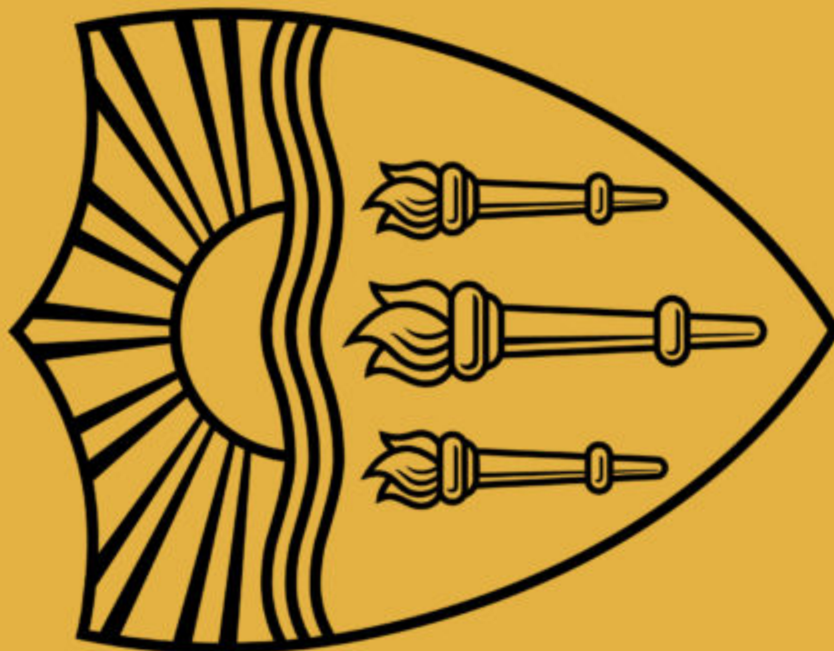
Lecture 11: Transformers: Building Blocks

Instructor: Swabha Swayamdipta

USC CSCI 499 LMs in NLP

Feb 26, Spring 2024



Slides adapted from Dan Jurafsky, Chris Manning, John Hewitt, Anna Goldie

Logistics / Announcements

- Today:
  - HW2
  - Quiz 3 in class
- This Wednesday:
  - In-class project discussions
  - Make sure the entire team is present!
- Upcoming guest / TA lectures
- No Office Hours in the Spring Break week

Date	Topic	Notes
Feb 26:	**Transformers - Building Blocks I**	HW2 Due
Feb 28:	PROJECT DISCUSSIONS	
Mar 4:	**Transformers - Building Blocks II**	HW3 Released PROGRESS REPORT DUE
Mar 6:	TA Lecture: PyTorch for Transformers	
~~Mar 11~~:	No Class *SPRING BREAK*	
~~Mar 13~~:	No Class *SPRING BREAK*	

Large Language Models (LLMs)

Date	Topic	Notes
Mar 18:	**Pre-training Transformers I**	HW3 Due
Mar 20:	**Pre-training Transformers II**	HW4 Released
Mar 25:	Guest Lecture: Limitations and Harms of LLMs	
Mar 27:	Generating from Language Models I	

Lecture Outline

- Quiz 3: FFNNs and RNNs
- Recap: Seq2Seq and Attention
- More on Attention
- Transformers: Self-Attention
- Transformers: Multi-headed Attention
- Transformers: Positional Embeddings
- Putting it all together: Transformer Blocks

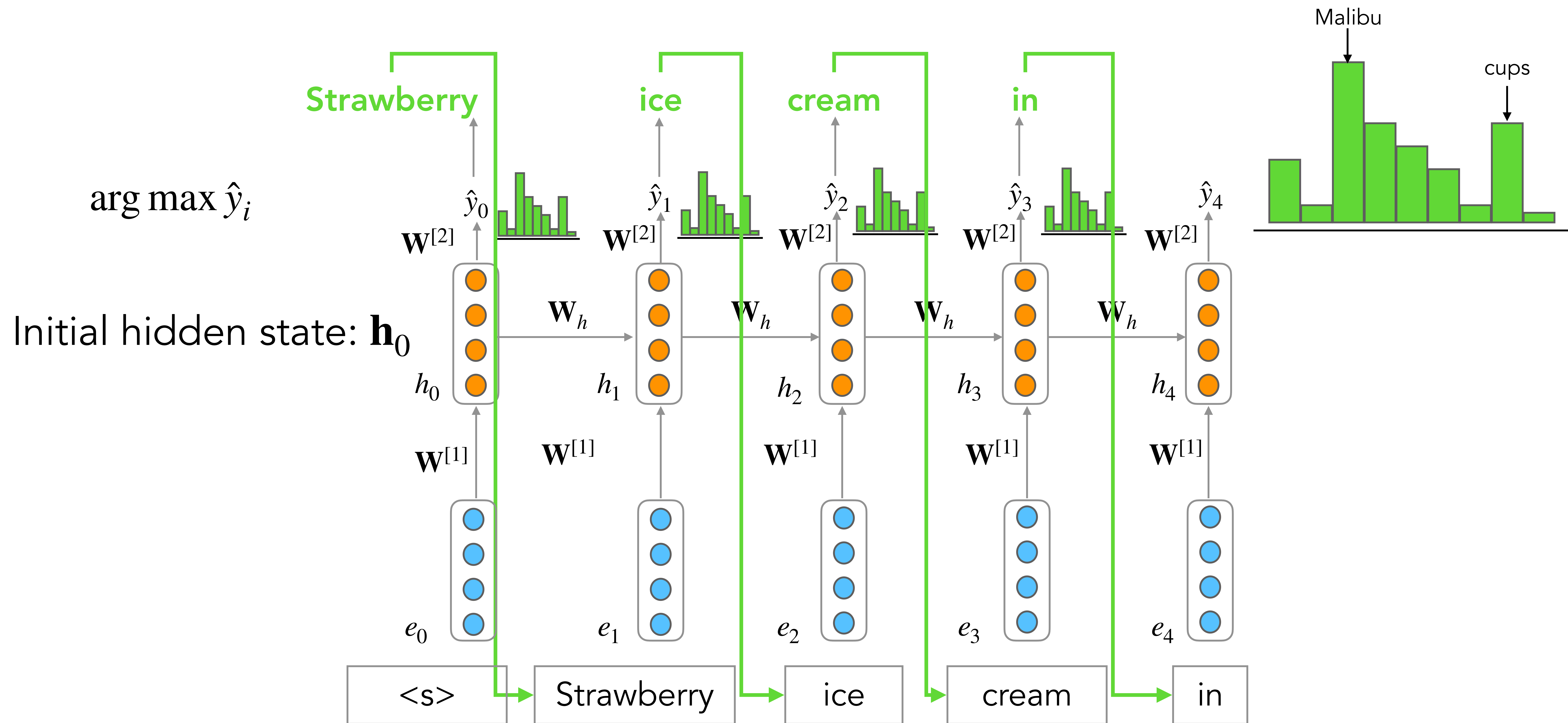
Quiz 3!

Recap:
Seq2Seq and Attention

USC Viterbi

Generation with RNNLMs

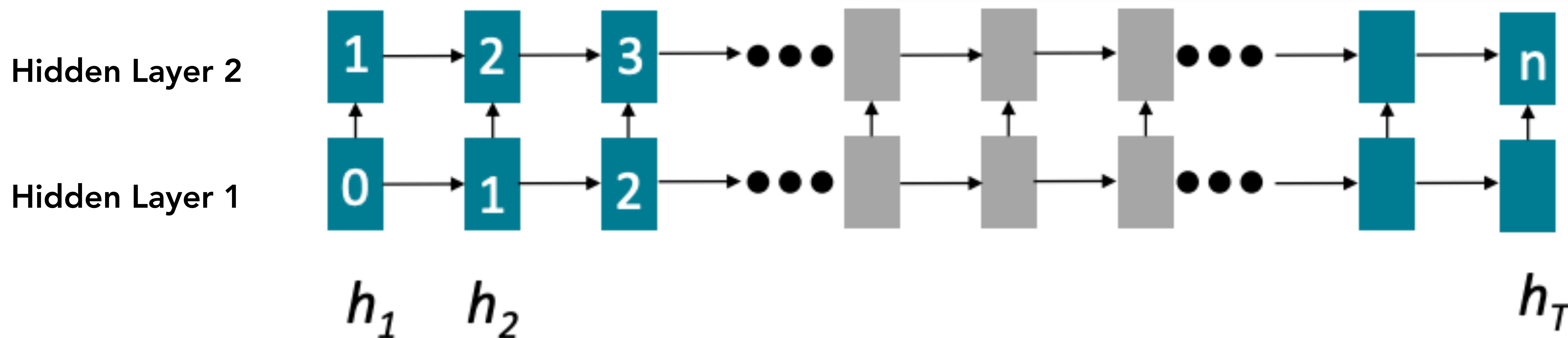
$\hat{y}_4 = P(x_5 \mid \text{Strawberry ice cream in})$



RNNs and parallelizability

- Forward and backward passes have **O(sequence length)** unparallelizable operations!
  - Future RNN hidden states can't be computed in full before past RNN hidden states have been computed
  - But GPUs can perform a bunch of independent computations at once!

Inhibits training on very large datasets!

