Neural networks

Fashion-MNIST

LECTURE 7 | GÉRON CH. 9

Applications of Machine Learning

BSc Mathematics of Artificial Intelligence · Third year

Part II begins

Part	Lectures	Chapters	Hardware
I · Tabular data and classical models	1–8	1–8	CPU
II · Neural networks	**9–11**	**9–11**	**CPU**
III · Computer vision	12–14	12	CPU
IV · Sequences and language	15–18	13–15	CPU
V · Search and recommendation	19–22	—	CPU
VI · Multimodal models	23	15–16	CPU

Part I ended without labels: four hundred faces compressed by PCA — the SVD of the centred data, whose error is the tail of the spectrum — then clustered by k-means with k chosen by silhouette, and forty labels spent on cluster representatives. Today the labels are back, and the model is new.

Every notebook in this course runs on Colab's free CPU tier. The corpora are subsampled and the networks are small enough to train there — deliberately, so that every result on a slide is one you can reproduce rather than one you must take on trust.

Today

1. The task, the corpus, and why a pixel is a feature
2. **The mathematics** — the multilayer perceptron, and why it needs a non-linearity between its layers
3. **The network** — a layer in matrices, the parameter count, softmax and cross-entropy, and the architecture table
4. **The task** — a metric and its baseline, one fit, ten more by hand, and where the errors are
5. What `MLPClassifier` cannot do, and why the next lecture exists
6. **Design exercise** — a radiology-software start-up's chest X-ray triage, in nine steps, with the pitfalls and how to address them

Open Lecture 7 in Colab

The stakeholder

A document-processing bureau scans incoming items and routes each one to one of **ten** queues.

- Items arrive as small greyscale scans, already cropped and centred
- A human currently sorts them; the operator wants to know how often a machine would be right
- Misrouting is recoverable but expensive: the item goes to the wrong desk and comes back a day later

Ten classes, one label each, no ranking, no probability requirement stated. Write that down: it decides the output layer.

The corpus we stand in with

The bureau's archive is confidential. **Fashion MNIST** is the dataset Chapter 9 uses, and it has the same shape of problem: small greyscale images, ten classes, one label each.

Property	Value
Training images	60,000
Test images (shipped as a test set)	10,000
Image size	28 × 28
Features per image, flattened	784
Pixel values	0–255, integer
Classes	10

The test split is the one the dataset ships with, so your number is comparable with every published result on it.

Ten classes, four examples each

Four examples of each of the ten classes, 28 × 28, greyscale



Pullover, coat and shirt are one silhouette at this resolution — and that is where the errors will be.

The ten classes

1. T-shirt/top
2. Trouser
3. Pullover
4. Dress
5. Coat
6. Sandal
7. Shirt
8. Sneaker
9. Bag
10. Ankle boot

Three of these are upper-body garments seen from the front. Ask the room which three, before the model tells you.

A pixel is a feature

We flatten each 28 × 28 image into a vector of 784 numbers and hand it to the model as an ordinary table of features.

> **WHAT THAT THROWS AWAY**
>
> The model is told nothing about which pixels are neighbours. Shuffle all 784 columns with one fixed permutation and it learns exactly as well. A human could not read a single shuffled image.

That is a real loss, and repairing it is Chapter 12 — four lectures away. Today we pay it, and it is the honest starting point.

> **FAILURE CONDITION — A MODEL TOLD NOTHING ABOUT WHICH PIXELS ARE NEIGHBOURS**
>
> Flattening an image to 784 independent features discards adjacency entirely: shuffle every column with one fixed permutation and the network learns exactly as well. That is a statement about what the architecture cannot use, and no training curve reveals it.

Scale the pixels

Divide by 255, so every feature lies in $[0, 1]$.

- Networks are trained by gradient descent, and the step size is one number shared by every weight
- Inputs two orders of magnitude apart make one step too large for some weights and too small for others

AND NO, THIS IS NOT A LEAK

Dividing by a constant that is a property of the file format fits nothing. `StandardScaler` estimates a mean and a variance *from data* and must therefore be fitted after the split. `x / 255` estimates nothing.

The split

Set	Images	Used for
Training	55,000	fitting the weights
Validation	5,000	choosing hyperparameters, by hand, today
Test	10,000	once, at the very end

The validation set is carved out of the training half with a fixed seed. It exists because we are about to try ten configurations, and a number chosen on the set it is reported on is optimistic. The model is fitted on the first 12,000 of the 55,000, so the whole lecture runs on a free CPU; every number from here on is from that run.

Where the idea came from

McCulloch and Pitts, 1943: a very simplified model of a biological neuron, which fires when enough of its inputs fire.

> *The analogy is historical and it is where the vocabulary comes from — neuron, activation, firing. It is not an argument for anything. Modern networks resemble brains about as closely as aeroplanes resemble birds.*

Chapter 9 opens with this and then drops it. So do we.

The multilayer perceptron

One unit computes a weighted sum of its inputs plus a bias, then an *activation*: $h = \phi(\mathbf{w}^\mathsf{T}\mathbf{x} + b)$. Units are arranged in layers:

- An **input layer**, which holds the features and has no weights
- One or more **hidden layers**, each fully connected to the previous one
- An **output layer**, one unit per class

Fully connected, or *dense*: every unit of one layer receives every unit of the previous one. That is where the parameters are.

