

CS60010: Deep Learning

Sudeshna Sarkar

Spring 2019

ATTENTION

28 Feb 2019

ATTENTION

- Defn: “the regarding of someone or something as interesting or important.”

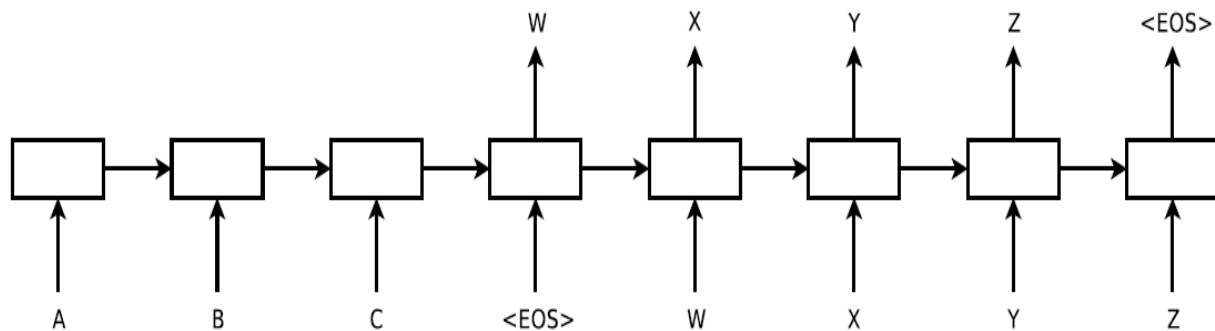
ATTENTION

2014: [Neural Translation](#) Breakthroughs

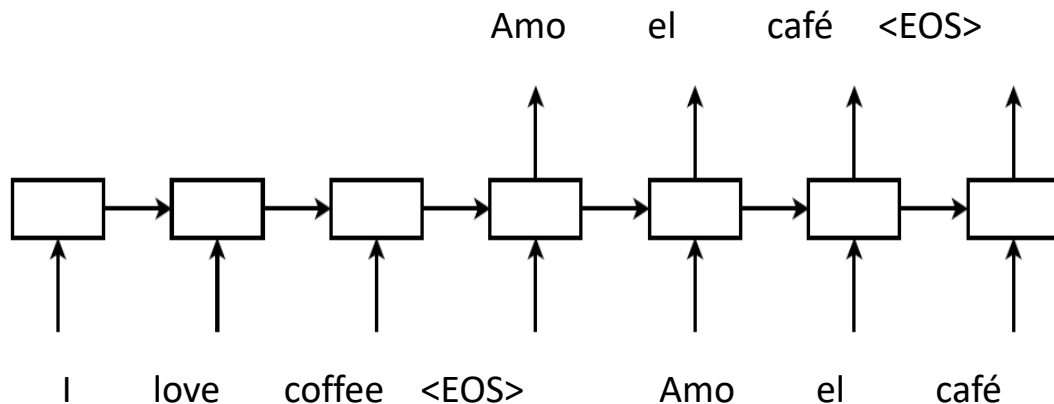
- Devlin et al, ACL'2014
 - Cho et al EMNLP'2014
 - **Bahdanau, Cho & Bengio, arXiv sept. 2014**
 - Jean, Cho, Memisevic & Bengio, arXiv dec. 2014
 - Sutskever et al NIPS'2014
-
- Ba et al 2014, Visual attention for recognition
 - Mnih et al 2014, Visual attention for recognition
 - Chorowski et al, 2014, Speech recognition
 - Graves et al 2014, Neural Turing machines
 - Yao et al 2015, Video description generation
 - Vinyals et al, 2015, Conversational Agents
 - Xu et al 2015, Image caption generation
 - Xu et al 2015, Visual Question Answering

Sequence-To-Sequence RNNs

An input sequence is fed to the left array, output sentence to the right array for training:

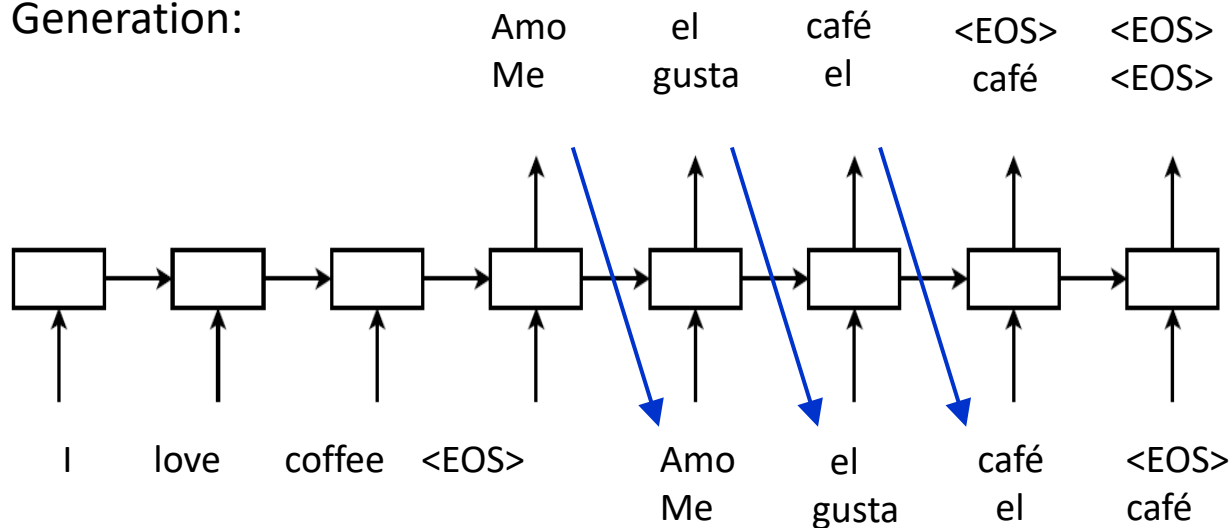


For translation:



Sequence-To-Sequence Model Translation

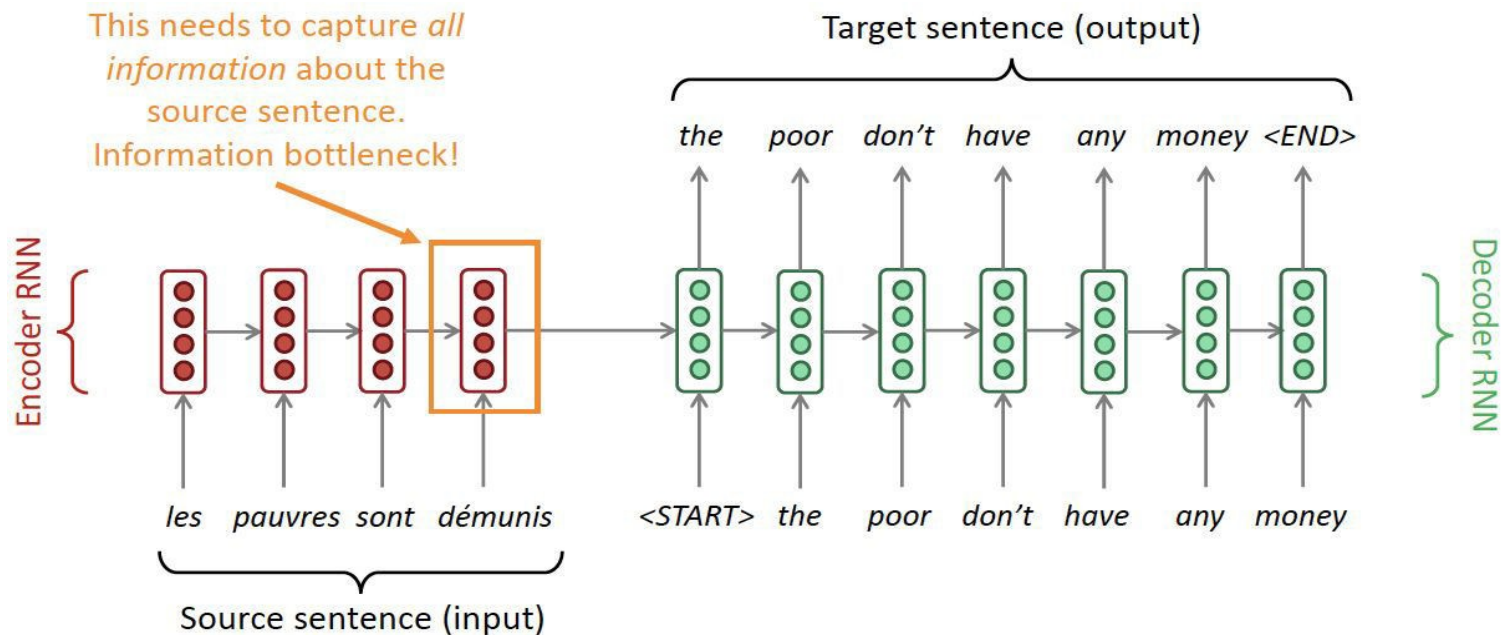
Generation:



Keep an n-best list of partial sentences, along with their partial softmax scores.

Seq2Seq: the bottleneck problem

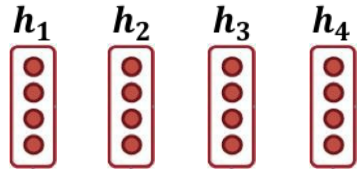
- Encoding of source sentence: a fixed length vector h_4



Attention: thinking process

How to solve the bottleneck problem?

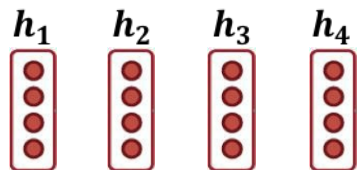
Instead of only using only h_4 , let's use all encoder hidden states!



Attention: thinking process

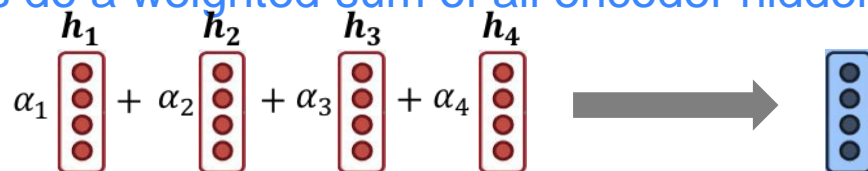
How to solve the bottleneck problem?

Instead of only using only h_4 , let's use all encoder hidden states!



How do we deal with variable length input sequence?

Let's do a weighted sum of all encoder hidden states!

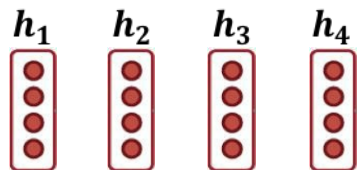


How do we get the weights α_i ?

Attention: thinking process

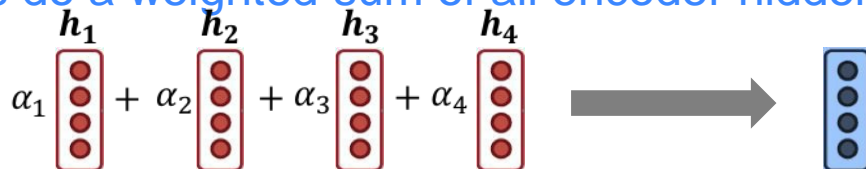
How to solve the bottleneck problem?

Instead of only using only h_4 , let's use all encoder hidden states!

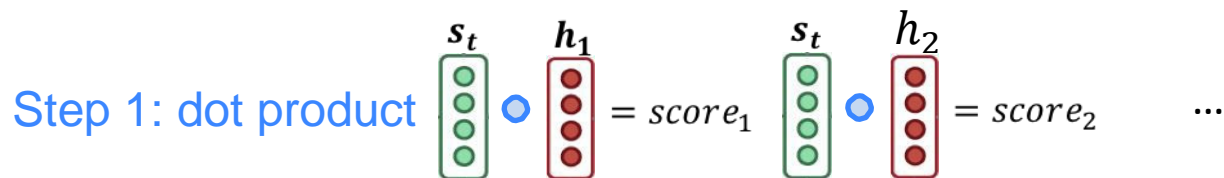


How do we deal with variable length input sequence?

Let's do a weighted sum of all encoder hidden states!



