

Transformers

CS 685, Fall 2025

Advanced Natural Language Processing

https://people.cs.umass.edu/~brenocon/cs685_f25/

Brendan O'Connor

College of Information and Computer Sciences

University of Massachusetts Amherst

[Incl. slides from Mohit Iyer, Richard Socher]

- Today: introduce "Transformer" network architecture
 - *Attention* for generative LMs, motivated by MT
 - *Self-attention* + feedforward for token embeddings

- Transformers: a neural network architecture for sequences, using feed-forward layers with *attention* mechanism to deal with sequences
 - Alternative to RNNs; core to **all** current major LLMs
 - generative transformers (GPT & many more)
 - bidirectional transformers (BERT)
- Introduced by the not-totally-insightful Vaswani et al. (2017)

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
lukaszkaizer@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

A very approximate timeline

1990 Static Word Embeddings

2003 Neural Language Model

2008 Multi-Task Learning

2015 Attention

2017 Transformer

2018 Contextual Word Embeddings and Pretraining

2019 Prompting

Task: translation LM $P(\text{target} \mid \text{source})$

Sequence-to-sequence: the bottleneck problem

Target sentence (output)

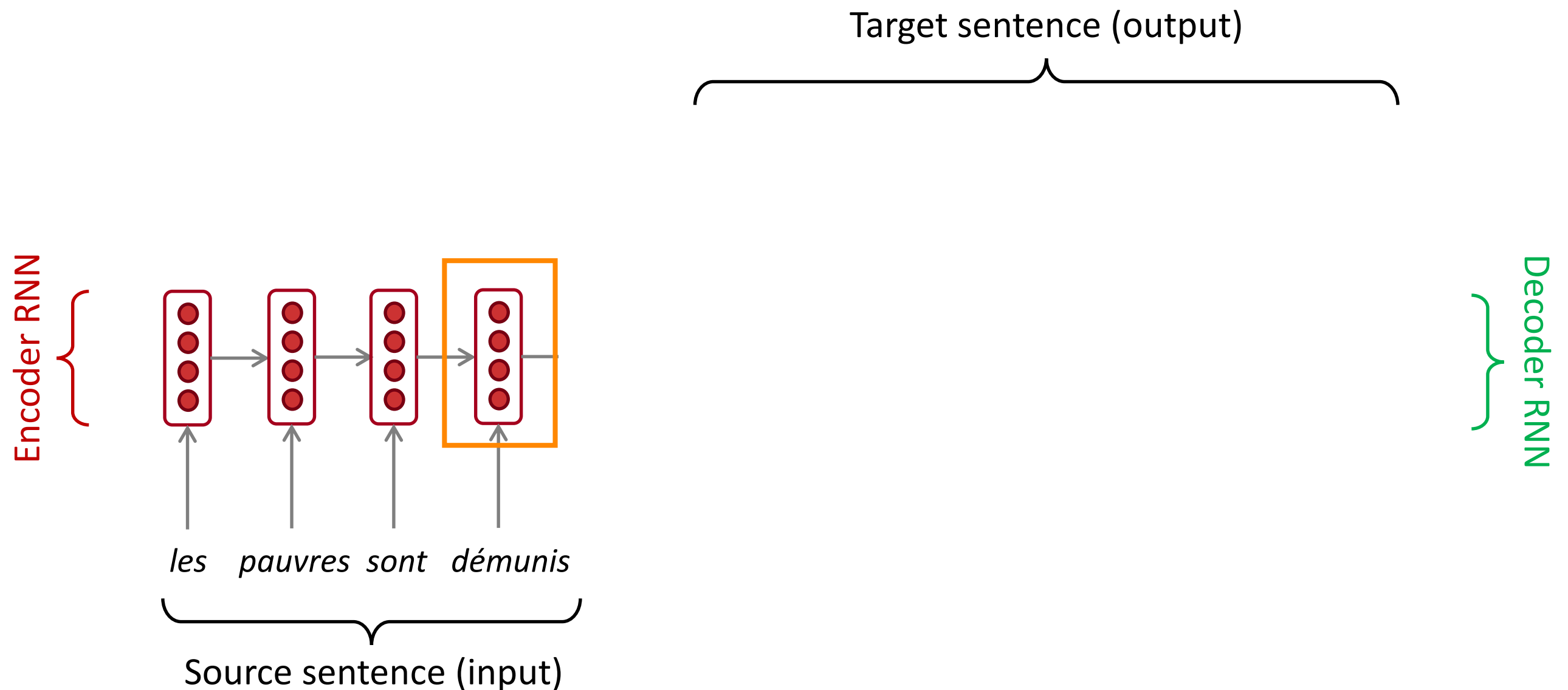
the poor don't have any money <END>

les pauvres sont démunis

Source sentence (input)

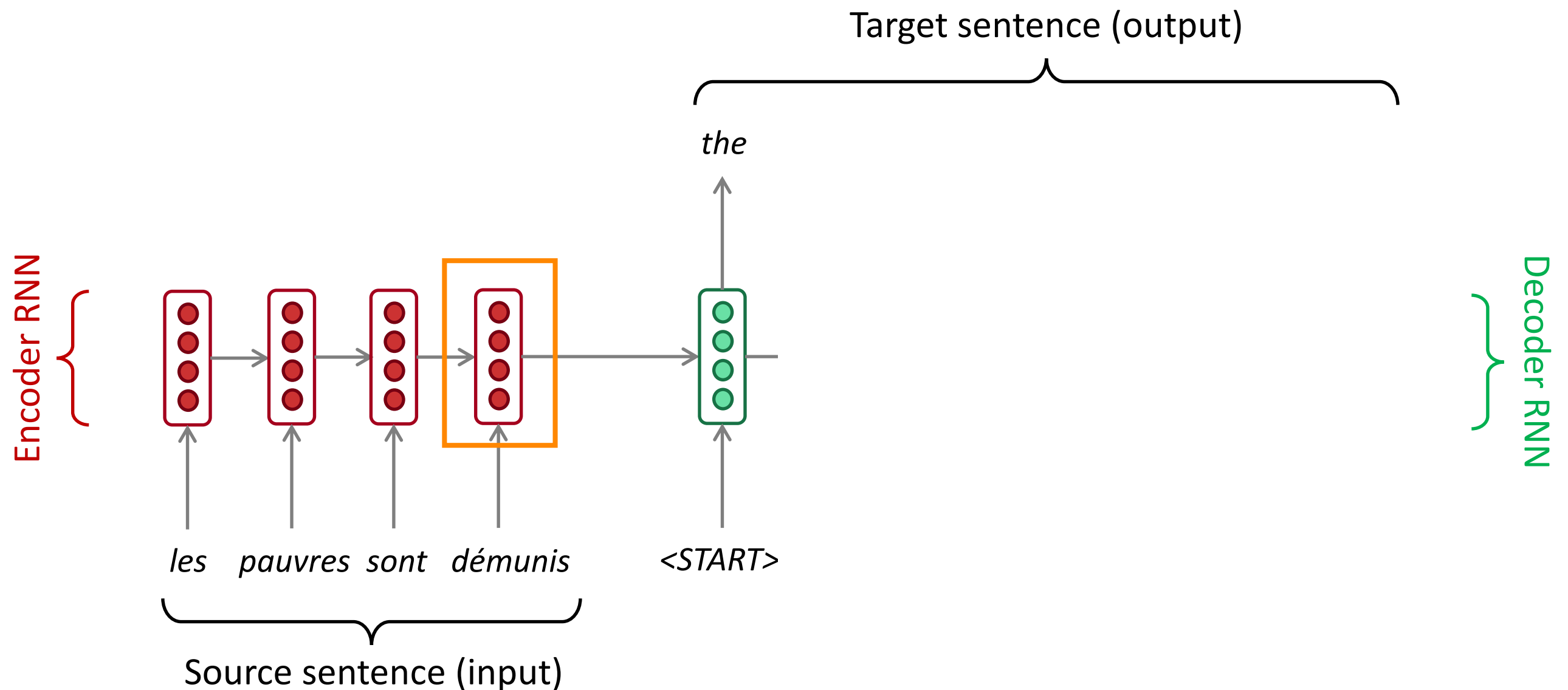
Task: translation LM $P(\text{target} \mid \text{source})$

