Yale University

# Autoencoders and Variational Autoencoders

S&DS 265 · Introductory Machine Learning · Lecture 17

Zhuoran Yang

Fall 2026

AI usage: figures were generated, and these slides polished, with the help of AI tools.

# Outline

# Learning Objectives

In this lecture you will learn:

1. Why a continuous latent variable can put the EM E-step out of reach;
2. How PCA and probabilistic PCA arise inside linear Gaussian families;
3. How restricting the variational family turns inference into optimization;
4. Why an encoder amortizes that optimization and reparameterization makes it differentiable;
5. What a fitted VAE actually produces, and where its training and its samples break down.

# Outline

# Motivating Example: Writing a New Digit

Last lecture's hidden variable named one of three kinds of car: a label with $K$ values. To draw a new handwritten digit we want a richer hidden variable — a continuous code that can be moved smoothly from one shape into another.

Can we fit such a model the way we fitted the mixture?

Example data: scikit-learn handwritten digits, 1,797 images.

# The Handwritten Digits Data

| label | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $\cdots$ | shape |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 5 | 13 | 9 | $\cdots$ | $8 \times 8$ |
| 1 | 0 | 0 | 0 | 12 | 13 | $\cdots$ | $8 \times 8$ |
| 2 | 0 | 0 | 0 | 4 | 15 | $\cdots$ | $8 \times 8$ |
| 3 | 0 | 0 | 7 | 15 | 13 | $\cdots$ | $8 \times 8$ |

The dataset contains 1,797 rows and 64 pixel-intensity columns. Only the pixels are used; the class labels are withheld throughout.

Source: Scikit-learn handwritten digits dataset.

# Reading One Image

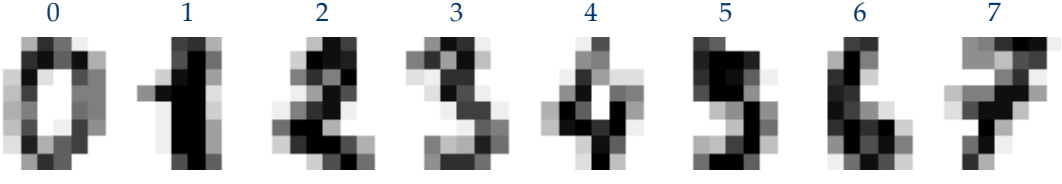
The first observation has label 0 and pixel intensities

$$\begin{bmatrix} 0 & 0 & 5 & 13 & 9 & 1 & 0 & 0 \\ 0 & 0 & 13 & 15 & 10 & 15 & 5 & 0 \\ 0 & 3 & 15 & 2 & 0 & 11 & 8 & 0 \\ 0 & 4 & 12 & 0 & 0 & 8 & 8 & 0 \\ 0 & 5 & 8 & 0 & 0 & 9 & 8 & 0 \\ 0 & 4 & 11 & 0 & 1 & 12 & 7 & 0 \\ 0 & 2 & 14 & 5 & 10 & 12 & 0 & 0 \\ 0 & 0 & 6 & 13 & 10 & 0 & 0 & 0 \end{bmatrix}.$$

Every model in this lecture sees a vector like this one and nothing else.

Source: First row of SklearnDigits.csv; intensity range 0 to 16.

# Handwritten Digits as 64-Dimensional Observations




Scikit-learn digits data; each observation is an 8 by 8 grayscale array.

# What EM Required

The two steps are not equally fragile:

$$\text{E-step:} \quad q^{(t+1)}(z) = p_{\theta^{(t)}}(z \mid x), \qquad \text{M-step:} \quad \theta^{(t+1)} = \arg\max_{\theta} \mathbb{E}_{q^{(t+1)}}[\log p_\theta(x, z)].$$

The M-step is a weighted maximum-likelihood fit. It survives almost any change of model: give it weights and it fits whatever the decoder is.

The E-step needs the posterior as a formula — and that is what we are about to lose.

# Why the Mixture Posterior Was Available

For $K$ kinds, Bayes' rule was a ratio we could write down and evaluate:

$$p_\theta(z = k \mid x) = \frac{\pi_k \cdot \mathcal{N}(x; \mu_k, \Sigma_k)}{\sum_{\ell=1}^{K} \pi_\ell \cdot \mathcal{N}(x; \mu_\ell, \Sigma_\ell)}.$$

The denominator is a sum of $K$ terms; with $K = 3$ we evaluated it by hand.

For a continuous code $z \in \mathbb{R}^r$ the same rule reads

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

and that sum has become an integral over $\mathbb{R}^r$.

# Continuity by Itself Is Not the Problem

Take $p(z) = \mathcal{N}(0, I_r)$ and a linear decoder,

$$x \mid z \sim \mathcal{N}(Wz + \mu,\ \sigma^2 I_d).$$

Everything in sight is Gaussian, the integral can be done, and the posterior is Gaussian with a closed form:

$$p(z \mid x) = \mathcal{N}(M^{-1}W^\top(x - \mu),\ \sigma^2 M^{-1}), \qquad M = W^\top W + \sigma^2 I_r.$$

So a continuous latent is perfectly workable — as long as the model stays linear and Gaussian.

**That model is probabilistic PCA, which we derive next.**

# What Actually Breaks It

Replace the linear map by a neural network $f_\theta$:

$$x \mid z \sim \mathcal{N}(f_\theta(z),\, \sigma^2 I_d).$$

Now

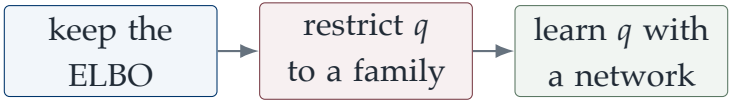
$$p_\theta(x) = \int p(z)\, \mathcal{N}(x; f_\theta(z), \sigma^2 I)\, dz$$

has no closed form, because $f_\theta$ sits inside the exponent behind layers of nonlinearity. Numerical integration does not rescue it either: ten grid points per axis in $r = 16$ dimensions is $10^{16}$ decoder evaluations, for one observation.

