

Variational AutoEncoder

Mohammad Salameh and Pat Virtue

What is a generative model?

A **discriminative** model predicts a label y from an input x , learning $p(y|x)$.

A **generative model** learns the distribution of the data itself, $p(x)$.

With it we can

1. **sample** new data $x \sim p(x)$
2. **evaluate** how likely an input is
3. learn useful **representations** without labels.

We never see $p(x)$. we only see a training set $D = \{x^{(1)}, \dots, x^{(N)}\}$ drawn from it.

60 real handwritten digits from the MNIST test set

4	3	6	4	3	5	2	8	0	7
8	1	8	3	6	6	7	0	5	0
4	9	4	4	9	3	4	1	6	8
6	9	4	8	8	2	3	4	7	3
0	0	2	5	4	8	8	6	1	7
1	1	7	4	6	7	1	2	4	8

What is a generative model?

- The same digit varies in slant, stroke width, size and style (open or closed 4s, crossed or plain 7s).
- A generative model must capture all of this variation and produce new variations, not copies.

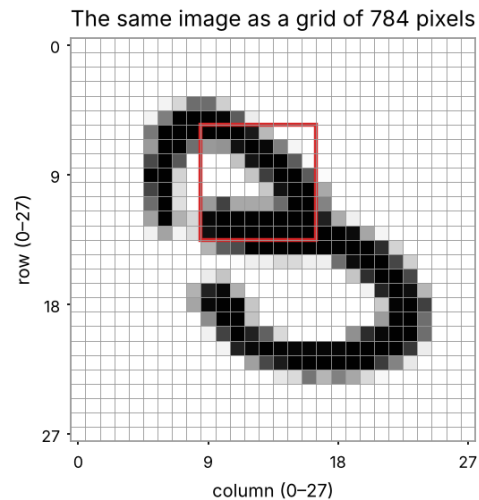
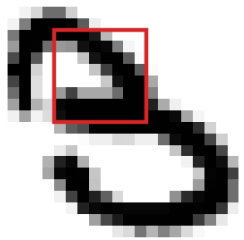
60 real handwritten digits from the MNIST test set

4	3	6	4	3	5	2	8	0	7
8	1	8	3	6	6	7	0	5	0
4	9	4	4	9	3	4	1	6	8
6	9	4	8	8	2	3	4	7	3
0	0	2	5	4	8	8	6	1	7
1	1	7	4	6	7	1	2	4	8

Images are points in a huge space

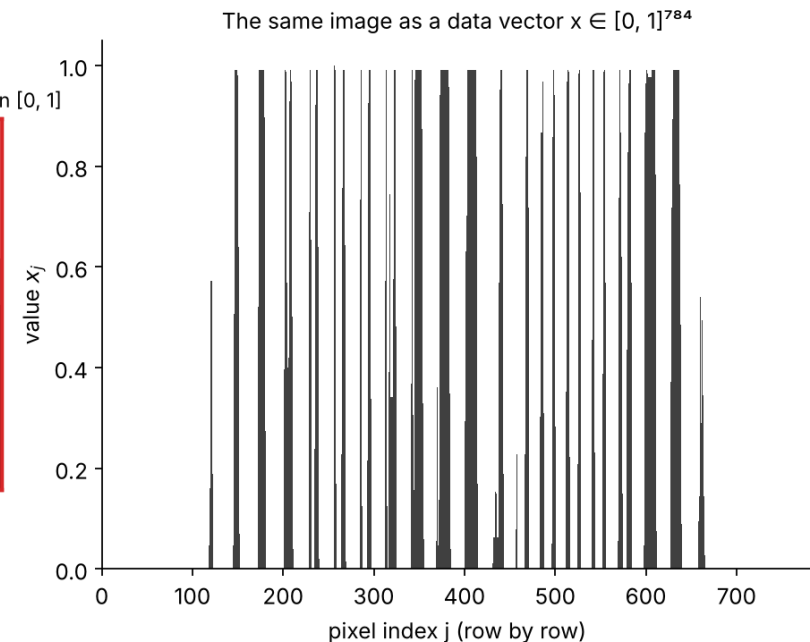
- A 28×28 image is $M = 784$ numbers.
- Writing them in a row turns the image into a vector $x^{(i)} \in [0,1]^M$, one point in an M -dimensional **data space** (or **pixel space**), with pixels $x_j^{(i)}, j = 1, \dots, M$

A 28×28 MNIST image of a 3



Zoom on the red box: each pixel is a number in $[0, 1]$

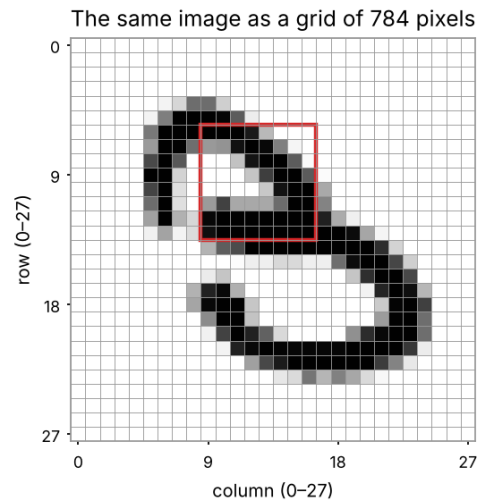
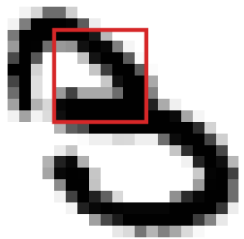
1.0	1.0	1.0	0.9	0.3	0.0	0.0	0.0
0.4	0.4	0.9	1.0	1.0	0.5	0.0	0.0
0.0	0.0	0.2	0.9	1.0	1.0	0.6	0.0
0.0	0.0	0.0	0.2	0.8	1.0	1.0	0.6
0.0	0.0	0.0	0.0	0.2	0.9	1.0	1.0
0.4	0.7	0.3	0.3	0.3	0.6	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0



Images are points in a huge space

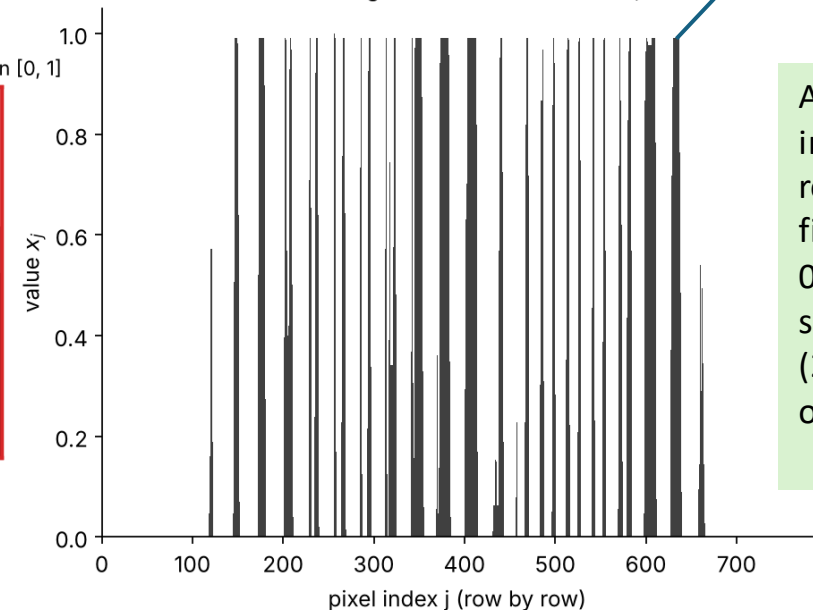
- A 28×28 image is $M = 784$ numbers.
- Writing them in a row turns the image into a vector $x^{(i)} \in [0,1]^M$, one point in an M -dimensional **data space** (or **pixel space**), with pixels $x_j^{(i)}, j = 1, \dots, M$

A 28×28 MNIST image of a 3



Zoom on the red box: each pixel is a number in $[0, 1]$

1.0	1.0	1.0	0.9	0.3	0.0	0.0	0.0
0.4	0.4	0.9	1.0	1.0	0.5	0.0	0.0
0.0	0.0	0.2	0.9	1.0	1.0	0.6	0.0
0.0	0.0	0.0	0.2	0.8	1.0	1.0	0.6
0.0	0.0	0.0	0.0	0.2	0.9	1.0	1.0
0.4	0.7	0.3	0.3	0.3	0.6	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0



Each cluster of tall bars is the ink in one row of the grid

All 784 numbers in one row, top row of pixels first (positions 0–27), then the second row (28–55), and so on.

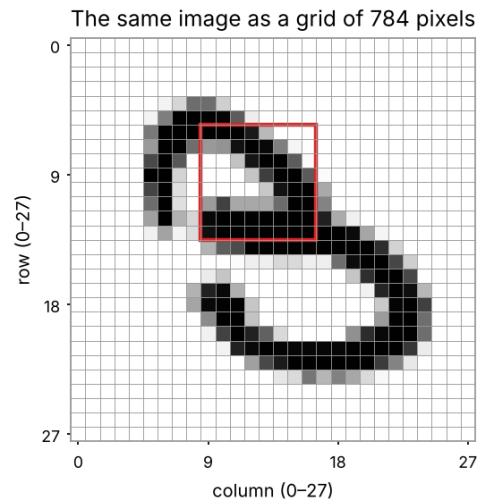
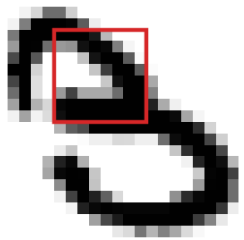
Images are points in a huge space

- A 28×28 image is $M = 784$ numbers.
- Writing them in a row turns the image into a vector $x^{(i)} \in [0,1]^M$, one point in an M -dimensional **data space** (or **pixel space**), with pixels $x_j^{(i)}, j = 1, \dots, M$

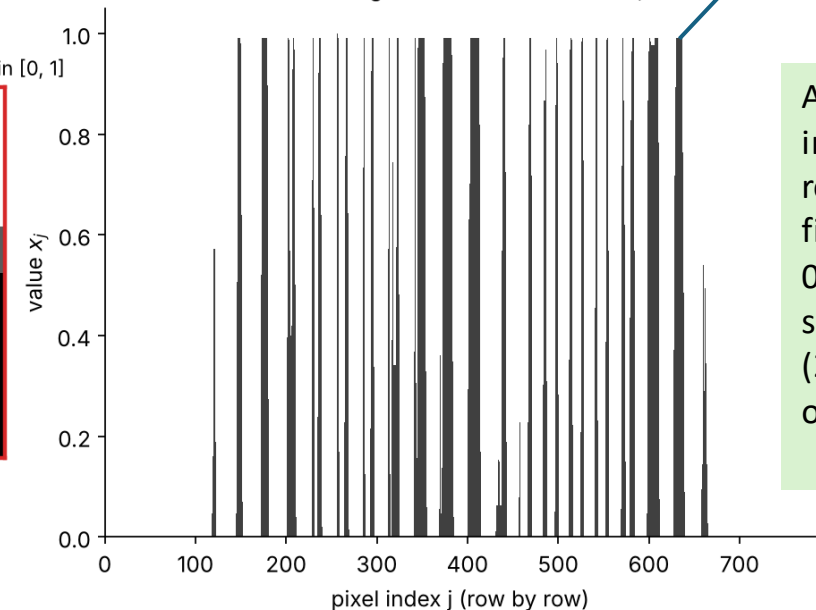
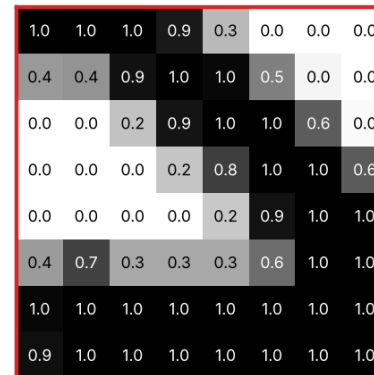
If you pick a point of this 784-dimensional space at random, what does it look like?

Each cluster of tall bars is the ink in one row of the grid

A 28×28 MNIST image of a 3



Zoom on the red box: each pixel is a number in $[0, 1]$

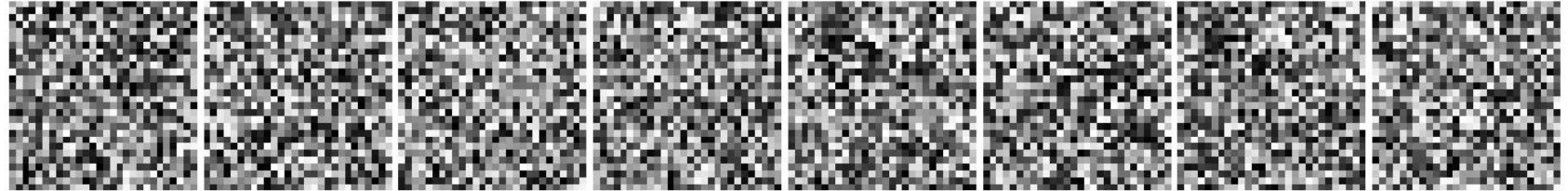


All 784 numbers in one row, top row of pixels first (positions 0–27), then the second row (28–55), and so on.

The Manifold hypothesis

Random points are static; real digits fill a very small part of pixel space.

Eight random points in the 784-dimensional pixel space: pure static



The Manifold hypothesis

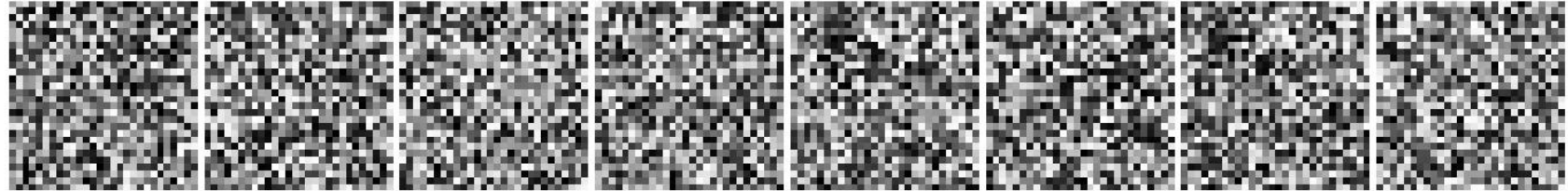
Random points are static; real digits fill a very small part of pixel space.

Real digits lie on a thin, curved band, the **manifold**

The **manifold hypothesis**: natural data lie near a low-dimensional curved surface inside the huge data space

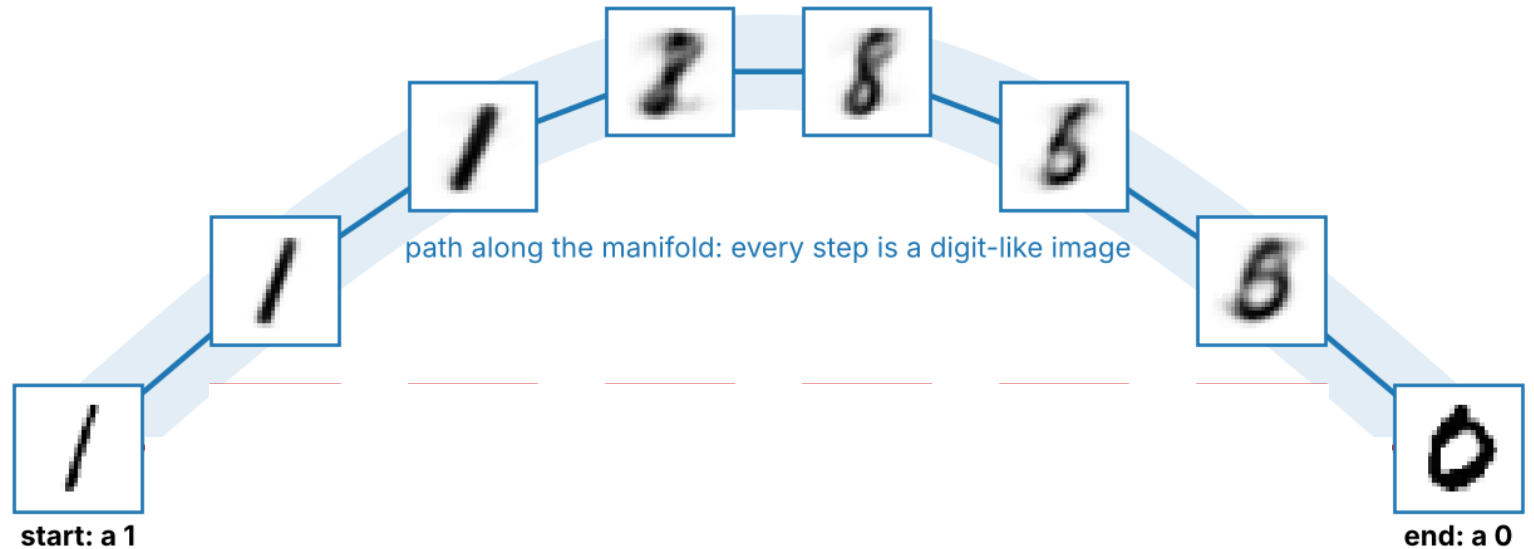
A path that follows the band (blue) passes only through digit-like images. Those images come from the VAE, which learns coordinates on the manifold.

Eight random points in the 784-dimensional pixel space: pure static



pixel space (784 dimensions, drawn in 2-D)

the data manifold: the thin region where images look like digits



The Manifold hypothesis

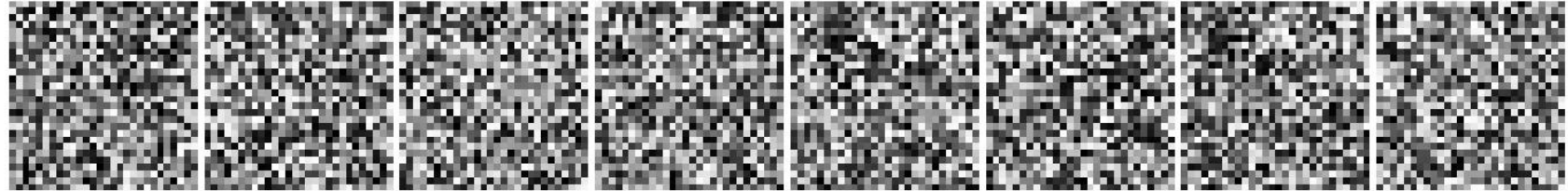
Random points are static; real digits fill a very small part of pixel space.

Real digits lie on a thin, curved band, the **manifold**

The **manifold hypothesis**: natural data lie near a low-dimensional curved surface inside the huge data space

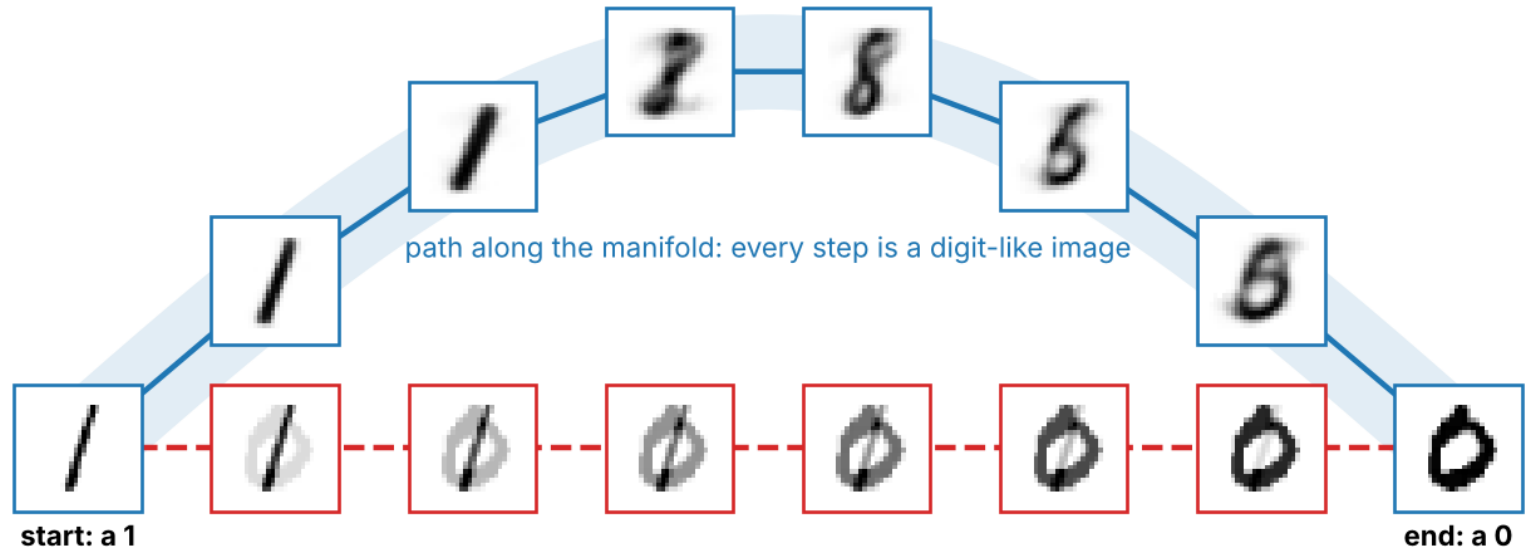
A path that follows the band (blue) passes only through digit-like images. Those images come from the VAE, which learns coordinates on the manifold.

Eight random points in the 784-dimensional pixel space: pure static



pixel space (784 dimensions, drawn in 2-D)

the data manifold: the thin region where images look like digits



straight line in pixel space: leaves the manifold, so its points are double exposures

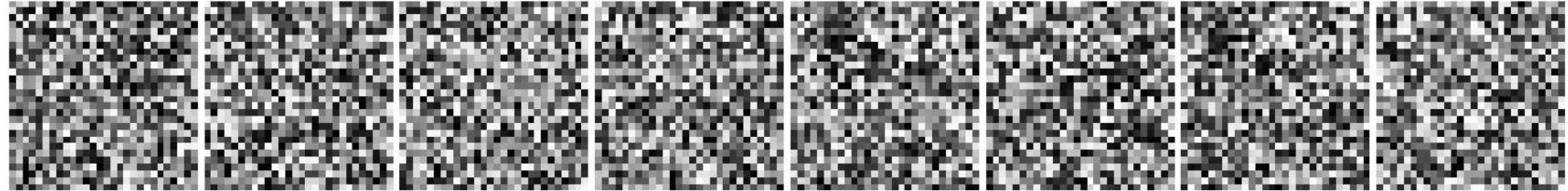
The Manifold hypothesis

Random points are static; real digits fill a very small part of pixel space.

If the digits live near a K -dimensional surface with $K \ll M$, then K coordinates should describe each one.

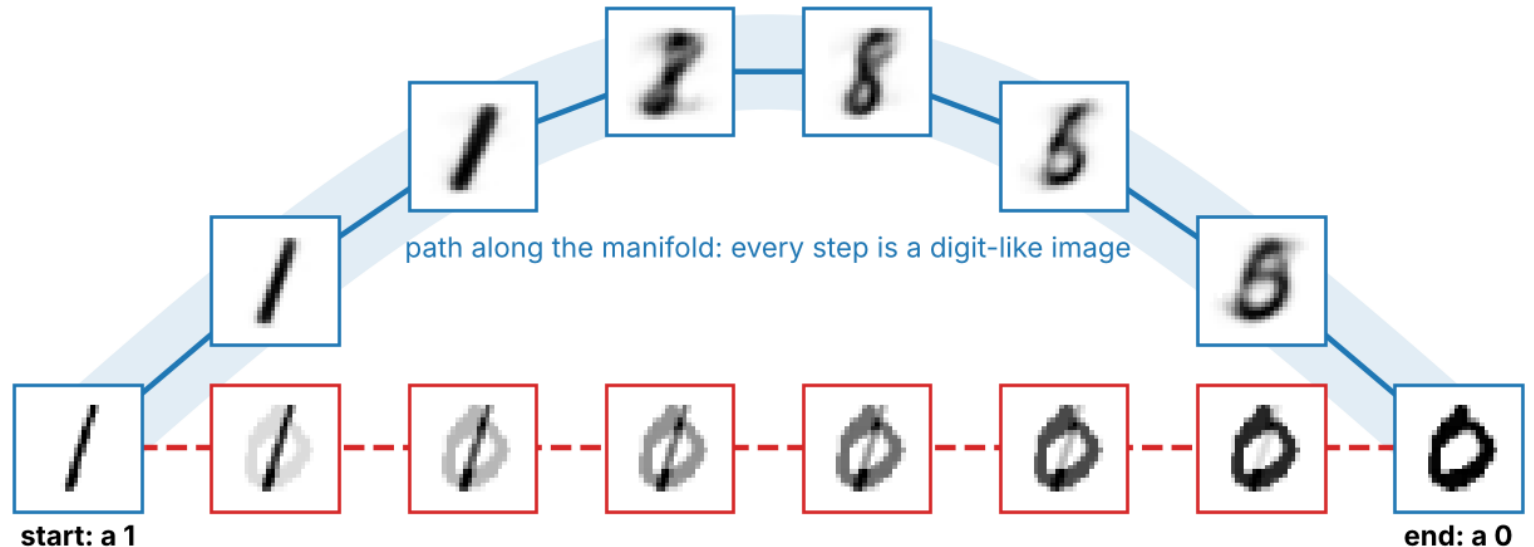
Finding its coordinates is **dimensionality reduction**.

