
LECTURE 2-1

Transformer Review

Part 1: From Words to Contextual Vectors



Two Lectures, One Step at a Time

Each step answers a question raised by the step before it.

Part 1 (this week)

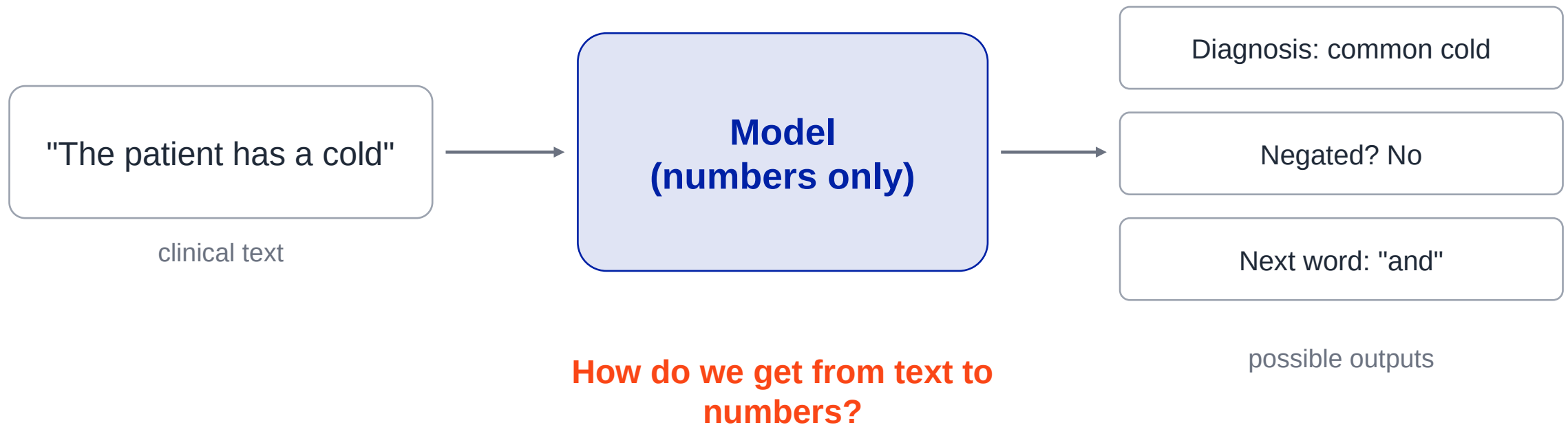
- 1 Tokens and embeddings
- 2 The problem: context
- 3 Attention, step by step
- 4 Queries, keys, and values
- 5 Building the encoder block

Part 2 (next week)

- 6 Predicting the next token
- 7 Masking and the decoder
- 8 Encoder-decoder models
- 9 Model families
- 10 Beyond text: images and other data

The Goal: Text In, Prediction Out

A neural network computes with numbers. The first job is to turn text into vectors.