Sequence-to-sequence: the bottleneck problem



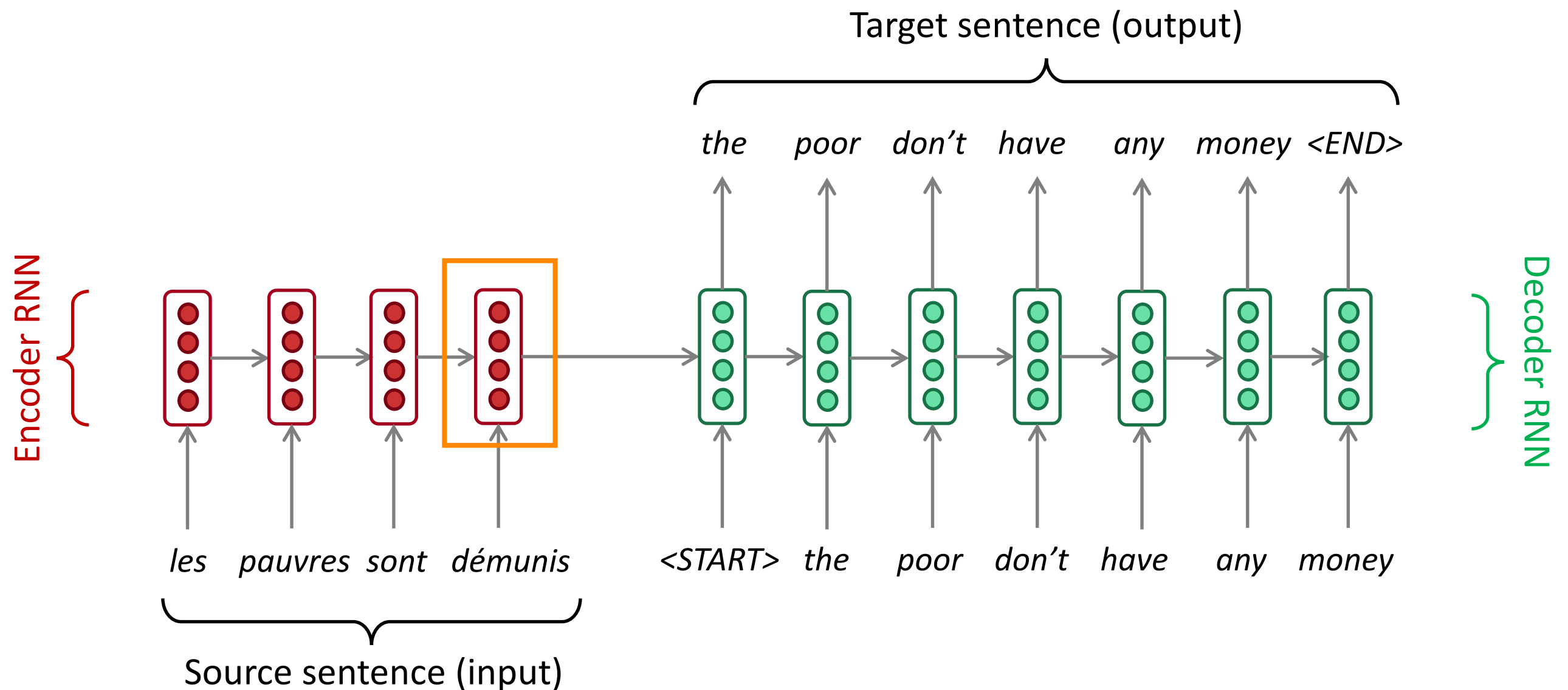
Task: translation LM $P(\text{target} | \text{source})$

Sequence-to-sequence: the bottleneck problem



Task: translation LM $P(\text{target} | \text{source})$

Sequence-to-sequence: the bottleneck problem

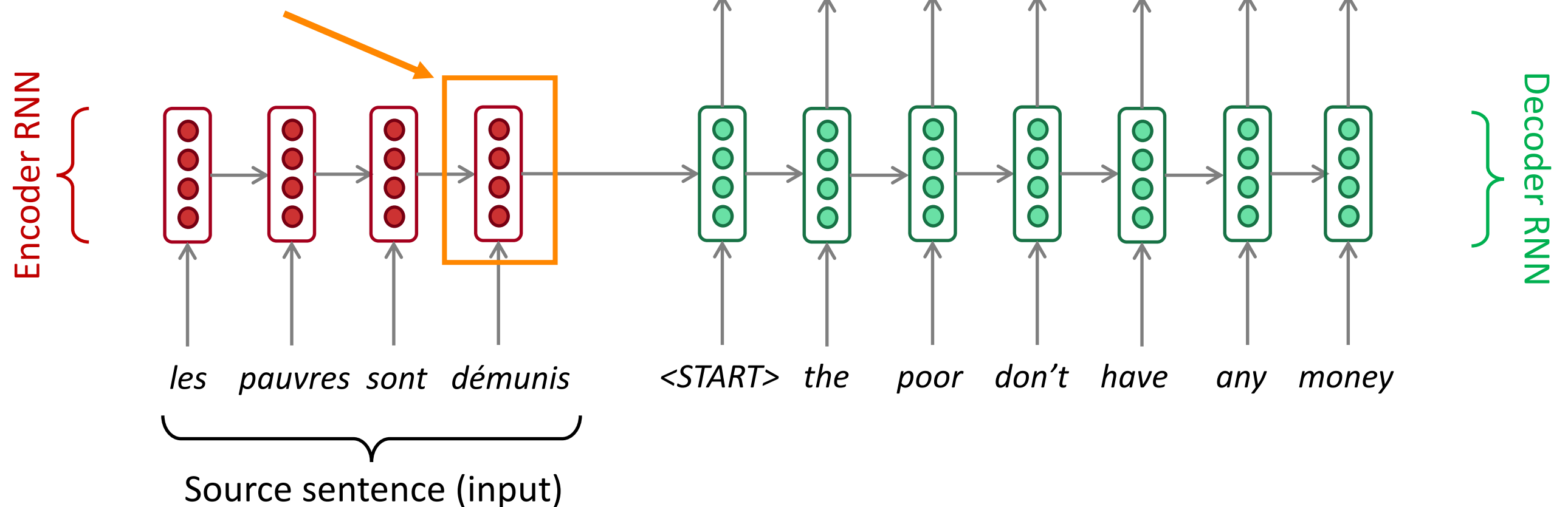


Task: translation LM $P(\text{target} \mid \text{source})$

Sequence-to-sequence: the bottleneck problem

Encoding of the
source sentence.

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



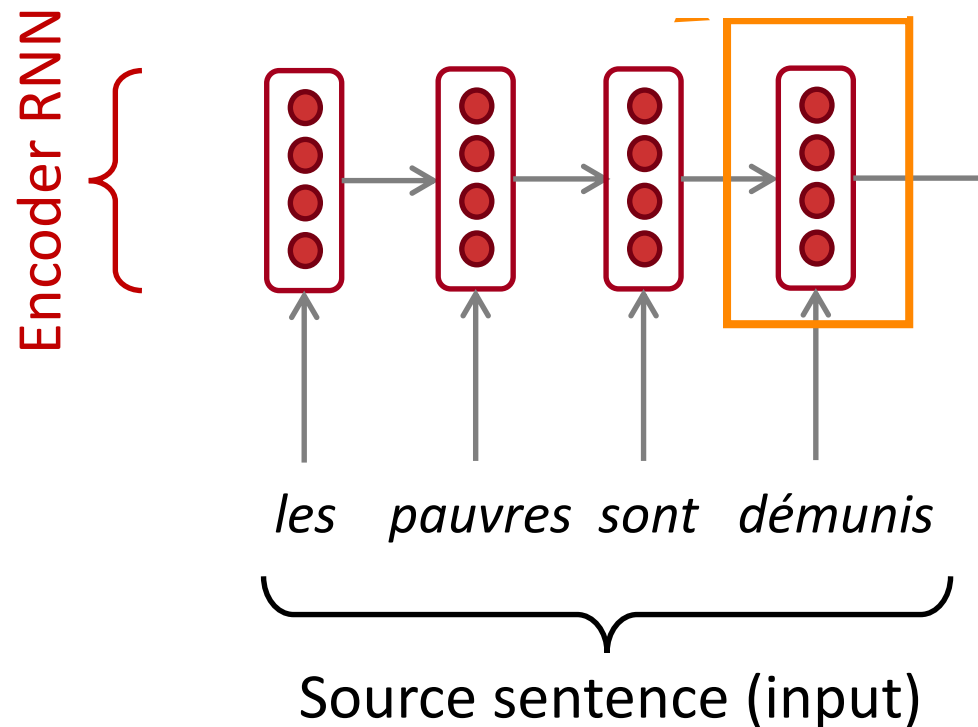
“you can’t cram the meaning
of a whole %&@#&ing
sentence into a single
\$*(&@ing vector!”

— Ray Mooney (famous NLP professor at UT Austin)

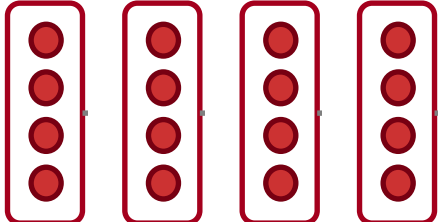
idea: what if we use multiple vectors?

Instead of:

les pauvres sont démunis = 



Let's try:

les pauvres sont démunis =  (all 4 hidden states!)

