Contents

1. Generative Models Comparison

1(a) Tractable Inverse Mapping

Question

Suppose we care about having a model with a tractable inverse mapping from data space to latent space. Which models provide this property by design?

Answer

Flow-based models (Normalizing Flows).

An "inverse mapping from data space to latent space" means: given a data point $\mathbf{x}$, we want to compute the corresponding latent code $\mathbf{z}$ *exactly* and *efficiently* (in closed form, no iterative optimization).
Flow-based models achieve this by constructing the generator as a **composition of invertible transformations**:

$$f = f_K \circ f_{K-1} \circ \cdots \circ f_1$$

Each f_i is a bijection (one-to-one and onto), typically designed so that both f_i and f_i^{-1} are computationally efficient. Because the composition of bijections is itself a bijection, the full model is invertible. Therefore we can compute:

$$\mathbf{z} = f^{-1}(\mathbf{x}) = f_1^{-1} \circ f_2^{-1} \circ \cdots \circ f_K^{-1}(\mathbf{x})$$

in a single pass through the network - no approximation, no optimization, no sampling.

Why not the other model classes?

- **VAEs:** The encoder $q_\phi(\mathbf{z}|\mathbf{x})$ provides only an *approximate* inverse. It outputs a distribution (not a point), and that distribution is trained to approximate the intractable true posterior $p_\theta(\mathbf{z}|\mathbf{x})$. There is always a variational gap.
- **GANs:** The generator maps $\mathbf{z} \to \mathbf{x}$, but there is *no built-in* mechanism to go from $\mathbf{x} \to \mathbf{z}$. One can train a separate encoder or use optimization-based inversion (GAN inversion), but this is not "by design."
- **Autoregressive models:** These factorize $p(\mathbf{x})$ directly as a product of conditionals $\prod_i p(x_i|x_{<i})$. There is no explicit latent space $\mathbf{z}$, so the question of an inverse mapping does not apply.

1(b) Non-Likelihood-Based Divergence

Question

Suppose we want to minimize a divergence that is not likelihood-based, but instead depends on distinguishing real and fake samples. Which class of models does this correspond to?

Answer

Generative Adversarial Networks (GANs).

Likelihood-based models (VAEs, flows, autoregressive models) are trained by maximizing

$\log p_\theta(\mathbf{x})$ for observed data - i.e., they directly fit a density. GANs take a fundamentally different approach. Instead of computing any density, GANs use a **discriminator** network $D(\mathbf{x})$ whose job is to tell apart real data from generated data. The GAN objective is:

$$\min_G \max_D \; V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

What divergence does this minimize? Goodfellow et al. (2014) showed that when the discriminator is optimal, the generator's objective reduces to minimizing the **Jensen-Shannon Divergence** between p_{data} and p_G:

$$\text{JSD}(p_{\text{data}} \| p_G) = \tfrac{1}{2}\, D_{\text{KL}}(p_{\text{data}} \| m) + \tfrac{1}{2}\, D_{\text{KL}}(p_G \| m), \qquad m = \tfrac{1}{2}(p_{\text{data}} + p_G)$$

This divergence is estimated entirely through the discriminator's ability to distinguish real from fake - *no likelihood is ever computed.* This makes GANs an **implicit generative model**: they define a sampling procedure (pass noise through G) but never explicitly specify a density $p_G(\mathbf{x})$.

1(c) Exact Log-Likelihood Gradient

Question

Suppose we want to compute the exact log-likelihood gradient with respect to model parameters for a given datapoint without using variational approximations. Which of these models allow this? Briefly explain.
(a) Autoregressive models (b) Latent variable models (c) Flow-based models (d) All of the above

Answer

Answer: (a) Autoregressive models and (c) Flow-based models.

The question asks: can we compute $\nabla_\theta \log p_\theta(\mathbf{x})$ *exactly* - without any approximation or bound?

(a) Autoregressive Models - YES.
These decompose the joint via the chain rule of probability:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(x_i \mid x_1, \ldots, x_{i-1})$$

Taking the log:

$$\log p_\theta(\mathbf{x}) = \sum_{i=1}^{n} \log p_\theta(x_i \mid x_1, \ldots, x_{i-1})$$

Each conditional $p_\theta(x_i \mid x_{<i})$ is modeled by a neural network (e.g., an RNN, Transformer, or masked network like MADE). Each term is a standard neural network output (e.g., a softmax probability or a Gaussian density). Therefore $\log p_\theta(\mathbf{x})$ is a **finite, differentiable sum of tractable terms**, and we can compute its exact gradient via backpropagation. No approximation is needed at any step.

(c) Flow-based Models - YES.

The change-of-variables formula gives the exact log-likelihood:

$$\log p_\theta(\mathbf{x}) = \log p_Z(f_\theta^{-1}(\mathbf{x})) + \log \left| \det \frac{\partial f_\theta^{-1}}{\partial \mathbf{x}} \right|$$

The first term is the log-density of the base distribution (e.g., a standard Gaussian) evaluated at the latent point $\mathbf{z} = f_\theta^{-1}(\mathbf{x})$ - this is trivially tractable. The second term is the log-absolute-determinant of the Jacobian of the inverse transformation - flow architectures (coupling layers, autoregressive flows) are specifically designed so this determinant is tractable (e.g., triangular Jacobians). Both terms are differentiable with respect to θ, so $\nabla_\theta \log p_\theta(\mathbf{x})$ is exact.

(b) Latent Variable Models (e.g., VAEs) - NO.
The marginal likelihood involves an intractable integral:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})\, p(\mathbf{z})\, d\mathbf{z}$$

This integral cannot be computed in closed form for complex decoders. VAEs approximate it via the Evidence Lower Bound (ELBO), introducing a variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$. Because of this approximation, we optimize a *bound* on $\log p_\theta(\mathbf{x})$, not the exact quantity.

Conclusion: Since (b) is excluded, (d) "all of the above" is also excluded. The correct answer is **(a) and (c)**.

1(d) Insufficient Encoder Capacity in VAEs

Question

VAEs optimize:

$$\log p_\theta(\mathbf{x}) = \text{ELBO} + D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \,\|\, p_\theta(\mathbf{z}|\mathbf{x}))$$

If the encoder has insufficient capacity, what effect does this have on generative quality?

Answer

The identity above is an *exact* decomposition. It tells us that:

$$\text{ELBO} = \log p_\theta(\mathbf{x}) - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \,\|\, p_\theta(\mathbf{z}|\mathbf{x}))$$

Since KL divergence is always ≥ 0, the ELBO is always $\leq \log p_\theta(\mathbf{x})$. The "gap" between the ELBO and the true log-likelihood is *exactly* $D_{\text{KL}}(q_\phi \| p_\theta)$.
If the encoder $q_\phi(\mathbf{z}|\mathbf{x})$ has **insufficient capacity** (e.g., it is restricted to a simple diagonal Gaussian while the true posterior is multimodal or has complex correlations), it **cannot closely approximate** $p_\theta(\mathbf{z}|\mathbf{x})$. This means $D_{\text{KL}}(q_\phi \| p_\theta)$ remains **persistently large**.

