Autoencoders & VAEs

Lecture 19 · ES 667: Deep Learning

Prof. Nipun Batra
IIT Gandhinagar · Aug 2026

Learning outcomes

By the end of this lecture you will be able to:

1. State what a **plain autoencoder** is and why it isn't generative.
2. Motivate the need for a **prior distribution** on latent space.
3. Write the **ELBO** from scratch using Jensen's inequality.
4. Derive the **KL term** for Gaussian posterior vs standard normal.
5. Explain and implement the **reparameterization trick**.
6. Train a **β-VAE** and discuss disentanglement.
7. Place VAEs in context · pre-compressor for Stable Diffusion in 2026.

Where we are

Module 9 opens · **generative models**. Until now every model *classified* or *predicted* — labels, tokens, pixels-given-labels. Today we switch to: **given a dataset, can I sample new examples that look like it?**

> REFERENCE
>
> Today maps to **Prince Ch 17** (Variational Autoencoders) + Kingma & Welling 2013.

Four questions:

1. What's a plain **autoencoder** and why isn't it generative?
2. What does **VAE** add, and why does it work?
3. What is the **reparameterization trick**?
4. Where do VAEs fit in the 2026 generative landscape?

Generative modeling · the task

KEY IDEA

Given N i.i.d. samples $\{x_1, \ldots, x_N\}$ from an unknown distribution p_{data}, learn a model from which we can **sample** new $x \sim p_{\text{model}}$ such that $p_{\text{model}} \approx p_{\text{data}}$.

Two sub-tasks, often pursued together:

1. **Density estimation** · assign a probability $p_{\text{model}}(x)$ to any candidate sample.
2. **Generation** · draw novel samples from p_{model}.
   Images live in $\mathbb{R}^{100{,}000+}$ on a low-dimensional manifold. Writing down p_{data} analytically is hopeless; learning it from samples is the whole game.

Three generative strategies

Density-based

Write $p_\theta(x)$ explicitly; maximize $\sum_i \log p_\theta(x_i)$.

- Pixel-RNN, PixelCNN, Normalizing flows.
- Clean likelihoods.
- Autoregressive or invertible only.

Latent-based

Hidden variable z generates x: $x \sim p(x|z), z \sim p(z)$.

- **VAE**, GAN (implicit).
- Compact latent, rich samples.
- Likelihood tractable only via ELBO.

Diffusion (L21) is a *layered* latent model with T levels. The recipe of "add structure in latent space" starts here with the VAE.

PART 1

The generative model family tree

A brief taxonomy

Four families of generative models

FAMILY	HOW IT SAMPLES	TRAINING
VAE (L19)	sample z ~ p(z), decode	ELBO
GAN (L20)	sample z ~ p(z), generator	minimax
Normalizing flows	invertible transforms of p(z)	exact likelihood
Diffusion (L21-22)	iterative denoising from noise	score matching / denoising

Today is VAE. Each family has tradeoffs between sample quality, training stability, and tractability.

PART 2

Autoencoder first

The building block

Autoencoder · the postcard analogy

KEY IDEA

A perfect forger writes the **most compact possible description** of a painting on a postcard (the latent code · maybe 16 numbers).

They mail it to their partner. The partner must **recreate the original painting** using only the postcard.

If the postcard is too small, they're forced to learn what's *truly essential* · the "essence" of the painting · not every brushstroke. That's compression. That's what an autoencoder learns.

Postcard analogy · in math terms

INTUITION

- **Forger** writes the postcard → **encoder** f.
- **Postcard** = compact description → **latent code** z.
- **Partner** reconstructs the painting → **decoder** g.
- "Goodness" of reconstruction → **MSE loss** $\|x - g(f(x))\|^2$.

Train encoder + decoder together. The bottleneck $d \ll n$ forces the network to keep only the most informative features.

Worked numeric · 4-pixel autoencoder

Tiny grayscale image $x = [0.9, 0.2, 0.8, 0.1]$. Latent dim $d = 1$.

1. **Encode.** Network outputs $z = 0.7$.
2. **Decode.** Network maps $z = 0.7 \to \hat{x} = [0.8, 0.3, 0.7, 0.2]$.
3. **Loss (MSE).**
   $\mathcal{L} = \frac{1}{4}\left((0.9-0.8)^2 + (0.2-0.3)^2 + (0.8-0.7)^2 + (0.1-0.2)^2\right)$
   $= \frac{1}{4}(0.01 + 0.01 + 0.01 + 0.01) = \mathbf{0.01}$

   Backprop adjusts encoder + decoder weights to push this lower.

Uses:

- Denoising · dimensionality reduction · pretraining / feature learning.
- Beats PCA for non-linear data.

Autoencoder vs PCA · what's added

PCA is **the** linear autoencoder with orthogonal weights. What does nonlinearity buy you?

DERIVATION

- PCA forces the latent space to be *linear subspace*. Fine for Gaussian-like data; poor for curved manifolds.
- An autoencoder (MLP or CNN) can fold arbitrary manifolds — digits on a swiss-roll latent, faces on a curved surface, etc.

Concretely · PCA on MNIST reaches ~85% explained variance with 32 dims; a deep AE matches the full-data reconstruction at ~16 dims. Curved manifold vs linear subspace · nonlinearity buys 2× compression.

