Autoencoders and representation learning

Fashion-MNIST

LECTURE 11 LECTURE NOTES

Applications of Machine Learning

BSc Mathematics of Artificial Intelligence · Third year

Where we are

Lecture	What it built	What it needed
7	a first network, and what a layer computes	a label for every image
8	the training loop, written out in five lines	a label for every image
9	a harmful-content detector, designed end to end	a label for every post
10	an audit of what a model relies on, and for whom it fails	a label for every row
11	**a network that learns from the images alone**	**no labels**

OUTSIDE THE BOOK

This lecture is taught from its notes rather than from Géron, and is examined on the same terms as every other. The notes are the primary source; the deck is the lecture.

Today

1. The task: a catalogue of photographs and no labels — find the similar ones, group them
2. **The method** — undercomplete autoencoders in PyTorch, against the best linear compression; what the loss sees; the code as a representation, measured on retrieval and clustering; denoising
3. The variational autoencoder, for context, and where representations go from here
4. **Design exercise** — a fashion marketplace's "see similar" feature, in eight steps, with the pitfalls and how to address them

Open Lecture 11 in Google Colab

The setting: a catalogue nobody labelled

A marketplace holds millions of product photographs uploaded by sellers. The category field is free text, often empty, sometimes wrong. Two things the product team wants from the photographs alone:

- **Similar items** — given one garment, the ten most like it, in milliseconds, across the whole catalogue
- **Groups** — a first structure for a catalogue that has none

✓ Every method so far needed a label per row. None of these questions comes with one. The network of today learns from the inputs by trying to *reproduce* them.

The data, and the one rule about labels

Fashion-MNIST	Count
Images to fit on	12,000
Validation images	5,000
Test images, the catalogue	10,000
Pixels per image	784
Classes	10

The same split as Lectures 7 and 8, so the numbers compare. Twelve thousand images are few enough to train every network here on a CPU.

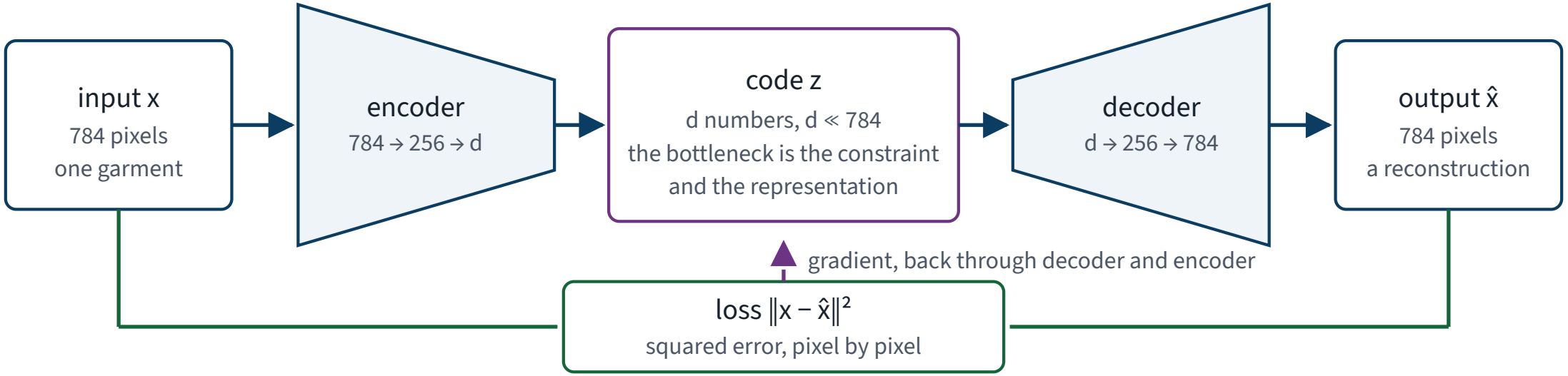
THE LABELS ARE LOADED, AND USED FOR EXACTLY ONE THING

Scoring. Nothing is fitted on them. Precision at ten and the clusters' purity both read the labels after the fact, to say whether a representation learned without them means anything.

An autoencoder: compress, then reconstruct

Encoder, code, decoder

the target is the input; no label appears anywhere



After training, the encoder alone is kept: image in, d numbers out.

The target is the input. The network is trained to return what it was given, through a code too small to hold it — so it has to learn what the images have in common.

The objective

- The **encoder** turns 784 pixels into d numbers — the **code**
- The **decoder** turns the d numbers back into 784 pixels — the **reconstruction**
- The **loss** is the mean squared difference between the input's pixels and the reconstruction's: the loss of least squares, with the input standing in for the label
- Trained by the five-line loop of Lecture 8; nothing in the loop knows there is no label

The whole design question is what stops the network from learning the identity.

Why a bottleneck: a constraint on what the network can reach

A regulariser — dropout, a penalty on the weights, randomly altered inputs — works by *shrinking the set of functions the network can reach*, so that fitting the training data no longer means memorising it. An autoencoder needs the same thing, for a sharper reason: with $d = n$ the identity is reachable, the loss is zero, and nothing was learned.

Constraint	What it forbids	Name
$d \ll n$	**copying: 784 numbers cannot pass through 32**	**undercomplete**
corrupt the input, keep the target clean	copying: what comes in is not what must go out	denoising
penalise the code	using every coordinate for every image	sparse

✓ Today: the first two.

