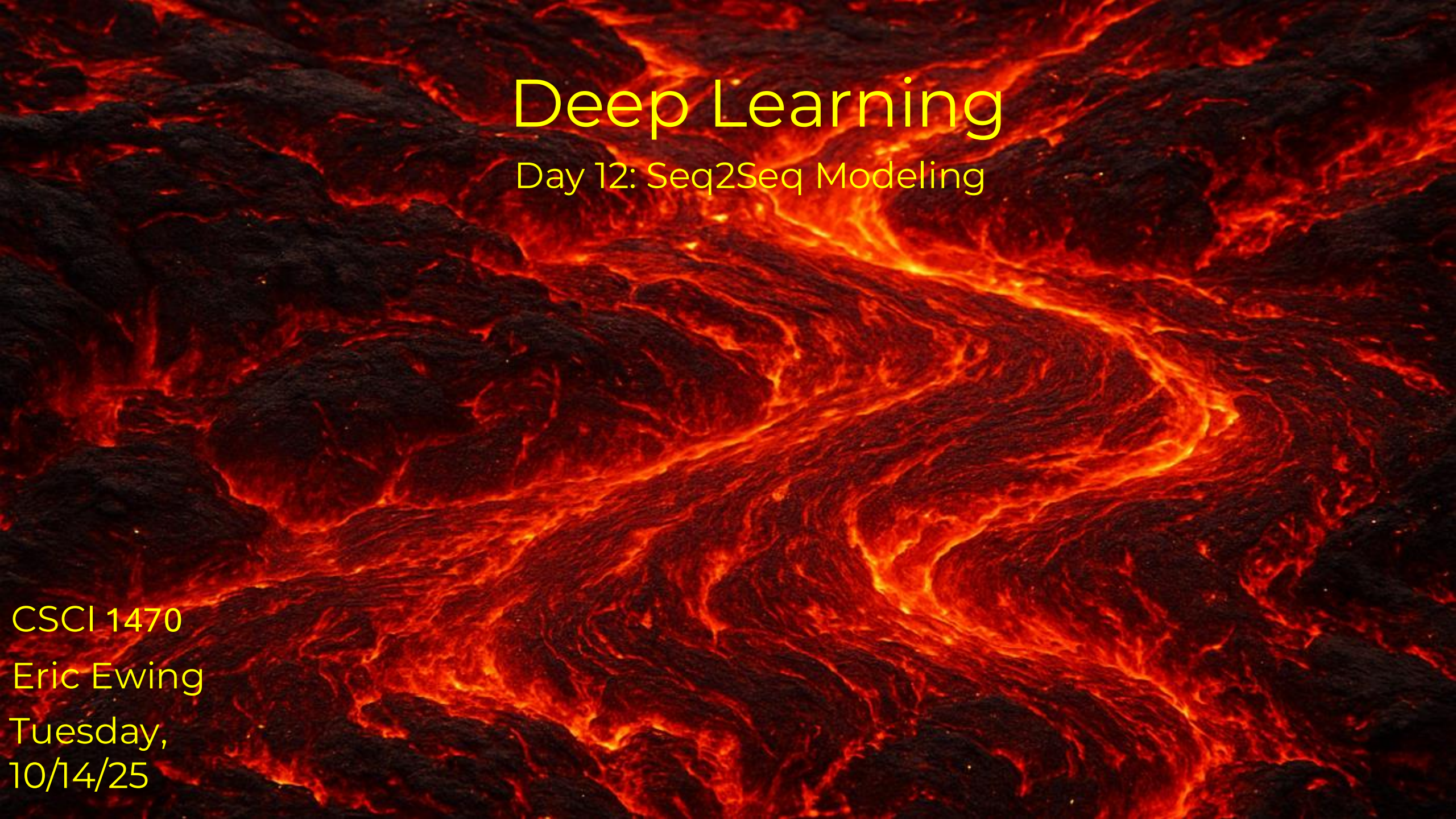
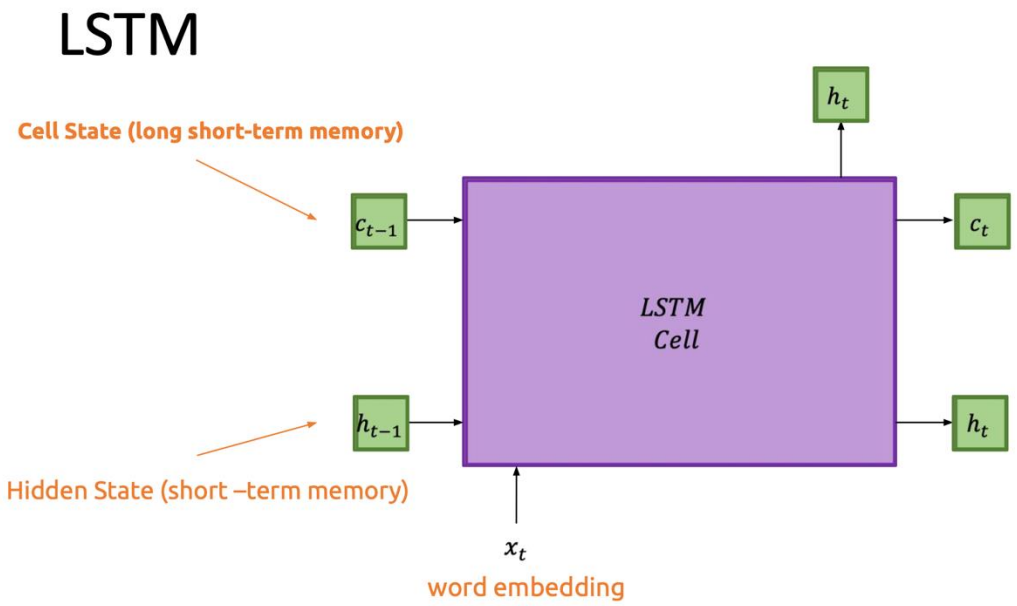
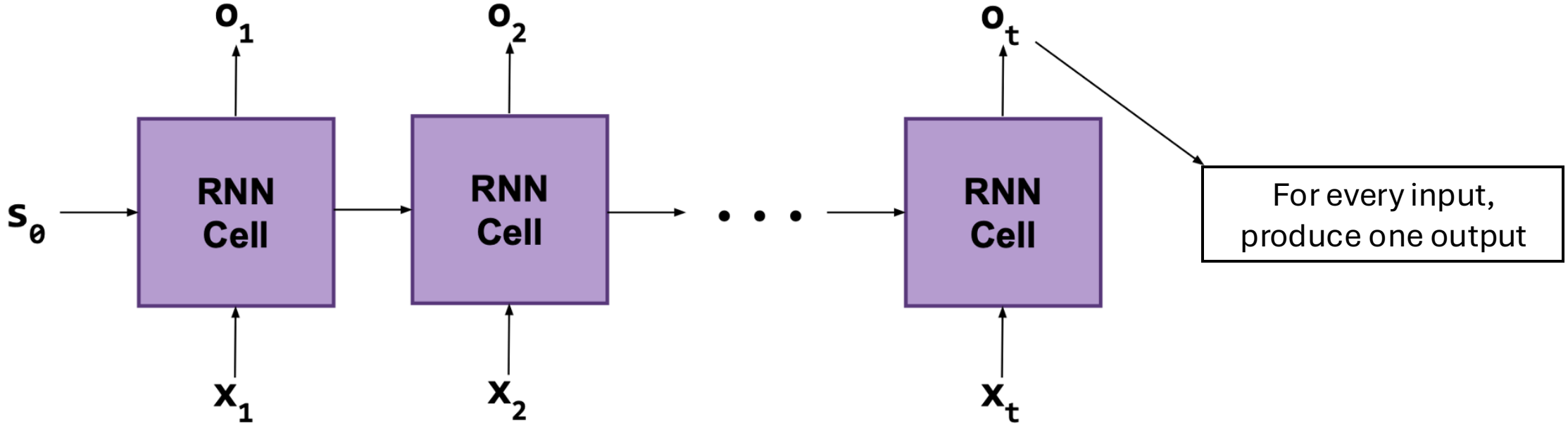

Deep Learning
Day 12: Seq2Seq Modeling
CSCI 1470
Eric Ewing
Tuesday,
10/14/25

RNN Recap

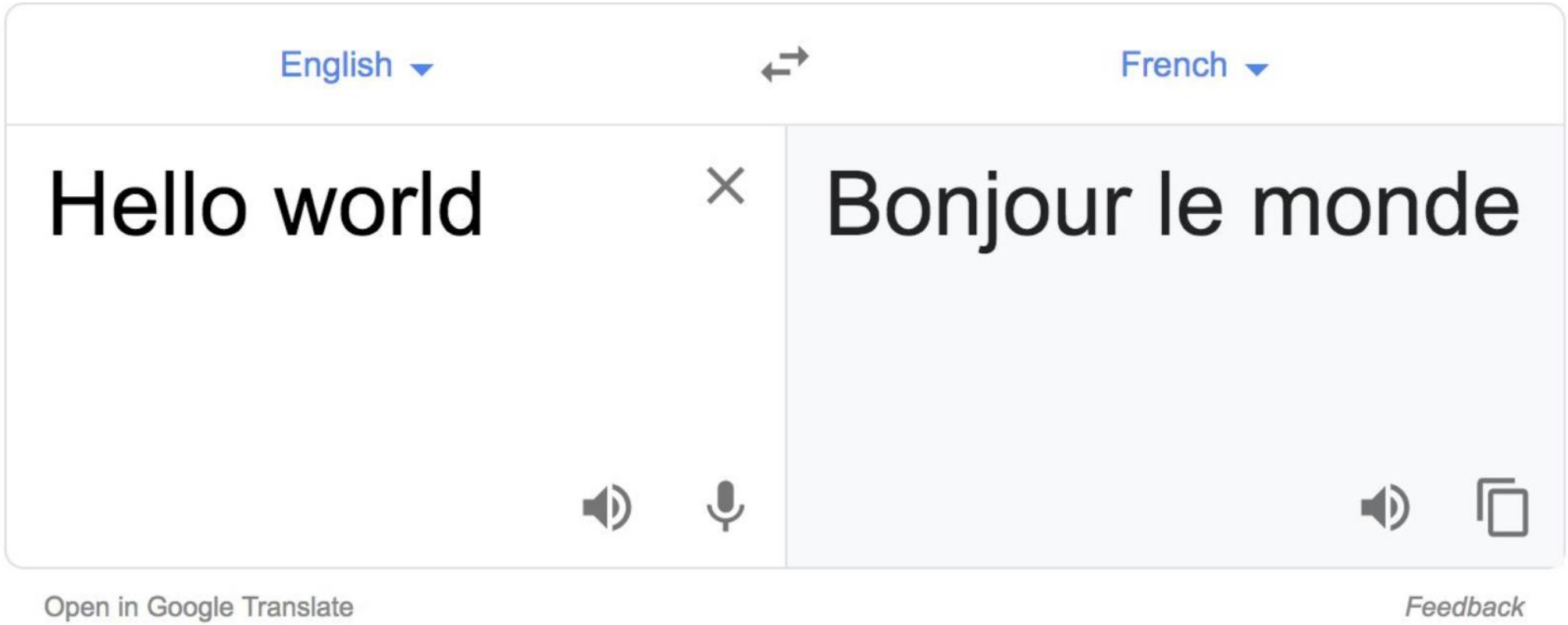




What if we don't know how many outputs there are?

Machine Translation

Software that translates one language to another



Why is this an interesting problem to solve?

- **Complex:** languages evolve rapidly and don't have a clear and well-defined structure
  - Example of language change: "awful" originally meant "full of awe", but is now strictly negative

- **Important**: billions per year spent on translation services
  - >CA$2.4 billion spent per year by Canadian government
  - >£100 million spent per year by UK government

Parallel Corpora

- We need pairs of equivalent sentences in two languages, called ***parallel corpora***

Canadian Hansards

- Hansards are transcripts of parliamentary debates
- Canada's official languages are English and French, so everything said in parliament is transcribed in both languages

Canadian Hansards: Examples

English	French
What a past to celebrate.	Nous avons un beau passé à célébrer.
We are about to embark on a new era in health research in this country.	Le Canada est sur le point d'entrer dans une nouvelle ère en matière de recherche sur la santé.

Canadian Hansards

- We can use this as a dataset for MT!
- Not perfect:
  - *Translations aren't literal*: in the example, "this country" is translated to "Le Canada"
  - *Biased in style*: not everyone speaks like politicians in parliamentary debate
  - *Biased in content*: some topics are never discussed in parliament

Other parallel corpora

- Europarl, a parallel corpus of 21 languages used in the European Parliament
- EUR-Lex, a parallel corpus of 24 languages used in EU law and public documents
- Japanese-English Bilingual Corpus of Wikipedia's Kyoto Articles

Any questions?

Problems with parallel corpora

- Expensive to produce
- Tend to be biased towards particular types of text – e.g. government documents containing formal language
- Translations aren't necessarily literal - e.g. "this country" -> "Le Canada"
- Parallel corpora are necessary, **but never perfect**

LM approach

- Language modelling works on a word-by-word basis, taking only previous words as input

$$P(w_{t,i}) = P(w_{t,i} \mid w_{s,i-1}, w_{s,i-2}, \dots, w_{s,0})$$

- Where $w_{t,i}$ is the i^{th} word in the target sentence, and $w_{s,i}$ is the i^{th} word in the source sentence

Will it work for MT task?

Why our LM approach doesn't work for MT

- Language modelling works on a word-by-word basis, taking only previous words as input

$$P(w_{t,i}) = P(w_{t,i} \mid w_{s,i-1}, w_{s,i-2}, \dots, w_{s,0})$$

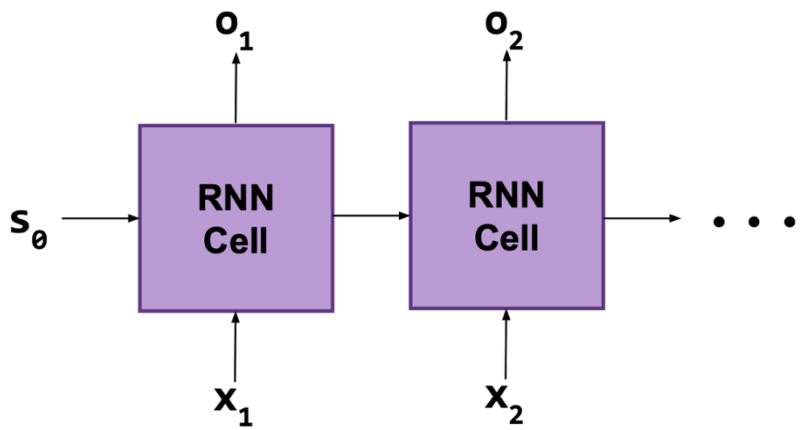
- Where $w_{t,i}$ is the i^{th} word in the target sentence, and $w_{s,i}$ is the i^{th} word in the source sentence
- However, **it is not a given that the information we need comes in the preceding words**
- The order and length of the source and target sentences are not necessarily equal

Example from Hansards

- For example, take the first entry in Hansard's:

edited hansard number 1

hansard révisé numéro 1



What should we do?

Further examples

French: “Londres me manque”

Naive translation: “London I miss”

Correct translation: “I miss London”

French: “Je viens de partir”

Naive translation: “I come of to go”

Correct translation: “I just left”

Sequence to Sequence (seq2seq)

Thus, we cannot simply use the previous words – we need to ***summarize the source sentence first***

This is called sequence to ***sequence to sequence learning***, or ***seq2seq***

Sequence to Sequence (seq2seq)

- Instead of:

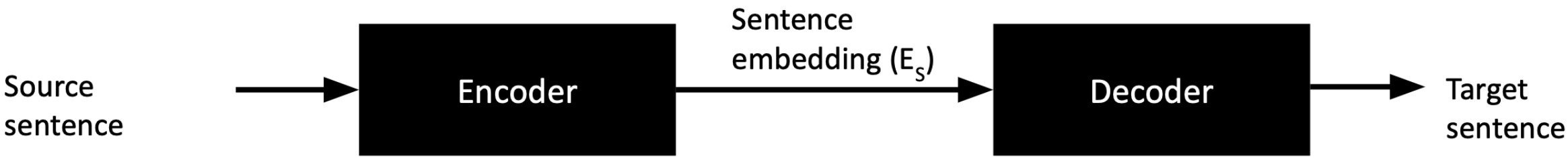
$$P(w_{i,t}) = P(w_{i,t} \mid w_{i-1,s}, w_{i-2,s}, \dots, w_{0,s})$$

Let's do:

$$P(w_{i,t}) = P(w_{i,t} \mid E_S, w_{i-1,t}, w_{i-2,t}, \dots, w_{0,t})$$

Where E_S is a summary, or ***embedding***, of the sentence taken from the source language, and w_i is the i^{th} word of the sentence in the target language

What will the neural net look like?

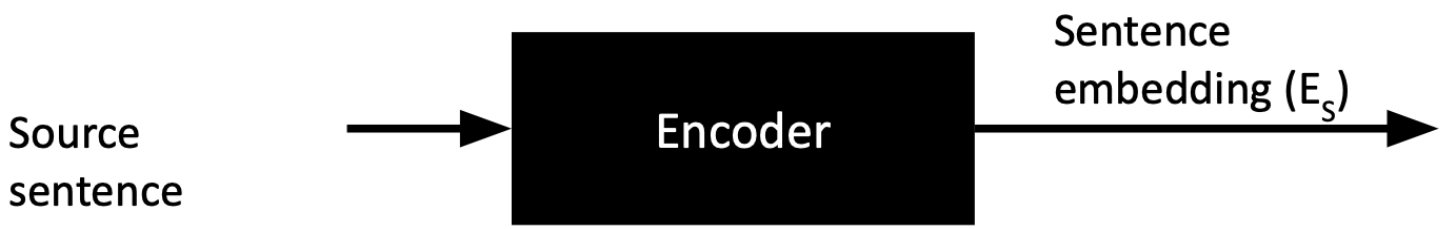


Origin of the encoder/decoder terminology: information theory

- The encoder “compresses” the source sentence into a compact “code”
- The decoder recovers the sentence (but in the target language) from this code

What will the neural net look like?

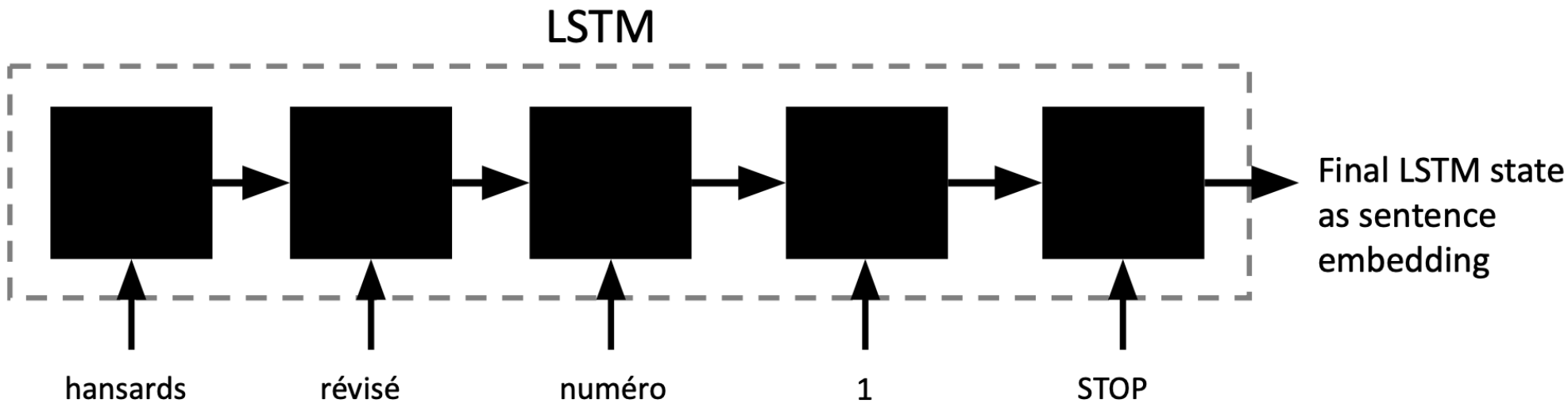
Any ideas?



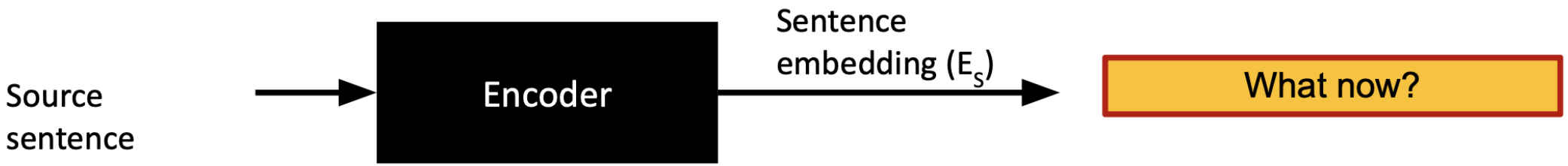
Encoder

- To generate the sentence embedding, we need an encoder
- Use an LSTM
- Feed in the source sentence
- Take the final LSTM state as the sentence embedding
- This will be a ***language-agnostic*** representation of the sentence
  - i.e. it will represent the *meaning* of the sentence without being tied to any particular language

Encoder architecture

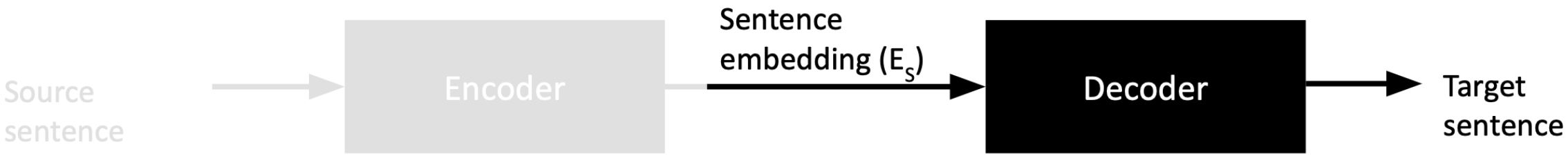


What will the neural net look like?



What will the neural net look like?

