RADONC AI CURRICULUM

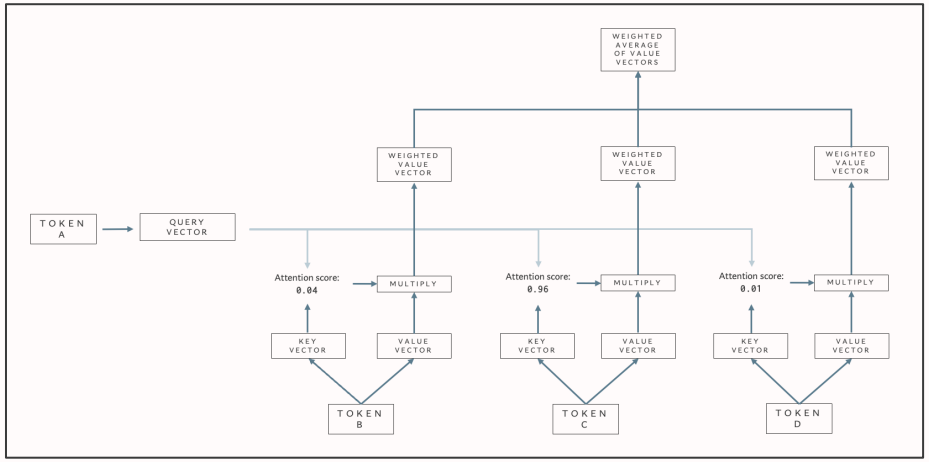
Introduction to Deep Learning

LECTURE 10: Transformers

Andrew Y. K. Foong, Ph.D.

MAYO CLINIC

Radiation Oncology
AI & Data Analytics
AIDA



Today's lecture

1. Network architectures
   - *Wiring neurons together*
2. Sequence processing
   - *Handling variable-length inputs*
3. The attention mechanism
   - *How tokens relate to each other*
4. The dot product
   - *Computing vector similarity*
5. Embedding layers
   - *Computing the key and query vectors*
6. The value vector
   - *The output of the attention mechanism*
7. Attention and the transformer
   - *Bringing it all together*

Network Architectures

Wiring neurons together

Neural network architectures

- An *architecture* is a specific way of connecting neurons in a neural network.
- This is *distinct* from:
  - What you use it for (classification, regression)
  - What kind of data you train it on (time-series, images, text, audio)
- Most architectures can be used for any kind of task and data.
  - *But*: some work much better than others for certain kinds of tasks!
  - E.g., *convolutional neural networks* are excellent for image data.
- The TRANSFORMER is the most successful architecture in the history of deep learning.
  - Was originally developed for language translation
  - It is now used for almost any kind of data: text, images, video, audio…
  - ChatGPT, Gemini, Claude, DeepSeek, Copilot…

Attention is all you need

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

- Pivotal 2017 paper introducing the transformer.
- Cited over 200,000 times.

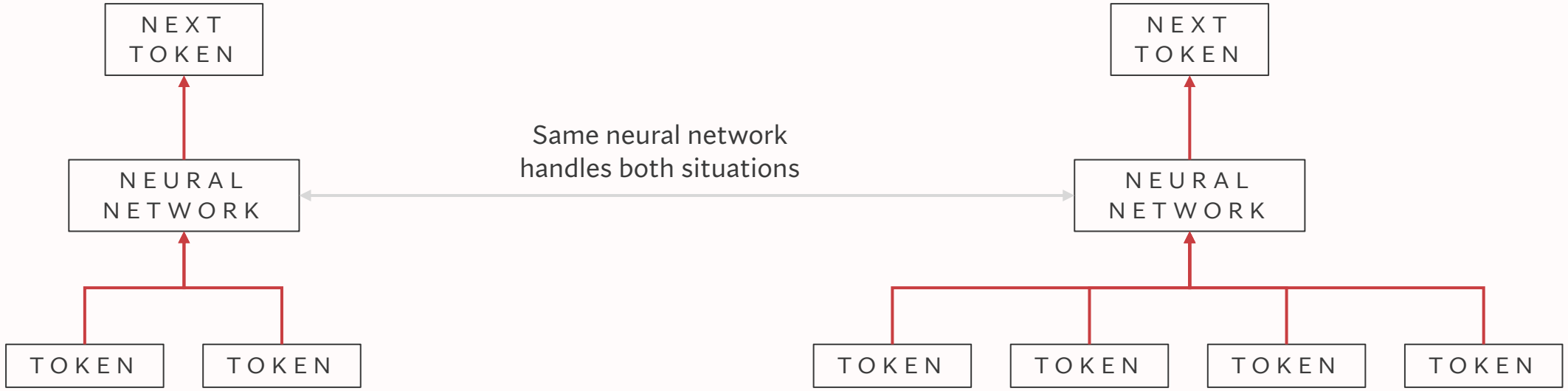
Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need.” Advances in neural information processing systems 30 (2017).

Sequence Processing

Handling variable-length inputs

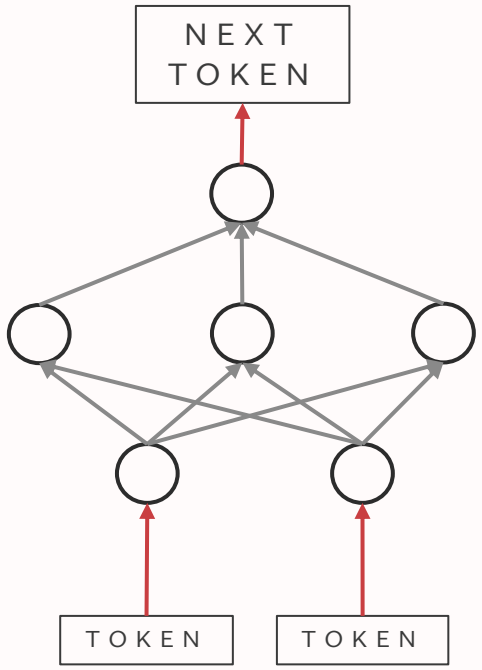
Sequence processing

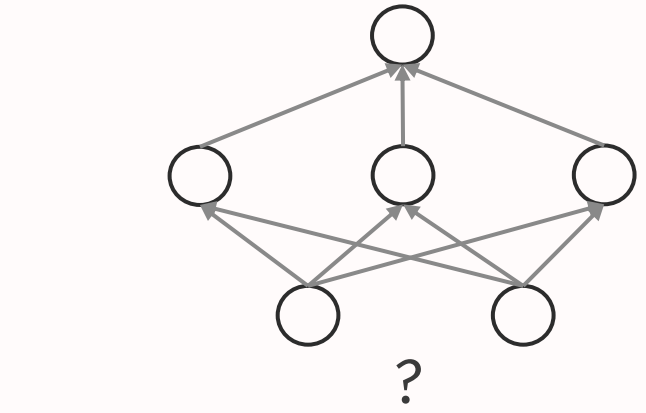
- Transformers answer the question: *how can we use a single neural network rto process arbitrary-length sequences of data?*
- EXAMPLE:
  - Text comes in arbitrary length: *questions, notes, chat logs, etc.*
  - Images can come in arbitrary size
  - Audio files can come in arbitrary length



Sequence processing

- Sequence processing cannot be handled by standard *fully connected networks*.
- If the right number of input neurons is used for one sequence, it will be the wrong number of neurons for another sequence.





TOKEN TOKEN TOKEN TOKEN

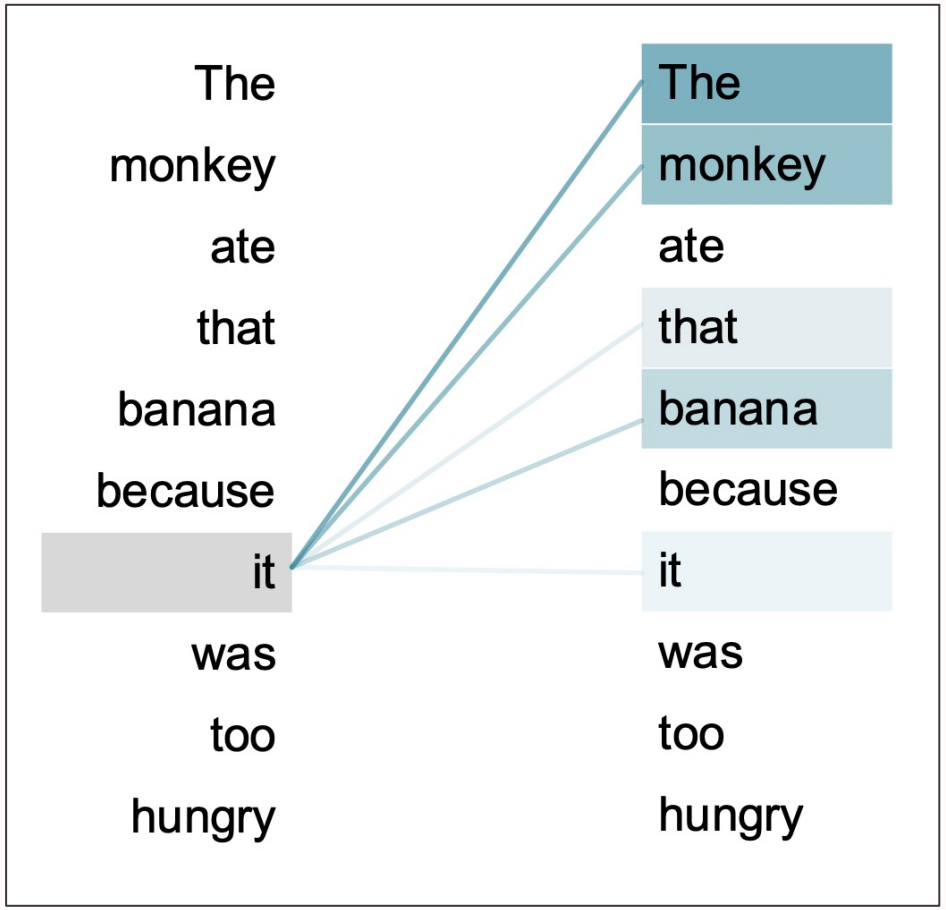
Sequence processing

- SOLUTION: Use a neural network to describe the *relationship between tokens.*
- In transformers, this is known as the *attention mechanism.*
- ANALOGY:
  - A standard fully-connected network is like a chef mixing all ingredients into a single dish:
    - Input goes in
    - Processed by a fixed “recipe”
    - Output comes out
  - The attention mechanism is like a chef who is evaluating how well each ingredient pairs with each other ingredient:
    - Garlic + tomato: strong connection
    - Basil + chocolate: weak
    - Chili + lime: strong

