقریبی از گویی بودن مسیر معکوس استفاده می‌کند تا یادگیری بهینه‌تری داشته باشد.

## دلیل این فرض

این فرض گویی به دلیل ویژگی‌های فرایند انتشار است:

- **نویز گویی در انتشار:** نویز گویی به داده‌ها اضافه می‌شود که این باعث می‌شود که مدل معکوس به راحتی قابل مدل‌سازی باشد.
- **کارایی محاسباتی:** فرض گویی باعث ساده‌تر شدن محاسبات می‌شود و مدل می‌تواند به راحتی این مسیر را آموزش دهد.

در نهایت، فرض گویی به مدل کمک می‌کند که به طور مؤثری مسیر معکوس را یاد بگیرد و بهینه‌سازی را انجام دهد.

## زیر سوال سوم :

در مدل‌های دیفیوژن احتمالاتی حذف نویز (DDPM)، تابع هزینه (Objective Function) از کران پایین تنوعی (VLB) روى منفی لگاریتم درستنمایی داده مشتق می‌شود. این تابع هزینه را می‌توان به سه بخش اصلی تفکیک کرد:

1. واگرایی کولبک-لایبلر بین توزیع پیشین و توزیع پسین در گام نهایی (LTL-T):

○ مفهوم:

این ترم اختلاف بین توزیع گاووسی استاندارد انتخاب می‌شود) و  $p(x_T)$  (که معمولاً یک توزیع گاووسی استاندارد انتخاب می‌شود) و

توزیع داده پس از طی فرآیند پیشرو  $q(x_T | x_0)$  را اندازه‌گیری می‌کند. هدف این ترم اطمینان از آن است که در انتهای فرآیند دیفیوژن (در زمان T)، نویز انتهایی با توزیع پیشین انتخابی (معمولاً

$\mathcal{N}(0, I)$ ) منطبق باشد. به بیان دیگر، این بخش تضمین می‌کند که پس از طی مراحل افزودن نویز، توزیع حاصل تقریباً همان توزیع نویز اولیه باشد، به طوری که مدل بتواند از این نقطه برای تولید نمونه‌های جدید استفاده کند.

2. مجموع واگرایی‌های کولبک-لایبلر در گام‌های میانی : (sum of KL divergences between intermediate steps)

○ مفهوم:

این بخش اصلی‌ترین سهم را در تابع هزینه دارد. این ترم مجموع واگرایی kl divergence بین توزیع

معکوس مدل  $p_\theta(x_{t-1} | x_t, x_0)$  و توزیع پسین واقعی  $q(x_{t-1} | x_t, x_0)$  را در تمامی

گام‌های میانی در بر می‌گیرد. از آنجا که  $q(x_{t-1} | x_t, x_0)$  تحلیلی و گاووسی است، مدل یاد می‌گیرد که وارون‌سازی هر مرحله دیفیوژن را به‌طور دقیق بیاموزد. بنابراین، این ترم مدل را هدایت می‌کند تا در بازگرداندن نویز افزوده شده در هر گام توانا شود و در نهایت بتواند از یک نویز تصادفی به سمت تصویری با کیفیت حرکت کند.

3. لگاریتم منفی درستنمایی داده با توجه به متغیر پنهان اول (L0):

○ مفهوم:

این بخش شبیه به ترم بازسازی در مدل‌های تنوعی (مثلاً VAE‌ها) است. در اینجا پس از رسیدن به  $x_0$ ، مدل باید داده‌ی اولیه  $x_0$  را بازسازی کند. این ترم، خطای بازسازی را کمینه می‌کند و تضمین می‌کند که نمونه‌های نهایی مدل به داده‌های حقیقی نزدیک باشند. هدف از این ترم آن است که مدل تنها به بازگشت از نویز به یک تصویر مشابه داده‌ی اصلی بستنده نکند، بلکه بتواند داده را با کیفیت بالا بازتولید نماید.

---

چرا برخی ترم‌ها در عمل در فرایند بهینه‌سازی استفاده نمی‌شوند؟

در مقاله‌ی DDPM، نویسنده‌گان یکتابع هزینه‌ی ساده‌شده (که اغلب با  $L_{\text{simple}}$  نشان داده می‌شود) معرفی می‌کنند. در این تابع هزینه‌ی ساده‌شده:

- ترم LT: این ترم معمولاً مقدار بسیار کمی دارد؛ چرا که با انتخاب  $p(x_T) = \mathcal{N}(0, I)$  و تعداد گام‌های کافی در

فرایند دیفیوژن،  $\mathcal{N}(0, I)$  پسیار نزدیک به  $q(x_T | x_0)$  خواهد شد. در نتیجه، LT تأثیر چندانی در بهینه‌سازی ندارد و می‌توان از آن صرف‌نظر کرد.

- ترم L0: اگرچه این ترم برای بازسازی نهایی داده مهم است، اما در نسخه‌ی ساده‌شده تابع هزینه، وزن کمتری می‌گیرد یا به صورت غیرمستقیم لحاظ می‌شود. به جای آن، مدل روی ترم میانی (واگرایی‌ها در گام‌های میانی) تمرکز می‌کند که از منظر تجربی منجر به نتایج بهتری می‌شود. در عمل، پژوهش‌ها نشان داده‌اند که با تمرکز بر بهینه‌سازی بازگشت نویز و دقت در پیش‌بینی آن در هر گام، مدل به‌طور غیرمستقیم قادر به تولید نمونه‌های باکیفیت‌تر می‌گردد.

- ترم اصلی یعنی  $\sum_{t=2}^T L_{t-1}$ : این ترم با پیش‌بینی دقیق نویز ( $\epsilon$ ) در هر گام، ساده‌سازی می‌شود. در عمل، یک شبکه عصبی آموزش داده می‌شود تا نویز افزوده‌شده در هر مرحله را مستقیماً پیش‌بینی کند. این راهبرد، جایگزین محاسبه‌ی کامل KL در هر گام می‌شود و باعث ساده‌سازی محاسبات و بهبود کیفیت نمونه‌های تولیدی می‌گردد.

نتیجه‌گیری:

در عمل، در DDPM تمرکز اصلی بر روی بازگرداندن نویز در گام‌های میانی است (ترم‌های  $L_{t-1}$ ) و ترم‌های LT و L0 یا نادیده گرفته شده یا وزنشان پایین آورده می‌شود. دلیل این امر آن است که:

1. ترم LT تأثیر کمی دارد زیرا مطابقت توزیع نهایی با  $\mathcal{N}(0, I)$  تقریباً تضمین شده است.
2. ساده‌سازی با حذف یا کم‌وزن کردن ترم L0 و تاکید بر ترم‌های میانی به لحاظ تجربی به بهبود کیفیت نمونه و تسهیل آموزش منجر می‌شود.

به این ترتیب، با تأکید بر ترم‌های میانی و حذف یا تضعیف دیگر ترم‌ها، DDPM‌ها قادرند به شیوه‌ای ساده‌تر و مؤثرتر آموزش دیده و تصاویر باکیفیت‌تری تولید کنند.

## زیر سوال چهارم :

### توضیح شرایط یکسان شدن DDPM با DDIM

مدل **DDIM** (مدل‌های دیفیوژن ضمنی) یک تعمیم از مدل **DDPM** (مدل‌های احتمالی دیفیوژن حذف نویز) است که ساختار فرآیند معکوس در آن انعطاف‌پذیرتر و غیر مارکوفی تعریف می‌شود. در **DDPM**، فرآیند پیشرو (اضافه کردن نویز) و فرآیند معکوس (حذف نویز) به صورت یک زنجیره مارکوف طراحی شده و به طور مشخص نویز گوسی با یک برنامه‌ریزی ثابت (schedule) در هر گام افزوده یا حذف می‌شود.

مدل **DDIM** این فرآیند را به گونه‌ای تعمیم می‌دهد که می‌توان گام‌های نویزدهی و بازسازی را به صورت غیر مارکوفی انجام داد و یک پارامتر  $\eta$  در فرآیند معکوس معرفی می‌کند که مقدار نویز بازافزوده شده در هر گام را کنترل می‌کند. با تغییر مقدار  $\eta$ ، DDIM می‌تواند بین یک فرآیند معکوس کاملاً تصادفی (مانند DDPM) و یک فرآیند معکوس کاملاً **قطعی** (Deterministic) جابه‌جا شود.

### شرایط یکسان شدن DDPM و DDIM

زمانی که مقدار پارامتر  $\eta=1$  تنظیم شود، مدل **DDIM** دقیقاً به مدل **DDPM** تبدیل می‌شود. در این حالت:

1. **فرآیند معکوس** در DDIM همان فرآیند مارکوفی در DDPM خواهد بود.
2. **مقدار نویز افزوده شده** در هر گام با همان واریانس ثابت تعریف شده در DDPM مطابقت خواهد داشت.
3. به دلیل تنظیم  $\eta=1$ ، ساختار غیر مارکوفی DDIM حذف می‌شود و همان گام‌های نویززدایی در DDPM بازسازی می‌شود.

## توضیح ریاضی

### فرآیند پیشرو در DDPM

در مدل DDPM، فرآیند پیشرو با اضافه کردن تدریجی نویز گوسی به داده اولیه  $x_0$  تعریف می‌شود. در نتیجه، پس از مرحله، می‌توان  $x_t$  را به صورت زیر نوشت:

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

$$\alpha_s = 1 - \beta_s \quad \bar{\alpha}_t = \prod_{s=1}^t \alpha_s \quad \text{که در آن}$$

### فرآیند معکوس در DDPM

مدل DDPM تلاش می‌کند فرآیند افزودن نویز را معکوس کند. این فرآیند معکوس به صورت یک توزیع گاووسی تعریف می‌شود:

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

که در آن  $\mu_\theta$  میانگین و  $\sigma_t$  واریانس مرحله‌ی معکوس هستند.

در عمل، گام معکوس برای بازسازی  $x_{t-1}$  از  $x_t$  به صورت زیر محاسبه می‌شود:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sqrt{1 - \alpha_t} z,$$

$$z \sim \mathcal{N}(0, I) \quad \text{که در آن}$$

### فرآیند معکوس در DDIM

در مدل DDIM، فرآیند معکوس تعمیم داده می‌شود و نویز بازافزوده شده در هر گام با پارامتری به نام  $\eta\backslash\text{eta}$  کنترل می‌شود. گام معکوس در DDIM به شکل زیر تعریف می‌شود:

$$x_{t-1} = \underbrace{\sqrt{\bar{\alpha}_{t-1}} \left( \frac{x_t - \sqrt{1 - \bar{\alpha}_t} \epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}} \right)}_{\text{instead of } x_0} + \sqrt{1 - \bar{\alpha}_{t-1}} \epsilon_\theta(x_t, t) + \eta \sigma_t z,$$

که در آن  $\sigma_t$  واریانس وابسته به نویز بوده و  $\eta$  مقدار نویز بازافزوده شده را تعیین می‌کند.

---

### حالت خاص 1 : $\eta=1$

زمانی که  $\eta=1$  تنظیم شود:

- ترم نویز  $\eta \sigma_t z$  دقیقاً معادل نویز افزوده شده در DDPM می‌شود.
- ساختار غیرمارکوفی DDIM از بین می‌رود و فرآیند معکوس دقیقاً مانند DDPM عمل می‌کند.
- در این شرایط، فرمول DDPM به فرمول DDIM تبدیل می‌شود و مدل‌ها یکسان خواهند شد.

### زیر سوال پنجم :

در مقاله‌ی ES-DDPM (Early-Stopped DDPM)، هدف آن است که سرعت نمونه‌گیری (Sampling) از مدل‌های دیفیوژن (DDPM) افزایش یابد. یادآوری می‌شود که مشکل اصلی در DDPM‌ها این است که برای رسیدن از نویز گاووسی انتهاهی به داده‌ی باکیفیت، باید به صورت تدریجی و مرحله‌به مرحله عمل حذف نویز صورت گیرد. این فرآیند معمولاً شامل صدها تا هزاران گام است و در نتیجه نمونه‌گیری بسیار زمان ببر می‌شود.

**رویکرد مقاله‌ی ES-DDPM** بر این اساس استوار است که فرآیند دیفیوژن را زودهنگام متوقف (Early Stop) کنیم. در DDPM معمولی، داده‌ی اصلی  $x_0$  را در طی  $T$  مرحله‌ی متوالی با افزودن نویز گاووسی به یک توزیع تقریباً گاووسی تبدیل

منکنیم و سپس در مسیر معکوس (Backward Path) از نویز گاووسی تا داده‌ی تمیز، هر بار با حذف نویز، به  $x_0$  مرسیم. تعداد این گام‌ها معمولاً بسیار زیاد است.

اما در ES-DDPM این ایده مطرح می‌شود که **لزومی ندارد تا انتهای مسیر نویزی کردن (Forward Process)** برویم. به عبارت دیگر، به جای آنکه داده را تا یک نویز گاووسی کامل پیش ببریم، در یک گام میانی  $T_0$  (که بسیار کمتر از  $T$  است) فرآیند نویزی کردن را متوقف می‌کنیم. در این مرحله، توزیع ما هنوز کاملاً گاووسی نشده و یک توزیع «غیرگاووسی» است. حال اگر مستقیماً بخواهیم از این توزیع غیرگاووس نمونه بگیریم، مشکل خواهیم داشت چون این توزیع فرم بسته و ساده‌ای ندارد.

**راه حل مقاله** این است که از یک مدل مولد (Generative Model) دیگری مثل یک **VAE** یا **GAN** یا از پیش آموزش دیده استفاده کنیم. این مدل مولد می‌تواند تقریب خوبی از توزیع داده‌های اصلی ارائه دهد. حال اگر ما نمونه‌ای از این مدل مولد بگیریم (تصویر تولید شده توسط VAE یا GAN)، می‌توانیم آن را به صورت یک «نسخه‌ی نویزدار شده در سطح  $T_0$ » در نظر بگیریم. سپس، کافی است از مدل DDPM که فقط تا مرحله‌ی  $T_0$  آموزش داده‌ایم (نه تا  $T$ ) استفاده کنیم تا این نمونه‌ی غیرگاووسی را در تنها  $T_0$  گام معکوس به داده‌ی تمیز تبدیل کنیم.

به عبارت دیگر، فرآیند به دو بخش تقسیم می‌شود:

1. **بخش اول (مولد اولیه):** نمونه‌گیری از یک مدل مولد ساده‌تر (مثل VAE یا GAN) که سریع و ارزان است و به

شما یک تصویر نسبتاً با کیفیت می‌دهد، اما احتمالاً تنوع یا کیفیت حداکثری را ندارد.

2. **بخش دوم (ES-DDPM):** تصویر به دست آمده را با مقداری نویز مطابق با مرحله‌ی  $T_0$  هم‌ردیف می‌کنیم

(یعنی از رابطه‌ی نویزی کردن داده‌ها با واریانس خاص  $\sigma_t$  استفاده کرده تا همان سطح از نویز را اعمال کنیم). اکنون

باید از این تصویر نویزدار که دیگر توزیعش «نژدیک به داده ولی غیر گاووسی» است، با یک DDPM آموزش دیده که

فقط روی همین تعداد گام کمتر ( $T_0$  به جای  $T$ ) متمرکز شده، نمونه‌گیری معکوس کنیم. با این کار، تعداد گام‌های

لازم برای پاکسازی (Denoising) بسیار کاهش می‌یابد.

