Neural Methods for NLP

Course 6: additional architectures

Master LiTL --- 2021-2022

chloe.braud@irit.fr

Content

RNNs:

- 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)
- allow to condition on the entire history

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

⟶ **encoder-decoder / sequence to sequence** = conditioned generators: the generated output is conditioned on a complex input

⟶ Based on RNNs and/or **Attention** mechanisms

Walk through code in PyTorch

RNN generators

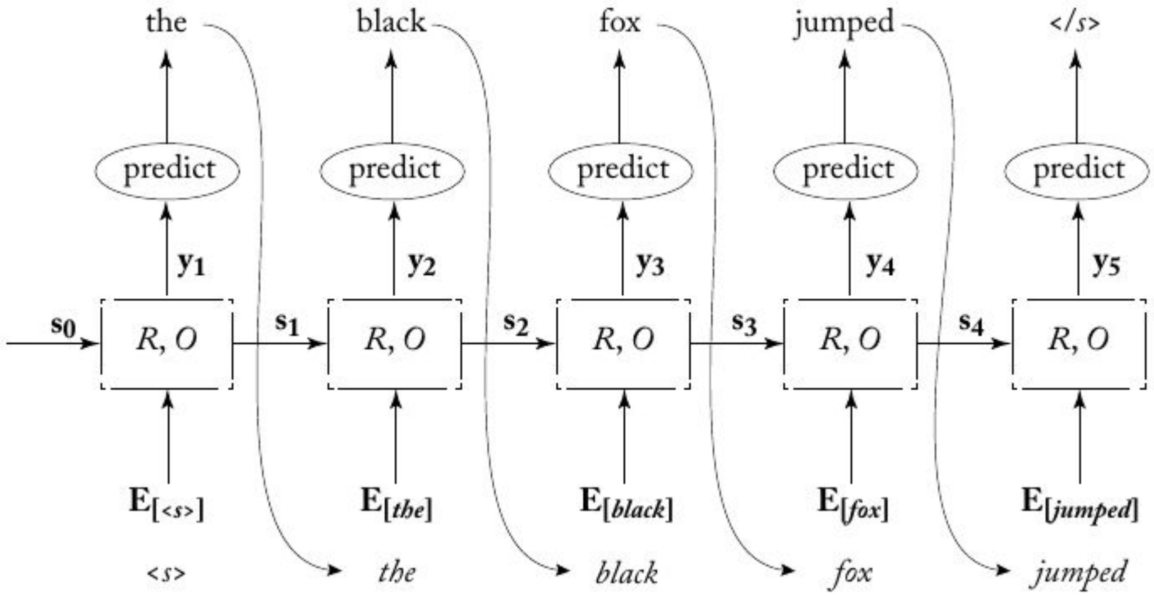
RNN transducer: Producing an output $\mathbf{y}_i$ for each input

⟶ 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 </s>



RNN Generator

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

- ability to condition on long histories
- the produced text resemble 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

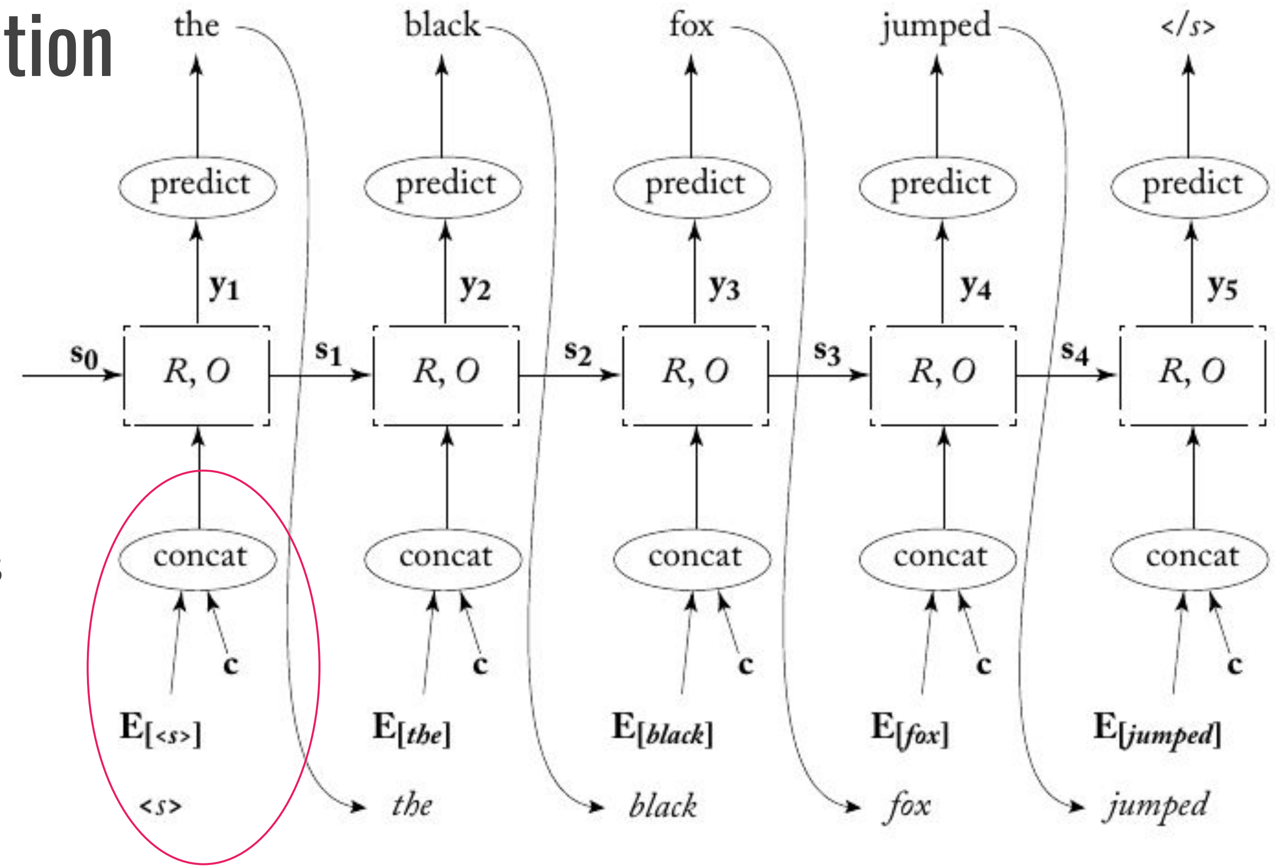
Real power of RNN transducer: *Conditioned generation framework*

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

Conditioned generation

at each stage:

- the context vector **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 **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

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

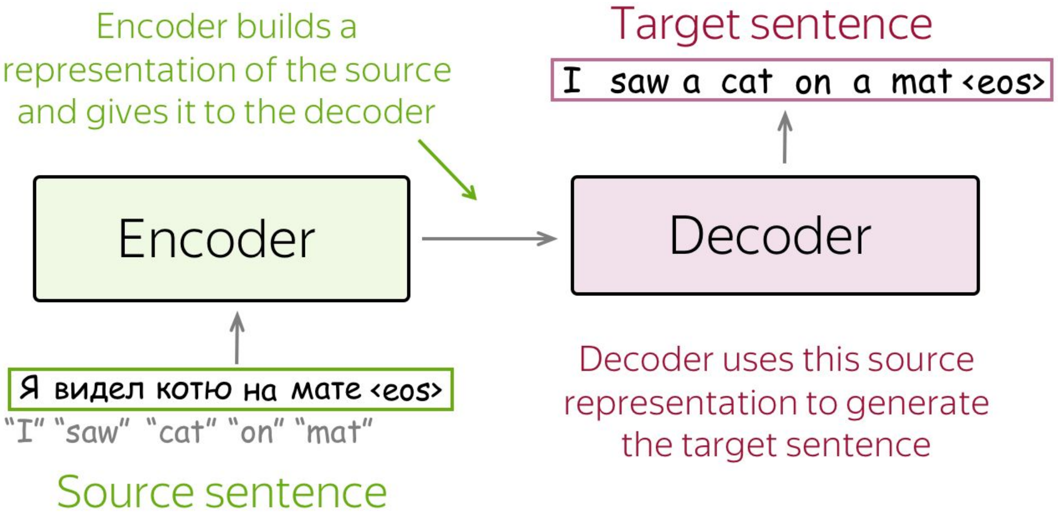
→ **another popular approach: c is itself a sequence of words**

Typical example: Machine translation

e.g. : Machine translation

- encoding the input in source language = produce a representation
- deocer: 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

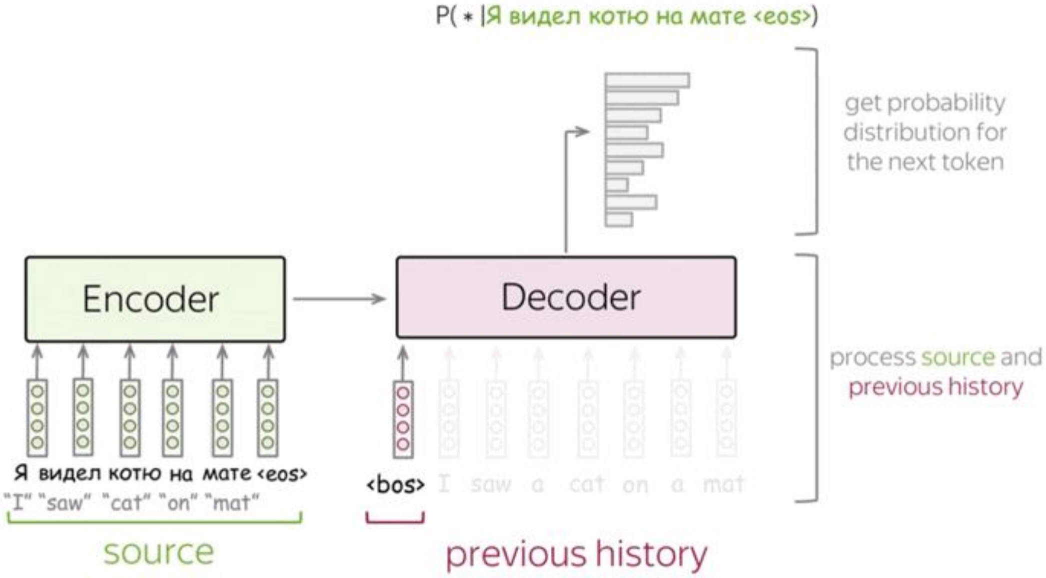
Encoder-Decoder or seq2seq

Sequence to sequence (seq2seq) or encoder-decoder framework [Cho et al, 2014; Sutskever et al 2014] $\longrightarrow$ **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 input

General idea



pipeline:

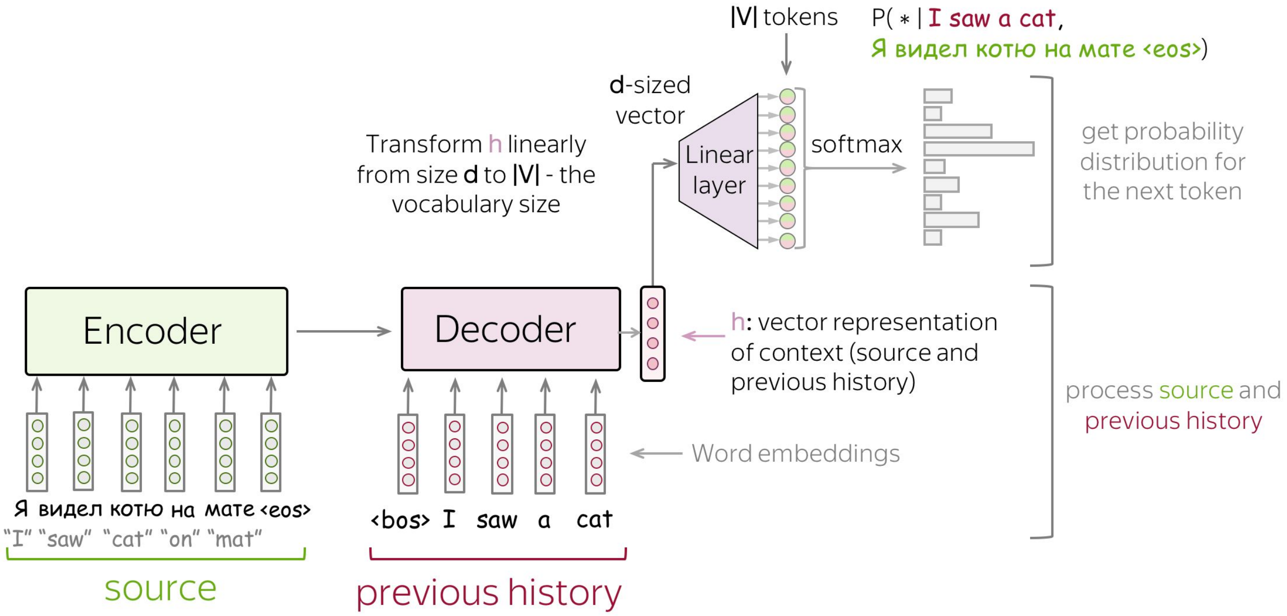
- feed source and previously generated target words into a network;
- get vector representation of context (both source and previous target) from the networks decoder;
- from this vector representation, predict a probability distribution for the next token.

