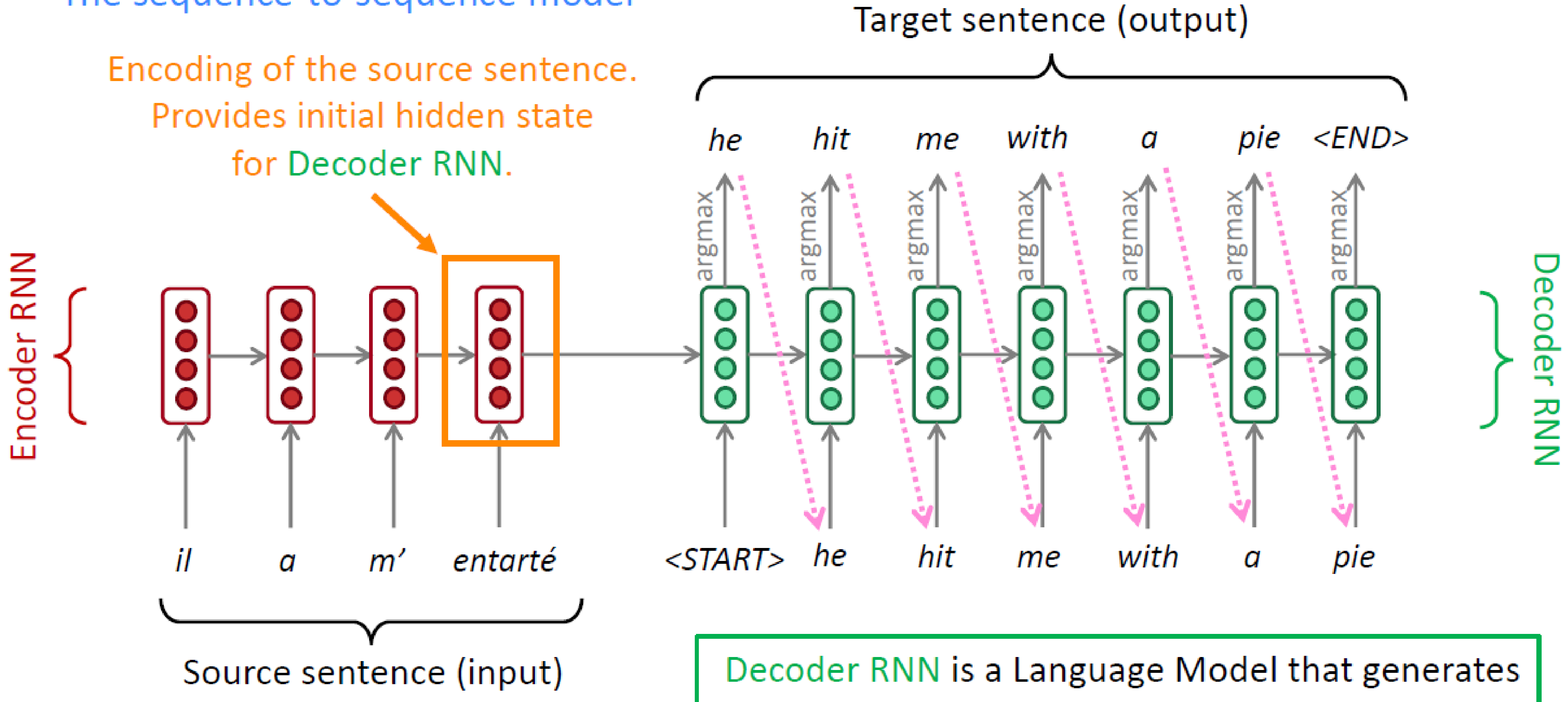
Neural Machine Translation with Attention

Neural Machine Translation (NMT)

The sequence-to-sequence model

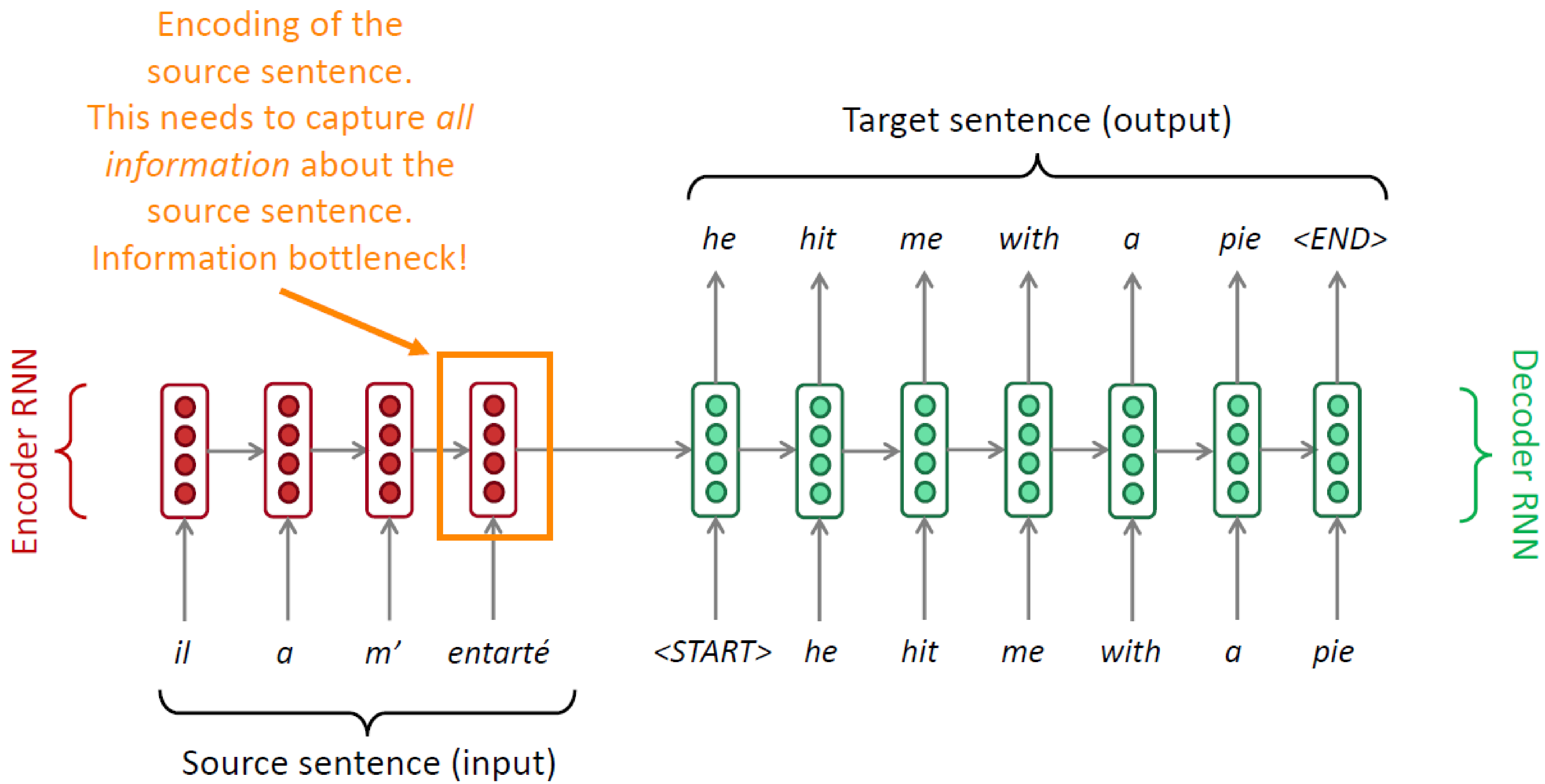


Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Encoder RNN produces an encoding of the source sentence.

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

Sequence-to-Sequence: the Bottleneck Problem



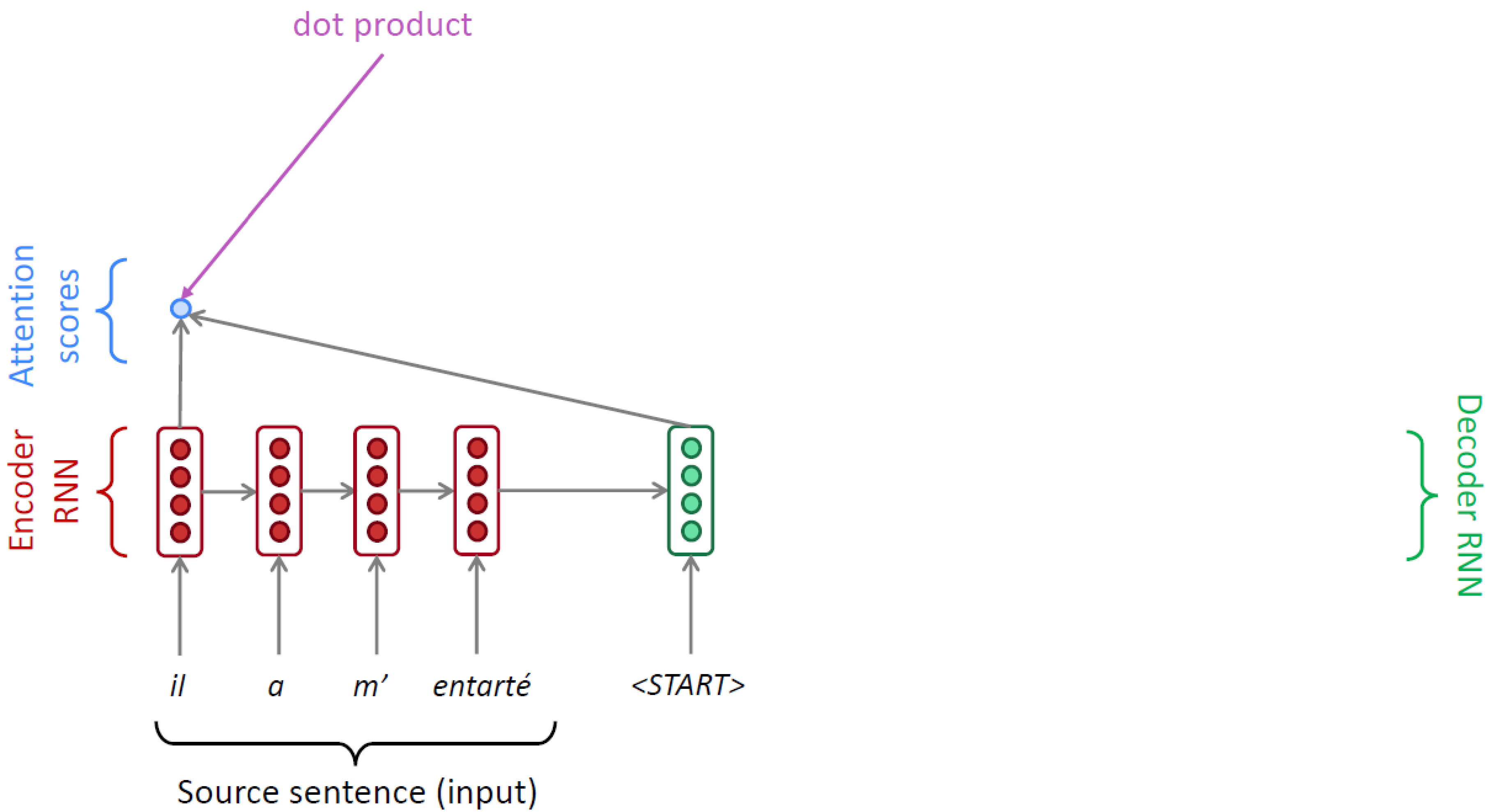
Attention

- **Attention** provides a solution to the bottleneck problem.

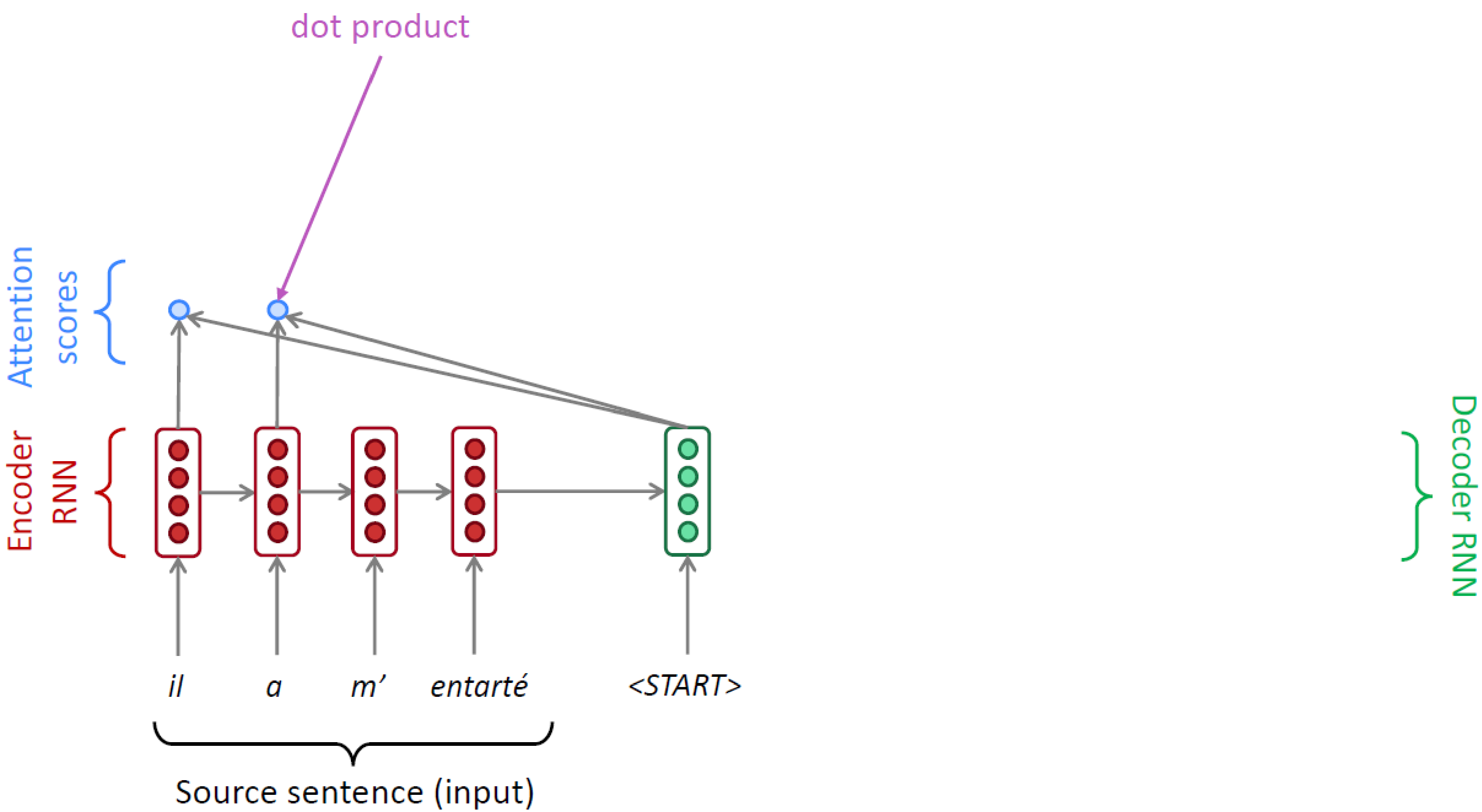
- Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