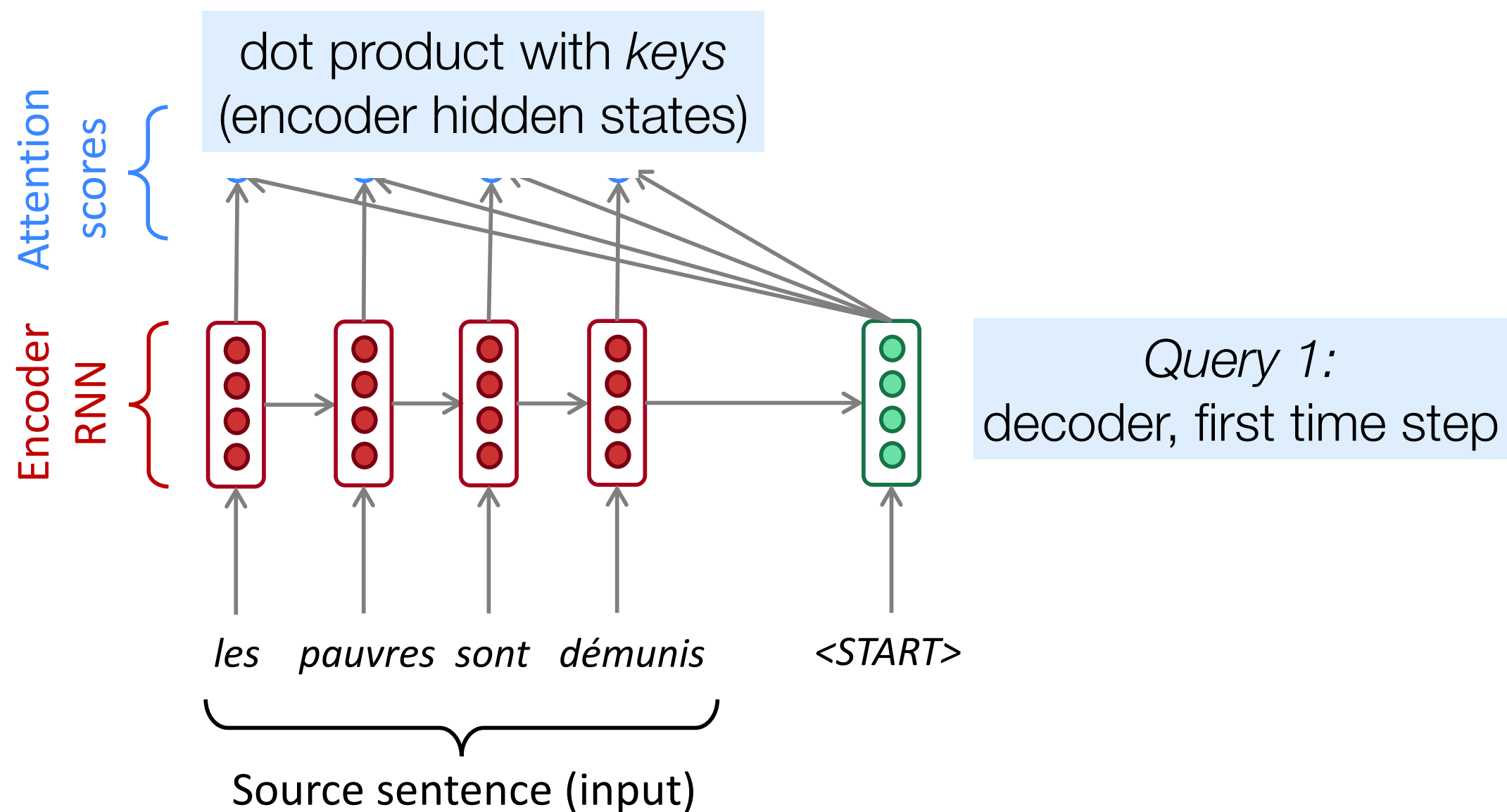
The solution: **attention**

- **Attention mechanisms** (Bahdanau et al., 2015) allow the decoder to focus on a particular part of the source sequence at each time step
 - Conceptually similar to *word alignments* in earlier MT models
 - For MT, can model differences in word order between languages

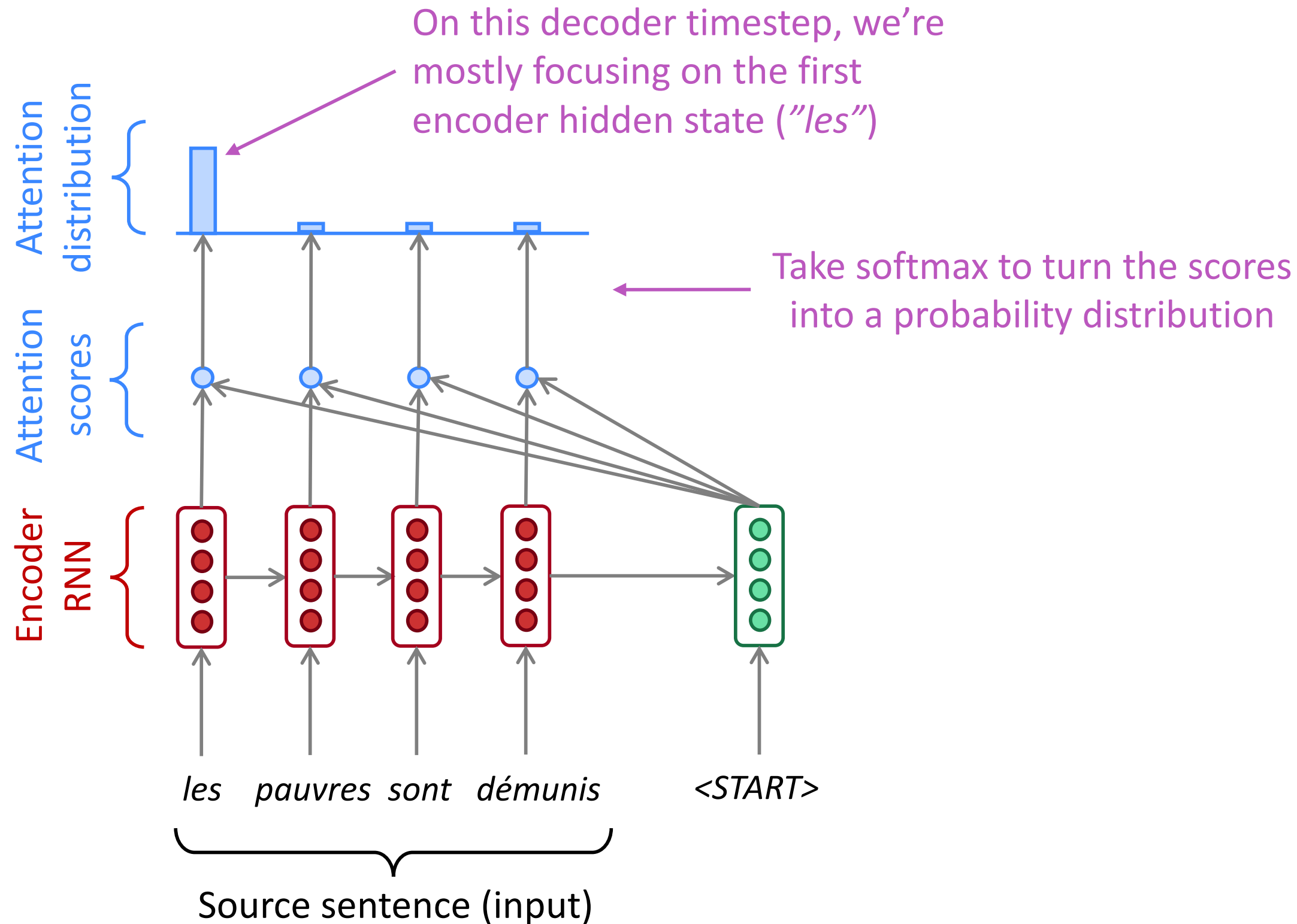
How does it work?

- in general, we have a single *query* vector and multiple *key* vectors. We want to score each query-key pair
 - Attention score based on query-key similarity
 - New representation = softmax-weighted average of token embeddings

