Autoencoders and Variational Autoencoders

From compressing inputs to sampling a distribution

Prof. Nipun Batra

ES 667 · Deep Learning · IIT Gandhinagar

Why reconstruction is not generation

The generative-modeling question

Q QUESTION

A classifier only learns the conditional $p(y \mid x)$. A **generative** model is more ambitious:

The generative-modeling question

Q QUESTION

A classifier only learns the conditional $p(y \mid x)$. A **generative** model is more ambitious:

learn $p_\theta(x)$ well enough to generate plausible new x

The generative-modeling question

Q QUESTION

A classifier only learns the conditional $p(y \mid x)$. A **generative** model is more ambitious:

learn $p_\theta(x)$ well enough to generate plausible new x

Discriminative — $p(y \mid x)$: given an image, name it. Never has to imagine an image.

Generative — $p_\theta(x)$: what does a **plausible image even look like**? Must model the data itself.

The generative-modeling question

Q QUESTION

A classifier only learns the conditional $p(y \mid x)$. A **generative** model is more ambitious:

learn $p_\theta(x)$ well enough to generate plausible new x

Discriminative — $p(y \mid x)$: given an image, name it. Never has to imagine an image.

Generative — $p_\theta(x)$: what does a **plausible image even look like**? Must model the data itself.

today: build a generative model by **learning a latent space we can sample from**

The autoencoder

V VISUAL

Compress the input to a code z, then reconstruct it:

The autoencoder

V VISUAL

Compress the input to a code z, then reconstruct it:

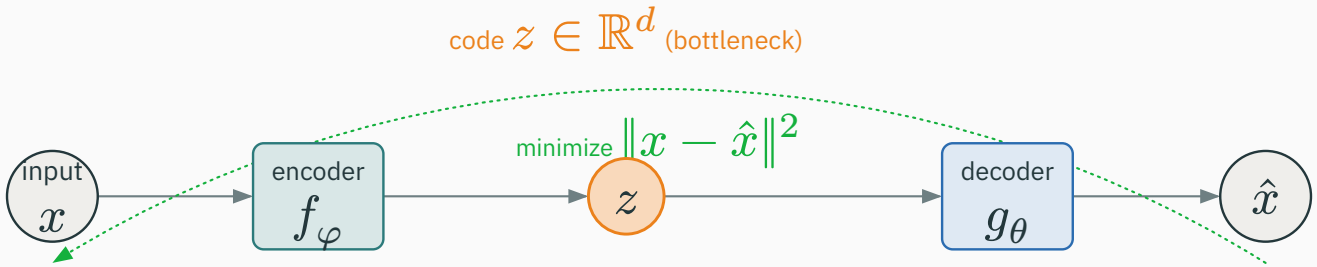
$$x \xrightarrow{f_\varphi} z \xrightarrow{g_\theta} \hat{x}$$

The autoencoder

V VISUAL

Compress the input to a code z, then reconstruct it:

$$x \xrightarrow{f_\varphi} z \xrightarrow{g_\theta} \hat{x}$$

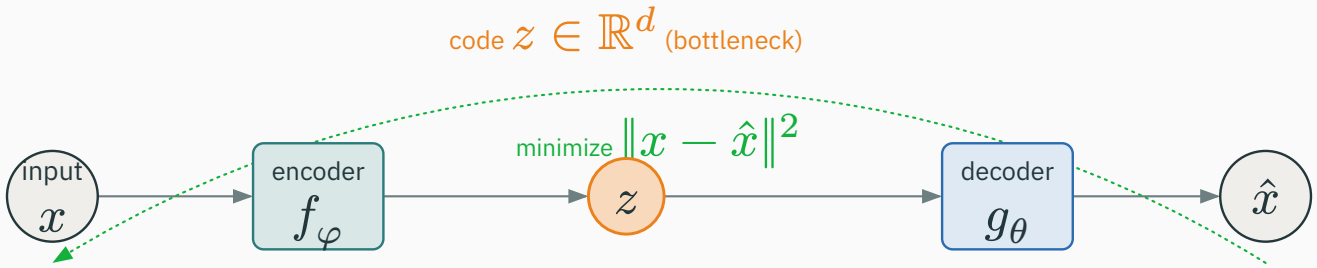


The autoencoder

V VISUAL

Compress the input to a code z, then reconstruct it:

$$x \xrightarrow{f_\varphi} z \xrightarrow{g_\theta} \hat{x}$$



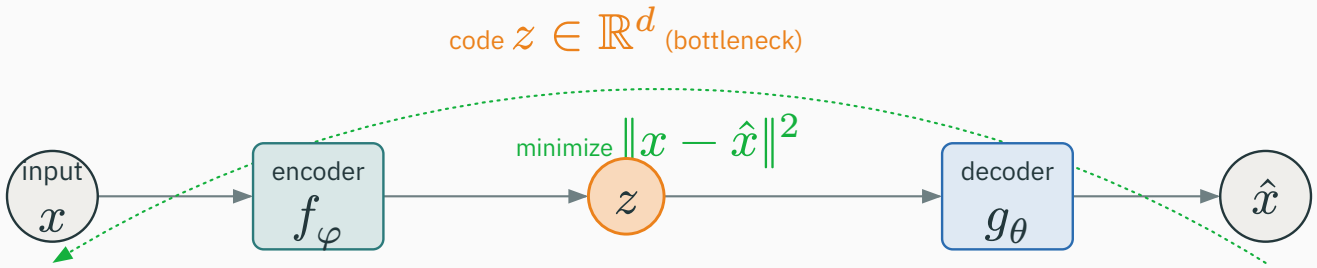
encoder f_φ maps $x \to z$; **decoder** g_θ maps $z \to \hat{x}$; train to make $\hat{x} \approx x$

The autoencoder

V VISUAL

Compress the input to a code z, then reconstruct it:

$$x \xrightarrow{f_\varphi} z \xrightarrow{g_\theta} \hat{x}$$



encoder f_φ maps $x \to z$; **decoder** g_θ maps $z \to \hat{x}$; train to make $\hat{x} \approx x$

(VAE convention: θ is the **generative / decoder** side, φ the encoder — the reverse of the classifier's $f_\theta \to g_\varphi$ in L18)

The bottleneck intuition

D DERIVATION

Why does a narrow code z force **learning** rather than **copying**?

The bottleneck intuition

D DERIVATION

Why does a narrow code z force **learning** rather than **copying**?

If z is constrained (few dimensions), the network cannot store every input coordinate. To reconstruct well it must keep the **salient structure** and discard the rest.

The bottleneck intuition

D DERIVATION

Why does a narrow code z force **learning** rather than **copying**?

If z is constrained (few dimensions), the network cannot store every input coordinate. To reconstruct well it must keep the **salient structure** and discard the rest.

a wide-open code learns the **identity** map and learns nothing useful

The reconstruction objective

D DERIVATION

Train encoder and decoder together to minimize squared reconstruction error:

The reconstruction objective

D DERIVATION

Train encoder and decoder together to minimize squared reconstruction error:

$$\mathcal{L}_{\text{AE}} = \sum_i \left\| x_i - g_\theta\big(f_\varphi(x_i)\big) \right\|_2^2$$

The reconstruction objective

D DERIVATION

Train encoder and decoder together to minimize squared reconstruction error:

$$\mathcal{L}_{\text{AE}} = \sum_i \left\| x_i - g_\theta\big(f_\varphi(x_i)\big) \right\|_2^2$$

This is not an arbitrary choice. A **Gaussian-output** decoder gives

$$p_\theta(x \mid z) = \mathcal{N}(g_\theta(z), \sigma_x^2 I) \quad \Rightarrow \quad -\log p_\theta(x \mid z) \propto \|x - g_\theta(z)\|^2$$

The reconstruction objective

D DERIVATION

Train encoder and decoder together to minimize squared reconstruction error:

$$\mathcal{L}_{\text{AE}} = \sum_i \left\| x_i - g_\theta\big(f_\varphi(x_i)\big) \right\|_2^2$$

This is not an arbitrary choice. A **Gaussian-output** decoder gives

$$p_\theta(x \mid z) = \mathcal{N}(g_\theta(z), \sigma_x^2 I) \quad \Rightarrow \quad -\log p_\theta(x \mid z) \propto \|x - g_\theta(z)\|^2$$

squared error = negative Gaussian log-likelihood — a first taste of the probabilistic view

What an autoencoder does well

D DERIVATION

Even without a probabilistic story, a trained autoencoder is useful:

What an autoencoder does well

D DERIVATION

Even without a probabilistic story, a trained autoencoder is useful:

- **denoising** — corrupt the input, reconstruct the clean signal;
- **compression** — a compact code z for storage or transmission;
- **visualization** — a 2-D or 3-D code to plot the data;
- **features** — reuse z (or the encoder) for a downstream task.

What an autoencoder does well

D DERIVATION

Even without a probabilistic story, a trained autoencoder is useful:

- **denoising** — corrupt the input, reconstruct the clean signal;
- **compression** — a compact code z for storage or transmission;
- **visualization** — a 2-D or 3-D code to plot the data;
- **features** — reuse z (or the encoder) for a downstream task.

all of these use z as a **representation** — none of them require **sampling** new x

Denoising autoencoder

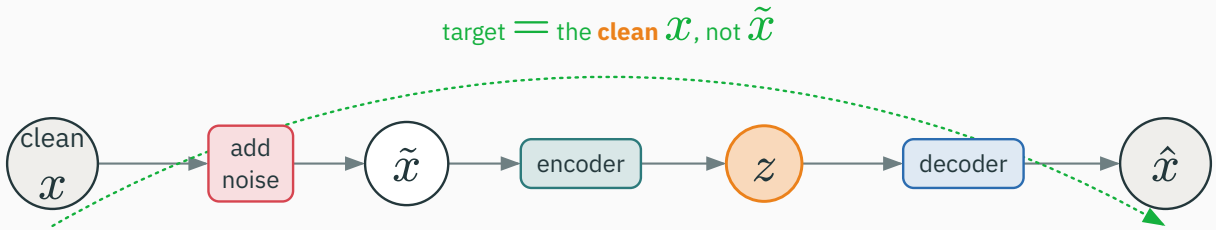
V VISUAL

Force the code to capture structure by making the task harder — corrupt, then restore:

Denoising autoencoder

V VISUAL

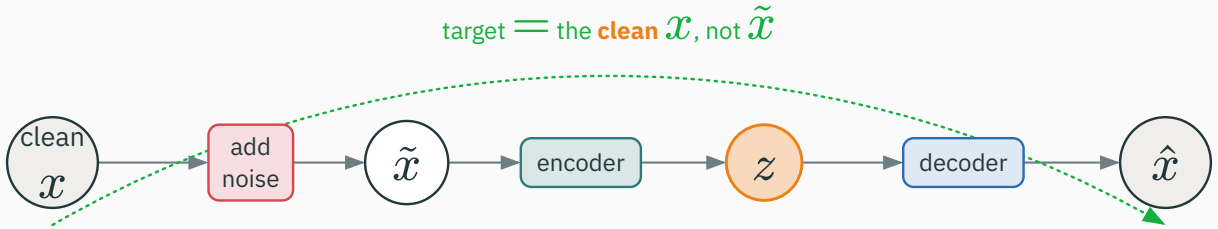
Force the code to capture structure by making the task harder — corrupt, then restore:



Denoising autoencoder

V VISUAL

Force the code to capture structure by making the task harder — corrupt, then restore:



the target is the **clean** x; the encoder must learn what survives noise, not memorize pixels

The sampling failure

D DERIVATION

So can we just feed a random z to the decoder and get a new image?

The sampling failure

D DERIVATION

So can we just feed a random z to the decoder and get a new image?

No. An autoencoder never constrains **where** codes live. Encoded training points occupy **disconnected, irregular** regions. A random z lands **between** them and decodes to nonsense.

The sampling failure

D DERIVATION

So can we just feed a random z to the decoder and get a new image?

