Variational Inference

2026-05-09 · quiet orange Antilope

EMBEDDINGS AND LATENT FEATURE SPACES.

The Why

THE CENTRAL PROBLEM OF UNSUPERVISED LEARNING is representation learning: finding a mapping from raw observations to a latent feature space in which the underlying factors of variation become explicit. Bengio, Courville, and Vincent[1] argue that the quality of a representation determines the success of any downstream task, and that a good representation disentangles the factors that explain the data.

PLATO'S CAVE ALLEGORY describes prisoners who see only shadows on a wall, never the objects that cast them. The shadows are high dimensional and entangled; the true forms behind them are few and independent. Dimensionality reduction, as we have seen, tries to recover those forms from the shadows. This chapter asks two follow-up questions: can we organise the recovered forms into a structured, reusable feature space, and once we have that space, can we cast new, plausible shadows we have never observed?
[1]

PCA gave us a first answer: project onto the directions of maximum variance. But PCA is linear; it cannot recover structure that lies on a curved manifold. What we lack is a parametric encoder that can be applied to new, unseen data or reused as a feature extractor on nonlinear feature spaces.

AUTOENCODERS close that gap. Hinton and Salakhutdinov[2] showed that a neural network can learn a nonlinear, low-dimensional embedding z from which a second network reconstructs $\hat{x}$, outperforming PCA on complex data. The encoder $\theta_E : \mathbb{R}^d \to \mathbb{R}^k$ and the decoder $\theta_D : \mathbb{R}^k \to \mathbb{R}^d$ are trained jointly to minimise reconstruction error $\|x - \theta_D(\theta_E(x))\|^2$. The bottleneck forces the network to discover a compact representation.
[2]

STRUCTURE REQUIRES A PROBABILISTIC FORMULATION.

If we replace the deterministic bottleneck with a distribution, and regularise that distribution toward a known prior, the latent space becomes smooth and complete: every region maps to a plausible output, and nearby embeddings correspond to similar inputs. Kingma

A PRIOR is the distribution we assume over the latent space before seeing any data.

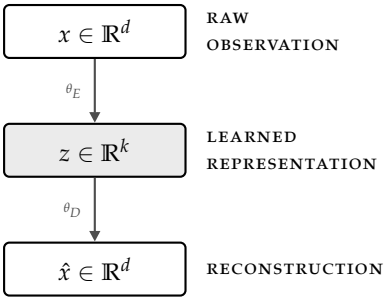


Figure 1: Autoencoder architecture. The encoder θ_E maps the input to a low-dimensional representation; the decoder θ_D reconstructs the input from it. The bottleneck forces the network to discover a compact latent feature space.

and Welling[3] formalised this idea as the Variational Autoencoder, deriving a principled training objective from variational inference that balances faithful reconstruction against latent space regularity.

[3]

ALL OF THIS MUST HAPPEN WITHOUT LABELS. Supervised learning can organise a feature space by asking "which class does this input belong to?" and adjusting representations until the answer is easy to read off. An unsupervised model has no such signal. Instead, it must derive structure from the data alone, using two complementary pressures: the requirement to reconstruct the input faithfully, and the requirement to keep the latent space close to a known prior. Reconstruction forces the embedding to retain information; the prior forces the embedding to be organised. Neither pressure requires a human to annotate a single example.

IN ORDER TO MOVE FROM RECONSTRUCTION-ONLY EMBEDDINGS TO WELL-ORGANISED LATENT FEATURE SPACES we have good reason to find ways to,

- formulate a learning objective that rewards both faithful reconstruction and a smooth, regular latent space.
- train a model that outputs distributions rather than points, so that the latent space has no undefined gaps.
- control how much structure the latent space carries, and understand how that choice affects the quality of the learned representation.

Hands-On Experience

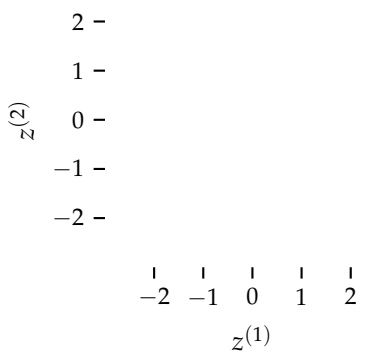
FORM GROUPS OF TWO: an encoder and a decoder. Prepare five digit cards showing **0**, **1**, **3**, **7**, and **8**. Shuffle them face down on the table.

STEP 1: DESIGN YOUR FEATURES. The encoder picks two properties of digit shapes and writes them down. These are your two latent features $z^{(1)}$ and $z^{(2)}$. For example: $z^{(1)}$ = "number of closed loops" (0 has one, 8 has two, 1 has none) and $z^{(2)}$ = "has a straight vertical stroke" (1 yes, 7 yes, 0 no); make sure to come up with your own The decoder does not see these definitions.

STEP 2: PLAY. Each round, the encoder flips one card, looks at the digit, and writes two numbers $(z^{(1)}, z^{(2)})$ on a slip of paper. The decoder receives only the slip and guesses which digit it is. Then reveal the card. The decoder now knows: "the embedding $(1, 0)$ meant the digit 3." Round by round, the decoder builds a mental map of what the embeddings mean. When all five cards are done, shuffle and keep going.

STEP 3: OBSERVE. In early rounds, the decoder guesses blindly. With each reveal, the decoder learns what the embeddings mean. At some point the reconstruction loss drops to zero: the decoder has learned the encoder's representation from examples alone.

STEP 4: RECORD AND COMPARE. Plot your five embeddings in the coordinate system below.



STEP 5: SWAP DECODERS. Find another group. Send your decoder to them, take theirs. Play one more pass with the swapped decoder.

SCORING. After the decoder guesses, flip the card. Correct digit = 0, wrong digit = 1. Sum over all rounds: that is your reconstruction loss.

Figure 2: Your latent space. Plot the five digit embeddings at the coordinates $(z^{(1)}, z^{(2)})$ your encoder chose.

- How many rounds did the decoder need before reconstructions became reliable?
- Which digits did the decoder confuse most often? What does that tell you about your feature choice?

The swapped decoder has learned a different codebook: in their group, $(1, 0)$ might have meant "7," in yours it means "3." The reconstruction loss jumps back up. The problem is not that either representation is wrong; both worked within their own group. The problem is that nothing forced the two groups to organise their embeddings the same way.

STEP 6: SHRINK THE BOTTLENECK. Drop to a single latent feature $z^{(1)}$. The encoder must pick the one property that matters most; the decoder has only one number to work with. Play a full pass and record the new reconstruction loss.

