CONVERSATIONAL AI LAB

ConversationalAILab.org

A forum for Brown folks to showcase the cool things they've **built** using large language models, generative AI, and related technologies, and a **research** center to advance these technologies.

Launch meeting on Monday, March 11, 5-6 pm, Macmillan Hall 115

Agenda

Organizational meeting and talk by Isaac Wecht (Masters, DSI) on *Using LangChain and OpenAI to create a question answering system*.

CSCI 1470/2470
Spring 2024

Ritambhara Singh

March 08, 2024
Friday

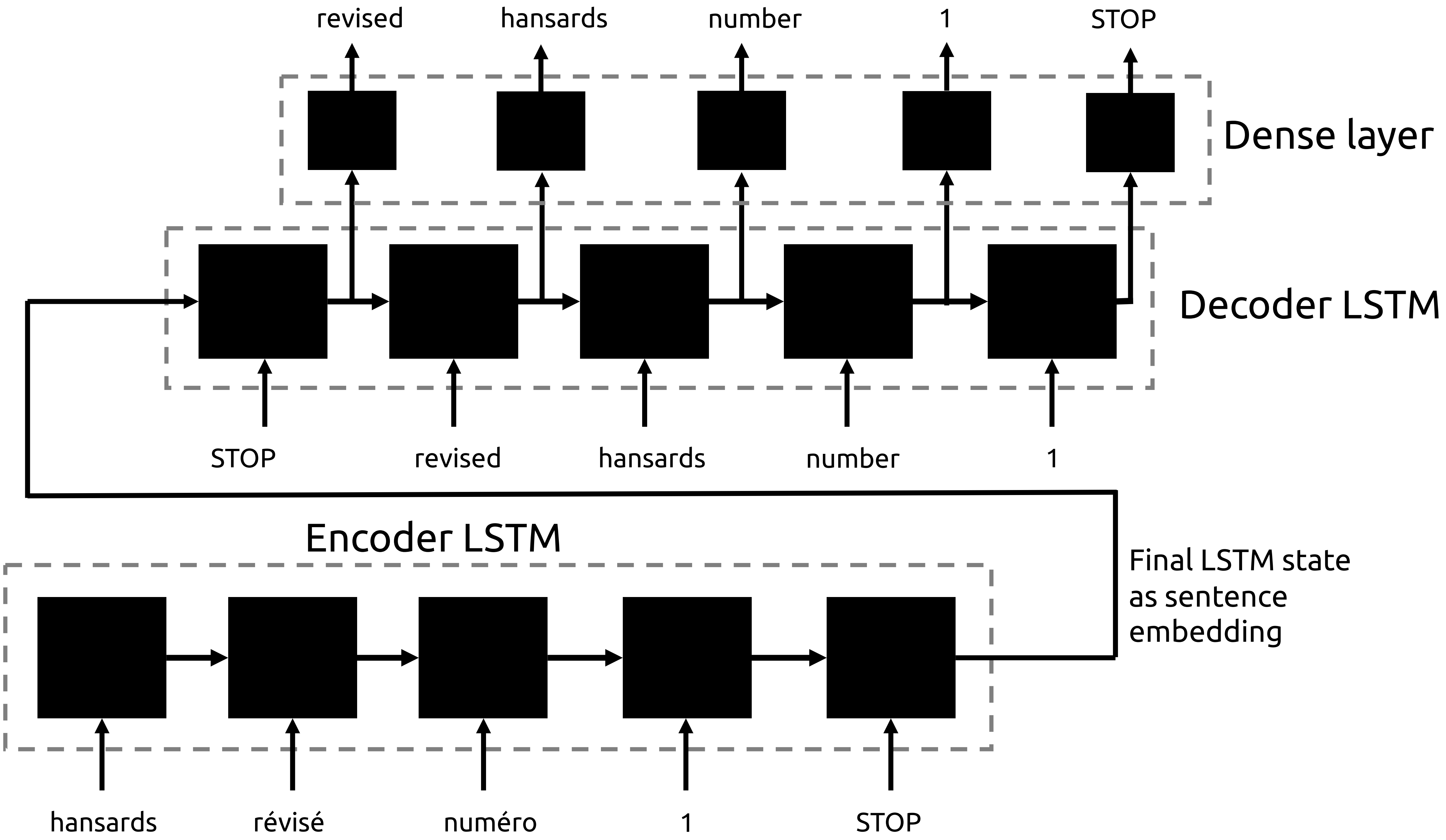
Attention

Deep Learning

ChatGPT prompt “minimalist landscape painting of a deep underwater scene with a blue tang fish in the bottom right corner”

Review: seq2seq Models

- Last time, we saw an encoder-decoder architecture for sequence-to-sequence learning

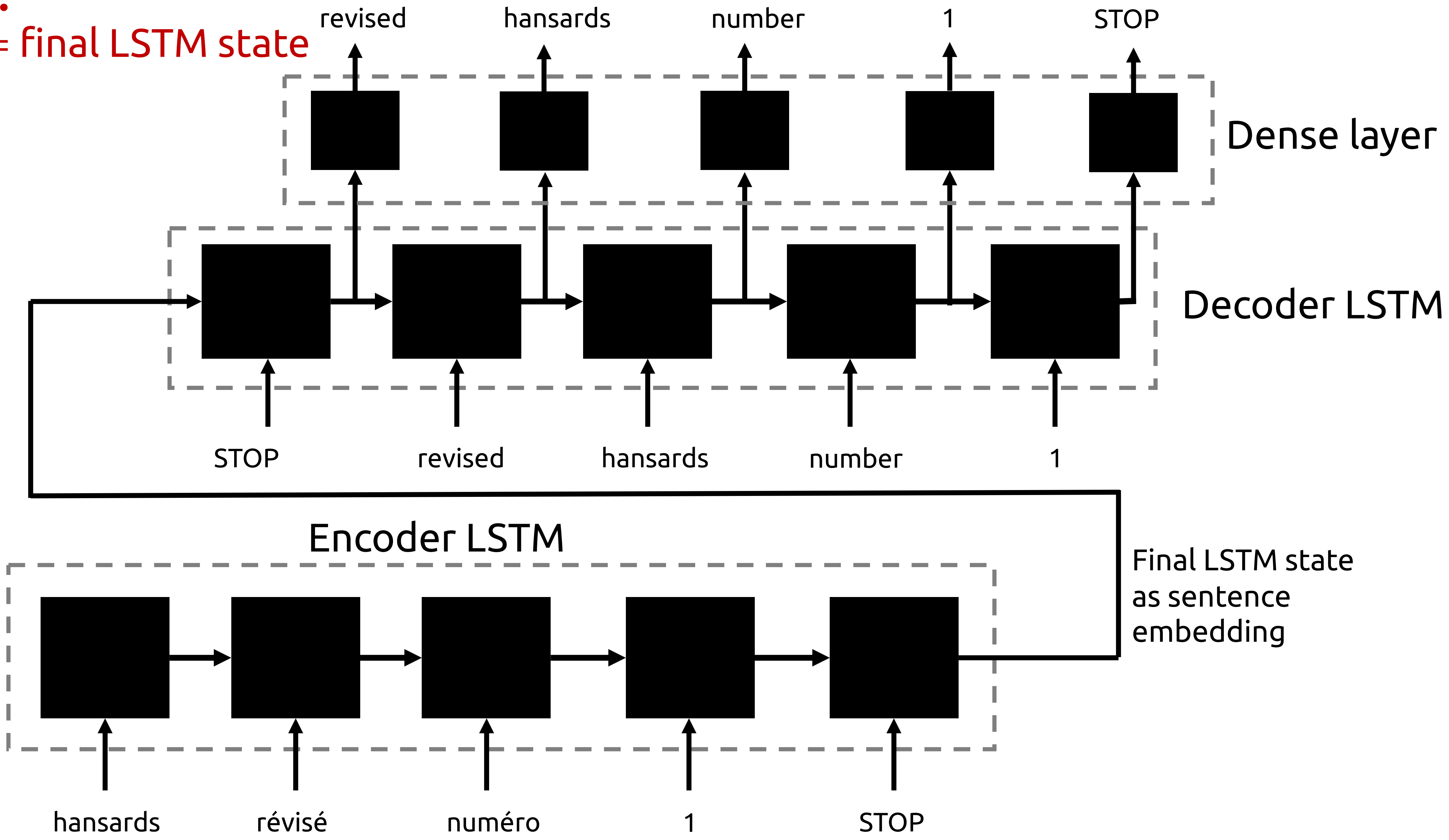

revised
hansards
number
1
STOP
Dense layer
Decoder LSTM
STOP
revised
hansards
number
1
Encoder LSTM
Final LSTM state
as sentence
embedding
hansards
révisé
numéro
1
STOP

Review: seq2seq Models

- Last time, we saw an encoder-decoder architecture for sequence-to-sequence learning
- We also saw how, instead of initializing the decoder with the final state of the encoder, we could use the sum of all encoder states

