



POLITECNICO
MILANO 1863

Credits for images and examples to Elena Voita's
NLP Course | For You
https://lena-voita.github.io/nlp_course.html

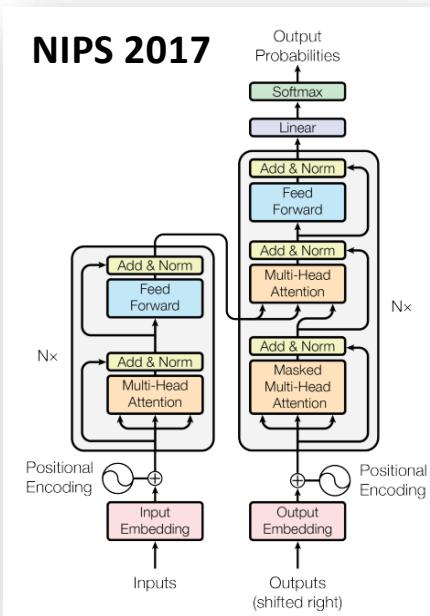
Advanced Deep Learning

- *Attention and Transformers* -

Matteo Matteucci, PhD (matteo.matteucci@polimi.it)
Artificial Intelligence and Robotics Laboratory
Politecnico di Milano

Lecture Objectives

Starting from the seminal work "Attention is all you need" the lecture aims at presenting different perspectives on the Transformer model. Originally introduced to solve the computational limits of recurrent neural models in Natural Language Processing, the Transformer has shown how (self-) attention plays a fundamental role also in other domains such as images understanding (next lecture!).