No formula for $p_\theta(x)$ means no formula for $p_\theta(z \mid x)$, so the E-step cannot be carried out.

# The Plan: Give Up Exactness, Keep the Bound

EM insisted on the exact posterior, and got a bound that touched the likelihood. If that posterior is out of reach, keep the bound and settle for the best $q$ we can actually compute.




The middle box is **variational inference**; the right-hand box turns it into a **variational autoencoder**.

# Outline

# Linear-Gaussian Latent Model

Probabilistic PCA specifies

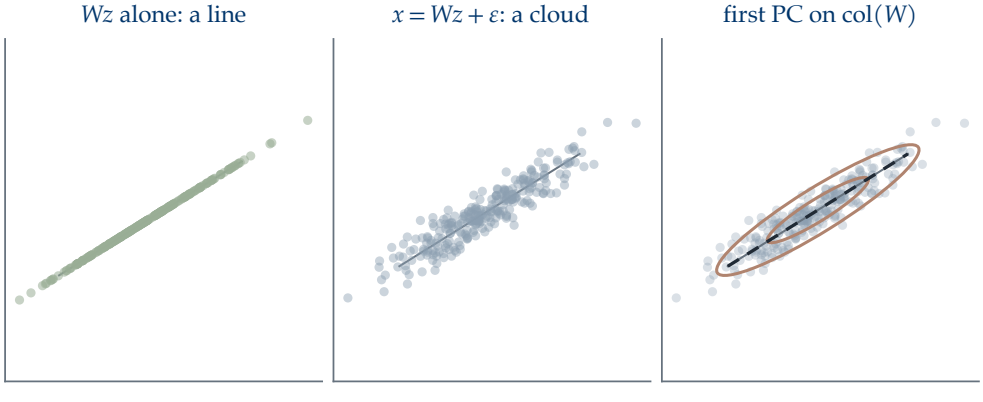
$$z \sim \mathcal{N}(0, I_r), \qquad x \mid z \sim \mathcal{N}(Wz + \mu, \sigma^2 I_d).$$

Equivalently,

$$x = \mu + Wz + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2 I_d), \qquad z \perp \epsilon.$$

Read the second form as **signal plus noise**. The signal $Wz$ can only ever land in the column space of $W$ — an $r$-dimensional subspace of $\mathbb{R}^d$ — and the noise $\epsilon$ pushes the observation off it, by the same amount in every direction.

# Building an Observation in Two Steps




With $r = 1$ the signal $Wz$ lies exactly on a line; isotropic noise turns it into a cloud around that line. Fitting PCA to the cloud returns a first principal direction within 0.5° of col$(W)$ — the next slides say why that is no accident.

Course simulation, seed 26520; 260 draws, $\|W\| = 1.7$, noise standard deviation .32.

# Marginal Distribution of the Observation

derive

From $x = \mu + Wz + \epsilon$,

$$\mathbb{E}[x] = \mu + W\mathbb{E}[z] + \mathbb{E}[\epsilon] = \mu,$$

and independence gives

$$\begin{aligned}\mathrm{Cov}(x) &= W\,\mathrm{Cov}(z)W^\top + \mathrm{Cov}(\epsilon) \\ &= WW^\top + \sigma^2 I_d.\end{aligned}$$

Therefore the distribution of $x$ is

$$x \sim \mathcal{N}(\mu, WW^\top + \sigma^2 I_d).$$

Every question about the observations is now a question about this one covariance matrix.

# Why That Covariance Is PCA

Write the loadings with orthogonal columns, $W = [a_1 u_1, \ldots, a_r u_r]$ with $u_1, \ldots, u_r$ orthonormal — always possible, since $W$ and $WR$ give the same $WW^\top$. Then

$$(WW^\top + \sigma^2 I)\, u_j = (a_j^2 + \sigma^2)\, u_j, \qquad (WW^\top + \sigma^2 I)\, v = \sigma^2 v \quad \text{for } v \perp \text{col}(W).$$

# Why That Covariance Is PCA

Write the loadings with orthogonal columns, $W = [a_1 u_1, \ldots, a_r u_r]$ with $u_1, \ldots, u_r$ orthonormal — always possible, since $W$ and $WR$ give the same $WW^\top$. Then

$$(WW^\top + \sigma^2 I)\, u_j = (a_j^2 + \sigma^2)\, u_j, \qquad (WW^\top + \sigma^2 I)\, v = \sigma^2 v \quad \text{for } v \perp \text{col}(W).$$

So the covariance has eigenvalue $a_j^2 + \sigma^2$ along each loading direction and $\sigma^2$ in every direction orthogonal to them.

# Why That Covariance Is PCA

Write the loadings with orthogonal columns, $W = [a_1 u_1, \dots, a_r u_r]$ with $u_1, \dots, u_r$ orthonormal — always possible, since $W$ and $WR$ give the same $WW^\top$. Then

$$(WW^\top + \sigma^2 I)\, u_j = (a_j^2 + \sigma^2)\, u_j, \qquad (WW^\top + \sigma^2 I)\, v = \sigma^2 v \quad \text{for } v \perp \mathrm{col}(W).$$

So the covariance has eigenvalue $a_j^2 + \sigma^2$ along each loading direction and $\sigma^2$ in every direction orthogonal to them.

The $r$ largest eigenvectors of $\mathrm{Cov}(x)$ therefore span exactly $\mathrm{col}(W)$: **PCA on the covariance returns the directions the latent code loads on.**

# What Exactly Are We Estimating?

The parameter is $\theta = (W, \mu, \sigma^2)$ — and nothing else:

| | what it controls | how many numbers |
|---|---|---|
| $W$ | the subspace, and the spread along it | $d \times r$ |
| $\mu$ | where the cloud sits | $d$ |
| $\sigma^2$ | how far observations drift off the subspace | 1 |

The codes $z_1, \ldots, z_n$ are not parameters. There is one per observation, and EM integrates them out rather than fitting them. $\mu$ is the easy one: its maximum-likelihood value is the sample mean $\bar{x}$, whatever $W$ and $\sigma^2$ do, so set $\mu = \bar{x}$ once and leave it. That leaves $W$ and $\sigma^2$ to iterate on.

