Generative Modeling: VAEs, GANs, Score Matching, and Diffusion

18

Chapter Abstract. Generative models must learn a distribution, not only predict a label. On a tiny grayscale image dataset, different training signals ask for different evidence: a decoder reconstructs from latent variables, a discriminator separates real from generated samples, a score model estimates denoising directions, and a diffusion model learns to reverse Gaussian noise. The chapter keeps those signals tied to sampling and debugging decisions in image generation.

18.1 Running Example and Chapter Roadmap

The previous vision chapters asked a model to read an image and return labels, boxes, or masks. Generative modeling asks a different question: can a model produce new examples that look as if they came from the same source as the training data? A classifier estimates a rule like $p(y \mid x)$. A generator tries to model, approximate, or sample from a distribution over images such as $p(x)$, or from a conditional distribution such as $p(x \mid c)$, where c might be a class label or prompt.

Generation is distribution learning plus a sampling rule.

The running example is a tiny dataset of grayscale 8×8 symbol images. Each image is flattened into a vector

$$x \in [0,1]^d, \qquad d = 64.$$

The entries of x are pixel intensities. A value near 0 is dark, a value near 1 is bright, and the whole vector describes one small image. We write the unknown data distribution as p_{data}. In practice we do not see p_{data} directly; we see a finite training set

$$\mathcal{D} = \{x_1, \ldots, x_m\}.$$

Definition 18.1.1 (Generative model) *A generative model is a parameterized mathematical procedure that defines a model distribution p_θ, an approximate model distribution, or a sampling procedure whose outputs are intended to resemble samples from p_{data}. The model is unconditional if it samples x, and conditional if it samples x using an additional input c.*

This definition is broad because modern image-generation systems use several different mathematical routes. A variational autoencoder uses a latent variable z. A generative adversarial network uses a generator and a discriminator. A score model learns a vector field related to $\nabla_x \log p(x)$. A diffusion model adds noise in a known forward process and learns to remove it [111, 68, 197, 87, 200]. On the toy image dataset, these routes ask for different evidence about the same samples: reconstruction, discrimination, score estimation, or denoising after a known noise process.

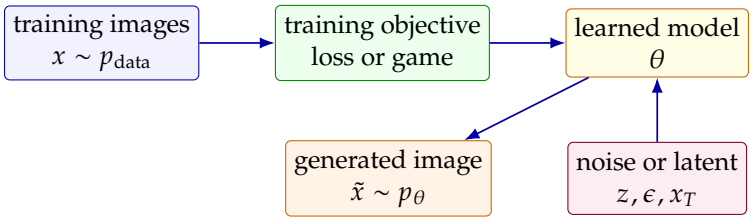

Figure 18.1: A generative model learns from training images, then uses randomness or a noisy state to produce a new image-like sample. The chapter studies the mathematical form of the training objective and the sampling rule.

Pause and predict

If a model always generates the same neat digit-like image, is its sample quality necessarily good? What aspect of the data distribution might it be missing?

The chapter begins with distribution learning and latent variables. It then develops variational autoencoders and generative adversarial networks as two classic answers to the training problem. The second half focuses on score functions and diffusion, because modern image generators often use denoising as the central learning signal. The last sections explain conditional guidance and diagnostics: how to check whether samples are diverse, aligned with a prompt, or suspiciously close to training examples.

18.2 Generative Models as Distribution Learning

A training set gives examples, not the full data distribution. If $\mathcal{D} = \{x_1, \ldots, x_m\}$ contains m tiny images, the empirical distribution places probability $1/m$ on each observed image. A useful image generator should do more than copy those m images. It should produce new images that preserve the important patterns of p_{data}: strokes should be where strokes belong, blank space should remain blank, and different symbol types should appear with reasonable frequency.

Generation is distribution matching, not curation of a sample gallery.

Definition 18.2.1 (Sample distribution) *The sample distribution of a generator is the probability law of its output $\tilde{x}$. If the generator first draws random noise $z \sim p(z)$ and then outputs $\tilde{x} = G_\theta(z)$, the distribution of $\tilde{x}$ is the model's generated distribution.*

This definition matters because a generator can fail even when individual samples look plausible. Suppose the tiny training set has four image types: vertical stroke, horizontal stroke, cross, and loop. A generator that produces only vertical strokes may have sharp samples, but it has poor coverage. It has ignored several modes of the data distribution.

Definition 18.2.2 (Mode and mode collapse) *A mode is a region or pattern with relatively high probability under a distribution. Mode collapse is a generative-model failure in which the model places too much probability on a small number of modes and too little probability on other important modes.*

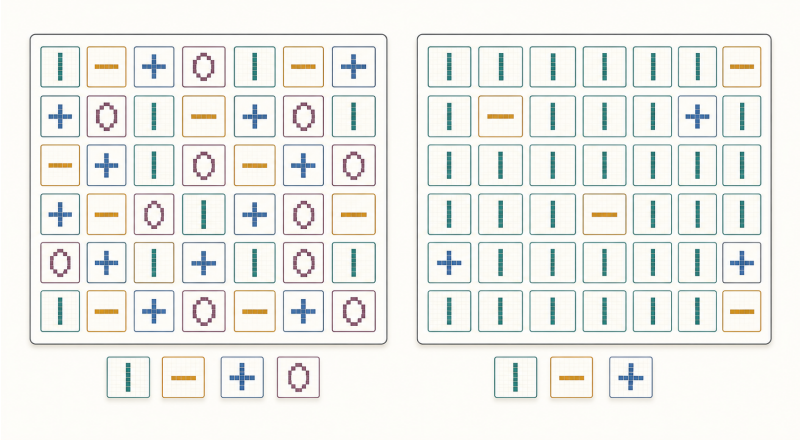

Figure 18.2: A generator can make sharp individual samples while still failing as a distribution. The balanced gallery covers all four symbol types, while the collapsed gallery concentrates on a few types and leaves one mode absent.

Example 18.2.1 (A tiny coverage check) Assume the data distribution puts probability $1/4$ on each of four symbol types. A generator produces the probabilities

$$(0.70,\ 0.20,\ 0.10,\ 0.00).$$

The fourth symbol has disappeared. A sample gallery might still contain many reasonable-looking images, but the distribution is wrong because one mode has zero model probability.

Pause and predict

In the tiny coverage example, the data puts probability $1/4$ on the missing fourth symbol. Predict the total-variation lower bound obtained by testing only that event, and decide whether sharper samples of the other symbols can remove this error.

Remark 18.2.1 (Missing modes create unavoidable distribution error) Let p and q be distributions on a finite set of symbol types, and let

$$d_{\mathrm{TV}}(p, q) = \sup_{A} |p(A) - q(A)|$$

be their total variation distance. If $p(j) > 0$ but $q(j) = 0$ for some type j, then testing total variation on the event $A = \{j\}$ gives

$$d_{\mathrm{TV}}(p, q) \geq |p(A) - q(A)| = |p(j) - 0| = p(j).$$

Thus a generator that assigns zero probability to a real mode cannot exactly match the data distribution.

Different generative-model families expose different mathematical objects. Likelihood-based models define or approximate $p_\theta(x)$ and can be trained by increasing the probability of training data. Implicit generators, such as standard GANs, may provide samples without an easy formula for $p_\theta(x)$. Score-based models learn gradients of log-density instead of the density itself. Table 18.1 summarizes the distinction.

Table 18.1: Generative-model families differ in what they learn directly. The best diagnostic depends on the mathematical object exposed by the model.

Family	Mathematical object	Applied question
Variational autoencoder (VAE)	Lower bound on latent-variable likelihood	Are reconstructions useful and is latent space regular?
GAN	Generator-discriminator game	Do generated samples look realistic and diverse?
Score model	$\nabla_x \log p(x)$ estimate	Which direction moves a noisy image toward data?
Diffusion model	Learned reverse noising process	Can repeated denoising turn noise into an image?

No single number settles generative quality. A likelihood can reward average pixel probability without matching human perception. A sample gallery can hide missing modes. A low denoising loss can still produce poor samples if the sampling procedure is misconfigured. Use objective values, sample inspection, coverage checks, conditioning checks, and nearest-neighbor comparisons together.

18.3 Latent Variables and Decoders

A latent-variable generator begins with a hidden coordinate z. The word "latent" means that z is not directly observed in the training set. In the running example, a two-dimensional z might encode informal factors such as stroke slant and stroke thickness, even if no one labels those factors.

Latent variables are modeling coordinates, not labels found in the dataset.

Definition 18.3.1 (Latent prior and decoder) *A latent prior is a probability distribution $p(z)$ over hidden variables. A decoder is a parameterized map or conditional distribution that converts latent variables into visible data. In deterministic form,*

$$\tilde{x} = g_\theta(z), \qquad z \sim p(z).$$

In probabilistic form, the visible sample is drawn from $p_\theta(x \mid z)$.