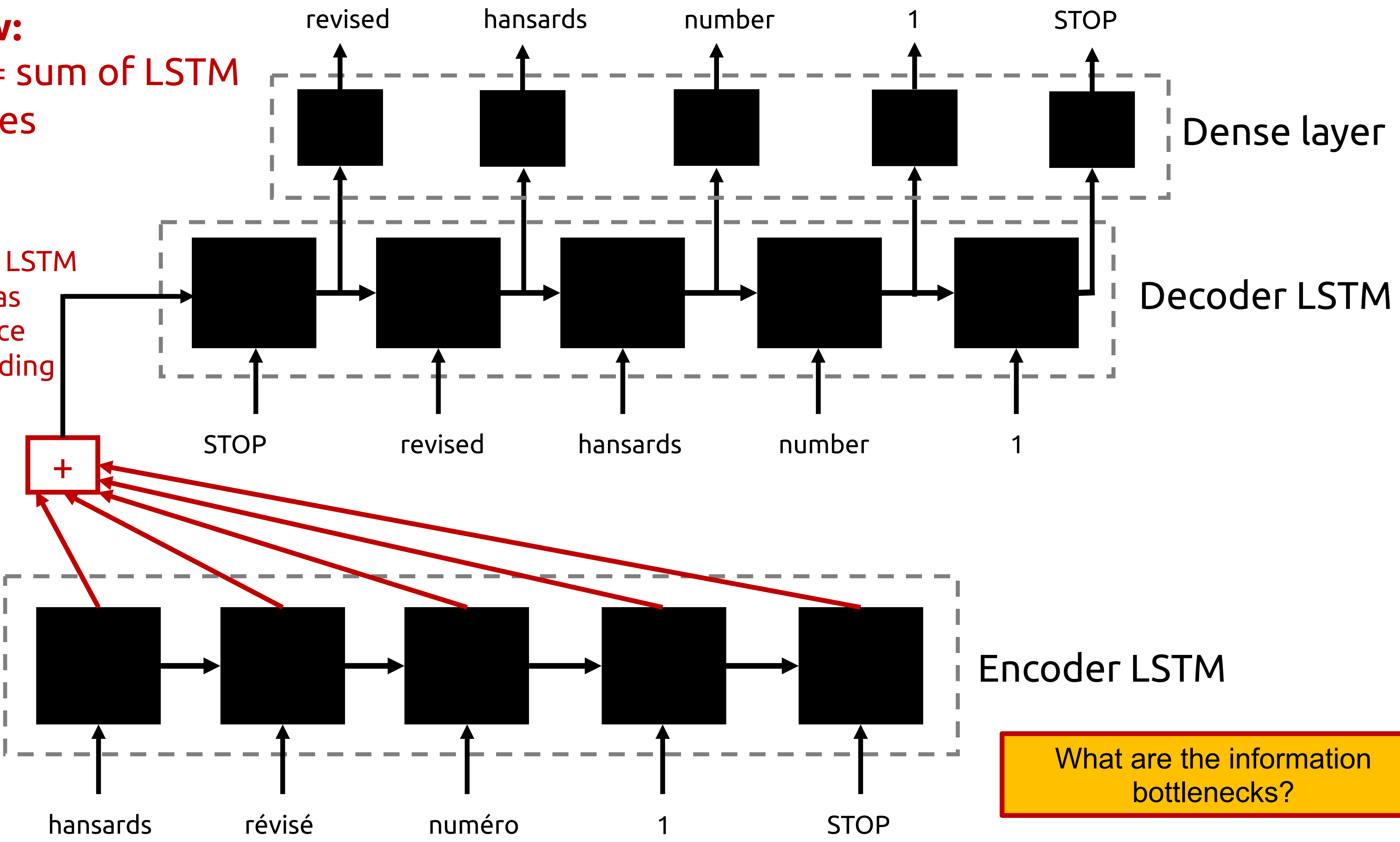
Old:

E_s = final LSTM state



New:
E_s = sum of LSTM states

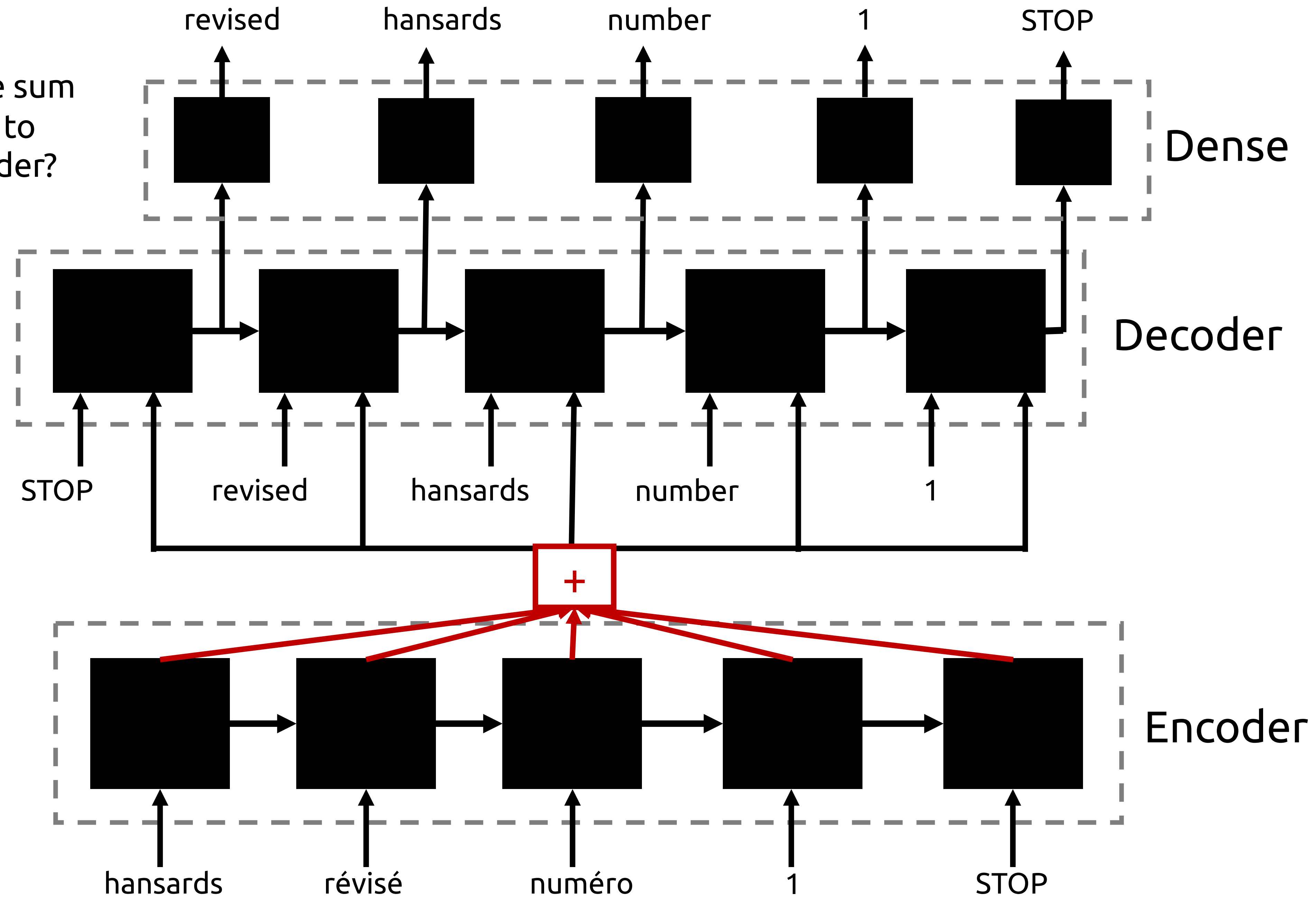
Sum of LSTM states as sentence embedding



What are the information bottlenecks?

What if the decoder LSTM forgets the sentence embedding?

