Introduction to Large Language Models

Spring 2026

Attention Mechanism in Transformers

(Some slides adapted from Ralph Grishman at NYU,
Yejin Choi at UWashington, N. Tomura at UDepaul, Jurafsky and Martin, CS224N, CS224d at Stanford and other resourses on the web)

Language Models

Evolution: From Statistical to Neural Language Models

Before neural networks, language modeling relied on:

- **n-gram models**
- **Hidden Markov models**
- **Count-based statistics**

These struggled with:

- long-range dependencies
- sparse data
- poor generalization

$$P(w_i \mid w_1 \ldots w_{i-1}) \approx P(w_i \mid w_{i-n+1} \ldots w_{i-1})$$

For some small n

When $n = 1$, a unigram model

$$P(w_1, w_2, .. w_m) = \prod_{i=1}^{m} P(w_i)$$

w_i

the dog barks

When $n = 2$, a bigram model

$$P(w_1, w_2, .. w_m) = \prod_{i=1}^{m} P(w_i \mid w_{i-1})$$

w_i

the dog barks

When $n = 3$, a trigram model

$$P(w_1, w_2, .. w_m) = \prod_{i=1}^{m} P(w_i \mid w_{i-2} w_{i-1})$$

w_i

the dog barks

N-Gram Model

$$P(w_i \mid w_1 \ldots w_{i-1}) \approx P(w_i \mid w_{i-n+1} \ldots w_{i-1})$$

For some small n

When $n = 1$, a unigram model

$$P(w_1, w_2, .. w_m) = \prod_{i=1}^{m} P(w_i)$$

w_i

the dog barks

When $n = 2$, a bigram model

$$P(w_1, w_2, .. w_m) = \prod_{i=1}^{m} P(w_i \mid w_{i-1})$$

w_i

the dog barks

When $n = 3$, a trigram model

$$P(w_1, w_2, .. w_m) = \prod_{i=1}^{m} P(w_i \mid w_{i-2} w_{i-1})$$

w_i

the dog barks

Maximum Likelihood Estimatiom

For unigram models,

$$P(w_i) = \frac{C(w_i)}{M}$$

$\frac{C(\text{barks})}{M}$

For bigram models,

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1} w_i)}{C(w_{i-1})}$$

$\frac{C(\text{dog barks})}{C(\text{dog})}$

For n-gram models generally,

$$P(w_i \mid w_{i-n+1} \ldots w_{i-1}) = \frac{C(w_{i-n+1} \ldots w_i)}{C(w_{i-n+1} \ldots w_{i-1})}$$

Neural Language Models

A **Neural Language Model (NLM)** is a machine learning model that uses neural networks to understand and generate human language. Their goal is to model the **probability distribution of text**, meaning:

Given some words, predict the next word (or the probability of words).

Example:
Input: *"The cat sat on the"*
Model predicts: *"mat"* with high probability.

Types of Neural Language Models

- Feed-Forward Neural Language Models

The earliest NLMs (e.g., Bengio et al., 2003).

Pipeline:

- Input words → **embeddings**
- Pass through neural layers
- Predict next word

Limitation: Fixed context window (like n-grams).

- Recurrent Neural Network (RNN) Language Models

Examples: **RNNs, LSTMs, GRUs**

These read text **sequentially**, maintaining a hidden state:

$$h_t = \text{RNN}(x_t, h_{(t-1)})$$

Pros:

- Handle longer dependencies than feed-forward models

Cons:

- Slow training
- Difficulty with very long sequences
- Vanishing gradient problems

Types of Neural Language Models

-Transformers (Modern Neural Language Models)
Introduced in *"Attention Is All You Need"* (2017).
Transformers use **self-attention**, not recurrence:

- Process all tokens in parallel
- Capture long-distance relationships easily
- Scale to billions of parameters

-Neural Language Models vs. LLMs

- All **Large Language Models (LLMs)** are Neural Language Models — but at extreme scale.

Model Type	Examples	Notes
NLMs	RNN-LM, LSTM-LM	Early models
Transformers	BERT, GPT-2	Modern architecture
LLMs	GPT-4/5, LLaMA, Claude	Massive-scale transformers

Introduction to Attention

LLMs are built out of transformers

- Transformer: a specific kind of network architecture, like a fancier feedforward network, but based on attention

---

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

A very approximate timeline

- 1990 Static Word Embeddings
- 2003 Neural Language Model
- 2008 Multi-Task Learning
- 2015 Attention
- 2017 Transformer
- 2018 Contextual Word Embeddings and Pretraining
- 2019 Prompting

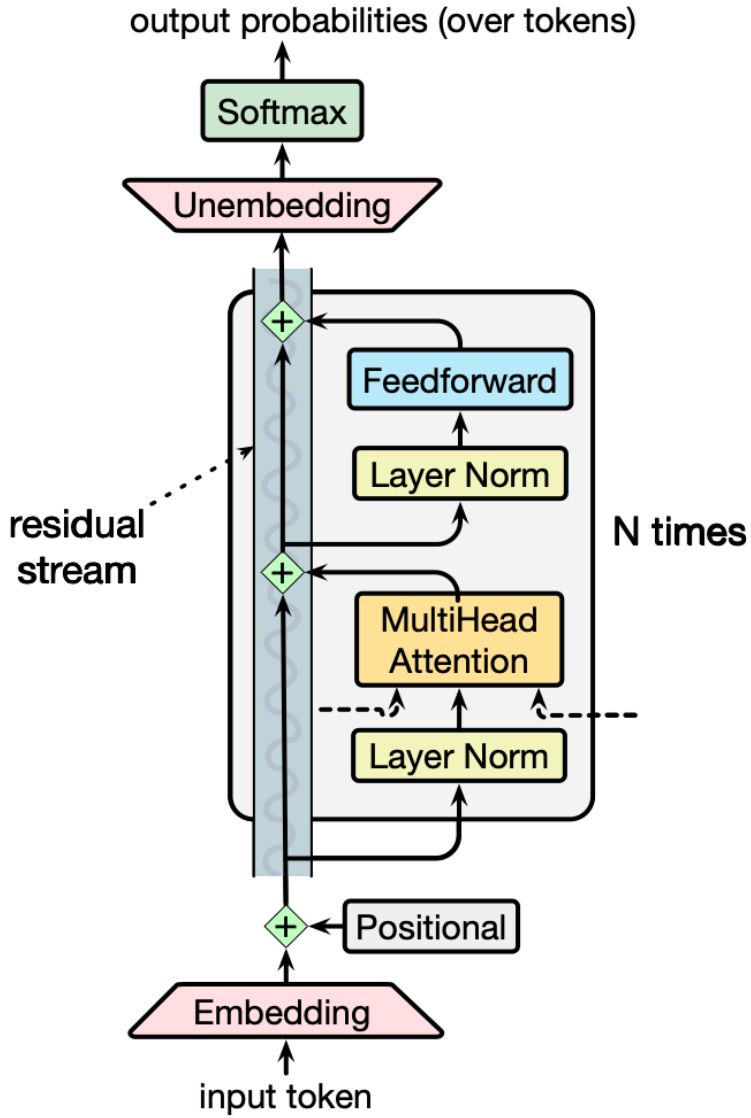
The transformer

- A **Transformer** is a neural network architecture introduced in the 2017 paper **"Attention Is All You Need"** (Vaswani et al.).

- It was revolutionary because unlike RNNs or LSTMs, it **does not process text sequentially**. Instead, it processes all tokens at once using a mechanism called **Self-Attention**.

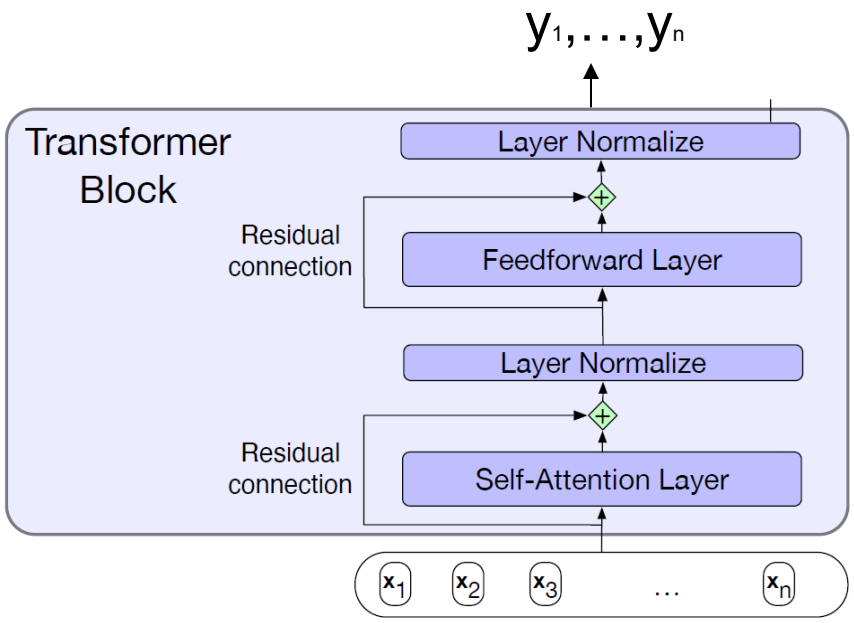
This makes transformers:

- Faster to train (parallelizable)
- Better at capturing long-range dependencies
- More scalable to huge datasets



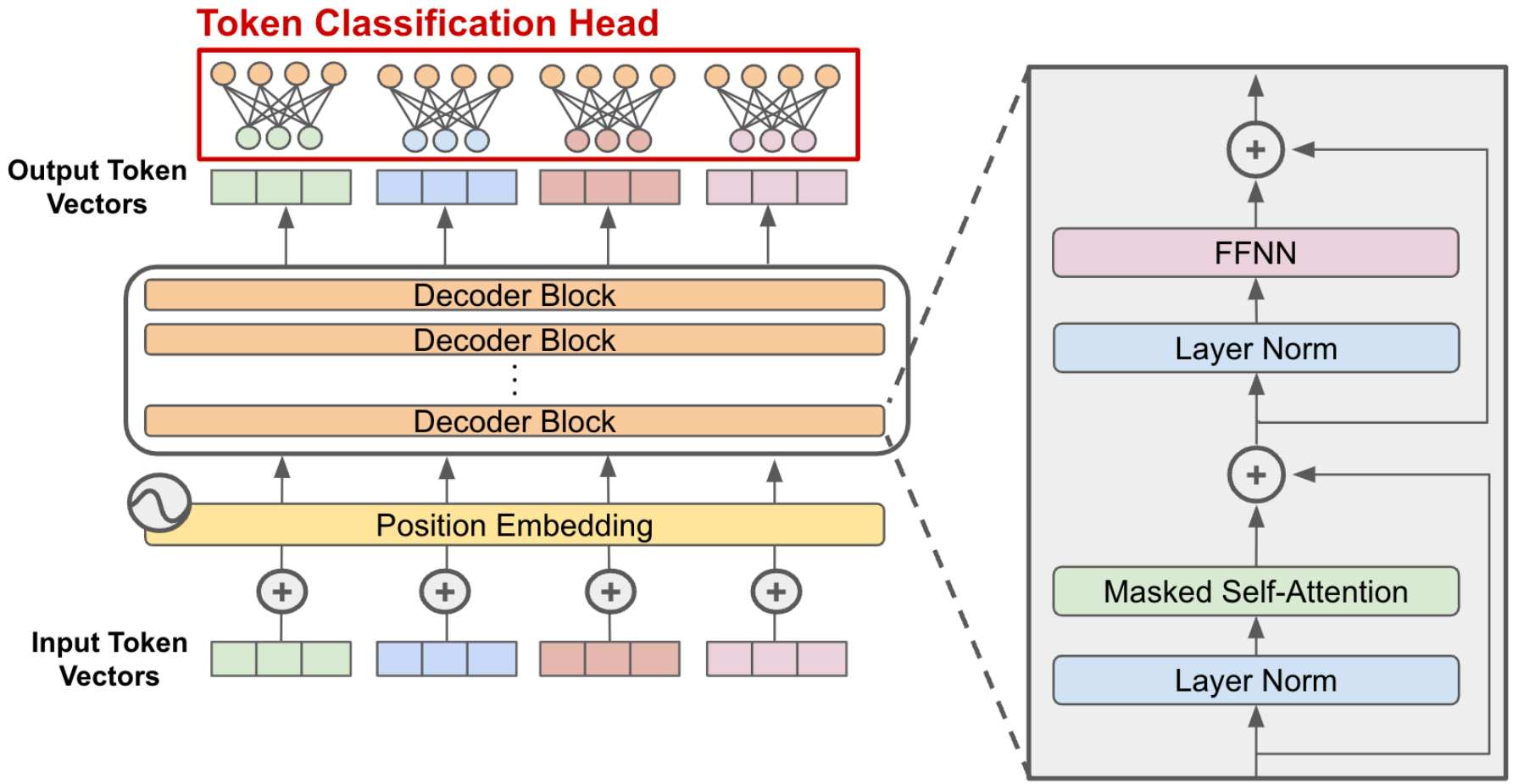
Transformers

- Transformers map sequences of input vectors (x_1, ..., x_n) to sequences of output vectors (y_1, ..., y_n) of the same length.
- A Transformer block/layer takes a sequence of token embeddings and produces a sequence of contextualized embeddings of the same length. Each output vector corresponds to one input token but enriched with information from the entire sequence.
- A Transformer is a multilayer network made by combining simple linear layers, feedforward networks, and **self-attention layers**, the key innovation of transformers.