STEP 7: DISCRETISE. Go back to two features, but restrict each to binary values (0 or 1). The encoder can no longer spread digits across a continuous range and must find features that cleanly partition the digit set with just two bits.

STEP 8: ADD DIGITS. Keep two binary features, but add the cards **2**, **4**, **5**, and **9**. A representation that separated five digits may collapse when it must also distinguish eight. But the extra classes also expose ad hoc feature choices: properties that happened to work for a small set get overwhelmed, and the encoder is pressured toward genuinely structural features (loops, strokes, intersections) that scale.

STEP 9: SWAP DECODERS AGAIN. Find a different group than in Step 5. Exchange decoders and play one pass with the expanded digit set. Compare the reconstruction loss to the Step 5 swap: did the stress tests in Steps 6 to 8 push both groups toward more compatible representations.

After each decoder swap, collect the chosen latent features from all groups on the board. Are the feature sets converging across the room? Which features survived the stress tests and which were abandoned?

EACH STEP HURTS RECONSTRUCTION but forces the encoder to find features that generalise across inputs rather than overfit to a narrow set. This tension between reconstruction fidelity and generalisation through compression or broader coverage is exactly the trade-off that the VAE formalises. And a trade-off you know from supervised learning as well, overfitting versus generalization.

THIS IS THE REPRESENTATION LEARNING PROBLEM. Every choice of features defines a different geometry: different neighbours, different distances. Within one encoder-decoder pair, reconstruction works. Across pairs, the embeddings are incompatible.

THE FUNDAMENTAL QUESTION is: can we add a shared convention to the latent space, so that the embeddings are not only good for

TWO OBJECTIVES, ONE GAME. Steps 2 to 4 trained the reconstruction part: the decoder learned to recover digits from embeddings. Steps 5 and 9 exposed the missing regularisation: without a shared convention for what the embeddings mean, a new decoder cannot read them. Steps 6 to 8 showed that compression and broader coverage force better features. A VAE adds exactly this second pressure: it forces every encoder to arrange its embeddings according to a shared prior, so that any decoder can use them.

reconstruction but also follow a common structure that any decoder could interpret?

The Learning Objectives of this lecture:

- The Autoencoder Training Cycle. Architecture, forward pass, reconstruction loss, backpropagation, and what the bottleneck forces the network to learn.
- From Autoencoders to VAEs. Why reconstruction alone leaves the latent space unstructured, and what a probabilistic formulation buys us.
- The Evidence Lower Bound. The training objective that balances faithful reconstruction against latent space regularity.
- The Reparameterisation Trick. How to backpropagate through a stochastic sampling step.
- VAE Training and Bottleneck Design. Practical choices that shape the latent space: loss functions, latent dimensionality, and numerical stability.
- Exploring the Latent Space. Interpolation and generation as tests of representation quality.
- VAE Variants. How they each change one structural assumption.
- Feature Extraction, Transfer, and Distillation. Using the trained encoder as a backbone for downstream tasks.

The Autoencoder

Follows the notion of a bottleneck architecture: an encoder that compresses the input into a low-dimensional embedding, and a decoder that reconstructs the input from that embedding. Due to the bottleneck, the encoder must learn to extract the most salient features of the input, and the decoder must learn to use those features to reconstruct the input as losslessly as possible.

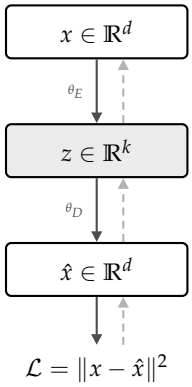


Figure 3: Autoencoder training cycle. Solid arrows: forward pass ($x \to z \to \hat{x} \to \mathcal{L}$). Dashed arrows: gradient flow back through decoder and encoder.

THE ENCODER $\theta_E \colon \mathbb{R}^d \to \mathbb{R}^k$ compresses a d-dimensional input x into a k-dimensional embedding z, with $k \ll d$.

NOTATION. θ_E denotes the encoder; θ_D denotes the decoder. Both are neural networks trained jointly.

THE DECODER $\theta_D \colon \mathbb{R}^k \to \mathbb{R}^d$ takes z and attempts to reconstruct the original input as $\hat{x}$.

THE FORWARD PASS maps a single training example through both networks. Given an input x, the encoder produces the embedding:

$$z = \theta_E(x)$$

The decoder then reconstructs:

$$\begin{aligned} \hat{x} &= \theta_D(z) \\ &= \theta_D(\theta_E(x)) \end{aligned}$$

THE RECONSTRUCTION LOSS measures how accurate the decoder reconstructs the input. The standard choice is the mean squared error over all dimensions:

$$\begin{aligned} \mathcal{L}(\theta_E, \theta_D;\, x) &= \|x - \hat{x}\|^2 \\ &= \sum_{j=1}^{d} (x_j - \hat{x}_j)^2 \end{aligned}$$

BACKPROPAGATION is the backbone of supervised learning, and it applies here unchanged. The loss $\mathcal{L}$ depends on θ_D (through $\hat{x}$) and on θ_E (through z, which feeds into $\hat{x}$). The chain rule gives:

$$\frac{\partial \mathcal{L}}{\partial \theta_D} = \frac{\partial \mathcal{L}}{\partial \hat{x}} \cdot \frac{\partial \hat{x}}{\partial \theta_D}$$

for the decoder, and

$$\frac{\partial \mathcal{L}}{\partial \theta_E} = \frac{\partial \mathcal{L}}{\partial \hat{x}} \cdot \frac{\partial \hat{x}}{\partial z} \cdot \frac{\partial z}{\partial \theta_E}$$

for the encoder. The gradient signal flows from the loss, through the decoder, through z, and into the encoder. Both networks are updated from the same single loss.

THE PARAMETER UPDATE applies standard gradient descent (or a variant such as Adam):

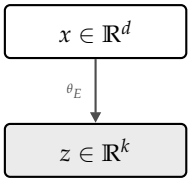
$$\theta_E \leftarrow \theta_E - \eta \frac{\partial \mathcal{L}}{\partial \theta_E}$$

$$\theta_D \leftarrow \theta_D - \eta \frac{\partial \mathcal{L}}{\partial \theta_D}$$

where η is the learning rate. One forward pass, one backward pass, one update: that is a single training step. Repeating this over minibatches and epochs, the encoder and decoder co-adapt until the reconstruction error converges.

