

Neural Methods for NLP

Master LiTL --- 2023-2024

chloe.braud@irit.fr

https://gitlab.irit.fr/melodi/andiamo/teaching_cbraud/master_litl

Course 6: Encoder-Decoder, Transformer

Schedule 2023-2024

1	28.11	13h30-15h30	2	(C1) ML Reminder + (TP ML) + TP POO
2	05.12	13h30-15h30	2	(C2) Intro DL + TP2 (part1)
3	12.12	13h30-16h	2.5	(C3) Embeddings + TP2 (part2) + TP3
4	19.12	13h30-16h	2.5	TP4 + Start projects
(holidays)				
5	09.01	13h-16h	3	(C4) Training a NN + TP5 + TP6
6	16.01	13h-16h	3	(C5) CNN, RNN + TP7 + TP8
→(15/01) Assignments Part 1 due				
7	23.01	13h-16h	3	Projects
8	01.02	13h-16h	3	(C6) Encoder-decoder, transformer + TP9
9	13.02	13h-16h	3	(C7) Current challenges + project defenses
→ (12/02) Assignments Part 2 due				

Content

RNNs:

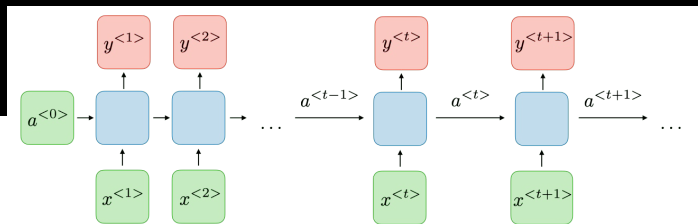
- allow to condition on the entire history
- can act as language models → learning the likelihood of occurrence of a word based on the previous sequence of words (or based on characters, sentences, paragraphs)

→ make them suitable for use as **generators**: generating natural language sequences

Combining encoding + generation = **encoder-decoder / sequence to sequence**

- = conditioned generators: the generated output is conditioned on a complex input
- based on RNNs and/or **Attention** mechanisms

TP: library Transformers (HuggingFace)



RNN generators

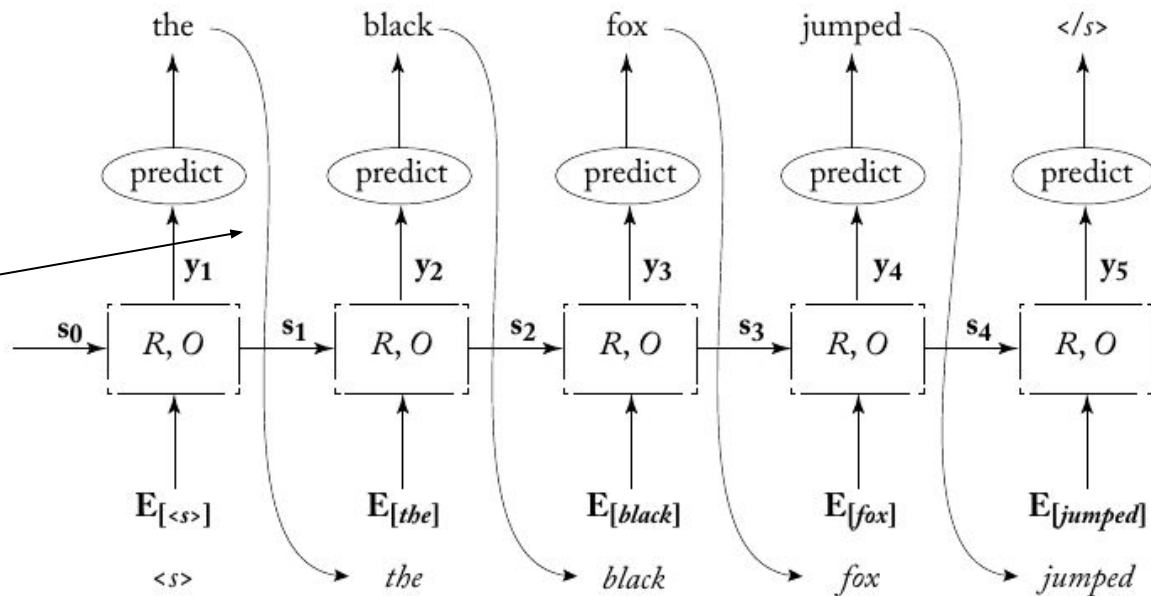
RNN transducer: Producing an output y_i for each input

→ use this architecture to do sequence generation

- Idea:
 - tying the output at time i with its input at time $i+1$, i.e. using the predicted token as next input
 - at each step, select the output with the highest probability (or use beam-search for finding a global high-probability output)

RNN Generator

- predict a distribution over the next output
- choose a token t_i
- its embedding vector is fed as input of the next step
- stop when generating a 'end-of-sequence' symbol $\langle /s \rangle$



RNN Generator

[Sutskever et al. 2011]: generation of sentences using a character based RNN

- ability to condition on long histories
- the produced text resembles fluent English
- and show sensitivity to properties such as nested parenthesis

For more analysis on RNN-based character-level language models [Karpathy et al. 2015]

Encoder-Decoder

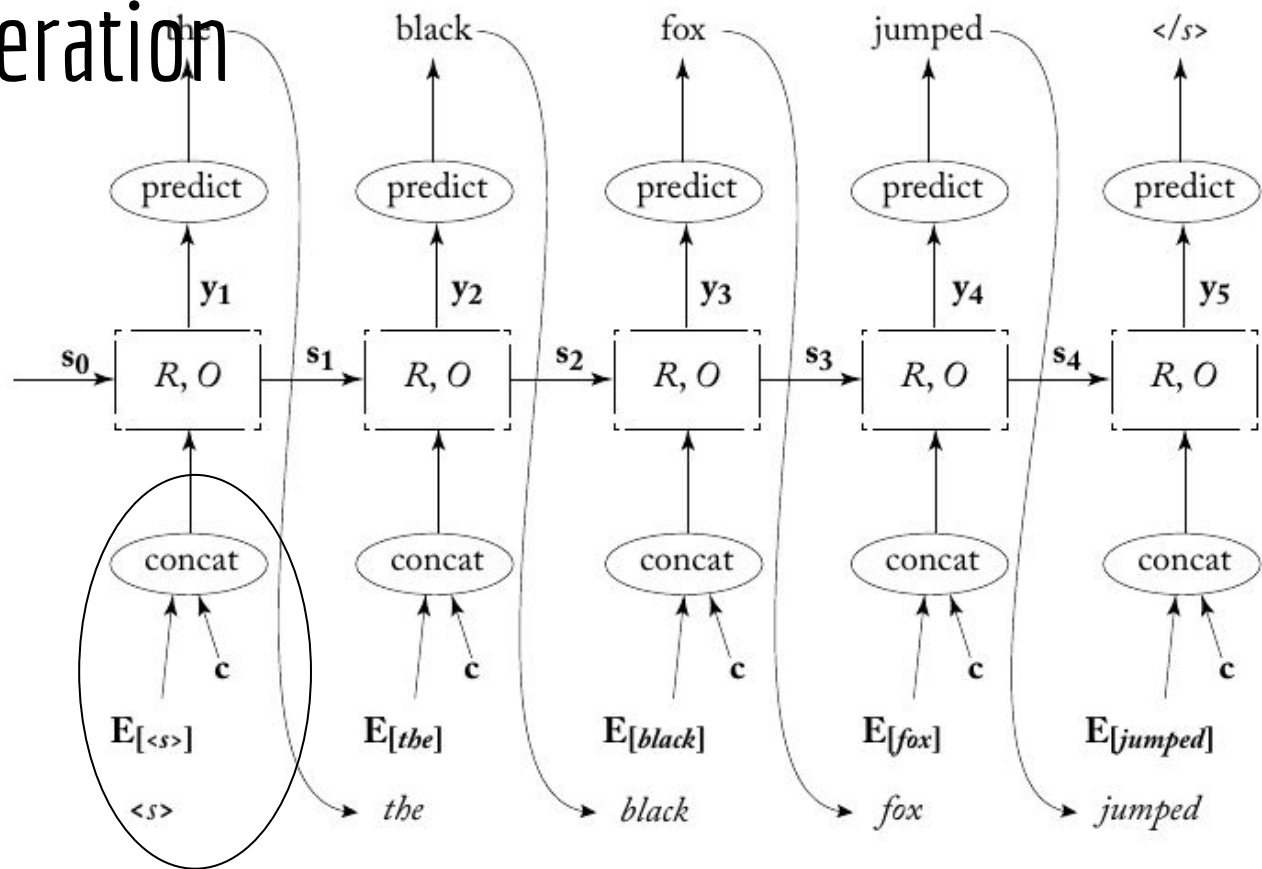
Real power of RNN transducer: *Conditioned generation framework*

- Until now: = generating the next token \mathbf{t}_{j+1}
 - based on the previously generated tokens $\mathbf{t}_{1:j}$
- **Conditioned generation** = generating the next token \mathbf{t}_{j+1}
 - based on the previously generated tokens $\mathbf{t}_{1:j}$
 - + **an additional conditioning context \mathbf{c}** (represented as a vector)

Conditioned generation

at each stage:

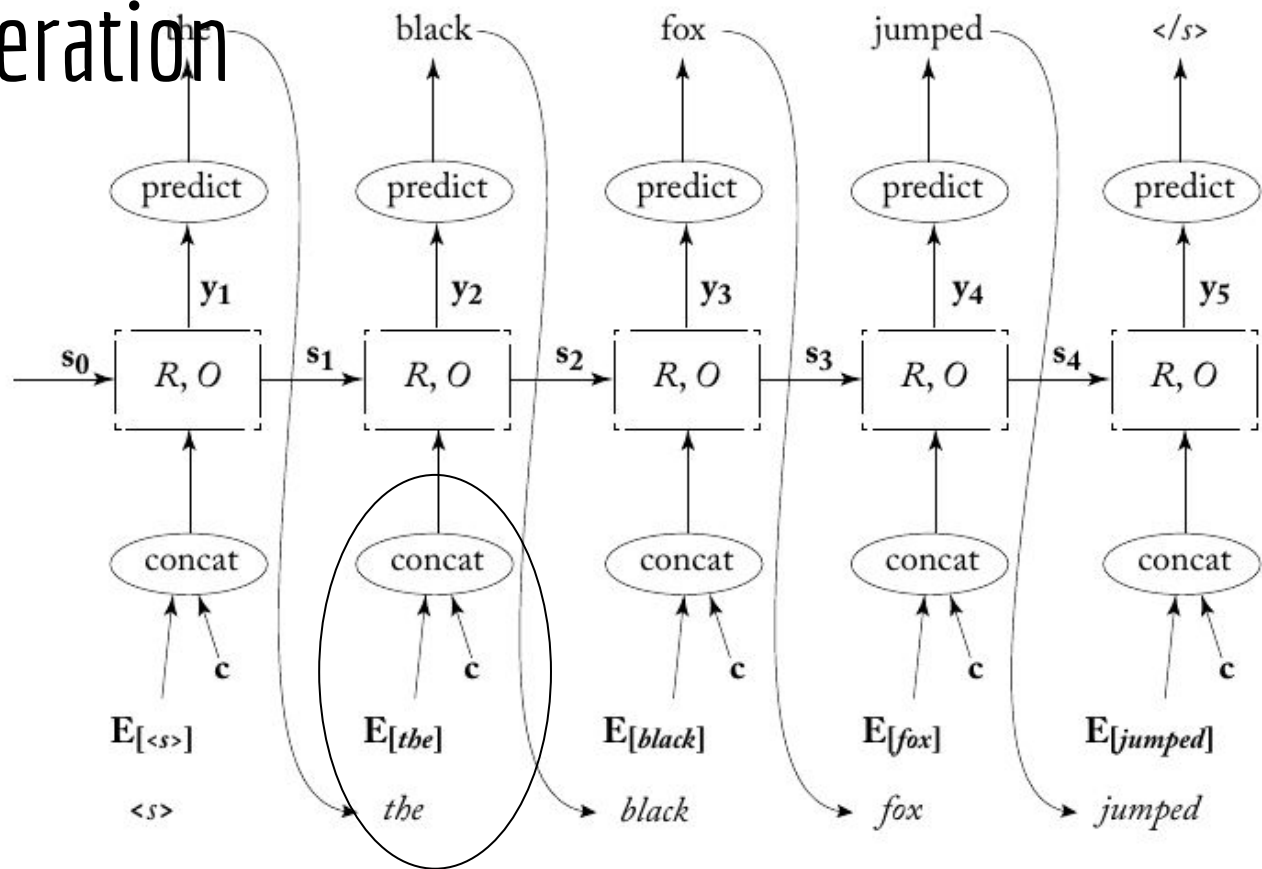
- the context vector \mathbf{c} is concatenated to the input (predicted) t_j
- and the concatenation is fed into the RNN to produce the next prediction



Conditioned generation

at each stage:

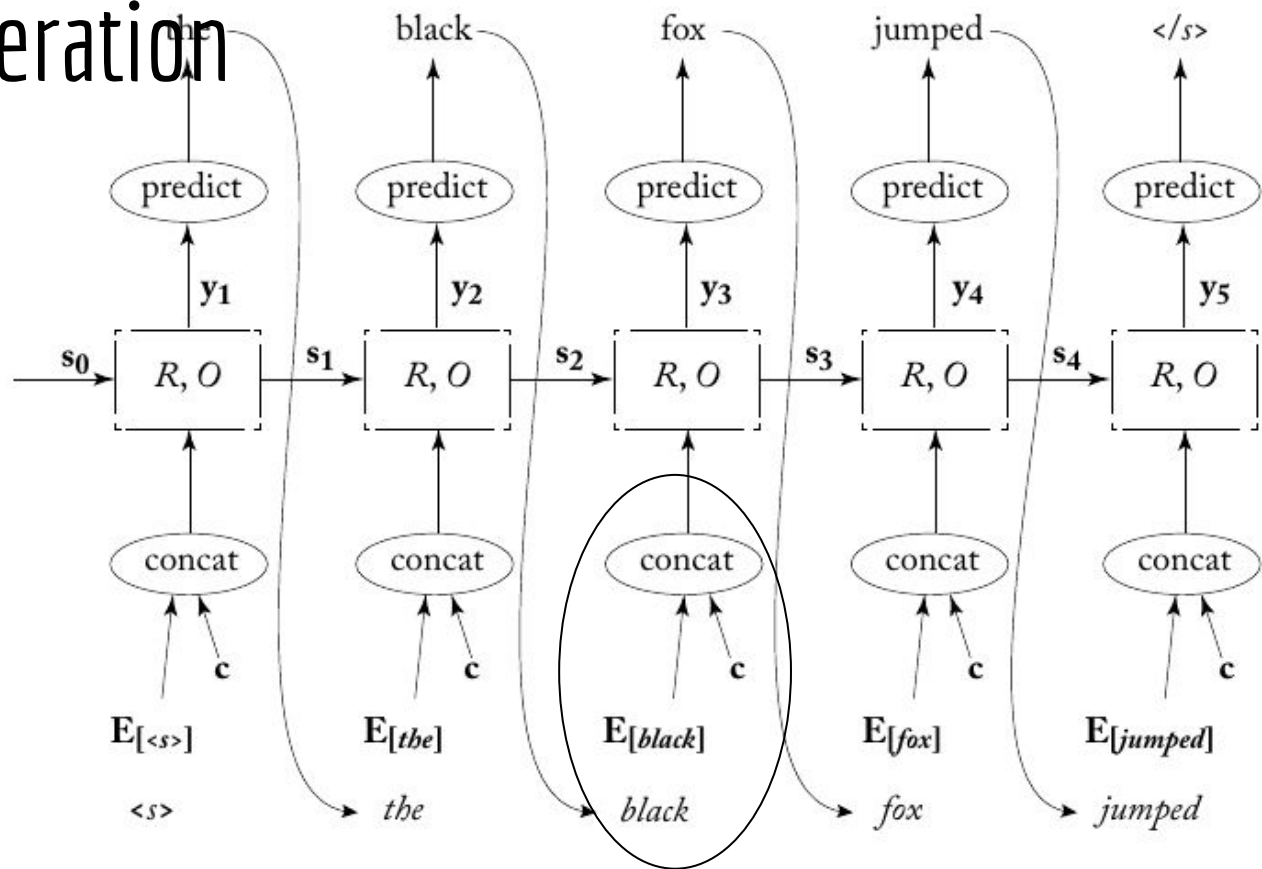
- the context vector \mathbf{c} is concatenated to the input (predicted) t_j
- and the concatenation is fed into the RNN to produce the next prediction



Conditioned generation

at each stage:

- the context vector \mathbf{c} is concatenated to the input (predicted) t_j
- and the concatenation is fed into the RNN to produce the next prediction



Conditioned generation

What can be encoded in the context vector \mathbf{c} ? anything that we find useful!

- use the **topic** associated with documents to generate texts conditioned on the topic
- rating / **sentiment** associated to a review: generate reviews with a specific polarity
- inferred properties, automatically derived from texts: if a sentence is written in first person, the level of vocabulary ...

= some fixed-length vectors

→ another popular approach: \mathbf{c} is itself a sequence of words

Conditioned generation

What can be encoded in the context vector c ? anything that we find useful!

- use the **topic** associated with documents to generate texts conditioned on the topic
- rating / **sentiment** associated to a review: generate reviews with a specific polarity
- inferred properties, automatically derived from texts: if a sentence is written in first person, the level of vocabulary ...

= some fixed-length vectors

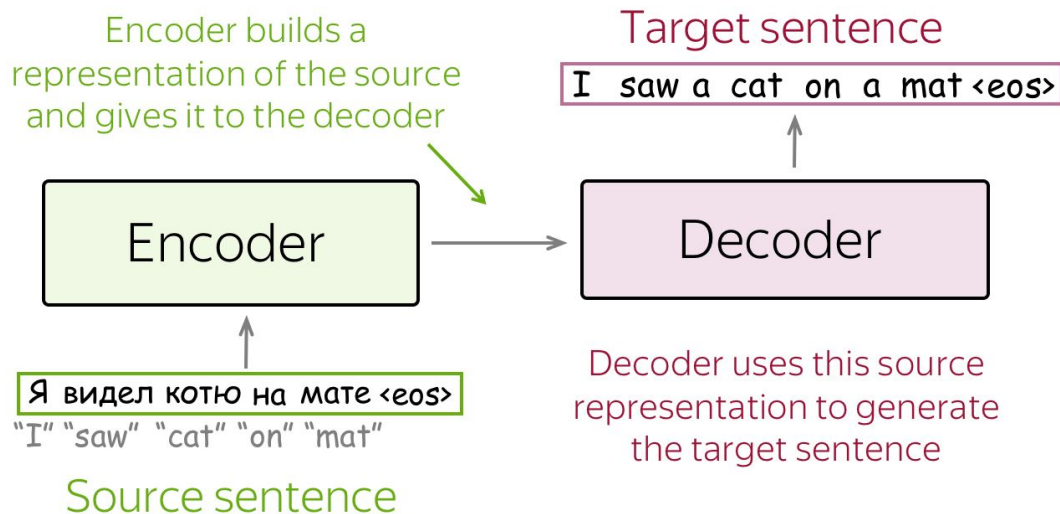
→ **another popular approach: c is itself a sequence of words**

Typical example: Machine translation

e.g. : Machine translation

- encoding the input in source language = produce a representation
- decoder: use the representation to condition the output in target language

decoder = generator of target language



Basic architecture of all the models presented in this course

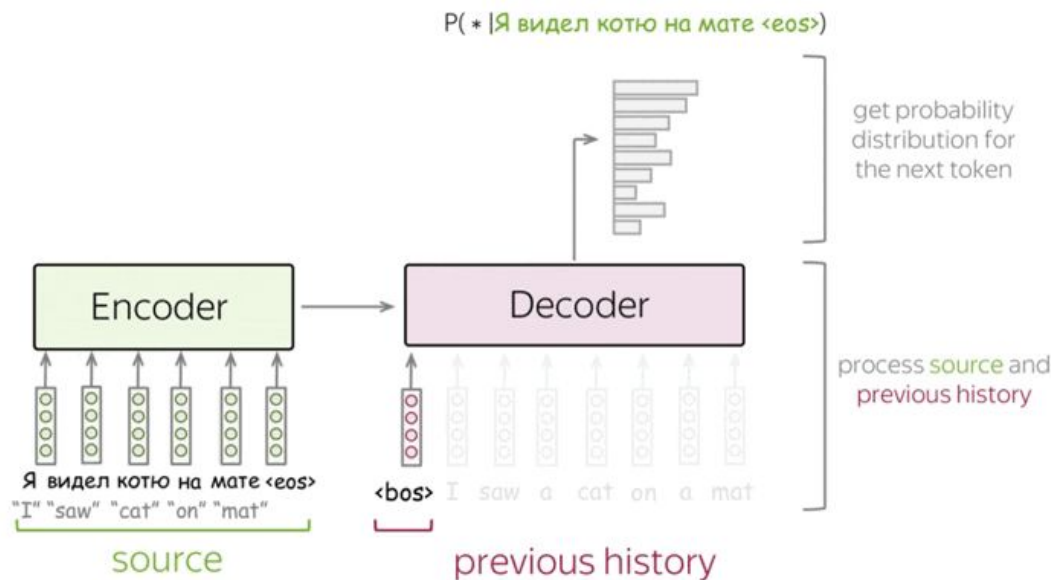
Seq2seq

Sequence to sequence (seq2seq) [Cho et al, 2014; Sutskever et al 2014] \rightarrow **c** is itself a sequence of words

- source sequence $\mathbf{x}_{1:n}$ (e.g. a sentence in French)
- target output sequence $\mathbf{t}_{1:m}$ (e.g. its translation in English)

Note: The length of the input can be different of the length of the output

General idea



Pipeline:

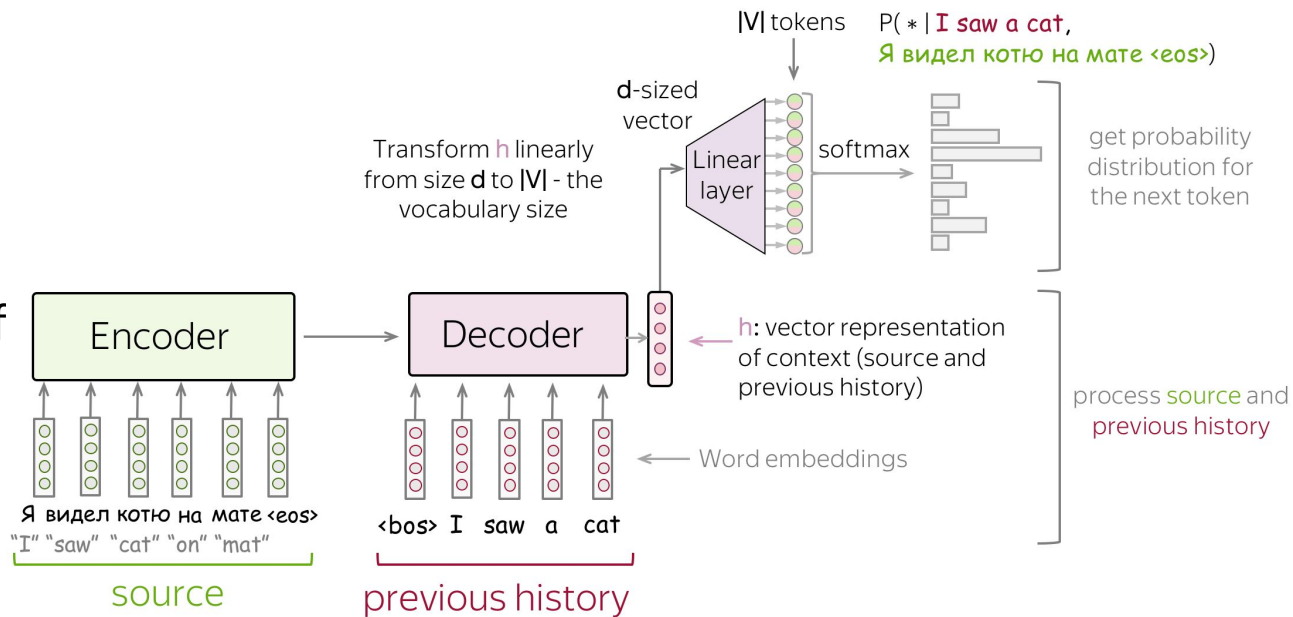
- feed source and previously generated target words into a network;
- get vector representation of context (both source and previous target);
- from this vector representation, predict a probability distribution for the next token.

