CSCI 1470

# Deep Learning

Eric Ewing

Day 21: Attention!

Monday, 3/10/25

# Logistics

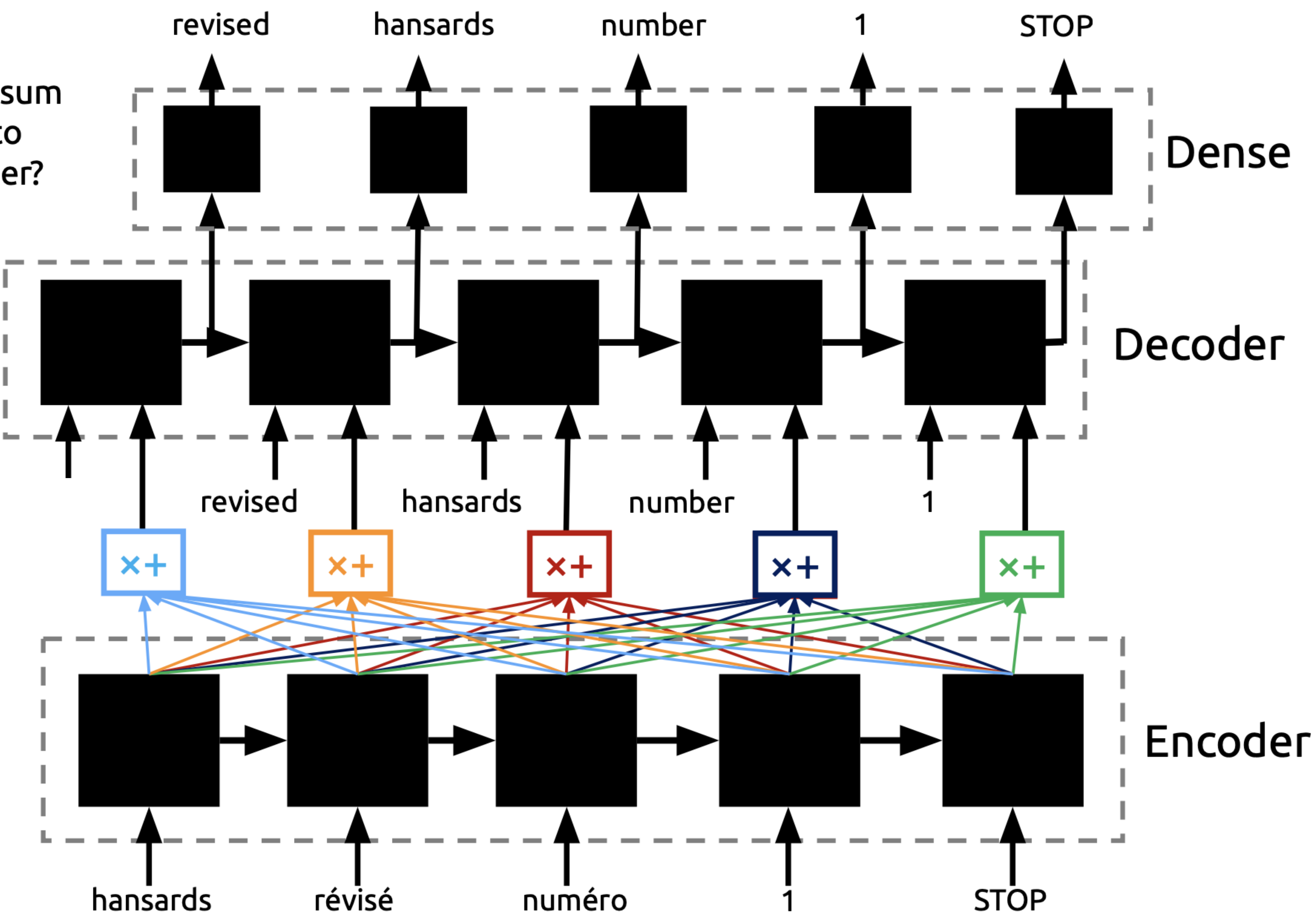
- Guest lecture on Monday by Jason Liu about language grounding
- Final Project groups (and your TA) will be finalized soon

What if we passed the sum of our encoder states to ***every cell*** in the decoder?

What if the sum was a ***weighted sum*** instead?

What if each decoder cell received a ***different*** weighted sum?

- Idea: different words in the input carry different importance *for each word in the output*



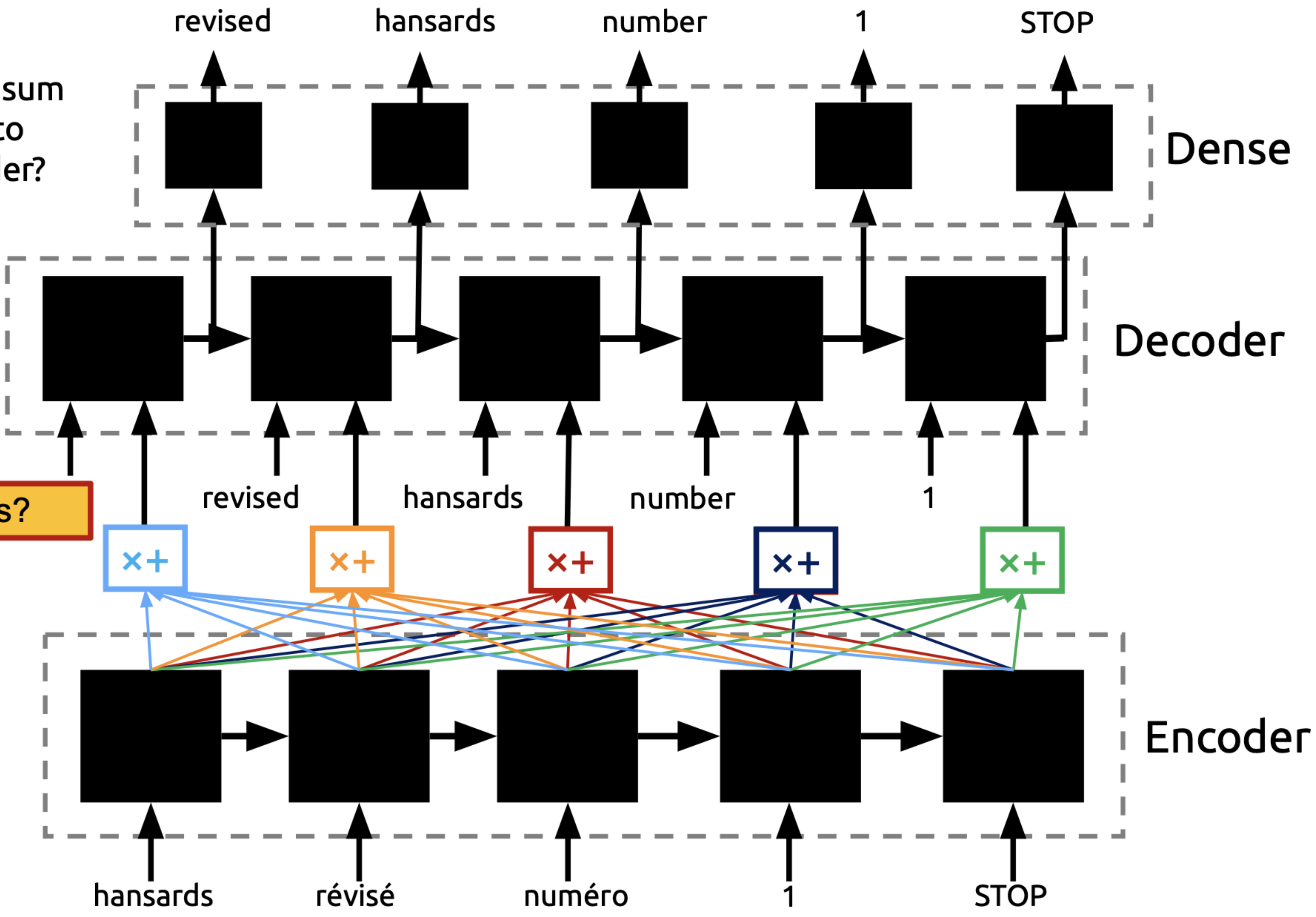
What if we passed the sum of our encoder states to ***every cell*** in the decoder?

What if the sum was a ***weighted sum*** instead?

How do we achieve this?

What if each decoder cell received a ***different*** weighted sum?

- Idea: different words in the input carry different importance *for each word in the output*



# "Attention"

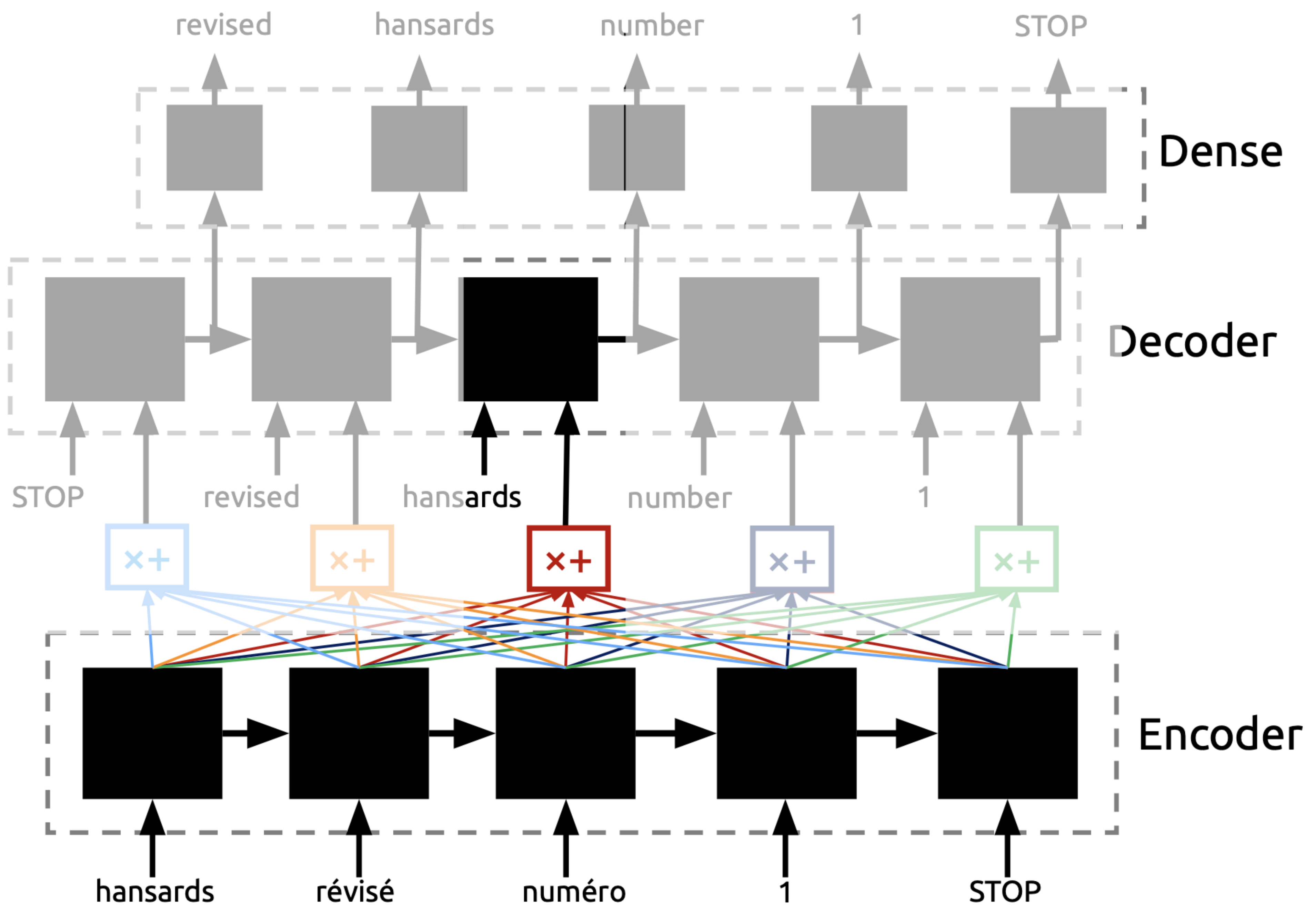
This idea of passing each cell of the decoder a weighted sum of the encoder states is called ***attention.***

- Different words in the output "pay attention" to different words in the input

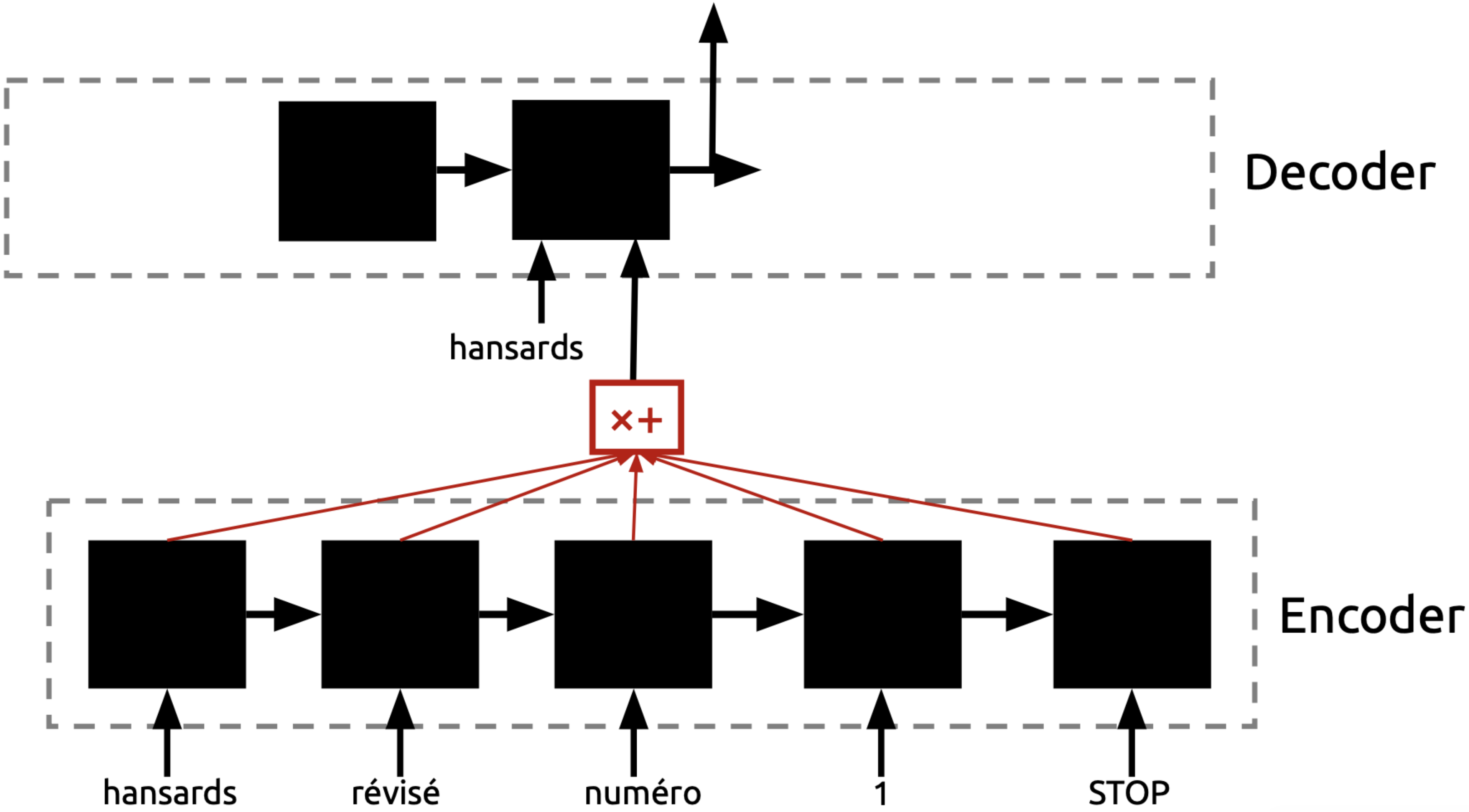
# “Attention” - intuition

“Park”

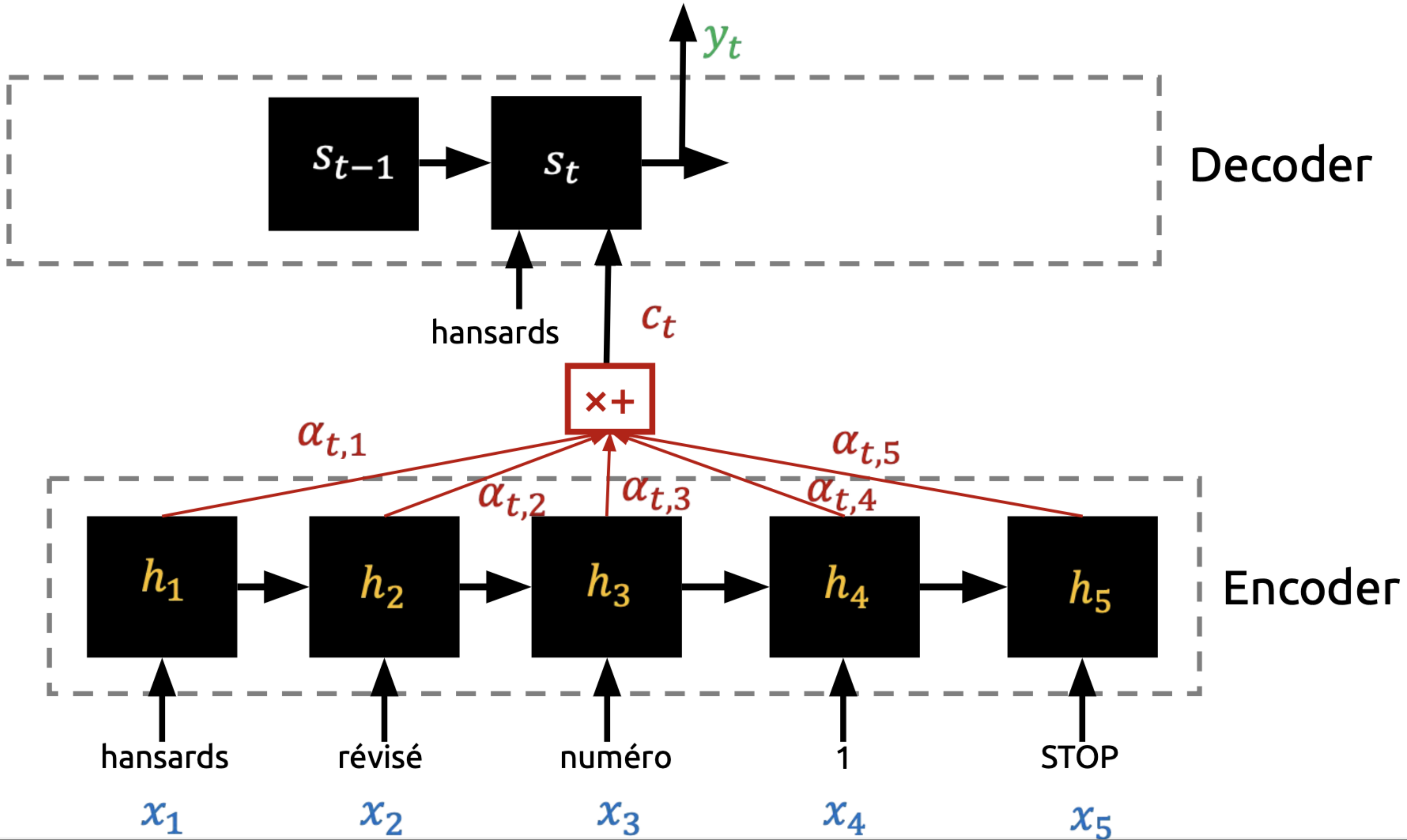
How about we let model learn what is relevant for a particular output


revised
hansards
number
1
STOP
Dense
Decoder
STOP
revised
hansards
number
1
×+
×+
×+
×+
×+
Encoder
hansards
révisé
numéro
1
STOP