Two or more hidden layers and the book calls it a **deep** neural network. The word means nothing more than that.

Why there must be a nonlinearity

Suppose the activation were the identity. Then

$$\mathbf{W}_2\left(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1\right)+\mathbf{b}_2 \;=\; \left(\mathbf{W}_2\mathbf{W}_1\right)\mathbf{x}+\left(\mathbf{W}_2\mathbf{b}_1+\mathbf{b}_2\right)$$

✗ A single affine map. Two hundred stacked layers would still be a single affine map.

Depth buys you nothing without a nonlinearity between the layers. This is one line of algebra and it is the reason activation functions exist at all.

Why ReLU won

Activation	Range	Cost	Problem
Sigmoid	$(0, 1)$	an exponential	saturates at both ends
tanh	$(-1, 1)$	an exponential	saturates at both ends
✓ ReLU	**✓** $[0, \infty)$	**✓ a comparison**	**✓ flat below zero**

`max(0, z)` is fast, does not saturate above, and in practice trains better. It is the default in Chapter 10 and it is our default today.

“Flat below zero” is a real defect and it has a name — dying ReLUs. It is diagnosed in Lecture 9, not here.

One layer, in matrices

A WHOLE MINI-BATCH AT ONCE

$$\mathbf{H} = \phi(\mathbf{X}\mathbf{W} + \mathbf{b})$$

- $\mathbf{X}$ — one row per instance in the batch, one column per input
- $\mathbf{W}$ — one row per input, one column per unit
- $\mathbf{b}$ — one entry per unit, broadcast down the rows
- ϕ — the activation, applied element by element

The whole forward pass is a handful of matrix products. That is why a GPU helps at all: a GPU is a machine for doing exactly this.

Count the parameters before you fit

```
784 x 300 + 300  =  235,500      layer 1
300 x 100 + 100  =   30,100      layer 2
100 x  10 +  10  =    1,010      output
                   --------
                    266,610      total
```

266,610 parameters, fitted from 12,000 images.

> **MORE THAN TWENTY PARAMETERS PER TRAINING IMAGE**
>
> Lecture 3's failure condition for least squares was $m < n$ — more parameters than instances. We are past it, comfortably, and the model will still work. Understanding why is Lectures 13 and 14.

The output layer

Ten classes, mutually exclusive. Ten output units, and a **softmax** across them:

$$\hat{p}_k = \frac{\exp(z_k)}{\sum_{j=1}^{10} \exp(z_j)}$$

- Every output is positive and they sum to 1
- The largest logit z_k gives the largest probability, so `argmax` is unchanged by the softmax

The full properties of softmax — shift invariance, the gradient, why the loss consumes logits rather than probabilities — are derived in Lecture 17.

The loss

CROSS-ENTROPY, ONE INSTANCE

$$L = -\sum_{k=1}^{10} y_k \log \hat{p}_k = -\log \hat{p}_{\text{true}}$$

The target is one-hot, so nine of the ten terms are zero. The loss is minus the log probability the model assigned to the right answer, and it goes to infinity as that probability goes to zero.

Chapter 4 called this log loss when there were two classes. Same object.

How the weights get updated

Backpropagation: one forward pass to compute the loss, one backward pass to compute every partial derivative, one gradient-descent step.

- Rumelhart, Hinton and Williams, 1986 — and the reason the field restarted
- The backward pass costs about as much as the forward one, no matter how many parameters there are

> **WHY "NO MATTER HOW MANY" IS REMARKABLE**
>
> 266,610 parameters, one number of interest, and a single sweep gets all of the derivatives. That claim is derived at the top of the next lecture, and it is the whole reason any of this is feasible.

Architecture — classification networks

Task	Output units	Output activation	Loss
Binary	1	sigmoid	binary cross-entropy
Multilabel binary	one per label	sigmoid	binary cross-entropy
✓ Multiclass	**✓ one per class**	**✓ softmax**	**✓ cross-entropy**

Our task is the third row: ten mutually exclusive classes, so ten units, softmax, cross-entropy.

The distinction between rows two and three is not cosmetic. Multilabel asks ten independent yes/no questions; multiclass asks one ten-way question. Softmax couples the outputs; ten sigmoids do not.

An output activation paired with the wrong loss

FAILURE CONDITION

Softmax with mean squared error trains — slowly and badly. The gradient through a saturated softmax nearly vanishes, so the run completes, the loss falls a little, and the pairing that would have worked is never suspected.

How wide, and how deep?

- **Depth** buys parameter efficiency: two narrow layers express things one wide layer needs exponentially many units for
- **Width** buys capacity within a layer, at $d \times n + n$ parameters a time for d inputs and n units
- In practice: pick a depth that is enough, then grow the width until validation stops improving — and stop there

> **THE HONEST ANSWER**
>
> There is no formula. What there is, is a budget you can compute before fitting anything — the parameter count — and a validation curve that tells you when you have spent it badly.