The undercomplete autoencoder, in PyTorch

```
class AutoEncoder(nn.Module):
    def __init__(self, d, hidden=256):
        super().__init__()
        self.enc = nn.Sequential(nn.Linear(784, hidden), nn.ReLU(),
                                 nn.Linear(hidden, d))
        self.dec = nn.Sequential(nn.Linear(d, hidden), nn.ReLU(),
                                 nn.Linear(hidden, 784), nn.Sigmoid())

    def forward(self, x):
        return self.dec(self.enc(x))
```

- $784 \to 256 \to d \to 256 \to 784$; a sigmoid on the output, because a reconstruction is an image in $[0, 1]$
- `enc` and `dec` are attributes on purpose: the code is read by calling the encoder alone
- At $d = 32$: 419,120 parameters — the two 784-wide layers are almost all of them

The loop is the one you already own

```
for epoch in range(EPOCHS):
    net.train()
    for xb in batches(X_fit):           # no labels in the loader
        opt.zero_grad()                 # 1
        out  = net(xb)                  # 2
        loss = F.mse_loss(out, xb)      # 3  the target is the input
        loss.backward()                 # 4
        opt.step()                      # 5
```

One change from Lecture 8: the line marked `# 3` compares the output with the *input*, where Lecture 8 compared it with the labels. Everything else in the loop — the optimiser, the batches, the evaluation — carries over unchanged, because nothing in it ever depended on the target being a label.

Twenty epochs of Adam at 10^{-3}, batches of 128, 12,000 images: a CPU is enough.

Run it at $d = 32$

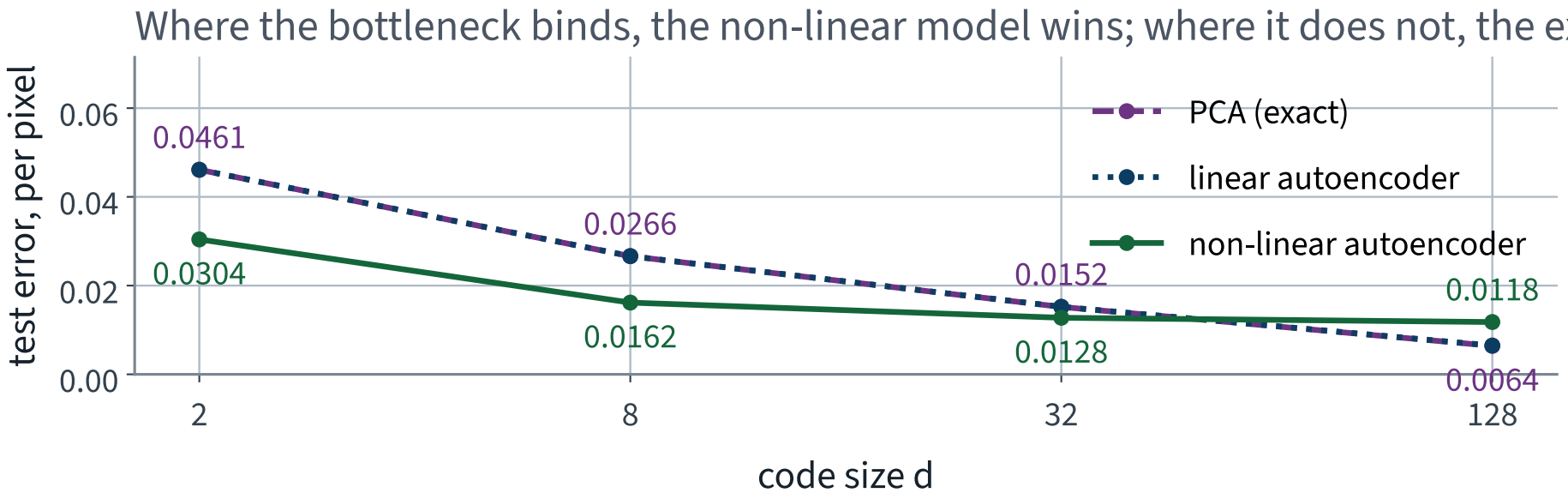
Model, test error per pixel	Error
PCA with 32 components — the best any *linear* compression can do	0.01519
Linear autoencoder: one linear encoder, one linear decoder	0.01523
Non-linear autoencoder, 419,120 parameters	**0.01275 ✓**

About a sixth below the best linear compression.

Without ReLUs a network lands where PCA is, to the third digit: a linear network can only find the best flat compression. The sixth is what a curved map buys over a flat one. It is the entire justification for the two ReLU layers.

Training error 0.01188, validation 0.01276: a small gap, so little overfitting, and small because the bottleneck is doing the regulariser's job.

The code size, swept



The linear network tracks the exact projection at every d. The non-linear one wins where the bottleneck binds — and at $d = 128$ it *loses*: 0.01179 against 0.00641.

