Deep Learning in Applications

# Lecture 4: Attention

**Anastasia Ianina**

11.06.2020

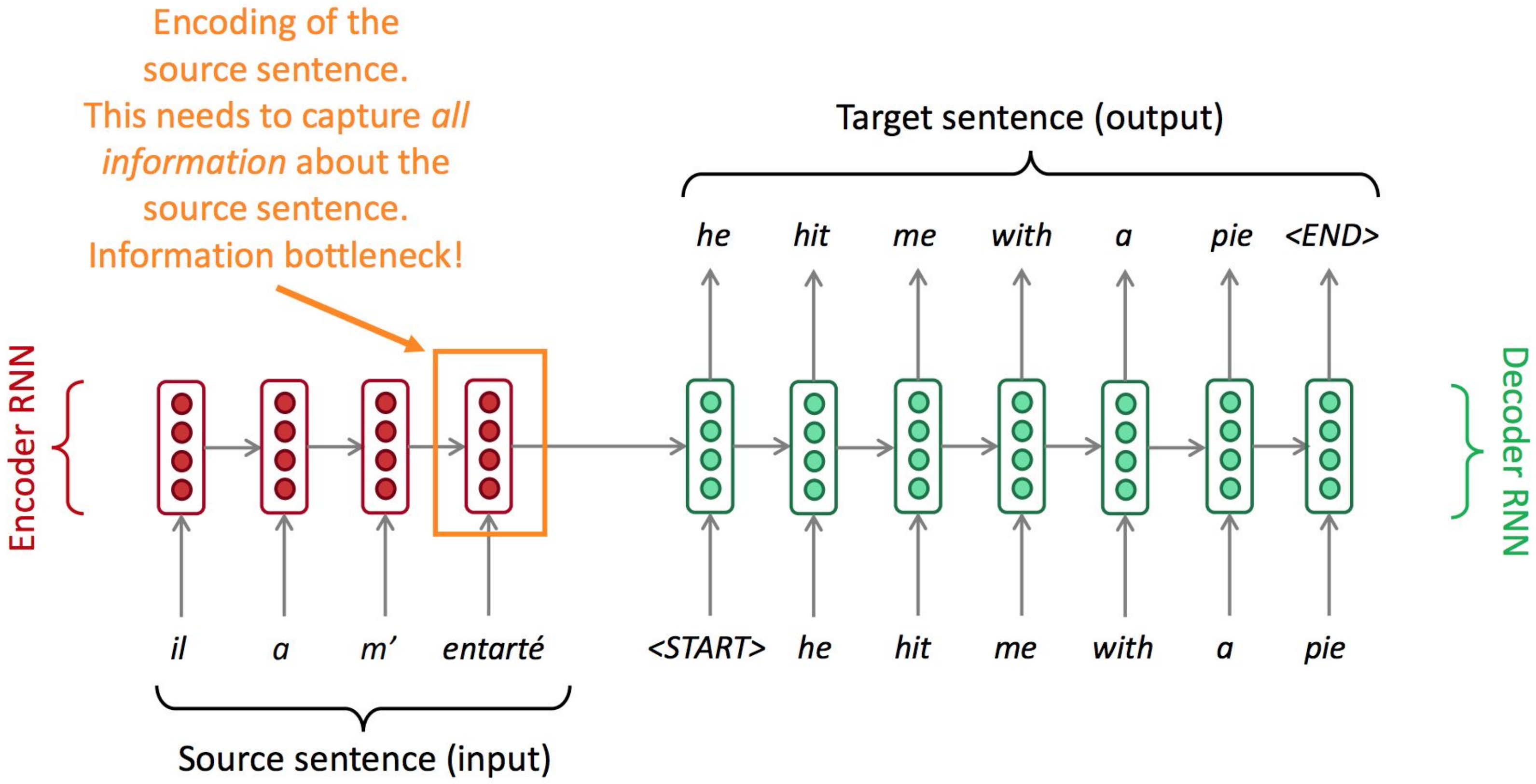
# Outline

1. Seq2seq architecture: recap
2. Attention
   - In pictures
   - In formulas
3. Word Alignments with Attention
4. Self-Attention
5. Q & A

Based on: http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf
https://jalammar.github.io/illustrated-transformer/

# Attention

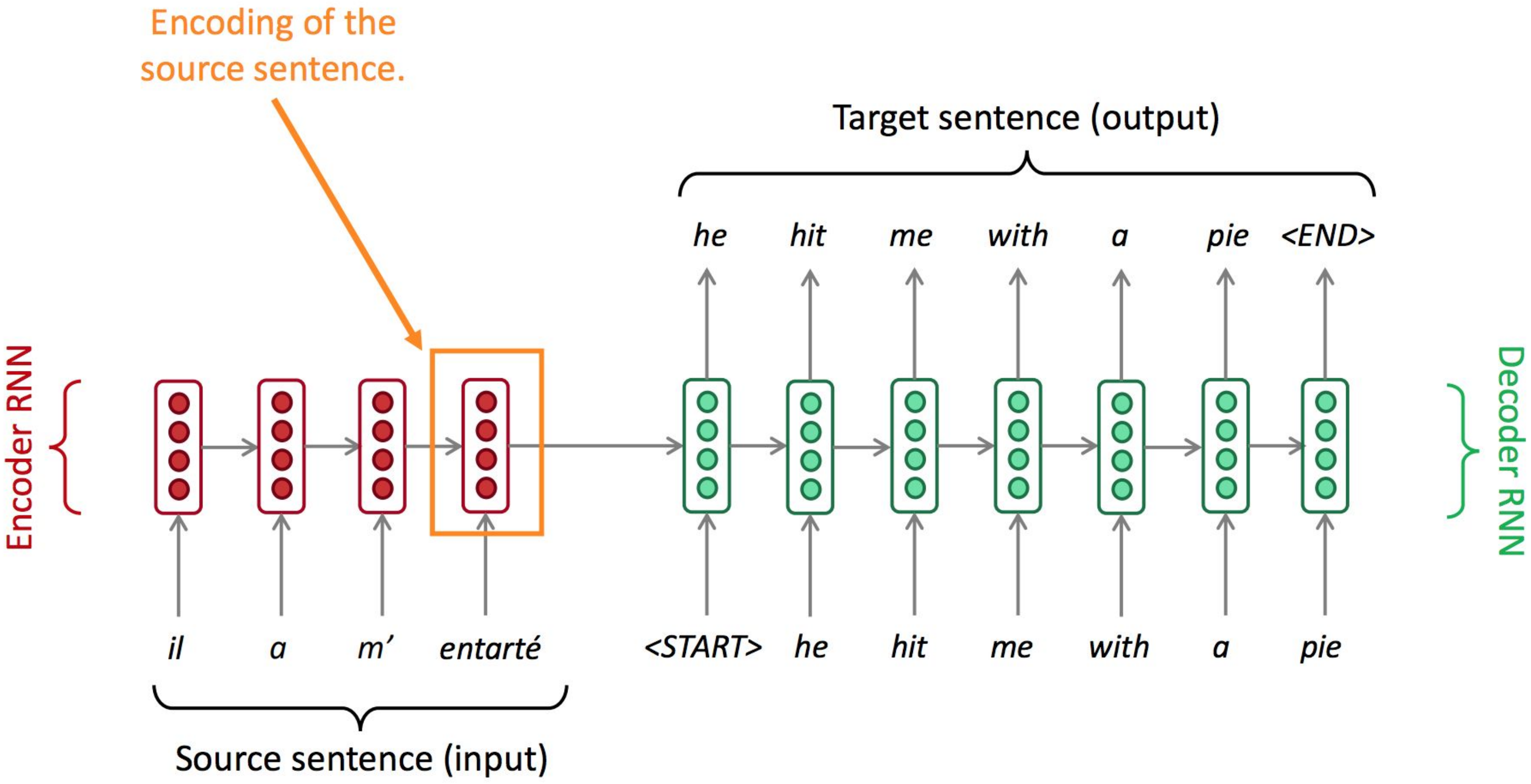
# Seq2seq: the bottleneck problem






http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf

# Seq2seq: the bottleneck problem



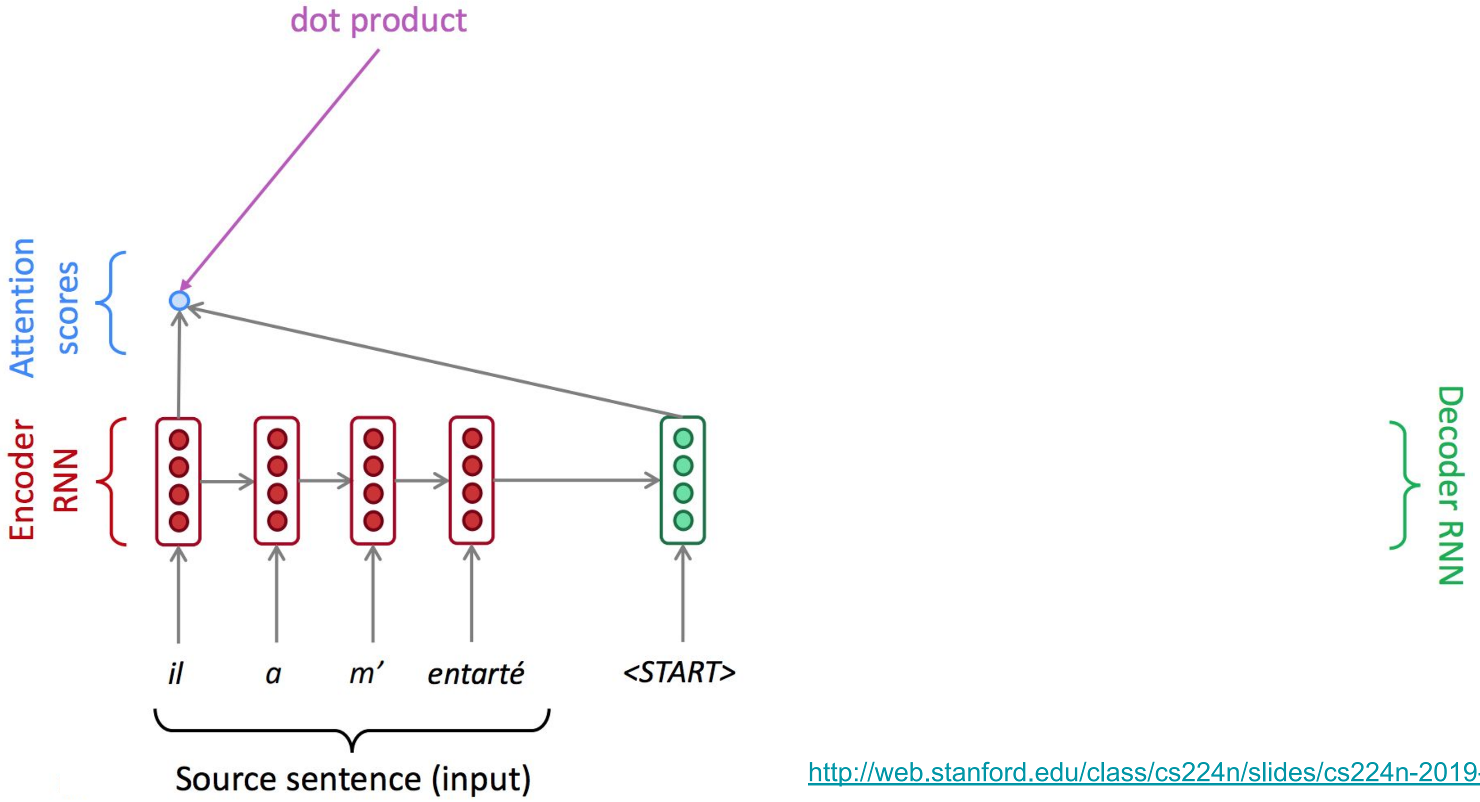

**Problems with this architecture?**

http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf

# Attention

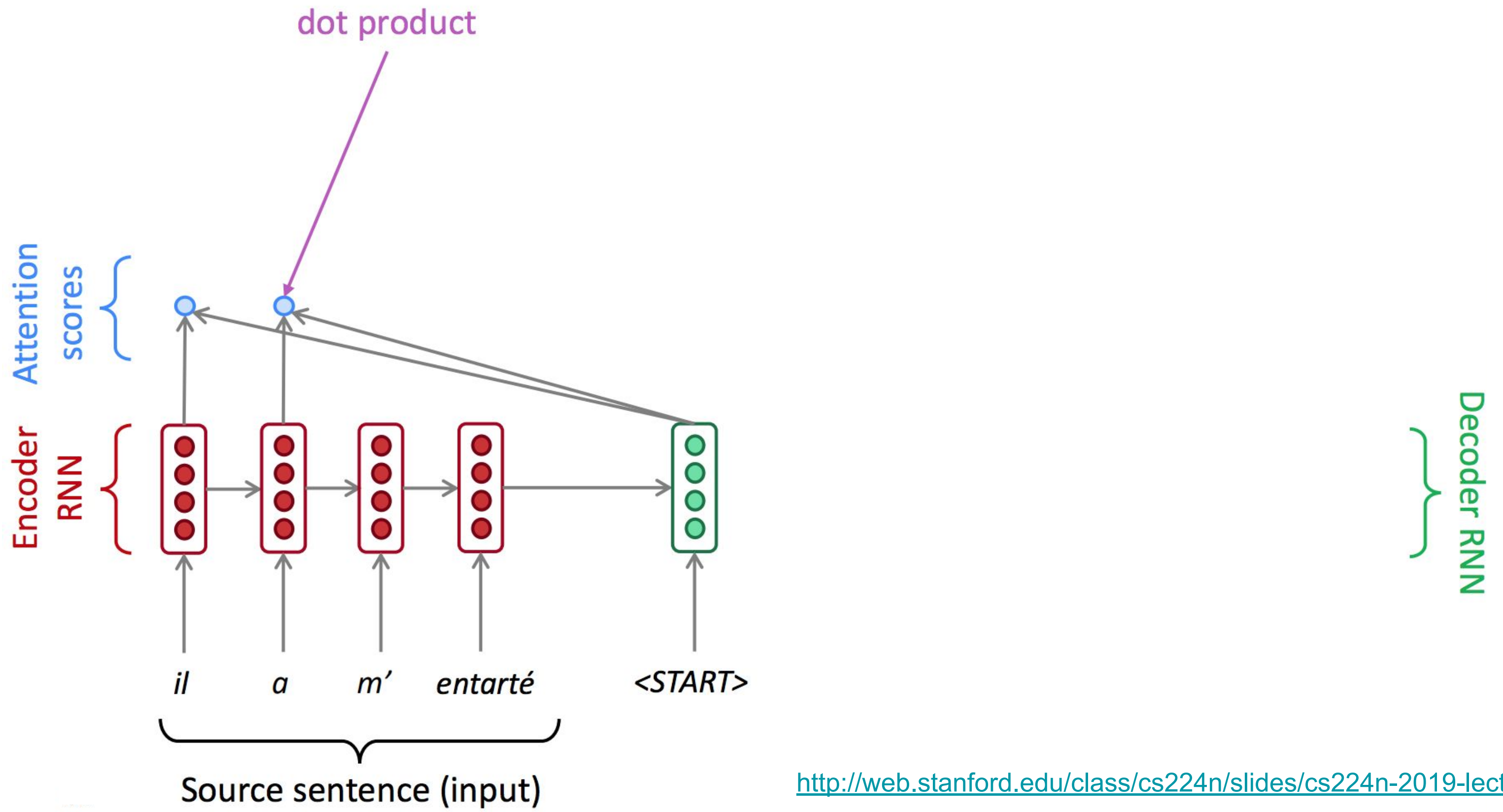
**<u>Main idea:</u>** on each step of the **decoder**, use **direct connection to the encoder** to focus on a particular part of the source sequence