How do we get the weights α_i ?

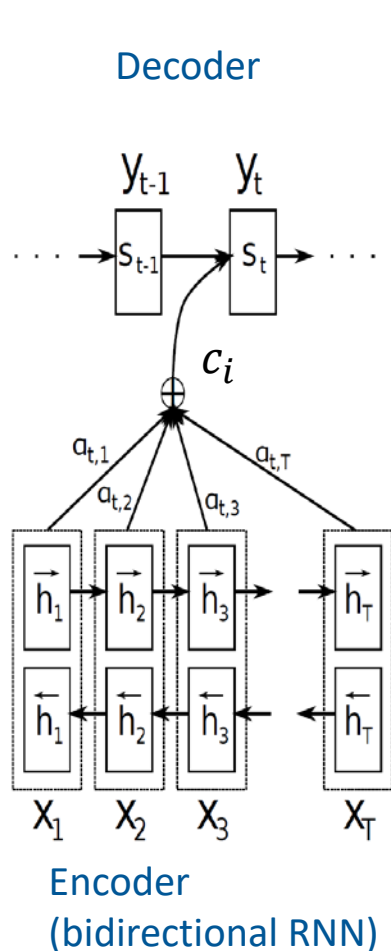


Step 2: softmax

$$\alpha_i = \frac{\exp(score_i)}{\sum_{j=1}^4 score_j}$$

Soft Attention for Translation – Bahdanau et al. model

For each output word, focus attention on a subset of all input words.



Context vector (input to decoder):

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

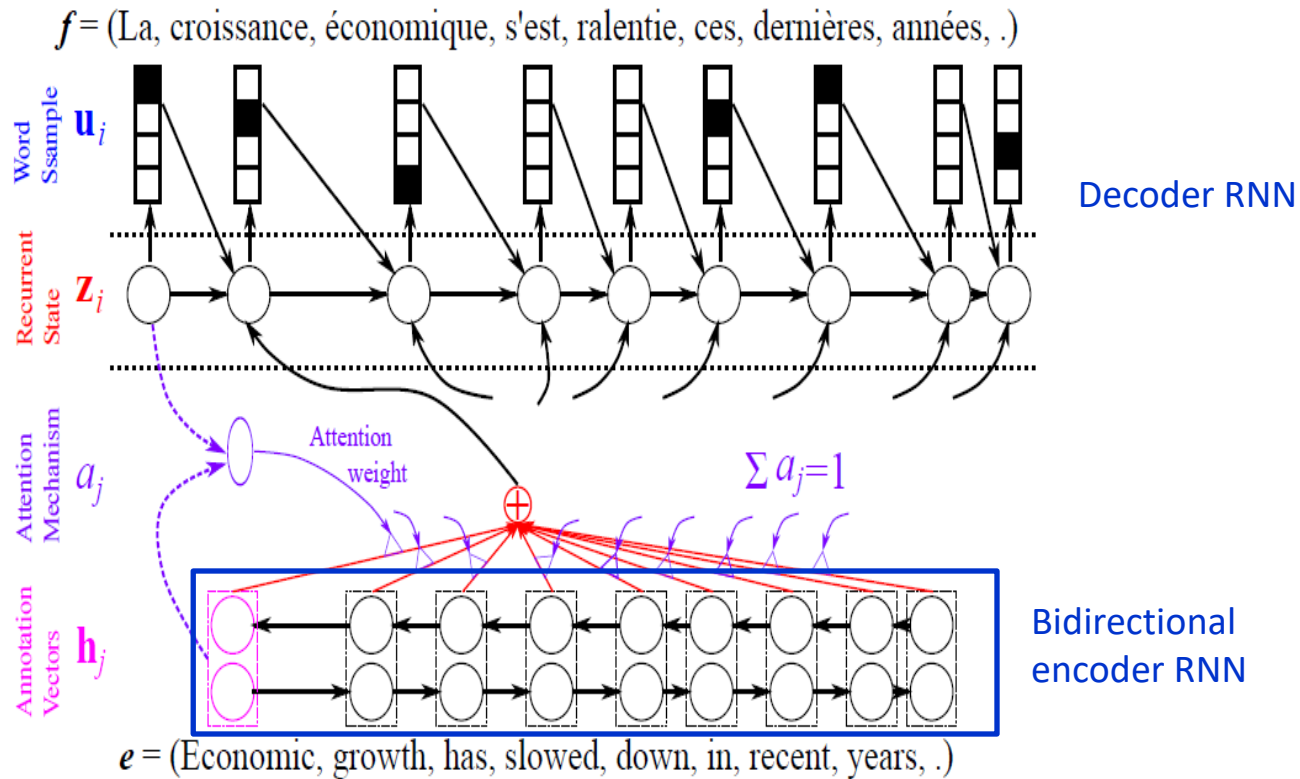
Mixture weights:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

Alignment score (how well do input words near j match output words at position i):

