Lecture 10: Transformers

Instructor: Swabha Swayamdipta

USC CSCI 444 NLP

Oct 01, 2025

Some slides adapted from Dan Jurafsky and Chris Manning

Announcements + Logistics

- Proposal grades out today
- Today: Quiz 2
- Next Monday: Guest Lecture by PhD student, Johnny Wei
  - PyTorch for Transformers - important for project / homework
  - No Office Hours, but send questions via email
- Next Wednesday: Fall Break
- HW2 due on October 15, was previously October 13
- Office Hours resume from October 13

Quiz 2
Password: activation

Lecture Outline

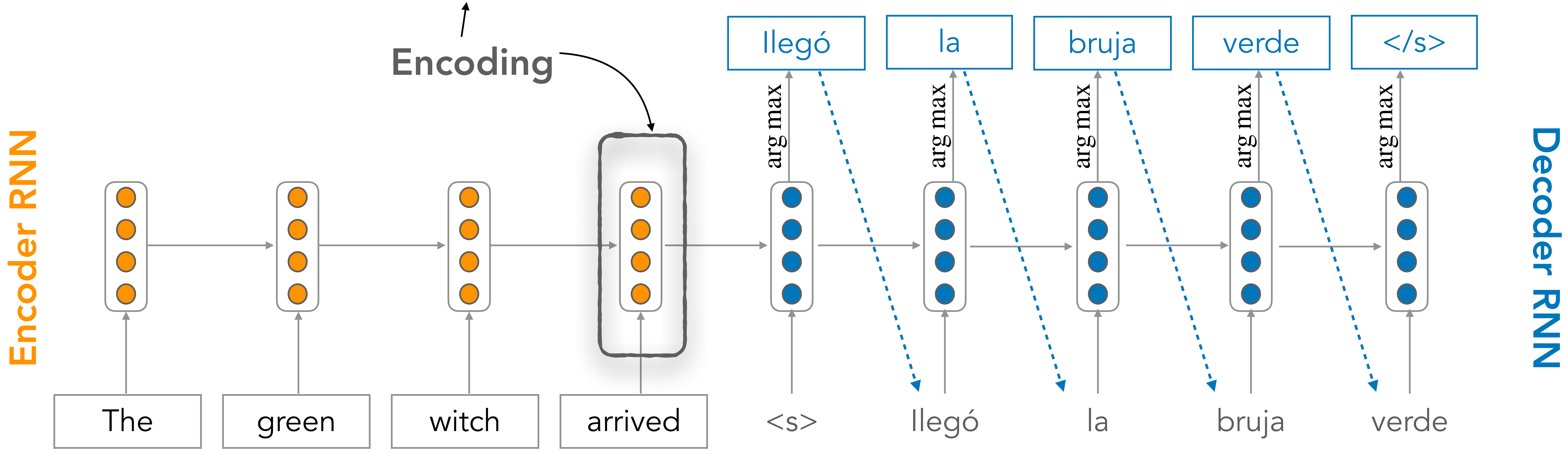
- Recap: Attention
- Transformers: Self-Attention Networks
  - Multi-Headed Attention
  - Positional Embeddings
- Transformer Blocks
- Transformer Encoders, Decoders and Encoder-Decoders

Recap: Encoder Decoder Networks and Attention

USC Viterbi

This needs to capture all information about the source sentence. Information bottleneck!

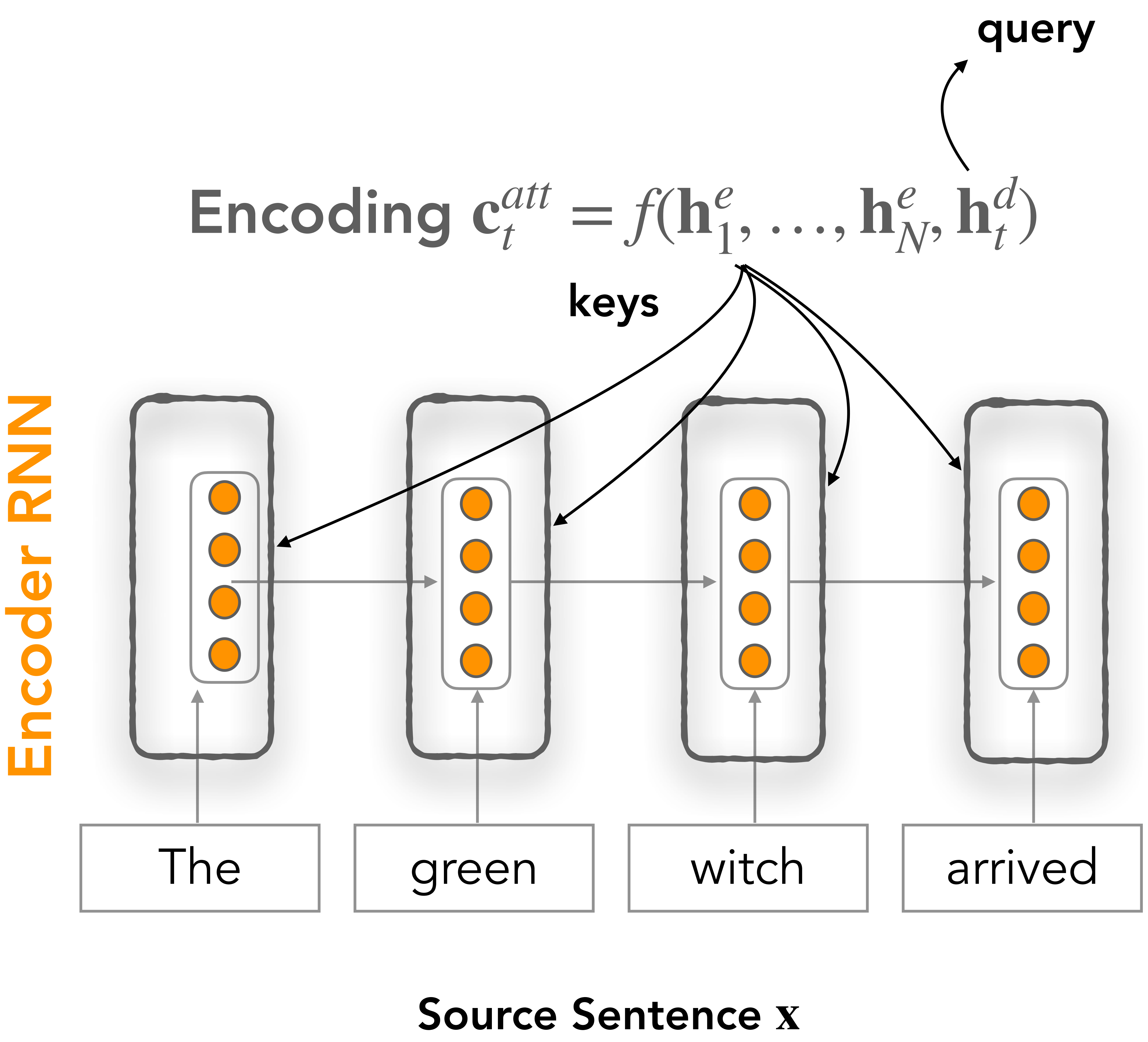
Target Sentence $\mathbf{y}$



Source Sentence $\mathbf{x}$

Attention Mechanism

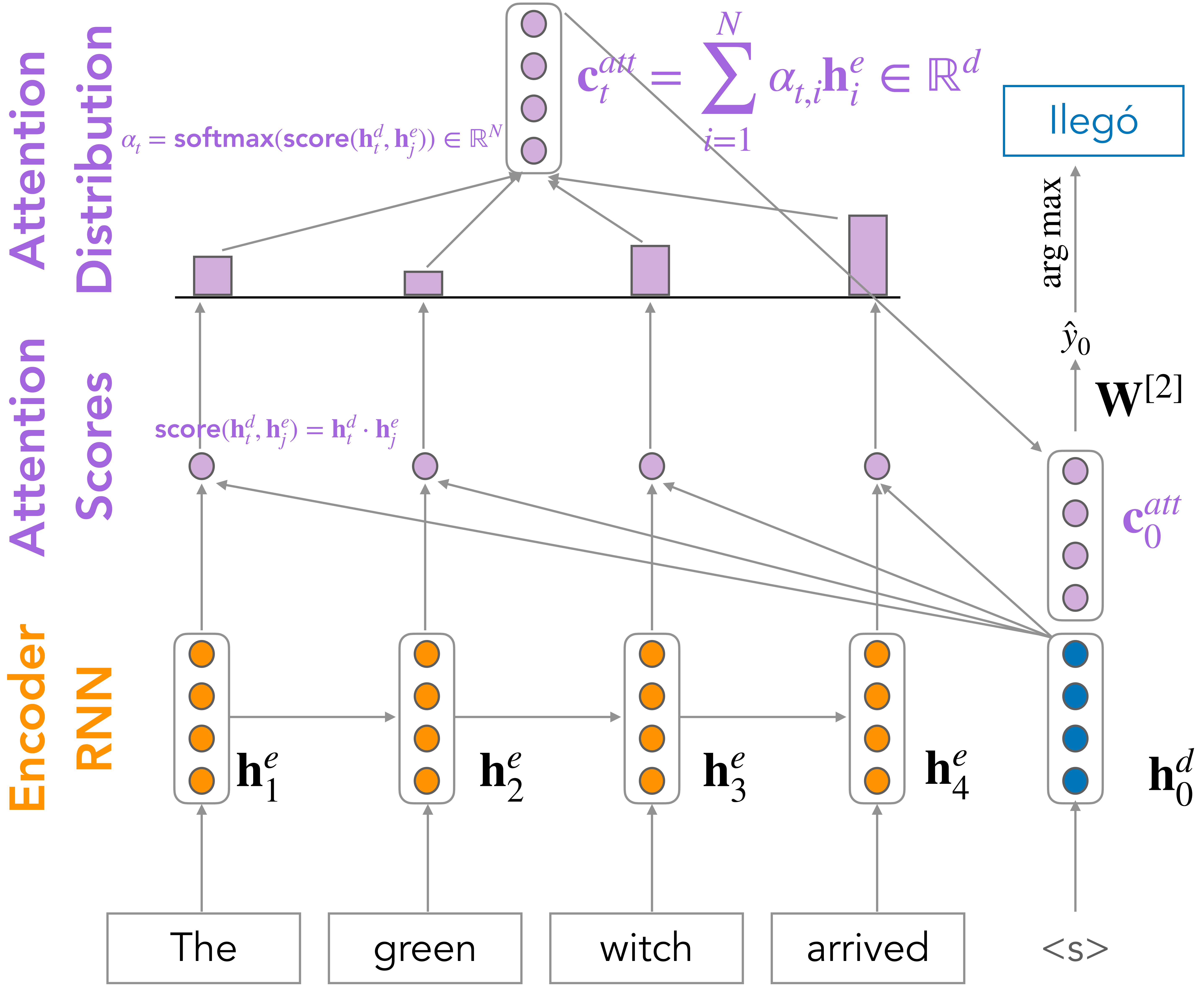
- Attention mechanisms allow the decoder to focus on a particular part of the source sequence at each time step
- Fixed-length vector $\mathbf{c}_t^{att}$ (attention context vector)
  - Take a weighted sum of all the encoder hidden states
  - One vector per time step *of the decoder*!
  - Weights *attend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