# Seq2seq with attention



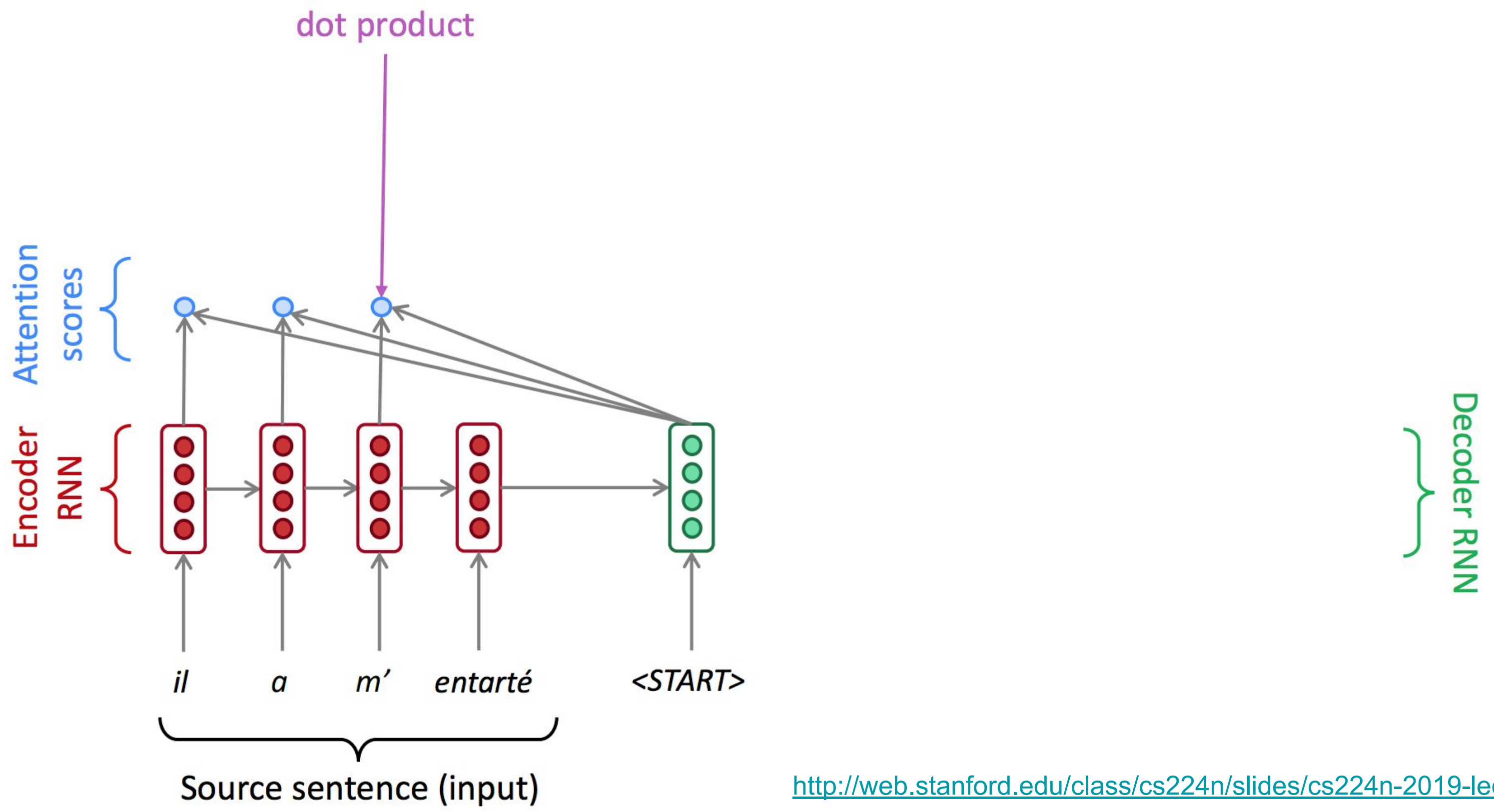

http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf

# Seq2seq with attention



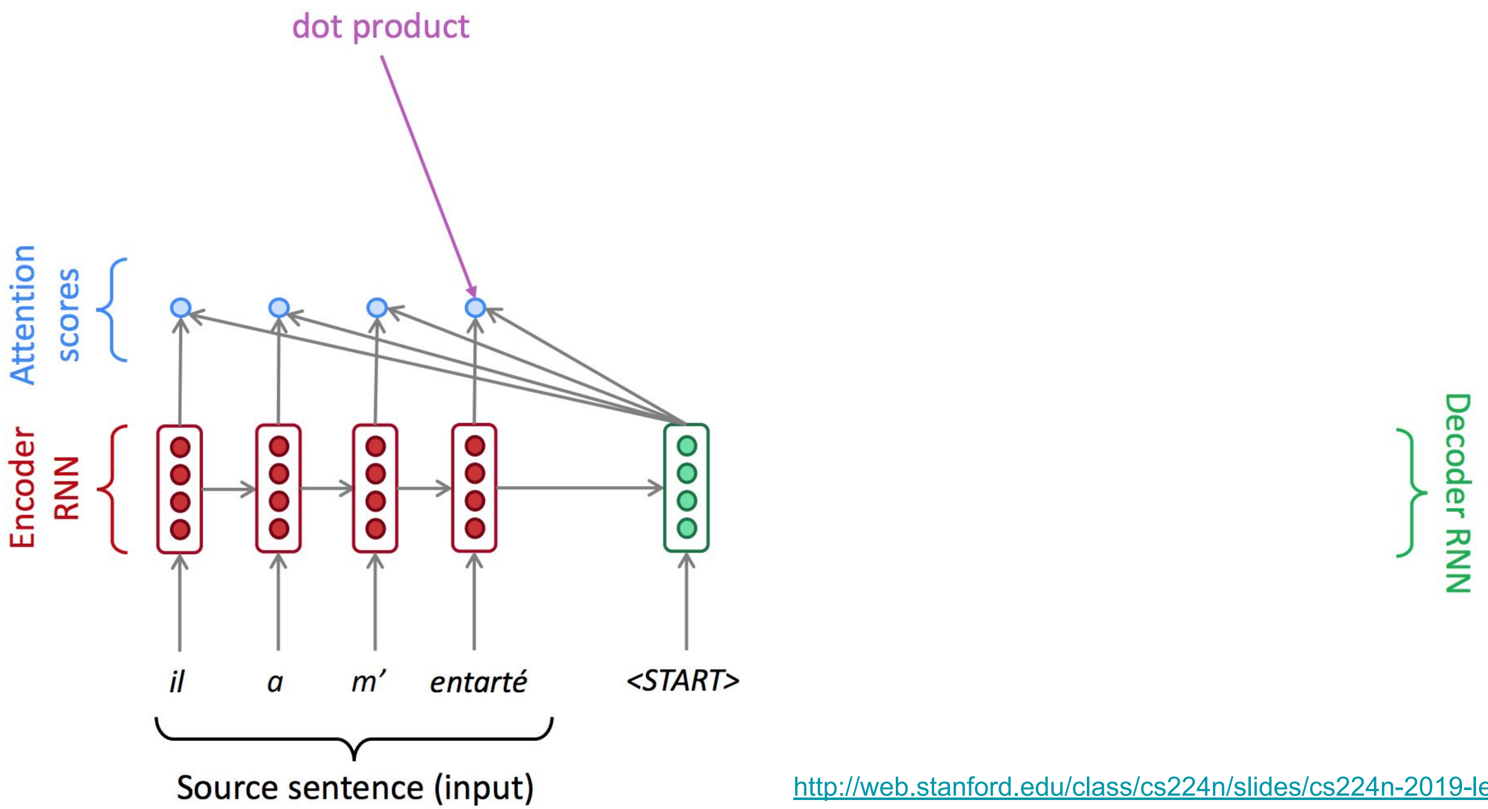

http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf

# Seq2seq with attention



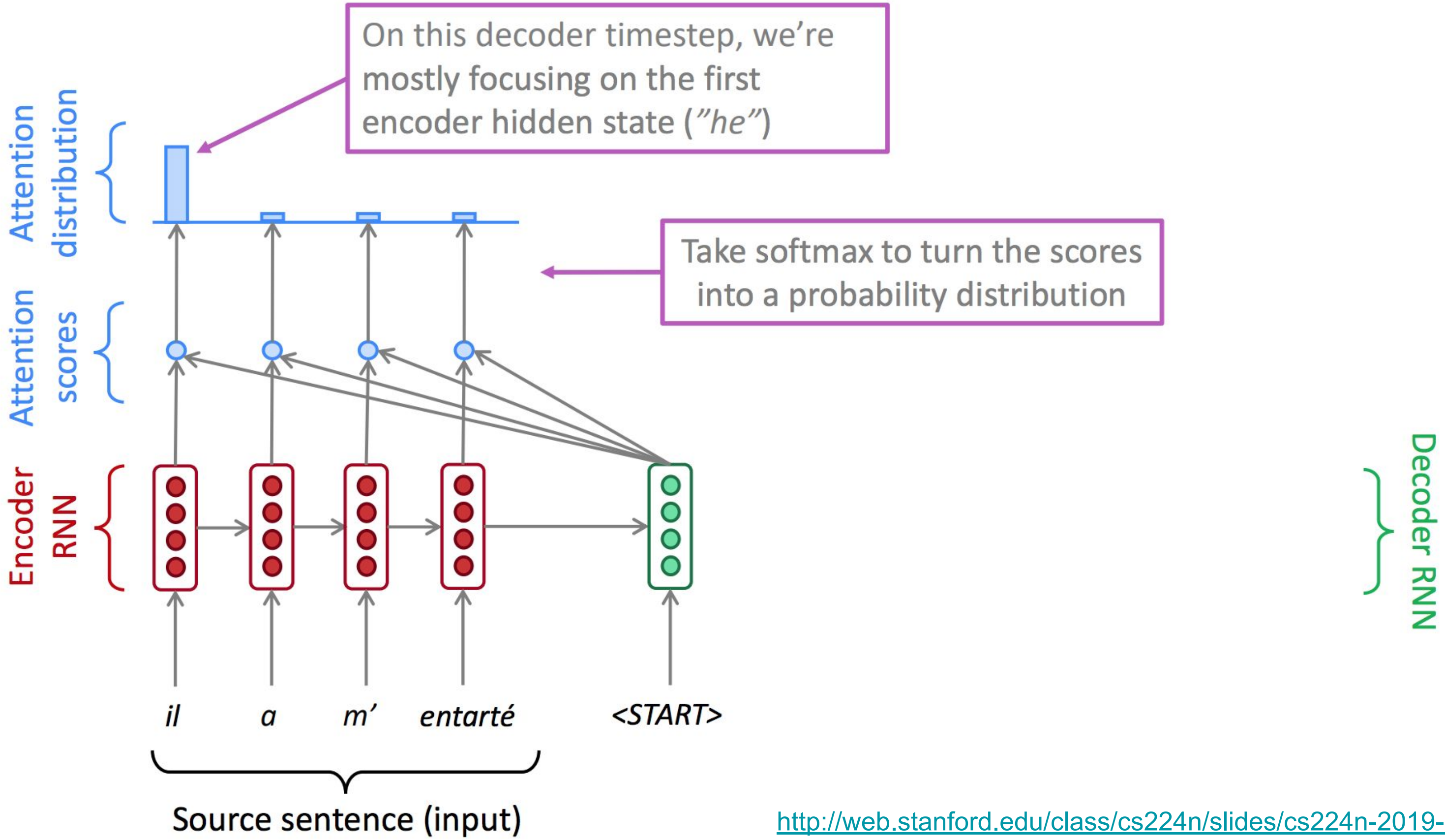

http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf

# Seq2seq with attention



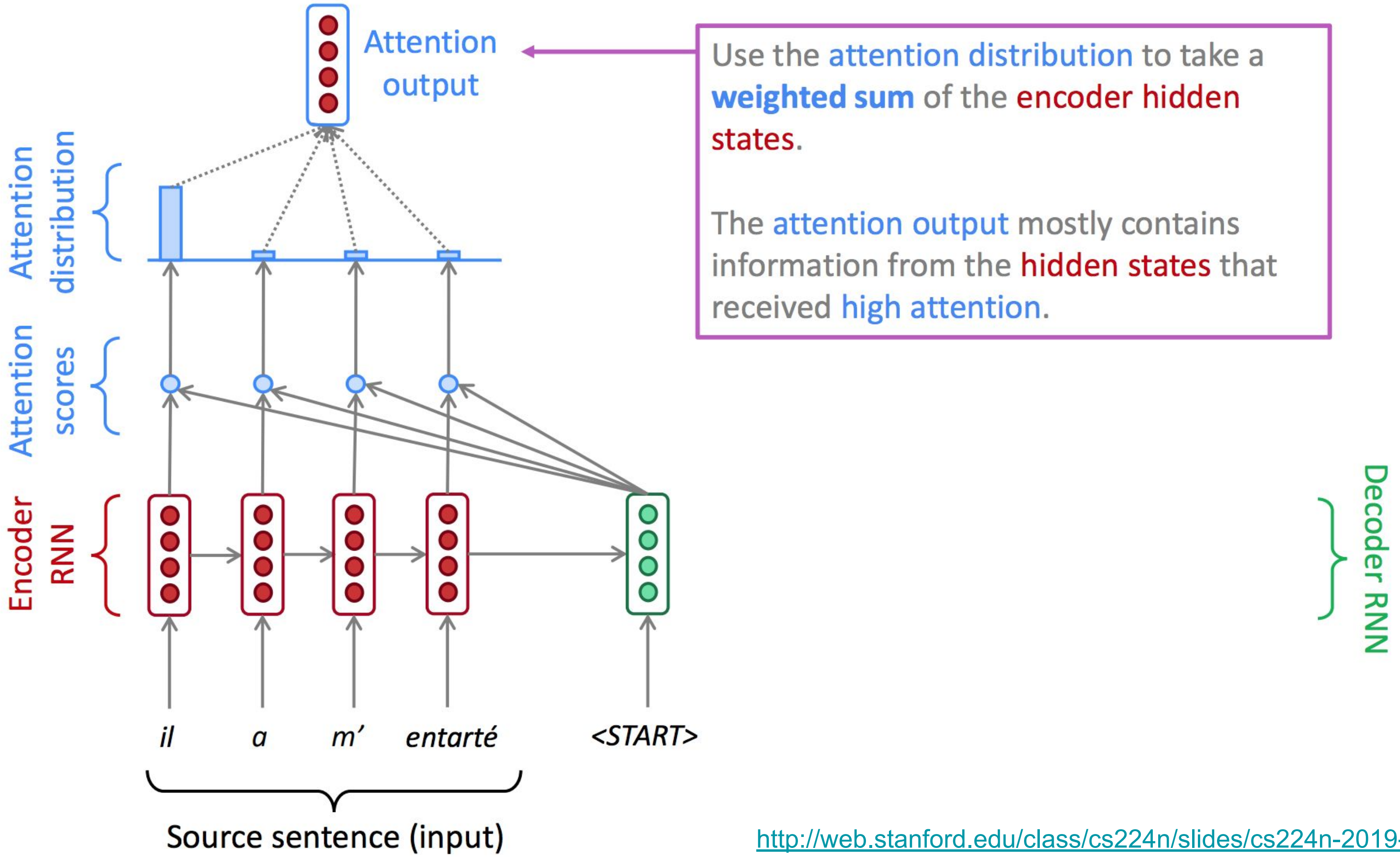

http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf

# Seq2seq with attention




http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf

# Seq2seq with attention

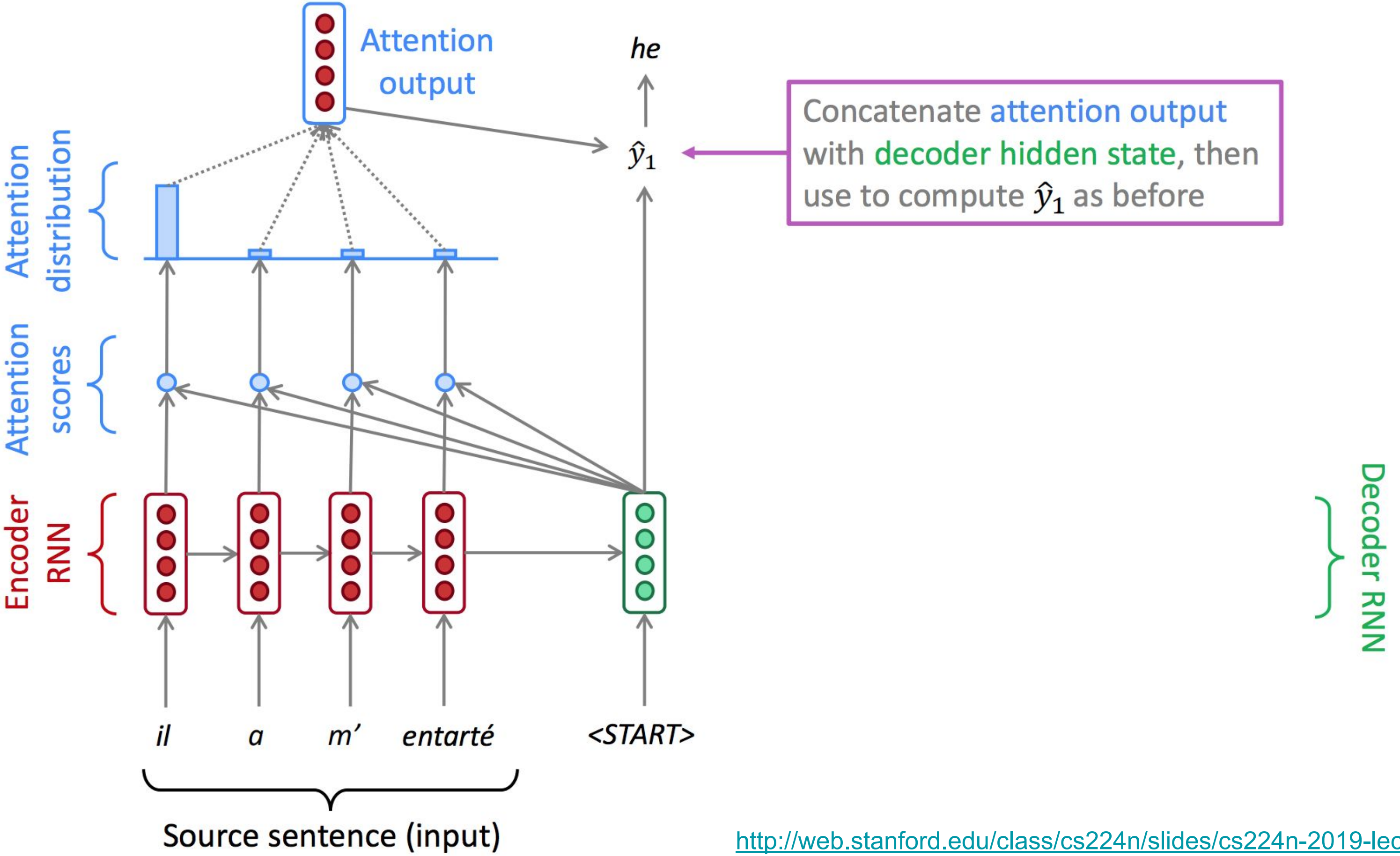



Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

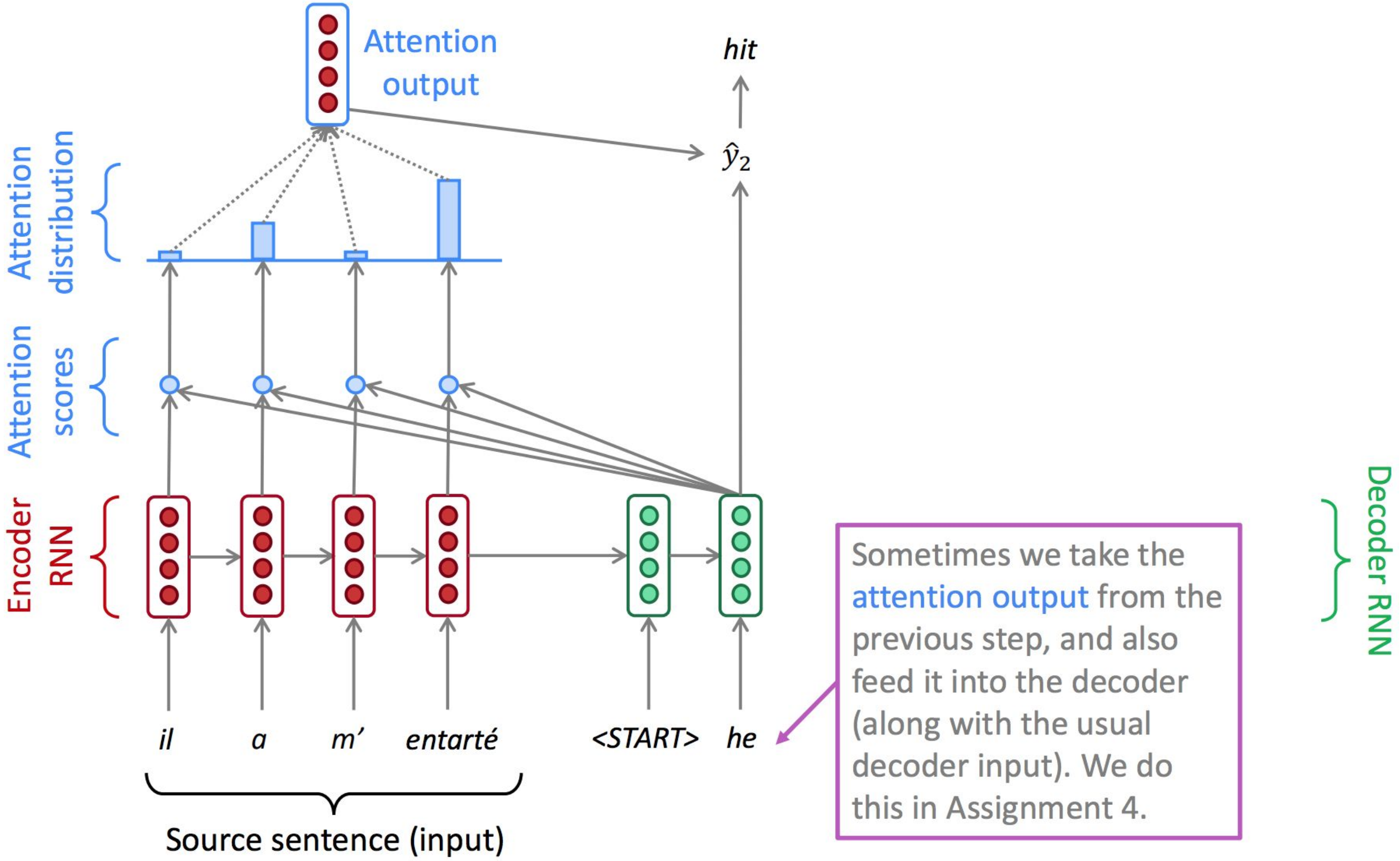
http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf

# Seq2seq with attention



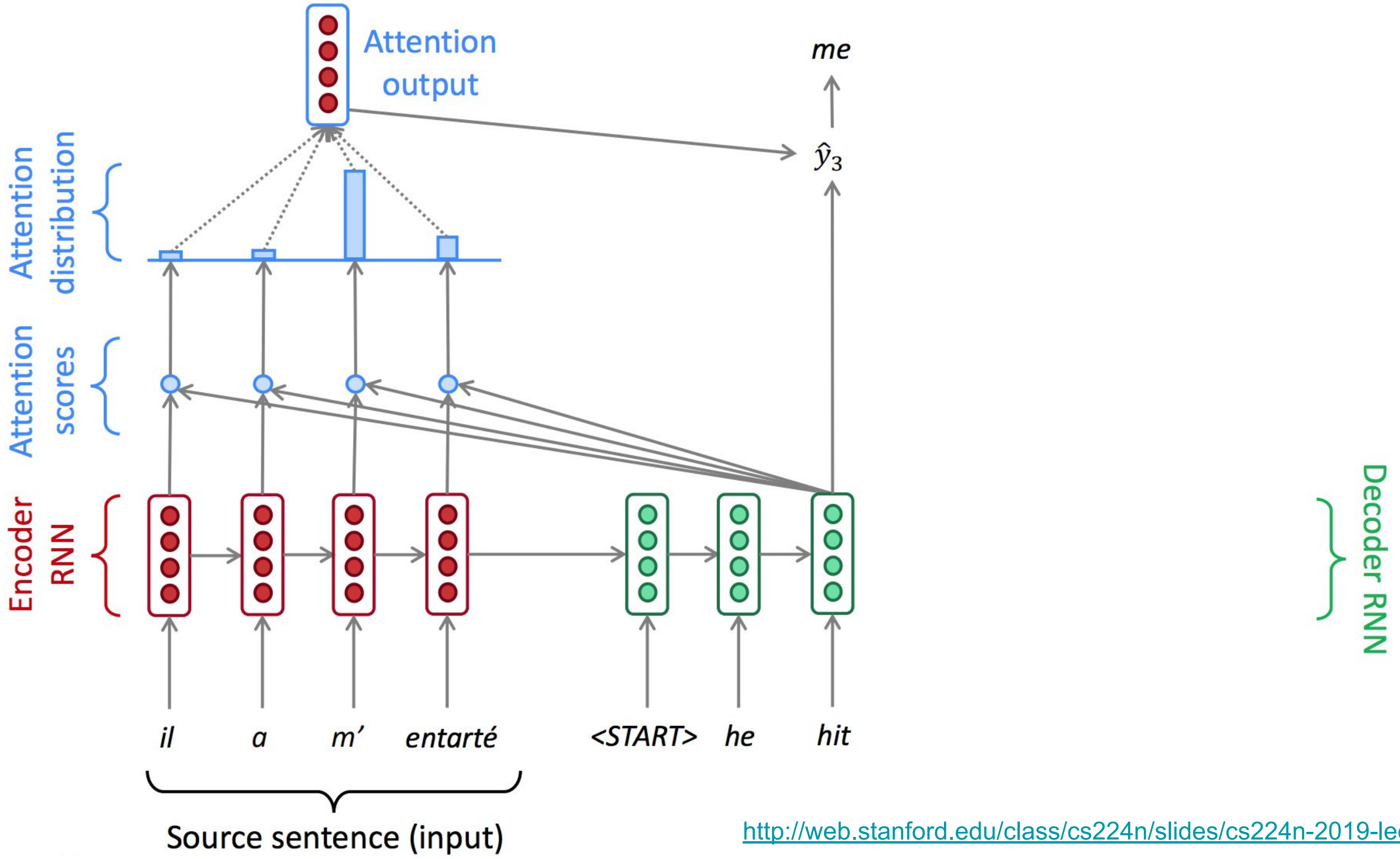

Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

# Seq2seq with attention



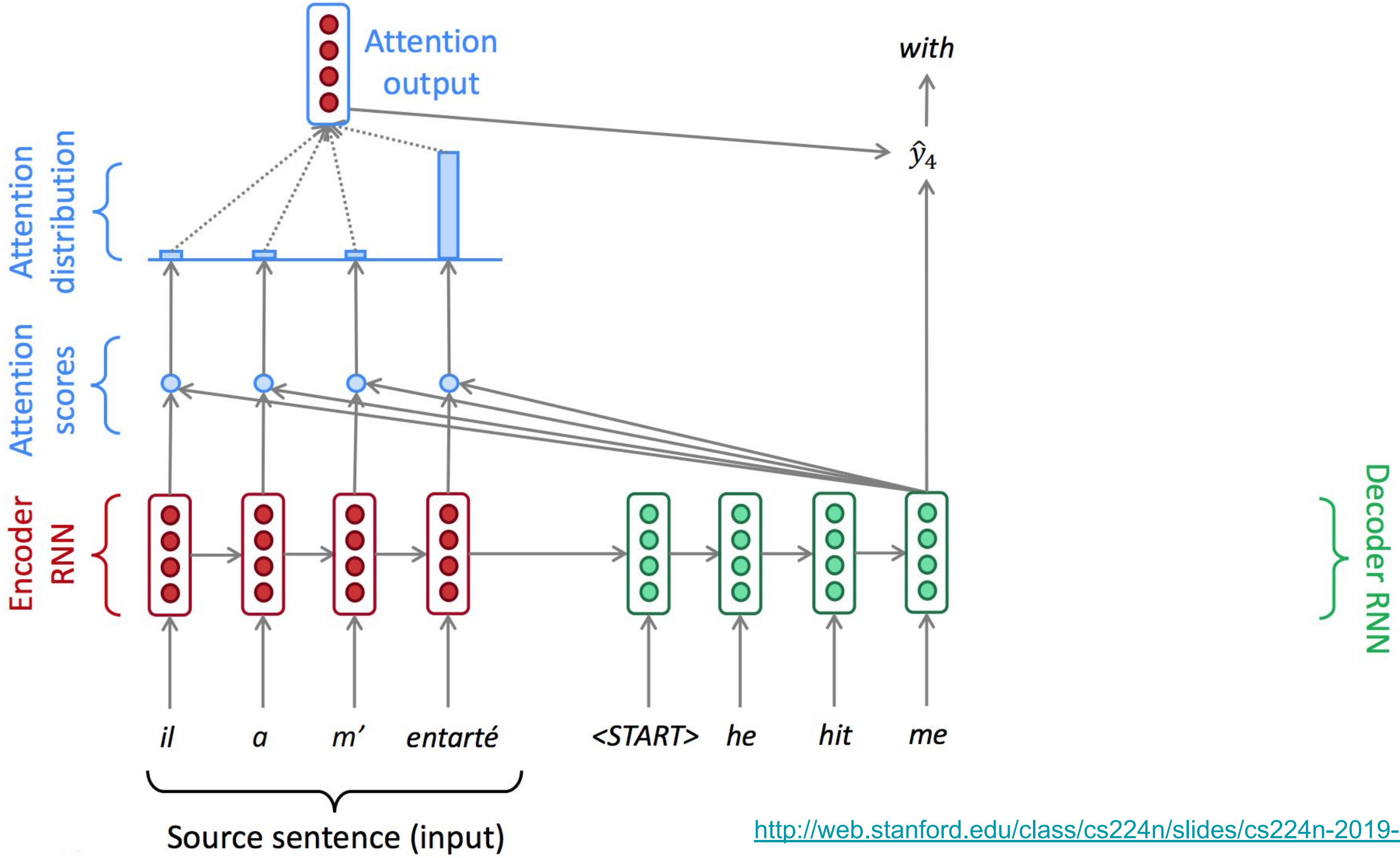

Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input). We do this in Assignment 4.

