GO UNDERNEATH

The Maths You Actually Need

Eight ideas that carry almost all the weight in machine learning.

First edition, September 2026

Addaly

Series editor: Dr Mustafa Mun

Draft, open for academic review

GO UNDERNEATH

The Maths You Actually Need

Eight ideas that carry almost all the weight in machine learning.

Series editor: Dr Mustafa Mun

Addaly

First edition, September 2026 · Draft, open for academic review

The Maths You Actually Need

First edition, September 2026, build 62a26075.

Written by AI models to a written standard, with Dr Mustafa Mun as series editor. Exam keys checked blind by a second model: 3,665 of 3,666 agreed. Academic review invited.

The manuscript was drafted with the assistance of a large language model and was edited and published under his name. That is stated plainly here rather than left to be discovered, because a reader is entitled to know how a book they are trusting was made.

How to cite: *Mun, M. (2026). The Maths You Actually Need. Addaly. addaly.com/learn/maths-you-need.*

This book is the printed form of the course of the same name at addaly.com/learn/maths-you-need. The website is the living version and is corrected first. Where the two differ, the website is right.

Free to read, free to download, free to print, and free to teach from. There is no paid edition and there never will be.

Every diagram in it is drawn from data by code, not generated as a picture, so the numbers on the page are the numbers in the argument. The answers to every check, test and examination question are at the back.

8 modules · 76 lessons · 27 figures · 8 case studies · 62,929 words · about 11 hours

Brief contents

1

MODULE 1

Vectors, and the space they live in

Turn a real piece of data into a vector and defend the choices made, compute a length, a distance, a projection and a cosine by hand, and explain using the $1/\sqrt{d}$ rule why a similarity threshold that works in a 3-dimensional picture is meaningless in a 768-dimensional embedding space.

Lesson 1 · 9 min

Everything a model sees is numbers

THE ONE THING TO KEEP

A model never sees your data, only the numbers you chose to represent it with, and information discarded in that step cannot be recovered by any amount of model capacity.

The step nobody writes about

Every course starts at "the model learns from data". It skips the step before, which is the one that decides how well anything afterwards can possibly work. A neural network cannot see a photograph, a Tuesday, or the word *shukriya*. It sees a list of numbers, and somebody chose that list.

That choice is called the representation, and it is not neutral. If you throw information away here, no amount of training recovers it.

Four things, and how they become numbers

A photograph. A 224×224 colour image is a grid of 224 rows, 224 columns and 3 colour channels. That is 224 × 224 × 3 = 150,528 numbers, each usually between 0 and 255 before scaling. The famous MNIST digit images are 28 × 28 greyscale, so 784 numbers each. When you read that a vision model "takes 224×224 input", that is the whole content of the claim: it eats a list of 150,528 numbers.

A word. Text is split into tokens, each token has an integer id, and each id indexes a row of a big lookup table. In a model with a 50,000-token vocabulary and 768-dimensional embeddings, that table is 50,000 × 768 = 38.4 million numbers. The word itself has vanished by the time the first layer runs. What remains is row number 8,417, whatever that row contains.

A category. Suppose a column holds `north`, `south`, `east`, `west`. The obvious move is to write 1, 2, 3, 4. This is almost always wrong, because it tells the model that west is four times north and that south sits between north and

east. The standard fix is one-hot encoding: four columns, exactly one of them 1. Alternatively you learn an embedding, which is what a model does with words.

The same trap catches postcodes, phone area codes, and customer ids. A pincode of 400001 is a name written with digits, not a quantity. Feeding it as a number invites the model to conclude that 400001 and 400002 are nearly identical, which is true of Mumbai's Fort and Ballard Estate but false of the boundary between two states.

A time. Hours run 0 to 23, and then wrap. Encode the hour as a plain number and the model believes 23:00 and 00:00 are as far apart as two numbers can be within a day. The usual repair is two columns, `sin(2π·hour/24)` and `cos(2π·hour/24)`, which places the hours on a circle so that 23 and 0 sit next to each other. It is a small trick and it routinely improves a demand-forecasting model more than a bigger network does.