No. An autoencoder never constrains **where** codes live. Encoded training points occupy **disconnected, irregular** regions. A random z lands **between** them and decodes to nonsense.

reconstruction says nothing about the **empty** parts of latent space

Visual: latent-space islands

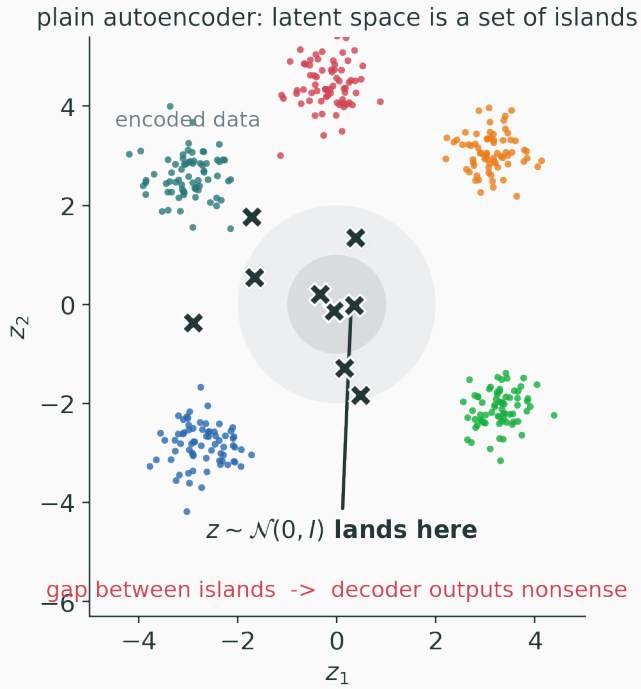
V VISUAL

Encode the data — the codes form islands with wide empty gaps:

Visual: latent-space islands

V VISUAL

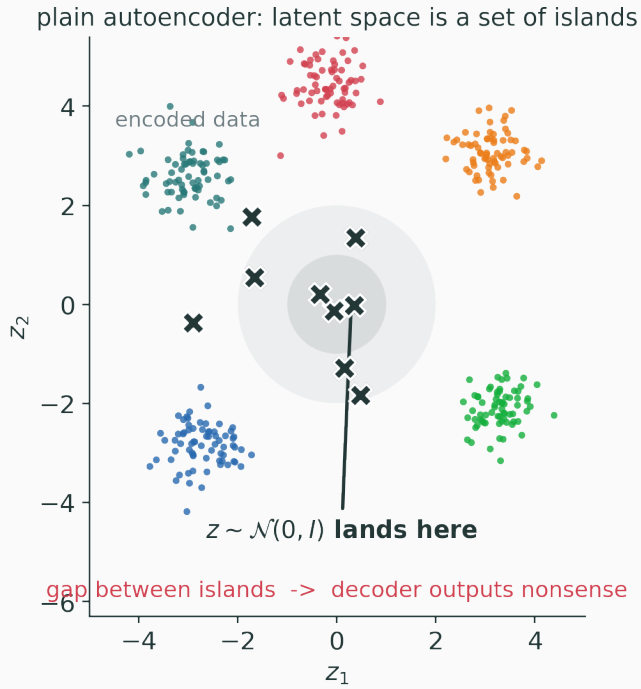
Encode the data — the codes form islands with wide empty gaps:



Visual: latent-space islands

V VISUAL

Encode the data — the codes form islands with wide empty gaps:



samples from $\mathcal{N}(0, I)$ mostly land in the **gap between islands** $\longrightarrow$ the decoder was never trained there

The design requirement

Q QUESTION

For a latent space to be **generative**, sampling the prior must produce plausible data:

The design requirement

Q QUESTION

For a latent space to be **generative**, sampling the prior must produce plausible data:

$$z \sim p(z) \quad \Rightarrow \quad g_\theta(z) \textbf{ should be plausible}$$

The design requirement

Q QUESTION

For a latent space to be **generative**, sampling the prior must produce plausible data:

$$z \sim p(z) \quad \Rightarrow \quad g_\theta(z) \textbf{ should be plausible}$$

We need two things at once: (1) a **known** distribution $p(z)$ we can sample, and (2) a decoder trained so that **every** likely z maps to something realistic. The VAE engineers exactly this.

Interactive: the latent space

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/vae-latent-explorer

Move a point around a 2-D latent space and watch the decoded output. See the islands of a plain autoencoder, and how prior samples fall into the gaps between them.

The VAE probabilistic model

The generative story

D DERIVATION

A VAE first **defines** how data is generated from a simple latent cause:

The generative story

D DERIVATION

A VAE first **defines** how data is generated from a simple latent cause:

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

The generative story

D DERIVATION

A VAE first **defines** how data is generated from a simple latent cause:

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

1. sample a simple latent z from a fixed **standard Gaussian** prior;
2. **render** an observation x with a neural decoder $p_\theta(x \mid z)$.

The generative story

D DERIVATION

A VAE first **defines** how data is generated from a simple latent cause:

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

1. sample a simple latent z from a fixed **standard Gaussian** prior;
2. **render** an observation x with a neural decoder $p_\theta(x \mid z)$.

> the prior is **fixed and sampleable by construction** — that is what fixes the islands problem

A decoder is a conditional distribution

D DERIVATION

The decoder does not output “an image” — it outputs the **parameters of a distribution** over x:

A decoder is a conditional distribution

D DERIVATION

The decoder does not output "an image" — it outputs the **parameters of a distribution** over x:

$$p_\theta(x \mid z) = \mathcal{N}(g_\theta(z), \sigma_x^2 I) \qquad \text{(Gaussian example)}$$

A decoder is a conditional distribution

D DERIVATION

The decoder does not output "an image" — it outputs the **parameters of a distribution** over x:

$$p_\theta(x \mid z) = \mathcal{N}(g_\theta(z), \sigma_x^2 I) \qquad \text{(Gaussian example)}$$

the familiar "reconstruction image" is just the conditional **mean** $g_\theta(z)$

Why inference is hard

D DERIVATION

To train, we would like the **posterior** over the latent cause of each x:

Why inference is hard

D DERIVATION

To train, we would like the **posterior** over the latent cause of each x:

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

Why inference is hard

D DERIVATION

To train, we would like the **posterior** over the latent cause of each x:

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

$$p_\theta(x) = \int p_\theta(x \mid z)\, p(z)\, dz \qquad \text{(intractable integral)}$$

Why inference is hard

D DERIVATION

To train, we would like the **posterior** over the latent cause of each x:

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

$$p_\theta(x) = \int p_\theta(x \mid z)\, p(z)\, dz \qquad \text{(intractable integral)}$$

The evidence $p_\theta(x)$ integrates over **all** z through a nonlinear decoder. No closed form — so the posterior is out of reach too.

Introduce an approximate posterior

D DERIVATION

Sidestep the intractable posterior: let the **encoder** predict a tractable Gaussian approximation.

Introduce an approximate posterior

D DERIVATION

Sidestep the intractable posterior: let the **encoder** predict a tractable Gaussian approximation.

$$q_\varphi(z \mid x) = \mathcal{N}\Big(\mu_\varphi(x), \operatorname{diag}\Big(\sigma_\varphi(x)^2\Big)\Big)$$

Introduce an approximate posterior

D DERIVATION

Sidestep the intractable posterior: let the **encoder** predict a tractable Gaussian approximation.

$$q_\varphi(z \mid x) = \mathcal{N}\Big(\mu_\varphi(x), \operatorname{diag}\Big(\sigma_\varphi(x)^2\Big)\Big)$$

The encoder now outputs a **distribution**, not a single point: a mean $\mu_\varphi(x)$ and a per-dimension variance $\sigma_\varphi(x)^2$. Each input maps to a small cloud in latent space.

The VAE, drawn

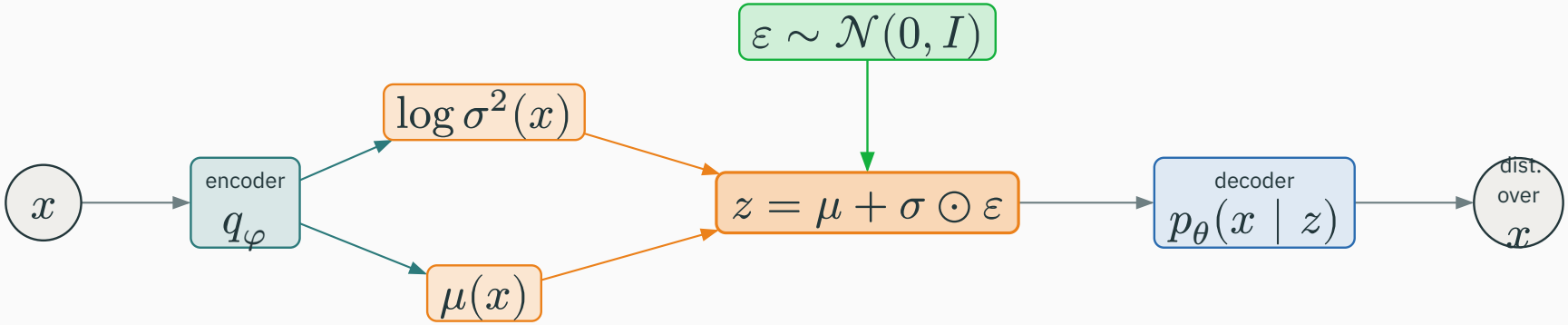
V VISUAL

The whole forward model — encoder predicts a distribution, we sample, the decoder renders:

The VAE, drawn

V VISUAL

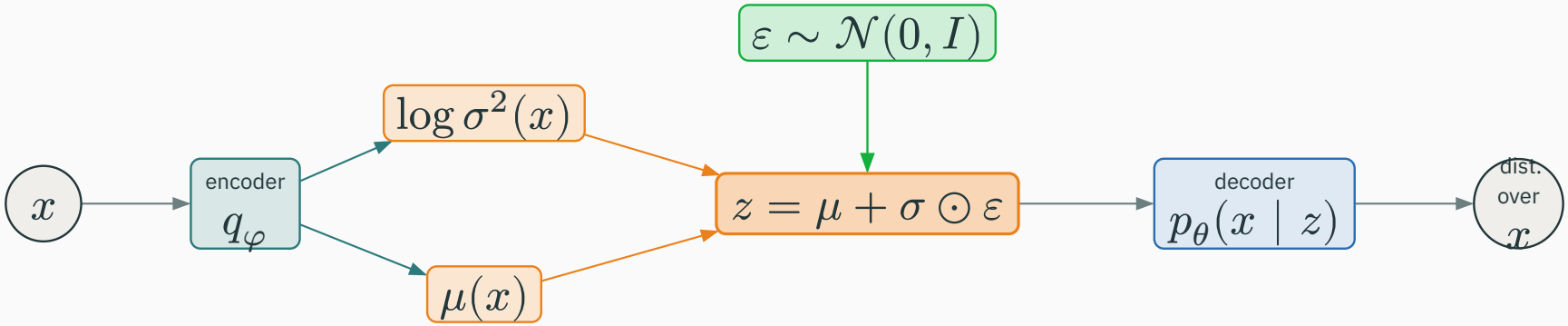
The whole forward model — encoder predicts a distribution, we sample, the decoder renders:



The VAE, drawn

V VISUAL

The whole forward model — encoder predicts a distribution, we sample, the decoder renders:



$x \to$ encoder $\to (\mu, \log \sigma^2) \to$ sample $z \to$ decoder $\to$ distribution over x

The learning target

D DERIVATION

We cannot maximize $\log p_\theta(x)$ directly — but we can maximize a **lower bound** on it:

The learning target

D DERIVATION

We cannot maximize $\log p_\theta(x)$ directly — but we can maximize a **lower bound** on it:

$$\log p_\theta(x) \geq \underbrace{\mathbb{E}_{q_\varphi(z \mid x)}[\log p_\theta(x \mid z)]}_{\text{reconstruct}} - \underbrace{D_{\mathrm{KL}}\big(q_\varphi(z \mid x) \,\|\, p(z)\big)}_{\text{organize latent space}}$$

The learning target

D DERIVATION

We cannot maximize $\log p_\theta(x)$ directly — but we can maximize a **lower bound** on it:

$$\log p_\theta(x) \geq \underbrace{\mathbb{E}_{q_\varphi(z \mid x)}[\log p_\theta(x \mid z)]}_{\text{reconstruct}} - \underbrace{D_{\mathrm{KL}}\big(q_\varphi(z \mid x) \,\|\, p(z)\big)}_{\text{organize latent space}}$$

two terms, two jobs — this is the object we actually optimize

Name it: the ELBO

D DERIVATION

This bound is the **Evidence Lower BOund**:

Name it: the ELBO

D DERIVATION

This bound is the **Evidence Lower BOund**:

maximize a tractable lower bound on the data log-likelihood

Name it: the ELBO

D DERIVATION

This bound is the **Evidence Lower BOund**:

maximize a tractable lower bound on the data log-likelihood

Understand what each term **does** before any algebra: the first term **explains the data**, the second **keeps latent codes near the prior** so we can sample them later.

The reconstruction term

D DERIVATION

$$\mathbb{E}_{q_\varphi(z \mid x)}[\log p_\theta(x \mid z)]$$

The reconstruction term

D DERIVATION

$$\mathbb{E}_{q_\varphi(z \mid x)}[\log p_\theta(x \mid z)]$$

Read it as a recipe:

The reconstruction term

D DERIVATION

$$\mathbb{E}_{q_\varphi(z \mid x)}[\log p_\theta(x \mid z)]$$

Read it as a recipe:

1. take input x, encode it, and **sample** a plausible latent explanation $z \sim q_\varphi(z \mid x)$;
2. require the decoder to **reconstruct** x from that z.

The reconstruction term

D DERIVATION

$$\mathbb{E}_{q_\varphi(z \mid x)}[\log p_\theta(x \mid z)]$$

Read it as a recipe:

1. take input x, encode it, and **sample** a plausible latent explanation $z \sim q_\varphi(z \mid x)$;
2. require the decoder to **reconstruct** x from that z.

high when the sampled latent actually explains the observed input

The KL term

D DERIVATION

$$D_{\mathrm{KL}}\Big(q_\varphi(z \mid x) \parallel p(z)\Big)$$

The KL term

D DERIVATION

$$D_{\mathrm{KL}}\left(q_\varphi(z \mid x) \parallel p(z)\right)$$

Read it as a constraint:

The KL term

D DERIVATION

$$D_{\mathrm{KL}}\left(q_\varphi(z \mid x) \,\|\, p(z)\right)$$

Read it as a constraint:

each example's encoded cloud $q_\varphi(z \mid x)$ should stay **close to the shared prior** $\mathcal{N}(0, I)$. This is what pulls every code toward one common, sampleable region.