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

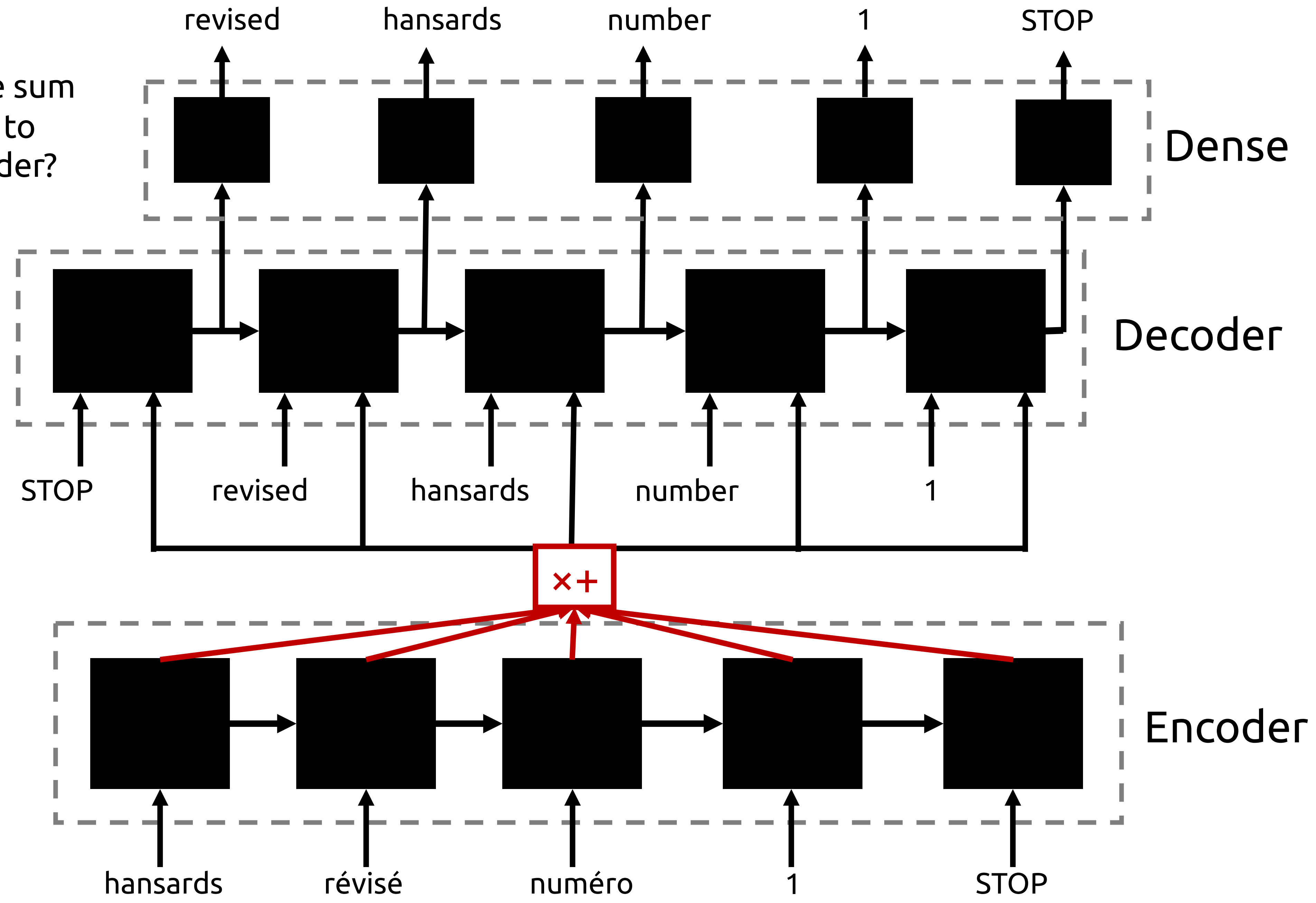


Just summing might not be sufficient! Different words in the source have different importance for the target

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



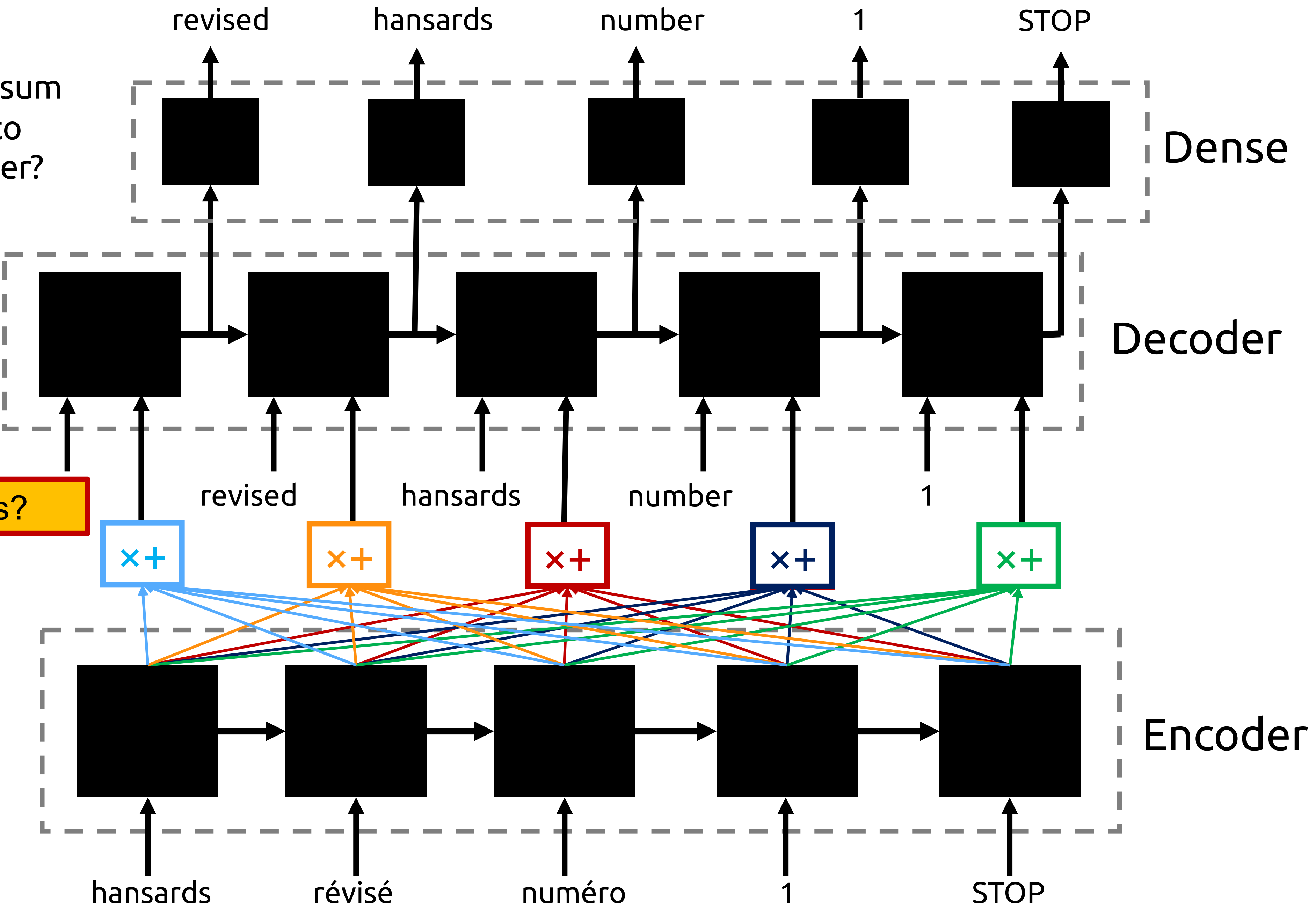
What if we passed the sum of our encoder states to ***every cell*** in the decoder?

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

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”

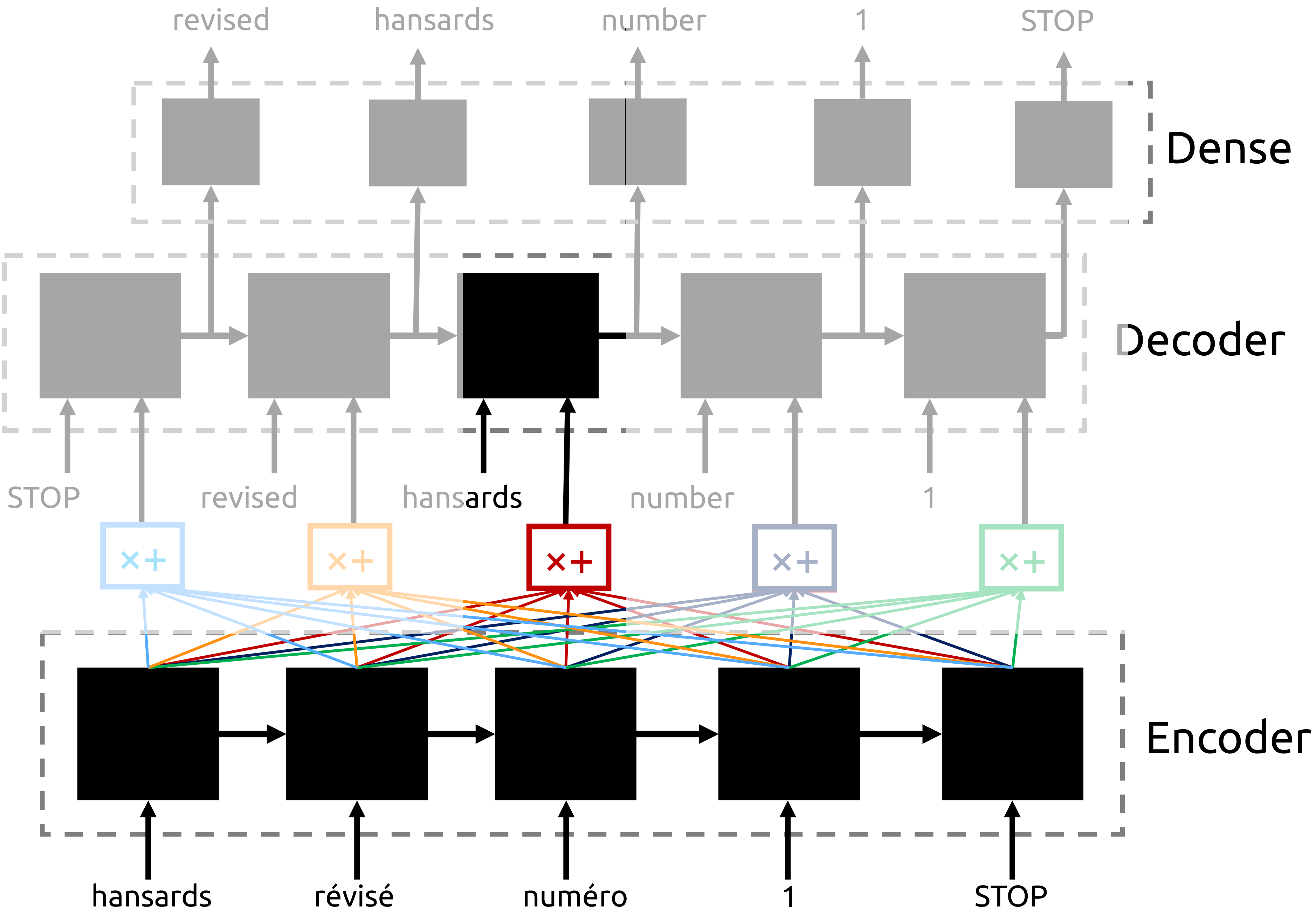
This idea of passing each cell of the decoder a weighted sum of the encoder states is called ***attention.***