Bottleneck intuition · why it's crucial

If the latent $d \geq n$, the network can just copy · $z = x$, $g(z) = z$. Loss is zero but nothing learned.

KEY IDEA

The bottleneck $d \ll n$ **forces compression** · the network must keep only the most informative features. Anything redundant gets dropped. This is why autoencoders produce useful representations even without labels.

Modern variants add noise (denoising AE) or masking (MAE, L17) *instead of* a small bottleneck — same idea, different forcing.

A concrete AE · MNIST dimensionality

DERIVATION

Input · `28 × 28 = 784` pixels. Encode to latent `z` of size 16. Decode back to 784.

LAYER	SHAPE	PARAMS
Input	784	—
Linear → ReLU	256	200,960
Linear → ReLU	64	16,448
Linear (μ only)	16	**bottleneck** · 1,040
Linear → ReLU	64	1,088
Linear → ReLU	256	16,640
Linear → sigmoid	784	201,488

Total · ~440k params. Reconstruction MSE on MNIST test · ~0.003 after 10 epochs. Compare PCA with 16 components · ~0.015. **5× better with nonlinearities.**

But autoencoders aren't generative

Suppose you train an AE on MNIST. To *generate* a new digit, you'd:

1. Pick a random z in latent space.
2. Decode it.

What happens? Usually garbage. Why?

> WATCH OUT
>
> The latent space is **irregular**. The encoder only learned to map *actual training images* to latent points. Random z values likely fall into "nothing-mapped-here" regions where the decoder is undefined.

You'd need the latent space to be **dense** and **structured** — that's what VAE adds.

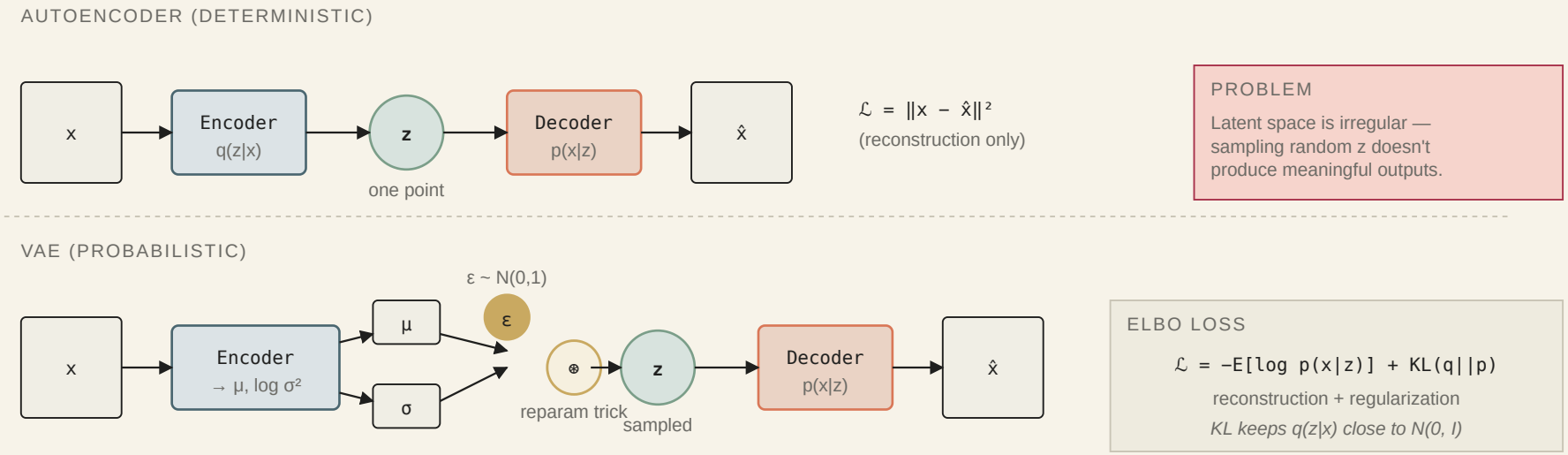
PART 3

The VAE fix

A prior and a KL penalty

AE vs VAE

Autoencoder vs VAE — a deterministic point vs a distribution in latent space



*VAE's KL term **regularizes the latent space** toward a standard normal → sampling random z produces valid outputs → **you can generate**.*

IN PRACTICE

▶ Interactive: slide the KL weight β, watch the latent space go from clumpy to Gaussian — vae-latent-explorer.

Why a prior? · two jobs it does

The prior $p(z) = \mathcal{N}(0, I)$ does two things for us:

1. Defines the sampling distribution

At generation time we draw $z \sim p(z)$ and decode. The prior is the *rule book* for producing valid z's.

Without a prior, you wouldn't know how to initialize z for generation.

2. Regularizes the posterior

The KL term pulls $q(z|x)$ toward $p(z)$ for every training example. Every encoded posterior overlaps in the same region → smooth latent.

Without this, training points occupy disjoint clusters.

KEY IDEA

A VAE is a plain AE **with a regularizer that makes the latent space match a known distribution**. Everything else follows from making that regularizer principled (the ELBO).

