

Neural Machine Translation with Attention

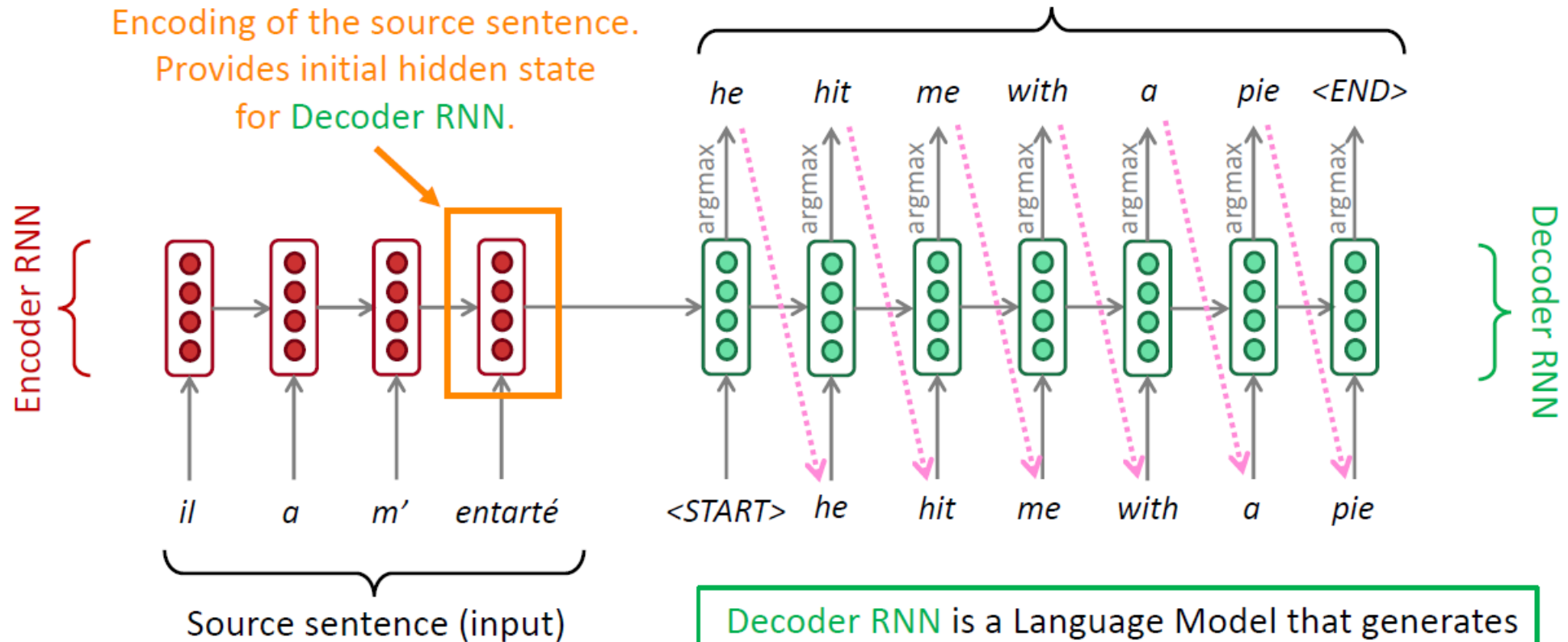


Youtube 영상 주소

- Transformer - <https://youtu.be/Ajt6MKXKOo8>
- Pix2Pix - <https://youtu.be/AsbRMfY4T2U>

Neural Machine Translation (NMT)

The sequence-to-sequence model

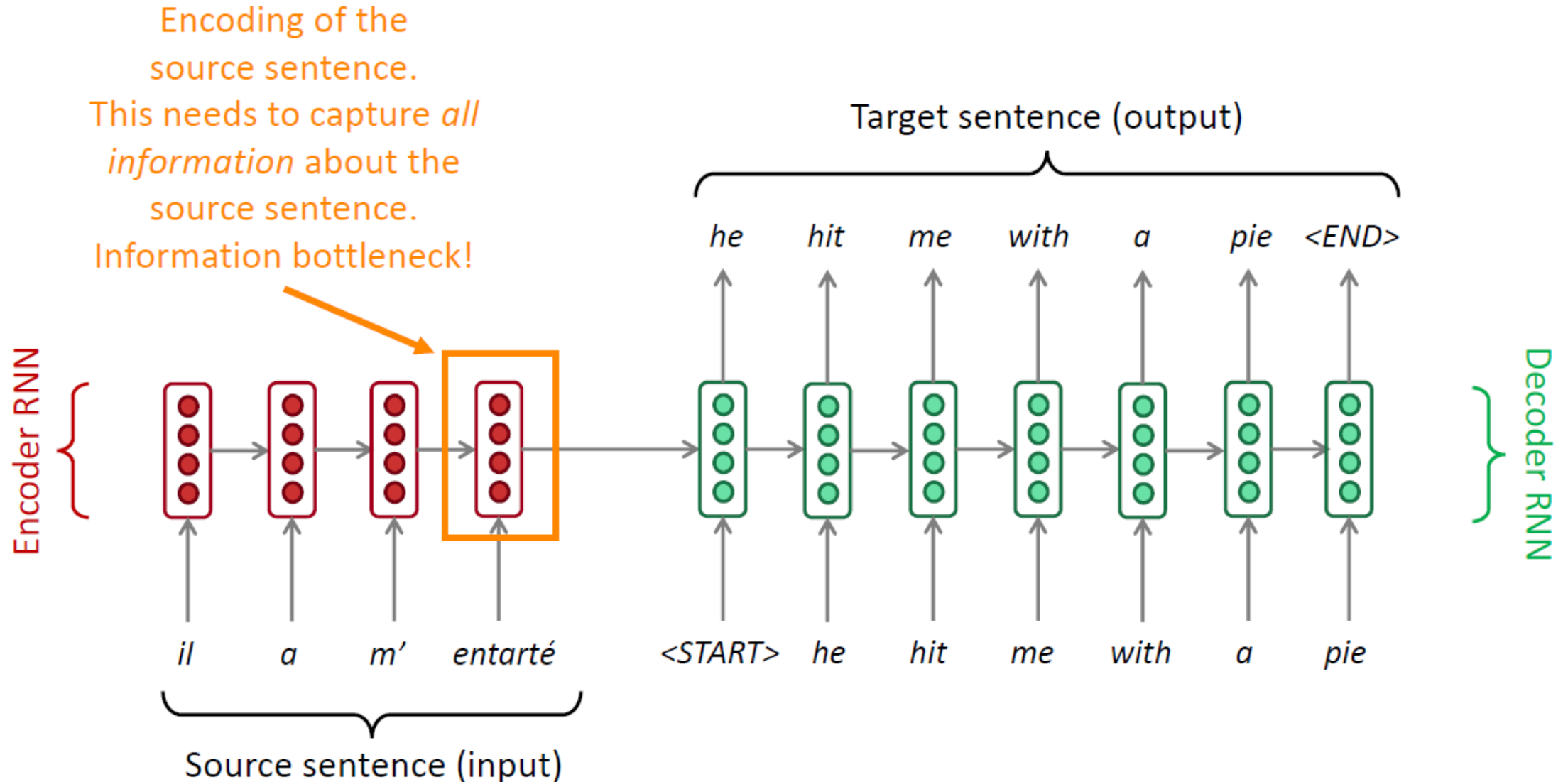


Encoder RNN produces
an **encoding** of the
source sentence.

Decoder RNN is a Language Model that generates
target sentence, *conditioned on encoding*.

Note: This diagram shows **test time** behavior:
decoder output is fed in> as next step's input

Sequence-to-Sequence: the Bottleneck Problem



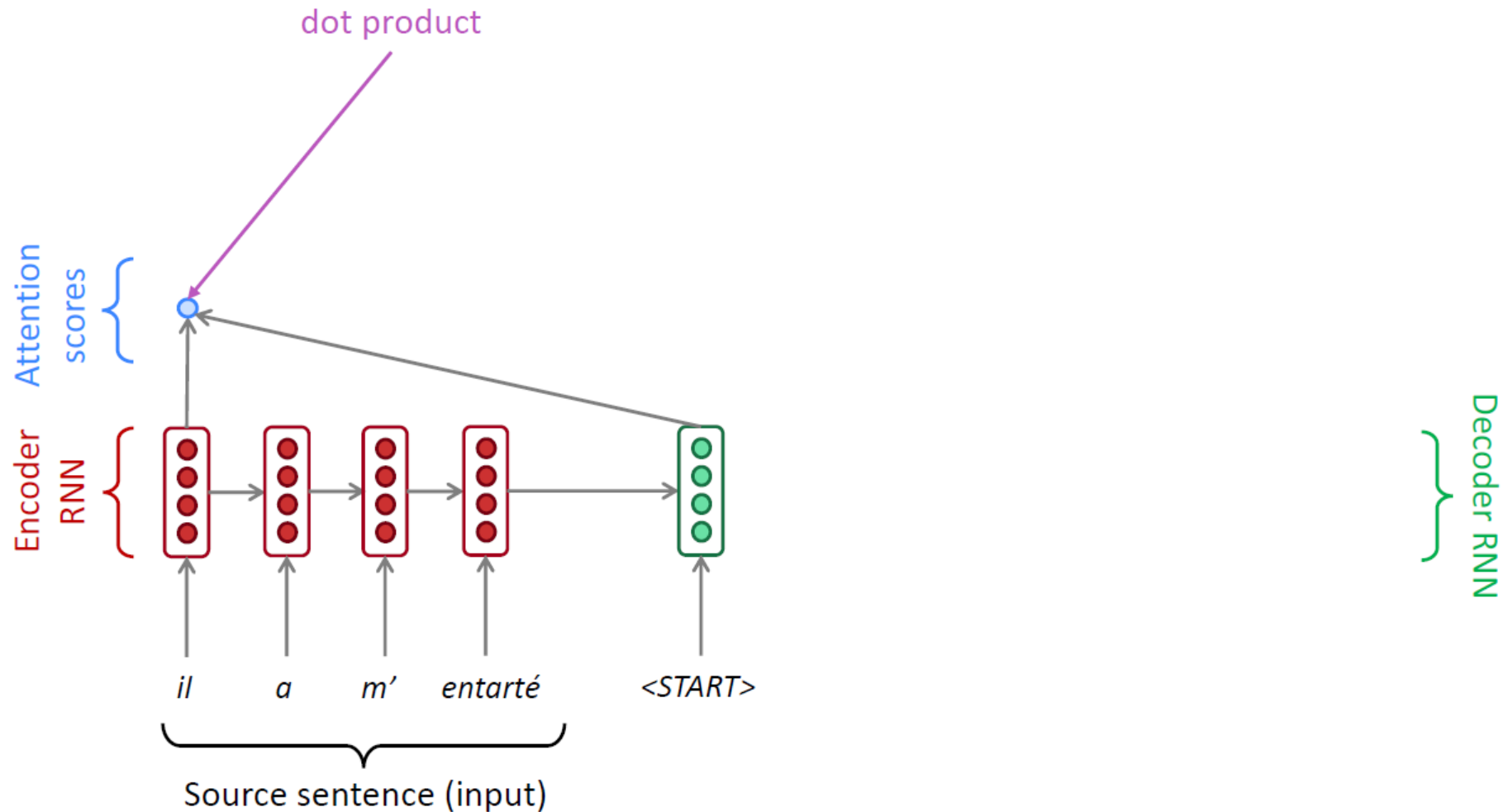
Attention

- **Attention** provides a solution to the bottleneck problem.
- Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

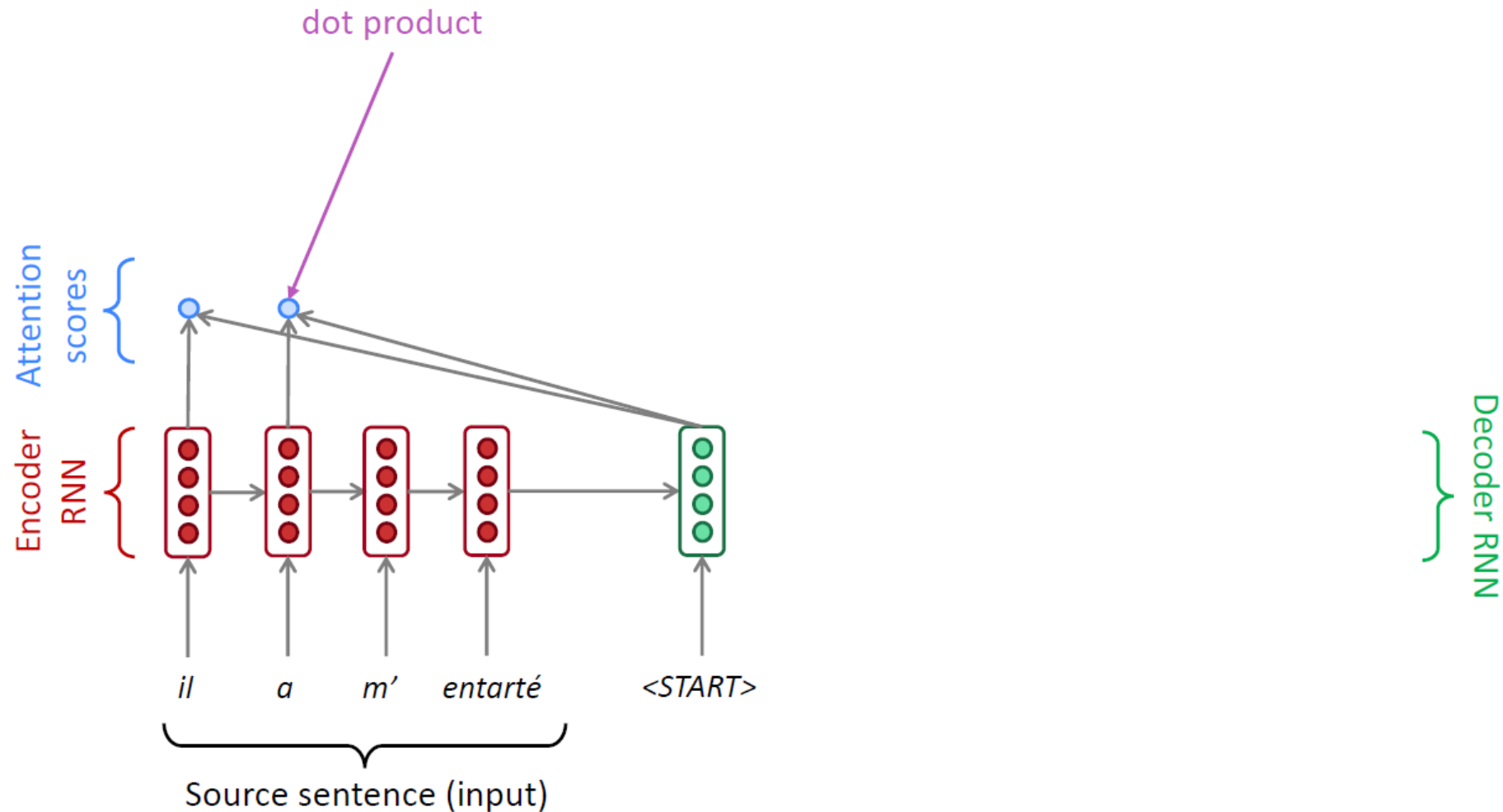


- First we will show via diagram (no equations), then we will show with equations

