UNIVERSITY OF GEORGIA

CSCI 4360/6360 Data Science II

Attention Mechanism

Ninghao Liu
Assistant Professor
School of Computing
University of Georgia

The majority of contents are adopted from Stanford CS224N/Ling284. (Abigail See and Richard Socher)

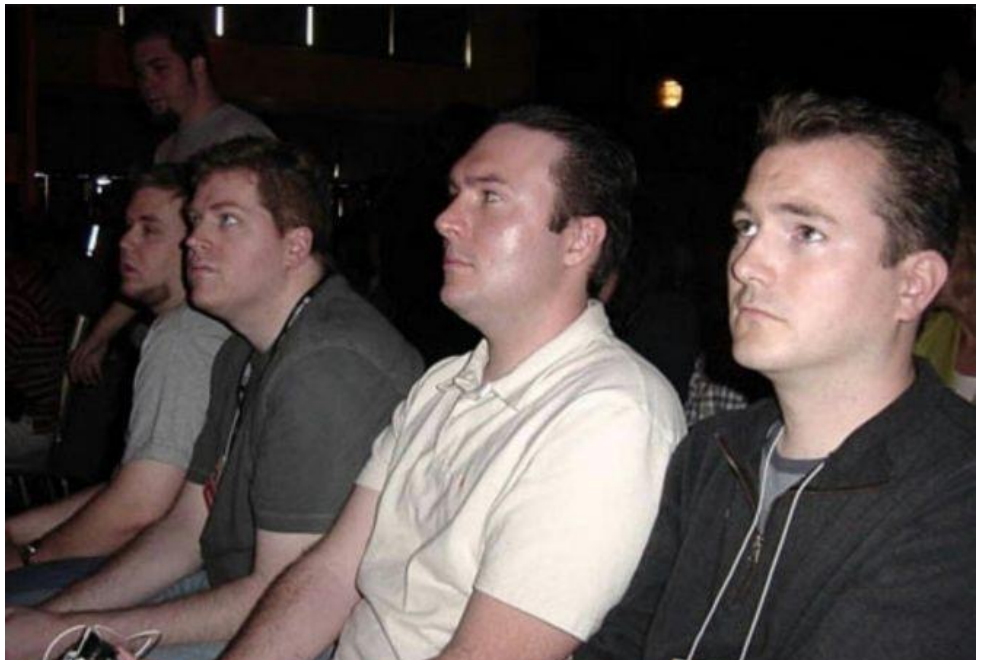
Where we left off

- Problems with RNNs!
  - Vanishing gradients

motivates

- Fancy RNN variants!
  - LSTM
  - GRU
  - multi-layer
  - bidirectional

People are still not happy!



Where we left off

- Problems with RNNs!
  - Vanishing gradients

motivates

- Fancy RNN variants!
  - LSTM
  - GRU
  - multi-layer
  - bidirectional

Transformer

Outline

- **Attention Models**
  - Why do we need “attentions”?
  - Definition of Attention Mechanism.

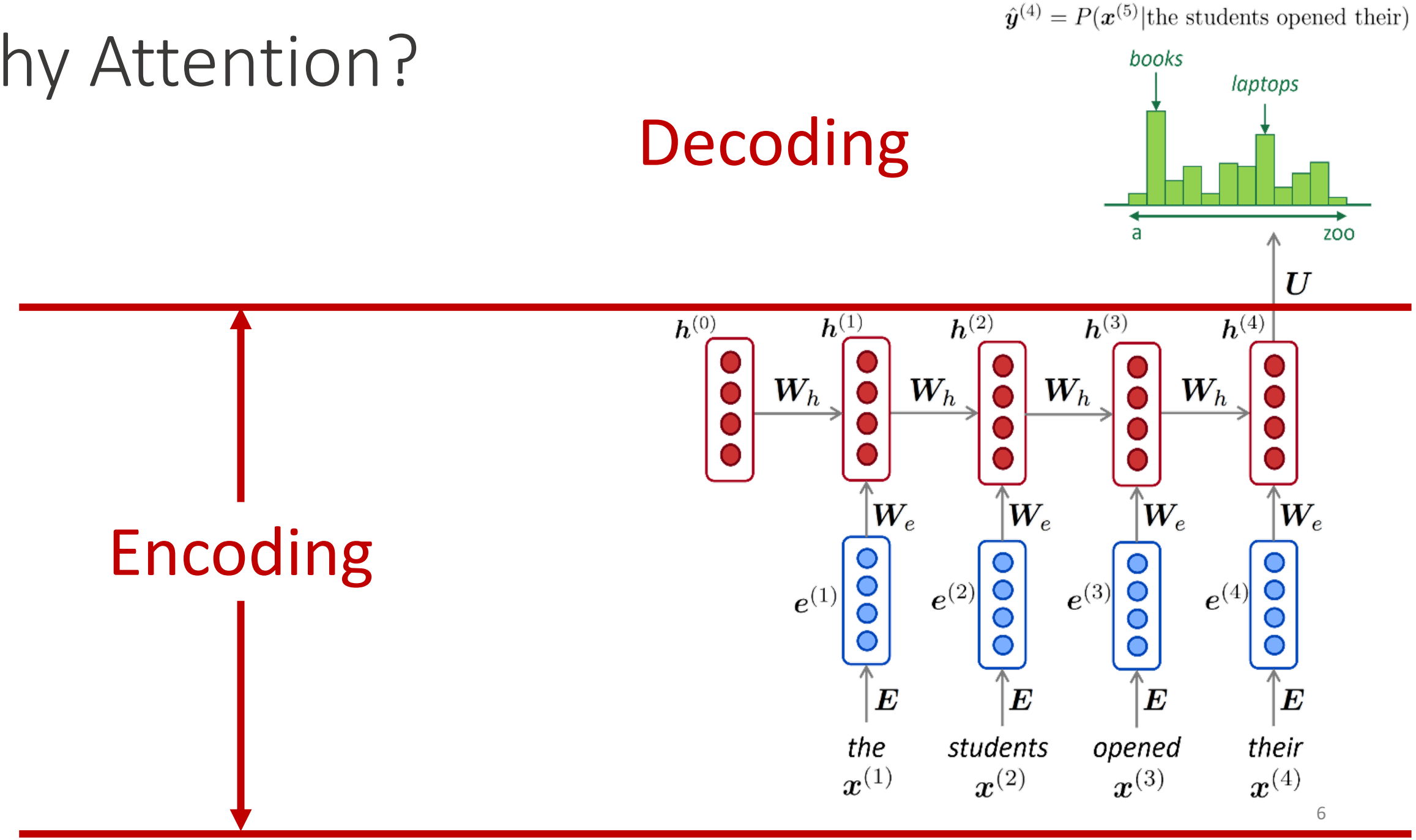
- **Self-Attention**

Outline

- **Attention Models**
  - Why do we need “attentions”?
  - Definition of Attention Mechanism.

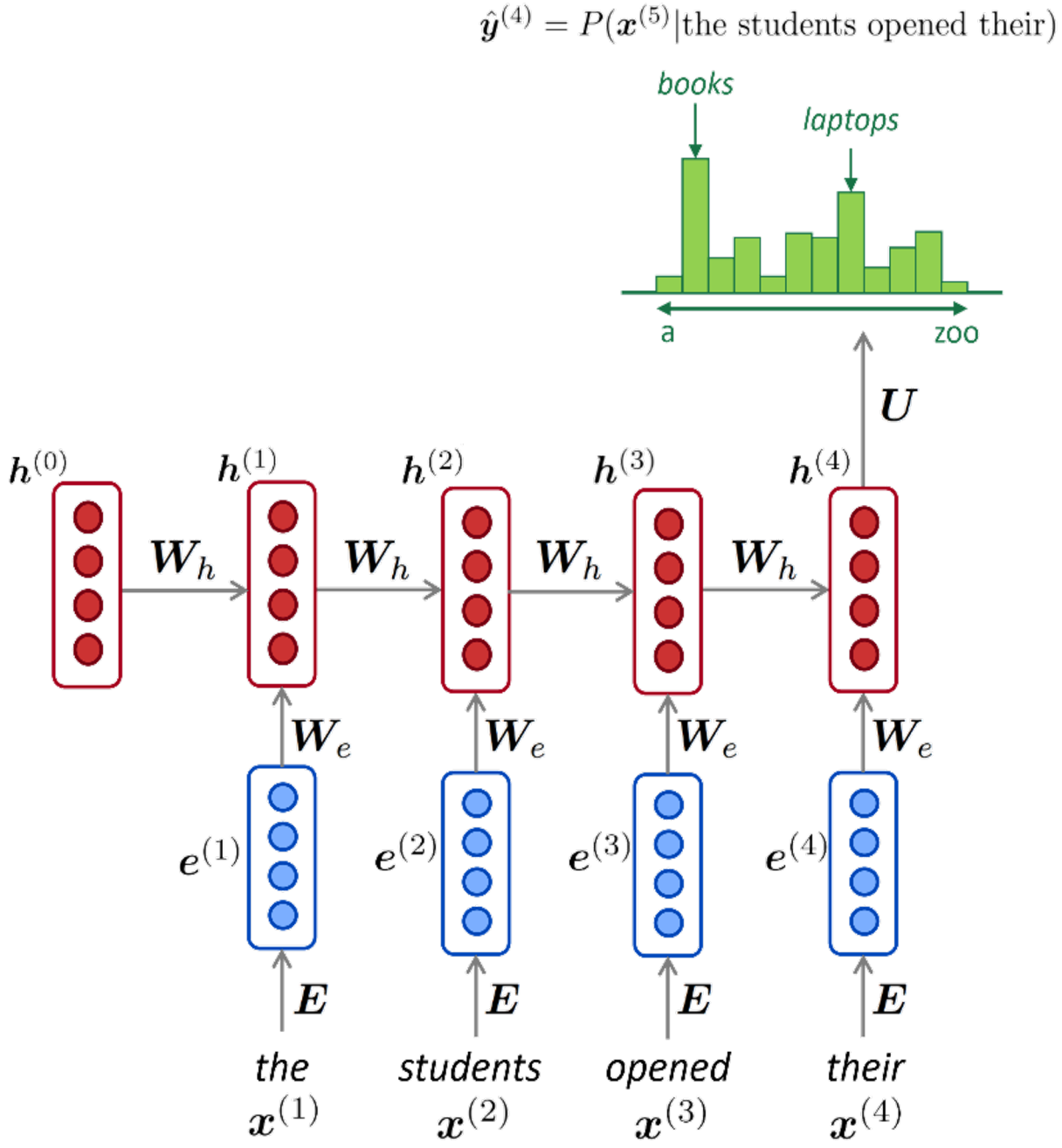
- **Self-Attention**

Why Attention?



Why Attention?