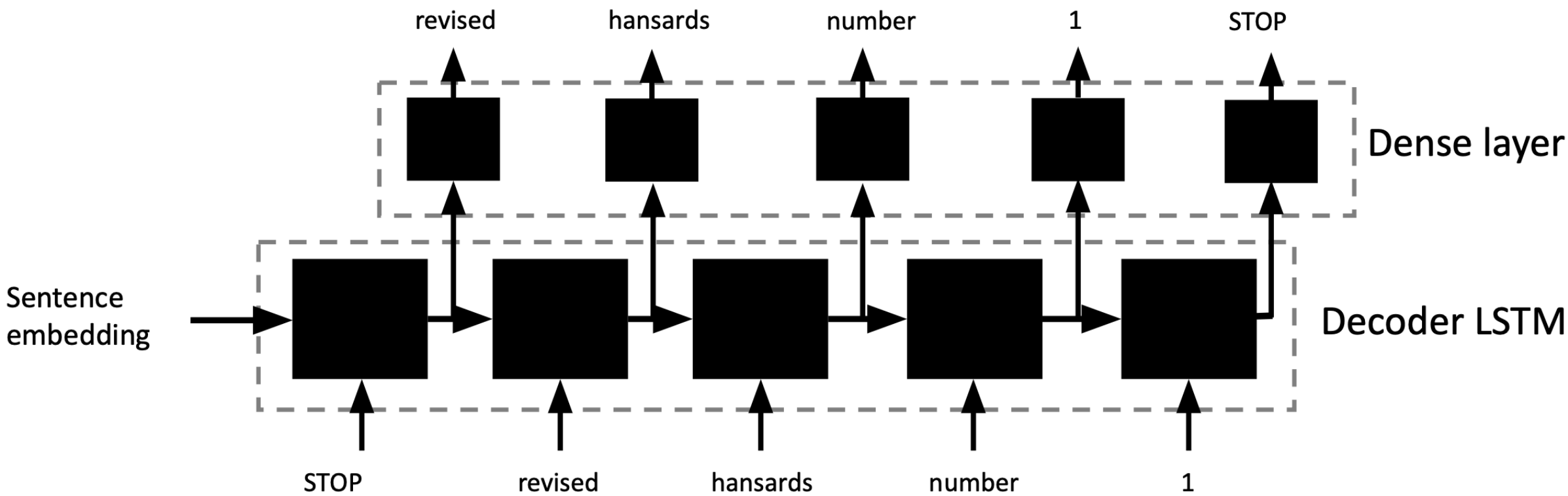
Any ideas?



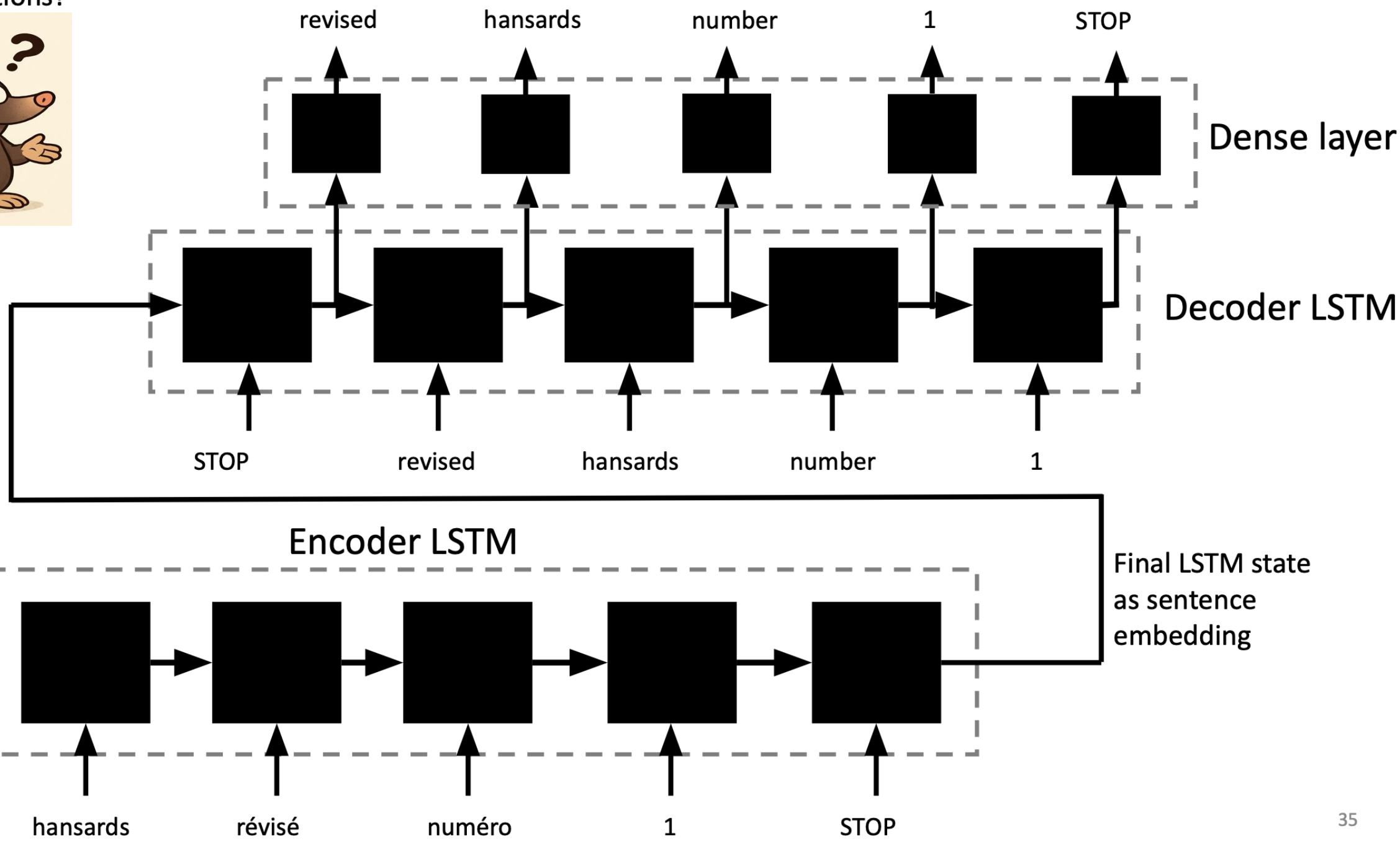
Decoder

- We now have a sentence embedding representing the meaning of the source sentence
- Now, let's generate a sentence in the target language with the same meaning
- Use an LSTM again, **with the sentence embedding** as its initial hidden state
- The rest is just like language modeling:
  - Input to the LSTM is the previous word from the target sentence
  - Take each LSTM output and put it through a fully connected layer
  - Softmax to convert to probability distribution over next word in target language

Decoder architecture



Any questions?



Any questions?

Do you think the embedding from the final RNN can encode enough information about the beginning of the sentence to accurately translate it?

Any questions?

Do you think the embedding from the final RNN can encode enough information about the beginning of the sentence to accurately translate it?

revised hansards number 1 STOP

Dense layer

Decoder LSTM

STOP revised hansards number

What if the decoder "forgets" the sentence embedding?

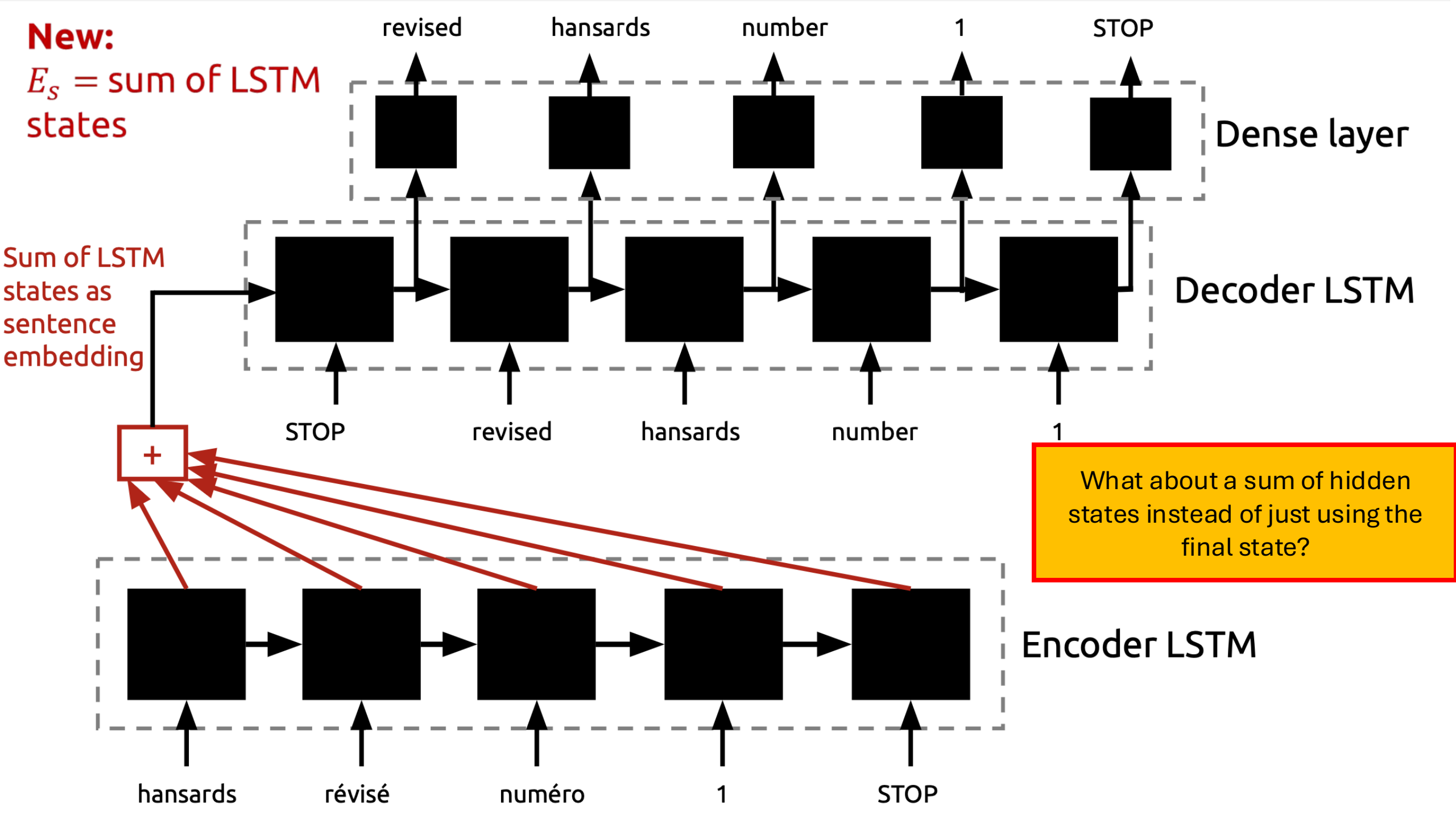
Encoder LSTM

Final LSTM state as sentence embedding

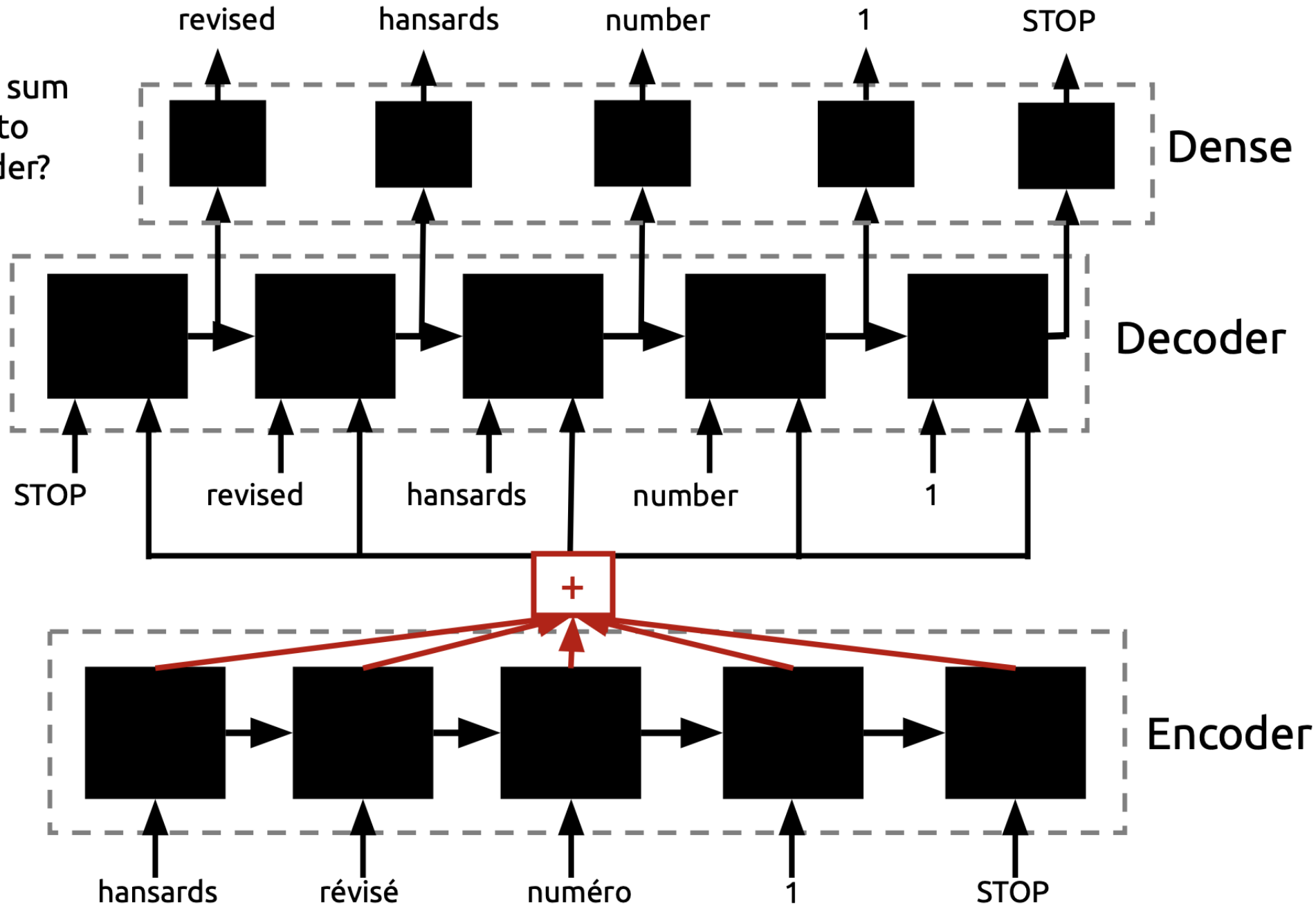
hansards révisé numéro 1 STOP

Issues with RNNs for Seq2Seq

- RNNs may “forget” the beginning of the sentence in the encoder
- RNNs (even LSTMs) may “forget” the embedding in the decoder


New:
E_s = sum of LSTM states
revised
hansards
number
1
STOP
Dense layer
Decoder LSTM
Sum of LSTM states as sentence embedding
+
STOP
revised
hansards
number
1
What about a sum of hidden states instead of just using the final state?
Encoder LSTM
hansards
révisé
numéro
1
STOP

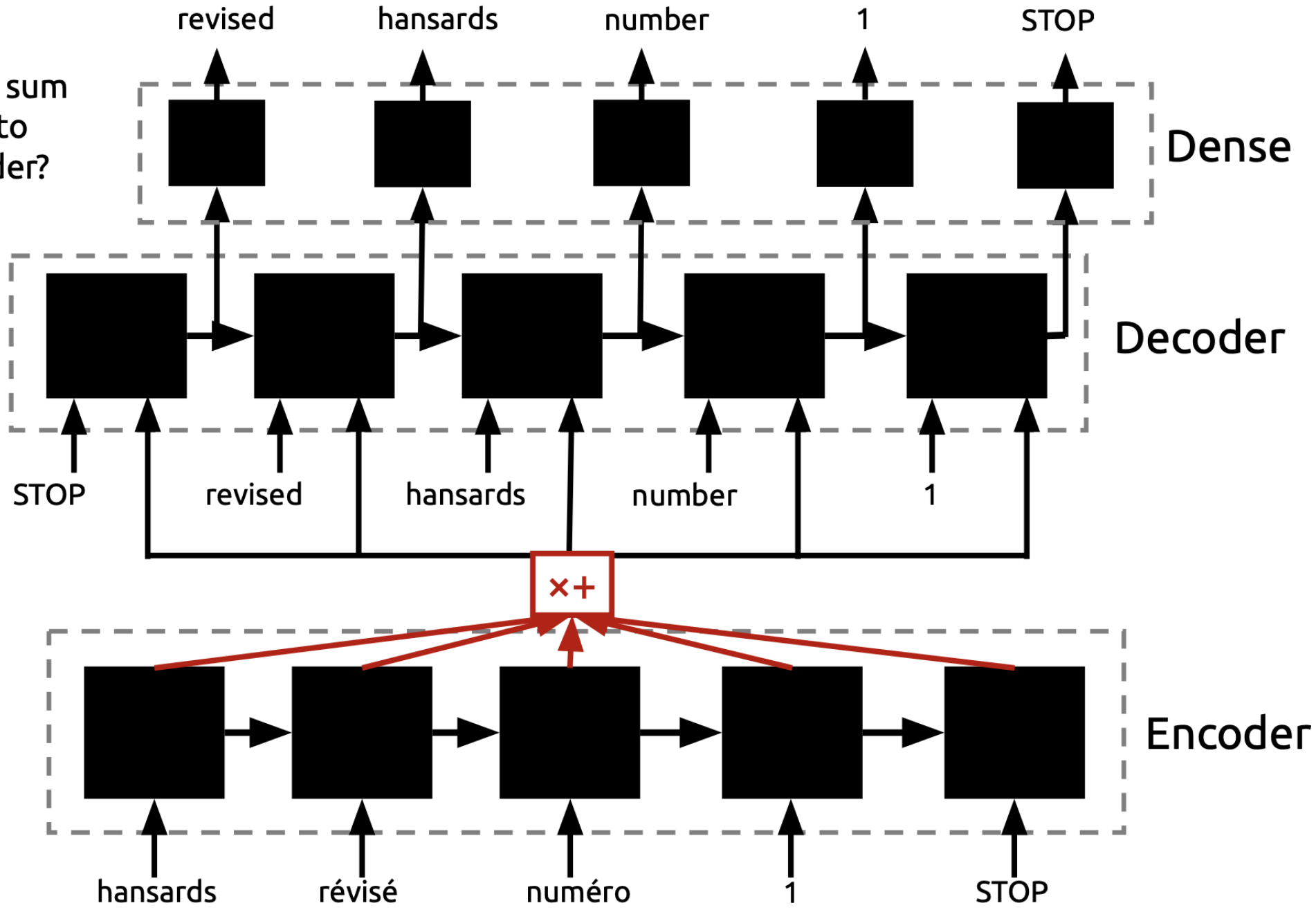
What if we passed the sum of our encoder states to ***every cell*** in the decoder?



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

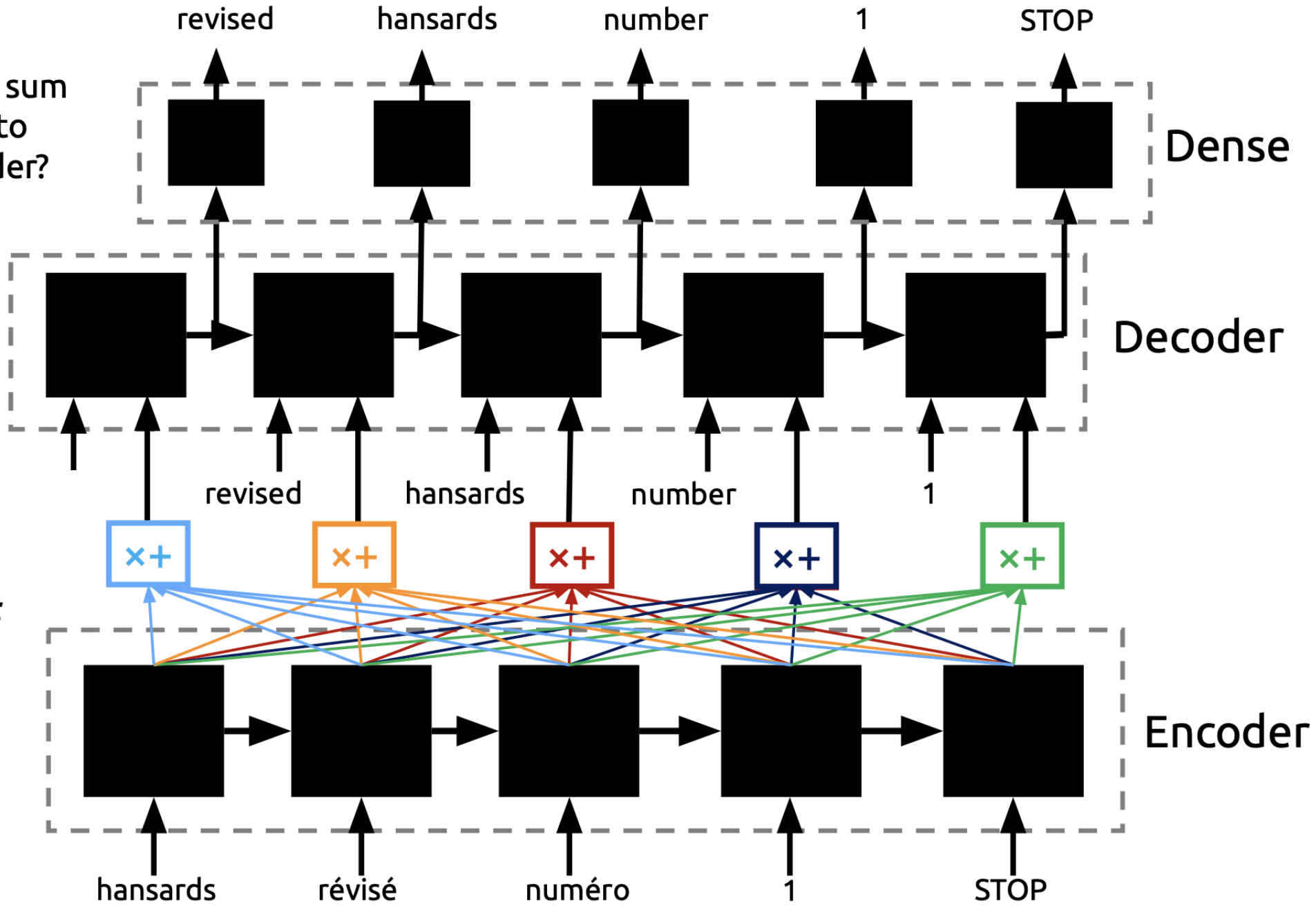


What if we passed the sum of our encoder states to ***every cell*** in the decoder?