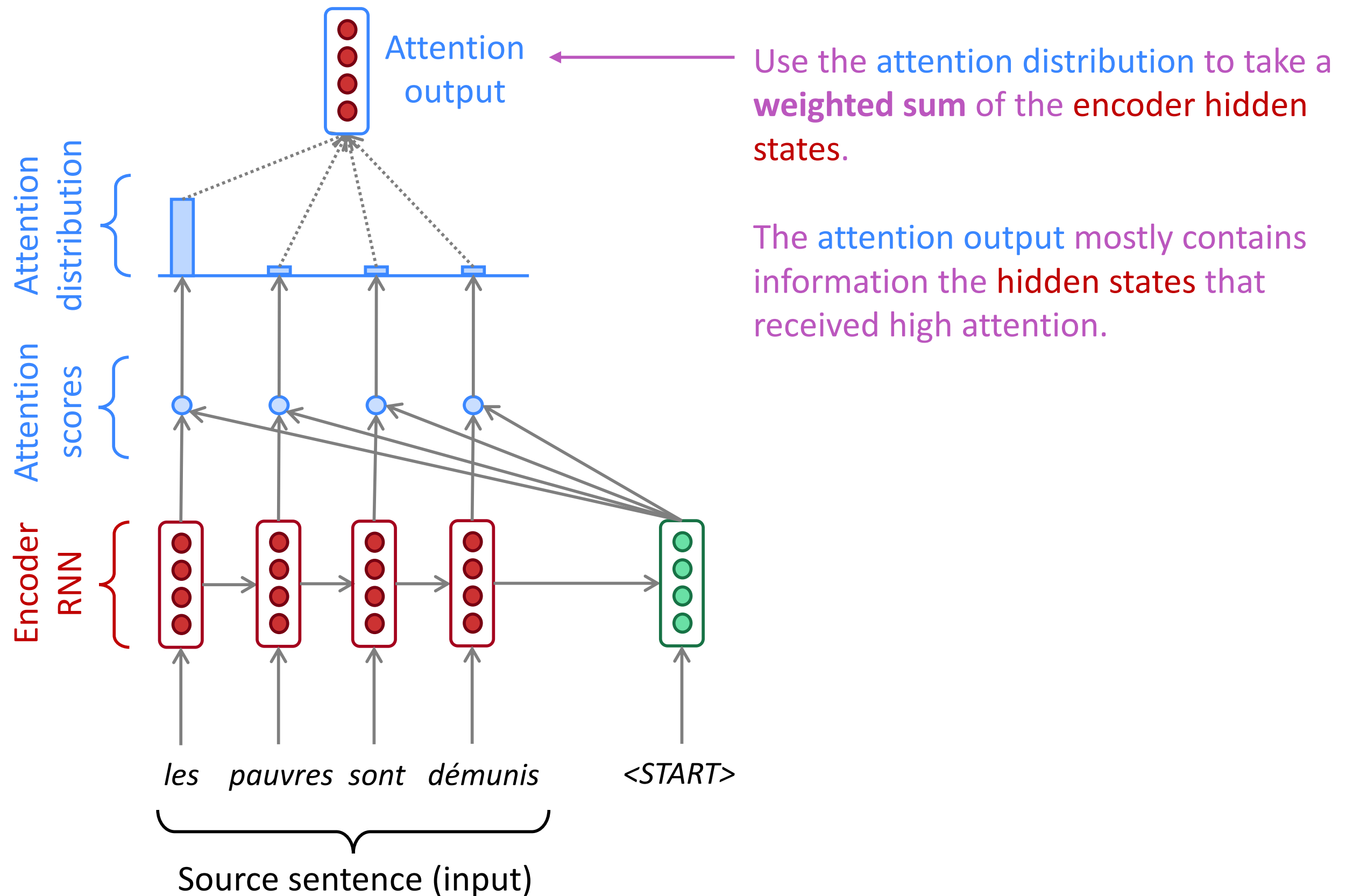
Sequence-to-sequence with attention



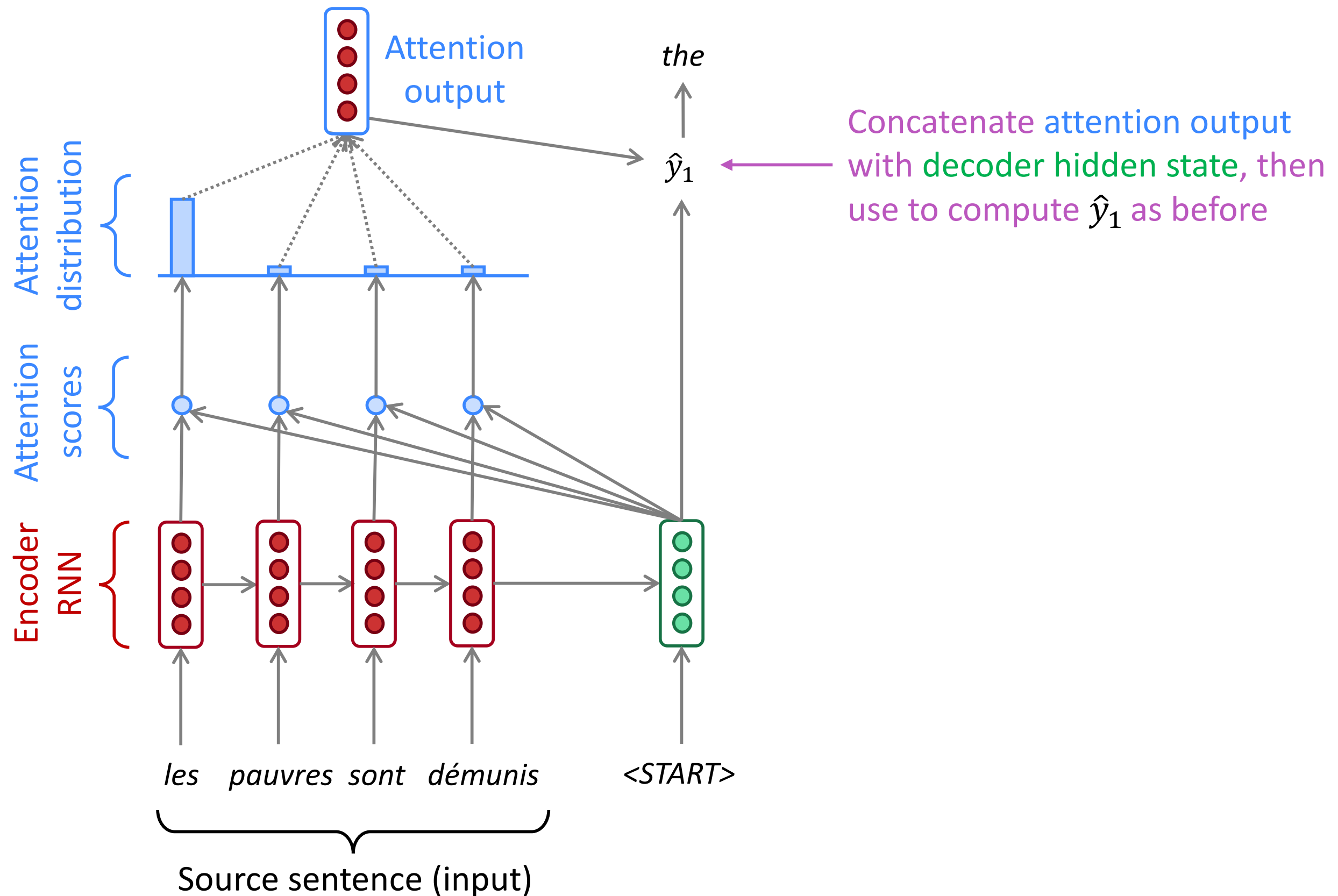
Sequence-to-sequence with attention



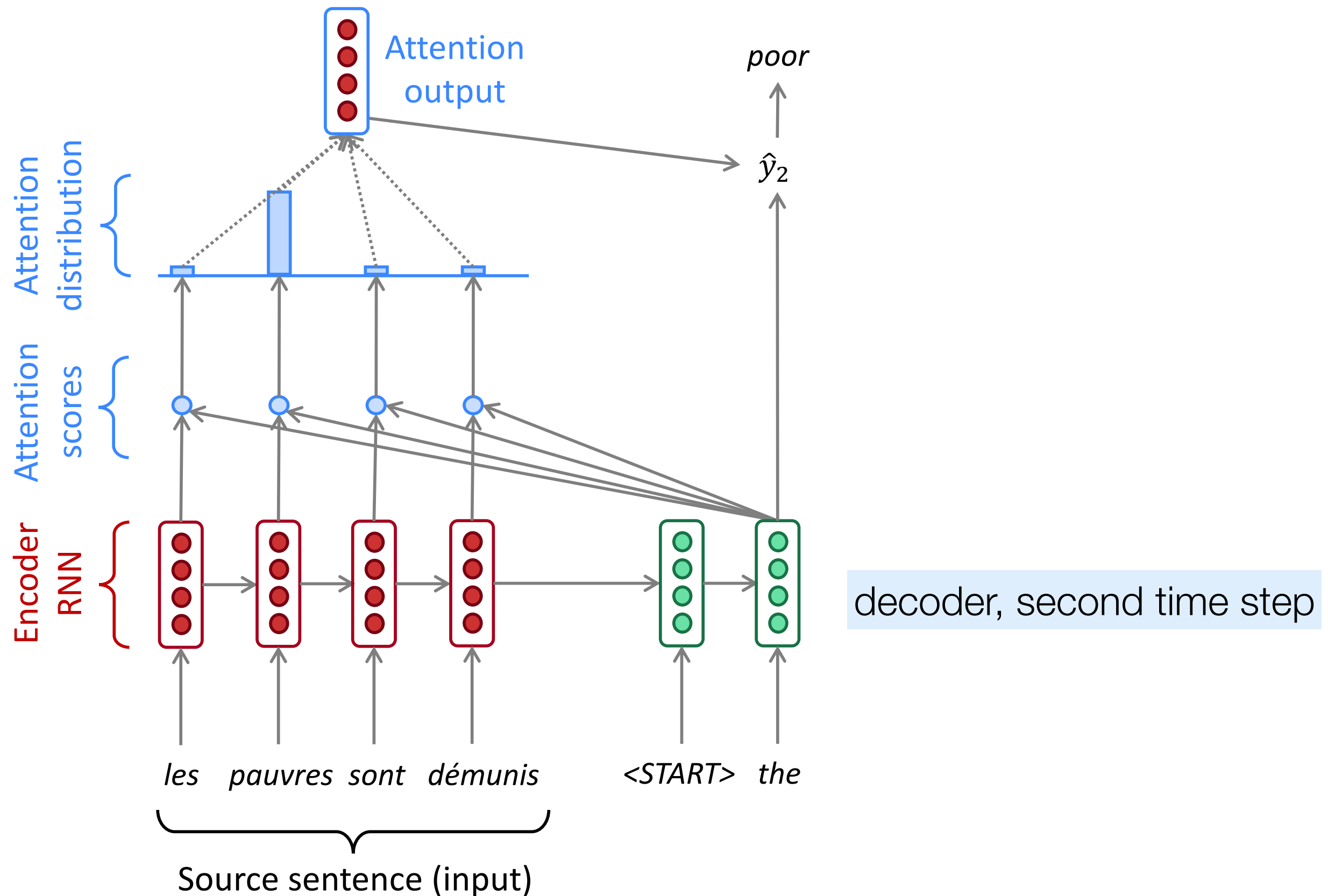
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention




Many variants of attention

- Dot product: $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T \mathbf{k}$ Luong et al., 2015
 - Scaled dot product: $a(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q}^T \mathbf{k}}{\sqrt{|\mathbf{k}|}}$ Vaswani et al., 2017
-
- Attention score based on query-key similarity
 - New representation = softmax-weighted average of token embeddings

Attention is great

- Attention significantly **improves NMT performance**
 - It's very useful to allow decoder to focus on certain parts of the source
- Attention **solves the 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!**
 - This is cool because we never explicitly trained an alignment system
 - The network just learned alignment by itself



The poor don't have any money

Les	pauvres	sont	démunis
■			
	■		
		■	■
		■	■
		■	■
		■	■

Transformer

- Goal: to calculate useful, context-aware **token embeddings**
- **Self-attention**: token-to-token attention, within a sentence or one text (*Vaswani et al. 2017*)

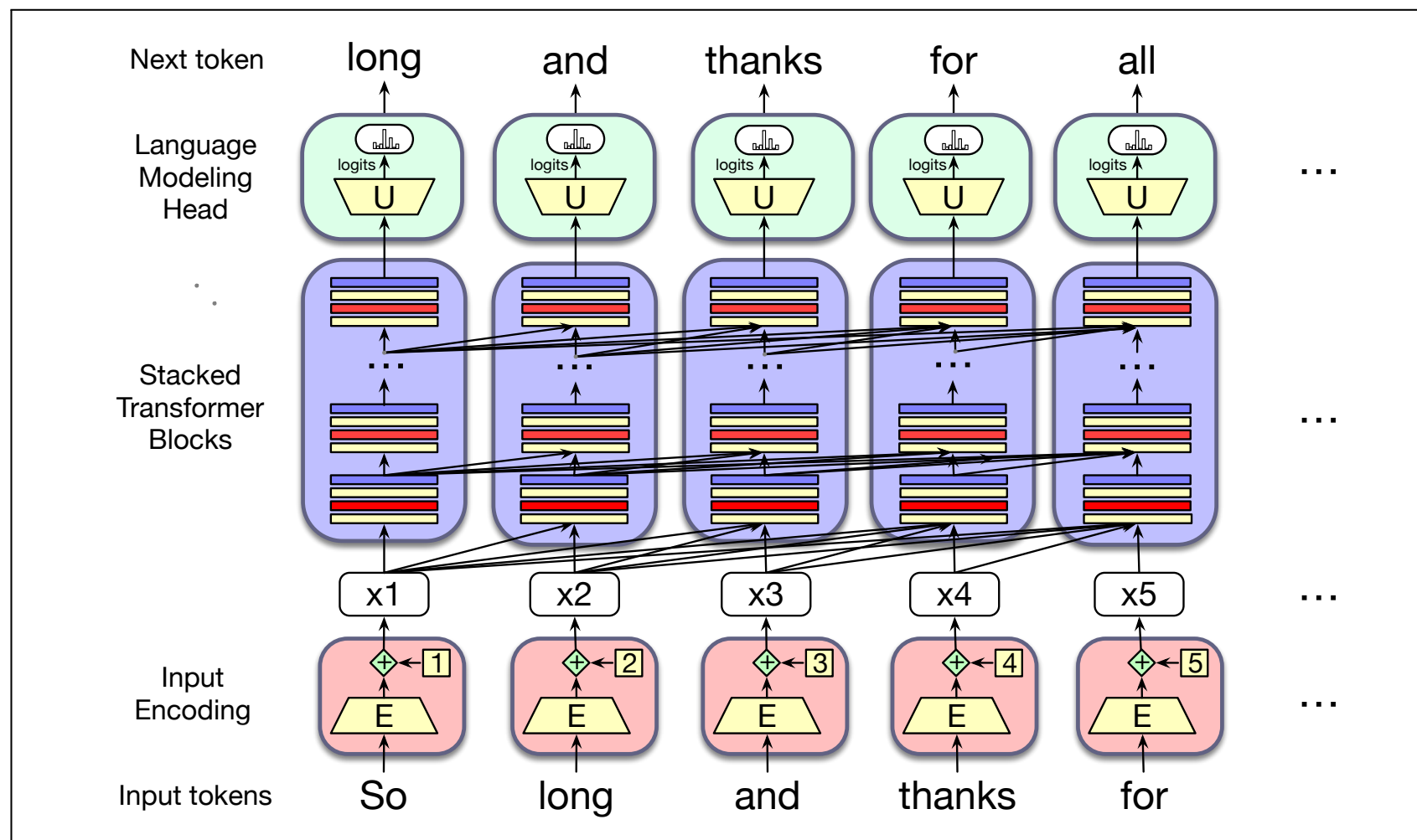


Figure 9.1 The architecture of a (left-to-right) transformer, showing how each input token get encoded, passed through a set of stacked transformer blocks, and then a language model head that predicts the next token.

Why self-attention?

Problem with static embeddings (word2vec)

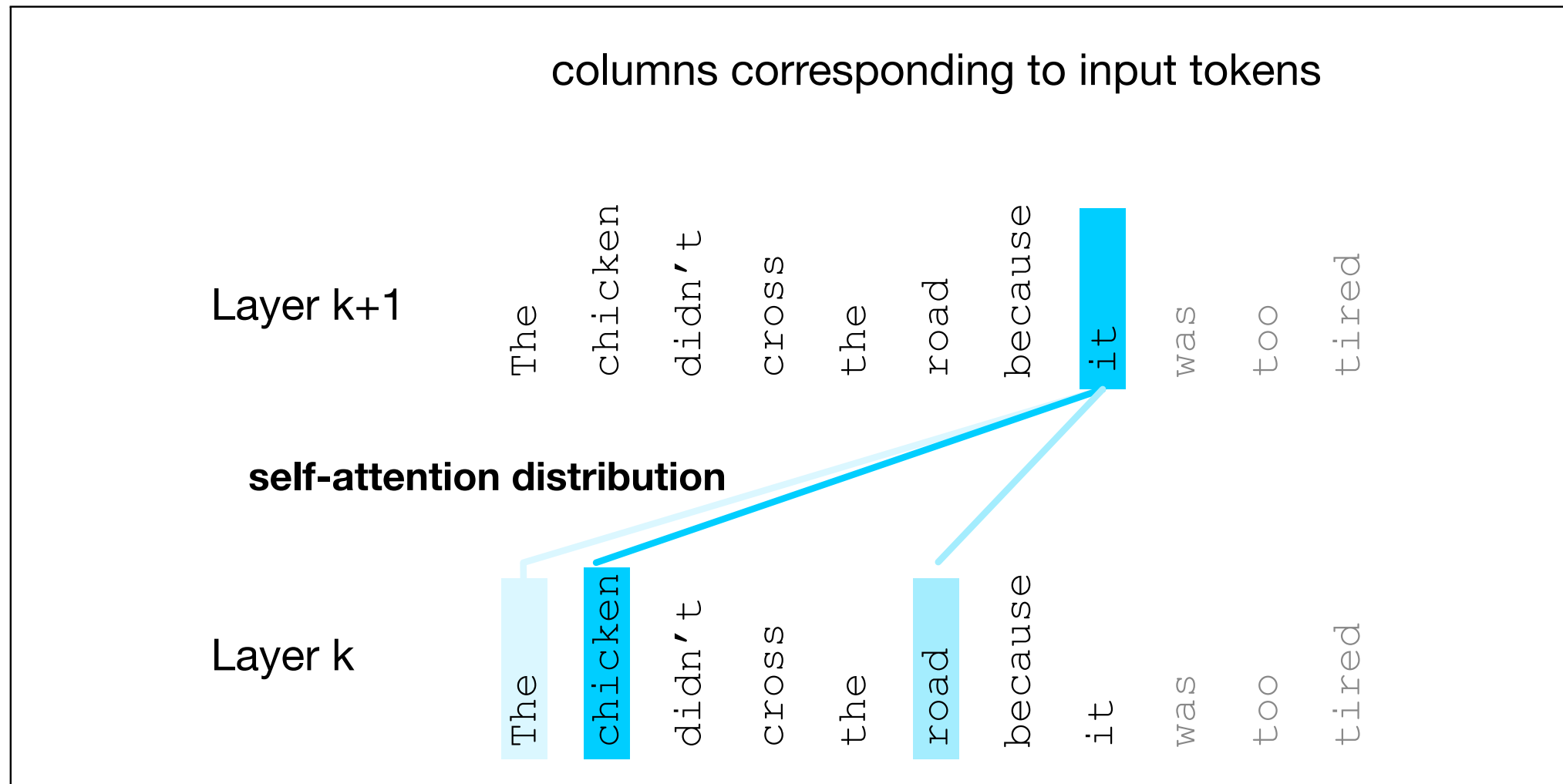
They are static! The embedding for a word doesn't reflect how its meaning changes in context.

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

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

Why self-attention?