**مزایای این روش:**

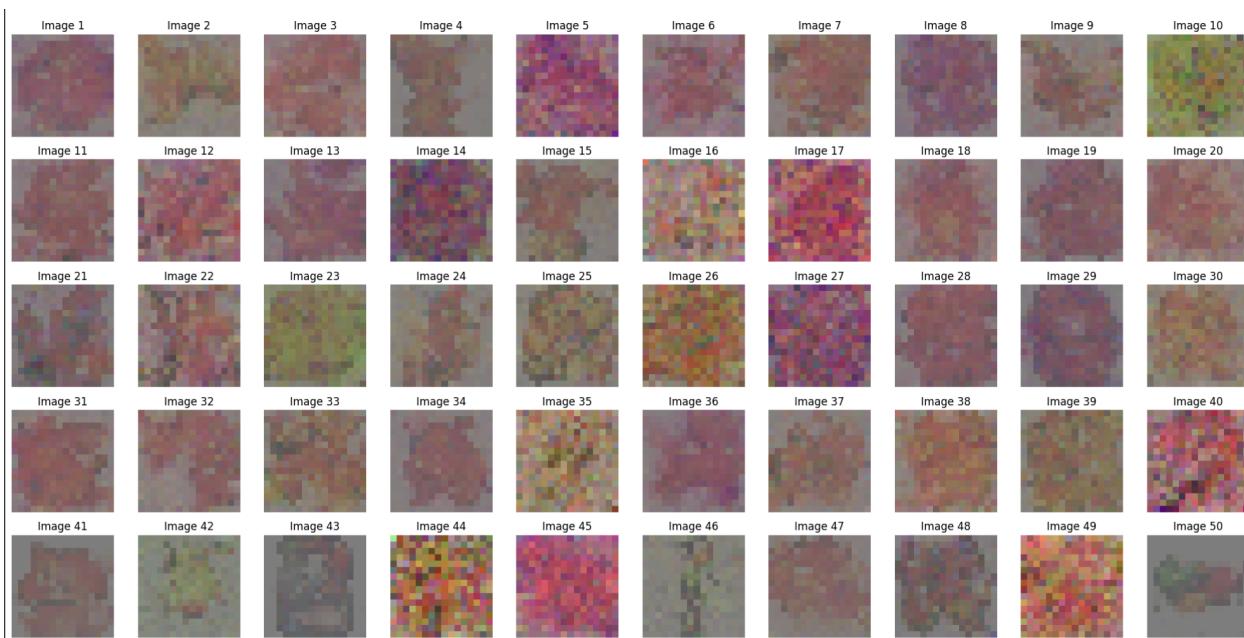
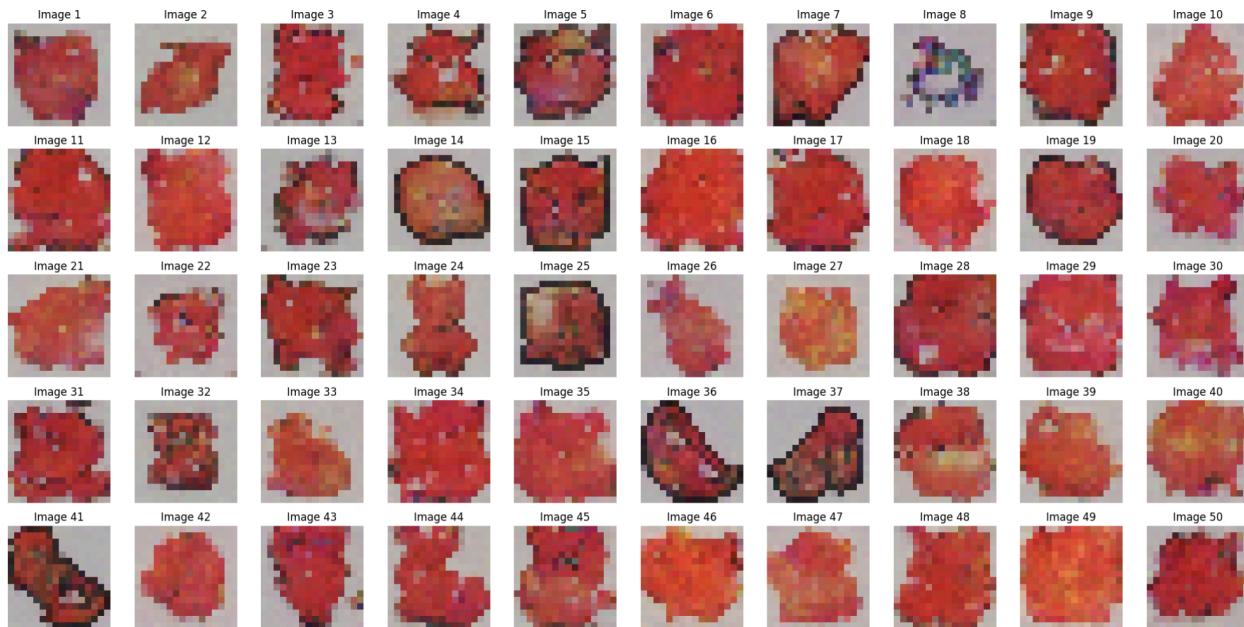
- کاهش چشمگیر تعداد گام‌های حذف نویز (به جای 1000 گام، ممکن است تنها 100 گام نیاز باشد).
  - بهبود کیفیت نمونه‌ها: جالب است که این روش نه تنها سرعت را افزایش می‌دهد، بلکه چون از توان مدل GAN یا VAE بهره می‌برد، کیفیت نهایی نمونه‌ها می‌تواند حتی از DDPM اصلی بهتر باشد.
  - مدل ES-DDPM همچنان می‌تواند از مزایای سایر روش‌های تسريع DDPM استفاده کند. برای مثال، می‌توانید مدل ES-DDPM روش‌هایی که «جهش در گام‌های حذف نویز» دارند را با ES-DDPM ترکیب کنید و زمان نمونه‌گیری را باز هم کاهش دهید.
  - این روش نیازی به ترکیب مدل‌ها از ابتدا ندارد. شما می‌توانید از یک DDPM از پیش آموزش دیده و یک مدل VAE یا GAN از پیش آموزش دیده استفاده کنید و بدون نیاز به آموزش از صفر، آن‌ها را با هم ترکیب کنید.
- جمع‌بندی:** مقاله‌ی ES-DDPM پیشنهاد می‌کند که به جای اینکه مدل دیفیوژن را تا رسیدن به یک نویز کاملاً گاووسی جلو ببریم (که منجر به هزاران گام در فرآیند بازسازی می‌شود)، زودتر متوقف شویم و از یک مدل مولد قدرتمند مثل GAN یا VAE بهره ببریم تا از توزیع غیرگاووسی نمونه بگیرد. سپس با یک تعداد گام بسیار کمتر در DDPM، تصویر نویزی را به تصویر تمیز تبدیل کنیم. این رویکرد باعث کاهش شدید زمان نمونه‌گیری و در برخی موارد افزایش کیفیت نمونه‌ها می‌شود.

## سوالات پیاده سازی :

### سوال ۱: در این سوال تصاویر مختلف را برای ایپاک های مختلف کشیده ایم

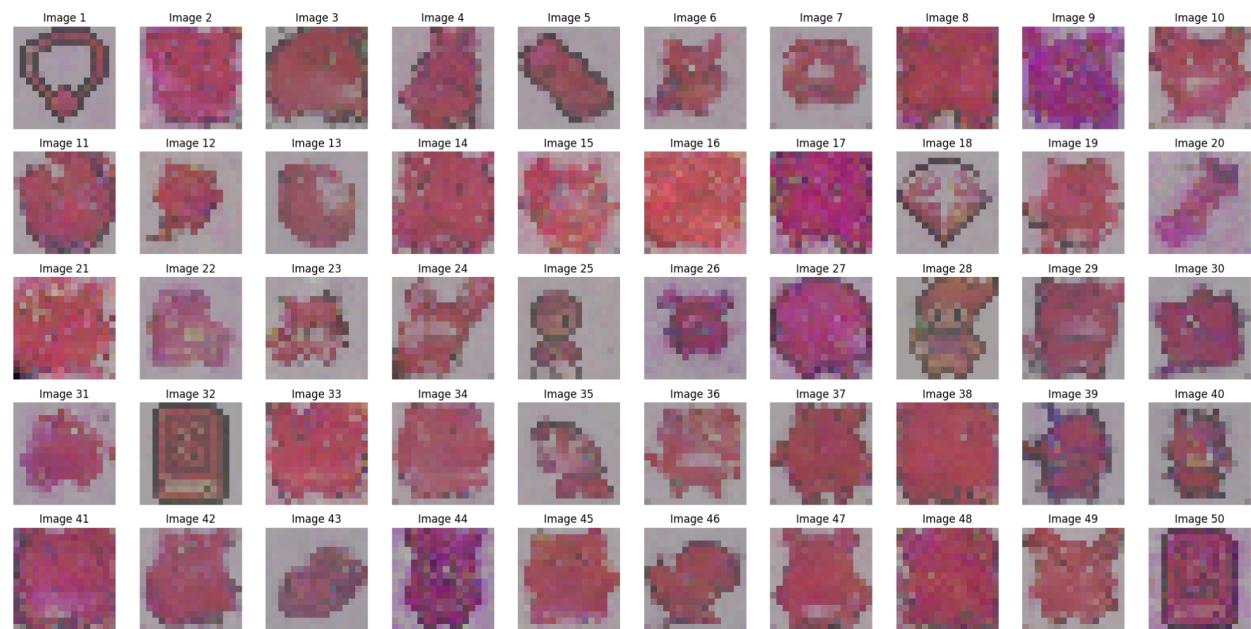
در اینجا چندین عکس از ایپاک های ابتدایی میانی و انتهایی را نمایش میدهیم:

ایپاک دوم :



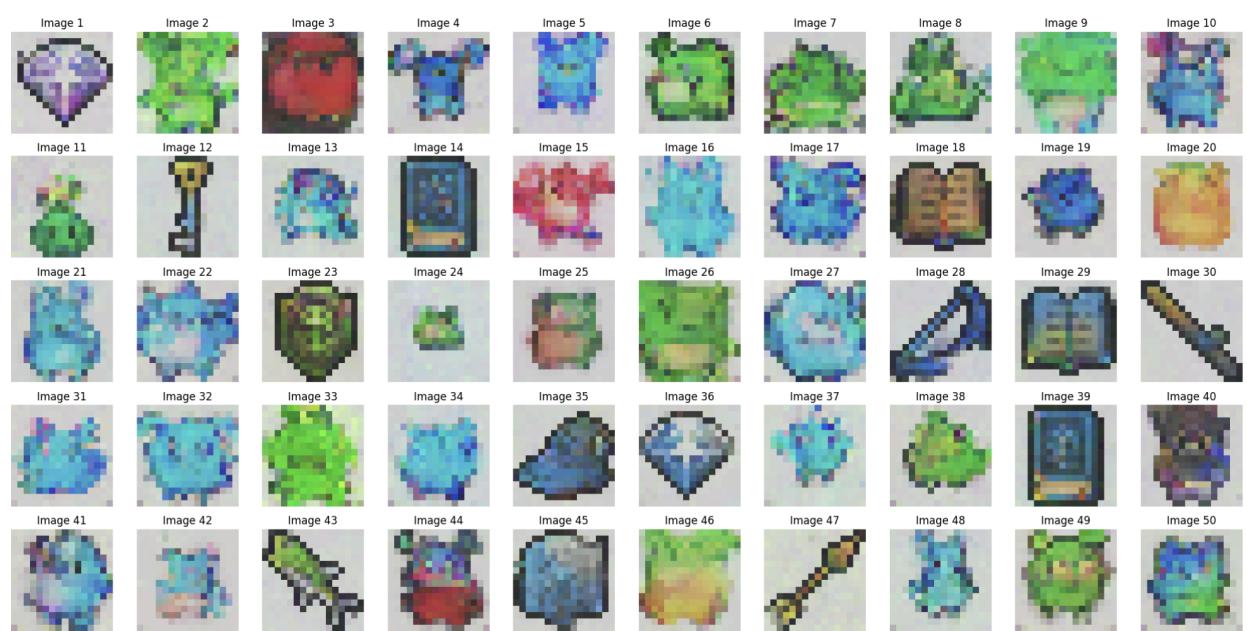
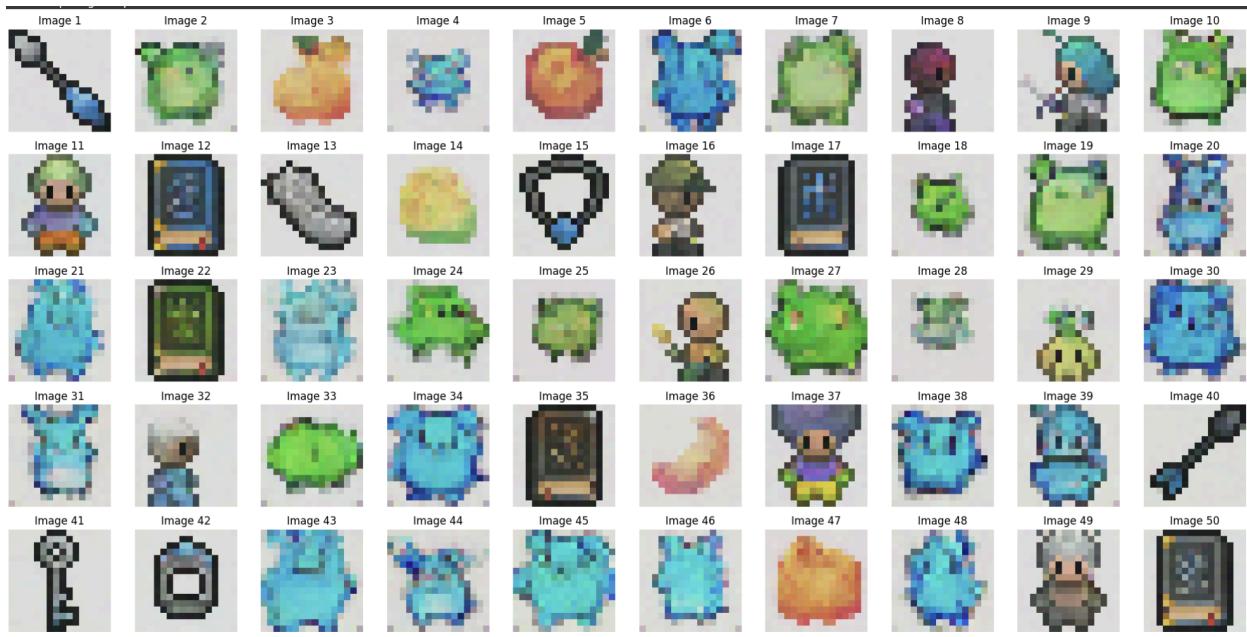
تصویر شماره (۱-۲) : تصاویر تولید شده از ایپاک دوم

ایپاک بیستم :



تصویر شماره (۲-۲) : تصاویر تولید شده از ایپاک بیستم

ایپاک آخر :



تصویر شماره (۳-۲) : تصاویر تولید شده از ایپاک چهلم

که تصاویر اول مربوط به DDPM و تصاویر دوم مربوط به DDIM میباشد .

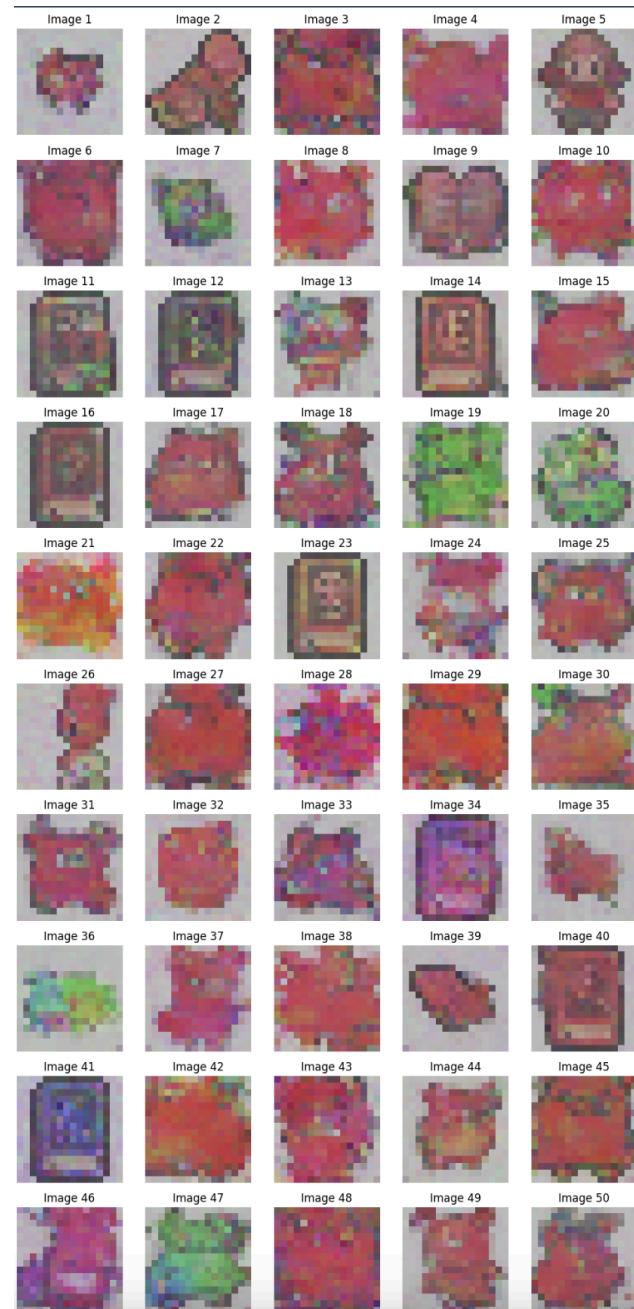
مشاهده میکنیم که تصاویری که با استفاده از روش **DDIM** تولید شده‌اند، با وجود اینکه این روش سرعت تولید را به میزان دو برابر افزایش داده است، دچار افت کیفیت نیز شده‌اند. این افت کیفیت می‌تواند ناشی از کاهش تعداد مراحل نمونه‌گیری و بهینه‌سازی کمتر در طی فرایند تولید باشد. با این حال، کاهش زمان ممکن است در

برخی موارد که نیاز به تولید سریع‌تر است، ارزشمند باشد، هرچند کیفیت نهایی تصاویر تولید شده تحت تأثیر قرار گیرد. در این قسمت سمپل‌های تولید شده از DDPM را داریم:



## تصویر شماره (۴-۲) : تصاویر تولید شده از DDPM

و این قسمت هم مربوط به بخش DDIM میباشد :