Output layer

classification part:

- vector representation of dimension \mathbf{d}
- we need a vector of size $|V|$

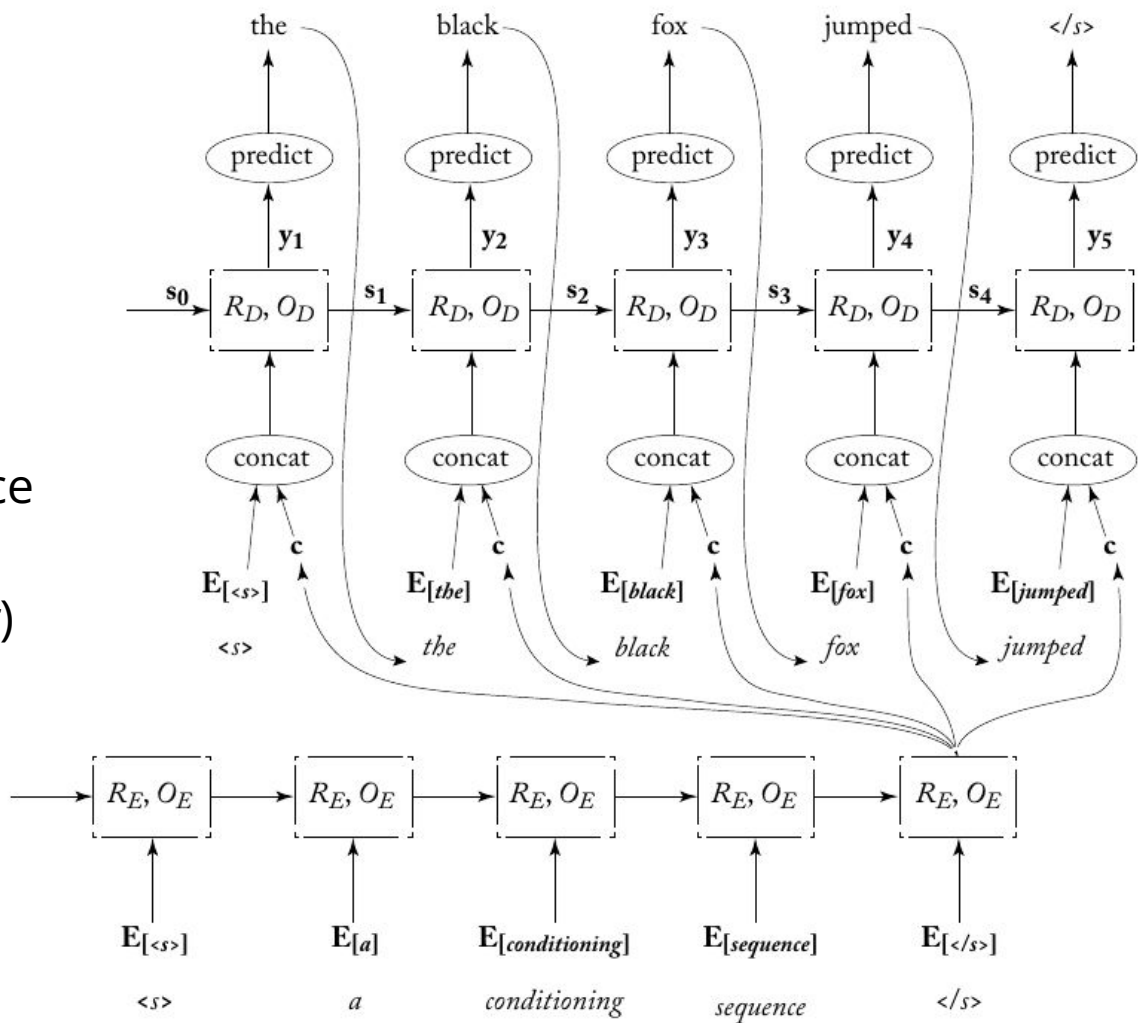
→ linear layer to perform the transformation (then softmax)



Encoder-decoder

Simplest architecture: 2 RNNs

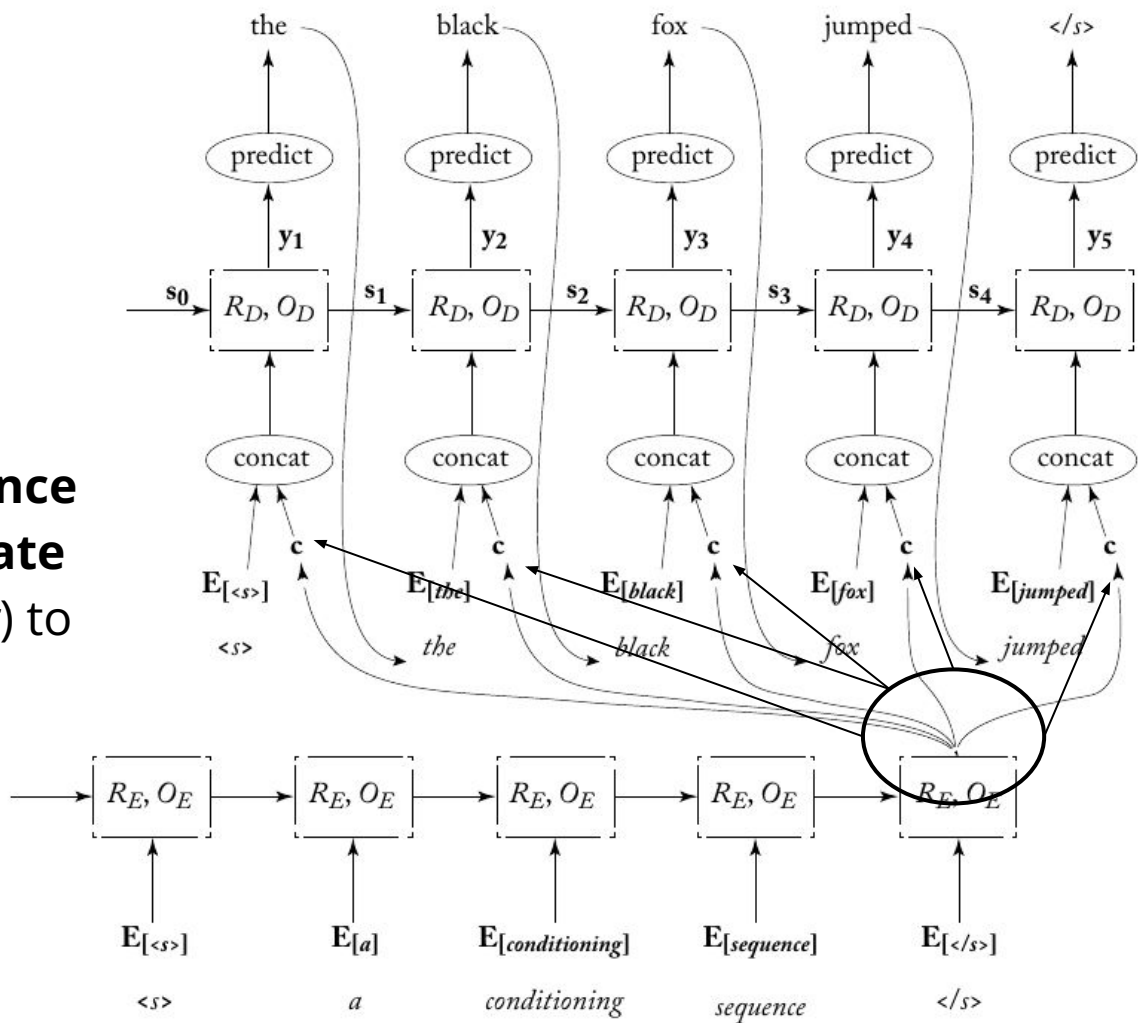
- *encoding* the source sentence $\mathbf{x}_{1:n}$ using an RNN
- using another RNN (*decoder*) to generate the output $\mathbf{t}_{1:m}$



Encoder-decoder

Simplest architecture: 2 RNNs

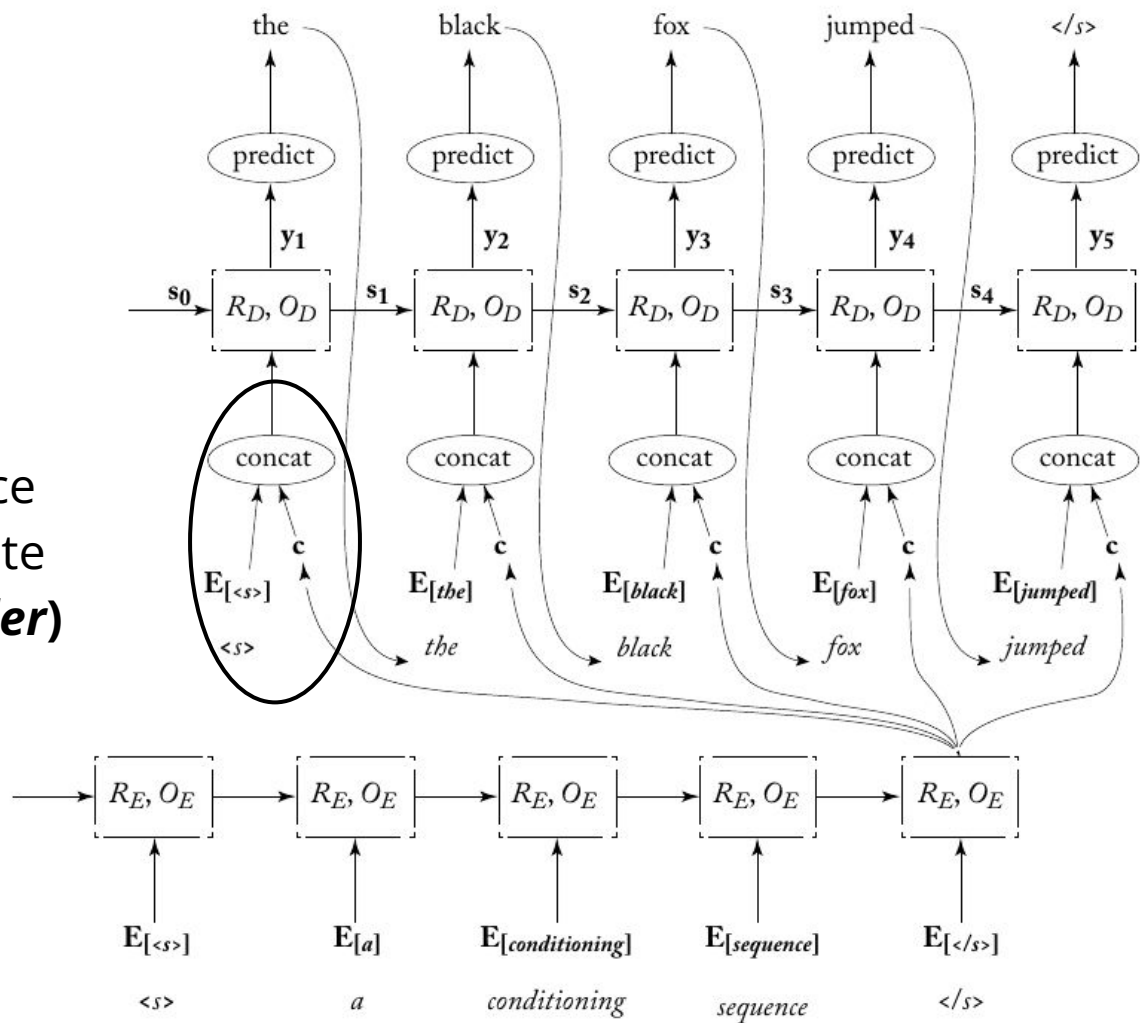
- **encoding** the source sentence $\mathbf{x}_{1:n}$ using an RNN \rightarrow **last state**
- using another RNN (*decoder*) to generate the output $\mathbf{t}_{1:m}$



Encoder-decoder

Simplest architecture: 2 RNNs

- *encoding* the source sentence $\mathbf{x}_{1:n}$ using an RNN \rightarrow last state
- **using another RNN (*decoder*) to generate the output $\mathbf{t}_{1:m}$**
 \rightarrow **predicted output + encoding of the input**



Encoder-Decoder

- originally built to solve Seq2Seq problems
- useful to map sequences of size n to sequences of length m
- encoder = summarizing the source sentence as a vector \mathbf{c}
- encoder and decoder are trained jointly:
 - supervision only for decoder, but propagation all the way back to the encoder
 - use of cross-entropy loss, as usual

Some modifications:

- e.g. encoder and decoder can have several layers
- decoding: greedy (most probable token) or beam-search (keep several hypothesis)

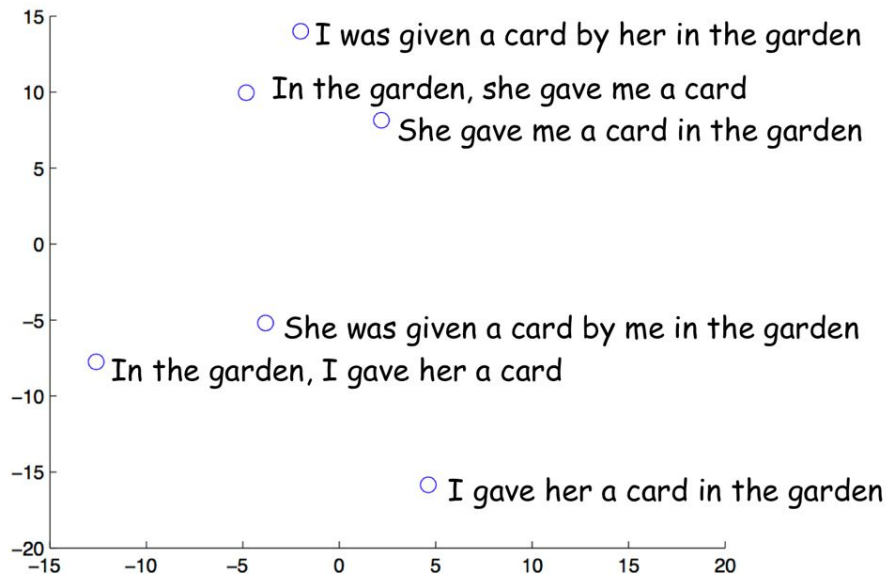
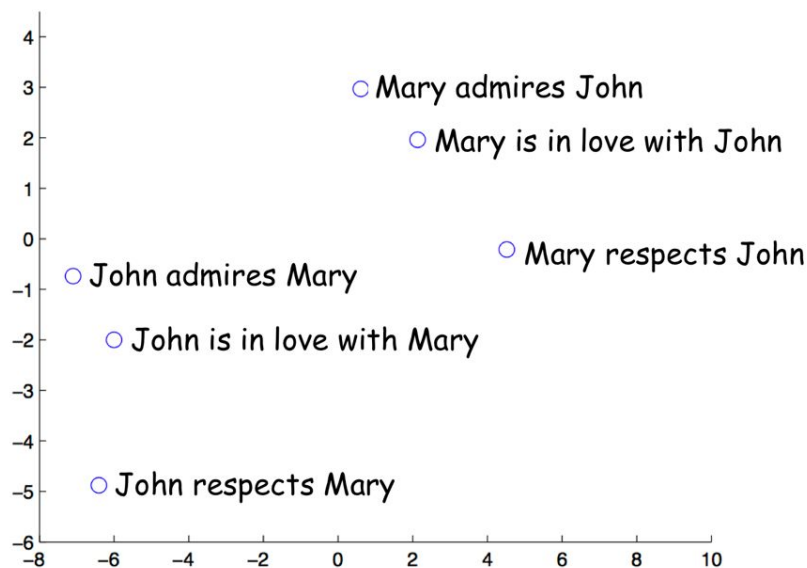
Encoder-Decoder

Applications examples:

- **Machine translation:** in [\[Sutskever et al. 2014\]](#), they feed the source sentence in reverse (then x_n is the first word) + approach with 8 layers of high-dimensional LSTMs → computationally expensive
- **Email auto-response:** map an email to a short answer [Kannan et al 2016] with LSTMs as encoder and decoder
- **Morphological inflection:** input is a base word + inflection request, the output is an inflected form. [Faruqui et al 2016]: character level seq2seq.
- Other uses: almost **any task** can be formulated this way (but there could be better, easier to learn architectures). It has also been used for e.g. **sentence compression** by deletion [Filippova and Altun, 2013], **POS tagging** and **NER** [Gillick et al 2016], **syntactic parsing** using constituency bracketing decisions [Vinyals et al 2014]

Learned representation

In [\[Sutskever et al. 2014\]](#) (MT) they looked at the last encoder state and visualize several examples



Encoder-Decoder: Other conditioning contexts

- The encoder can be also a single word, a CBOW encoding, or generated by another network
- The context can encode extra-linguistic information: user information (age, gender ...) e.g. dialogue generation [Li et al 2016]
- Image captioning: encoding input image (using a CNN) and the vector is used as conditioning context for an RNN generator trained to predict image description

Unsupervised sentence similarity

Use encoder-decoder framework to produce **vector representations of sentences**

→ we want similar sentences to have similar vectors (rather ill-defined...)