Figure 1.1

Two ways to write down eleven at night

Hour as one number, 0 to 23

- 23:00 and 00:00 are 23 apart, the largest gap the column allows
- Midnight looks as far from 11 p.m. as 11 p.m. is from midday
- The model learns a cliff at midnight that is not in the world
- Nothing warns you: the column is numeric and the fit converges

Hour as a point on a circle

- Two columns: sin of 2 pi h over 24, and cos of the same
- 23:00 and 00:00 land next to each other, as they should
- The distance between two hours is the distance round the clock
- One extra column, and no extra model capacity needed

The same fact, two encodings, and the model can only see the numbers. The circular version routinely improves a demand forecast more than a larger network does, and it costs two lines of pandas.

Scaling, and why it is not cosmetic

Put two columns side by side: annual income in rupees, ranging over hundreds of thousands, and number of children, ranging 0 to 5. Any method based on distance — nearest neighbours, k-means, anything with a gradient — is now driven almost entirely by income, because a difference of ₹50,000 dwarfs a difference of 2 children numerically. The maths has no idea these are different units.

Two standard repairs:

- **Min-max scaling** maps each column onto 0 to 1: `(x - min) / (max - min)`. Simple, and badly damaged by a single outlier.

- **Standardisation** subtracts the mean and divides by the standard deviation, so each column has mean 0 and spread 1. This is the default in most pipelines and the one to reach for first.

Both are computed on the training data and then applied unchanged to test data. Computing them over the whole dataset first is a genuine leak: your test numbers have quietly informed the scaling.

What you have already decided

By the time the first matrix multiplication happens, you have committed to several claims without stating them: that these columns matter and others do not, that this ordering is meaningful and that one is not, that a missing value should be filled with the median rather than flagged as missing. Each is a modelling decision made in the data-loading code, usually by whoever wrote it in a hurry.

The honest limitation is this: representation errors are invisible in the loss curve. A model fed pincodes as quantities will train perfectly happily, converge, and produce a respectable-looking number. It has learned a real pattern in a fictional space. Nothing in the training process can tell you that the space was fictional, because the training process only ever saw the numbers.

The free path

You do not need anything paid to do any of this. `pandas` and `NumPy` handle loading and scaling; `scikit-learn` has `StandardScaler`, `MinMaxScaler` and `OneHotEncoder`; all three are free and run on a laptop with no GPU. If you have no Python at all, GNU Octave does the array arithmetic and Google Colab gives you a free browser notebook. The maths in this course never needs hardware you have to buy.

BEFORE YOU MOVE ON

A team encodes a six-digit Indian pincode as a single numeric column and trains a delivery-time model. The model trains cleanly and reports a good validation score. What has most likely gone wrong?

A. The pincodes should have been scaled to 0-1 first, and the unscaled magnitude is dominating the gradient.

B. The model has learned real structure in an ordering that does not exist, and neither the loss curve nor the validation score can reveal that.

C. Nothing has gone wrong, because a neural network with enough capacity can learn any mapping from the numeric pincode.

D. The validation score is inflated because pincode leaks the answer from the training set.

The answer and why: p. 429

Lesson 2 · 7 min

A vector is just a list

THE ONE THING TO KEEP

A vector is an ordered list of numbers, and in a model its direction carries the meaning, not its length.

Start with a spreadsheet row

A flat in Bengaluru: rent 32000, area 850 square feet, floor 4, walking minutes to the metro 12. Write those numbers in a fixed order and you have a vector:

```
[32000, 850, 4, 12]
```

That is the whole definition. An ordered list of numbers. The order is the only rule — the third slot always means floor, in every flat you describe.

The word scares people because school introduced it as an arrow in physics, with force and velocity and diagrams. In machine learning, forget the arrow for a moment. It is a row.

What models actually store

When a model reads the word "chai", it does not store the letters. It looks up a list of numbers — 384 of them, or 768, or 1536, depending on the model. That list is called an embedding. Nobody sat down and chose those numbers. Training produced them.