Transformers

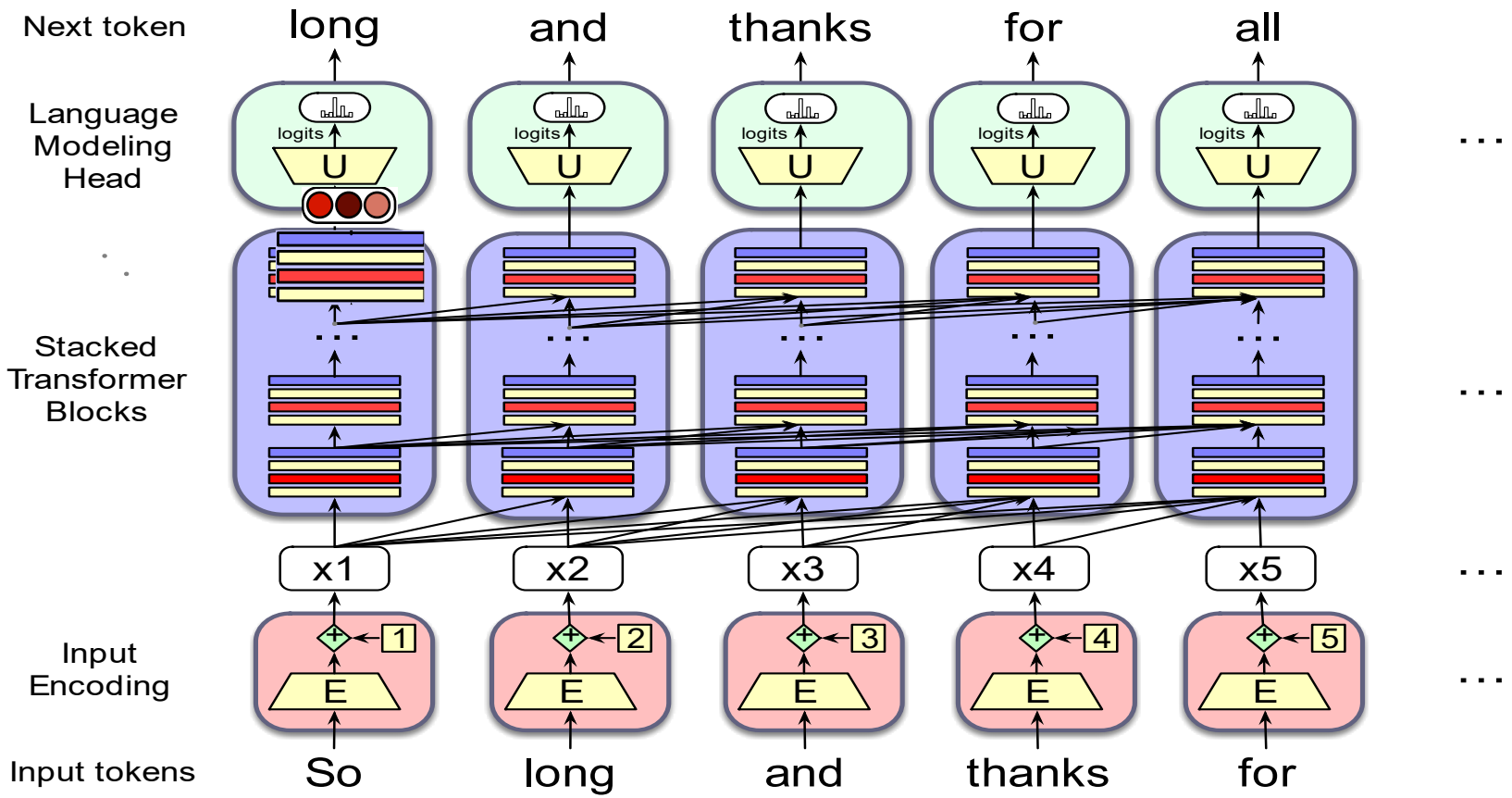
Transformers are made up of stacks of transformer blocks



Source: https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

Instead of starting with the big picture

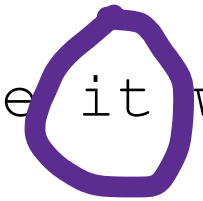
- The architecture of a (left-to-right) transformer, showing how each input token get encoded, passed through a set of stacked transformer blocks, and then a language model head that predicts the next token.
- The embeddings at each token position in the residual stream are passed up the stack, and the arrows in the figure shows how information from the hidden representations of preceding tokens are also incorporated.



Problem with static embeddings (word2vec)

•They are static! The embedding for a word doesn't reflect how its meaning changes in context.

• The chicken didn't cross the road because it was too tired



• What is the meaning represented in the static embedding for "it"?

•

Contextual Embeddings

- Intuition: a representation of meaning of a word should be different in different contexts!
- **Contextual Embedding**: each word has a different vector that expresses different meanings depending on the surrounding words
- How to compute contextual embeddings?
  - **Attention**

Contextual Embeddings

- `The chicken didn't cross the road because it`

- What should be the properties of "it"?

- `The chicken didn't cross the road because it was too` **`tired`**
- `The chicken didn't cross the road because it was too` **`wide`**

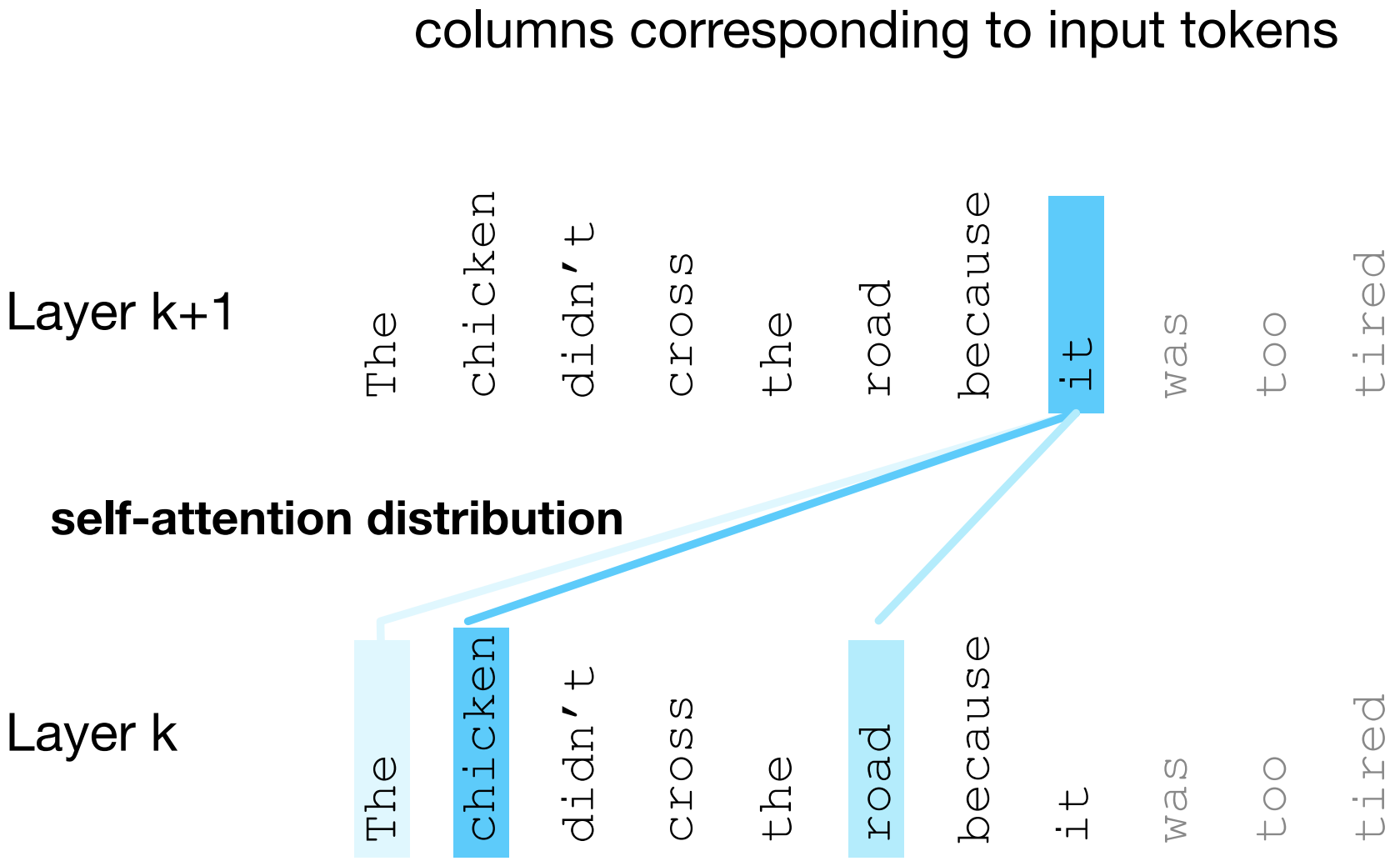
- At this point in the sentence, it's probably referring to either the chicken or the street

-

Intuition of attention

- Build up the contextual embedding from a word by selectively integrating information from all the neighboring words
- We say that a word "attends to" some neighboring words more than others

Intuition of attention:



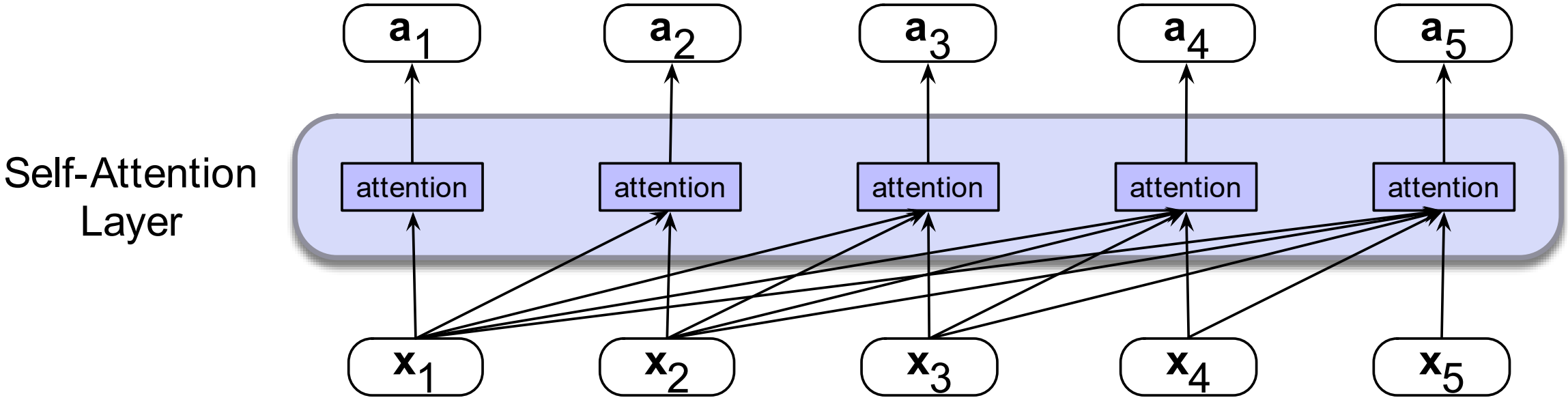
Attention definition

- A mechanism for helping compute the embedding for a token by selectively attending to and integrating information from surrounding tokens (at the previous layer).

- More formally: a method for doing a weighted sum of vectors.

Attention is left-to-right

- Fig. 10.1 illustrates the flow of information in a single causal, or backward looking, self-attention layer. As with the overall transformer, a self-attention layer maps input sequences ($x_1,\ldots,x_n$) to output sequences of the same length ($a_1,\ldots,a_n$).
- When processing each item in the input, the model has access to all of the inputs up to and including the one under consideration, but no access to information about inputs beyond the current one.



Self-Attention Layer

- The computation performed for each item is independent of all the other computations.
- The first point ensures that we can use this approach to create language models and use them for autoregressive generation, and
- The second point means that we can easily parallelize both forward inference and training of such models.
- At the core of an attention-based approach is the ability to compare an item of interest to a collection of other items in a way that reveals their relevance in the current context.

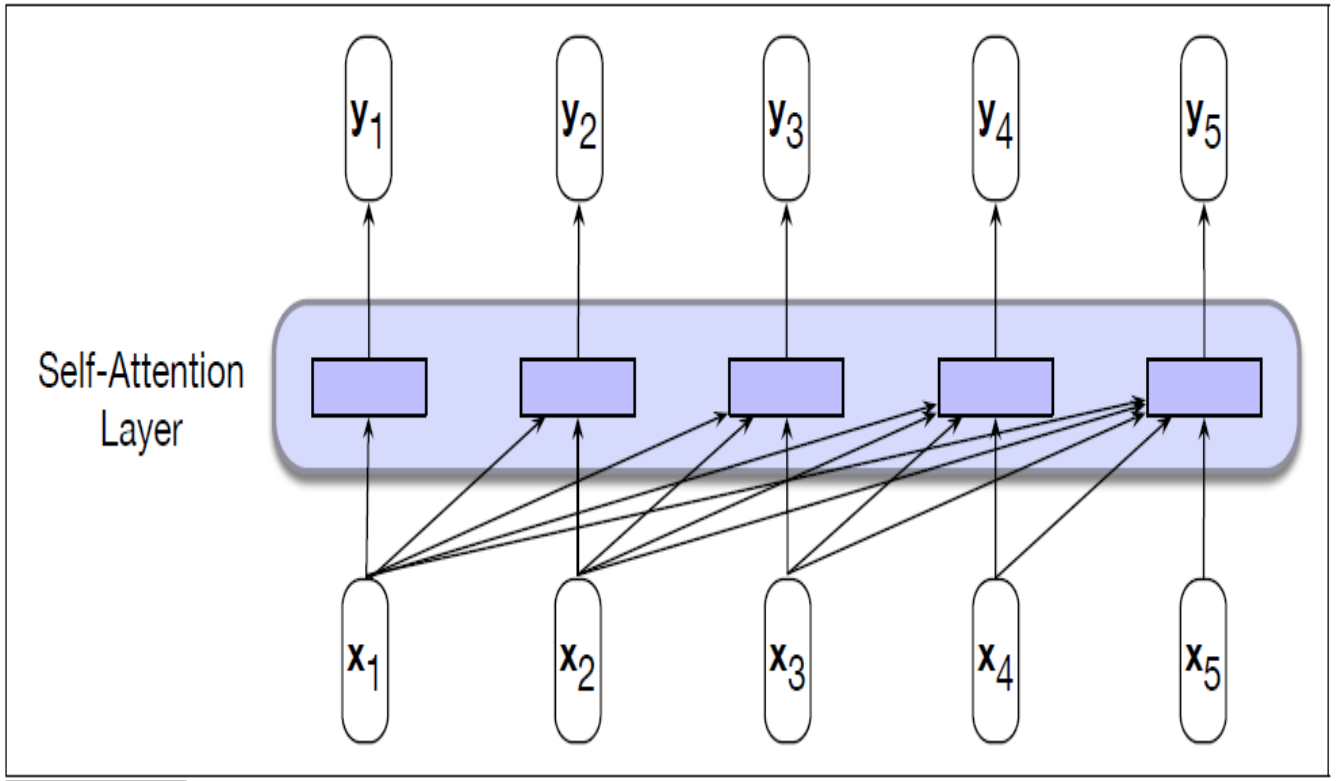


Figure 10.1 Information flow in a causal (or masked) self-attention model. In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one. Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