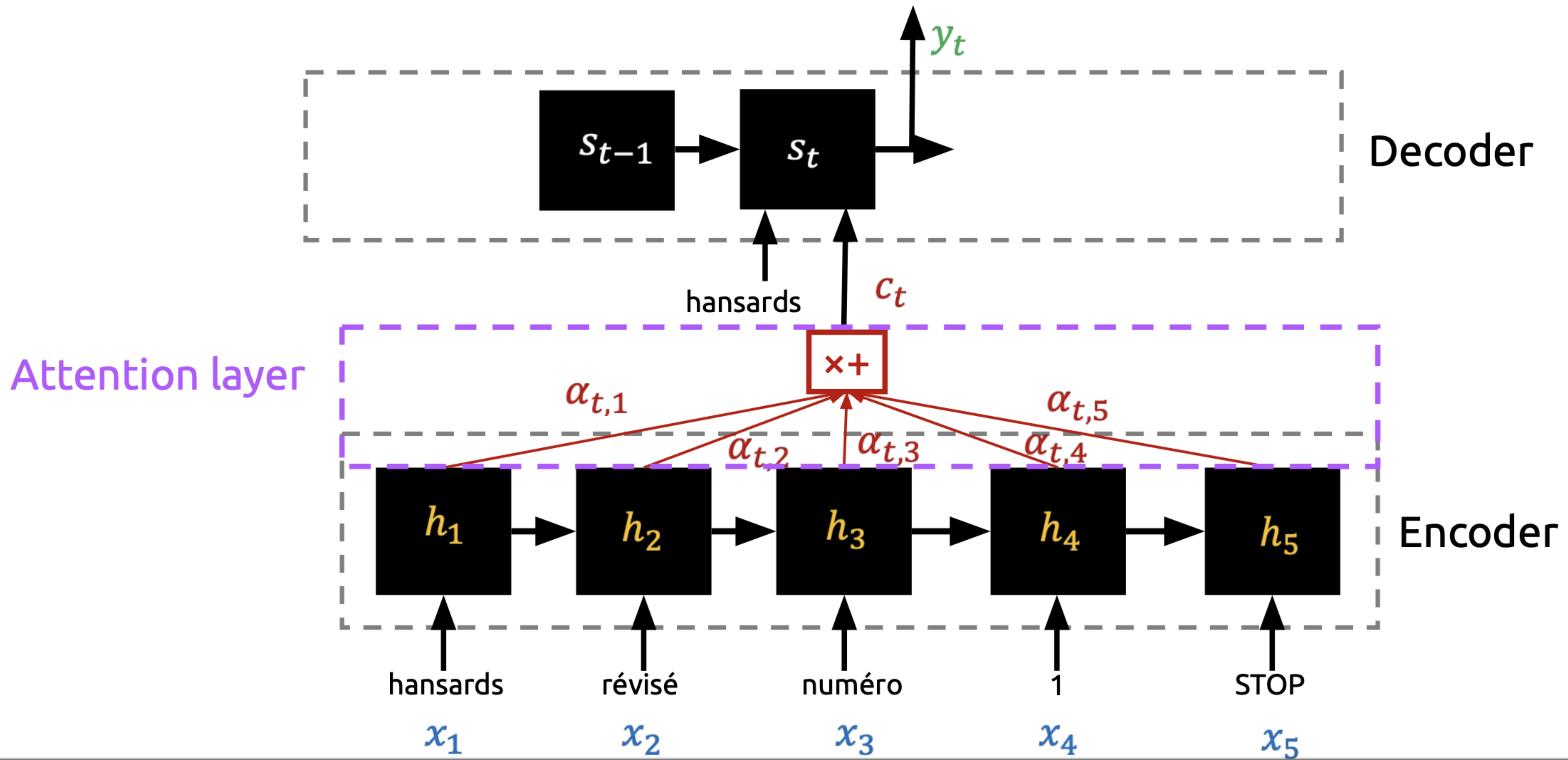
# Attention - implementation



# Attention - implementation



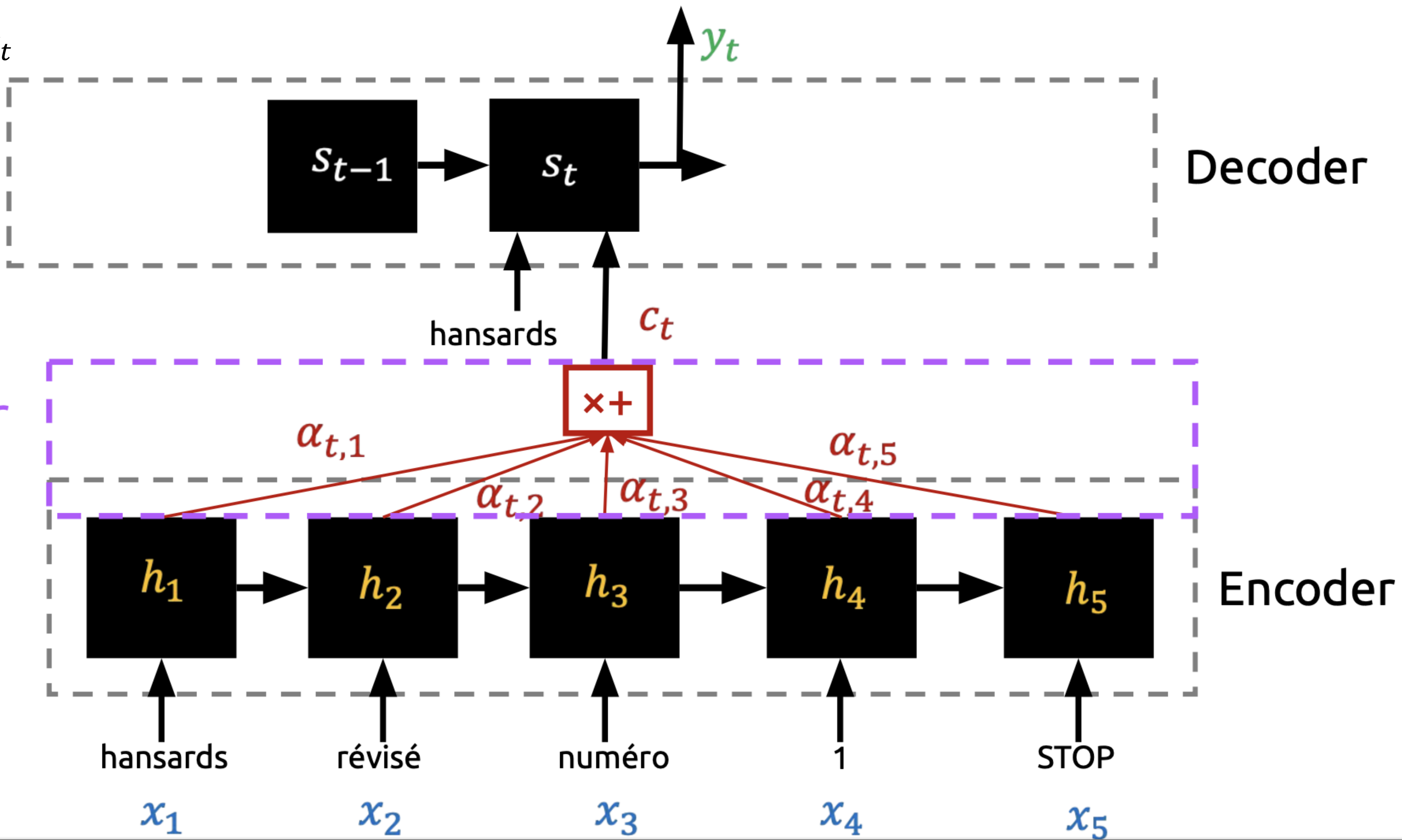
# Attention - implementation



# Attention - implementation

Context Vector for output $y_t$

$$c_t = \sum_{i=1} a_{t,,i} h_i$$



# Attention - implementation

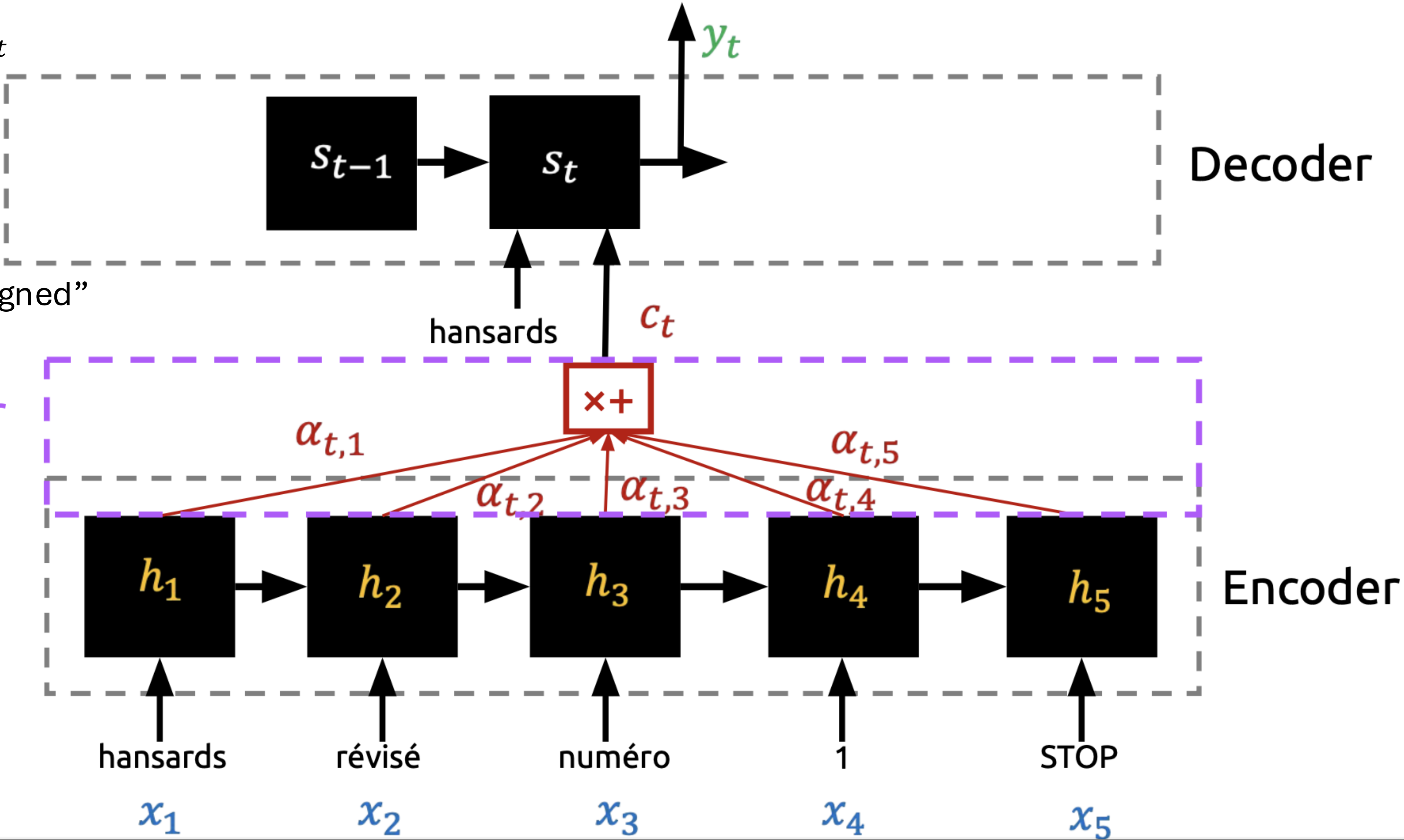
Context Vector for output $y_t$

$$c_t = \sum_{i=1} a_{t,,i} h_i$$

How well two words are "aligned"

$$a_{t,i} = align(x_t, y_t)$$

Attention layer



# Attention - implementation

Context Vector for output $y_t$

$$c_t = \sum_{i=1} a_{t,,i} h_i$$

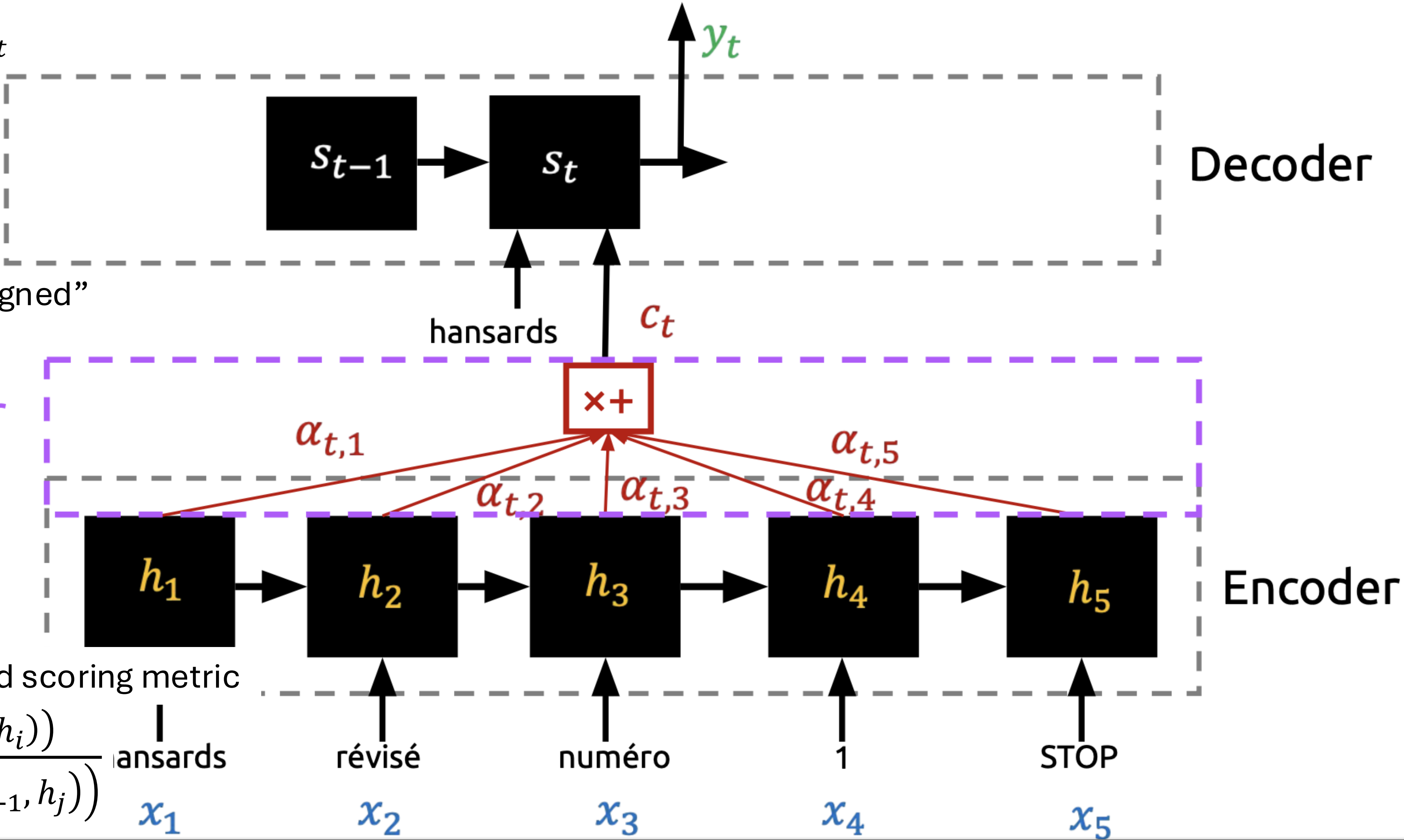
How well two words are "aligned"

$$a_{t,i} = align(x_t, y_t)$$

Attention layer

Softmax of some predefined scoring metric

$$a_{t,i} = \frac{\exp(score(s_{t-1}, h_i))}{\sum_{j=1} \exp\left(score(s_{t-1}, h_j)\right)}$$



# Attention Alignment Score

- Need to determine how well output word $y_t$ aligns with each input word $x_i$
- How can we determine the similarity between two words (or at least the vectors that represent them)?