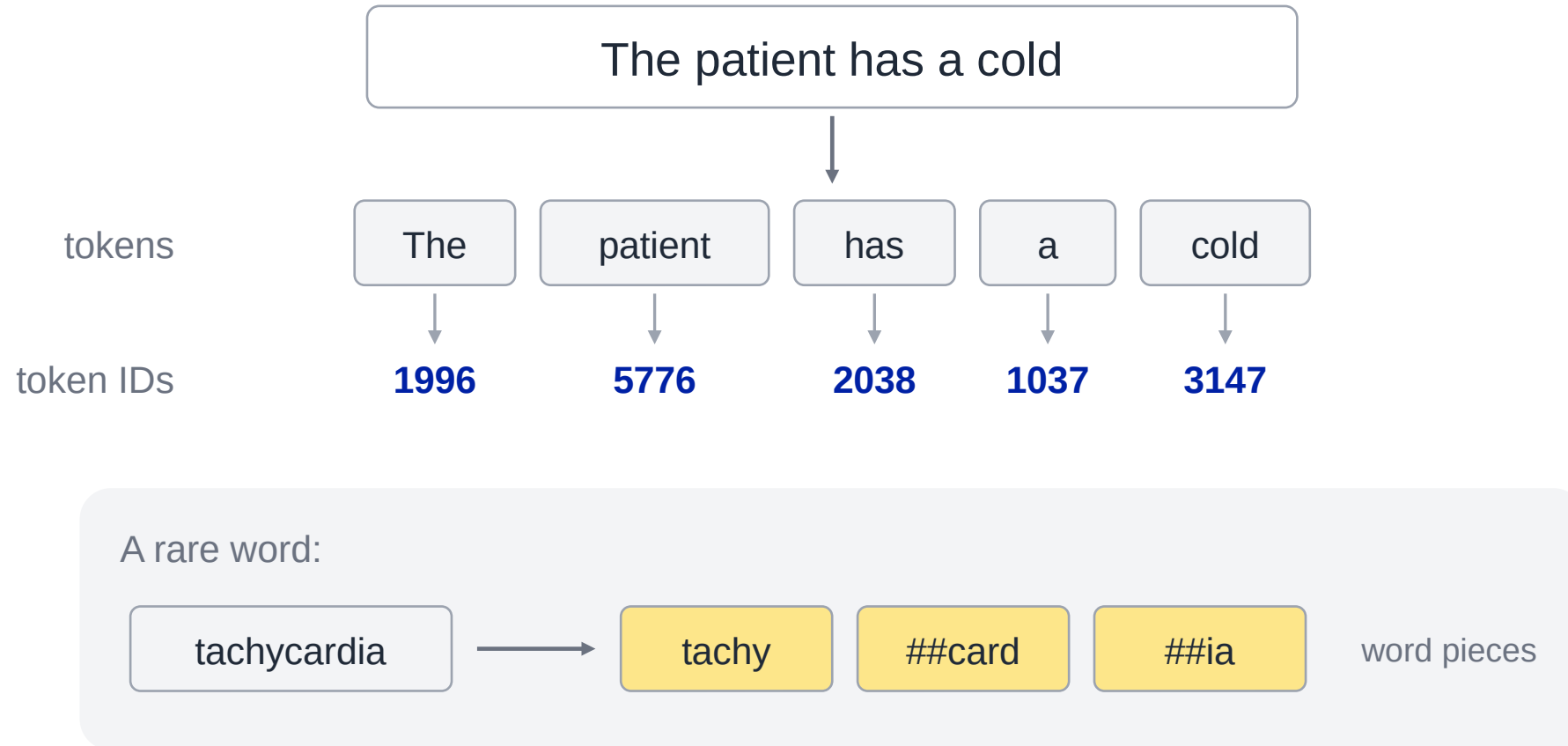




Tokens and Embeddings

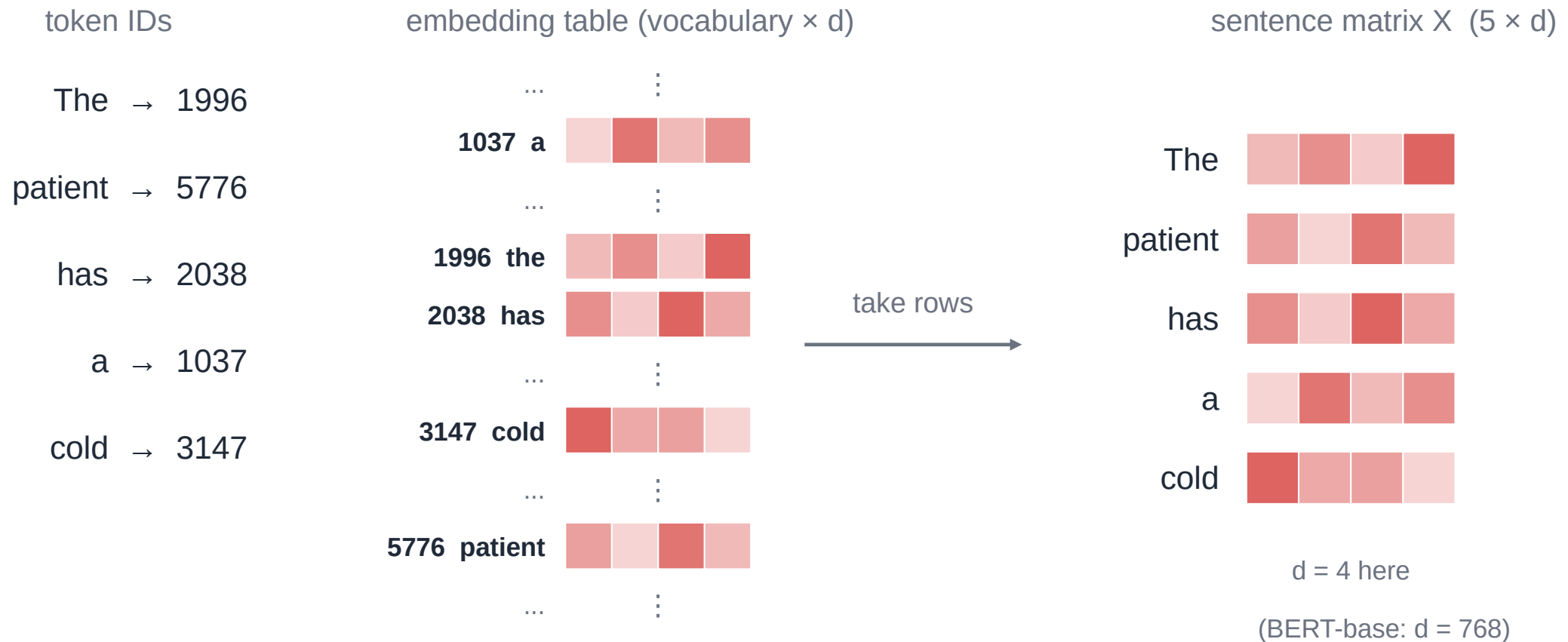
Step 1: Split the Text into Tokens

A tokenizer cuts text into pieces from a fixed vocabulary and gives each piece an integer ID.



Step 2: Look Up a Vector for Each Token

The embedding table holds one learned vector per vocabulary entry. Our 5 tokens become a $5 \times d$ matrix.



Embeddings Place Similar Words Close Together

Training moves the vectors so that words used in similar ways end up near each other.

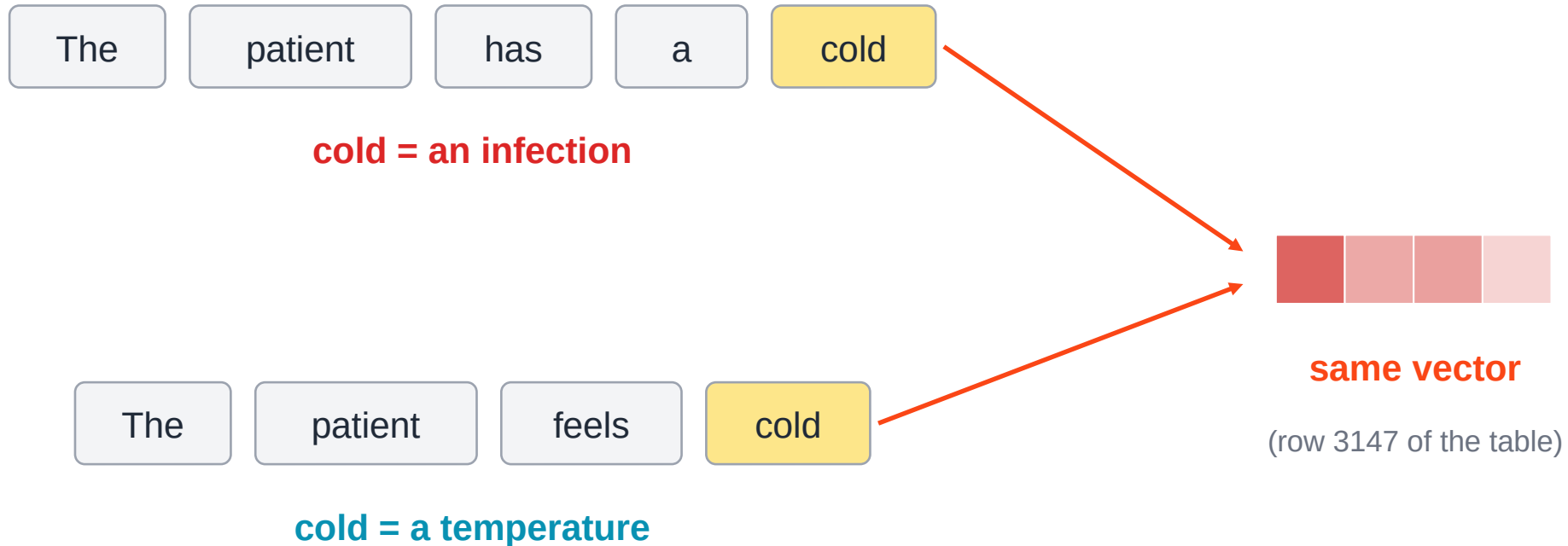




The Problem: Context

One Word, Two Meanings

The lookup gives 'cold' the same vector in both sentences. Its meaning depends on its neighbors.