Here is the honest part. Individual slots in that list almost never mean anything you could name. Dimension 412 is not "spiciness" or "how Indian the word is". The meaning is smeared across all 768 numbers at once, and any single one is close to unreadable. You may have heard that `king - man + woman` lands near `queen`. That is real for some older word embeddings, and much weaker and more cherry-picked than the story suggests. Enjoy it as a demo. Do not build on it.

What "direction" means

Two vectors point the same way when one is a multiple of the other. `[2, 1]` and `[6, 3]` have different sizes and identical direction.

In an embedding space, the direction is where the meaning lives. The length usually tracks something else entirely — how common the word is, how long the document was, how much text the encoder had to work with.

Make that concrete. A 400-word review of a restaurant in Lagos and a 40-word review saying the same thing produce vectors pointing in nearly the same direction, but the long one is a longer vector. Compare them by angle and they match, correctly. Compare them by raw size and the long one wins, for a reason that has nothing to do with what it says.

This is why almost every retrieval system normalises: divide every number in the vector by the vector's length, so each vector sits on a sphere of radius 1 and only direction is left.

```
import math

v = [3.0, 4.0]
length = math.sqrt(3*3 + 4*4)      # 5.0
unit = [x / length for x in v]     # [0.6, 0.8]
```

That `sqrt(sum of squares)` is Pythagoras, extended to as many slots as you like. It works identically for 768 numbers. Nothing new happens.

Adding and scaling

Add two vectors by adding matching slots. Scale by multiplying every slot. That is it.

This has a real use. Take every sentence embedding in a paragraph and average them, slot by slot, and you get a crude paragraph embedding. It is used in production more than anyone admits. It also throws away word order completely, so "the dog bit the man" and "the man bit the dog" land in the same place. Cheap, useful, and wrong in a specific way you should know about.

Why people say "space"

A list of 768 numbers is a point in 768-dimensional space. You cannot picture that, and you never need to. Everything you will do with it is one of three things: measure a distance, measure an angle, or add.

One warning where 2D intuition breaks. High-dimensional space is mostly empty, and points in it tend to be roughly the same distance from each other. So a rule like "call them similar if the distance is under 0.5" gets tuned on one embedding model and then quietly means nothing when you switch models. Angles survive that switch a little better than distances. Neither survives it completely.

BEFORE YOU MOVE ON

A retrieval system ranks results by the angle between vectors. You multiply every number in one stored embedding by 10 and re-run the same search.

A. It ranks higher for every query now, because a bigger vector scores higher.

B. Its rank is unchanged, because scaling stretches a vector without turning it.

C. It moves to a different region of the space, so it gets different neighbours.

D. It now reads as a stronger, more emphatic version of the same meaning.

The answer and why: p. 430

Lesson 3 · 7 min

What a dot product measures

THE ONE THING TO KEEP

A dot product scores how much two lists agree, but it also grows with their length, so normalise before you compare.

Multiply the matching slots, add them up

Two lists of the same length. Multiply slot 1 by slot 1, slot 2 by slot 2, and so on, then add every result into one number.

```
a = [2, 0, 3]
b = [1, 5, -1]
dot = 2*1 + 0*5 + 3*-1   # 2 + 0 - 3 = -1
```

One number out. That number is a score of how much the two lists agree.

CASE STUDY · MODULE 1

Six hundred circulars and a scheme code nobody could find

An illustrative case, built from documented patterns.

A district co-operative bank in Kolhapur, forty branches, building a search over its own loan circulars for the officers at the counter.

The bank had six hundred circulars going back eleven years. Every one of them told a loan officer what a scheme allowed, what it required and when it changed. They lived in a shared folder, named by date, and finding the right one took a phone call to head office. In the sowing season that call took two days.

Two people in the IT cell built a search. They split the circulars into about seven thousand passages, embedded each with a free 384-dimensional model that runs on a laptop, and stored the vectors. The arithmetic said the scale was never going to be the problem: seven thousand passages at 384 numbers of four bytes each is eleven megabytes, and a brute-force comparison against all of them takes under a millisecond. Nothing here needed a vector database, a server, or a subscription.