Eight random points in the 784-dimensional pixel space: pure static



pixel space (784 dimensions, drawn in 2-D)

the data manifold: the thin region where images look like digits



straight line in pixel space: leaves the manifold, so its points are double exposures

PCA: the best straight-line compression

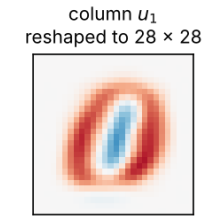
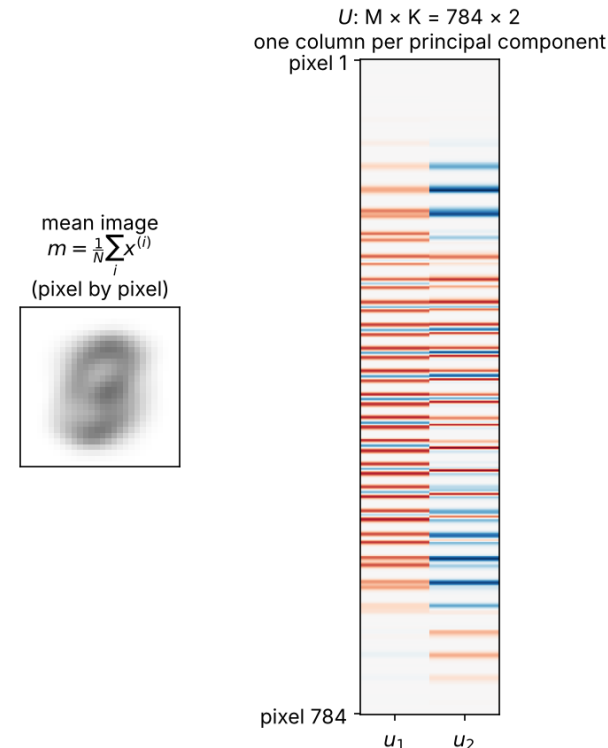
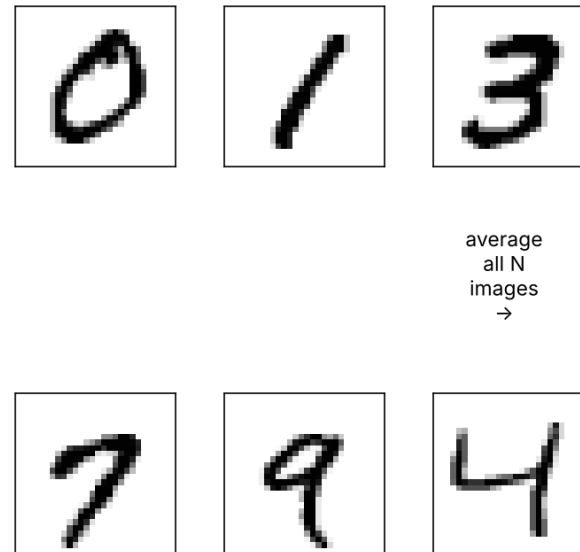
PCA finds the K directions along which the data vary most and describes each image by its position along them

It is already an encoder-decoder pair trained with squared error.

$M \times K$ matrix of **principal components**, the top eigenvectors of the data's covariance matrix

The **mean image** m is the average of all N training images, computed pixel by pixel

$$\text{encode: } z^{(i)} = U^T (x^{(i)} - m), \quad \text{decode: } \hat{x}^{(i)} = U z^{(i)} + m$$



Shapes

$$x^{(i)}, m: 784 \times 1$$

$$U: 784 \times 2$$

$$U^T: 2 \times 784$$

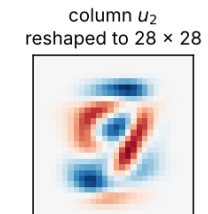
$$z^{(i)} = U^T (x^{(i)} - m)$$

$$(2 \times 784)(784 \times 1) = 2 \times 1$$

$$\hat{x}^{(i)} = U z^{(i)} + m$$

$$(784 \times 2)(2 \times 1) = 784 \times 1$$

$$U^T U = I \text{ (orthonormal columns)}$$

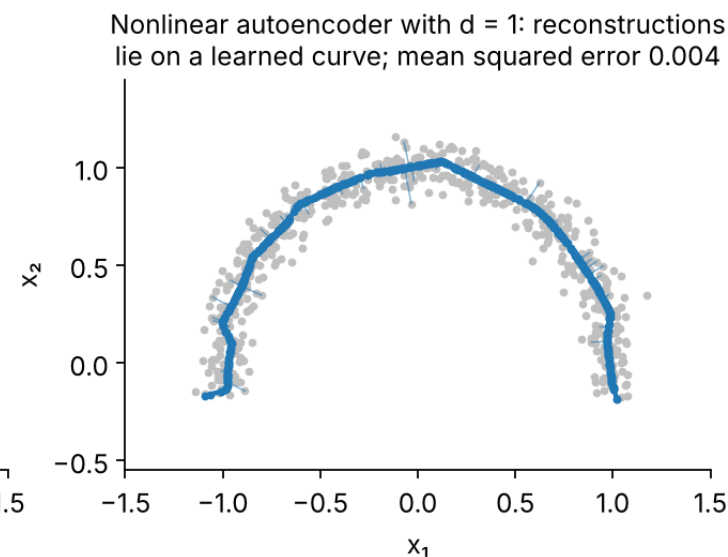
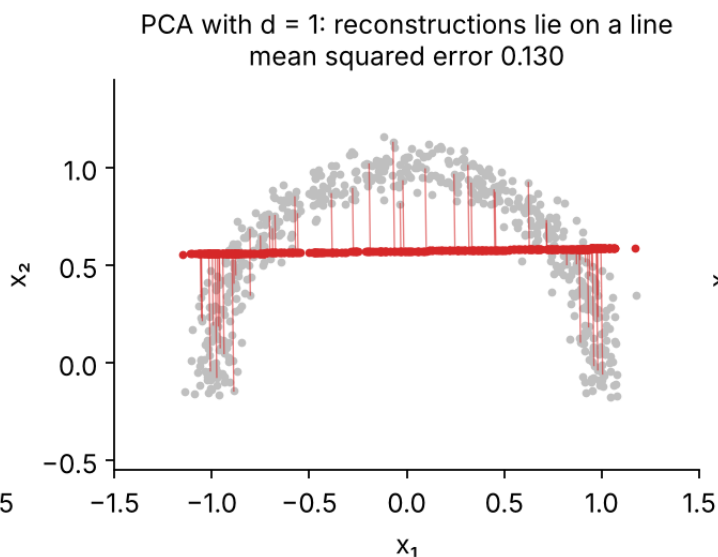
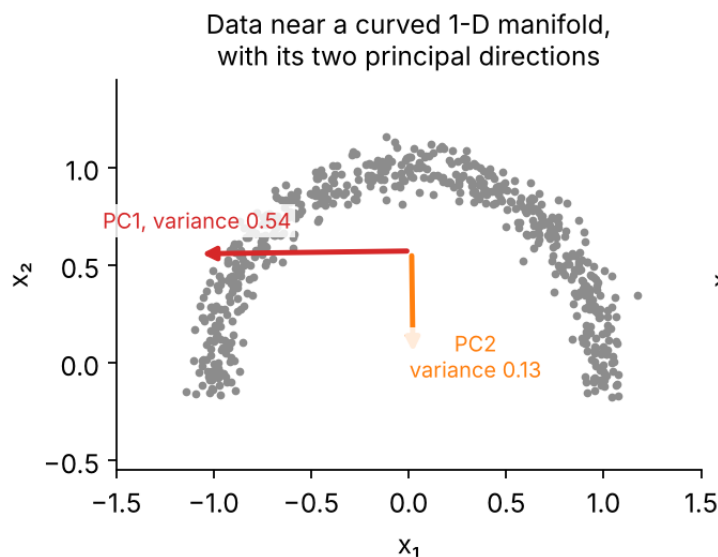


PCA: the best straight-line compression

Its weakness is that both maps are straight lines

PCA (middle) can only place reconstructions on a straight line, so points at the ends of the arc are reconstructed badly: mean squared error 0.130

An **autoencoder** with small networks in place of the two matrices (right) learns a curve that follows the arc, with error 0.004, 35 times smaller.

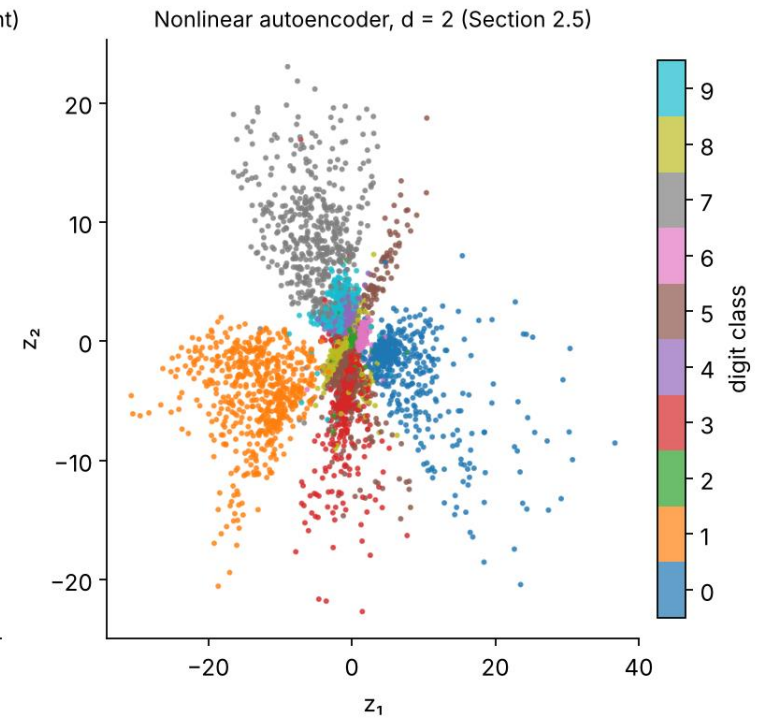
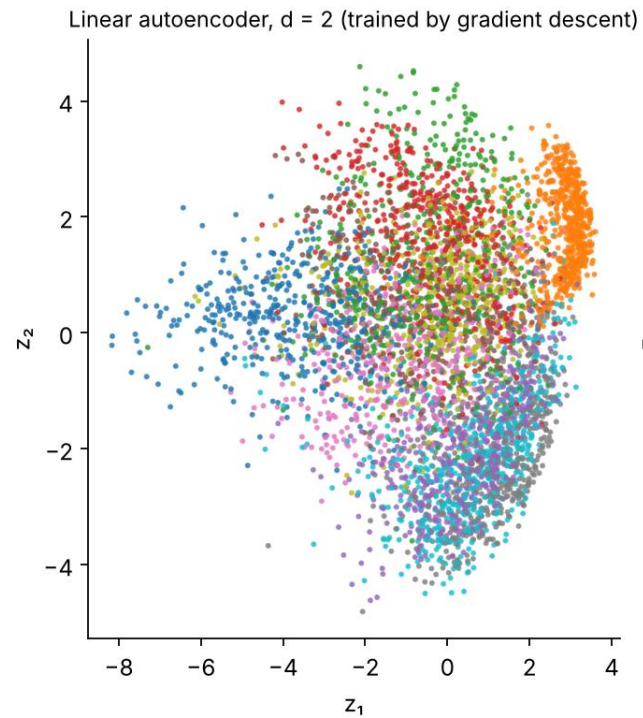
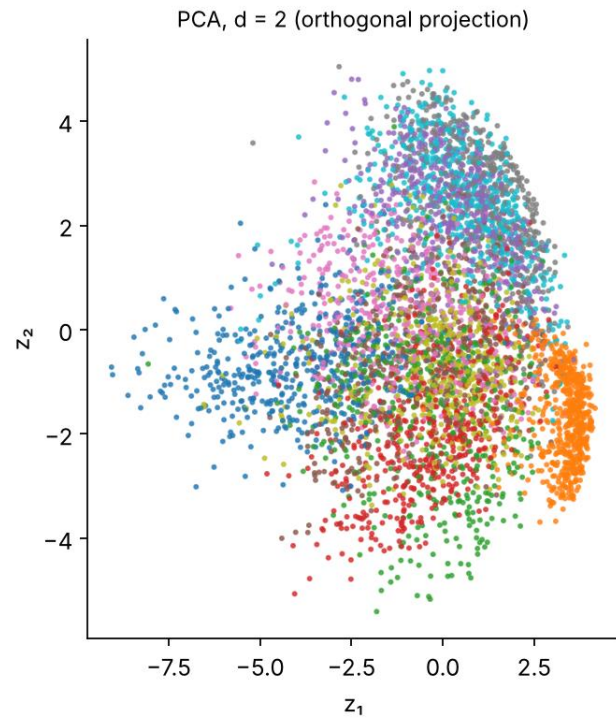


From PCA to the autoencoder

Train a **linear autoencoder**, an encoder matrix and a decoder matrix with no activation functions, by gradient descent on the squared error.

It spans the same directions as PCA, with its own rotated and stretched axes.

Replace the two matrices by neural networks and you have an **autoencoder**.

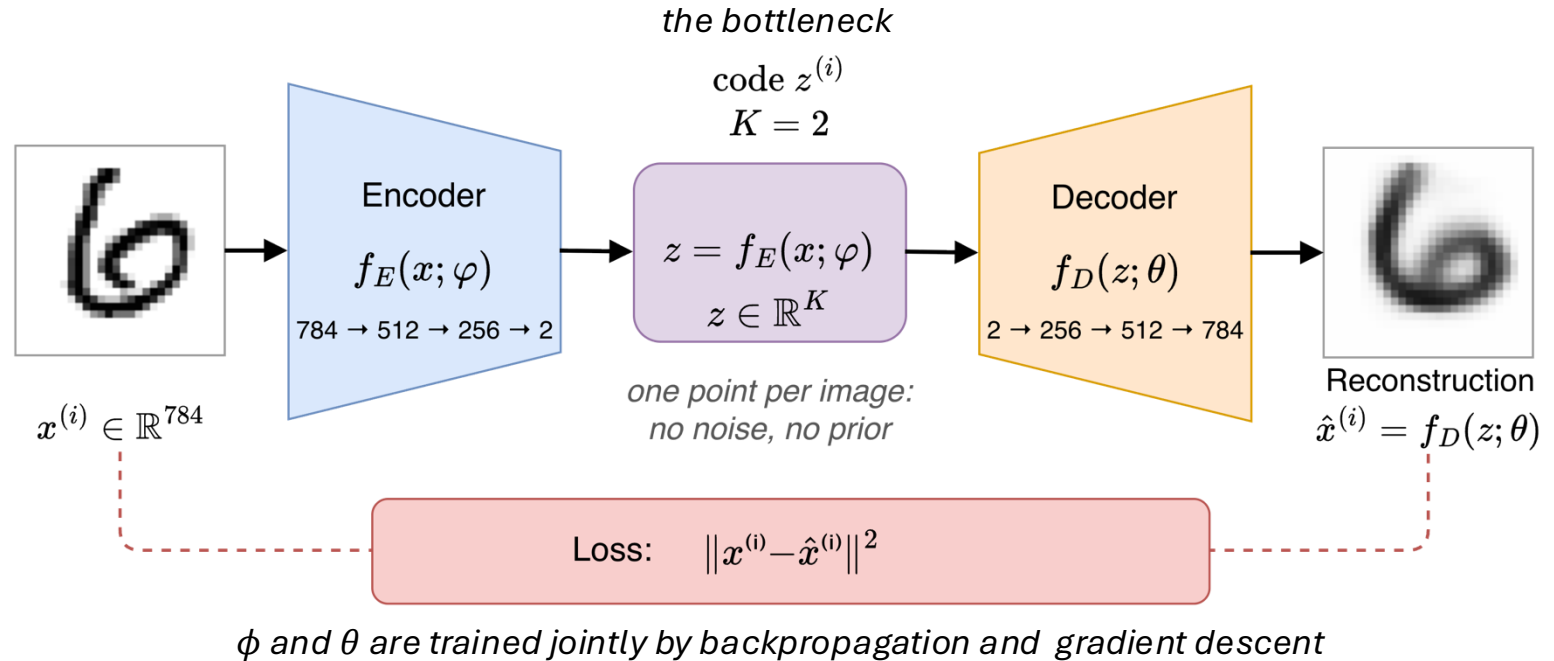


The AutoEncoder

An **autoencoder** has an

- **encoder** $f_E(\cdot; \varphi)$ that compresses an image $x^{(i)} \in \mathbb{R}^M$ into a **latent code** $z^{(i)} = f_E(x^{(i)}; \varphi) \in \mathbb{R}^K$
- **decoder** $f_D(\cdot; \theta)$ that expands it back into a **reconstruction** $\hat{x}^{(i)} = f_D(z^{(i)}; \theta)$

The small code layer is a **bottleneck**: the network cannot copy its input, so it must learn what matters.

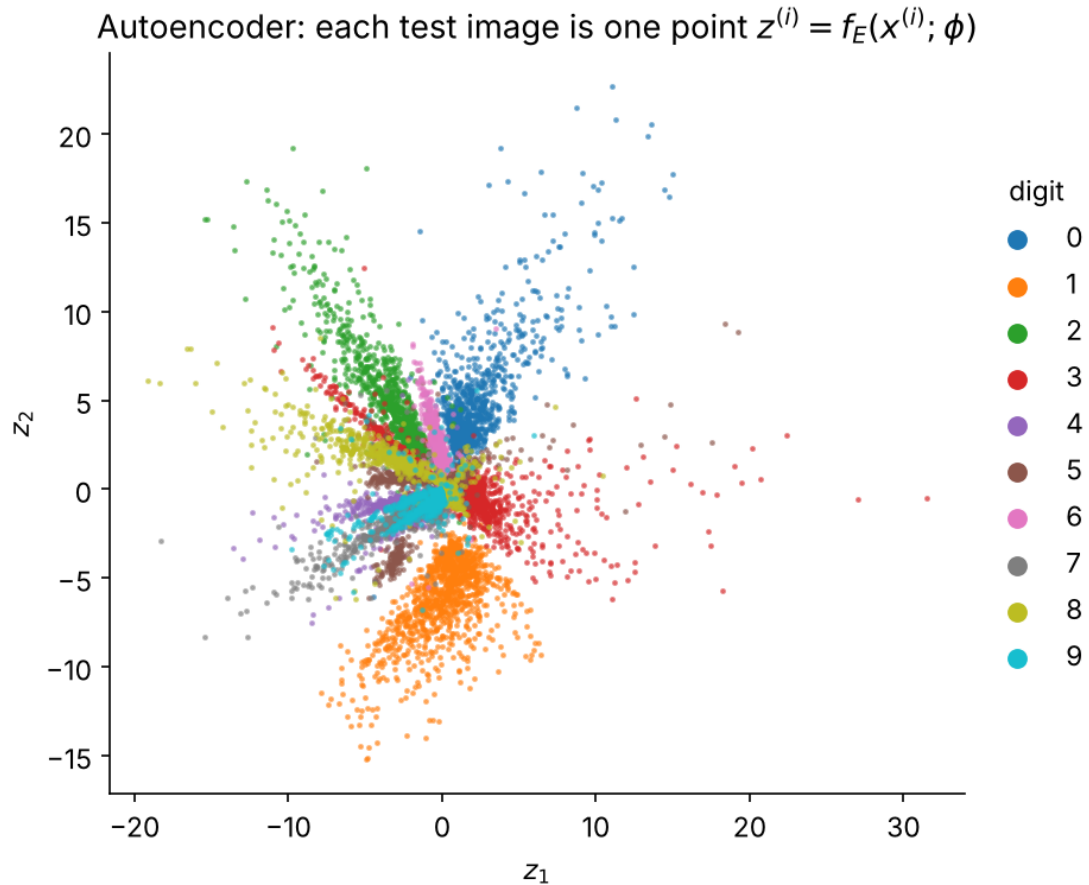


The **objective** is the squared error between $x^{(i)}$ and $\hat{x}^{(i)}$, averaged over the N training images

$$J(\theta, \phi) = \frac{1}{N} \sum_{i=1}^N \|x^{(i)} - f_D(f_E(x^{(i)}; \phi); \theta)\|_2^2$$

φ and θ are trained jointly by backpropagation and gradient descent

Why an AutoEncoder cannot generate?



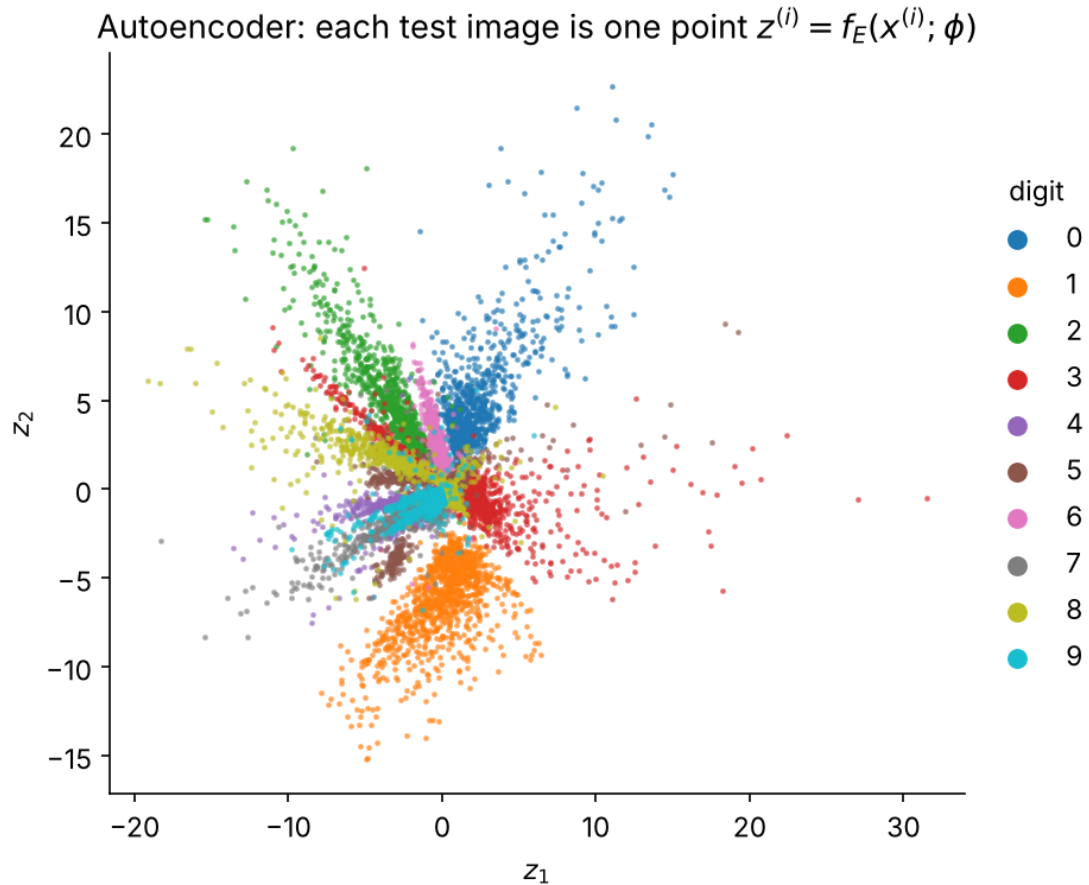
Each dot is the code $z^{(i)} = f_E(x^{(i)}; \phi)$ of one test image.

The autoencoder never saw the labels, yet each digit forms its own arm.

Issues:

- ??

Why an AutoEncoder cannot generate?