The simplest prior is a standard normal distribution. For $z \in \mathbb{R}^2$, this means

$$z \sim \mathcal{N}(0, I_2).$$

If g_θ is continuous, nearby latent points usually produce related outputs. This is why latent interpolation is a useful diagnostic: if a straight line between two latent points produces abrupt, nonsensical image changes, the latent geometry may not be useful for downstream control.

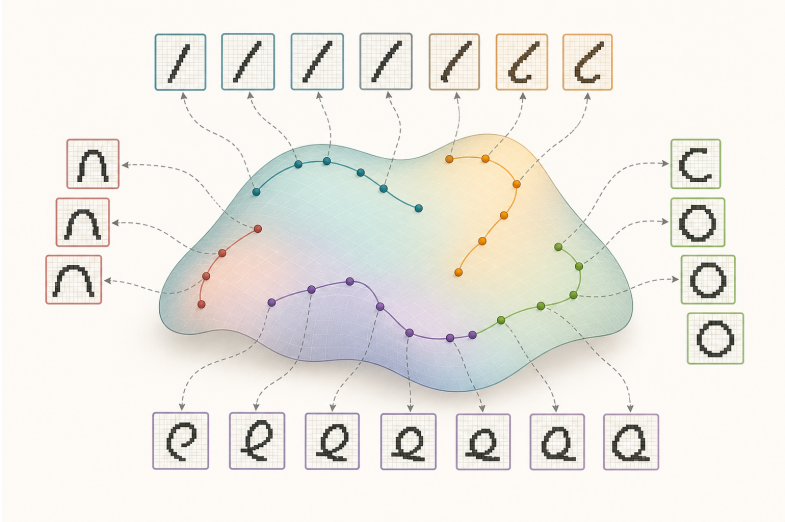

Figure 18.3: A decoder turns latent coordinates into visible samples. Nearby locations in a useful latent space tend to produce related symbols, so paths through the latent plane become an interpolation diagnostic.

Example 18.3.1 (A linear decoder) Let

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}, \qquad g_\theta(z) = Az + b,$$

where

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

For $z = (2, -1)^\top$, the generated three-pixel image vector is

$$g_\theta(z) = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}.$$

Real image decoders are nonlinear and high-dimensional, but the shape logic is the same: latent coordinates are mapped into pixel coordinates.

Definition 18.3.2 (Pushforward distribution) *If $z \sim p(z)$ and $\tilde{x} = g_\theta(z)$, then the distribution of $\tilde{x}$ is the pushforward of $p(z)$ through g_θ. It is the probability distribution induced on generated images by sampling z and applying the decoder.*

For applied work, the important point is not the word "pushforward" but the dependency it names. Changing the prior changes which latent coordinates are visited. Changing the decoder changes how those coordinates become images. Changing both can affect diversity, controllability, and sample realism.

Proposition 18.3.1 (Affine Gaussian decoder moments) *If* $z \sim \mathcal{N}(0, I_k)$ *and* $\tilde{x} = Az + b$, *where* $A \in \mathbb{R}^{d \times k}$ *and* $b \in \mathbb{R}^d$, *then*

$$\mathbb{E}[\tilde{x}] = b, \qquad \mathrm{Cov}(\tilde{x}) = AA^\top.$$

Proof. Linearity of expectation gives

$$\mathbb{E}[\tilde{x}] = A\mathbb{E}[z] + b = b.$$

Since adding b does not change covariance and $\mathrm{Cov}(z) = I_k$,

$$\mathrm{Cov}(\tilde{x}) = A\,\mathrm{Cov}(z)\,A^\top = AA^\top.$$

□

In this affine case, b sets the center of the generated distribution, while A controls the variance and correlations through $AA^\top$. Those two objects correspond to mean appearance and sample diversity in more realistic decoders.

Pause and predict

If two nearby latent points generate unrelated images, what practical workflow might become unreliable: random sampling, latent interpolation, or both?

18.4 Variational Autoencoders

A VAE trains an encoder for inference and a decoder for generation.

A variational autoencoder, or VAE, is a latent-variable generative model with two learned pieces. The previous section described how a decoder can generate an image from a hidden code z. Training needs the reverse direction too: for an observed image x, the code that could have produced it is not observed. The encoder maps an observed image to an approximate distribution over latent variables. The decoder maps latent variables back to an image distribution. The method was introduced independently in closely related forms by Kingma and Welling and by Rezende, Mohamed, and Wierstra [111, 176].

Definition 18.4.1 (VAE encoder and decoder) *For an image x, a posterior is a distribution over latent variables after observing that image. The exact posterior $p_\theta(z \mid x)$ would describe this distribution under the decoder and prior. Bayes' rule would compute it as*

$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z)p(z)}{p_\theta(x)}, \qquad p_\theta(x) = \int p_\theta(x \mid z)p(z)\,dz.$$

The denominator averages over all possible latent variables. For the neural decoders considered here, that integral is the obstacle that makes exact posterior inference unavailable, so a VAE uses an approximate posterior

$$q_\phi(z \mid x)$$

and a decoder likelihood

$$p_\theta(x \mid z).$$

The prior $p(z)$ describes where latent variables should live before seeing a particular image. In this chapter's examples, $p(z) = \mathcal{N}(0, I)$.

The symbols $q_\phi(z \mid x)$ and $p_\theta(x \mid z)$ name distributions, not single arrays. In a neural implementation, the encoder network usually outputs parameters of $q_\phi(z \mid x)$, such as a mean vector and variances. A latent vector z is then a sample from that distribution, or sometimes a summary such as its mean. Similarly, the decoder network outputs parameters of the likelihood $p_\theta(x \mid z)$, such as Bernoulli probabilities or a Gaussian mean and variance. A displayed reconstruction is a sample from that likelihood, or a summary of it, not the decoder distribution itself.

The denominator $p_\theta(x)$ is the marginal likelihood of the observed image under the latent-variable model. Since this integral is usually hard, VAEs train by maximizing a computable lower bound instead.

The algebra below uses one identity repeatedly:

$$D_{\mathrm{KL}}(q(z) \,\|\, p(z)) = \mathbb{E}_{z\sim q}\left[\log \frac{q(z)}{p(z)}\right].$$

The word "variational" refers to optimizing over a chosen family of approximate posterior distributions $q_\phi(z \mid x)$. The VAE trick is to compare the encoder distribution with two different distributions: the prior $p(z)$, which is easy to sample from, and the exact posterior $p_\theta(z \mid x)$, which is usually not available for direct training.

Definition 18.4.2 (Evidence lower bound) *For a VAE, the evidence lower bound, or ELBO, for one example is*

$$\mathcal{L}_{\mathrm{ELBO}}(\theta, \phi; x) = \mathbb{E}_{z\sim q_\phi(\cdot\mid x)}[\log p_\theta(x \mid z)] - D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p(z)).$$

It satisfies

$$\log p_\theta(x) \geq \mathcal{L}_{\mathrm{ELBO}}(\theta, \phi; x).$$

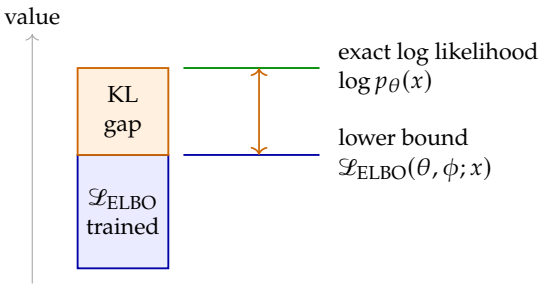

Figure 18.4: The ELBO is below the exact log likelihood by the KL gap to the exact posterior. Maximizing the ELBO raises the bound; the gap disappears only when the encoder distribution matches $p_\theta(z \mid x)$.

The expectation subscript $z \sim q_\phi(\cdot \mid x)$ makes the sampled variable explicit: for this fixed observed image x, draw z from the encoder distribution and average the decoder log-likelihood. Many VAE papers write the same term more compactly as $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)]$; both notations mean the same expectation.

Read the definition as a ledger with one reward and one penalty. The reconstruction log-likelihood term should be large, while the KL term is nonnegative and is subtracted. If the reconstruction term stays fixed, a larger KL term always lowers the ELBO.

For a first pass, the identity below is more important than the algebraic details of the proof. It says the hard-to-compute log likelihood splits into a computable lower bound plus a nonnegative gap. Training uses the lower bound because the exact posterior term in the gap is not available.