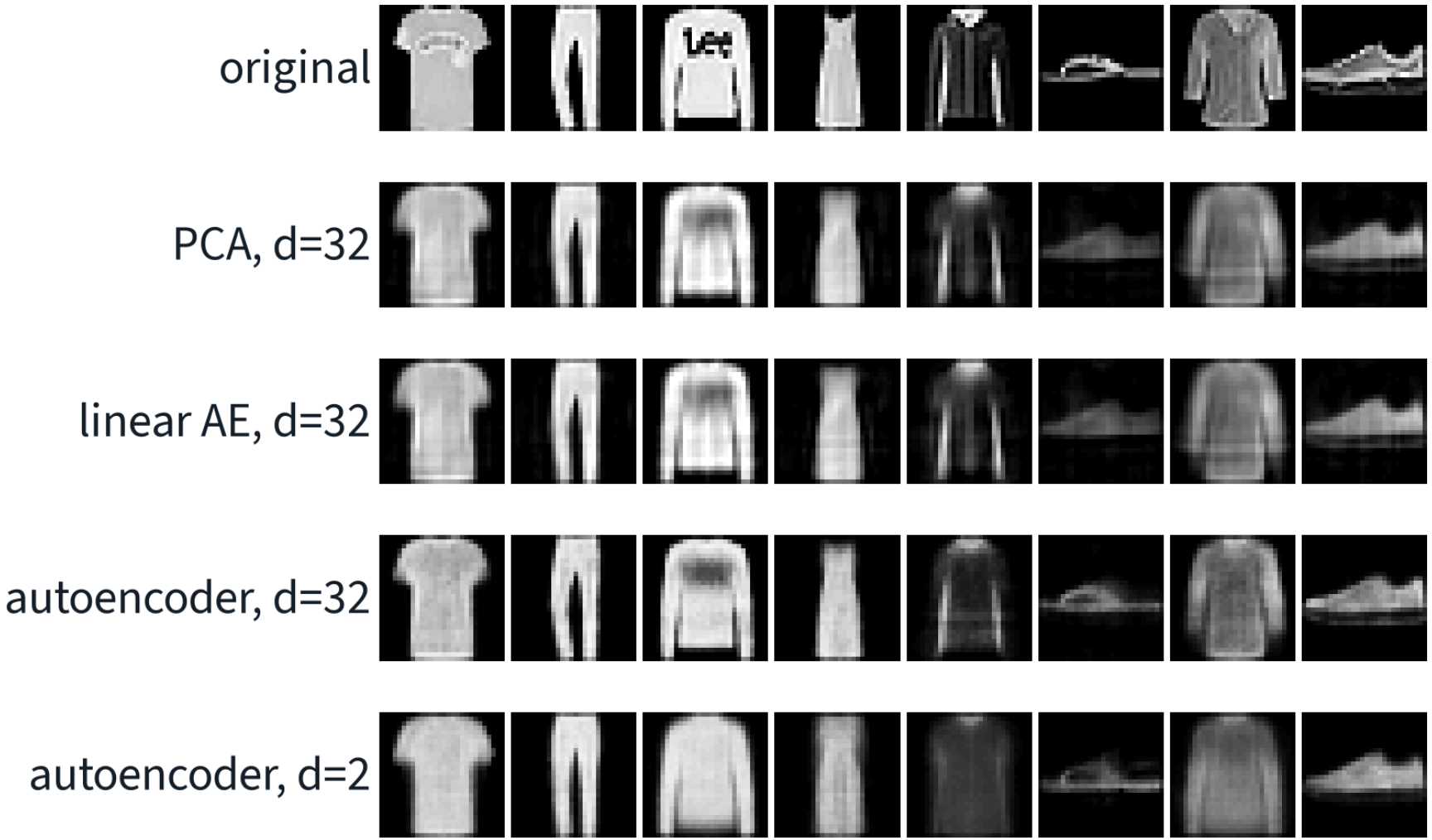
Reading the last column first

d	Projection (exact)	Linear autoencoder	Non-linear autoencoder
2	0.04610	0.04611	**0.03041 ✓**
8	0.02662	0.02665	**0.01617 ✓**
32	0.01519	0.01523	**0.01275 ✓**
128	**0.00641 ✓**	0.00649	**✗ 0.01179**

FAILURE CONDITION — A TRAINED APPROXIMATION AGAINST AN EXACT OPTIMUM

PCA is computed exactly, with no training. The network is a *trained* approximation to the best of a larger class, and at $d = 128$ its budget — one hidden layer of 256, twenty epochs — leaves it above what PCA gets with no training at all. More capacity is not more accuracy until the optimiser has found it.

What the loss sees



Rows two and three, PCA and the linear autoencoder, are the same reconstruction. Row five is what two numbers buy.

