Trasforming eye movement classification

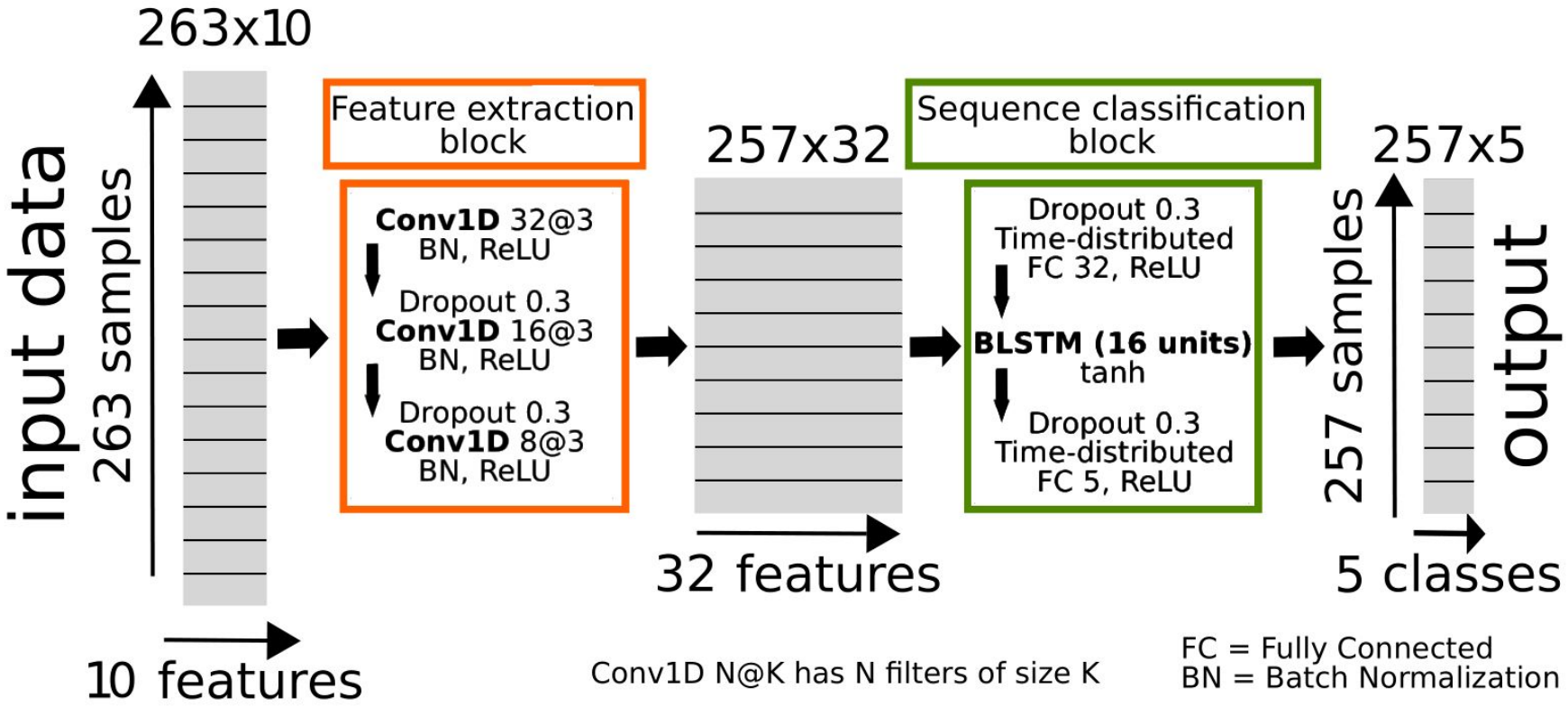
Olya Yakovleva, Irina Matyulko,
Kristina Vodorezova, Georgii Zhulikov

28.02.2019

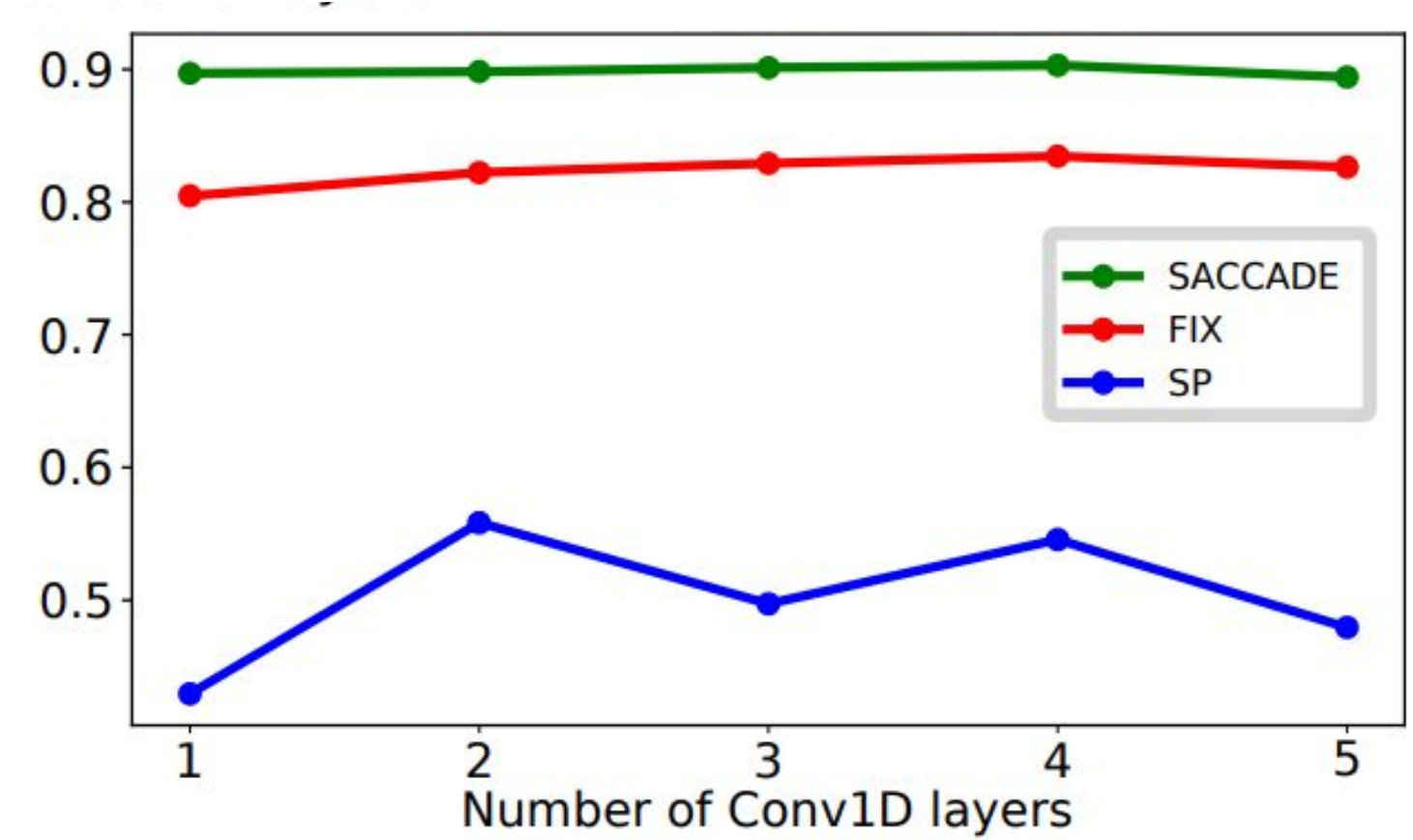
1D CNN with BLSTM for automated classification of fixations, saccades, and smooth pursuits (2018)

- 1D: One-dimensional (a linear sequence of eye movements)
- CNN: convolutional neural network
- BLSTM: Bidirectional Long short-term memory

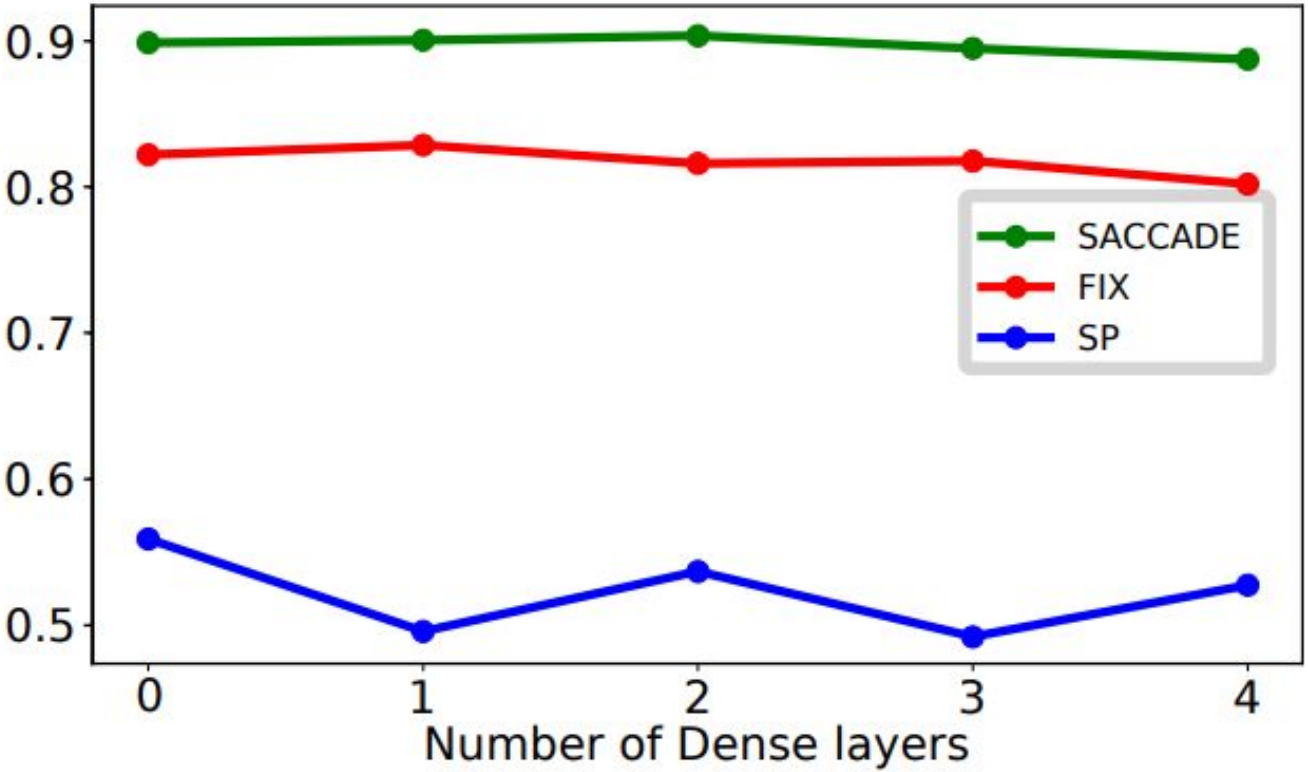
Overview (standard model):