- *The **keys** to the cabinet {are / is} on the table*
- *The **chicken** didn't cross the **road** because **it** was too {tired / wide}*
- Idea: LM-relevant contextual information may be pretty far away!
 - *The **keys** to the cabinet, which I love so much and are important and I think about all the time, [...] {are / is} on the table*



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

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

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

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

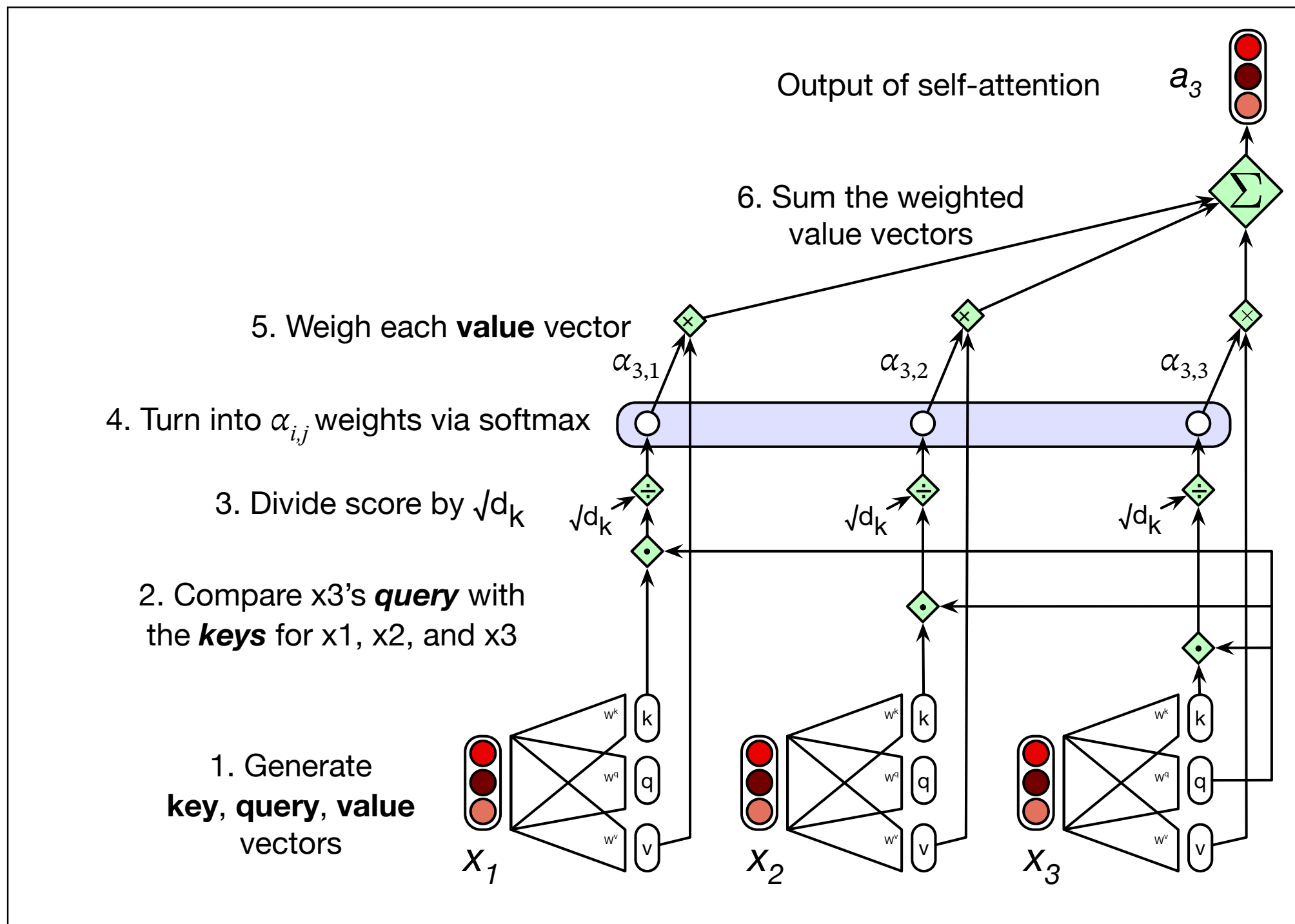


Figure 9.4 Calculating the value of a_3 , the third element of a sequence using causal (left-to-right) self-attention.

- Mathematical comparisons
 - Associate array, but soft
 - Kernel functions, but learned
 - Sequence automaton (choosing, copying, etc.)
- Parallelization of query-key products
 - Comp. advantage vs. RNNs
 - Need to mask out future information

Full Transformer block

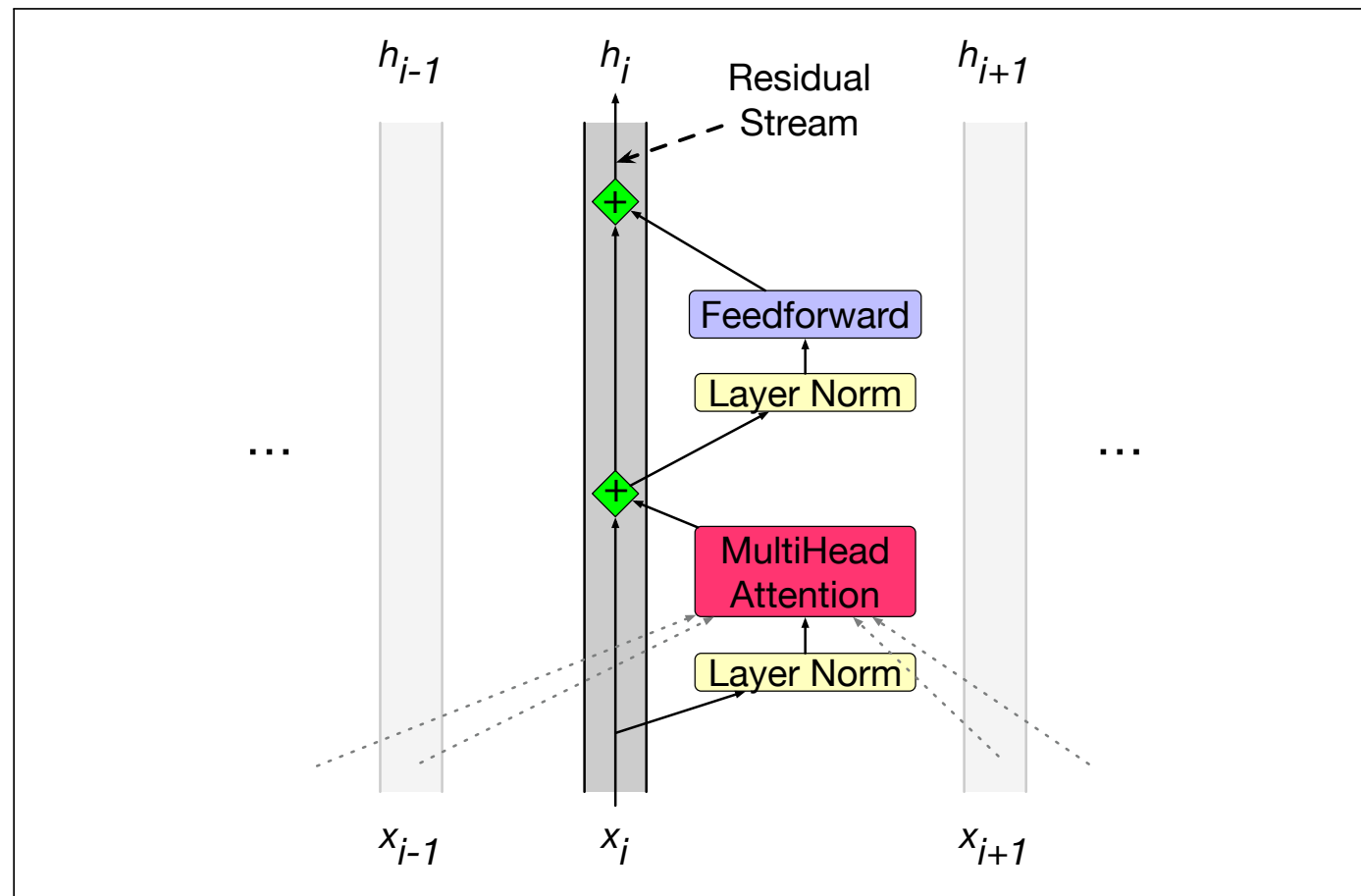


Figure 8.6 The architecture of a transformer block showing the **residual stream**. This figure shows the **prenorm** version of the architecture, in which the layer norms happen before the attention and feedforward layers rather than after.

$$\begin{aligned}
 \mathbf{t}_i^1 &= \text{LayerNorm}(\mathbf{x}_i) \\
 \mathbf{t}_i^2 &= \text{MultiHeadAttention}(\mathbf{t}_i^1, [\mathbf{t}_1^1, \dots, \mathbf{t}_N^1]) \\
 \mathbf{t}_i^3 &= \mathbf{t}_i^2 + \mathbf{x}_i \\
 \mathbf{t}_i^4 &= \text{LayerNorm}(\mathbf{t}_i^3) \\
 \mathbf{t}_i^5 &= \text{FFN}(\mathbf{t}_i^4) \\
 \mathbf{h}_i &= \mathbf{t}_i^5 + \mathbf{t}_i^3
 \end{aligned}$$

$$\text{FFN}(\mathbf{x}_i) = \text{ReLU}(\mathbf{x}_i \mathbf{W}_1 + b_1) \mathbf{W}_2 + b_2$$

$$\text{LayerNorm}(\mathbf{x}) = \gamma \frac{(\mathbf{x} - \mu)}{\sigma} + \beta$$