Unsupervised approaches (trained using un-annotated data) using encoder-decoder:

- an encoder RNN is used to produce context vectors \mathbf{c}
- then used by an RNN decoder to perform a task: the information important from the sentence for the task are captured in \mathbf{c}
- finally: the encoder is used to generate sentence representations \mathbf{c}

→ the similarity relies on the task

Unsupervised sentence similarity

Auto-encoding:

- the decoder attempts to reconstruct the input sentence
- may not be ideal, not considering similar sentences with similar meaning but different words

Machine translation:

- trained to translate sentences from English to another language
- encode what is needed to translate properly: sentences translated similarly will have similar vectors; requires a large parallel corpus

Skip-thoughts [Kiros et al 2015]:

- one decoder is trained to reconstruct the previous sentence, and a second decoder the following sentence
- extend the distributional hypothesis from words to sentences; impressive results

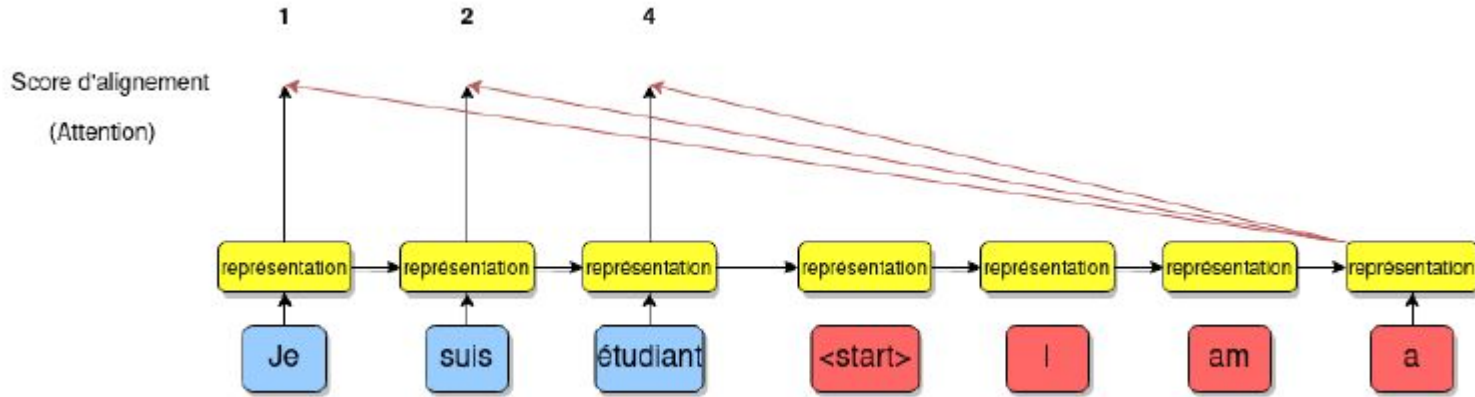
Conditioned generation with attention

Encoder-decoder = the input sentence is encoded into a single vector

- the **encoder vector c must contain all the information required**
 - but it is hard for the encoder to compress the sentence
- the generator must be able to extract the information from this fixed-length vector
 - but for the decoder, different information may be relevant at different steps

This compression in one representation is suboptimal → ***attention mechanism***
[[Bahdanau et al 2014](#); Luong et al 2015]

Attention



Idea:

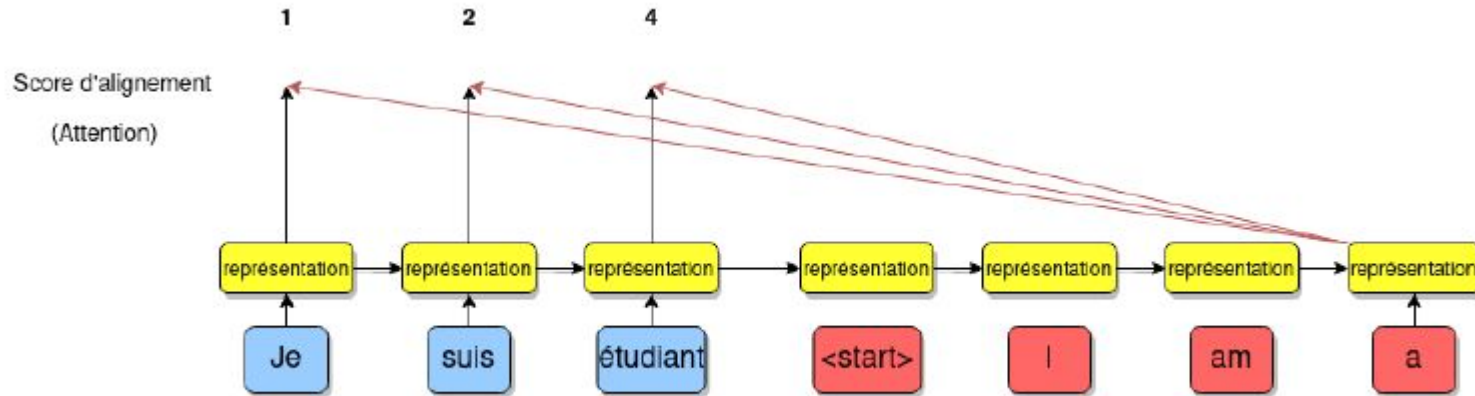
- **at different steps, let a model 'focus' on different parts of the input**

More formally:

- the input sentence corresponds to a set of vectors, all source tokens / RNN states (not only the final state)
- at each step, the decoder decides on which parts of the encoding input it should focus / which source parts are more important

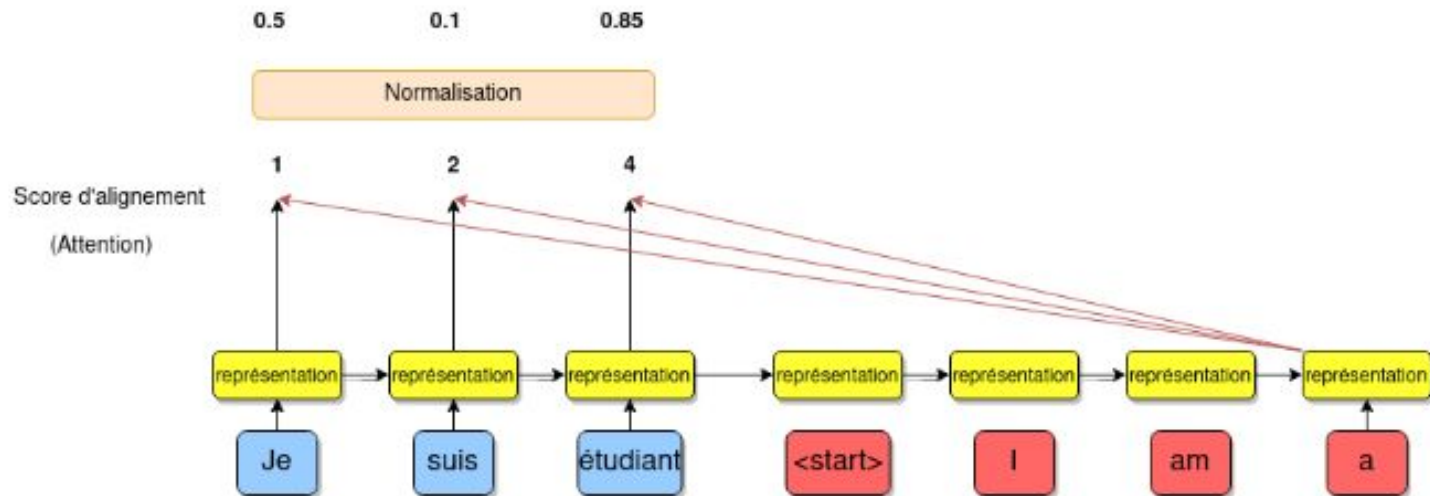
Attention

we try to align the current state of the decoder with relevant inputs from the encoder



Attention

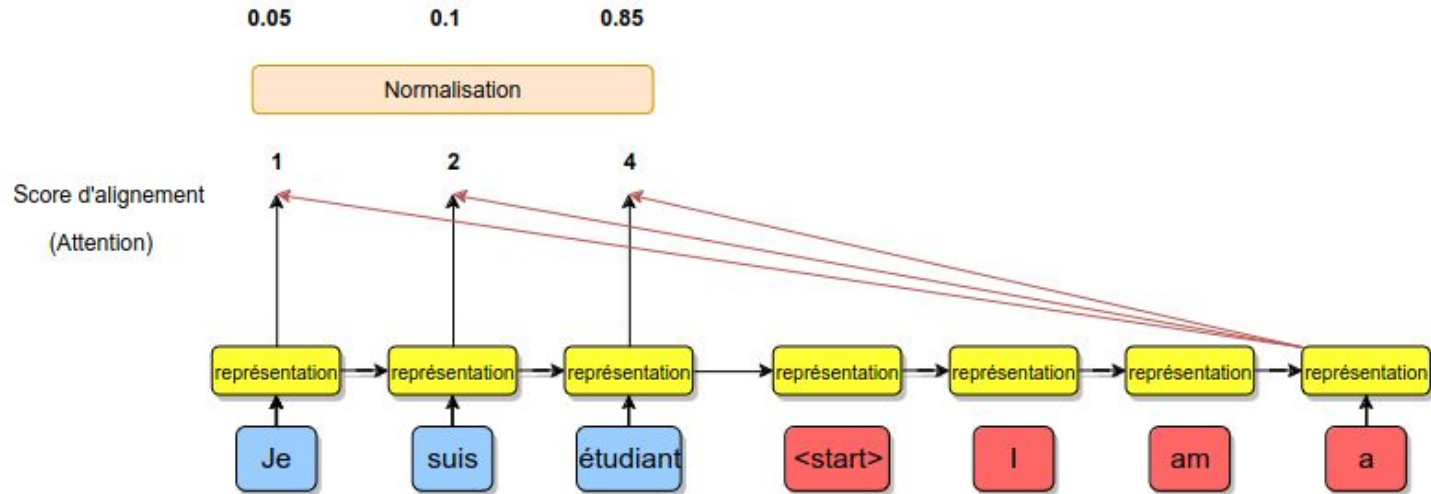
Weight normalization



Attention

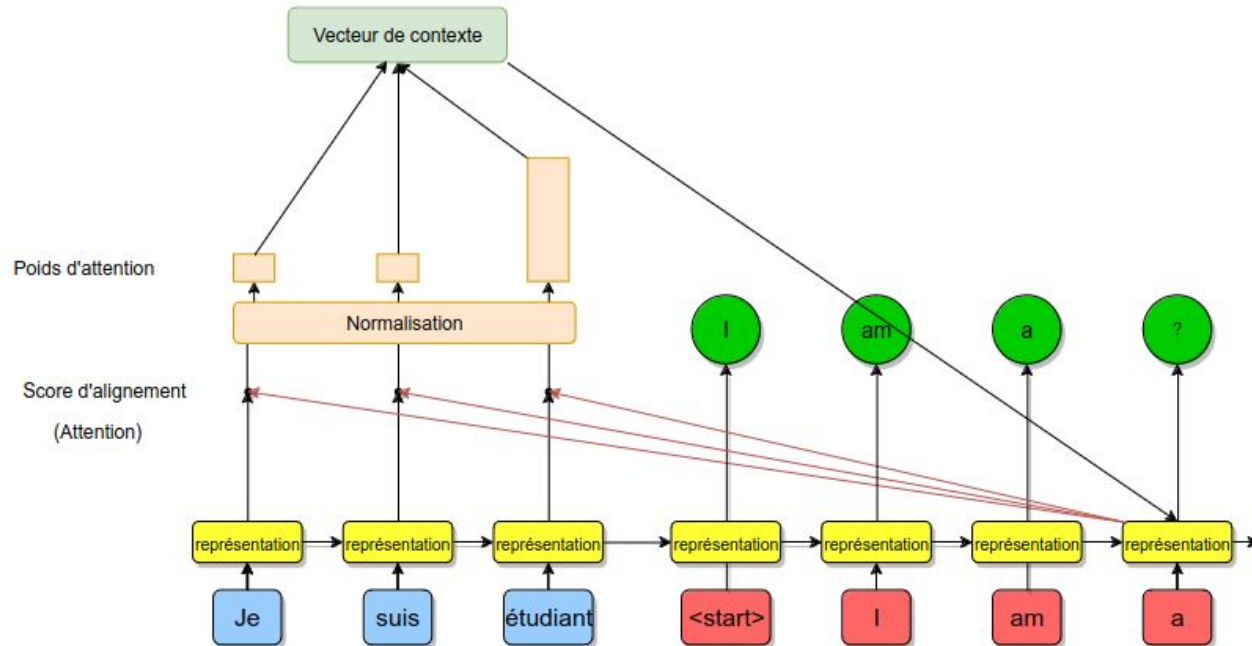
context vector = use the attention weights to make a weighted sum of the encoder inputs

Vecteur de contexte : $0.05 \cdot \text{Je} + 0.1 \cdot \text{suis} + 0.85 \cdot \text{étudiant}$



Attention

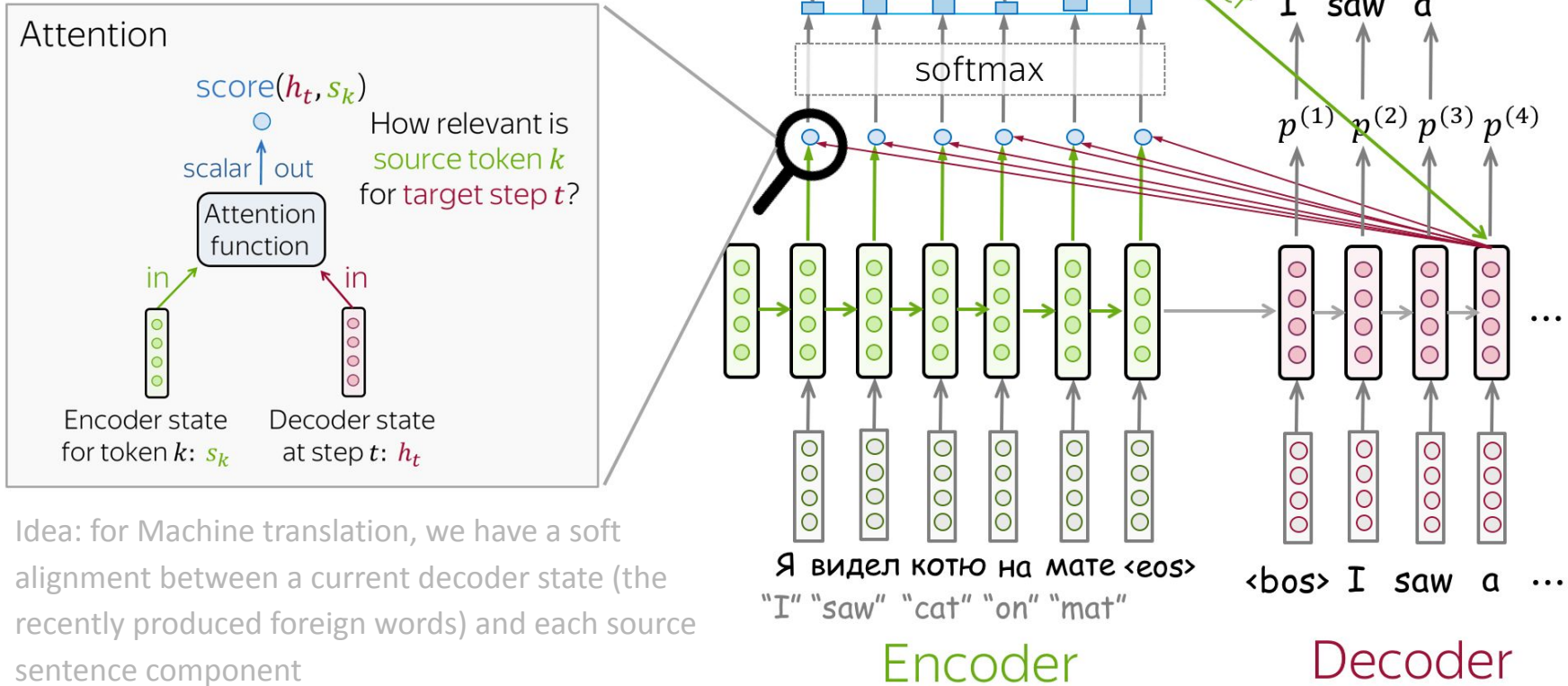
The context vector is combined to the current state of the decoder to make a prediction



Attention output: weighted sum of encoder states with attention weights

Attention weights: distribution over source tokens

A model can learn to “pay attention” to the most relevant source tokens for each step



Encoder-decoder with attention

Steps:

- encode an input sequence $\mathbf{x}_{1:n}$ using a RNN \rightarrow produce \mathbf{n} state vectors $\mathbf{c}_{1:n}$
- the decoder compute the **relevance of the** $\mathbf{c}_{1:n}$ / which of the vectors $\mathbf{c}_{1:n}$ it should attend to \rightarrow **context vector** $\mathbf{c}^j \leftarrow (\mathbf{c}_{1:n}, \mathbf{t}_{1:j})$
- the context vector is used to generate the next token

$$p(t_{j+1} = k \mid \hat{t}_{1:j}, \mathbf{x}_{1:n}) = f(O(s_{j+1}))$$

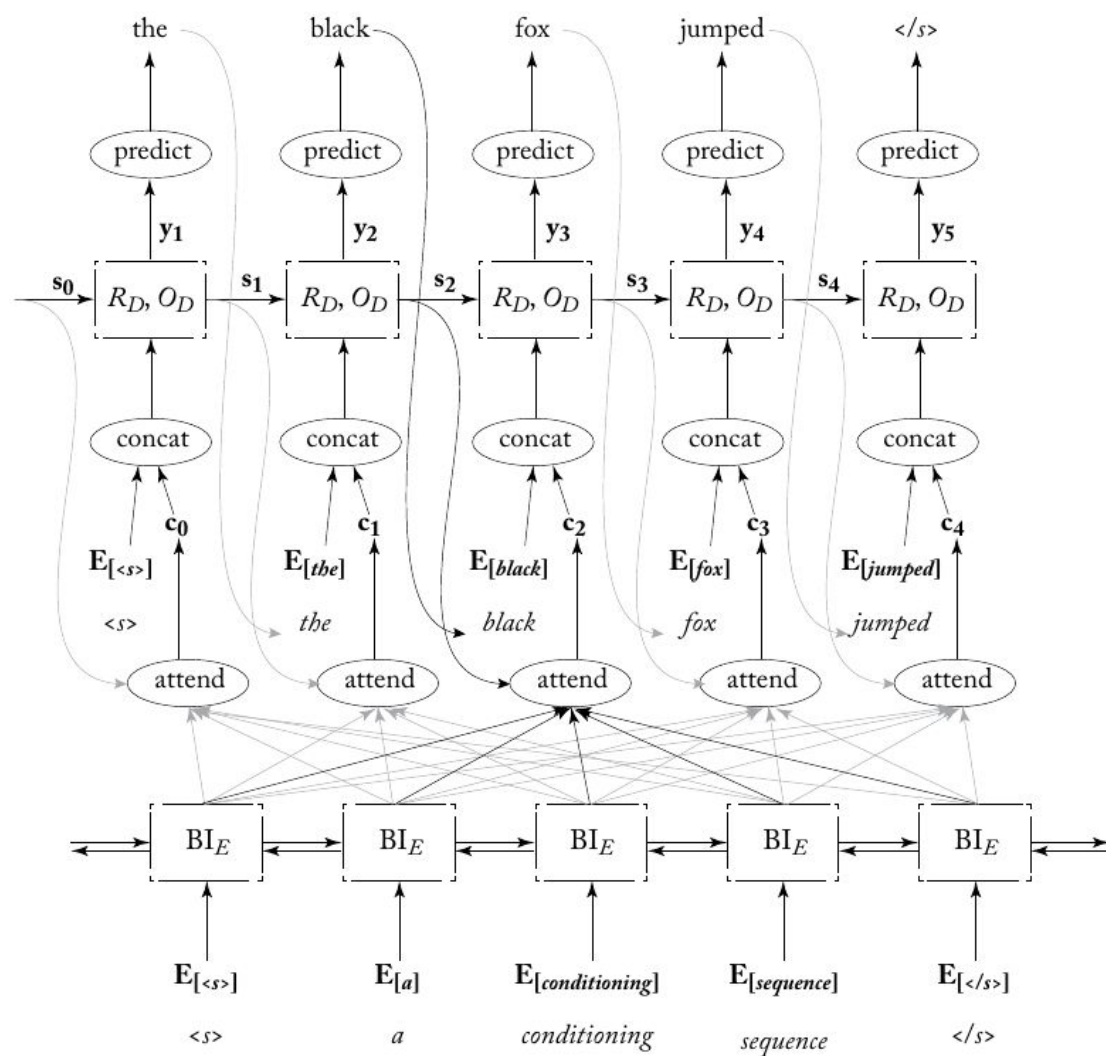
$$s_{j+1} = R(s_j, [\hat{t}_j; \mathbf{c}^j])$$

$$\mathbf{c}^j = \text{attend}(\mathbf{c}_{1:n}, \hat{t}_{1:j})$$

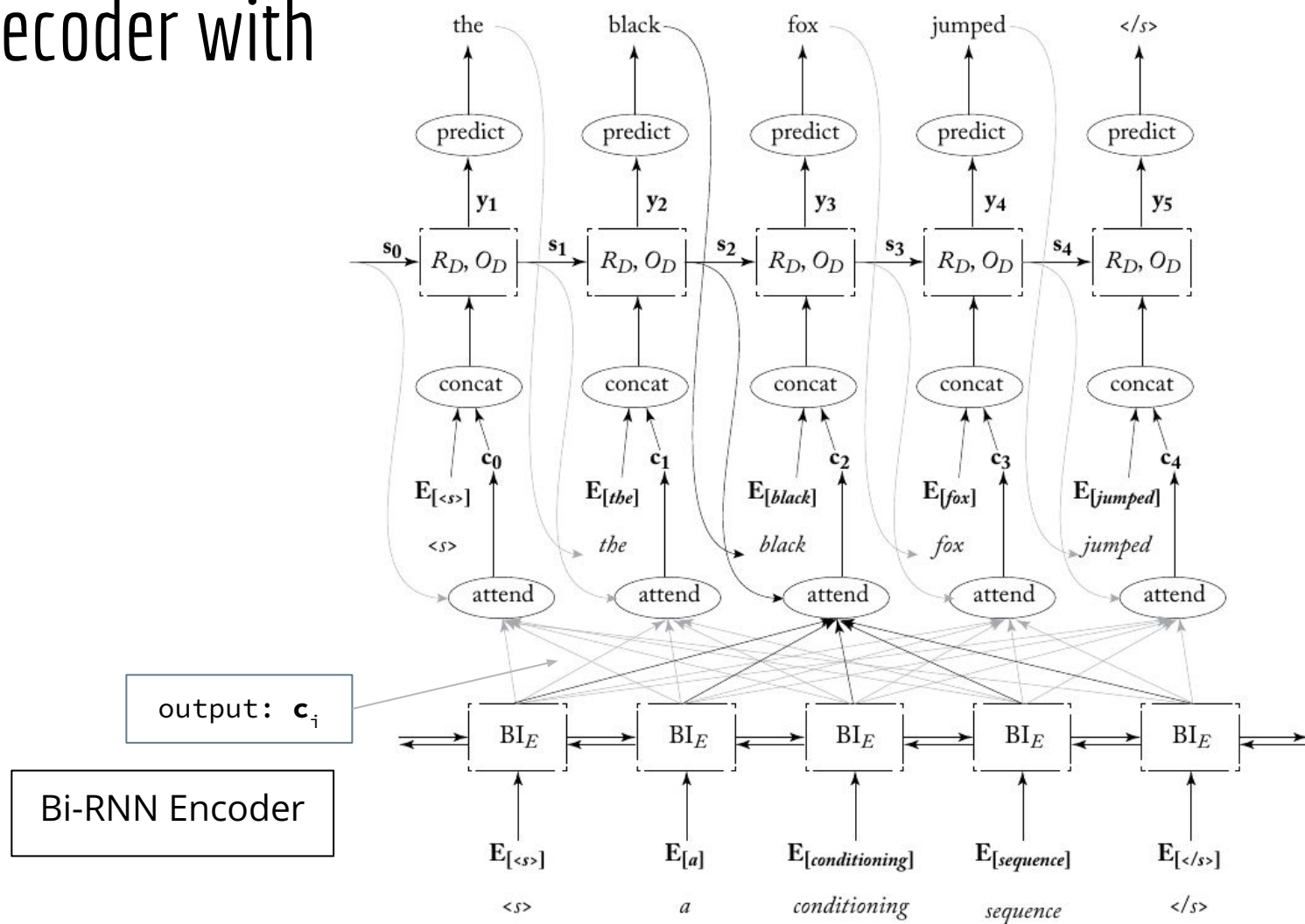
$$\hat{t}_j \sim p(t_j \mid \hat{t}_{1:j-1}, \mathbf{x}_{1:n}).$$

note: f is a function that maps the RNN state to a distribution over words, e.g. softmax

