Lecture 11: Transformers: Self-Attention Networks

Instructor: Swabha Swayamdipta
USC CSCI 544 Applied NLP
Oct 01, Fall 2024

Some slides adapted from Dan Jurafsky and Chris Manning

Announcements

- Thu: Quiz 3
  - Before that: **Install Lockdown Browser**
  - Cannot take Quiz 3 onwards otherwise
  - 36 students did not sign the acknowledgment for the lockdown browser, and may not be able to take the quiz in time… I won't be making exceptions for anyone
- Next Tue:
  - HW2 due - please follow naming format etc. (see Brightspace announcement)
  - Guest lecture by TA Sayan Ghosh on PyTorch for Transformers
- Next Thu: No class / Fall Break
- Tue 10/15: Midterm Exam
  - 1 hr - format similar to quizzes
- HW1 / Project Proposal grades will be available by the end of the week
- Sign up sheet now open for Paper Presentation and Final Project Presentation dates (see Brightspace announcement)

Lecture Outline

- Announcements
- Recap: Seq2Seq and Attention
- More on Attention
- Transformers: Self-Attention Networks
  - Multiheaded Attention
  - Positional Embeddings
  - Transformer Blocks
- Transformers as Encoders, Decoders and Encoder-Decoders

Recap: Sequence-to-Sequence and Attention

RNNLMs are Autoregressive Models

- Autoregressive models predict a value at time t based on a function of the previous values at times $t-1$, $t-2$, and so on
- Word generated at each time step is conditioned on the word selected by the network from the previous step
- State-of-the-art generation approaches are all autoregressive!
    - Machine translation, question answering, summarization
- Key technique: prime the generation with the most suitable **context**

Can do better than <s>!

Provide rich task-appropriate context!

(Neural) Machine Translation

Provide rich task-appropriate context!

(Neural) Machine Translation

Provide rich task-appropriate context!

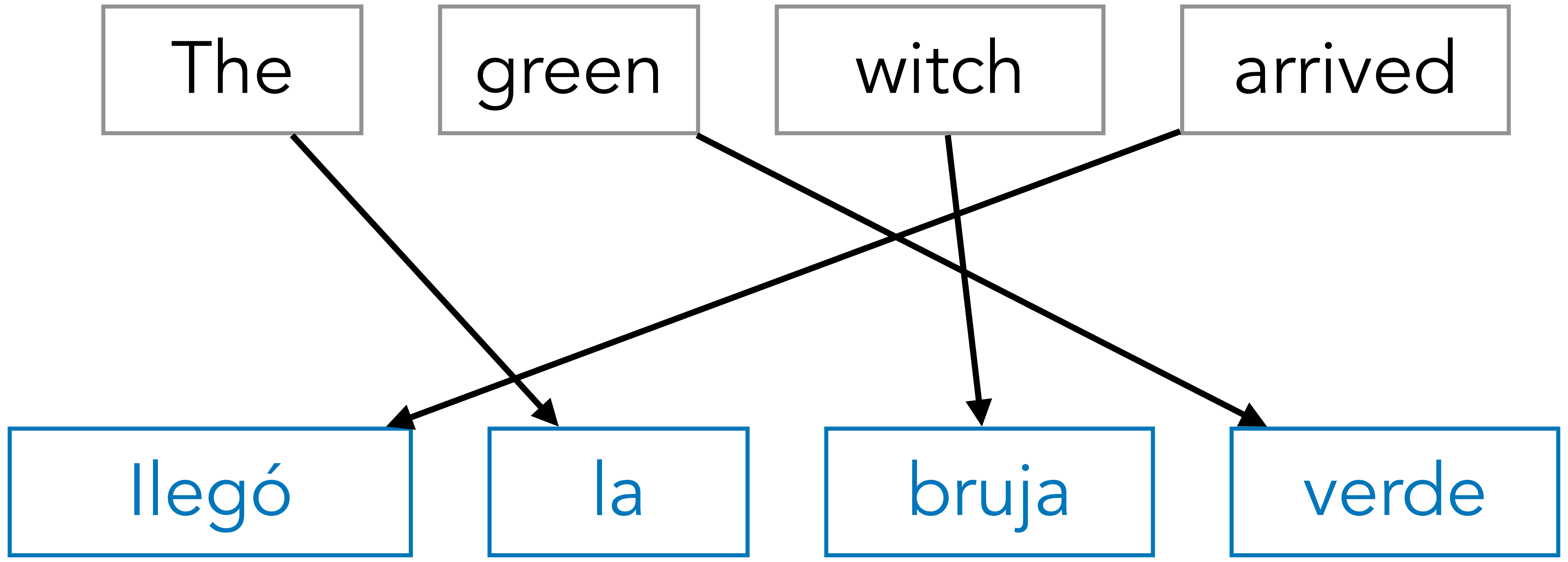
- Sequence Generation Problem (as opposed to sequence classification)
  - $\mathbf{x}$ = Source sequence of length n
  - $\mathbf{y}$ = Target sequence of length m

Sequence-to-Sequence (Seq2seq)

(Neural) Machine Translation

Provide rich task-appropriate context!

- Sequence Generation Problem (as opposed to sequence classification)
  - $\mathbf{x}$ = Source sequence of length n
  - $\mathbf{y}$ = Target sequence of length m

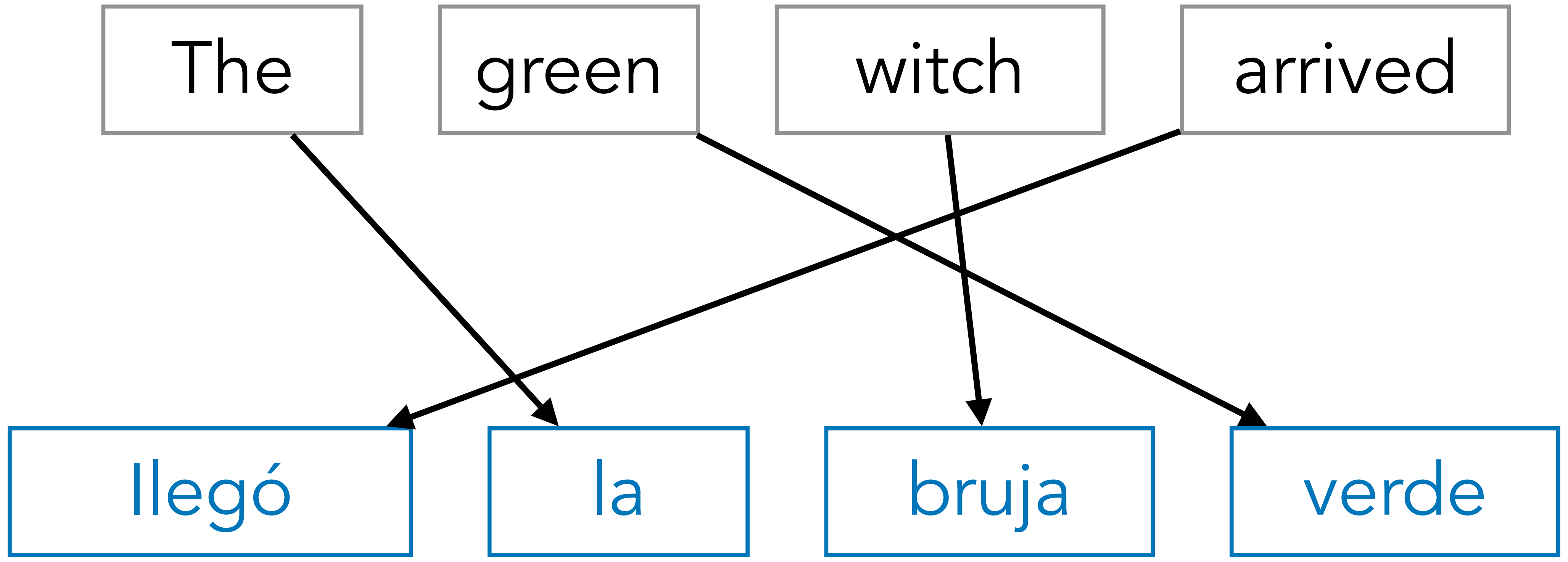


Sequence-to-Sequence (Seq2seq)

(Neural) Machine Translation

Provide rich task-appropriate context!

- Sequence Generation Problem (as opposed to sequence classification)
  - $\mathbf{x}$ = Source sequence of length n
  - $\mathbf{y}$ = Target sequence of length m
- Different from regular generation from an LM
  - Since we expect the target sequence to serve a specific utility (translate the source)



Sequence-to-Sequence (Seq2seq)

Sequence-to-Sequence Models

- Models capable of generating contextually appropriate, arbitrary length, output sequences given an input sequence.
- The key idea underlying these networks is the use of an **encoder network** that takes an input sequence and creates a contextualized representation of it, often called the context.
- This representation is then passed to a **decoder network** which generates a task- specific output sequence.

Encoder-Decoder Networks

Encoder-Decoder Networks

Encoder-Decoder Networks

Encoder-decoder networks consist of three components:

Encoder-Decoder Networks

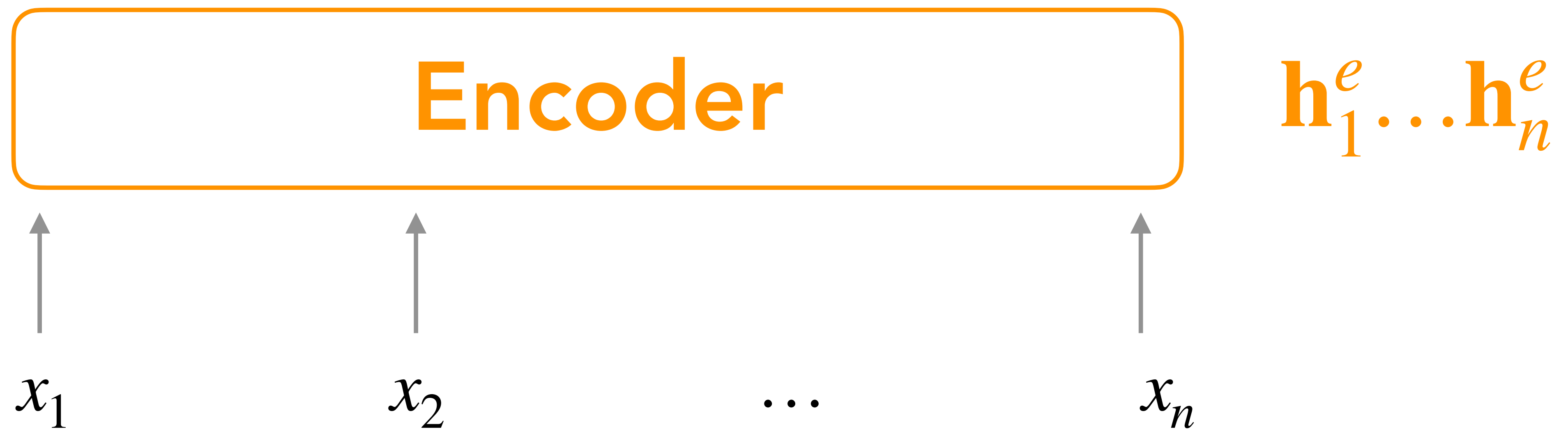
Encoder-decoder networks consist of three components:

1. An **encoder** that accepts an input sequence, $\mathbf{x}_{1:n}$ and generates a corresponding sequence of contextualized representations, $\mathbf{h}_1^e \ldots \mathbf{h}_n^e$

Encoder-Decoder Networks

Encoder-decoder networks consist of three components:

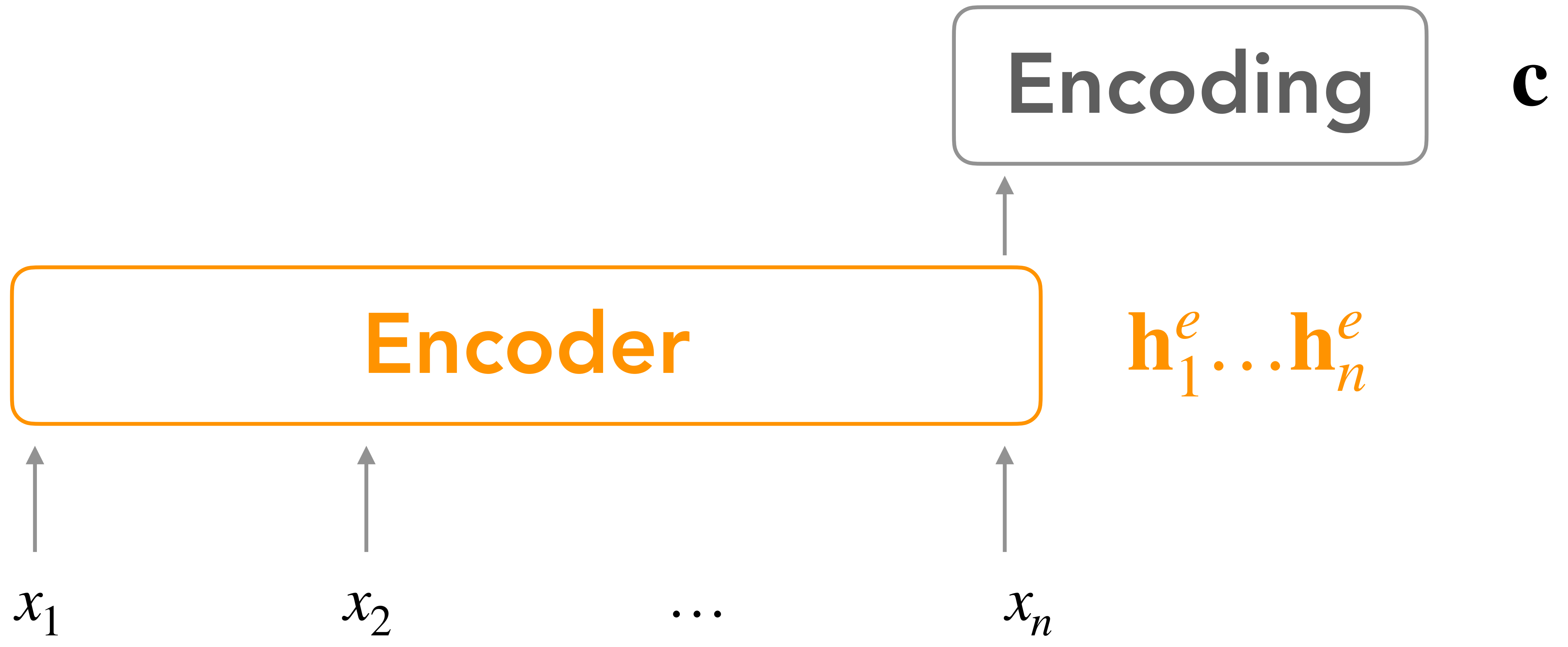
1. An **encoder** that accepts an input sequence, $\mathbf{x}_{1:n}$ and generates a corresponding sequence of contextualized representations, $\mathbf{h}_1^e \ldots \mathbf{h}_n^e$



Encoder-Decoder Networks

Encoder-decoder networks consist of three components:

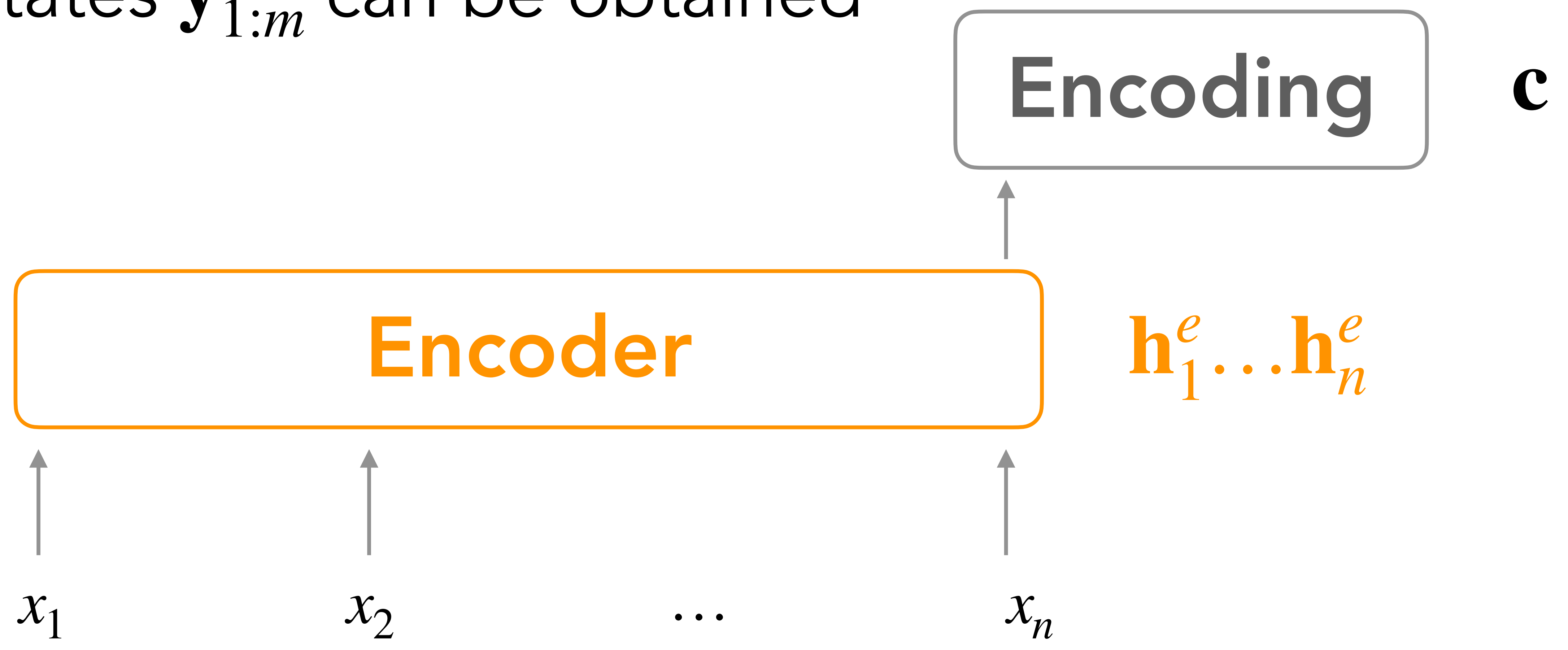
1. An **encoder** that accepts an input sequence, $\mathbf{x}_{1:n}$ and generates a corresponding sequence of contextualized representations, $\mathbf{h}^e_1 \ldots \mathbf{h}^e_n$
2. A **encoding** vector, $\mathbf{c}$ which is a function of $\mathbf{h}^e_1 \ldots \mathbf{h}^e_n$ and conveys the essence of the input to the decoder



Encoder-Decoder Networks

Encoder-decoder networks consist of three components:

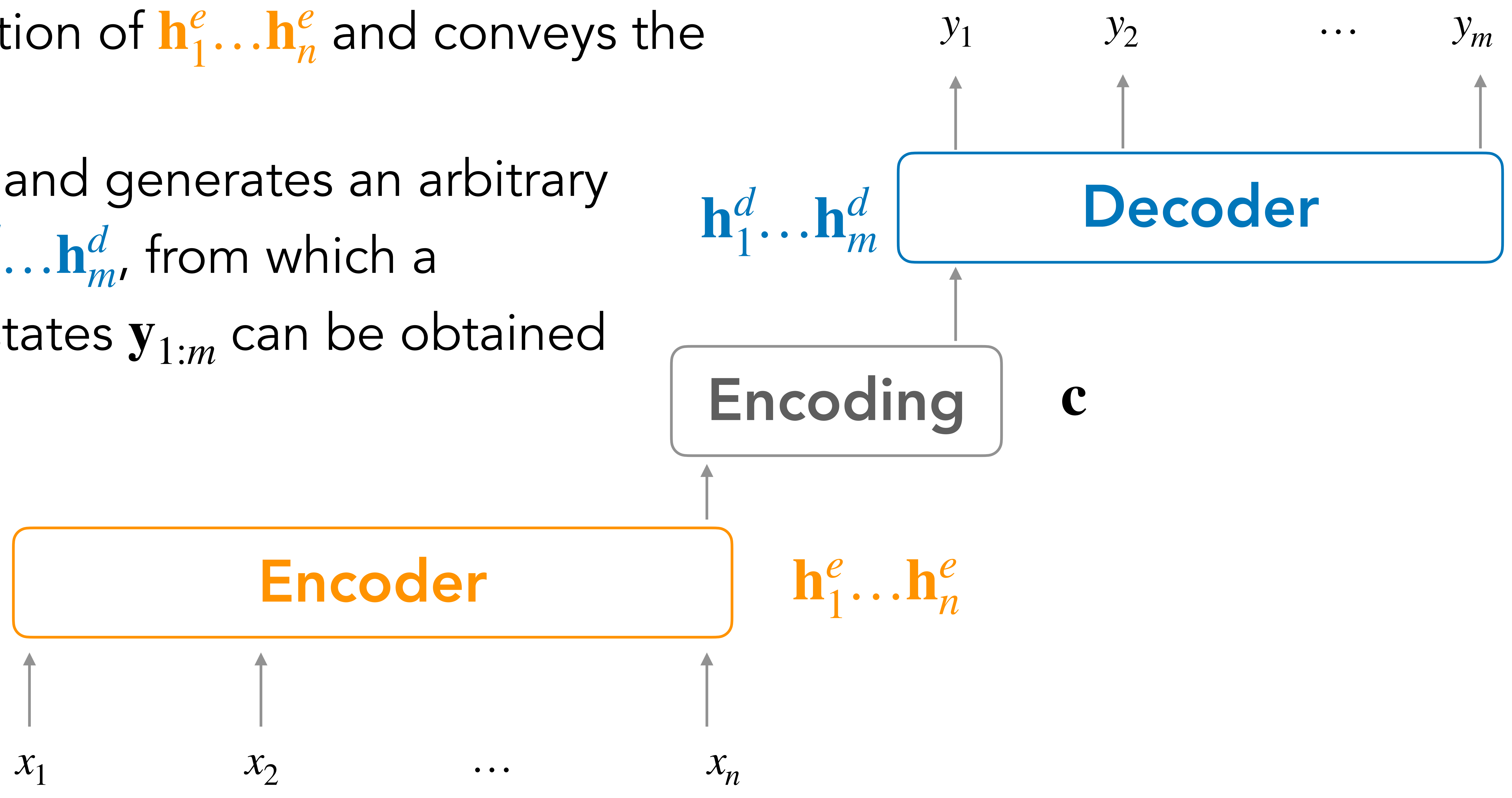
1. An **encoder** that accepts an input sequence, $\mathbf{x}_{1:n}$ and generates a corresponding sequence of contextualized representations, $\mathbf{h}^e_1 \ldots \mathbf{h}^e_n$
2. A **encoding** vector, $\mathbf{c}$ which is a function of $\mathbf{h}^e_1 \ldots \mathbf{h}^e_n$ and conveys the essence of the input to the decoder
3. A **decoder** which accepts $\mathbf{c}$ as input and generates an arbitrary length sequence of hidden states $\mathbf{h}^d_1 \ldots \mathbf{h}^d_m$, from which a corresponding sequence of output states $\mathbf{y}_{1:m}$ can be obtained



Encoder-Decoder Networks

Encoder-decoder networks consist of three components:

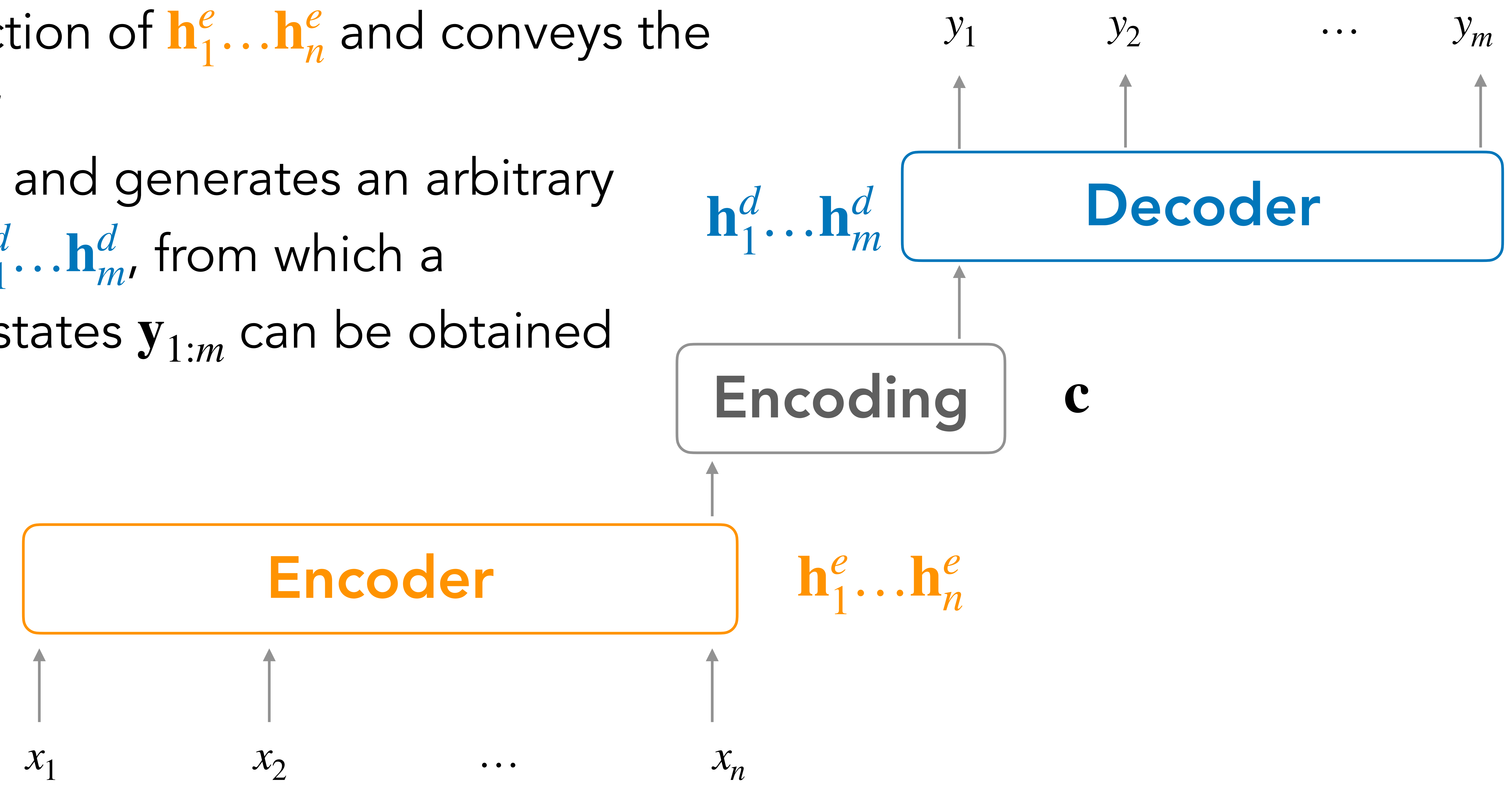
1. An **encoder** that accepts an input sequence, $\mathbf{x}_{1:n}$ and generates a corresponding sequence of contextualized representations, $\mathbf{h}^e_1 \ldots \mathbf{h}^e_n$
2. A **encoding** vector, $\mathbf{c}$ which is a function of $\mathbf{h}^e_1 \ldots \mathbf{h}^e_n$ and conveys the essence of the input to the decoder
3. A **decoder** which accepts $\mathbf{c}$ as input and generates an arbitrary length sequence of hidden states $\mathbf{h}^d_1 \ldots \mathbf{h}^d_m$, from which a corresponding sequence of output states $\mathbf{y}_{1:m}$ can be obtained



Encoder-Decoder Networks

Encoder-decoder networks consist of three components:

1. An **encoder** that accepts an input sequence, $\mathbf{x}_{1:n}$ and generates a corresponding sequence of contextualized representations, $\mathbf{h}_1^e \ldots \mathbf{h}_n^e$
2. A **encoding** vector, $\mathbf{c}$ which is a function of $\mathbf{h}_1^e \ldots \mathbf{h}_n^e$ and conveys the essence of the input to the decoder
3. A **decoder** which accepts $\mathbf{c}$ as input and generates an arbitrary length sequence of hidden states $\mathbf{h}_1^d \ldots \mathbf{h}_m^d$, from which a corresponding sequence of output states $\mathbf{y}_{1:m}$ can be obtained



Encoders and decoders can be made of FFNNs, RNNs, or Transformers

The	green	witch	arrived

Source Sentence $\mathbf{x}$

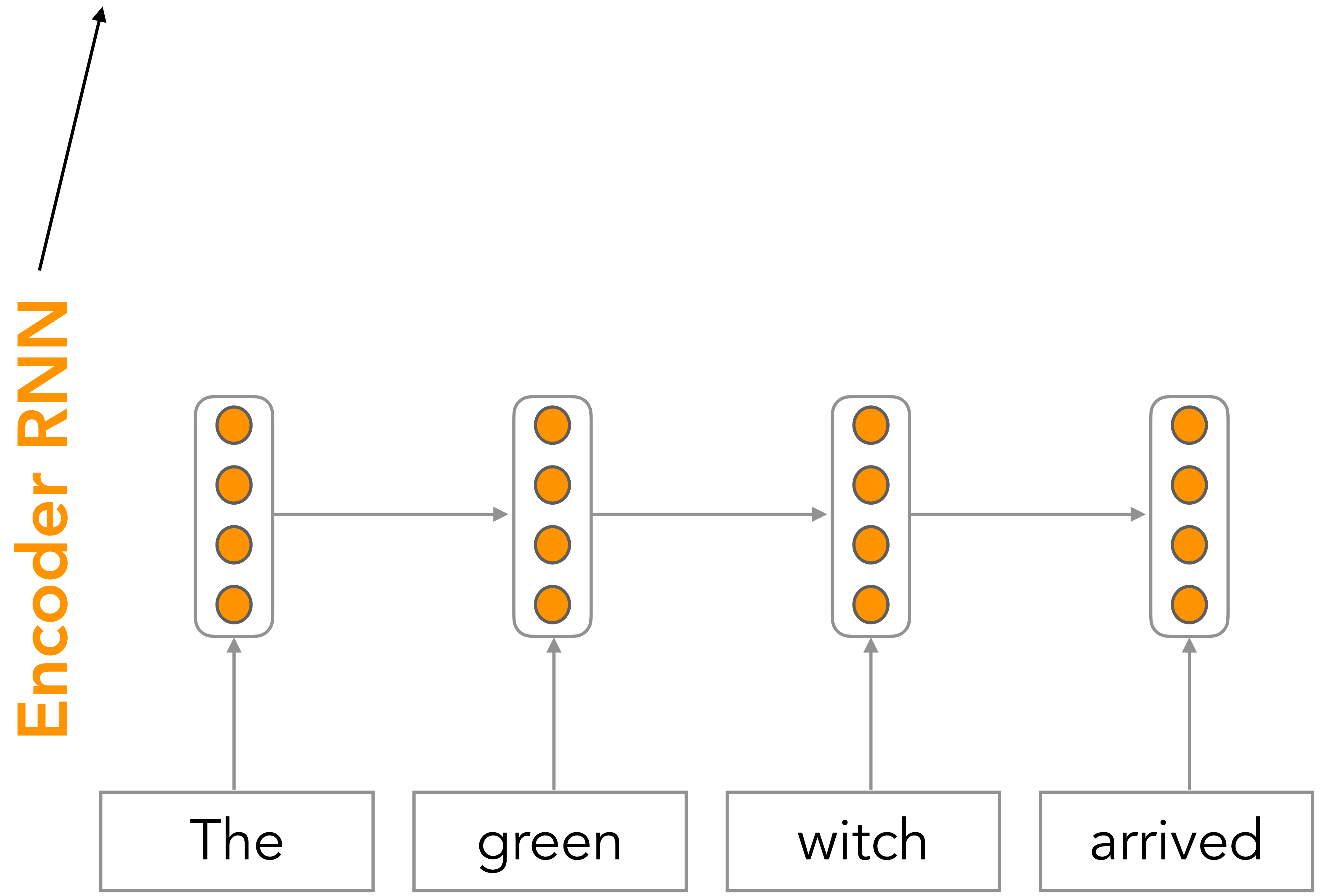
Produces an encoding of the source sequence

Encoder RNN

The	green	witch	arrived

Source Sentence $\mathbf{x}$

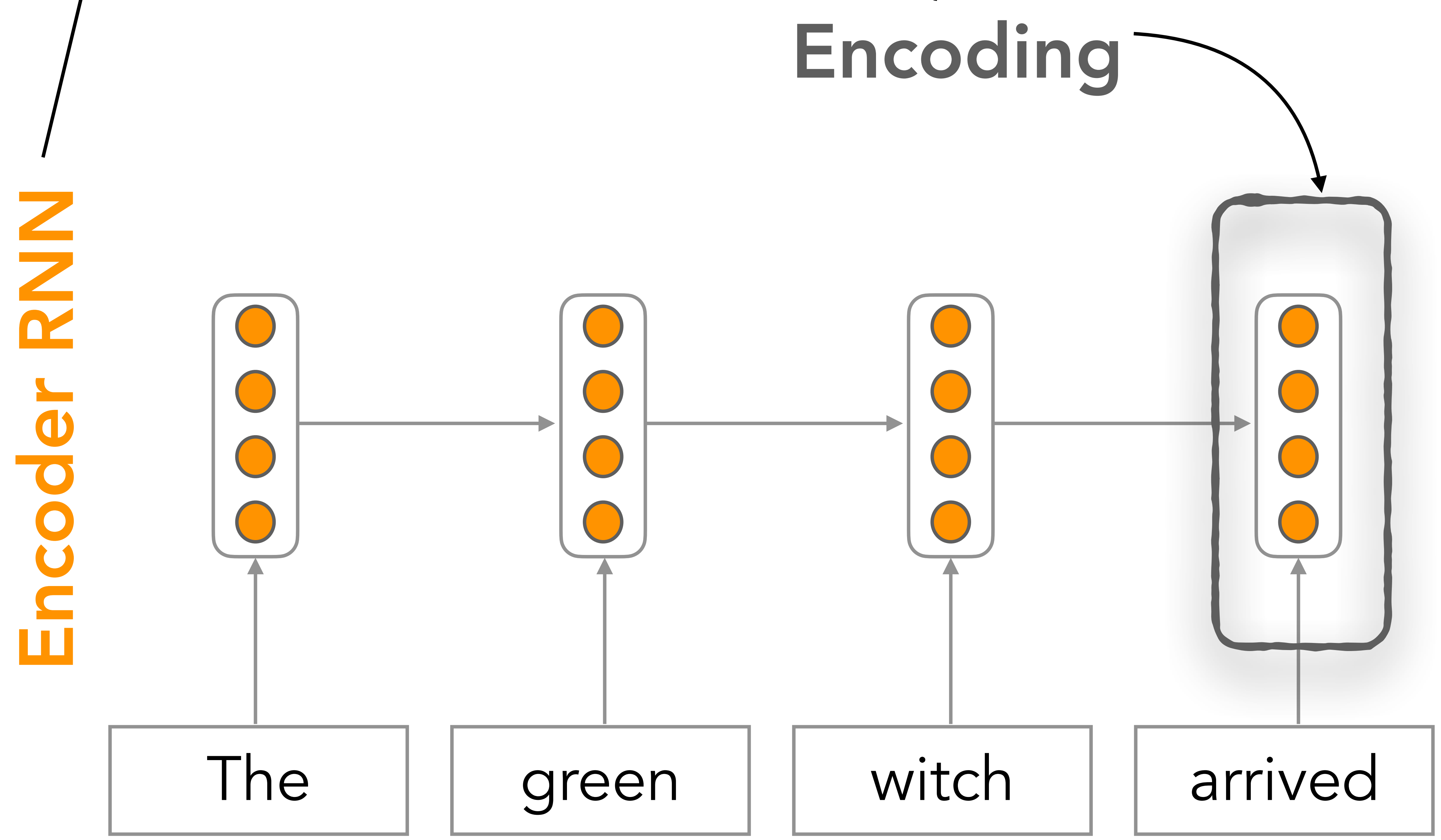
Produces an encoding of the source sequence



Source Sentence x

Produces an encoding of the source sequence

Represents input sequence. Provides initial hidden state for Decoder RNN

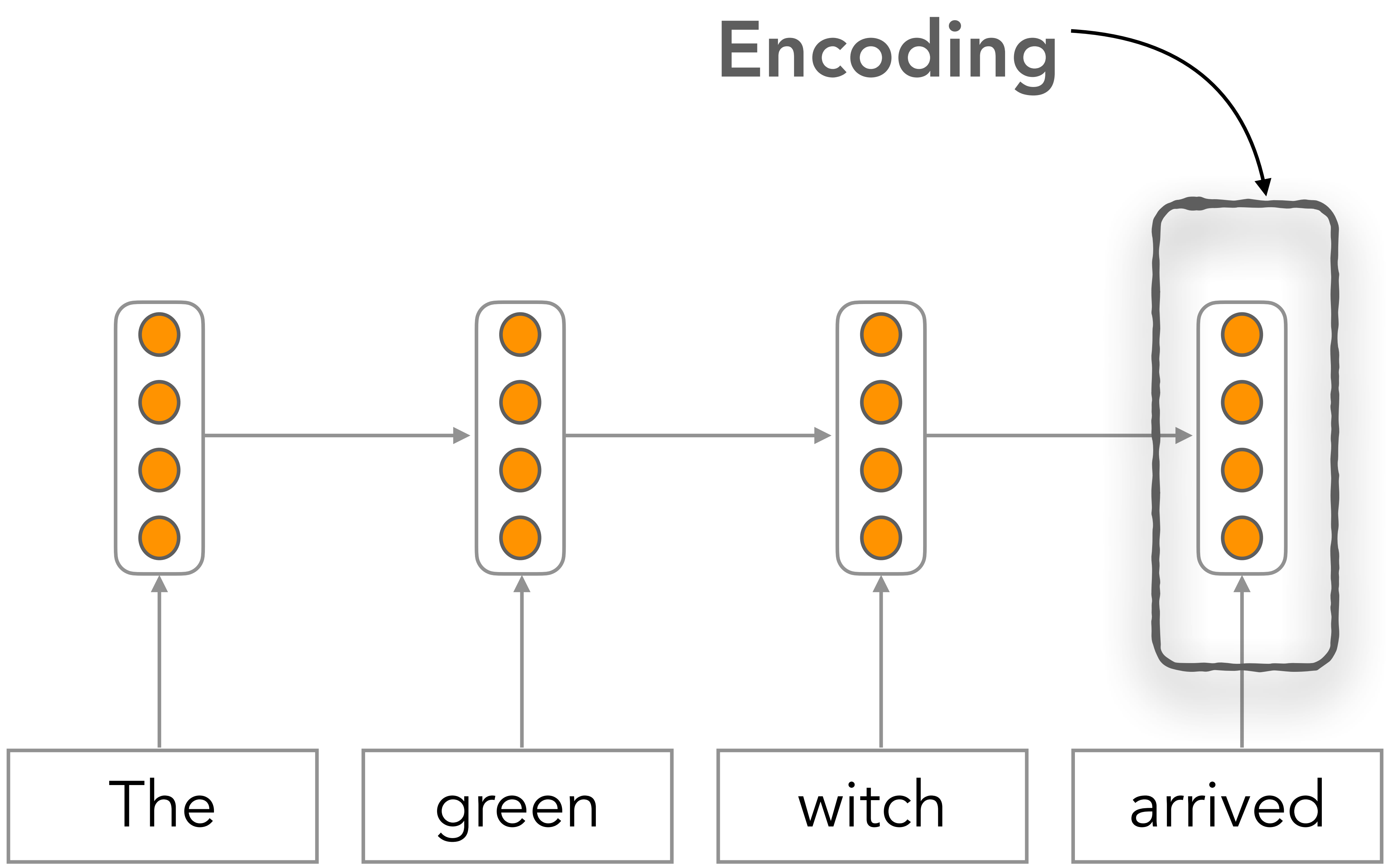


Source Sentence x

Produces an encoding of the source sequence

Represents input sequence. Provides initial hidden state for Decoder RNN

Encoder RNN



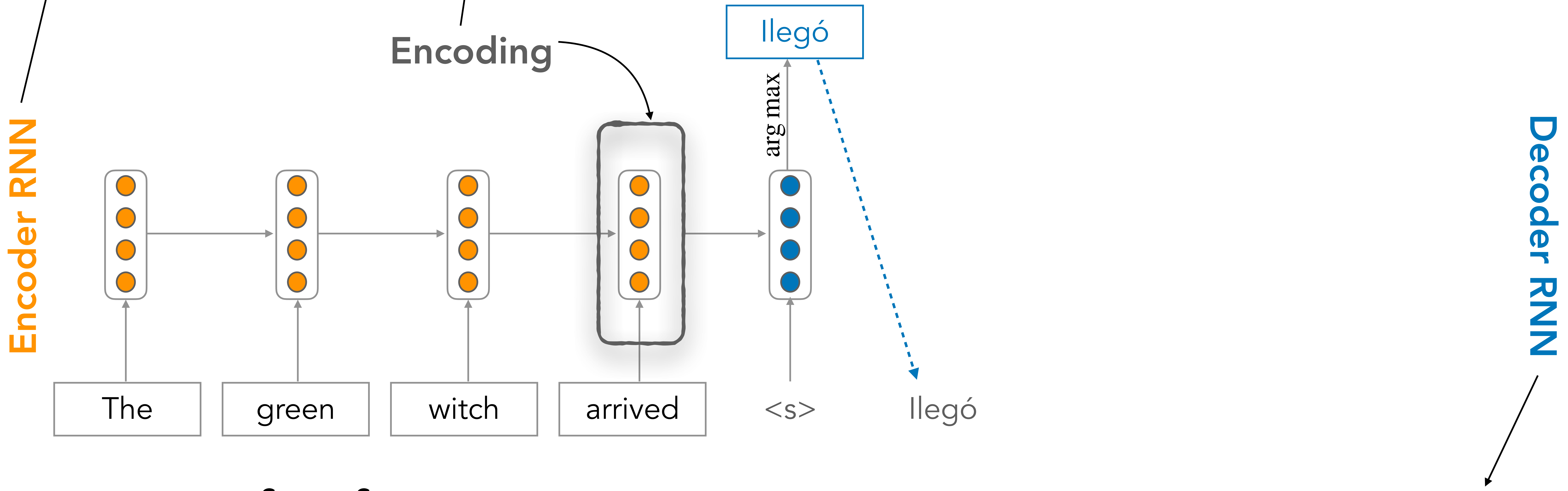
Source Sentence x

Decoder RNN

Language Model that produces the target sentence conditioned on the encoding

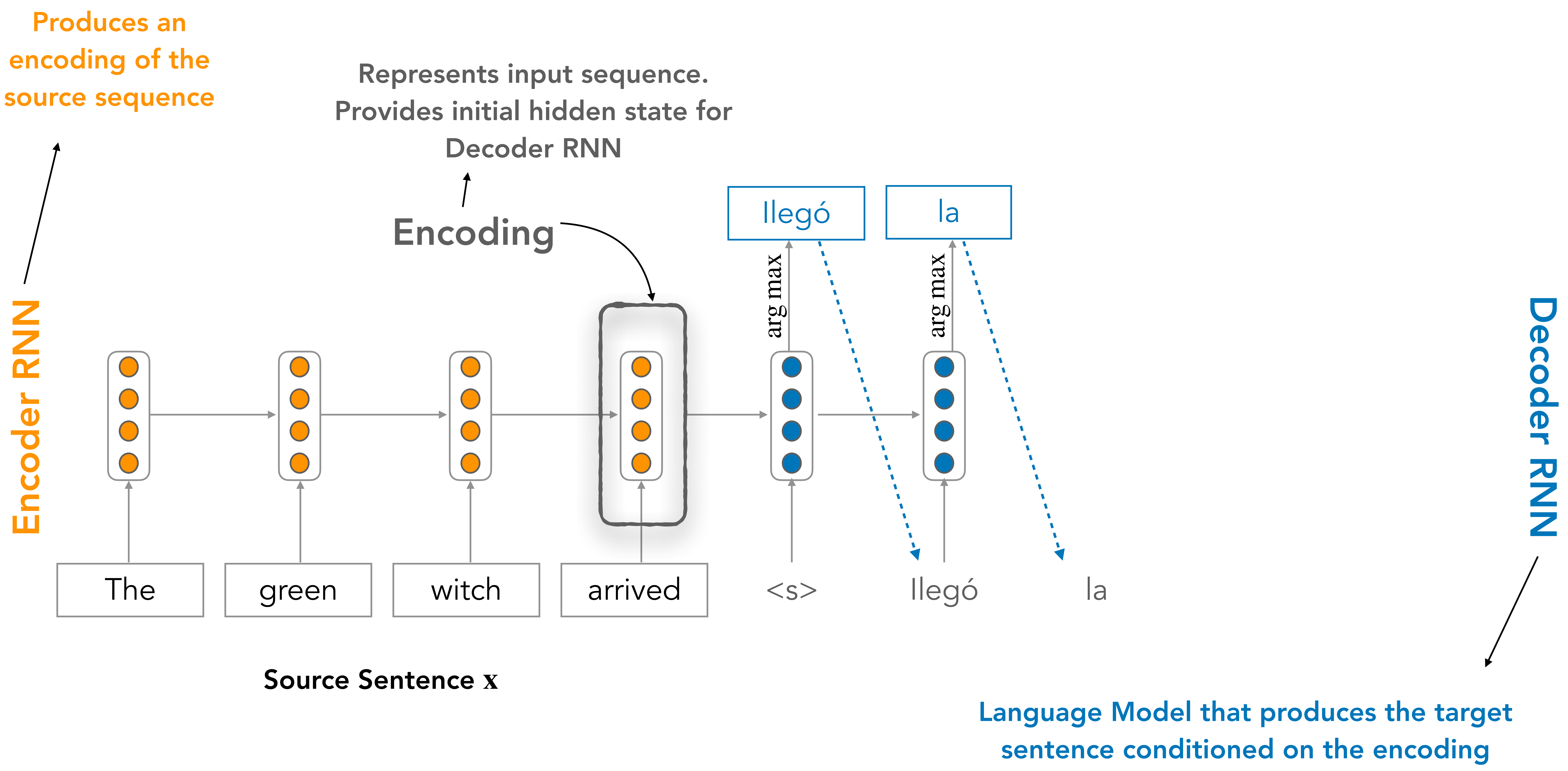
Produces an encoding of the source sequence

