

Building an LLM from Zero

Chapter 3: Tensors and PyTorch

Module 1: Foundations

Truong (Jack) Luu · jackluu.io/book

Where we are

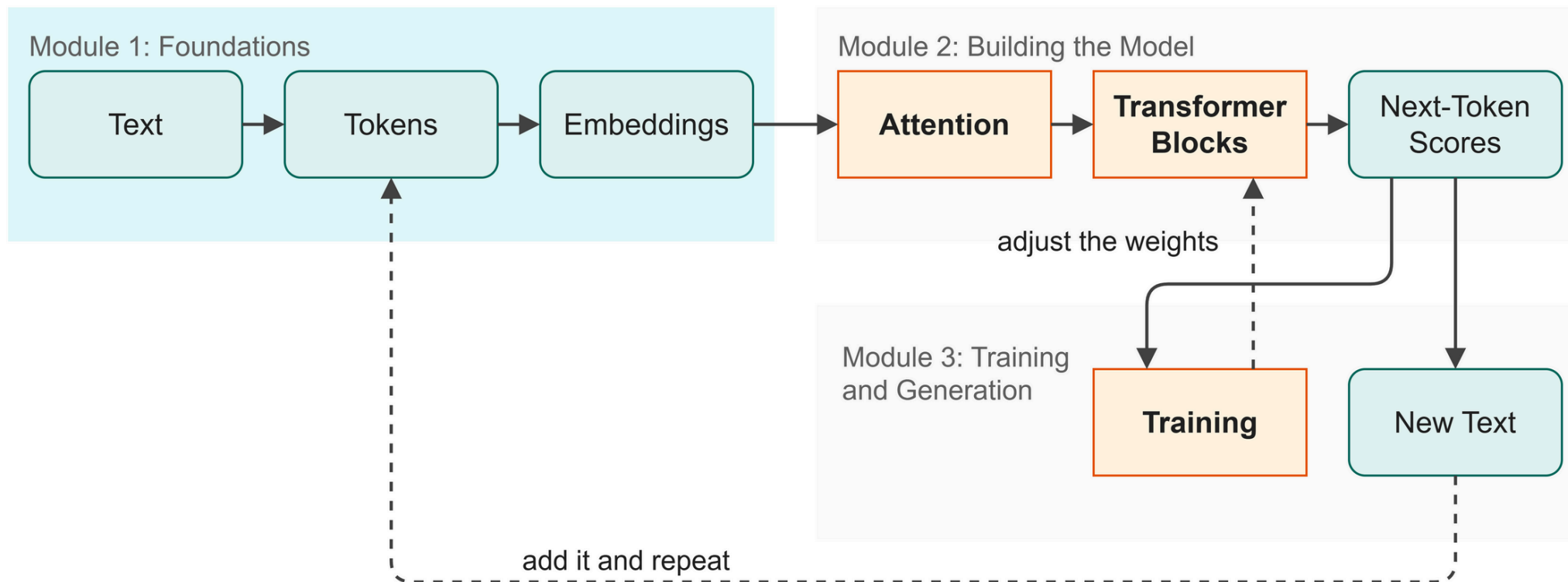


Figure 3.1: Tensors are the containers that carry data through every stage.

Every arrow on this map carries a tensor. Before we build any stage, we learn the container and the engine that moves it.

What this builds on (60-second recap)

- **Chapter 2:** a language model does one thing, **predict the next token** (one small piece of text)
- Generation is **autoregressive**: each prediction is appended and the model predicts again
- The text is its own answer key, so no human labels are needed

Next: the model only does math. What does it do the math on?

In this chapter you will

1. Learn what a **tensor** is and how it stores data
2. See how tensor **shapes** represent different dimensions
3. Understand how **matrix multiplication** transforms data
4. Turn raw scores into probabilities using **softmax**

Words to know (each is glossed again where it first appears):

- **PyTorch**: a library for doing math on large arrays of numbers very quickly
- **Tensor**: a multi-dimensional array of numbers
- **Shape**: the dimensions of a tensor (like rows and columns)
- **Matrix multiplication**: combining two tensors to transform data
- **Softmax**: a function that turns any numbers into probabilities that sum to 1

Why PyTorch

PyTorch is a Python library for doing math on large arrays of numbers, really fast.

- Pure Python is too slow for millions of calculations
- Without PyTorch, training would take **weeks instead of minutes**
- Everything in this book is ordinary Python calling PyTorch

PyTorch is the engine. Tensors are what it moves.

