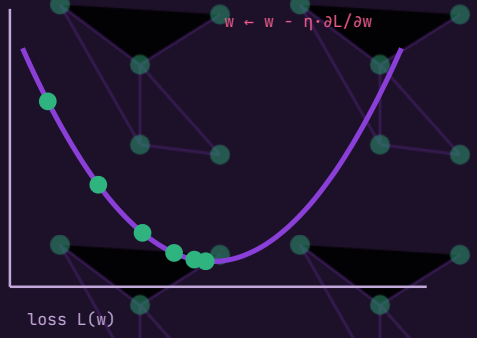
STUDY PAGE · AREA 02 OF 14

Math for ML

02

Just the maths that ML engineers use: vectors and dot products for predictions, gradients for learning, probability for outputs and statistics for knowing when to trust a number — all in code.



16 h of study · 4 modules · 10 diagrams · 3 interactives

Free tools · Python + NumPy, or Google Colab · 20-question test · 70% to pass

Lab: gradient descent from scratch

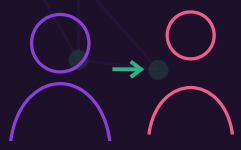
1 · Linear algebra → 2 · Calculus → 3 · Probability → 4 · Statistics → ab · gradient descen → Area test · 70%

dot · matmul · cosine | gradients · chain rule | Bayes · softmax | spread · CLT · CI | three maths bugs | 20 questions

BEFORE YOU START · A

Who this area is for

You do not need a maths degree. This area teaches the four ideas every model uses, each shown in NumPy so you see what the symbols compute.



CS / data / stats students

Students preparing for ML internships and new-grad roles who want the practical path, not only theory.

Software engineers

Backend and full-stack developers moving into ML engineering, who already write good code and want the ML on top.

Data analysts

Analysts who know SQL and spreadsheets and want to build and ship models, not just report on data.

QA & test engineers

Test automation engineers heading into ML: evaluation, data checks and model testing use the same mindset.

Data engineers

Pipeline builders who want to own the models their data feeds, and the MLOps around them.

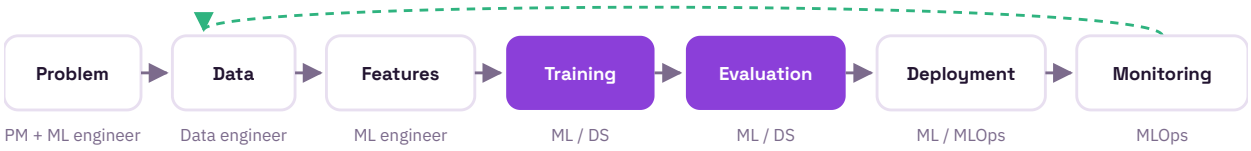
Career switchers

Engineers from other fields (EE, mechanical, finance) with some Python, aiming for an ML engineer job.

Prerequisites

You need	Level	Where to get it
Python and NumPy arrays	Comfortable	Area 01
School algebra (y = mx + c, powers)	Basic	Any refresher; Module 2 recaps slopes
Calculus	None	Module 2 starts from slopes

Where this area sits in the ML lifecycle — and the roles that use it



Highlighted: the maths here drives training and evaluation

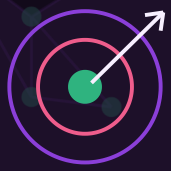
Gradients train models; statistics judge them. Linear algebra makes both fast.

Role	How you use this area on the job	Typical first title
ML engineer	Debug training (exploding loss, bad learning rates) and read model code fluently	ML Engineer I / Machine Learning Engineer
Data scientist	Choose distributions, build confidence intervals, avoid causal claims from correlations	Data Scientist / Applied Scientist
AI / LLM engineer	Understand embeddings, cosine similarity and softmax over tokens	AI Engineer / LLM Engineer
MLOps / platform	Reason about numerical stability and monitoring statistics	MLOps Engineer / ML Platform Engineer
Data engineer	Compute correct aggregate statistics at scale	Data Engineer

BEFORE YOU START · B

What you will be able to do

Every outcome is practised in the lab and checked by the area test.

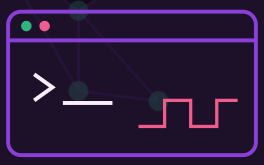


- Compute dot products and matrix products and predict their shapes.
- Measure distance and cosine similarity between vectors.
- Explain derivatives and gradients as slopes and use them to minimise a loss.
- Choose a learning rate by its effect on convergence.
- Apply the chain rule to a small model.
- Use Bayes' rule and explain the base-rate effect.
- Turn scores into probabilities with sigmoid and softmax.
- Summarise data, compute correlations and bootstrap a confidence interval.

BEFORE YOU START · C

Tools & software

Only Python and NumPy are needed; Matplotlib helps you see curves.



Tool	What it does	Used in this area for	Cost
Python 3.10 – 3.13	The language of ML: every library on this page is a Python package	Every example and the lab	Free
venv + **pip**	An isolated environment per project and the package installer	Installing the area's packages	Free
NumPy	Fast n-dimensional arrays and maths	Vectors, matrices, gradients, sampling	Free
Matplotlib	Charts: histograms, scatter plots, learning curves	Plotting losses and distributions	Free
JupyterLab	Notebooks: run code cell by cell and see tables and plots inline	Playing with formulas cell by cell	Free
Google Colab	Hosted notebooks in the browser, with a free GPU on request	No-install option	Free

Platforms you will meet at work

Platform	Vendor	What teams use it for
SageMaker	AWS	Managed notebooks, training jobs, model registry and endpoints
Vertex AI	Google Cloud	Training, pipelines, feature store, model serving and Gemini APIs
Azure Machine Learning	Microsoft	Workspaces, MLflow tracking, managed endpoints
Databricks	Databricks	Spark + notebooks + MLflow on a lakehouse
Weights & Biases	W&B	Experiment tracking and model evaluation dashboards

Same code, bigger machines

The same NumPy maths is what PyTorch and JAX run on GPUs; the notation does not change.

BEFORE YOU START · D

Install step by step

Pick your computer. About 15 minutes. If anything fails, Google Colab works with nothing installed.



Windows

Recommended: WSL 2 (Ubuntu inside Windows) — the same commands as Linux, cloud servers and most ML teams.

1

Turn on WSL and install Ubuntu

Open **PowerShell as Administrator**, run the command, then restart.

```
wsl --install
```

After the restart an Ubuntu window opens: choose a user name and password.

2

Install Python's venv support

```
sudo apt update
sudo apt install -y python3 python3-venv python3-pip git make
```

3

Create an environment and install the packages

```
python3 -m venv ~/ml
source ~/ml/bin/activate
pip install --upgrade pip
pip install numpy matplotlib jupyterlab
```

Run `source ~/ml/bin/activate` in every new terminal.

4

Editor

Install VS Code, the **WSL**, **Python** and **Jupyter** extensions, then type `code .` in Ubuntu.

5

Check it works

```
python -c "import numpy; print(numpy.__version__)"
```

i

No WSL on your laptop?