THE BOTTLENECK. Because $k \ll d$, the embedding z cannot simply copy x. The encoder must decide which aspects of the input to preserve and which to discard. This forced compression is what generalizes the encoder into a feature extractor.[4]

THE TRAINED ENCODER IS A FEATURE EXTRACTOR, OR DIMENSIONALITY REDUCER. Once training converges, the decoder can be discarded. The encoder alone maps any new input x to an embedding z that captures the factors of variation learned during training. This embedding can serve as input to a classifier, a clustering algorithm, or a retrieval system. The autoencoder has learned a representation; the question is whether that representation is generalized.



CONNECTION TO PCA. If the encoder and decoder are both linear (no activation functions) and the loss is MSE, the autoencoder recovers the same subspace as PCA[5]. The k embedding dimensions span the top-k principal component directions. Adding nonlinear activations allows the network to capture structure that PCA cannot: curved manifolds, discrete clusters, hierarchical features.

[4] WHY DEEP (LEARNING)? Each layer applies a linear transformation followed by a nonlinear activation. A single hidden layer can in principle approximate any function, but may need impractically many neurons and is hard to learn. Deeper networks compose simple nonlinearities into increasingly abstract features: the first layer detects edges, the second combines edges into textures, the third recognises parts. Each layer reuses the representations below it, so the network expresses complex structure with far fewer parameters than a shallow one. Hinton and Salakhutdinov showed that a deep autoencoder with the same total capacity as a shallow one produces qualitatively better embeddings, because hierarchical composition is a more efficient way to represent the kind of structure found in real data.

[5]

From Autoencoders to Variational Autoencoders

THE LATENT SPACE HAS NO GEOMETRIC GUARANTEES. The objective only rewards reconstruction; nothing constrains how embeddings are arranged. Nearby points may decode to unrelated outputs, gaps between embeddings are undefined, and the layout can change arbitrarily across training runs. The representation captures enough to reconstruct, but it does not organise the factors of variation in a way that supports interpolation, sampling, or transfer.

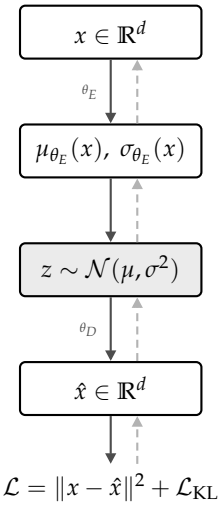
TO FIX THIS, WE ADD A CONSTRAINT. In a plain autoencoder, the encoder's output layer produces a single vector $z \in \mathbb{R}^k$: one fixed point per input. In a Variational Autoencoder, the output layer produces two vectors instead: a mean $\mu \in \mathbb{R}^k$ and a variance $\sigma^2 \in \mathbb{R}^k$. Together they define a Gaussian distribution over the latent space:

$$q_{\theta_E}(z|x) = \mathcal{N}(\mu, \sigma^2)$$

During training, we do not pass μ directly to the decoder. Instead we sample from that distribution:

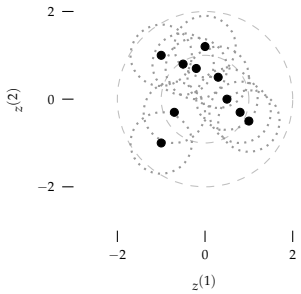
$$z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

So the decoder sees a slightly different z every time, even for the same input.



WE ENFORCE THAT BY ADDING A LOSS a KL divergence $\mathcal{L}_{\text{KL}}(q_{\theta_E}(z|x)\|p(z))$, computed for each input separately. It measures how far that input's

STANDARD AUTOENCODERS learn $x \to z \to \hat{x}$, minimising $\|x - \hat{x}\|^2$.

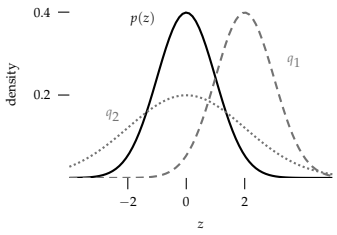


VAE latent space. Each small circle is one input's encoder distribution $q_{\theta_E}(z|x_i)$; the dot marks its mean μ. The large dashed circles show the prior $p(z) = \mathcal{N}(0, I)$ at 1σ and 2σ. The KL term pushes each small circle toward unit variance and its mean toward the origin, so that the mixture of all encoder distributions approximates the prior.

Figure 4: VAE architecture. Compare with Figure 3: the deterministic bottleneck is replaced by distribution parameters μ, σ and a stochastic sample. The loss combines reconstruction fidelity with KL regularisation toward the prior.

encoder distribution $\mathcal{N}(\mu, \sigma^2)$ is from the prior $p(z) = \mathcal{N}(0, I)$. The KL between two Gaussians penalises two things: a mean μ that is far from the origin, and a variance σ^2 that deviates from one. So for each input, the KL term pulls the mean toward zero and the variance toward unity.

IF THE KL TERM ACTED ALONE, every input would collapse onto the same distribution $\mathcal{N}(0, I)$, and the decoder could not tell any two inputs apart. The reconstruction term fights back: it needs different inputs to have distinguishable embeddings, so it pushes their means apart. Training finds a compromise. Each input gets a distribution that is close to the prior but not identical to it: the means spread out just enough for the decoder to reconstruct, and the variances stay close to one. When all these slightly shifted, roughly unit-variance Gaussians are added together, their aggregate approximates the prior. That is how the prior fills in the gaps between embeddings and gives the latent space its geometric structure. Together they turn the encoder into a feature extractor whose embeddings are both informative and geometrically well behaved: nearby embeddings correspond to similar inputs, and every region of the space is meaningful.



KL PENALTY. $p(z) = \mathcal{N}(0, 1)$ is the prior (solid). $q_1 = \mathcal{N}(2, 1)$ (dashed) is shifted; $q_2 = \mathcal{N}(0, 4)$ (dotted) is widened. Spreading mass (q_2, $\mathcal{L}_{\text{KL}}$=0.81) costs more than translating it (q_1, $\mathcal{L}_{\text{KL}}$=0.50, evaluated in Exercise 2).

THE RESULT is a latent space where geometric proximity reflects semantic similarity.