Consequences for generative quality:

1. **Loose ELBO (poor training signal):** A large KL gap means the ELBO substantially underestimates $\log p_\theta(\mathbf{x})$. Since we optimize the ELBO (not the true likelihood), the decoder receives a distorted training signal. It is being trained against a misspecified posterior, which limits how well it can learn to generate.
2. **Poor latent representations:** The approximate posterior fails to capture the true latent structure. For example, if the true posterior for a face image has separate modes for "smiling" and "neutral," a unimodal Gaussian encoder will average over

them, producing a latent code that represents neither expression well. This means the decoder never gets sharp, informative latent codes during training.

3. **Blurry / low-quality generations:** Because the decoder is trained with "smeared out" latent codes, it learns to produce outputs that average over multiple possible data points. This manifests as blurry images, muffled speech, or other artifacts that lack fine-grained detail.
4. **Posterior collapse:** In extreme cases, the encoder may "give up" entirely and output $q_\phi(\mathbf{z}|\mathbf{x}) \approx p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ regardless of the input $\mathbf{x}$. When this happens, $\mathbf{z}$ carries *zero information* about $\mathbf{x}$, and the decoder degenerates into a unconditional model that ignores the latent variable. This is known as **posterior collapse** or **KL vanishing**, and results in poor sample diversity.

2. Latent Variable Model and ELBO

Question

Consider a latent variable model with joint $p(\mathbf{x}, z)$ that samples only two images $\{x^{(1)}, x^{(2)}\}$, where $x^{(i)} \in \{0,1\}^{784}$. The latent variable is scalar $z \in \mathbb{R}$. The model is:

$$p(z) = \mathcal{N}(z;\, 0, 1), \qquad p(\mathbf{x} \mid z) = \begin{cases} 1 & \text{if } z \geq 0 \text{ and } \mathbf{x} = x^{(1)} \\ 1 & \text{if } z < 0 \text{ and } \mathbf{x} = x^{(2)} \\ 0 & \text{otherwise} \end{cases}$$

2(a) Compute $\log p(x^{(1)})$

Question

Compute $\text{like}(x) := \log p(x)$. What is $\text{like}(x^{(1)})$?

Answer

To find the marginal likelihood $p(x^{(1)})$, we marginalize out the latent variable z by integrating over the entire real line:

$$p(x^{(1)}) = \int_{-\infty}^{\infty} p(x^{(1)} \mid z)\, p(z)\, dz$$

Step 1: Identify where the integrand is non-zero. From the model definition, $p(x^{(1)} \mid z) = 1$ only when $z \geq 0$, and $p(x^{(1)} \mid z) = 0$ when $z < 0$. So:

$$p(x^{(1)}) = \int_{0}^{\infty} 1 \cdot \mathcal{N}(z;\, 0, 1)\, dz + \int_{-\infty}^{0} 0 \cdot \mathcal{N}(z;\, 0, 1)\, dz = \int_{0}^{\infty} \mathcal{N}(z;\, 0, 1)\, dz$$

Step 2: Evaluate. The integral $\int_0^\infty \mathcal{N}(z;\, 0, 1)\, dz$ is the probability that a standard normal random variable is non-negative. By the **symmetry** of $\mathcal{N}(0, 1)$ about zero:

$$P(z \geq 0) = 0.5$$

Step 3: Take the log:

$$\boxed{\log p(x^{(1)}) = \log(0.5) = -\ln 2 \approx -0.693}$$

Key Insight

By identical reasoning, $p(x^{(2)}) = \int_{-\infty}^{0} \mathcal{N}(z; 0, 1)\, dz = 0.5$, so $\log p(x^{(2)}) = -\ln 2$ as well. The model assigns equal probability to both images.

2(b) Posterior $p(z \mid x^{(1)})$

Question

Compute the posterior distribution $p(z \mid x^{(1)})$ explicitly.

Answer

We apply Bayes' theorem:

$$p(z \mid x^{(1)}) = \frac{p(x^{(1)} \mid z)\, p(z)}{p(x^{(1)})}$$

Step 1: Substitute the known quantities.

- $p(x^{(1)} \mid z) = \mathbf{1}[z \geq 0]$ (equals 1 if $z \geq 0$, else 0)
- $p(z) = \mathcal{N}(z;\, 0, 1)$
- $p(x^{(1)}) = 0.5$ (from part (a))

Step 2: Plug in:

$$p(z \mid x^{(1)}) = \frac{\mathbf{1}[z \geq 0] \cdot \mathcal{N}(z;\, 0, 1)}{0.5} = 2\,\mathcal{N}(z;\, 0, 1)\, \mathbf{1}[z \geq 0]$$

Step 3: Interpret the result:

$$\boxed{p(z \mid x^{(1)}) = 2\,\mathcal{N}(z;\, 0, 1)\ \mathbf{1}[z \geq 0]}$$

This is a **truncated standard normal** distribution (also called a *half-normal*), supported on $[0, \infty)$. It takes the standard normal density, restricts it to the non-negative real line, and multiplies by 2 to renormalize (since the restricted region had probability 0.5 under the original distribution, multiplying by $1/0.5 = 2$ ensures the density integrates to 1).

Verification: $\int_0^\infty 2\,\mathcal{N}(z; 0, 1)\, dz = 2 \times 0.5 = 1$ ✓

2(c) KL Divergence with $q(z) = \mathcal{N}(z;\, 0, 1)$

Question

Let $q(z) = \mathcal{N}(z;\, 0, 1)$. Compute $D_{\mathrm{KL}}(q(z) \,\|\, p(z \mid x^{(1)}))$.

Answer

By definition:

$$D_{\mathrm{KL}}(q \,\|\, p(\cdot|x^{(1)})) = \int_{-\infty}^{\infty} q(z)\, \log \frac{q(z)}{p(z \mid x^{(1)})}\, dz$$

We split the integral into two regions based on the support of the posterior.

Region 1: $z \geq 0$**.** Here $q(z) = \mathcal{N}(z; 0, 1)$ and $p(z|x^{(1)}) = 2\mathcal{N}(z; 0, 1)$, so:

$$\frac{q(z)}{p(z \mid x^{(1)})} = \frac{\mathcal{N}(z; 0, 1)}{2\,\mathcal{N}(z; 0, 1)} = \frac{1}{2}$$

The contribution from this region is:

$$\int_0^\infty \mathcal{N}(z; 0, 1) \cdot \log\left(\tfrac{1}{2}\right) dz = (-\log 2) \cdot \underbrace{\int_0^\infty \mathcal{N}(z; 0, 1)\, dz}_{=0.5} = -\tfrac{1}{2} \log 2 \approx -0.347$$

This is a *finite, negative* contribution.