Install Python 3.12 from python.org (tick *Add python.exe to PATH*), then in PowerShell: `py -m venv %USERPROFILE%\ml` , `%USERPROFILE%\ml\Scripts\activate` and the same `pip install` line. Every lab runs the same.

macOS

1 **Install Apple's command-line tools**

```
xcode-select --install
```

2 **Install Homebrew and Python**

Follow the one-line command on brew.sh, then:

```
brew install python@3.12 git make
```

3 **Create an environment and install the packages**

```
python3 -m venv ~/ml
source ~/ml/bin/activate
pip install --upgrade pip
pip install numpy matplotlib jupyterlab
```

4 **Editor**

Install VS Code with the **Python** and **Jupyter** extensions. Apple-silicon Macs run every lab natively.

5 **Check it works**

```
python -c "import numpy; print(numpy.__version__)"
```

Linux

1 **Ubuntu / Debian**

```
sudo apt update
sudo apt install -y python3 python3-venv python3-pip git make
```

2 **Fedora**

```
sudo dnf install -y python3 python3-pip git make
```

3 **Arch**

```
sudo pacman -S python python-pip git make
```

4 **Create an environment and install the packages**

```
python3 -m venv ~/ml
source ~/ml/bin/activate
pip install --upgrade pip
pip install numpy matplotlib jupyterlab
```

5 **Check it works**

```
python -c "import numpy; print(numpy.__version__)"
```

Browser only

1 **Open Google Colab**

Go to colab.research.google.com and sign in with a Google account.

2 **New notebook**

File → New notebook. NumPy, pandas, scikit-learn, Matplotlib and PyTorch are already installed.

3 **Upload the lab**

Click the folder icon, upload the lab zip, then run `!unzip lab.zip && cd lab && python check.py` in a cell (prefix shell commands with `!`).

4 **Check the versions**

```
python -c "import numpy; print(numpy.__version__)"
```

BEFORE YOU START · E

First run in 5 minutes

A dot product, a numerical slope and a mean — the three ideas of this area in eight lines.



1

Create `first.py`

```
import numpy as np

X = np.array([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]])
w = np.array([0.5, -1.0])
print("X @ w      =", X @ w)                 # a dot product per row
f = lambda v: (v - 3) ** 2
h = 1e-6
print("slope at 1 =", round((f(1 + h) - f(1 - h)) / (2 * h), 4))   # nu
print("mean, std  =", X.mean(axis=0), X.std(axis=0).round(3))
```

2

Run it

```
python first.py
```

Output (captured from a real run):

```
X @ w      = [-1.5 -2.5 -3.5]
slope at 1 = -4.0
mean, std  = [3. 4.] [1.633 1.633]
```

3

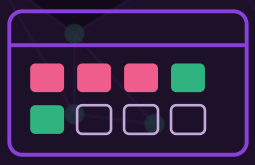
What to notice

The slope of $(w - 3)^2$ at $w = 1$ is -4: the loss goes down if w increases. That one number is all gradient descent needs to take a step.

BEFORE YOU START · F

Study plan

16 hours at about 8 hours a week.

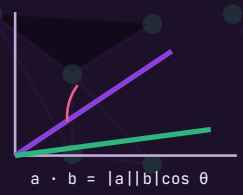


Session	Do this	Time
Day 1	Install · first run · Module 1 (vectors, matrices) + dot-product widget	3.5 h
Day 2	Module 2 (derivatives, gradient descent) + playground	4 h
Day 3	Module 3 (probability, Bayes, softmax) + calculator	3.5 h
Day 4	Module 4 (statistics, bootstrap)	3 h
Day 5	Lab: gradient descent · area test	2 h

MODULE 1 OF 4

Vectors, matrices & dot products

How a model turns features into a prediction, one dot product at a time.



- Vectors and matrices as data
- Dot products and X @ w
- Matrix shapes
- Distance and cosine similarity

⏱ 3.5 hours · Area 02 · Math for ML

Start from zero

New to this? Work through these steps first, in order. Already comfortable? Skim them and jump to the reference below.

STEP 1

A vector is a list of measurements

In plain words

Describe one thing with several numbers — a house by rooms, age and size — and you have a vector.

Think of it like

A recipe card's ingredient amounts: flour 200, sugar 50, eggs 2. Same card format for every recipe.

Worked example

	rooms	age	size
house A	3	1	4
house B	2	5	1

The word for it

Vector (one row of features); a stack of them is a **matrix**.

Why it matters — Every example a model sees is a vector of features; the whole dataset is a matrix X.

2 STEP 2 — The dot product is a weighted total

In plain words — Multiply each number by a weight and add everything up.

Think of it like — A shopping bill: quantities × prices, summed. Quantities are the features, prices the weights.

Worked example

1. x = [3, 1, 4], w = [2, −1, 0.5]
2. 3×2 = 6, 1×(−1) = −1, 4×0.5 = 2
3. 6 − 1 + 2 = 7
4. add the bias b = 1 → prediction 8

The word for it — **Dot product** x·w; with a bias it is a **linear model**.

Why it matters — Linear regression, logistic regression and every neuron in a network compute exactly this.

PAUSE & TRY **[1, 2] · [3, 4] = ?** Answer

1×3 + 2×4 = 11.

3 STEP 3 — Matrix multiplication = many dot products at once

In plain words — Put all examples in a matrix X and all weights in w; `X @ w` computes every prediction in one go.

Think of it like — Pricing a whole basket of orders at once instead of one receipt at a time.

Worked example

```python
import numpy as np
X = np.array([[3, 1, 4], [2, 5, 1]])
w = np.array([2, -1, 0.5])
print(X @ w + 1)
```

```
# output
[8.  0.5]
```

The word for it — **Matrix product**; shape rule (n, d) @ (d,) → (n,).

Why it matters — GPUs are matrix-multiplication machines — this is why deep learning is fast.

4 STEP 4 — Similar things point the same way

In plain words — Two vectors are similar if they point in the same direction, even if one is longer.

Think of it like — Two people walking north-east: one walked further, but they head the same way.

Worked example

pair	cosine
same direction	1.0
right angle	0.0
opposite	−1.0

The word for it — **Cosine similarity**.

Why it matters — Search, recommendations and RAG compare embeddings with cosine similarity.

Going further: the reference

The same ideas at the next level: precise definitions, tables, rules and interview questions.

Key definitions

Scalar

A single number, like a price or a learning rate.

Vector

An ordered list of numbers. One example's features form a vector, e.g. [rooms, age, size].

Matrix

A 2-D grid of numbers. A dataset X is a matrix: one row per example, one column per feature.

Dot product

Multiply matching entries and add: $a \cdot b = \Sigma\, a_i b_i$. A linear model's prediction is $x \cdot w + b$.

Transpose

Flip rows and columns: X^T. A (5, 3) matrix becomes (3, 5).

Matrix multiplication

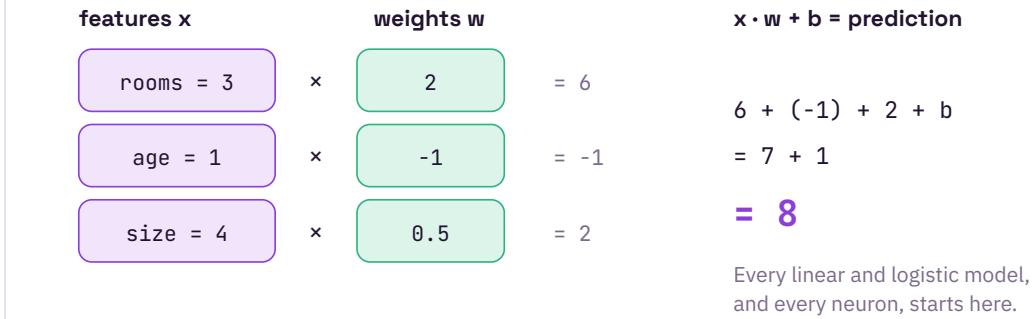
(n, d) @ (d, k) → (n, k): each result entry is a row-column dot product.