VAE Variants

β-VAE[6] scales up the KL term to pressure the encoder to use the latent space efficiently. The loss reweights the with a single hyperparameter:

[6]

$$\mathcal{L}_\beta = \text{Reconstruction} + \beta \cdot \mathcal{L}_{\text{KL}}$$

VQ-VAE[7] replaces continuous latents with discrete codes from a learned codebook. Each encoder output z_e is snapped to the nearest codebook entry, giving reconstructional structure:

[7]

$$z_q = \arg \min_{e_k \in \text{codebook}} \|z_e - e_k\|$$

CONDITIONAL VAE[8] feeds a conditioning variable c, such as a class label, into both encoder and decoder. The model now learns conditional distributions, which enables directed generation:

[8]

$$q_{\theta_E}(z|x, c), \quad p_{\theta_D}(x|z, c)$$

The Reparameterisation Trick

How do we now the distribution of the current embedding space? Our model now reconstructs data in two steps: draw z from the prior, then decode. To measure how well the model explains an observation x, we would need to sum over all possible embeddings z and check which ones decode close to x:

$$p_{\theta_D}(x) = \int p_{\theta_D}(x|z)\, p(z)\, dz$$

In words. The probability that the model generates x equals, summed over every possible latent z, the probability that the decoder produces x given that z, weighted by the prior probability of drawing that z in the first place. Each z is one possible explanation; the integral combines all of them.

Spoiler. We can't. This integral has no closed form and the latent space is too high-dimensional to sum over by brute force. Instead of summing over all of latent space, the encoder $q_{\theta_E}(z|x)$ output is used as an estimate, for each input x, where useful embeddings z for x live. We sample only from that region[9], turning an impossible integral into a practical expectation over a handful of samples.

[9]

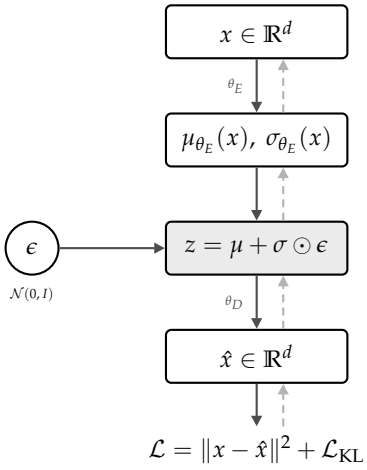


Figure 5: Reparameterised VAE or **Kingma's view**. Compare with Figure 4: the stochastic sample is replaced by $z = \mu + \sigma \odot \epsilon$ where ϵ carries the randomness and ,is drawn from a fixed distribution. Gradients (dashed) flow through μ and σ without passing through the random sample.

Designing the Bottleneck

THE LATENT DIMENSIONALITY k is the most consequential hyperparameter in a VAE. It controls the capacity of the information channel between encoder and decoder: how many bits of information about x can pass through z.

$$z \in \mathbb{R}^k$$

The bottleneck embedding. The dimensionality k controls how much information passes from encoder to decoder.

TOO FEW DIMENSIONS and the encoder cannot represent the factors of variation in the data. Reconstruction quality degrades because the embedding lacks the capacity to distinguish meaningfully different inputs. Two inputs that differ in important ways may map to nearly identical latent vectors, and the decoder produces a blurred average of what those embeddings could mean.

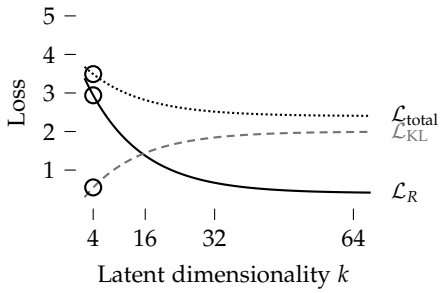


Figure 6: Bottleneck too small. At k=4, reconstruction loss is large ($\mathcal{L}_R \approx 2.9$) because the few active dimensions cannot encode all factors of variation. Each active dimension carries a lot of signal, but only a handful of them are paying KL ($\mathcal{L}_{\text{KL}} \approx 0.55$). The total ($\circ$ on the dotted curve) sits high because reconstruction error dominates.

TOO MANY DIMENSIONS and the model faces the opposite problem. With excess capacity, the encoder can encode each training point precisely without needing to discover shared structure. The KL term penalises unused dimensions toward the prior, but is blended out by the reconstruction success.

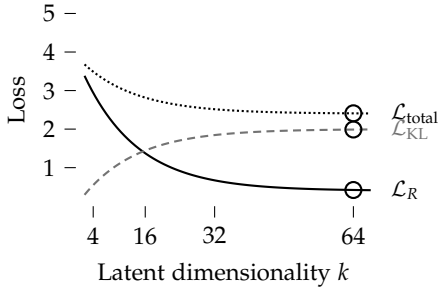


Figure 7: Bottleneck too large. At k=64, reconstruction loss has saturated near its floor ($\mathcal{L}_R \approx 0.42$) and the KL cost has saturated near its ceiling ($\mathcal{L}_{\text{KL}} \approx 1.99$): adding more dimensions changes neither term, because extra dimensions collapse to the prior and pay no KL. The total ($\circ$ on the dotted curve) sits at the plateau. The cost of k too large is therefore wasted parameters and slower training, not a higher loss.

THE INFORMATION BOTTLENECK PRINCIPLE[10] provides a theoretical lens. The optimal embedding retains all information in x that is relevant to reconstruction. The latent dimensionality k sets the ceiling

[10]

on how much can be stored; the KL weight determines how aggressively the model compresses toward that ceiling to discarding weak representations.

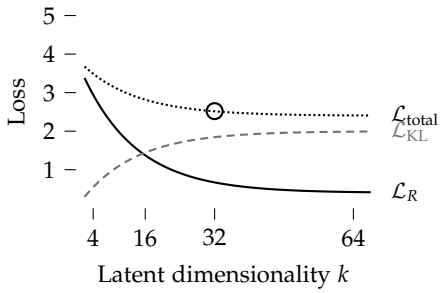


Figure 8: Bottleneck sizing trade-off. Reconstruction loss (solid) falls as k grows; the KL cost (dashed) rises until the data's information has been encoded, then plateaus as additional dimensions collapse to the prior. The total (dotted) is L-shaped: it drops sharply, then flattens. A good k sits near the elbow (○ at k=32), where the marginal gain per added dimension has fallen below what the extra parameters and training time cost.

ACTIVE UNITS. Inactive dimensions have collapsed to the prior, as there is no potential reconstruction gain contributed by their information. Monitoring the number of active units during training reveals how much of the latent capacity the model actually uses[11].

ACTIVE-UNIT CRITERION. A latent dimension j is called active if

$$\mathcal{L}_{\text{KL}}(q(z_j|x) \,\|\, p(z_j)) > \delta$$

for a meaningful fraction of the training data.