Today's Special

Sequence to sequence modeling

- Seq2seq for language translation
- Attention mechanism in seq2seq

The Transformer model explained

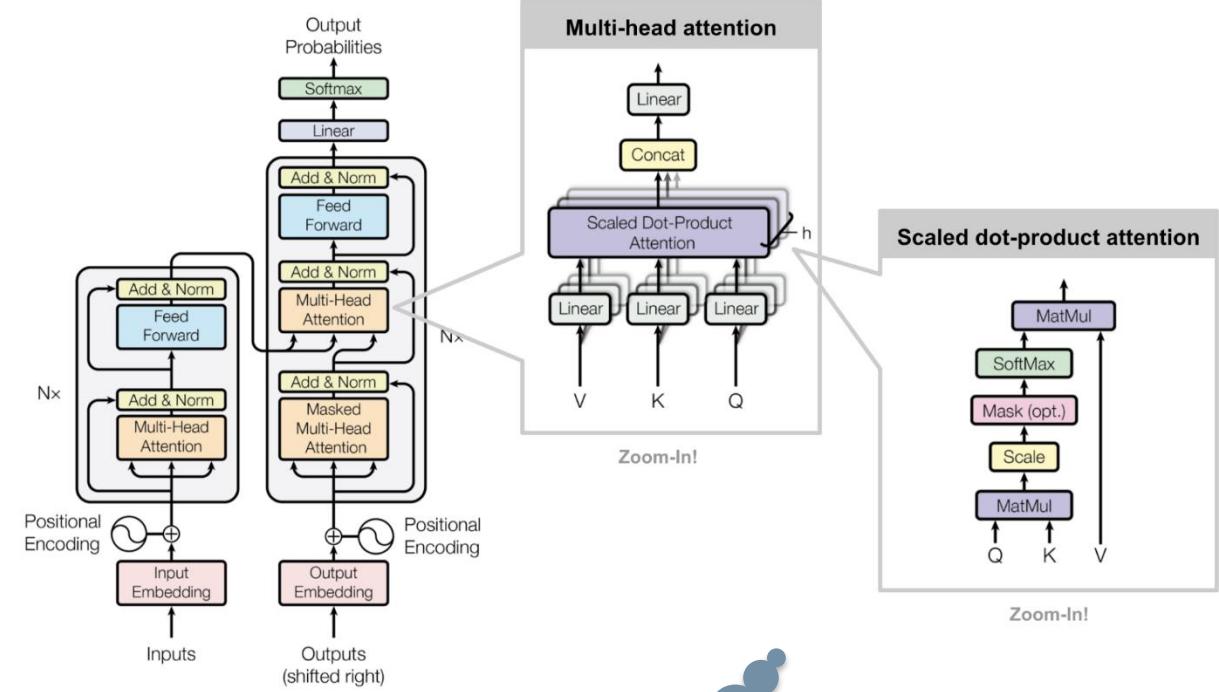
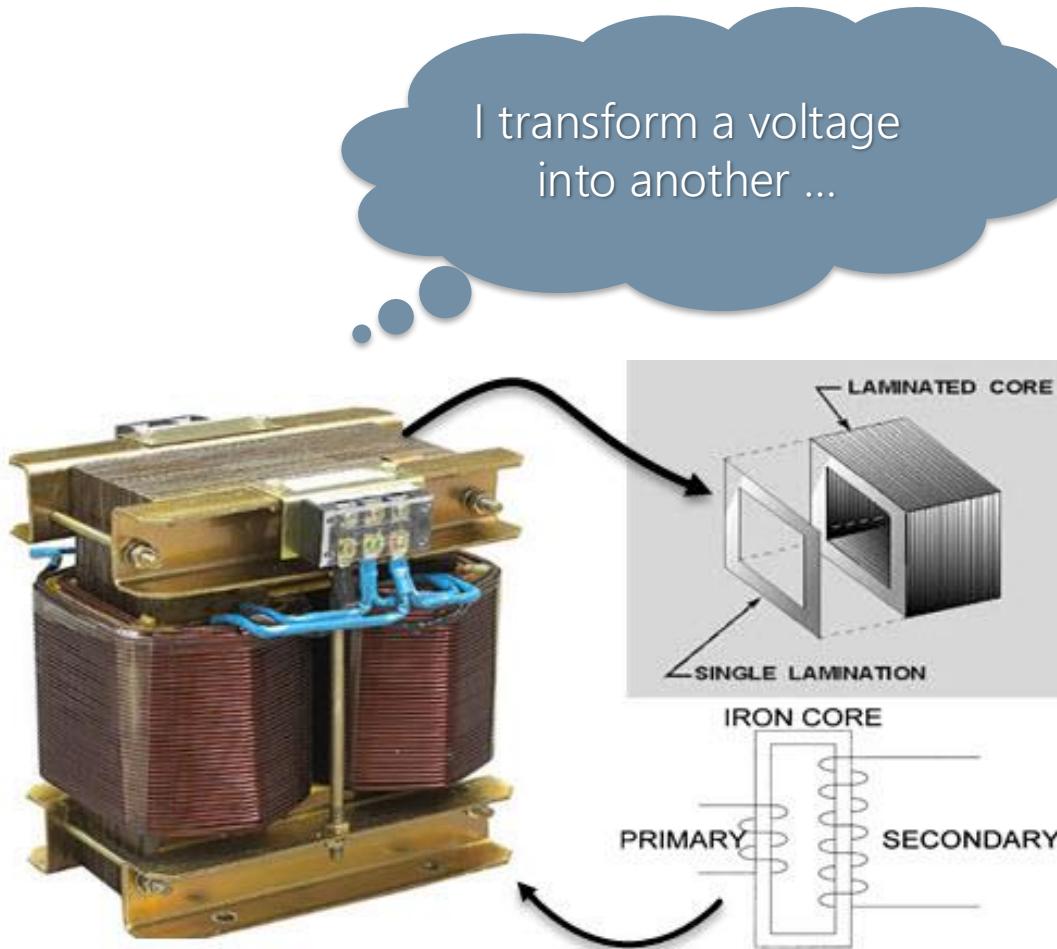
- Self-attention and masked self-attention
- Multi-head attention
- The transformer architecture
- Self-attention invariance and positional encoding
- Multi-head attention interpretation
- ...

