DDA4220 · Fall 2026

Lecture 15

Deep Learning and Applications

Autoencoders and variational autoencoders

Zhen Liu

CUHK-Shenzhen

Lecture outline

01 Learning by reconstruction

02 From reconstruction to generation

03 The evidence lower bound

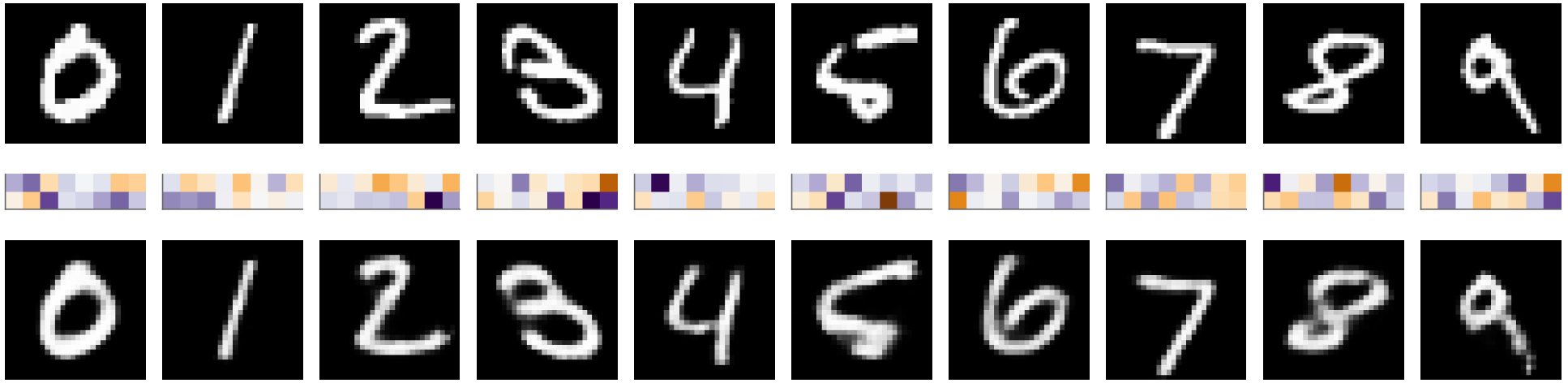
04 Training a VAE

05 Trade-offs and limits

06 Discrete codes and learned priors

* denotes optional slides

Compact codes from unlabeled images

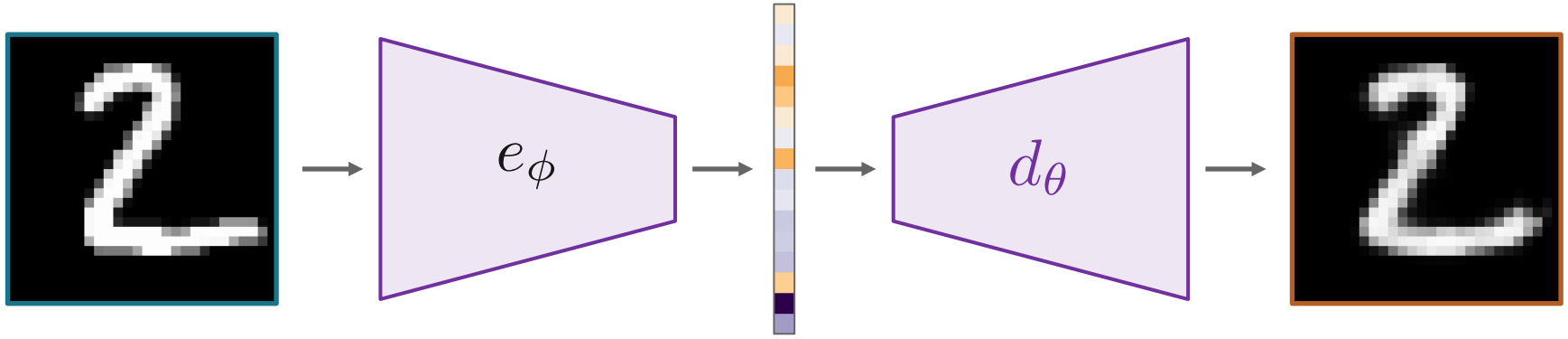


784 pixels $\rightarrow$ 16 numbers $\rightarrow$ 784 pixels

No labels: the target is the input itself

16 numbers must keep what digits share

The autoencoder



$$\ell(\theta, \phi) = \| \underbrace{x}_{\text{input}} - \underbrace{d_\theta(e_\phi(x))}_{\text{reconstruction}} \|^2$$

e_ϕ, d_θ: neural networks with parameters ϕ and θ

$\|v\|^2$: sum of squared entries; here, squared pixel errors

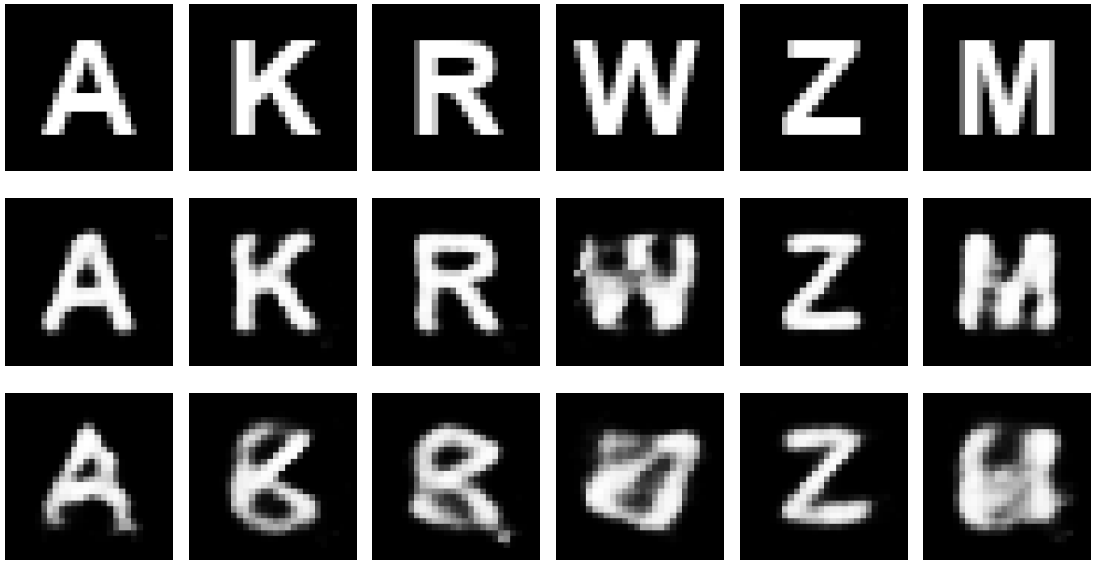
A large code can simply copy

A code as large as the input can copy: $e(x) = x$, $d(z) = z$, zero error

input (not a digit)

code of 1024
error 9 on average

code of 16
error 32 on average

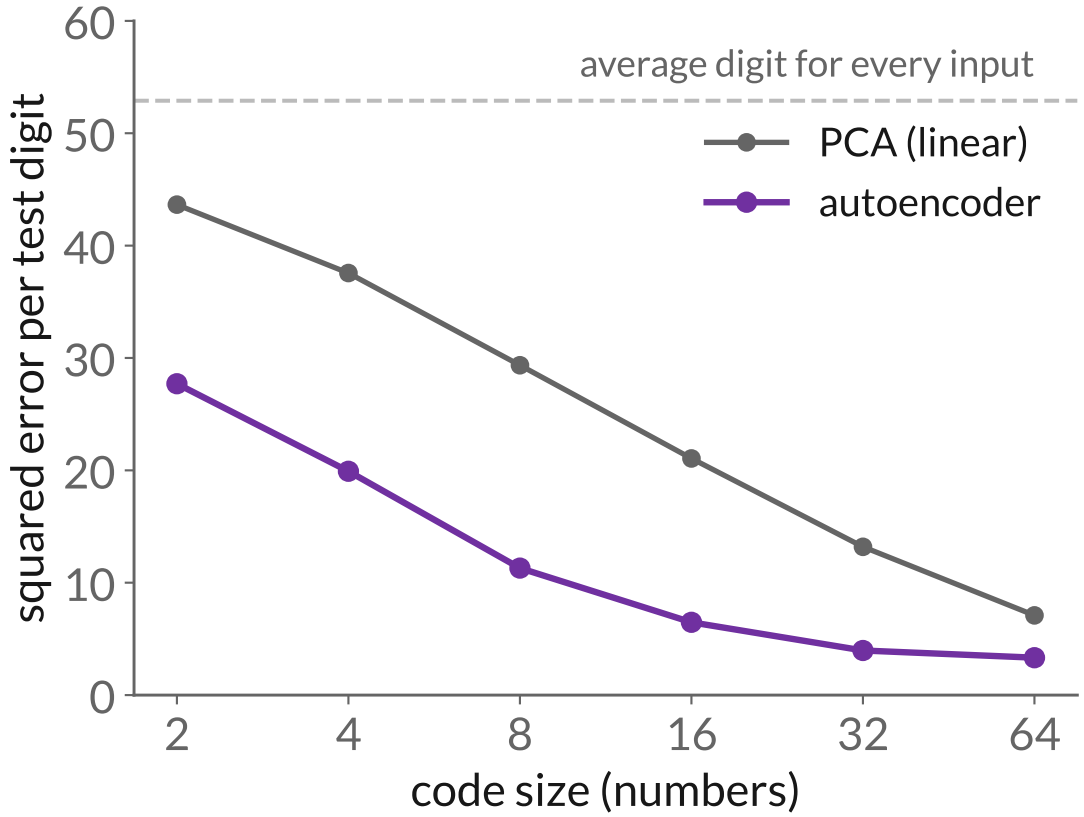


Letters never seen in training: 1024 numbers copy them, 16 turn them into digits

On test digits both do well: error 3 and 7

Only a small code forces the model to learn about digits

The bottleneck: how many numbers?





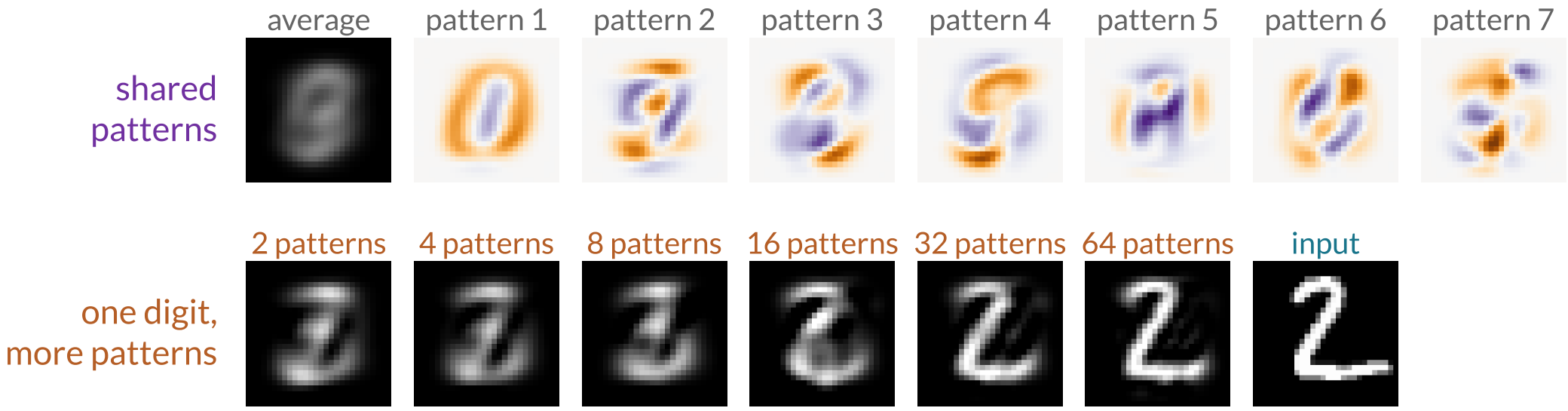
PCA, a linear method, needs far more numbers for the same error

2 numbers keep the digit's identity and slant; 32 keep most strokes

A linear autoencoder is PCA

$$x \approx \bar{x} + z_1 w_1 + z_2 w_2 + \cdots + z_k w_k$$

$\bar{x}$: average digit; w_j : shared patterns (images); z_j : the code, one weight per pattern

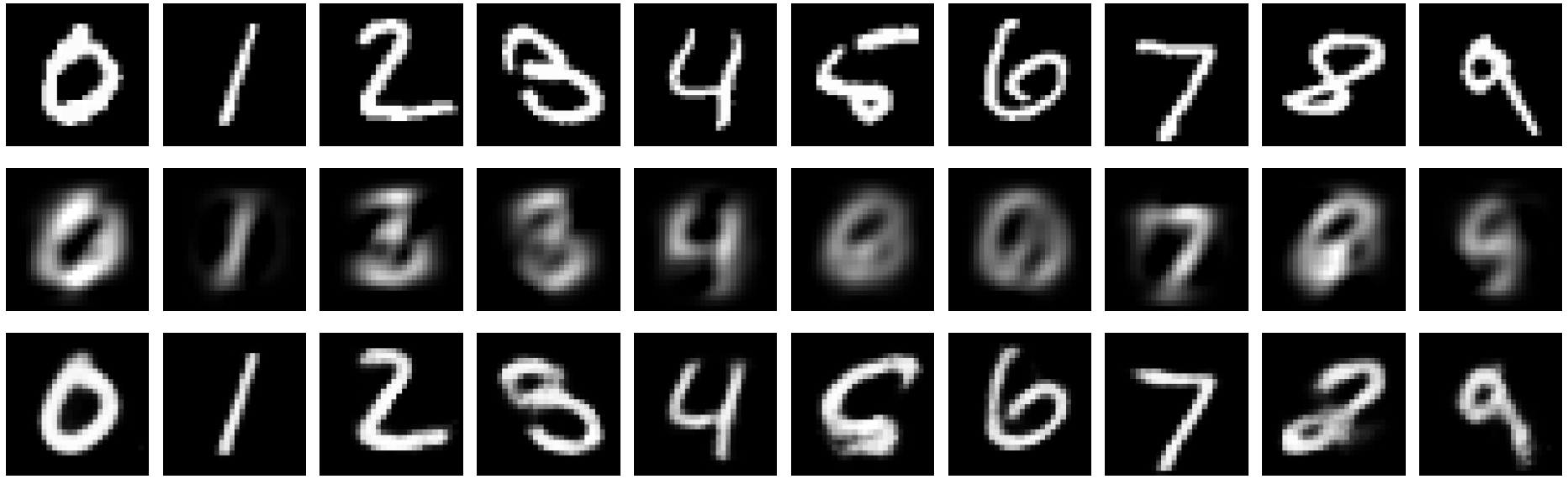


Linear encoder and decoder, squared error: the best k patterns span PCA's first k directions

Check, 8 numbers: linear autoencoder error 29.5, PCA 29.4

Nonlinear codes reconstruct better

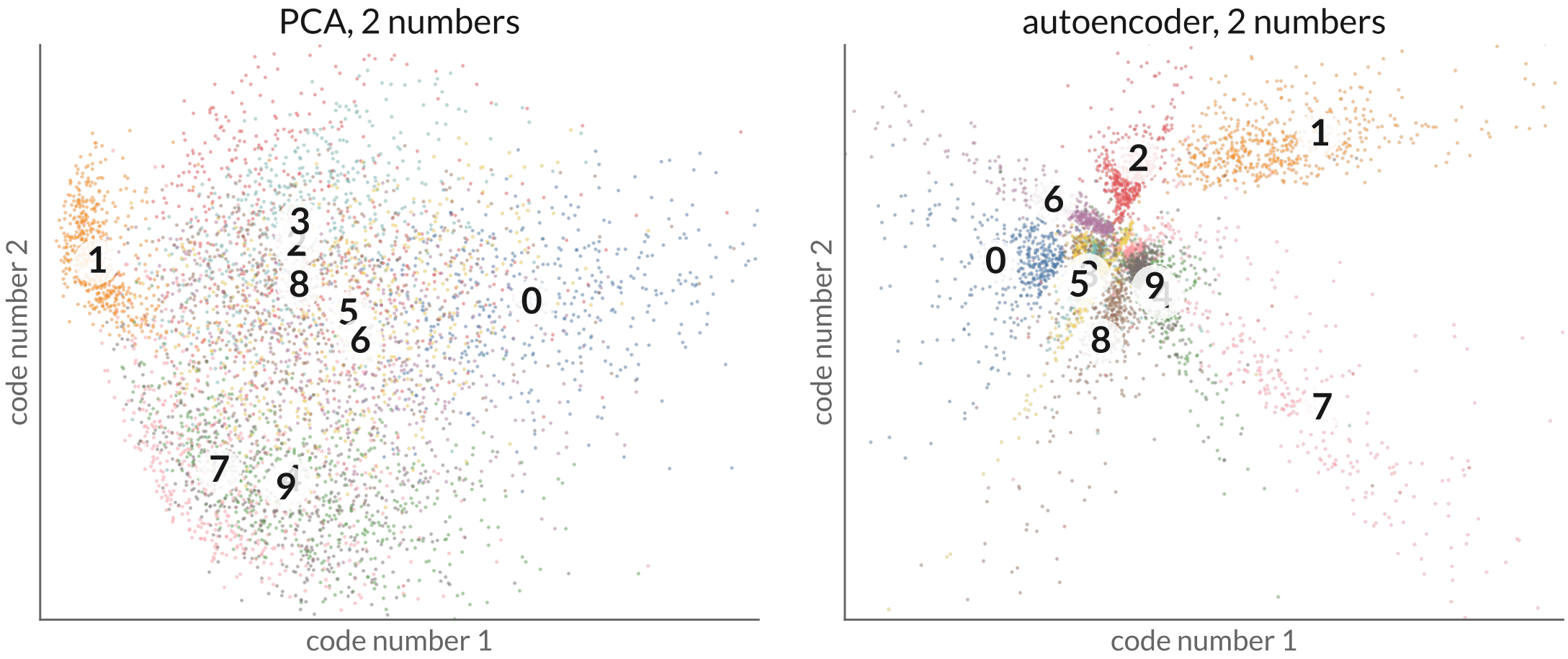
Every digit coded with 8 numbers



8 autoencoder numbers even beat 32 PCA numbers (error 13)