[11]

THIS READING ONLY HOLDS IN A PARTICULAR REGIME. The active-unit count is meaningful when k is rather large that some dimensions could afford to drop, and when the reconstruction loss is competitive with the KL pull. The diagnostic is really a measure of how many dimensions the reconstruction loss can keep alive against the KL pull. So cross-dimension activity comparisons are needed.

A PRACTICAL PROTOCOL for choosing k:

- Start with a generously large k. Adding capacity rarely hurts validation reconstruction; it just stops helping past the elbow.
- Train once and read off the effective capacity. The number of active dimensions is a direct estimate of the data's information ceiling.
- Use validation reconstruction, not training loss, as the model-selection signal to be sensitive to overfitting.

RULE OF THUMB. For input dimensionality d, a reasonable starting point is k on the order of $\sqrt{d}$ for tabular and pixel data. Text is a special case: vocabulary size makes d huge (30k+ tokens), but semantic information for sentence-level text VAEs, start at $k \approx 32-128$.

TWO EXCEPTIONS where the protocol does not apply. For visualisation, set k=2 or k=3 outright and accept the fidelity loss; the goal is a readable scatter, not optimal reconstruction. For very large models, where a single training run is expensive, pick a k from a comparable published architecture and skip the sweep entirely.

BEYOND DIMENSIONALITY, the structure of the prior itself shapes what the latent space can express. The standard choice $p(z) = \mathcal{N}(0, I)$ assumes the latent factors are independent and identically scaled. When the true factors of variation have different scales, correlations, or discrete structure, a mismatched prior forces the encoder to waste capacity compensating for the mismatch.

KL ANNEALING ramps the KL weight from zero over the first N epochs, letting the encoder learn useful representations before regularisation kicks in and prevents posterior collapse.

Exploring the Latent Space

LATENT INTERPOLATION reveals the structure that the VAE has learned. Given two data points x_1 and x_2, encode, interpolate in the latent space, then decode each step:

$$
\begin{aligned}
z_1 &= \mu_{\theta_E}(x_1) \\
z_2 &= \mu_{\theta_E}(x_2) \\
z_t &= (1-t)\, z_1 + t\, z_2 \\
\hat{x}_t &= \theta_D(z_t)
\end{aligned}
$$

Because the latent space is regularised, the decoded path transitions smoothly between the two inputs[12]. Smooth interpolation is the simplest test of whether the encoder has learned a genuinely continuous feature space rather than a scattered collection of embeddings.

[12]

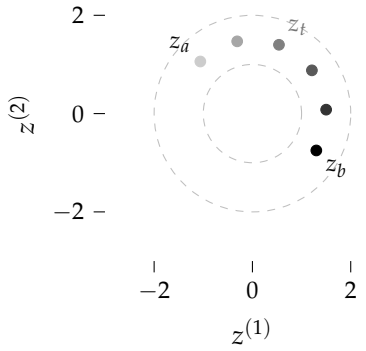


Figure 9: Latent interpolation. Two inputs encode to z_a and z_b. Linear interpolation (lerp, dashed) cuts through the low-density middle of the prior; in high dimensions this region is empty under $\mathcal{N}(0, I)$ and decoded midpoints look unnatural. Spherical linear interpolation (slerp, solid arc) follows the high-density shell at roughly $\|z\| \approx \sqrt{d}$, keeping every z_t on-distribution. Each waypoint is decoded to produce a smooth transition.

GENERATION is equally straightforward: sample from the prior and decode.

$$
\begin{aligned}
z &\sim \mathcal{N}(0, I) \\
\hat{x} &= \theta_D(z)
\end{aligned}
$$

The regularisation ensures that every region of the prior generates plausible data. The same property that makes the encoder a reliable feature extractor, a latent space with no dead zones, is what makes generation possible.

WELCOME TO GENERATIVE AI. A trained decoder paired with a tractable prior is, in essence, a generative model: draw a vector from a Gaussian, run it through the network, and a new sample falls out the other side. Everything that followed in the field, from diffusion models to large image and video generators, builds on the same trick of learning a mapping from a simple prior to a complex data distribution.

Examples & Exercises

The exercises start with a few numerical passes to fix the mechanics, then a set of concept and diagnostic prompts that target the key ideas of the chapter one at a time, and finally a hands-on notebook on MNIST. Worked solutions for the compute exercises are in the body; concept exercises carry the answer in a margin note. Try it yourself first, then check.

VAE loss by the numbers. For a 1D VAE with encoder output μ=1, σ=1 on input x=3 with reconstruction $\hat{x}$=2.7, compute the total loss $\mathcal{L} = \mathcal{L}_R + \mathcal{L}_{\text{KL}}$ using squared error and the Gaussian KL closed form $\frac{1}{2}(\mu^2 + \sigma^2 - \log\sigma^2 - 1)$.

Reconstruction loss.

$$\begin{aligned}\mathcal{L}_R &= (x - \hat{x})^2 \\ &= (3 - 2.7)^2 \\ &= 0.3^2 \\ &= 0.09\end{aligned}$$

KL term.

$$\begin{aligned}\mathcal{L}_{\text{KL}} &= \tfrac{1}{2}(\mu^2 + \sigma^2 - \log\sigma^2 - 1) \\ &= \tfrac{1}{2}(1 + 1 - \log 1 - 1) \\ &= \tfrac{1}{2}(1 + 1 - 0 - 1) \\ &= \tfrac{1}{2}(1) \\ &= 0.50\end{aligned}$$

Total loss.

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_R + \mathcal{L}_{\text{KL}} \\ &= 0.09 + 0.50 \\ &= 0.59\end{aligned}$$

SECOND CONFIGURATION: μ=0, σ=1, x=3, $\hat{x}$=2.0.

$$\begin{aligned}\mathcal{L}_R &= (3-2.0)^2 \\ &= 1.00 \\ \mathcal{L}_{\text{KL}} &= \tfrac{1}{2}(0+1-0-1) \\ &= 0 \\ \mathcal{L} &= 1.00+0 \\ &= 1.00\end{aligned}$$

The encoder matches the prior exactly so the KL drops to zero, but reconstruction more than triples. The first configuration achieves a lower total loss by spending some KL for a much better fit; that trade-off is what training resolves.

WHERE THE TRADE-OFF LIVES. Reconstruction is small in configuration one but the KL penalty is five times larger; in configuration two the KL is free but reconstruction dominates. Training finds the encoder that minimises the sum, not either term in isolation.