Output layer

classification part:

- vector representation of dimension **d**
- we need a vector of size |V|

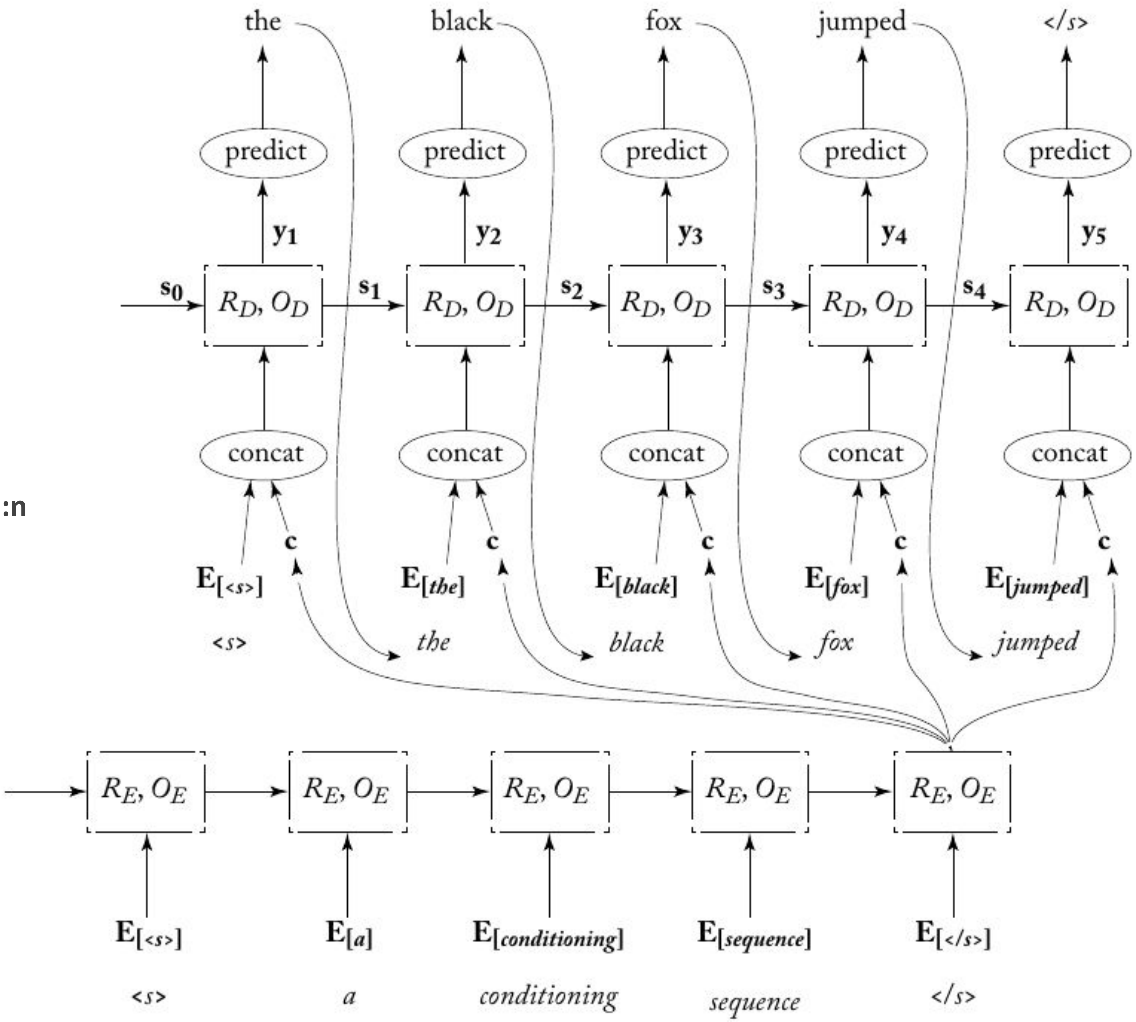
→ linear layer to perform the transformation (then softmax)



Encoder-decoder

Simplest architecture: 2 RNNs

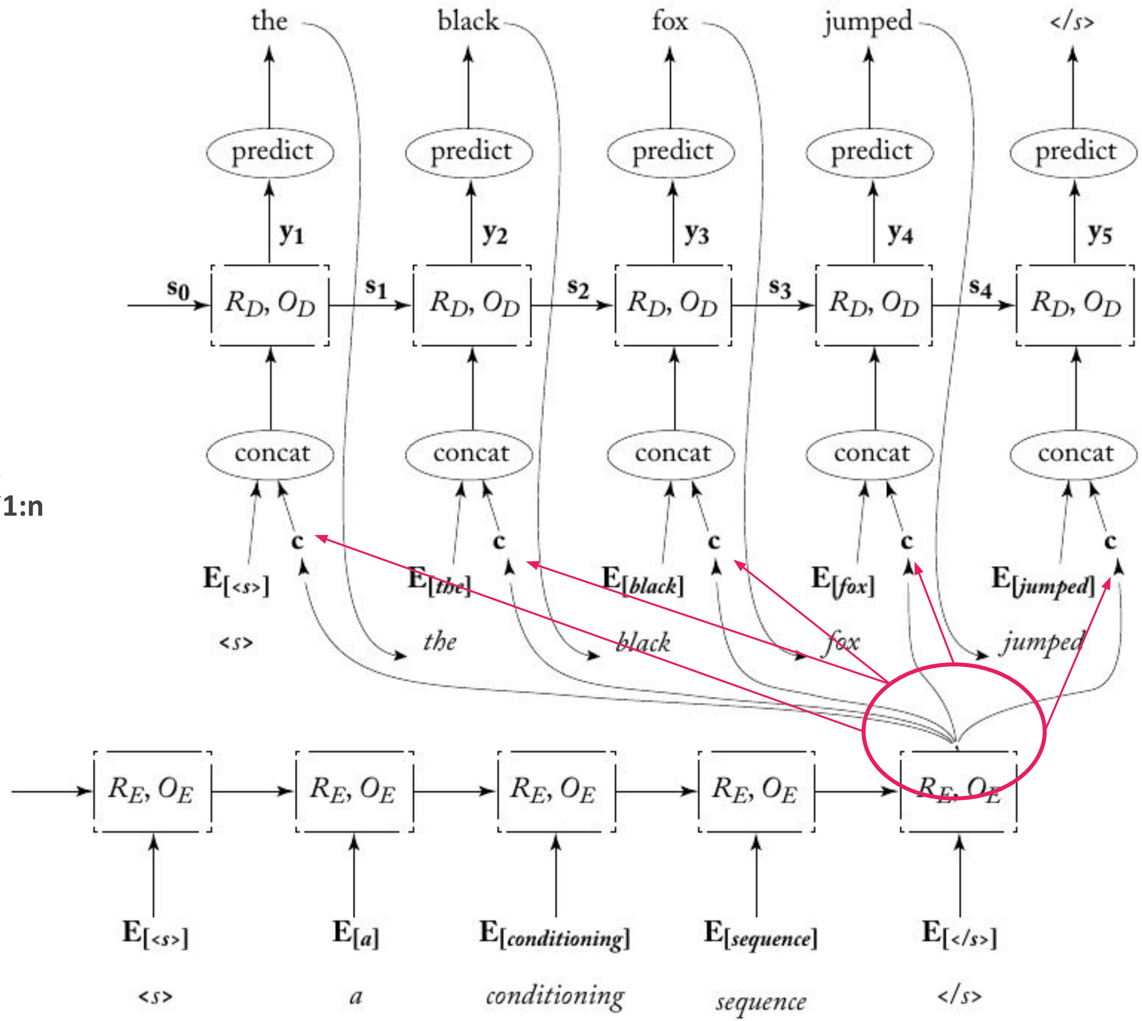
- *encoding* the source sentence $\mathbf{x}_{1:n}$ using an RNN
- using another RNN (*decoder*) to generate the output $\mathbf{t}_{1:m}$



Encoder-decoder

Simplest architecture: 2 RNNs

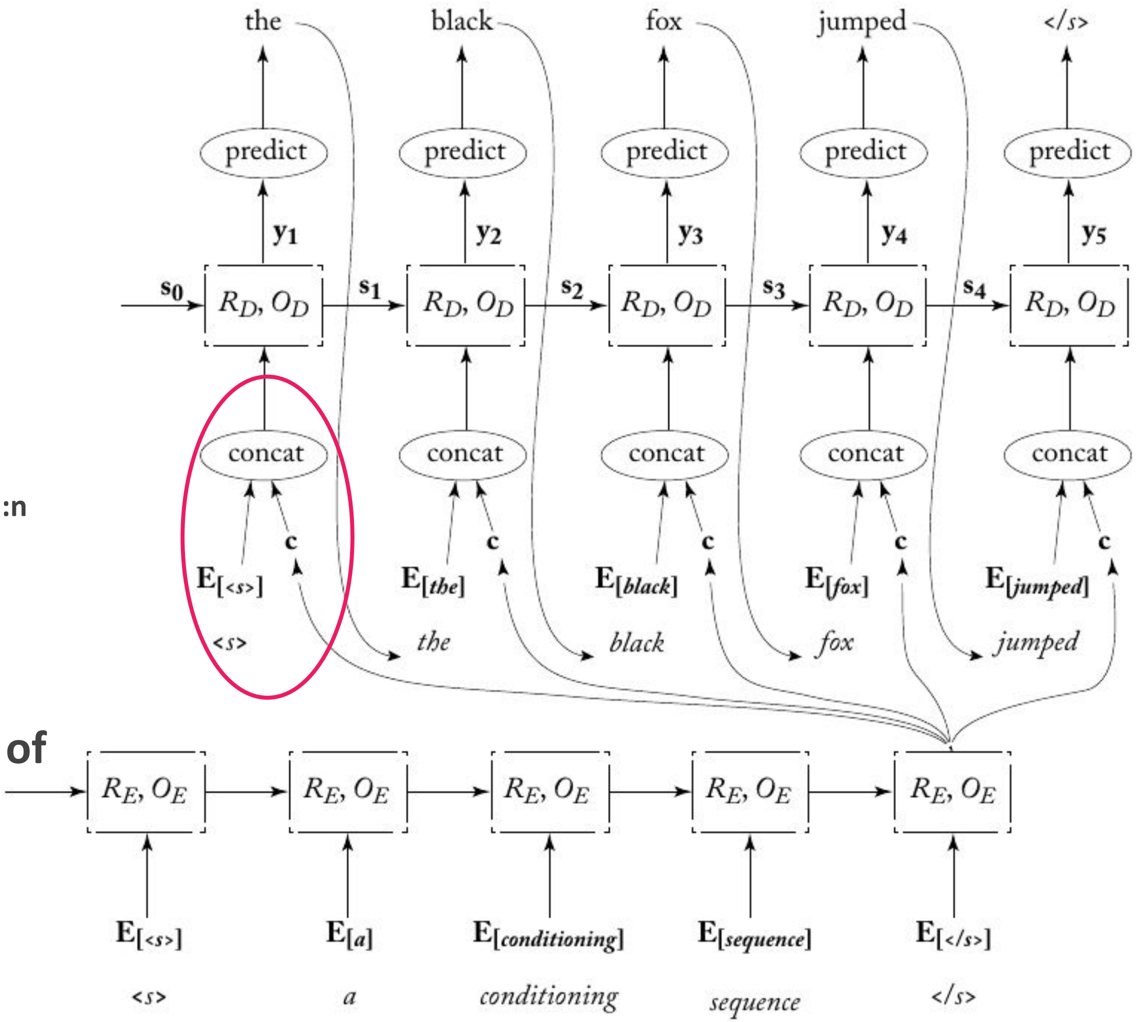
- ***encoding*** **the source sentence $\mathbf{x}_{1:n}$** using an RNN $\longrightarrow$ **last state**
- using another RNN (*decoder*) to generate the output $\mathbf{t}_{1:m}$



Encoder-decoder

Simplest architecture: 2 RNNs

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



Encoder-decoder

- useful to map sequences of size **n** to sequences of length **m**
- encoder = summarizing the source sentence as a vector **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)