A python notebook making sense of this all ...

I knew you expected this
joke at some point 😊



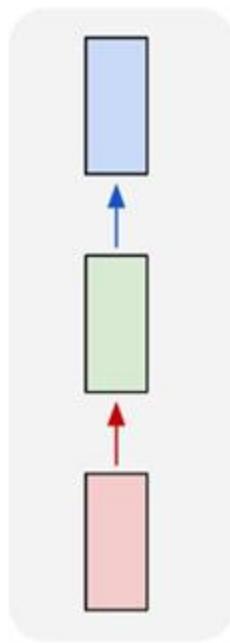
What is a Transformer?



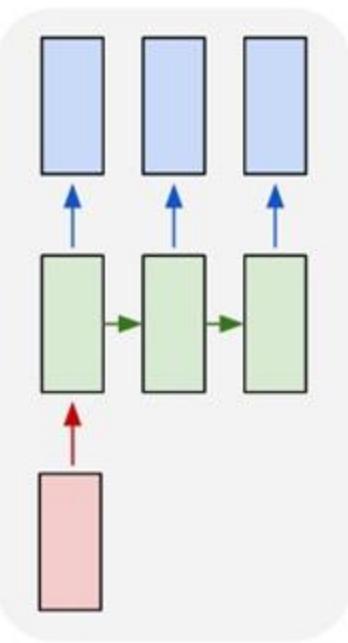
I transform a sequence into
another ... and much more!