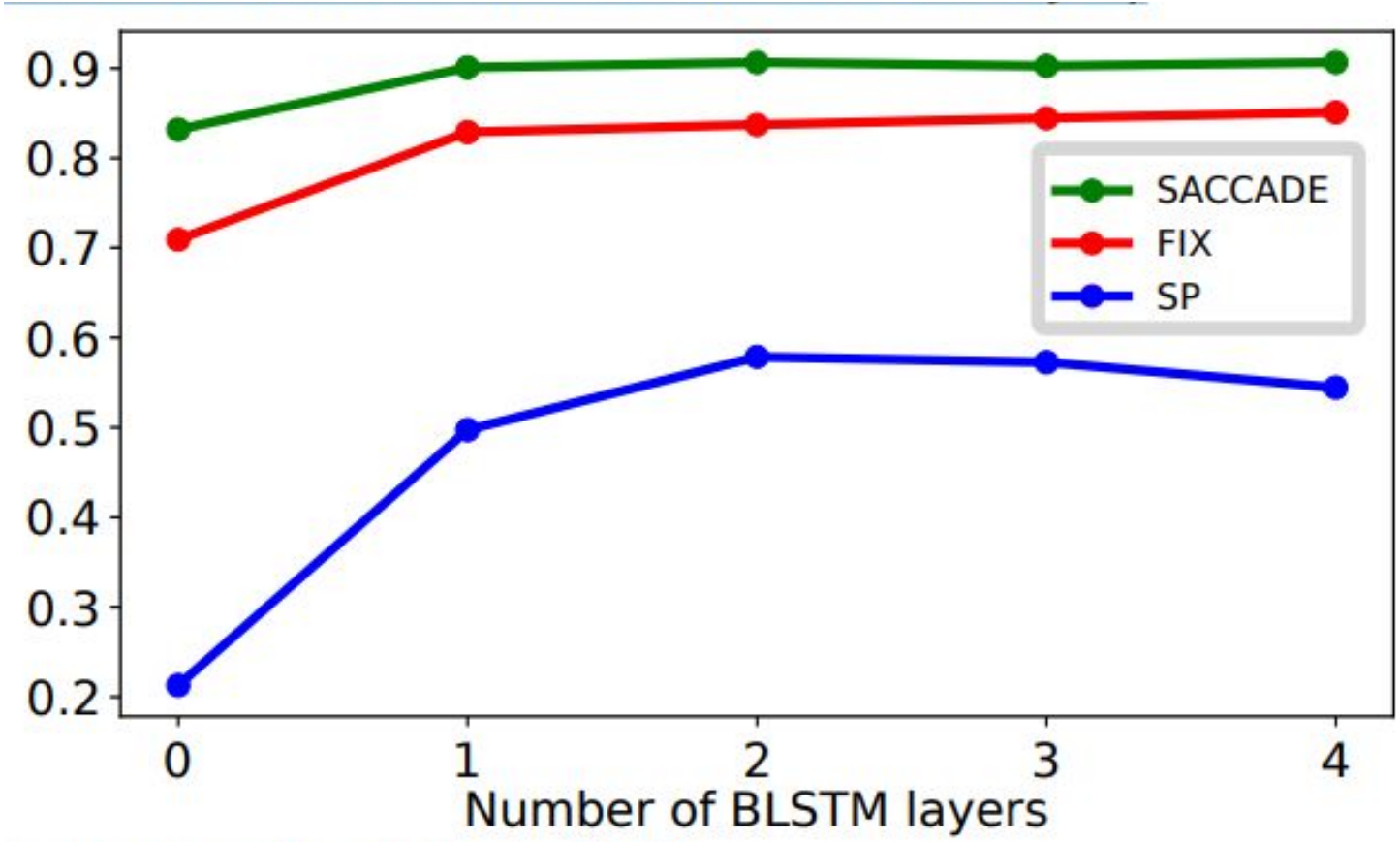
Convolutional block



Dense block



Bidirectional LSTM block



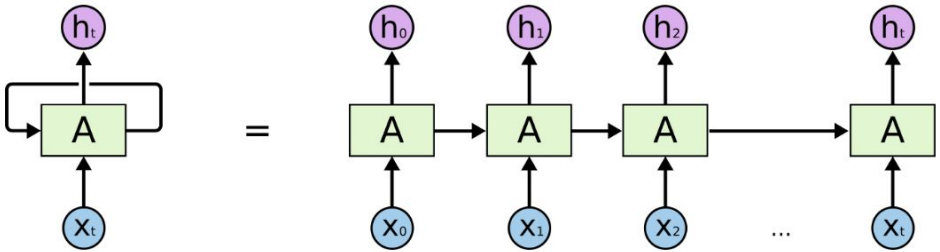
	Individual samples, F1 score			*Whole episodes, F1 score*		
	Fixations	Saccades	Pursuits	Fixations	Saccades	Pursuits
Final model	**93.8%**	**89.6%**	**70.7%**	**91.5%**	**94.9%**	**62.9%**
[Agtzidis et al. 2016]	88.6%	86.4%	64.6%	81.0%	88.4%	52.7%
I-VMP (optimized)	90.9%	68.0%	58.1%	79.2%	81.5%	53.1%
[Larsson et al. 2015]	91.2%	86.1%	45.9%	87.3%	88.4%	39.2%
[Berg et al. 2009]	88.3%	69.7%	42.2%	88.6%	85.6%	42.4%
Increase over state of the art (absolute)	**1.9%**	**3.2%**	**6.1%**	**1.3%**	**6.5%**	**9.8%**

Our version:

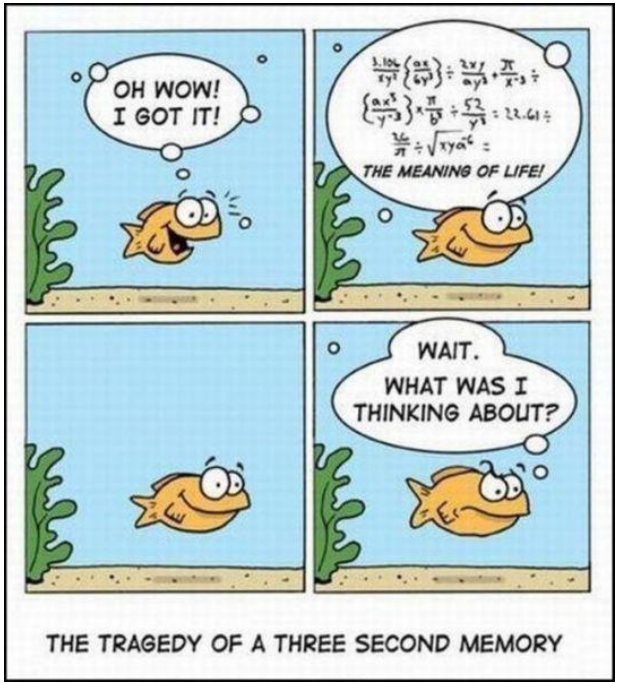
- “Drop your RNN and LSTM, they are no good!”

- Transformer algorithm = (Attention + RNN)-RNN

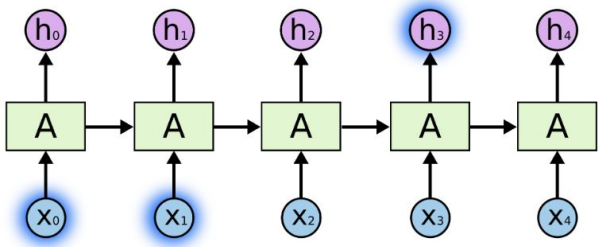
Recurrent Neural Networks (RNNs)

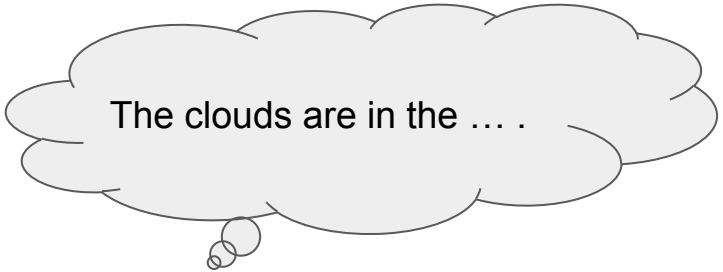


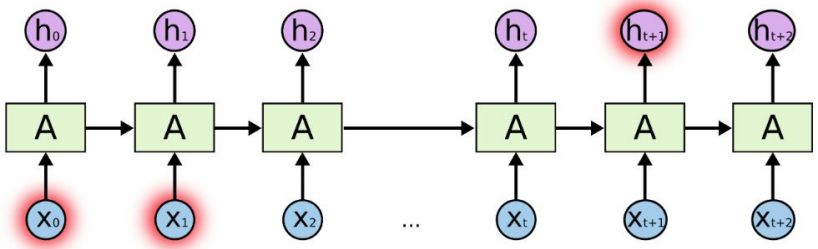
An unrolled recurrent neural network.