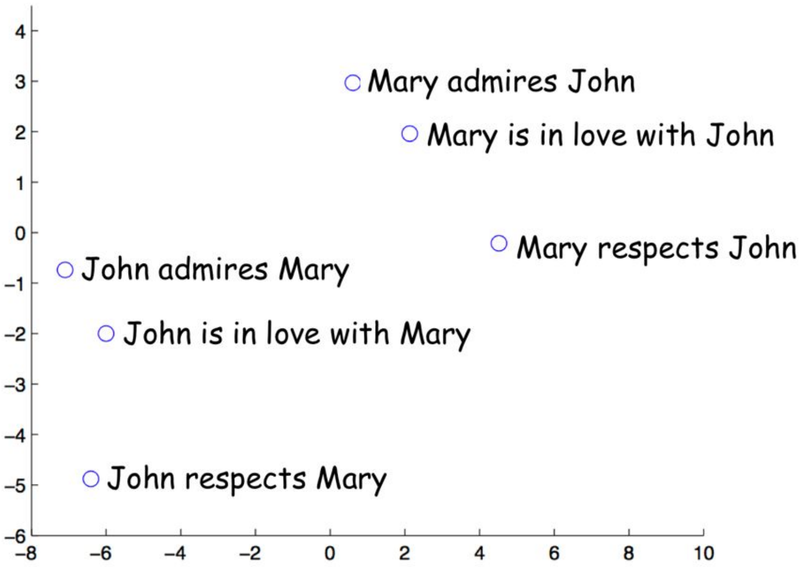
Encoder-decoder

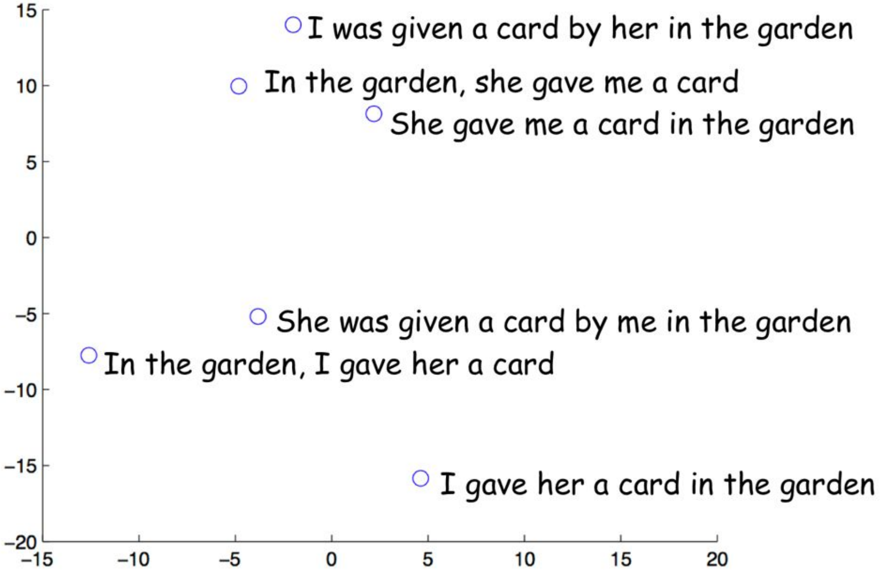
Applications examples:

- 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: sentence compression by deletion [Filippova and Altun, 2013], POS tagging and NER [Gillick et al 2106], 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





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 **c**
- then used by an RNN decoder to perform a task: the information important from the sentence for the task are captured in **c**
- finally: the encoder is used to generate sentence representations **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

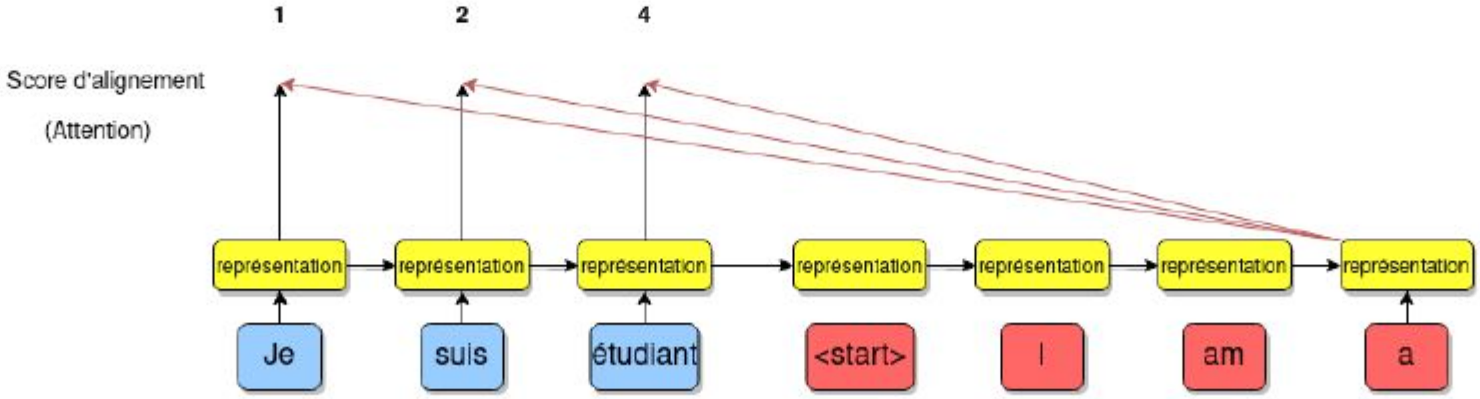
Encoder-decoder: the input sentence is encoded into a single vector

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

⟶ this compression in one representation is suboptimal

⟶ Adding some information from the input: ***attention mechanism*** [Bahdanau et al 2014; Luong et al 2015]

Attention

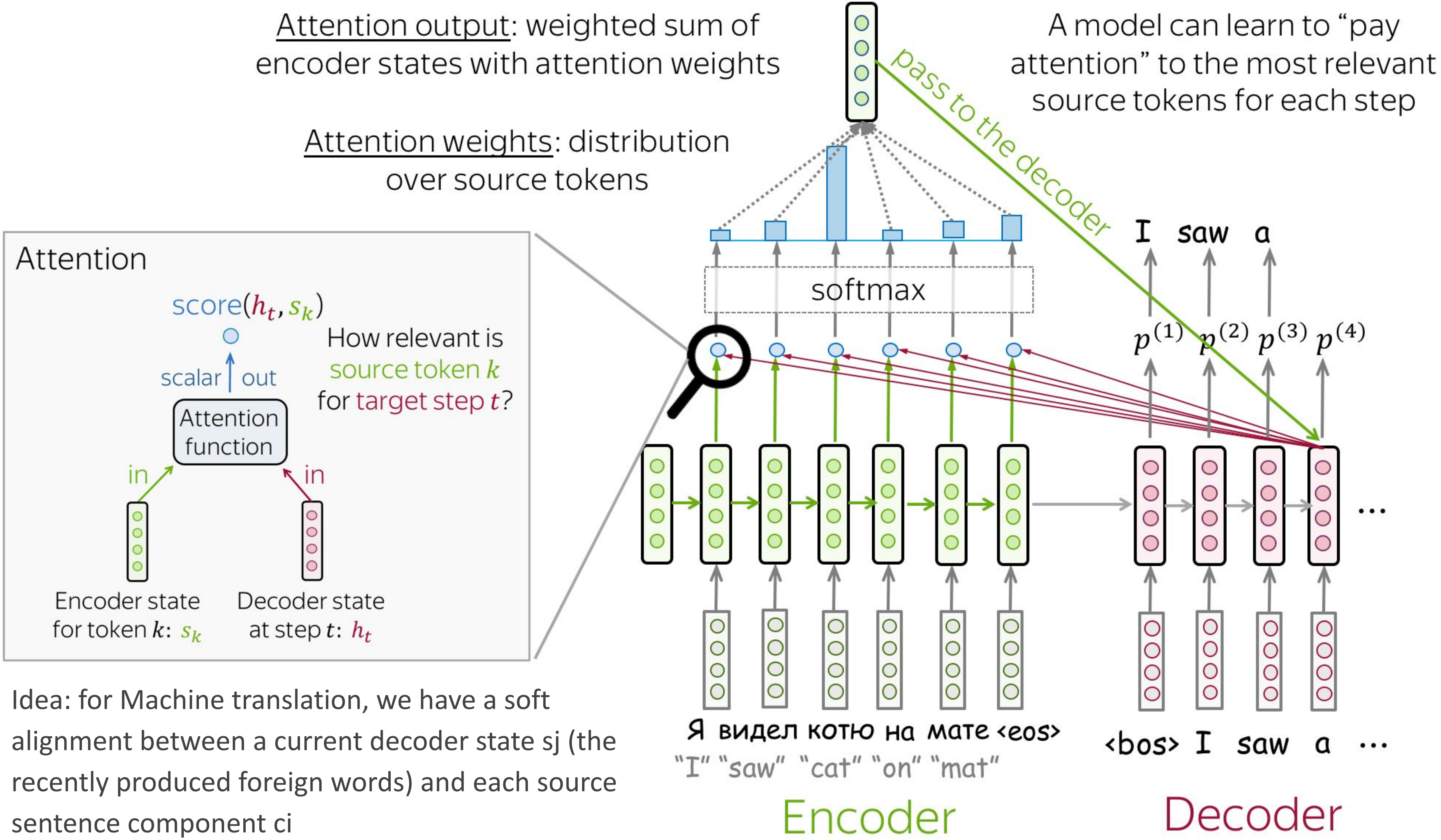


Idea:

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



Idea: for Machine translation, we have a soft alignment between a current decoder state sj (the recently produced foreign words) and each source sentence component ci

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

https://towardsdatascience.com/deconstructing-bert-part-2-visualizing-the-inner-workings-of-attention-60a16d86b5c1

Encoder-decoder with attention

Steps:

- encode an input sequence $\mathbf{x}_{1:n}$ using a RNN → produce **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 → **context vector** $\mathbf{c}^j \leftarrow (\mathbf{c}_{1:n}, \mathbf{t}_{1:j})$
- the context vector is used to generate the next token

$$
\begin{aligned}
p(t_{j+1} = k \mid \hat{t}_{1:j}, \boldsymbol{x}_{1:n}) &= f(O(\boldsymbol{s}_{j+1})) \\
\boldsymbol{s}_{j+1} &= R(\boldsymbol{s}_j, [\hat{\boldsymbol{t}}_j ; \boldsymbol{c}^j]) \\
\boldsymbol{c}^j &= \text{attend}(\boldsymbol{c}_{1:n}, \hat{t}_{1:j}) \\
\hat{t}_j &\sim p(t_j \mid \hat{t}_{1:j-1}, \boldsymbol{x}_{1:n}).
\end{aligned}
$$

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

Encoder-decoder with attention

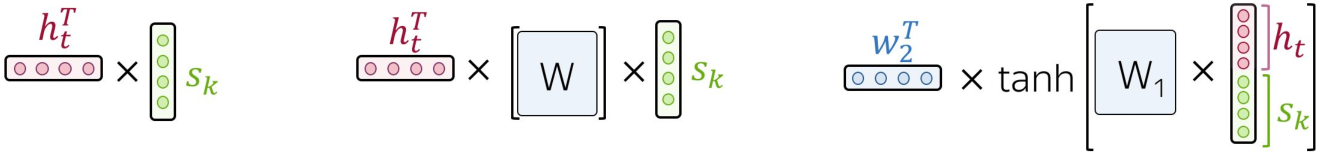
$$\boxed{c^j = \text{attend}(c_{1:n}, \hat{t}_{1:j})}$$

- 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 $\mathbf{c}_{1:n}$

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

1. produce unnormalized weights based on the decoder state at time j, $\mathbf{s}_j$ and the state of the encoder $\mathbf{c}_i$ (using dot product or more complex function)