Self-Attention Layer

- In the case of self-attention, the set of comparisons are to other elements within a given sequence. The result of these comparisons is then used to compute an output for the current input.
- The computation of y_3 is based on a set of comparisons between the input x_3 and its preceding elements x_1 and x_2, and to x_3 itself. The simplest form of comparison between elements in a self-attention layer is a dot product.
- Let's refer to the result of this comparison as a score

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

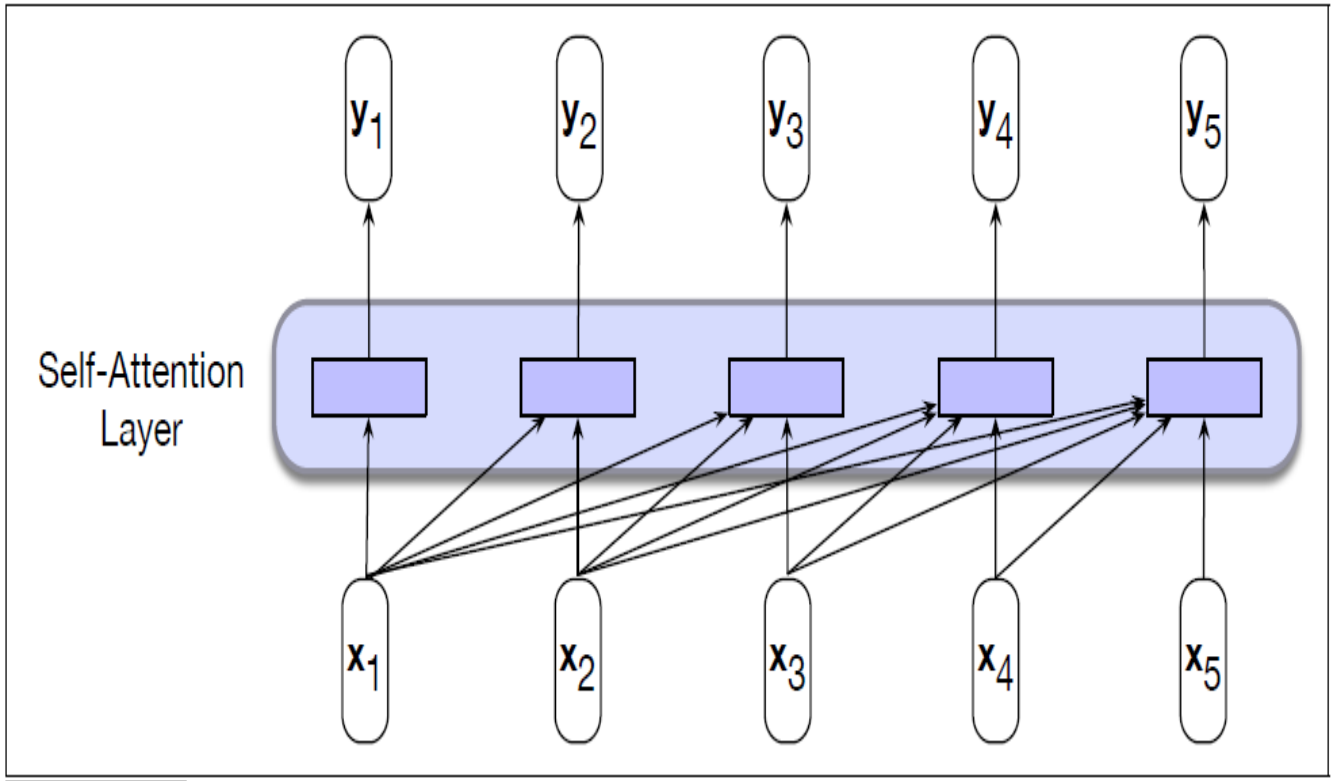


Figure 10.1 Information flow in a causal (or masked) self-attention model. In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one. Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

Self-Attention Layer

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

- The result of a dot product is a scalar value ranging from - ∞ to ∞. Then to make effective use of these scores, we'll normalize them with a softmax:

$$\begin{aligned} \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\; \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \;\; \forall j \leq i \end{aligned}$$

- Generate an output value y_i by taking the sum of the inputs seen so far, weighted by their respective a value.

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

- The steps embodied in Equations represent the core of an attention-based approach: a set of comparisons to relevant items in some context, a normalization of those scores to provide a probability distribution, followed by a weighted sum using this distribution. The output y is the result of this straightforward computation over the inputs.

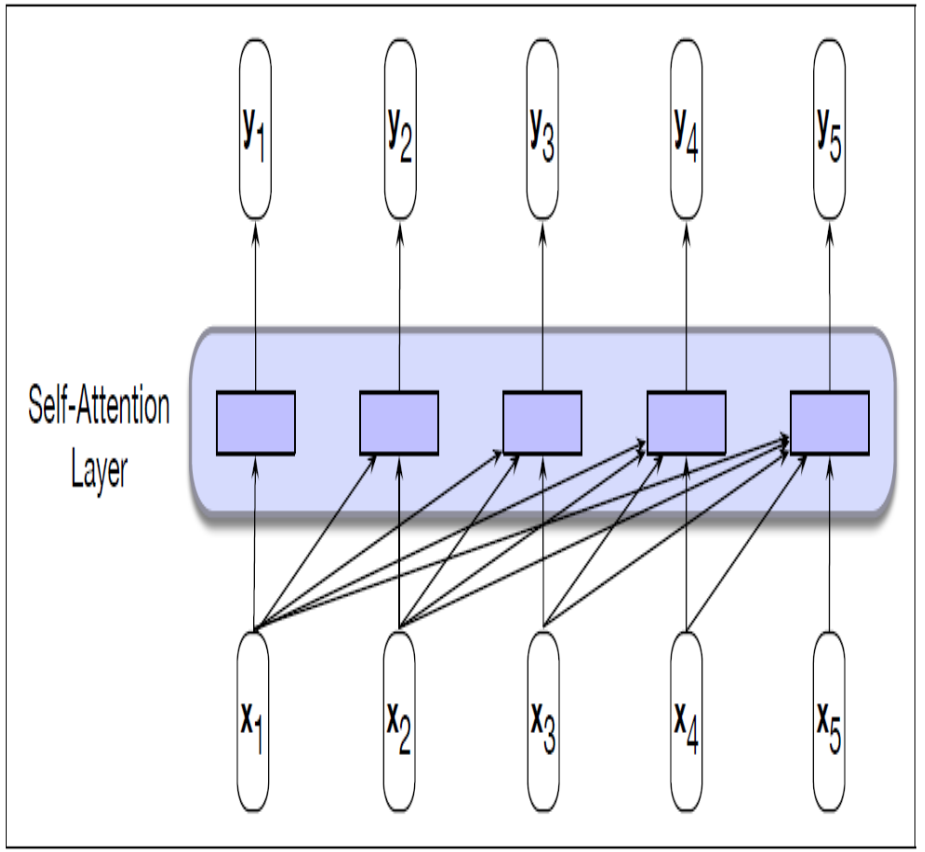
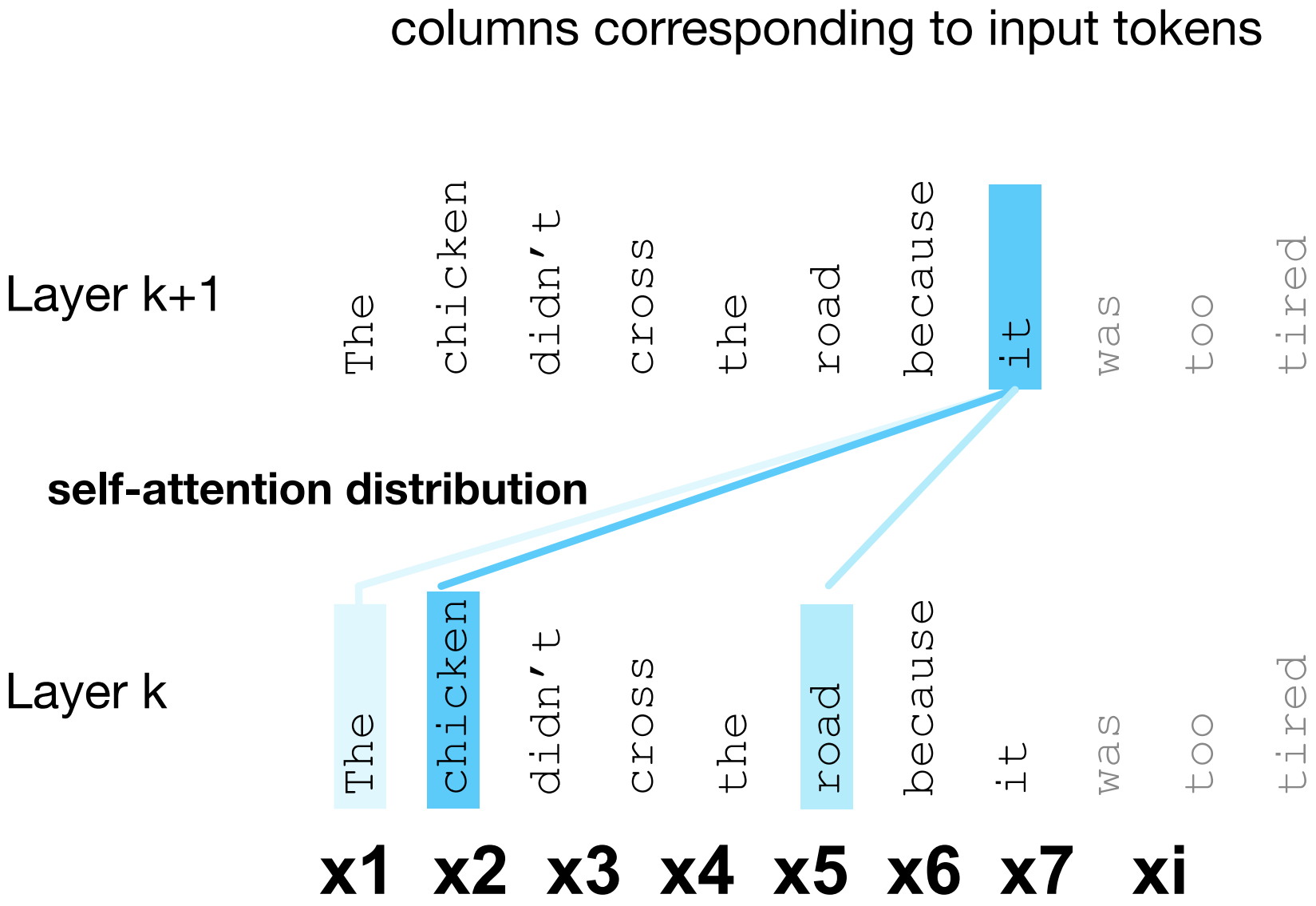


Figure 10.1 Information flow in a causal (or masked) self-attention model. In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one. Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

Intuition of attention:

- The self-attention weight distribution α that is part of the computation of the representation for the word it at layer k+1.
- In computing the representation for it, we attend differently to the various words at layer k, with darker shades indicating higher self-attention values.
- Note that the transformer is attending highly to the columns corresponding to the tokens chicken and road , a sensible result, since at the point where it occurs, it could plausibly corefer with the chicken or the road, and hence we'd like the representation for it to draw on the representation for these earlier words



Uses a more sophisticated way of represention

- Transformers allow us to create a more sophisticated way of representing how words can contribute to the representation of longer inputs. Consider the three different roles that each input embedding plays during the course of the attention process.

query
- As *the current focus of attention* when being compared to all of the other preceding inputs. We'll refer to this role as a **query**.

key
- In its role as *a preceding input* being compared to the current focus of attention. We'll refer to this role as a **key**.

value
- And finally, as a **value** used to compute the output for the current focus of attention.

To capture these three different roles, transformers introduce weight matrices $\mathbf{W}^Q$, $\mathbf{W}^K$, and $\mathbf{W}^V$. These weights will be used to project each input vector $\mathbf{x}_i$ into a representation of its role as a key, query, or value.

$$\mathbf{q}_i = \mathbf{W}^Q\mathbf{x}_i; \quad \mathbf{k}_i = \mathbf{W}^K\mathbf{x}_i; \quad \mathbf{v}_i = \mathbf{W}^V\mathbf{x}_i \tag{10.5}$$

Self-Attention Calculation

Given these projections, the score between a current focus of attention, $\mathbf{x}_i$, and an element in the preceding context, $\mathbf{x}_j$, consists of a dot product between its query vector $\mathbf{q}_i$ and the preceding element's key vectors $\mathbf{k}_j$. This dot product has the right shape since both the query and the key are of dimensionality $1 \times d$. Let's update our previous comparison calculation to reflect this, replacing Eq. 10.1 with Eq. 10.6:

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j \tag{10.6}$$

The ensuing softmax calculation resulting in $\alpha_{i,j}$ remains the same, but the output calculation for $\mathbf{y}_i$ is now based on a weighted sum over the value vectors $\mathbf{v}$.