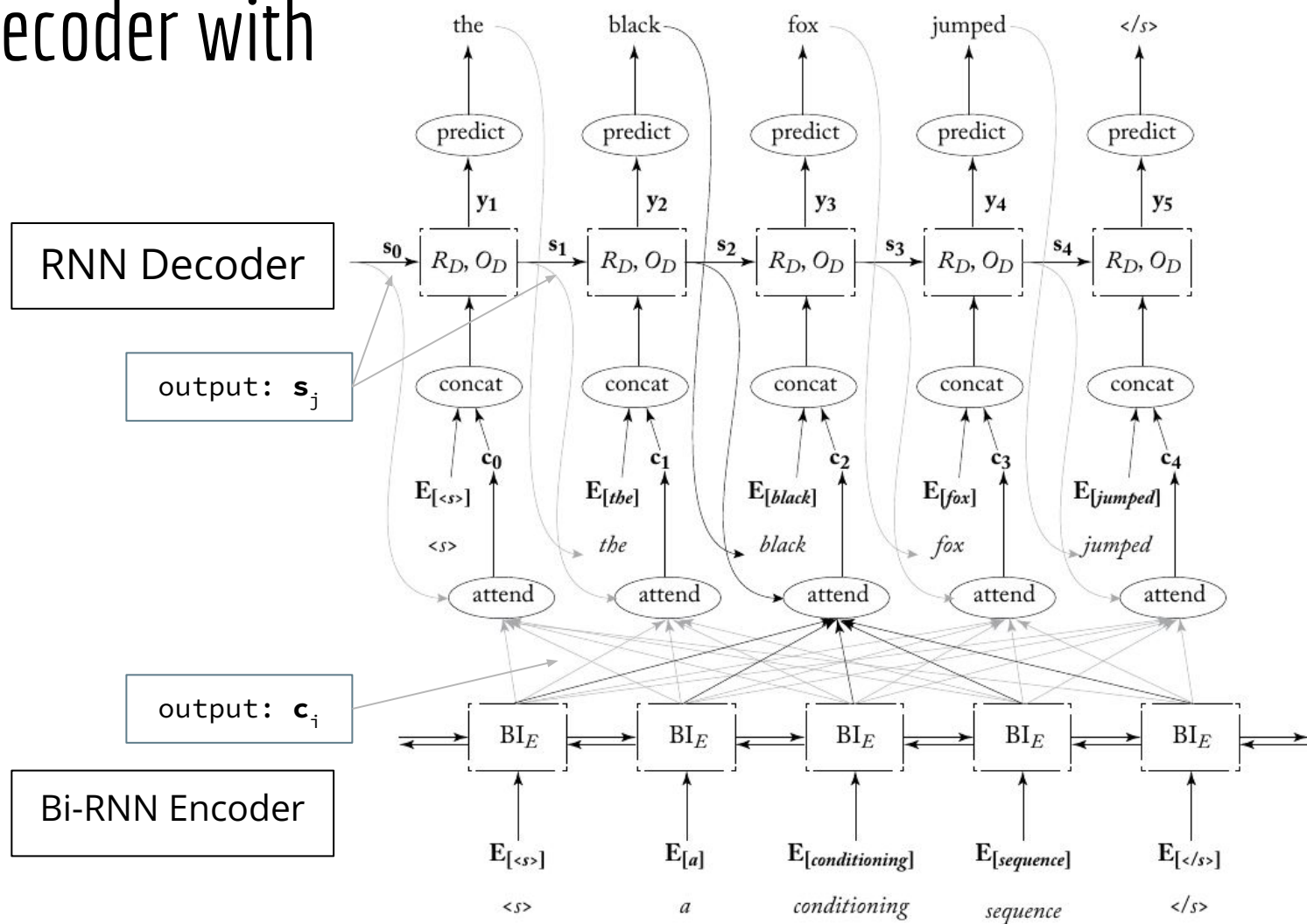
Encoder-decoder with attention



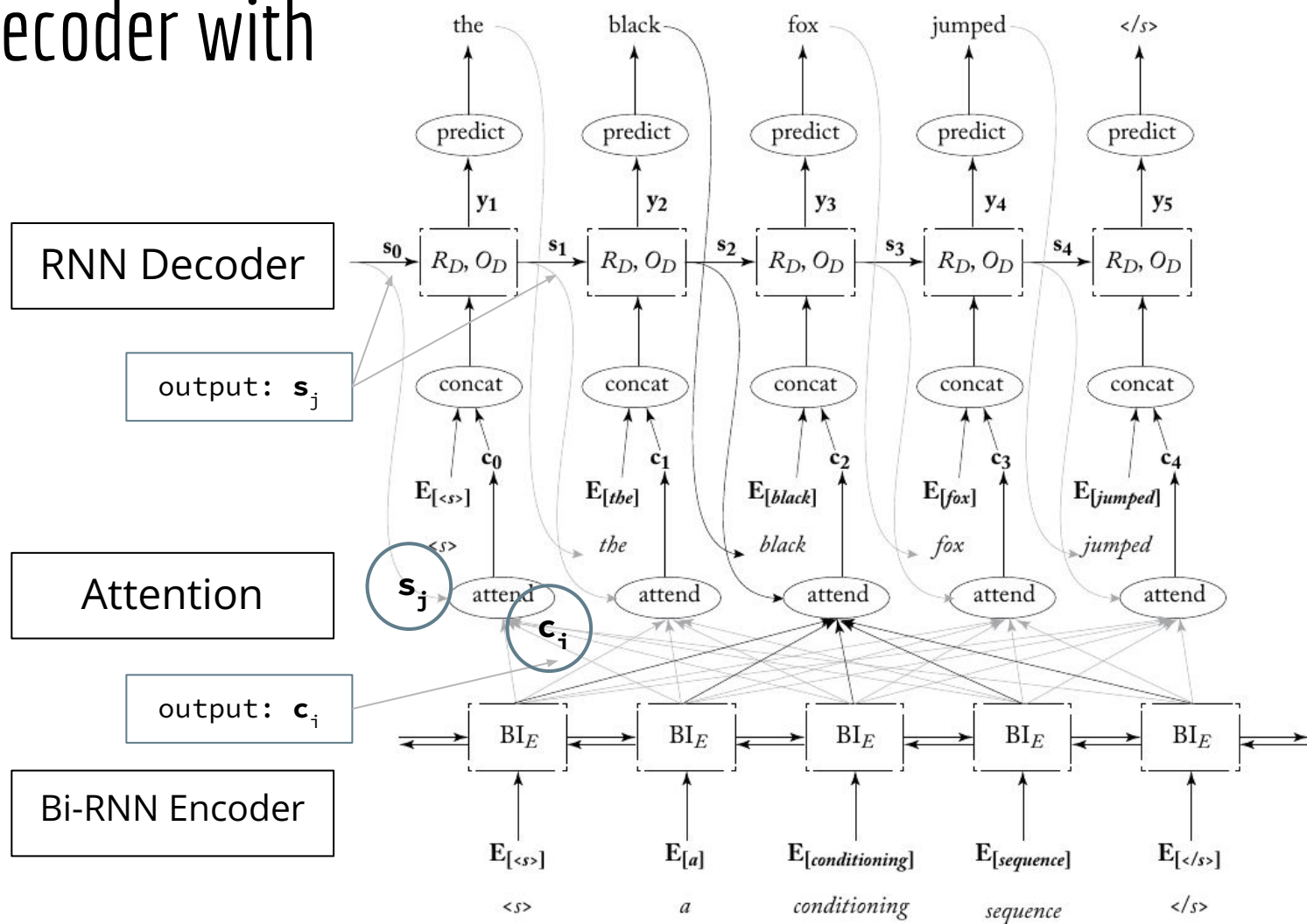
Encoder-decoder with attention



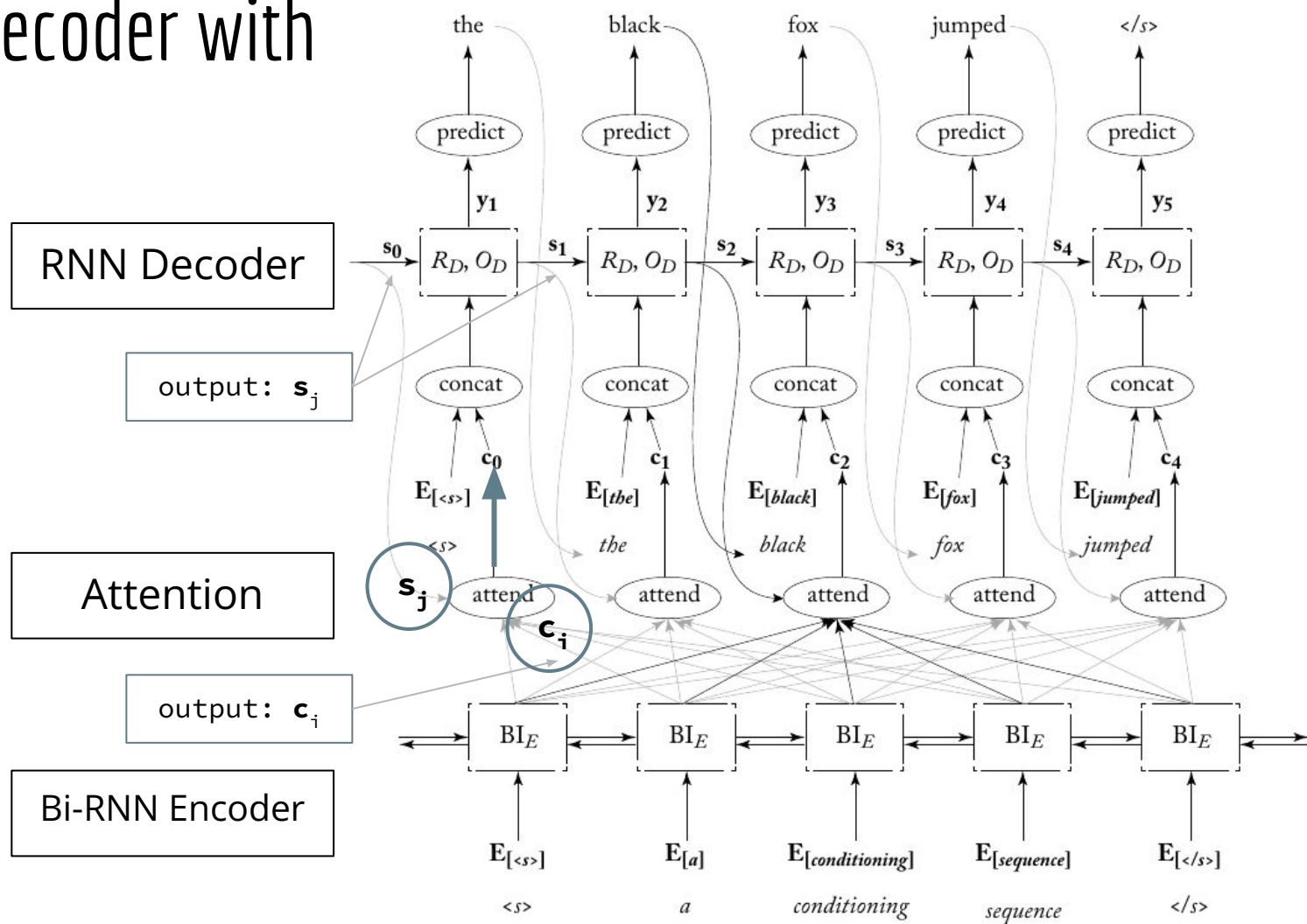
Encoder-decoder with attention



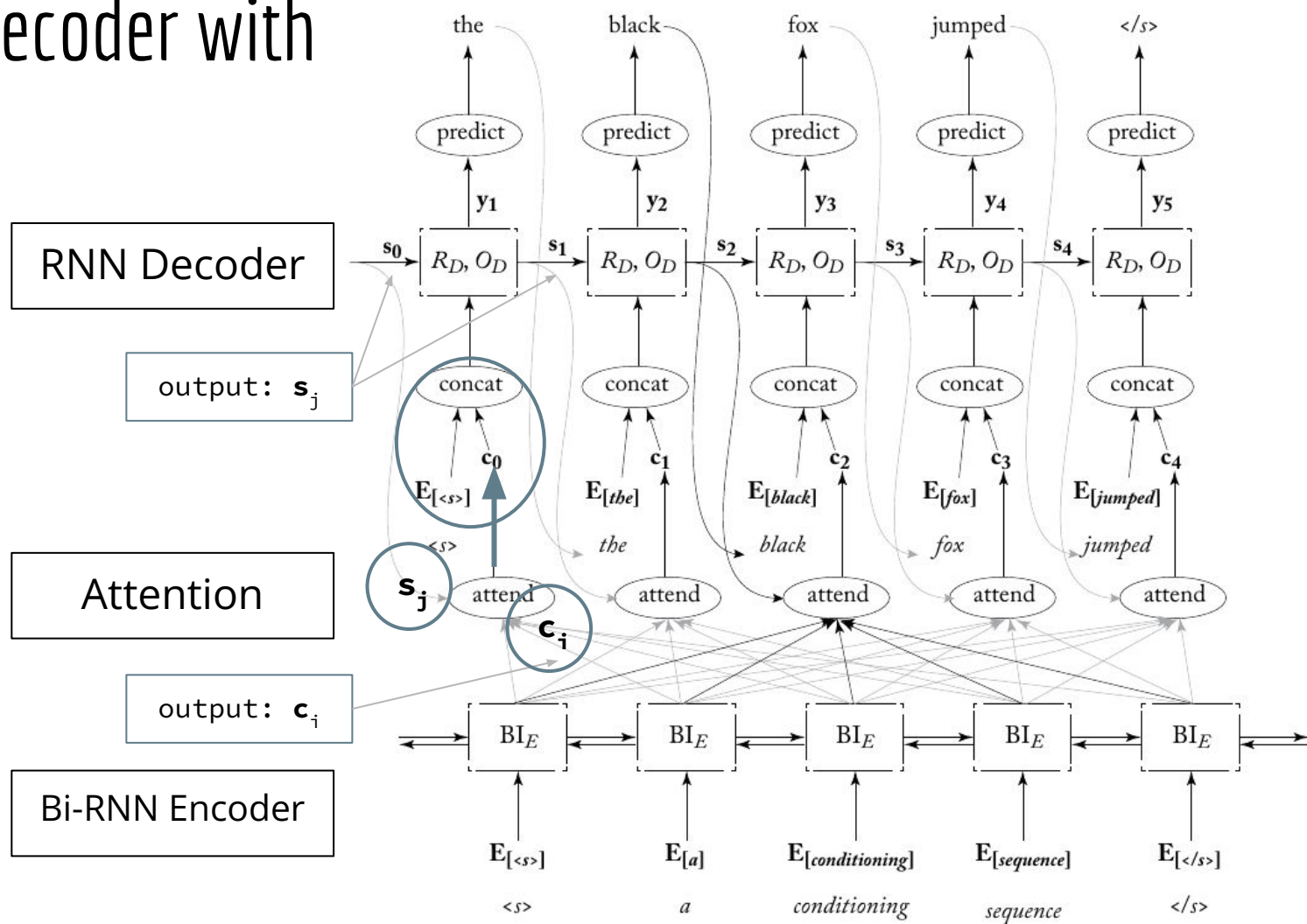
Encoder-decoder with attention



Encoder-decoder with attention

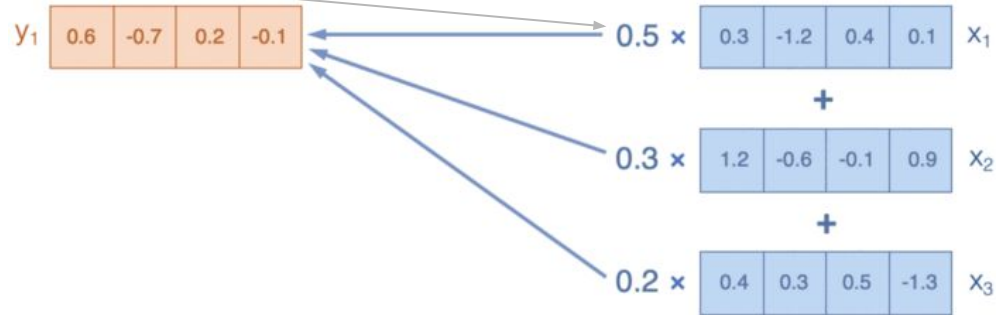


Encoder-decoder with attention



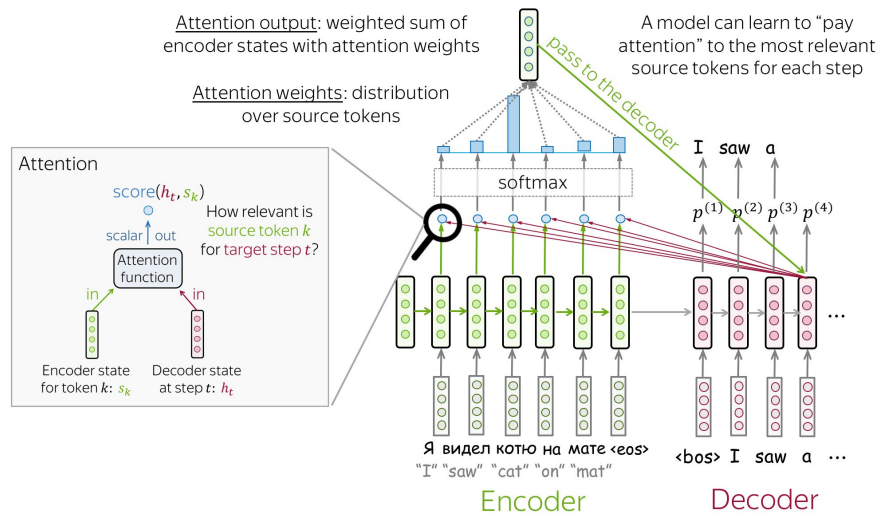
Attention is a weighted average

- Attention is a function that takes some sequence **X** as input and output some sequence **Y**
- where each vector in **Y** is simply a weighted average of the vectors in **X**
- The (attention) weights show how much the model *attends* to each input in **X** when computing the output



- X = word embeddings
- Y = composite of the input word embeddings

Encoder-Decoder with attention



At each decoder step, attention

- receives attention input: a decoder state h_t and all encoder states s_1, s_2, \dots, s_k ;
- computes attention scores: For each encoder state s_k , attention computes its "relevance" for this decoder state h_t . Formally, it applies an attention function which receives one decoder state and one encoder state and returns a scalar value **score(h_t, s_k)** ;
- computes attention weights: a probability distribution - softmax applied to attention scores;
- computes attention output: the weighted sum of encoder states with attention weights.

Encoder-Decoder with attention

Attention output

↑
(weighted
sum)

Attention weights

↑
(softmax)

Attention scores

Attention input

$$c^{(t)} = a_1^{(t)} s_1 + a_2^{(t)} s_2 + \dots + a_m^{(t)} s_m = \sum_{k=1}^m a_k^{(t)} s_k$$

↑
"source context for decoder step t "

$$a_k^{(t)} = \frac{\exp(\text{score}(h_t, s_k))}{\sum_{i=1}^m \exp(\text{score}(h_t, s_i))}, k = 1..m$$

↑
"attention weight for source token k at decoder step t "

$$\text{score}(h_t, s_k), k = 1..m$$

↑
"How relevant is source token k for target step t ?"

s_1, s_2, \dots, s_m h_t
all encoder states one decoder state

Encoder-Decoder with attention

Attention output

(weighted sum)

Attention weights

(softmax)

Attention scores

Attention input

$$c^{(t)} = a_1^{(t)} s_1 + a_2^{(t)} s_2 + \dots + a_m^{(t)} s_m = \sum_{k=1}^m a_k^{(t)} s_k$$

↑
"source context for decoder step t "

$$a_k^{(t)} = \frac{\exp(\text{score}(h_t, s_k))}{\sum_{i=1}^m \exp(\text{score}(h_t, s_i))}, k = 1..m$$

↑
"attention weight for source token k at decoder step t "

$$\text{score}(h_t, s_k), k = 1..m$$

↑
"How relevant is source token k for target step t ?"

s_1, s_2, \dots, s_m
all encoder states

h_t
one decoder state

Dot-product

$$h_t^T \times \begin{bmatrix} \circ \\ \circ \\ \circ \\ \circ \end{bmatrix} s_k$$

$$\text{score}(h_t, s_k) = h_t^T s_k$$

Bilinear

$$h_t^T \times \begin{bmatrix} \text{W} \end{bmatrix} \times \begin{bmatrix} \circ \\ \circ \\ \circ \\ \circ \end{bmatrix} s_k$$

$$\text{score}(h_t, s_k) = h_t^T W s_k$$

Multi-Layer Perceptron

$$w_2^T \times \tanh \left[\begin{bmatrix} \text{W}_1 \end{bmatrix} \times \begin{bmatrix} \circ \\ \circ \\ \circ \\ \circ \end{bmatrix} \begin{bmatrix} h_t \\ s_k \end{bmatrix} \right]$$

$$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1 [h_t, s_k])$$

Encoder-decoder with attention

$$c^j = \text{attend}(c_{1:n}, \hat{t}_{1:j})$$

- the *attend(.)* function should be trainable, parameterized [Bahdanau et al 2014]
- soft attention: at each stage, gives the decoder a weighted average of the vectors $c_{1:n}$

→ the attention weights $\alpha_{[i]}^j$ are chosen by the attention mechanism:

1. produce unnormalized weights based on the decoder state at time j , s_j and the state of the encoder h / c_i (using dot product or more complex function)

Dot-product

$$\begin{matrix} h_t^T \\ \text{---} \end{matrix} \times \begin{matrix} \begin{bmatrix} \circ \\ \circ \\ \circ \\ \circ \end{bmatrix} \\ s_k \end{matrix}$$

$$\text{score}(h_t, s_k) = h_t^T s_k$$

Bilinear

$$\begin{matrix} h_t^T \\ \text{---} \end{matrix} \times \begin{bmatrix} W \end{bmatrix} \times \begin{matrix} \begin{bmatrix} \circ \\ \circ \\ \circ \\ \circ \end{bmatrix} \\ s_k \end{matrix}$$

$$\text{score}(h_t, s_k) = h_t^T W s_k$$

Multi-Layer Perceptron

$$\begin{matrix} w_2^T \\ \text{---} \end{matrix} \times \tanh \left[\begin{matrix} \begin{bmatrix} W_1 \end{bmatrix} \times \begin{matrix} \begin{bmatrix} \circ \\ \circ \\ \circ \\ \circ \end{bmatrix} \\ h_t \\ \begin{bmatrix} \circ \\ \circ \\ \circ \\ \circ \end{bmatrix} \\ s_k \end{matrix} \end{matrix} \right]$$

$$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1 [h_t, s_k])$$

2. normalize the weights into a probability distribution (sum to 1) using softmax
3. the final context vector is

$$c^j = \sum_{i=1} \alpha_{[i]}^j \cdot c_i$$

Encoder-decoder with attention

The complete attend function is then:

$$\text{attend}(\mathbf{c}_{1:n}, \hat{t}_{1:j}) = \mathbf{c}^j$$

$$\mathbf{c}^j = \sum_{i=1}^n \alpha_{[i]}^j \cdot \mathbf{c}_i$$

$$\alpha^j = \text{softmax}(\bar{\alpha}_{[1]}^j, \dots, \bar{\alpha}_{[n]}^j)$$

$$\bar{\alpha}_{[i]}^j = \text{MLP}^{\text{att}}([s_j; \mathbf{c}_i]),$$

and the entire sequence-to-sequence generation with attention is given by:

$$p(t_{j+1} = k \mid \hat{t}_{1:j}, \mathbf{x}_{1:n}) = f(O_{\text{dec}}(s_{j+1}))$$

$$s_{j+1} = R_{\text{dec}}(s_j, [\hat{t}_j; \mathbf{c}^j])$$

$$\mathbf{c}^j = \sum_{i=1}^n \alpha_{[i]}^j \cdot \mathbf{c}_i$$

$$\mathbf{c}_{1:n} = \text{biRNN}_{\text{enc}}^*(\mathbf{x}_{1:n})$$

$$\alpha^j = \text{softmax}(\bar{\alpha}_{[1]}^j, \dots, \bar{\alpha}_{[n]}^j)$$

$$\bar{\alpha}_{[i]}^j = \text{MLP}^{\text{att}}([s_j; \mathbf{c}_i])$$

$$\hat{t}_j \sim p(t_j \mid \hat{t}_{1:j-1}, \mathbf{x}_{1:n})$$

$$f(\mathbf{z}) = \text{softmax}(\text{MLP}^{\text{out}}(\mathbf{z}))$$

$$\text{MLP}^{\text{att}}([s_j; \mathbf{c}_i]) = \mathbf{v} \tanh([s_j; \mathbf{c}_i]U + \mathbf{b}).$$

Encoder-decoder with attention

- why using attention vectors instead of the x_i directly?