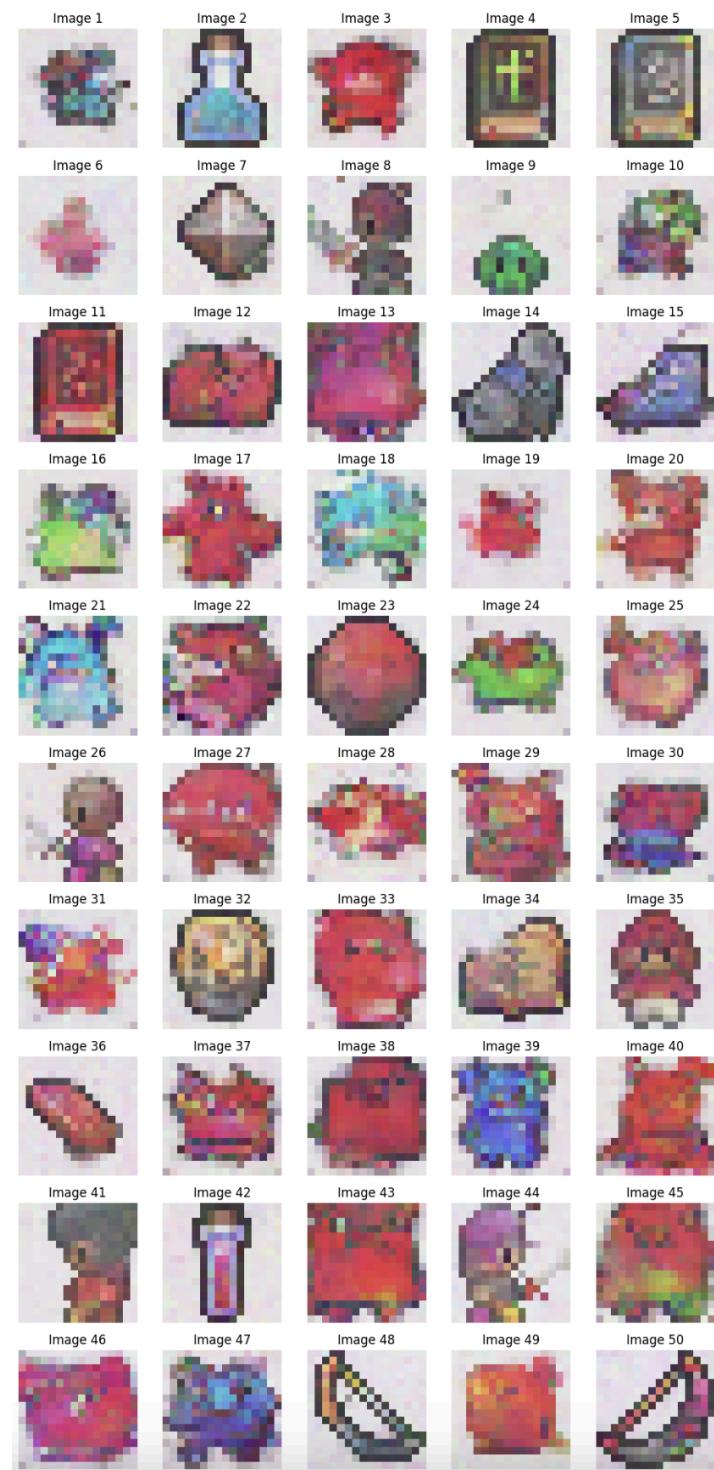


```

final_samples_ddpm = sample_ddpm(nn_model, n_sample=50, timesteps=timesteps, beta=beta, alpha=alpha, alpha_bar=alpha_bar, device=device, height=height).cpu()
final_samples_ddim = sample_ddim(nn_model, n_sample=50, timesteps=timesteps, beta=beta, alpha=alpha, alpha_bar=alpha_bar, device=device, height=height, eta=0.1, n_steps=500).cpu()
plot_sample(final_samples_ddpm, 50, nrows=10)

```

و این تصاویر مربوط به حالت  $\theta$  میباشد که حالت کاملا deterministic میباشد



تصویر شماره (۵-۲) : تصاویر تولید شده از DDIM

```

print("\nGenerating final samples using DDPM and DDIM...")

final_samples_ddpm = sample_ddpm(nn_model, n_sample=50, timesteps=timesteps, beta=beta, alpha=alpha, alpha_bar=alpha_bar, device=device, height=height).cpu()
final_samples_ddim = sample_ddim(nn_model, n_sample=50, timesteps=timesteps, beta=beta, alpha=alpha, alpha_bar=alpha_bar, device=device, height=height, eta= 0, n_steps=400).cpu()

plot_sample(final_samples_ddpm, 50, nrows=10)
plt.savefig(os.path.join(final_ddpm_dir, 'ddpm_final_samples.png'))
plt.close()

plot_sample(final_samples_ddim, 50, nrows=10)

```

## تصویر شماره (۶-۲) : مقادیر اتا و تعداد سمپل های DDIM

مشکلی که در استفاده از مدل DDIM با آن مواجه شدم، این بود که اگر تعداد تکرارها (iterations) کمتر از 300 بود، مدل به طور کلی به نویز کامل می رسید. این اتفاق احتمالاً به دلیل زمان کمی بود که برای آموزش مدل اختصاص داده بودم. وقتی که مدل برای تعداد کمتری از تکرارها آموزش دیده باشد، نتایج تولید شده دچار افت کیفیت و بیشتر شبیه نویز می شود، چرا که مدل هنوز به اندازه کافی برای یادگیری ویژگی ها و جزئیات داده ها آموزش ندیده است. این مشکل معمولاً در فرآیند آموزش مدل های مبتنی بر DDIM که به تعداد زیادی تکرار نیاز دارند تا ویژگی های داده ها را به درستی استخراج کنند، بروز می کند.

در فرآیند آموزش، کاهش مقادیر **Loss** به این شکل نشان می دهد که مدل به تدریج در حال یادگیری و بهبود توانایی خود در بازسازی داده ها و تطبیق با توزیع داده های واقعی است. اگرچه کاهش مقادیر Loss رضایت بخش بوده است، می توان این موضوع را به صورت زیر تحلیل کرد:

## 1. کاهش مقادیر Loss

- کاهش مقادیر Loss نشان می دهد که مدل به تدریج نویز موجود در داده ها را بهتر پیش بینی کرده و قادر است آن را حذف کند. این روند معمولاً بهبود عملکرد مدل را تأیید می کند.
- با این حال، مقدار Loss نهایی می تواند وابسته به زمان آموزش و میزان منابع محاسباتی اختصاص یافته باشد.

## 2. احتمال بهبود با زمان بیشتر

- اگر زمان بیشتری برای آموزش مدل اختصاص داده می شد:

  - مدل می توانست به **مقادیر کمتری از Loss** دست پیدا کند.
  - این کاهش Loss احتمالاً باعث بازسازی تصاویر با کیفیت بالاتر می شد، چرا که مدل زمان بیشتری برای درک بهتر جزئیات داده ها و کاهش خطای پیش بینی نویز داشت.

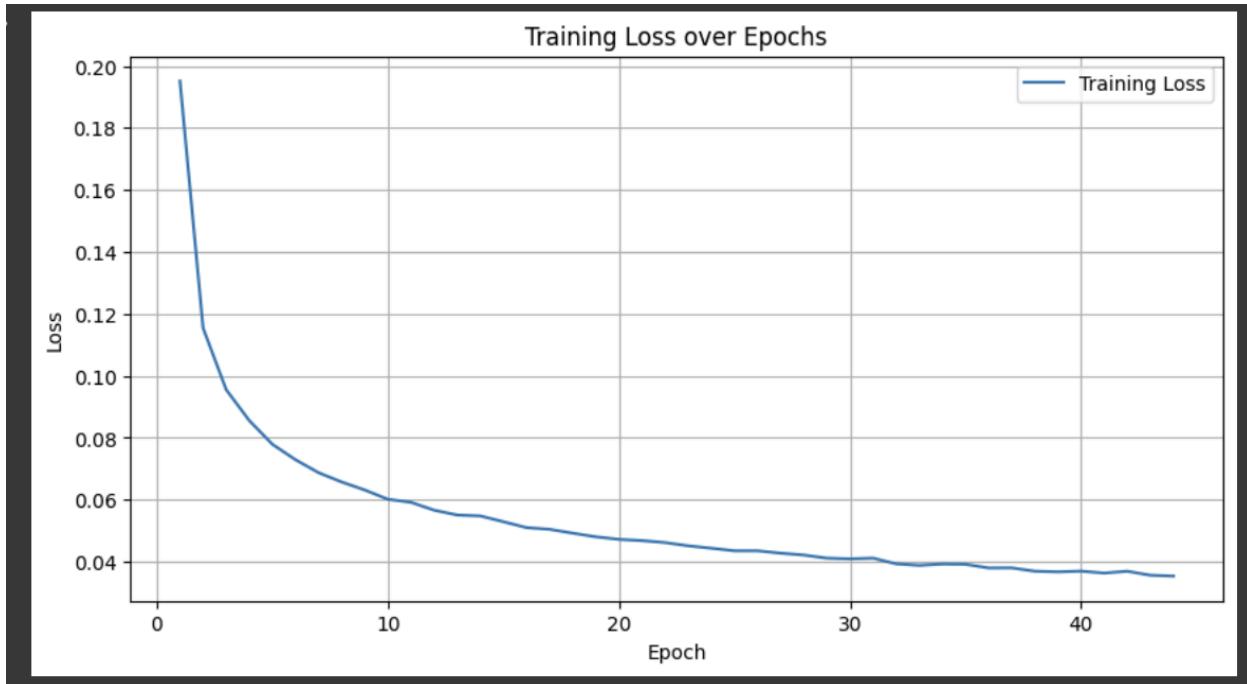
- در واقع، یک مدل با زمان آموزش محدود ممکن است فقط بخش‌های عمومی توزیع داده را یاد بگیرد و نتواند به جزئیات پیچیده‌تر دست یابد.
- 

### 3. کیفیت تصاویر تولیدشده توسط DDPM

- **کیفیت تصاویر DDPM:** با توجه به تصاویر تولیدشده، به نظر می‌رسد که مقدار Loss نهایی برای این میزان از زمان آموزش، مقدار مناسبی بوده است. تصاویر DDPM نشان می‌دهند که مدل به خوبی نویز را حذف کرده و به توزیع داده‌های واقعی نزدیک شده است.
  - **عملکرد مناسب نسبت به شرایط:** این نشان می‌دهد که حتی با زمان محدود برای آموزش، مدل توانسته عملکرد مناسبی ارائه دهد.
- 

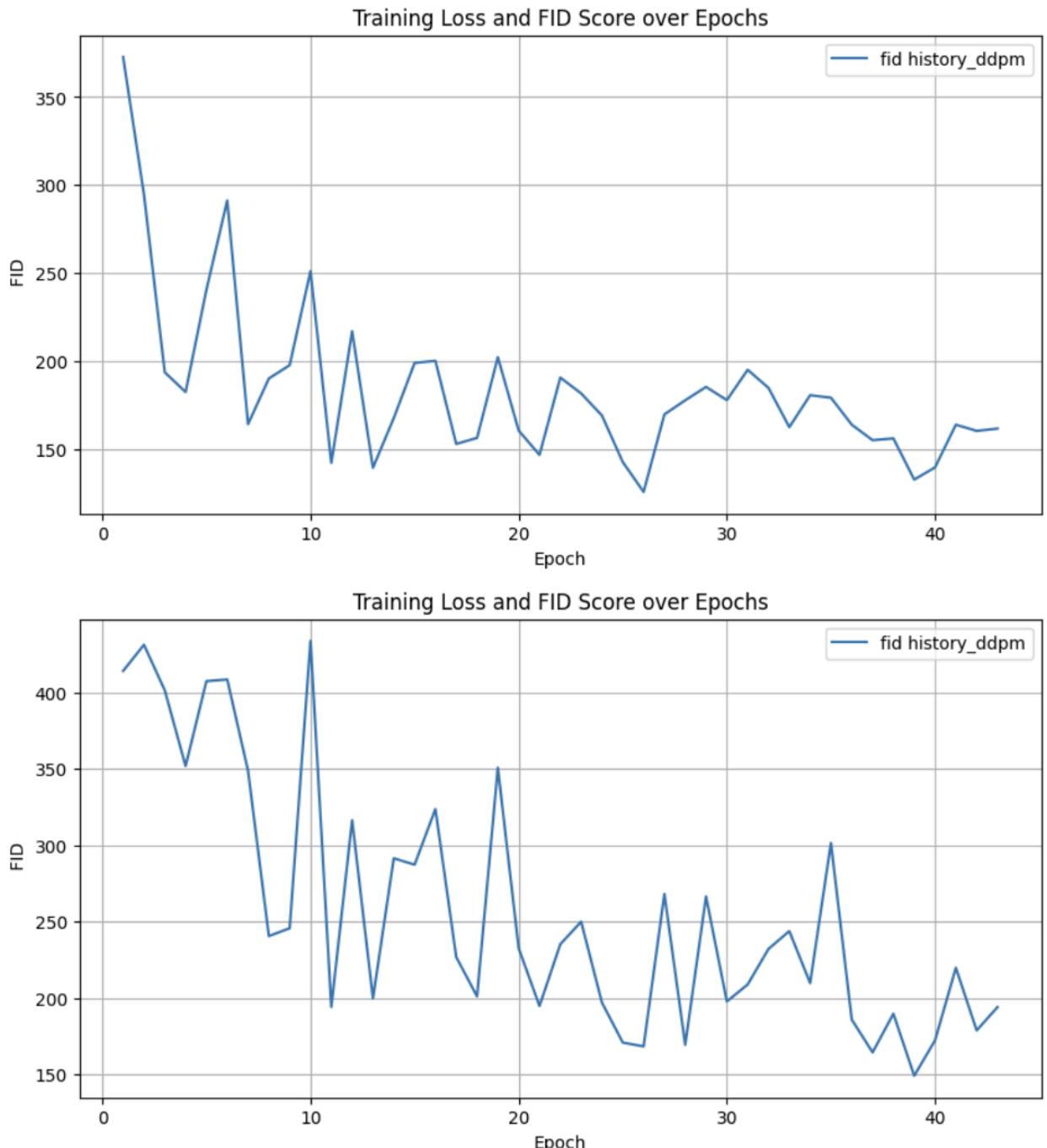
### 4. توازن بین زمان و کیفیت

- **تخصیص زمان بیشتر:** اگرچه افزایش زمان آموزش می‌توانست به کاهش بیشتر Loss منجر شود، باید در نظر داشت که این به هزینه‌های محاسباتی بیشتر و زمان طولانی‌تر نیاز دارد.
  - **کیفیت فعلی قابل قبول:** با توجه به کاهش Loss و تصاویر تولیدشده، به نظر می‌رسد که مدل در شرایط فعلی کیفیت قابل قبولی دارد.
-



تصویر شماره (۷-۲) : نمودار loss DDPM

نمودار های FID هم به این صورت خواهند بود :



تصویر شماره (۷-۲) : مقادیر FID مربوط به DDPM و DDIM

با توجه به اینکه نمودار بالا مربوط به **DDPM** و نمودار پایین مربوط به **DDIM** است، تحلیل را بر اساس ویژگی‌های هر یک از این دو مدل و مقایسه‌ی آن‌ها ارائه می‌کنیم:

---

## 1. تحلیل روند در نمودار DDPM (نمودار بالا):

- شروع آموزش:
  - مقدار اولیه FID در حدود 350 است، که نشان می‌دهد مدل در ابتدای آموزش خروجی‌هایی با کیفیت بسیار پایین تولید می‌کند. این موضوع منطقی است، زیرا مدل هنوز مراحل یادگیری را آغاز نکرده است.
  - روند کاهش:
    - با گذشت زمان، مقدار FID به تدریج کاهش می‌یابد و مدل توانایی بیشتری در تولید تصاویر شبیه به داده‌های واقعی پیدا می‌کند.
    - در میانه‌های آموزش (بین 10 تا 30 دوره)، نوسانات کاهش می‌یابد، و FID به طور پیوسته بهبود می‌یابد.
  - پایداری در انتهای آموزش:
    - در دوره‌های پایانی (حدود 40 به بعد)، مقدار FID به حدود 150 تا 175 می‌رسد و تقریباً پایدار می‌شود. این نشان می‌دهد که مدل توانسته است به خوبی ساختار داده‌های اصلی را یاد بگیرد و تصاویر باکیفیتی تولید کند.

## 2. تحلیل روند در نمودار DDIM (نمودار پایین):

- شروع آموزش:
  - مقدار اولیه FID در حدود 400 است، که از مقدار FID اولیه در DDPM بالاتر است. این نشان‌دهنده این است که در ابتدای آموزش کیفیت تصاویر کمتری نسبت به DDPM دارد. دلیل این موضوع می‌تواند تفاوت در ساختار الگوریتم و کاهش تصادفی‌سازی در DDIM باشد.
- روند کاهش:
  -

- در DDIM نیز به سرعت کاهش می‌باید، اما نوسانات بیشتری در میانه‌های آموزش مشاهده می‌شود
- مثلًاً در دوره‌های 10 تا 30). این نوسانات نشان می‌دهد که DDIM به دلیل ماهیت قطعی (deterministic) خود، در برخی مراحل توانایی تطابق دقیق با داده‌های واقعی را از دست می‌دهد.
- پایداری در انتهای آموزش:

- در دوره‌های پایانی، FID به حدود 150 تا 180 می‌رسد که تقریباً با مقدار نهایی FID در DDPM برابر است.
  - اما همچنان کمی نوسانات باقی مانده است که نشان‌دهنده عملکرد نسبتاً ناپایدارتر DDIM در مقایسه با DDPM است.
- 

### 3. مقایسه بین DDIM و DDPM :

مزایای DDPM (نمودار بالا):

- روند یادگیری پایدارتر:

- DDPM به صورت تدریجی و پیوسته کیفیت تصاویر را بهبود می‌دهد و در انتها به مقادیر FID پایدار و کمنوسانتری دست پیدا می‌کند.
- کیفیت بهتر در مراحل ابتدایی:

- در مراحل اولیه آموزش، DDPM کمتری دارد، که نشان‌دهنده خروجی‌های باکیفیت‌تر در مقایسه با DDIM است.
- مدل احتمالی (Stochastic)

- DDPM به دلیل استفاده از تصادفی‌سازی در مراحل نمونه‌گیری، قادر است تنوع بیشتری در خروجی‌ها ایجاد کند و کیفیت بهتری در مراحل یادگیری اولیه ارائه دهد.
- 

مزایای DDIM (نمودار پایین):

- سرعت بیشتر:
- الگوریتم DDIM تعداد کمتری مرحله در فرآیند بازسازی نیاز دارد، که باعث می‌شود فرآیند نمونه‌گیری بسیار سریع‌تر از DDPM انجام شود.