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j \tag{10.7}$$

Example

Self-Attention Hypothetical Example

1. Each word is turned into three vectors: q, k, v

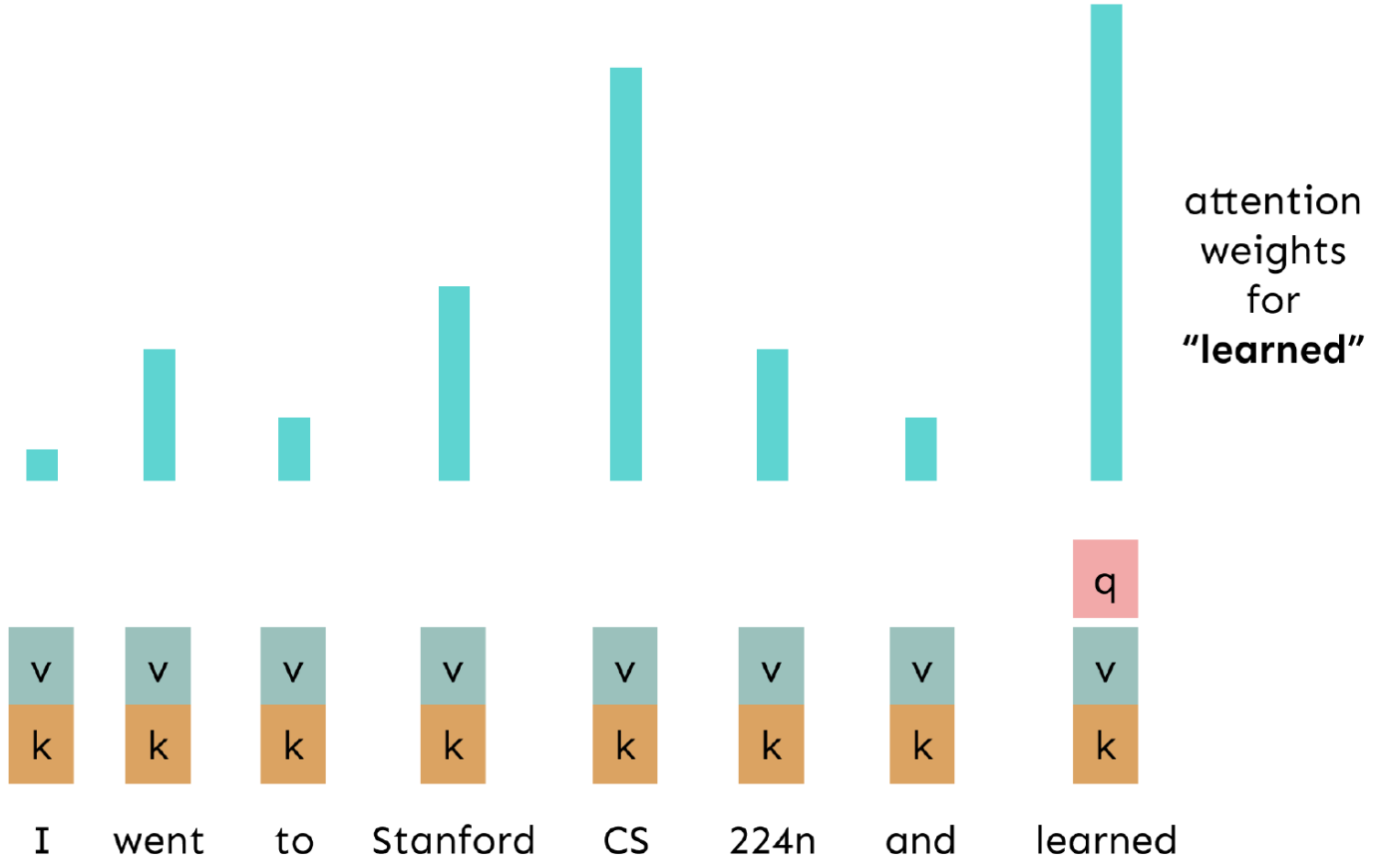
But for *self-attention*, **each position uses its own q only when we compute its output**. In the image:

- Only **"learned"** has its **q** highlighted because we are computing attention *for that single word*.

2. "learned" asks: "Who should I pay attention to?"

Self-attention computes **similarity(q, k)** between "learned" and every other word.

- If q · k is large → strong attention
- If q · k is small → weak attention



https://web.stanford.edu/class/cs224n/

Self-Attention Example

3. The tall turquoise bars represent the attention weights

Each bar above a word shows **how much "learned" attends to that word**.

In the image:

- Some bars are short → "learned" doesn't care much about those words
- Some bars are tall → "learned" finds those words semantically relevant

For example:

- A tall bar over **"Stanford"** or **"CS"** would mean:

  "learned" finds these words helpful for understanding context

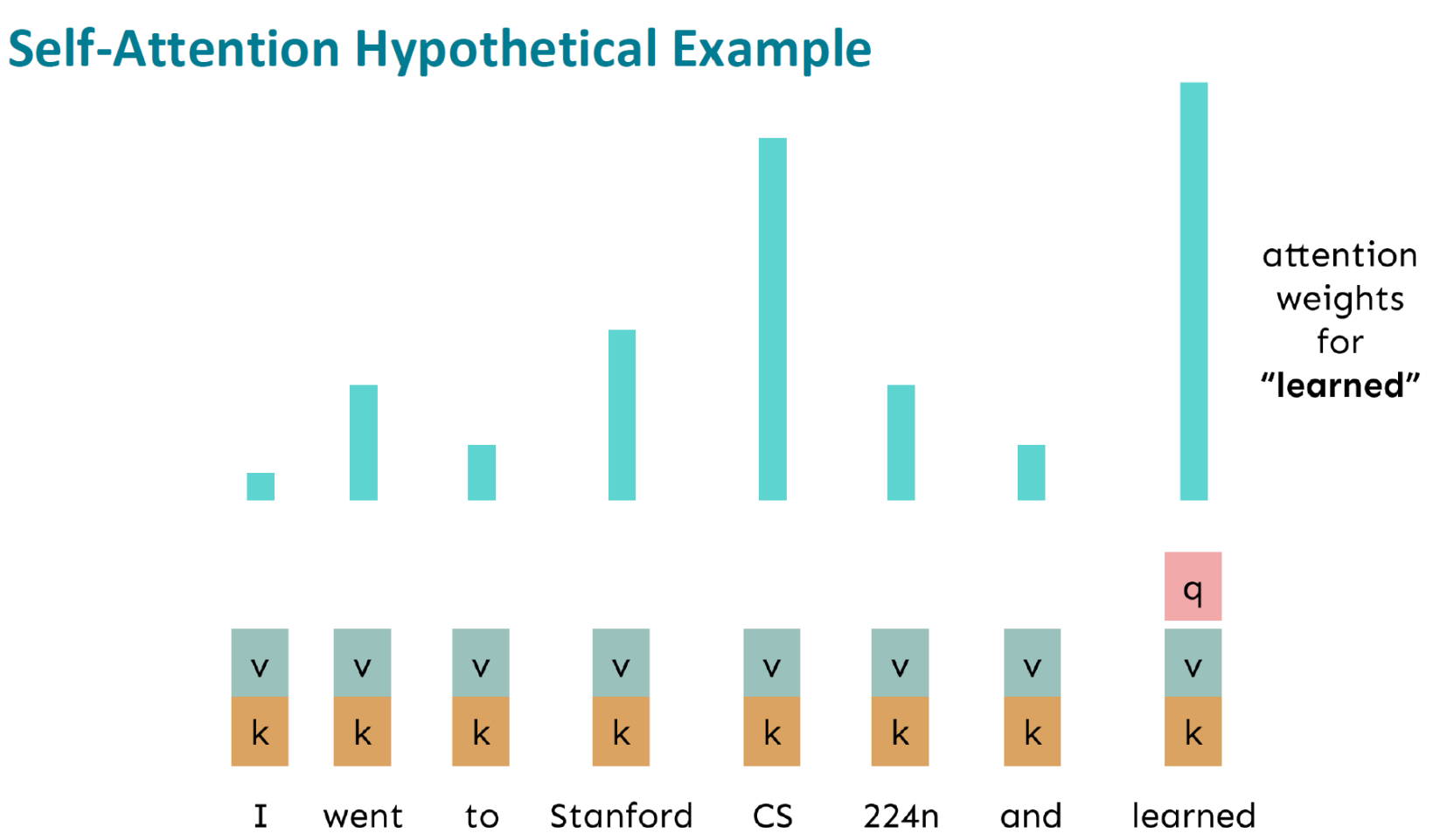
- A very tall bar over **"learned" itself** is common—many models attend heavily to the current token

4. The output for "learned" is a weighted sum of all the value vectors

The attention weights (the tall bars) multiply the **v** vectors:

- Strong attention → v gets added with high weight
- Weak attention → v gets added with low weight

The result becomes the new representation of "learned."



https://web.stanford.edu/class/cs224n/

An Actual Attention Head: slightly more complicated

- We'll use matrices to project each vector $\mathbf{x}_i$ into a representation of its role as query, key, value:
  - **query**: $\mathbf{W}^Q$
  - **key**: $\mathbf{W}^K$
  - **value**: $\mathbf{W}^V$

$$\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^Q; \quad \mathbf{k}_i = \mathbf{x}_i\mathbf{W}^K; \quad \mathbf{v}_i = \mathbf{x}_i\mathbf{W}^V$$

- To compute similarity of current element $\mathbf{x}_i$ with some prior element $\mathbf{x}_j$
- We'll use dot product between $\mathbf{q}_i$ and $\mathbf{k}_j$.
- And instead of summing up $\mathbf{x}_j$, we'll sum up $\mathbf{v}_j$

Final equations for one attention head

$$\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{Q}}; \quad \mathbf{k}_j = \mathbf{x}_j\mathbf{W}^{\mathbf{K}}; \quad \mathbf{v}_j = \mathbf{x}_j\mathbf{W}^{\mathbf{V}}$$

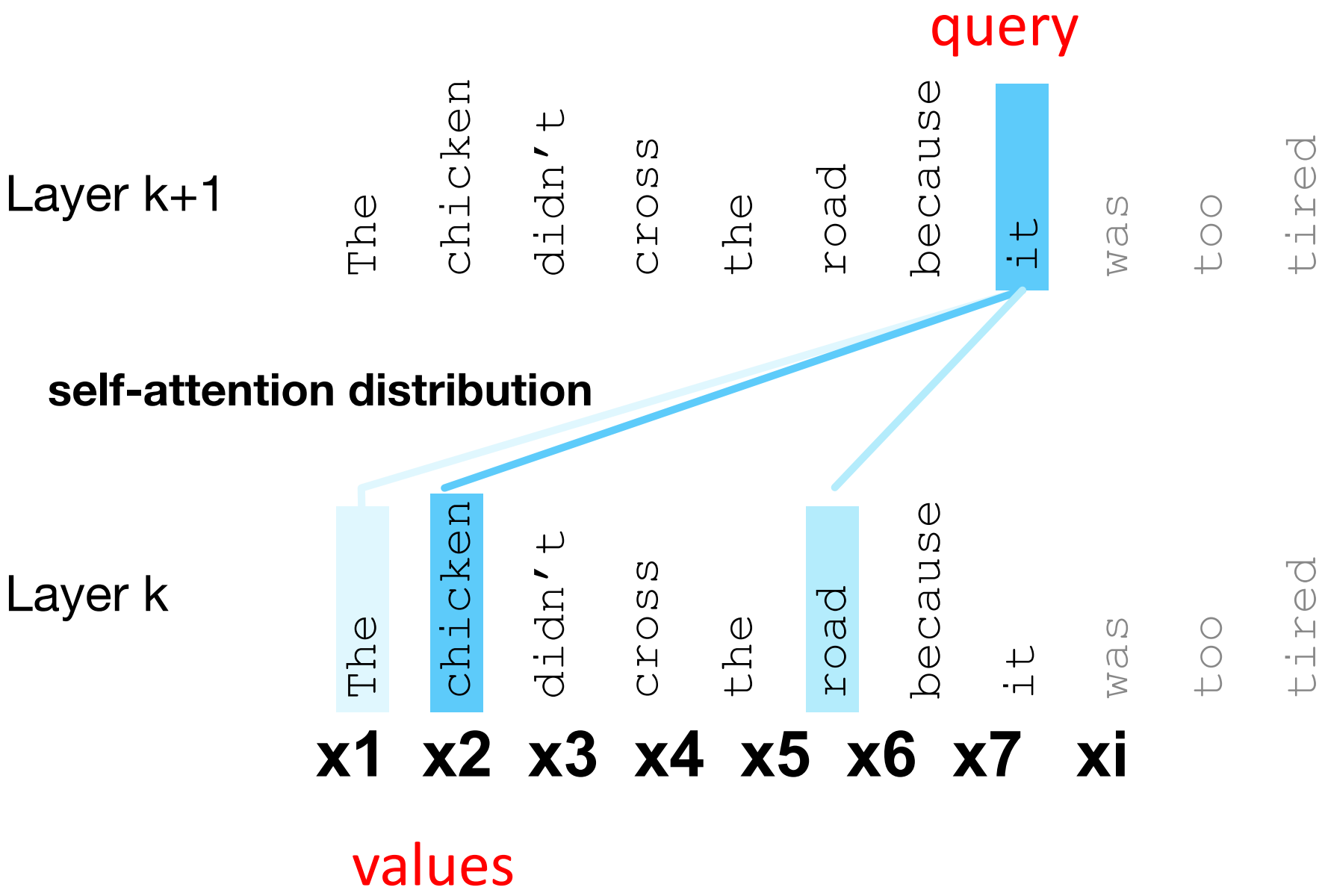
$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i$$