The Problem of Long-term Dependencies

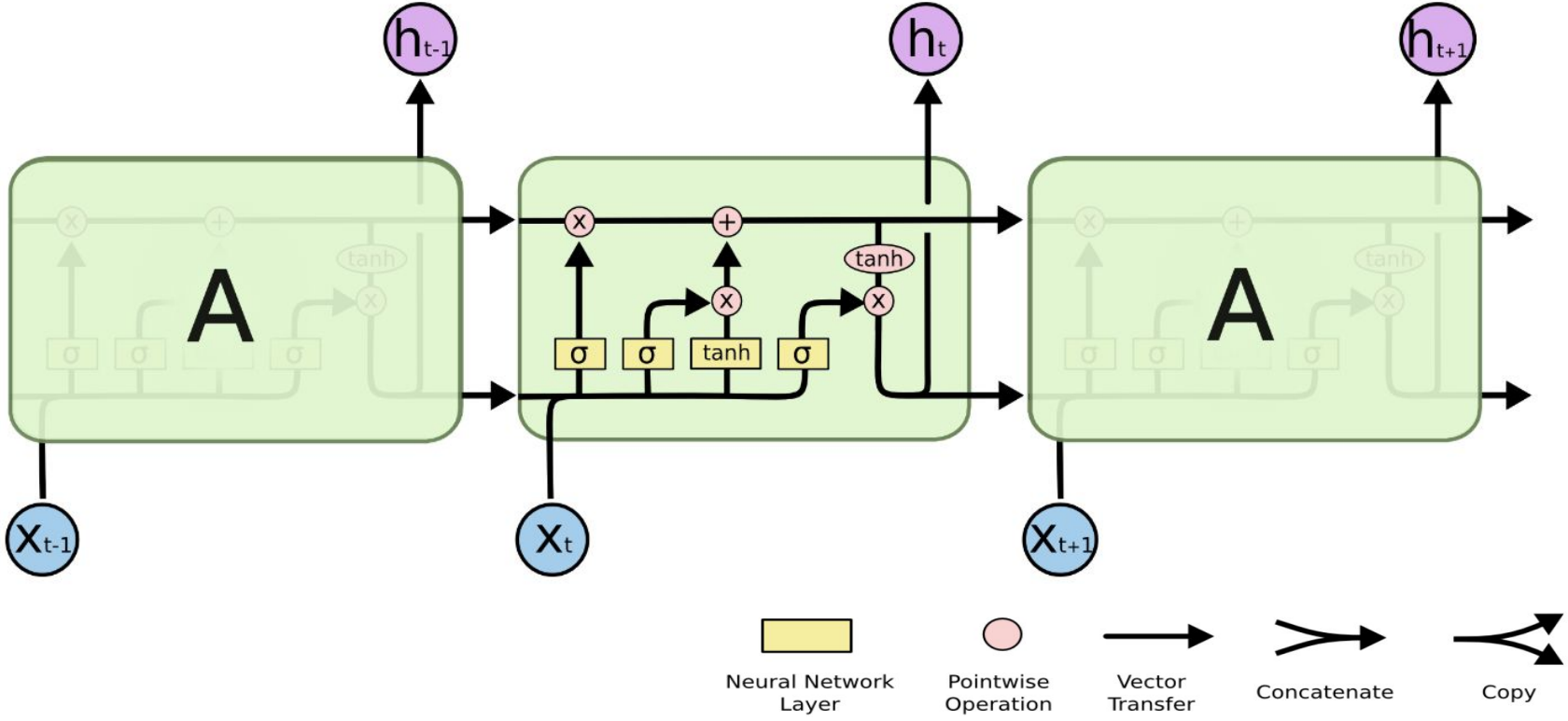






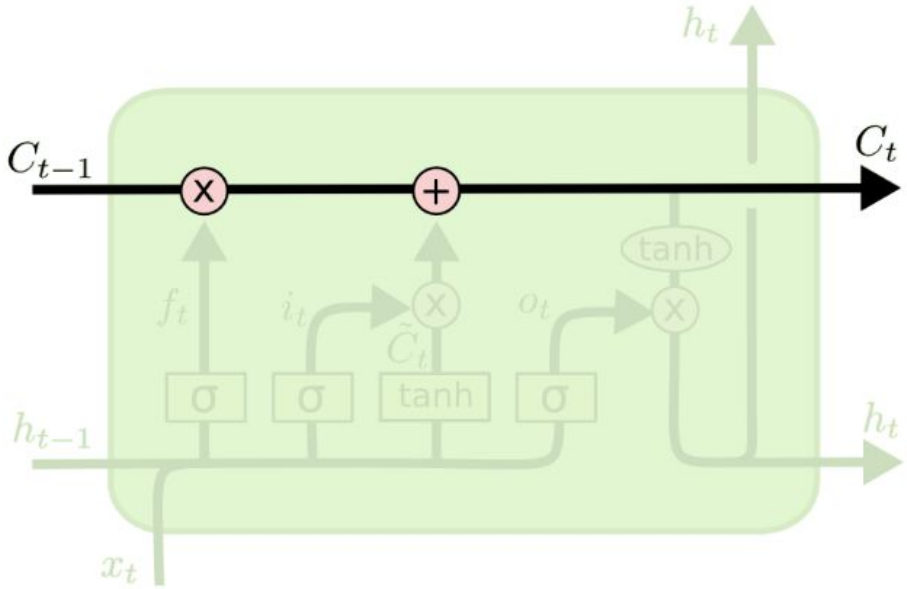
The reason is vanishing gradients!

LSTM (Long Short Term Memory) Networks



LSTM Structure

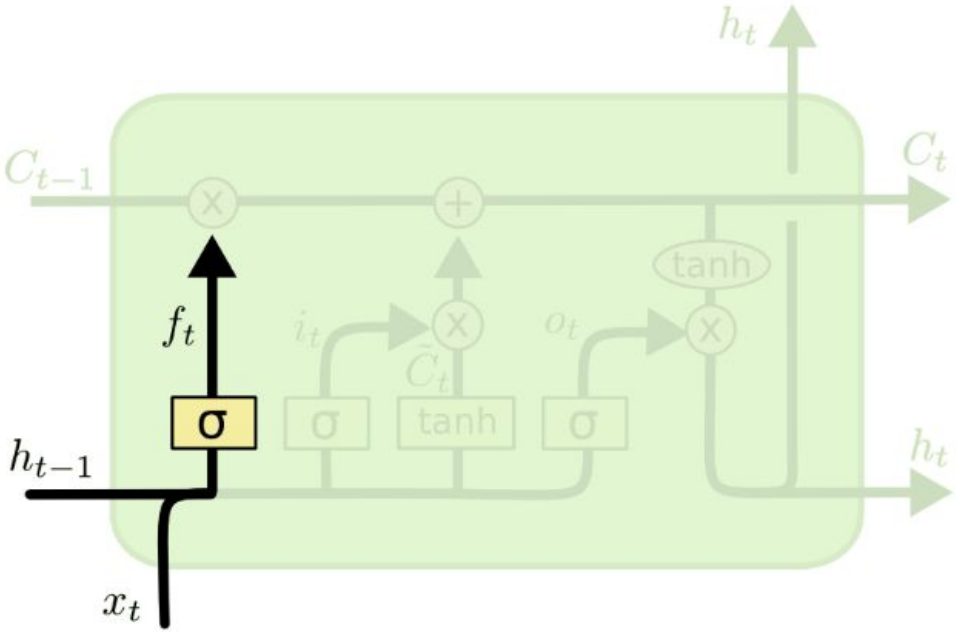
1



1. pass the cell state
2. forget gate layer
3. input gate layer+tanh layer
4. update the cell state
5. output filtered cell state

LSTM Structure

2

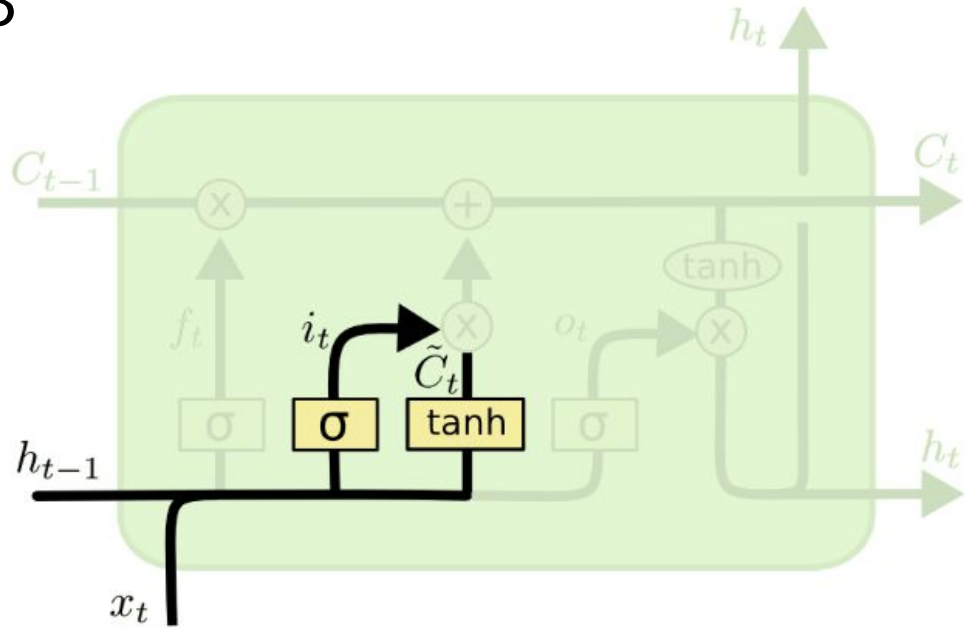


1. pass the cell state
2. forget gate layer
3. input gate layer+tanh layer
4. update the cell state
5. output filtered cell state