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

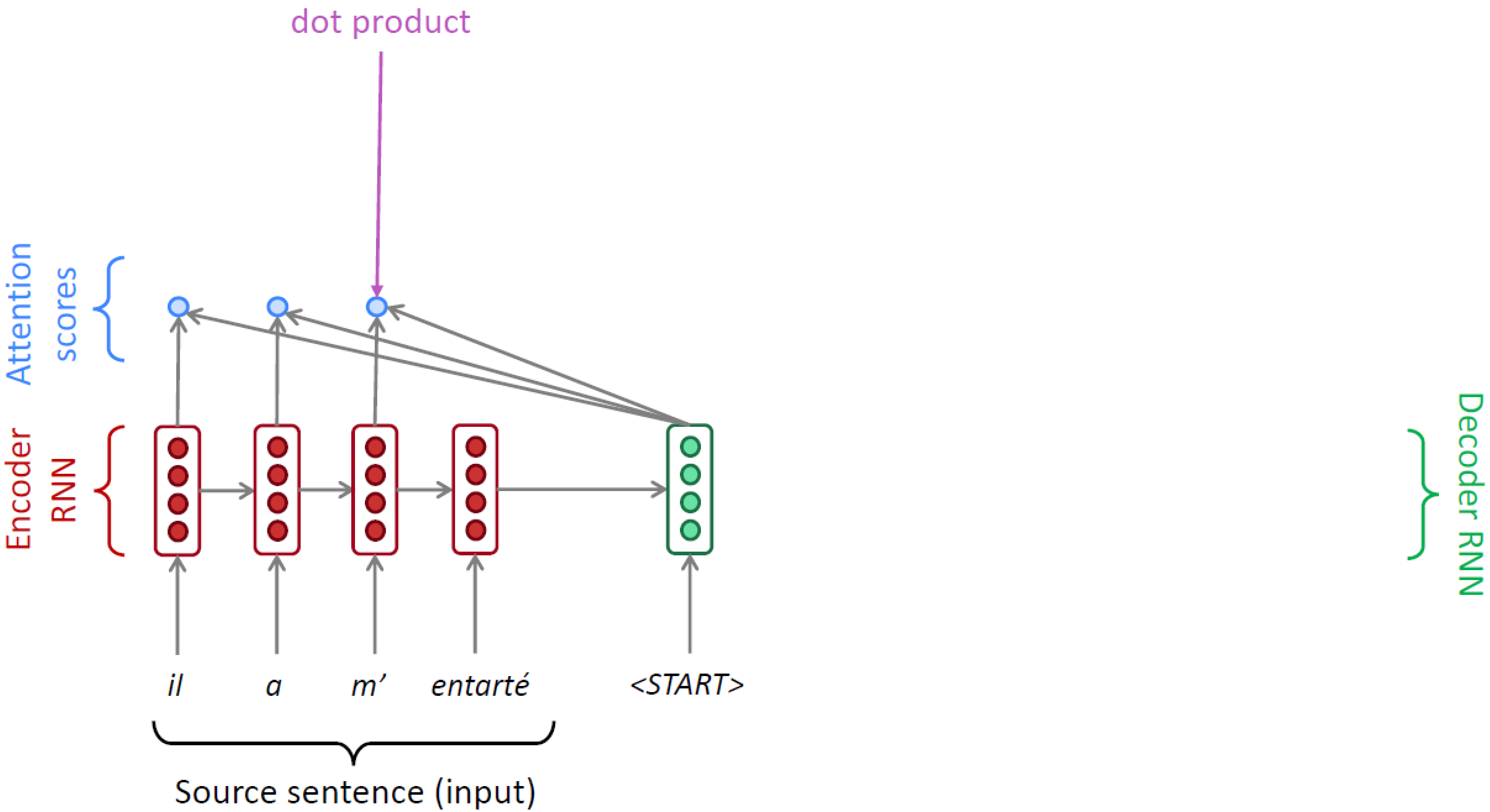
Sequence-to-Sequence with Attention



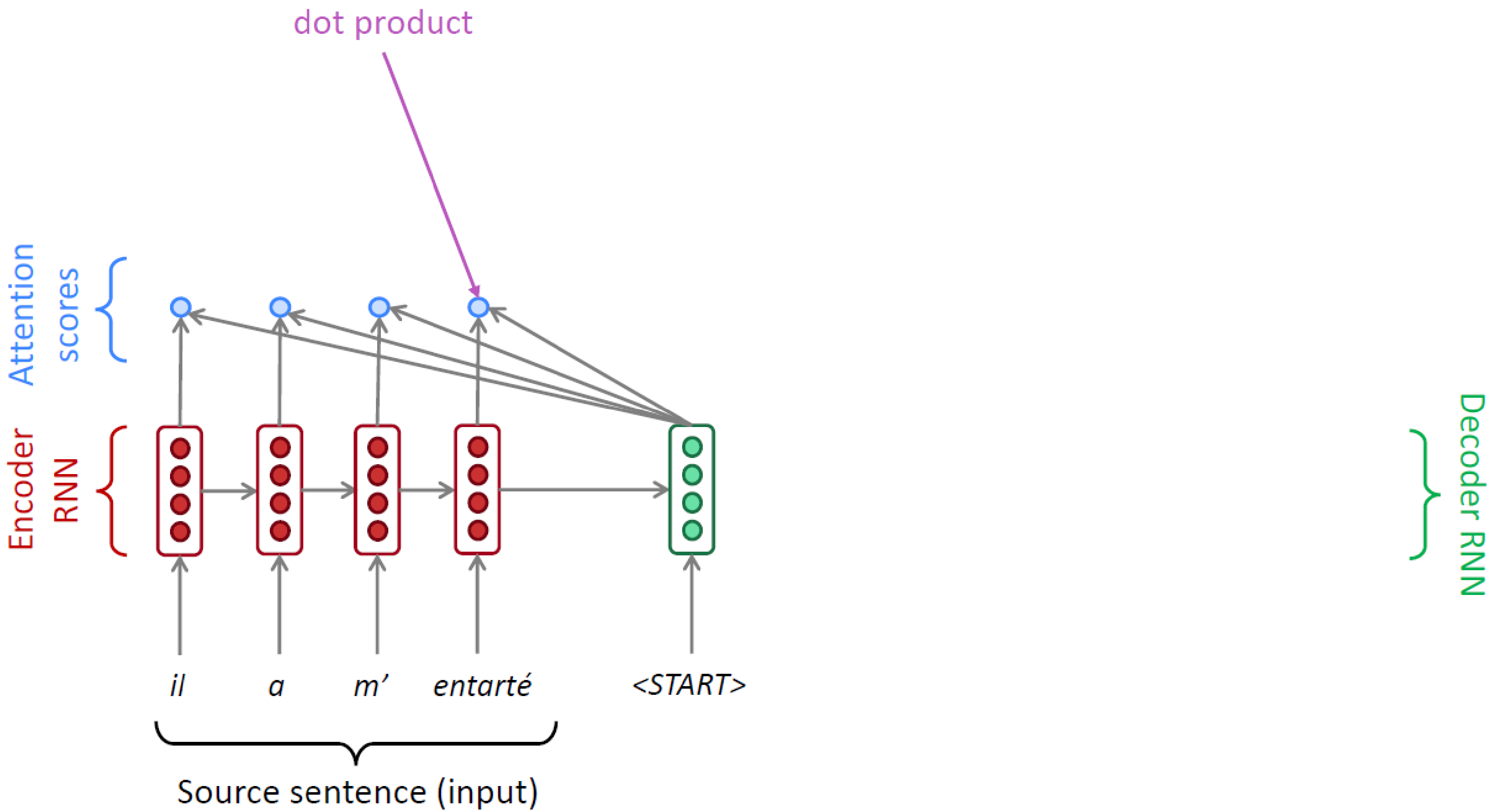
Sequence-to-Sequence with Attention



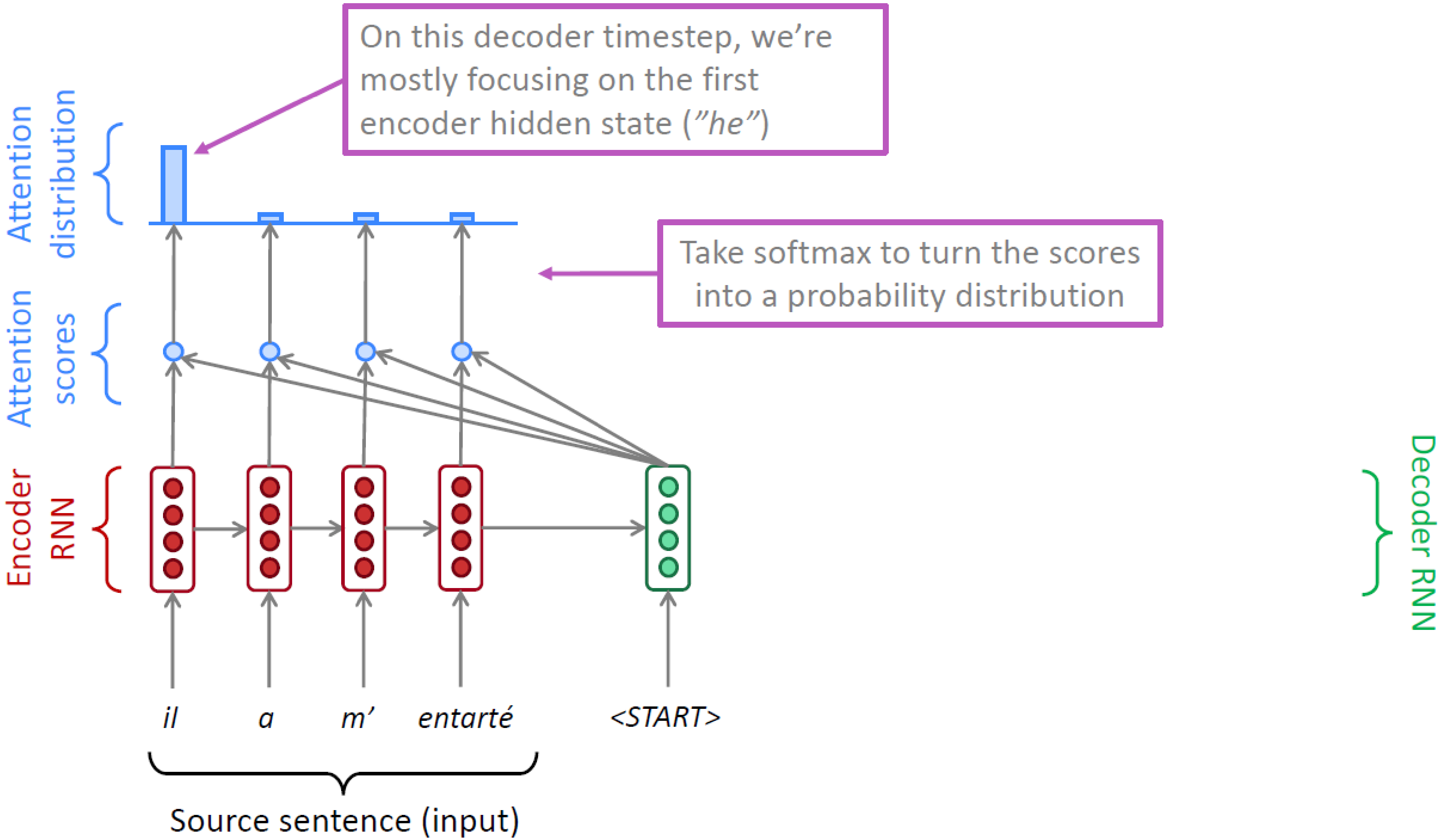
Sequence-to-Sequence with Attention



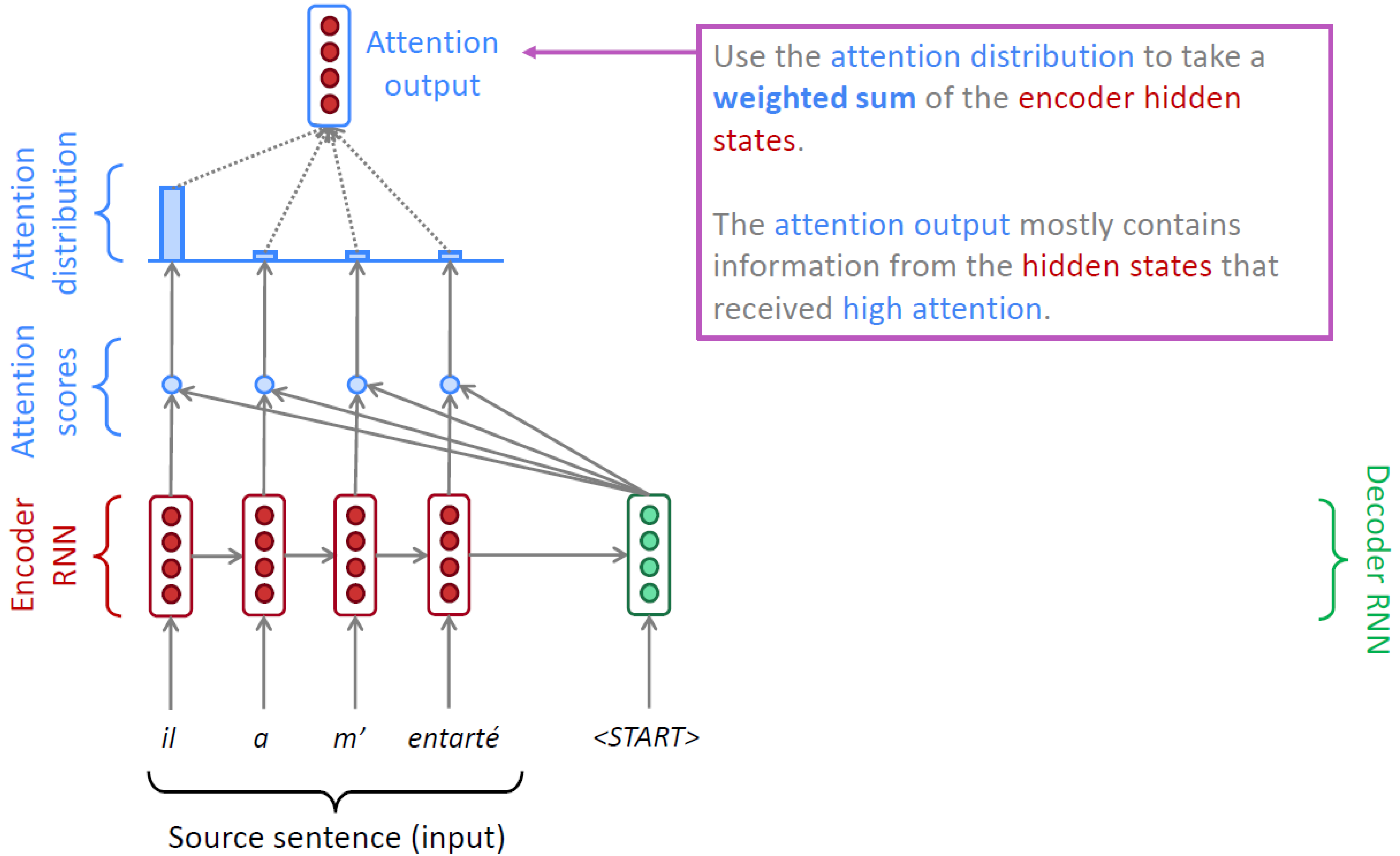
Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