# Seq2seq with attention



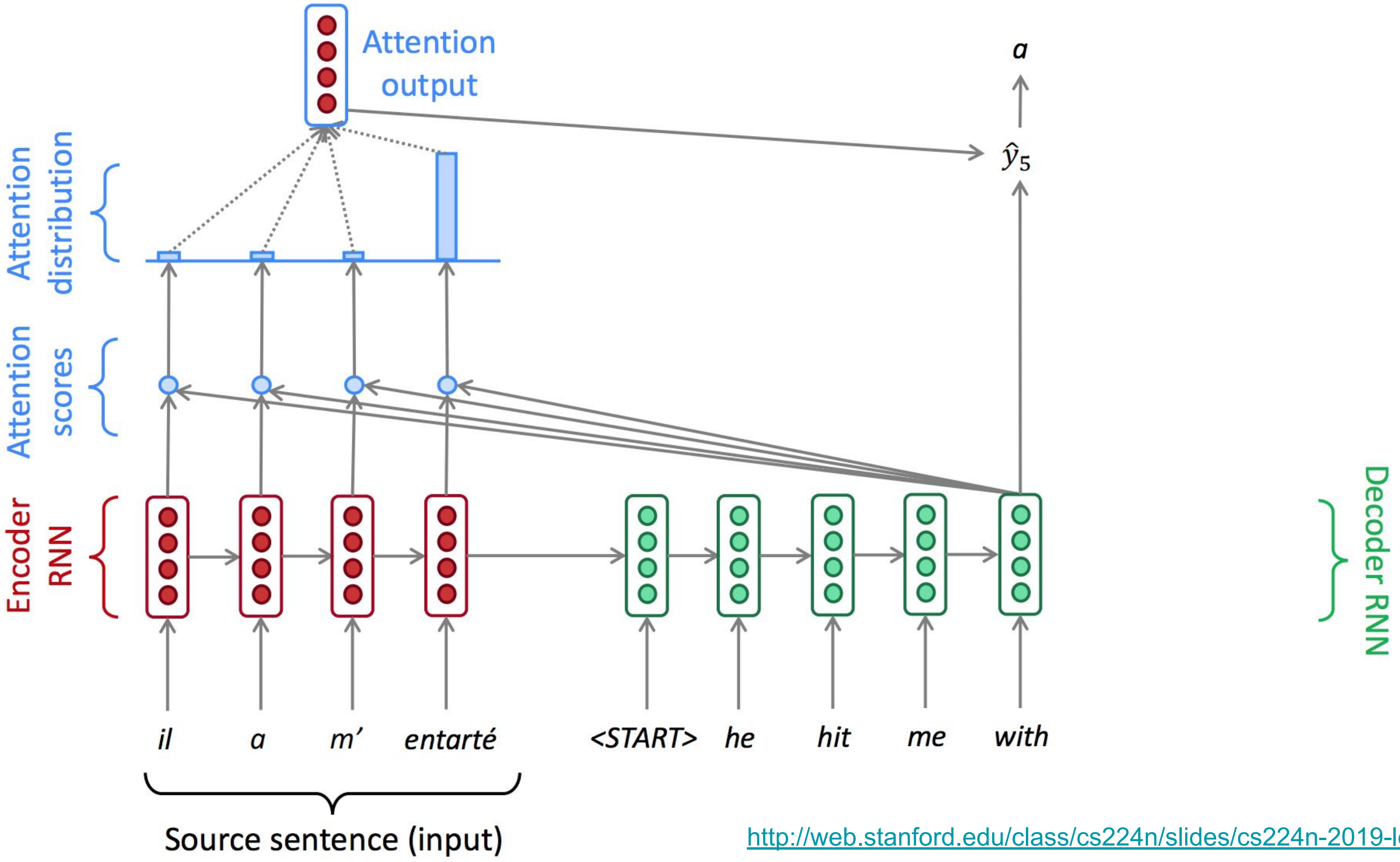

http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf

# Seq2seq with attention



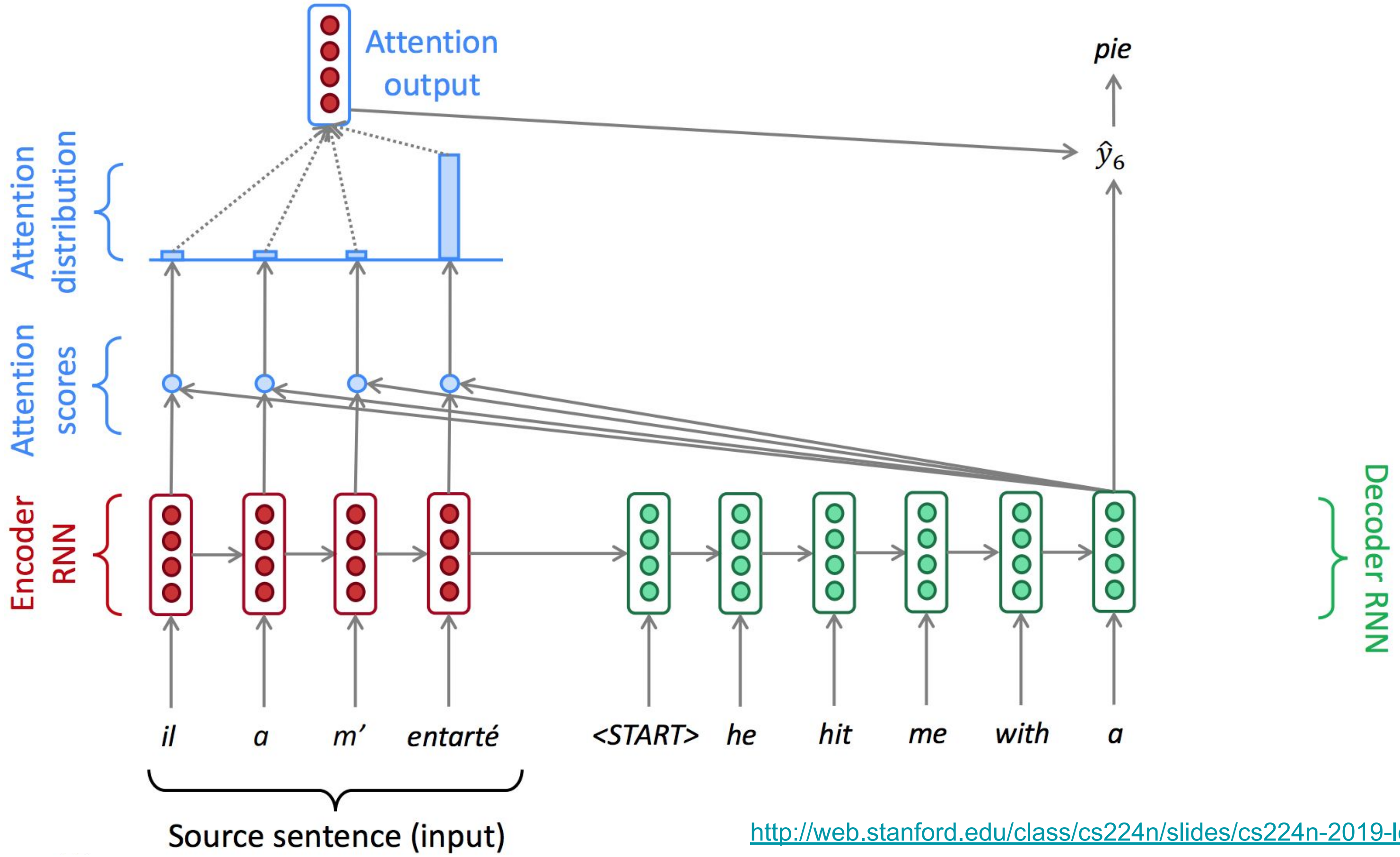

http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf

# Seq2seq with attention




http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf

# Seq2seq with attention




http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf

# Attention in equations

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

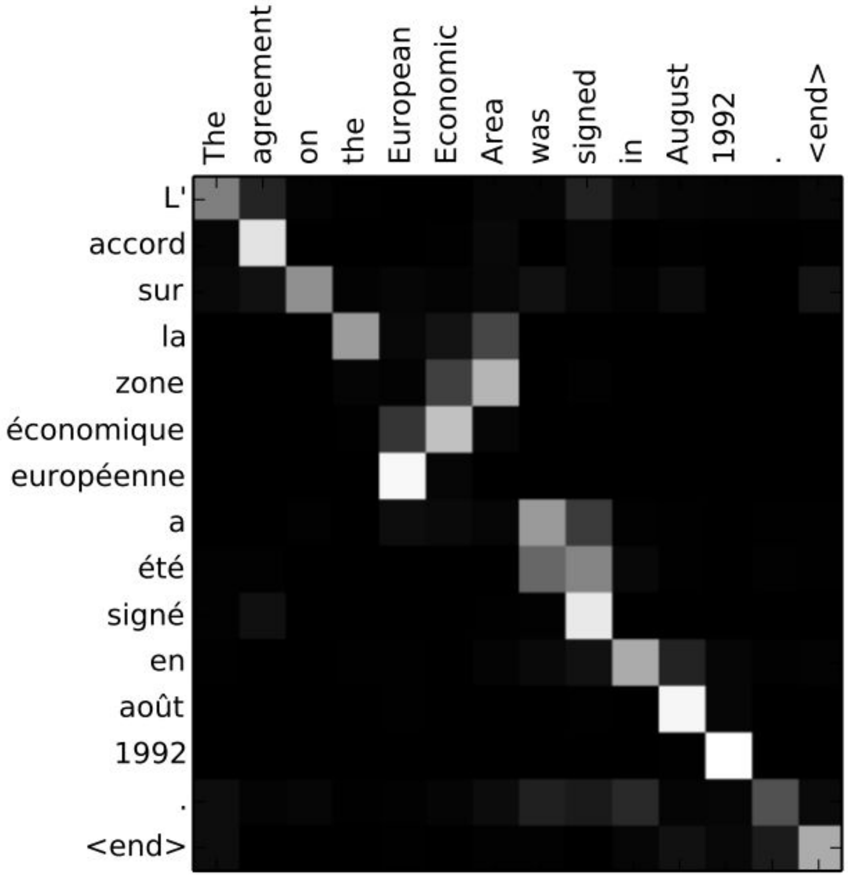
$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

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 provides interpretability

- We may see what the decoder was focusing on
- We get word alignment for free!



# Attention variants

- Basic dot-product (the one discussed before): $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
- Multiplicative attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$
    - $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ - weight matrix
- Additive attention: $\boldsymbol{e}_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}) \in \mathbb{R}$
    - $\boldsymbol{W}_1 \in \mathbb{R}^{d_3 \times d_1}, \boldsymbol{W}_2 \in \mathbb{R}^{d_3 \times d_2}$ - weight matrices
    - $\boldsymbol{v} \in \mathbb{R}^{d_3}$ - weight vector