Proposition 18.4.1 (ELBO identity) *For any approximate posterior $q_\phi(z \mid x)$ whose support lies where $p_\theta(z \mid x) > 0$ (so q_ϕ does not put probability on latent values that the exact posterior rules out),*

$$\log p_\theta(x) = \mathcal{L}_{\text{ELBO}}(\theta, \phi; x) + D_{\text{KL}}(q_\phi(z \mid x) \,\|\, p_\theta(z \mid x)).$$

Proof. Bayes' rule gives

$$\log p_\theta(z \mid x) = \log p_\theta(x \mid z) + \log p(z) - \log p_\theta(x).$$

Substitute this expression into the KL divergence to the exact posterior:

$$\begin{aligned} &D_{\text{KL}}(q_\phi(z \mid x) \,\|\, p_\theta(z \mid x)) \\ &\quad = \mathbb{E}_{z \sim q_\phi(\cdot|x)} \left[\log q_\phi(z \mid x) - \log p_\theta(z \mid x)\right] \\ &\quad = \mathbb{E}_{z \sim q_\phi(\cdot|x)} \left[\log q_\phi(z \mid x) - \log p(z) - \log p_\theta(x \mid z)\right] + \log p_\theta(x). \end{aligned}$$

The first two log terms form $D_{\text{KL}}(q_\phi(z \mid x) \,\|\, p(z))$, and the reconstruction log-likelihood appears with a minus sign. Therefore

$$D_{\text{KL}}(q_\phi(z \mid x) \,\|\, p_\theta(z \mid x)) = -\mathcal{L}_{\text{ELBO}}(\theta, \phi; x) + \log p_\theta(x).$$

The displayed identity follows. Since KL divergence is nonnegative, the ELBO is indeed a lower bound on $\log p_\theta(x)$. □

Figure 18.4 shows the identity as a vertical bookkeeping picture. The true log likelihood is usually hard to compute, so VAE training raises the computable lower part instead. This is a surrogate objective, not a claim that the lower bound equals the true objective. The identity says that maximizing the ELBO pushes up a guaranteed lower bound on $\log p_\theta(x)$; the bound becomes tight only when the encoder distribution matches the exact posterior. When the encoder family is imperfect, the learned parameters optimize this tractable lower-bound objective rather than exact maximum likelihood.

The first term rewards reconstruction: latent samples drawn from the encoder should let the decoder assign high probability to the original image. The second term regularizes the encoder: $q_\phi(z \mid x)$ should not drift too far from the shared prior. In the tiny-image example, this prevents every training image from receiving an isolated private code that cannot be sampled from later.

Example 18.4.1 (A one-example ELBO calculation) Let $q(z \mid x) = \mathcal{N}(\mu, \sigma^2)$ and $p(z) = \mathcal{N}(0, 1)$ in one dimension. Then

$$D_{\mathrm{KL}}(q \,\|\, p) = \frac{1}{2}\left(\mu^2 + \sigma^2 - \log \sigma^2 - 1\right).$$

If $\mu = 1$ and $\sigma^2 = 2$, then

$$D_{\mathrm{KL}}(q \,\|\, p) = \frac{1}{2}(1 + 2 - \log 2 - 1) \approx 0.653.$$

Suppose the reconstruction term for this example is

$$\mathbb{E}_{z\sim q(\cdot|x)}[\log p_\theta(x \mid z)] = -3.2.$$

The ELBO is therefore

$$\mathcal{L}_{\mathrm{ELBO}} = -3.2 - 0.653 \approx -3.853.$$

With the reconstruction term fixed, increasing the KL term would make this number smaller. With the KL term fixed, improving reconstruction would make it larger.

For a dataset $\mathcal{D} = \{x_1, \ldots, x_m\}$, the practical training objective is to maximize the average ELBO,

$$\max_{\theta,\phi} \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}_{\mathrm{ELBO}}(\theta, \phi; x_i).$$

Implementations usually minimize the negative minibatch estimate of this quantity. This gives the code-facing loss: reconstruction cost plus a KL penalty.

Training also needs gradients through random latent samples. A direct notation such as $z \sim q_\phi(z \mid x)$ hides how the sample changes when the encoder parameters change. The sampling operation itself is not a

differentiable deterministic map from ϕ to z, so ordinary backpropagation cannot pass a pathwise gradient through the instruction "sample from $q_\phi(z \mid x)$." The reparameterization trick moves the randomness into a noise variable whose distribution does not depend on the encoder.

In one dimension, if

$$q_\phi(z \mid x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x)^2),$$

draw $\eta \sim \mathcal{N}(0, 1)$ and set

$$z = \mu_\phi(x) + \sigma_\phi(x)\eta.$$

This produces the same distribution for z, because an affine transform of a standard normal has mean $\mu_\phi(x)$ and variance $\sigma_\phi(x)^2$. In the diagonal multivariate case, the same idea becomes

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \eta, \qquad \eta \sim \mathcal{N}(0, I).$$

During one training step, the sampled η is treated as fixed. Changing $\mu_\phi(x)$ or $\sigma_\phi(x)$ then changes z smoothly, so the gradient of the reconstruction term can backpropagate through the decoder and into the encoder. The stochasticity has not disappeared; it has been separated from the differentiable parameters.

Remark 18.4.1 (Why displayed VAE outputs can look blurry) When $p_\theta(x \mid z)$ treats pixels with a simple independent Gaussian likelihood, the reconstruction term can reward decoder means that average several plausible images. Displaying the mean of several plausible sharp images can produce a blurry reconstruction. This is a modeling and display choice, not a theorem that all latent-variable models or all stochastic samples must be blurry.

The VAE diagnostic checklist follows directly from the ELBO. Inspect reconstruction quality to understand the first term. Inspect latent samples and KL values to understand the second term. If reconstructions are good but random samples are poor, the model may be using latent regions that are not well covered by the prior.

18.5 Adversarial Generative Modeling

Generative adversarial networks, or GANs, use a different training signal from the VAE lower-bound objective. Instead of scoring a reconstruction with a specified decoder likelihood term, a GAN learns a discriminator that judges whether samples look like training data [68]. The generator tries to produce samples that look real. The discriminator tries to distinguish real training images from generated images. This turns generative modeling into a two-player optimization problem.

GAN training judges samples through a learned realism signal, not a fixed pixel loss.

Definition 18.5.1 (Generator and discriminator) *A GAN generator is a map $G_\theta(z)$ from random noise $z \sim p(z)$ to a generated sample. A discriminator is a function $D_\psi(x) \in (0,1)$ whose output is interpreted as the probability that x is real rather than generated.*

The original GAN objective is

$$\min_\theta \max_\psi \mathbb{E}_{x\sim p_{\text{data}}}[\log D_\psi(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D_\psi(G_\theta(z)))].$$

The discriminator maximizes the displayed objective. It wants real images to receive values near 1, making $\log D_\psi(x)$ close to 0, and generated images to receive values near 0, making $\log(1-D_\psi(G_\theta(z)))$ close to 0. The generator controls only the generated term. In the minimax form, it tries to make $\log(1 - D_\psi(G_\theta(z)))$ small; for example, changing $D_\psi(G_\theta(z))$ from 0.1 to 0.9 changes this log term from about -0.105 to about -2.303. Thus the generator is pushing $D_\psi(G_\theta(z))$ toward 1: generated samples should look real to the discriminator.

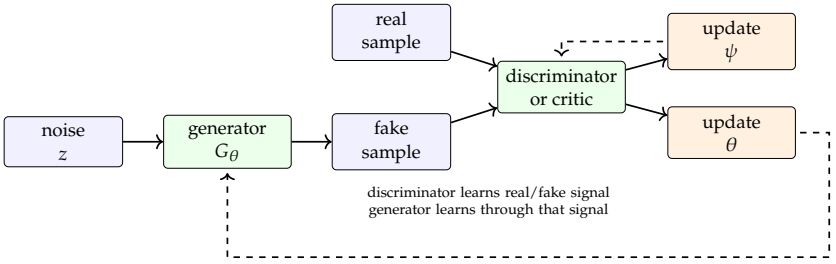

Figure 18.5: GAN training alternates two coupled updates. Real samples and generated samples train the discriminator or critic, while the generator is updated through the learned realism signal assigned to its samples.

Proposition 18.5.1 (Optimal discriminator for a fixed generator) *Let p_g be the generator distribution and suppose both p_{data} and p_g have densities. For fixed G, the discriminator that maximizes*

$$\mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{x\sim p_g}[\log(1 - D(x))]$$

pointwise is

$$D^\star(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}$$

where $p_{\text{data}}(x) + p_g(x) > 0$.

Proof. For each x, maximize

$$f(D) = p_{\text{data}}(x)\log D + p_g(x)\log(1 - D)$$

over $0 < D < 1$. Differentiating gives

$$f'(D) = \frac{p_{\text{data}}(x)}{D} - \frac{p_g(x)}{1 - D}.$$

Setting $f'(D) = 0$ yields $p_{\text{data}}(x)(1 - D) = p_g(x)D$, hence the stated $D^\star(x)$. The second derivative is negative, so this critical point is a maximum. □

The Jensen-Shannon divergence between two distributions p and q is

$$\mathrm{JS}(p \,\|\, q) = \frac{1}{2} D_{\mathrm{KL}}(p \,\|\, m) + \frac{1}{2} D_{\mathrm{KL}}(q \,\|\, m), \qquad m = \frac{1}{2}(p + q).$$

It is nonnegative and equals zero when the two distributions match. Plugging $D^\star$ back into the original minimax objective gives a quantity that equals $-\log 4 + 2\,\mathrm{JS}(p_{\mathrm{data}} \,\|\, p_g)$, so it is minimized when $p_g = p_{\mathrm{data}}$. This idealized calculation explains the target of GAN training, while the finite-sample neural-network game can still be unstable.