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

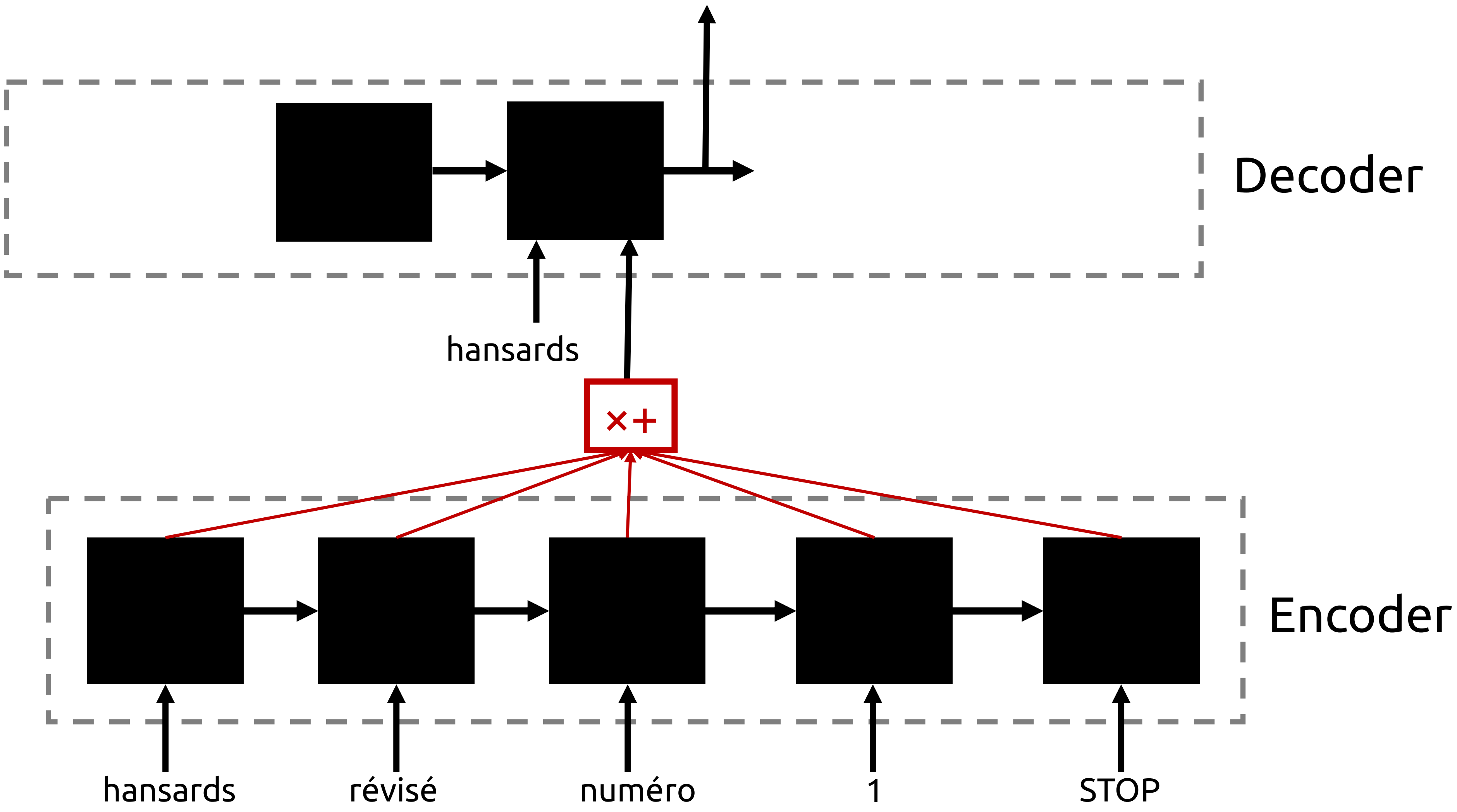
“Attention” - intuition

“Park”

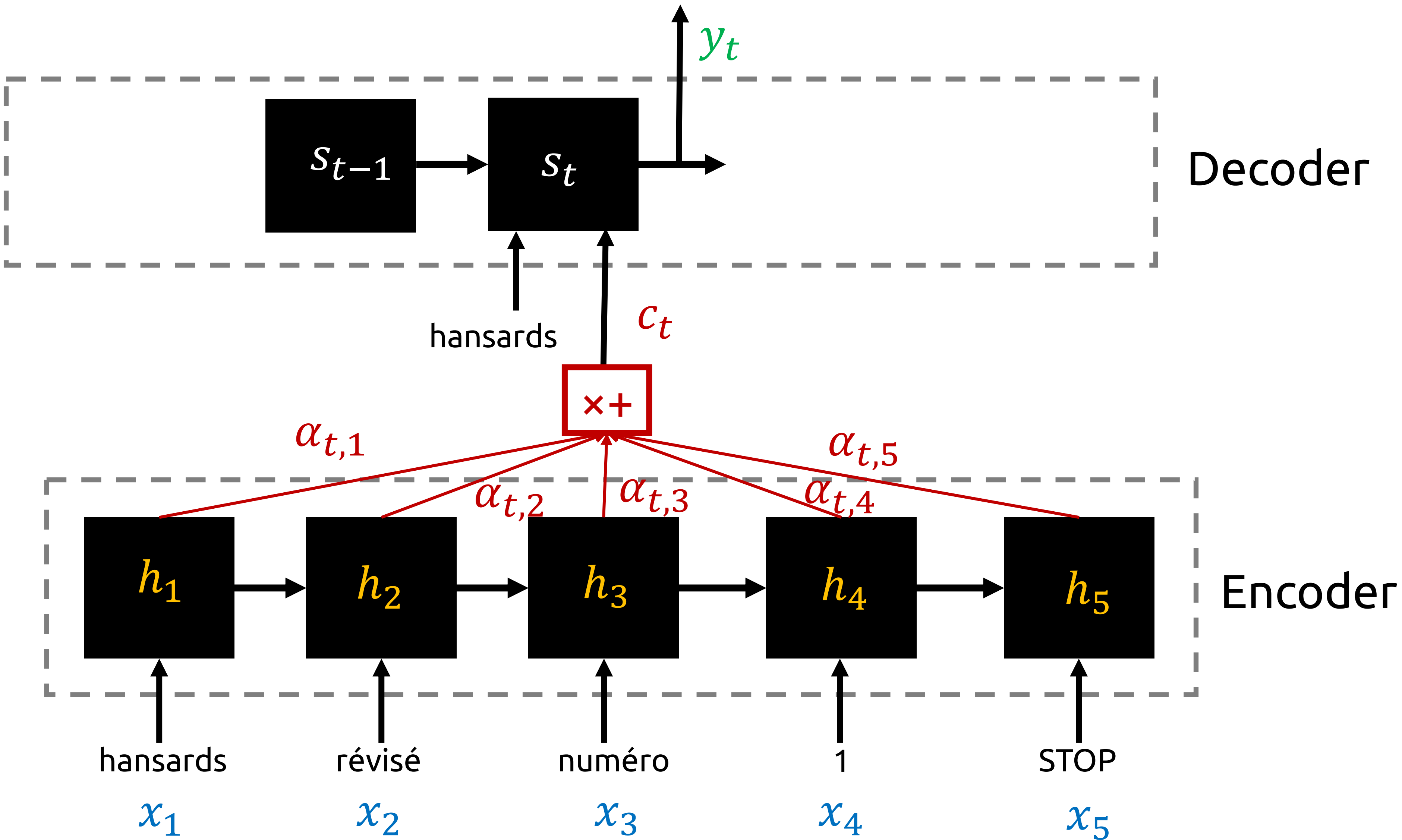
How about we let model learn what is relevant for a particular output