Region 2: $z < 0$**.** Here $q(z) = \mathcal{N}(z; 0, 1) > 0$ but $p(z|x^{(1)}) = 0$. The integrand becomes:

$$q(z) \log \frac{q(z)}{0} = q(z) \cdot (+\infty) = +\infty$$

Since q assigns positive probability to this region ($P_q(z < 0) = 0.5 > 0$), this integral **diverges to** $+\infty$.

Combining both regions:

$$\boxed{D_{\mathrm{KL}}(q(z) \,\|\, p(z \mid x^{(1)})) = -\tfrac{1}{2} \log 2 + \infty = +\infty}$$

Key Insight

This illustrates a fundamental property of KL divergence: $D_{\mathrm{KL}}(q\|p) = +\infty$ whenever the support of q is *not contained in* the support of p, i.e., $\mathrm{supp}(q) \not\subseteq \mathrm{supp}(p)$. Here, $q = \mathcal{N}(0, 1)$ has support $(-\infty, \infty)$ but $p(z|x^{(1)})$ has support $[0, \infty)$. Since q places mass where p is zero, the KL diverges.

2(d) $P_q(z \geq 0)$ for $q(z) = \mathcal{N}(z;\, 1, 1)$

Question

Now consider $q(z) = \mathcal{N}(z;\, 1, 1)$. Compute numerically $P_q(z \geq 0)$.

Answer

We have $z \sim \mathcal{N}(1, 1)$, meaning z has mean $\mu = 1$ and variance $\sigma^2 = 1$ (standard deviation $\sigma = 1$).

Step 1: Standardize. To use the standard normal CDF Φ, we convert z to a standard normal variable $Z = \frac{z-\mu}{\sigma}$:

$$P_q(z \geq 0) = P\left(\frac{z-1}{1} \geq \frac{0-1}{1}\right) = P(Z \geq -1)$$

where $Z \sim \mathcal{N}(0, 1)$.

Step 2: Use symmetry. For the standard normal:

$$P(Z \geq -1) = P(Z \leq 1) = \Phi(1)$$

The second equality uses the symmetry property $P(Z \geq -a) = P(Z \leq a)$ for the standard normal.

Step 3: Look up $\Phi(1)$**.** From standard normal tables:

$$\boxed{P_q(z \geq 0) = \Phi(1) \approx 0.8413}$$

This means approximately 84.13% of the probability mass of $q(z) = \mathcal{N}(1, 1)$ lies in the region $z \geq 0$, and the remaining $P_q(z < 0) = 1 - 0.8413 = 0.1587$ ($\approx$15.87%) lies in $z < 0$.

2(e) ELBO Computation

Question

Using your answer above, compute:

$$\text{ELBO}(x^{(1)};\, q) = \log p(x^{(1)}) - D_{\text{KL}}(q(z) \,\|\, p(z \mid x^{(1)}))$$

Is it finite?

Answer

We can approach this from two equivalent angles.

Approach 1: Direct formula from the question.

$$\text{ELBO}(x^{(1)}; q) = \log p(x^{(1)}) - D_{\text{KL}}(q(z) \| p(z|x^{(1)}))$$

We know $\log p(x^{(1)}) = -\ln 2$ (finite). But what about the KL divergence with this new $q = \mathcal{N}(1, 1)$? Let us check: $q(z) = \mathcal{N}(z; 1, 1)$ assigns probability $P_q(z < 0) = 0.1587 > 0$ to the region where $p(z|x^{(1)}) = 0$. By the same argument as in part (c), $D_{\text{KL}}(q \| p(\cdot|x^{(1)})) = +\infty$.
Therefore: ELBO $= -\ln 2 - \infty = -\infty$.

Approach 2: Standard three-term decomposition.
The ELBO can also be written as:

$$\text{ELBO}(x^{(1)};\, q) = \underbrace{\mathbb{E}_{q(z)}[\log p(x^{(1)} \mid z)]}_{\text{Expected log-likelihood}} + \underbrace{\mathbb{E}_{q(z)}[\log p(z)] - \mathbb{E}_{q(z)}[\log q(z)]}_{-D_{\text{KL}}(q(z)\|p(z))}$$

Let us evaluate each term:
Term 1 (expected log-likelihood):

$$\mathbb{E}_q[\log p(x^{(1)} \mid z)] = \log(1) \cdot P_q(z \geq 0) + \log(0) \cdot P_q(z < 0)$$

$$= 0 \times 0.8413 + (-\infty) \times 0.1587 = -\infty$$

Since $P_q(z < 0) = 0.1587 > 0$ and $\log(0) = -\infty$, this term diverges to $-\infty$.
Terms 2 and 3: $-D_{\text{KL}}(q(z)\|p(z))$ is the negative KL between $\mathcal{N}(1, 1)$ and $\mathcal{N}(0, 1)$, which equals $-\frac{1}{2}[1 + 0 - 1 - 1] = -\frac{1}{2}$. This is **finite**.
But it does not matter: since Term 1 $= -\infty$ and the other terms are finite, the sum is $-\infty$.

$$\boxed{\text{ELBO}(x^{(1)};\, q) = -\infty \qquad \text{(not finite)}}$$

Key Insight

The ELBO is $-\infty$ because $q(z) = \mathcal{N}(1, 1)$ places $\sim$15.87% of its mass on $z < 0$, where $p(x^{(1)}|z) = 0$. The expected log-likelihood term hits $\log(0) = -\infty$ on this region, making the entire ELBO degenerate.

Practical lesson: A variational distribution whose support extends beyond the support of the likelihood can produce a $-\infty$ ELBO. A suitable q for this problem would need support restricted to $z \geq 0$ (e.g., a truncated normal, log-normal, or half-normal distribution).

3. Theory Questions

3(a) Goal of Generative AI

Question

What is the goal of Generative AI? How does it differ from discriminative modeling?

Answer

Generative AI aims to learn the underlying data distribution $p_{\text{data}}(\mathbf{x})$ such that the model can **generate new, realistic samples** statistically indistinguishable from real data. Formally, we learn a parametric model $p_\theta(\mathbf{x}) \approx p_{\text{data}}(\mathbf{x})$.
Discriminative modeling learns the conditional distribution $p(y \mid \mathbf{x})$ - the mapping from inputs to labels (or outputs). It models only the *decision boundary* between classes, not how the data $\mathbf{x}$ itself is distributed.

	Generative	**Discriminative**
What it models	$p(\mathbf{x})$ or $p(\mathbf{x}, y)$	$p(y \mid \mathbf{x})$
Core question	"What does this data look like?"	"What label does this input have?"
Can generate new data?	Yes	No
Can evaluate density?	Often yes	Not applicable
Examples	VAE, GAN, Diffusion, Flow	Logistic regression, SVM, BERT

In short: generative models learn the full data distribution (enabling both generation and density evaluation), while discriminative models learn only the input-output mapping (sufficient for classification but not for generation).