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTM Structure

3



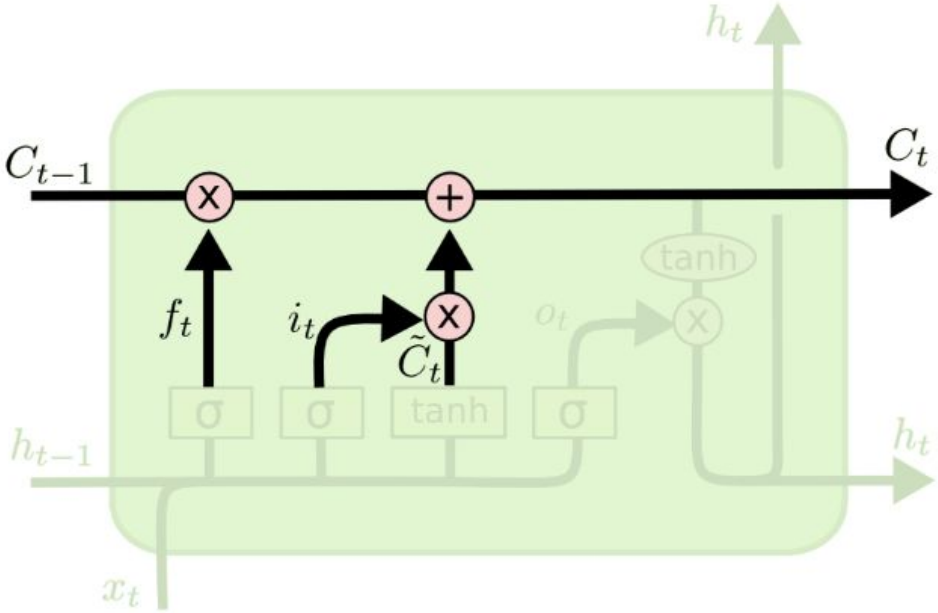
1. pass the cell state
2. forget gate layer
3. input gate layer+tanh layer
4. update the cell state
5. output filtered cell state

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

LSTM Structure

4

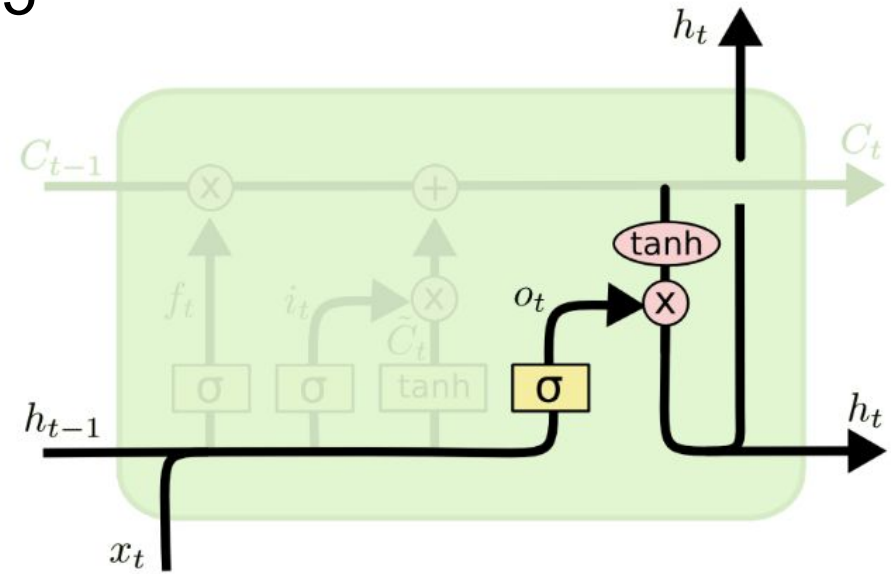


1. pass the cell state
2. forget gate layer
3. input gate layer+tanh layer
4. update the cell state
5. output filtered cell state

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM Structure

5



1. pass the cell state
2. forget gate layer
3. input gate layer+tanh layer
4. update the cell state
5. output filtered cell state

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

The main drawback of LSTM

sequential processing (from cell to cell)

requires HUGE memory

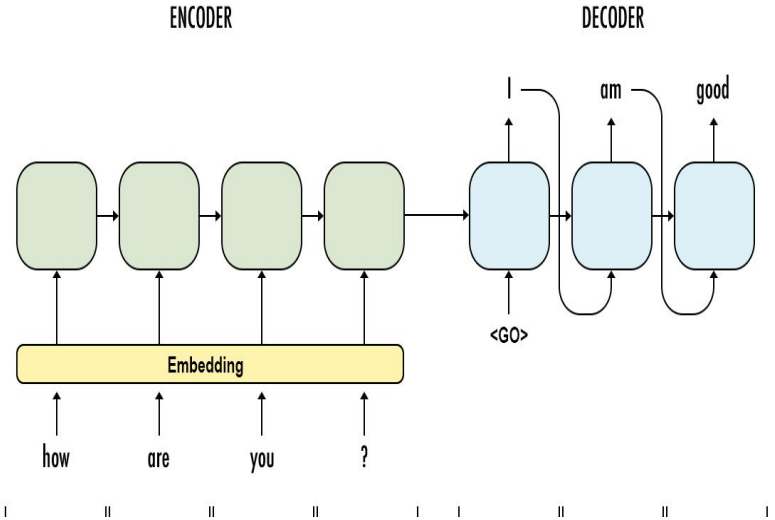
not hardware friendly

hard to train

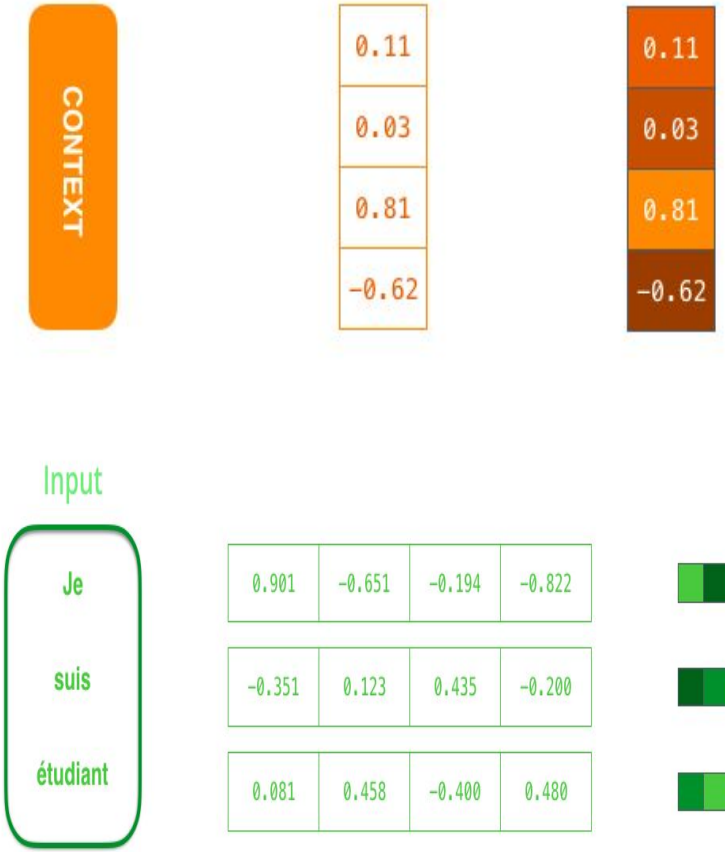
- Can remember sequences of 100s, **but not 10 000s**

SEQUENCE-TO-SEQUENCE MODELS WITH ATTENTION

- Deep learning models (machine translation, text summarization, image captioning);
- Takes a sequence of items (words, letters, features of images) and outputs another sequence of items;
- Composed of an **encoder** and a **decoder** (which both are the recurrent neural networks).



- **Encoder** compiles the information (each item in the input sequence) into a vector – **the context** (the number of hidden units in the encoder RNN) and sends the context over to the **decoder**, which begins producing the output sequence;
- To turn the input words into vector spaces that capture the meaning/semantic information of the words, the **"word embedding" algorithms** are used. Typical embedding vectors are of size 200 or 300.

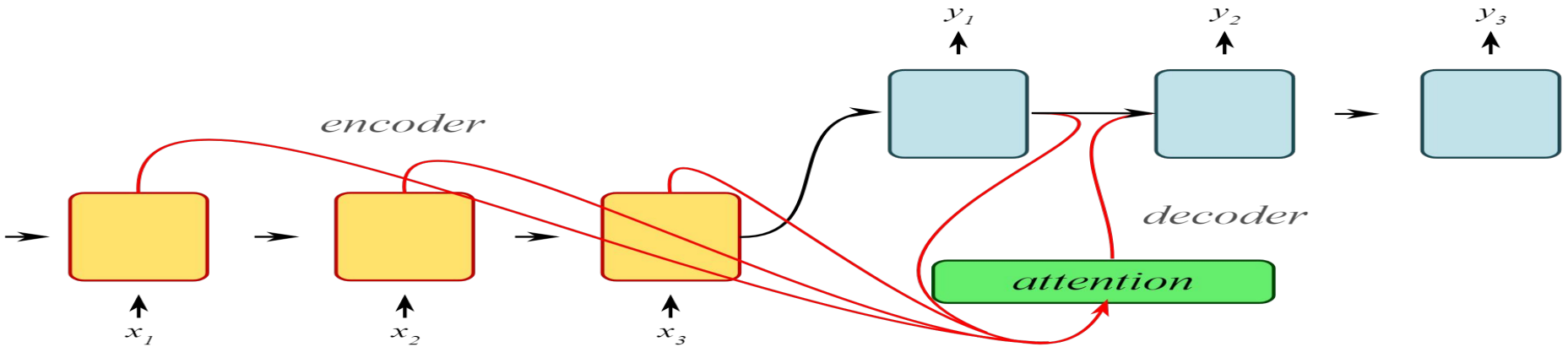


- RNN takes **two inputs** at each time step: **an input** and **a hidden state**;
- Each time step one of the RNNs does some processing, it updates its hidden state based on its inputs and previous inputs it has seen;
- The next RNN step takes **the second input vector** and **hidden state #1** to create the output of that time step.

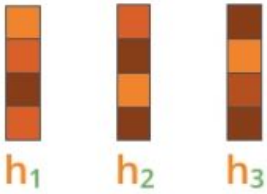
ATTENTION

Allows the model to focus on the relevant parts of the input sequence as needed by amplifying the signal from the relevant part of the input sequence.

- **Encoder** passes a lot more data to the **decoder**. Instead of passing the last hidden state of the encoding stage, the **encoder** passes all the hidden states to the **decoder**:
- Aan attention **decoder** does an extra step before producing its output. In order to focus on the parts of the input that are relevant to this decoding time step, the **decoder** does the following:
  1. Look at the set of **encoder** hidden states it received;
  2. Give each hidden states a score;
  3. Multiply each hidden state by its softmaxed score.