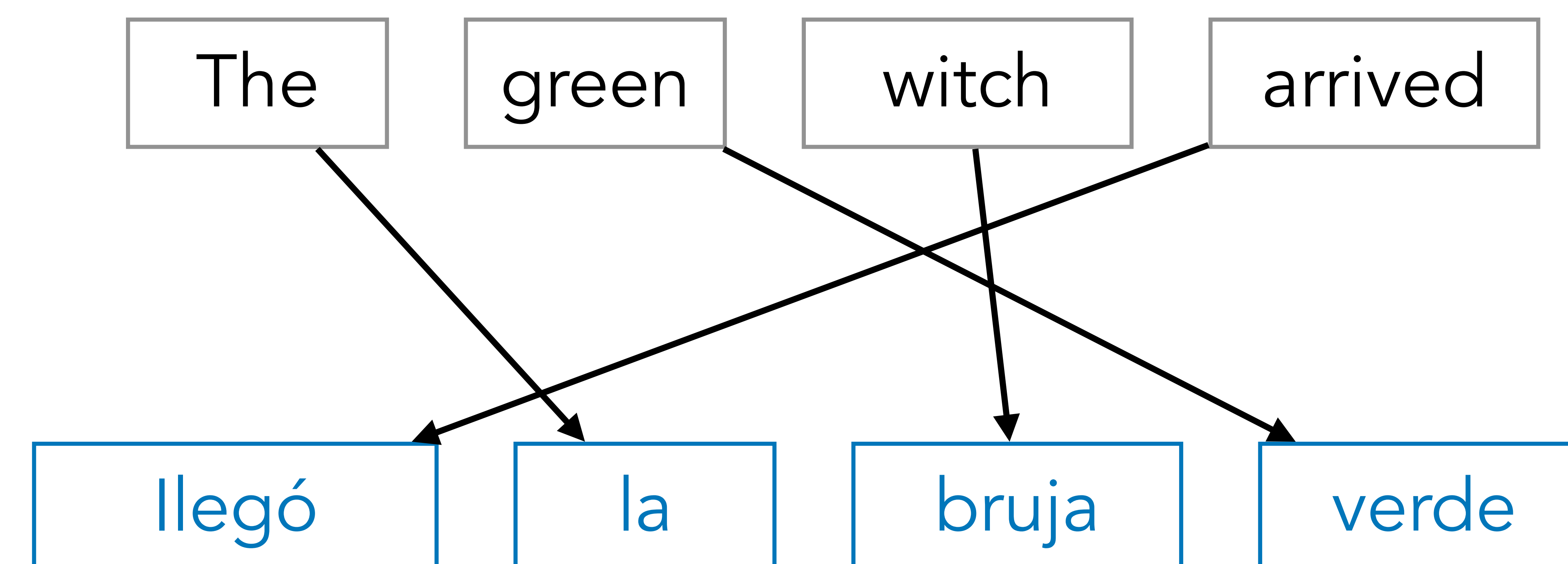


Numbers indicate min # of steps before a state can be computed

(Neural) Machine Translation

Seq2Seq uses 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)

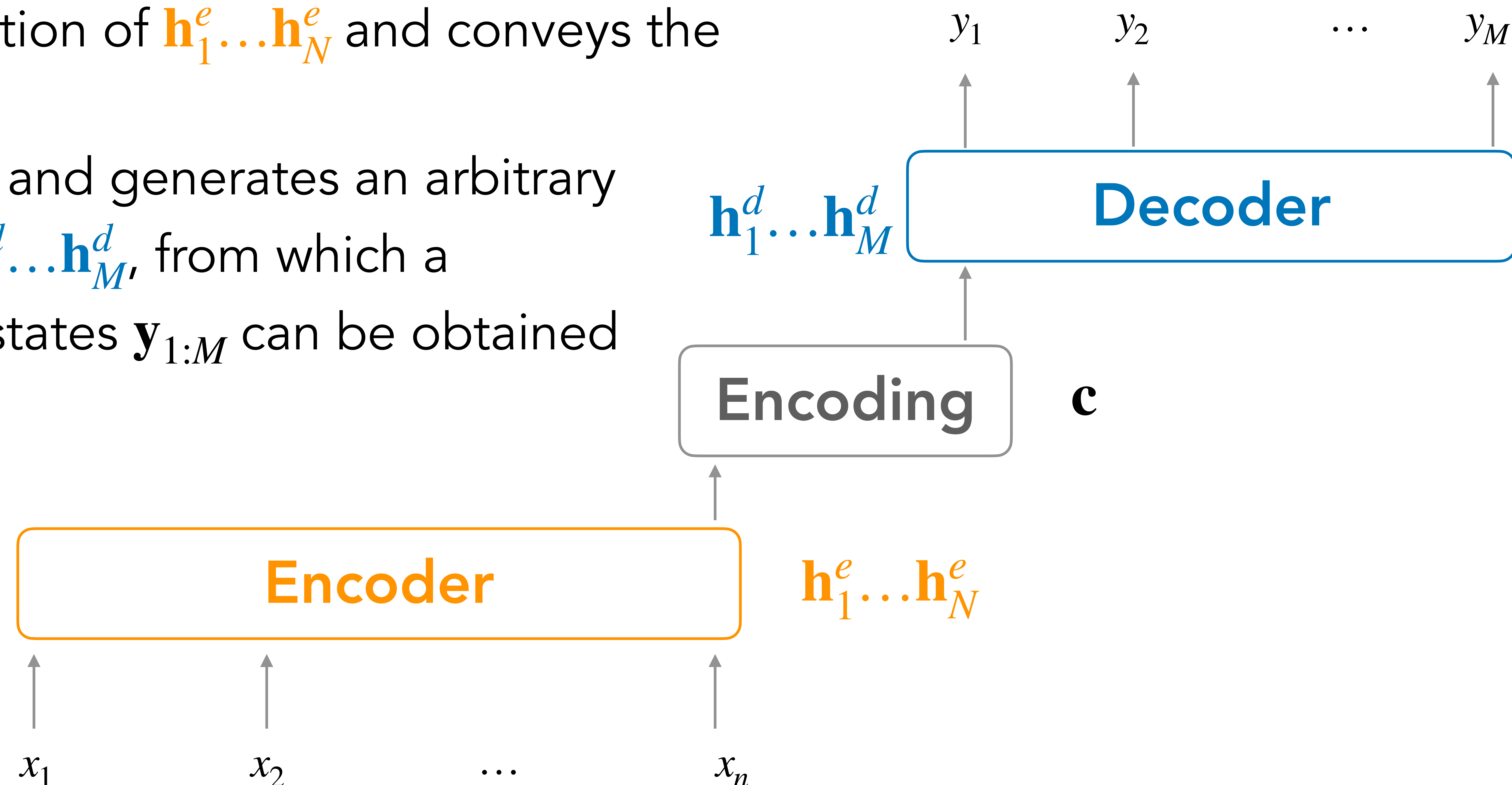


Sequence-to-Sequence (Seq2seq)

Encoder-Decoder Networks

Encoder-decoder networks consist of three components:

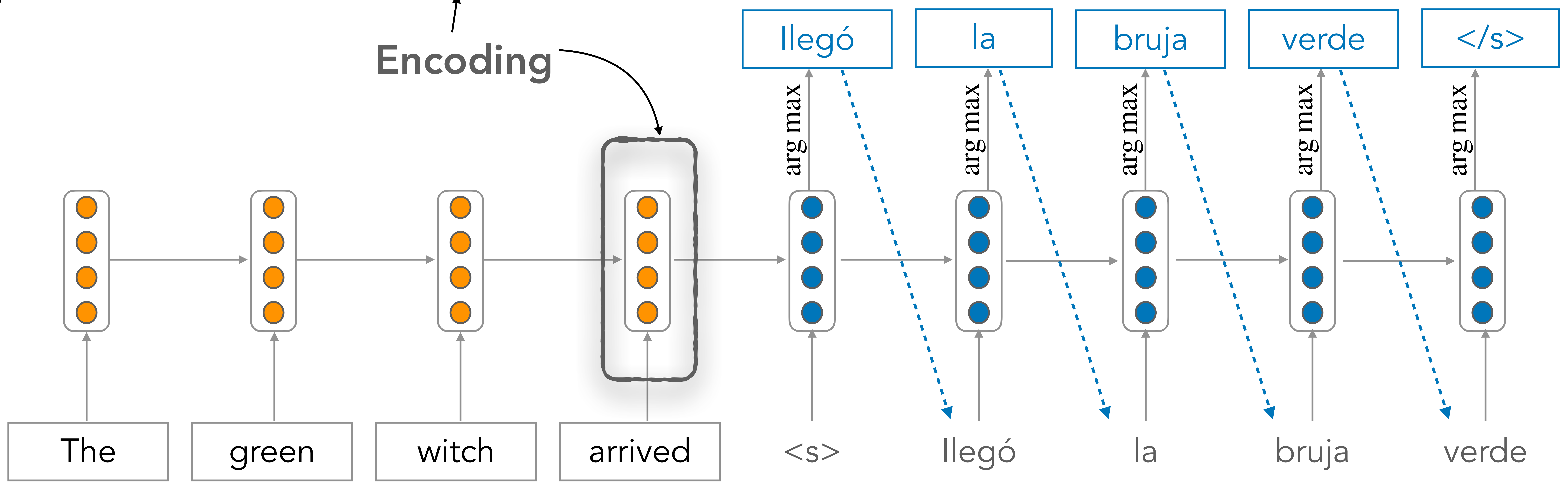
1. An **encoder** that accepts an input sequence, $\mathbf{x}_{1:N}$ and generates a corresponding sequence of contextualized representations, $\mathbf{h}^e_1 \ldots \mathbf{h}^e_N$
2. A **encoding** vector, $\mathbf{c}$ which is a function of $\mathbf{h}^e_1 \ldots \mathbf{h}^e_N$ and conveys the essence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