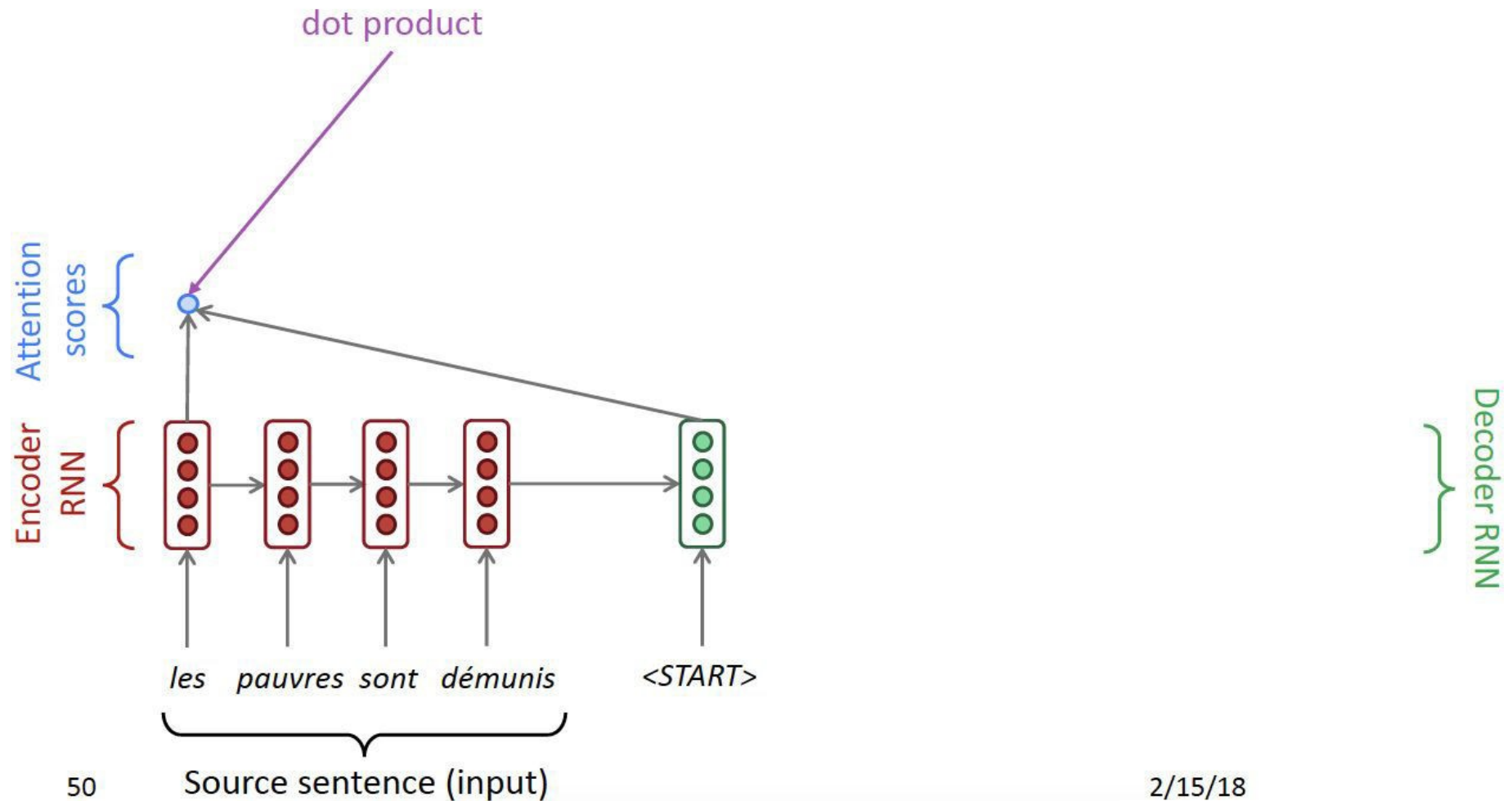
$$e_{ij} = a(s_{i-1}, h_j)$$

Soft Attention for Translation

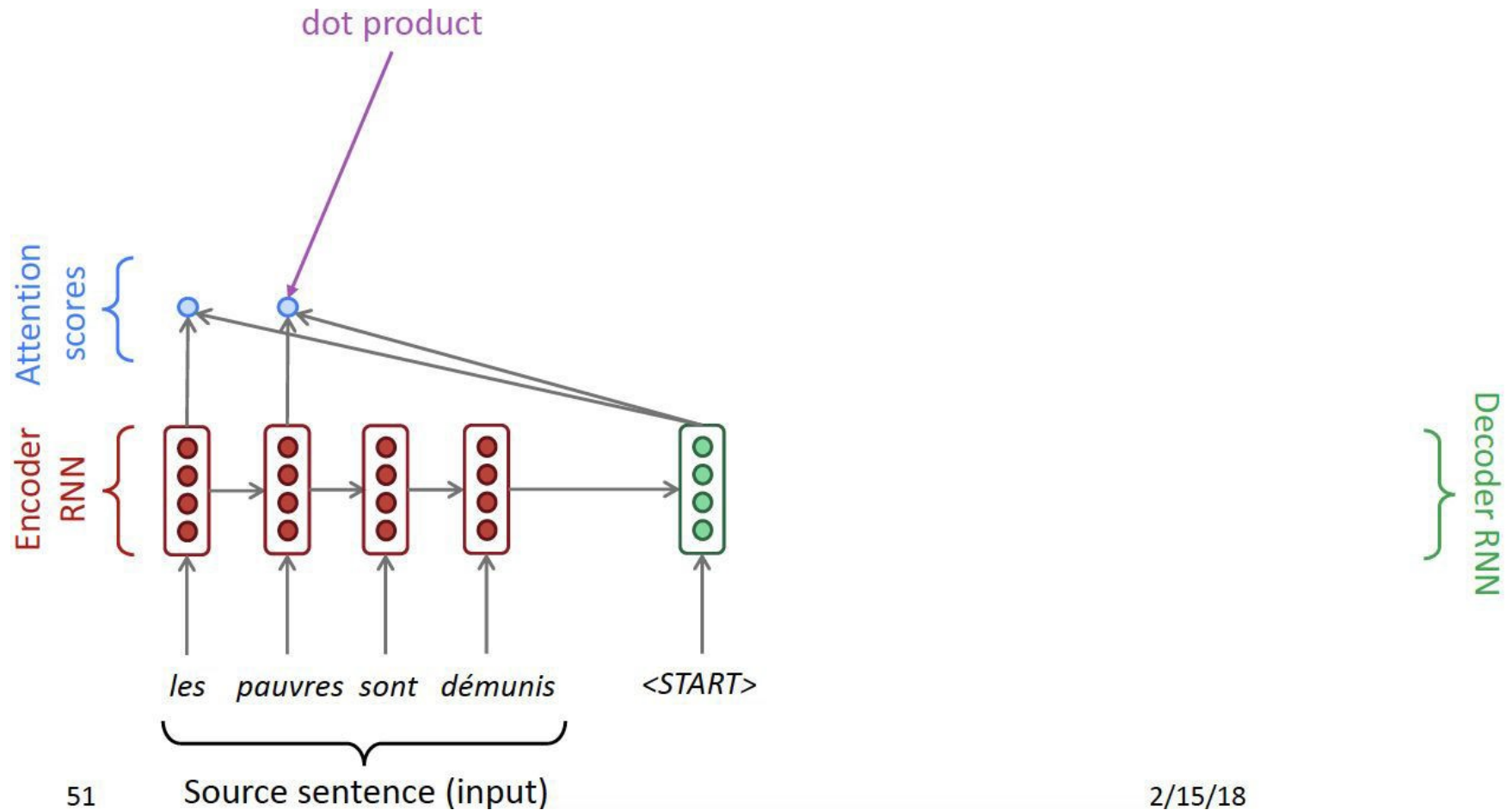


From Y. Bengio CVPR 2015 Tutorial

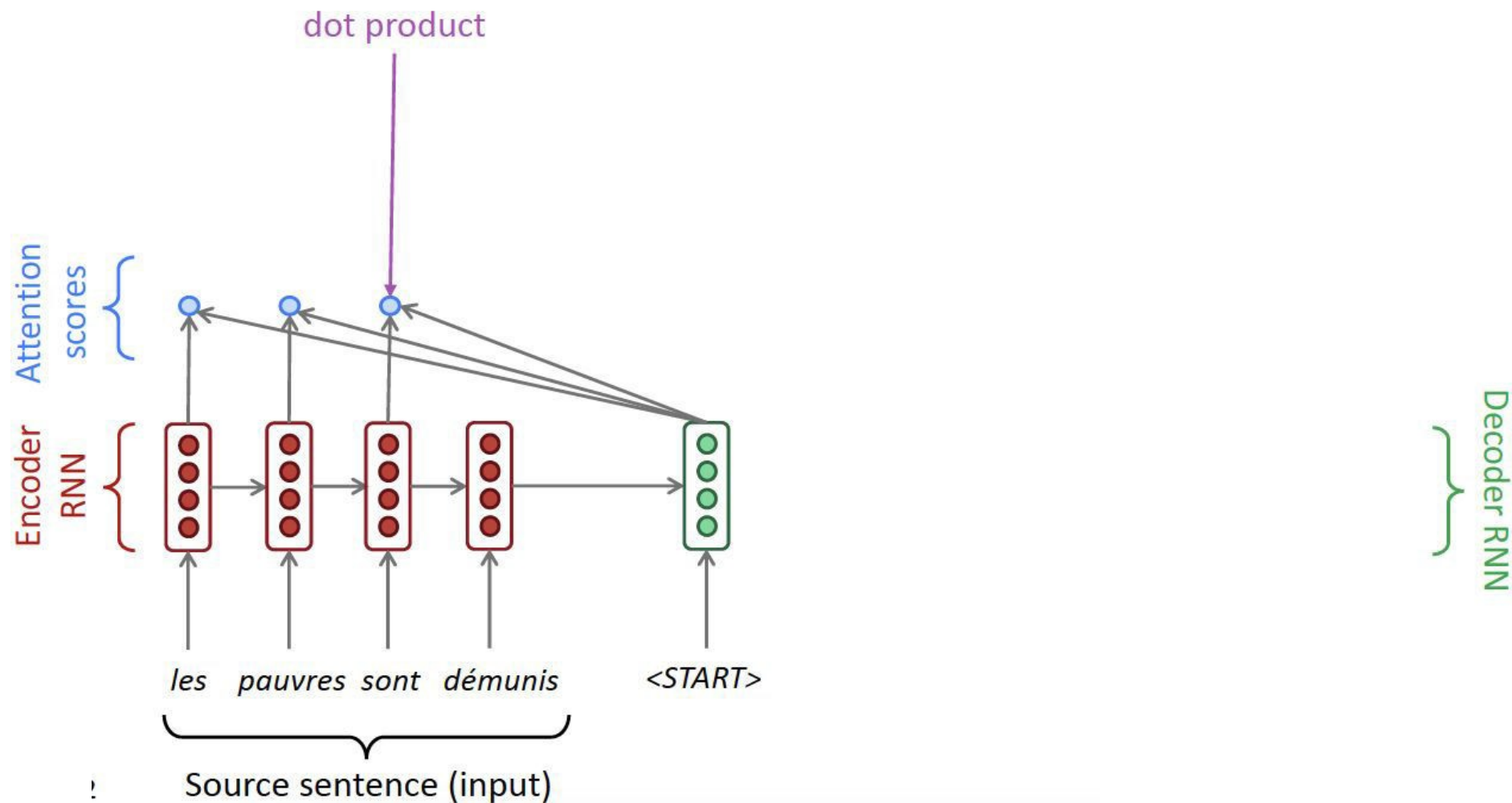
Seq2Seq with Attention



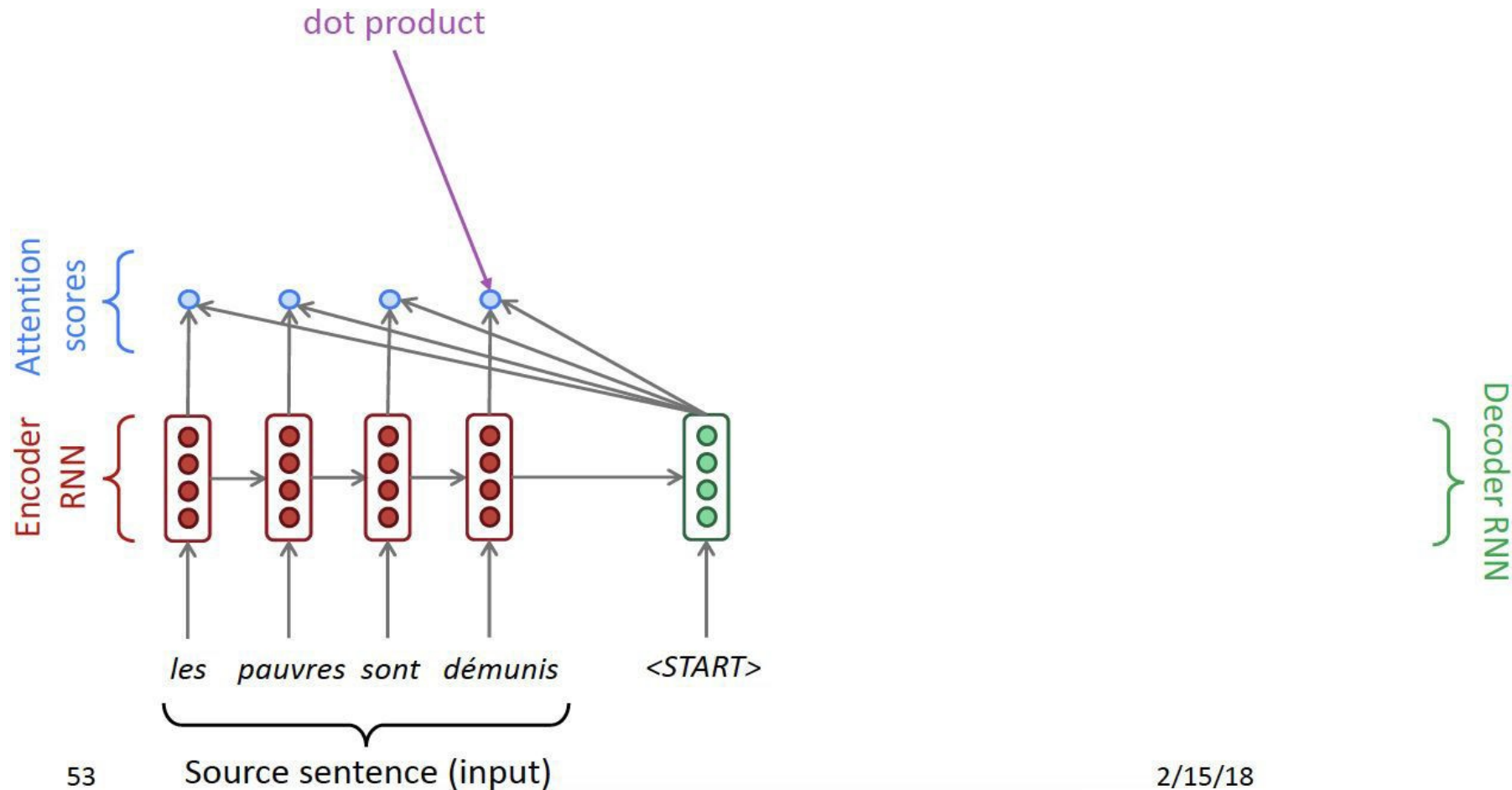
Seq2Seq with Attention



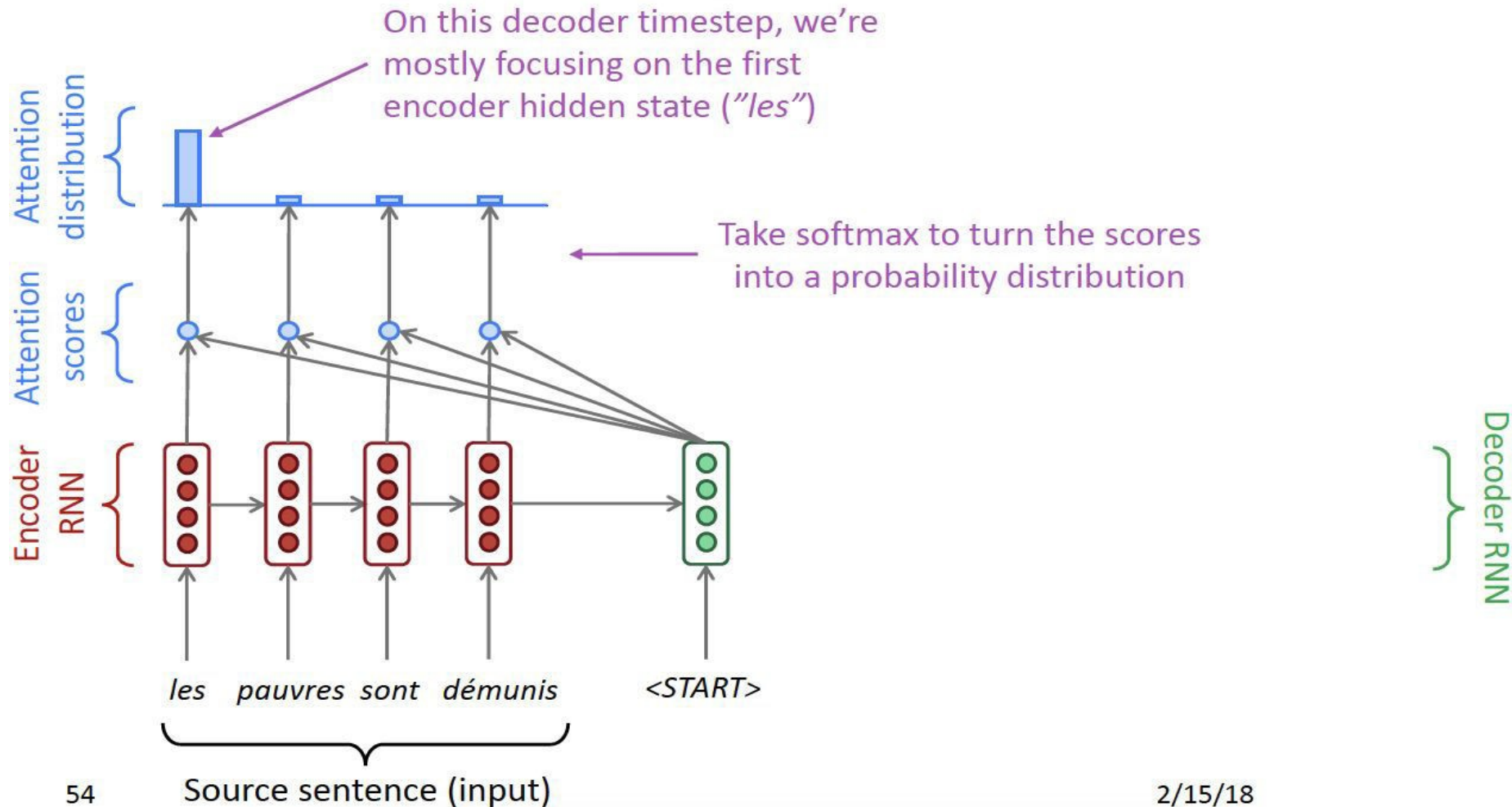
Seq2Seq with Attention



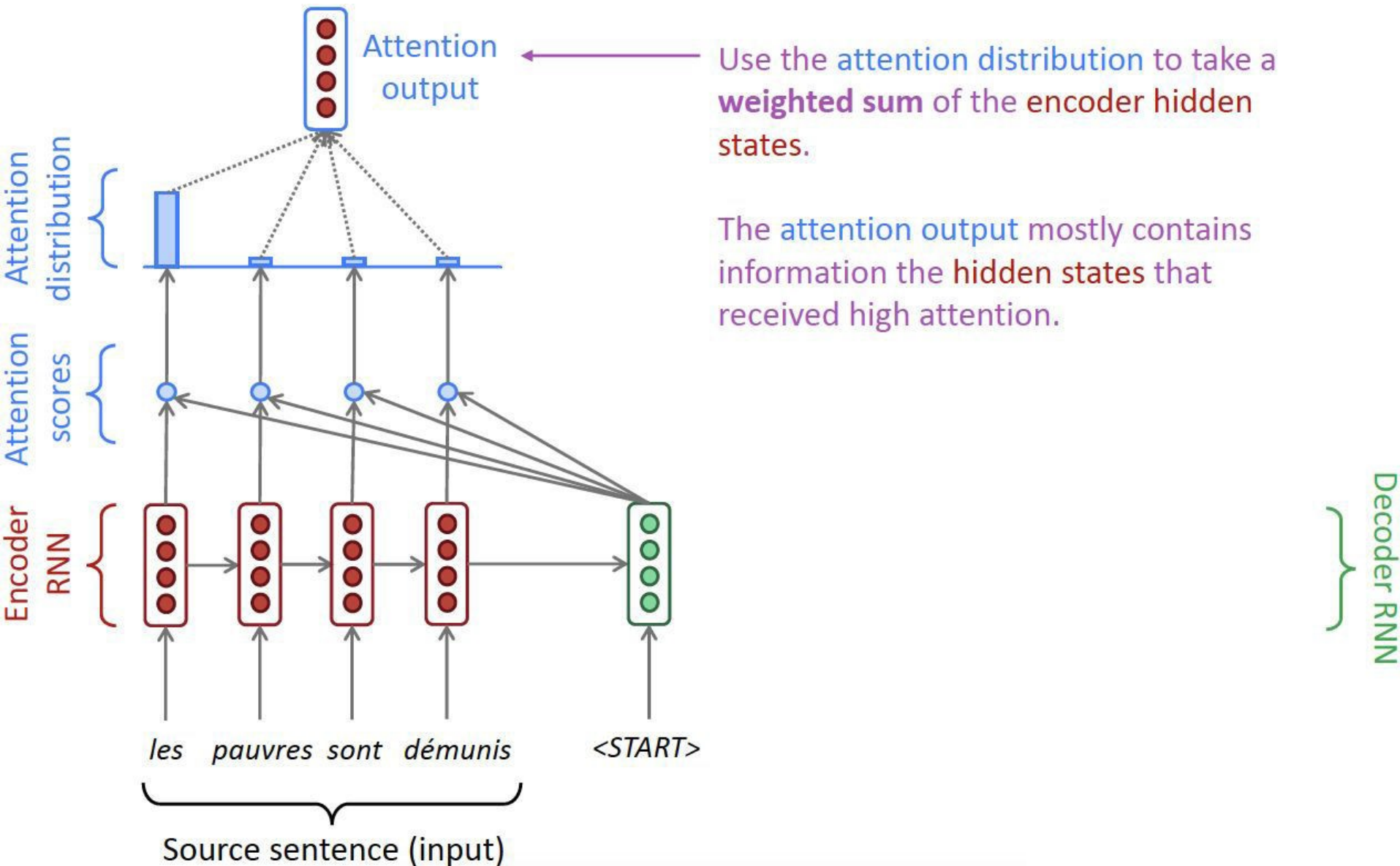
Seq2Seq with Attention



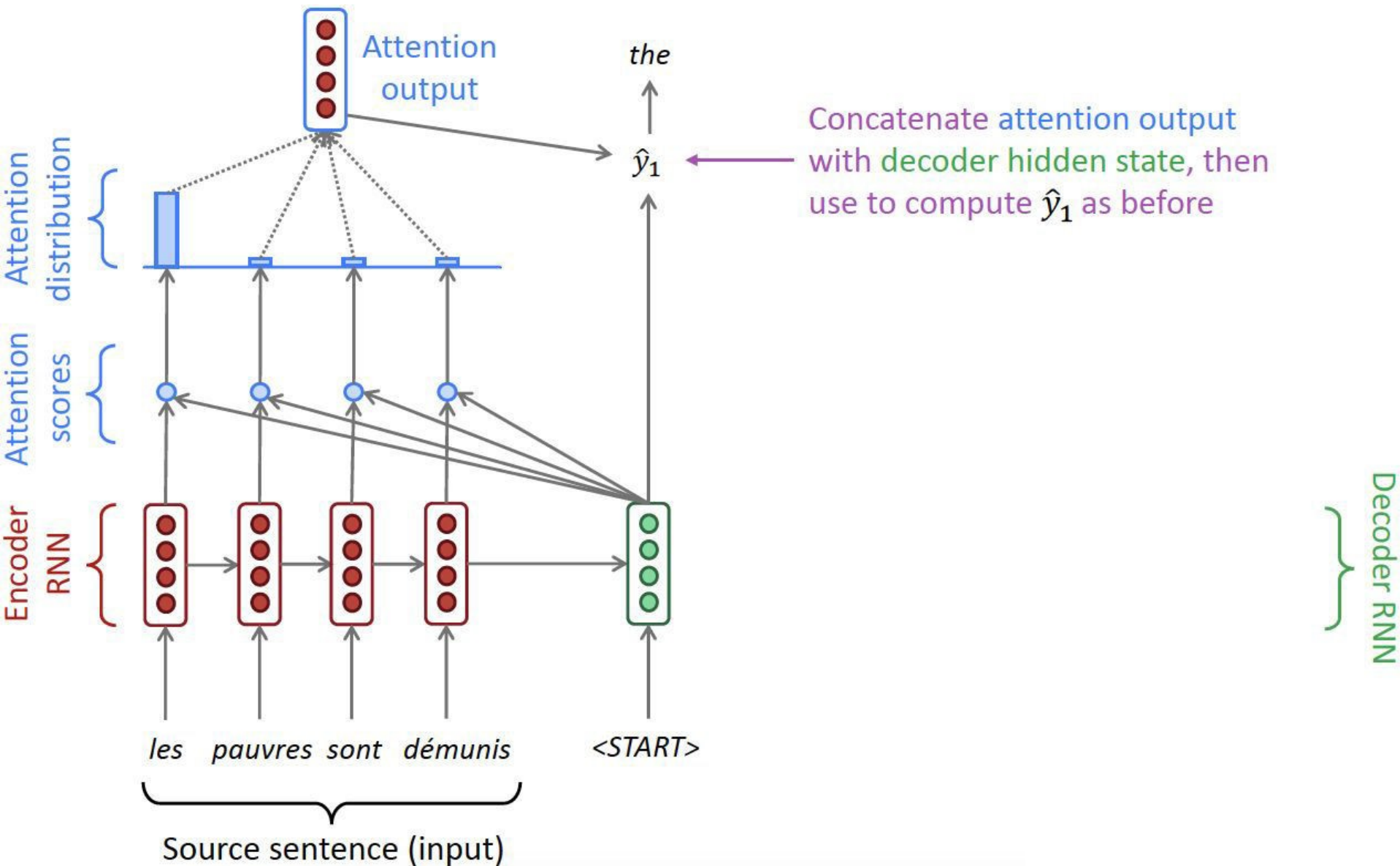
Seq2Seq with Attention



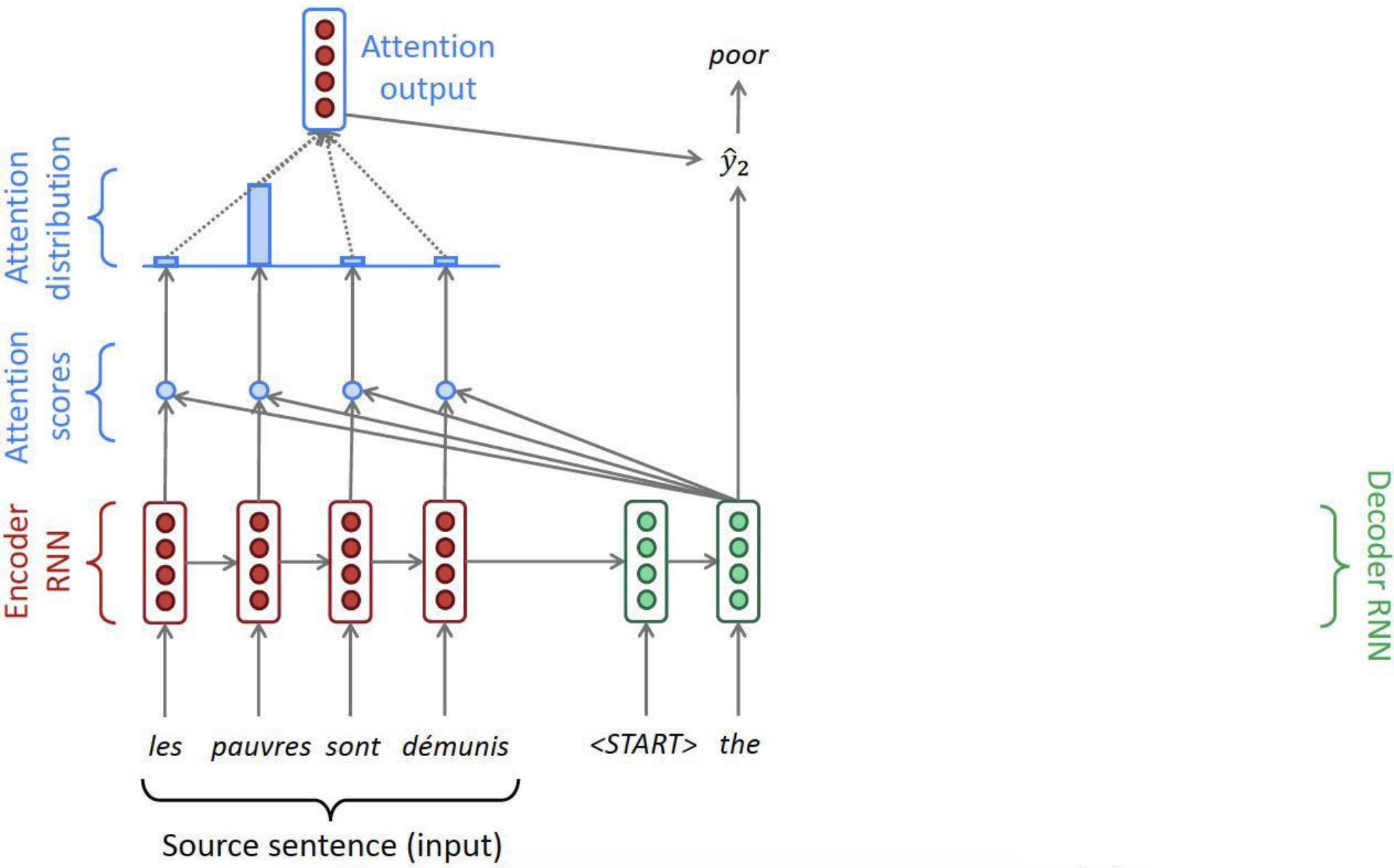
Seq2Seq with Attention



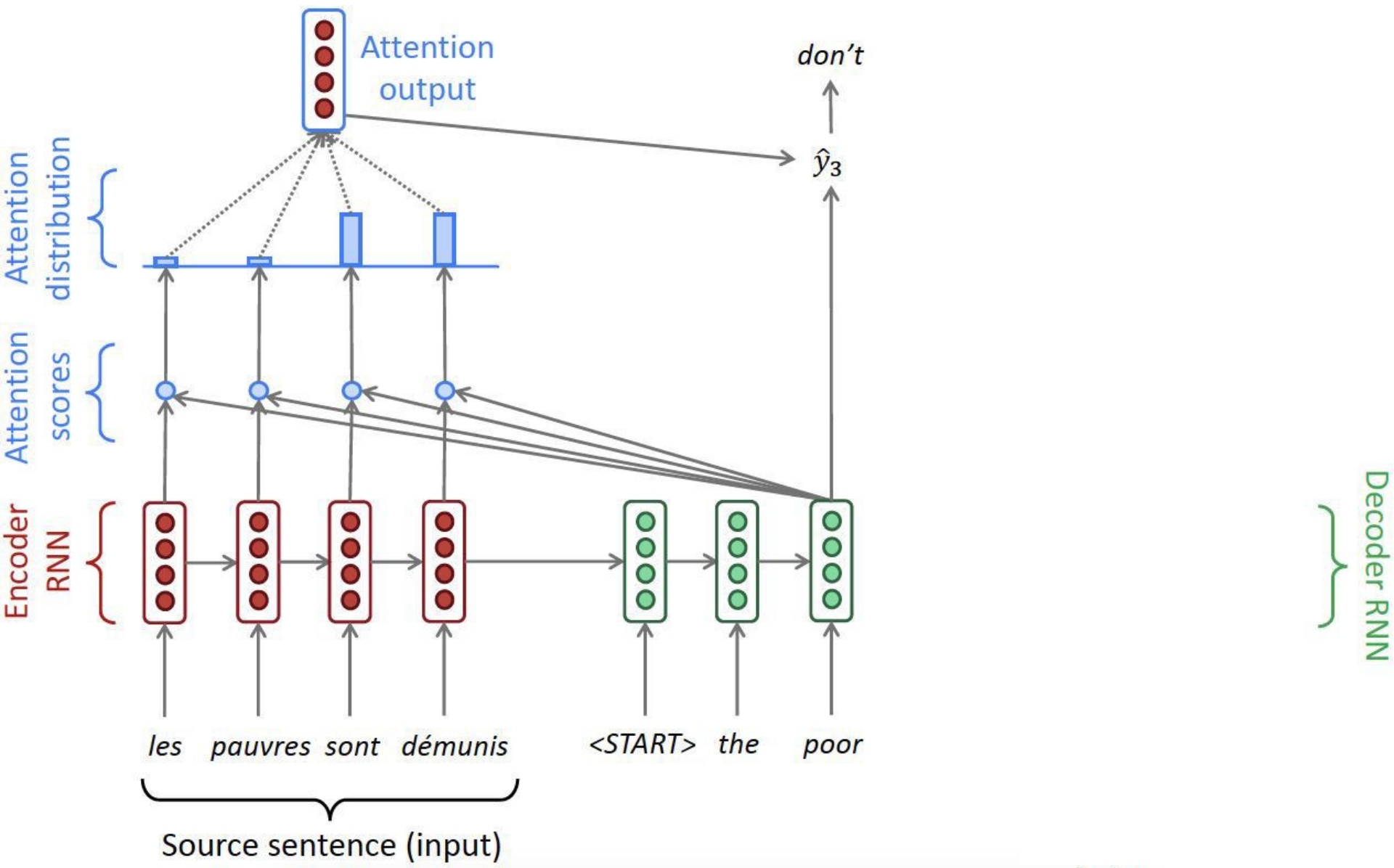
Seq2Seq with Attention



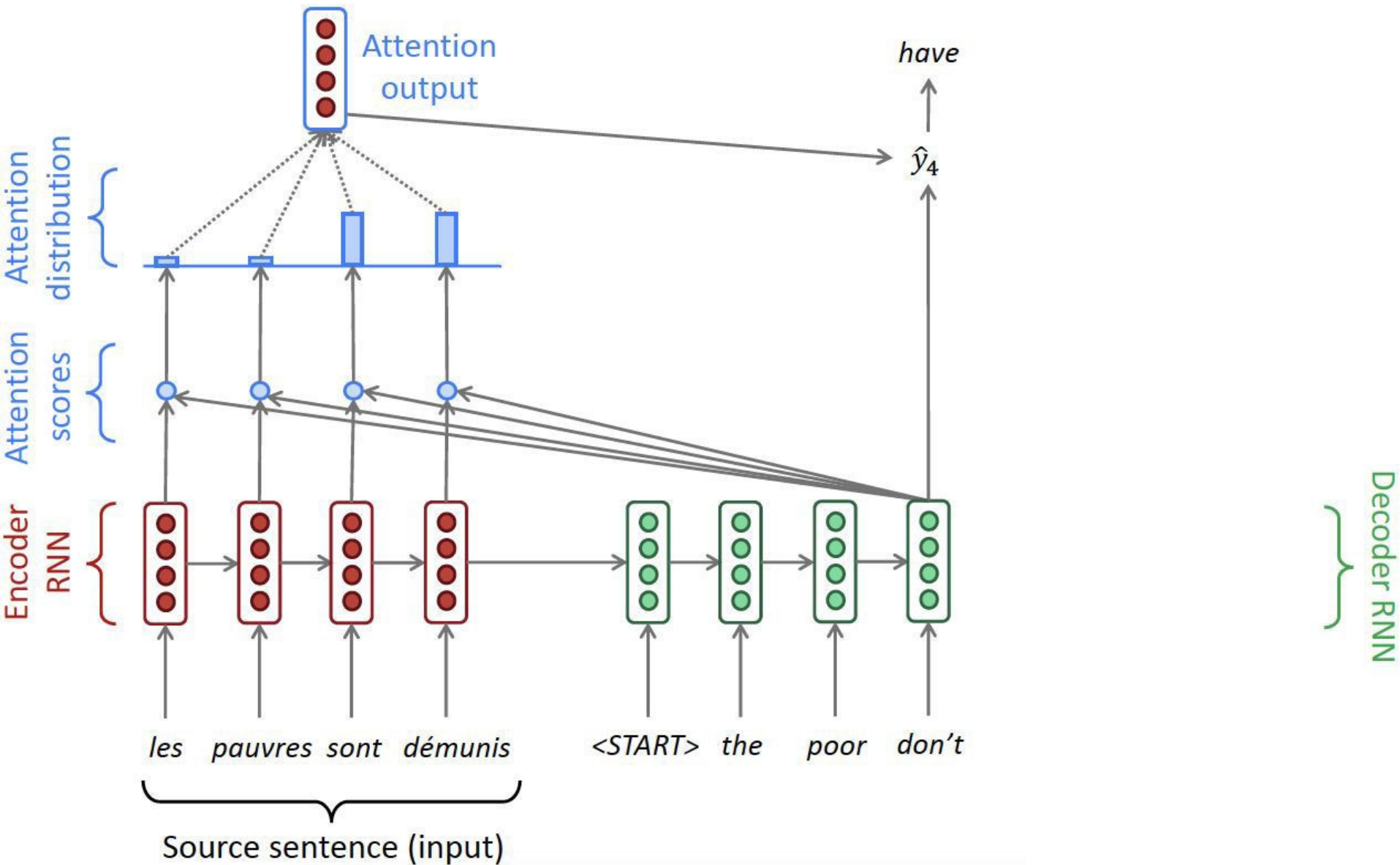
Seq2Seq with Attention



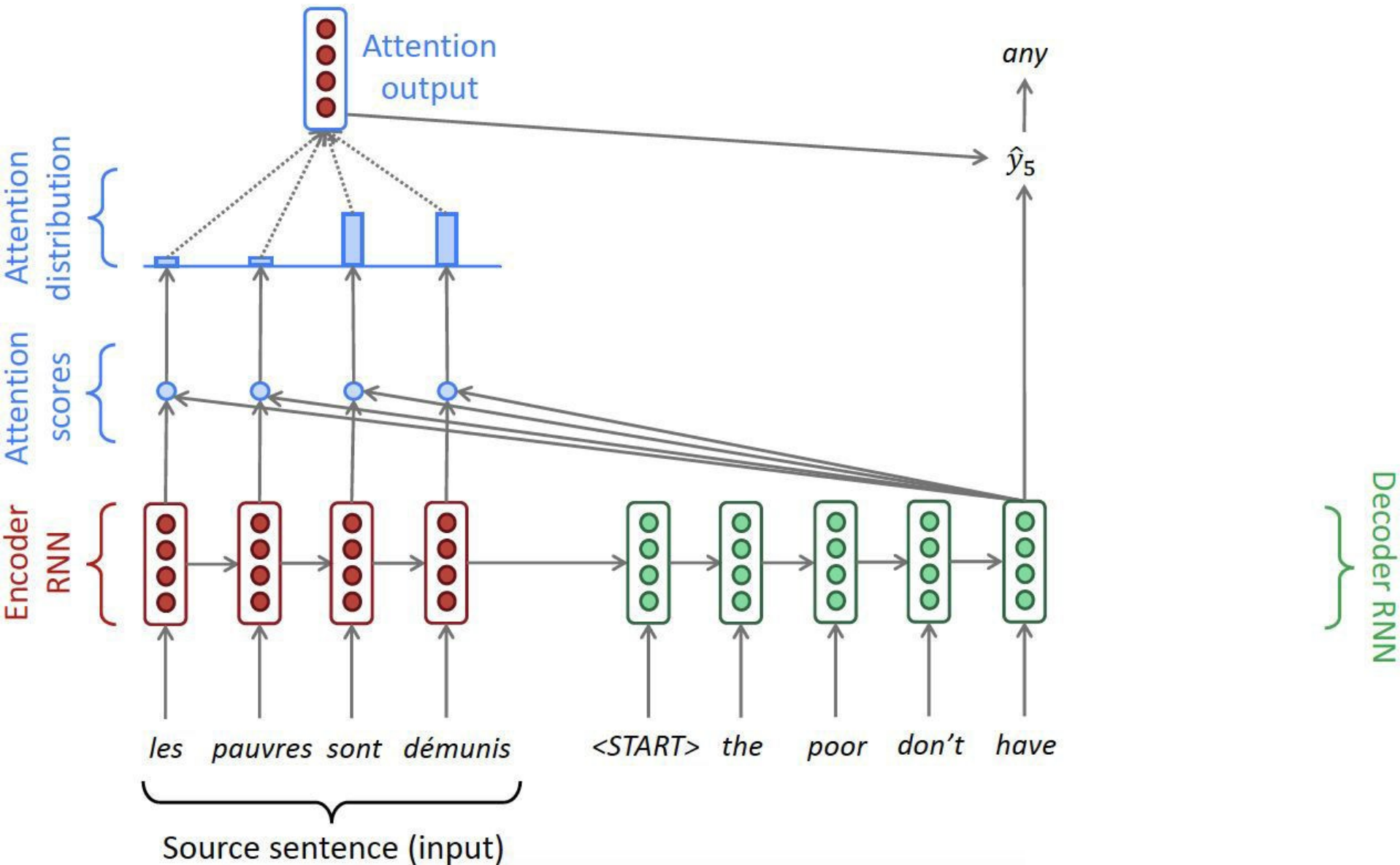
Seq2Seq with Attention



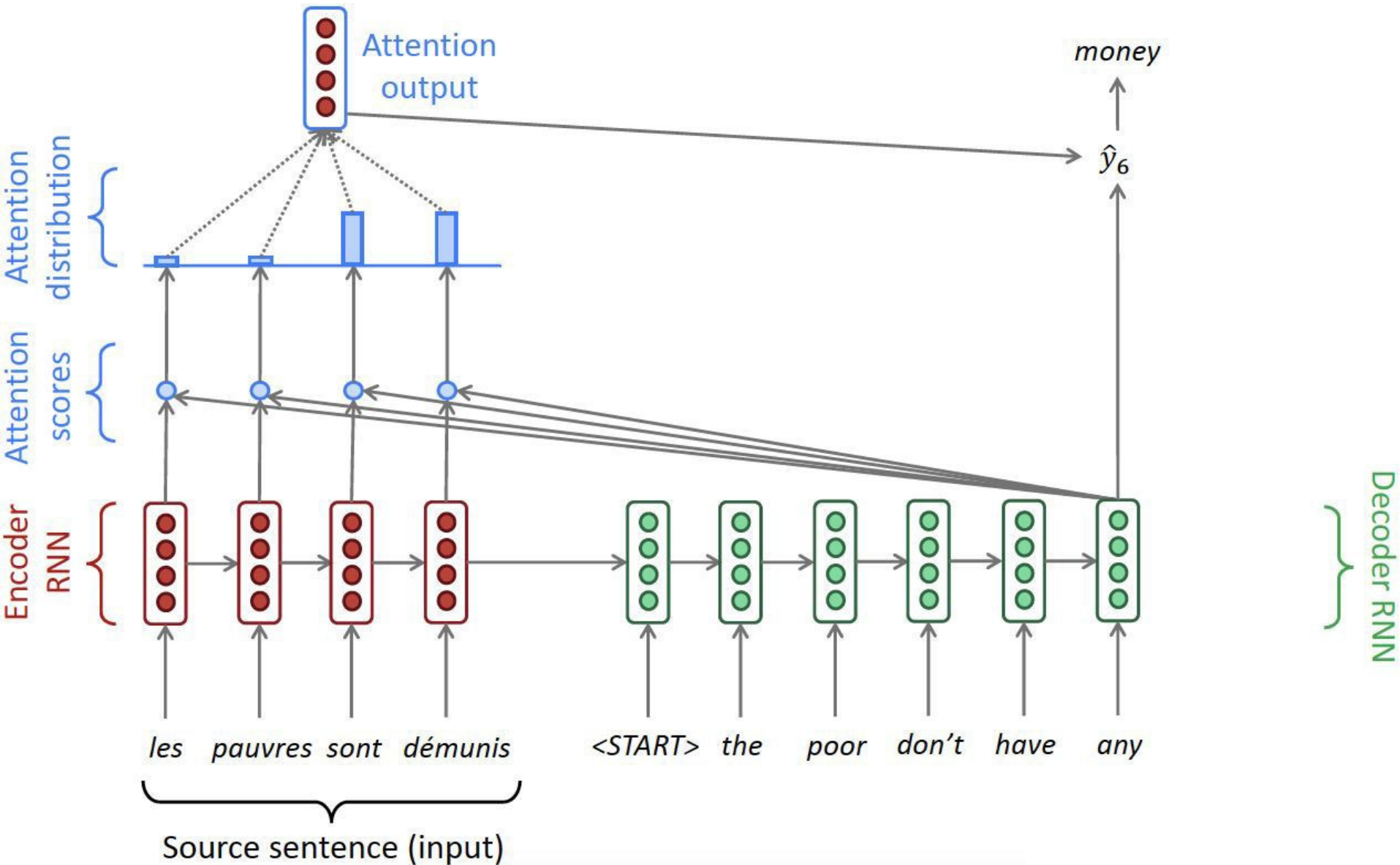
Seq2Seq with Attention



Seq2Seq with Attention



Seq2Seq with Attention



Attention: in equations

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

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

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