Norm

A vector's length: $\|x\| = \sqrt{(\Sigma\, x_i^2)}$. Used for distances and regularization.

Cosine similarity

$a \cdot b \,/\, (\|a\| \|b\|)$: direction similarity from −1 to 1; the basis of text and image embeddings search.



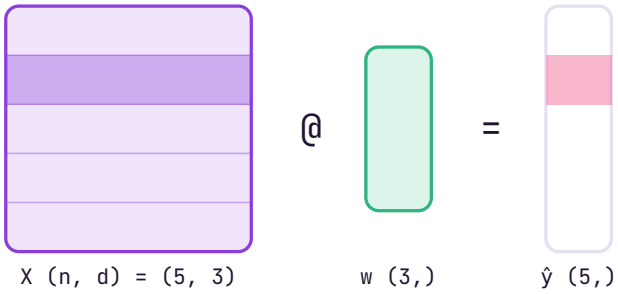
A prediction is a weighted sum. The weights say how much each feature matters and in which direction; training is finding good weights.

```python
import numpy as np

x = np.array([3, 1, 4])           # one house: rooms, age, size
w = np.array([2, -1, 0.5])        # learned weights
b = 1.0
print("x · w + b =", x @ w + b)

X = np.array([[3, 1, 4],          # 3 houses → a (3, 3) matrix
              [2, 5, 1],
              [4, 0, 6]])
print("all predictions:", X @ w + b)
print("X.T shape:", X.T.shape)
```

```
# output
x · w + b = 8.0
all predictions: [ 8.   0.5 12. ]
X.T shape: (3, 3)
```



Shape rule: (n, d) @ (d, k) → (n, k)

The inner sizes (d) must match;
the outer sizes (n, k) are the result.

Highlighted row · w = one prediction

All 5 predictions in one call: X @ w

Shapes tell you if the maths is legal. Write the shapes next to every matrix product before you run it.

Distance and similarity

Two vectors can be compared by how far apart they are (Euclidean distance, used by k-NN and k-means) or by the angle between them (cosine similarity, used to compare text embeddings in search and RAG).

```python
import numpy as np

a = np.array([1.0, 2.0, 2.0])
b = np.array([2.0, 4.0, 4.0])     # same direction, twice as long
c = np.array([-2.0, 1.0, 0.0])    # at a right angle to a

def cosine(u, v):
    return u @ v / (np.linalg.norm(u) * np.linalg.norm(v))

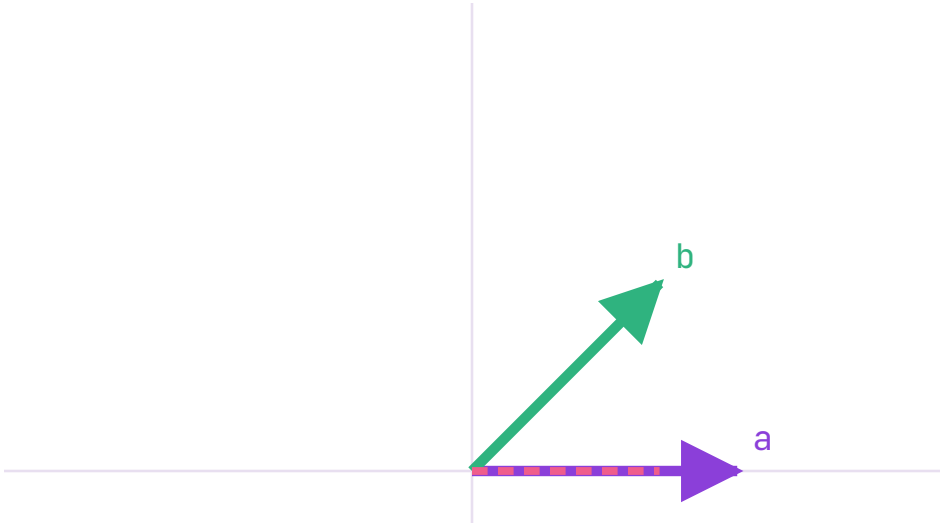
print("‖a‖ =", np.linalg.norm(a))
print("distance(a, b) =", np.linalg.norm(a - b))
print("cosine(a, b) =", round(cosine(a, b), 3), " cosine(a, c) =", round(cosine(
```

```
# output
‖a‖ = 3.0
distance(a, b) = 3.0
cosine(a, b) = 1.0  cosine(a, c) = 0.0
```

Try it: dot product and angle

angle between a and b

length of b




```
a = [3, 0]   b = [2.12, 2.12]
a · b = 6.36   cos θ = 0.707 → similar direction
(the dashed line is b's shadow on a: a·b = |a| × shadow length)
```

Operation	NumPy	Shape rule
Dot product	`a @ b` or `np.dot(a, b)`	(d,)·(d,) → scalar
Matrix × vector	`X @ w`	(n,d)@(d,) → (n,)
Matrix × matrix	`A @ B`	(n,d)@(d,k) → (n,k)
Transpose	`X.T`	(n,d) → (d,n)
Length	`np.linalg.norm(x)`	(d,) → scalar
Element-wise	`a * b`	same shape (or broadcast)

`*` is not matrix multiplication

In NumPy `A * B` multiplies element by element; `A @ B` is the matrix product. Mixing them up gives an answer with the wrong shape — or, worse, the right shape and wrong numbers.

Common mistakes

Using `*` where you meant `@` · mismatched inner dimensions · comparing embeddings with Euclidean distance when their lengths differ (use cosine) · forgetting the bias term b.

INTERVIEW CORNER

Q. What does a dot product tell you? **A.** A weighted sum; geometrically $\|a\|\|b\|\cos\theta$, so it is large when vectors point the same way.

Q. Shapes of X @ W for X (64, 784), W (784, 128)? **A.** (64, 128): the inner 784s match and disappear.

Key takeaways

1. A model's prediction is a dot product plus a bias.
2. (n, d) @ (d, k) → (n, k): check shapes first.
3. Distance for nearness, cosine for direction.

MODULE 2 OF 4

Derivatives, gradients & the chain rule

How a model learns: follow the slope of the loss downhill.

- Derivatives as slopes
- Gradient descent and the learning rate
- Gradients of many weights
- The chain rule and backprop



⏱ 4 hours | Area 02 · Math for ML

Start from zero

New to this? Work through these steps first, in order. Already comfortable? Skim them and jump to the reference below.

STEP 1

1 A derivative is a slope

≈ In plain words

The derivative tells you how much the output changes when you nudge the input a little.

☀ Think of it like

The steepness of a road at the spot where you stand — uphill, flat or downhill.

Worked example

```python
f = lambda w: (w - 3) ** 2
h = 0.001
for w in (1, 3, 5):
    print(w, round((f(w + h) - f(w - h)) / (2 * h), 3))
```

```
# output
1 -4.0
3 0.0
5 4.0
```

The word for it — **Derivative** f′(w); for (w − 3)² it is 2(w − 3).

Why it matters — The slope of the loss tells you which way to move each weight.