3(b) Historical Milestones

Question

Give two historical milestones in the development of modern Generative AI and explain why they were significant.

Answer

1. Generative Adversarial Networks (GANs) - Goodfellow et al., NeurIPS 2014:
GANs introduced the **adversarial training** paradigm: a generator network G produces fake samples, and a discriminator network D tries to distinguish them from real data. Both are trained simultaneously in a minimax game.
Why significant: GANs were the first framework to bypass tractable likelihood computation entirely. They showed that high-quality, sharp image generation was possible without ever computing $p(\mathbf{x})$. This opened an entirely new class of *implicit* generative models and inspired thousands of follow-up works (DCGAN, StyleGAN, CycleGAN, etc.).

2. Denoising Diffusion Probabilistic Models (DDPMs) - Ho et al., NeurIPS 2020:

DDPMs showed that iteratively denoising Gaussian noise through a learned reverse Markov chain could produce samples *rivaling and surpassing* GANs in both quality and diversity.
Why significant: DDPMs solved two major problems of GANs - **training instability** and **mode collapse** - while providing tractable likelihood bounds. They became the foundation for modern text-to-image systems (DALL-E 2, Stable Diffusion, Imagen), fundamentally changing how generative models are built and deployed.

3(c) Latent Variable Models for High-Dimensional Data

Question

Explain why latent variable models are useful for high-dimensional data such as images or speech.

Answer

High-dimensional data (e.g., a $1024 \times 1024 \times 3$ image has $\sim$3 million dimensions) lives in an extremely large ambient space. However, real data typically occupies a much **lower-dimensional manifold** within this space. For instance, natural face images are determined by a relatively small number of factors: pose, lighting, expression, identity, etc.
Latent variable models formalize this insight by introducing a compact latent space $\mathbf{z} \in \mathbb{R}^d$ (where $d \ll$ data dimensionality) and defining the generative process as:

$$\mathbf{z} \sim p(\mathbf{z}), \qquad \mathbf{x} \sim p_\theta(\mathbf{x} \mid \mathbf{z})$$

Why this is useful:

1. **Dimensionality reduction:** The latent space provides a compact, structured representation that captures the most salient factors of variation, discarding irrelevant high-dimensional noise.
2. **Tractable generation:** Sampling is easy - draw $\mathbf{z}$ from the low-dimensional prior (e.g., a Gaussian), then decode it. No need to model complex high-dimensional dependencies directly.
3. **Meaningful latent structure:** The latent space supports operations like interpolation (smooth transitions between data points), clustering (grouping similar data), and arithmetic (e.g., "smiling face" $-$ "neutral face" $+$ "neutral male" $=$ "smiling male").
4. **Principled uncertainty:** The probabilistic framework allows reasoning about uncertainty - e.g., which regions of the latent space are well-supported by data and which are extrapolations.

3(d) Latent Variable Models vs. Autoregressive for Medical Imaging

Question

Suppose you are modeling medical images where interpretability of latent factors is important. What advantages do latent variable models provide compared to autoregressive models?

Answer

Latent Variable Models (e.g., VAEs): Learn a continuous latent space where each dimension can potentially correspond to a **clinically meaningful factor** (e.g., tumor size, tissue density, organ shape, disease stage). With disentangled variants like β-VAE, individual latent dimensions can be isolated to represent single factors. This enables:

- **Factor-level analysis:** A clinician can vary one latent dimension and observe its effect on the generated image, asking "what happens if tumor size increases while everything else stays the same?"
- **Explainability:** The compact latent representation makes the model's internal representation inspectable and interpretable.
- **Anomaly detection via latent outliers:** Unusual latent codes can flag anomalous images.

Autoregressive Models (e.g., PixelCNN): Decompose the image into a sequence of pixel-level conditionals: $p(x_i \mid x_1, \ldots, x_{i-1})$. They have *no explicit latent space*. This means:

- There is no compact set of factors to interpret.
- One cannot easily ask "what changed between these two images?" because the model reasons at *pixel level*, not *semantic level*.
- For medical applications demanding **explainability and clinical interpretability**, this pixel-level reasoning is a major limitation.

3(e) Variational Inference vs. Exact Bayesian Inference

Question

Why is variational inference often preferred over exact Bayesian inference in deep generative models?

Answer

Exact Bayesian inference requires computing the posterior:

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})\,p(\mathbf{z})}{p(\mathbf{x})}, \qquad \text{where} \quad p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})\,p(\mathbf{z})\,d\mathbf{z}$$

The marginal $p(\mathbf{x})$ requires integrating over *all possible latent configurations*. For deep generative models with complex, nonlinear decoders, this integral is **intractable** - it has no closed-form solution and cannot be computed analytically.

Alternative approaches and their limitations:

- **MCMC (Markov Chain Monte Carlo):** Provides asymptotically exact samples from the posterior, but is extremely slow for high-dimensional latent spaces, requires careful tuning of proposal distributions, and does not parallelize well over large datasets. Impractical for training deep networks on millions of data points.
- **Exact enumeration:** Only works for discrete, finite latent spaces - inapplicable to continuous latent variables.

Variational inference provides a scalable alternative: it introduces an approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ parameterized by a neural network (the "encoder") and optimizes the ELBO:

$$\text{ELBO} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))$$

This converts the intractable integration problem into a **tractable optimization problem** solvable with standard gradient-based methods (SGD, Adam). The *reparameteriza-*

tion trick ($\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$) enables backpropagation through the sampling step, making the entire pipeline end-to-end differentiable and scalable to large datasets.

3(f) Limitations of Variational Inference

Question

What are the main limitations of variational inference? Discuss the role of the variational gap.

Answer

Main Limitations:

1. **Approximate, not exact:** The ELBO is only a *lower bound* on $\log p(\mathbf{x})$. Maximizing the ELBO does not guarantee maximizing the true log-likelihood.
2. **Constrained variational family:** If $q_\phi(\mathbf{z}|\mathbf{x})$ is restricted to a simple parametric family (e.g., a diagonal Gaussian), it may not capture complex posterior structures such as **multimodality** (multiple distinct modes), **skewness**, or **strong correlations** between latent dimensions.
3. **Posterior collapse:** In practice, the model can learn to ignore the latent variable entirely, especially with powerful decoders (e.g., autoregressive decoders), leading to $D_{\mathrm{KL}}(q\|p) \approx 0$ and an uninformative latent space.

The Variational Gap. Recall the exact identity:

$$\log p_\theta(\mathbf{x}) = \mathrm{ELBO} + \underbrace{D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \,\|\, p_\theta(\mathbf{z}|\mathbf{x}))}_{\text{variational gap} \geq 0}$$

The gap $= D_{\mathrm{KL}}(q_\phi \| p_\theta) \geq 0$ and equals zero *only* when $q_\phi(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{z}|\mathbf{x})$ exactly. A large variational gap means:

(a) The ELBO provides a **misleading training signal** - the model may appear to have low likelihood when it actually does not.