1. Prepare inputs



Encoder hidden states

Decoder hidden state at time step 4

2. Score each hidden state

13	9	9

scores
Attention weights for decoder time step #4

3. Softmax the scores

0.96	0.02	0.02

softmax scores

4. Multiply each vector by its softmaxed score

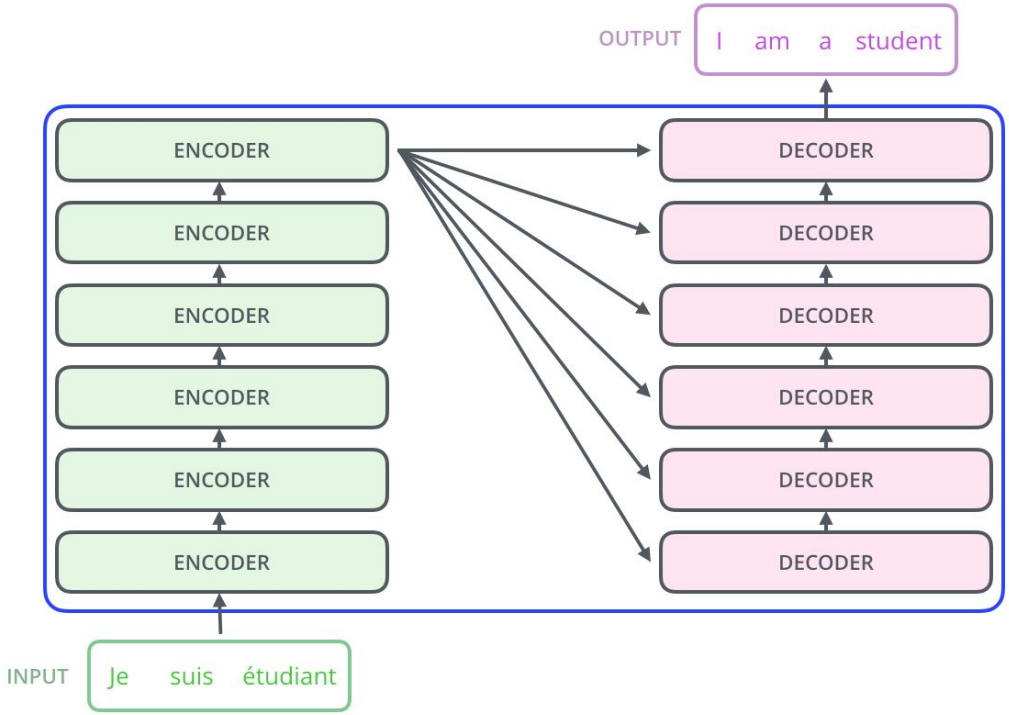
5. Sum up the weighted vectors

To summarize: How the attention process works

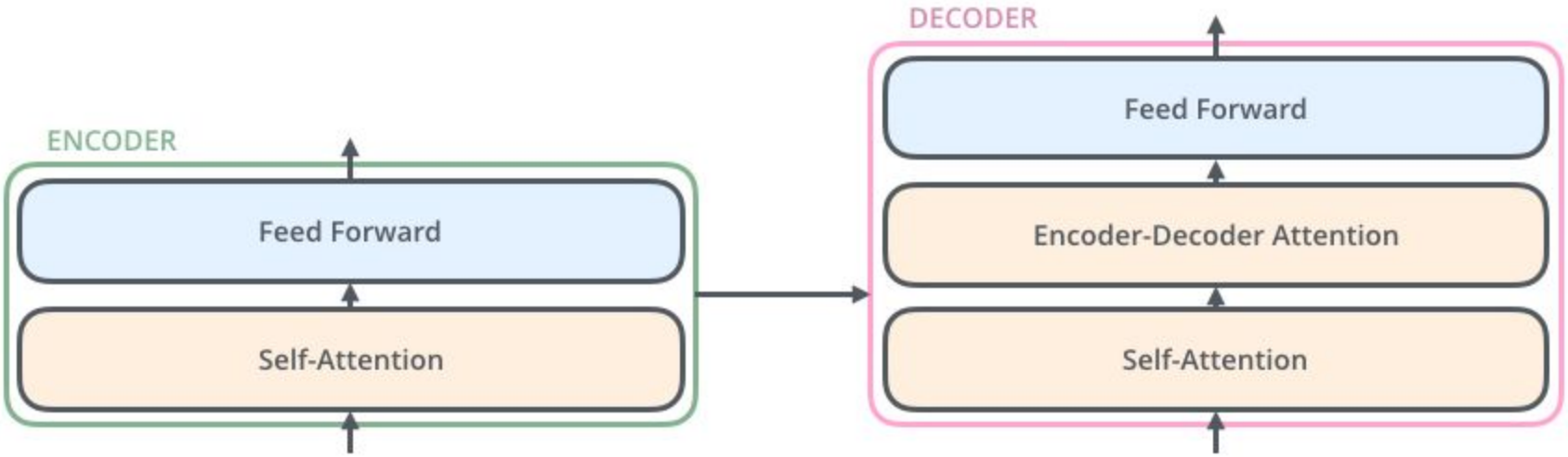
1. The attention decoder RNN takes in the embedding of the <END> token, and an initial decoder hidden state.
2. The RNN processes its inputs, producing an output and a new hidden state vector (h4). The output is discarded.
3. **Attention Step**: we use the encoder hidden states and the h4 vector to calculate a context vector (C4) for this time step.
4. We concatenate h4 and C4 into one vector.
5. We pass this vector through a feedforward neural network (one trained jointly with the model).
6. The output of the feedforward neural networks indicates the output word of this time step.
7. Repeat for the next time steps