VAE · the encoder outputs a distribution

The encoder no longer outputs a point z. It outputs **parameters of a Gaussian**:

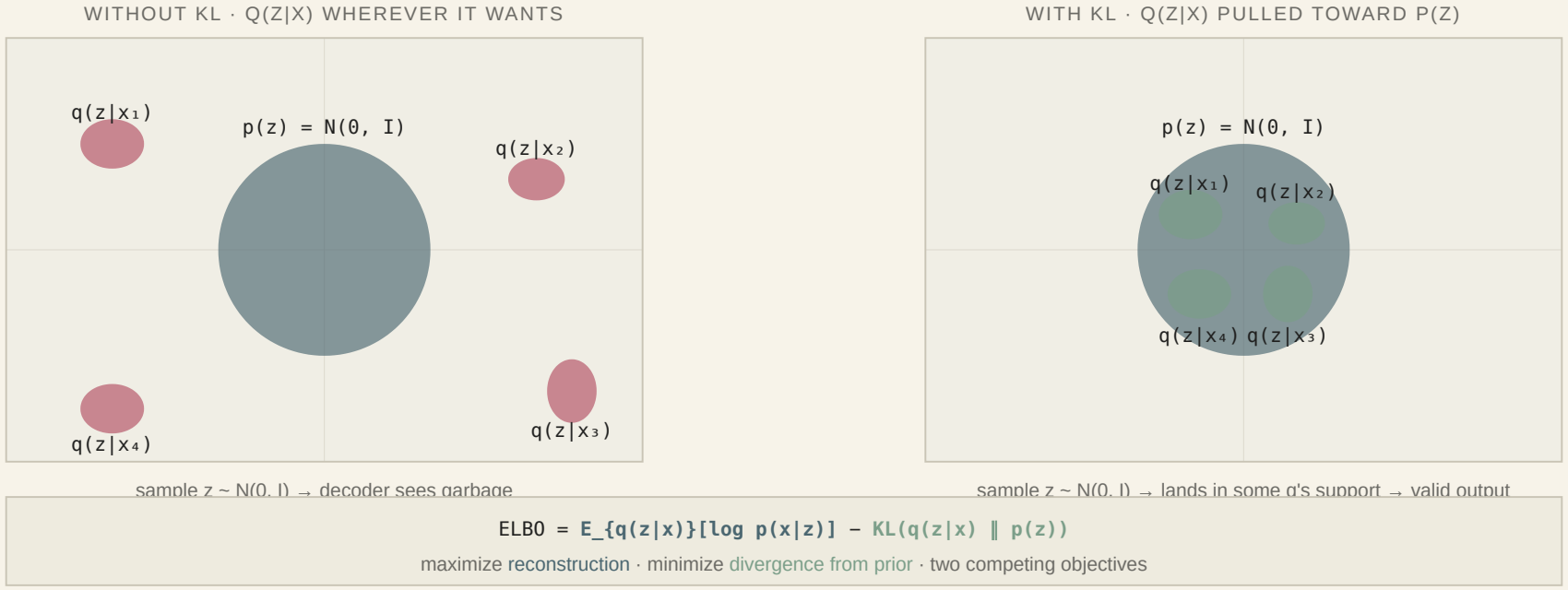
$$q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \sigma^2_\phi(x))$$

Both μ and σ are network outputs. During training:

1. Given x, get $\mu(x), \sigma(x)$.
2. **Sample** $z \sim q_\phi(z|x)$.
3. Decode · $\hat{x} = g_\theta(z)$.
4. Loss · reconstruction **+** KL divergence to prior.

ELBO geometry

ELBO geometry · KL pulls the approximate posterior q(z|x) toward the true prior p(z)



$$\text{ELBO} = \mathbb{E}_{q(z|x)}[\log p(x|z)] - \text{KL}(q(z|x) \,\|\, p(z))$$

maximize reconstruction · minimize divergence from prior · two competing objectives

The punchline · you can skip the derivation

KEY IDEA

The VAE loss is just **reconstruction + KL-to-prior**. Train to minimize it. That's all you need to use a VAE.

$$\mathcal{L} = \underbrace{\|x - \text{decode}(z)\|^2}_{\text{reconstruction}} + \underbrace{\text{KL}(q(z|x) \,\|\, \mathcal{N}(0, I))}_{\text{regularizer}}$$

The next two slides *derive* this from first principles (Jensen's inequality). **If you trust me, you can skip them** · come back to the math later.

ELBO · the hard problem and the easy trick

Hard problem. We want $p(x) = \int p(x, z)\, dz$. Intractable for high-dim z.

Easy trick (variational inference).

1. Define a tractable distribution $q(z|x)$ (our encoder).
2. Derive a **lower bound** on $\log p(x)$ that we *can* compute: the **ELBO**.
3. Maximizing the ELBO pushes up $\log p(x)$ — like getting good practice-exam grades.