Example 18.5.1 (A discriminator-objective calculation) Suppose a discriminator outputs 0.9 and 0.8 on two real images, and 0.3 and 0.4 on two generated images. The discriminator objective for these four examples is

$$\frac{1}{2}(\log 0.9 + \log 0.8) + \frac{1}{2}(\log(1 - 0.3) + \log(1 - 0.4)).$$

If generated images start receiving discriminator values near 0.9, the second term becomes much more negative because $\log(1 - 0.9)$ is a large negative number. The discriminator has been fooled.

GANs are called implicit generative models because sampling is easy but evaluating $p_\theta(x)$ is usually not. Draw z, compute $G_\theta(z)$, and an image appears. But there may be no tractable formula for the density value at a proposed image x. This is one reason GAN diagnostics are different from likelihood-model diagnostics.

Definition 18.5.2 (Implicit generative model) *An implicit generative model specifies a sampling procedure without necessarily providing an easily evaluated probability density $p_\theta(x)$.*

The saturating minimax generator objective can give weak gradients when the discriminator confidently rejects generated samples. A common nonsaturating alternative trains the generator with

$$\mathcal{L}_G^{\mathrm{NS}} = -\mathbb{E}_{z \sim p(z)}[\log D_\psi(G_\theta(z))].$$

This does not change the desired fixed point $p_g = p_{\mathrm{data}}$, but it changes the local optimization pressure seen by the generator.

The original classification game can give weak or uninformative gradients when the generated distribution and the data distribution have little overlap. Wasserstein GANs make a deeper change to the adversarial objective [3]: they replace the probability-valued discriminator with a real-valued critic whose scores estimate a transport distance.

Theorem 18.5.2 (Kantorovich–Rubinstein duality, informal) *For probability distributions p and q on a metric space with finite first moment, the Wasserstein-1 distance has the dual form*

$$W_1(p, q) = \sup_{\mathrm{Lip}(f)\le 1} \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)].$$

The constraint $\mathrm{Lip}(f) \le 1$ means $|f(x) - f(y)| \le \|x - y\|_2$: the critic cannot change faster than the distance between inputs. This duality is the theorem that turns the WGAN critic into a trainable proxy for a transport distance [3, 223].

Definition 18.5.3 (Wasserstein GAN objective) *A Wasserstein GAN trains a generator with*

$$\min_{\theta} \max_{\mathrm{Lip}(f_\psi)\le 1} \mathbb{E}_{x\sim p_{\text{data}}}[f_\psi(x)] - \mathbb{E}_{z\sim p(z)}[f_\psi(G_\theta(z))].$$

The discriminator has become a critic $f_\psi \in \mathbb{R}$, and the inner maximum estimates a transport-distance proxy rather than a classification log-loss game.

The practical difficulty is enforcing the Lipschitz constraint. The original WGAN used weight clipping; later work introduced gradient penalties as a more stable way to keep the critic close to 1-Lipschitz [74]. Other GAN variants, such as least-squares, hinge-loss, and spectral-normalized GANs, can also be read as changes to the adversarial loss or critic regularization.

GANs can produce sharp samples because the discriminator supplies a learned notion of realism rather than a simple pixelwise reconstruction penalty. The same adversarial setup also creates practical risks. Training can oscillate if the generator and discriminator improve at incompatible speeds. The generator can collapse to a few sample types that fool the discriminator while ignoring other modes. The discriminator can become too strong, giving gradients that are not useful for improving the generator.

Pause and predict

If generated class counts are 95% one symbol and 5% all others while the discriminator loss looks low, which diagnostic would expose the failure? Predict what a sample grid might hide if it shows only the best-looking outputs.

In applied work, adversarial loss should be read together with sample grids, coverage checks, class-frequency counts when labels are available, nearest-neighbor inspection, and held-out sample diagnostics. A low generator loss by itself does not prove that the generated distribution matches the data distribution.

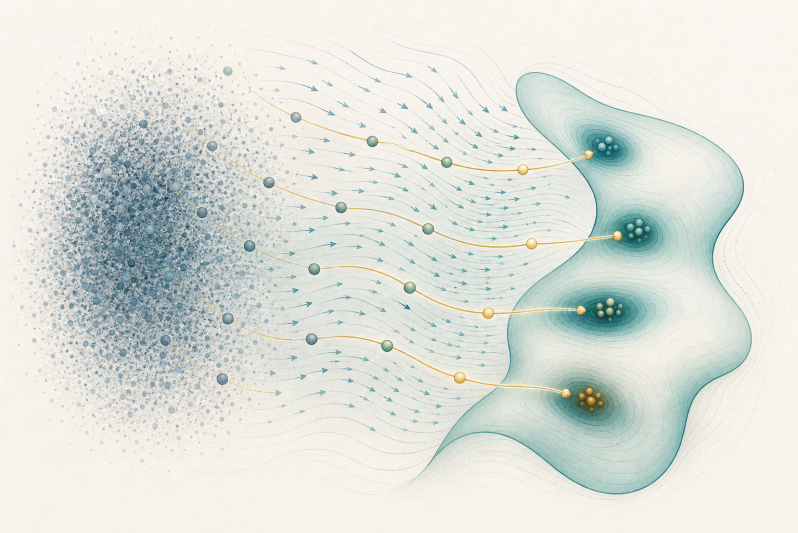

Figure 18.6: Score-based generation can be read as a learned direction field over noisy states. The model supplies local moves that steer samples from diffuse noise toward regions that resemble the data distribution.

18.6 Score Functions and Denoising

A score is a direction field in data space, not a probability value.

A diffusion model needs a local instruction at every noisy point: which way should this point move so that it becomes more like the training data? A density score function gives this kind of instruction. It is not a probability and not a classifier score; it is a direction field in data space.

Formally, the score of a density $p(x)$ is the gradient of its log-density with respect to the input.

Definition 18.6.1 (Density score function) *For a differentiable positive density $p(x)$ on $\mathbb{R}^d$, the density score function is*

$$s(x) = \nabla_x \log p(x).$$

It is a vector in the same space as x. Locally, it points in a direction that increases log-density fastest.

Proposition 18.6.1 (Gaussian score) *For a one-dimensional Gaussian density*

$$p(x) = \mathcal{N}(x; \mu, \sigma^2),$$

the score is

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

Proof. The Gaussian log-density is

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

where C is constant with respect to x. Differentiating gives

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

□

If $x > \mu$, the score is negative and points left. If $x < \mu$, the score is positive and points right. The vector points back toward the high-density center.

Example 18.6.1 (Gaussian score by hand) For $p(x) = \mathcal{N}(0, 4)$, we have $\mu = 0$ and $\sigma^2 = 4$. At $x = 2$,

$$\nabla_x \log p(x) = -\frac{2-0}{4} = -0.5.$$

At $x = -1$, the score is

$$-\frac{-1-0}{4} = 0.25.$$

The signs point both values back toward the mean 0.

Score matching estimates this gradient field without requiring a normalized density in the ordinary maximum-likelihood form [94]. Denoising gives an applied route to a similar object. If a clean image x_0 is corrupted by Gaussian noise to produce x_t, then a denoising model can learn which direction moves x_t back toward a likely clean image. Vincent showed a formal connection between denoising autoencoders and score matching [224]; modern score-based generative modeling develops this idea at many noise levels [199, 200].

There are three related objects to keep separate. The clean-data score $\nabla_x \log p_{\text{data}}(x)$ describes the original image distribution. The conditional noisy score $\nabla_{x_t} \log q(x_t \mid x_0)$ is available once a particular clean image and noising step are fixed. The marginal noisy-data score $\nabla_{x_t} \log q_t(x_t)$ averages over possible clean images and is the field a denoiser learns to approximate from noisy samples alone.

Definition 18.6.2 (Score-matching objective) *For a model score $s_\theta(x)$, the ideal score-matching risk is*

$$\mathbb{E}_{x\sim p}\left[\left\|s_\theta(x) - \nabla_x \log p(x)\right\|_2^2\right].$$

Classical score matching rewrites this objective so the unknown normalized density does not need to be evaluated directly.

Definition 18.6.3 (Denoising objective) *A denoising objective trains a model to recover clean information from a corrupted input. In a simple squared-error form, if*

$$x_t = x_0 + \sigma\epsilon, \qquad \epsilon \sim \mathcal{N}(0, I),$$

then a model $r_\theta(x_t, \sigma)$ may be trained to minimize

$$\mathbb{E}\left[\|x_0 - r_\theta(x_t, \sigma)\|_2^2\right].$$

Diffusion models in this chapter predict the added noise ϵ instead of predicting x_0 directly.

The next two propositions connect noise prediction to score estimation. The first is conditional on the clean image x_0. The second removes that conditioning and explains why squared-error noise prediction learns an average direction for the noisy-data distribution.