The first version worked well on questions and badly on facts. "What is the margin requirement for a dairy loan?" returned the right passage. "KCC-2019/07" returned nothing useful at all. The embedding model had never seen that string as a token, so it had no row to look up and no direction to point in; the circular that carried the code was ranked forty-first.

They also found their threshold was meaningless. They had copied a rule from a tutorial: accept a match above cosine 0.8. On their model almost nothing scored above 0.8, and the search returned an empty list for most queries. So they labelled two hundred query-passage pairs by hand, matching and not matching, and plotted the scores. Unrelated pairs averaged 0.61. Matching pairs averaged 0.78. The whole signal lived in a band about fifteen points wide, sitting well above zero, because the model pushed all its vectors into a narrow cone. The right cut-off for their model and their documents was 0.71, and it could not have been guessed.

That left the real decision. The dense search alone would ship in a week. It would answer questions about schemes and it would fail, quietly and confidently, on scheme codes, account formats, circular numbers and the surnames of the officers who signed them. A hybrid search, running a BM25 index alongside the embeddings and merging the two rankings by position, would take about three more weeks: building the index was an afternoon, but the merge, the evaluation set and the retraining of forty branch managers were not.

The cost of waiting was concrete. The sowing season started in six weeks and the counter queues would double. Every week without search was another week of two-day phone calls, and the branch managers had been promised something before the season, not after it.

The cost of shipping the dense version alone was harder to see and worse. A loan officer who searches a scheme code, gets a confident-looking passage from a superseded 2017 circular, and quotes it at the counter has been actively misled by the tool. Nothing in the interface would say the code had not matched; the score would look ordinary. The IT cell had seen that failure in testing and could not think of a way to warn a user about it, because a dense model does not know that it has never seen a token.

They put both options to the general manager with the numbers attached: eleven megabytes, one millisecond, a measured threshold of 0.71, and a list of twelve real queries from the branches on which the dense search returned the wrong circular.

YOUR ANSWERS

1. The dense search could not find "KCC-2019/07". Why does adding a keyword index fix that when a better embedding model would not?

2. Why was a threshold of 0.8, copied from a tutorial, the wrong number here, and what would make 0.71 wrong too?

3. The team merged the two rankings by position rather than by score. What goes wrong if you add a BM25 score to a cosine?

STOP HERE

Write your answers before you turn the page. How it played out starts on p. 50.

CASE STUDY · MODULE 1

How it played out

The general manager took the three-week delay. The team shipped hybrid search two weeks before the season, running BM25 and the embeddings together and merging by rank position rather than by score, because the two scores were on scales that could not be compared. The threshold went in as 0.71, with a note in the code saying it had been measured on two hundred labelled pairs against that specific model and had to be re-measured if the model ever changed. On the twelve failing queries, hybrid search returned the right circular for eleven; the twelfth was a code that had been typed wrongly in the original document. The team also added a line to the results page saying which half of the system had found each result, which turned out to be the feature the branch managers mentioned most.

The questions, discussed

1. The dense search could not find "KCC-2019/07". Why does adding a keyword index fix that when a better embedding model would not?

A dense model represents a string only if its tokeniser produced tokens it has learned rows for. A rare code is split into fragments the model has almost no signal about, so its direction in the space is close to noise, and no amount of model quality changes that for a string it has never usefully seen. A sparse index does not represent the code at all; it matches it. The two systems fail in complementary ways, which is the whole argument for running both.

2. Why was a threshold of 0.8, copied from a tutorial, the wrong number here, and what would make 0.71 wrong too?

Cosine scores are a property of one model's output distribution, not of meaning. This model was anisotropic: it packed its vectors into a narrow cone, so even unrelated passages averaged 0.61 and the useful signal sat in a band above that. A threshold has to be read off a labelled sample from the actual model. The 0.71 would become wrong the moment they changed embedding model, or re-chunked the documents, since both change the distribution the number was read from.