The KL term

D DERIVATION

$$D_{\mathrm{KL}}\big(q_\varphi(z \mid x) \,\|\, p(z)\big)$$

Read it as a constraint:

each example's encoded cloud $q_\varphi(z \mid x)$ should stay **close to the shared prior** $\mathcal{N}(0, I)$. This is what pulls every code toward one common, sampleable region.

without it, the encoder would scatter codes into islands again

The Gaussian KL — show, do not derive

D DERIVATION

For a diagonal-Gaussian posterior against the standard-normal prior, the KL is **closed form**:

The Gaussian KL — show, do not derive

D DERIVATION

For a diagonal-Gaussian posterior against the standard-normal prior, the KL is **closed form**:

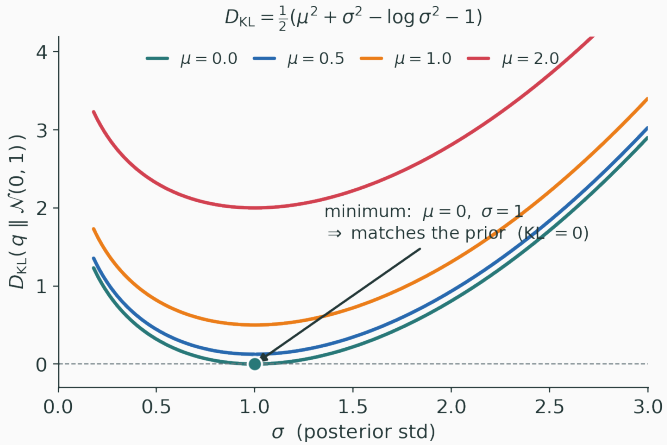
$$D_{\mathrm{KL}}\big(\mathcal{N}(\mu, \sigma^2) \,\|\, \mathcal{N}(0, 1)\big) = \frac{1}{2} \sum_j \big(\mu_j^2 + \sigma_j^2 - \log \sigma_j^2 - 1\big)$$

The Gaussian KL — show, do not derive

D DERIVATION

For a diagonal-Gaussian posterior against the standard-normal prior, the KL is **closed form**:

$$D_{\mathrm{KL}}\big(\mathcal{N}(\mu, \sigma^2) \,\|\, \mathcal{N}(0,1)\big) = \frac{1}{2}\sum_{j}\big(\mu_j^2 + \sigma_j^2 - \log \sigma_j^2 - 1\big)$$

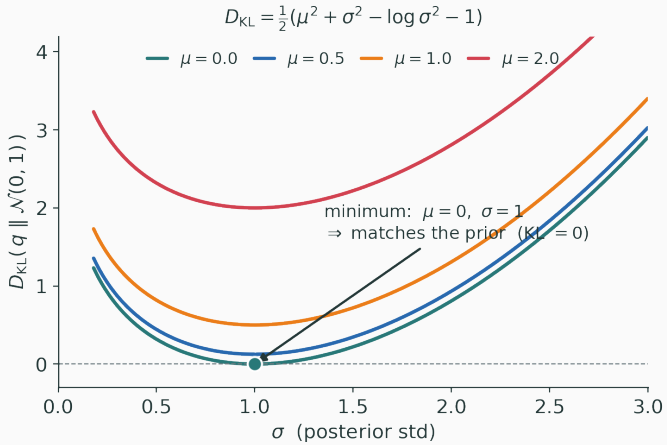


The Gaussian KL — show, do not derive

D DERIVATION

For a diagonal-Gaussian posterior against the standard-normal prior, the KL is **closed form**:

$$D_{\mathrm{KL}}\big(\mathcal{N}(\mu, \sigma^2) \,\|\, \mathcal{N}(0,1)\big) = \frac{1}{2}\sum_j \big(\mu_j^2 + \sigma_j^2 - \log \sigma_j^2 - 1\big)$$



minimized at $\mu = 0, \sigma = 1$ (KL $= 0$) — the KL pushes each code toward the prior

The tension

D DERIVATION

The two terms **pull in opposite directions** — the objective finds the balance:

The tension

D DERIVATION

The two terms **pull in opposite directions** — the objective finds the balance:

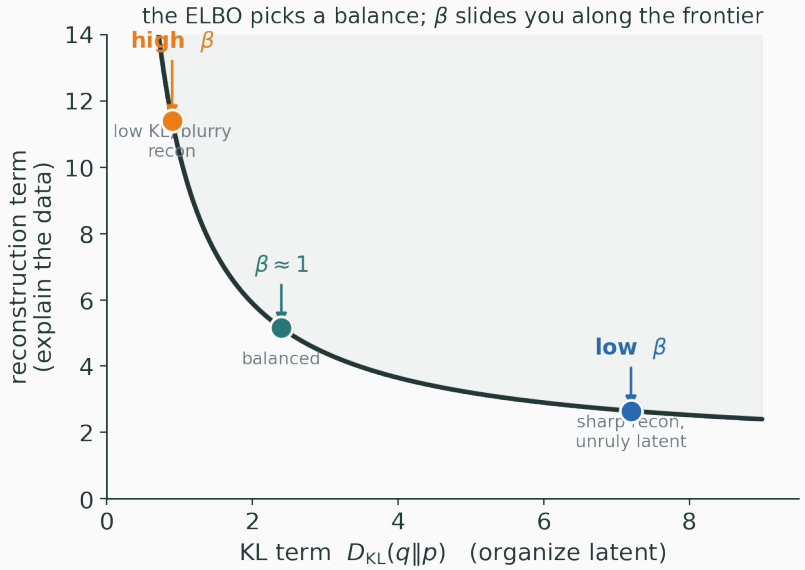
Reconstruction wants every input to keep its **own detailed** code; **KL** wants **all** codes to collapse to **one** Gaussian cloud at the origin.

The tension

D DERIVATION

The two terms **pull in opposite directions** — the objective finds the balance:

Reconstruction wants every input to keep its **own detailed** code; **KL** wants **all** codes to collapse to **one** Gaussian cloud at the origin.



Backpropagation through sampling

The apparent obstacle

D DERIVATION

The ELBO needs a **sample** $z \sim \mathcal{N}(\mu, \sigma^2)$ inside the forward pass. But:

The apparent obstacle

D DERIVATION

The ELBO needs a **sample** $z \sim \mathcal{N}(\mu, \sigma^2)$ inside the forward pass. But:

Sampling is not differentiable. A raw draw $z \sim \mathcal{N}(\mu, \sigma^2)$ seems to block gradients from flowing back into μ and σ — the encoder cannot learn.

The apparent obstacle

D DERIVATION

The ELBO needs a **sample** $z \sim \mathcal{N}(\mu, \sigma^2)$ inside the forward pass. But:

Sampling is not differentiable. A raw draw $z \sim \mathcal{N}(\mu, \sigma^2)$ seems to block gradients from flowing back into μ and σ — the encoder cannot learn.

we need the randomness **without** breaking the chain rule

The reparameterization trick

D DERIVATION

Move the randomness to an **external** variable, then transform it deterministically:

The reparameterization trick

D DERIVATION

Move the randomness to an **external** variable, then transform it deterministically:

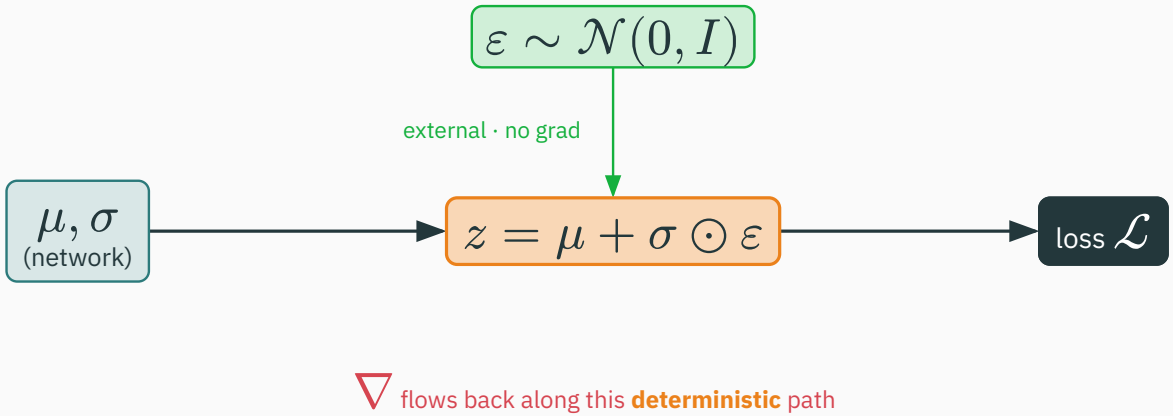
$$\varepsilon \sim \mathcal{N}(0, I), \qquad z = \mu + \sigma \odot \varepsilon$$

The reparameterization trick

D DERIVATION

Move the randomness to an **external** variable, then transform it deterministically:

$$\varepsilon \sim \mathcal{N}(0, I), \qquad z = \mu + \sigma \odot \varepsilon$$

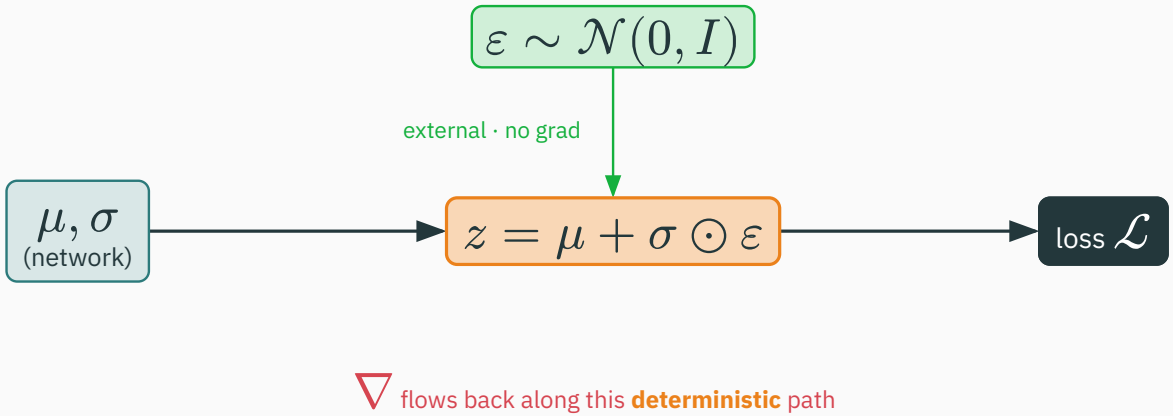


The reparameterization trick

D DERIVATION

Move the randomness to an **external** variable, then transform it deterministically:

$$\varepsilon \sim \mathcal{N}(0, I), \qquad z = \mu + \sigma \odot \varepsilon$$



ε is sampled **outside** the network; the map $\mu, \sigma \longrightarrow z$ is a smooth, differentiable function

Board calculation

D DERIVATION

Take a 2-D latent with $\mu = [1, -1]$, $\sigma = [0.5, 0.2]$, and a draw $\varepsilon = [-0.4, 1.5]$:

Board calculation

D DERIVATION

Take a 2-D latent with $\mu = [1, -1]$, $\sigma = [0.5, 0.2]$, and a draw $\varepsilon = [-0.4, 1.5]$:

$$z = \mu + \sigma \odot \varepsilon = \begin{pmatrix} 1 + 0.5 \cdot (-0.4) \\ -1 + 0.2 \cdot 1.5 \end{pmatrix} = \begin{pmatrix} \mathbf{0.8} \\ \mathbf{-0.7} \end{pmatrix}$$

Board calculation

D DERIVATION

Take a 2-D latent with $\mu = [1, -1]$, $\sigma = [0.5, 0.2]$, and a draw $\varepsilon = [-0.4, 1.5]$:

$$z = \mu + \sigma \odot \varepsilon = \begin{pmatrix} 1 + 0.5 \cdot (-0.4) \\ -1 + 0.2 \cdot 1.5 \end{pmatrix} = \begin{pmatrix} \mathbf{0.8} \\ \mathbf{-0.7} \end{pmatrix}$$

grow $\sigma \longrightarrow$ the same ε moves z **further** from μ: more spread, a larger KL

The training loop

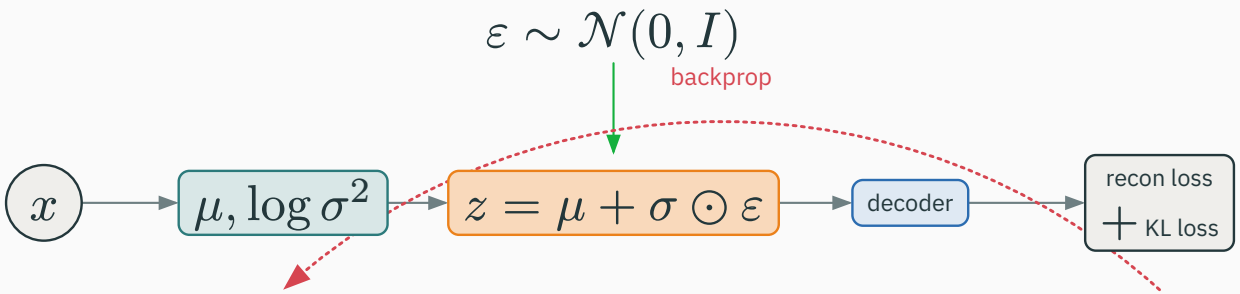
V VISUAL

Every step: encode, reparameterize, decode, score both terms, and backprop.

