STATS-413-HW-7

Author: Hochan Son

UID: 206547205

Date: @December 7, 2025

Problem 1.

Consider 1 billion people distributed across three states $(1, 2, 3)$. Let $p(x)$be the number (in billion) of people in state $x, x \subseteq (1, 2, 3)$. Let $p(y|x)$ be the fraction of those in state x who will move to state $y, y \subseteq (1, 2, 3)$. If we randomly sample a person, $p(x)$ is the probability that the person is in x, and $p(y|x)$ is the conditional probability that this person will move to y if this person is in x. Let $\tilde{p}(y)$ be the number (in billion) of people who will end up in y, which can again be translated into the probability that the random person ends up in y.

(1) Explain that

$$p(x, y) = p(x)p(y|x),$$

where $p(x, y)$ is the number of people who are in x and who end up in y. This is the chain rule.

Explain that

$$\tilde{p}(y) = \sum_x p(x,y) = \sum_x p(x)p(y|x)$$

The first equation is the rule of $marginalization$. The last equation is the rule of total probability.

1. **Chain rule**: the join count $p(x,y)$ of people initially in x and ending in y is:

   $$p(x,y) = p(x)p(y|x)$$

   where

   - $p(x)$: number of people in state x
   - $p(y|x)$: fraction of people moving from state x to state y
   - $\tilde{p}(y)$: number ending up in state y

2. The Marginalization rule: the final population $\tilde{p}(y)$ in state y is the sums up people arriving in y from all possible starting states.

   $$\tilde{p}(y) = \sum_x p(x,y) = \sum_x p(x)p(y|x)$$

(2) Let p(x|y) be the faction of those people in y who come from x. Explain that

$$p(x|y) = \frac{p(x,y)}{\tilde{p}(y)} = \frac{p(x)p(y|x)}{\tilde{p}(y)} = \frac{p(x)p(y|x)}{\sum_x p(x)p(y|x)}$$

The first equation is the rule of conditioning. The final equation is known as Bayes' rule if we interpret x as the cause and y as the effect.

As to conditionals, we may call $p(y|x)$ the forward conditional, and $p(x|y)$ the backward conditional. The above three rules (chain rule, marginalization, conditioning) underlie all the probability calculations in machine learning

1. The Chain rule: explained above (1.1)
2. The Marginalization rule: explained above (1.2)

3. The Conditioning rule: the fraction p(x|y) of people in state y who comes

The rule of conditioning (conditional probability law):

$$p(x|y) = \frac{p(x,y)}{\tilde{p}(y)} = \frac{p(x)p(y|x)}{\tilde{p}(y)}$$

where

- $p(x|y)$ is probability of x given y
- $p(x,y)$= reward function for given (x,y)
- $\tilde{p}(y)$ = total probability of y

Bayes' Rule:

$$p(x|y) = \frac{p(x)p(y|x)}{\sum_x p(x)p(y|x)}$$

where

- $p(y|x)$is the **forward conditional**, where people go from state x to state y.
- $p(x|y)$ is the **backward conditional**, where people move from state y to state x.
- $p(x,y)$ is the **joint probability**, people in state x initially AND transition to state y.

$$p(x,y) = p(x)p(y|x)$$

- $\tilde{p}(y)$ is the new marginal distribution, where the population ends up after moving from the initial distribution $p(x)$:

$$\tilde{p}(y) = \sum_x p(x)p(y|x)$$

(3) Suppose for those $\tilde{p}(y)$ billion people in y, we send $p(x|y)$ of them back to x. Explain that the distribution of the 1 billion people will be back to $p(x)$, even though the people in x now may not be the same people who were in x before.

1. Forward: people transition from x to y using p(y|x)
2. Backward: people transition in y return to x using p(x|y)

Why it works: p(x|y) is calibrated so that:

- Number going x→ y = $p(x)p(y|x)$
- Number going y→x from those $\tilde{p}(y) = \tilde{p}(y)p(x|y) = p(x)p(y|x)$
- net result: exact reversal

(4) Please illustrate the above using concrete numbers:

Let's illustrate with some numbers:

Using these numbers, we can calculate with 1 billion people, p(x) becomes concrete:

- p(1) = 0.4 means 400 million people are in state 1
- p(2|1) = 0.3 means 30% of people in state 1 transition to state 2
- p(1|2) = of people currently in state 2, what fraction come from state 1

Table 1.4.1: Initial distribution, $p(x)$

$State(x)$	$p(x)$	Interpretation
1	0.4	0.4 billion people
2	0.3	0.3 billion people
3	0.3	0.3 billion people

To calculate the backward conditional probability $p(x|y)$

$$p(x|y) = \frac{p(x)p(y|x)}{\tilde{p}(x)}$$

For state $y = 1$: (people who ended in state 1)

For $\tilde{p}(1)$:

$$\tilde{p}(1) = \sum_{x=1}^{3} p(x)p(1|x)$$

- $\tilde{p}(1) = p(1)p(1|1) + p(2)p(1|2) + p(3)p(1|3)$

$= (0.4)(0.6) + (0.3)(0.2) + (0.3)(0.1)$
$= 0.24 + 0.06 + 0.03$
$= 0.33$

$$p(x|1) = \frac{p(x)p(1|x)}{\tilde{p}(1)} = \frac{p(x)p(1|x)}{0.33}$$

- $p(1|1) = (0.4 \times 0.6)/\tilde{p}(1) = 0.24/0.33 = 0.727,$
  $p(2|1) = (0.4 \times 0.3)/\tilde{p}(1) = 0.06/0.33 = 0.182,$
  $p(3|1) = (0.4 \times 0.1)/\tilde{p}(1) = 0.04/0.33 = 0.121$

For state $y = 2$: (people who ended in state 2)

- $\tilde{p}(2) = p(1)p(2|1) + p(2)p(2|2) + p(3)p(2|3) = (0.4)(0.3) + (0.3)(0.5) + (0.3)(0.4) = 0.12 + 0.15 + 0.12 = 0.39$

$$p(x|2) = \frac{p(x)p(2|x)}{0.39}$$

- $p(1|2) = (0.3)(0.3)/\tilde{p}(2) = 0.09/0.39 = 0.231$
  $p(2|2) = (0.3)(0.5)/\tilde{p}(2) = 0.15/0.39 = 0.385$
  $p(3|2) = (0.3)(0.4)/\tilde{p}(2) = 0.12/0.39 = 0.308$