#### • کیفیت قابل قبول در انتهای:

- با وجود افت کیفیت در مراحل اولیه و نوسانات بیشتر، DDIM در نهایت به FID مشابه با DDPM (حدود 150) می‌رسد.
  - **ماهیت قطعی (Deterministic):** DDIM به جای استفاده از فرآیند تصادفی، از فرآیند قطعی استفاده می‌کند که در کاربردهایی که تنوع کمتر ولی سرعت بیشتر نیاز است، مفید است.
- 

#### 4. جمع‌بندی کلی:

##### • کیفیت تصاویر تولیدی:

- DDPM در طول فرآیند یادگیری عملکرد روان‌تر و پایدارتر دارد و در نهایت به کیفیت بهتری در تصاویر تولیدی دست پیدا می‌کند.
- DDIM هرچند سریع‌تر است، اما در مراحل ابتدایی کیفیت کمتری ارائه می‌دهد و نوسانات بیشتری در طول آموزش دارد.
- **کاربردها:**

- **DDPM:** مناسب برای کاربردهایی که کیفیت خروجی و ثبات در تولید تصاویر اهمیت بیشتری دارد.
  - **DDIM:** مناسب برای کاربردهایی که سرعت نمونه‌گیری اولویت دارد و افت جزئی کیفیت قابل قبول است.
- 

#### نتیجه‌گیری:

اگرچه هر دو مدل در نهایت به مقادیر مشابهی از FID دست پیدا می‌کنند، DDPM با روند یادگیری پایدارتر و کیفیت بالاتر در تصاویر تولیدی، گزینه‌ی بهتری برای کاربردهایی است که نیاز به دقیق و جزئیات بیشتری دارند. از سوی دیگر، DDIM برای مواردی که زمان تولید داده محدود است و سرعت بیشتری نیاز است، می‌تواند مناسب باشد، البته با پذیرش کمی افت کیفیت.

حال تصاویر DDIM را در حالت‌های مختلف اتا میخواهیم ببینیم :

```

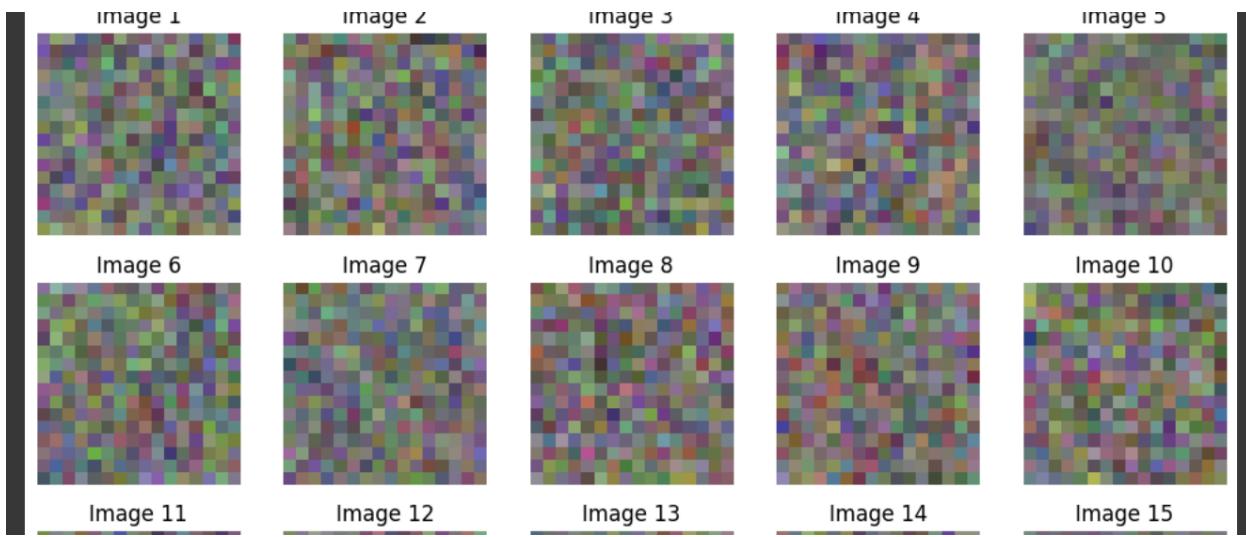
print("\nGenerating final samples using DDPM and DDIM...")

final_samples_ddpm = sample_ddpm(nn_model, n_sample=50, timesteps=timesteps, beta=beta, alpha=alpha, alpha_bar=alpha_bar, device=device, height=height).cpu()
final_samples_ddim = sample_ddim(nn_model, n_sample=50, timesteps=timesteps, beta=beta, alpha=alpha, alpha_bar=alpha_bar, device=device, height=height, eta= 1, n_steps=400).

plot_sample(final_samples_ddpm, 50, nrows=10)
plt.savefig(os.path.join(final_ddpm_dir, 'ddpm_final_samples.png'))
plt.close()

plot_sample(final_samples_ddim, 50, nrows=10)
plt.savefig(os.path.join(final_ddim_dir, 'ddim_final_samples.png'))
plt.close()

```



تصویر شماره (۸-۲) : تصاویر و مقادیر DDIM

```

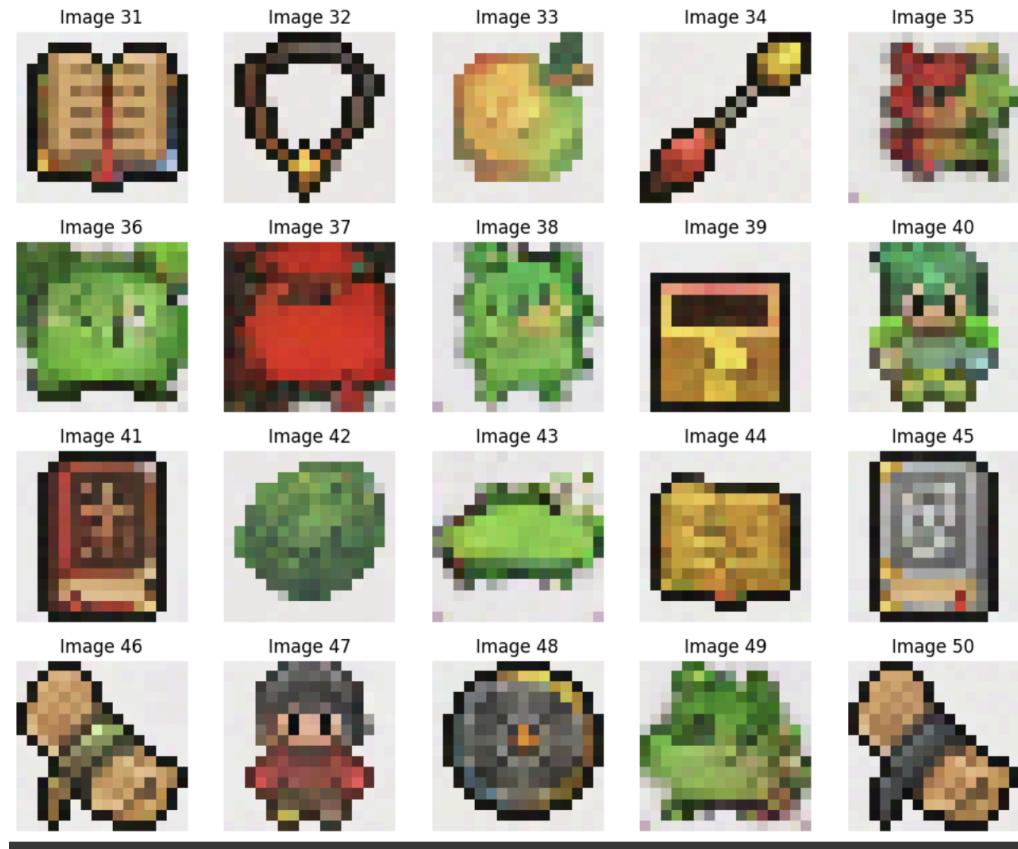
print("\nGenerating final samples using DDPM and DDIM...")

final_samples_ddpm = sample_ddpm(nn_model, n_sample=50, timesteps=timesteps, beta=beta, alpha=alpha, alpha_bar=alpha_bar, device=device, height=height).cpu()
final_samples_ddim = sample_ddim(nn_model, n_sample=50, timesteps=timesteps, beta=beta, alpha=alpha, alpha_bar=alpha_bar, device=device, height=height, eta= 1, n_steps=600).

plt_sample(final_samples_ddpm, 50, nrows=10)
plt.savefig(os.path.join(final_ddpm_dir, 'ddpm_final_samples.png'))
plt.close()

plt_sample(final_samples_ddim, 50, nrows=10)
plt.savefig(os.path.join(final_ddim_dir, 'ddim_final_samples.png'))
plt.close()

```



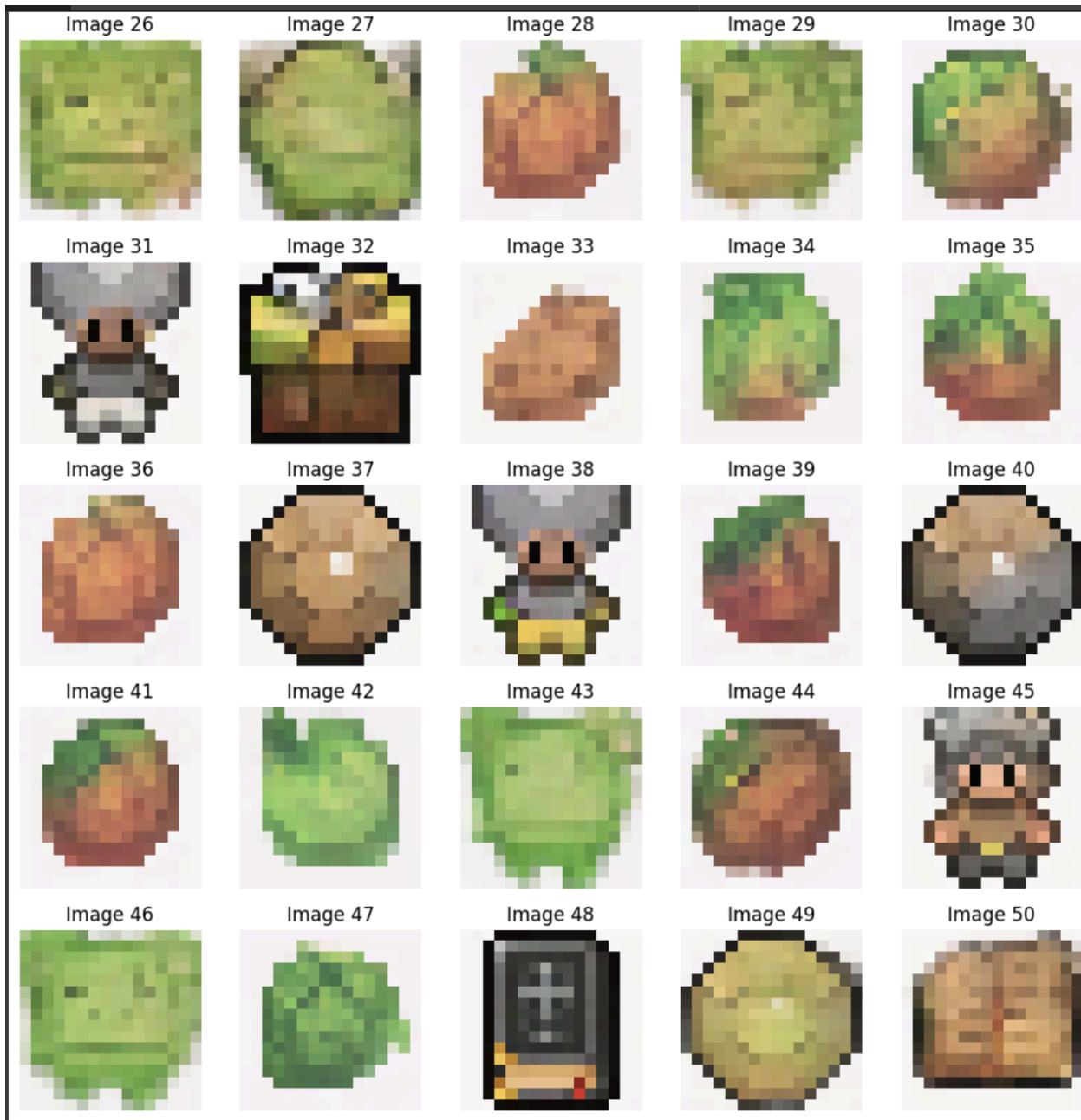
تصویر شماره (۹-۲) : تصاویر و مقادیر DDIM

```
\nGenerating final samples using DDPM and DDIM..."
```

```
amples_ddpm = sample_ddpm(nn_model, n_sample=50, timesteps=timesteps, beta=beta, alpha=alpha, alpha_bar=alpha_bar, device=device, height=height).cpu()
amples_ddim = sample_ddim(nn_model, n_sample=50, timesteps=timesteps, beta=beta, alpha=alpha, alpha_bar=alpha_bar, device=device, height=height, eta= 0.1, n_steps=600).cpu()

mples(final_samples_ddpm, 50, nrows=10)
efig(os.path.join(final_ddpm_dir, 'ddpm_final_samples.png'))
se()

mple(final_samples_ddim, 50, nrows=10)
efig(os.path.join(final_ddim_dir, 'ddim_final_samples.png'))
se()
```



تصویر شماره (۱۰-۲) : تصاویر و مقادیر DDIM

## سوال ۶

این سوال را در ۲ حالت مختلف برای DDIM حل کرده ایم : و جواب هارا با هم مقایسه میکنیم :

```
ce=device, height=height, eta=0.0, n_steps=400
```



```
Summary:  
DDPM Sampling Time: 6.68 seconds  
DDIM Sampling Time: 3.25 seconds
```

```
import time
```

تصویر شماره (۱۱-۲) : تصاویر و مقادیر DDIM و زمان تولید آن ها

```
ce, height=height, eta=1, n_steps=600
```



```
Summary:  
DDPM Sampling Time: 7.09 seconds  
DDIM Sampling Time: 5.61 seconds
```

## تصویر شماره (۱۲-۲) : تصاویر و مقادیر DDIM و زمان تولید آن ها

در حالت اول، زمان تولید تصاویر با **DDIM** نصف شده، اما کیفیت تصاویر به طور محسوسی افت کرده است. علت این افت کیفیت، کاهش زیاد مراحل بازسازی و زمان ناکافی برای یادگیری جزئیات است. این حالت مناسب برای کاربردهایی است که سرعت تولید اولویت اصلی است و افت کیفیت قابل قبول است.

در حالت دوم، زمان تولید تنها ۲ ثانیه کمتر از **DDPM** است و کیفیت تصاویر تقریباً حفظ شده است. این به دلیل تنظیمات مشابه با **DDPM** و کاهش کم مراحل بازسازی است. این حالت تعادل خوبی بین **کیفیت و سرعت** ایجاد می‌کند و برای استفاده عمومی مناسب‌تر است.

## سوال چهارم :

در الگوریتم DDPM Diffusion Probabilistic Models، نویز  $z$  نقش مهمی در فرآیند بازسازی تصویر ایفا می‌کند. در اینجا توضیحات بیشتری در این مورد ارائه می‌شود:

---

### نقش نویز $z$ در فرآیند نمونه‌گیری

در الگوریتم نمونه‌گیری DDPM، هدف این است که از یک توزیع نویز گوسی استاندارد  $(\mathcal{N}(0, I))$  شروع کنیم و به تدریج نویز را کاهش داده و یک تصویر اصلی  $x$  را بازسازی کنیم. نویز  $z$  به فرآیند بازسازی (stochasticity) اضافه می‌شود تا حالت تصادفی (denoising) در فرآیند حفظ شود. این تصادفی‌سازی دو هدف مهم را دنبال می‌کند:

#### 1. حفظ تنوع در خروجی‌ها:

- اگر نویز  $z$  در هر مرحله به فرآیند اضافه نشود، مدل همیشه خروجی‌های یکسانی تولید می‌کند.
- اضافه کردن نویز، مدل را قادر می‌سازد تا از نویز اولیه  $x_T$  (که تصادفی است) به نمونه‌های مختلفی از  $x_0$  برسد.
- این ویژگی به خصوص در تولید داده‌های متنوع، مثل تولید تصاویر مختلف از یک سبک، اهمیت دارد.