→ take into account the context (window) + trainable (may learn properties e.g. the position of x_i)

- computationally more complex (but really powerful)
- helps interpretability: at each stage of the decoding process, we can look at the produced attention weights and see which parts of the input were used

Application: Machine translation

State-of-the-art architecture for MT: [Bahdanau et al 2015] bi-GRU, beam-search ; some improvements:

- **Sub-word units** [Sennrich et al 2016]: allow to deal with highly inflected languages (and restrict size of the vocabulary). Also character level [Chung et al 2016]
- **Linguistic annotations**: [Sennrich and Haddow 2016] the sentence is run through a pipeline incl. POS tagging, syntactic parsing, lemmatization. Each word is then supplemented with a vector encoding this info (concatenated) → linguistic info is useful even with powerful NN architectures!
- Incorporating **monolingual data** for translation models [Sennrich et al 2016]

Application: Machine translation

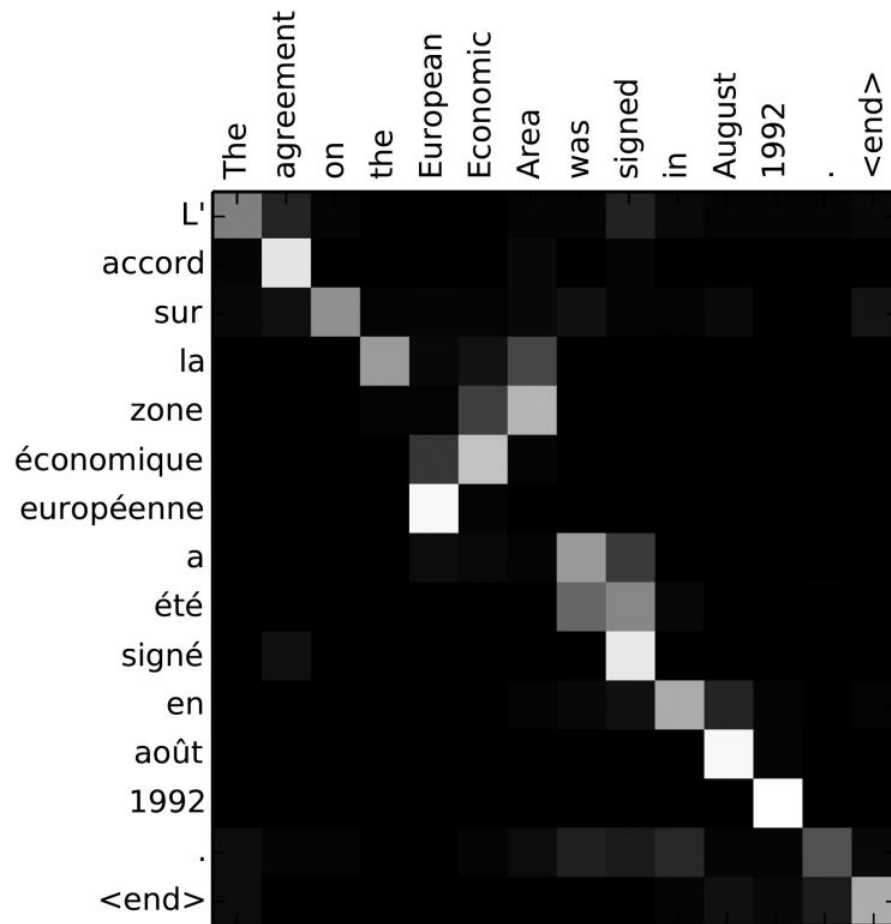
State-of-the-art architecture for MT: [Bahdanau et al 2015] bi-GRU, beam-search ; some improvements:

- **Sub-word units** [Sennrich et al 2016]: allow to deal with highly inflected languages (and restrict size of the vocabulary). Also character level [Chung et al 2016]
- **Linguistic annotations**: [Sennrich and Haddow 2016] the sentence is run through a pipeline incl. POS tagging, syntactic parsing, lemmatization. Each word is then supplemented with a vector encoding this info (concatenated) → linguistic info is useful even with powerful NN architectures!
- Incorporating **monolingual data**: previously, systems were based on a translation model (parallel data) + a separate language model (monolingual data), but seq2seq models does not allow such a separation. [Sennrich et al 2016]: train a translation model from target to source, use it to translate a large monolingual corpus of target sentences, add the resulting pairs (source,

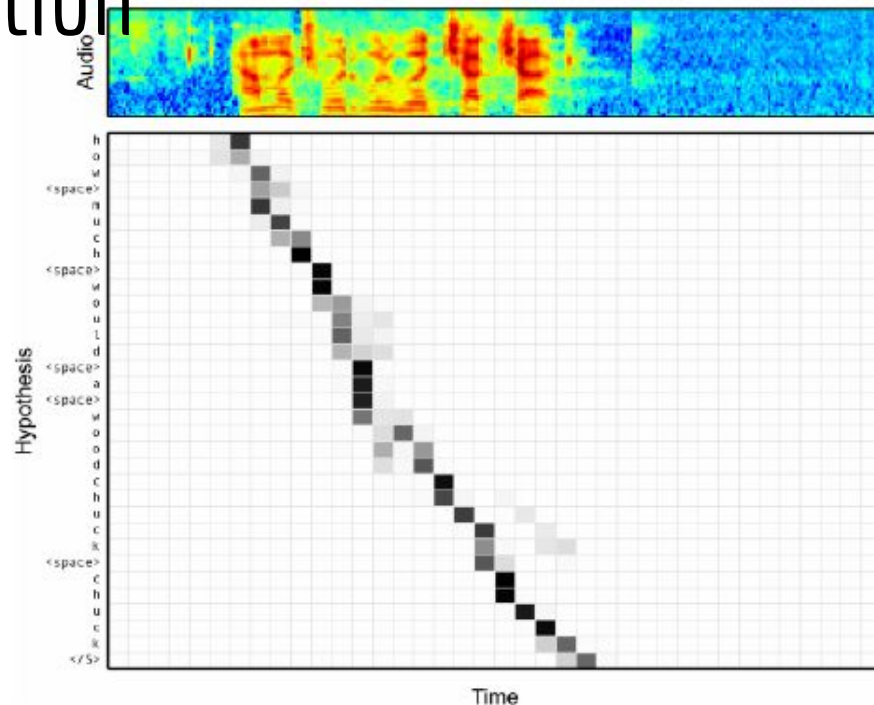
Machine translation

Visualization of the alignment

from [\[Bahdanau et al 2015\]](#)

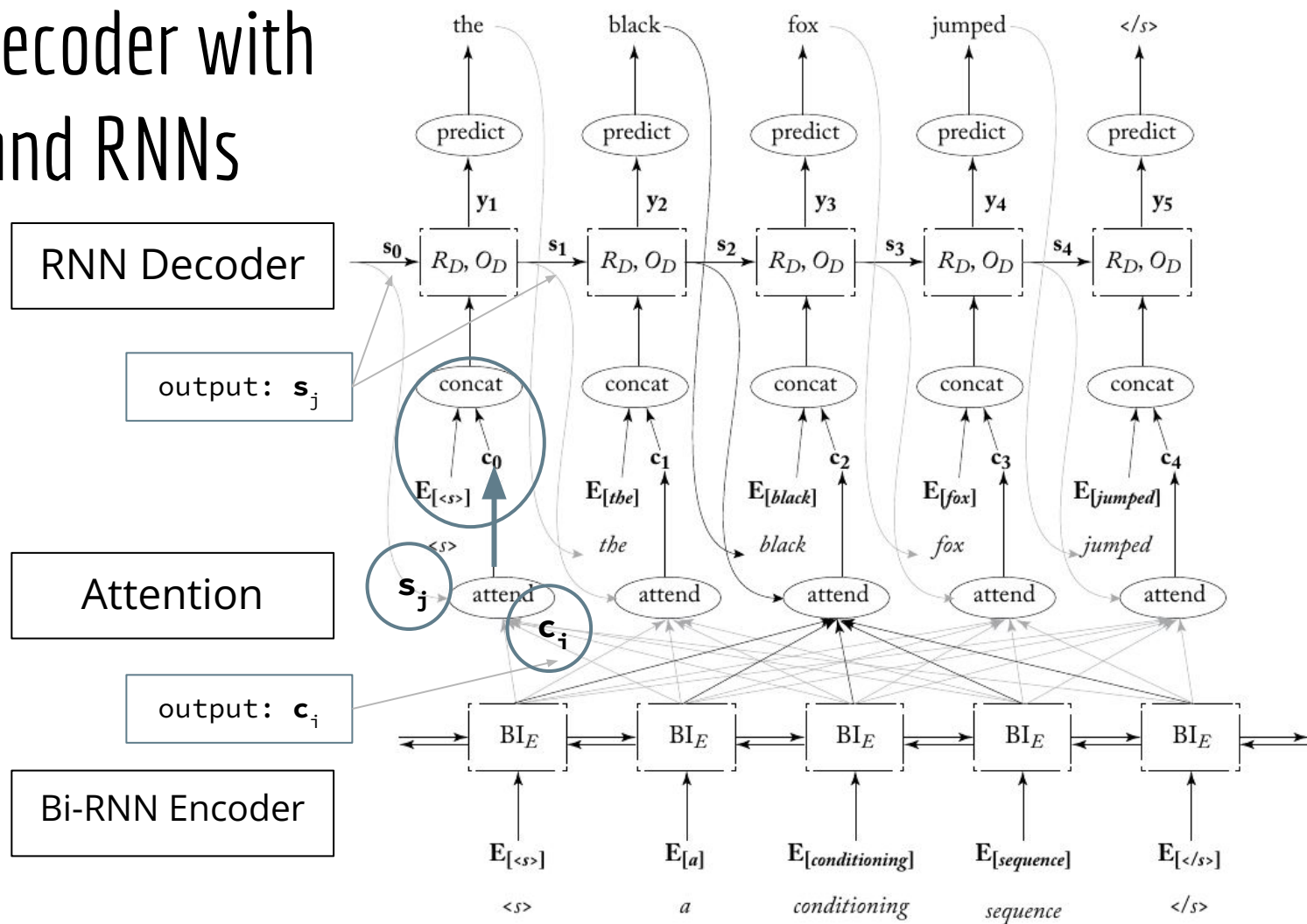


Speech recognition



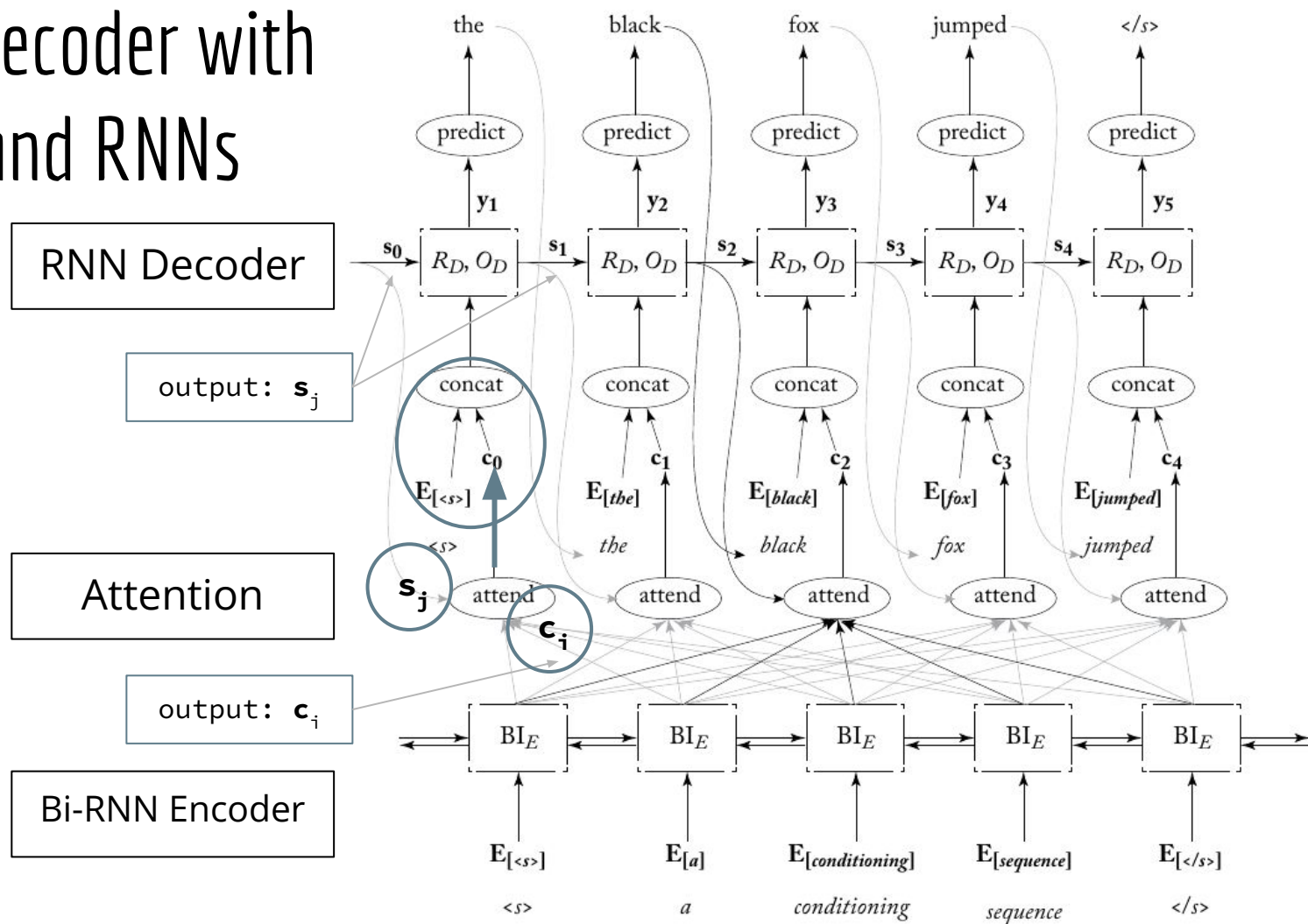
Source : Chan, Jaitly, Le, Vinyals : *Listen, attend and spell : A neural network for large vocabulary conversational speech recognition.*
ICASSP 2016

Encoder-decoder with attention and RNNs



Encoder-decoder with attention and RNNs

now what if we try to remove the RNNs?



Attention is all you need

Transformer models:

- also takes sequence as input
- but based on attention mechanism **without the RNN architecture**
- = it is not required to read **in any order** the sequence

→ make it easier to parallelize computation: thus to train on larger corpora, leading to BERT, GPT language models

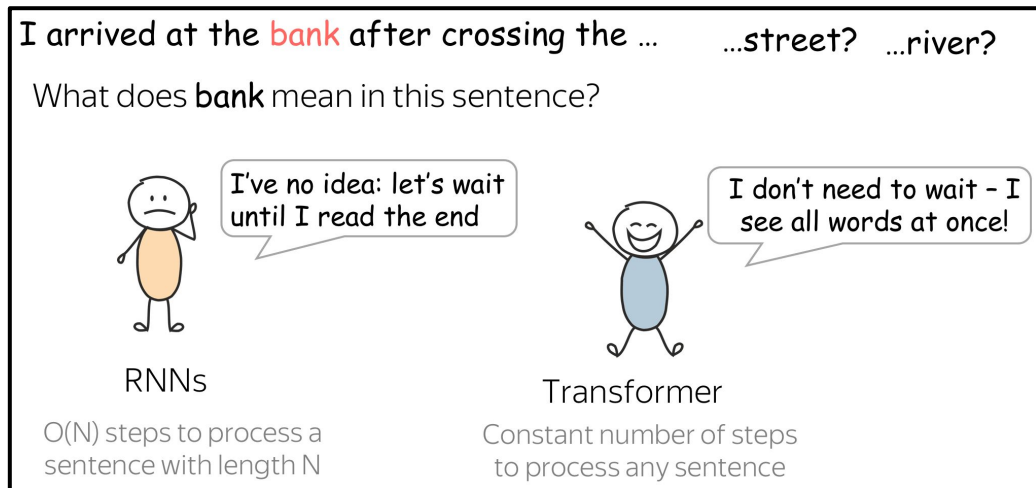
[Vaswani et al 2017] <https://arxiv.org/abs/1706.03762> : new state-of-the-art on Machine translation (with “only” 3.5 days on eight GPUs :D), high performance for constituency parsing

<https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>

	Seq2seq without attention	Seq2seq with attention	Transformer
processing within encoder	RNN/CNN	RNN/CNN	attention
processing within decoder	RNN/CNN	RNN/CNN	attention
decoder-encoder interaction	static fixed-sized vector	attention	attention

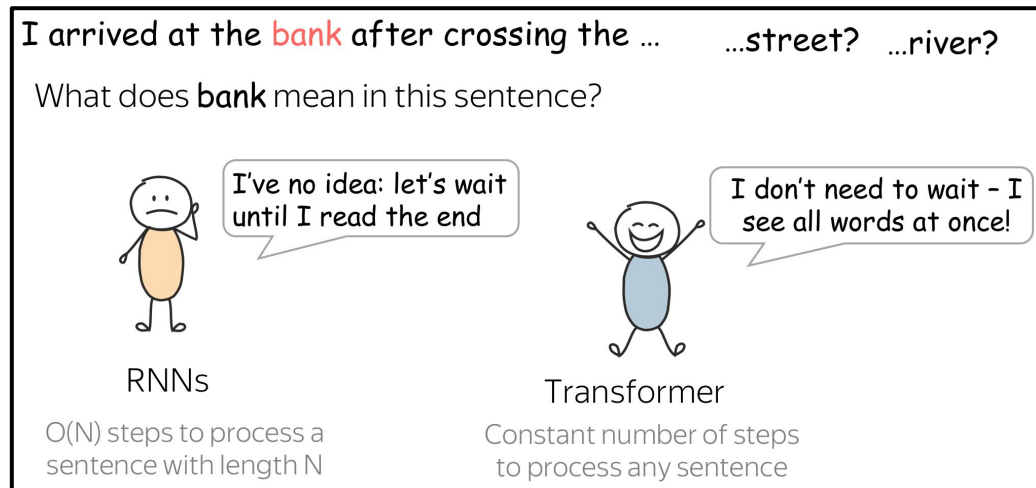
General idea

- When encoding a sentence, RNNs won't understand what **bank** means until they read the whole sentence,
- Transformer's encoder tokens interact with each other all at once.