# Attention Alignment Score

- Need to determine how well output word $y_t$ aligns with each input word $x_i$
- How can we determine the similarity between two words (or at least the vectors that represent them)?

$$\text{Cosine Similarity}(h_i, s_{t-1}) = \frac{h_i s_{t-1}}{||h_i|| * ||s_{t-1}||}$$

# Attention Alignment Score

- Need to determine how well output word $y_t$ aligns with each input word $x_i$
- How can we determine the similarity between two words (or at least the vectors that represent them)?

Cosine Similarity$(h_i, s_{t-1}) = \frac{h_i s_{t-1}}{||h_i|| * ||s_{t-1}||}$

Dot product similarity$(h_i, s_{t-1}) = h_i s_{t-1}$

# Attention Alignment Score

- Need to determine how well output word $y_t$ aligns with each input word $x_i$
- How can we determine the similarity between two words (or at least the vectors that represent them)?

$$\text{Cosine Similarity}(h_i, s_{t-1}) = \frac{h_i s_{t-1}}{||h_i|| * ||s_{t-1}||}$$

$$\text{Dot product similarity}(h_i, s_{t-1}) = h_i s_{t-1}$$

$$\text{Generalized Similarity}(h_i, s_{t-1}) = h_i W_a s_{t-1}$$

Learned attention weight matrix
How much do we care about each part of embedding?

# There are many ways to measure similarity...

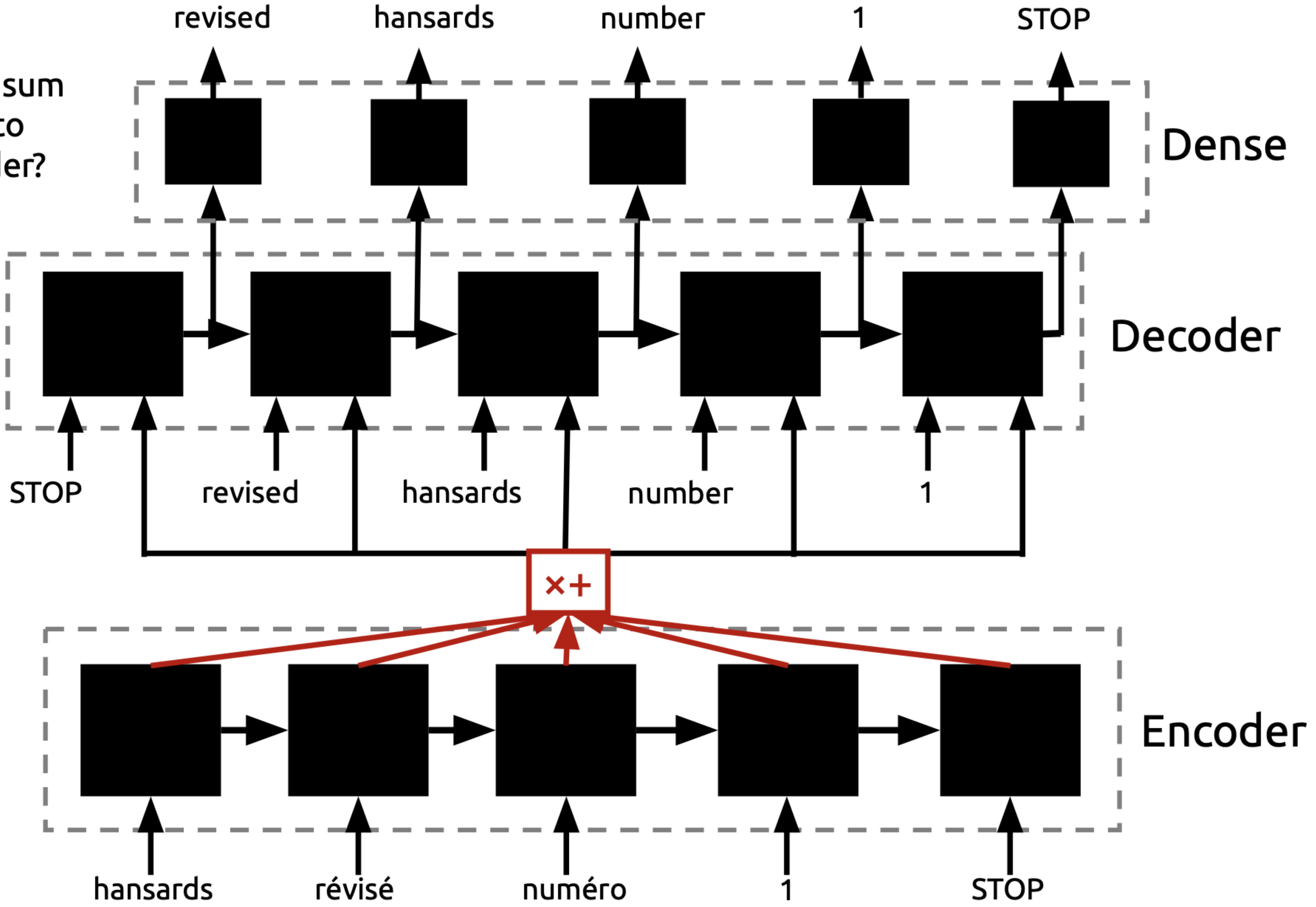
| Name | Alignment score function | Citation |
|---|---|---|
| Content-base attention | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$ | Graves2014 |
| Additive(*) | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$ | Bahdanau2015 |
| Location-Base | $\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$<br>Note: This simplifies the softmax alignment to only depend on the target position. | Luong2015 |
| General | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$<br>where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer. | Luong2015 |
| Dot-Product | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$ | Luong2015 |
| Scaled Dot-Product(^) | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$<br>Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state. | Vaswani2017 |

Source: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

What if we passed the sum of our encoder states to ***every cell*** in the decoder?

What if the sum was a ***weighted sum*** instead?

- Idea: different words in the input carry different importance



# Attention Example

We can represent the attention weights as a matrix:

Columns: words in the input

Rows: words in the output

| | hansards | révisé | numéro | 1 | STOP |
|---|---|---|---|---|---|
| revised | 1/2 | 1/4 | 1/4 | 0 | 0 |
| hansards | 1/4 | $\boxed{1/2}$ | 1/4 | 0 | 0 |
| number | 0 | 1/4 | 1/2 | 1/4 | 0 |
| 1 | 0 | 0 | 1/4 | 1/2 | 1/4 |
| STOP | 0 | 0 | 1/4 | 1/4 | 1/2 |

$\alpha_{j,i}$ : how much 'attention' output word j pays to input word i

***What do the values in this particular matrix imply about the attention relationship between input/output words?***

# Attention Example

Target: “Der Hund <u>bellte</u> mich an.”

Input: “The dog barked at me.”

[0, 1/4, 1/2, 1/4, 0]

We see that when we apply the attention to our inputs, we will pay attention to relatively important words for translation when predicting “bellte”.

# Attention is great!

- Attention significantly improves MT 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

Credit: Stanford CS224n

# Attention is a general deep learning technique

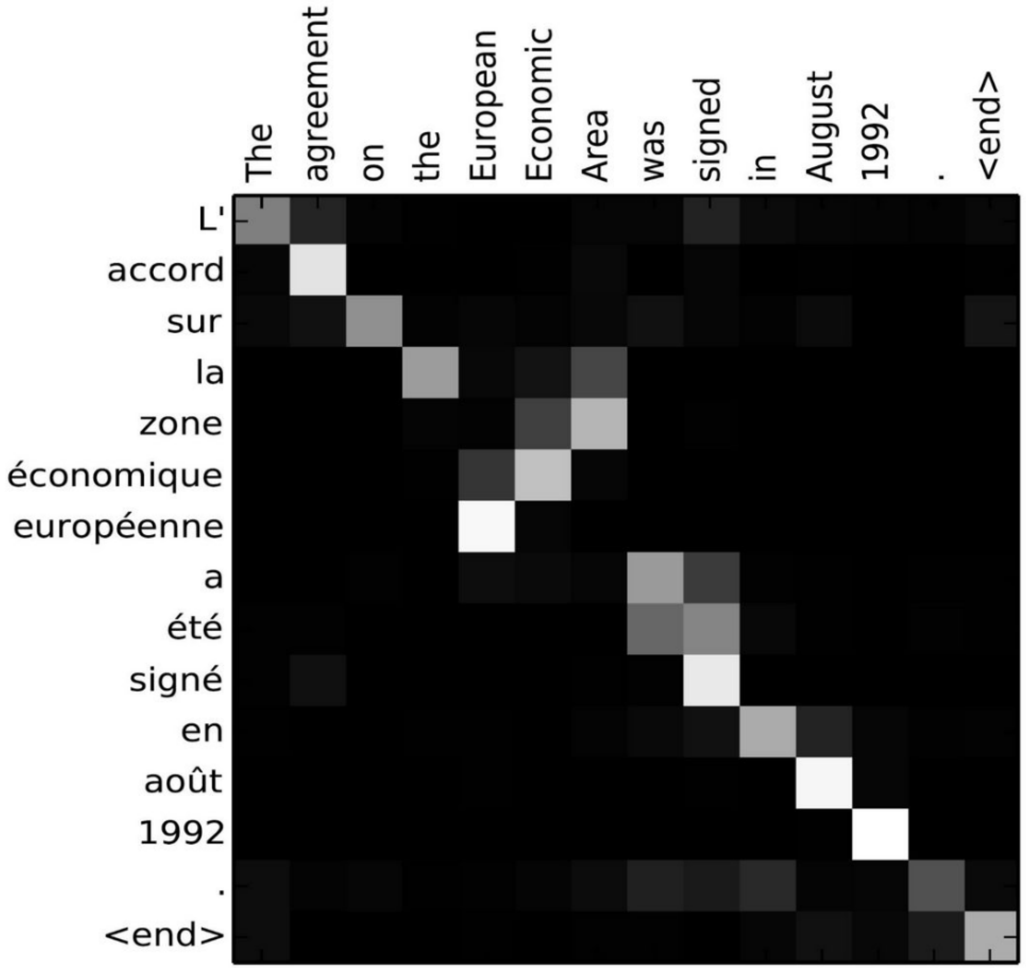
**More general definition of attention**:

Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.

**Intuition**:

- The weighted sum is a *selective summary* of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

# Attention in Language Translation




Courtesy: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

# Image captioning with CNNs, RNNs, and Attention

A dog is standing on a hardwood floor.


A stop sign is on a road with a mountain in the background.

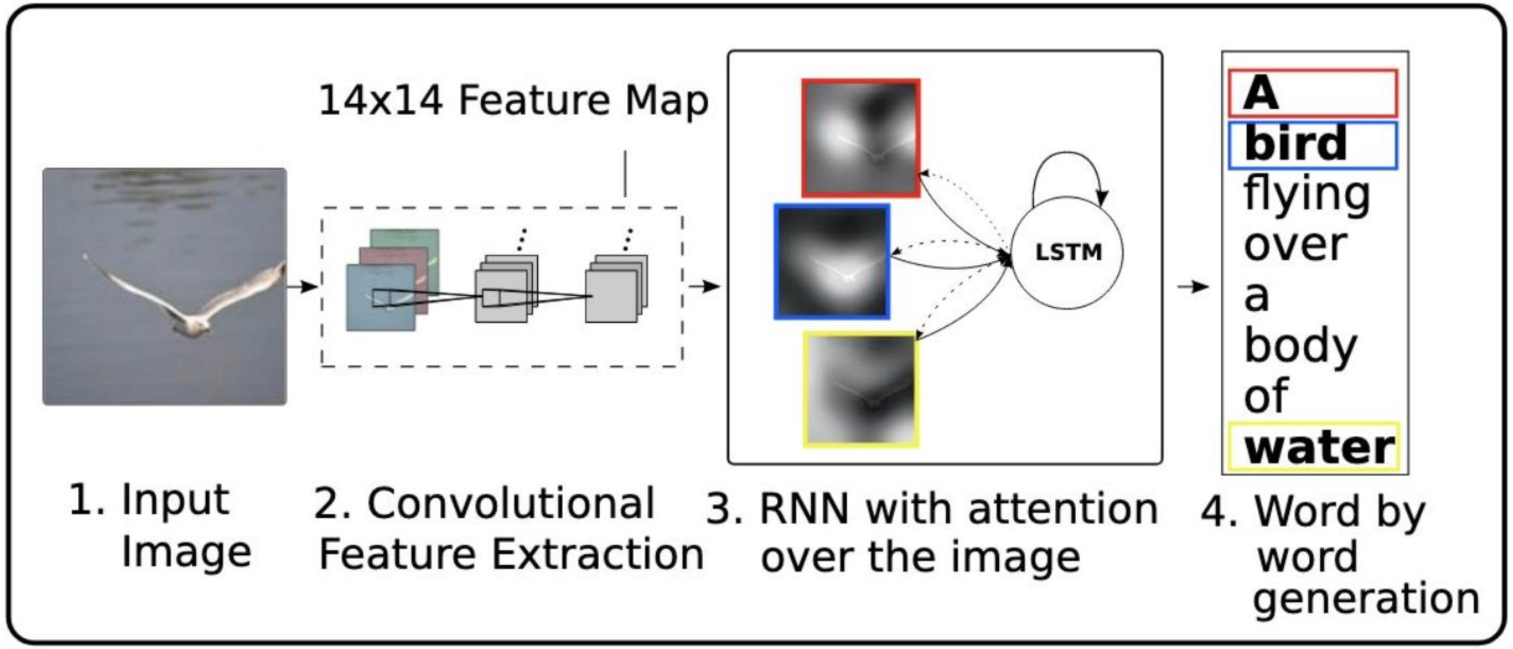
A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

***Think-pair-share:***

***How would you design this architecture with attention?***

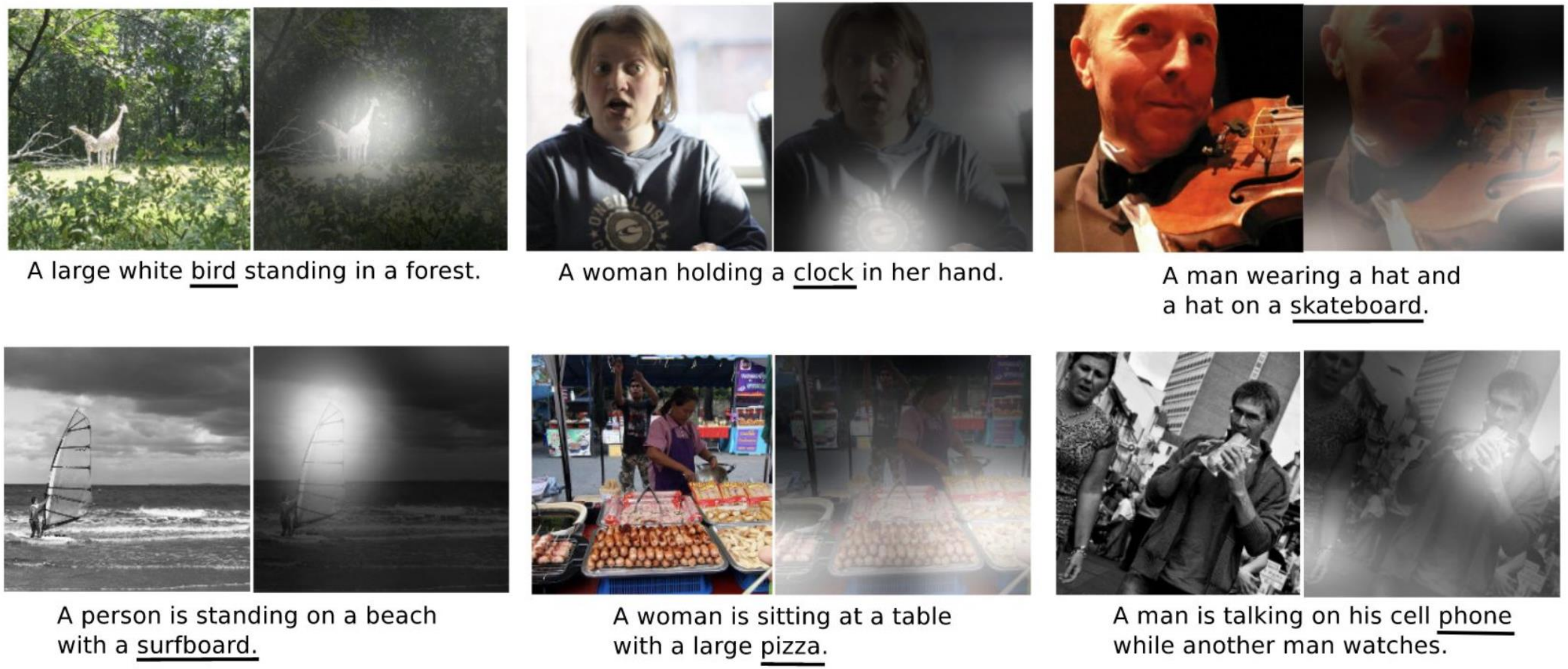
# Image captioning with CNNs, RNNs, and Attention




Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

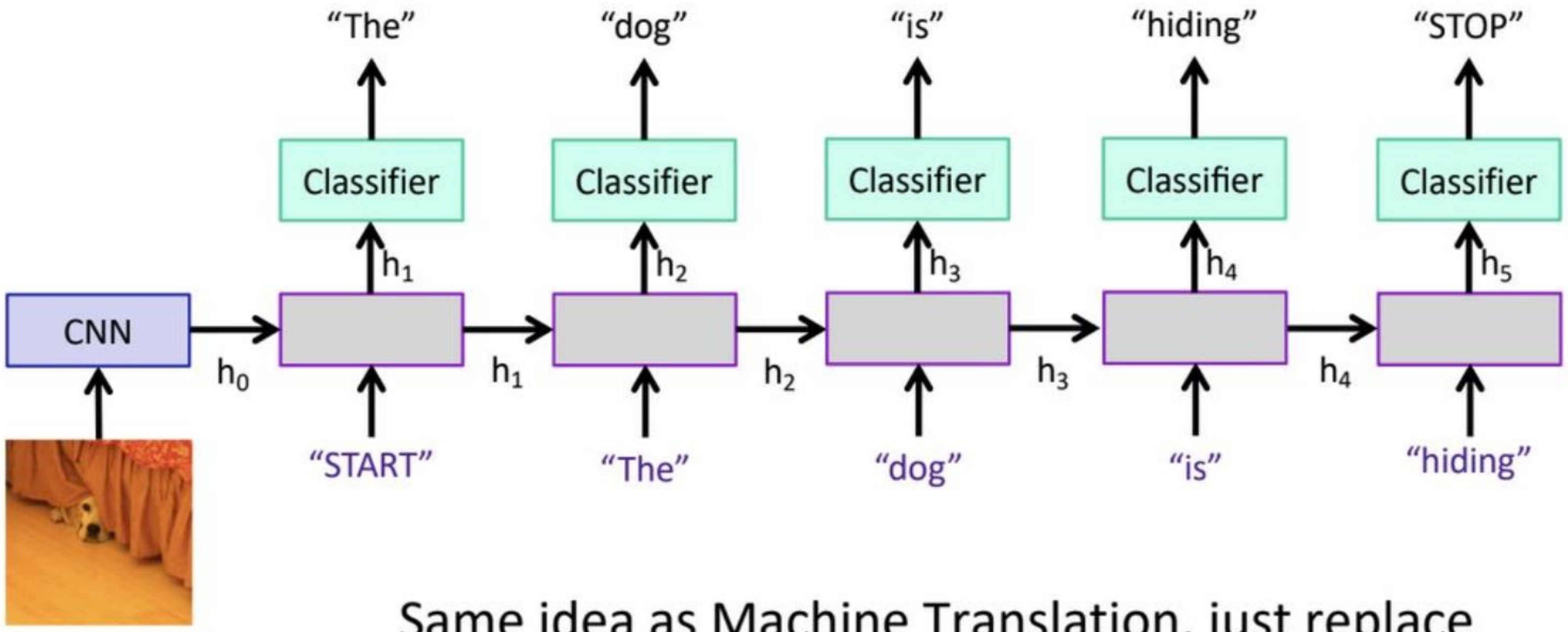
# Image captioning with CNNs, RNNs, and Attention

*Figure 5.* Examples of mistakes where we can use attention to gain intuition into what the model saw.