Bahdanau et al., 2015

Note: Notation different from J&M

USC Viterbi

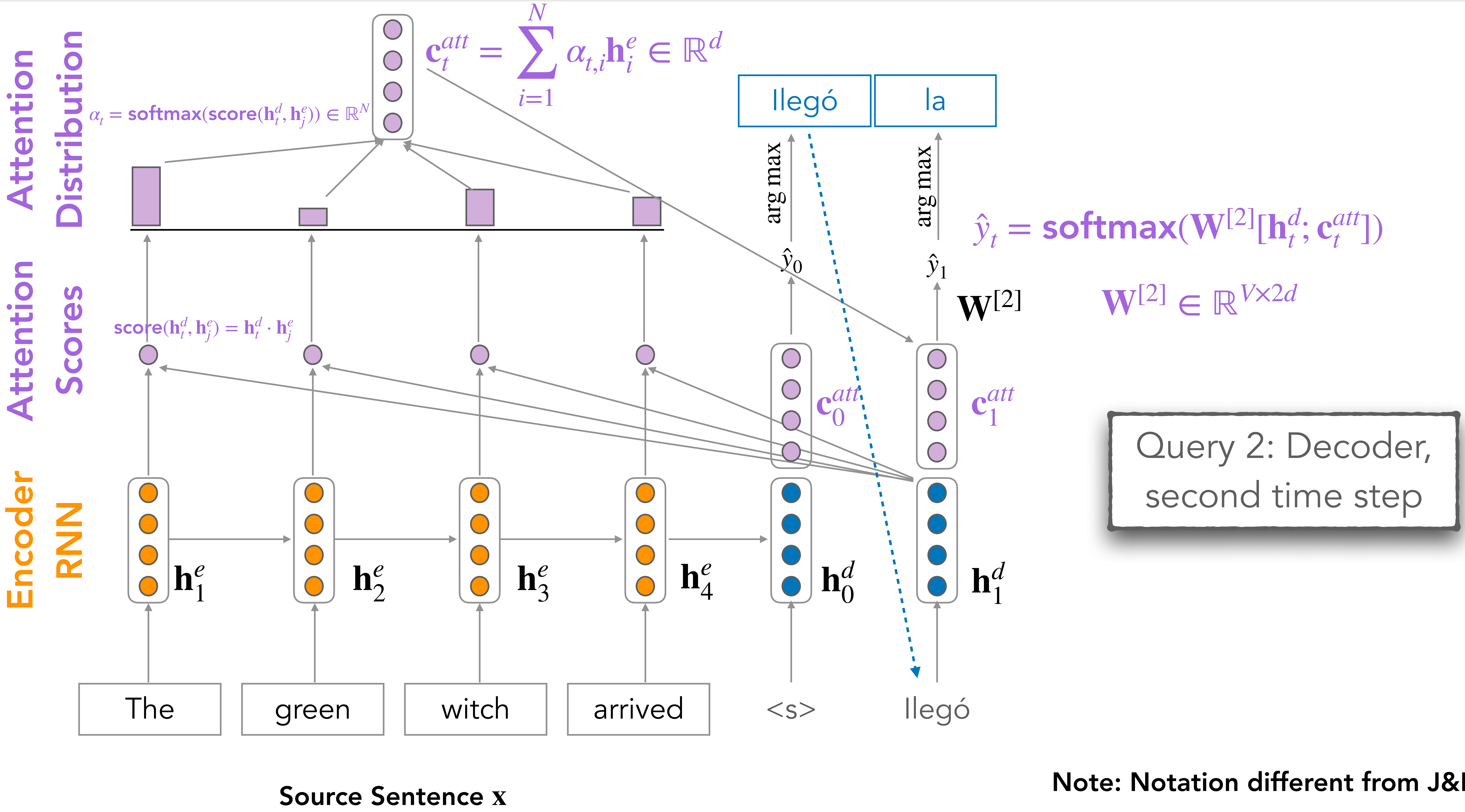


$$\hat{y}_t = \text{softmax}(\mathbf{W}^{[2]}[\mathbf{h}_t^d; \mathbf{c}_t^{att}])$$