The Attention Mechanism

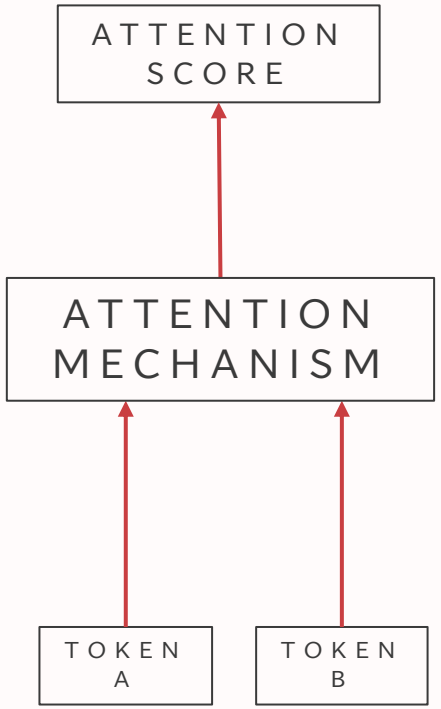
How tokens relate to each other

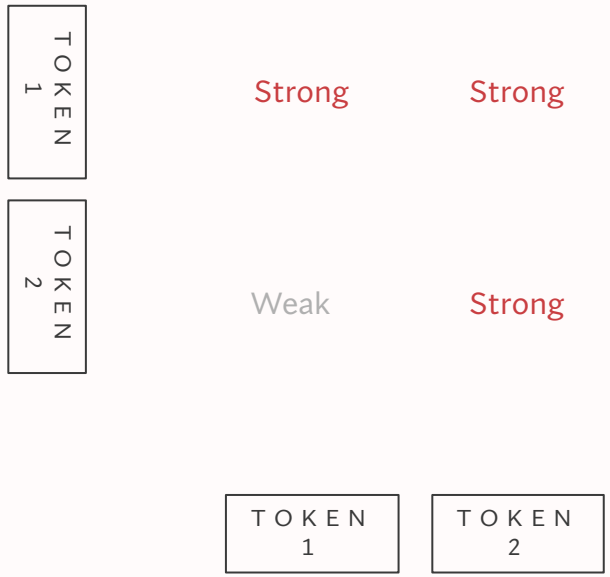
Modeling relationships with attention



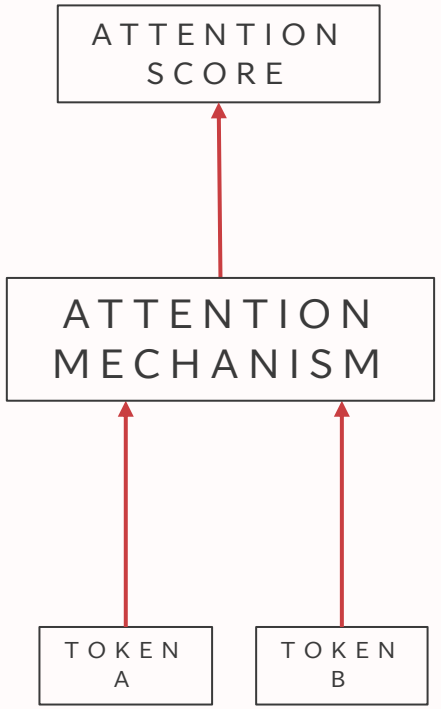
Xie, Huiqiang, Zhijin Qin, Geoffrey Ye Li, and Biing-Hwang Juang. "Deep learning enabled semantic communication systems." IEEE transactions on signal processing 69 (2021): 2663-2675.