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*



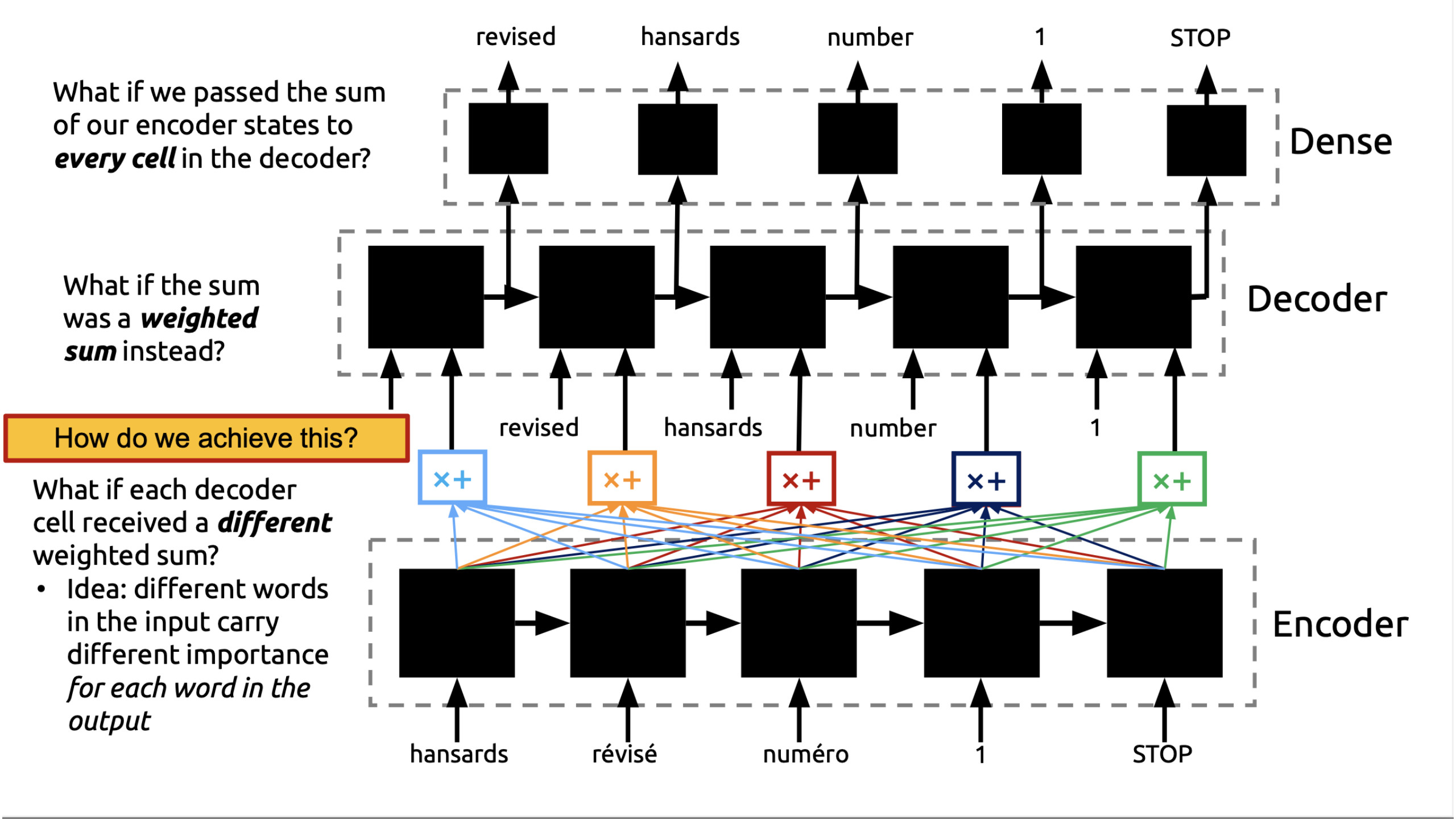
What if we passed the sum of our encoder states to ***every cell*** in the decoder?

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”

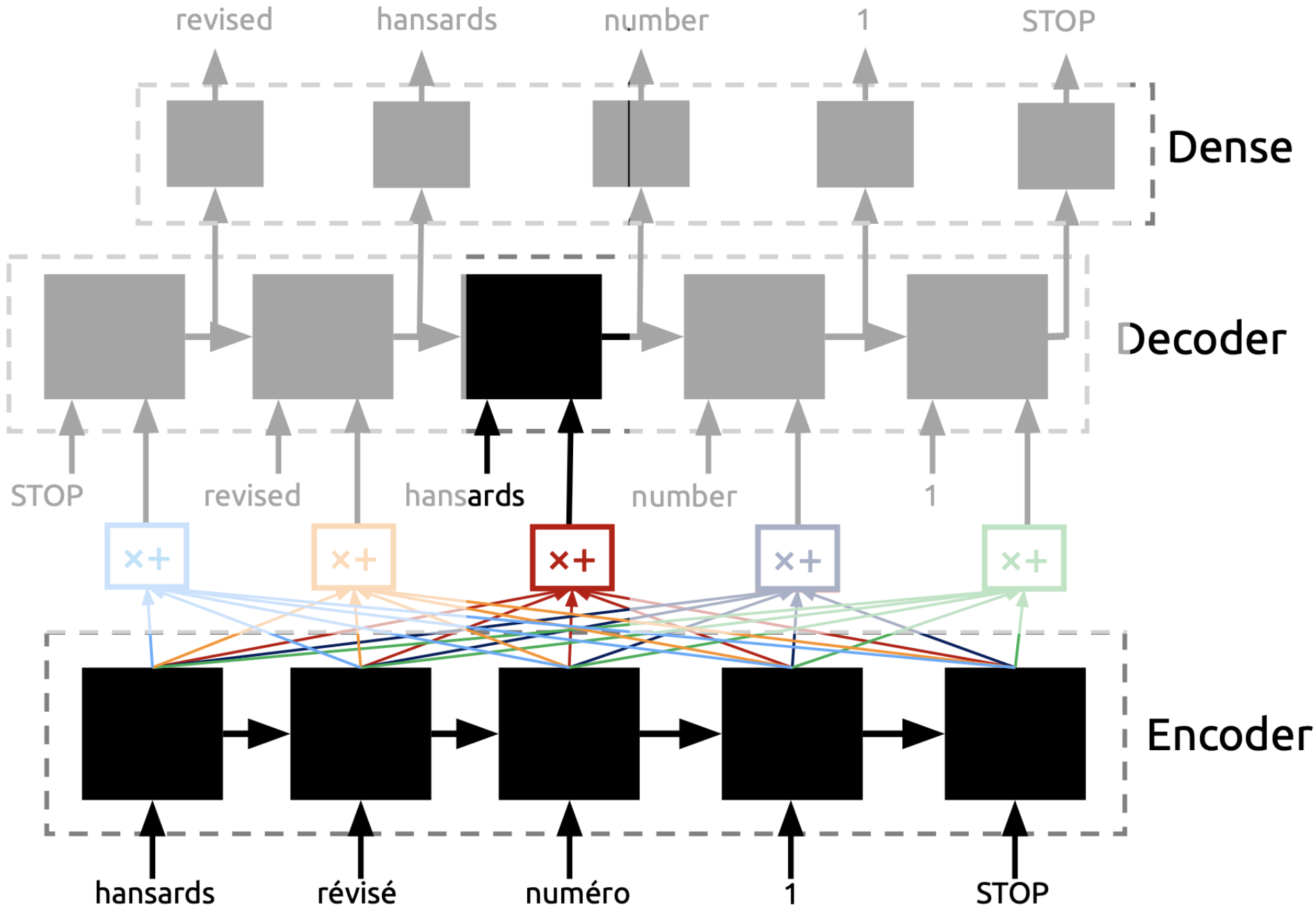
This idea of passing each cell of the decoder a weighted sum of the encoder states is called ***attention.***

- Different words in the output “pay attention” to different words in the input

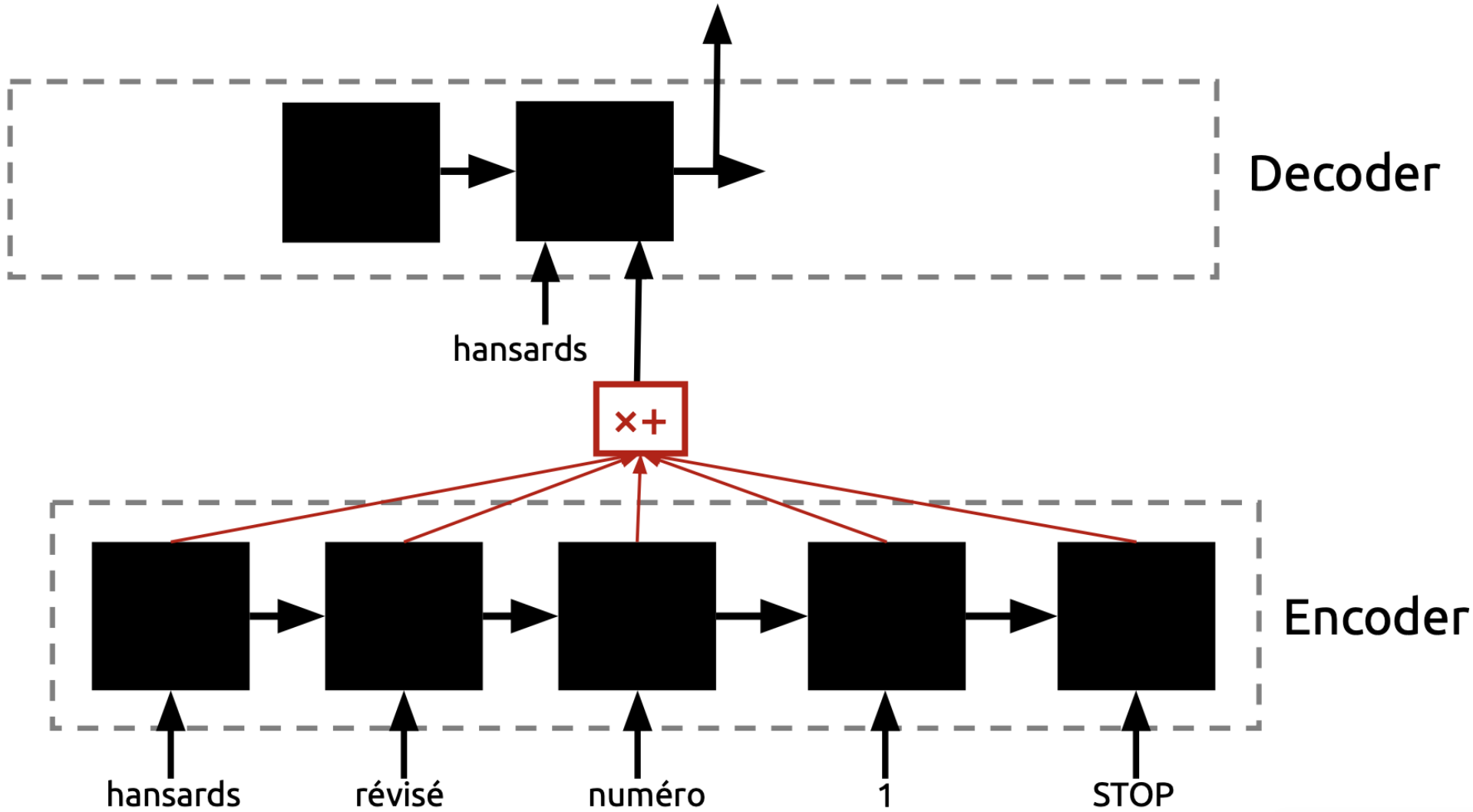
“Attention” - intuition

“Park”

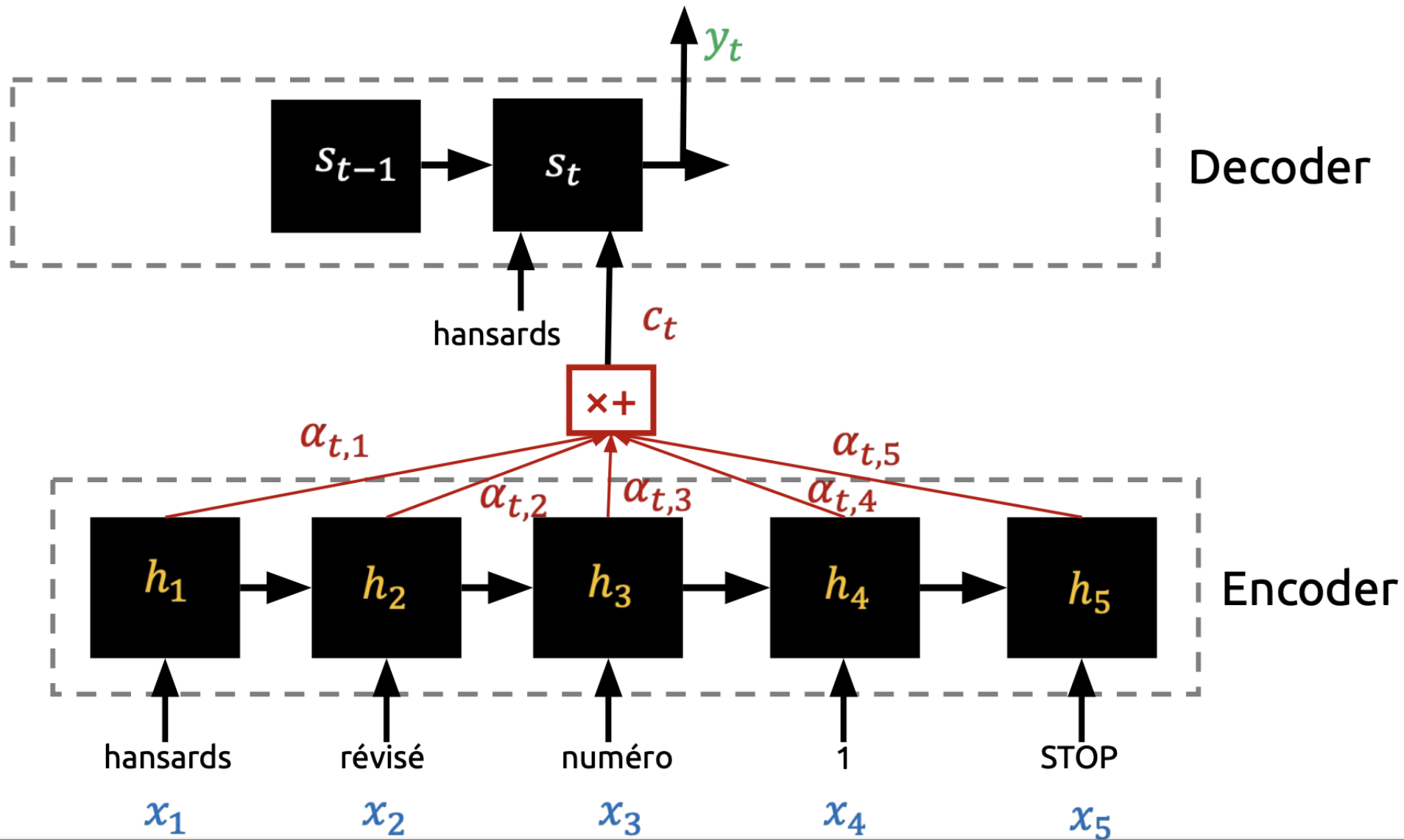
How about we let model learn what is relevant for a particular output


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

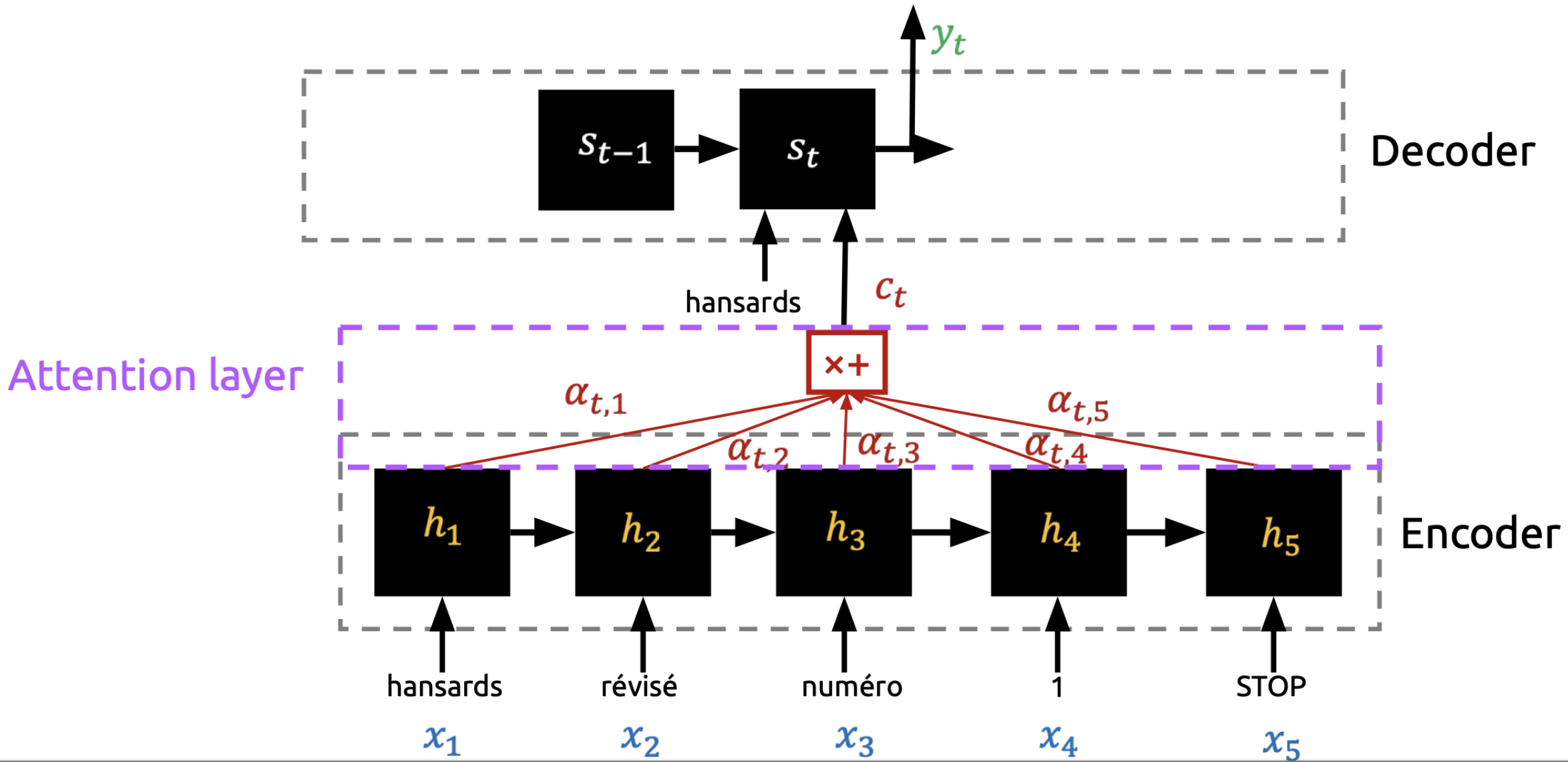
Attention - implementation



Attention - implementation



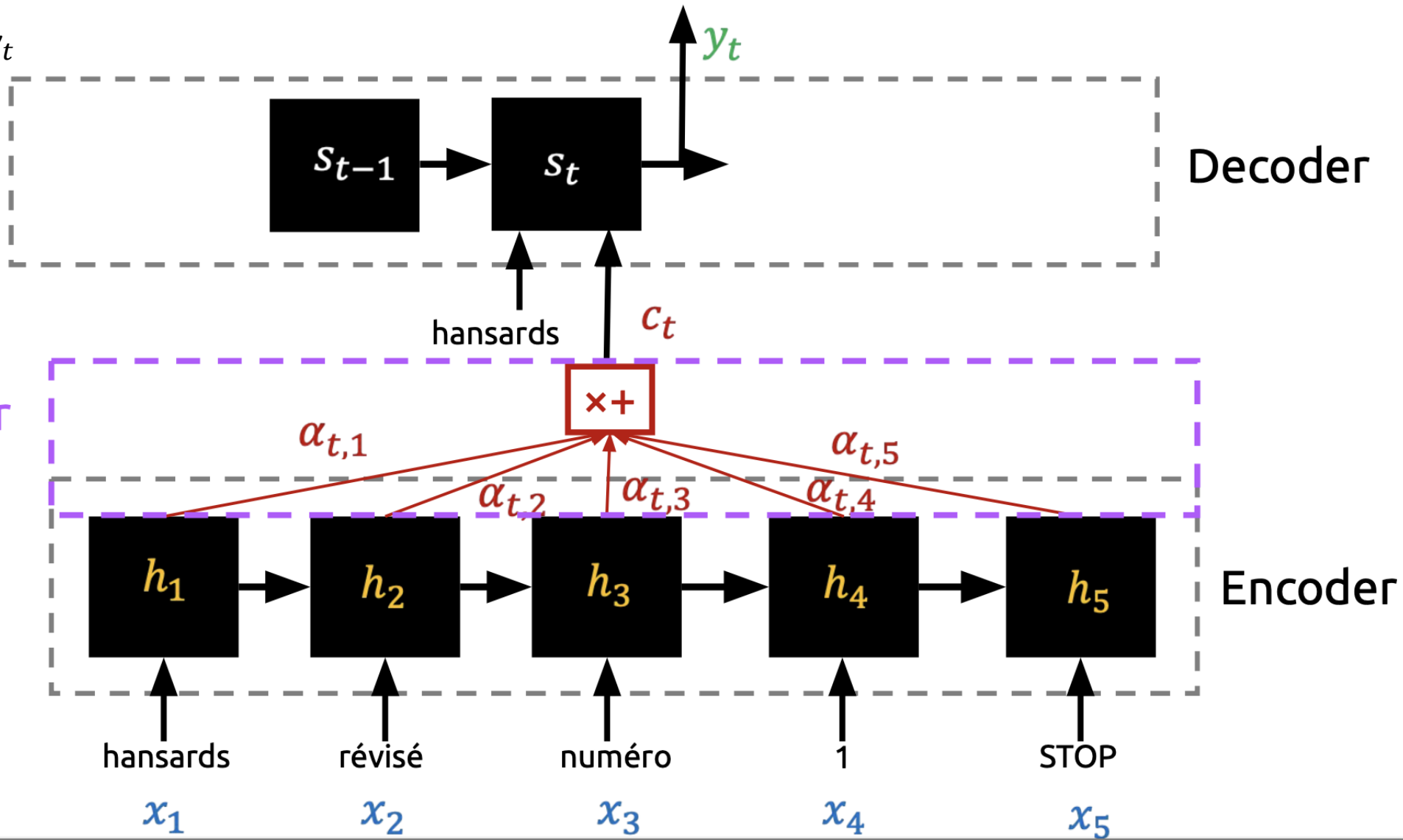
Attention - implementation



Attention - implementation

Context Vector for output y_t

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



Attention - implementation

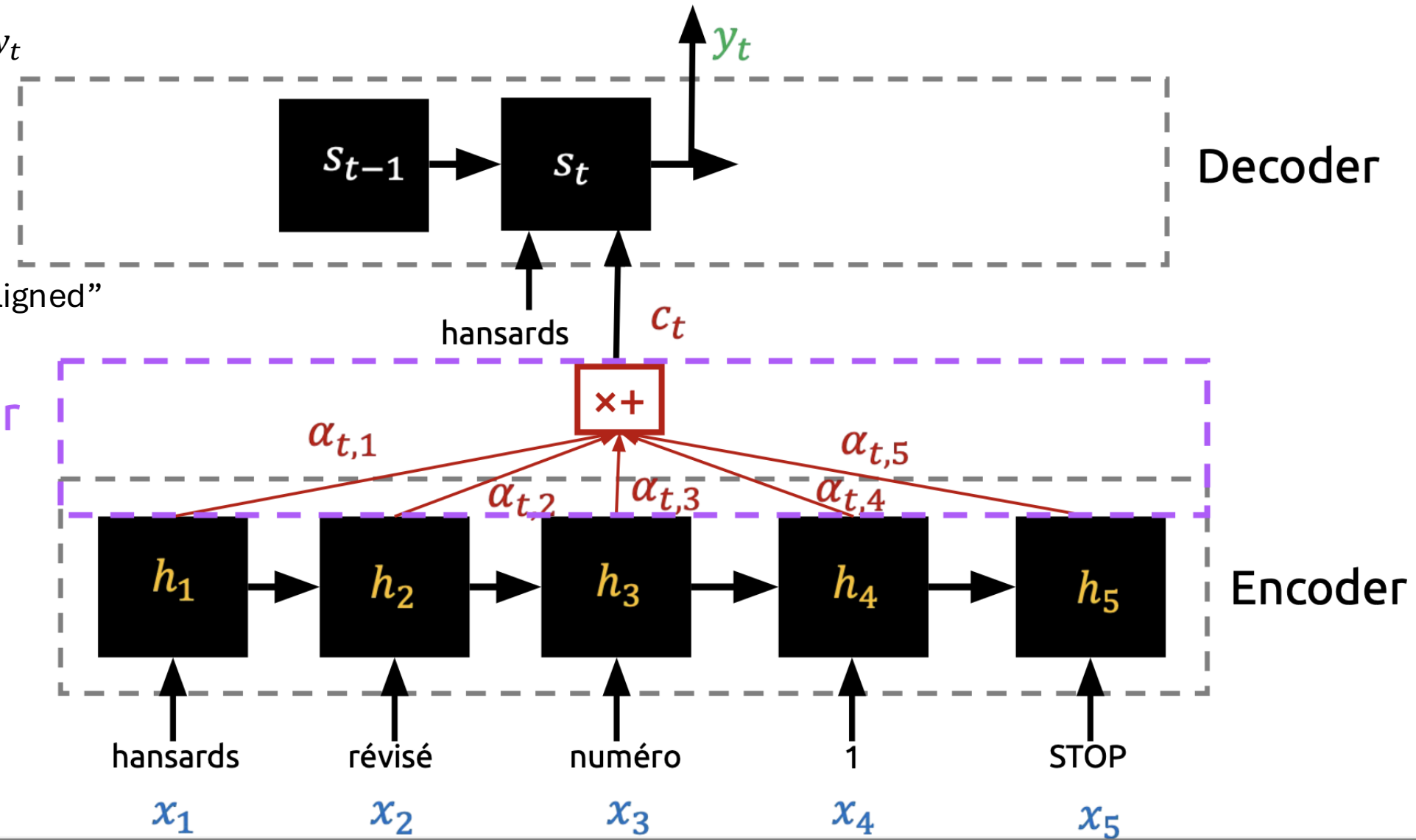
Context Vector for output y_t

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

How well two words are "aligned"