Deriving the ELBO · step by step

Step 1. Multiply and divide by $q(z|x)$:

$$\log p(x) = \log \int q(z|x) \cdot \frac{p(x,z)}{q(z|x)}\, dz = \log \mathbb{E}_q\left[\frac{p(x,z)}{q(z|x)}\right]$$

Step 2 · Jensen's inequality. $\log$ is concave → $\log \mathbb{E}[X] \geq \mathbb{E}[\log X]$. (For two values: $\log \frac{a+b}{2} \geq \frac{\log a + \log b}{2}$.)
Move log inside expectation:

$$\log p(x) \geq \mathbb{E}_q\left[\log \tfrac{p(x,z)}{q(z|x)}\right] \quad \text{(ELBO)}$$

Step 3 · expand. Using $p(x,z) = p(x|z)\,p(z)$:

$$\text{ELBO} = \mathbb{E}_q[\log p(x|z)] + \mathbb{E}_q[\log p(z) - \log q(z|x)]$$

The second bracket is exactly $-D_{\text{KL}}(q(z|x) \,\|\, p(z))$:

$$\boxed{\log p(x) \geq \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{\text{KL}}(q(z|x) \,\|\, p(z))}$$

- **First term** · reconstruction (maximize).

The ELBO · reconstruction + KL

DERIVATION

Evidence Lower Bound — a tractable lower bound on $\log p(x)$:

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

- **First term** · data likelihood under the decoder, averaged over encoder samples.
- **Second term** · how far the posterior q is from the prior $p(z) = \mathcal{N}(0, I)$.

Maximize the ELBO = minimize the negative. This is the VAE loss. Every term is tractable and backpropagatable.

KL · the "cost of being different"

The KL term is the price for $q(z|x)$ deviating from the standard-normal prior. Two parts:

1. **Mean cost** μ^2 · zero when $\mu = 0$, grows quadratically.
2. **Variance cost** $\sigma^2 - \log \sigma^2 - 1$ · zero at $\sigma^2 = 1$, positive everywhere else.

For Gaussian q vs $\mathcal{N}(0, 1)$:

$$D_{\mathrm{KL}} = \tfrac{1}{2} \sum_{i=1}^{d} \left(\mu_i^2 + \sigma_i^2 - 1 - \log \sigma_i^2 \right)$$

Quick check on the variance term:

- $\sigma^2 = 1 \Rightarrow 1 - 1 - 0 = 0$ ✓
- $\sigma^2 = 0.5 \Rightarrow 0.5 - 1 - (-0.693) = 0.193$ (penalty)
- $\sigma^2 = 2 \Rightarrow 2 - 1 - 0.693 = 0.307$ (penalty)

The VAE pushes $\mu \to 0$ and $\sigma^2 \to 1$ unless the reconstruction term needs different values.

KL · worked numeric example

Suppose $q(z|x) = \mathcal{N}(\mu = 1.2, \sigma^2 = 0.25)$ for a particular image x. Standard normal prior $p(z) = \mathcal{N}(0, 1)$.

DERIVATION

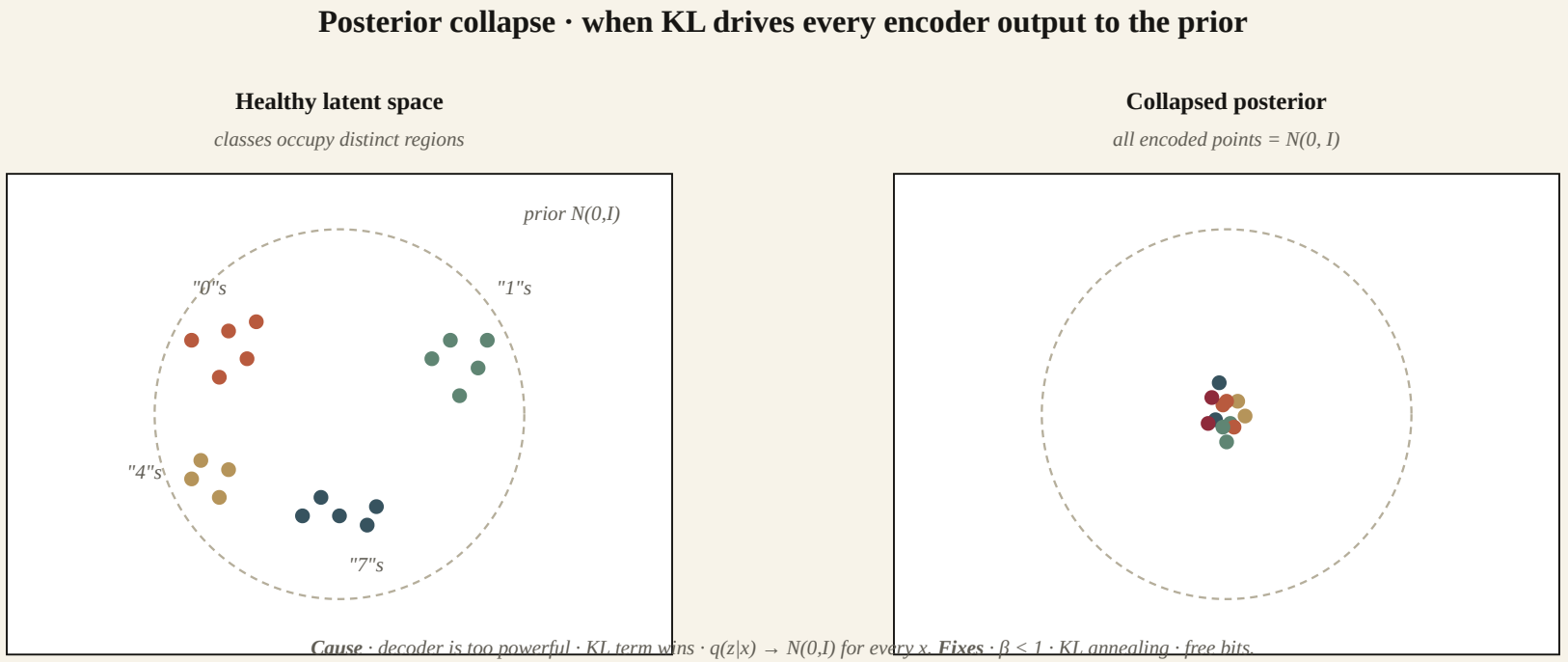
$$\mathrm{KL} = \frac{1}{2}(\sigma^2 + \mu^2 - 1 - \log \sigma^2)$$