The numbers depend on the corpus. The method for finding them does not.

NOT EXAMINABLE — ENGINEERING

Which metric?

The operator asked: *how often is it right?*

That is accuracy. Lecture 3 spent a whole lecture on why accuracy is a trap — so before using it, check the condition under which it failed.

Lecture 3	Today
One class held 10% of the data or worse	Every class holds exactly 10%
“Always negative” scored above 90%	“Always *k*” scores 10.0%
Accuracy hid a useless model	Accuracy cannot hide one

The rule you keep is not “never use accuracy”. It is **count the classes first**.

The trivial baseline, computed

```
majority = np.bincount(y_fit).argmax()
baseline = np.full(len(y_test), majority)
print(accuracy_score(y_test, baseline))       # 0.1
```

Ten classes, 1,000 test images each, one guess for every image.

Not an estimate and not a convention — a measurement, and it happens to be exact.

The whole model, in Scikit-Learn

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(hidden_layer_sizes=(300, 100),
                    activation="relu",
                    solver="adam",
                    learning_rate_init=1e-3,
                    batch_size=128,
                    max_iter=20,
                    random_state=42)
clf.fit(X_fit, y_fit)
```

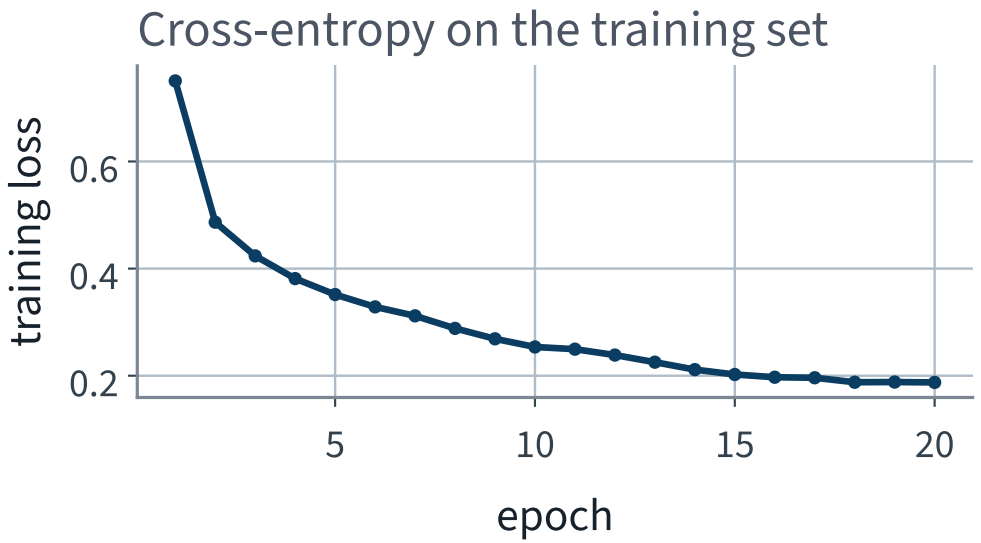
Nine lines, and every architectural decision so far is one of the arguments. The softmax and the cross-entropy are not arguments — `MLPClassifier` chooses them for you, from the number of classes it sees in `y_fit`.

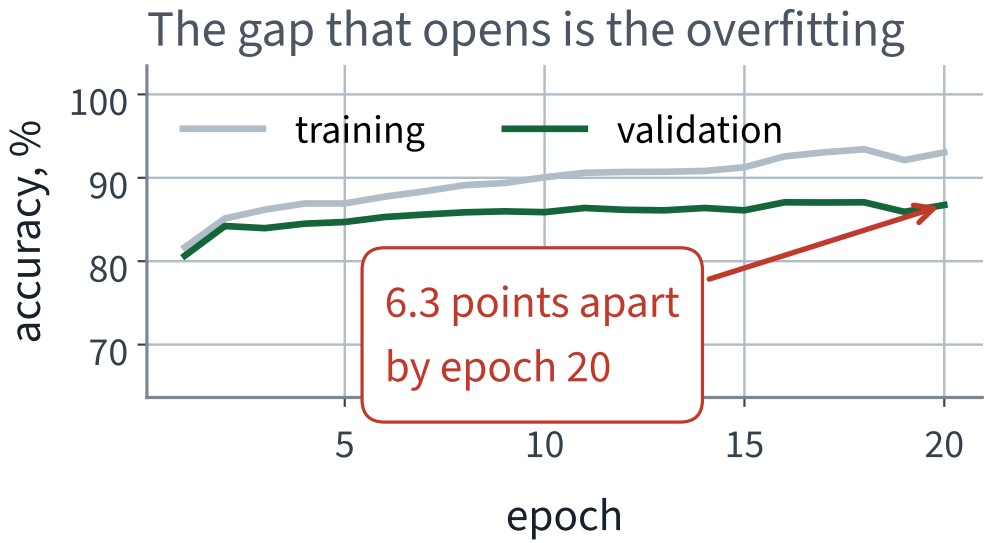
Run it

Quantity	Measured
Training images	12,000
Epochs	20
Validation accuracy	86.74%