Sequential data problems

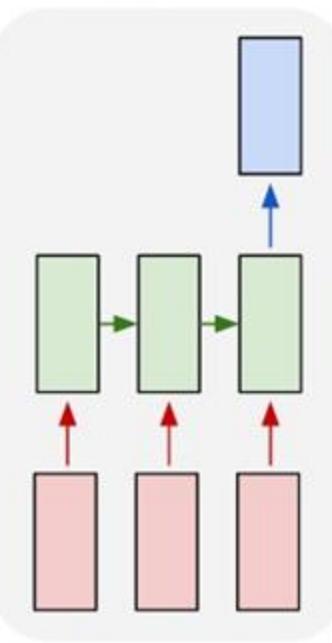
one to one



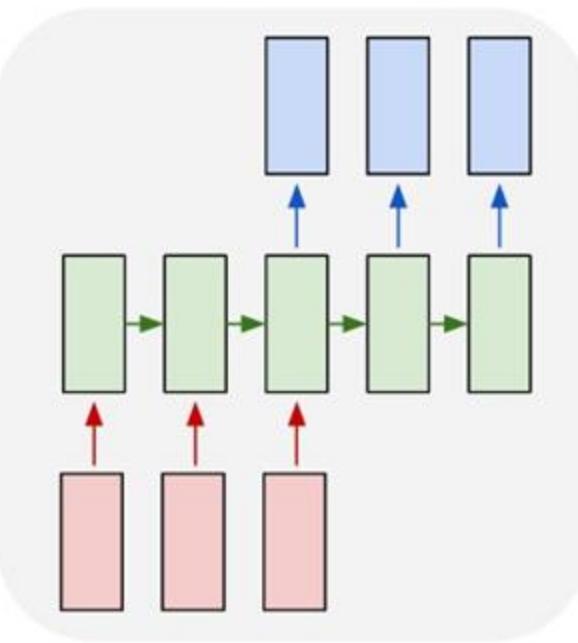
one to many



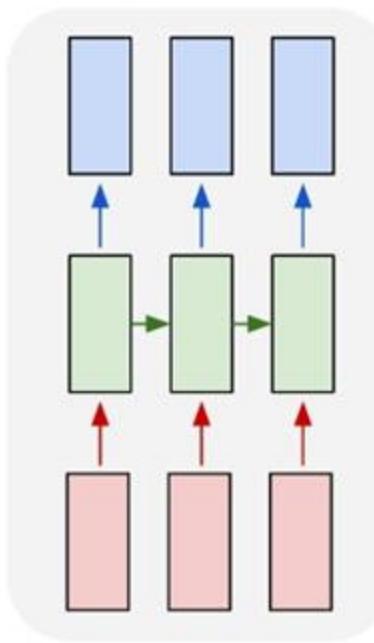
many to one



many to many



many to many



Fixed-sized input to fixed-sized output
(e.g. image classification)

Sequence output
(e.g. image captioning takes an image and outputs a sentence of words).

Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).

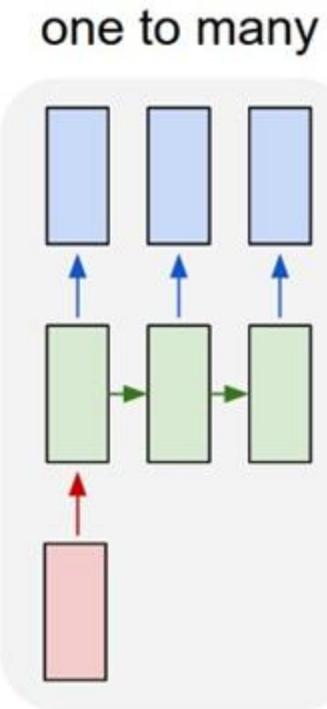
Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French)

Synced sequence input and output (e.g. video classification where we wish to label each frame of the video)

The Unreasonable Effectiveness of Recurrent Neural Networks: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

Sequence to Sequence Learning Examples (1/3)

Image Captioning: input a single image and get a series or sequence of words as output which describe it. The image has a fixed size, but the output has varying length.



A person riding a motorcycle on a dirt road.



Two dogs play in the grass.



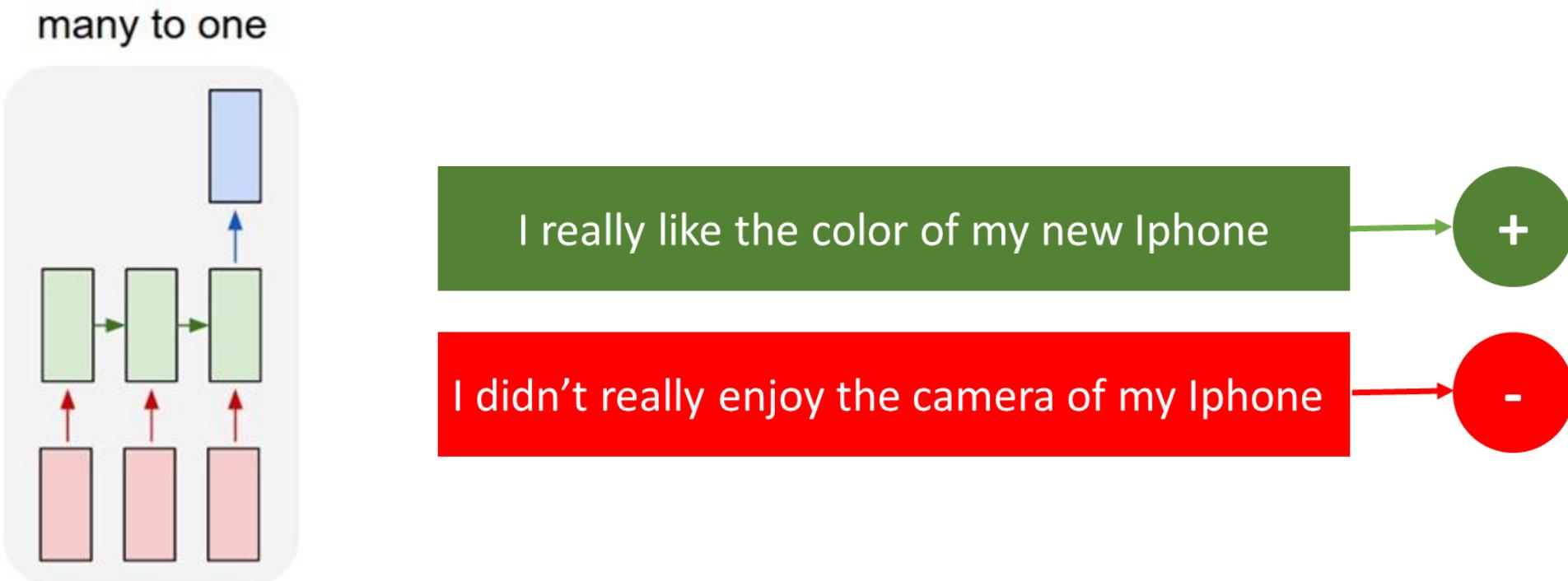
A group of young people playing a game of frisbee.