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

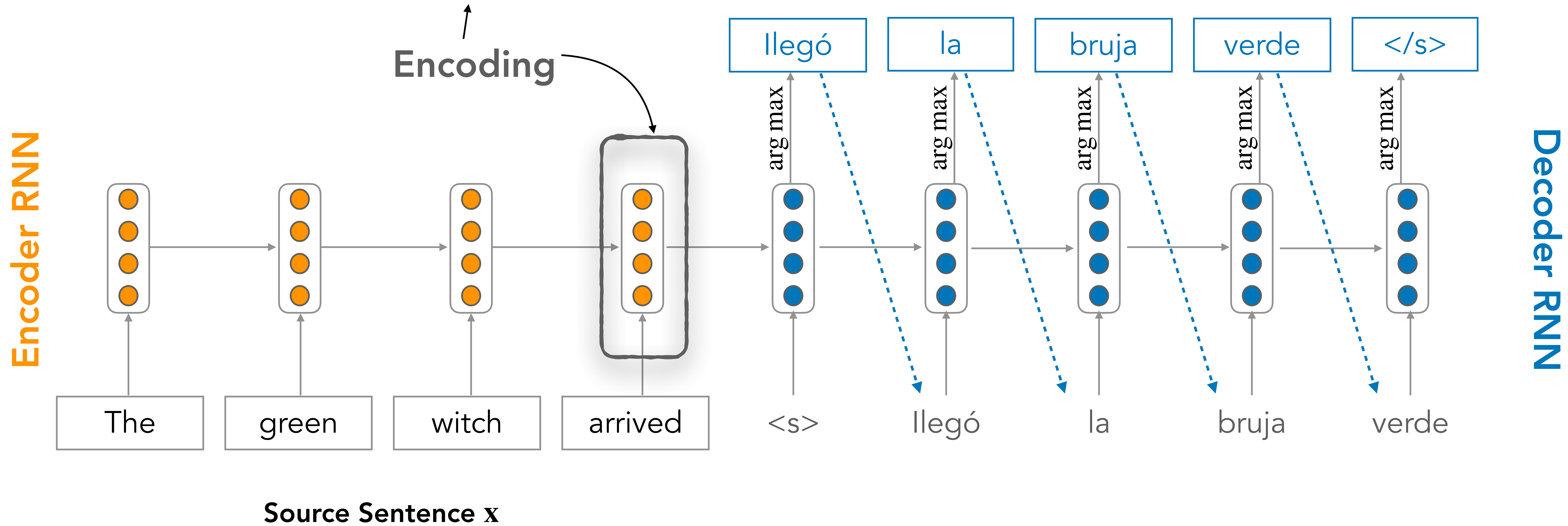
USC Viterbi

Produces an encoding of the source sequence



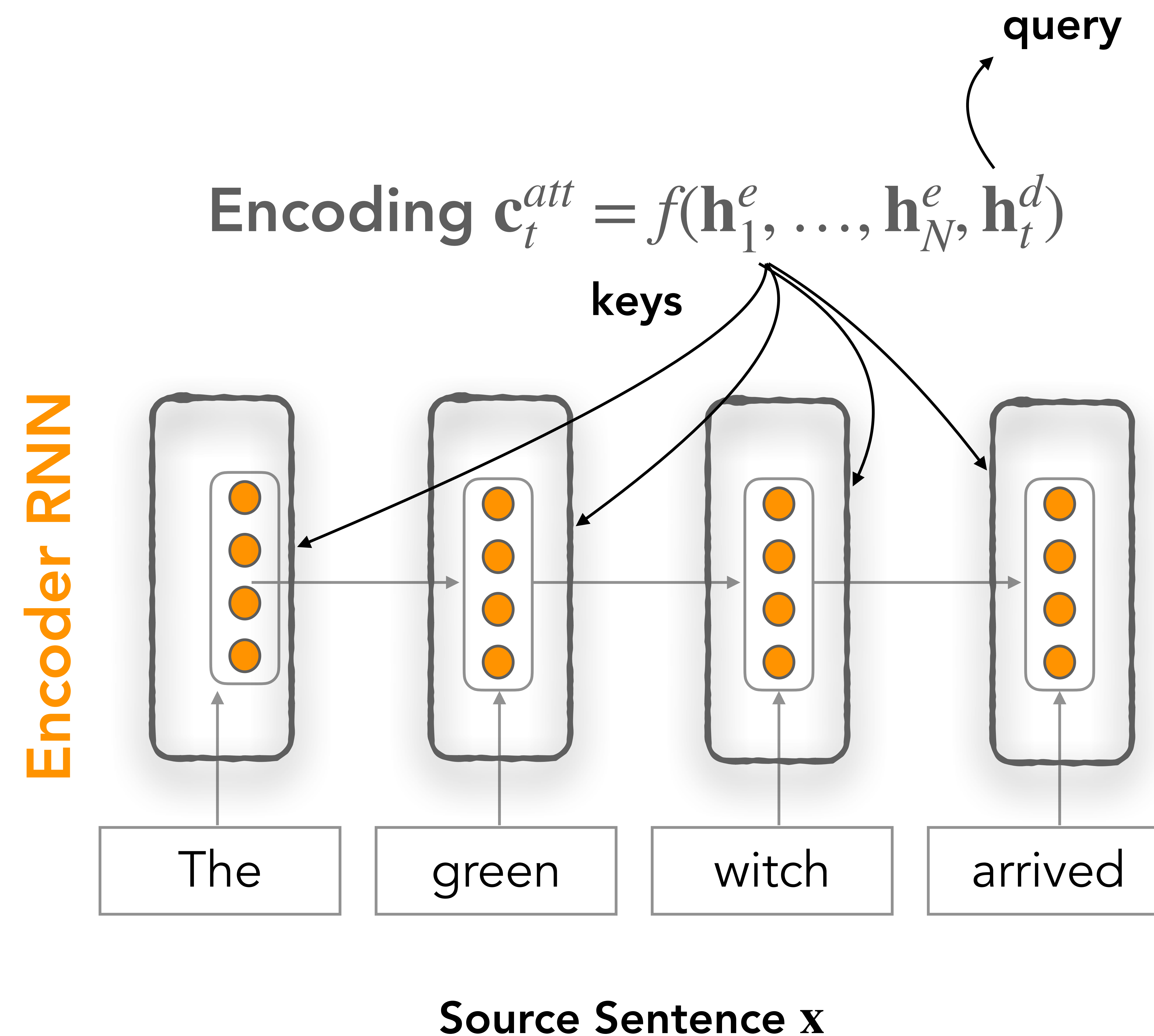
Source Sentence x

Language Model that produces the target sentence conditioned on the encoding


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

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



Note: Notation different from J&M

Bahdanau et al., 2015

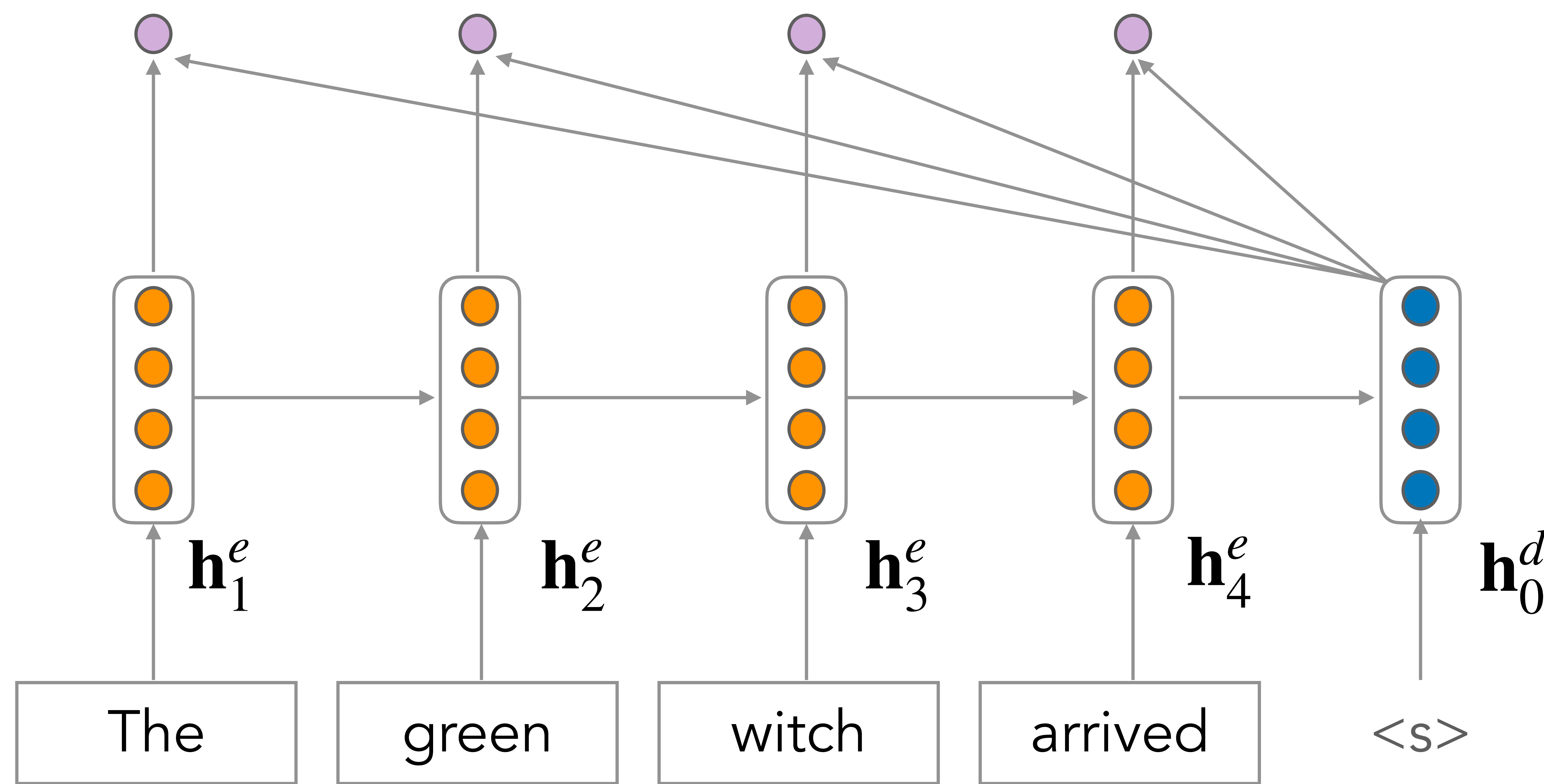
Seq2Seq with Attention

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…

Encoder RNN

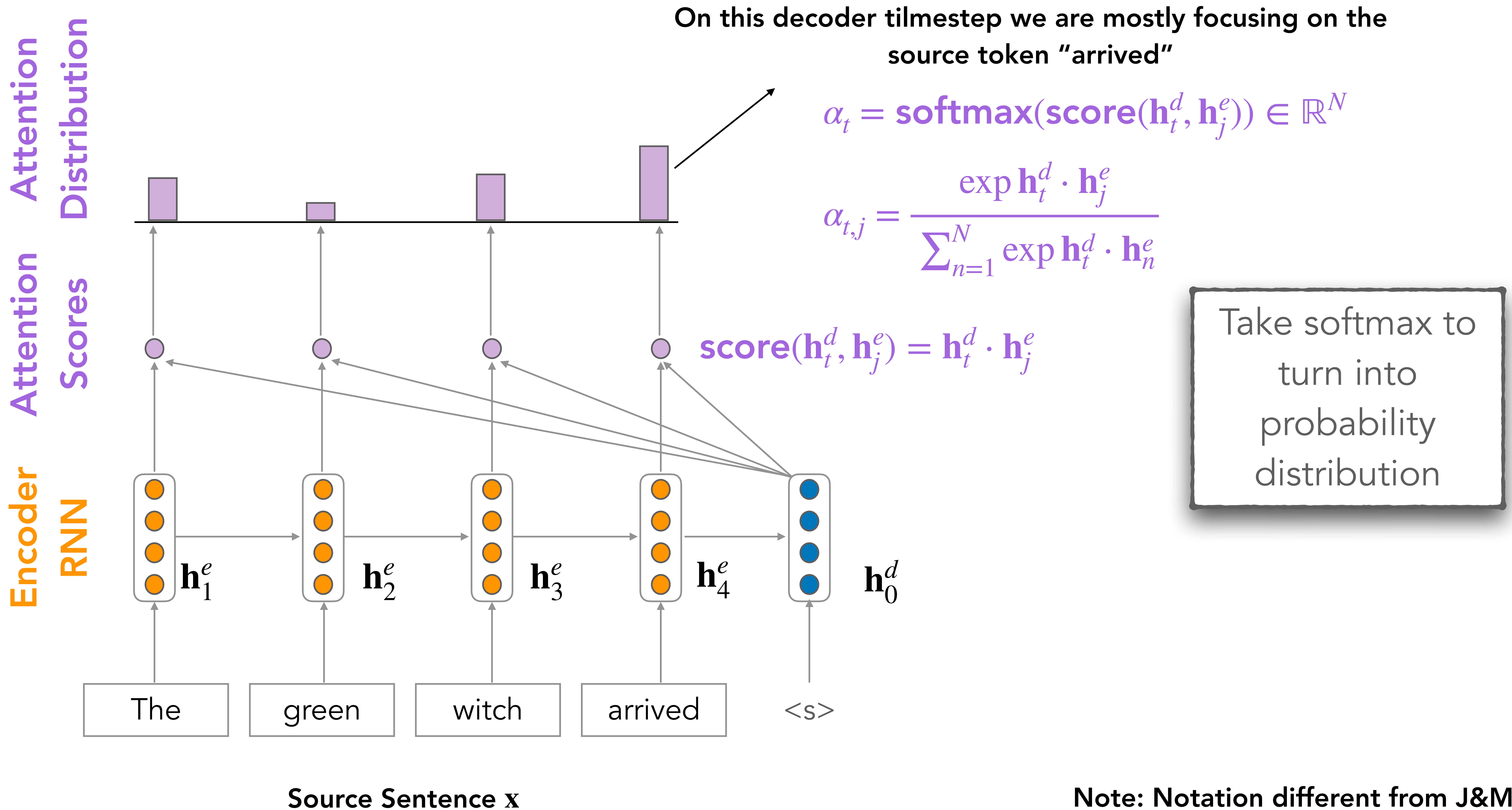


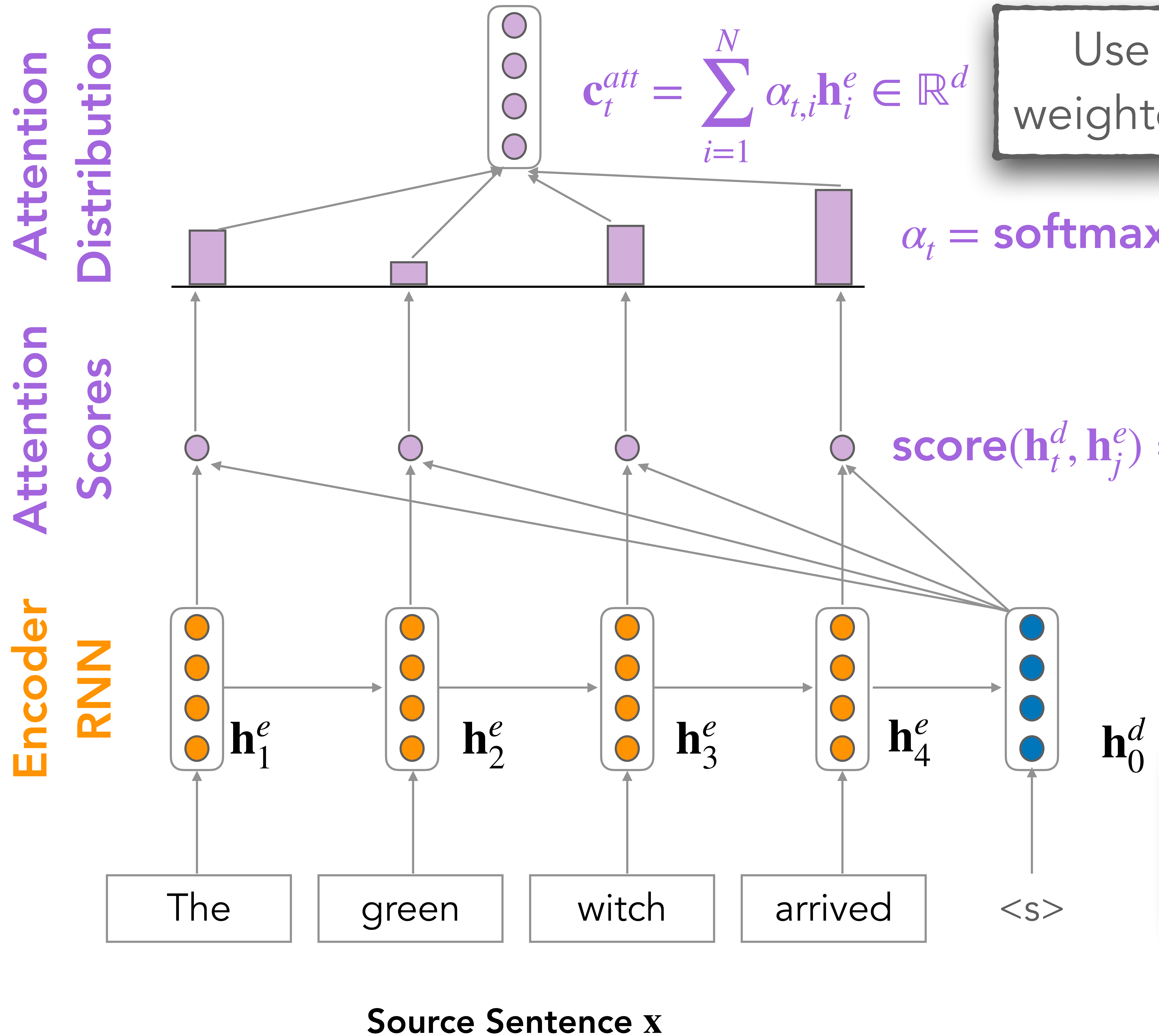
Query 1: Decoder, first time step

Source Sentence $\mathbf{x}$

Dot product attention

Note: Notation different from J&M


USC Viterbi
On this decoder tilmestep we are mostly focusing on the source token "arrived"
$\alpha_t = \mathbf{softmax}(\mathbf{score}(\mathbf{h}_t^d, \mathbf{h}_j^e)) \in \mathbb{R}^N$
$\alpha_{t,j} = \frac{\exp \mathbf{h}_t^d \cdot \mathbf{h}_j^e}{\sum_{n=1}^N \exp \mathbf{h}_t^d \cdot \mathbf{h}_n^e}$
Take softmax to turn into probability distribution
$\mathbf{score}(\mathbf{h}_t^d, \mathbf{h}_j^e) = \mathbf{h}_t^d \cdot \mathbf{h}_j^e$
Attention Distribution
Attention Scores
Encoder RNN
$\mathbf{h}_1^e$ $\mathbf{h}_2^e$ $\mathbf{h}_3^e$ $\mathbf{h}_4^e$ $\mathbf{h}_0^d$
The
green
witch
arrived
<s>
Source Sentence $\mathbf{x}$
Note: Notation different from J&M