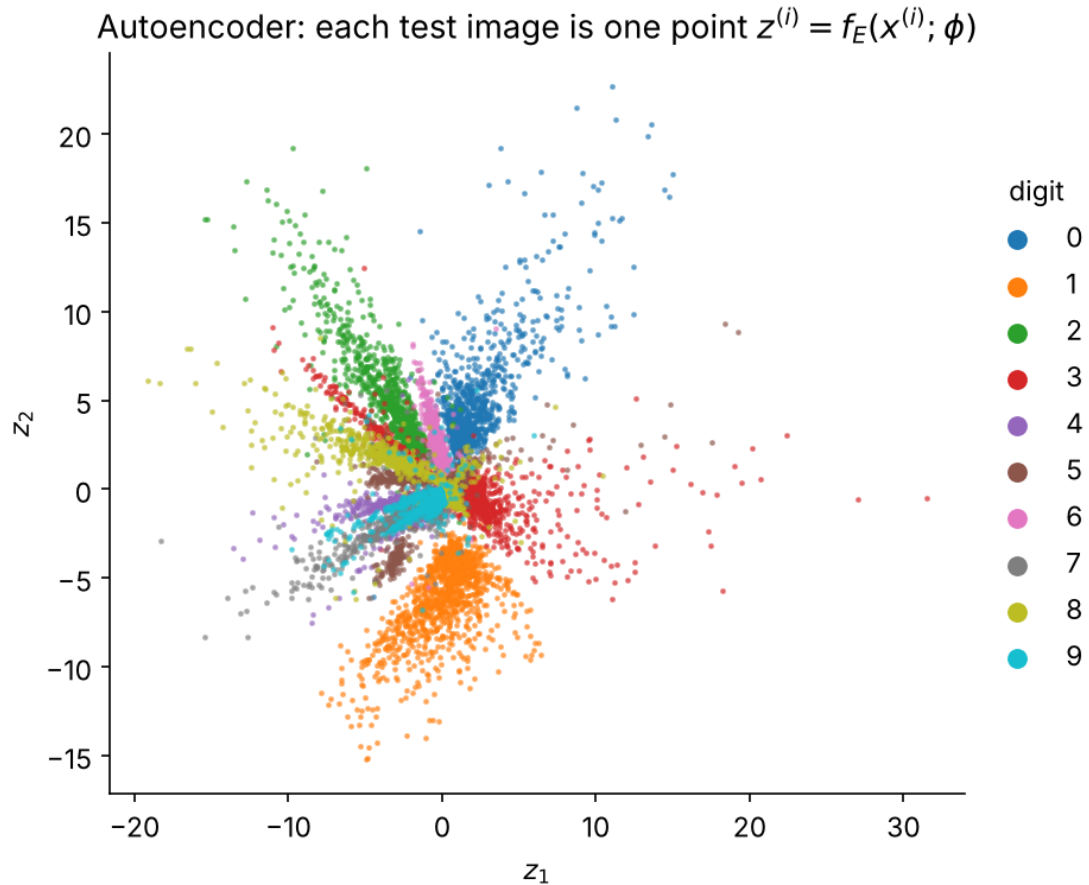
Each dot is the code $z^{(i)} = f_E(x^{(i)}; \phi)$ of one test image.

The autoencoder never saw the labels, yet each digit forms its own arm.

Issues:

- the codes have no fixed scale (from about -19 to +31)
- their density is uneven
- there are empty regions between arms.

Why an AutoEncoder cannot generate?



Each dot is the code $z^{(i)} = f_E(x^{(i)}; \phi)$ of one test image.

The autoencoder never saw the labels, yet each digit forms its own arm.

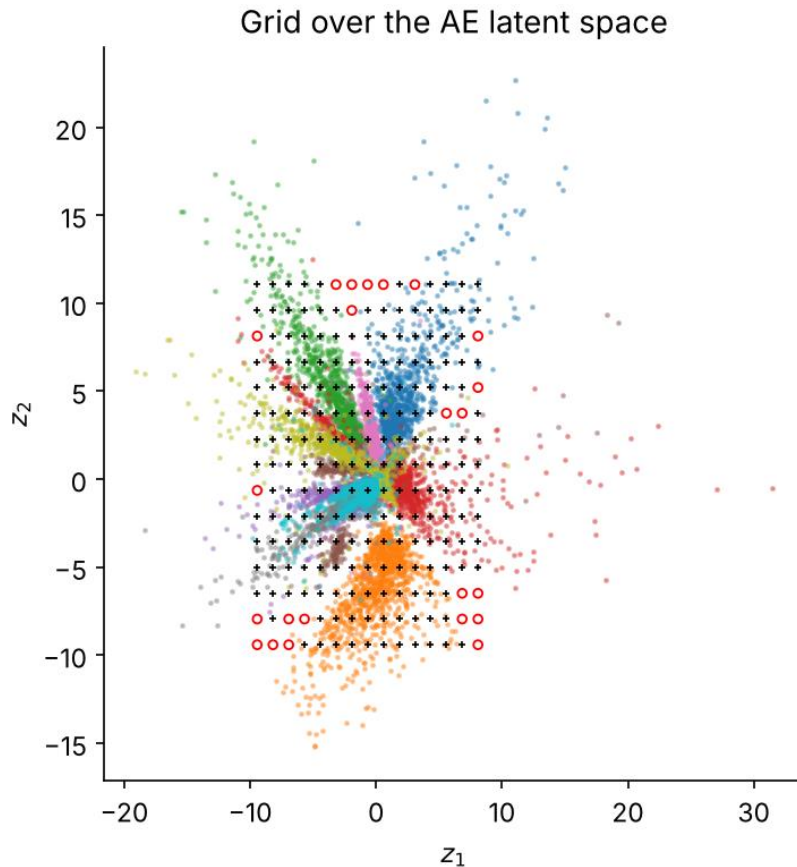
Issues:

- the codes have no fixed scale (from about -19 to +31)
- their density is uneven
- there are empty regions between arms.

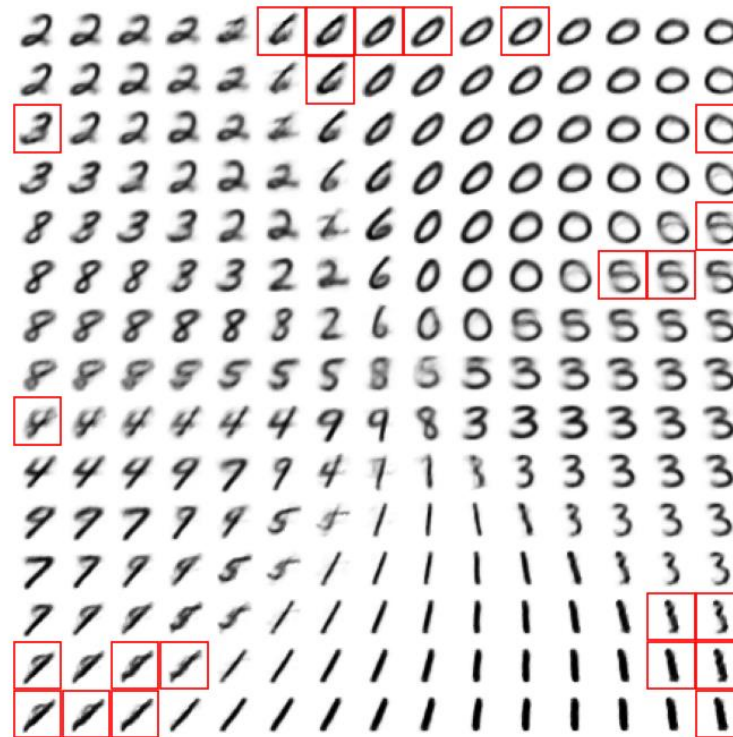
We want a new digit.

Which point of this plane do you decode?

Why an AutoEncoder cannot generate?



f_D on the grid (23 of 225 points far from every code, red)



We decode a 15×15 grid.

23 of the 225 points (red) are far from every code.

No distribution over codes. Nothing says which codes are likely, so there is no principled way to pick one to decode.

No training between the training points. The decoder only ever sees N isolated codes. Nothing forces the latent space to be **continuous** (nearby codes give similar images) or **complete** (every region gives a plausible image).

Variational autoencoder (VAE)

A **variational autoencoder (VAE)** is similar to the autoencoder

- It has the same two networks: an encoder that reads images and a decoder that makes them.

The **difference** is that the encoder now outputs a small *cloud* of codes for each image instead of a single point

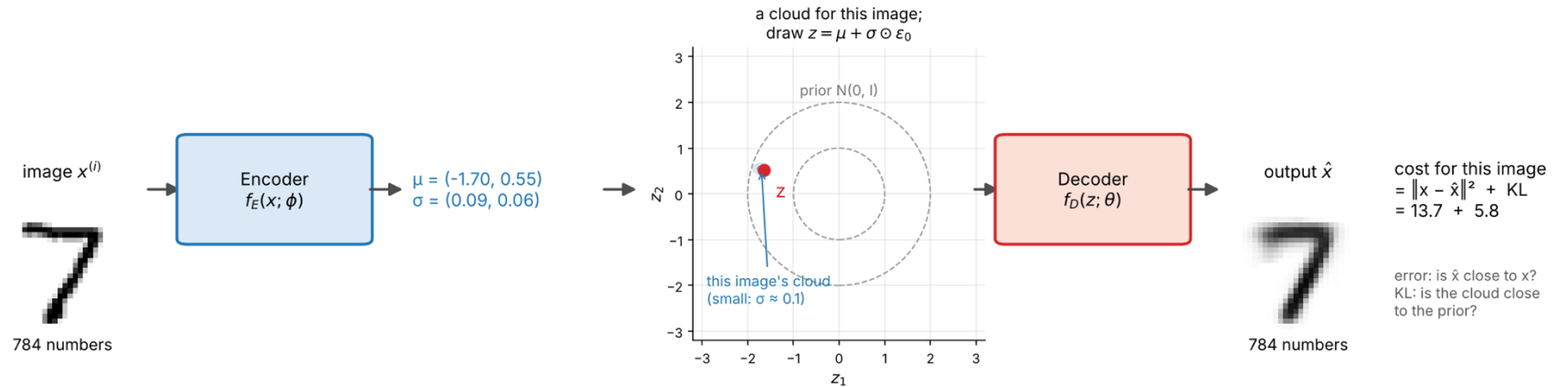
Every cloud is pulled toward one known distribution, $N(0, I)$, so that we know where to pick new codes.

VAE: Training vs Generation

Training

- Encoder reads the 7 and outputs two numbers per code dimension: a centre $\mu = (-1.70, 0.55)$ and a width $\sigma = (0.09, 0.06)$
- Together they describe a small Gaussian cloud in latent space (tiny here because $\sigma \approx 0.1$).
- We draw one code z from that cloud, decode it into $\hat{x}$, and pay two costs: the squared error between the 7 and $\hat{x}$, and the KL term, which measures how far the blue cloud is from the grey prior $N(0, I)$.

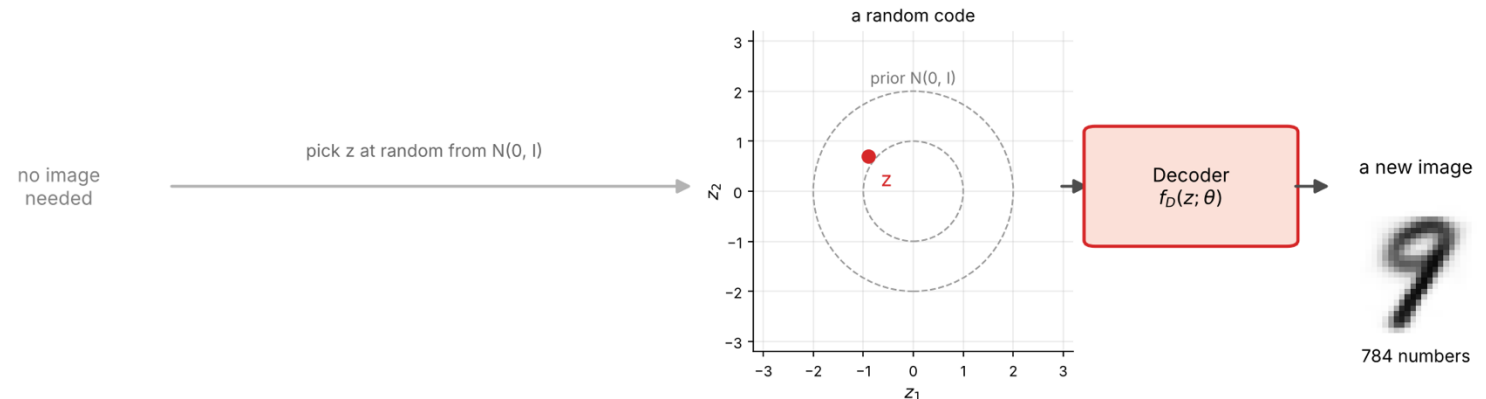
Training: an image goes through the whole VAE



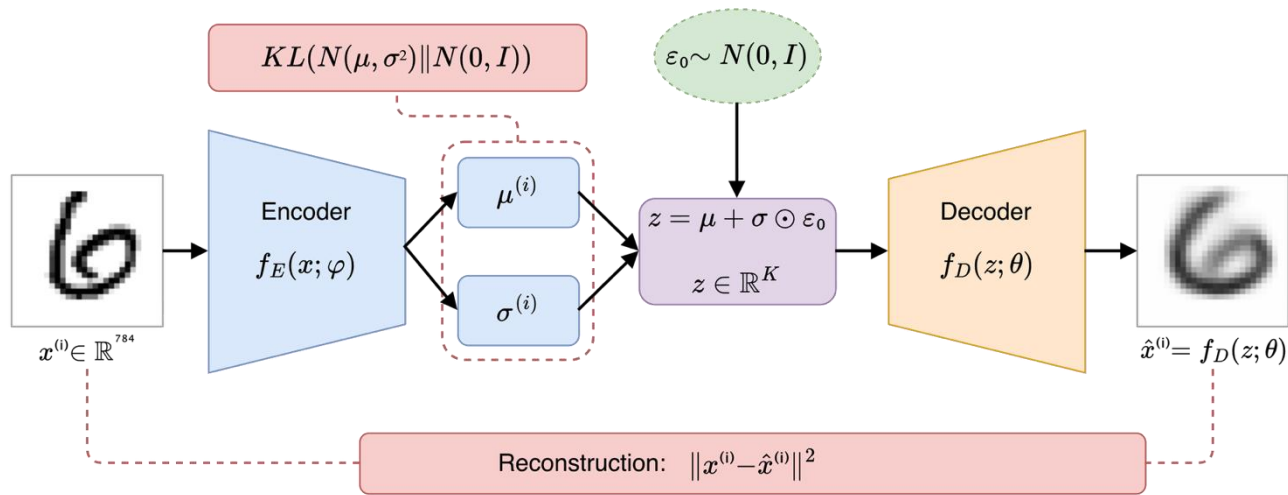
Generation:

- No image and no encoder
- we pick z at random from $N(0, I)$ and decode it.
- This code gives a 9 that is not in the data.

Generating: only the prior and the decoder (the encoder is not used)



VAE Summary



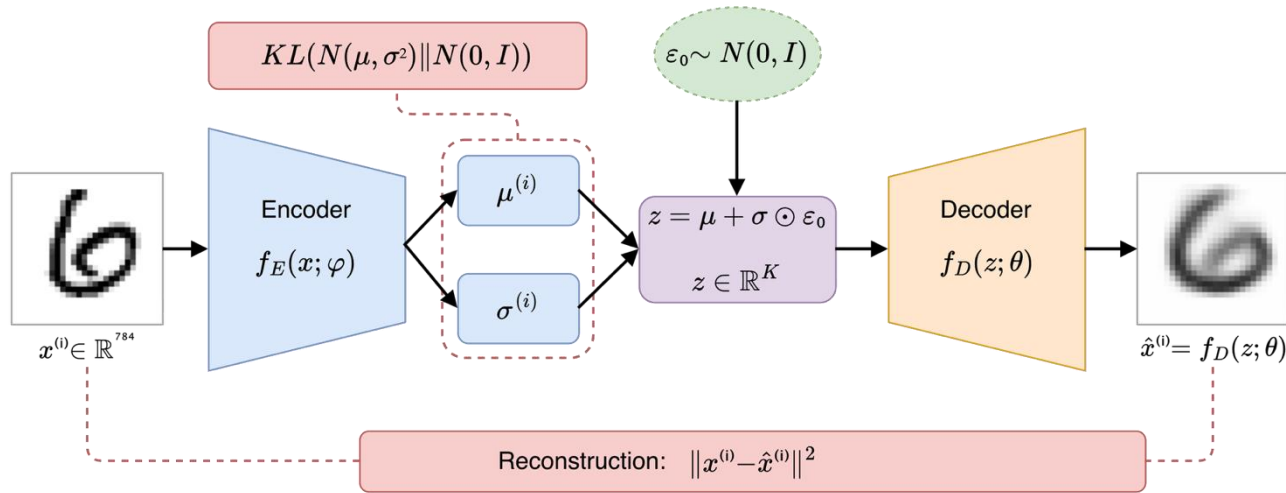
Training: one step per image

- 1 **Encode:** $(\mu, \sigma) = f_E(x; \varphi)$
- 2 **Sample:** $z = \mu + \sigma \odot \varepsilon_0$
- 3 **Decode:** $\hat{x} = f_D(z; \theta)$
- 4 **Loss:** $\|x - \hat{x}\|^2 + KL$
- 5 **Backprop** to both θ and φ

Generation

- 1 **Draw** $z \sim N(0, I)$
- 2 **Decode:** $\hat{x} = f_D(z)$
The encoder is not used.

VAE Summary



Training: one step per image

- 1 **Encode:** $(\mu, \sigma) = f_E(x; \phi)$
- 2 **Sample:** $z = \mu + \sigma \odot \varepsilon_0$
- 3 **Decode:** $\hat{x} = f_D(z; \theta)$
- 4 **Loss:** $\|x - \hat{x}\|^2 + KL$
- 5 **Backprop** to both θ and ϕ

Generation

- 1 **Draw** $z \sim N(0, I)$
- 2 **Decode:** $\hat{x} = f_D(z)$
The encoder is not used.

1 Generative model

$$z \sim \mathcal{N}(0, I), \quad p_\theta(x | z) = \mathcal{N}(x; f_D(z; \theta), \tau^2 I)$$

2 Marginal likelihood

$$p_\theta(x) = \int p_\theta(x | z) p(z) dz \quad (\text{intractable})$$

3 Encoder: amortized variational inference

$$q_\phi(z | x^{(i)}) = \mathcal{N}(\mu^{(i)}, \text{diag}(\sigma^{(i)2})), \quad (\mu^{(i)}, \sigma^{(i)}) = f_E(x^{(i)}; \phi)$$

4 ELBO: a lower bound on $\log p(x)$

$$\log p_\theta(x^{(i)}) = \text{ELBO}(x^{(i)}) + D_{KL}(q_\phi(z | x^{(i)}) \| p(z | x^{(i)})) \geq \text{ELBO}(x^{(i)})$$

5 Objective we minimize ($\tau^2 = \frac{1}{2}$)

$$J(\theta, \phi) = \sum_{i=1}^N \left[\|x^{(i)} - f_D(z^{(i)}; \theta)\|_2^2 + D_{KL}(\mathcal{N}(\mu^{(i)}, \sigma^{(i)2}) \| \mathcal{N}(0, I)) \right]$$

6 KL term in closed form

$$D_{KL}(\mathcal{N}(\mu, \sigma^2) \| \mathcal{N}(0, I)) = \frac{1}{2} \sum_{k=1}^K (\mu_k^2 + \sigma_k^2 - 1 - \log \sigma_k^2)$$

7 Reparameterization trick

$$z^{(i)} = \mu^{(i)} + \sigma^{(i)} \odot \varepsilon_0, \quad \varepsilon_0 \sim \mathcal{N}(0, I)$$

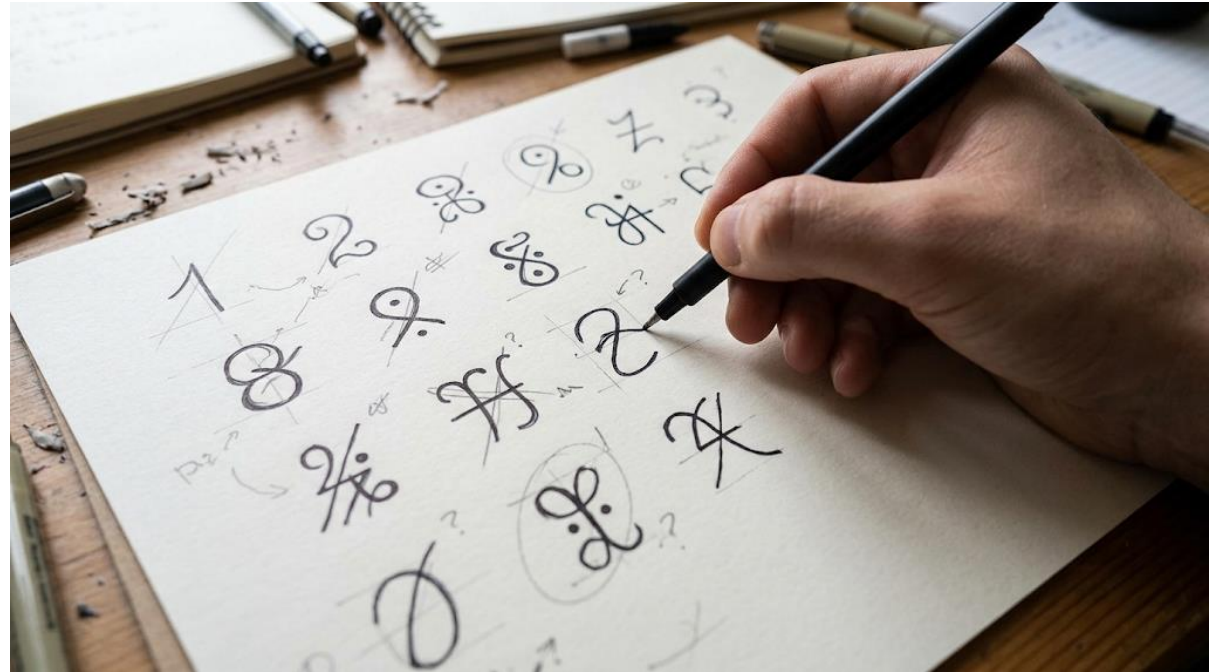
The Generative Process

The Decoder

How does a VAE make an image? (Prior, decoder, ancestral sampling, Gaussian pixels give squared error)

How images are made (the decoder)?

Write a program that *invents* new handwritten digits. Where would you begin?



How images are made (the decoder)?

Write a program that *invents* new handwritten digits. Where would you begin?

The VAE tells this story in two steps:

Pick a code.

Draw z from the **prior** $N(0, I)$. That means K random numbers, each drawn on its own from a bell curve centred at 0 with spread 1. In our model $K = 2$, so a code is just a point on a sheet of paper, and it will usually land within 2 of the center

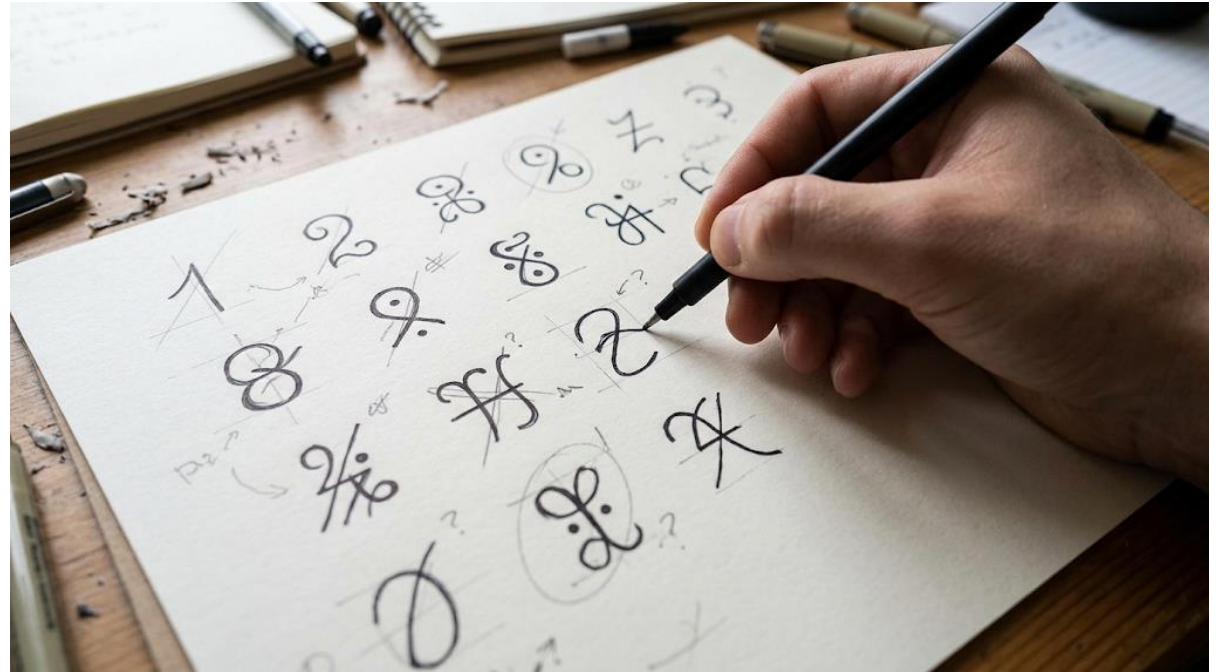
Turn it into an image.