✓ Test accuracy	**85.63% ✓**
Baseline	10.0%

It works. Hold on to this model — the next lecture writes the same one again, in PyTorch.

What twenty epochs look like





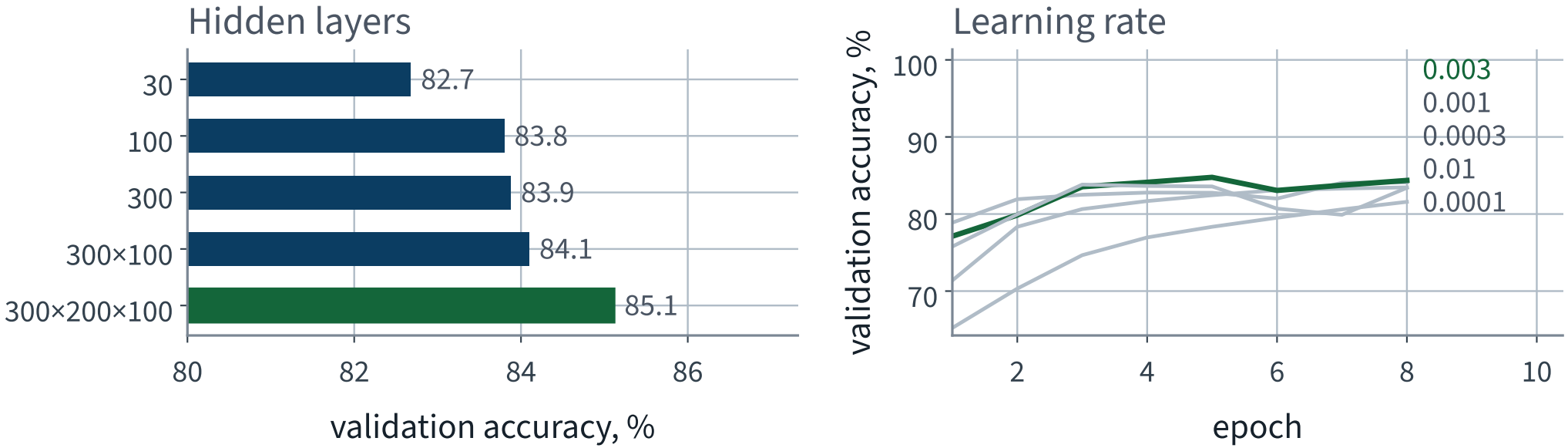
✗ **The two accuracy curves separate and stay separated. That gap is overfitting: the training-versus-validation gap, drawn as lines.**

Reading the curve

- The loss falls smoothly — the optimiser is working
- Training accuracy keeps climbing — the model has capacity to spare
- Validation accuracy flattens — the extra capacity is being spent on memorising

> **THE QUESTION A CURVE ANSWERS THAT A NUMBER CANNOT**
>
> *Would more epochs help?* Hardly: over the last ten epochs validation moved less than a point while training gained three. Without the curve you would have guessed.

Ten fits, by hand



✗ **Ten fits on a 6,000-image subset for 8 epochs, so they fit in the lecture: five architectures, then five learning rates. The architectures span 2.4 accuracy points and the learning rates 2.8. The deepest network wins, and the default learning rate does not.**

Architecture, then learning rate

Hidden layers	Parameters	Validation
30	23,860	✗ 82.68%
100	79,510	83.80%
300	238,510	83.88%
300, 100	266,610	84.10%
✓ 300, 200, 100	316,810	85.12% ✓

Learning rate	Validation
0.0001	✗ 81.58%
0.0003	83.46%
0.001 — default	84.10%
✓ 0.003	84.36% ✓
0.01	83.40%

On this subset **depth pays, a little**: a second hidden layer adds about a fifth of a point, and a third about a point more, for a fifth more parameters.

The learning rates are swept on the (300, 100) network, with the same data and seed: only the size of the step changes, and **the library's default is not the best value on this grid.**