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

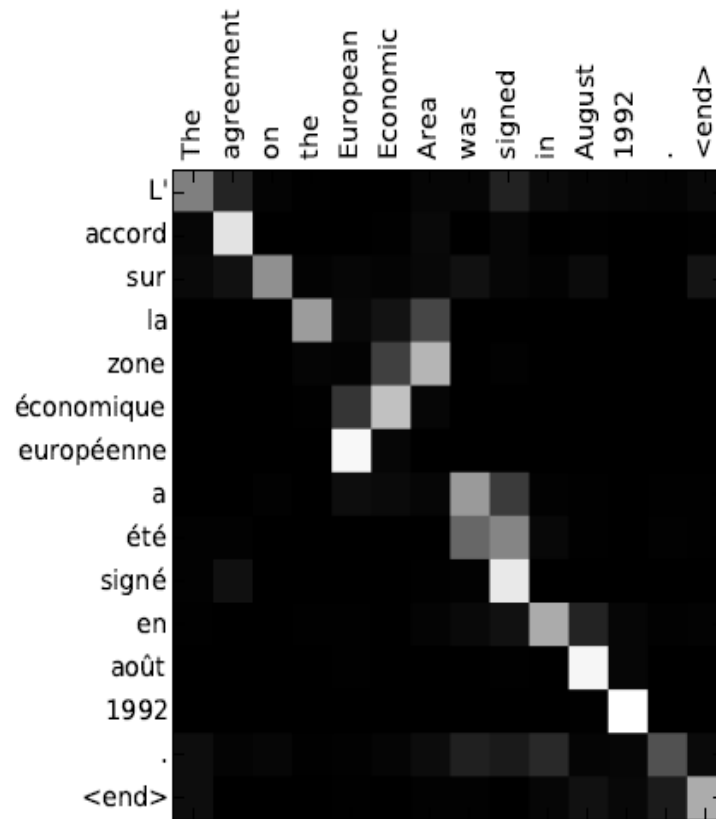
- We use α^t to take a weighted sum of the encoder hidden states to get the attention output a_t

$$a_t = \sum_{i=1}^N \alpha_i^t h_i \in \mathbb{R}^h$$

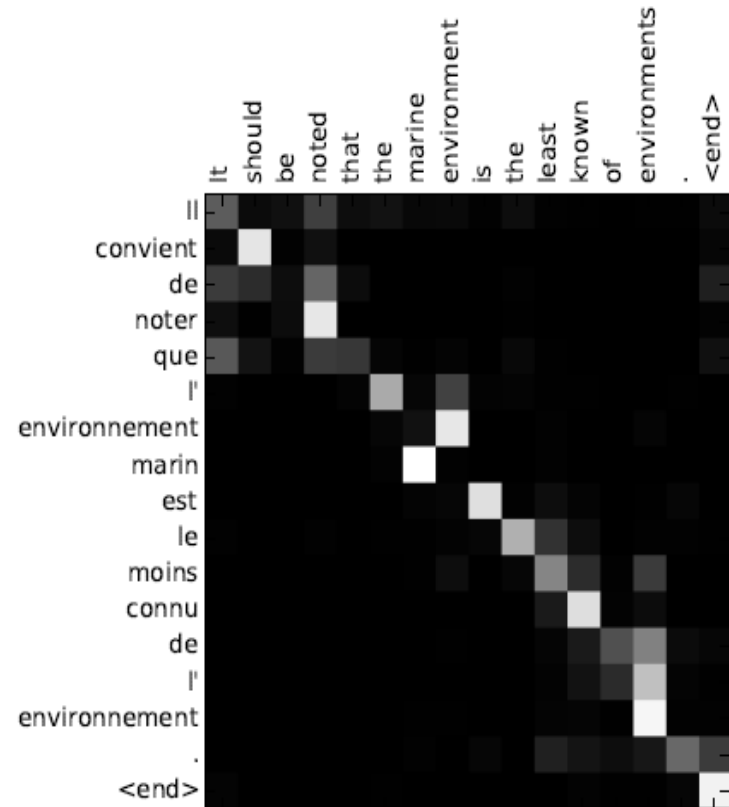
- Finally we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[a_t; s_t] \in \mathbb{R}^{2h}$$

Soft Attention for Translation



(a)




(b)

Bahdanau et al, "Neural Machine Translation by Jointly Learning to Align and Translate", ICLR 2015

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



	Les	pauvres	sont	démunis
The	■			
poor		■		
don't			■	■
have			■	■
any			■	■
money			■	■

Soft vs Hard Attention Models

- **Soft attention:**

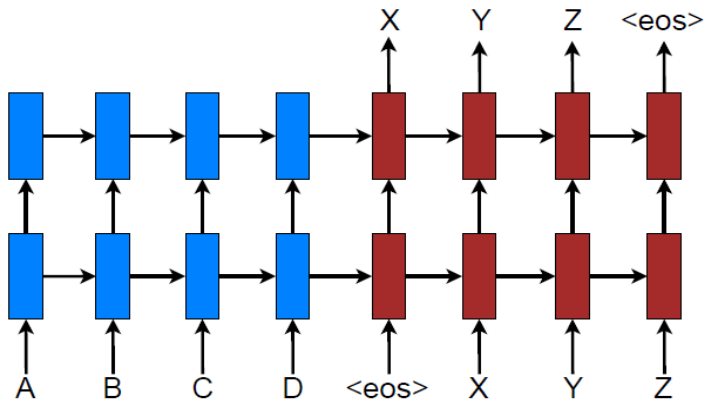
- Compute a weighted combination (attention) over some inputs using an attention network.
- Can use backpropagation to train end-to-end.

- **Hard attention:**

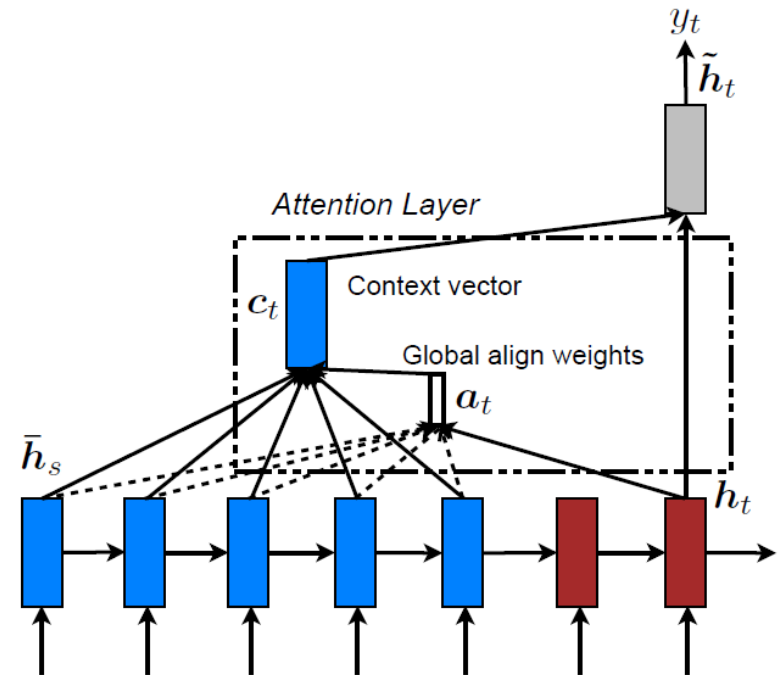
- Attend to a single input location.
- Can't use gradient descent.
- Need **reinforcement learning**.