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

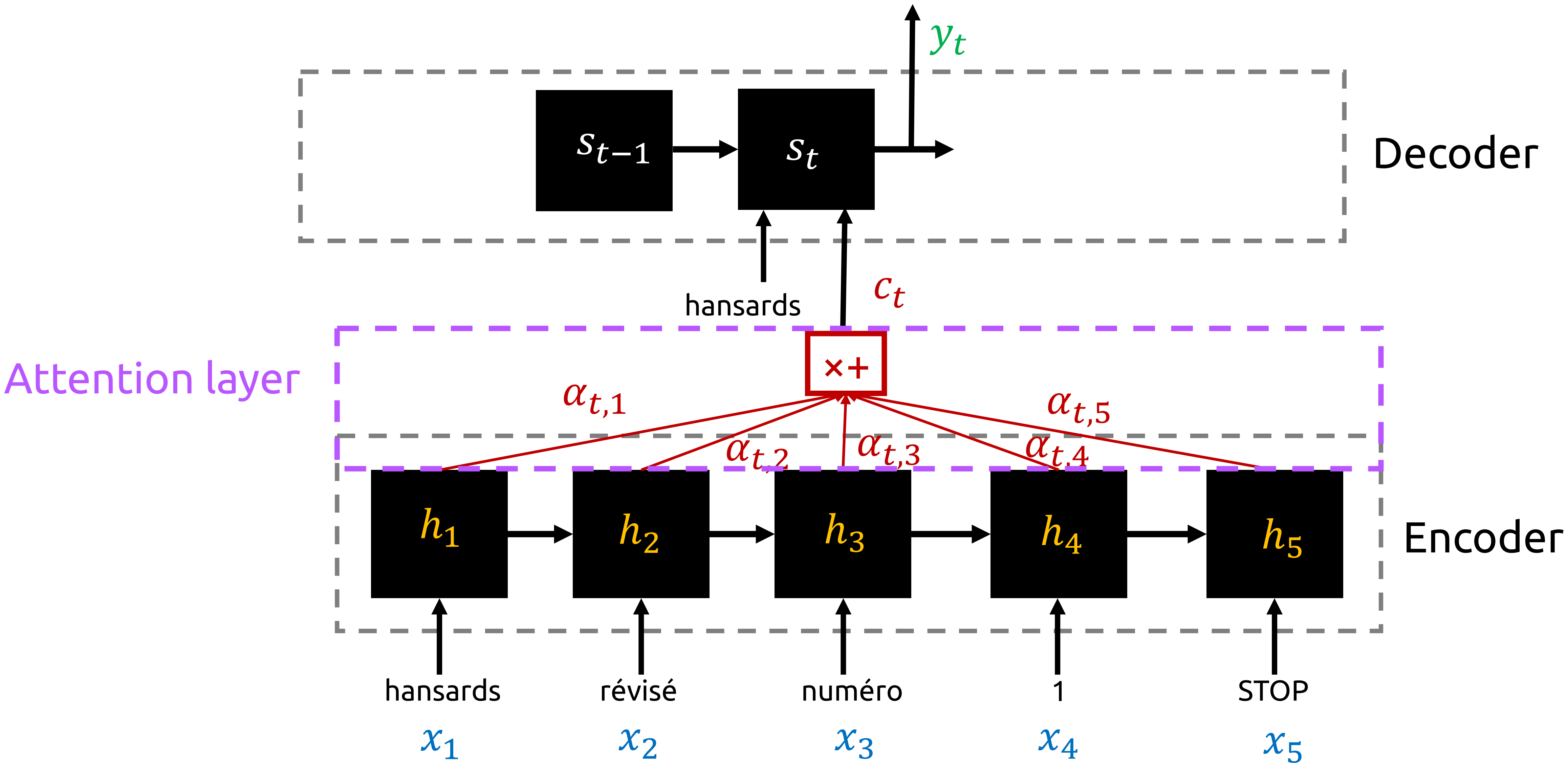
Attention - implementation



Attention - implementation



Attention - implementation



$$\mathbf{x} = [x_1, x_2, \ldots, x_n]$$
$$\mathbf{y} = [y_1, y_2, \ldots, y_m]$$

$$\mathbf{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i$$

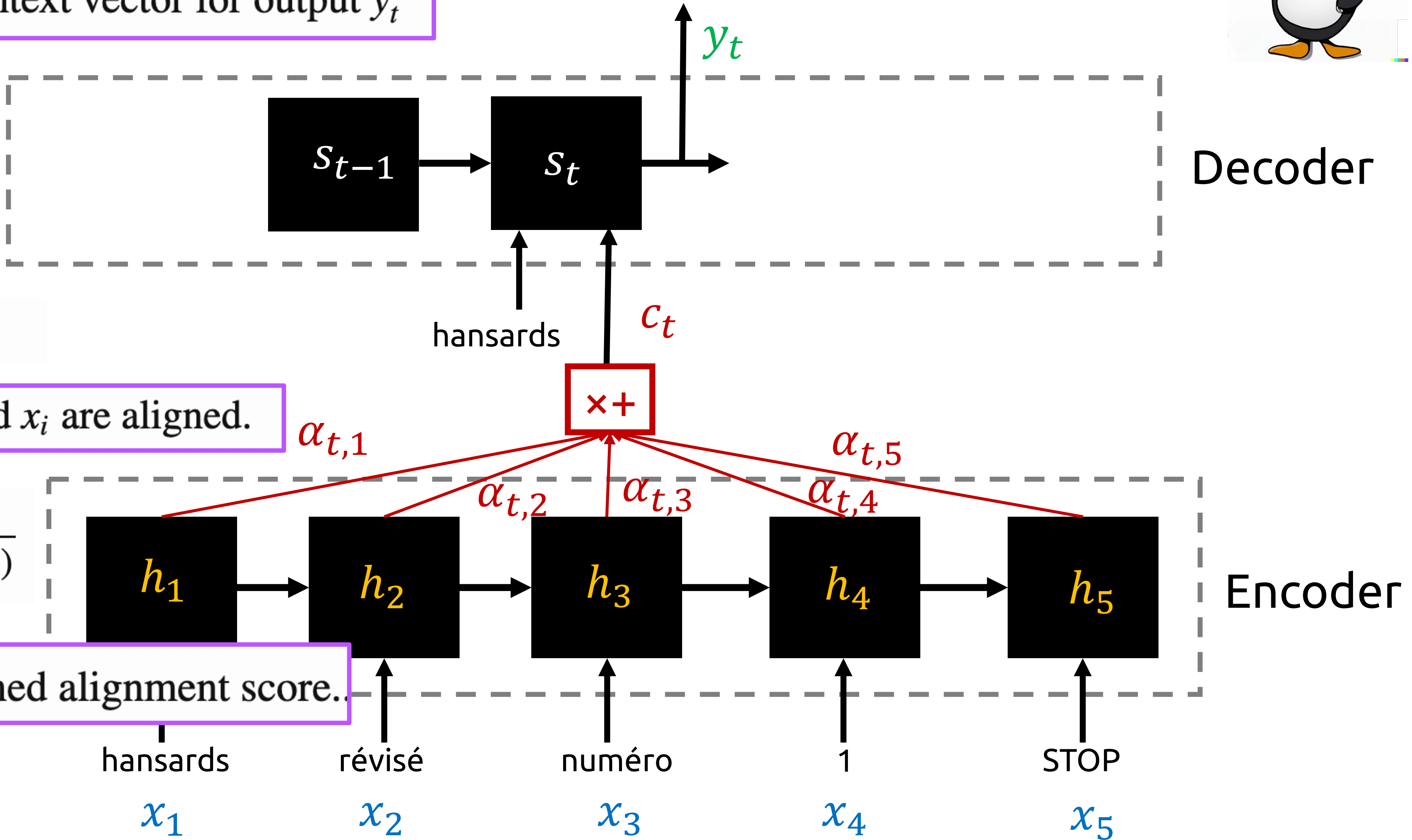
Context vector for output y_t

$$\alpha_{t,i} = \text{align}(y_t, x_i)$$

How well two words y_t and x_i are aligned.

$$= \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

Softmax of some predefined alignment score.



Any questions?

Attention alignment score functions

$$\alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

Softmax of some predefined alignment score..

How well two words y_t and x_i are aligned.

How to measure this? Any ideas?

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

Attention alignment score functions

$$\alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

Softmax of some predefined alignment score..

How well two words y_t and x_i are aligned.

General attention:

$$score(s_{t-1}, h_i) = s_{t-1}^T \boldsymbol{W}_a h_i$$

Attention alignment score functions

$$\alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

Softmax of some predefined alignment score..

How well two words y_t and x_i are aligned.

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

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

Attention types

$$\alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

Softmax of some predefined alignment score..

How well two words y_t and x_i are aligned.

Name	Definition	Citation
Global/Soft	Attending to the entire input state space.	Xu2015

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

Attention types

$$\alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

Softmax of some predefined alignment score..

How well two words y_t and x_i are aligned.

Name	Definition	Citation
Global/Soft	Attending to the entire input state space.	Xu2015
Local/Hard	Attending to the part of input state space; i.e. a patch of the input image.	Xu2015; Luong2015

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

Any questions?

Attention types

$$\alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

Softmax of some predefined alignment score..

How well two words y_t and x_i are aligned.

Name	Definition	Citation
Self-Attention(&)	Relating different positions of the same input sequence. Theoretically the self-attention can adopt any score functions above, but just replace the target sequence with the same input sequence.	Cheng2016
Global/Soft	Attending to the entire input state space.	Xu2015
Local/Hard	Attending to the part of input state space; i.e. a patch of the input image.	Xu2015; Luong2015

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

$$\mathbf{x} = [x_1, x_2, \ldots, x_n]$$
$$\mathbf{y} = [y_1, y_2, \ldots, y_m]$$