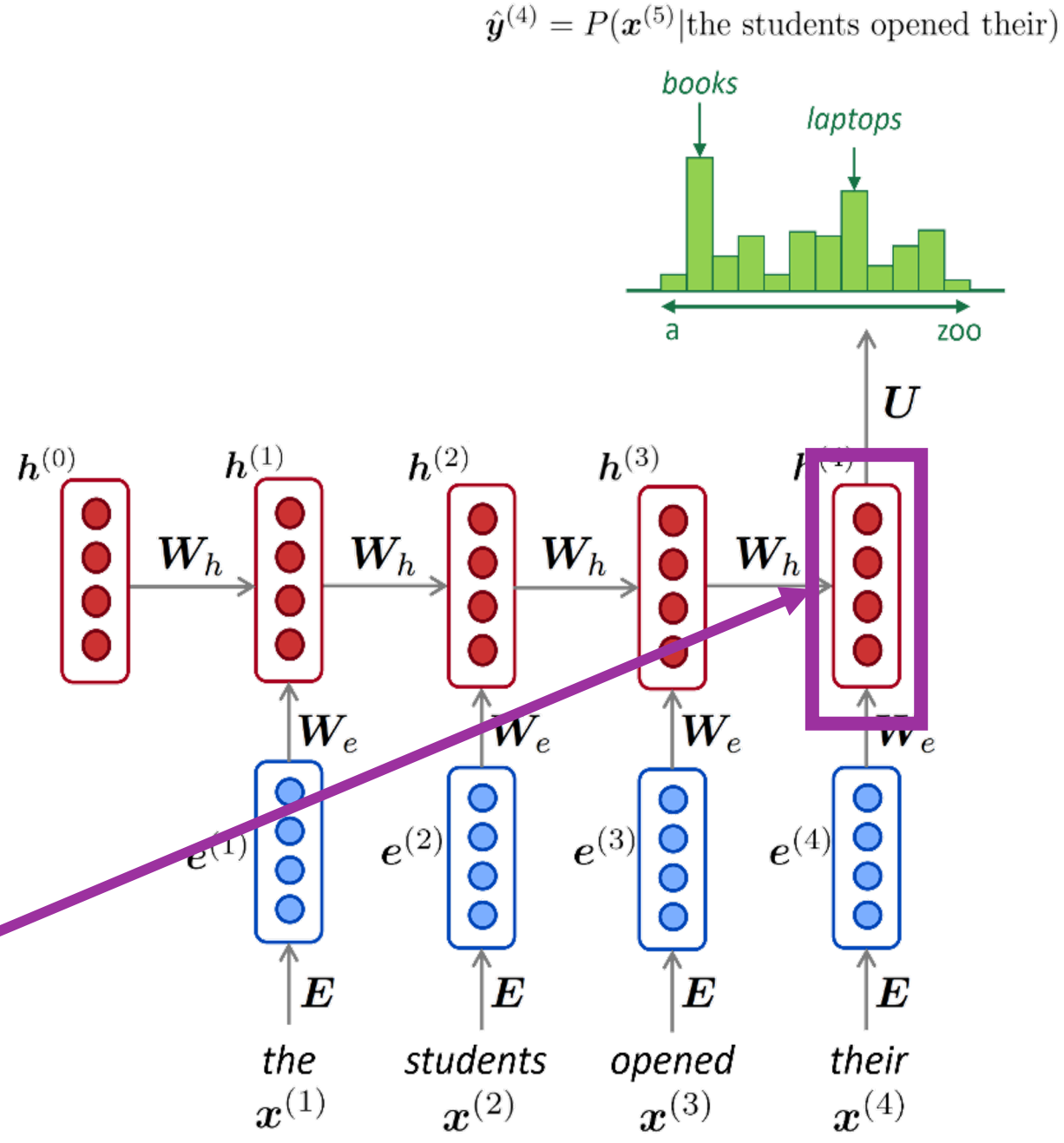
- Limitation of RNNs: Representation **bottleneck**.
- A single representation vector must encode all of the information about the text observed so far.



Why Attention?

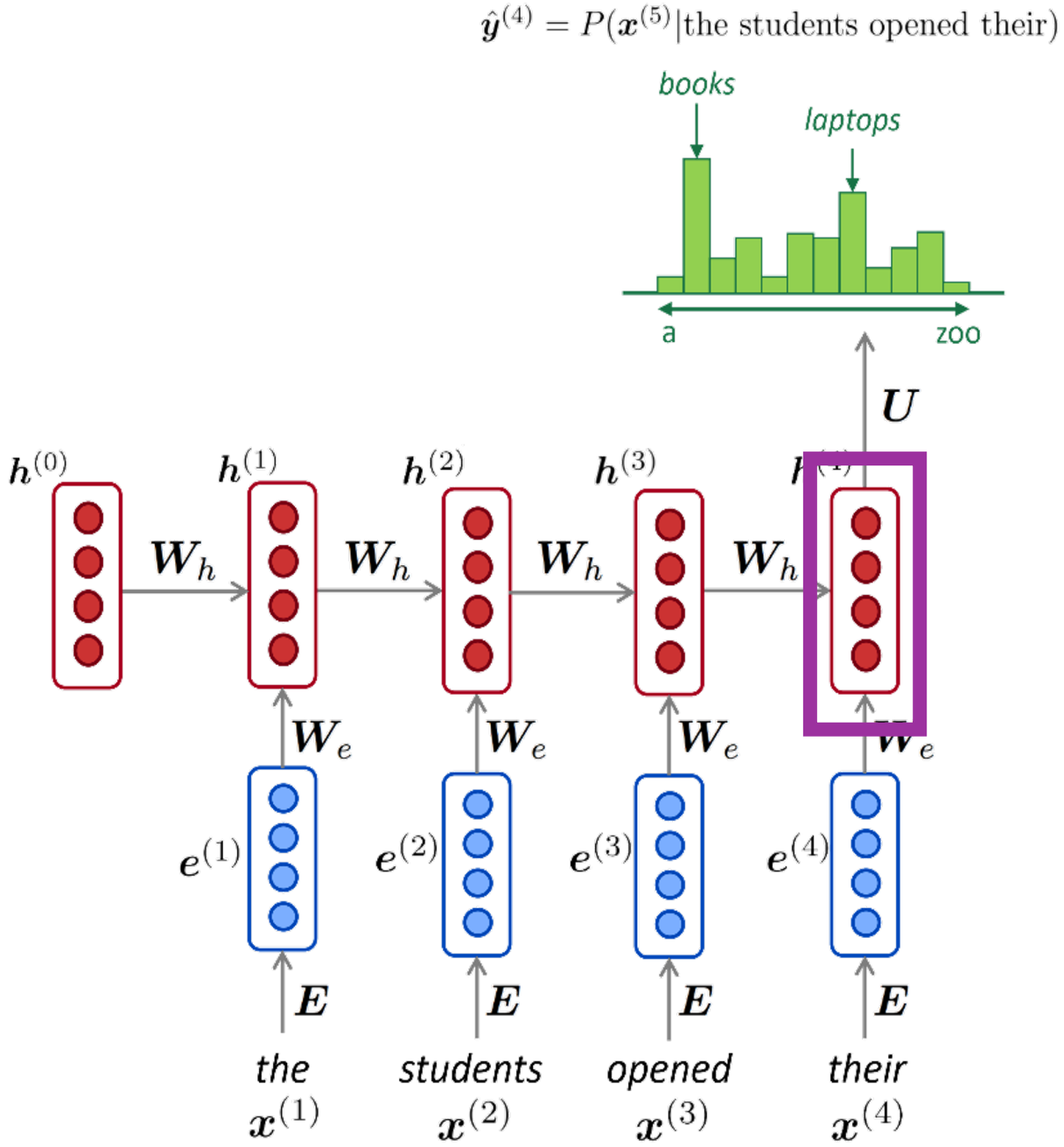
- Limitation of RNNs: Representation **bottleneck**.
- A single representation vector must encode all of the information about the text observed so far.

The representation of "the students opened their"



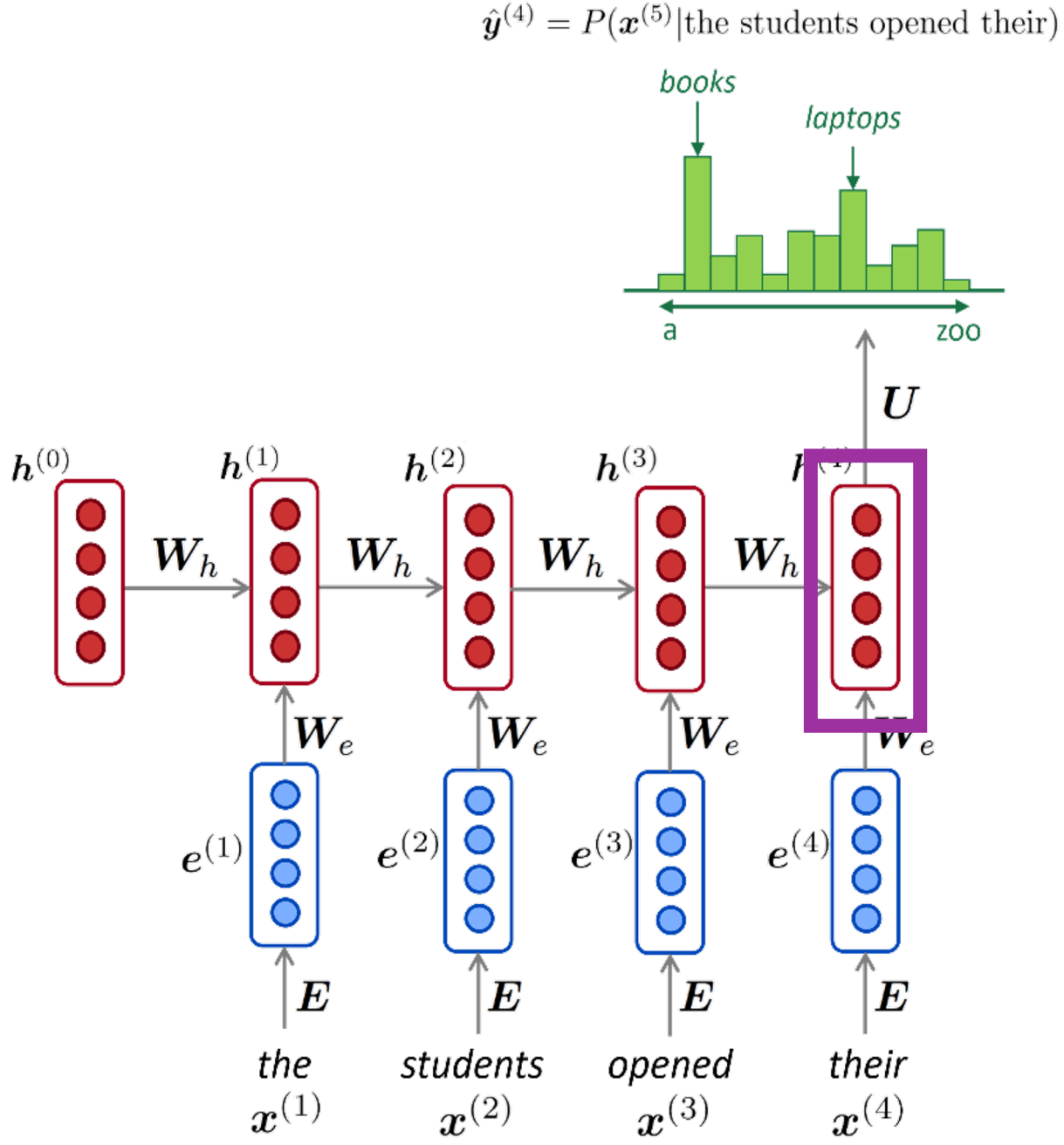
Why Attention?

- Limitation of RNNs: Representation **bottleneck**.
- A single representation vector must encode all of the information about the text observed so far.
- This is a problem **if the sentence is long**.



Why Attention?

- Limitation of RNNs: Representation **bottleneck**.
- A single representation vector must encode all of the information about the text observed so far.
- This is a problem **if the sentence is long**.

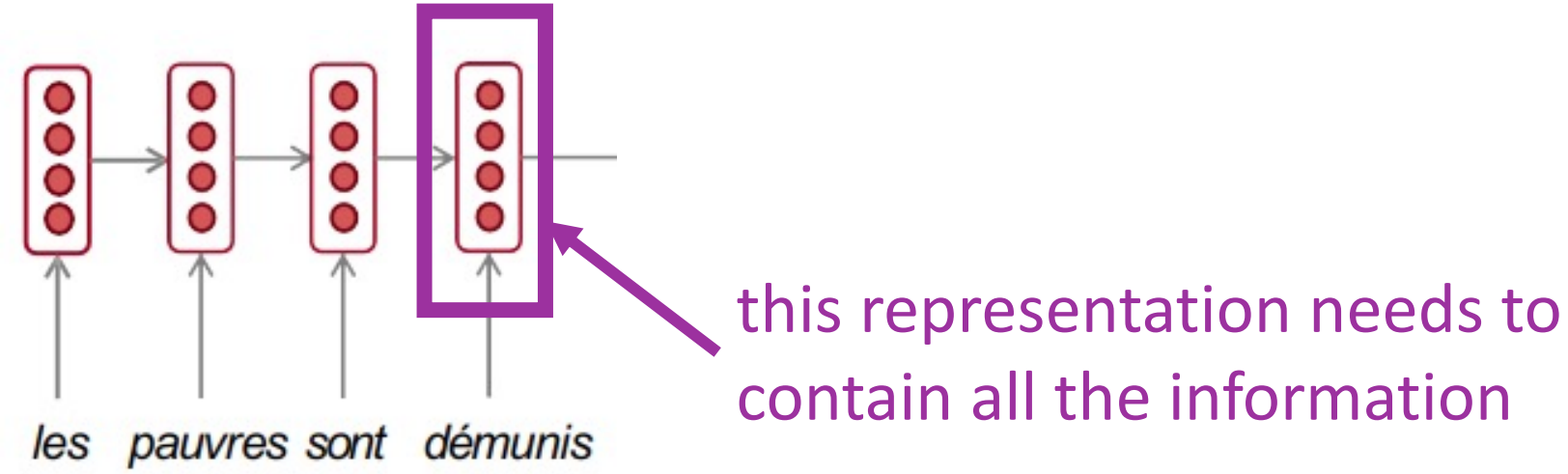


“you can’t cram the meaning of a whole %&@#&ing sentence into a single $*(&@ing vector!”