#### 2. تطبیق با فرآیند پیش‌رو (Forward Process):

- در فرآیند پیش‌رو (Forward Process)، که داده‌های اصلی با افزودن نویز به تدریج خراب می‌شوند، نویز در هر مرحله اضافه می‌شود. بنابراین، در فرآیند معکوس (Reverse Process)، باید به صورت تقابلی نویز را حذف کنیم و همزمان بخشی از نویز تصادفی را برای شبیه‌سازی دقیق فرآیند حفظ کنیم. این کمک می‌کند مدل، فرآیند بازسازی را بهتر شبیه‌سازی کند.
- 

### چرا نویز $z$ در $t=0$ صفر است؟

در مرحله‌ی  $t=0$ ، آخرین مرحله‌ی الگوریتم نمونه‌گیری است که در آن مدل  $x$  (تصویر بازسازی شده‌ی نهایی) را تولید می‌کند. دلایلی که در این مرحله نویز به صفر تنظیم می‌شود عبارتند از:

#### 1. نیازی به تصادفی‌سازی نیست:

- در مراحل قبلی (زمانی که  $t > 0$  است)، نویز  $z$  به فرآیند اضافه می‌شود تا حالت تصادفی حفظ شود. اما در مرحله‌ی آخر، هدف این است که یک خروجی نهایی ثابت و بدون نویز تولید شود. بنابراین، اضافه کردن نویز در این مرحله منطقی نیست.

#### 2. بازسازی دقیق تصویر:

- در  $t=0$ ، مدل از نویز نهایی به  $x$  تبدیل می‌شود که باید دقیقاً بازسازی تصویر اصلی باشد. اگر در این مرحله نویز اضافه شود، بازسازی دقیق ممکن نخواهد بود و تصویر نهایی دچار خرابی یا نویز خواهد شد.

#### 3. مطابقت با فرآیند معکوس:

- فرآیند معکوس (Reverse Process) بر اساس معادله‌ای طراحی شده که به تدریج نویز را حذف می‌کند. در مرحله‌ی  $t=0$ ، مدل فرض می‌کند که نویز کاملاً حذف شده و تنها تصویر بازسازی شده باقی می‌ماند. اضافه کردن نویز در این مرحله با این فرض در تضاد است.
- 

## فرمول ریاضی نویز $z$ در فرآیند بازسازی

در هر مرحله  $t$  از فرآیند معکوس، نویز  $z$  به شکل زیر در معادله وارد می‌شود:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sigma_t z$$

- زمانی که  $t > 0$

- نویز  $z \sim \mathcal{N}(0, I)$  از یک توزیع گاووسی استاندارد نمونه‌گیری می‌شود.
- این نویز در فرآیند بازسازی نقش دارد و به  $x_{t-1}$  اضافه می‌شود.

• زمانی که  $t=0$ :

- $z=0$  تنظیم می‌شود، زیرا در این مرحله تنها تصویر بازسازی شده‌ی  $x_0$  باید باقی بماند و نیازی به تصادفی‌سازی بیشتر نیست.
- 

## ارتباط نویز $z$ با کیفیت تصاویر تولیدی

مقدار نویز  $Z$  و نحوه‌ی استفاده از آن تاثیر مستقیمی بر کیفیت تصاویر تولیدی دارد:

1. مقدار نویز در مراحل اولیه (زمانی که  $t$  بزرگ است):

- در مراحل اولیه، نویز تأثیر بیشتری دارد زیرا  $xt$  کاملاً نویز است. در اینجا، نویز به فرآیند کمک می‌کند تا مسیر معکوس را به درستی طی کند.
2. کاهش تدریجی نویز:

- با کاهش  $t$ ، مقدار نویز کاهش می‌یابد زیرا مدل به سمت بازسازی تصویر پیش می‌رود.
3. بدون نویز در مرحله نهایی:

- در  $t=0$ ، حذف نویز به مدل اجازه می‌دهد تا یک خروجی پایدار و بدون اختلال تولید کند.
- 

## جمع‌بندی: چرا نویز $z$ مهم است؟

1. نویز  $z$  به فرآیند تصادفی‌سازی و حفظ تنوع در نمونه‌های تولیدی کمک می‌کند.
2. در مراحل ابتدایی، نویز برای تطبیق با فرآیند پیش‌رو ضروری است.
3. در مرحله نهایی، حذف نویز باعث تولید یک تصویر بازسازی شده‌ی پایدار و دقیق می‌شود.

## SCORE-BASED MODELS سؤال دوم

### زیر سوال اول :

در نمایش یک توزیع احتمالی به فرم زیر:

$$p_{\theta}(x) = \frac{e^{-f_{\theta}(x)}}{Z_{\theta}}$$

مشکل اصلی:

بخشی که در این نمایش مشکل‌ساز است، محاسبه و یادگیری ترم نرمال‌کننده یا  $Z\theta$  است. این ترم که به آن ضریب پارامتر نیز می‌گویند، معمولاً به صورت زیر تعریف می‌شود:

$$Z_{\theta} = \int e^{-f_{\theta}(x)} dx.$$

محاسبه‌ی دقیق  $Z_{\theta}$  اغلب بسیار دشوار و هزینه‌بر است، چرا که نیاز به انجام یک انتگرال بر تمامی فضای ورودی دارد. در عمل، برای توزیع‌های پیچیده و پربعد، محاسبه‌ی این انتگرال به شکل تحلیلی ممکن نیست و نیازمند تقریب‌های عددی پیچیده و گران است. این موضوع یادگیری پارامترهای  $\theta$  از طریق ماکزیمم‌سازی درستنمایی یا گرادیان آن را بسیار دشوار می‌کند.

---

راه حل در مدل‌های Score-based

در مدل‌های مبتنی بر اسکُور (Score-based Models) به جای یادگیری خود توزیع  $p_{\theta}(x)$  به صورت صریح، تابع اسکُور آن را یاد می‌گیریم. تابع اسکُور به صورت زیر تعریف می‌شود:

$$s_{\theta}(x) = \nabla_x \log p_{\theta}(x).$$

می‌توان نشان داد که:

$$s_\theta(x) = -\nabla_x f_\theta(x)$$

(با صرف نظر از ترم نرمال‌کننده، زیرا  $\log p_\theta(x) = -f_\theta(x) - \log Z_\theta$  است و  $\nabla_x \log Z_\theta = 0$  وابسته به  $x$  نیست).

به این ترتیب، در مدل‌های Score-based به جای تخمین خود توزیع یا ضریب پارتویشن، مستقیماً شبیه لگاریتم احتمال  $\log p_\theta(x)$  نسبت به  $x$  را یاد می‌گیریم. این کار اجازه می‌دهد بدون نیاز به محاسبه  $Z_\theta$  توزیع را به صورت ضمنی مدل کنیم. در واقع، با داشتن میدان گرادیان  $(s_\theta(x))$  می‌توان از روش‌های مختلف نمونه‌گیری (نظیر شبیه‌سازی معکوس لانژون یا روش‌های MCMC) استفاده کرده و از توزیع ضمنی نمونه کشید.

## زیر سوال دوم :

در مدل‌های مبتنی بر Score-based Models، هدف اصلی یادگیری گرادیان لگاریتم توزیع داده یا مدل، یعنی  $s_\theta(x) = \nabla_x \log p_\theta(x)$  است. روش Score Matching کلاسیک، برای محاسبه تابع هزینه و یادگیری این اسکوچورها به محاسبه هسین (مشتق دوم) نسبت به بردار ورودی نیاز دارد. این محاسبه معمولاً بسیار هزینه‌بر است، زیرا برای هر بعد از ورودی، باید یک بار عملیات تفاضل‌گیری (backpropagation) انجام شود. در داده‌های پربعد یا مدل‌های عمیق، این امر محاسبات را بسیار سنگین و گند می‌کند.

ایده‌ی **Sliced Score Matching** (SSM) این است که به جای محاسبه کامل هسین و جمع‌زن روی تمام ابعاد ورودی،  $\text{tr}(\nabla_x s_\theta(x))$  بردار اسکوچور را بر روی جهت‌های تصادفی پروجکشن می‌کنیم. بدین ترتیب، به جای اینکه لازم باشد ترم  $\text{tr}(\nabla_x s_\theta(x))$  (که همان جمع قطر هسین است) را مستقیماً محاسبه کنیم، از خاصیت انتظاری و بردارهای تصادفی  $v$  بهره می‌بریم:

$$\text{tr}(\nabla_x s_\theta(x)) = \mathbb{E}_v[v^\top (\nabla_x s_\theta(x)) v].$$

با استفاده از این رابطه، نیاز به محاسبه هر یک از المان‌های قطری هسین از بین رفته و در عوض تنها باید عبارت  $v^\top (\nabla_x s_\theta(x)) v$  را محاسبه کنیم. برای یک جهت تصادفی  $v$ ، محاسبه چنین عبارتی با تعداد اندکی عملیات backpropagation انجام می‌شود (به طور معمول دو بار backprop برای هر پروجکشن). از آنجا که می‌توان تعداد پروجکشن‌ها ( $M$ ) را کنترل کرد، با انتخاب مناسب  $M$  می‌توان هزینه‌ی محاسباتی را به میزان زیادی کاهش داد و در عین حال تقریب قابل قبولی از تابع هزینه‌ی Score Matching به دست آورد.

**خلاصه:**

روش Sliced Score Matching با استفاده از بردارهای تصادفی و پروژه کردن اسکور بر روی این جهت‌ها، هزینه‌ی سنگین محاسبه‌ی هسین کامل را کاهش می‌دهد. این امر باعث می‌شود که بتوان مدل‌های عمیق‌تر و داده‌های پربعدتر را با روشی مبتنی بر Score Matching، اما با هزینه‌ی محاسباتی کمتر، آموزش داد.

## زیر سوال سوم :

اضافه کردن نویز به داده‌ها در هنگام آموزش مدل‌های مبتنی بر Score (بهویژه در روش‌های Score Matching) با هدف ساده‌سازی محاسبات و پایدارکردن فرآیند یادگیری اسکورها (گرادیان لگاریتم توزیع) انجام می‌شود. این کار در حقیقت توزیع داده‌ی اصلی را به یک توزیع نویزدار تبدیل می‌کند و سپس مدل را وادار می‌کند تا اسکور این توزیع نویزدار را بیاموزد. با تقریب زدن اسکور توزیع نویزدار، در نهایت می‌توان اسکور توزیع اصلی داده را نیز تقریب زد.

**توضیح ریاضیاتی:**

فرض کنید داده‌های ما طبق توزیع  $p(x)$  نمونه‌برداری می‌شوند. در روش Score Matching کلاسیک، هدف بهینه‌سازی تابع هزینه‌ای است که با حذف جمله ثابت به صورت زیر نوشته می‌شود:

$$L(\theta) = \frac{1}{2} \mathbb{E}_{x \sim p(x)} [\|s_\theta(x) - s_p(x)\|_2^2]$$

که در آن:

$$s_\theta(x) = \nabla_x \log p_\theta(x) \quad .$$

$$s_p(x) = \nabla_x \log p(x) \quad .$$

مشکل اصلی در اینجا آن است که محاسبه‌ی  $s_p(x)$  (یا همارز آن، جایگذاری در فرمول Score Matching کلاسیک) به محاسبه‌ی هسین  $\nabla_x s_\theta(x)$  نیاز دارد که معمولاً پرهزینه است.

**:Denoising Score Matching DSM**

در Denoising Score Matching، ابتدا یک نویز گوسی با واریانس  $\sigma^2$  به داده‌ها اضافه می‌شود. یعنی اگر داده‌ی اصلی  $x$

از  $p(x)$  نمونه‌برداری شده باشد، داده‌ی نویزدار  $\tilde{x}$  به صورت زیر تعریف می‌شود:

$$\tilde{x} \mid x \sim q_\sigma(\tilde{x} \mid x) = \mathcal{N}(x; \tilde{x}, \sigma^2 I).$$

این کار توزیع داده را از  $\tilde{p}(\tilde{x})$  به توزیع نویزدار تبدیل می‌کند، به طوری که:

$$\tilde{p}(\tilde{x}) = \int p(x)q_\sigma(\tilde{x} \mid x)dx.$$

حالا تابع هزینه‌ای که در DSM استفاده می‌شود از طریق انتگرال‌گیری بر روی توزیع نویز دار نوشته می‌شود:

$$L_{\text{DSM}}(\theta) = \frac{1}{2} \mathbb{E}_{x \sim p(x), \tilde{x} \sim q_\sigma(\tilde{x} \mid x)} [\|s_\theta(\tilde{x}) - \nabla_{\tilde{x}} \log q_\sigma(\tilde{x} \mid x)\|_2^2].$$

نکته مهم آن است که  $\nabla_{\tilde{x}} \log q_\sigma(\tilde{x} \mid x)$  را می‌توان به راحتی محاسبه کرد، زیرا:

$$\log q_\sigma(\tilde{x} \mid x) = -\frac{1}{2\sigma^2} \|\tilde{x} - x\|_2^2 + \text{const.}$$

و لذا

$$\nabla_{\tilde{x}} \log q_\sigma(\tilde{x} \mid x) = -\frac{1}{\sigma^2} (\tilde{x} - x).$$

این عبارت بسیار ساده‌تر از محاسبه‌ی هسین و جایگذاری در فرمول اصلی Score Matching است.

رابطه‌ی نویز با پارامترهای مدل:

هنگامی که نویز افزوده می‌شود، مدل  $\theta$  دیگر مستقیماً با اسکُور توزیع اصلی  $p(x)$  سروکار ندارد، بلکه به صورت غیرمستقیم از طریق اسکُور توزیع نویزدار  $\tilde{p}(\tilde{x})$  آن را یاد می‌گیرد. مدل سعی می‌کند اسکُور توزیع نویزدار را تخمین بزند و از آنجا که نویز گاووسی ساده است، این کار بهینه‌سازی را تسهیل و پایدارتر می‌کند. پارامترهای مدل با کمینه‌کردن DSM به سمت تخمینی نزدیک به اسکُور توزیع اصلی حرکت می‌کنند. هرچه  $5$  کوچکتر باشد، توزیع نویزدار به توزیع اصلی نزدیک‌تر است و اسکُور تخمینی نیز دقیق‌تر خواهد بود، ولی انتخاب  $5$  خیلی کوچک ممکن است به مشکلاتی مانند حساسیت زیاد

مدل به نویز منجر شود. از سوی دیگر، ۵ خیلی بزرگ باعث می‌شود مدل یک اسکُور بیش از حد صاف (smooth) بیاموزد که از توزیع اصلی فاصله دارد.

#### مقایسه با DDPM

در مدل‌های DDPM Denoising Diffusion Probabilistic Models نیز ایده مشابهی دنبال می‌شود. آنجا نیز با افزودن نویز مرحله به مرحله و سپس یادگیری معکوس کردن این فرآیند، مدل در نهایت توزیعی نزدیک به داده‌ها را یاد می‌گیرد. ایده‌ی نویز افزایی در DSM و DDPM هر دو بر این اصل استوار است که یادگیری اسکُور یک توزیع نویز دار آسان‌تر و پایدارتر است و سپس با کاهش تدریجی سطح نویز می‌توان به توزیع اصلی داده نزدیک شد.

## زیر سوال چهارم :

پاسخ:

در مدل‌های مبتنی بر Score-based Models، آنچه آموخته می‌شود، تخمینی از گرادیان لگاریتم توزیع داده، یعنی  $s_\theta(x) = \nabla_x \log p_\theta(x)$  است. با داشتن این اسکُور، هدف اصلی این است که بتوان از توزیعی که مدل یاد گرفته

نمونه‌برداری کنیم، یعنی نمونه‌هایی از  $p_\theta(x)$  تولید کنیم. اما برخلاف مدل‌های نرمالیزه‌شده‌ی معمول (که می‌توان مستقیماً از تابع توزیع نمونه کشید)، در اینجا توزیع  $p_\theta(x)$  به طور غیرمستقیم و از طریق اسکُور آن در دسترس است. این بدان معناست که ما نمی‌توانیم مستقیماً از  $p_\theta(x)$  نمونه بگیریم، بلکه باید فرایندی را طی کنیم که با استفاده از اسکُور، نمونه‌ها را به تدریج به سمت نواحی با احتمال بالاتر هدایت کند.

#### ایده‌ی Langevin Dynamics Sampling

Langevin Dynamics یک فرایند نمونه‌برداری است که از روش‌های زنجیره مارکوف مونت کارلو (MCMC) محسوب می‌شود. ایده‌ی اصلی Langevin Dynamics این است که اگر  $x_t$  را حالت (سمپل) فعلی بدانیم، با استفاده از گرادیان لگاریتم توزیع (اسکُور) و یک نویز گاوی، نمونه‌های جدید  $x_{t+1}$  را تولید کنیم. به بیان ریاضی، به روزرسانی به این صورت است:

$$x_{t+1} = x_t + \frac{\epsilon^2}{2} \nabla_x \log p_\theta(x_t) + \epsilon z_t$$

$z_t \sim \mathcal{N}(0, I)$  که در آن نویز گوسی مستقل است و  $\epsilon$  گام قدم (step size) است.

چرا از Langevin Dynamics استفاده می‌کنیم؟

1. دسترسی مستقیم به  $p_\theta(x)$  ممکن نیست:

در مدل‌های Score-based تنها اسکور  $\nabla_x \log p_\theta(x)$  یا تابع  $p_\theta(x)$  در اختیار است، نه خود

نمونه‌برداری مستقیم از آن. Langevin Dynamics نیازی به دانستن  $p_\theta(x)$  به طور صریح ندارد. کافی است اسکور توزیع را بدانیم تا جهت حرکت به سمت مناطق با احتمال بالاتر مشخص شود.