A tensor is a container of numbers

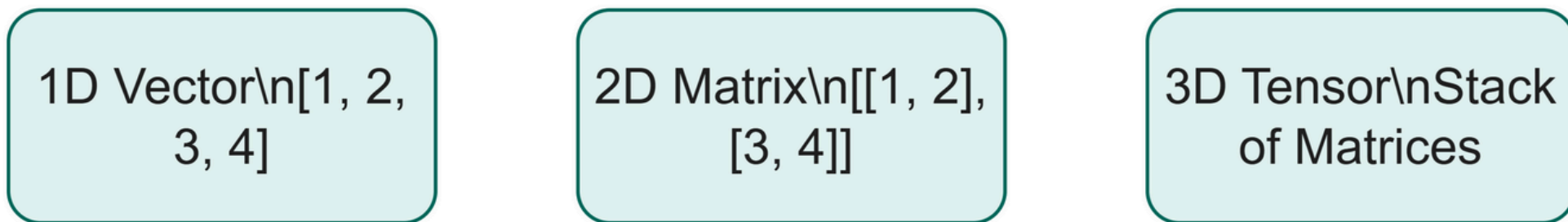


Figure 3.2: Tensors can be 1D (a list), 2D (a table), or 3D (a stack of tables).

A **tensor** (a multi-dimensional array of numbers) comes in as many dimensions as you need: a list, a table, a stack of tables.

If you have used a spreadsheet, you already know 2D tensors. This book works mostly in 3D: a stack of spreadsheets.

Shapes: (B, T, C)

The **shape** (the size of each dimension) of our 3D tensors is written (B, T, C) :

- **B = batch** size: how many sequences we process at once
- **T = time**: how many tokens in each sequence
- **C = channel** size: how many numbers describe each token

Example: (32, 128, 128) means 32 sequences, each 128 tokens long, each token described by 128 numbers.

Read every shape in this book as "how many sequences, how many tokens, how many numbers per token".

Matrix multiplication transforms data

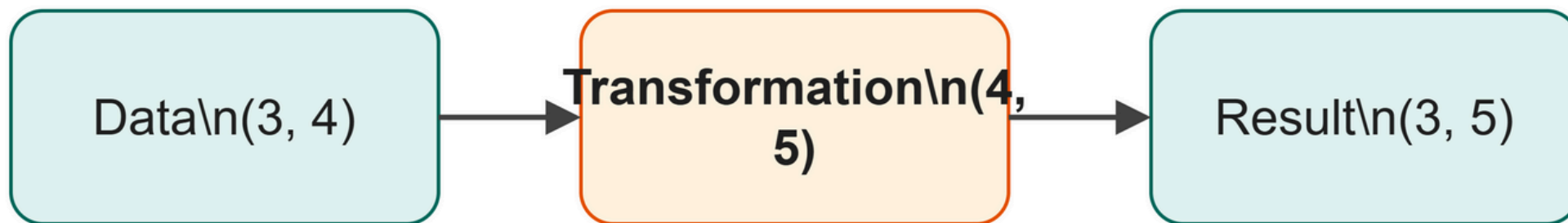


Figure 3.3: Matrix multiplication transforms data from one shape to another.

Matrix multiplication (written `@` in Python) is how neural networks transform data.

The rule: the **inner dimensions must match**. A `(3, 4)` tensor times a `(4, 5)` tensor gives `(3, 5)`.

The 4 in the middle disappears. The result keeps the outer sizes: 3 rows in, 5 columns out.

Softmax turns scores into probabilities



Figure 3.4: Softmax forces numbers into a 0-to-1 range where they total exactly 1.0.

Softmax takes any list of numbers (called **logits**, raw scores) and turns them into probabilities: all positive, summing to exactly 1.0.

It uses exponentials, so the highest score stands out even more: `1.0, 2.0, 3.0` becomes `0.09, 0.24, 0.67`.

Softmax is how the model's raw scores become "how likely is each next character".