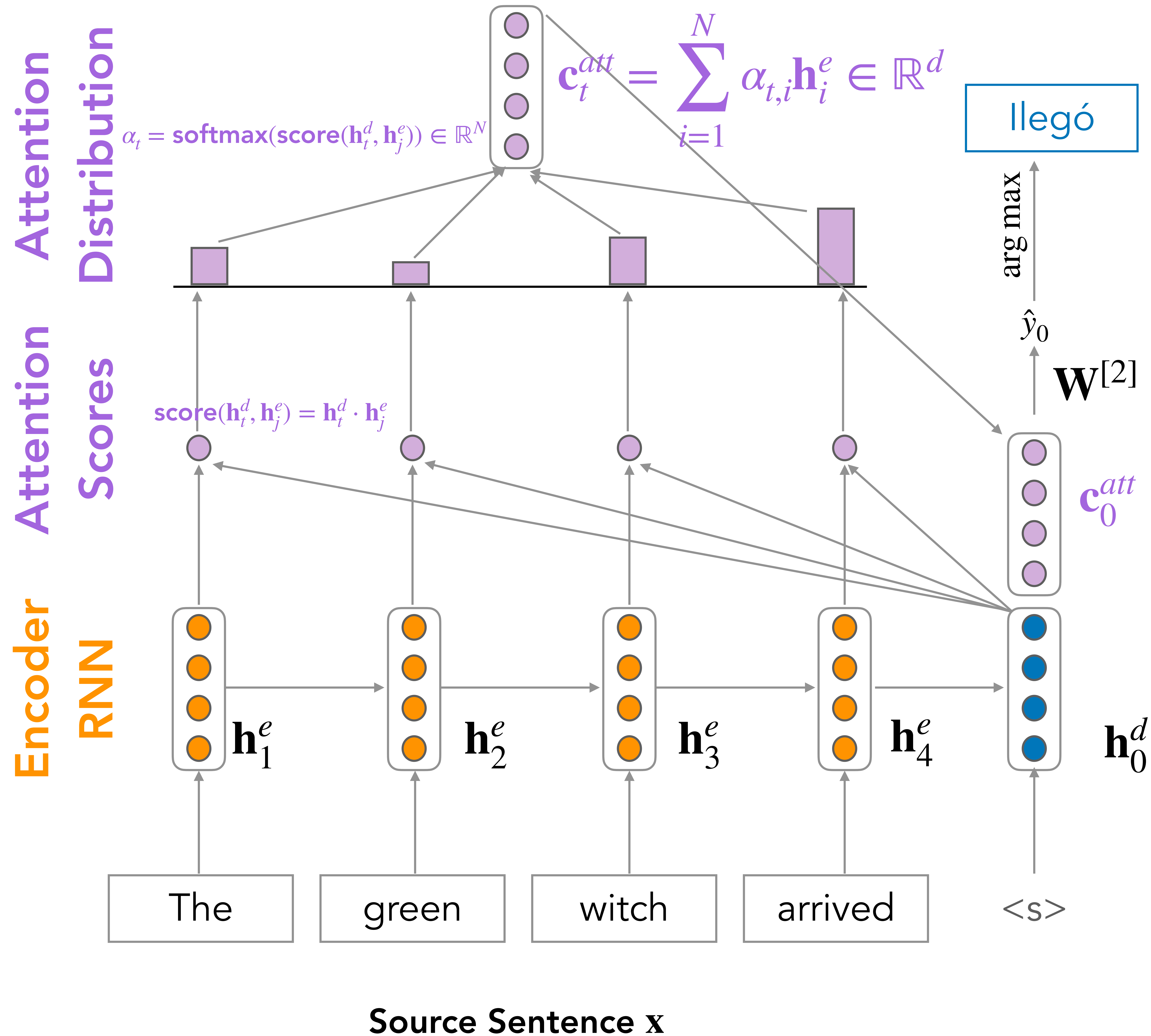
Use the attention distribution to take a weighted sum of the encoder hidden states.

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

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

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

Note: Notation different from J&M



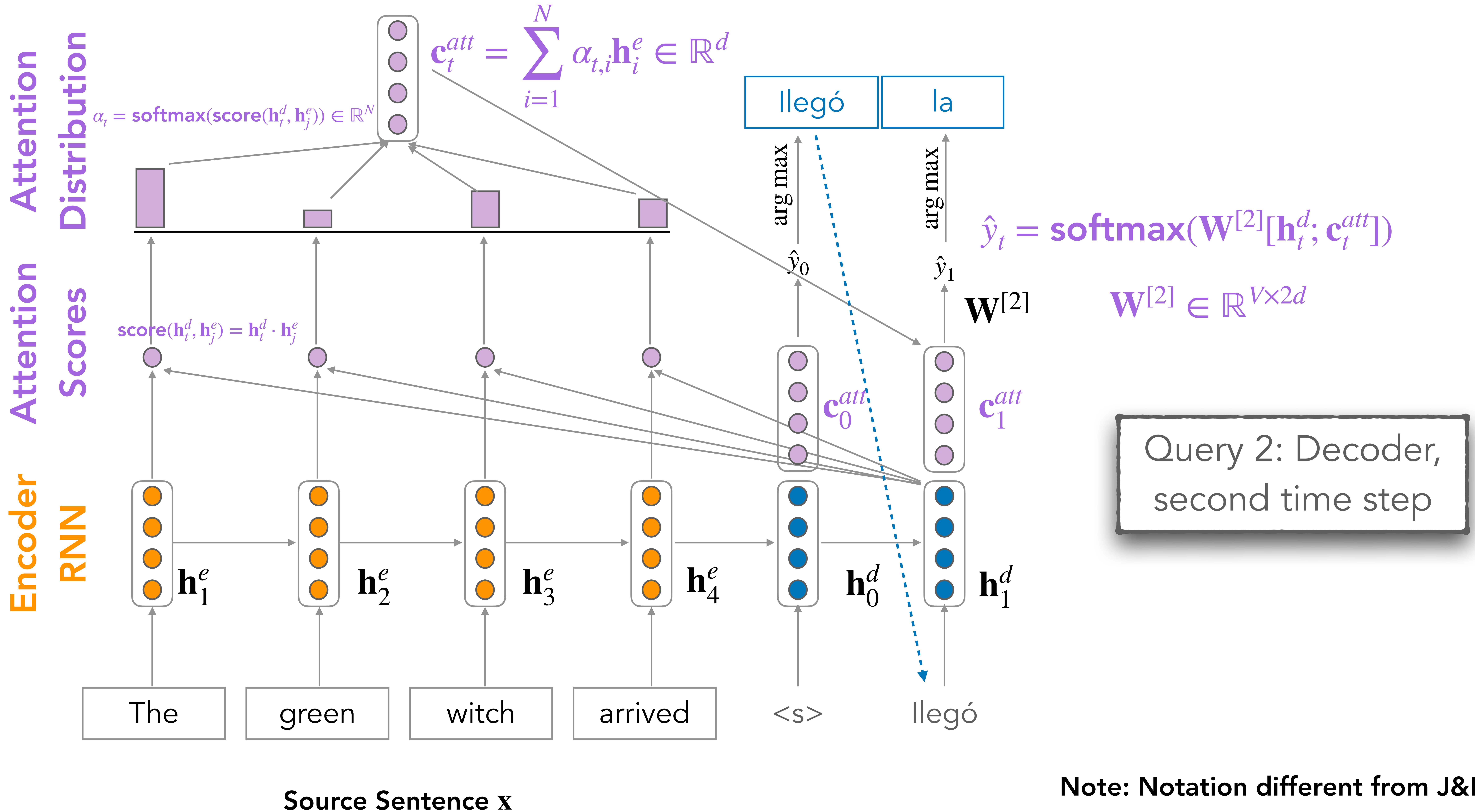
$$\hat{y}_t = \text{softmax}(W^{[2]}[h_t^d; c_t^{att}])$$