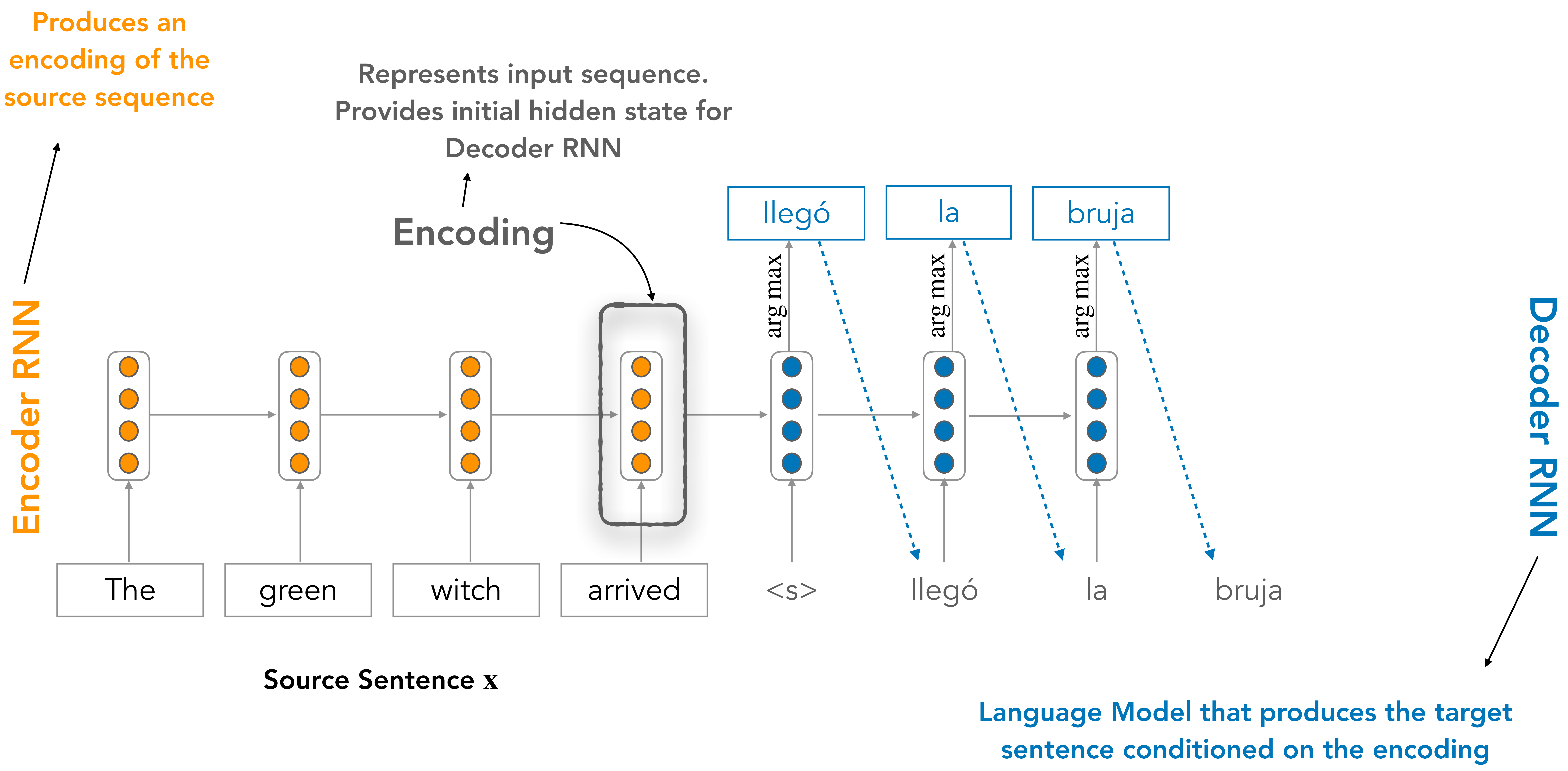
Represents input sequence. Provides initial hidden state for Decoder RNN



Source Sentence **x**

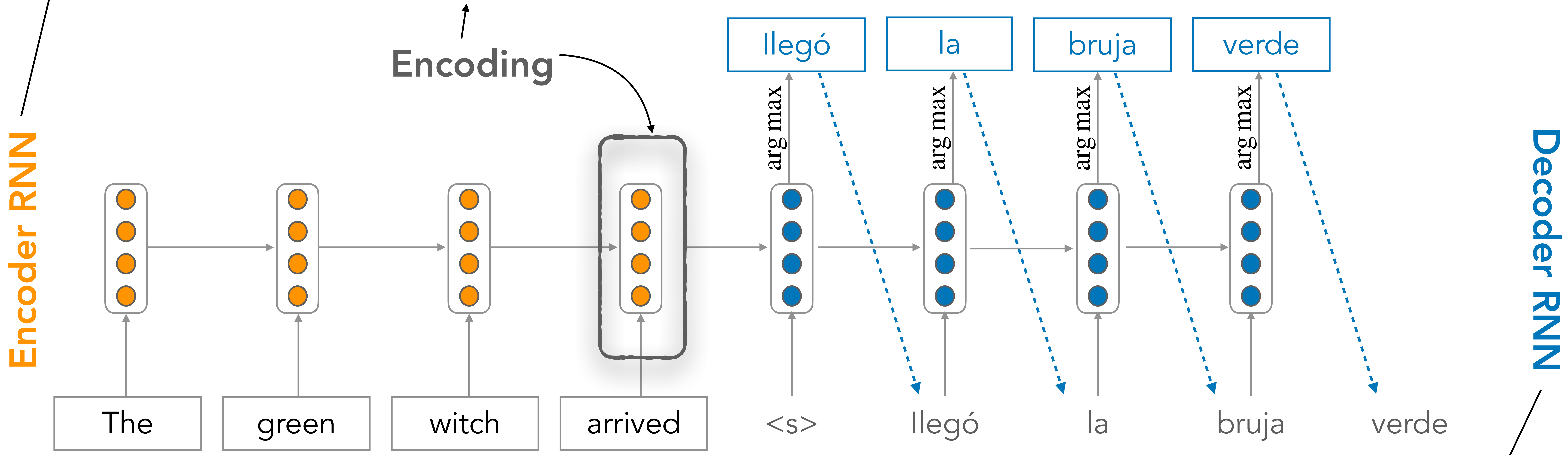
Language Model that produces the target sentence conditioned on the encoding


Produces an encoding of the source sequence
Represents input sequence. Provides initial hidden state for Decoder RNN
Encoding
Encoder RNN
Decoder RNN
llegó
la
arg max
arg max
The
green
witch
arrived
<s>
llegó
la
Source Sentence $\mathbf{x}$
Language Model that produces the target sentence conditioned on the encoding


Produces an encoding of the source sequence
Represents input sequence. Provides initial hidden state for Decoder RNN
Encoding
Encoder RNN
llegó
la
bruja
arg max
arg max
arg max
The
green
witch
arrived
<s>
llegó
la
bruja
Decoder RNN
Source Sentence $\mathbf{x}$
Language Model that produces the target sentence conditioned on the encoding

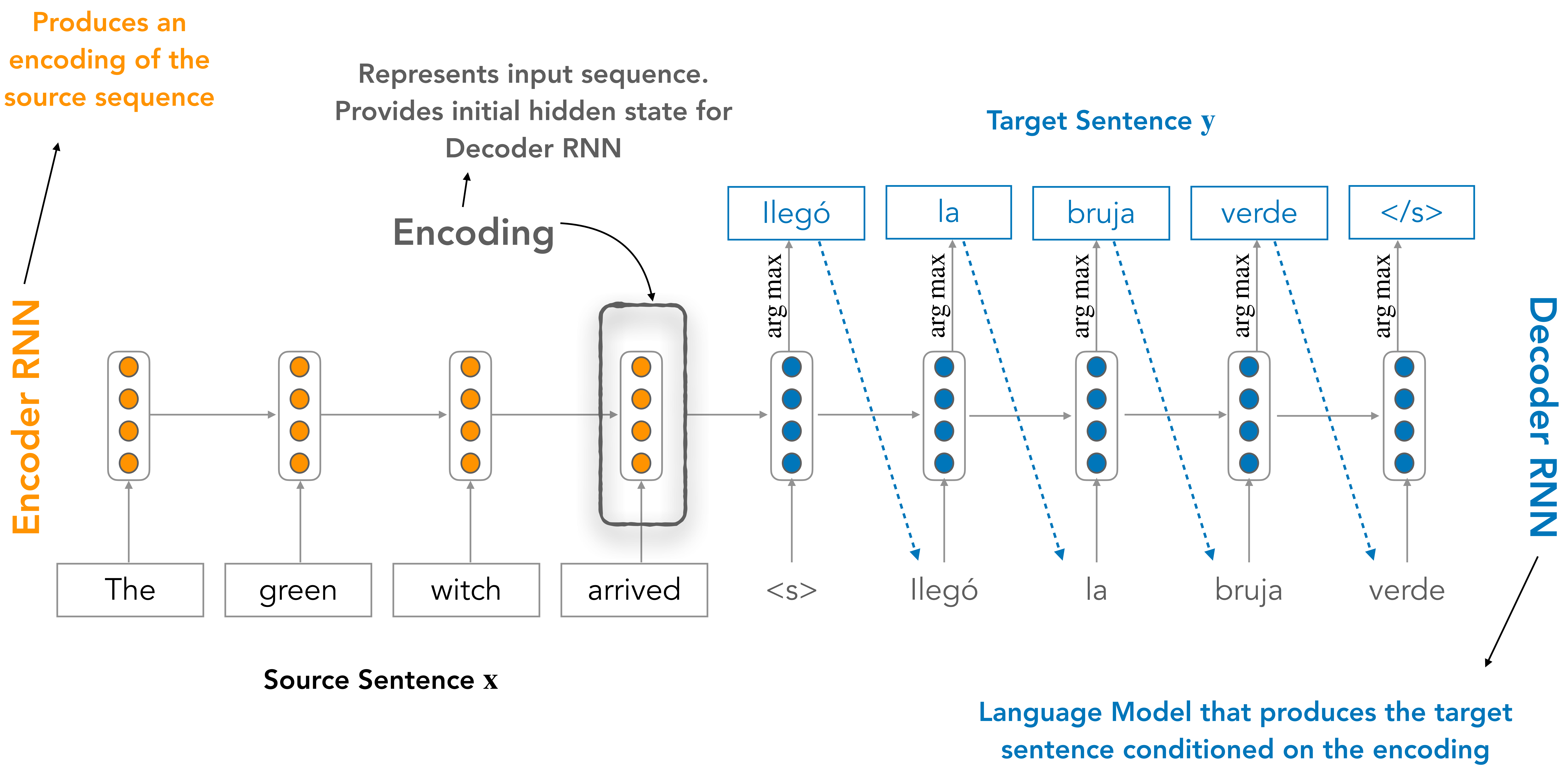
Produces an encoding of the source sequence

Represents input sequence. Provides initial hidden state for Decoder RNN



Source Sentence x

Language Model that produces the target sentence conditioned on the encoding

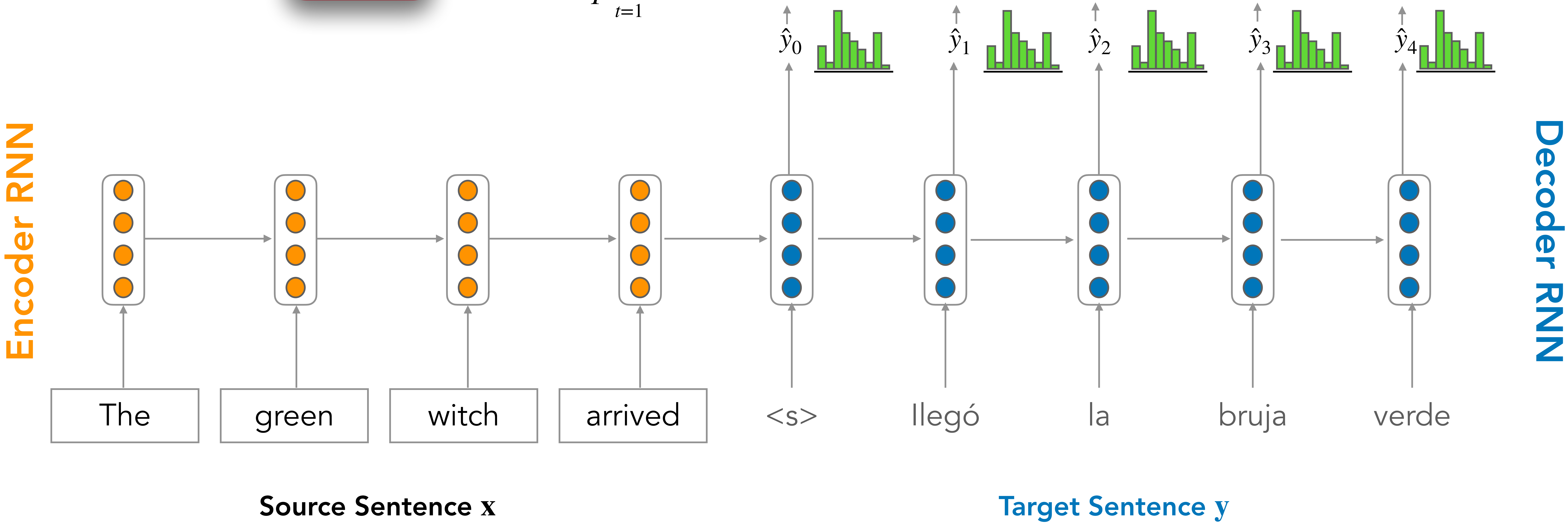

Produces an encoding of the source sequence
Represents input sequence. Provides initial hidden state for Decoder RNN
Target Sentence y
Encoding
llegó
la
bruja
verde
</s>
arg max
Encoder RNN
Decoder RNN
The
green
witch
arrived
<s>
llegó
la
bruja
verde
Source Sentence x
Language Model that produces the target sentence conditioned on the encoding

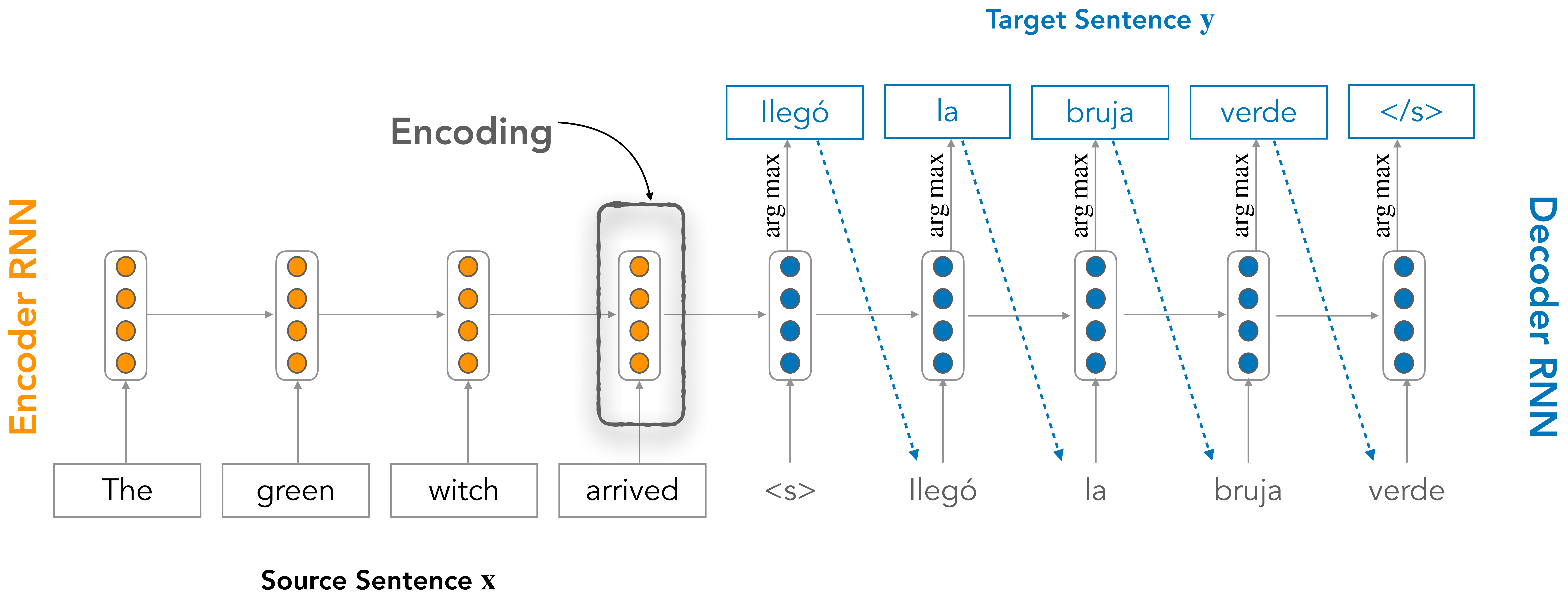
Loss

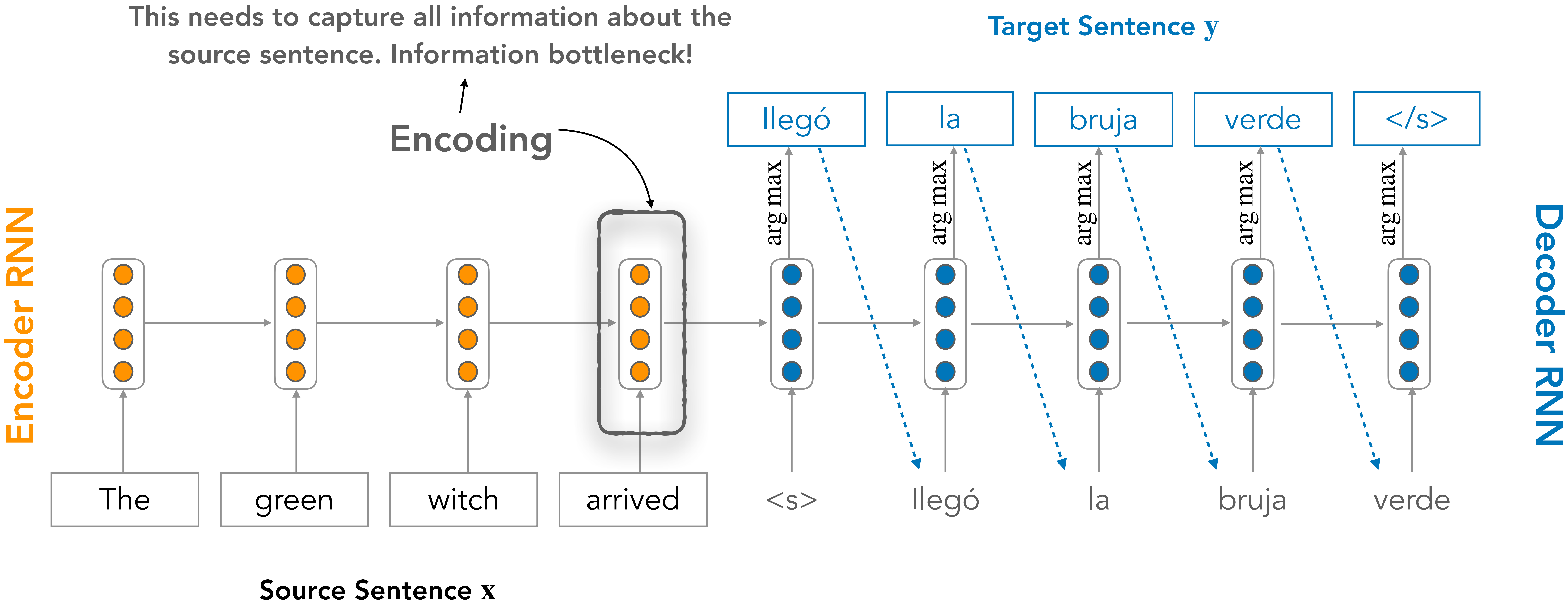
negative log prob. of "llegó"

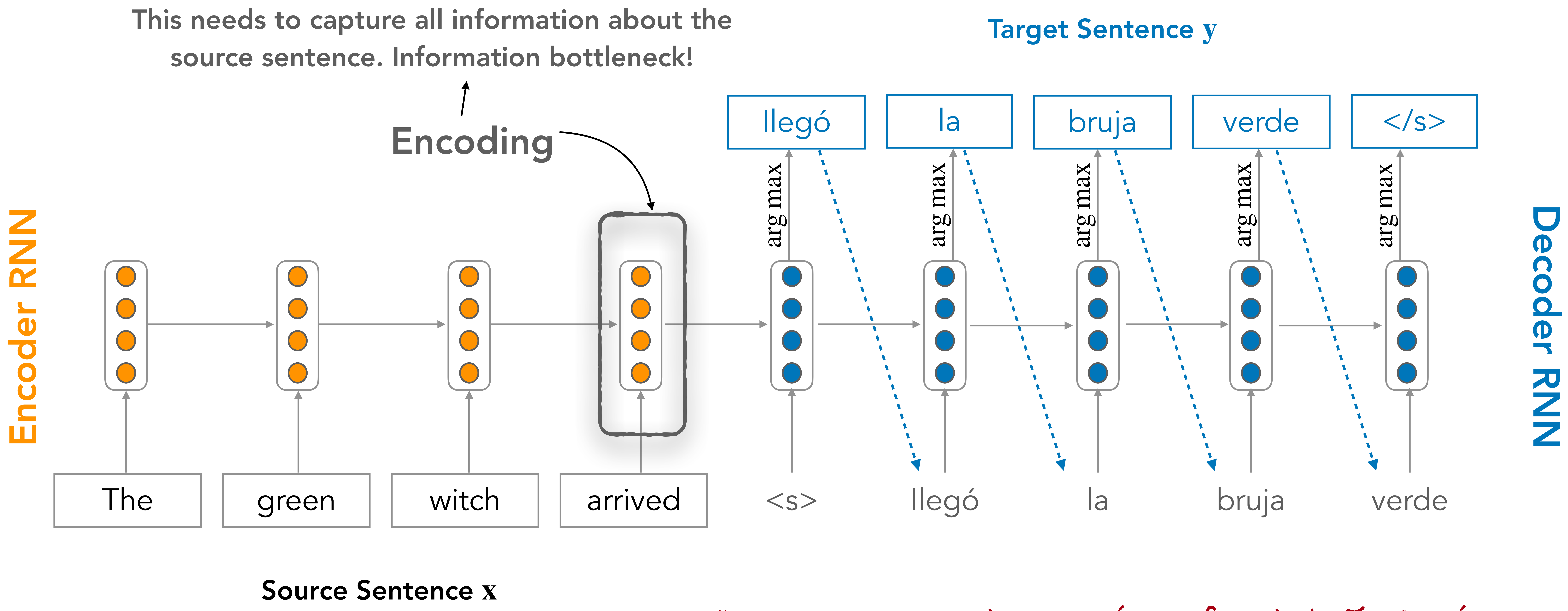
negative log prob. of "</s>"

$$L(\theta) = \frac{1}{T}\sum_{t=1}^{T} L_t(\theta) = L_0(\theta) + L_1(\theta) + L_2(\theta) + L_3(\theta) + L_4(\theta)$$




Target Sentence y
llegó
la
bruja
verde
</s>
arg max
Encoding
Encoder RNN
Decoder RNN
The
green
witch
arrived
<s>
llegó
la
bruja
verde
Source Sentence x

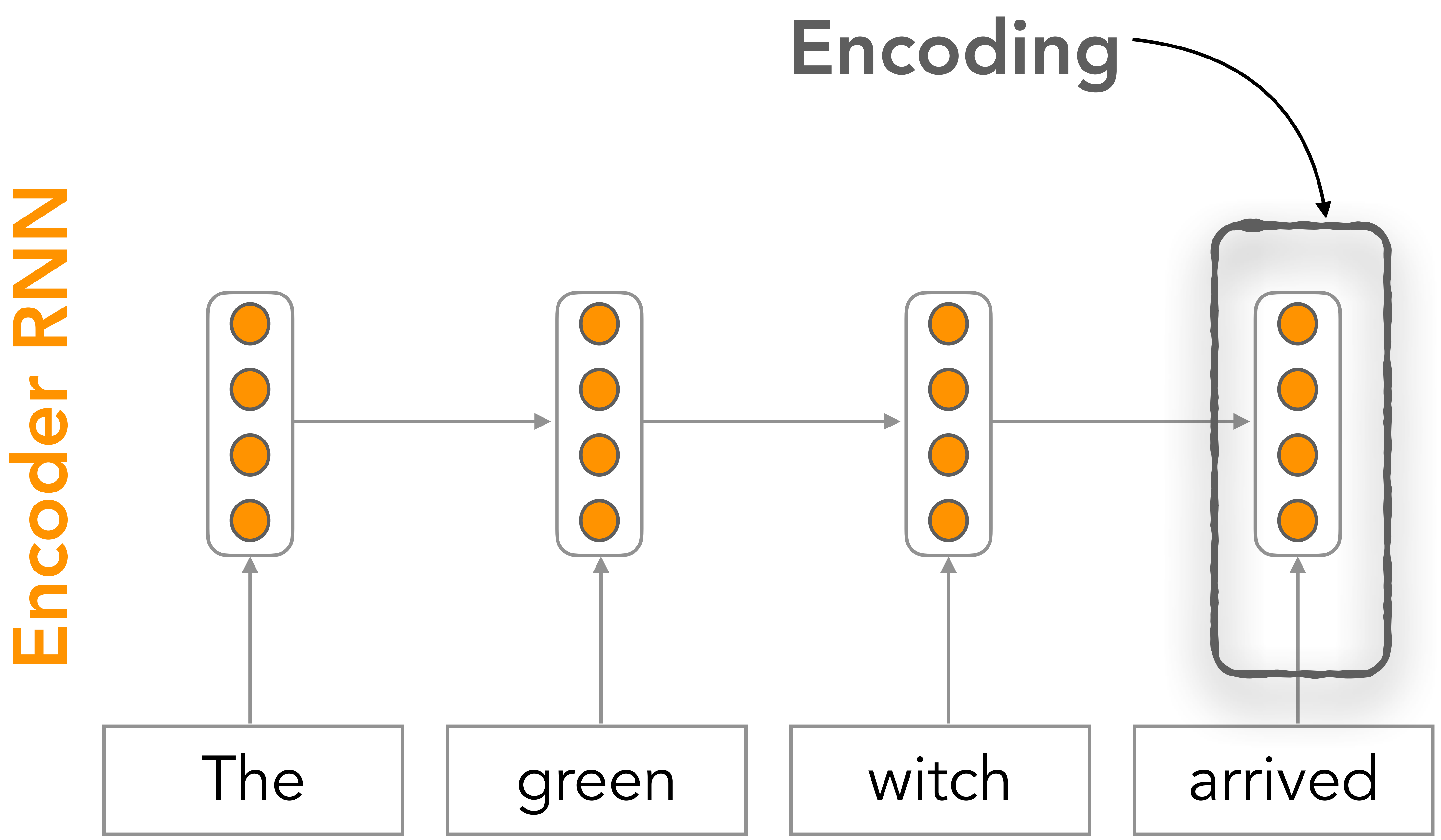

This needs to capture all information about the source sentence. Information bottleneck!
Target Sentence y
Encoding
llegó
la
bruja
verde
</s>
arg max
arg max
arg max
arg max
arg max
Encoder RNN
Decoder RNN
The
green
witch
arrived
<s>
llegó
la
bruja
verde
Source Sentence x



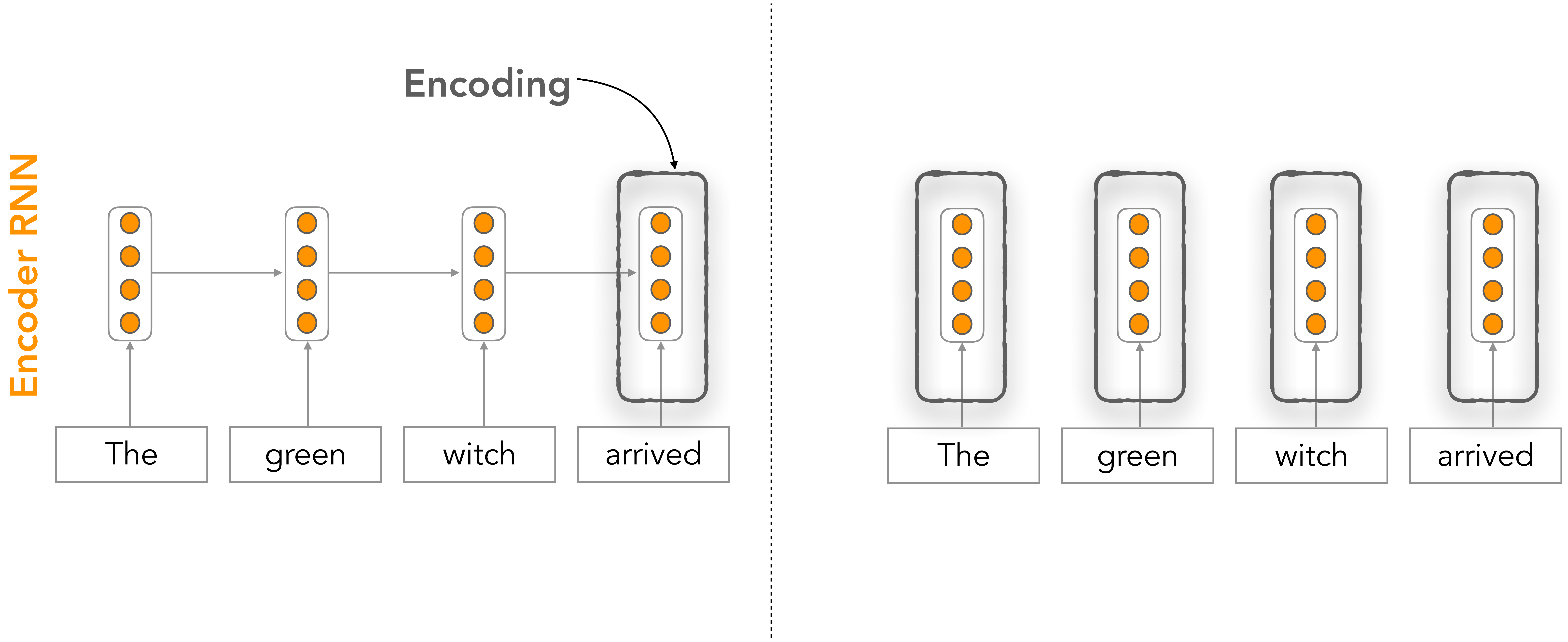
"you can't cram the meaning of a whole %&@#&ing sentence into a single $(&@ing vector!"*

– Ray Mooney, Professor of Computer Science, UT Austin

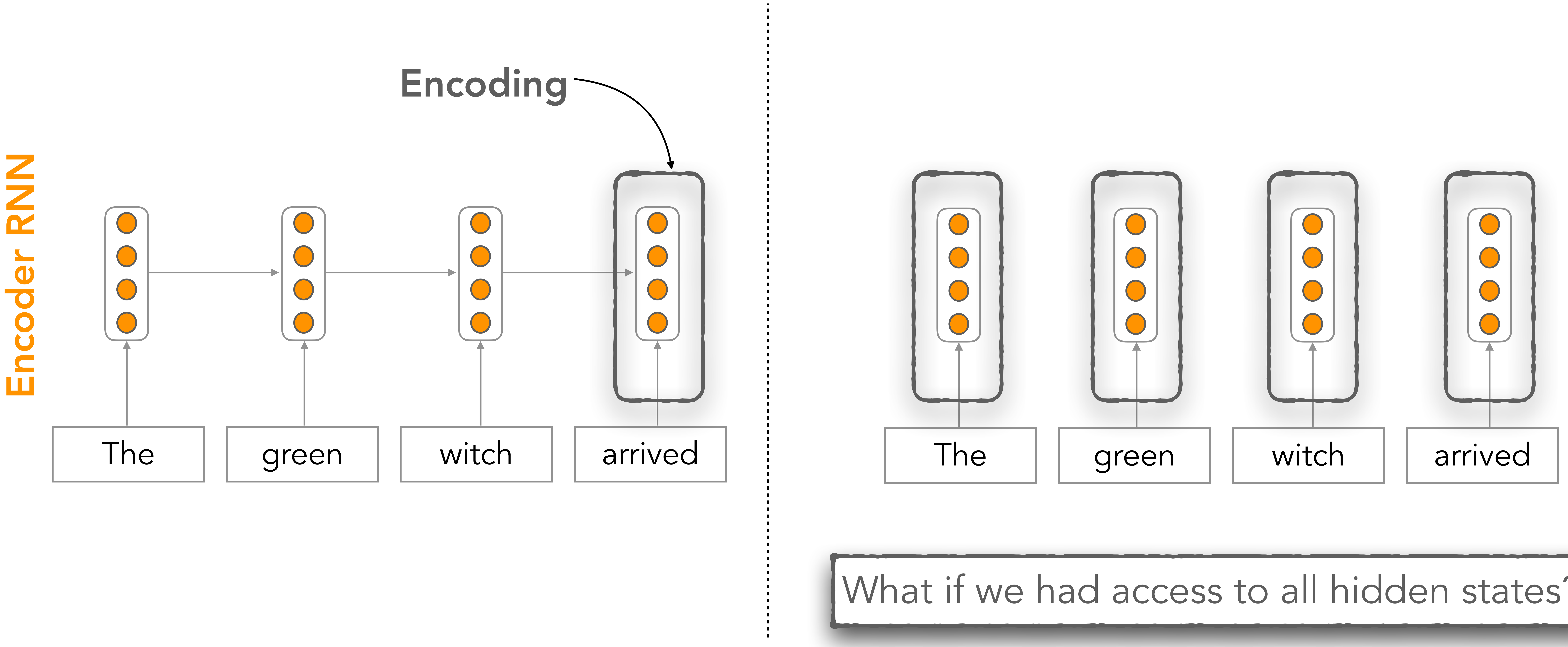
Information Bottleneck: One Solution



Information Bottleneck: One Solution



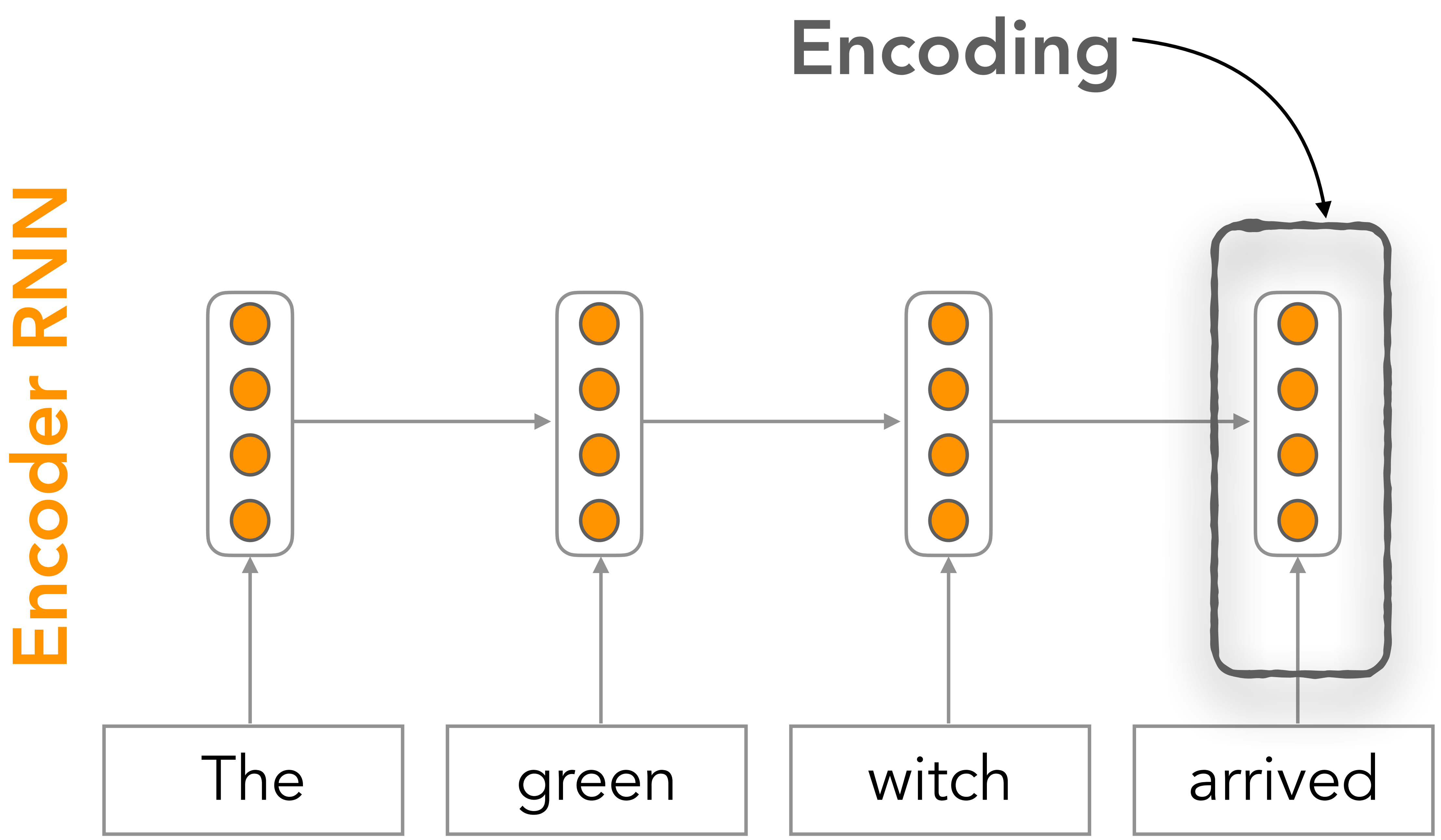
Information Bottleneck: One Solution

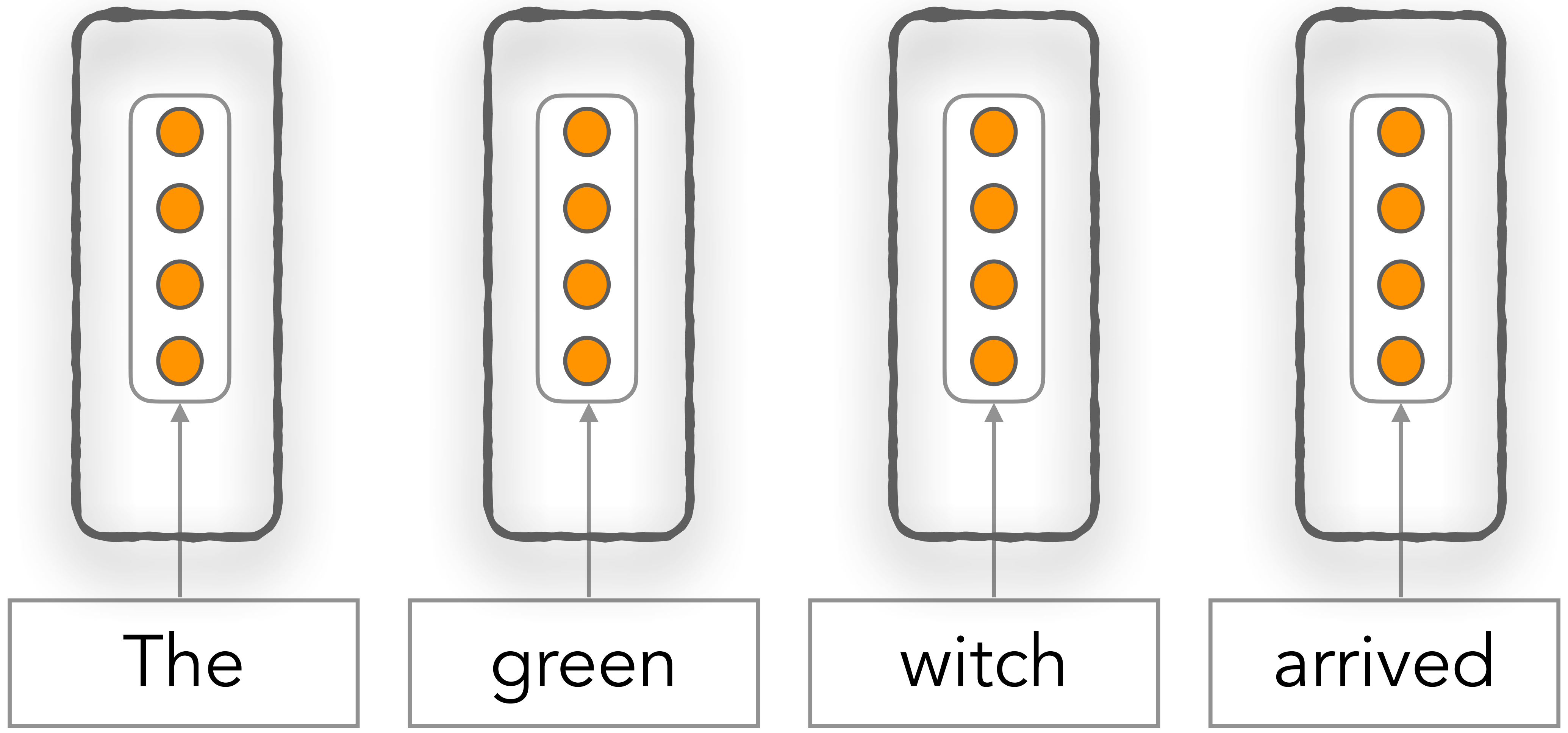


What if we had access to all hidden states?

USC Viterbi

Information Bottleneck: One Solution



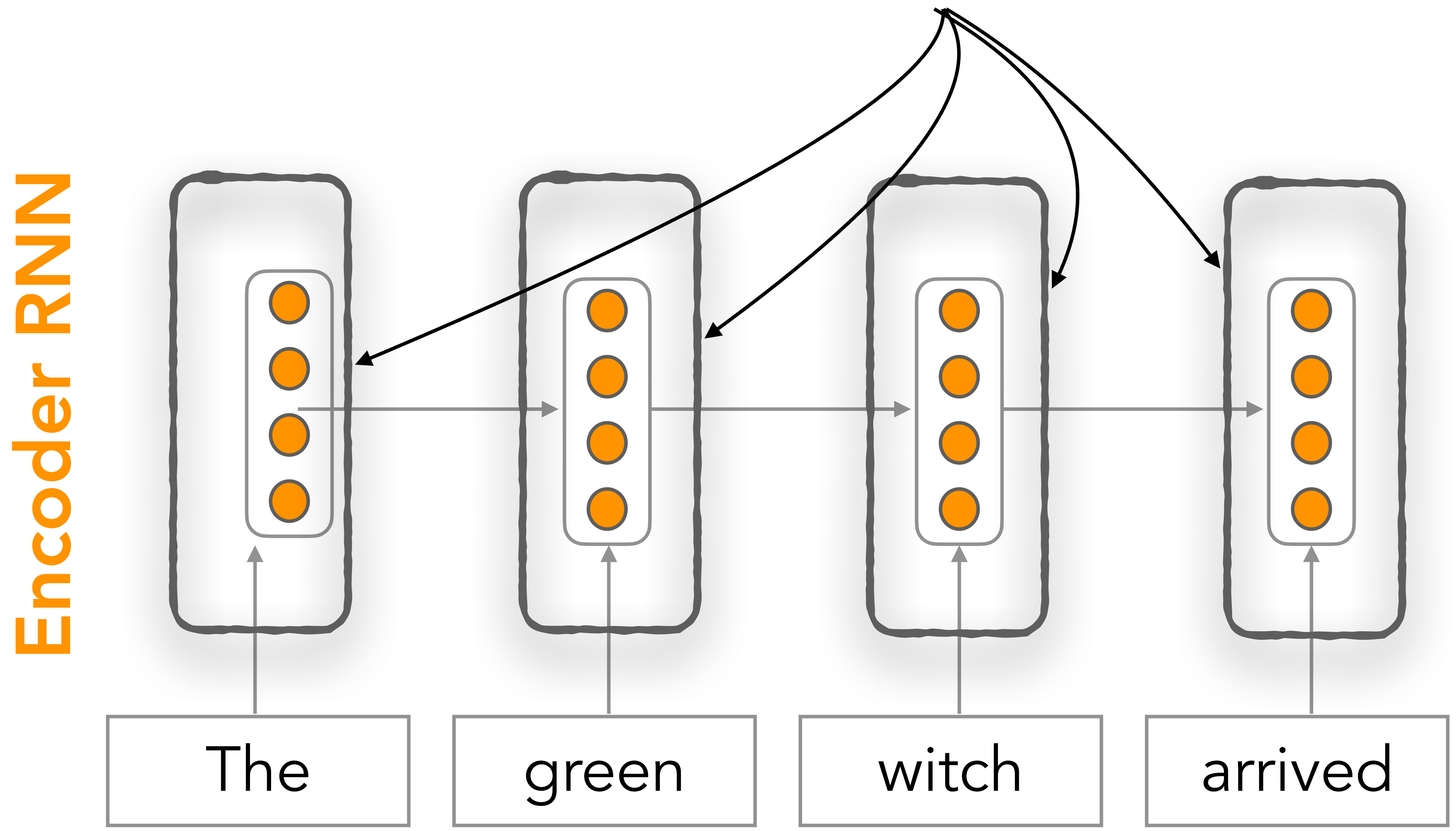


What if we had access to all hidden states?

How to create this?