— Ray Mooney (NLP professor at UT Austin)

Attention Models

- Let’s consider a machine translation task. Previously,



Attention Models

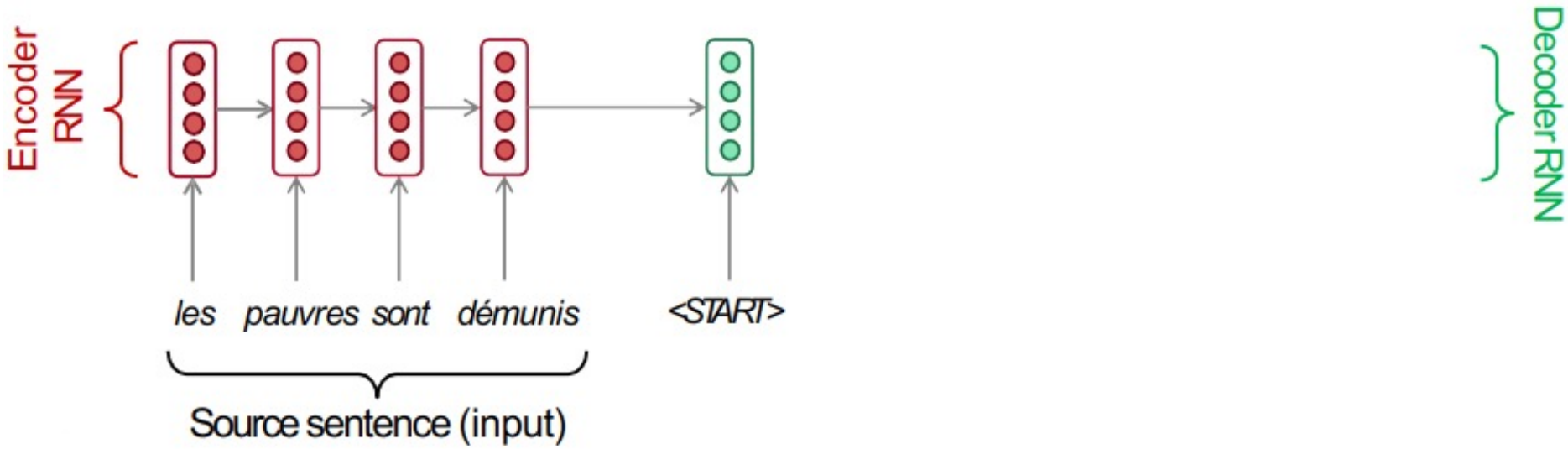
- Let's consider a machine translation task. Previously,

- Now, let's try this

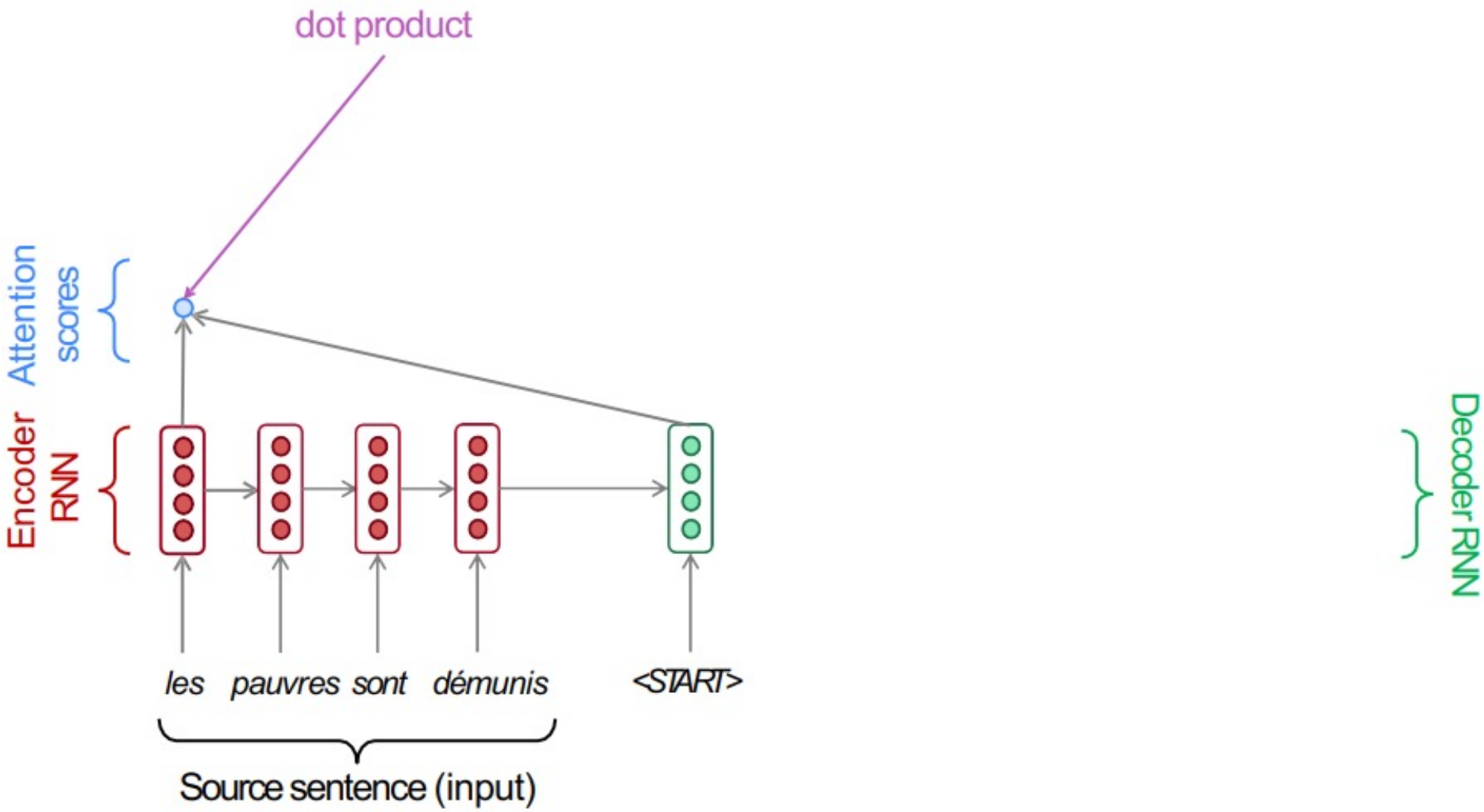
Attention Models

- **Attention mechanisms** (Bahdanau et al., 2015) allow language models to focus on a particular part of the observed context at each time step
  - Originally developed for machine translation, and intuitively similar to *word alignments* between different languages