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

Source Sentence x

Note: Notation different from J&M

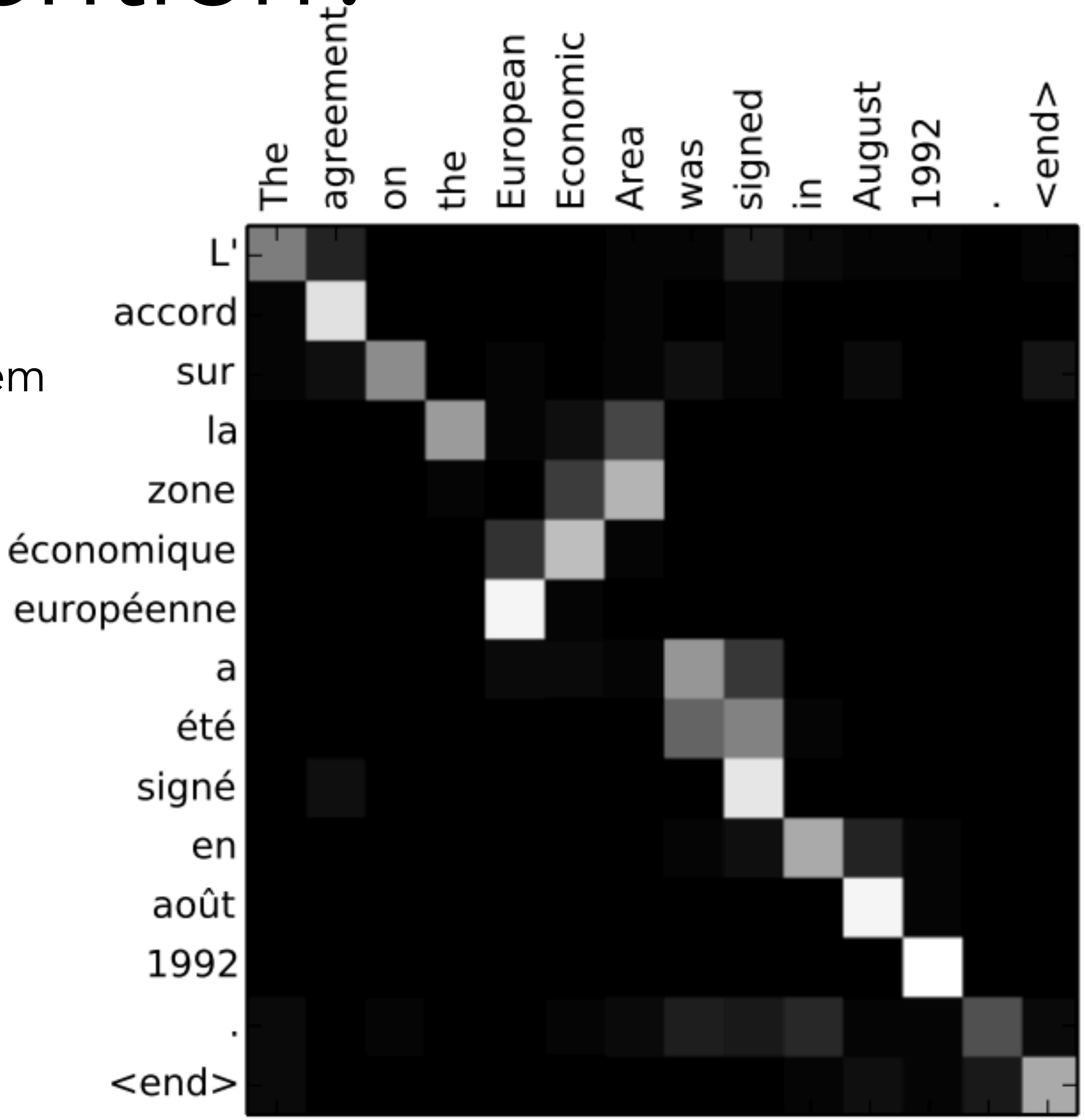


Note: Notation different from J&M

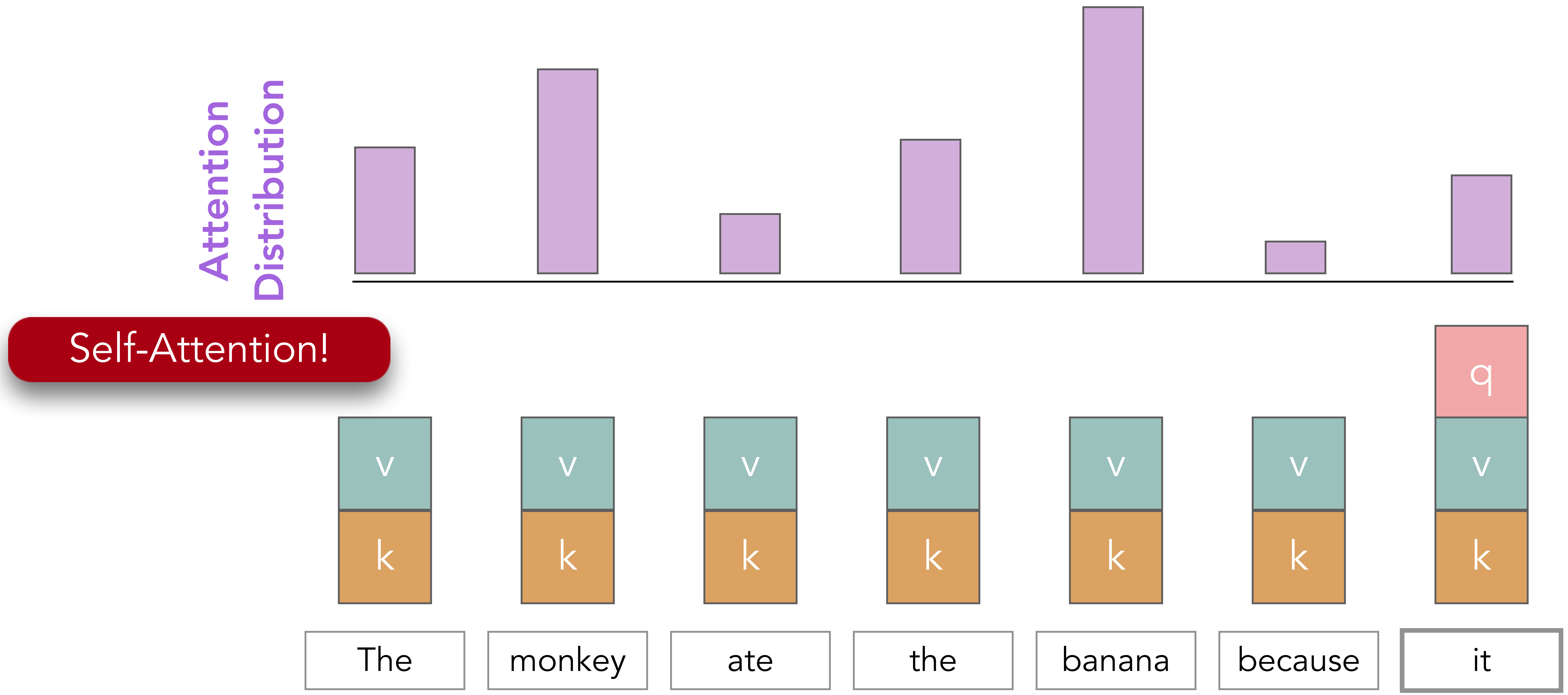
USC Viterbi

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

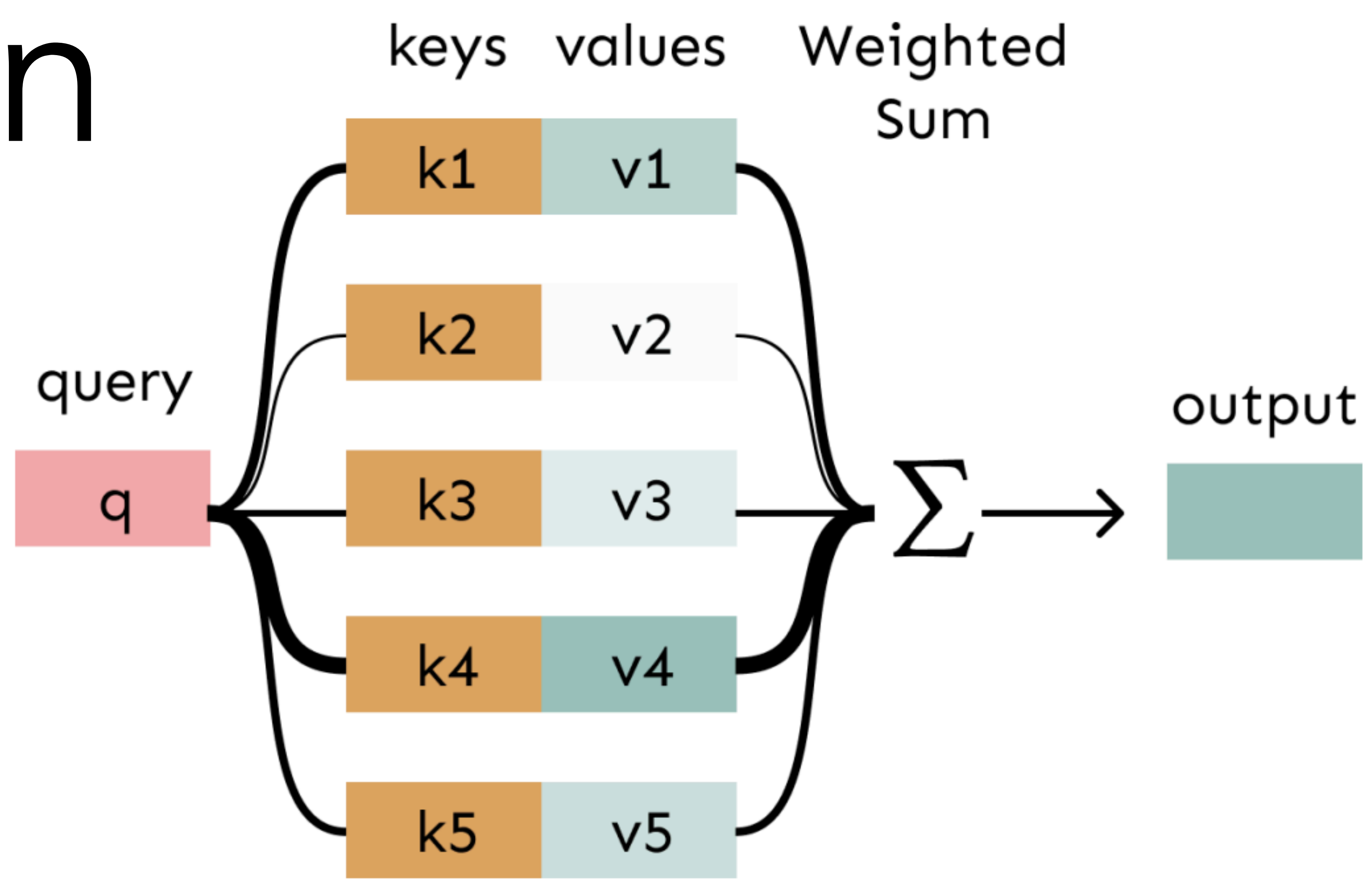


Attention in the decoder



Transformers:
Self-Attention Networks

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

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

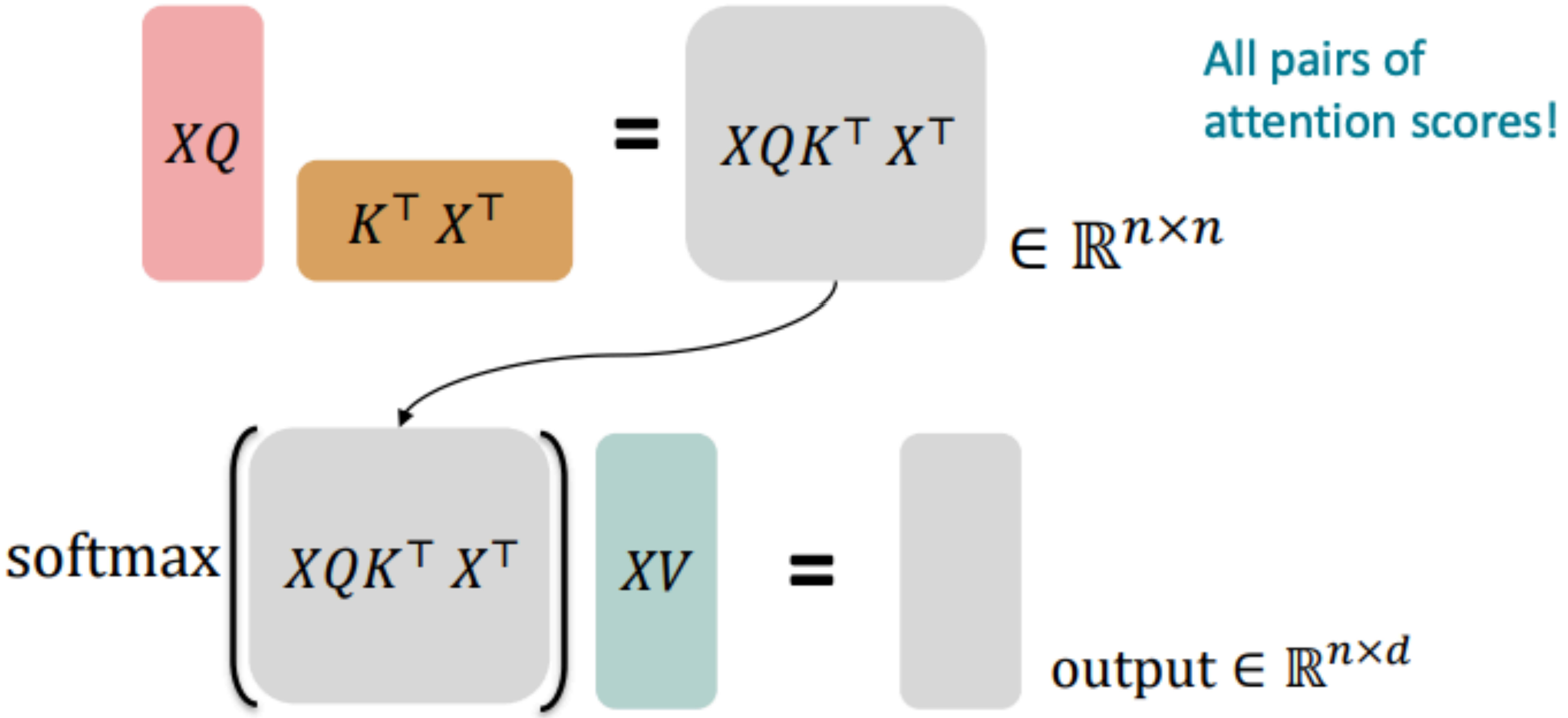
3. Compute output for each word as weighted sum of values