(b) The learned latent representations are **poor** because they do not faithfully capture the structure of the true posterior.

3(g) Standard Autoencoder vs. VAE

Question

Explain the key differences between a standard autoencoder and a VAE.

Answer

Standard Autoencoder:

- **Architecture:** Encoder $\mathbf{x} \to \mathbf{z}$, Decoder $\mathbf{z} \to \hat{\mathbf{x}}$ - both are *deterministic* mappings.
- **Loss:** Reconstruction loss only, e.g., $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$.
- **Latent space:** Has *no probabilistic structure*. The encoder maps each input to a single point in latent space. The resulting latent space can be irregular, with gaps, discontinuities, and no guarantee that nearby points decode to similar outputs.
- **Generation:** Cannot sample meaningfully. If you pick a random point in the latent space, the decoder may produce garbage because it was never trained on that region.

Variational Autoencoder (VAE):

- **Architecture:** Encoder outputs *distribution parameters*: $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_\phi(\mathbf{x}), \boldsymbol{\sigma}^2_\phi(\mathbf{x}))$. A latent code $\mathbf{z}$ is then *sampled* from this distribution.
- **Loss:** The ELBO, which has two terms:

$$\text{ELBO} = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{reconstruction quality}} - \underbrace{D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))}_{\text{regularization}}$$

- **Latent space:** The KL term regularizes the latent space to be smooth and continuous (close to $\mathcal{N}(\mathbf{0}, \mathbf{I})$). This ensures that nearby latent points decode to similar outputs and that every region of the latent space produces meaningful outputs.
- **Generation:** Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and decode. Because the latent space is regularized, this produces coherent samples.
- **Key innovation:** The **reparameterization trick** ($\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$) allows backpropagation through the stochastic sampling step, enabling end-to-end training.

3(h) Anomaly Detection in Industrial Equipment

Question

In a real-world application such as anomaly detection in industrial equipment, which model would you choose and why?

Answer

Recommended: VAE or Normalizing Flow.

Anomaly detection requires two capabilities: (1) learn what "normal" data looks like, and (2) assign a score to new data indicating how anomalous it is. The most principled anomaly score is the **data likelihood** $p(\mathbf{x})$ - anomalous inputs will have low likelihood under a model trained on normal data.

- **Normalizing Flows:** Provide *exact* $\log p(\mathbf{x})$ via the change-of-variables formula. This is the gold standard for density-based anomaly detection. Anomaly score $= -\log p(\mathbf{x})$.
- **VAEs:** Provide two complementary anomaly signals: (i) the ELBO as a proxy for $\log p(\mathbf{x})$, and (ii) the reconstruction error $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$. Anomalous inputs are poorly reconstructed and have low ELBO.
- **GANs:** *Unsuitable* - they provide no density estimate and no natural anomaly score. Discriminator scores are unreliable for out-of-distribution detection.
- **Diffusion models:** Can compute (bounds on) likelihood, but are computationally expensive for real-time industrial monitoring where low latency is critical.

3(i) GANs: Advantages and Disadvantages

Question

Discuss two advantages and two disadvantages of GANs compared to likelihood-based models.

Answer

Advantages:

1. **Superior sample quality:** GANs produce sharper, more visually realistic samples than likelihood-based models (especially VAEs). This is because the adversarial loss directly optimizes for *perceptual realism* - the generator must fool a discriminator trained to detect fakes - rather than pixel-wise metrics like MSE, which tend to reward blurry averages.
2. **Architectural flexibility:** Since GANs never compute $\log p(\mathbf{x})$, the generator and discriminator can use *any* architecture. There is no need for invertibility (as in flows), tractable conditionals (as in autoregressive models), or differentiable sampling (as in VAEs). This allows arbitrary, expressive network designs.

Disadvantages:

1. **Training instability and mode collapse:** GAN training is a minimax game that is notoriously hard to stabilize. *Mode collapse* occurs when the generator learns to produce only a few high-quality samples (or even a single sample) that fool the discriminator, failing to capture the full diversity of the data distribution. Many heuristics and architectural choices are needed for stable training.
2. **No likelihood evaluation:** GANs are implicit models - they can generate samples but cannot compute $p(\mathbf{x})$ for a given $\mathbf{x}$. This makes them unsuitable for tasks requiring density estimation (anomaly detection), model comparison (via held-out log-likelihood), or any application requiring calibrated uncertainty.

3(j) Generating Sports Player Avatars

Question

You are building a system to generate realistic sports player avatars for a video game. Would you choose a GAN, VAE, or diffusion model? Justify your answer.

Answer

Recommended: GAN (specifically StyleGAN or similar).

The primary requirements for game avatar generation are: (i) high visual quality at high resolution, (ii) controllability over attributes (hair, face shape, skin tone, expression), and (iii) fast generation for game asset pipelines.

- **StyleGAN** excels on all three counts. It produces photorealistic human faces and bodies at high resolution (1024^2). Its *style-based architecture* provides fine-grained control over visual attributes through latent space manipulation - change one dimension to adjust hair style, another for facial expression, etc. Most critically, it generates images in a **single forward pass** ($\sim$50 ms), enabling rapid batch generation of game assets.
- **VAEs** produce blurrier outputs due to the pixel-level reconstruction loss, which averages over modes. This results in avatars that lack sharpness and fine detail - unacceptable for a commercial video game.
- **Diffusion models** produce excellent quality (comparable to or better than GANs), but require **hundreds of sequential denoising steps** per image. This makes them orders of magnitude slower - a significant bottleneck when generating thousands of unique avatars for a game.

3(k) Normalizing Flows vs. VAEs

Question

What are the main advantages of normalizing flows over VAEs?

Answer

1. **Exact likelihood computation:** Flows compute the *exact* $\log p(\mathbf{x})$ via the change-of-variables formula:

$$\log p(\mathbf{x}) = \log p_Z(f^{-1}(\mathbf{x})) + \log |\det J_{f^{-1}}(\mathbf{x})|$$

   VAEs only optimize a *lower bound* (ELBO). This means flows enable exact model comparison, exact density estimation, and exact anomaly scoring.
2. **Exact latent inference:** The inverse mapping $\mathbf{z} = f^{-1}(\mathbf{x})$ is computed exactly. There is *no variational gap*, no encoder approximation error, and the latent representation is lossless. In a VAE, the encoder $q_\phi(\mathbf{z}|\mathbf{x})$ is an approximation that introduces error.
3. **Richer posterior approximation:** Flows can transform a simple Gaussian into an arbitrarily complex distribution through a sequence of invertible mappings. They can also be used *within* VAEs to build more flexible approximate posteriors (replacing the diagonal Gaussian encoder with a flow-based encoder), giving the best of both worlds.