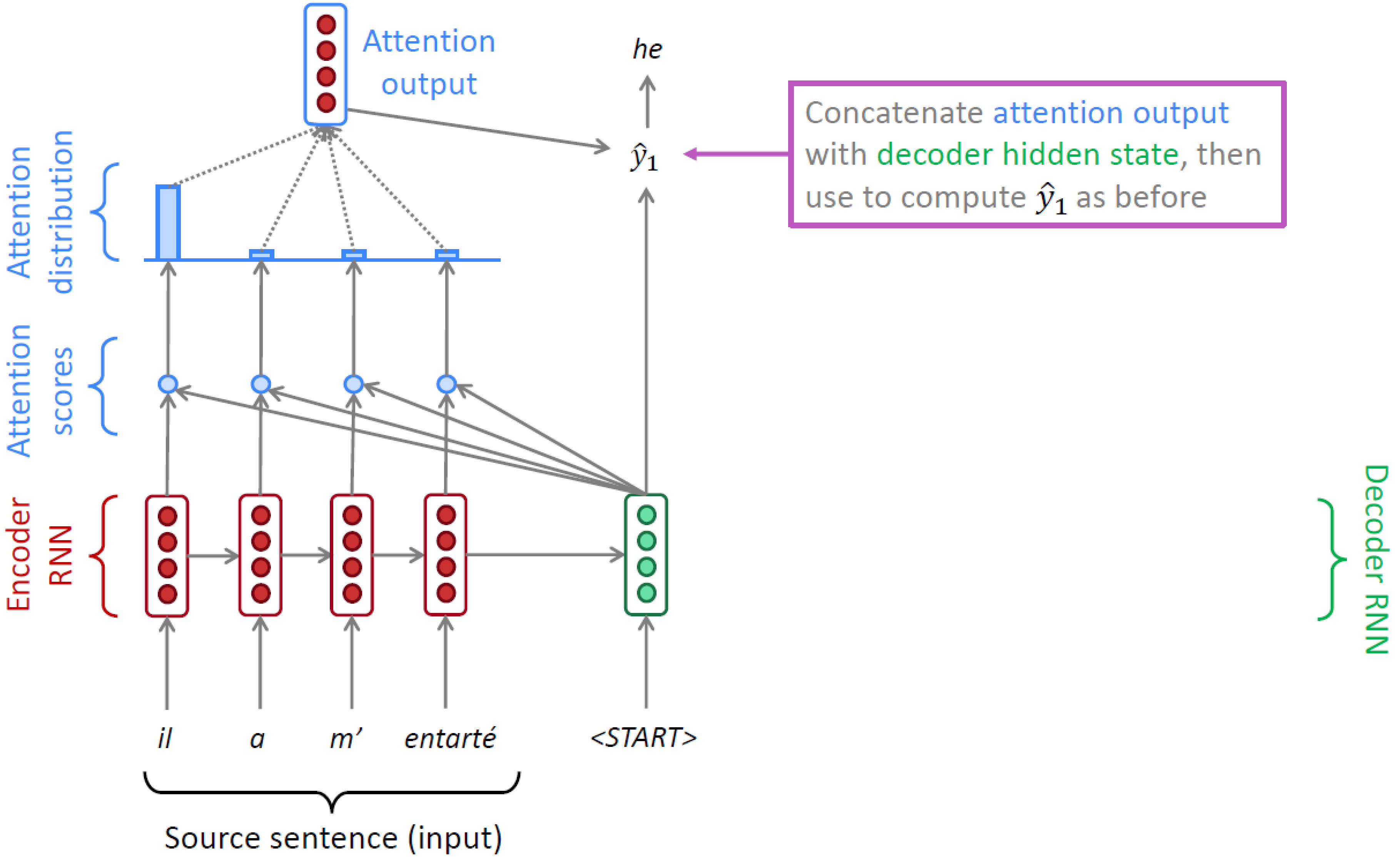
Sequence-to-Sequence with Attention



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

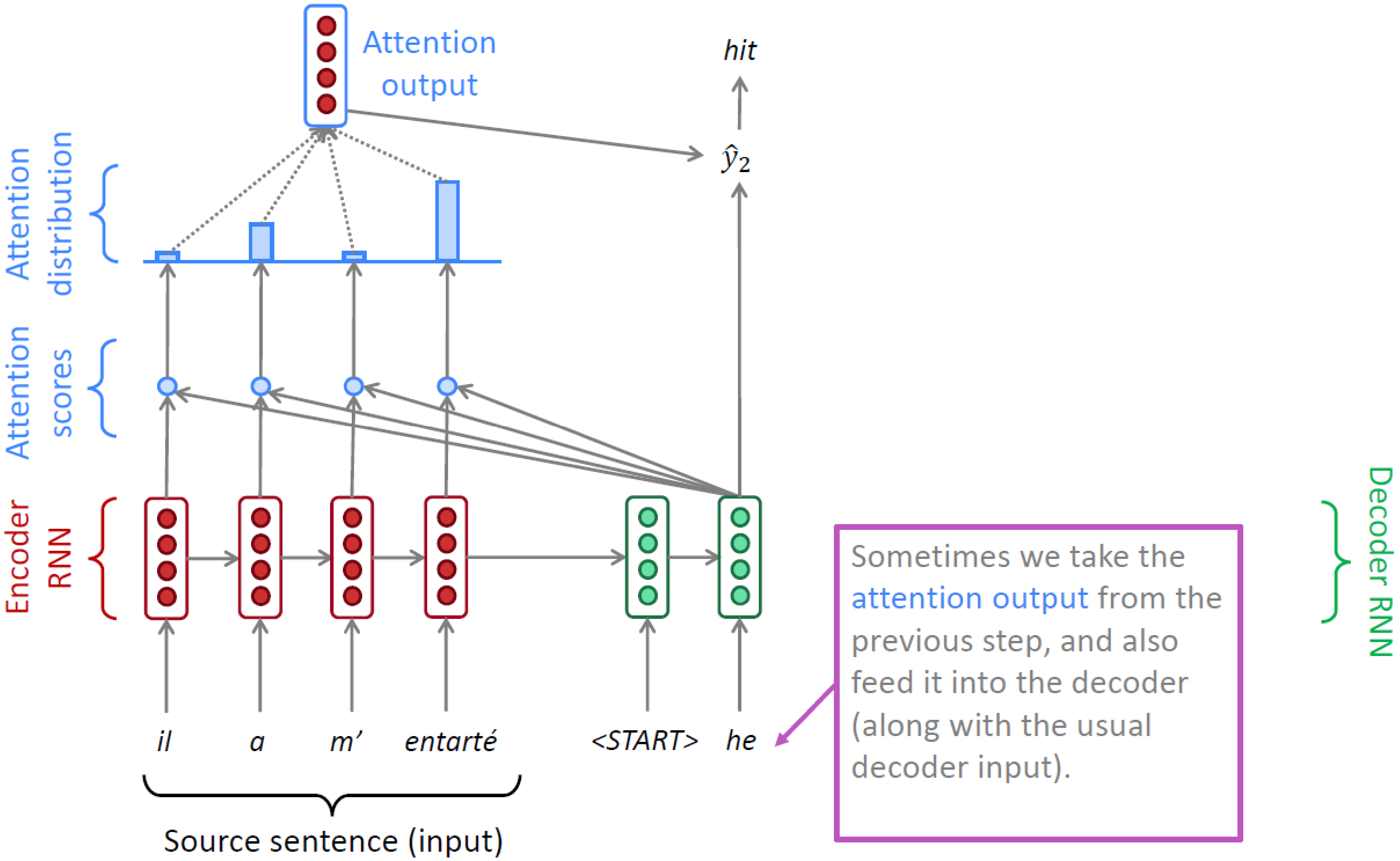
The attention output mostly contains information from the hidden states that received high attention.

Sequence-to-Sequence with Attention



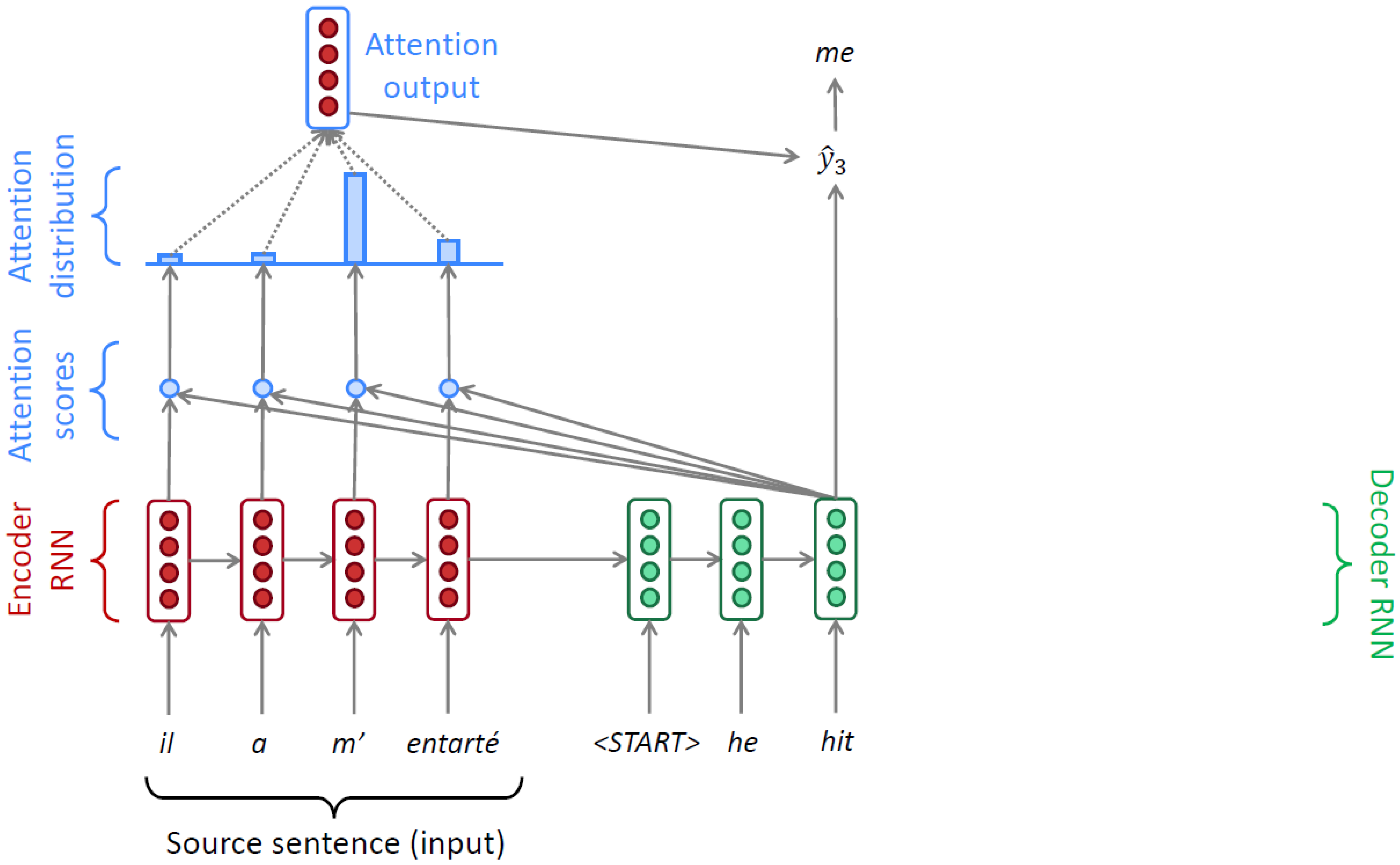
Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

Sequence-to-Sequence with Attention

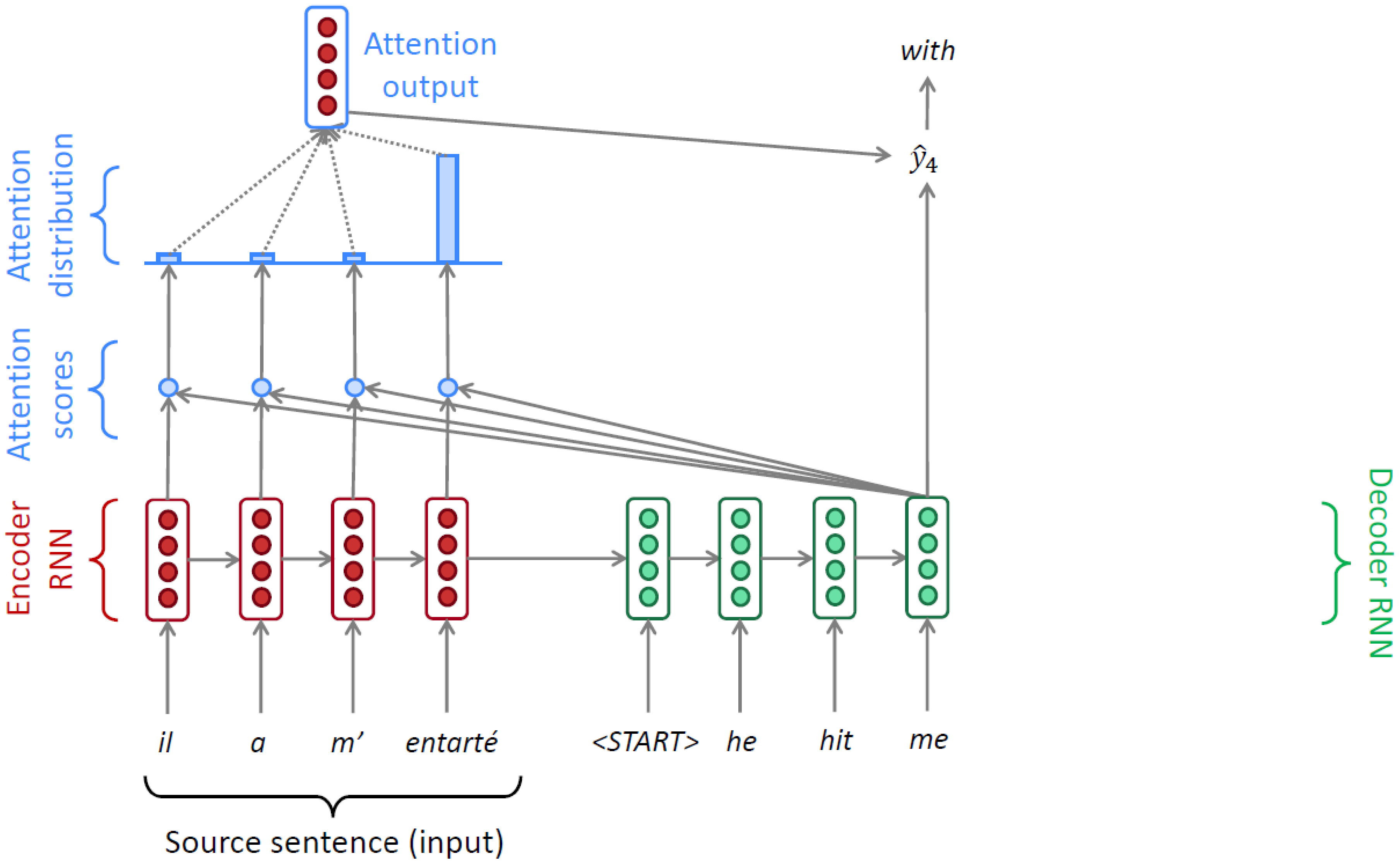


Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input).