Xu et al, “Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

# Image captioning (HW5)




Same idea as Machine Translation, just replace E_s with an image-level embedding.

Lazebnik

# Do we still need the RNNs?

After all, we always compute the weighted sum of **all encoder states**.

# "Attention Is All You Need"

A 2017 paper that introduced the ***Transformer*** model for machine translation

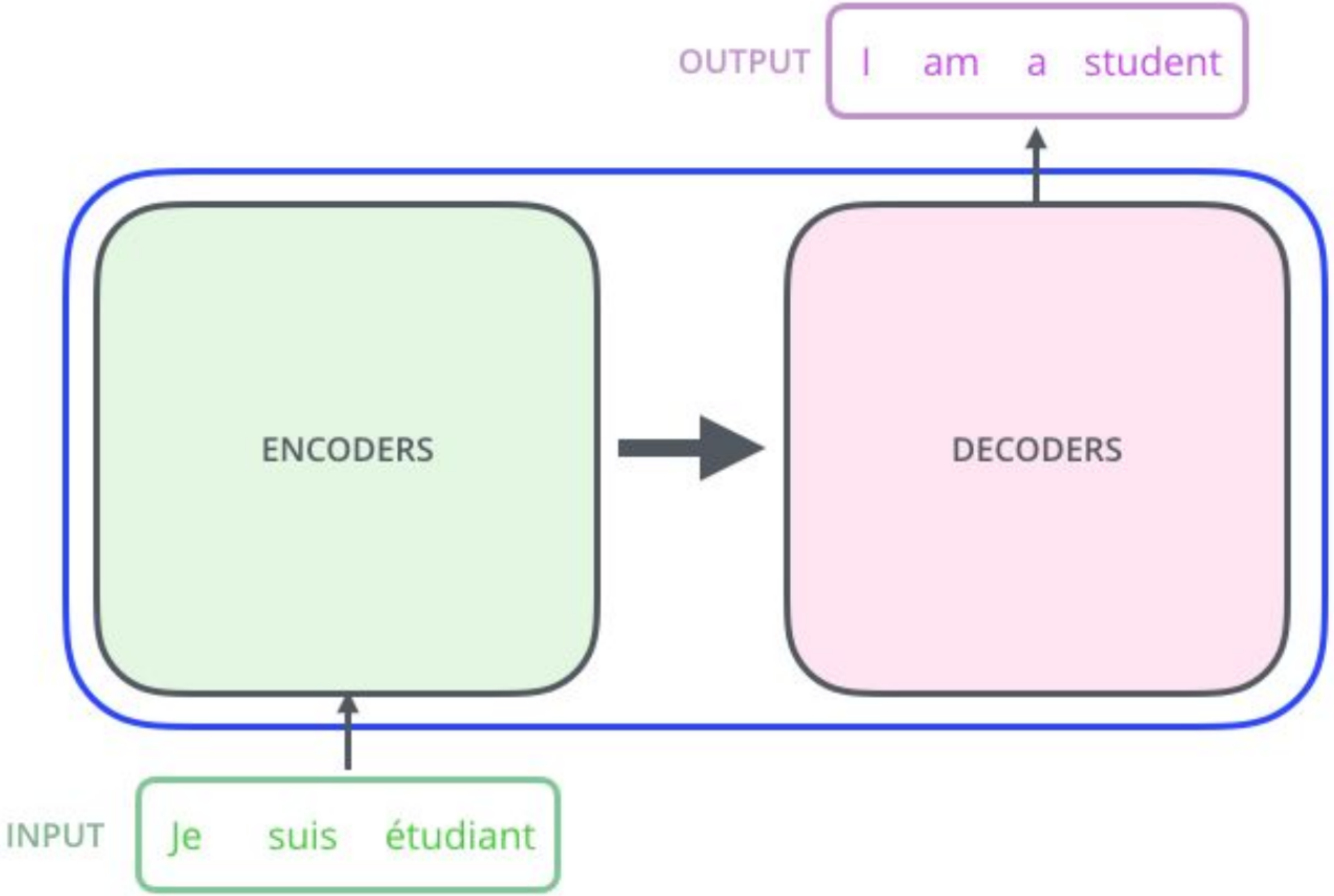
- Has no recurrent networks!
- ***Only*** uses attention

**Motivation:**

- RNN training is hard to parallelize since the previous word must be processed before next word
  - Transformers are trivially parallelizable
- Even with LSTMs / GRUs, preserving important linguistic context over ***very*** long sequences is difficult
  - Transformers don't even try to remember things (every step looks at a weighted combination of ***all*** words in the input sentence)

# Transformer Model Overview

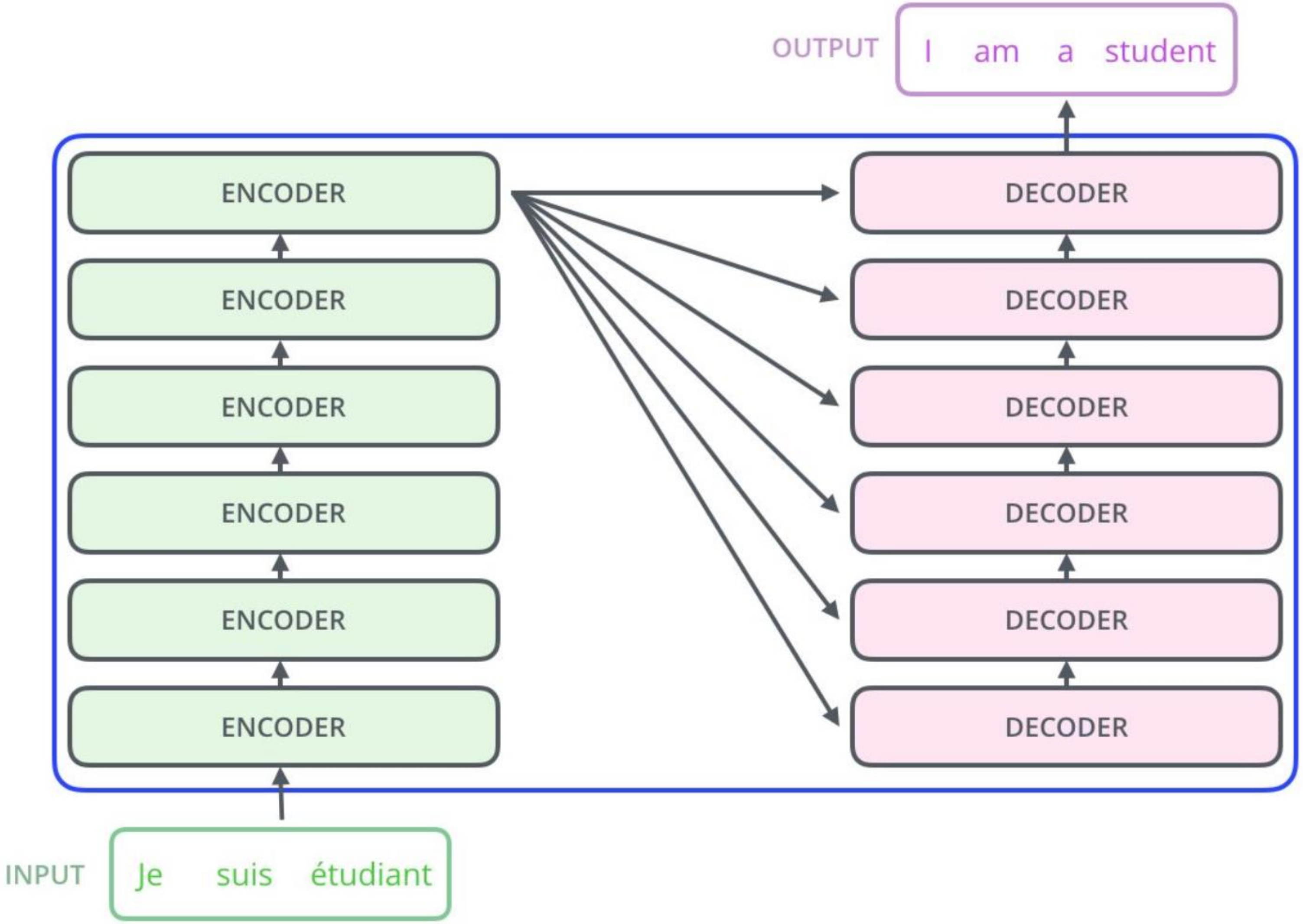
- The Transformer model breaks down into Encoder and Decoder blocks.
- At a high level, similar to the seq2seq architecture we've seen already...
- ...but there are no recurrent nets inside the Encoder and Decoder blocks!




Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Transformer Model Overview

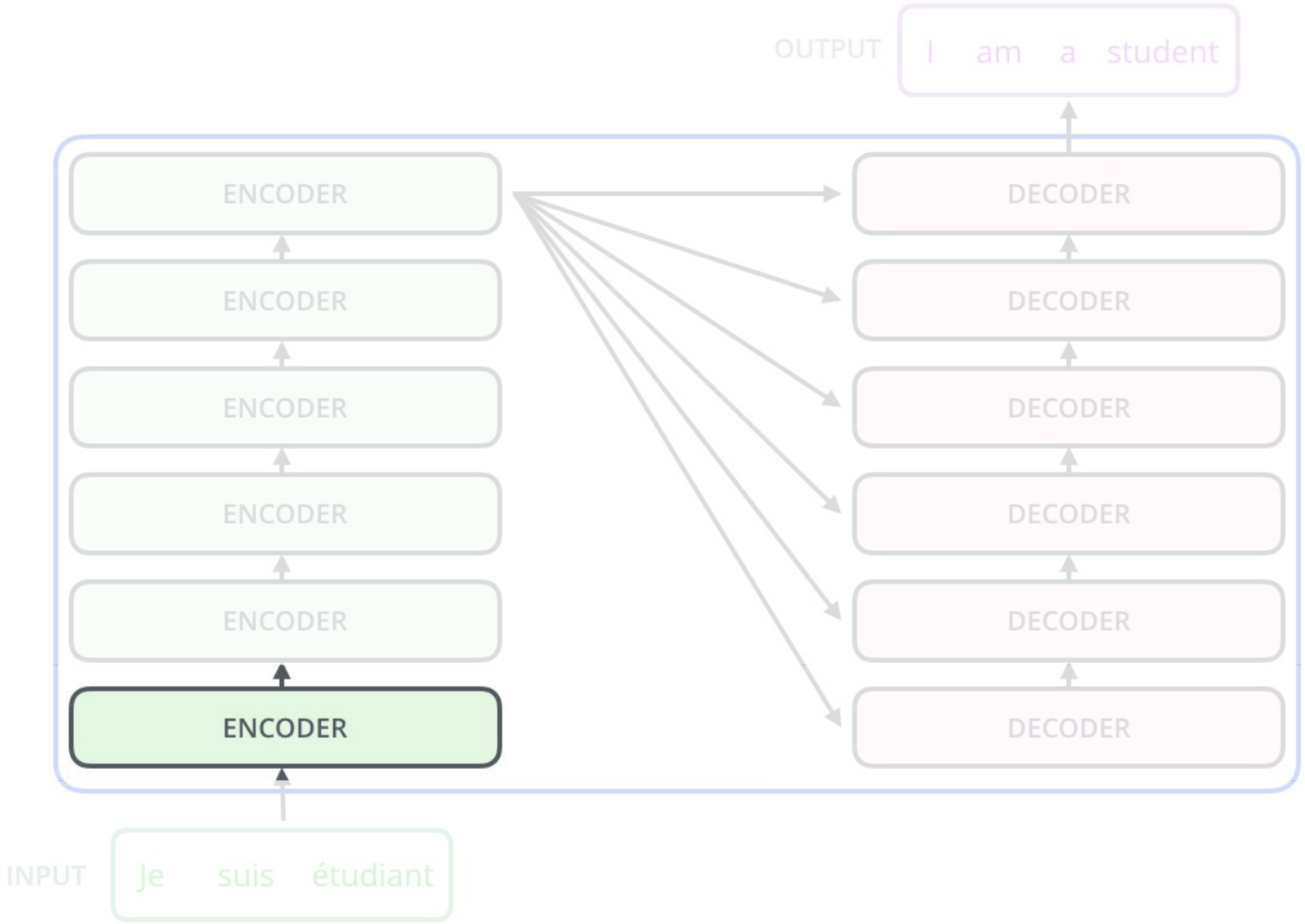
- The Transformer model breaks down into Encoder and Decoder blocks.
- At a high level, similar to the seq2seq architecture we've seen already…
- …but there are no recurrent nets inside the Encoder and Decoder blocks!
- For better performance, often stack multiple Encoder and Decoder blocks (deeper network)




Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Transformer Model Overview

- Let's look at what goes on inside one of these Encoder blocks



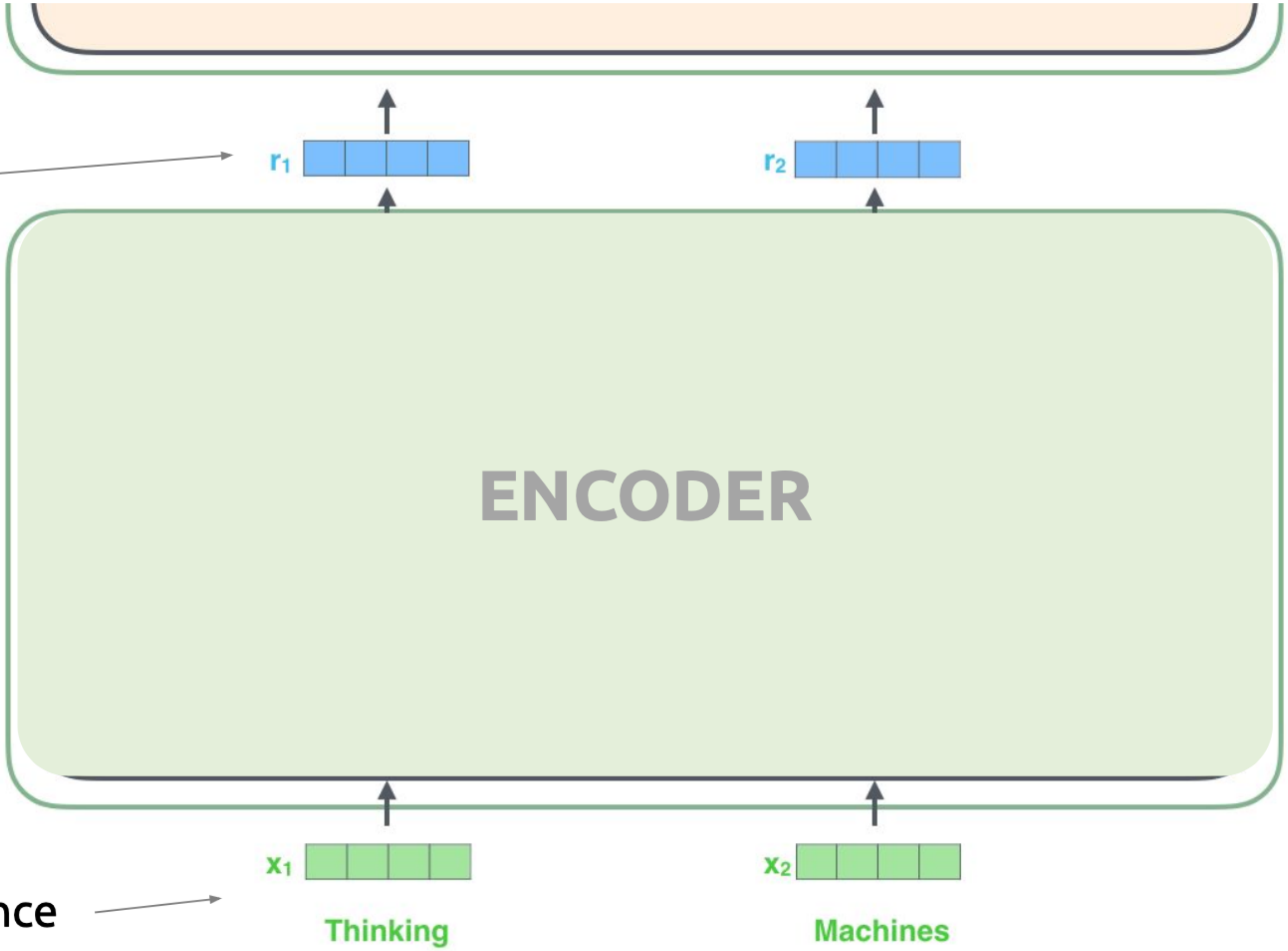



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Encoder Block Map

These per-word output vectors are analogous to the LSTM hidden states from the seq2seq2 model