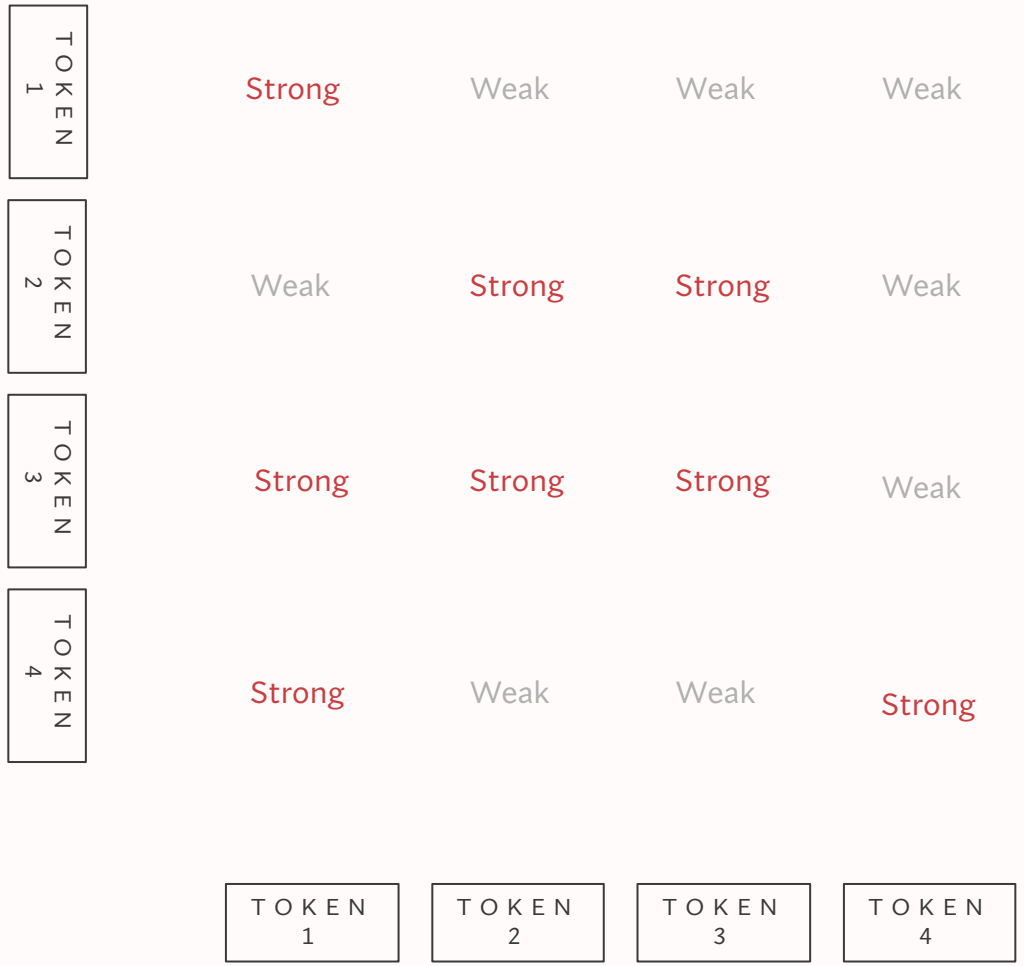
Modeling relationships with attention





Modeling relationships with attention

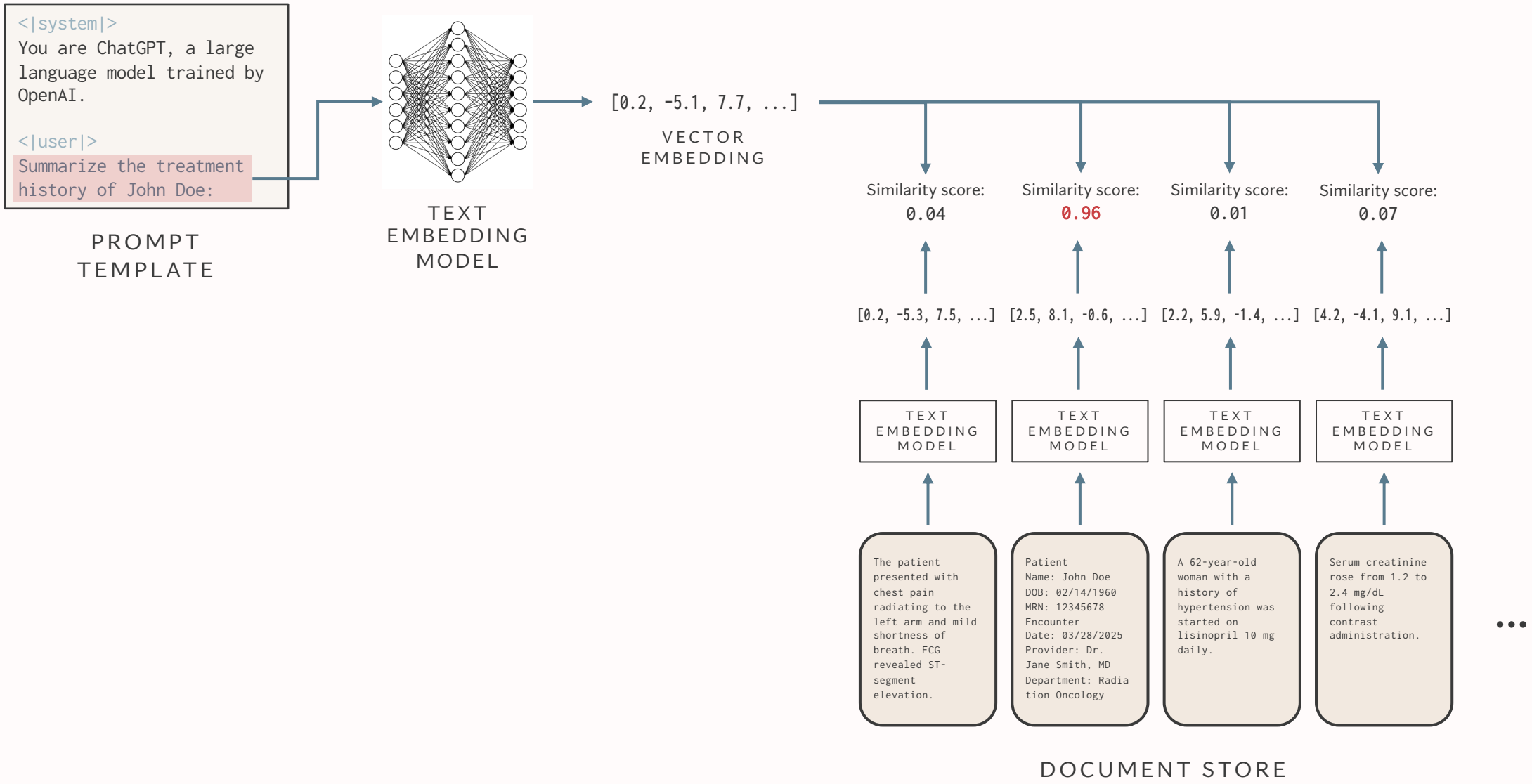




Dot-product attention

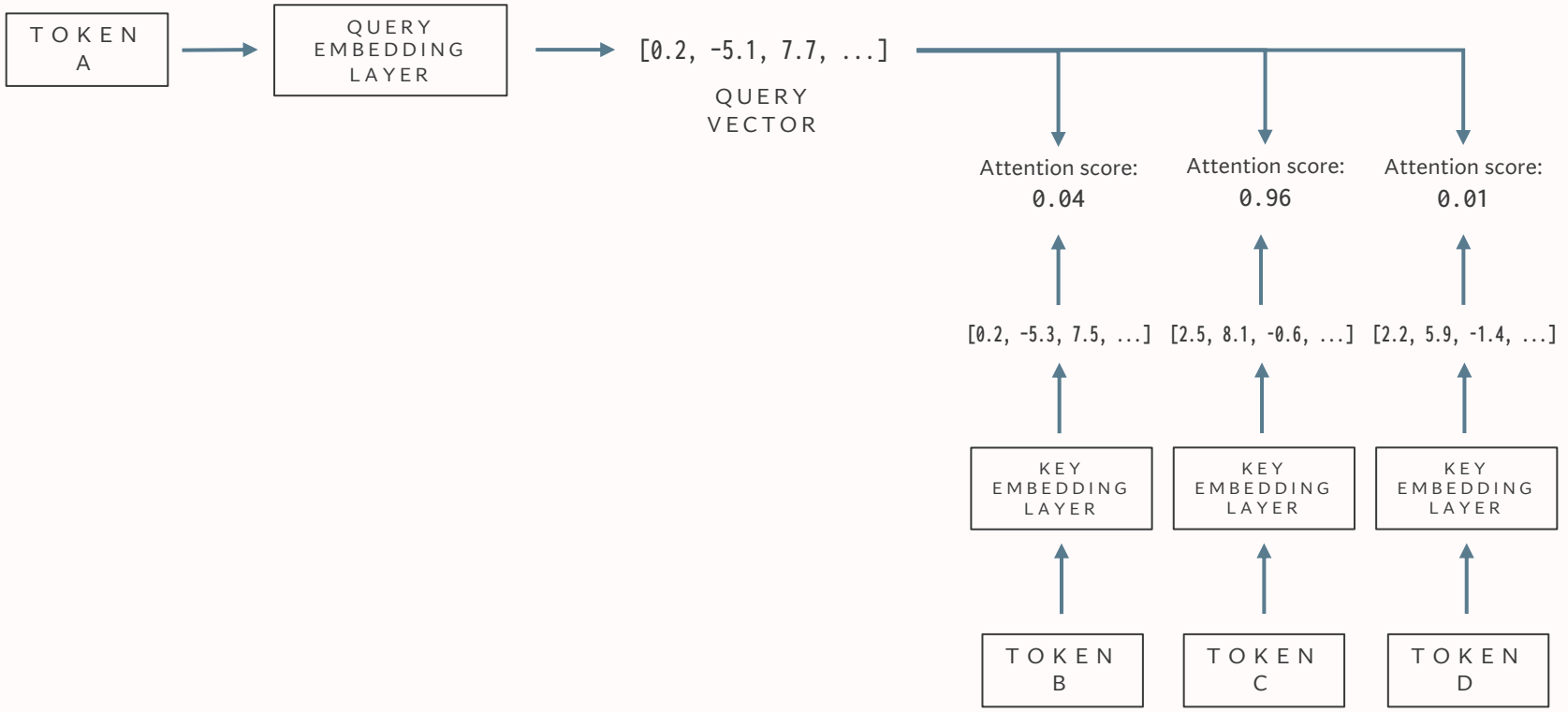
- How does this attention mechanism work?
- IDEA: use a similar idea to *retrieval-augmented generation*.
- For each token, we create an embedding corresponding to that token, and we find a "match" between that token's embedding and another token's embedding.
- The match is computed using a *dot-product*.
- A high value for the dot product corresponds to a strong match, and a low value corresponds to a weak match.

Recap: Vector RAG



Dot-product attention

- Each token is processed by the same *key embedding layer*
- Each token is processed by the same *query embedding layer*
- The *query vector* is compared to each *key vector* to form an *attention score*



Dot-product attention

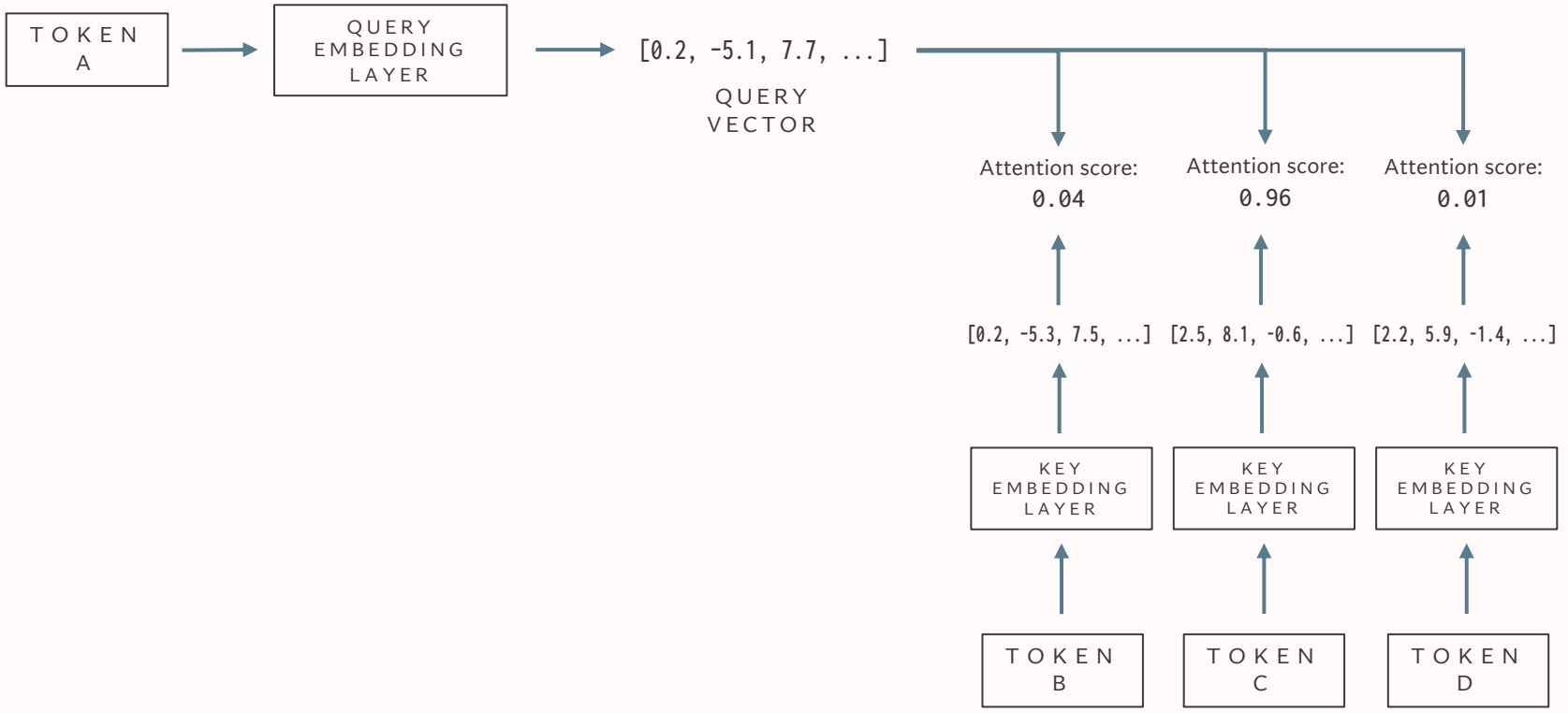
- The query and key embedding layers are *learnable*—they are initialized randomly and optimized by gradient descent
- Attention provides a general way for networks to represent the strength of relationships between inputs

The Dot Product

Computing vector similarity

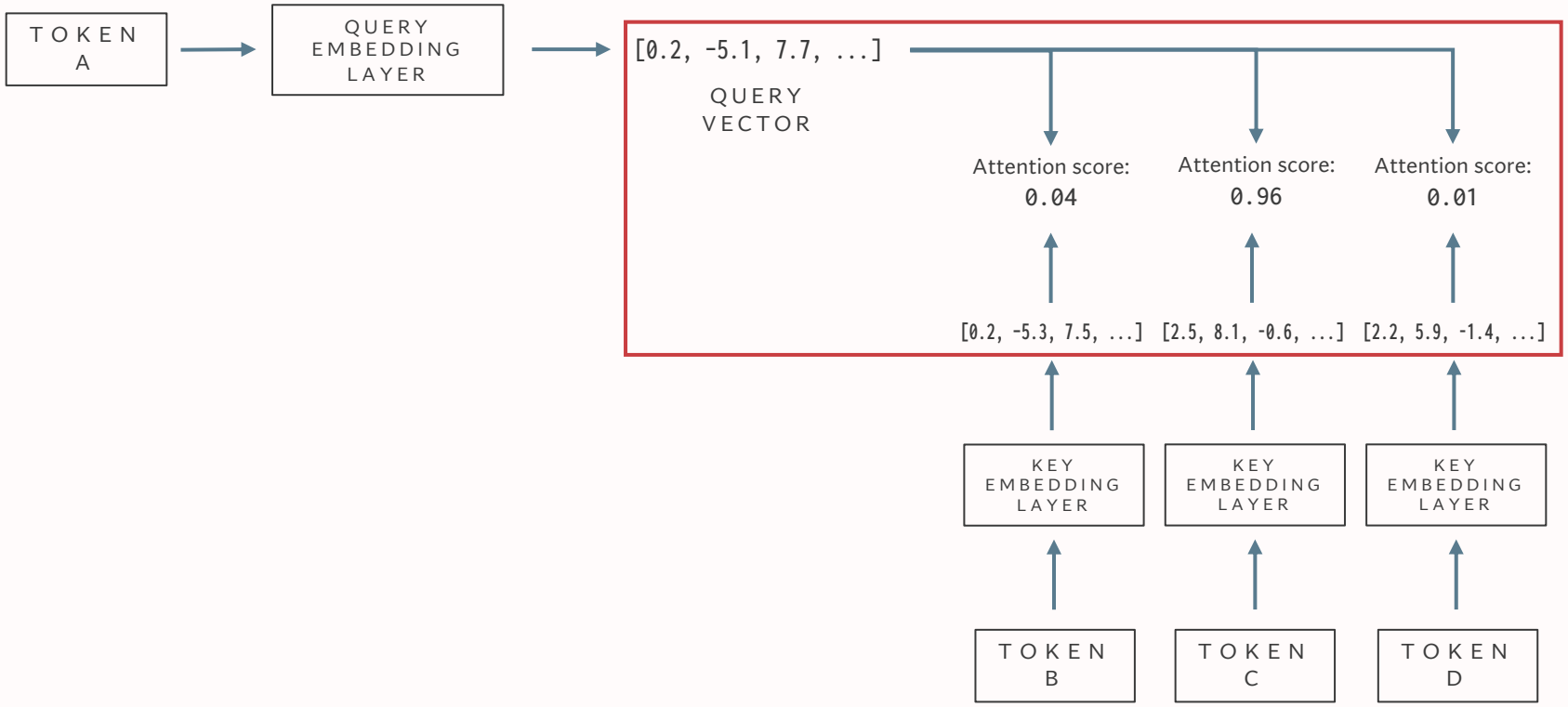
Recap: dot-product attention

- Each token is processed by the same *key embedding layer*
- Each token is processed by the same *query embedding layer*
- The *query vector* is compared to each *key vector* to form an *attention score*



Recap: dot-product attention

- Each token is processed by the same *key embedding layer*
- Each token is processed by the same *query embedding layer*
- The *query vector* is compared to each *key vector* to form an *attention score*