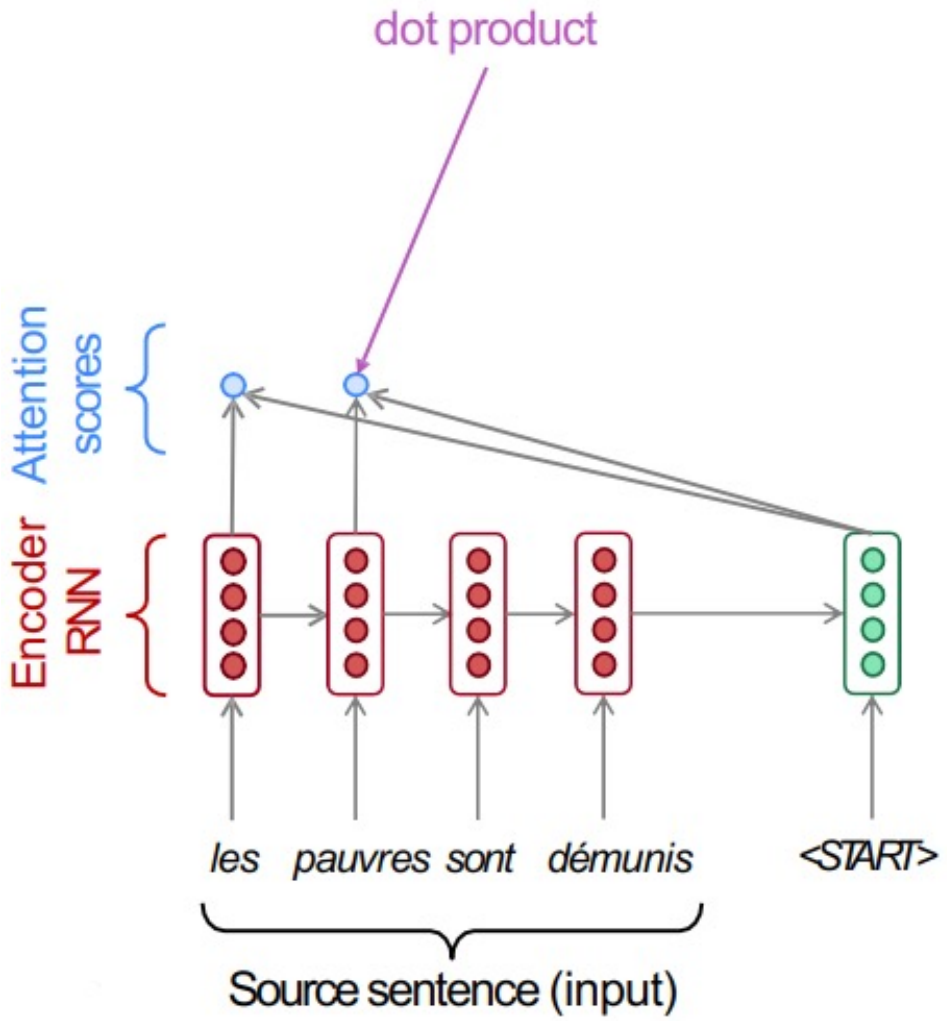
Attention Models



Attention Models

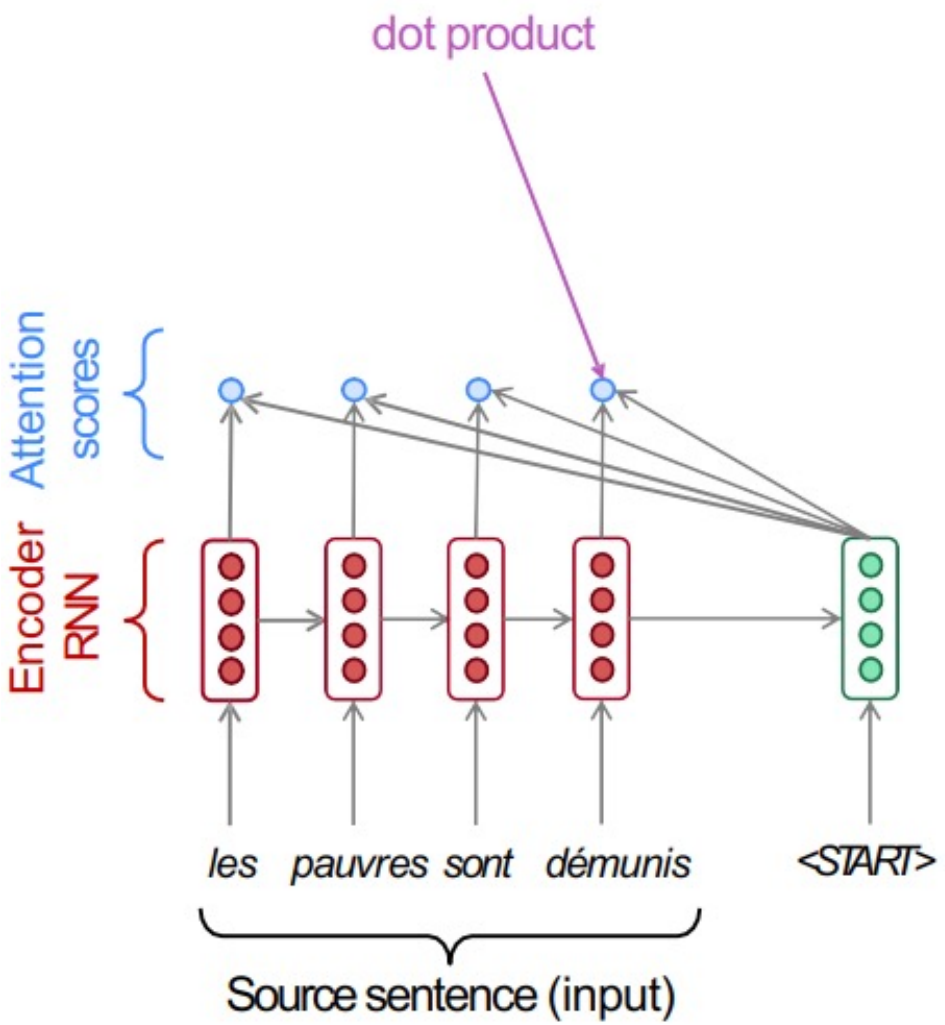


Attention Models

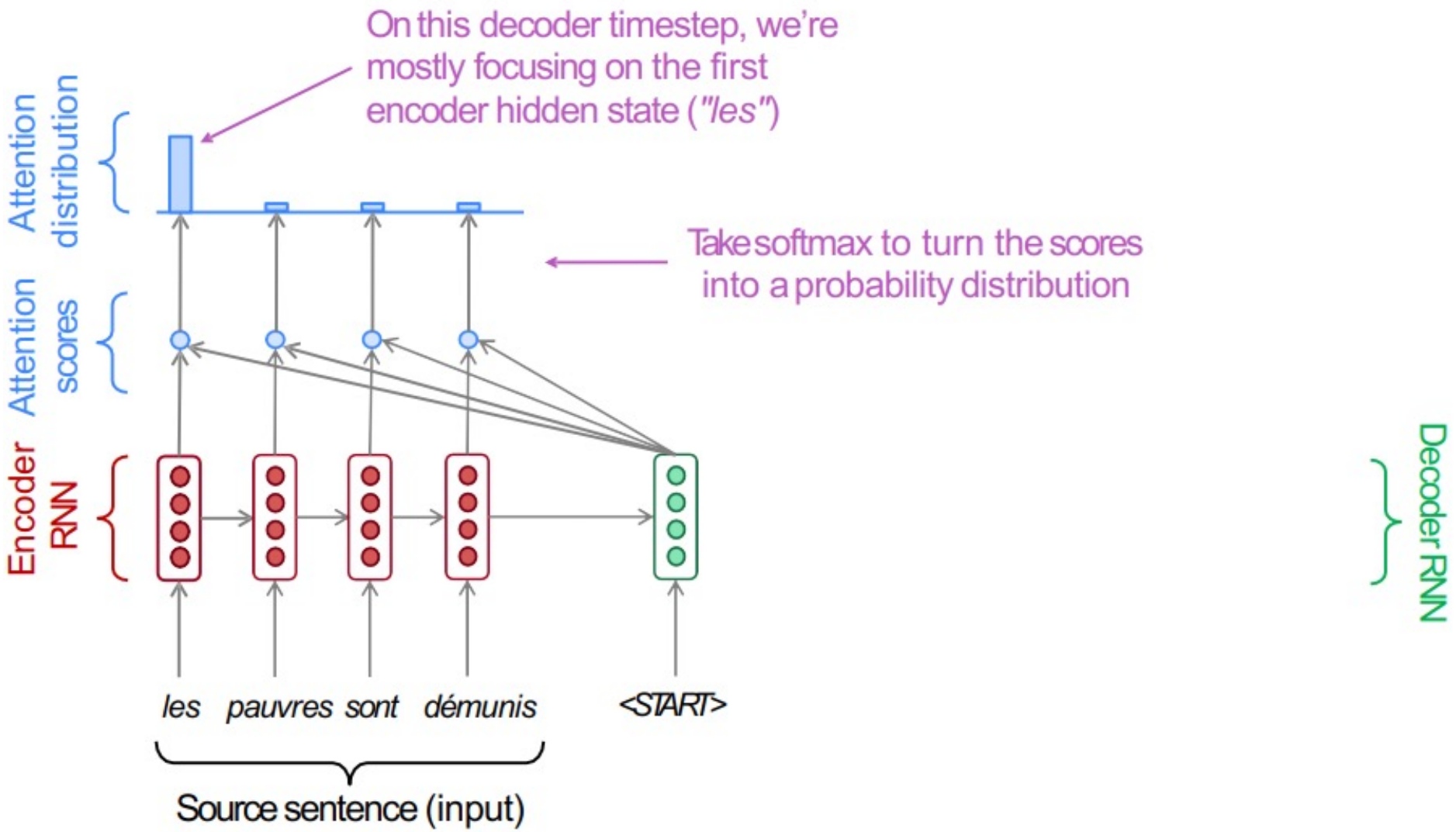


Decoder RNN

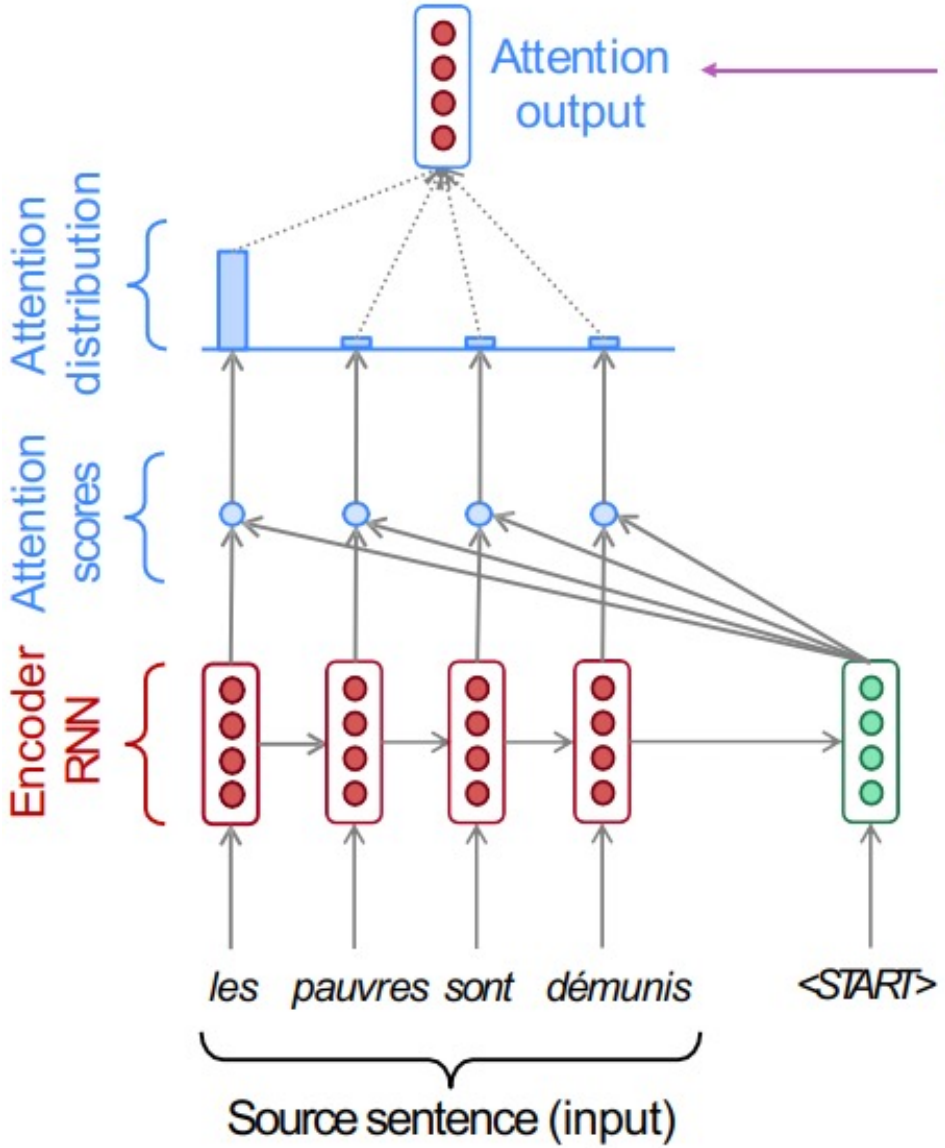
Attention Models



Attention Models



Attention Models

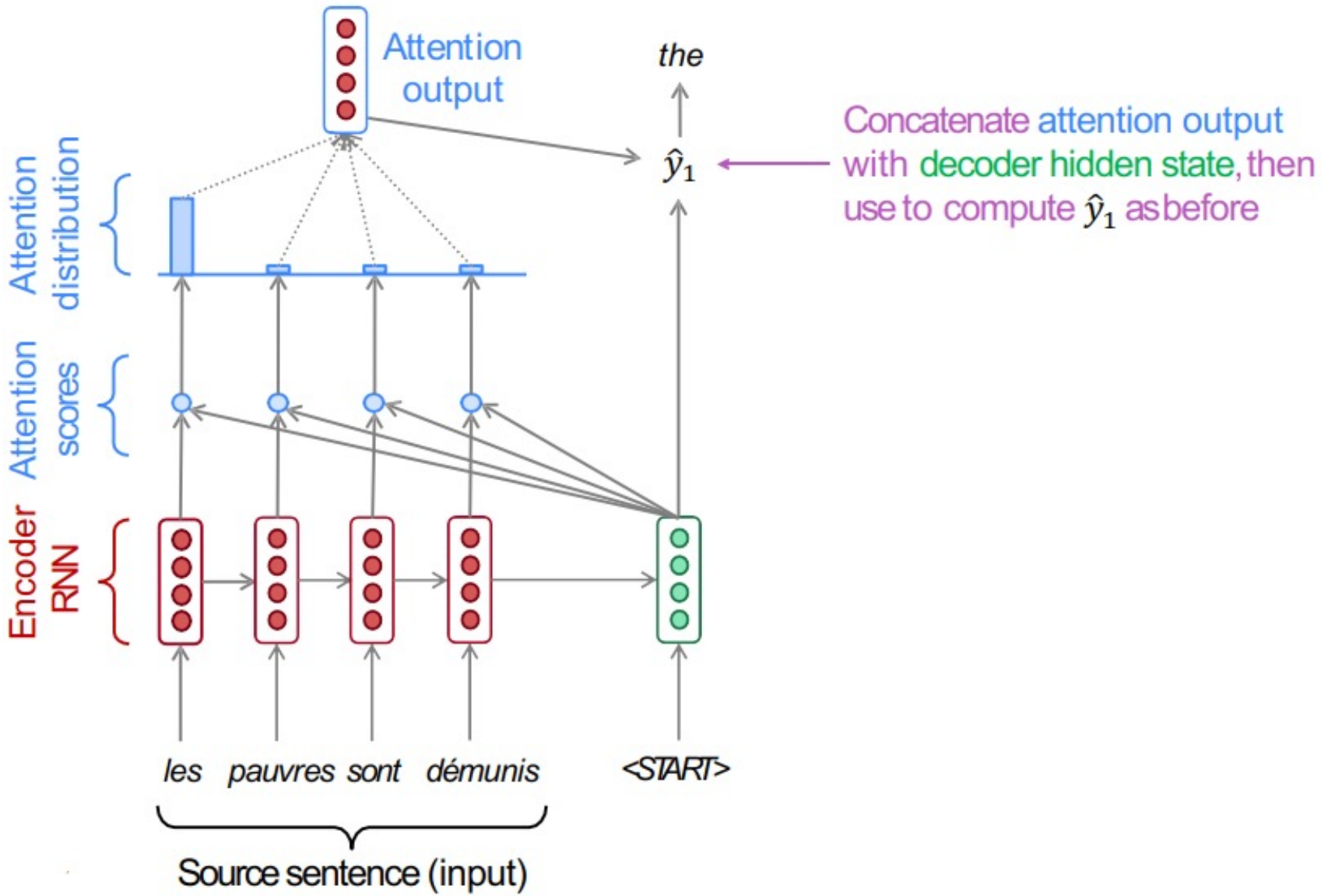


Use the attention distribution to take a **weighted sum** of the **encoder hidden states**.