$$= \frac{1}{2}(0.25 + 1.44 - 1 - \log 0.25) = \frac{1}{2}(0.69 + 1.386) = 1.038$$

Plugging in: $\sigma^2 = 0.25$ means **narrow** posterior (confident encoding); $\mu = 1.2$ means **off-centre** from the prior. The KL penalty of ~1.0 will pull μ back toward 0 during training, unless the reconstruction term needs a wide-apart μ to distinguish this image from others.

This is the trade-off the VAE balances at every sample.

Posterior collapse · the picture



Posterior collapse · what to watch for

If the decoder is too powerful, the KL term will drive $q(z|x) \to p(z) = \mathcal{N}(0, I)$ · every image encodes to the same latent, z carries no information about x.

> WATCH OUT
>
> **Posterior collapse** · the VAE becomes an autoencoder where z is just noise. Reconstructions are fine (the decoder ignores z), but samples are junk (there's no latent structure to exploit).

Fixes:

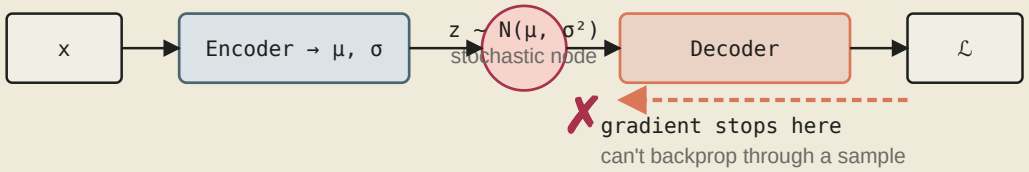
- Reduce decoder capacity (smaller FFN).
- Use β-VAE with $\beta < 1$ (less KL weight).
- KL annealing · start with $\beta = 0$, ramp up over training.
- Free bits · allow some KL "for free" before penalizing.