Global Attention Model

Stacked LSTM with arbitrary depth



- Compute a best aligned position p_t first
- Then compute a context vector centered at that position



Effective Approaches to Attention-based Neural Machine Translation
Minh-Thang Luong, Hieu Pham, Christopher D. Manning, EMNLP 15

Attention-only Translation Models

- Problems with recurrent networks:
- **Sequential training and inference**: time grows in proportion to sentence length. Hard to parallelize.
- **Long-range dependencies** have to be remembered across many single time steps.
- **Tricky to learn hierarchical structures** (“car”, “blue car”, “into the blue car” ...)
- Alternative:
- Convolution – but has other limitations.

Self-attention

- An attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence.
- Information flows from within the same subnetwork (either encoder or decoder).
- Convolution applies fixed transform weights. Self-attention applies variable weights

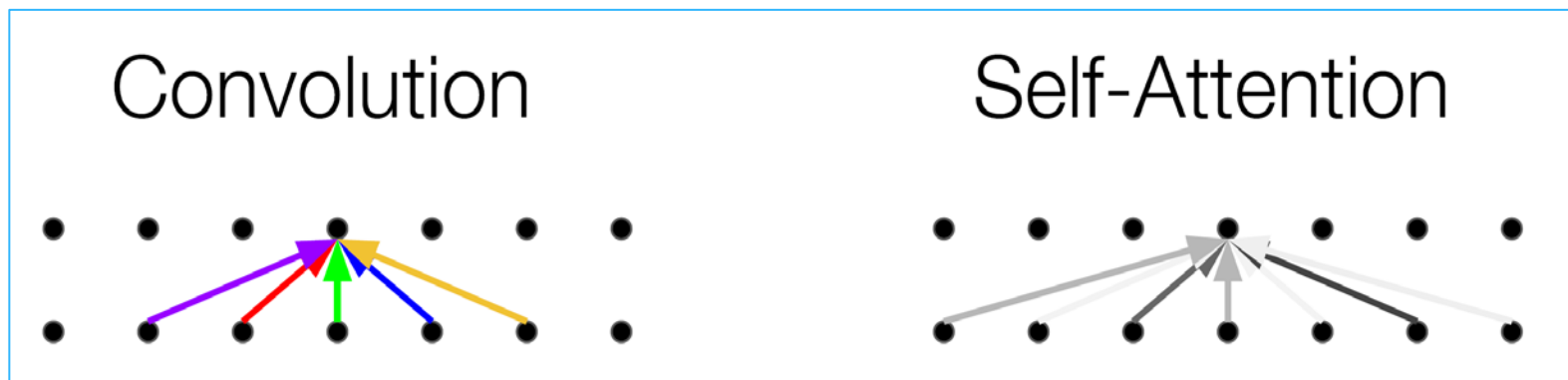


image from Lukas Kaiser, Stanford NLP seminar

Self-Attention “Transformers”

- the Transformer performs a small, constant number of steps (chosen empirically).
- In each step, it applies a self-attention mechanism which directly models relationships between all words in a sentence.
- “I arrived at the bank after crossing the river”: to determine that the word “bank” refers to the shore of a river, the Transformer can learn to immediately attend to the word “river”.
- To compute the next representation for a given word - “bank” - the Transformer compares it to every other word in the sentence. The result of these comparisons is an attention score for every other word in the sentence. These attention scores determine how much each of the other words should contribute to the next representation of “bank”
- The Transformer starts by generating initial representations for each word. Then, using self-attention, it aggregates information from all of the other words, generating a new representation per word informed by the entire context. This step is then repeated multiple times in parallel for all words, successively generating new representations.