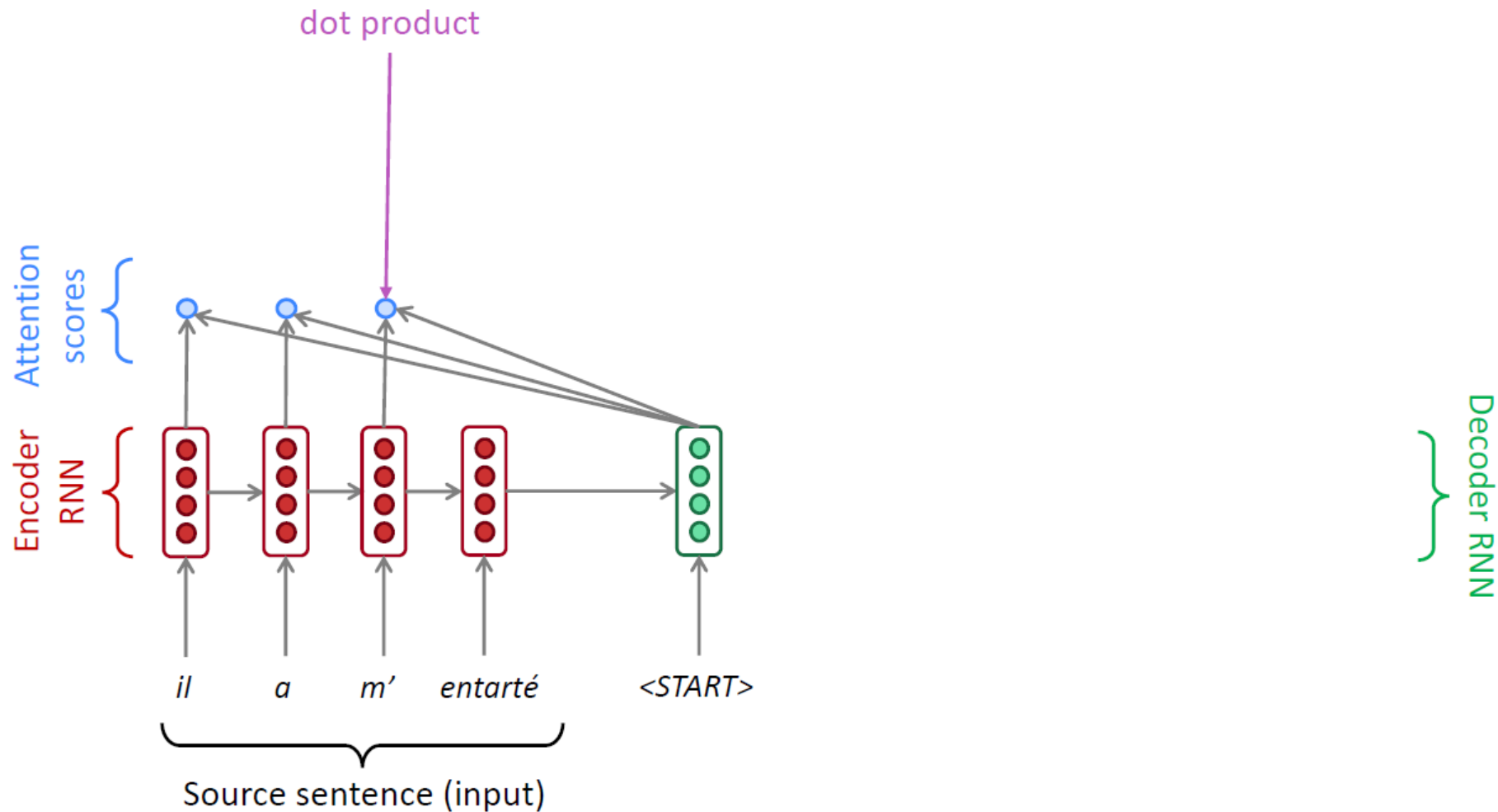
Sequence-to-Sequence with Attention



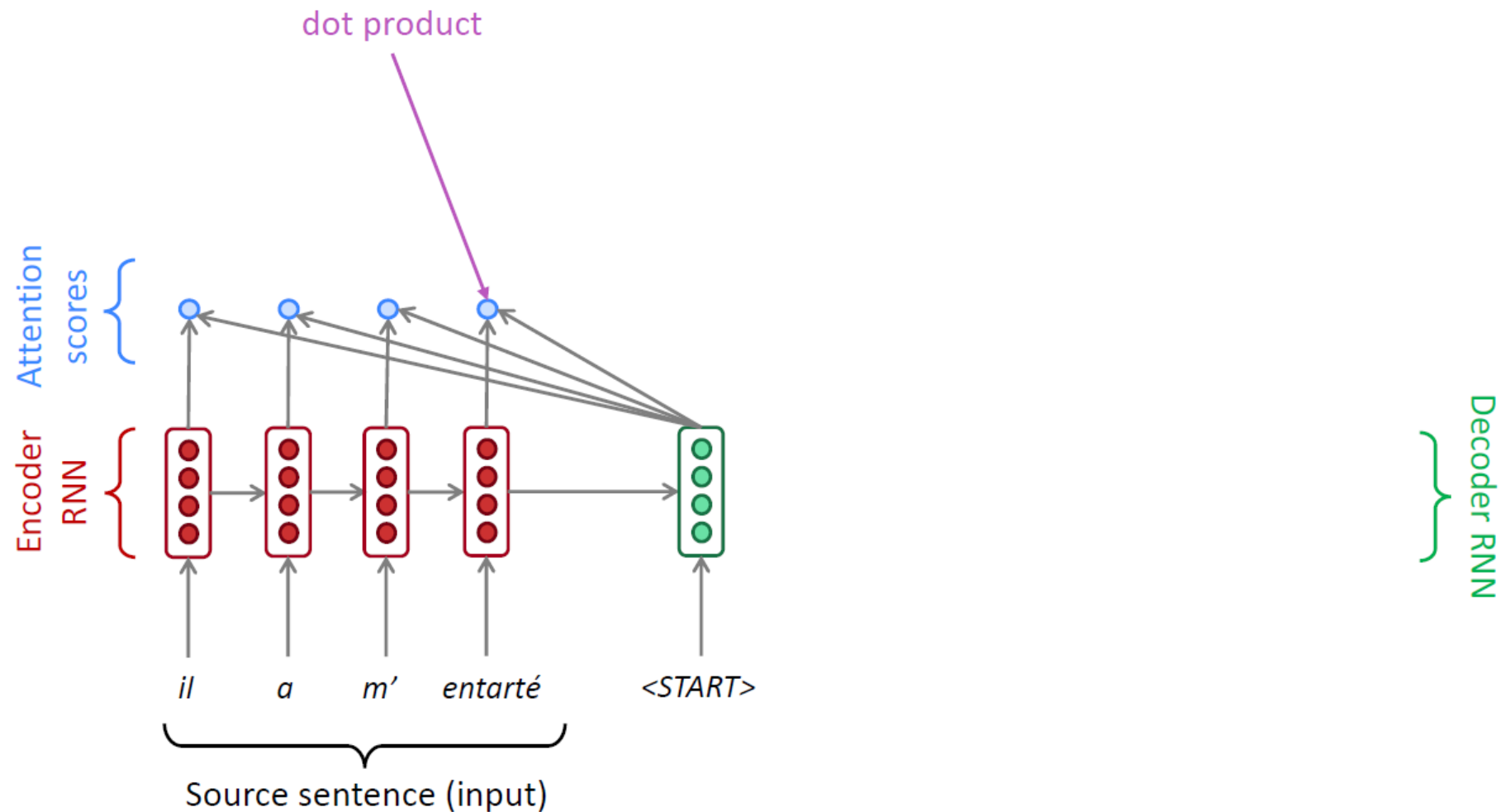
Sequence-to-Sequence with Attention



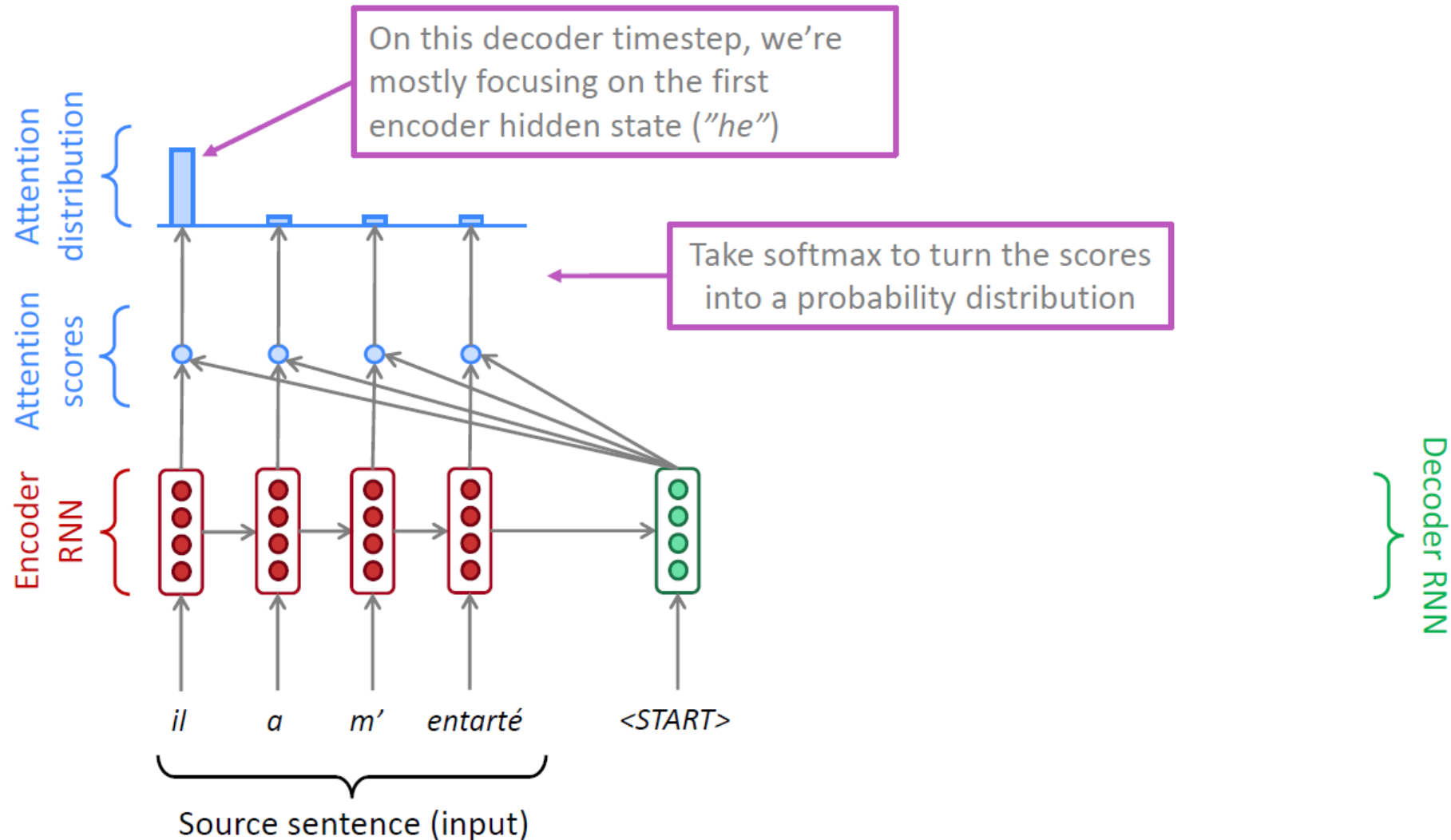
Sequence-to-Sequence with Attention



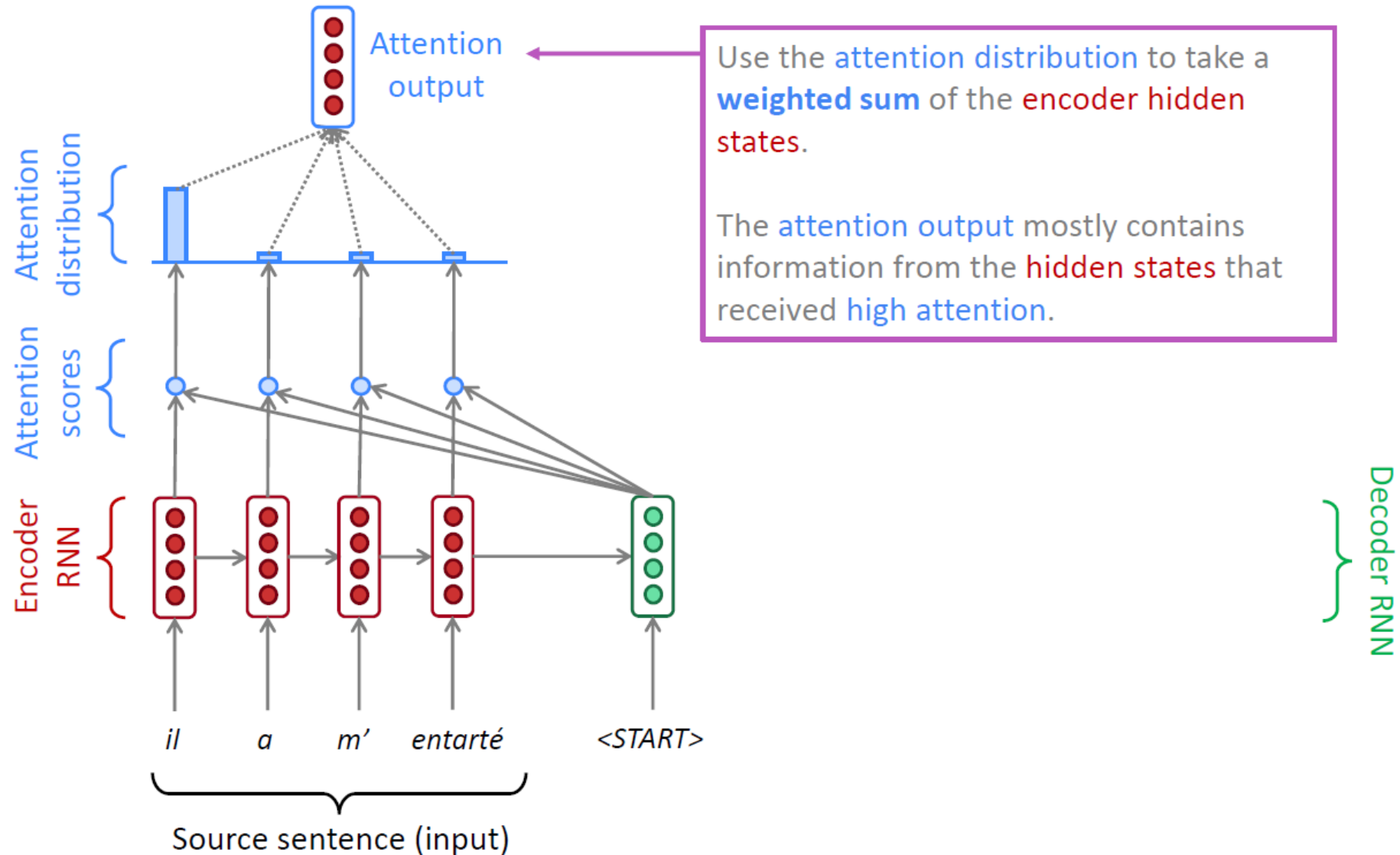
Sequence-to-Sequence with Attention



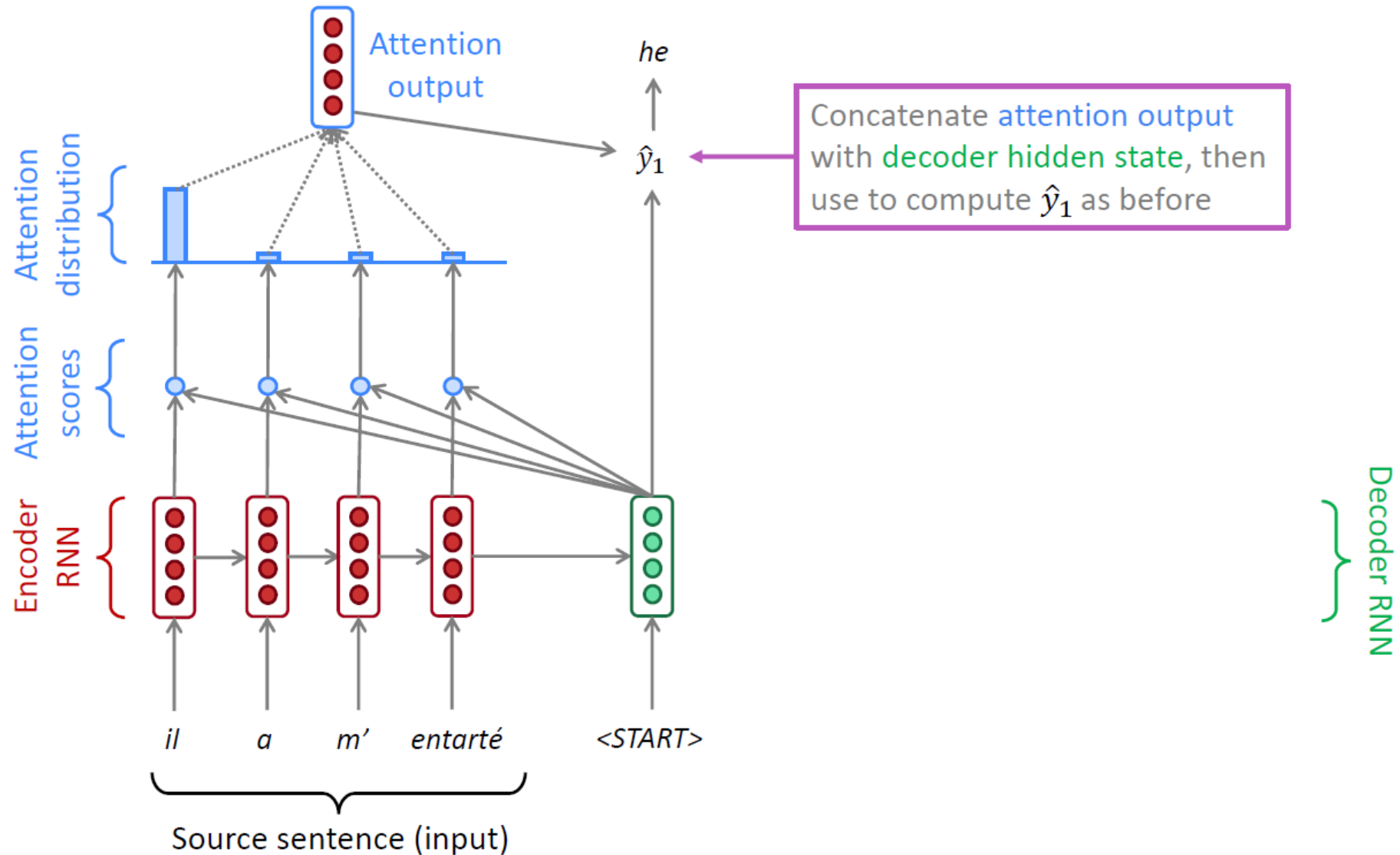
Sequence-to-Sequence with Attention