PART 4

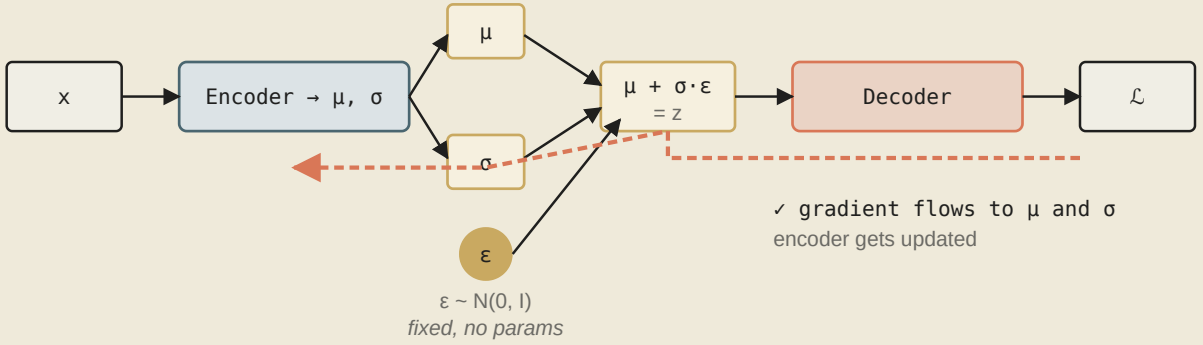
Reparameterization in one picture

Reparameterization trick · move randomness off the gradient path

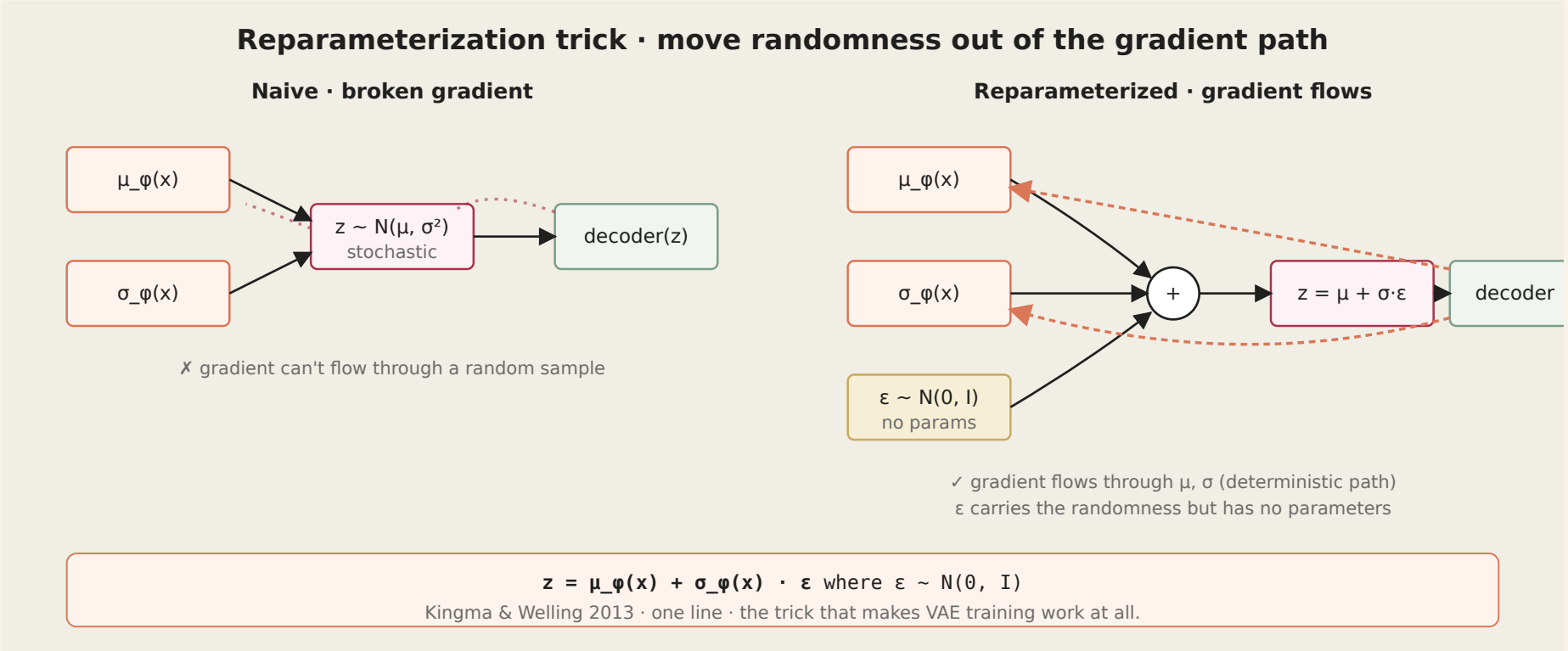
NAIVE · SAMPLING BLOCKS THE GRADIENT



REPARAMETERIZED · Z = M + Σ·E MOVES E OFF THE GRADIENT PATH



Reparameterization · gradient flow



Reparameterization · the conveyor-belt analogy

KEY IDEA

Imagine training a robot arm that **randomly picks** a part from a bin. You can't train the picking motion · "your random pick was wrong" gives no gradient.

Now change the system · the robot picks a **specific** part from a conveyor belt. The randomness is in *how the belt is loaded*, not in the robot's motion.

The belt-loading randomness ≡ the noise ϵ. The robot's picking motion ≡ the deterministic map $z = \mu + \sigma\epsilon$. Now we *can* compute gradients through the picking motion · which is exactly what we needed to train μ and σ.

The reparameterization trick

How to backprop through a sample

Reparameterization trick · making randomness differentiable

Problem. We need $z \sim \mathcal{N}(\mu, \sigma^2)$. But `sample_from_gaussian(μ, σ)` is a black box — no gradient through it.

Trick (Kingma & Welling 2013). Any sample from $\mathcal{N}(\mu, \sigma^2)$ can be written:

$$z = \mu + \sigma \cdot \epsilon, \qquad \epsilon \sim \mathcal{N}(0, 1)$$

The randomness now sits **outside** μ, σ. The path from z back to μ, σ is just deterministic add + multiply → gradients flow.

Worked numeric. Encoder outputs $\mu = 2.0$, $\sigma = 0.5$. Sample $\epsilon = 1.2$.
$z = 2.0 + 0.5 \cdot 1.2 = 2.6$.
Suppose $\partial \mathcal{L} / \partial z = 4.0$.

Then:

- $\partial z / \partial \mu = 1 \rightarrow \partial \mathcal{L} / \partial \mu = 4.0 \cdot 1 = \mathbf{4.0}$
- $\partial z / \partial \sigma = \epsilon = 1.2 \rightarrow \partial \mathcal{L} / \partial \sigma = 4.0 \cdot 1.2 = \mathbf{4.8}$

  Optimizer can now update μ, σ as usual.