Hand that point to the decoder.

It computes $\hat{x} = f_D(z; \theta)$:

784 numbers between 0 and 1, **one per pixel**.

Think of $\hat{x}$ as the decoder's best guess at the image.



The story then says the real image x is that guess plus a little random noise, drawn separately for every pixel:

$$x_j = \hat{x}_j + \text{noise}_j, \\ \text{noise}_j \sim N(0, \tau^2).$$

How images are made?

- We call z a **latent (hidden) variable**.
- 60,000 images $x^{(i)}$, but nobody wrote down the code behind any of them.
- A model built around a hidden cause like this is a **latent variable model**.
- z is a very short note describing the image (which digit, how slanted, how thick), and the decoder is the hand that reads the note and draws.
- In symbols, the two steps are
 - $z \sim N(0, I)$ and
 - $x|z \sim N(f_D(z; \theta), \tau^2 I)$.

60 real handwritten digits from the MNIST test set

4	3	6	4	3	5	2	8	0	7
8	1	8	3	6	6	7	0	5	0
4	9	4	4	9	3	4	1	6	8
6	9	4	8	8	2	3	4	7	3
0	0	2	5	4	8	8	6	1	7
1	1	7	4	6	7	1	2	4	8

Ancestral Sampling



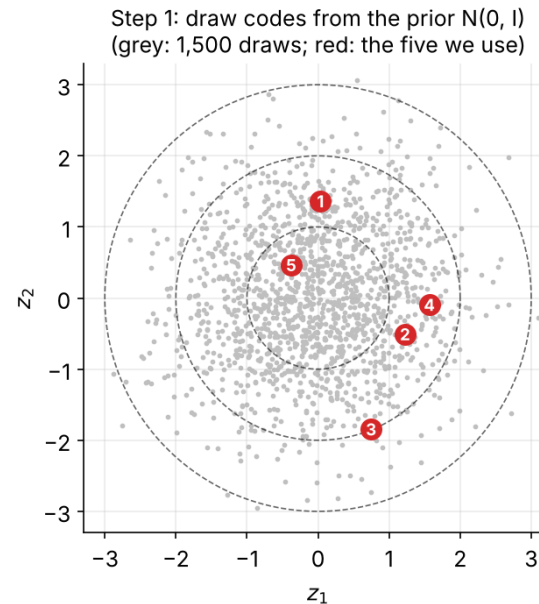
Step1:

Every grey dot is one roll of the dice, one draw from $N(0, I)$

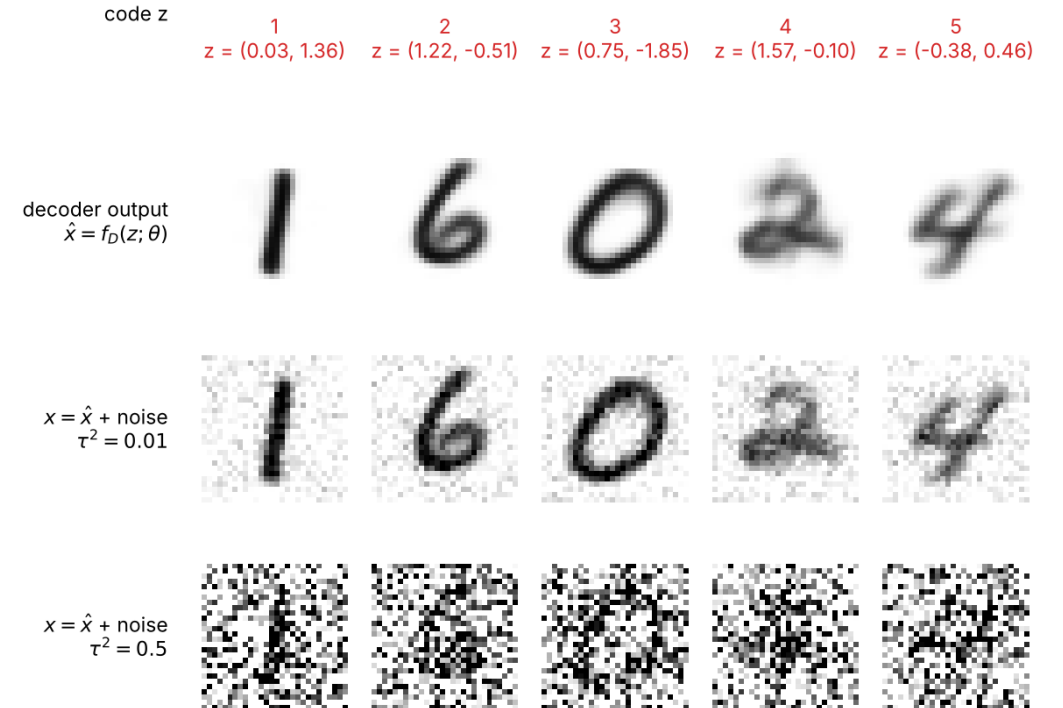
Draw a code from the prior, for example $z = (0.03, 1.36)$

Why start from a normal distribution?

Any simple distribution would do, as long as it is easy to sample from
We pick $N(0, I)$ because it has nothing to learn, it looks the same in every direction, and its KL divergence to another Gaussian has a closed form,



Step 2: decode each code, then add pixel noise



Ancestral Sampling

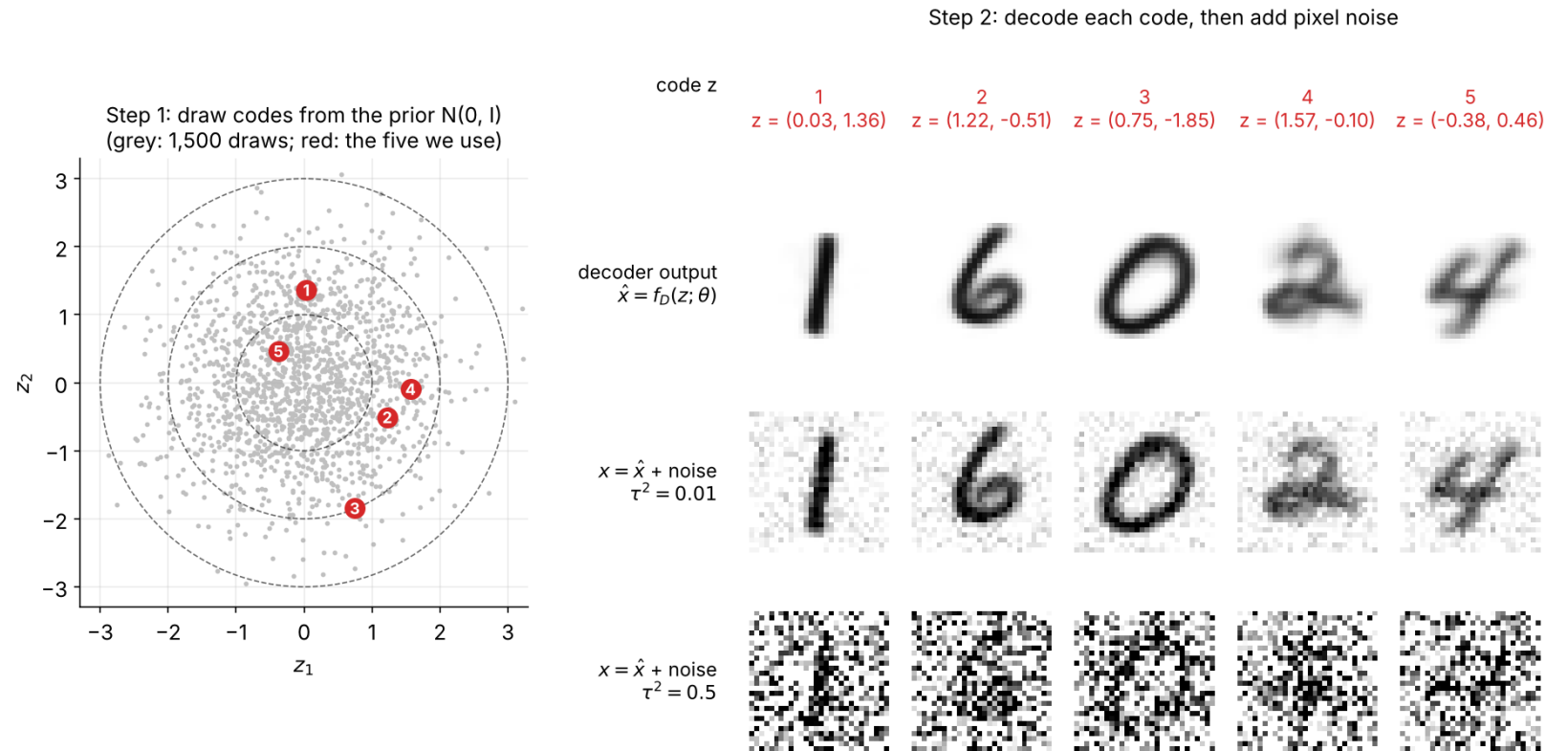
Step2:

Hand it to the Decoder and output comes a 1

Decoder's output: $\hat{x} = f_D(z; \theta)$ is 784 numbers in $[0,1]$, one per pixel
The sigmoid keeps them in range.
Reshaped, that is a 28×28 grey image

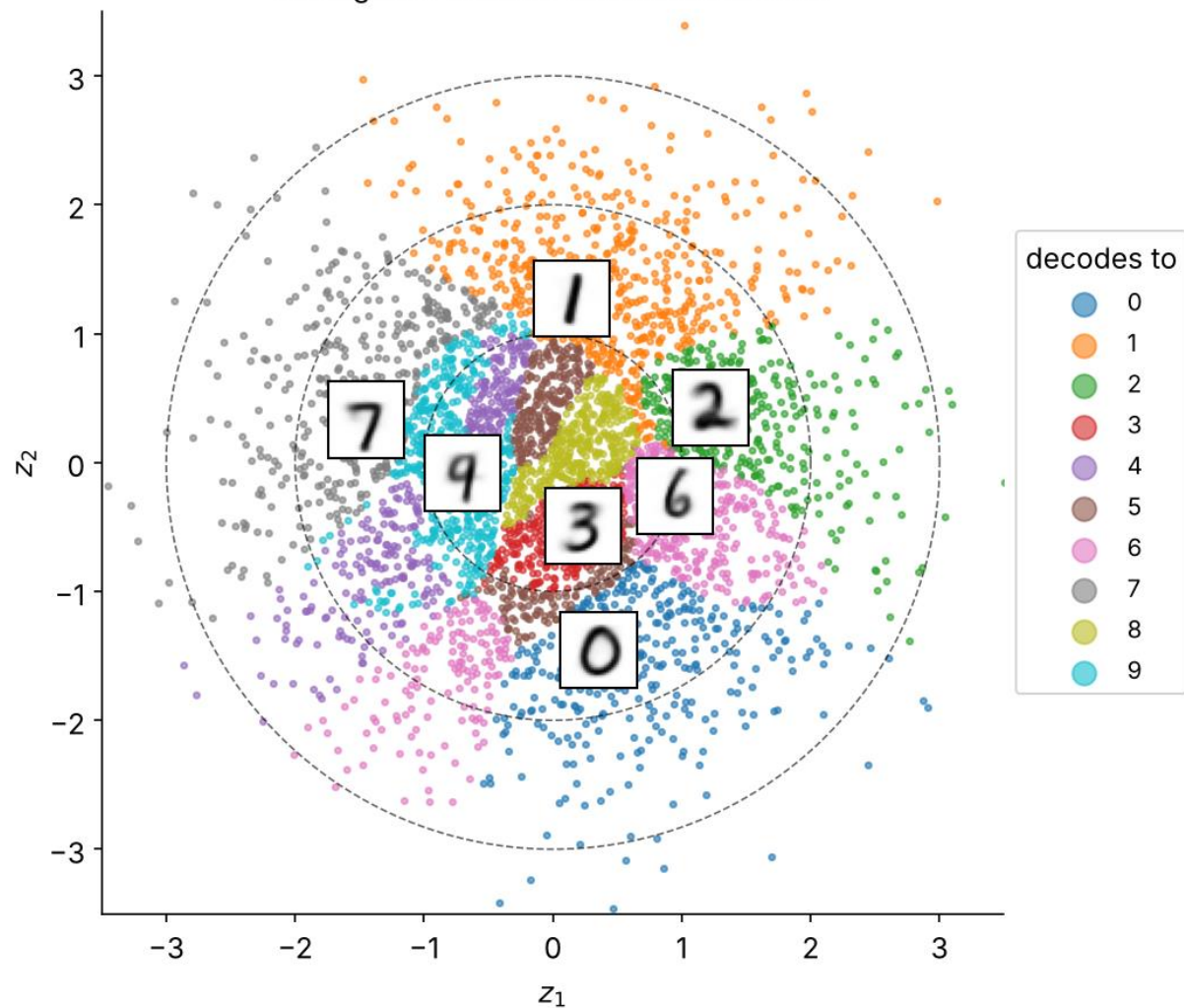
The noise is not there to make better pictures; **in practice we show $\hat{x}$, never the noisy x** . It is there so the model gives a probability to real images. Without noise the decoder could only produce exact $\hat{x}$ images, and every real MNIST digit would get probability 0, leaving nothing to train on. With noise $N(0, \tau^2)$ per pixel, an image close to $\hat{x}$ is likely and one far away is unlikely.

$-\log$ of that probability is $\|x - \hat{x}\|^2 / 2\tau^2$, the squared-error loss



Our model uses $\tau^2 = 0.5$, so a true sample from the story looks like the random noise in the bottom row, and nobody wants to look at that. The noise is there to tell the model how far a real image is allowed to be from $\hat{x}$.

4,000 codes from $N(0, I)$, coloured by the digit the decoder turns each one into

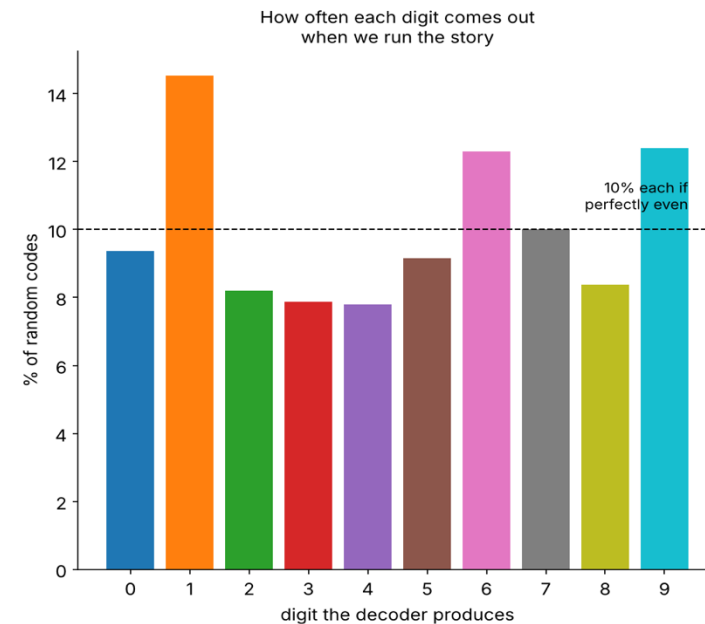


Let us draw 4,000 codes from $N(0, I)$, decode each one, and color the dot by the digit that came out.

The decoder cuts the round cloud into slices, one per digit.

The 1s come out most often (14.5%) and the 4s least often (7.8%), because the decoder gave them slices of different sizes.

Who decides those sizes? **The encoder and the KL term**



The whole story in one line

The VAE is a **latent variable model (LVM)**.

It specifies how data are generated:

$$p(x, z) = \underbrace{p(z)}_{\text{step 1: } \mathcal{N}(z; 0, I)} \times \underbrace{p(x | z)}_{\text{step 2: } \mathcal{N}(x; f_D(z; \theta), \tau^2 I)}$$

The chance of picking a code z and then getting the image x from it is the chance of step 1 times the chance of step 2.
This product is the **joint distribution**

how likely is this code

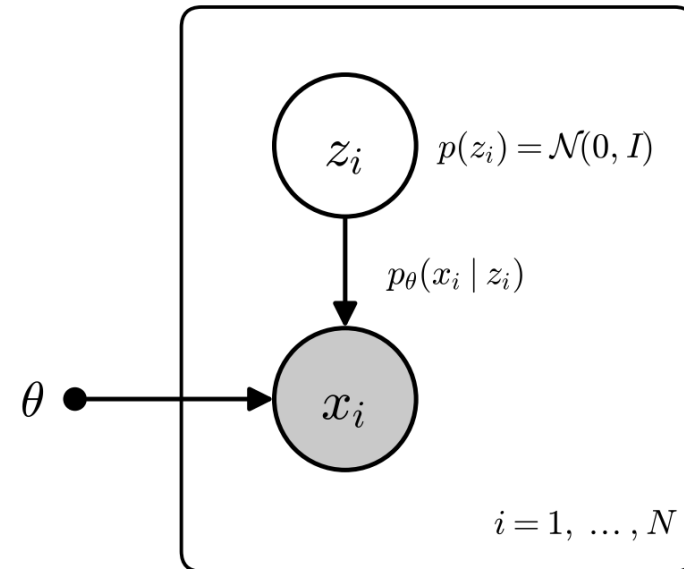
how likely is this code to draw this image

The whole story in one line

As a **probabilistic graphical model**, this is a single latent node z with an edge to an observed node x , inside a **plate** repeated N times.

Generating new data by following this process forward is called **ancestral sampling**. It uses only the prior and the decoder

(a) Generative model



The decoder predicts the center of a Gaussian for each pixel

Starting point: the reconstruction objective.

For an input $x^{(i)} \in \mathbb{R}^M$ the autoencoder computes a code and a reconstruction, and is trained on the mean squared reconstruction error over the N training images:

$$z^{(i)} = f_E(x^{(i)}; \phi), \quad \hat{x}^{(i)} = f_D(z^{(i)}; \theta)$$

$$J(\theta, \phi) = \frac{1}{N} \sum_{i=1}^N \|x^{(i)} - \hat{x}^{(i)}\|_2^2$$

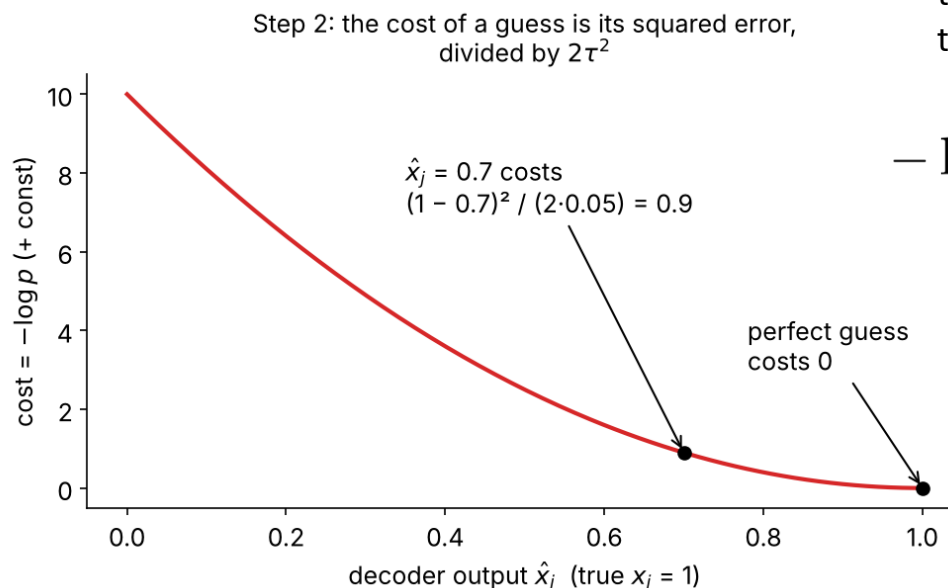
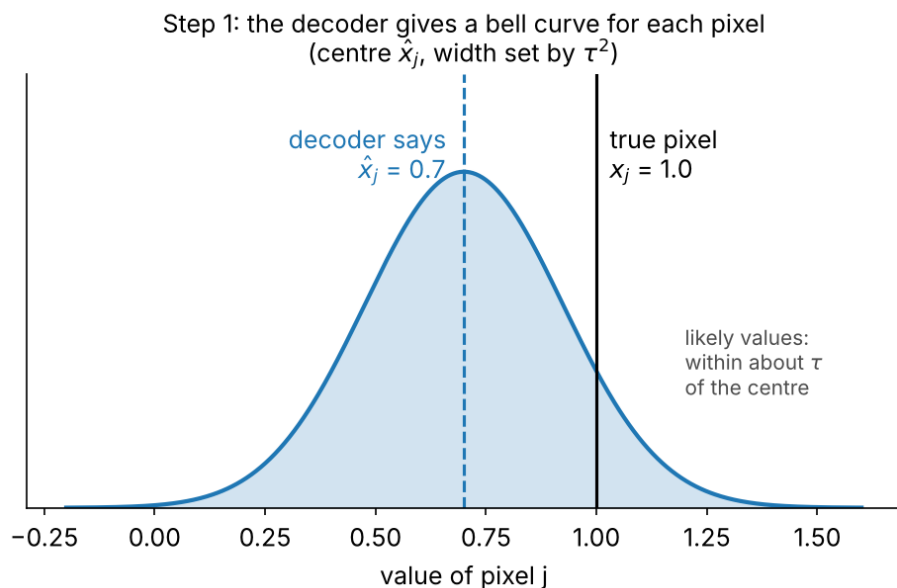
The question is whether this objective follows from a probabilistic model.

Model for one pixel.

Given a code z , the decoder outputs $\hat{x} = f_D(z; \theta) \in [0,1]^M$. The model assumes that the observed value of pixel j is Gaussian around the decoder's output, with a fixed variance τ^2

$$p_\theta(x_j | z) = \mathcal{N}(x_j; \hat{x}_j, \tau^2) = \frac{1}{\sqrt{2\pi\tau^2}} \exp\left(-\frac{(x_j - \hat{x}_j)^2}{2\tau^2}\right), \quad \hat{x}_j = [f_D(z; \theta)]_j$$

The **negative log-likelihood** of the observation as a function of the prediction $\hat{x}_j$



$$-\log p_\theta(x_j | z) = \frac{(x_j - \hat{x}_j)^2}{2\tau^2} + \frac{1}{2} \log(2\pi\tau^2)$$

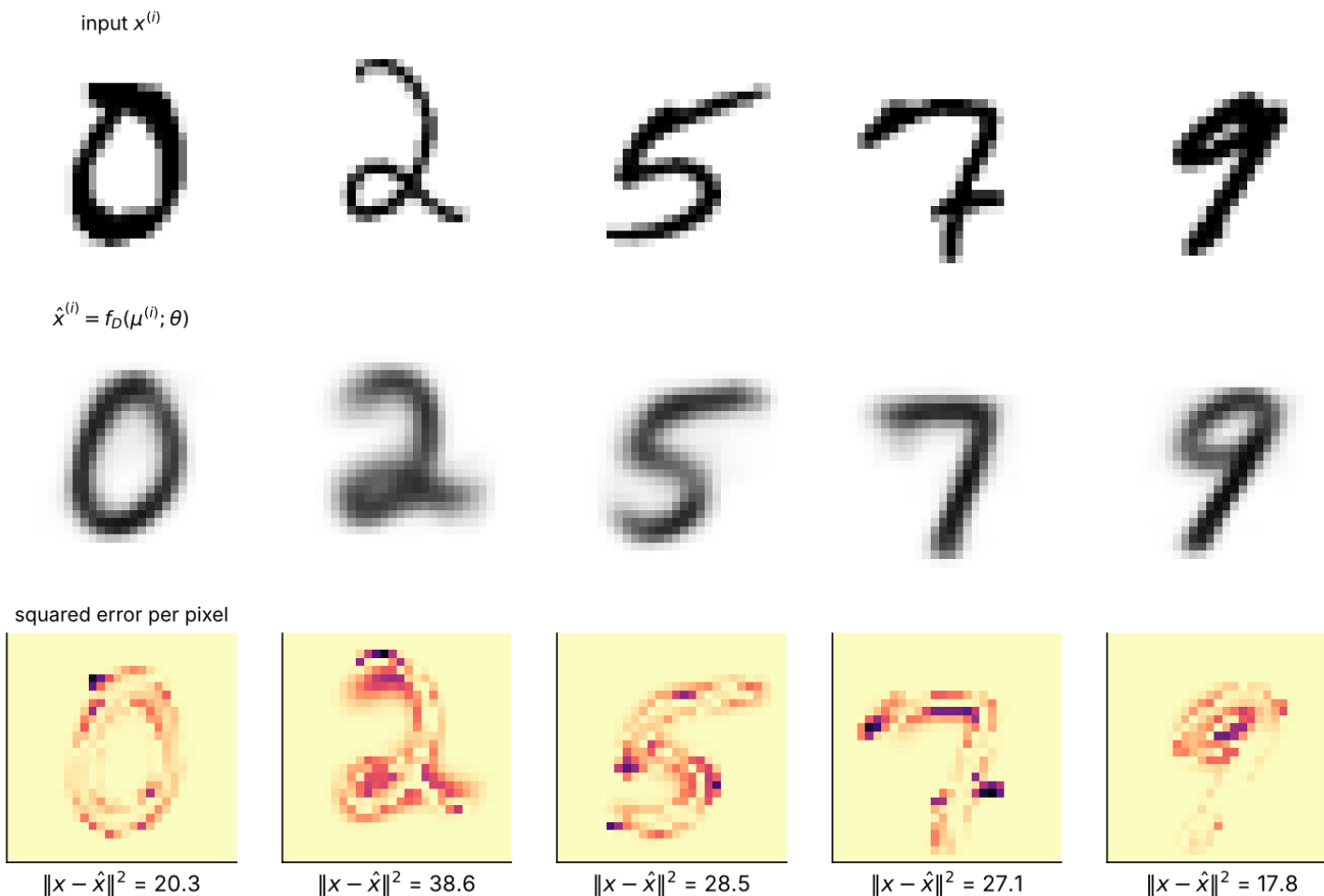
L2 Loss example

The model treats every pixel's noise as independent, so to get the cost of a whole image we simply add up the costs of all M pixels

$$\begin{aligned}\text{cost of image } x &= -\log p(x | z) \\ &= \frac{\|x - f_D(z; \theta)\|_2^2}{2\tau^2} + \text{const}\end{aligned}$$

- The squared error per-pixel is $(x_j - \hat{x}_j)^2$
- The number under each column is the *sum* $\|x - \hat{x}\|^2$.
- With $\tau^2 = 0.5$ this sum is exactly $-\log p_\theta(x|z)$ up to the constant $(M/2)\log\pi \approx 448.7 \text{ nats}$.