Animation: From Tokens to Vectors

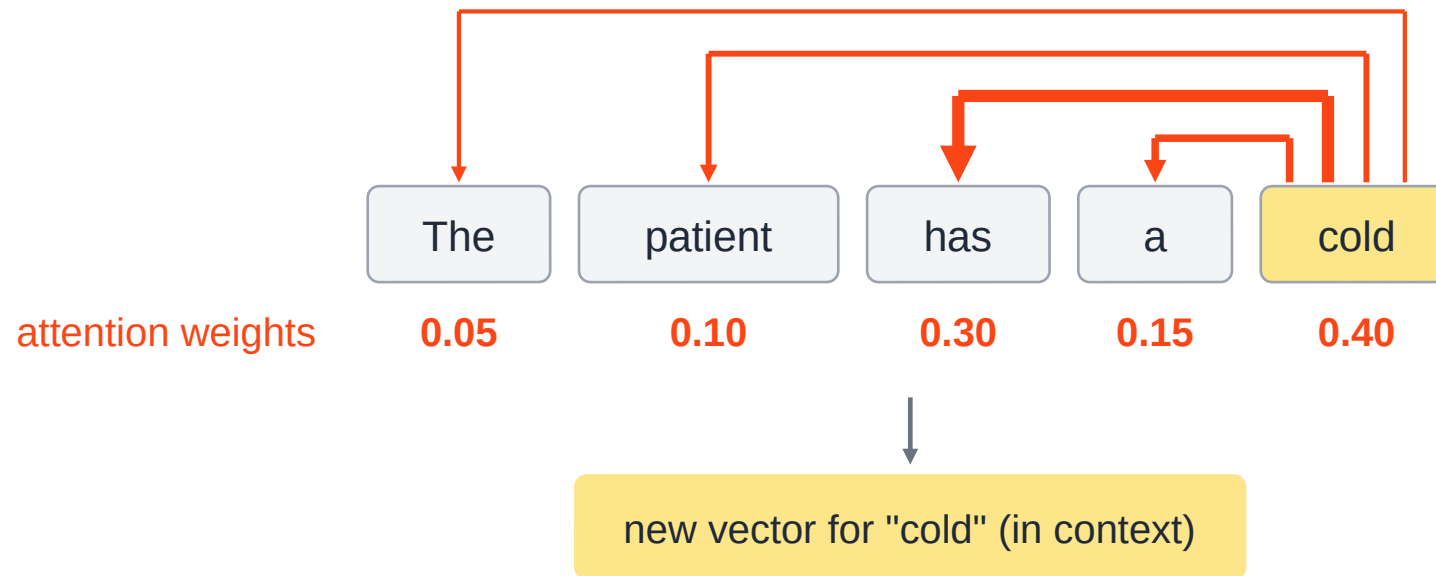
Animation clip: week02-clip1-tokens-to-vectors.mp4

Silent version embedded (click to play). The narrated version is in the course repository.

The Fix: Let Each Word Look at Its Neighbors

Update each token's vector using the other tokens in the sentence. The result is a contextual embedding.

"Which words help me understand what I mean here?"

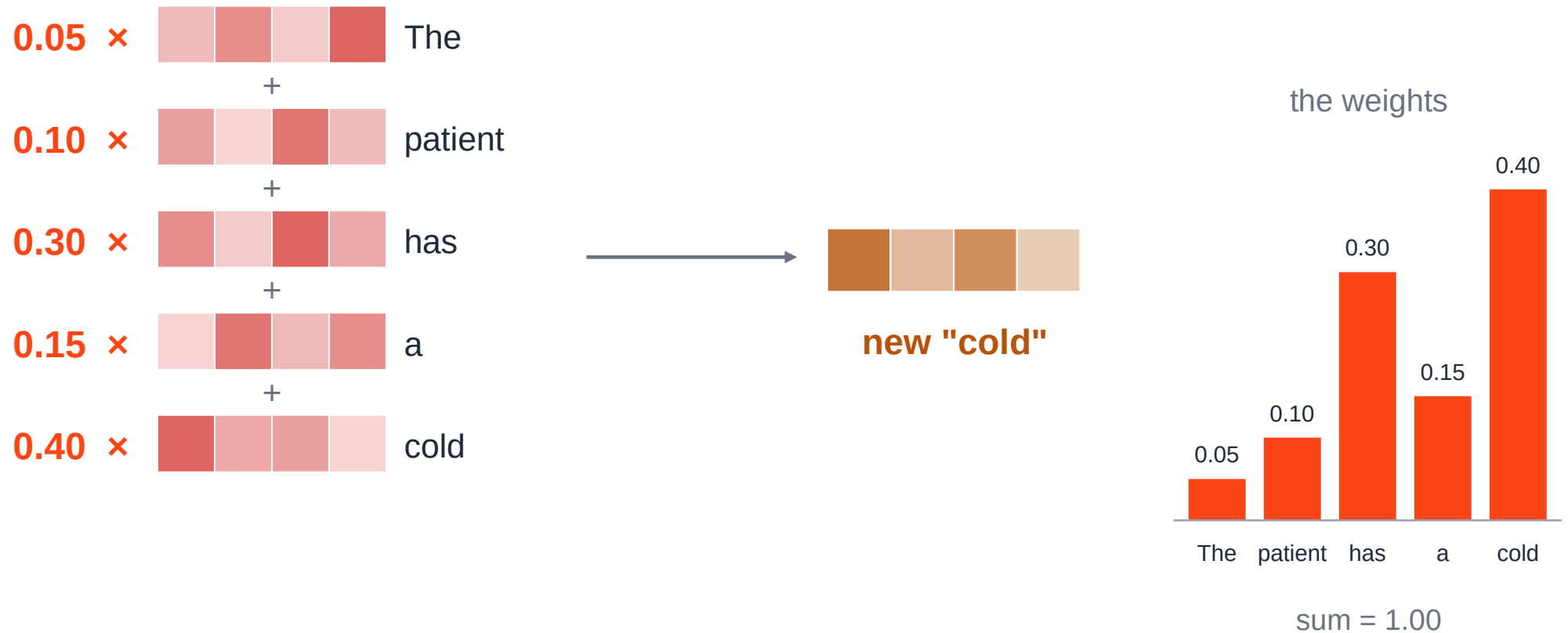




Attention, Step by Step

Attention Is a Weighted Average

The new vector for 'cold' is a weighted average of all the token vectors. The weights sum to 1.



Where Do the Weights Come From? Similarity

The dot product of two vectors is large when they point the same way. We use it as a similarity score.

cold

1	0	2
---	---	---

 $\cdot$ **has**

2	1	1
---	---	---

 $= (1)(2) + (0)(1) + (2)(1)$ **= 4**

similar direction: high score

cold

1	0	2
---	---	---

 $\cdot$ **The**

-1	2	0
----	---	---

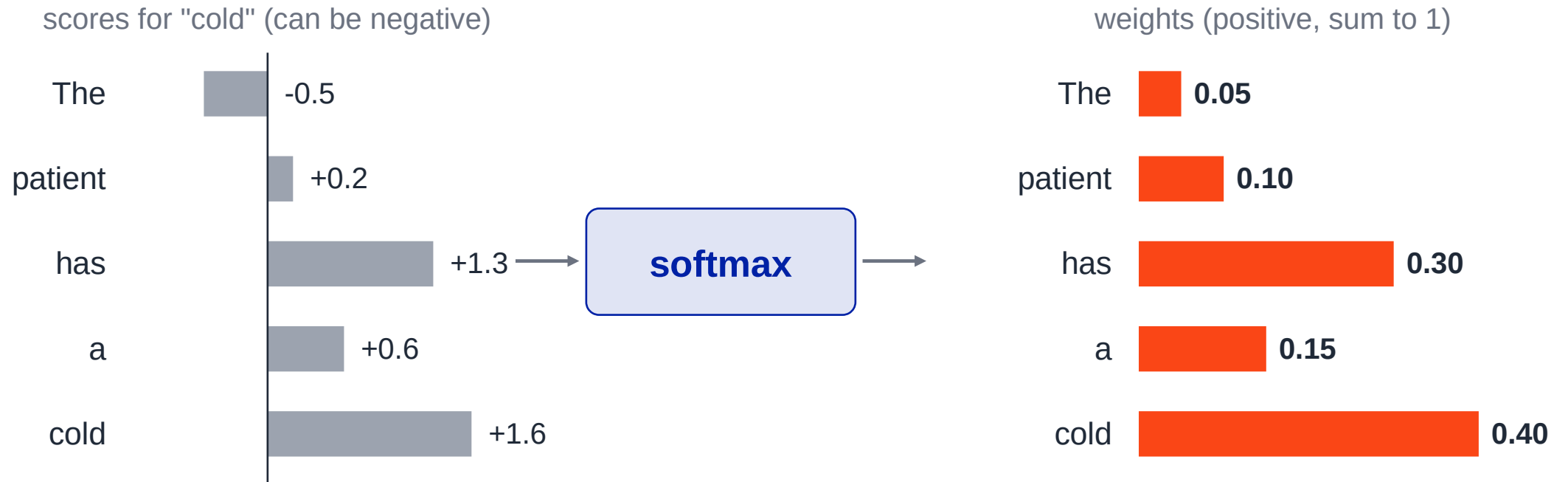
 $= (1)(-1) + (0)(2) + (2)(0)$ **= -1**

different direction: low score

Every token computes a score like this against every other token, including itself.