- They should capture *"what information about the input sentence is relevant to translating this word?"*



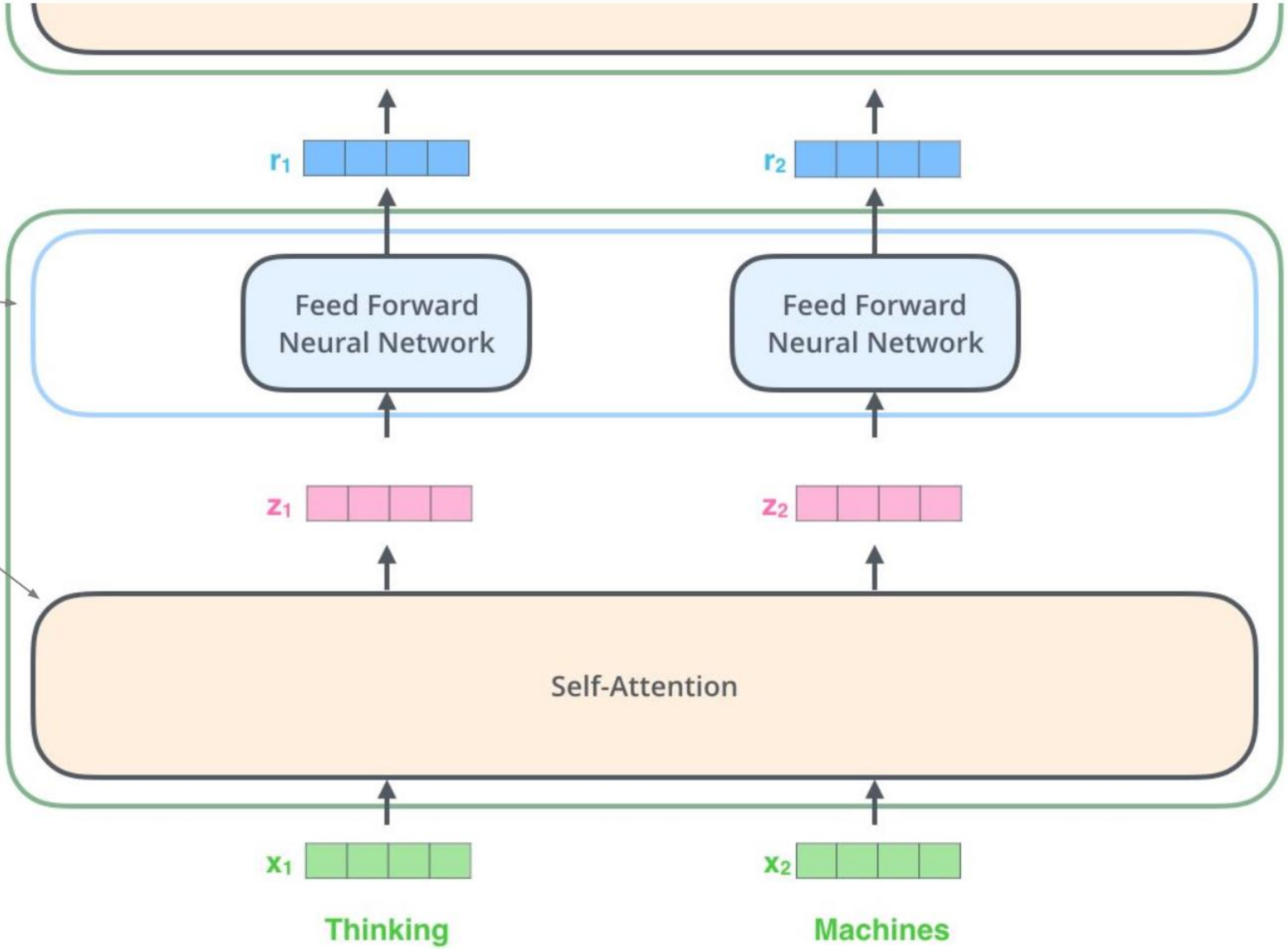

Words in input sentence



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Encoder Block Map

- Encoder block breaks down into two main parts: Self-Attention, and Feed Forward layers.

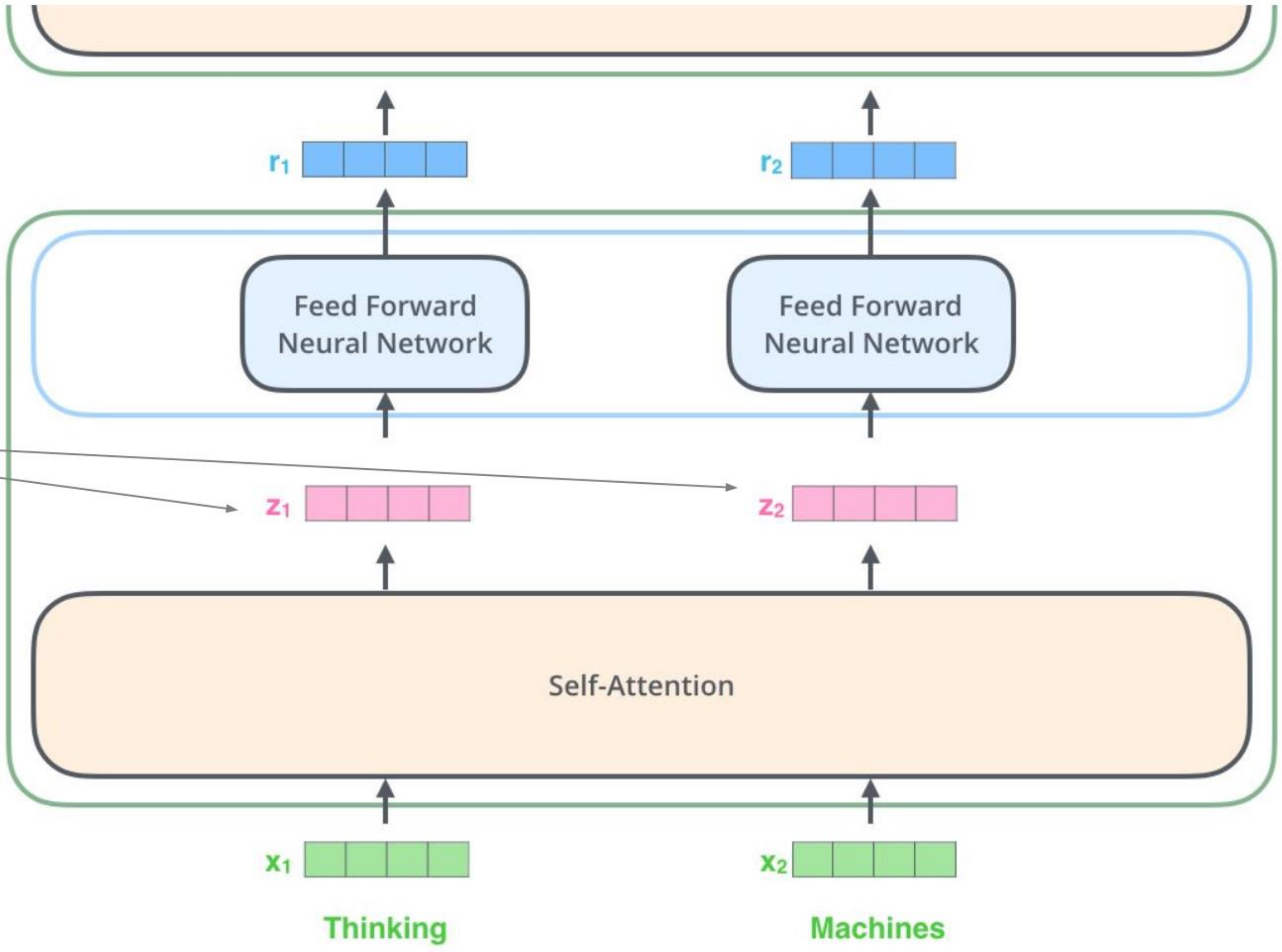



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Encoder Block Map

- Encoder block breaks down into two main parts: Self-Attention, and Feed Forward layers.

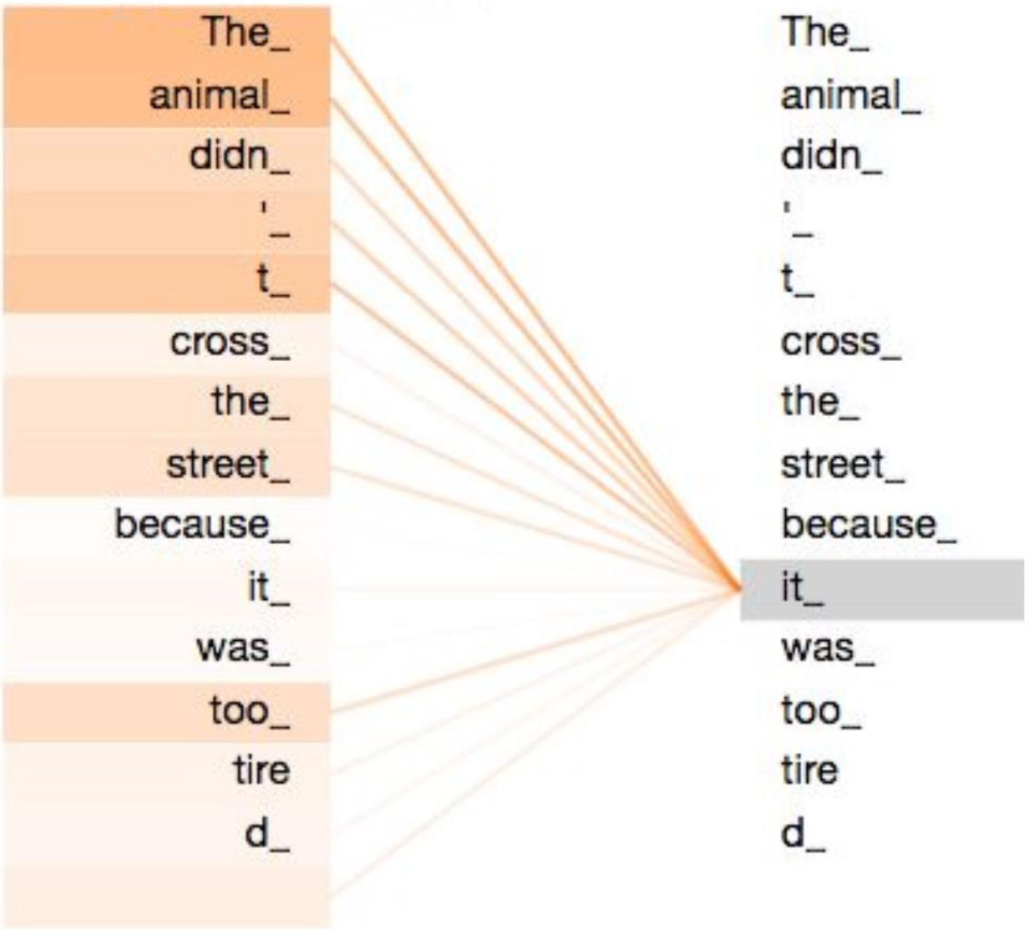
- Self-Attention layer is applied to each word individually.






Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

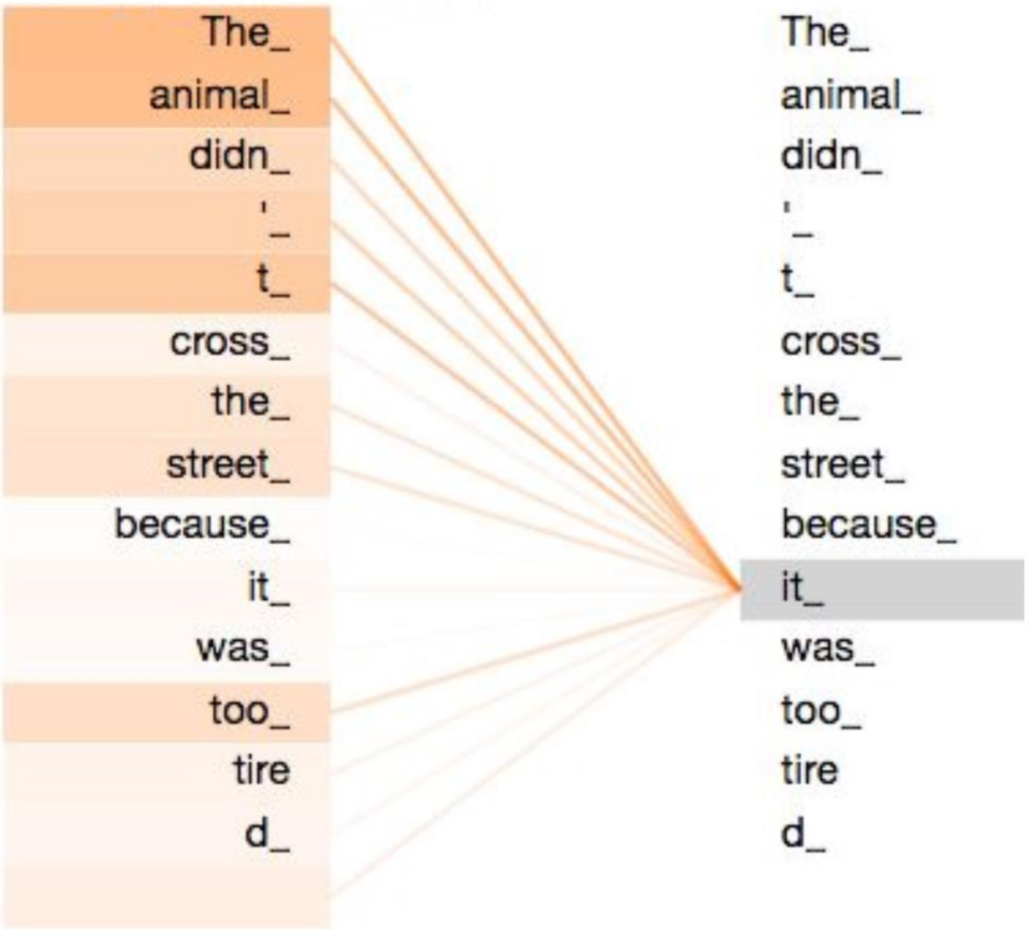
# Self-Attention: Input’s attention on itself




Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Self-Attention: Input’s attention on itself

What do we do next?



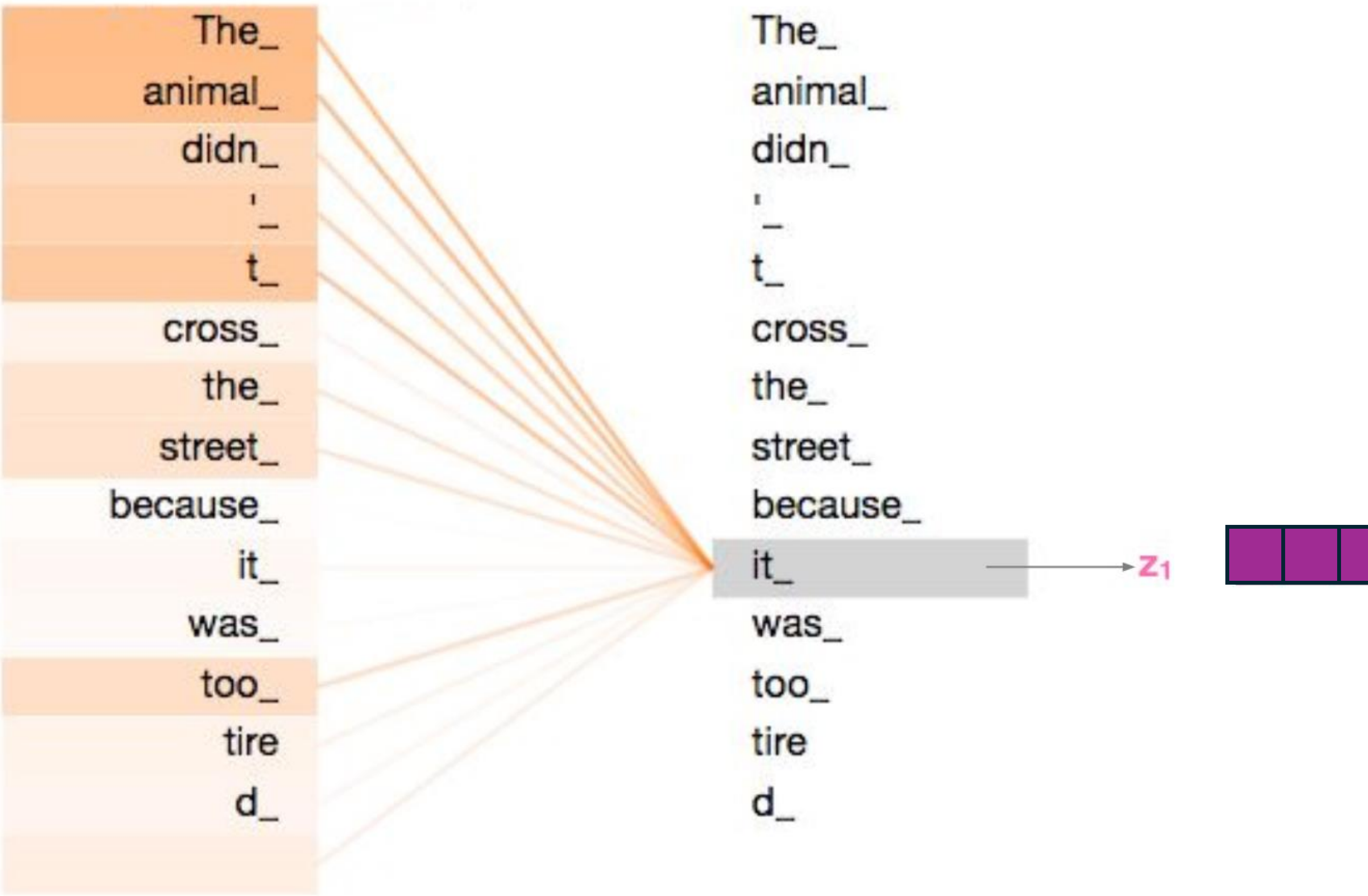

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Self-Attention: Overview

- **The big idea:**
  Self-attention computes the output vector $z_i$ for each word via a weighted sum of vectors extracted from each word in the input sentence