- Multiple layers with MLPs (FF layers)
- Multiple heads

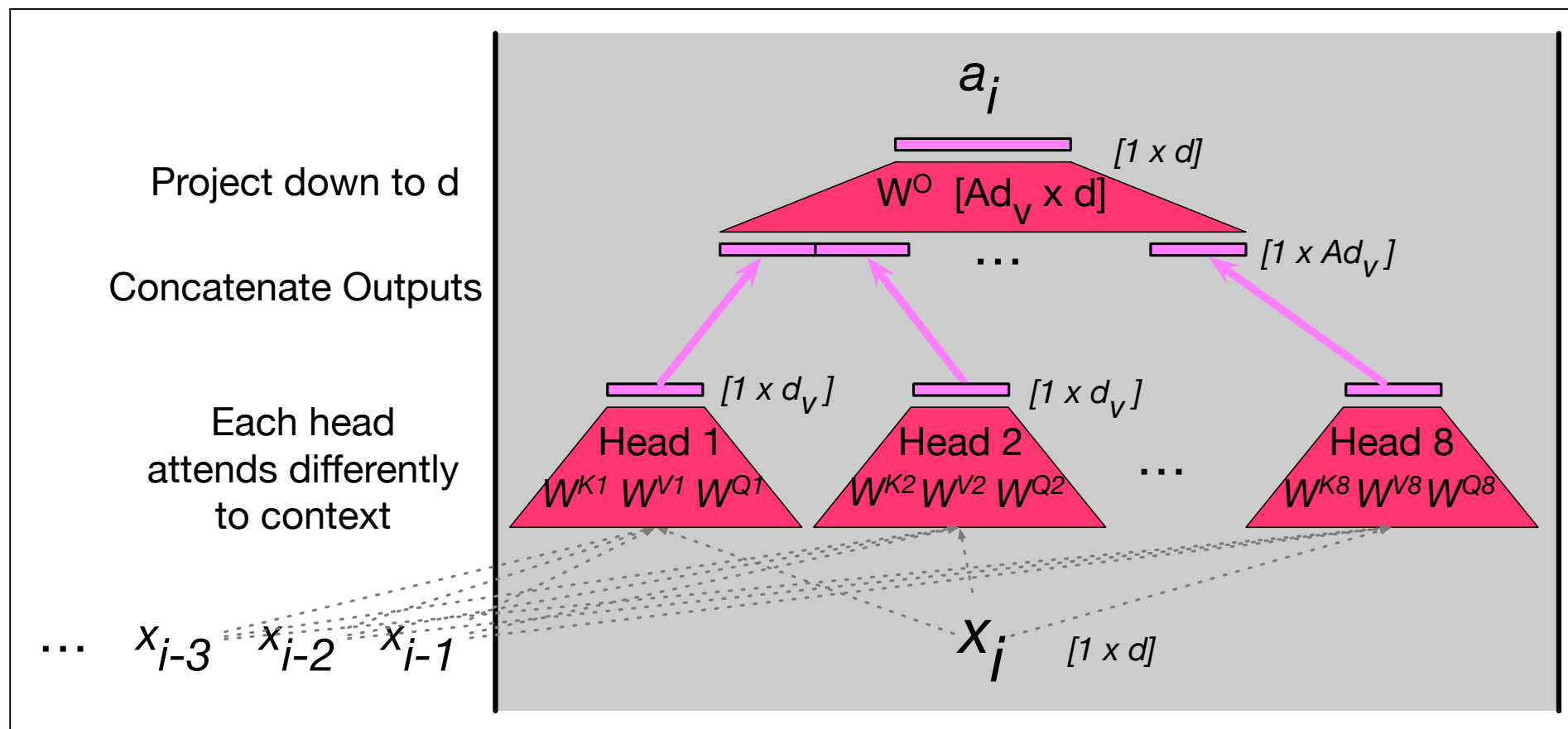


Figure 8.5 The multi-head attention computation for input \mathbf{x}_i , producing output \mathbf{a}_i . A multi-head attention layer has A heads, each with its own query, key, and value weight matrices. The outputs from each of the heads are concatenated and then projected down to d , thus producing an output of the same size as the input.

$$\mathbf{q}_i^c = \mathbf{x}_i \mathbf{W}^{Qc}; \quad \mathbf{k}_j^c = \mathbf{x}_j \mathbf{W}^{Kc}; \quad \mathbf{v}_j^c = \mathbf{x}_j \mathbf{W}^{Vc}; \quad \forall c \quad 1 \leq c \leq A$$

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

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

$$\text{head}_i^c = \sum_{j \leq i} \alpha_{ij}^c \mathbf{v}_j^c$$

$$\mathbf{a}_i = (\text{head}^1 \oplus \text{head}^2 \dots \oplus \text{head}^A) \mathbf{W}^O$$

$$\text{MultiHeadAttention}(\mathbf{x}_i, [\mathbf{x}_1, \dots, \mathbf{x}_{i-1}]) = \mathbf{a}_i$$

- Multiple layers with MLPs (FF layers)
- Multiple heads

BERT

- “**B**idirectional encoder representations from **T**ransformers”
 - Transformer: a "**self-attention**" neural net architecture that infers **context-aware** token embeddings
 - Bidirectional: Pretraining with a **masked LM**, predicting missing word(s) from rest of words in sentence
- Usage
 - 1. **Pretrain** the network via masked language model on a large corpus
 - 2. **Fine-tune**: further learn better parameters for a specific task, e.g. classification
- BERT (+ variants) are really useful, and work because they learn both word embeddings and linguistic structure from pretraining
 - many implementations:
<https://huggingface.co/docs/transformers/index>

What does BERT learn?

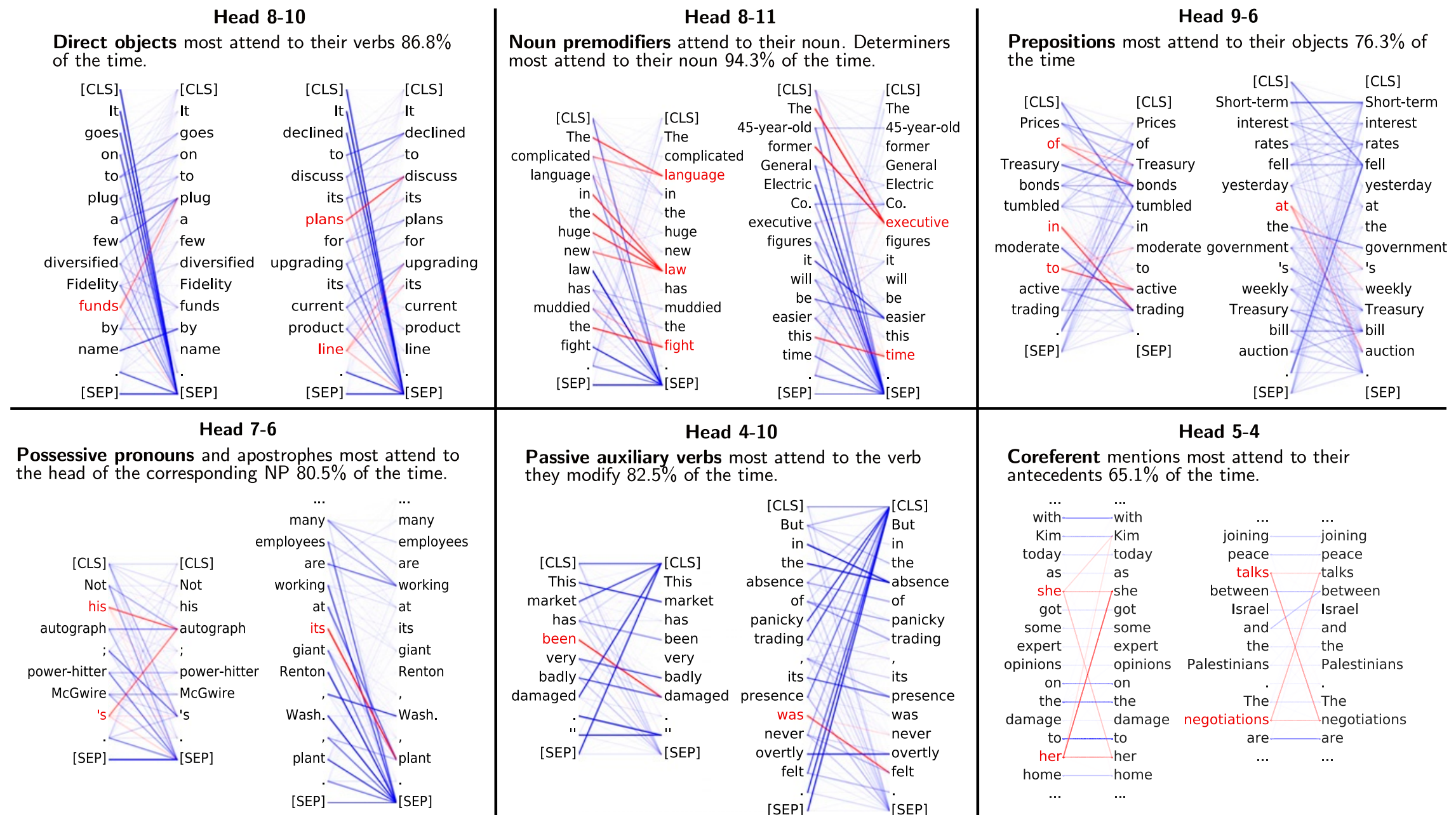


Fig. 6. Some BERT attention heads that appear sensitive to linguistic phenomena, despite not being explicitly trained on linguistic annotations. In the example attention maps, the darkness of a line indicates the size of the attention weight. All attention to/from red words is colored red; these words are chosen to highlight certain of the attention heads' behaviors. [CLS] (classification) and [SEP] (separator) are special tokens BERT adds to the input during preprocessing. Attention heads are numbered by their layer and index in BERT. Reprinted with permission from ref. 59, which is licensed under [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/).