3(l) Scaling Flows to High-Resolution Images

Question

Why are normalizing flows typically harder to scale to very high-resolution images?

Answer

Normalizing flows require the transformation f to be **bijective** with a **tractable Jacobian determinant**. For high-resolution images (e.g., $1024 \times 1024 \times 3 \approx 3.1$M dimensions), these constraints impose severe challenges:

1. **Jacobian cost:** Computing $\log |\det J|$ naively costs $O(d^3)$ for d-dimensional data. Even with architectural tricks (coupling layers make the Jacobian triangular, reducing cost to $O(d)$), the constant factor is large for millions of dimensions, and these coupling-layer designs **limit the expressiveness** of each layer.
2. **No dimensionality reduction:** Because f must be a bijection, the latent space $\mathbf{z}$ must have the *same dimensionality* as the data space $\mathbf{x}$. This means a $1024 \times 1024 \times 3$ image requires a $\sim$3M-dimensional latent vector - no compression is possible. This wastes capacity on modeling noise and irrelevant dimensions.
3. **Depth requirements:** Because each layer has limited expressiveness (constrained by the tractable Jacobian requirement), achieving a sufficiently complex overall transformation requires **very deep** networks with many layers. This dramatically increases memory consumption and training time.

By contrast, GANs and diffusion models face *none* of these constraints: they use arbitrary (non-invertible) architectures, can map between spaces of different dimensions, and have no Jacobian requirements.

3(m) Main Idea Behind Diffusion Models

Question

Explain the main idea behind diffusion models.

Answer

Diffusion models are based on a simple but powerful two-phase idea:
Phase 1 - Forward Process (Destruction): Start with a clean data sample $\mathbf{x}_0$ from the data distribution. Gradually add small amounts of Gaussian noise over T timesteps:

$$\mathbf{x}_0 \xrightarrow{+\text{noise}} \mathbf{x}_1 \xrightarrow{+\text{noise}} \mathbf{x}_2 \xrightarrow{+\text{noise}} \cdots \xrightarrow{+\text{noise}} \mathbf{x}_T \approx \mathcal{N}(\mathbf{0}, \mathbf{I})$$

After sufficiently many steps, the data is completely destroyed and $\mathbf{x}_T$ is indistinguishable from pure Gaussian noise. This forward process is *fixed* (not learned) and has a known, simple mathematical form.
Phase 2 - Reverse Process (Generation): A neural network learns to **undo each noise step**. Given a noisy image $\mathbf{x}_t$, the network predicts the noise that was added, allowing us to recover a slightly cleaner image $\mathbf{x}_{t-1}$. Starting from pure noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, the learned reverse process iteratively denoises:

$$\mathbf{x}_T \xrightarrow{\text{denoise}} \mathbf{x}_{T-1} \xrightarrow{\text{denoise}} \cdots \xrightarrow{\text{denoise}} \mathbf{x}_1 \xrightarrow{\text{denoise}} \mathbf{x}_0$$

producing a clean data sample.
Key insight: Each individual denoising step only needs to remove a *small* amount of noise, which is a tractable Gaussian-to-Gaussian transition. By chaining many small, easy steps, the model can learn to generate complex data distributions that would be impossible to capture in a single step.

3(n) Diffusion Models vs. GANs: Quality

Question

Why do diffusion models often produce higher-quality samples than GANs?

Answer

1. **Stable training:** Diffusion models optimize a simple, well-behaved objective - the mean squared error between predicted and actual noise: $\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2$. This is a standard regression loss with no adversarial dynamics, no delicate generator-discriminator balance, and no mode collapse. Training is as stable as training a standard supervised model.
2. **Full distribution coverage:** The forward process injects noise *everywhere*, so the model must learn to denoise from every part of the data distribution. There is no mechanism to "ignore" underrepresented modes. In contrast, GANs can suffer from **mode dropping**: the generator finds a few modes that consistently fool the discriminator and ignores the rest.
3. **Iterative refinement:** The multi-step reverse process allows progressive refinement: early steps establish coarse structure (layout, shapes), middle steps add medium-scale features (textures, edges), and final steps refine fine details (skin pores, hair strands). GANs must produce all levels of detail in a **single forward pass**, which is fundamen-

tally harder.

3(o) Computational Disadvantage of Diffusion Models

Question

What is the primary computational disadvantage of diffusion models compared to GANs?

Answer

Slow inference (generation) speed.
Diffusion models require T **sequential** denoising steps to generate a single sample. In the original DDPM, $T = 1000$, meaning 1000 full neural network forward passes per generated image. Even with accelerated sampling methods like DDIM (which reduces this to 20 to 50 steps), generation is still *far slower* than GANs.

Comparison:

Model	Forward passes per sample	Typical time (256×256)
GAN	1	∼10 to 50 ms
DDPM	1000	∼10 to 60 seconds
DDIM	20 to 50	∼0.5 to 2 seconds

This makes diffusion models **unsuitable for real-time applications** without significant optimization (progressive distillation, consistency models, etc.).

3(p) Forward and Reverse Processes in DDPM

Question

Explain the forward and reverse processes in Denoising Diffusion Probabilistic Models (DDPM).

Answer

Forward Process (Diffusion):
Starting from clean data $\mathbf{x}_0 \sim p_{\text{data}}$, we define a Markov chain that progressively adds Gaussian noise according to a fixed variance schedule $\{\beta_t\}_{t=1}^{T}$ with $0 < \beta_t < 1$:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}\Big(\mathbf{x}_t;\ \sqrt{1-\beta_t}\,\mathbf{x}_{t-1},\ \beta_t\mathbf{I}\Big)$$

At each step, the mean shrinks by $\sqrt{1-\beta_t}$ (slightly less than 1) and noise with variance β_t is added. After T steps (with T large enough), $\mathbf{x}_T$ is approximately $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Crucially, there is a *closed-form* for any step:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\Big(\mathbf{x}_t;\ \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0,\ (1-\bar{\alpha}_t)\mathbf{I}\Big), \qquad \bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$$

This allows us to sample *any* noisy version $\mathbf{x}_t$ directly from $\mathbf{x}_0$ without iterating through all intermediate steps.

Reverse Process (Denoising):

Start from pure noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The reverse process learns Gaussian transitions:

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}\Big(\mathbf{x}_{t-1};\ \boldsymbol{\mu}_\theta(\mathbf{x}_t, t),\ \sigma_t^2 \mathbf{I}\Big)$$

A neural network $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ predicts the noise that was added at step t. The mean is then computed as:

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\, \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right)$$

After T reverse steps, the result is a clean sample $\mathbf{x}_0$ from the learned distribution.

3(q) Real-Time Deployment Challenges for DDPM

Question

In a real-time content generation application (e.g., live video enhancement), what challenges would arise when deploying DDPM models?