Dot-product | Bilinear | Multi-Layer Perceptron

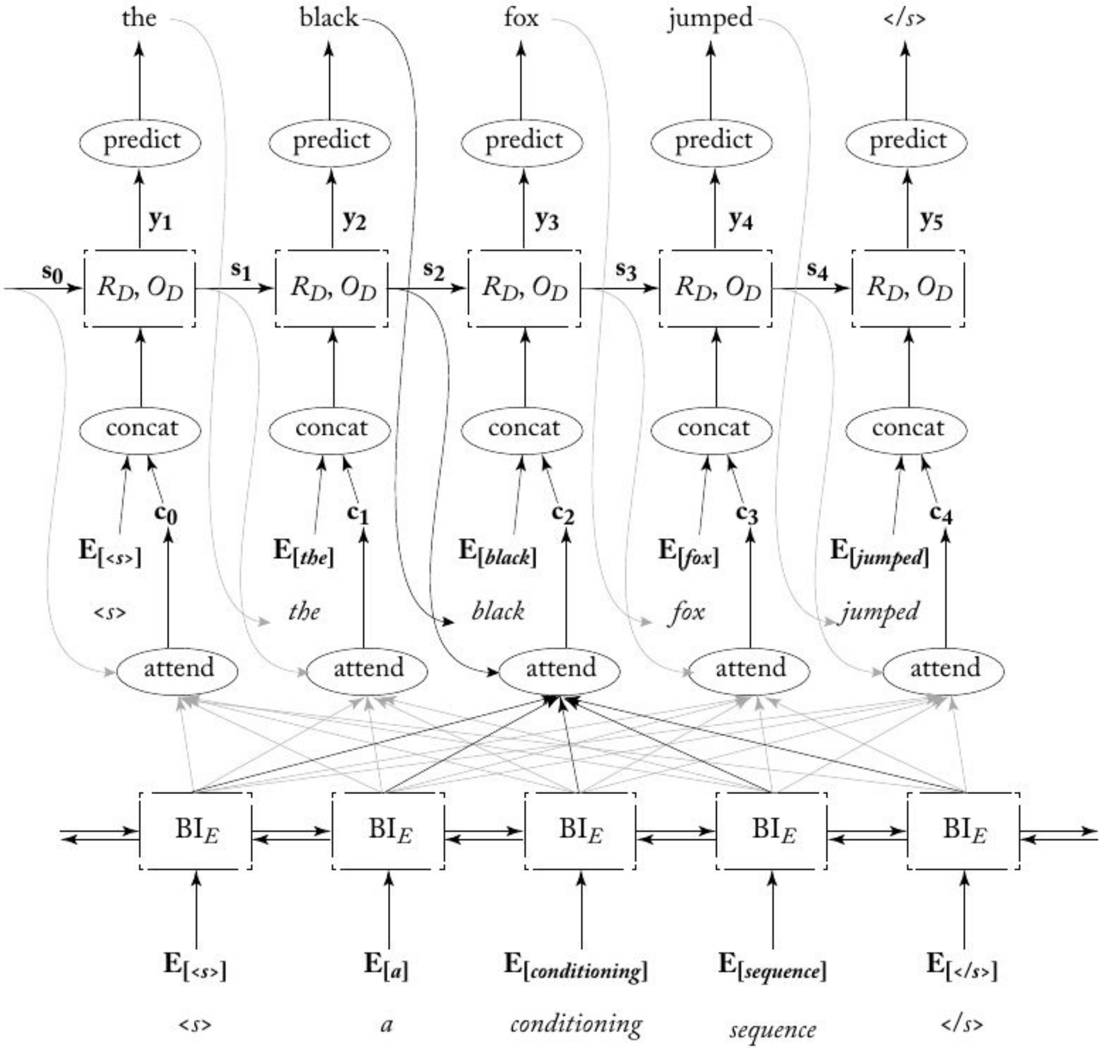


$\text{score}(h_t, s_k) = h_t^T s_k$ | $\text{score}(h_t, s_k) = h_t^T W s_k$ | $\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1[h_t, s_k])$

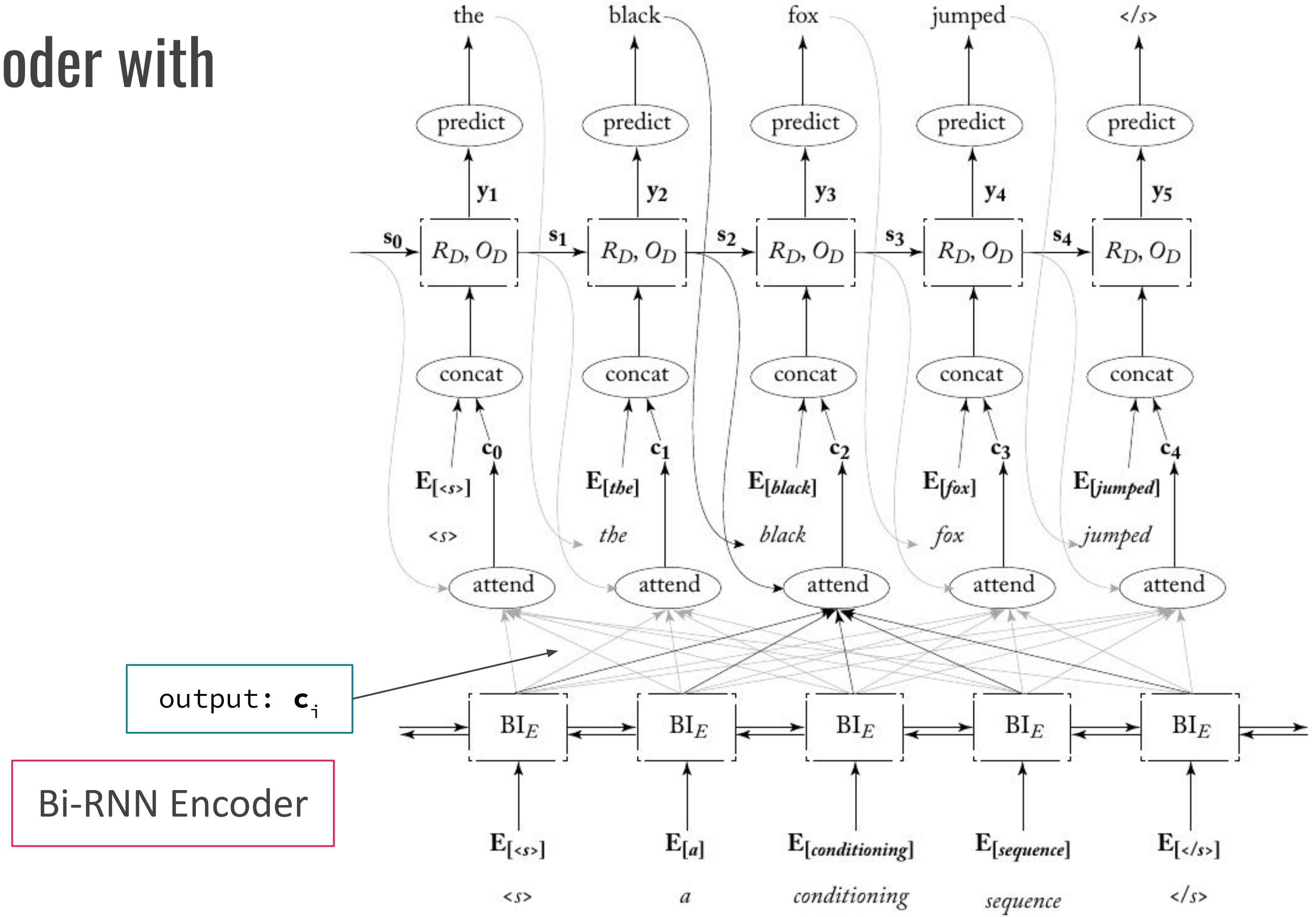
2. normalize the weights into a probability distribution (sum to 1) using softmax
3. the final context vector is