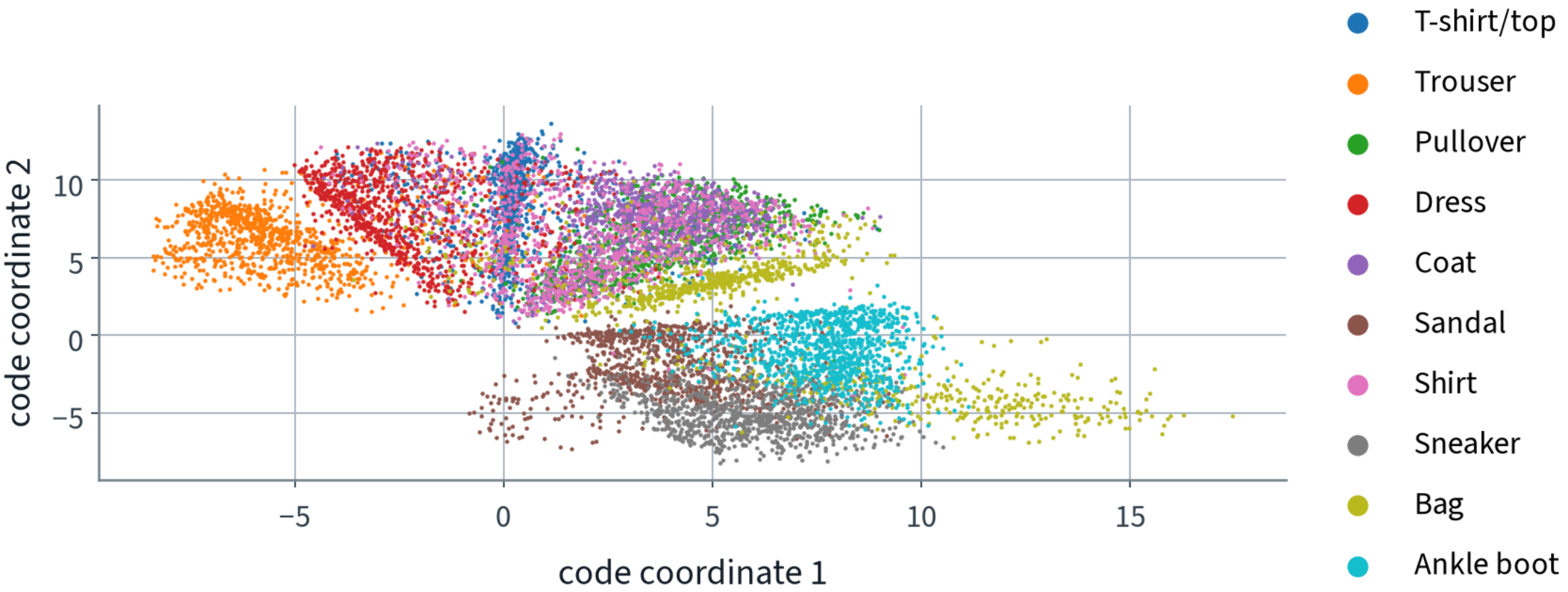
What the loss cannot see

Squared error on pixels is a distance between two arrays. It has no idea which pixels matter.

- **It rewards blur.** The average of a class is close, in pixel distance, to every member of the class. A model short of capacity reconstructs the average and is scored well for it.
- **It is blind to a shift.** Move a sharp edge one pixel and the error is large; smear it over three pixels and the error is smaller. The sharper reconstruction is the worse one, by this loss.
- **It measures pixels, not garments.** A missing button costs nothing; a slightly different grey over the whole sleeve costs a lot.

> **FAILURE CONDITION — A LOSS THAT REWARDS THE AVERAGE**
>
> Reconstruction error falling is not the same as reconstructions improving. Look at them. A code chosen by the loss alone is a code tuned to reproduce the common, and the common is what you already knew.

The code as a picture: $d = 2$



Learned without labels; the colour is the label, added afterwards.

The code as a representation

Thirty-two numbers per garment, computed from the pixels, learned from the pixels. The product questions become geometry in that space:

Question	Operation on the codes	Scored by
similar items	nearest neighbours by Euclidean distance	precision at ten, against the labels
groups	k-means with $k = 10$	mean purity of the clusters, against the labels

✓ Each of them has a baseline that needs no learning: the same operation on the 784 raw pixels, and on the 32 PCA scores. A learned code that cannot beat its own input has learned nothing the question needed.

Similar items: nearest neighbours, and how they are scored

```
def precision_at_10(Z):              # Z: one row per catalogue image
    Zq = Z[q]                          # 1,000 fixed queries
    D2 = (Zq**2).sum(1)[:, None] - 2 * Zq @ Z.T + (Z**2).sum(1)[None, :]
    D2[np.arange(len(q)), q] = np.inf          # not your own neighbour
    nn10 = np.argsort(D2, axis=1)[:, :10]
    return (y_test[nn10] == y_test[q][:, None]).mean()
```

- Catalogue: the 10,000 test images; queries: 1,000 of them, each searching the other 9,999
- $\lVert a - b\rVert^2 = \lVert a\rVert^2 - 2a\cdot b + \lVert b\rVert^2$: one matrix product instead of a loop over queries
- **Precision at ten**: of the ten returned, the fraction of the query's class. The label is read here and nowhere else.

One shirt, its ten nearest neighbours, found two ways



Top: the ten catalogue images closest to the query, comparing all 784 pixels. Bottom: the ten closest, comparing only the 32-number codes. Green: also a shirt; red: another class. Nine of ten in both rows: the code finds what the pixels find, with a 24th of the numbers. The next slide shows the same over 1,000 queries.