# The E-Step

Hold the current $W$ and $\sigma^2$ fixed. Because prior and likelihood are both Gaussian and the map is linear, the posterior of the code is again Gaussian, and its mean and covariance are formulas in the current parameters:

$$z_i \mid x_i \;\sim\; \mathcal{N}(m_i,\, S), \qquad m_i,\, S \;\text{ computed from } (W, \sigma^2, x_i).$$

Last lecture this step produced $K$ numbers per observation. Here it produces a distribution, and what the M-step consumes is its first two moments, $\mathbb{E}[z_i \mid x_i]$ and $\mathbb{E}[z_i z_i^\top \mid x_i]$.

Source: Both formulas, and the fact that $S$ is the same for every $i$, in Appendix A.1.

# The M-Step

Hold that distribution fixed. Maximizing $\sum_i \mathbb{E}[\log p_\theta(x_i, z_i)]$ over $W$ is a **least-squares fit of $x$ on the codes**, with the squared error averaged over the posterior:

$$W^{\text{new}} = \arg\min_W \sum_{i=1}^{n} \mathbb{E}_{z_i \sim p_{\theta^{(t)}}(z|x_i)}[\|x_i - \mu - W z_i\|_2^2],$$

and $\sigma^2$ becomes the average residual left behind.

The observation $x_i$ is fixed; the only random quantity is its code $z_i$, drawn from the posterior the E-step just computed at $\theta^{(t)}$.

# The M-Step

Hold that distribution fixed. Maximizing $\sum_i \mathbb{E}[\log p_\theta(x_i, z_i)]$ over $W$ is a **least-squares fit of $x$ on the codes**, with the squared error averaged over the posterior:

$$W^{\text{new}} = \arg\min_W \sum_{i=1}^{n} \mathbb{E}_{z_i \sim p_{\theta^{(t)}}(z|x_i)}[\|x_i - \mu - W z_i\|_2^2],$$

and $\sigma^2$ becomes the average residual left behind.

The observation $x_i$ is fixed; the only random quantity is its code $z_i$, drawn from the posterior the E-step just computed at $\theta^{(t)}$. So the normal equations use $\mathbb{E}[z_i z_i^\top]$ rather than $m_i m_i^\top$, and the fit is charged for the uncertainty in the codes.

Source: Closed-form updates in Appendix A.2.

# Where EM Lands

Run the two steps to convergence and $\mathrm{col}(W)$ is the span of the top $r$ eigenvectors of the sample covariance — the PCA subspace, with $\sigma^2$ absorbing the variance left outside it.

Only that subspace is identified: $W$ and $WR$ describe the same model for any orthogonal $R$, so the individual columns are not pinned down by the data.

**None of this is new machinery. It is Lecture 16's EM, run on a continuous latent variable that happens to keep its posterior in closed form.**

Source: Closed-form maximum likelihood and the EM updates: Tipping and Bishop (1999).

# Outline

# Restrict $q$ to a Family We Can Compute With

The identity from last lecture holds for every $q$, whether or not it is the posterior:

$$\log p_\theta(x) = \mathcal{L}(q, \theta) + D_{\mathrm{KL}}(q(z) \,\|\, p_\theta(z \mid x)).$$

EM maximized over all $q$, the winner was the posterior, and the gap closed.

Instead choose a **variational family** $\mathcal{Q}$ of distributions we can write down — say diagonal Gaussians $q = \mathcal{N}(m, \mathrm{diag}(s^2))$ — and search only inside it:

$$q^\star = \arg\max_{q \in \mathcal{Q}} \; \mathcal{L}(q, \theta) = \arg\min_{q \in \mathcal{Q}} \; D_{\mathrm{KL}}(q \,\|\, p_\theta(z \mid x)).$$

The move is a familiar one. In nonlinear regression we could not search over all functions, so we fixed a **model class** — polynomials, splines, a network — and searched inside it. Here the unknown object is a distribution rather than a function, and $\mathcal{Q}$ plays the same role.

# Two Descriptions of the Same Search

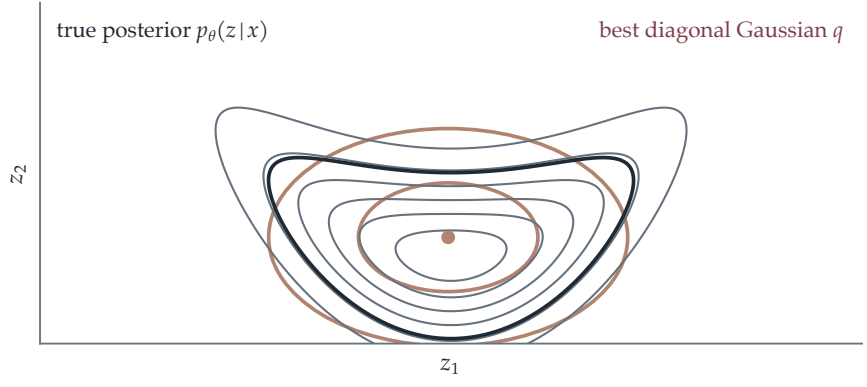
Those two problems are identical because $\log p_\theta(x)$ does not depend on $q$: pushing the bound up and pulling $q$ towards the posterior are the same motion, viewed from the two ends of the identity.

**What this buys**

Inference has become **optimization**. We never write the posterior down; we search a family for the member closest to it, using an objective we can evaluate.

**And what it costs: unless the posterior happens to lie in $\mathcal{Q}$, the gap never closes. The bound stays strictly below $\log p_\theta(x)$.**

# The Gap Is the Price of a Convenient Family




A curved posterior, and the diagonal Gaussian closest to it in $D_{\mathrm{KL}}(q \,\|\, p)$. No member of the family can bend, so the best one spreads across the middle: mass where the posterior has little, and nothing along the two arms. That residual KL is the height lost from the bound.