From Scores to Weights: Softmax

Softmax turns any list of scores into positive weights that sum to 1. A higher score gets more weight.



$$\text{weight}_i = \exp(\text{score}_i) / \sum_j \exp(\text{score}_j)$$



Queries, Keys, and Values

One Vector, Three Jobs

In attention each token asks, gets matched, and supplies content. One vector is too limited for all three.

Query

What am I looking for?

"cold" looks for words that tell it which sense it has

Library:
your search terms

Key

What do I offer to be matched on?

"has" advertises that it is a verb of possession

Library:
the catalog entry

Value

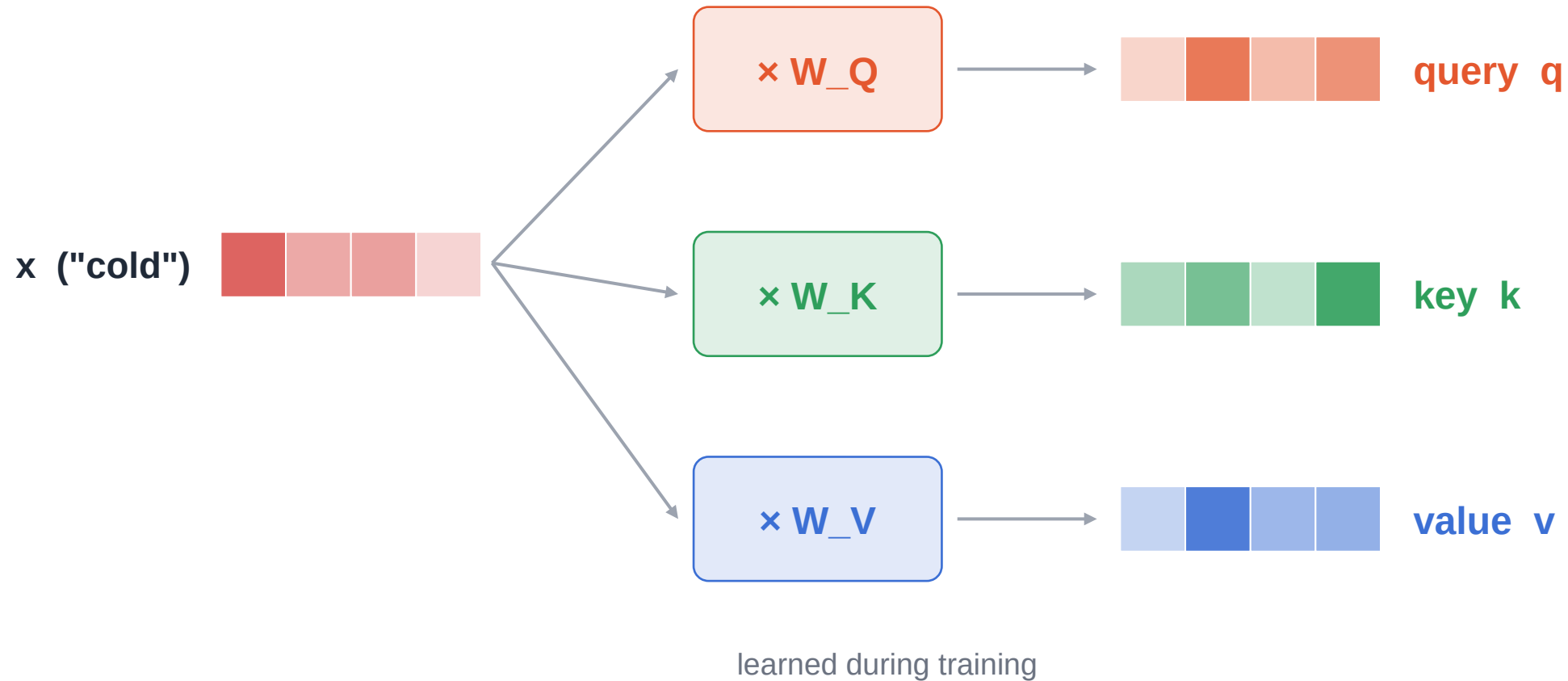
What do I pass on if chosen?

"has" hands over its content to "cold"