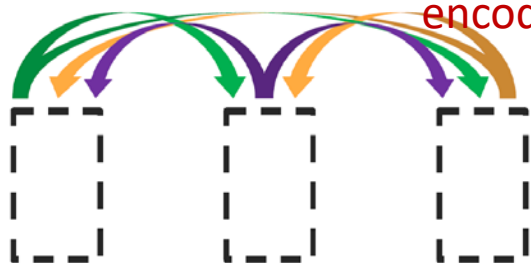


Attention in Transformer Networks

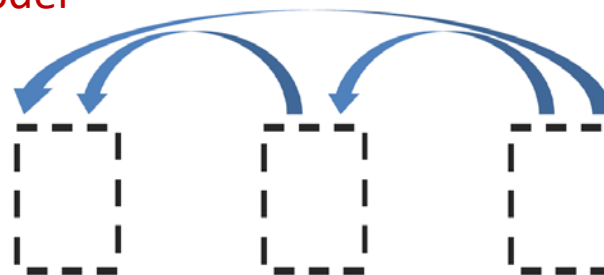


Encoder-Decoder Attention

Replaces word recurrence in
encoder and decoder



Encoder Self-Attention



Masked Decoder Self-Attention

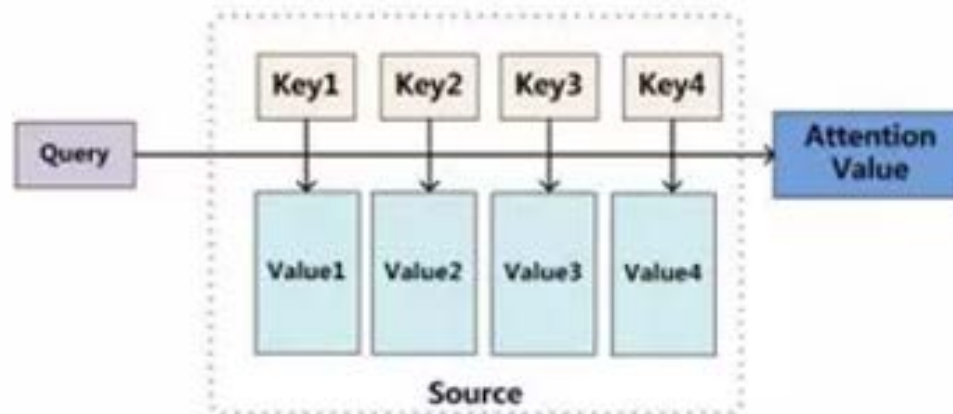
Masking limits attention to earlier units:
 y_i depends only on y_j for $j < i$.

image from Lukas Kaiser, Stanford NLP seminar

Flow of Information

- we can visualize what other parts of a sentence the network attends to when processing or translating a given word, thus gaining insights into how information travels through the network.

- An attention function can be described as **mapping a query and a set of key-value pairs to an output.**



$$\text{Attention}(\text{Query}, \text{Source}) = \sum_{i=1}^L \text{Similarity}(\text{Query}, \text{Key}_i) * \text{Value}_i$$

www.aifababacloud.com

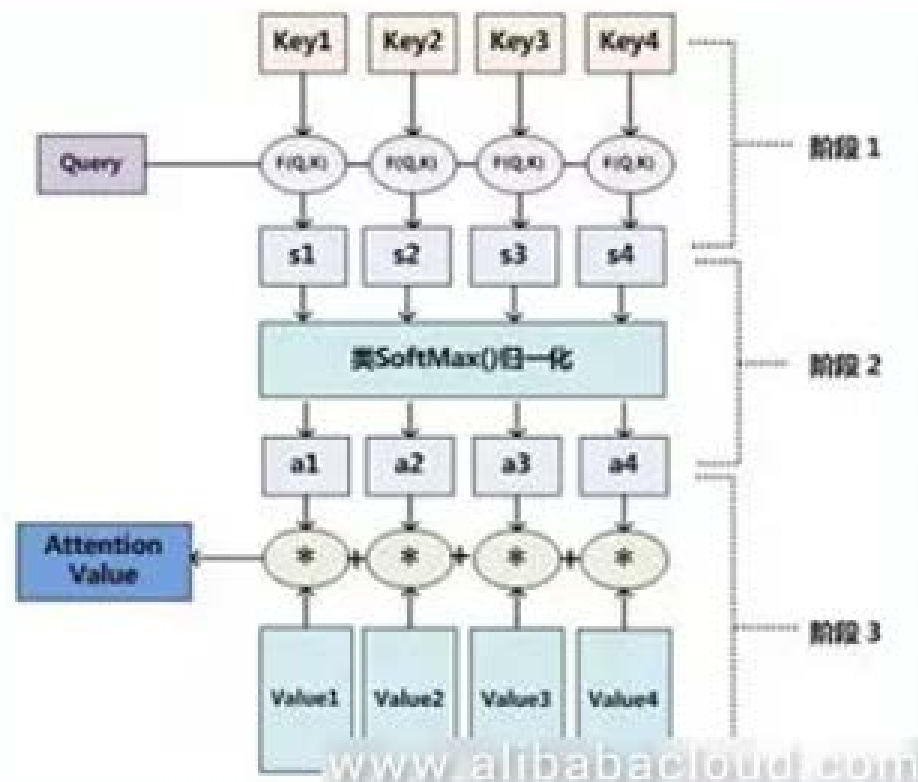
Calculate attention in three steps.

1. we take the query and each key and compute the similarity between the two to obtain a weight.
2. Use a softmax function to normalize these weights
3. weight these weights in conjunction with the corresponding values and obtain the final Attention.

$$f(Q, K_i) = \begin{cases} Q^T K_i & \text{dot} \\ Q^T W_a K_i & \text{general} \\ W_a [Q; K_i] & \text{concat} \\ v_a^T \tanh(W_a Q + U_a K_i) & \text{perceptron} \end{cases}$$

$$a_i = \text{soft max}(f(Q, K_i)) = \frac{\exp(f(Q, K_i))}{\sum_j \exp(f(Q, K_j))}$$

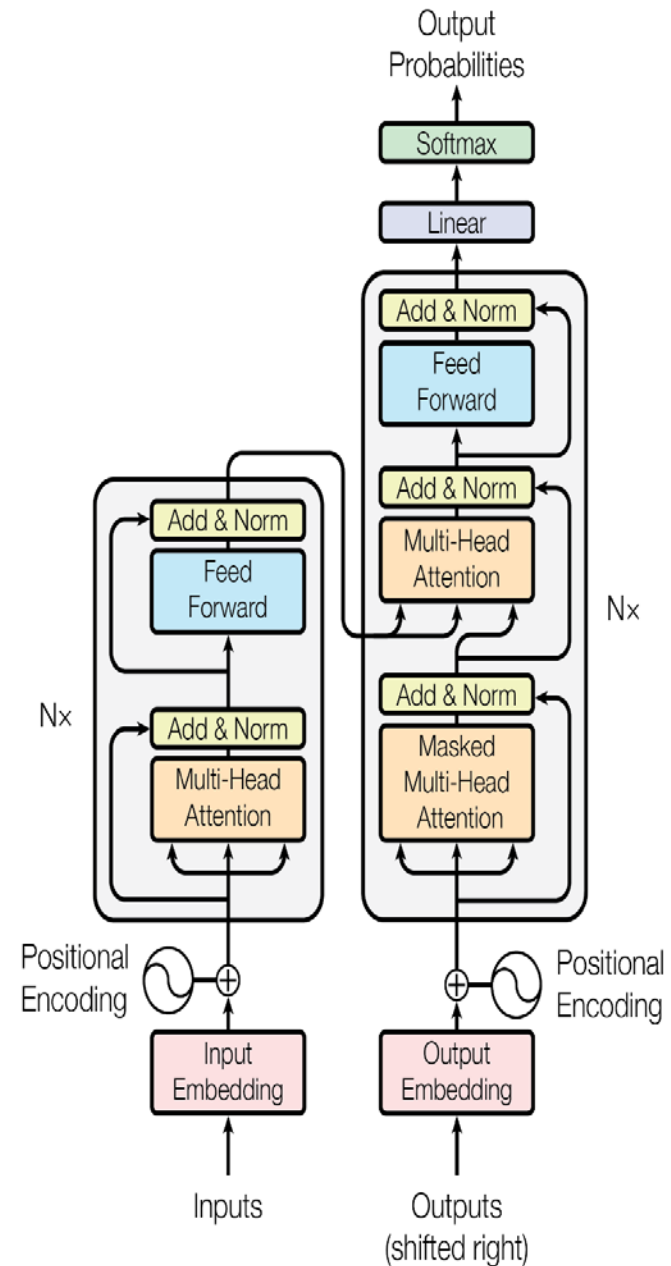
$$\text{Attention}(Q, K, V) = \sum_i a_i V_i$$



In current NLP work, the key and value are frequently the same, therefore key=value

The Transformer

- Encoder and Decoder Stacks
 - Attention
 - Position-wise FF Networks
 - Positional Encoding
 - Add & Norm



Attention Implementation

Scaled Dot-Product Attention

- Attention is modeled as a key-value store:

Q = query vector

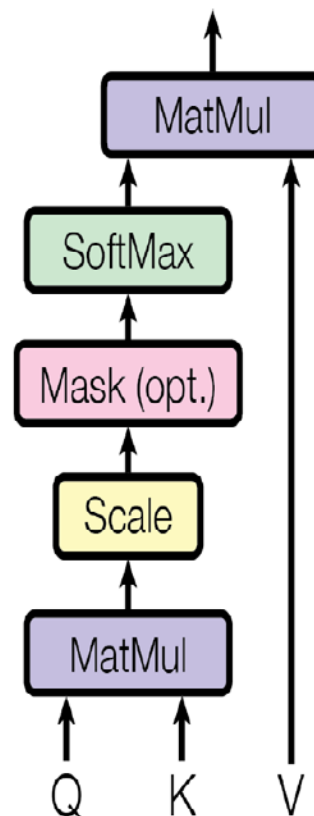
K = key

V = value

Encoder-decoder layer: the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. (Similar to Bahdanau).