2. حرکت تدریجی به سمت توزیع هدف:

با افزودن گرادیان به عنوان نیروی رانشی (drift term) نمونه را به سمت نواحی با چگالی Langevin Dynamics بالاتر هدایت می‌کند و با افزودن نویز گوسی یک انتشار (diffusion) تصادفی فراهم می‌شود که تضمین می‌کند کل فضای حالت بررسی می‌شود. در نهایت، پس از تکرار کافی، توزیع نقاط نمونه‌برداری شده به توزیع هدف نزدیک می‌شود.

3. ارتباط با روش‌های MCMC:

یک روش MCMC است که دارای کارایی بالاتری نسبت به نمونه‌برداری ساده‌ی تصادفی Langevin Dynamics است، زیرا از اطلاعات گرادیان (اسکور) بهره می‌برد. این اطلاعات اضافی باعث می‌شود نمونه‌ها سریع‌تر به توزیع هدف همگرا شوند.

4. عدم نیاز به نرمال‌کننده (Partition Function):

در بسیاری از مدل‌های انرژی‌بنیان (Energy-based Models) یا Score-based Models، محاسبه‌ی ضریب نرمال‌کننده  $Z_\theta$  دشوار است. Langevin Dynamics برای نمونه‌برداری نیاز به این ضریب ندارد، تنها چیزی که نیاز دارد گرادیان لگاریتم توزیع (اسکور) است.

## زیر سوال پنجم :

پاسخ:

در روش‌های **Score-based**، ما ابتدا «اسکور» (گرادیان لگاریتم چگالی داده) را می‌آموزیم و سپس با استفاده از دینامیک لانژون (Langevin Dynamics) از توزیع داده نمونه‌برداری می‌کنیم. اما این رویکرد دو مشکل کلیدی دارد:

1. **منیفولد کم‌بعدی:** داده‌های واقعی غالباً روی یک منیفولد (چندريختار) کم‌بعدی در فضای پربعد متراکزند. در چنین حالتی، اسکور توزیع روی کل فضای پربعد خوب تعریف نمی‌شود و یادگیری مستقیم اسکور با مشکل مواجه می‌شود.
2. **ناحیه‌های با چگالی کم:** اگر توزیع داده چندین «مود» (حالت) جدا افتاده داشته باشد، لانژون دینامیکس استاندارد برای جابه‌جایی بین این مد‌ها که توسط ناحیه‌های با چگالی بسیار کم جدا شده‌اند، به سختی عمل می‌کند. این به معنای گند بودن فرایند نمونه‌گیری و احتمال گیر افتادن در یک مود مشخص است.

ایده روش **Annealed Langevin Dynamics** چیست؟

«بازپخت» (Annealing) در اینجا به معنای شروع نمونه‌گیری از یک توزیع نویزدار و کم‌کم کاهش دادن نویز است تا در نهایت به توزیع اصلی داده نزدیک شویم. به این ترتیب:

- ابتدا به داده‌ها نویز گاوی با واریانس زیاد اضافه می‌کنیم تا «توزیع نویزدار سطح بالا» ایجاد شود. در این توزیع، پشتیبانی داده‌ها دیگر محدود به یک منیفولد کم‌بعدی نیست و همچنین مودها به شکلی هستند که گذر از نواحی با چگالی کم آسان‌تر است.
  - با داشتن یک شبکه اسکور شرطی نسبت به مقدار نویز (Noise Conditional Score Network)، می‌توانیم اسکور را در سطوح نویز مختلف تخمین بزنیم.
  - حال برای نمونه‌گیری:
1. از توزیع نویزدار با نویز زیاد (و در نتیجه مودهای «دسترسی‌پذیرتر») شروع می‌کنیم و با لانژون دینامیکس نمونه‌های مقدماتی خوبی به دست می‌آوریم.
  2. سپس مرحله به مرحله نویز را کاهش می‌دهیم (آن را «anneal» می‌کنیم). در هر مرحله، توزیع کمی کمتر نویزدار می‌شود و ما از نمونه‌های مرحله قبلی به عنوان «نقطه شروع» استفاده می‌کنیم. چون این نمونه‌ها قبلاً در مودهای پرچگالی و نواحی قابل دسترس‌تر قرار گرفته‌اند، گذار به توزیع با نویز کمتر - که به توزیع اصلی داده نزدیک‌تر است - راحت‌تر و پایدار‌تر می‌شود.

مزیت این کار چیست؟

- با وجود نویز زیاد در ابتدا، مشکلات تعریف اسکُور روی منیفولد کم‌بعدی از بین می‌رود؛ چون داده‌ی نویزدار پشتیبانی کامل در سراسر فضای ورودی دارد و اسکُور در همه جا تعریف می‌شود.
- با استفاده از نویز زیاد، گذر از نواحی با چگالی کم آسان‌تر می‌شود و نمونه‌گیری بین مودهای مختلف توزیع اولیه ساده‌تر است.
- در ادامه که نویز کم می‌شود، توزیع نویزدار به توزیع اصلی شباهت پیدا می‌کند و در نهایت نمونه‌هایی به دست می‌آیند که از کیفیت و تنوع بالایی برخوردارند.

#### خلاصه:

یک روند چندمرحله‌ای است که از نویز زیاد به نویز کم حرکت می‌کند. در هر مرحله با لانژون دینامیکس و اسکُور آن مرحله نمونه‌ها بهبود می‌یابند و در نهایت به توزیع اصلی نزدیک می‌شویم. این روش موجب می‌شود مشکل تعریف اسکُور روی منیفولد حل شود و نمونه‌گیری سریع‌تر و مؤثرتر از توزیع داده اصلی صورت گیرد.

## سوالات پیاده سازی

### سوال یک:

در این بخش به پیاده سازی مدل های score based می پردازیم :

```
x_min, x_max = -15, 15
y_min, y_max = -15, 15
grid_size = 200
x, y = np.meshgrid(np.linspace(x_min, x_max, grid_size), np.linspace(y_min, y_max, grid_size))
points_grid = np.stack([x, y], axis=-1).reshape(-1, 2)

extent = [x_min, x_max, y_min, y_max]

def gaussian_pdf(x, mean, cov):
    diff = x - mean
    inv_cov = np.linalg.inv(cov)
    det_cov = np.linalg.det(cov)
    norm_const = 1 / (2 * np.pi * np.sqrt(det_cov))
    exponent = -0.5 * np.sum(diff @ inv_cov * diff, axis=1)
    return norm_const * np.exp(exponent)

def p_x(points, params):
    pdf1 = params['w1'] * gaussian_pdf(points, params['mean1'], params['cov1'])
    pdf2 = params['w2'] * gaussian_pdf(points, params['mean2'], params['cov2'])
    return pdf1 + pdf2

z = p_x(points_grid, params).reshape(x.shape)

def plot_heatmap(z, extent, title="Heatmap of Mixture of Gaussians"):
    plt.figure(figsize=(6, 6))
    plt.imshow(z, extent=extent, origin="lower", cmap="Reds")
    plt.title(title)
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.colorbar(label='Probability Density')
    plt.show()

def plot_train_test_samples(train_samples, test_samples, extent, title="Train and Test Samples"):
    plt.figure(figsize=(6, 6))
    plt.scatter(train_samples[:, 0], train_samples[:, 1], s=10, alpha=0.7, label="Train Samples", color='blue')
    plt.scatter(test_samples[:, 0], test_samples[:, 1], s=10, alpha=0.7, label="Test Samples", color='green')
    plt.xlim(extent[0], extent[1])
    plt.ylim(extent[2], extent[3])
    plt.title(title)
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_heatmap(z, extent=extent)

plot_train_test_samples(train_samples, test_samples, extent=extent, title="Train and Test Samples")
```

تصویر شماره (۱-۱-۴) : قطعه کد مربوط به توابع gaussian\_pdf

### ۳. بصری سازی داده ها

مرور کلی

برای درک بهتر ساختار داده‌های تولید شده و اطمینان از صحت فرآیند تولید، از روش‌های بصری‌سازی مختلفی استفاده شد. این بخش به تجسم چگالی احتمال داده‌ها و توزیع نمونه‌های آموزشی و آزمون می‌پردازد.

## روش‌شناسی

1. **ایجاد شبکه برای نقشه حرارتی:** یک شبکه دو بعدی با دامنه  $[15, 15]$  در هر دو بعد  $x_1$  و  $x_2$  و با وضوح  $200 \times 200$  نقطه ایجاد شد. این شبکه برای محاسبه چگالی احتمال مخلوط توزیع‌های گاوی در هر نقطه مورد استفاده قرار گرفت.

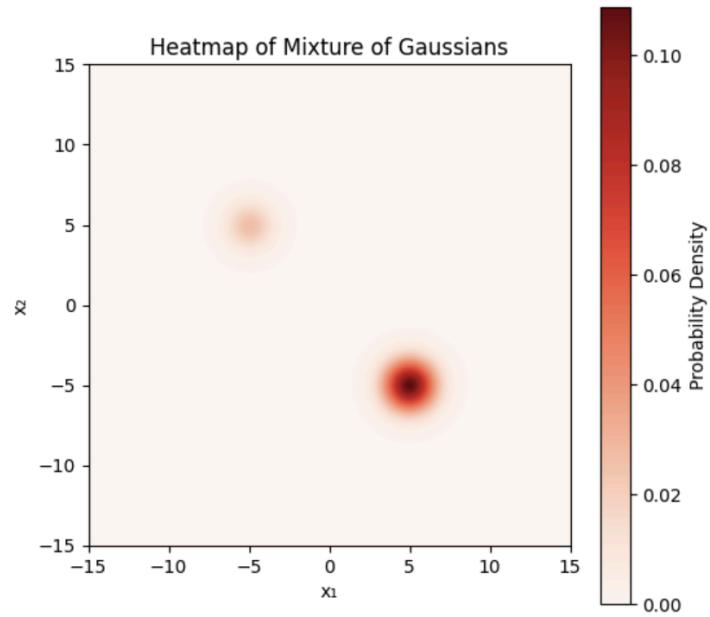
2. **محاسبه چگالی احتمال:** با استفاده از پارامترهای مخلوط توزیع‌های گاوی  $(w_1, \Sigma_1, \Sigma_2, \mu_1, \mu_2)$ ، چگالی احتمال در هر نقطه شبکه محاسبه شد. این مقادیر برای تولید نقشه حرارتی استفاده گردید.

3. **تجسم نقشه حرارتی:** با استفاده از مقادیر چگالی احتمال محاسبه شده، نقشه حرارتی توزیع داده‌ها رسم شد. این نقشه با استفاده از رنگ‌های گرم‌تر برای نشان دادن چگالی‌های بالاتر و رنگ‌های سرد‌تر برای چگالی‌های پایین‌تر ایجاد گردید.

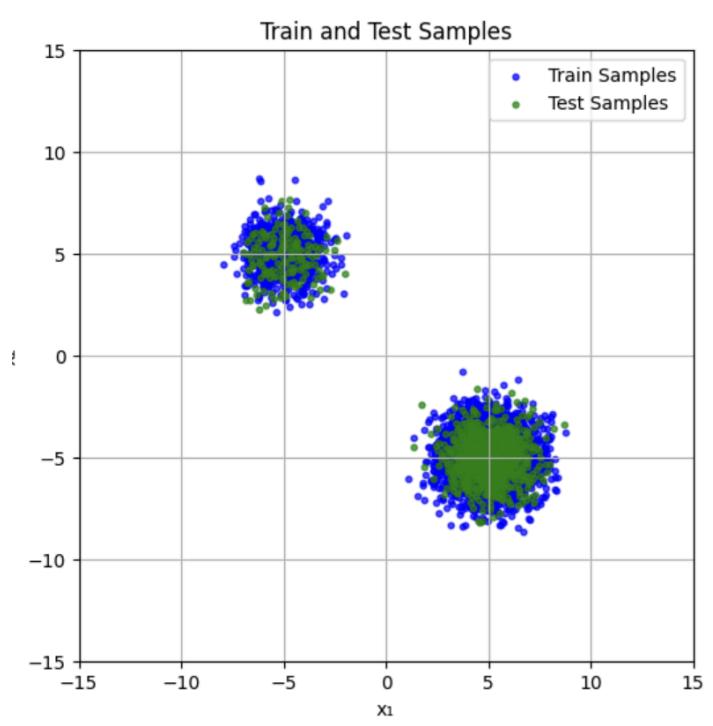
4. **نمودار پراکندگی نمونه‌های آموزشی و آزمون:** توزیع نمونه‌های آموزشی و آزمون با استفاده از نمودارهای پراکندگی بصری‌سازی شد تا اطمینان حاصل شود که داده‌ها به طور متعادل و بدون تعصب بین دو مجموعه تقسیم شده‌اند.

## نتایج

در شکل 3.1 نقشه حرارتی توزیع داده‌ها نمایش داده شده است که نواحی با چگالی بالا و پایین را به وضوح نشان می‌دهد. همچنین، شکل 3.2 توزیع نمونه‌های آموزشی و آزمون را به صورت پراکندگی نشان می‌دهد که به خوبی نمایانگر تقسیم متعادل داده‌ها بین دو مجموعه مختلف است.



تصویر شماره (۲-۱-۴) : نقشه حرارتی چگالی احتمال مخلوط دو توزیع گاوسی.



تصویر شماره (۳-۱-۴) : نمودار پراکندگی نمونه‌های آموزشی و آزمون.

## **خلاصه**

بصری‌سازی داده‌ها با استفاده از نقشه حرارتی و نمودارهای پراکنده‌گی، درک عمیق‌تری از توزیع داده‌ها و تقسیم‌بندی نمونه‌ها فراهم کرد. این تجسم‌ها اطمینان حاصل کردند که فرآیند تولید داده‌ها به درستی اجرا شده و داده‌های تولید شده برای مراحل بعدی آموزش و ارزیابی مدل مناسب هستند.

## **سوال ۲ :**

```

def add_noise(x, sigma):
    """
    Add Gaussian noise to the input data.

    Args:
        x (torch.Tensor): Tensor of shape [batch_size, 2]
        sigma (torch.Tensor): Tensor of shape [batch_size, 1]

    Returns:
        x_noisy (torch.Tensor): Noisy data
        epsilon (torch.Tensor): Noise added
    """
    epsilon = torch.randn_like(x) * sigma.expand_as(x)
    x_noisy = x + epsilon
    return x_noisy, epsilon

def score_matching_loss(x, x_noisy, epsilon, sigma, outputs):
    """
    Compute the score matching loss.

    Args:
        x (torch.Tensor): Original data [batch_size, 2]
        x_noisy (torch.Tensor): Noisy data [batch_size, 2]
        epsilon (torch.Tensor): Noise added [batch_size, 2]
        sigma (torch.Tensor): Noise level [batch_size, 1]
        outputs (torch.Tensor): Model outputs [batch_size, 2]

    Returns:
        torch.Tensor: Scalar loss value
    """
    term = outputs + epsilon / (sigma ** 2)
    loss = 0.5 * torch.mean(term ** 2)
    return loss

```

تصویر شماره (۱-۲-۴) : قطعه کد مربوط به `score_matching_loss` و `add_noise`

از این شبکه استفاده کرده ایم:

```

class ScoreNet(nn.Module):
    """Neural network to estimate the score function."""

    def __init__(self, input_dim=3, hidden_dim=256, num_layers=6):
        """
        Initialize the ScoreNet model.

        Args:
            input_dim (int): Dimension of the input (2 for data + 1 for sigma).
            hidden_dim (int): Number of hidden units in each layer.
            num_layers (int): Total number of layers.
        """
        super(ScoreNet, self).__init__()
        layers = []
        layers.append(nn.Linear(input_dim, hidden_dim))
        layers.append(nn.BatchNorm1d(hidden_dim))
        layers.append(nn.LeakyReLU())
        for _ in range(num_layers - 2):
            layers.append(nn.Linear(hidden_dim, hidden_dim))
            layers.append(nn.BatchNorm1d(hidden_dim))
            layers.append(nn.LeakyReLU())
        layers.append(nn.Linear(hidden_dim, 2))

        self.net = nn.Sequential(*layers)

    def forward(self, x, sigma):
        """
        Forward pass of the network.

        Args:
            x (torch.Tensor): Noisy data [batch_size, 2]
            sigma (torch.Tensor): Noise level [batch_size, 1]

        Returns:
            torch.Tensor: Estimated score [batch_size, 2]
        """
        x_sigma = torch.cat([x, sigma], dim=1)
        return self.net(x_sigma)

```

تصویر شماره (۴-۳-۲) : قطعه کد مربوط به کلاس ScoreNet

**توضیح کوتاه درباره مدل ScoreNet**

مدل ScoreNet یک شبکه عصبی است که برای تخمین تابع نمره (Score Function) در مدل‌های تولیدی مبتنی بر نمره طراحی شده است. این شبکه به‌طور خاص برای تخمین گرادیان لگاریتم چگالی احتمال داده‌ها آموزش دیده است، که برای تولید داده‌های جدید و شبیه‌سازی از توزیع داده‌های واقعی کاربرد دارد.