KL DIVERGENCE BY HAND. Compute $\mathcal{L}_{\text{KL}}\big(\mathcal{N}(\mu,\sigma^2)\,\|\,\mathcal{N}(0,1)\big) = \tfrac{1}{2}(\mu^2+\sigma^2-\log\sigma^2-1)$ for three settings, and split each result into a shift cost $\tfrac{1}{2}\mu^2$ and a spread cost $\tfrac{1}{2}(\sigma^2-\log\sigma^2-1)$.

SETTING (I): μ=0, σ=1 (posterior matches the prior).

$$\begin{aligned}\tfrac{1}{2}\mu^2 &= 0 \\ \tfrac{1}{2}(\sigma^2-\log\sigma^2-1) &= \tfrac{1}{2}(1-0-1) \\ &= 0 \\ \mathcal{L}_{\text{KL}} &= 0+0 \\ &= 0\end{aligned}$$

SETTING (II): μ=1, σ=1 (mean shifted, same spread).

$$\begin{aligned}\tfrac{1}{2}\mu^2 &= \tfrac{1}{2}\times 1 \\ &= 0.50 \\ \tfrac{1}{2}(\sigma^2-\log\sigma^2-1) &= \tfrac{1}{2}(1-0-1) \\ &= 0 \\ \mathcal{L}_{\text{KL}} &= 0.50+0 \\ &= 0.50\end{aligned}$$

SETTING (III): $\mu=0$, $\sigma=2$ (correct mean, too wide).

$$\begin{aligned}\tfrac{1}{2}\mu^2 &= 0\\ \tfrac{1}{2}(\sigma^2-\log\sigma^2-1) &= \tfrac{1}{2}(4-\log 4-1)\\ &= \tfrac{1}{2}(4-1.386-1)\\ &\approx 0.81\\ \mathcal{L}_{\mathrm{KL}} &\approx 0+0.81\\ &\approx 0.81\end{aligned}$$

WHY THE ASYMMETRY. The shift cost is quadratic in μ. The spread cost is asymmetric: it grows roughly linearly as σ exceeds 1, but the $-\log\sigma^2$ term blows up as $\sigma \to 0$, so narrow posteriors are the most expensive of all. Setting (iii) costs more than (ii) because spreading mass away from the prior is harder than translating it.

β-VAE BALANCE PASS. Reuse the first configuration from *VAE loss by the numbers* ($\mathcal{L}_R$=0.09, $\mathcal{L}_{\mathrm{KL}}$=0.50) and compute the β-VAE loss $\mathcal{L}_\beta = \mathcal{L}_R + \beta\cdot\mathcal{L}_{\mathrm{KL}}$ for $\beta\in\{0.5,\,1,\,4\}$.

β=0.5.

$$\begin{aligned}\mathcal{L}_{\beta=0.5} &= 0.09+0.5\times 0.50\\ &= 0.09+0.25\\ &= 0.34\end{aligned}$$

β=1 (STANDARD ELBO).

$$\begin{aligned}\mathcal{L}_{\beta=1} &= 0.09+1\times 0.50\\ &= 0.09+0.50\\ &= 0.59\end{aligned}$$

β=4.

$$\begin{aligned}\mathcal{L}_{\beta=4} &= 0.09+4\times 0.50\\ &= 0.09+2.00\\ &= 2.09\end{aligned}$$

The KL contribution to the total grows from 73% (β=0.5) to 96% (β=4). At β=4 the encoder has every incentive to shrink μ toward zero and σ toward one, at the cost of reconstruction. At β=0.5 the prior pull is relaxed and the encoder can spread out for sharper reconstructions. β=1 is the standard ELBO balance.

WHY β MATTERS. Training does not just evaluate this loss once; it follows the gradient. Whichever term dominates the loss dominates the gradient, and that is what shapes the final encoder.

REPARAMETERISATION. For μ=0.5, σ=1.2, and a sampled $\epsilon=-0.3$, compute the reparameterised sample $z=\mu+\sigma\epsilon$ and verify the gradients with respect to μ and σ.

FORWARD PASS.

$$\begin{aligned} z &= \mu + \sigma \cdot \epsilon \\ &= 0.5 + 1.2 \times (-0.3) \\ &= 0.5 - 0.36 \\ &= 0.14 \end{aligned}$$

GRADIENTS.

$$\begin{aligned} \frac{\partial z}{\partial \mu} &= \frac{\partial}{\partial \mu}(\mu + \sigma\epsilon) \\ &= 1 \\ \frac{\partial z}{\partial \sigma} &= \frac{\partial}{\partial \sigma}(\mu + \sigma\epsilon) \\ &= \epsilon \\ &= -0.3 \end{aligned}$$

Both gradients are finite scalars, so μ and σ get a clean signal from backpropagation; the randomness lives in ϵ, off the gradient path.

KL COST, BY INSPECTION. Without computing, rank the following encoder distributions by KL cost against the prior $\mathcal{N}(0, 1)$, lowest first.

- (i) μ=0, σ=1.
- (ii) μ=0, σ=0.3.
- (iii) μ=1.5, σ=1.
- (iv) μ=0, σ=3.

ANSWER. (i) < (iii) < (iv) < (ii). (i) matches the prior exactly so the KL is zero. (iii) only shifts the mean; the cost grows quadratically with μ. (iv) is too wide but the spread cost is roughly linear in σ^2. (ii) is too narrow, and the $-\log\sigma^2$ term blows up as $\sigma \to 0$, making narrow posteriors the most expensive.

AE VERSUS VAE: PREDICT THE EMBEDDING. You train a vanilla autoencoder and a VAE with k=2 on MNIST, both to convergence, and plot the validation embeddings.

- Which space concentrates encodings near the origin, and which spreads them arbitrarily across the plane?
- Does proximity in the latent space imply the inputs are visually similar? Is the answer the same for both models?

- If you pick a random point from a sparsely populated region and decode it, in which space is the result a plausible digit?
- Why k=2 and not k=10, given that MNIST has ten digit classes?

ANSWER. The AE places encodings wherever reconstruction loss is lowest; the layout is arbitrary, with no global structure and large unused regions. The VAE pulls every encoding toward the prior, so the cloud concentrates near the origin and fills the unit disc. Proximity reflects visual similarity in both, because the decoder is smooth, but only the VAE gives geometric position a global meaning: any z sampled from the prior decodes to something plausible, while a random point in an AE's empty region decodes to noise. k=2 is for visualisation; you plot $z^{(1)}$ against $z^{(2)}$ directly. At k=10 you would need t-SNE or UMAP to see the layout, and the differences you observe would be artefacts of the projection rather than of the AE-vs-VAE distinction.