Original course calculation, seed 26520; reverse-KL fit by Monte Carlo with 4,000 fixed draws.

# Approximate Variational Inference

The bound splits into two computable pieces (Appendix A.4–A.5):

$$\mathcal{L}(q, \theta) = \underbrace{\mathbb{E}_q[\log p_\theta(x \mid z)]}_{\text{reconstruction}} - \underbrace{D_{\mathrm{KL}}(q(z) \,\|\, p(z))}_{\text{stay near the prior}}.$$

Index the family by a parameter, $q_\phi(z \mid x)$, and replace the population problem by its sample version over the observed data:

$$\max_{\theta,\phi} \frac{1}{n} \sum_{i=1}^{n} \Big\{ \mathbb{E}_{q_\phi(z|x_i)}[\log p_\theta(x_i \mid z)] - D_{\mathrm{KL}}(q_\phi(z \mid x_i) \,\|\, p(z)) \Big\}.$$

**This is approximate variational inference.** Neither term needs the true posterior: the first is an average over draws from $q_\phi$, the second a formula when both distributions are Gaussian.

# A Linear $q_\phi$ Gives PPCA

Take the family of linear encoders: a matrix $A$ maps the observation to a mean, and the spread does not depend on $x$,

$$q_\phi(z \mid x) = \mathcal{N}(A(x - \mu),\ \Sigma_q), \qquad \phi = (A, \Sigma_q).$$

For the linear-Gaussian model above the **exact** posterior has precisely this shape, with $A = M^{-1}W^\top$ and $\Sigma_q = \sigma^2 M^{-1}$. The family contains the posterior, so the gap closes, and optimizing $\phi$ reproduces the E-step.

# A Linear $q_\phi$ Gives PPCA

Take the family of linear encoders: a matrix $A$ maps the observation to a mean, and the spread does not depend on $x$,

$$q_\phi(z \mid x) = \mathcal{N}(A(x - \mu),\ \Sigma_q), \qquad \phi = (A, \Sigma_q).$$

For the linear-Gaussian model above the **exact** posterior has precisely this shape, with $A = M^{-1}W^\top$ and $\Sigma_q = \sigma^2 M^{-1}$. The family contains the posterior, so the gap closes, and optimizing $\phi$ reproduces the E-step.

**PPCA is approximate variational inference in the case where the approximation happens to be exact.**

# Outline

# A Neural $q_\phi$ Gives the Variational Autoencoder

Let the two outputs come from a network instead of a matrix:

$$q_\phi(z \mid x) = \mathcal{N}(\mu_\phi(x), \mathrm{diag}(\sigma^2_\phi(x))), \qquad \mu_\phi : \mathbb{R}^d \to \mathbb{R}^r, \quad \sigma_\phi : \mathbb{R}^d \to \mathbb{R}^r_{>0},$$

with images in $\mathbb{R}^d$ and codes in $\mathbb{R}^r$.

Nothing in the objective changes. What changes is the family: it is now as expressive as the network, and the true posterior generally lies outside it, so the gap no longer closes.

This is the usual recipe. Fix a convenient distribution family — Gaussian here, Bernoulli or categorical elsewhere — and let a network output its parameters as a function of the input. The family stays simple enough to sample from and to take a KL of; the network supplies the flexibility.

# Size of the Two Networks

**Encoder, weights $\phi$**

in: one image, $d$ numbers
out: $2r$ numbers — a mean and a spread for each code coordinate

**Decoder, weights $\theta$**

in: one code, $r$ numbers
out: $d$ numbers — one distribution parameter per pixel

For the digits with a two-dimensional code, the encoder maps $64 \to 4$ numbers and the decoder maps $2 \to 64$.

Both are functions of their input, so one forward pass serves any observation, including one the model has never seen.

# The Training Loss, Term by Term

Minimize the negative bound over the data:

$$\mathcal{J}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} \Big\{ \underbrace{- \mathbb{E}_{q_\phi(z|x_i)}[\log p_\theta(x_i \mid z)]}_{\text{reconstruction: } \theta \text{ and } \phi} + \underbrace{D_{\mathrm{KL}}(q_\phi(z \mid x_i) \,\|\, p(z))}_{\phi \text{ only, closed form}} \Big\}$$

**Reconstruction: sample it**

an expectation over $z \sim q_\phi(\cdot \mid x_i)$, estimated with one draw per image

**KL: no sampling**

for Gaussian $q_\phi$ and $p(z) = \mathcal{N}(0, I)$ it has a closed form

# The Training Loss, Term by Term

Minimize the negative bound over the data:

$$\mathcal{J}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} \Big\{ \underbrace{- \mathbb{E}_{q_\phi(z|x_i)}[\log p_\theta(x_i \mid z)]}_{\text{reconstruction: } \theta \text{ and } \phi} + \underbrace{D_{\mathrm{KL}}(q_\phi(z \mid x_i) \,\|\, p(z))}_{\phi \text{ only, closed form}} \Big\}$$

**Reconstruction: sample it**

an expectation over $z \sim q_\phi(\cdot \mid x_i)$, estimated with one draw per image

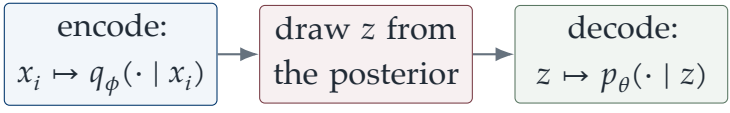
**KL: no sampling**

for Gaussian $q_\phi$ and $p(z) = \mathcal{N}(0, I)$ it has a closed form

The $\theta$-gradient passes inside the expectation; the $\phi$-gradient cannot, since $\phi$ decides which $z$ we draw — the **reparameterization trick** handles that next.

# Why It Is Called an Autoencoder

Read the reconstruction term for one image as a procedure:




It is scored by $-\log p_\theta(x_i \mid z)$: how well the decoded distribution explains the image we started from. The input is its own target, and that is the "auto".