Attention Mechanism

Encoding $\mathbf{c}_t^{att} = f(\mathbf{h}_1^e, \ldots, \mathbf{h}_N^e, \mathbf{h}_t^d)$



Source Sentence $\mathbf{x}$

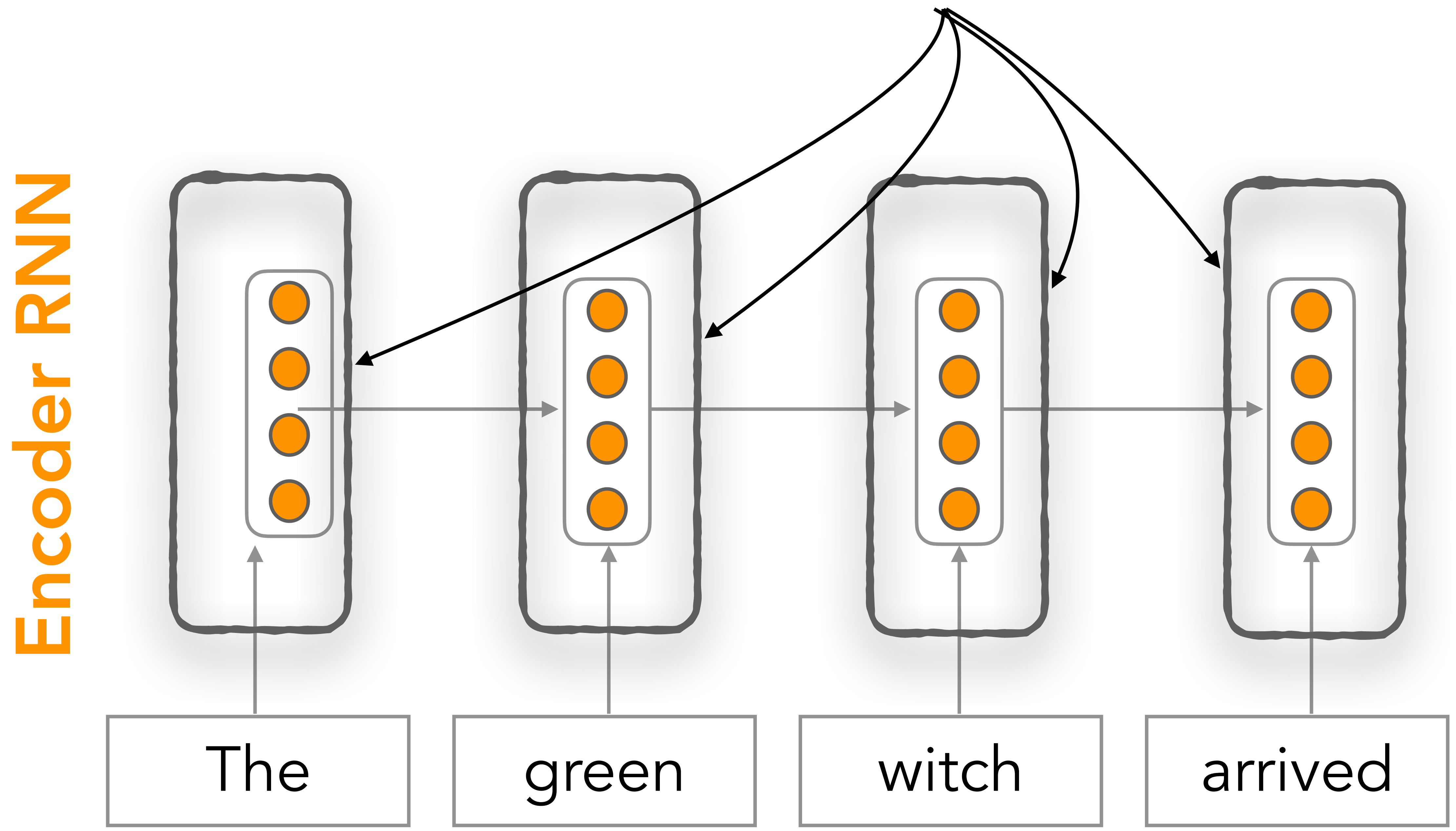
Note: Notation different from J&M

Bahdanau et al., 2015

Attention Mechanism

- Attention mechanisms allow the decoder to focus on a particular part of the source sequence at each time step

Encoding $\mathbf{c}_t^{att} = f(\mathbf{h}_1^e, \ldots, \mathbf{h}_N^e, \mathbf{h}_t^d)$



Source Sentence $\mathbf{x}$

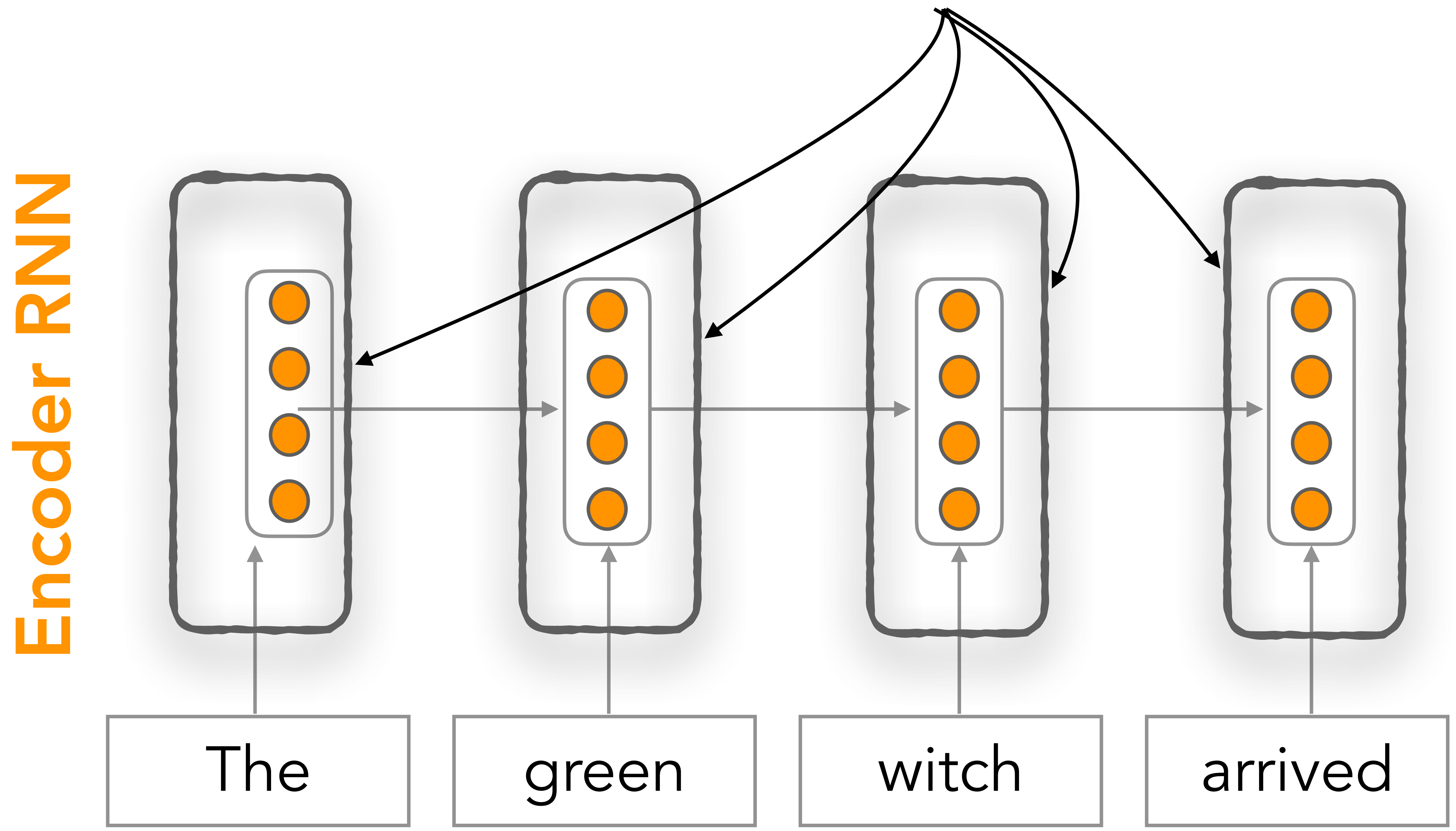
Bahdanau et al., 2015

Note: Notation different from J&M

Attention Mechanism

- Attention mechanisms allow the decoder to focus on a particular part of the source sequence at each time step
- Fixed-length vector $\mathbf{c}_t^{att}$ (attention context vector)

Encoding $\mathbf{c}_t^{att} = f(\mathbf{h}_1^e, \ldots, \mathbf{h}_N^e, \mathbf{h}_t^d)$



Source Sentence $\mathbf{x}$

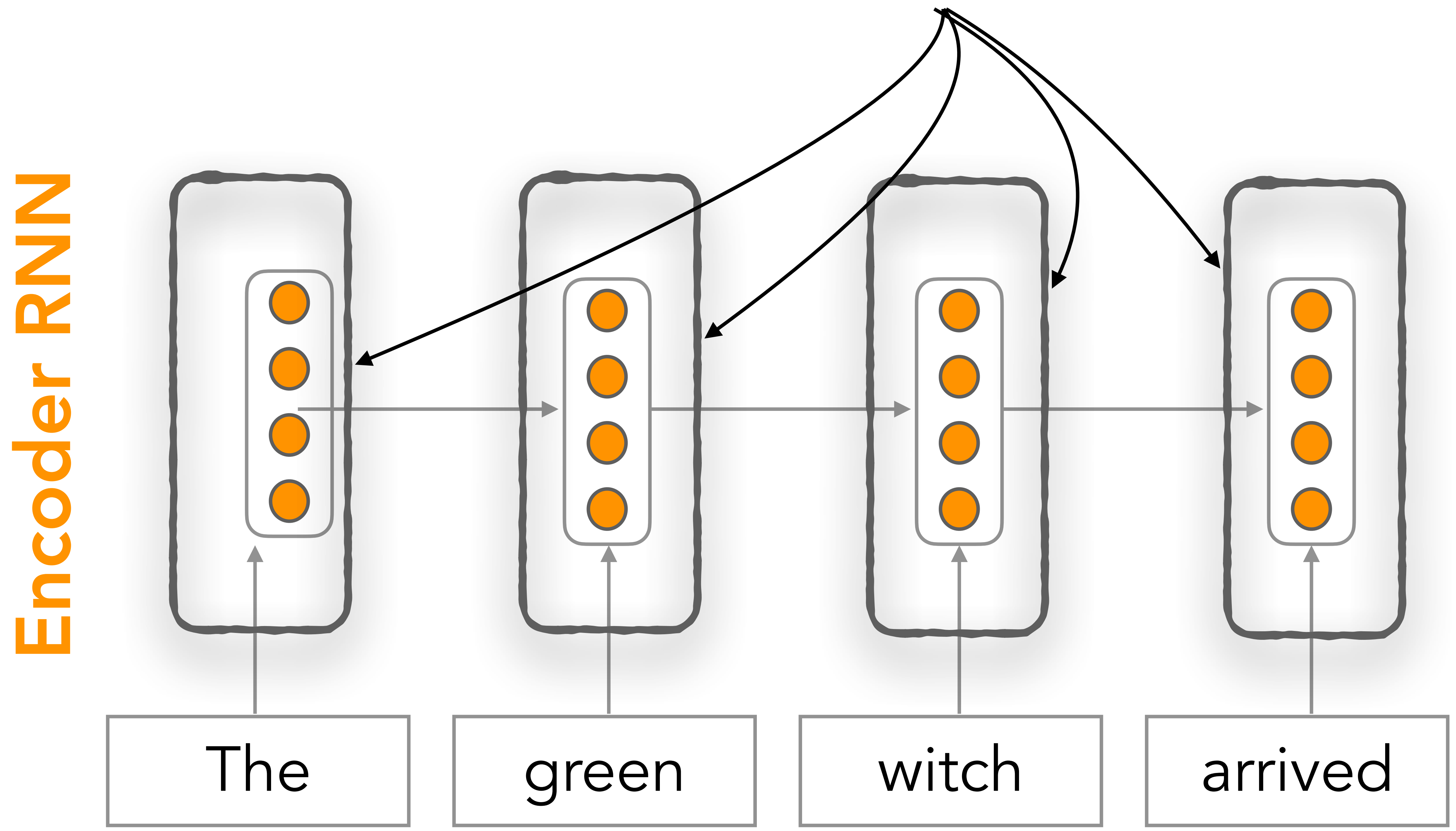
Bahdanau et al., 2015

Note: Notation different from J&M

Attention Mechanism

- Attention mechanisms allow the decoder to focus on a particular part of the source sequence at each time step
- Fixed-length vector $\mathbf{c}_t^{att}$ (attention context vector)
  - Take a weighted sum of all the encoder hidden states

Encoding $\mathbf{c}_t^{att} = f(\mathbf{h}_1^e, \ldots, \mathbf{h}_N^e, \mathbf{h}_t^d)$



Source Sentence $\mathbf{x}$

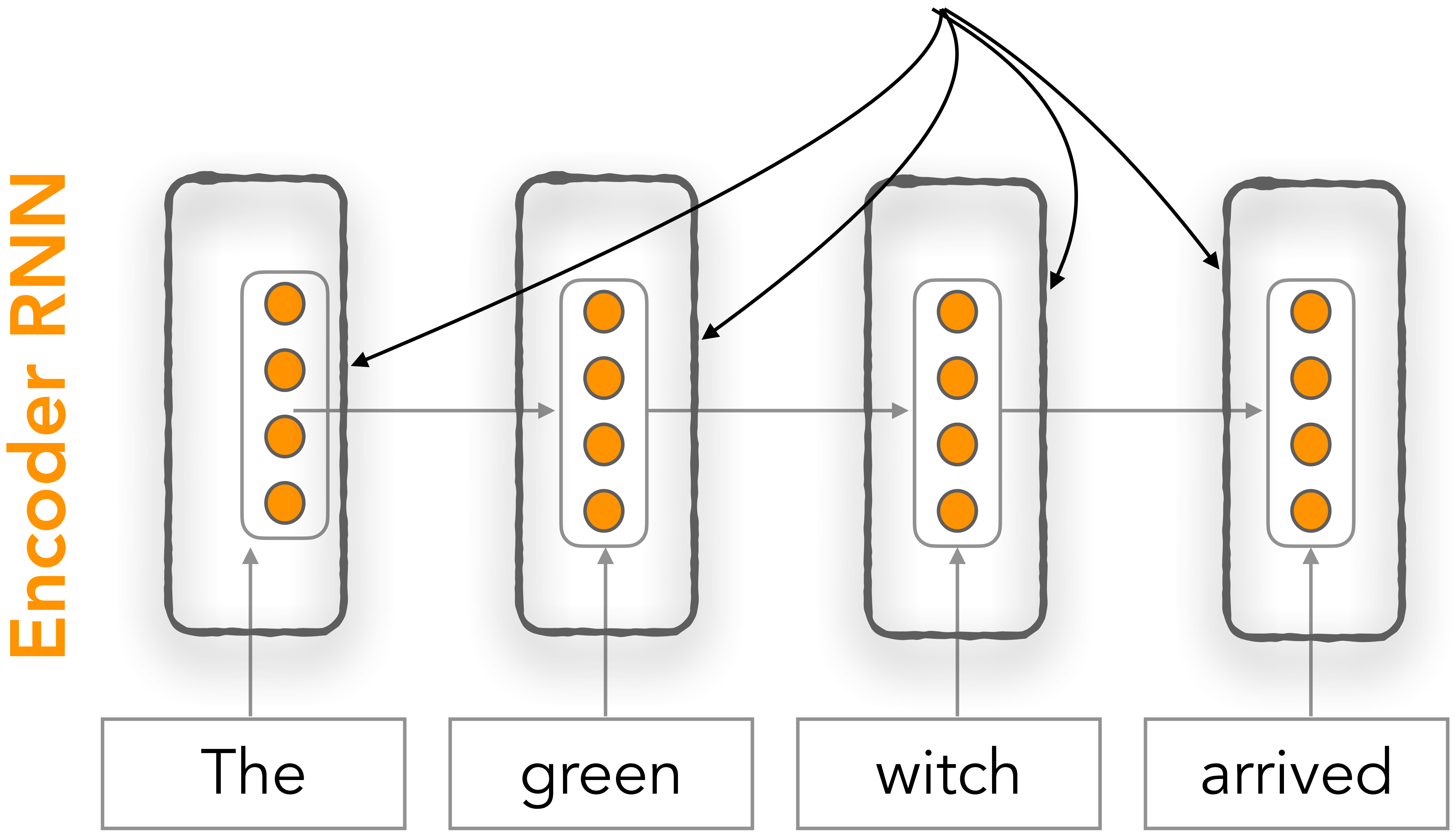
Note: Notation different from J&M

Bahdanau et al., 2015

Attention Mechanism

- Attention mechanisms allow the decoder to focus on a particular part of the source sequence at each time step
- Fixed-length vector $\mathbf{c}_t^{att}$ (attention context vector)
  - Take a weighted sum of all the encoder hidden states
  - One vector per time step *of the decoder*!

Encoding $\mathbf{c}_t^{att} = f(\mathbf{h}_1^e, \ldots, \mathbf{h}_N^e, \mathbf{h}_t^d)$



Source Sentence $\mathbf{x}$

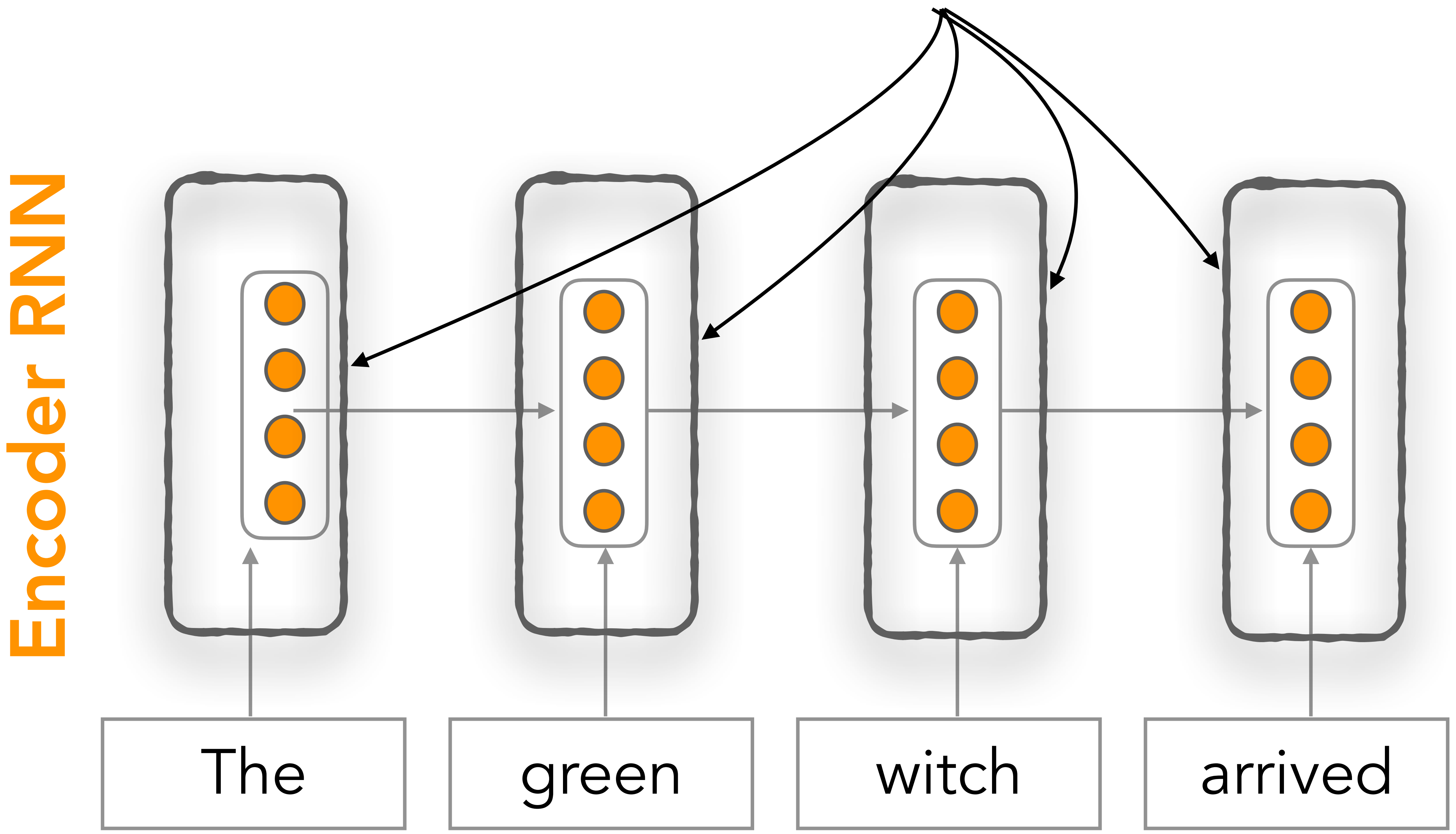
Note: Notation different from J&M

Bahdanau et al., 2015

Attention Mechanism

- Attention mechanisms allow the decoder to focus on a particular part of the source sequence at each time step
- Fixed-length vector $\mathbf{c}_t^{att}$ (attention context vector)
    - Take a weighted sum of all the encoder hidden states
    - One vector per time step *of the decoder*!
    - Weights *attend to* part of the source text relevant for the token the decoder is producing at step t

Encoding $\mathbf{c}_t^{att} = f(\mathbf{h}_1^e, \ldots, \mathbf{h}_N^e, \mathbf{h}_t^d)$



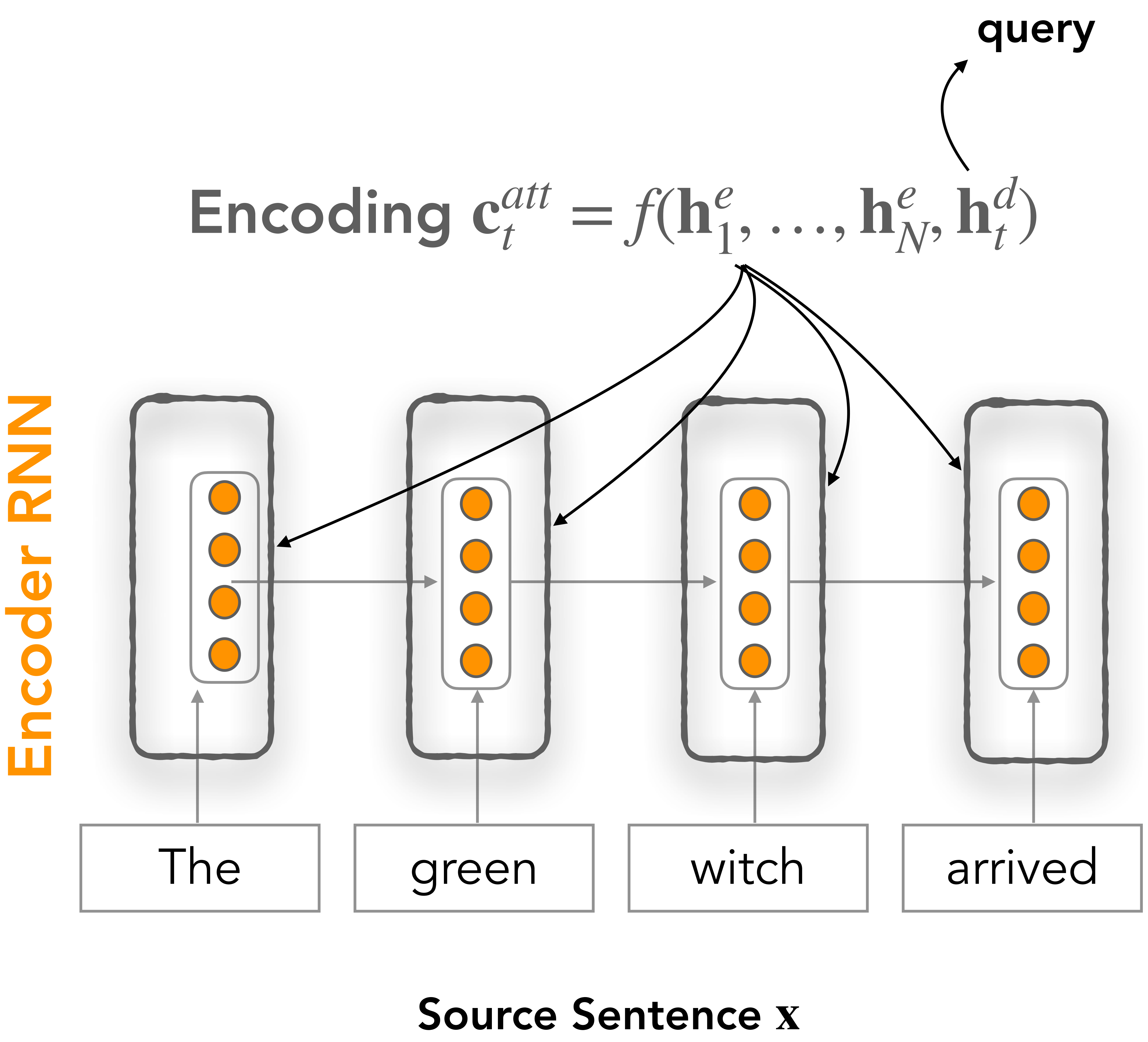
Source Sentence $\mathbf{x}$

Note: Notation different from J&M

Bahdanau et al., 2015

Attention Mechanism

- Attention mechanisms allow the decoder to focus on a particular part of the source sequence at each time step
- Fixed-length vector $\mathbf{c}_t^{att}$ (attention context vector)
  - Take a weighted sum of all the encoder hidden states
  - One vector per time step *of the decoder*!
  - Weights *attend to* part of the source text relevant for the token the decoder is producing at step t
- In general, we have a single **query** vector and multiple **key** vectors.

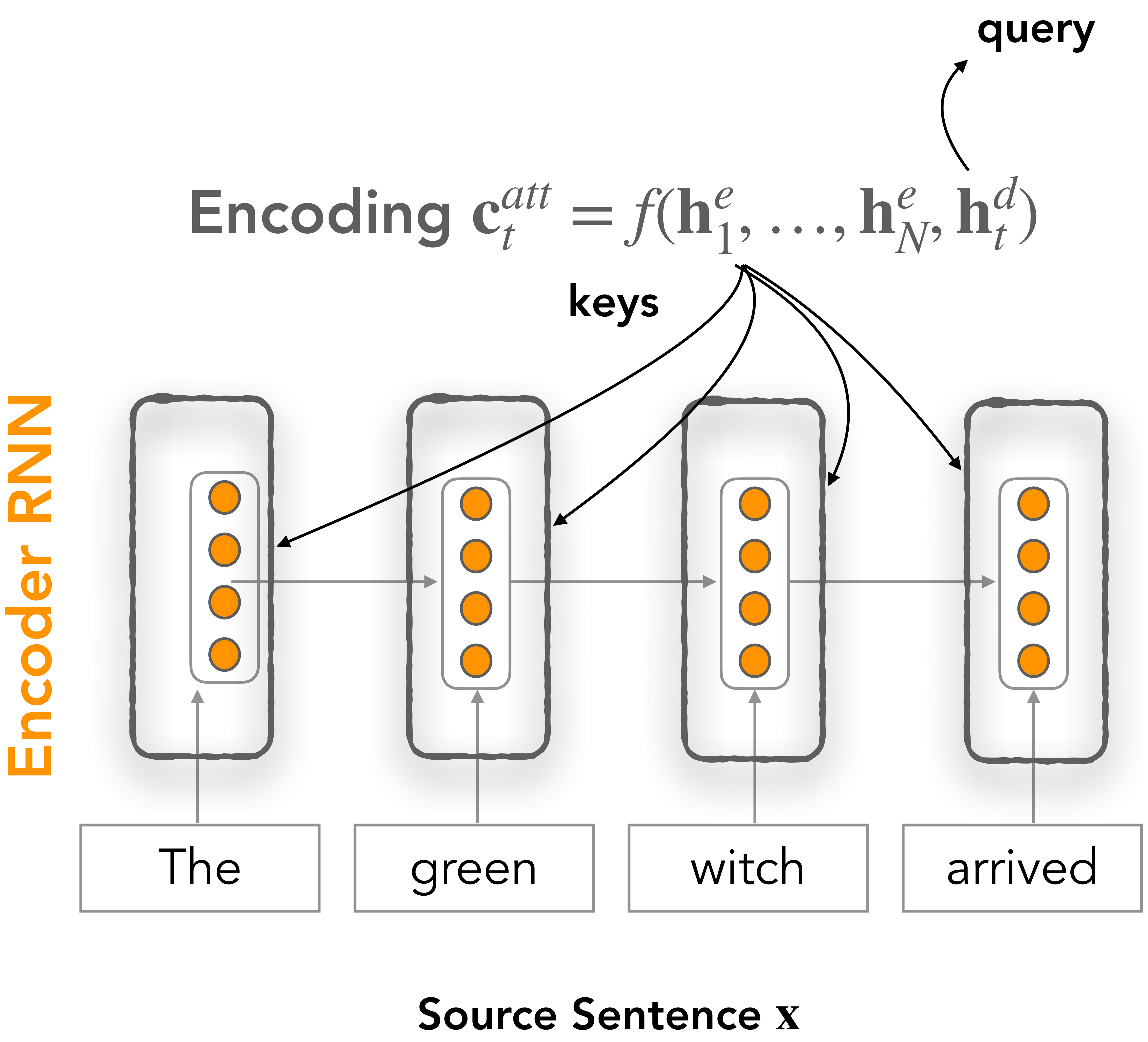


Note: Notation different from J&M

Bahdanau et al., 2015

Attention Mechanism

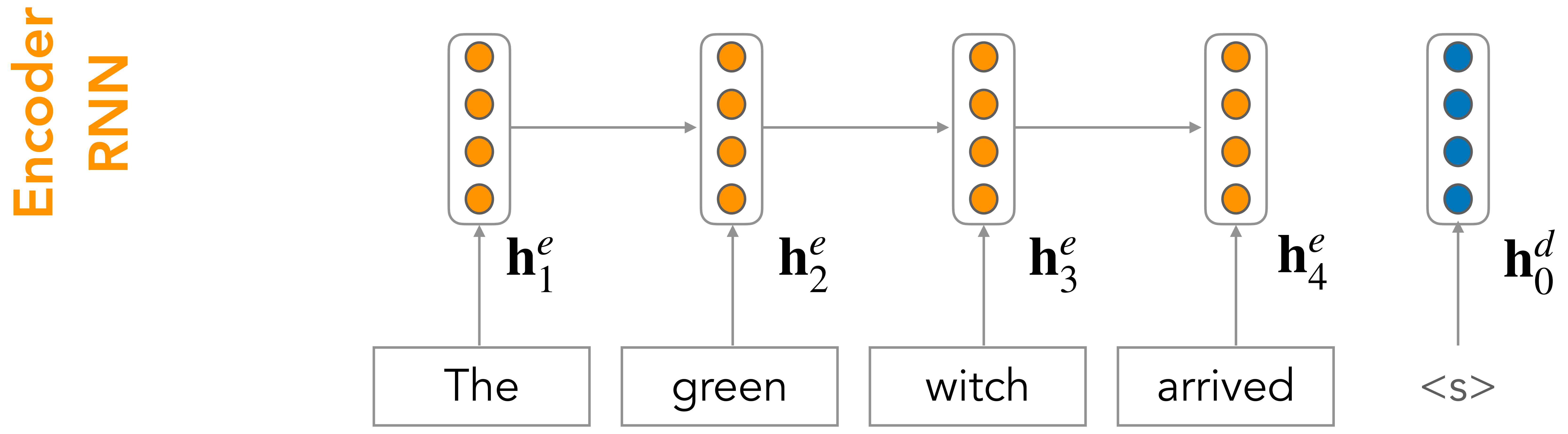
- Attention mechanisms allow the decoder to focus on a particular part of the source sequence at each time step
- Fixed-length vector $\mathbf{c}_t^{att}$ (attention context vector)
  - Take a weighted sum of all the encoder hidden states
  - One vector per time step *of the decoder*!
  - Weights *attend to* part of the source text relevant for the token the decoder is producing at step t
- In general, we have a single **query** vector and multiple **key** vectors.
  - We want to score each query-key pair



Bahdanau et al., 2015

Note: Notation different from J&M

Seq2Seq with Attention

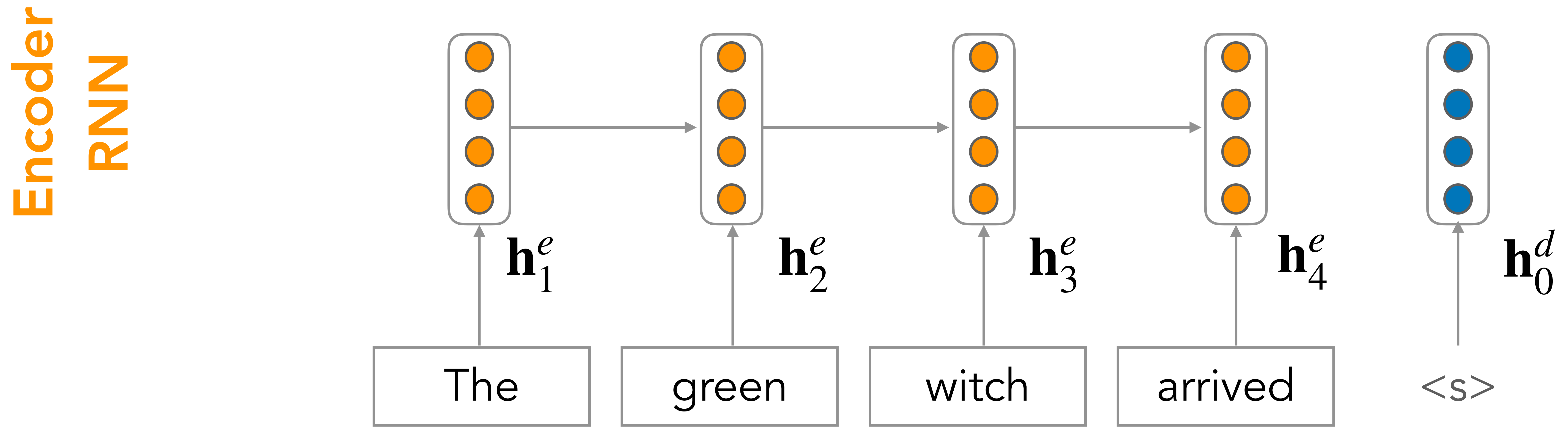


Source Sentence $\mathbf{x}$

Note: Notation different from J&M

Seq2Seq with Attention

Query 1: Decoder, first time step



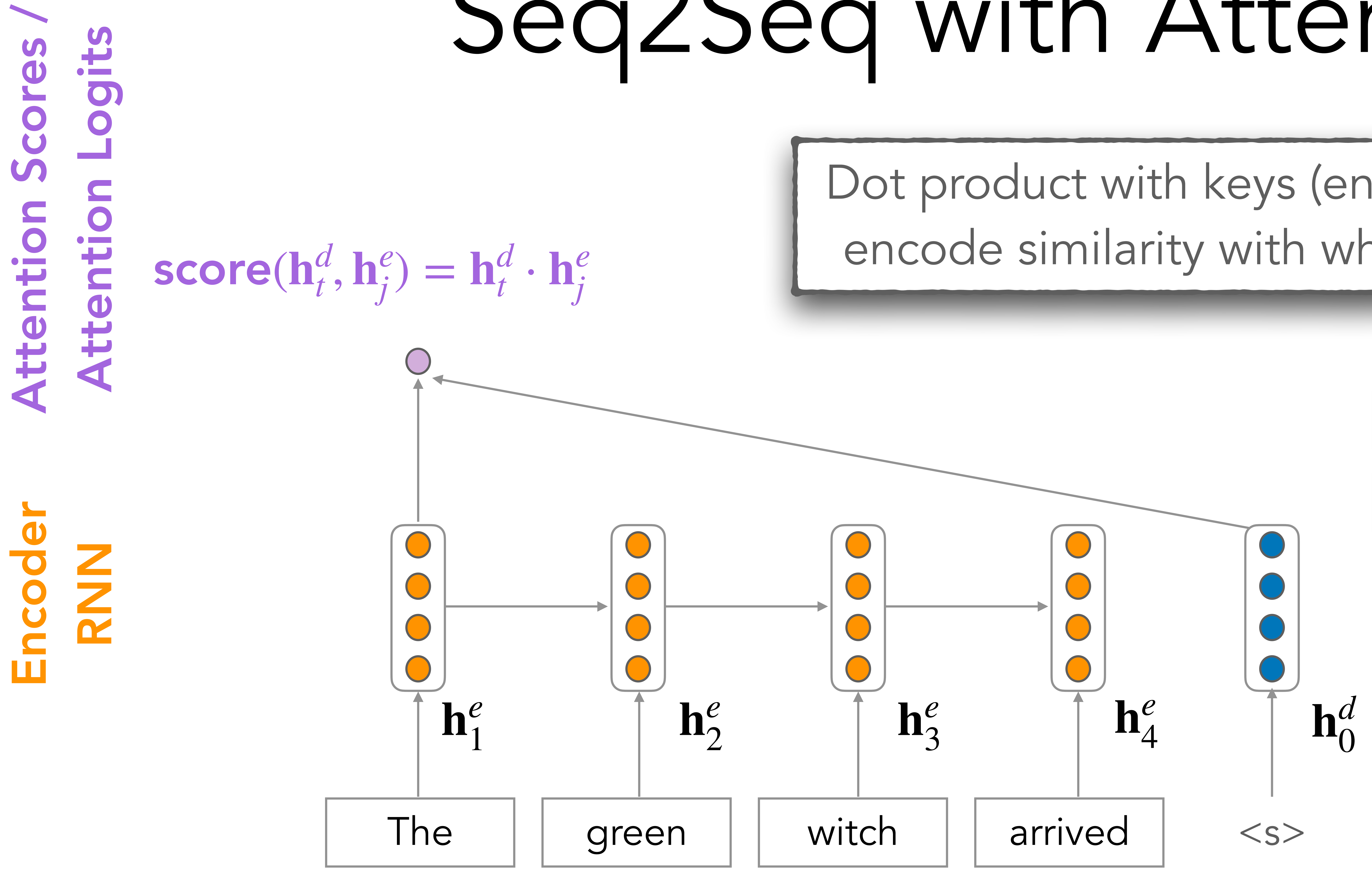
Source Sentence $\mathbf{x}$

Note: Notation different from J&M

Seq2Seq with Attention

Dot product with keys (encoder hidden states) to encode similarity with what is decoded so far…

Query 1: Decoder, first time step



Source Sentence $\mathbf{x}$

Note: Notation different from J&M

Seq2Seq with Attention

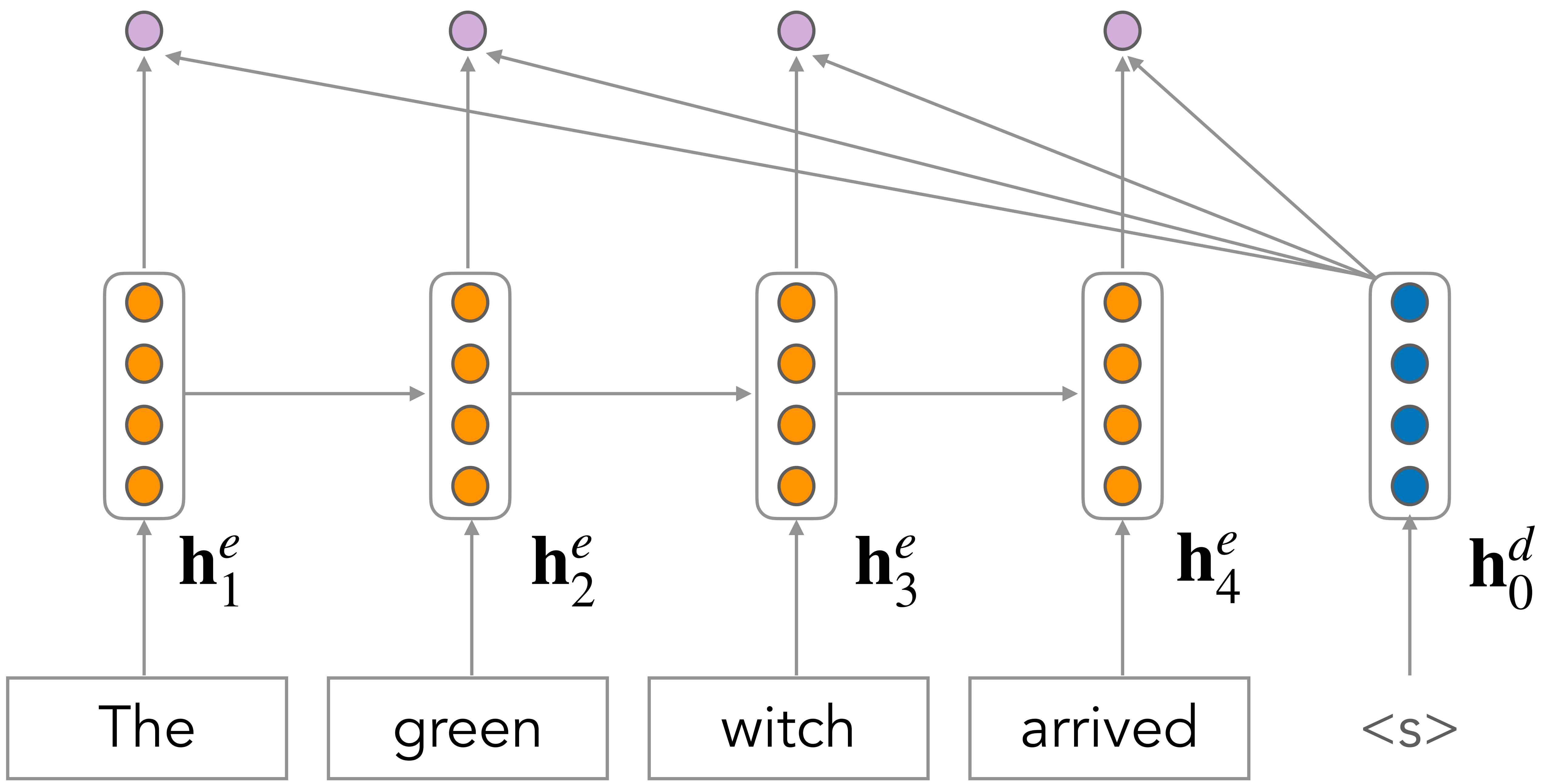
Attention Scores / Attention Logits

$$\text{score}(\mathbf{h}_t^d, \mathbf{h}_j^e) = \mathbf{h}_t^d \cdot \mathbf{h}_j^e$$

Dot product with keys (encoder hidden states) to encode similarity with what is decoded so far…

Query 1: Decoder, first time step

Encoder RNN



Source Sentence $\mathbf{x}$

Note: Notation different from J&M

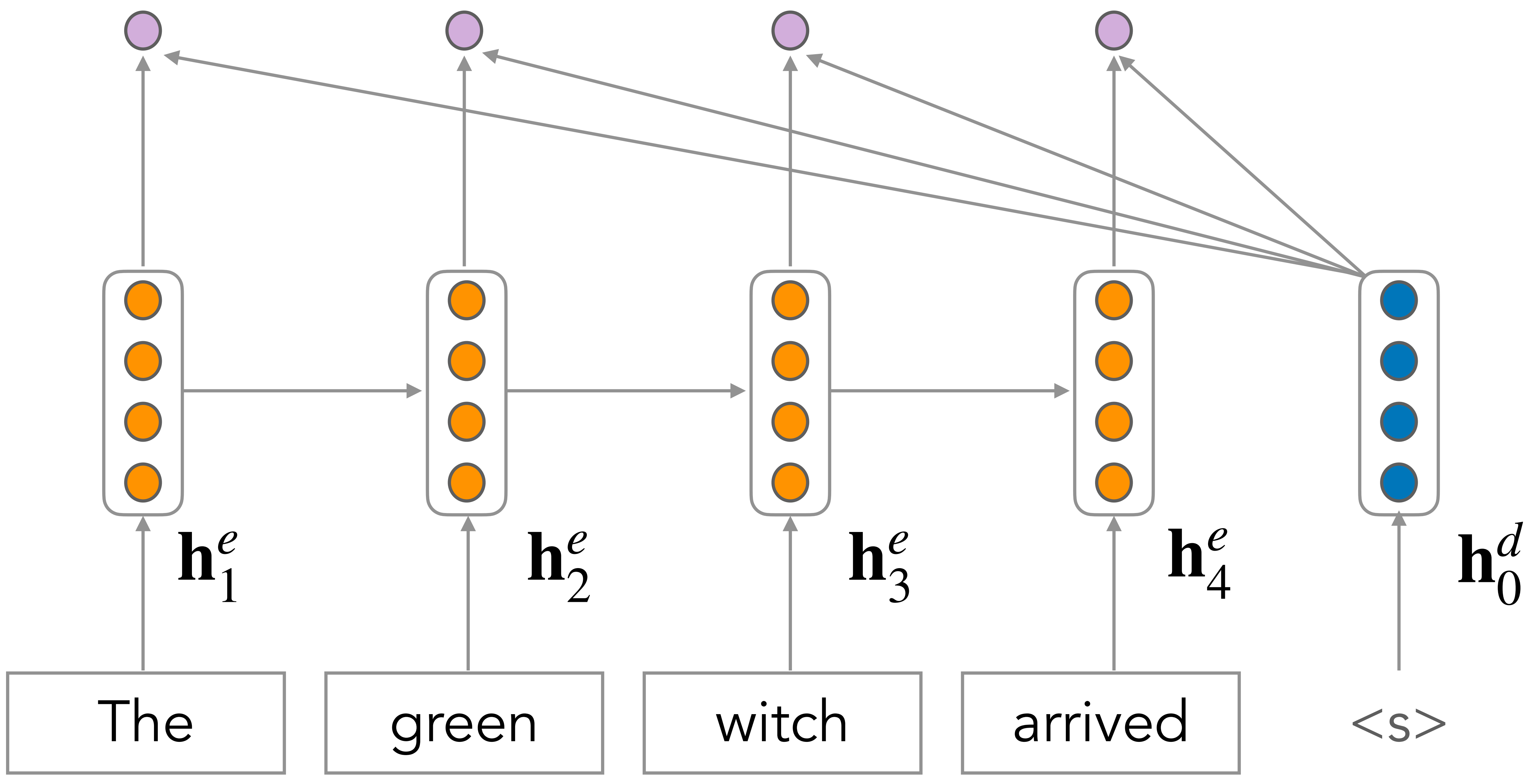
Seq2Seq with Attention

Attention Scores / Attention Logits

$\mathrm{score}(\mathbf{h}_t^d, \mathbf{h}_j^e) = \mathbf{h}_t^d \cdot \mathbf{h}_j^e$

Dot product with keys (encoder hidden states) to encode similarity with what is decoded so far…