A nonlinear decoder can do more than add fixed patterns

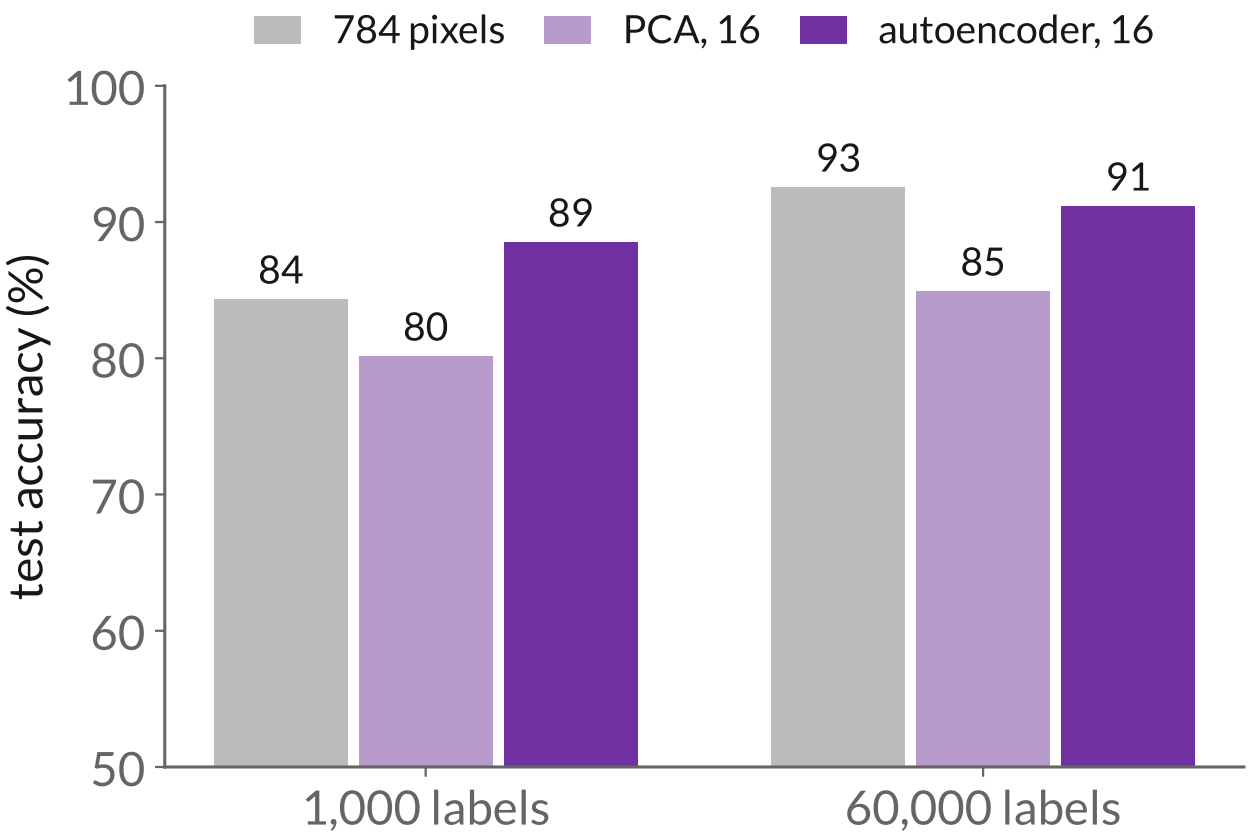
Codes organize the data



5,000 test digits coded with 2 numbers; colors: labels, never used in training

Autoencoder: each digit class gathers in its own region

Reusing the code



A linear classifier (logistic regression) on each representation

With few labels, 16 code numbers beat 784 pixels

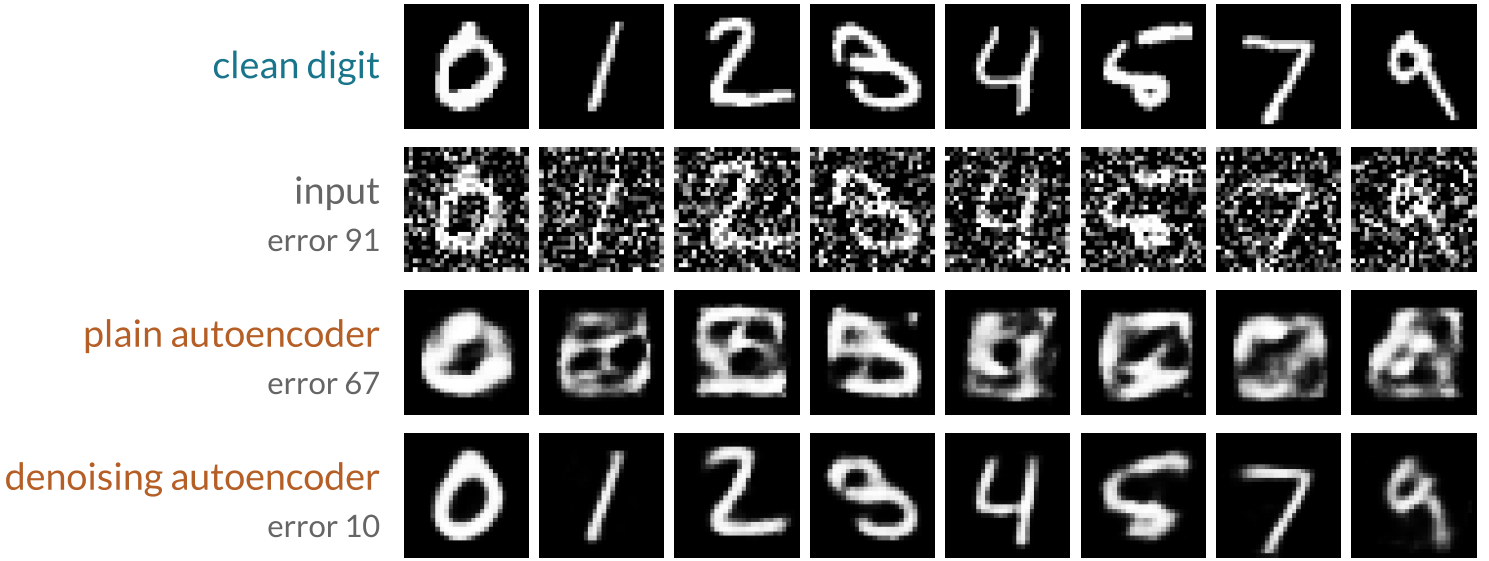
Keeps class information; not built for it

L14: masked and contrastive objectives learn stronger features

Denoising autoencoders

$$\ell = \| \underbrace{x} - d_\theta(e_\phi(\underbrace{\tilde{x}})) \|^2$$

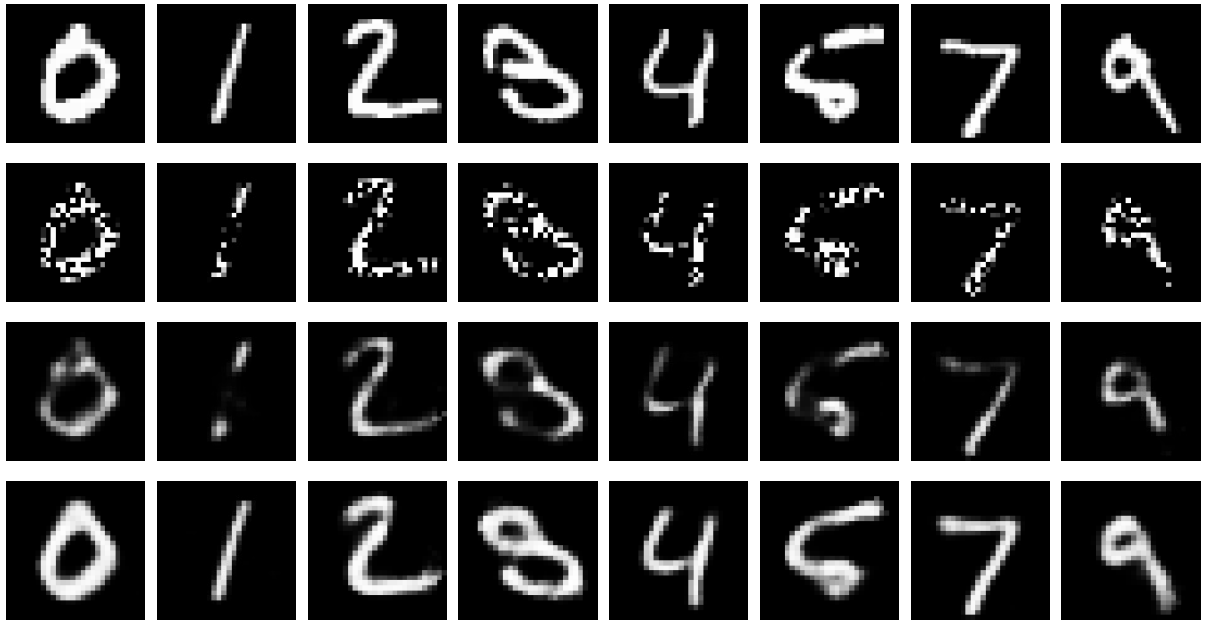
clean target corrupted input



To remove the noise, the model must know what digits look like

$\tilde{x}$: x plus Gaussian noise (std 0.5), clipped to $[0, 1]$; codes of 32 numbers

Other corruptions: masking

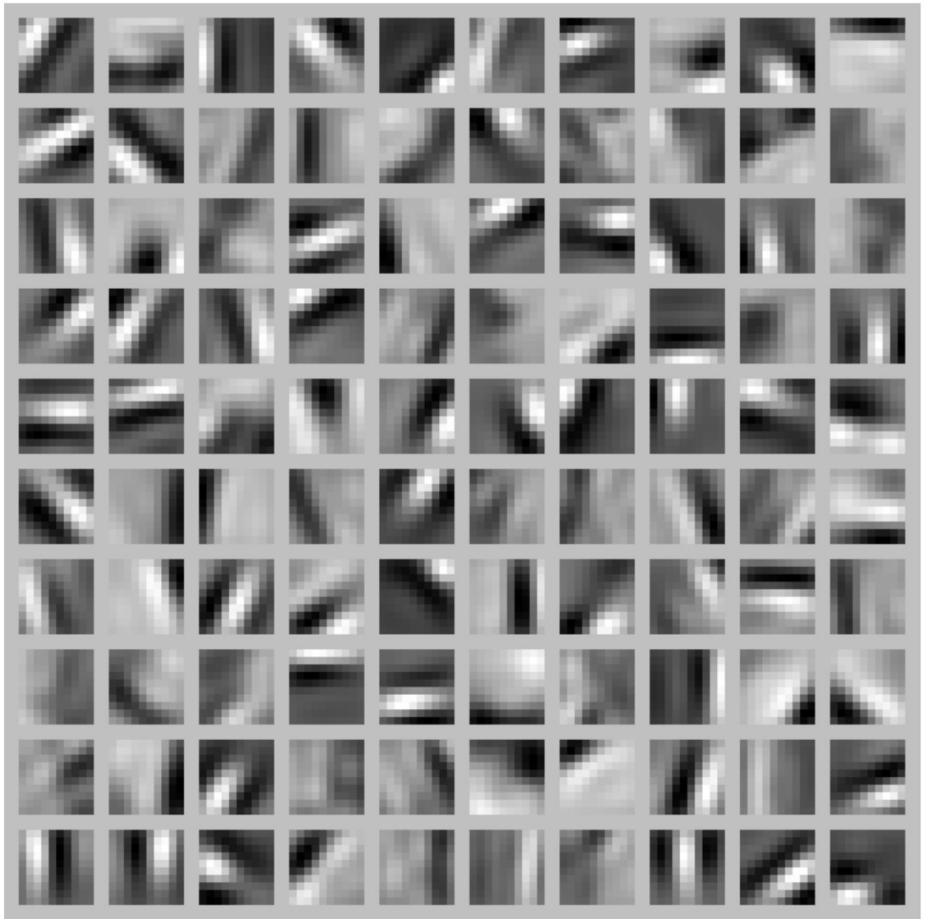


Remove half the pixels at random; the target is still the clean digit

Masked autoencoders (L14) do this with image patches

Diffusion models (L17) train exactly this kind of denoiser, at many noise levels

Sparse autoencoders



$$\ell = \|x - d_\theta(e_\phi(x))\|^2 + \underbrace{\lambda \sum_j |z_j|}_{\text{few nonzero } z_j}$$

Bottleneck by sparsity instead of size: many code numbers, few active per input

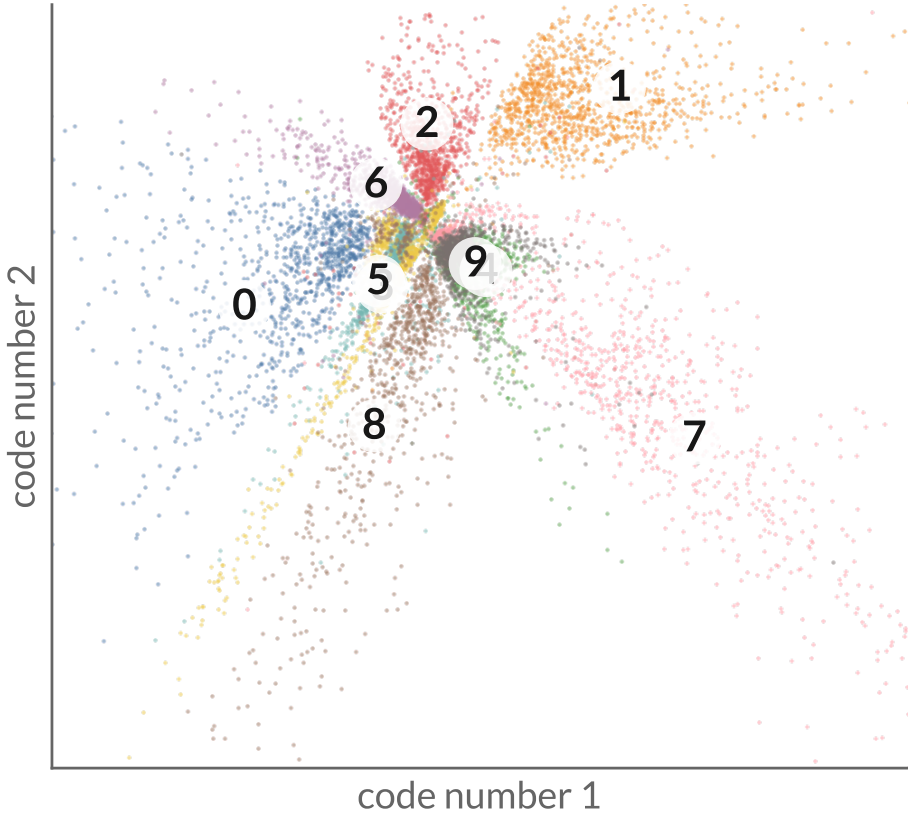
Left: the input that most activates each of 100 code numbers (natural-image patches)

Each number becomes an edge detector

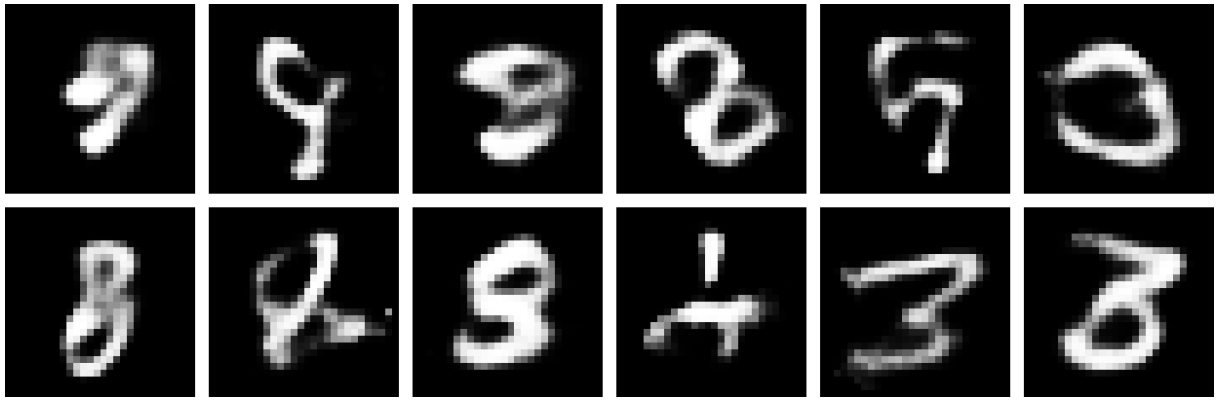
Source: Ng, Sparse autoencoder (CS294A notes)

Can the decoder generate?

2-number autoencoder: training codes



16-number autoencoder: random codes, decoded



drawn with the training codes' mean and spread

Valid codes: an irregular region

Random codes: some digits, many broken strokes

No distribution over codes to sample from

Lecture outline

01 Learning by reconstruction

02 From reconstruction to generation

03 The evidence lower bound

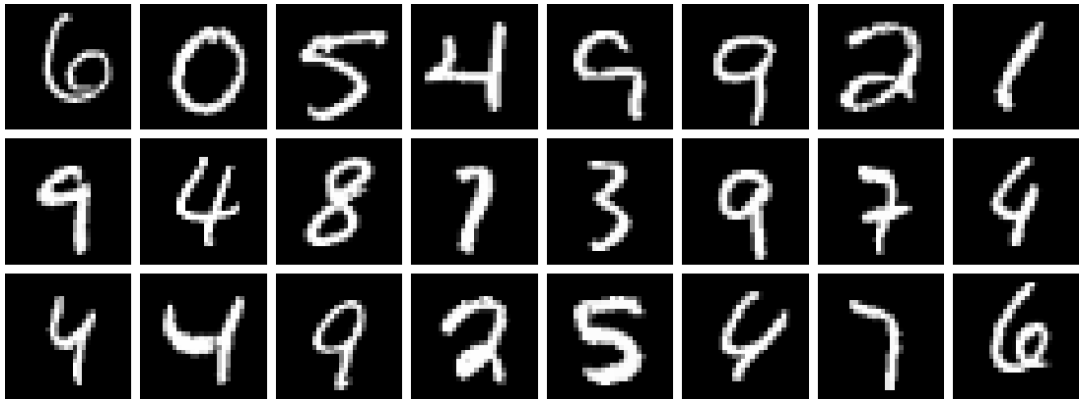
04 Training a VAE

05 Trade-offs and limits

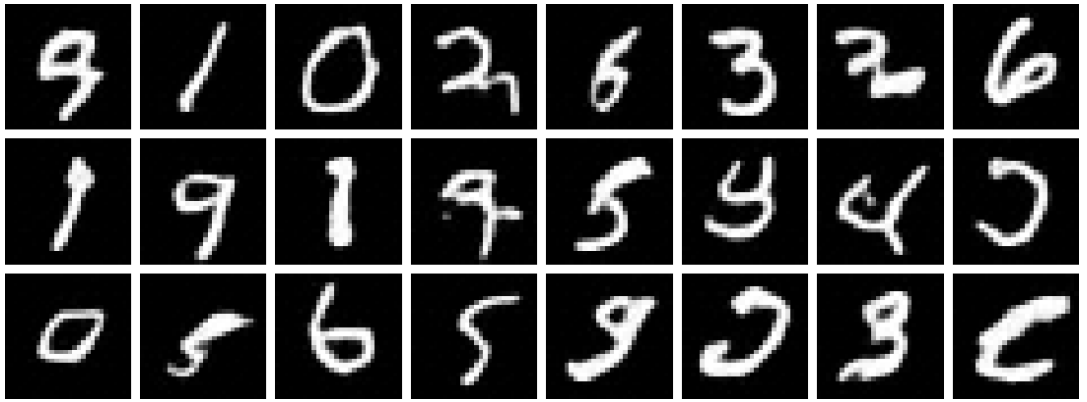
06 Discrete codes and learned priors

Generation: learn the data distribution

training digits



new digits from a model (end of this lecture)

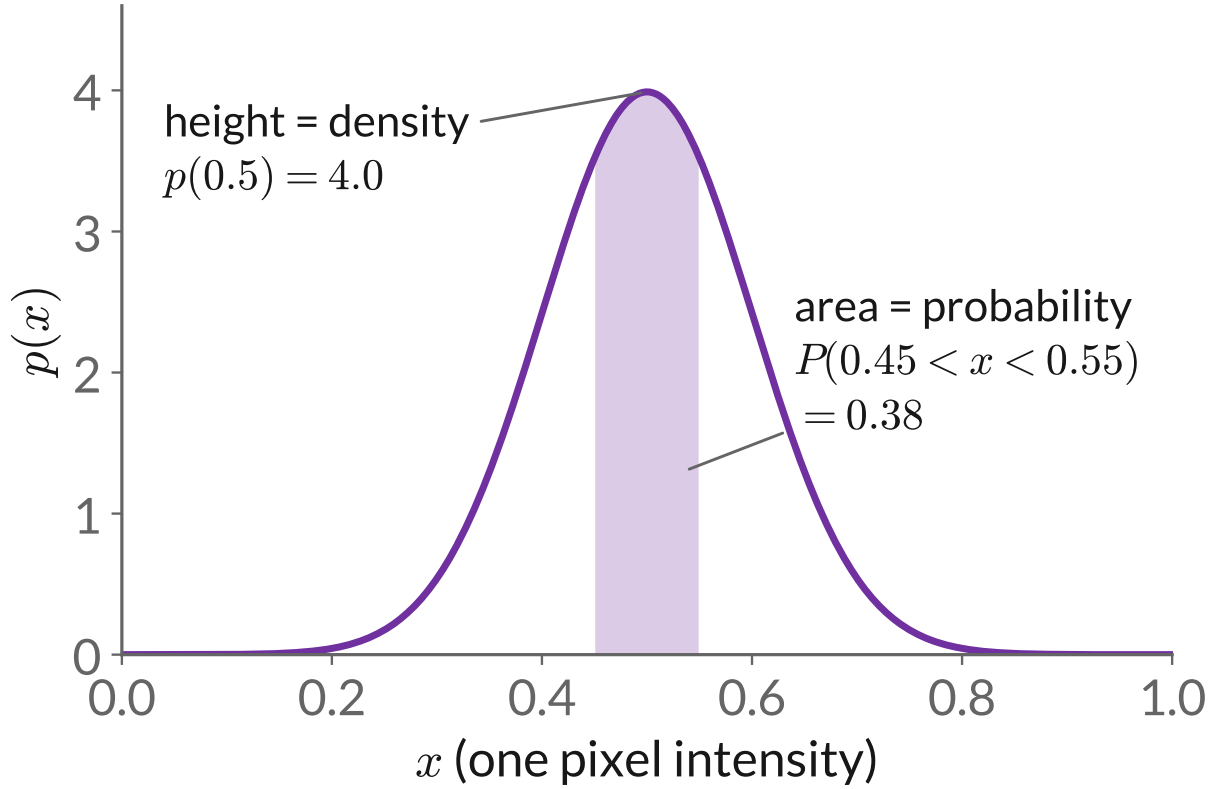


Classification: $p(y \mid x)$, the label given the image

Generation: $p(x)$, how likely each image is; then sample from it

Goal: fit $p_\theta(x)$ to examples, in a form we can sample

Probability density



Continuous pixels: $p(x)$ is a density

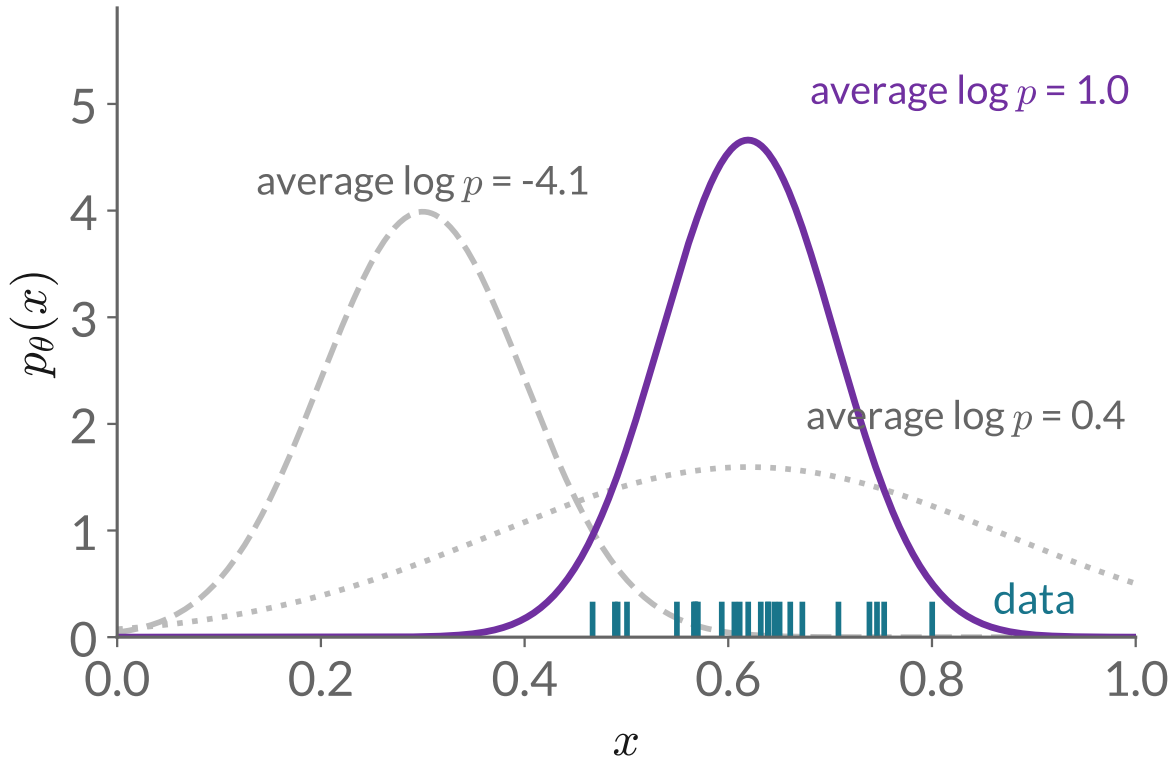
Probability = area; height can exceed 1

An image: a density over 784 numbers

Log densities of digits are large and positive: hundreds of nats for our models

Maximum likelihood

$$\theta^\star = \arg\max_\theta \frac{1}{N} \sum_{i=1}^{N} \log p_\theta\left(x^{(i)}\right)$$