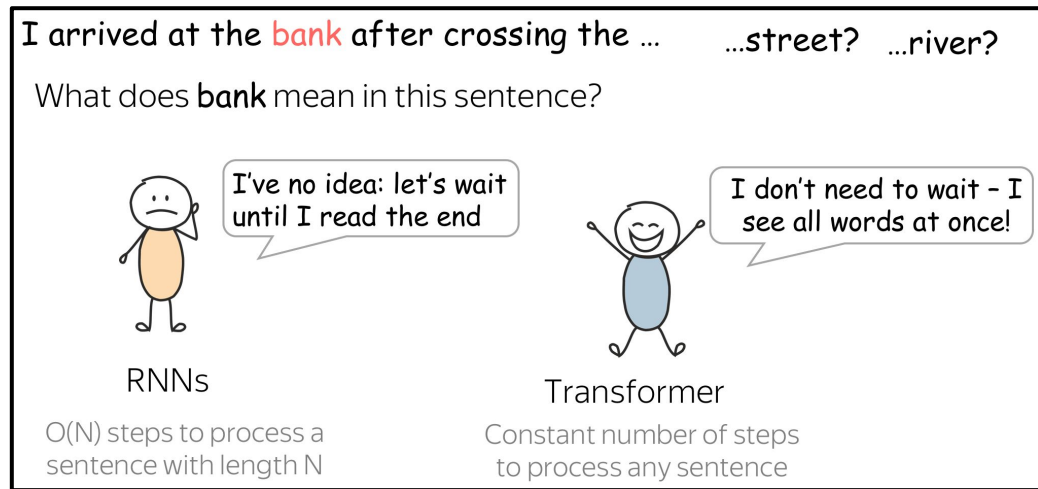
General idea

- When encoding a sentence, RNNs won't understand what **bank** means until they read the whole sentence,
- Transformer's encoder tokens interact with each other all at once.
- Transformer's encoder: **at each step, tokens look at each other = self-attention**, extract information and try to understand each other better in the context of the whole sentence

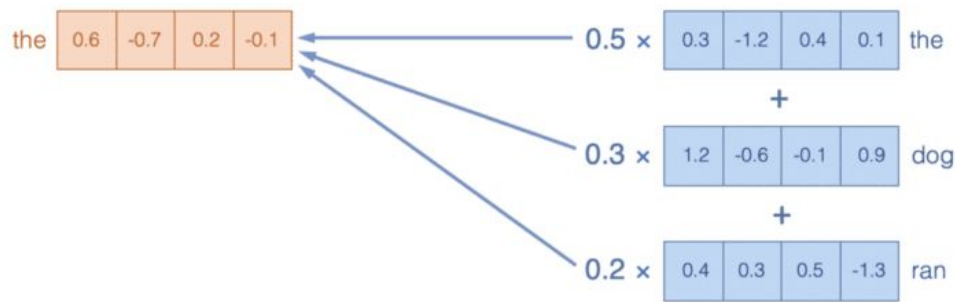


General idea

- When encoding a sentence, RNNs won't understand what **bank** means until they read the whole sentence,
- Transformer's encoder tokens interact with each other all at once.
- Transformer's encoder: **at each step, tokens look at each other = self-attention**, extract information and try to understand each other better in the context of the whole sentence
- Transformer's decoder: tokens predicted **also interact with each other** + look at the encoder states



Self-Attention



Self-Attention = Attention over the sequence itself

Transformer model: relies entirely on **self-attention** to compute representations of its input and output (without using sequence aligned RNNs or convolution)

→ the model must understand how the words relate to each other in the context of the sentence

- used for reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [Cheng et al 2016, Parikh et al 2016, Lin et al 2017, Paulus et al 2017]

Each vector receives three representations (“roles”)

$$\begin{bmatrix} W_Q \end{bmatrix} \times \begin{bmatrix} \text{green} \\ \text{green} \\ \text{green} \end{bmatrix} = \begin{bmatrix} \text{blue} \\ \text{blue} \\ \text{blue} \end{bmatrix}$$

Query: vector **from** which the attention is looking

“Hey there, do you have this information?”

$$\begin{bmatrix} W_K \end{bmatrix} \times \begin{bmatrix} \text{green} \\ \text{green} \\ \text{green} \end{bmatrix} = \begin{bmatrix} \text{yellow} \\ \text{yellow} \\ \text{yellow} \end{bmatrix}$$

Key: vector **at** which the query looks to compute weights

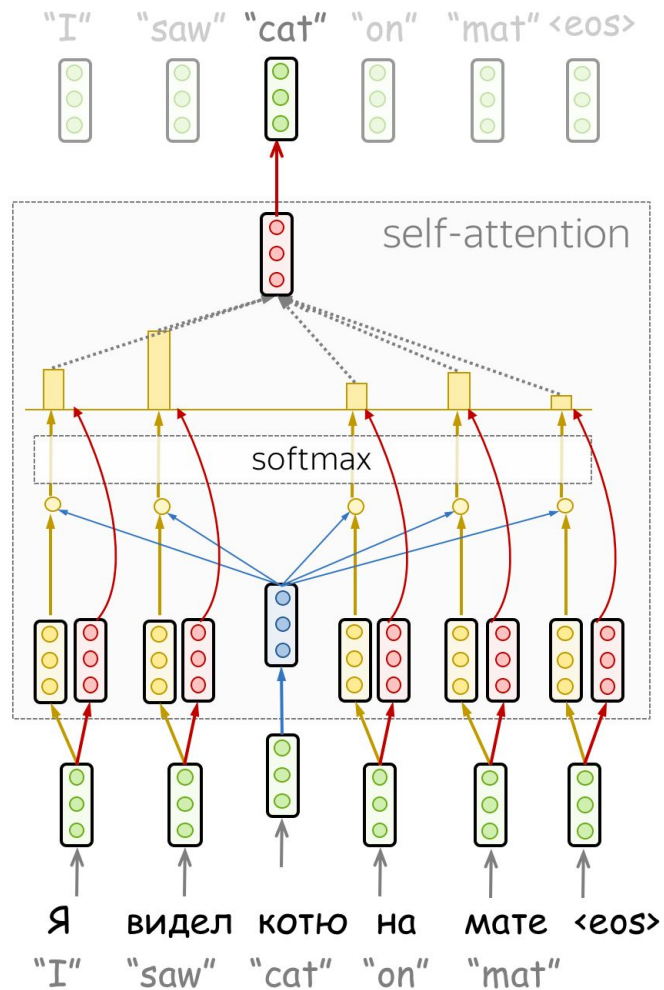
“Hi, I have this information – give me a large weight!”

$$\begin{bmatrix} W_V \end{bmatrix} \times \begin{bmatrix} \text{green} \\ \text{green} \\ \text{green} \end{bmatrix} = \begin{bmatrix} \text{red} \\ \text{red} \\ \text{red} \end{bmatrix}$$

Value: their weighted sum is attention output

“Here’s the information I have!”

Attention is a “query” on the inputs, that we map to a “key” to operate on a specific input with a specific “value”



Each vector receives three representations (“roles”)

$$\begin{bmatrix} W_Q \end{bmatrix} \times \begin{bmatrix} \text{green} \\ \text{green} \\ \text{green} \end{bmatrix} = \begin{bmatrix} \text{blue} \\ \text{blue} \\ \text{blue} \end{bmatrix}$$

Query: vector **from** which the attention is looking

“Hey there, do you have this information?”

$$\begin{bmatrix} W_K \end{bmatrix} \times \begin{bmatrix} \text{green} \\ \text{green} \\ \text{green} \end{bmatrix} = \begin{bmatrix} \text{yellow} \\ \text{yellow} \\ \text{yellow} \end{bmatrix}$$

Key: vector **at** which the query looks to compute weights

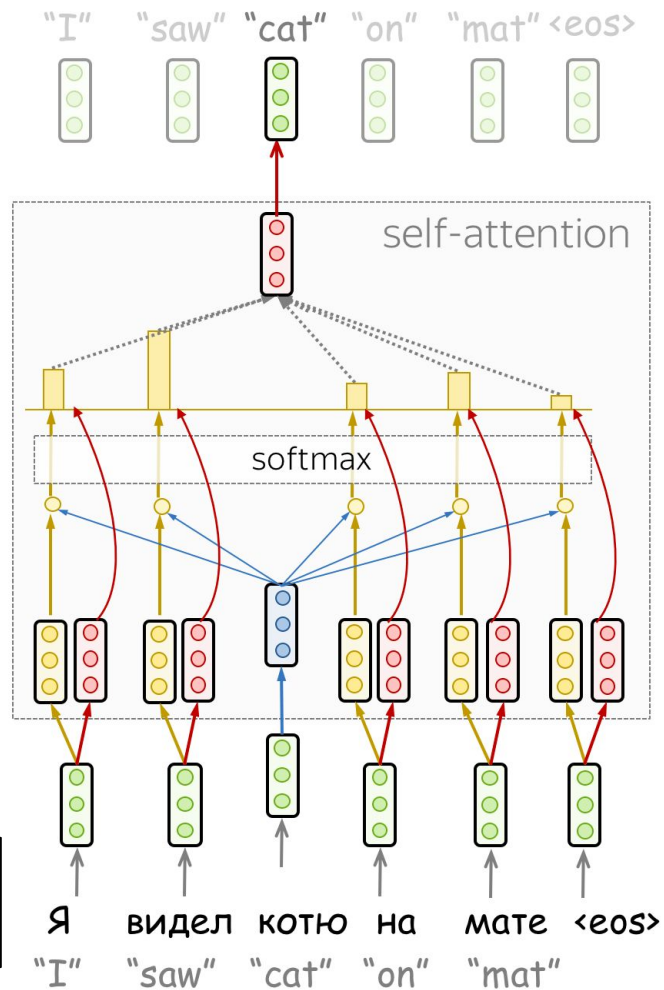
“Hi, I have this information – give me a large weight!”

$$\begin{bmatrix} W_V \end{bmatrix} \times \begin{bmatrix} \text{green} \\ \text{green} \\ \text{green} \end{bmatrix} = \begin{bmatrix} \text{red} \\ \text{red} \\ \text{red} \end{bmatrix}$$

Value: their weighted sum is attention output

“Here’s the information I have!”

Note: masked attention for the decoder = it can’t look ahead



Computing attention

- first: assigns to each word a *query* vector and a *key* vector
- compute a *compatibility function*: assigns a score to each pair of words indicating how strongly they should attend to one another, using dot product between one query and one key $\mathbf{w}_{ij} = \mathbf{q}_i \mathbf{k}_j$

	query				key			
The	0.7	0.6	-0.4		0.5	-0.9	0.2	The
dog	0.3	-0.2	0.4		1.1	-0.3	0.5	dog
ran	-1.2	0.1	0.9		-1.0	0.3	-0.7	ran

$$\text{dog} \begin{bmatrix} 0.3 & -0.2 & 0.4 \end{bmatrix} \cdot \begin{bmatrix} -1.0 & 0.3 & -0.7 \end{bmatrix} \text{ran} = \begin{bmatrix} -0.6 \end{bmatrix}$$

- then normalize the scores: to be positive and sum to one (softmax)

	query				key				score	softmax
dog	0.3	-0.2	0.4	•	0.5	-0.9	0.2	The	= 0.4	0.4
dog	0.3	-0.2	0.4	•	1.1	-0.3	0.5	dog	= 0.6	0.5
dog	0.3	-0.2	0.4	•	-1.0	0.3	-0.7	ran	= -0.6	0.1

Final
attention
weights

Computing attention

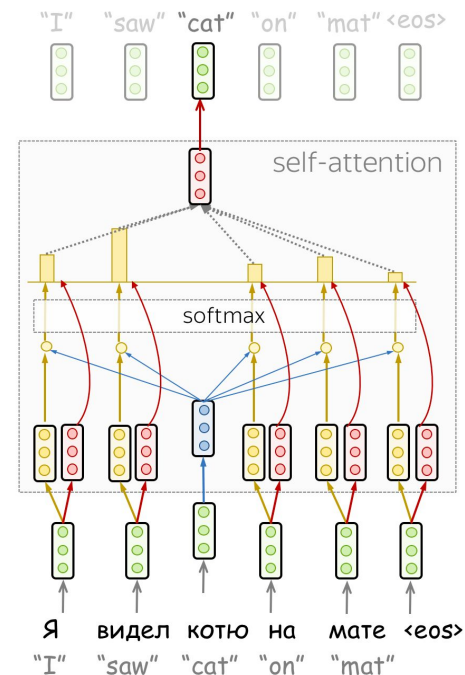
- query: interaction between x_i and other x_j to compute an attention score x_i, x_j (< current state of the decoder)
- key: used to compute the weights when another x_j is the query (sim. to s_k)
- value: used for the final computation of y_j , for the weighted sum (s_k to compute c)

Each vector receives three representations ("roles")

$[W_Q] \times \begin{bmatrix} \text{green} \\ \text{green} \\ \text{green} \end{bmatrix} = \begin{bmatrix} \text{blue} \\ \text{blue} \\ \text{blue} \end{bmatrix}$ **Query:** vector **from** which the attention is looking
 "Hey there, do you have this information?"

$[W_K] \times \begin{bmatrix} \text{green} \\ \text{green} \\ \text{green} \end{bmatrix} = \begin{bmatrix} \text{yellow} \\ \text{yellow} \\ \text{yellow} \end{bmatrix}$ **Key:** vector **at** which the query looks to compute weights
 "Hi, I have this information - give me a large weight!"

$[W_V] \times \begin{bmatrix} \text{green} \\ \text{green} \\ \text{green} \end{bmatrix} = \begin{bmatrix} \text{red} \\ \text{red} \\ \text{red} \end{bmatrix}$ **Value:** their weighted sum is attention output
 "Here's the information I have!"

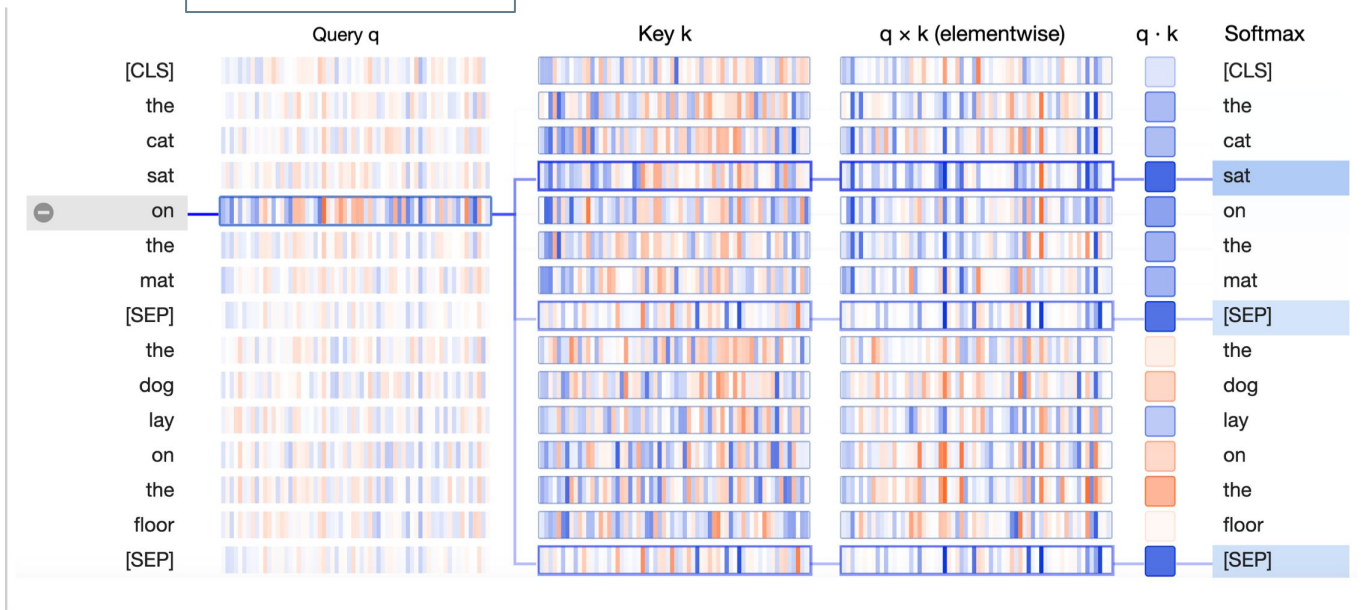


Computing attention

query: the word that
is paying attention /
querying the other
words

key: the word to
which attention is
being paid

compatibility
score +
normalized

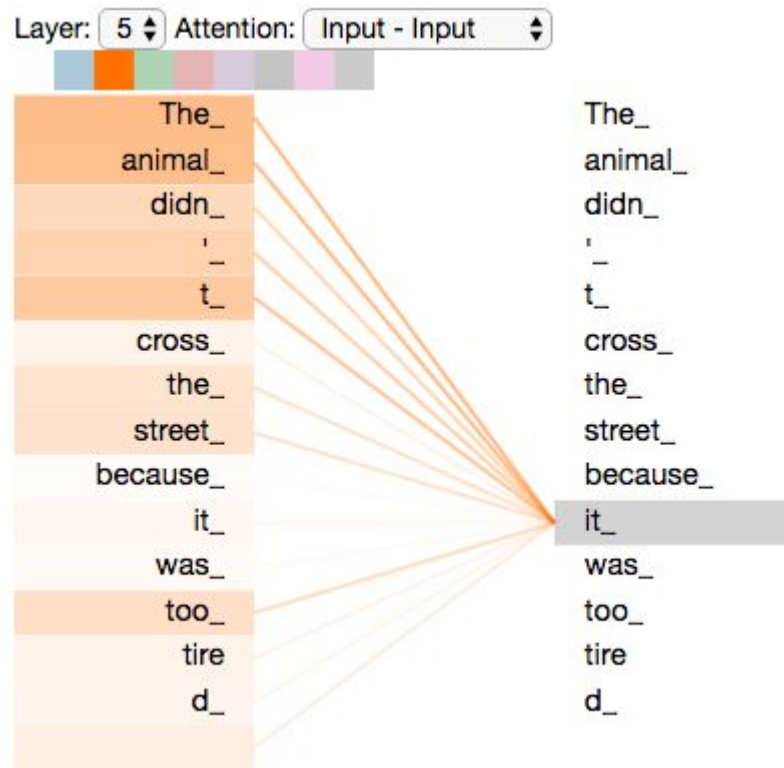


Self-Attention

Visualization:

- the model puts a large attention weight between “the” and “animal” and “it”, allowing to ‘understand’ that “it” refers to “animal”

→ similar to the memory of RNNs,
allow to keep an history



Multi-head attention

Multiple attention mechanisms = called *heads*

- operate in parallel to one another / independently **focus on different things**
- expand ability to focus on many positions
- enables the model to capture a broader range of relationships between words
- the attention heads do not share parameters, each head learns a unique attention pattern
- If we do the same self-attention calculation eight different times with different weight matrices, we end up with eight different attention matrices, and all these matrices are combined

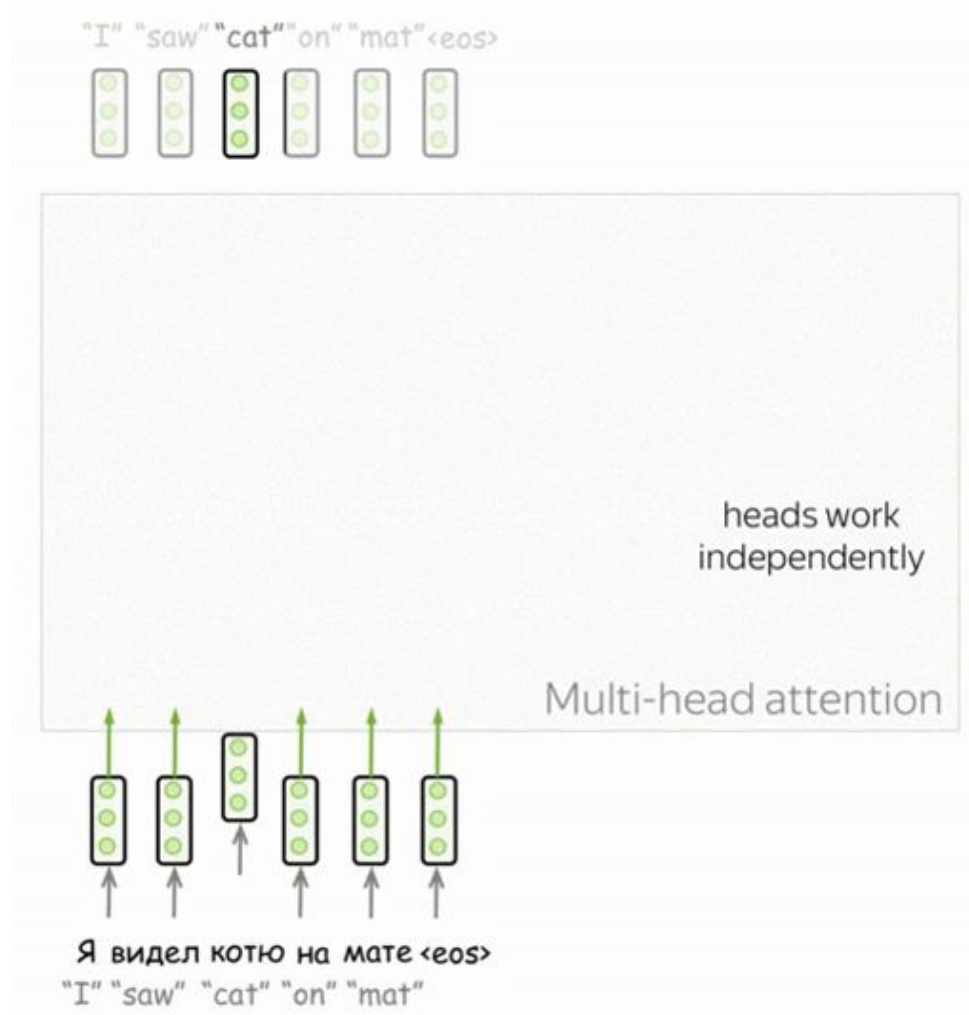
Multi-head attention

idea: understanding the role of a word in a sentence requires understanding how it is related to different parts of the sentence

- e.g. in some languages, subjects define verb inflection (e.g., gender agreement), verbs define the case of their objects...