For state y=3: (people who ended in state 3)

- $\tilde{p}(3) = p(1)p(3|1) + p(2)p(3|2) + p(3)p(3|3) = (0.4)(0.1) + (0.3)(0.3) + (0.3)(0.5) = 0.04 + 0.09 + 0.15 = 0.28$

$$p(x|3) = \frac{p(x)p(3|x)}{0.28}$$

- $p(1|3) = (0.3)(0.1)/\tilde{p}(3) = 0.03/0.28 = 0.107$
  $p(2|3) = (0.3)(0.3)/\tilde{p}(3) = 0.09/0.28 = 0.321$
  $p(3|3) = (0.3)(0.5)/\tilde{p}(3) = 0.15/0.28 = 0.536$

Table 1.4.4 The join probability distribution $p(x, y)$

$p(x,y)$	$y=1$	$y=2$	$y=3$	$p(x)$
$x=1$	.24	.12	.04	.4

$x=2$	.06	.15	.09	.3
$x=3$	.03	.12	.15	.3
$p(y)$	.33	.39	.28	1.0

Solution:

- The transition probabilities $p(y|x)$are:

Table 1.4.2: Forward transition probabilities $p(y|x)$

$p(y\|x)$	$y=1$	$y=2$	$y=3$	sum
$x=1$	0.6	0.3	0.1	1.0
$x=2$	0.2	0.5	0.3	1.0
$x=3$	0.1	0.4	0.5	1.0

- The inverse transition probabilities $p(x|y)$ are:

Table 1.4.3 Backward Condition

$p(x\|y)$	$x=1$	$x=2$	$x=3$	sum
$y=1$	0.727	0.182	0.121	1
$y=2$	0.231	0.385	0.308	1
$y=3$	0.107	0.321	0.536	1

Problem 2. For a continuous random variable, we have probability = density × size, or density =probability/size. Here, the probability can be a conditional probability. We can assume the continuous random variables are one-dimensional scalars.

(1) For continuous x and y, we can interpret $p(x)\Delta x$ to be the number of people in $(x, x+\Delta x)\cdot p(y|x)\Delta y$ be the

proportion of those in $(x, x+\Delta x)$who will move to $(y, y+\Delta y)$. Then among

Table 2: Transition probabilities

$p(y\|x)$	$y=1$	$y=2$	$y=3$
$x=1$			
$x=2$			
$x=3$			

Table 3: Inverse transition probabilities

$p(x\|y)$	$x=1$	$x=2$	$x=3$
$y=1$			
$y=2$			
$y=3$			

those who end up in $(y, y+\Delta y)$, the proportion of those who come from $(x, x+\Delta x)$ is $p(x|y)\Delta x$. Please show

$$
\begin{gathered}
p(x, y)=p(x) p(y|x) \\
\tilde{p}(y)=\int p(x, y)\, dx=\int p(x) p(y|x)\, dx \\
p(x, y)=p(x) p(y|x)=\tilde{p}(y) p(x|y) \\
p(x|y)=\frac{p(x, y)}{\tilde{p}(y)} \propto p(x) p(y|x)
\end{gathered}
$$

where the above proportionality is in terms of functions of x, where y is fixed, and x is the variable of the functions.

- $P(x)\Delta x$ = number of people in interval $(x, x+\Delta x)$
- $p(y|x)\Delta y$ = fraction of moving those people in $(x, x+\Delta x)$ to $(y, y+\Delta y)$
- Where,
  - $p(y|x)$ is the conditional density of y given x
  - Δy is the size of the target interval.
  - This explain the transition probabilities

For **Forward process**(noising): $p(x_t|x_{t-1})$, $x_t = x_{t-1} + e_t$ where $e_t \sim N(0, \sigma^2)$

- the conditional densities $p(y|x)$:

$$\begin{aligned} p(y|x) &= p(x_t|x_{t-1}) \\ &= \frac{p(x_t)p(x_{t-1}|x_t)}{\tilde{p}(x_{t-1})} \\ &\propto p(x)p(x_{t-1}|x_t) \\ &= p(x)p(y|x) \end{aligned}$$

For **Backward process**(denoising): $p(x_{t-1}|x_t)$ involves conditional densities

Using Bayes' rule $p(x|y)$:

$$\begin{aligned} p(x|y) &= p(x_{t-1}|x_t) \\ &= \frac{p(x_t|x_{t-1})p(x_{t-1})}{\tilde{p}(x)} \\ &\propto p(x_{t-1})p(x_t|x_{t-1}) \\ &= p(x)p(y|x) \end{aligned}$$

Marginalization:

$$\begin{aligned} \tilde{p}(y) &= \sum_x p(x, y) \\ &= \Sigma_x p(y|x)p(x) \\ &= \sum p(x|x_{t-1})p(x_{t-1}) \end{aligned}$$

For Chain rule:

$$\begin{aligned} p(x, y) &= p(x)p(y|x) \\ &= p(x_{t-1})p(x|x_{t-1}) \end{aligned}$$

(2)

Let $p(x)$ be the probability density function of random variable x.

Let $y = x + e$, where $e \sim N(0, \sigma^2)$ for a small σ^2, and e is independent of x,

i.e., $p(y|x) \sim N(x, \sigma^2)$, with

$$p(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2\sigma^2}(y-x)^2\right] \propto \exp\left[-\frac{1}{2\sigma^2}(y-x)^2\right]$$

Please show that for small σ^2, approximately, $p(x|y) \sim N(y + \sigma^2 \nabla logp(y), \sigma^2)$, where $\nabla f(x)$ is the derivative (slope, gradient) of the function f at x. You can use the equation $p(x|y) \propto p(x)p(y|x)$, and the first order Taylor expansion of $logp(x)$ around y.