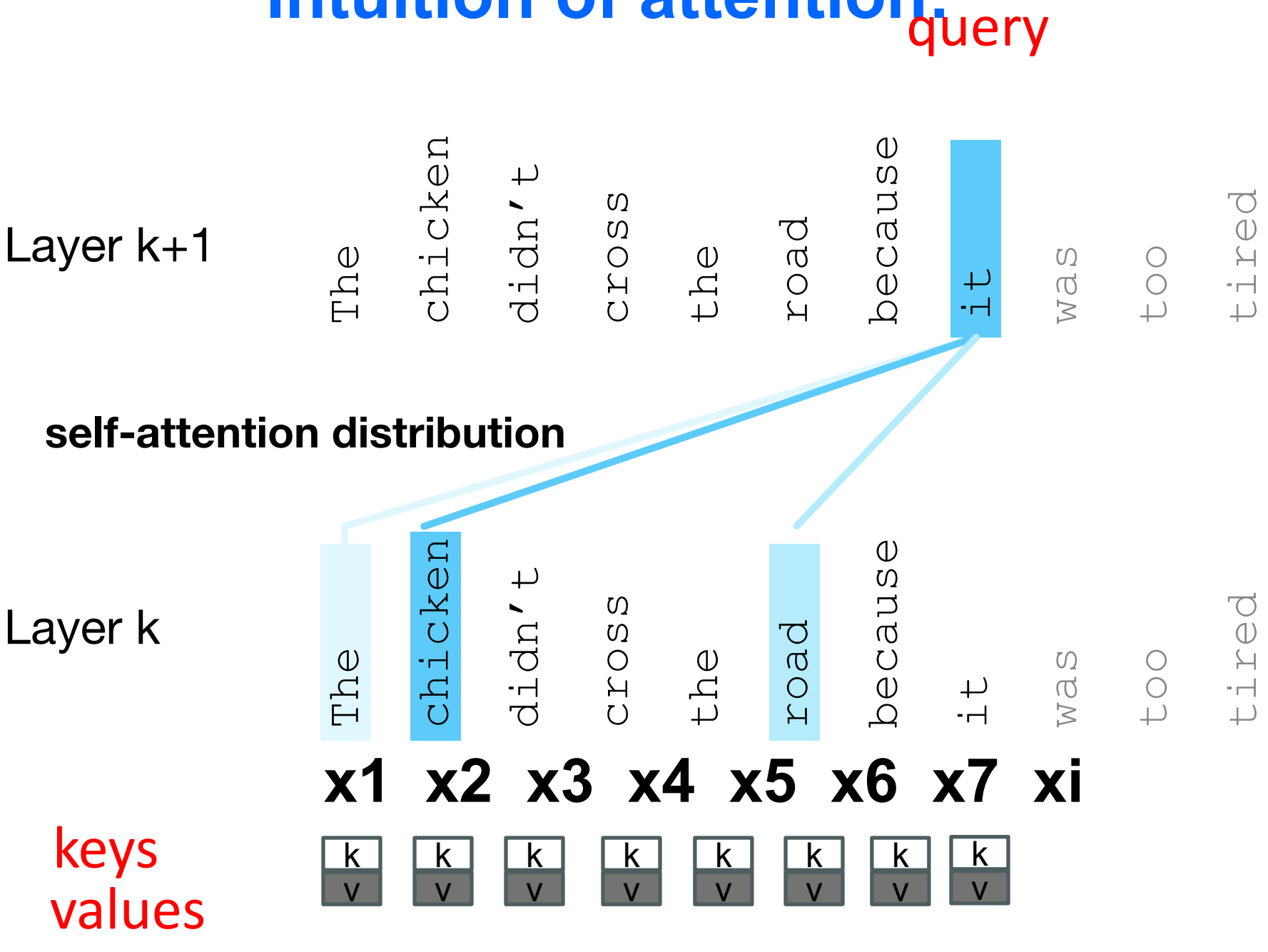
$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij}\mathbf{v}_j$$

$$\text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathsf{T}}}{\sqrt{d_k}}\right)\mathbf{V}$$

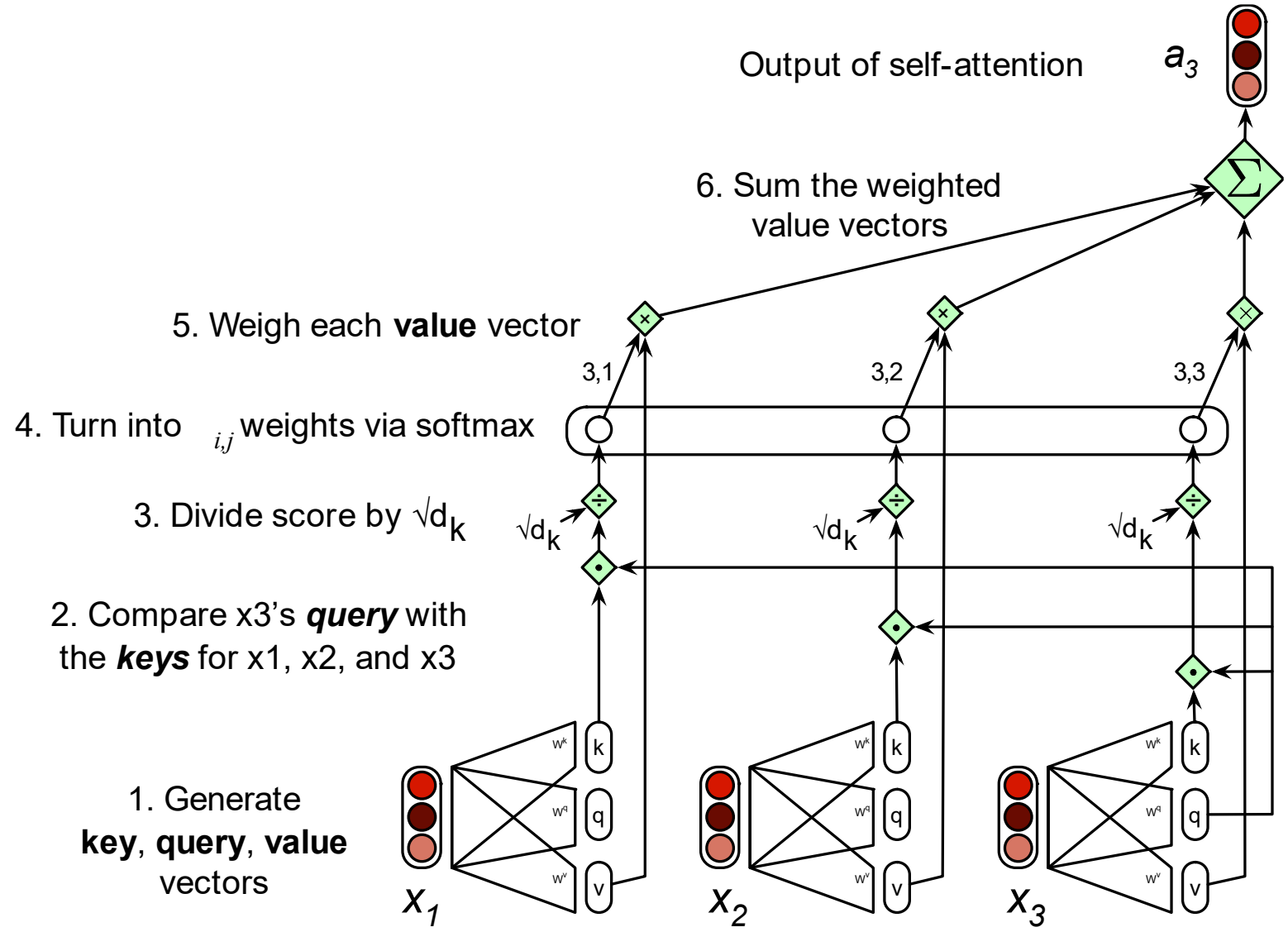
Attention intuition



Intuition of attention:



Calculating the value of a3



Multi-head attention

- **What Is Multi-Head Attention?**
  - **Multi-Head Attention (MHA)** is a mechanism used in Transformers that allows the model to focus on **different types of relationships** in the input sequence **at the same time**.
  - Instead of computing *one* attention operation, the model computes **multiple attention "heads" in parallel**, each learning a different pattern.

- **Why do we need multiple heads?**
  - A single attention head can focus on **only one type of relationship** at a time.
  - But language contains **multiple simultaneous patterns**, such as:
    - subject → verb dependencies
    - long-range connections
    - coreference ("he", "she", "it")
    - style and tone
    - positional relationships
    - phrase boundaries
- With **multiple heads**, each head specializes in a different aspect of the sequence.

Multi-head attention

How Multi-Head Attention Works

For each token embedding **x**, the model creates:

- Query vector (Q)
- Key vector (K)
- Value vector (V)

But with *multiple heads*, the model does this **h times**:

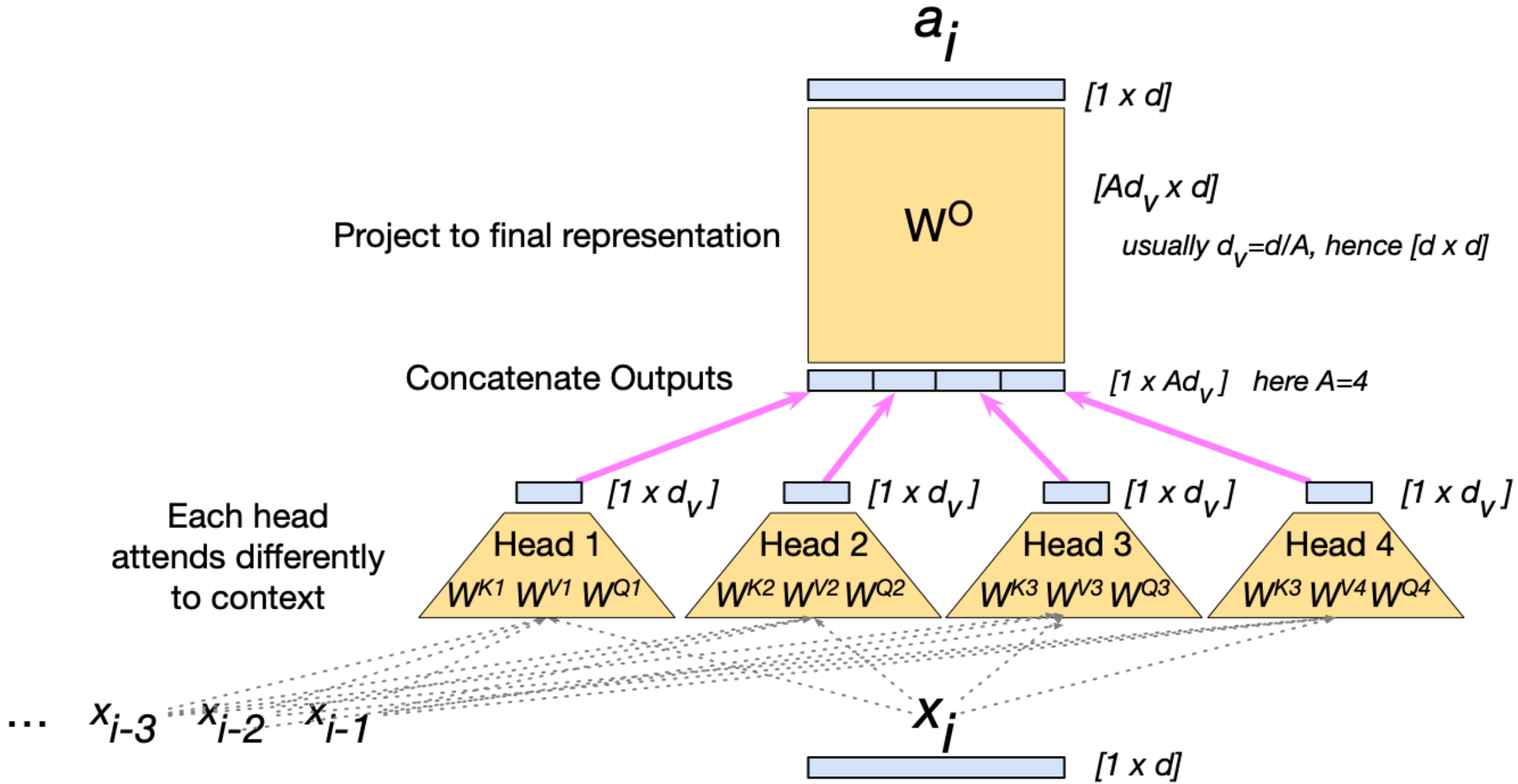
```
Head 1: Q1, K1, V1
Head 2: Q2, K2, V2
Head 3: Q3, K3, V3
...
Head h: Qh, Kh, Vh
```

Actual Attention: slightly more complicated

- Instead of one attention head, we'll have lots of them!
- Intuition: each head might be attending to the context for different purposes
  - Different linguistic relationships or patterns in the context

$$
\begin{aligned}
\mathbf{q}_i^c = \mathbf{x}_i\mathbf{W}^{\mathbf{Qc}};\quad \mathbf{k}_j^c = \mathbf{x}_j\mathbf{W}^{\mathbf{Kc}};\quad \mathbf{v}_j^c &= \mathbf{x}_j\mathbf{W}^{\mathbf{Vc}};\quad \forall\, c \quad 1 \le c \le h \\
\text{score}^c(\mathbf{x}_i, \mathbf{x}_j) &= \frac{\mathbf{q}_i^c \cdot \mathbf{k}_j^c}{\sqrt{d_k}} \\
\alpha_{ij}^c &= \text{softmax}(\text{score}^c(\mathbf{x}_i, \mathbf{x}_j))\ \ \forall j \le i \\
\mathbf{head}_i^c &= \sum_{j \le i} \alpha_{ij}^c \mathbf{v}_j^c \\
\mathbf{a}_i &= (\mathbf{head}^1 \oplus \mathbf{head}^2 \ldots \oplus \mathbf{head}^h)\mathbf{W}^O \\
\text{MultiHeadAttention}(\mathbf{x}_i, [\mathbf{x}_1, \cdots, \mathbf{x}_N]) &= \mathbf{a}_i
\end{aligned}
$$

Multi-head attention



Different projection matrices for each head

- In a Transformer layer, each head has its own learnable matrices:

$$W_i^Q,\ W_i^K,\ W_i^V$$

- These matrices are **initialized randomly** at the start of training.
  For example, PyTorch or TensorFlow initializes them with Xavier or Kaiming initialization.
- So at time step 0:
  - Head 1 has random parameters
  - Head 2 has different random parameters
  - Head h has different random parameters
- They are **different from the very beginning**.

- **During training, each head gets *different gradients***
- Even though all heads receive the same input data X:
  - each head computes a different attention pattern
  - each head then contributes differently to the loss
  - each head receives a different gradient signal
  - therefore each head's parameter update is unique

- So training pushes each head's W matrices in **different directions**.
  - They take different notes and observe different patterns.
  - Over time, each becomes an expert in a different aspect.
- We get different learnable Q, K, V projection matrices because each head has its own randomly initialized parameters that are independently updated by backpropagation. Even though the input is the same, the gradients each head receives are different, so the projection matrices diverge and learn different patterns.