Encoder RNN

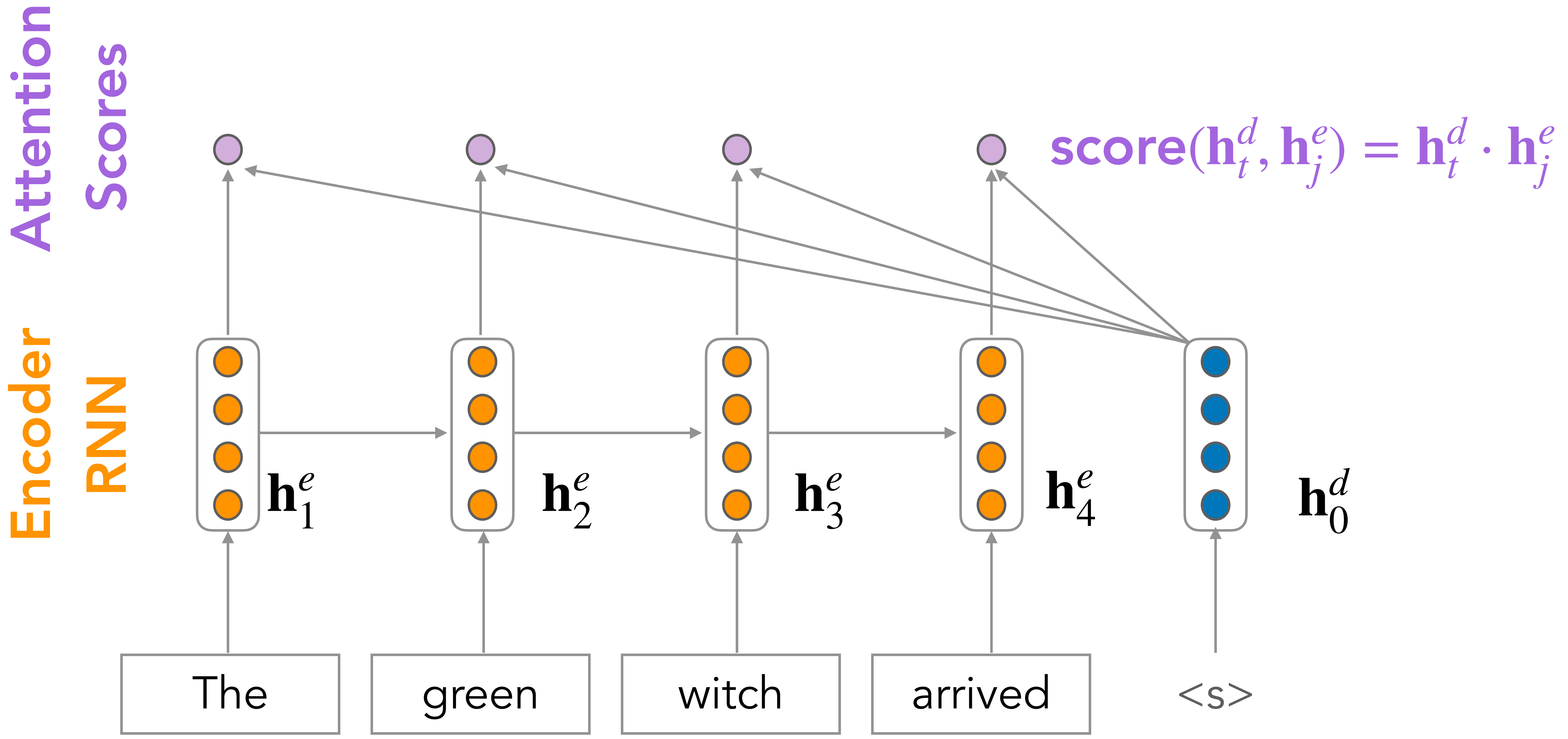


Query 1: Decoder, first time step

Source Sentence x

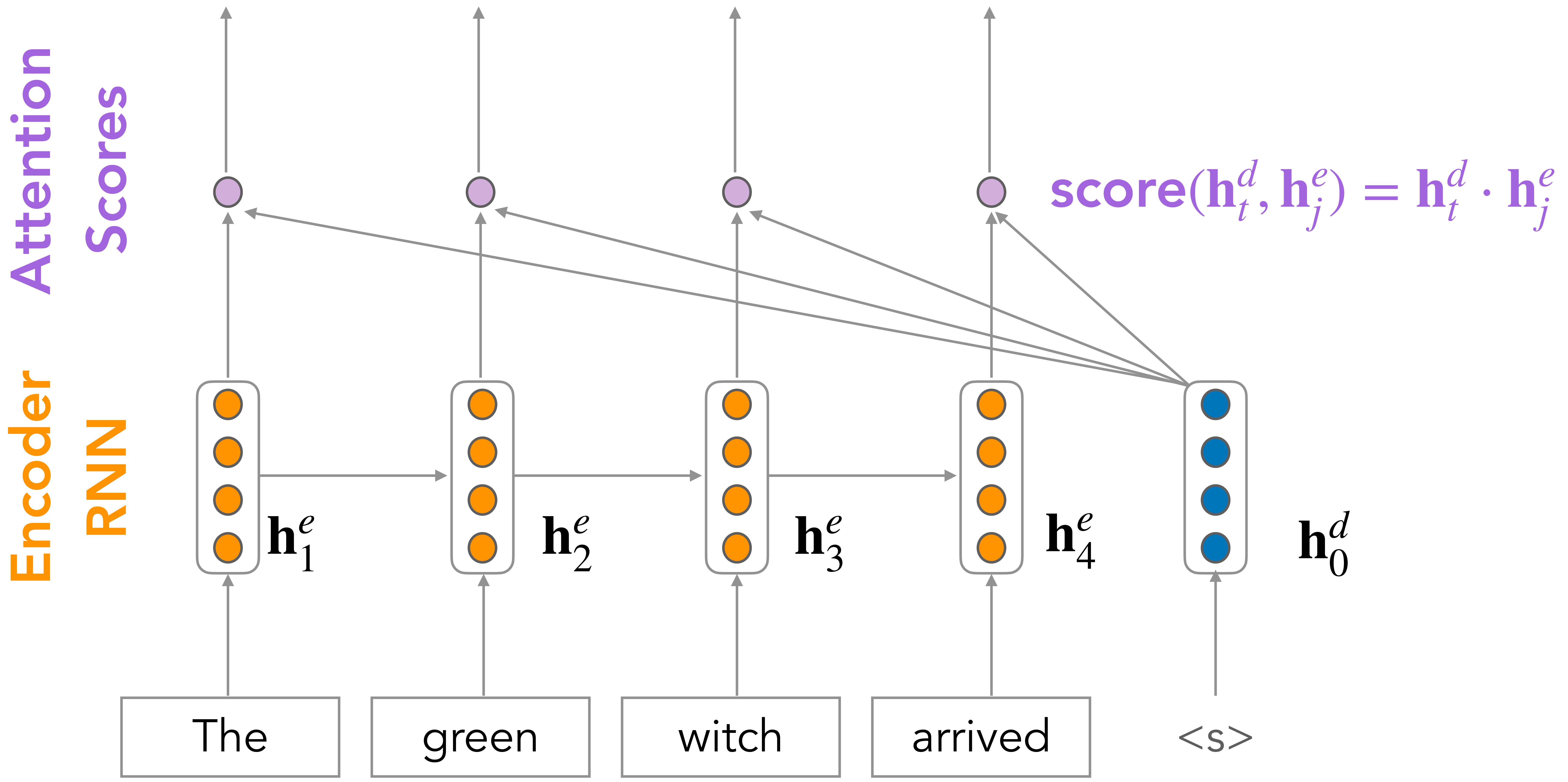
Note: Notation different from J&M

Dot product attention



Source Sentence $\mathbf{x}$

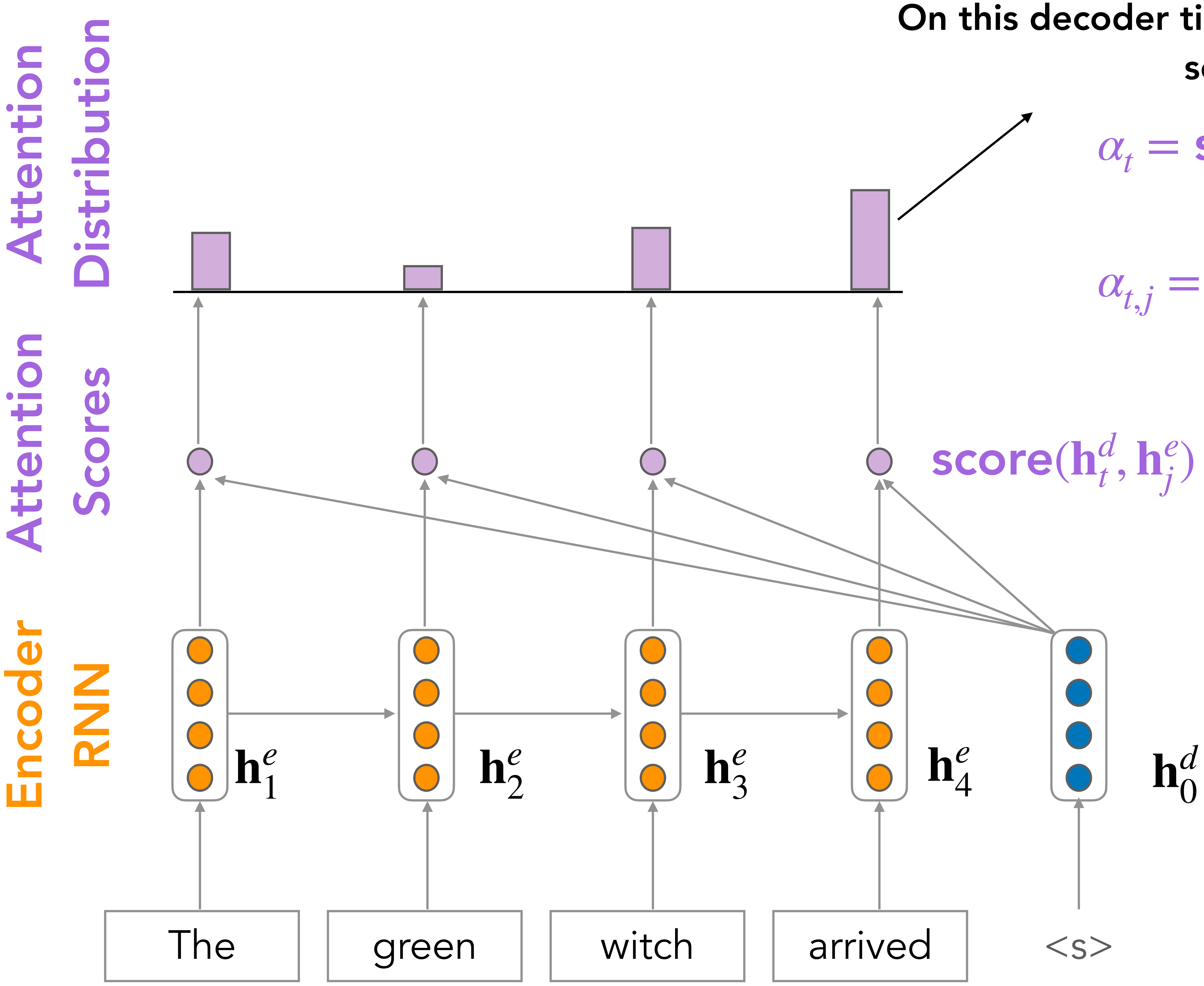
Note: Notation different from J&M



Source Sentence x

Take softmax to turn into probability distribution

Note: Notation different from J&M



Source Sentence $\mathbf{x}$

On this decoder tilmestep we are mostly focusing on the source token "arrived"

$$\alpha_t = \textbf{softmax}(\textbf{score}(\mathbf{h}_t^d, \mathbf{h}_j^e)) \in \Delta^N$$

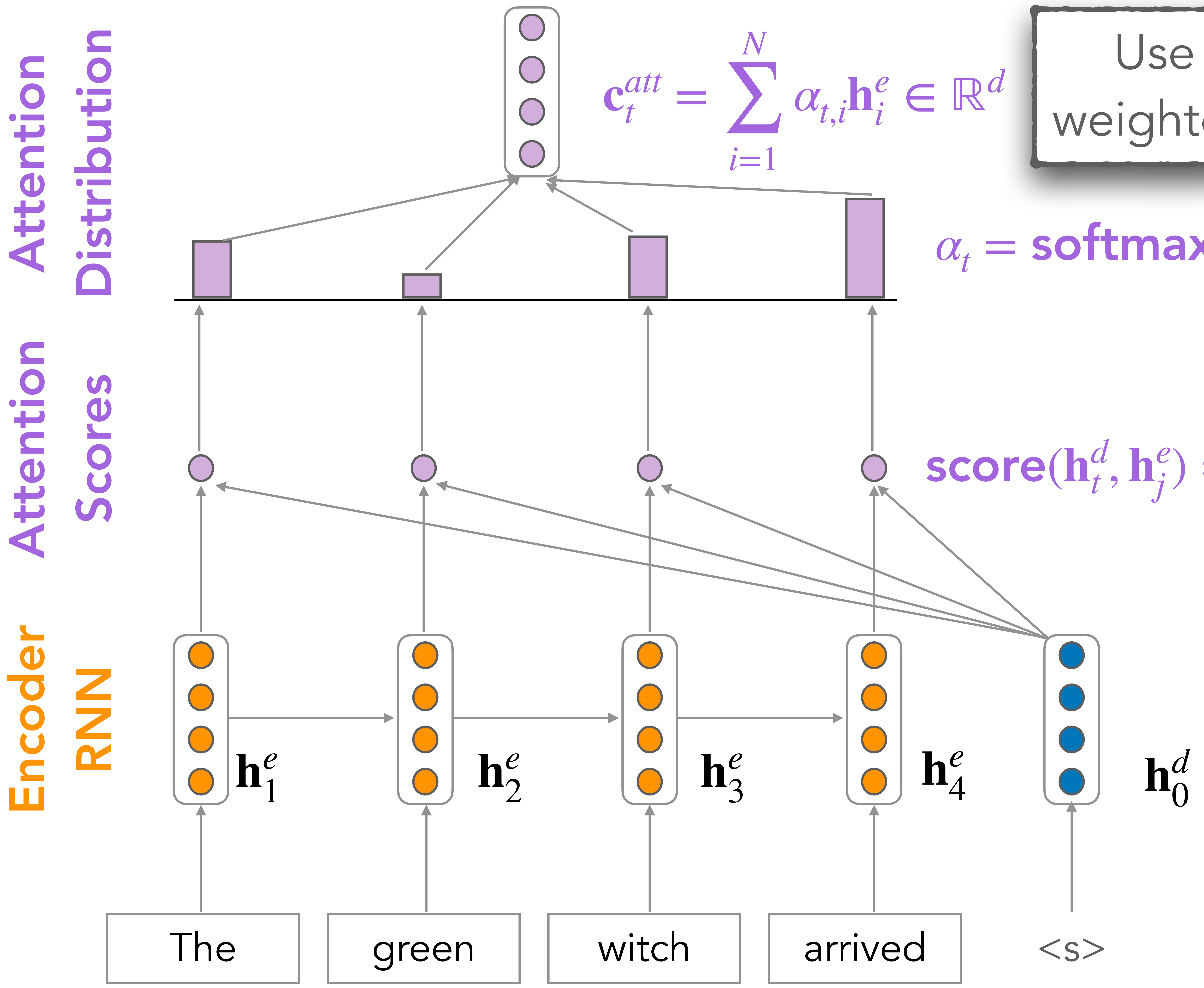
$$\alpha_{t,j} = \frac{\exp \mathbf{h}_t^d \cdot \mathbf{h}_j^e}{\sum_{n=1}^N \exp \mathbf{h}_t^d \cdot \mathbf{h}_n^e}$$

$$\textbf{score}(\mathbf{h}_t^d, \mathbf{h}_j^e) = \mathbf{h}_t^d \cdot \mathbf{h}_j^e$$

Take softmax to turn into probability distribution

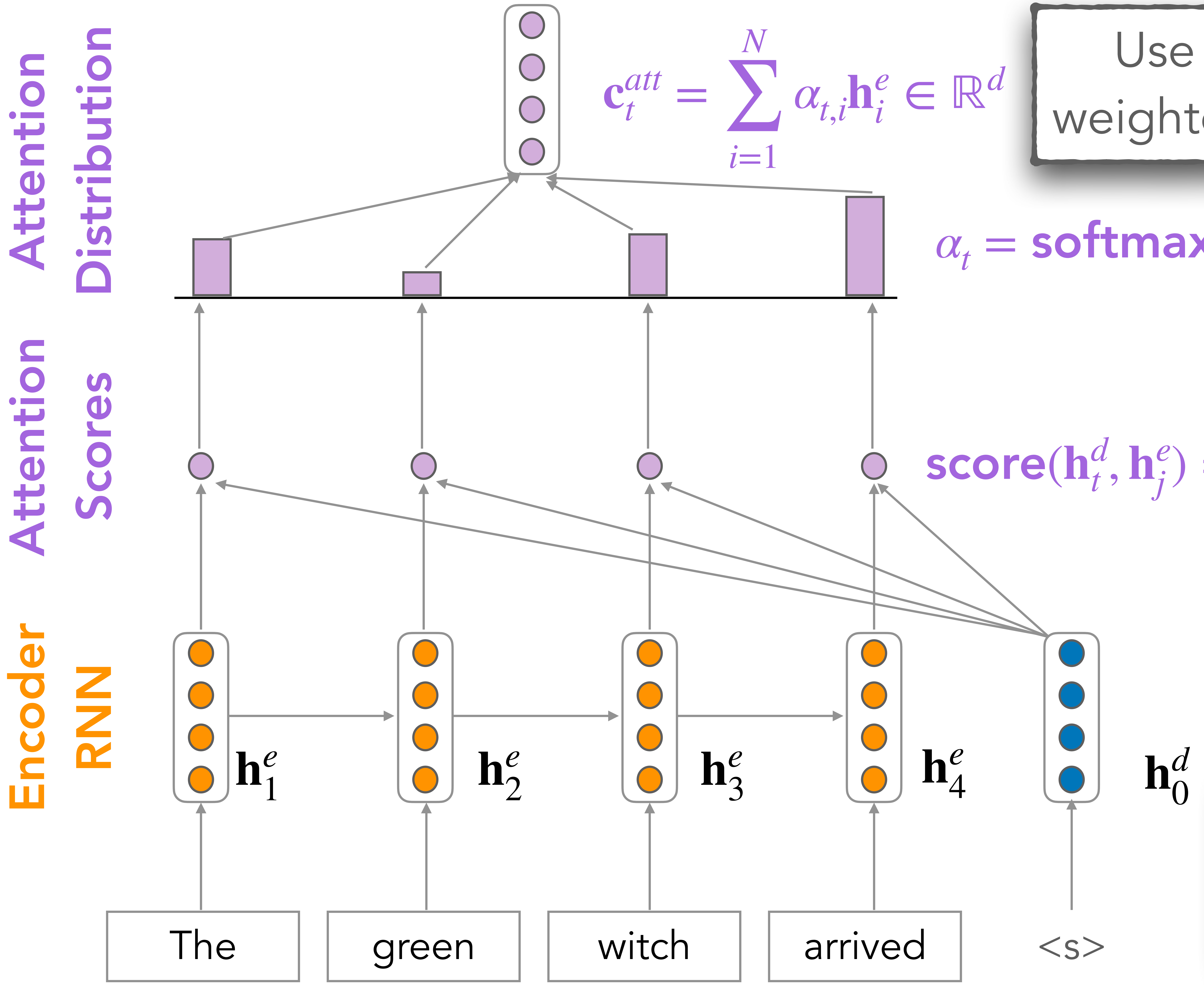
Note: Notation different from J&M

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



Source Sentence $\mathbf{x}$

Note: Notation different from J&M



$$\mathbf{c}_t^{att} = \sum_{i=1}^{N} \alpha_{t,i}\mathbf{h}_i^e \in \mathbb{R}^d$$

$$\alpha_t = \textbf{softmax}(\textbf{score}(\mathbf{h}_t^d, \mathbf{h}_j^e)) \in \Delta^N$$

$$\textbf{score}(\mathbf{h}_t^d, \mathbf{h}_j^e) = \mathbf{h}_t^d \cdot \mathbf{h}_j^e$$

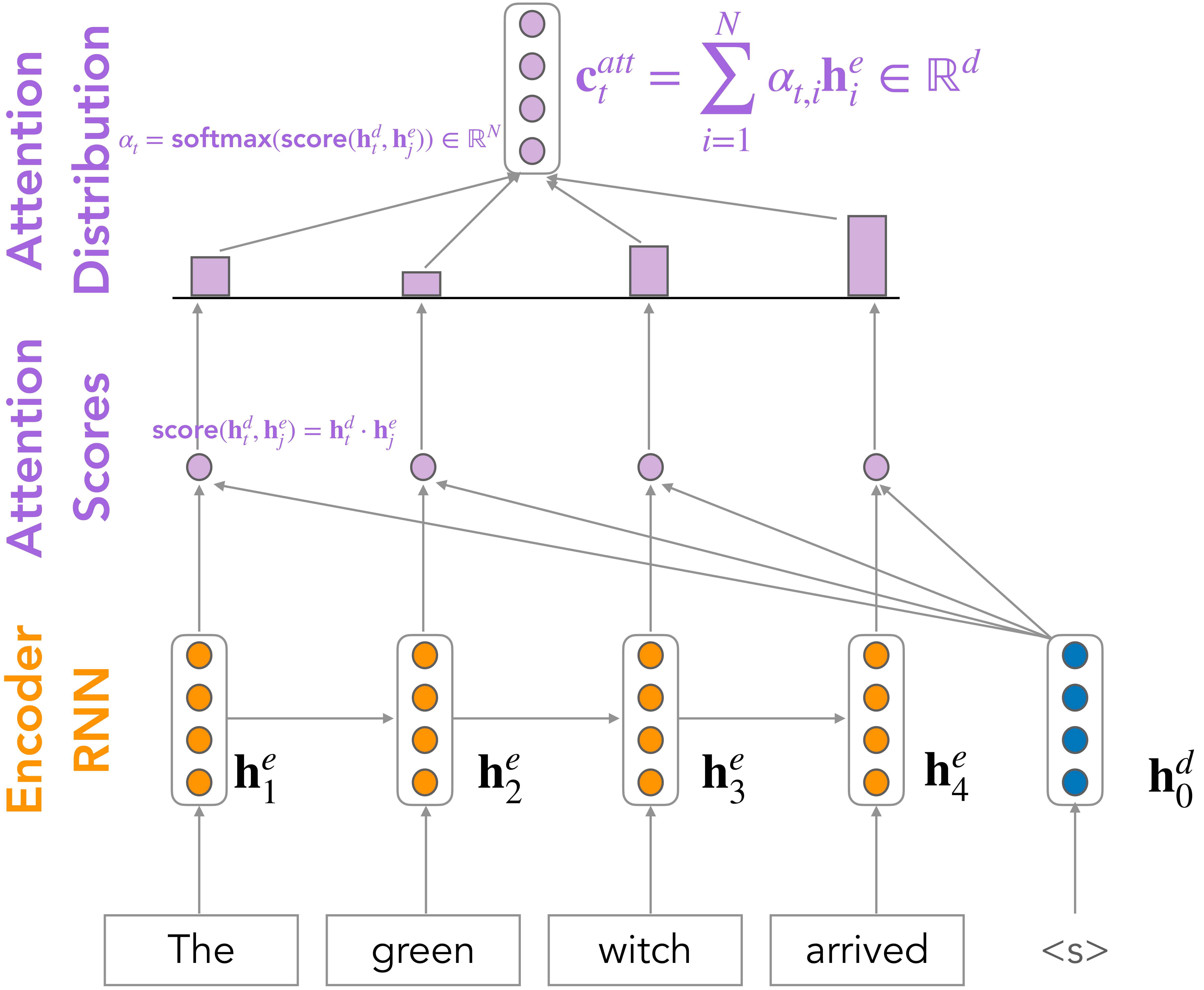
Use the attention distribution to take a weighted sum of the encoder hidden states.

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

Source Sentence $\mathbf{x}$

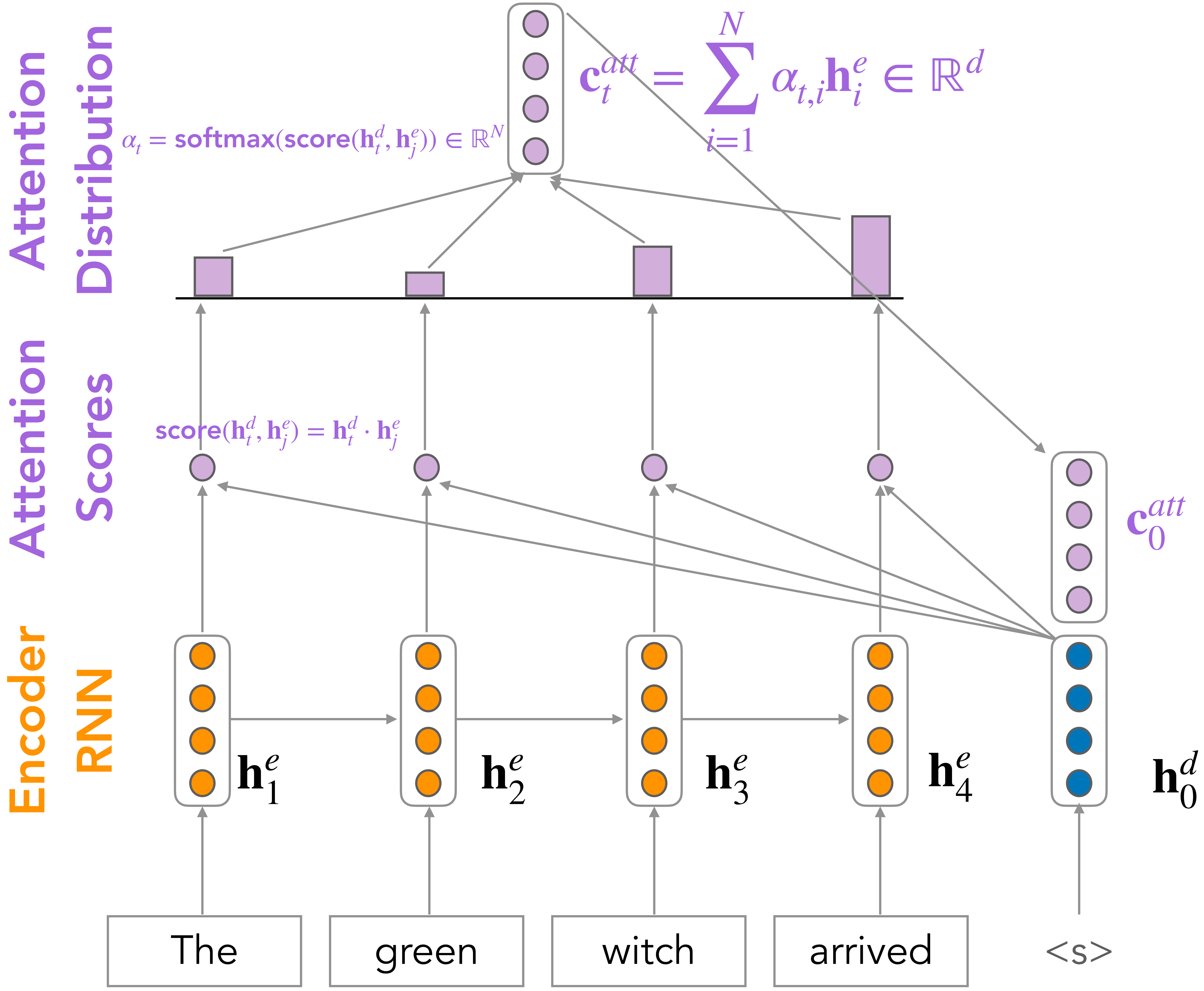
Note: Notation different from J&M

USC Viterbi



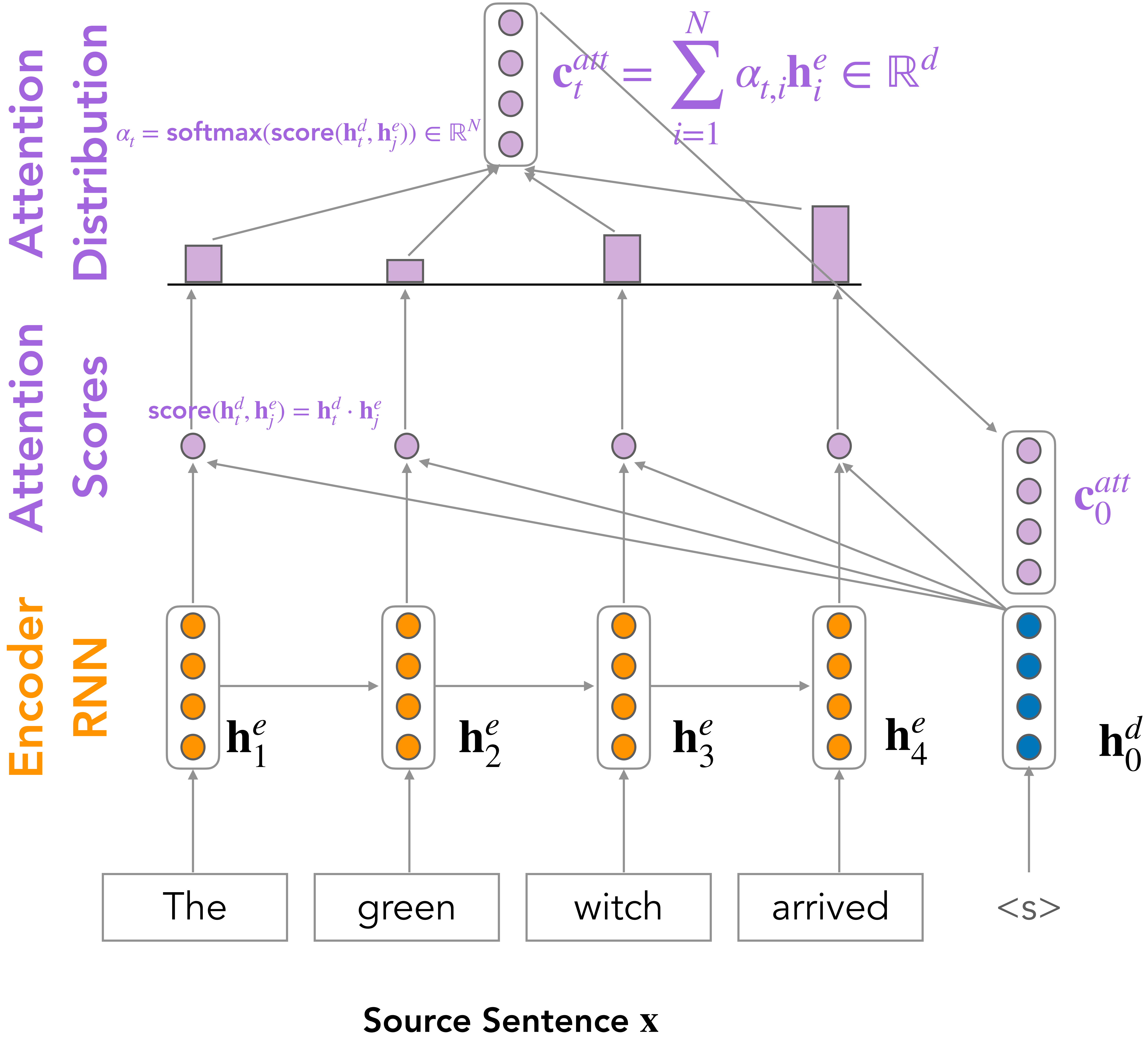
Source Sentence $\mathbf{x}$

Note: Notation different from J&M



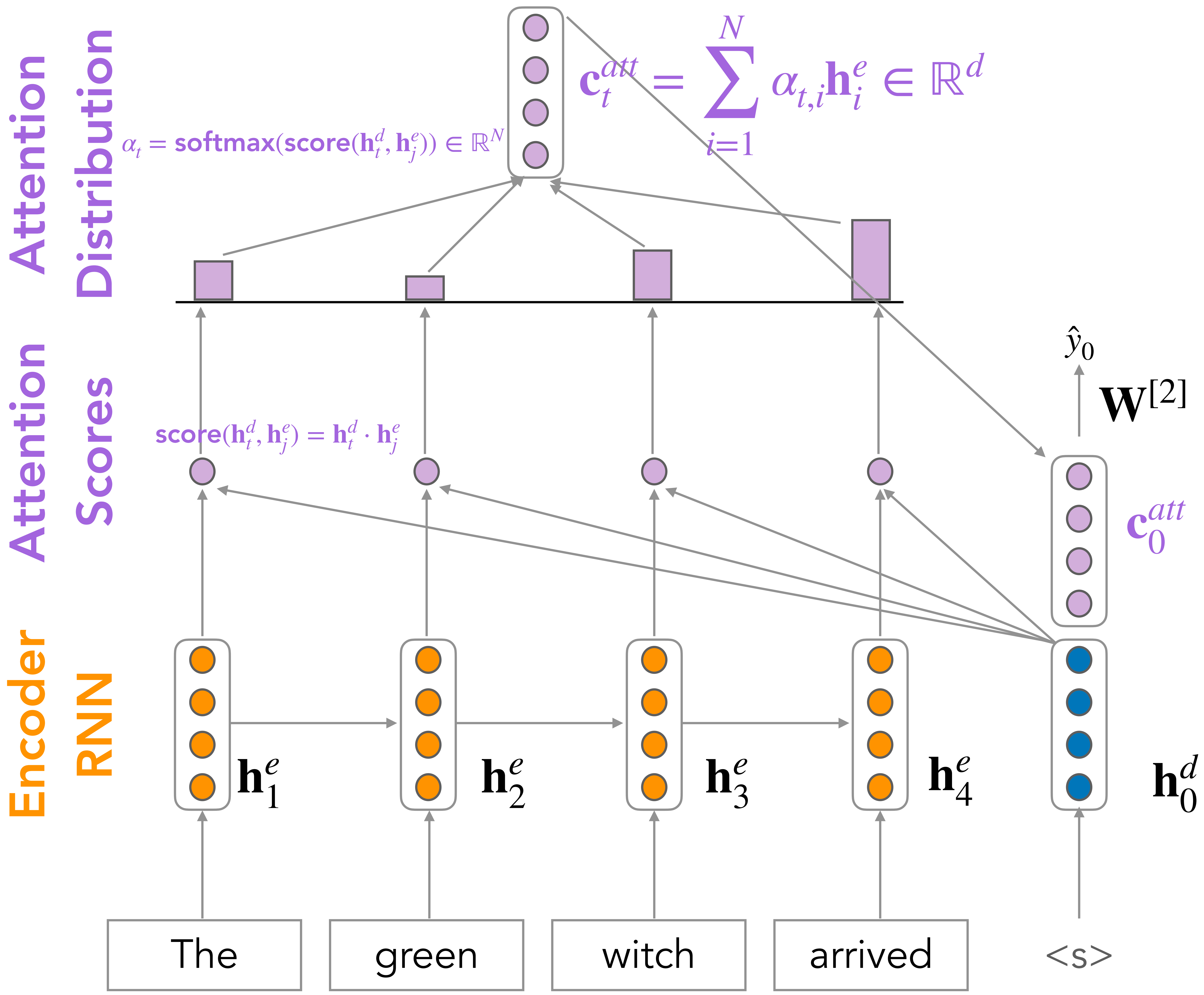
Source Sentence $\mathbf{x}$

Note: Notation different from J&M



Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_0$ as before

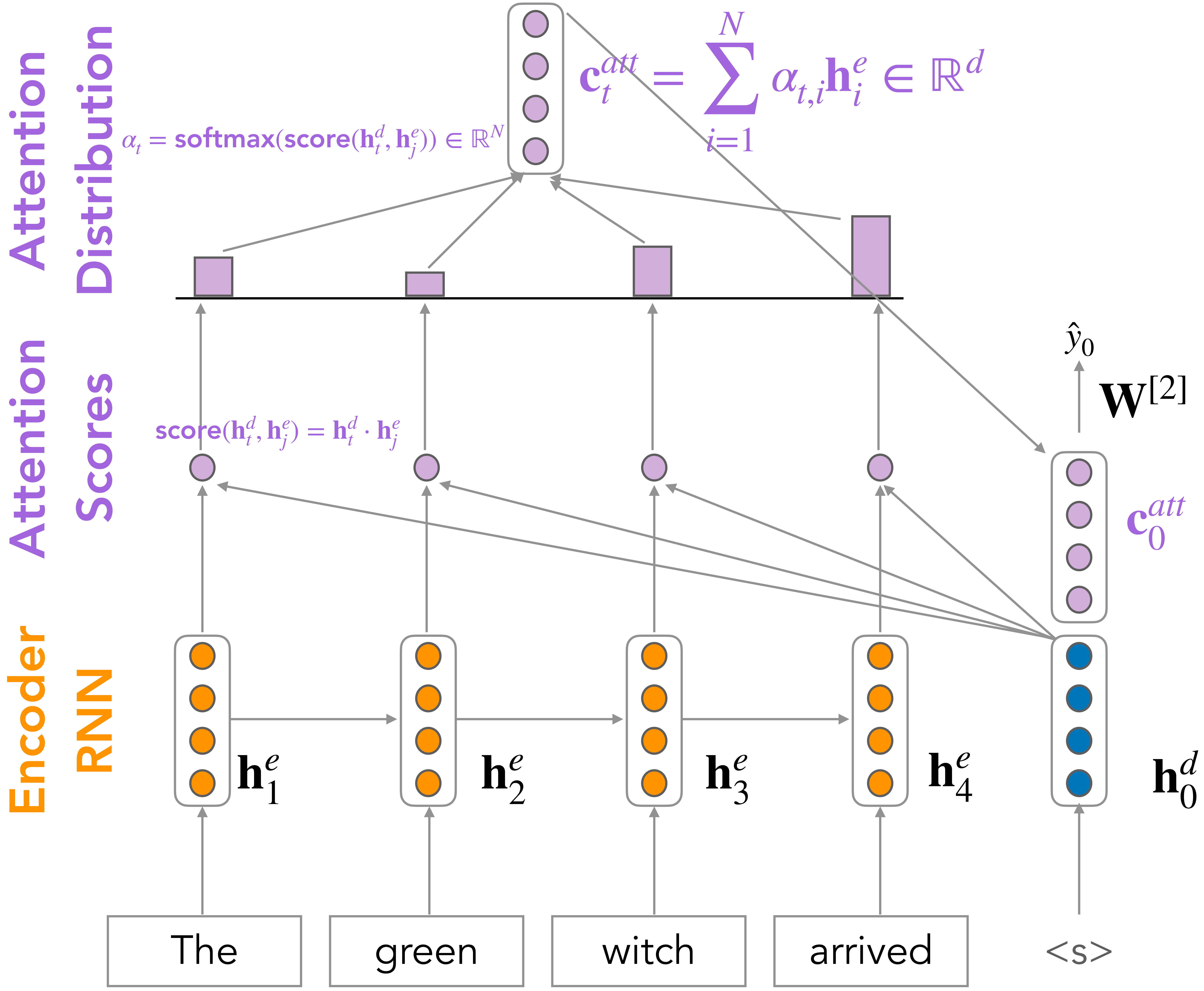
Note: Notation different from J&M



Source Sentence $\mathbf{x}$

Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_0$ as before

Note: Notation different from J&M



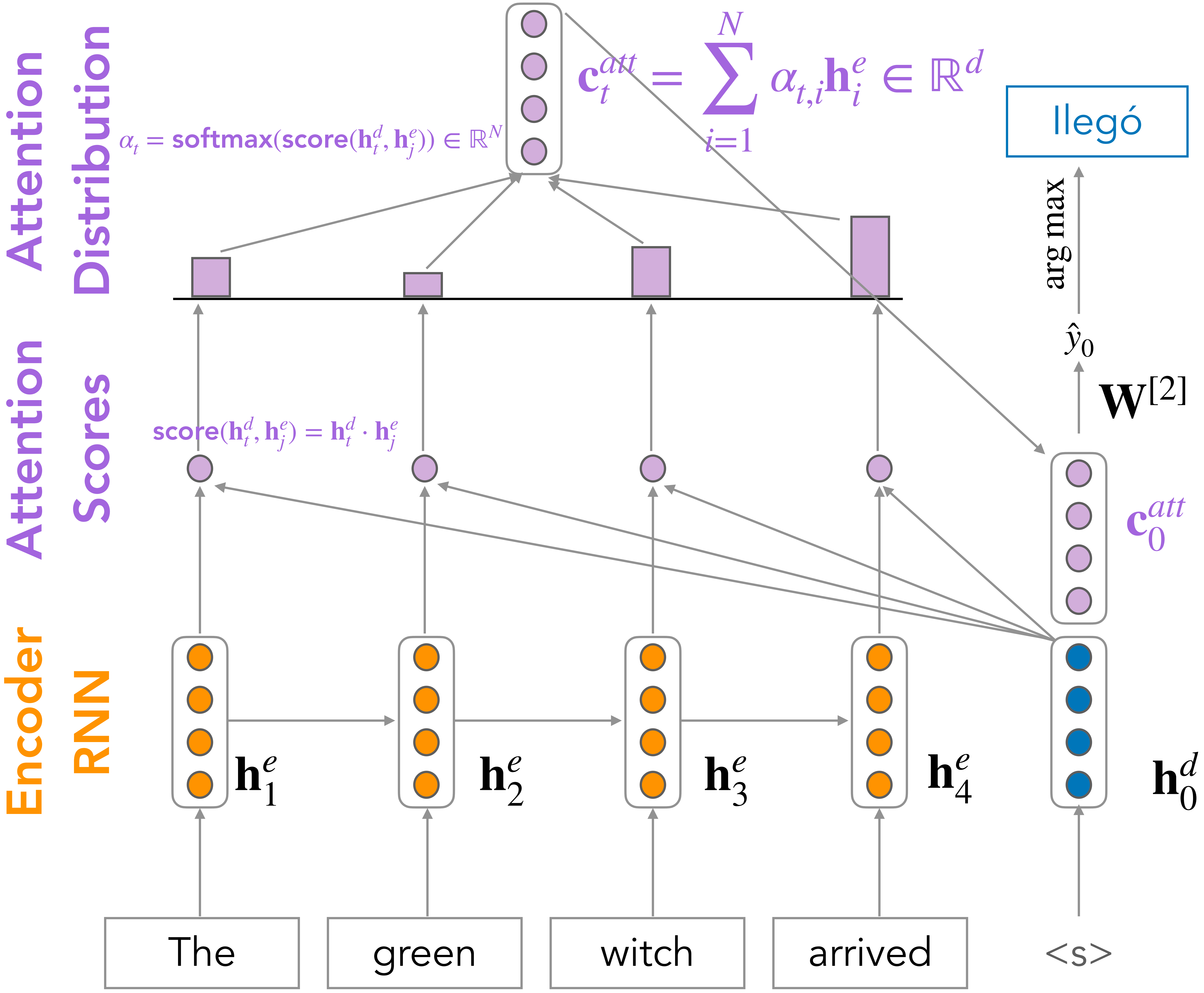
$$\hat{y}_t = \text{softmax}(\mathbf{W}^{[2]}[\mathbf{h}_t^d; \mathbf{c}_t^{att}])$$

$$\mathbf{W}^{[2]} \in \mathbb{R}^{V \times 2d}$$

Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_0$ as before

Source Sentence x

Note: Notation different from J&M



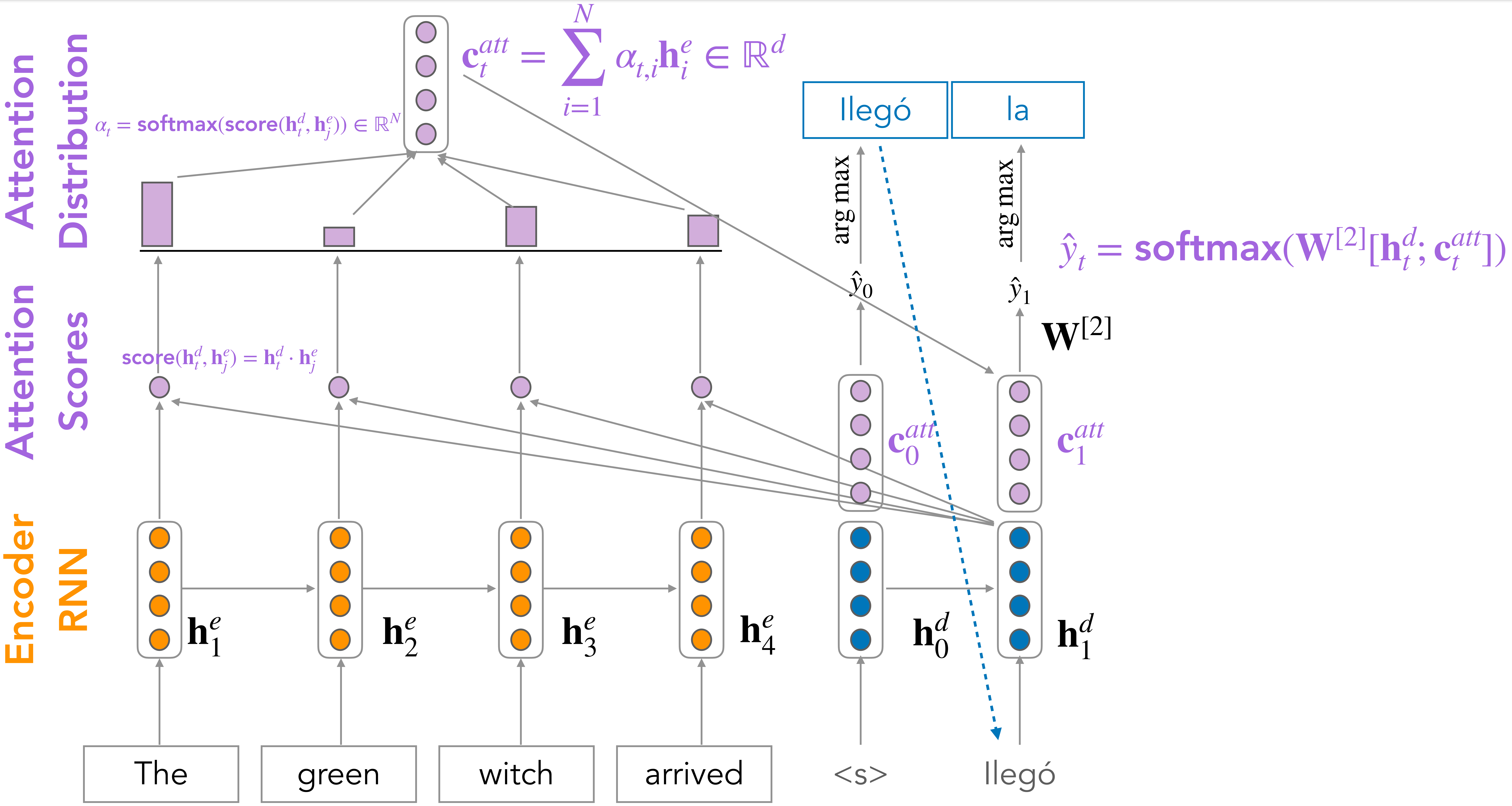
$$\hat{y}_t = \mathbf{softmax}(\mathbf{W}^{[2]}[\mathbf{h}_t^d; \mathbf{c}_t^{att}])$$

$$\mathbf{W}^{[2]} \in \mathbb{R}^{V \times 2d}$$

Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_0$ as before

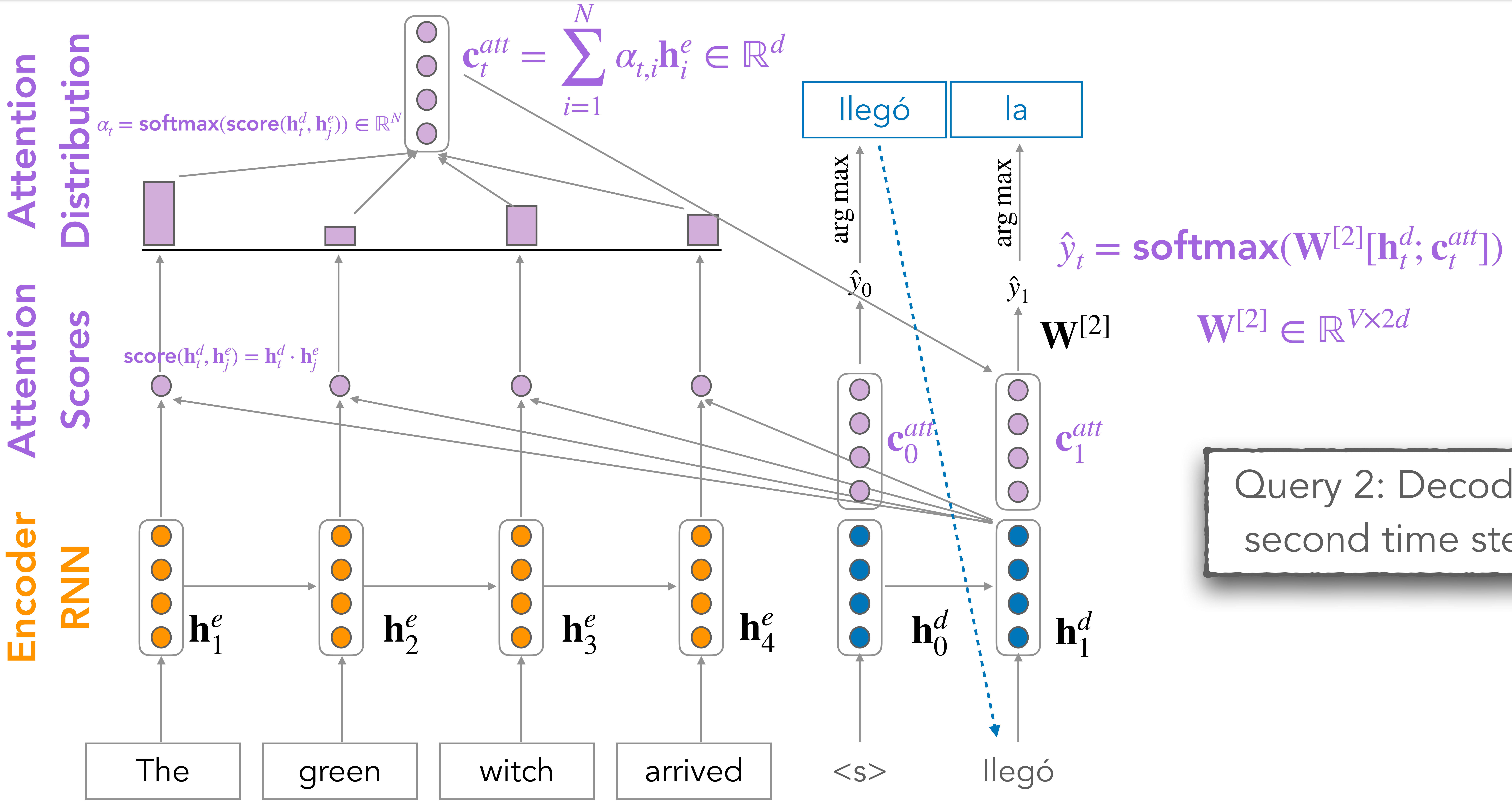
Source Sentence $\mathbf{x}$

Note: Notation different from J&M



Source Sentence $\mathbf{x}$

Note: Notation different from J&M



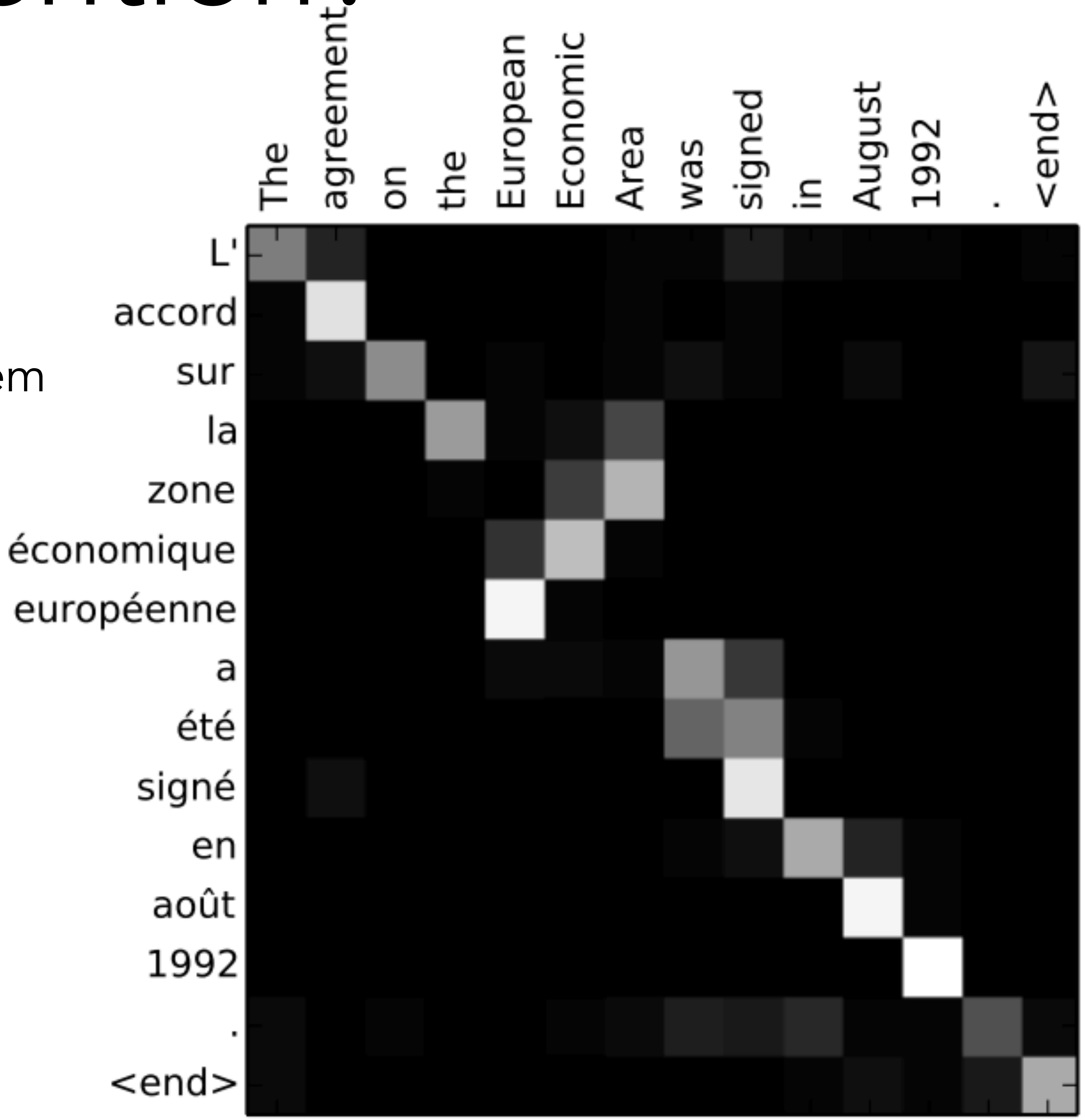
Query 2: Decoder, second time step

Source Sentence x

Note: Notation different from J&M

Why Attention?