3. The team merged the two rankings by position rather than by score. What goes wrong if you add a BM25 score to a cosine?

The two numbers are on incompatible scales. A cosine is bounded between minus one and one and, on this model, clustered around 0.6; a BM25 score is unbounded and depends on corpus statistics and document length. Adding or averaging them lets whichever happens to have the larger numbers dominate the ranking for reasons unrelated to relevance. Merging by rank position discards the magnitudes and keeps only the ordering each system is entitled to assert.

MODULE 1 TEST

Check what stuck

6 questions, one right answer each. This one is for you, not for a record. The answers, and the reason behind each, are in the key on p. 428. If two go wrong, re-read the module before the exam.

1 A first search system ranks documents by the raw dot product between the query vector and each document vector, with no normalisation anywhere. Long pages keep coming top. What is the mechanism?

A. A dot product is both lengths times the cosine of the angle, so a long vector scores high against everything

B. Long documents contain more of the query's words, so they genuinely are more relevant

C. The embedding model produces higher-quality vectors for long documents than for short ones

D. Dot products are undefined between vectors of different lengths, so the library pads the shorter one

2 A team moves from a 384-dimensional embedding model to a 1,536-dimensional one and keeps its cosine threshold of 0.35. What does the one-over-root-d rule say about that threshold?

A. Nothing changes, because a cosine is already normalised and is comparable across models

B. It now demands a far more unusual match: random pairs spread about 0.026, not 0.051

C. It is now far looser, because more dimensions give two vectors more chances to agree

D. It should be doubled, because each vector now holds four times as many numbers

3 Two error vectors over four items: A is (10, 0, 0, 0) and B is (3, 3, 3, 3). Which norm calls A the larger, and why does the answer matter for a loss function?

A. L1 calls A larger, because it adds absolute values without squaring them

B. Both call A larger, since 10 is the biggest single number in either vector

C. L2 calls A larger, because squaring makes one big deviation count for more than four small ones

D. Neither: the two vectors have the same size under every Lp norm

4 A team finds a direction in embedding space that encodes an unwanted attribute and projects every vector onto the space perpendicular to it. A classifier trained on the result still predicts the attribute well above chance. What went wrong?

A. The projection formula was applied with the wrong sign, so the direction was doubled rather than removed

B. The attribute was never confined to one direction, and subtracting one direction removes exactly one

C. Projection works only on unit vectors, and theirs had not been normalised first

D. The classifier is overfitting, and would fall to chance on a larger test set

5 Why can you not average a 768-number embedding from one model with a 768-number embedding from another model trained with a different seed?

A. The two models were trained on different data, so the averaged vector would be biased towards the larger corpus

B. Averaging is defined only for vectors of the same length, and these two differ in scale

C. Most training objectives are unchanged by a rotation, so each model lands on arbitrary axes and slot 42 means different things

D. They would need to be concatenated first and the result halved

6 A support search must handle both the sentence "my order has not arrived" and the order code AX-99182. Why does a dense embedding model handle the first and fail the second?

A. Dense models cap the number of characters they can represent, and the code exceeds that cap

B. The code contains digits, and embedding models discard numeric tokens during training

C. The code is too short, and an embedding needs a full sentence to produce a usable vector

D. A dense vector records what text is about, and a rare code splits into fragments the model has almost no signal for

Lesson 10 · 8 min

Doing a matrix multiplication yourself, once

THE ONE THING TO KEEP

Every entry of a matrix product is one dot product between a row of the left matrix and a column of the right, which is why the inner dimensions must match and why order cannot be swapped.

Do it once and you never fear it again

Matrix multiplication looks like a rule to memorise. It is one sentence: **entry (i, j) of the answer is the dot product of row i of the left matrix with column j of the right matrix.** Everything else follows from that, including the shape rule and the fact that order matters.

Here is a full worked example. Take

```
A = [ 1  2  3 ]        B = [ 7   8 ]
    [ 4  5  6 ]            [ 9  10 ]
                           [ 11 12 ]
```

A is 2×3, B is 3×2. Work out each of the four answer entries.