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

Self-Attention as Matrix Multiplications

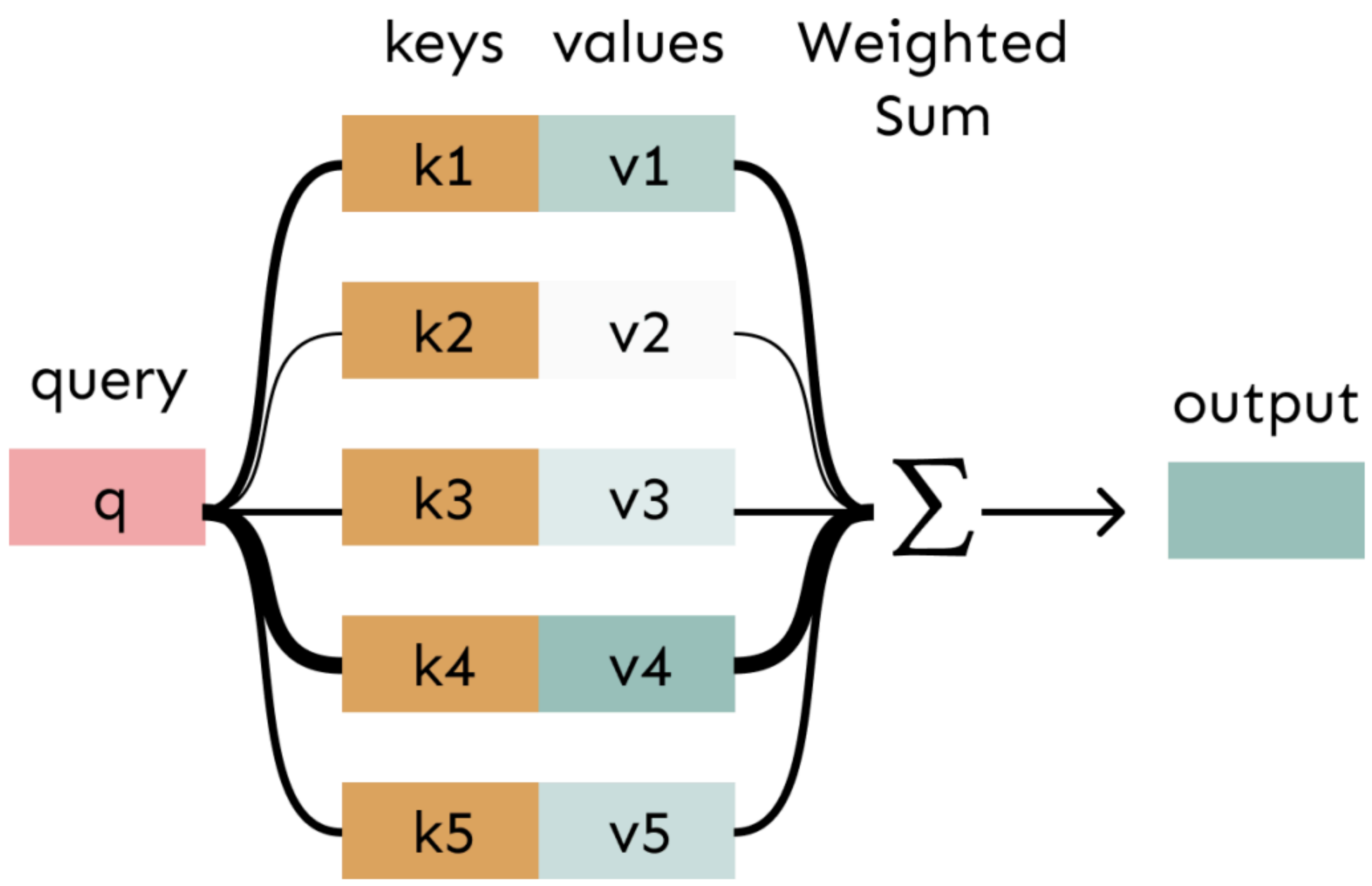
- Key-query-value attention is typically computed as matrices.
  - Let $\mathbf{X} = [\mathbf{x}_1; \ldots; \mathbf{x}_n] \in \mathbb{R}^{n \times d}$ be the concatenation of input vectors
  - First, note that $\mathbf{XK} \in \mathbb{R}^{n \times d}$, $\mathbf{XQ} \in \mathbb{R}^{n \times d}$, and $\mathbf{XV}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

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!

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
    - No 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

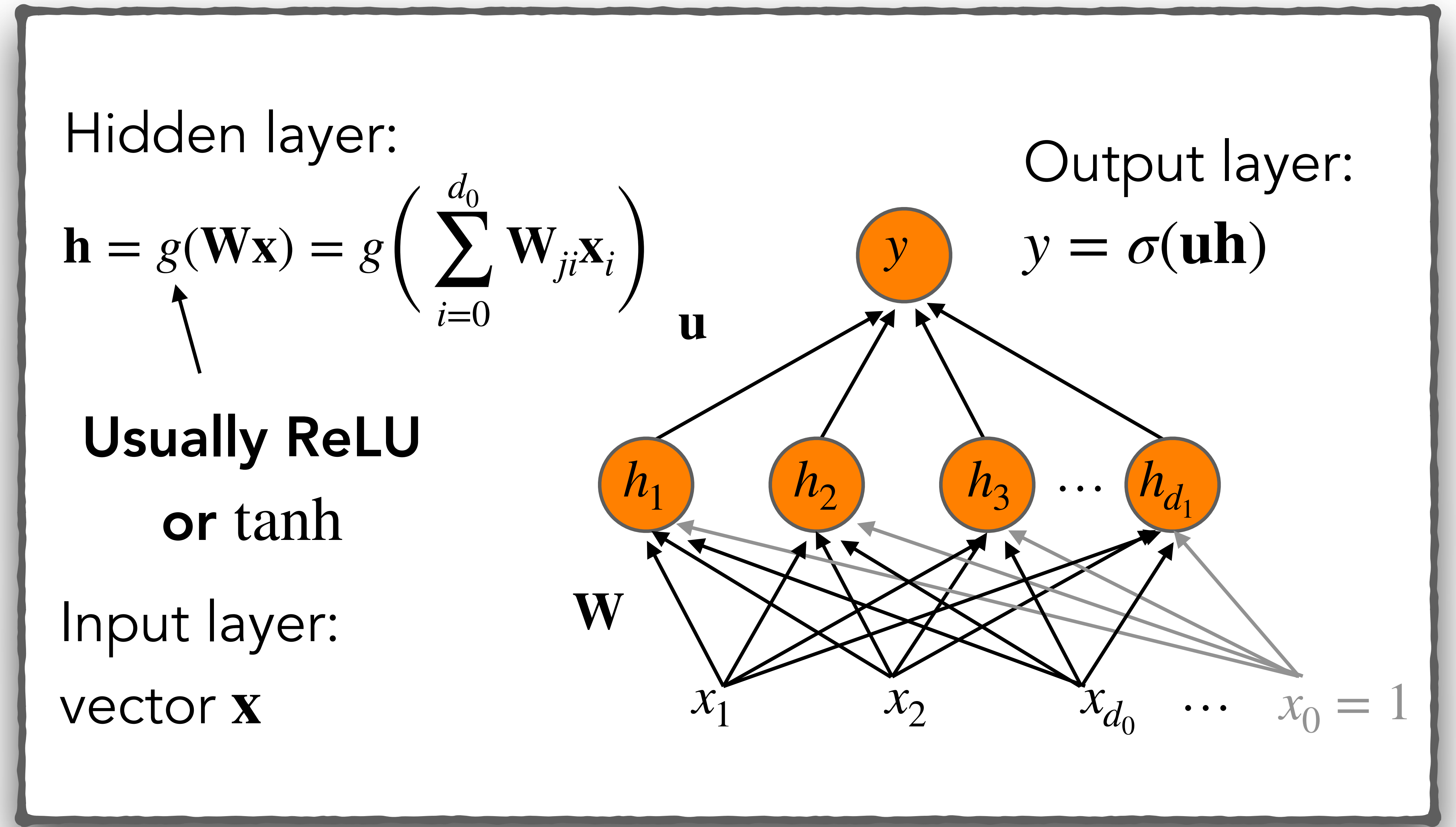
Aidan N. Gomez[*] [†] University of Toronto aidan@cs.toronto.edu