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

Attention layer



Attention - implementation

Context Vector for output y_t

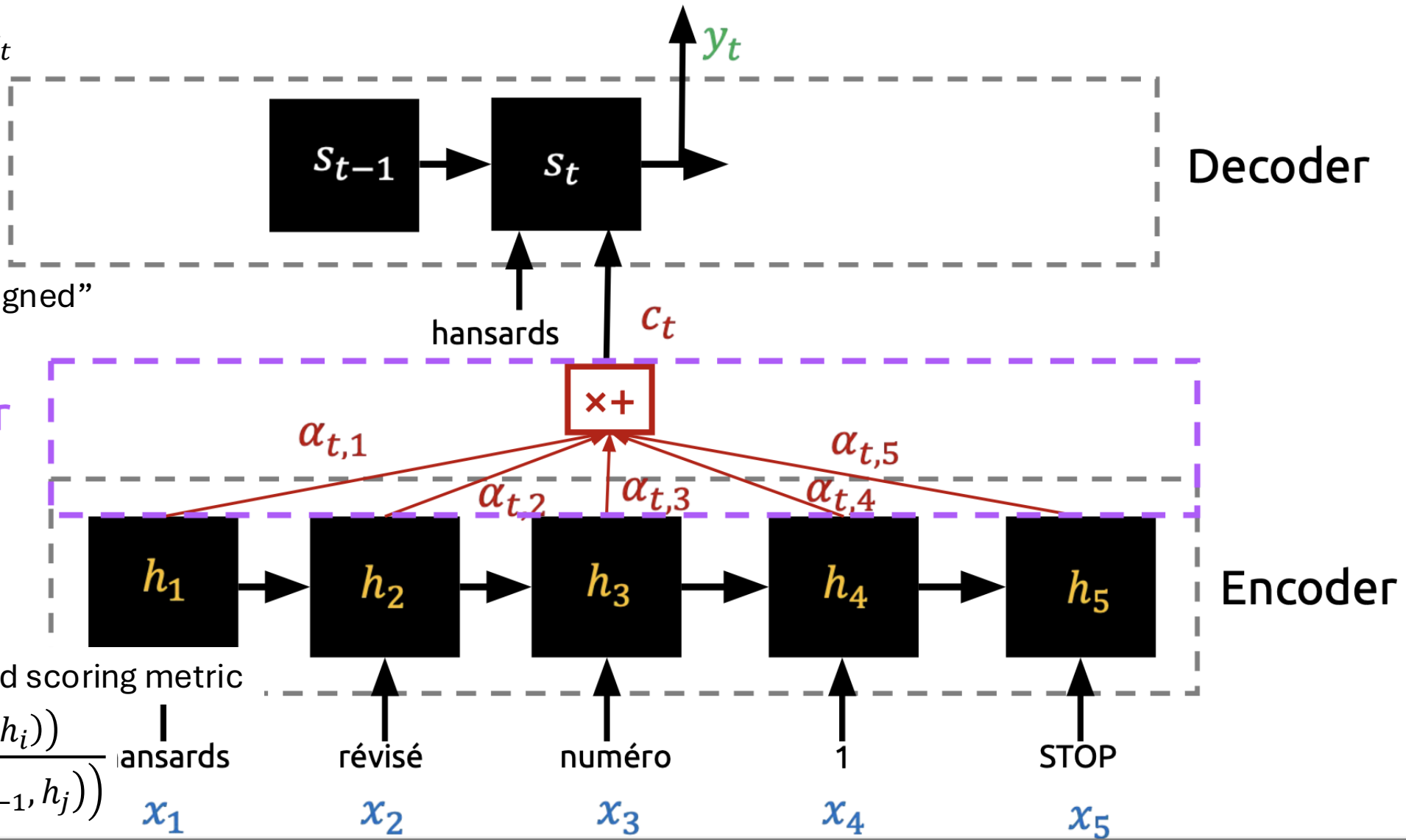
$$c_t = \sum_{i=1} a_{t,,i} h_i$$

How well two words are "aligned"

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

Softmax of some predefined scoring metric

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



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)?

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)?

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

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

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

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

There are many ways to measure similarity…

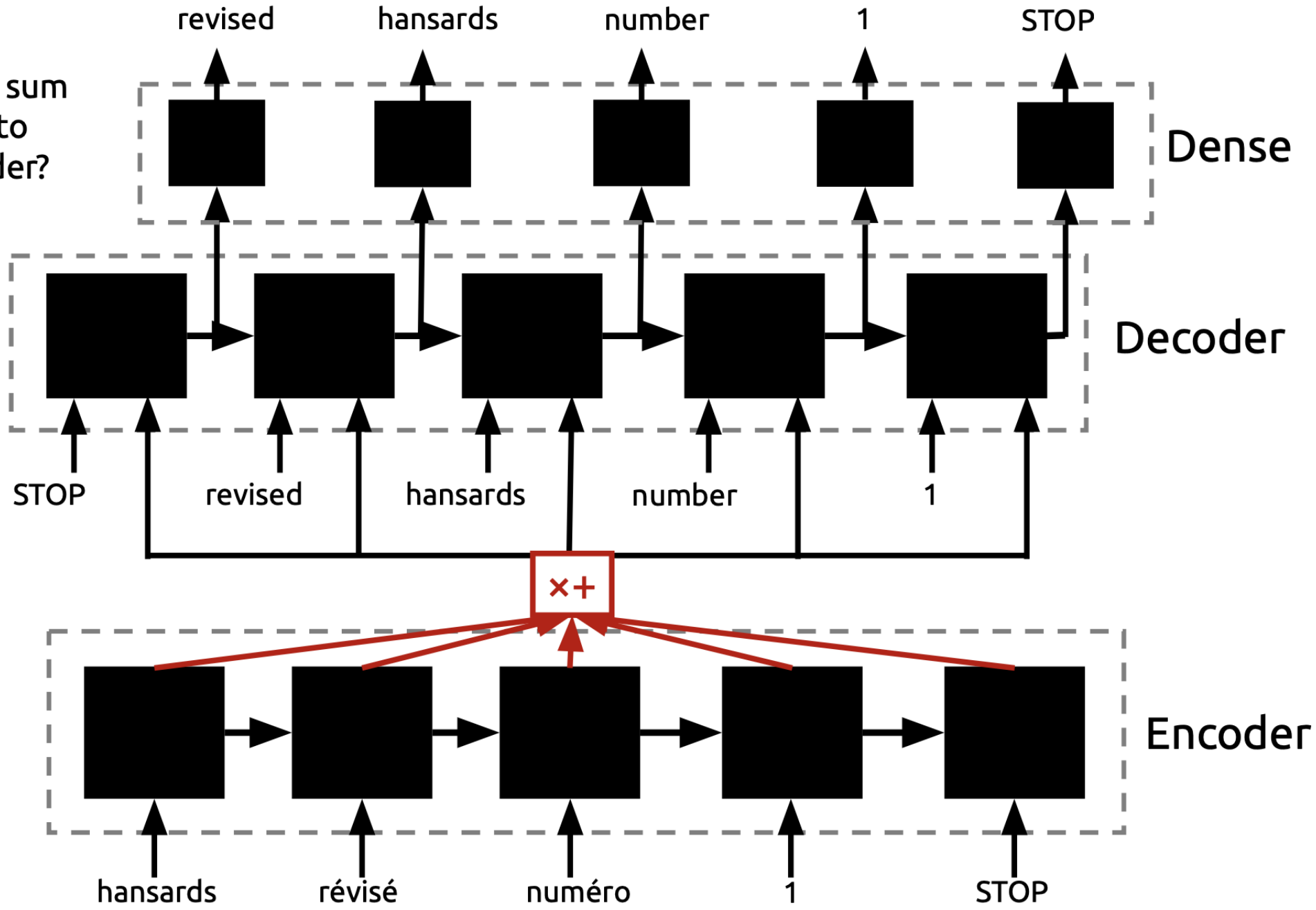
Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ 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(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ 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

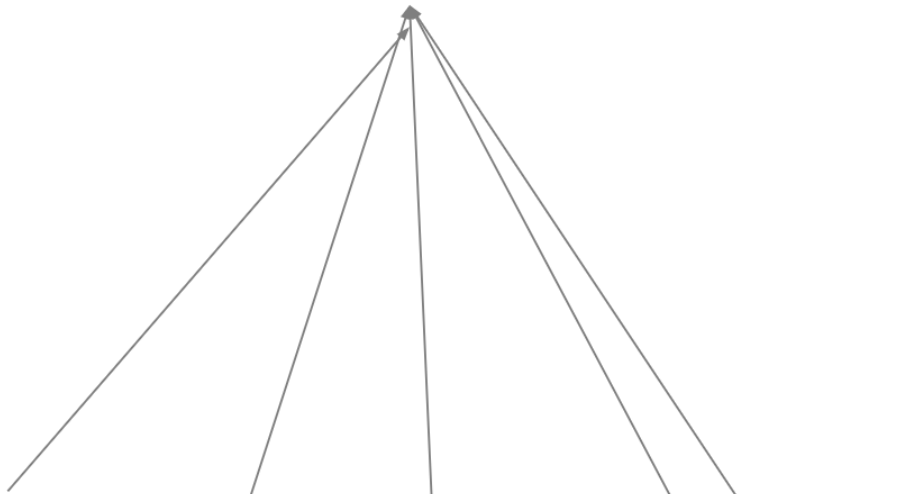
	hansards	révisé	numéro	1	STOP
revised	1/2	1/4	1/4	0	0
hansards	1/4	1/2	1/4	0	0
number	0	1/4	1/2	1/4	0
1	0	0	1/4	1/2	1/4
STOP	0	0	1/4	1/4	1/2

$\alpha_{j,i}$: 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 bellte 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:

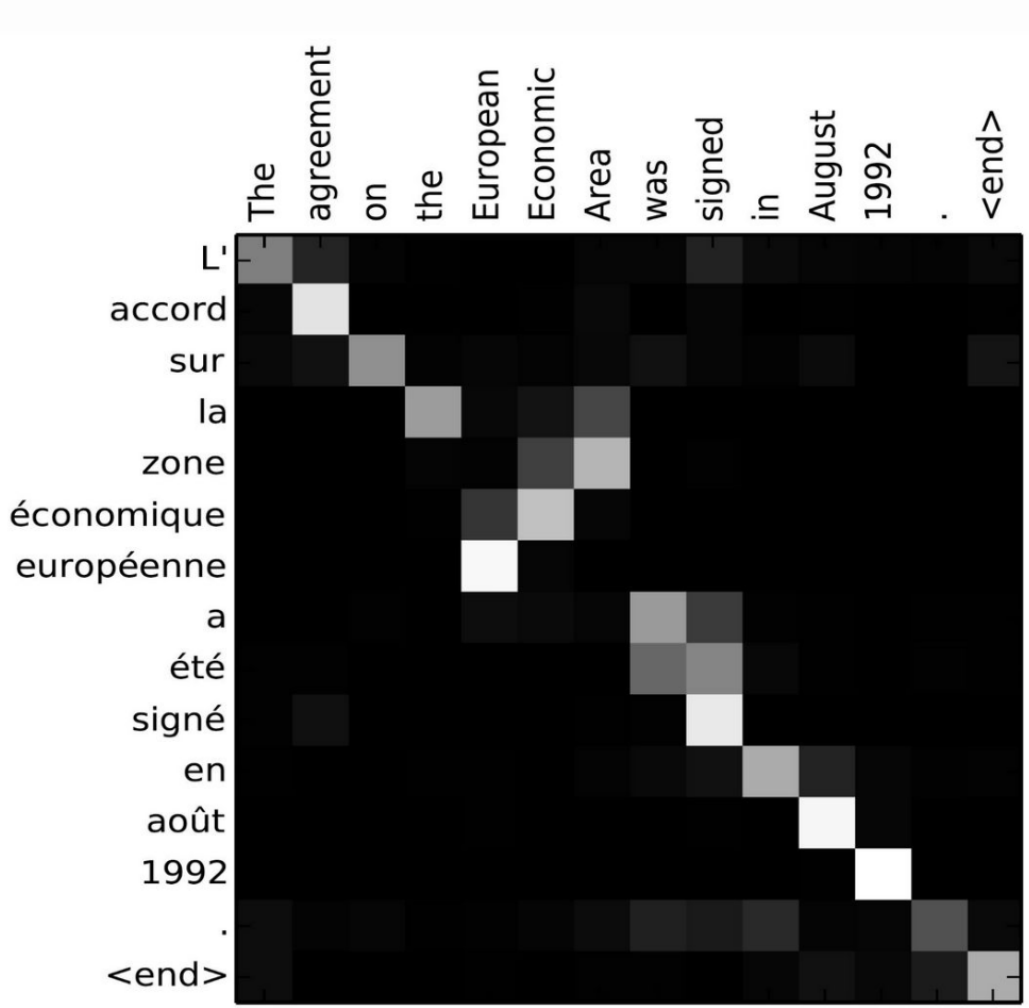
Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.

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).

Credit: Stanford CS224n

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.

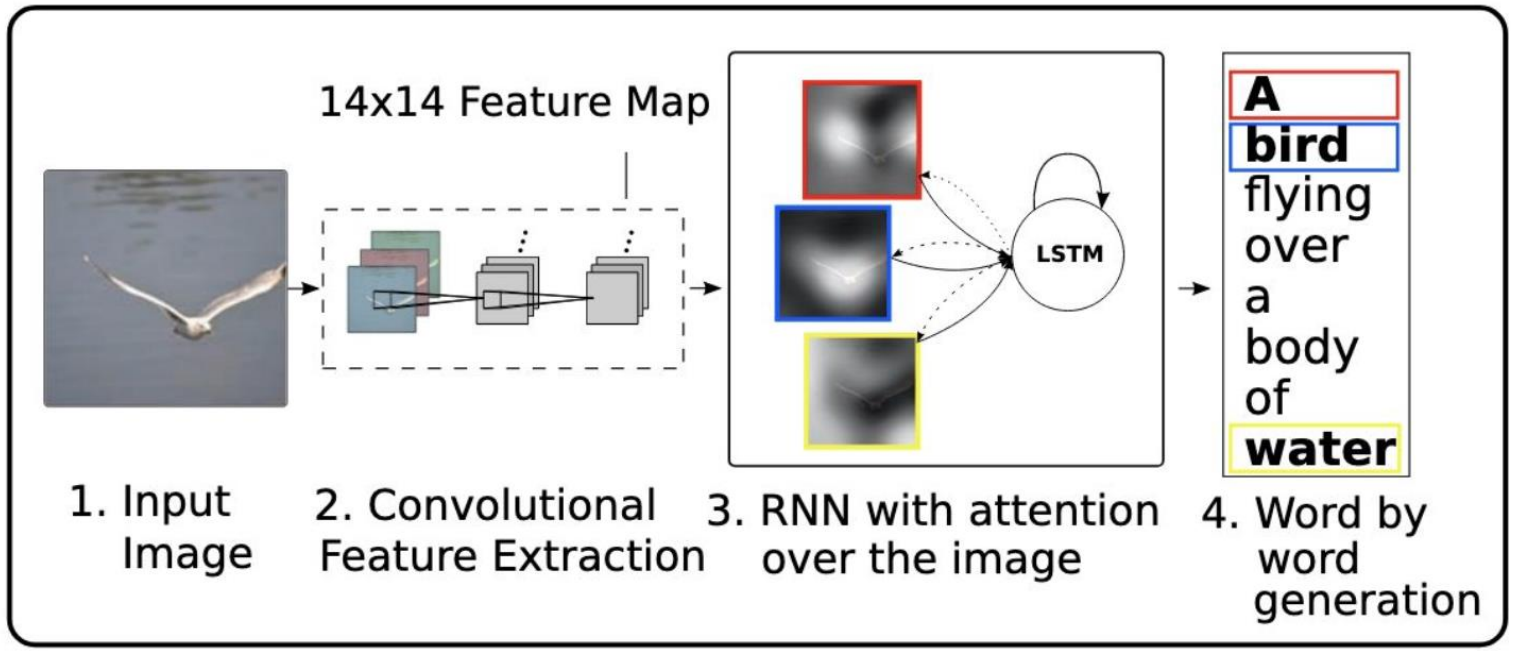
A group of people sitting on a boat in the water.

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.

A large white bird standing in a forest.

A woman holding a clock in her hand.

A man wearing a hat and
a hat on a skateboard.

A person is standing on a beach
with a surfboard.

A woman is sitting at a table
with a large pizza.

A man is talking on his cell phone
while another man watches.

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

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

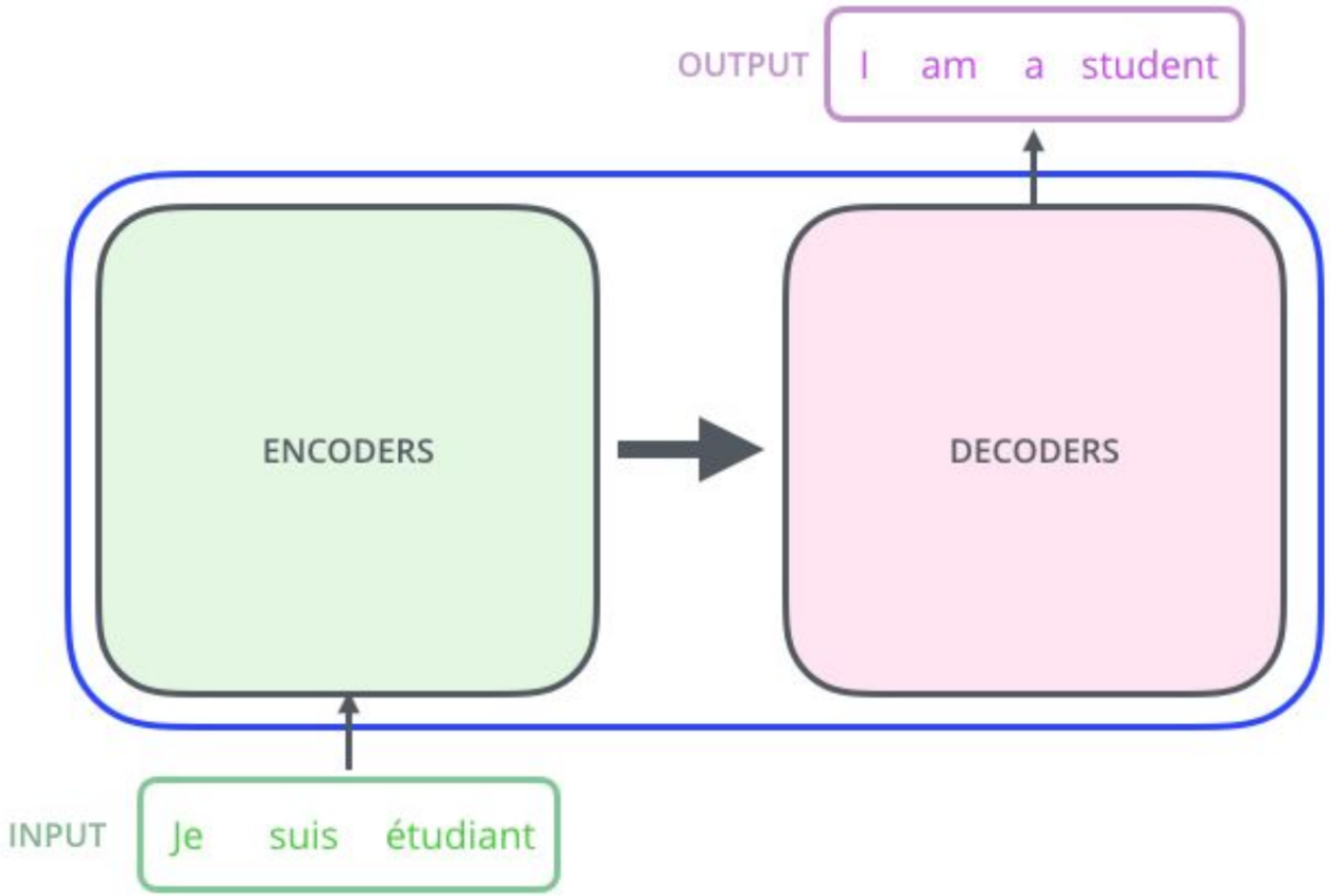
- Has no recurrent networks!
- ***Only*** uses attention