2 STEP 2 — Walk downhill: gradient descent

In plain words — Look at the slope, take a small step the opposite way, repeat. The loss gets smaller each time.

Think of it like — Finding the bottom of a valley in fog: feel the ground, step downhill, repeat.

Worked example

1. w = 0, slope f′(0) = −6 (downhill is to the right)
2. step: w ← 0 − 0.1 × (−6) = 0.6
3. slope at 0.6 is −4.8 → w ← 1.08
4. … steps shrink as the slope flattens near w = 3

The word for it — **Gradient descent** with a **learning rate** (step size) η.

Why it matters — This loop — compute slope, step, repeat — trains almost every model in this roadmap.

PAUSE & TRY **At w = 5 the slope of $(w - 3)^2$ is +4. With η = 0.1, where does w go?** Answer

5 − 0.1 × 4 = 4.6: downhill, toward 3.

3 STEP 3 Many weights → a gradient

≈ **In plain words** With many weights, measure the slope for each one separately. Together these slopes are the gradient.

☀ **Think of it like** A hillside map with a slope arrow at each spot: the arrow points the steepest way up.

✎ **Worked example**

weight	slope ∂L/∂w	step direction
w_1	+2.0	decrease
w_2	−0.5	increase
b	0.0	stay

Aa **The word for it** **Gradient** ∇L — the vector of partial derivatives.

! **Why it matters** A network with a million weights gets a million slopes per step; the update rule is the same.

4 STEP 4 Chain rule: slopes multiply

≈ **In plain words** If A affects B and B affects C, the effect of A on C is (effect of A on B) × (effect of B on C).

☀ **Think of it like** Exchange rates: dollars → euros → yen. Multiply the two rates to go dollars → yen.

Worked example

1. $u = 2w$, $L = u^2$
2. $dL/du = 2u$, $du/dw = 2$
3. $dL/dw = 2u \times 2 = 8w$

The word for it

Chain rule; done backwards through a network it is **backpropagation**.

Why it matters

It is how deep learning gets gradients for every layer.

Going further: the reference

The same ideas at the next level: precise definitions, tables, rules and interview questions.

Key definitions

Derivative

How fast a function's output changes as its input changes: the slope of the curve at a point.

Partial derivative

The derivative with respect to one input while holding the others fixed: $\partial L/\partial w_1$.

Gradient

The vector of all partial derivatives. It points uphill — the direction of steepest increase.

Loss function

A number that says how wrong the model is, e.g. mean squared error. Training minimises it.

Gradient descent

Repeat: compute the gradient, take a small step the opposite way: $w \leftarrow w - \eta \nabla L$.

Learning rate (η)

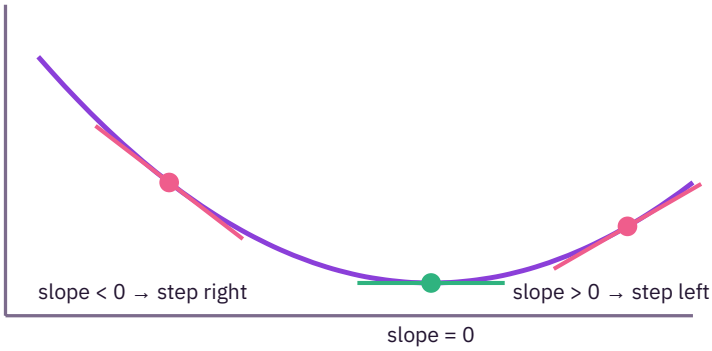
The step size. Too small is slow; too large overshoots or diverges.

Chain rule

The derivative of a composition is the product of the local derivatives: $(f \circ g)' = f'(g) \cdot g'$.

Backpropagation

The chain rule applied backwards through a network to get every weight's gradient at once.



f(w) = (w − 3)²

`f'(w) = 2(w - 3)`

w = 1 → f' = −4 (downhill to the right)
w = 3 → f' = 0 (the minimum)
w = 5 → f' = +4 (downhill to the left)

Gradient descent: w ← w − η · f'(w)

The sign of the slope says which way is downhill. Gradient descent steps against it; at the minimum the slope is 0 and the steps stop.

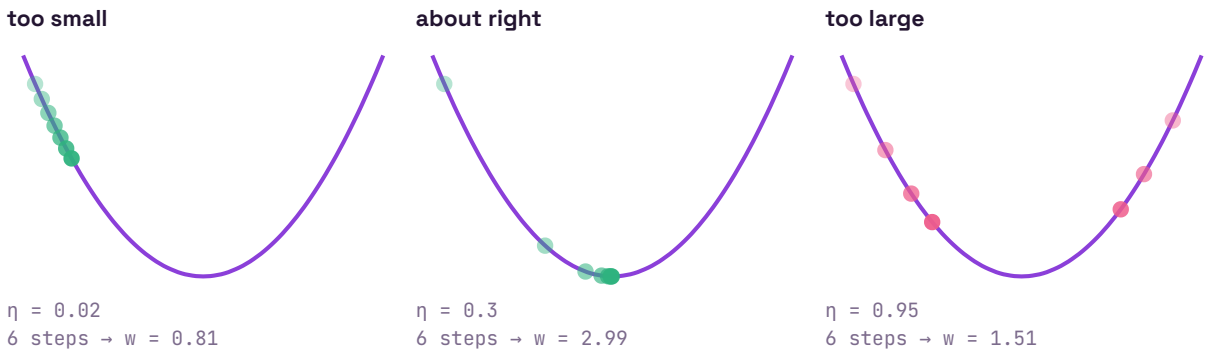
Function	Derivative	Used in
c (constant)	0	biases
x^2	2x	squared error
x^n	$n \cdot x^{n-1}$	polynomials
e^x	e^x	softmax, sigmoid
ln x	1/x	log-loss, cross-entropy
$\sigma(x) = 1/(1+e^{-x})$	$\sigma(x)(1 - \sigma(x))$	logistic regression, neurons
max(0, x) (ReLU)	1 if x > 0 else 0	deep networks

Gradient descent on one weight

```
def f(w):  return (w - 3) ** 2          # loss, minimum at w = 3
def df(w): return 2 * (w - 3)           # its derivative

w, lr = 0.0, 0.1
for step in range(1, 26):
    w = w - lr * df(w)                  # step against the slope
    if step in (1, 2, 5, 10, 25):
        print(f"step {step:2}: w = {w:.4f}  loss = {f(w):.6f}")
```

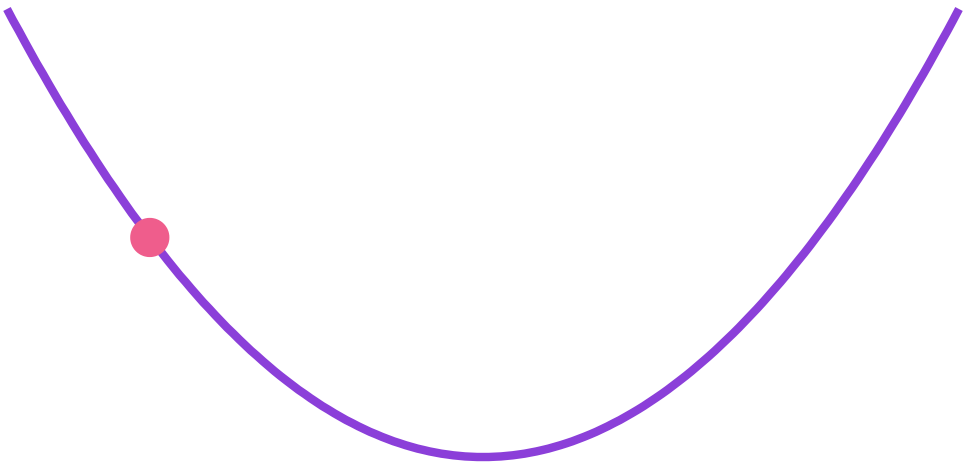
```
# output
step  1: w = 0.6000  loss = 5.760000
step  2: w = 1.0800  loss = 3.686400
step  5: w = 2.0170  loss = 0.966368
step 10: w = 2.6779  loss = 0.103763
step 25: w = 2.9887  loss = 0.000128
```



The learning rate is the most important knob. Here, on f(w) = (w − 3)² starting at 0.2: η = 0.02 crawls, 0.3 converges fast, 0.95 bounces across the valley.