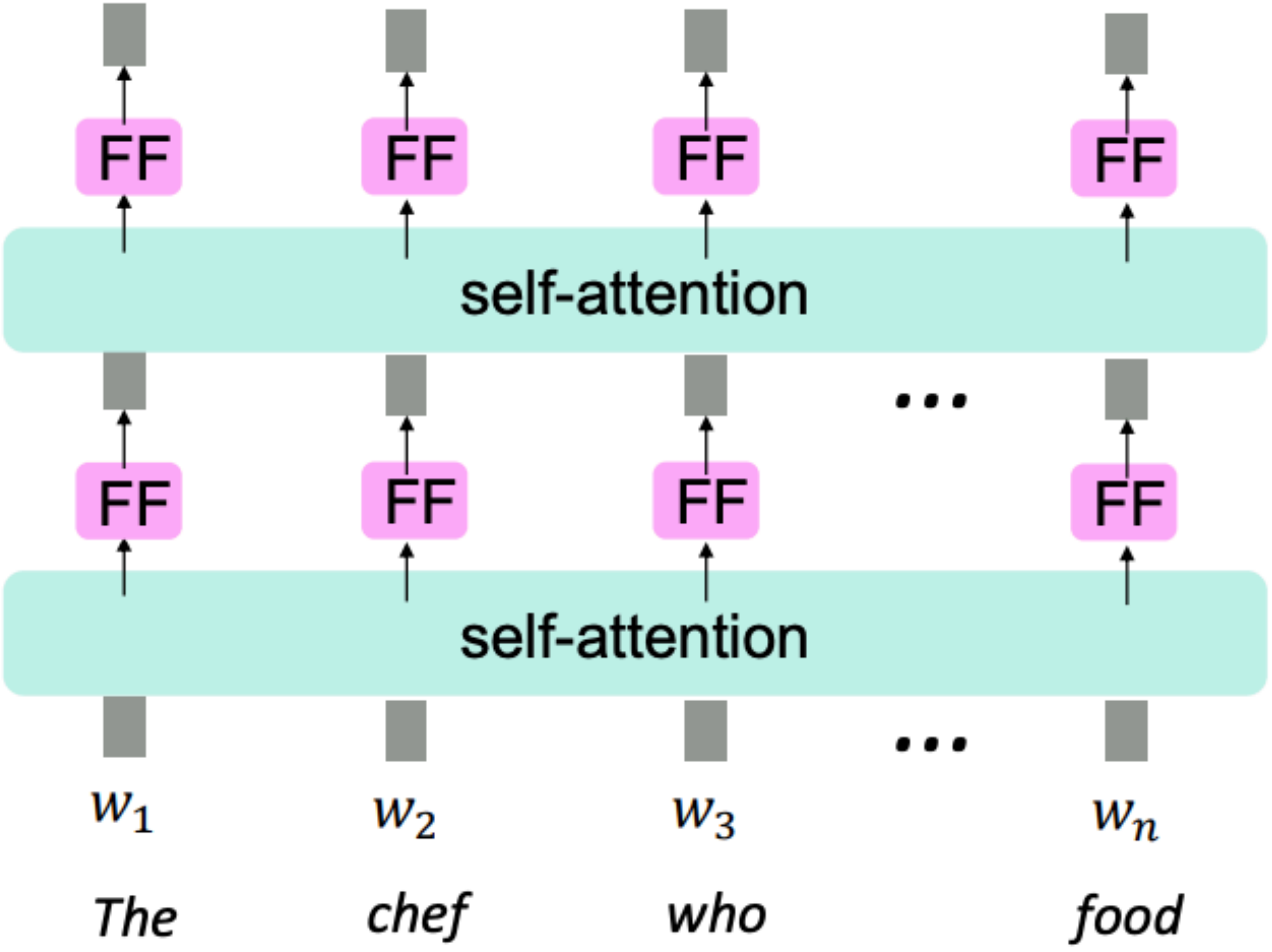
Łukasz Kaiser[*] Google Brain lukaszkaiser@google.com

Illia Polosukhin[*] [‡] illia.polosukhin@gmail.com

Self-Attention and Weighted Averages

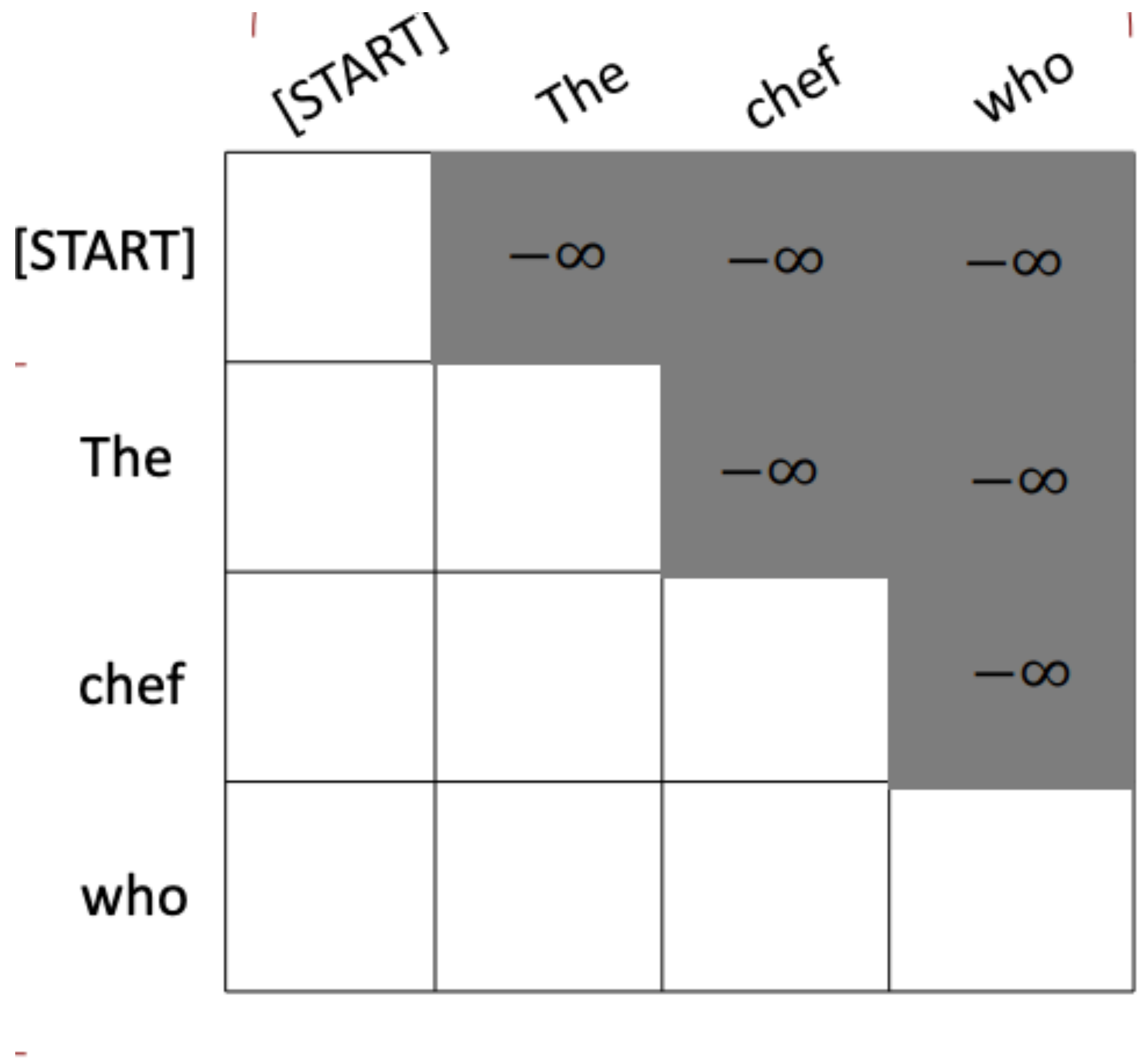
- **Problem**: there are no element-wise nonlinearities in self-attention; stacking more self-attention layers just re-averages value vectors
- **Solution**: add a feed-forward network to post-process each output vector.