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

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

Source Sentence x

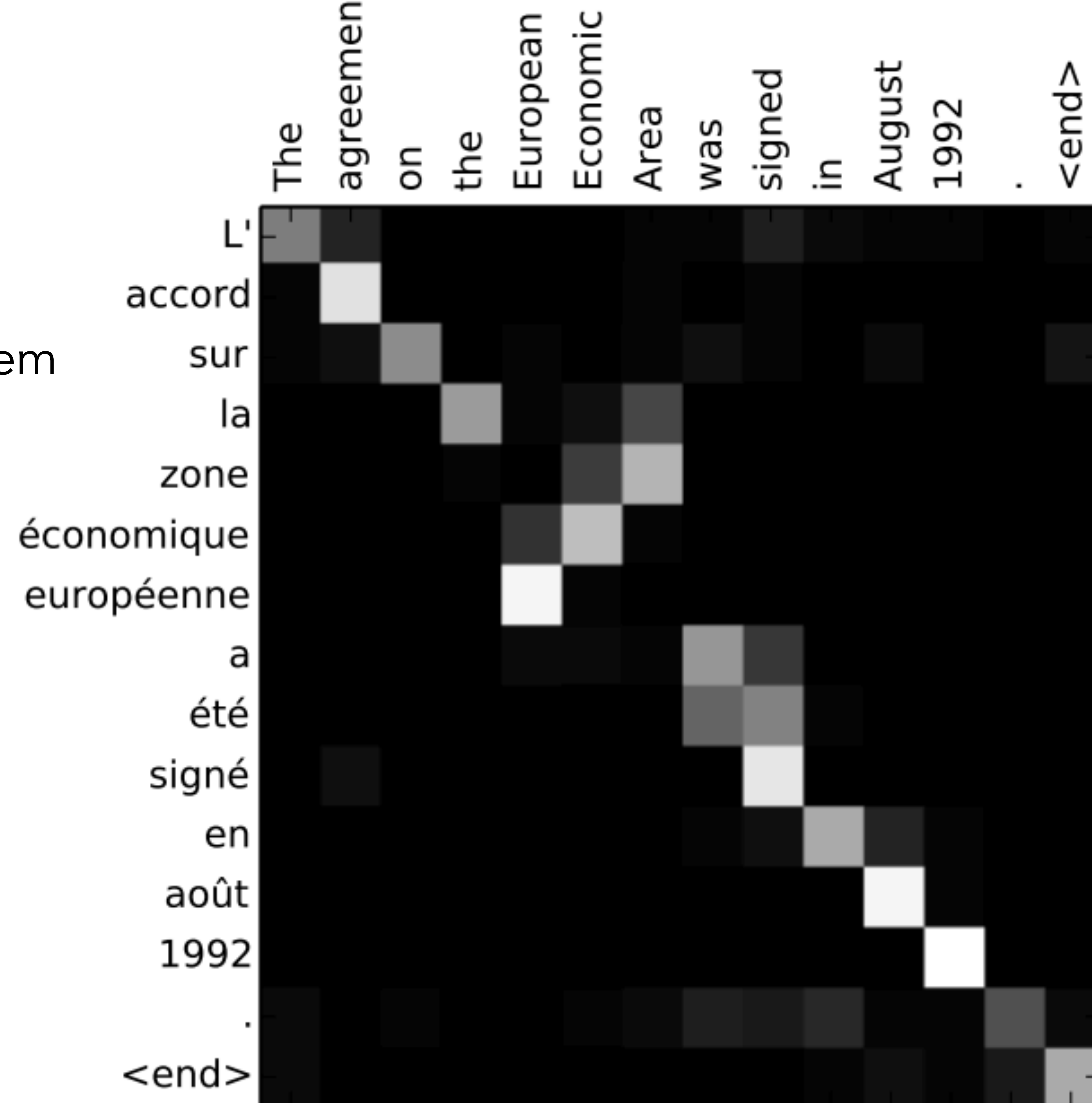
Note: Notation different from J&M



Note: Notation different from J&M

Why Attention?

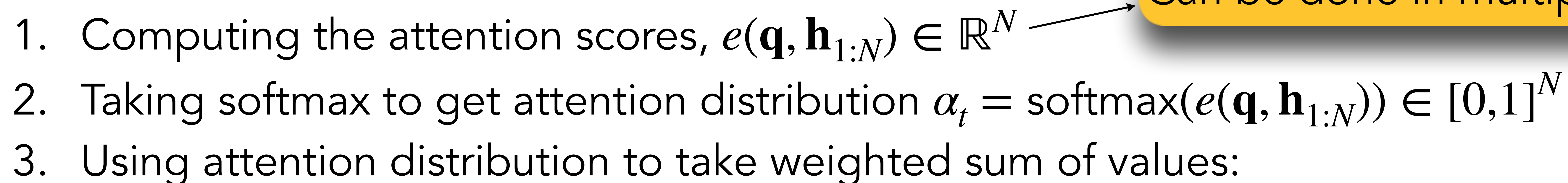
- Attention significantly **improves** neural machine translation **performance**
  - Very useful to allow decoder to focus on certain parts of the source
- Attention **solves the information bottleneck** problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention **helps 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



More on Attention

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:



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

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

USC Viterbi

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 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.
    - Also called “bilinear attention”

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

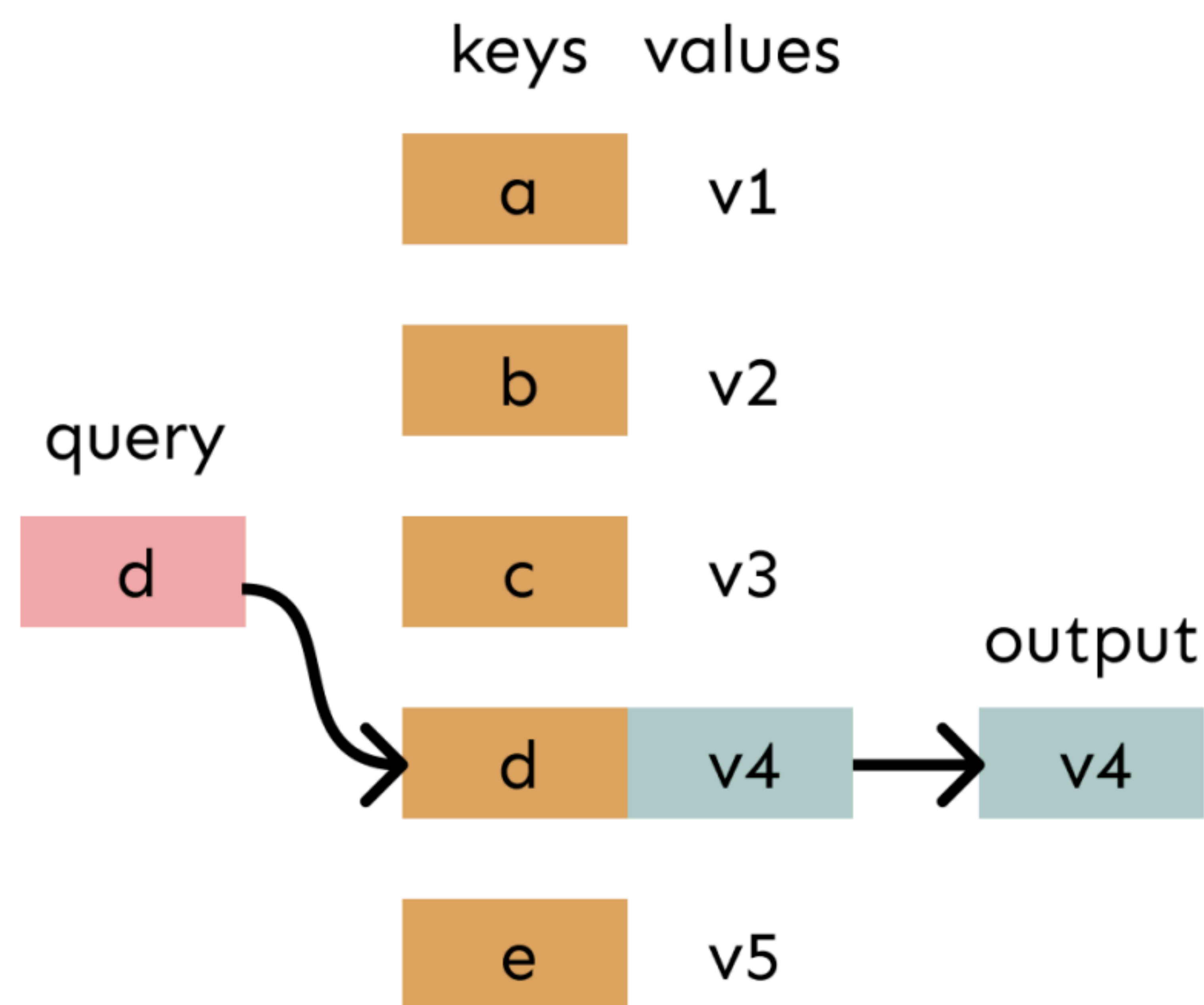
- We sometimes say that the query attends to the values.
  - For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values)
  - Here, keys and values are the same!
- The weighted sum is a **selective summary** of the information contained 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