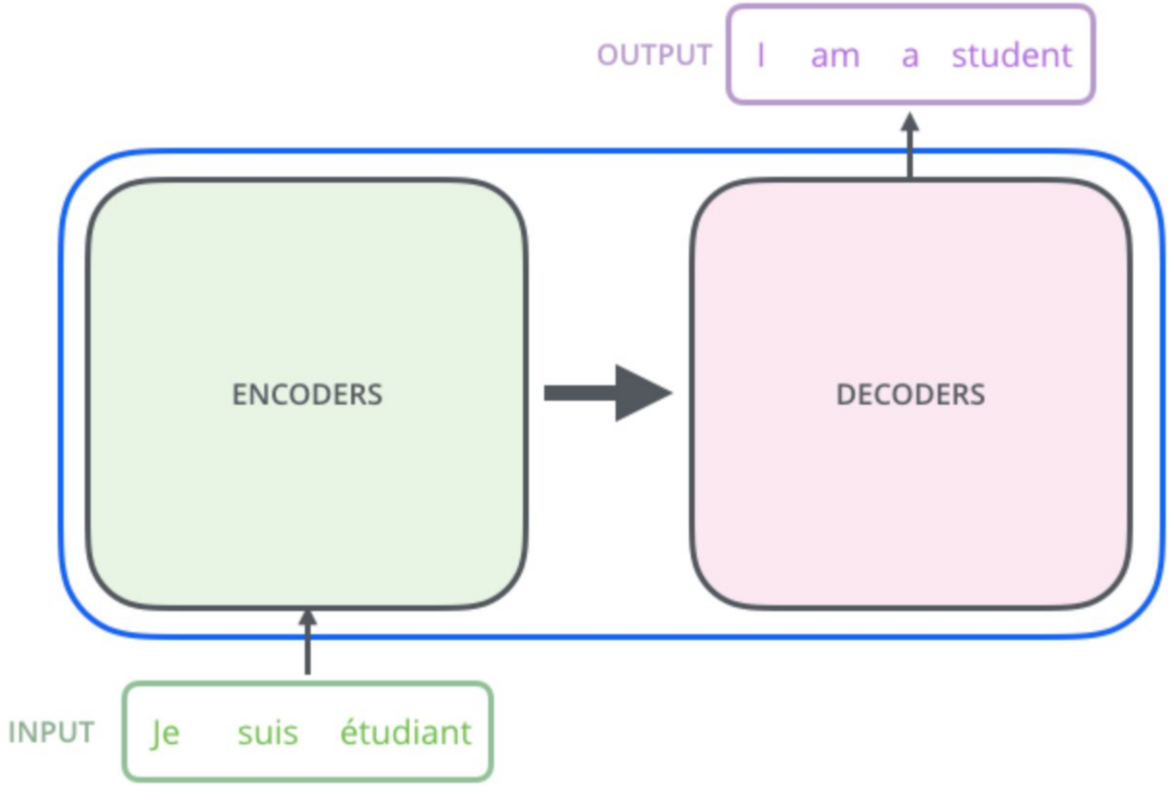
# The Transformer

# Self-Attention: when it has started?



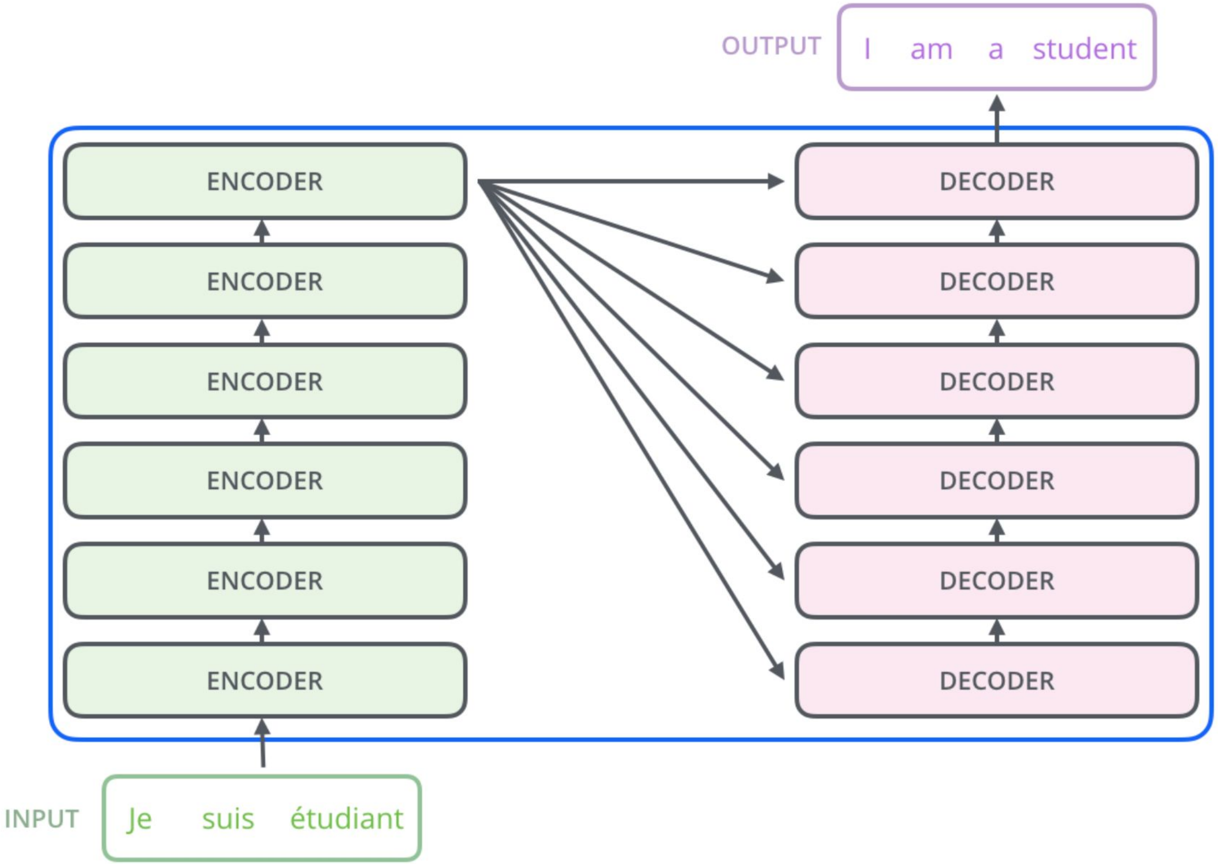

https://jalammar.github.io/illustrated-transformer/

# The Transformer



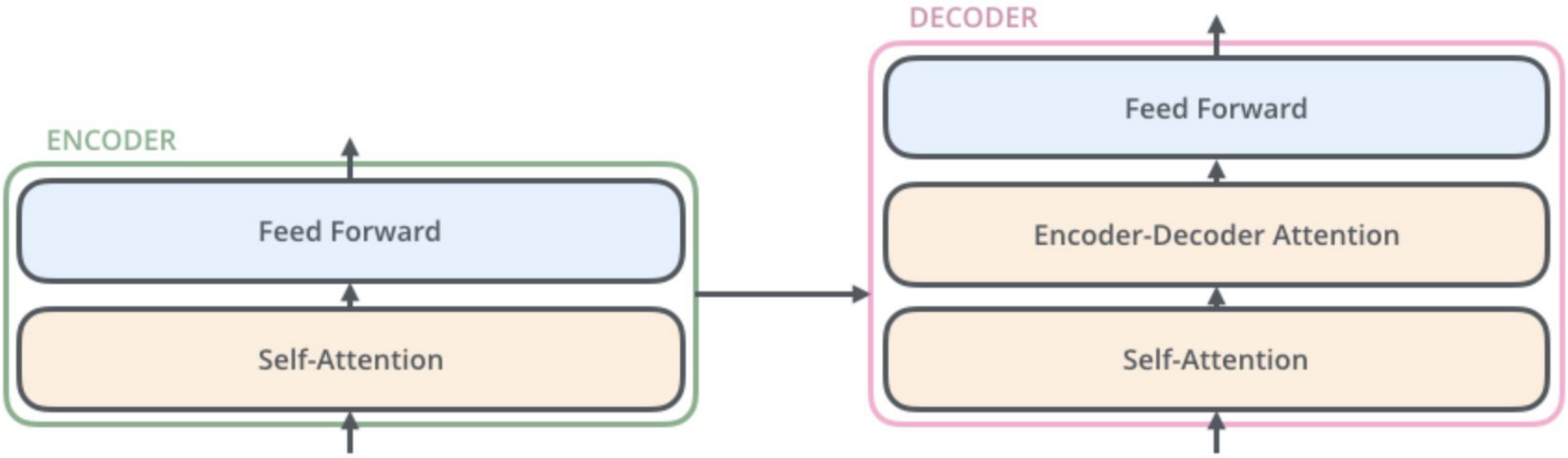

https://jalammar.github.io/illustrated-transformer/

# The Transformer




https://jalammar.github.io/illustrated-transformer/

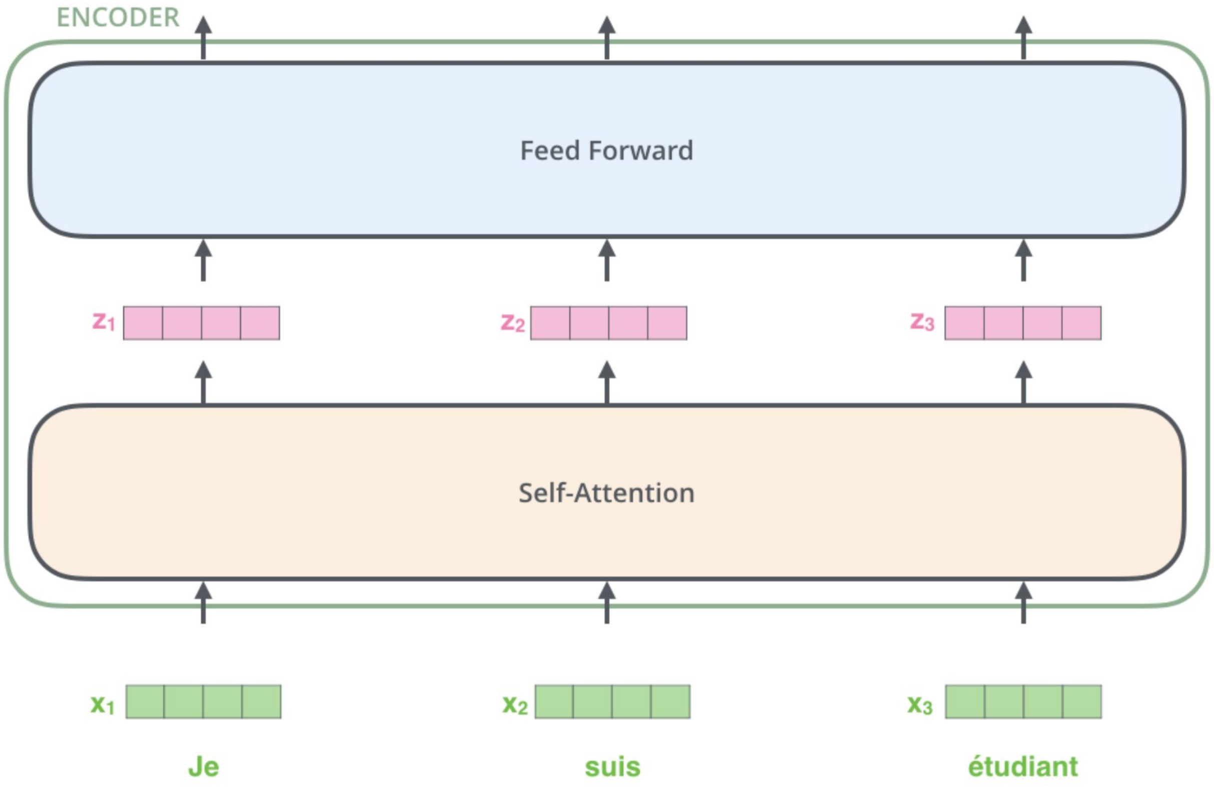
# The Transformer





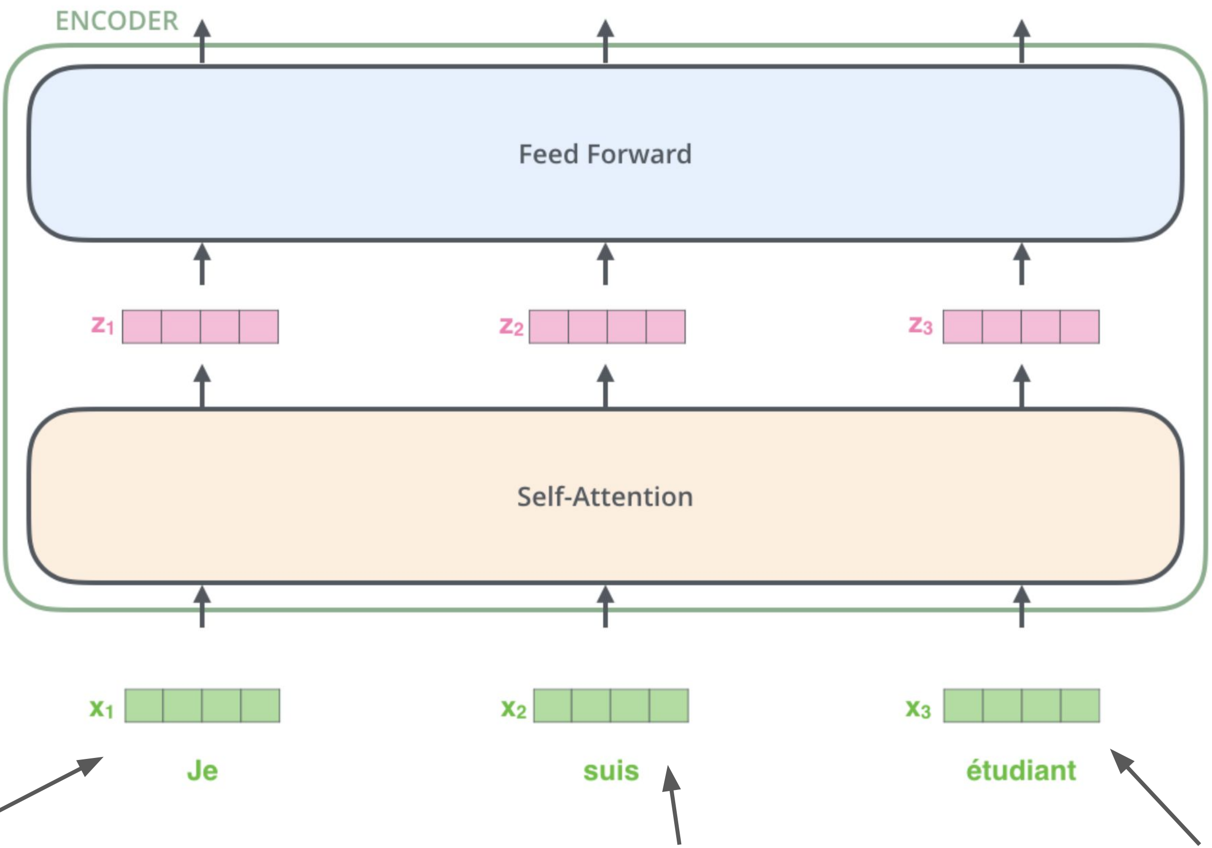

https://jalammar.github.io/illustrated-transformer/

# The Transformer




https://jalammar.github.io/illustrated-transformer/

# The Transformer

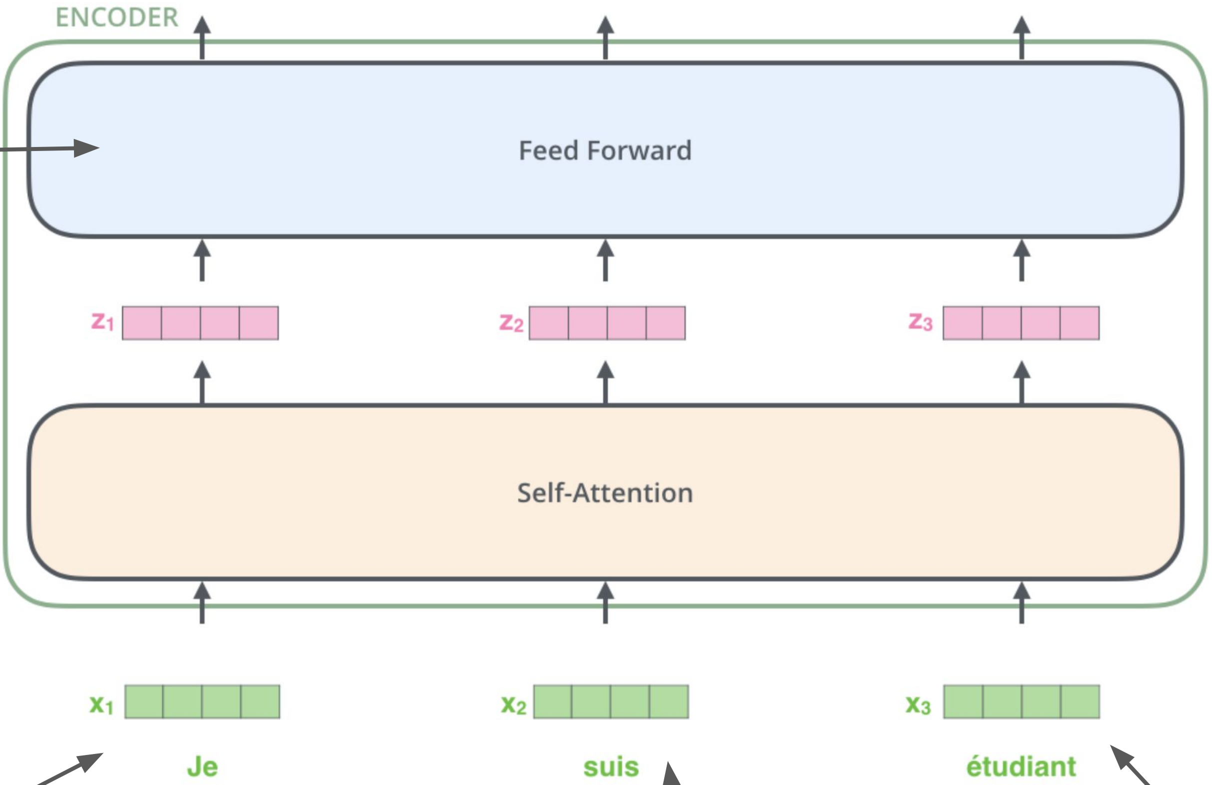



the word in each position flows through its own path in the encoder

https://jalammar.github.io/illustrated-transformer/

# The Transformer

Can be parallelized




the word in each position flows through its own path in the encoder

https://jalammar.github.io/illustrated-transformer/

# The Transformer: quick overview

- Proposed in the paper “Attention is All You Need” (Ashish Vaswani et al.)
- No recurrent or convolutional neural networks -> just attention
- Beats seq2seq in machine translation task
    - 28.4 BLEU on the WMT 2014 English-to-German translation task
- Much faster
- Uses **<u>self-attention</u>** concept

# Self-Attention

# Self-Attention at a High Level

”`The animal didn't cross the street because it was too tired`”

- What does “it” in this sentence refer to?

# Self-Attention at a High Level

”`The animal didn't cross the street because it was too tired`”

- What does “it” in this sentence refer to?
- We want self-attention to associate “it” with “animal”

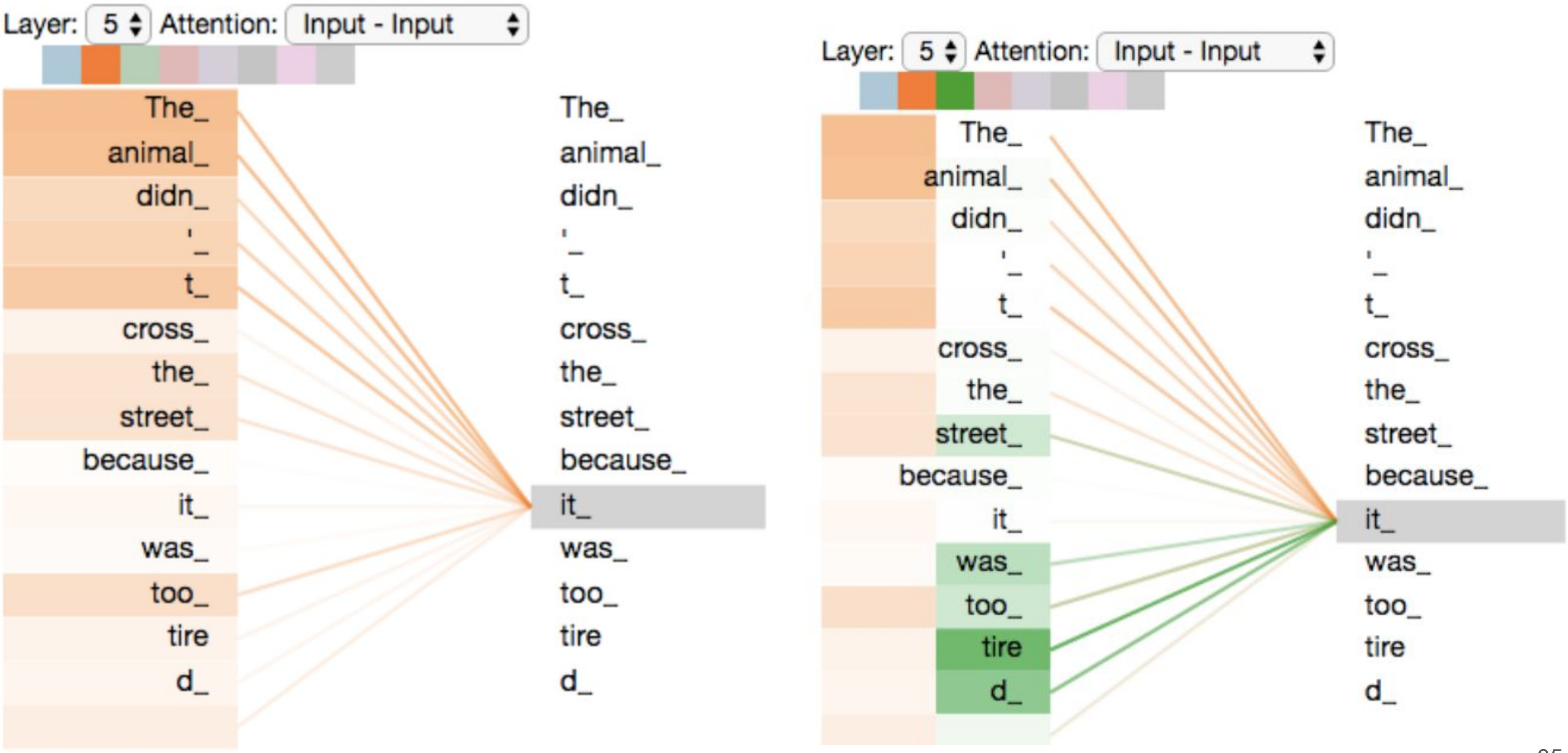
# Self-Attention at a High Level

”`The animal didn't cross the street because it was too tired`”

- What does “it” in this sentence refer to?
- We want self-attention to associate “it” with “animal”

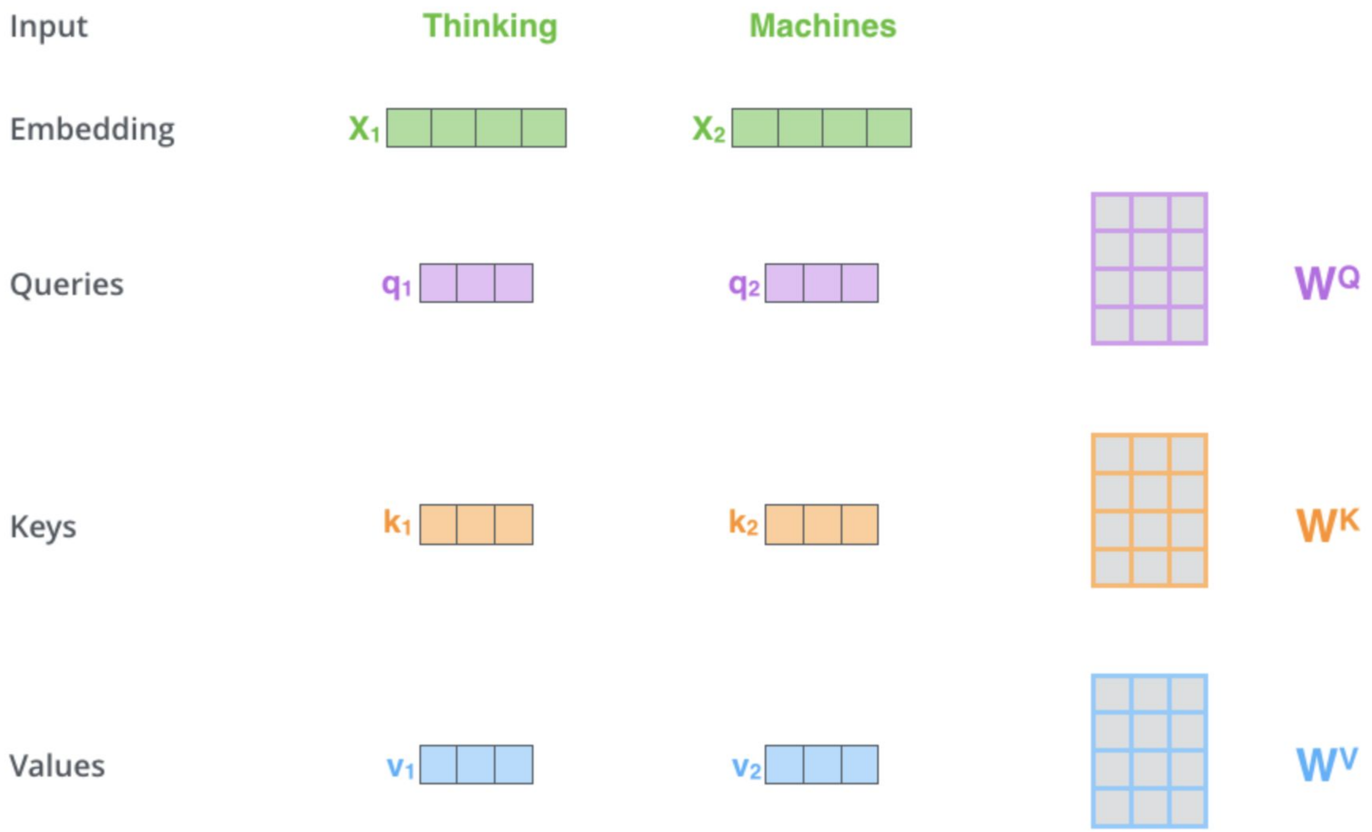
- Self-attention is the method for baking the “understanding” of other relevant words into the one we’re currently processing

# Self-Attention at a High Level




https://jalammar.github.io/illustrated-transformer/

# Self-Attention in Detail



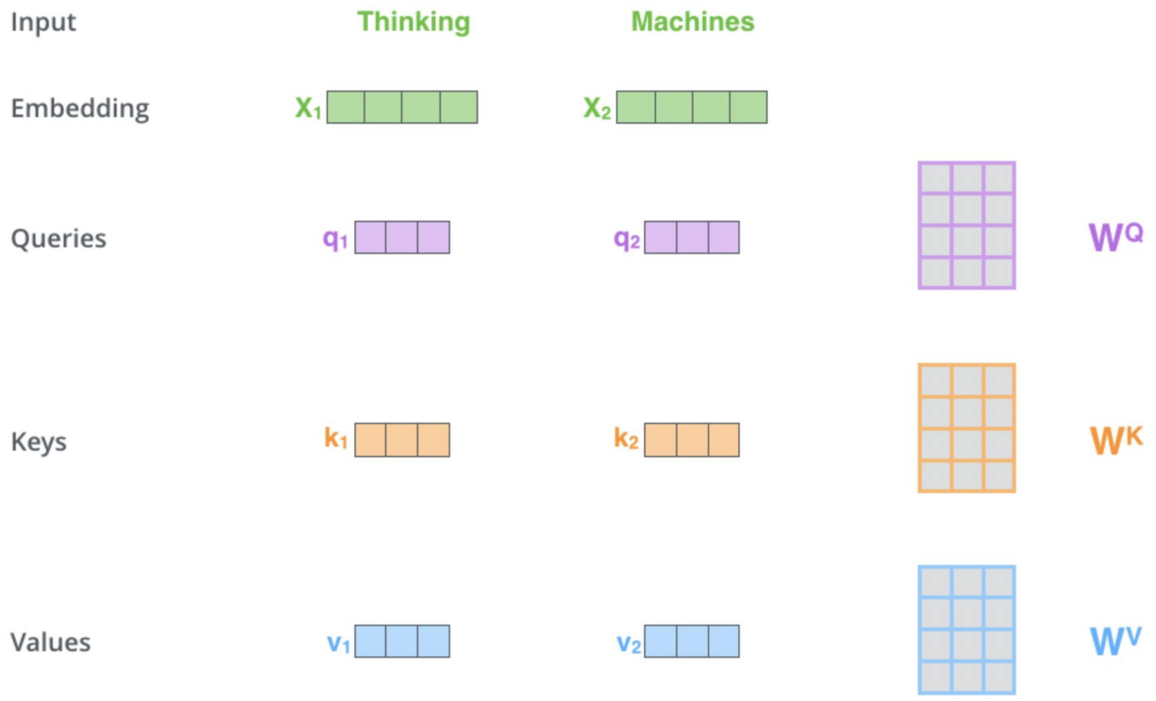

https://jalammar.github.io/illustrated-transformer/

# Self-Attention in Detail

**STEP 1:**

create 3 vectors (Query, Key, Value)

from each of the encoder's input vectors




https://jalammar.github.io/illustrated-transformer/

# Self-Attention in Detail

What are the “query”, “key”, and “value” vectors?