Proposition 18.6.2 (Noise prediction determines a conditional noisy-data score) *For*

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \qquad \epsilon \sim \mathcal{N}(0, I),$$

the conditional score of $q(x_t \mid x_0)$ is

$$\nabla_{x_t} \log q(x_t \mid x_0) = -\frac{\epsilon}{\sqrt{1-\bar{\alpha}_t}}.$$

Thus a model that predicts ϵ also gives the score estimate

$$s_\theta(x_t, t) = -\frac{\epsilon_\theta(x_t, t)}{\sqrt{1-\bar{\alpha}_t}}.$$

Proof. Conditioned on x_0, x_t is Gaussian with mean $\sqrt{\bar{\alpha}_t}x_0$ and covariance $(1-\bar{\alpha}_t)I$. The Gaussian score is

$$-\frac{x_t - \sqrt{\bar{\alpha}_t}x_0}{1-\bar{\alpha}_t}.$$

Since $x_t - \sqrt{\bar{\alpha}_t}x_0 = \sqrt{1-\bar{\alpha}_t}\epsilon$, the formula follows. □

This is a conditional score because x_0 is fixed. The next proposition explains how averaging over possible clean images turns noise prediction into a marginal noisy-data score.

Proposition 18.6.3 (Noise MSE learns a marginal noisy-data score) *Let $x_0 \sim p_{\text{data}}$ and*

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \qquad \epsilon \sim \mathcal{N}(0, I).$$

Among all functions $f(x_t, t)$, the squared-error risk

$$\mathbb{E}\left[\left\|\epsilon - f(x_t, t)\right\|_2^2\right]$$

is minimized by $f^\star(x_t, t) = \mathbb{E}[\epsilon \mid x_t]$. If $q_t(x_t)$ is the marginal density of the noisy variable, then

$$\nabla_{x_t} \log q_t(x_t) = -\frac{\mathbb{E}[\epsilon \mid x_t]}{\sqrt{1-\bar{\alpha}_t}}.$$

Proof. For a fixed value of x_t, the conditional squared-error risk $\mathbb{E}[\|\epsilon - v\|_2^2 \mid x_t]$ is minimized at $v = \mathbb{E}[\epsilon \mid x_t]$. This is the same least-squares fact used in ordinary regression: the best prediction under squared error is a conditional mean.

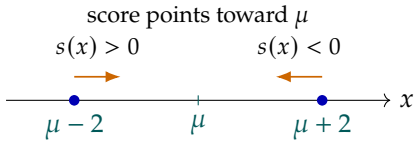

Figure 18.7: For a Gaussian density, the score points back toward the mean. In high-dimensional image space, score models learn analogous directions for noisy images.

For the score identity, write

$$q_t(x_t) = \int q(x_t \mid x_0) p_{\text{data}}(x_0)\, dx_0.$$

Differentiating under the integral gives

$$\nabla_{x_t} q_t(x_t) = \int q(x_t \mid x_0) \nabla_{x_t} \log q(x_t \mid x_0) p_{\text{data}}(x_0)\, dx_0.$$

Dividing by $q_t(x_t)$ gives

$$\nabla_{x_t} \log q_t(x_t) = \int \nabla_{x_t} \log q(x_t \mid x_0) \frac{q(x_t \mid x_0) p_{\text{data}}(x_0)}{q_t(x_t)}\, dx_0.$$

The fraction is the conditional density of x_0 given x_t, so the integral is a conditional expectation. The previous proposition gives $\nabla_{x_t} \log q(x_t \mid x_0) = -\epsilon/\sqrt{1-\bar{\alpha}_t}$, which proves the result. □

Denoising gives local supervision. A training image supplies many related training examples: lightly corrupted versions, heavily corrupted versions, and everything between. A denoiser that works across these noise levels learns an estimate of the noisy-data score field, not only a collection of isolated pixel corrections.

Pause and predict

For $p(x) = \mathcal{N}(0, 4)$, what is the score at $x = 2$? Does it point toward or away from the mean?

18.7 The Diffusion Forward Process

Diffusion models begin with a deliberately destructive process. Starting from a clean image x_0, the forward process adds Gaussian noise over many steps. This process is fixed by the model designer; it is not learned from data. The learned part comes later, when the model tries to reverse the damage.

The forward process is chosen so the reverse problem has supervised targets.

Definition 18.7.1 (Forward diffusion step) *Let $0 < \beta_t < 1$. A forward diffusion step is*

$$q(x_t \mid x_{t-1}) = \mathcal{N}\left(\sqrt{1-\beta_t}\, x_{t-1},\ \beta_t I\right).$$

The number β_t is the noise variance added at step t.

It is common to write

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

Then a useful closed form is

$$q(x_t \mid x_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_t}\, x_0,\ (1 - \bar{\alpha}_t) I\right).$$

Equivalently, one can sample x_t directly as

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I).$$

This formula is central to practical diffusion training because it creates a noisy training example at any time step without simulating all earlier steps [197, 87].

Proposition 18.7.1 (Closed-form noising distribution) *Under the forward diffusion steps above, the marginal distribution of x_t given x_0 is*

$$q(x_t \mid x_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_t}\, x_0,\ (1 - \bar{\alpha}_t) I\right).$$

Proof. Write one step as

$$x_t = \sqrt{\alpha_t}\, x_{t-1} + \sqrt{\beta_t}\, \xi_t, \qquad \xi_t \sim \mathcal{N}(0, I).$$

Assume

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\, x_0 + \sqrt{1 - \bar{\alpha}_{t-1}}\, \eta$$

for $\eta \sim \mathcal{N}(0, I)$. Then the conditional mean is $\sqrt{\alpha_t \bar{\alpha}_{t-1}}\, x_0 = \sqrt{\bar{\alpha}_t}\, x_0$, and the conditional noise variance is

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

The independent Gaussian noise terms combine into a Gaussian with that variance. The case $t = 1$ is the same calculation with $\bar{\alpha}_0 = 1$, so induction proves the claim. □

Example 18.7.1 (One noisy scalar pixel) Let $x_0 = 1$, $\bar{\alpha}_t = 0.64$, and $\epsilon = -0.5$. Then

$$x_t = \sqrt{0.64}(1) + \sqrt{0.36}(-0.5) = 0.8 - 0.3 = 0.5.$$

The noisy pixel is halfway between a bright pixel and a darker value. The training process knows the exact noise ϵ that was added.

Pause and predict

As $\bar{\alpha}_t$ gets smaller, does x_t keep more clean-image signal or more Gaussian noise? Predict using $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$.

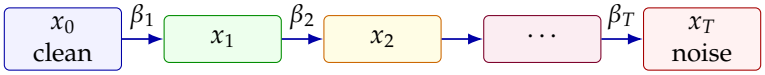

Figure 18.8: The forward diffusion process gradually destroys image structure. Training samples random times t, creates x_t, and asks the model to recover information about the noise or the clean image.

Definition 18.7.2 (Noise schedule and signal-to-noise ratio) *A noise schedule is the sequence $\beta_1, \ldots, \beta_T$. At time t, one useful signal-to-noise ratio is*

$$\mathrm{SNR}(t) = \frac{\bar{\alpha}_t}{1 - \bar{\alpha}_t}.$$

Large $\mathrm{SNR}(t)$ means the clean image still dominates. Small $\mathrm{SNR}(t)$ means noise dominates.

The noising schedule decides how hard each recovery step becomes. If noise is added too quickly, the reverse model must solve a hard recovery problem. If too many tiny steps are used, sampling may become expensive. Diffusion systems balance sample quality, training stability, and the number of denoising steps needed at generation time.

18.8 The Learned Reverse Process

Generation runs in the reverse direction. Start from noise x_T, then apply a learned denoising rule repeatedly until a sample x_0-like image appears. The reverse process is learned because the true reverse conditional $q(x_{t-1} \mid x_t)$ depends on the unknown data distribution.

Sampling quality depends on the reverse rule as well as the trained denoiser.

Definition 18.8.1 (Noise-prediction model) *A diffusion noise-prediction model is a neural network*

$$\epsilon_\theta(x_t, t)$$

trained to estimate the Gaussian noise ϵ used to form x_t from x_0.

The common training objective is

$$\mathbb{E}_{x_0,\epsilon,t}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|_2^2\right], \qquad x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon.$$

Ho, Jain, and Abbeel showed that this simple loss is closely connected to a variational diffusion objective and works well in practice [87]. Later work refined variance learning, schedules, and sampling efficiency [155].

Definition 18.8.2 (One-step reverse Gaussian parameterization) *A DDPM-style reverse step uses the Gaussian parameterization*

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

where μ_θ is built from the noise prediction $\epsilon_\theta(x_t, t)$ and σ_t^2 is fixed or learned.

The posterior formula below is a training-time guide, not the final sampler by itself. If x_0 were known, Gaussian algebra would tell us how to move

from x_t toward x_{t-1}. At generation time x_0 is unknown, so the denoising network first estimates the missing clean image information and then uses that estimate inside the reverse mean.