The dot-product operation

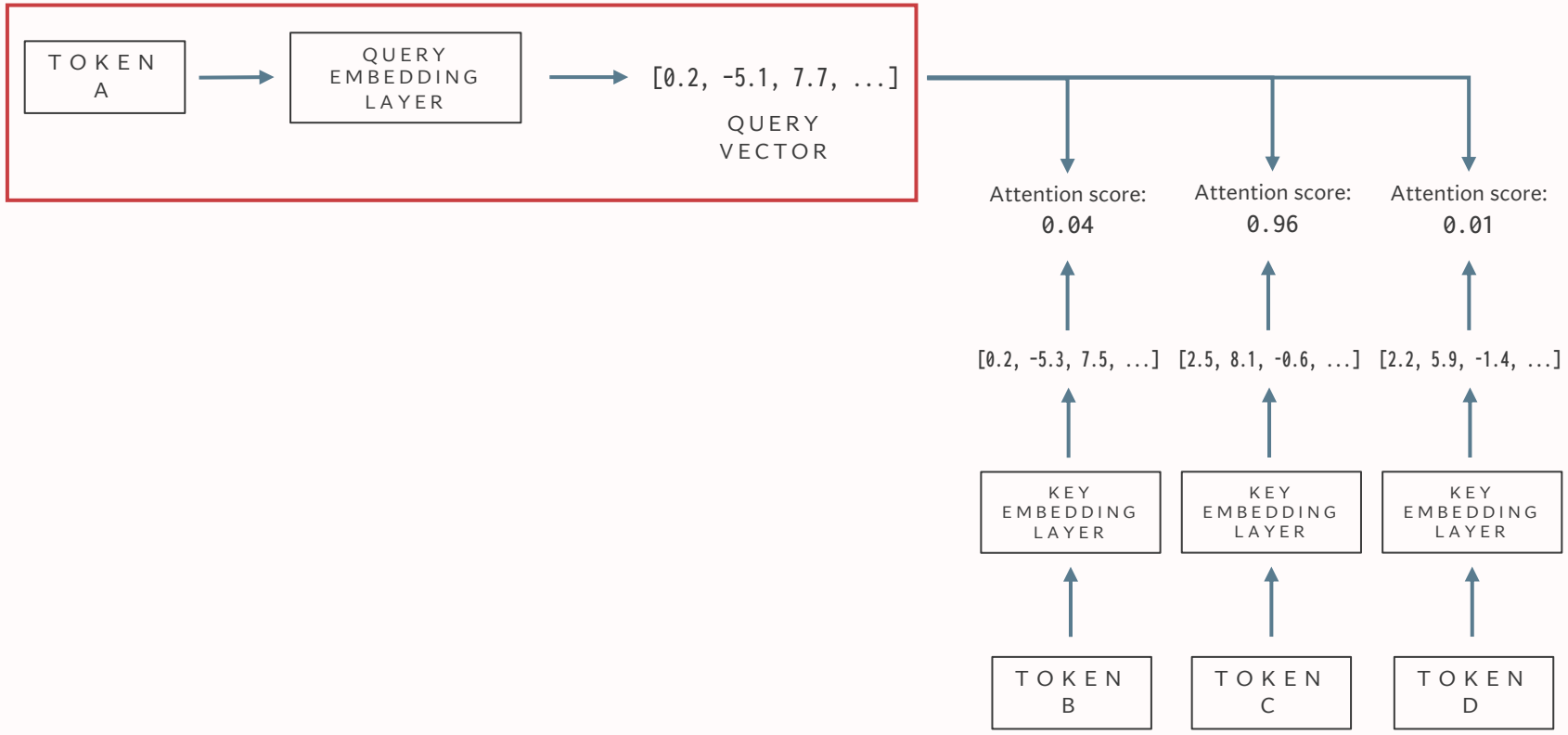
- How are these attention scores computed?
- The *dot-product* takes two vectors and outputs a single number:
  - $\mathbf{a} = [a_1, a_2, ..., a_N]$
  - $\mathbf{b} = [b_1, b_2, ..., b_N]$
  - $\mathbf{a} \cdot \mathbf{b} = a_1 \times b_1 + a_2 \times b_2 + ... + a_N \times b_N$
  - $\mathbf{a} \cdot \mathbf{b}$ can be positive, negative or zero.
  - If $\mathbf{a} \cdot \mathbf{b}$ is very positive, can roughly be thought of as "**a** *and* **b** *point in the same direction.*"—positively correlated entries
  - If $\mathbf{a} \cdot \mathbf{b}$ is very negative, can roughly be thought of as "**a** *and* **b** *point in the opposite direction.*"—negatively correlated entries
  - If $\mathbf{a} \cdot \mathbf{b}$ is close to zero, can roughly be thought of as "**a** *and* **b** *point in perpendicular/orthogonal directions.*" —uncorrelated entries

Embedding Layers

Computing the key and query vectors

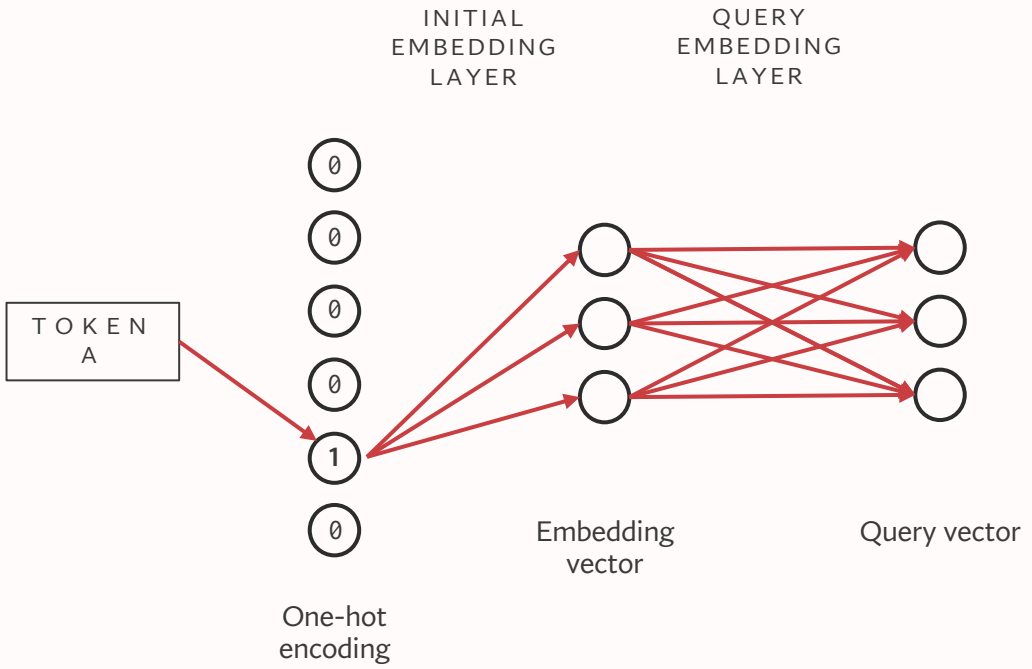
Recap: dot-product attention

- Each token is processed by the same *key embedding layer*
- Each token is processed by the same *query embedding layer*
- The *query vector* is compared to each *key vector* to form an *attention score*



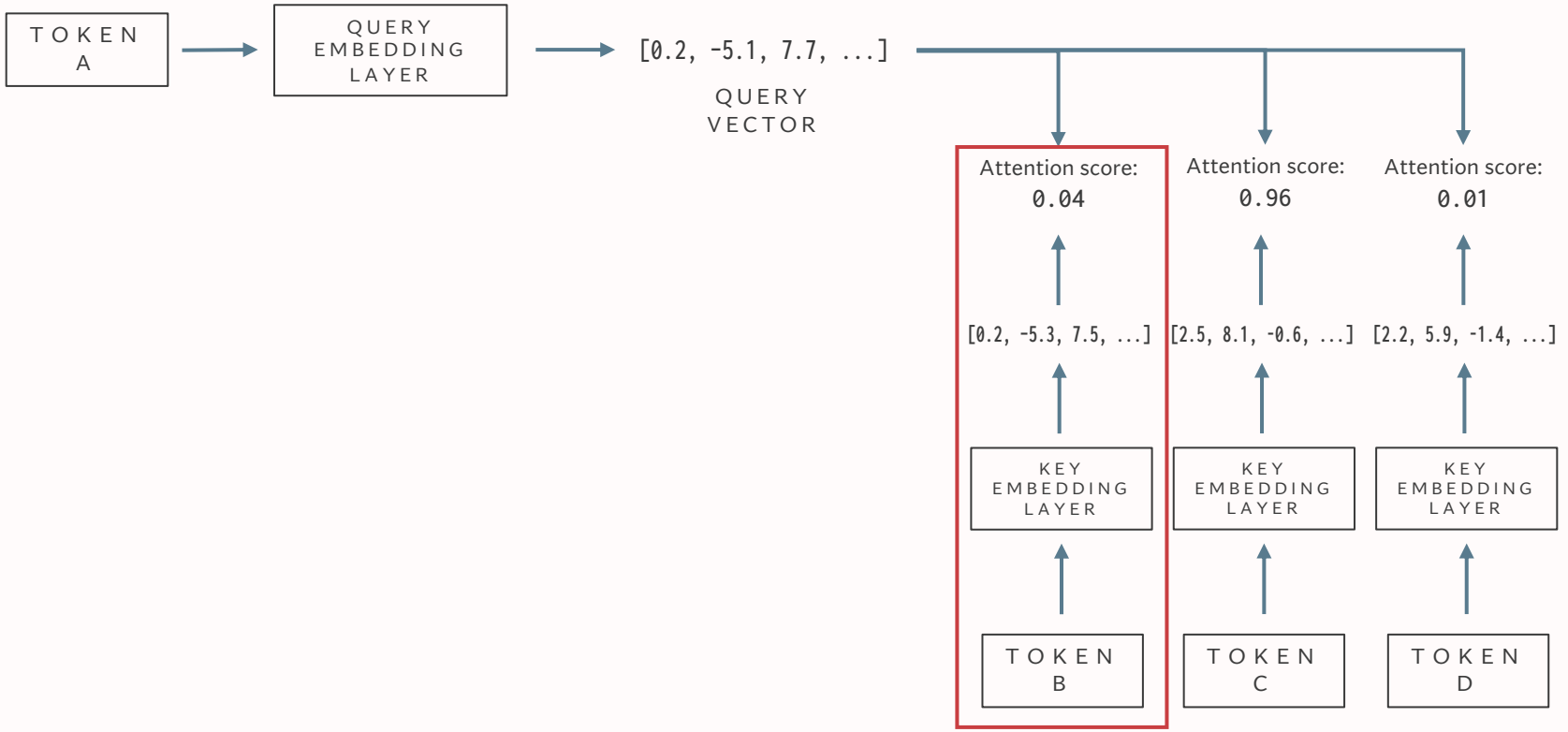
The query embedding

- How does the query vector get computed from a token?
  1. The token is one-hot encoded
  2. The one-hot encoded vector is embedded with a single linear layer of weights
  3. The embedded vector is processed with another linear layer of weights to form the query vector