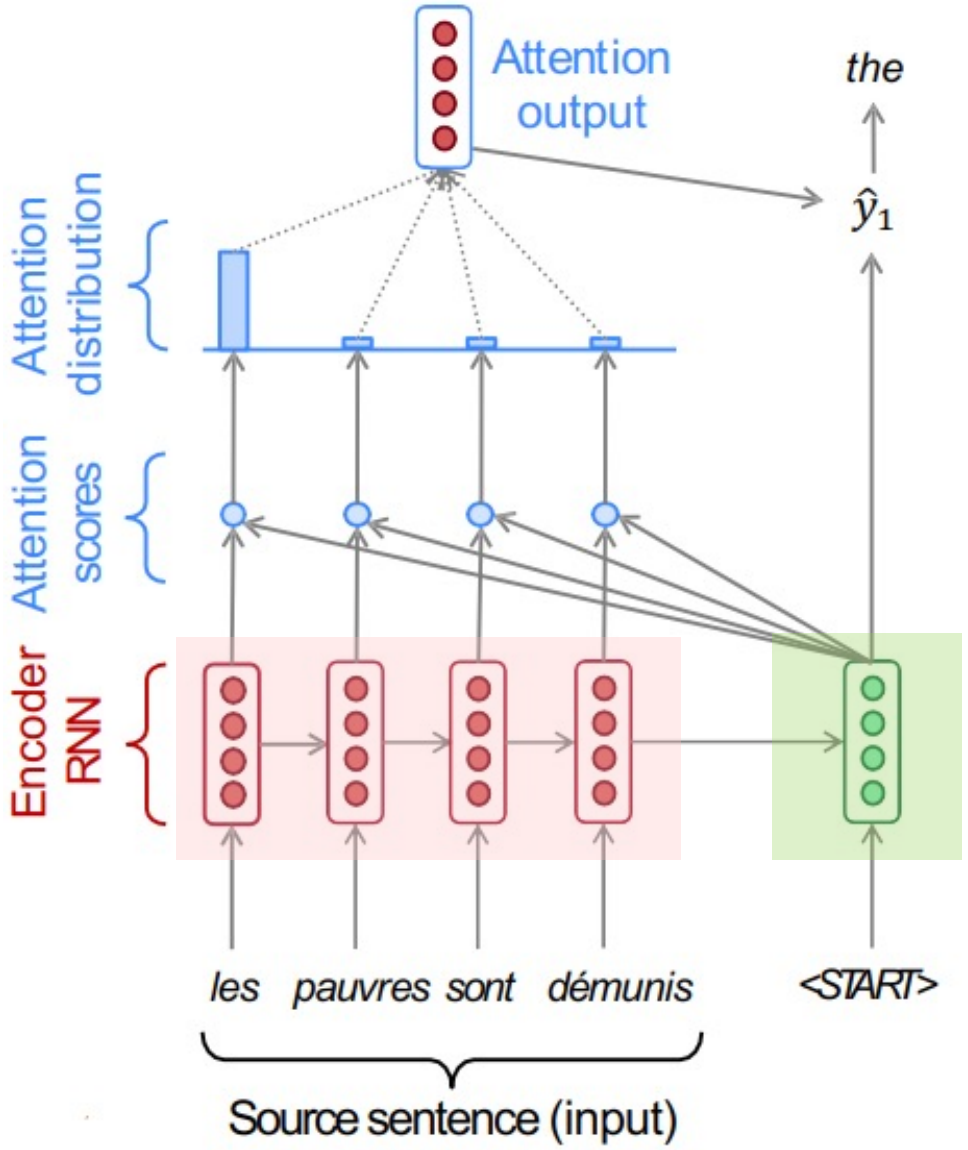
The attention output mostly contains information the **hidden states** that received high attention.