Transformer

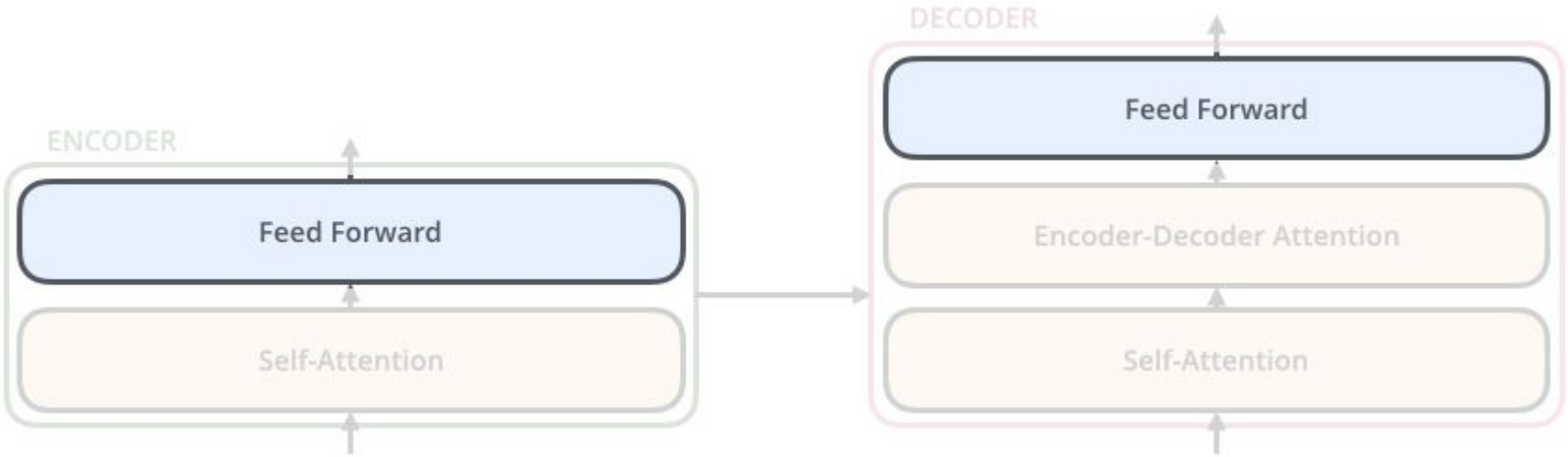
Transformer



Transformer

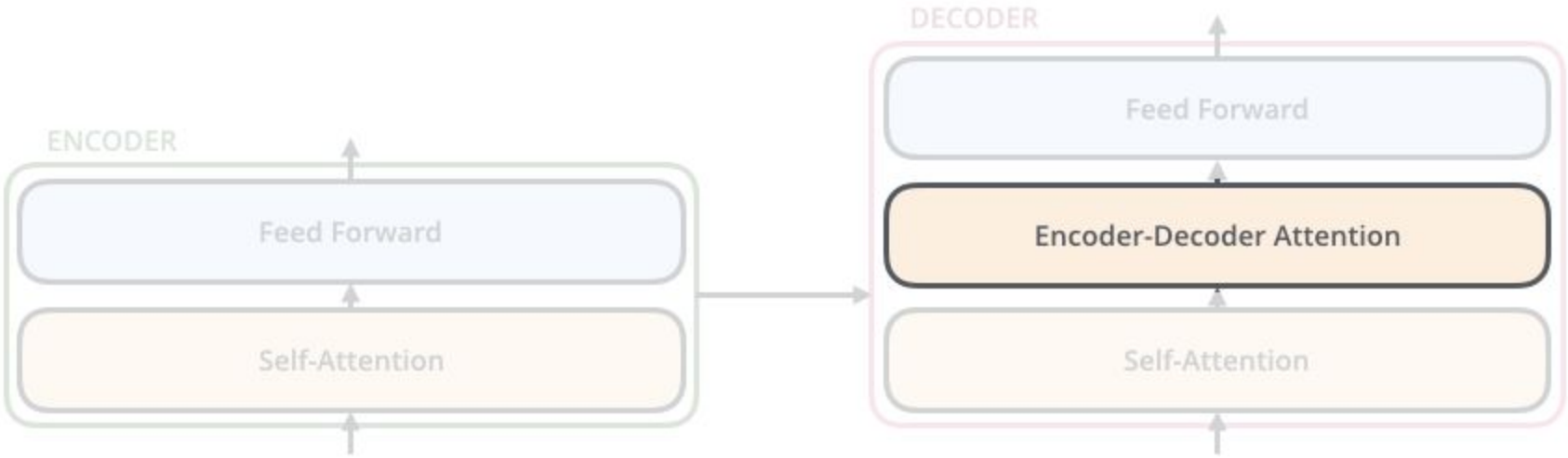


Transformer



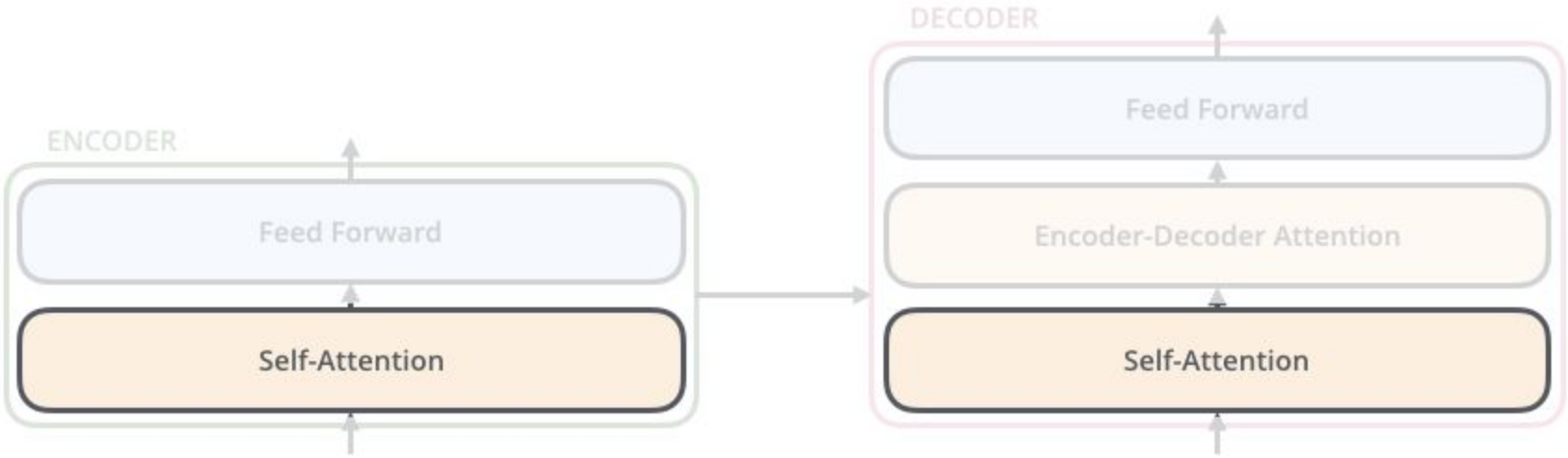
Fully connected layers, nothing unusual

Transformer



Similar to Seq2seq attention
“Which encoder values should the decoder look at”

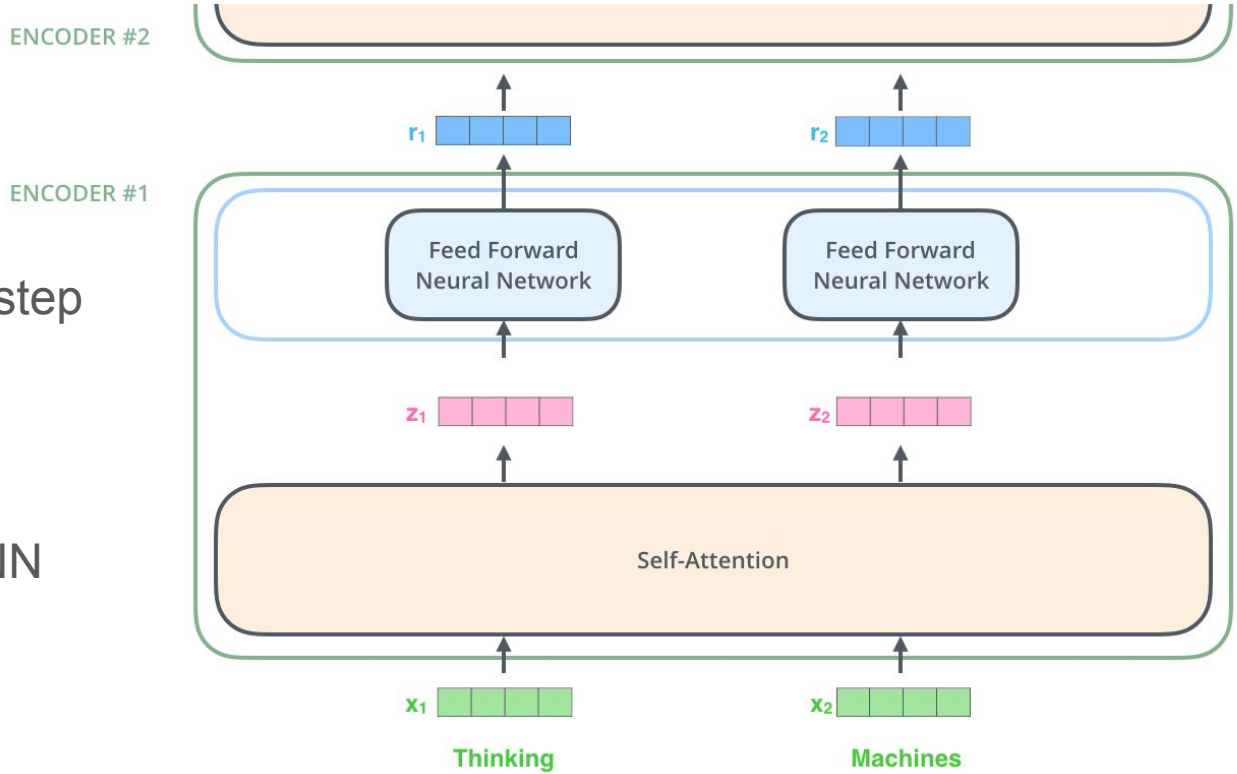
Transformer



Makes it possible to process sequences
Will be explained in depth shortly

Transformer (encoder)

- After the self-attention step the processing is done independently (and *in parallel*) for each word
- The weights of the FFNN are shared

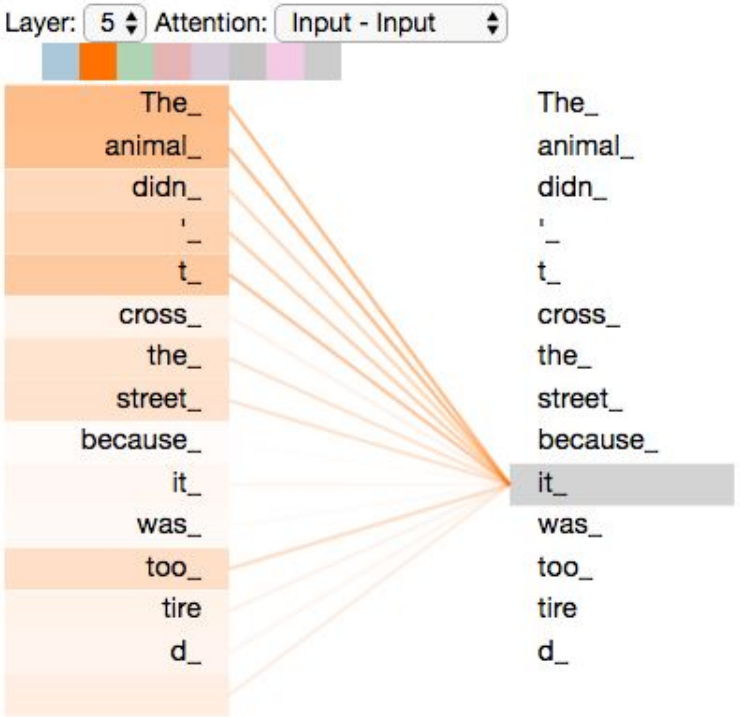


Transformer (self-attention)

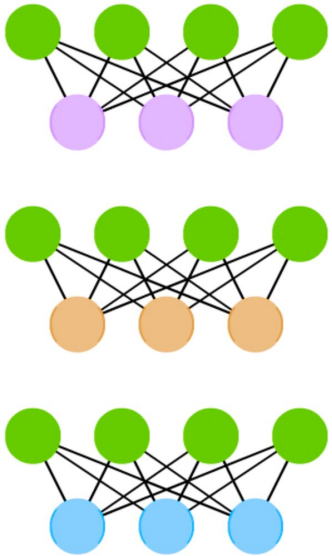
Attention links words in the input to words in the output

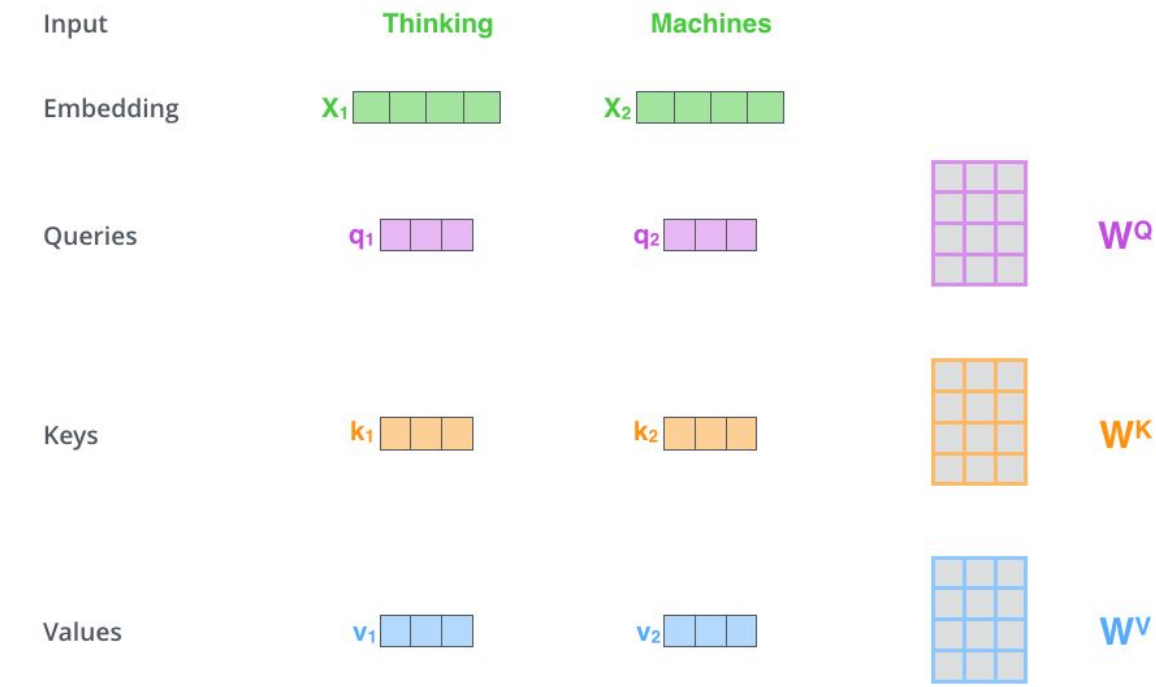
Self-attention shows links between words in a sequence