The reconstruction term is the sum of 784 squared pixel errors



Maximum likelihood to L2 Loss

$$-\log p_{\theta}(x_j | z) = \frac{(x_j - \hat{x}_j)^2}{2\tau^2} + \frac{1}{2} \log(2\pi\tau^2) \quad \text{for one pixel in one image}$$

The likelihood of the training set

$$-\log p(D | \theta, \phi) = -\sum_{i=1}^N \sum_{j=1}^M \log p(x_j^{(i)} | z^{(i)}) \quad p(x_j^{(i)} | z^{(i)}) = \mathcal{N}(\hat{x}_j^{(i)}, \tau^2)$$

Substitute the Gaussian density.

Each term is the one-pixel cost

$$-\log \left(\frac{1}{\sqrt{2\pi\tau^2}} \exp \left(-\frac{(x_j^{(i)} - \hat{x}_j^{(i)})^2}{2\tau^2} \right) \right) = \frac{(x_j^{(i)} - \hat{x}_j^{(i)})^2}{2\tau^2} + \log \sqrt{2\pi\tau^2}$$

Sum over pixels and images.

The squared terms over the M pixels of one image add up to a squared Euclidean norm, and the constants add up to $NM \log \sqrt{2\pi\tau^2}$

$$-\log p(D | \theta, \phi) = \frac{1}{2\tau^2} \sum_{i=1}^N \|x^{(i)} - \hat{x}^{(i)}\|_2^2 + NM \log \sqrt{2\pi\tau^2}$$

L2 Loss **constant**

Drop what does not depend on the parameters.

$$\arg \min_{\theta, \phi} -\log p(D | \theta, \phi) = \arg \min_{\theta, \phi} \sum_{i=1}^N \sum_{j=1}^M (x_j^{(i)} - \hat{x}_j^{(i)})^2$$

Maximum likelihood under Gaussian pixels is least squares, equivalent to minimizing the L2 reconstruction error

The Gaussian model's cost is the autoencoder's squared error, scaled by $1/(2\tau^2)$

$$= \arg \min_{\theta, \phi} \sum_{i=1}^N \|x^{(i)} - f_D(f_E(x^{(i)}; \phi); \theta)\|_2^2$$

The marginal likelihood $p(\mathbf{x})$

How likely is an image?

We have been dealing with “how likely an image is if we know its code: $p_\theta(x|z)$ ”.

But a training set contains images and no codes.

To train by maximum likelihood, we need the probability of each image on its own, $p_\theta(x)$.

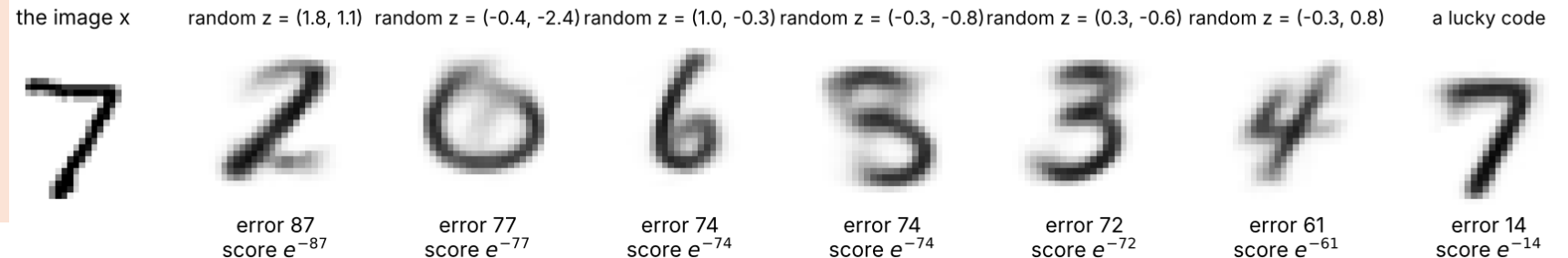
Why can't we just train it by maximum likelihood? ($p(x)$ is intractable)

We don't know which code produced the image, so we average over all the codes the prior could pick. This average is called the **marginal likelihood**, or **evidence**.

$$p_\theta(x) = \int p_\theta(x | z) p(z) dz = \mathbb{E}_{z \sim \mathcal{N}(0, I)} [p_\theta(x | z)]$$

1. draw S random codes from the priors
2. decode them
3. score how well the results matches x
4. average the scores

$p(x)$ = average score over random codes $z \sim \mathcal{N}(0, I)$. Each code is decoded, compared with x , and scored $e^{-\|x - f_D(z)\|^2}$



$$\hat{p}_S(x) = \frac{1}{S} \sum_{s=1}^S p_\theta(x | z_s), \quad z_s \sim \mathcal{N}(0, I)$$

$$\log \hat{p}_S(x) = \log \sum_{s=1}^S \exp(-\|x - f_D(z_s)\|_2^2) - \log S - \frac{M}{2} \log \pi$$

The marginal likelihood $p(\mathbf{x})$

How likely is an image?

We have been dealing with “how likely an image is if we know its code: $p_\theta(x|z)$ ”.

But a training set contains images and no codes.

To train by maximum likelihood, we need the probability of each image on its own, $p_\theta(x)$.

Why can't we just train it by maximum likelihood? ($p(x)$ is intractable)

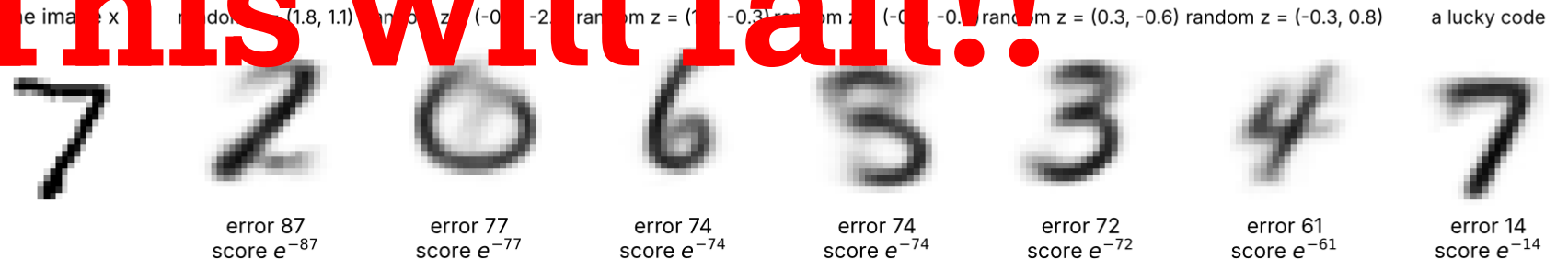
We don't know which code produced the image, so we average over all the codes the prior could pick. This average is called the **marginal likelihood**, or **evidence**.

$$p_\theta(x) = \int p_\theta(x | z) p(z) dz = \mathbb{E}_{z \sim \mathcal{N}(0, I)} [p_\theta(x | z)]$$

1. draw S random codes from the priors
2. decode them
3. score how well the results matches x
4. average the scores

$p(x)$ = average score over random codes $z \sim \mathcal{N}(0, I)$. Each code is decoded, compared with x , and scored $e^{-\|x - f_D(z)\|^2}$

This will fail!!



$$\hat{p}_S(x) = \frac{1}{S} \sum_{s=1}^S p_\theta(x | z_s), \quad z_s \sim \mathcal{N}(0, I)$$

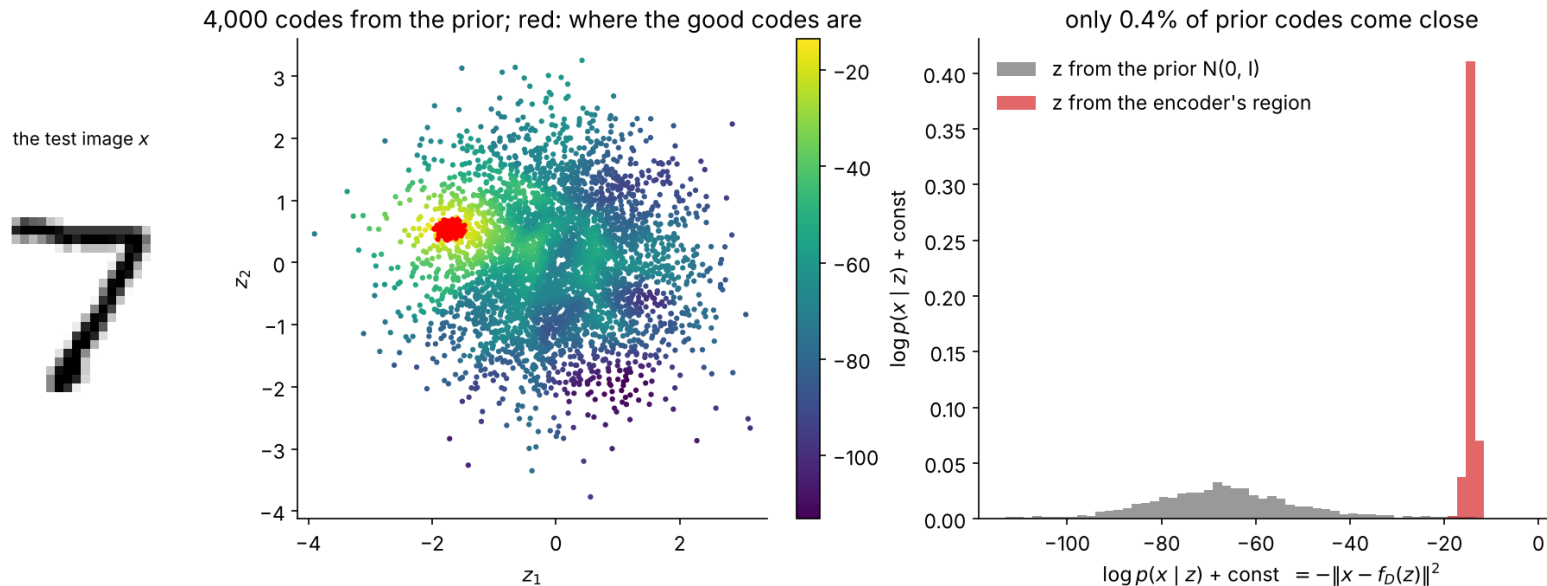
$$\log \hat{p}_S(x) = \log \sum_{s=1}^S \exp(-\|x - f_D(z_s)\|_2^2) - \log S - \frac{M}{2} \log \pi$$

How likely is an image?

Why this fails. $p(x)$ is **intractable**: it cannot be computed exactly, and sampling from the prior is too inefficient to estimate it. These are two distinct problems:

1. **No closed form.** The integrand $p(x|z)p(z)$ contains $\exp(-\|x - f_D(z)\|^2/2\tau^2)$, and f_D is a neural network. Integrating this over all of $\mathbb{R}^K$ has no analytic solution.

Numerical integration on a grid needs a number of points that grows exponentially with K .



2. **High-variance Monte Carlo.** The likelihood $p(x|z)$ is sharply peaked. It is non-negligible only for codes in a small region around the posterior $p(z|x)$, and that region holds a tiny fraction of the prior's mass

Inference: the Posterior and the Encoder

Given an image, which codes explain it? (Posterior, variational inference, amortized encoder, the gaps)

The inference problem: the posterior

The numerator is easy to
evaluate at any z

Given an image $x^{(i)}$, which codes z could have produced it? This is called **Inference**

The answer is a distribution, the **posterior** $p_{\theta}(z|x^{(i)})$.

The exact posterior cannot be computed;
it is **intractable**.

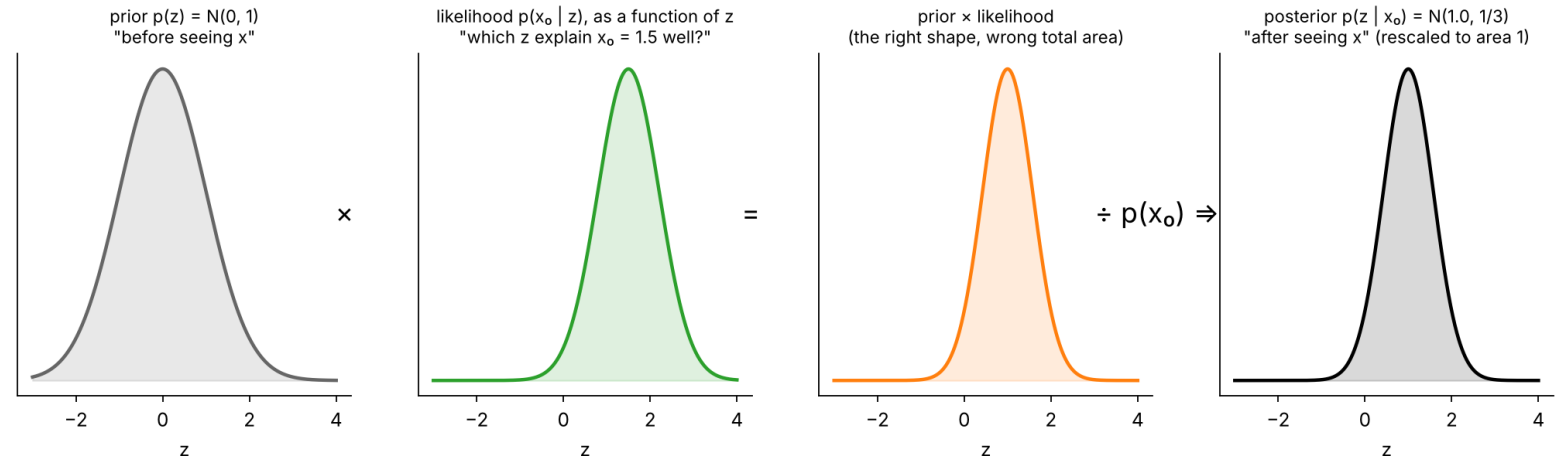
$$p(z | x^{(i)}) = \frac{\mathcal{N}(x^{(i)}; f_D(z; \theta), \tau^2 I) \mathcal{N}(z; 0, I)}{p(x^{(i)})}$$

intractable marginal likelihood,
so the posterior is intractable too

So we follow the standard route of **variational inference**: approximate it with the closest member of a simple family, measuring closeness with the KL divergence.

The VAE adds one more idea:
amortized inference:

A single network, the **encoder**, produces the approximation for every image



Approximating the posterior with a Gaussian

- **Variational inference** turns the problem around: Pick a family of simple distributions and find the member closest to the posterior.
- The VAE uses diagonal Gaussians.

- **Add noise**

Keep the encoder's code, now called

$$z_{original} = f_E(x^{(i)}; \varphi),$$

but decode a noisy copy of it,

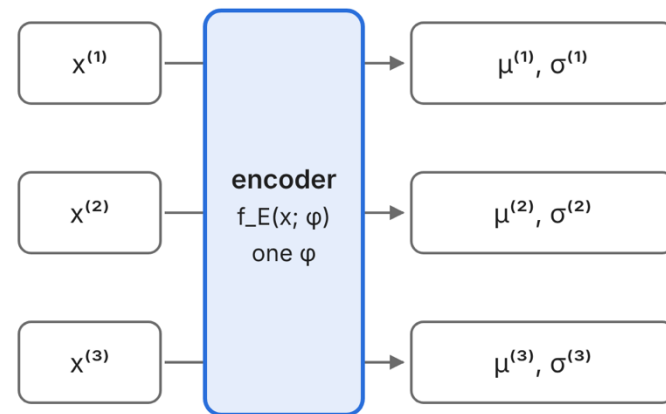
$$z_{noisy} = z_{original} + \varepsilon.$$

The decoder must now work for a whole neighborhood of codes.

- **Learn how much noise**

Let the encoder also output the covariance of the noise, $\varepsilon \sim N(0, \Sigma^{(i)})$, so each image gets the noise that suits it.

Amortized inference (VAE)



$(\mu^{(i)}, \sigma^{(i)}) = f_E(x^{(i)}; \varphi)$ for every image; a new image needs one forward pass

draw a code from the encoder's distribution

$$z \sim q_\phi(z \mid x^{(i)}) = \mathcal{N}(\mu^{(i)}, \Sigma^{(i)}) \iff$$

take the encoder's code $z_{original} = \mu^{(i)}$ and add noise ε

$$z = \underbrace{\mu^{(i)}}_{z_{original}} + \underbrace{\varepsilon}_{\text{added noise}}, \quad \varepsilon \sim \mathcal{N}(0, \Sigma^{(i)})$$

both are
 z_{noisy}

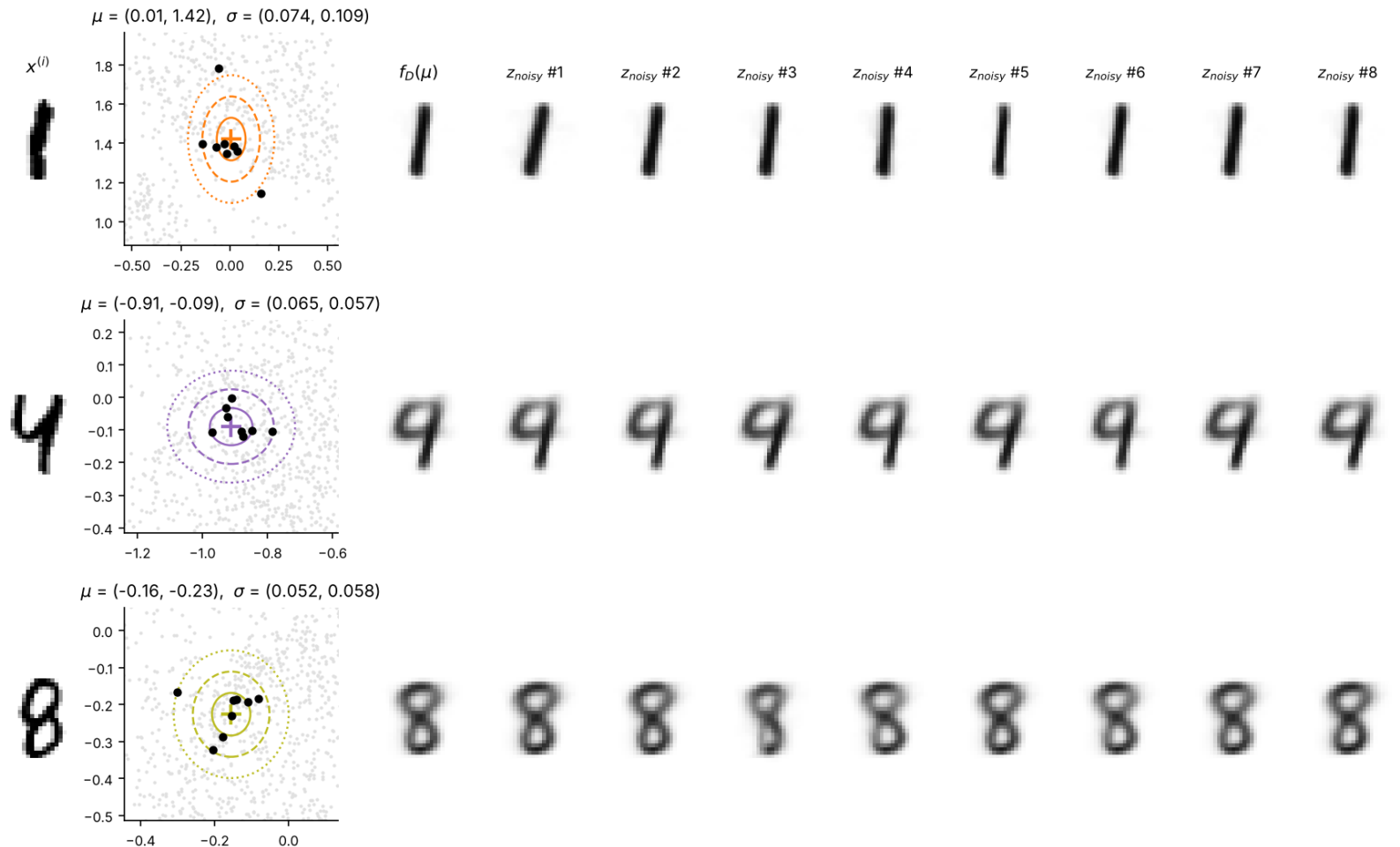
Amortized inference: one encoder for all images

What does $q_\phi(z|x^{(i)})$ look like for real images?

Three test images through the trained encoder and decoder

For each one we see $\mu^{(i)}, \sigma^{(i)}$, the cloud $N(\mu^{(i)}, \Sigma^{(i)})$, eight noisy codes drawn from it, and what each of them decodes to.