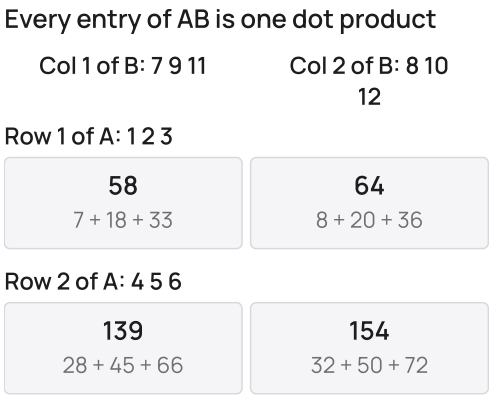
```
(1,1): row 1 of A . column 1 of B = 1*7 + 2*9 + 3*11 = 7 + 18 +
33  = 58
(1,2): row 1 of A . column 2 of B = 1*8 + 2*10 + 3*12 = 8 + 20 +
36 = 64
(2,1): row 2 of A . column 1 of B = 4*7 + 5*9 + 6*11 = 28 + 45 +
66 = 139
(2,2): row 2 of A . column 2 of B = 4*8 + 5*10 + 6*12 = 32 + 50
+ 72 = 154
```

So

```
AB = [ 58   64 ]
     [ 139 154 ]
```

Do that once, slowly, with a pen. It takes four minutes and it converts a rule into a thing you understand.

Figure 2.1



Four entries, four dot products of length three. The inner dimension has to be 3 on both sides because a dot product needs two lists of equal length, and that is the whole explanation for the shape error you will meet most often in your first month.

The shape rule, and why it is what it is

An (m × n) matrix times an (n × p) matrix gives an (m × p) matrix. Write the shapes next to each other:

```
(2 x 3) (3 x 2)  ->  (2 x 2)
     ^---^  the inner numbers must match; they vanish
   ^---------^     the outer numbers survive
```

The inner numbers must match because each answer entry is a dot product, and a dot product needs two lists of equal length. Row i of A has n entries; column j of B has n entries. If they differ, there is nothing to compute. That is the whole explanation for the error message you will see most often in your first month.

EXAMINATION

The Maths You Actually Need — final examination

This paper covers the whole course:

- vectors and the space they live in
- matrices and what a layer does
- derivatives and gradient descent
- probability and distributions
- logarithms and where losses come from
- estimation from samples
- the arithmetic of cost and scale
- how numbers behave inside a machine

56 questions, the whole pool. The site draws 25 for a sitting, pass mark 70%. Work any 25 under your own conditions. Answers from page 478.

1 Module 1 · Vectors, and the space they live in

Every vector in an index has been normalised to length one. A team benchmarks ranking by cosine against ranking by Euclidean distance. What will they find?

A. Cosine ranks better, because Euclidean distance ignores direction
B. Euclidean ranks better on long documents, where cosine saturates
C. The two orders agree only in fewer than about a hundred dimensions
D. The two produce the identical ordering, since squared distance is two minus twice the cosine

2 Module 1 · Vectors, and the space they live in

What is an embedding lookup, in terms of the matrix arithmetic it replaces?

A. A one-hot vector multiplied by the embedding matrix, optimised into an array index
B. A dot product between the token's vector and every row of the table
C. A projection of the token onto the first 768 principal components of the corpus
D. A hash of the token's characters, rescaled to the width of the model

3 Module 1 · Vectors, and the space they live in

On one embedding model, unrelated sentence pairs average a cosine of 0.6 and matching pairs average 0.78. What follows for the threshold you should use?

A. The model is broken, since unrelated pairs should sit near zero
B. Use 0.5, the midpoint of the range a cosine can take
C. The signal is the variation around 0.6, so the cut-off must be read off labelled pairs from this model
D. Subtract 0.6 from every score and then apply the usual 0.8 threshold

4 Module 1 · Vectors, and the space they live in

Why must the mean and spread used to standardise features be computed on the training rows alone?