VAE in PyTorch · the whole thing

```python
class VAE(nn.Module):
    def __init__(self, d_in, d_z):
        super().__init__()
        self.enc = nn.Sequential(nn.Linear(d_in, 256), nn.ReLU(),
                                 nn.Linear(256, 2 * d_z))
        self.dec = nn.Sequential(nn.Linear(d_z, 256), nn.ReLU(),
                                 nn.Linear(256, d_in))

    def forward(self, x):
        h = self.enc(x)
        mu, log_var = h.chunk(2, dim=-1)

        # Reparam: z = mu + sigma · eps
        std = torch.exp(0.5 * log_var)
        eps = torch.randn_like(std)
        z = mu + std * eps

        recon = self.dec(z)

        # KL against N(0, I)
        kl = -0.5 * (1 + log_var - mu.pow(2) - log_var.exp()).sum(-1)

        return recon, kl
```

Entire VAE in 15 lines. The trick is making sure the KL goes in the loss alongside reconstruction.

Training loop

```
for x in loader:
    recon, kl = model(x)
    recon_loss = F.mse_loss(recon, x, reduction='none').sum(-1)
    loss = (recon_loss + BETA * kl).mean()
    opt.zero_grad(); loss.backward(); opt.step()
```

Tuning `BETA`:

- `BETA = 1` · standard VAE (follows the ELBO derivation).
- `BETA > 1` · β-VAE (Higgins 2017). Stronger regularization → more disentangled, often blurrier.
- `BETA < 1` · more weight to reconstruction, less to latent structure.

β · the seesaw between recon and KL

VAE loss · two terms in tension · β knob trades them off

β = 0 (plain AE)

no KL · latent unconstrained

recon · 0.5 (low)
KL · n/a

→ *great reconstruction*
→ *samples = garbage*

β = 1 (standard VAE)

balance · ELBO derivation

recon · 0.7 (slightly worse)
KL · 1.0 (latent constrained)

→ *decent reconstruction*
→ *samples plausible*

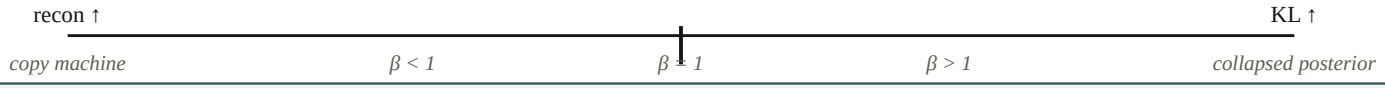
β = 4 (β-VAE)

over-regularize · disentangled

recon · 1.2 (blurrier)
KL · 0.4 (very tight to prior)

→ *blurry reconstruction*
→ *disentangled latent dims*

The seesaw



Disentanglement · what β-VAE buys you

With $\beta = 4$ on a faces dataset (Higgins 2017), each latent dimension starts to control ONE semantic factor:

DERIVATION

- z_1 · azimuth (face angle)
- z_2 · lighting direction
- z_3 · smile / frown
- z_4 · hairstyle length
- ...

INTUITION

No supervision — the structure emerges from the KL regularization plus the reconstruction pressure. Disentanglement lets you do **editable generation** · "same face with different smile" by perturbing one z coordinate.

The trade-off · stronger KL forces shared structure, but loses reconstruction detail. β = 1 is the theoretical sweet spot; higher β sacrifices quality for interpretability.

Conditional VAE · putting labels into the game

If you have class labels y, a **Conditional VAE** extends the game:

DERIVATION

- Encoder: $q_\phi(z \mid x, y)$
- Decoder: $p_\theta(x \mid z, y)$
- Prior: $p(z) = \mathcal{N}(0, I)$ unchanged.

At inference · sample $z \sim \mathcal{N}(0, I)$, fix y to the desired class, decode. Generate class-specific samples without retraining.

IN PRACTICE

CVAE was used for controllable generation before diffusion + CFG took over. Still shipped in some specialized systems (molecule generation, time-series imputation).

PART 5

Generating with a VAE

Worked example · one VAE forward pass

Suppose · $d_x = 4$ (4-pixel input), $d_z = 2$ (2D latent). Input · $x = [1.0, 0.5, 0.2, 0.8]$.

DERIVATION

Step 1 · encoder. Suppose it outputs $\mu = [0.4, -0.3]$, $\log \sigma^2 = [-1.4, -2.0] \rightarrow \sigma = [0.5, 0.37]$.

Step 2 · sample $\epsilon = [0.6, -0.2]$ (one draw from $\mathcal{N}(0, I)$).
$z = \mu + \sigma \odot \epsilon = [0.4 + 0.5 \cdot 0.6, -0.3 + 0.37 \cdot (-0.2)] = [0.70, -0.37]$

Step 3 · decode. Suppose decoder outputs $\hat{x} = [0.92, 0.45, 0.25, 0.81]$.

Step 4 · loss.