Try it: gradient descent playground

learning rate η **η = 0.10**



w = 3 (minimum)

```
step 0: w = 0.2000   f(w) = 8.8400   f'(w) = -5.6000
```

```
next: w ← w - 0.10 × f'(w)
```

Many weights: the gradient

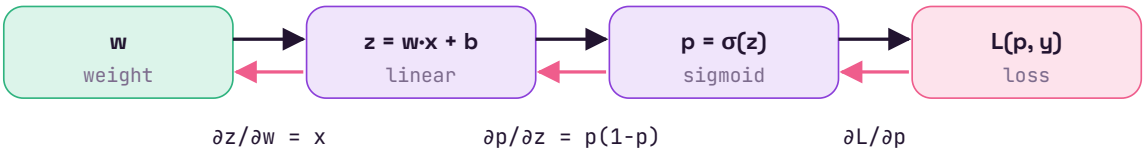
With weights $w_1 \ldots w_d$ the loss depends on all of them. The gradient ∇L collects one partial derivative per weight. For linear regression with mean squared error the gradient has a closed form you will implement in the lab:

```python
import numpy as np

rng = np.random.default_rng(0)
X = rng.normal(size=(100, 2))
y = X @ np.array([3.0, -2.0]) + 1.0

w, b = np.zeros(2), 0.0
for epoch in range(200):
    err = X @ w + b - y
    dw = (2 / len(y)) * X.T @ err        # ∂L/∂w, shape (2,)
    db = (2 / len(y)) * err.sum()        # ∂L/∂b
    w, b = w - 0.1 * dw, b - 0.1 * db
print("w =", w.round(3), " b =", round(b, 3))
```

```
# output
w = [ 3. -2.]  b = 1.0
```



$$\partial L/\partial w = \partial L/\partial p \cdot \partial p/\partial z \cdot \partial z/\partial w$$

Multiply the local slopes backwards along the path — this is backpropagation.

The chain rule turns a deep model into a product of easy slopes. PyTorch's `loss.backward()` (Area 07) does exactly this for millions of weights.

★ **Check gradients numerically**

Nudge a weight by ε in each direction and measure the loss change: (L(w+ε) − L(w−ε)) / 2ε. If it disagrees with your formula, the formula is wrong. The lab uses this to catch the gradient bug.

× **Common mistakes**

Stepping with the gradient instead of against it (planted bug 3) · forgetting to average over the batch (planted bug 2 — the step size then grows with the dataset) · a learning rate so large the loss becomes NaN · not scaling features, so one weight needs a tiny step and another a big one.

INTERVIEW CORNER

Q. Why subtract the gradient? **A.** The gradient points uphill (steepest increase of the loss); minimising means moving the opposite way.

Q. What happens if the learning rate is too high? **A.** Steps overshoot the minimum; the loss oscillates or explodes to inf/NaN.

Key takeaways

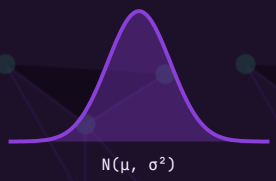
1. Derivative = slope; gradient = all the slopes.
2. $w \leftarrow w - \eta \cdot \nabla L$, repeated, is how models learn.
3. The chain rule (backprop) makes it work for deep models.

MODULE 3 OF 4

Probability & distributions

Reason about uncertainty and read a model's probabilities.

- Conditional probability and Bayes
- Common distributions
- Sigmoid and softmax
- Cross-entropy



3.5 hours · Area 02 · Math for ML

Start from zero

New to this? Work through these steps first, in order. Already comfortable? Skim them and jump to the reference below.

STEP 1

Probability measures how likely

In plain words

A probability is a number from 0 (never) to 1 (certain).

Think of it like

A weather forecast: 0.3 chance of rain means it rains on about 3 of 10 days like this one.

Worked example

```
import numpy as np
rng = np.random.default_rng(0)
print((rng.random(10_000) < 0.3).mean())
```

```
# output
0.3008
```

The word for it

Probability; the long-run frequency of an event.

! **Why it matters** Classifiers output probabilities; you pick the threshold that turns them into decisions.

2 STEP 2 “Given that” changes everything

≈ **In plain words** The chance of A when you already know B can be very different from the chance of B given A.

✳ **Think of it like** Most basketball players are tall; most tall people are not basketball players.

✎ **Worked example**

question	answer
P(tall | basketball player)	high
P(basketball player | tall)	low

Aa **The word for it** **Conditional probability** P(A | B); **Bayes' rule** flips it using the base rate.

! **Why it matters** It explains why rare-event detectors (fraud, disease) produce many false alarms.

PAUSE & TRY **Fraud is 0.1% of transactions. Your model flags 2% of all transactions. At best, what share of flags can be real fraud?** Answer

At most 0.1 / 2 = 5%.

3 STEP 3 Scores become probabilities

In plain words

A model produces a raw score; sigmoid squashes it between 0 and 1, softmax turns several scores into probabilities that add to 1.

Think of it like

Converting raw votes into percentages of the total.

Worked example

```
import numpy as np
s = np.array([2.0, 1.0, 0.1])
print((np.exp(s) / np.exp(s).sum()).round(2))
```

```
# output
[0.66 0.24 0.1 ]
```

The word for it

Sigmoid, **softmax**.

Why it matters

Every classifier and every LLM next-token prediction ends in one of these.

Going further: the reference

The same ideas at the next level: precise definitions, tables, rules and interview questions.

Key definitions

Probability

A number from 0 to 1 for how likely an event is.

Conditional probability

P(A | B): the probability of A given that B happened.

Bayes' rule

P(A | B) = P(B | A)·P(A) / P(B): flip a conditional using the base rate P(A).

Random variable

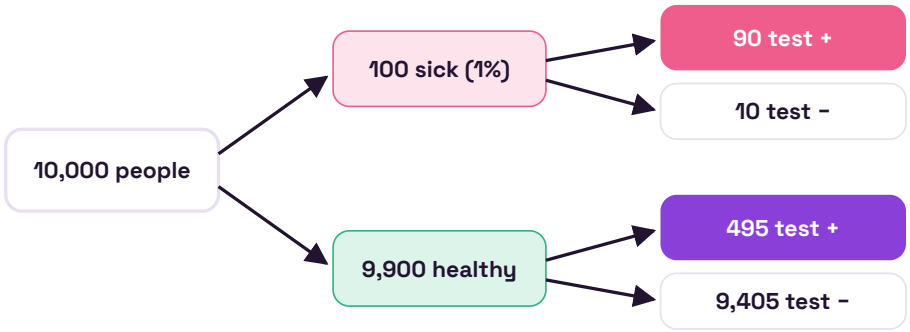
A quantity whose value is uncertain, e.g. "clicks tomorrow".