# Reconstruction in the Distributional Sense

The same round trip, read as distributions rather than as one image:

$$\underbrace{x \sim p_{\text{data}},\ \ z \sim q_\phi(\cdot \mid x)}_{\text{encode the data}} \qquad \text{against} \qquad \underbrace{z \sim p(z),\ \ x \sim p_\theta(\cdot \mid z)}_{\text{decode the prior}}.$$

At a good fit the two agree: encoding data produces codes that look like prior draws, and decoding prior draws produces images that look like data — reconstruction of a **distribution**, not just of each image.

# Reconstruction in the Distributional Sense

The same round trip, read as distributions rather than as one image:

$$\underbrace{x \sim p_{\text{data}},\ \ z \sim q_\phi(\cdot \mid x)}_{\text{encode the data}} \qquad \text{against} \qquad \underbrace{z \sim p(z),\ \ x \sim p_\theta(\cdot \mid z)}_{\text{decode the prior}}.$$

At a good fit the two agree: encoding data produces codes that look like prior draws, and decoding prior draws produces images that look like data — reconstruction of a **distribution**, not just of each image.

The second term of the loss, $\frac{1}{n}\sum_i D_{\mathrm{KL}}(q_\phi(z \mid x_i) \,\|\, p(z))$, pulls the encoded codes towards $p(z)$.

# Gaussian Encoder and Reparameterization Trick

Recall that we use a Gaussian encoder

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

Sampling a code for a given image $x$ is then rewritten as

$$\epsilon \sim \mathcal{N}(0, I), \qquad z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon \quad \sim \quad q_\phi(\cdot \mid x).$$

Drawing $\epsilon$ first and then shifting and scaling it gives exactly the same distribution over $z$. What changes is where the randomness sits: outside $\phi$, in $\epsilon$, so that for a fixed $\epsilon$ the code is a differentiable function of the encoder outputs.

# Reparameterization Trick

**The problem.** We need $\nabla_\phi \mathbb{E}_{q_\phi}[L(z)]$, but $\phi$ sets the distribution being sampled, so the gradient has nothing to pass through.

# Reparameterization Trick

**The problem.** We need $\nabla_\phi \mathbb{E}_{q_\phi}[L(z)]$, but $\phi$ sets the distribution being sampled, so the gradient has nothing to pass through.

**The fix.** Move the randomness out of the parameters: with $z = \mu_\phi + \sigma_\phi \odot \epsilon$ and $\epsilon \sim \mathcal{N}(0, I)$, the quantity we average over no longer depends on $\phi$, so

$$\nabla_\phi \mathbb{E}_\epsilon[L(\mu_\phi + \sigma_\phi \odot \epsilon)] = \mathbb{E}_\epsilon[\nabla_\phi L(\mu_\phi + \sigma_\phi \odot \epsilon)].$$

# Reparameterization Trick

**The problem.** We need $\nabla_\phi \mathbb{E}_{q_\phi}[L(z)]$, but $\phi$ sets the distribution being sampled, so the gradient has nothing to pass through.

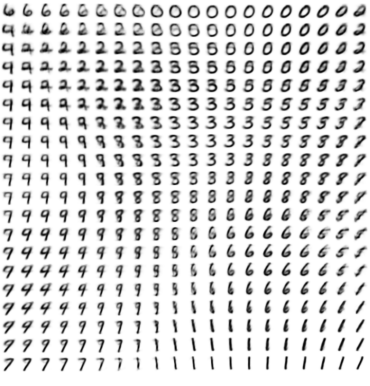
**The fix.** Move the randomness out of the parameters: with $z = \mu_\phi + \sigma_\phi \odot \epsilon$ and $\epsilon \sim \mathcal{N}(0, I)$, the quantity we average over no longer depends on $\phi$, so

$$\nabla_\phi \mathbb{E}_\epsilon[L(\mu_\phi + \sigma_\phi \odot \epsilon)] = \mathbb{E}_\epsilon[\nabla_\phi L(\mu_\phi + \sigma_\phi \odot \epsilon)].$$

**Draw $\epsilon$ once per image, then differentiate as usual: the network sees a deterministic function of $\phi$ and a fixed number.**

Source: The one-coordinate calculation is in Appendix A.6.

# A Decoded Two-Dimensional Manifold



Source: Kingma and Welling (2013), Figure 4(b), credited educational excerpt.

# Generating with the Prior and the Decoder

Once trained, generation is two draws and no encoder at all:

$$z \sim p(z) = \mathcal{N}(0, I), \qquad \text{then} \qquad x \sim p_\theta(\cdot \mid z).$$

The decoder network turns the code into the parameters of the second distribution — a mean for a Gaussian likelihood, pixel probabilities for a Bernoulli one — and the image is a draw from it.

The encoder was only ever scaffolding for training. It does not appear here.

# Outline

# A Small VAE on the Digits

Everything that follows comes from one fitted model:

- Encoder $64 \to 128 \to 128 \to (\mu, \log \sigma^2)$ with $r = 2$; decoder the mirror image, Bernoulli likelihood over pixels;
- Trained by Adam on the evidence lower bound, with one reparameterized draw per image per step.

A two-dimensional code is far too small for this data. That is deliberate: it makes the whole code space visible on one slide.

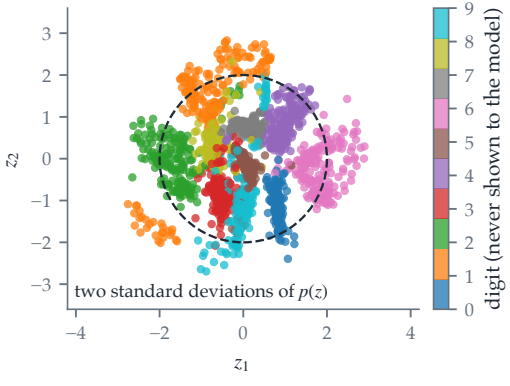
Source: Course PyTorch training run, seed 26520; 1,797 digits, 400 epochs.