The training loop

V VISUAL

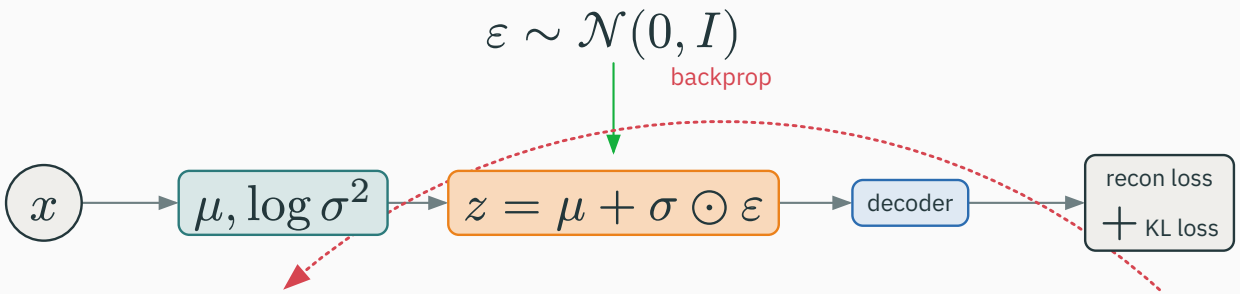
Every step: encode, reparameterize, decode, score both terms, and backprop.



The training loop

V VISUAL

Every step: encode, reparameterize, decode, score both terms, and backprop.



data **enters** the loop; gradients flow through z into the encoder **and** the decoder

The generation loop

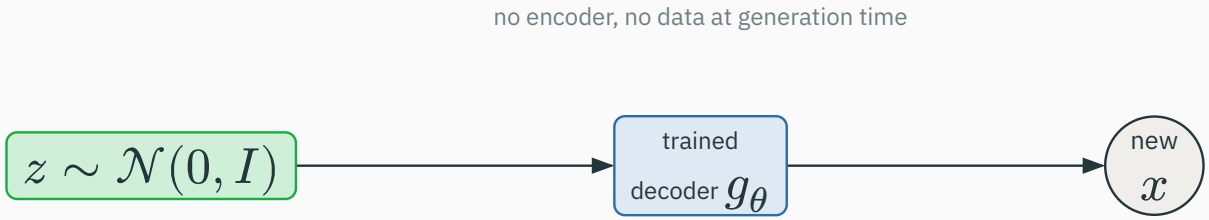
V VISUAL

Generation throws away the encoder — just sample the prior and decode:

The generation loop

V VISUAL

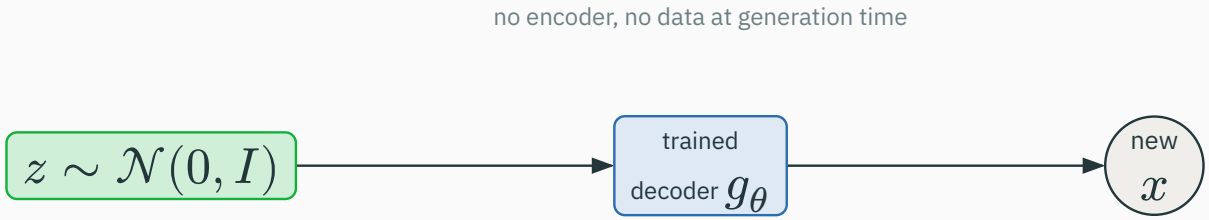
Generation throws away the encoder — just sample the prior and decode:



The generation loop

V VISUAL

Generation throws away the encoder — just sample the prior and decode:



training begins with data; **generation** begins with $z \sim \mathcal{N}(0, I)$ — nothing else

Interpolation as a diagnostic

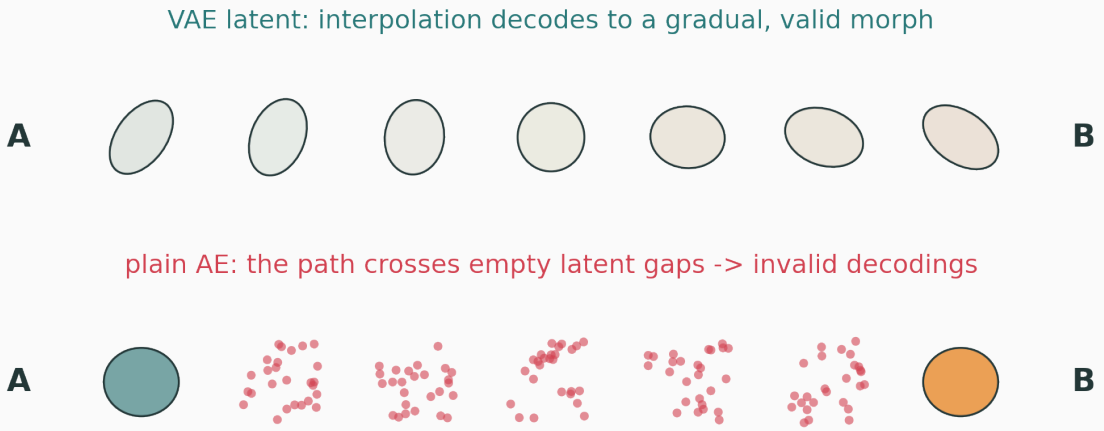
V VISUAL

Walk a straight line between two codes and decode along the path:

Interpolation as a diagnostic

V VISUAL

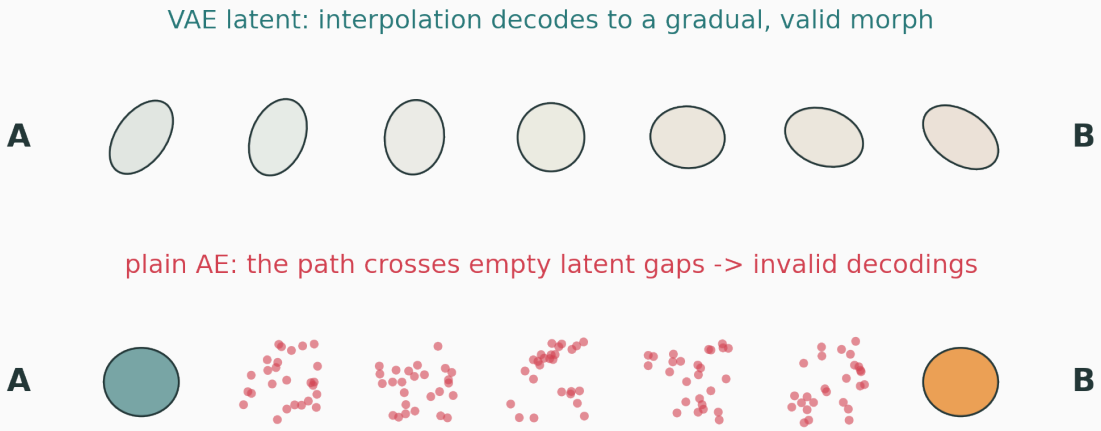
Walk a straight line between two codes and decode along the path:



Interpolation as a diagnostic

V VISUAL

Walk a straight line between two codes and decode along the path:



a good latent space yields **gradual semantic change**; a bad one crosses gaps into invalid images

Conditional VAE

D DERIVATION

Feed a **condition** c (a class label, a text embedding) into both networks:

Conditional VAE

D DERIVATION

Feed a **condition** c (a class label, a text embedding) into both networks:

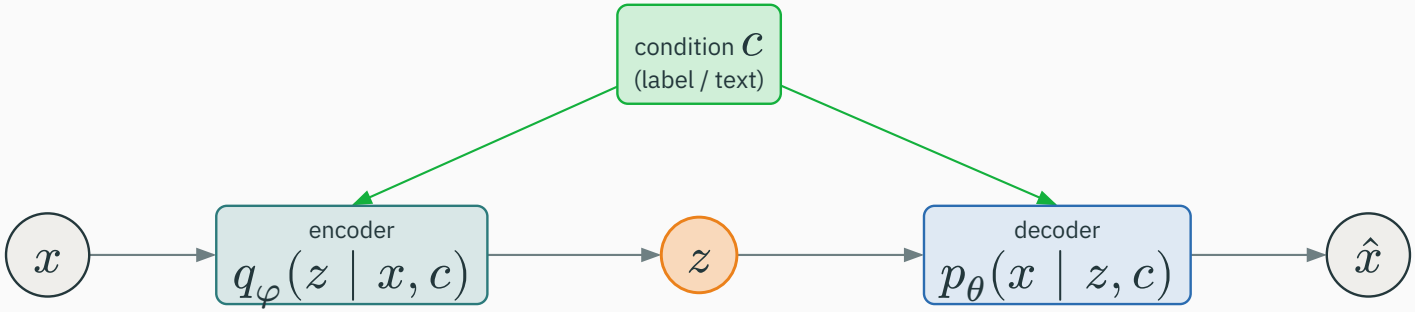
$$q_\varphi(z \mid x, c), \qquad p_\theta(x \mid z, c)$$

Conditional VAE

D DERIVATION

Feed a **condition** c (a class label, a text embedding) into both networks:

$$q_\varphi(z \mid x, c), \qquad p_\theta(x \mid z, c)$$

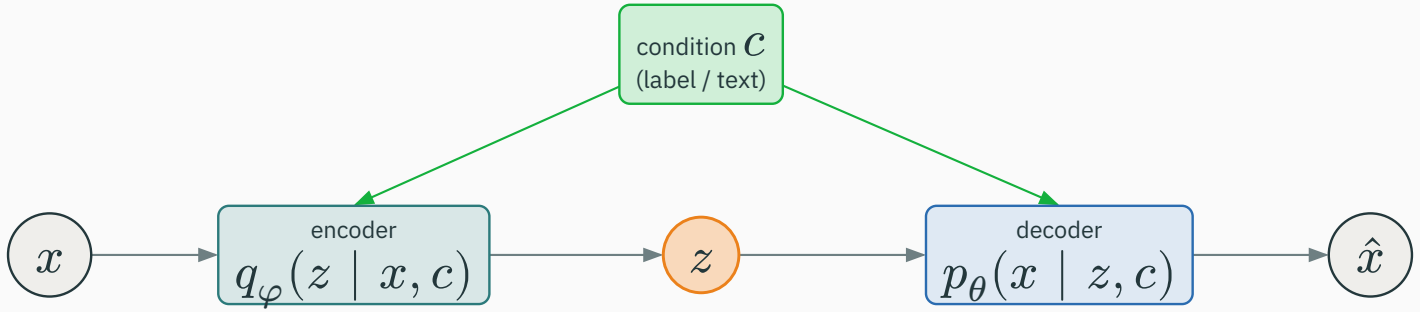


Conditional VAE

D DERIVATION

Feed a **condition** c (a class label, a text embedding) into both networks:

$$q_\varphi(z \mid x, c), \qquad p_\theta(x \mid z, c)$$



at generation time: **choose** c, sample $z \sim p(z)$, decode — c steers **what** is generated

The β-VAE trade-off

D DERIVATION

Put a **weight** β on the KL term and dial the balance explicitly:

The β-VAE trade-off

D DERIVATION

Put a **weight** β on the KL term and dial the balance explicitly:

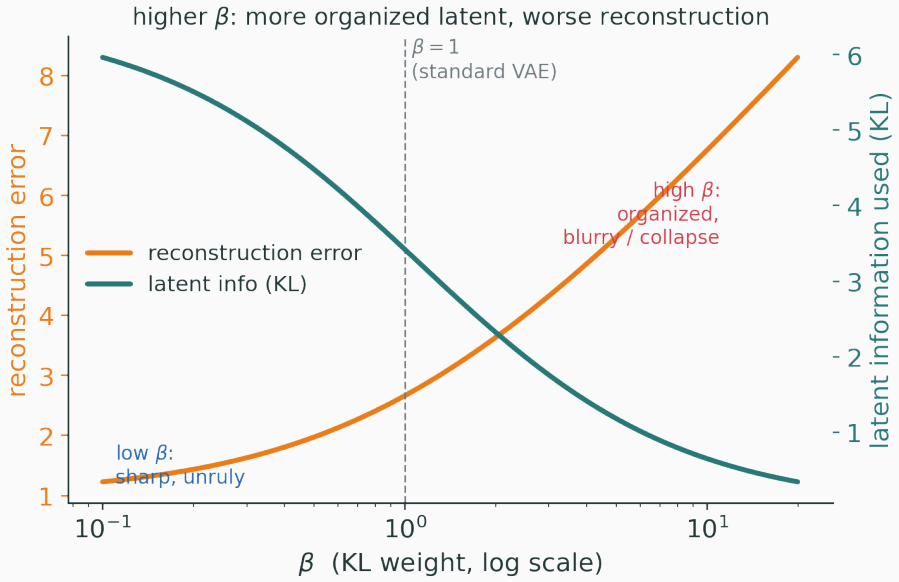
$$\mathcal{L} = -\mathbb{E}_{q[\log p_\theta(x \mid z)]} + \beta\, D_{\mathrm{KL}}\Big(q_\varphi(z \mid x) \,\|\, p(z)\Big)$$

The β-VAE trade-off

D DERIVATION

Put a **weight** β on the KL term and dial the balance explicitly:

$$\mathcal{L} = -\mathbb{E}_{q[\log p_\theta(x \mid z)]} + \beta\, D_{\mathrm{KL}}\Big(q_\varphi(z \mid x) \,\|\, p(z)\Big)$$