→ each word is part of many relations

→ several attention results concatenated

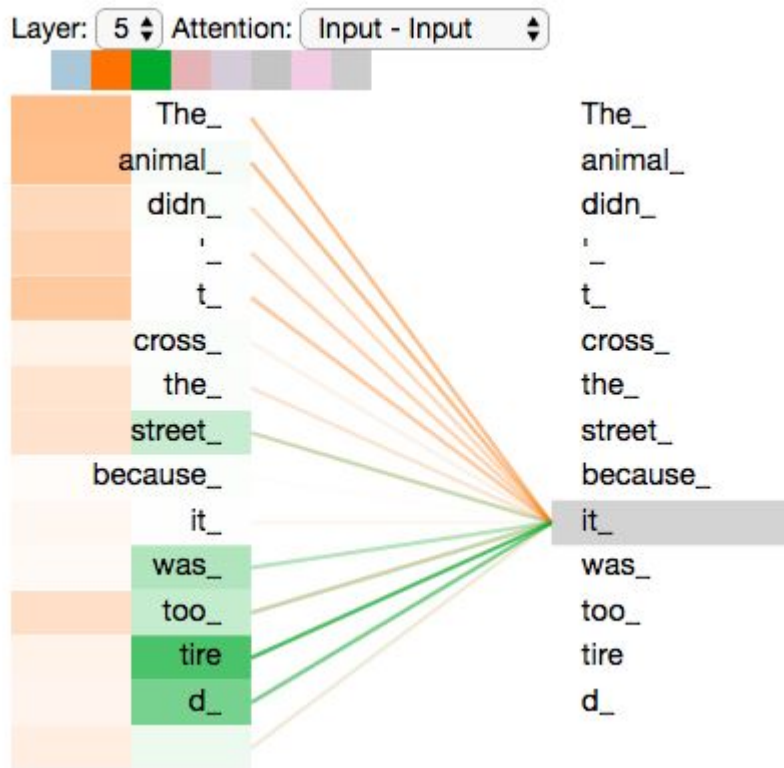


Multi-head attention

- orange head: focuses on “animal”
- green head: focuses on “tired”

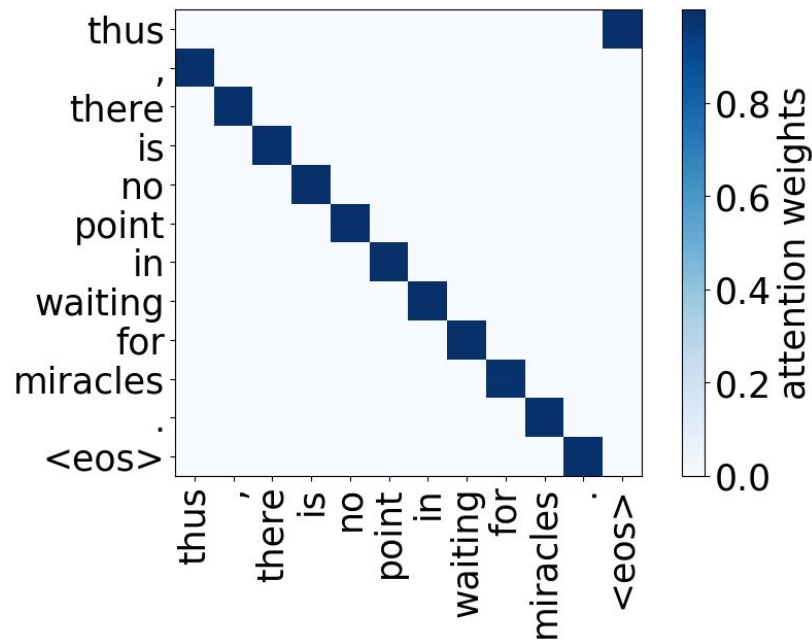
[\[Voita et al 2019\]](#): some heads play interpretable roles

- positional: attend to neighbors
- syntactic: learn major syntactic relations
- rare tokens: attend to the least frequent tokens

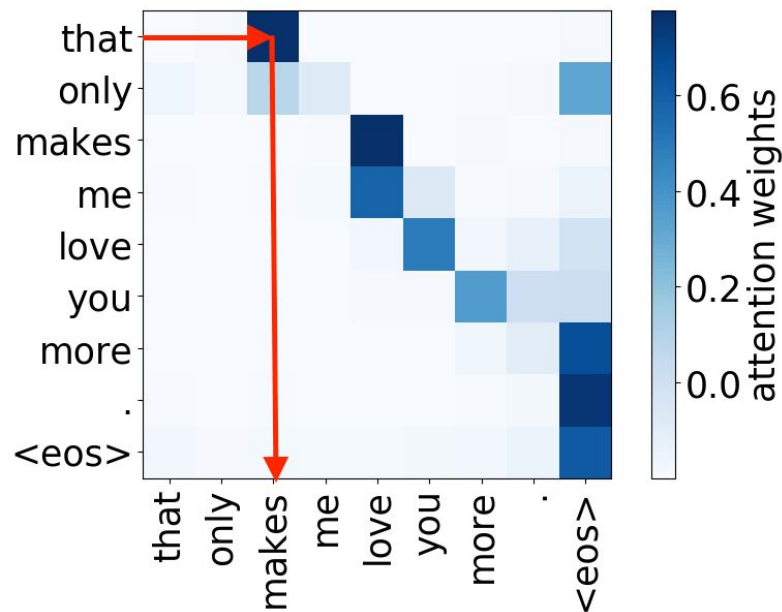


Multi-heads

Positional heads

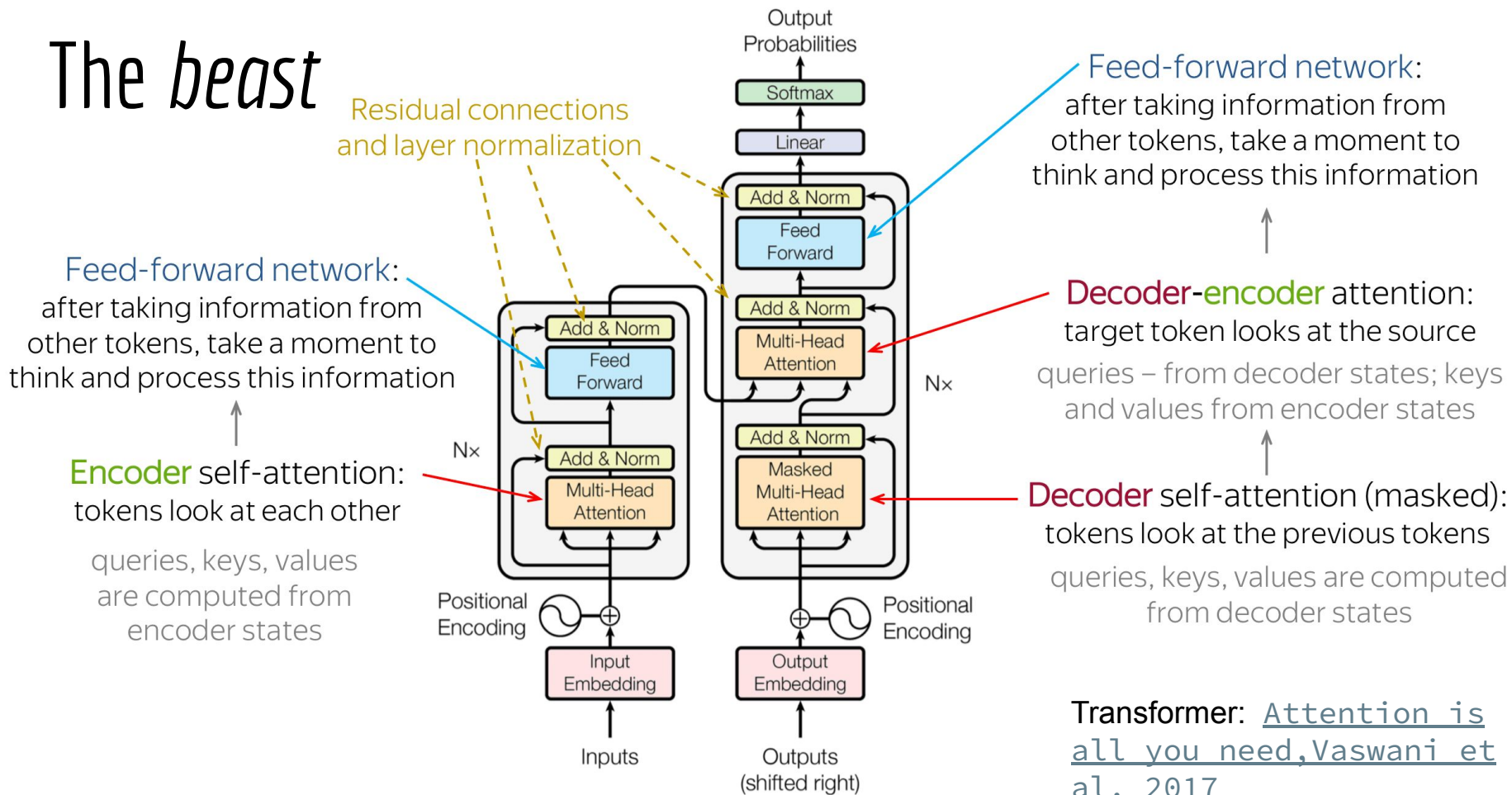


Syntactic heads (subject → verb)