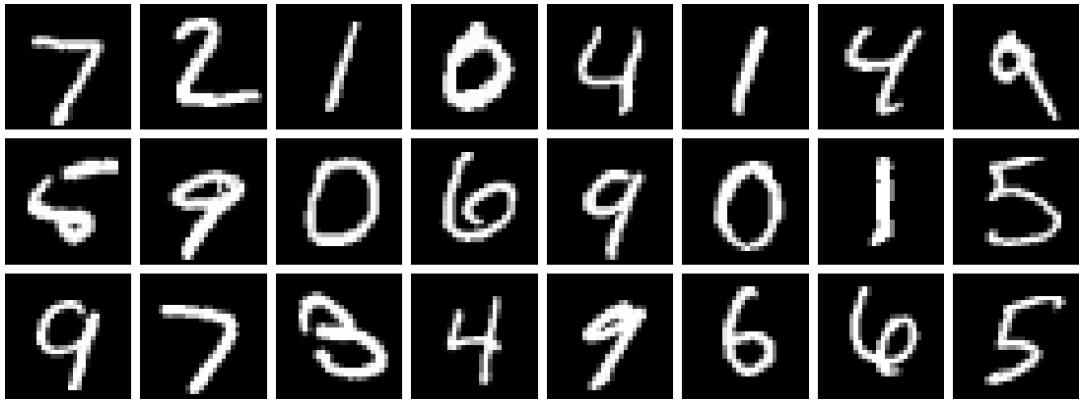
$x^{(i)}$: the i-th of N training examples

Left: of three Gaussians, the best has the highest average $\log p$

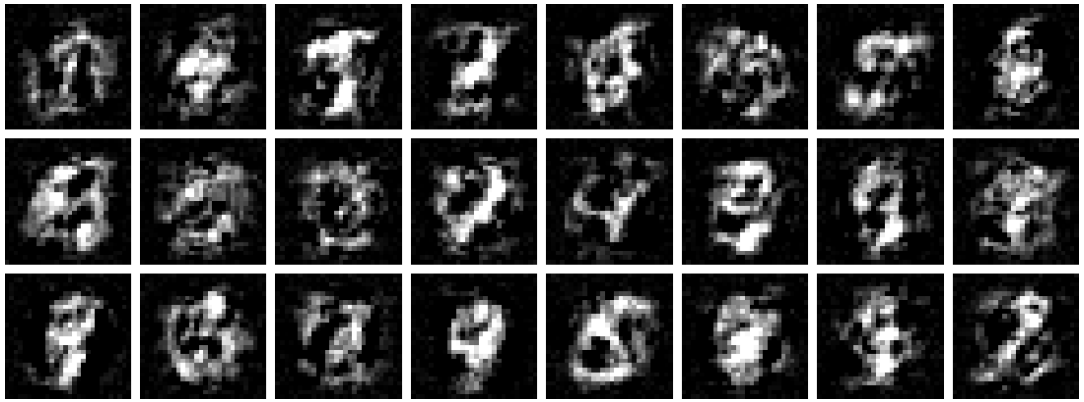
Same as minimizing the KL divergence from the data distribution (L02)

One Gaussian is too simple

training digits



samples from one fitted Gaussian



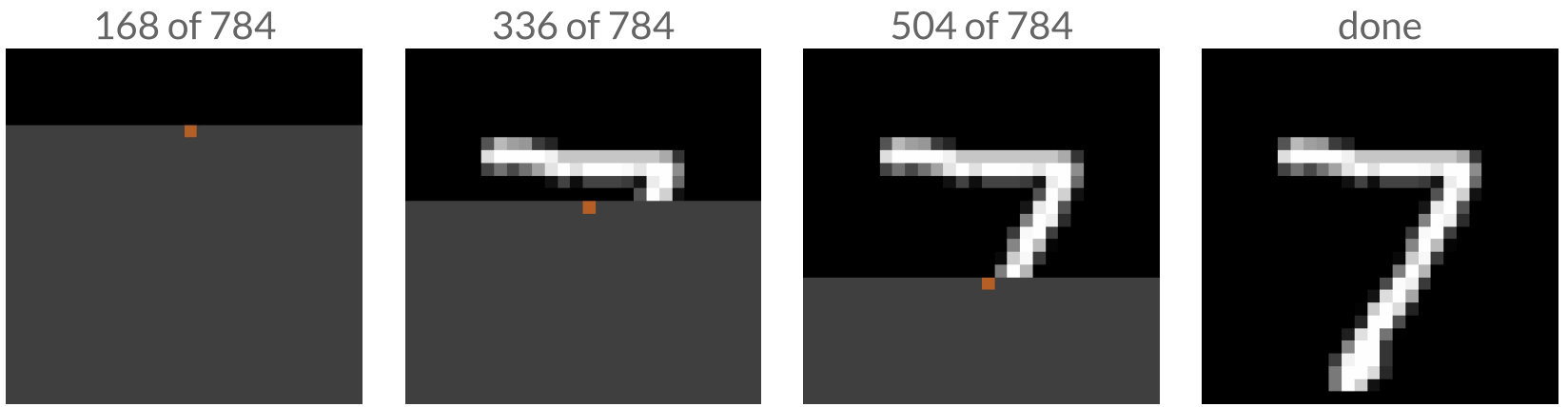
Mean and covariance of all 784 pixels by maximum likelihood: 308,504 parameters

Samples: the average layout, no clean strokes

Digits are not one Gaussian blob

Pixel by pixel: autoregressive models

$$p_\theta(x) = \underbrace{\prod_{i=1}^{784} p_\theta\left(x_i \mid x_1, \ldots, x_{i-1}\right)}_{\text{each pixel given all earlier pixels}}$$



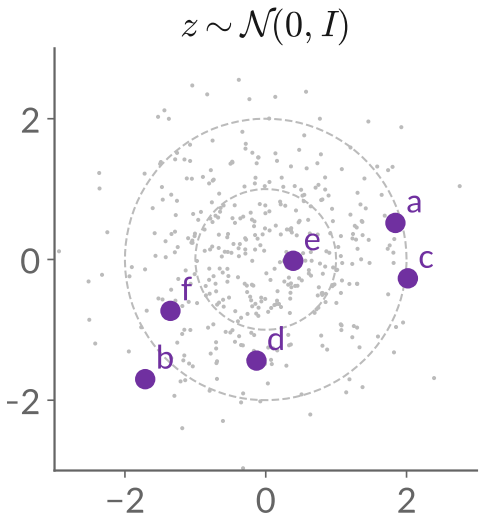
Exact likelihood, like a language model over words (L10, L12)

Sampling: 784 steps in sequence; a 256×256 color image needs about 200,000

Can a few random numbers decide the whole image in one pass?

A Gaussian code and a learned decoder

$$\underbrace{z \sim \mathcal{N}(0, I)}_{\text{code from a fixed distribution}} \qquad x \mid z \sim \mathcal{N}\Big(\underbrace{\mu_\theta(z)}_{\text{decoder network}}, \underbrace{\sigma^2 I}_{\text{pixel noise}} \Big)$$



$\mu_\theta(z)$ ⟶

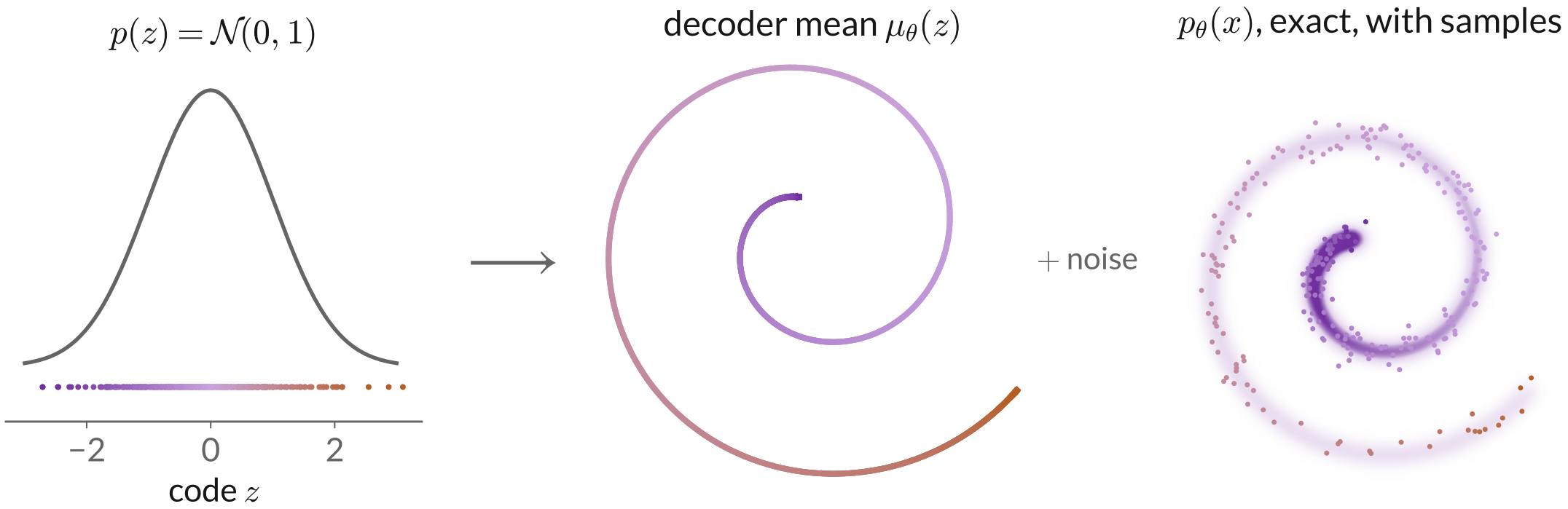


μ_θ : an autoencoder's decoder, fed codes from a distribution we choose

Right: six codes, decoded by a trained 2-number decoder

Generation in one pass: draw z, decode

A simple code can make complex data



Code $z \sim \mathcal{N}(0, 1)$; the decoder bends it onto a spiral, plus a little noise

Exact $p_\theta(x)$: a thin curved ridge

A nonlinear decoder makes a simple code distribution complex

The likelihood is an average over codes

$$p_\theta(x) = \int \underbrace{p_\theta(x \mid z)}_{\text{how well } z \text{ explains } x} \; \underbrace{p(z)}_{\text{how common } z \text{ is}} \, dz$$

Maximum likelihood needs $\log p_\theta(x)$ for every training image

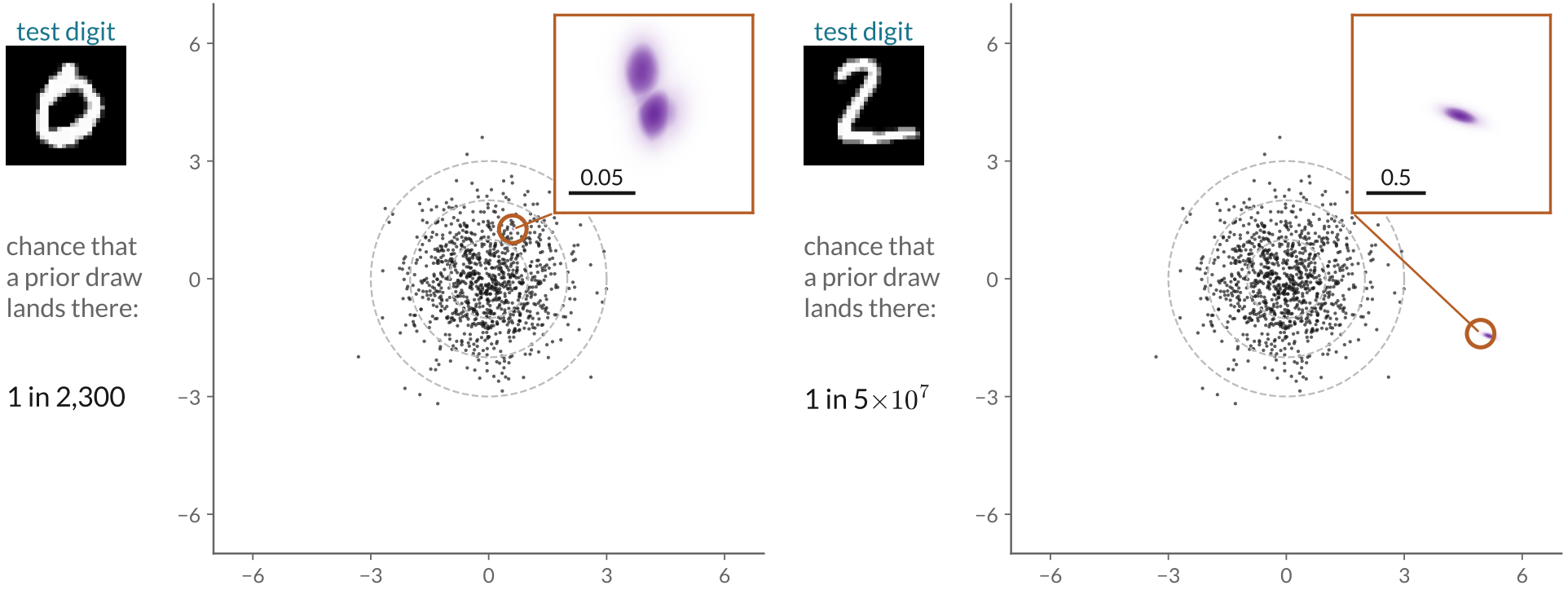
No closed form: the decoder is a neural network

The obvious estimate: average over random codes

$$p_\theta(x) \approx \frac{1}{K} \sum_{k=1}^{K} p_\theta\left(x \mid z^{(k)}\right), \qquad z^{(k)} \sim p(z)$$

Does it work?

Few codes explain one image

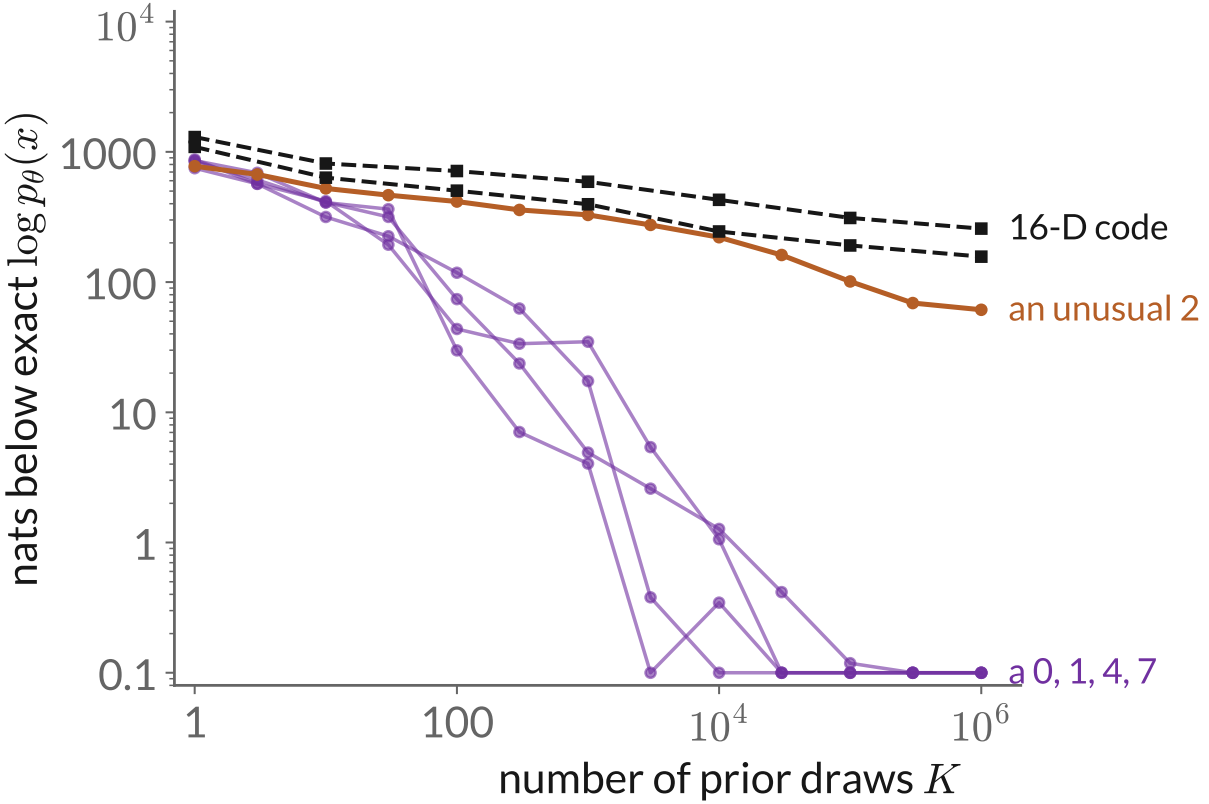


2-number VAE decoder; purple: high $p_\theta(x \mid z)$; dots: 1,000 draws from $p(z)$

The codes that explain a digit fill a tiny region; no draw lands in it

Orange: the codes holding 99% of $p_\theta(x \mid z)\, p(z)$, magnified in the inset; gray: 1, 2, 3 prior spreads

Random codes give a poor estimate



Typical digits: within 1 nat by 10^4 draws

Unusual 2: still 60 nats short at 10^6

16 code numbers: over 150 nats short

Needed for every image at every step: far too costly

Exact: a grid over the code plane; 16 numbers: importance-sampled; medians of 8 runs