Decoder RNN

Attention Models

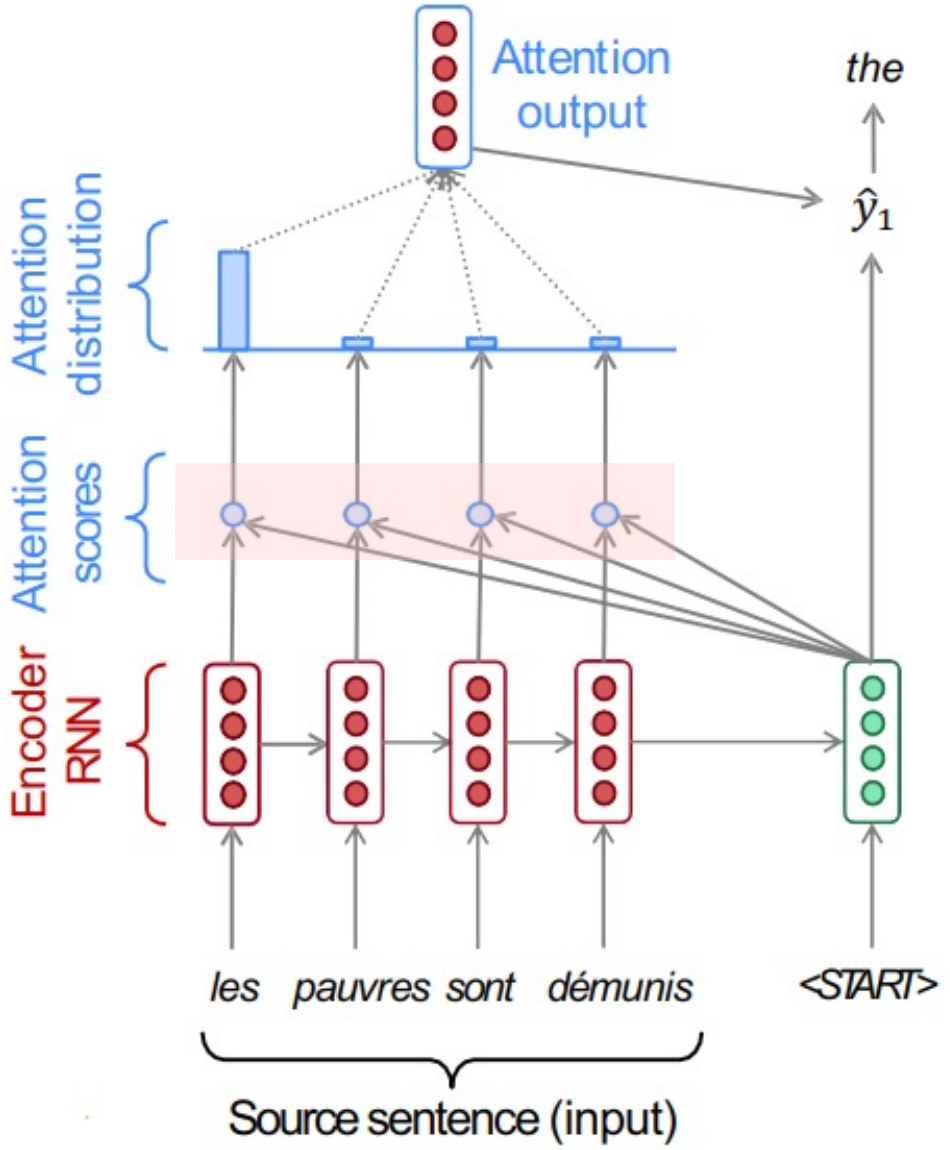


Attention Models



- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$

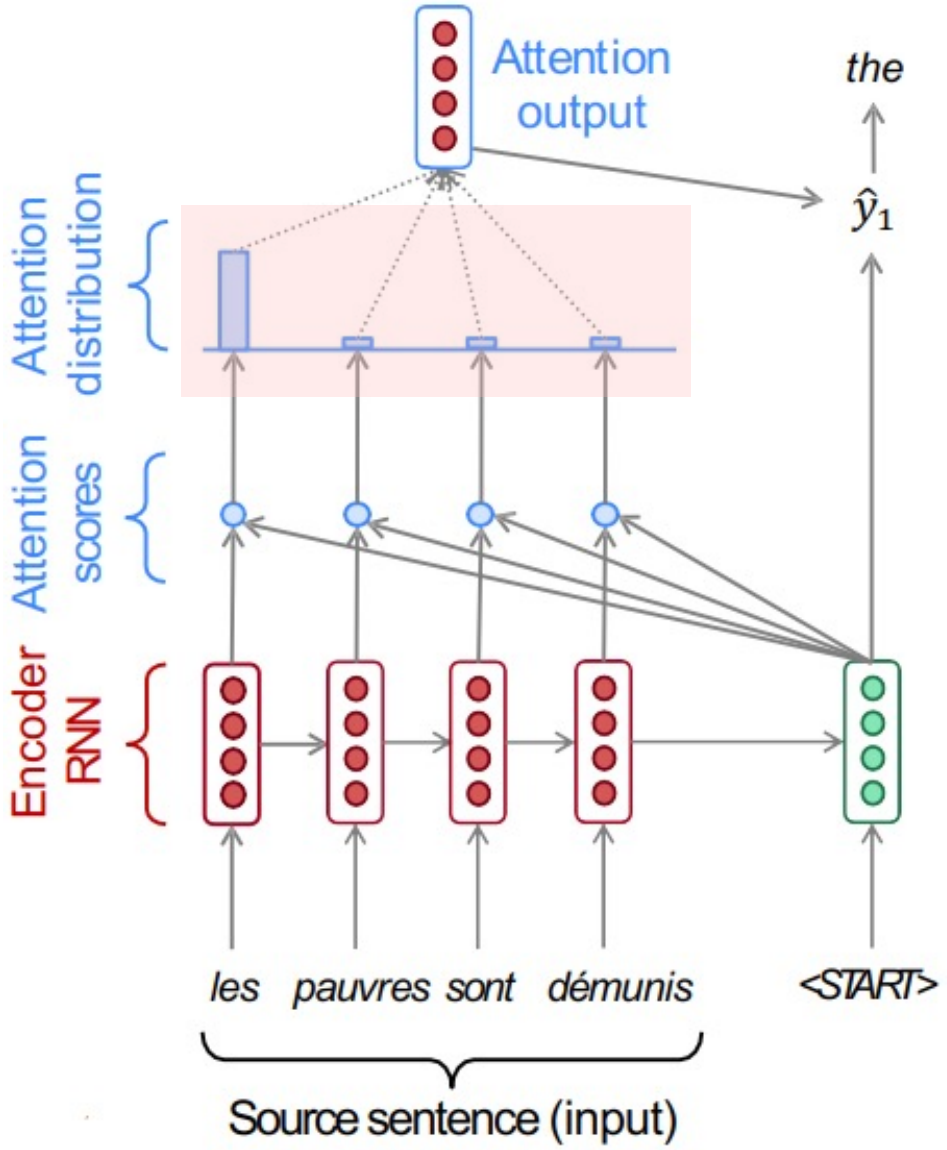
Attention Models



- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

Attention Models



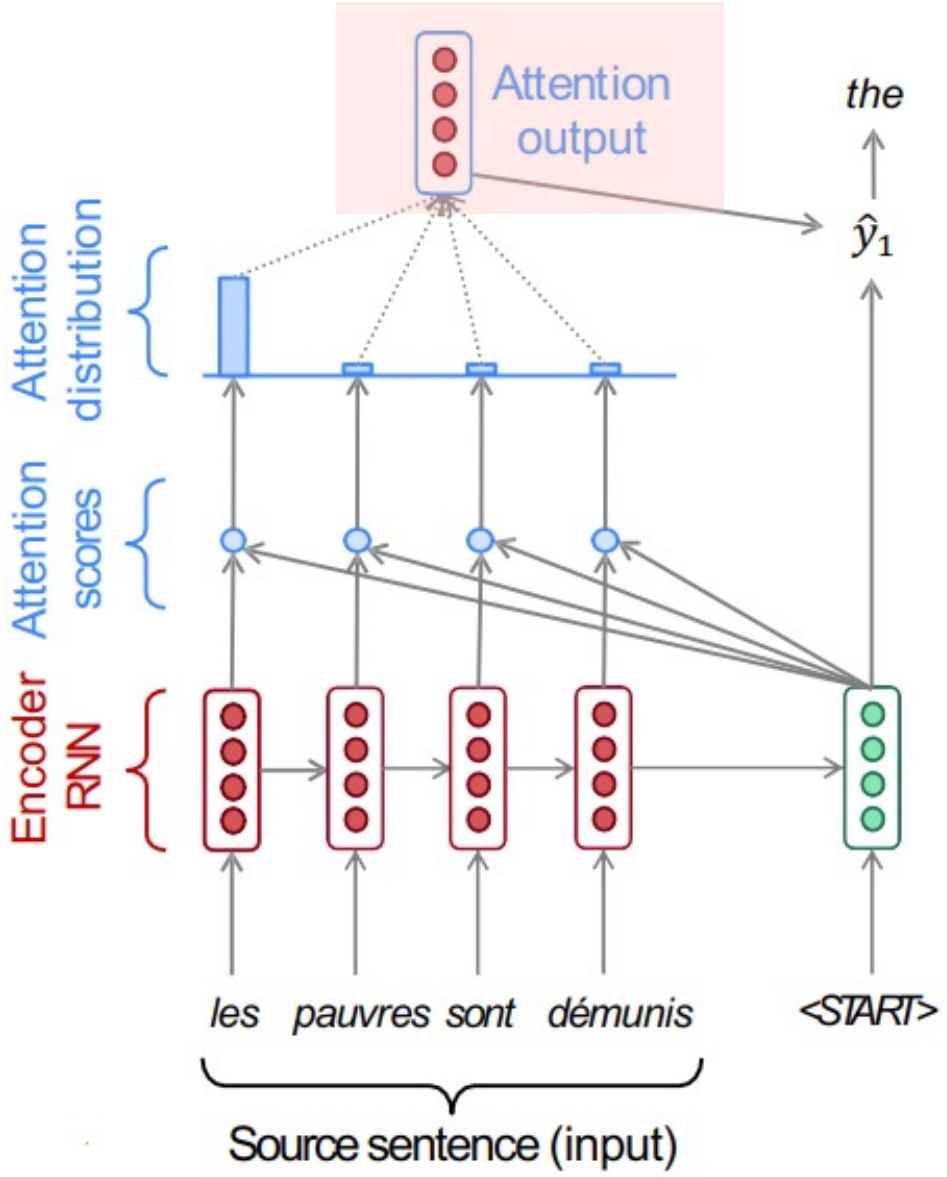
- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

Attention Models



- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

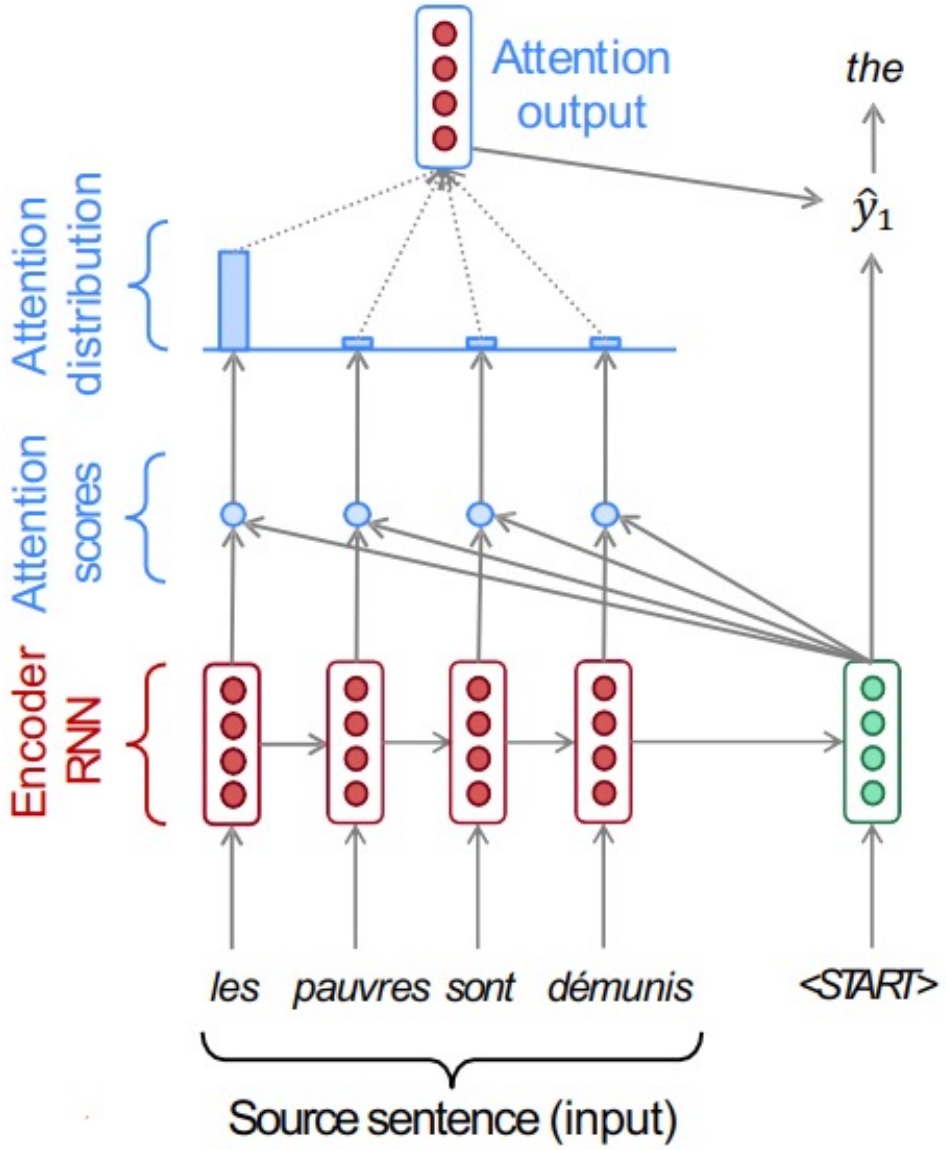
- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

Attention Models



- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

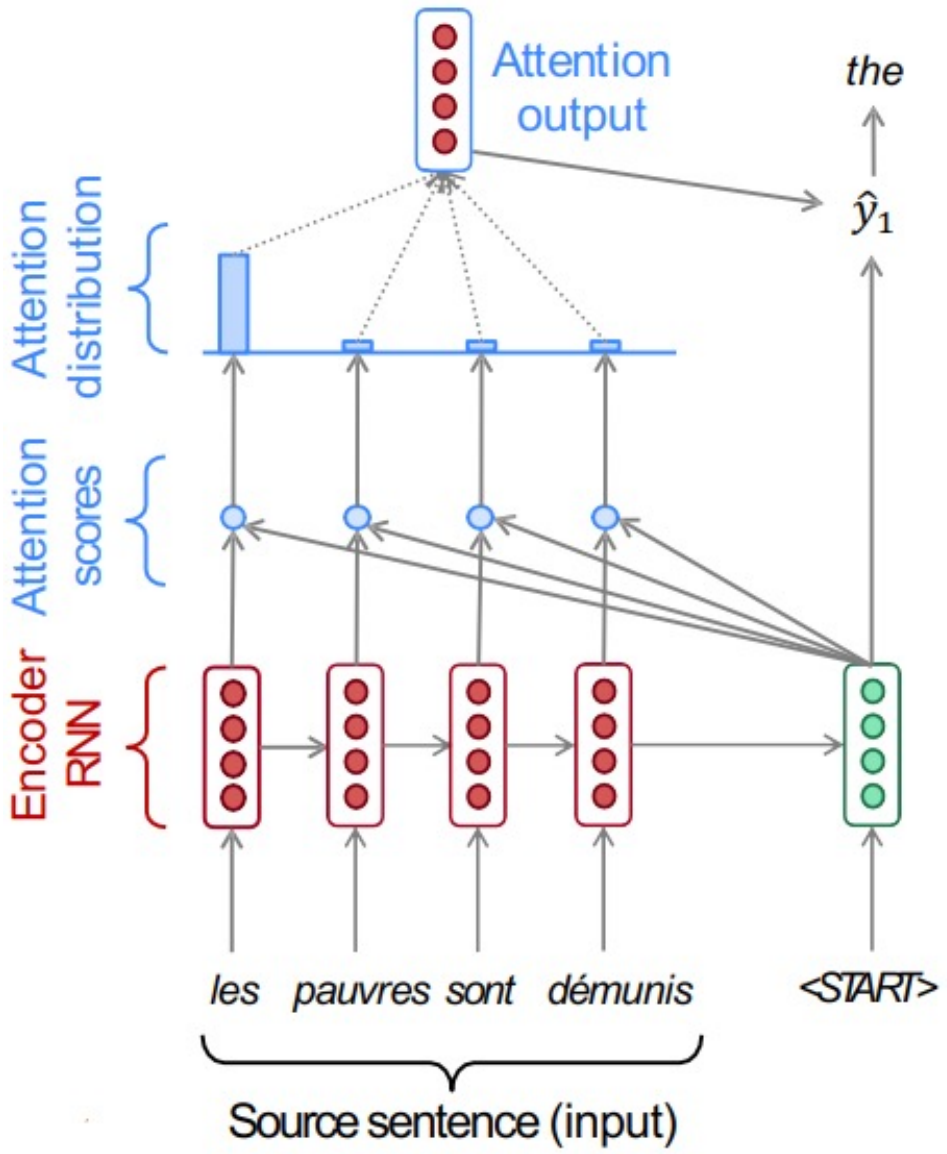
- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

Attention Models



Key vectors

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$ Query vector
- We get the attention scores e^t for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

Value vector

- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

Attention Models

- Three inputs:
  - Query vector(s)
  - Key vectors
  - Value vectors

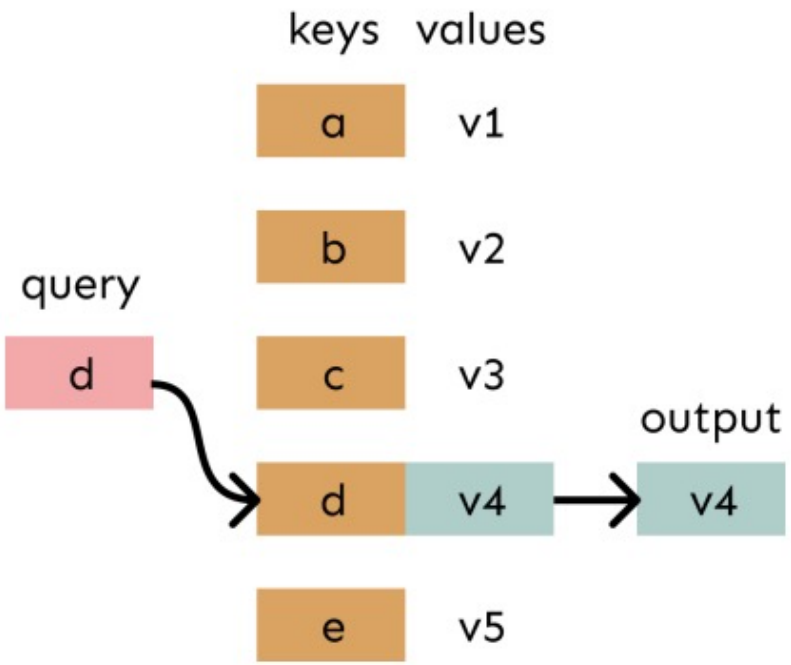
- Computation steps:
  - Compute attention scores
  - Normalize attention scores