This is why the posterior is worth deriving before the sampler. It tells us what the ideal one-step reverse move would look like if the clean endpoint were available. The mean will combine the current noisy state x_t with clean state information from x_0; the variance will describe how much random uncertainty remains in the step. The learned sampler keeps the same template but replaces the unavailable x_0 information with a neural estimate.

If the Gaussian algebra is new, read the next proposition as a bookkeeping result: multiplying two Gaussian factors gives another Gaussian, and completing the square identifies its mean and variance. The sampler uses the resulting mean formula; the proof is included to show where the schedule coefficients come from.

Proposition 18.8.1 (Forward posterior for one reverse step) *For the Gaussian forward process with $\alpha_t = 1 - \beta_t$, the posterior conditioned on x_0 is*

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

where

$$\widetilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t$$

and

$$\widetilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}x_t.$$

Proof. Let $u = x_{t-1}$. As a function of u, the posterior is proportional to

$$q(x_t \mid x_{t-1})q(x_{t-1} \mid x_0).$$

These two factors are Gaussian:

$$q(x_t \mid x_{t-1}) = \mathcal{N}(\sqrt{\alpha_t}x_{t-1}, \beta_t I), \qquad q(x_{t-1} \mid x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_{t-1}}x_0, (1-\bar{\alpha}_{t-1})I).$$

Their product has exponent, up to constants not depending on u,

$$-\frac{1}{2}\left[\frac{\|x_t - \sqrt{\alpha_t}u\|^2}{\beta_t} + \frac{\|u - \sqrt{\bar{\alpha}_{t-1}}x_0\|^2}{1 - \bar{\alpha}_{t-1}}\right].$$

The quadratic coefficient in u is

$$\lambda_t = \frac{\alpha_t}{\beta_t} + \frac{1}{1 - \bar{\alpha}_{t-1}} = \frac{1 - \bar{\alpha}_t}{\beta_t(1 - \bar{\alpha}_{t-1})},$$

using $\bar{\alpha}_t = \alpha_t\bar{\alpha}_{t-1}$. Therefore the posterior variance is $\lambda_t^{-1}I = \widetilde{\beta}_t I$. The linear coefficient is

$$\frac{\sqrt{\alpha_t}}{\beta_t}x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1 - \bar{\alpha}_{t-1}}x_0.$$

Multiplying this coefficient by λ_t^{-1} gives the stated mean $\widetilde{\mu}_t(x_t, x_0)$. □

Stage	Available quantities	Computed quantity
Training noising	clean data x_0, sampled noise ϵ, time t	noisy state x_t
Sampling input	noisy state x_t, time t	predicted noise ϵ_θ
Clean estimate	x_t, ϵ_θ, time t	estimated clean state $\hat{x}_0$
Reverse update	x_t, $\hat{x}_0$, schedule terms	mean μ_θ and next state x_{t-1}

Table 18.2: The learned reverse process reuses the Gaussian reverse-step template, but at sampling time the clean image x_0 is unavailable. The denoiser supplies the missing information through a noise estimate.

The result separates the reverse step into two interpretable pieces. The variance $\widetilde{\beta}_t$ is fixed by the noise schedule, so it does not need to be learned in this basic version. The mean $\widetilde{\mu}_t(x_t, x_0)$ is the important direction: it is a schedule-weighted combination of the noisy state x_t and clean-state information from x_0. Sampling is hard because that clean information is exactly what is missing at generation time.

Remark 18.8.1 (Noise prediction gives an x_0 estimate) For the forward relation

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

a noise prediction $\epsilon_\theta(x_t, t)$ induces

$$\hat{x}_0(x_t, t) = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}}.$$

This solves the forward equation for x_0, replacing the unknown noise ϵ by the model prediction $\epsilon_\theta(x_t, t)$.

This estimate is the bridge from the learned object to the sampler. The model does not directly memorize a clean image; it predicts the noise component in the current noisy state, and the algebra converts that prediction into an estimated clean state.

Example 18.8.1 (One-dimensional clean-state estimate) Suppose that at one reverse step $\bar{\alpha}_t = 0.64$, the current noisy value is $x_t = 0.20$, and the model predicts $\epsilon_\theta(x_t, t) = -0.75$. Then

$$\sqrt{\bar{\alpha}_t} = 0.8, \qquad \sqrt{1-\bar{\alpha}_t} = 0.6,$$

so the estimated clean value is

$$\hat{x}_0 = \frac{0.20 - 0.6(-0.75)}{0.8} = 0.8125.$$

The negative predicted noise says that the observed x_t was pushed downward by the forward corruption, so the clean-state estimate moves upward. The reverse mean formula below turns this estimate into the actual one-step sampler update.

Proposition 18.8.2 (DDPM reverse mean from a noise prediction) *Substituting $\hat{x}_0(x_t, t)$ into the forward-posterior mean gives*

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right).$$

Proof. Insert

$$\hat{x}_0 = \frac{x_t - \sqrt{1 - \bar{\alpha}_t}\,\epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}}$$

into $\widetilde{\mu}_t(x_t, x_0)$. The x_t terms are

$$\frac{\beta_t}{1 - \bar{\alpha}_t} x_t + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t,$$

with the first term coming from the substituted $\hat{x}_0$. Factoring out $1/\sqrt{\alpha_t}$, the coefficient of x_t simplifies to

$$\frac{1}{\sqrt{\alpha_t}} \left(\frac{\beta_t}{1 - \bar{\alpha}_t} + \frac{\alpha_t(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \right) = \frac{1}{\sqrt{\alpha_t}},$$

because $\beta_t + \alpha_t(1 - \bar{\alpha}_{t-1}) = 1 - \bar{\alpha}_t$. The noise term has coefficient $-\beta_t / \{\sqrt{\alpha_t}\sqrt{1 - \bar{\alpha}_t}\}$, giving the stated formula. □

The formula is the sampler's mean update. The network's noise estimate determines both the direction and the size of the denoising move from x_t toward x_{t-1}.

Example 18.8.2 (Tiny denoising loss) Suppose the true noise for a two-pixel example is $\epsilon = (1, -2)$, and the model predicts $\epsilon_\theta(x_t, t) = (0.5, -1.5)$. The squared-error noise loss is

$$(1 - 0.5)^2 + (-2 + 1.5)^2 = 0.25 + 0.25 = 0.50.$$

This is ordinary squared error, but the target is the noise that was added, not a class label.

The time input t is essential. At small t, the image is only lightly corrupted; at large t, the image may be almost pure noise. A single network is trained to denoise across many noise levels, so it must know which level it is handling.

Definition 18.8.3 (Sampling trajectory) *A sampling trajectory is the sequence*

$$x_T,\ x_{T-1},\ \ldots,\ x_1,\ x_0$$

produced by a reverse diffusion sampler. Each step uses the denoising model and the chosen sampler formula to move from a noisier state to a less noisy state.

A simple ancestral sampler is:

Step 1. Draw $x_T \sim \mathcal{N}(0, I)$.

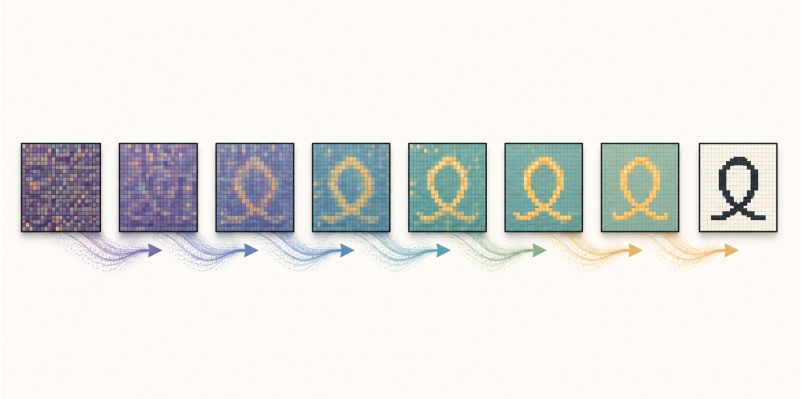

Figure 18.9: Reverse diffusion starts from a noisy state and repeatedly applies a learned denoising rule. The sampling trajectory gradually turns random texture into a structured symbol-like sample.

Step 2. For $t = T, T-1, \ldots, 1$, compute $\epsilon_\theta(x_t, t)$, form a reverse mean $\mu_\theta(x_t, t)$, and draw

$$x_{t-1} = \mu_\theta(x_t, t) + \sigma_t z, \qquad z \sim \mathcal{N}(0, I),$$

with $z = 0$ at the final step if a deterministic last step is desired.

Step 3. Return x_0.

Different samplers implement this trajectory with different numerical rules. DDIM is the first alternative worth naming because it changes the sampling path without requiring a new denoising model [198].

DDIM changes the sampler, not the noise-prediction training target.

A DDPM ancestral sampler treats the next state as a random draw from a Gaussian reverse step. DDIM instead uses the same $\epsilon_\theta(x_t, t)$ and the same $\hat{x}_0(x_t, t)$, but chooses a path that can jump from the current time t to an earlier, less noisy time $s < t$. This matters in practice because sampling can use a shorter subsequence of time steps. The price is that each step covers more distance, so denoiser error has fewer chances to be corrected.