The lesson from ten fits

1. **Depth paid, a little.** On 6,000 images the deepest network beat the single layer of 300 by about a point and a quarter.
2. **The learning rate mattered slightly more — on this grid.** The five architectures span 2.4 accuracy points; the five learning rates span 2.8. One badly chosen scalar lost more than any structural choice gained. Which is a claim about a grid and a scale, never about the world.
3. **Ten fits is not a search.** Ten points from a space with five axes, varied one axis at a time with everything else held fixed.

Point 3 is the one to remember. You have not optimised anything; you have looked at ten places.

Does the ranking survive more data?

Hidden layers	6,000 images, 8 epochs	12,000 images, 20 epochs
30	82.68%	85.38%
100	83.80%	86.26%
300	83.88%	86.58%
300, 100	84.10%	86.74%
✓ 300, 200, 100	**85.12% ✓**	**87.14% ✓**

✓ Here it does: the same order at both scales. Each architecture gains two to three points from the larger run — as much as the whole spread between the five on either.

THE RULE

A ranking measured on a fraction of your data is a measurement of that fraction. This one survived; elsewhere it need not. The only way to know is to re-run the comparison on all the data: the right-hand column.

An accuracy is one number over ten classes

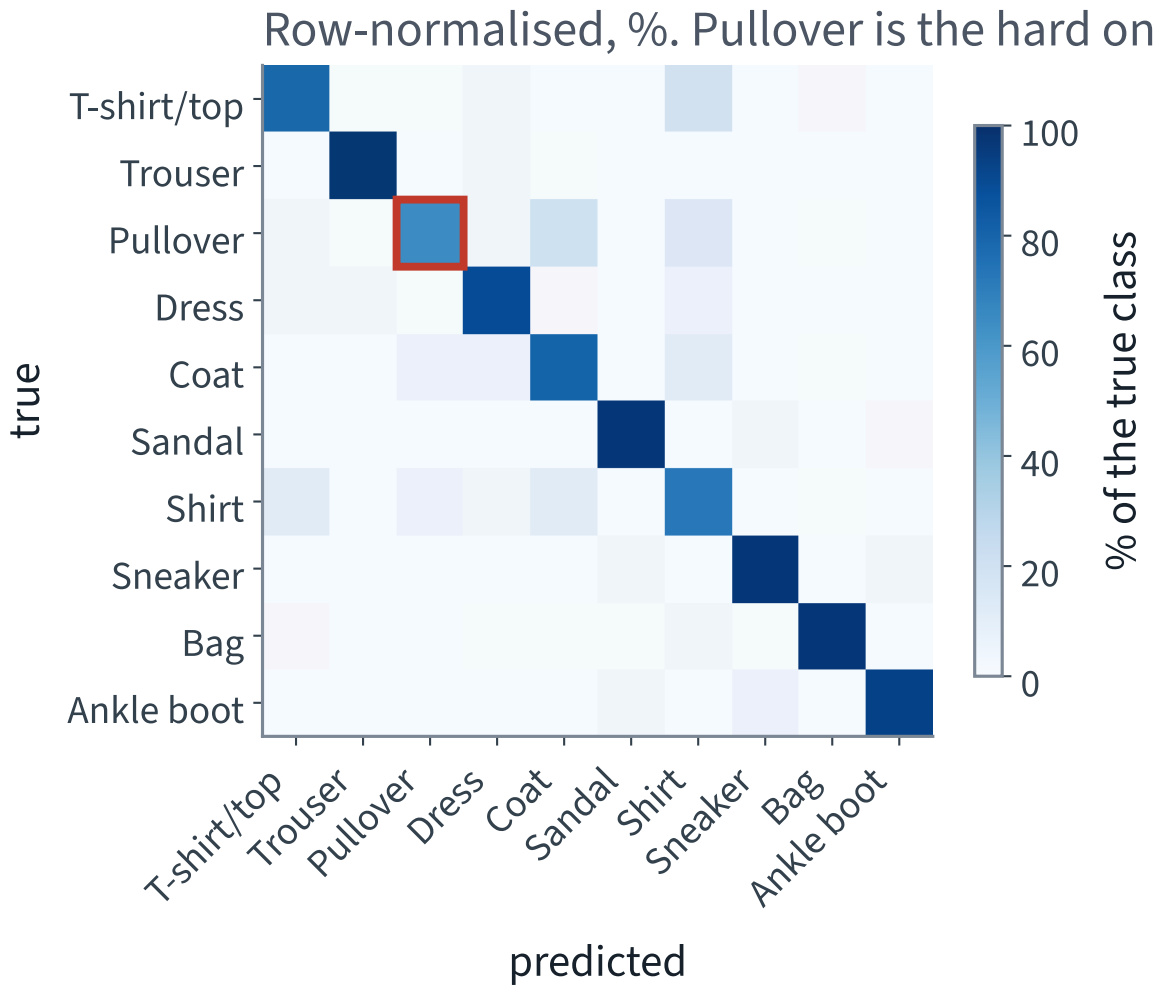
Split it. Lecture 3 built the confusion matrix for two classes; the object generalises without change.

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, clf.predict(X_test))
recall = cm.diagonal() / cm.sum(axis=1)      # one per class
```