Self Attention and Future Information

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

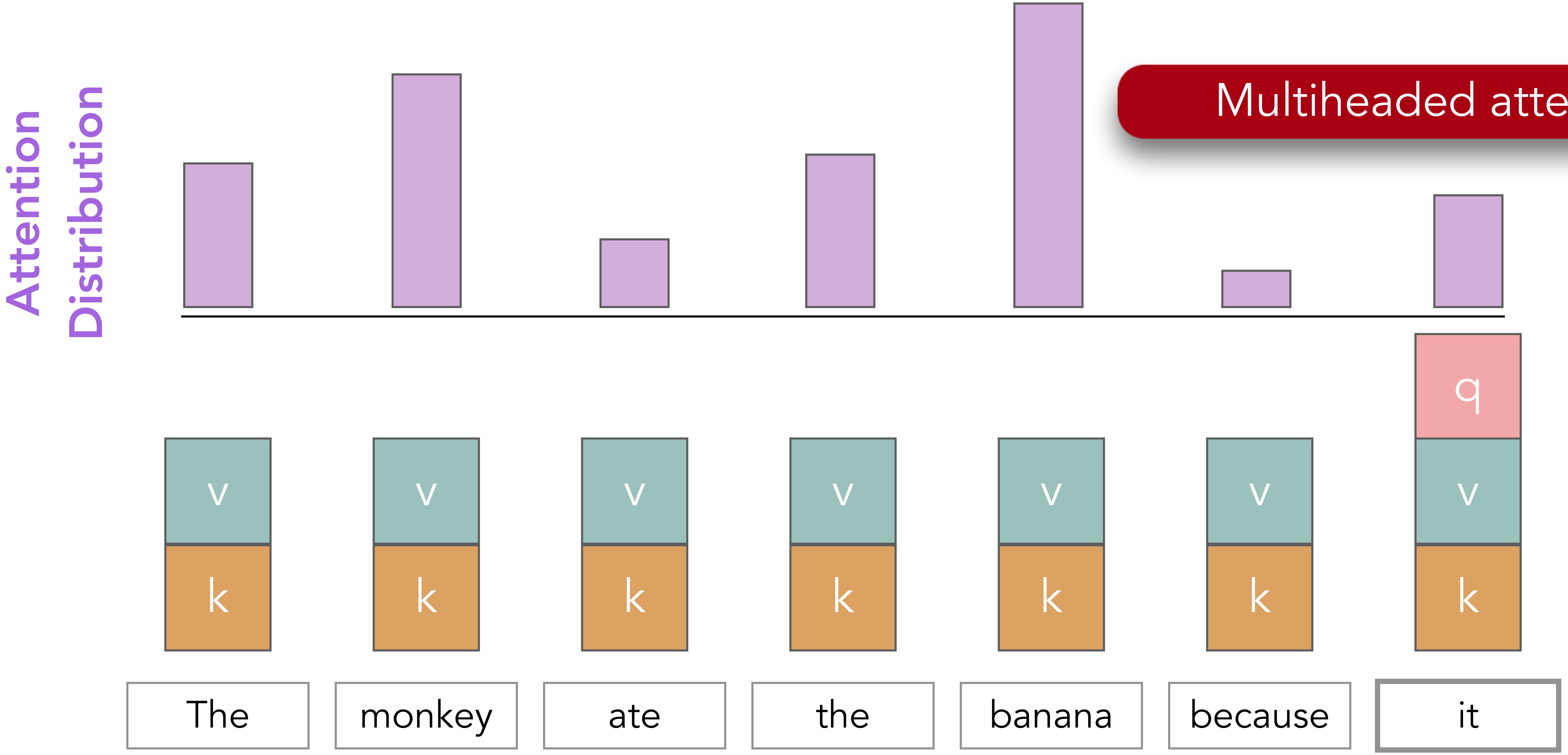


USC Viterbi

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

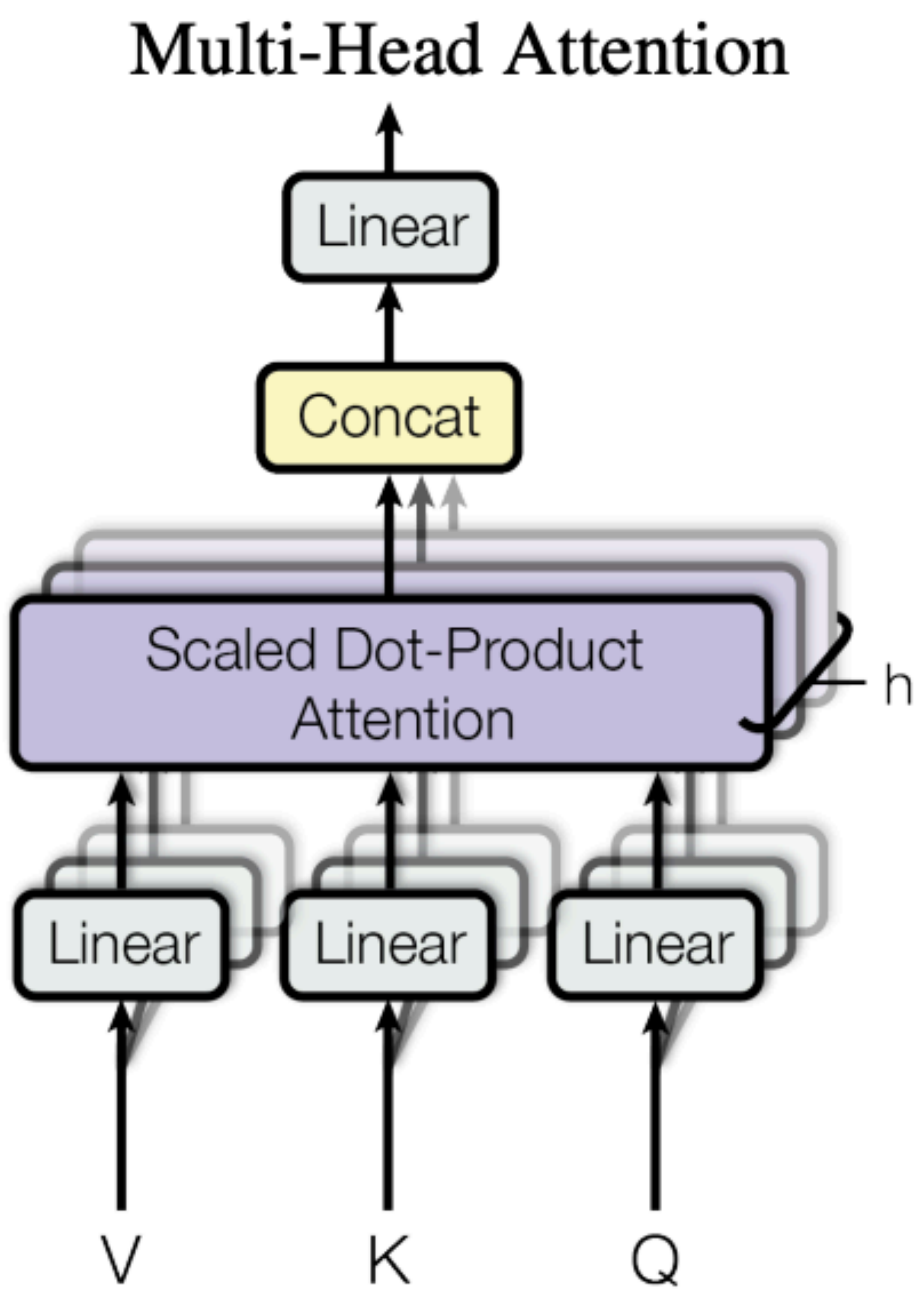
Multiheaded attention



Transformers: Multiheaded 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?
- 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!



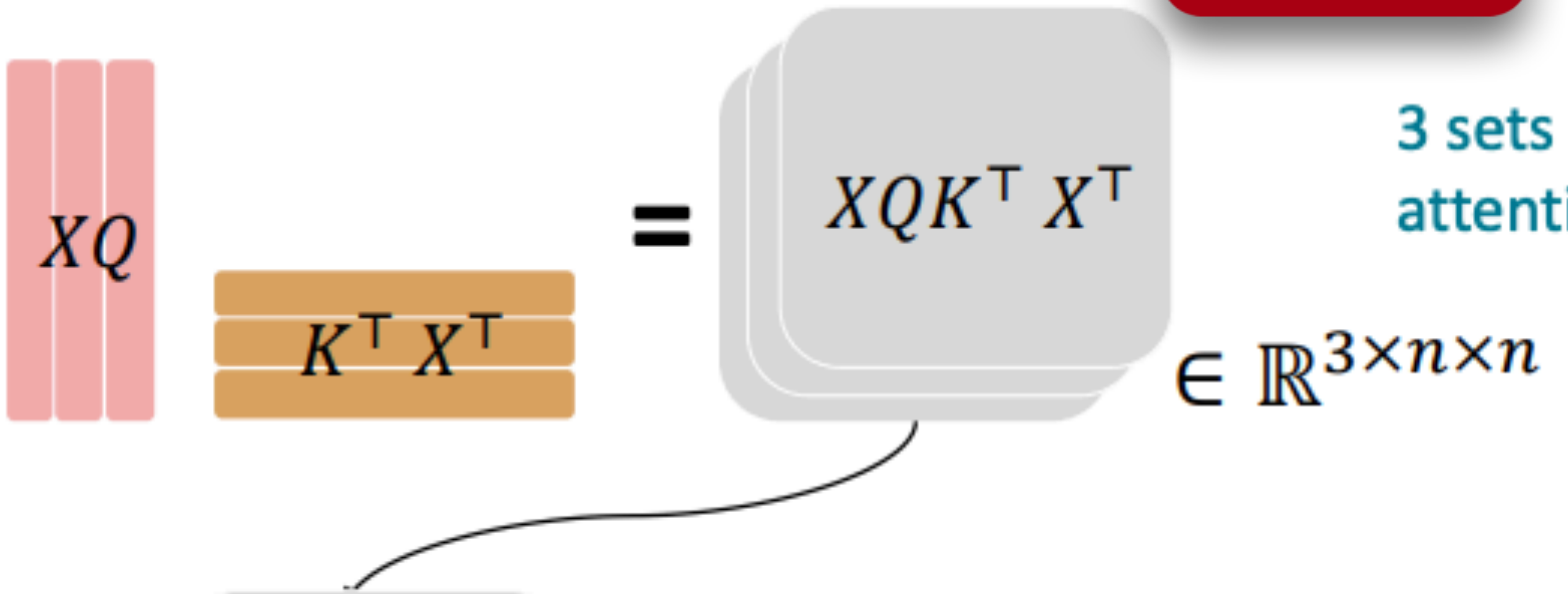
Each head gets to "look" at different things, and construct value vectors differently

Multiheaded Attention: Visualization

Still efficient, can be parallelized!

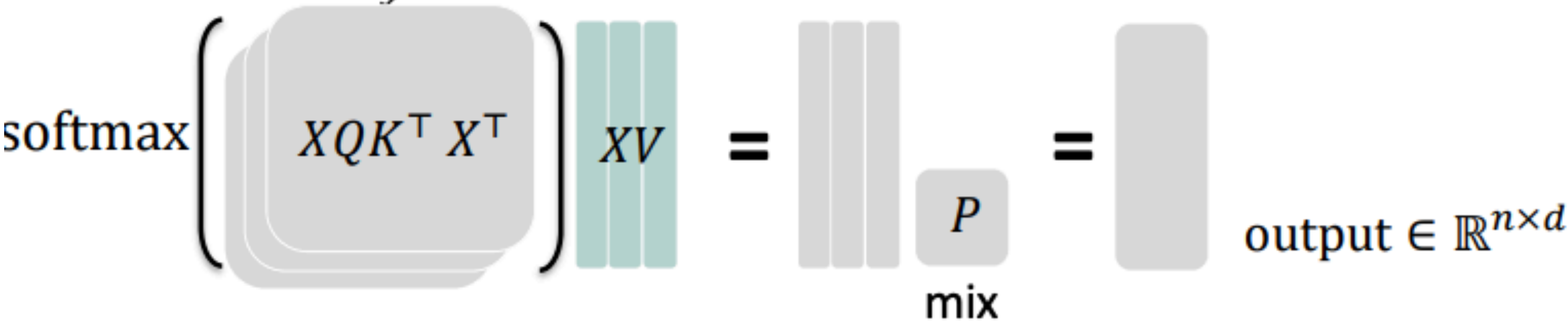
Tensor!

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



3 sets of all pairs of attention scores!