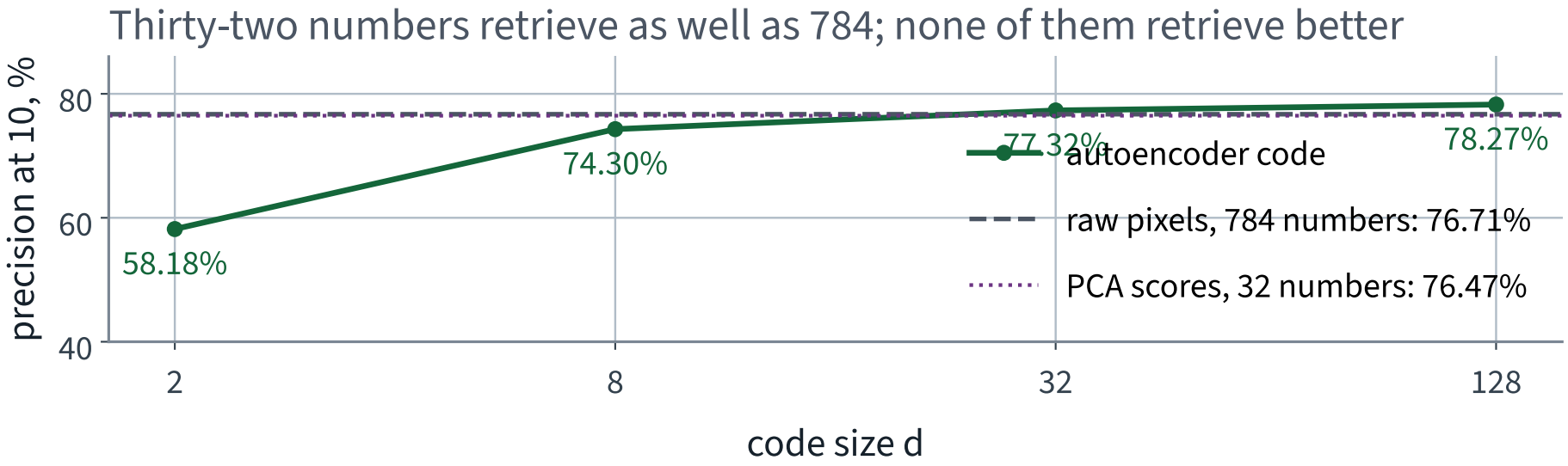
Retrieval, measured

Space	Numbers per item	Precision at 10
Raw pixels	784	76.71%
PCA scores	32	76.47%
Linear autoencoder code	32	77.07%
Autoencoder code	**32**	**77.32% ✓**

Four spaces within a point of one another.

The code did not beat the pixels, and it could not have: nothing in the reconstruction loss is about which garments are *similar*. What it bought is the second column.

How many numbers a garment needs



Two numbers lose a fifth of the precision; eight recover most of it; thirty-two match the pixels; 128 add a point. The representation question is a cost question from eight upwards.

What the code buys: the bill for a catalogue

Per item	Raw pixels	Code	Factor
Numbers stored	784	32	**24.5 ✓**
Multiplications per comparison	784	32	**24.5 ✓**
Bytes, as float32	3,136	128	**24.5 ✓**

A nearest-neighbour search is a comparison against every item in the catalogue. At equal precision, the factor in the last column is the factor on the index's memory and on the query's cost — and for a photograph of real size, 28 × 28 becomes thousands of pixels and the factor grows with it.

The case for a learned representation in a product is, first of all, this table. Quality is the next lecture's problem.

Groups: k-means on the code

Space, $k = 10$	Numbers per item	Mean purity
Raw pixels	784	0.6473
PCA scores	32	0.6471
Autoencoder code	32	0.5891

Purity: for each cluster, the share of its members that belong to its commonest class, averaged over the ten clusters. Pixels and PCA scores tie; the learned code loses six points — nothing in the reconstruction loss is about which garments belong together, and here it shows.

✓ Same lesson as retrieval. The structure k-means finds is the structure the two-dimensional picture drew — footwear, trousers, bags, torso — and it is in the pixels already. The code carries it in 32 numbers, at a price the audit shows.

The denoising autoencoder: the idea

Give the network a *noisy* image, and score it on how close its output is to the *clean* one.

- **When.** The images you will process are noisy: phone photos, scans, low light
- **Why.** A plain autoencoder reproduces the noise along with the garment. One trained noisy-in, clean-out has learned to remove it
- **When not.** Clean images: there it is slightly worse than the plain one (the measured slide shows by how much)
- **Also, not measured here.** It stops the network copying even when the code is wide

It needs only clean images: the noise is added by us, during training.

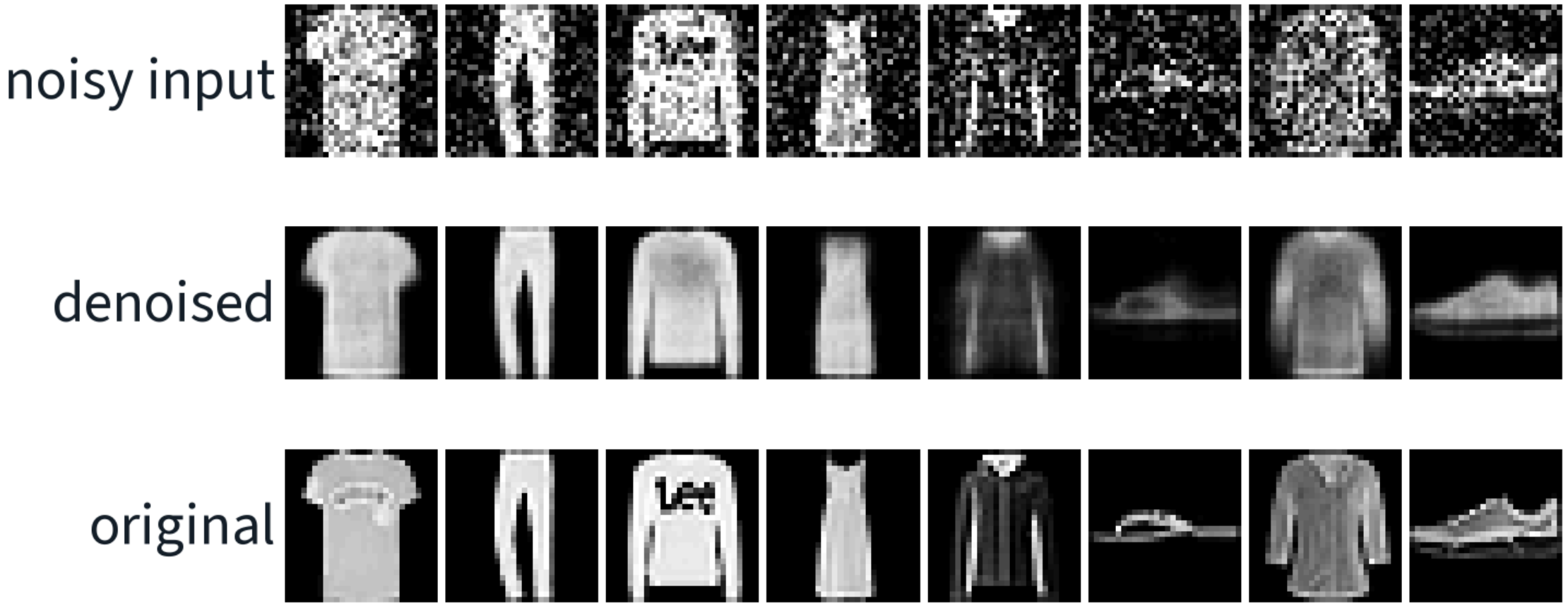
The denoising autoencoder: one line changes

```
for xb in batches(X_fit):
    xin  = (xb + 0.3 * torch.randn_like(xb)).clamp(0, 1)   # noisy input
    opt.zero_grad()
    loss = F.mse_loss(net(xin), xb)                        # clean target
    loss.backward(); opt.step()
```