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

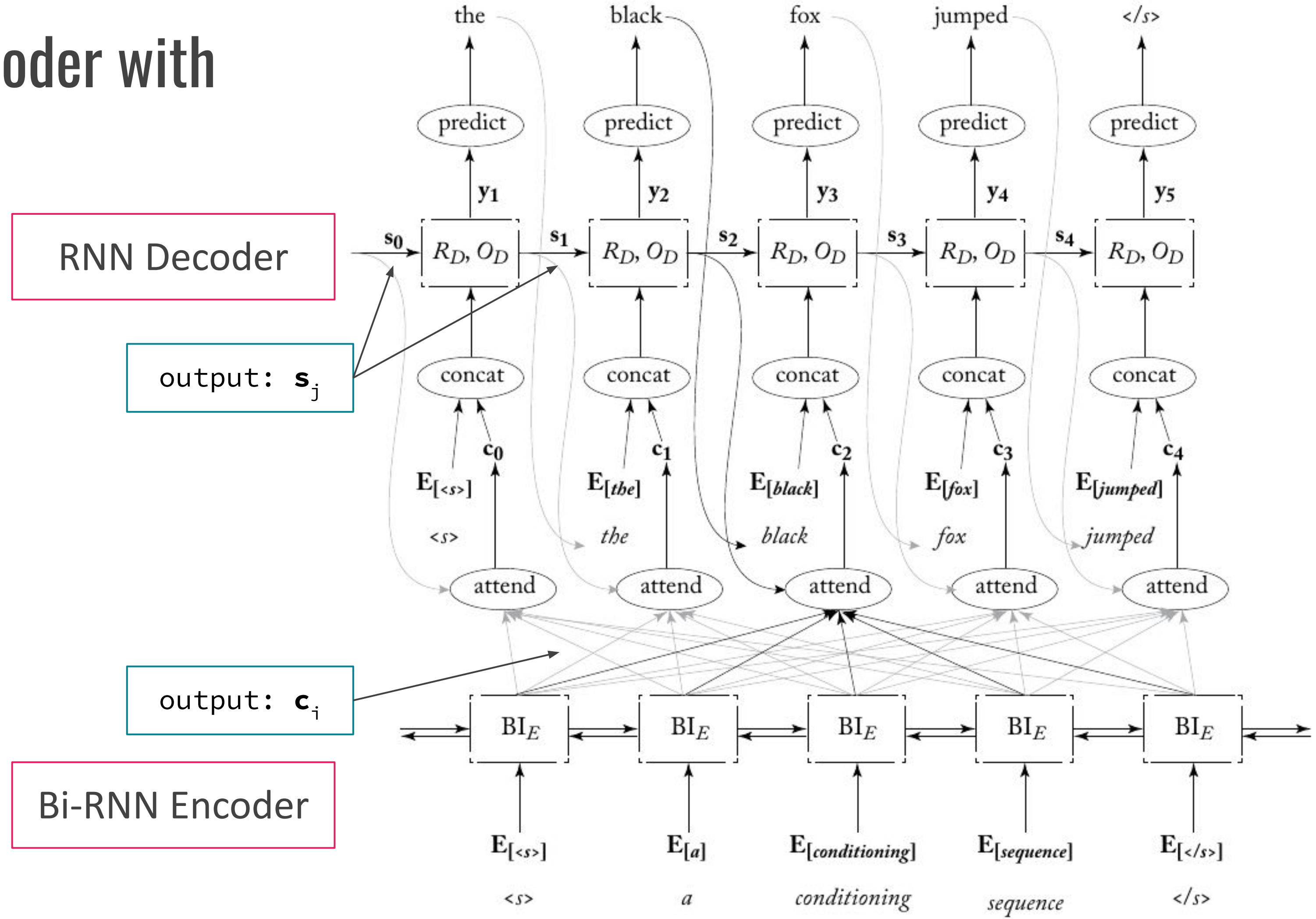
Encoder-decoder with attention



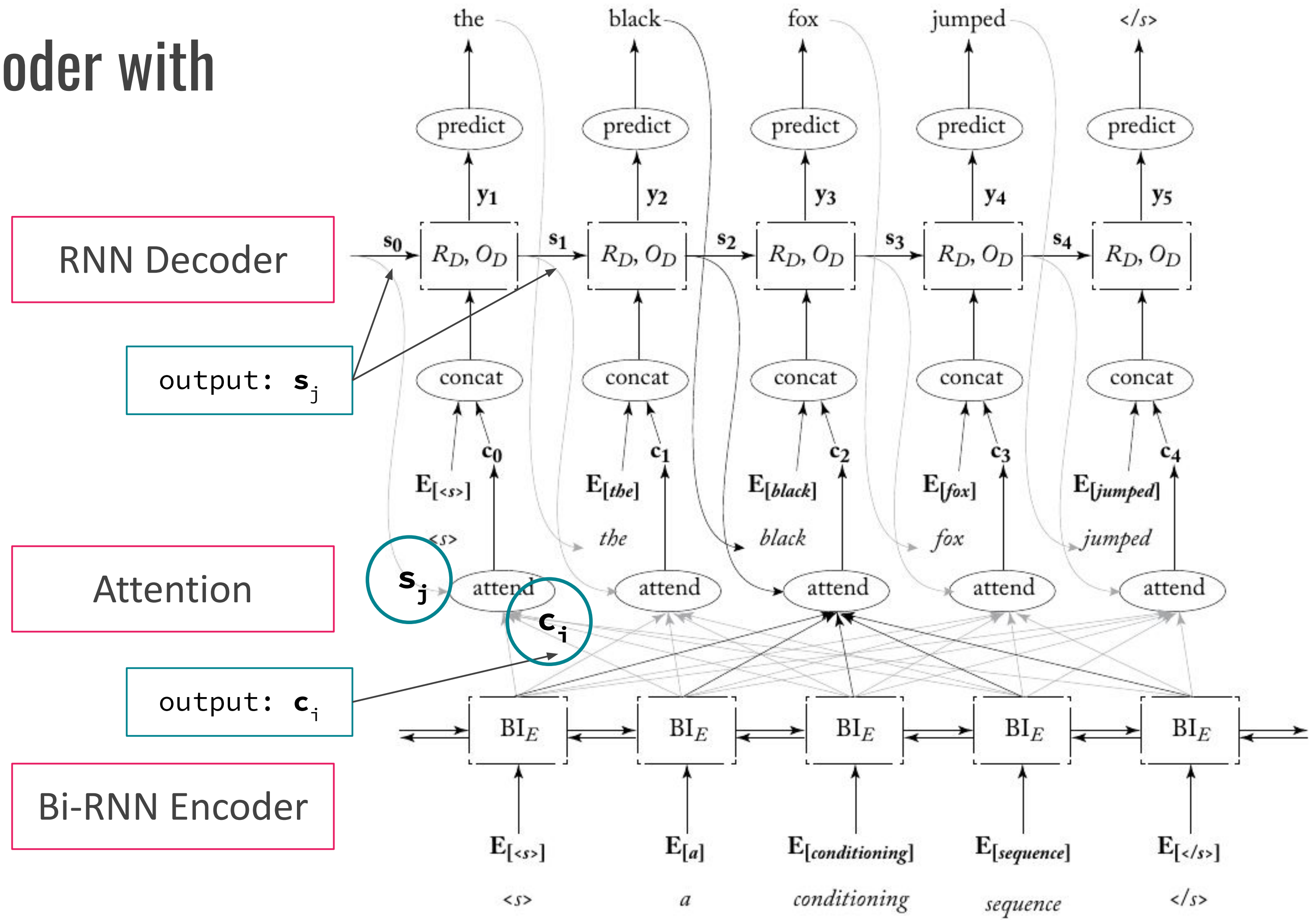
Encoder-decoder with attention



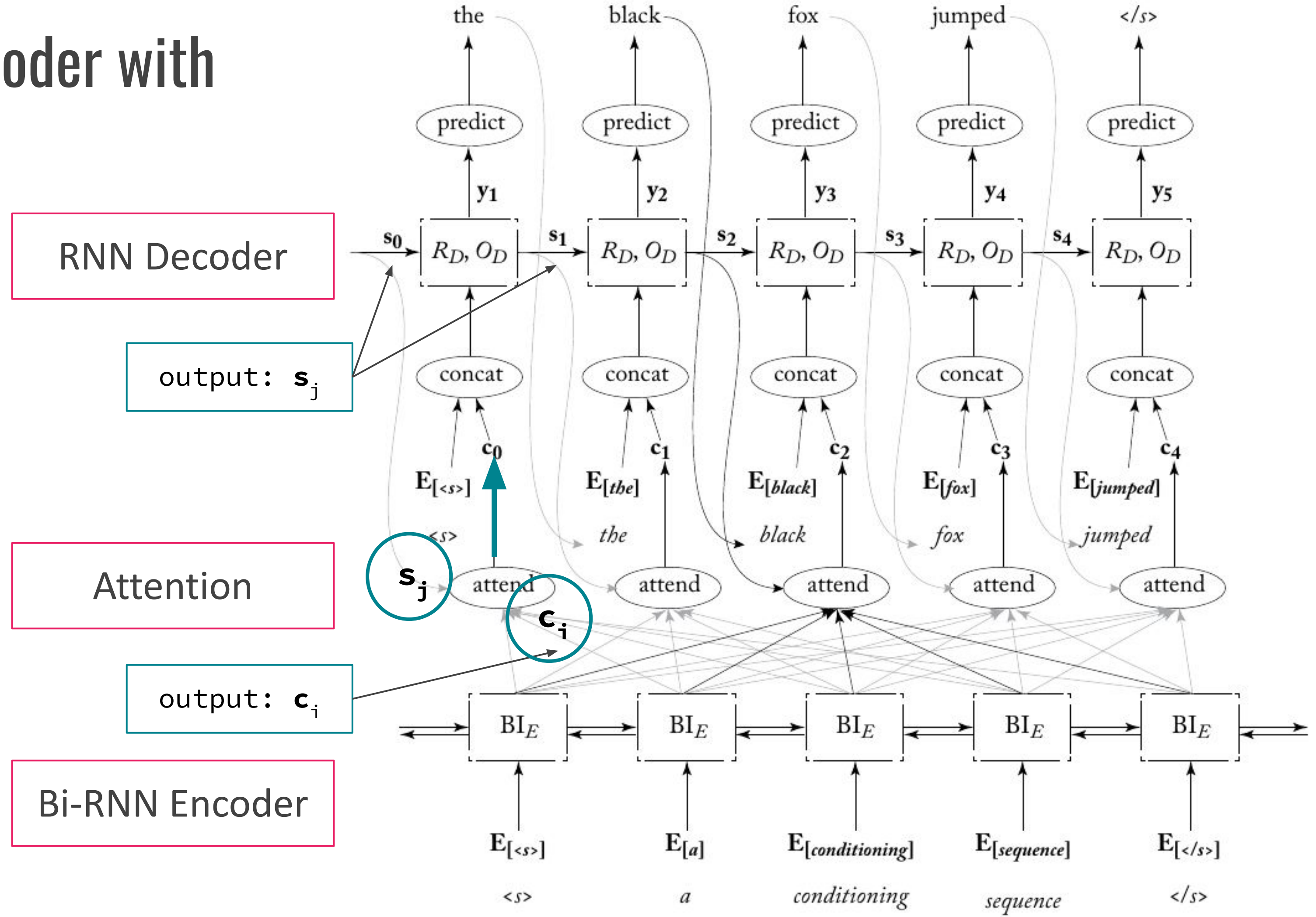
Encoder-decoder with attention



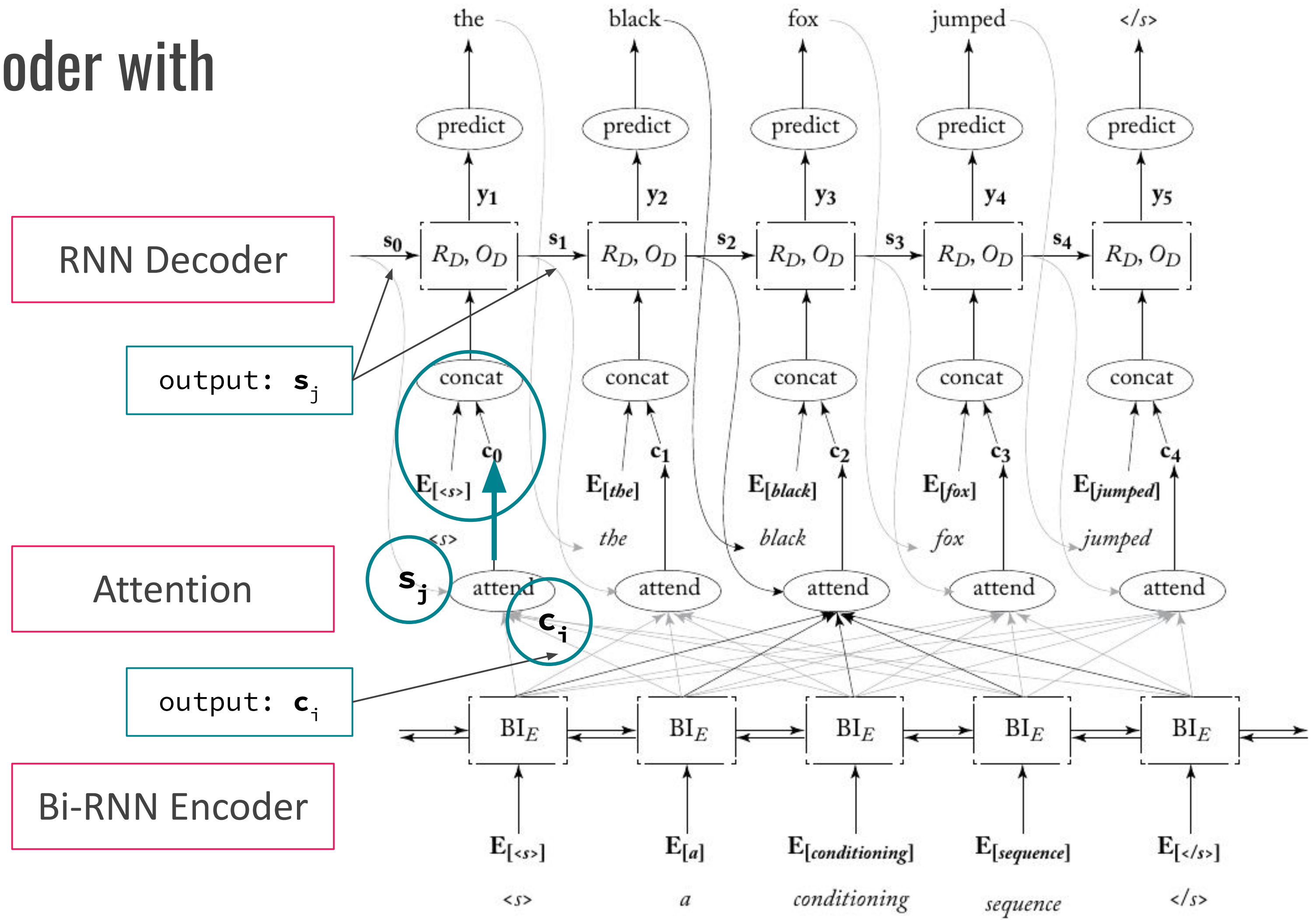
Encoder-decoder with attention



Encoder-decoder with attention



Encoder-decoder with attention



Encoder-decoder with attention

The complete attend function is then:

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

$$\boldsymbol{c^j} = \sum_{i=1}^{n} \boldsymbol{\alpha^j_{[i]}} \cdot \boldsymbol{c_i}$$

$$\boldsymbol{\alpha^j} = \text{softmax}(\boldsymbol{\bar{\alpha}^j_{[1]}}, \ldots, \boldsymbol{\bar{\alpha}^j_{[n]}})$$

$$\boldsymbol{\bar{\alpha}^j_{[i]}} = \text{MLP}^{\text{att}}([\boldsymbol{s_j}; \boldsymbol{c_i}]),$$

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

$$\begin{aligned}
p(t_{j+1} = k \mid \hat{t}_{1:j}, \boldsymbol{x_{1:n}}) &= f(O_{\text{dec}}(\boldsymbol{s_{j+1}})) \\
\boldsymbol{s_{j+1}} &= R_{\text{dec}}(\boldsymbol{s_j}, [\boldsymbol{\hat{t}_j}; \boldsymbol{c^j}]) \\
\boldsymbol{c^j} &= \sum_{i=1}^{n} \boldsymbol{\alpha^j_{[i]}} \cdot \boldsymbol{c_i} \\
\boldsymbol{c_{1:n}} &= \text{biRNN}^{\star}_{\text{enc}}(\boldsymbol{x_{1:n}}) \\
\boldsymbol{\alpha^j} &= \text{softmax}(\boldsymbol{\bar{\alpha}^j_{[1]}}, \ldots, \boldsymbol{\bar{\alpha}^j_{[n]}}) \\
\boldsymbol{\bar{\alpha}^j_{[i]}} &= \text{MLP}^{\text{att}}([\boldsymbol{s_j}; \boldsymbol{c_i}]) \\
\hat{t}_j &\sim p(t_j \mid \hat{t}_{1:j-1}, \boldsymbol{x_{1:n}}) \\
f(\boldsymbol{z}) &= \text{softmax}(\text{MLP}^{\text{out}}(\boldsymbol{z}))
\end{aligned}$$

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

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

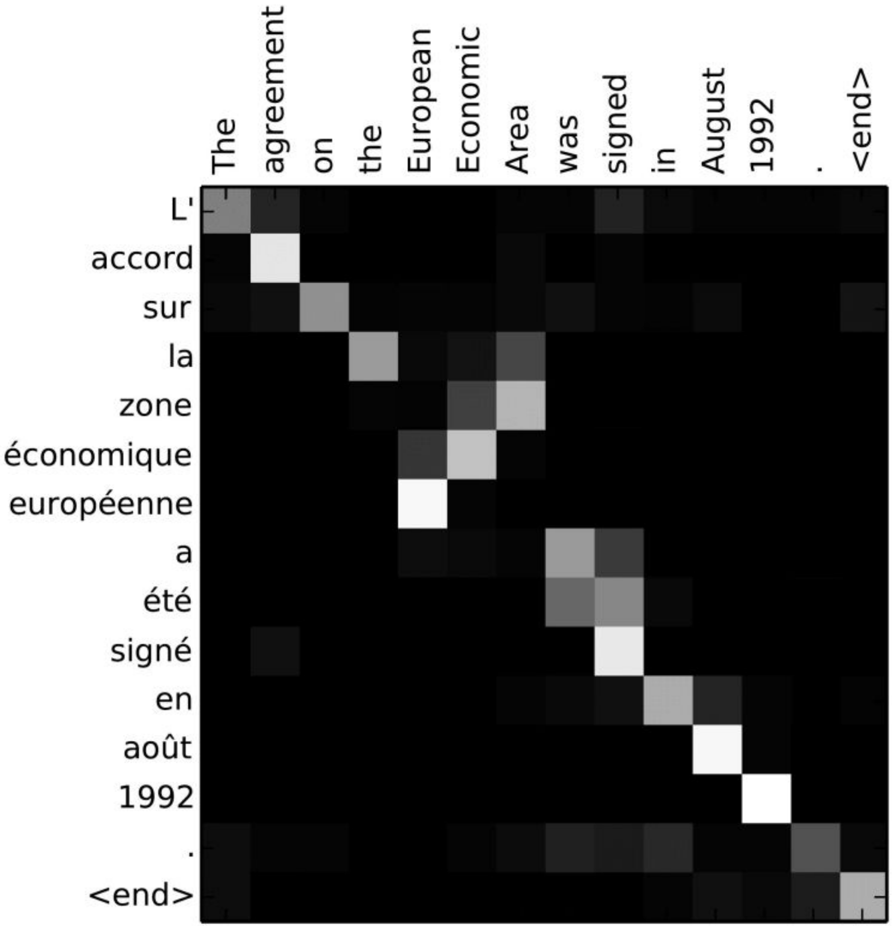
State-of-the-art architecture for MT: [Bahdanau et al 2015] bi-GRU, beam-search ; some improvements:

- **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]
- 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 at 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, target) to training set (target sentences are all natural)
- **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!