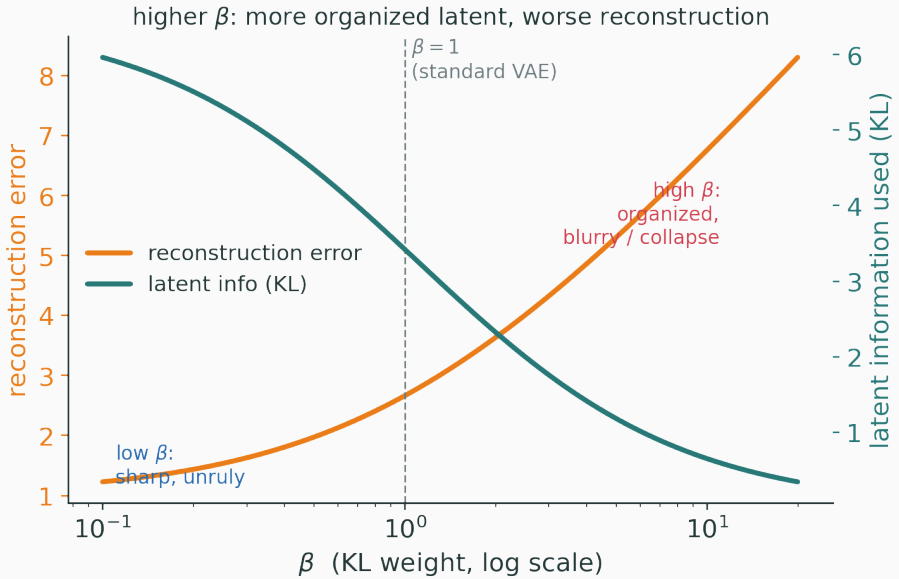


The β-VAE trade-off

D DERIVATION

Put a **weight** β on the KL term and dial the balance explicitly:

$$\mathcal{L} = -\mathbb{E}_{q[\log p_\theta(x \mid z)]} + \beta\, D_{\mathrm{KL}}\Big(q_\varphi(z \mid x) \,\|\, p(z)\Big)$$



higher $\beta \longrightarrow$ more organized latent factors, but **softer / worse** reconstructions

Honest limitations and the bridge onward

Blurry samples: why?

D DERIVATION

A recurring complaint: VAE samples look **soft**. The cause is the per-pixel likelihood.

Blurry samples: why?

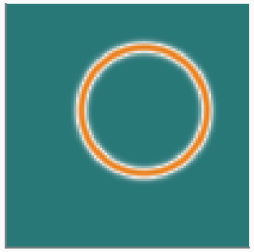
D DERIVATION

A recurring complaint: VAE samples look **soft**. The cause is the per-pixel likelihood.

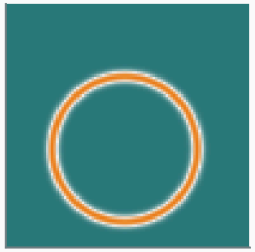
each sample is a plausible sharp output; a per-pixel likelihood rewards their average

sharp sample 1

sharp sample 2



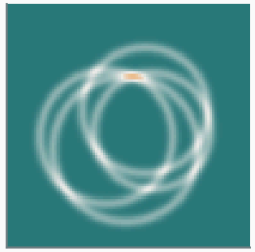
sharp sample 3



+ ⋯

average

pixelwise mean = blurry



Blurry samples: why?

D DERIVATION

A recurring complaint: VAE samples look **soft**. The cause is the per-pixel likelihood.

each sample is a plausible sharp output; a per-pixel likelihood rewards their average

several sharp outputs are all plausible; a per-pixel Gaussian rewards their **average**, which is blurry

Posterior collapse

D DERIVATION

The opposite failure — the model **ignores** the latent entirely:

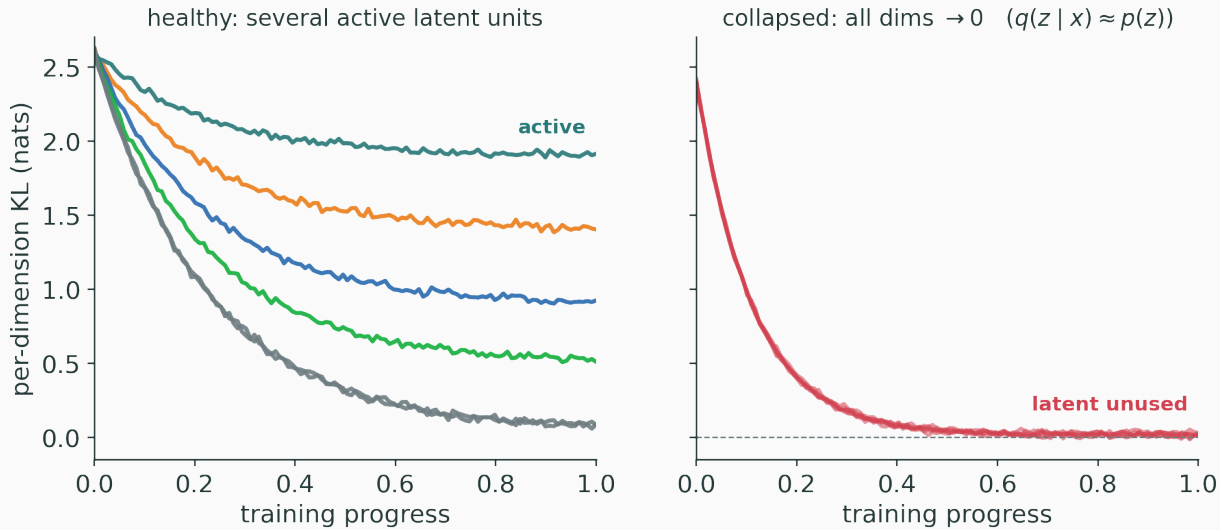
Posterior collapse

D DERIVATION

Posterior collapse

D DERIVATION

The opposite failure — the model **ignores** the latent entirely:



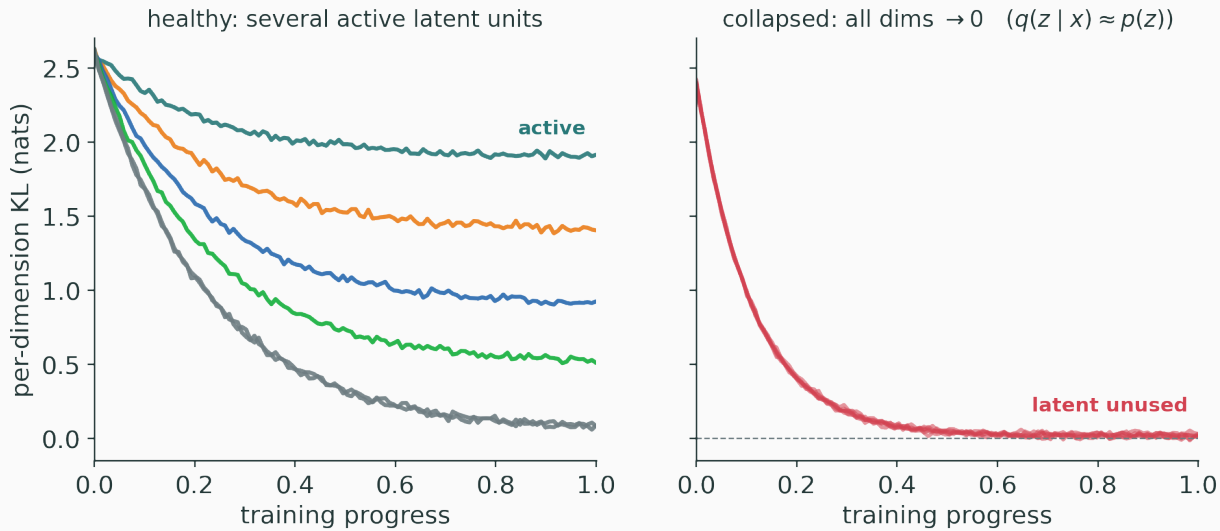
Posterior collapse

D DERIVATION

Posterior collapse

D DERIVATION

The opposite failure — the model **ignores** the latent entirely:



Two routes to the same failure: an over-powerful **autoregressive** decoder can predict x from its own context alone, and — more often for a plain feed-forward

Posterior collapse

D DERIVATION

decoder like ours — the KL term simply dominates. Either way $q_\varphi(z \mid x) \to p(z)$ for every x and the latent goes **unused** — not magically regularized.

VAE versus deterministic autoencoder

V VISUAL

	Autoencoder	VAE
code	one point	a distribution
sampling	unreliable (islands)	defined by the prior
objective	reconstruction	likelihood lower bound
typical output	useful features	smooth, often softer samples

VAE versus deterministic autoencoder

V VISUAL

	Autoencoder	**VAE**
code	one point	a distribution
sampling	unreliable (islands)	defined by the prior
objective	reconstruction	likelihood lower bound
typical output	useful features	smooth, often softer samples

the KL term is the single change that turns a point-code into a **sampleable** latent space

VAE versus GAN — a preview

D DERIVATION

Both use a latent z, but their **learning signals** differ radically:

VAE versus GAN — a preview

D DERIVATION

Both use a latent z, but their **learning signals** differ radically:

VAE — an **explicit** probabilistic story; maximize a likelihood bound (reconstruction + KL).

GAN — **no** explicit likelihood; a generator learns by trying to **fool a discriminator**.

VAE versus GAN — a preview

D DERIVATION

Both use a latent z, but their **learning signals** differ radically:

VAE — an **explicit** probabilistic story; maximize a likelihood bound (reconstruction + KL).

GAN — **no** explicit likelihood; a generator learns by trying to **fool a discriminator**.

VAEs optimize a bound on $p(x)$; GANs play a game — sharper samples, no density

Connection to diffusion

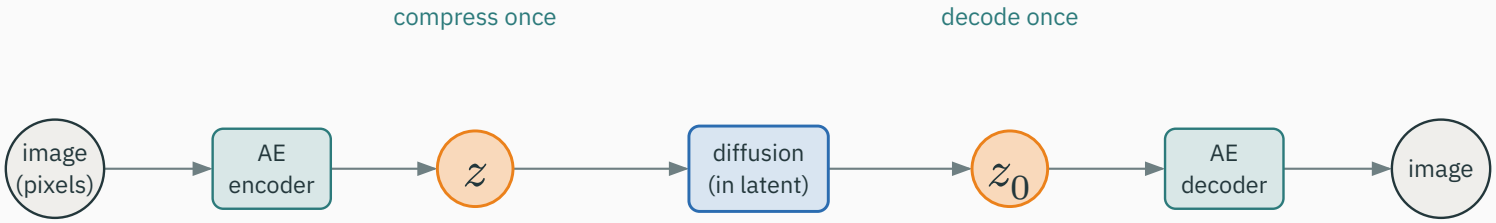
V VISUAL

Modern **latent diffusion** leans directly on the autoencoder:

Connection to diffusion

V VISUAL

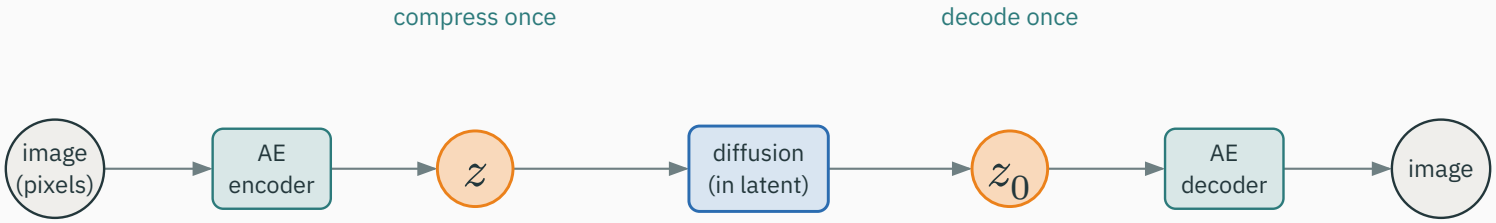
Modern **latent diffusion** leans directly on the autoencoder:



Connection to diffusion

V VISUAL

Modern **latent diffusion** leans directly on the autoencoder:



compress pixels to a compact latent, run diffusion **there**, decode once — the AE is an active component

Exit ticket

Q QUESTION

On a small VAE diagram, label the three steps and predict a failure:

Exit ticket

Q QUESTION

On a small VAE diagram, label the three steps and predict a failure:

Label — (1) the **reconstruction** step, (2) the **KL** step, (3) the **sampling** step.

Predict — what happens to the latent space if we **remove the KL term** entirely?

Exit ticket

Q QUESTION

On a small VAE diagram, label the three steps and predict a failure:

Label — (1) the **reconstruction** step, (2) the **KL** step, (3) the **sampling** step.