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

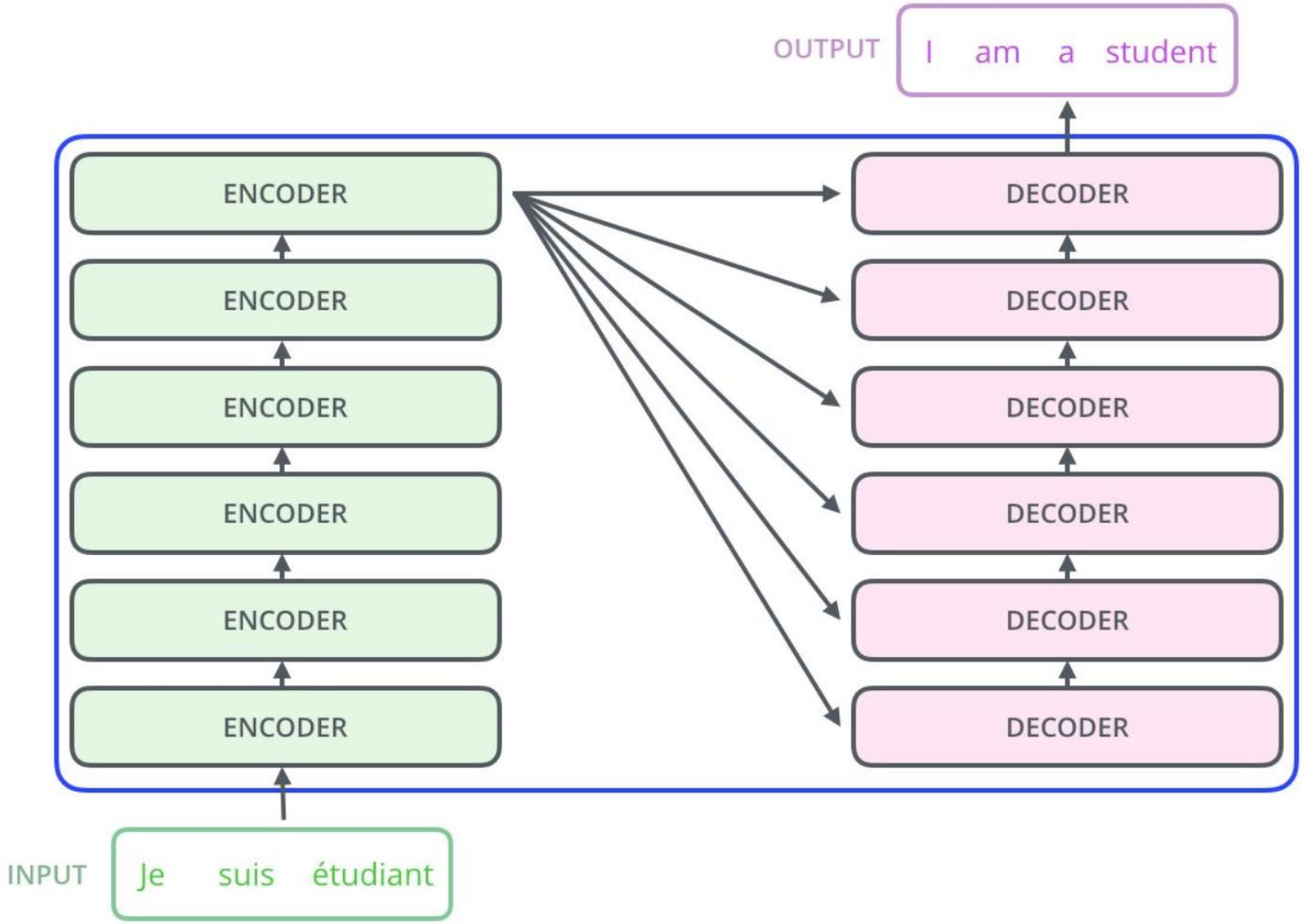
- The Transformer model breaks down into Encoder and Decoder blocks.
- At a high level, similar to the seq2seq architecture we've seen already...
- ...but there are no recurrent nets inside the Encoder and Decoder blocks!



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

Transformer Model Overview

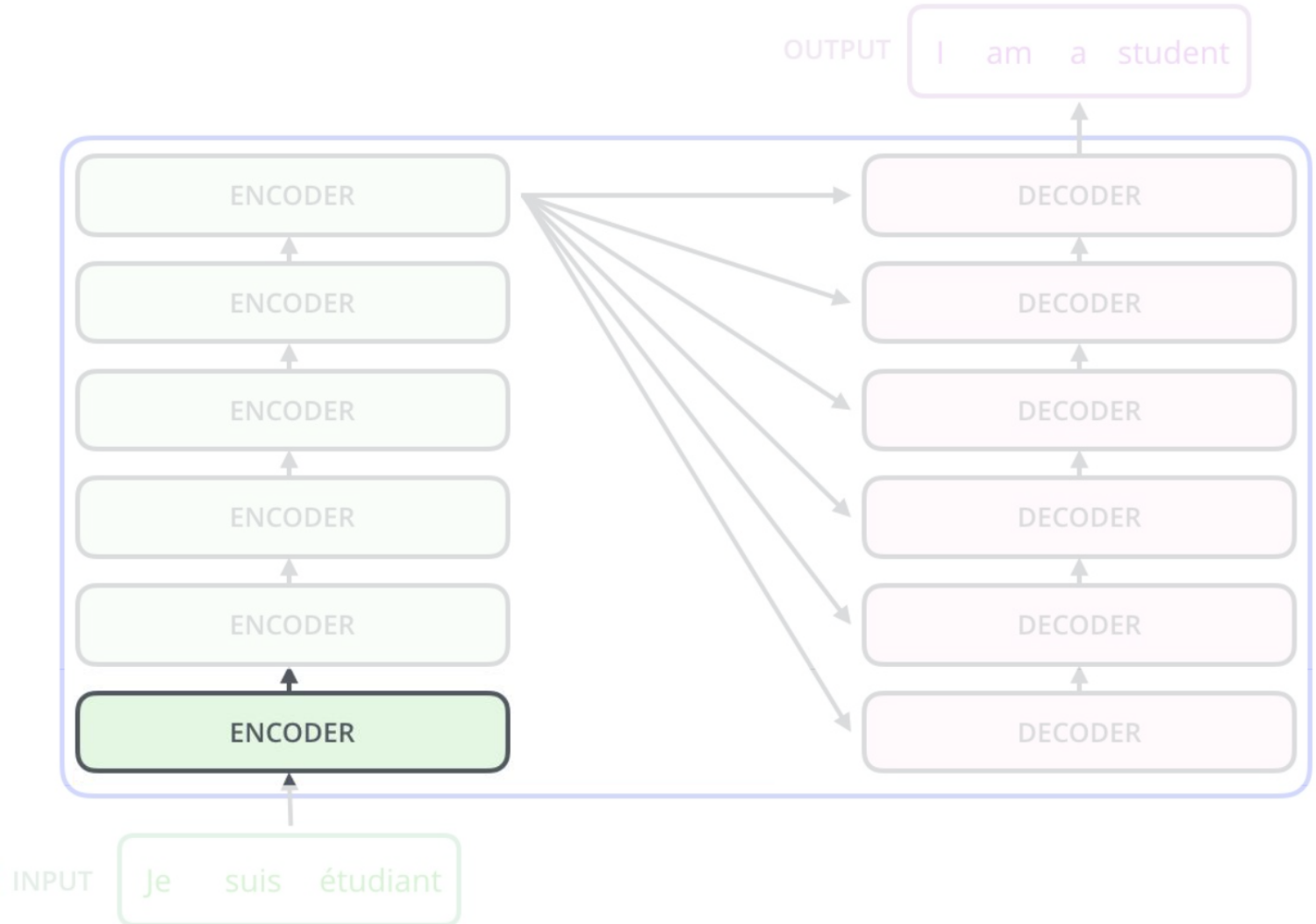
- The Transformer model breaks down into Encoder and Decoder blocks.
- At a high level, similar to the seq2seq architecture we've seen already...
- ...but there are no recurrent nets inside the Encoder and Decoder blocks!
- For better performance, often stack multiple Encoder and Decoder blocks (deeper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

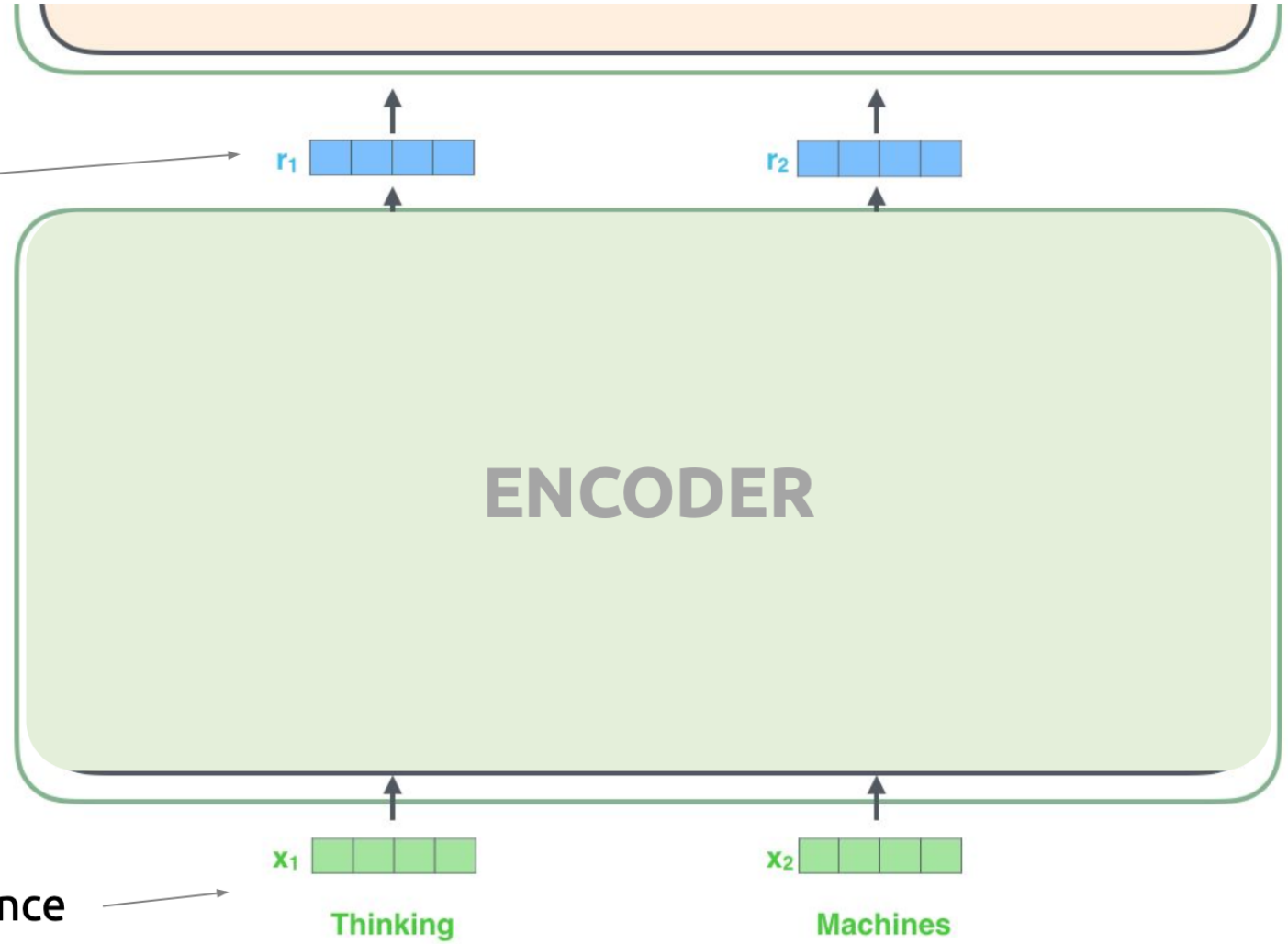


Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

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?"*

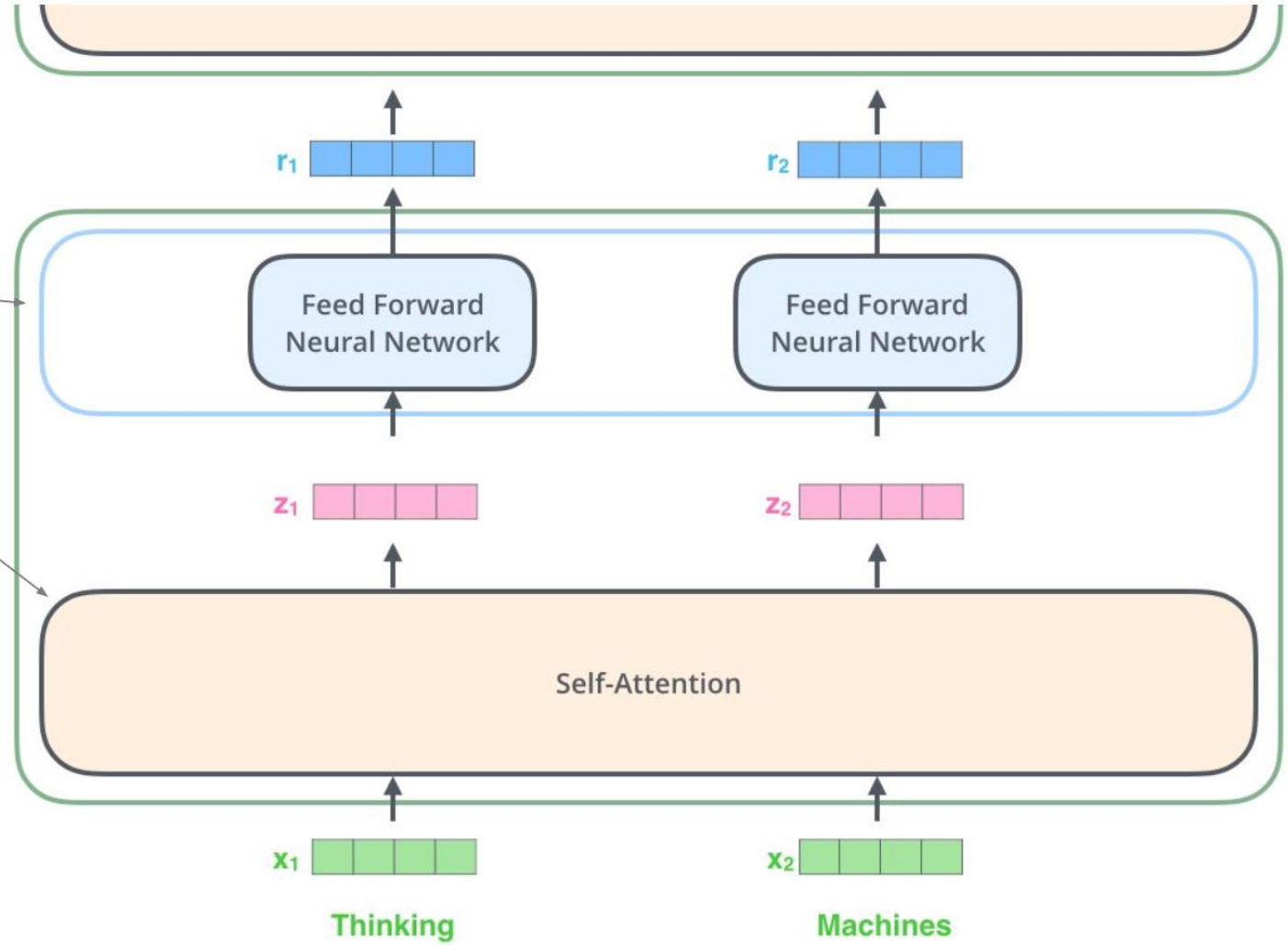


Words in input sentence

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

Encoder Block Map

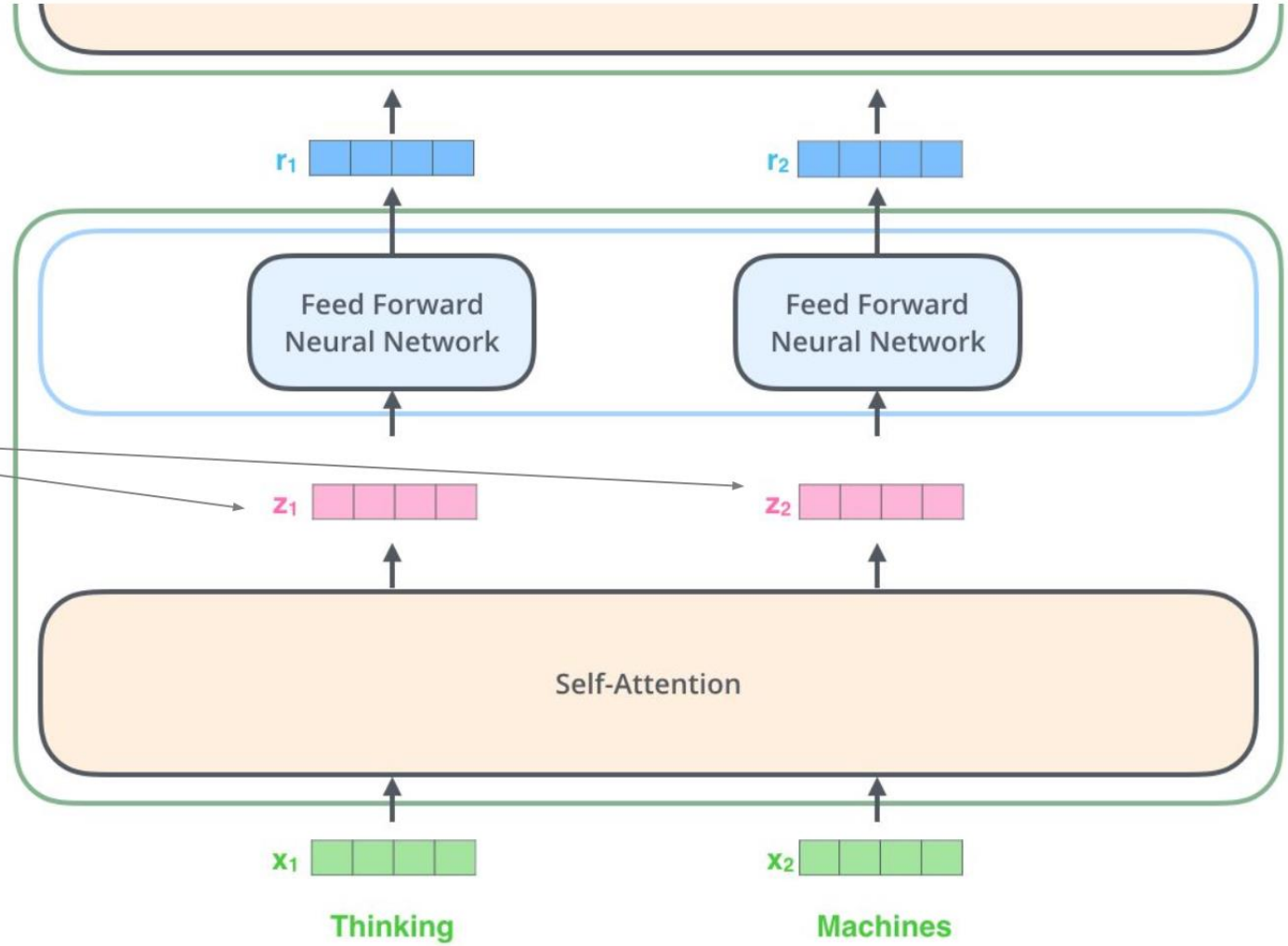
- Encoder block breaks down into two main parts: Self-Attention, and Feed Forward layers.