Library:
the book itself

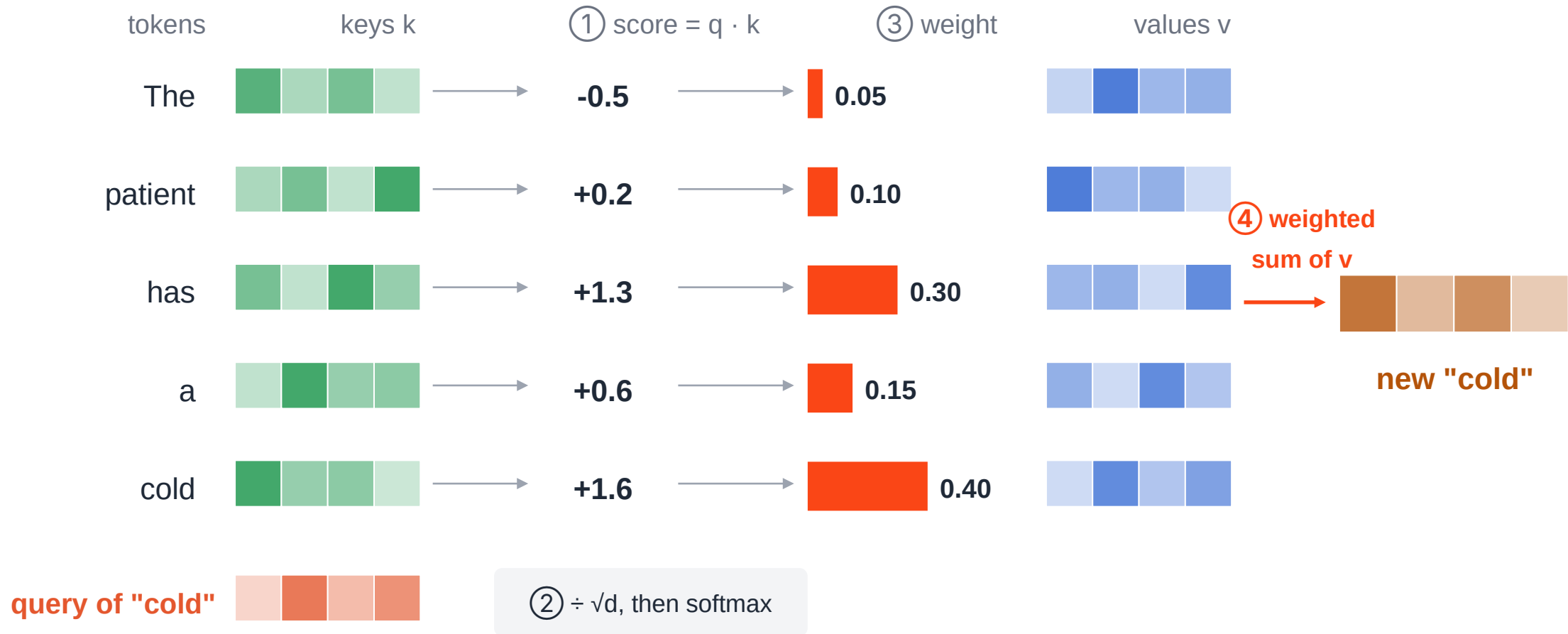
Three Learned Projections

Three weight matrices turn each token's embedding into a query, a key, and a value.



Attention for One Token

Compare the query of 'cold' with every key, turn the scores into weights, then mix the values.



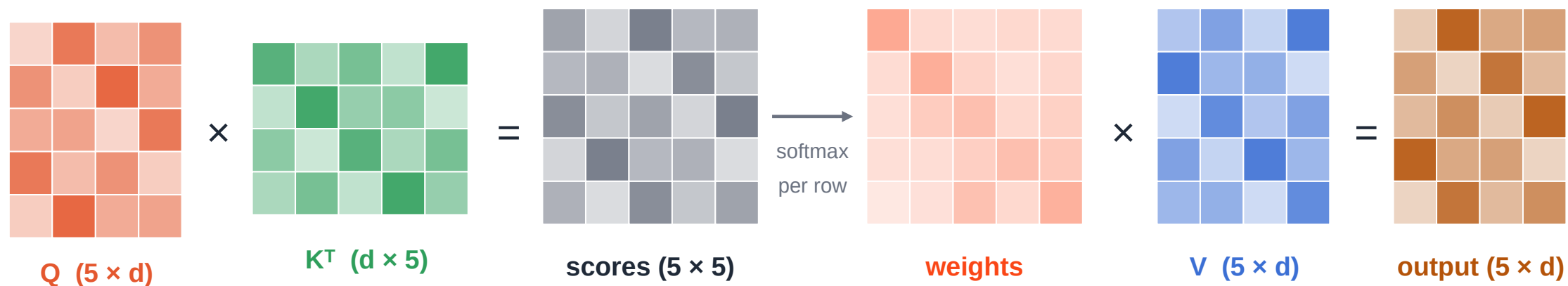
Animation: Attention for One Token

Animation clip: week02-clip2-attention-one-token.mp4

Silent version embedded (click to play). The narrated version is in the course repository.

All Tokens at Once: The Matrix Form

Stack the vectors into matrices and every token is computed in parallel. This is the formula in the paper.