The encoder outputs $\mu^{(i)}$ and $\Sigma^{(i)}$; noisy codes drawn from $\mathcal{N}(\mu^{(i)}, \Sigma^{(i)})$ decode to nearly the same image



Variational inference

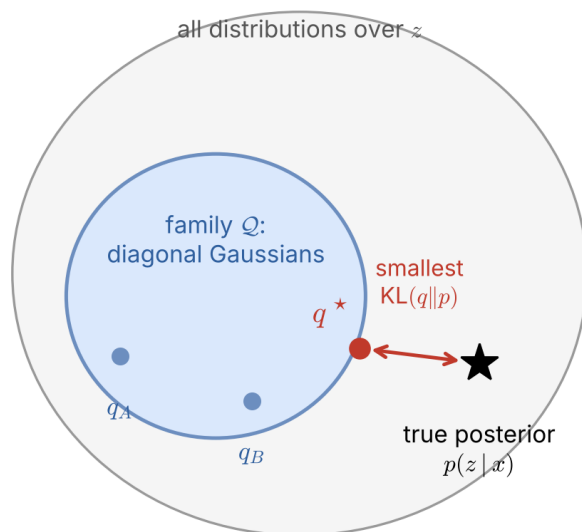
Variational inference (VI) replaces an intractable posterior with the closest member of a family Q of tractable distributions. It requires three choices:

1. The family. The VAE uses Gaussians with a diagonal covariance, the **mean-field Gaussian** family:

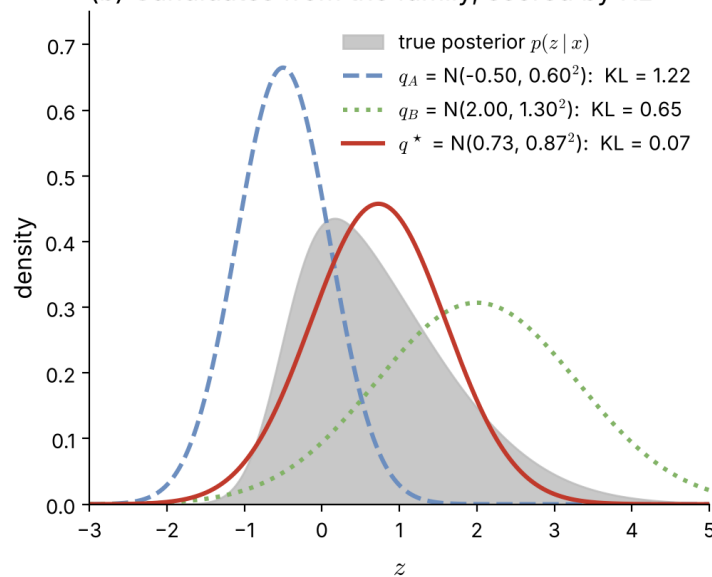
2. The measure of closeness. VI uses the **Kullback–Leibler (KL) divergence**

3. The optimization problem. For one image, variational inference solves

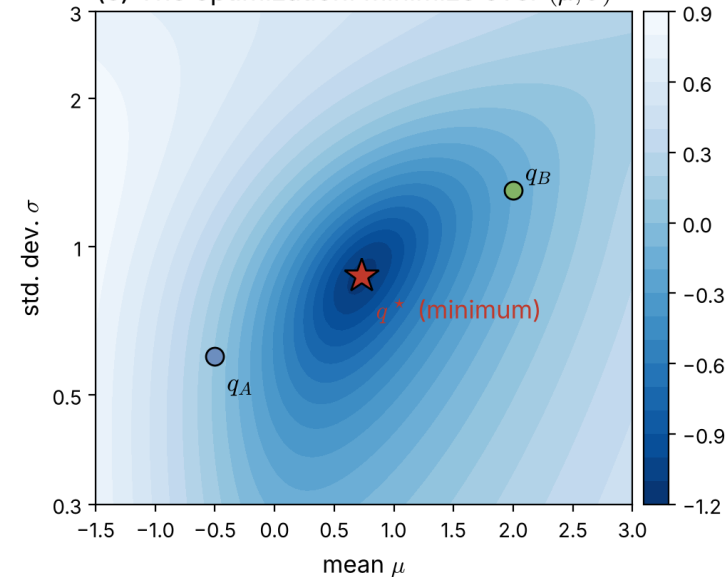
(a) Variational inference as a search



(b) Candidates from the family, scored by KL



(c) The optimization: minimize over (μ, σ)



Variational inference

Variational inference (VI) replaces an intractable posterior with the closest member of a family Q of tractable distributions. It requires three choices:

1. The family. The VAE uses Gaussians with a diagonal covariance, the **mean-field Gaussian** family:

$$q(z) = \mathcal{N}(z; \mu, \text{diag}(\sigma_1^2, \dots, \sigma_K^2)) = \prod_{k=1}^K \mathcal{N}(z_k; \mu_k, \sigma_k^2)$$

Each member is described by $2K$ numbers: the mean μ and the standard deviations σ .

Example with $K = 2$: $N(\mu = (-0.74, -1.95), \sigma = (0.085, 0.141))$ is one member, described by 4 numbers. These are also exactly the 4 outputs of the encoder. Searching the family means searching over these $2K$ numbers.

A diagonal covariance makes the coordinates of z independent under q . That keeps every computation cheap

Variational inference

Variational inference (VI) replaces an intractable posterior with the closest member of a family $\mathcal{Q}$ of tractable distributions. It requires three choices:

1. The family. The VAE uses Gaussians with a diagonal covariance, the **mean-field Gaussian** family:

$$q(z) = \mathcal{N}(z; \mu, \text{diag}(\sigma_1^2, \dots, \sigma_K^2)) = \prod_{k=1}^K \mathcal{N}(z_k; \mu_k, \sigma_k^2)$$

2. The measure of closeness. VI uses the **Kullback–Leibler (KL) divergence**

$$D_{KL}(Q \parallel P) = \mathbb{E}_{z \sim Q} [\log Q(z) - \log P(z)] = H(Q, P) - H(Q)$$

It is zero only when $Q = P$, positive otherwise, and not symmetric.

VI uses the **reverse KL**, $D_{KL}(q \parallel p)$, with the approximation first.

Its expectation is over q , which we can sample from.

The forward KL, $D_{KL}(p \parallel q)$, would need samples from the posterior, which is exactly what we cannot compute

Variational inference

Classical VI solves this problem separately for each image: it finds one $\mu^{(i)}$ and one $\sigma^{(i)}$ per image, each by its own optimization.

Variational inference (VI) replaces an intractable posterior with the closest member of a family $\mathcal{Q}$ of tractable distributions. It requires three choices:

1. The family. The VAE uses Gaussians with a diagonal covariance, the **mean-field Gaussian** family:

$$q(z) = \mathcal{N}(z; \mu, \text{diag}(\sigma_1^2, \dots, \sigma_K^2)) = \prod_{k=1}^K \mathcal{N}(z_k; \mu_k, \sigma_k^2)$$

2. The measure of closeness. VI uses the **Kullback–Leibler (KL) divergence**

$$D_{KL}(Q \parallel P) = \mathbb{E}_{z \sim Q} [\log Q(z) - \log P(z)] = H(Q, P) - H(Q)$$

3. The optimization problem. For one image, variational inference solves

$$q^* = \arg \min_{q \in \mathcal{Q}} D_{KL}(q(z) \parallel p_{\theta}(z \mid x^{(i)}))$$

$$D_{KL}(q \parallel p_{\theta}(z \mid x^{(i)})) = \mathbb{E}_{z \sim q} [\log q(z) - \log p_{\theta}(x^{(i)}, z)] + \log p_{\theta}(x^{(i)})$$

Amortized inference: the encoder $q_{\phi}(z|x)$

Classical VI needs 60,000 optimizations for MNIST, and a new one for every new image.

Amortized inference replaces them with a single network, the **encoder** or **inference network**. It maps any image to the parameters of its approximate posterior in one forward pass.

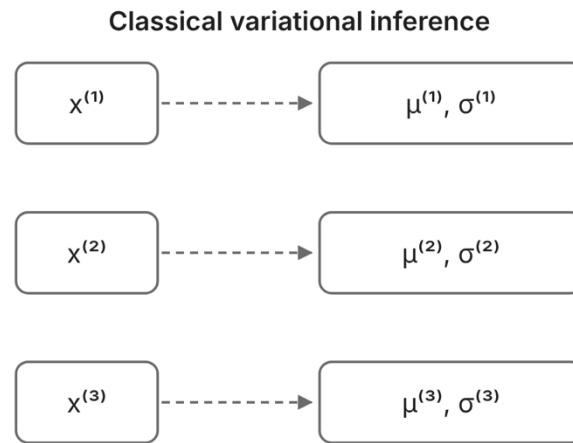
The encoder then defines, for each image

$$q_{\phi}(z | x^{(i)}) = \mathcal{N}(z; \mu^{(i)}, \Sigma^{(i)})$$

$$(\mu^{(i)}, \Sigma^{(i)}) = f_E(x^{(i)}; \phi)$$

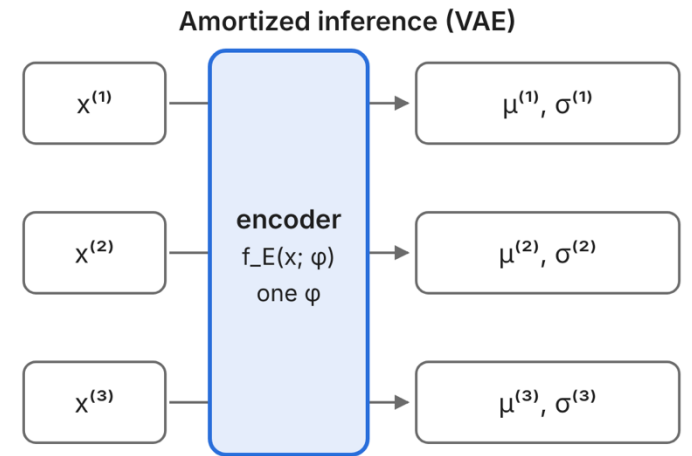
$$\Sigma^{(i)} = \text{diag}(\sigma_1^{(i)2}, \dots, \sigma_K^{(i)2})$$

Amortized inference: one encoder $f_E(x; \phi)$ replaces a separate fit per image



a separate optimization
for every image
(parameters grow with N)

Classical VI: every image has its own parameters, found by its own optimization



$(\mu^{(i)}, \sigma^{(i)}) = f_E(x^{(i)}; \phi)$ for every image; a new image needs one forward pass

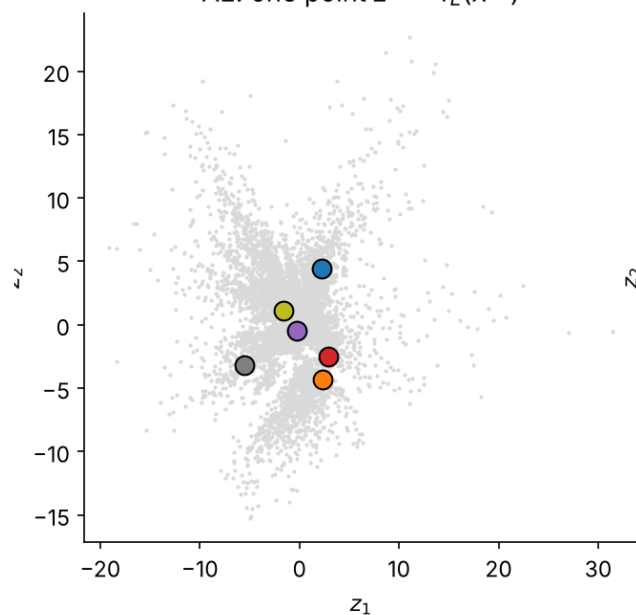
Amortized Inference: One encoder computes them for any image. In practice it outputs $\mu^{(i)}$ and $\log \sigma^{(i)2}$ (the log keeps the variance positive), so with $K = 2$ it has 4 outputs

A point versus a distribution

This is the key difference from an autoencoder. The autoencoder maps each image to a point; the VAE encoder maps it to a distribution, a small cloud of plausible codes.

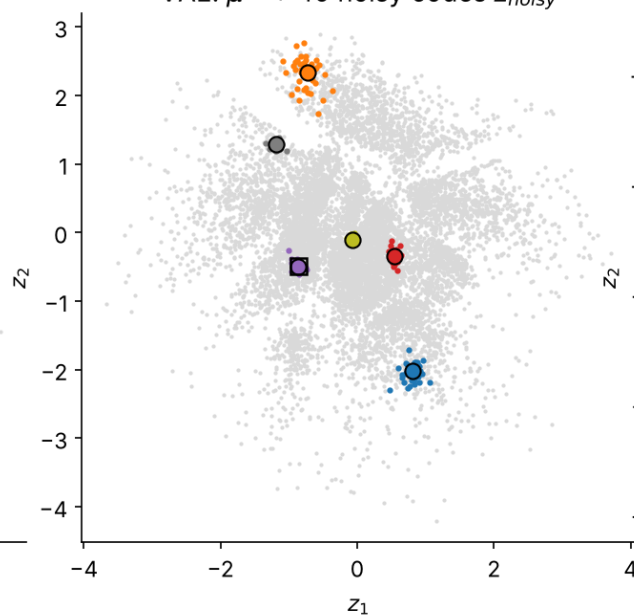


AE: one point $z^{(i)} = f_E(x^{(i)})$



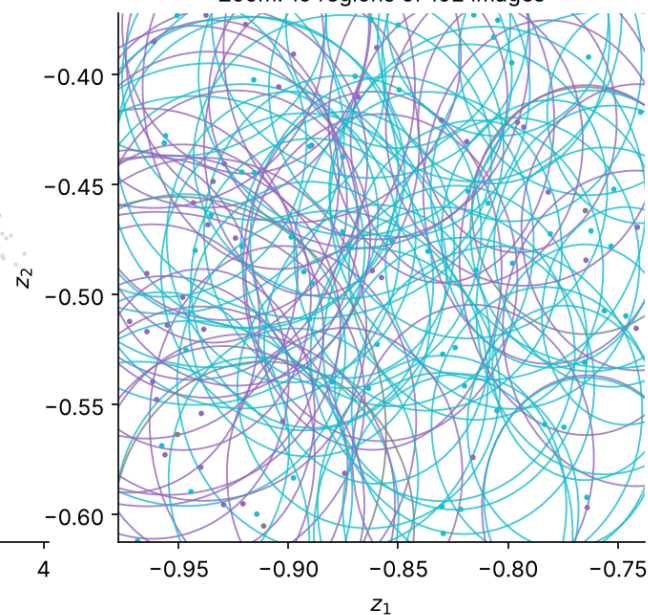
The autoencoder gives each image one point, on an arbitrary scale from -20 to 30

VAE: $\mu^{(i)} + 40$ noisy codes z_{noisy}



The VAE encoder maps each image to a **distribution**, a small cloud of plausible codes on the prior's scale.

Zoom: 1σ regions of 102 images



Zooming into a 0.24×0.24 box around the 4 reveals the 1σ ellipses of 102 test image

Each is small, but together they overlap and tile the box. Because the decoder is trained on noisy codes from all of them, it learns to decode every point of the box: this is how the VAE fills the hole

The effect of “add noise”

Take the code of a 2 and push it a little in some direction.
What does the autoencoder decode? And the VAE?

Move the code of a 2 in a fixed direction by a fraction of the codes' spread, then decode

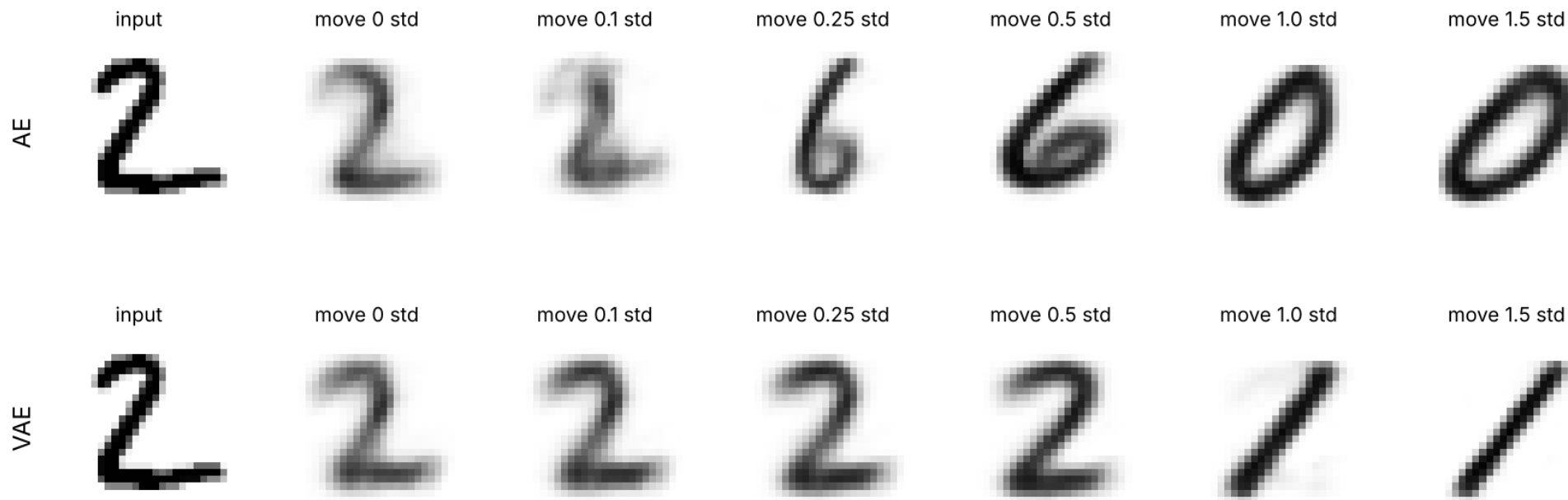


The effect of “add noise”

Take the code of a 2 and push it a little in some direction. What does the autoencoder decode? And the VAE?

The VAE keeps a clean 2 up to half a standard deviation, then turns smoothly into a 1. The VAE's decoder was trained on noisy codes around every training image, so the neighborhood of each code decodes to sensible images; The autoencoder's decoder only ever saw exact codes.

Move the code of a 2 in a fixed direction by a fraction of the codes' spread, then decode

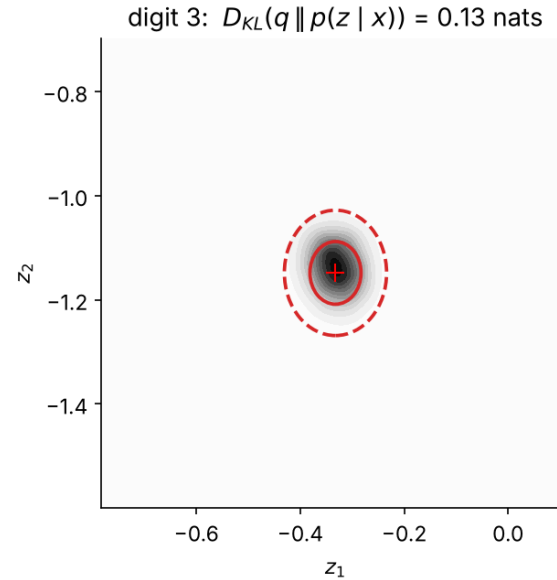


How good is the approximation?

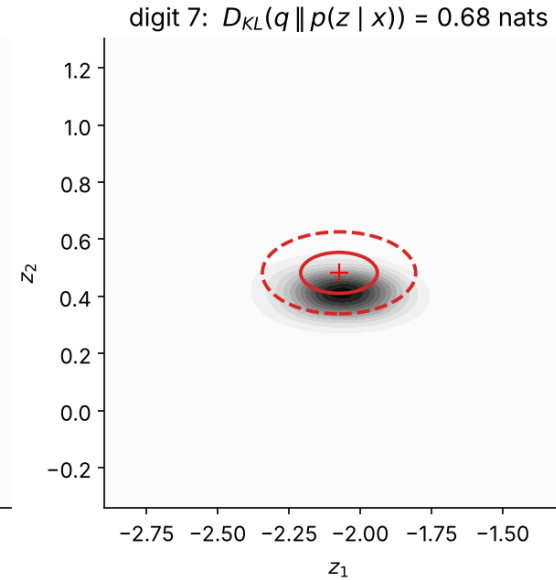
With $K = 2$ we can compute the true posterior on a grid.

The numerator of Bayes' rule is $N(x^{(i)}; f_D(z), \tau^2 I) N(z; 0, I)$, which we can evaluate at every grid point.

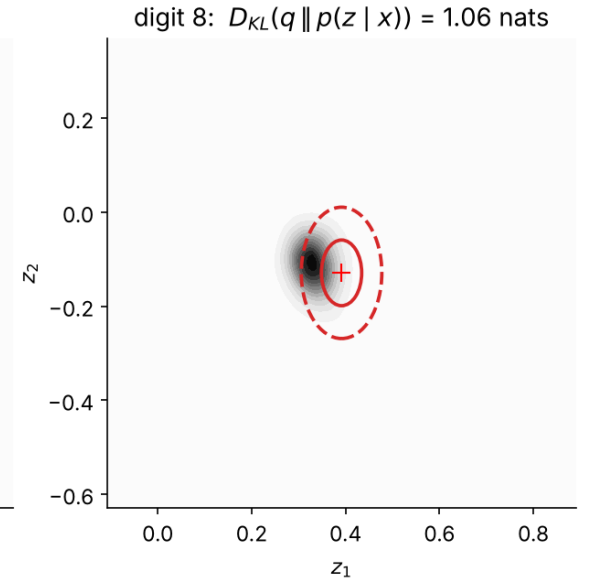
Compares that true posterior with the encoder's Gaussian for three test images



the encoder's Gaussian sits on the posterior and the KL is only 0.13 nats.



The true posterior is tilted, which a diagonal Σ cannot express: KL 0.68



the posterior is offset and curved: KL 1.06

$$\begin{aligned} D_{KL}(Q \| P) &= \mathbb{E}_{z \sim Q} [\log Q(z) - \log P(z)] \\ &= H(Q, P) - H(Q), \end{aligned}$$

The ELBO and The objective $J(\theta, \varphi)$

If we can't maximize $\log p(x)$, what do we maximize instead?
(ELBO, the bound and its gap, reconstruction + KL, J)

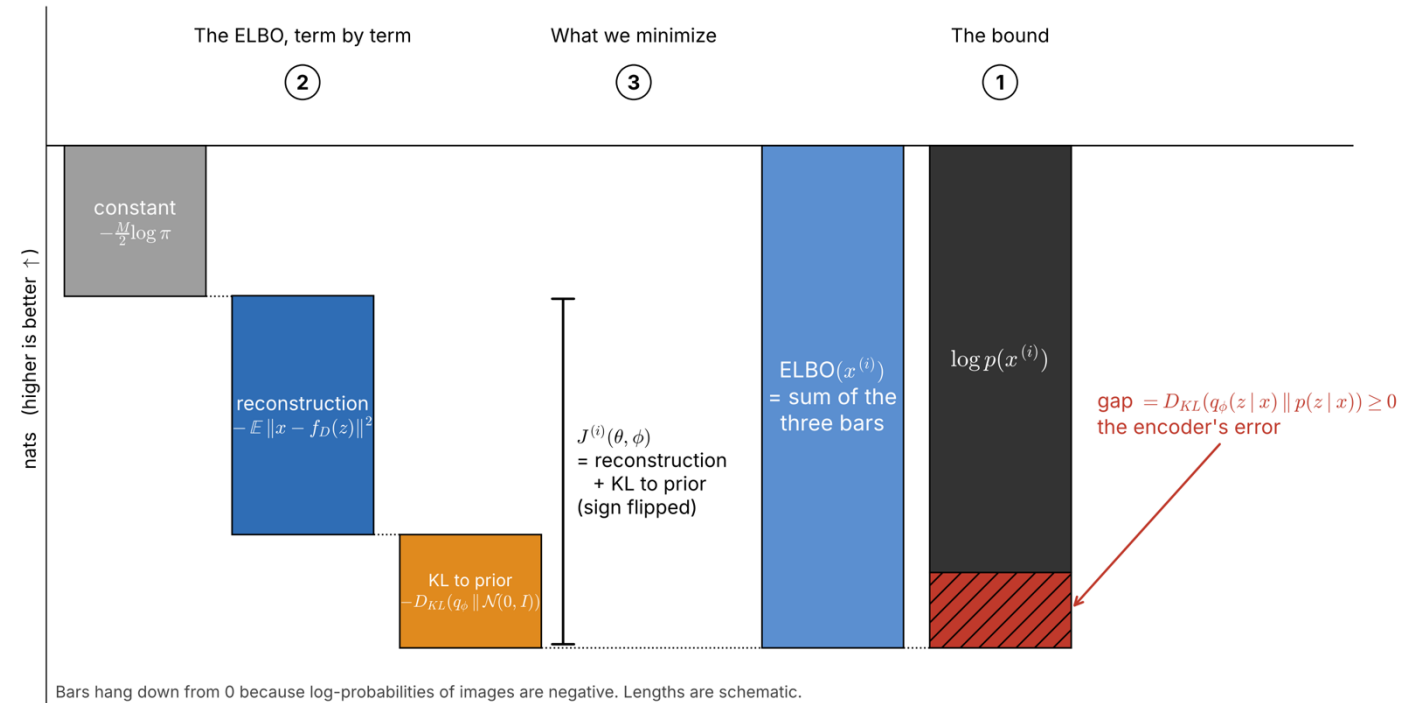
The goal and the obstacle

Training should maximize $\sum \log p_{\theta}(x^{(i)})$, the maximum-likelihood

We cannot compute $\log p(x^{(i)})$
 $p(x)$ is an integral with no closed form, and sampling from the prior is too inefficient

Can we optimize something we can compute?

We can compute a number that is always below it, the **evidence lower bound (ELBO)**. The gap is exactly how far the encoder is from the true posterior.