PER-DIMENSION KL. For a k=4 VAE on a single input, the encoder outputs the parameters below (independent dimensions, prior $\mathcal{N}(0,1)$ per dim). Compute $\mathcal{L}_{\mathrm{KL}}^{(j)} = \frac{1}{2}(\mu_j^2 + \sigma_j^2 - \log\sigma_j^2 - 1)$ for each dimension and the total $\mathcal{L}_{\mathrm{KL}} = \sum_j \mathcal{L}_{\mathrm{KL}}^{(j)}$, then label each dimension active or collapsed.

DIM 1: μ_1=1.5, σ_1=0.5.

$$\begin{aligned}\mathcal{L}_{\mathrm{KL}}^{(1)} &= \tfrac{1}{2}(1.5^2 + 0.5^2 - \log 0.5^2 - 1)\\ &= \tfrac{1}{2}(2.25 + 0.25 + 1.386 - 1)\\ &= \tfrac{1}{2}(2.886)\\ &\approx 1.44\end{aligned}$$

DIM 2: μ_2=0, σ_2=1.

$$\begin{aligned}\mathcal{L}_{\mathrm{KL}}^{(2)} &= \tfrac{1}{2}(0 + 1 - 0 - 1)\\ &= 0\end{aligned}$$

DIM 3: μ_3=0.2, σ_3=0.9.

$$\begin{aligned}\mathcal{L}_{\mathrm{KL}}^{(3)} &= \tfrac{1}{2}(0.2^2 + 0.9^2 - \log 0.9^2 - 1)\\ &= \tfrac{1}{2}(0.04 + 0.81 + 0.211 - 1)\\ &= \tfrac{1}{2}(0.061)\\ &\approx 0.03\end{aligned}$$

DIM 4: $\mu_4 = -0.8$, σ_4=0.7.

$$\begin{aligned}\mathcal{L}_{\mathrm{KL}}^{(4)} &= \tfrac{1}{2}((-0.8)^2 + 0.7^2 - \log 0.7^2 - 1)\\ &= \tfrac{1}{2}(0.64 + 0.49 + 0.713 - 1)\\ &= \tfrac{1}{2}(0.843)\\ &\approx 0.42\end{aligned}$$

TOTAL AND VERDICT.

$$\begin{aligned}\mathcal{L}_{\mathrm{KL}} &= 1.44 + 0 + 0.03 + 0.42\\ &\approx 1.89\end{aligned}$$

Dimensions 1 and 4 are active: large per-dim KL, σ well below 1, mean clearly away from zero. Dimension 2 has collapsed to the prior. Dimension 3 is borderline: nearly at the prior, contributing almost no information.

WHAT THE PER-DIM KL TELLS YOU. Posterior collapse is a per-dimension phenomenon: a single VAE can have some dimensions active and others collapsed in the same training run. Summing the KL hides that detail; reading it per dimension exposes which axes the encoder uses and which it has given up on.

COUNTING ACTIVE UNITS. A VAE with k=8 reports the following per-dimension posterior statistics, averaged over the validation set: the squared mean $\overline{\mu_j^2}$, the variance σ_j^2, and the resulting per-dimension KL $\mathcal{L}_{\mathrm{KL}}^{(j)} = \frac{1}{2}(\overline{\mu_j^2} + \sigma_j^2 - \log \sigma_j^2 - 1)$.

Dim j	1	2	3	4	5	6	7	8
$\overline{\mu_j^2}$	1.50	0.01	0.80	0.00	1.20	0.00	0.60	0.70
σ_j^2	0.12	0.97	0.31	0.99	0.18	1.00	0.41	0.95
$\mathcal{L}_{\mathrm{KL}}^{(j)}$	1.37	0.01	0.64	0.00	1.05	0.00	0.45	0.35

Table 1: Per-dimension posterior statistics for a k=8 VAE on the validation set. Inactive dimensions have $\overline{\mu_j^2} \approx 0$ and $\sigma_j^2 \approx 1$, so their KL contribution falls to zero and they have collapsed back to the prior. Active dimensions either spread the means (large $\overline{\mu_j^2}$), tighten the variance (small σ_j^2), or both, and pay a positive KL for the information they carry.

How many dimensions are doing real work? Which ones can you remove without losing reconstruction quality? If you re-trained at k=4, would you expect the same number of active dimensions?

ANSWER. Five active dimensions: 1, 3, 5, 7, 8, identifiable by $\mathcal{L}_{\mathrm{KL}}^{(j)}$ well above zero. Dimensions 2, 4, 6 have $\mathcal{L}_{\mathrm{KL}}^{(j)} \approx 0$ and have collapsed to the prior; dropping them costs nothing. Note that dim 8 looks borderline by variance alone ($\sigma_8^2 = 0.95$), but its mean spread $\overline{\mu_8^2} = 0.70$ keeps it carrying information, which is why the KL is the cleaner diagnostic. Re-training at k=4 should still yield about four active units, because the active count is set by the data's information content, not by k once k is past the elbow.

DIAGNOSE THE TRAINING RUN. Three runs of the same VAE on the same data give different end-of-training behaviour. For each, name what is happening and what you would change.

Run	$\mathcal{L}_R$	$\mathcal{L}_{\mathbf{KL}}$	**Reconstructions**
A	high	≈ 0	blurry mean digit
B	low	high	sharp but unrelated to input
C	low	moderate	sharp and semantically sound

Table 2: Three end-of-training profiles for the same VAE on the same data. One is healthy training, one is posterior collapse, and one is a memorising encoder that bypasses the prior.

ANSWER. **A** is posterior collapse: KL near zero means the encoder ignores x, so the decoder outputs the data mean. Lower β or anneal it from zero. **B** has the encoder bypassing the prior to memorise inputs (high KL, low reconstruction, but sharp outputs that do not generalise to sampled z). Raise β or check for an over-parameterised encoder. **C** is what you want: both terms contribute and reconstructions track the inputs.