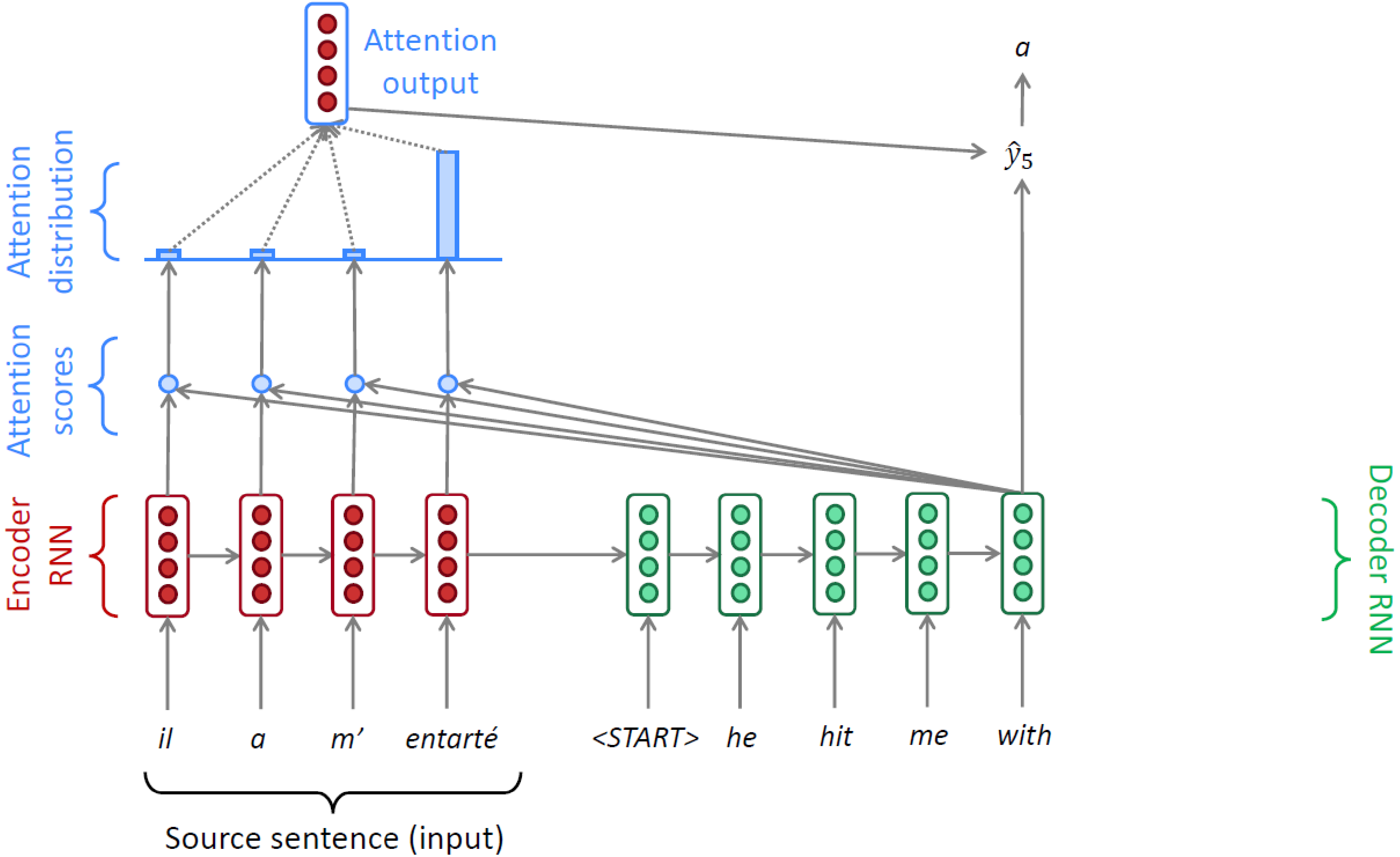
Sequence-to-Sequence with Attention



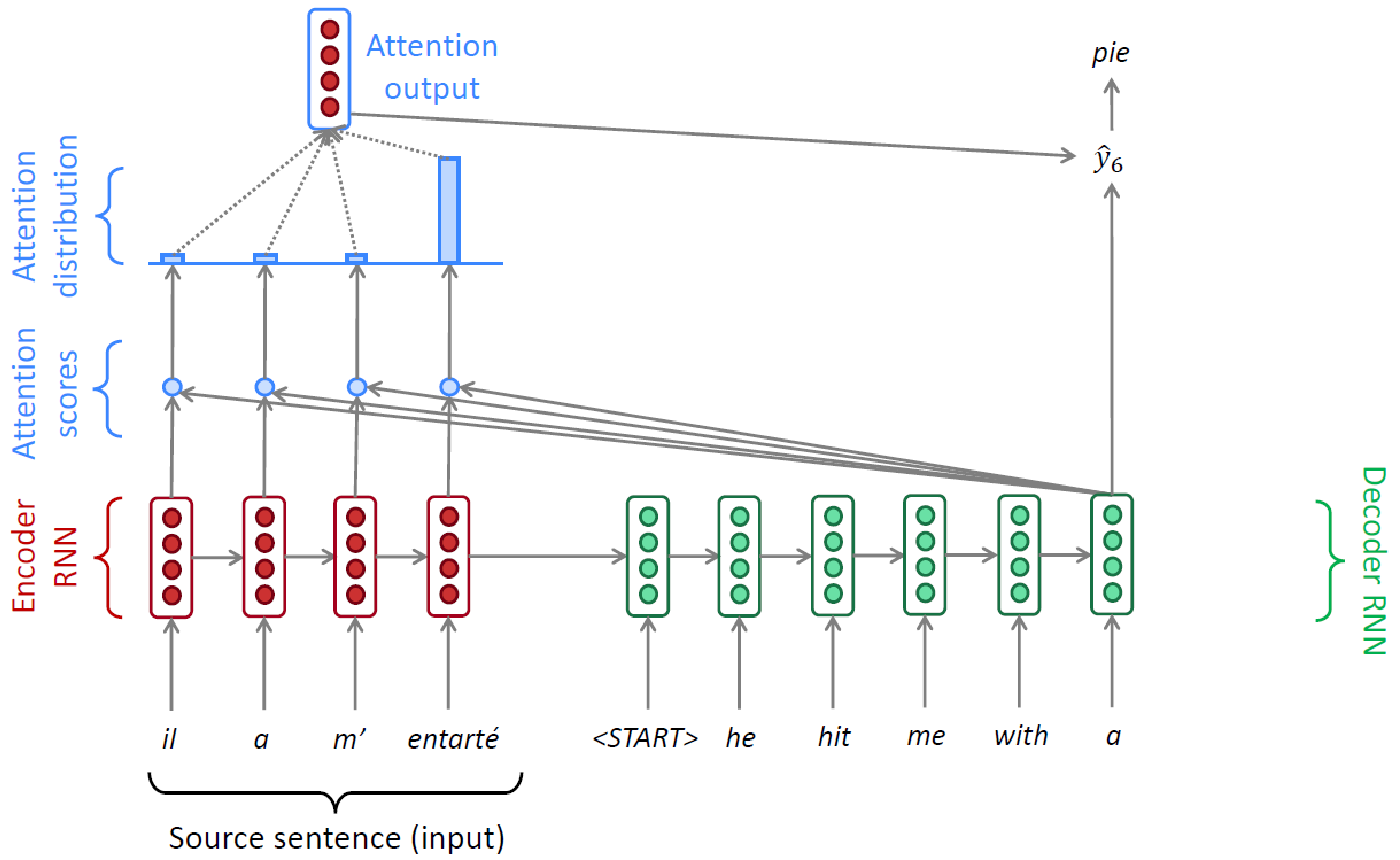
Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



Attention: In Equation

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

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

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

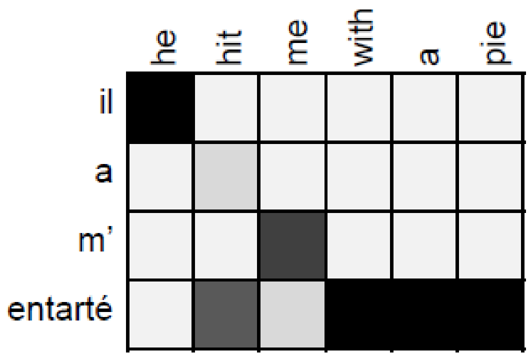
$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- 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

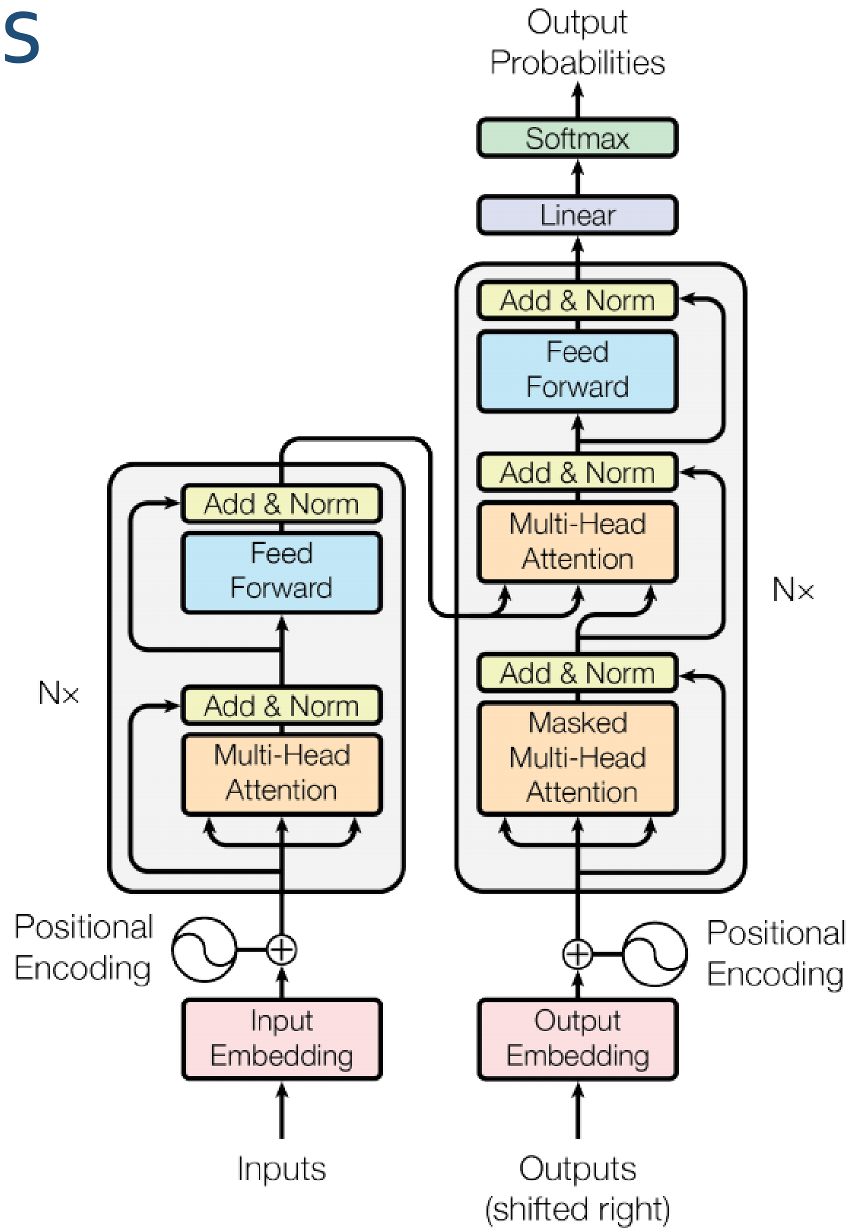
$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

Attention is Great

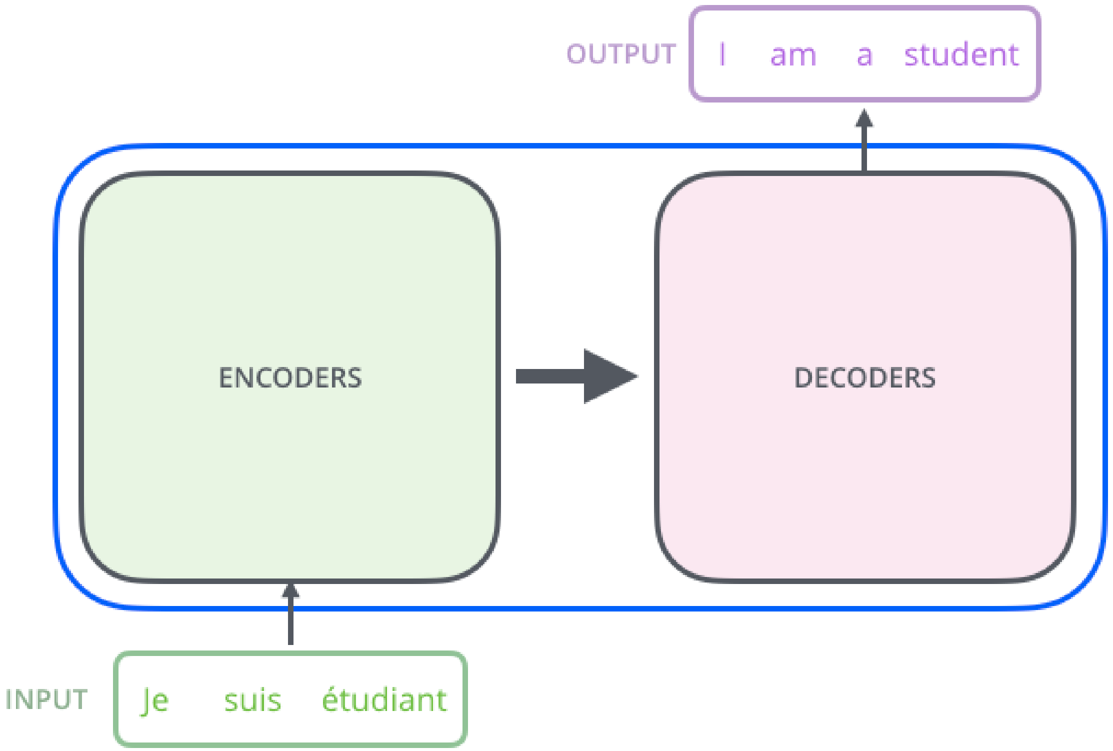
- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides 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