Equation 1, the bound The ELBO falls short of the log-likelihood by exactly the KL from the encoder to the true posterior. A KL is never negative, so the ELBO is a lower bound:

$$\log p(x^{(i)}) = \text{ELBO}(x^{(i)}) + D_{KL}(q_{\phi}(z | x^{(i)}) \parallel p(z | x^{(i)})) \geq \text{ELBO}(x^{(i)})$$

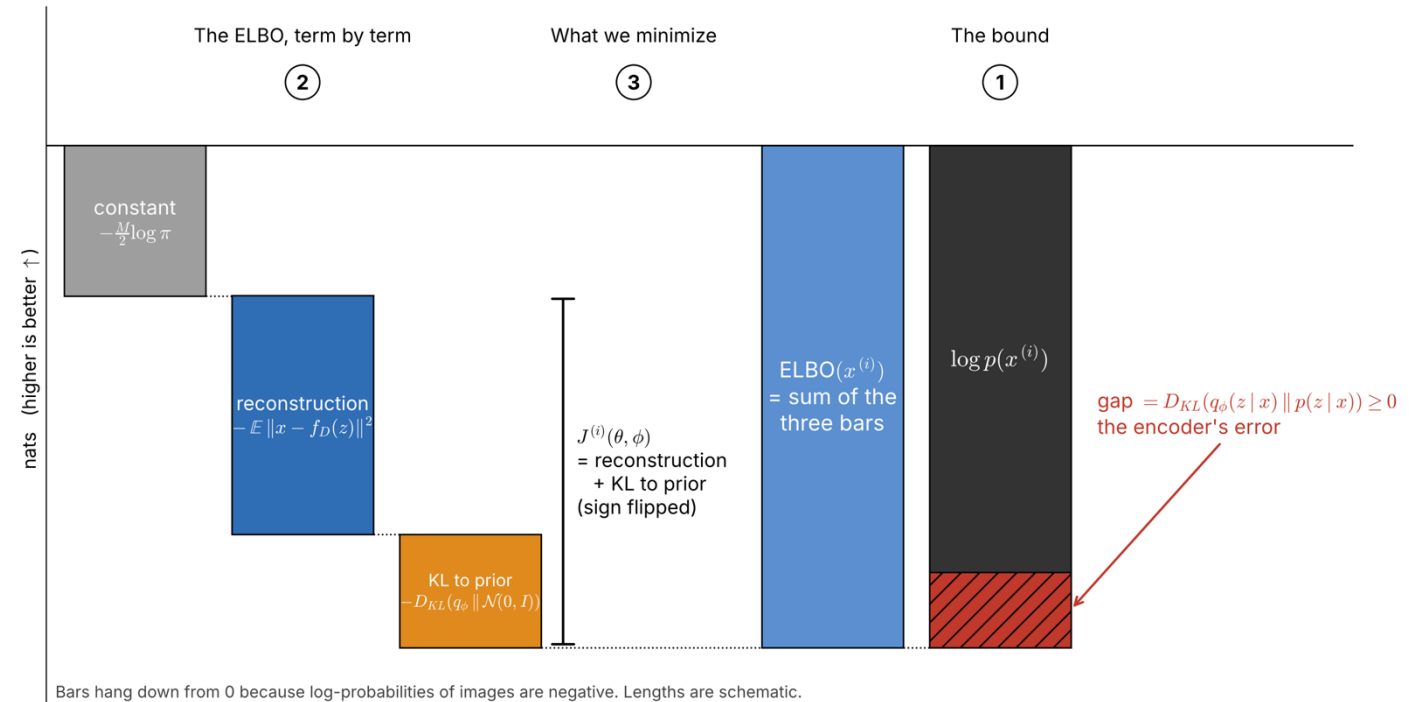
The goal and the obstacle

Training should maximize $\sum \log p_{\theta}(x^{(i)})$, the maximum-likelihood

We cannot compute $\log p(x^{(i)})$
 $p(x)$ is an integral with no closed form, and sampling from the prior is too inefficient

Can we optimize something we can compute?

We can compute a number that is always below it, the **evidence lower bound (ELBO)**. The gap is exactly how far the encoder is from the true posterior.



Equation 2, the ELBO term by term: With Gaussian pixels and $\tau^2 = 0.5$, the ELBO is minus a squared reconstruction error, minus a KL to the prior, plus a constant that depends on neither network:

$$\text{ELBO}(x^{(i)}) = -\mathbb{E}_{z \sim q_{\phi}(z|x^{(i)})} \|x^{(i)} - f_D(z; \theta)\|_2^2 - D_{KL}(q_{\phi}(z | x^{(i)}) \| \mathcal{N}(0, I)) - \frac{M}{2} \log \pi$$

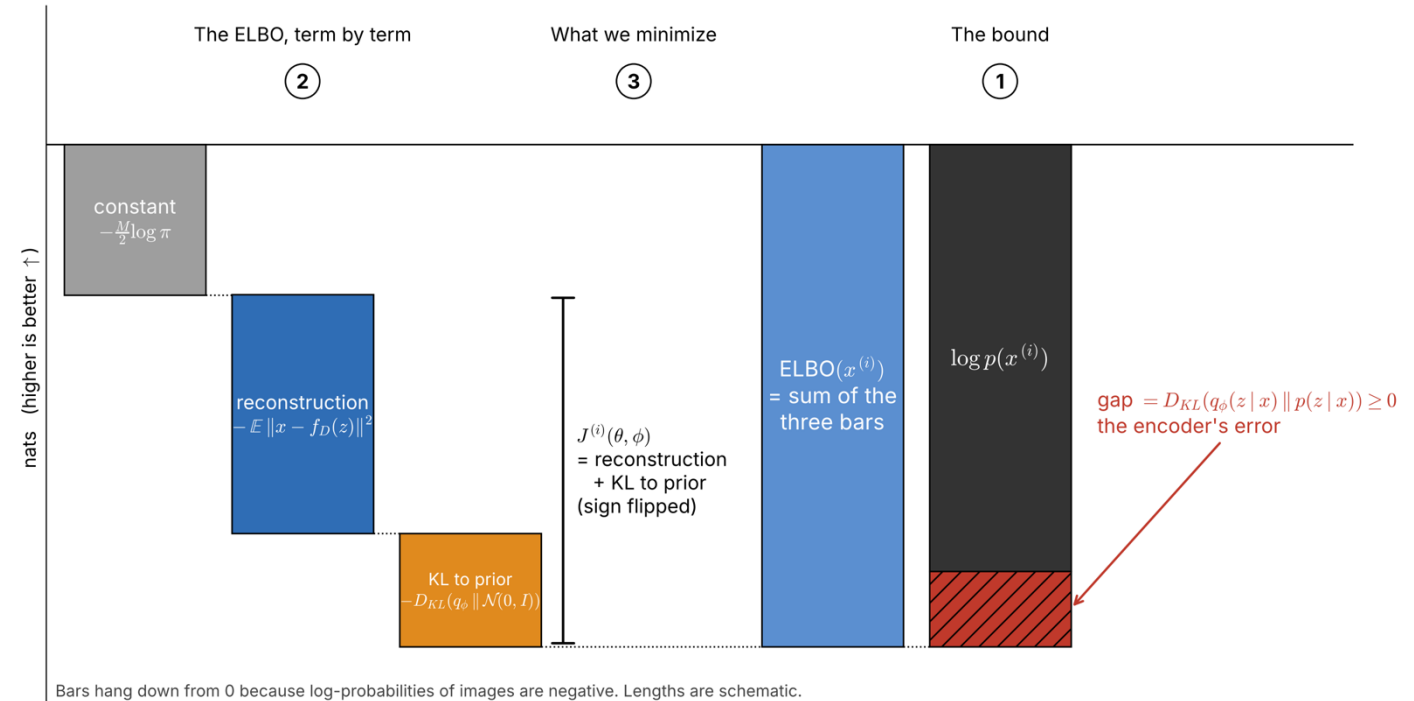
The goal and the obstacle

Training should maximize $\sum \log p_{\theta}(x^{(i)})$, the maximum-likelihood

We cannot compute $\log p(x^{(i)})$
 $p(x)$ is an integral with no closed form, and sampling from the prior is too inefficient

Can we optimize something we can compute?

We can compute a number that is always below it, the **evidence lower bound (ELBO)**. The gap is exactly how far the encoder is from the true posterior.



Equation 3, what we minimize: Keep the two terms that depend on θ and ϕ , flip the sign and sum over the training set. **So minimizing J means maximizing the total ELBO:**

$$J(\theta, \phi) = \sum_{i=1}^N \left[\mathbb{E}_z \left\| x^{(i)} - f_D(z; \theta) \right\|_2^2 + D_{KL}(q_{\phi}(z | x^{(i)}) \| \mathcal{N}(0, I)) \right]$$

$$= - \sum_{i=1}^N \text{ELBO}(x^{(i)}) + \text{const}$$

The ELBO: A lower bound we can compute

Importance sampling and Jensen's inequality

- Start from the integral we cannot compute and multiply and divide by the encoder's distribution $q_\phi(z|x^{(i)}) = N(\mu^{(i)}, \Sigma^{(i)})$
- Nothing changes, but the integral becomes an average over noisy codes:

$$p(x^{(i)}) = \int q_\phi(z | x^{(i)}) \frac{p(x^{(i)}, z)}{q_\phi(z | x^{(i)})} dz = \mathbb{E}_{z \sim q_\phi(z|x^{(i)})} \left[\frac{p(x^{(i)}, z)}{q_\phi(z|x^{(i)})} \right]$$

This is **importance sampling**:

- Instead of drawing codes from the prior (which fails), draw them from the encoder, which knows where the good codes are, and reweight. Then take the log.

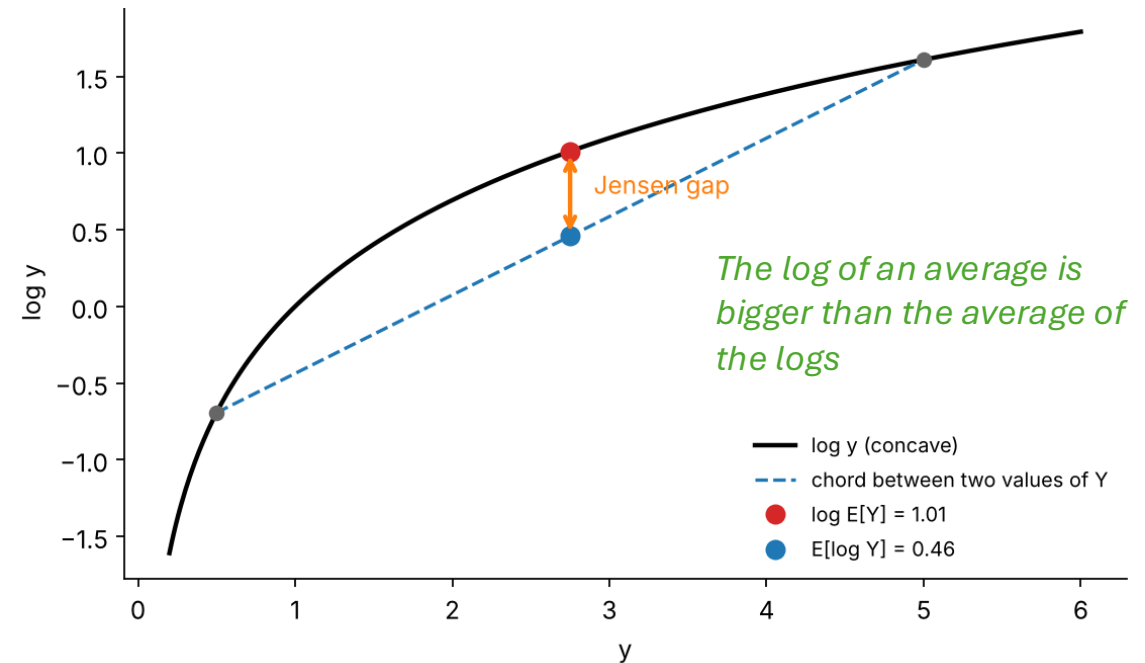
$$\log p(x^{(i)}) \geq \mathbb{E}_{z \sim q_\phi} \left[\log \frac{p(x^{(i)}, z)}{q_\phi(z|x^{(i)})} \right] =: \text{ELBO}(x^{(i)}; \theta, \phi)$$

Jensen, concave log

$$\mathbb{E}[\log Y] \leq \log \mathbb{E}[Y]$$

the VAE: $Y = p(x, z)/q_\phi(z | x)$

$$\underbrace{\mathbb{E}_{z \sim q_\phi} \left[\log \frac{p(x^{(i)}, z)}{q_\phi(z | x^{(i)})} \right]}_{\text{ELBO}(x^{(i)})} \leq \underbrace{\log \mathbb{E}_{z \sim q_\phi} \left[\frac{p(x^{(i)}, z)}{q_\phi(z | x^{(i)})} \right]}_{\log p(x^{(i)})}$$



The ELBO: A lower bound we can compute

Importance sampling and Jensen's inequality

- Start from the integral we cannot compute and multiply and divide by the encoder's distribution $q_\phi(z|x^{(i)}) = N(\mu^{(i)}, \Sigma^{(i)})$
$$p(x^{(i)}) = \int q_\phi(z | x^{(i)}) \frac{p(x^{(i)}, z)}{q_\phi(z|x^{(i)})} dz = \mathbb{E}_{z \sim q_\phi(z|x^{(i)})} \left[\frac{p(x^{(i)}, z)}{q_\phi(z|x^{(i)})} \right]$$
- Nothing changes, but the integral becomes an average over noisy codes:

This is **importance sampling**:

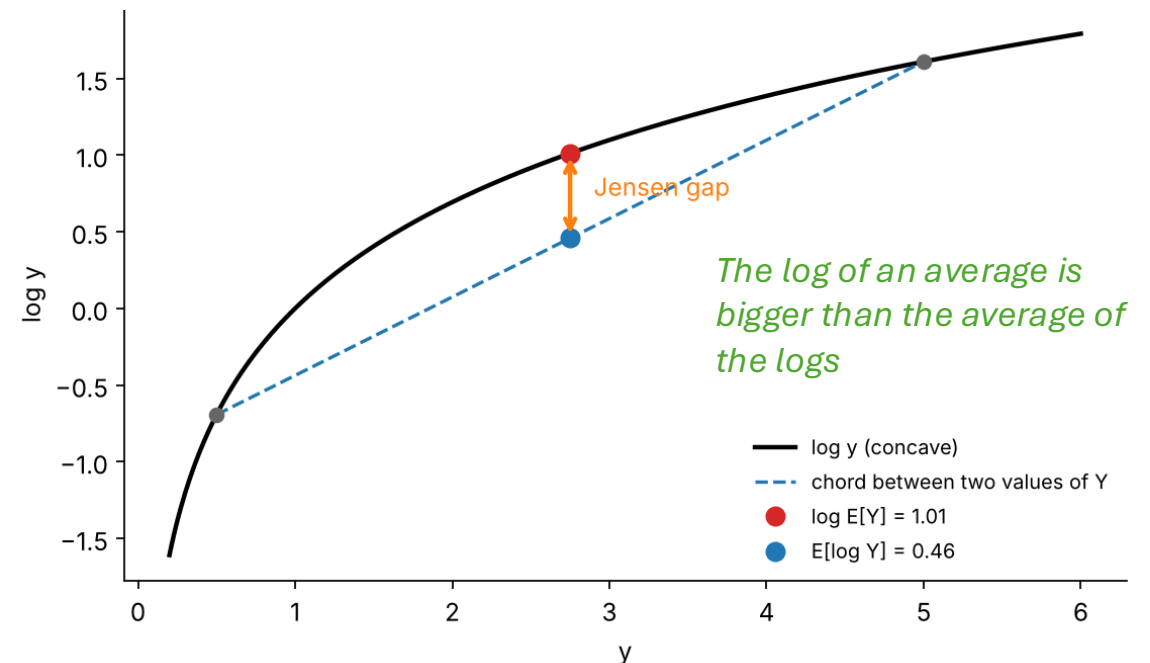
- Instead of drawing codes from the prior (which fails), draw them from the encoder, which knows where the good codes are, and reweight. Then take the log.

$$\log p(x^{(i)}) \geq \mathbb{E}_{z \sim q_\phi} \left[\log \frac{p(x^{(i)}, z)}{q_\phi(z|x^{(i)})} \right] =: \text{ELBO}(x^{(i)}; \theta, \phi)$$

In the VAE, Y is the weight $p(x, z)/q_\phi(z|x)$ for a code z drawn from the encoder.

- $\log(\text{average of } Y) = \log p(x)$, which we cannot compute.
- average of $\log Y$ = the ELBO, which we can estimate from one noisy code.

So the red-minus-blue gap is exactly the gap between $\log p(x)$ and the ELBO. It is zero when Y is the same for every code, which happens when the encoder equals the true posterior.



The ELBO: A lower bound we can compute

Importance sampling and Jensen's inequality

- Start from the integral we cannot compute and multiply and divide by the encoder's distribution $q_\phi(z|x^{(i)})$
 $= N(\mu^{(i)}, \Sigma^{(i)})$
- Nothing changes, but the integral becomes an average over noisy codes:

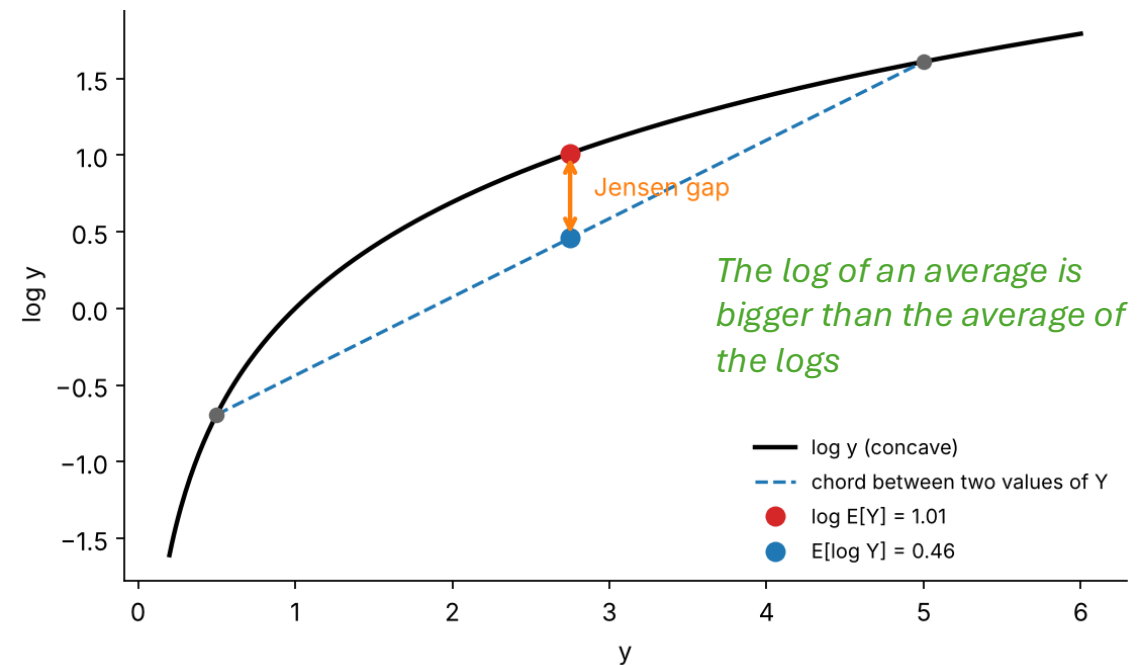
$$p(x^{(i)}) = \int q_\phi(z | x^{(i)}) \frac{p(x^{(i)}, z)}{q_\phi(z | x^{(i)})} dz = \mathbb{E}_{z \sim q_\phi(z | x^{(i)})} \left[\frac{p(x^{(i)}, z)}{q_\phi(z | x^{(i)})} \right]$$

This is **importance sampling**:

- Instead of drawing codes from the prior (**which fails**), draw them from the encoder, which knows where the good codes are, and reweight. Then take the log.

$$\log p(x^{(i)}) \geq \mathbb{E}_{z \sim q_\phi} \left[\log \frac{p(x^{(i)}, z)}{q_\phi(z | x^{(i)})} \right] =: \text{ELBO}(x^{(i)}; \theta, \phi)$$

The **ELBO**: It is an average of something we can evaluate for any z , so a single noisy code gives an estimate of it.



How tight is the bound?

The gap is exactly a KL divergence

average, over z
drawn from q

q is the encoder's
 $q_\phi(z|x^{(i)})$

p is the true
posterior $p(z|x^{(i)})$

$$\begin{aligned}
 D_{KL}(q_\phi(z | x^{(i)}) || p(z | x^{(i)})) &= \mathbb{E}_{z \sim q_\phi} [\log q_\phi(z | x^{(i)}) - \log p(z | x^{(i)})] \\
 &= \mathbb{E}_{z \sim q_\phi} [\log q_\phi(z | x^{(i)}) - \log p(x^{(i)}, z) + \log p(x^{(i)})] \\
 &= \mathbb{E}_{z \sim q_\phi} [\log q_\phi(z | x^{(i)}) - \log p(x^{(i)}, z)] + \log p(x^{(i)}) \\
 &= -\mathbb{E}_{z \sim q_\phi} \left[\log \frac{p(x^{(i)}, z)}{q_\phi(z | x^{(i)})} \right] + \log p(x^{(i)}) \\
 &= -\text{ELBO}(x^{(i)}) + \log p(x^{(i)})
 \end{aligned}$$

definition of KL

Bayes' rule

$\log p(x^{(i)})$ does not depend on z

combine the logs

definition of the ELBO (Section 4.1)

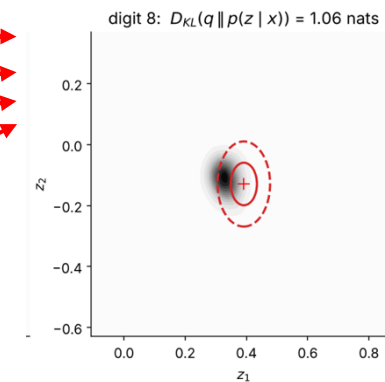
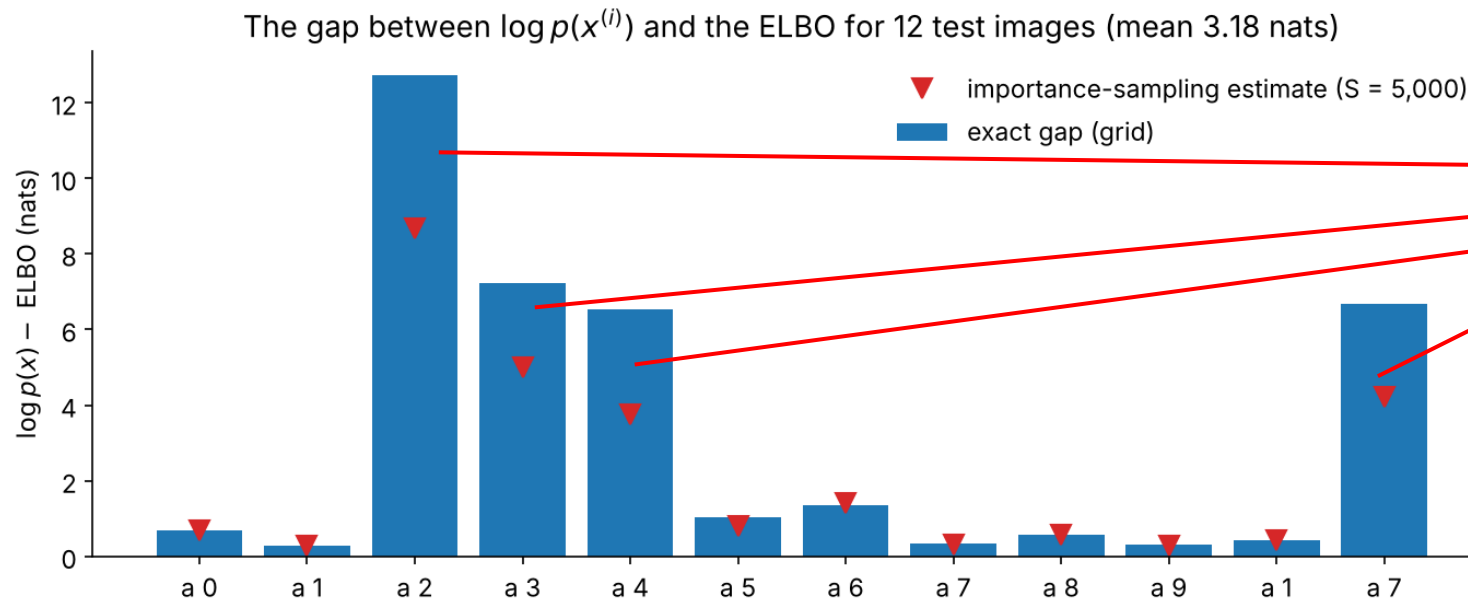
$$\log p(x^{(i)}) = \text{ELBO}(x^{(i)}; \theta, \phi) + D_{KL}(q_\phi(z | x^{(i)}) || p(z | x^{(i)}))$$

1. **It is a bound.** A KL is never negative, so the ELBO is never above $\log p(x^{(i)})$.
2. **The gap is the encoder's error.** The bound is tight exactly when $N(\mu^{(i)}, \Sigma^{(i)})$ equals the true posterior
3. **Training does two jobs at once.** $\log p(x^{(i)})$ does not depend on ϕ , so raising the ELBO in ϕ can only shrink the gap. Raising it in θ either raises the likelihood or tightens the bound.

How tight is the bound?