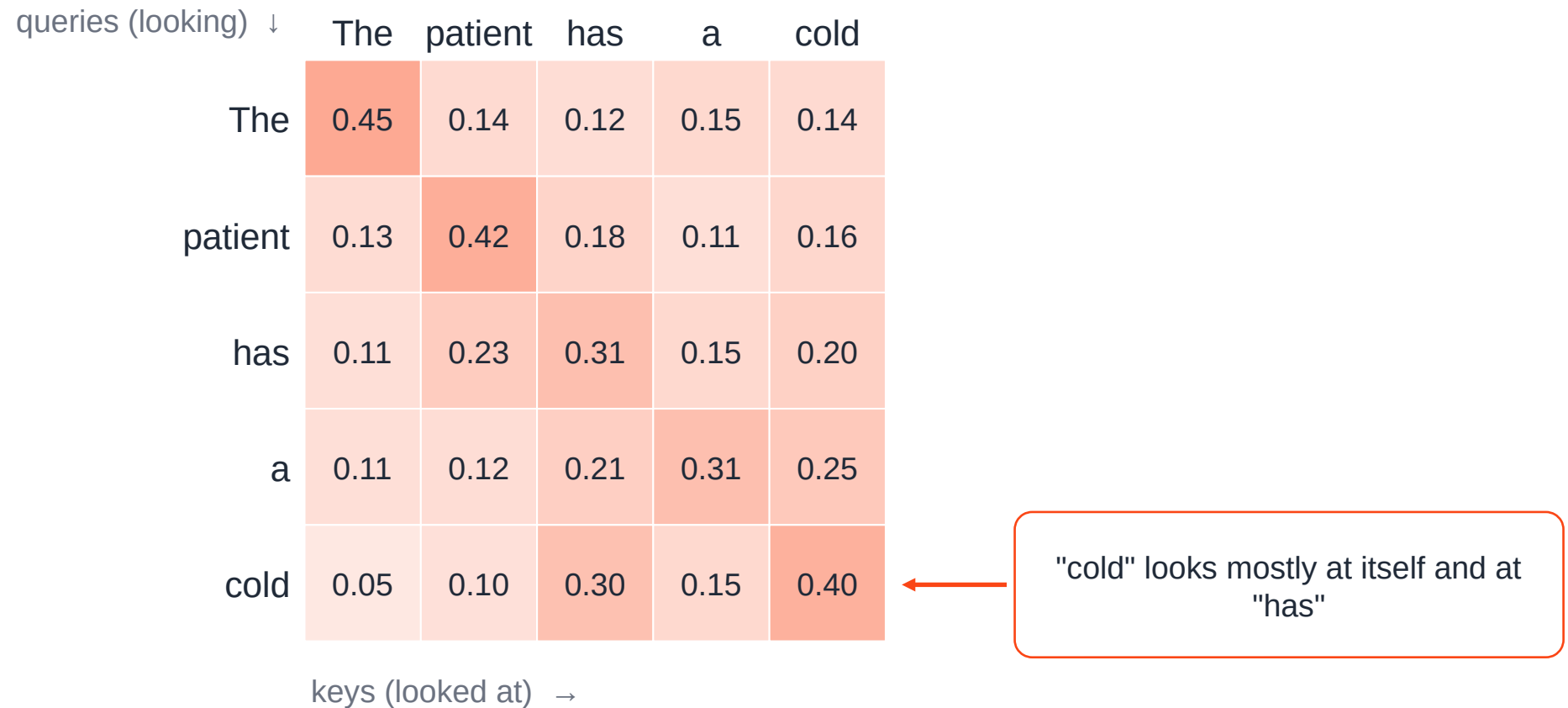


$$\text{Attention}(Q, K, V) = \text{softmax}(Q K^T / \sqrt{d_k}) V$$

Vaswani et al., "Attention Is All You Need," 2017

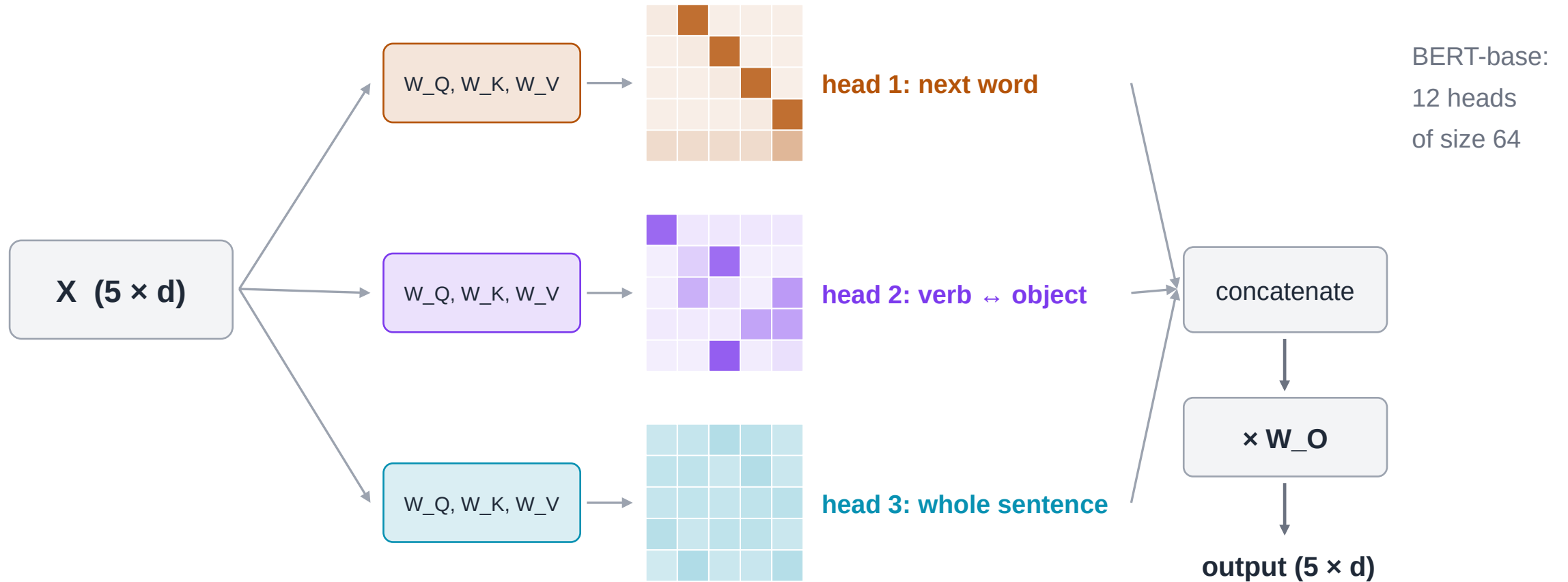
Reading an Attention Map

Row i shows where token i looks. Each row sums to 1.



Multi-Head Attention

Several attention heads run side by side, each with its own W_Q , W_K , W_V , so each can learn a different relationship.





Building the Encoder Block

Attention Ignores Word Order

Attention compares every token with every token, so shuffling the input only shuffles the output.

chest pain , no pressure

reports chest pain

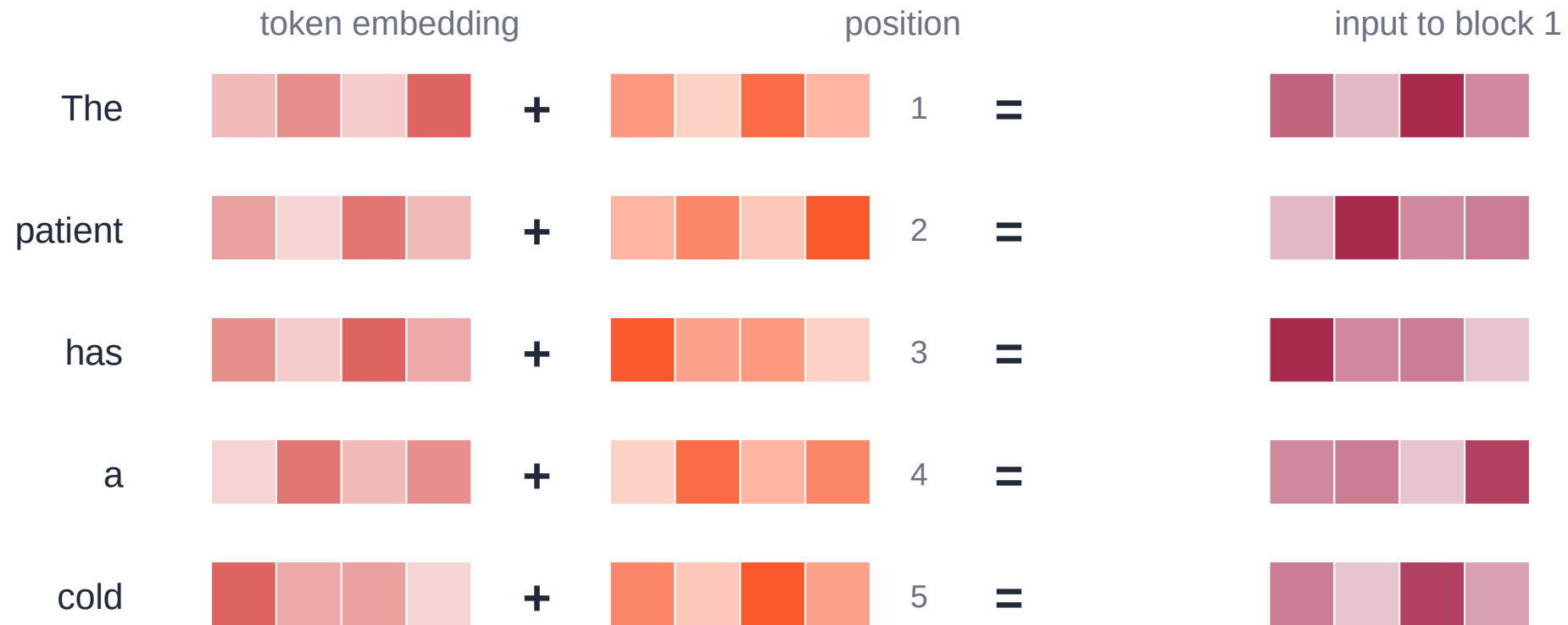
no pain , chest pressure

reports chest pressure, no pain

Same five tokens. Without position information, attention cannot tell these apart.

Add a Position Vector to Each Token

Each position gets its own vector, which is added to the token embedding before the first block.



Learned (BERT) · sines and cosines (original transformer) · rotary, inside attention (Llama)

Animation: Giving Tokens a Position

Animation clip: week02-clip3-positions.mp4

Silent version embedded (click to play). The narrated version is in the course repository.

Relative Positions: RoPE and ALiBi

Newer models build distance into attention instead of adding positions at the input.