Here “it” means “the animal” and not “the street”



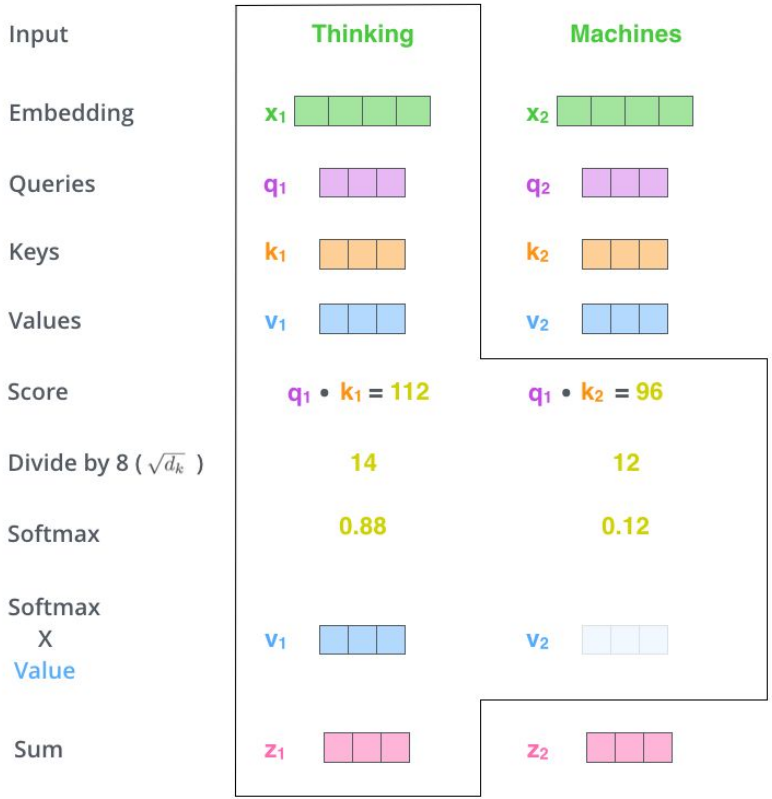
Transformer (self-attention)



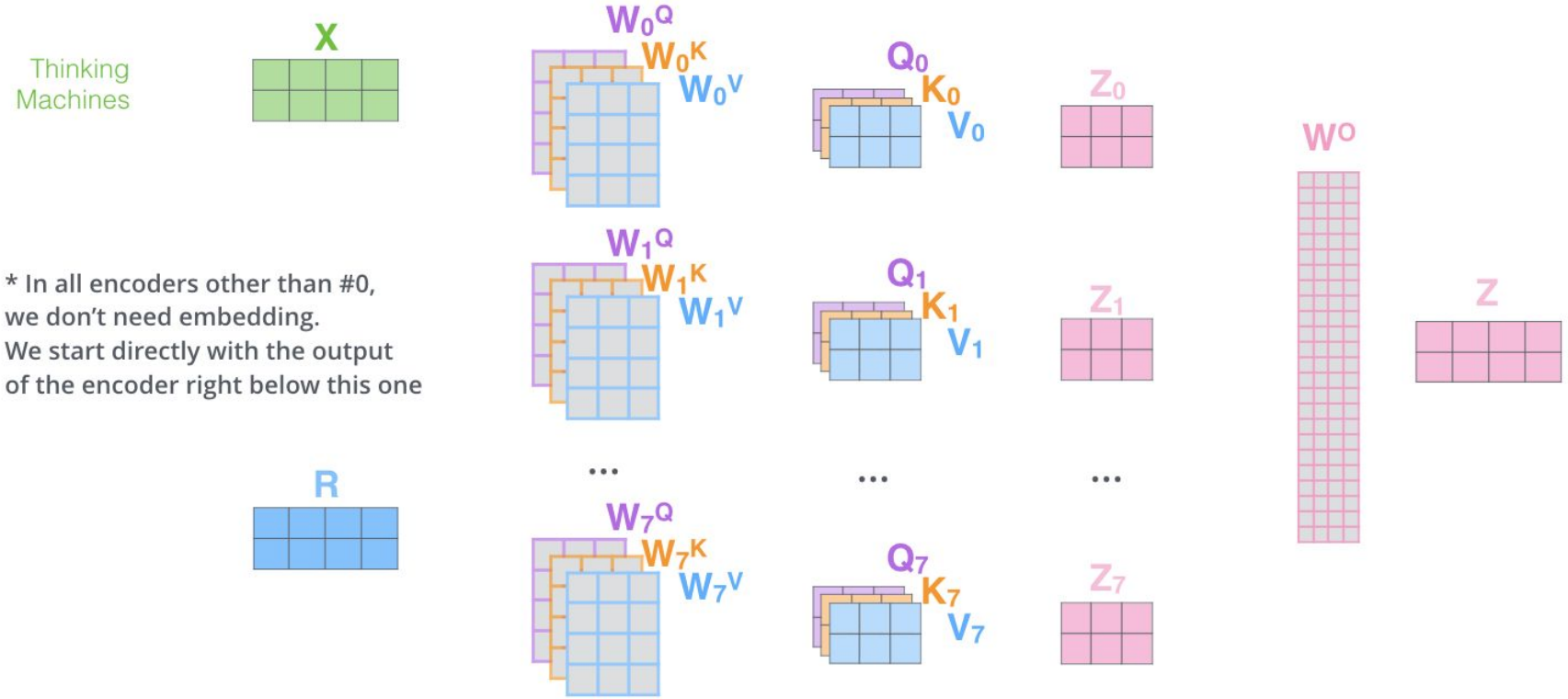


Transformer (self-attention)

- Query - current word's ID vector
- Key - target word's ID vector
- If Query and Key are aligned, their dot product will be high (like for vectors in the usual Euclidean 2D/3D space)
- The current word takes its Query and searches for a word with a close enough Key
- Value - the actual information the word holds



Transformer (self-attention)



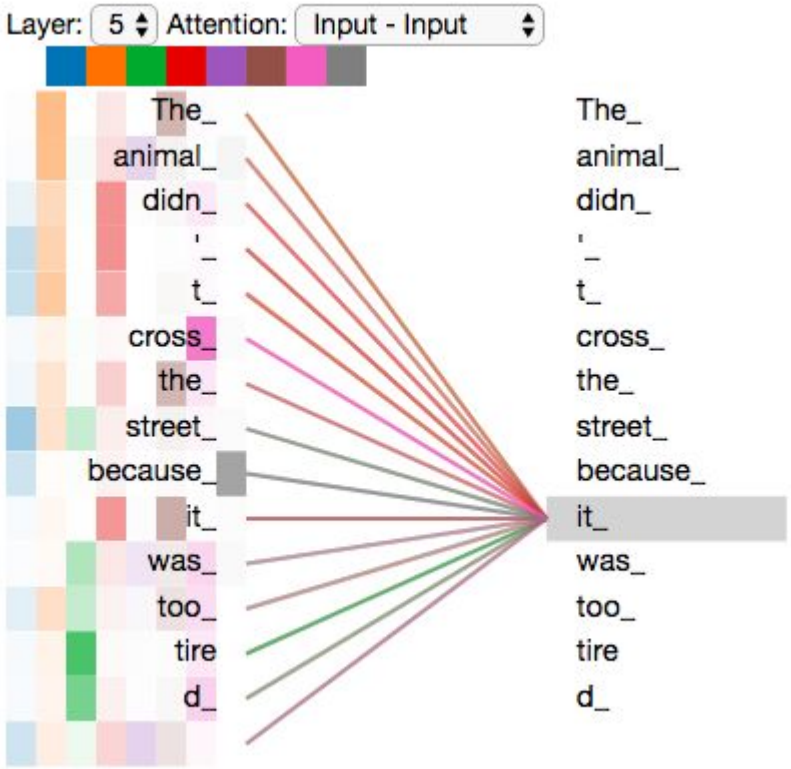
Transformer (self-attention)

Is “it_” the animal or the street?

What if the word “tired” was replaced with “crowded” or “wide”?

What other words are important to translate “it_” or to understand it?