# Reconstructions: Recognizable and Blurry


Each digit passes through a two-number bottleneck and comes back identifiable — but softened, with the sharp strokes averaged away. The blur is not a bug in the fit; it is what a likelihood over pixels rewards when the code cannot specify the details.

Course VAE, seed 26520; first occurrence of each digit; decoder means shown.

# The Code Space, Coloured by Digit




The encoder was never shown a label, yet the ten digits occupy distinguishable regions of the plane, and the codes sit where the prior expects them — mostly inside two standard deviations. Neighbouring classes overlap, which is what a two-dimensional code forces.

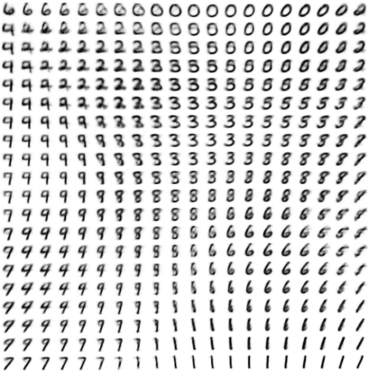
Course VAE, seed 26520; posterior means $\mu_\phi(x_i)$ for all 1,797 digits.

# Samples from the Prior

Thirty codes drawn from $p(z) = \mathcal{N}(0, I)$ and decoded. No encoder is involved. Most are recognizable digits; a few land between classes and come out as shapes that are not any digit at all.

Course VAE, seed 26520; decoder means for thirty prior draws.

# A Decoded Two-Dimensional Manifold



Source: Kingma and Welling (2013), Figure 4(b), credited educational excerpt: the same construction on full-size MNIST.

# Outline

# Two Networks, One Bound

Training is harder than the objective makes it look:

- The encoder and decoder are fitted against each other. Each one's target moves as the other improves, so early training optimizes a bound around a decoder that cannot yet reconstruct anything.
- The gradient in $\phi$ is a sampled quantity. One draw per image makes it unbiased but noisy.
- What we maximize is a lower bound, not the likelihood. A larger ELBO can come from a tighter $q_\phi$ rather than a better model, so the number we watch does not cleanly report progress.

# The Two Terms Pull Against Each Other

$$\underbrace{-\,\mathbb{E}_{q_\phi}[\log p_\theta(x \mid z)]}_{\text{wants an informative code}} \qquad \text{versus} \qquad \underbrace{D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p(z))}_{\text{wants every code to look like } p(z)}$$

A code that carries a lot about $x$ must differ from image to image, and that is exactly what the KL charges for. The **$\beta$-VAE** makes the trade explicit by reweighting the second term:

$$-\mathbb{E}_{q_\phi}[\log p_\theta(x \mid z)] + \beta\, D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p(z)), \qquad \beta = 1 \text{ recovers the ELBO.}$$

# What $\beta$ Does, Measured


At $\beta = .2$ both code coordinates carry information and the digits come back sharp. At $\beta = 6$ the KL is driven to exactly zero in both coordinates: the encoder has stopped reading the image, and the decoder returns the same blur for every input.

Course VAE, seed 26531; 250 epochs at each $\beta$; the same three digits shown throughout.

# Posterior Collapse

The failure at $\beta = 6$ has a name — posterior collapse. If

$$q_\phi(z \mid x) = p(z) \quad \text{for every } x,$$

the KL term is zero and the code carries nothing about the image. The decoder falls back on the average digit.

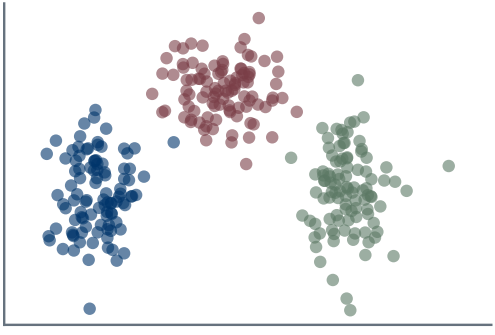
It happens for two different reasons:

- The KL is weighted too heavily, as above;
- The decoder is powerful enough not to need the code: it can model the data alone, so ignoring $z$ saves the whole KL and costs nothing.

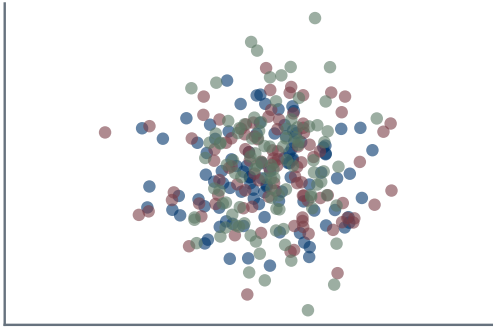
Diagnose it by watching the KL per coordinate, not the total loss.

# A Powerful Decoder Can Ignore the Code

Posterior depends on x



Every posterior resembles prior



KL per coordinate under increasing decoder capacity.

# Where This Leads Next

Two limitations point in two directions.

**Continuous codes blur**

a Gaussian code and a per-pixel likelihood average over the detail. Make the code discrete instead and the decoder can commit to sharp alternatives — the **VQ-VAE**, which we turn to next

**The encoder is the hard part**

it must be learned, jointly, through a sampled gradient. Replace it with a fixed process that needs no training — **diffusion**, Lecture 18

# Where This Leads Next

Two limitations point in two directions.