A. Because test data usually has a different distribution, which would bias the scaling

B. Because computing them over everything lets the test rows inform the model, which is a leak

C. Because the test set is smaller and would dominate the averages

D. Because standardisation is defined only for data the model has already seen

5 Module 1 · Vectors, and the space they live in

If distances concentrate in high dimensions, why does embedding search return sensible results every day?

A. Because search libraries correct for concentration with a scaling factor

B. Because concentration only sets in above about ten thousand dimensions

C. Because trained embeddings occupy a low-dimensional structure inside the big space rather than filling it

D. Because cosine is immune to concentration while Euclidean distance is not

6 Module 1 · Vectors, and the space they live in

Why is BM25 hard to beat on queries containing a part number or an error code?

A. It scores rare exact terms highly and matches the string rather than representing it

B. It was tuned on technical documentation, unlike general embedding models

C. It converts every query into a dense vector before matching

D. It assigns common words a high inverse document frequency

Module 1 · Vectors, and the space they live in

The module starts on p. 11.

MODULE 1 TEST

Module 1 test, Q1, p. 52
A first search system ranks documents by the raw dot product between the query vector and each document vector, with no normalisation anywhere. Long pages…
Answer A: A dot product is both lengths times the cosine of the angle, so a long vector scores high against everything
Why: The dot product equals the length of one vector times the length of the other times the cosine of the angle, so it mixes agreement with size. A six-thousand-word page has a long vector and scores high against every query, whatever it says. Dividing by both lengths cancels the magnitudes and leaves the cosine, which depends on angle alone.

Module 1 test, Q2, p. 52
A team moves from a 384-dimensional embedding model to a 1,536-dimensional one and keeps its cosine threshold of 0.35. What does the one-over-root-d rule say…
Answer B: It now demands a far more unusual match: random pairs spread about 0.026, not 0.051
Why: The spread of the cosine between two random vectors is about one over the square root of the dimension: 0.051 at 384 dimensions and 0.026 at 1,536. A cosine of 0.35 is therefore about seven spreads out in the first space and about thirteen in the second, so the same number is a much stricter demand. The rule gives the direction; the threshold itself still has to be measured on labelled pairs from the model you are actually using.

Module 1 test, Q3, p. 53
Two error vectors over four items: A is (10, 0, 0, 0) and B is (3, 3, 3, 3). Which norm calls A the larger…
Answer C: L2 calls A larger, because squaring makes one big deviation count for more than four small ones
Why: L2 of A is the square root of 100, which is 10, and L2 of B is the square root of 36, which is 6, so L2 calls A larger. L1 gives 10 against 12 and calls B larger. Squaring says one deviation of ten is worse than four of three, which is exactly why squared-error loss is dragged around by a single wild data point and absolute-error loss mostly ignores it.

Module 1 test, Q4, p. 53
A team finds a direction in embedding space that encodes an unwanted attribute and projects every vector onto the space perpendicular to it. A classifier…
Answer B: The attribute was never confined to one direction, and subtracting one direction removes exactly one
Why: Projection removes precisely the component along the direction you named, and afterwards every vector's dot product with that direction is zero. If the attribute is spread across a subspace of many correlated directions, or is encoded non-linearly, everything outside that one direction survives untouched. The arithmetic did what was asked; the ask was too small.

Module 1 test, Q5, p. 53

Why can you not average a 768-number embedding from one model with a 768-number embedding from another model trained with a different seed?

Answer C: Most training objectives are unchanged by a rotation, so each model lands on arbitrary axes and slot 42 means different things

Why: Rotate every embedding by an orthogonal matrix and rotate the next layer's weights back by its inverse, and every dot product, distance and output is identical, so the loss cannot prefer one rotation over another. Training therefore lands on an arbitrary basis, different for every seed. Coordinates are relative to a basis; distances and neighbourhoods are not, so build what you rely on out of the second kind.