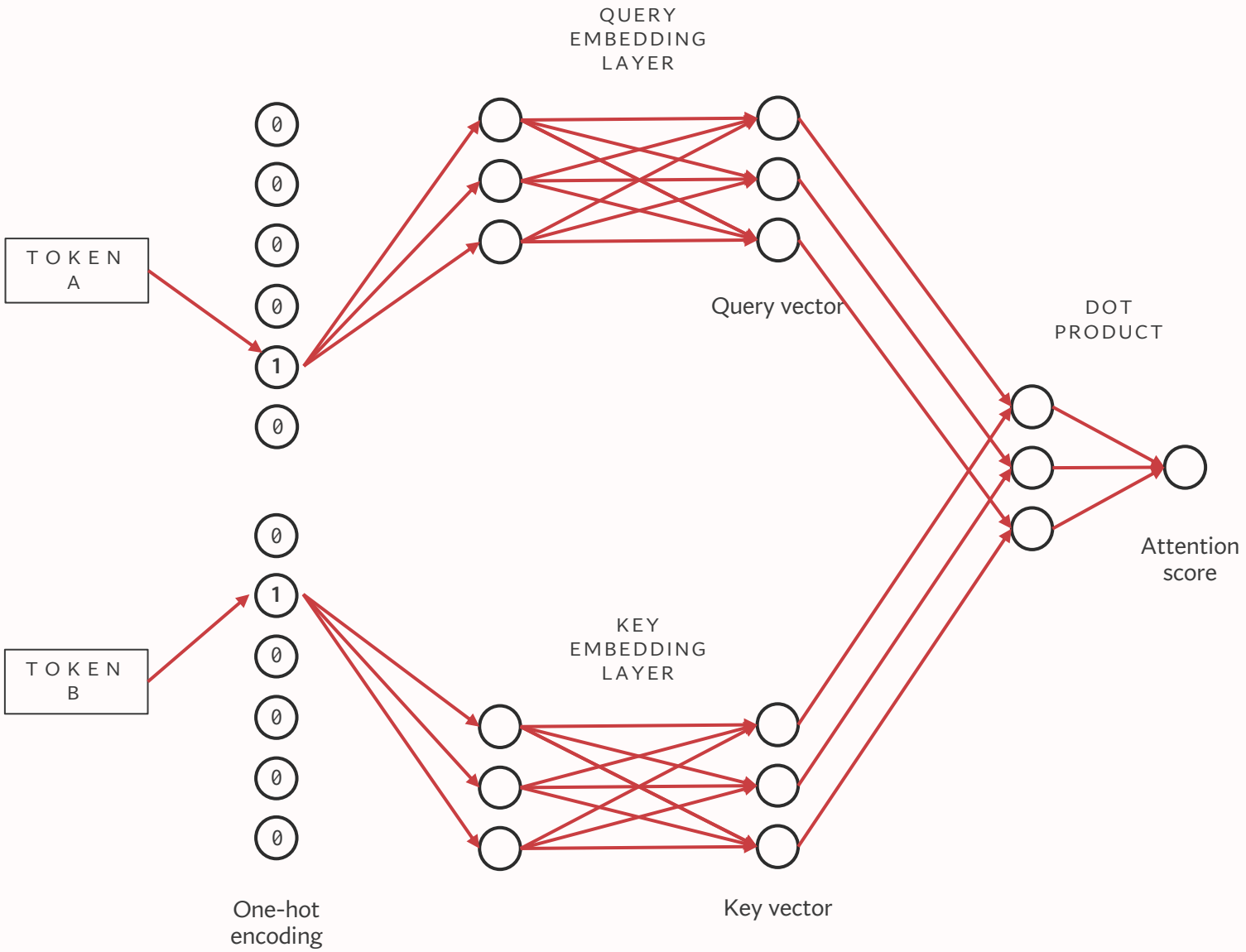


Recap: dot-product attention

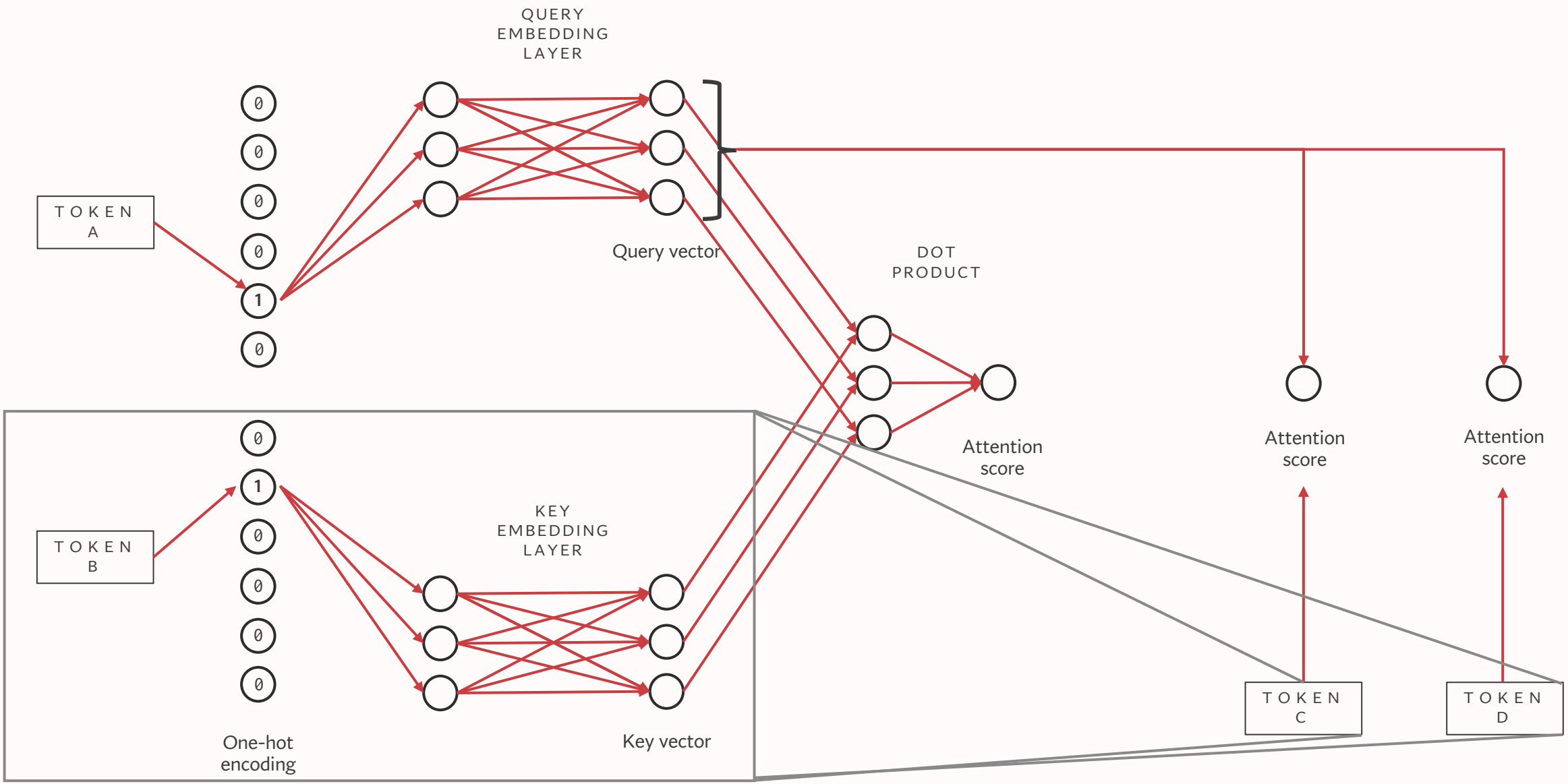
- Each token is processed by the same *key embedding layer*
- Each token is processed by the same *query embedding layer*
- The *query vector* is compared to each *key vector* to form an *attention score*