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

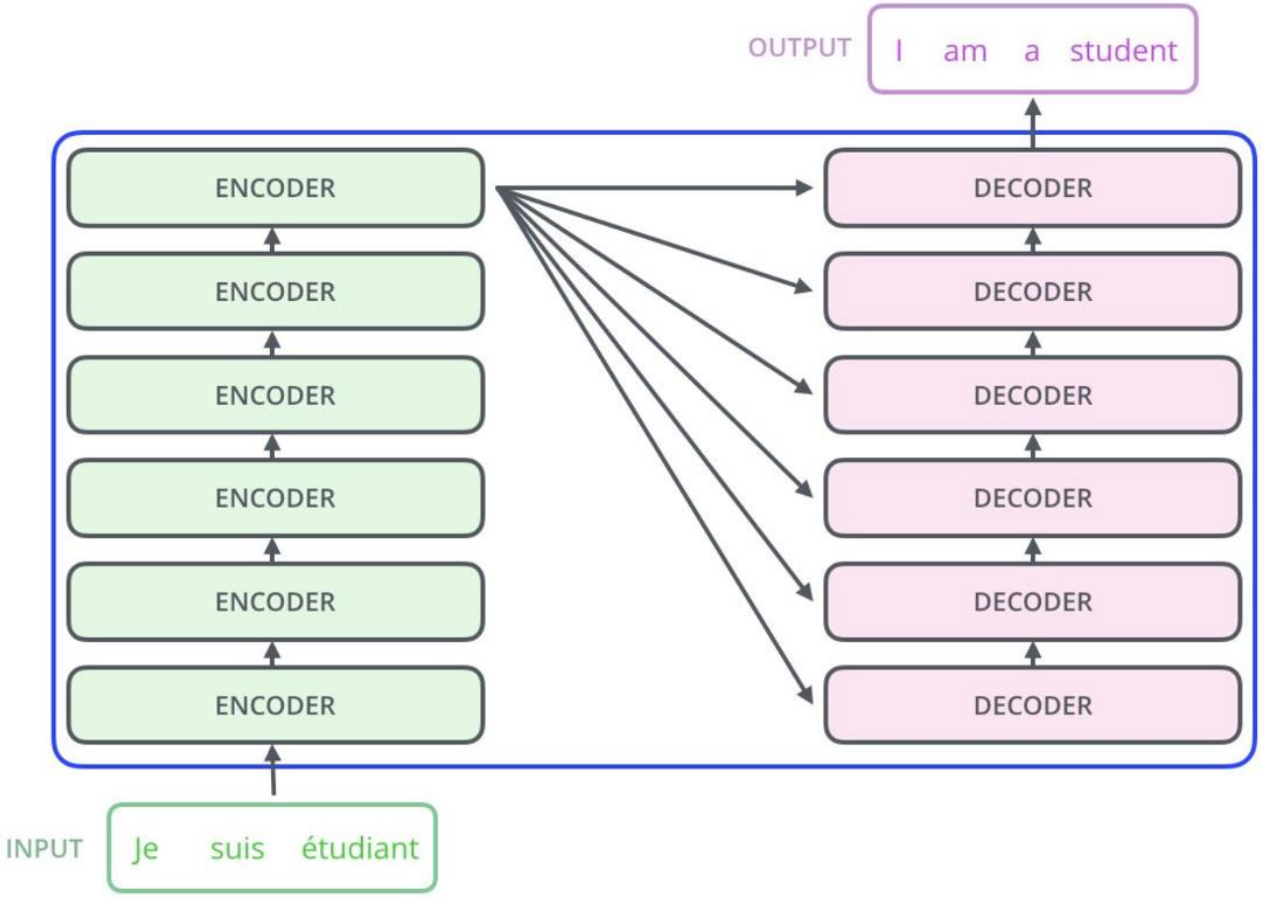
Encoder Block Map

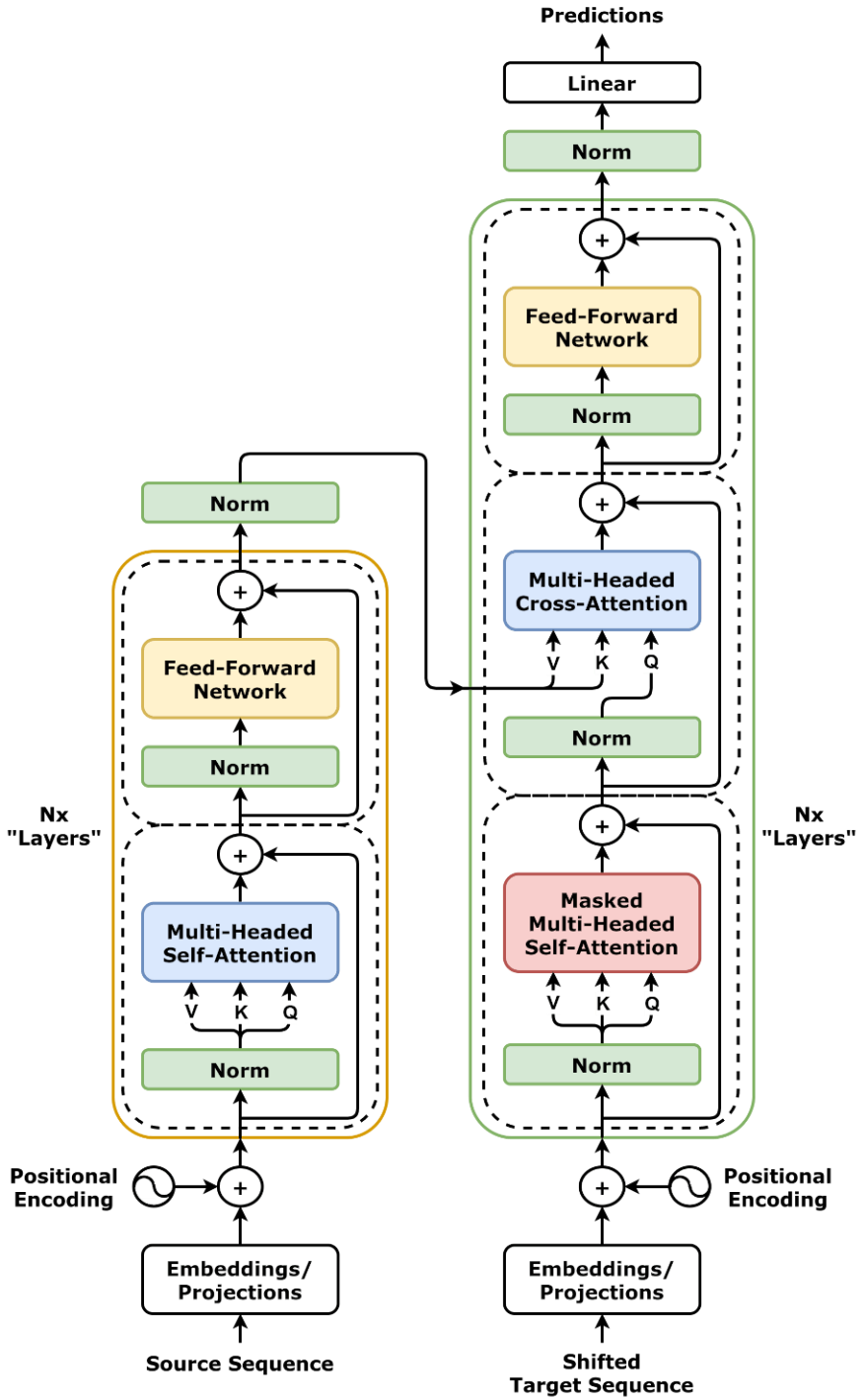
- Encoder block breaks down into two main parts: Self-Attention, and Feed Forward layers.
- Self-Attention layer is applied to each word individually.



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

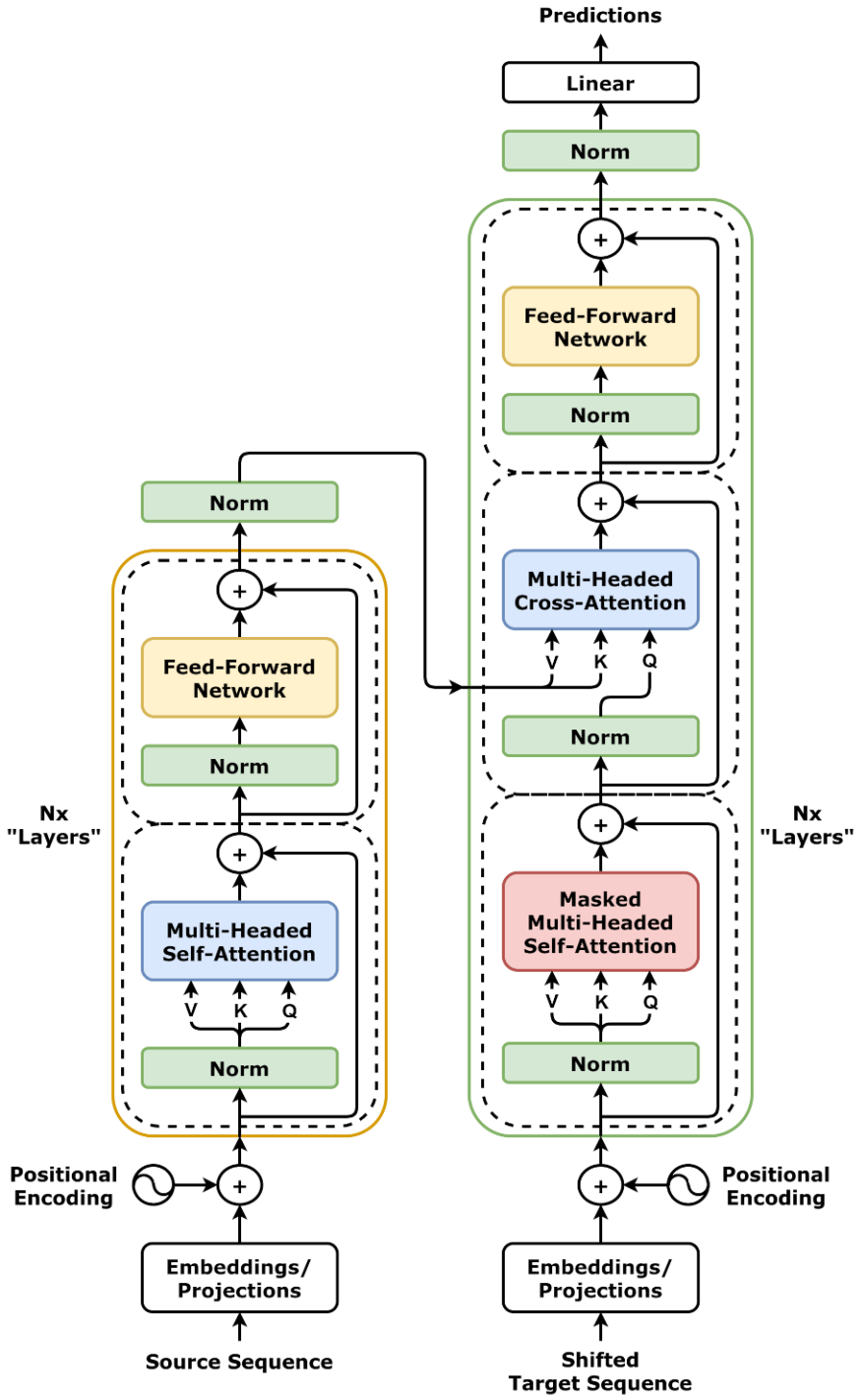
The Transformer



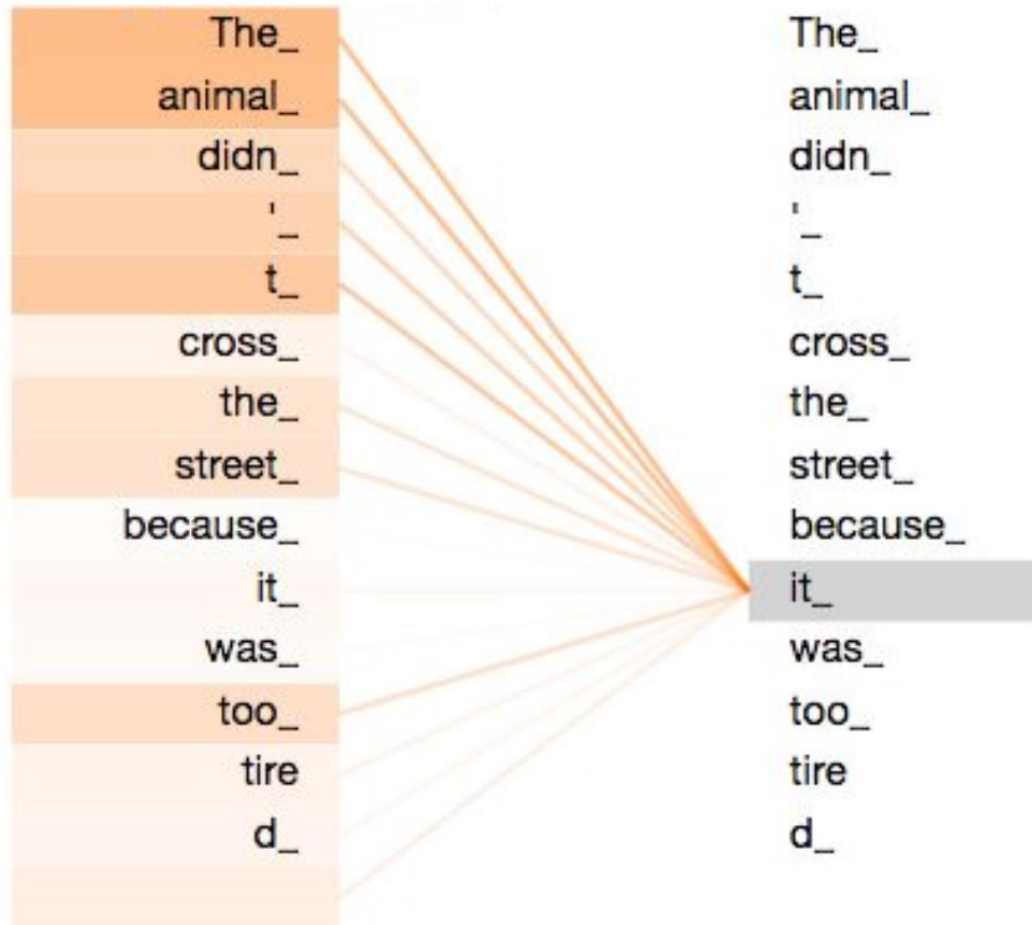


Components

- Self-Attention
- Cross-Attention
- Position Encoding
- Norm



Self-Attention: Input's attention on itself



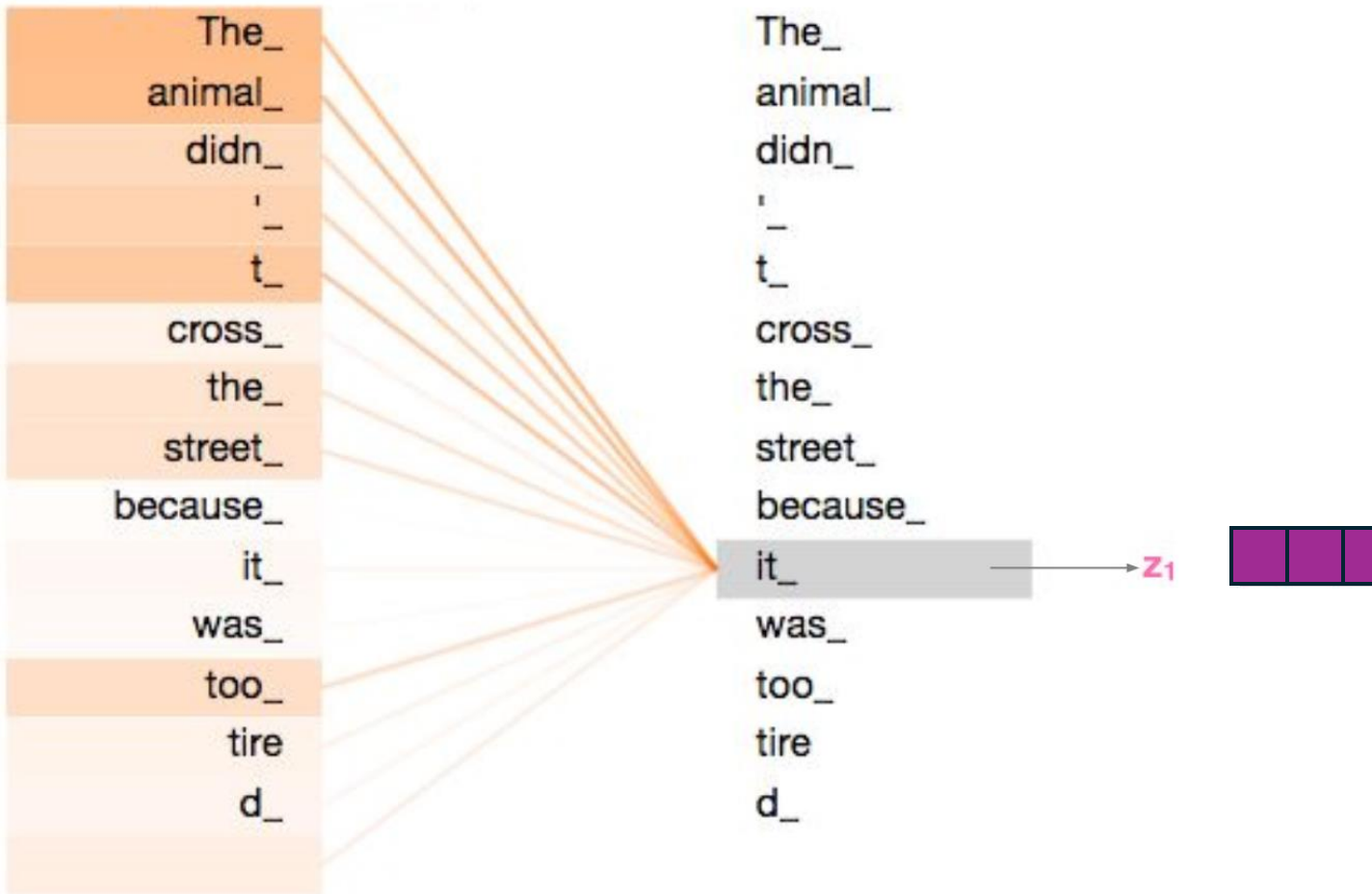
Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Self-Attention: 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

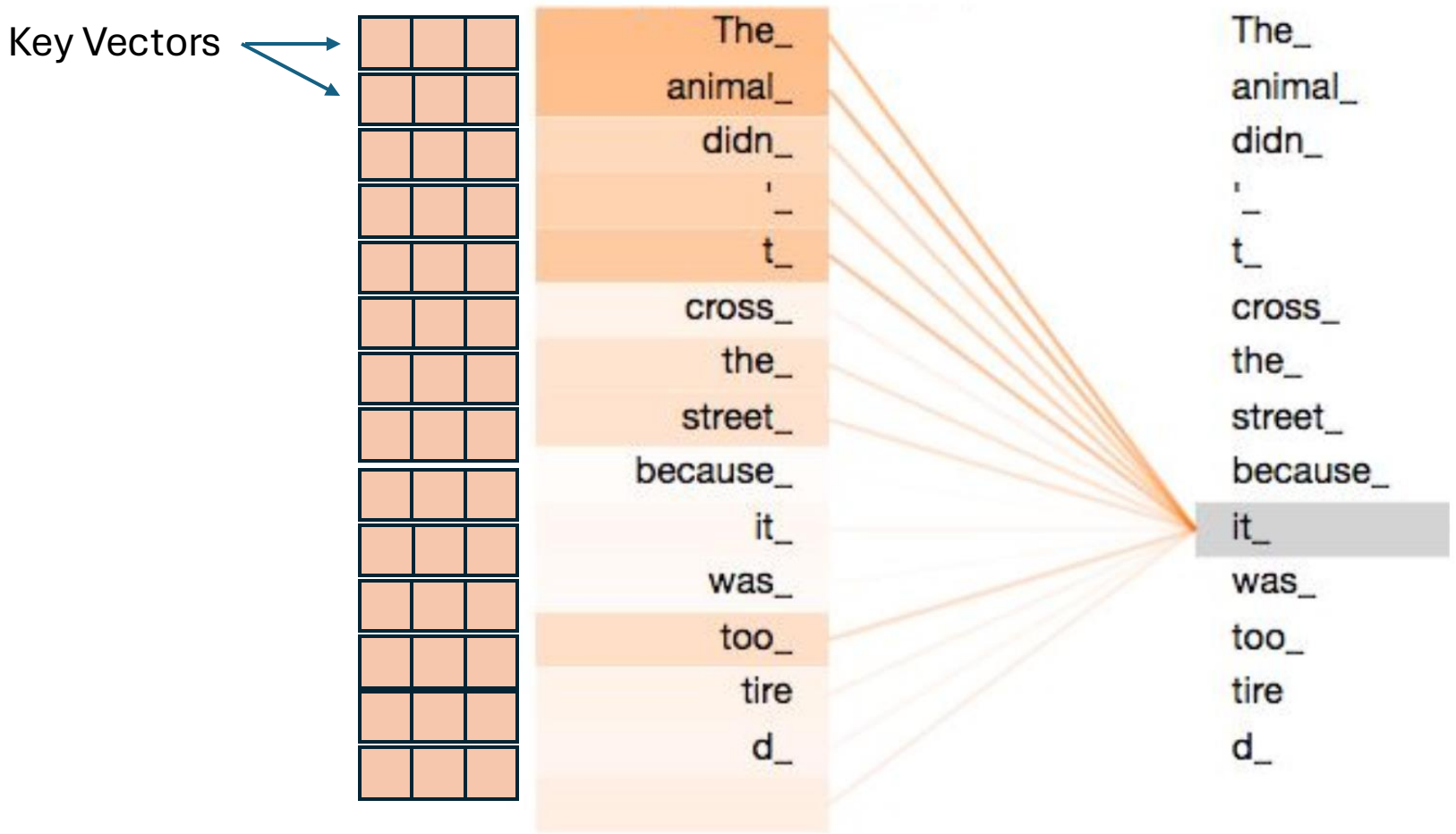
- Here, self-attention learns that “it” should pay attention to “the animal” (i.e. the entity that “it” refers to)

- 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/

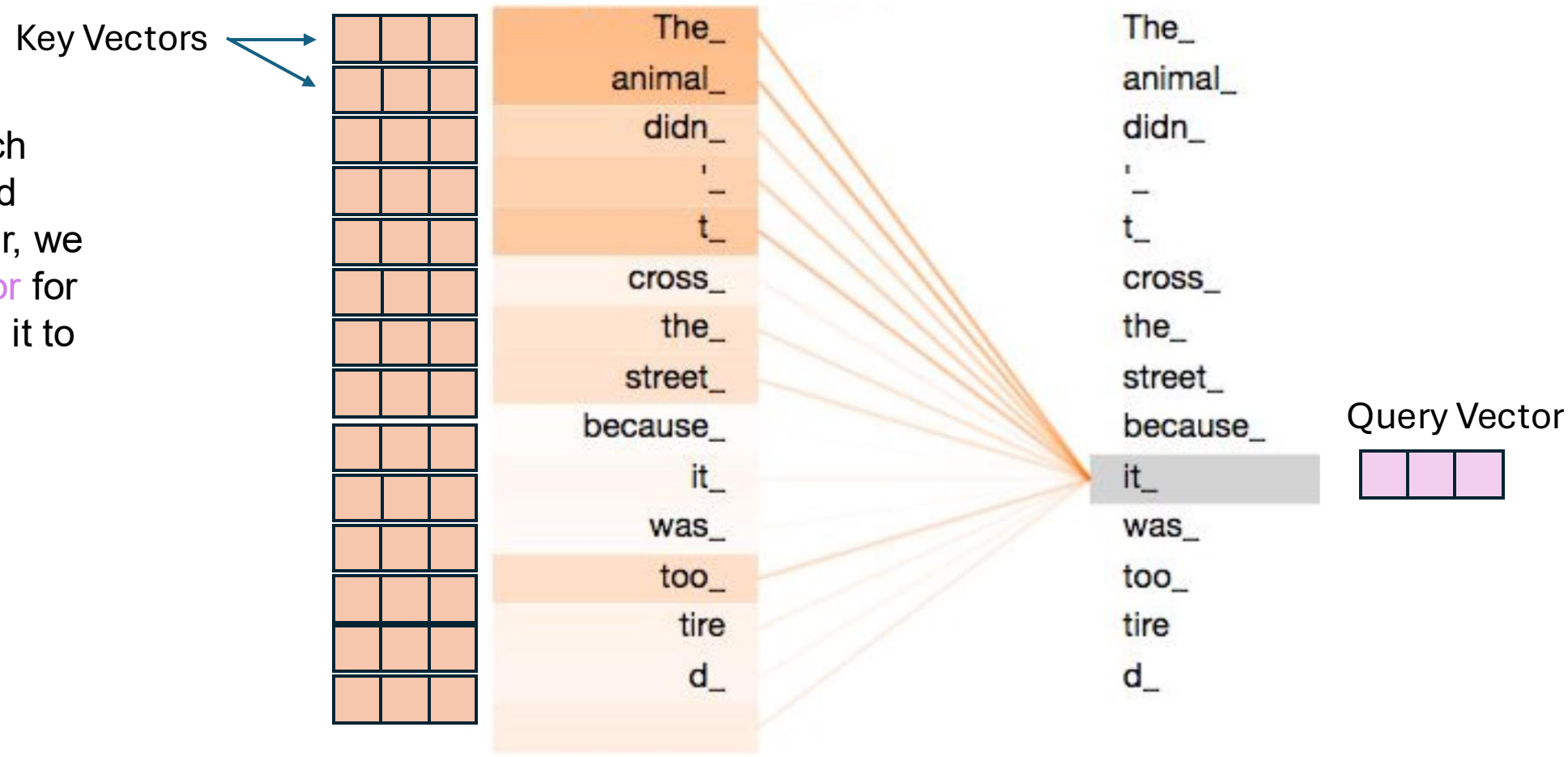
Self-Attention: Input's attention on itself



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

Self-Attention: Input's attention on itself

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...