### معماری مدل:

1. **ورودی‌ها:** شبکه ورودی‌هایی با ابعاد ۳ دریافت می‌کند:
  - ۲ بعد برای داده‌های اصلی.
  - ۱ بعد برای سطح نویز (sigma)، که نشان‌دهنده میزان نویز اضافه شده به داده‌ها است.
2. **لایه‌های مخفی:** شبکه شامل ۶ لایه است که در هر لایه:
  - یک تبدیل خطی (Linear) اعمال می‌شود.
  - نرمال‌سازی بج (Batch Normalization) برای بهبود سرعت و ثبات آموزش.
  - فعال‌سازی Leaky ReLU برای جلوگیری از مشکلات فحال‌سازی و معرفی غیرخطی بودن به مدل.
3. **لایه خروجی:** شبکه به‌طور خطی ۲ مقدار (گرادیان‌های نمره) برای هر داده تولید می‌کند.

### عملکرد مدل:

- ورودی‌های داده (با نویز) به شبکه داده می‌شوند و شبکه نمره‌ای برای هر نقطه داده پیش‌بینی می‌کند.
- این پیش‌بینی‌ها برای محاسبه خطای مدل و بهروزرسانی وزن‌ها از طریق الگوریتم‌های یادگیری استفاده می‌شود.

### کاربرد در مدل‌های تولیدی:

- پس از آموزش، ScoreNet می‌تواند برای نمونه‌برداری از توزیع داده‌ها استفاده شود. این مدل با استفاده از تخمین‌های نمره، فرآیندهایی مانند داینامیک‌های لانژون را برای تولید داده‌های جدید هدایت می‌کند.

```

sigma_values = [1, 3, 7]
models = {}
loss_histories = {}

for sigma in sigma_values:
    print(f"\n--- Phase 1: Training with Fixed Sigma={sigma} ---\n")

    model = ScoreNet(input_dim=3, hidden_dim=256, num_layers=6)
    print(f"Training ScoreNet with fixed sigma={sigma}...")

    train_data = torch.tensor(train_samples).float()
    dataset = TensorDataset(train_data)
    data_loader = DataLoader(dataset, batch_size=256, shuffle=True, num_workers=2)

    model, loss_history = train_score_model(
        model,
        data_loader,
        sigma_schedule=None,
        epochs=100,
        learning_rate=0.001,
        scheduler_step=100,
        scheduler_gamma=0.5,
        clip_grad=1.0,
        patience=100,
        fixed_sigma=sigma
    )

    models[sigma] = model
    loss_histories[sigma] = loss_history

    plt.figure(figsize=(8, 5))
    plt.plot(loss_history, label=f'Training Loss (Fixed Sigma={sigma})')
    plt.xlabel("Epoch")
    plt.ylabel("Average Loss")
    plt.title(f'Training Loss History (Fixed Sigma={sigma})')
    plt.grid(True)
    plt.legend()
    plt.show()

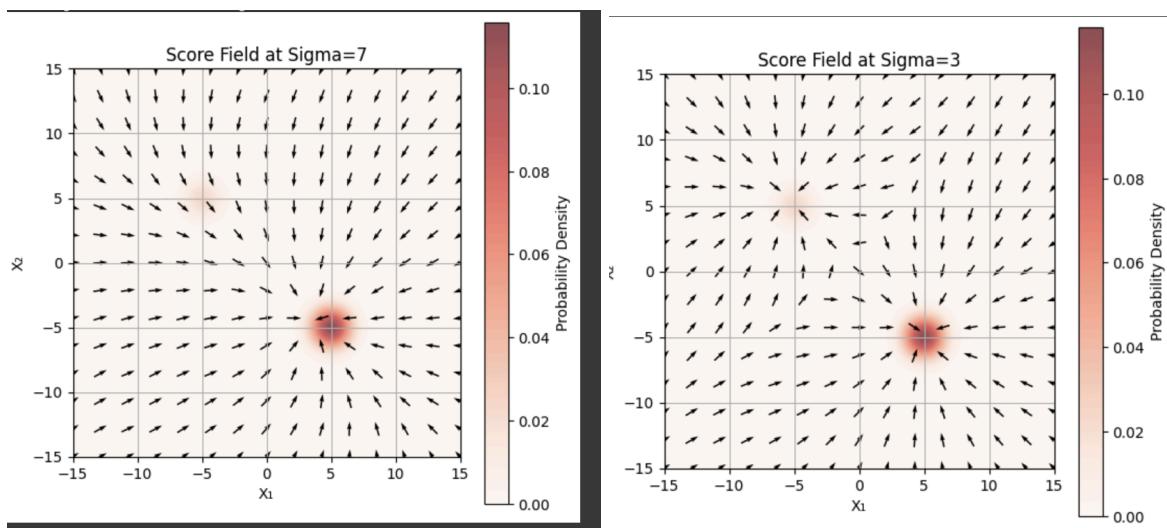
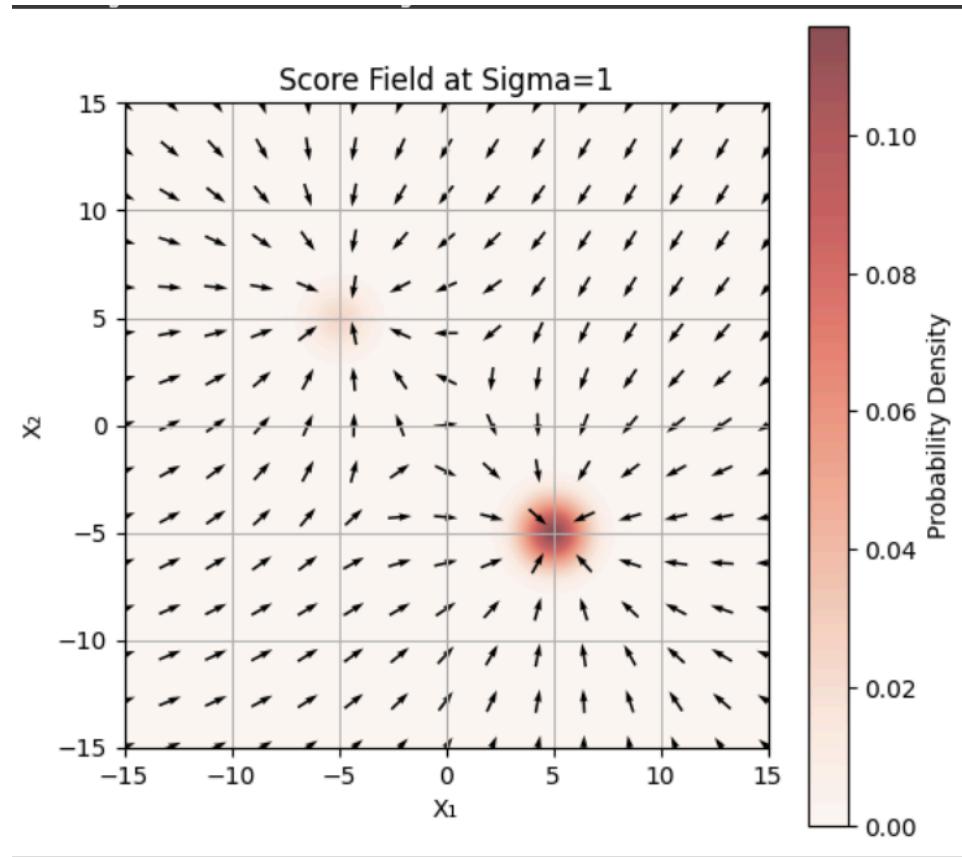
for sigma in sigma_values:
    print(f"\nVisualizing score fields for Sigma={sigma}...")
    sigma_visualization = [1, 3, 7]
    plot_score_field_individual(score_function_gt, params, extent=extent, sigma_values=[sigma],
                                 title=f'Ground Truth Score Fields at Different Sigma Levels (Sigma={sigma})')

    print(f"\nPerforming sampling with fixed sigma={sigma}...")
    perform_sampling_and_plot(models[sigma], epoch=f'Fixed Sigma={sigma}', num_points=1000)

```

تصویر شماره (۱-۳-۴) : قطعه کد مربوط به فاز یک - سیگما برابر با ثابت برای ۱ و ۳ و ۷

در این قسمت از کد داریم به ازای سیگما های ۱ و ۳ و ۷ مقادیر را رسم میکنیم و سمپل گیری هم می کنیم :



تصویر شماره (۲-۳-۴) : تصاویر Score\_Field مربوط به سه سیگما

### 1. حالت سیگما 1:

- در این حالت، با توجه به مقدار پایین سیگما، نقاط به طور غیر یکنواخت و در نواحی محدودتری توزیع می‌شوند. این یعنی داده‌ها به درستی شبیه‌سازی نشده‌اند و پراکندگی غیر طبیعی ایجاد می‌شود. انتظار می‌رود که در این حالت، نقاط به طور متتمرکز در مرکز توزیع قرار بگیرند، اما به دلیل تأثیر نویز، پراکندگی بیشتری نسبت به حالت ایده‌آل داریم که این نشان‌دهنده عدم توزیع صحیح داده‌هاست.

### 2. حالت سیگما 3:

- در این حالت، که سیگما مقداری بیشتر از 1 دارد، پراکندگی داده‌ها همچنان به شکل نادرستی توزیع شده‌اند. نقاط به طور مساوی در نواحی مختلف پخش شده‌اند، در حالی که این یک توزیع طبیعی نیست. در اینجا هم همچنان مدل نتوانسته است به درستی نقاط را با توجه به ویژگی‌های داده‌ها توزیع کند و نتیجه مشابه حالت سیگما 1 است.

### 3. حالت سیگما 7:

- در این حالت، سیگما بیشتر است و اثر نویز به وضوح دیده می‌شود. در اینجا، مدل توانسته است به طور بهتری داده‌ها را توزیع کند و نقاط بیشتر به سمت مرکز پایینی متمایل شده‌اند. این نشان‌دهنده توزیع صحیح‌تری است که بر اساس احتمال بالاتر در ناحیه‌ای خاص متتمرکز شده است. این رفتار نشان می‌دهد که مدل به درستی شبیه‌سازی کرده است و به سمت ناحیه‌ای با احتمال بالاتر حرکت کرده است.

---

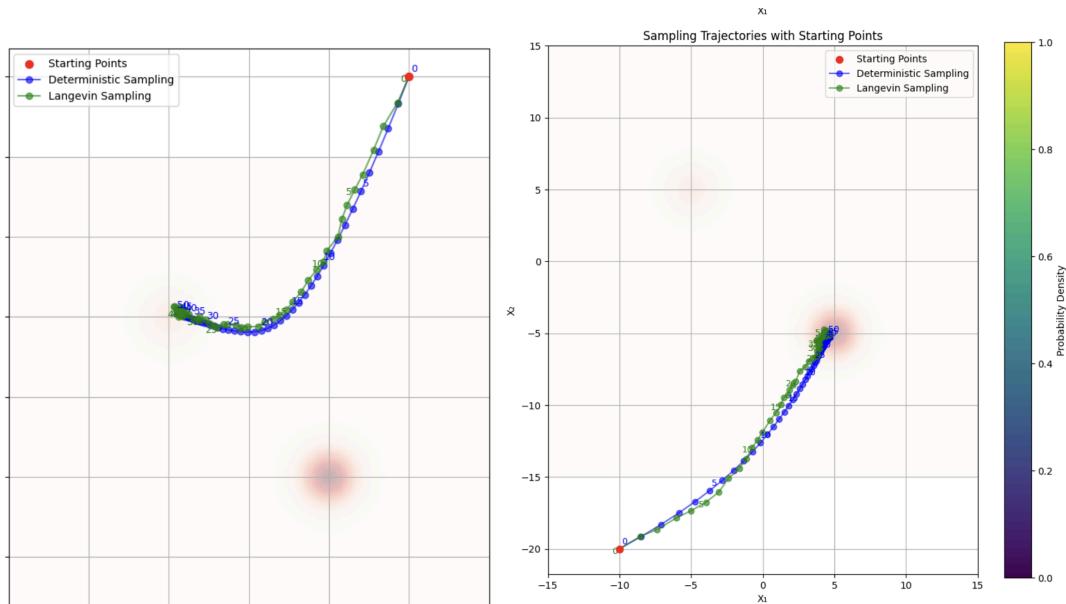
### مقایسه کلی:

- در حالت سیگما 1، پراکندگی داده‌ها به طور غیر یکنواخت است و توزیع طبیعی ندارد.
- در حالت سیگما 3، داده‌ها به طور نادرست و مساوی در نواحی مختلف پراکنده شده‌اند.
- در حالت سیگما 7، داده‌ها به طور صحیح‌تر و به سمت ناحیه‌ای با احتمال بالاتر متتمرکز شده‌اند.

این اصلاحات به وضوح نشان می‌دهد که با افزایش سیگما، مدل بهتر توانسته است نقاط را به درستی توزیع کند و توزیع صحیح‌تر را شبیه‌سازی نماید.

---

### سوال سوم:



تصویر شماره (۳-۴): تصاویر مربوط به تفاوت دو نوع سمپل گیری

در این آزمایش، ما از دو نقطه شروع متفاوت (یکی با مختصات [10,20] و دیگری با مختصات [10,-20]) برای مقایسه دو روش نمونه‌گیری تصادفی **Deterministic Sampling** و **Langevin Sampling** استفاده کردیم. هدف این است که تأثیر تصادفی بودن در فرآیند Langevin و همچنین تطبیق مدل با توزیع اصلی داده‌ها را بررسی کنیم.

---

### نقطه شروع اول: [10,20]

برای نقطه اول، در روش **Deterministic Sampling**، مشاهده می‌کنیم که حرکت به صورت مستقیم‌تر و بدون هیچ‌گونه نوسان یا رفتار تصادفی است. این حرکت به سمت مرکز توزیع داده‌ها (حالت پایدار) نشان می‌دهد که مدل توانسته گرادیان‌های صحیح را برای جهت‌دهی به نقاط به سمت توزیع هدف محاسبه کند.

در مقابل، در روش **Langevin Sampling**، مسیر حرکت تصادفی‌تر است و نقاط در طول مسیر نوسان دارند. دلیل این رفتار، اضافه شدن نویز تصادفی در هر مرحله از نمونه‌گیری است که به مدل کمک می‌کند نواحی جدیدی از فضای داده را جستجو کند. این ویژگی در کاربردهایی که به تنوع بیشتری در نمونه‌گیری نیاز دارند (مثل یادگیری احتمالاتی) مفید است.

---

## نقطه‌ی شروع دوم: [20-10]

برای نقطه‌ی دوم نیز الگوی مشابهی مشاهده می‌شود. در روش **Deterministic Sampling**, مسیر حرکت صاف و بدون نوسان است و نقاط به صورت مستقیم به سمت مرکز توزیع حرکت می‌کنند. این نشان می‌دهد که مدل گرادیان‌ها را به درستی محاسبه کرده و بدون اعمال هیچ رفتار تصادفی، نمونه‌گیری قابل اعتمادی انجام می‌دهد.

در روش **Langevin Sampling**, نوسانات و رفتار تصادفی بیشتری مشاهده می‌شود. این رفتار تصادفی ممکن است باعث شود که نمونه‌ها به نواحی غیرمرکز تری از توزیع بروند، اما همچنان به صورت کلی به سمت مرکز توزیع همگرا می‌شوند. این ویژگی نشان‌دهنده تاثیر نویز تصادفی در بهبود تنوع نمونه‌ها است.

---

## نتیجه‌گیری

- این روش بسیار دقیق و قابل پیش‌بینی است و به دلیل نبود نوسانات، مسیرهای کامل‌آقاً قابل اعتمادی ارائه می‌دهد. اما ممکن است در برخی کاربردها، مانند کشف نواحی غیرمرکز توزیع داده‌ها، محدودیت داشته باشد.
  - این روش، به دلیل اعمال نویز تصادفی، حرکت متنوع‌تر و اکتشافی‌تری ارائه می‌دهد. این ویژگی به خصوص در مسائل یادگیری احتمالاتی یا در جستجوی حالت‌های کمتر محتمل توزیع داده‌ها، مفید است.
- بنابراین، در کاربردهایی که نیاز به اکتشاف و تنوع بیشتری وجود دارد، روش **Langevin Sampling** مناسب‌تر است، در حالی که در مواردی که به دقت بالاتر و پیش‌بینی‌پذیری نیاز داریم، روش **Deterministic Sampling** ترجیح داده می‌شود.