Predict — what happens to the latent space if we **remove the KL term** entirely?

no KL ⟶ the codes drift back into islands ⟶ prior samples decode to nonsense again

Closing line

Q QUESTION

VAEs make latent spaces sampleable by regularizing them

Closing line

Q QUESTION

VAEs make latent spaces sampleable by regularizing them

Next: **GANs** trade the explicit likelihood for an **adversarial game**

— giving up density estimation to chase **sharper** samples

Deeper: latent-variable models

Autoencoders are nonlinear compression

D DERIVATION

PCA finds the best **linear** low-dimensional reconstruction:

Autoencoders are nonlinear compression

D DERIVATION

PCA finds the best **linear** low-dimensional reconstruction:

$$x \longrightarrow W^\top x \longrightarrow WW^\top x$$

Autoencoders are nonlinear compression

D DERIVATION

PCA finds the best **linear** low-dimensional reconstruction:

$$x \longrightarrow W^\top x \longrightarrow WW^\top x$$

An autoencoder replaces the two linear maps with **neural networks**, so it can bend around **nonlinear** manifolds.

Autoencoders are nonlinear compression

D DERIVATION

PCA finds the best **linear** low-dimensional reconstruction:

$$x \longrightarrow W^\top x \longrightarrow W W^\top x$$

An autoencoder replaces the two linear maps with **neural networks**, so it can bend around **nonlinear** manifolds.

an intuition link, **not** a claim that every autoencoder is PCA — the nonlinearity is the point

Shape bookkeeping

D DERIVATION

Track the dimensions as data flows through the encoder:

Shape bookkeeping

D DERIVATION

Track the dimensions as data flows through the encoder:

$$x \in \mathbb{R}^{H \times W \times C} \quad \xrightarrow{\text{encoder}} \quad z \in \mathbb{R}^{d}$$

Shape bookkeeping

D DERIVATION

Track the dimensions as data flows through the encoder:

$$x \in \mathbb{R}^{H \times W \times C} \quad \overset{\text{encoder}}{\longrightarrow} \quad z \in \mathbb{R}^{d}$$

A concrete example — a 32×32 grayscale image compressed to a 16-dimensional code:

$$x \in \mathbb{R}^{1{,}024} \quad \longrightarrow \quad z \in \mathbb{R}^{16} \qquad (\text{a } 64\times \text{ squeeze})$$

Shape bookkeeping

D DERIVATION

Track the dimensions as data flows through the encoder:

$$x \in \mathbb{R}^{H \times W \times C} \quad \overset{\text{encoder}}{\longrightarrow} \quad z \in \mathbb{R}^{d}$$

A concrete example — a 32×32 grayscale image compressed to a 16-dimensional code:

$$x \in \mathbb{R}^{1{,}024} \quad \longrightarrow \quad z \in \mathbb{R}^{16} \qquad (\text{a } 64\times \text{ squeeze})$$

what must the 16 numbers preserve so the decoder can rebuild all 1,024 pixels?

Undercomplete and overcomplete codes

V VISUAL

Code dimension	Pressure on the model
$d < \dim(x)$ (undercomplete)	must compress salient structure
$d \geq \dim(x)$ (overcomplete)	may learn an identity shortcut
overcomplete + regularization	can still learn sparse / robust features

Undercomplete and overcomplete codes

V VISUAL

Code dimension	**Pressure on the model**
$d < \dim(x)$ (undercomplete)	must compress salient structure
$d \geq \dim(x)$ (overcomplete)	may learn an identity shortcut
overcomplete + regularization	can still learn sparse / robust features

the bottleneck can be **dimensional**, **stochastic**, **architectural**, or **regularization**-based

Reconstruction likelihood, not just MSE

D DERIVATION

Grounding the familiar loss in probability. If the decoder is Gaussian:

Reconstruction likelihood, not just MSE

D DERIVATION

Grounding the familiar loss in probability. If the decoder is Gaussian:

$$p_\theta(x \mid z) = \mathcal{N}(g_\theta(z), \sigma_x^2 I)$$

Reconstruction likelihood, not just MSE

D DERIVATION

Grounding the familiar loss in probability. If the decoder is Gaussian:

$$p_\theta(x \mid z) = \mathcal{N}(g_\theta(z), \sigma_x^2 I)$$

then the negative log-likelihood is **proportional to squared error**:

$$-\log p_\theta(x \mid z) = \frac{1}{2\sigma_x^2}\|x - g_\theta(z)\|^2 + \text{const}$$

Reconstruction likelihood, not just MSE

D DERIVATION

Grounding the familiar loss in probability. If the decoder is Gaussian:

$$p_\theta(x \mid z) = \mathcal{N}(g_\theta(z), \sigma_x^2 I)$$

then the negative log-likelihood is **proportional to squared error**:

$$-\log p_\theta(x \mid z) = \frac{1}{2\sigma_x^2}\|x - g_\theta(z)\|^2 + \text{const}$$

MSE is a **modeling choice** (a Gaussian decoder), not a law of nature

Data type determines the decoder output

V VISUAL

Data	Decoder distribution	Reconstruction term
normalized continuous pixels	Gaussian	squared error
binary image	Bernoulli	binary cross-entropy
tokens	categorical	cross-entropy

Data type determines the decoder output

V VISUAL

Data	Decoder distribution	Reconstruction term
normalized continuous pixels	Gaussian	squared error
binary image	Bernoulli	binary cross-entropy
tokens	categorical	cross-entropy

the decoder predicts **distribution parameters** — match the distribution to the data type

The VAE graphical model

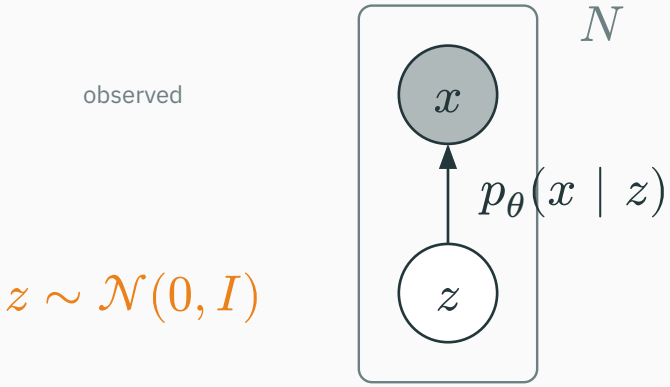
V VISUAL

At generation time the arrow is simple: cause $\rightarrow$ observation.

The VAE graphical model

V VISUAL

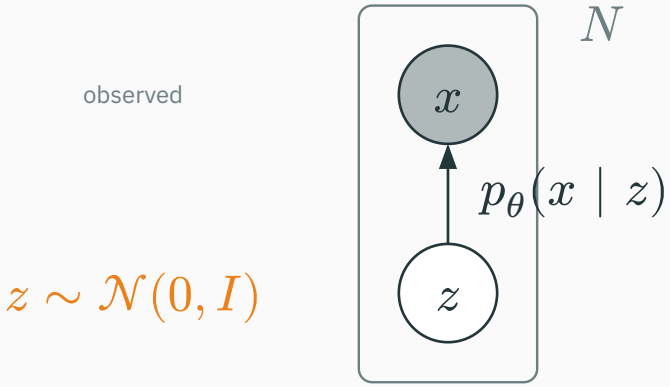
At generation time the arrow is simple: cause $\rightarrow$ observation.



The VAE graphical model

V VISUAL

At generation time the arrow is simple: cause → observation.



training **inverts** it: infer the hidden cause z that plausibly generated each observed x

Marginal likelihood is an integral

D DERIVATION

The quantity we truly want to maximize:

Marginal likelihood is an integral

D DERIVATION

The quantity we truly want to maximize:

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

Marginal likelihood is an integral

D DERIVATION

The quantity we truly want to maximize:

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

For a nonlinear neural decoder and high-dimensional z, this integral has **no closed form** and cannot be evaluated exactly — hence the lower bound.

Variational inference: a tractable stand-in

D DERIVATION

Choose a tractable family $q_\varphi(z \mid x)$ to **imitate** the true posterior $p_\theta(z \mid x)$:

Variational inference: a tractable stand-in

D DERIVATION

Choose a tractable family $q_\varphi(z \mid x)$ to **imitate** the true posterior $p_\theta(z \mid x)$:

Amortized inference. One encoder network φ handles **every** input — instead of solving a fresh optimization for each image, we **amortize** the cost into shared weights.

Variational inference: a tractable stand-in

D DERIVATION

Choose a tractable family $q_\varphi(z \mid x)$ to **imitate** the true posterior $p_\theta(z \mid x)$:

Amortized inference. One encoder network φ handles **every** input — instead of solving a fresh optimization for each image, we **amortize** the cost into shared weights.

classical VI optimizes per-datapoint; the VAE trains one network to predict q for all of them

The exact decomposition

D DERIVATION

The identity behind the bound's **name**:

The exact decomposition

D DERIVATION

The identity behind the bound's **name**:

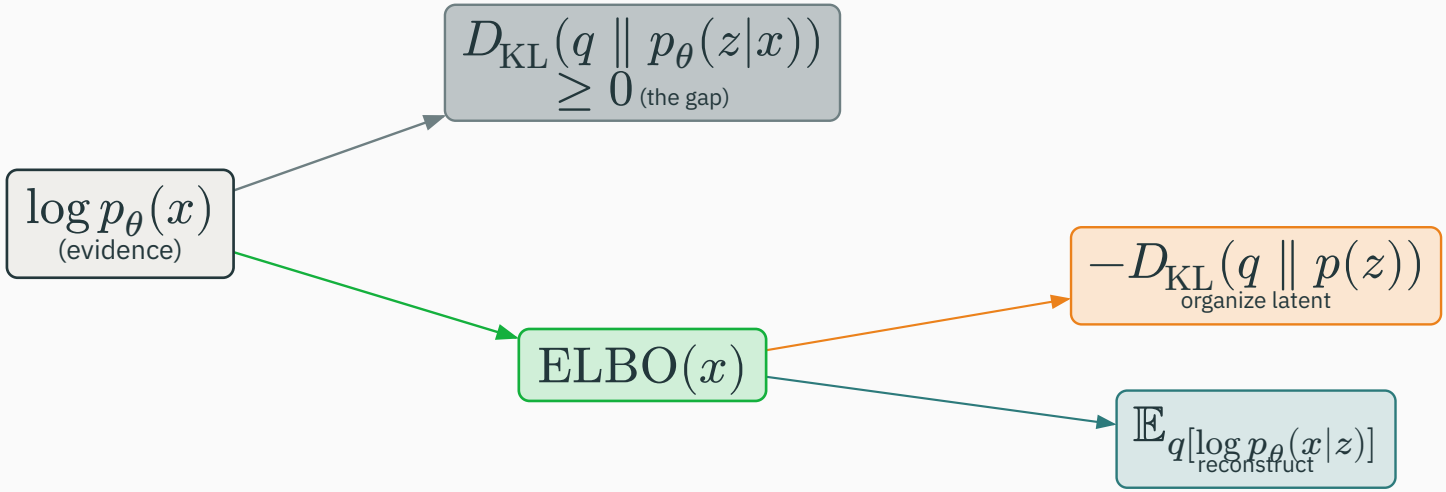
$$\log p_\theta(x) = \mathrm{ELBO}(x) + D_{\mathrm{KL}}\Big(q_\varphi(z \mid x) \,\|\, p_\theta(z \mid x)\Big)$$

The exact decomposition

D DERIVATION

The identity behind the bound's **name**:

$$\log p_\theta(x) = \mathrm{ELBO}(x) + D_{\mathrm{KL}}\Big(q_\varphi(z \mid x) \,\|\, p_\theta(z \mid x)\Big)$$

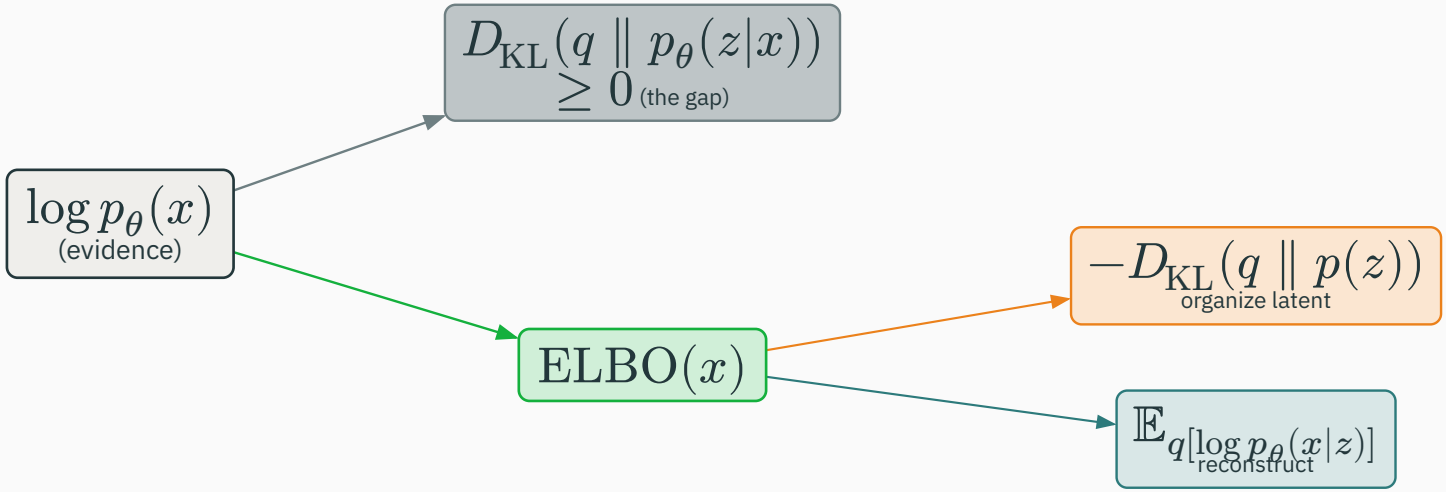


The exact decomposition

D DERIVATION

The identity behind the bound's **name**:

$$\log p_\theta(x) = \mathrm{ELBO}(x) + D_{\mathrm{KL}}\big(q_\varphi(z \mid x) \,\|\, p_\theta(z \mid x)\big)$$



the KL gap is ≥ 0, so ELBO $\leq \log p_\theta(x)$ — a genuine **lower bound**

Expand the ELBO

D DERIVATION

Start from the definition and substitute the joint:

Expand the ELBO

D DERIVATION

Start from the definition and substitute the joint:

$$\mathrm{ELBO}(x) = \mathbb{E}_{q_\varphi(z \mid x)}\left[\log p_\theta(x, z) - \log q_\varphi(z \mid x)\right]$$

Expand the ELBO

D DERIVATION

Start from the definition and substitute the joint:

$$\mathrm{ELBO}(x) = \mathbb{E}_{q_\varphi(z \mid x)}\left[\log p_\theta(x, z) - \log q_\varphi(z \mid x)\right]$$

$$p_\theta(x, z) = p_\theta(x \mid z)\, p(z)$$

Expand the ELBO

D DERIVATION

Start from the definition and substitute the joint:

$$\mathrm{ELBO}(x) = \mathbb{E}_{q_\varphi(z \mid x)}\left[\log p_\theta(x, z) - \log q_\varphi(z \mid x)\right]$$

$$p_\theta(x, z) = p_\theta(x \mid z)\, p(z)$$

substitute the joint, then split the logarithm — the next slide collects the pieces

Rearrange into the two familiar terms

D DERIVATION

Grouping the split logarithm gives back the working objective:

Rearrange into the two familiar terms

D DERIVATION

Grouping the split logarithm gives back the working objective:

$$\mathrm{ELBO} = \underbrace{\mathbb{E}_{q[\log p_\theta(x \mid z)]}}_{\text{explain the data}} - \underbrace{D_{\mathrm{KL}}\big(q_\varphi(z \mid x) \,\|\, p(z)\big)}_{\text{organize the latent}}$$

Rearrange into the two familiar terms

D DERIVATION

Grouping the split logarithm gives back the working objective:

$$\mathrm{ELBO} = \underbrace{\mathbb{E}_{q[\log p_\theta(x \mid z)]}}_{\text{explain the data}} - \underbrace{D_{\mathrm{KL}}\big(q_\varphi(z \mid x) \,\|\, p(z)\big)}_{\text{organize the latent}}$$

same two terms as Part II — now **derived**, not asserted

Which KL is which?

D DERIVATION

Two KL expressions appear — students routinely conflate them:

Which KL is which?

D DERIVATION

Two KL expressions appear — students routinely conflate them:

$D_{\mathrm{KL}}(q(z \mid x) \,\|\, p(z))$ regularizes each code toward the **prior** (in the loss)

Which KL is which?

D DERIVATION

Two KL expressions appear — students routinely conflate them:

$D_{\mathrm{KL}}(q(z \mid x) \,\|\, p(z))$ regularizes each code toward the **prior** (in the loss)

$D_{\mathrm{KL}}(q(z \mid x) \,\|\, p(z \mid x))$ the **gap** between ELBO and true log-likelihood

Which KL is which?

D DERIVATION

Two KL expressions appear — students routinely conflate them:

$D_{\mathrm{KL}}(q(z \mid x) \,\|\, p(z))$ regularizes each code toward the **prior** (in the loss)

$D_{\mathrm{KL}}(q(z \mid x) \,\|\, p(z \mid x))$ the **gap** between ELBO and true log-likelihood

one is a **term you optimize**; the other is **how loose the bound is**

Training loss signs

D DERIVATION

We **maximize** the ELBO, equivalently **minimize** its negative:

Training loss signs

D DERIVATION

We **maximize** the ELBO, equivalently **minimize** its negative:

$$\mathcal{L}_{\text{VAE}} = \underbrace{-\mathbb{E}_{q[\log p_\theta(x \mid z)]}}_{\text{“reconstruction loss“}} + \underbrace{D_{\text{KL}}\left(q_\varphi(z \mid x) \parallel p(z)\right)}_{\text{“KL loss“}}$$

Training loss signs

D DERIVATION

We **maximize** the ELBO, equivalently **minimize** its negative:

$$\mathcal{L}_{\text{VAE}} = \underbrace{-\mathbb{E}_{q}[\log p_\theta(x \mid z)]}_{\text{“reconstruction loss“}} + \underbrace{D_{\text{KL}}\left(q_\varphi(z \mid x) \,\|\, p(z)\right)}_{\text{“KL loss“}}$$

in code the first term is literally the MSE or BCE; the second is the closed-form Gaussian KL

Monte Carlo estimate

D DERIVATION

The expectation over $q_\varphi(z \mid x)$ is approximated by **sampling** latent vectors:

Monte Carlo estimate

D DERIVATION

The expectation over $q_\varphi(z \mid x)$ is approximated by **sampling** latent vectors:

$$\mathbb{E}_{q[\cdot]} \approx \frac{1}{K} \sum_{k=1}^{K} [\cdot]_{z^{(k)}}, \qquad z^{(k)} = \mu + \sigma \odot \varepsilon^{(k)}$$

Monte Carlo estimate

D DERIVATION

The expectation over $q_\varphi(z \mid x)$ is approximated by **sampling** latent vectors:

$$\mathbb{E}_{q[\cdot]} \approx \frac{1}{K} \sum_{k=1}^{K} [\cdot]_{z^{(k)}}, \qquad z^{(k)} = \mu + \sigma \odot \varepsilon^{(k)}$$

one reparameterized sample per input ($K = 1$) is usually enough for the standard objective

Why output log-variance?

D DERIVATION

Networks predict $\ell = \log \sigma^2$ rather than σ directly, then recover:

Why output log-variance?

D DERIVATION

Networks predict $\ell = \log \sigma^2$ rather than σ directly, then recover:

$$\ell = \log \sigma^2 \qquad \longrightarrow \qquad \sigma = \exp\left(\frac{1}{2}\,\ell\right)$$

Why output log-variance?

D DERIVATION

Networks predict $\ell = \log \sigma^2$ rather than σ directly, then recover:

$$\ell = \log \sigma^2 \qquad \longrightarrow \qquad \sigma = \exp\left(\frac{1}{2}\,\ell\right)$$

ℓ ranges over **all** of $\mathbb{R}$ — safe for an unconstrained linear output — while $\exp(\cdot)$ keeps the variance **strictly positive**. No clamping, no $\log 0$.

Why output log-variance?

D DERIVATION

Networks predict $\ell = \log \sigma^2$ rather than σ directly, then recover:

$$\ell = \log \sigma^2 \qquad \longrightarrow \qquad \sigma = \exp\left(\frac{1}{2}\,\ell\right)$$

ℓ ranges over **all** of $\mathbb{R}$ — safe for an unconstrained linear output — while $\exp(\cdot)$ keeps the variance **strictly positive**. No clamping, no $\log 0$.

$$\text{e.g. } \ell = -1.386 \to \sigma = \exp(-0.693) = 0.5$$

Reparameterization computation graph

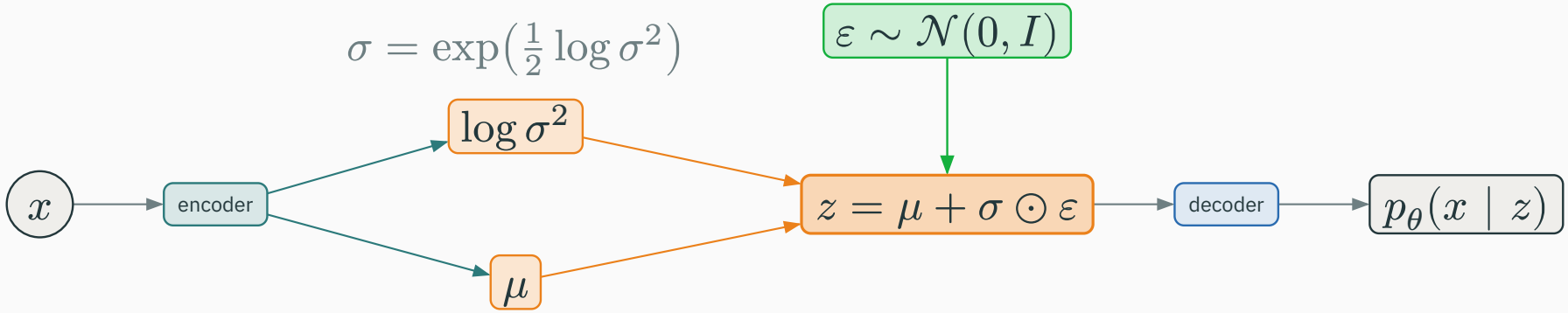
V VISUAL

The full graph with the log-variance parameterization:

Reparameterization computation graph

V VISUAL

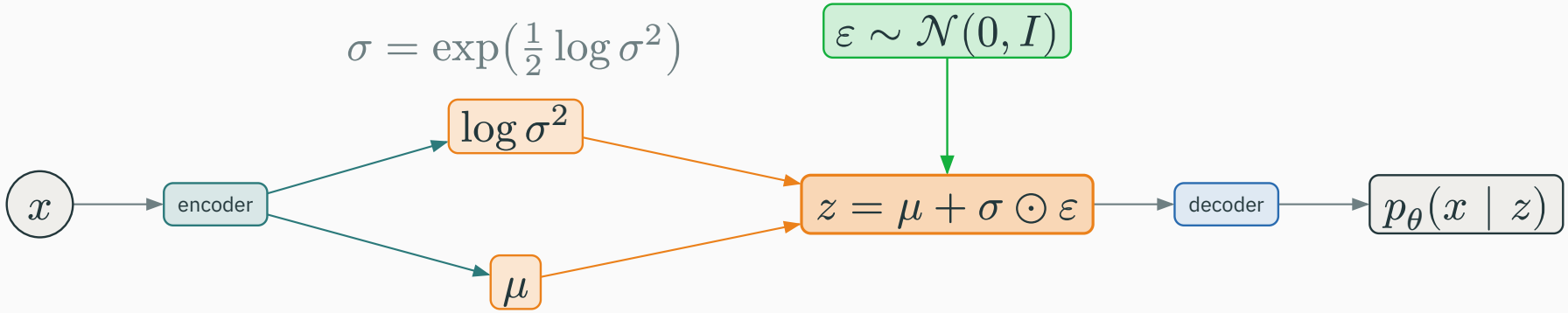
The full graph with the log-variance parameterization:



Reparameterization computation graph

V VISUAL

The full graph with the log-variance parameterization:



ε is drawn independently; gradients flow through the **deterministic** transform $\mu, \sigma \to z$

Worked: one training step's loss

D DERIVATION

Take the same encoded point, $\mu = [0.8, -0.7]$, $\sigma = [0.5, 0.2]$, with $\sigma_x^2 = 1$ and $\|x - \hat{x}\|^2 = 3.0$:

Worked: one training step's loss

D DERIVATION

Take the same encoded point, $\mu = [0.8, -0.7]$, $\sigma = [0.5, 0.2]$, with $\sigma_x^2 = 1$ and $\|x - \hat{x}\|^2 = 3.0$:

$$\mathrm{KL} = \frac{1}{2}[(0.64 + 0.25 + 1.386 - 1) + (0.49 + 0.04 + 3.219 - 1)] = \mathbf{2.01}$$

Worked: one training step's loss

D DERIVATION

Take the same encoded point, $\mu = [0.8, -0.7]$, $\sigma = [0.5, 0.2]$, with $\sigma_x^2 = 1$ and $\|x - \hat{x}\|^2 = 3.0$:

$$\mathrm{KL} = \frac{1}{2}[(0.64 + 0.25 + 1.386 - 1) + (0.49 + 0.04 + 3.219 - 1)] = \mathbf{2.01}$$

$$\mathrm{recon} = \frac{1}{2}\|x - \hat{x}\|^2 = 1.50, \qquad \mathcal{L}_{\mathrm{VAE}} = 1.50 + 2.01 = \mathbf{3.51}$$

Worked: one training step's loss

D DERIVATION

Take the same encoded point, $\mu = [0.8, -0.7]$, $\sigma = [0.5, 0.2]$, with $\sigma_x^2 = 1$ and $\|x - \hat{x}\|^2 = 3.0$:

$$\mathrm{KL} = \frac{1}{2}[(0.64 + 0.25 + 1.386 - 1) + (0.49 + 0.04 + 3.219 - 1)] = \mathbf{2.01}$$

$$\mathrm{recon} = \frac{1}{2}\|x - \hat{x}\|^2 = 1.50, \qquad \mathcal{L}_{\mathrm{VAE}} = 1.50 + 2.01 = \mathbf{3.51}$$

both terms are in **nats**; the optimizer trades one against the other

Sampling versus reconstruction

V VISUAL

Operation	Latent source	Question answered
reconstruction	$z \sim q(z \mid x)$	can the model explain this input?
unconditional sample	$z \sim p(z)$	what does the model generate?
interpolation	path between two codes	is the latent geometry smooth?