What does BERT learn?

- BERT is typically the highest accuracy way to predict POS, syntax, NER, etc.
- If you use multiple layers to predict linguistic structures, what layers encode the information?

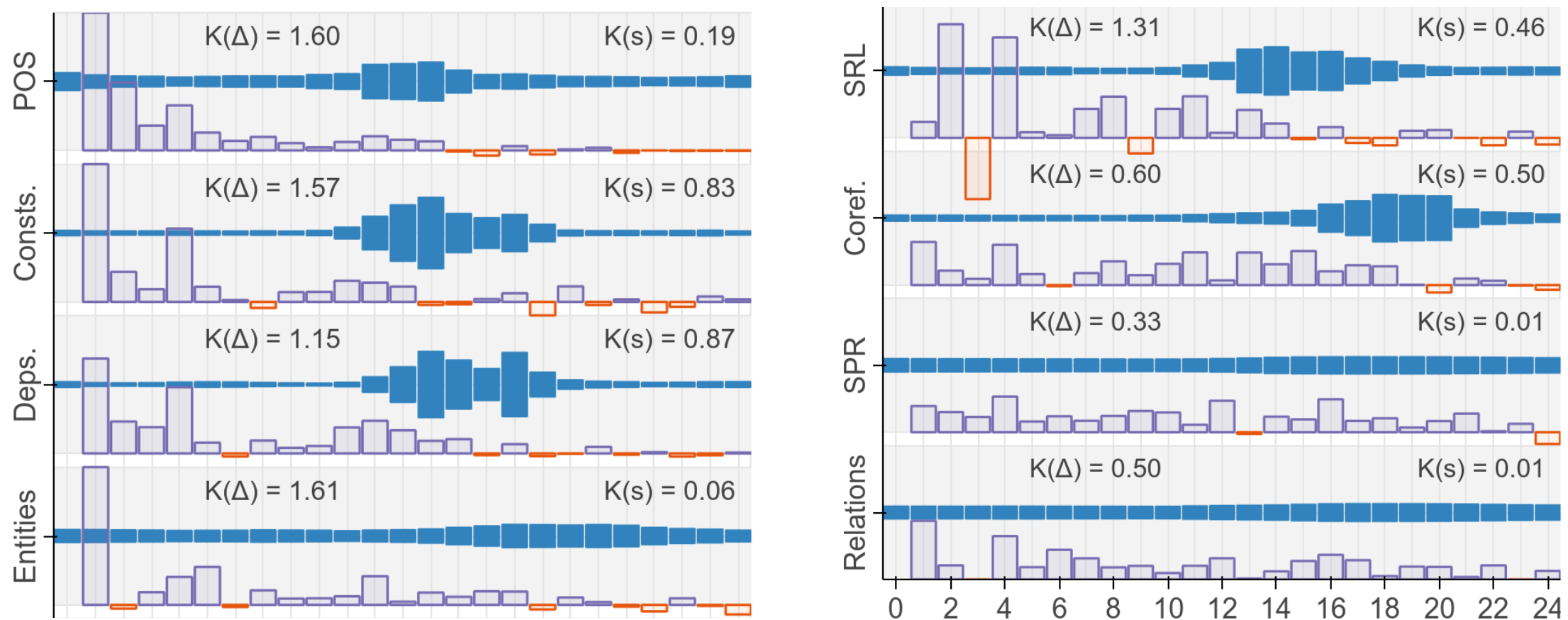


Figure 2: Layer-wise metrics on BERT-large. Solid (blue) are mixing weights $s_{\tau}^{(\ell)}$ (§3.1); outlined (purple) are differential scores $\Delta_{\tau}^{(\ell)}$ (§3.2), normalized for each task. Horizontal axis is encoder layer.

- stopped here 10/1

What does BERT learn?

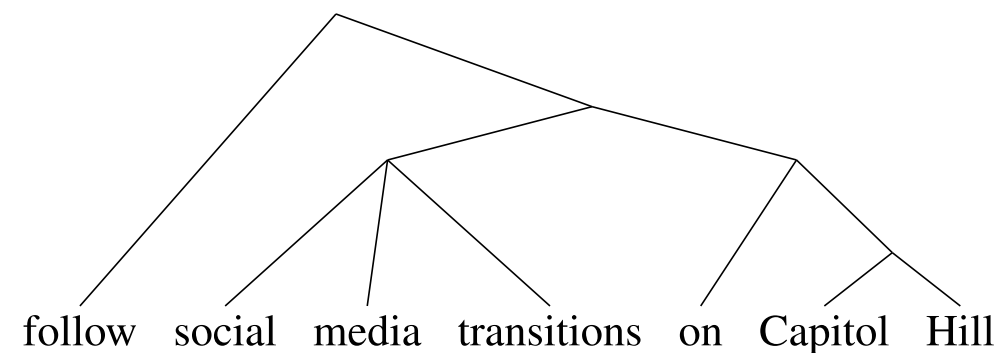


Figure 1: Parameter-free probe for syntactic knowledge: words sharing syntactic subtrees have larger impact on each other in the MLM prediction (Wu et al., 2020).

Using BERT

- You get
 - Per-token embeddings
 - Multiple layers of embeddings (!)
 - Embedding for per-sentence “[CLS]” symbol
- Use as input for tasks
 - Fine-tuning: add a prediction head, then backprop through the actual BERT model itself
 - The transformer network (with fine-tuned parameters) *is* your final classifier/tagger
 - Less common: directly use embeddings

Byte pair encoding (BPE)

- BERT is a neural LM designed to be used on arbitrary text later. But what should the vocabulary be?
- Deal with rare words / large vocabulary by using **subword tokenization**
 - Initial analysis step iteratively merges frequent character n-grams to form the vocabulary
 - Confusing name comes from data compression literature - not actually about bytes for us
 - Poor tokenization can cause many problems in practice

system	sentence
source	health research institutes
reference	Gesundheitsforschungsinstitute
WDict	Forschungsinstitute
C2-50k	Fo rs ch un gs in st it ut io ne n
BPE-60k	Gesundheits forsch ungsinstitu ten
BPE-J90k	Gesundheits forsch ungsin stitute
source	asinine situation
reference	dumme Situation
WDict	asinine situation → UNK → asinine
C2-50k	as in in e situation → As in en si tu at io n
BPE-60k	as in ine situation → A in line- Situation
BPE-J90K	as in ine situation → As in in- Situation

Application

- Fine-tuned BERT is one of the most accurate ways to train a text classifier or tagger if you have a moderate (>100) amount of labeled data
- "BERT" sometimes means the original release, but sometimes means the general class of models (!)
- Many pretrained BERT-like, MLM-trained models are available
 - RoBERTa is a good, general-purpose one
 - mBERT and XLM-R: multilingual models
 - Many specific languages or language families (AfriBERTa, LatinBERT, ...)
 - Many domains (LegalBERT, BERTweet, SciBERT, ...)
- Check out HuggingFaces' examples
 - <https://huggingface.co/transformers/examples.html>

SentenceBERT

- Also there are many released BERT-likes tuned for specific tasks - sentiment, toxicity, etc.
- **SentenceBERT** is worth mentioning: designed to encode sentences to embedding vectors
 - Cosine similarity often works very well!
 - The model is trained to give high cosine similarity to human-annotated pairs of similar sentences
- <https://sbert.net/>

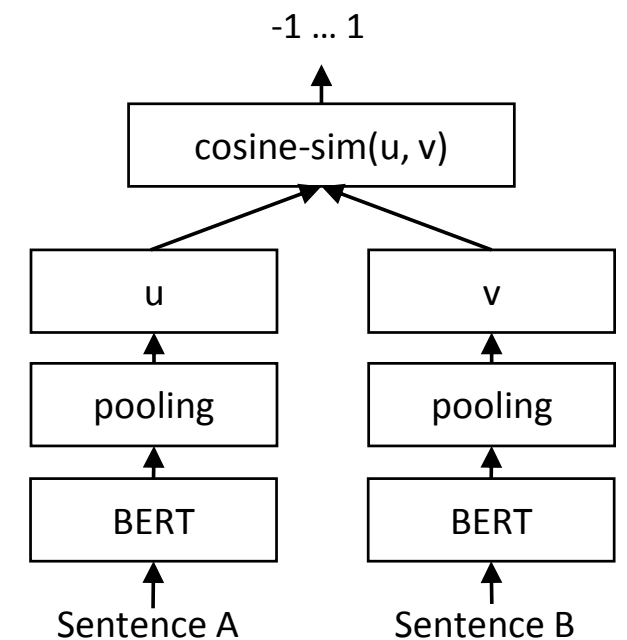


Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

Challenges

- Some issues
 - Bidirectional models can't generate
 - BERT fails to model plenty of tricky phenomena
 - How to collect a large pretraining corpus?
 - Why does all this work?
- BERT fine-tuning is often the best classifier you can make.
 - Note widely used variants: RoBERTa, DeBERTa