RoPE: rotate q and k by position

chest at 2, pain at 3	0.93	distance 1
chest at 7, pain at 8	0.93	distance 1
chest at 2, pain at 6	0.28	distance 4

before rotation: 1.04
same distance, same score

ALiBi: subtract slope × distance

The	patient	has	a	cold
-0.5	0.2	1.3	0.6	1.6
-1	-0.75	-0.5	-0.25	0
-1.5	-0.55	0.8	0.35	1.6

weights for cold: 0.05 0.10 0.30 0.15 0.40

with ALiBi: 0.02 0.06 0.24 0.15 0.53

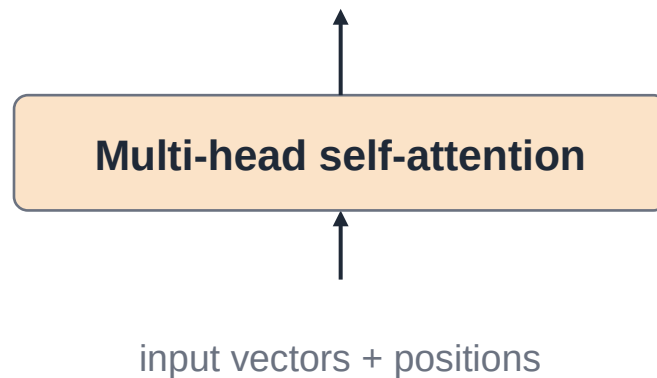
Animation: Relative Positions: RoPE and ALiBi

Animation clip: week02-clip4-relative-positions.mp4

Silent version embedded (click to play). The narrated version is in the course repository.

Block Step 1: Self-Attention

The block starts with the multi-head self-attention we just built. Each token gathers context from the others.



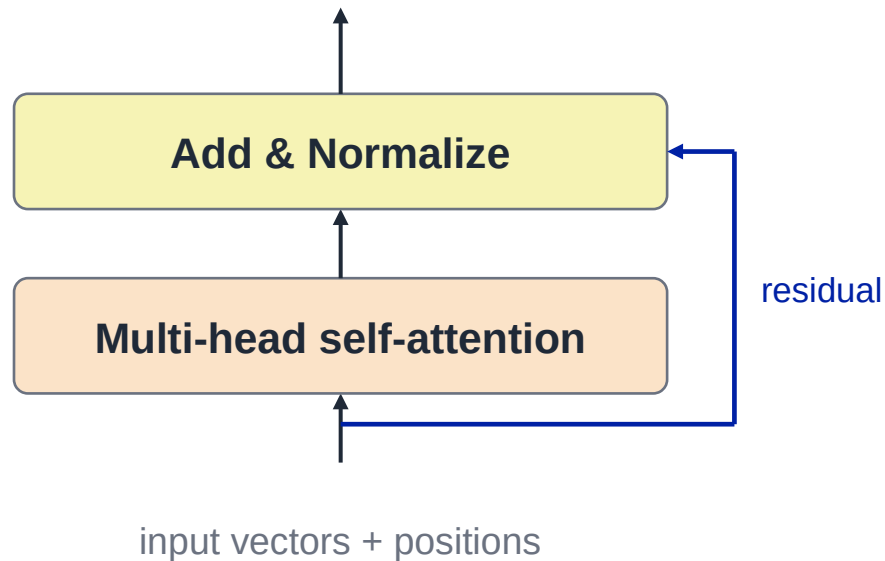
In: one vector per token

Out: one contextual vector per token,
same size

Mixes information across tokens

Block Step 2: Add & Normalize

Add the block's input back to the attention output, then normalize. This keeps the original information and stabilizes training.



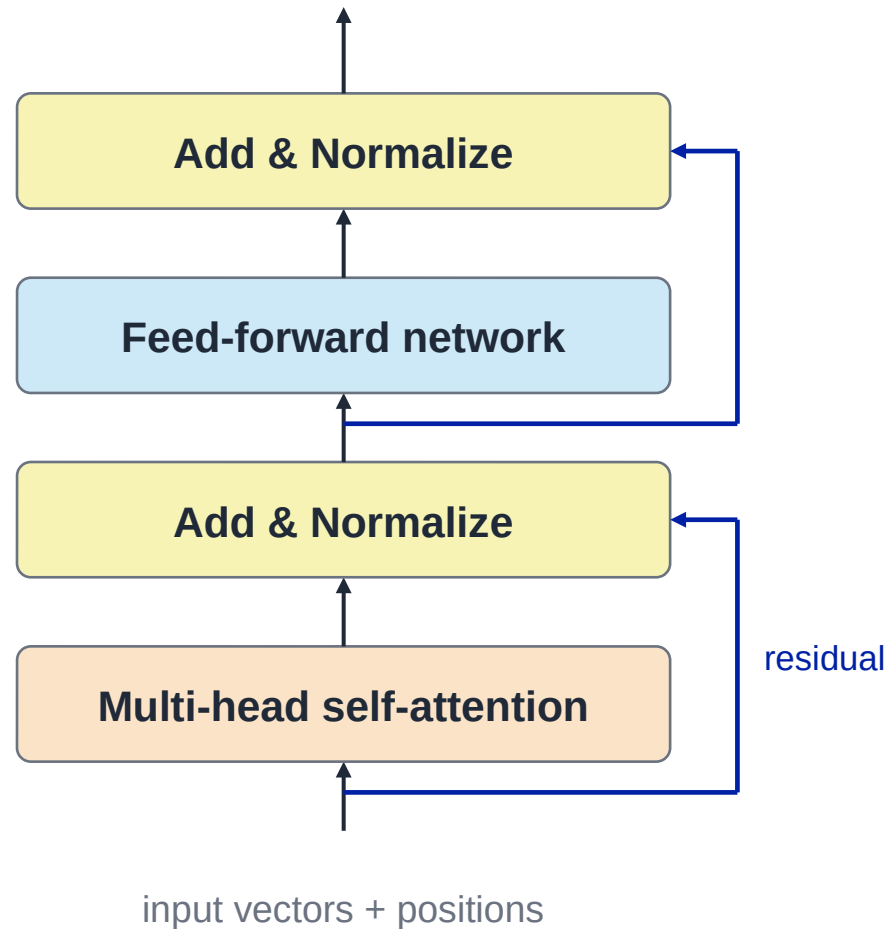
Add: $\text{output} = \text{input} + \text{attention}(\text{input})$

Normalize: rescale each token vector

Why: deep stacks train reliably

Block Step 3: Feed-Forward Network

A small two-layer network then processes each token on its own, followed by another Add & Normalize.