Definition 18.8.4 (DDIM sampler update) *Let $s < t$ be two selected sampling times. Define*

$$\sigma_{t\to s}(\eta) = \eta \sqrt{\frac{1-\bar{\alpha}_s}{1-\bar{\alpha}_t}} \sqrt{1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_s}}, \qquad 0 \le \eta \le 1.$$

A DDIM-style update is

$$x_s = \sqrt{\bar{\alpha}_s}\, \hat{x}_0(x_t, t) + \sqrt{1 - \bar{\alpha}_s - \sigma_{t\to s}(\eta)^2}\, \epsilon_\theta(x_t, t) + \sigma_{t\to s}(\eta) z, \qquad z \sim \mathcal{N}(0, I).$$

When $\eta = 0$, the update is deterministic:

$$x_s = \sqrt{\bar{\alpha}_s}\, \hat{x}_0(x_t, t) + \sqrt{1 - \bar{\alpha}_s}\, \epsilon_\theta(x_t, t).$$

The deterministic formula has a simple interpretation. It decomposes the current state into a predicted clean part and a predicted noise part, then recombines those two pieces at the lower noise level s. The network will be evaluated again at x_s, so the predicted direction can change on the next step.

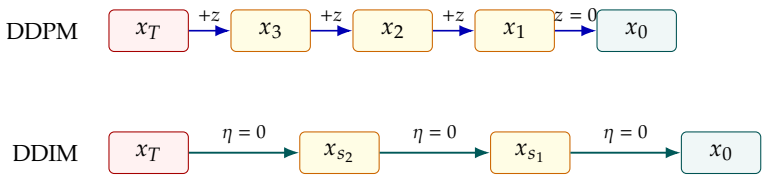

Figure 18.10: DDPM ancestral sampling usually follows many adjacent stochastic reverse steps. Deterministic DDIM sampling can use a shorter selected time grid: each update reuses the denoiser to recombine a predicted clean state with a predicted noise direction at a lower noise level.

Pause and predict

In the deterministic DDIM update, moving from t to $s < t$ makes $\bar{\alpha}_s$ larger than $\bar{\alpha}_t$. Does the coefficient on $\hat{x}_0$ increase or decrease? What should happen to the coefficient on ϵ_θ?

Example 18.8.3 (One deterministic DDIM jump) Use the clean-state estimate from the previous scalar example: $\bar{\alpha}_t = 0.64$, $x_t = 0.20$, $\epsilon_\theta(x_t, t) = -0.75$, and $\hat{x}_0 = 0.8125$. Suppose DDIM jumps to a less noisy time s with $\bar{\alpha}_s = 0.81$ and $\eta = 0$. Then

$$x_s = \sqrt{0.81}(0.8125) + \sqrt{0.19}(-0.75) \approx 0.7313 - 0.3269 = 0.4044.$$

The new state has moved from 0.20 toward the clean estimate 0.8125, but it still retains the predicted noise direction. The next DDIM step recomputes $\epsilon_\theta(x_s, s)$ rather than reusing the old prediction forever.

DDIM is one discrete example of a broader idea: the learned denoiser and the numerical sampler are separable design choices. ODE/SDE solvers make that separation explicit in continuous time.

Theorem 18.8.3 (Reverse-time SDE and probability-flow ODE, informal) *Consider a forward noising SDE*

$$dx = f(x, t)\, dt + g(t)\, dW_t$$

with time-t density $p_t(x)$. Under regularity assumptions such as smooth positive densities, well-posed SDE coefficients, and enough moment or decay control for the score terms, the reverse process can be written, when integrated from T down to 0, as

$$dx = \left[f(x, t) - g(t)^2 \nabla_x \log p_t(x)\right] dt + g(t)\, d\bar{W}_t.$$

There is also a deterministic probability-flow ODE

$$dx = \left[f(x, t) - \frac{1}{2} g(t)^2 \nabla_x \log p_t(x)\right] dt$$

with the same time-marginal densities p_t.

This result explains why score models can support both stochastic samplers and deterministic ODE-style samplers. The learned object is the score field $\nabla_x \log p_t(x)$; the sampler chooses a numerical path that uses that field.

Training teaches a local denoising prediction, while generation applies that prediction repeatedly. If the denoiser is biased at some noise levels, the sampling trajectory can accumulate artifacts [200].

Pause and predict

If the denoiser is accurate near clean images but biased in the first reverse steps from nearly pure noise, predict where sample errors enter the trajectory. Would adding more low-noise refinement steps alone fix that failure?

18.9 Conditional Generation and Guidance

Unconditional generation samples images without an extra request. Conditional generation samples images using information c, such as a class label, a caption, a text embedding, a sketch, a mask, or another image. The model target is no longer just $p(x)$, but $p(x \mid c)$.

Definition 18.9.1 (Conditional generative model) *A conditional generative model uses a conditioning variable c and samples from, approximates, or represents a distribution $p_\theta(x \mid c)$. In the tiny-image example, c could request a vertical stroke instead of a loop.*

For diffusion models, conditioning is usually supplied to the denoising network:

$$\epsilon_\theta(x_t, t, c).$$

The network is asked to predict the noise in a way that is consistent with the condition. If c says "loop", the reverse trajectory should move toward loop-like samples rather than arbitrary symbols.

Guidance changes the sampling trajectory to emphasize the condition. Classifier guidance uses gradients from an external classifier [51]. Classifier-free guidance avoids a separate classifier by training one denoising model that can run both with and without the condition [88].

Proposition 18.9.1 (Conditional score decomposition) *For a noisy-time density $p_t(x, c)$ with $p_t(c) > 0$,*

$$\nabla_x \log p_t(x \mid c) = \nabla_x \log p_t(x) + \nabla_x \log p_t(c \mid x).$$

Proof. Bayes' rule gives

$$\log p_t(x \mid c) = \log p_t(c \mid x) + \log p_t(x) - \log p_t(c).$$

The final term does not depend on x, so its x-gradient is zero. □

Classifier guidance uses this identity: an unconditional score supplies the data-distribution direction, while the classifier gradient supplies the condition-adherence direction.

Definition 18.9.2 (Classifier-free guidance) *Let $\epsilon_\theta(x_t, t, c)$ be the conditional noise prediction and $\epsilon_\theta(x_t, t, \varnothing)$ be the unconditional prediction. With guidance scale $w \geq 0$, classifier-free guidance uses*

$$\widehat{\epsilon} = \epsilon_\theta(x_t, t, \varnothing) + w\left(\epsilon_\theta(x_t, t, c) - \epsilon_\theta(x_t, t, \varnothing)\right).$$

When $w = 0$, the sampler uses the unconditional prediction. When $w = 1$, it uses the conditional prediction. Values $w > 1$ push farther in the direction that separates the conditional prediction from the unconditional prediction. This can improve condition adherence, but it can also reduce diversity or introduce artifacts. The tradeoff is similar in spirit to lowering temperature in a language model: stronger control may produce less varied outputs.

Example 18.9.1 (A guided noise prediction) Let the unconditional prediction be $u = (2, 1)$, the conditional prediction be $v = (1, 3)$, and $w = 2$. Then

$$\widehat{\epsilon} = u + 2(v - u) = (2, 1) + 2(-1, 2) = (0, 5).$$

The guided prediction is not merely the conditional prediction. It extrapolates away from the unconditional direction.

Remark 18.9.1 (Latent diffusion) Many high-resolution systems run diffusion in a compressed latent space rather than directly in pixel space. Latent diffusion keeps the same denoising logic but changes the space in which x_t lives, reducing computational cost for large images [178].

Sweep w when checking guidance. If w is too small, samples may ignore the condition. If w is too large, samples may become repetitive, saturated, or distorted. A single attractive sample is not enough; the sweep reveals the tradeoff.

18.10 Evaluation, Diagnostics, and Failure Modes

A generative model is a distribution, not only an image maker. In the running digit example, a beautiful "7" can hide missing "3"s; a good average loss can hide bad class conditioning; an impressive sample can still be a near-copy of a training image. The diagnostic question is which failure would matter for the intended use.

For VAEs, start with reconstructions, random samples, and the KL term. A model that reconstructs well but samples poorly may have learned an encoder-decoder pair whose useful latent codes are not well matched to the prior. A model with very small KL may ignore the latent variable. A model with very large KL may use an irregular latent space that is hard to sample from.

For GANs, inspect both realism and coverage. The discriminator loss and generator loss are useful during training, but they do not directly report whether the generated distribution covers the data distribution. A frequency

Failure mode	Symptom	Useful check
Poor sample quality	blurry, distorted, or implausible images	sample grids and human inspection
Poor coverage	missing symbol types or repeated outputs	class/count tables and precision-recall style checks
Memorization	generated images nearly copy training images	nearest-neighbor inspection
Bad conditioning	prompt or class is ignored	condition sweep and matched labels
Over-guidance	saturated or repetitive outputs	guidance-scale sweep
Training mismatch	low training loss but poor samples	evaluate by noise level and sampler setting

Table 18.3: Generative diagnostics should be matched to the failure mode. No single metric detects every problem.

table over known classes in the running example can reveal mode collapse. For natural images, learned metrics such as the Frechet Inception Distance were introduced to compare generated and real feature distributions [85], but they should be treated as summaries rather than complete evidence.