The gap is exactly a KL divergence

8/12 images the encoder is nearly perfect, with gaps of 0.3 to 1.3 nats



Images whose true posteriors are tilted or split, so a diagonal Gaussian cannot fit them well. That is the approximation and amortization gaps

$$\log p(x^{(i)}) = \text{ELBO}(x^{(i)}; \theta, \phi) + D_{KL}(q_{\phi}(z | x^{(i)}) \| p(z | x^{(i)}))$$

1. **It is a bound.** A KL is never negative, so the ELBO is never above $\log p(x^{(i)})$.
2. **The gap is the encoder's error.** The bound is tight exactly when $N(\mu^{(i)}, \Sigma^{(i)})$ equals the true posterior
3. **Training does two jobs at once.** $\log p(x^{(i)})$ does not depend on ϕ , so raising the ELBO in ϕ can only shrink the gap. Raising it in θ either raises the likelihood or tightens the bound.

What does the ELBO look like in terms of the network?

Opening up the ELBO: reconstruction minus regularization

We showed that the ELBO is the right thing to maximize: it is a lower bound on $\log p(x^{(i)})$, and it is tight when the encoder matches the posterior.

But in the form

$ELBO = E_q[\log p(x, z) - \log q(z|x)]$,
it is not yet something you can type into PyTorch.

How is this related to the networks f_E and f_D or the pixels of an image?

- **Split it into two interpretable terms:**
how well the decoder reconstructs the image, and how far the encoder's Gaussian strays from the prior.
- **Plug in the Gaussian pixel model:**
the reconstruction term becomes an ordinary squared error.

The result is exactly the objective $J(\theta, \varphi)$

$$\begin{aligned} ELBO(x^{(i)}) &= \mathbb{E}_q [\log p(x^{(i)}, z) - \log q(z)] \\ &= \mathbb{E}_q [\log p_\theta(x^{(i)} | z) + \log p(z) - \log q(z)] \\ &= \mathbb{E}_q [\log p_\theta(x^{(i)} | z)] - \mathbb{E}_q [\log q(z) - \log p(z)] \\ &= \mathbb{E}_q [\log p_\theta(x^{(i)} | z)] - D_{KL}(q \| \mathcal{N}(0, I)) \end{aligned}$$

What does the ELBO look like in terms of the network?

Opening up the ELBO: reconstruction minus regularization

$$\begin{aligned}\text{ELBO}(x^{(i)}) &= \mathbb{E}_q[\log p(x^{(i)}, z) - \log q(z)] \\ &= \mathbb{E}_q[\log p_\theta(x^{(i)} | z) + \log p(z) - \log q(z)] \\ &= \mathbb{E}_q[\log p_\theta(x^{(i)} | z)] - \mathbb{E}_q[\log q(z) - \log p(z)] \\ &= \mathbb{E}_q[\log p_\theta(x^{(i)} | z)] - D_{KL}(q \| \mathcal{N}(0, I))\end{aligned}$$

Opening up the ELBO:
reconstruction minus
regularization

(a) definition (Section 4.1)

(b) joint = decoder $\times$ prior

(c) split the expectation

(d) definition of KL

What does the ELBO look like in terms of the network?

Opening up the ELBO: reconstruction minus regularization

$$\begin{aligned}\text{ELBO}(x^{(i)}) &= \mathbb{E}_q[\log p(x^{(i)}, z) - \log q(z)] \\ &= \mathbb{E}_q[\log p_\theta(x^{(i)} | z) + \log p(z) - \log q(z)] \\ &= \mathbb{E}_q[\log p_\theta(x^{(i)} | z)] - \mathbb{E}_q[\log q(z) - \log p(z)] \\ &= \mathbb{E}_q[\log p_\theta(x^{(i)} | z)] - D_{KL}(q \| \mathcal{N}(0, I))\end{aligned}$$

$$\begin{aligned}\log p_\theta(x^{(i)} | z) &= \sum_{j=1}^M \log \mathcal{N}(x_j^{(i)}; f_D(z)_j, \tau^2) \\ &= \sum_{j=1}^M \left[-\frac{1}{2} \log(2\pi\tau^2) - \frac{(x_j^{(i)} - f_D(z)_j)^2}{2\tau^2} \right] \\ &= -\frac{\|x^{(i)} - f_D(z)\|_2^2}{2\tau^2} - \frac{M}{2} \log(2\pi\tau^2)\end{aligned}$$

$$\Rightarrow \text{ELBO}(x^{(i)}) = -\mathbb{E}_q\left[\frac{\|x^{(i)} - f_D(z)\|_2^2}{2\tau^2}\right] - D_{KL}(q \| \mathcal{N}(0, I)) - \frac{M}{2} \log(2\pi\tau^2)$$

Opening up the ELBO:
reconstruction minus
regularization

(a) definition (Section 4.1)

(b) joint = decoder $\times$ prior

(c) split the expectation

(d) definition of KL

Gaussian pixels turn
reconstruction into squared
error

(e) independent pixels

(f) log of a 1-D Gaussian

(g) sum over the M pixels

(h) put (g) into (d)

What does the ELBO look like in terms of the network?

From maximizing likelihood to minimizing $J(\theta, \phi)$

$$\max_{\theta} \sum_{i=1}^N \log p_{\theta}(x^{(i)})$$

(a) the goal: maximum likelihood (intractable)

$$\max_{\theta, \phi} \sum_{i=1}^N \text{ELBO}(x^{(i)})$$

(b) maximize the lower bound instead

$$\min_{\theta, \phi} - \sum_{i=1}^N \text{ELBO}(x^{(i)})$$

(c) maximizing f = minimizing $-f$

$$-\text{ELBO}(x^{(i)}) = \mathbb{E}_q \left[\frac{\|x^{(i)} - f_D(z)\|_2^2}{2\tau^2} \right] + D_{KL}(q \parallel \mathcal{N}(0, I)) + \frac{M}{2} \log(2\pi\tau^2)$$

(d) negate the ELBO above

$$= \mathbb{E}_q \|x^{(i)} - f_D(z)\|_2^2 + D_{KL}(q \parallel \mathcal{N}(0, I)) + \frac{M}{2} \log \pi$$

(e) set $\tau^2 = 0.5$, so $2\tau^2 = 1$

$$- \sum_{i=1}^N \text{ELBO}(x^{(i)}) = \underbrace{\sum_{i=1}^N \left[\mathbb{E}_q \|x^{(i)} - f_D(z)\|_2^2 + D_{KL}(q \parallel \mathcal{N}(0, I)) \right]}_{J(\theta, \phi)} + \underbrace{\frac{NM}{2} \log \pi}_{\text{const}}$$

(f) sum over the training set

$$\Rightarrow \arg \min_{\theta, \phi} \left(- \sum_i \text{ELBO} \right) = \arg \min_{\theta, \phi} J(\theta, \phi)$$

(g) a constant does not move the minimum

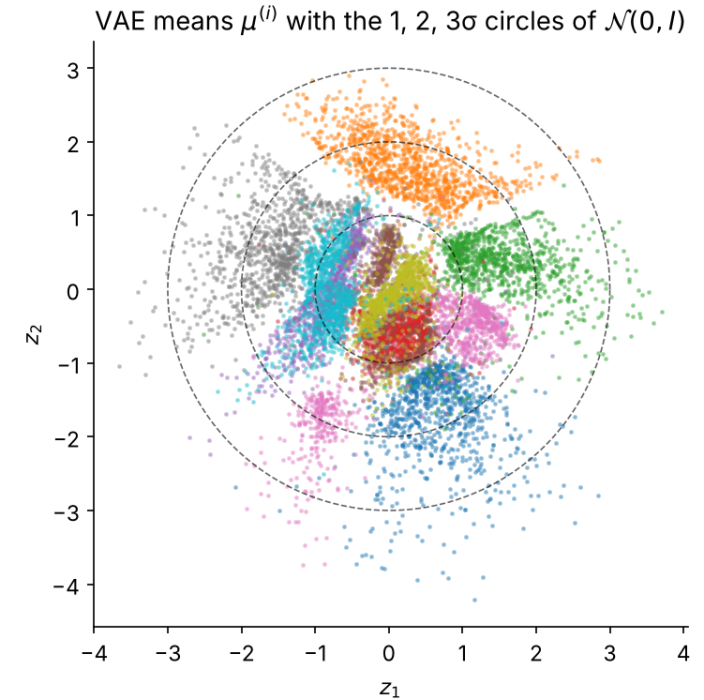
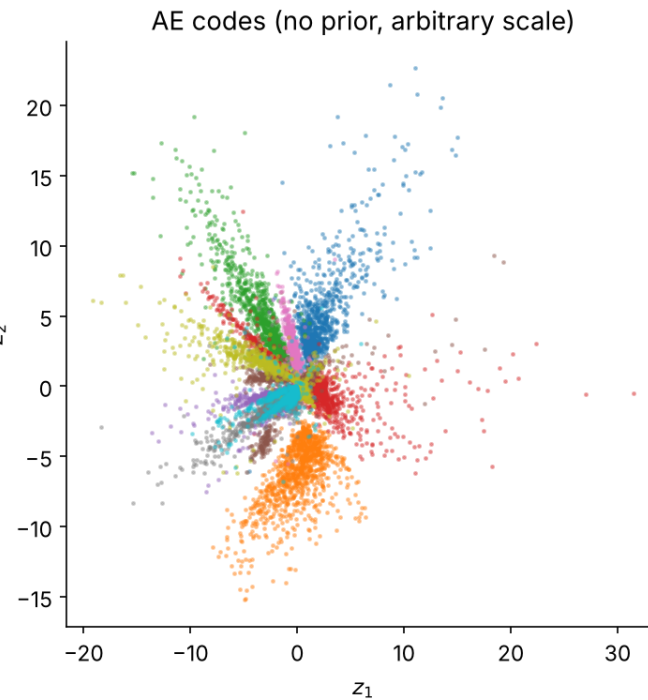
The KL term in closed form

Both distributions in the KL term are Gaussians, so it can be computed exactly, with no sampling.

For one dimension:

$$D_{KL}(\mathcal{N}(\mu, \sigma^2) \parallel \mathcal{N}(0, 1)) = \frac{1}{2}(\mu^2 + \sigma^2 - 1 - \log \sigma^2)$$

For a diagonal $\Sigma^{(i)}$ the K dimensions are independent, so the KL term is this formula summed over $k = 1, \dots, K$



Training: noisy codes and the reparameterization trick

How do we compute gradients through random codes?
(Monte Carlo, reparameterization)

Monte Carlo Estimation

$$\mathbb{E}_{z \sim \mathcal{N}(\mu^{(i)}, \Sigma^{(i)})} \left[\left\| x^{(i)} - f_D(z; \theta) \right\|_2^2 \right]$$

The reconstruction term for one image

Monte Carlo Estimation

That average is an integral over z .
Because f_D is a neural network, there is no closed-form solution

4. Average this over *all* possible noisy codes

3. Measure the squared error

$$\mathbb{E}_{z \sim \mathcal{N}(\mu^{(i)}, \Sigma^{(i)})} \left[\left\| x^{(i)} - f_D(z; \theta) \right\|_2^2 \right]$$

1. Draw a noisy code from the encoder's Gaussian

2. Decode it

Monte Carlo Estimation

That average is an
integral over z .
Because f_D is a neural
network, there is no
closed-form solution

4. Average this over *all*
possible noisy codes

3. Measure the
squared error

$$\mathbb{E}_{z \sim \mathcal{N}(\mu^{(i)}, \Sigma^{(i)})} \left[\left\| x^{(i)} - f_D(z; \theta) \right\|_2^2 \right]$$

1. Draw a noisy code
from the encoder's
Gaussian

2. Decode it

Monte Carlo Estimation

That average is an integral over z .
Because f_D is a neural network, there is no closed-form solution

4. Average this over *all* possible noisy codes

3. Measure the squared error

$$\mathbb{E}_{z \sim \mathcal{N}(\mu^{(i)}, \Sigma^{(i)})} \left[\|x^{(i)} - f_D(z; \theta)\|_2^2 \right]$$

1. Draw a noisy code from the encoder's Gaussian

2. Decode it

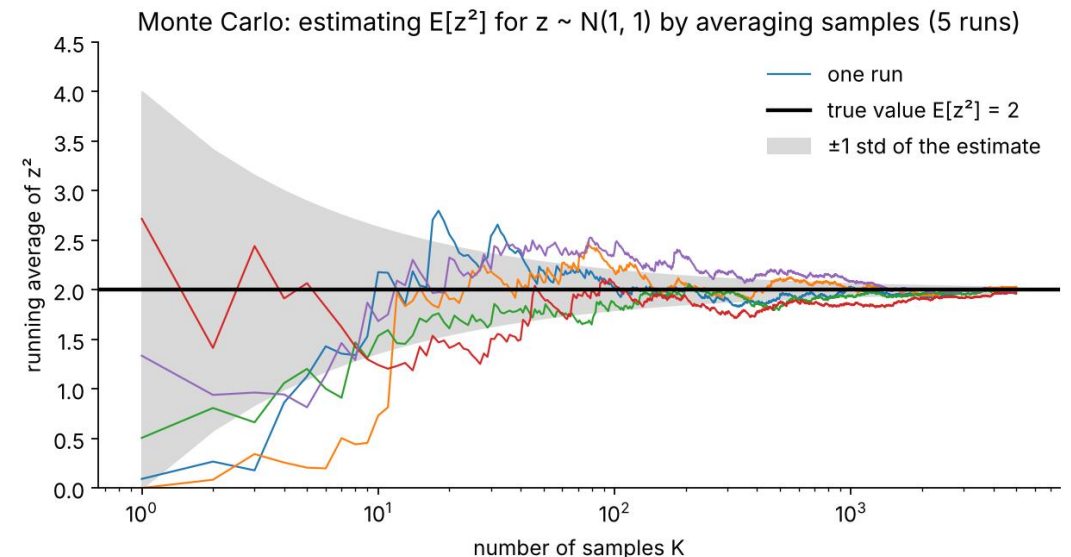
Solution: average over a few samples instead. We cannot average over every code, but we can draw some codes and average over those. This is **Monte Carlo estimation**

$$\mathbb{E}_{z \sim q} [f(z)] \approx \frac{1}{L} \sum_{l=1}^L f(z_l), \quad z_1, \dots, z_L \sim q$$

q is the encoder's $N(\mu^{(i)}, \Sigma^{(i)})$

$f(z_l) = \|x^{(i)} - f_D(z_l)\|^2$ is its squared error

Each z_l is one noisy code



Monte Carlo Estimation

That average is an integral over z .
Because f_D is a neural network, there is no closed-form solution

4. Average this over *all* possible noisy codes

3. Measure the squared error

$$\mathbb{E}_{z \sim \mathcal{N}(\mu^{(i)}, \Sigma^{(i)})} \left[\|x^{(i)} - f_D(z; \theta)\|_2^2 \right]$$

1. Draw a noisy code from the encoder's Gaussian

2. Decode it

Solution: average over a few samples instead. We cannot average over every code, but we can draw some codes and average over those. This is **Monte Carlo estimation**

$$\mathbb{E}_{z \sim q} [f(z)] \approx \frac{1}{L} \sum_{l=1}^L f(z_l), \quad z_1, \dots, z_L \sim q$$

q is the encoder's $N(\mu^{(i)}, \Sigma^{(i)})$

$f(z_l) = \|x^{(i)} - f_D(z_l)\|^2$ is its squared error

Each z_l is one noisy code

- **It is right on average (unbiased).** If you repeated the estimate many times with fresh samples, the results would average to the true value
- **Its error shrinks like $1/\sqrt{L}$.** More samples give a more precise estimate

The Reparameterization Trick

Training with gradient descent requires the gradient of J with respect to every parameter.
Follow the path backward from the loss for one image:

$$x^{(i)} \xrightarrow{f_E(\cdot; \phi)} (\mu^{(i)}, \sigma^{(i)}) \xrightarrow{\text{draw } z \sim \mathcal{N}(\mu^{(i)}, \sigma^{(i)2})} z \xrightarrow{f_D(\cdot; \theta)} \hat{x}^{(i)} \longrightarrow J$$

Without $\partial J / \partial \sigma$, the encoder never learns how much noise to use.

“Learn how much noise” would be impossible, and the encoder's σ outputs would stay wherever they started.

The Reparameterization Trick

Training with gradient descent requires the gradient of J with respect to every parameter.
Follow the path backward from the loss for one image:

The decoder θ is fine.
 z is just an input to f_D , so $\partial J / \partial \theta$
is ordinary backpropagation
through the decoder.

$$x^{(i)} \xrightarrow{f_E(\cdot; \phi)} (\mu^{(i)}, \sigma^{(i)}) \xrightarrow{\text{draw } z \sim \mathcal{N}(\mu^{(i)}, \sigma^{(i)2})} z \xrightarrow{f_D(\cdot; \theta)} \hat{x}^{(i)} \longrightarrow J$$

Without $\partial J / \partial \sigma$, the encoder never learns how much noise to use.

“Learn how much noise” would be impossible, and the encoder's σ outputs would stay wherever they started.

The Reparameterization Trick

Training with gradient descent requires the gradient of J with respect to every parameter.
Follow the path backward from the loss for one image:

The mean $\mu^{(i)}$ is fine.

$z_{noisy} = \mu^{(i)} + \varepsilon$ depends on $\mu^{(i)}$ like any sum: $\partial z / \partial \mu = 1$.

The decoder θ is fine.

z is just an input to f_D , so $\partial J / \partial \theta$ is ordinary backpropagation through the decoder.

$$x^{(i)} \xrightarrow{f_E(\cdot; \phi)} (\mu^{(i)}, \sigma^{(i)}) \xrightarrow{\text{draw } z \sim \mathcal{N}(\mu^{(i)}, \sigma^{(i)2})} z \xrightarrow{f_D(\cdot; \theta)} \hat{x}^{(i)} \longrightarrow J$$

Without $\partial J / \partial \sigma$, the encoder never learns how much noise to use.

“Learn how much noise” would be impossible, and the encoder's σ outputs would stay wherever they started.

The Reparameterization Trick

Training with gradient descent requires the gradient of J with respect to every parameter.
Follow the path backward from the loss for one image:

The mean $\mu^{(i)}$ is fine.

$z_{noisy} = \mu^{(i)} + \varepsilon$ depends on $\mu^{(i)}$ like any sum: $\partial z / \partial \mu = 1$.

The decoder θ is fine.

z is just an input to f_D , so $\partial J / \partial \theta$ is ordinary backpropagation through the decoder.

$$x^{(i)} \xrightarrow{f_E(\cdot; \phi)} (\mu^{(i)}, \sigma^{(i)}) \xrightarrow{\text{draw } z \sim \mathcal{N}(\mu^{(i)}, \sigma^{(i)2})} z \xrightarrow{f_D(\cdot; \theta)} \hat{x}^{(i)} \longrightarrow J$$

The width $\sigma^{(i)}$ is the problem.

σ does not appear in a formula for z .

It only sets how spread out the distribution is that ε is drawn from.

“Draw a random number” is not a function of σ : the same σ gives a different z on every call, so $\partial z / \partial \sigma$ is **undefined**.

The Reparameterization Trick

Training with gradient descent requires the gradient of J with respect to every parameter.
Follow the path backward from the loss for one image:

The mean $\mu^{(i)}$ is fine.

$z_{noisy} = \mu^{(i)} + \varepsilon$ depends on $\mu^{(i)}$ like any sum: $\partial z / \partial \mu = 1$.

The decoder θ is fine.

z is just an input to f_D , so $\partial J / \partial \theta$ is ordinary backpropagation through the decoder.

$$x^{(i)} \xrightarrow{f_E(\cdot; \phi)} (\mu^{(i)}, \sigma^{(i)}) \xrightarrow{\text{draw } z \sim \mathcal{N}(\mu^{(i)}, \sigma^{(i)2})} z \xrightarrow{f_D(\cdot; \theta)} \hat{x}^{(i)} \longrightarrow J$$

How to fix:

Rewrite the “draw” so that σ appears in a formula for z

The width $\sigma^{(i)}$ is the problem.

σ does not appear in a formula for z .

It only sets how spread out the distribution is that ε is drawn from.

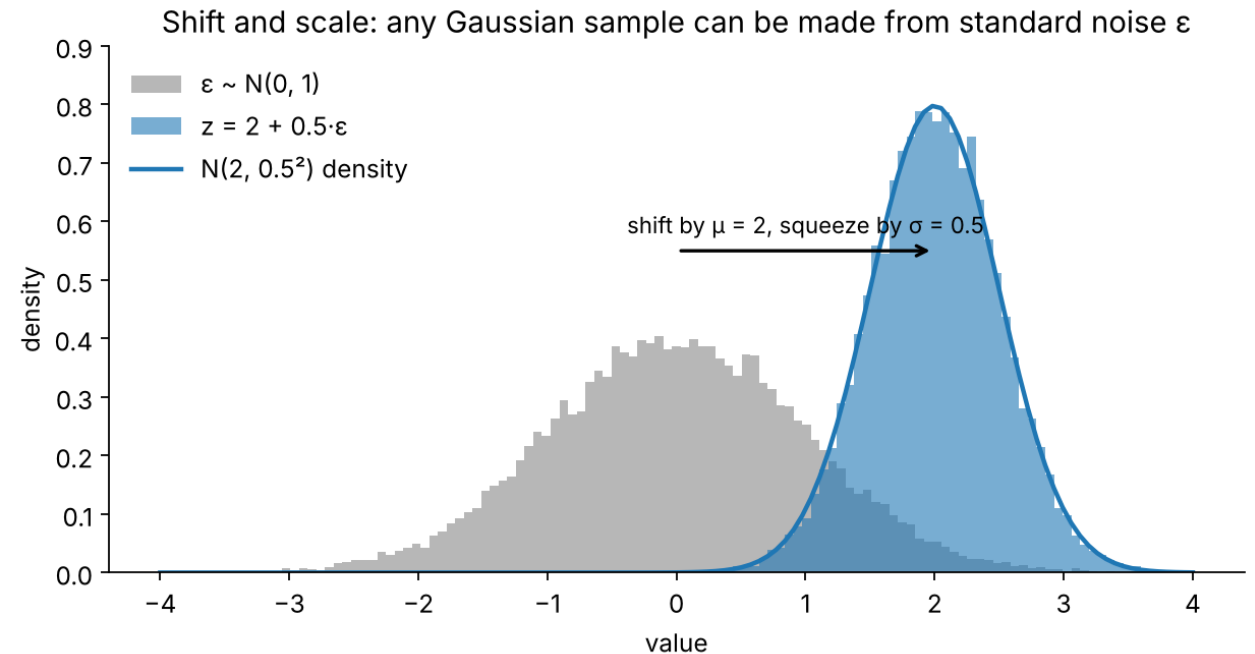
“Draw a random number” is not a function of σ : the same σ gives a different z on every call, so $\partial z / \partial \sigma$ is **undefined**.

Without $\partial J / \partial \sigma$, the encoder never learns “how much noise to use”.

The Reparameterization Trick

Any Gaussian sample can be made by shifting and scaling a standard one
if $\varepsilon_0 \sim N(0,1)$, then $\mu + \sigma\varepsilon_0 \sim N(\mu, \sigma^2)$

1. **Draw the noise.** Sample $\varepsilon_0 \sim N(0, I)$, one standard-normal number per code dimension (K numbers). This is the only random step, and **it does not involve the encoder at all.**
2. **Scale.** Multiply by the encoder's width, element by element: $\sigma^{(i)} \odot \varepsilon_0$. This is now noise with standard deviation $\sigma^{(i)}$, i.e. $\varepsilon \sim N(0, \Sigma^{(i)})$.
3. **Shift.** Add the encoder's mean: $z = \mu^{(i)} + \sigma^{(i)} \odot \varepsilon_0$. The result has mean $\mu^{(i)}$ and standard deviation $\sigma^{(i)}$, so $z \sim N(\mu^{(i)}, \Sigma^{(i)})$.



The Reparameterization Trick

Rewrite the noise as scaled standard noise

$$\varepsilon_0 \sim \mathcal{N}(0, I)$$

$$\varepsilon = \sigma^{(i)} \odot \varepsilon_0 \sim \mathcal{N}(0, \Sigma^{(i)})$$

$$z_{\text{noisy}} = \mu^{(i)} + \sigma^{(i)} \odot \varepsilon_0$$

The randomness now lives in ε_0 , which does not depend on φ , and z_{noisy} is an ordinary function of μ and σ with $\partial z / \partial \mu = 1$ and $\partial z / \partial \sigma = \varepsilon_0$.

The gradient moves inside the average:

$$\nabla_{\phi} \mathbb{E}_{z \sim \mathcal{N}(\mu^{(i)}, \Sigma^{(i)})} [f(z)] = \mathbb{E}_{\varepsilon_0 \sim \mathcal{N}(0, I)} [\nabla_{\phi} f(\mu^{(i)} + \sigma^{(i)} \odot \varepsilon_0)]$$