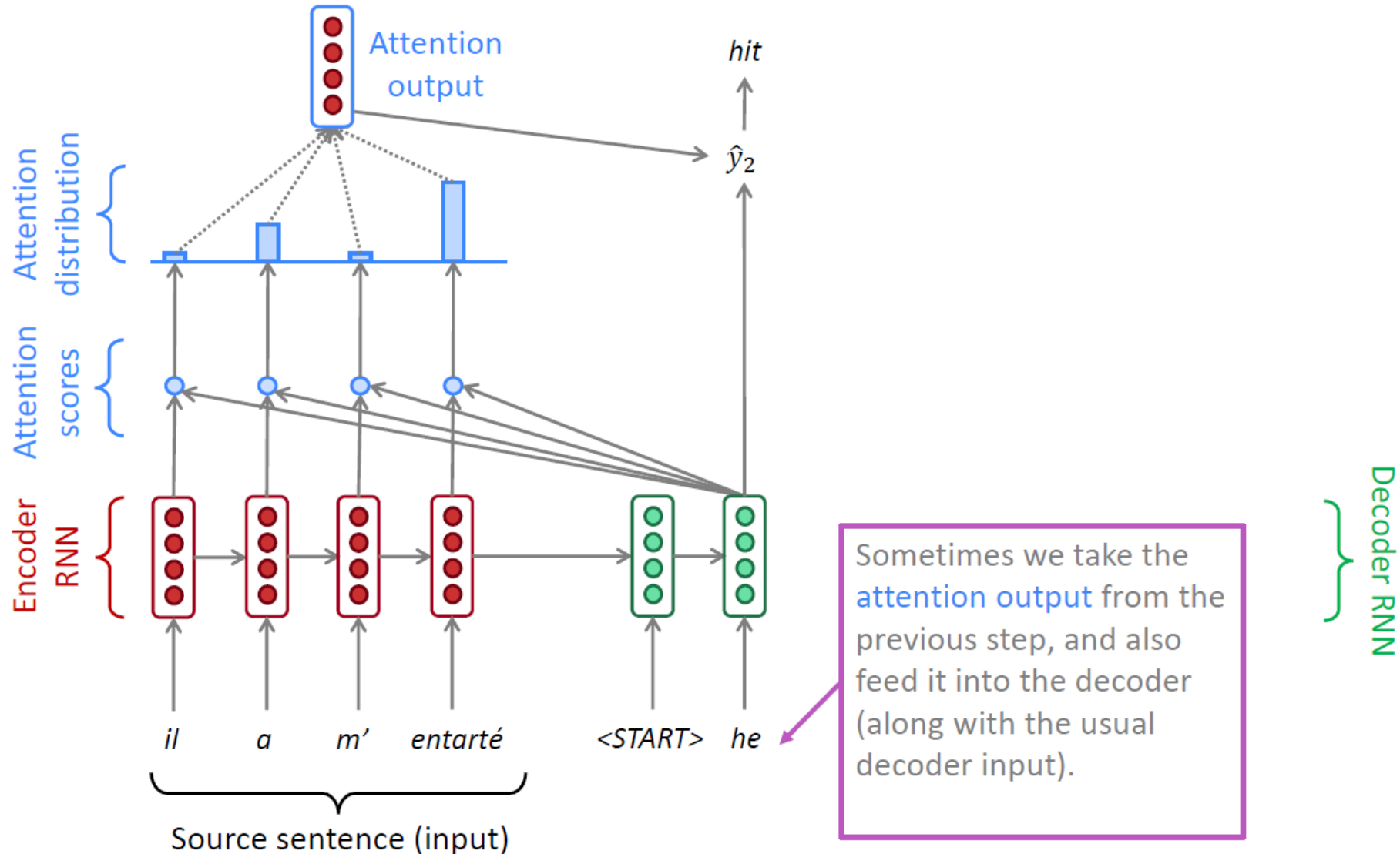
Sequence-to-Sequence with Attention



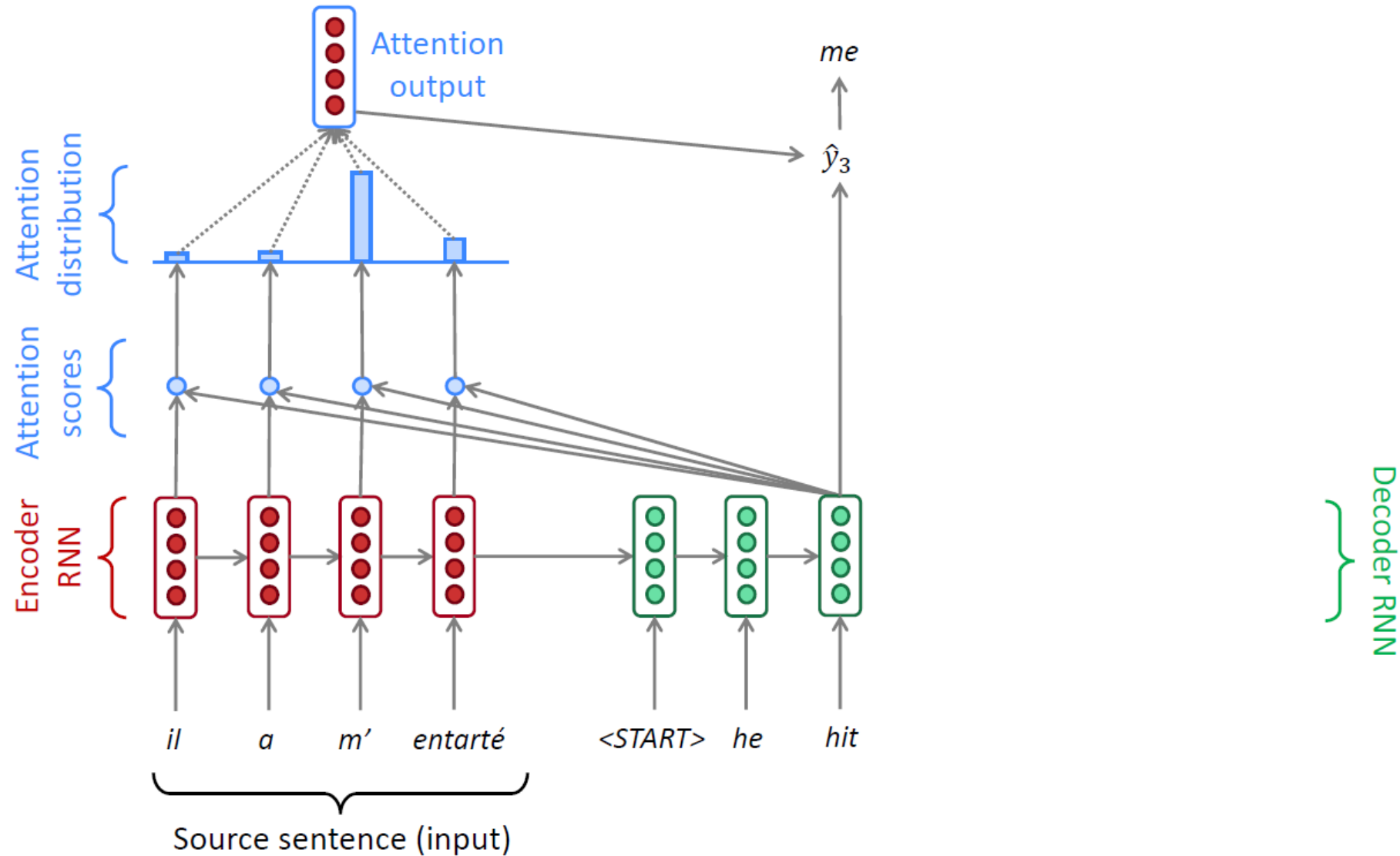
Sequence-to-Sequence with Attention



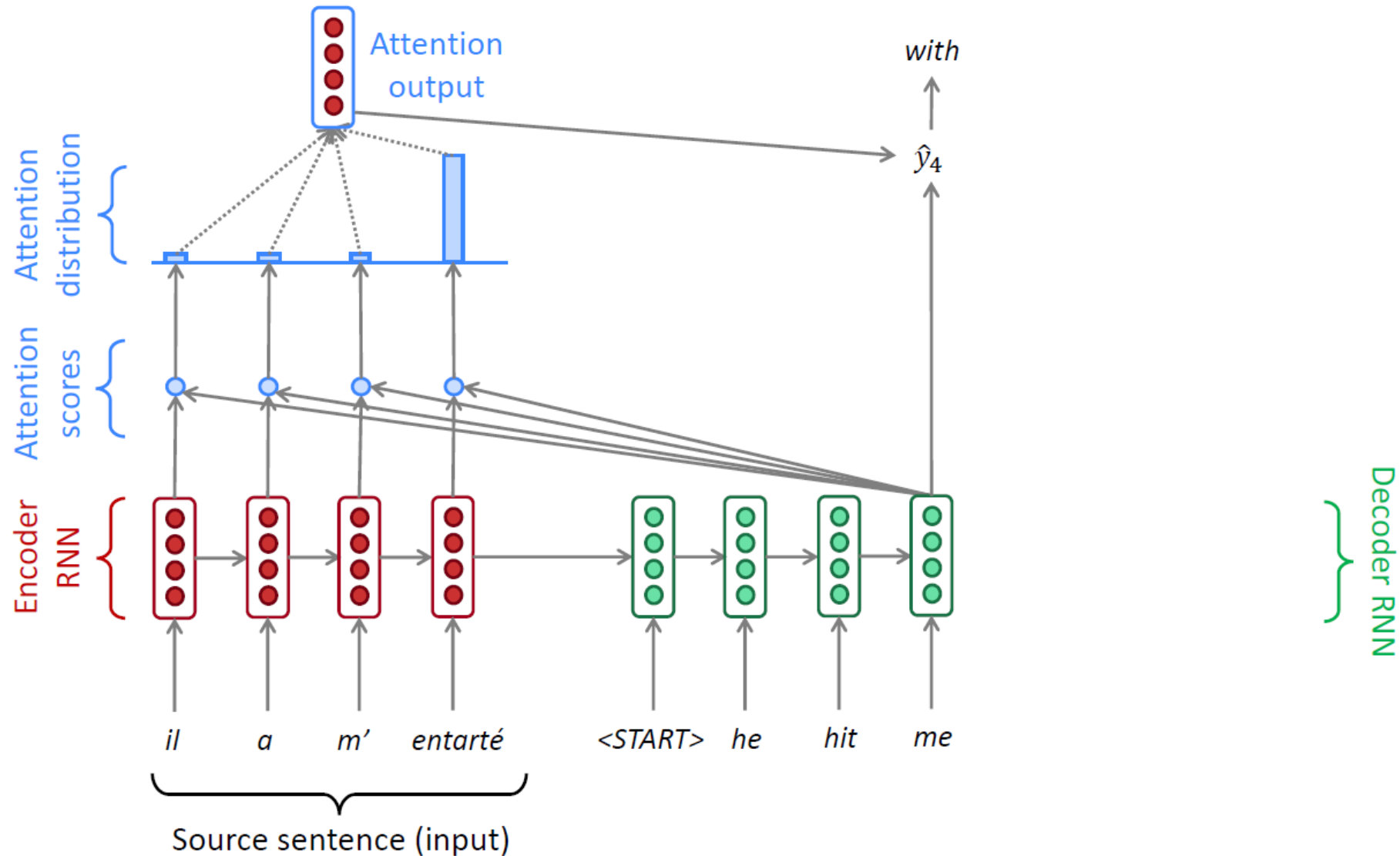
Sequence-to-Sequence with Attention



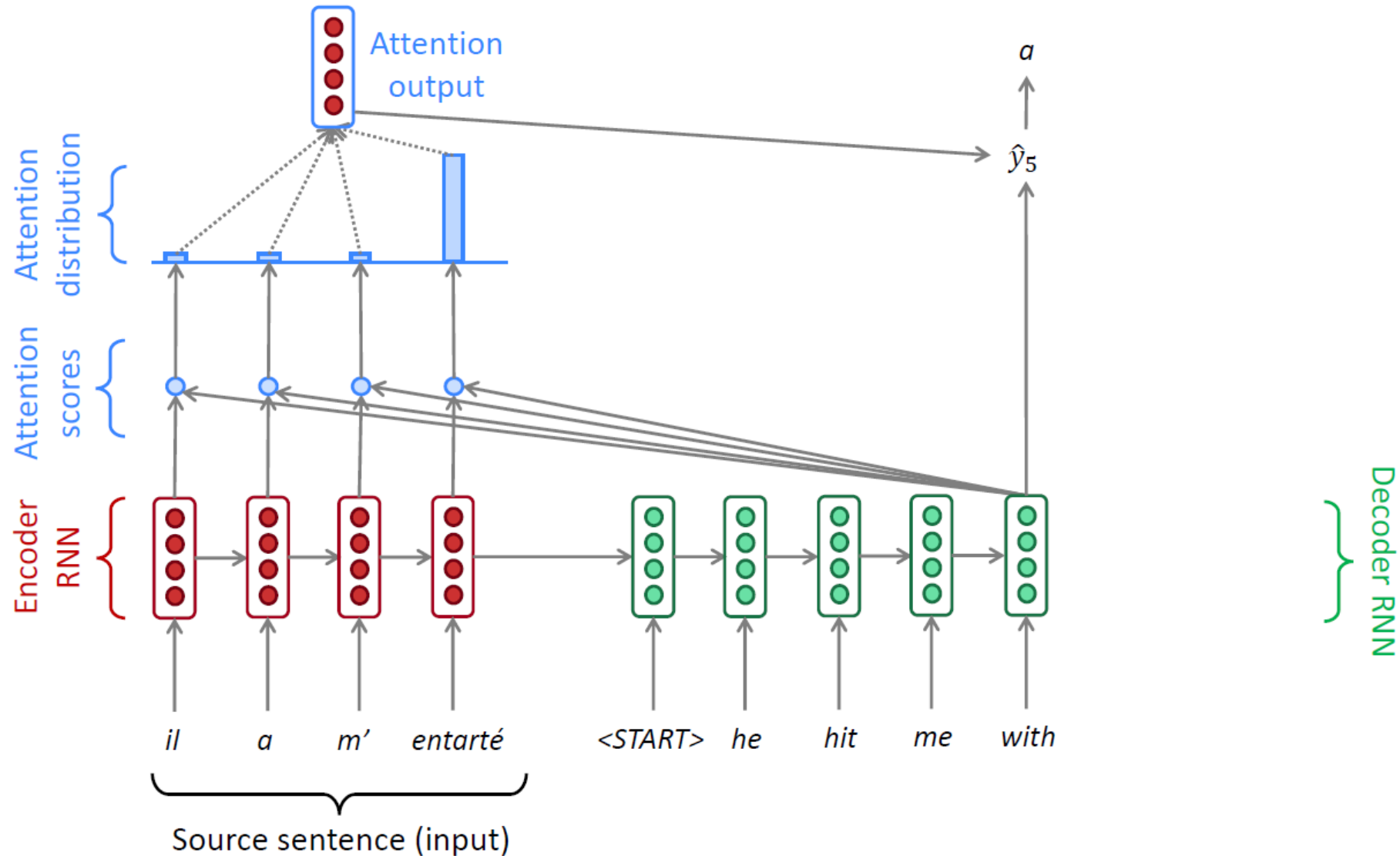
Sequence-to-Sequence with Attention



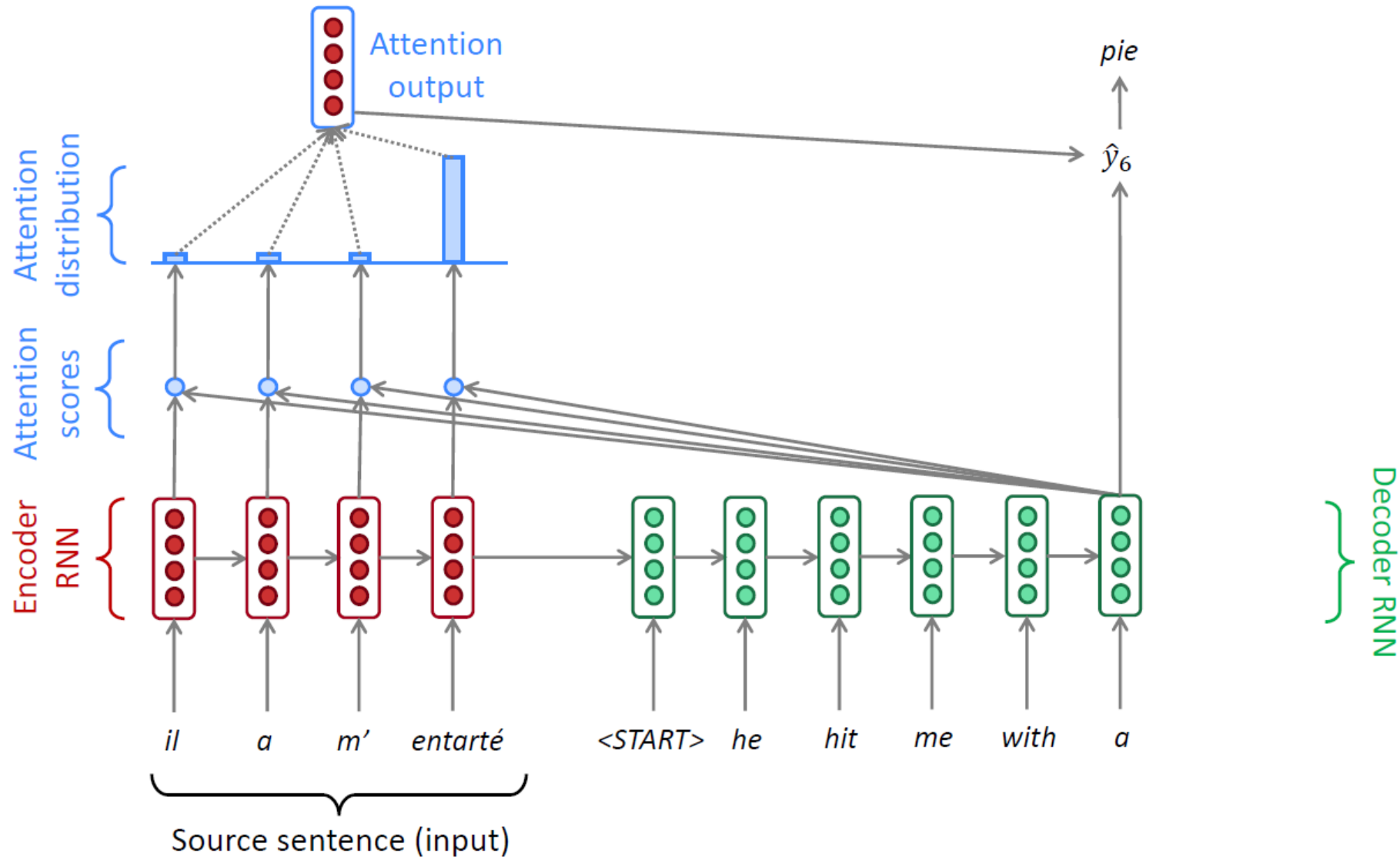
Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