- Attention significantly **improves** neural machine translation **performance**
  - Very useful to allow decoder to focus on certain parts of the source
- Attention **solves the information bottleneck** problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention **helps with vanishing gradient problem**
  - Provides shortcut to faraway states
- Attention provides some **interpretability**
  - By inspecting attention distribution, we can see what the decoder was focusing on →
  - We get alignment for free! We never explicitly trained an alignment system! The network just learned alignment by itself



Lecture Outline

- Announcements
- Recap: Seq2Seq and Attention
- More on Attention
- Transformers: Self-Attention Networks
    - Multiheaded Attention
    - Positional Embeddings
    - Transformer Blocks
- Transformers as Encoders, Decoders and Encoder-Decoders

More on Attention

Attention Variants

Attention Variants

- In general, we have some keys $\mathbf{h}_1, \ldots, \mathbf{h}_N \in \mathbb{R}^{d_1}$ and a query $\mathbf{q} \in \mathbb{R}^{d_2}$

Attention Variants

- In general, we have some keys $\mathbf{h}_1, \ldots, \mathbf{h}_N \in \mathbb{R}^{d_1}$ and a query $\mathbf{q} \in \mathbb{R}^{d_2}$
- Attention always involves

Attention Variants

- In general, we have some keys $\mathbf{h}_1, \ldots, \mathbf{h}_N \in \mathbb{R}^{d_1}$ and a query $\mathbf{q} \in \mathbb{R}^{d_2}$
- Attention always involves
    1. Computing the attention scores, $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$

Attention Variants

- In general, we have some keys $\mathbf{h}_1, \ldots, \mathbf{h}_N \in \mathbb{R}^{d_1}$ and a query $\mathbf{q} \in \mathbb{R}^{d_2}$
- Attention always involves
  1. Computing the attention scores, $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$

Can be done in multiple ways!

Attention Variants

- In general, we have some keys $\mathbf{h}_1, \ldots, \mathbf{h}_N \in \mathbb{R}^{d_1}$ and a query $\mathbf{q} \in \mathbb{R}^{d_2}$
- Attention always involves
  1. Computing the attention scores, $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$ → Can be done in multiple ways!
  2. Taking softmax to get attention distribution $\alpha_t = \text{softmax}(e(\mathbf{q}, \mathbf{h}_{1:N})) \in [0,1]^N$

Attention Variants

- In general, we have some keys $\mathbf{h}_1, \ldots, \mathbf{h}_N \in \mathbb{R}^{d_1}$ and a query $\mathbf{q} \in \mathbb{R}^{d_2}$
- Attention always involves
  1. Computing the attention scores, $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$ → Can be done in multiple ways!
  2. Taking softmax to get attention distribution $\alpha_t = \text{softmax}(e(\mathbf{q}, \mathbf{h}_{1:N})) \in [0,1]^N$
  3. Using attention distribution to take weighted sum of values:

$$\mathbf{c}_t^{att} = \sum_{i=1}^{N} \alpha_{t,i} \mathbf{h}_i \in \mathbb{R}^{d_1}$$

USC Viterbi

Attention Variants

- In general, we have some keys $\mathbf{h}_1, \ldots, \mathbf{h}_N \in \mathbb{R}^{d_1}$ and a query $\mathbf{q} \in \mathbb{R}^{d_2}$
- Attention always involves
    1. Computing the attention scores, $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$
    2. Taking softmax to get attention distribution $\alpha_t = \text{softmax}(e(\mathbf{q}, \mathbf{h}_{1:N})) \in [0,1]^N$
    3. Using attention distribution to take weighted sum of values:

Can be done in multiple ways!

$$\mathbf{c}_t^{att} = \sum_{i=1}^{N} \alpha_{t,i} \mathbf{h}_i \in \mathbb{R}^{d_1}$$

This leads to the attention output $\mathbf{c}_t^{att}$ (sometimes called the attention context vector)

Attention Variants

Attention Variants

- There are several ways you can compute $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$ from $\mathbf{h}_1 \ldots \mathbf{h}_N \in \mathbb{R}^{d_1}$ and $\mathbf{q} \in \mathbb{R}^{d_2}$

Attention Variants

- There are several ways you can compute $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$ from $\mathbf{h}_1 \ldots \mathbf{h}_N \in \mathbb{R}^{d_1}$ and $\mathbf{q} \in \mathbb{R}^{d_2}$
- Basic dot-product attention: $e(\mathbf{q}, \mathbf{h}_{1:N}) = [\mathbf{q} \cdot \mathbf{h}_j]_{j=1:N}$

Attention Variants

- There are several ways you can compute $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$ from $\mathbf{h}_1 \ldots \mathbf{h}_N \in \mathbb{R}^{d_1}$ and $\mathbf{q} \in \mathbb{R}^{d_2}$
- Basic dot-product attention: $e(\mathbf{q}, \mathbf{h}_{1:N}) = [\mathbf{q} \cdot \mathbf{h}_j]_{j=1:N}$
  - This assumes $d_1 = d_2$

Attention Variants

- There are several ways you can compute $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$ from $\mathbf{h}_1 \ldots \mathbf{h}_N \in \mathbb{R}^{d_1}$ and $\mathbf{q} \in \mathbb{R}^{d_2}$
- Basic dot-product attention: $e(\mathbf{q}, \mathbf{h}_{1:N}) = [\mathbf{q} \cdot \mathbf{h}_j]_{j=1:N}$
  - This assumes $d_1 = d_2$
  - We applied this in encoder-decoder RNNs

Attention Variants

- There are several ways you can compute $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$ from $\mathbf{h}_1 \ldots \mathbf{h}_N \in \mathbb{R}^{d_1}$ and $\mathbf{q} \in \mathbb{R}^{d_2}$
- Basic dot-product attention: $e(\mathbf{q}, \mathbf{h}_{1:N}) = [\mathbf{q} \cdot \mathbf{h}_j]_{j=1:N}$
  - This assumes $d_1 = d_2$
  - We applied this in encoder-decoder RNNs
- Multiplicative (bilinear) attention: $e(\mathbf{q}, \mathbf{h}_{1:N}) = [\mathbf{q}^T \mathbf{W} \mathbf{h}_j]_{j=1:N}$

Attention Variants

- There are several ways you can compute $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$ from $\mathbf{h}_1 \ldots \mathbf{h}_N \in \mathbb{R}^{d_1}$ and $\mathbf{q} \in \mathbb{R}^{d_2}$
- Basic dot-product attention: $e(\mathbf{q}, \mathbf{h}_{1:N}) = [\mathbf{q} \cdot \mathbf{h}_j]_{j=1:N}$
  - This assumes $d_1 = d_2$
  - We applied this in encoder-decoder RNNs
- Multiplicative (bilinear) attention: $e(\mathbf{q}, \mathbf{h}_{1:N}) = [\mathbf{q}^T \mathbf{W} \mathbf{h}_j]_{j=1:N}$
  - Where $\mathbf{W} \in \mathbb{R}^{d_2 \times d_1}$ is a learned weight matrix.

Attention Variants

- There are several ways you can compute $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$ from $\mathbf{h}_1 \ldots \mathbf{h}_N \in \mathbb{R}^{d_1}$ and $\mathbf{q} \in \mathbb{R}^{d_2}$
- Basic dot-product attention: $e(\mathbf{q}, \mathbf{h}_{1:N}) = [\mathbf{q} \cdot \mathbf{h}_j]_{j=1:N}$
  - This assumes $d_1 = d_2$
  - We applied this in encoder-decoder RNNs
- Multiplicative (bilinear) attention: $e(\mathbf{q}, \mathbf{h}_{1:N}) = [\mathbf{q}^T \mathbf{W} \mathbf{h}_j]_{j=1:N}$
  - Where $\mathbf{W} \in \mathbb{R}^{d_2 \times d_1}$ is a learned weight matrix.
- Linear attention: No non-linearity, i.e. no step (2).

Attention Variants

- There are several ways you can compute $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$ from $\mathbf{h}_1 \ldots \mathbf{h}_N \in \mathbb{R}^{d_1}$ and $\mathbf{q} \in \mathbb{R}^{d_2}$
- Basic dot-product attention: $e(\mathbf{q}, \mathbf{h}_{1:N}) = [\mathbf{q} \cdot \mathbf{h}_j]_{j=1:N}$
  - This assumes $d_1 = d_2$
  - We applied this in encoder-decoder RNNs
- Multiplicative (bilinear) attention: $e(\mathbf{q}, \mathbf{h}_{1:N}) = [\mathbf{q}^T \mathbf{W} \mathbf{h}_j]_{j=1:N}$
  - Where $\mathbf{W} \in \mathbb{R}^{d_2 \times d_1}$ is a learned weight matrix.
- Linear attention: No non-linearity, i.e. no step (2).
  - Unsurprisingly, does not work too well…

More on Attention

Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query

More on Attention

Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query

- We sometimes say that the query attends to the values.

More on Attention

Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query

- We sometimes say that the query attends to the values.
  - For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values)

More on Attention

Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query

- We sometimes say that the query attends to the values.
  - For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values)
  - Keys and values correspond to the same entity (the encoded sequence).

More on Attention

Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query

- We sometimes say that the query attends to the values.
  - For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values)
  - Keys and values correspond to the same entity (the encoded sequence).
- The weighted sum is a **selective summary** of the information contained in the values, where the query determines which values to focus on.

More on Attention

Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query

- We sometimes say that the query attends to the values.
  - For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values)
  - Keys and values correspond to the same entity (the encoded sequence).
- The weighted sum is a **selective summary** of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a **fixed-size representation** of an **arbitrary set of representations** (the values), dependent on some other representation (the query).

More on Attention

Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query

- We sometimes say that the query attends to the values.
  - For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values)
  - Keys and values correspond to the same entity (the encoded sequence).
- The weighted sum is a **selective summary** of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a **fixed-size representation** of an **arbitrary set of representations** (the values), dependent on some other representation (the query).
- Attention is a powerful, flexible, general deep learning technique in all deep learning models.

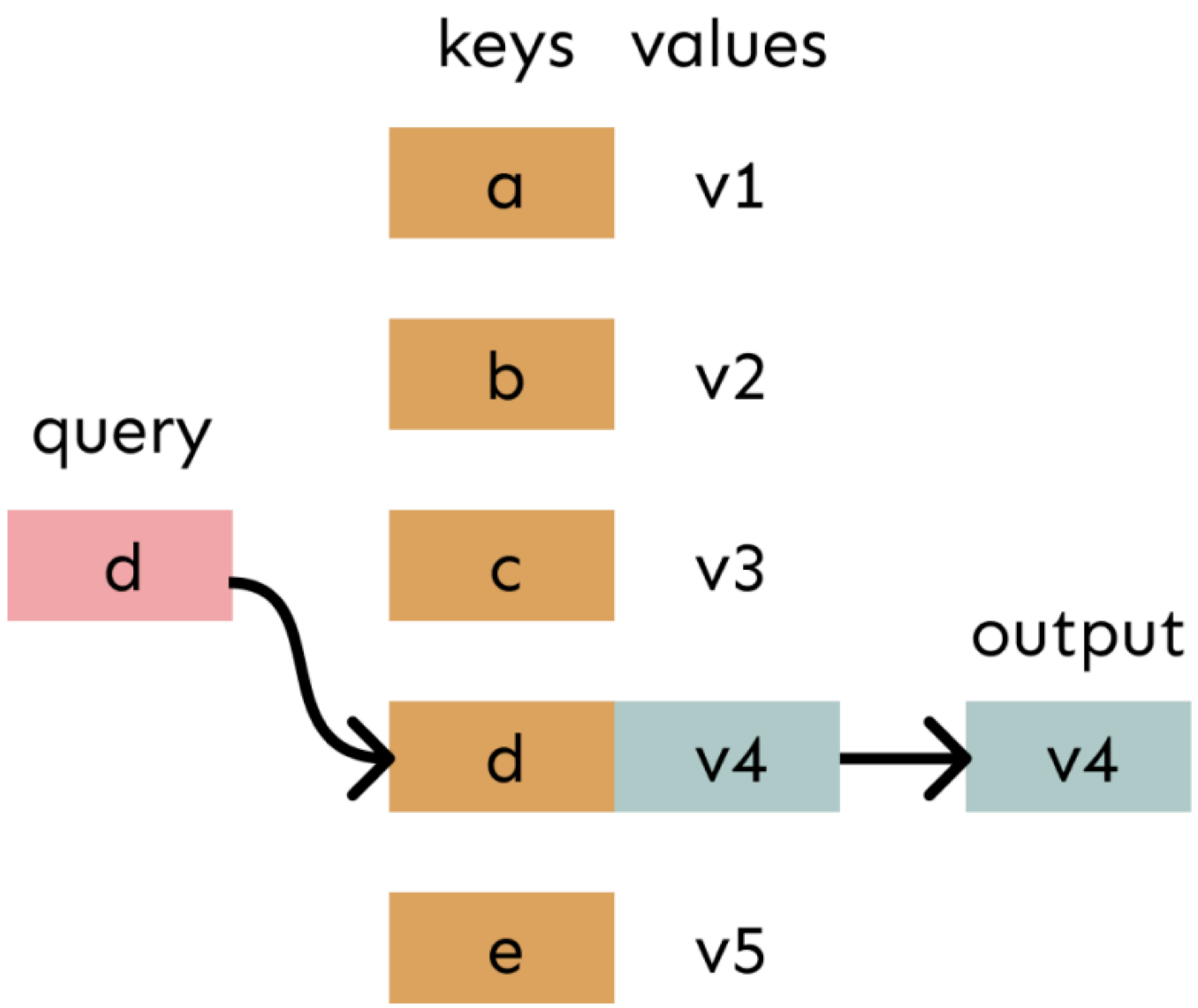
More on Attention

Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query

- We sometimes say that the query attends to the values.
  - For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values)
  - Keys and values correspond to the same entity (the encoded sequence).
- The weighted sum is a **selective summary** of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a **fixed-size representation** of an **arbitrary set of representations** (the values), dependent on some other representation (the query).
- Attention is a powerful, flexible, general deep learning technique in all deep learning models.
  - A new idea from after 2010! Originated in NMT

Attention and lookup tables

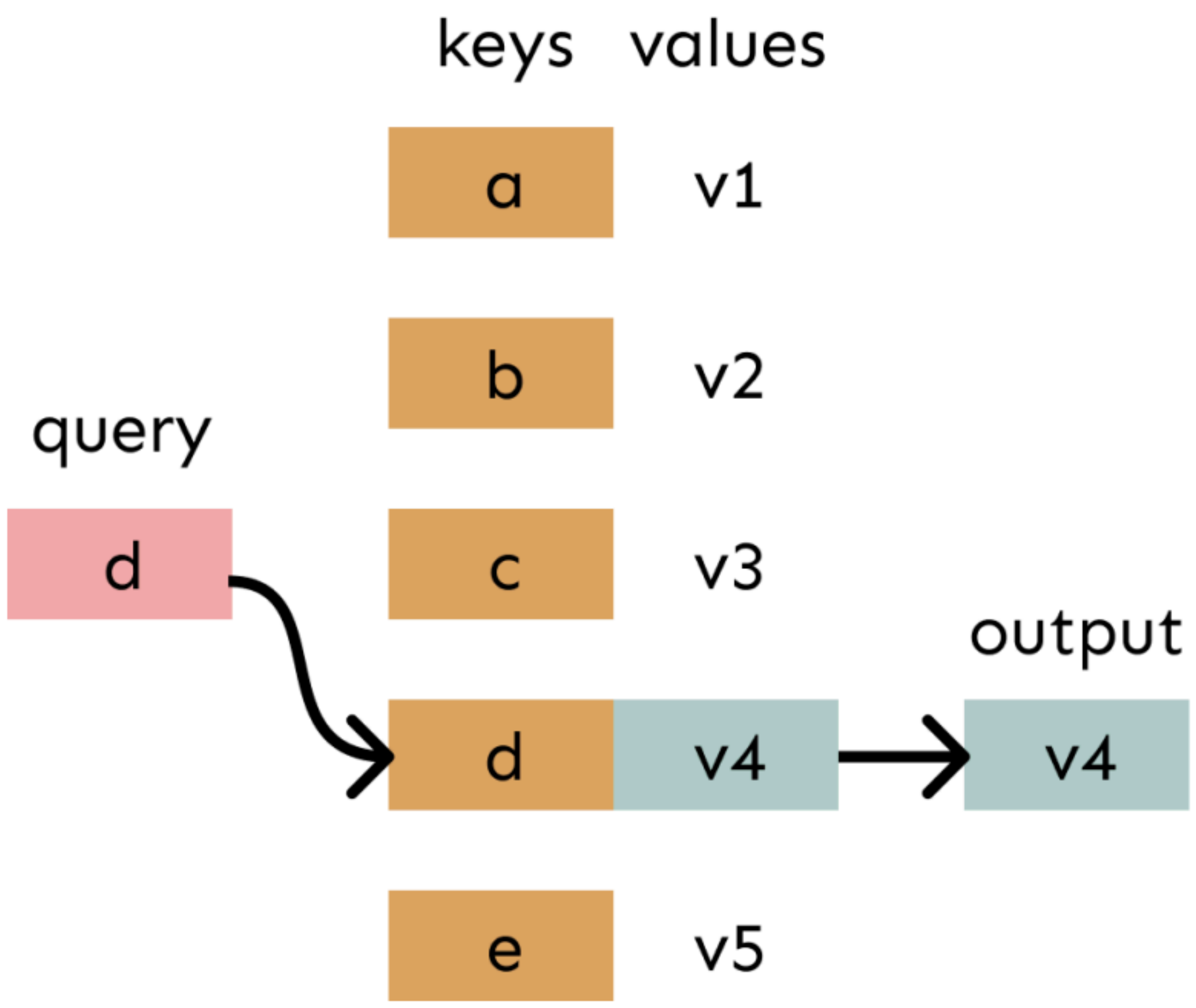
In a lookup table, we have a table of keys that map to values. The query matches one of the keys, returning its value.

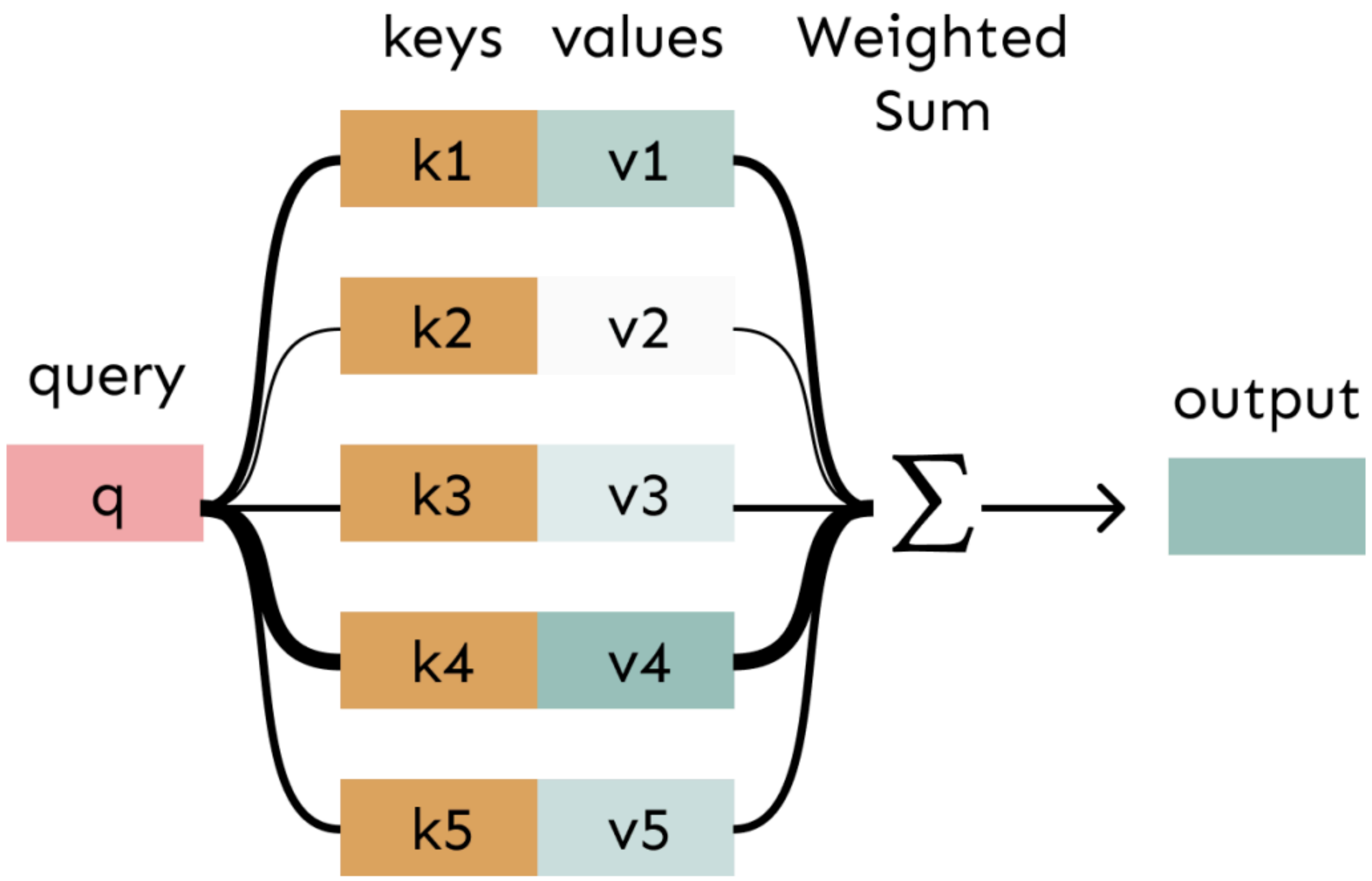


Attention and lookup tables

In a lookup table, we have a table of keys that map to values. The query matches one of the keys, returning its value.

In attention, the query matches all keys softly, to a weight between 0 and 1. The keys' values are multiplied by the weights and summed.



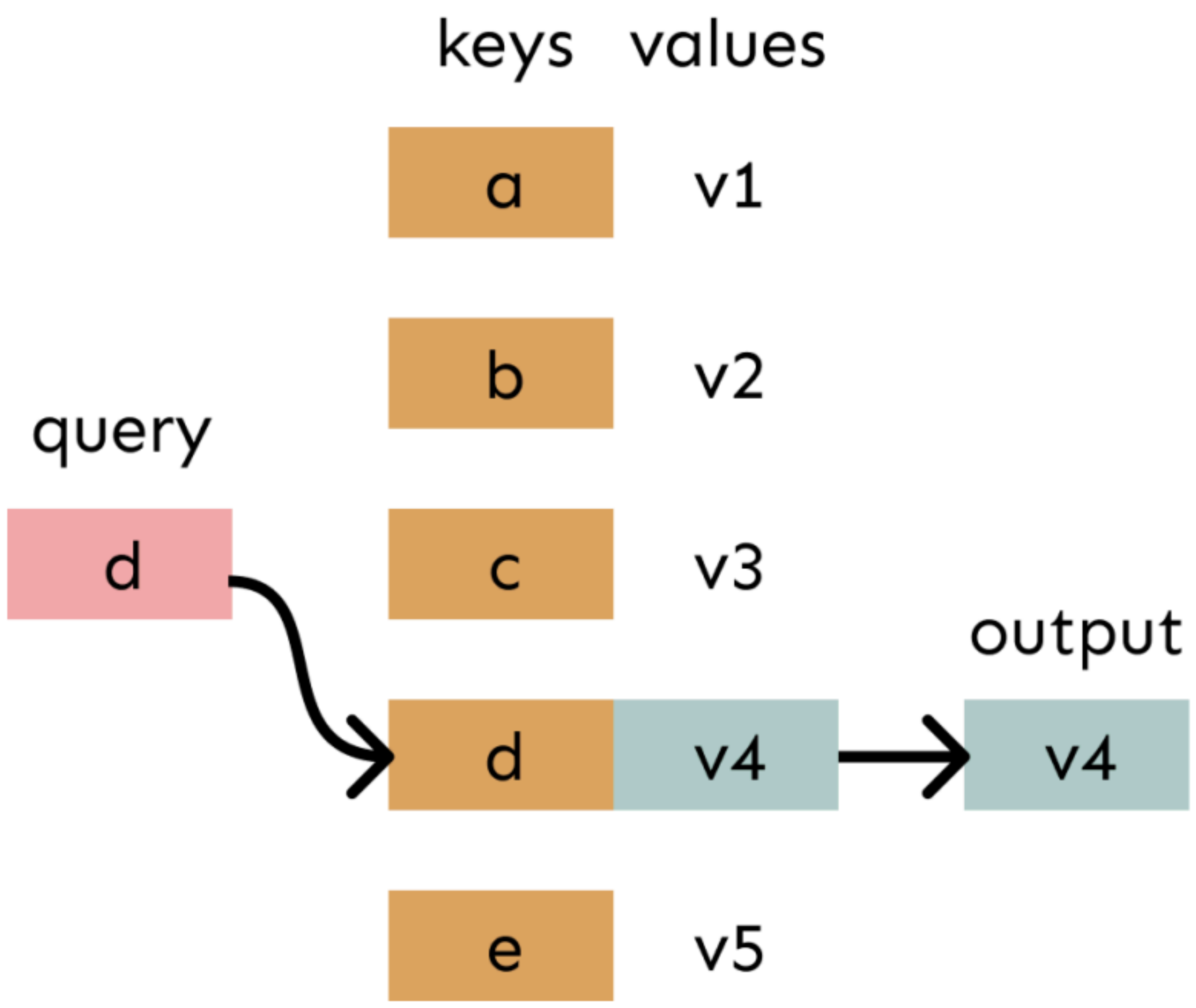


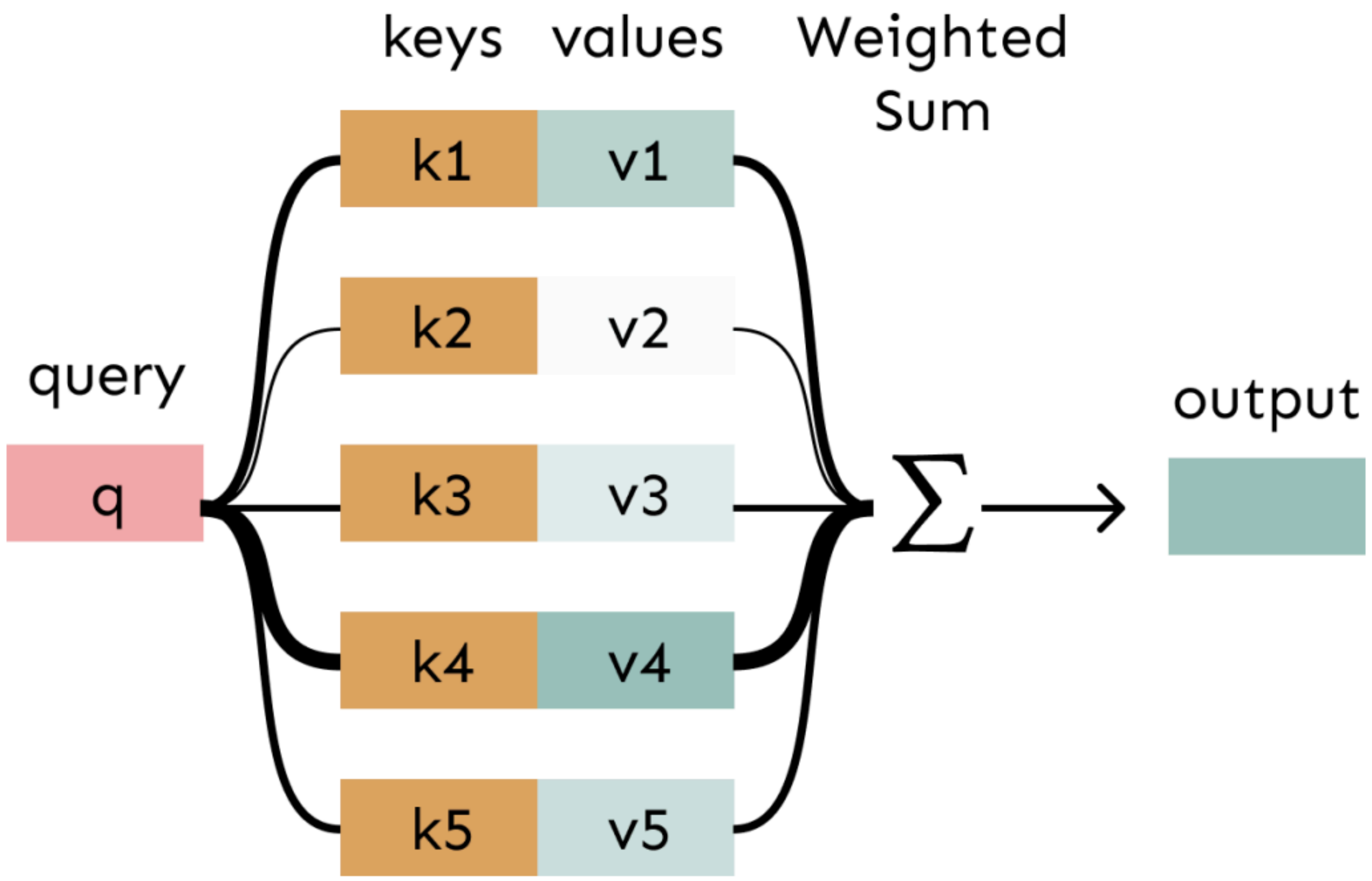
Attention and lookup tables

Attention performs fuzzy lookup in a key-value store

In a lookup table, we have a table of keys that map to values. The query matches one of the keys, returning its value.

In attention, the query matches all keys softly, to a weight between 0 and 1. The keys' values are multiplied by the weights and summed.

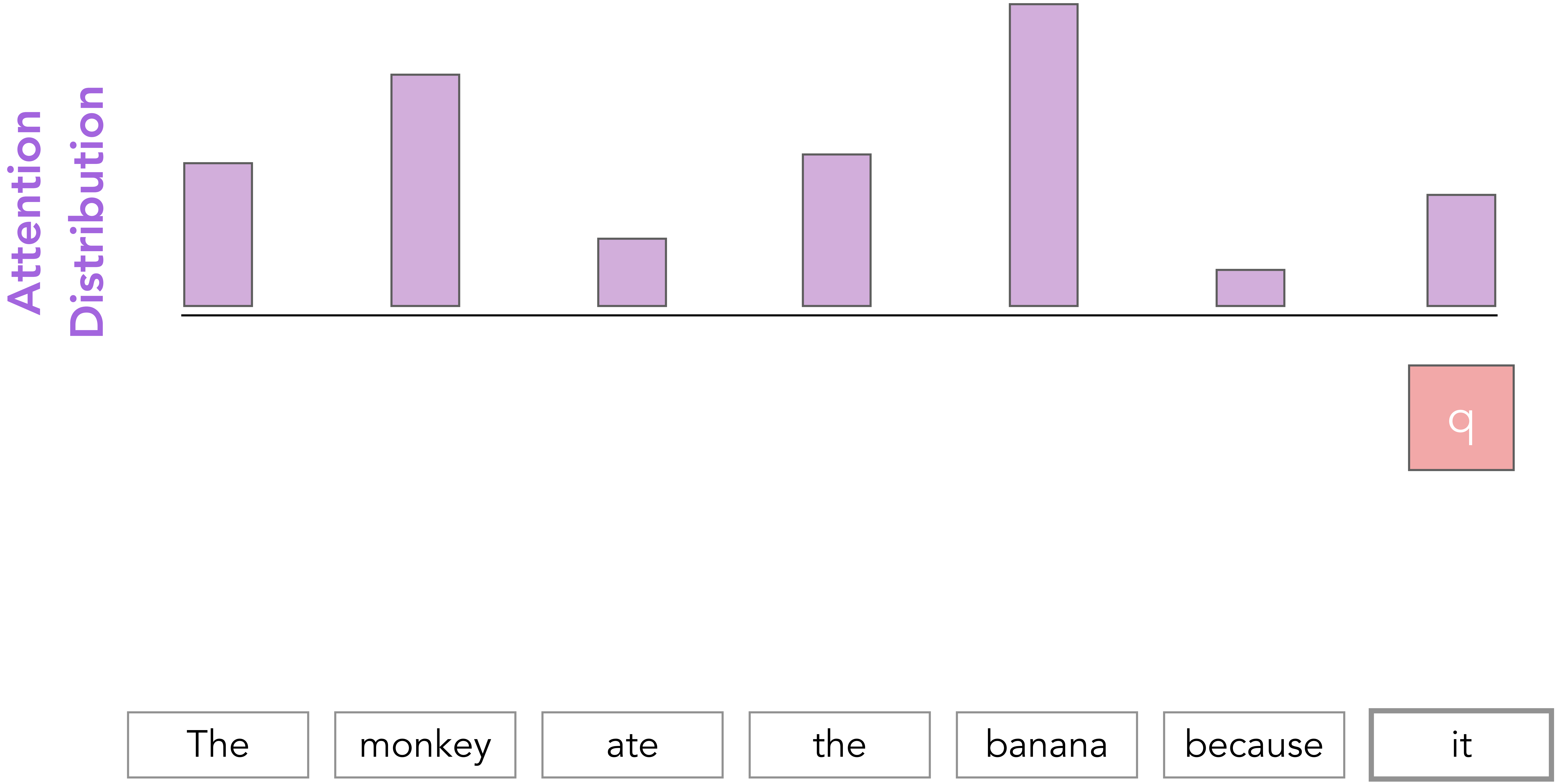


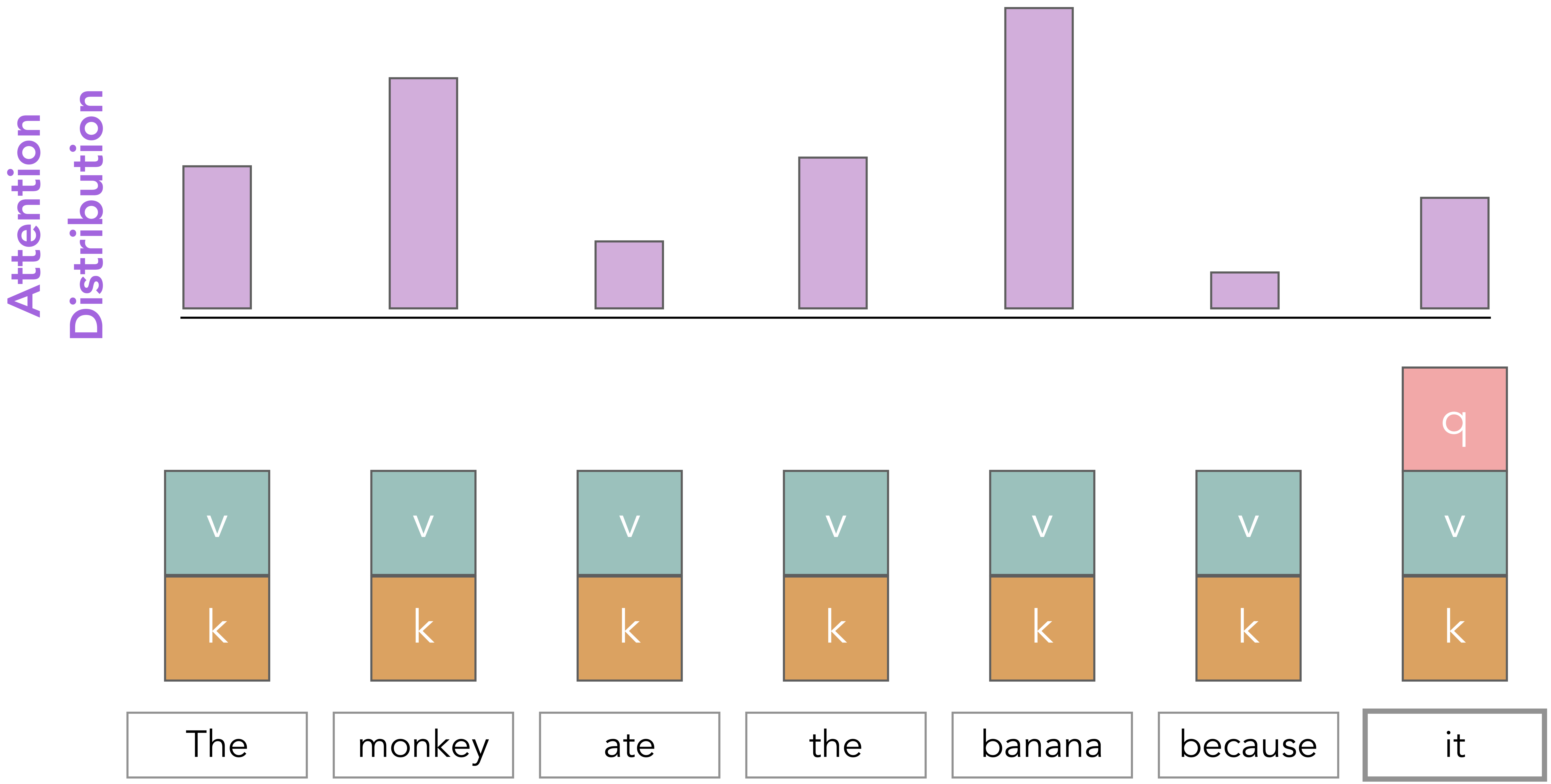


The	monkey	ate	the	banana	because	it

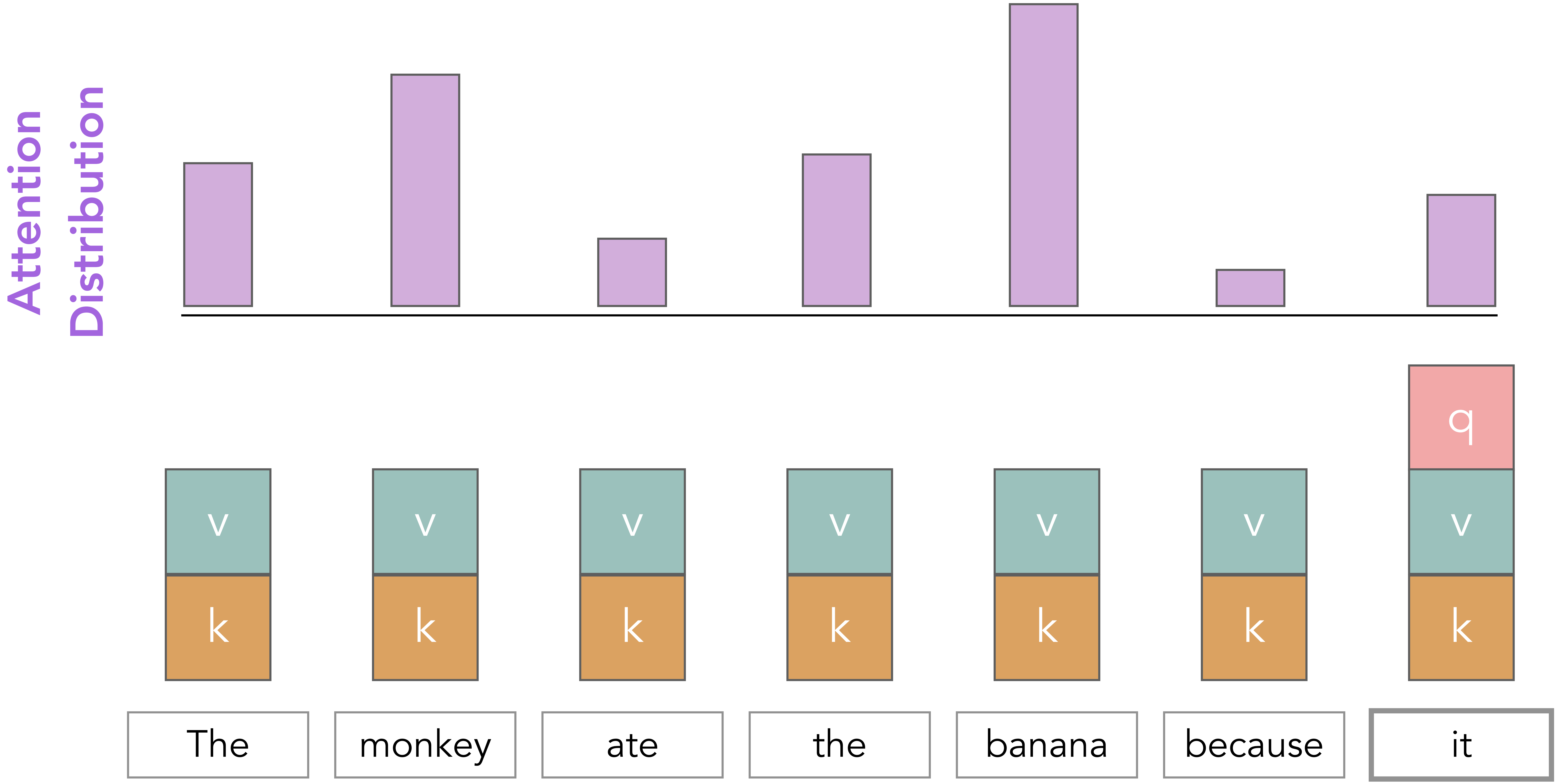
q

The	monkey	ate	the	banana	because	it

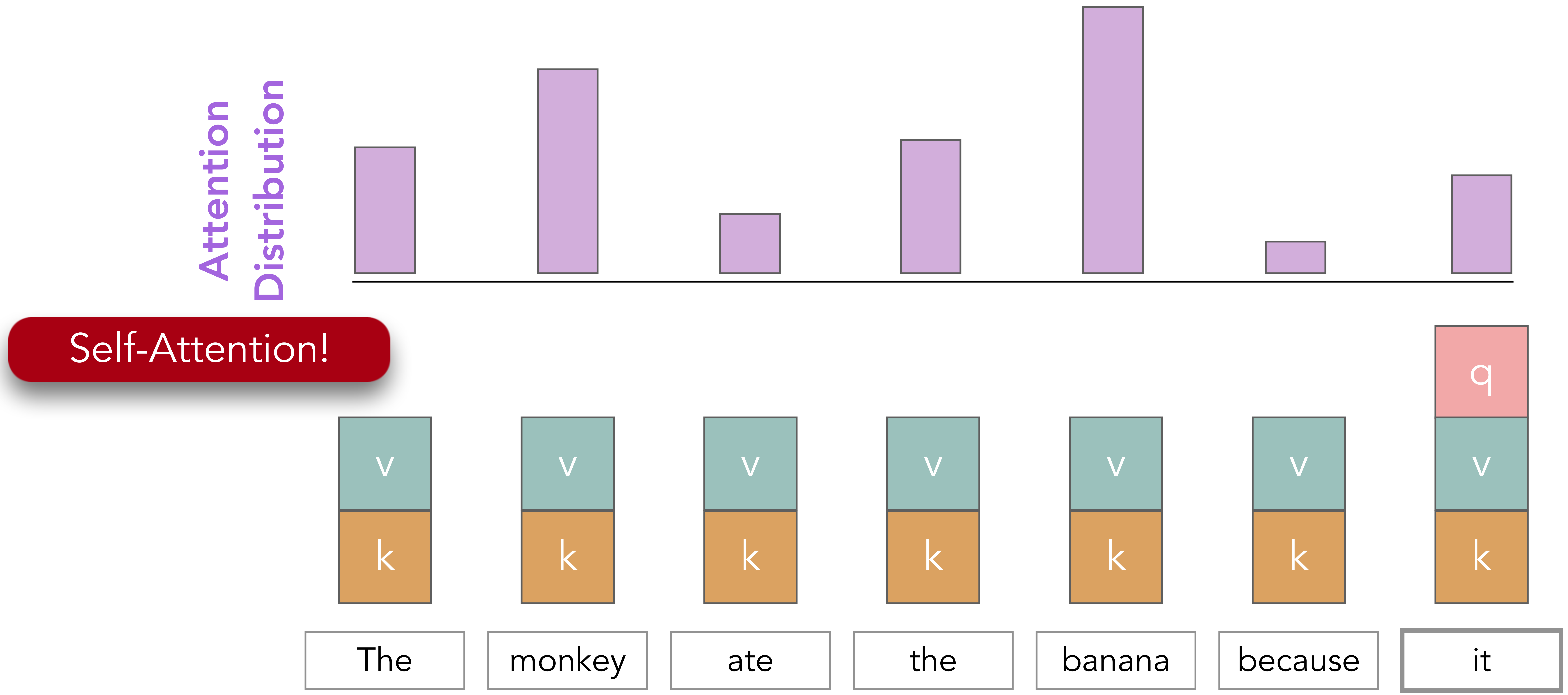

Attention Distribution
q
The
monkey
ate
the
banana
because
it


Attention Distribution
q
v
v
v
v
v
v
v
k
k
k
k
k
k
k
The
monkey
ate
the
banana
because
it