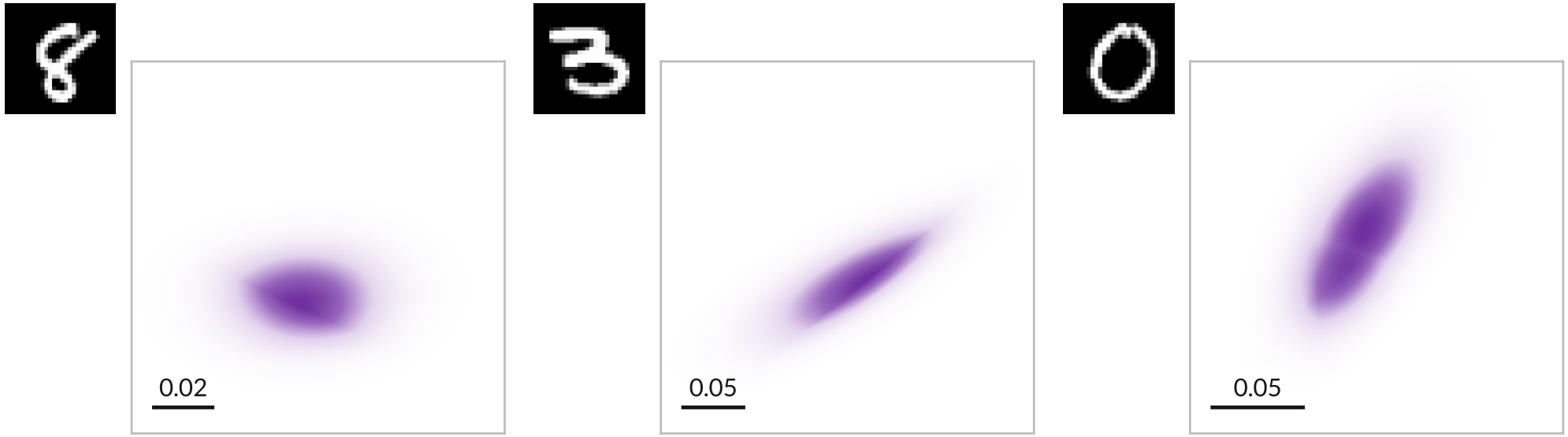
The codes that explain an image

$$\underbrace{p_\theta(z \mid x)}_{\text{posterior}} = \frac{p_\theta(x \mid z)\, p(z)}{p_\theta(x)}$$

Codes that explain x, weighted by how common

Draw from it: a few codes would suffice

But it divides by $p_\theta(x)$, the unknown



Exact posteriors of three test digits, magnified; bars: scale in code units

An encoder proposes codes

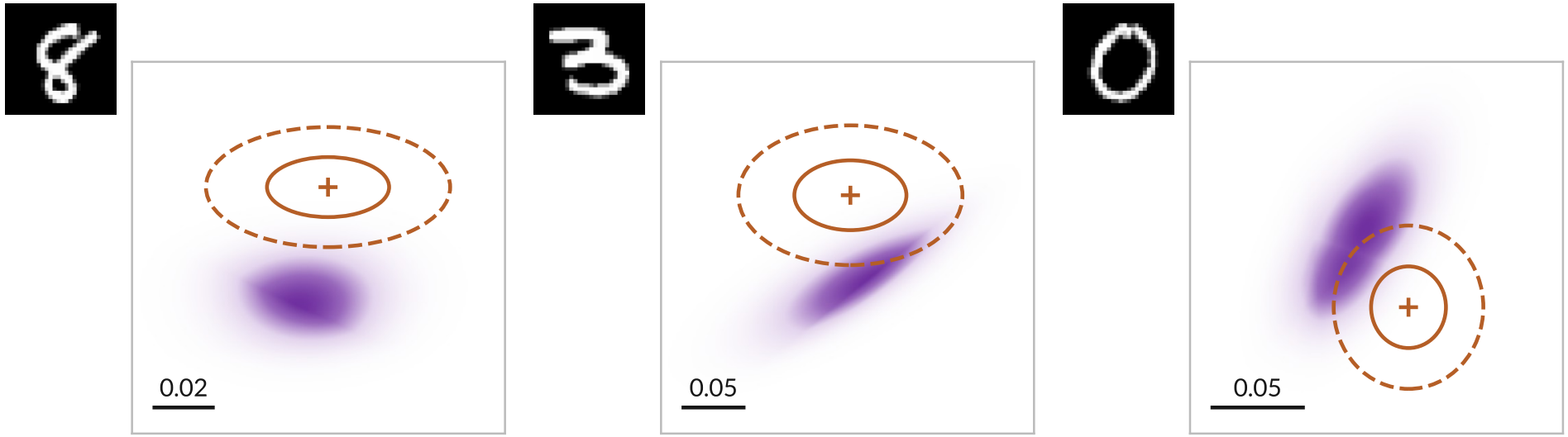
$$q_\phi(z \mid x) = \mathcal{N}\big(\underbrace{m_\phi(x)}_{\text{mean}}, \underbrace{\operatorname{diag} s_\phi(x)^2}_{\text{spread of each number}}\big)$$

A network maps x to a Gaussian over codes

Orange: 1 and 2 spreads of q_ϕ

Purple: the exact posterior

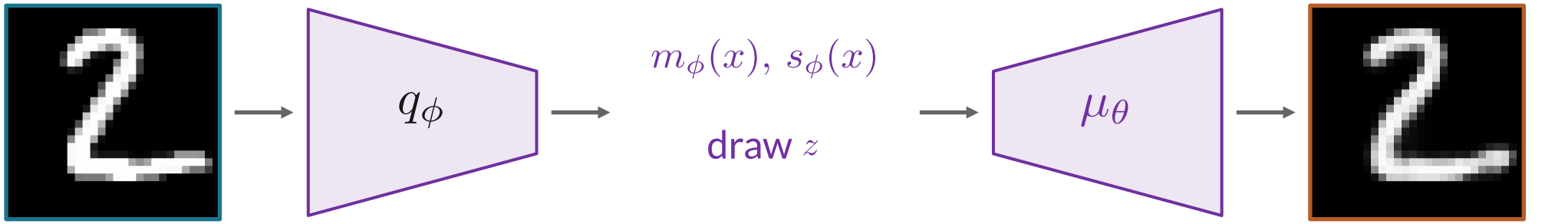
Close, not exact: off by under 0.1



Panels magnified; the whole code plane is 8 units wide

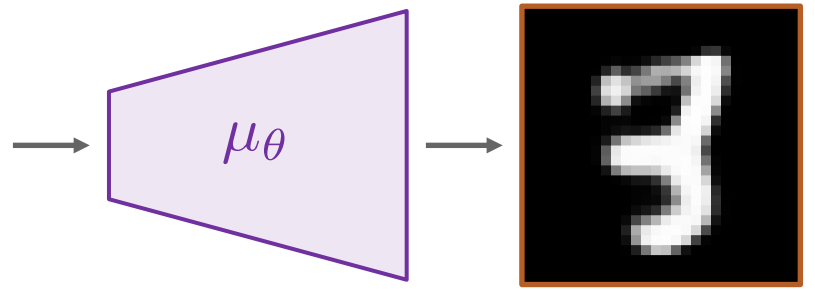
The variational autoencoder

Training



Generation

no encoder: start from a random code



An autoencoder with a probabilistic encoder, trained for high $\log p_\theta(x)$

Decoder: generates. Encoder: makes training possible

Lecture outline

01 Learning by reconstruction

02 From reconstruction to generation

03 The evidence lower bound

04 Training a VAE

05 Trade-offs and limits

06 Discrete codes and learned priors

Averaging over the encoder's codes

Goal: $\log p_\theta(x)$ using only codes drawn from $q_\phi(z \mid x)$

$$\log p_\theta(x) = \underbrace{\mathbb{E}_{z \sim q_\phi(z|x)}}_{\text{over the encoder's codes}} \left[\log p_\theta(x)\right]$$

Any distribution works: the bracket does not depend on z

Bayes' rule, for every code z:

$$p_\theta(x) = \frac{p_\theta(x \mid z)\, p(z)}{p_\theta(z \mid x)} \qquad \Longrightarrow \qquad \log p_\theta(x) = \mathbb{E}_{q_\phi}\left[\log \frac{p_\theta(x \mid z)\, p(z)}{p_\theta(z \mid x)}\right]$$

The posterior is still inside; next, bring in the encoder

Multiply and divide by the encoder

$$\log p_\theta(x) = \mathbb{E}_{q_\phi}\left[\log\left(p_\theta(x \mid z) \cdot \frac{p(z)}{q_\phi(z \mid x)} \cdot \frac{q_\phi(z \mid x)}{p_\theta(z \mid x)}\right)\right]$$

$$= \underbrace{\mathbb{E}_{q_\phi}\left[\log p_\theta(x \mid z)\right]}_{\text{reconstruction}} - \underbrace{D_{\mathrm{KL}}\left(q_\phi(z \mid x) \,\|\, p(z)\right)}_{\text{encoder vs prior}} + \underbrace{D_{\mathrm{KL}}\left(q_\phi(z \mid x) \,\|\, p_\theta(z \mid x)\right)}_{\text{encoder vs posterior}}$$

KL divergence (L02): how far one distribution is from another

$$D_{\mathrm{KL}}(q \,\|\, p) = \mathbb{E}_q\left[\log \frac{q(z)}{p(z)}\right] \geq 0, \qquad = 0 \text{ only when } q = p$$

Log of a product: one term per factor

The evidence lower bound

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

$$\log p_\theta(x) = \mathrm{ELBO}(x) + \underbrace{D_{\mathrm{KL}}\left(q_\phi(z \mid x) \,\|\, p_\theta(z \mid x)\right)}_{\text{the gap, never negative}}$$

Evidence: $\log p_\theta(x)$; so $\mathrm{ELBO} \leq \log p_\theta(x)$

Computable: needs only encoder, decoder and prior

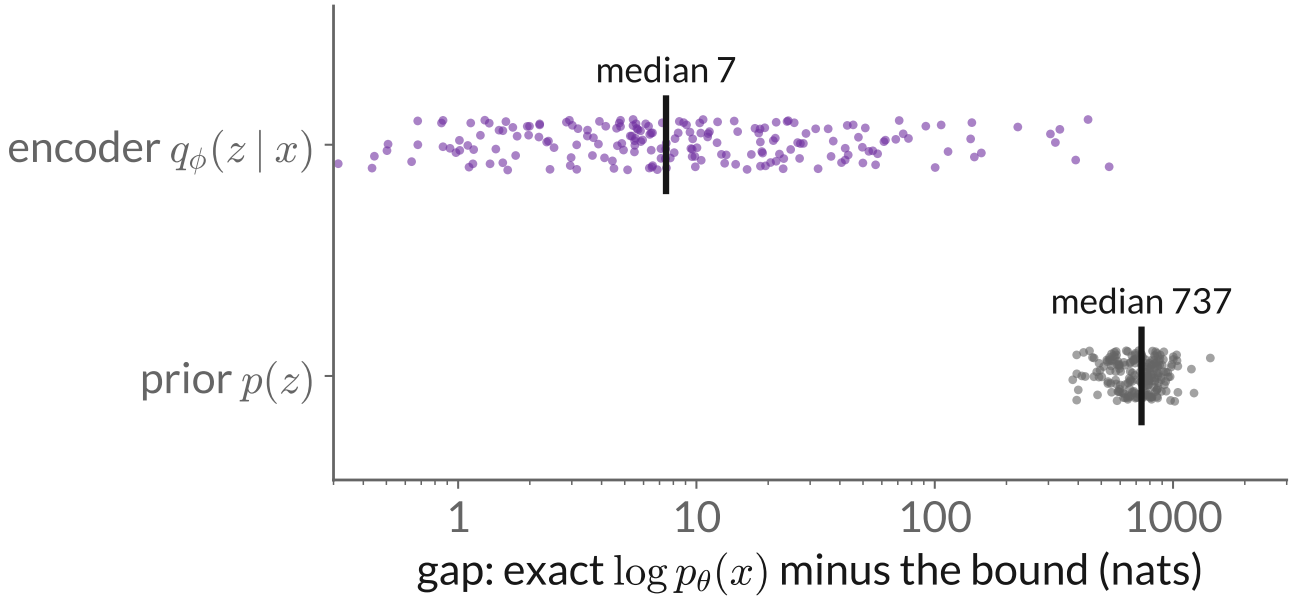
Maximize the average ELBO over θ and ϕ together

Raising the ELBO raises $\log p_\theta(x)$, or shrinks the gap

Why the encoder matters

Use the prior as $q_\phi(z \mid x)$: the KL term vanishes, a weaker bound remains

$$\mathbb{E}_{p(z)}\left[\log p_\theta(x \mid z)\right] \;\leq\; \log p_\theta(x) \qquad \text{gap} = D_{\mathrm{KL}}\left(p(z) \,\|\, p_\theta(z \mid x)\right)$$

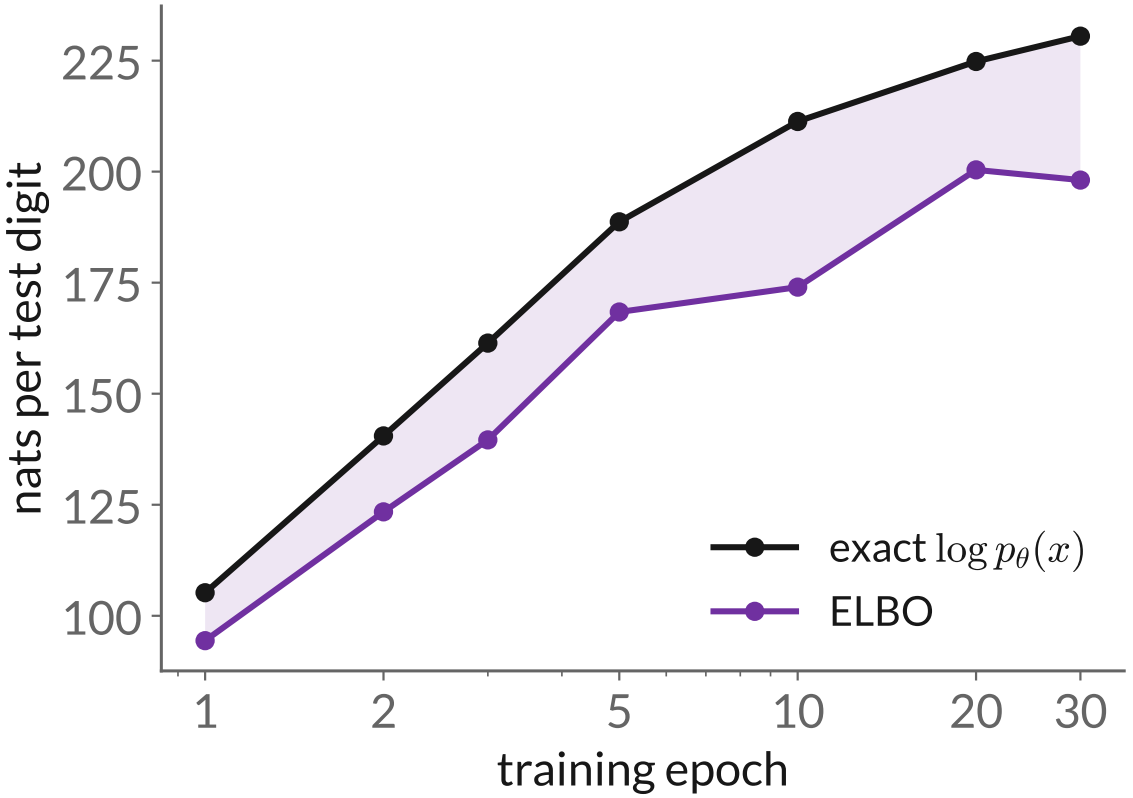


200 test digits, 2-number code, exact $\log p_\theta(x)$

16 numbers: median gap 12 vs 979

An encoder that reads x makes the bound tight

The bound during training



Test digits after each epoch, 2-number code

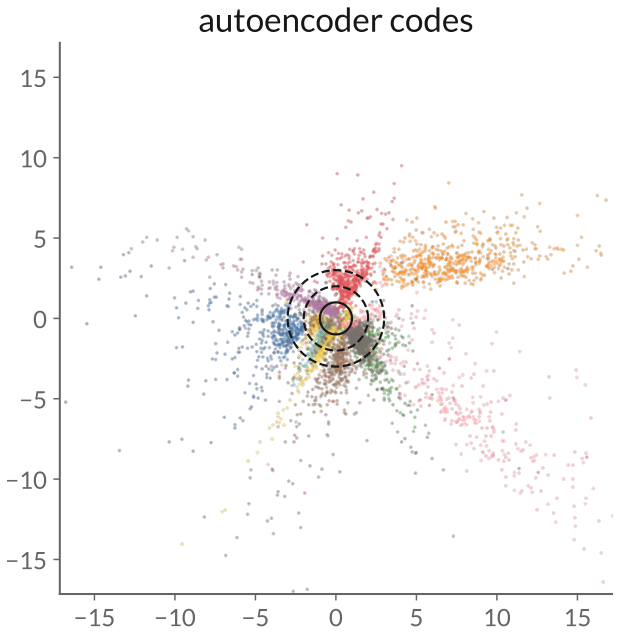
Average gap: 11 after epoch 1, 32 after 30

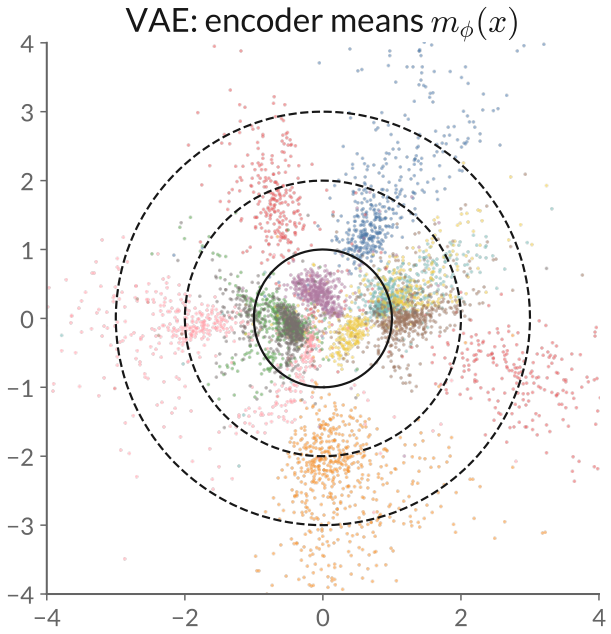
Maximizing the ELBO trained the model

The gap grows on unusual digits whose codes the encoder misses

Two terms, two jobs

$$\text{ELBO} = \underbrace{\mathbb{E}_{q_\phi}\left[\log p_\theta(x \mid z)\right]}_{\text{codes must identify } x} - \underbrace{D_{\text{KL}}\left(q_\phi(z \mid x) \,\|\, p(z)\right)}_{\text{codes must look like prior draws}}$$





2-number codes of test digits

Autoencoder: spread over ± 15

VAE: inside the prior's 1, 2, 3 spreads

Draws from $p(z)$ land where the decoder was trained

The bound from Jensen's inequality

$$\log p_\theta(x) = \log \mathbb{E}_{q_\phi}\left[\frac{p_\theta(x \mid z)\, p(z)}{q_\phi(z \mid x)}\right] \;\geq\; \mathbb{E}_{q_\phi}\left[\log \frac{p_\theta(x \mid z)\, p(z)}{q_\phi(z \mid x)}\right] = \mathrm{ELBO}$$

First equality: $p_\theta(x) = \int p_\theta(x \mid z) p(z)\, dz$, multiplied and divided by q_ϕ

$\log$ is concave: log of an average $\geq$ average of logs

Shorter, but it hides the gap $D_{\mathrm{KL}}(q_\phi \,\|\, p_\theta(z \mid x))$

Average K ratios inside the log: a tighter bound (importance sampling)

$$\log p_\theta(x) \;\geq\; \mathbb{E}\left[\log \frac{1}{K}\sum_{k=1}^{K} \frac{p_\theta(x \mid z^{(k)})\, p(z^{(k)})}{q_\phi(z^{(k)} \mid x)}\right], \qquad z^{(k)} \sim q_\phi(z \mid x)$$

Our 16-number $\log p_\theta(x)$: this estimate, $K = 5{,}000$

Lecture outline

01 Learning by reconstruction

02 From reconstruction to generation

03 The evidence lower bound

04 Training a VAE

05 Trade-offs and limits

06 Discrete codes and learned priors

Gaussian decoder: squared error

One pixel: the Gaussian density with mean μ_j and spread σ

$$\log \mathcal{N}\left(x_j;\, \mu_j, \sigma^2\right) = -\frac{(x_j - \mu_j)^2}{2\sigma^2} - \frac{1}{2}\log\left(2\pi\sigma^2\right)$$

All $D = 784$ pixels, independent given z: add the logs

$$-\log p_\theta(x \mid z) = \underbrace{\frac{1}{2\sigma^2}\left\|x - \mu_\theta(z)\right\|^2}_{\text{squared error, weighted}} + \underbrace{\frac{D}{2}\log\left(2\pi\sigma^2\right)}_{\text{constant if } \sigma \text{ is fixed}}$$