Example 18.10.1 (A tiny coverage count) Suppose the running dataset has three equally important symbol types: vertical strokes, loops, and crosses. A generator produces 120 samples:

$$70 \text{ vertical strokes}, \qquad 50 \text{ loops}, \qquad 0 \text{ crosses}.$$

A balanced generator would produce about $120/3 = 40$ samples of each type. The samples may look clean, but the count table shows a missing mode.

For diffusion models, separate training diagnostics from sampling diagnostics. The noise-prediction loss should be checked across time steps or noise levels. If high-noise prediction is poor, early reverse steps may point in the wrong direction. If low-noise prediction is poor, final samples may keep small artifacts. Sampling also depends on the number of reverse steps, the sampler rule, and any guidance scale.

Definition 18.10.1 (Nearest-neighbor memorization check) *A nearest-neighbor memorization check compares each generated sample $\tilde{x}$ with nearby training examples under a chosen distance, such as pixel distance or learned feature distance. If many generated samples are extremely close to training images, the model may be copying rather than generalizing.*

Conditional generation adds another diagnostic axis. A generated image can be high quality but wrong for its condition. In the tiny-image example, a clean loop is not a good answer to the condition "vertical stroke." In text-to-image systems, the same issue appears as prompt noncompliance, missing objects, or incorrect relationships. The relevant check is not only image realism, but alignment between x and c.

No single artifact is enough evidence. Objective values monitor training, samples expose visual quality, coverage checks reveal missing modes,

nearest neighbors test for memorization, and condition sweeps show prompt or label sensitivity.

18.11 Summary

Generative modeling learns to produce data-like samples. The central object is a distribution over images, not only a prediction for a given image. This shift changes the questions an applied practitioner must ask: What randomness is sampled? What objective is optimized? What does the model predict during training? Which sampling rule turns the learned object into visible images?

Latent-variable models introduce hidden coordinates z. A decoder maps those coordinates into data space. VAEs add an encoder and train an ELBO with two parts: a reconstruction term and a KL regularization term. The reconstruction term keeps generated images tied to the input; the KL term keeps latent codes compatible with a shared prior.

GANs train a generator through a discriminator. The adversarial objective can encourage sharp realistic samples, but the training game can be unstable and can hide mode collapse. Wasserstein GANs replace the probability discriminator with a Lipschitz critic so that the adversarial game estimates a transport-distance proxy. GAN evaluation therefore needs coverage checks, not only sample sharpness.

Score functions point in directions that increase log-density. Denoising connects this idea to images: a model can learn how to move corrupted examples back toward likely clean examples. Diffusion models formalize this by adding Gaussian noise in a fixed forward process and learning a reverse process. The common practical objective predicts the added noise with squared error. Sampler choice then matters: DDPM-style ancestral sampling injects Gaussian noise along the reverse path, while DDIM can reuse the same denoiser for deterministic or partly stochastic jumps across a selected time grid.

Conditional generation uses a condition c, such as a class or prompt. Classifier-free guidance combines unconditional and conditional denoising predictions. Larger guidance can improve condition adherence, but it may reduce diversity or create artifacts. The final check is diagnostic: inspect quality, coverage, memorization, conditioning, and sampler choices together.

18.12 Exercises

Work through these ten generative-modeling exercises. When a prompt asks for a calculation, include enough intermediate values to make the answer checkable.

Exercise 1. A tiny training distribution has four symbol types with probabilities $(0.25, 0.25, 0.25, 0.25)$. A generator produces symbol-type probabilities $(0.50, 0.50, 0, 0)$. Which modes are missing, and why can a sample gallery be misleading in this case?

Exercise 2. Let $p(x) = \mathcal{N}(0, 4)$. Compute $\nabla_x \log p(x)$ at $x = -2$, $x = 0$, and $x = 2$. State the direction each score points.

Exercise 3. Let $z \sim \mathcal{N}(0, I_2)$ and $g(z) = Az + b$, where

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 1 & -1 \end{bmatrix}, \qquad b = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}.$$

Compute $g(z)$ for $z = (1, 0)^\top$ and for $z = (0, -1)^\top$.

Exercise 4. For a one-dimensional VAE encoder $q(z \mid x) = \mathcal{N}(\mu, \sigma^2)$ and prior $p(z) = \mathcal{N}(0, 1)$, use

$$D_{\mathrm{KL}}(q \,\|\, p) = \frac{1}{2}(\mu^2 + \sigma^2 - \log \sigma^2 - 1).$$

Compute the KL term for $\mu = 1$, $\sigma^2 = 2$.

Exercise 5. A VAE has reconstruction term -3.2 and KL term 0.7 for one example. Compute the ELBO. If the KL term increases while the reconstruction term stays fixed, does the ELBO improve or worsen?

Exercise 6. A discriminator outputs $0.8, 0.6$ on two real samples and $0.3, 0.2$ on two generated samples. Compute the average discriminator objective

$$\frac{1}{2}(\log D(x_1) + \log D(x_2)) + \frac{1}{2}(\log(1 - D(\tilde{x}_1)) + \log(1 - D(\tilde{x}_2))).$$

Use natural logarithms. Report a decimal approximation.

Exercise 7. For a fixed generator with $p_{\mathrm{data}}(x) = 0.75$ and $p_g(x) = 0.25$ at one point x, compute the optimal discriminator value $D^\star(x)$. Then repeat for $p_{\mathrm{data}}(x) = p_g(x)$ and explain the result.

Exercise 8. Given $x_0 = (1, -1)$, $\bar{\alpha}_t = 0.81$, and $\epsilon = (2, 0)$, compute

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon.$$

Report the two-vector result. Report a decimal approximation.

Exercise 9. The true diffusion noise is $\epsilon = (1, -2, 0)$, and the model predicts $\epsilon_\theta = (0, -1, 1)$. Compute the squared-error noise-prediction loss $\|\epsilon - \epsilon_\theta\|_2^2$. If $\bar{\alpha}_t = 0.75$, convert the noise prediction into the corresponding score estimate.

Exercise 10. Let the unconditional noise prediction be $u = (2, 1)$, the conditional prediction be $v = (1, 3)$, and the guidance scale be $w = 1.5$. Compute

$$\widehat{\epsilon} = u + w(v - u).$$

Explain in one sentence what increasing w usually trades off.

18.13 Homework: Latent Variables, KL Terms, and Denoising Losses

This homework uses SymPy to compute small symbolic pieces of the chapter. The goal is to connect Gaussian scores, VAE KL terms, forward diffusion, and denoising loss without using a deep-learning library [145].

Submission model. Submit the completed starter script, the main hand derivation, the SymPy output needed to verify it, and a short interpretation. External computer-algebra tools may be used only as optional sanity checks.

SymPy reminder. The full SymPy Survival Guide appears in Chapter 1's symbolic foundations homework. For this homework, the same patterns for symbols, substitution, simplification, exact rational values, and `sp.Matrix([...])` are enough.

Mathematical setup

Use a one-dimensional Gaussian density with mean μ and variance σ^2:

$$\log p(x) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{(x-\mu)^2}{2\sigma^2}.$$

Use the one-dimensional VAE KL term

$$D_{\mathrm{KL}}(q \,\|\, p) = \frac{1}{2}(\mu_q^2 + \sigma_q^2 - \log \sigma_q^2 - 1),$$

where $q(z) = \mathcal{N}(\mu_q, \sigma_q^2)$ and $p(z) = \mathcal{N}(0, 1)$.

Tasks

Task 1. Define symbolic variables for x, μ, and σ^2.

Task 2. Build the Gaussian log-density above.

Task 3. Differentiate the log-density with respect to x to obtain the score.

Task 4. Substitute $\mu = 0$, $\sigma^2 = 4$, and $x = -2, 0, 2$ into the score.

Task 5. Define symbolic μ_q and σ_q^2, then build the VAE KL term.

Task 6. Build a toy VAE objective from reconstruction loss r and KL term: $\mathcal{L}_{\mathrm{ELBO}} = -r - D_{\mathrm{KL}}(q \,\|\, p)$.

Task 7. Build the forward-diffusion expression

$$x_t = \sqrt{\bar{\alpha}}\,x_0 + \sqrt{1-\bar{\alpha}}\,\epsilon.$$

Task 8. Substitute $x_0 = 1$, $\bar{\alpha} = 0.81$, and $\epsilon = 2$.

Task 9. Define a two-coordinate denoising loss

$$(\epsilon_1 - \hat{\epsilon}_1)^2 + (\epsilon_2 - \hat{\epsilon}_2)^2.$$

Task 10. Differentiate that denoising loss with respect to $\hat{\epsilon}_1$ and explain what the derivative is trying to reduce.