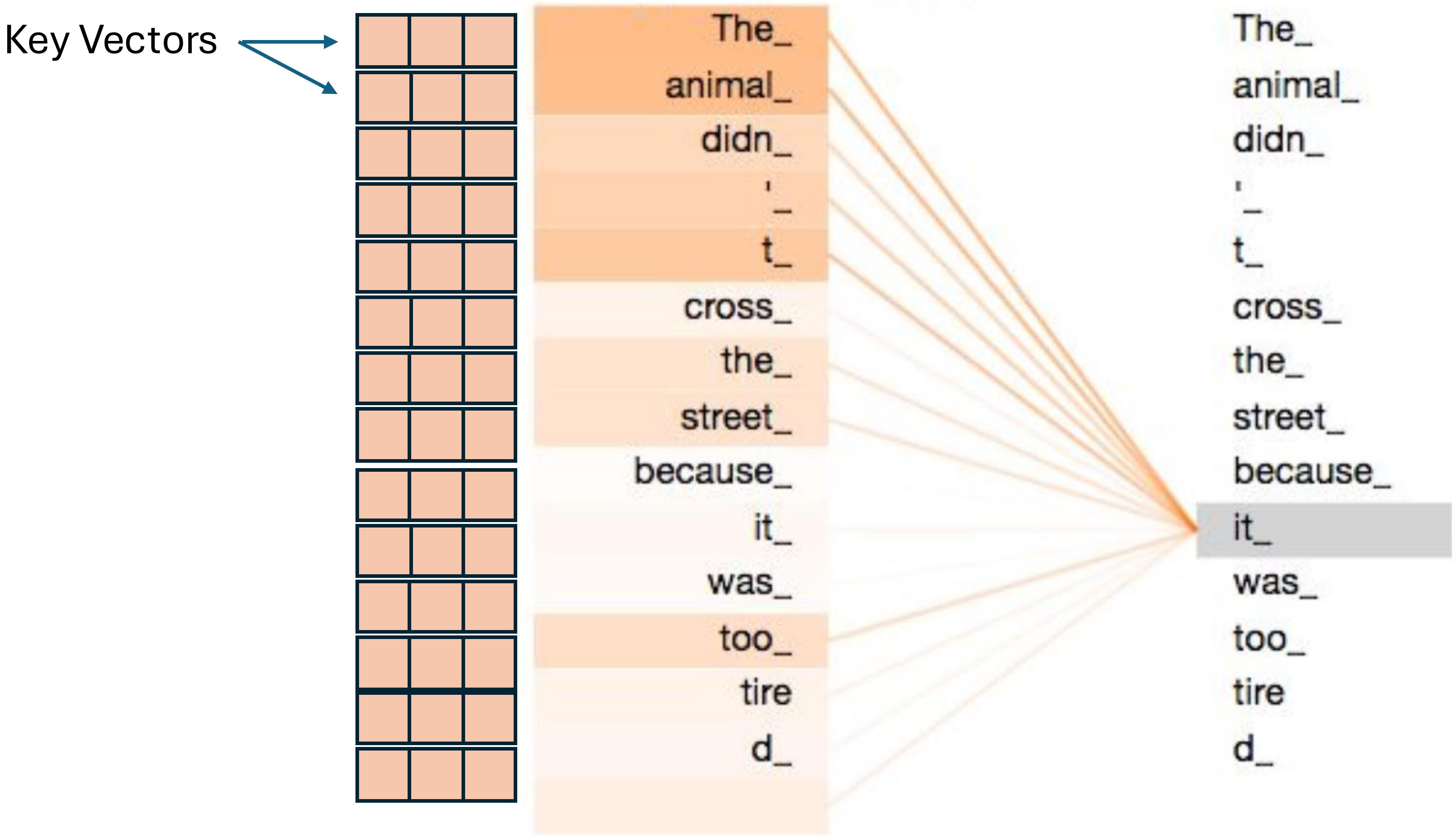
- Here, self-attention learns that "it" should pay attention to "the animal" (i.e. the entity that "it" refers to)

- Why the name ***self***-attention?
  This describes attention that the input sentence pays to itself



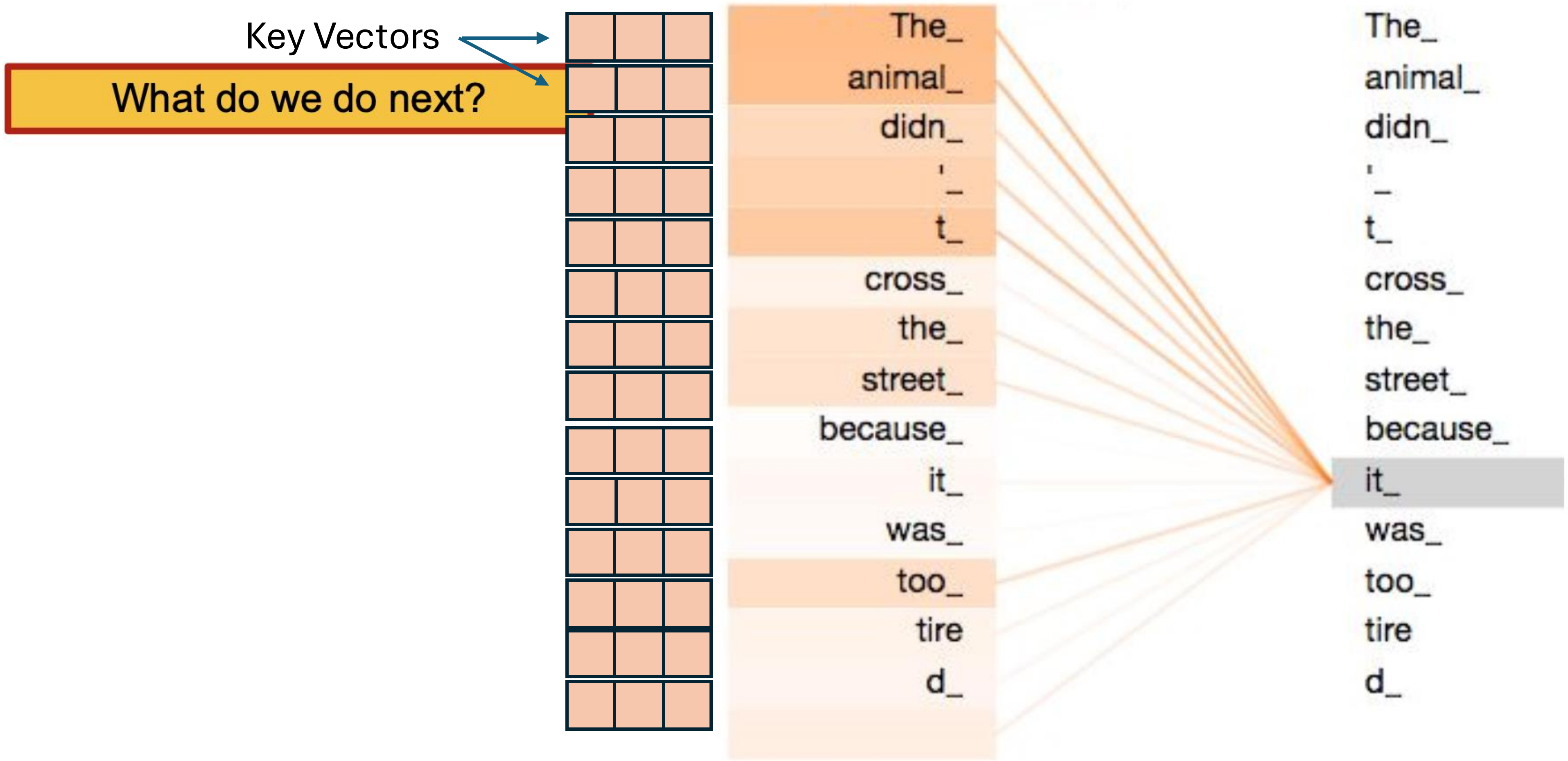

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Self-Attention: Input's attention on itself




Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

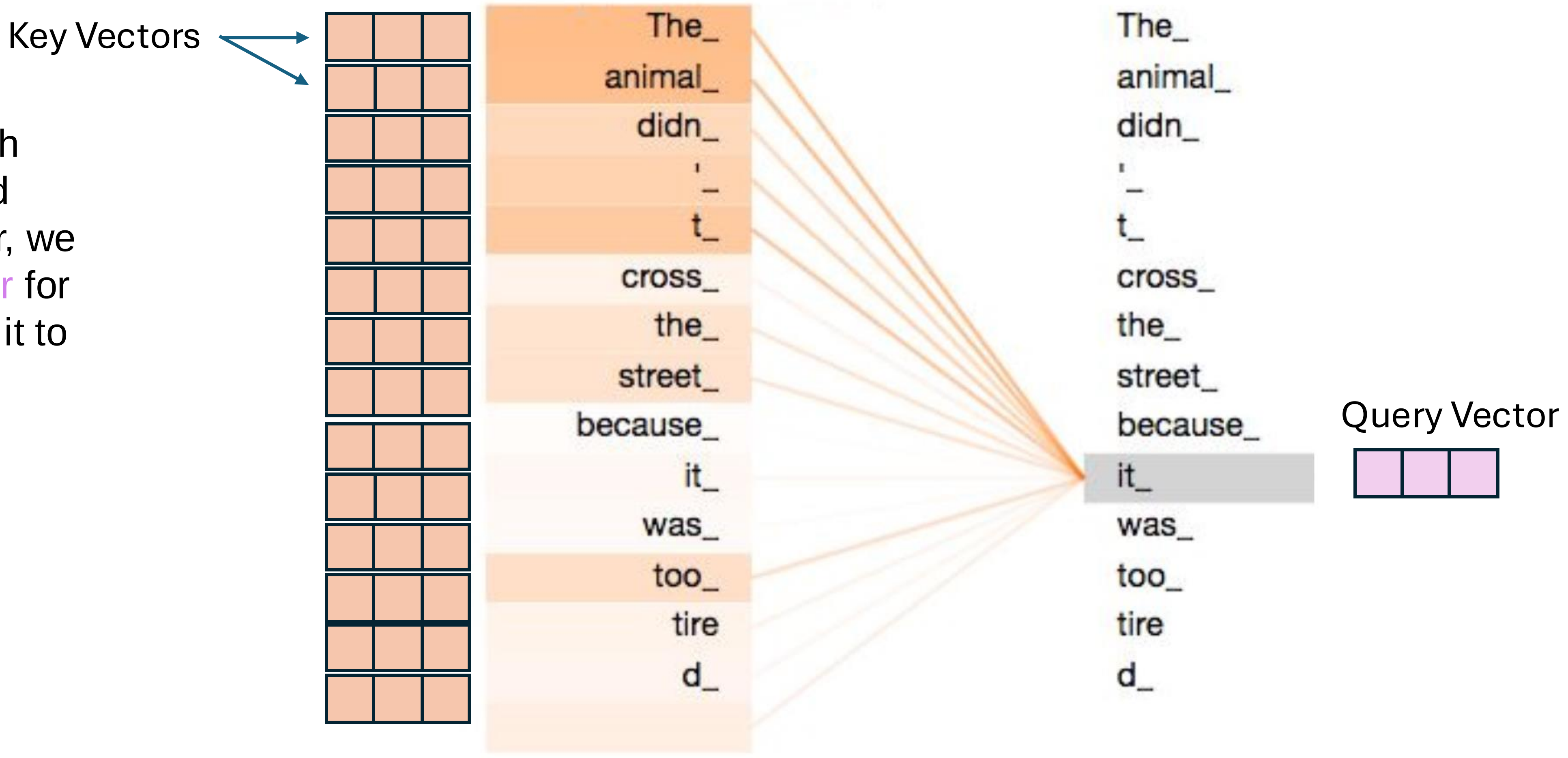
# Self-Attention: Input's attention on itself




Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Self-Attention: Input's attention on itself

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...



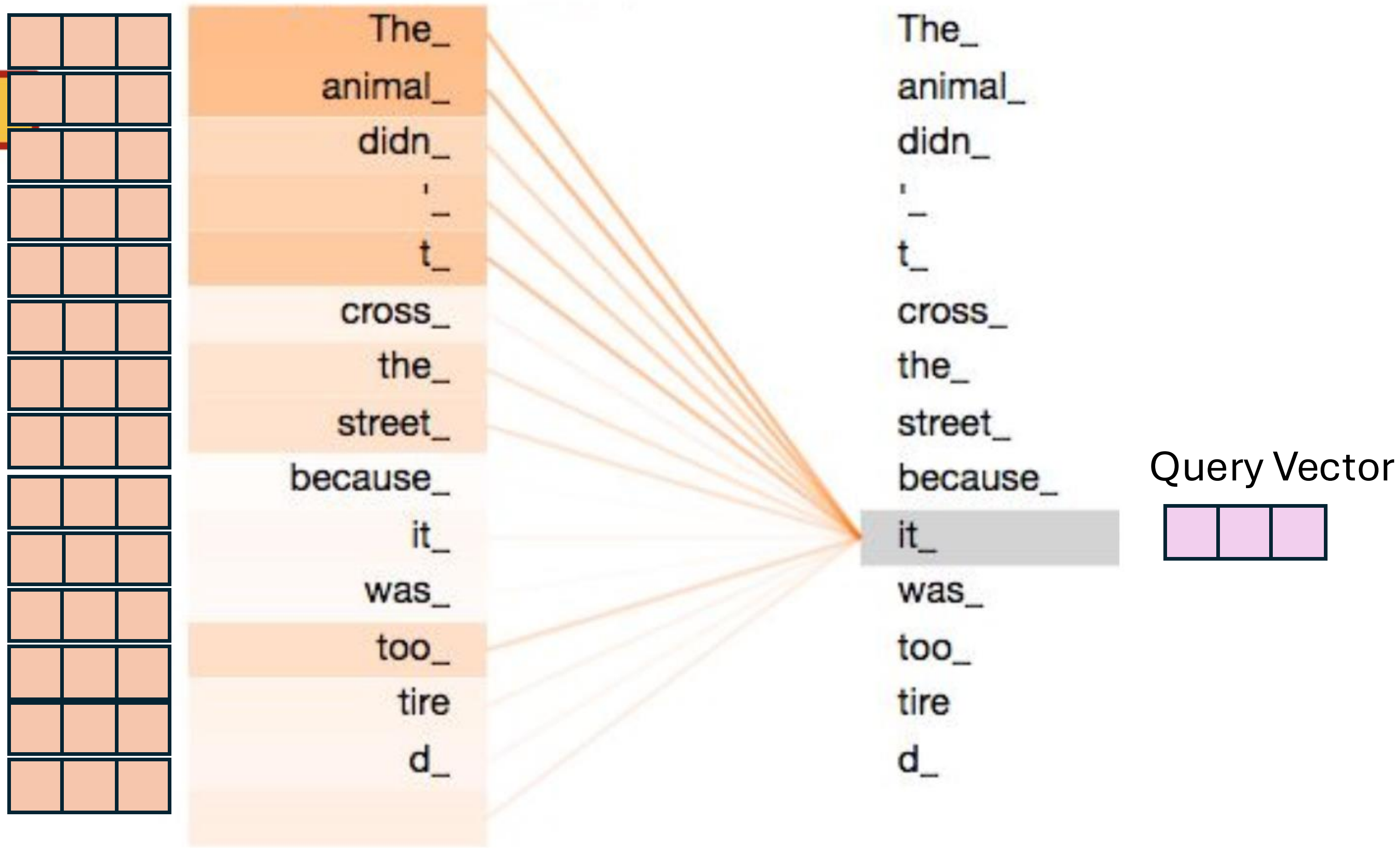

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Self-Attention: Input’s attention on itself

Key Vectors

**What do we do next?**

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

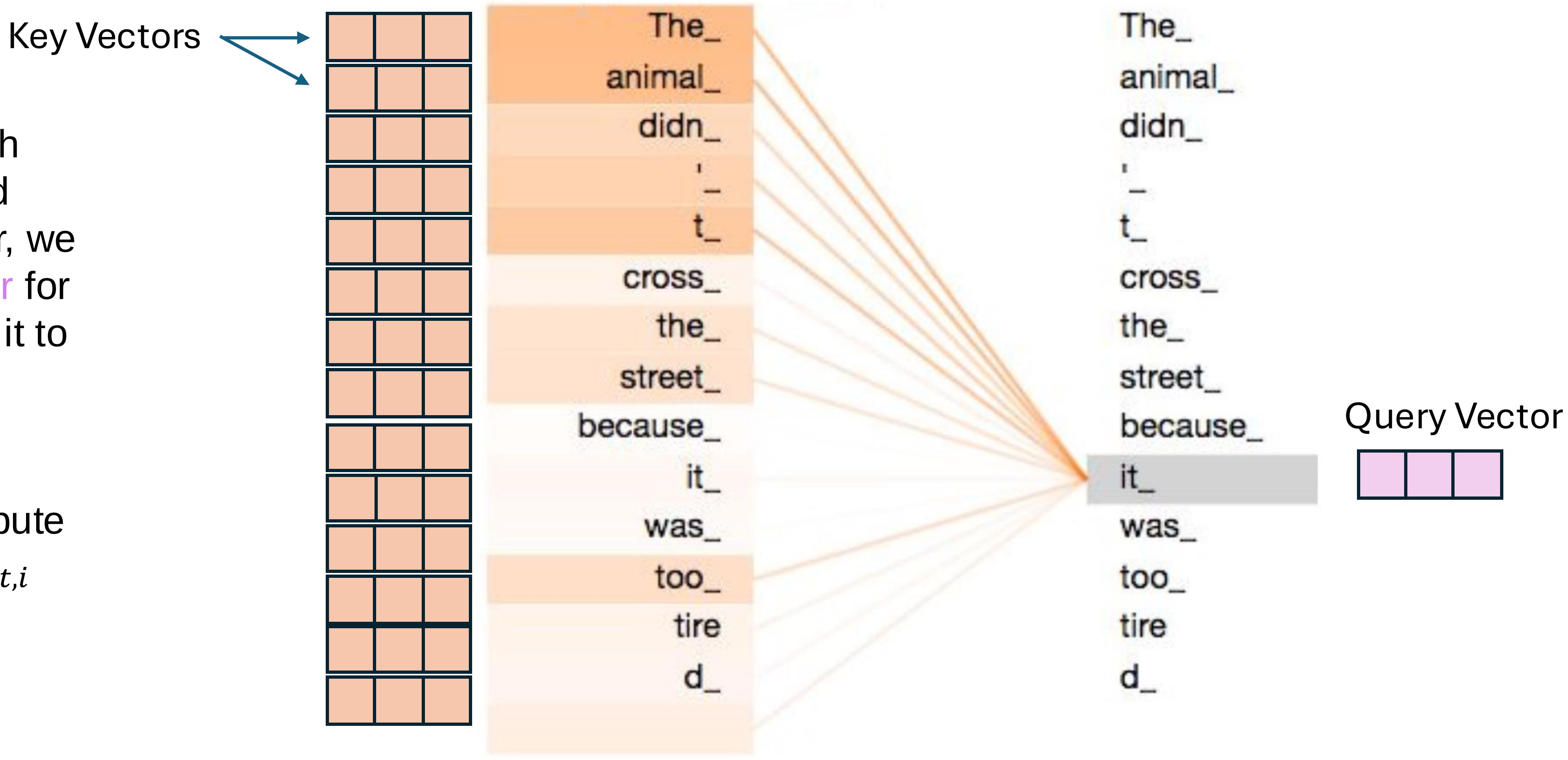



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Self-Attention: Input's attention on itself