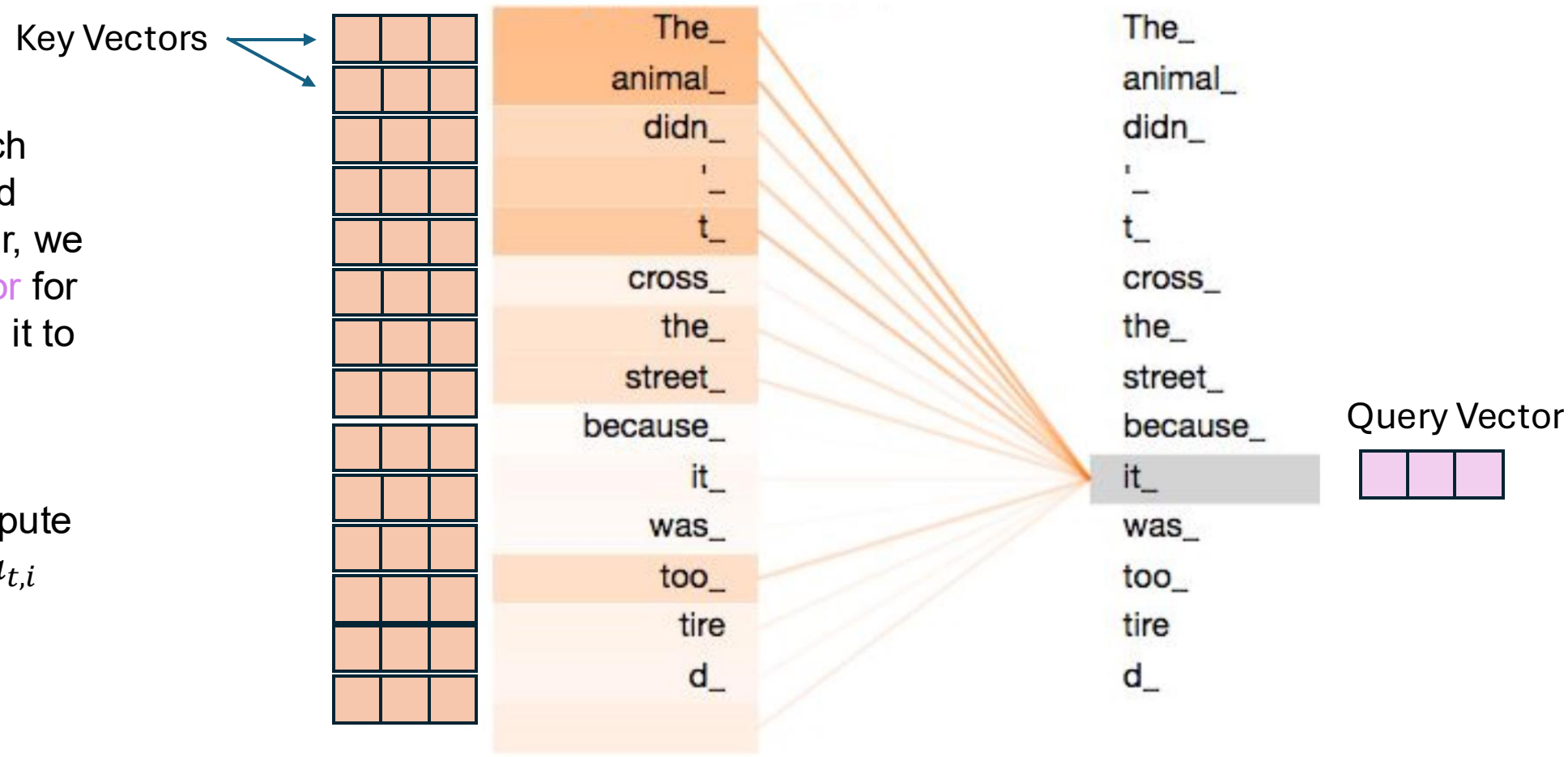


Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

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 $a_{t,i}$

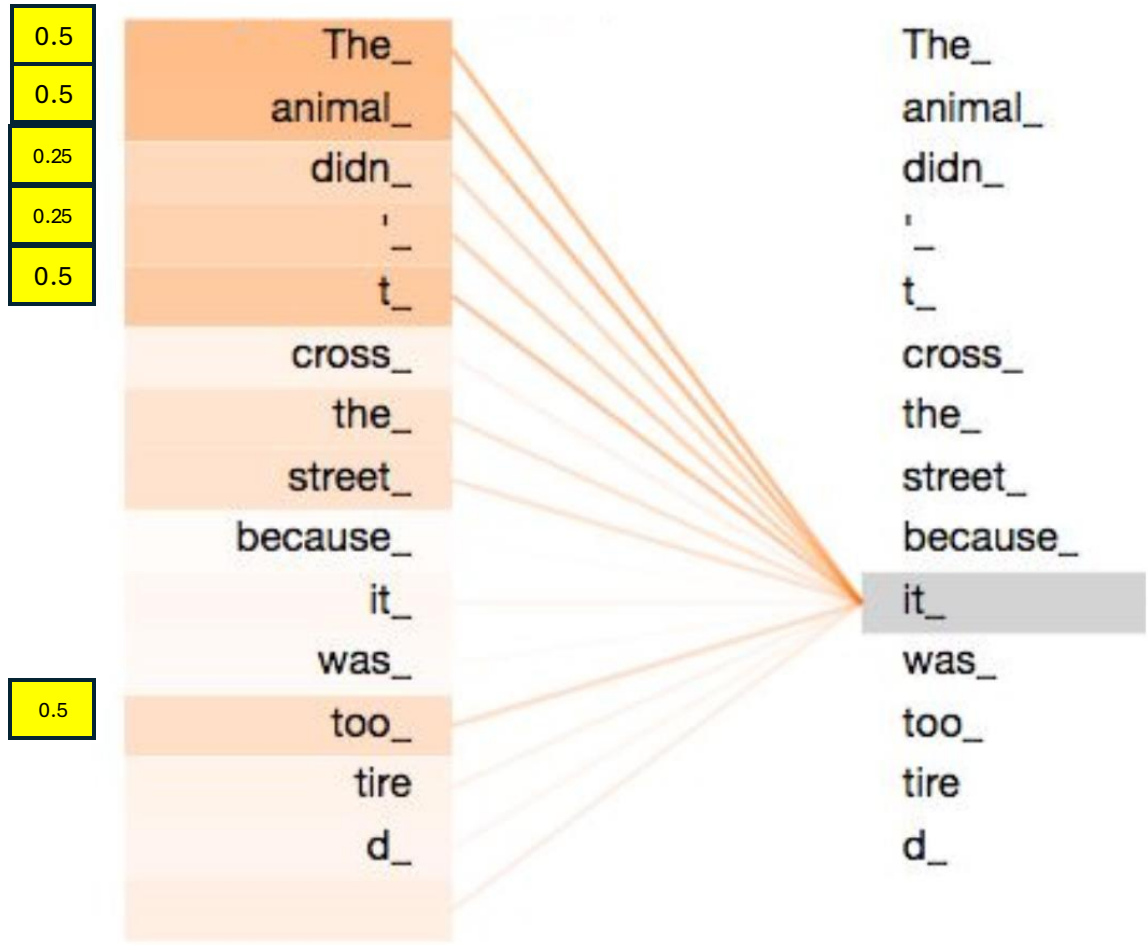


Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Self-Attention: Input's attention on itself

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 $a_{t,i}$

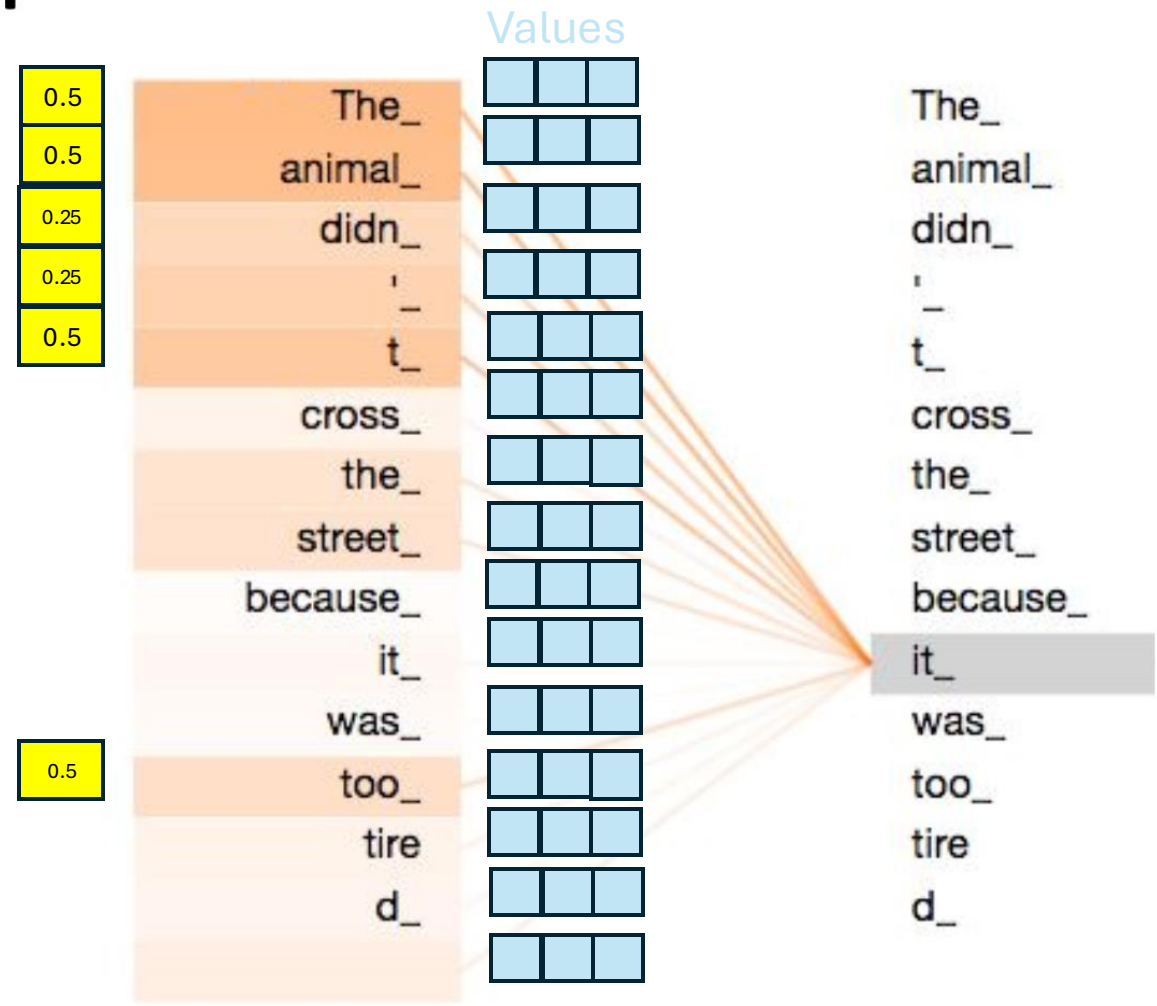


Self-Attention: Input's attention on itself

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 $a_{t,i}$

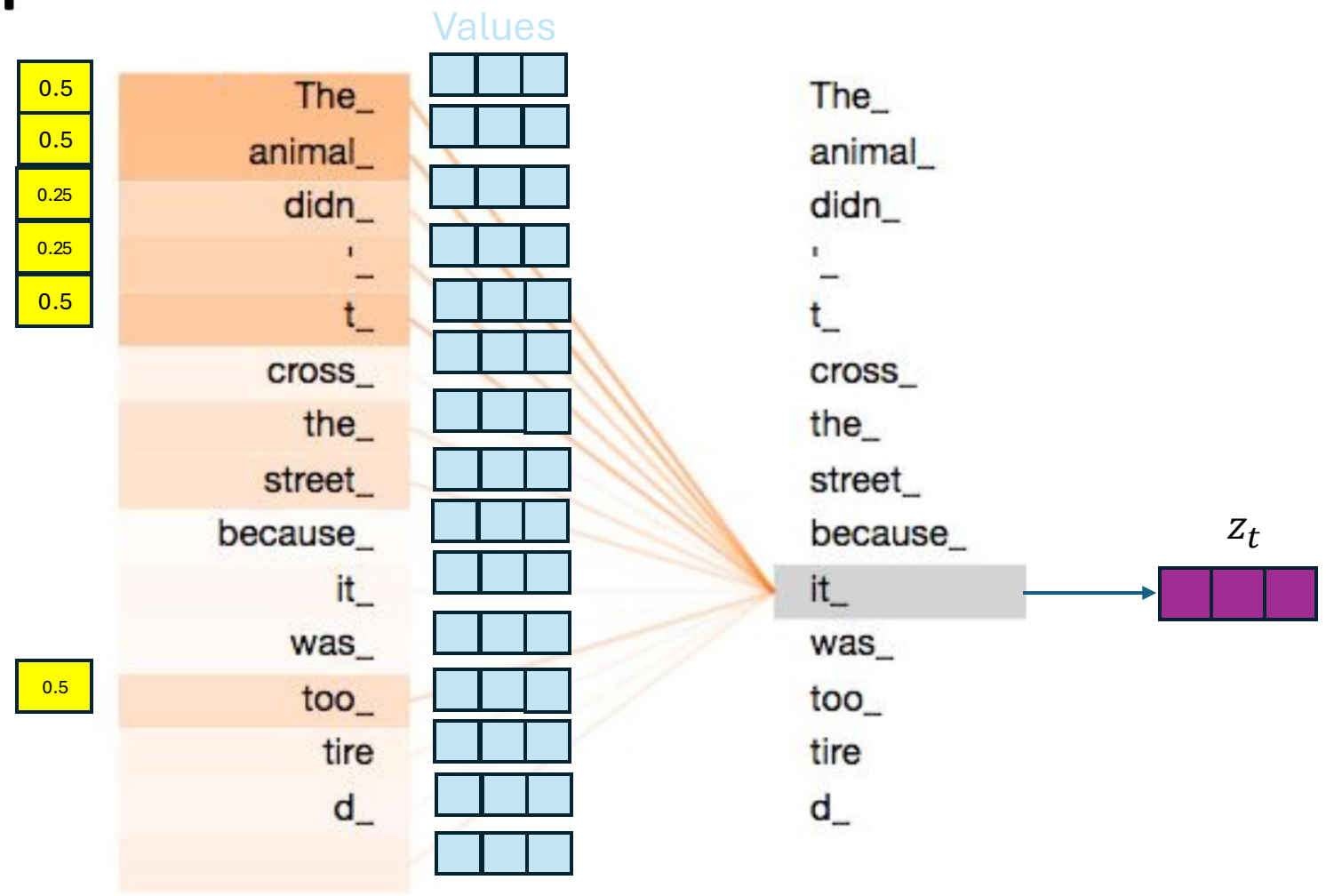


Self-Attention: Input's attention on itself

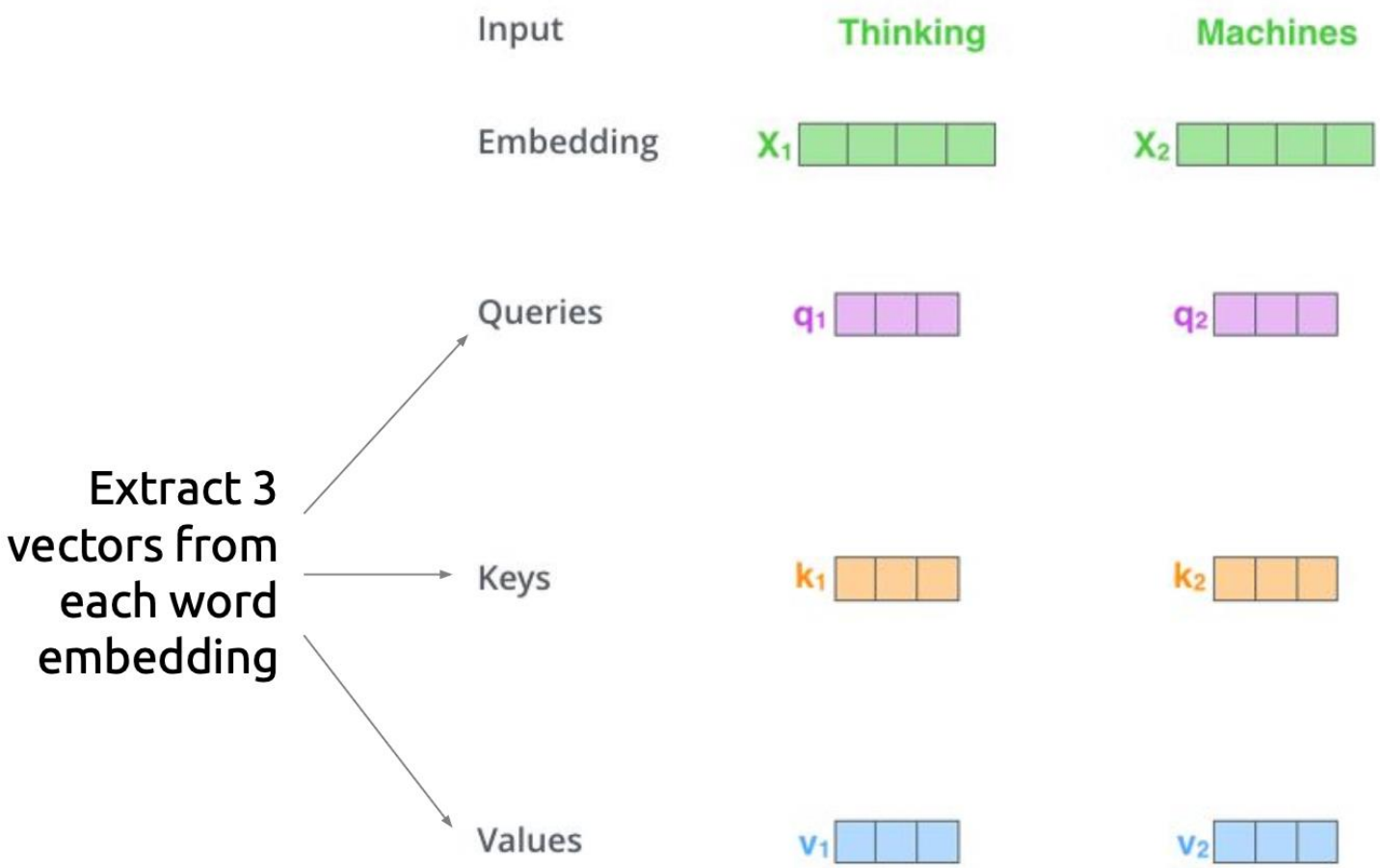
To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

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

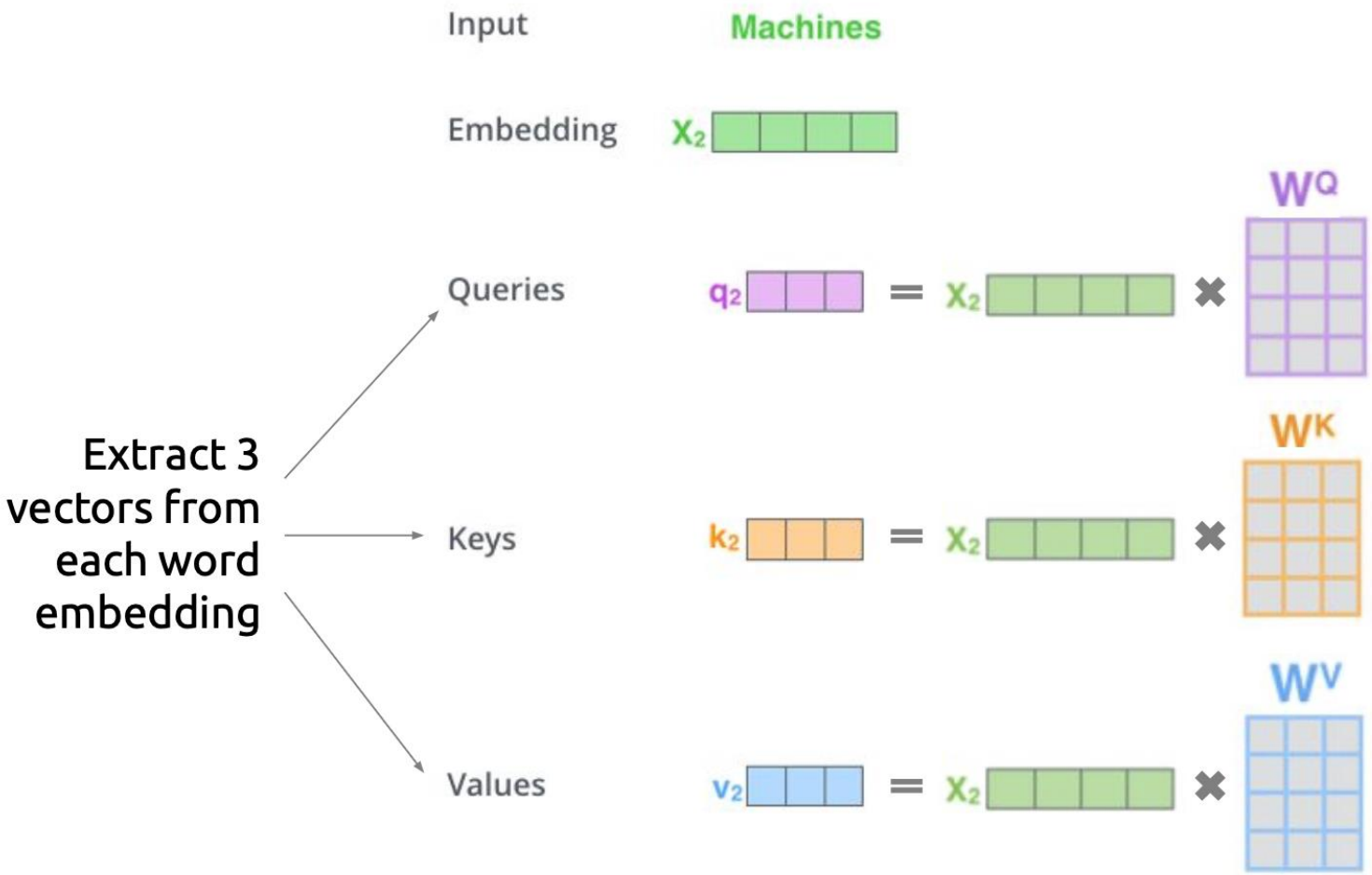


Self-Attention: Details



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

Self-Attention: Details



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

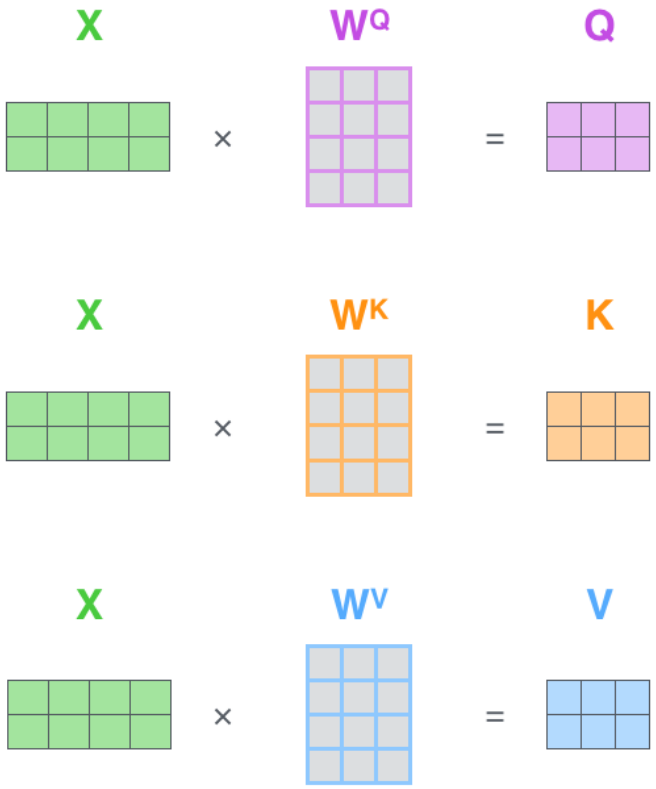
How do we get these weight matrices?