Code: basic operations

File: `src/ch02_tensors.py` (source files are numbered one lower than chapters).

```
# A 1D tensor is simply a list of numbers
a = torch.tensor([1.0, 2.0, 3.0, 4.0])

# A 2D tensor represents a table or matrix of numbers
b = torch.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])

x = torch.randn(2, 5, 8)
W = torch.randn(8, 4)
y = x @ W
```

- Lines 2 and 5 create a 1D and a 2D tensor
- Line 7 creates a 3D tensor of random numbers with shape `(2, 5, 8)`: `(B, T, C)`
- Line 9 multiplies the 3D tensor by a 2D matrix with `@`

Run it

```
$ python src/ch02_tensors.py
--- 1. Creating tensors ---
1D tensor: tensor([1., 2., 3., 4.])
  shape: torch.Size([4])
...
```

What just happened

- Lines 2 and 5 created 1D and 2D tensors and printed their shapes
- Line 7 looked at a 3D tensor with shape `(2, 5, 8)`, meaning `(Batch, Time, Channels)`
- Line 9 multiplied a 3D tensor by a 2D matrix. PyTorch applied the multiplication across the whole batch by itself: **batched matrix multiplication**, no slow Python loops

The random values differ on your machine. The shapes do not.

Shape check

Tensor	Shape	Meaning
x	(2, 5, 8)	2 sequences, 5 tokens each, 8 numbers per token
W	(8, 4)	Transformation weights: 8 inputs to 4 outputs
y	(2, 5, 4)	2 sequences, 5 tokens each, now 4 numbers per token

- The inner 8s match, so the multiplication is allowed
- The last dimension changed from 8 to 4: that is the transformation

Every chapter from here ends its code with a shape check. Get into the habit now.

Try it: make softmax certain

1. Open `src/ch02_tensors.py`
2. Change `logits = torch.tensor([1.0, 2.0, 3.0])` to `[1.0, 2.0, 10.0]`
3. Run the script again

What the chapter says you will see: the highest number takes **almost 100%** of the probability after softmax.

A big gap in the scores becomes an even bigger gap in the probabilities. Remember this in Chapter 16, where we control that gap on purpose.

In business: how data is represented matters

For the house-style email assistant:

- The archive of company text is turned into **multi-dimensional tensors**
- The operations from today, multiplication and softmax, are exactly how the assistant processes that data to find patterns in the company's writing voice

There is no separate "business version" of the math. This is the math.

Watch out: the shape mismatch

A **shape mismatch** is the most common error in PyTorch.

- Try to multiply `(3, 4)` by `(5, 6)` and PyTorch crashes: the inner dimensions, 4 and 5, do not match
- The error message names both shapes; read it before you change anything

Always check your shapes. Most bugs in this book are shape bugs.

Discussion

Where is the "meaning" in a tensor of numbers?

In pairs, 3 minutes:

1. A `(32, 128, 128)` tensor holds 32 sequences of 128 tokens, 128 numbers each. What could one of those 128 numbers mean?
2. Softmax turned `1.0, 2.0, 3.0` into `0.09, 0.24, 0.67`. Why is it useful that the outputs sum to 1?

Take two answers from the room.

Recap: today in 5 points

1. A **tensor** is a multi-dimensional array of numbers.
2. Shape `(B, T, C)` = batch × time × channels, the standard in this book.
3. `@` is matrix multiplication. **Inner dimensions must match.**
4. **Softmax** turns any numbers into probabilities that sum to 1.
5. PyTorch handles batches automatically with batched matrix multiplication. No loops.

Tensors can hold our data. Next we turn text into the numbers that go inside them.

Check your understanding

1. What does the shape `(32, 128, 128)` represent in our `(B, T, C)` format?
2. Why do the inner dimensions need to match in matrix multiplication?
3. What is the difference between logits and probabilities?

Next up: Chapter 4, Tokenization

- Neural networks only eat numbers, so we build a **tokenizer** that turns text into them
- You will see all 65 characters of the vocabulary on one line, and encode a phrase and get it back

Before next class: run `python src/ch02_tensors.py` and write down the three shapes from the shape check in your own words.

References

Book and figures

- Luu, T. (2026). *Building an LLM from Zero*, Chapter 3. <https://jackluu.io/book/>
- Figures 3.1 to 3.4 are the book's own diagrams (draw.io sources in the book repository, `diagrams/`).
- Terminal output: `src/ch02_tensors.py` , run on the author's laptop.

Further reading (from the chapter)

- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). *PyTorch: An imperative style, high-performance deep learning library* (arXiv:1912.01703). arXiv.

Dataset

- Tiny Shakespeare, from Andrej Karpathy's char-rnn project: <https://github.com/karpathy/char-rnn>