Attention: In Equation

- 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

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

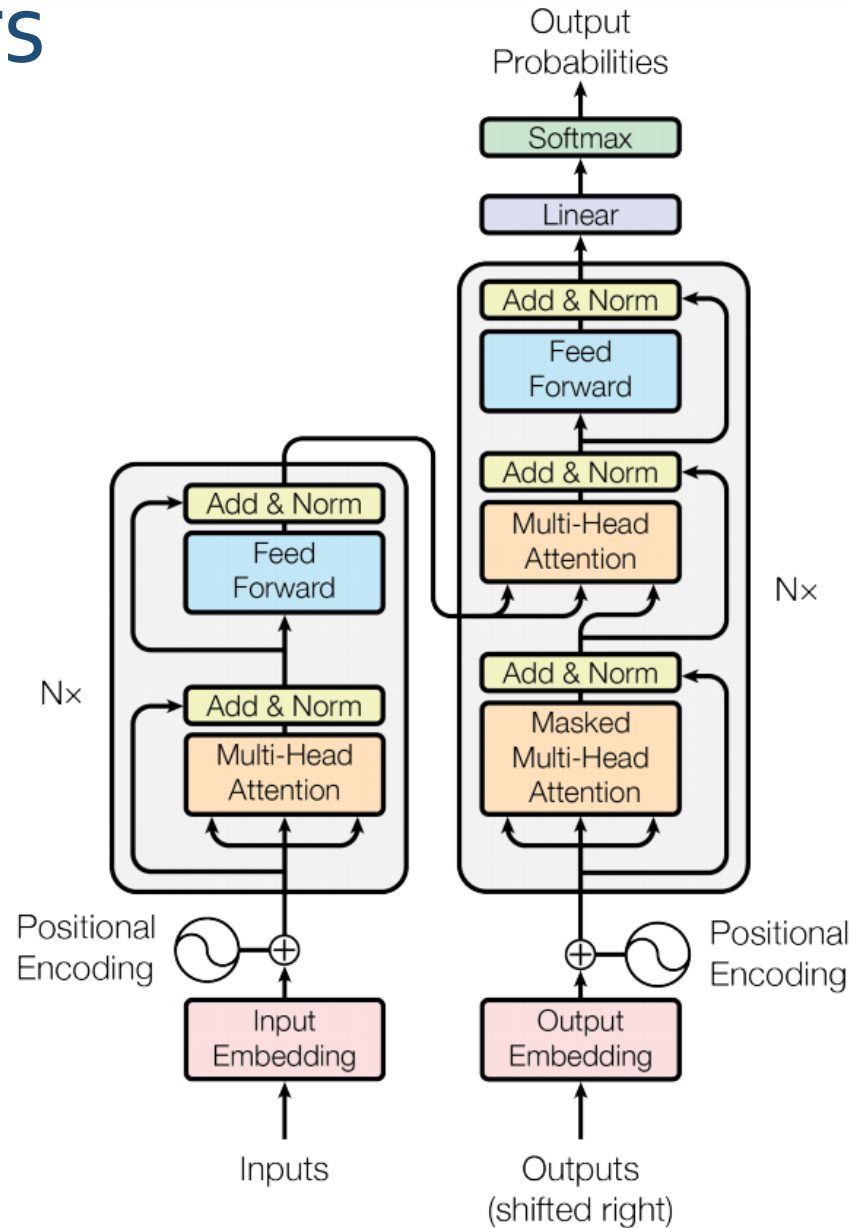
Attention is Great

- 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 (soft) **alignment for free!**
 - This is cool because we never explicitly trained an alignment system
 - The network just learned alignment by itself