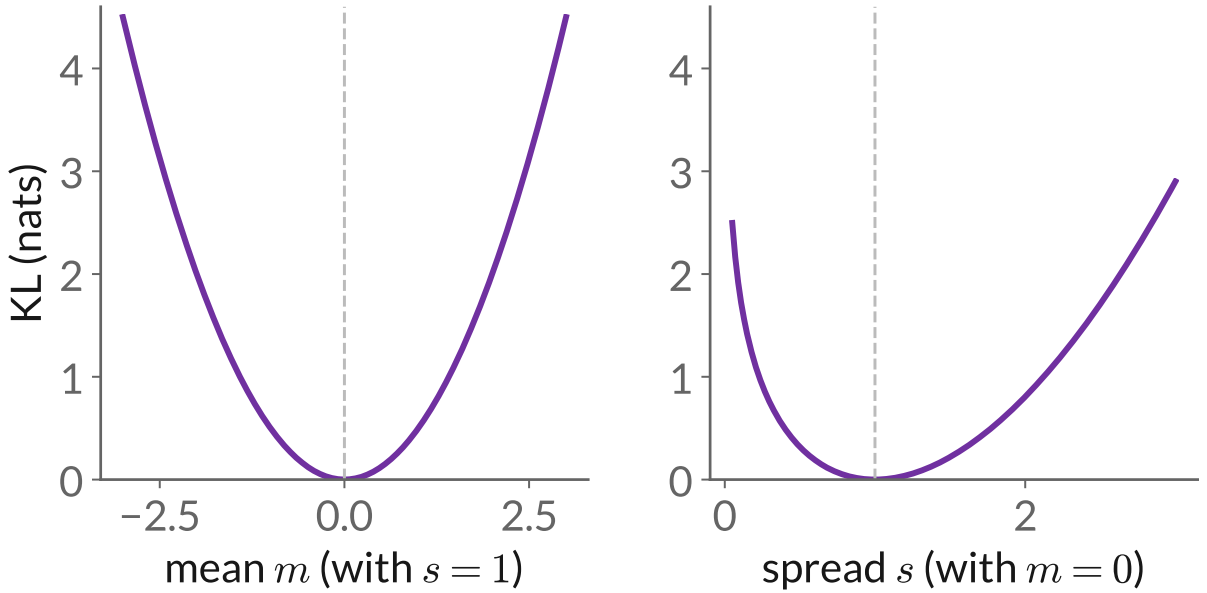
Reconstruction term: the autoencoder's loss, scaled by $1/(2\sigma^2)$

Smaller σ: pixel errors weigh more against the KL term

Ours: σ learned for 2 numbers (0.19); fixed at 0.2 for 16 (Section 05)

KL between Gaussians

$$D_{\mathrm{KL}}\Big(\mathcal{N}\big(m, \operatorname{diag} s^2\big) \,\Big\|\, \mathcal{N}(0, I)\Big) = \frac{1}{2}\sum_{j=1}^{J}\Big(m_j^2 + s_j^2 - 1 - \log s_j^2\Big)$$



m_j, s_j: encoder mean and spread of code number j, of J

m_j^2 pulls each mean toward 0

$s_j^2 - 1 - \log s_j^2$ pulls each spread toward 1; zero at $s_j = 1$

Exact formula: no sampling needed for this term

Deriving the Gaussian KL

One code number; $q = \mathcal{N}(m, s^2)$, $p = \mathcal{N}(0, 1)$; the sum over j follows since both factorize

$$\log q(z) = -\frac{(z-m)^2}{2s^2} - \log s - \tfrac{1}{2}\log 2\pi, \qquad \log p(z) = -\frac{z^2}{2} - \tfrac{1}{2}\log 2\pi$$

Under q: $\mathbb{E}_q[(z-m)^2] = s^2$ and $\mathbb{E}_q[z^2] = m^2 + s^2$

$$D_{\mathrm{KL}}(q \,\|\, p) = \mathbb{E}_q\big[\log q(z) - \log p(z)\big] = -\frac{1}{2} - \log s + \frac{m^2 + s^2}{2} = \frac{1}{2}\big(m^2 + s^2 - 1 - \log s^2\big)$$

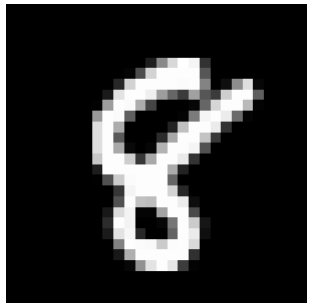
The $\frac{1}{2}\log 2\pi$ terms cancel

Same steps for any two Gaussians; this is the case the VAE needs

One code sample per image

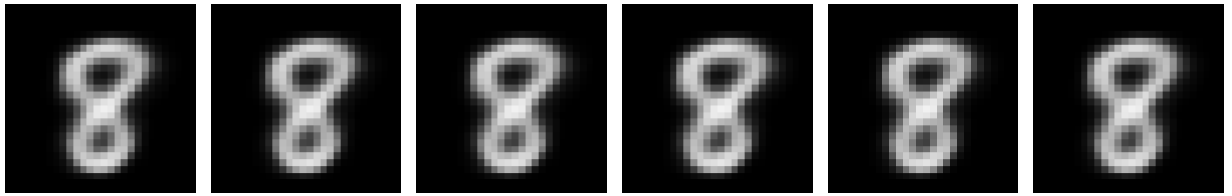
$$\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x \mid z)\right] \approx \log p_\theta(x \mid z), \qquad z \sim q_\phi(z \mid x)$$

test digit



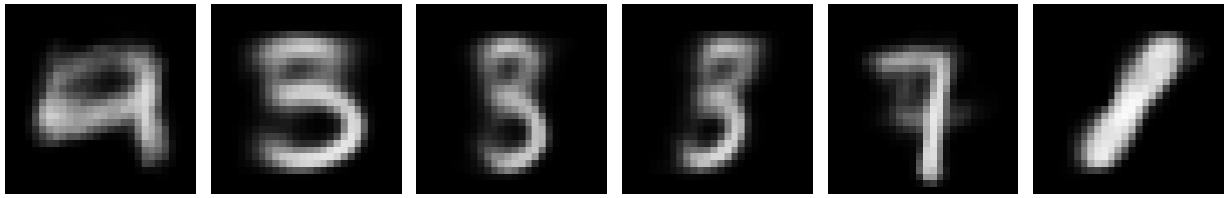
6 codes from $q_\phi(z \mid x)$

$\log p_\theta(x \mid z)$ 214 to 218



6 codes from $p(z)$

$\log p_\theta(x \mid z)$ −695 to −248



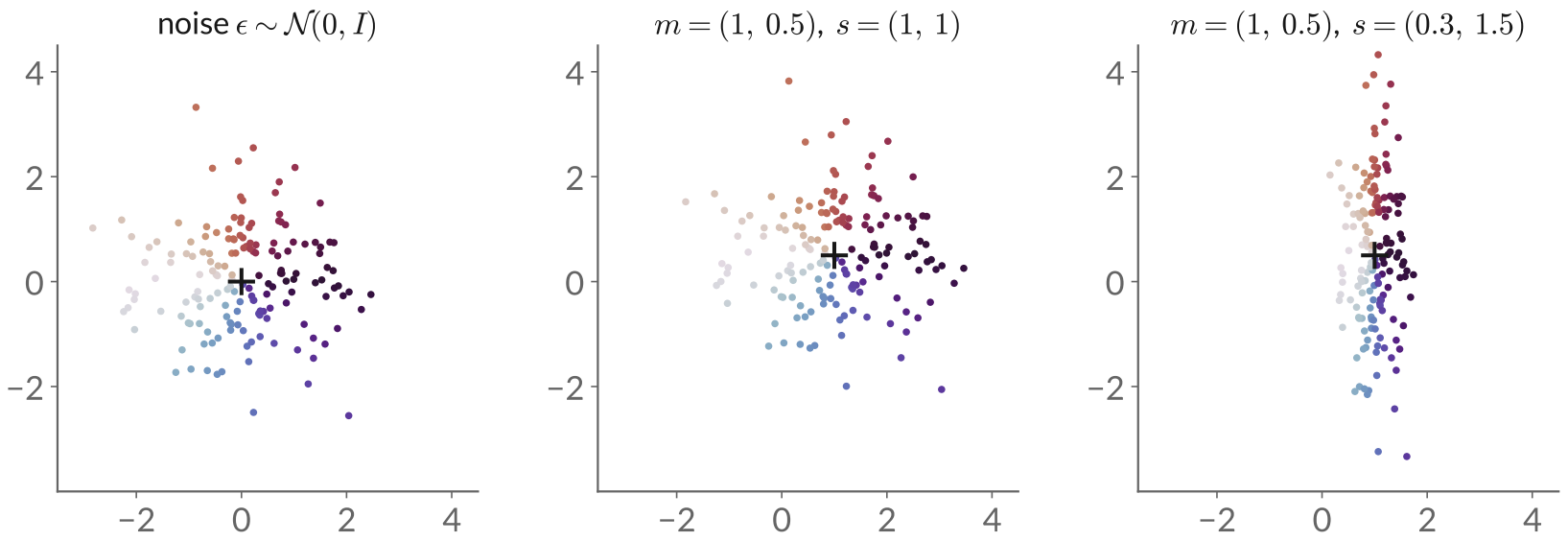
One draw per image and step; 2,000 draws: 214 $\pm$ 4 nats

Noise averages out over minibatches and steps

Codes from q_ϕ all explain x; random prior codes did not

Gradients through a random sample

$$z = m_\phi(x) + s_\phi(x) \odot \underbrace{\varepsilon}_{\text{noise, drawn first}}, \qquad \varepsilon \sim \mathcal{N}(0, I)$$

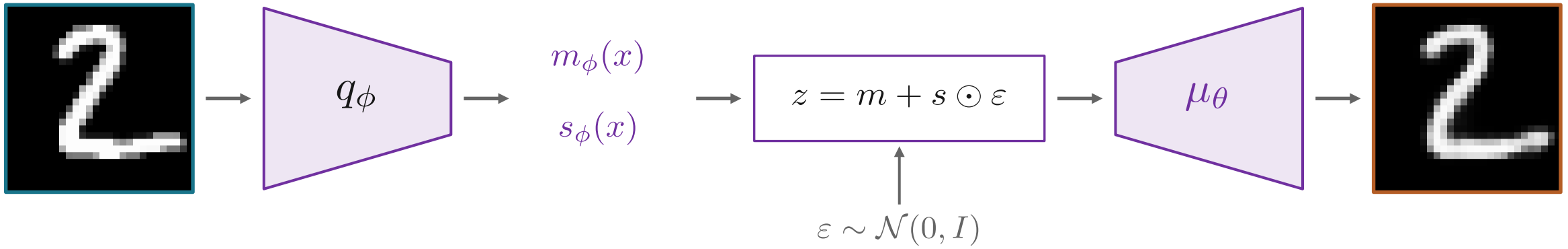


Drawing $z \sim q_\phi$ directly: the draw has no derivative in ϕ

Fix the noise, then shift and scale; $\odot$: number-by-number product

z is differentiable in m_ϕ and s_ϕ: backpropagation works

The complete VAE



$$\ell(x) = \underbrace{\frac{1}{2\sigma^2}\left\|x - \mu_\theta(z)\right\|^2}_{\text{reconstruction}} + \underbrace{\frac{1}{2}\sum_j \left(m_j^2 + s_j^2 - 1 - \log s_j^2\right)}_{\text{KL to the prior}}$$

$\ell(x) = -\mathrm{ELBO}(x)$ up to a constant, one code sample

All differentiable: one loss trains both networks

VAE training algorithm

VAE training

repeat

  minibatch $x^{(1)}, \ldots, x^{(B)}$ from the training data

  encoder: $m_\phi(x^{(b)})$, $s_\phi(x^{(b)})$ for each image

  $\varepsilon^{(b)} \sim \mathcal{N}(0, I)$, $z^{(b)} = m_\phi(x^{(b)}) + s_\phi(x^{(b)}) \odot \varepsilon^{(b)}$

  loss: average of $\ell(x^{(b)})$, reconstruction plus KL

  gradient step on θ and ϕ

until converged

One code per image per step

One loss for both networks

Generation afterwards:

$z \sim \mathcal{N}(0, I)$, output $\mu_\theta(z)$

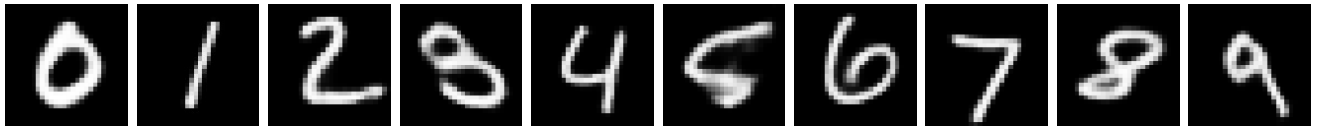
Source: Kingma & Welling, Auto-Encoding Variational Bayes (2014)

Reconstructions and samples

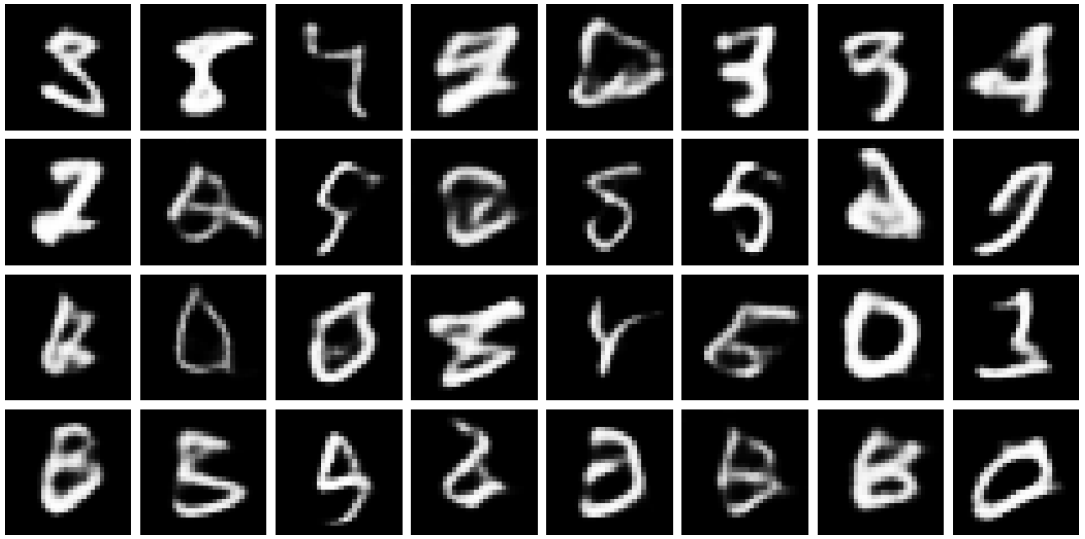
test digit



reconstruction



samples: $z \sim \mathcal{N}(0, I)$, decoded



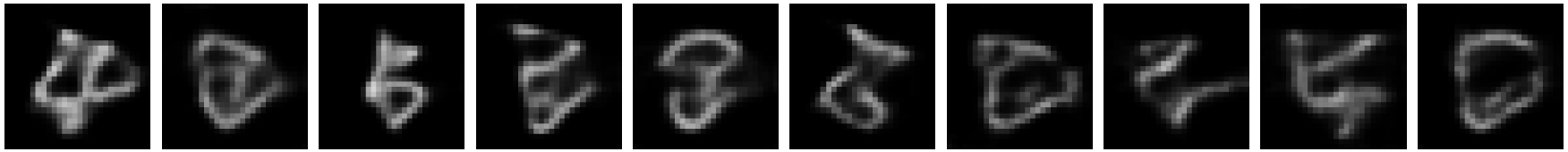
Convolutional VAE, 16 code numbers, $\sigma = 0.2$

Reconstruction error 5.2; same-shape autoencoder 5.0

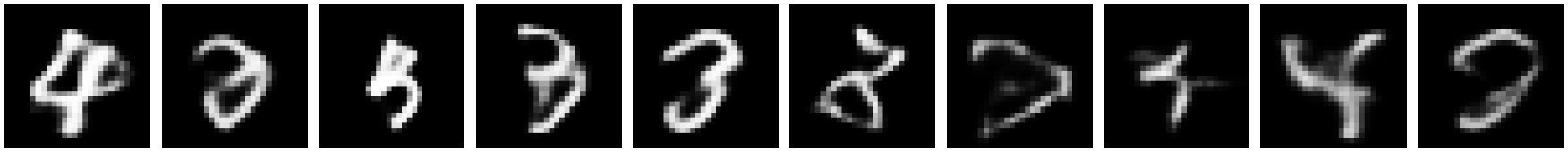
About half are clear digits

Random codes, revisited

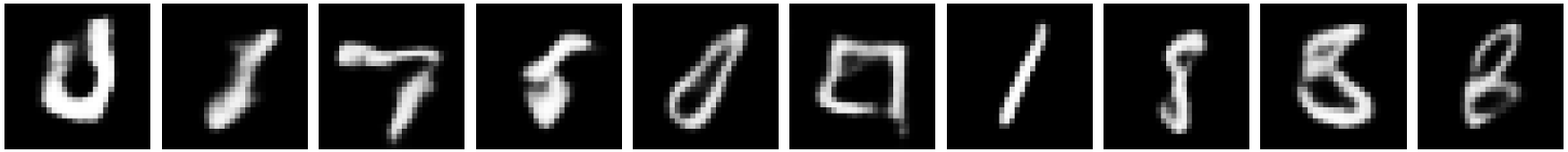
autoencoder,
$\mathcal{N}(0, I)$ draws



autoencoder,
matched to its codes



VAE,
$\mathcal{N}(0, I)$ draws



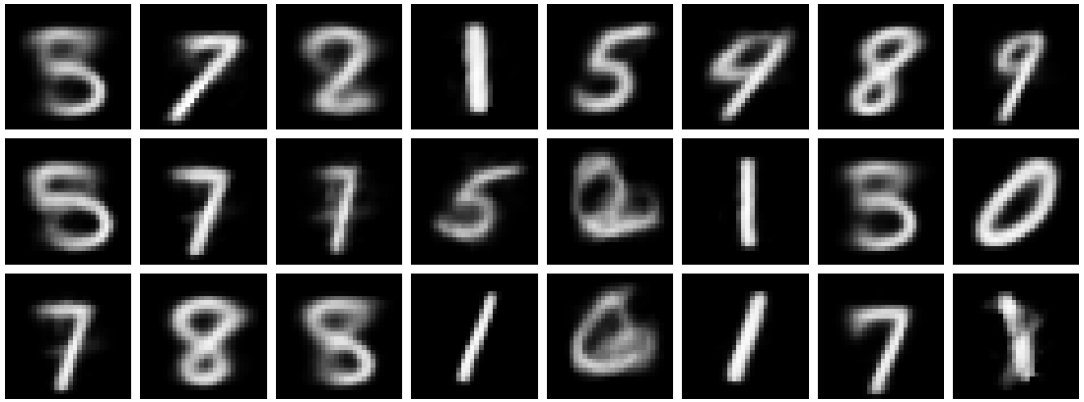
Same draw in each column; 16 numbers; same network shape

Autoencoder codes spread 2.2–3.6 times wider: raw draws decode to faint blobs

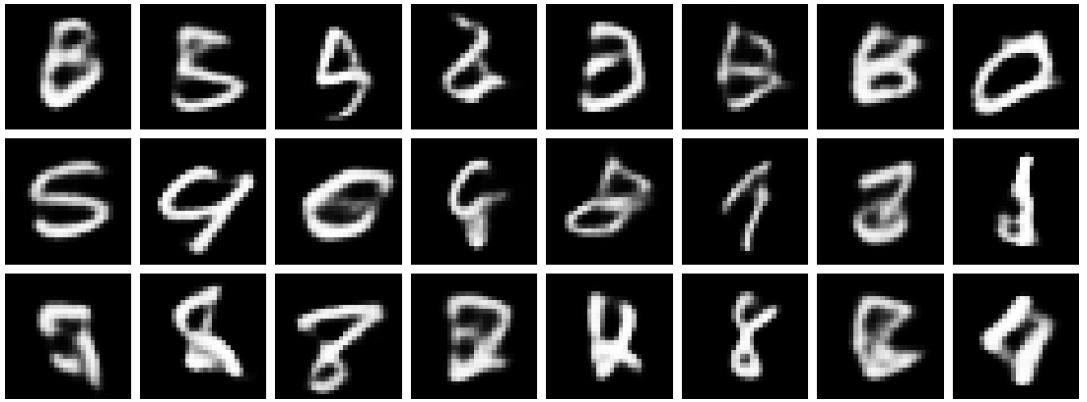
VAE: codes fit $\mathcal{N}(0, I)$ by design; a Gaussian fitted afterwards does about as well

How many code numbers?