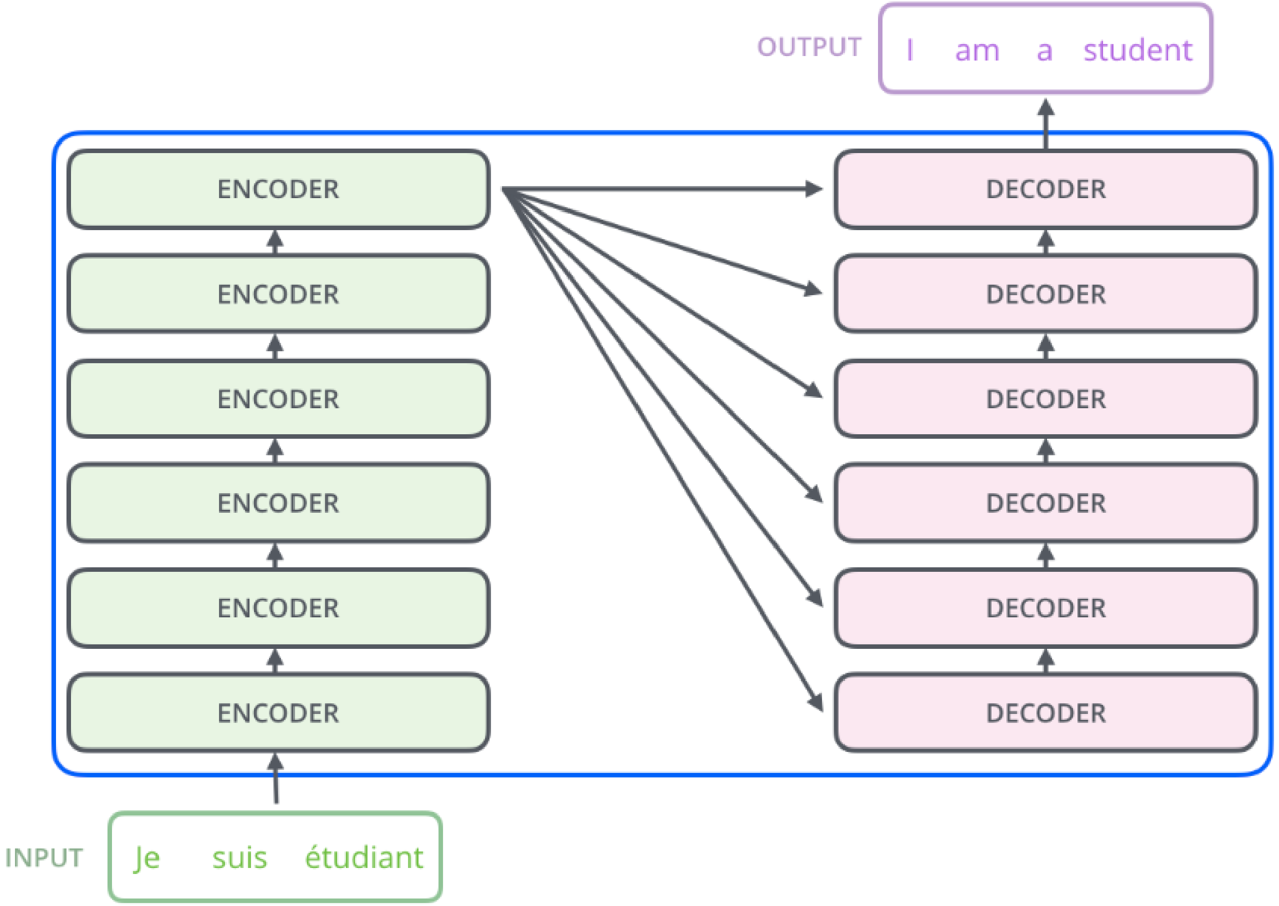


Attention is All You Need(NIPS 2017, Google) - Transformers



Seq2Seq vs Transformer

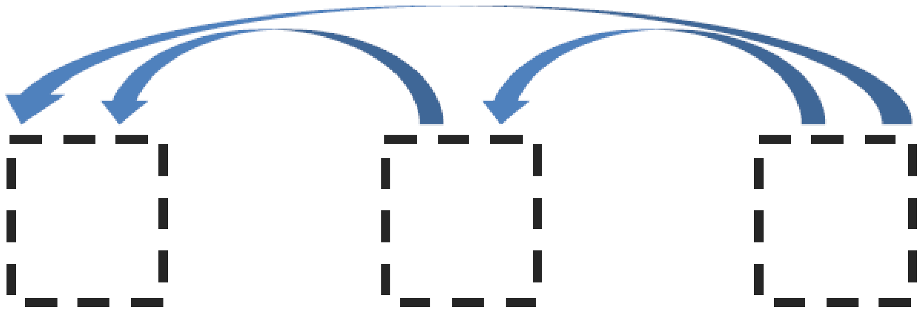




3 Types of Attention

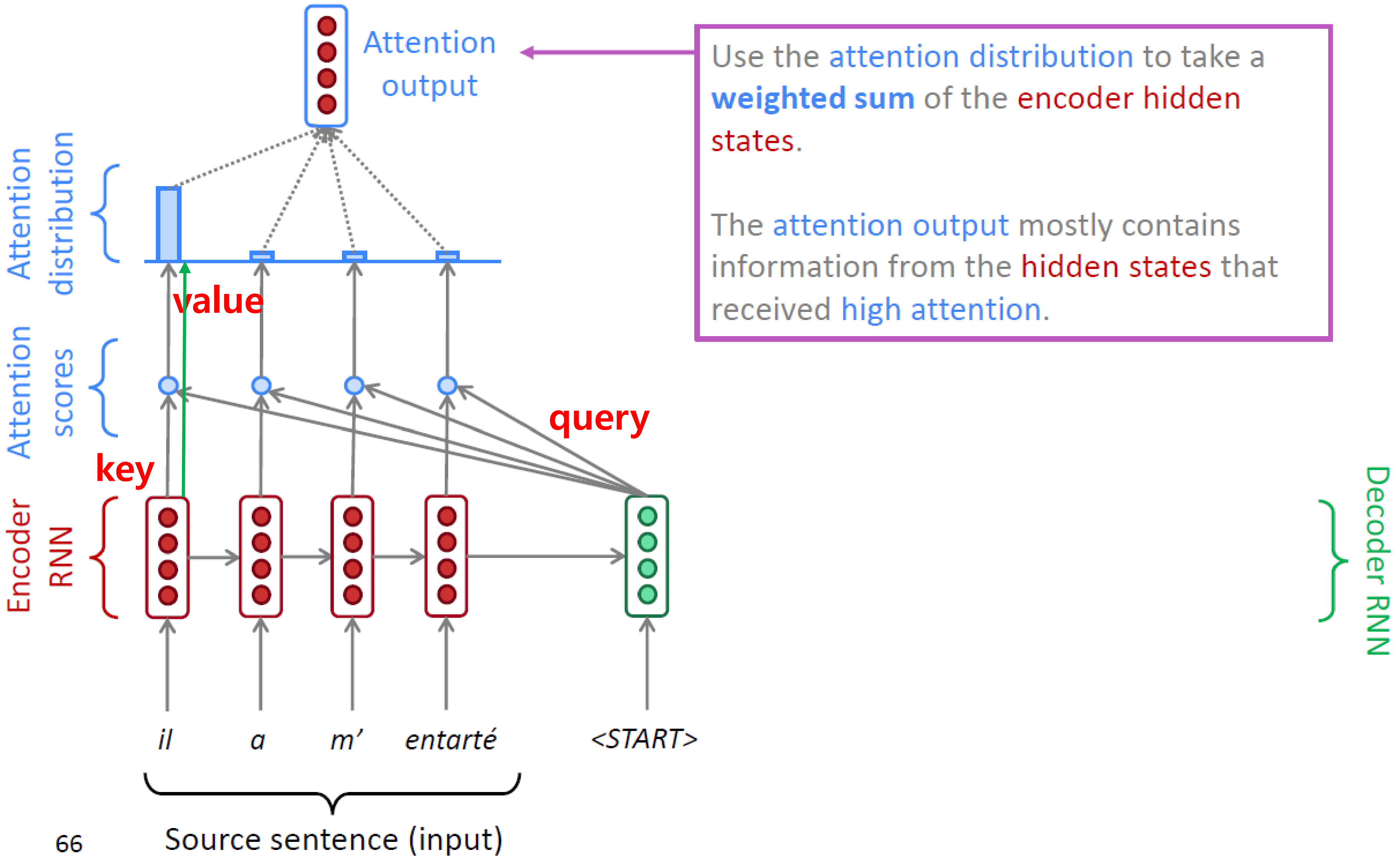
Encoder-Decoder Attention

Encoder Self-Attention



MaskedDecoder Self-Attention

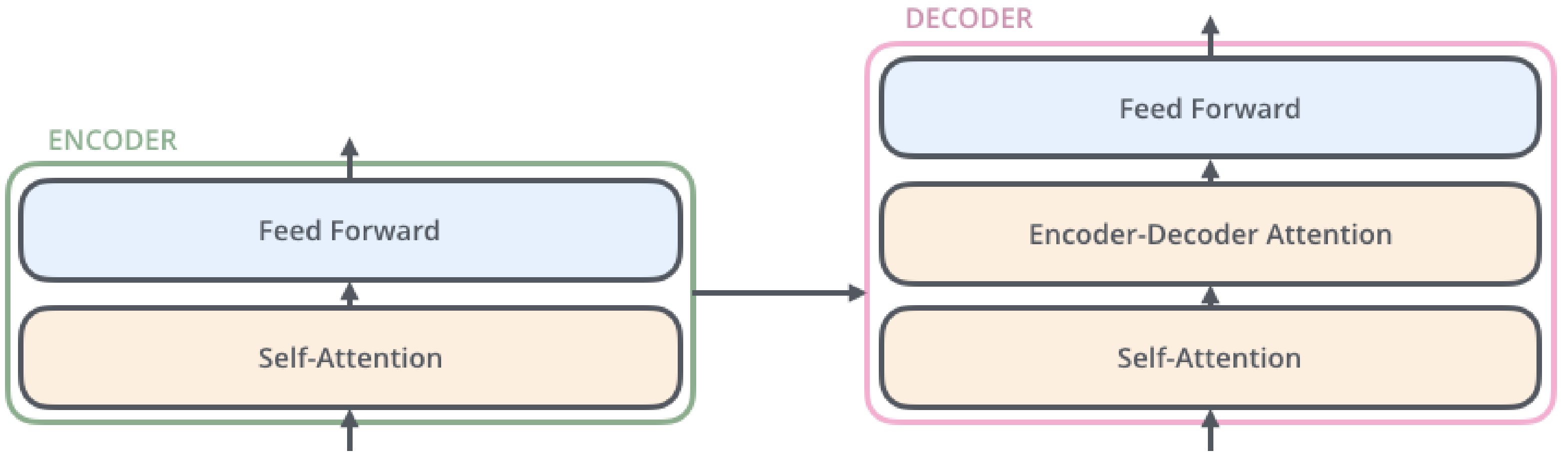
Query, Key, Value

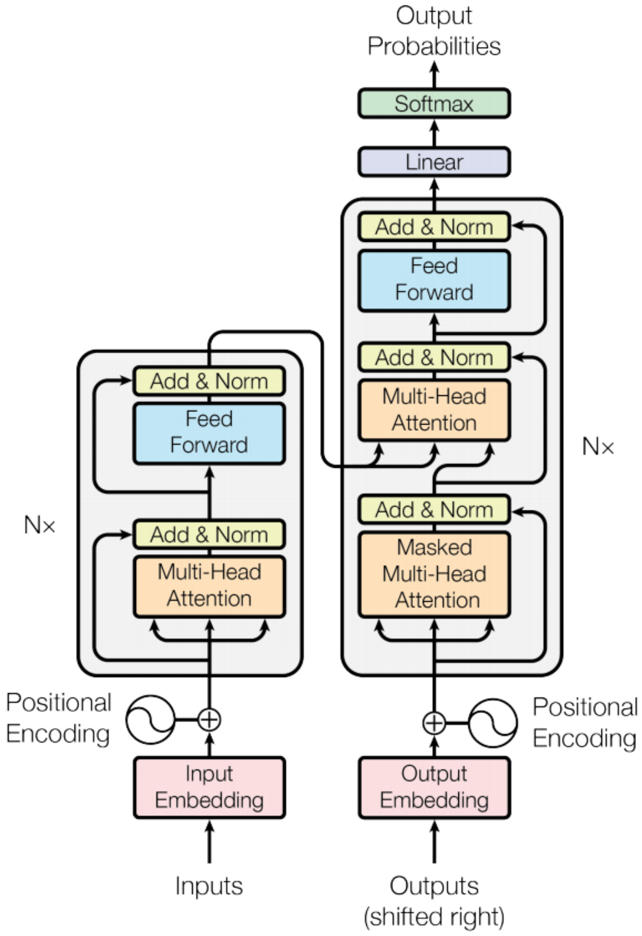


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

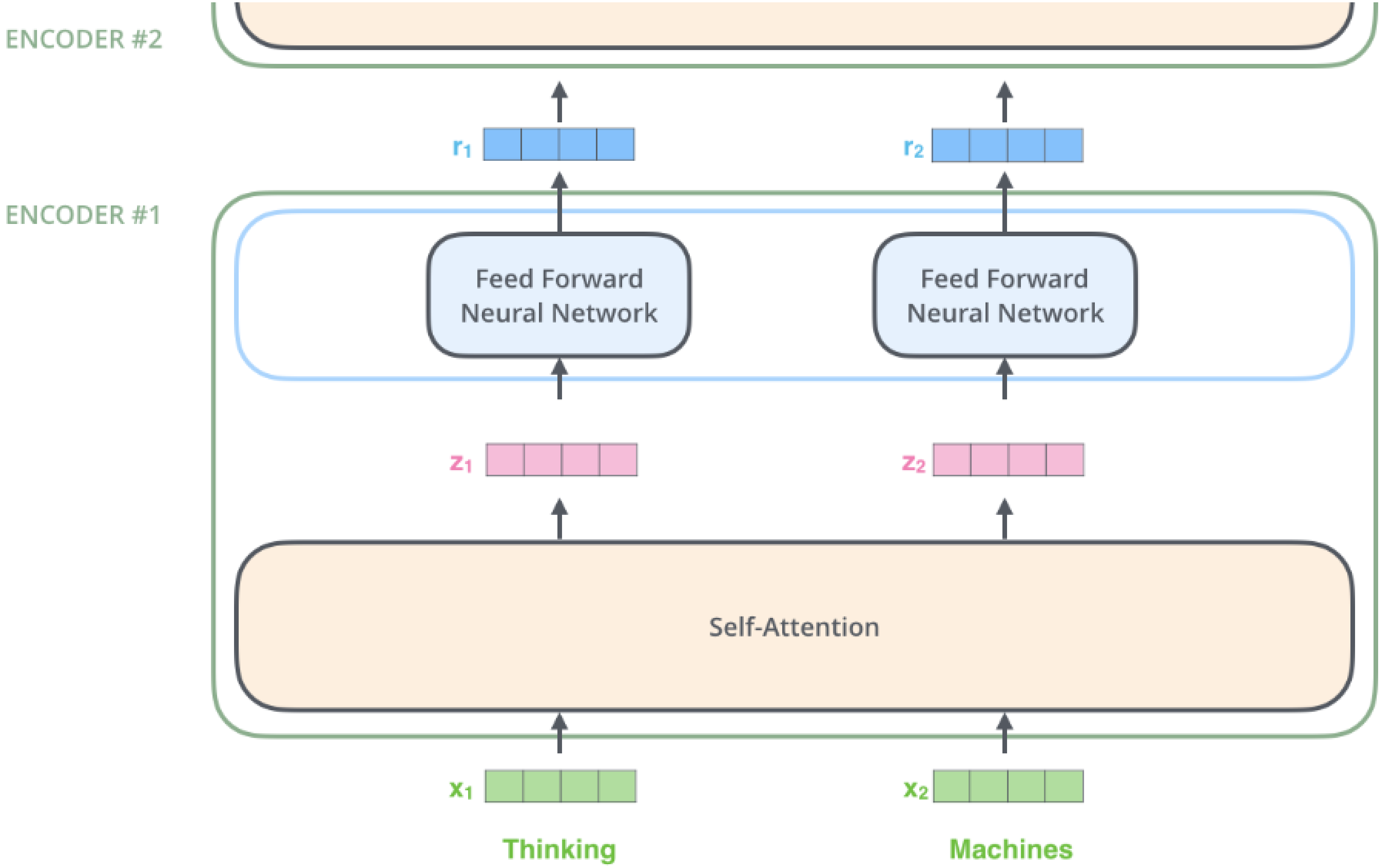
The attention output mostly contains information from the hidden states that received high attention.