1. Apply Bayes Rule,

$$p(x|y) \propto p(y|x)p(x)$$

Take log on both side:

$$\log p(x|y) = \log p(x) + \log p(y|x) + C(\text{normalization constant})$$

2. Substitute the Likelihood:

$$\log p(y|x) = -\frac{1}{2\sigma^2}(y - x)^2 + C$$

substitute $logp(y|x)$:

$$\log p(x|y) = \log p(x) - \frac{1}{2\sigma^2}(y - x)^2 + C$$

3. first order taylor expansion of $logp(x)$ around y:

$$f(x) \approx f(y) + (x - y) \cdot f'(y)$$

applied to log $p(x)$:

$$logp(x) \approx logp(y) + (x - y) \cdot \nabla logp(y)$$

where:

- $\log p(y)$ is the function value at y
- (x-y) is thw displacement from y
- $\nabla logp(y) = \frac{\partial d}{\partial x} logp(x)|_{x=y}$ is the Gradient (slope) at y
- ignoring the second order term due to the small σ^2

4. Substitute Taylor Expansion:

$$logp(x|y) \approx logp(y) + (x-y) \cdot \nabla logp(y) - \frac{1}{2\sigma^2}(y-x)^2 + C$$

Note that $(y-x)^2 = (x-y)^2$,

$$logp(x|y) \approx logp(y) + (x-y) \cdot \nabla logp(y) - \frac{1}{2\sigma^2}(x-y)^2 + C$$

since the $\log p(y)$ doesn't depend on x, so it is a constant C:

$$logp(x|y) \approx (x-y) \cdot \nabla logp(y) - \frac{1}{2\sigma^2}(x-y)^2 + C$$

5. factor out $1/(2\sigma^2)$

$$logp(x|y) \approx -\frac{1}{2\sigma^2}\left[(x-y)^2 - 2\sigma^2(x-y)\nabla logp(y)\right] + C$$

6. let $a = (x-y)$ and $b = \sigma^2 \nabla \log p(y)$:

$$a^2 - 2ab = (a-b)^2 - b^2$$

$$\begin{aligned}(x-y)^2 - 2\sigma^2(x-y)\nabla \log p(y) &= a^2 - 2ab \\ &= (a-b)^2 - b^2 \\ &= \left[(x-y) - \sigma^2\nabla \log p(y)\right]^2 - \left[\sigma^2 \nabla \log p(y)\right]^2 \\ &= \left[x - (y + \sigma^2 \nabla \log p(y))\right]^2 - \sigma^4[\nabla \log p(y)]^2\end{aligned}$$

7. Substitute all:

$$\begin{aligned}\log p(x|y) &= -\frac{1}{2\sigma^2}[x - (y + \sigma^2 \nabla \log p(y))]^2 - \sigma^4[\nabla \log p(y)]^2 + C \\ &= -\frac{1}{2\sigma^2}[x - (y + \sigma^2 \nabla \log p(y))]^2 + \frac{\sigma^2}{2}[\nabla \log p(y)]^2 + C \\ &= -\frac{1}{2\sigma^2}\left[x - (y + \sigma^2 \nabla \log p(y))\right]^2 + \underbrace{\frac{\sigma^2}{2}[\nabla \log p(y)]^2}_{constant} + C \\ &= -\frac{1}{2\sigma^2}[x - (y + \sigma^2 \nabla \log p(y))]^2 + C\end{aligned}$$

8. Identify the Gaussian form

A Gaussian $N(\mu, \sigma^2)$ PDF :

$$\log p(x) = -\frac{(x-\mu)^2}{2\sigma^2} + C$$

So, substitute $u = y + \sigma^2 \nabla \log p(y)$:

$$\begin{aligned} logp(x|y) &= -\frac{1}{2\sigma^2}\left[x - \underbrace{(y + \sigma^2 \nabla \log p(y))}_{u}\right]^2 + C \\ &= -\frac{(x-\mu)^2}{2\sigma^2} + C \end{aligned}$$

Therefore,

$$\boxed{p(x|y) \sim N(y + \sigma^2 \nabla \log p(y), \sigma^2)}$$

Problem 3

(1) Let $x_0 \sim p_0(x)$. Let $x_t = x_{t-1} + e_t$, where $e_t \sim N(0, \sigma^2)$ for a small σ^2, and e_t are independent for different $t, t = \{1, ..., T\}$, so that for large T , approximately $x^T \sim N(0, T\sigma^2)$. Let p_t be the marginal density of x_t. Based on the previous problem, explain that for small σ^2, approximately,

$$p(x_{t-1}|x_t) \sim N(x_t + \sigma^2 \nabla \log p_{t-1}(x_t), \sigma^2).$$

Explain that we can estimate the score function $\nabla \log p_{t-1}(x_t)$ using a neural network $s_\theta(x_t, t)$ by minimizing the least squares loss $L(\theta) = E_{t,x_0,x_{t-1},x_t}[(x_{t-1} - (x_t + \sigma^2 s_\theta(x_t, t)))^2]$,where E_{t,x_0,x_{t-1},x_t} can be approximated by averaging over $t = \{1, ..., T\}$ and (x_0, x_{t-1}, x_t).

Given

$x_0 \sim p_0(x)$
$x_t = x_{t-1} + e_t, \quad e_t \sim \mathcal{N}(0, \sigma^2),$
independent for $t = 1, ..., T x_T \sim \mathcal{N}(0, T\sigma^2)$
$(\text{approximately, for large } T)$

Forward diffusion process:

1. Apply Bayes Rule:

from the backward condition, $p(x_{t-1}|x_t)$ using Bayes's rule

$$p(x_{t-1}|x_t) = \frac{p(x_t|x_{t-1})p(x_t - 1)}{p_t(x_t)}$$

where:

- $p(x_t|x_{t-1})$ = forward noising process (likelihood)
- p(x_{t-1}|x_t) = posterior (reverse / denoising step)
- $p(x_{t-1}) = p_{t-1}(x_{t-1})$ = marginal at time $t-1$ (prior)
- $p_t(x_t)$ = evidence/marginal at t

2. Gaussian Forward Kernel

given: $x_t = x_{t-1} + e_t, \text{ where } e_t \sim N(0, \sigma^2)$

$$p(x_t|x_{t-1}) = N(x_t; x_{t-1}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_t - x_{t-1})^2}{2\sigma^2}\right)$$

3. Take Logarithm of Bayes' rule

$$\begin{aligned}\log p(x_{t-1}|x_t) &= log\left[\frac{p(x_t|x_{t-1}) \cdot p_{t-1}(x_{t-1})}{p_t(x_t)}\right] \\ &= \log p(x_t|x_{t-1}) + \log p_{t-1}(x_{t-1}) - \underbrace{\log p_t(x_t)}_{constant} \\ &= \underbrace{\log p(x_t|x_{t-1})}_{likelihood} + \underbrace{\log p_{t-1}(x_{t-1})}_{priorterm} - C\end{aligned}$$