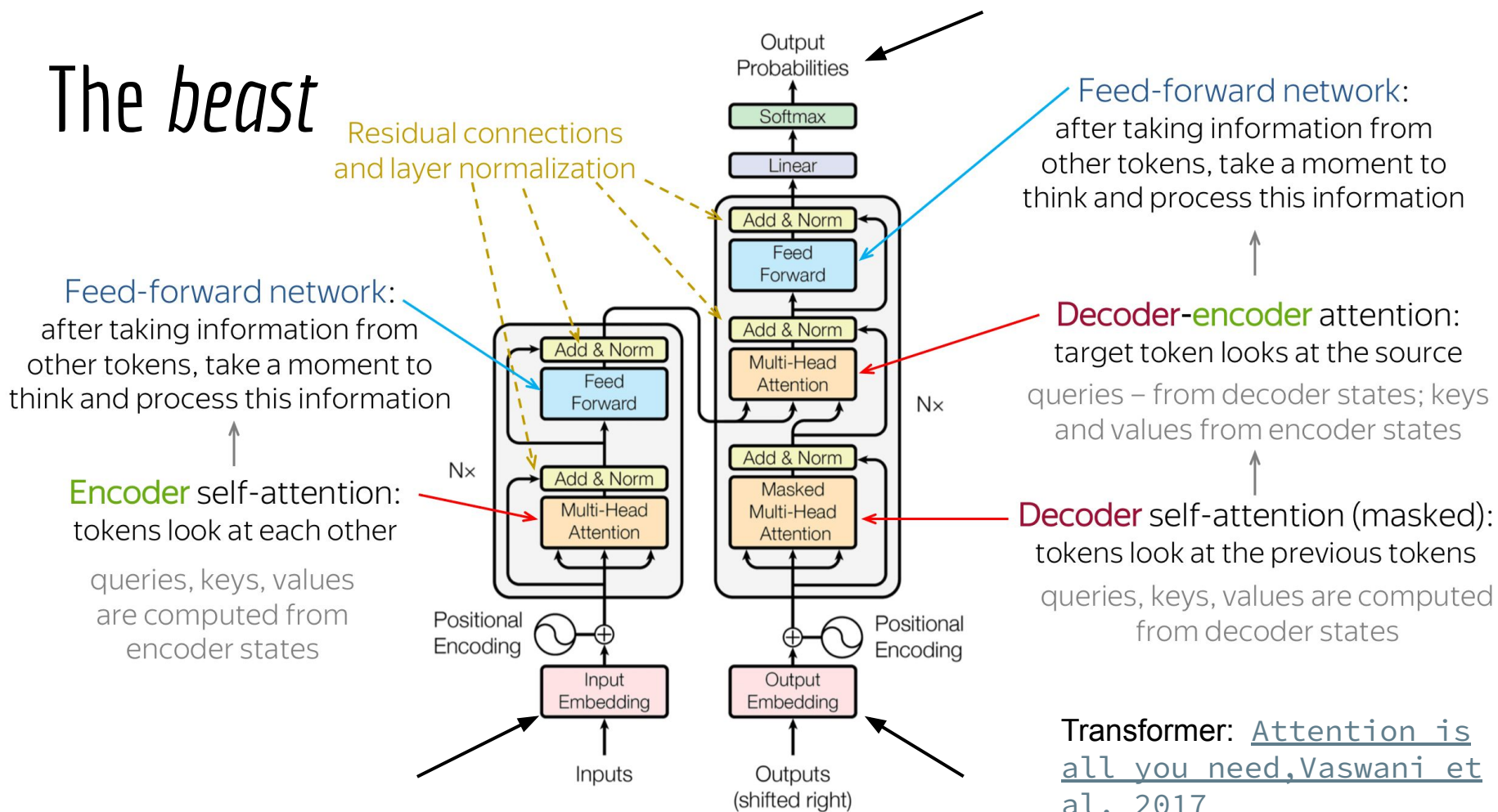


https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

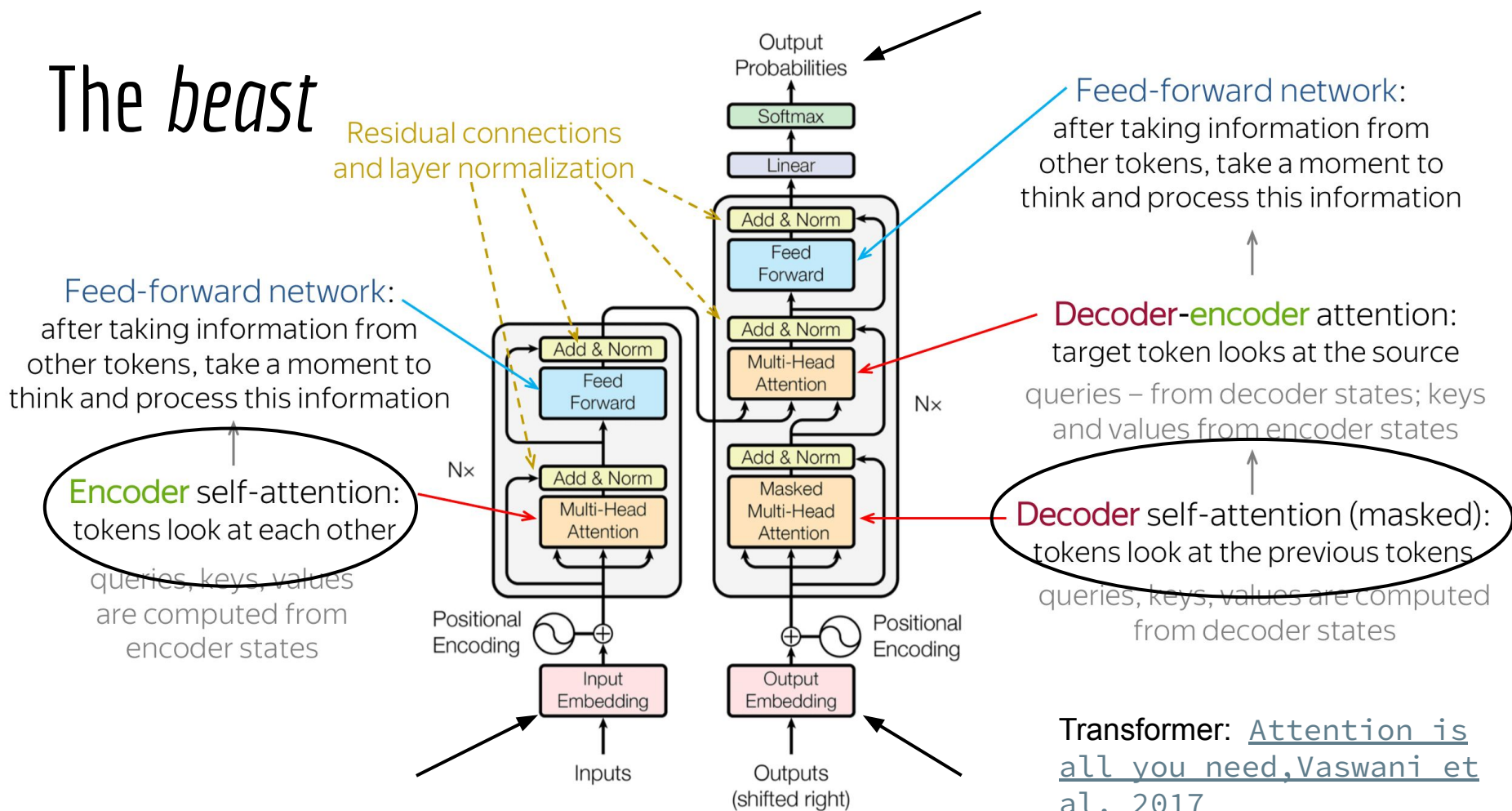
The *beast*



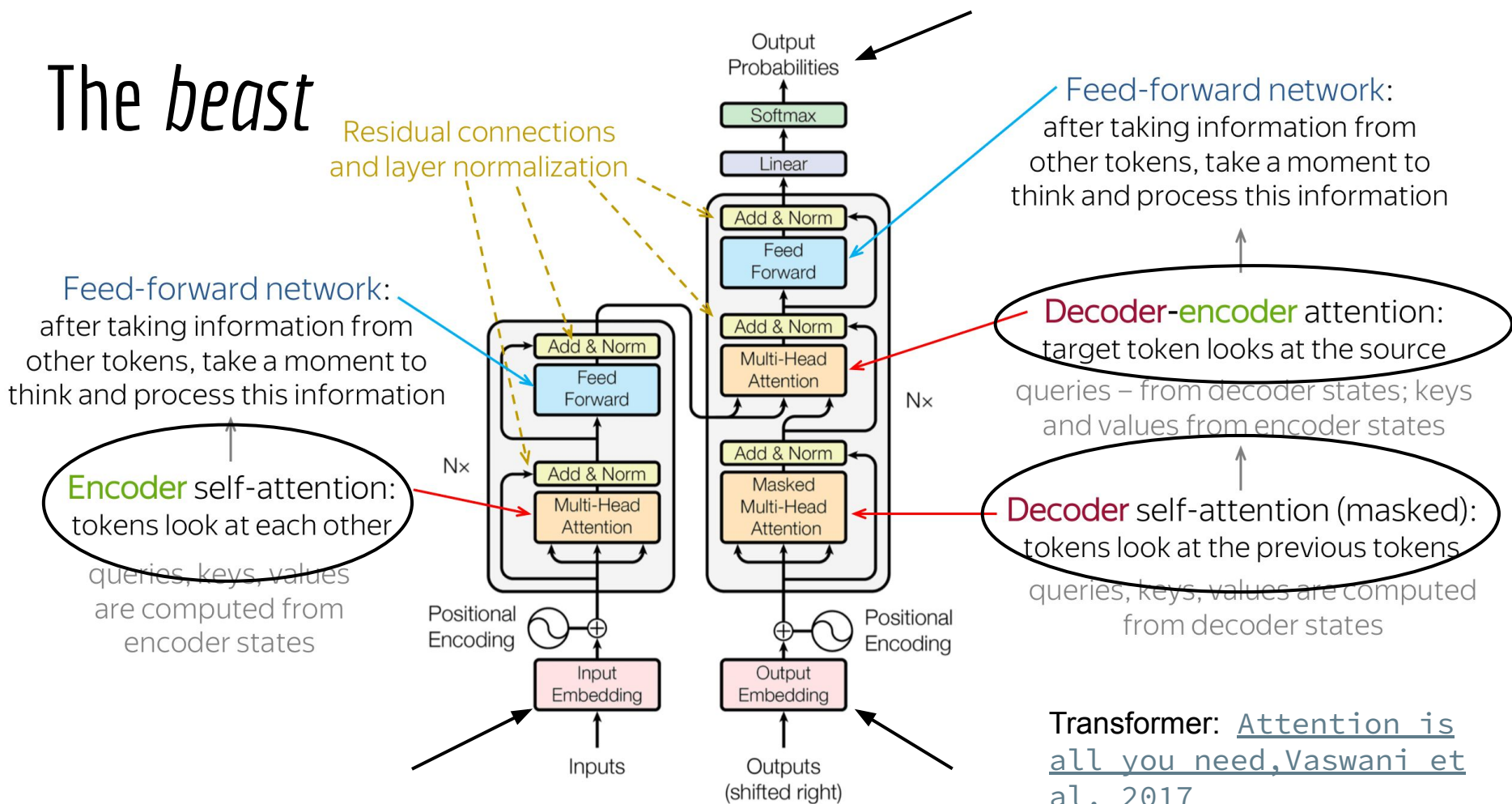
The *beast*



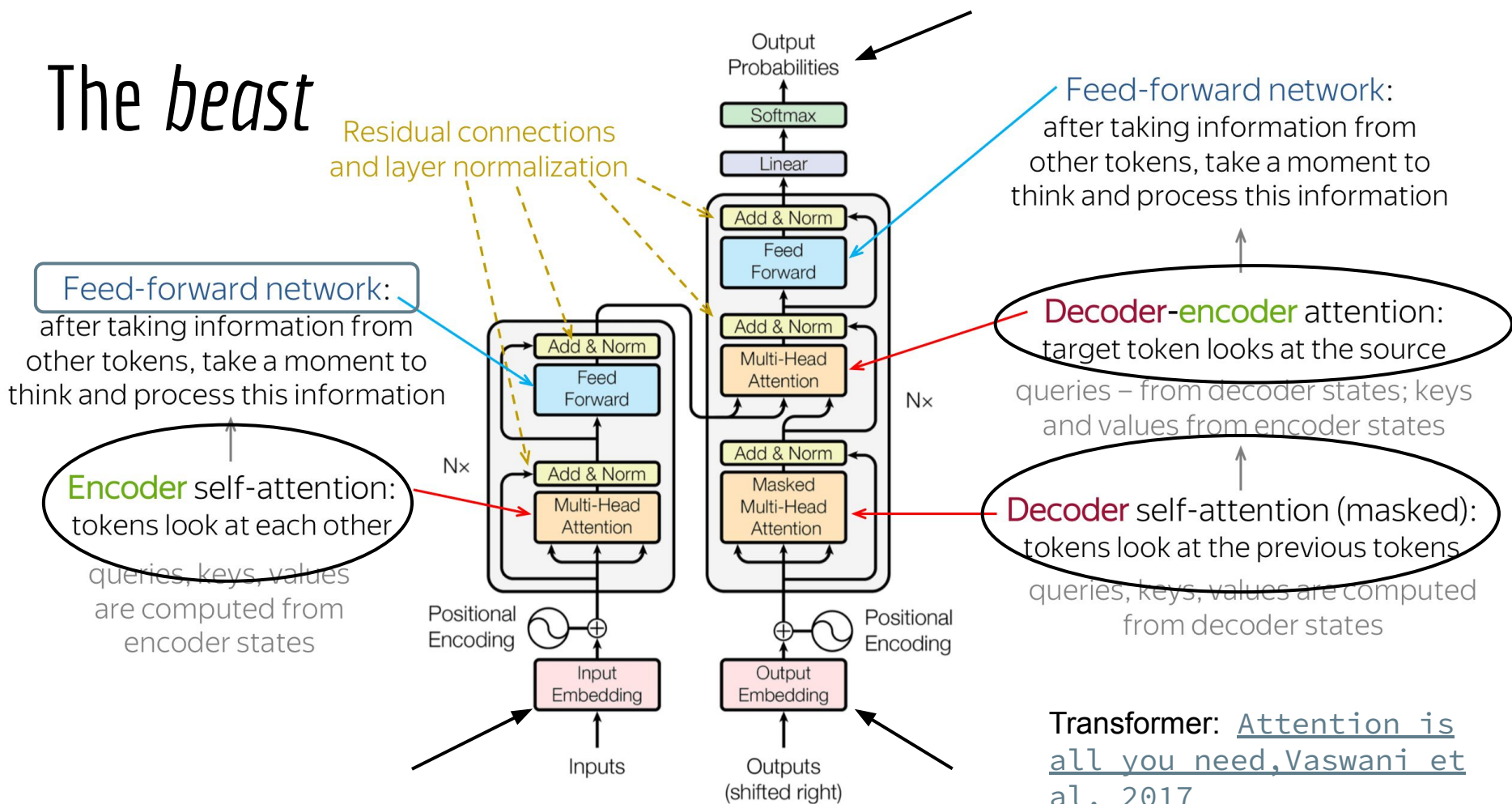
The beast



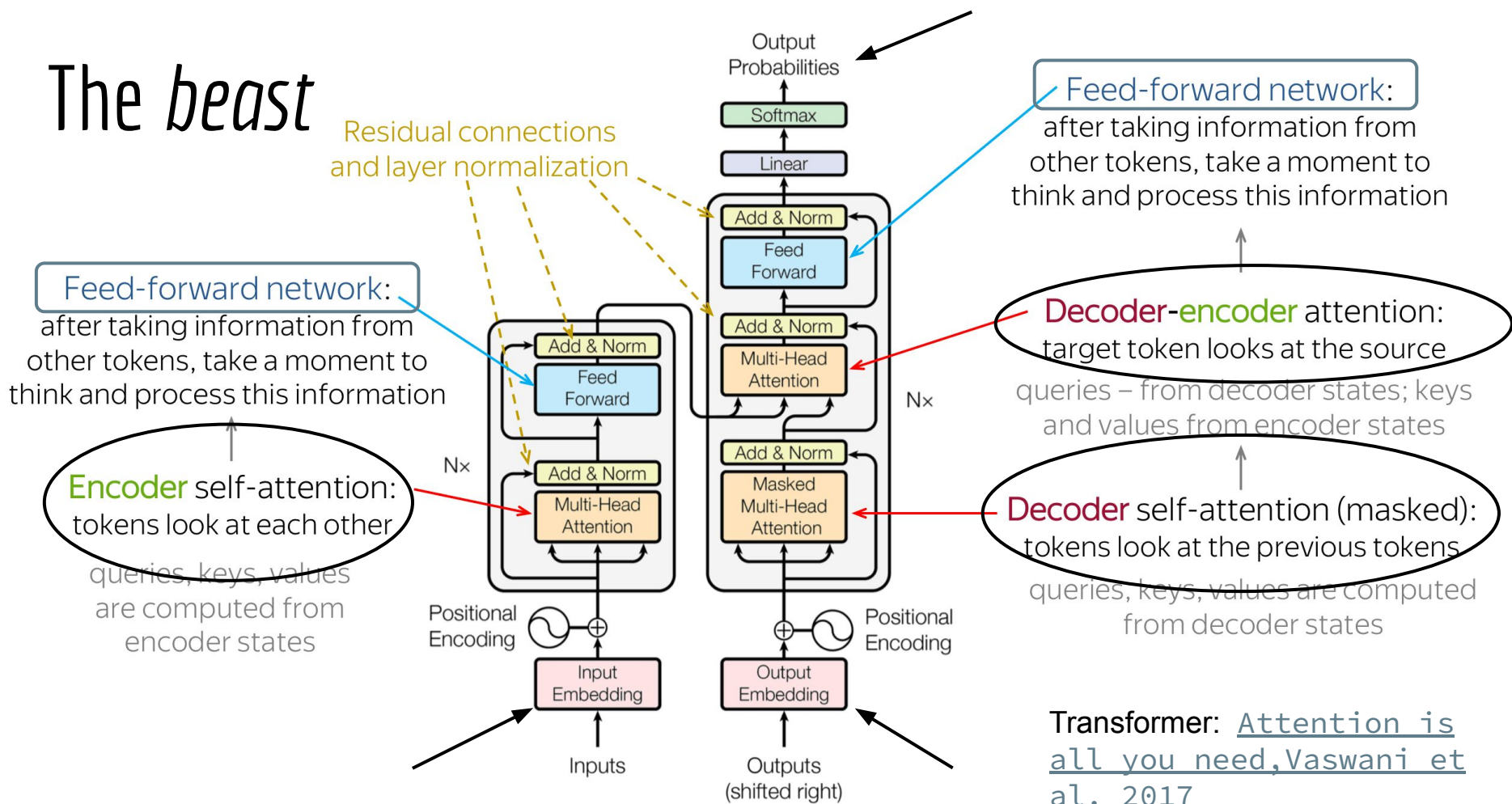
The beast



The beast



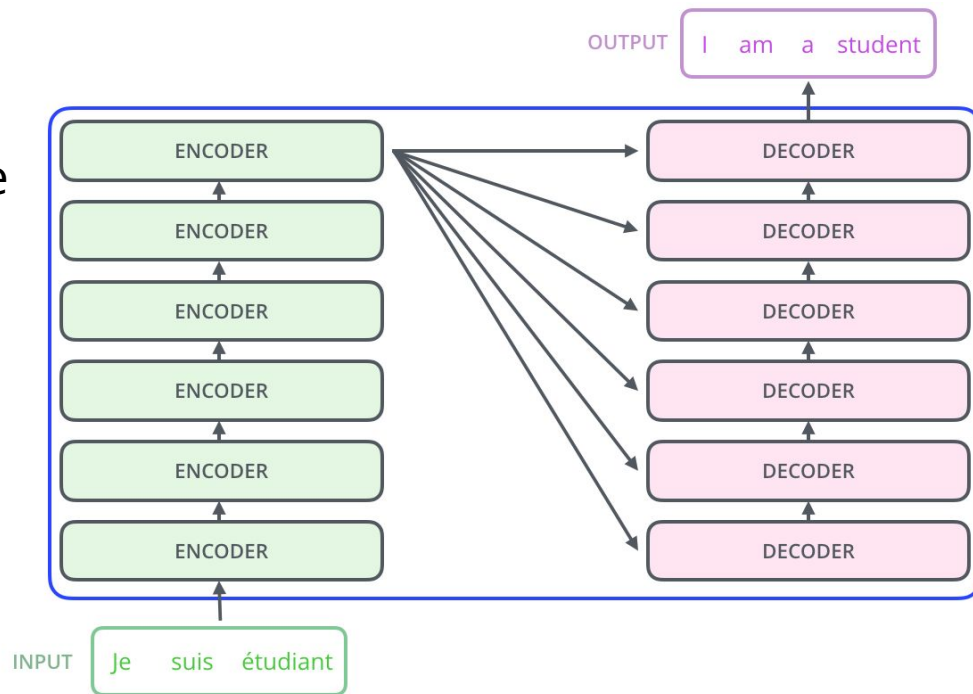
The *beast*



The *beast*

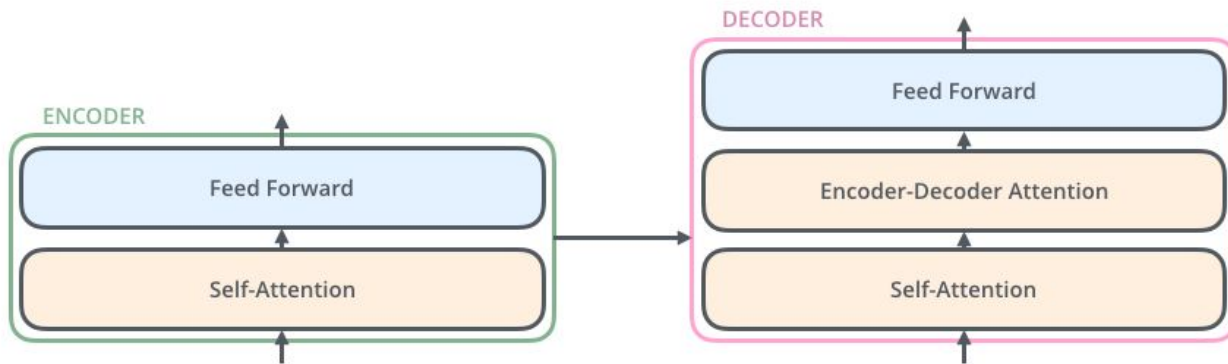
In the paper:

- a stack of 6 encoders (could be any number, do not share weights) and same number of decoders
- each encoders passes its output to the next encoder



The *beast*

- each encoder = self-attention layer + FFNN (2 linear + ReLU)
- each decoder: add attention over the source



Transformers

Many elements in the model:

- self-attention
- multi-head
- non linearities (MLP)
- layer normalizations (improve convergence stability)
- residual connections (ease the learning)
- positional embeddings

Transformers

Many elements in the model:

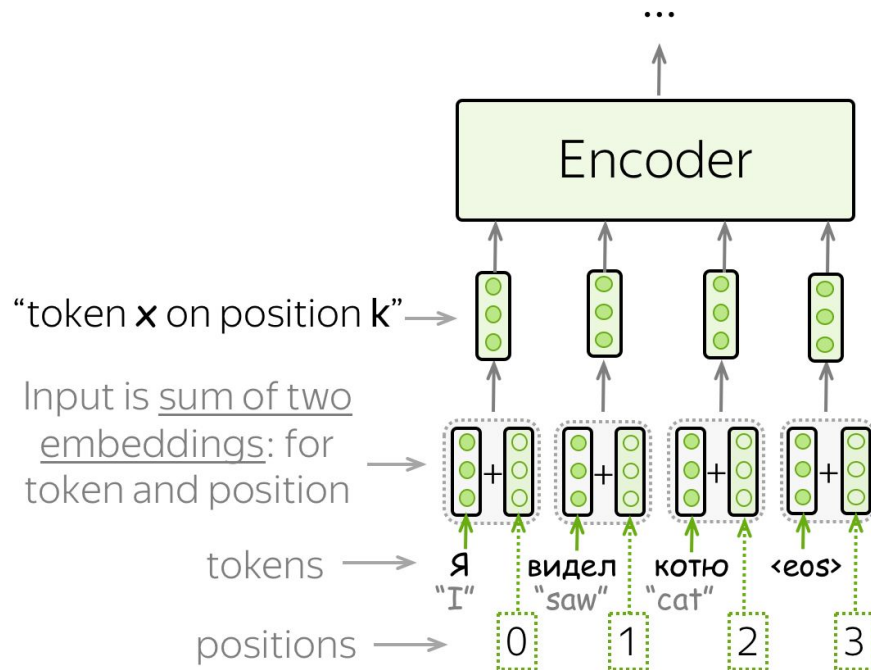
- self-attention
- multi-head
- non linearities (MLP)
- layer normalizations (improve convergence stability)
- residual connections (ease the learning)
- **positional embeddings**

Sequence ordering?

Add a representation of the position:

- fixed representation
- or learned representation

Philipp Dufter, Martin Schmitt, Hinrich Schütze :
Position Information in Transformers : An Overview.

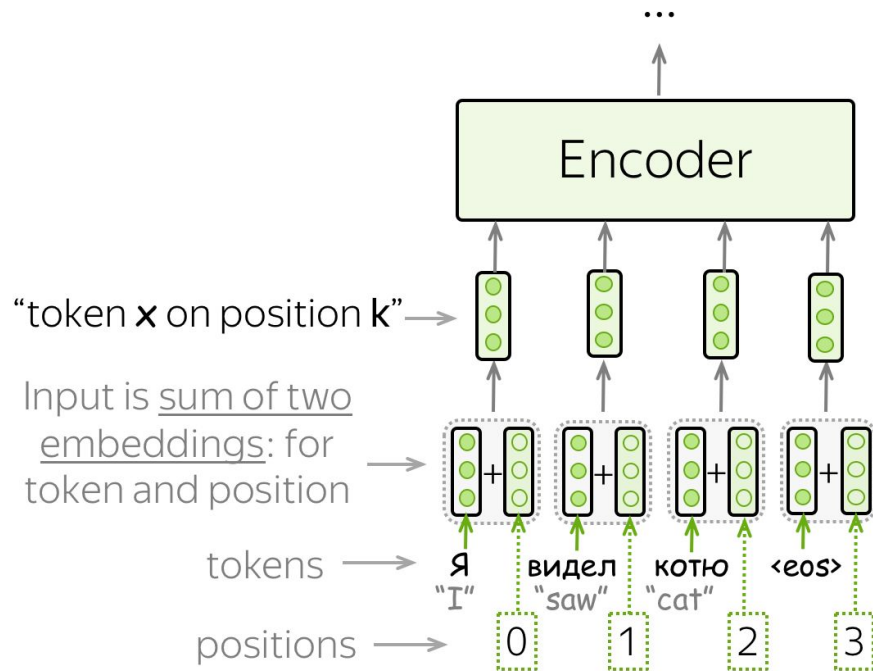


Sequence ordering?

Add a representation of the position:

- **fixed representation**
- or learned representation

Philipp Dufter, Martin Schmitt, Hinrich Schütze :
Position Information in Transformers : An Overview.



Sequence ordering?

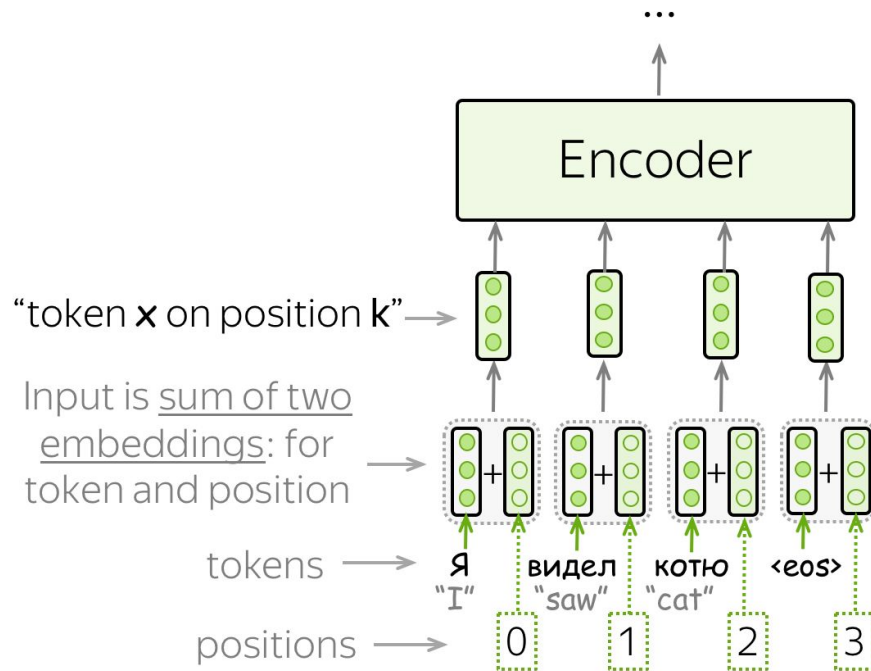
Add a representation of the position:

- ordinal = absolute position
- fixed representation to encode relative position
- learned representation

Philipp Dufter, Martin Schmitt, Hinrich Schütze :
Position Information in Transformers : An Overview.

→ not having info hurts for some tasks, but not all of them: but present in all models

Sinha et al. Masked Language Modeling and the
Distributional Hypothesis : Order Word Matters
Pre-training for Little, EMNLP 2021



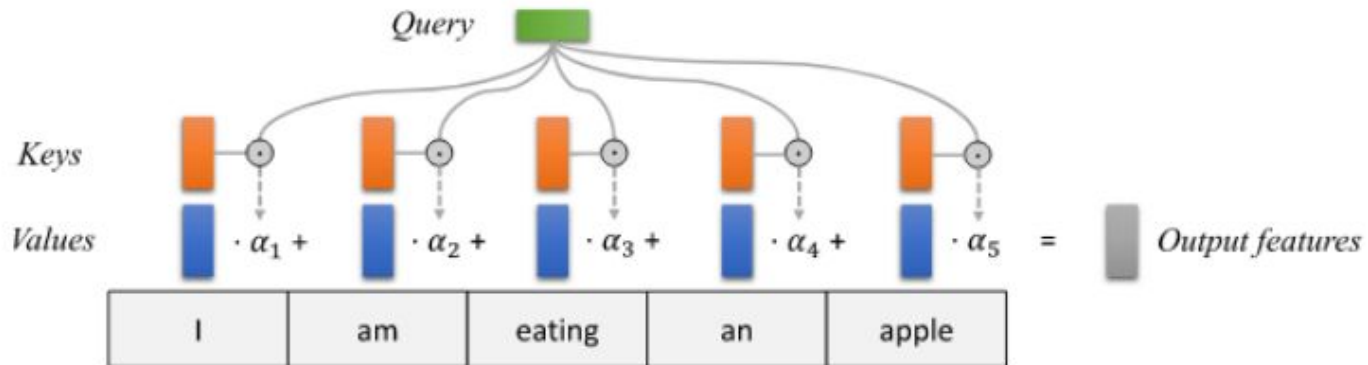
Source

- Very clear explanation (and nice pictures / videos):
https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html
- <https://ledatascientist.com/a-la-decouverte-du-transformer/>
- https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb
- <https://www.analyticsvidhya.com/blog/2019/11/comprehensive-guide-attention-mechanism-deep-learning/>
- <https://towardsdatascience.com/deconstructing-bert-part-2-visualizing-the-inner-workings-of-attention-60a16d86b5c1>
- <https://colab.research.google.com/drive/1hXIQ77A4TYS4y3UthWF-Ci7V7vVUoxmQ?usp=sharing>
-
- <https://www.analyticsvidhya.com/blog/2020/08/build-a-natural-language-generation-nlg-system-using-pytorch/>
- <https://www.kaggle.com/ab971631/beginners-guide-to-text-generation-pytorch/notebook>

More general view on attention

General idea:

- attention is a *query* on the input
- that we align with a *key*
- to operate over an input *value*



More general view on attention

General idea:

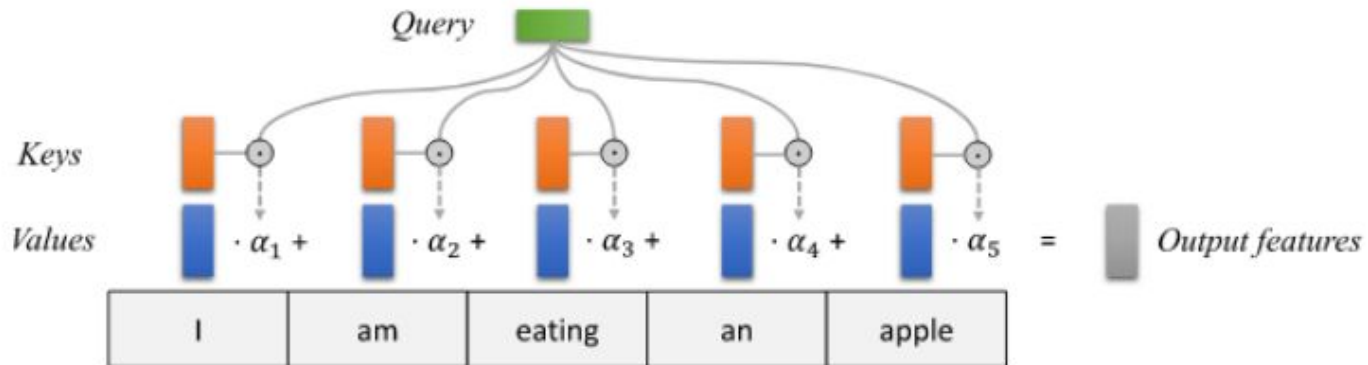
- attention is a *query* on the input
- that we align with a *key* representations \mathbf{c}_i
- to operate over an input *value* \mathbf{c}^j

until now (with RNNs):

← come from the decoding state \mathbf{s}_j

← corresponding to the input

← also \mathbf{c}_i used to weight the context vector



Self-Attention

Self-Attention (or intra-attention): attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.

Idea: decomposing the input into varied functions of x_i wrt the attention computation:

- **query**: interaction with other x_j to compute attention score $x_i, x_j \rightarrow$ some $q_i = W_q x_i$
- **key**: computation of the weights for the output of another x_j viewed as the query \rightarrow some $k_i = W_k x_i$
- **value**: final weighting to compute the output $y_j \rightarrow$ some $v_i = W_v x_i$

$w_{ij} = q_i k_j$: attention score x_i, x_j

output: $y_j = \sum w_{ij} v_j$

Self-Attention

Self-Attention (or intra-attention): attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.

Idea: decomposing the input into varied functions of x_i wrt the attention computation:

- **query**: interaction with other x_j to compute attention score $x_i, x_j \rightarrow$ some $q_i = W_q x_i$
- **key**: computation of the weights for the output of another x_j viewed as the query \rightarrow some $k_i = W_k x_i$
- **value**: final weighting to compute the output $y_j \rightarrow$ some $v_i = W_v x_i$

$w_{ij} = q_i k_j$: attention score x_i, x_j

output: $y_j = \sum w_{ij} v_j$

Self-Attention

Self-Attention (or intra-attention): attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.

Idea: decomposing the input into varied functions of x_i wrt the attention computation:

- **query**: interaction with other x_j to compute attention score $x_i, x_j \rightarrow$ some $q_i = W_q x_i$
- **key**: computation of the weights for the output of another x_j viewed as the query \rightarrow some $k_i = W_k x_i$
- **value**: final weighting to compute the output $y_j \rightarrow$ some $v_i = W_v x_i$

$w_{ij} = q_i k_j$: attention score x_i, x_j + normalization \Rightarrow attention weights

output: $y_j = \sum w_{ij} v_j$

Self-Attention

Self-Attention (or intra-attention): attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.

Idea: decomposing the input into varied functions of x_i wrt the attention computation:

- **query**: interaction with other x_j to compute attention score $x_i, x_j \rightarrow$ some $q_i = W_q x_i$
- **key**: computation of the weights for the output of another x_j viewed as the query \rightarrow some $k_i = W_k x_i$
- **value**: final weighting to compute the output $y_j \rightarrow$ some $v_i = W_v x_i$

$w_{ij} = q_i \cdot k_j$: score d'attention x_i/x_j + normalization \Rightarrow attention weights

output: $y_i = \sum_j w_{ij} v_j$