Attention in the decoder



Attention in the decoder



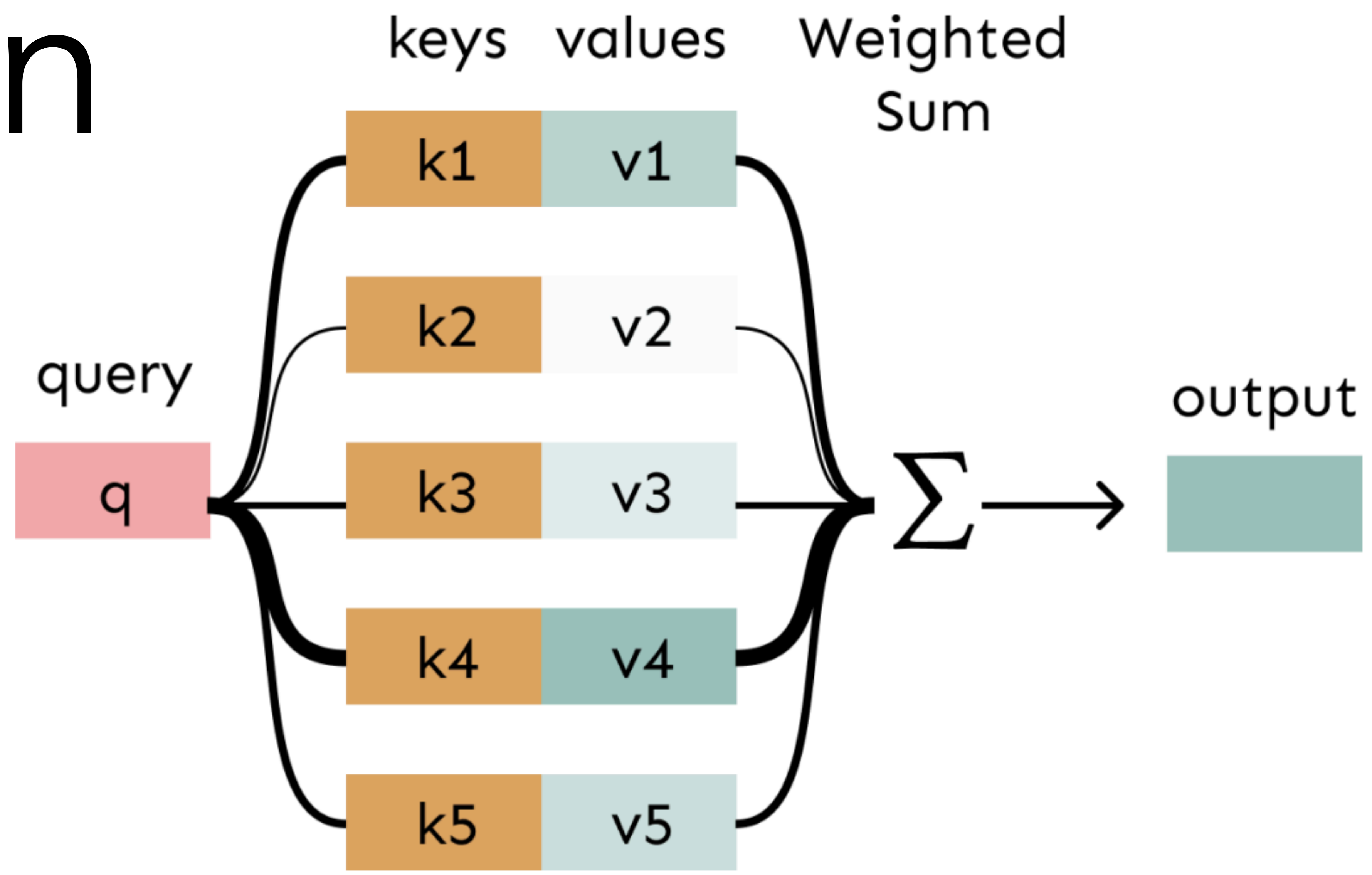
Lecture Outline

- Announcements
- Recap: Seq2Seq and Attention
- More on Attention
- Transformers: Self-Attention Networks
  - Multiheaded Attention
  - Positional Embeddings
  - Transformer Blocks
- Transformers as Encoders, Decoders and Encoder-Decoders

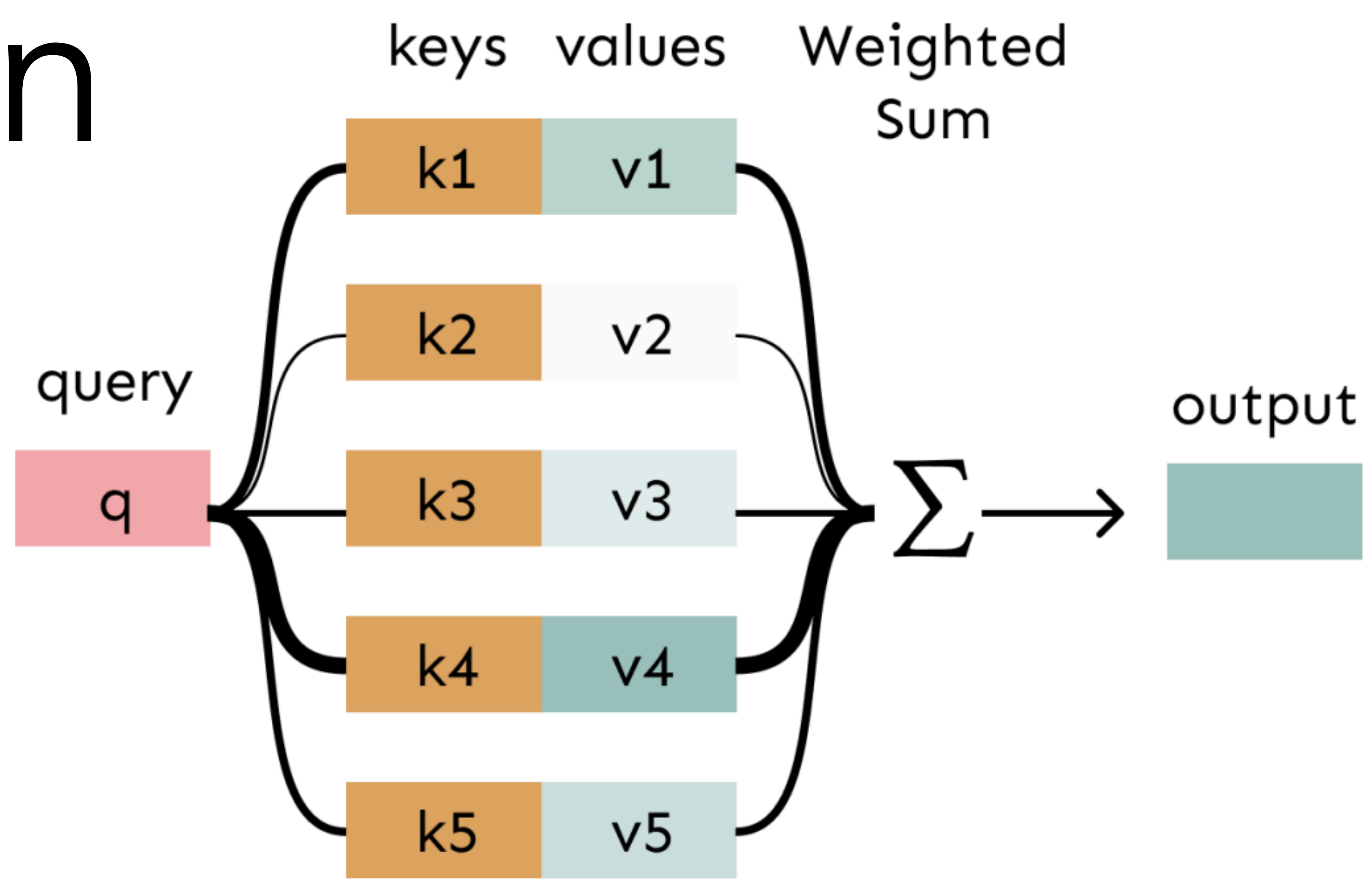
Transformers:
Self-Attention Networks

Self-Attention

Keys, Queries, Values from the same sequence



Self-Attention

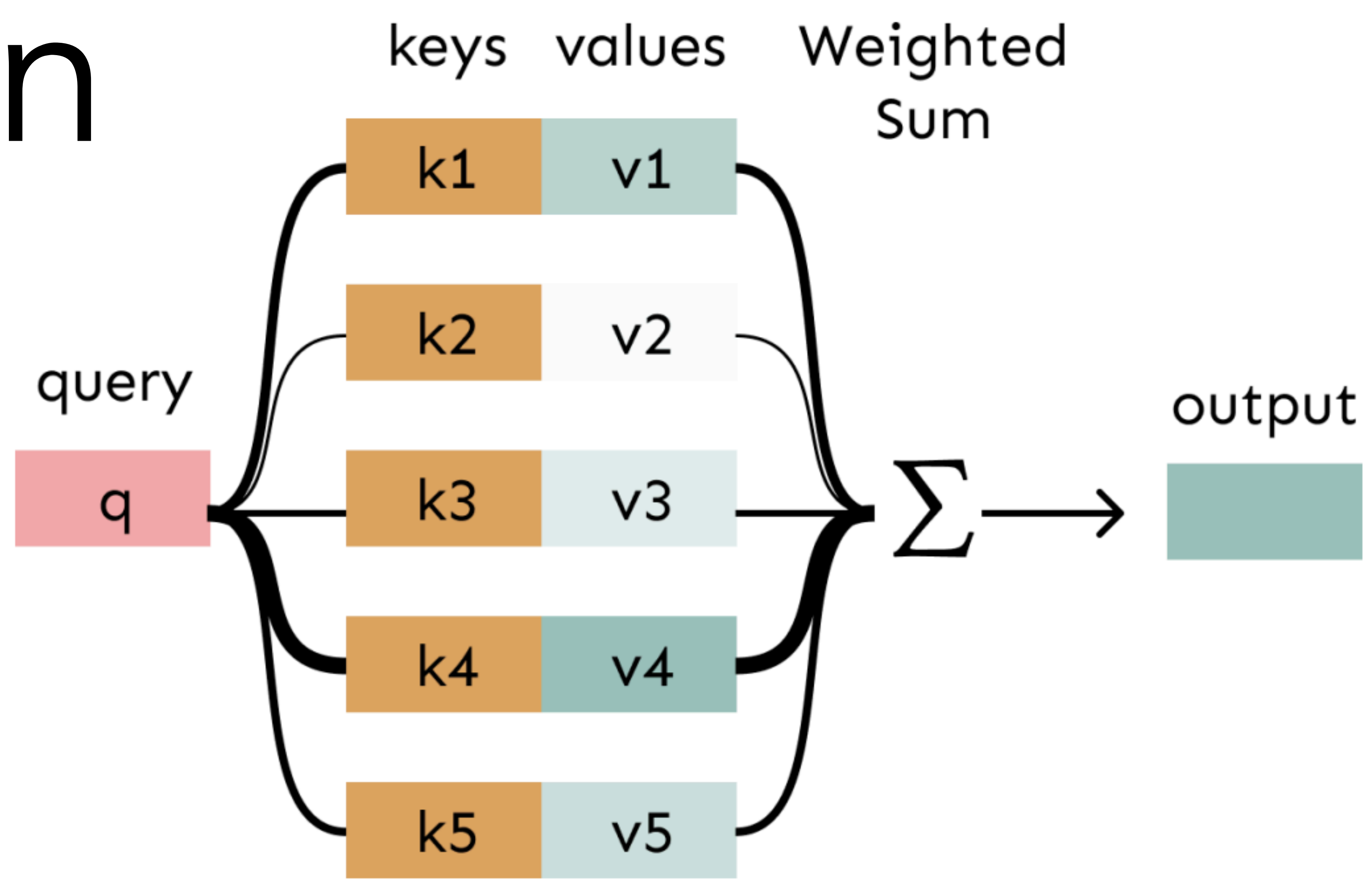


Keys, Queries, Values from the same sequence

Let $w_{1:N}$ be a sequence of words in vocabulary V

For each w_i , let $\mathbf{x}_i = \mathbf{E}_{w_i}$, where $\mathbf{E} \in \mathbb{R}^{d \times V}$ is an embedding matrix.

Self-Attention



Keys, Queries, Values from the same sequence

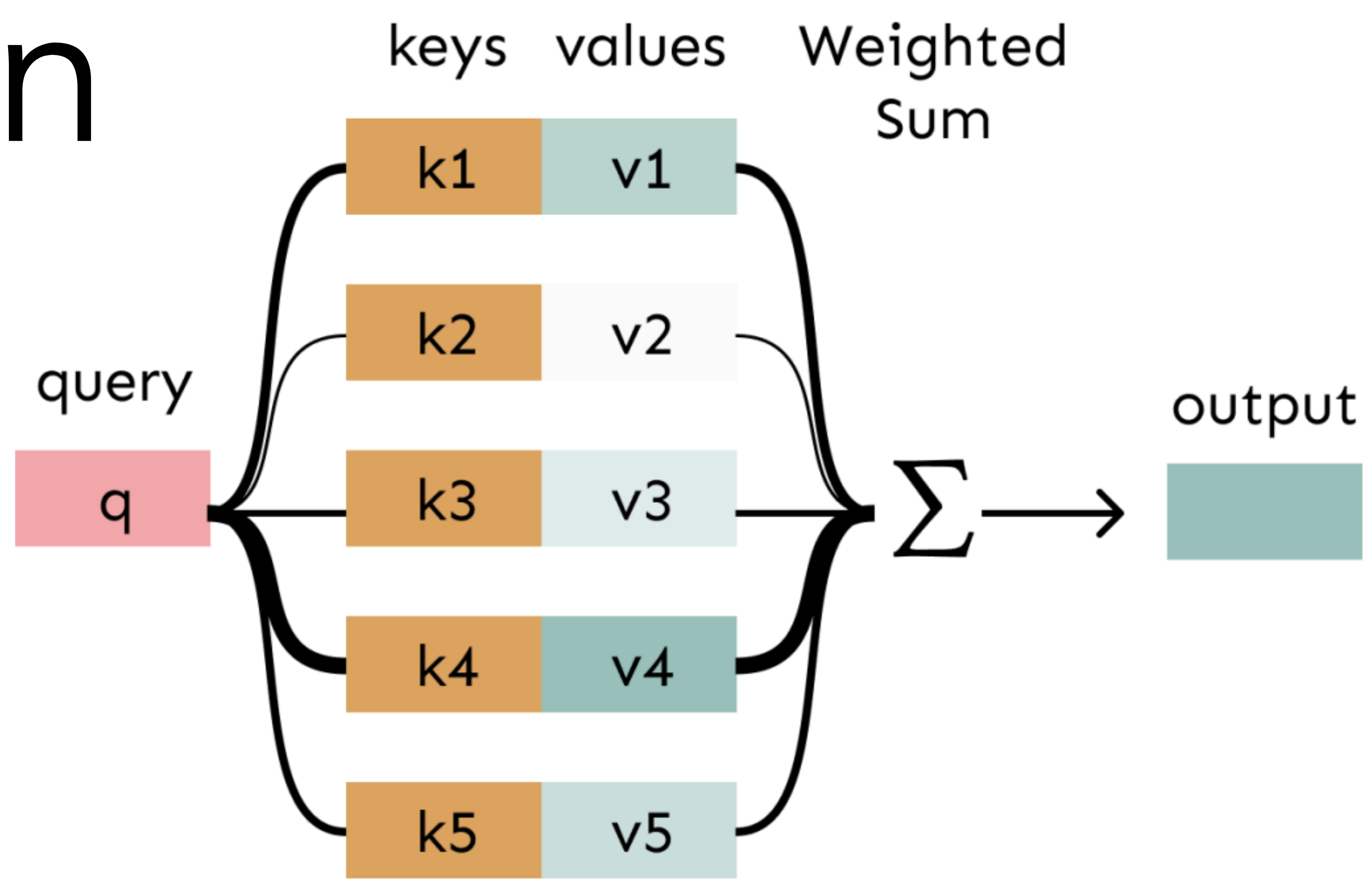
Let $w_{1:N}$ be a sequence of words in vocabulary V

For each w_i , let $\mathbf{x}_i = \mathbf{E}_{w_i}$, where $\mathbf{E} \in \mathbb{R}^{d \times V}$ is an embedding matrix.

1. Transform each word embedding with weight matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, each in $\mathbb{R}^{d \times d}$

$q_i = Q\boldsymbol{x_i}$ (queries) $k_i = K\boldsymbol{x_i}$ (keys) $v_i = V\boldsymbol{x_i}$ (values)

Self-Attention



Keys, Queries, Values from the same sequence

Let $w_{1:N}$ be a sequence of words in vocabulary V

For each w_i , let $\mathbf{x}_i = \mathbf{E}_{w_i}$, where $\mathbf{E} \in \mathbb{R}^{d \times V}$ is an embedding matrix.

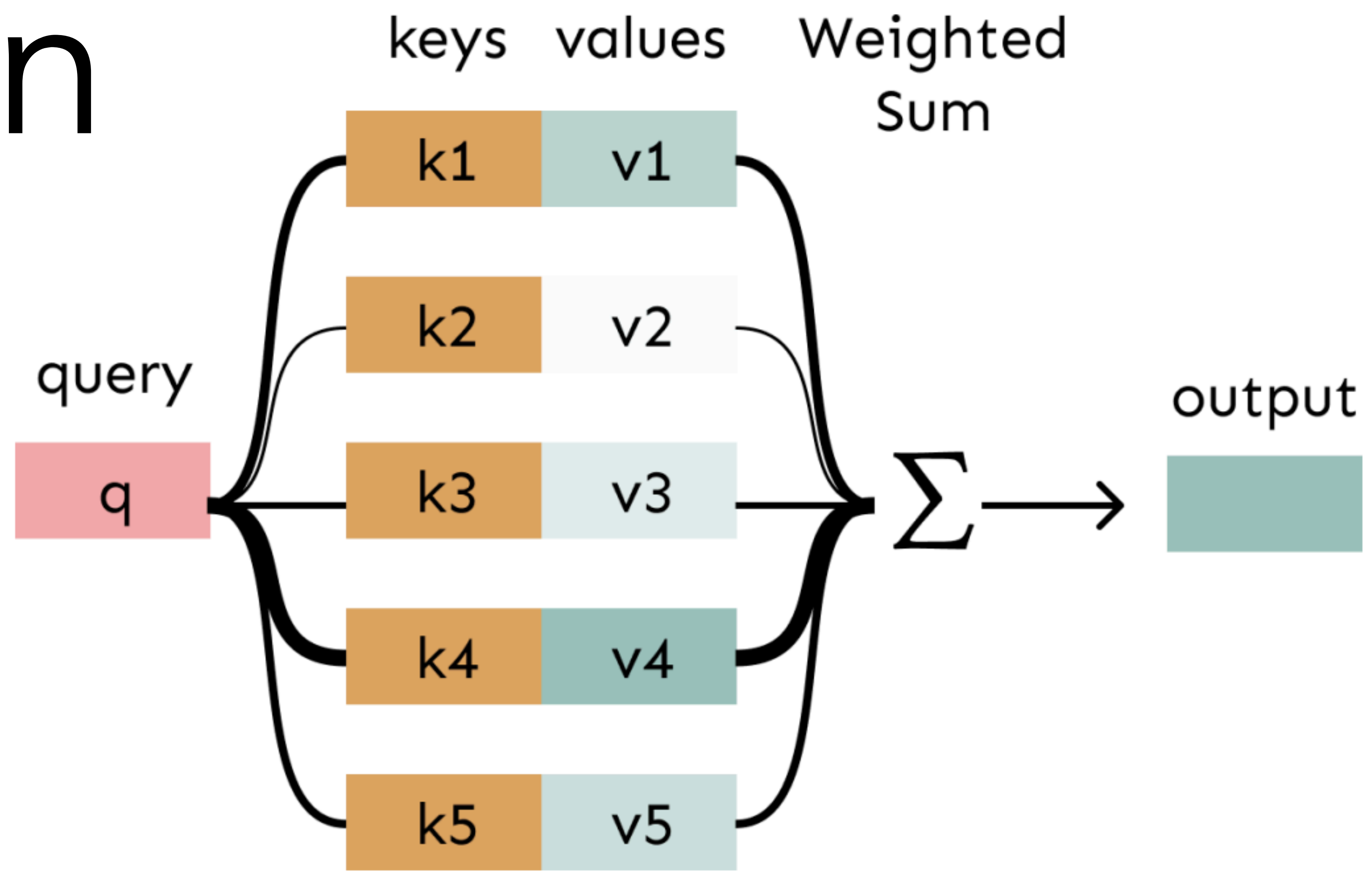
1. Transform each word embedding with weight matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, each in $\mathbb{R}^{d \times d}$

$q_i = Q\boldsymbol{x_i}$ (queries) $\quad$ $k_i = K\boldsymbol{x_i}$ (keys) $\quad$ $v_i = V\boldsymbol{x_i}$ (values)

2. Compute pairwise similarities between keys and queries; normalize with softmax

$$\boldsymbol{e}_{ij} = \boldsymbol{q}_i^{\top}\boldsymbol{k}_j \qquad \boldsymbol{\alpha}_{ij} = \frac{\exp(\boldsymbol{e}_{ij})}{\sum_{j'}\exp(\boldsymbol{e}_{ij'})}$$

Self-Attention



Keys, Queries, Values from the same sequence

Let $w_{1:N}$ be a sequence of words in vocabulary V

For each w_i , let $\mathbf{x}_i = \mathbf{E}_{w_i}$, where $\mathbf{E} \in \mathbb{R}^{d \times V}$ is an embedding matrix.

1. Transform each word embedding with weight matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, each in $\mathbb{R}^{d \times d}$

$q_i = Q\boldsymbol{x_i}$ (queries) $\quad$ $k_i = K\boldsymbol{x_i}$ (keys) $\quad$ $v_i = V\boldsymbol{x_i}$ (values)

2. Compute pairwise similarities between keys and queries; normalize with softmax

$$\boldsymbol{e}_{ij} = \boldsymbol{q}_i^\top \boldsymbol{k}_j \qquad \boldsymbol{\alpha}_{ij} = \frac{\exp(\boldsymbol{e}_{ij})}{\sum_{j'} \exp(\boldsymbol{e}_{ij'})}$$

3. Compute output for each word as weighted sum of values

$$\boldsymbol{o}_i = \sum_j \boldsymbol{\alpha}_{ij} \, v_i$$

Self-Attention as Matrix Multiplications

Self-Attention as Matrix Multiplications

- Key-query-value attention is typically computed as matrices.

Self-Attention as Matrix Multiplications

- Key-query-value attention is typically computed as matrices.
  - Let $\mathbf{X} = [\mathbf{x}_1; \ldots; \mathbf{x}_n] \in \mathbb{R}^{n \times d}$ be the concatenation of input vectors

Self-Attention as Matrix Multiplications

- Key-query-value attention is typically computed as matrices.
  - Let $\mathbf{X} = [\mathbf{x}_1; \ldots; \mathbf{x}_n] \in \mathbb{R}^{n \times d}$ be the concatenation of input vectors
  - First, note that $\mathbf{XK} \in \mathbb{R}^{n \times d}$, $\mathbf{XQ} \in \mathbb{R}^{n \times d}$, and $\mathbf{XV} \in \mathbb{R}^{n \times d}$

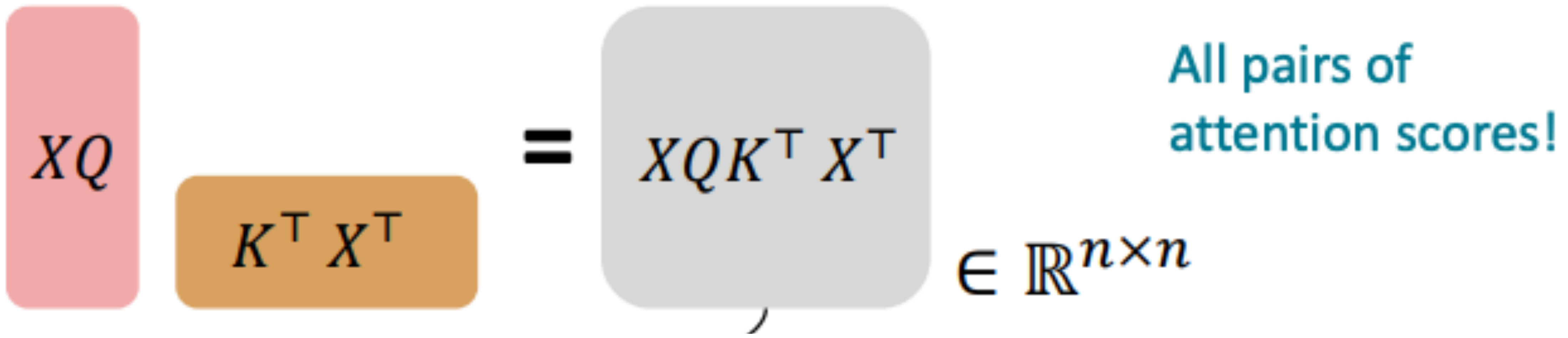
Self-Attention as Matrix Multiplications

- Key-query-value attention is typically computed as matrices.
  - Let $\mathbf{X} = [\mathbf{x}_1; \ldots; \mathbf{x}_n] \in \mathbb{R}^{n \times d}$ be the concatenation of input vectors
  - First, note that $\mathbf{XK} \in \mathbb{R}^{n \times d}$, $\mathbf{XQ} \in \mathbb{R}^{n \times d}$, and $\mathbf{XV} \in \mathbb{R}^{n \times d}$
  - The output is defined as $\mathrm{softmax}(\mathbf{XQ}(\mathbf{XK})^T)\mathbf{XV} \in \mathbb{R}^{n \times d}$

Self-Attention as Matrix Multiplications

- Key-query-value attention is typically computed as matrices.
  - Let $\mathbf{X} = [\mathbf{x}_1; \ldots; \mathbf{x}_n] \in \mathbb{R}^{n \times d}$ be the concatenation of input vectors
  - First, note that $\mathbf{XK} \in \mathbb{R}^{n \times d}$, $\mathbf{XQ} \in \mathbb{R}^{n \times d}$, and $\mathbf{XV} \in \mathbb{R}^{n \times d}$
  - The output is defined as $\mathrm{softmax}(\mathbf{XQ}(\mathbf{XK})^T)\mathbf{XV} \in \mathbb{R}^{n \times d}$

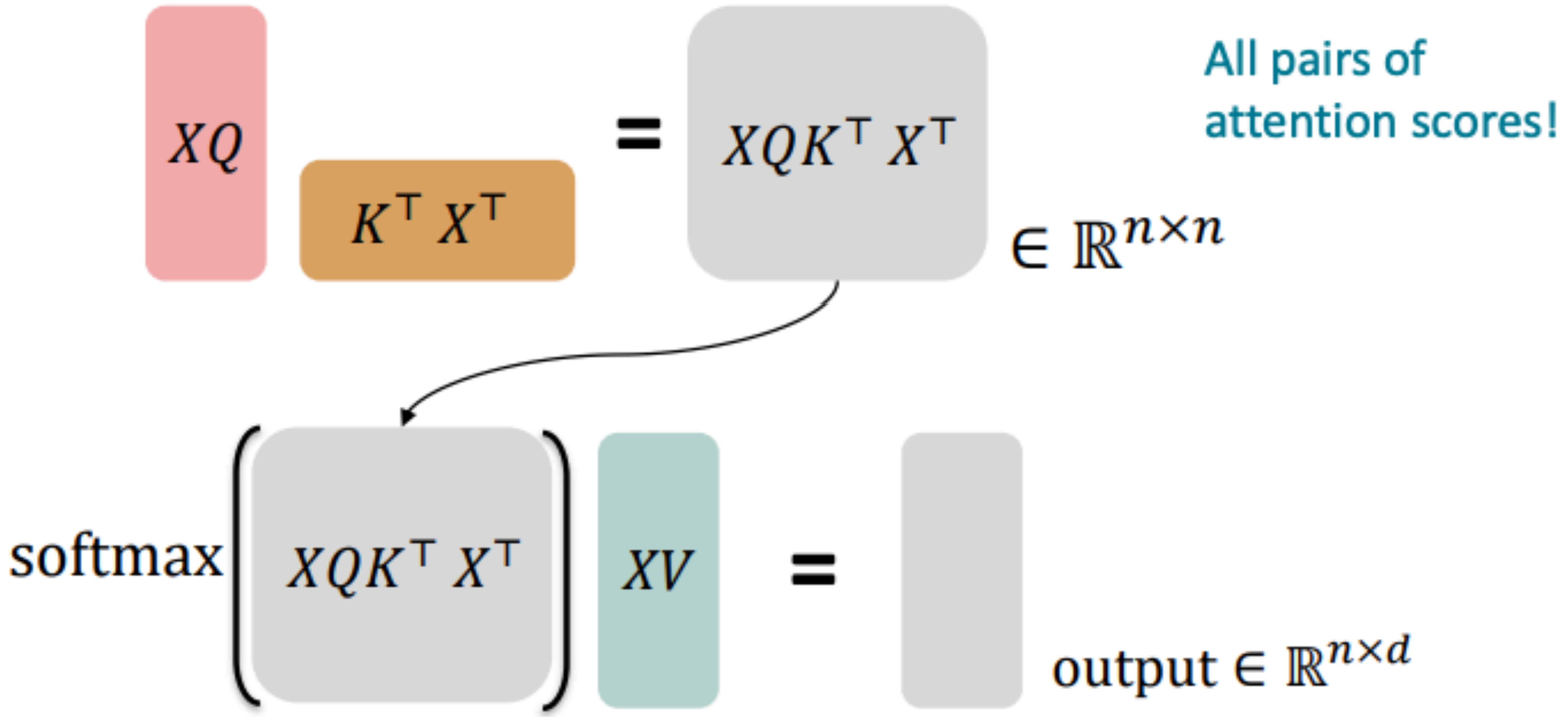
First, take the query-key dot products in one matrix multiplication: $\mathbf{XQ}(\mathbf{XK})^T$



Self-Attention as Matrix Multiplications

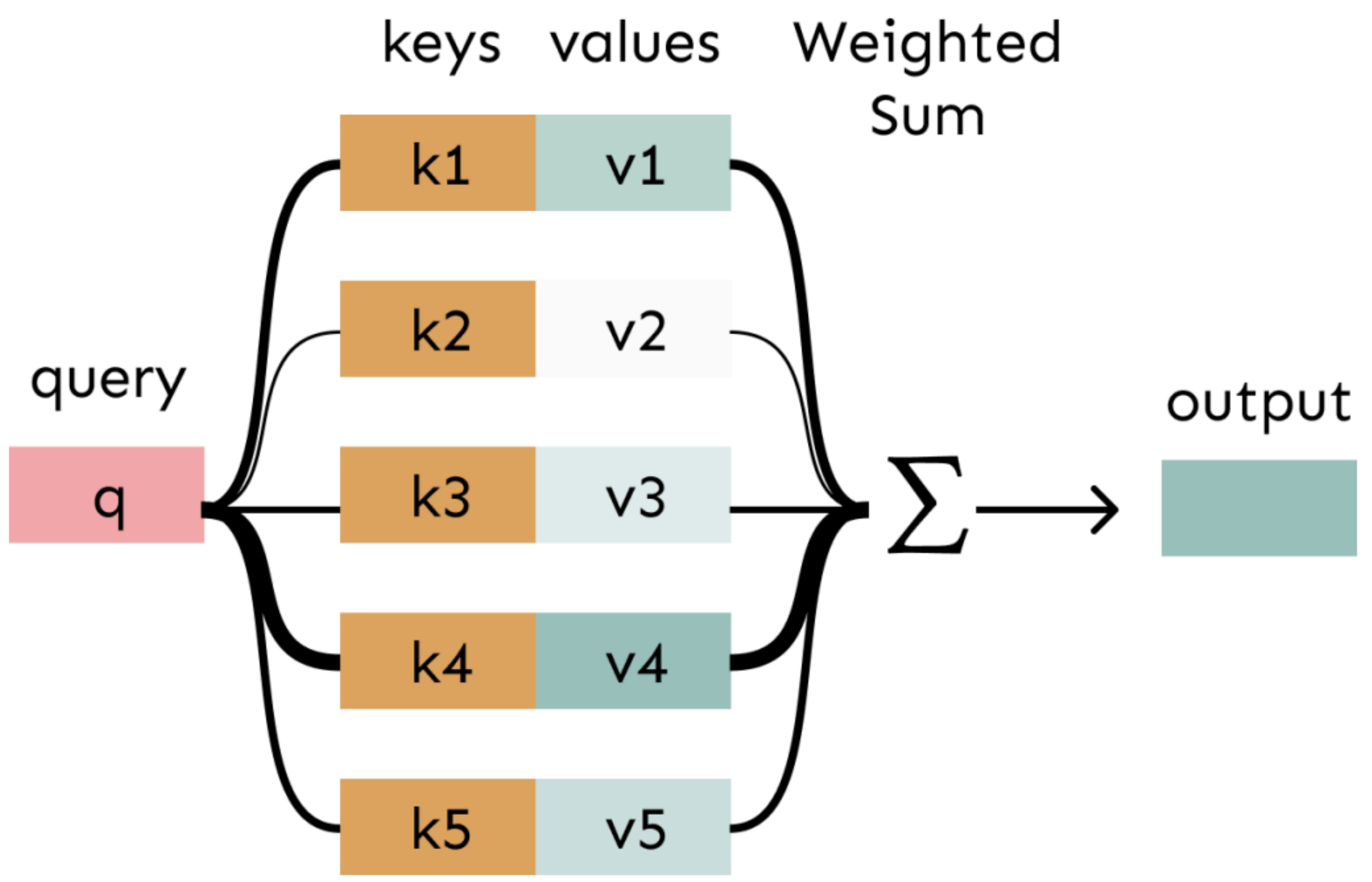
- Key-query-value attention is typically computed as matrices.
  - Let $\mathbf{X} = [\mathbf{x}_1; \ldots; \mathbf{x}_n] \in \mathbb{R}^{n \times d}$ be the concatenation of input vectors
  - First, note that $\mathbf{XK} \in \mathbb{R}^{n \times d}$, $\mathbf{XQ} \in \mathbb{R}^{n \times d}$, and $\mathbf{XV} \in \mathbb{R}^{n \times d}$
  - The output is defined as softmax$(\mathbf{XQ}(\mathbf{XK})^T)\mathbf{XV} \in \mathbb{R}^{n \times d}$

First, take the query-key dot products in one matrix multiplication: $\mathbf{XQ}(\mathbf{XK})^T$

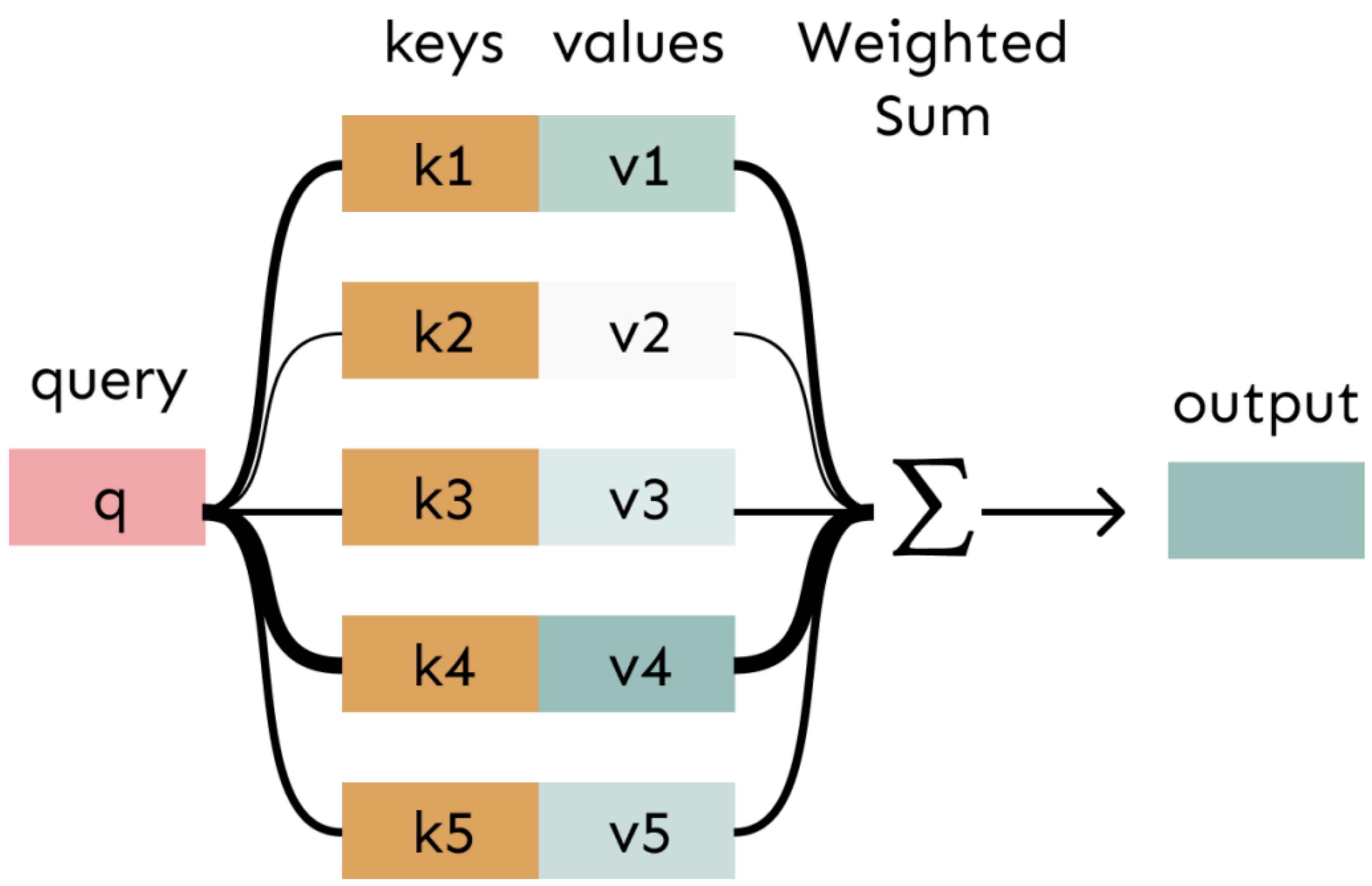


Next, softmax, and compute the weighted average with another matrix multiplication.

Why Self-Attention?

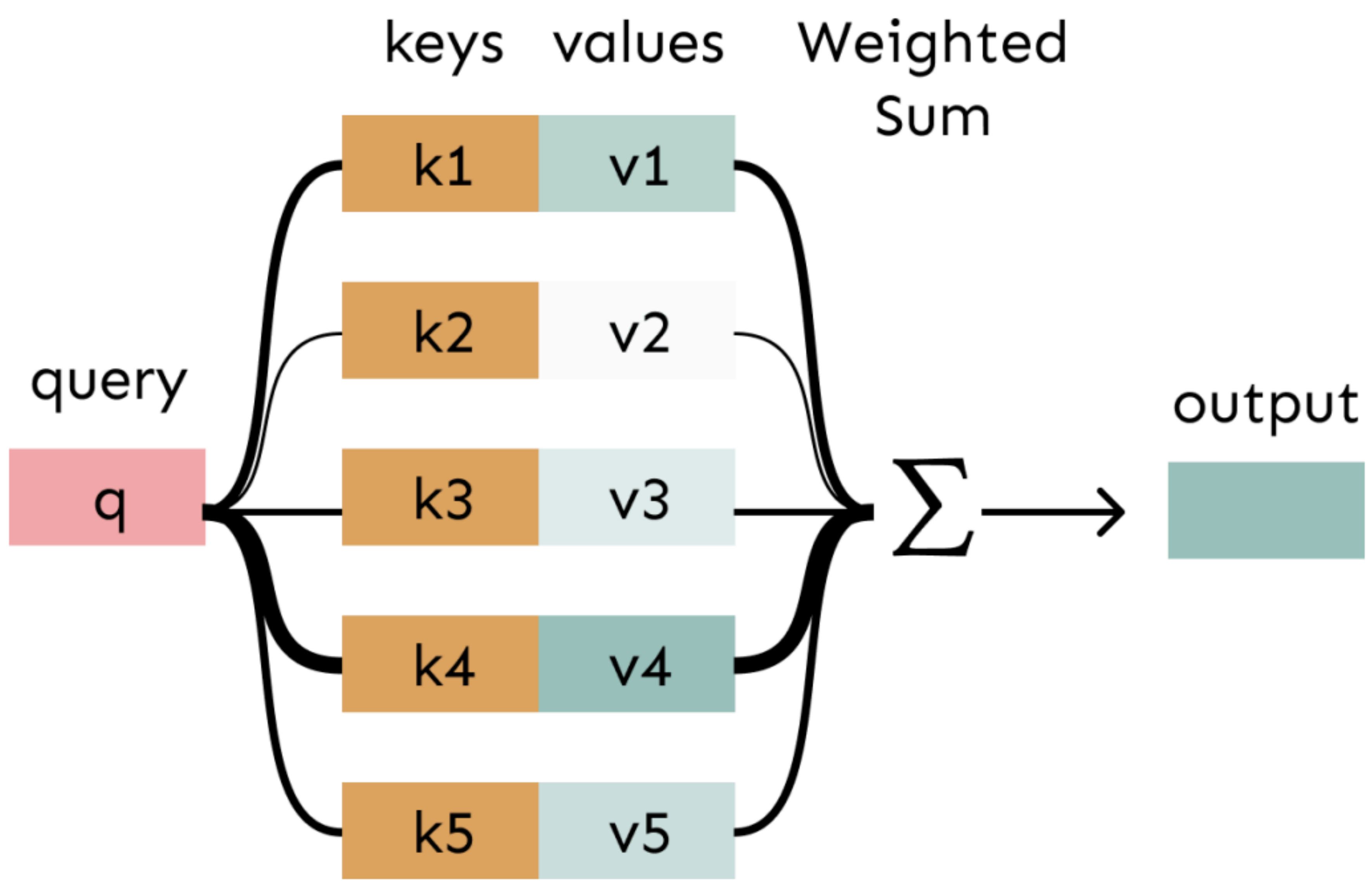


Why Self-Attention?