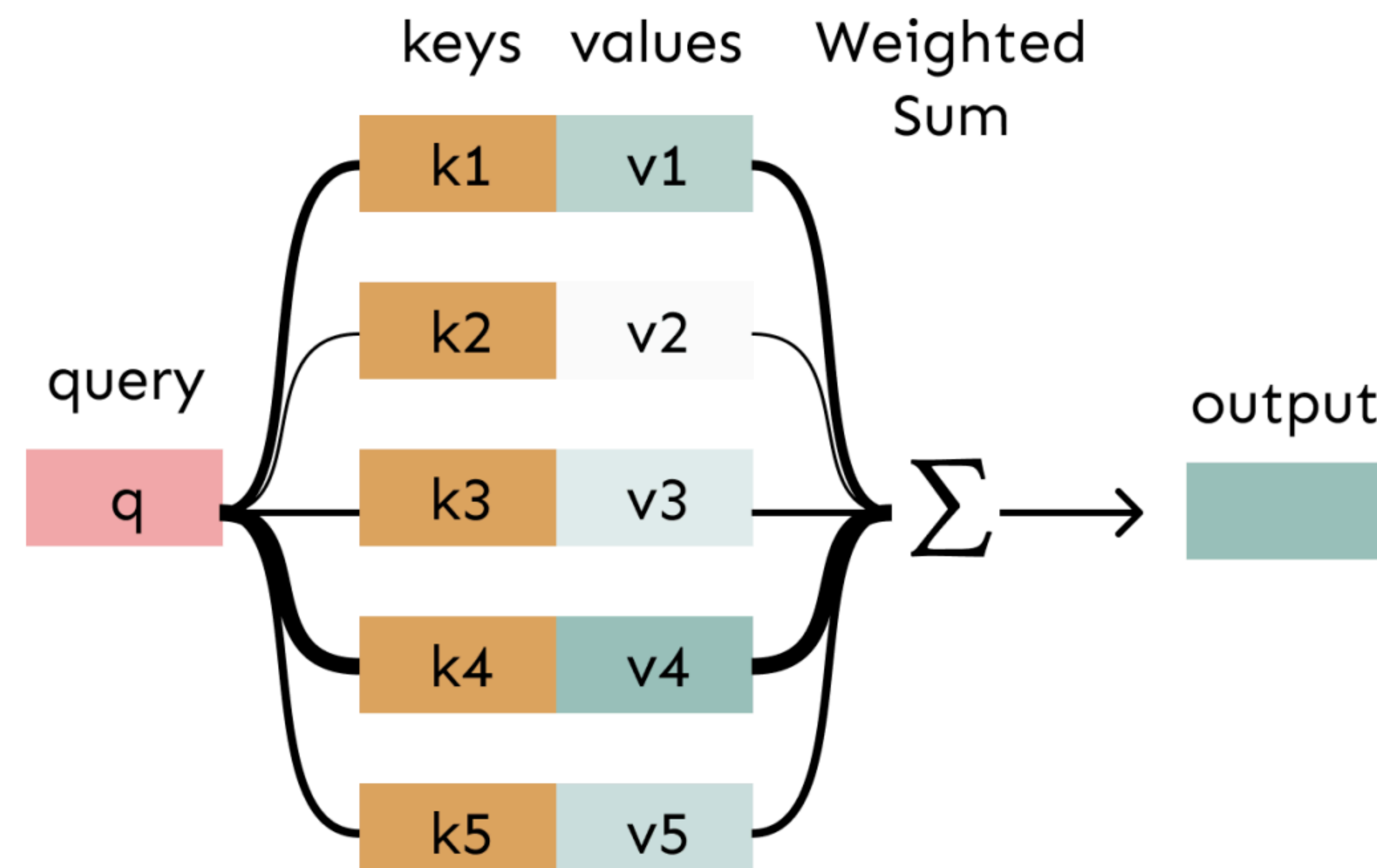
Attention and lookup tables

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.

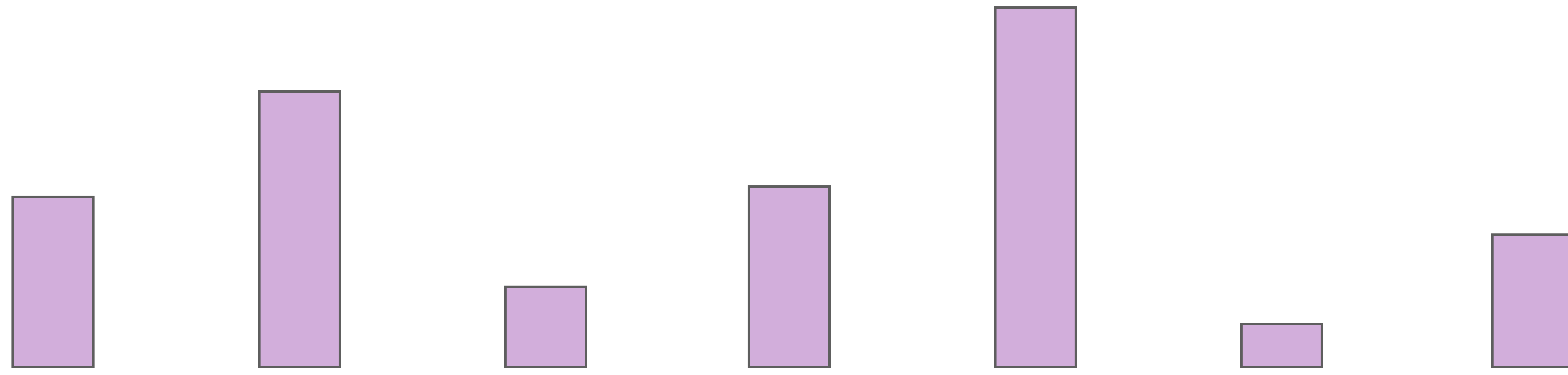
In attention, the query matches all keys softly, to a weight between 0 and 1. The keys' values are multiplied by the weights and summed.



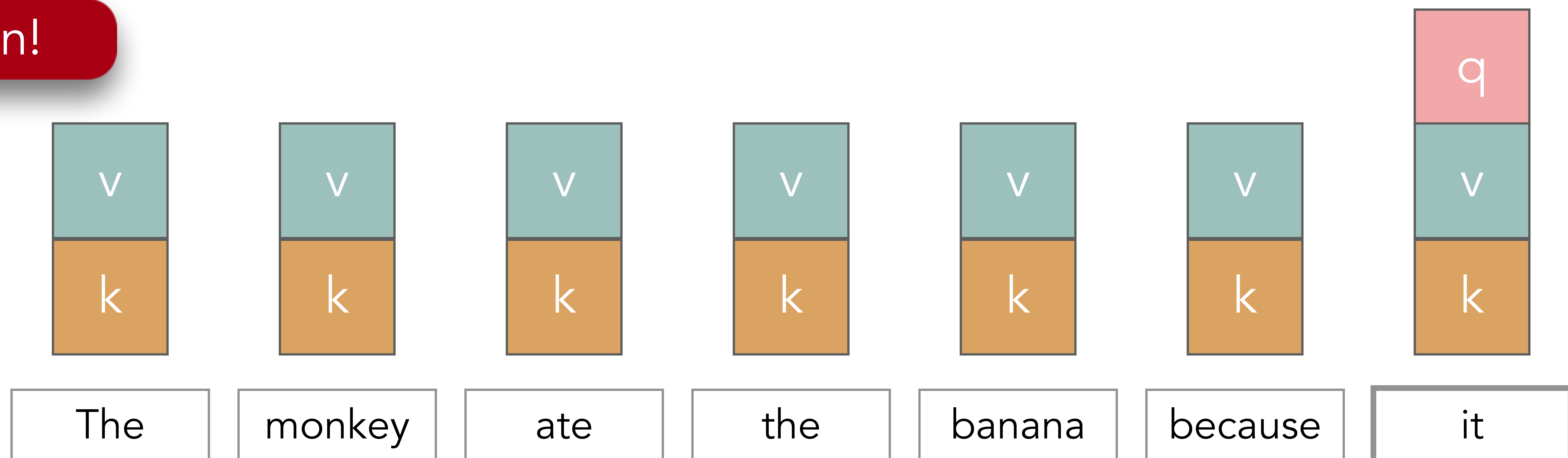


USC Viterbi

Attention in the decoder



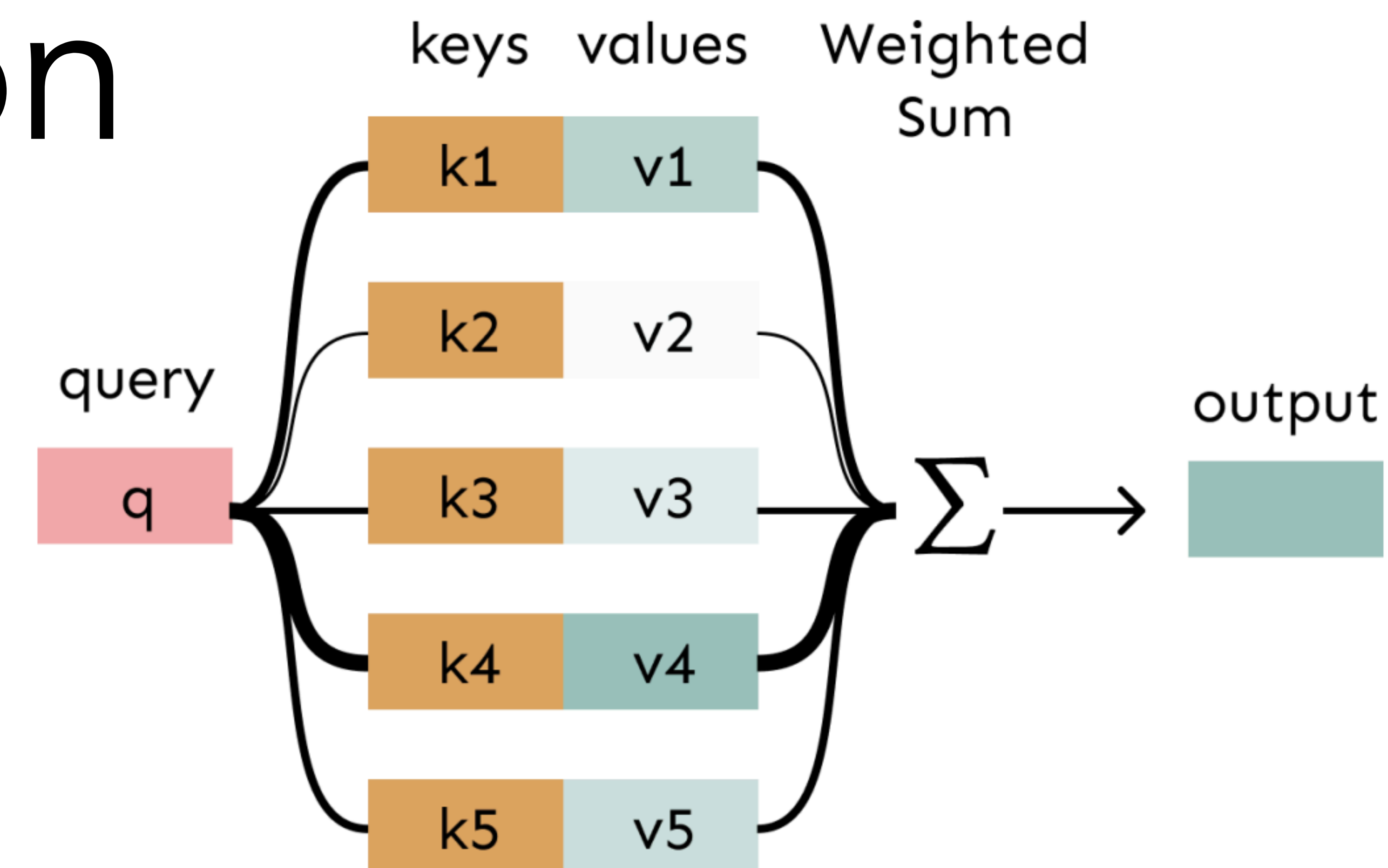
Self-Attention!