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

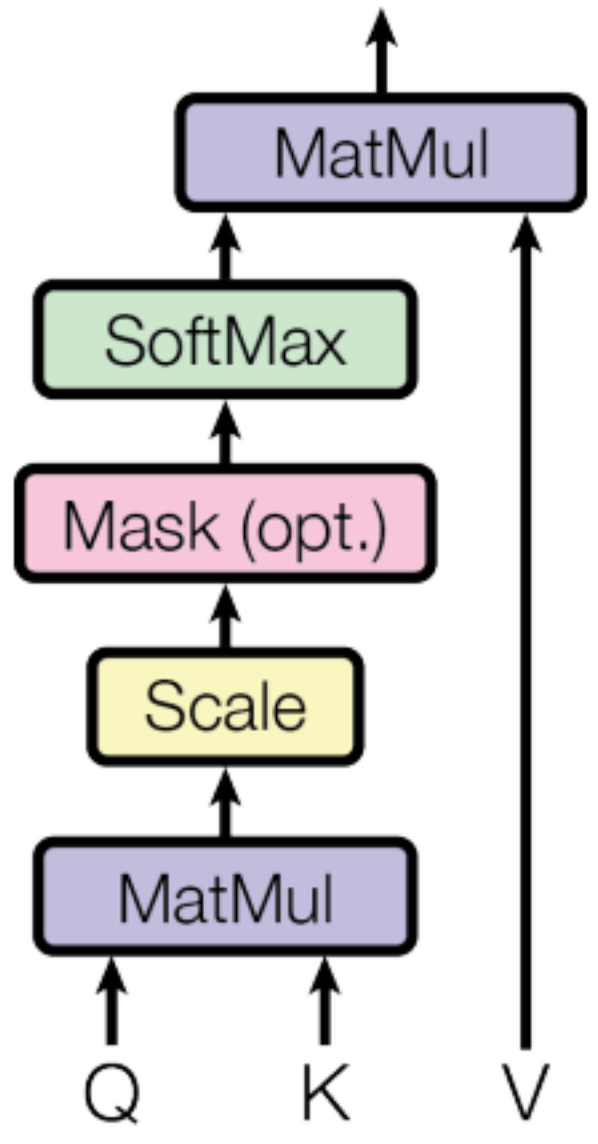


 USCViterbi

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



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

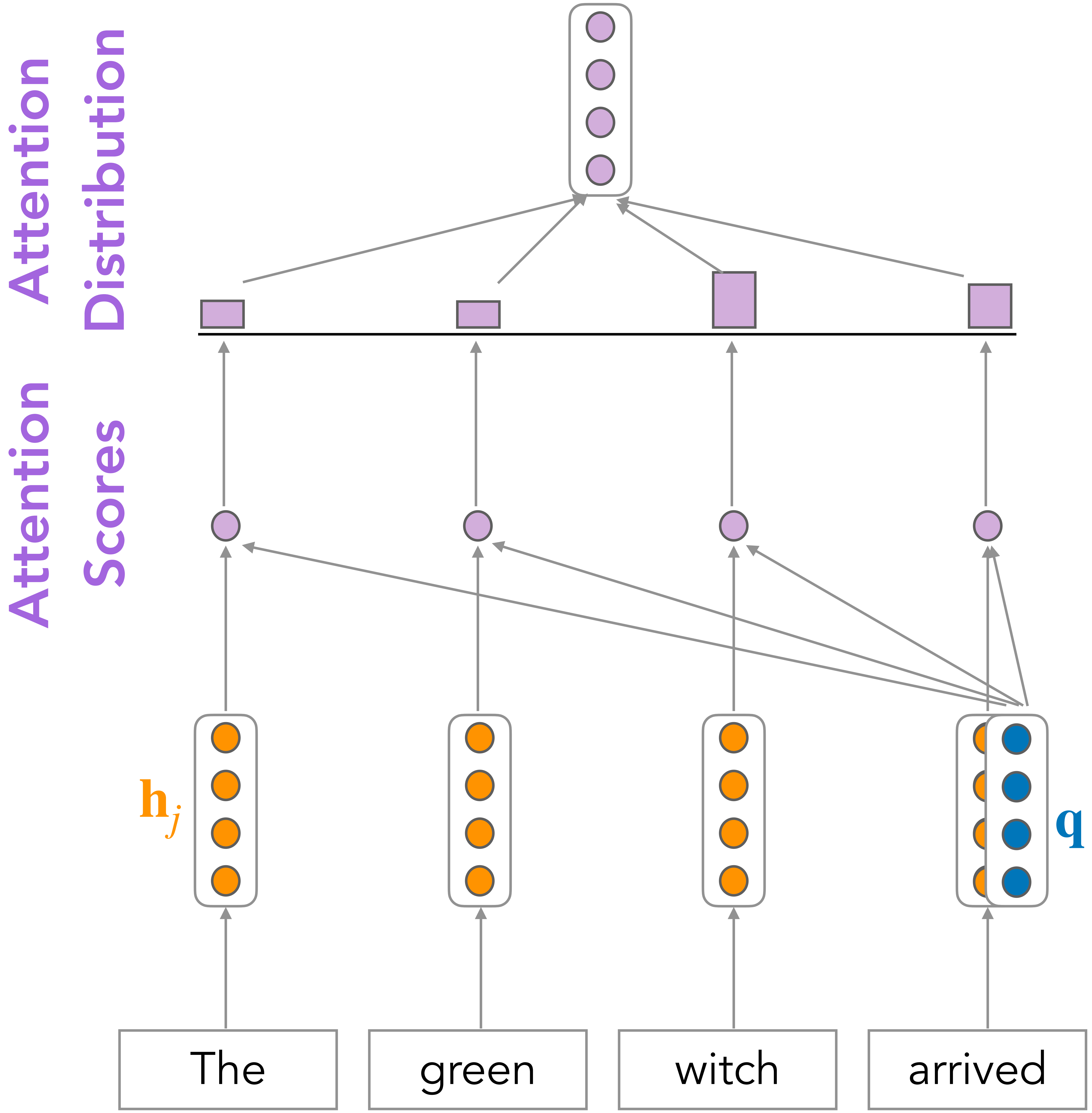
- We divide the attention scores by $\sqrt{d/h}$, to stop the scores from becoming large just as a function of d/h, where h is the number of heads

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

Self-Attention: Order Information?

- Self-attention 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

- 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!
- 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…