Distribution
All possible values of a random variable and how likely each is.

Bernoulli / binomial
Yes/no outcome with probability p; the count of yeses in n tries.

Normal (Gaussian)
The bell curve $N(\mu, \sigma^2)$; sums and averages of many small effects tend to be normal.

Expected value
The long-run average: $E[X] = \Sigma\, x \cdot P(x)$.



P(sick | +)
`= 90 / (90 + 495)`
≈ 15%

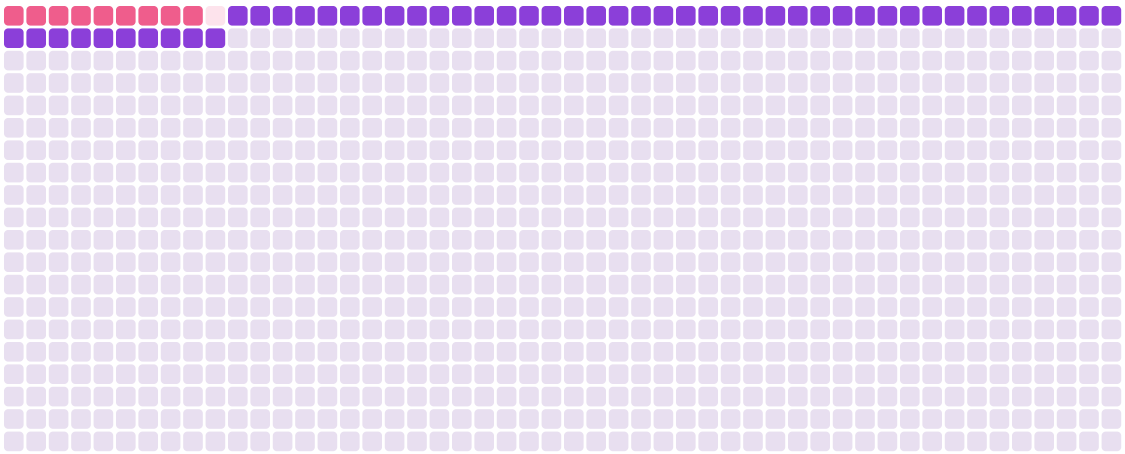
Rare events + imperfect tests = many false alarms.

The base rate dominates. A 90%-sensitive test for a 1% condition gives mostly false positives — the same reason fraud and spam models need careful thresholds (Area 05).

Try it: Bayes calculator

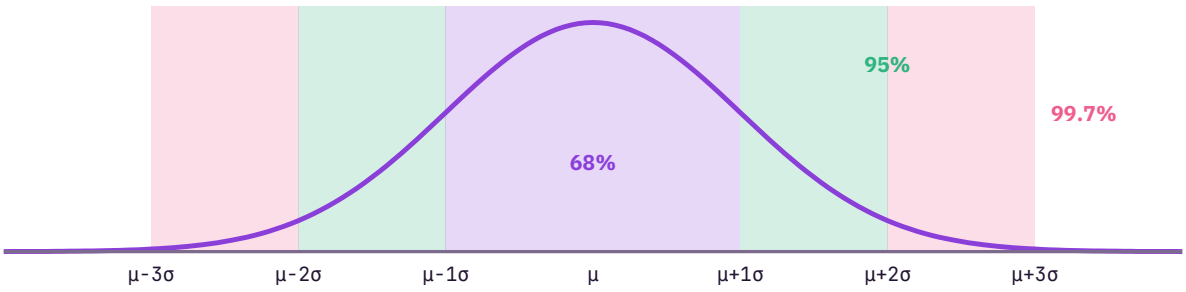
base rate sensitivity

false-positive rate



```
Of 1,000 people: 10 sick → 9 test positive (pink).  990 healthy → 50
test positive anyway (violet).
P(sick | positive) = 9 / (9 + 50) = 15.3%
```

Distributions you will meet



68–95–99.7. In a normal distribution, values beyond 3σ are rare — a simple rule for spotting outliers (Area 03) and anomalies (Area 06).

```python
import numpy as np

rng = np.random.default_rng(1)
flips = rng.binomial(n=1, p=0.3, size=10_000)        # Bernoulli: did the user c
heights = rng.normal(loc=170, scale=8, size=10_000)  # normal
print("click rate ≈", flips.mean())
print("mean height ≈", heights.mean().round(1), " std ≈", heights.std().round(1)
print("share within 1 std:", np.mean(np.abs(heights - 170) < 8).round(3))
```

```
# output
click rate ≈ 0.3039
mean height ≈ 169.9  std ≈ 7.9
share within 1 std: 0.69
```

Distribution	Models	NumPy	ML use
Bernoulli	one yes/no	`rng.binomial(1, p)`	the target of a binary classifier
Binomial	count of yeses in n	`rng.binomial(n, p)`	conversions per day
Categorical	one of k classes	`rng.choice(k, p=…)`	softmax outputs, next-token prediction
Normal	sums of many effects	`rng.normal(μ, σ)`	noise, weight initialisation
Uniform	anything equally likely	`rng.uniform(a, b)`	random search over hyperparameters
Poisson	counts of rare events	`rng.poisson(λ)`	visits, failures per hour

From scores to probabilities

Classifiers output a score z; the sigmoid squashes it into a probability between 0 and 1, and softmax does the same for many classes so the probabilities add to 1. The loss used to train them, cross-entropy, is the negative log of the probability given to the true answer — confident wrong answers are punished hardest.

```
import numpy as np

z = np.array([-2.0, 0.0, 3.0])
print("sigmoid:", (1 / (1 + np.exp(-z))).round(3))

scores = np.array([2.0, 1.0, 0.1])                    # 3 classes
p = np.exp(scores) / np.exp(scores).sum()
print("softmax:", p.round(3), "sum =", p.sum())
print("cross-entropy if class 0 is right:", round(-np.log(p[0]), 3))
print("cross-entropy if class 2 is right:", round(-np.log(p[2]), 3))
```

```
# output
sigmoid: [0.119 0.5   0.953]
softmax: [0.659 0.242 0.099] sum = 1.0
cross-entropy if class 0 is right: 0.417
cross-entropy if class 2 is right: 2.317
```

i

Probabilities are what you sell

A fraud model that says 0.97 is more useful than one that says “fraud”: the business picks the threshold. That is why calibration and thresholds get a whole module in Area 05.

×

Common mistakes

Confusing P(A | B) with P(B | A) · ignoring the base rate · assuming everything is normal (incomes, wait times and clicks are skewed) · treating a model score as a calibrated probability without checking.

INTERVIEW CORNER

Q. A test is 99% accurate and the disease affects 1 in 1,000. A positive result means?
A. Mostly a false positive: ~1 true positive per ~10 false positives, so P(sick | +) ≈ 9%.

Q. Why softmax? **A.** It turns any real-valued scores into positive probabilities that sum to 1, and it is differentiable.