Answer

1. **Latency:** DDPM requires hundreds of sequential denoising steps per frame. For live video at 30+ fps, each frame must be processed in < 33 ms, but a single DDPM generation takes seconds to minutes. This is a **fundamental bottleneck**.
2. **Computational cost:** Each denoising step is a full U-Net forward pass at the full image resolution. At 30 fps with $T = 50$ steps (using DDIM), that is $\sim$ 1500 U-Net forward passes per second - requiring prohibitively expensive GPU resources.
3. **Temporal flickering:** DDPM generates each frame independently by starting from random noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Different random seeds for adjacent frames produce visually inconsistent outputs, causing **temporal flickering**. Maintaining smooth video requires architectural modifications (e.g., temporal attention, shared noise schedules).
4. **Mitigation strategies:** Progressive distillation (train a student model to denoise in fewer steps), consistency models (single-step generation), DDIM reduced-step sampling, latent diffusion (operate in a compressed latent space instead of pixel space), and hardware-specific optimization (TensorRT, model quantization).

3(r) Model Selection for Specific Tasks

Question

You are designing a generative model for the following tasks:

(a) Generating realistic synthetic MRI scans where likelihood evaluation is required for anomaly scoring.

(b) Generating creative artwork for marketing campaigns where visual quality is the primary objective.

For each case, select an appropriate model class and justify your choice.

Answer

Task (a) - Synthetic MRI with Likelihood-Based Anomaly Scoring:
Normalizing Flow or Diffusion Model.
The requirement for likelihood-based anomaly scoring means we need a model that can

compute (or tightly bound) $\log p(\mathbf{x})$:

- **Normalizing flows** provide *exact* $\log p(\mathbf{x})$ - the gold standard for density-based anomaly scoring.
- **Diffusion models** provide tight variational bounds on $\log p(\mathbf{x})$, which serve as effective anomaly scores.
- **VAEs** can use the ELBO as a proxy, but the bound is less tight due to the variational gap.
- **GANs** are *unsuitable* - they cannot compute $p(\mathbf{x})$ at all.

Task (b) - Creative Artwork for Marketing:
Diffusion Model or GAN.
Visual quality and creative diversity are paramount; density estimation is not needed:

- **Diffusion models** (Stable Diffusion, DALL-E 2, Imagen) produce the highest-quality images with excellent text-conditioned generation - ideal for marketing where specific creative briefs must be followed.
- **GANs** (StyleGAN) are excellent for specific visual domains (faces, art styles) with very fast inference, suitable for rapid iteration.
- **VAEs** and **flows** produce lower visual quality, making them less suitable when visual impact is the priority.

4. Numericals - DDPM

4(a) Closed-Form Derivation of $q(\mathbf{x}_t \mid \mathbf{x}_0)$

Question

Consider a DDPM with forward diffusion:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}\Big(\mathbf{x}_t;\ \sqrt{1-\beta_t}\,\mathbf{x}_{t-1},\ \beta_t\mathbf{I}\Big)$$

Show that the forward process admits the closed-form:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\Big(\mathbf{x}_t;\ \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0,\ (1-\bar{\alpha}_t)\mathbf{I}\Big)$$

where $\alpha_t = 1-\beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t}\alpha_s$.

Answer

Proof by induction on t.

Setup. Define $\alpha_t = 1-\beta_t$. Using the reparameterization trick, we can write the forward step as:

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

This is equivalent to $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\sqrt{\alpha_t}\,\mathbf{x}_{t-1},\ (1-\alpha_t)\mathbf{I})$.

Base case ($t=1$):

$$\mathbf{x}_1 = \sqrt{\alpha_1}\,\mathbf{x}_0 + \sqrt{1-\alpha_1}\,\boldsymbol{\epsilon}_1 \implies q(\mathbf{x}_1|\mathbf{x}_0) = \mathcal{N}(\sqrt{\alpha_1}\,\mathbf{x}_0,\ (1-\alpha_1)\mathbf{I})$$

Since $\bar{\alpha}_1 = \alpha_1$, this matches $\mathcal{N}(\sqrt{\bar{\alpha}_1}\,\mathbf{x}_0,\ (1-\bar{\alpha}_1)\mathbf{I})$. ✓

Inductive step: $t-1 \to t$. Assume $q(\mathbf{x}_{t-1}|\mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_{t-1}}\,\mathbf{x}_0,\ (1-\bar{\alpha}_{t-1})\mathbf{I})$. Substituting the reparameterized form of $\mathbf{x}_{t-1}$:

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\,\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\,\boldsymbol{\epsilon}_t \\
&= \sqrt{\alpha_t}\Big(\sqrt{\bar{\alpha}_{t-1}}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\,\boldsymbol{\epsilon}'_{t-1}\Big) + \sqrt{1-\alpha_t}\,\boldsymbol{\epsilon}_t \\
&= \underbrace{\sqrt{\alpha_t\bar{\alpha}_{t-1}}}_{\sqrt{\bar{\alpha}_t}}\,\mathbf{x}_0 + \underbrace{\sqrt{\alpha_t(1-\bar{\alpha}_{t-1})}\,\boldsymbol{\epsilon}'_{t-1} + \sqrt{1-\alpha_t}\,\boldsymbol{\epsilon}_t}_{\text{sum of two independent Gaussians}} \qquad (1)
\end{aligned}$$

where $\boldsymbol{\epsilon}'_{t-1} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ is the accumulated noise from steps 1 to $t-1$, and $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ is the fresh noise at step t.

Merging the noise terms. Since $\boldsymbol{\epsilon}'_{t-1}$ and $\boldsymbol{\epsilon}_t$ are *independent* standard Gaussians, the sum $a\boldsymbol{\epsilon}'_{t-1} + b\boldsymbol{\epsilon}_t$ is Gaussian with mean $\mathbf{0}$ and variance a^2+b^2:

$$\text{Var} = \alpha_t(1-\bar{\alpha}_{t-1}) + (1-\alpha_t) = \alpha_t - \alpha_t\bar{\alpha}_{t-1} + 1 - \alpha_t = 1 - \alpha_t\bar{\alpha}_{t-1} = 1-\bar{\alpha}_t$$

Therefore the combined noise term equals $\sqrt{1-\bar{\alpha}_t}\,\bar{\boldsymbol{\epsilon}}$ where $\bar{\boldsymbol{\epsilon}} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$.

Result: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\bar{\boldsymbol{\epsilon}}$, which gives:

$$\boxed{q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\Big(\mathbf{x}_t;\ \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0,\ (1-\bar{\alpha}_t)\mathbf{I}\Big)} \quad \blacksquare$$

4(b) DDPM Training Objective

Question

Write the full training objective used in DDPM when the model predicts the noise $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$.