VAE samples, 2 code numbers



VAE samples, 16 code numbers

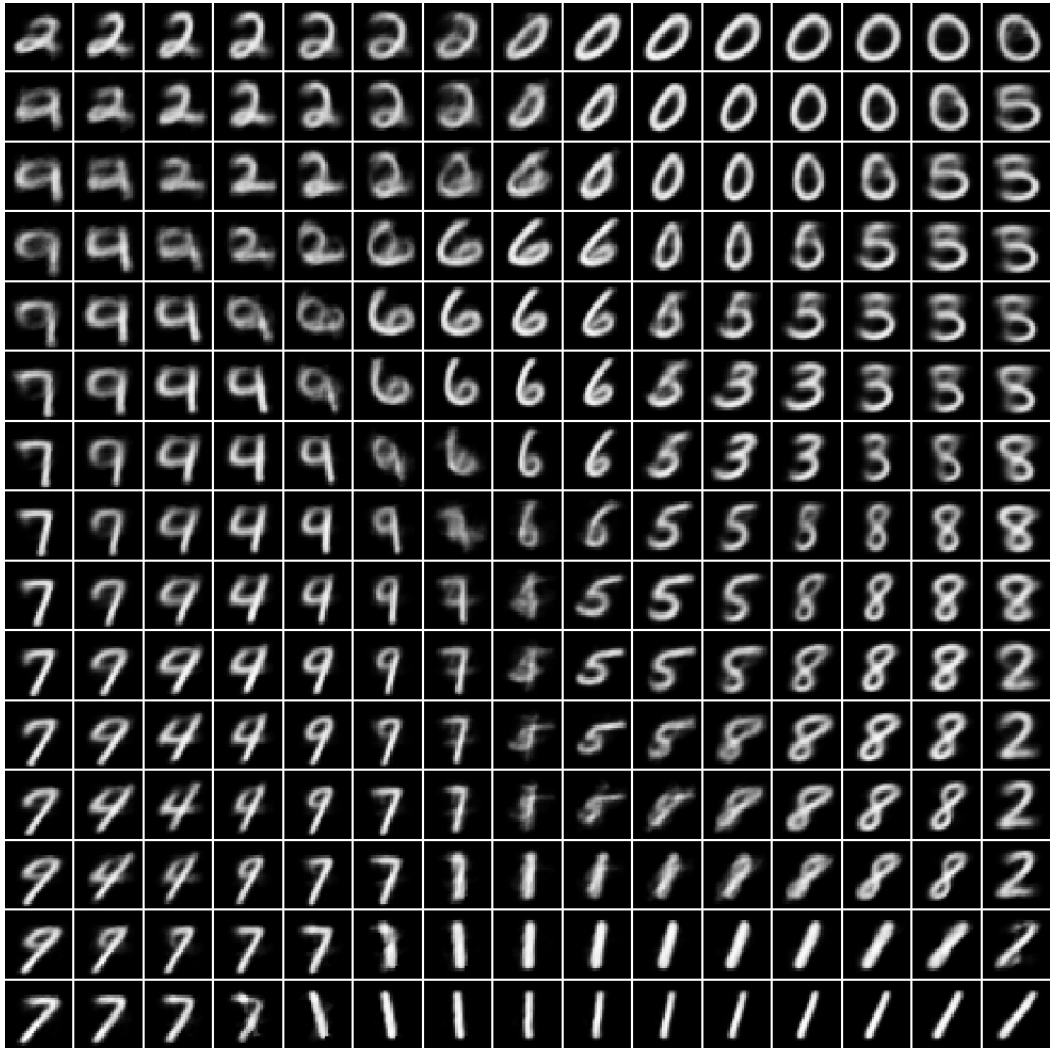


2 numbers: smooth, average-looking digits

16 numbers: more styles, sharper, some failures

More numbers: more detail, more space for the prior to cover

A map of the code space



A 15 × 15 grid of 2-number codes at the prior's quantiles, decoded

Neighbors are similar; a 4 blends into a 9

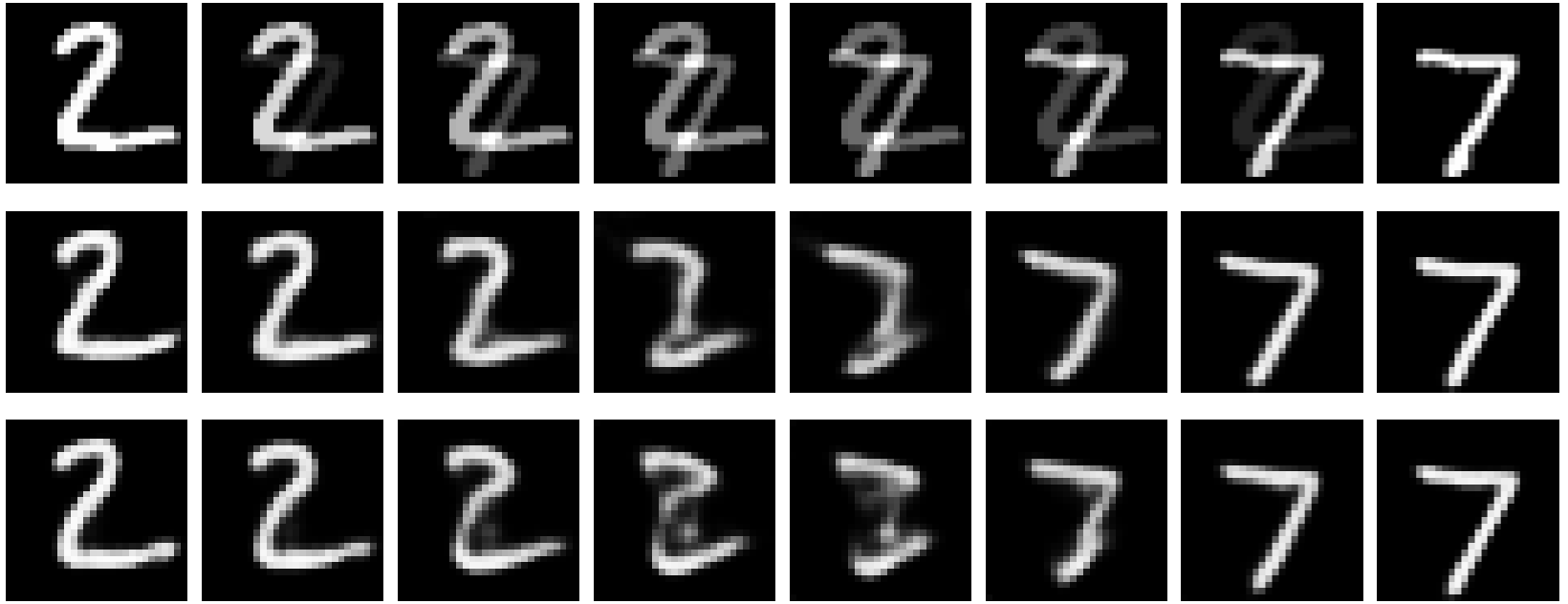
Every point decodes to a plausible image

Walking between codes

pixel average

autoencoder

VAE



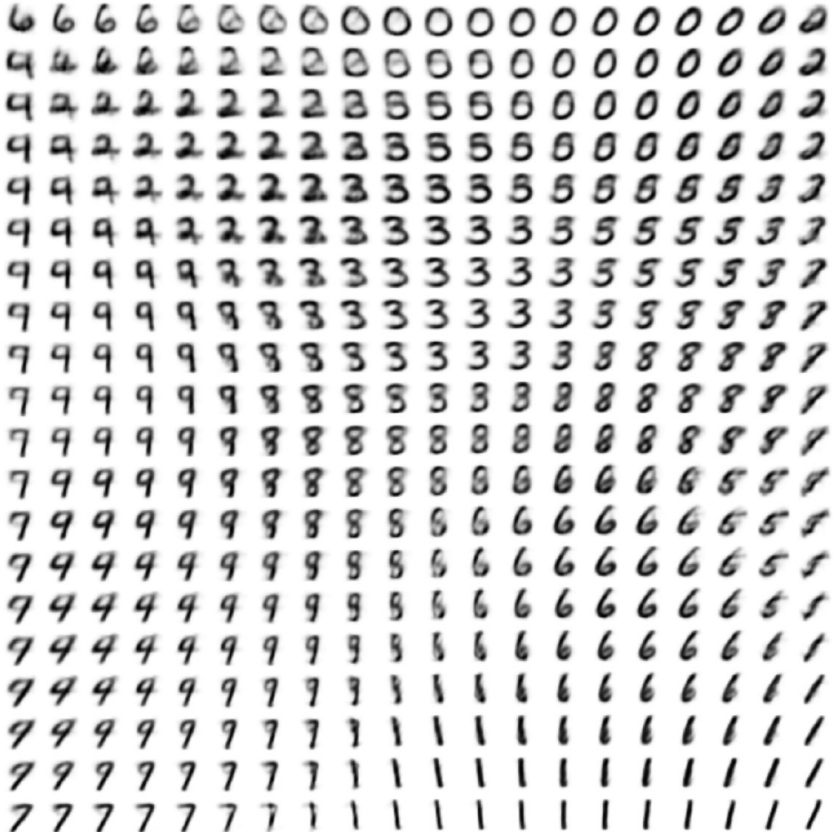
From a 2 to a 7: mix pixels, or mix codes and decode

Pixels: two faded digits overlaid; codes: one stroke that bends

Straight lines in code space are paths through digits

Published results





Kingma and Welling (2014): decoded 2-number grids; Frey faces (left), MNIST (right)

As in our map: pose, expression or digit identity change smoothly

Source: Kingma & Welling, Auto-Encoding Variational Bayes (2014)

Lecture outline

01 Learning by reconstruction

02 From reconstruction to generation

03 The evidence lower bound

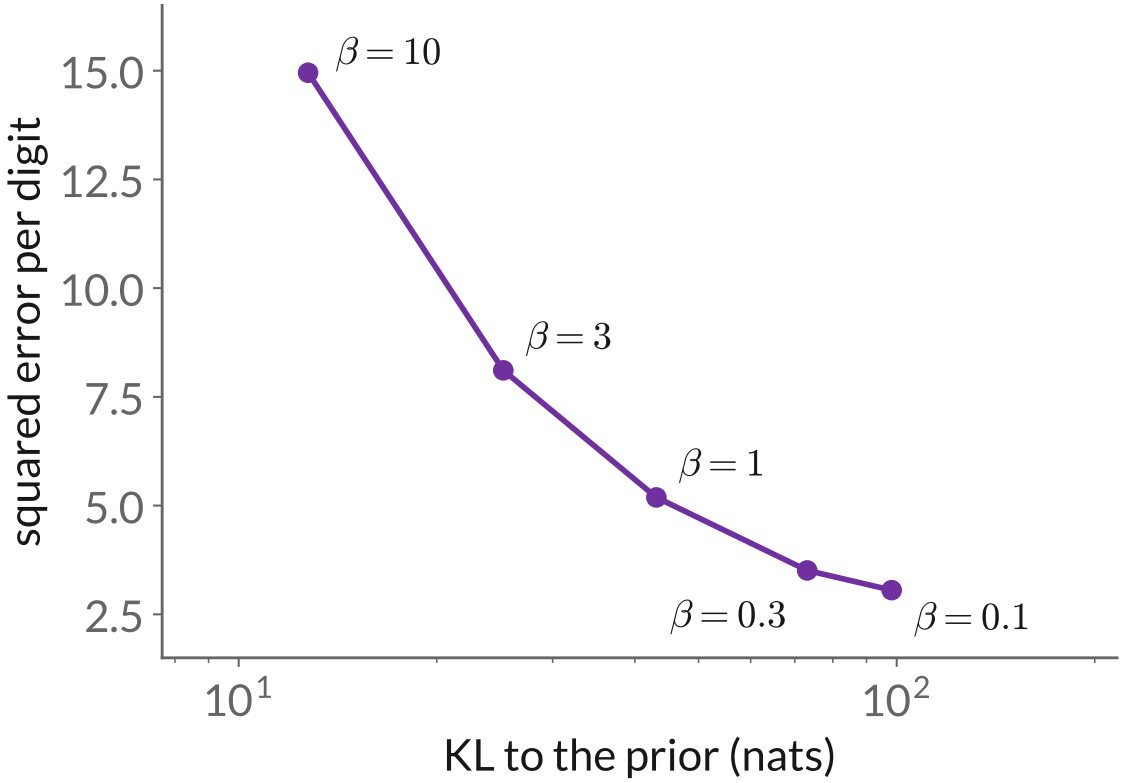
04 Training a VAE

05 Trade-offs and limits

06 Discrete codes and learned priors

Weighting the KL term

$$\mathcal{L}_\beta(x) = \mathbb{E}_{q_\phi}\left[\log p_\theta(x \mid z)\right] - \underbrace{\beta}_{\text{the weight}} D_{\mathrm{KL}}\left(q_\phi(z \mid x) \,\|\, p(z)\right)$$



$\beta = 1$: the ELBO; $\beta > 1$: a β-VAE

Five VAEs, 32 code numbers, $\sigma = 0.2$

Larger β acts like larger σ

One knob: reconstruct well, or match the prior

Source: Higgins et al., beta-VAE (ICLR 2017)

Reconstruction or prior match



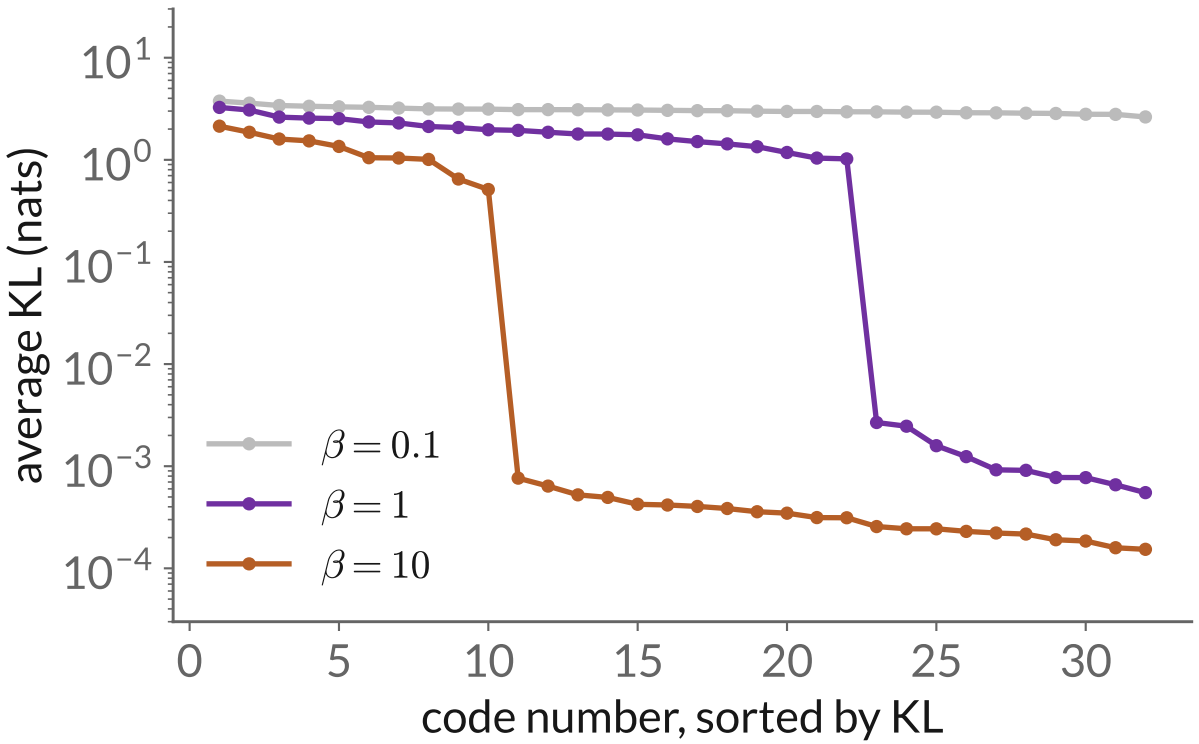
Small β: sharp reconstructions, broken samples

Large β: digit samples, blurrier reconstructions

Good samples need the training codes to fill the prior

Learned σ in our 16-number VAE: 0.076, like a small β; so we fixed $\sigma = 0.2$

Unused code numbers



KL near 0: $q_\phi(z_j \mid x) \approx \mathcal{N}(0, 1)$ for every digit, so z_j carries nothing

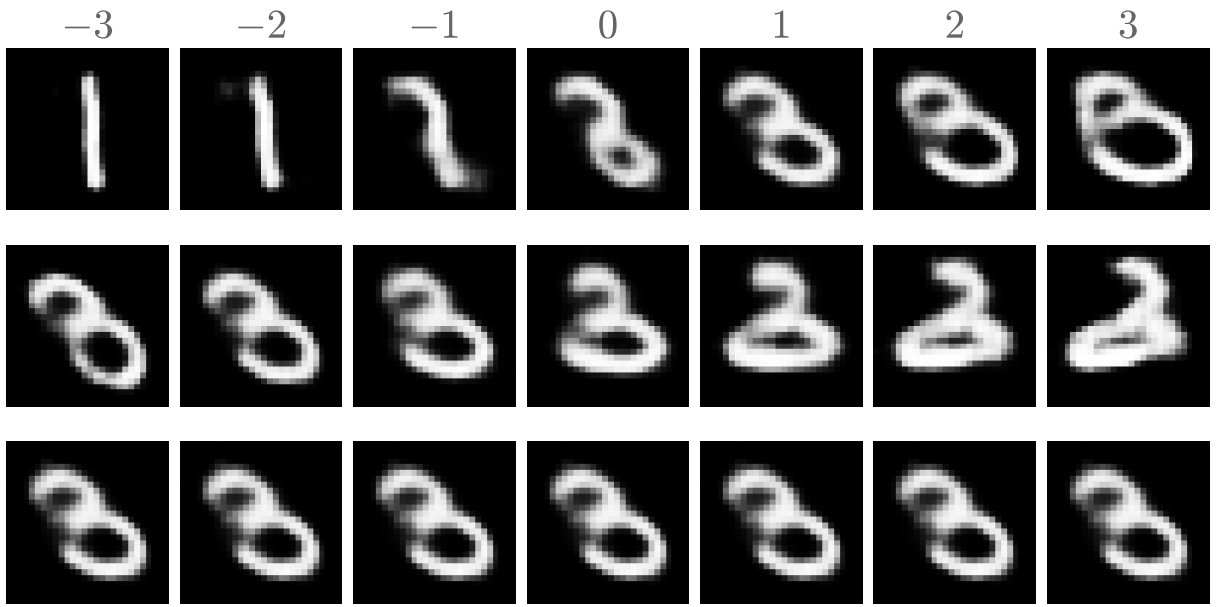
In use: 22 at $\beta = 1$, 10 at $\beta = 10$

The KL term switches off numbers the decoder can do without

In use: encoder mean m_j varies across digits (variance above 0.01)

Posterior collapse

One test digit's code ($\beta = 10$); move one number from -3 to 3



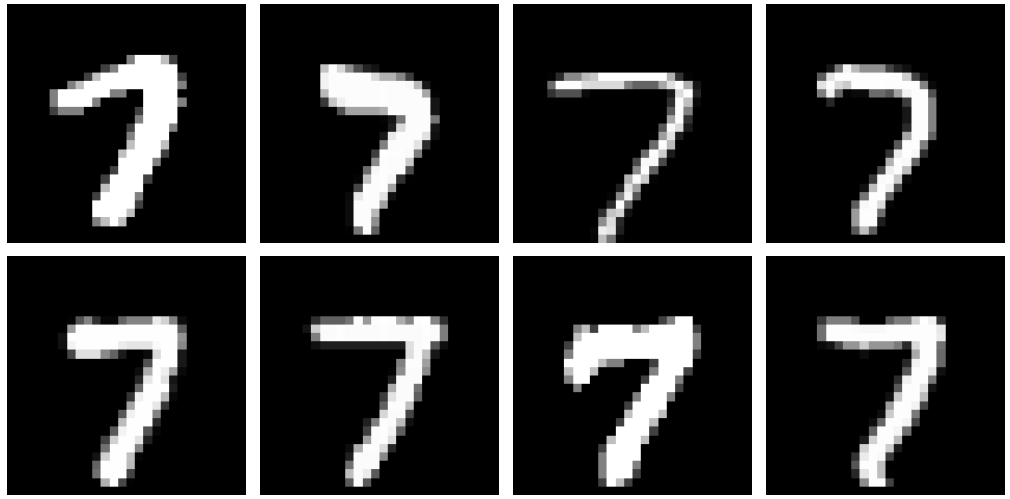
Used numbers change the stroke; an unused one changes nothing

Posterior collapse: $q_\phi(z_j \mid x) = p(z_j)$; in the extreme, for every j

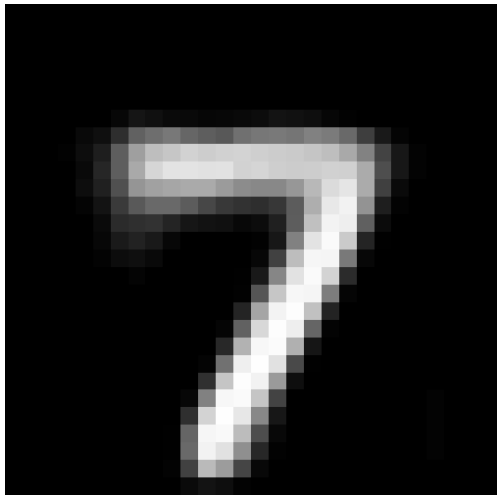
A decoder that can model x alone stops listening to z