For $\log p(x_t|x_{t-1})$ likelihood term:

$$\log p(x_t|x_{t-1}) = \log\left[\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x_t - x_{t-1})^2}{2\sigma^2}\right)\right] \tag{1}$$

$$= \underbrace{-\frac{1}{2}\log(2\pi\sigma^2)}_{constant} - \frac{(x_t - x_{t-1})^2}{2\sigma^2} \tag{2}$$

$$= -\frac{(x_t - x_{t-1})^2}{2\sigma^2} + C_1 \tag{3}$$

Since we have small noise $e_t \sim N(0, \sigma^2)$ to get from x_{t-1} to x_t, we expect $x_{t-1} \approx x_t$

For prior term logp_t-1(x_t-1) with Taylor expansion:

$$\log p_{t-1}(x_{t-1}) \approx \log p_{t-1}(x_t) + (x_{t-1} - x_t)\nabla \log p_{t-1}(x_t) \tag{4}$$

Apply (3) and (4) into (5):

$$\log p(x_{t-1}|x_t) = \log p(x_t|x_{t-1}) + \log p_{t-1}(x_{t-1}) - C \tag{5}$$

$$= \underbrace{-\frac{(x_t - x_{t-1})^2}{2\sigma^2}}_{likelihood} + \underbrace{\log p_{t-1}(x_{t-1})}_{priorterm} - C \tag{6}$$

$$= -\frac{(x_t - x_{t-1})^2}{2\sigma^2} + \underbrace{\log p_{t-1}(x_t)}_{const} + (x_{t-1} - x_t)\nabla \log p_{t-1}(x_t) \tag{7}$$

$$= -\frac{(x_t - x_{t-1})^2}{2\sigma^2} + (x_{t-1} - x_t)\nabla \log p_{t-1}(x_t) + C \tag{8}$$

$$= -\frac{1}{2\sigma^2}[(x_t - x_{t-1})^2 + 2\sigma^2(x_{t-1} - x_t)\nabla \log p_{t-1}(x_t)] + C \tag{9}$$

Substitute $u = (x_{t-1} - x_t)$ and $b = \sigma^2\nabla \log p_{t-1}(x_t)$,

$$\tag{10}$$

$$= -\frac{1}{2\sigma^2}[u^2 + 2u \cdot b] + C \tag{11}$$

$$= -\frac{1}{2\sigma^2}[(u - b)^2 - b^2] + C \tag{12}$$

by taking part of [(u-b)^-b^2]:

$$
\begin{aligned}
u^2 - 2\sigma^2 u \cdot \nabla \log p_{t-1}(x_t) &= \left[u - \sigma^2 \nabla \log p_{t-1}(x_t)\right]^2 - \left[\sigma^2 \nabla \log p_{t-1}(x_t)\right]^2 \\
&= \left[(x_{t-1} - x_t) - \sigma^2 \nabla \log p_{t-1}(x_t)\right]^2 - \left[\sigma^2 \nabla \log p_{t-1}(x_t)\right]^2 \\
&= \left[x_{t-1} - (x_t + \sigma^2 \nabla \log p_{t-1}(x_t))\right]^2 - \left[\sigma^2 \nabla \log pt - 1(x_t)\right]^2
\end{aligned}
$$

Plug back into (12)

$$
\log p(x_{t-1} \mid xt) = -\frac{1}{2\sigma^2}\left[x_{t-1} - (x_t + \sigma^2 \nabla \log p_{t-1}(x_t))\right]^2 + \underbrace{\frac{\sigma^2}{2}[\nabla \log p_{t-1}(x_t)]^2 + C}_{const w.r.t. x_{t-1}}
$$

Therefore it proof of gaussian distribution,

$$
p(x_{t-1}|x_t) \sim N(x_t + \sigma^2 \nabla \log p_{t-1}(x_t), \sigma^2)
$$

(3) After learning $s_\theta(x, t)$, explain that we can generate a new x_0 by sampling $x_T \sim N(0, T\sigma^2)$, and iterating $x_{t-1} = x_t + \sigma^2 s_\theta(x_t, t) + \tilde{e}_t$, where $\tilde{e}_t \sim N(0, \sigma^2)$ independently, for $t = \{T, ..., 1\}$. This is the reverse denoising process.

1. Start: Pure Noise: $x_T \sim N(0, T\sigma^2)$
    a. since large T and small σ^2, $x_T \approx N(0, T\sigma^2)$
2. Iterate: reverse steps of $t = [T, T-1, \ldots, 2, 1]$

    From (1) and (2) we derived:

    $$
    p(x_{t-1}|x_t) \sim N(x_t + \sigma^2 \nabla \log p_{t-1}(x_t), \sigma^2)
    $$

    We approximate the score function:

    $$
    \nabla \log p_{t-1}(x_t) \approx s_\theta(x_t, t)
    $$

    Therefore,

    $$
    p_\theta(x_{t-1}|x_t) = N(x_t + \sigma^2 s_\theta(x_t, t), \sigma^2)
    $$

    To sample $x_{t-1} \sim p_\theta(x_{t-1}|x_t)$:

    For $X \sim N(\mu, \sigma^2)$: $X = \mu + \sigma\epsilon, \epsilon \sim N(0, 1)$

$$x_{t-1} = \underbrace{x_t + \sigma^2 s_\theta(x_t, t)}_{\mu} + \underbrace{\sigma\tilde{\epsilon}}_{\sigma \cdot noise}$$

3. End: Clean data x_0

$$\boxed{\begin{aligned}
&\text{Initialize: } x_T \sim \mathcal{N}(0, T\sigma^2) && (13)\\
&\text{For } t = T, T-1, ..., 2, 1: && (14)\\
&\quad x_{t-1} = \underbrace{x_t}_{\text{current state}} + \underbrace{\sigma^2 s_\theta(x_t, t)}_{\text{denoising direction}} + \underbrace{\tilde{e}t}_{\text{fresh noise}} && (15)\\
&\quad \text{where } \tilde{e}t \sim \mathcal{N}(0, \sigma^2) \text{ independently} && (16)\\
&\text{Output: } x_0 \sim p\text{data}(x) && (17)
\end{aligned}}$$

(4) Explain that we can also iterate $x_{t-2} = x_t + \sigma^2 s_\theta(x_t, t)$, which is a deterministic denoising process.