# Self-Attention at in Detail

What are the “query”, “key”, and “value” vectors?

They’re abstractions that are useful for

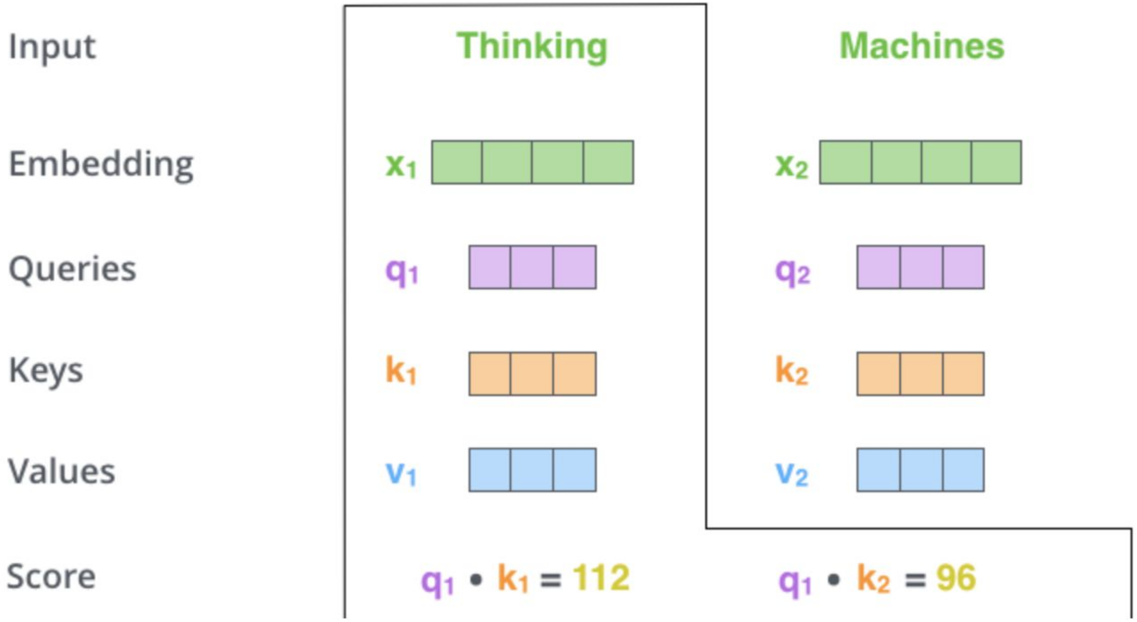
calculating and thinking about attention.

# Self-Attention at in Detail

**STEP 2:**

calculate a score

(score each word of the input sentence against the current word)




https://jalammar.github.io/illustrated-transformer/

# Self-Attention at in Detail

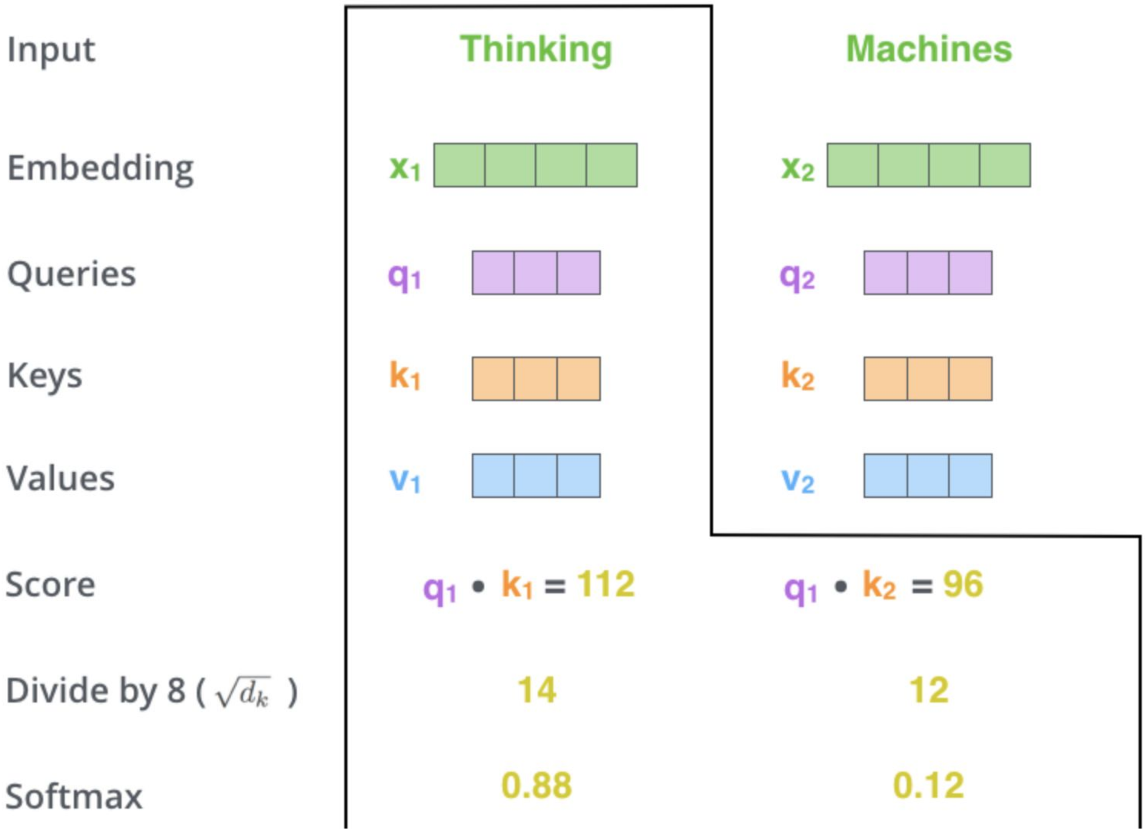
**STEP 3:**

divide the scores by 8

(the square root of the dimension of the key vectors)

**STEP 4:**

softmax




https://jalammar.github.io/illustrated-transformer/

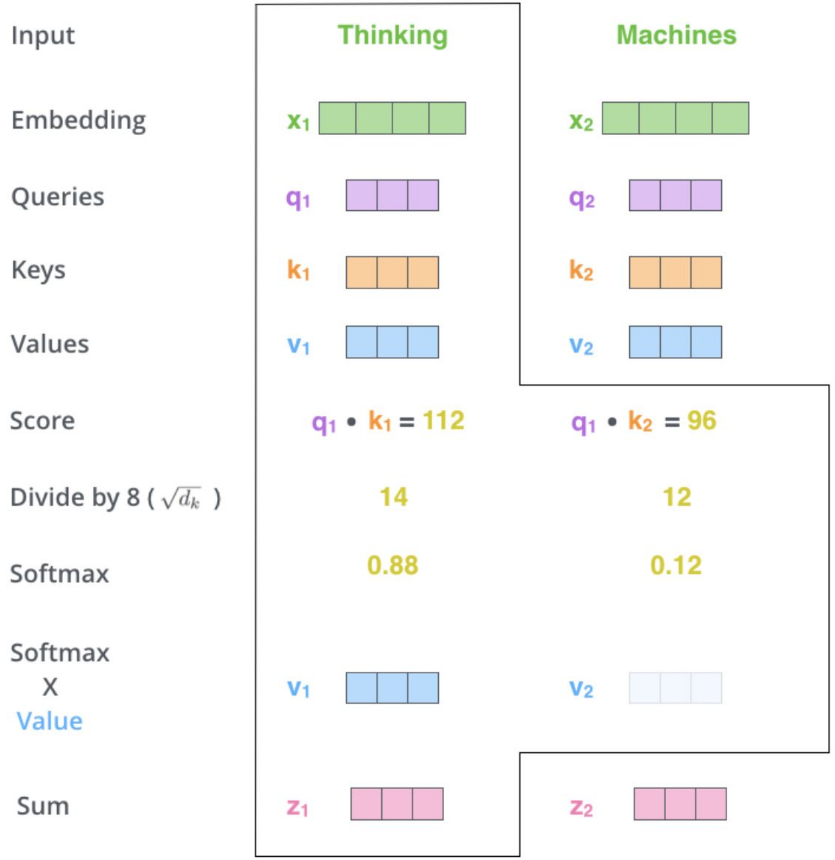
# Self-Attention in Detail

**STEP 5:**

multiply each value vector by the softmax score

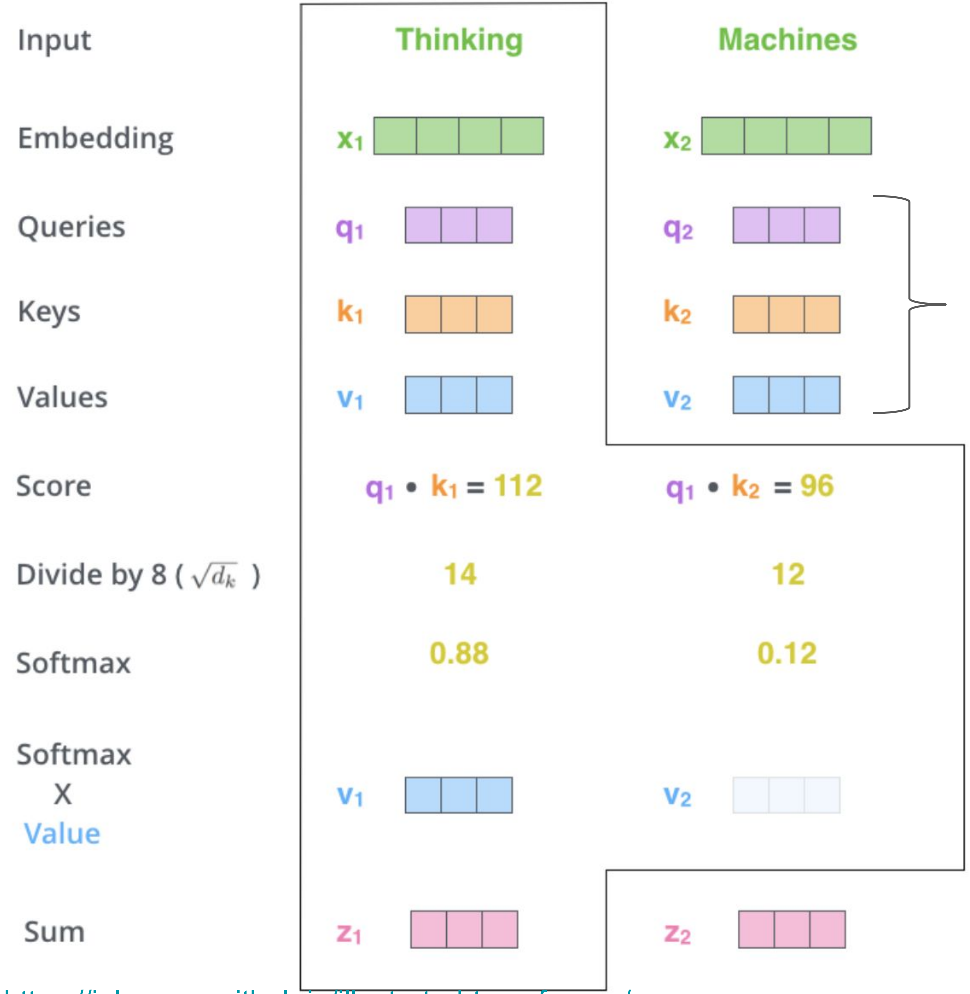
**STEP 6:**

sum up the weighted value vectors




https://jalammar.github.io/illustrated-transformer/

# Self-Attention in Detail




**STEP 1:** create Query, Key, Value

**STEP 2:** calculate scores

**STEP 3:** divide by $\sqrt{d_k}$

**STEP 4:** softmax

**STEP 5:** multiply each value vector by the softmax score

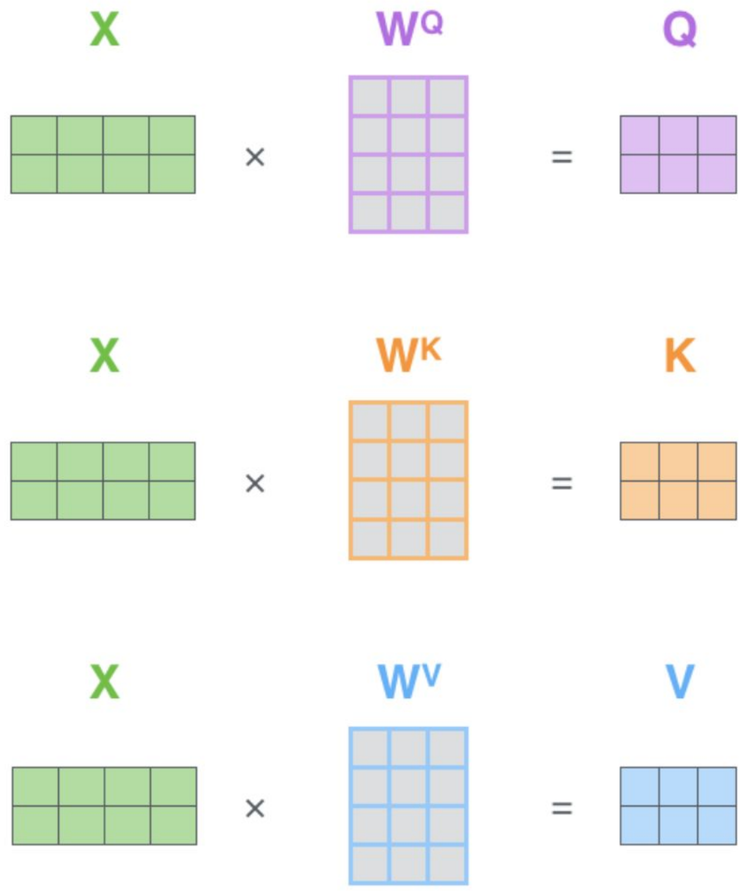
**STEP 6:** sum up the weighted value vectors

https://jalammar.github.io/illustrated-transformer/

# Self-Attention: Matrix Calculation

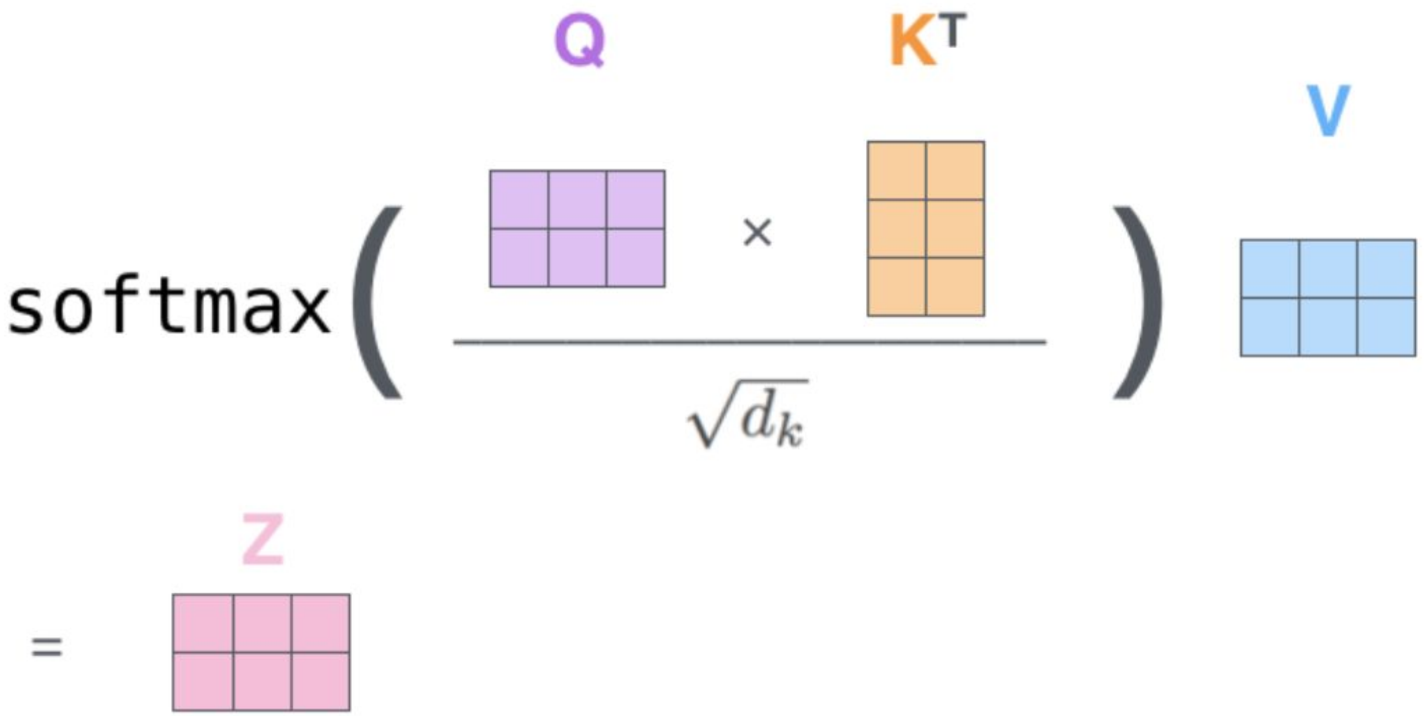
Pack embeddings into matrix **X**

Multiply **X** by weight matrices we've trained (**Wk, Wq, Wv**)