**Continuous codes blur**

a Gaussian code and a per-pixel likelihood average over the detail. Make the code discrete instead and the decoder can commit to sharp alternatives — the **VQ-VAE**, which we turn to next

**The encoder is the hard part**

it must be learned, jointly, through a sampled gradient. Replace it with a fixed process that needs no training — **diffusion**, Lecture 18

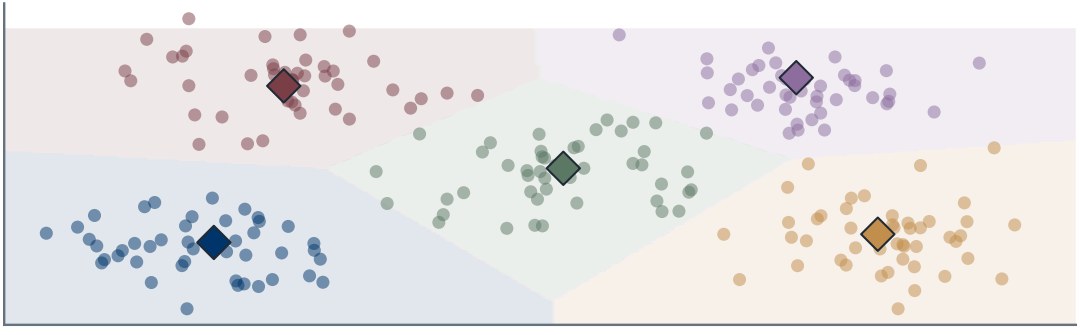
**Both keep the bound from this lecture. They change only what plays the role of $q$.**

# Outline

# Why Use a Discrete Code?

| Compression | Abstraction | Modeling |
| --- | --- | --- |
| store one small integer per latent location | discard small encoder changes inside one code region | learn a sequence prior over image, audio, or other tokens |

# Nearest-Neighbor Codebook Assignment



Seed 26520; diamonds are learned code vectors.

# The Model: Encoder, Codebook, Decoder

The encoder is now an ordinary network returning one vector, not a distribution:

$$z_e = e_\phi(x) \in \mathbb{R}^m.$$

Alongside it we keep a **codebook**, $K$ vectors $e_1, \dots, e_K \in \mathbb{R}^m$, and replace $z_e$ by its nearest entry:

$$k^\star = \arg\min_k \|z_e - e_k\|_2^2, \qquad z_q = e_{k^\star}.$$

The decoder then reads $z_q$. An image is summarized by the integer $k^\star$. With one code per image this is a hard clustering of the code space; real systems quantize a grid of locations, giving a grid of integers.

# What Is Learnable

| | what it is | trained by |
|---|---|---|
| $\phi$ | encoder weights | reconstruction and commitment |
| $e_1, \dots, e_K$ | the codebook: $K$ vectors in $\mathbb{R}^m$ | the codebook term |
| $\theta$ | decoder weights | reconstruction |

Notice what is absent: no $\sigma_\phi$, no sampling, and no KL to a prior. The VQ-VAE is trained as a deterministic autoencoder whose bottleneck is a lookup in a table.

# The Loss Function of VQ-VAE

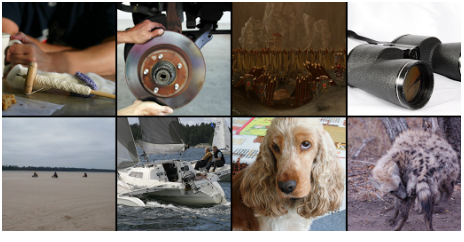
$$-\log p_\theta(x \mid z_q) \;+\; \|\mathrm{sg}[z_e] - e_{k^\star}\|_2^2 \;+\; \beta\,\|z_e - \mathrm{sg}[e_{k^\star}]\|_2^2$$

| term | what it asks for | parameters it updates |
|---|---|---|
| reconstruction | decode $z_q$ back to $x$ | $\theta$, and $\phi$ |
| codebook | move the chosen code onto $z_e$ | $e_{k^\star}$ only |
| commitment | move $z_e$ onto the code it chose | $\phi$ only |

The last two measure the same distance; the stop-gradient $\mathrm{sg}[\cdot]$ decides which end of it moves.

Source: How $\phi$ gets a gradient through a nearest-neighbour lookup: Appendix A.7.

# Discrete Reconstructions



originals

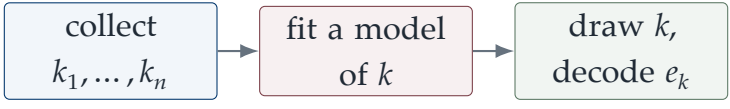
reconstructions

Source: van den Oord, Vinyals, and Kavukcuoglu (2017), Figure 2, credited educational excerpt.

# Generation Needs a Second Model

Write $k_i = k^\star(x_i)$ for the index the encoder assigns to image $i$. A VAE could generate by drawing $z \sim p(z)$, because the KL term had pushed the codes towards that prior; VQ training has no such term, so the distribution of $k_1, \dots, k_n$ is unknown and certainly not uniform.

So generation is a **second stage**, fitted once the autoencoder is finished:




One integer per image here, so the second model is a distribution over $\{1, \dots, K\}$. Quantizing a grid of locations makes each image a sequence of indices, and stage two autoregressive.

# Summary: Continuous Latent Models

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

1. Restricting $q$ to a family turns inference into optimization, at the cost of a gap that never closes.
2. PPCA is the linear-Gaussian case where that gap is zero and the fitted subspace is PCA's.
3. An encoder amortizes the per-observation optimization, and reparameterization makes it differentiable.

# Summary: Tradeoffs and Discrete Codes

$$k^* = \arg\min_k \|z_e - e_k\|^2, \qquad z_q = e_{k^*}.$$

1. Reconstruction and KL pull against each other; $\beta$ names the trade, and too much KL collapses the code.
2. Watch the KL per coordinate: the total loss hides a collapsed latent.
3. Quantizing replaces the sampled code by a lookup, and moves generation into a second model over indices.

# Next Lecture

Every difficulty in this lecture traces back to the encoder: it has to be learned, jointly with the decoder, through a sampled gradient.

**Lecture 18 — Diffusion, Latent Diffusion, and Flow Matching** replaces it with a fixed noising process that has nothing to learn, so only the reverse direction is fitted — against the same bound derived today.

Reading for this lecture: DL Chapter 14; Kingma and Welling (2019), *An Introduction to Variational Autoencoders*.