Ten rows, ten columns. Row i, column j counts the images of class i the model called class j.

Where the errors are



✗ Four upper-body classes are far worse than the other six, and their errors go to a few specific neighbours.

The hard class

Recall on **Pullover**: **✗ 63.6%**, against 97.2% on Trouser.

Pullovers are called…	Share of all pullovers
Coat	18.8%
Shirt	12.7%
Dress	2.0%

Coat and Shirt, two of the three the room named from the image grid, take 31.5 of the 36.4 points it loses — and Shirt is the next-worst class itself, at 71.1%. The model is not confused about anything a person would find easy.

What to tell the operator

> **A DEFENSIBLE REPORT**
>
> “Overall it is right 85.6% of the time. But it is right 63.6% of the time on Pullover, and almost all of those errors go to two visually similar queues. If those three desks are adjacent, the practical error rate is much lower than the headline. If they are in different buildings, it is much worse.”

The headline accuracy does not answer the operator’s question. The per-class breakdown does, and it took one extra function call.

Now try to change it

Four things you will want, in the order you will want them.

1. Make it faster
2. Change what it optimises
3. See what the gradients are doing
4. Stop part-way through and look

Try each one now, in the notebook, before the next slide.

Wall 1 · one CPU, and nowhere to move it

20 epochs, 12,000 images, 266,610 parameters, one CPU.

The cost grows in step with the data and the epochs, on the same single core, however large the job becomes.

> **THERE IS NO `device=`**
>
> Scikit-Learn does not support GPUs, and says so in its own FAQ: it is a deliberate scope decision, not an oversight. PyTorch, from the next lecture, gives you the choice — and Lecture 8 measures when it is worth taking.

Wall 2 · you cannot change the loss

```
>>> MLPClassifier(hidden_layer_sizes=(300, 100), loss="mae")
TypeError: MLPClassifier.__init__() got an unexpected
keyword argument 'loss'
```

Cross-entropy is hard-coded. That is fine until the operator says “calling a coat a shirt is cheap, calling a bag a sandal is expensive” — a weighted loss, which is one line in a framework that lets you write one.

✗ And that is the ordinary case, not an exotic one.

Wall 3 · you cannot see a gradient

```
>>> [a for a in dir(clf) if "grad" in a.lower()]
[]
```

Backpropagation ran 20 times per image and computed 266,610 partial derivatives on every batch. None of them was kept.

> **WHY THAT MATTERS IN THREE LECTURES**
>
> Lecture 9 builds a twenty-layer network whose loss barely moves. The diagnosis is a per-layer gradient statistic. You cannot compute it here.

Wall 4 · the smallest unit of control is one epoch

```
>>> inspect.signature(MLPClassifier.partial_fit)
(self, X, y, classes=None)
```

`partial_fit` is the finest handle there is, and one call is one complete pass over everything you hand it.

- No per-batch loss
- No hook between the forward pass and the backward pass
- No way to clip a gradient, or to skip a bad batch, or to log anything

The problem underneath all four

> **THE LOOP WAS WRITTEN FOR YOU**
>
> `fit()` contains a `for` loop over epochs and batches that you did not write, cannot see, and cannot enter. Every one of the four walls is the same wall.

This is not a criticism of Scikit-Learn. `MLPClassifier` is documented as a convenience, and for a tabular problem with a standard loss it is the right tool. It is the wrong tool from here on, and now you know precisely why.

The notebook for this lecture

Open Lecture 7 in Colab

- **Runs on free CPU.** Fashion-MNIST is subsampled to fit it
- The class counts, the scaling and the split; the trivial baseline; the parameter-count arithmetic, checked against what the fitted model reports — a number you can verify on paper first
- The first fit epoch by epoch, the learning curves, five architectures and five learning rates by hand, the confusion matrix, and the walls `fit()` puts round you

> **WHAT TO CHANGE**
>
> Remove the activation — set it to `identity` — and refit. Accuracy collapses towards what a linear model gets, which is the one line of algebra on “Why there must be a nonlinearity”, measured.

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

The task: a radiology-software start-up's chest X-ray triage

> **THE BRIEF**
>
> A radiology-software start-up sells a plug-in for the hospital worklist: for every chest X-ray it raises a **flag**, *may show a tumour*, so a radiologist reads it first; it reorders, never diagnoses. Hospitals pay for a shorter wait on the tumours; a false flag costs radiologist minutes; a miss reaches a patient. It is a **medical device**: cleared for one intended use, every model change one the regulator must see.

- **Given:** past scans from partner hospitals, with the radiologist's report and any biopsy; grouped by patient and hospital
- **Scale:** hundreds of thousands of scans, tumours a few per hundred; thousands a day, flagged within seconds
- **Consumers of the output:** the worklist, the radiologist, the hospital's quality team, the regulator's file

✓ Nine steps follow, each with the pitfall that catches a first attempt and what to do about it. The model at the centre is today's: a multilayer perceptron on the downsampled pixels, fitted with `MLPClassifier`.

Step 1 — Frame it honestly: the intended use decides the metric

- One row per scan, two exclusive classes — a tumour in the report, or not — written down first, because it decides the output layer
- **Count the classes first:** tumours are a few per hundred, so “never flag” scores far above ninety and accuracy hides a useless model; the metric is recall on the tumours at the share of scans the worklist may flag, with the confusion matrix
- Two trivial baselines: never flag, high accuracy and no recall; flag everything, full recall, nothing reordered

> **PITFALL**
>
> The pitch says “right on more than nine scans in ten”, which never flagging also achieves. The regulator asks for sensitivity and specificity at the operating point, under an intended use in one sentence: a triage aid that reorders a queue, not a diagnosis. Written first, it decides the metric now and the device’s label later.

Step 2 — A pixel is a feature, and the split is by patient

- Downsample every scan to one small fixed size and flatten it: one column per pixel. The model knows nothing of which pixels are neighbours — a real loss, paid today, dearest on a small nodule
- Divide by the largest pixel value, so every feature lies in the unit interval: a property of the file format, it fits nothing and is not a leak
- Carve a validation set from the training half with a fixed seed; test once

> **PITFALL**
>
> A random split over *scans* puts one patient's follow-up — the same lungs, the same tumour a month later — on both sides, and the model is scored on tumours it has seen. Split by patient, and hold out a whole hospital as the test set: the device will be sold to hospitals not in the archive, and that is the regulator's number.