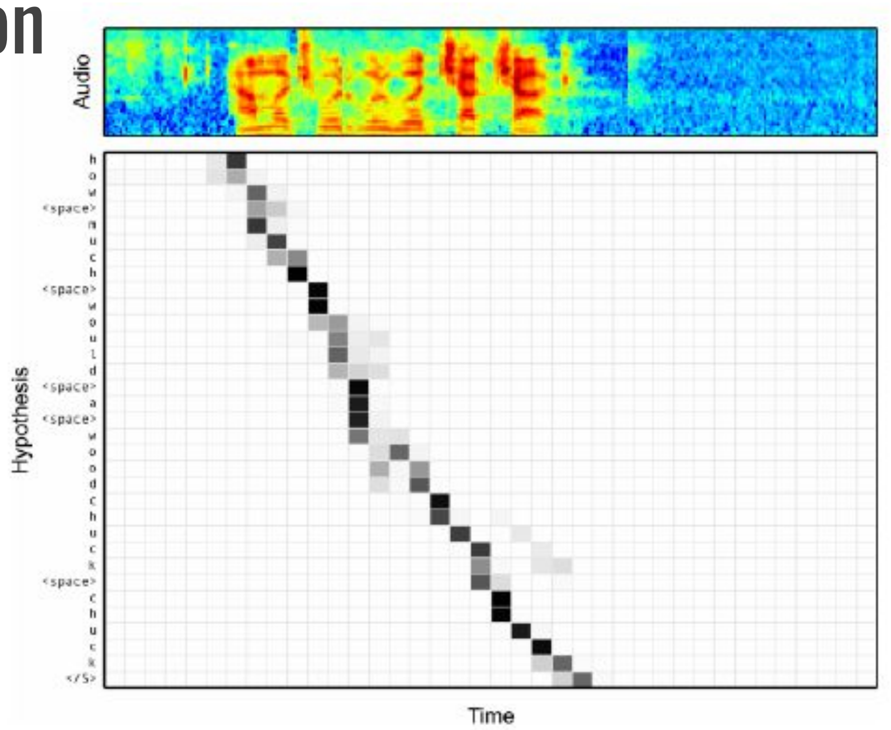
Machine translation

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

Attention is all you need

	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

Transformer models:

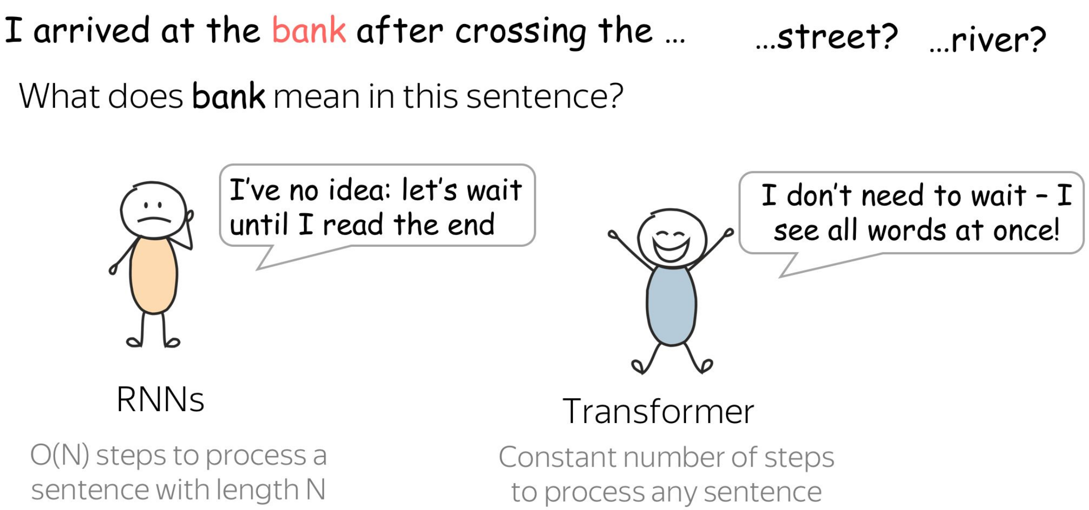
- also takes sequence as input
- but based on attention mechanism **without the RNN architecture**
- it is not required to read in any order the sequence

→ 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

General idea



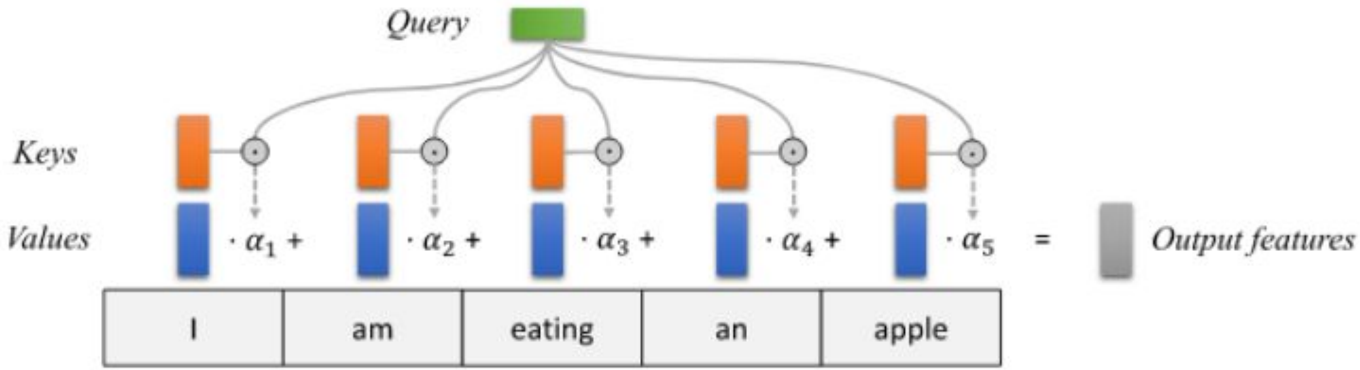
- When encoding a sentence, RNNs won't understand what **bank** means until they read the whole sentence,
- Transformer's encoder tokens interact with each other all at once.

- Transformer's encoder: at each step, tokens look at each other (self-attention), excgane 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

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*

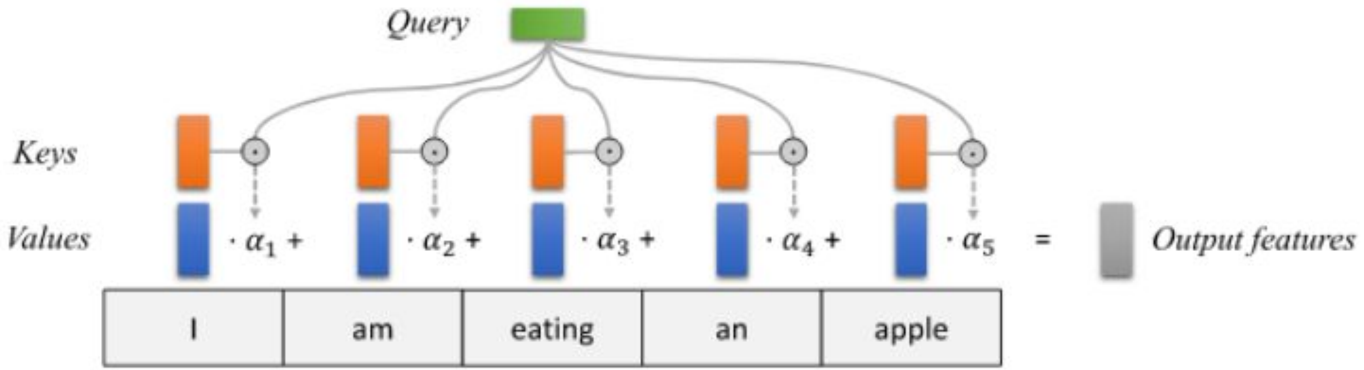


More general view on attention

General idea:

`until now (with RNNs):`

- attention is a *query* on the input ← come from the decoding state $\mathbf{s}_i$
- that we align with a *key* ←corresponding to the input representations $\mathbf{c}_i$
- to operate over an input *value* ←also $\mathbf{c}_i$ used to weight the context vector $\mathbf{c}^j$



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]

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 $\longrightarrow$ some $q_i = W_q x_i$
- **key**: computation of the weights for the output of another x_j viewed as the query $\longrightarrow$ some $k_i = W_k x_i$
- **value**: final weighting to compute the output y_j $\longrightarrow$ 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 → some $q_i = W_q x_i$
- **key**: computation of the weights for the output of another x_j viewed as the query → some $k_i = W_k x_i$
- **value**: final weighting to compute the output y_j → 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 ⇒ 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:

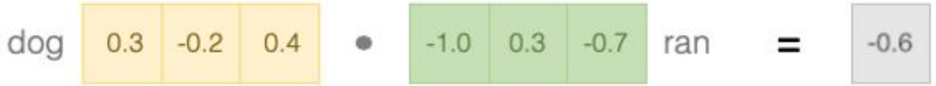
- **query**: interaction with other x_j to compute attention score $x_i, x_j \rightarrow$ some $q_i = W_q x_i$
- **key**: computation of the weights for the output of another x_j viewed as the query $\rightarrow$ some $k_i = W_k x_i$
- **value**: final weighting to compute the output $y_j \rightarrow$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

$w_{ij} = q_i.k_j$: score d'attention x_i/x_j + normalization ⇒ attention weights

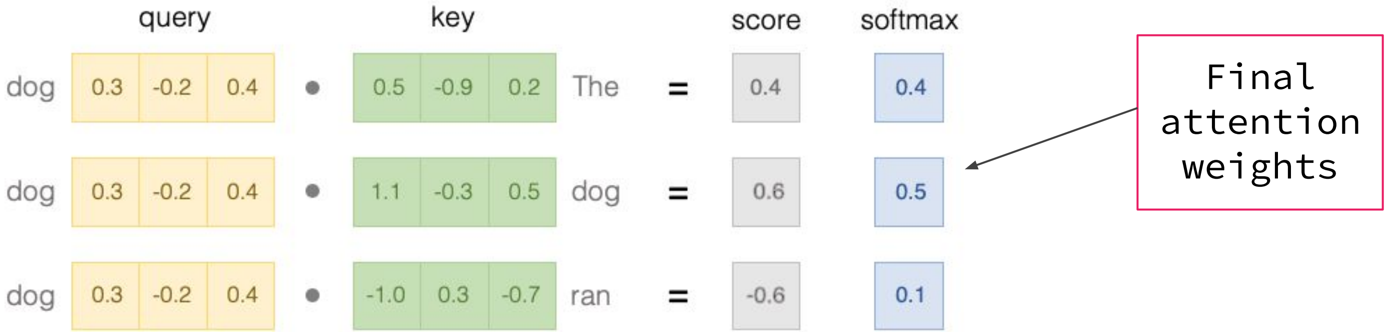
output: $y_i = \sum_j w_{ij}\, v_j$

Computing attention

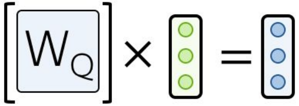
- first: assigns to each word a *query* vector and a *key* vector
- compute a *compatibility function*: assigns a score to each pair of words indicating how strongly they should attend to one another, using dot product between one query and one key $\mathbf{w_{ij} = q_i k_j}$



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

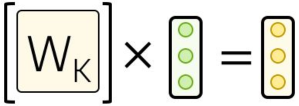


Each vector receives three representations (“roles”)



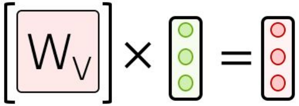
Query: vector **from** which the attention is looking

“Hey there, do you have this information?”