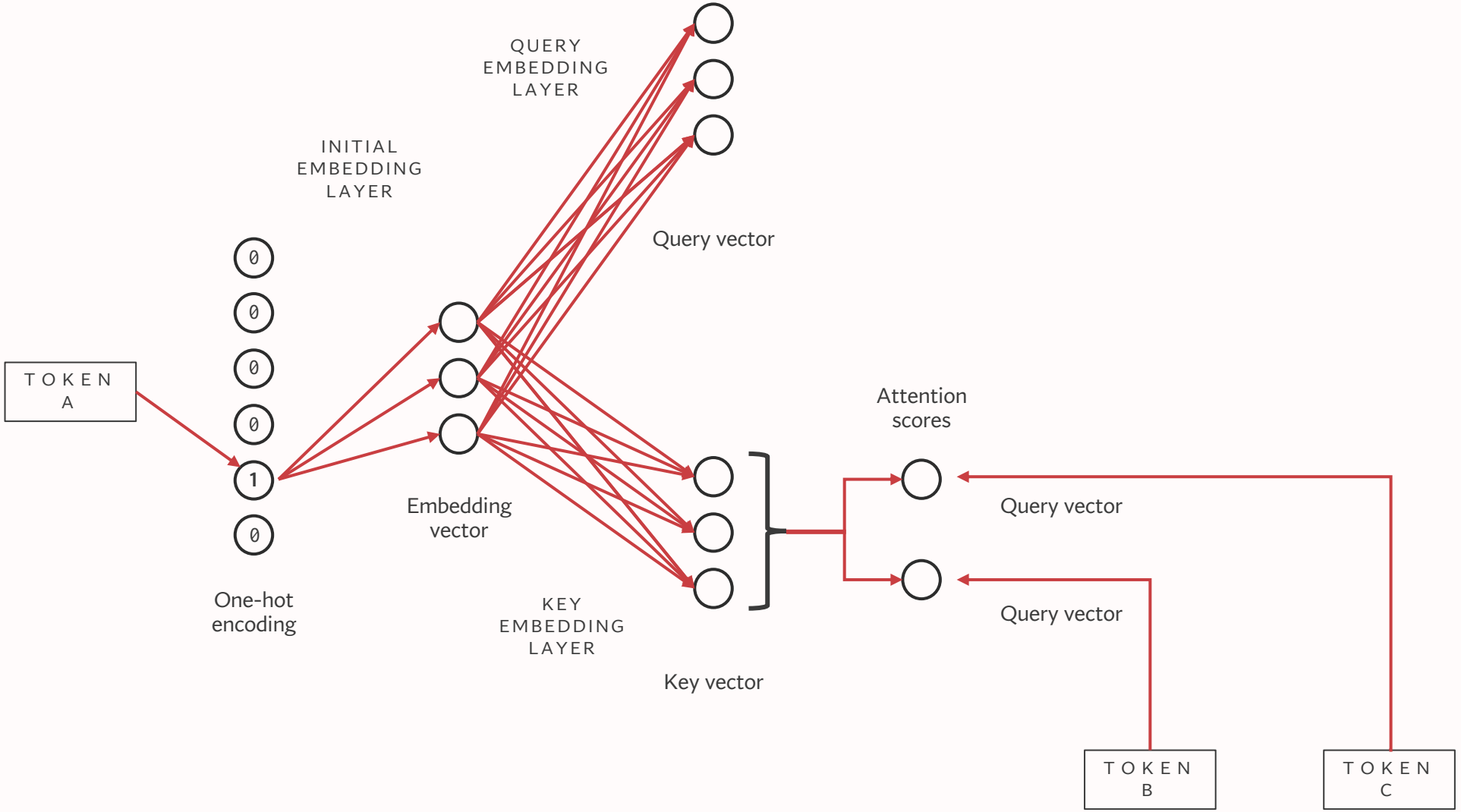
The key embedding



The key embedding



Each token produces a query and a key



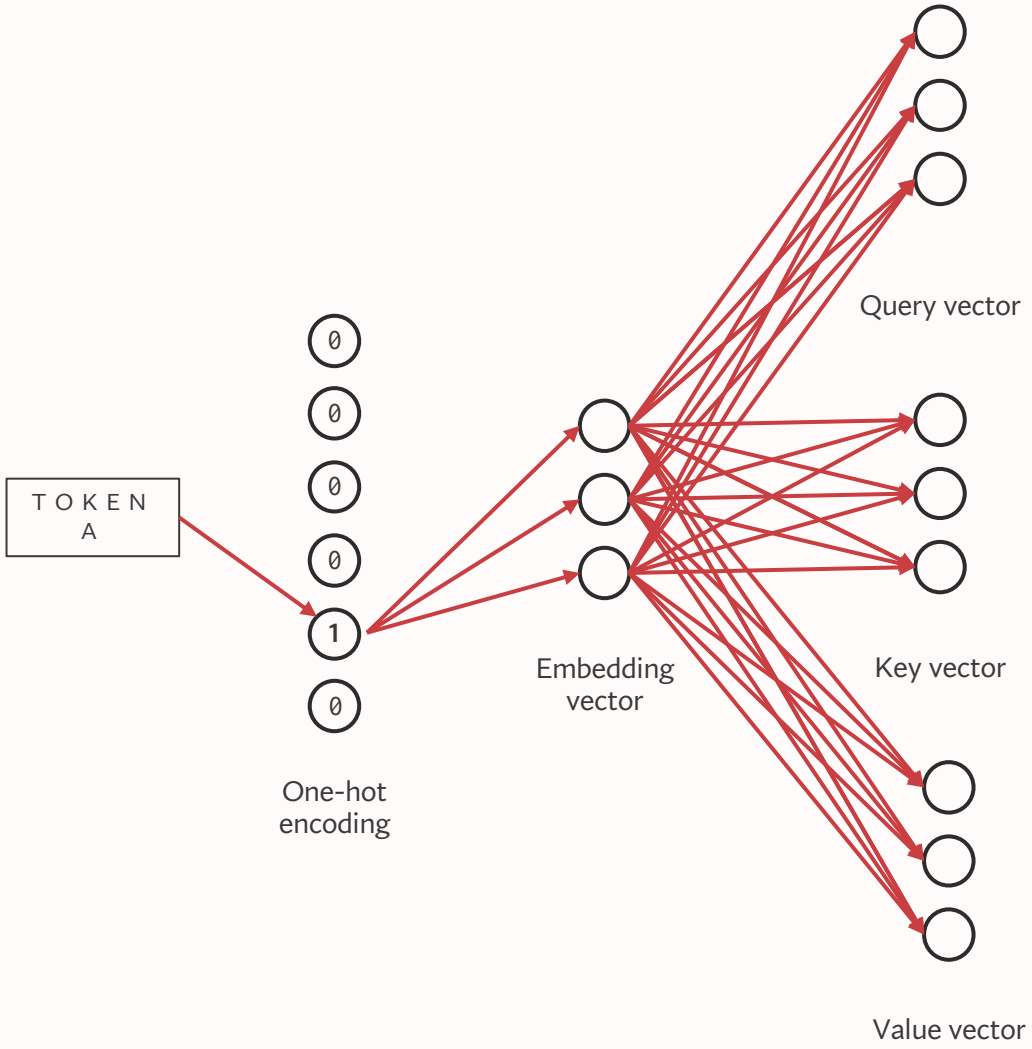
The Value Vector

The output of the attention mechanism

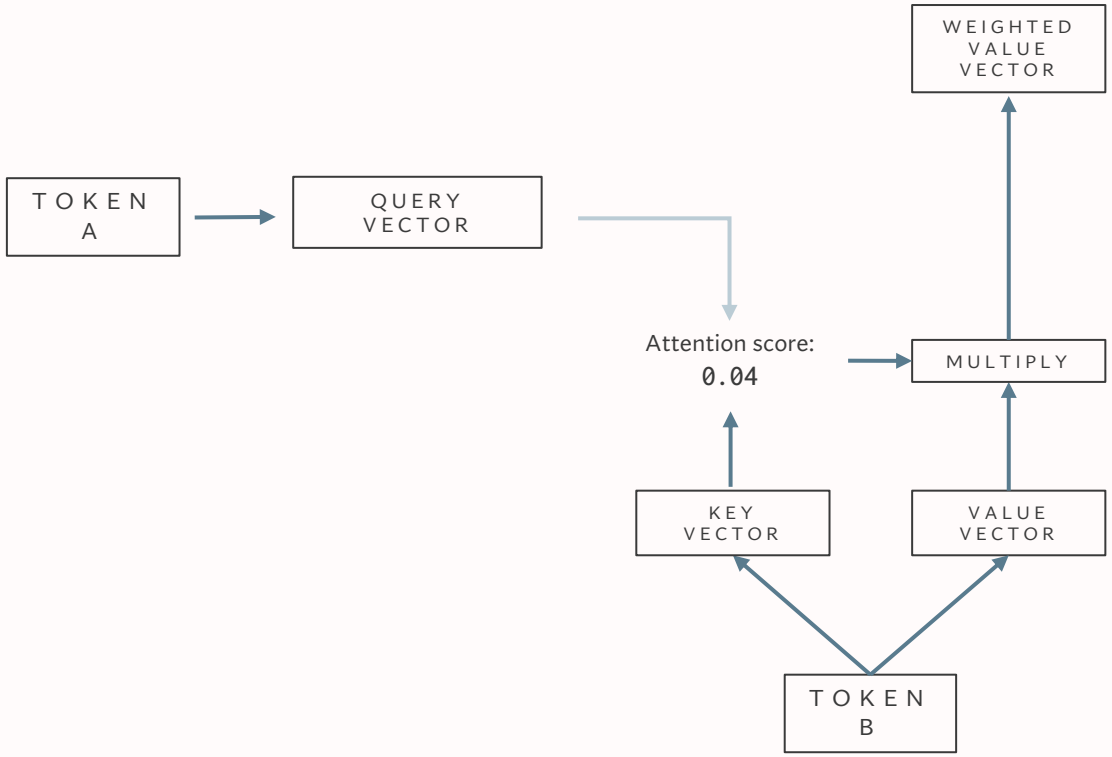
The value vector

- Attention scores indicate how *relevant* each token is to another token.
- But what do we *do* with these scores?
- The attention scores are used to weight a *third* kind of vector: the value vector.