KL ANNEALING BY THE NUMBERS. A linear-ramp schedule sets

$$\beta(t) = \min\left(\tfrac{t}{10}, 1\right)$$

so β rises from 0 at epoch 0 to 1 at epoch 10 and stays at 1 afterwards. Assume that on a fixed validation batch the encoder reports $\mathcal{L}_R$=0.4 and $\mathcal{L}_{\mathrm{KL}}$=1.2 throughout training (the values themselves do not change, only the weight). Compute the total loss $\mathcal{L}_\beta(t) = \mathcal{L}_R + \beta(t) \cdot \mathcal{L}_{\mathrm{KL}}$ at epochs $t \in \{1, 5, 10, 20\}$.

EPOCH 1: β=0.1.

$$\begin{aligned}\mathcal{L}_\beta(1) &= 0.4 + 0.1 \times 1.2 \\ &= 0.4 + 0.12 \\ &= 0.52\end{aligned}$$

EPOCH 5: β=0.5.

$$\begin{aligned}\mathcal{L}_\beta(5) &= 0.4 + 0.5 \times 1.2 \\ &= 0.4 + 0.60 \\ &= 1.00\end{aligned}$$

EPOCH 10: β=1.

$$\begin{aligned}\mathcal{L}_\beta(10) &= 0.4 + 1 \times 1.2 \\ &= 1.60\end{aligned}$$

EPOCH 20: β=1 (CLAMPED).

$$\mathcal{L}_\beta(20) = 1.60$$

At which epoch does the KL contribution first match the reconstruction term, and what would $\mathcal{L}(t)$ look like if you set β=1 from the start?

WHEN THE TERMS MATCH. $\beta(t) \cdot \mathcal{L}_{\mathrm{KL}} = \mathcal{L}_R$ when $\beta(t) = 0.4/1.2 \approx 0.33$, i.e. around epoch 3 on this schedule. Without the ramp, the KL would dominate from epoch 1, giving the encoder every incentive to ignore x before it has learned a useful representation; the warm-up window buys the encoder time to first encode information, then take the regularisation pressure.

READ THE OBSERVATIONS FROM THE EMBEDDING SPACE EXPLORATION. You linearly interpolate between two encoded digits in latent space and decode each step. For each described path, name what the latent space tells you.

- Path 1: a smooth morph from a 3 into an 8, passing through plausible intermediate shapes.
- Path 2: the 3 stays fixed for half the path, then suddenly jumps to the 8.
- Path 3: the path passes through several unrelated digit shapes (a 2, then a 5) before reaching the 8.

ANSWER. Path 1 indicates a well-organised latent space: nearby points decode to similar digits and the regulariser has filled the region between the two encodings. Path 2 means the encoder placed the two digits far apart with empty space in between; the prior has not pulled them close enough. Path 3 means the line crosses other class regions; that is geometrically honest but suggests the chosen direction is not a meaningful axis.

AN MNIST SUBSET TO EXPLORE. You are given the same thousand handwritten digits as in the previous chapter. Implement three pieces from scratch in numpy: a linear autoencoder, the closed-form KL divergence to a standard normal prior, and the reparameterisation trick $z = \mu + \sigma \odot \varepsilon$. Sweep k on the linear AE and inspect the reconstruction curve, then compare precomputed AE and VAE embeddings at k=2 and at $k \in \{2, 8, 32\}$. Optionally lift a k=16 VAE: bar-chart its per-dimension KL using your own implementation, and decode an interpolation between an encoded 0 and an encoded 1.

CODE is provided as a Python notebook. Use it as a starting point, break things, and observe what changes.

1 000 MNIST digits
784 pixels each
ten classes

implement: linear AE, KL divergence, reparameterisation trick

WHAT TO OBSERVE. The linear AE elbow matches the PCA reconstruction curve, confirming that a tied-weight linear AE is PCA in disguise. The AE and VAE scatters at k=2 differ in geometry: AE encodings spread arbitrarily with large empty regions, VAE encodings concentrate near the origin and fill the unit disc. Active units saturate well below the chosen k. The interpolation passes through plausible intermediate digits, showing that the latent space is genuinely continuous.

Self-Reflection and Recap

SELF-REFLECTION questions to guide your thinking:

- What does the autoencoder bottleneck force the encoder to discover, and how does it relate to the PCA subspace?
- Why does a deterministic autoencoder fail as a generative model?
- In the Gaussian KL, why is a narrow posterior more expensive than a shifted or widened one?
- When would you reach for β-VAE, VQ-VAE, or a conditional VAE over the plain VAE?
- Why does the reparameterisation trick work, and what does it exploit about the Gaussian?
- What is posterior collapse, and which design lever brings it on most often?
- How does the active-unit count tell you whether k is set correctly?
- Why does sampling $z \sim \mathcal{N}(0, I)$ and decoding produce plausible new data in a VAE but not in a plain autoencoder?

RECAP of Key Concepts:

- Autoencoders learn a nonlinear θ_E, θ_D through a bottleneck; linear AE recovers PCA, nonlinear goes beyond it.
- VAEs replace the point embedding with $q(z|x)$ regularised toward a prior, giving a continuous, generative latent space.
- The Gaussian KL trades a quadratic shift cost against an asymmetric spread cost; narrow posteriors cost the most.
- β-VAE buys disentanglement, VQ-VAE buys discrete codes, conditional VAE buys controlled generation.
- The reparameterisation trick $z = \mu + \sigma \odot \varepsilon$ keeps the sample differentiable in μ and σ.
- Active-unit count diagnoses bottleneck use; KL warm-up is the standard fix for posterior collapse.
- Sampling $z \sim p(z)$ and decoding generates new data; smooth interpolation tests latent continuity.

THE ROOT OF MODEL BLINDNESS. Any model can tell you what it has learned, but not how much to trust what it says about inputs it has never seen.

UNSUPERVISED DEEP LEARNING is at root the project of modelling what counts as familiar, and the VAE does this directly: low reconstruction error and high prior likelihood under $p(z)$ mean an input resembles the training data. Yet every model trained on finite data, generative or not, has a blind spot for inputs outside its experience: a VAE's error rises on unfamiliar inputs and a classifier's softmax stays peaked on adversarial ones, but neither flags the gap in any principled way.

UNCERTAINTY ESTIMATION turns familiarity into a principled tool, and it is itself an unsupervised problem: knowing where a model's knowledge ends depends on the learned distribution of inputs, not on a held-out label. The next chapter develops model-agnostic methods for separating irreducible noise from reducible ignorance, and uses them for anomaly detection, out-of-distribution flagging, and graceful degradation in deployment.

TEASER. How does a model trained without labels know when a new input falls outside the structure it has learned?

FEEDBACK.