- **Line 2** adds random noise to every pixel (pixels run from 0 to 1; the noise has standard deviation 0.3), then clips the result back to between 0 and 1
- **Line 4** feeds the noisy image in, and compares the output with the clean image `xb`
- The noise is drawn anew every time an image is used. If each image always carried the same noise, the network could learn that noise by heart

Same network, same loop: the plain autoencoder is this loop with line 2 removed.

Noisy in, garment out



Top: the input. Middle: the output. Bottom: the clean original, never shown to the network. The garment comes back, smoothed; the noise does not.

Denoising, measured: what it buys, and what it costs

Given a noisy test image	Error vs clean
do nothing: the noisy image itself	✗ 0.05264
plain autoencoder	0.02408
denoising autoencoder	**0.01551 ✓**

The plain autoencoder already removes about half the noise: noise does not fit through 32 numbers. Training on noisy inputs removes most of the rest.

Given a clean test image	Plain	Denoising
error	0.01275	0.01781
precision@10	77.32%	75.87%

The price: on clean images the denoiser is a little worse, at reconstructing and at finding similar items.

✓ Which one? Noisy images in use → the denoising autoencoder. Clean images → the plain one.

What can go wrong, in one place

When	What happens	Measured
The code is as wide as the image	the network can copy its input: zero loss, nothing learned	at $d = 784$ the loss can reach 0
The code is wide, but not that wide	the trained network does worse than PCA	$d = 128$: 0.01179 against 0.00641

And for every autoencoder: squared error rewards a blurry average, so a lower loss does not mean better-looking images. Look at the reconstructions.

The variational autoencoder

BEYOND THE SYLLABUS, FOR CONTEXT

The autoencoders so far give one code per image and nothing in between: decode a point between two codes and there is no reason the output is a garment. The **variational autoencoder** makes the encoder output a *distribution* — a mean and a spread for each number of the code — samples a code from it, and adds a second term to the loss that keeps every such distribution close to a standard normal.

That term is a regulariser on the code — the third row of the bottleneck table — and it fills the code space: a code drawn from a standard normal decodes to something plausible, which makes the decoder a *generator*.

The reparametrisation trick

A sampled code breaks the chain rule of Lecture 8: there is no derivative of a random draw with respect to the parameters that chose its distribution. The repair moves the randomness outside the graph: BEYOND THE SYLLABUS, FOR CONTEXT

- Draw standard normal noise first, and treat it as an input
- The code is then the mean plus the spread times the noise: an ordinary computation on the encoder's outputs
- Reverse mode differentiates through the mean and the spread as through any layer; the draw is never differentiated

Not examined. It is here because every generative model you will meet after this course contains this move.

Where this leads: representations, and what trains them

Representation	Trained by	What distances in it mean	Lecture
autoencoder code	**reconstructing the input**	**pixel distance, through a curved map, in a basis nobody chose**	**11**
a classifier's penultimate layer	labels	whatever separates the classes	13
a dense-retrieval embedding	query–document pairs	relevance	20
a contrastive embedding	pairs that belong together	similarity, as the pairs defined it	23

✓ Today's code matched the pixels on retrieval and could not beat them, because the loss never saw a pair. The rest of the course is, in part, the story of what happens when it does.

The notebook for this lecture

Open Lecture 11 in Colab

- **Runs on free CPU**, and on the CPU by choice: 12,000 images and small networks, and the numbers on these slides are its run
- PCA and the linear autoencoder, the baseline any non-linear network has to beat
- The non-linear autoencoder, the sweep over d, the reconstructions, the two-dimensional code drawn
- Precision at ten and k-means in four spaces; and the denoiser

> **WHAT TO CHANGE**
>
> Set `NOISE = 0` in the denoising section and run again. The denoising rows become the plain autoencoder's, to the digit — they were always the same model with one line changed.

Design exercise · in the manner of an ML system-design interview

The task: “see similar” on a fashion marketplace

> **THE BRIEF**
>
> An online fashion marketplace sells garments listed by thousands of independent brands and shops. The product is its shopping app; the feature is a **“see similar”** strip under every item page, and a camera search that finds catalogue items like a photo the shopper took. The users are shoppers; the customers are the sellers, who pay a commission on every sale. The business measures the strip’s click-through, the conversion it adds, and the return rate of what it sells.