- Output:
  - Sum of value vectors weighted by normalized attention scores

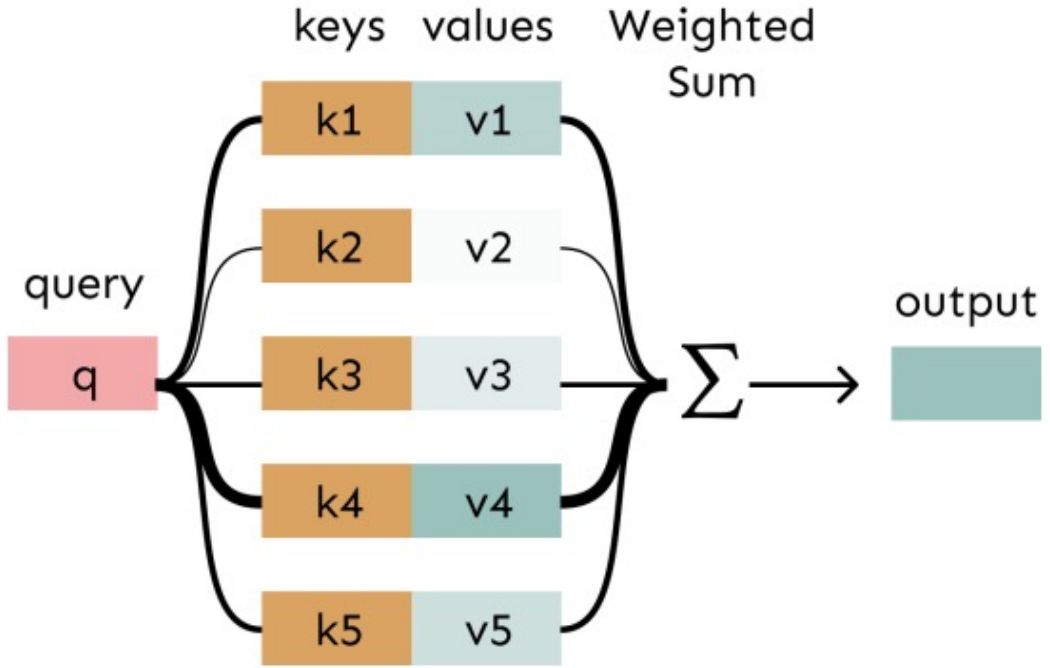
Attention Models

We can think of **attention** as performing 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.

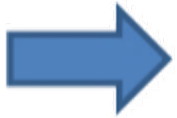


In **attention**, the query matches all keys *softly*, to a weight between 0 and 1. The keys' values are multiplied by the weights and summed.



Attention Models

- Three inputs:
  - Query vector(s)
  - Key vectors
  - Value vectors



- Three input matrices:
  - $Q \in \mathbb{R}^{d_q \times n_q}$
  - $K \in \mathbb{R}^{d_k \times n_k}$
  - $V \in \mathbb{R}^{d_v \times n_v}$

Constraints:

$$\boldsymbol{d_q = d_k}$$

$$\boldsymbol{n_v = n_k}$$

- Computation steps:
  - Compute attention scores: $\boldsymbol{A = Q^T K} \in \mathbb{R}^{\boldsymbol{n_q \times n_k}}$
  - Normalize attention scores: $\boldsymbol{A = Softmax(A)} \in \mathbb{R}^{\boldsymbol{n_q \times n_k}}$

- Output:
  - Sum of value vectors weighted by normalized attention scores:

$$\boldsymbol{Output = V \cdot A^T} \in \mathbb{R}^{\boldsymbol{d_v \times n_q}}$$

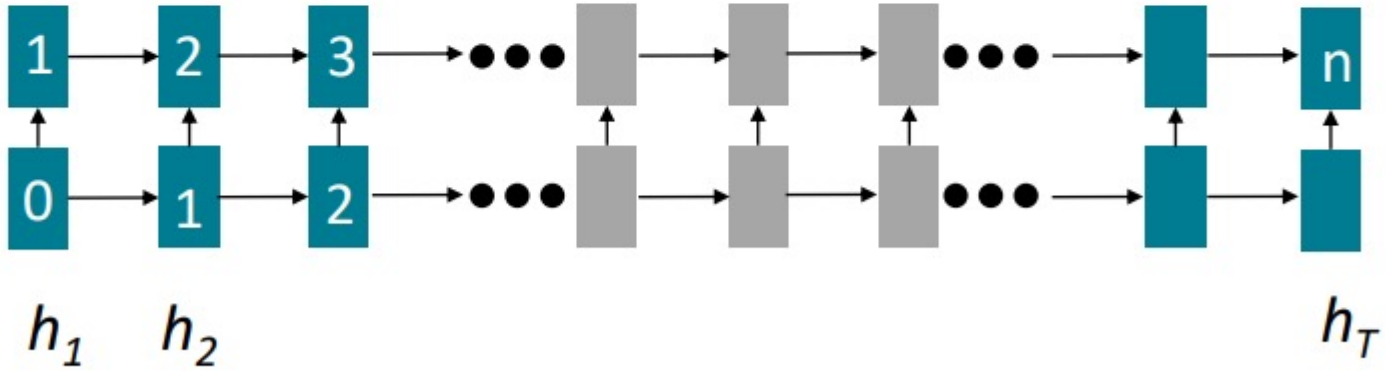
Outline

- **Attention Models**
  - Why do we need “attentions”?
  - Definition of Attention Mechanism.

- **Self-Attention**

Self-Attention

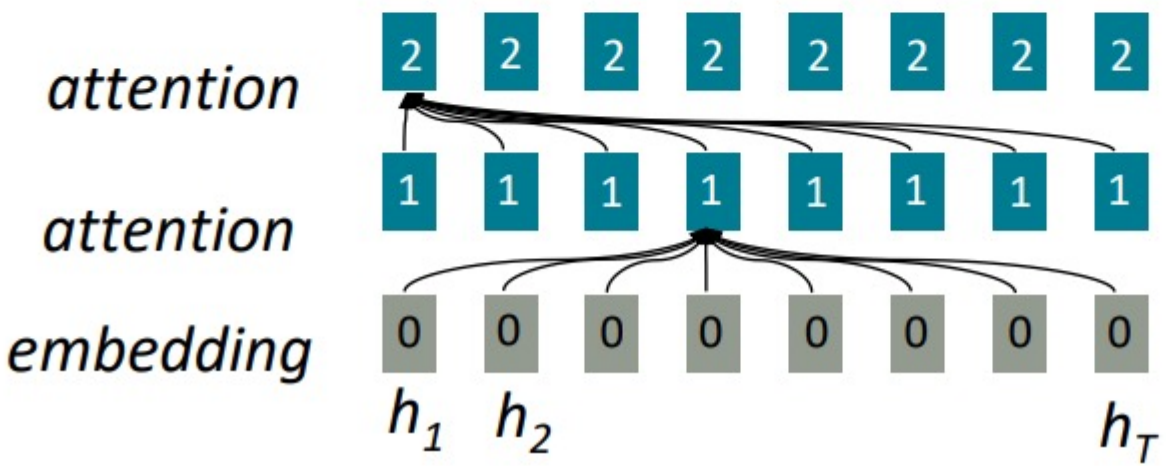
- Forward and backward passes have **O(sequence length)** unparallelizable operations
  - GPUs can perform a bunch of independent computations at once!
  - But future RNN hidden states can't be computed in full before past RNN hidden states have been computed
  - Inhibits training on very large datasets!



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

Self-Attention

- **Attention** treats each word's representation as a **query** to access and incorporate information from **a set of values.**
  - We saw attention from the **decoder** to the **encoder**; today we'll think about attention **within a single sentence**.
- Number of unparallelizable operations does not increase with sequence length.
- Maximum interaction distance: O(1), since all words interact at every layer!

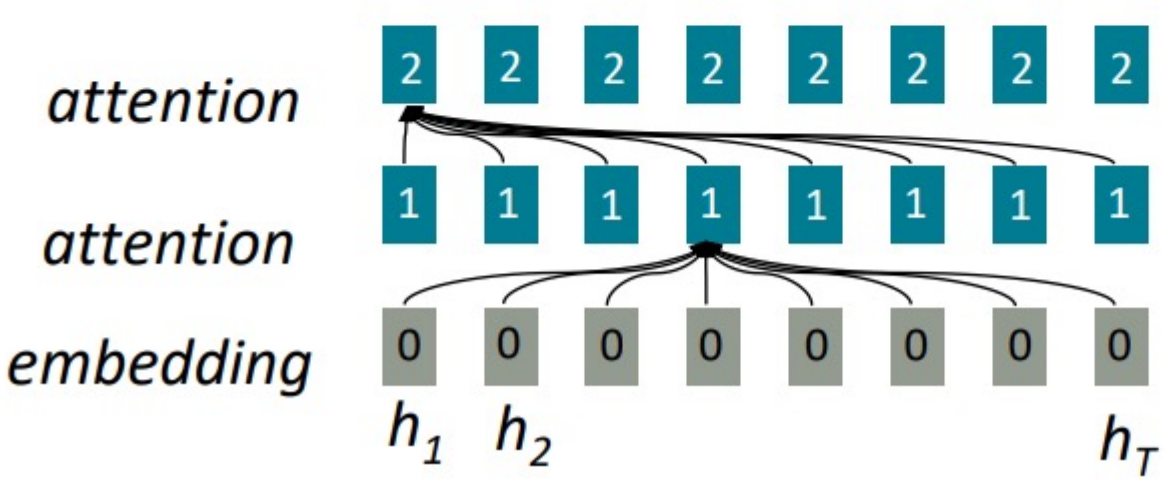


All words attend to all words in previous layer; most arrows here are omitted

Self-Attention

Self-Attention:

- Words attend to themselves.
- A word plays the role of query, key, and value simultaneously.



All words attend to all words in previous layer; most arrows here are omitted

Self-Attention

Let $\boldsymbol{w}_{1:n}$ be a sequence of words in vocabulary V, like *Zuko made his uncle tea*.

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

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

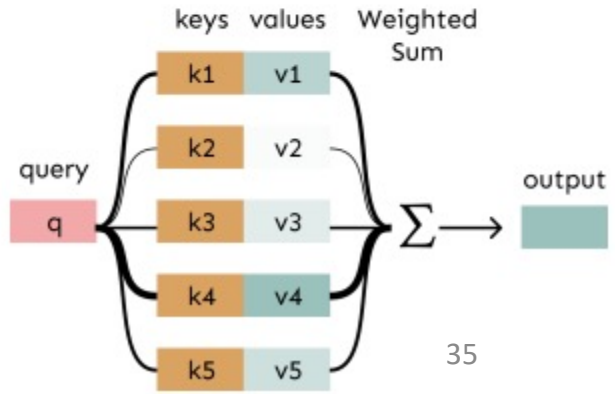
$\boldsymbol{q}_i = Q\boldsymbol{x}_i$ (queries) $\quad \boldsymbol{k}_i = K\boldsymbol{x}_i$ (keys) $\quad \boldsymbol{v}_i = V\boldsymbol{x}_i$ (values)

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

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



Self-Attention – Order Information

Barriers and solutions for Self-Attention as a building block

Barriers

- Doesn't have an inherent notion of order!