$$\mathbf{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i$$

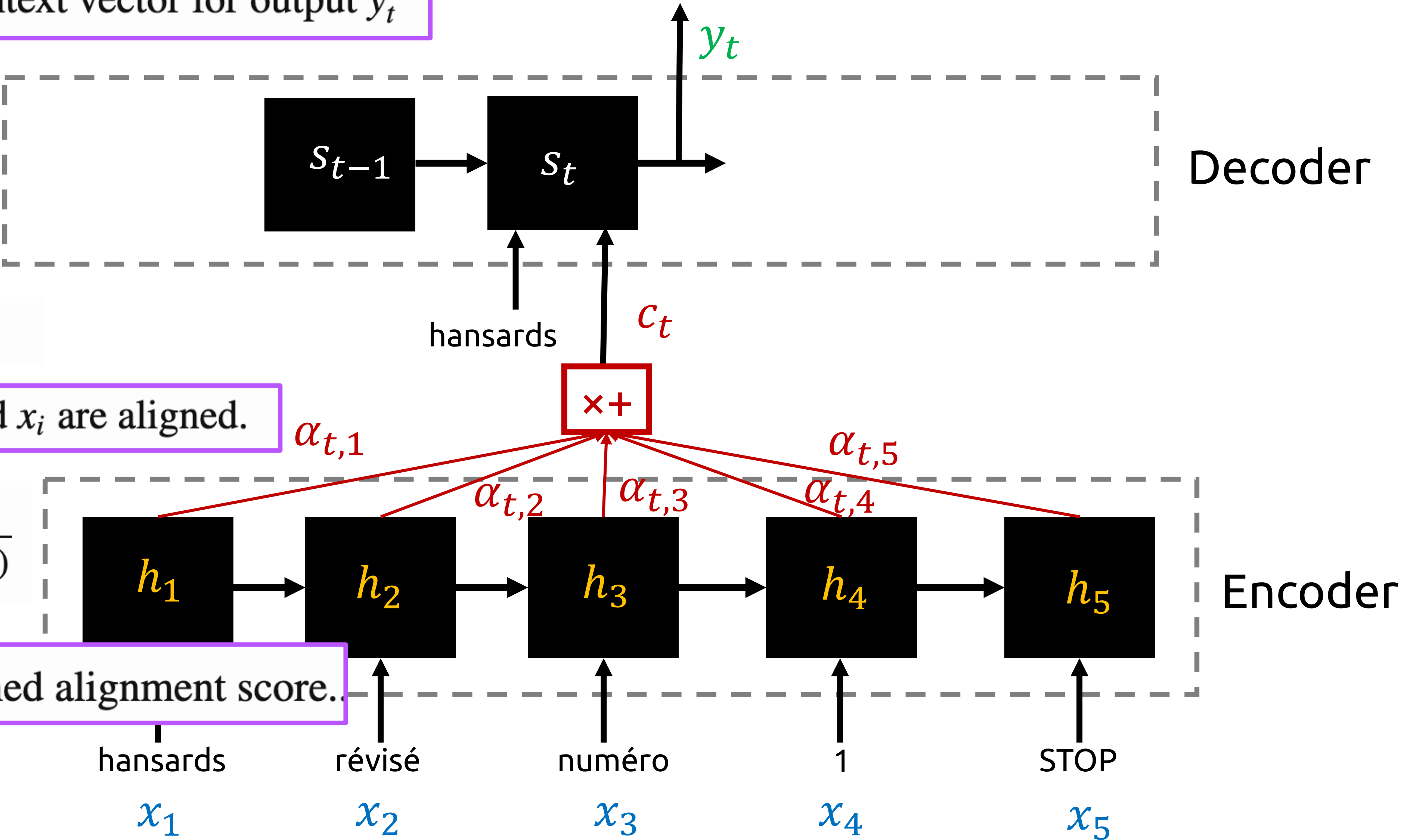
Context vector for output y_t

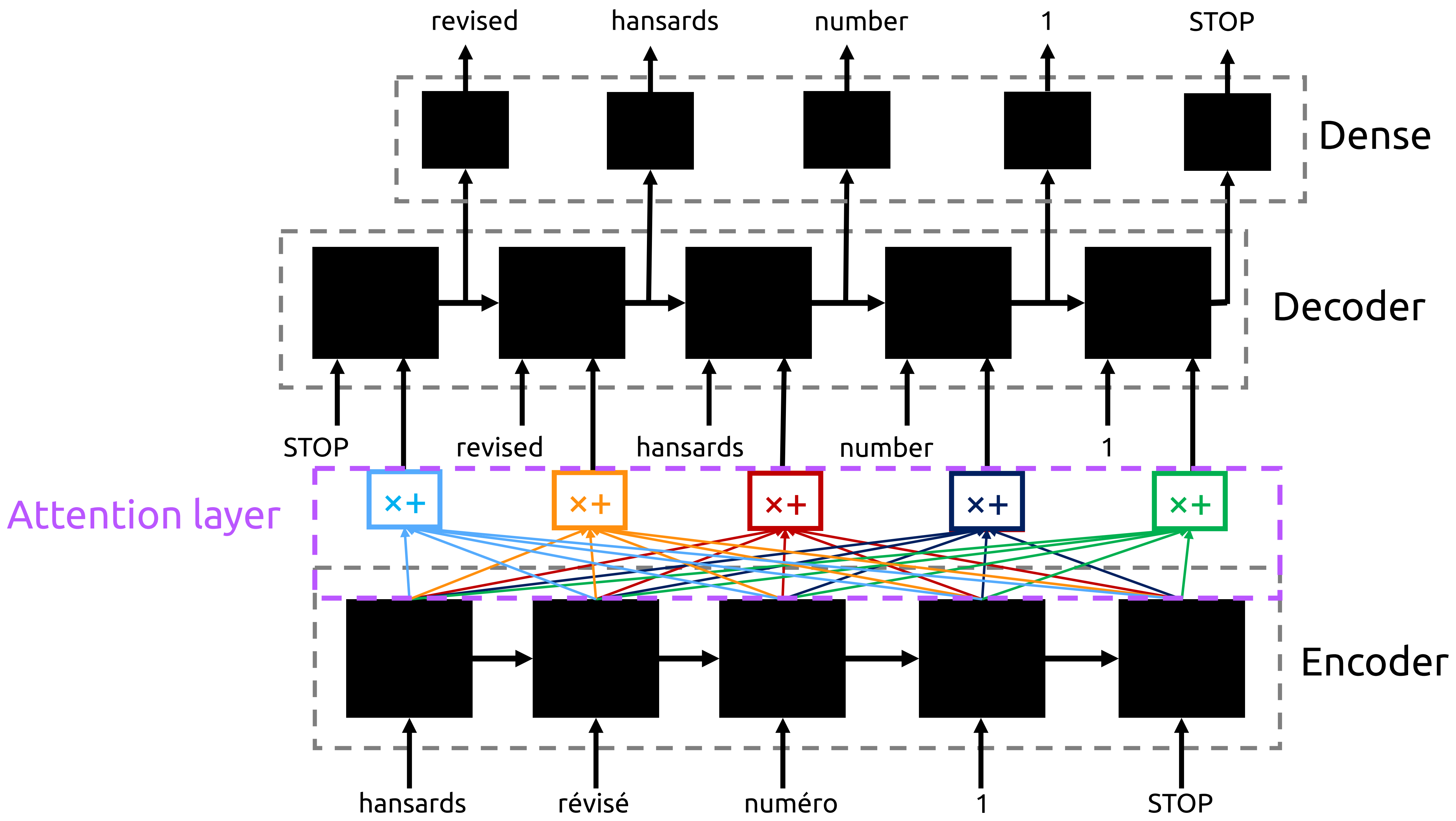
$$\alpha_{t,i} = \text{align}(y_t, x_i)$$

How well two words y_t and x_i are aligned.

$$= \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

Softmax of some predefined alignment score.




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

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

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

“Words that are similar between the input and output influence each other the most”

Another Attention Example

Target: “Der Hund bellte mich an.”

Input: “The dog barked at me.”

Attention Example

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

Input: "The dog barked at me."

[0, 1/4, 1/2, ¼, 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".

Another Attention Example

Attention weight matrix is ***another learnable parameter of the model!***

Model will re-adjust the weights

Target: “Der Hund hatte mich angebellt.”

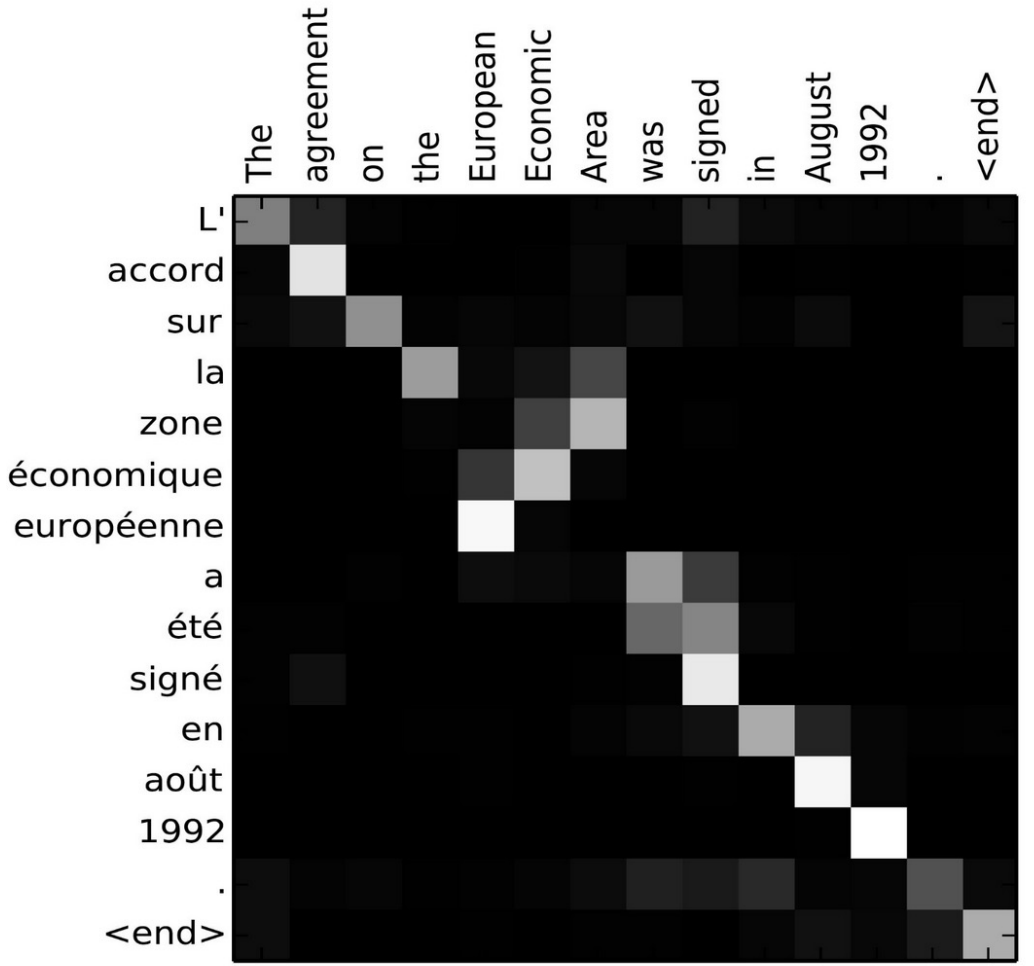
Input: “The dog had barked at me.”