Two hockey players are fighting over the puck.

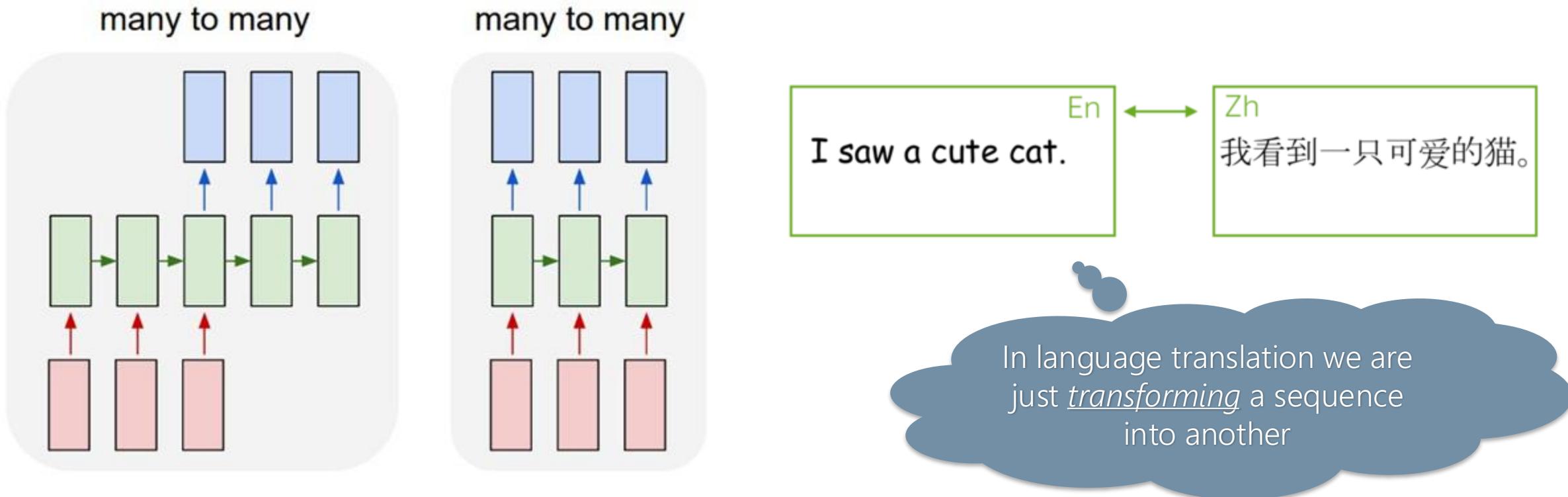
Sequence to Sequence Learning Examples (2/3)

Sentiment Classification/Analysis: input a sequence of characters or words, e.g., a tweet, and classify the sequence into positive or negative sentiment. Input has varying lengths; output is of a fixed type and size.