The SDE(Stochastic) vs ODE(Deterministic)

$$\begin{aligned}
&\text{Stochastic: } x_{t-1} = x_t + \sigma^2 s_\theta(x_t, t) + \tilde{e}_t \quad (\text{SDE}) && (18)\\
&\text{Deterministic: } x_{t-1} = x_t + \sigma^2 s_\theta(x_t, t) \quad (\text{ODE}) && (19)\\
&\text{t-2 Phenomenon: Deterministic update } \approx \text{ landing at } x_{t-2} && (20)\\
&\text{Enables: Step skipping, faster sampling, exact inversion} && (21)\\
&\text{Trade-off: Speed \& control vs diversity \& robussness} && (22)
\end{aligned}$$

Comparison: Stochastic vs Deterministic

Aspect	Stochastic	Deterministic
Update rule	$x_{t-1} = x_t + \sigma^2 s_\theta(x_t, t) + \tilde{e}_t$	$x_{t-1} = x_t + \sigma^2 s_\theta(x_t, t)$
Mathematical form	SDE (diffusion process)	ODE (flow)
Reproducibility	Different each run	Same output for same x_T
Diversity	High (from intermediate noise)	Lower (only from x_T)
Speed	Slower (must take small steps)	Faster (can skip steps)
Inversion	Not exact	Exact (reversible ODE)
Variance	Higher	Lower
Textbook reference	Chapter 6.2-6.9	Chapter 6.10-6.11

Key Objective

Use Stochastic when:

- You want maximum diversity in generated samples
- Training a model (need noise for proper gradients)
- Exploring the space of possible outputs
- Don't care about exact reproducibility

Use Deterministic when:

- You need fast generation (10-50 step vs 1000)
- You want exact reconstruction (image editing)
- You need reproducible results (debugging, demos)
- You're doing latent interpolation (smooth transition)

(5) Explain the above in terms of the movements of 1 billion particles on the real line. In particular, in (4), why deterministic movements reverse random noising perturbations n as far as the overall marginal distribution is concerned?

The key insight is that we care about **population distributions**, not individual particle trajectories. During forward noising, each of 1 billion particles takes its own random walk - particle #1 might go $0 \to 0.3 \to -0.1 \to 0.5$, while particle #2 starting at the same position goes $0 \to -0.2 \to 0.4 \to 0.1$ However, the **marginal distribution** $p_t(x)$ - the histogram showing how many particles are at each location - evolves predictably from concentrated (data) to spread out (Gaussian noise). To reverse this process, we don't need each particle to retrace its exact random path; we only need the population histogram to compress back to the original shape. The score function $\nabla \log p_t(x)$ encodes where particles should move based on population density (pointing toward high-density regions), and with 1 billion particles, the law of large numbers makes this population-level guidance effectively deterministic.

Both stochastic (SDE) and deterministic (ODE) methods achieve the same population evolution through a beautiful compensation: the SDE uses **strong drift** ($\sigma^2 \nabla \log p_t$) to push particles toward high density, but adds **random jitter** that spreads them back out - these two effects balance to produce moderate compression. The ODE uses **gentle drift** ($\frac{\sigma^2}{2} \nabla \log p_t$ - exactly half the speed) with
no jitter, achieving the same moderate compression rate. This factor-of-2 difference is not

arbitrary - it comes from stochastic calculus (Itô's lemma) ensuring both methods transport the same marginal distributions $p_t(x)$. Think of it like traffic: cars either drive fast but randomly change lanes (SDE), or drive at exactly half speed with perfect coordination (ODE) - both produce identical traffic density evolution. This equivalence enables fast sampling (ODE can skip steps), exact image editing (ODE is reversible), and controllable generation (ODE is reproducible), which is why modern diffusion models like Stable Diffusion use deterministic sampling by default

Problem 4

Instead of predicting x_{t-1} from x_t, we can also predict x_0 directly from x_t to learn $s_\theta(x_t, t)$, by minimizing

$$L(\theta) = E_{x_0,t,x_t}[(x_0 - (x_t + t\theta^2 s_\theta(x_t, t)))^2],$$

where $x_t = x_0 + \varepsilon_t, \varepsilon_t \sim N(0, t\sigma^2)$. x_0 is a better target than x_{t-1} because x_0 is the clean version without noise.

$$Let\ \varepsilon_\theta(x, t) = -t\sigma^2 s_\theta(x, t),$$

we can write

$$L(\theta) = E_{x_0,t,\varepsilon_t}[(\varepsilon_t - \varepsilon_\theta(x_0 + \varepsilon_t, t))^2].$$

That is, we learn an ε_θ network to estimate the noise εt from the noisy observation $x_0 + \varepsilon_t$.

After estimating $\varepsilon_\theta(x, t)$, please derive the backward denoising process, both the stochastic version and the deterministic version.

Note: In real implementation, people also introduce some scaling coefficients such as $x_t = \sqrt{\alpha_t x_{t-1}} + \sqrt{\beta_t e_t}$. But this is less essential.

$$\text{Training: } L(\theta) = \mathbb{E}|\epsilon_t - \epsilon_\theta(x_0 + \epsilon_t, t)|^2 \quad (23)$$

$$\text{Stochastic sampling: } x_{t-1} = x_t - \frac{1}{t}\epsilon_\theta(x_t, t) + \sigma z \quad (24)$$

$$\text{Deterministic sampling: } x_{t-1} = x_t - \frac{1}{t}\epsilon_\theta(x_t, t) \quad (25)$$