Encoder-Decoder of Transformer



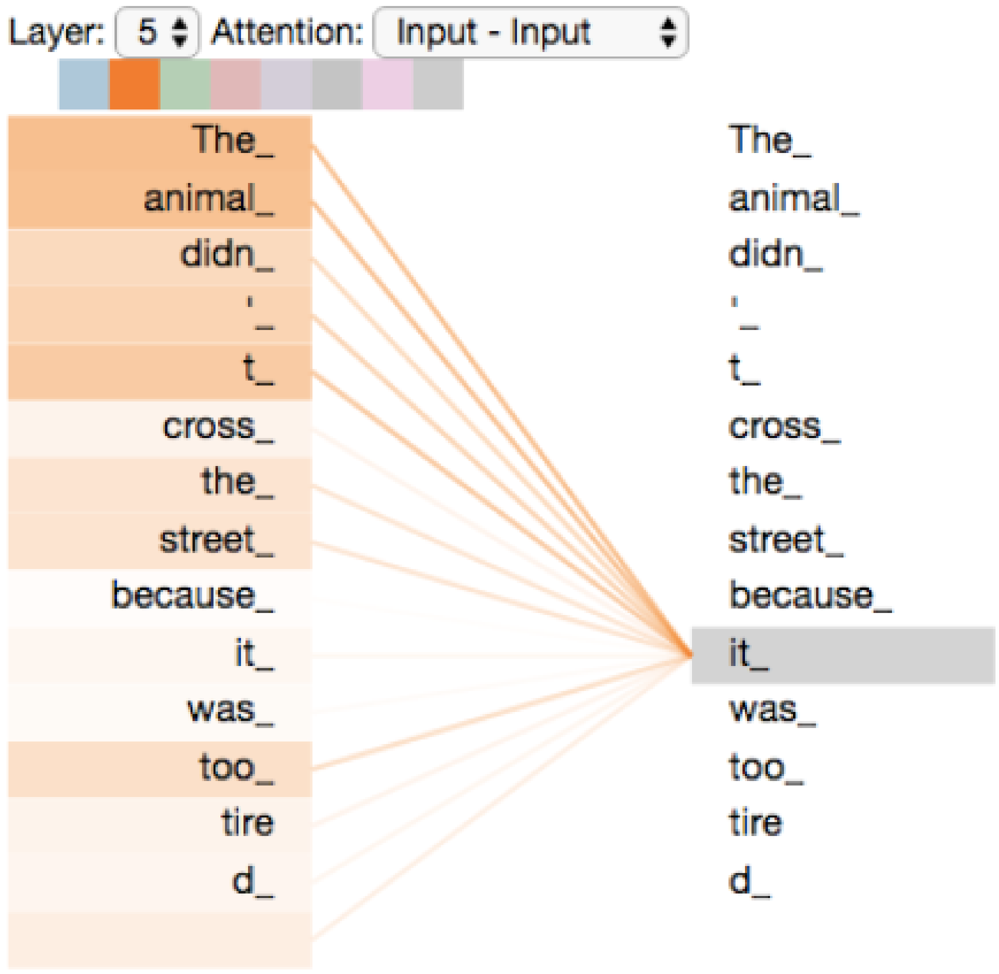


Encoder

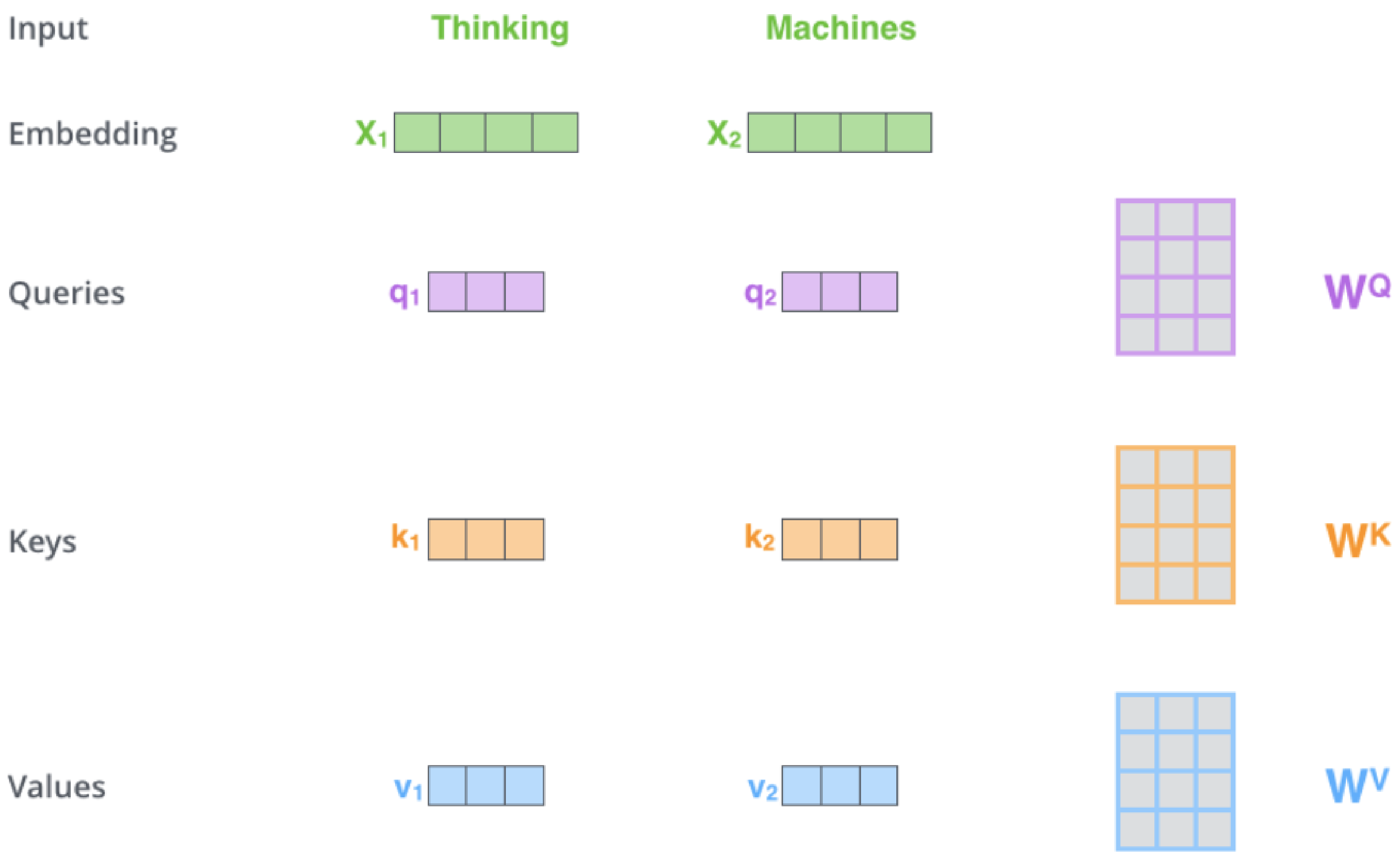


ENCODER #2
r_1
r_2
ENCODER #1
Feed Forward Neural Network
Feed Forward Neural Network
z_1
z_2
Self-Attention
x_1
x_2
Thinking
Machines

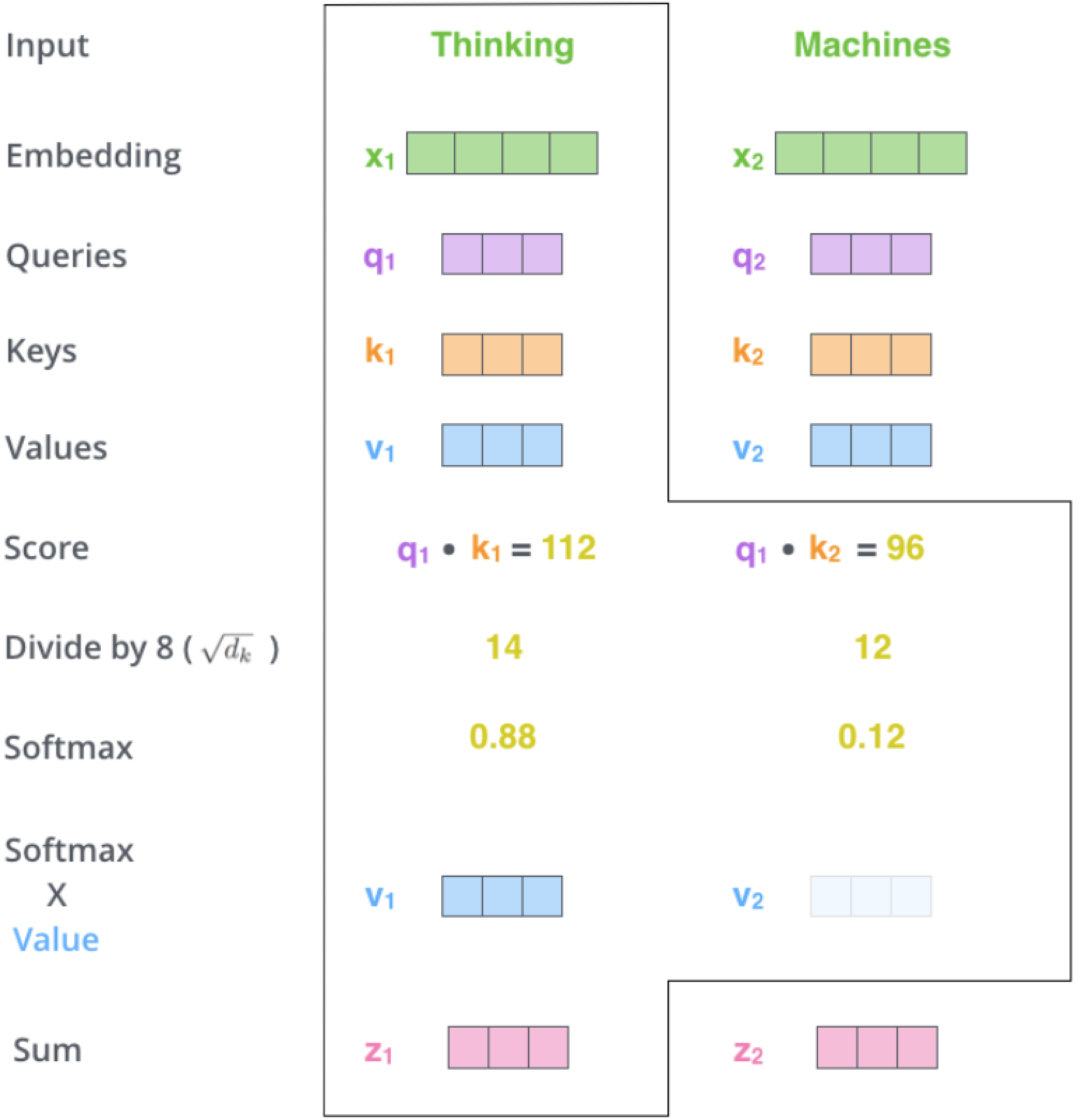
Self Attention



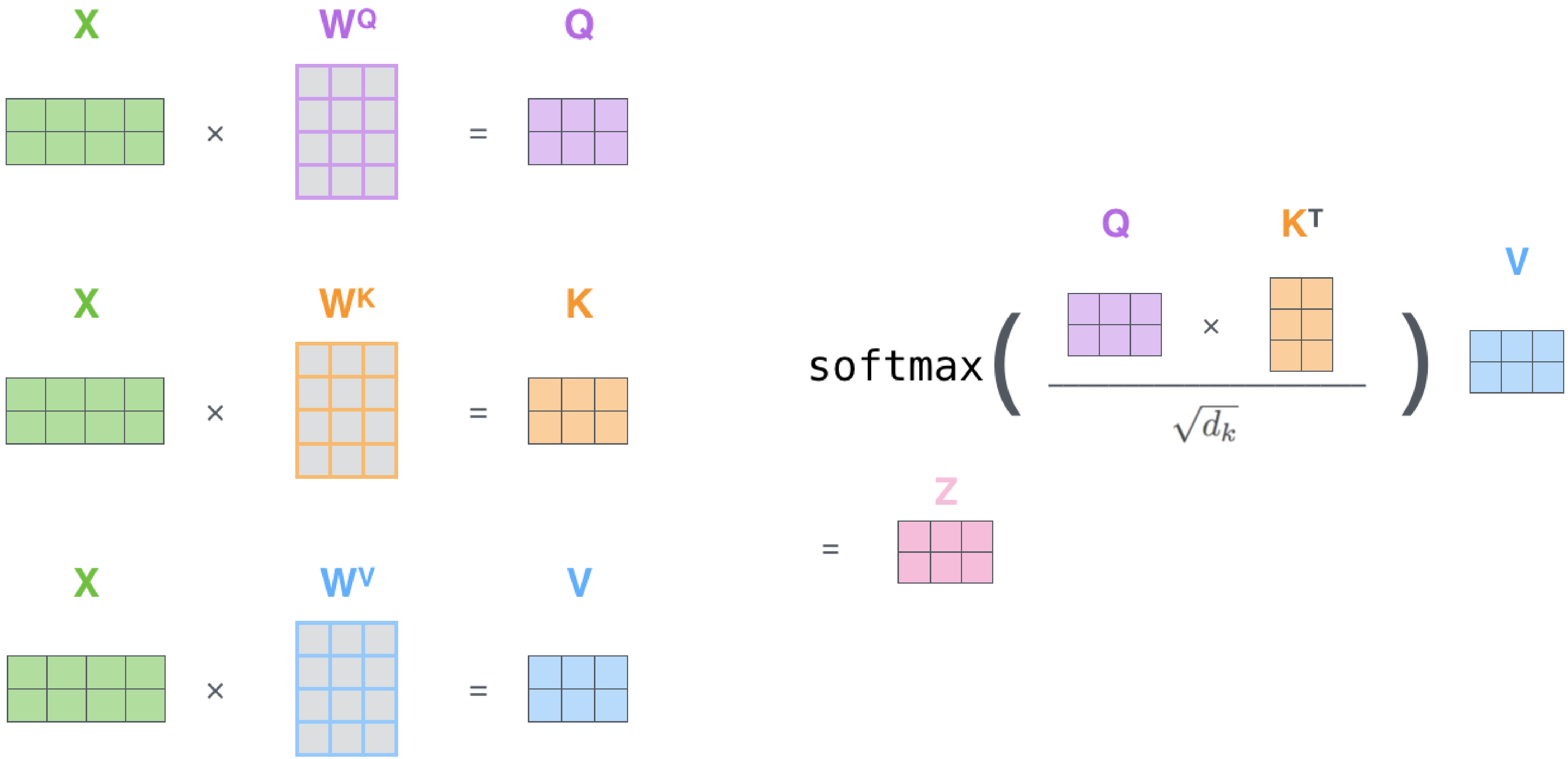
Self Attention



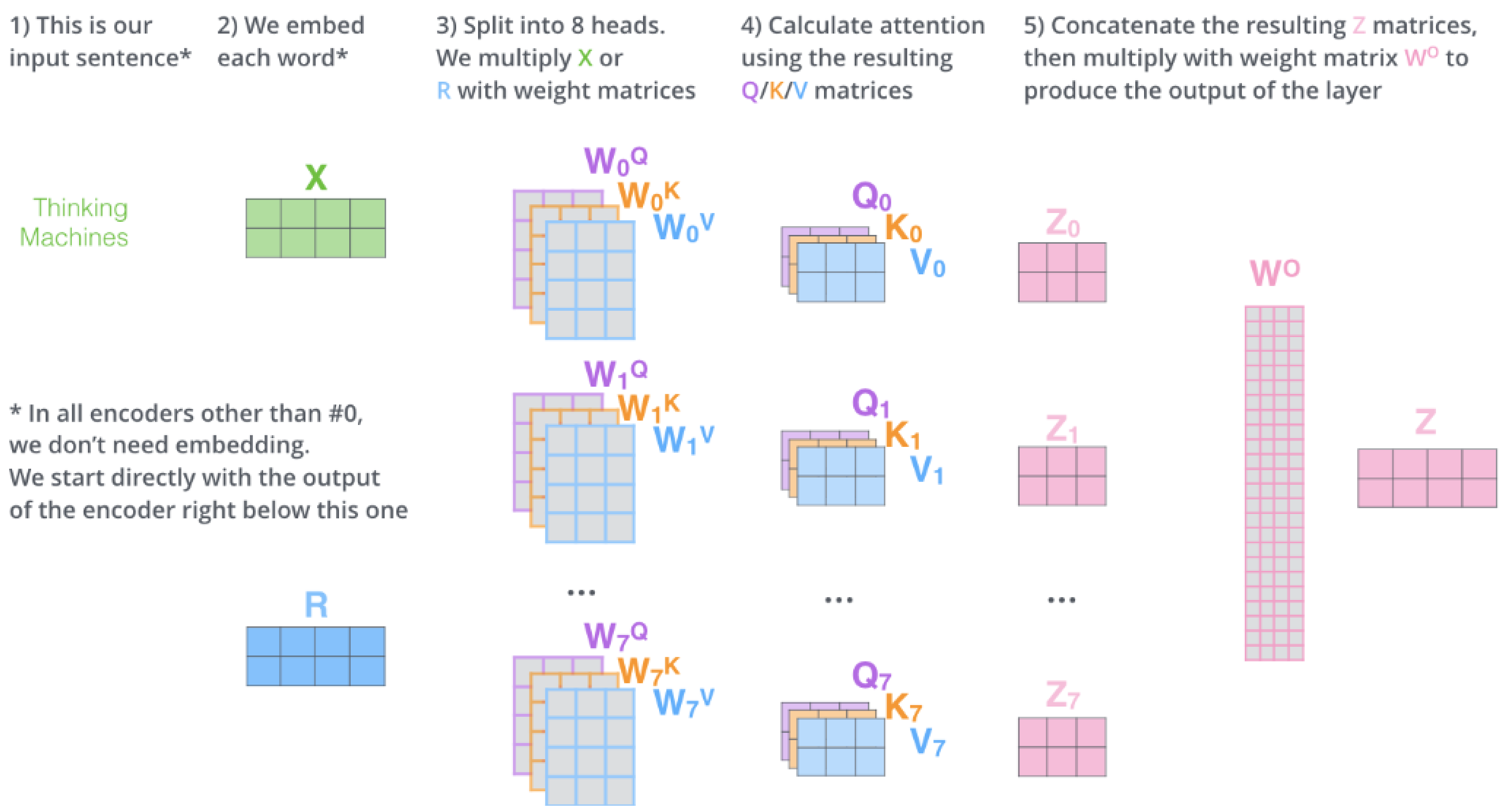
Self Attention



Matrix Calculation of Self-Attention

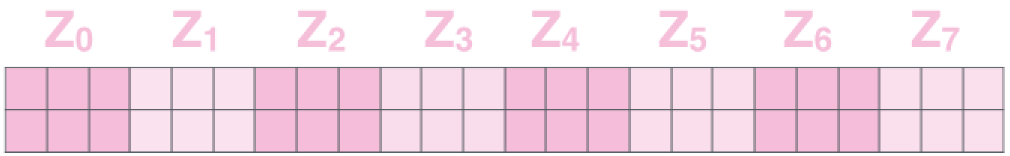


Multi-head Attention



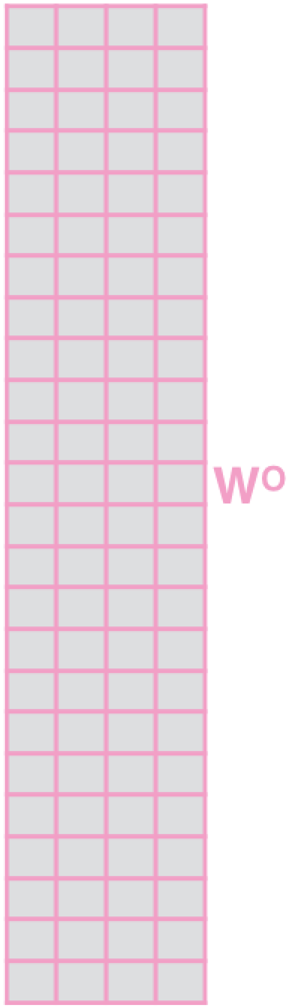
Multi-head Attention

1) Concatenate all the attention heads



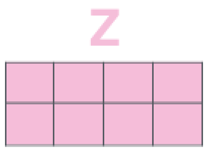
2) Multiply with a weight matrix W^O that was trained jointly with the model