Cross-Attention

- **Cross-attention** is an attention mechanism where:
  - **Queries (Q)** come from **one sequence**
  - **Keys (K) and Values (V)** come from **another sequence**
- In short:

$$Q = \textit{from sequence } A, \qquad K, V = \textit{from sequence } B$$

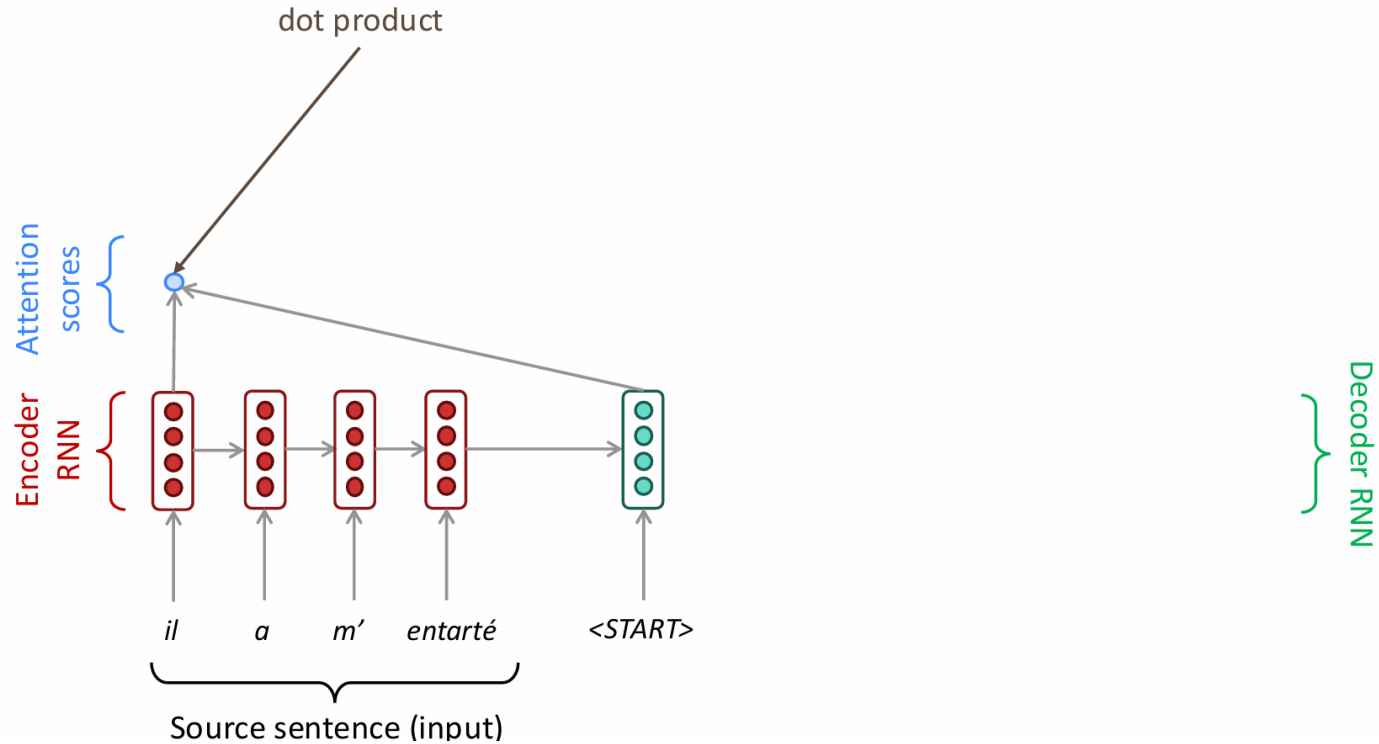
- This allows one sequence to **attend to** or **look at** another sequence.
- Cross-attention enables the model to *connect information between different sources*. It is crucial in:
  - Encoder–Decoder Transformers (e.g., original Transformer for translation)
  - Diffusion models (image generation)
  - Retrieval-augmented models
  - Multimodal LLMs (Vision+Language, Speech+Language)

Type	Q	K	V	Purpose
Self-Attention	same sequence	same sequence	same sequence	Understand internal relationships
Cross-Attention	sequence A	sequence B	sequence B	Connect two different sequences

Cross-Attention

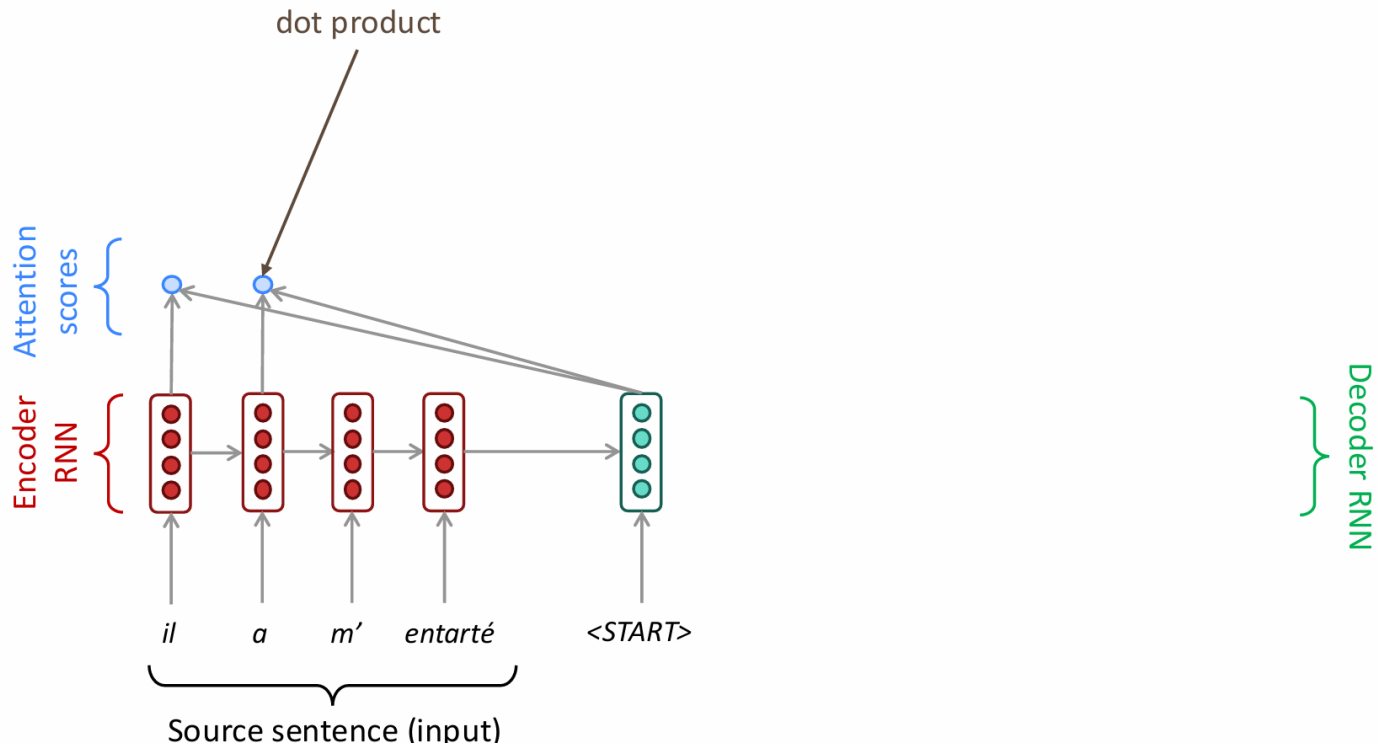
Sequence-to-sequence with attention

Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on particular part* of the source sequence