Step 3 — Count the parameters before you fit

- A dense layer from d inputs to n units costs $d \times n + n$ parameters, and the first layer — pixels into the first hidden width — holds almost all of them
- That count against the number of training scans is the budget: pick a depth that is enough, grow the width until validation stops improving, and stop there

PITFALL

The scanner delivers millions of pixels a scan, the engineer flattens them as they come, and the first layer alone is billions of parameters for a few hundred thousand scans — nothing fits in memory or generalises. Downsample to the size the count was done for, agreed with the radiologists so the nodule survives it; write the arithmetic down and check it against the fitted model.

Step 4 — The output layer and the loss, paired from the table

Task	Output units	Activation	Loss
✓ Binary — this one	**✓ one**	**✓ sigmoid**	**✓ binary cross-entropy**
Multilabel	one per label	sigmoid	binary cross-entropy
Multiclass	one per class	softmax	cross-entropy

Tumour or not: one unit, a sigmoid, and minus the log probability of the right answer as the loss; `MLPClassifier` chooses both from the labels.

PITFALL

An engineer who wants “normal, benign, tumour” for the radiologist builds three sigmoids — independent yes/no questions whose answers need not sum to one, so the threshold the device is cleared at has no meaning; softmax with squared error trains slowly and badly, and the pairing that would have worked is never suspected. Take the row from the table; add classes only when the intended use says so.

Step 5 — Learning curves: when to stop, and what a subset sweep tells you

- Record the loss and the validation recall at every epoch: training still climbing while validation flattens says the extra capacity is memorising
- Stop where validation stops moving, and read the held-out hospital once — a validation number that chose the epoch is optimistic
- Tune width, depth and the learning rate by hand on a subset, one axis at a time, then rerun the best few on all the scans; log every fit, as the regulator asks how the model was chosen

> **PITFALL**
>
> Ten fits is not a search, and a ranking measured on a fraction of the archive is a measurement of that fraction: the deeper network that lost on the subset may win on all of them. On today's grid one bad learning rate lost more than any structural choice gained — a claim about that grid, not the world.

Step 6 — Where the errors are: the confusion that matters clinically

- One number hides the shape: build the confusion matrix and read its two errors apart. A miss leaves a tumour waiting in the queue like any other scan; a false flag costs a radiologist minutes on a normal one. Not the same price
- Then recall per subgroup: small nodules against large, each hospital's scanner, portable against fixed, so the misses have an address

> **PITFALL**
>
> The headline recall is defended at the hospital's quality meeting while the misses sit in one place — the small nodules the downsampling erased, or the one hospital with an older scanner — and those are the ones that reach a patient. Report recall per subgroup, which errors go where, and the price of each; the regulator's file carries the per-hospital table, not the headline.

Step 7 — Abstain to the radiologist, and never say “clear”

- The sigmoid output is the model’s score. Sweep the threshold on the *validation* set and draw two curves — the share flagged, and the recall among the tumours; the radiologists pick the point, since their minutes and a miss both have prices
- Three outputs reach the worklist: *flagged*, *not flagged*, and *not assessed* — a scan whose metadata or reconstruction error says it is unlike the archive

> **PITFALL**
>
> The sigmoid must give a number for every scan, so a lateral view gets a confident score of something; and a radiologist who reads “not flagged” as “normal” reads less carefully — the device has started to diagnose. Bucket the validation scans by score and check that flagged scans are tumours as often as the number claims; show the score only if it passes; abstain visibly out of scope.

Step 8 — In the product: what the wrapper cannot do

The hospital asks for	The wrapper has
a loss that charges a miss more than a false flag	cross-entropy, hard-coded
a flag within seconds	one CPU, no `device=`
a per-batch log, a stop on a bad batch	one epoch as the smallest unit
a model that knows which pixels are neighbours	a table of pixels; convolution later

PITFALL

The prototype ships as the device: the weighted loss has nowhere to go, and a quiet retrain puts a model in hospitals the regulator never saw. Four walls, one wall: `fit()` holds a loop you did not write. Keep the wrapper for the prototype; the product's model runs in a loop you own, every retrain is a numbered version in the device file, released per hospital in *shadow mode* — flags computed, not shown, compared with the radiologists' reads — then monitored, a safety officer on call.

Step 9 — Ship it: the pipeline, and the pitfalls in one table

Archive → downsampled, scaled → split by patient, a hospital held out → parameters counted → one unit, sigmoid, cross-entropy → curves, the subset ranking rerun → confusion matrix, recall per subgroup → threshold on validation, abstention out of scope → the held-out hospital once → a numbered version in shadow mode.