Self-attention layer: all of the keys, values and queries come from the output of the previous layer in the encoder.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



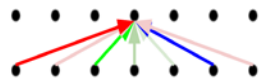
Multi-Headed Attention

- Simple attention blends the results of all the attended-to inputs. It doesn't allow a per-input transformation, as convolution does.
- The solution is to use “multi-headed attention”:

Convolution



Multi-Head Attention

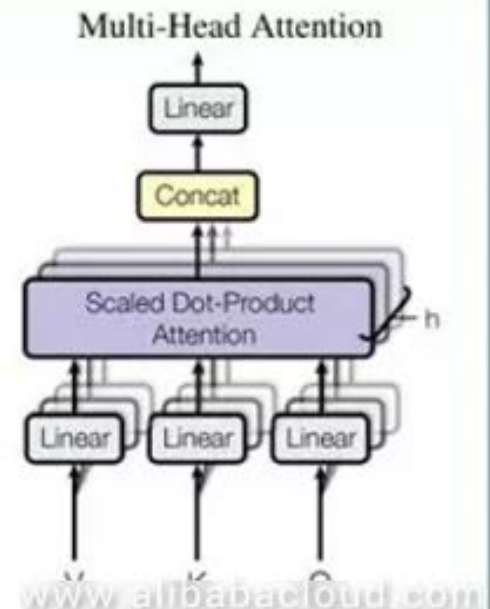


Multi-Head Attention

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

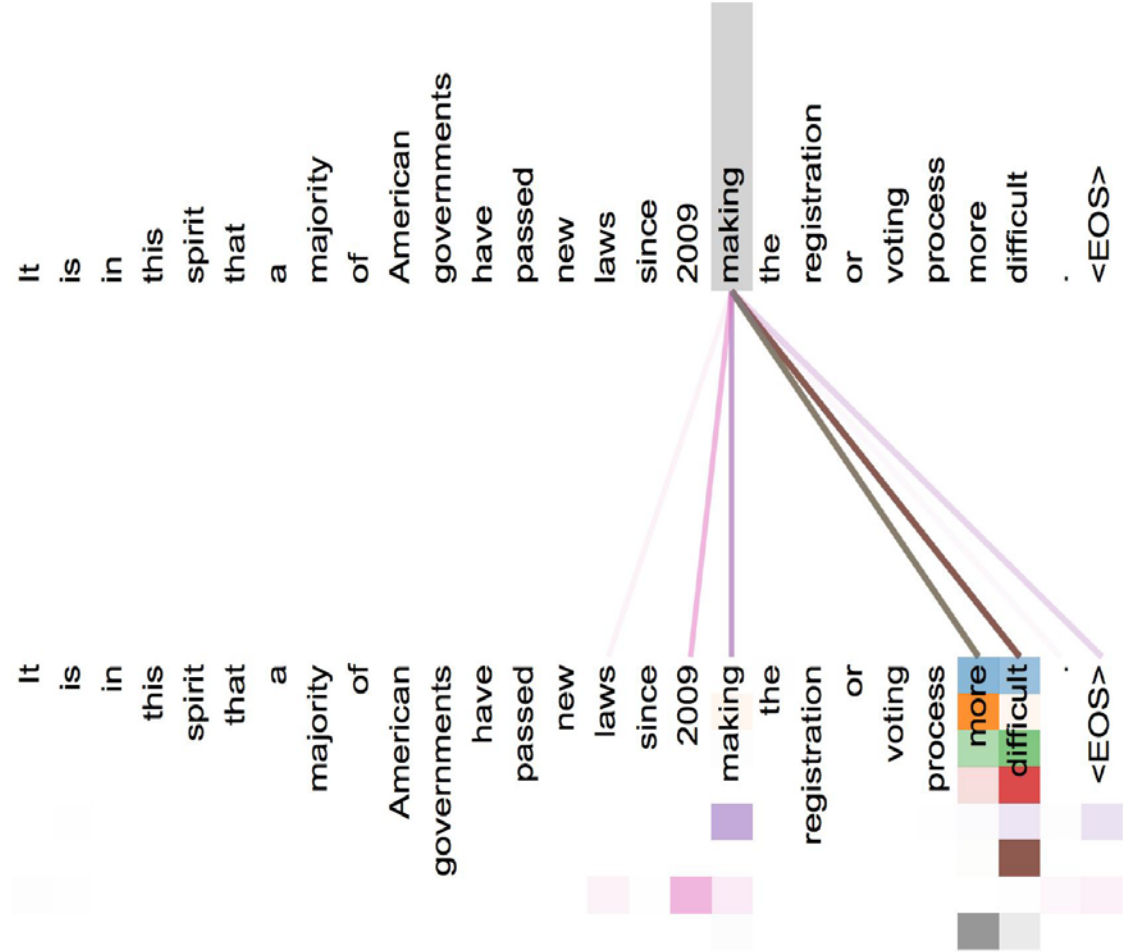
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

- Multi-head attention allows the model to jointly attend to **information from different representation subspaces** at different positions.

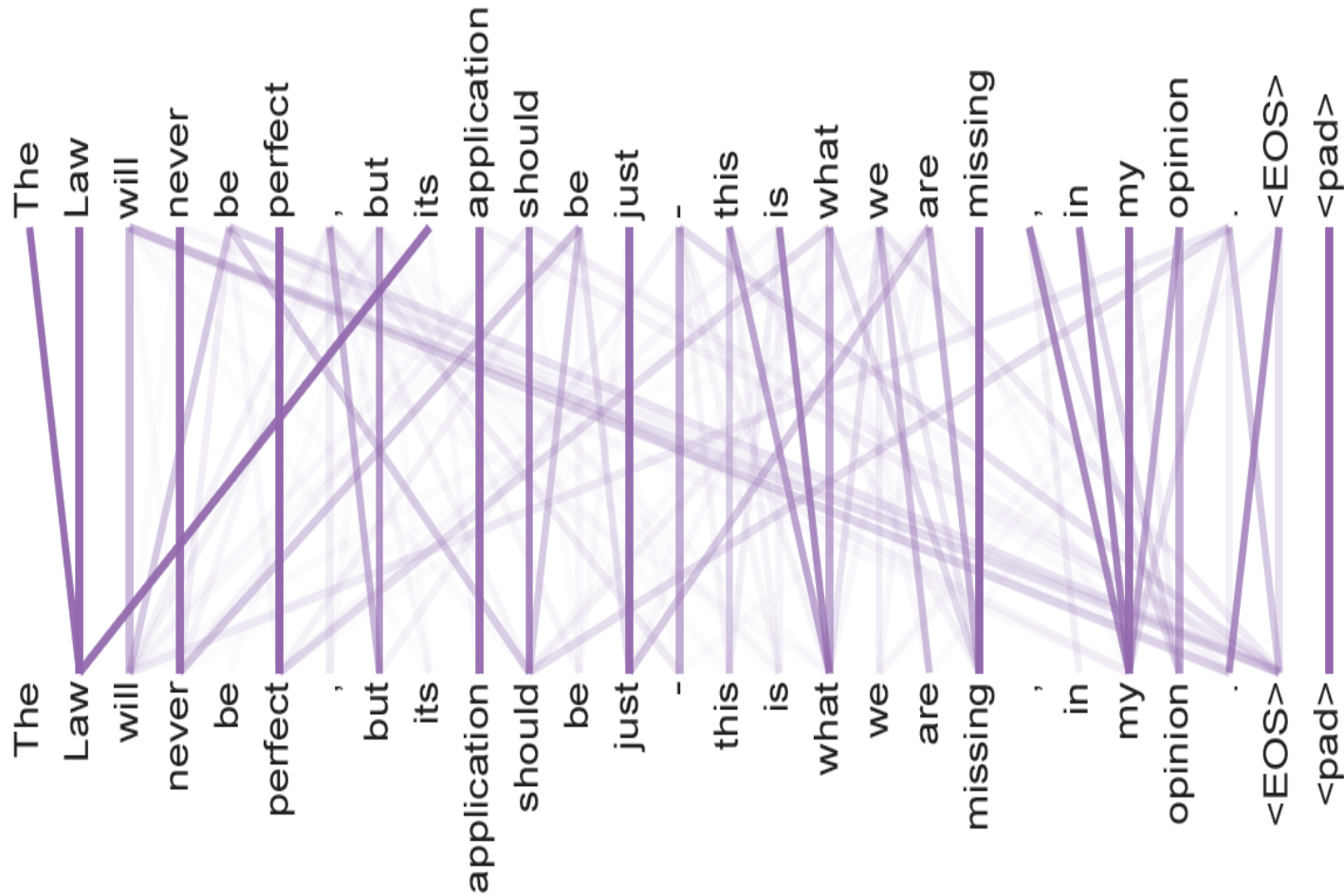


- <https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>
- <http://jalammar.github.io/illustrated-transformer/>

Multi-Headed Attention

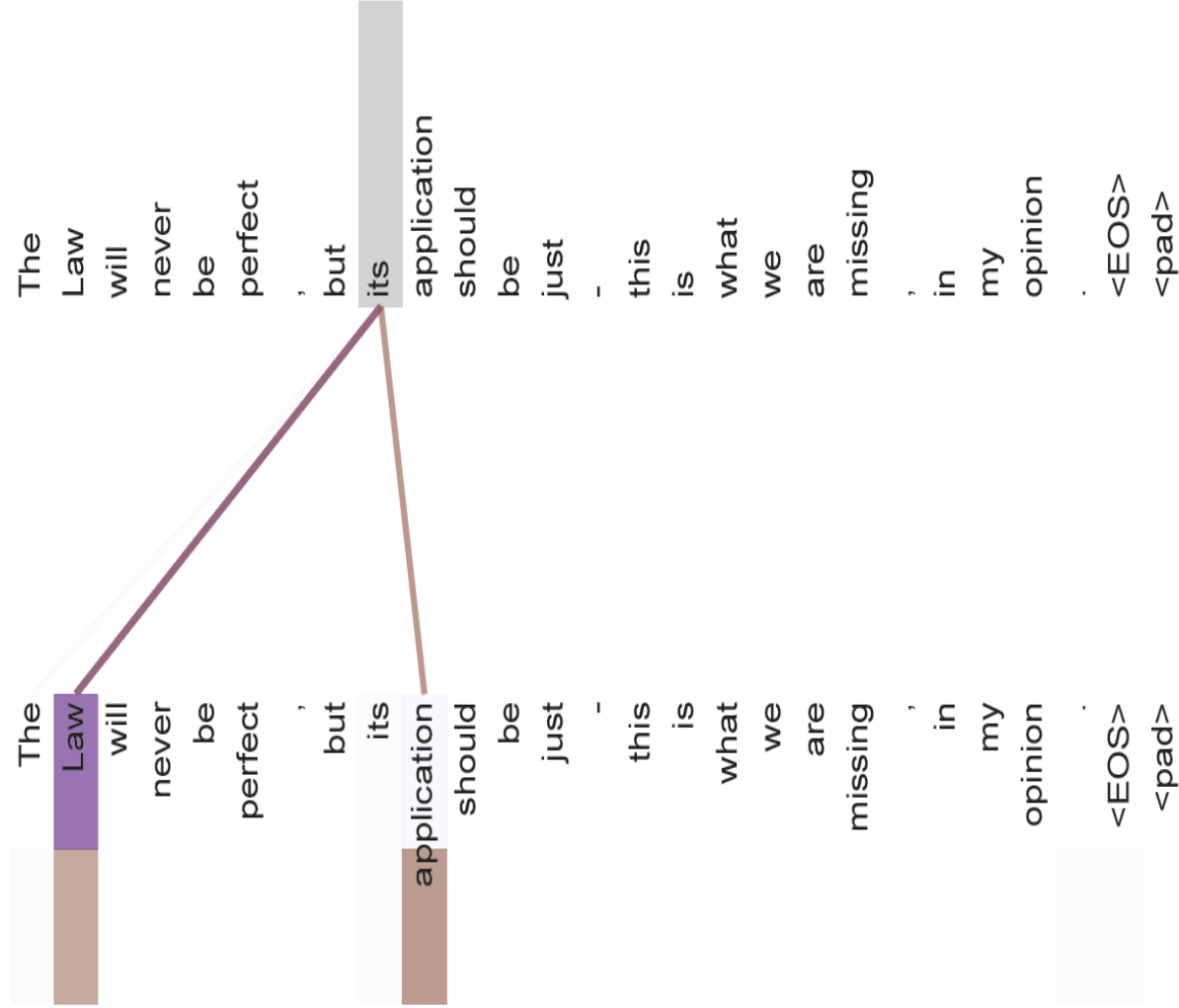


Multi-Headed Attention



Anaphora (pronoun or article) resolution

Multi-Headed Attention



Anaphora (pronoun or article) resolution

Attention and Interpretability

- Attention models learn to predict salient (important) inputs.
- Attention visualizations help users understand the causes of the network's behavior.
- Not every attended region is actually important, but post-processing can remove regions that aren't.