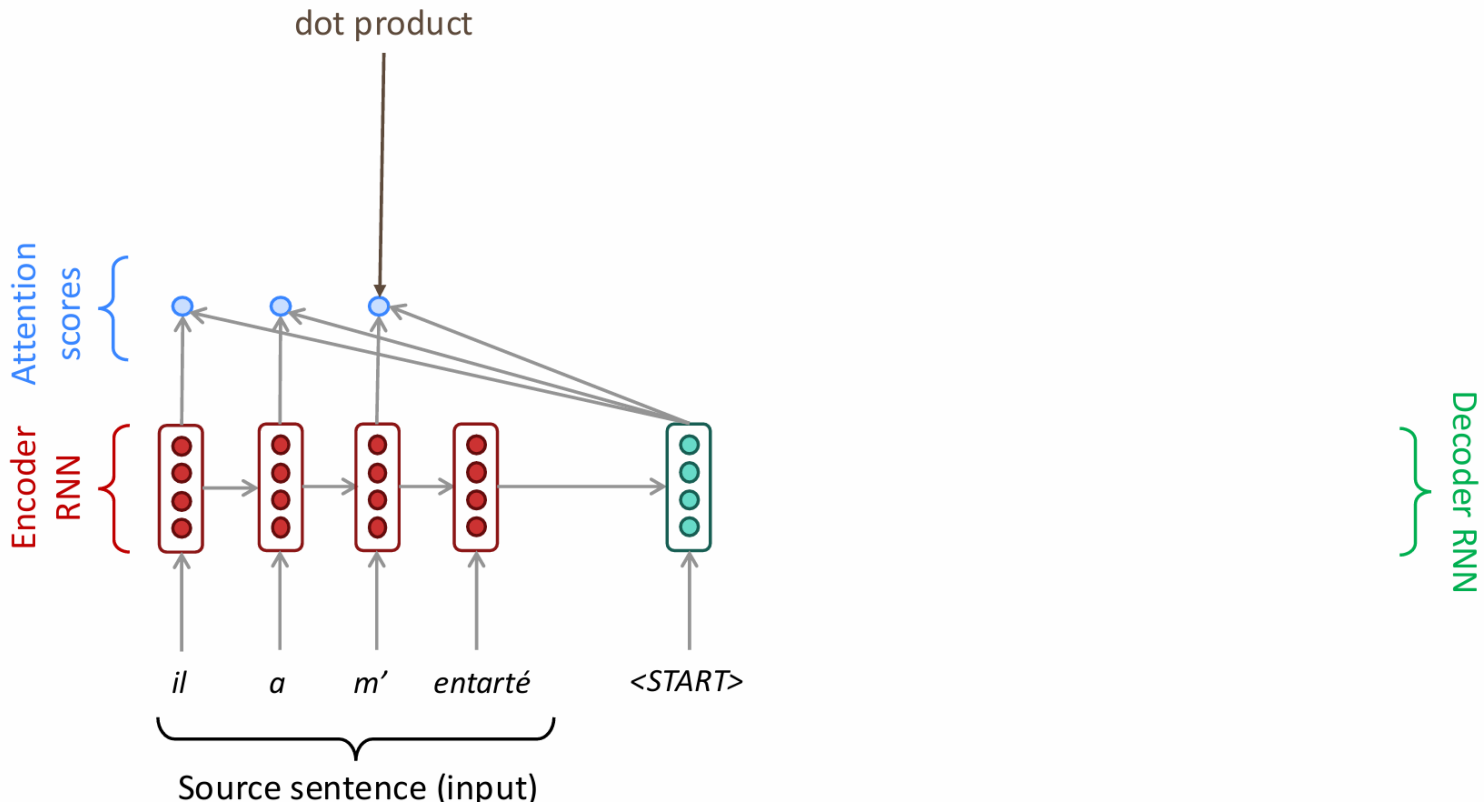
Cross-Attention

Sequence-to-sequence with attention



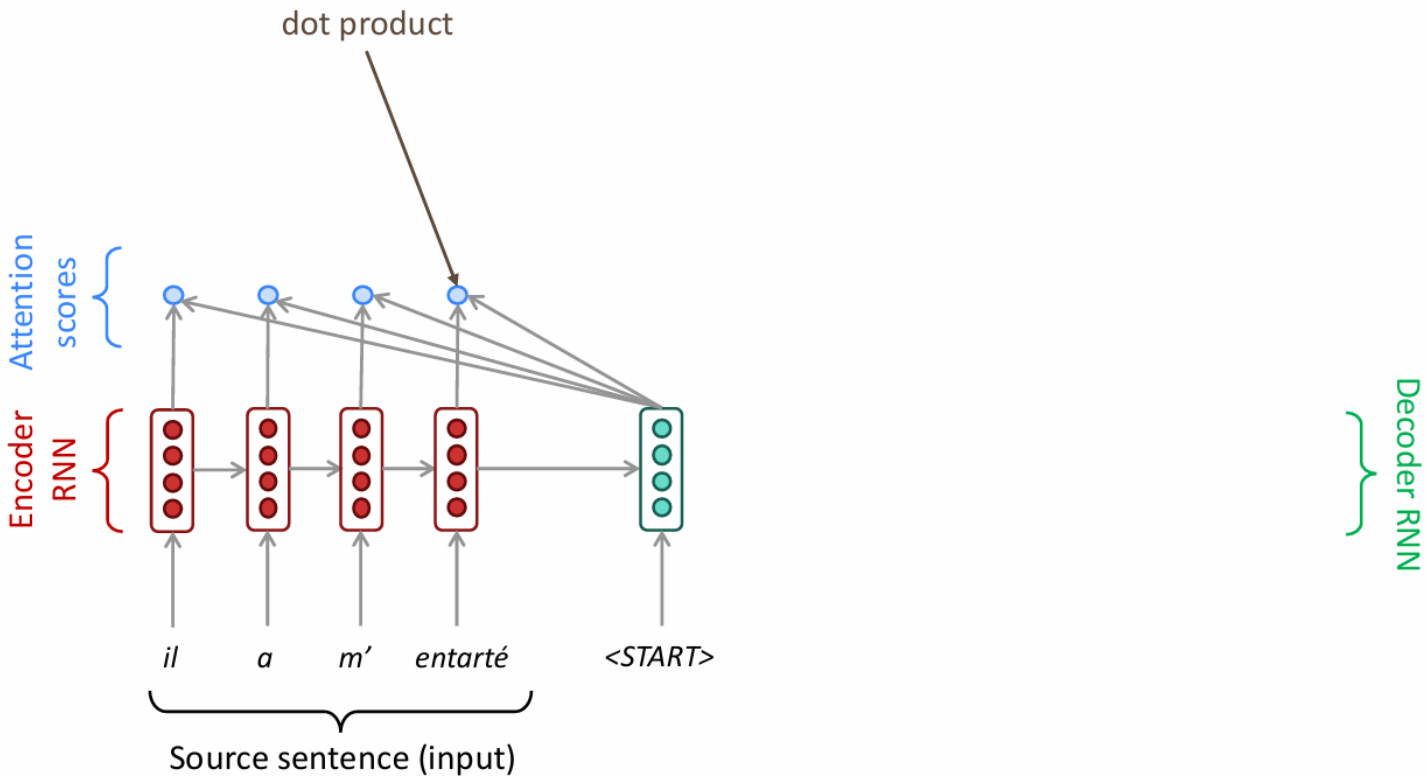
Cross-Attention

Sequence-to-sequence with attention



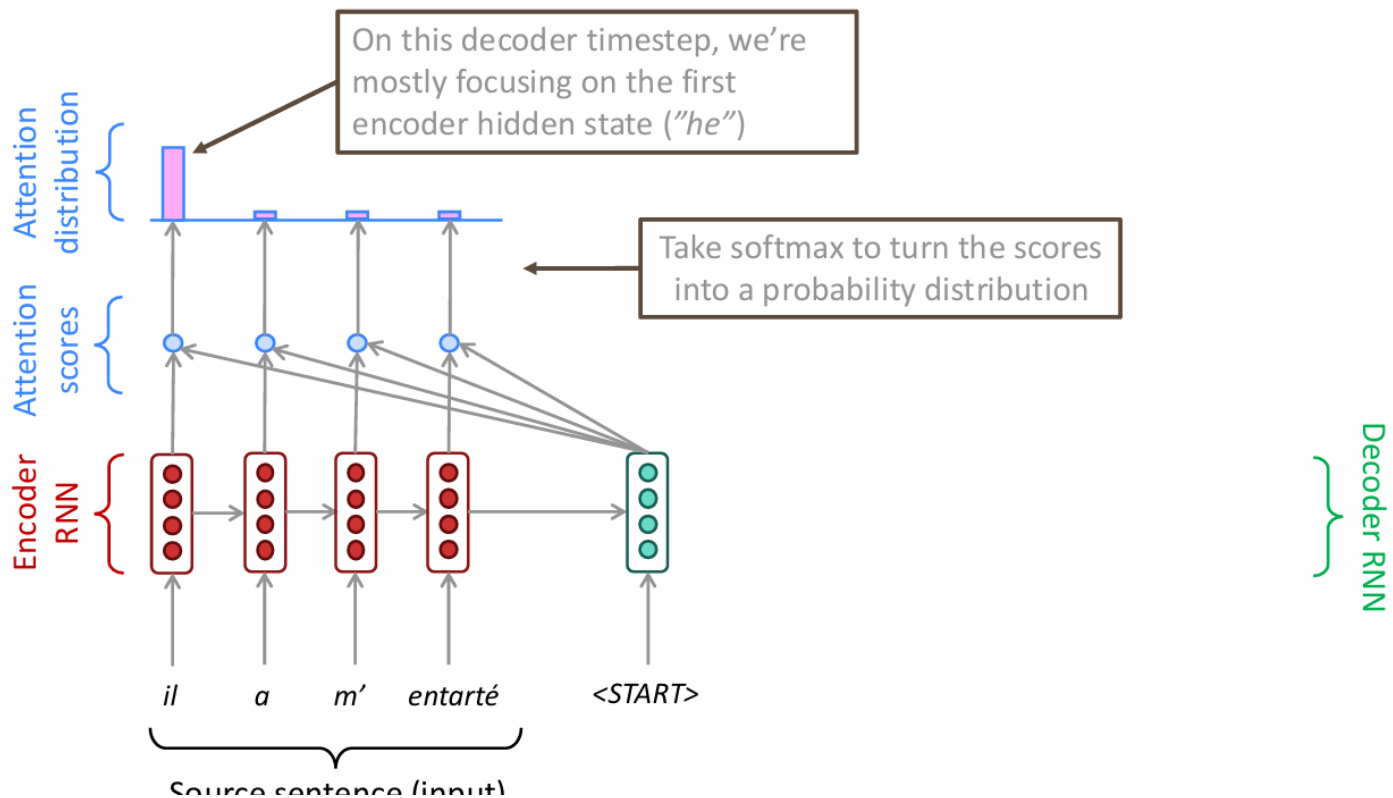
Cross-Attention

Sequence-to-sequence with attention



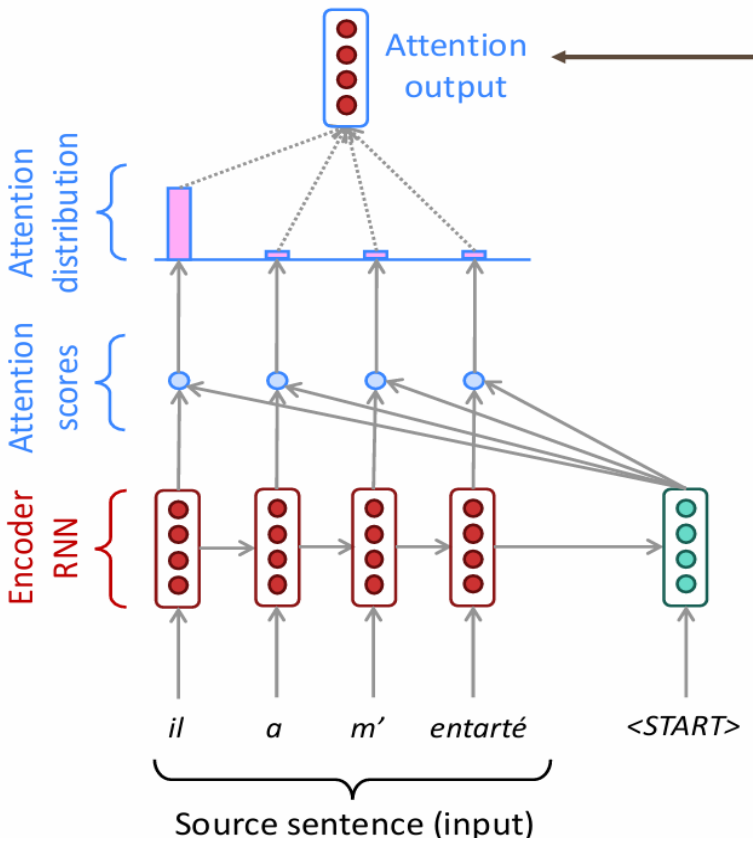
Cross-Attention

Sequence-to-sequence with attention



Cross-Attention

Sequence-to-sequence with attention



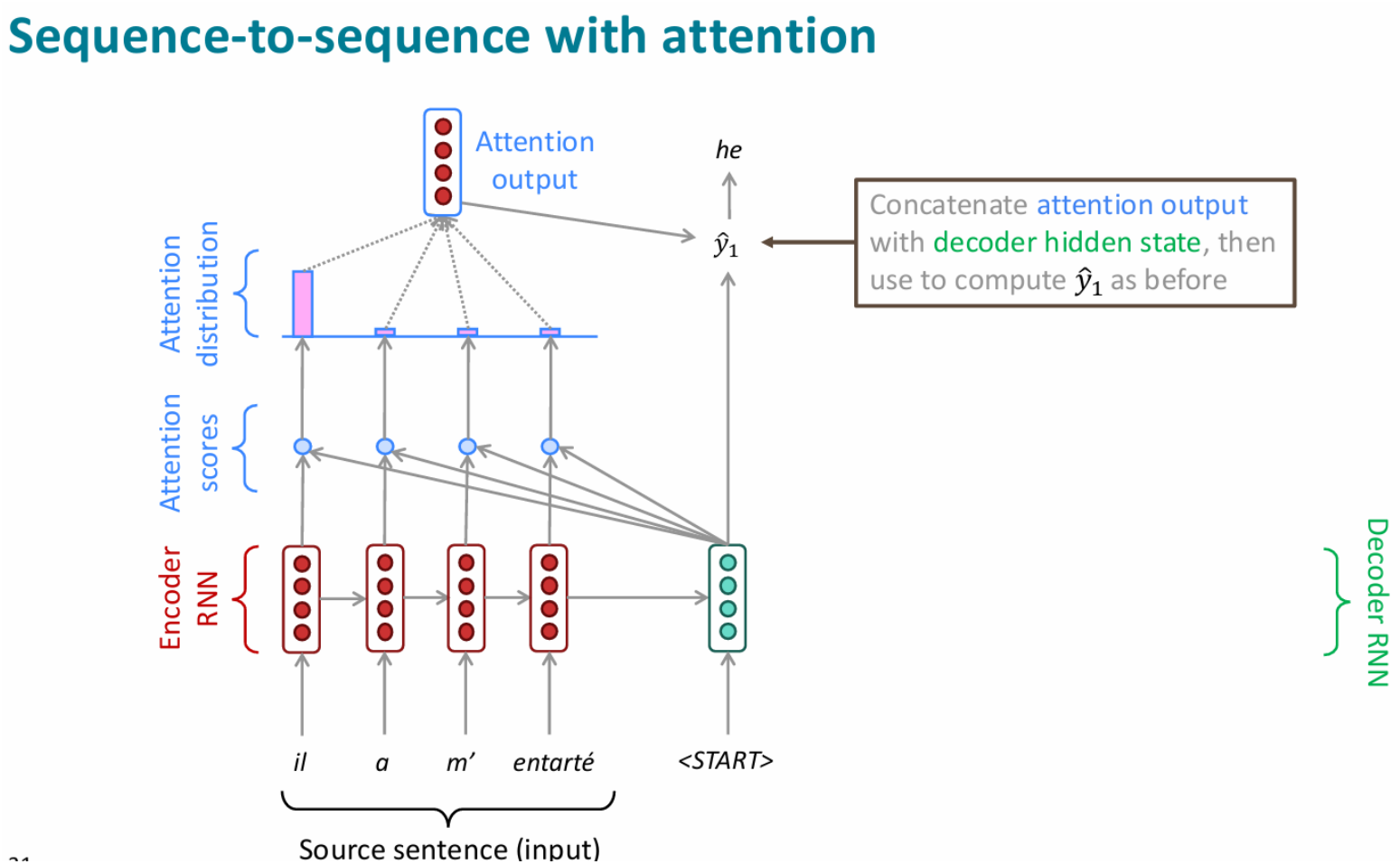
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information from the hidden states that received high attention.