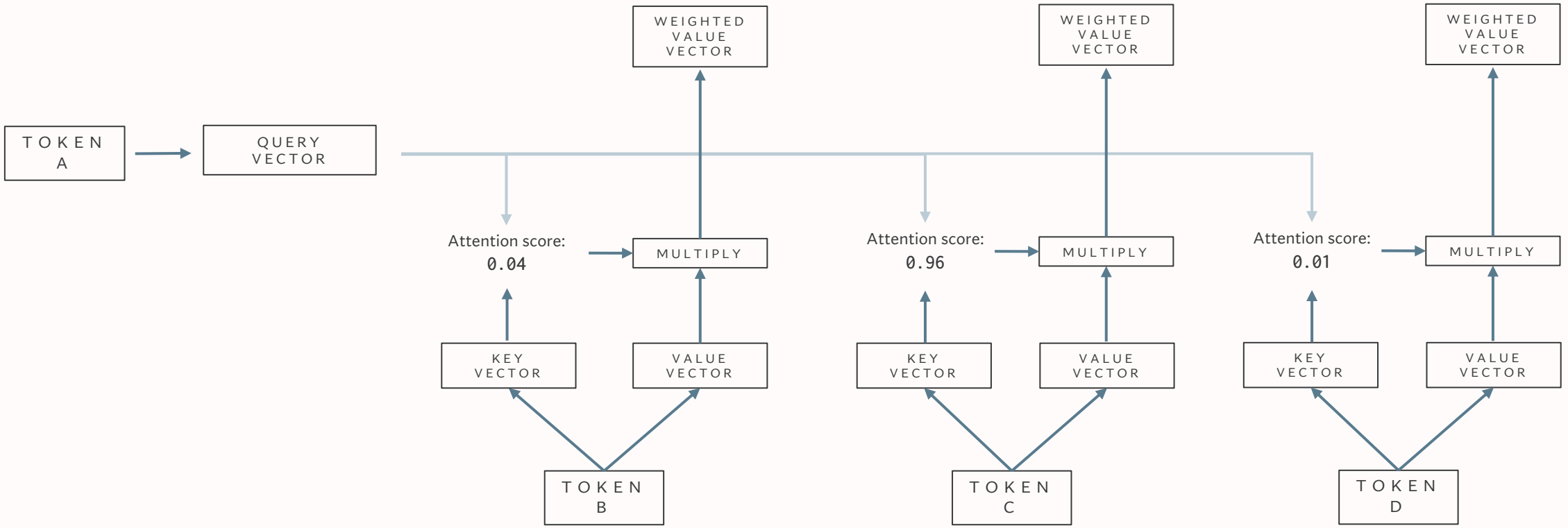
The value vector



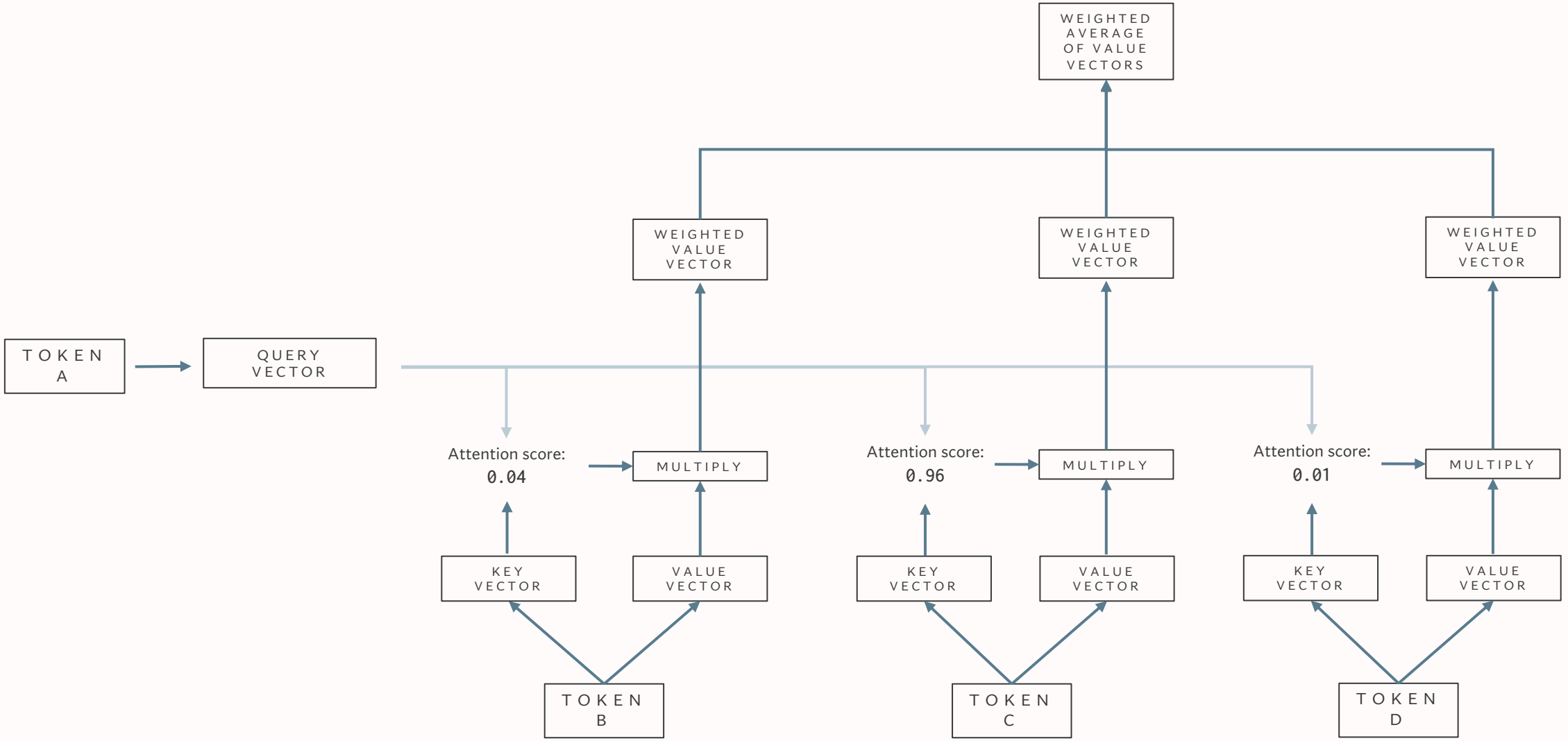
The value vector



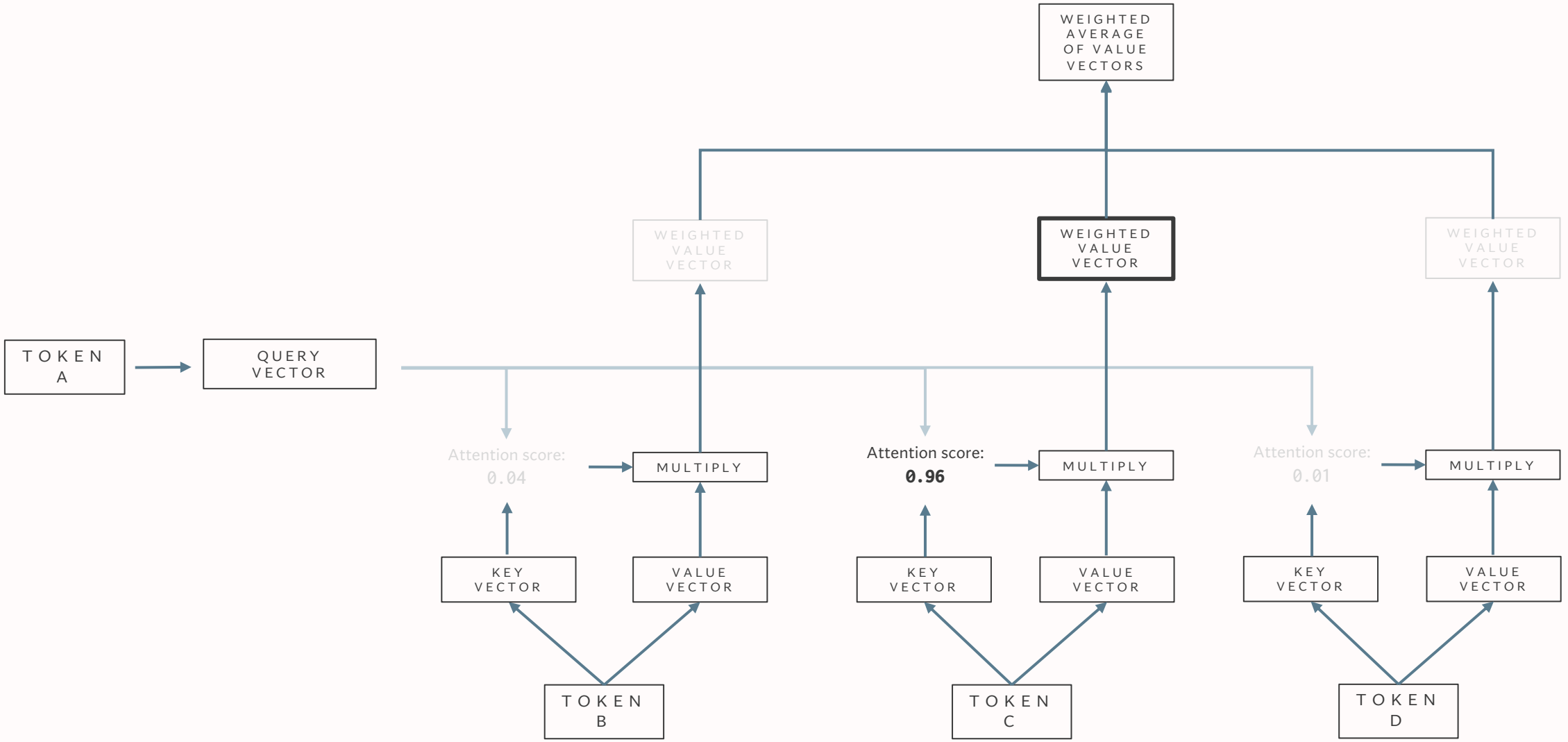
The value vector



The value vector



The value vector



Attention and the Transformer

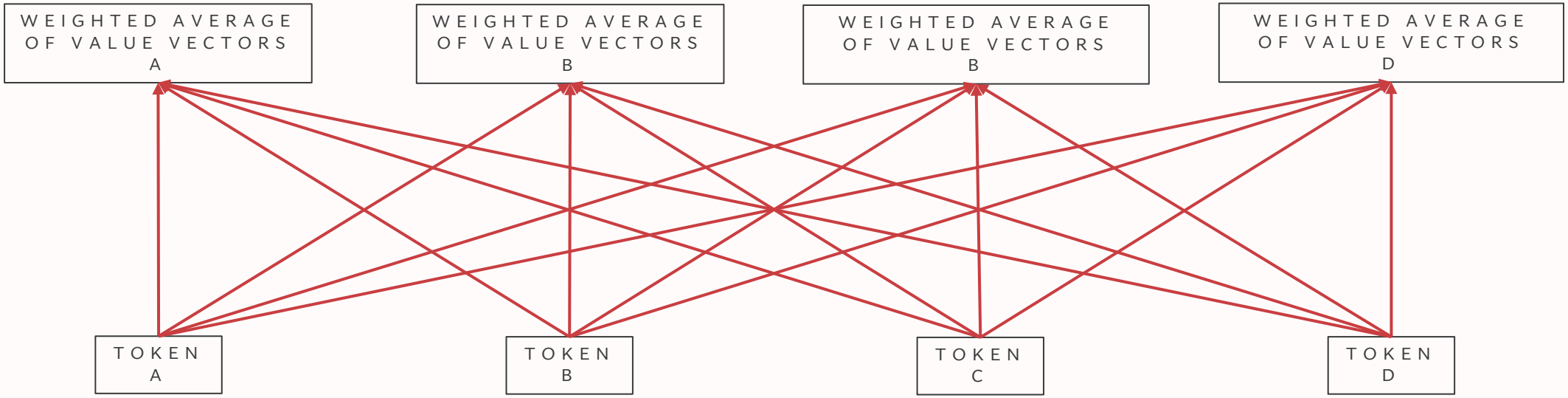
Bringing it all together

Attention overview

1. Each token computes its own *key*, *query*, and *value* vector
2. These vectors are created by the *key embedding layer*, *query embedding layer*, and *value embedding layer*, respectively
3. Each token's *query* vector is dot-producted with every token's *key* vector:
   - The query can be thought of as "asking": *what other tokens are relevant to me?*
   - The key can be thought of as "answering": *which other tokens will find me relevant?*
4. The results of each dot product form the *attention weights*.
   - High attention weight implies that the token is very relevant
5. The attention weights are then used to weight every token's *value* vector
   - More relevant tokens have their value vectors weighted more strongly
6. The output is the weighted average of all value vectors
7. This is repeated for each token

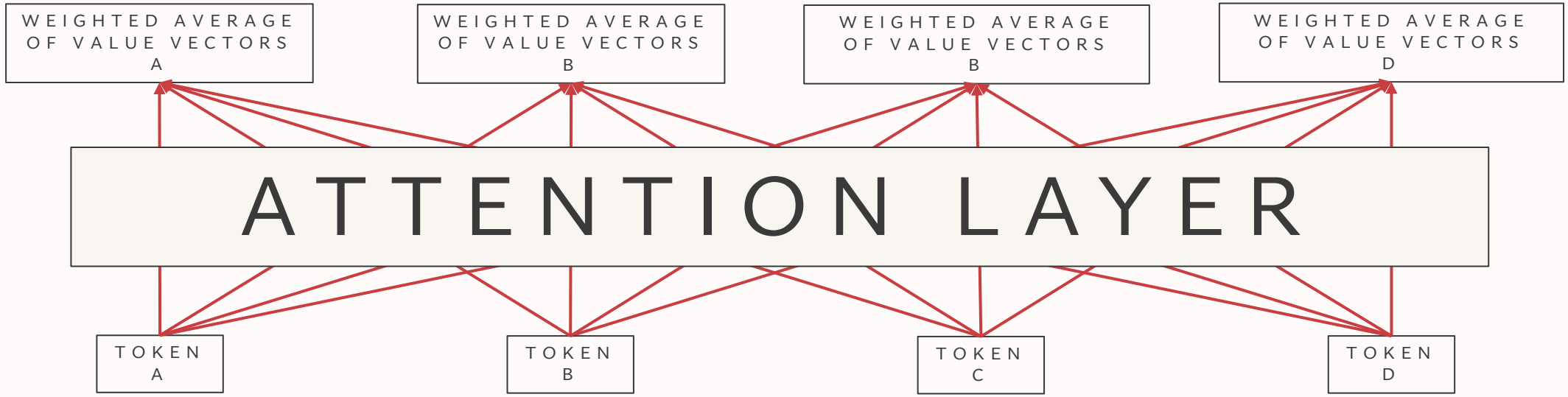
Attention overview

- Attention takes a sequence of tokens, and outputs a processed version of that sequence
- The input and output sequences have the same length
- Attention allows the tokens to *interact* with each other, where more relevant vectors can affect the corresponding output more strongly