- Allows a network to directly extract and use information from arbitrarily large contexts without the need to pass it through intermediate recurrent connections as in RNNs

Why Self-Attention?



- Allows a network to directly extract and use information from arbitrarily large contexts without the need to pass it through intermediate recurrent connections as in RNNs
- Used often with feedforward networks!

Transformers are Self-Attention Networks

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

Transformers are Self-Attention Networks

- Self-Attention is the key innovation behind Transformers!

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

USC Viterbi

Transformers are Self-Attention Networks

- Self-Attention is the key innovation behind Transformers!
- Transformers (self-attention networks) map sequences of input vectors $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to sequences of output vectors $(\mathbf{y}_1, \ldots, \mathbf{y}_n)$ of the same length.

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

Transformers are Self-Attention Networks

- Self-Attention is the key innovation behind Transformers!
- Transformers (self-attention networks) map sequences of input vectors $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to sequences of output vectors $(\mathbf{y}_1, \ldots, \mathbf{y}_n)$ of the same length.
- Made up of stacks of Transformer blocks
  - each of which is a multilayer network made by combining
    - simple linear layers,
    - feedforward networks, and
    - self-attention layers
  - No more recurrent connections!

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

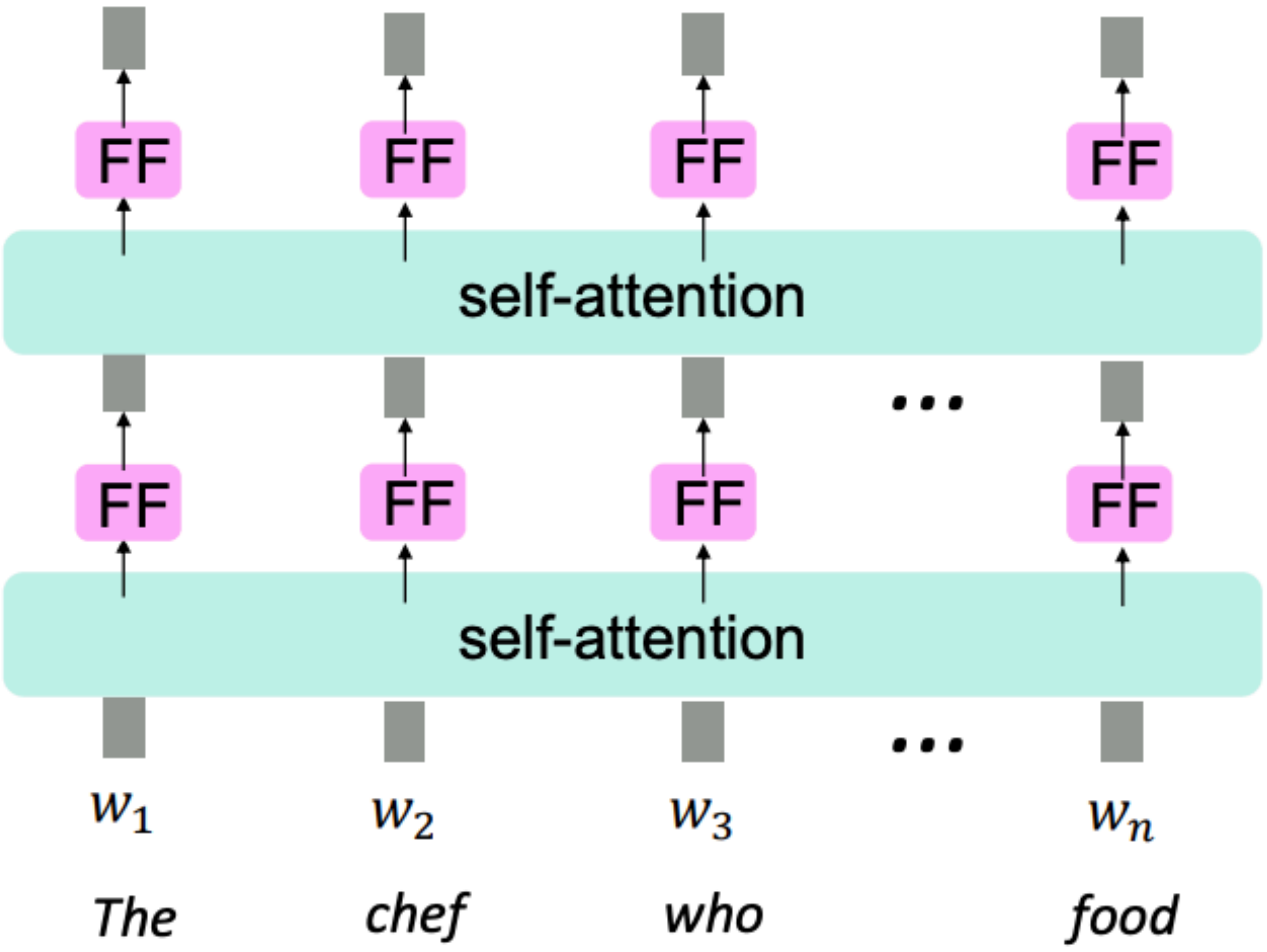
Self-Attention and Weighted Averages

Self-Attention and Weighted Averages

- **Problem**: there are no *element-wise* nonlinearities in self-attention; stacking more self-attention layers just re-averages value vectors

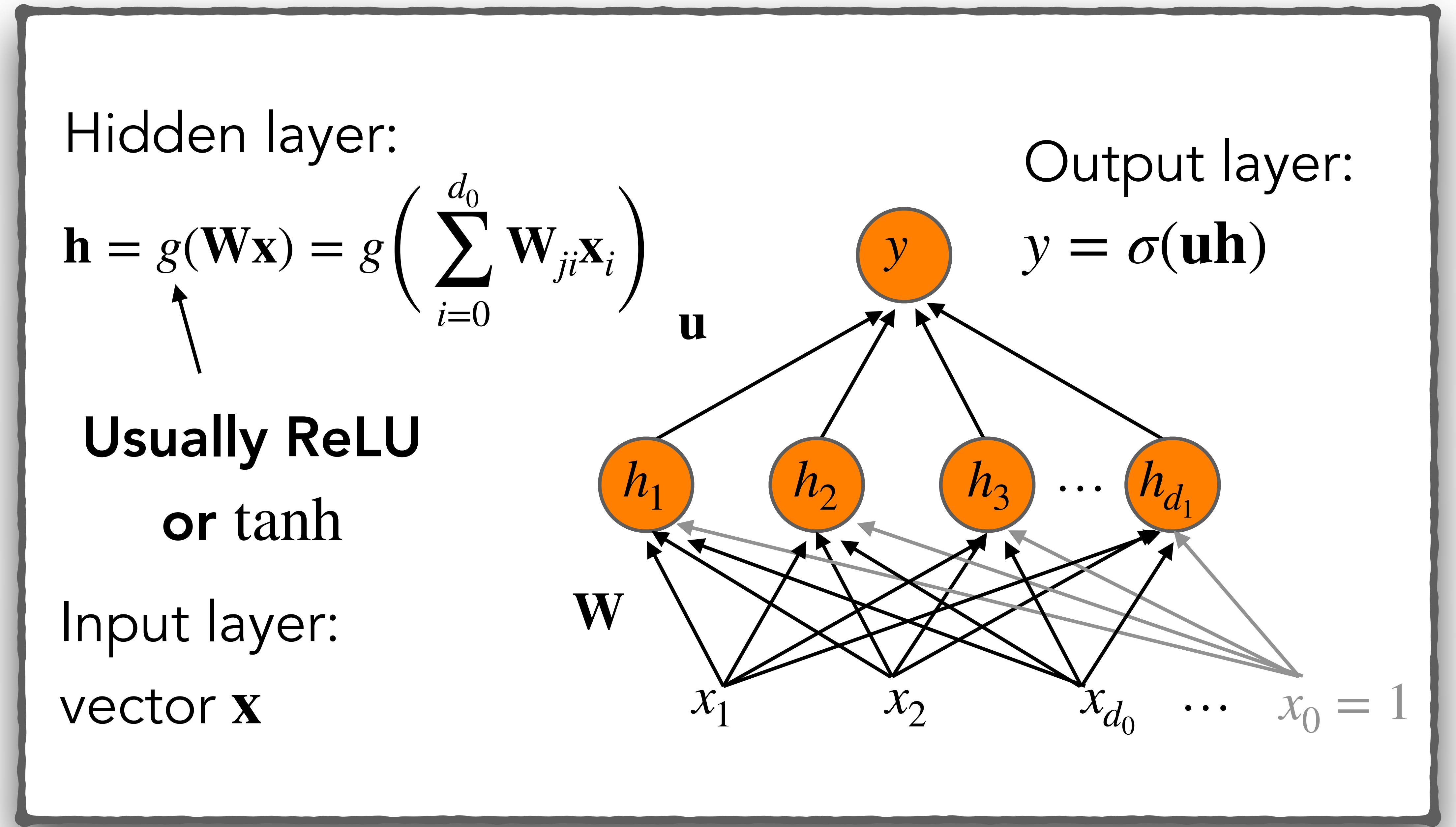
Self-Attention and Weighted Averages

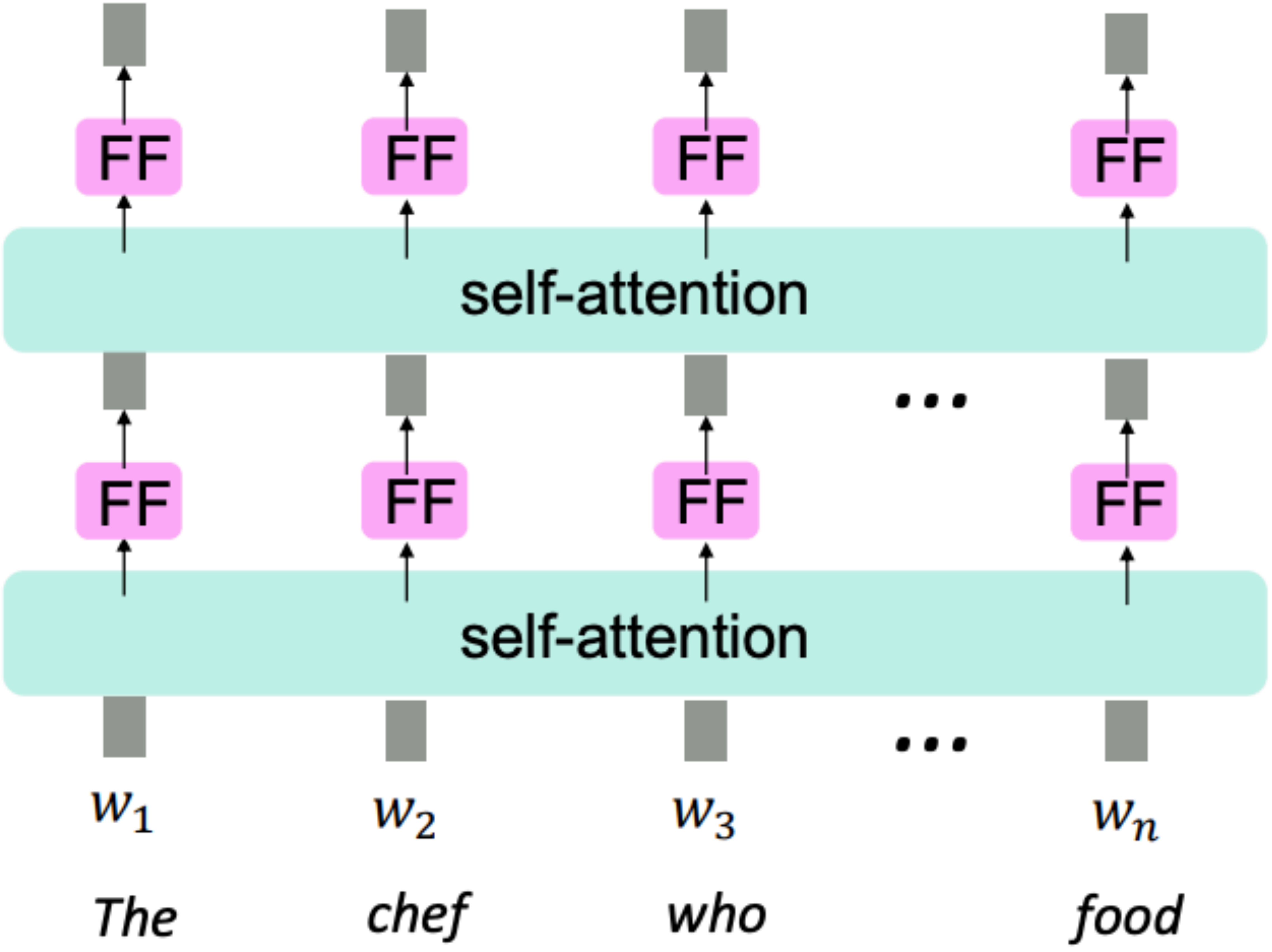
- **Problem**: there are no *element-wise* nonlinearities in self-attention; stacking more self-attention layers just re-averages value vectors
- **Solution**: add a feed-forward network to post-process each output vector.



Self-Attention and Weighted Averages

- **Problem**: there are no *element-wise* nonlinearities in self-attention; stacking more self-attention layers just re-averages value vectors
- **Solution**: add a feed-forward network to post-process each output vector.





Self Attention and Future Information

Self Attention and Future Information

- **Problem**: Need to ensure we don't "look at the future" when predicting a sequence during training

Self Attention and Future Information

- **Problem**: Need to ensure we don't "look at the future" when predicting a sequence during training
  - e.g. Target sentence in machine translation or generated sentence in language modeling

Self Attention and Future Information

- **Problem**: Need to ensure we don't "look at the future" when predicting a sequence during training
  - e.g. Target sentence in machine translation or generated sentence in language modeling
  - To use self-attention in decoders, we need to ensure we can't peek at the future.

Self Attention and Future Information

- **Problem**: Need to ensure we don't "look at the future" when predicting a sequence during training
  - e.g. Target sentence in machine translation or generated sentence in language modeling
  - To use self-attention in decoders, we need to ensure we can't peek at the future.
- **Solution** (**Naïve**): At every time step, we could change the set of keys and queries to include only past words.

Self Attention and Future Information

- **Problem**: Need to ensure we don't "look at the future" when predicting a sequence during training
  - e.g. Target sentence in machine translation or generated sentence in language modeling
  - To use self-attention in decoders, we need to ensure we can't peek at the future.
- **Solution** (**Naïve**): At every time step, we could change the set of keys and queries to include only past words.
  - (Inefficient!)

Self Attention and Future Information

- **Problem**: Need to ensure we don't "look at the future" when predicting a sequence during training
  - e.g. Target sentence in machine translation or generated sentence in language modeling
  - To use self-attention in decoders, we need to ensure we can't peek at the future.
- **Solution** (**Naïve**): At every time step, we could change the set of keys and queries to include only past words.
  - (Inefficient!)
- **Solution:** To enable parallelization, we mask out attention to future words by setting attention scores to $-\infty$

Self Attention and Future Information

- **Problem**: Need to ensure we don't "look at the future" when predicting a sequence during training
  - e.g. Target sentence in machine translation or generated sentence in language modeling
  - To use self-attention in decoders, we need to ensure we can't peek at the future.
- **Solution (Naïve)**: At every time step, we could change the set of keys and queries to include only past words.
  - (Inefficient!)
- **Solution:** To enable parallelization, we mask out attention to future words by setting attention scores to $-\infty$

	[START]	The	chef	who
[START]		$-\infty$	$-\infty$	$-\infty$
The			$-\infty$	$-\infty$
chef				$-\infty$
who				

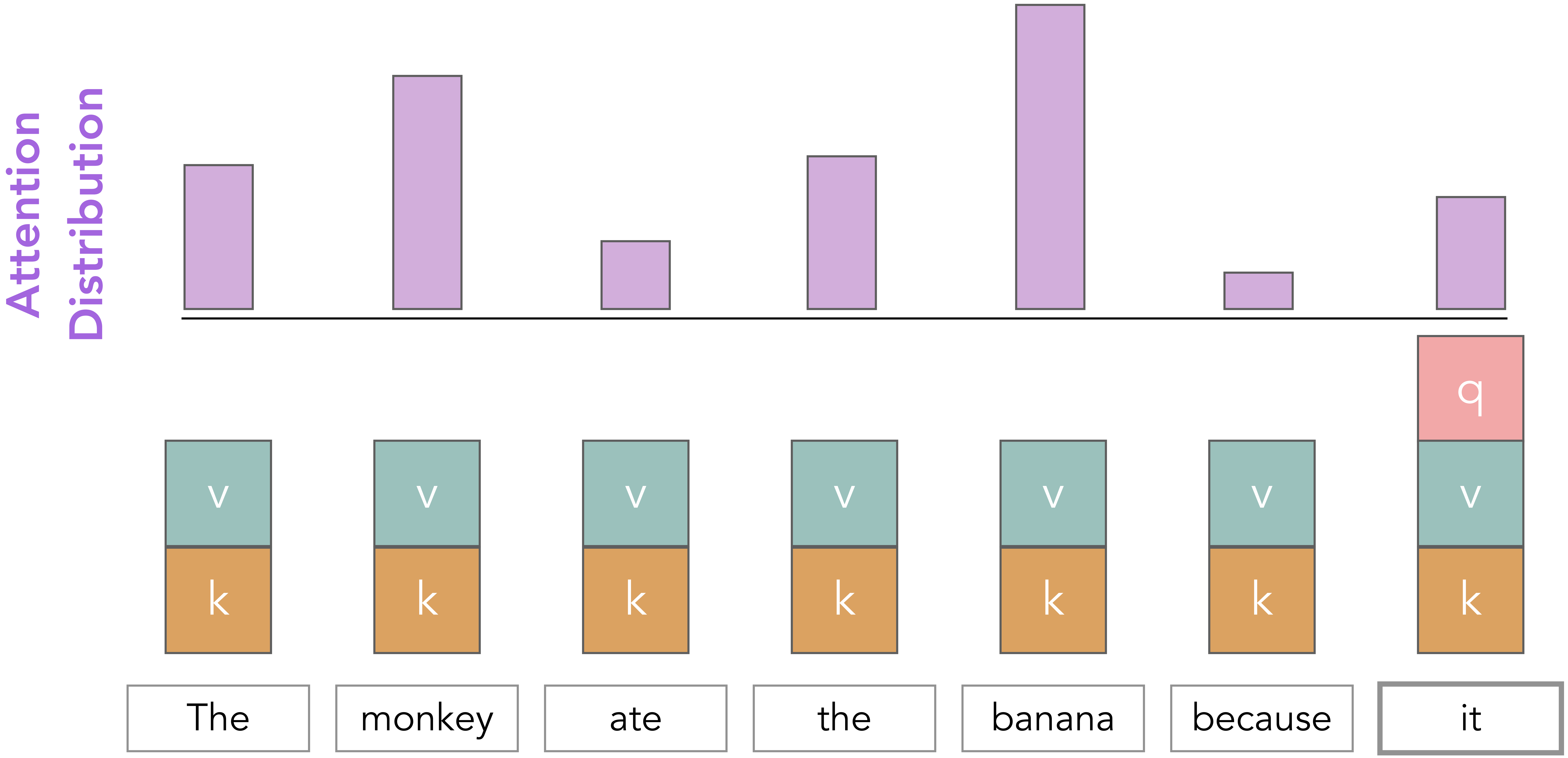
Self-Attention and Heads

Self-Attention and Heads

- What if we needed to pay attention to multiple different kinds of things e.g. entities, syntax

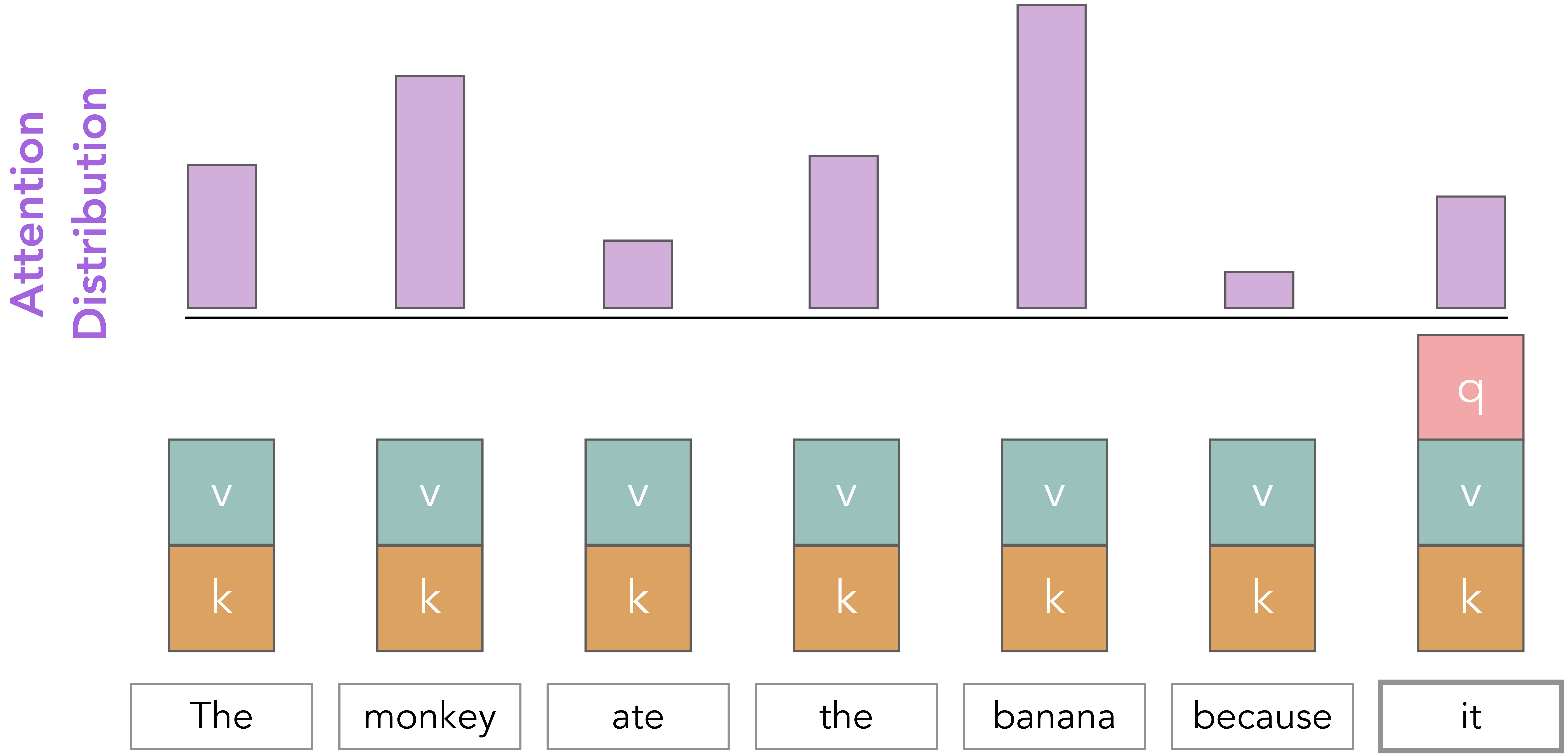
Self-Attention and Heads

- What if we needed to pay attention to multiple different kinds of things e.g. entities, syntax



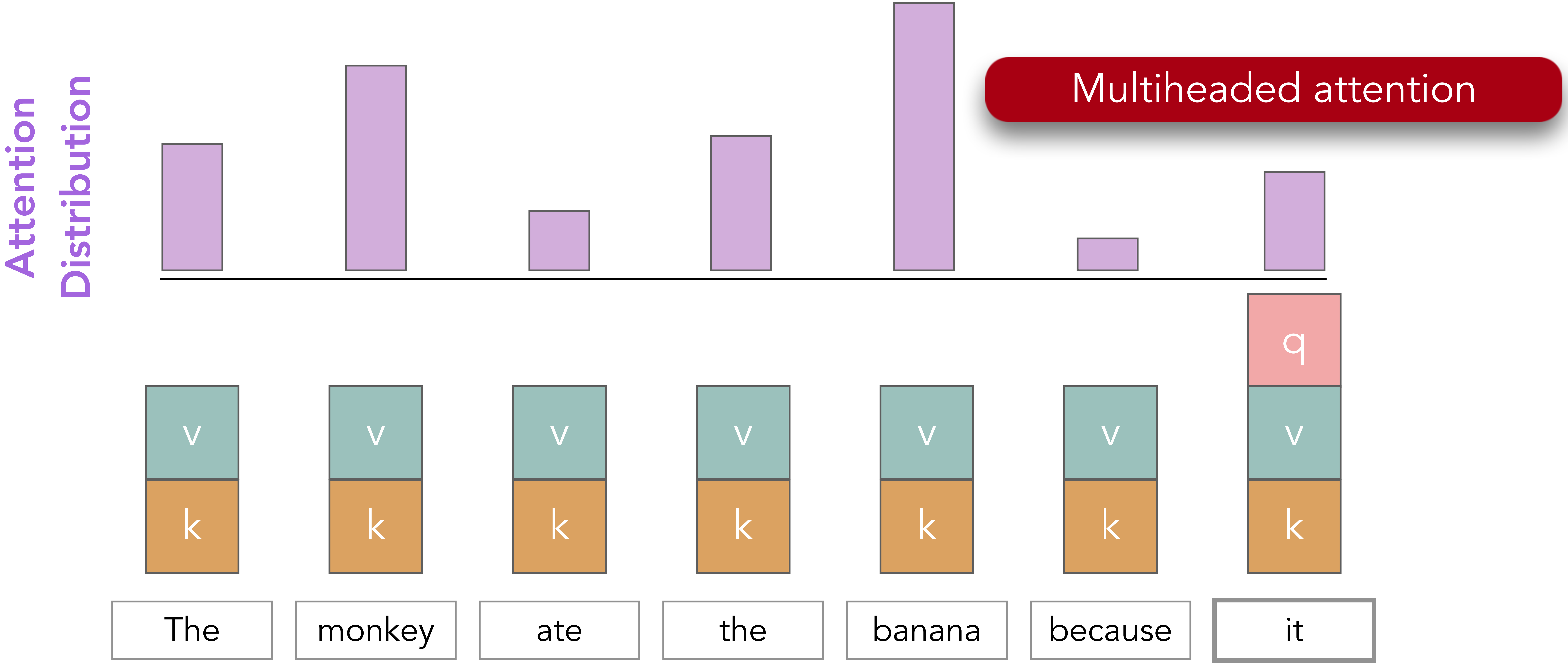
Self-Attention and Heads

- What if we needed to pay attention to multiple different kinds of things e.g. entities, syntax
- **Solution**: Consider multiple attention computations in parallel



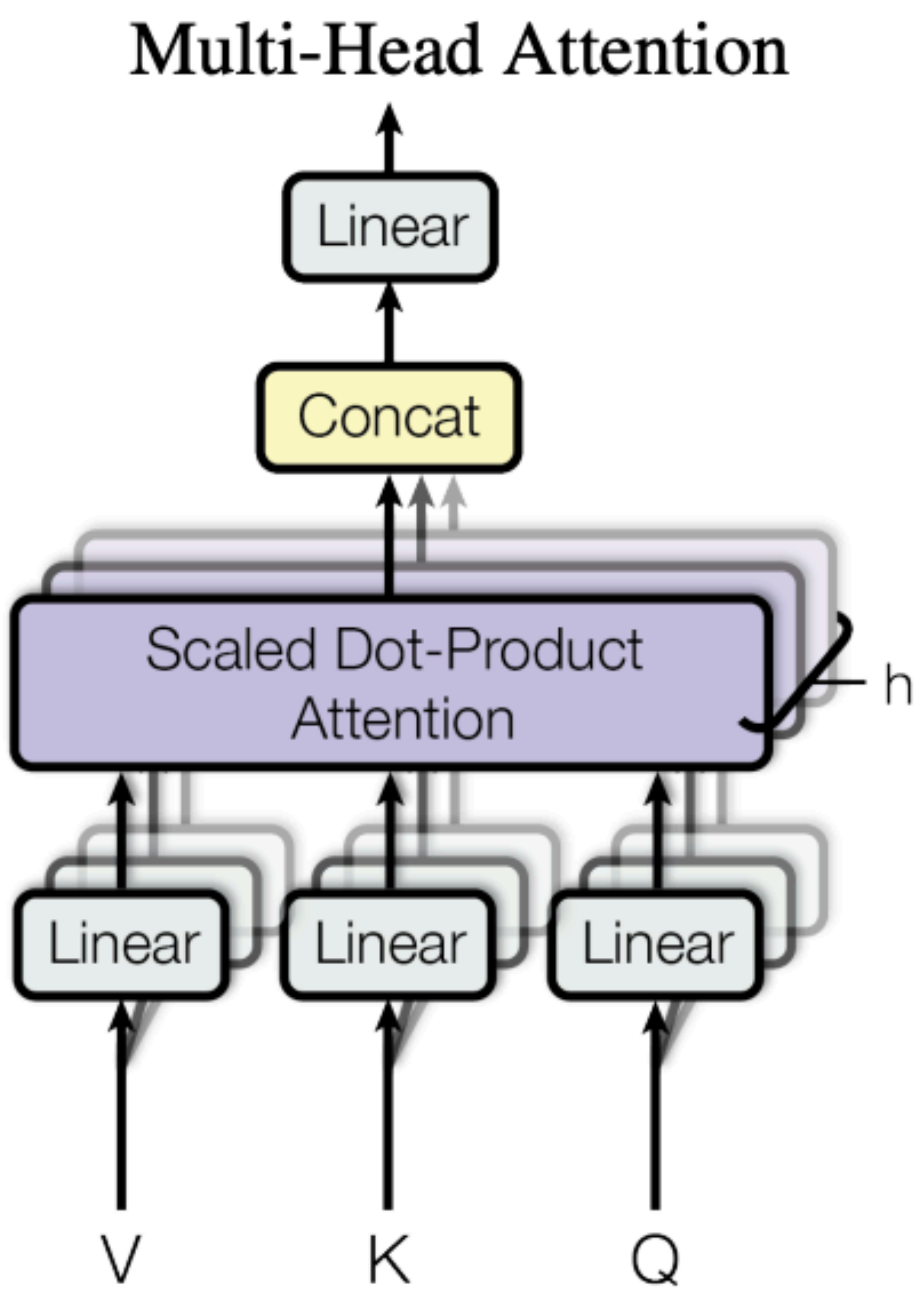
Self-Attention and Heads

- What if we needed to pay attention to multiple different kinds of things e.g. entities, syntax
- **Solution**: Consider multiple attention computations in parallel



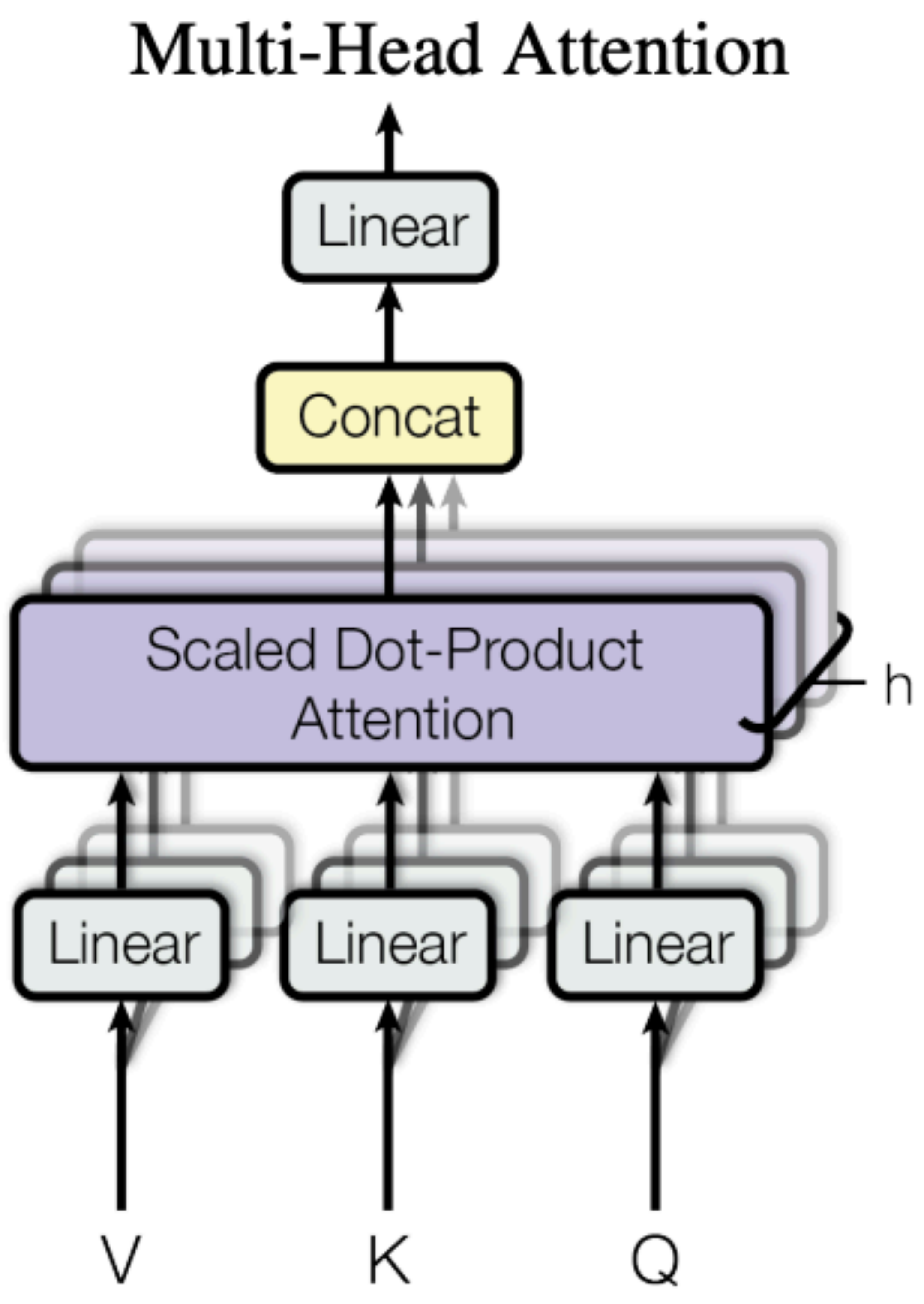
Transformers: Multiheaded Attention

Multi-headed attention



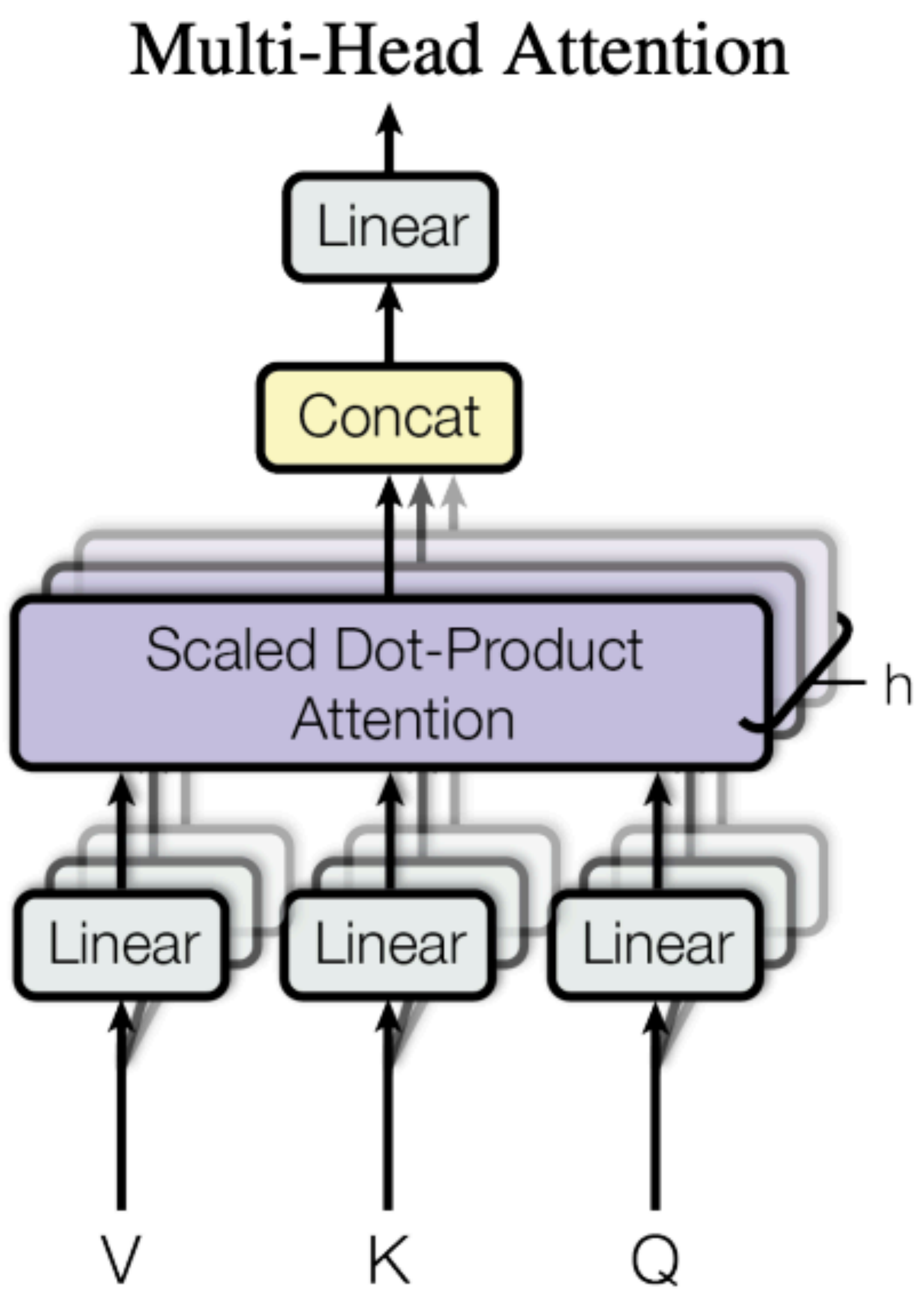
Multi-headed attention

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $\mathbf{x}_i^T \mathbf{Q}^T (\mathbf{K}\mathbf{x}_j)$ is high, but maybe we want to focus on different j for different reasons?



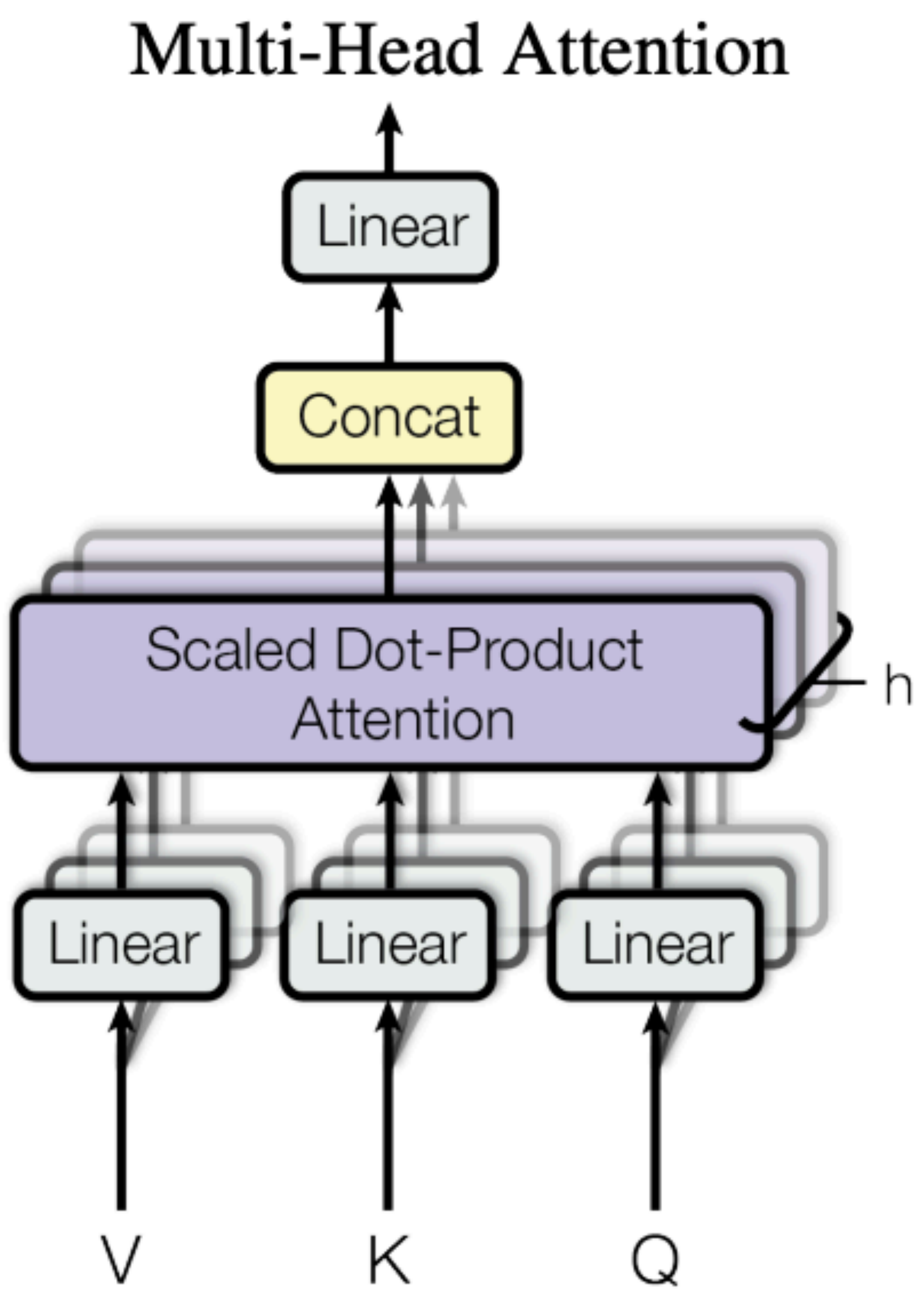
Multi-headed attention

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $\mathbf{x}_i^T \mathbf{Q}^T (\mathbf{K}\mathbf{x}_j)$ is high, but maybe we want to focus on different j for different reasons?
- Define multiple attention "heads" through multiple $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ matrices



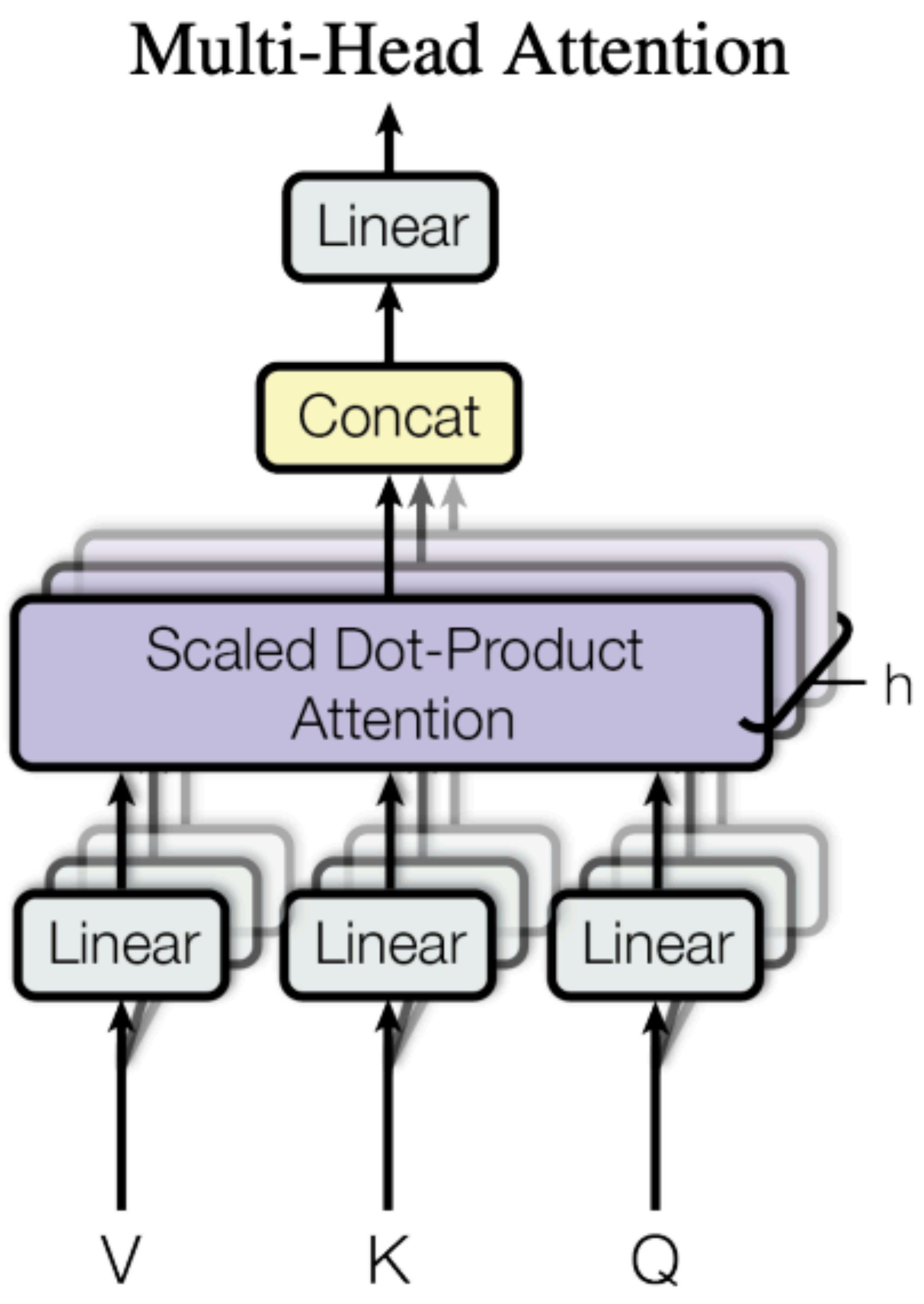
Multi-headed attention

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $\mathbf{x}_i^T \mathbf{Q}^T (\mathbf{K}\mathbf{x}_j)$ is high, but maybe we want to focus on different j for different reasons?
- Define multiple attention "heads" through multiple $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ matrices
- Let $\mathbf{Q}_l, \mathbf{K}_l, \mathbf{V}_l$, each in $\mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and $1 \leq l \leq h$.