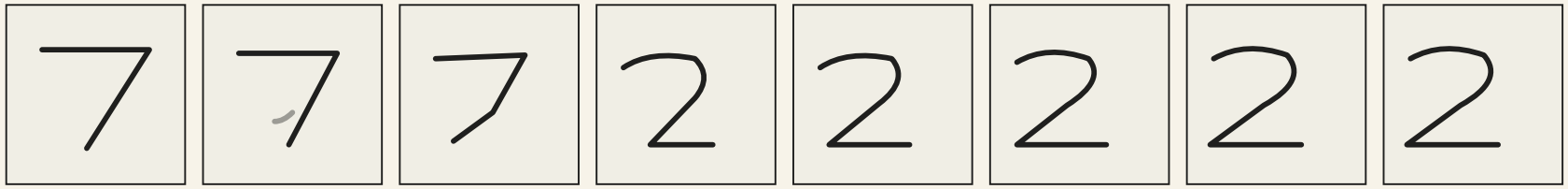
- Reconstruction MSE · $\|\hat{x} - x\|^2 = 0.008^2 + 0.05^2 + 0.05^2 + 0.01^2 \approx 0.005$
- KL · $\frac{1}{2} \sum(\sigma^2 + \mu^2 - 1 - \log \sigma^2)$
  - z_1: $0.25 + 0.16 - 1 + 1.4 = 0.81$
  - z_2: $0.137 + 0.09 - 1 + 2.0 = 1.23$
  - sum / 2 = **1.02**
- Total loss = 0.005 + 1.02 = **1.025**

Backprop through this. Update params. Repeat.

Latent-space interpolation · in pictures

Latent-space interpolation · morph one digit into another by walking z linearly

INTERPOLATE A = 0 → 1 · Z = (1−A)·Z_7 + A·Z_2



α = 0 "7" | α = 1/7 | α = 2/7 | α = 3/7 | α = 4/7 | α = 5/7 | α = 6/7 | α = 1 "2"

LATENT SPACE (2D PROJECTION)

smooth, continuous latent space · every α gives a valid decoded digit

Sampling + interpolation

```
# Generate new examples
with torch.no_grad():
    z = torch.randn(16, d_z)              # sample from N(0, I)
    samples = model.dec(z)                # decode into image space

# Interpolate between two inputs in latent space
z_a = encoder(x_a)[0]  # mu for image A
z_b = encoder(x_b)[0]  # mu for image B
for alpha in torch.linspace(0, 1, 10):
    z = (1 - alpha) * z_a + alpha * z_b
    morph = model.dec(z)                  # smooth transition
```

The interpolation is the magic · it produces *valid* intermediate images because the latent space is smooth.

Sampling gotchas

WATCH OUT

Truncated sampling. Sampling $z \sim \mathcal{N}(0, I)$ occasionally gives large-$\|z\|$ points where training coverage was sparse. Truncate · sample z from $\mathcal{N}(0, I)$ and **reject** if $\|z\| > \tau$ (e.g., $\tau = 2.5$). Samples look cleaner.

Decoder stochasticity. If decoder outputs a Gaussian $p(x|z) = \mathcal{N}(g(z), \sigma_x^2 I)$, add $\sigma_x \cdot \epsilon_x$ to the mean for a single sample. If you only decode means you get the "mode"; adding variance makes samples diverse.

VAE blur. The KL pulls posteriors toward a simple prior · posteriors overlap significantly. The decoder averages over possible z given x → samples are blurry means. This is the fundamental VAE limitation diffusion (L21) fixes.

VAE vs GAN vs Diffusion · quality ranking

	SAMPLE QUALITY	TRAINING STABILITY	LIKELIHOOD	SAMPLING SPEED
VAE	✗ often blurry	✓ stable	✓ ELBO	✓✓ one pass
GAN	✓✓ sharp	✗ brittle	✗ no	✓✓ one pass
Diffusion	✓✓✓ SOTA	✓ stable	≈	✗ many passes

INTUITION

VAEs remain useful for **latent-space exploration** and **pre-compression** — Stable Diffusion uses a VAE to compress images into a 4× smaller latent space *before* running diffusion there.

Common questions · FAQ

Q. Why is VAE blurrier than GAN?

A. VAE's loss is MSE on pixels, which is the mean of possible reconstructions. When multiple outputs are possible (e.g., any detailed face), the mean is a smoothed average of those — blurry. GANs don't average; they commit.

Q. Can I use a perceptual loss (feature-space MSE) instead of pixel MSE?

A. Yes — produces sharper reconstructions. VQ-VAE (Van den Oord 2017) combines VAE-like structure with discrete latents and perceptual losses. Stable Diffusion's VAE uses this trick.

Q. Is the posterior truly Gaussian?

A. No — the *true* posterior is arbitrary. The Gaussian parameterization is an **approximation** (the "amortized variational" part). Normalizing flow encoders and hierarchical VAEs address this; vanilla VAE trades approximation quality for simplicity.

Summary · Lecture 19 — summary

- **Autoencoder** · encode → bottleneck → decode. Great for compression; *not* generative.
- **VAE** · encoder outputs a *distribution* (μ, σ); sample; decode; ELBO loss.
- **ELBO** · reconstruction term + KL divergence to prior.
- **KL term** · closed form for Gaussian; pulls $q(z|x)$ toward $N(0, I)$.
- **Reparameterization trick** · $z = \mu + \sigma \cdot \varepsilon$; differentiable.
- **β-VAE** · tune the KL weight for disentanglement.
- **2026 role** · pre-compressor in latent diffusion models (next week).

Read before Lecture 20

Prince Ch 15 · GANs.

Next lecture

GANs — minimax training, DCGAN, mode collapse, non-saturating loss.

NOTEBOOK

Notebook 19 · `19-vae-mnist.ipynb` — build and train a VAE on MNIST; visualize 2D latent; interpolate digits; sample from N(0, I).