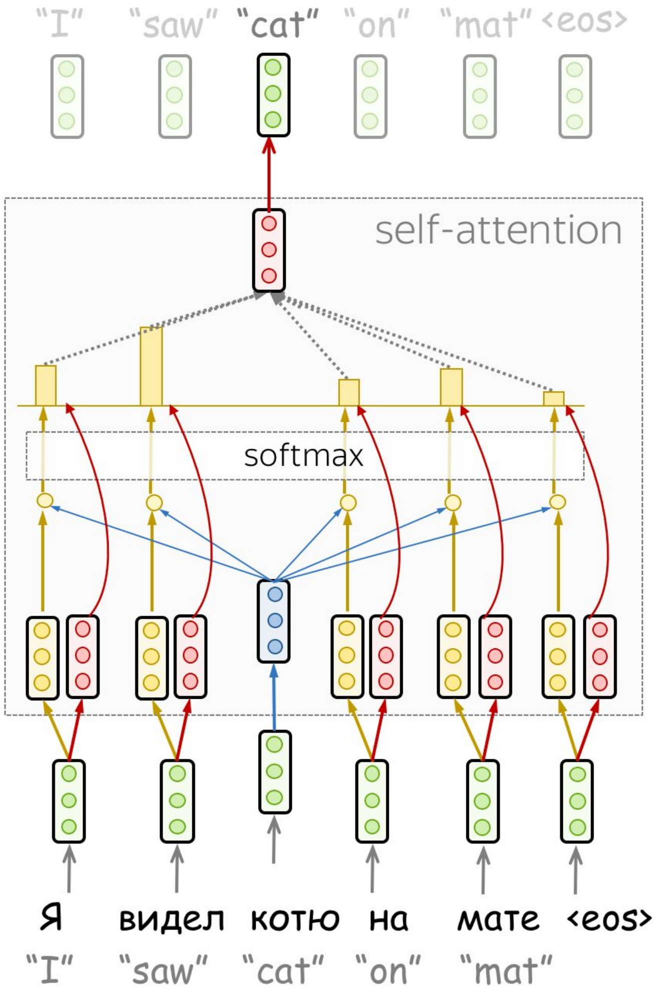
Key: vector **at** which the query looks to compute weights

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

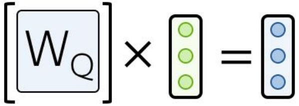


Value: their weighted sum is attention output

“Here’s the information I have!”

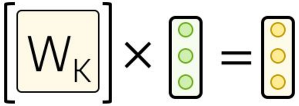


Each vector receives three representations ("roles")



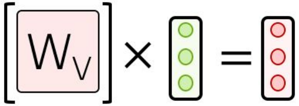
Query: vector **from** which the attention is looking

"Hey there, do you have this information?"



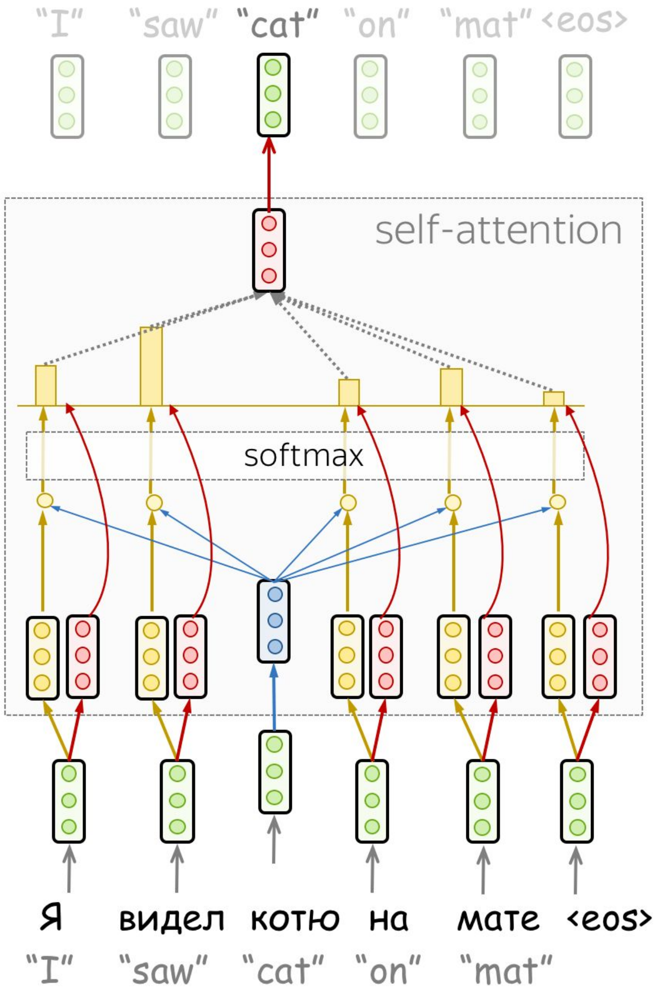
Key: vector **at** which the query looks to compute weights

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



Value: their weighted sum is attention output

"Here's the information I have!"



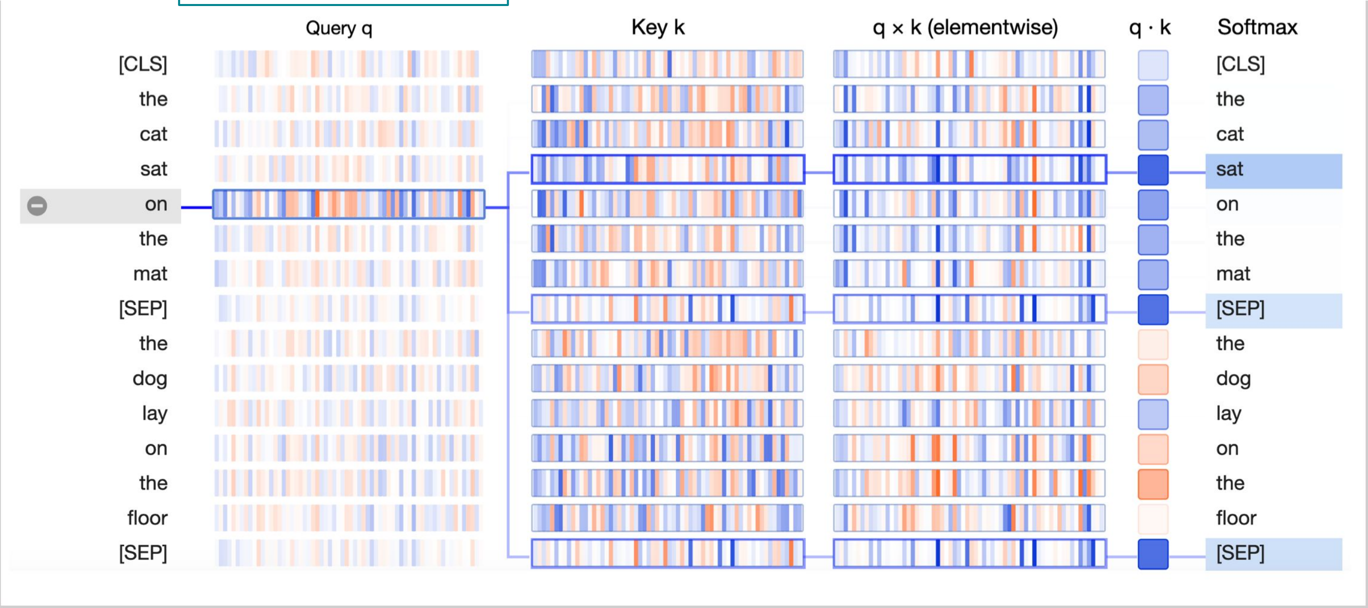
Note: masked attention for the decoder = it can't look ahead https://towardsdatascience.com/deconstructing-bert-part-2-visualizing-the-inner-workings-of-attention-60a16d86b5c1

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

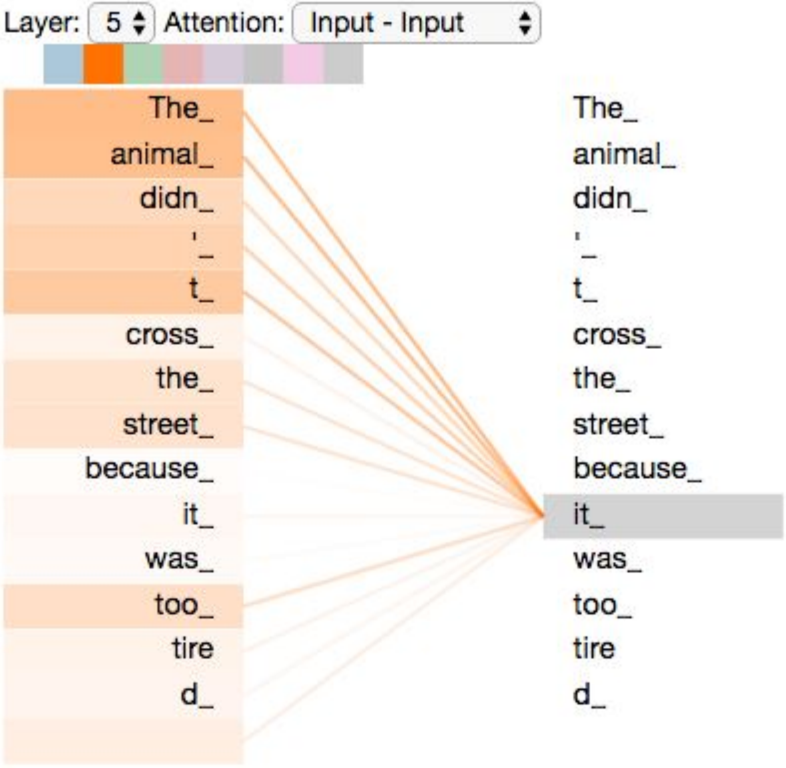


Self-Attention

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*, which 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 all these matrices are combined)