- **Given:** millions of seller photographs with an unreliable category field, the shoppers’ click and purchase logs, the sellers’ stock and price feeds
- **Consumers of the output:** the strip, and the camera search
- **Regulators:** data protection, for the shoppers’ photographs; consumer protection and brand owners, for what “similar” may show

✓ Eight steps follow, each with its pitfall; the model at the centre is today’s code, learned from the photographs alone.

Step 1 — Frame it honestly: a ranking, scored as one

- The output is a ranked list of catalogue items for one query image; the metric is precision at the strip's length on held-out items, read beside its anchor: the same strip filled from the query's category at random
- The label for scoring comes from what shoppers did — items viewed or bought in the same session — cleaned of the strip's own clicks; the seller's category field is a weak second source
- The baseline before any model: the same category, in stock, most popular first

> **PITFALL**
>
> Reporting the reconstruction error. It is the training signal, and the shopper never sees it: a code can reconstruct every photograph beautifully and return the wrong garments. Score the ranking the product shows, and report it against the anchor, or the first number will say nothing about the feature.

Step 2 — The photographs are not as alike as the dataset's

Pitfall	What happens	What to do
Studio, not garment	sellers shoot on different backgrounds, at different sizes, on or off a model; a code learned from the pixels learns the studio, and the strip returns the same seller's other items	crop to the garment, resize, normalise the background before anything is fitted; check the strip for seller clumping
Shoppers' photographs	a camera-search image is personal data, and is nothing like a studio shot	encode on the device or discard after encoding; never keep the photo; train the encoder with camera-style augmentations
Duplicates	the same garment listed by four sellers, and in six colours	link variants to one item before splitting, or the split leaks

✓ Write down what one "item" is before the first fit. Every later number is a number about *that* unit.

Step 3 — Split by item, and by seller

- Hold out whole items — every photograph and variant of them — and hold out whole sellers for a second check: the feature has to work for a shop that joined this morning
- Choose the code size, the distance and the ranking rules on the validation items; read the test items once

> **PITFALL**
>
> A random split over photographs puts the red and the blue of one dress on both sides. The nearest neighbour of a held-out photo is then its own item in another colour, precision looks superb, and the number says nothing about a garment the catalogue has never seen. Split on the item identifier from step 2, and keep a seller-level split beside it.

Step 4 — Learn the code from the catalogue, and size it by the ranking

- Train the autoencoder on the whole catalogue of standardised photographs — no labels needed, so every listing counts, and a convolutional encoder once the photographs are real-sized
- Read the code from the encoder alone; sweep its size and pick the smallest that holds the validation precision — the cost table says why smallest
- Denoising and augmentation as the regulariser: the same garment, shifted, in another colour, on another background, must land on the same code

> **PITFALL**
>
> Choosing the code size by reconstruction error. It falls at every increase, forever; the ranking metric flattens early and the index cost does not. A code sized by the loss is a code sized by the wrong question, and the bill for it is paid on every query.

Step 5 — The index: codes stored once, searched in milliseconds

- Encode every catalogue item once, when it is listed, and store the code beside it; the query is encoded at request time and compared against the stored codes, never against the photographs
- Exact search is a matrix product at a hundred thousand items; at millions, an approximate-neighbour index trades a little recall for a hundred-fold speed, and that recall is measured, not assumed
- Normalise the codes to unit length and compare by cosine, so that a bright photograph is not far from a dark one of the same dress

> **PITFALL**
>
> A distance nobody chose. The code’s coordinates are in a basis the optimiser landed on, and one coordinate with a large range dominates the Euclidean distance. Normalise, or whiten against the validation set, and check that the strip’s neighbours do not all share one bright background.

Step 6 — Similarity is one feature of the ranking, not the ranking

- The strip ranks the nearest hundred by a second model that sees similarity, stock, size availability, price band, the seller’s return rate and the shopper’s history — a table, and a boosted ensemble on it
- Business rules sit on top: in stock only, the shopper’s size, nothing from a seller under review

> **PITFALL — THE LOOK-ALIKE UNDER A BRAND’S PAGE**
>
> Visual similarity is blind to who made the garment. Under a luxury brand’s listing the strip shows the imitations that look most like it, the brand — a paying customer — sees its page selling look-alikes, and the counterfeit complaints land with the regulator. Brand and seller constraints belong in the ranking layer, and the similarity model must never be the last word.

Step 7 — Release, monitor, retrain

- Release behind an A/B test on live traffic, read on the strip's click-through, the conversion it adds and the return rate of what it sells, with the sellers' exposure held fair across both arms
- Monitor the distribution of codes as the catalogue turns over with the seasons; retrain on a schedule and re-encode the whole catalogue in one atomic swap of the index
- Keep a randomised slot in the strip, so that the clicks the next model trains on are not only the ones this model chose

> **PITFALL**
>
> Two encoders in one index. A retrained encoder puts the same garment at a different code, so an index holding codes from both versions compares numbers that no longer mean the same thing, and the strip fills with garments that are near nothing. Version the encoder with the index, and never serve a query from one against codes from the other.

Step 8 — Ship it: pipeline and pitfalls

Standardised photographs, variants linked into items → split by item and seller → autoencoder, code sized on validation precision → normalised codes, versioned index → ranking with business rules → A/B test → retrain, re-encode, monitor.