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

Here, the verb portion of a past participle in German appears at the end of the sequence (What now?)

Any questions?

Attention in Language Translation



Attention helps solve the alignment problem!

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

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

Can you think of any another advantage?

- 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 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

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.

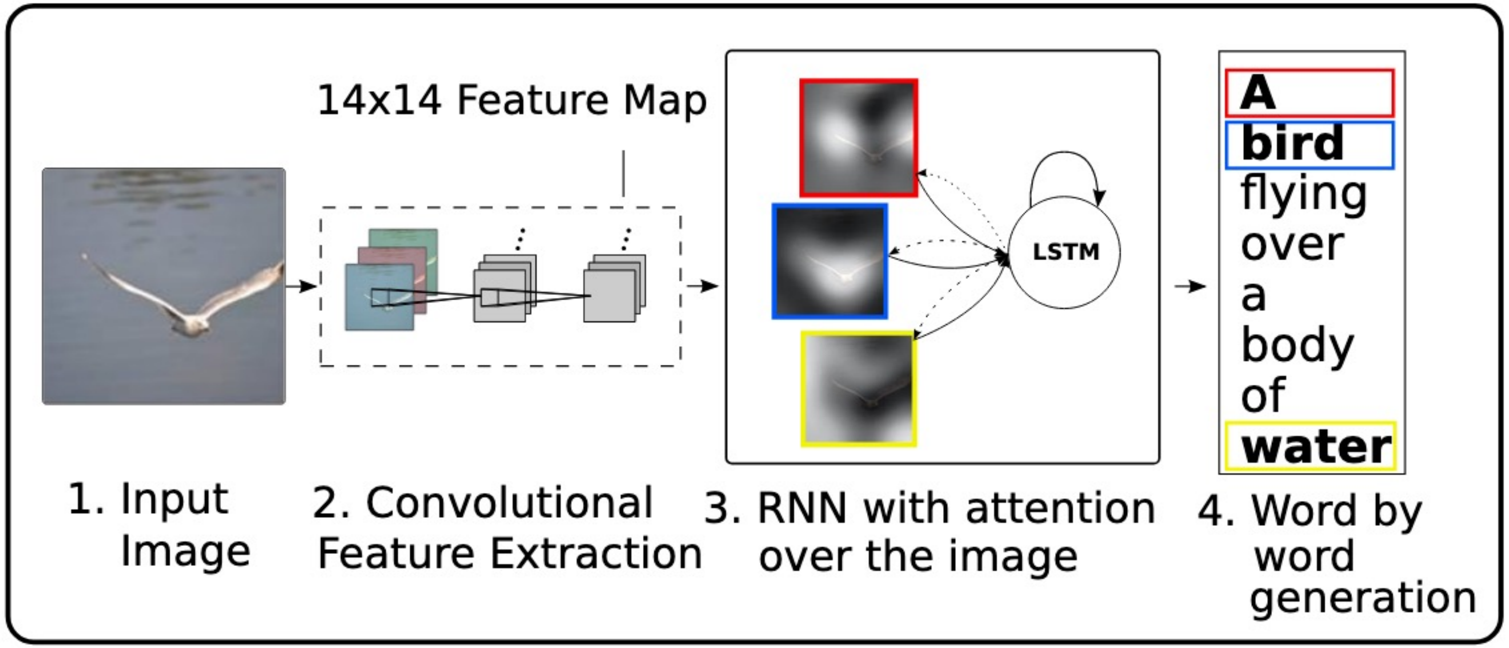
A group of people sitting on a boat in the water.

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

Think-pair-share:

How would you design this architecture with attention?

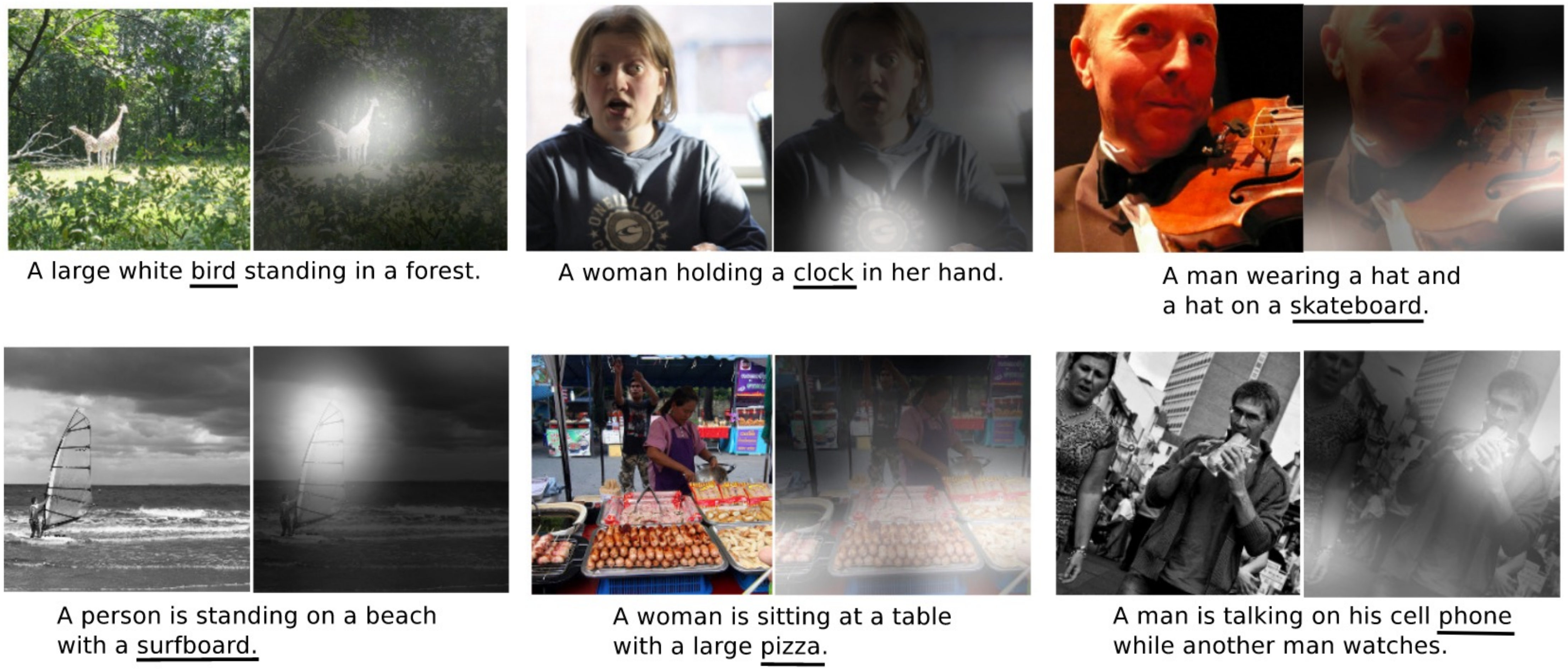
Image captioning with CNNs, RNNs, and Attention



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

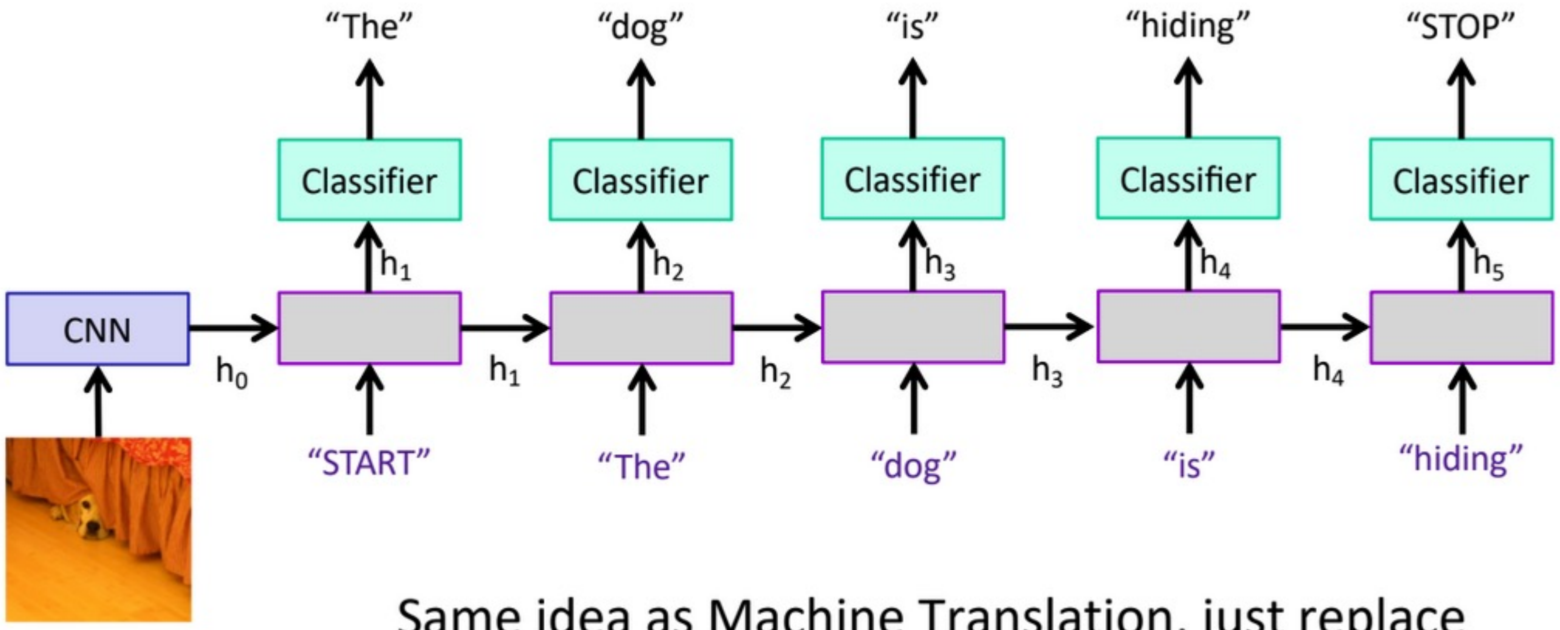
Image captioning with CNNs, RNNs, and Attention