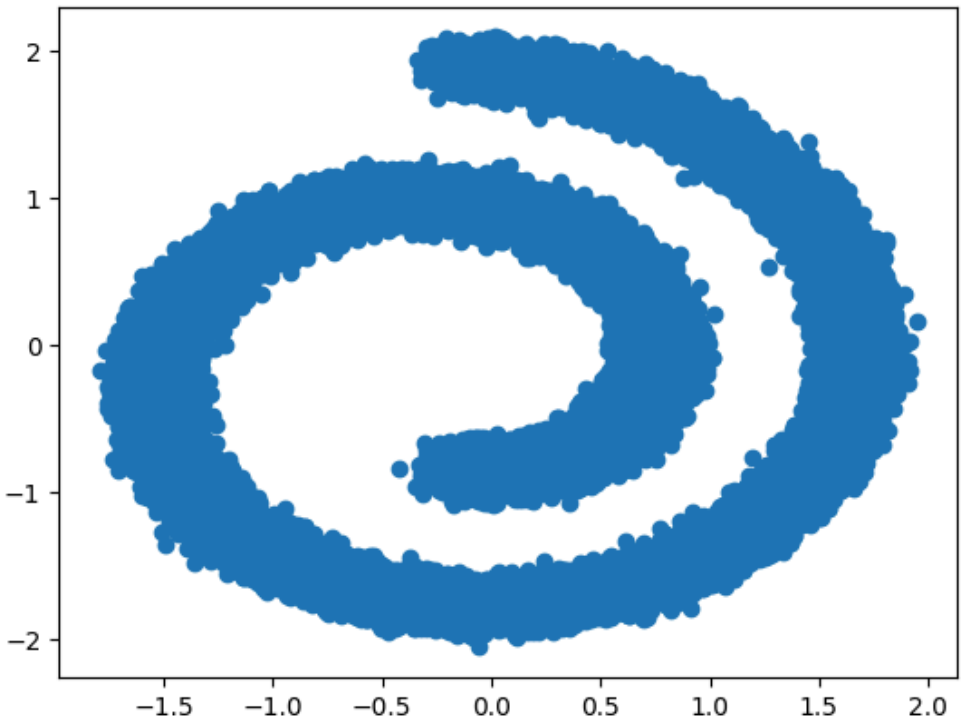
$$\text{Clean prediction: } \hat{x}0 = x_t - \epsilon\theta(x_t, t) \quad (26)$$

Problem 5

1. Architecture Components: (*Section 6.5 in the book)*
    - **UNet backbone**: Encoder-decoder with skip connection
    - **Time Embedding**: Sinusoidal encoding of timestamp t
    - **Output**: Noise prediction $\epsilon_\theta(x_t, t)$
2. **Training loop**: (*Section 6.8 in book*)
    - Sample x_0 from data (e.g. 2D point clouds)
    - Sample $t \sim \text{Unfiform}(1, T)$
    - Sample noise $\epsilon \sim N(0, I)$
    - Crease noisy sample: $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{(1 - \bar{\alpha}_t)} \epsilon$
    - **Predict**: $\hat{\epsilon} = \epsilon_\theta(x_t, t)$
    - **Loss fn**: $MSE(\epsilon, \hat{\epsilon})$
3. **Sampling/Generation**:
    - start from $x_t \sim N(0, I)$
    - Iteratively denoise using learned ϵ_θ
    - Either DDPM (stochastic) or DDIM (deterministic)
4. **Variance Schedule**: (*Section 6.17 in book*)
    - β_t schedule: linear, cosine, and so on
    - $\bar{\alpha}_t = \prod_{i=1}^{t} (1 - \beta_i)$
    - controls noise level at each step