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

Which use use to compute the alignment scores $a_{t,i}$






Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

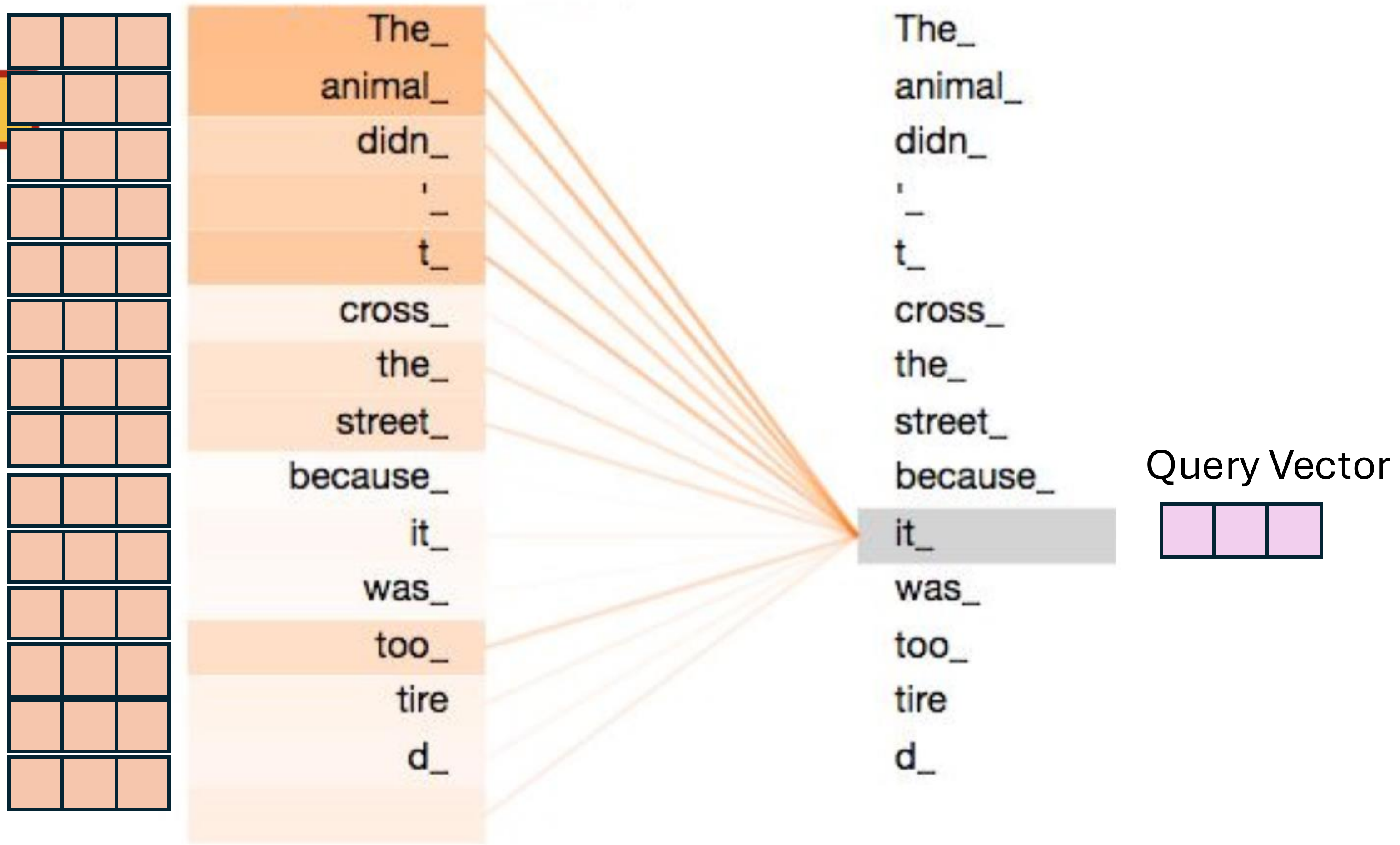
# Self-Attention: Input's attention on itself

Key Vectors

What do we do next?

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

Which use use to compute the alignment scores $a_{t,i}$



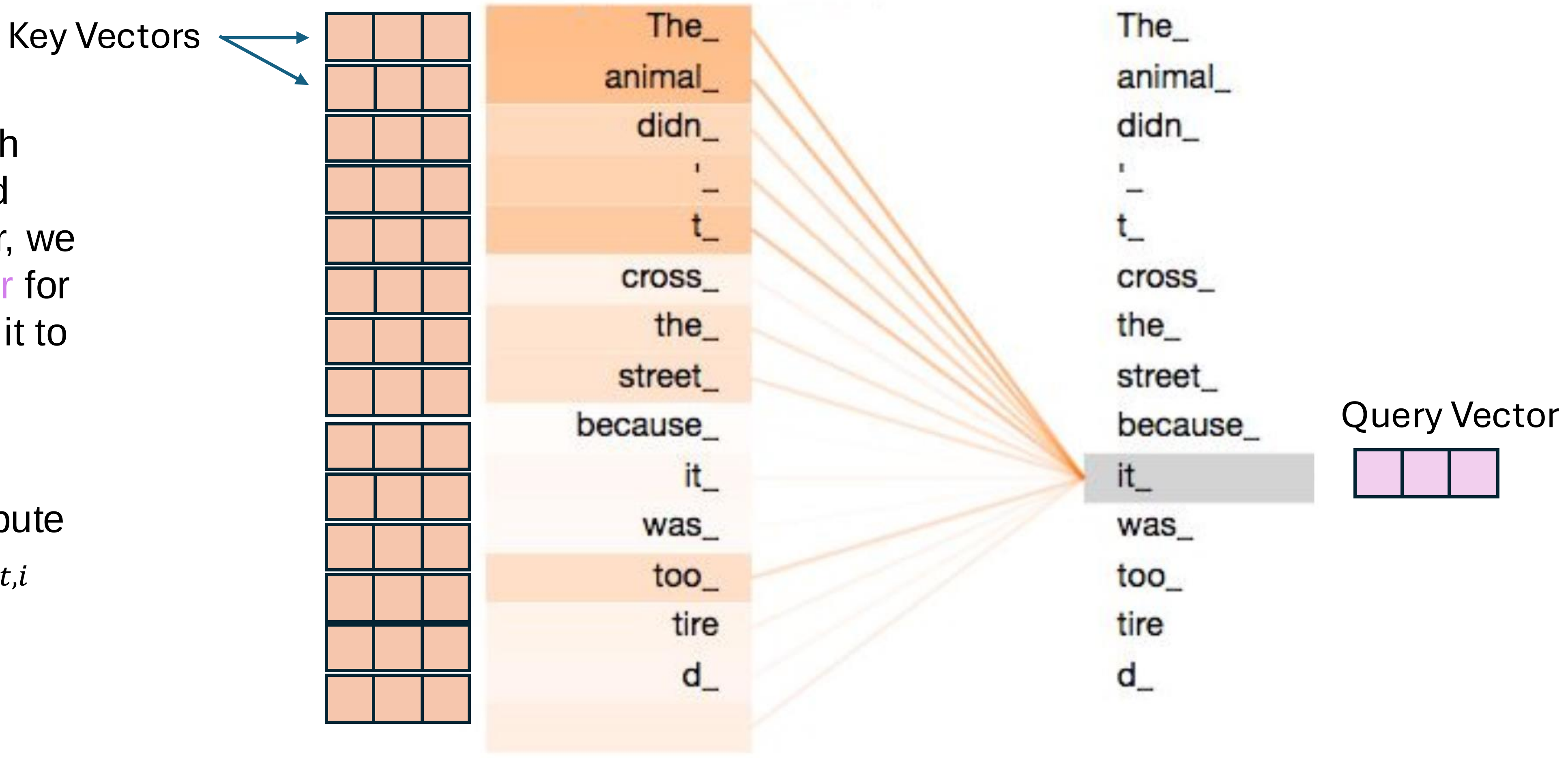



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Self-Attention: Input's attention on itself

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

Which use use to compute the alignment scores $a_{t,i}$




Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

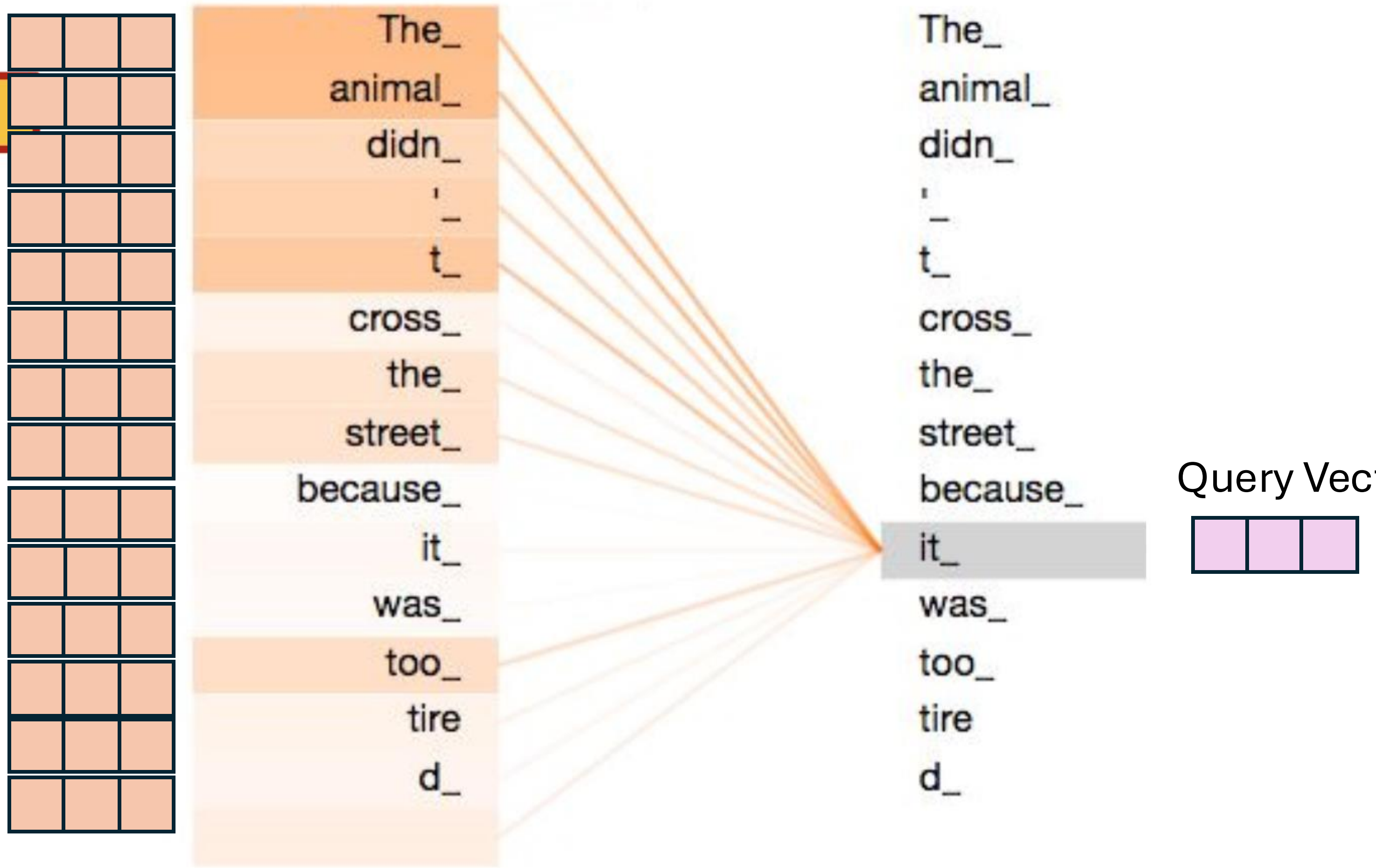
# Self-Attention: Input's attention on itself

Key Vectors

What do we do next?

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

Which use use to compute the alignment scores $a_{t,i}$




Query Vector

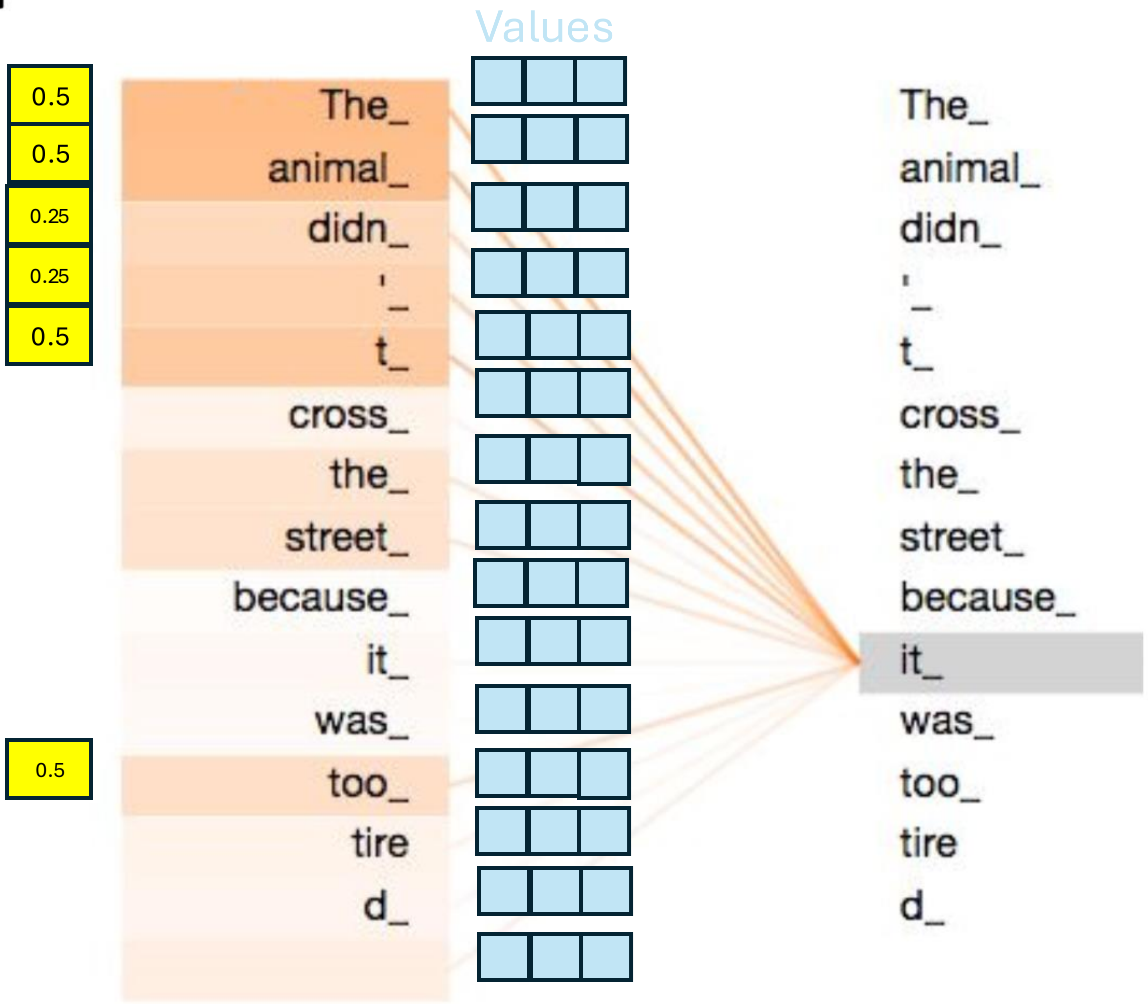


Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Self-Attention: Input's attention on itself