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

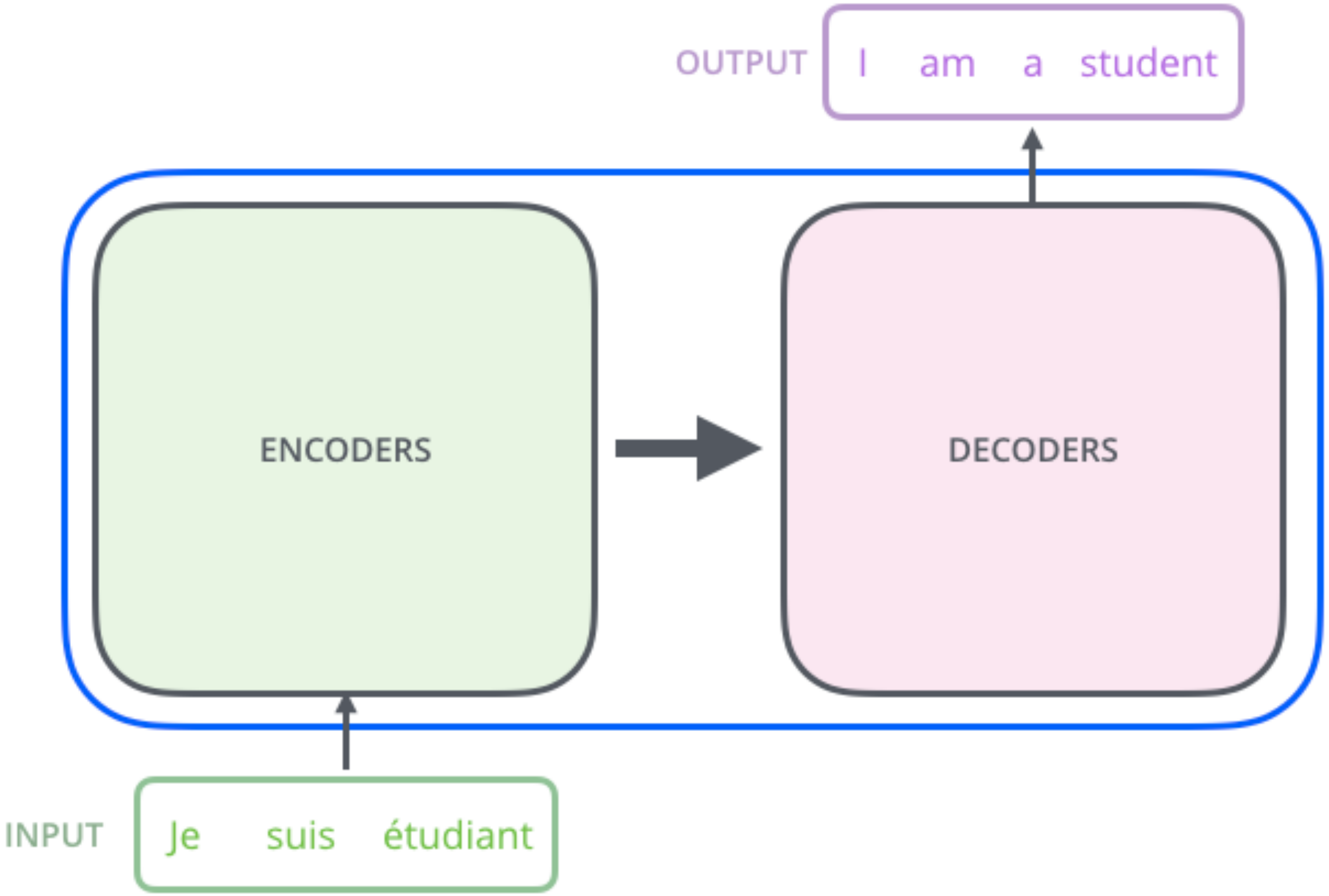
- Has no recurrent networks!
- ***Only*** uses attention

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!

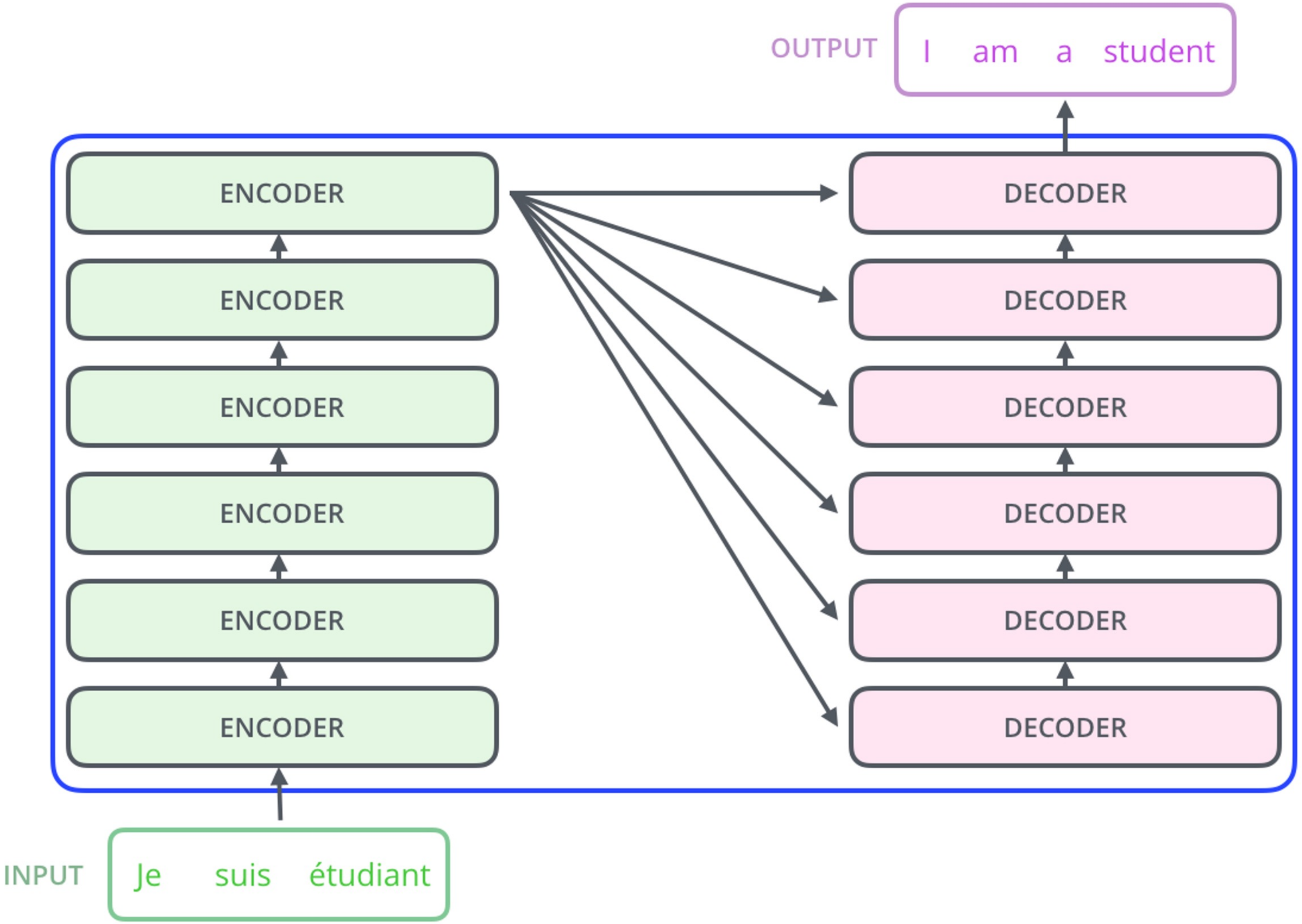


Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

To be continued in next class!

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)

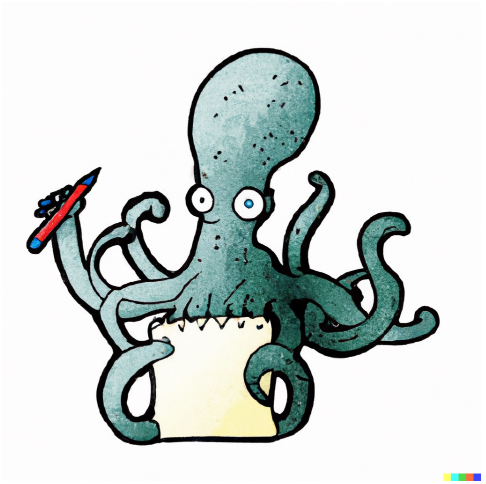


Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Recap

Attention for MT

- Attention helps remove bottlenecks in simple encoder-decoder model
- Attention score functions and types
- Attention weights are learnable



Attention as a general technique

- Interpretation
- Image captioning (HW5)
- Attention is all you need (Transformers)

A dog is standing on a hardwood floor.