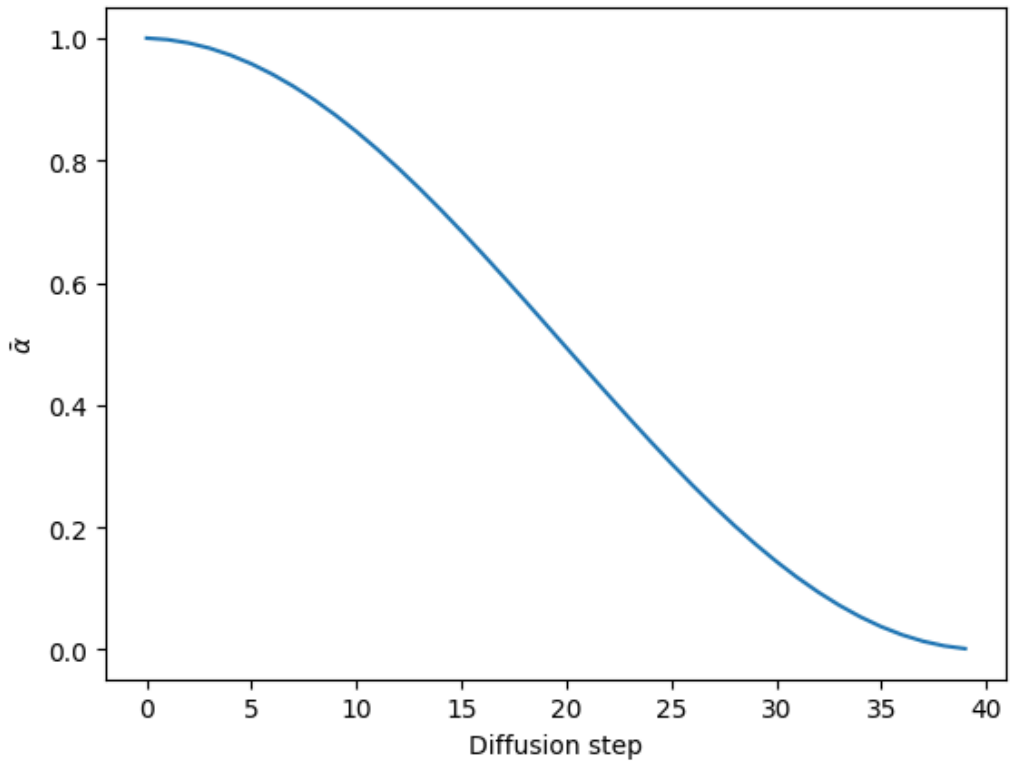
Output from the codebase

- Plot 5.1,



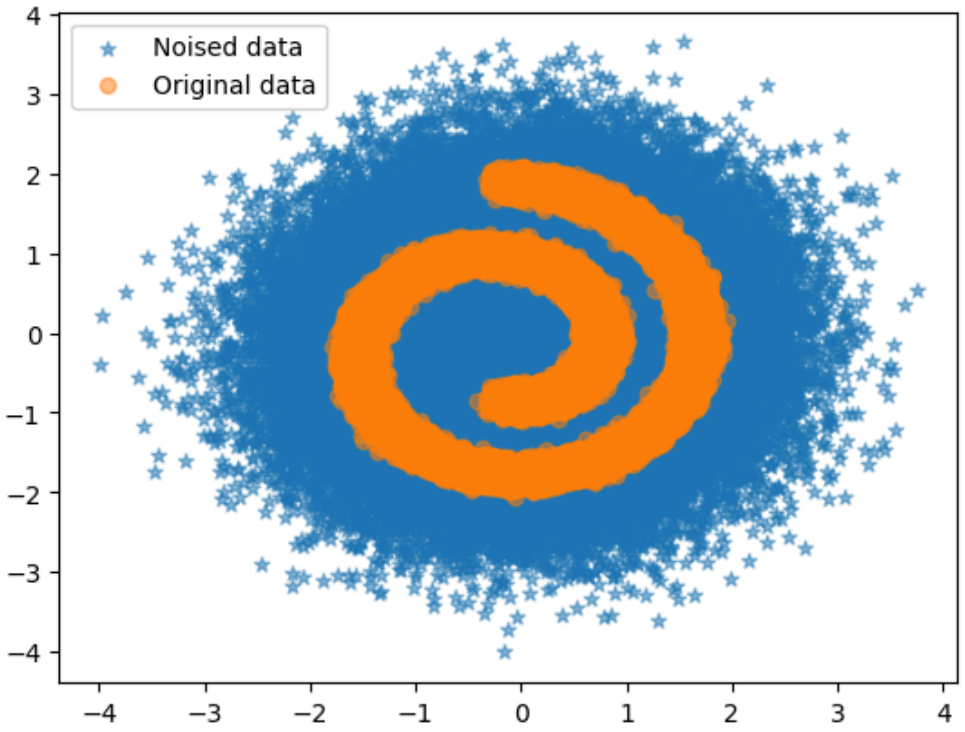
It displays a 2D scatter plot of the training data. This data is generated using `make_swiss_roll` from `sklearn.datasets`, which typically produces a 3D spiral shape resembling a Swiss roll. However, in this code, it's projected onto a 2D plane by selecting only the first and third components (`x[:, [0, 2]]`). The data is then standardized (mean-centered and scaled to unit variance). The scatter plot visually represents the distribution of this 2D `swiss roll` data, which serves as the target distribution for the diffusion model to learn.

- Plot 5.2. Diffusion model hyperparameters



It shows the value of $\bar{\alpha}$ (bar-alpha) across the diffusion steps. In diffusion models, represents the cumulative product of $(1 - \beta_t)$ values up to a given time / step t. It essentially indicates how much of the original signal remains at each diffusion step. As expected, you'll observe that $\bar{\alpha}$ decreases as the number of diffusion steps increases, meaning that more noise is added and the original data signal is progressively attenuated throughout the diffusion process. (https://arxiv.org/pdf/2102.09672)

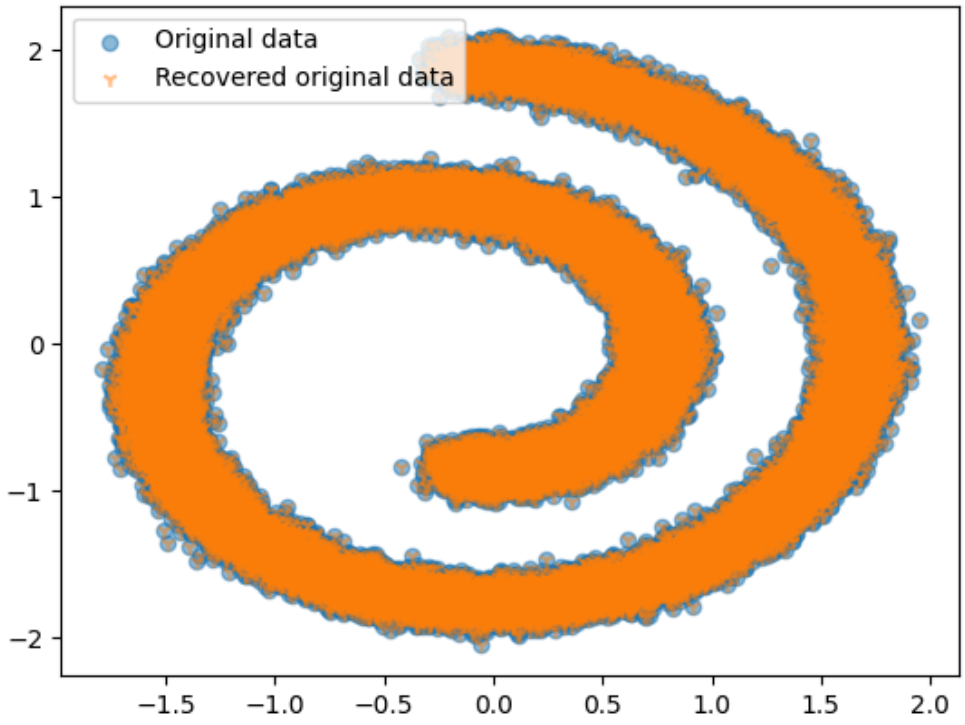
- Plot 5.3, Test noising the training data-1



It visualizes the effect of the `noise` function, which simulates a step in the diffusion process. It shows two distinct sets of points:

1. **Original Data**: These are the clean data points, representing the underlying distribution that the model aims to learn and generate. You can see the `swiss roll` shape from earlier in the notebook.
2. **Noised Data (blue asterisks)**: These points are generated by adding noise to the original data according to the diffusion schedule at a specific `noiselevel` (set to 20 in this case). You can observe how the original structure becomes blurred and more spread out, demonstrating how the diffusion process gradually transforms clean data into random noise. This is a key step in understanding how the diffusion model works, as it learns to reverse this noise-adding process.

- Plot 5.4, Test noising the training data -2



It demonstrates the perfect recovery of the original data when the exact noise component is known. It shows:

1. **Original data (blue)**: These are the initial clean data points from the 'swiss roll' distribution.
2. **Recovered original data (orange '1' markers)**: These points are obtained by analytically 'denoising' the previously noised data. Since the exact noise (`eps`) that was added is known in this controlled test, the original data can be precisely reconstructed. This visually confirms the mathematical relationship used in the diffusion process, where knowing the noise allows for the reversal of the noising step.