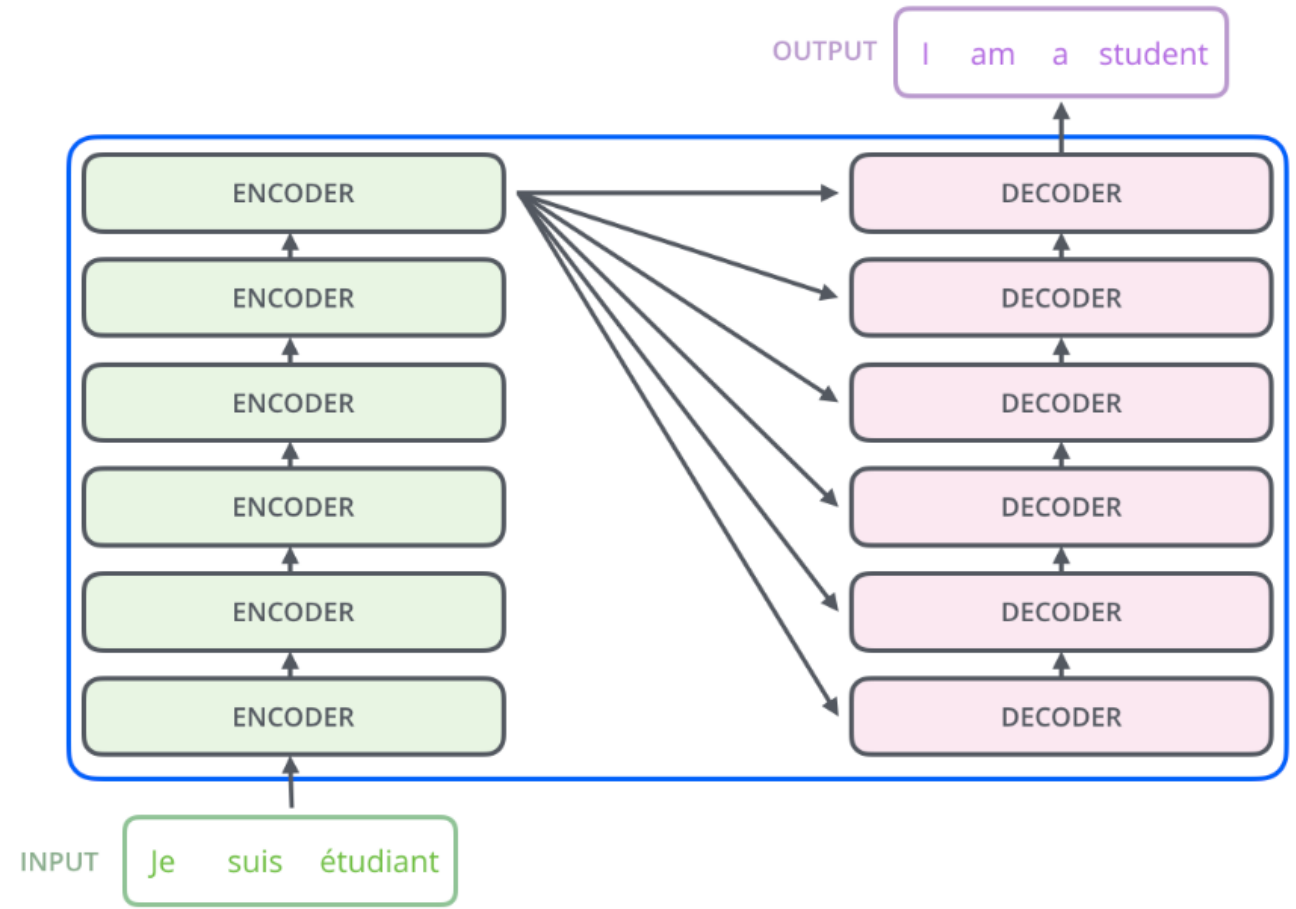
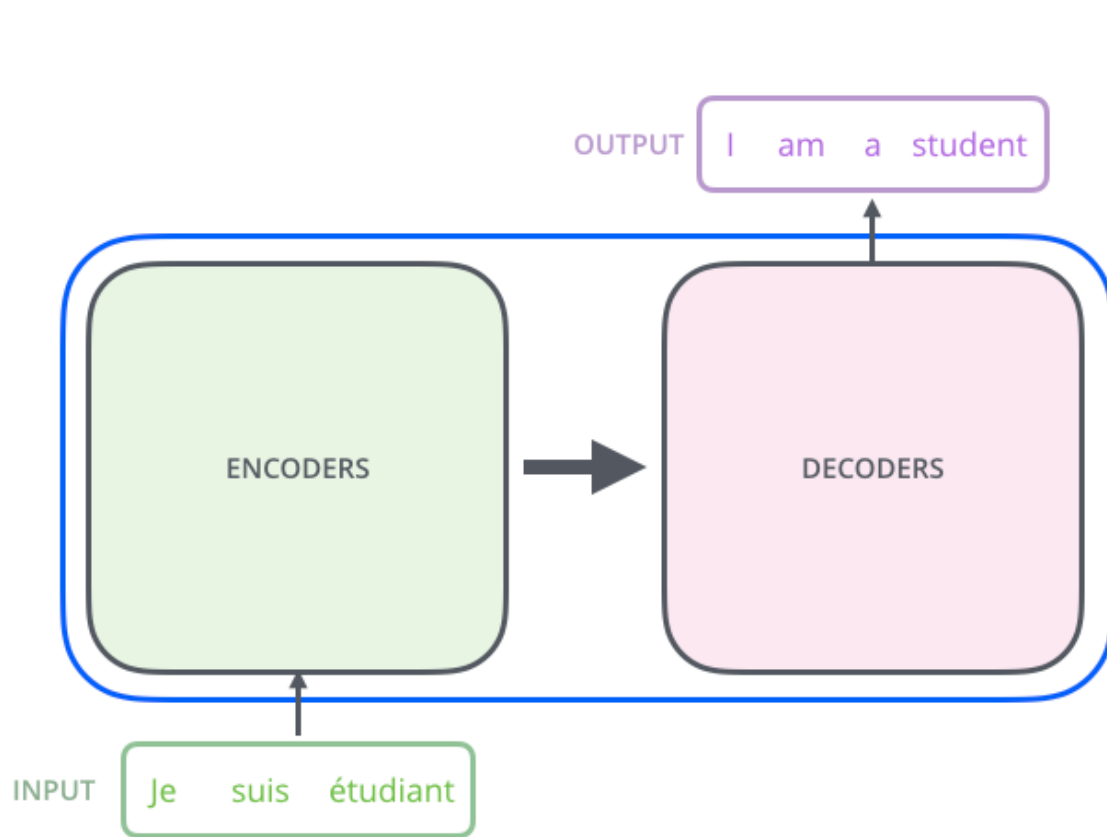
	he	hit	me	with	a	pie
il						
a						
m'						
entarté						

Attention is All You Need(NIPS 2017, Google)

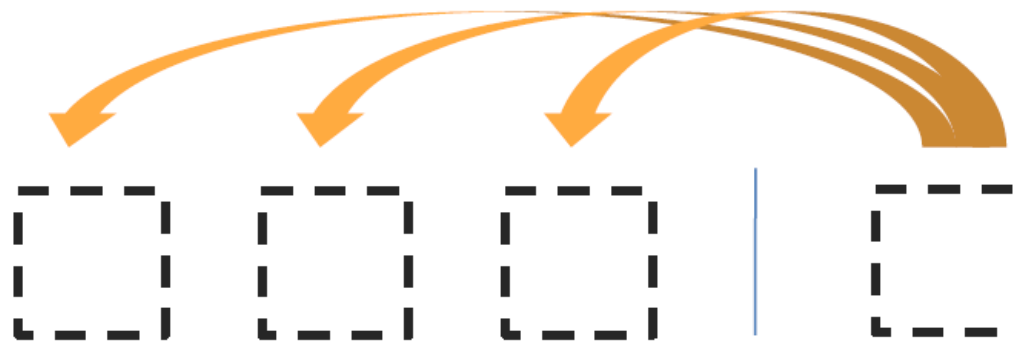
- Transformers



Seq2Seq vs Transformer



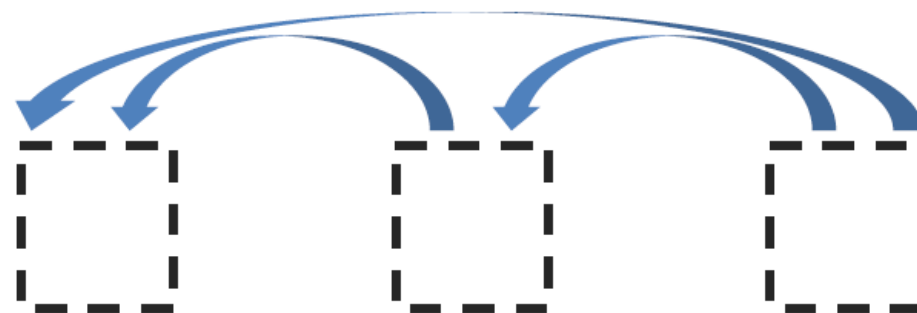
3 Types of Attention



Encoder-Decoder Attention

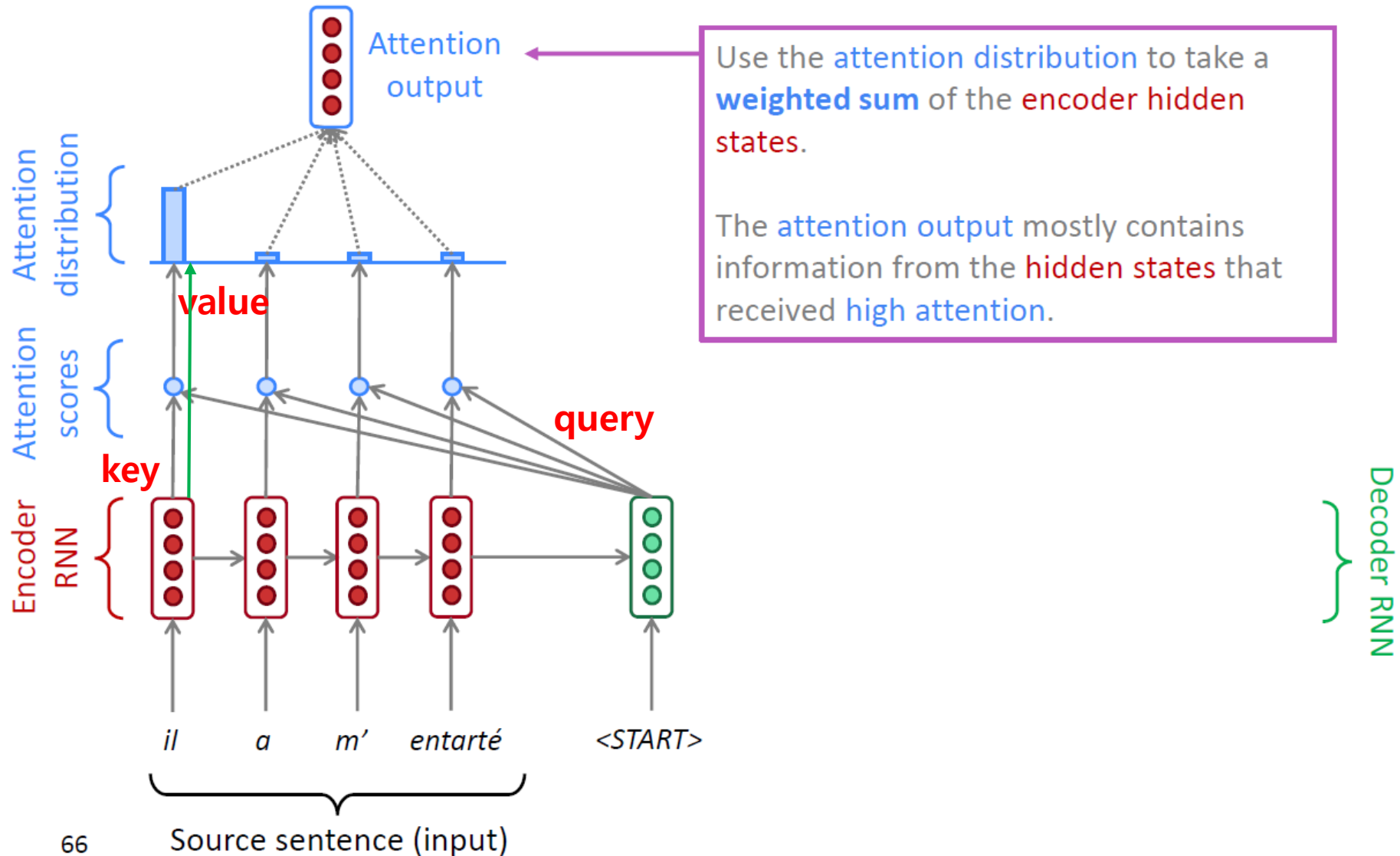


Encoder Self-Attention

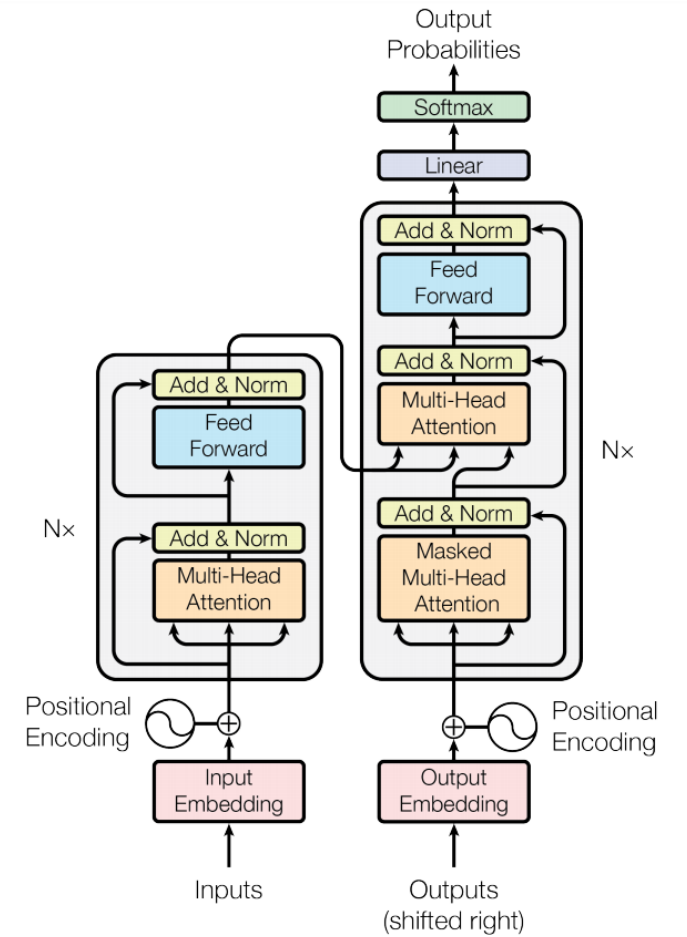
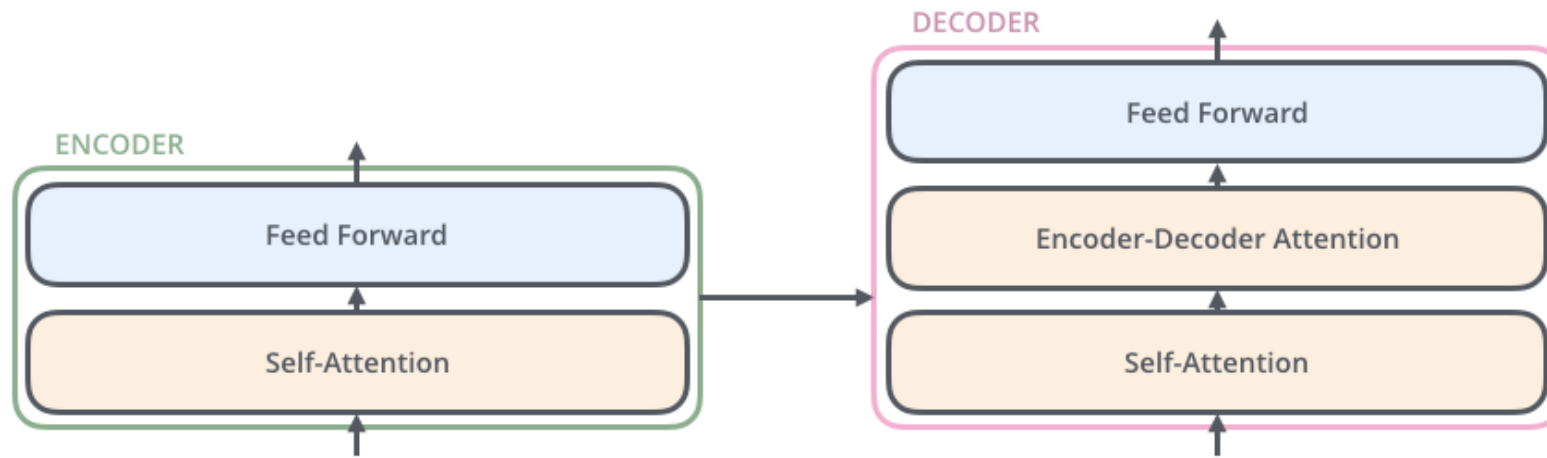