Answer

Derivation sketch. The full DDPM objective starts from the variational lower bound (VLB) on $\log p(\mathbf{x}_0)$:

$$\log p(\mathbf{x}_0) \geq \mathbb{E}_q\left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right]$$

After decomposition, most terms reduce to KL divergences between Gaussians. Ho et al. (2020) showed that when the model is parameterized to predict the noise $\boldsymbol{\epsilon}_\theta$, all terms simplify to a single unified loss:

$$\boxed{L_{\text{simple}} = \mathbb{E}_{t,\,\mathbf{x}_0,\,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\underbrace{\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}}_{\mathbf{x}_t},\; t\right)\right\|^2\right]}$$

where:

- $t \sim \text{Uniform}\{1, 2, \ldots, T\}$ - randomly sample a timestep
- $\mathbf{x}_0 \sim p_{\text{data}}$ - sample a clean data point
- $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ - sample random noise
- $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$ - create the noisy input using the closed-form from part (a)
- $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ - the neural network predicts the noise $\boldsymbol{\epsilon}$ from the noisy input $\mathbf{x}_t$ and timestep t

Intuition: The training procedure is remarkably simple: (1) pick a random data point, (2) add a random amount of noise, (3) ask the neural network to predict what noise was added, (4) penalize the prediction error with MSE. No adversarial training, no complex loss functions.

Numerical Computation: $\beta_1 = 0.1,\ \beta_2 = 0.2,\ x_0 = 2$

(a) Compute α_1, α_2, and $\bar{\alpha}_2$

Question

Let $\beta_1 = 0.1$, $\beta_2 = 0.2$. Let $x_0 = 2$. Compute α_1, α_2 and $\bar{\alpha}_2$.

Answer

By definition, $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$:

$$\alpha_1 = 1 - \beta_1 = 1 - 0.1 = \boxed{0.9}$$

$$\alpha_2 = 1 - \beta_2 = 1 - 0.2 = \boxed{0.8}$$

$$\bar{\alpha}_2 = \alpha_1 \cdot \alpha_2 = 0.9 \times 0.8 = \boxed{0.72}$$

(b) Mean and Variance of $q(x_2 \mid x_0)$

Question

Compute the mean and variance of $q(x_2 \mid x_0)$.

Answer

From the closed-form derived in 4(a), with the 1D scalar case ($\mathbf{I} = 1$):

$$q(x_2 \mid x_0) = \mathcal{N}\Big(x_2;\ \sqrt{\bar{\alpha}_2}\, x_0,\ 1 - \bar{\alpha}_2\Big)$$

Substituting $\bar{\alpha}_2 = 0.72$ and $x_0 = 2$:

$$\begin{aligned} \text{Mean} &= \sqrt{\bar{\alpha}_2} \cdot x_0 = \sqrt{0.72} \times 2 \\ &= 0.8485 \times 2 = \boxed{1.6971} \end{aligned}$$

$$\text{Variance} = 1 - \bar{\alpha}_2 = 1 - 0.72 = \boxed{0.28}$$

$$q(x_2 \mid x_0 = 2) = \mathcal{N}(x_2;\ 1.6971,\ 0.28)$$

Interpretation: After two noising steps, the original value $x_0 = 2$ has been attenuated toward zero (mean went from 2 to 1.70) and spread out (variance grew from 0 to 0.28). With more steps, the mean would approach 0 and the variance would approach 1, converging to $\mathcal{N}(0, 1)$.

(c) Why is $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ Gaussian?

Question

Explain why the reverse distribution $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ is Gaussian.

Answer

Step 1: Condition on $\mathbf{x}_0$. By Bayes' rule:

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

Since the forward process is Markov, $q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) = q(\mathbf{x}_t|\mathbf{x}_{t-1})$. So:

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t,\, \mathbf{x}_0) \ \propto\ \underbrace{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}_{\text{Gaussian}} \cdot \underbrace{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}_{\text{Gaussian}}$$

Step 2: Product of Gaussians. Both factors are Gaussian distributions in $\mathbf{x}_{t-1}$:

- $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t;\ \sqrt{\alpha_t}\ \mathbf{x}_{t-1},\ \beta_t\mathbf{I}) \implies$ viewed as a function of $\mathbf{x}_{t-1}$, this is a Gaussian in $\mathbf{x}_{t-1}$.
- $q(\mathbf{x}_{t-1}|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1};\ \sqrt{\bar{\alpha}_{t-1}}\ \mathbf{x}_0,\ (1 - \bar{\alpha}_{t-1})\mathbf{I})$ is directly a Gaussian in $\mathbf{x}_{t-1}$.

The product of two Gaussians (in the same variable) is proportional to another Gaussian. Therefore $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is **Gaussian**, with closed-form mean and variance:

$$\tilde{\mu}_t = \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\, \mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\, \beta_t}{1 - \bar{\alpha}_t}\, \mathbf{x}_0, \qquad \tilde{\beta}_t = \frac{(1 - \bar{\alpha}_{t-1})\, \beta_t}{1 - \bar{\alpha}_t}$$

Step 3: Unconditional reverse. The unconditional reverse $q(\mathbf{x}_{t-1}|\mathbf{x}_t) = \int q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\, q(\mathbf{x}_0|\mathbf{x}_t)\, d\mathbf{x}_0$ is technically a *mixture* of Gaussians (integrating over all possible $\mathbf{x}_0$). However, when β_t is small (which it is by design), each step removes only a small amount of noise, and this mixture is **well-approximated by a single Gaussian**. This is exactly what the learned reverse model $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_\theta, \sigma_t^2\mathbf{I})$ parameterizes.

(d) Generated Sample Distribution as $T \to \infty$

Question

If the model perfectly predicts the true noise, what distribution does the generated sample converge to as $T \to \infty$?

Answer

Setup. If the model perfectly predicts the true noise ($\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) = \boldsymbol{\epsilon}$ for all t), then the learned reverse transition matches the true reverse transition exactly:

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = q(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \quad \text{for all } t = 1, \ldots, T$$

Consequence. The learned reverse process perfectly inverts the forward process. Starting from $\mathbf{x}_T$ and applying the exact reverse transitions recovers the original data distribution:

$$p_\theta(\mathbf{x}_0) = \int p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\, d\mathbf{x}_{1:T}$$

As $T \to \infty$: With an appropriately scheduled $\beta_t \to 0$ (and correspondingly more, smaller steps), the forward process converges to a continuous-time stochastic differential equation (SDE). The terminal distribution $\mathbf{x}_T$ converges exactly to $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The perfect reverse SDE starting from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ then exactly recovers p_{data}:

$$\boxed{p_\theta(\mathbf{x}_0) \longrightarrow p_{\text{data}}(\mathbf{x}_0) \quad \text{as } T \to \infty}$$

In words: the generated samples converge in distribution to the **true data distribution**. This is the theoretical guarantee that makes diffusion models well-founded: with a perfect noise predictor and sufficient steps, the model can generate samples that are statistically indistinguishable from real data.

- End of Answer Guide