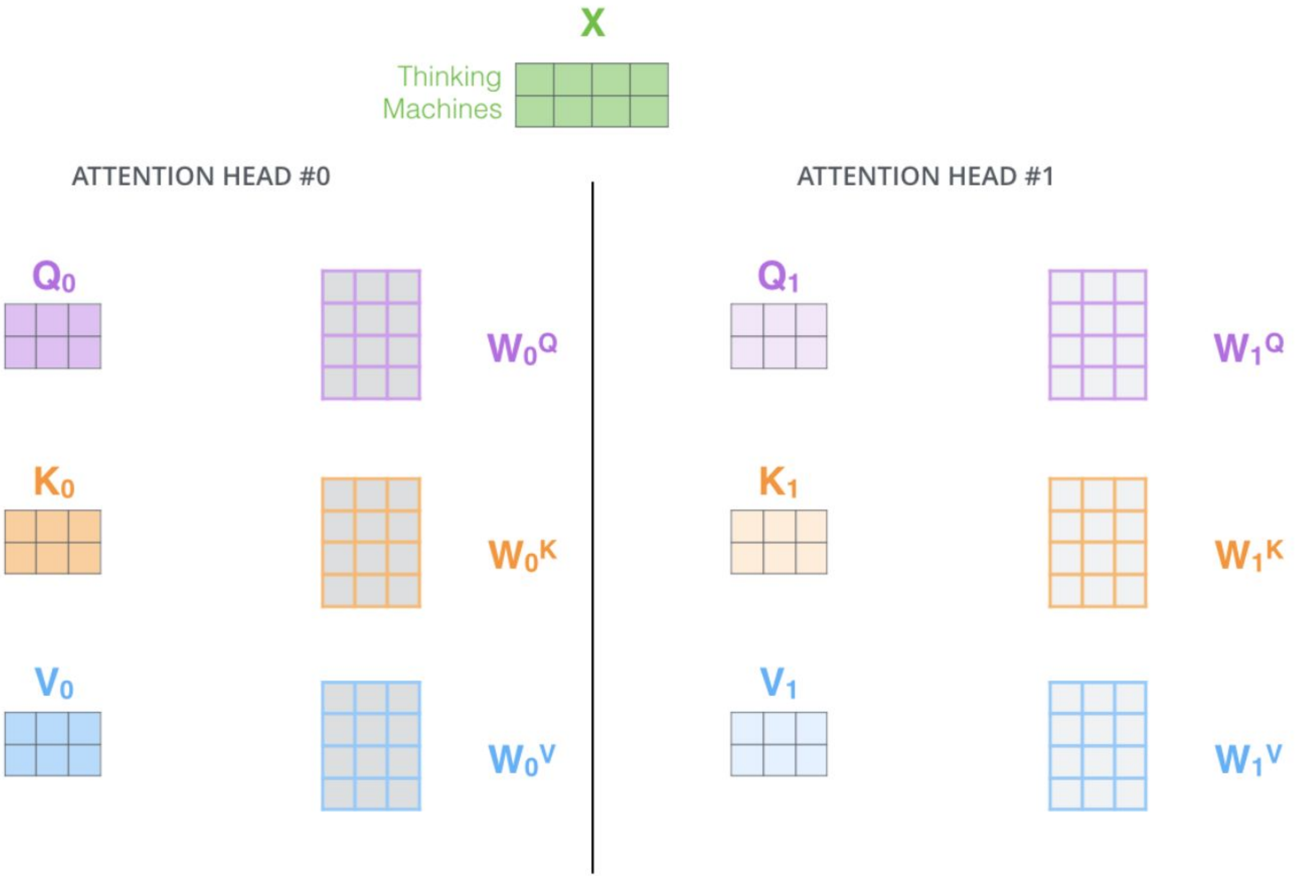

https://jalammar.github.io/illustrated-transformer/

# Self-Attention: Matrix Calculation



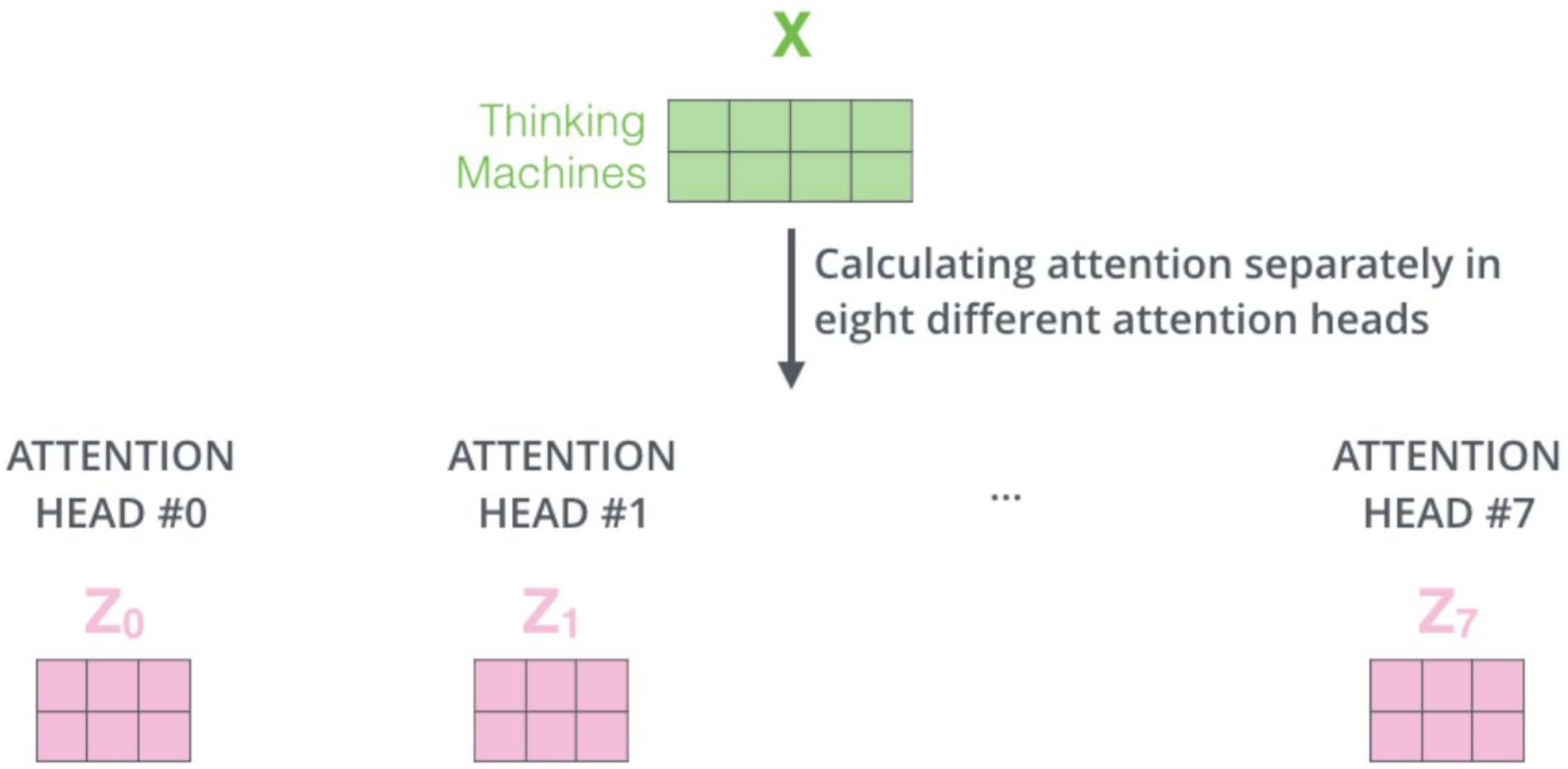

https://jalammar.github.io/illustrated-transformer/

# Multi-Head Attention



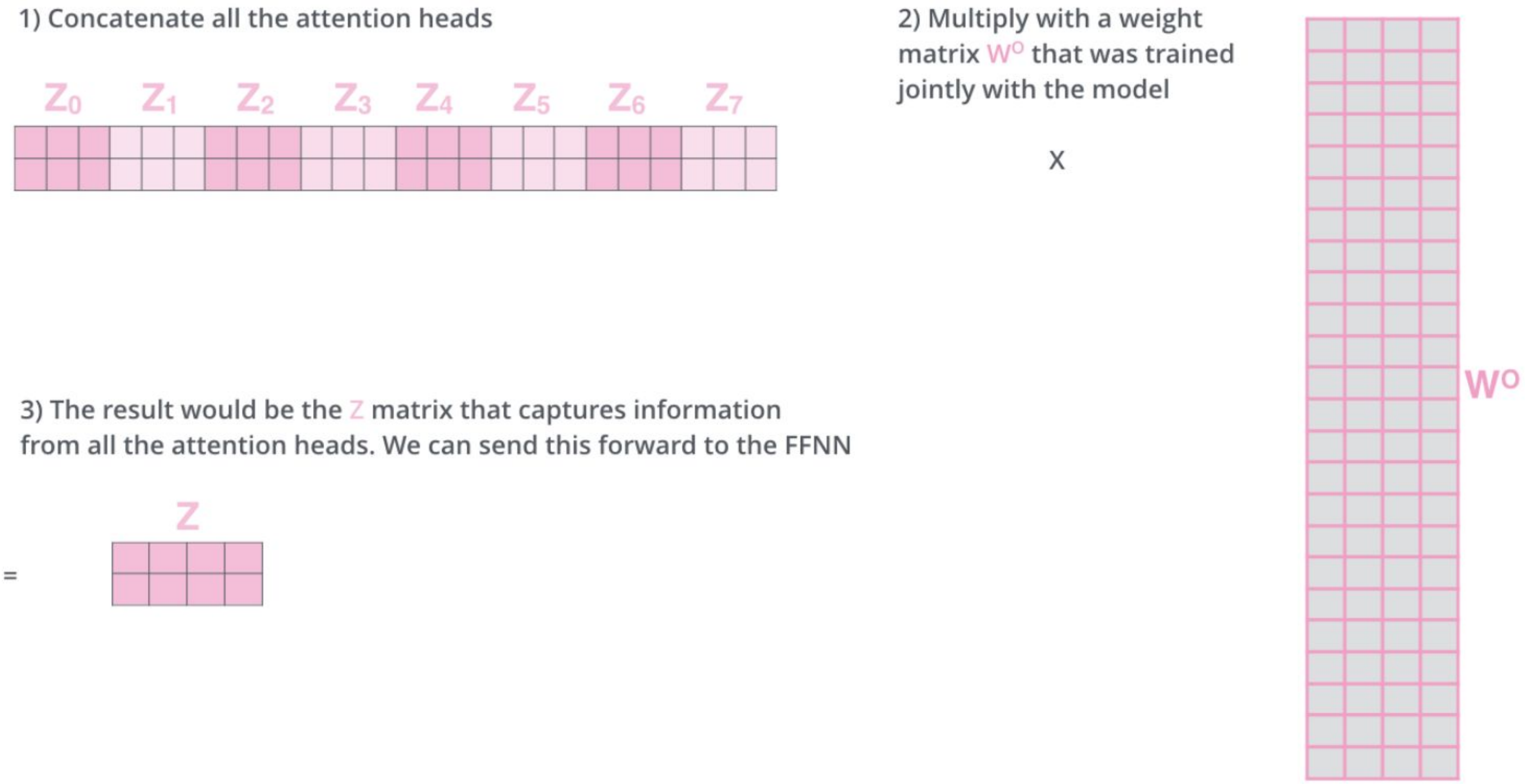

https://jalammar.github.io/illustrated-transformer/

# Multi-Head Attention




https://jalammar.github.io/illustrated-transformer/

# Multi-Head Attention




https://jalammar.github.io/illustrated-transformer/

1) This is our input sentence*

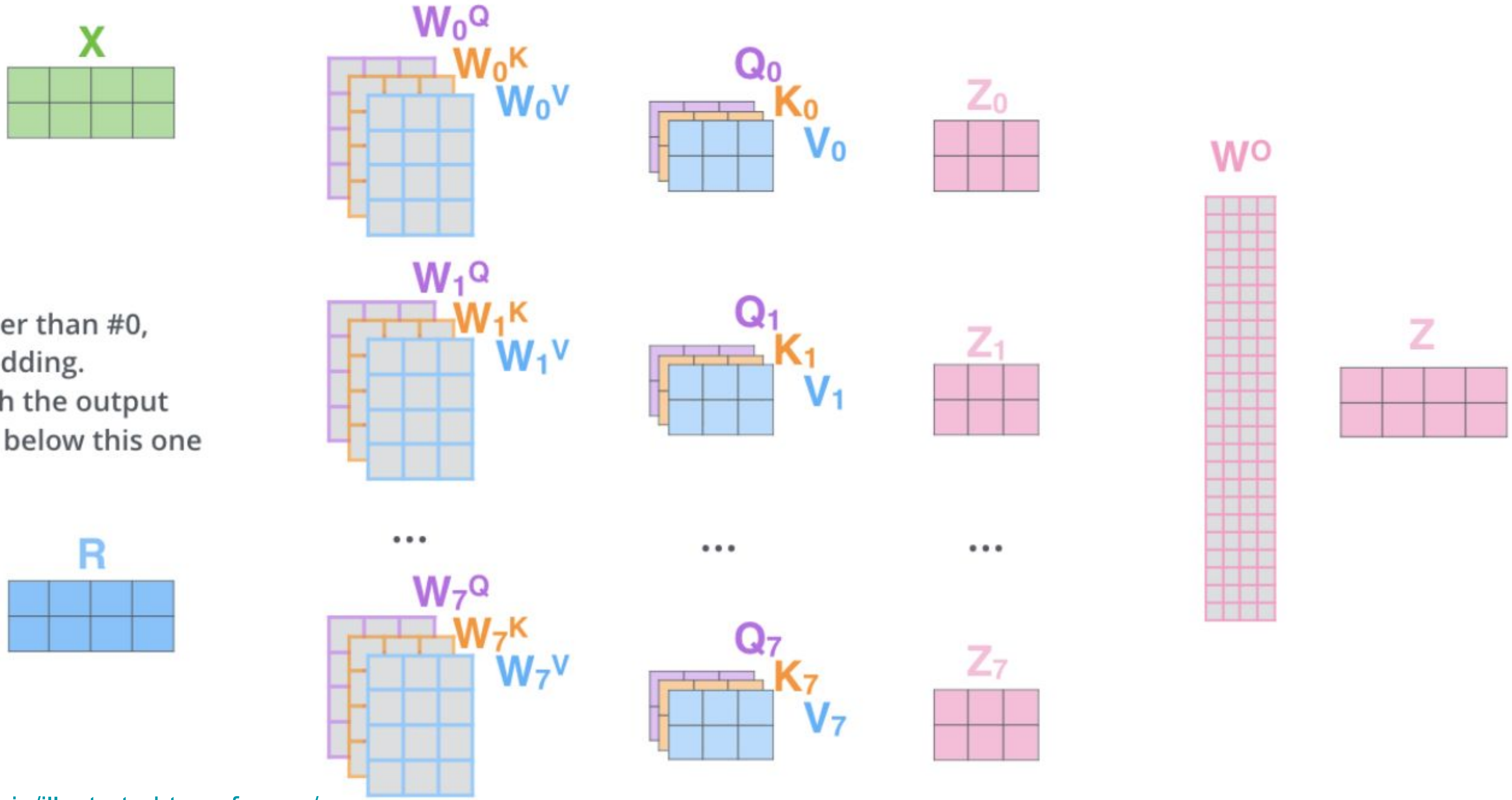
2) We embed each word*

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

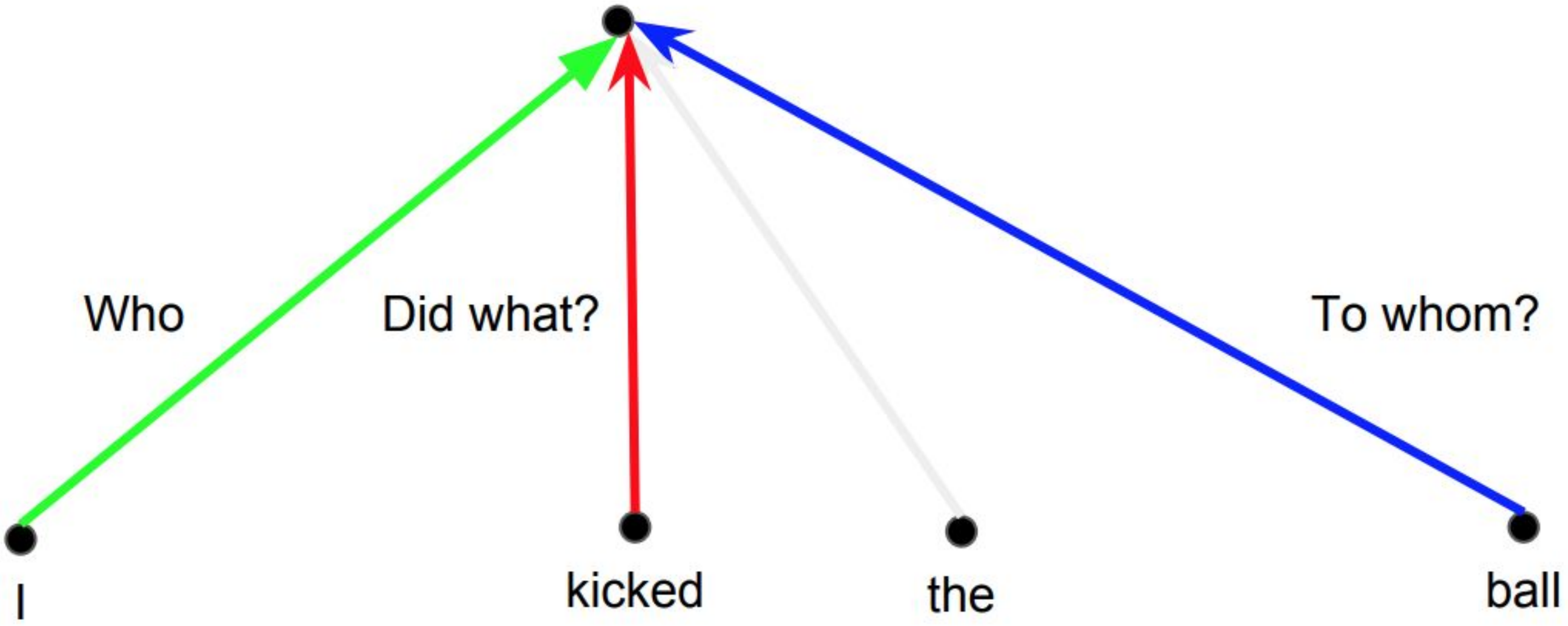
Thinking
Machines



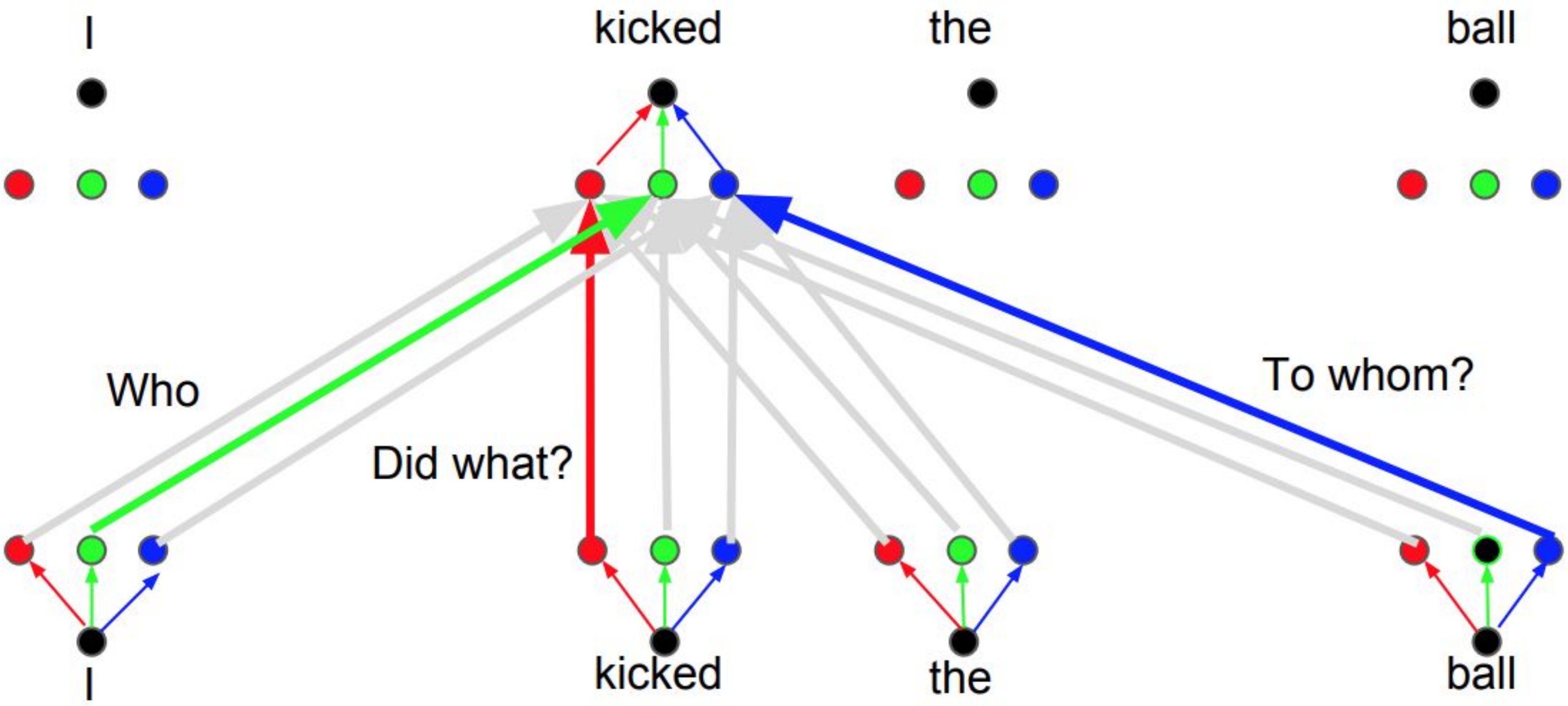

* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

https://jalammar.github.io/illustrated-transformer/

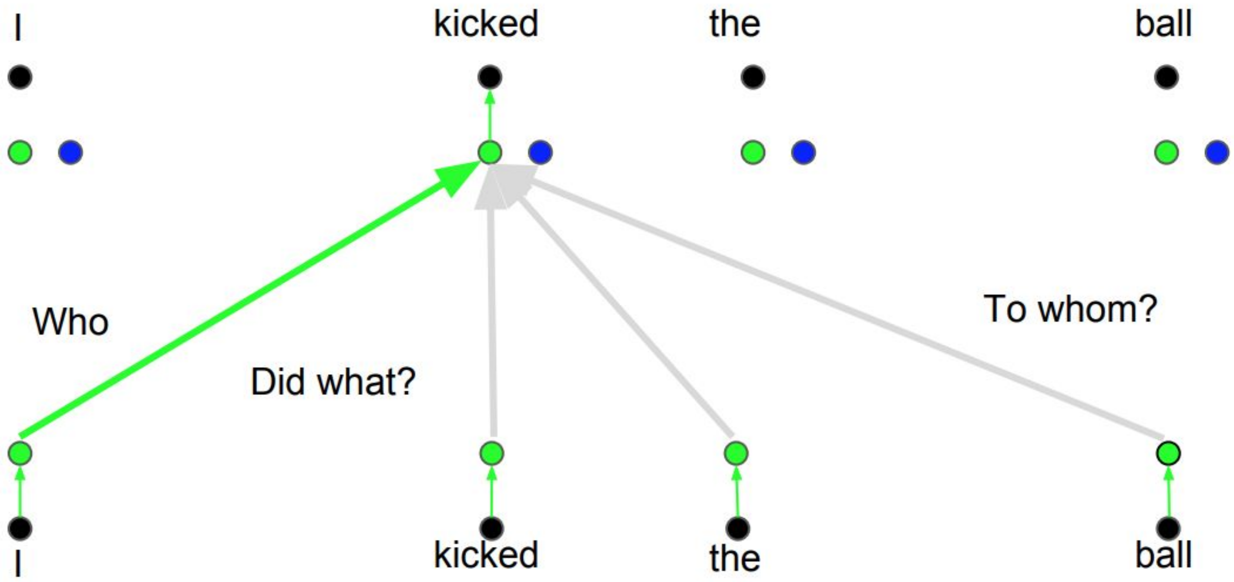
# Why Multi-Head Attention?



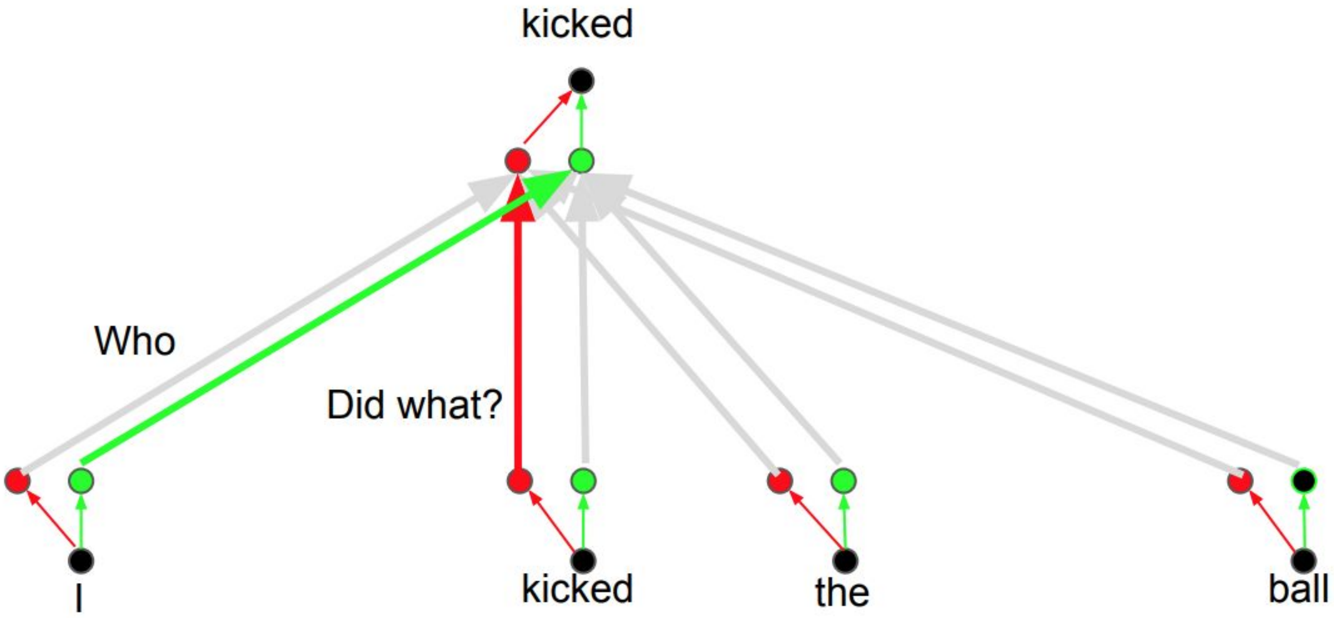
# Why Multi-Head Attention?



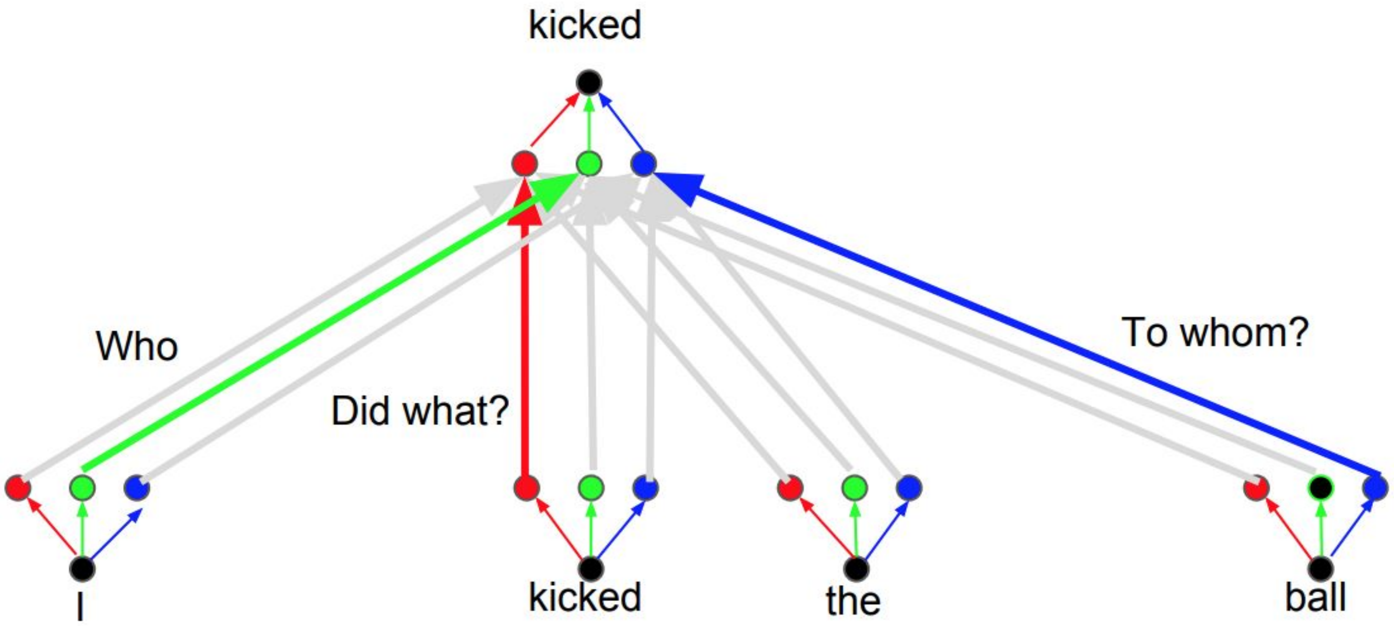
# Attention head: Who



# Attention head: Did What?

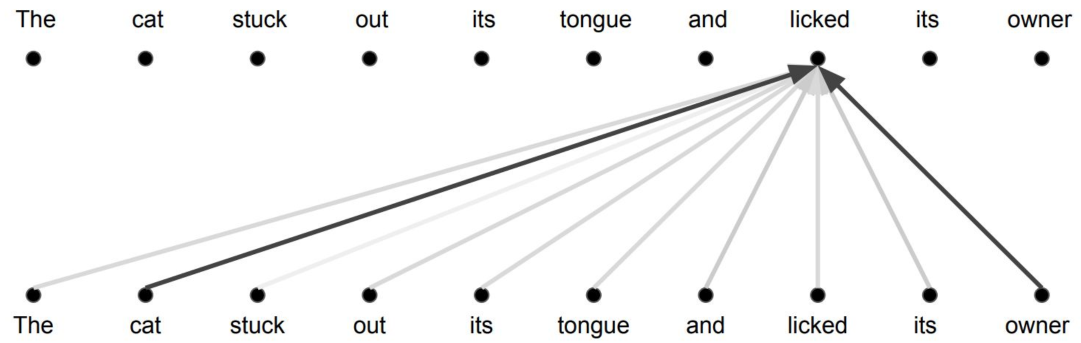


# Attention head: To Whom?



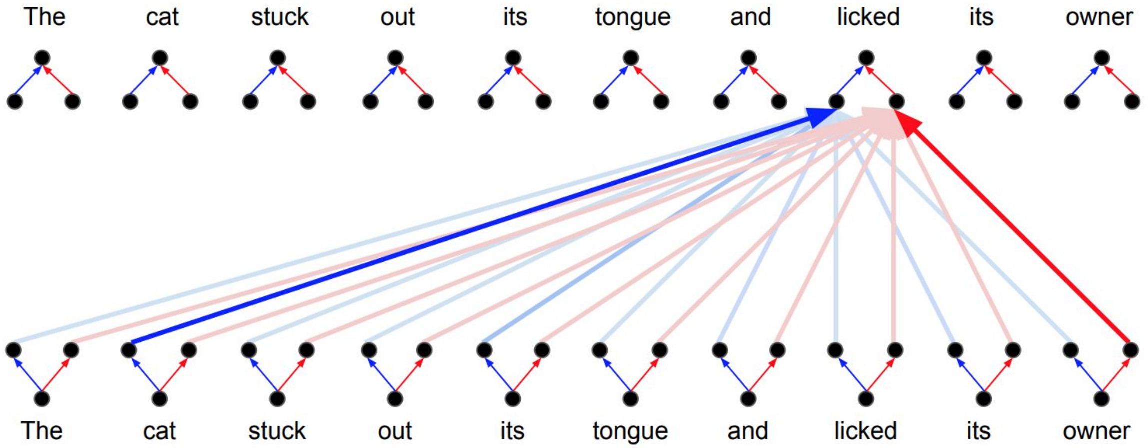