- Output 5.1, Training Epoch and Loss fn

```
Epoch 0 loss = 0.7389426827430725
Epoch 1 loss = 0.5305495858192444
Epoch 2 loss = 0.5119237303733826
Epoch 3 loss = 0.4977395832538605
Epoch 4 loss = 0.4868130385875702
Epoch 5 loss = 0.484809547662735
```

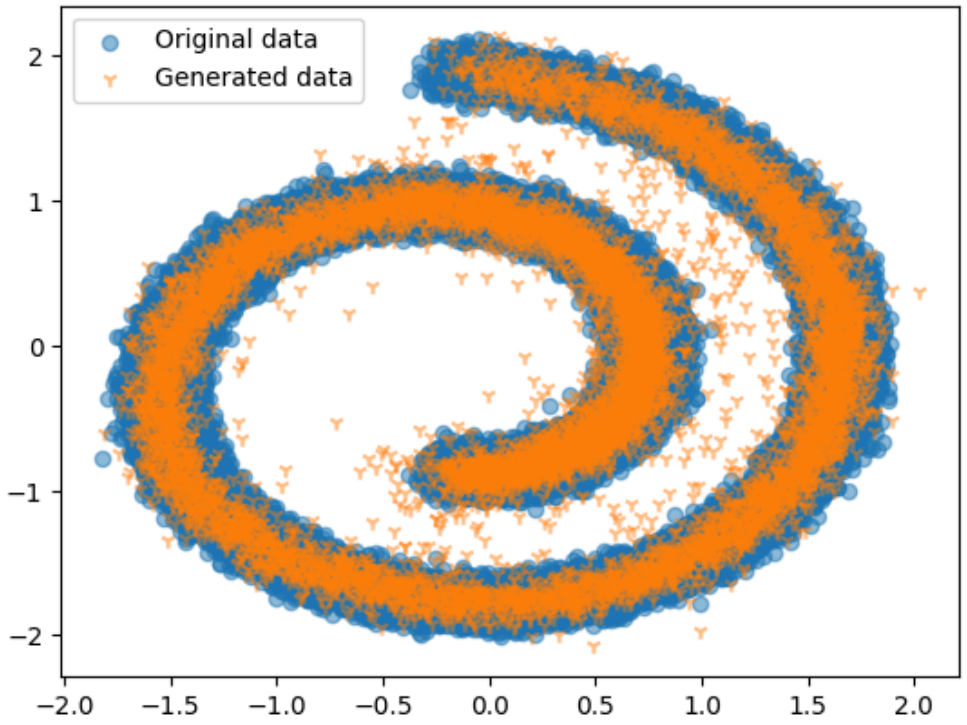
```
Epoch 6 loss = 0.480057954788208

....

Epoch 99 loss = 0.4396243393421173
```

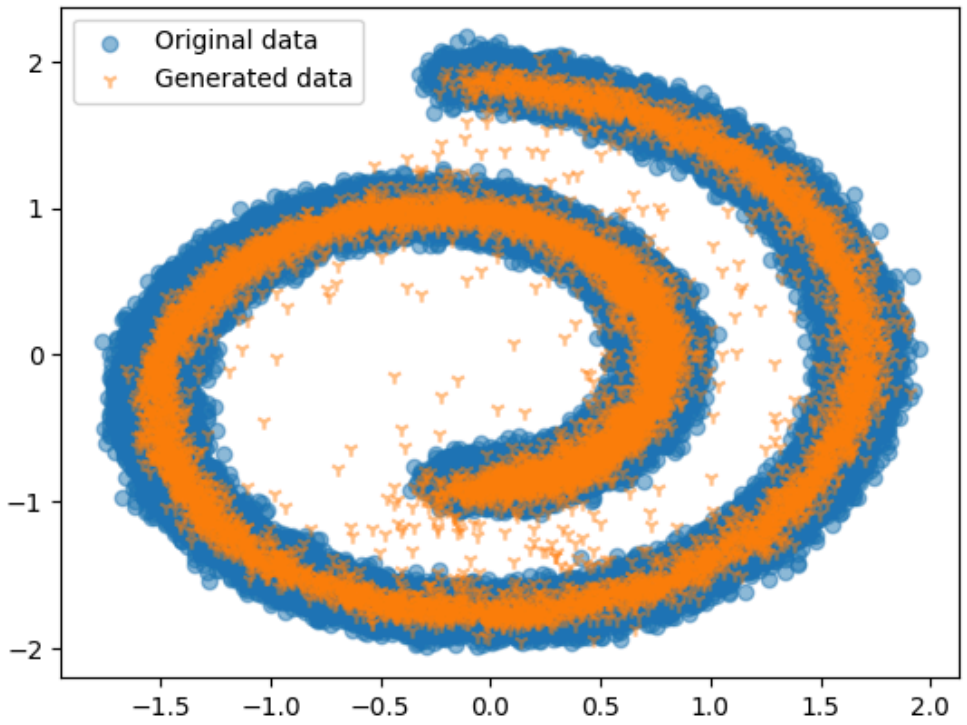
The code trains a diffusion model to predict the noise component added to data points at various timesteps. It iterates over a specified number of epochs, processing data in mini-batches. For each batch, random noise is added to the data at a randomly selected diffusion timestep. The model then attempts to predict this added noise. The **Mean Squared Error** (MSE) between the predicted and actual noise is calculated as the loss, which is then used by the Adam optimizer to update the model's parameters through backpropagation. A learning rate scheduler is also employed to adjust the learning rate over time, with the overall goal of minimizing the error in noise prediction, thereby enabling the model to effectively reverse the diffusion process during sampling.

- Plot 5.5, Sampler with DDPM denoise generation



This demonstrates the generative power of the trained diffusion model by calling the `sample_ddpm` function to create 10,000 new data points, starting from random noise and iteratively refining them using the learned denoising process. The resulting graph then visually compares these `Generated data` (orange '1' markers) with the `Original data` (blue points) from the `make_swiss_roll` dataset. This comparison is crucial for assessing how well the model has learned the underlying data distribution; a successful model will produce generated samples that closely mimic the shape, density, and structure

- Plot 5.6, alternative sampler with DDPM for x_0 → x_{t-1}



This graph defines an alternative sampling function, `sample_ddpm_x0`, which implements a different approach to the DDPM (Denoising Diffusion Probabilistic Models) sampling algorithm. Unlike the previous `sample_ddpm` function that directly predicts the noise to step back, this method first predicts the original, clean sample x_0 using an intermediate calculation based on the predicted noise. Then, it uses this predicted x_0 to determine the previous noisy sample x_{t-1}. This approach is common in implementations like **HuggingFace Diffusers** and allows for a slightly different formulation of the denoising process, especially concerning the variance added in each step. The goal remains the same: to generate new data samples by iteratively denoising a starting random noise vector.