Module 1 test, Q6, p. 54

A support search must handle both the sentence "my order has not arrived" and the order code AX-99182. Why does a dense embedding model handle…

Answer D: A dense vector records what text is about, and a rare code splits into fragments the model has almost no signal for

Why: An embedding is a lookup of learned rows. A code the tokeniser has to split into fragments it has seen almost nothing about produces a direction close to noise, whatever the model's quality. A sparse index does not represent the code at all: it matches it. The two systems fail in complementary ways, which is the argument for running both and merging their rankings by position.

THE LESSON CHECKS

Lesson 1, p. 16

A team encodes a six-digit Indian pincode as a single numeric column and trains a delivery-time model. The model trains cleanly and reports a good…

Answer B: The model has learned real structure in an ordering that does not exist, and neither the loss curve nor the validation score can reveal that.

Why: A model never sees your data, only the numbers you chose to represent it with, and information discarded in that step cannot be recovered by any amount of model capacity.

A The pincodes should have been scaled…: Identifies a real problem with unscaled columns, but scaling a meaningless ordering just produces a meaningless ordering between 0 and 1.

C Nothing has gone wrong, because a…: Universal approximation says a network can fit a function of the input it is given; it does not conjure back the adjacency information the encoding destroyed.

D The validation score is inflated because…: Describes leakage, which is a different failure; here the encoding is wrong even with a clean split.

Lesson 2, p. 19

A retrieval system ranks results by the angle between vectors. You multiply every number in one stored embedding by 10 and re-run the same search.

Answer B: Its rank is unchanged, because scaling stretches a vector without turning it.

Why: A vector is an ordered list of numbers, and in a model its direction carries the meaning, not its length.

A It ranks higher for every query…: Treats similarity as growing with size, when the ranking only ever looked at angle.

C It moves to a different region…: Assumes a vector's position is what matters, when only its direction from the origin does.

D It now reads as a stronger…: Reads length as intensity, so a longer vector must mean the thing more strongly.

Lesson 3, p. 23

A search over unnormalised embeddings keeps putting very long documents at the top, no matter what the query is. What is going on?

Answer C: A dot product grows with vector length, and long documents produce longer vectors, so size is outvoting angle.

Why: A dot product scores how much two lists agree, but it also grows with their length, so normalise before you compare.

A Long documents contain more of the…: Assumes the score reflects real relevance rather than an artefact of vector length.

B A dot product counts how many…: Thinks a dot product is term overlap, so more words mechanically means a higher score.

D The embedding model was trained mostly…: Blames the model when the scoring function, not the model, is doing the damage.

Lesson 4, p. 26

A search index stores 200,000 product-description embeddings, all normalised to unit length on insert. A developer adds a new query path that sends raw, un-normalised…

Answer D: Ranking within one query survives, but any fixed similarity threshold or cross-query score comparison becomes meaningless, and no error is ever raised.

Why: A norm turns a whole vector into one number measuring size, and which norm you pick decides whether outliers dominate your answer or barely register.

A Every score comes back as exactly…: Confuses normalisation with dimensionality; the arithmetic still runs, which is precisely why the bug survives.

B Results are unaffected, because scaling the…: This is actually right for a single query, and the trap is that it feels wrong; the real damage appears when scores are compared across queries or against a fixed threshold.

C Nothing detectable, and the index silently…: Multiplying by a positive constant cannot reverse an order; nothing here flips signs.

GO UNDERNEATH

The Maths You Actually Need

Eight ideas that carry almost all the weight in machine learning.

WHAT IT TEACHES	Most people who avoid machine learning maths were failed by a teacher, not by their own brain. This course teaches the genuine minimum: vectors, dot products, matrices, derivatives, probability, distributions, expectation, and logarithms.
WHAT IS INSIDE	Eight modules and 76 lessons, 27 figures drawn from data, eight case studies, eight module tests, a 56-question examination, and every answer with the reason behind it.
LICENCE	CC BY-NC-SA 4.0. Copy, print, share and adapt it for any non-commercial purpose, with credit, under the same licence. There is no paid edition.
THE COURSE	Free to read, with the lessons, the films and the examination, at **addaly.com/learn/maths-you-need**

Scan to open the course

Addaly

First edition, September 2026 · build 62a26075