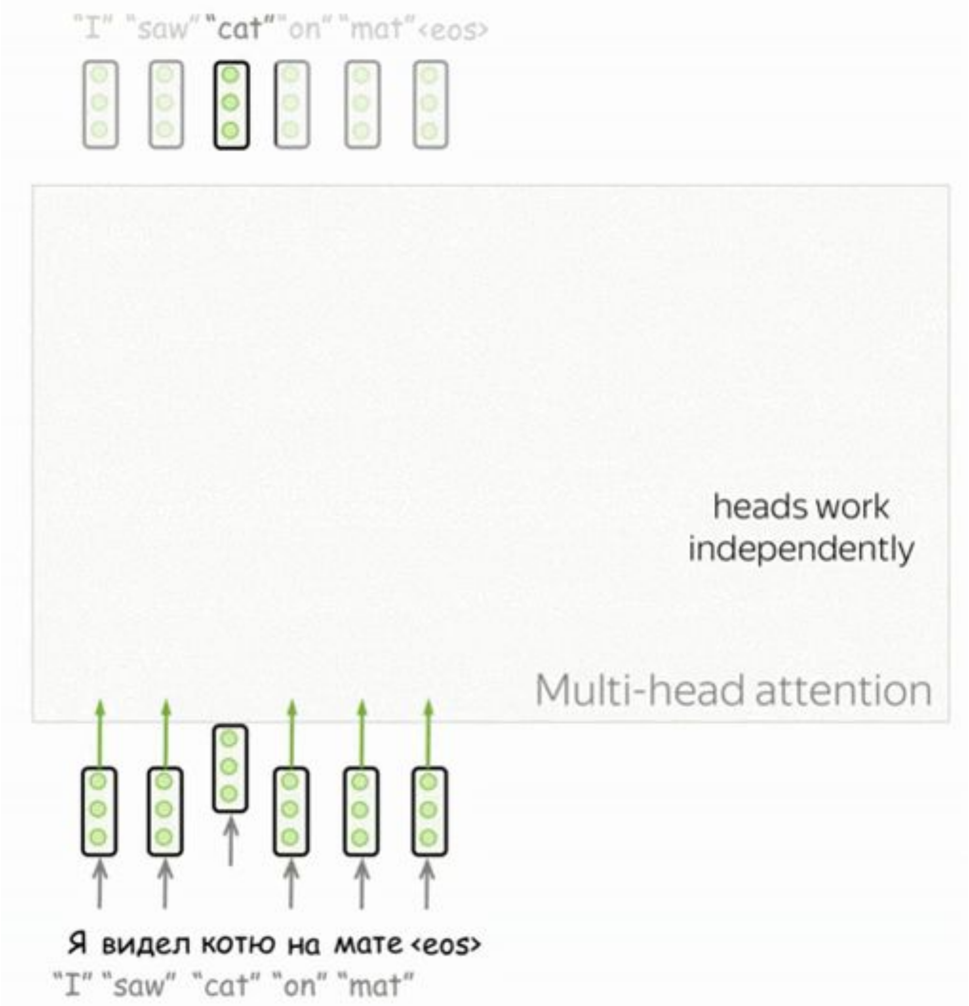
Multi-head attention

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

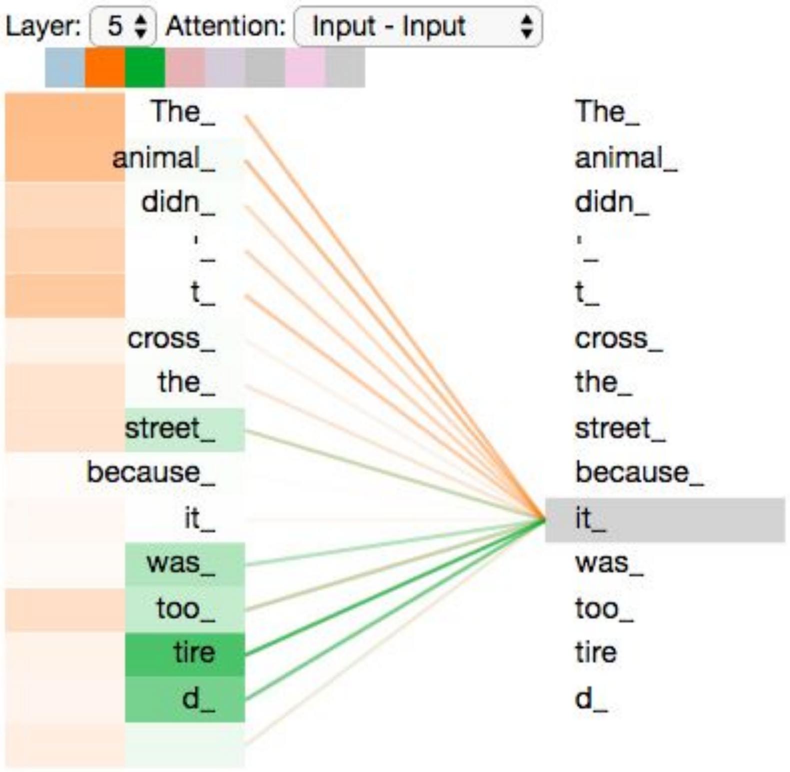


Multi-head attention

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

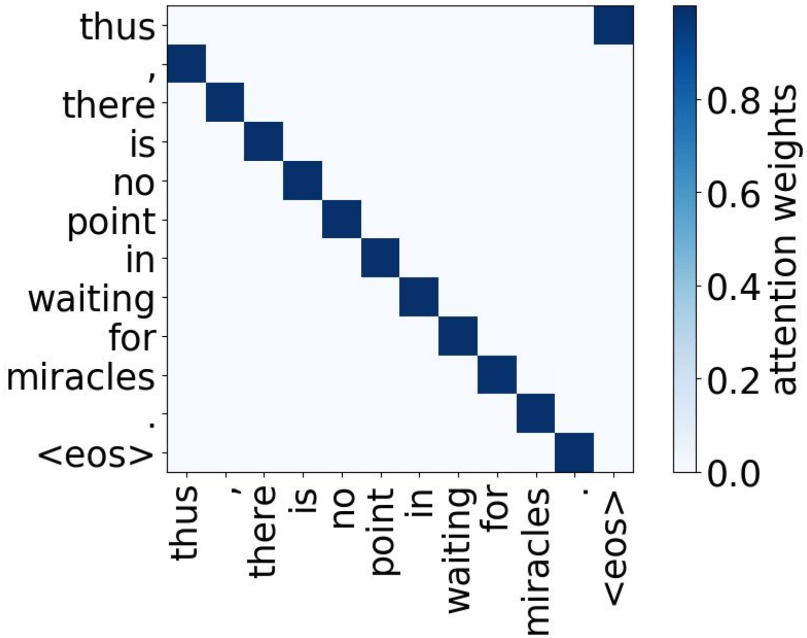
[Voita et al 2019]: some heads play interpretable roles

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

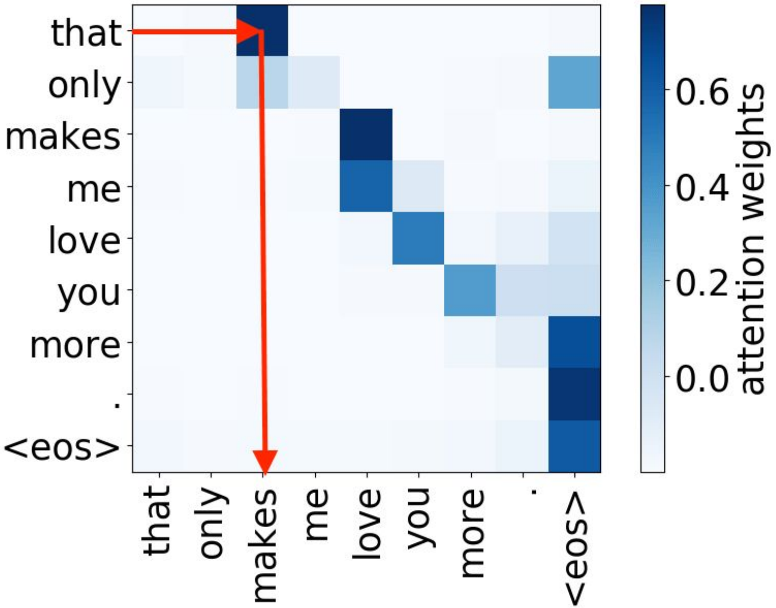


Multi-heads

Positional heads



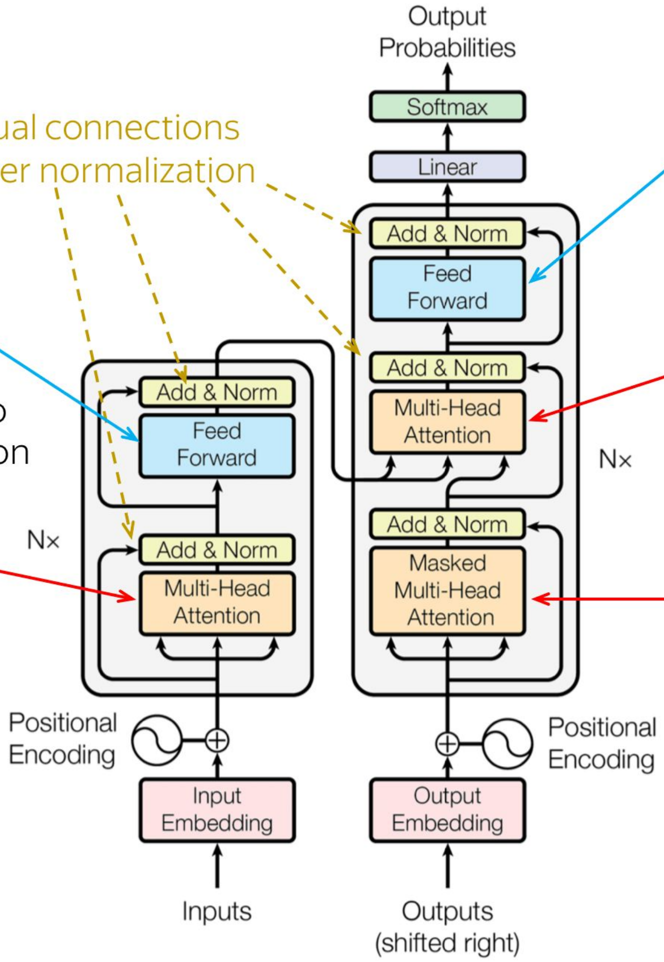
Syntactic heads (subject → verb)



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

The “beast”

Residual connections and layer normalization



Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Encoder self-attention: tokens look at each other

queries, keys, values are computed from encoder states

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Decoder-encoder attention: target token looks at the source

queries – from decoder states; keys and values from encoder states

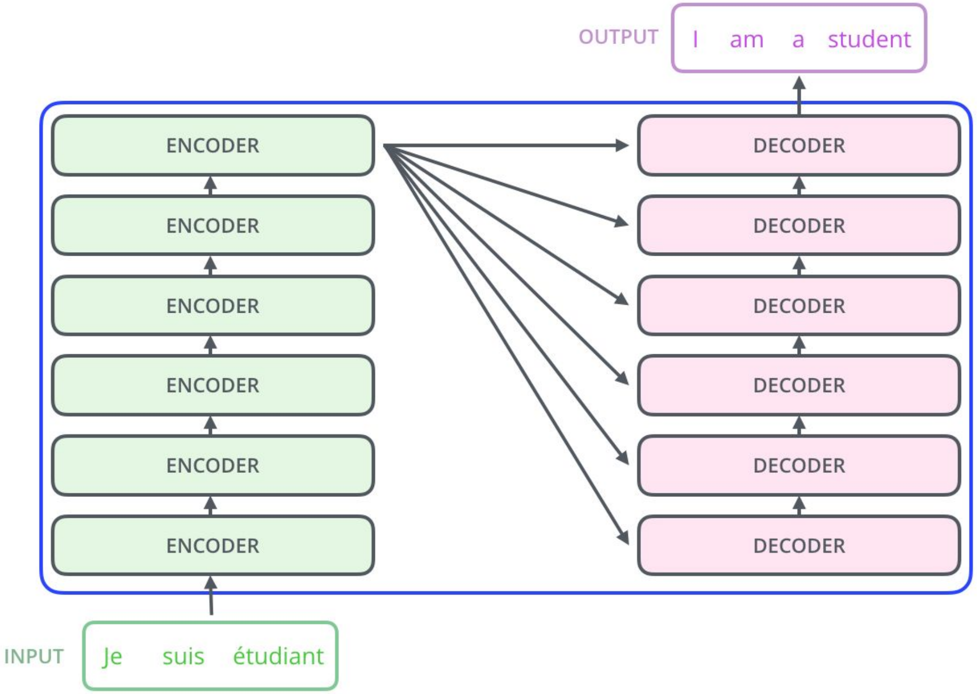
Decoder self-attention (masked): tokens look at the previous tokens

queries, keys, values are computed from decoder states

The *beast*

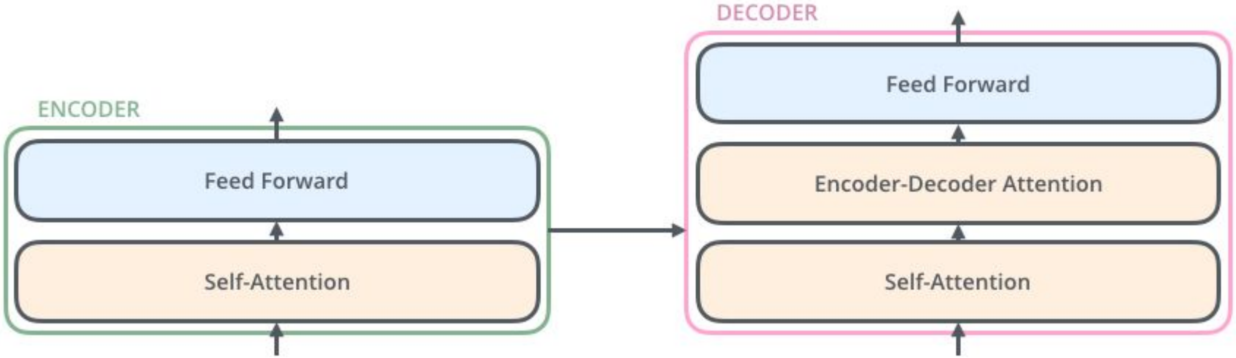
In the paper:

- 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

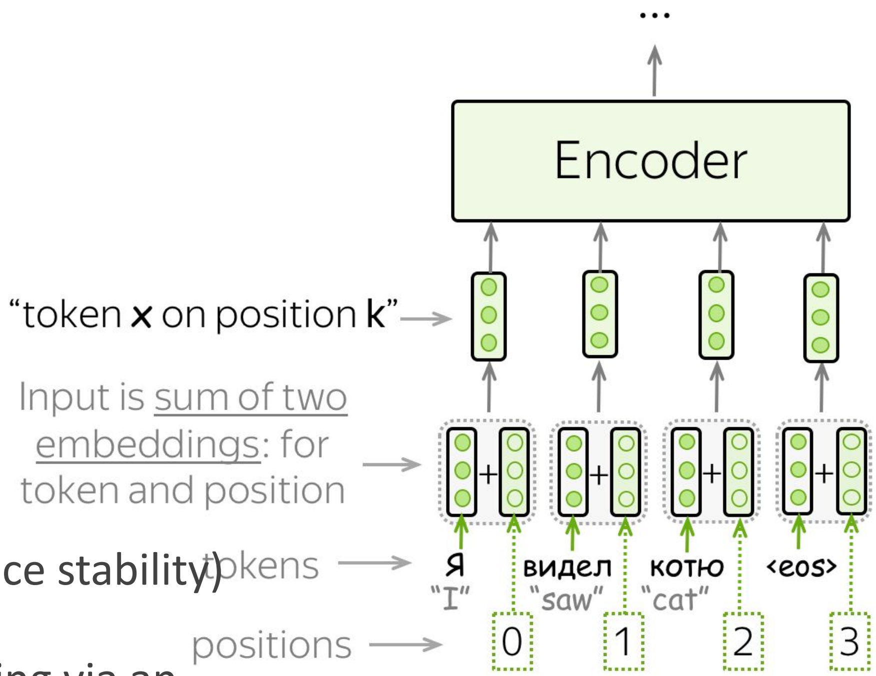
Many elements in the model:

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

Transformers

Many elements in the model:

- self-attention
- multi-head
- non linearities (MLP)
- layer normalizations (improve convergence stability)
- residual connections (ease the learning)
- **sequence ordering?** → positional encoding via an additional embedding (ordinal, relative position or learned)



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