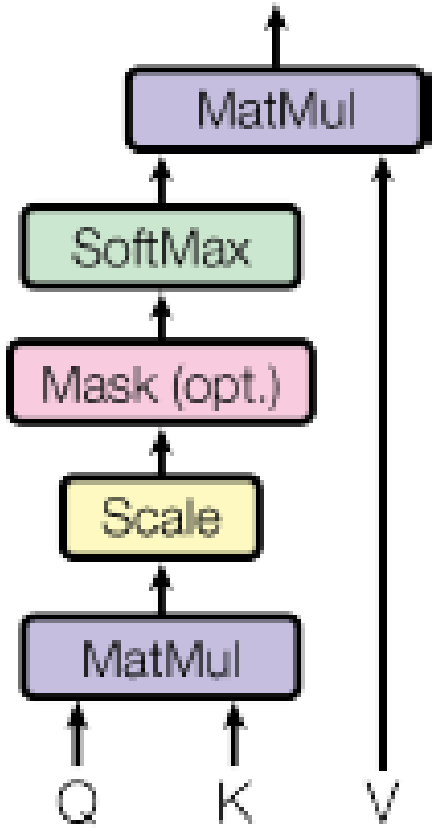
These matrices are the ***trainable parameters*** of the network

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

Scaled Dot Product Attention

Generate Q, K, V, by multiplying word embedding X by weight matrix (i.e., pass through a fully connected layer)





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

Multi-Headed Attention

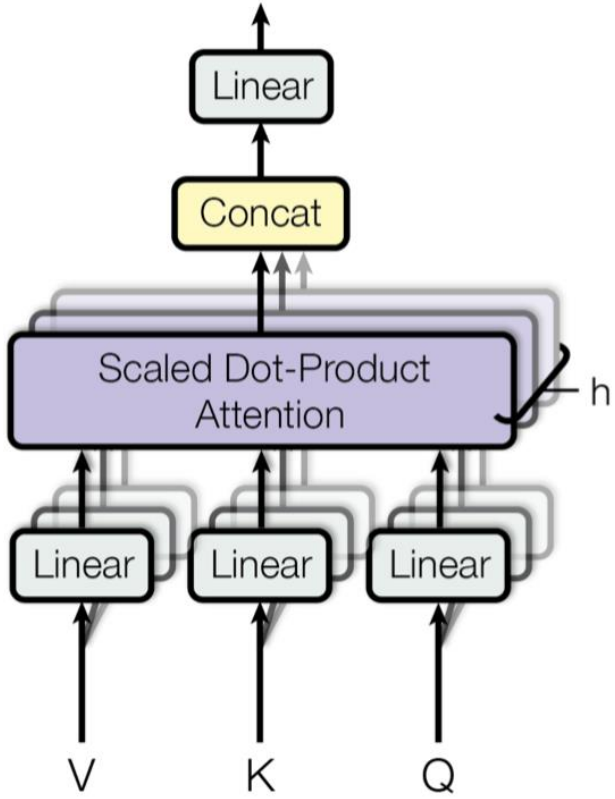
Similar to convolutional layers with multiple filters, we can have "multi-headed attention"

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_\text{h})W^0$$

Where: $head_i = \text{Attention}(\text{QW}_\text{i}^\text{q}, \text{KW}_\text{i}^\text{k}, \text{VW}_\text{i}^\text{v})$

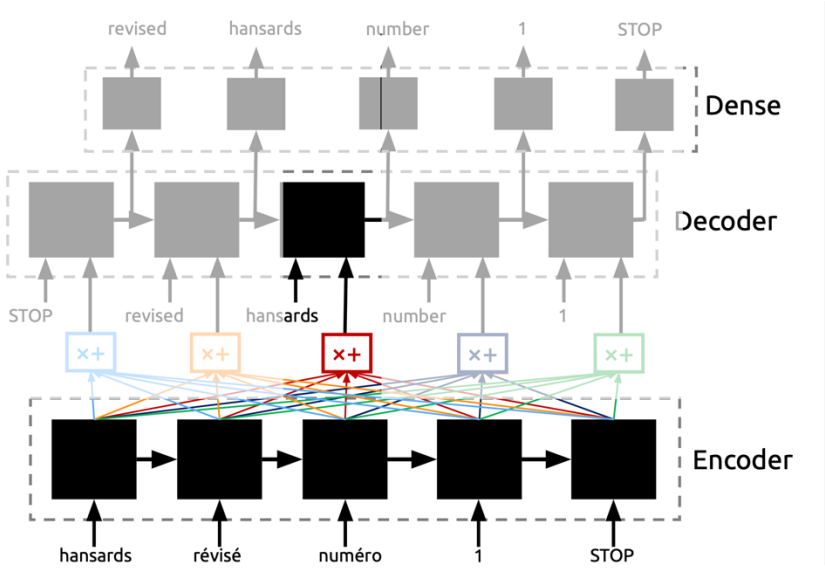
Projected Attention: Project (Q, K, V) with learned parameters W^q, W^k, W^v

Separate learned fully-connected layer for each head *i* and for each of (Q, K, V)



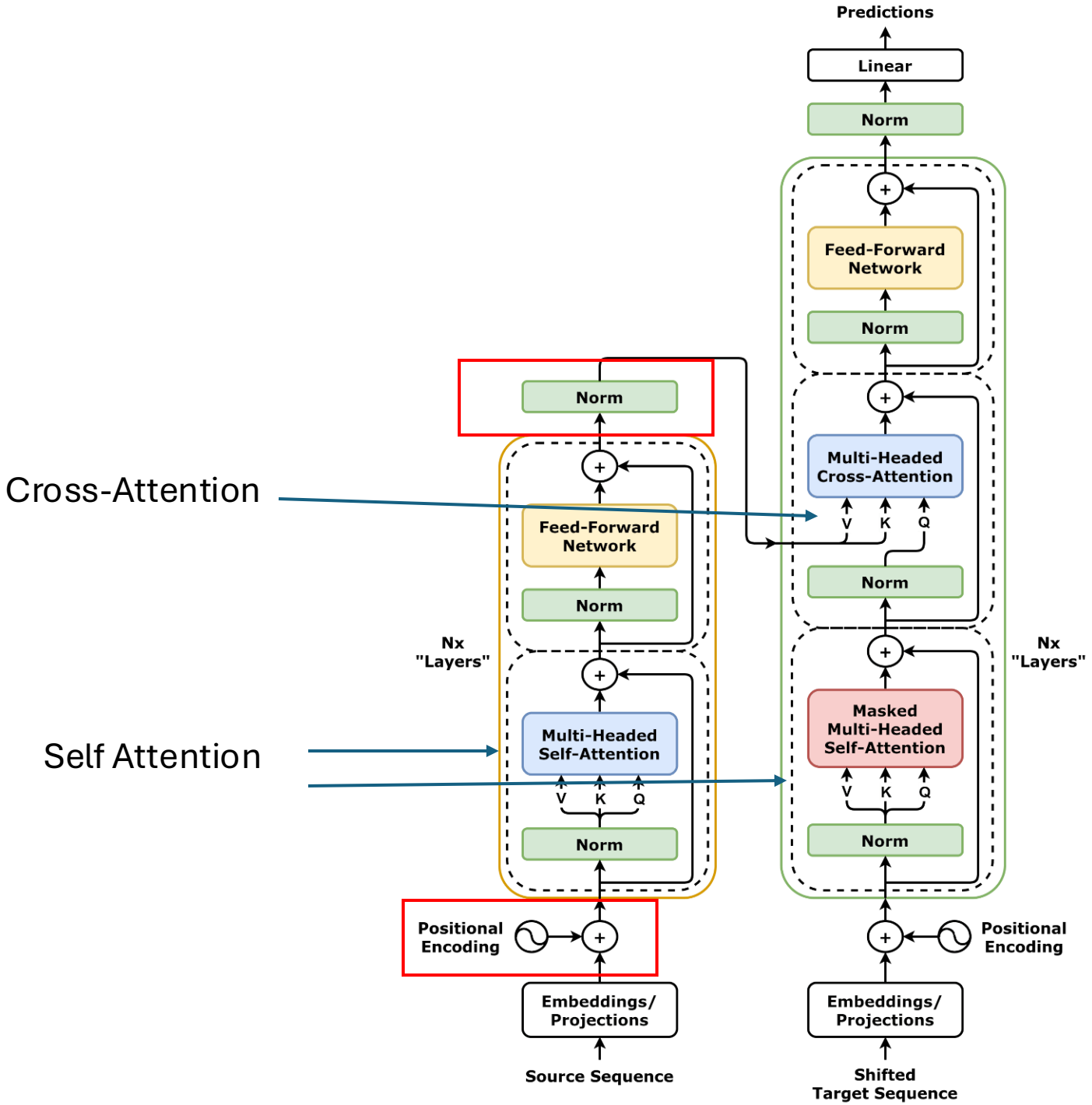
Cross Attention

- Self-Attention is how much each input “attends” to every other input
- Cross-Attention is how much every output “attends” to every input (i.e., our original motivation for attention)



Transformer

What's left?
1. Position Encoding
2. Norm



Position Encoding

- Part of the original motivation behind using RNNs for sequence data was to incorporate the *structure* of the problem (i.e., that order matters in the sequence)
- Attention (so far) does not care about the order that inputs arrive, all the operations are symmetric
- How can we get our networks to realize that there is an ordering to our inputs without using RNNs?

What's the difference between:

1. The cow jumped over the moon
2. Over the jumped cow moon the

Word order matters!

Positional Encoding

Want: a unique encoding (vector) for every value of position

Option 1:

Make this a learnable parameter.

Learn an embedding for every position a word can be in (i.e., 1, 2, 3,… max_length)

Option 2:

Do what “Attention is All you Need” did

$$PE(\text{pos}, 2i) = \sin(\text{pos}/10000^{\frac{2i}{d}})$$

$$PE(\text{pos}, 2i+1) = \cos(\text{pos}/10000^{\frac{2i}{d}})$$

Positional Encoding

Option 1:

Make this a learnable parameter.

Learn an embedding for every position a word can be in (i.e., 1, 2, 3,… max_length)

Option 2:

Do what “Attention is All you Need” did

$$PE(\text{pos}, 2i) = \sin(\text{pos}/10000^{\frac{2i}{d}})$$

$$PE(\text{pos}, 2i+1) = \cos(\text{pos}/10000^{\frac{2i}{d}})$$

1. Fix a size for the output of your Position embedding d (has to match size of embeddings/projections)

Positional Encoding

Option 1:

Make this a learnable parameter.

Learn an embedding for every position a word can be in (i.e., 1, 2, 3,… max_length)

Option 2:

Do what “Attention is All you Need” did

$$PE(\text{pos}, 2i) = \sin(\text{pos}/10000^{\frac{2i}{d}})$$

$$PE(\text{pos}, 2i+1) = \cos(\text{pos}/10000^{\frac{2i}{d}})$$

1. Fix a size for the output of your Position embedding d (has to match size of embeddings/projections)

2. At each index of the encoding, evaluate the proper formula (i.e., even positions use sin, odd positions use cos)

Positional Encoding

"We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE(pos+k) can be represented as a linear function of PE(pos)."

-- Vaswani et al. 2017, Attention is All You Need

$$PE(pos + k) = A \cdot PE(pos)$$

Any Linear function
can be represented as
a matrix multiplication

Positional Encoding

For every pair of adjacent values in the position encoding

$$A \cdot \begin{pmatrix} \sin(c \cdot pos) \\ \cos(c \cdot pos) \end{pmatrix} = \begin{pmatrix} \sin(c \cdot (pos + k)) \\ \cos(c \cdot (pos + k)) \end{pmatrix}$$

$$A = \begin{pmatrix} \cos(\mathrm{c} \cdot k), \ \sin(c \cdot k) \\ -\sin(\mathrm{c} \cdot k), \cos(c \cdot k) \end{pmatrix}$$

Normalization

BatchNorm: Normalize outputs of neurons based on mean and standard deviation of the values for a batch of inputs

Issues:

1. RNNs don’t batch well (LayerNorm came before Transformers)
2. When batches are small, mean and standard deviation can vary highly

LayerNorm: Instead of normalizing based on the batch dimension, normalize the outputs of a layer based on the mean and standard deviation of the outputs of that layer.

LayerNorm

Two learnable parameters, because… why not, it's deep learning…

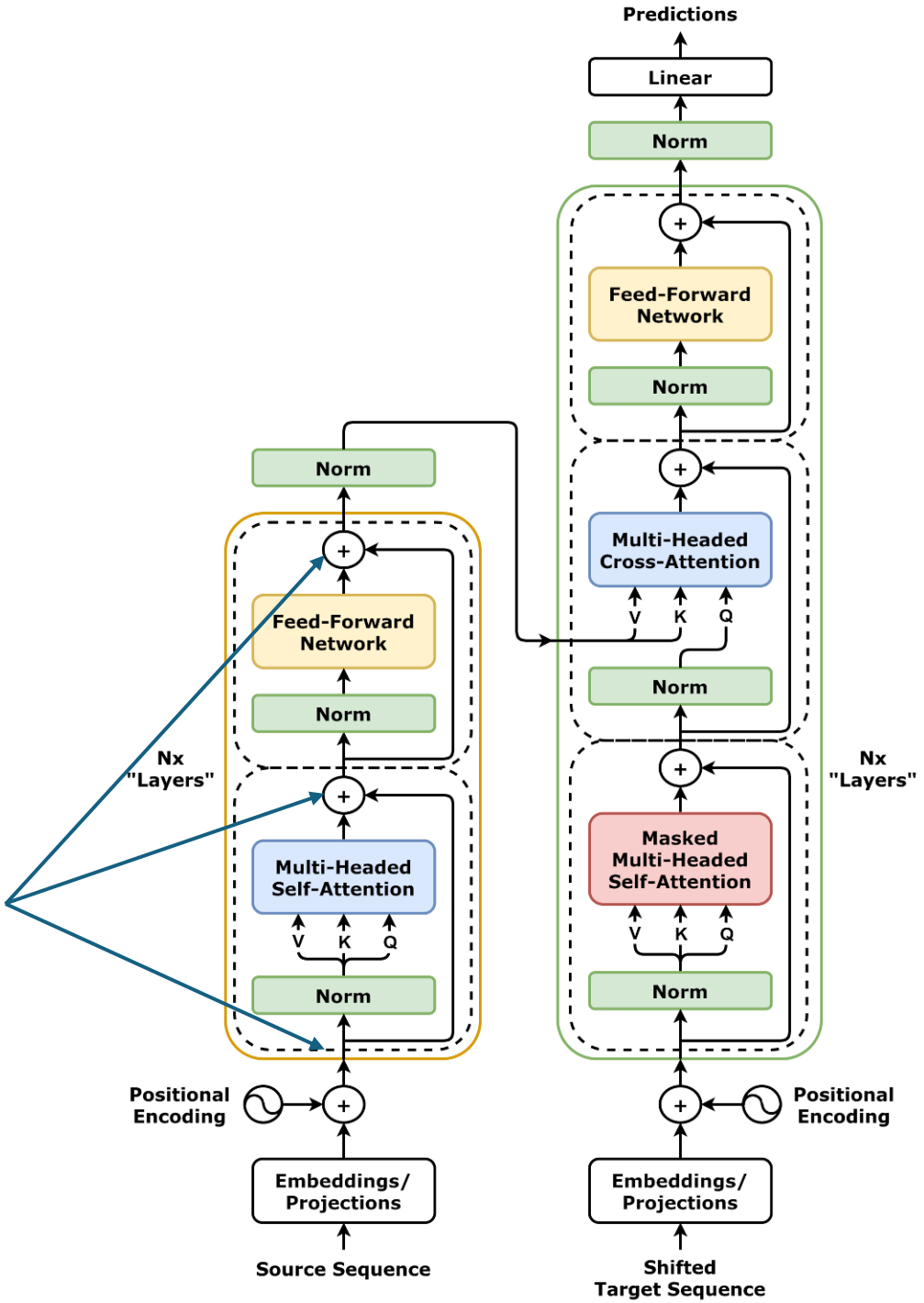
$$y = \frac{x - E[x]}{std(x)} * \gamma + \beta$$

Normalization

Outputs before normalization (i.e., inputs to LayerNorm layer)

Transformer

All intermediate outputs have same dimension, only one hyperparameter for dimension (many more for number of heads, number of encoder/decoders Nx)



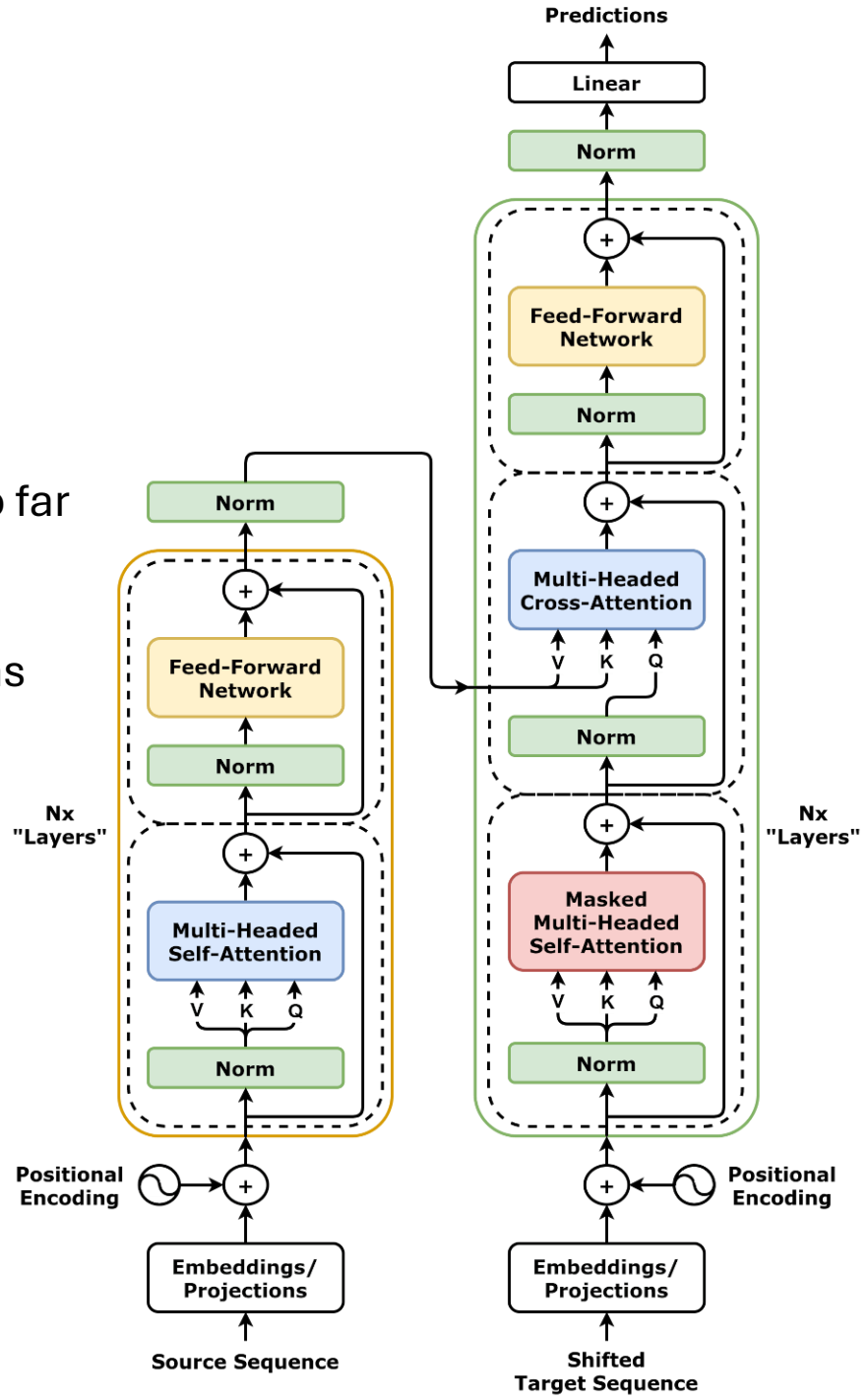
Transformer

Transformers are… complicated

They have many unique components, unlike networks we've covered so far

- CNNs can be large, but they only really have 2 components: Convolutions and linear layers
- The internals of an RNN can be complicated, but it's 3 or 4 operations

Why do they work so well?



Transformer Strengths

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

1. Attention is faster than RNNs ($n \ll d$)
2. Don't require sequential operations, like RNNs
3. Have a lower path length (how many operations does it take for information about words *n* distance apart to spread to each other)

Transformer Strengths

params	dimension	n heads	n layers	learning rate	batch size	n tokens
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

Table 2: **Model sizes, architectures, and optimization hyper-parameters.**

Deep Networks do better. More parameters are better. Transformers have many learnable parameters.

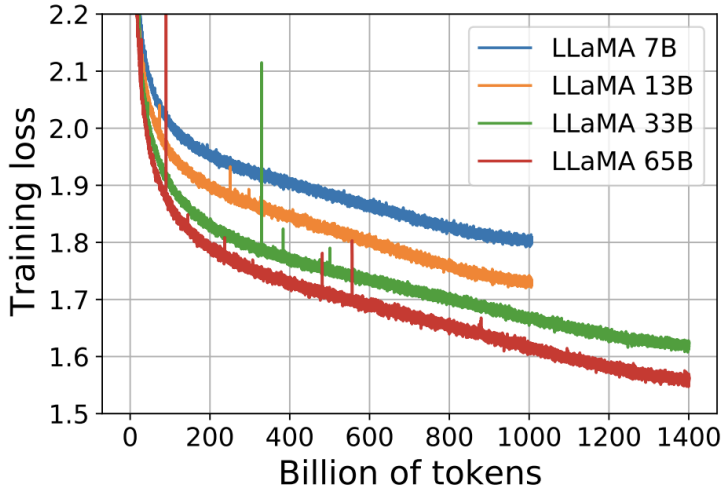


Figure 1: **Training loss over train tokens for the 7B, 13B, 33B, and 65 models.** LLaMA-33B and LLaMA-65B were trained on 1.4T tokens. The smaller models were trained on 1.0T tokens. All models are trained with a batch size of 4M tokens.

LLaMA: Open and Efficient Foundation Language Models

Transformer Weaknesses

- Transformers are not good at small scale tasks, they have many parameters and tend to overfit easily.
- There are really not that many hyperparameters in transformers, just the number of attention heads, number of layers, and embedding size.
  - Hard to get them to not overfit

Why is this weakness not actually a problem?