Why VAE samples blur

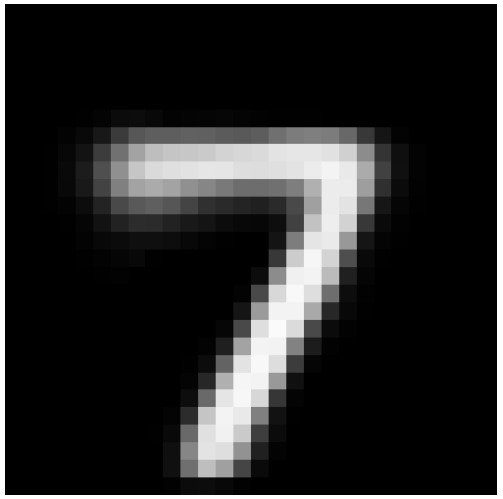
training digits with the nearest codes (8 of 30)



their pixel average



decoder output



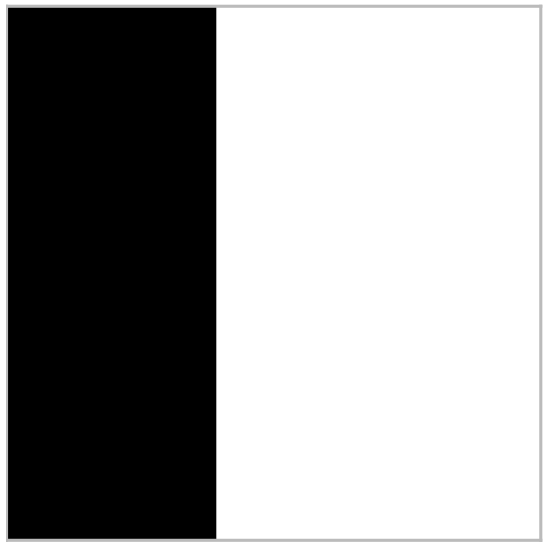
2-number VAE; one code, and the 30 training digits encoded nearest to it

Squared distance from the output: 1 to their average, 18 to a typical one

The decoder outputs the average of digits that share its code

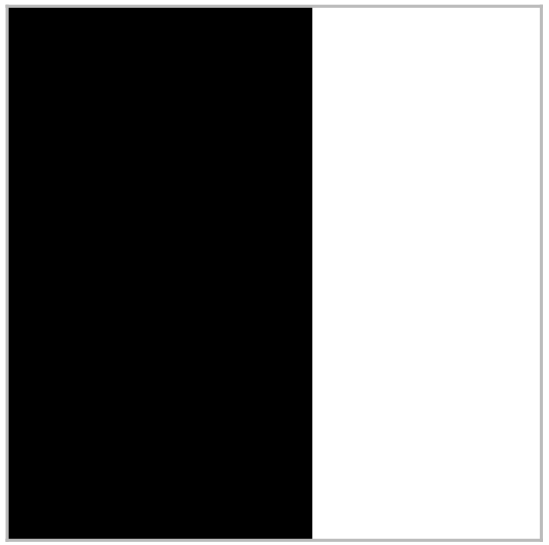
Two plausible edges

plausible edge 1



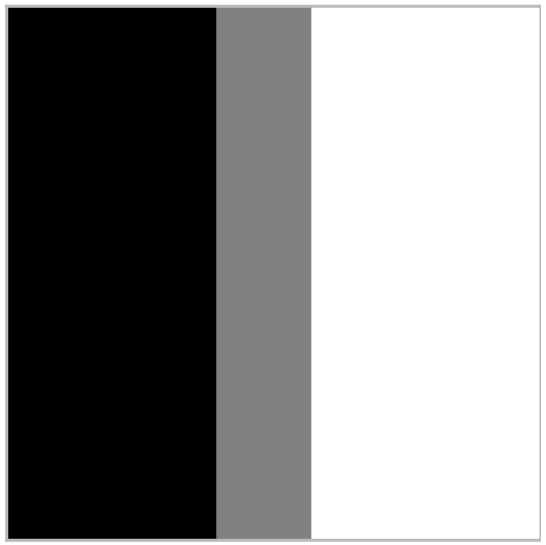
expected error 70

plausible edge 2



expected error 70

their mean



expected error 35

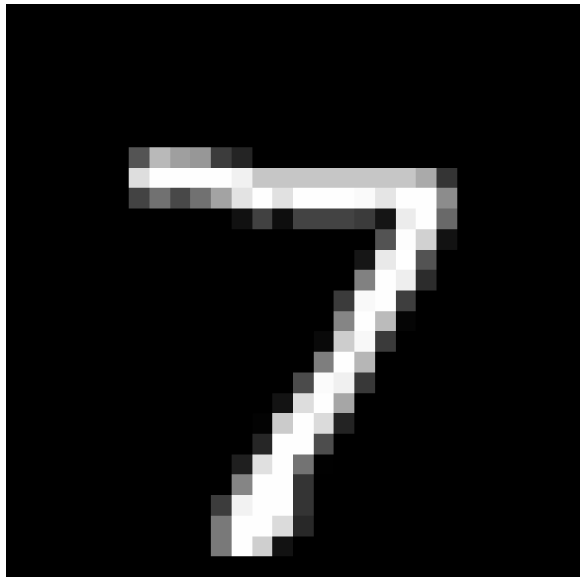
One code; its training images show the edge at either place, half the time each

Expected error: squared error of that image as the output, averaged over the two edges

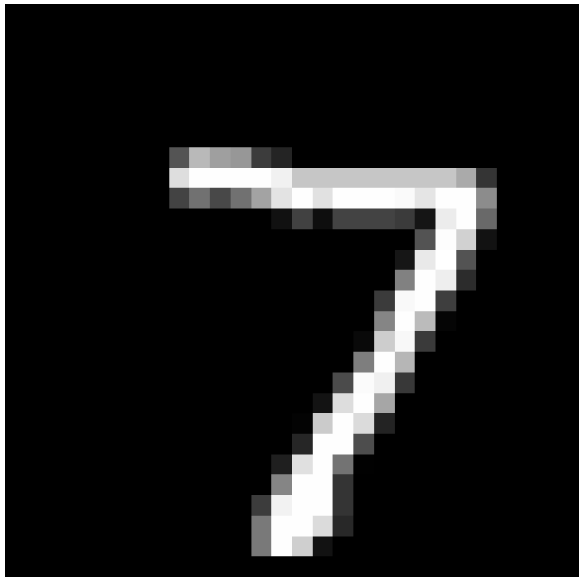
Squared error picks their mean: $\arg\min_{\hat{x}} \mathbb{E}\|x - \hat{x}\|^2 = \mathbb{E}[x]$

Pixel error and similarity

test digit

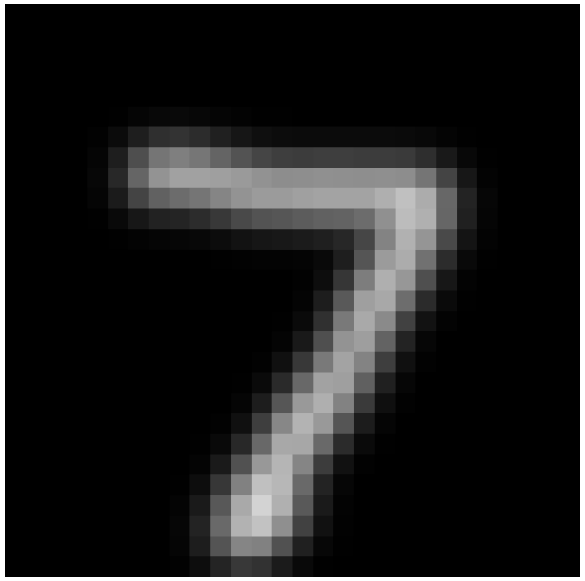


shifted 2 pixels



squared error 57

blurred



squared error 9

The shifted 7 is the same digit, with 6 times the blurred one's error

Sharper samples need other losses (L16) or stronger priors (next)

Lecture outline

01 Learning by reconstruction

02 From reconstruction to generation

03 The evidence lower bound

04 Training a VAE

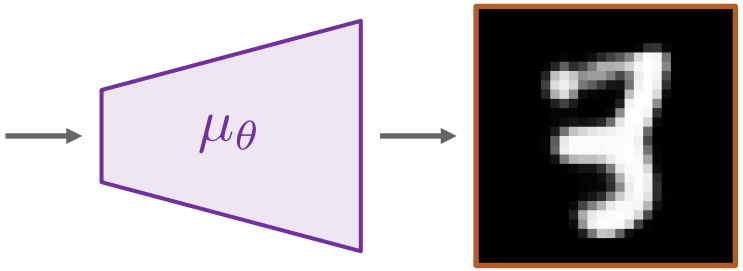
05 Trade-offs and limits

06 Discrete codes and learned priors

A fixed prior, or a learned one?

Gaussian VAE

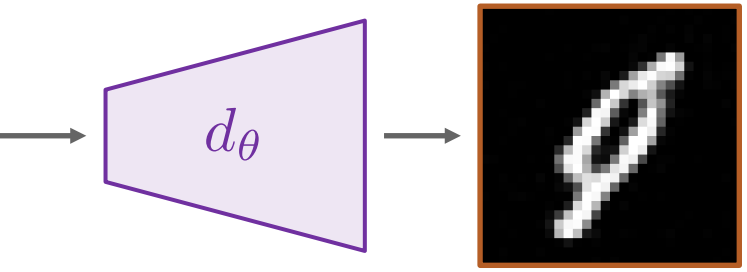
$z \sim \mathcal{N}(0, I)$, fixed



Tokens and a learned prior

Transformer

63	34	43	43	45	13	2
26	19	27	41	26	24	2
35	55	63	62	35	26	2
37	29	51	26	14	37	2
26	19	47	35	35	43	28
49	1	6	51	38	27	28
26	26	35	26	26	26	59



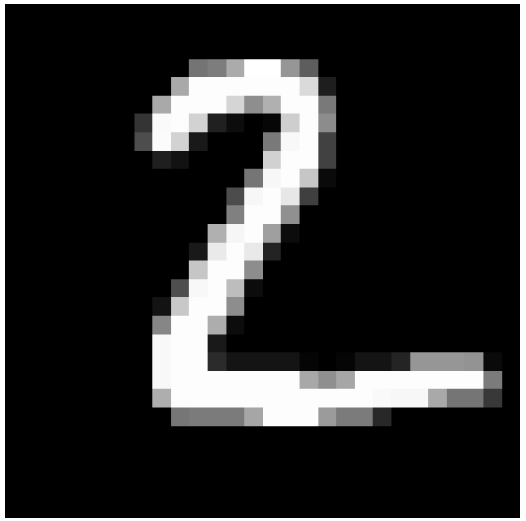
$\mathcal{N}(0, I)$ is fixed: codes are squeezed to fit (Section 05)

Language models (L10, L12) learn token distributions well

Idea: an image as a grid of tokens; learn the prior

Images as grids of tokens

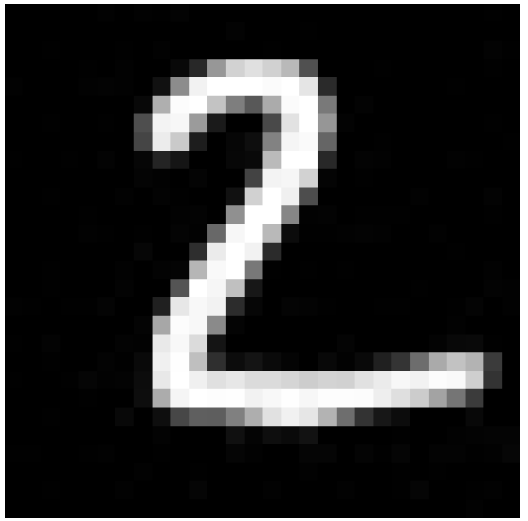
test digit



7 × 7 token indices (0 to 63)

63	34	43	43	45	13	2
26	19	27	41	26	24	2
35	55	63	62	35	26	2
37	29	51	26	14	37	2
26	19	47	35	35	43	28
49	1	6	51	38	27	28
26	26	35	26	26	26	59

decoded



Encoder: a 7×7 grid of 16-number vectors

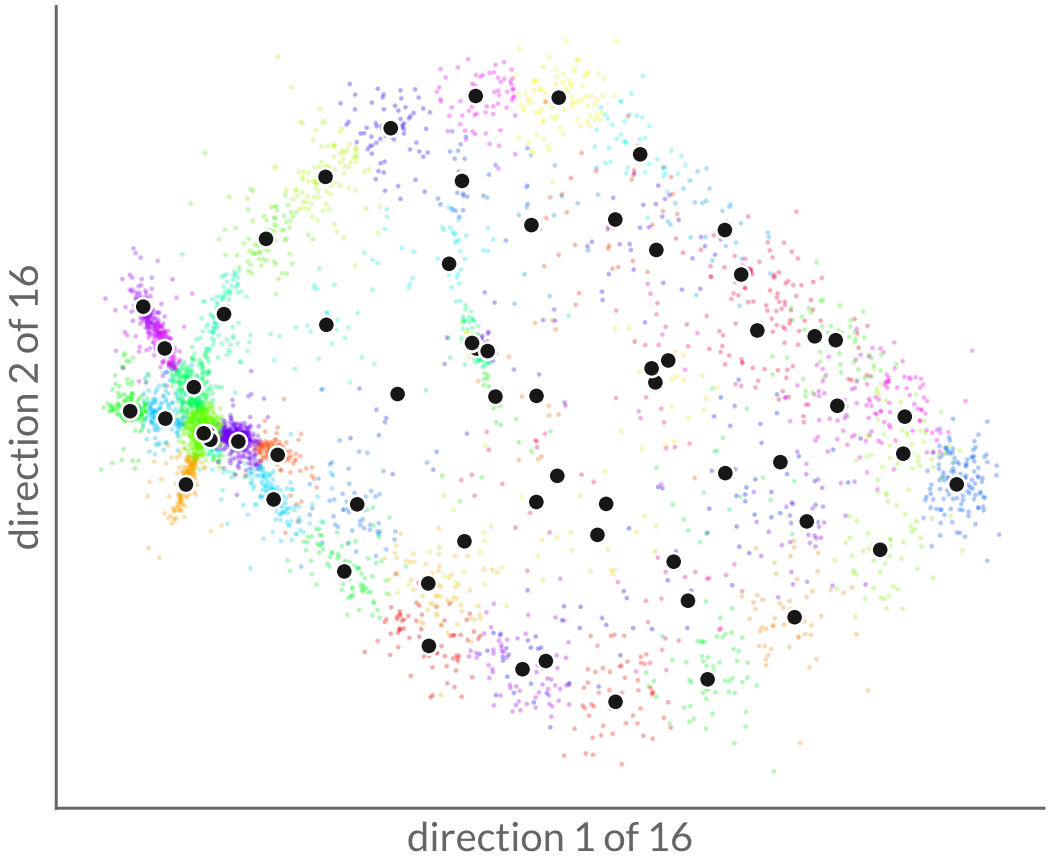
Each replaced by the nearest of 64 learned vectors; keep its index

Decoder: rebuilds the digit from 49 indices

An image becomes 49 tokens from a vocabulary of 64

A learned codebook

$$k_i = \arg\min_k \left\| z_e(x)_i - e_k \right\|^2, \qquad z_q(x)_i = e_{k_i}$$



$z_e(x)_i$: encoder vector at grid cell i

$e_1, \ldots, e_{64}$: codebook entries, learned

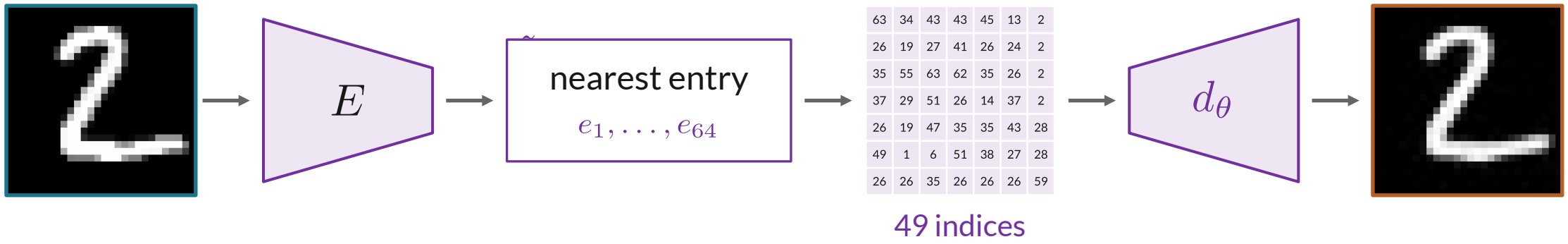
k_i : nearest entry's index; z_q : grid of entries

Plot: black, entries; colored, encoder vectors by nearest entry (2-D projection)

Each entry stands for nearby encoder vectors

Source: van den Oord et al., Neural Discrete Representation Learning (2017)

VQ-VAE



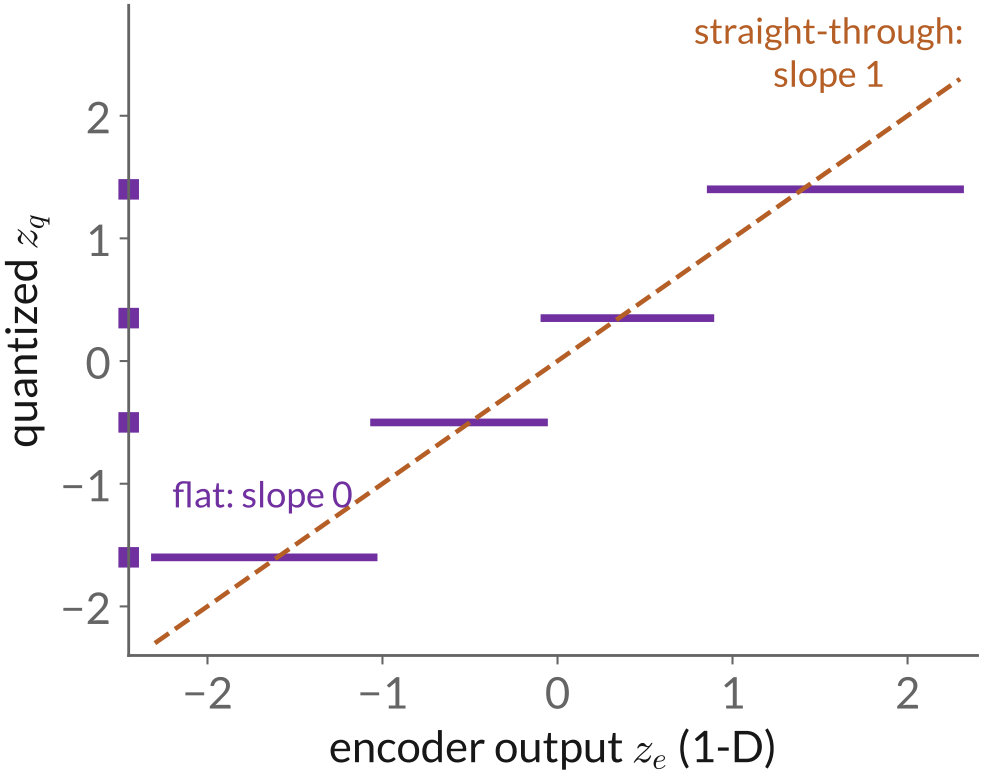
Encoder, decoder, squared error as before; a lookup between them

No noise, no KL; a prior over indices is learned afterwards

Open: training through the lookup, and training the entries

Source: van den Oord et al., Neural Discrete Representation Learning (2017)

No gradient through the nearest entry



One number instead of 16: chosen entry vs encoder output

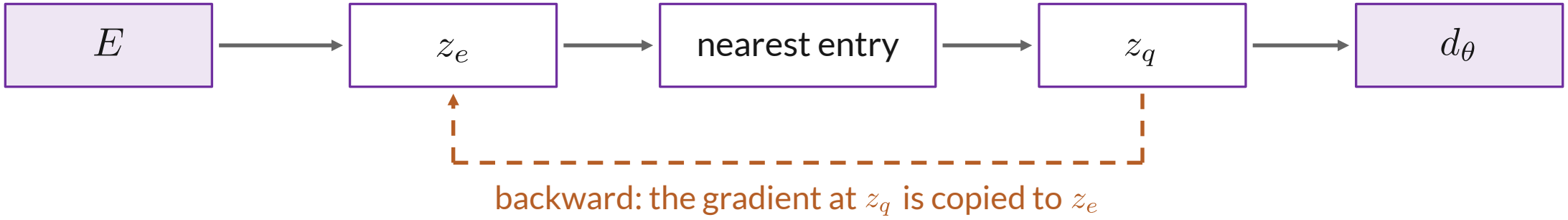
A staircase: slope 0 except at the jumps