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

Which use use to compute the alignment scores $a_{t,i}$



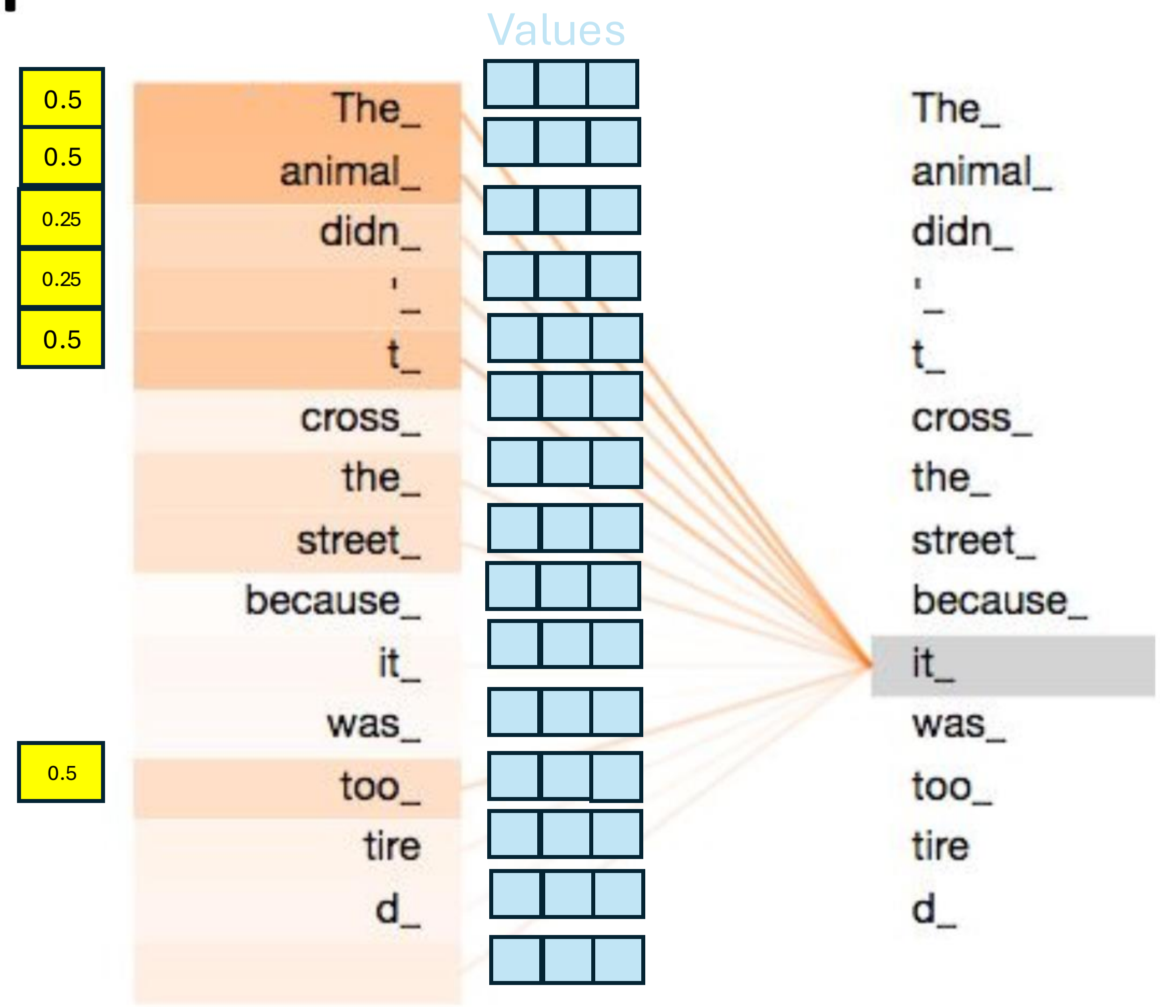

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Self-Attention: Input's attention on itself

What do we do next?

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

Which use use to compute the alignment scores $a_{t,i}$





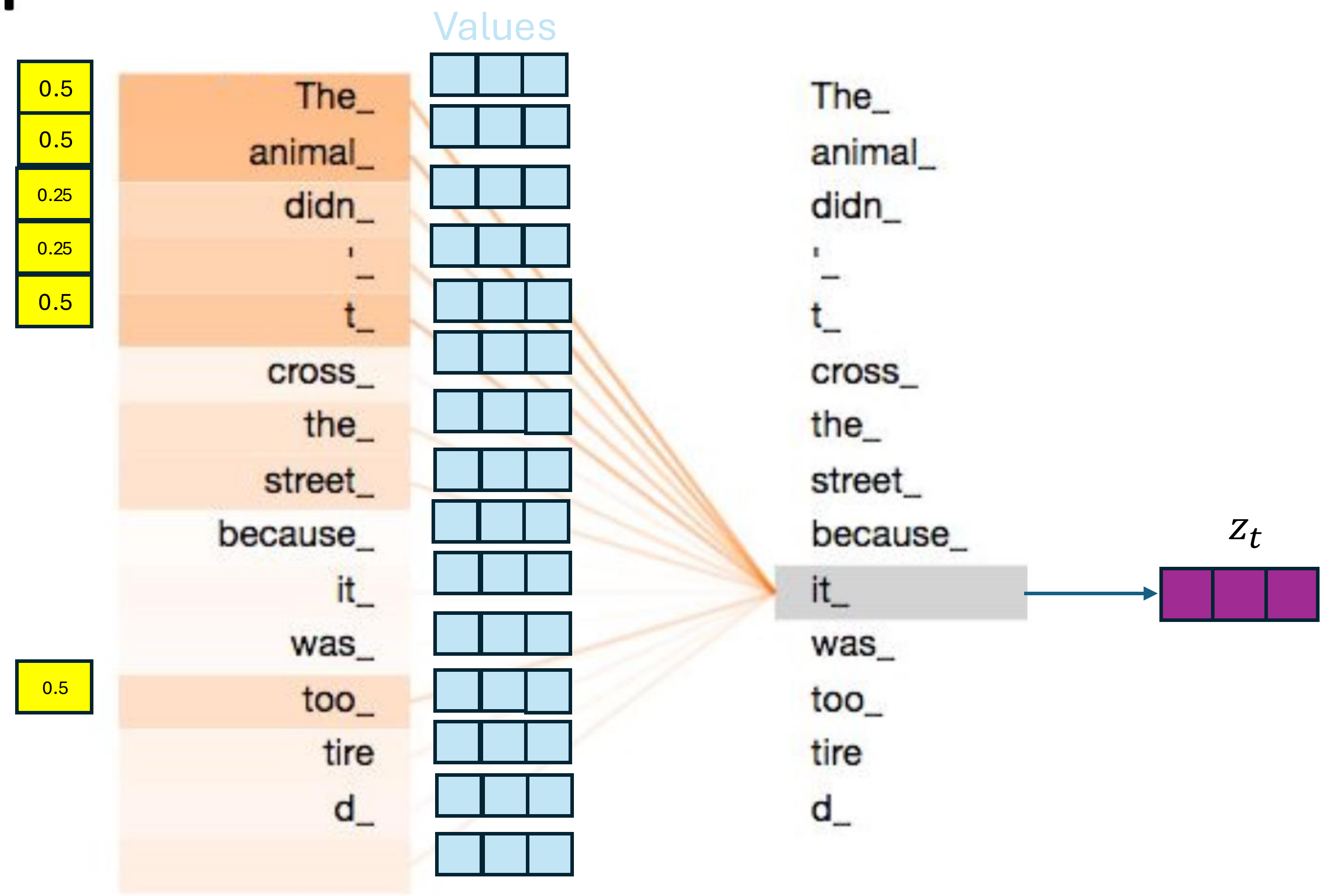

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Self-Attention: Input’s attention on itself

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

Which use use to compute the alignment scores $a_{t,i}$

To produce the output vector, we sum up the value vectors for each word, weighted by the score we computed in step 1






Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

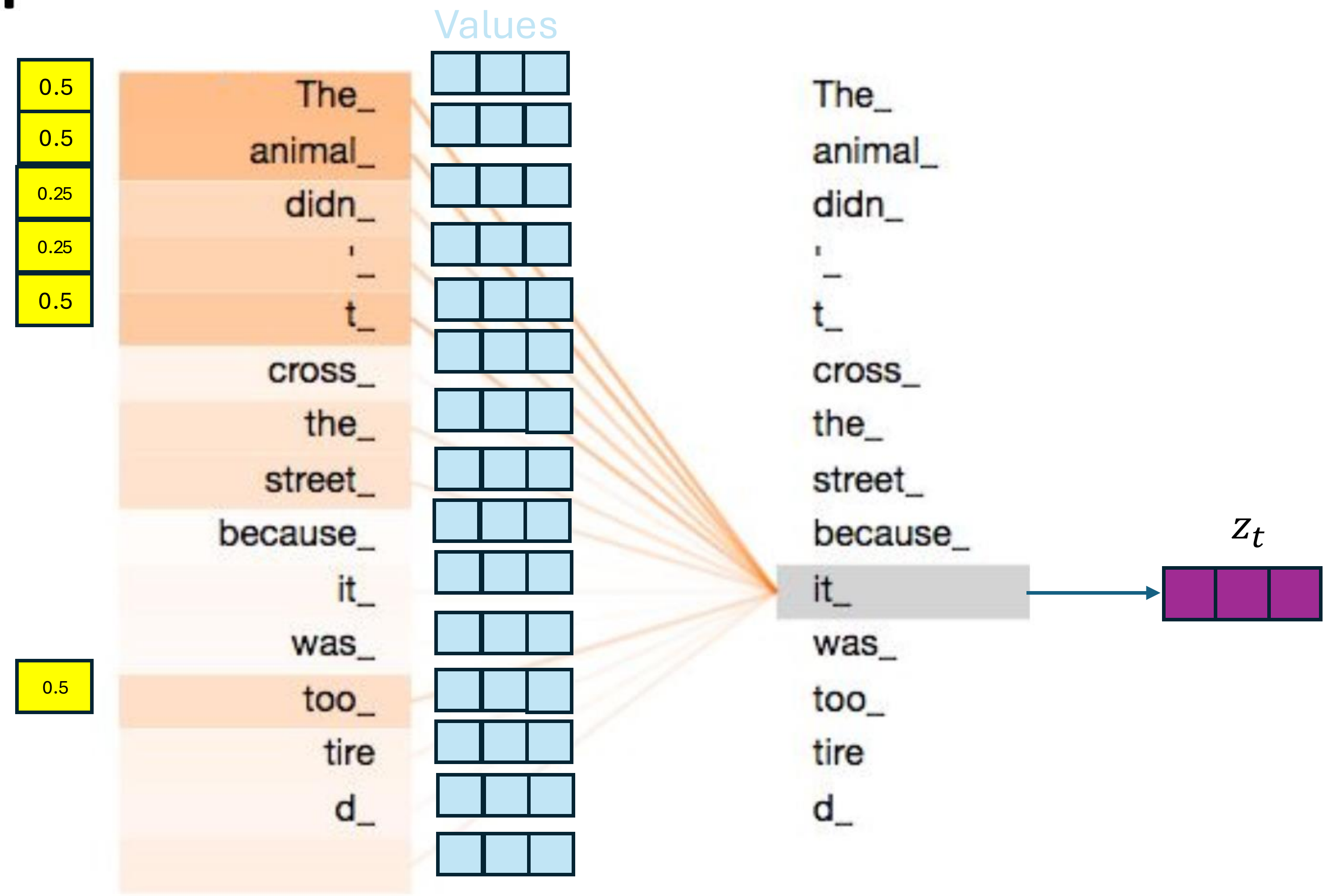
# Self-Attention: Input's attention on itself

What do we do next?

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

Which use use to compute the alignment scores $a_{t,i}$

To produce the output vector, we sum up the value vectors for each word, weighted by the score we computed in step 1

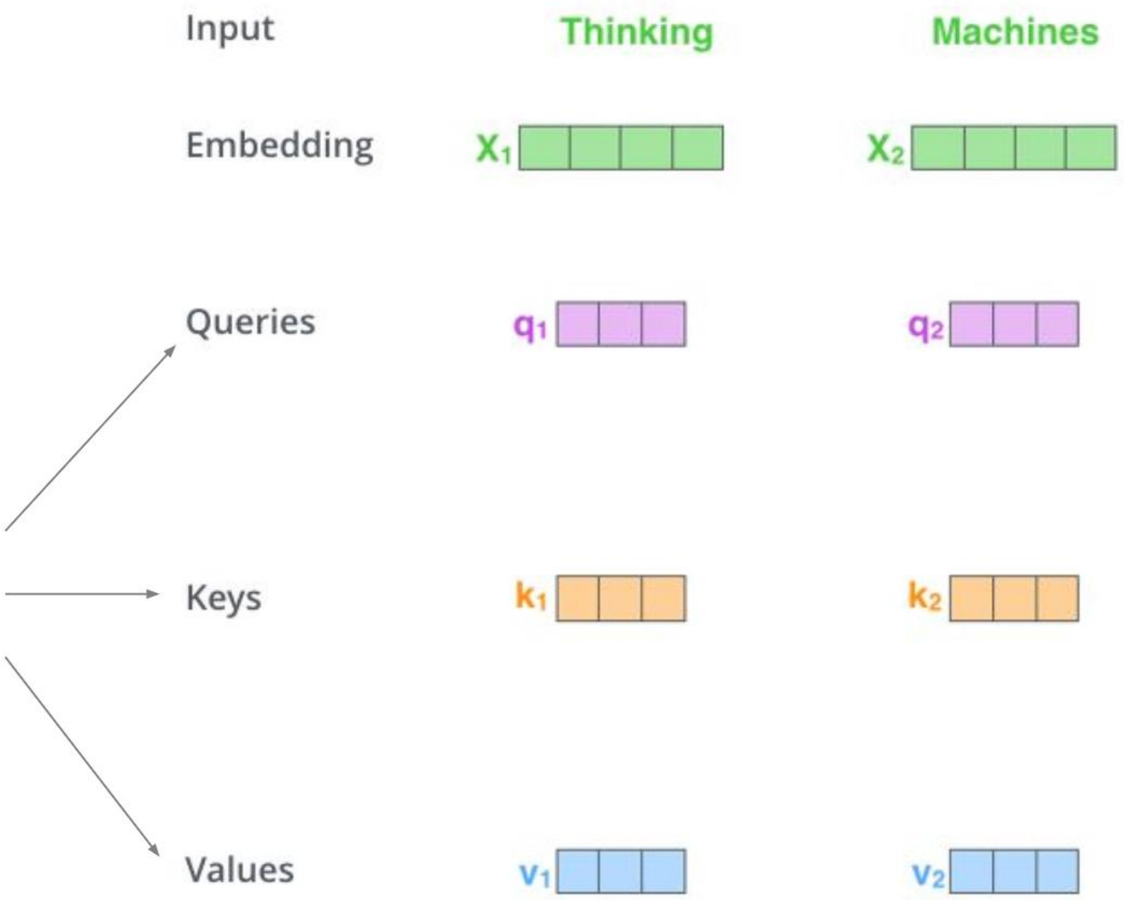





Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

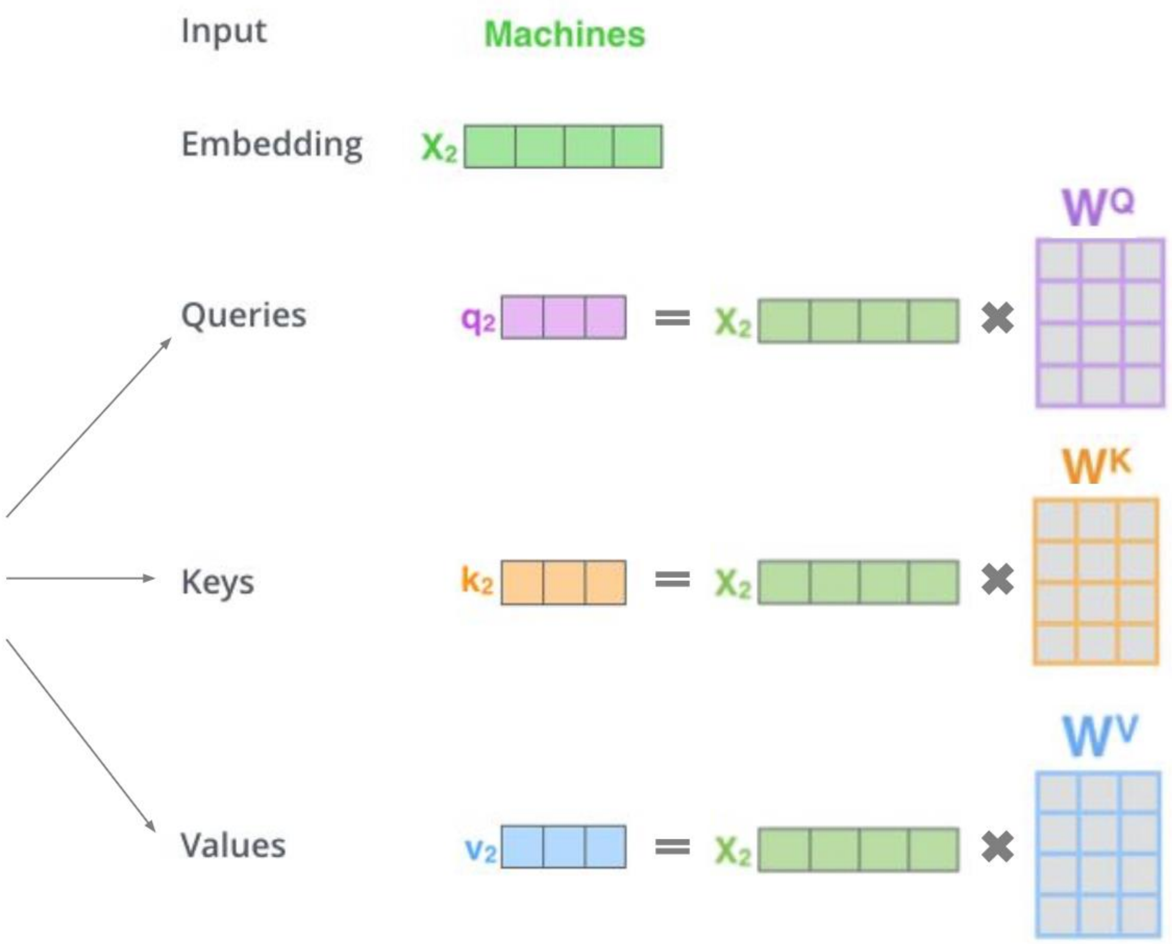
# Self-Attention: Details

Extract 3 vectors from each word embedding




Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

# Self-Attention: Details




Extract 3 vectors from each word embedding

Each vector is obtained by multiplying the embedding with the respective weight matrix.

How do we get these weight matrices?

These matrices are the ***trainable parameters*** of the network

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/