Self-Attention – 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.
- Consider representing each **sequence index** as a **vector**

$$\boldsymbol{p}_i \in \mathbb{R}^d, \text{ for } i \in \{1,2,\dots,n\} \text{ are position vectors}$$

- Don't worry about what the p_i are made of yet!
- Easy to incorporate this info into our self-attention block: just add the $\boldsymbol{p}_i$ to our inputs!
- Recall that $\boldsymbol{x}_i$ is the embedding of the word at index i. The positioned embedding is:

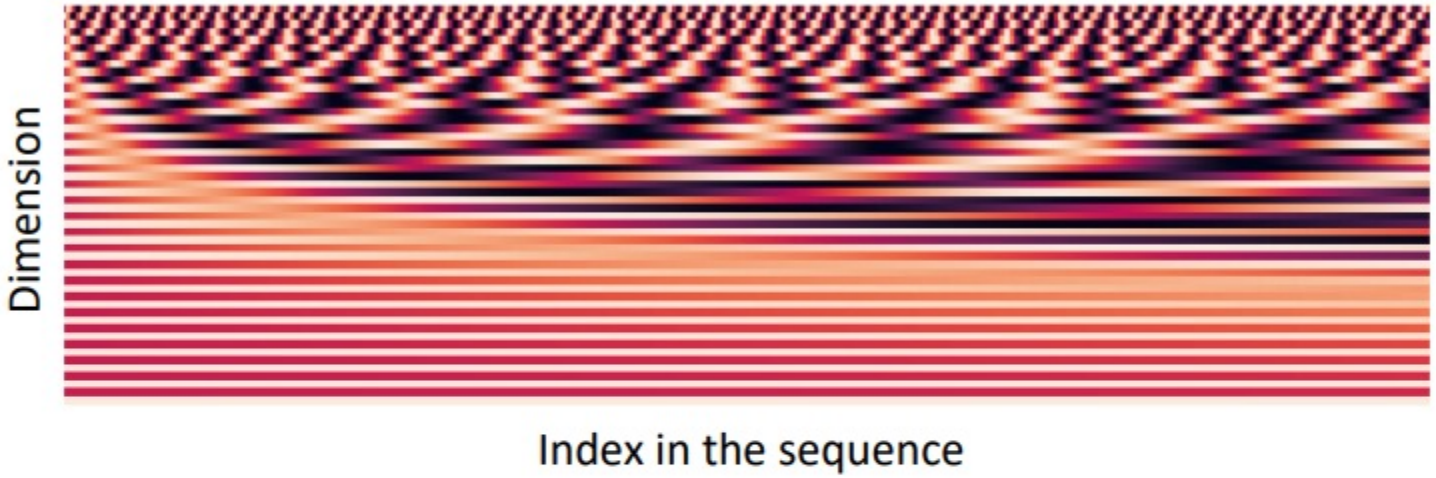
$$\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i + \boldsymbol{p}_i$$

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

Self-Attention – Order Information

- **Sinusoidal position representations:** concatenate sinusoidal functions of varying periods:

$$\boldsymbol{p}_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



- Pros:
  - Periodicity indicates that maybe "absolute position" isn't as important
  - Maybe can extrapolate to longer sequences as periods restart!
- Cons:
  - Not learnable; also the extrapolation doesn't really work!

Self-Attention – Order Information

- **Learned absolute position representations:** Let all p_i be learnable parameters!

  Learn a matrix $\boldsymbol{p} \in \mathbb{R}^{d \times n}$, *and let each* $\boldsymbol{p}_i$ *be a column of that matrix!*

- Pros:
  - Flexibility: each position gets to be learned to fit the data
- Cons:
  - Definitely can't extrapolate to indices outside $1, \ldots, n$.
- Most systems use this!

- Sometimes people try more flexible representations of position:

  "Self-Attention with Relative Position Representations"
  "Self-Attention with Structural Position Representations"

Self-Attention – Order Information

Barriers and solutions for Self-Attention as a building block

Barriers		Solutions
• Doesn't have an inherent notion of order!	→	• Add position representations to the inputs

Self-Attention

Barriers and solutions for Self-Attention as a building block

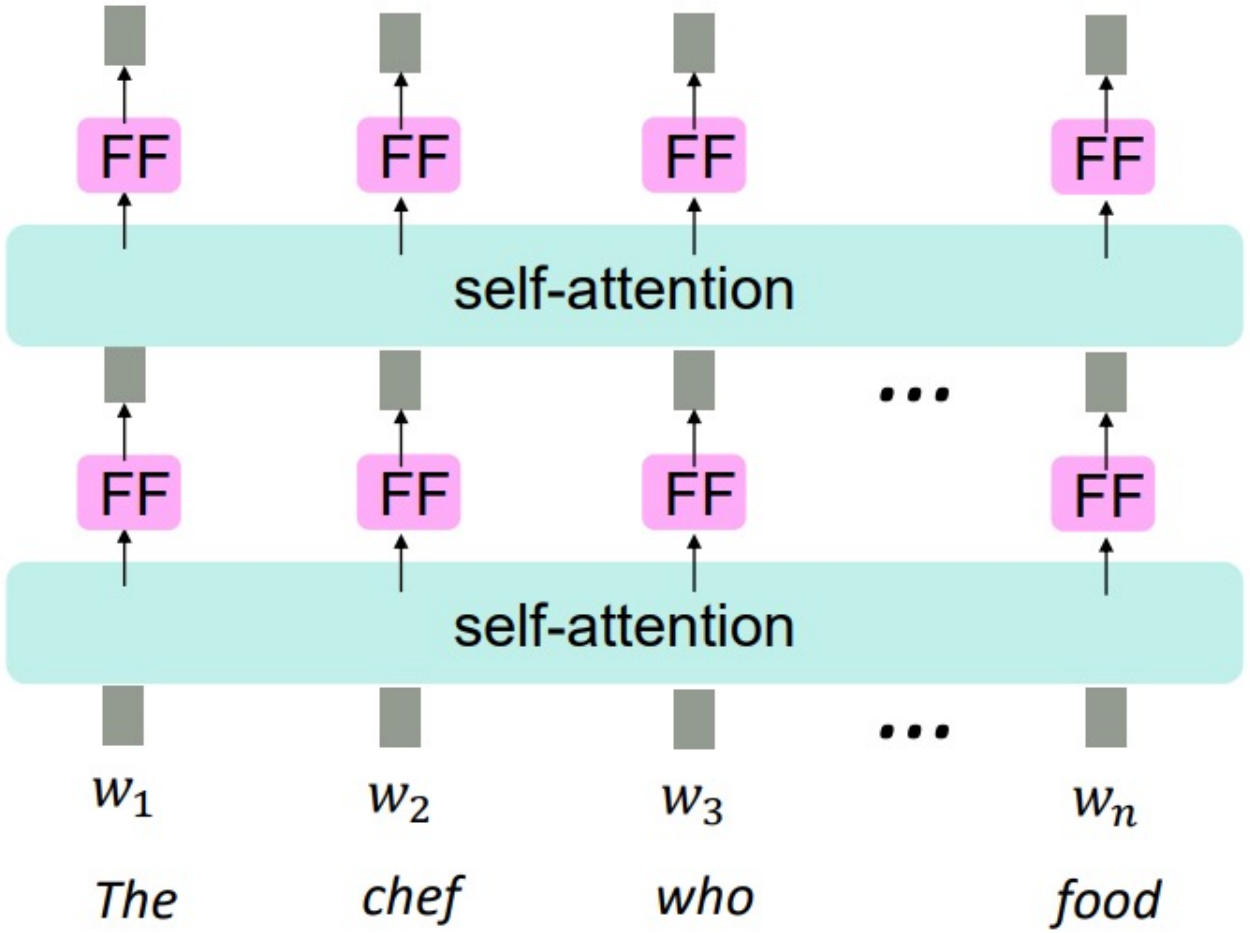
Barriers		Solutions
• Doesn't have an inherent notion of order!	→	• Add position representations to the inputs
• No nonlinearities for deep learning magic! It's all just weighted averages	→	

Self-Attention - Nonlinearity

- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages **value** vectors (Why? Look at the notes!)

- Easy fix: add a **feed-forward network** to post-process each output vector.

$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1\ \text{output}_i + b_1) + b_2$$



Intuition: the FF network processes the result of attention

Self-Attention - Nonlinearity

Barriers and solutions for Self-Attention as a building block

Barriers		Solutions
• Doesn't have an inherent notion of order!	→	• Add position representations to the inputs
• No nonlinearities for deep learning magic! It's all just weighted averages	→	• Easy fix: apply the same feedforward network to each self-attention output.

Self-Attention

Barriers and solutions for Self-Attention as a building block

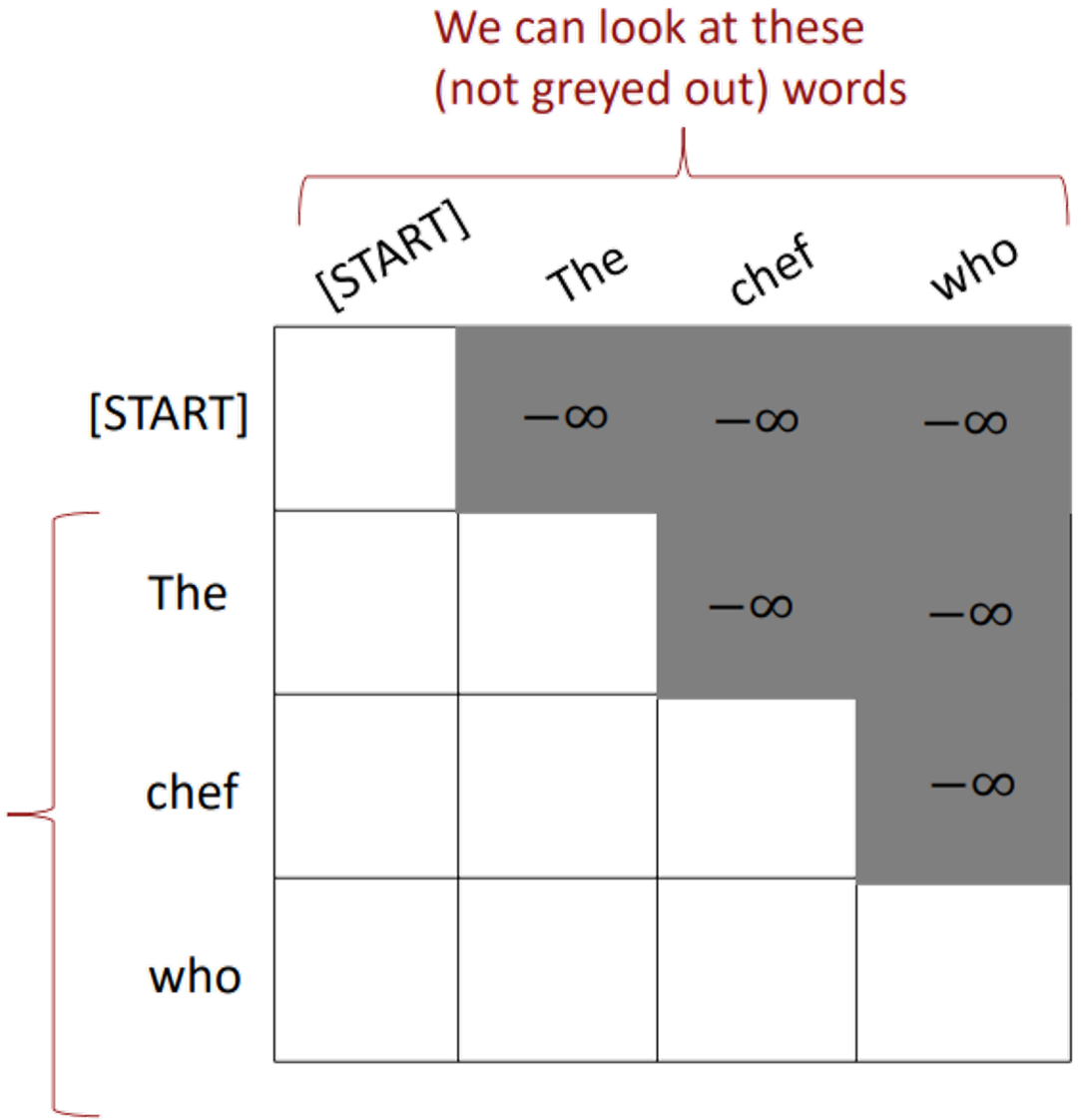
Barriers	Solutions
• Doesn't have an inherent notion of order!	• Add position representations to the inputs
• No nonlinearities for deep learning magic! It's all just weighted averages	• Easy fix: apply the same feedforward network to each self-attention output.

• Need to ensure we don't "look at the future" when predicting a sequence
  • Like in machine translation
  • Or language modeling	

Self-Attention

- To use self-attention in **decoders**, we need to ensure we can't peek at the future.
- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, j \leq i \\ -\infty, j > i \end{cases}$$



Self-Attention

Barriers and solutions for Self-Attention as a building block

Barriers	Solutions
• Doesn't have an inherent notion of order!	• Add position representations to the inputs
• No nonlinearities for deep learning magic! It's all just weighted averages	• Easy fix: apply the same feedforward network to each self-attention output.

• Need to ensure we don't "look at the future" when predicting a sequence
  • Like in machine translation
  • Or language modeling
	• Mask out the future by artificially setting attention weights to 0!

Self-Attention Block– Wrapping Things Up

- **Self-attention**:
  - the basis of the method.
- **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 feed-forward network.
- **Masking**:
  - In order to parallelize operations while not looking at the future.
  - Keeps information about the future from "leaking" to the past.