# A.1 The PPCA Posterior

derive

Collect the terms of $\log p(z) + \log p(x \mid z)$ that involve $z$:

$$-\frac{1}{2} z^\top z - \frac{1}{2\sigma^2} \|x - \mu - Wz\|_2^2 = -\frac{1}{2\sigma^2} \left[ z^\top M z - 2 z^\top W^\top (x - \mu) \right] + \text{const},$$

with $M = W^\top W + \sigma^2 I_r$. Completing the square gives a Gaussian,

$$p(z \mid x) = \mathcal{N}(M^{-1} W^\top (x - \mu),\ \sigma^2 M^{-1}), \qquad \mathbb{E}[z z^\top \mid x] = \sigma^2 M^{-1} + m m^\top.$$

The covariance does not involve $x$, so it is shared by every observation.

# A.2 Closed-Form EM Updates for PPCA

derive

Writing $m_i = \mathbb{E}[z_i \mid x_i]$ and $S = \mathrm{Cov}(z_i \mid x_i)$, the M-step minimizers are

$$W^{\text{new}} = \Big[\sum_i (x_i - \mu)\, m_i^\top\Big]\Big[\sum_i (m_i m_i^\top + S)\Big]^{-1},$$

$$(\sigma^2)^{\text{new}} = \frac{1}{nd}\sum_{i=1}^{n} \mathbb{E}\big[\|x_i - \mu - W^{\text{new}} z_i\|_2^2 \mid x_i\big], \qquad \hat{\mu} = \bar{x}.$$

Setting $S = 0$ recovers the ordinary least-squares fit of $x$ on known codes.

Source: Tipping and Bishop (1999).

# A.3 The Posterior with Orthogonal Loadings

derive

If $W = [a_1 u_1, \dots, a_r u_r]$ with $u_j$ orthonormal, then $M = \operatorname{diag}(a_j^2 + \sigma^2)$ and the posterior separates:

$$z_j \mid x \;\sim\; \mathcal{N}\!\left( \frac{a_j}{a_j^2 + \sigma^2}\, u_j^\top (x - \mu),\;\; \frac{\sigma^2}{a_j^2 + \sigma^2} \right).$$

The mean is the projection score $u_j^\top (x - \mu)$, shrunk towards zero; the variance says what the data leave undetermined. As $\sigma^2 \to 0$ the shrinkage disappears and the posterior concentrates on the PCA score.

# A.4 The ELBO Identity

derive

For any distribution $q$ whose support covers the posterior, define

$$\mathcal{L}(q, \theta) = \mathbb{E}_q[\log p_\theta(x, z) - \log q(z)].$$

Factor the joint the first way, $p_\theta(x, z) = p_\theta(z \mid x)\, p_\theta(x)$:

$$\begin{aligned}\mathcal{L}(q, \theta) &= \mathbb{E}_q[\log p_\theta(z \mid x) + \log p_\theta(x) - \log q(z)] \\ &= \log p_\theta(x) - \mathbb{E}_q\left[\log \frac{q(z)}{p_\theta(z \mid x)}\right] = \log p_\theta(x) - D_{\mathrm{KL}}(q \,\|\, p_\theta(z \mid x)).\end{aligned}$$

Since $D_{\mathrm{KL}} \geq 0$, this single line gives both the bound and the size of the gap; Jensen's inequality is not needed.

# A.5 Four Forms of the Same Identity

derive

Factoring the joint the second way, $p_\theta(x,z) = p(z)\,p_\theta(x \mid z)$, gives the remaining forms. Writing $H(q) = -\mathbb{E}_q[\log q(z)]$:

| $\mathcal{L}(q,\theta)$ equals | useful for |
|---|---|
| $\log p_\theta(x) - D_{\mathrm{KL}}(q \,\|\, p_\theta(z \mid x))$ | the E-step: the best $q$ is the posterior |
| $\mathbb{E}_q[\log p_\theta(x \mid z)] - D_{\mathrm{KL}}(q \,\|\, p(z))$ | training: reconstruction against a prior |
| $\mathbb{E}_q[\log p_\theta(x,z)] + H(q)$ | the classical energy-plus-entropy form |
| $\mathbb{E}_q[\log p_\theta(x,z)] + \text{const}$ | the M-step: only this term involves $\theta$ |

Line one uses $p_\theta(z \mid x)p_\theta(x)$, line two uses $p(z)p_\theta(x \mid z)$, and the last two follow by grouping the $q$ terms and by holding $q$ fixed. All four are the same quantity.

# A.6 The Pathwise Gradient

derive

For one coordinate $z = \mu + \sigma\epsilon$,

$$\frac{\partial z}{\partial \mu} = 1, \qquad \frac{\partial z}{\partial \sigma} = \epsilon.$$

If decoder loss is $L(z)$, the chain rule gives

$$\frac{\partial L}{\partial \mu} = \frac{\partial L}{\partial z}, \qquad \frac{\partial L}{\partial \sigma} = \frac{\partial L}{\partial z}\epsilon.$$

Randomness is outside the differentiable parameter path, enabling low-variance stochastic gradients.

# A.7 The Straight-Through Gradient

derive

The index $k^\star = \arg\min_k \|z_e - e_k\|_2^2$ is piecewise constant in $z_e$, so $\partial z_q / \partial z_e = 0$ almost everywhere and the encoder would receive no gradient at all from the reconstruction term. The straight-through estimator puts

$$z_q = z_e + \mathrm{sg}[e_{k^\star} - z_e]$$

into the graph: the forward value is unchanged, $z_q = e_{k^\star}$, while the backward pass sees $\partial z_q / \partial z_e \approx I$, so the decoder's gradient is copied straight onto the encoder output — a biased estimator, justified by working.

# A.8 The Gaussian KL Term

derive

For diagonal $q = \mathcal{N}(\mu, \mathrm{diag}(\sigma^2))$ and $p = \mathcal{N}(0, I)$,

$$\begin{aligned} D_{\mathrm{KL}}(q\|p) &= \mathbb{E}_q[\log q(z) - \log p(z)] \\ &= \frac{1}{2}\sum_{j=1}^{r}\left(\mu_j^2 + \sigma_j^2 - 1 - \log \sigma_j^2\right). \end{aligned}$$

The term is zero exactly when every posterior coordinate has $\mu_j = 0$ and $\sigma_j = 1$.