Sampling versus reconstruction

V VISUAL

Operation	Latent source	Question answered
reconstruction	$z \sim q(z \mid x)$	can the model explain this input?
unconditional sample	$z \sim p(z)$	what does the model generate?
interpolation	path between two codes	is the latent geometry smooth?

where z comes from changes the meaning of the decoder's output

A KL trade-off numeric intuition

D DERIVATION

Two candidate encoders for the **same** data:

A KL trade-off numeric intuition

D DERIVATION

Two candidate encoders for the **same** data:

Encoder	Reconstruction	KL
highly individualized codes	1	100
codes near the prior	10	1

A KL trade-off numeric intuition

D DERIVATION

Two candidate encoders for the **same** data:

Encoder	Reconstruction	KL
highly individualized codes	1	100
codes near the prior	10	1

totals 101 vs 11: neither is “obviously best” — the **objective** picks the balance, not reconstruction alone

Visualize a 2-D latent space

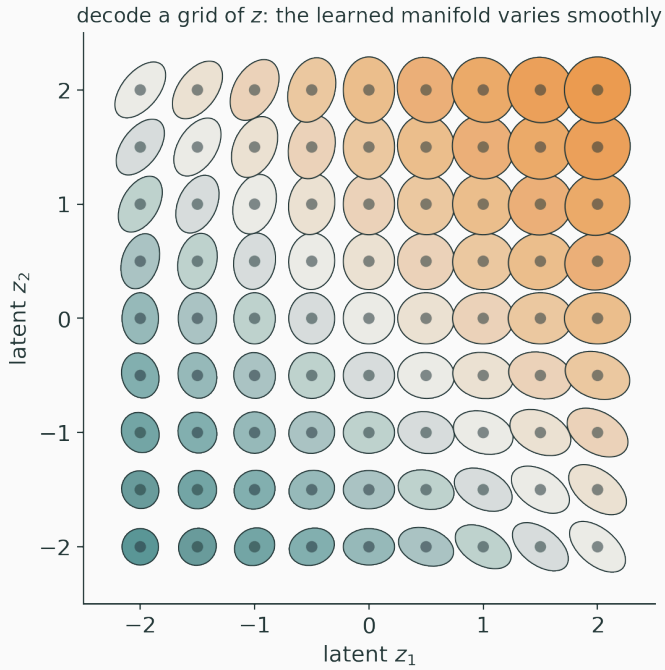
V VISUAL

Encode a dataset into (z_1, z_2) and decode a **grid** of latent points:

Visualize a 2-D latent space

V VISUAL

Encode a dataset into (z_1, z_2) and decode a **grid** of latent points:



Visualize a 2-D latent space

V VISUAL

Encode a dataset into (z_1, z_2) and decode a **grid** of latent points:

a good map shows **clusters**, **smooth directions**, **empty regions**, and where **prior samples** land

Latent traversal

D DERIVATION

Hold every coordinate fixed and move a **single** latent dimension:

Latent traversal

D DERIVATION

Hold every coordinate fixed and move a **single** latent dimension:

$$z_j \leftarrow z_j + \delta, \qquad \text{(decode several values of } \delta\text{)}$$

Latent traversal

D DERIVATION

Hold every coordinate fixed and move a **single** latent dimension:

$$z_j \leftarrow z_j + \delta, \qquad \text{(decode several values of } \delta\text{)}$$

Interpretable changes along z_j are **interesting evidence** about the learned representation — but **not proof** that the factors are perfectly disentangled.

Conditional VAE, expanded

D DERIVATION

For a label or text condition c, condition **both** networks:

Conditional VAE, expanded

D DERIVATION

For a label or text condition c, condition **both** networks:

$$q_\varphi(z \mid x, c), \qquad p_\theta(x \mid z, c)$$

Conditional VAE, expanded

D DERIVATION

For a label or text condition c, condition **both** networks:

$$q_\varphi(z \mid x, c), \qquad p_\theta(x \mid z, c)$$

Controlled by c — the attribute you set (the digit class, the caption).

Still diverse through z — the style, stroke, and variation **within** that attribute.

Conditional VAE, expanded

D DERIVATION

For a label or text condition c, condition **both** networks:

$$q_\varphi(z \mid x, c), \qquad p_\theta(x \mid z, c)$$

Controlled by c — the attribute you set (the digit class, the caption).

Still diverse through z — the style, stroke, and variation **within** that attribute.

sample: choose c, draw $z \sim p(z)$, decode $p_\theta(x \mid z, c)$

β-VAE and disentanglement claims

D DERIVATION

$$\mathcal{L} = -\mathbb{E}_{q[\log p_\theta(x \mid z)]} + \beta\, D_{\mathrm{KL}}(q \parallel p)$$

β-VAE and disentanglement claims

D DERIVATION

$$\mathcal{L} = -\mathbb{E}_{q[\log p_\theta(x \mid z)]} + \beta\, D_{\mathrm{KL}}(q \parallel p)$$

Increasing β strengthens the pressure toward a simple prior.

β-VAE and disentanglement claims

D DERIVATION

$$\mathcal{L} = -\mathbb{E}_{q[\log p_\theta(x \mid z)]} + \beta\, D_{\mathrm{KL}}(q \parallel p)$$

Increasing β strengthens the pressure toward a simple prior.

It **can** encourage more separated factors on curated data — but "disentangled" is **not** a guaranteed, universal outcome, and reconstruction suffers.

Posterior collapse, diagnose it

D DERIVATION

The symptom to watch for:

Posterior collapse, diagnose it

D DERIVATION

The symptom to watch for:

$q_\varphi(z \mid x) \approx p(z)$ for **every** x

Posterior collapse, diagnose it

D DERIVATION

The symptom to watch for:

$q_\varphi(z \mid x) \approx p(z)$ for **every** x

A powerful decoder predicts x well from context alone, so the KL term drives every posterior to the prior. The latent has become **unused** — inspect per-dimension KL, not just reconstruction.

Posterior collapse, high-level responses

D DERIVATION

Design choices, not a new module:

Posterior collapse, high-level responses

D DERIVATION

Design choices, not a new module:

- **weaken or schedule** the KL pressure early in training (KL annealing);
- **constrain** the decoder's capacity so it cannot ignore z;
- **force** more information through the latent (e.g. a free-bits floor);
- **inspect** latent-use statistics, not only reconstructions.

Posterior collapse, high-level responses

D DERIVATION

Design choices, not a new module:

- **weaken or schedule** the KL pressure early in training (KL annealing);
- **constrain** the decoder's capacity so it cannot ignore z;
- **force** more information through the latent (e.g. a free-bits floor);
- **inspect** latent-use statistics, not only reconstructions.

the fixes all **rebalance** how much the model is allowed to rely on the decoder alone

VQ-style discrete latents

D DERIVATION

Some autoencoders encode into a **finite codebook** instead of a continuous Gaussian:

VQ-style discrete latents

D DERIVATION

Some autoencoders encode into a **finite codebook** instead of a continuous Gaussian:

continuous VAE latent versus discrete codebook token

VQ-style discrete latents

D DERIVATION

Some autoencoders encode into a **finite codebook** instead of a continuous Gaussian:

continuous VAE latent versus discrete codebook token

this previews **tokenized** visual representations — the basis of many image + video generators

VQ-style discrete latents

D DERIVATION

Some autoencoders encode into a **finite codebook** instead of a continuous Gaussian:

continuous VAE latent versus discrete codebook token

this previews **tokenized** visual representations — the basis of many image + video generators

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/vq-codebook

A codebook widget: snap continuous encodings to their nearest discrete code vectors.

Why VAEs matter to modern diffusion

V VISUAL

Latent diffusion uses the autoencoder to change **what space** the generator works in:

Why VAEs matter to modern diffusion

V VISUAL

Latent diffusion uses the autoencoder to change **what space** the generator works in:

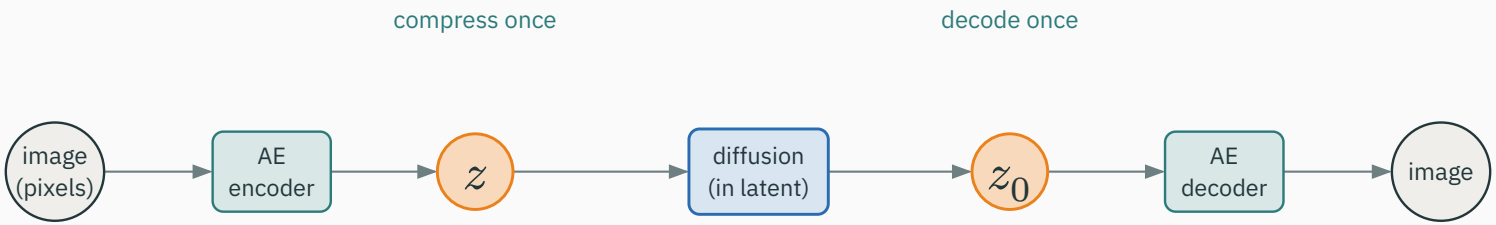
$$\text{pixels} \longrightarrow \text{compact continuous latents}$$

Why VAEs matter to modern diffusion

V VISUAL

Latent diffusion uses the autoencoder to change **what space** the generator works in:

pixels $\longrightarrow$ compact continuous latents

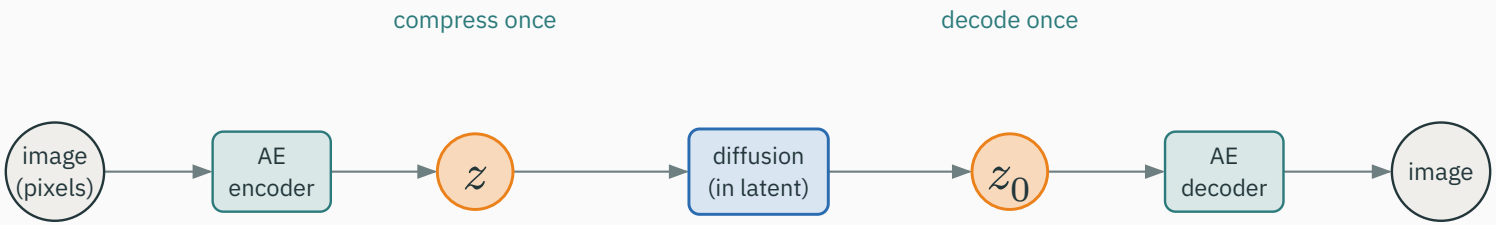


Why VAEs matter to modern diffusion

V VISUAL

Latent diffusion uses the autoencoder to change **what space** the generator works in:

pixels $\longrightarrow$ compact continuous latents



the VAE is not historical background — it is an **active systems component** of latent diffusion

How should generative models be evaluated?

Q QUESTION

No single number suffices — use several lenses:

How should generative models be evaluated?

Q QUESTION

No single number suffices — use several lenses:

- **sample quality and diversity** (are outputs realistic **and** varied?);
- **reconstruction / likelihood** metrics where appropriate;
- **downstream utility** of the learned features;
- **human evaluation** with a transparent protocol;
- **memorization and data-provenance** checks.

How should generative models be evaluated?

Q QUESTION

No single number suffices — use several lenses:

- **sample quality and diversity** (are outputs realistic **and** varied?);
- **reconstruction / likelihood** metrics where appropriate;
- **downstream utility** of the learned features;
- **human evaluation** with a transparent protocol;
- **memorization and data-provenance** checks.

a model can score well on one lens and fail badly on another

Common misconceptions

V VISUAL

Misconception	Correction
“the encoder outputs one latent point”	it outputs a **distribution’s parameters**
“KL makes samples better directly”	it **organizes codes** to match the sampling prior
“blurry samples prove VAEs are wrong”	they reflect likelihood / output trade-offs
“interpolation proves understanding”	it is a **diagnostic**, not proof

Retrieval questions

Q QUESTION

Test yourself before the next lecture:

Retrieval questions

Q QUESTION

Test yourself before the next lecture:

1. Why is an ordinary autoencoder **not guaranteed** to generate from a random z?

Retrieval questions

Q QUESTION

Test yourself before the next lecture:

1. Why is an ordinary autoencoder **not guaranteed** to generate from a random z?
2. What does the KL term **compare** in a VAE?

Retrieval questions

Q QUESTION

Test yourself before the next lecture:

1. Why is an ordinary autoencoder **not guaranteed** to generate from a random z?
2. What does the KL term **compare** in a VAE?
3. Why does $z = \mu + \sigma \odot \varepsilon$ **permit** backpropagation?

Retrieval questions

Q QUESTION

Test yourself before the next lecture:

1. Why is an ordinary autoencoder **not guaranteed** to generate from a random z?
2. What does the KL term **compare** in a VAE?
3. Why does $z = \mu + \sigma \odot \varepsilon$ **permit** backpropagation?
4. What is **posterior collapse**, and how would you **detect** it?

Next lecture

Q QUESTION

VAE: explain data through a latent **likelihood**

↓

GAN: learn realism through an adversarial **critic**

Next lecture

Q QUESTION

VAE: explain data through a latent **likelihood**

↓

GAN: learn realism through an adversarial **critic**

the next model **removes** the explicit reconstruction target and introduces a **game**

A VAE turns a code into a **sampleable latent variable**: reconstruction $+$ a KL-regularized prior $=$ a generative model.

Next: **Generative Adversarial Networks** $\rightarrow$ diffusion $\rightarrow$ modern generative models.