X

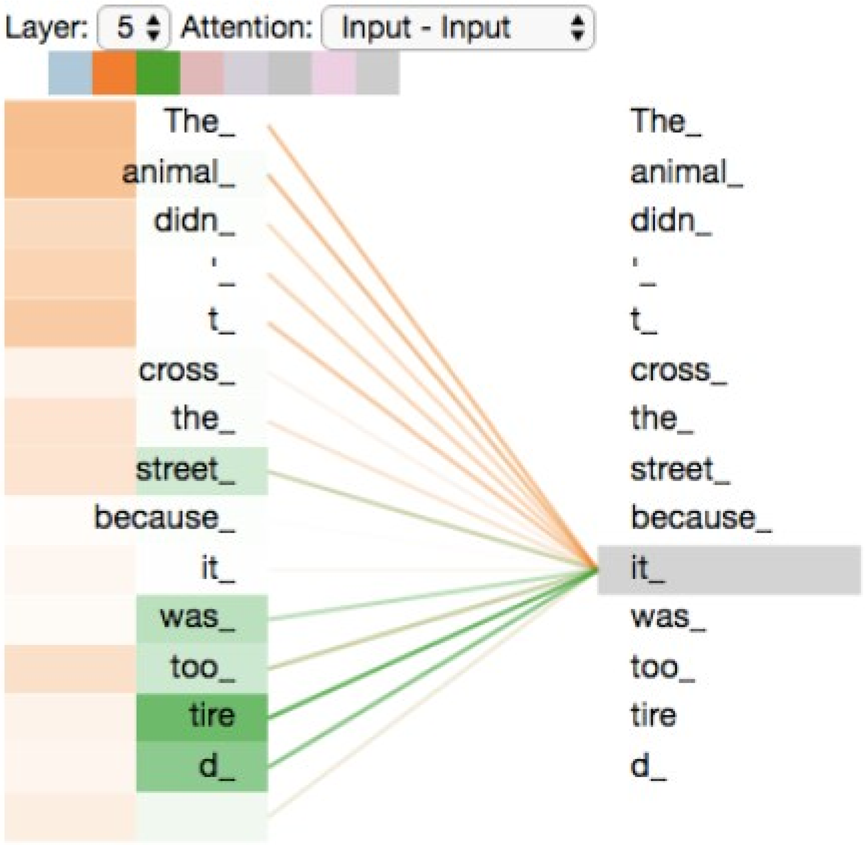


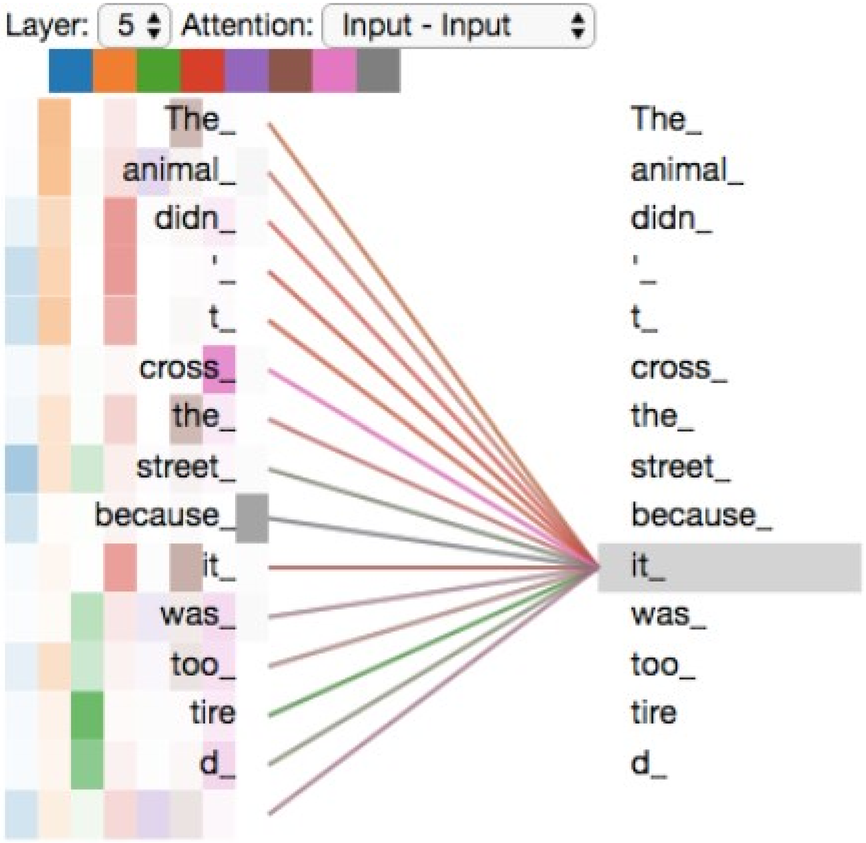
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=