Attention overview

- Attention takes a sequence of tokens, and outputs a processed version of that sequence
- The input and output sequences have the same length
- Attention allows the tokens to *interact* with each other, where more relevant vectors can affect the corresponding output more strongly

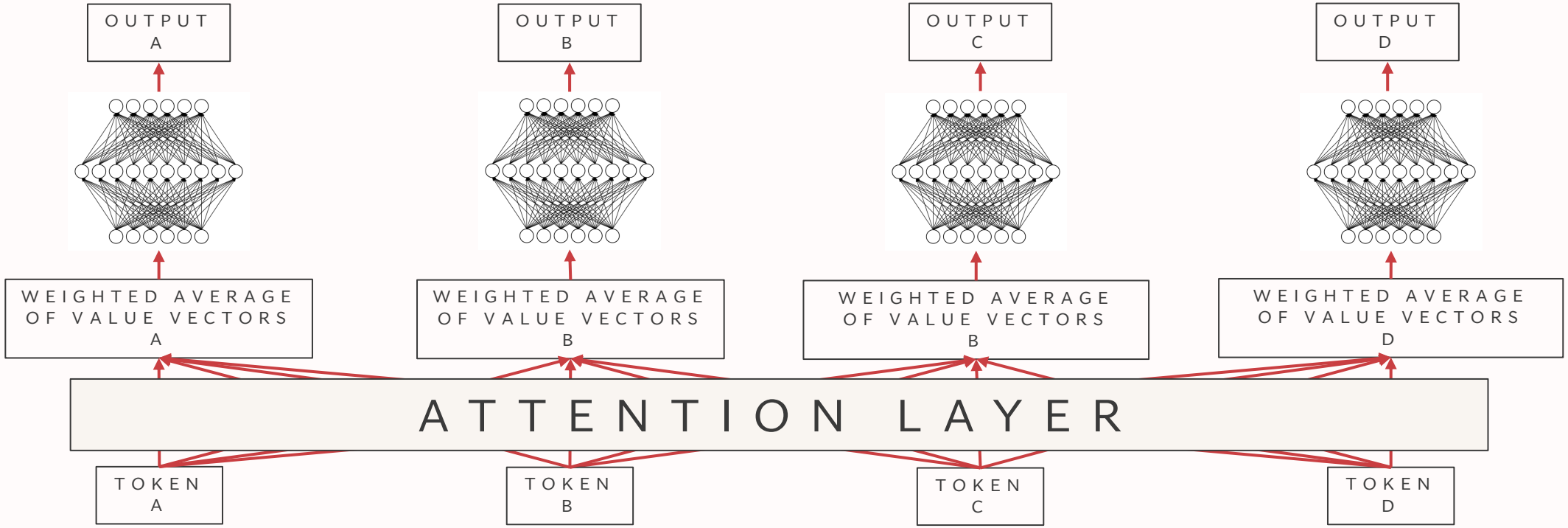


Attention parameters

- The learnable parameters of attention are the weights in the:
  - Query embedding layer
  - Key embedding layer
  - Value embedding layer
- These three sets of weights can be used to process *any* number of tokens
- *Note*: there are no ReLU functions in the attention mechanism

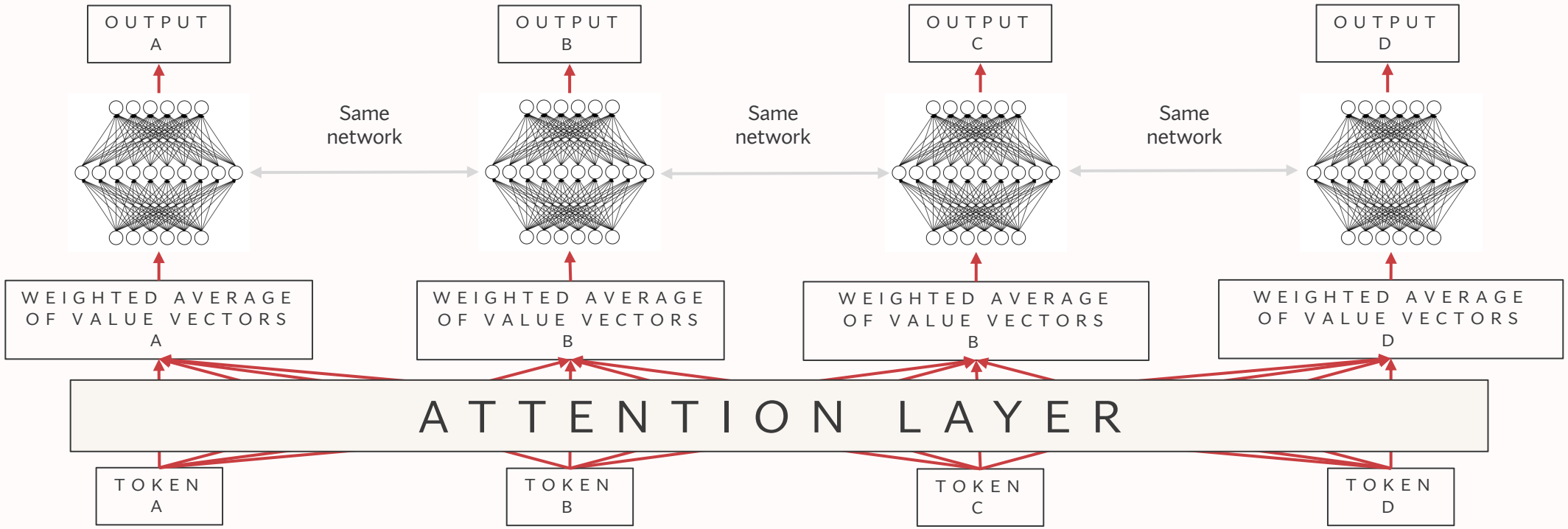
The Transformer

Transformers work by *interleaving* layers of attention and fully connected networks.



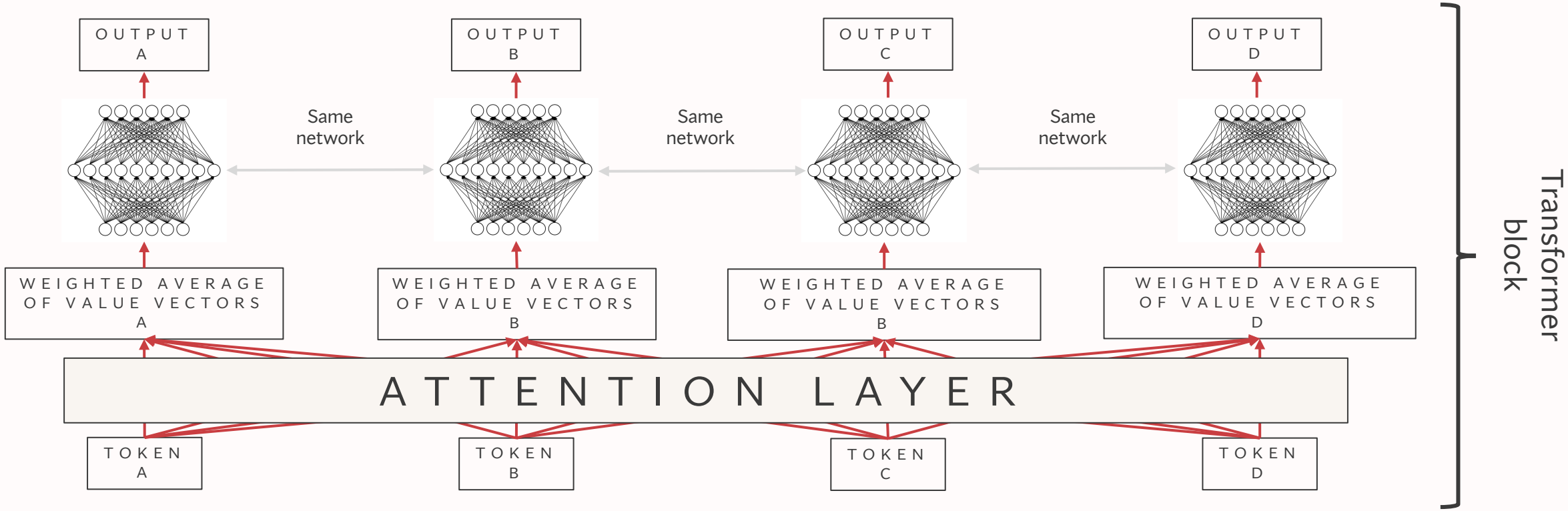
The Transformer

Transformers work by *interleaving* layers of attention and fully connected networks.



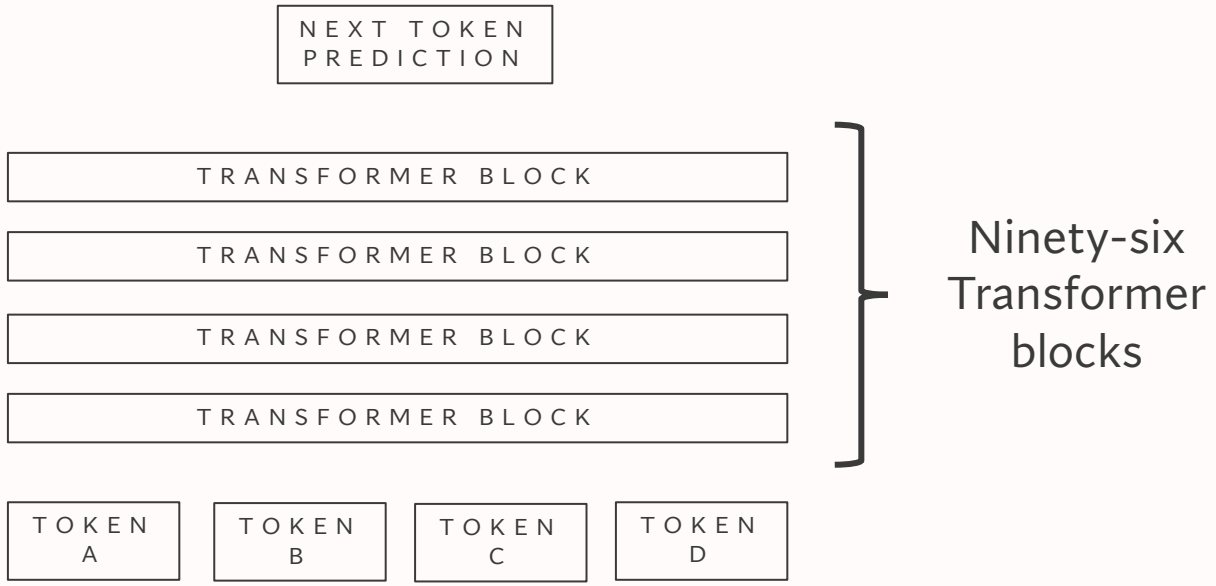
The Transformer

Transformers work by *interleaving* layers of attention and fully connected networks.



GPT-3

- One of the largest models whose details were published before the release of ChatGPT
- ChatGPT-3.5, 4, 4o, 5 and beyond are likely *much* larger



Question & Answer