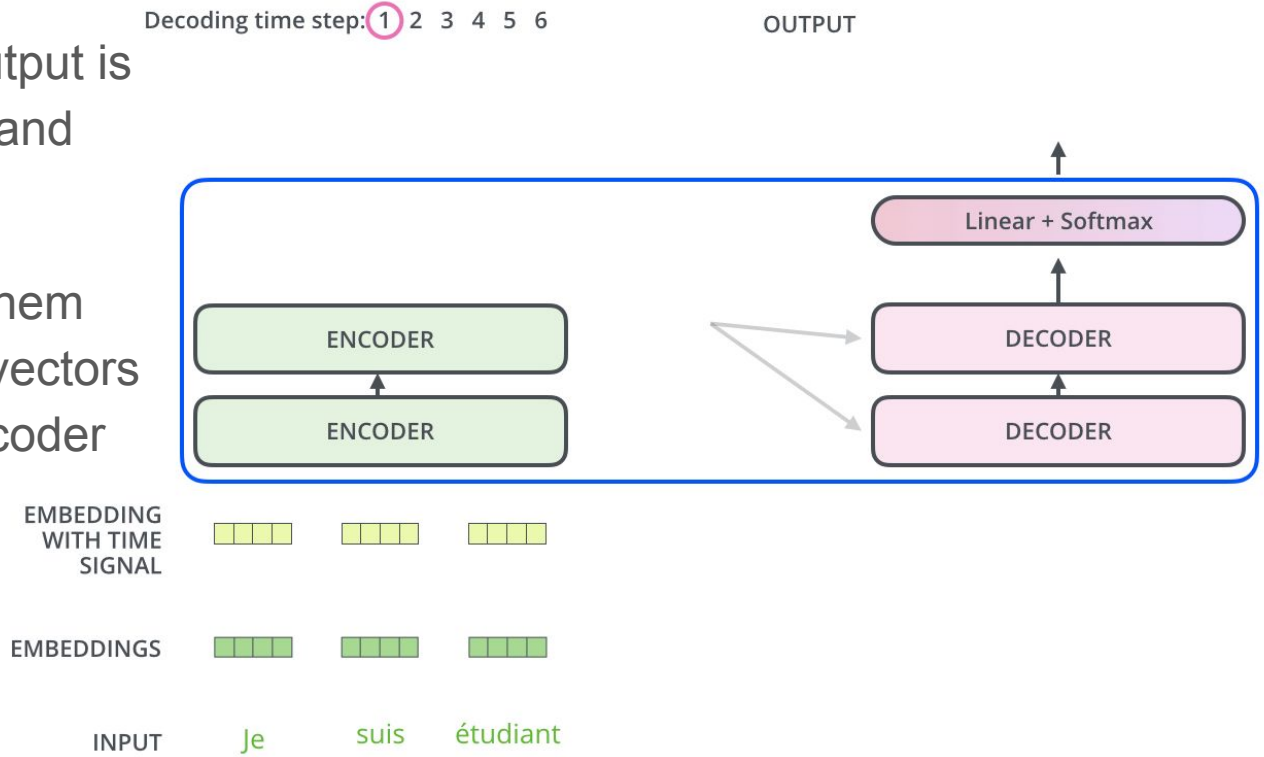
Here we can try to interpret some of the information



Transformer (encoder-decoder interaction)

The last encoder output is used to create Key and Value vectors

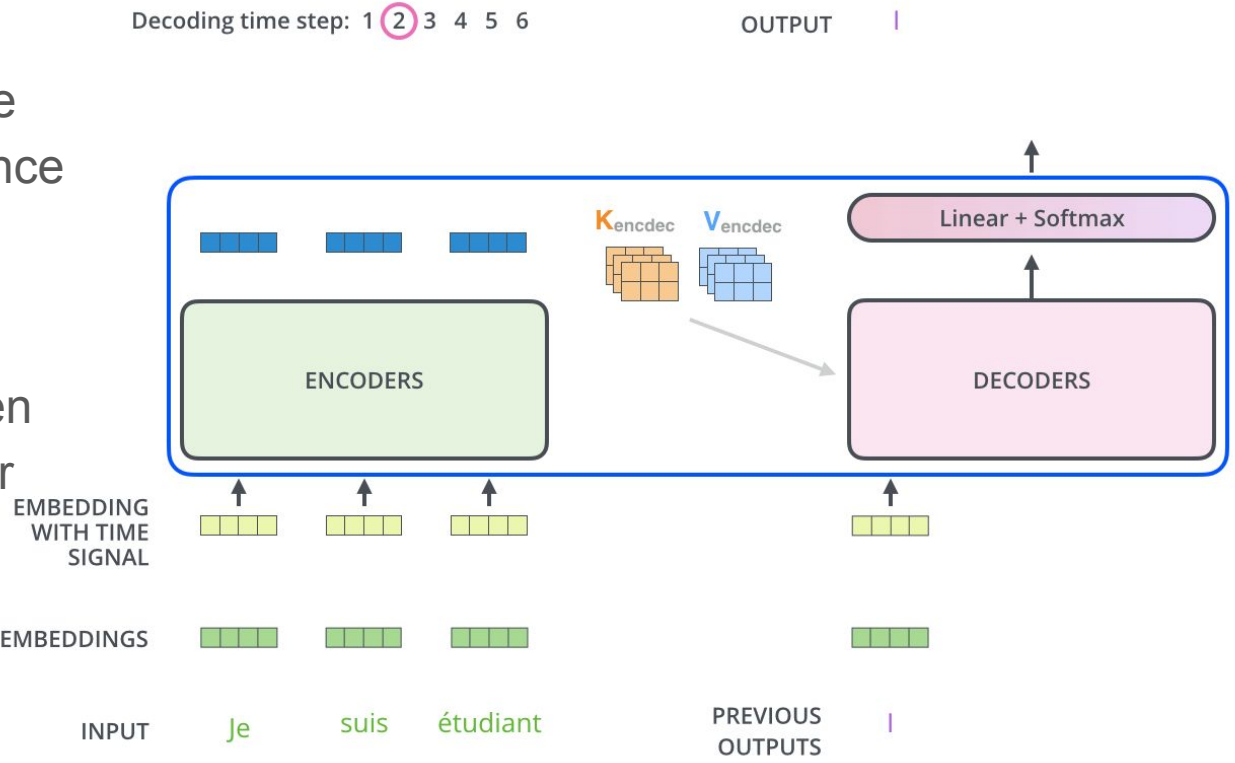
The decoder uses them with its own Query vectors to form encoder-decoder attention



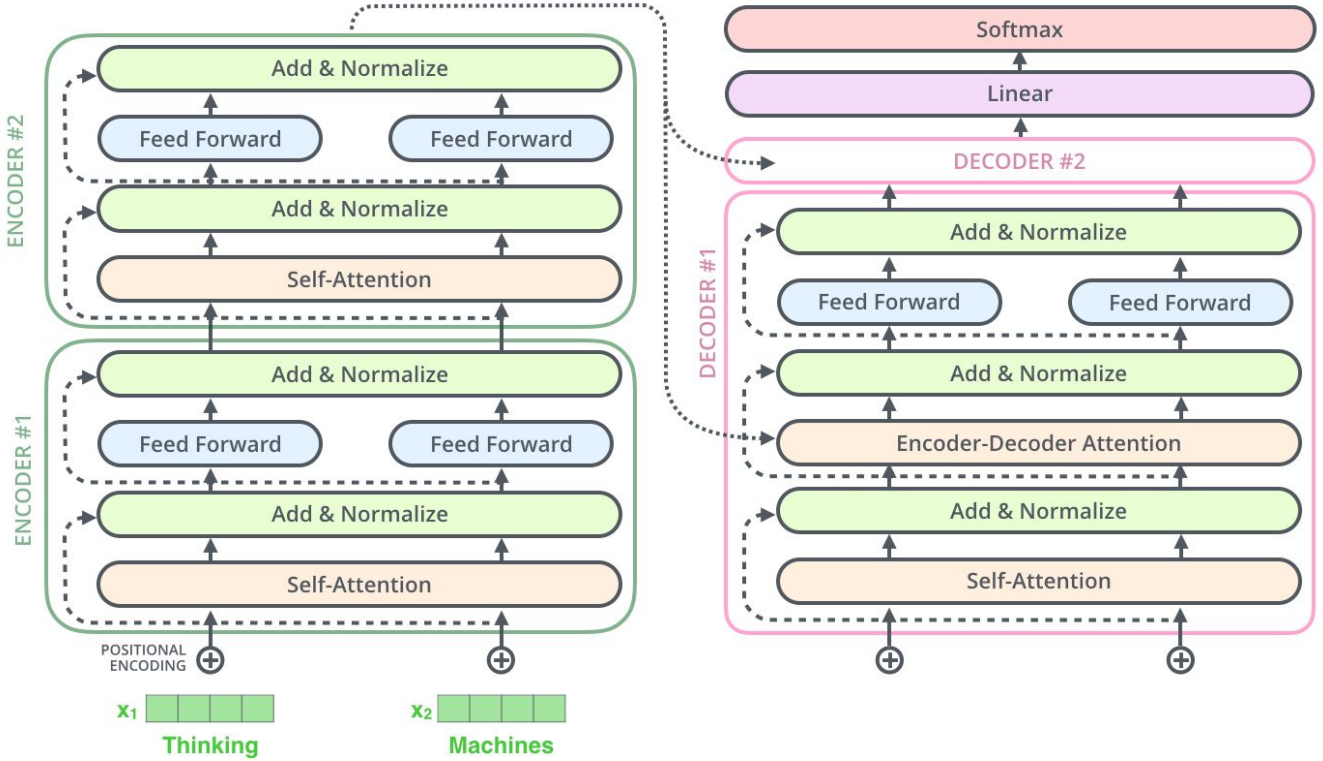
Transformer (decoding process)

The decoder's self-attention uses the current output sequence

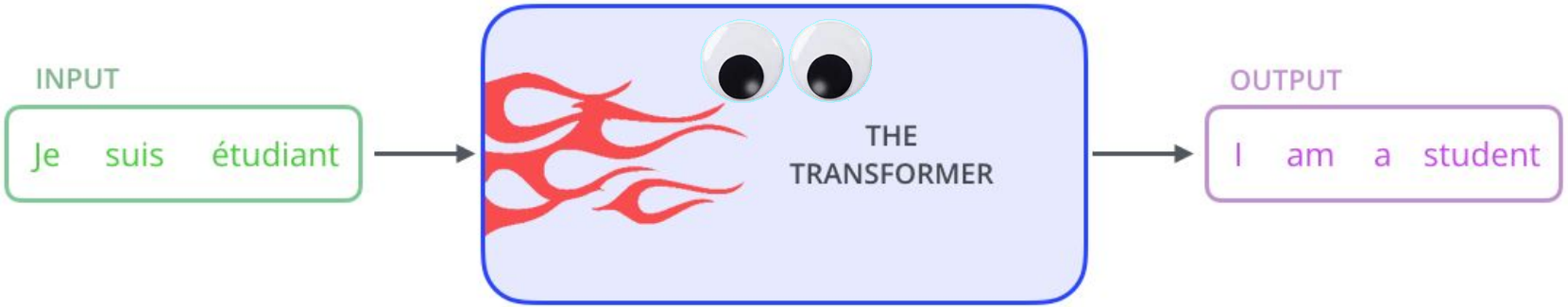
Then it applies encoder-decoder attention and only then the feed-forward layer



Transformer (overview)



Transformer



Thank you