Pitfall	Address it by
Reconstruction error as the product metric	precision on the strip against the category anchor
A code that learns the studio	standardised photographs; seller clumping checked
Variants across the split	splitting by item and by seller
Code sized by the loss	the smallest size that holds validation precision
A distance nobody chose	normalised codes, cosine, whitening
Look-alikes under a brand	brand and seller rules in the ranking layer
Two encoders in one index	versioned encoder and index, atomic re-encoding

What we did

1. Trained a network with no labels: compress to a code, reconstruct, and make the bottleneck the constraint — a regulariser, because without it the identity is reachable
2. Measured the baseline: a linear autoencoder lands where PCA is, the best flat compression
3. Built the non-linear one: a sixth below PCA at $d = 32$, and above it at $d = 128$, because a trained approximation is not an exact optimum
4. Read the code as a representation: 32 numbers retrieve and cluster as well as 784 pixels, and no better, at a twenty-fourfold saving — the loss never saw a pair
5. Denoising: noisy images in, clean images out. It removes the noise a plain autoencoder reproduces, and costs a little on clean images
6. Designed a marketplace's "see similar" feature in eight steps — a ranking metric against its anchor, standardised photographs, a split by item, a code sized by precision, a normalised versioned index, business rules above similarity, an A/B test — each named for the pitfall it avoids

NOT PRESENTED — FOR AFTERWARDS

Exercises Lecture 11

five, in the style of the written paper

Exercises — Lecture 11

1 At a code of 32 numbers, the test error per pixel is 0.01519 for PCA with 32 components, 0.01523 for a linear autoencoder and 0.01275 for the same autoencoder with ReLUs. Explain why the first two agree, and what the third bought. [4]

2 At a code of 128 numbers the non-linear autoencoder's test error is 0.01179, above PCA's 0.00641. A colleague concludes that non-linear autoencoders are worse than PCA at wide codes. State what is wrong with the conclusion. [4]

3 The code size is set equal to the number of pixels and nothing else constrains the network. State what the trained network computes, and what its code is worth as a representation. [3]

Solutions on Lecture 12’s deck.

Exercises — Lecture 11 (continued)

4 On noisy test images, the error against the clean image is 0.05264 for the noisy image itself, 0.02408 for the plain autoencoder and 0.01551 for the denoising one. On clean test images it is 0.01275 for the plain autoencoder and 0.01781 for the denoising one. A team will serve phone photographs. Say which autoencoder it should use, and why each result comes out the way it does. [5]

5 Nearest-neighbour retrieval in a 32-number autoencoder code scores the same precision at ten as in the 784 raw pixels. State what the code bought, and why it could not have bought more. [4]

Solutions on Lecture 12’s deck.

THE FIVE SET AT THE END OF LECTURE 10

Solutions Lecture 10

not part of this lecture

Solutions — Lecture 10

1. A random forest ranks a census sampling weight first by impurity importance, yet shuffling that column on…

✓ It is computed on the training rows, where a fully grown forest fits even noise; and a column with a distinct value per row offers a split almost anywhere, so some split on it always cleans a small node.

- It measures how much the trees used the column to fit, not how much predictions on new rows depend on it
- On the census data the weight is first by impurity and eleventh of twelve by permutation on held-out rows

2. The partial dependence of a model's probability on age rises from 0.015 at 20 to 0.270 at 50. Say what the…

✓ It averages the ICE curves, one per applicant with age varied and the rest kept. It hides interactions — the married rise from 0.030 to 0.490, everyone else from 0.003 to 0.092 — so the average describes nobody.

- Draw the ICE curves, coloured by the suspected interaction, beside the average
- The average sits between the two groups, at a slope no applicant has

Solutions — Lecture 10 (2)

3. Define the Shapley value ϕ_i of feature i for one prediction, given a value function v over…

✓ $\phi_i = \sum_{S\subseteq N\setminus\{i\}} \frac{|S|!\,(M-|S|-1)!}{M!}\bigl(v(S\cup\{i\})-v(S)\bigr)$, the marginal contribution of i averaged over all $M!$ orders of the features. In one order the contributions telescope to $v(N)-v(\varnothing)$; averaging a constant over the orders leaves it unchanged.

4. A classifier is applied to two groups whose base rates differ, and at its threshold the two groups have the…

✓ Counting the true positives two ways gives $\text{TPR}\cdot p = \text{selected}\cdot\text{PPV}$ in each group. With equal selection rates and equal PPV the right-hand sides are equal, so $\text{TPR}_1 p_1 = \text{TPR}_2 p_2$, and equal TPRs would force $p_1 = p_2$.

- The true positives are $\text{TPR}\cdot p\,n$ as a share of those with $Y=1$, and $\text{PPV}\cdot\text{selected}\cdot n$ as a share of those selected
- Any two of the three criteria fix the third by the ratio of the base rates

Solutions — Lecture 10 (3)

5. A bank removes the applicant's sex from its model's inputs and reports that the model is therefore fair.…

✓ The other answers predict sex — on the census data the twelve remaining answers predict it at an AUC of 0.9327 — so the model can use it through proxies; and without the attribute nobody can measure whether outcomes differ by sex. Keep the attribute under separate access for the audit, and measure the criteria per group.

- Removing a column removes neither its correlates nor what the model learned through them: the household relationship alone names the sex of every married applicant
- Fairness is a property of outcomes per group, which can only be measured with the group recorded

LECTURE 12 · GÉRON CH. 12

Convolutional networks

Why a dense layer is the wrong shape for an image
Weight sharing and equivariance, derived
A network that sees Flowers102 at its real size

Run the Lecture 11 notebook first