Decoder RNN

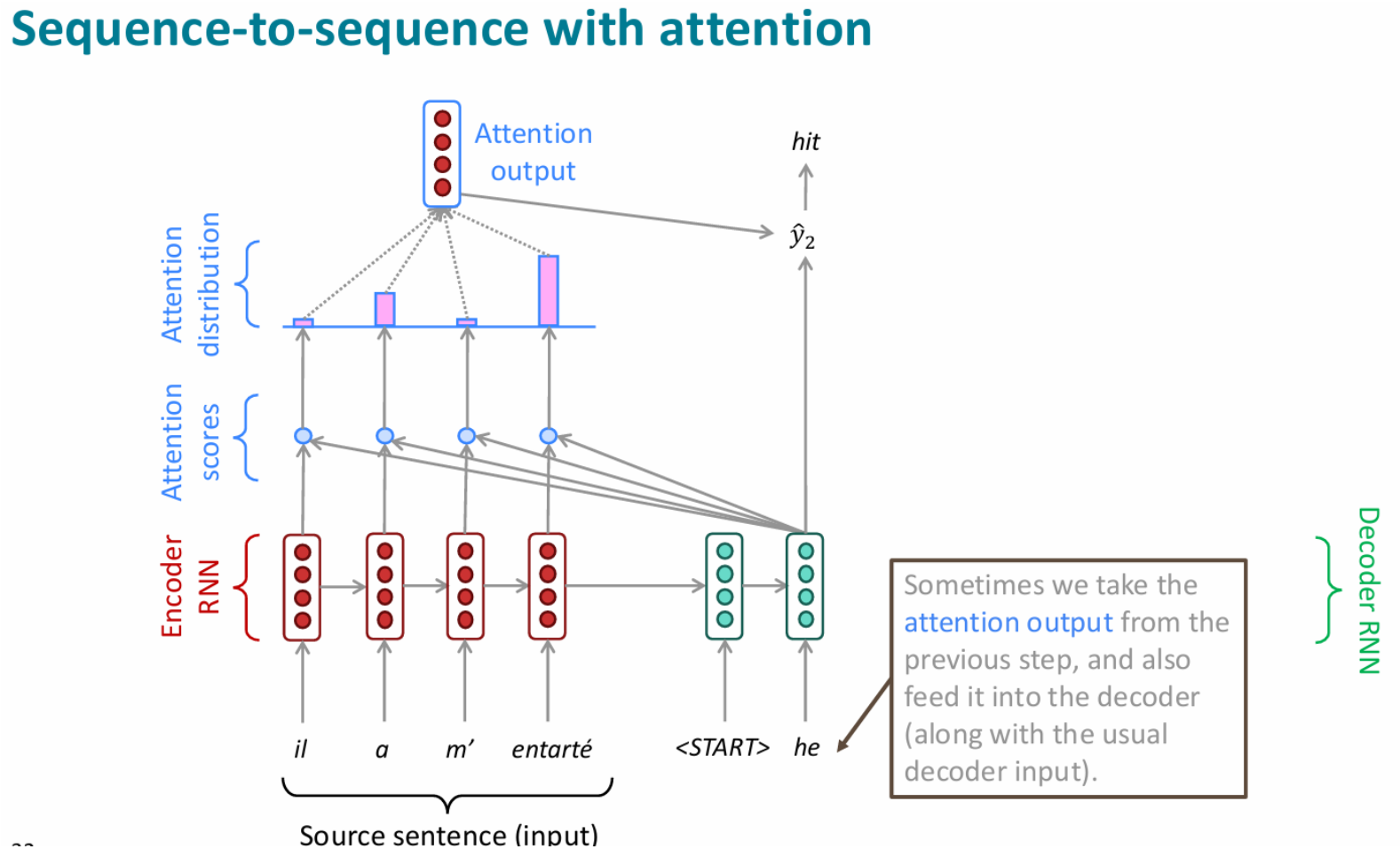
Cross-Attention

Sequence-to-sequence with attention



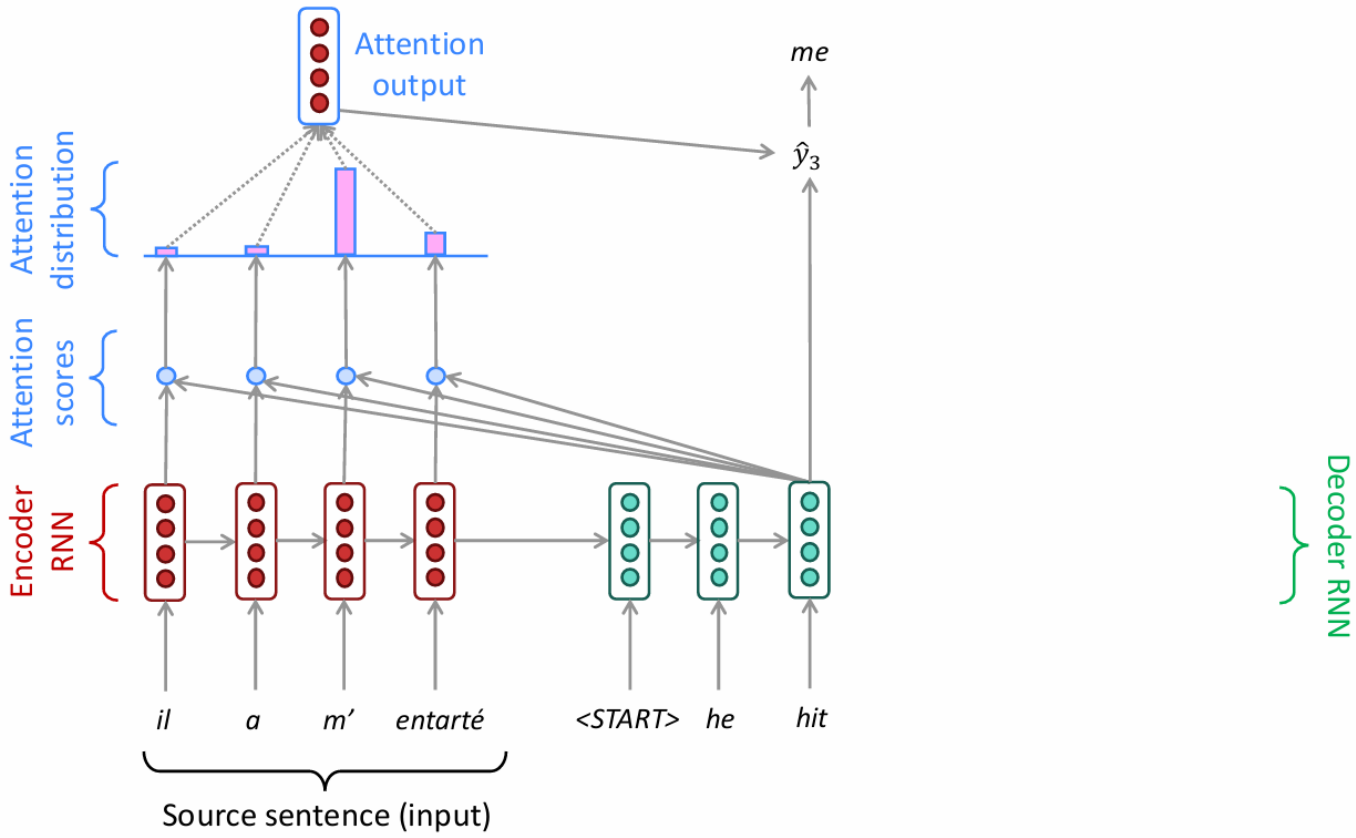
Cross-Attention

Sequence-to-sequence with attention



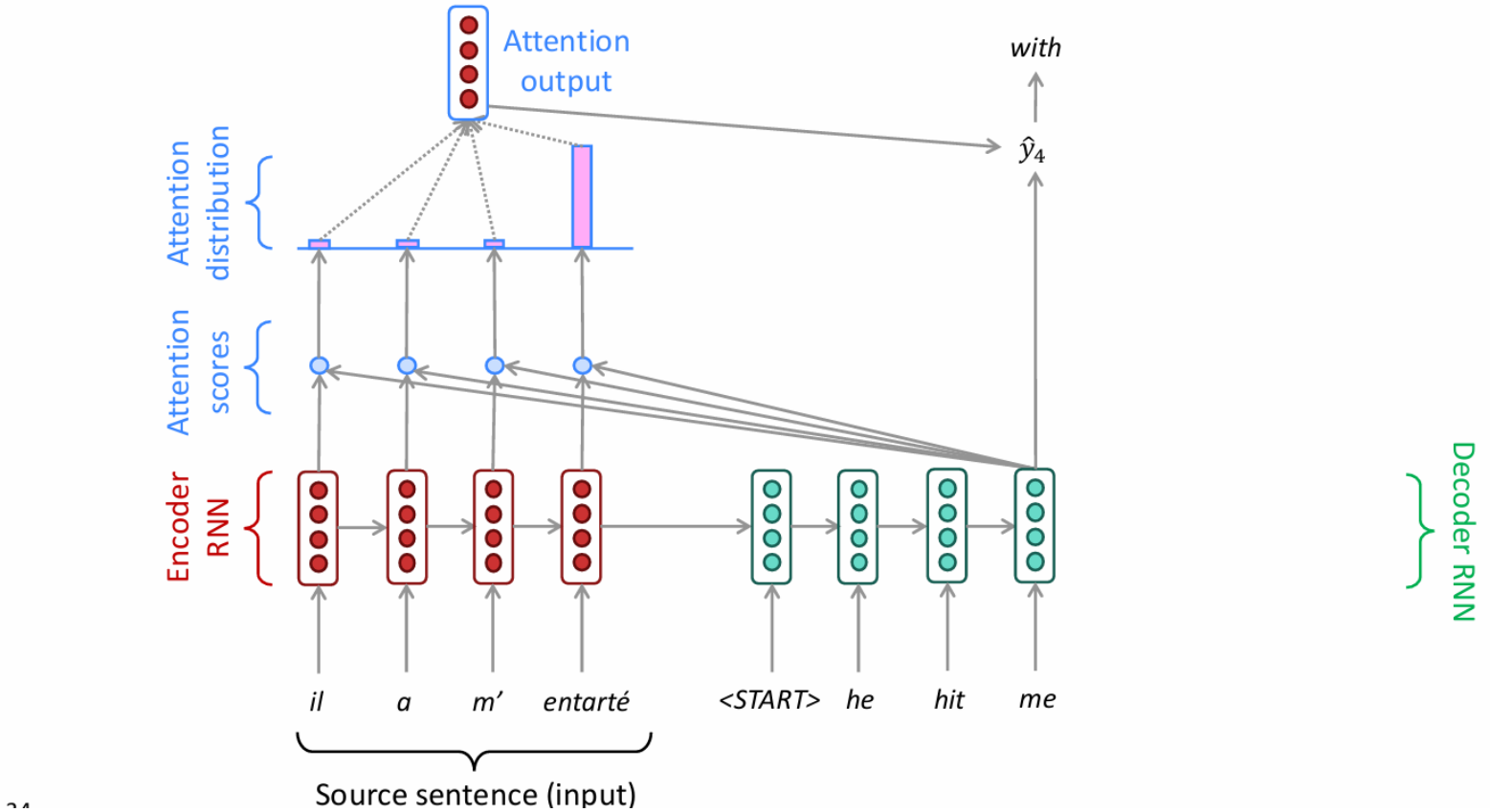
Cross-Attention

Sequence-to-sequence with attention



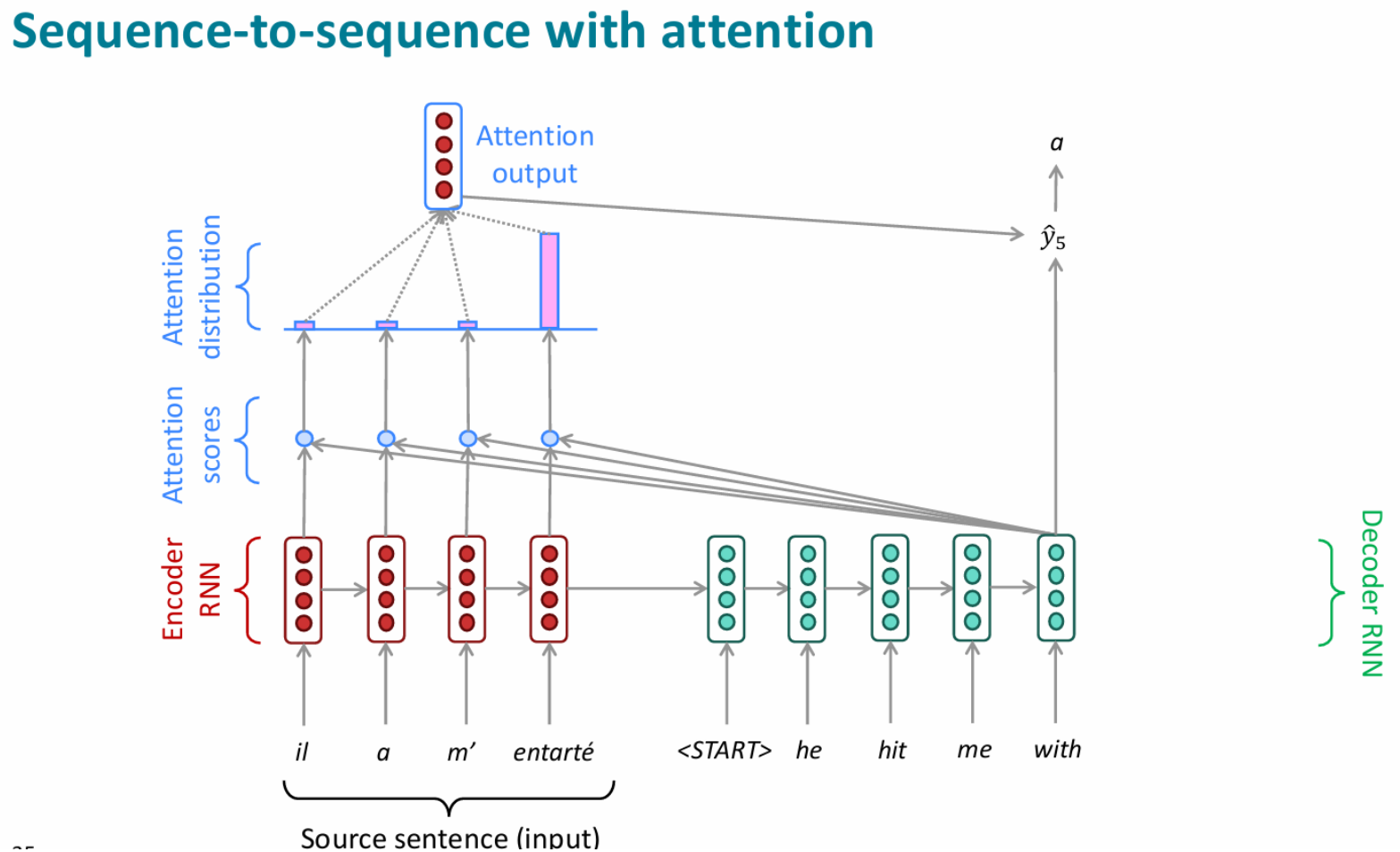
Cross-Attention

Sequence-to-sequence with attention



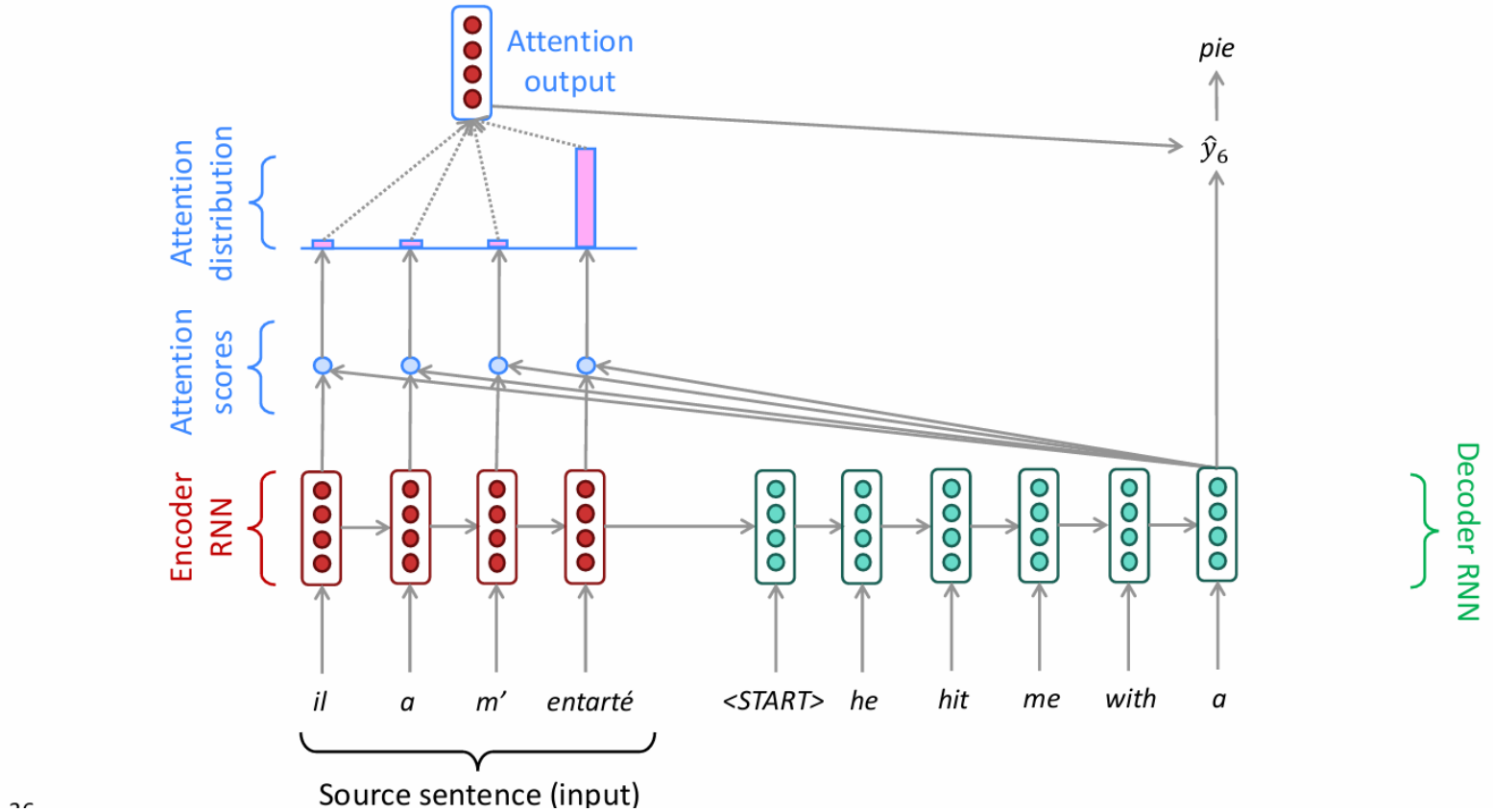
Cross-Attention

Sequence-to-sequence with attention



Cross-Attention

Sequence-to-sequence with attention



Summary

- Attention is a method for enriching the representation of a token by incorporating contextual information
- The result: the embedding for each word will be different in different contexts!
- Contextual embeddings: a representation of word meaning in its context.

Summary

Attention is great!

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention provides a more "human-like" model of the MT process
  - You can look back at the source sentence while translating, rather than needing to remember it all
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with the vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we see what the decoder was focusing on
  - We get (soft) alignment for free!
  - The network just learned alignment by itself
- (One issue – attention has *quadratic* cost with respect to sequence length)