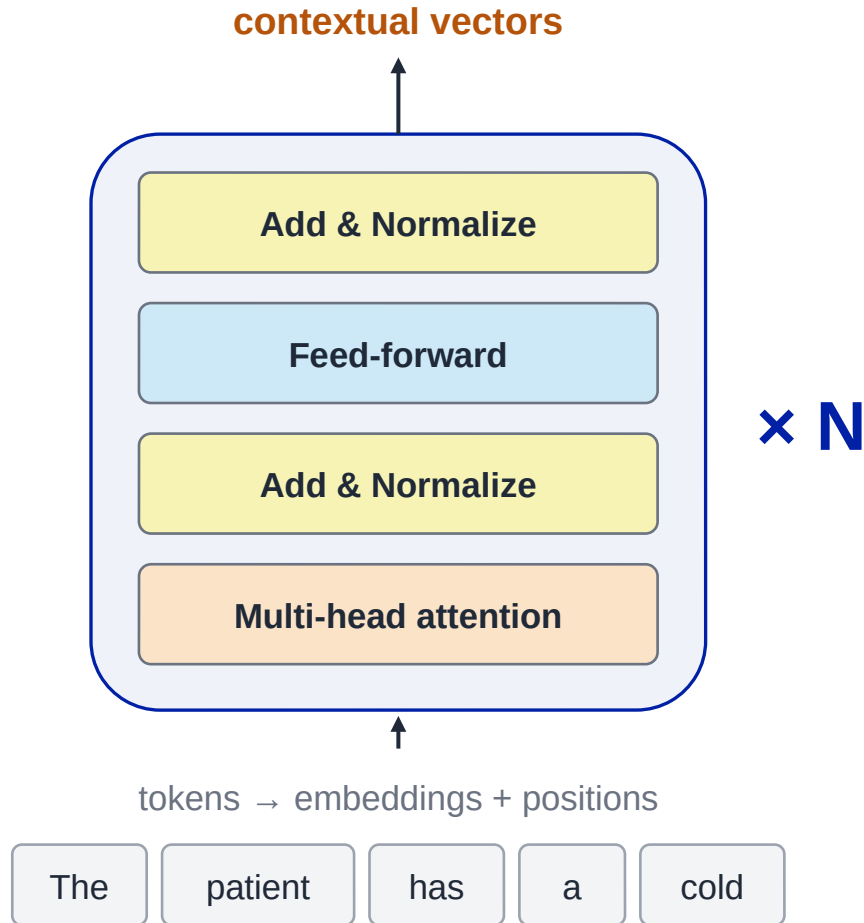


Per token: $d \rightarrow 4d \rightarrow d$
(with a GELU nonlinearity)

Attention: mixes across tokens
Feed-forward: transforms within a token

Stack the Blocks

Each block keeps the shape (tokens $\times$ d), so blocks can be stacked. The output is one contextual vector per token.



BERT-base

12 blocks, $d = 768$, 12 heads, 110M parameters

BERT-large

24 blocks, $d = 1024$, 16 heads, 340M parameters

Shape in = shape out

(5 tokens $\times$ d) $\rightarrow$ (5 tokens $\times$ d) at every block

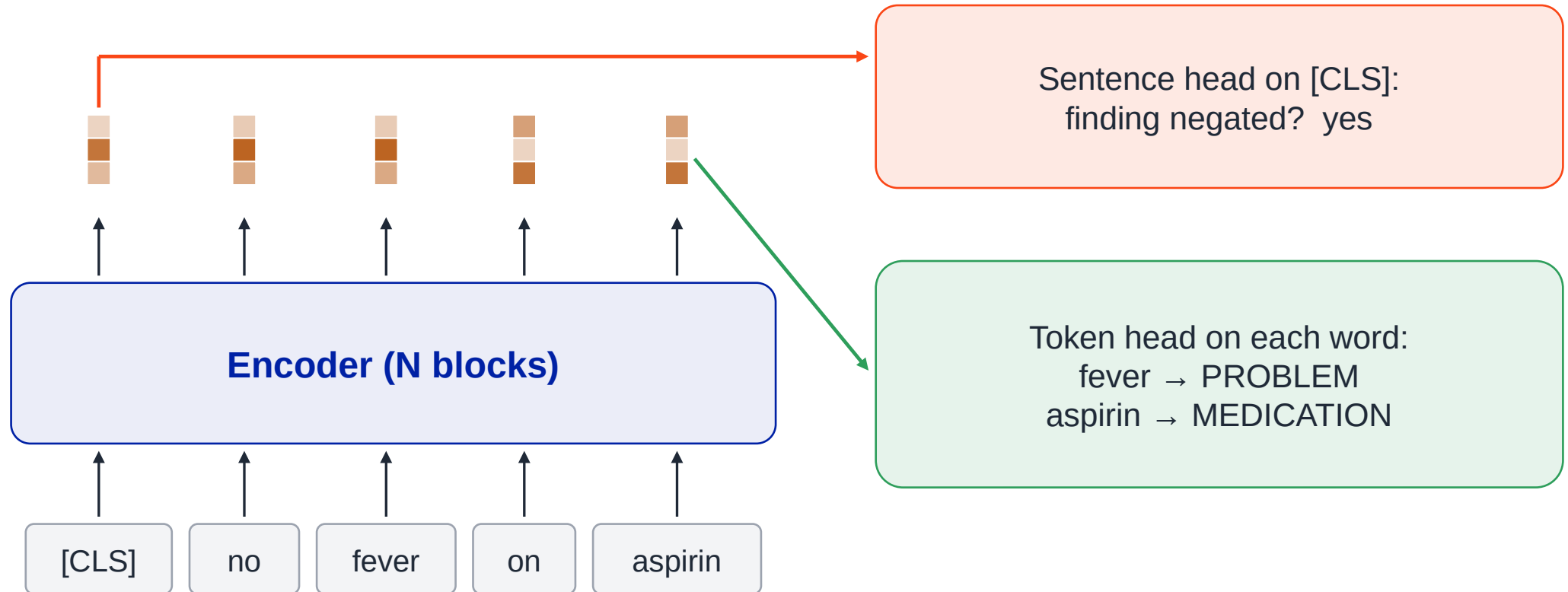
Animation: Inside the Encoder Block

Animation clip: week02-clip5-encoder-block.mp4

Silent version embedded (click to play). The narrated version is in the course repository.

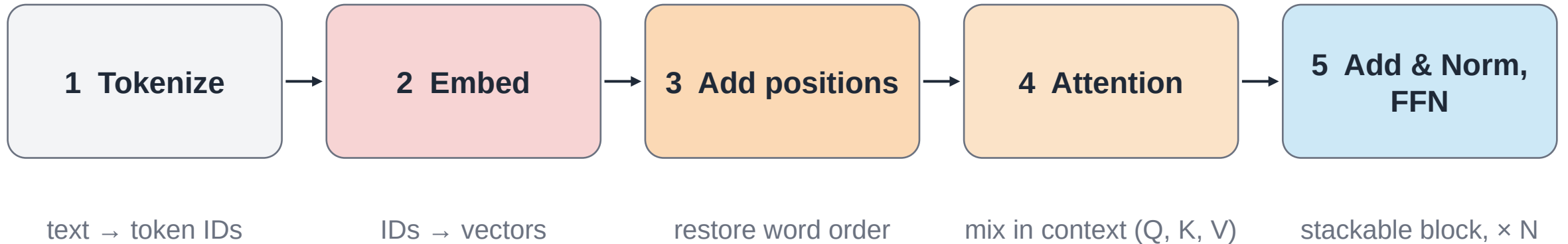
Using the Encoder Output

A small prediction head on top of the contextual vectors turns them into task outputs.



Recap: From Text to Contextual Vectors

Five steps, each fixing a limitation of the one before.



$$\text{Attention}(Q, K, V) = \text{softmax}(Q K^T / \sqrt{d_k}) V$$

Before Thursday

Read

Devlin et al., BERT (NAACL 2019) and Yang et al., GatorTron (npj Digital Medicine 2022). Both are encoders built from today's block.

Write

Reading responses on Canvas before Thursday's class: one paragraph and one discussion question per paper.

Explore

Week 2 reference notebook (ungraded): every step of today's lecture as code, with the same numbers.

Next week

Part 2: predicting the next token, masking, the decoder, model families, and transformers for images.