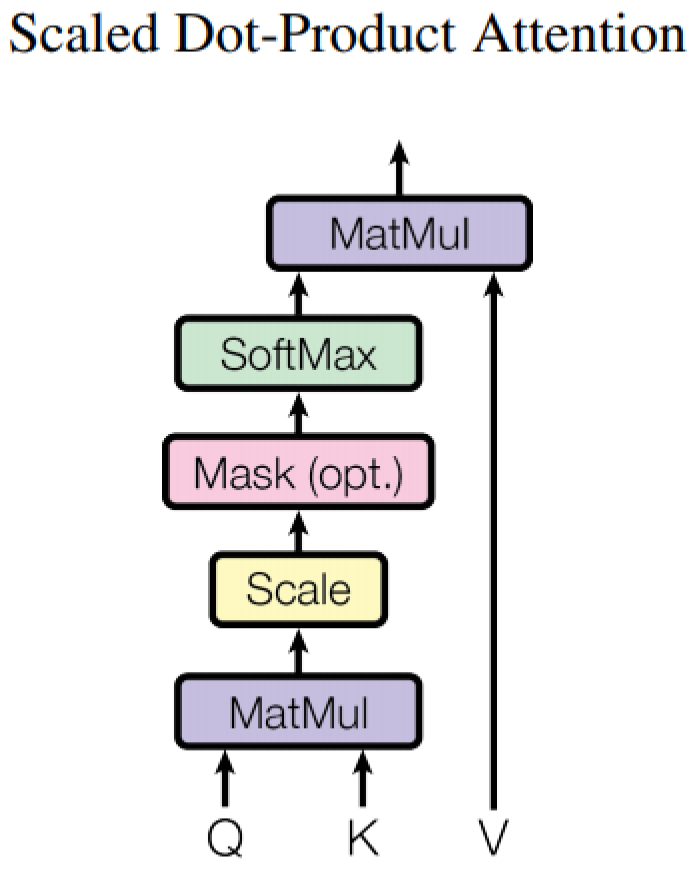


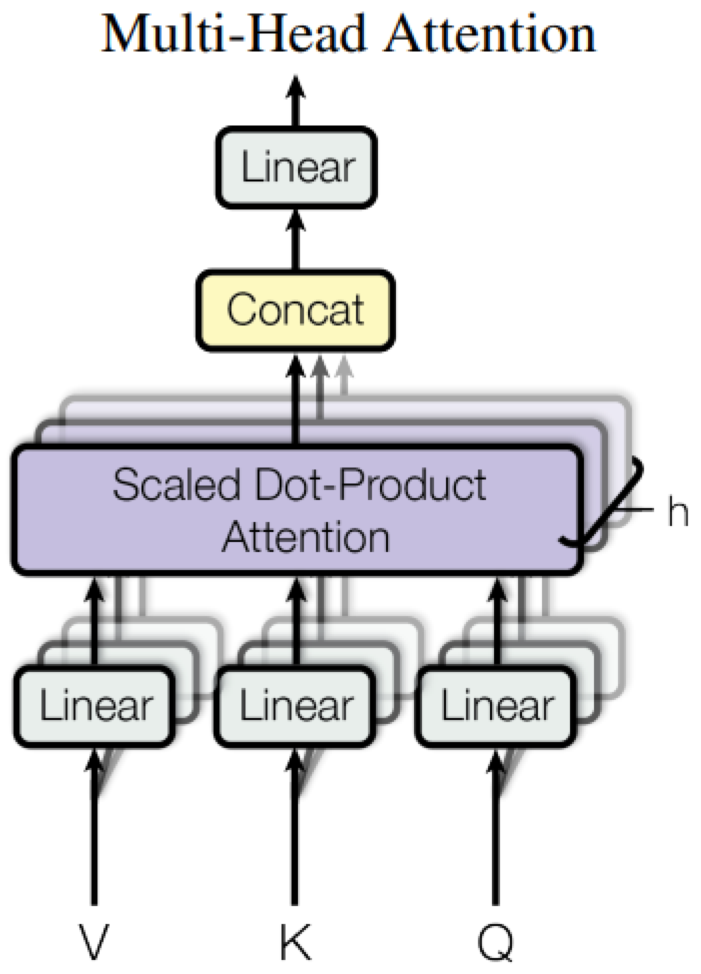
Multi-head Attention



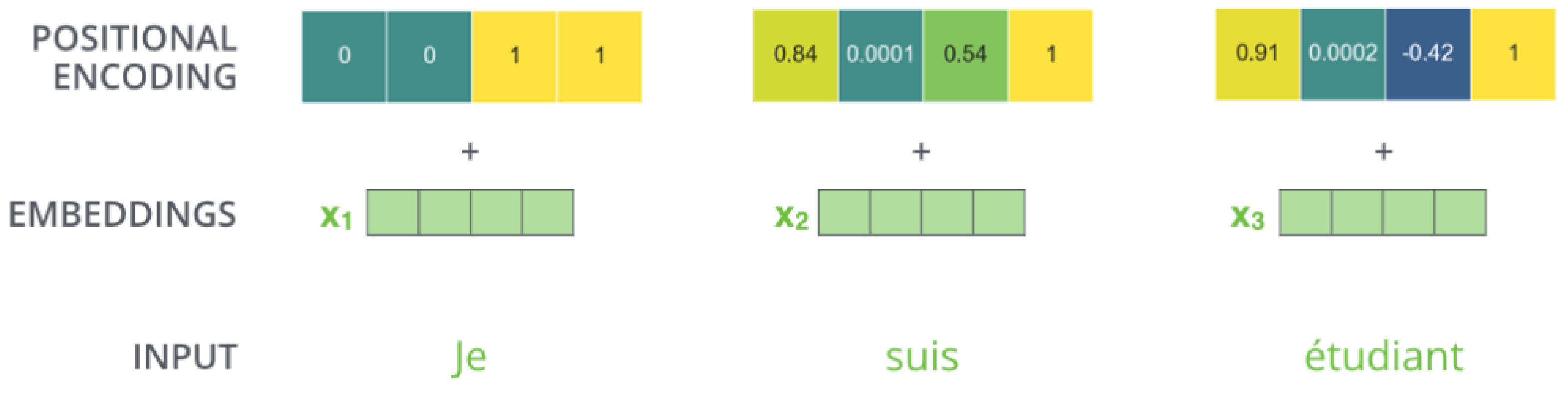


Self-Attention of Transformer

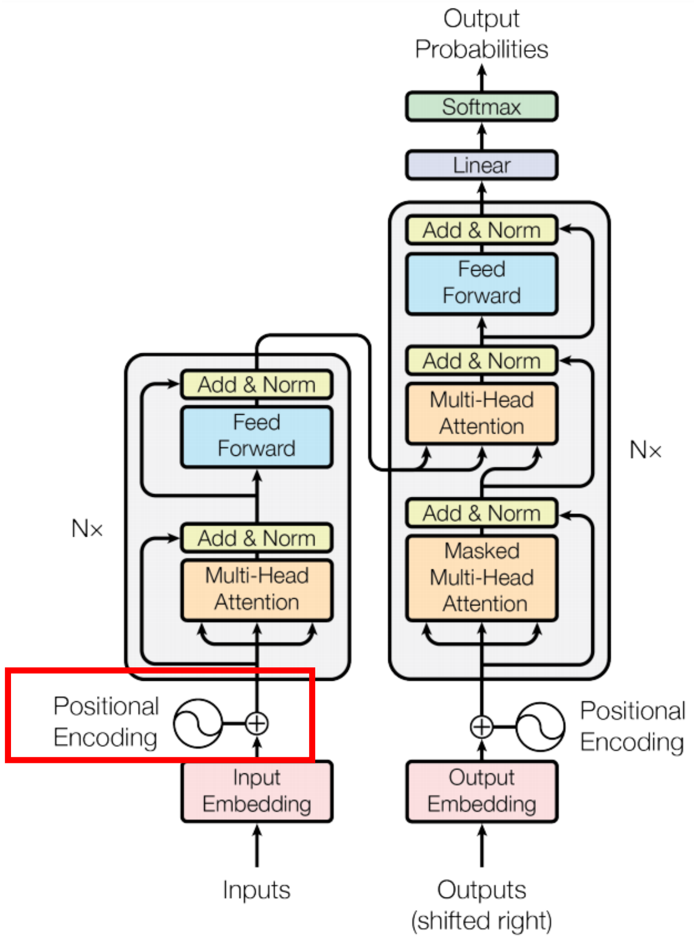




Positional Encoding

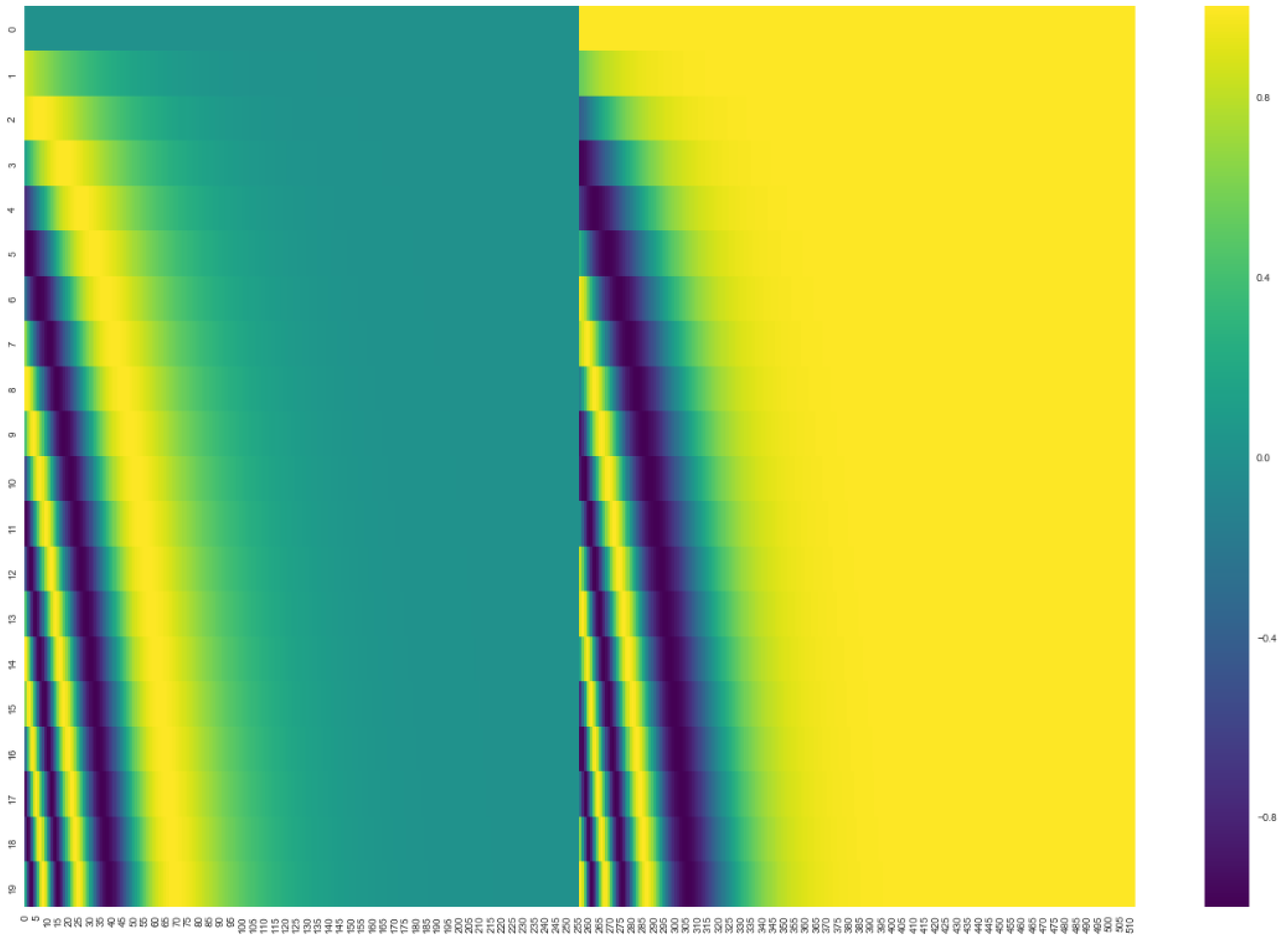


A real example of positional encoding with a toy embedding size of 4

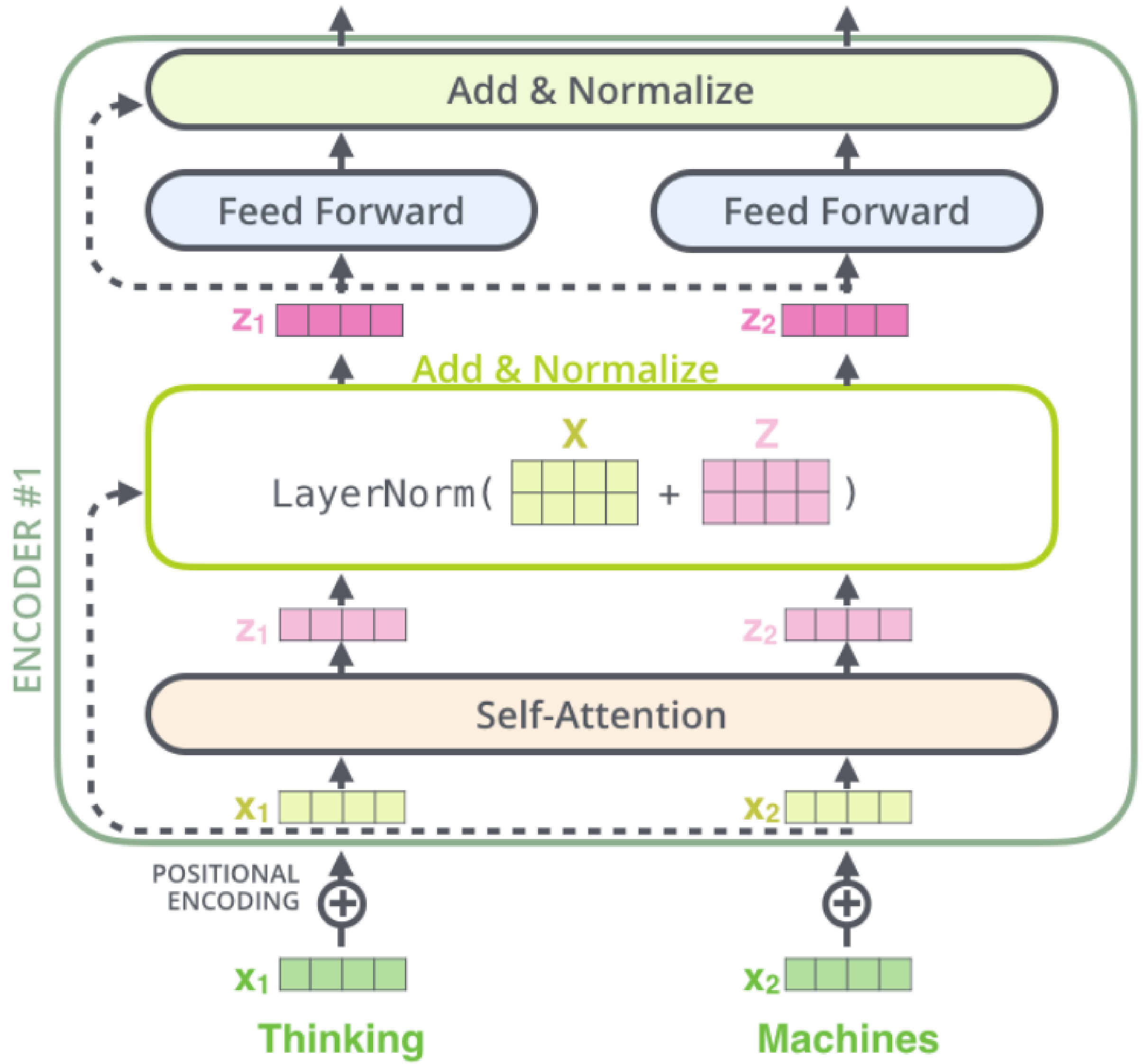


Positional Encoding

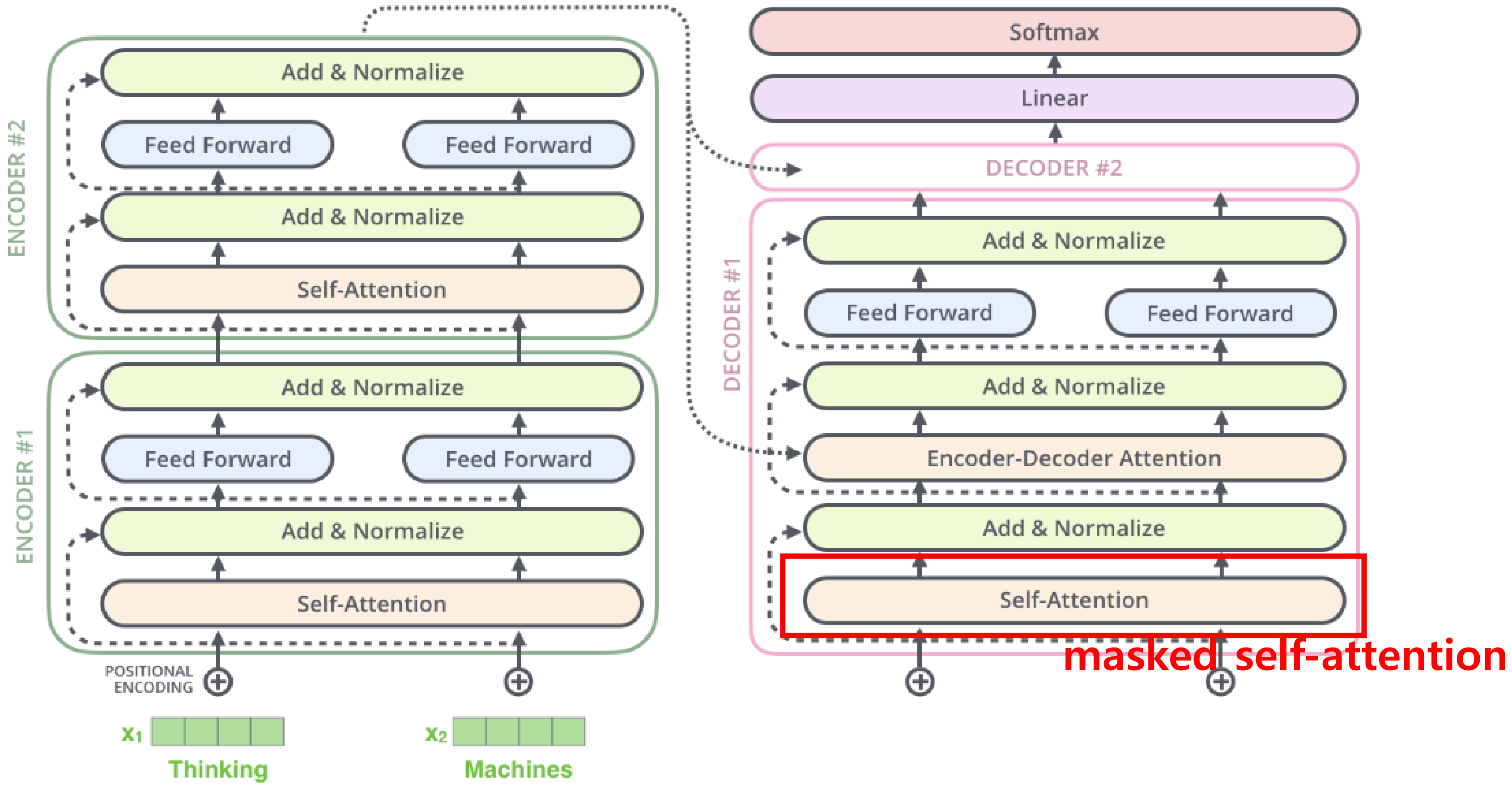
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



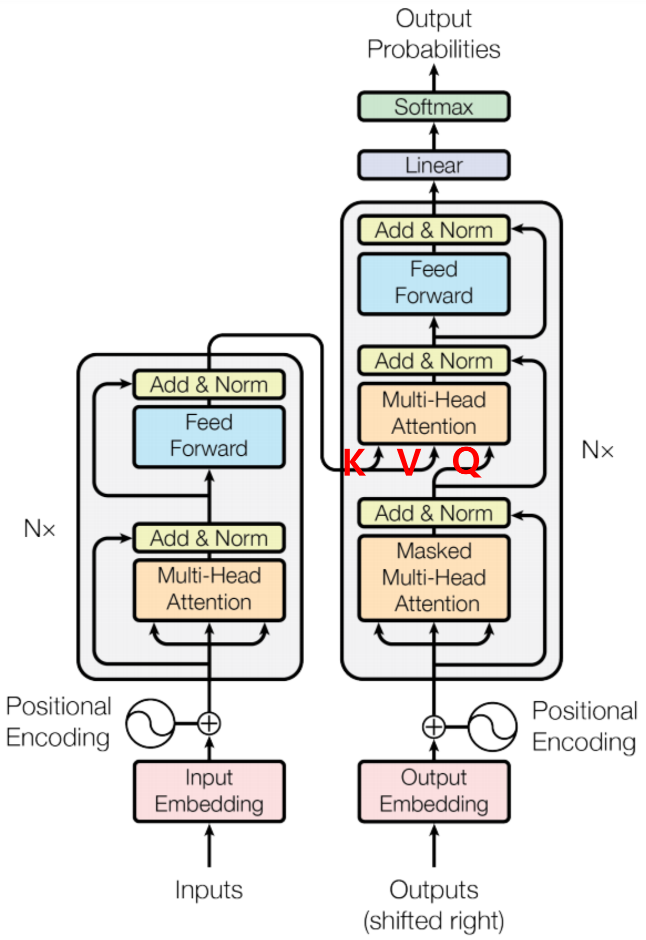
Skip Connection & Layer Norm



Decoder



Softmax
Linear
DECODER #2
DECODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Encoder-Decoder Attention
Add & Normalize
Self-Attention
masked self-attention
ENCODER #2
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
POSITIONAL ENCODING
x_1
Thinking
x_2
Machines



Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
K V Q
Add & Norm
Masked Multi-Head Attention
N×
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N×
Positional Encoding
Input Embedding
Inputs
Positional Encoding
Output Embedding
Outputs (shifted right)

Transformer

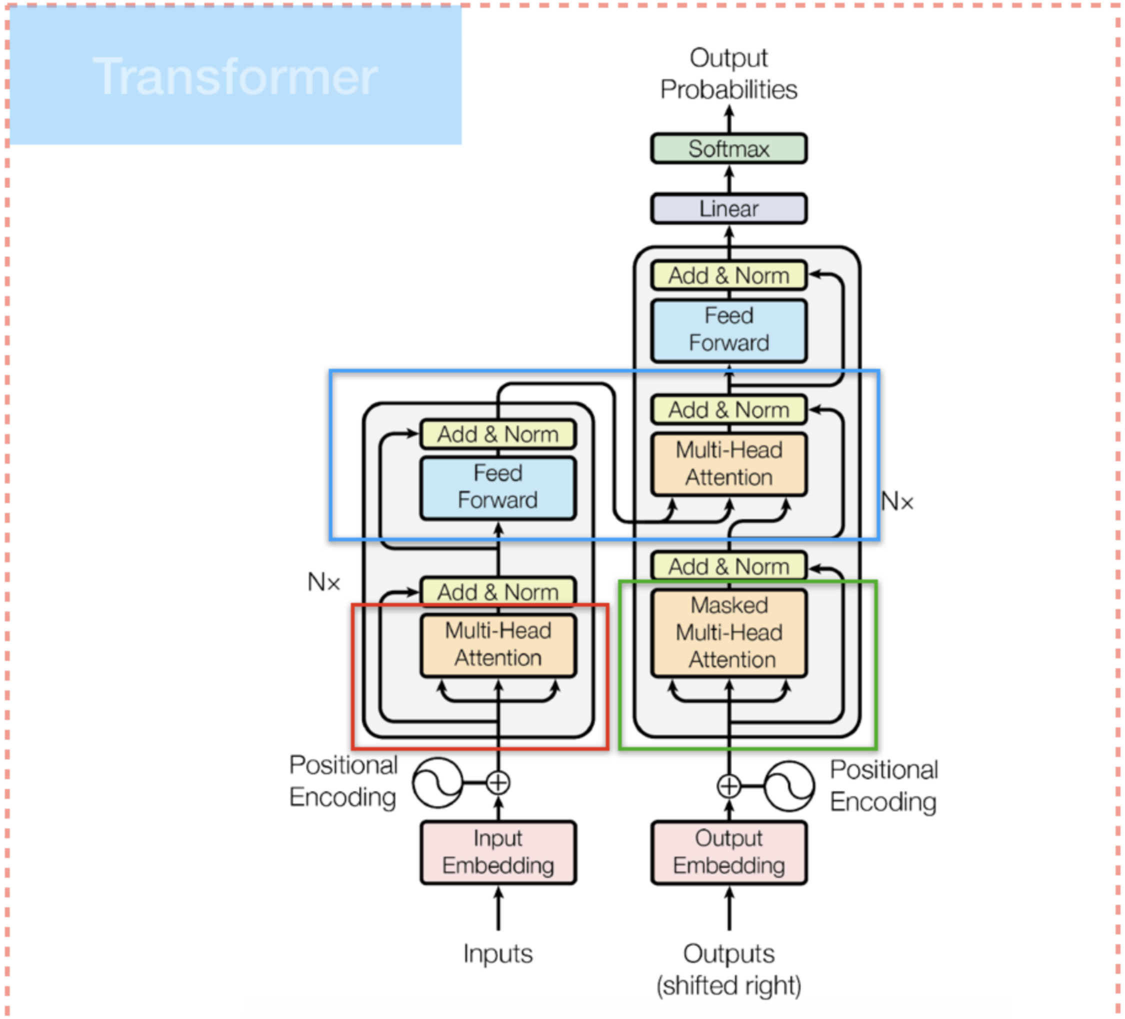


Figure 1: The Transformer - model architecture.

encoder self attention

1. **Multi-head Attention**
2. **Query=Key=Value**

decoder self attention

1. **Masked Multi-head Attention**
2. **Query=Key=Value**

encoder-decoder attention

1. **Multi-head Attention**
2. **Encoder Self attention=Key=Value**
3. **Decoder Self attention=Query**