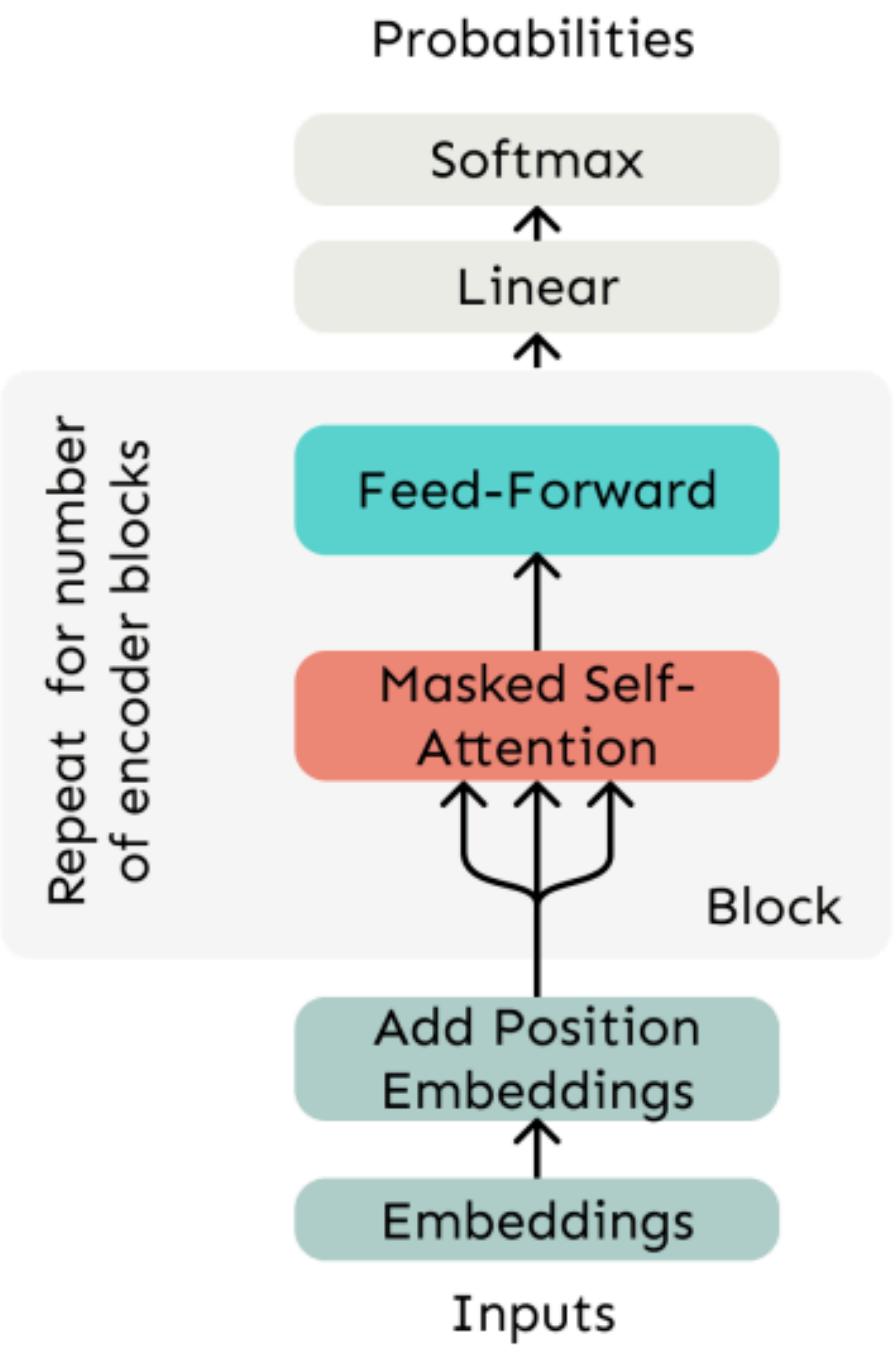
Positional Embeddings

- 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

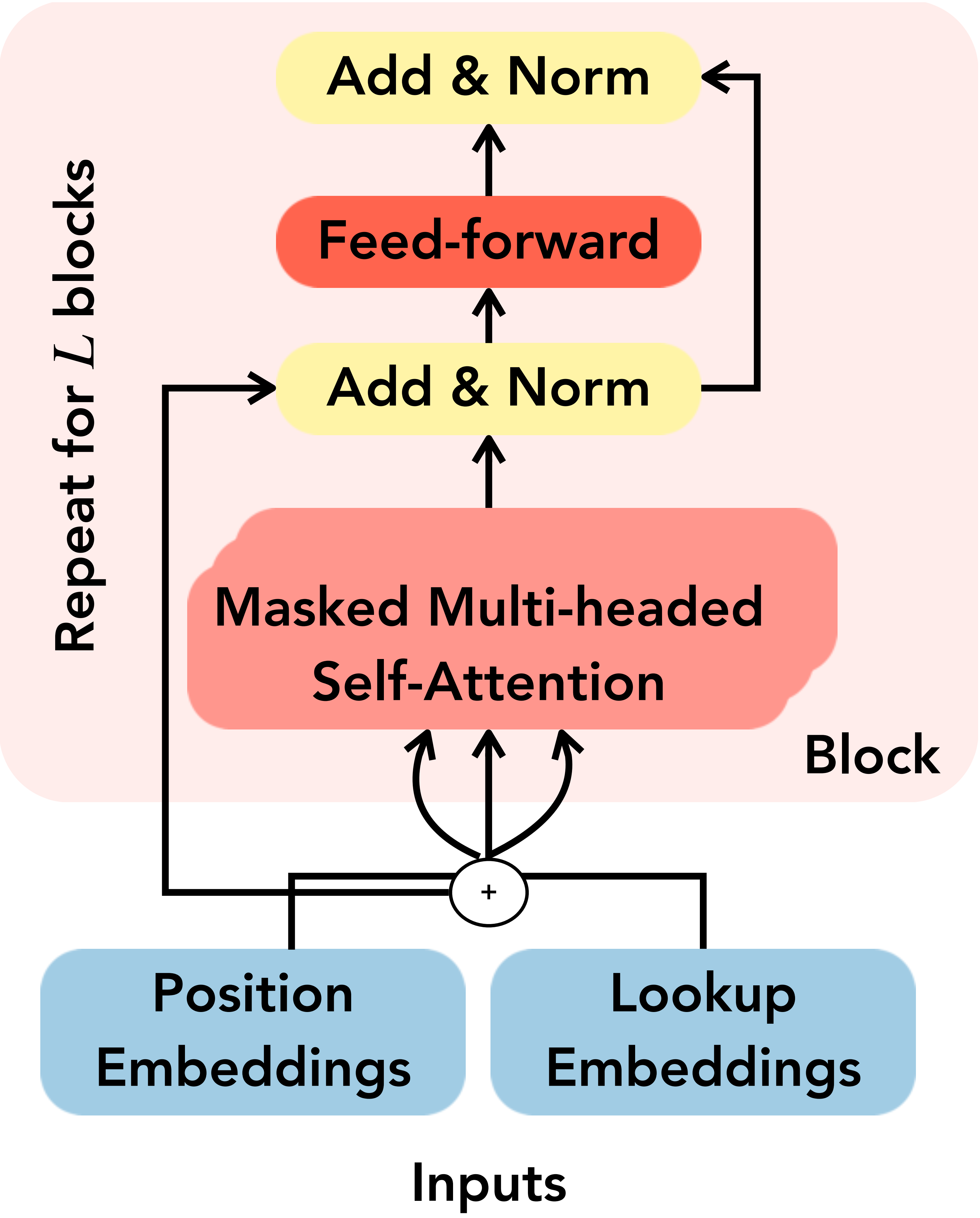
- Self-attention:
  - the basis of the method; with multiple heads
- Position representations:
  - Specify the sequence order, since self-attention is an unordered function of its inputs.
- Nonlinearities:
  - At the output of the self-attention block
  - Frequently implemented as a simple feedforward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



Transformers as Encoders, Decoders and Encoder-Decoders

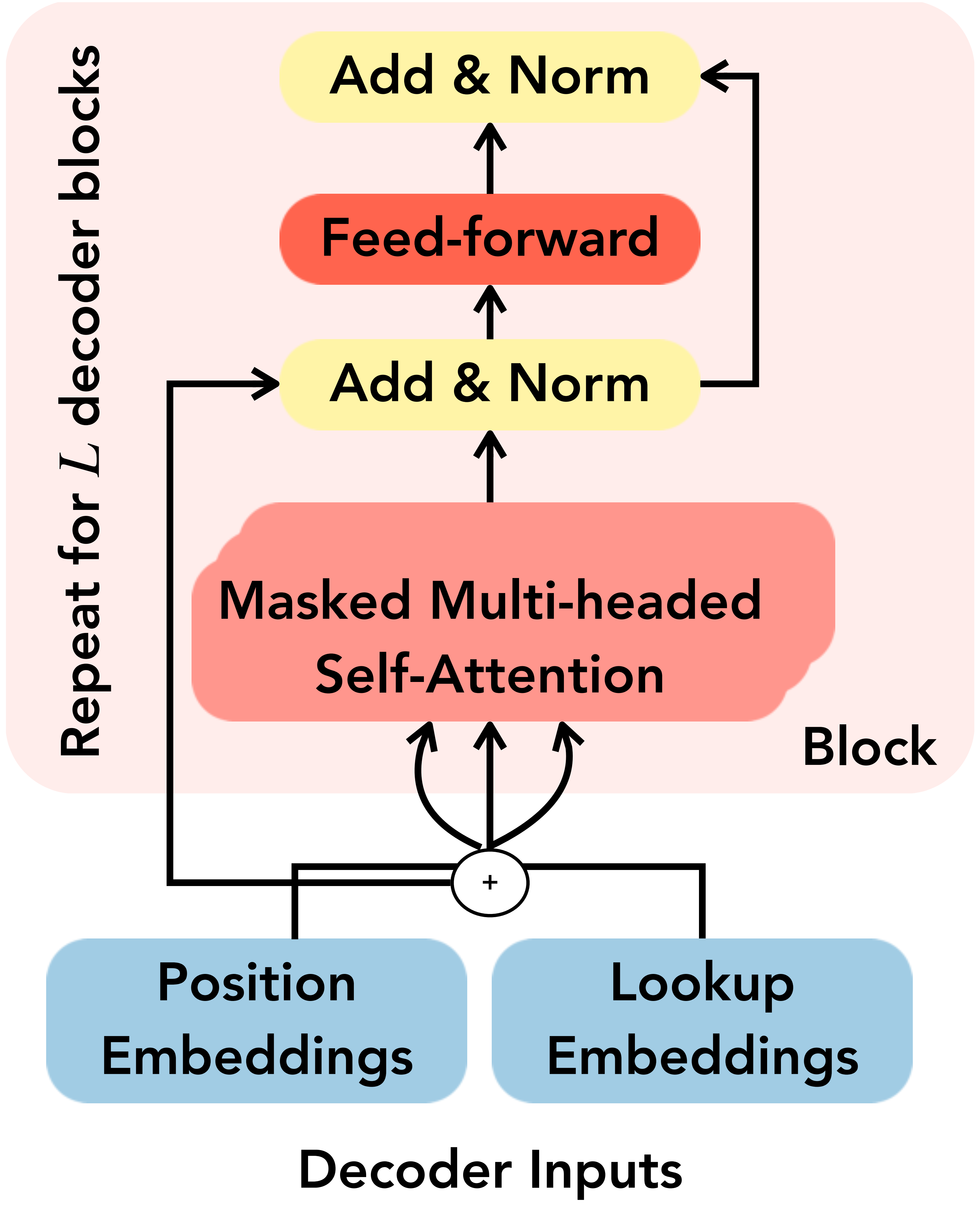
The Transformer Model

- Transformers are made up of stacks of transformer blocks, each of which is a multilayer network made by combining feedforward networks and **self-attention layers**, the key innovation of self-attention transformers
- The Transformer Decoder-only model corresponds to
  - a Transformer language model
- Lookup embeddings for tokens are usually randomly initialized
  - Input tokenization (in 1-2 classes)



The Transformer Decoder

- Two optimization tricks that help training:
  - Residual Connections
  - Layer Normalization
- In most Transformer diagrams, these are often written together as "Add & Norm"
  - Add: Residual Connections
  - Norm: Layer Normalization



Transformer Decoder

Residual Connections

- Original Connections: $X^{(i)} = \text{Layer}(X^{(i-1)})$ where i represents the layer
- **Residual Connections** : trick to help models train better.
  - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$
    - so we only have to learn "the residual" from the previous layer

Allowing information to skip a layer improves learning and gives higher level layers **direct access to information** from lower layers (He et al., 2016).

Layer Normalization

- Layer normalization is another trick to help models train faster
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation **within each layer**
- Let $x \in \mathbb{R}^d$ be an individual (word) vector in the model.

$$\mu = \frac{1}{d}\sum_{j=1}^{d} x_j; \quad \mu \in \mathbb{R} \qquad\qquad \sigma = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(x_j - \mu)^2}; \quad \sigma \in \mathbb{R}$$

Result: New vector with zero mean and a standard deviation of one

$$\hat{x} = \frac{x - \mu}{\sigma}$$

Component-wise subtraction

- Let $\gamma \in \mathbb{R}$ and $\beta \in \mathbb{R}^d$ be learned "gain" and "bias" parameters. (Can omit!)

$$\textbf{LayerNorm} = \gamma\hat{x} + \beta$$

Xu et al., 2019

The Transformer Decoder

- The Transformer Decoder is a stack of Transformer Decoder Blocks.
- Each Block consists of:
  - Self-attention
  - Add & Norm
  - Feed-Forward
  - Add & Norm
- Output layer is as always a softmax layer