Key takeaways

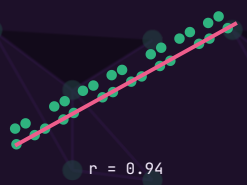
1. Conditional probability is directional — base rates matter.
2. Know Bernoulli, categorical and normal by heart.
3. Sigmoid/softmax make probabilities; cross-entropy scores them.

MODULE 4 OF 4

Statistics: mean, variance & sampling

Summarise data and know how much to trust a number.

- Centre and spread
- z-scores and correlation
- Sampling and the CLT
- Bootstrap confidence intervals



⏱ 3 hours | Area 02 · Math for ML

Start from zero

New to this? Work through these steps first, in order. Already comfortable? Skim them and jump to the reference below.

1

STEP 1

Centre and spread

≈ In plain words

The mean is the balance point; the median is the middle value; the standard deviation is the typical distance from the mean.

☀ Think of it like

Average salary vs median salary when one billionaire walks into the room.

✎ Worked example

```
import numpy as np
x = np.array([3, 4, 4, 5, 100])
print(x.mean(), np.median(x), x.std().round(1))
```

```
# output
23.2 4.0 38.4
```

Aa The word for it

Mean, **median**, **standard deviation**.

! **Why it matters** Skewed features (prices, durations) need the median and often a log transform.

2 STEP 2 Moving together isn't causing

≈ **In plain words** Correlation says two numbers rise and fall together. It doesn't say one causes the other.

✳ **Think of it like** Ice-cream sales and sunburns both rise in summer; ice cream doesn't cause sunburn.

✎ **Worked example**

pair	r	causal?
ad spend, sales (A/B tested)	0.9	yes (tested)
ice cream, sunburn	0.8	no (summer)

Aa **The word for it** **Correlation**; **confounder**.

! **Why it matters** Models find correlations; only experiments prove causes.

PAUSE & TRY **Customers who use feature X churn less (r = −0.6). Should you force everyone onto X?** Answer

Not yet: engaged customers may both use X and stay. Run an A/B test.

3 STEP 3 A number from a sample wobbles

≈ **In plain words** Measure a different sample and you'd get a slightly different answer. The wobble shrinks as samples grow.

Think of it like

Tasting soup: one spoonful can mislead; ten spoonfuls from different spots tell you more.

Worked example

test-set size	typical wobble (1 std error, accuracy ≈ 80%)
100	± 4%
1,000	± 1.3%
10,000	± 0.4%

The word for it

Sampling variability; a **confidence interval** reports it.

Why it matters

Before declaring a new model better, check the difference is bigger than the wobble.

Going further: the reference

The same ideas at the next level: precise definitions, tables, rules and interview questions.

Key definitions

Mean / median

The average, and the middle value; the median ignores extreme outliers.

Variance / std

Average squared distance from the mean, and its square root (same units as the data).

z-score

(x – mean) / std: how many standard deviations from the mean. Standardization uses it.

Correlation (r)

Strength of a linear relationship, from –1 to +1. Not causation.

Population / sample

Everyone you care about, and the subset you actually observed.

Sampling variability

Different samples give different statistics; that wobble is uncertainty.

Central limit theorem

Averages of many independent values are approximately normal, with spread σ/√n.

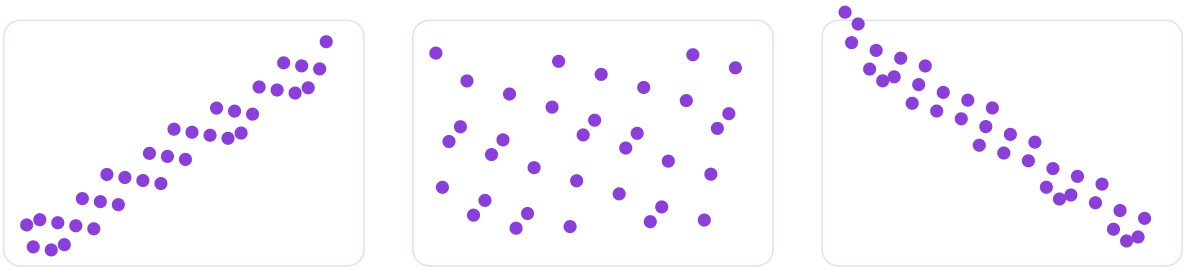
Confidence interval

A range that would contain the true value in, e.g., 95% of repeated samples.

```python
import numpy as np

wait_min = np.array([2, 3, 3, 4, 4, 5, 5, 6, 7, 48])   # one very slow day
print("mean   =", wait_min.mean())
print("median =", np.median(wait_min))
print("std    =", wait_min.std().round(2))
z = (wait_min - wait_min.mean()) / wait_min.std()
print("z-scores:", z.round(2))
```

```
# output
mean   = 8.7
median = 4.5
std    = 13.18
z-scores: [-0.51 -0.43 -0.43 -0.36 -0.36 -0.28 -0.28 -0.2  -0.13  2.98]
```



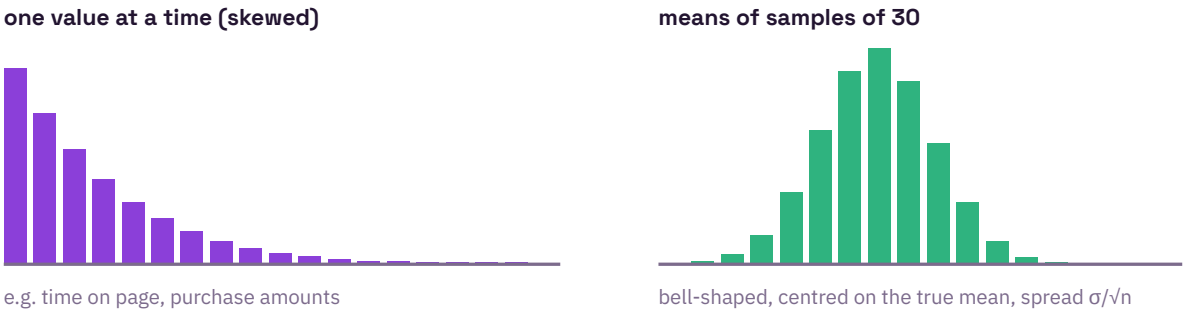
r ≈ +0.95 r ≈ 0 r ≈ -0.95

r only measures straight-line relationships. r ≈ 0 can still hide a strong curved relationship — always plot.

```python
import numpy as np

rng = np.random.default_rng(3)
ads = rng.uniform(0, 10, 200)
sales = 5 * ads + rng.normal(0, 8, 200)
icecream = rng.uniform(0, 10, 200)
print("corr(ads, sales)      =", np.corrcoef(ads, sales)[0, 1].round(2))
print("corr(ads, ice cream)  =", np.corrcoef(ads, icecream)[0, 1].round(2))
```

```
# output
corr(ads, sales)      = 0.86
corr(ads, ice cream)  = -0.0
```



Averages are better behaved than single values. That is why a test-set accuracy computed on 10,000 examples is trustworthy and one on 50 is not.