Masked Decoder Self-Attention

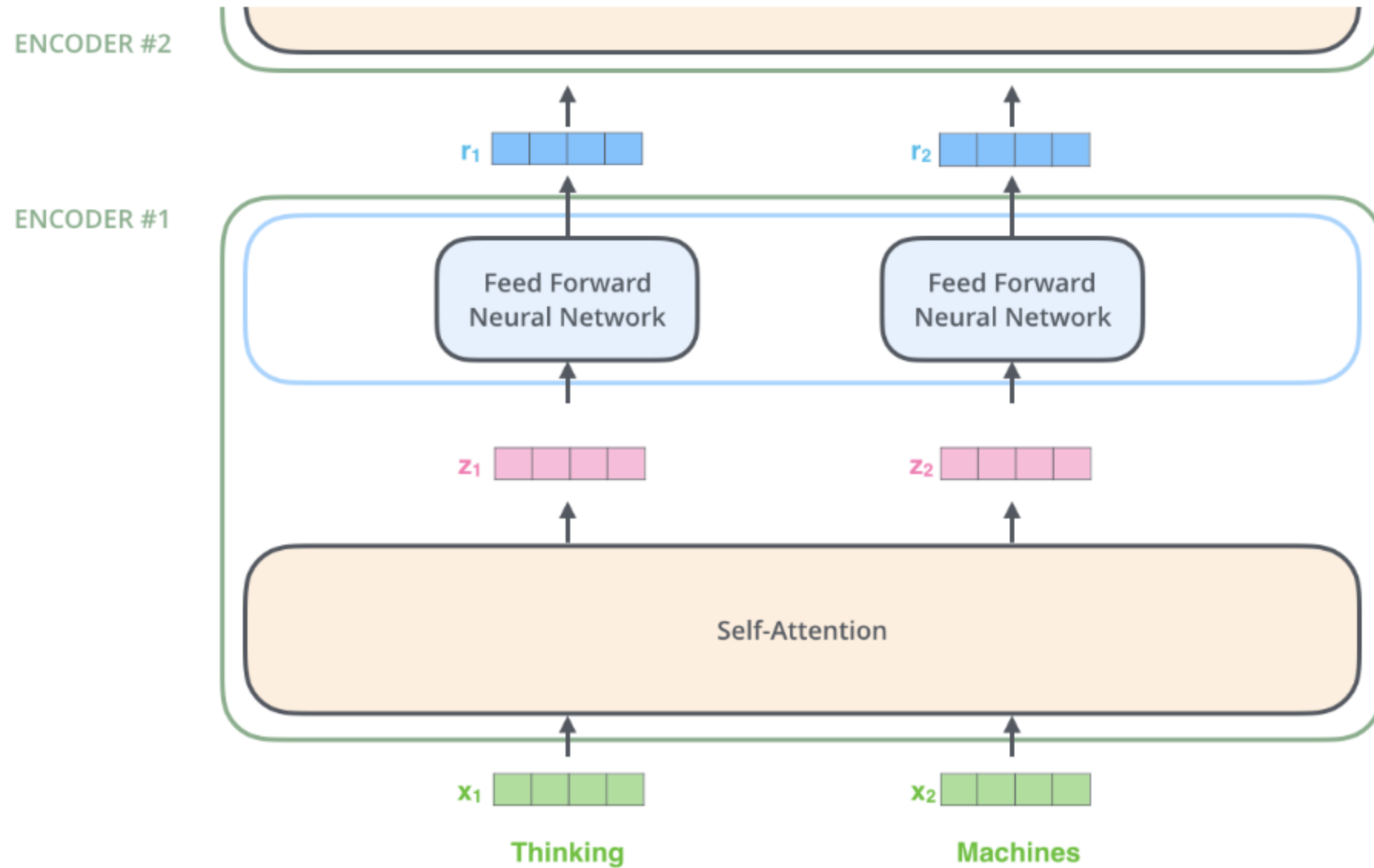
Query, Key, Value



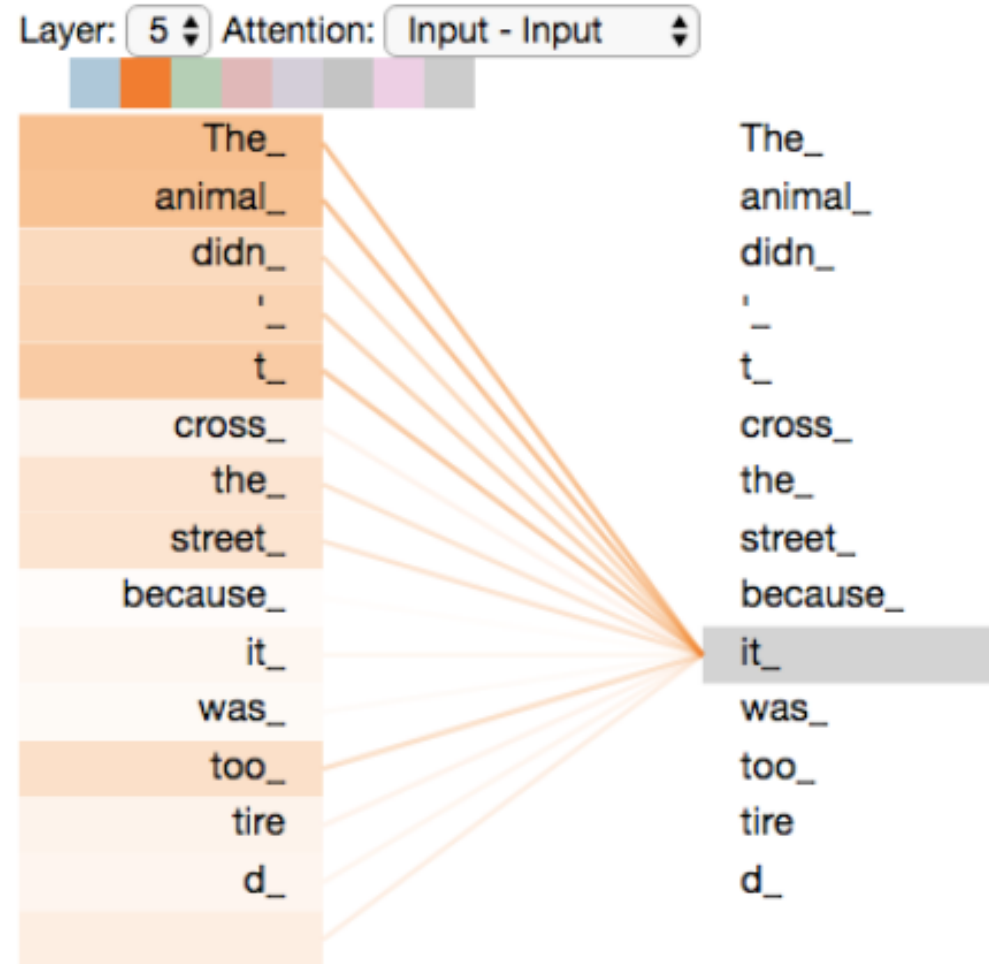
Encoder-Decoder of Transformer



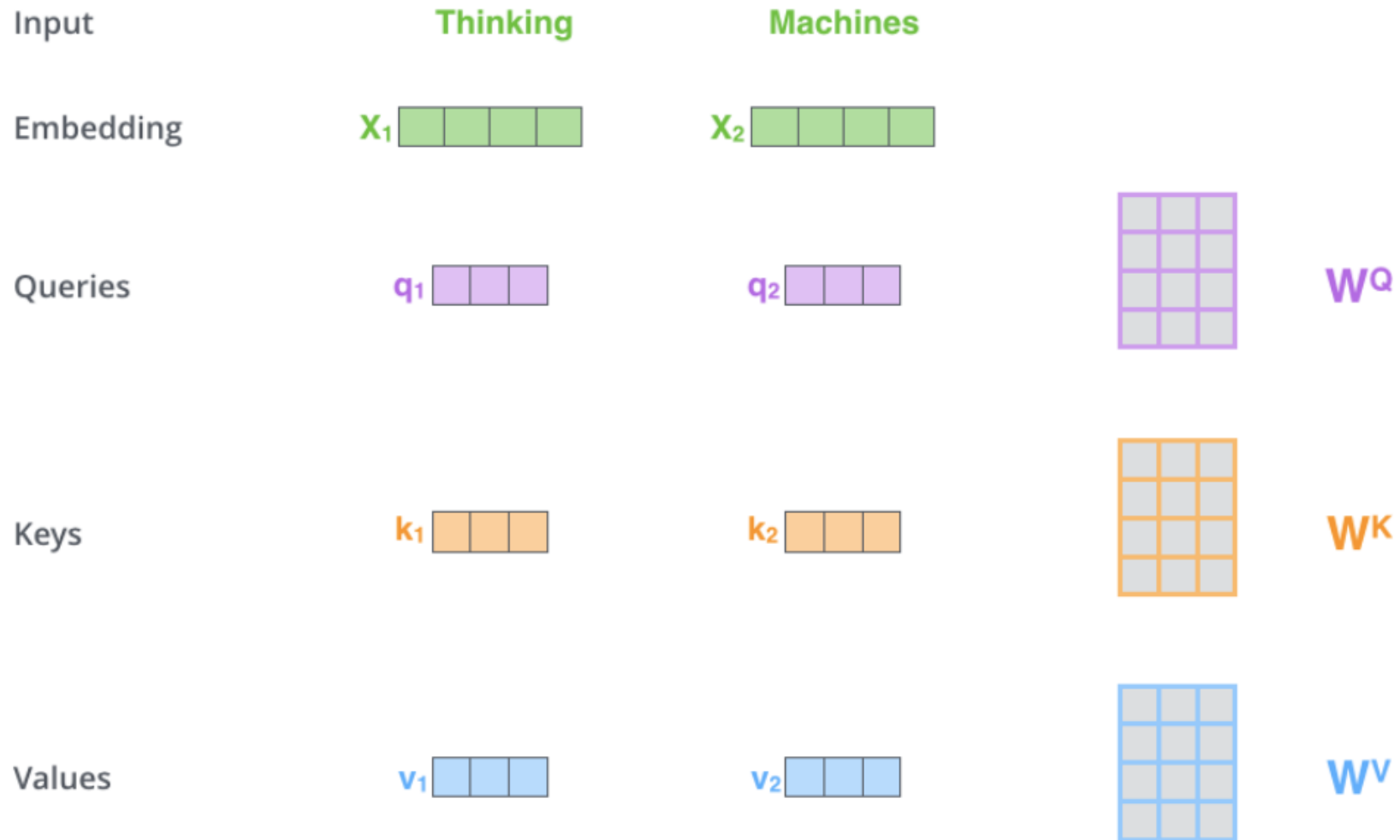
Encoder



Self Attention



Self Attention



Self Attention

Input

Embedding

Queries

Keys

Values

Score

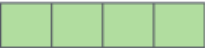
Divide by 8 ($\sqrt{d_k}$)

Softmax

Softmax
X
Value

Sum

Thinking

x_1 

q_1 

k_1 

v_1 

$q_1 \cdot k_1 = 112$

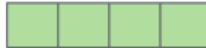
14

0.88

v_1 

z_1 

Machines

x_2 

q_2 

k_2 

v_2 

$q_1 \cdot k_2 = 96$

12

0.12

v_2 

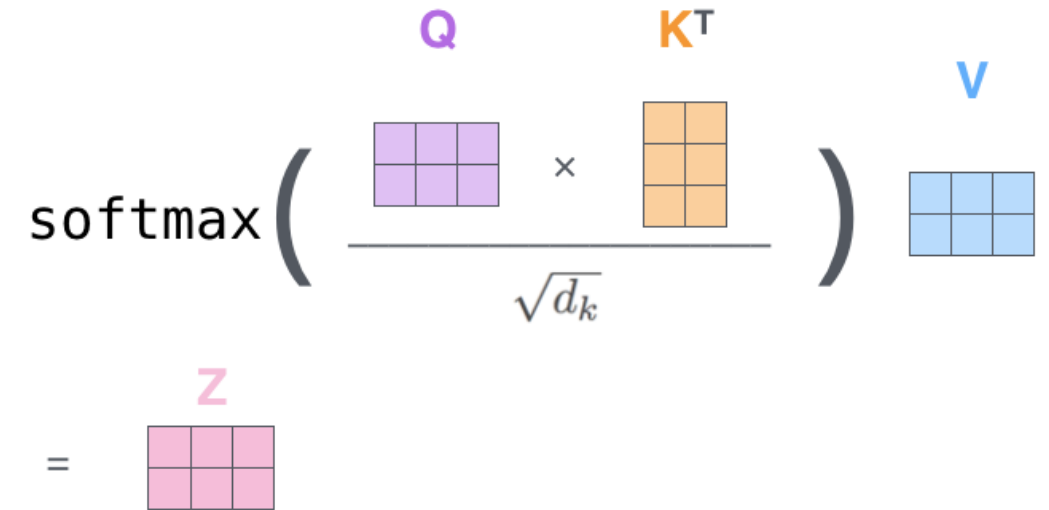
z_2 

Matrix Calculation of Self-Attention

$$\mathbf{X} \times \mathbf{W}^Q = \mathbf{Q}$$


$$\mathbf{X} \times \mathbf{W}^K = \mathbf{K}$$


$$\mathbf{X} \times \mathbf{W}^V = \mathbf{V}$$


$$\text{softmax}\left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V}$$

$$= \mathbf{Z}$$

Multi-head Attention

1) This is our input sentence*

Thinking
Machines

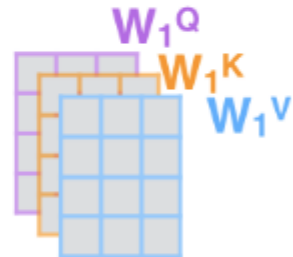
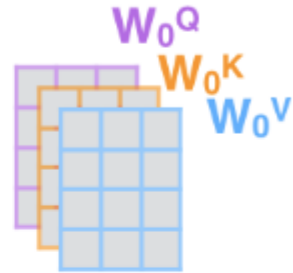
2) We embed each word*



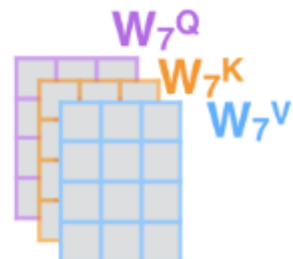
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



3) Split into 8 heads. We multiply X or R with weight matrices



...



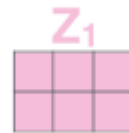
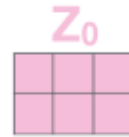
4) Calculate attention using the resulting $Q/K/V$ matrices



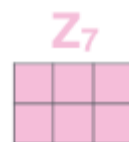
...



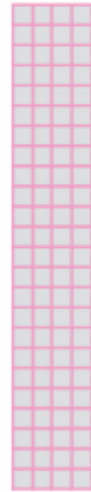
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



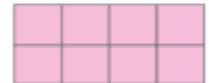
...