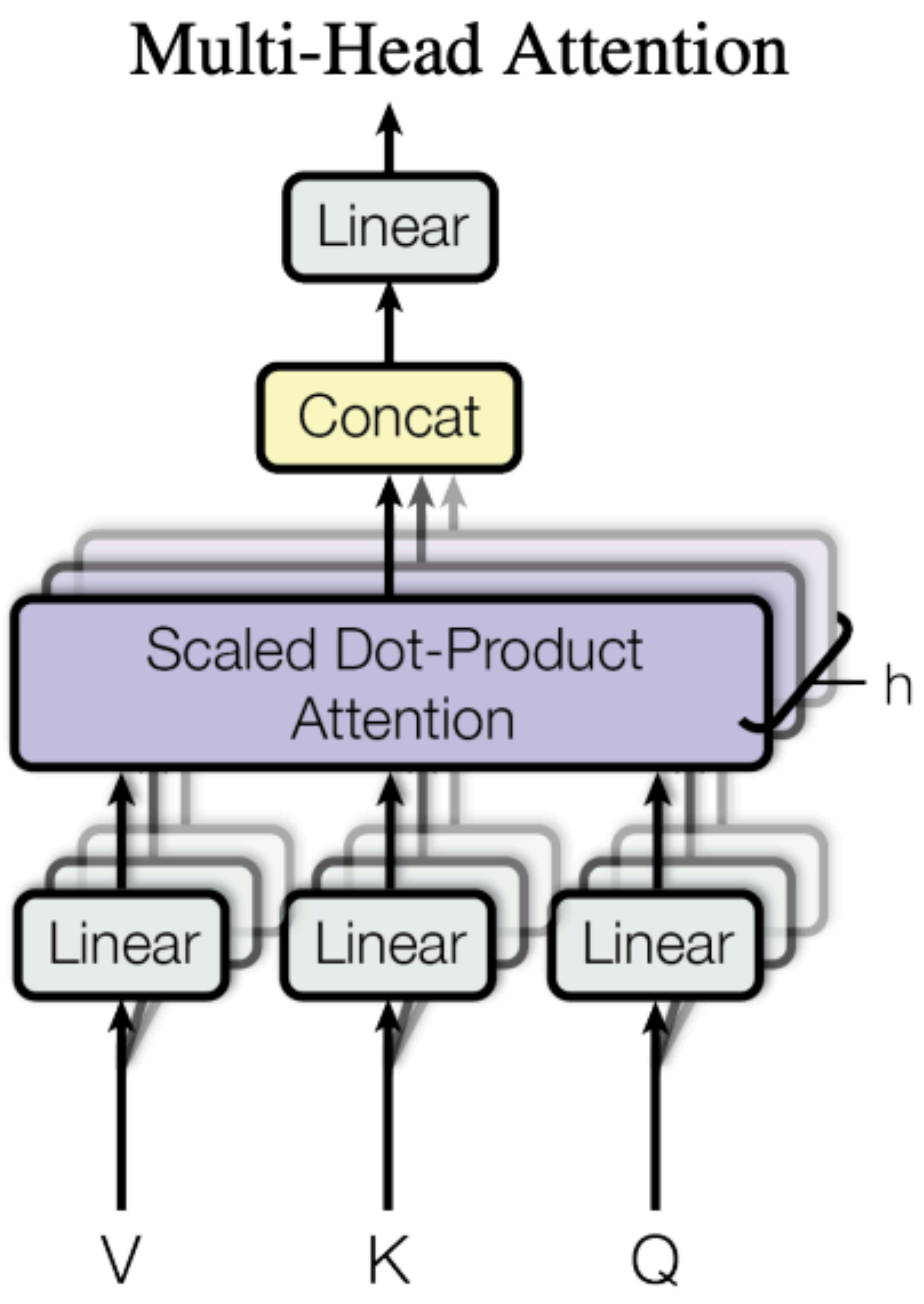
Multi-headed attention

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $\mathbf{x}_i^T\mathbf{Q}^T(\mathbf{K}\mathbf{x}_j)$ is high, but maybe we want to focus on different j for different reasons?
- Define multiple attention "heads" through multiple $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ matrices
- Let $\mathbf{Q}_l, \mathbf{K}_l, \mathbf{V}_l$, each in $\mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and $1 \leq l \leq h$.
- Each attention head performs attention independently:

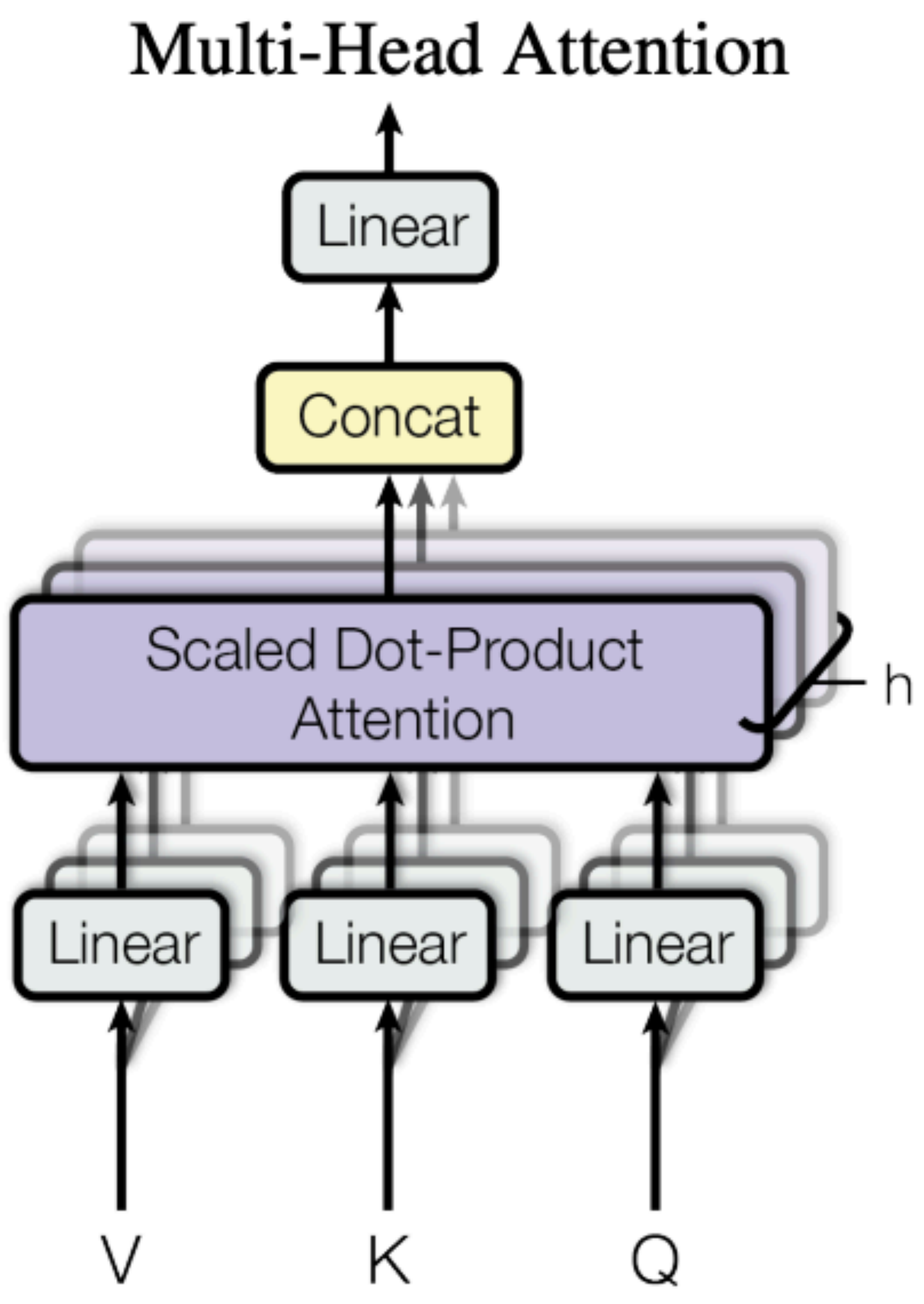


Multi-headed attention

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $\mathbf{x}_i^T\mathbf{Q}^T(\mathbf{K}\mathbf{x}_j)$ is high, but maybe we want to focus on different j for different reasons?
- Define multiple attention "heads" through multiple $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ matrices
- Let $\mathbf{Q}_l, \mathbf{K}_l, \mathbf{V}_l$, each in $\mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and $1 \leq l \leq h$.
- Each attention head performs attention independently:
- Then the outputs of all the heads are combined!



Multi-headed attention



- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $\mathbf{x}_i^T\mathbf{Q}^T(\mathbf{K}\mathbf{x}_j)$ is high, but maybe we want to focus on different j for different reasons?
- Define multiple attention "heads" through multiple $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ matrices
- Let $\mathbf{Q}_l, \mathbf{K}_l, \mathbf{V}_l$, each in $\mathbb{R}^{d\times\frac{d}{h}}$, where h is the number of attention heads, and $1 \leq l \leq h$.
- Each attention head performs attention independently:
- Then the outputs of all the heads are combined!

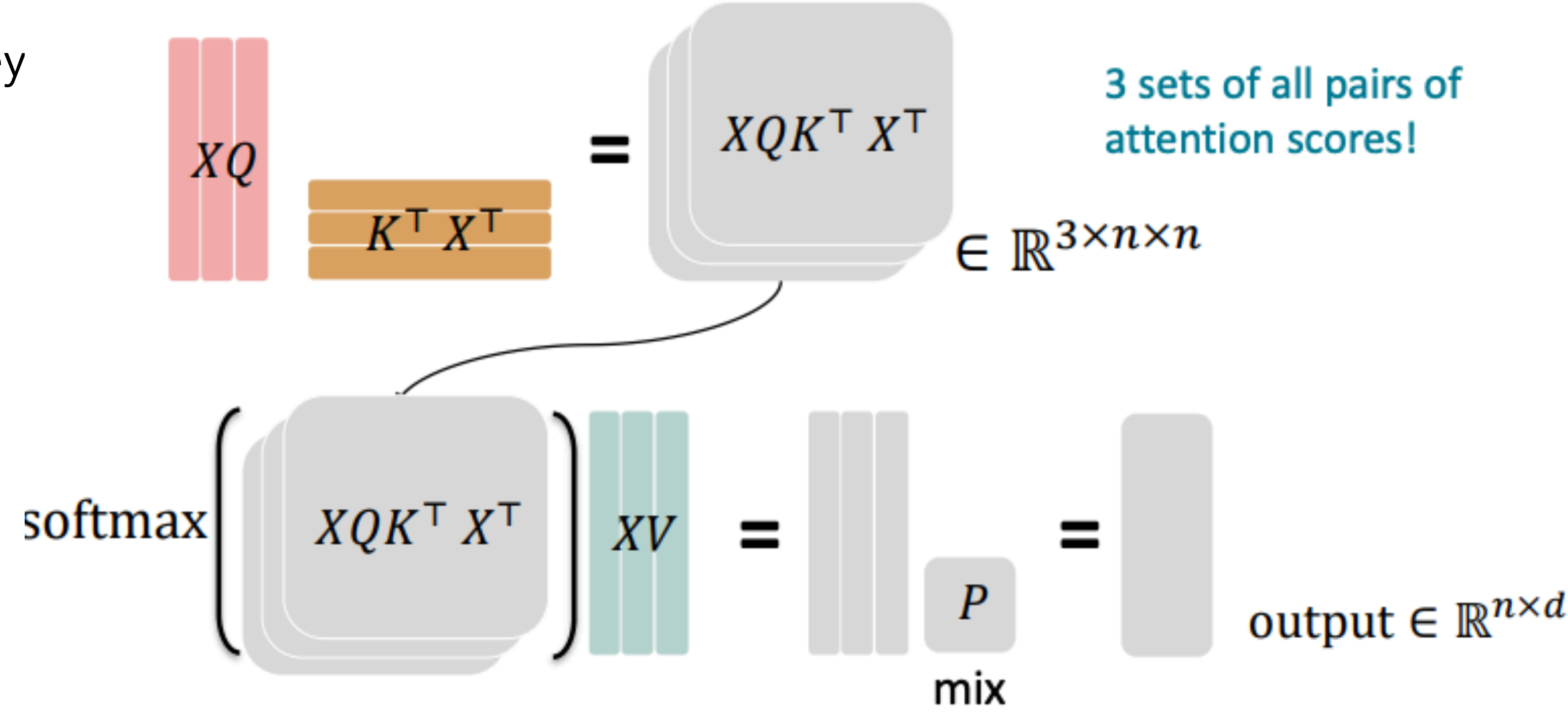
Each head gets to "look" at different things, and construct value vectors differently

Multiheaded Attention: Visualization

Still efficient, can be parallelized!

First, take the query-key dot products in one matrix multiplication: $\mathbf{XQ}_l(\mathbf{XK}_l)^T$

Next, softmax, and compute the weighted average with another matrix multiplication.

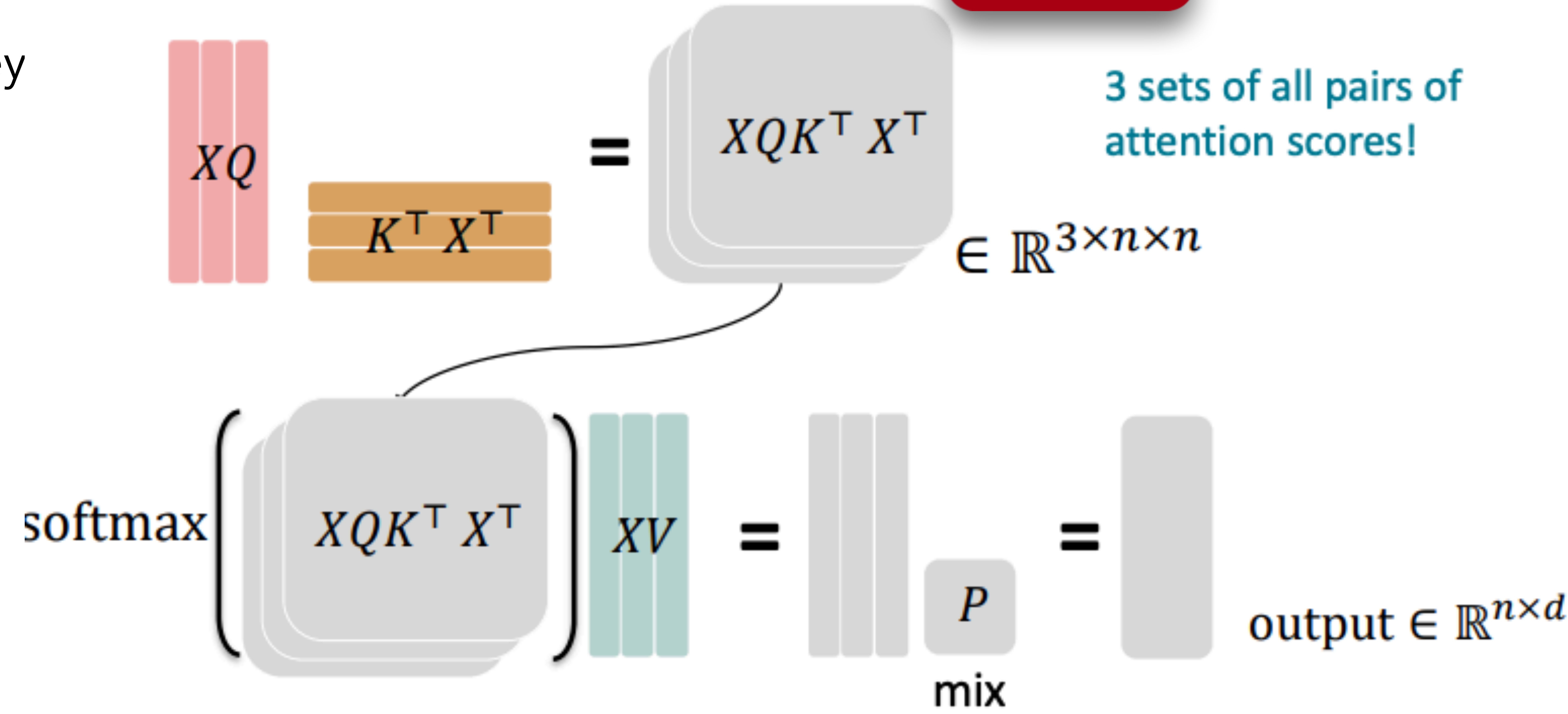


Multiheaded Attention: Visualization

Still efficient, can be parallelized!

Tensor!

First, take the query-key dot products in one matrix multiplication: $\mathbf{XQ}_l(\mathbf{XK}_l)^T$



Next, softmax, and compute the weighted average with another matrix multiplication.

Scaled Dot Product Attention

$$\textbf{output}_\ell = \textbf{softmax}(XQ_\ell K_\ell^T X^T) * XV_\ell$$

Attention is all you need (Vaswani et al., 2017)

Scaled Dot Product Attention

$$\textbf{output}_\ell = \textbf{softmax}(XQ_\ell K_\ell^T X^T) * XV_\ell$$

- So far: Dot product self-attention

Attention is all you need (Vaswani et al., 2017)

Scaled Dot Product Attention

$$\textbf{output}_\ell = \textbf{softmax}(XQ_\ell K_\ell^T X^T) * XV_\ell$$

- So far: Dot product self-attention
- When dimensionality d becomes large, dot products between vectors tend to become large

Attention is all you need (Vaswani et al., 2017)

Scaled Dot Product Attention

$$\textbf{output}_\ell = \textbf{softmax}(XQ_\ell K_\ell^T X^T) * XV_\ell$$

- So far: Dot product self-attention
- When dimensionality d becomes large, dot products between vectors tend to become large
- Because of this, inputs to the softmax function can be large, making the gradients small

Attention is all you need (Vaswani et al., 2017)

Scaled Dot Product Attention

$$\textbf{output}_\ell = \textbf{softmax}(XQ_\ell K_\ell^T X^T) * XV_\ell$$

- So far: Dot product self-attention
- When dimensionality d becomes large, dot products between vectors tend to become large
- Because of this, inputs to the softmax function can be large, making the gradients small
- Now: Scaled Dot product self-attention to aid in training

$$\textbf{output}_\ell = \textbf{softmax}\left(\frac{XQ_\ell K_\ell^T X^T}{\sqrt{d/h}}\right) * XV_\ell$$

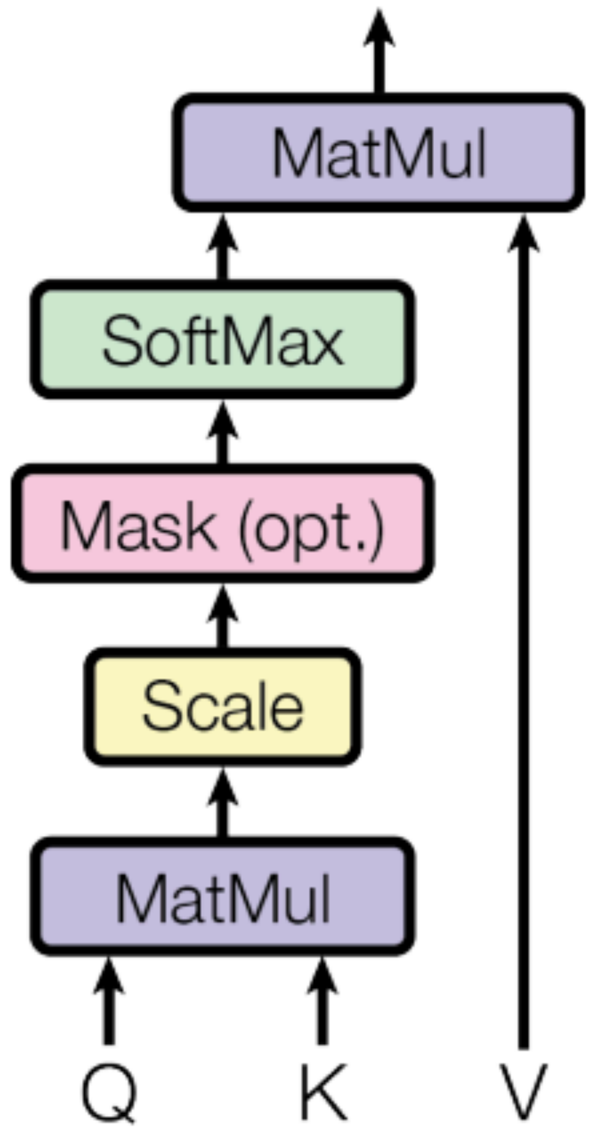
Attention is all you need (Vaswani et al., 2017)

Scaled Dot Product Attention

$$\textbf{output}_\ell = \textbf{softmax}(XQ_\ell K_\ell^T X^T) * XV_\ell$$

- So far: Dot product self-attention
- When dimensionality d becomes large, dot products between vectors tend to become large
- Because of this, inputs to the softmax function can be large, making the gradients small
- Now: Scaled Dot product self-attention to aid in training

$$\textbf{output}_\ell = \textbf{softmax}\left(\frac{XQ_\ell K_\ell^T X^T}{\sqrt{d/h}}\right) * XV_\ell$$



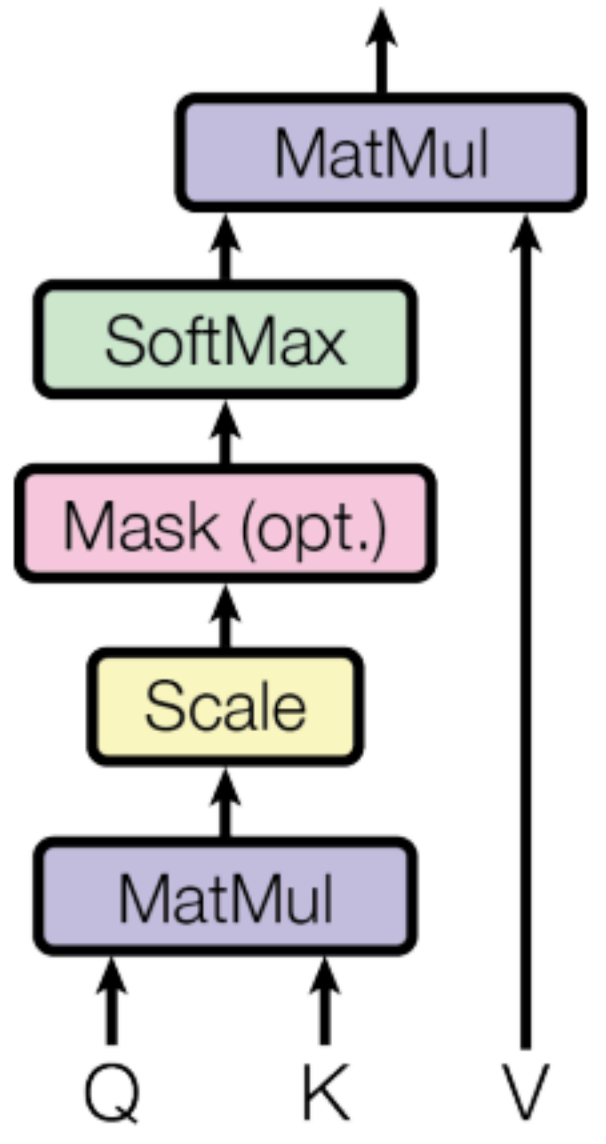
Attention is all you need (Vaswani et al., 2017)

Scaled Dot Product Attention

$$\textbf{output}_\ell = \textbf{softmax}(XQ_\ell K_\ell^T X^T) * XV_\ell$$

- So far: Dot product self-attention
- When dimensionality d becomes large, dot products between vectors tend to become large
- Because of this, inputs to the softmax function can be large, making the gradients small
- Now: Scaled Dot product self-attention to aid in training

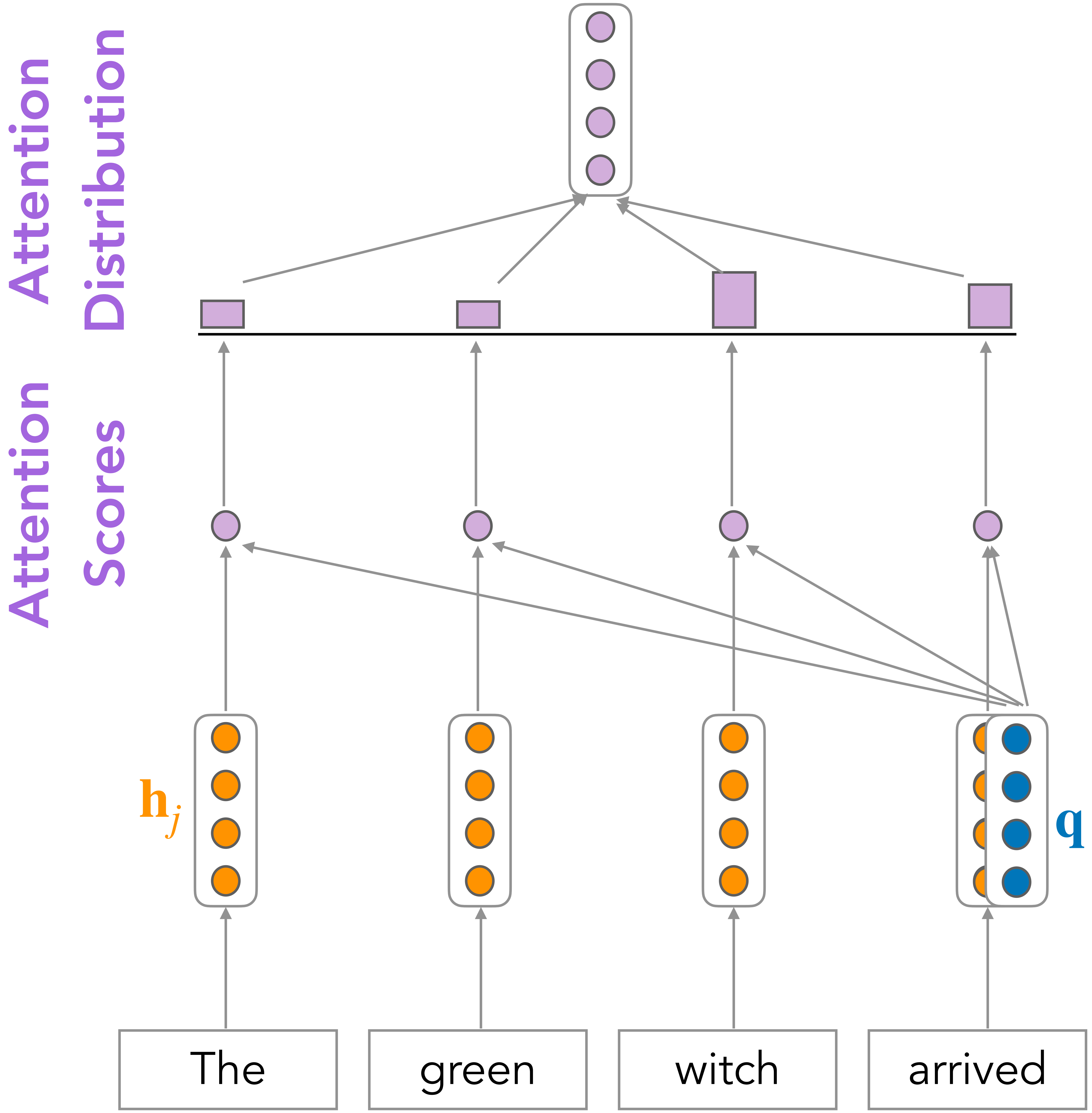
$$\textbf{output}_\ell = \textbf{softmax}\left(\frac{XQ_\ell K_\ell^T X^T}{\sqrt{d/h}}\right) * XV_\ell$$



- We divide the attention scores by $\sqrt{d/h}$, to stop the scores from becoming large just as a function of d/h, where h is the number of heads

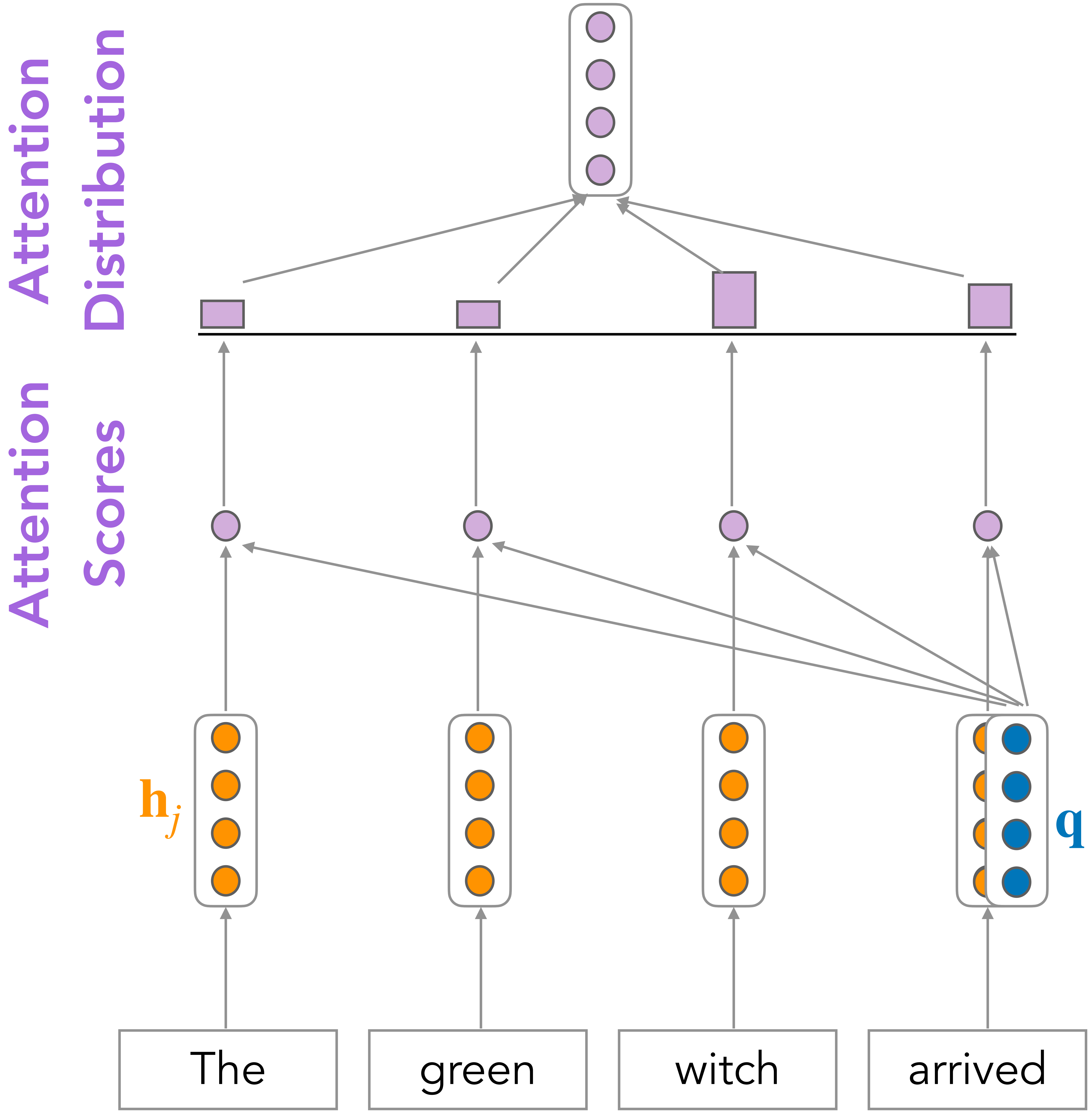
Attention is all you need (Vaswani et al., 2017)

Self-Attention: Order Information?



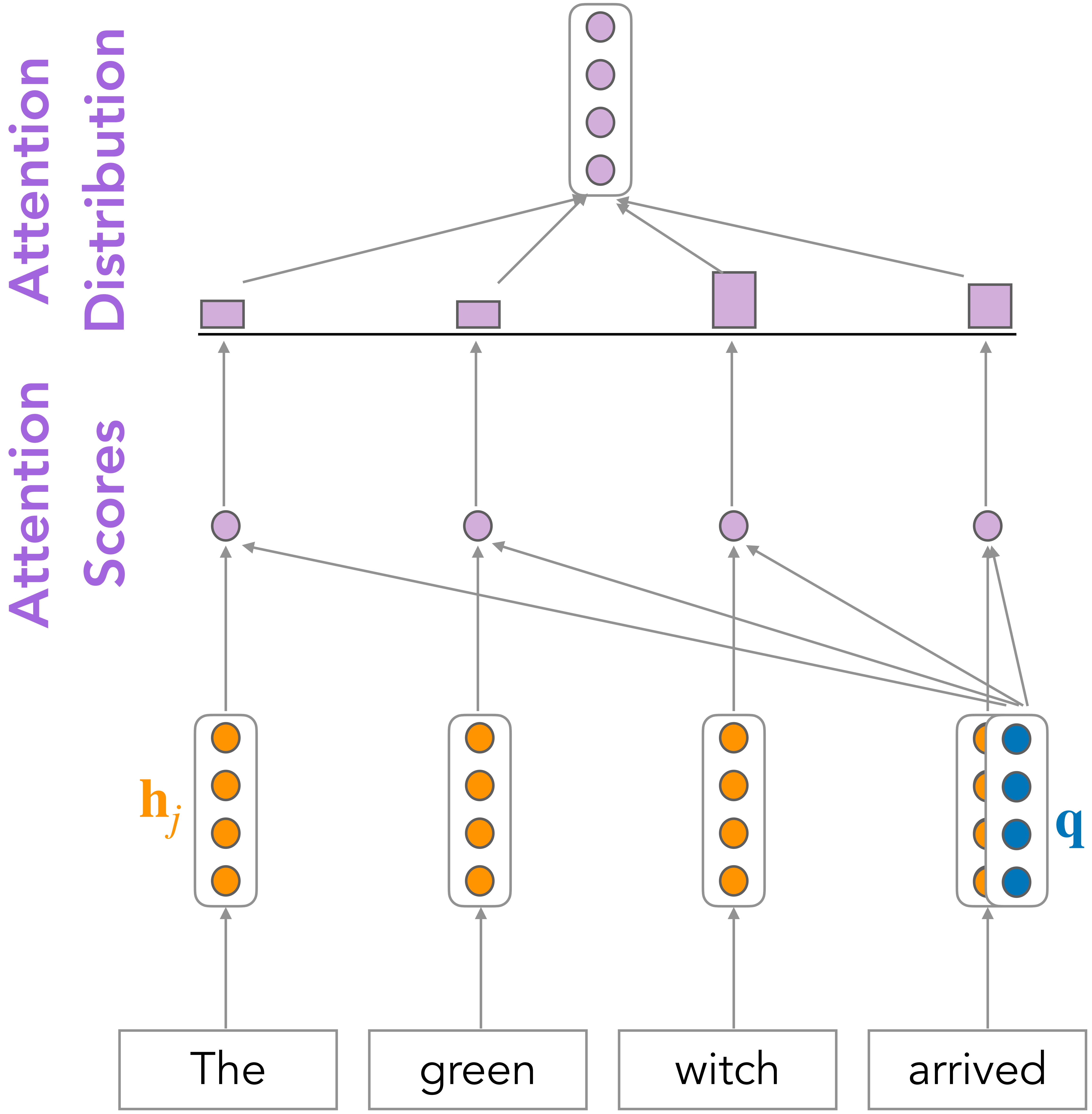
Self-Attention: Order Information?

- Self-attention networks are not necessarily (and not typically) based on Recurrent Neural Nets



Self-Attention: Order Information?

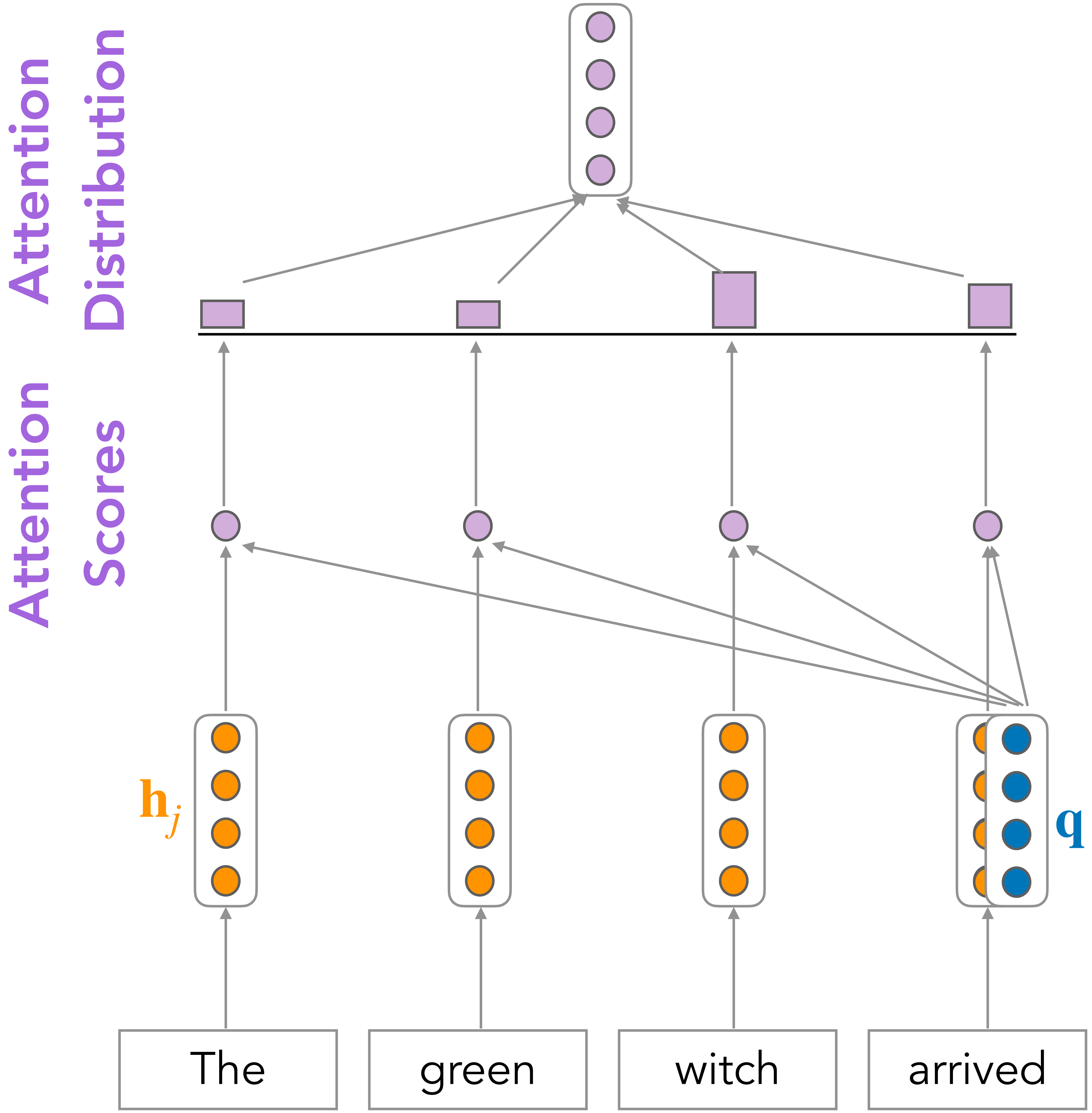
- Self-attention networks are not necessarily (and not typically) based on Recurrent Neural Nets
  - No more order information!



USC Viterbi

Self-Attention: Order Information?

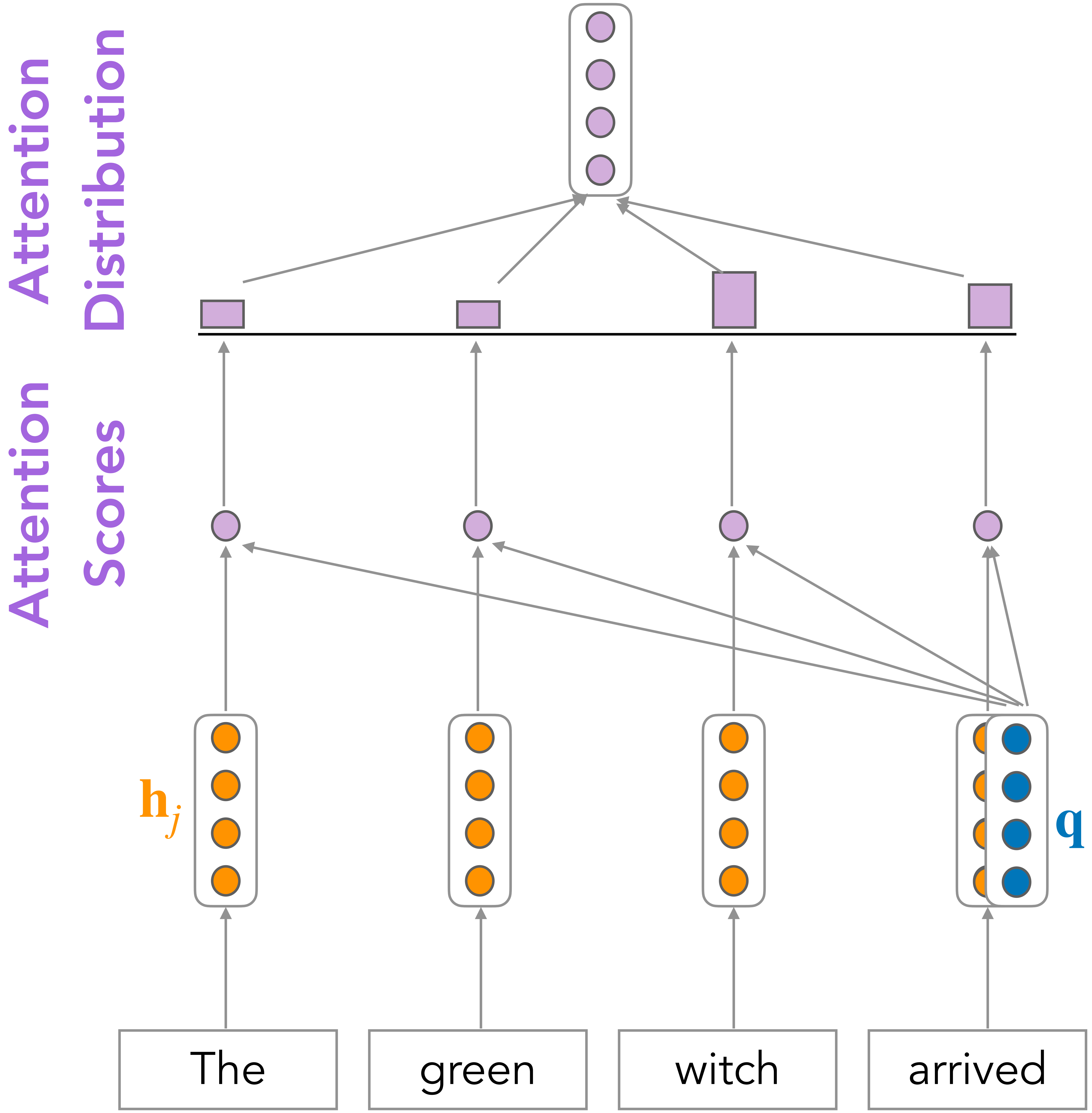
- Self-attention networks are not necessarily (and not typically) based on Recurrent Neural Nets
  - No more order information!
- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.



Self-Attention: Order Information?

- Self-attention networks are not necessarily (and not typically) based on Recurrent Neural Nets
  - No more order information!
- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.

Do feedforward nets contain order information?



Transformers: Positional Embeddings

Missing: Order Information

Missing: Order Information

- Consider representing each sequence index as a vector

Missing: Order Information

- Consider representing each sequence index as a vector
    - $\mathbf{p}_i \in \mathbb{R}^d$, for $i \in \{1,2, \ldots , n\}$ are position vectors

Missing: Order Information

- Consider representing each sequence index as a vector
    - $\mathbf{p}_i \in \mathbb{R}^d$, for $i \in \{1,2, \dots, n\}$ are position vectors
- Don't worry about what the $\mathbf{p}_i$ are made of yet!

Missing: Order Information

- Consider representing each sequence index as a vector
  - $\mathbf{p}_i \in \mathbb{R}^d$, for $i \in \{1,2, \ldots , n\}$ are position vectors
- Don't worry about what the $\mathbf{p}_i$ are made of yet!
- Easy to incorporate this info: just add the $\mathbf{p}_i$ to our inputs!

Missing: Order Information

- Consider representing each sequence index as a vector
  - $\mathbf{p}_i \in \mathbb{R}^d$, for $i \in \{1,2, \dots, n\}$ are position vectors
- Don't worry about what the $\mathbf{p}_i$ are made of yet!
- Easy to incorporate this info: just add the $\mathbf{p}_i$ to our inputs!
- Recall that $\mathbf{x}_i$ is the embedding of the word at index i. The positioned embedding is:

Missing: Order Information

- Consider representing each sequence index as a vector
    - $\mathbf{p}_i \in \mathbb{R}^d$, for $i \in \{1,2, \dots, n\}$ are position vectors
- Don't worry about what the $\mathbf{p}_i$ are made of yet!
- Easy to incorporate this info: just add the $\mathbf{p}_i$ to our inputs!
- Recall that $\mathbf{x}_i$ is the embedding of the word at index i. The positioned embedding is:
    - $\tilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{p}_i$

Missing: Order Information

- Consider representing each sequence index as a vector
  - $\mathbf{p}_i \in \mathbb{R}^d$, for $i \in \{1,2, \dots, n\}$ are position vectors
- Don't worry about what the $\mathbf{p}_i$ are made of yet!
- Easy to incorporate this info: just add the $\mathbf{p}_i$ to our inputs!
- Recall that $\mathbf{x}_i$ is the embedding of the word at index i. The positioned embedding is:
  - $\tilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{p}_i$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add…

Positional Embeddings

Positional Embeddings

- Maps integer inputs (for positions) to real-valued vectors
  - one per position in the entire context

Positional Embeddings

- Maps integer inputs (for positions) to real-valued vectors
  - one per position in the entire context
- Can be randomly initialized and can let all $\mathbf{p}_i$ be learnable parameters (most common)

Positional Embeddings

- Maps integer inputs (for positions) to real-valued vectors
    - one per position in the entire context
- Can be randomly initialized and can let all $\mathbf{p}_i$ be learnable parameters (most common)
- Pros:
    - Flexibility: each position gets to be learned to fit the data

Positional Embeddings

- Maps integer inputs (for positions) to real-valued vectors
    - one per position in the entire context
- Can be randomly initialized and can let all $\mathbf{p}_i$ be learnable parameters (most common)
- Pros:
    - Flexibility: each position gets to be learned to fit the data
- Cons:
    - Definitely can't extrapolate to indices outside $1, \ldots, n$, where n is the maximum length of the sequence allowed under the architecture
    - There will be plenty of training examples for the initial positions in our inputs and correspondingly fewer at the outer length limits

Putting it all together: Transformer Blocks

Self-Attention Transformer Building Block

- Self-attention:
  - the basis of the method; with multiple heads
- Position representations:
  - Specify the sequence order, since self-attention is an unordered function of its inputs.
- Nonlinearities:
  - At the output of the self-attention block
  - Frequently implemented as a simple feedforward network.
- Masking:
  - In order to parallelize operations while not looking at the future.
  - Keeps information about the future from "leaking" to the past.