# Attention vs. Multi-Head Attention

**Attention:** a weighted average




**Multi-Head Attention:**

parallel attention layers with different linear transformations on input and output.



# Performance: WMT 2014 BLEU

| | EN-DE | EN-FR |
|---|---|---|
| GNMT (orig) | 24.6 | 39.9 |
| ConvSeq2Seq | 25.2 | 40.5 |
| Transformer* | **28.4** | **41.8** |

*Transformer models trained >3x faster than the others.

# Attention is fast

**Complexity per layer**

| | |
|---|---|
| Self-Attention | |
| RNN (LSTM) | |

# Attention is fast

**Complexity per layer**

| | |
|---|---|
| Self-Attention | O($\text{length}^2 \cdot \text{dim}$) |
| RNN (LSTM) | O($\text{length} \cdot \text{dim}^2$) |

- length - the length of the sequence.
- dim - the dimension of the representation (512, 1024 in general)

# Attention is fast

| | FLOPs | |
|---|---|---|
| Self-Attention | $O(\text{length}^2 \cdot \text{dim})$ | $= 4 \cdot 10^9$ |
| RNN (LSTM) | $O(\text{length} \cdot \text{dim}^2)$ | $= 16 \cdot 10^9$ |

length=1000 dim=1000 kernel_width=3

# Research Challenges

- Constant ‘path length’ between any two positions.
- Unbounded memory.
- Trivial to parallelize (per layer).
- Models Self-Similarity.
- Relative attention provides expressive timing, equivariance, and extends naturally to graphs.