در اینجا دوتابع برای سمبول گیری را می‌بینیم:

```

def deterministic_sampling(model, start_points, num_steps=50, step_size=0.1):
    """
    Perform deterministic sampling using the score model.

    Args:
        model (nn.Module): Trained score network
        start_points (torch.Tensor): Starting points [batch_size, 2]
        num_steps (int): Number of steps to perform
        step_size (float): Step size for updates

    Returns:
        trajectory (list): List of tensors representing the trajectory
    """
    current_points = start_points.to(device).detach()
    trajectory = [current_points]
    with torch.no_grad():
        for _ in range(num_steps):
            sigma_val = torch.ones(current_points.shape[0], 1, device=device)
            grad = model(current_points, sigma_val)
            current_points = current_points + step_size * grad
            trajectory.append(current_points)
    return trajectory

```

تصویر شماره (۴-۴-۴) : قطعه کد مربوط به deterministic\_sampling

```

def langevin_sampling(model, start_points, num_steps=50, step_size=0.1):
    """
    Perform Langevin sampling using the score model.

    Args:
        model (nn.Module): Trained score network
        start_points (torch.Tensor): Starting points [batch_size, 2]
        num_steps (int): Number of steps to perform
        step_size (float): Step size for updates

    Returns:
        trajectory (list): List of tensors representing the trajectory
    """
    current_points = start_points.to(device).detach()
    trajectory = [current_points]
    with torch.no_grad():
        for _ in range(num_steps):
            sigma_val = torch.ones(current_points.shape[0], 1, device=device)
            grad = model(current_points, sigma_val)
            noise = torch.randn_like(current_points) * np.sqrt(0.1 * step_size)
            current_points = current_points + step_size * grad + noise
            trajectory.append(current_points)
    return trajectory

```

تصویر شماره (۵-۴-۴) : قطعه کد مربوط به langevin sampling

## توضیحات مربوط به توابع Deterministic Sampling و Langevin Sampling

در این دو تابع، فرآیند نمونهگیری (Sampling) با استفاده از مدل آموزش‌دیده (Score Model) انجام می‌شود. هدف این است که نقاط اولیه (Starting Points) با استفاده از گرادیان تابع احتمال، به سمت توزیع اصلی داده‌ها هدایت شوند.

---

### تابع Langevin Sampling

**هدف:** این روش از نویز تصادفی در فرآیند نمونهگیری استفاده می‌کند. این نویز به مدل کمک می‌کند تا نواحی مختلفی از فضای نمونهگیری را کشف کرده و از گیر افتادن در نقاط خاص جلوگیری کند. این ویژگی در یادگیری احتمالاتی بسیار مفید است.

**ساختار تابع:**

#### 1. ورودی‌ها:

- **model:** مدل یادگیری (شبکه عصبی) که گرادیان توزیع داده‌ها را پیش‌بینی می‌کند.
- **start\_points:** نقاط اولیه نمونهگیری، به صورت یک `Tensor` در فضای دو بعدی.
- **num\_steps:** تعداد مراحل (Iterations) برای نمونهگیری.
- **step\_size:** اندازه گام برای هر مرحله حرکت.

#### 2. خروجی:

- **trajectory:** لیستی از نقاطی که در هر مرحله از نمونهگیری تولید می‌شوند.
- **3. روند اجرا:**

- ابتدا نقاط اولیه به عنوان نقطه شروع نمونهگیری تعیین می‌شوند.
- در هر مرحله:
  - گرادیان توزیع در نقطه فعلی با استفاده از مدل محاسبه می‌شود.

- نویز تصادفی با اندازه‌ای مناسب با `step_size` اضافه می‌شود.
  - نقطه جدید با حرکت به سمت گرادیان و اعمال نویز محاسبه می‌شود.
    - نقاط جدید به لیست مسیر (`trajectory`) اضافه می‌شوند.
    - این فرآیند برای تعداد مراحل مشخص (`num_steps`) ادامه می‌یابد.
- 

## Deterministic Sampling

هدف: این روش نمونه‌گیری کاملاً غیرتصادفی است. در اینجا هیچ نویز تصادفی اضافه نمی‌شود و نقاط فقط بر اساس گرادیان تابع احتمال حرکت می‌کنند. این ویژگی باعث پیش‌بینی‌پذیری و دقت بالای حرکت می‌شود.

ساختار تابع:

1. ورودی‌ها:

- مشابه تابع `.Langevin Sampling`

2. خروجی:

- مشابه تابع `.Langevin Sampling`

3. روند اجرا:

ابتدا نقاط اولیه تعیین می‌شوند.

- در هر مرحله:

■ گرادیان توزیع در نقطه فعلی محاسبه می‌شود.

■ نقطه جدید فقط با استفاده از گرادیان و بدون اضافه شدن نویز محاسبه می‌شود.

- نقاط جدید به لیست مسیر اضافه می‌شوند.

- این فرآیند برای تعداد مشخصی از مراحل ادامه می‌یابد.

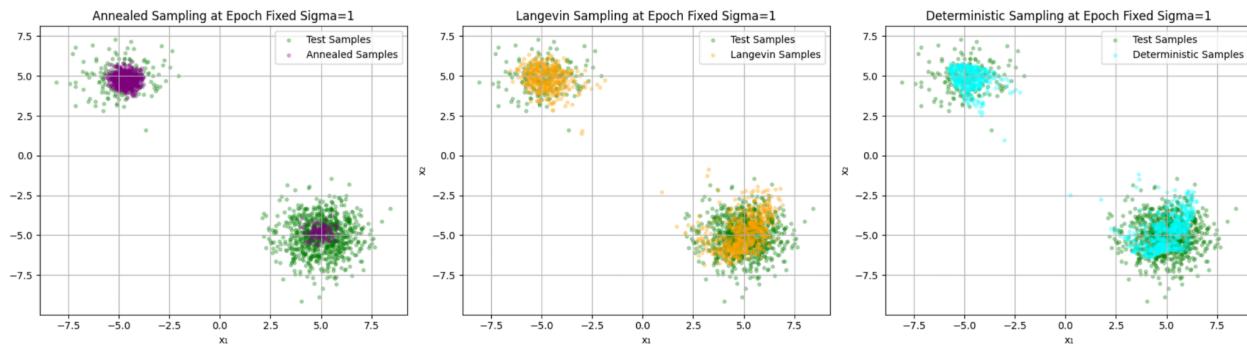
## نتیجه‌گیری

- روش **Langevin Sampling** به دلیل اضافه کردن نویز تصادفی می‌تواند نواحی جدیدی از فضای داده را کشف کند و برای کاربردهایی که نیاز به تنوع و اکتشاف بیشتر دارند، مفید است.

روش Deterministic Sampling به دلیل نبود نویز، مسیرهای صاف و پیش‌بینی‌پذیری ایجاد می‌کند و برای کاربردهایی که به دقت و حرکت مستقیم نیاز دارند، مناسب‌تر است.

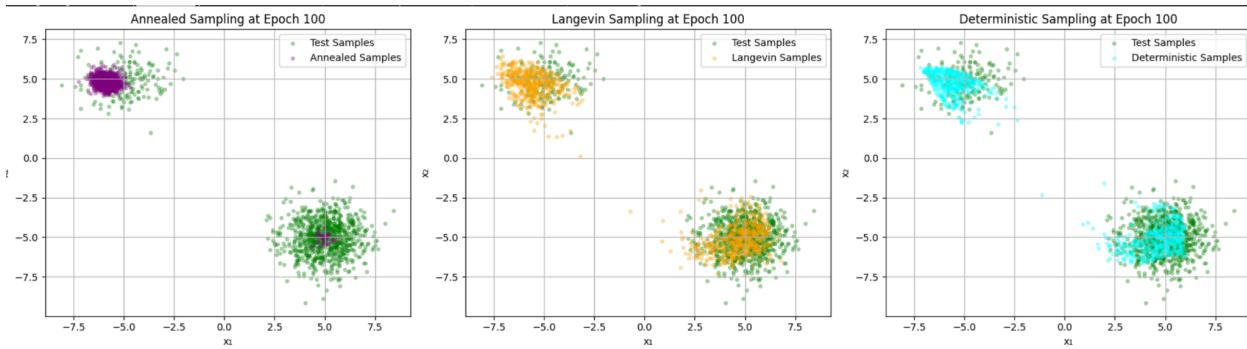
## سوال ۴:

### حالت ۱



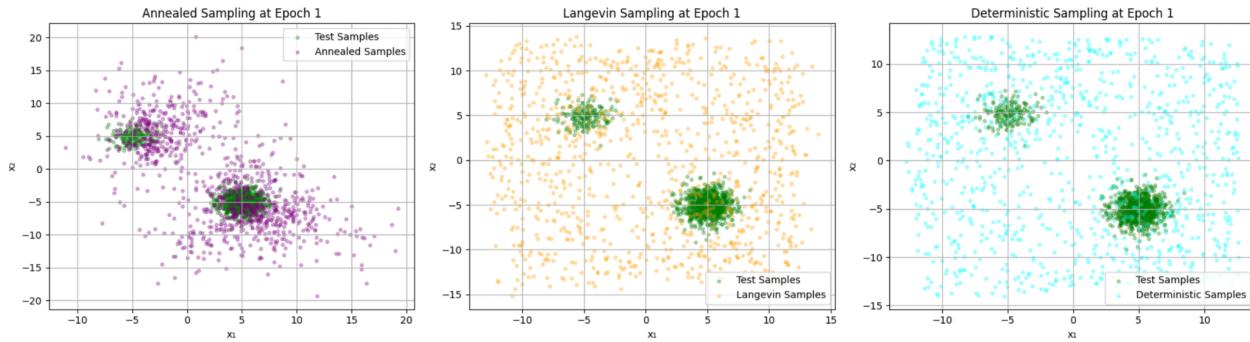
تصویر شماره (۱-۴-۴) : سمپل ها با  $\sigma = 1$

### حالت ۳



تصویر شماره (۲-۴-۴) : سمپل ها با  $\sigma = 3$

### حالت ۷



تصویر شماره (۴-۳) : سمپل ها با  $\sigma = 7$

مشاهده می‌کنیم که در حالتی که سیگما مقدار ثابتی باشد و مدل تنها با همان مقدار سیگما آموزش داده شود، تعداد نمونه‌هایی که به دو ناحیه اصلی توزیع داده‌ها می‌رسند، تقریباً یکسان است. این وضعیت در تضاد با هدف اصلی ما قرار دارد؛ چرا که در حالت ایده‌آل، تعداد نمونه‌هایی که به هر ناحیه می‌رسند، باید متناسب با احتمال آن ناحیه باشد. به عبارت دیگر، ناحیه‌ای که احتمال بیشتری دارد باید تعداد نمونه بیشتری دریافت کند و ناحیه‌ای که احتمال کمتری دارد، تعداد کمتری نمونه دریافت کند.

در حالت **annealed sampling** نیز رفتار مشابهی دیده می‌شود. در این روش، نمونه‌ها ابتدا به سمتی حرکت می‌کنند که احتمال بیشتری دارد. این رفتار ناشی از آن است که در مراحل اولیه نمونه‌گیری، سیگما مقدار زیادی دارد و نویز غالب است. این موضوع باعث می‌شود که نمونه‌ها از نقاط شروع مختلف، به سمت ناحیه‌هایی که احتمال بیشتری دارند حرکت کنند. به همین دلیل، اگر سیگما در کل فرآیند ثابت نگه داشته شود یا کاهش کنترل شده‌ای نداشته باشد، بسیاری از نمونه‌ها به صورت نعادلانه در ناحیه‌ای با احتمال بیشتر متمرکز می‌شوند.

این مشکل به ویژه در مدل‌هایی که تنها با یک سیگما ثابت آموزش دیده‌اند، قابل مشاهده است. چنین مدل‌هایی توانایی درک و تمایز بهتر میان نواحی با احتمال‌های مختلف را ندارند و به همین دلیل تمامی نمونه‌ها را به سمت ناحیه‌ای با احتمال بیشتر هدایت می‌کنند. این موضوع نشان می‌دهد که مدل نیاز به تنظیم دقیق‌تر در انتخاب مقادیر سیگما و استفاده از نویز متغیر دارد.

با اعمال **annealed sampling** به درستی، یعنی کاهش تدریجی و برنامه‌ریزی شده سیگما در طول مراحل نمونه‌گیری، می‌توان به تعادل بیشتری در توزیع نمونه‌ها رسید. به این معنا که در طول فرآیند، احتمال‌ها به دقت بیشتری در نظر گرفته شوند و نواحی با احتمال کمتر نیز به نسبت اهمیت خود تعداد نمونه دریافت کنند. اما

بدون اعمال کاهش مناسب سیگما یا تنظیمات صحیح، نمونه‌ها به صورت ناعادلانه به سمت نواحی با احتمال بیشتر منحرف خواهند شد.

این رفتار به طور کلی بیانگر اهمیت استفاده از رویکردهای تطبیقی‌تر در فرآیند نمونه‌گیری و همچنین آموزش مدل با مقادیر مختلف سیگما به جای ثابت نگه داشتن آن است. در نهایت، تنظیم مناسب سیگما و پیاده‌سازی دقیق فرآیند annealed sampling می‌تواند منجر به نمونه‌گیری واقعی‌تر و متعادل‌تر شود.

## سوال ۵

این تابع سمپل گیری annealed sampling میباشد :

```
def annealed_sampling(model, start_points, sigma_start=20, sigma_end=1, steps_per_sigma=50, num_sigmas=20, step_size=0.1):
    """
    Perform annealed Langevin sampling: reduce sigma in steps, performing Langevin updates at each sigma.

    Args:
        model (nn.Module): Trained score network
        start_points (torch.Tensor): Starting points [batch_size, 2]
        sigma_start (float): Initial sigma value
        sigma_end (float): Final sigma value
        steps_per_sigma (int): Number of Langevin steps per sigma
        num_sigmas (int): Number of sigma steps
        step_size (float): Step size for updates

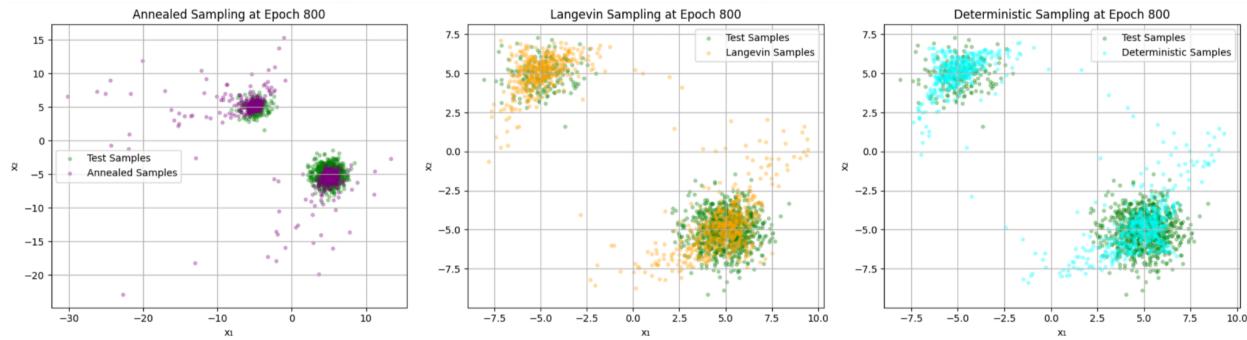
    Returns:
        final_samples (np.ndarray): Final sampled points [batch_size, 2]
        samples_at_each_sigma (list): List of sampled points at each sigma step
        sigma_values (np.ndarray): Array of sigma values used
    """
    current = start_points.clone().to(device)
    sigma_values = np.linspace(sigma_start, sigma_end, num_sigmas)
    samples_at_each_sigma = []

    with torch.no_grad():
        for si in sigma_values:
            sigma_t = torch.ones(current.shape[0], 1, device=device) * si
            for _ in range(steps_per_sigma):
                grad = model(current, sigma_t)
                noise = torch.randn_like(current) * np.sqrt(0.1 * step_size) * si
                current = current + step_size * grad + noise
            samples_at_each_sigma.append(current.clone().cpu().numpy())

    return current.cpu().numpy(), samples_at_each_sigma, sigma_values
```

تصویر (۱-۵-۱۴) : قطعه کد مربوط به annealed sampling

برای این قسمت از مقدار زندوم واریانس استفاده کرده بودیم که بین ۱ و ۲۰ بود و هر بار سمپل گیری کرده بودیم



تصویر شماره (۴-۵-۲):

در حالت‌های مختلف سمپلگیری (Sampling)، تفاوت‌های قابل‌توجهی در نحوه توزیع داده‌ها مشاهده می‌شود:

۱. در این دو حالت، مقادیر به‌طور یکسان و مساوی

بین کلاس‌ها یا نقاط مختلف پخش می‌شوند. این روش‌ها معمولاً به‌دبال پیدا کردن توزیع‌های یکنواخت و به‌هم‌ریخته در فضای ویژگی‌ها هستند.

۲. برخلاف حالت‌های قبلی، در این روش به‌طور نابرابر و به اندازه‌ی احتمال هر کدام از کلاس‌ها، داده‌ها سمپل می‌شوند. این بدان معناست که داده‌ها به‌طور غیر یکنواخت توزیع می‌شوند و به احتمال هر کدام از کلاس‌ها بیشتر توجه می‌شود. در این روش، فرآیند نمونه‌گیری به‌تدریج دقیق‌تر و متناسب با احتمال هر کلاس می‌شود.

- Goodfellow, I., et al. (2014). Generative Adversarial Nets. *Advances in Neural Information Processing Systems*, 27, 2672-2680.
- Salimans, T., et al. (2016). Improved Techniques for Training GANs. *Advances in Neural Information Processing Systems*, 29, 2234-2242.
- Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein GAN. arXiv preprint arXiv:1701.07875.
- Behrmann, J., Grathwohl, W., Chen, R. T. Q., Duvenaud, D., Jacobsen, J.-H. (2019). **Invertible Residual Networks**. *International Conference on Machine Learning (ICML)*.
- Chen, R. T. Q., Behrmann, J., Duvenaud, D., Jacobsen, J.-H. (2019). **Residual Flows for Invertible Generative Modeling**. *Advances in Neural Information Processing Systems (NeurIPS)*.
- Hutchinson, M. F. (1990). **A Stochastic Estimator of the Trace of the Influence Matrix for Laplacian Smoothing Splines**. *Communications in Statistics - Simulation and Computation*.
- Chatgpt.com