Pitfall	Address it by
Accuracy that never flagging also achieves	recall at the operating point under the intended use
One patient on both sides of the split	splitting by patient; a hospital held out
A first layer nobody priced	the count on paper, checked against the fit
Three sigmoids, or softmax with squared error	the binary row of the table
A subset ranking trusted	the best few rerun on all scans, every fit logged
Headline recall over concentrated misses	recall per subgroup and hospital, the price of each
“Not flagged” read as clear; a score out of scope	a threshold checked bucket by bucket; visible abstention
The prototype shipped as the device	owning the loop: numbered versions, shadow mode

What we did

1. Wrote a layer as one matrix product and one elementwise non-linearity, $\mathbf{H} = \phi(\mathbf{X}\mathbf{W} + \mathbf{b})$ — and saw that without the non-linearity any stack collapses to a single affine map
2. Counted the parameters before fitting: 266,610, almost all in the first layer
3. Read the classification table: ten exclusive classes means ten units, softmax and cross-entropy, and the pairing is not optional
4. Checked the class counts before choosing accuracy, anchored it at a 10% baseline, fitted a network with `MLPClassifier`, tuned it by hand, and split the accuracy by class
5. Named four things that object will not let you do — and saw that all four are the same thing: the training loop is written for you, and you cannot enter it
6. Designed a radiology-software start-up's chest X-ray triage in nine steps — recall at the operating point under a written intended use, a split by patient with a hospital held out, the parameters counted at the downsampled size, the binary row of the table, the curves read for the stopping epoch, the confusion that matters clinically, abstention to the radiologist, and the loop owned in a numbered version of a medical device run in shadow mode — each step named for the pitfall it avoids

NOT PRESENTED — FOR AFTERWARDS

Exercises Lecture 7

five, in the style of the written paper

Exercises — Lecture 7

1 A multilayer perceptron with the activation removed is written as $W_3 W_2 W_1 x$. State what function class it can represent, and why depth then buys nothing. [4]

2 Give the parameter count of a dense layer from m inputs to n units, and evaluate it for the first layer of a network on 28×28 images with 300 units. [4]

3 A ten-class classifier for mutually exclusive classes is built with ten sigmoid output units and binary cross-entropy. Name the row of the classification architecture table this is, and the row it should be. [4]

Solutions on Lecture 8's deck.

Exercises — Lecture 7 (continued)

4 Fashion-MNIST is exactly balanced across ten classes. State the trivial baseline's accuracy, and why balance is what makes accuracy defensible here. [3]

5 An architecture sweep finds the best hidden-layer size on this dataset. State what that result does and does not transfer to. [4]

Solutions on Lecture 8's deck.

THE FIVE SET AT THE END OF LECTURE 6

Solutions Lecture 6

not part of this lecture

Solutions — Lecture 6

1. A marketing team wants every customer placed in one of a handful of segments; a fraud team wants the few…

✓ Segments: k-means, because every account must land in one of a few compact groups and the number of groups is a business choice. Anomalies: DBSCAN, because its noise label is exactly the output wanted, with no k to choose.

- Behaviour data has no empty space between groups, so DBSCAN calls most accounts noise or everything one cluster, and nobody can write to noise
- k-means forces every transaction into a cluster, so an anomaly becomes a far-off member of the nearest one and drags its centroid

2. The explained variance ratio is sometimes called a heuristic for choosing d. Say why it is not one, and…

✓ The principal subspace leaves exactly $\sum_{j>d} \sigma_j^2$, so $\sum_{j\le d} r_j$ is exactly the fraction of squared error removed. Keeping 95% of the variance leaves 5% of the total squared error of the centred data.

- The minimum of $E(\mathbf{W})$ is the tail of the spectrum, and the total is $\sum_j \sigma_j^2$, so the ratio is the theorem normalised, not a rule of thumb
- On the Olivetti faces 123 of 4,096 components reach 95%, so the price of a factor of 33 fewer numbers is stated before d is chosen

Solutions — Lecture 6 (2)

3. A customer-segmentation job reruns every month on fresh orders, and the marketing team complains that the…

✓ Fix `n_init` and `random_state`, and keep the scaler with the model so the columns are in the same units every month. Then cluster two resamples of the accounts, match the segments by overlap and count the accounts that keep theirs: a segment that does not survive a resample is not one.

- Lloyd's algorithm converges to a local minimum that depends on

4. Inertia always falls as k rises, so it cannot choose k. State what silhouette adds, and one situation…

✓ Silhouette compares within-cluster to nearest-other-cluster distance, so it is penalised for splitting a group and for merging two, and has an interior maximum. It fails to decide when the clusters overlap with no clean gaps: the curve is nearly flat near its maximum.

- Inertia is non-increasing in k by construction — refining a partition cannot lengthen a distance to the assigned centre
- Where two people's photographs are about as far apart as two of

Solutions — Lecture 6 (3)

5. You have 4,096-dimensional face vectors and are told two faces are “close”. Explain why that…

✓ Distance is not identity — lighting and pose move a face further than identity does.

- Pixel distance is dominated by whatever varies most, and that is illumination
- The metric has to be one under which the thing you care about is the thing that varies

LECTURE 8 · GÉRON CH. 10

PyTorch

Backpropagation as reverse-mode automatic differentiation, derived
What the wrapper costs, and when an accelerator pays
Tensors, autograd, and the training loop written out in full
Two bugs that never raise an exception

Run the Lecture 7 notebook first