Backpropagation through the lookup gives the encoder no gradient

Straight-through estimation

$$z_q = z_e + \underbrace{\mathrm{sg}\left[z_q - z_e\right]}_{\text{stop-gradient: value kept, gradient zero}}$$

forward: the value of z_q



Works because z_q stays close to z_e: next, a loss that keeps it so

Codebook and commitment losses

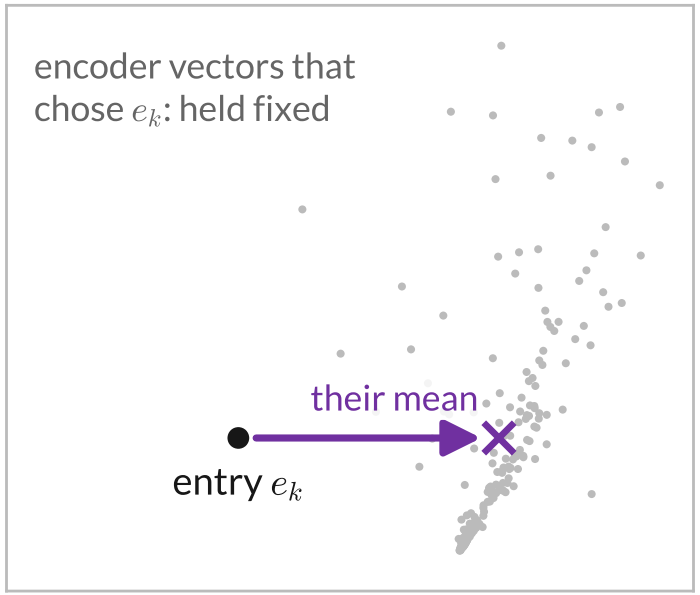
$$\ell = \underbrace{\|x - d_\theta(z_q)\|^2}_{\text{reconstruction}} + \underbrace{\|\text{sg}[z_e] - z_q\|^2}_{\text{codebook}} + \underbrace{\beta\,\|z_e - \text{sg}[z_q]\|^2}_{\text{commitment}}$$

reconstruction
trains decoder, encoder

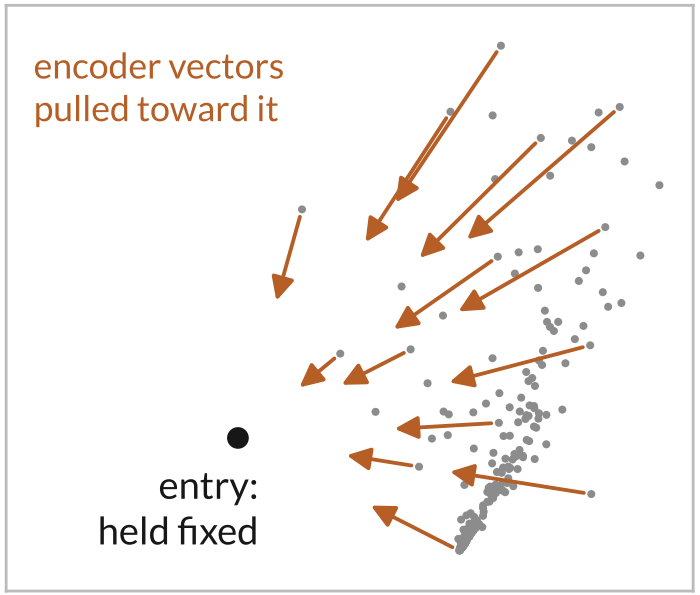
codebook
trains the entries only

commitment
trains the encoder only

codebook loss: the entry moves



commitment loss: the encoder moves



$\text{sg}[\cdot]$: held fixed

Same distance; a different side moves

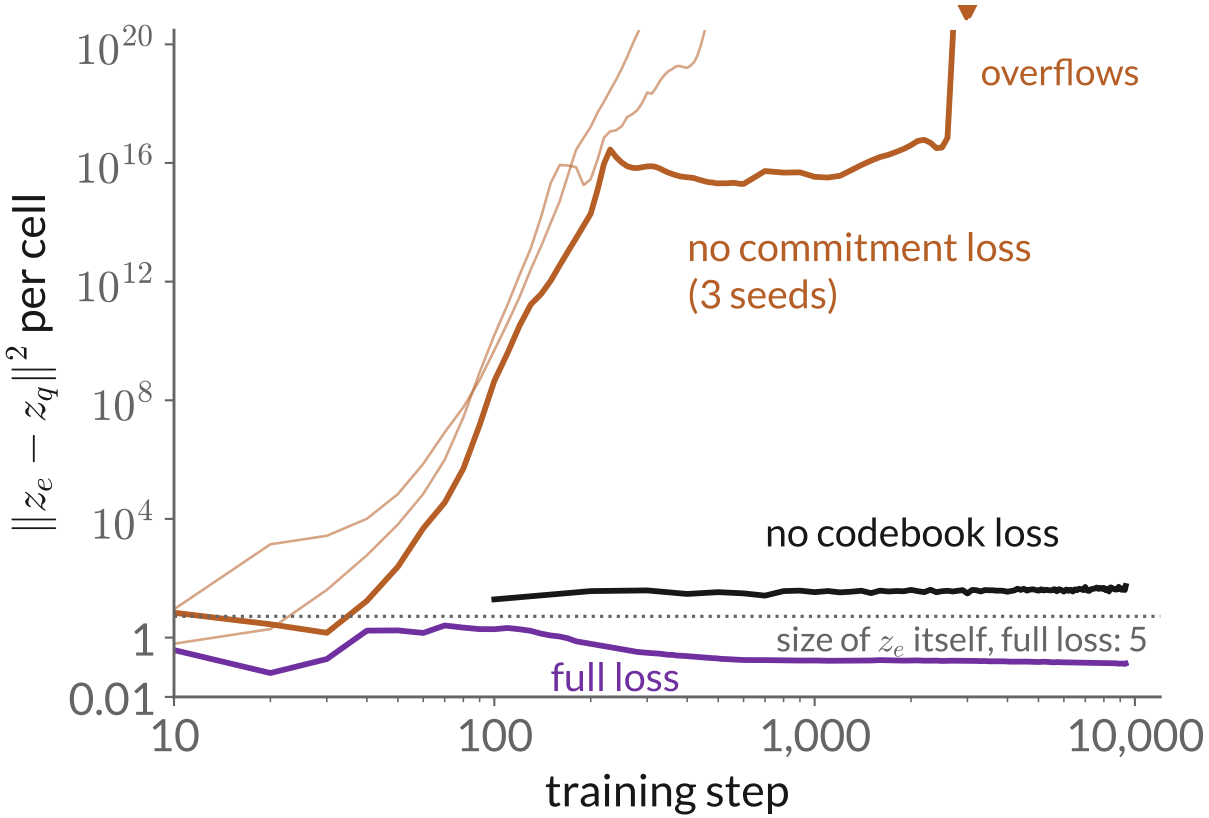
Codebook: the entry goes to the mean of its vectors (k-means)

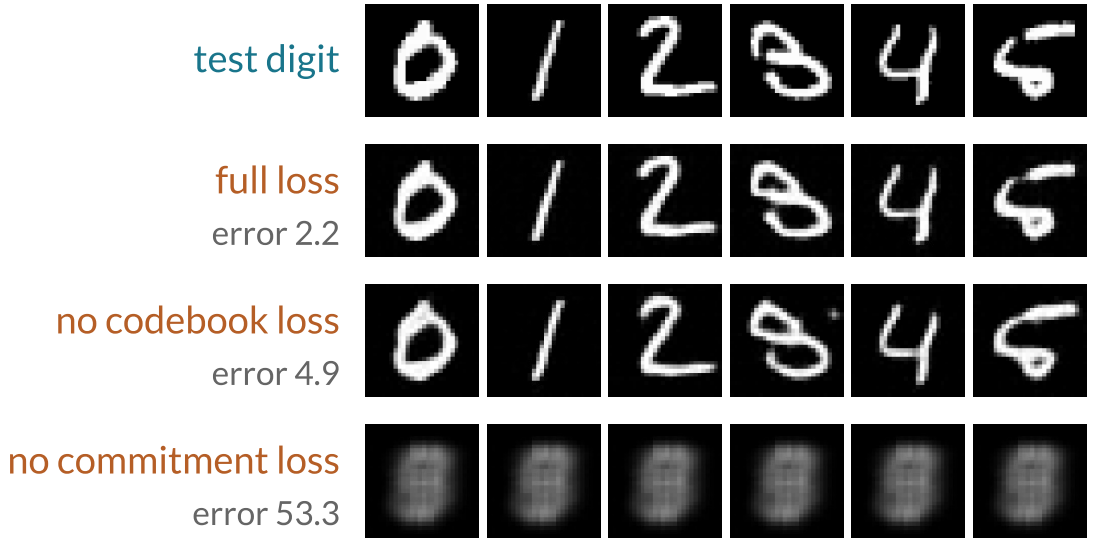
$\beta = 0.25$; values 0.1 to 2 work

Our model at step 100; 2 of 16 directions

Source: van den Oord et al., Neural Discrete Representation Learning (2017)

Removing either loss





Same tokenizer and seed; one loss removed

No commitment: z_e grows; one entry for every cell

No codebook loss: the entries stay where they start

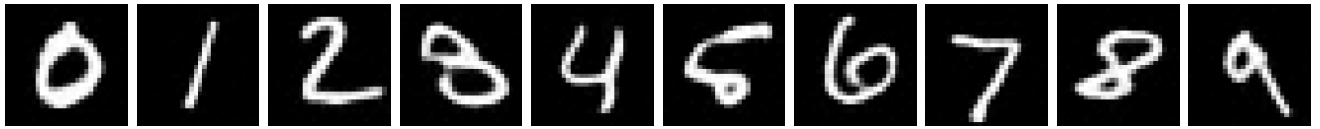
Straight-through needs $z_q \approx z_e$: each loss keeps one side close

Reconstructions from 49 tokens

test digit

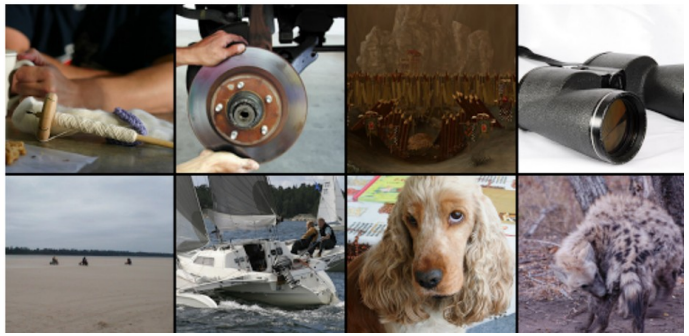


from 49 tokens



Error 2.2 per digit; the 16-number VAE: 5.2

49 indices of 6 bits each: 294 bits per digit

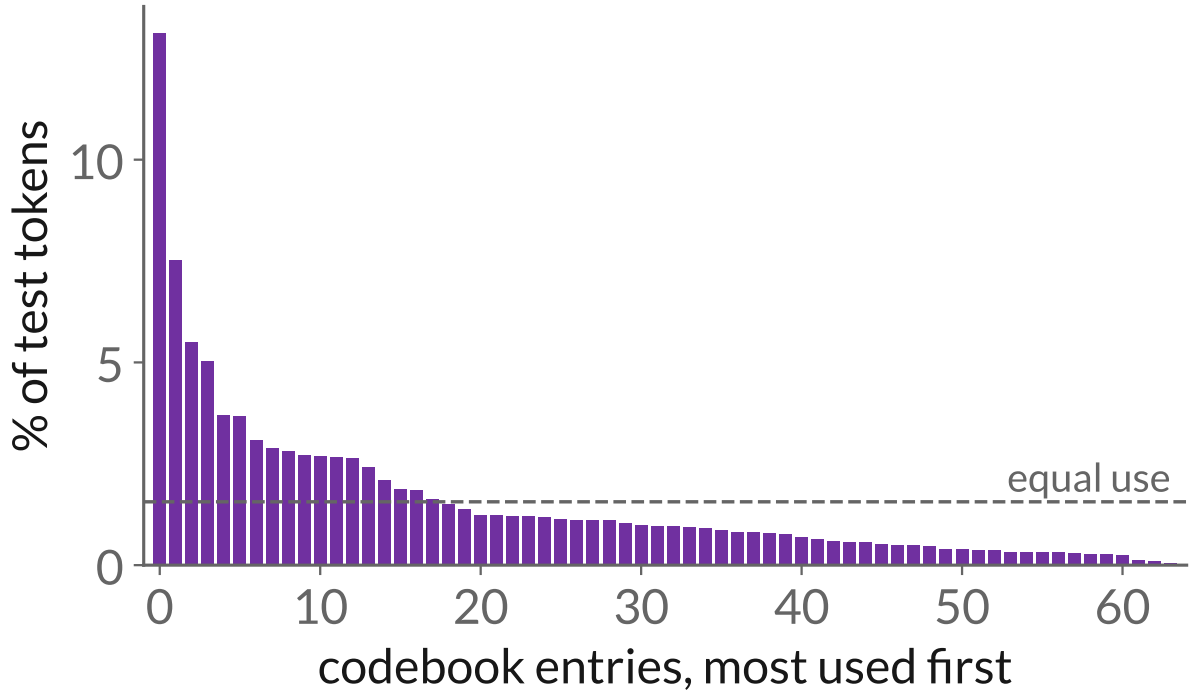




Published: ImageNet 128×128 (left) and reconstructions from 32×32 tokens, $K = 512$ (right)

Source: van den Oord et al., Neural Discrete Representation Learning (2017)

Which codes are used



Each of 64 entries, over all cells of 10,000 test digits

All in use; the top one takes 13% (mostly background)

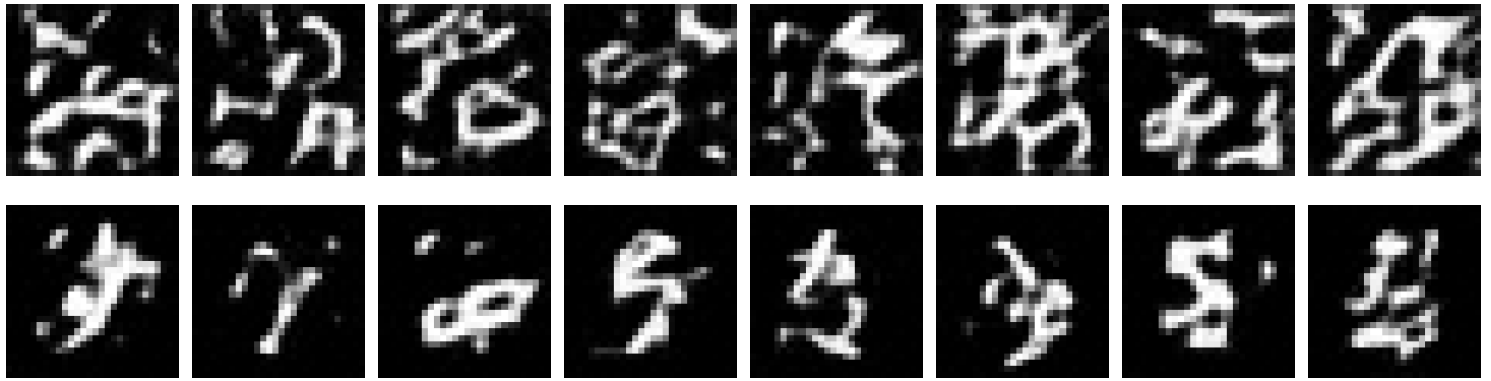
Tokens are far from uniform

Larger models often have unused entries

Random tokens do not make a digit

uniform tokens

each position from its own frequencies



Uniform: every entry equally likely in every cell

Per-position frequencies: each cell from its own histogram, independently

Cells must agree: the prior has to model dependencies

A Transformer prior over tokens

$$p_\psi(k_1, \ldots, k_{49}) = \prod_{i=1}^{49} p_\psi\left(k_i \mid k_1, \ldots, k_{i-1}\right)$$

9 tokens known

63	37	37	37	34	29	28
37	37	?				

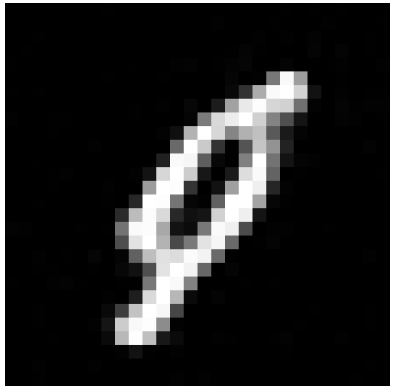
24 tokens known

63	37	37	37	34	29	28
37	37	34	22	54	13	48
37	34	15	27	7	45	2
26	26	62	?			

40 tokens known

63	37	37	37	34	29	28
37	37	34	22	54	13	48
37	34	15	27	7	45	2
26	26	62	44	32	24	2
35	39	23	38	26	22	2
26	43	62	45	22	?	

all 49, decoded

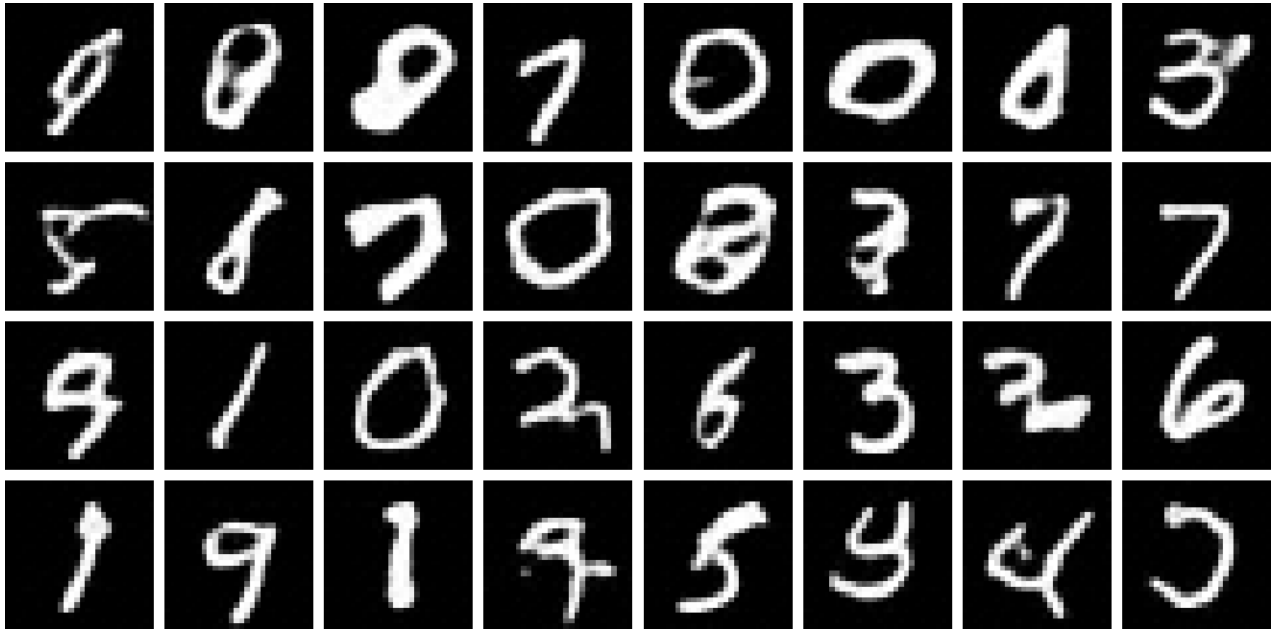


Raster order; a small causal Transformer (L12) predicts the next index

Trained on the 60,000 training digits' tokens; tokenizer frozen

Test: 1.48 nats per token, 105 bits per digit (uniform: 294)

Generating with VQ-VAE



1. Sample 49 indices from the Transformer, one at a time
2. Look up their entries
3. Decode once

Our samples, not selected

Most are clean digits, unlike random tokens

Larger tokenizers and priors



VQ-VAE-2: two token grids per 256×256 image, 32×32 (layout) and 64×64 (detail)

Autoregressive priors over tokens, given the class; above: ImageNet samples

Source: Razavi et al., VQ-VAE-2 (2019)

Tokenizers beyond this lecture

L16	**VQGAN**	VQ-VAE with an adversarial loss; Transformer prior
L18	**Latent diffusion**	autoencoder compresses; diffusion generates codes
Also	**Token-based generators**	image tokens in one sequence with text tokens

Same split as today: compress, then learn a distribution over codes

Three ways to get new codes

autoencoder, fitted Gaussian

VAE, Gaussian prior

VQ-VAE, Transformer prior



Autoencoder: a Gaussian fitted to its codes afterwards

VAE: a fixed Gaussian, made to fit by the KL term

VQ-VAE: discrete tokens, prior learned from data

The learned prior gives the cleanest samples

Summary

- Autoencoders: compact codes by reconstruction; the bottleneck forces shared structure
- Generation needs $p_\theta(x)$; random prior codes rarely explain a given image
- An encoder proposes codes; $\log p_\theta(x) \geq \mathrm{ELBO}$, with the encoder's mismatch as the gap
- Training: squared error, closed-form KL, one reparameterized sample
- KL weight: reconstruction vs samples; squared error makes samples blur
- VQ-VAE: images as tokens, with a learned Transformer prior

Next time: generative adversarial networks

A different loss

A learned critic, telling real from generated, replaces the pixel error

No likelihood

The generator learns to fool the critic: a two-player game

Builds on today: the decoder as generator, blur, VQ tokens (VQGAN)