Transformers:
Self-Attention Networks

USC Viterbi

Self-Attention



Keys, Queries, Values from the same sequence

Let $\mathbf{w}_{1:N}$ be a sequence of words in vocabulary V

For each $\mathbf{w}_i$, 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

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

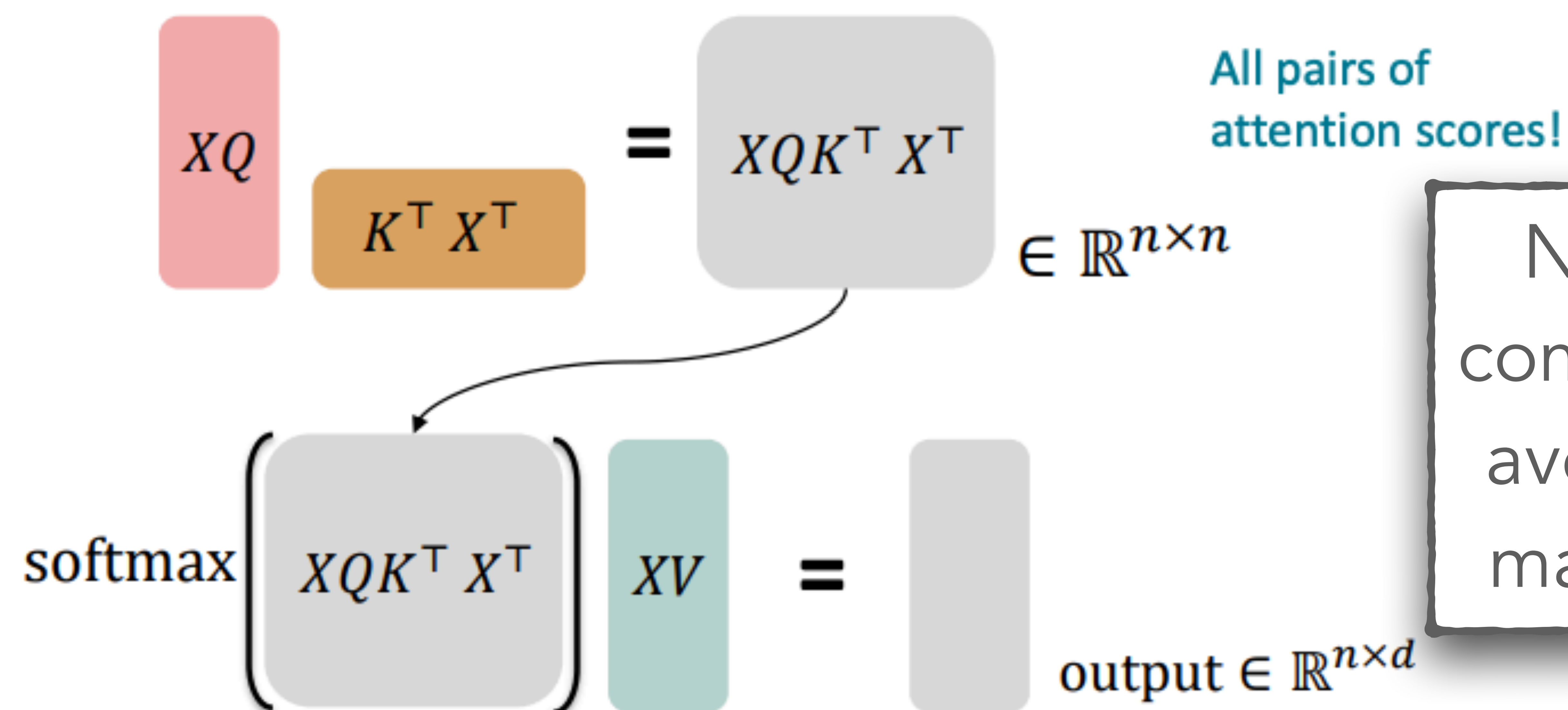
3. Compute output for each word as weighted sum of values

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

Self-Attention as Matrix Multiplications

- 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$(\mathbf{XQ}(\mathbf{XK})^T)\mathbf{XV} \in \mathbb{R}^{n \times d}$

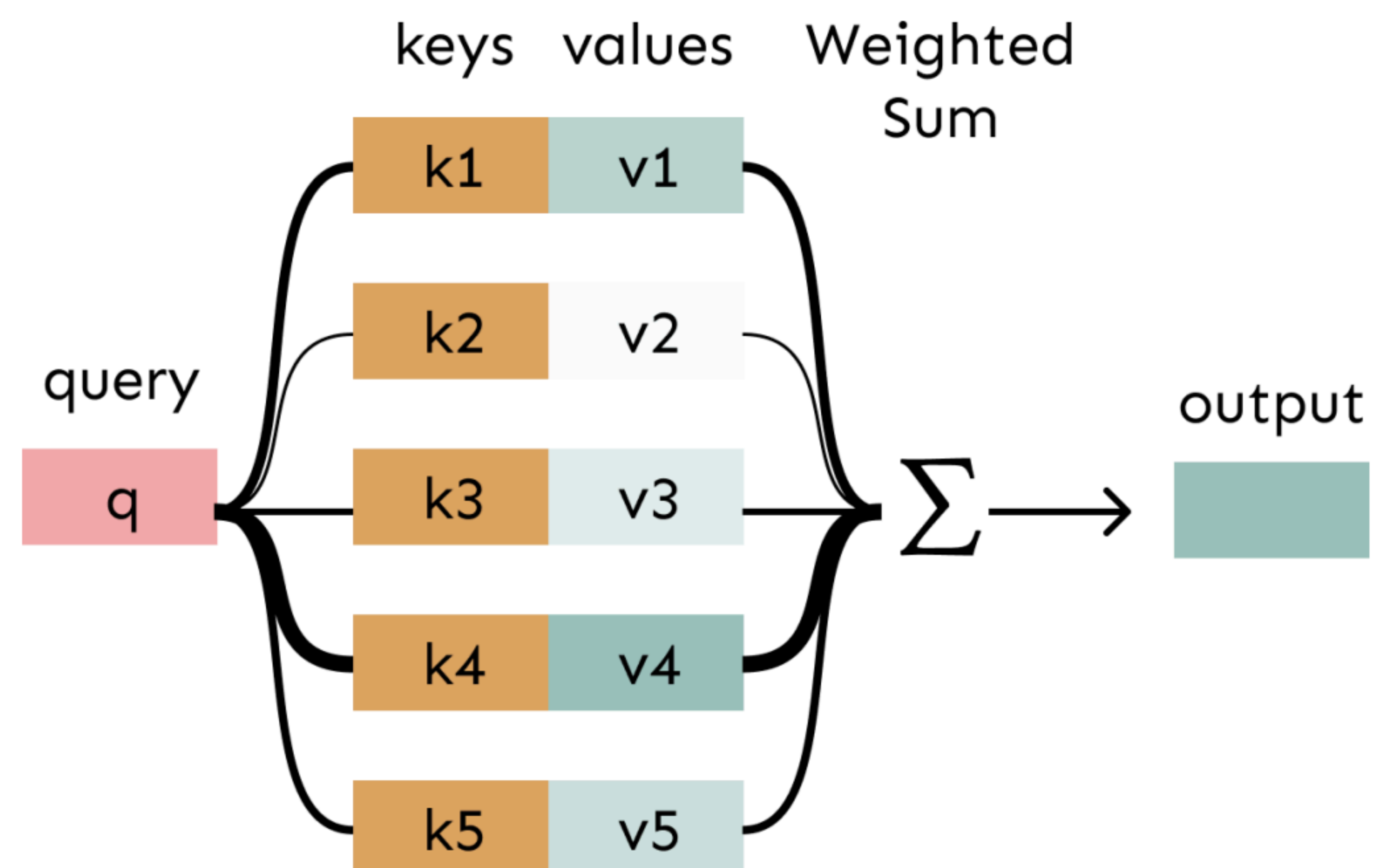
First, take the query-key dot products in one matrix multiplication: $\mathbf{XQ}(\mathbf{XK})^T$



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

USC Viterbi

Why Self-Attention?



- 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!

USC Viterbi

Transformers are Self-Attention Networks

- Self-Attention is the key innovation behind Transformers!
- Transformers 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

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

- **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.