How sure are you? The bootstrap

A model's accuracy on a test set is itself an estimate from a sample. Resampling the test set with replacement many times shows how much it would wobble — a confidence interval without formulas.

```python
import numpy as np

rng = np.random.default_rng(0)
correct = rng.random(500) < 0.82            # 500 test predictions, ~82% right
acc = correct.mean()
boot = [rng.choice(correct, size=correct.size, replace=True).mean() for _ in ran
lo, hi = np.percentile(boot, [2.5, 97.5])
print(f"accuracy = {acc:.3f}   95% CI ≈ [{lo:.3f}, {hi:.3f}]")
```

```
# output
accuracy = 0.788   95% CI ≈ [0.752, 0.824]
```

★ **Report a range, not just a number**

“Model B is 0.6% better” means nothing if the 95% interval is ±3%. Bootstrap the test set before claiming a win — interviewers love this question.

The lab: gradient descent, by hand



Three independent ways to know the code is right: a hand-computed loss, numerical gradients and the exact least-squares answer.

× **Common mistakes**

Reporting the mean of skewed data without the median · reading causation into correlation · comparing models on tiny test sets · computing std with the wrong ddof without knowing (NumPy uses ddof=0, pandas ddof=1).

INTERVIEW CORNER

Q. Correlation vs causation? **A.** Correlation is co-movement; causation needs an intervention or a design (A/B test) that rules out confounders.

Q. Why standardize features? **A.** So features on different scales get comparable weights and gradient descent converges faster.

Key takeaways

1. Median for skewed data; std for spread; z-scores for scaling.
2. Correlation is linear and not causal.
3. Every metric is a sample estimate — bootstrap it.

PRACTICE

Check yourself

Answer in your head, then open each card. The area test has 20 more.



- [x] **[1, 2, 3] · [4, 0, −1]?**

  4 + 0 − 3 = 1.

- [x] **Shape of (32, 10) @ (10, 4)?**

  (32, 4).

- [x] **f(w) = w², what is f′(3)? Which way does gradient descent step from w = 3?**

  6; it steps toward smaller w (left).

Loss turns to NaN after a few steps. First thing to try?

Lower the learning rate (and check feature scaling).

Chain rule: L = (2w)², dL/dw?

dL/du · du/dw with u = 2w: 2u · 2 = 8w.

Rare disease 0.1%, test 99% sensitive, 1% false positives. Is a positive likely sick?

No — about 9%: ~1 true positive per ~10 false ones.

Softmax of [0, 0]?

[0.5, 0.5].

Accuracy 0.84 on 50 test examples vs 0.83 on 5,000. Which is more trustworthy?

The 5,000-example estimate — its sampling variability is about 10× smaller.

REFERENCE

Interview questions

Asked in ML engineer and data scientist interviews. Try an answer out loud before opening.



Explain gradient descent to a non-technical person.

Standing on a foggy hill, feel the slope underfoot and take a small step downhill; repeat until it is flat.

Batch vs stochastic gradient descent?

Batch uses all data per step (accurate, slow); stochastic/mini-batch uses a sample (noisy, fast, and the noise can help escape poor regions).

Why do we need non-linear activation functions?

Stacked linear layers collapse into one linear map; non-linearities let networks represent curved decision boundaries.

What is the vanishing gradient problem?

Multiplying many small derivatives (e.g. sigmoid's ≤ 0.25) through deep chains drives gradients to ~0, so early layers stop learning.

Bayes' theorem — give an ML example.

Naive Bayes spam filters: $P(\text{spam} \mid \text{words}) \propto P(\text{words} \mid \text{spam}) \cdot P(\text{spam})$.

Why cross-entropy instead of MSE for classification?

It is the negative log-likelihood of a categorical model and gives strong gradients for confident wrong answers.

How would you build a confidence interval for a model's accuracy?

Bootstrap the test set (resample with replacement many times) and take the 2.5th–97.5th percentiles.

What does cosine similarity measure and why use it for embeddings?

The angle between vectors; it ignores length, so documents of different sizes with similar meaning still match.

REFERENCE

Glossary

Every term used on this page, A to Z.

A a

Backpropagation
Chain rule applied backwards through a model.

Base rate
Prior probability of an event before evidence.

Bayes' rule
$P(A|B) = P(B|A)P(A)/P(B)$.

Bootstrap
Resampling with replacement to estimate uncertainty.

Central limit theorem
Averages tend to normal with spread $\sigma/\sqrt{n}$.

Chain rule
Derivative of a composition is a product of derivatives.

Confidence interval
Range covering the true value at a stated rate.

Correlation
Linear co-movement from −1 to +1.

Cosine similarity
Dot product of unit vectors.

Cross-entropy
−log of the probability given to the true class.

Derivative
Instantaneous rate of change.

Dot product
Sum of element-wise products.

Gradient
Vector of partial derivatives.

Gradient descent
Iterative minimisation by stepping against the gradient.

Learning rate
Step size in gradient descent.

Loss
Number measuring model error.

Matrix
2-D array of numbers.

Mean squared error
Average of squared errors.

Norm
Vector length.

Normal distribution
Bell curve $N(\mu, \sigma^2)$.

Partial derivative
Derivative w.r.t. one variable.

Sigmoid

$1/(1+e^{-z})$, maps scores to (0, 1).

Softmax

Exponentiate and normalise scores to probabilities.

Standard deviation

Square root of variance.

Transpose

Swap rows and columns.

z-score

Distance from the mean in standard deviations.

REFERENCE

FAQ & troubleshooting

The problems learners hit most in this area, and the fix for each.



You see	Why	Fix
ValueError: matmul: Input operand 1 has a mismatch	Inner dimensions differ	Print both shapes; transpose one (`.T`) if needed
Loss becomes inf or nan	Learning rate too high, or unscaled features	Lower η by 10×; standardize X
Loss barely changes	Learning rate too low, or gradient formula wrong	Raise η; compare to a numerical gradient
RuntimeWarning: overflow in exp	Large scores in softmax/sigmoid	Subtract the max score before exp (stable softmax)
Std differs between NumPy and pandas	ddof 0 vs 1	Pass `ddof=1` to NumPy or `ddof=0` to pandas
My bootstrap interval is huge	Small test set	Collect more evaluation data before comparing models

REFERENCE

Further reading

Go deeper when you are ready.



Resource	Why read it
Deisenroth, Faisal & Ong, *Mathematics for Machine Learning* (Cambridge, free PDF at mml-book.github.io)	Linear algebra, calculus and probability written for ML
3Blue1Brown, *Essence of Linear Algebra* and *Essence of Calculus* (YouTube)	Visual intuition for every idea in Modules 1–2
Goodfellow, Bengio & Courville, *Deep Learning*, ch. 2–4 (free at deeplearningbook.org)	The maths background chapters used by deep learning
Efron & Hastie, *Computer Age Statistical Inference* (free PDF)	The bootstrap and modern statistics, by the bootstrap's inventor

HANDS-ON LAB · ABOUT 2 HOURS

Lab 02 · Gradient descent from scratch

`gd.py` trains linear regression on 200 samples using only NumPy. The loss, the gradient and the update each hide one bug. `check.py` compares against a loss you can compute by hand, numerical gradients and the exact least-squares answer.

1. `pip install -r requirements.txt`, then `python check.py`.
2. Start with check 1: do the hand calculation in the message. What does `mse` forget?
3. Checks 2–3: which gradient disagrees with the numerical one, and by roughly what factor of n?
4. Checks 4–5: the loss explodes. Which way should the update step go?

5. Re-run after each fix until all five checks are ok.

Done when you see `PASS: all 5 checks passed — gradient descent recovers w=[ 1.99 -1.01 0.51], b=4.00`.