W^O

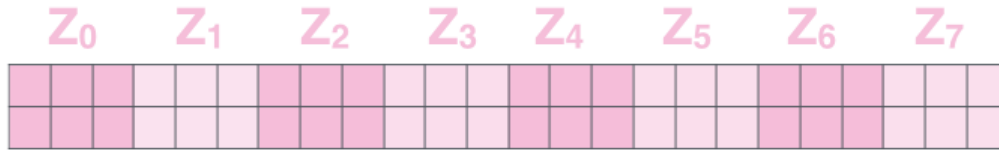


Z

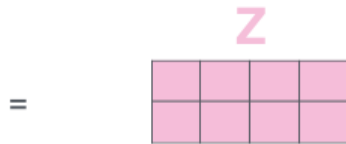


Multi-head Attention

1) Concatenate all the attention heads

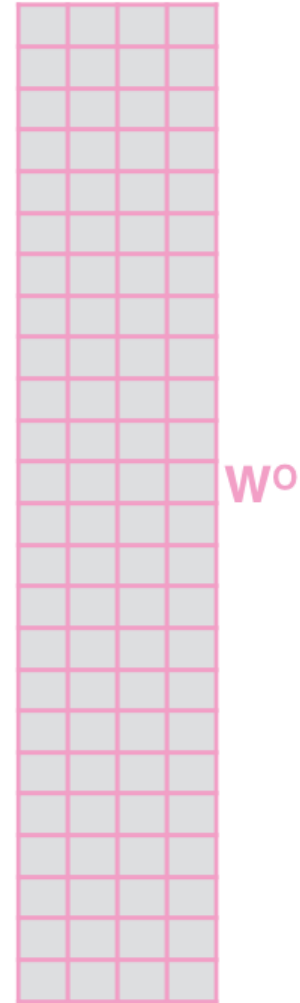


3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

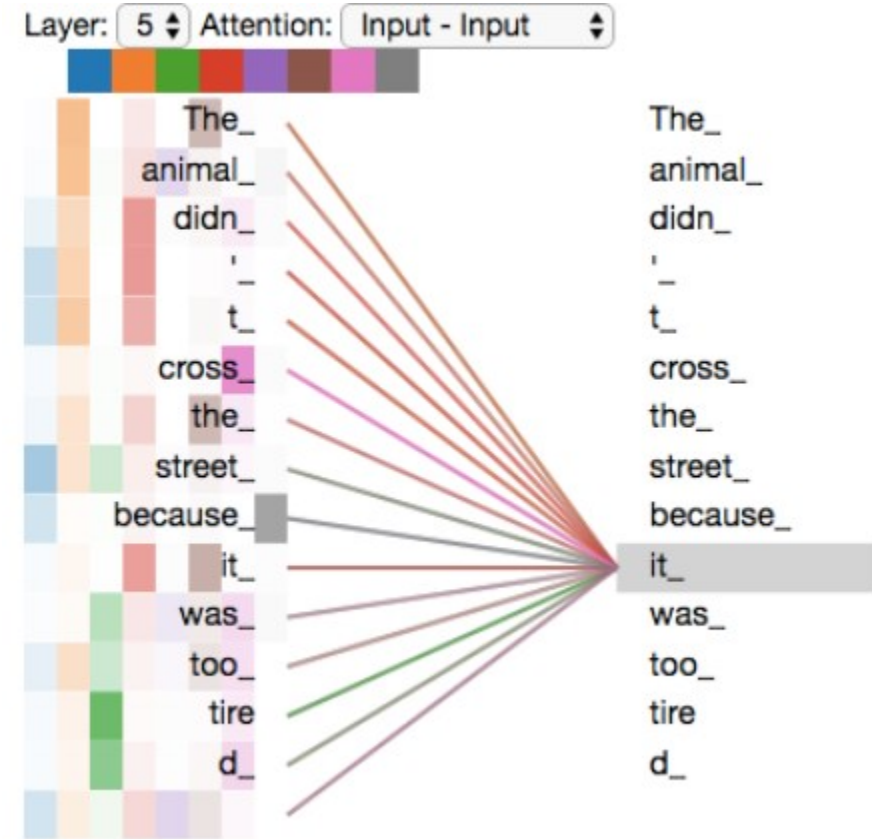
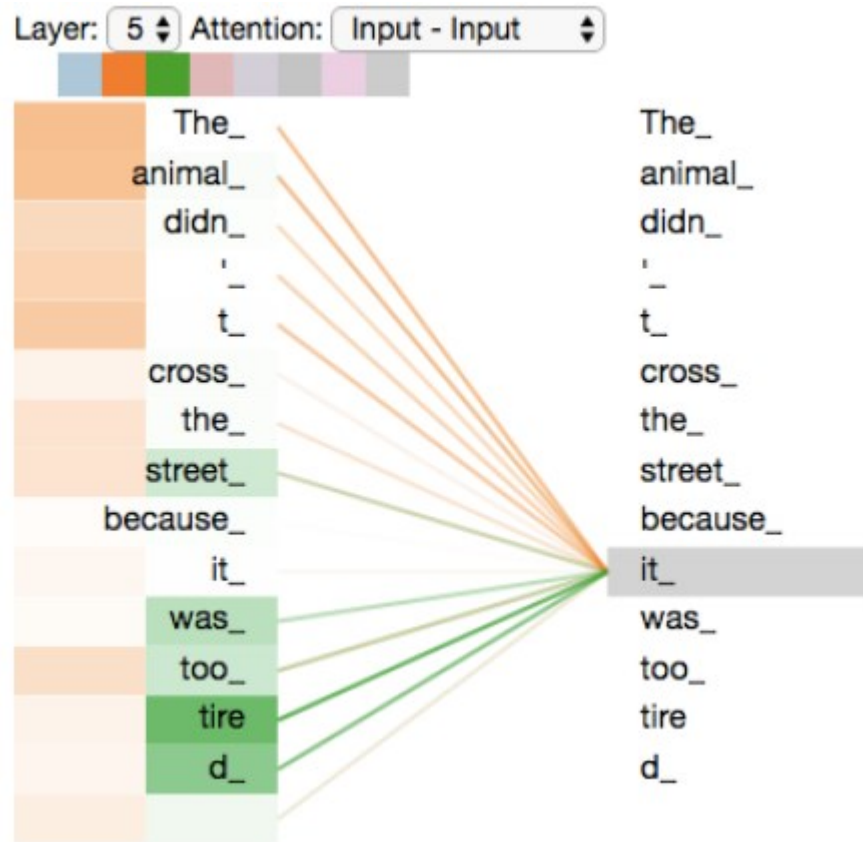


2) Multiply with a weight matrix W^O that was trained jointly with the model

X

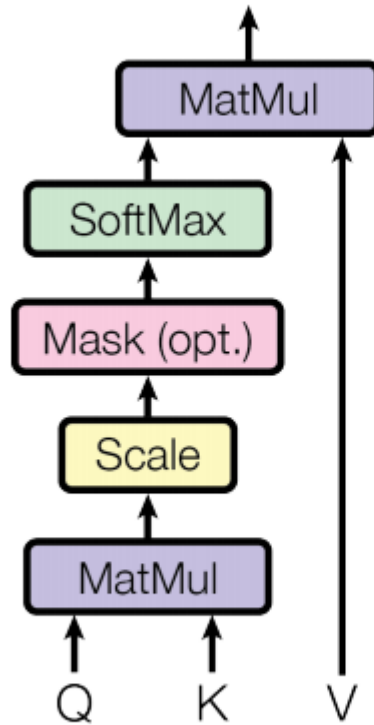


Multi-head Attention

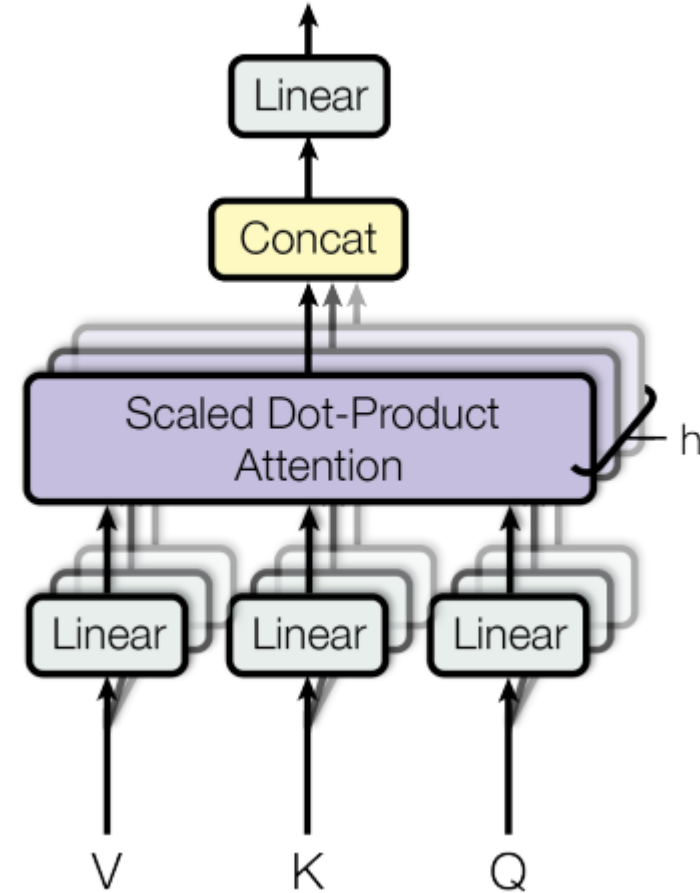


Self-Attention of Transformer

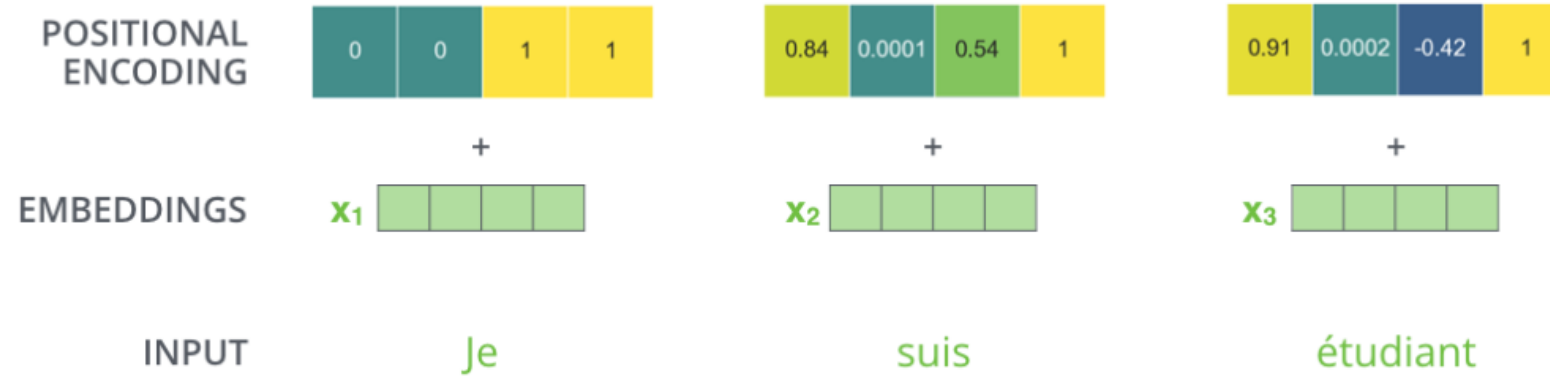
Scaled Dot-Product Attention



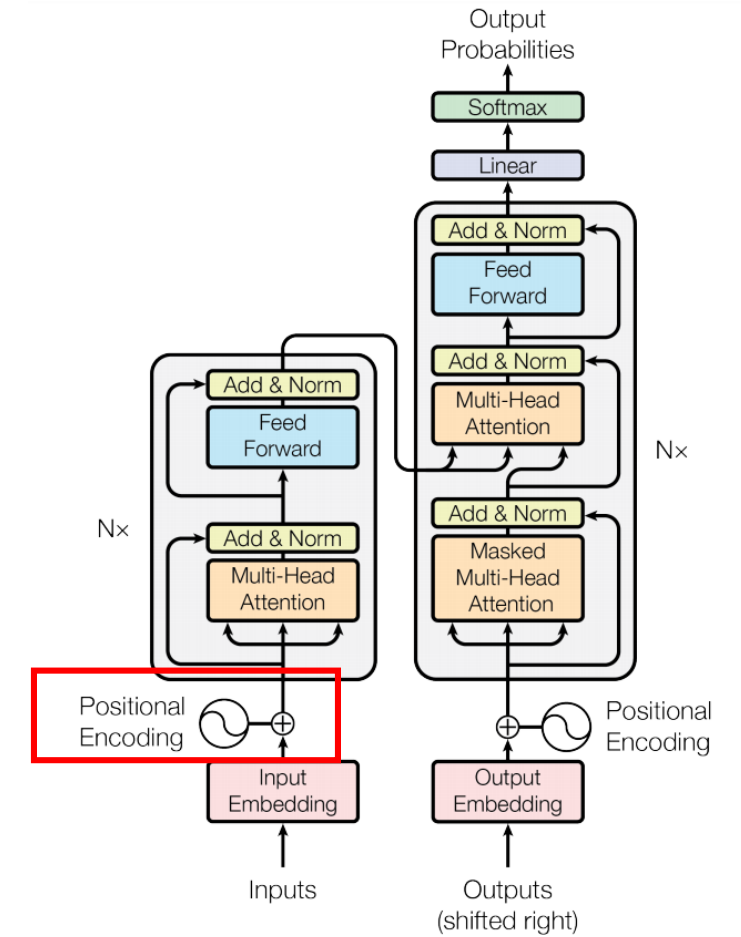
Multi-Head Attention



Positional Encoding



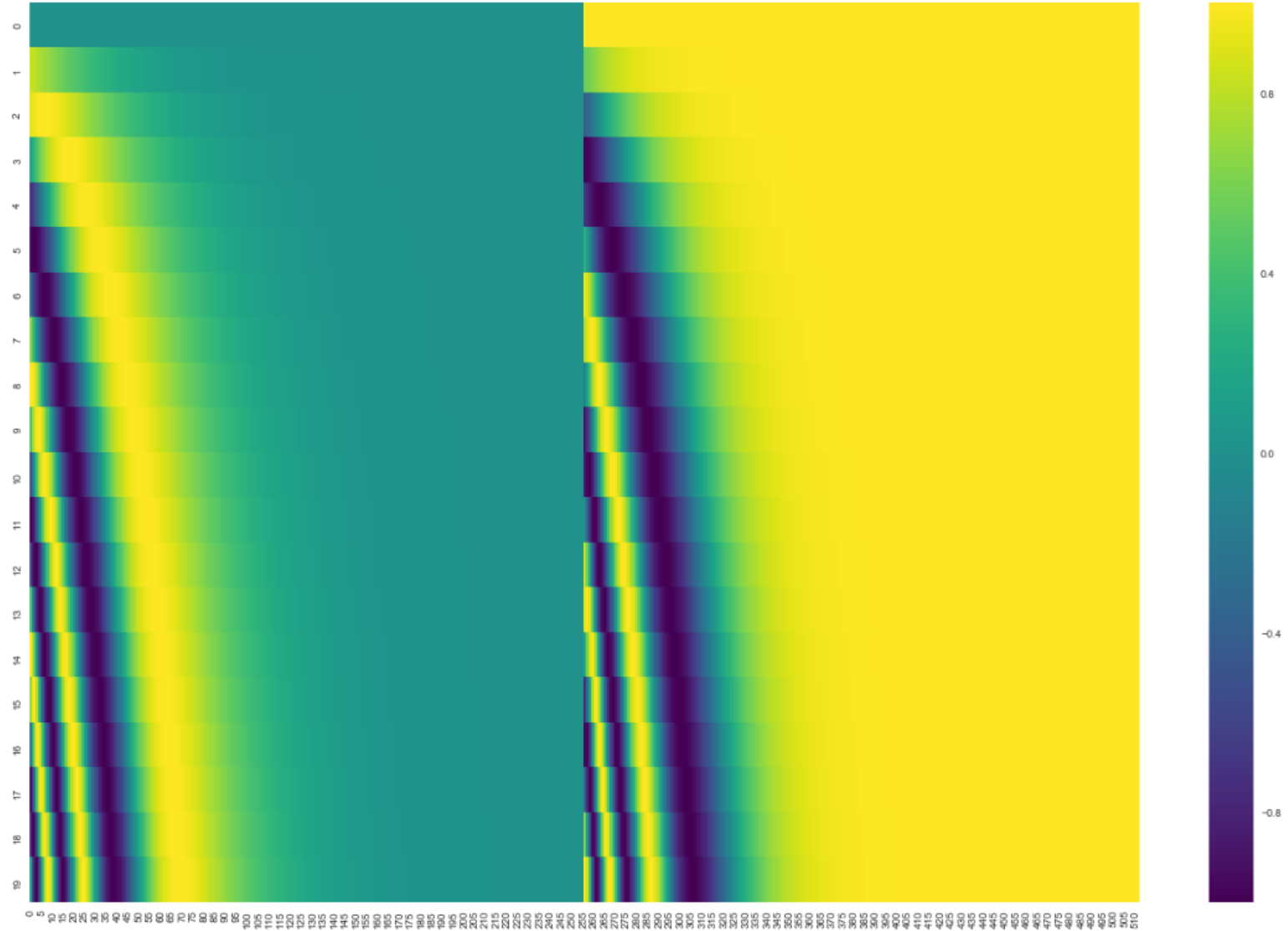
A real example of positional encoding with a toy embedding size of 4



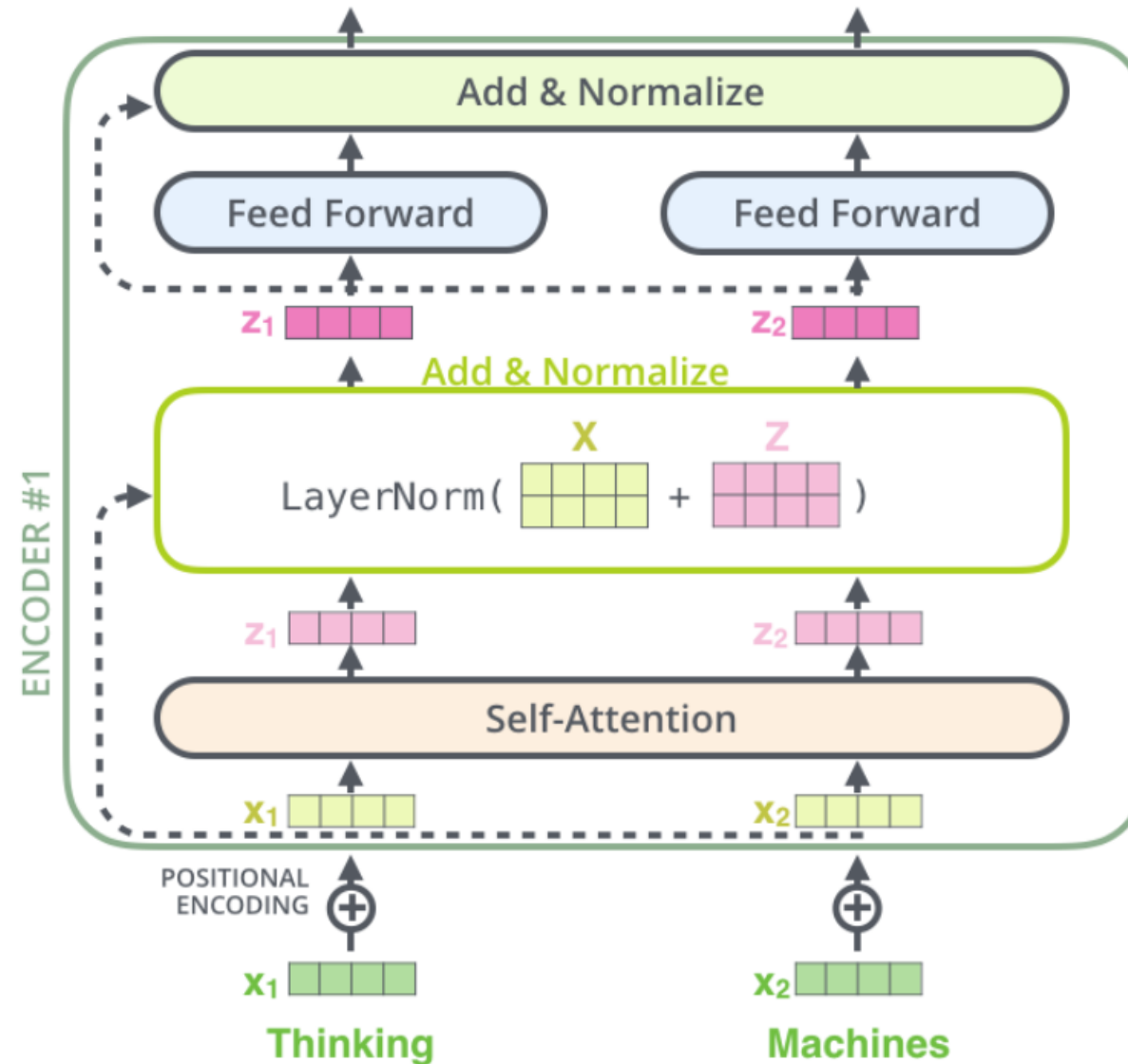
Positional Encoding

$$PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

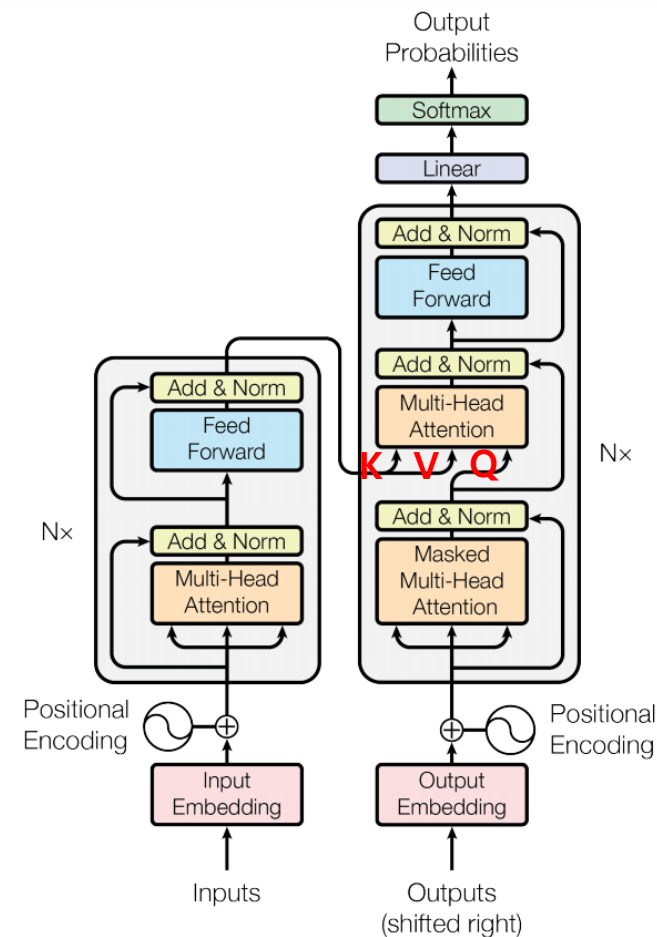
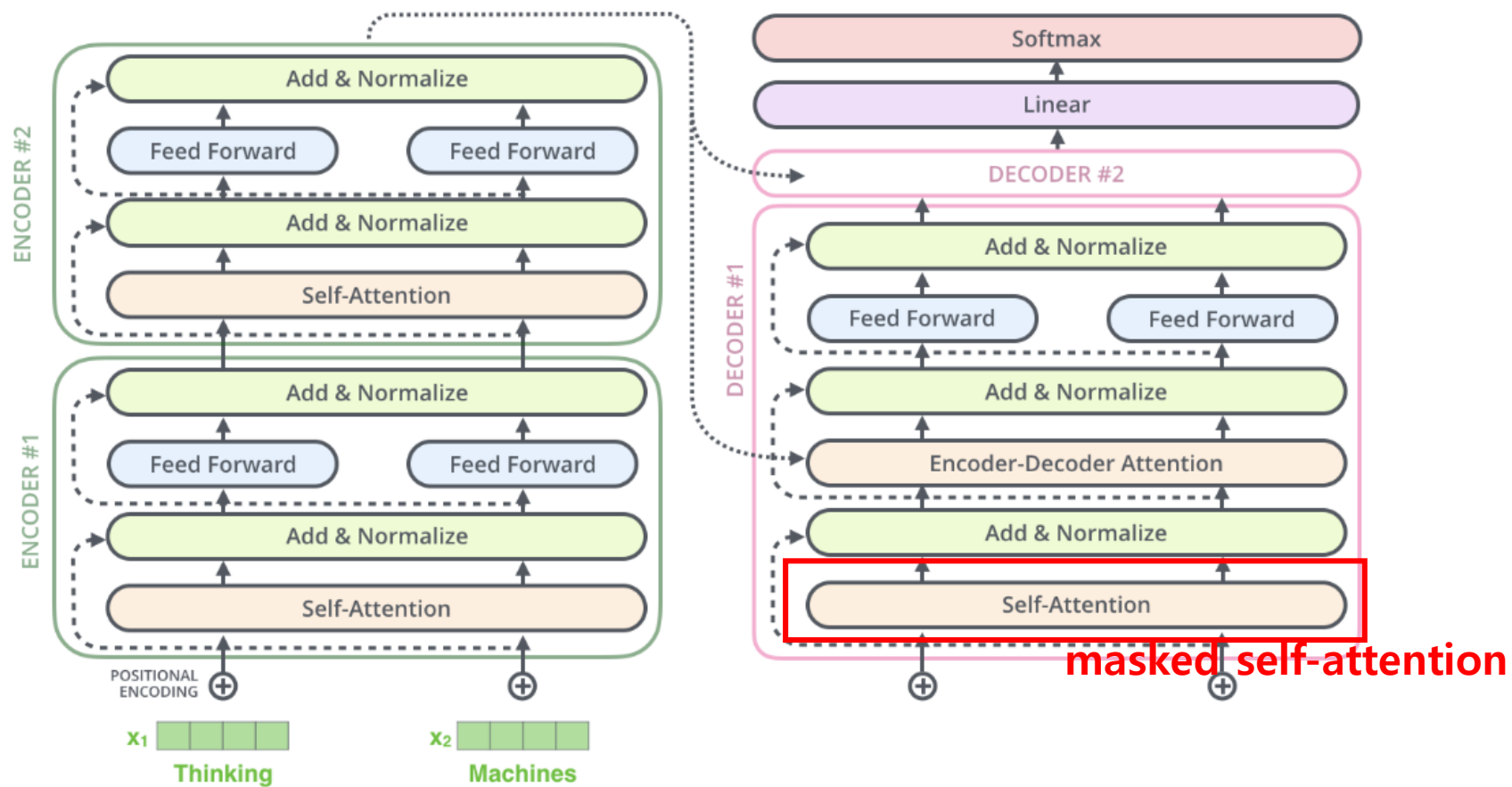
$$PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$



Skip Connection & Layer Norm



Decoder



Transformer

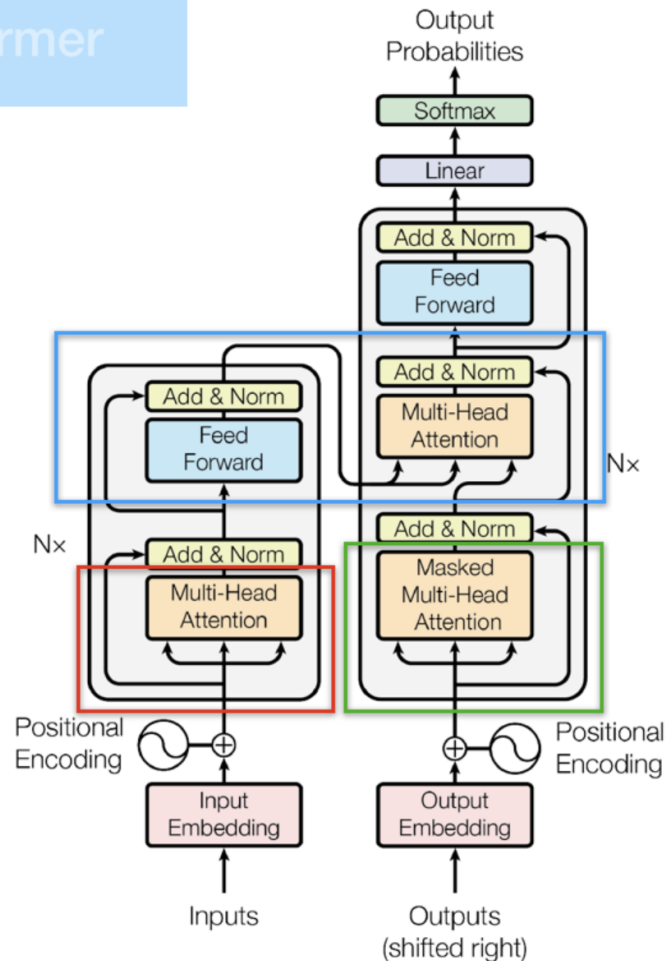


Figure 1: The Transformer - model architecture.

encoder self attention

1. Multi-head Attention
2. **Q**uery=**K**ey=**V**alue

decoder self attention

1. **M**asked Multi-head Attention
2. **Q**uery=**K**ey=**V**alue

encoder-decoder attention

1. Multi-head Attention
2. Encoder Self attention=**K**ey=**V**alue
3. Decoder Self attention=**Q**uery