Sequence to Sequence Learning Examples (3/3)

Language Translation: having some text in a particular language, e.g., English, we wish to translate it in another, e.g., French. Each language has its own semantics and it has varying lengths for the same sentence.



Sequence to Sequence Basics

Given an input sequence

$$x_1, x_2, \dots, x_m$$

and a target output sequence

$$y_1, y_2, \dots, y_n$$

we aim the sequence which maximizes the conditional probability $P(y|x)$

$$y^* = \operatorname{argmax}_y P(y_1, y_2, \dots, y_n | x_1, x_2, \dots, x_m)$$

in sequence-to-sequence modeling, we learn the model $P(y|x, \theta)$ from data and our prediction now becomes

$$y' = \operatorname{argmax}_y P(y_1, y_2, \dots, y_n | x_1, x_2, \dots, x_m, \theta)$$

Called also Conditional Language Model



Conditional Language Models

Language model represents the probability of a sentence (sequence)

$$P(y_1, y_2, \dots, y_n) = \prod_{t=1}^n p(y_t | y_{<t})$$

Conditional language model conditions on a source sentence (sequence)

$$P(y_1, y_2, \dots, y_n | x_1, x_2, \dots, x_m) = \prod_{t=1}^n p(y_t | y_{<t}, x_1, x_2, \dots, x_m)$$

In image captioning x_1, x_2, \dots, x_m can be replaced by an image x



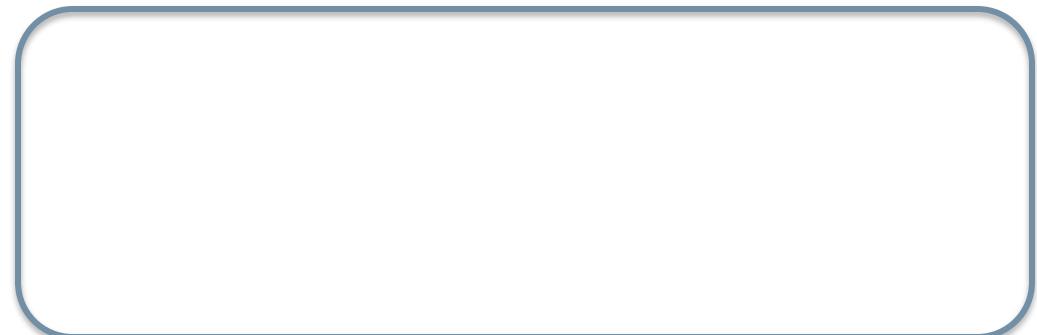
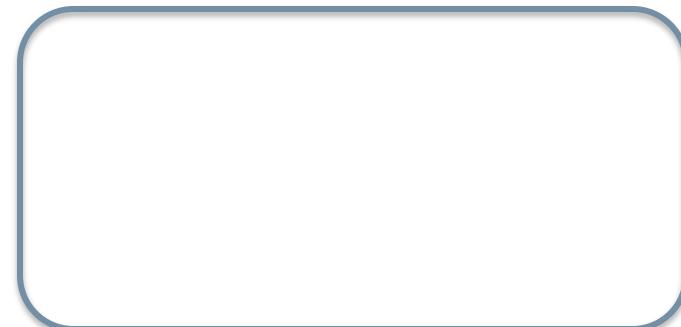
Machine Translation Humans vs Machines

Human Translation

$$y^* = \arg \max_y p(y|x)$$



The “probability” is
intuitive and is given
by a human
translator’s expertise



Machine Translation Humans vs Machines

Human Translation

$$y^* = \arg \max_y p(y|x)$$

The “probability” is intuitive and is given by a human translator’s expertise

Machine Translation

$$y' = \arg \max_y p(y|x, \theta)$$

model

parameters

Questions we need to answer

- **modeling**

How does the model for $p(y|x, \theta)$ look like?

- **learning**

How to find θ ?

- **search**

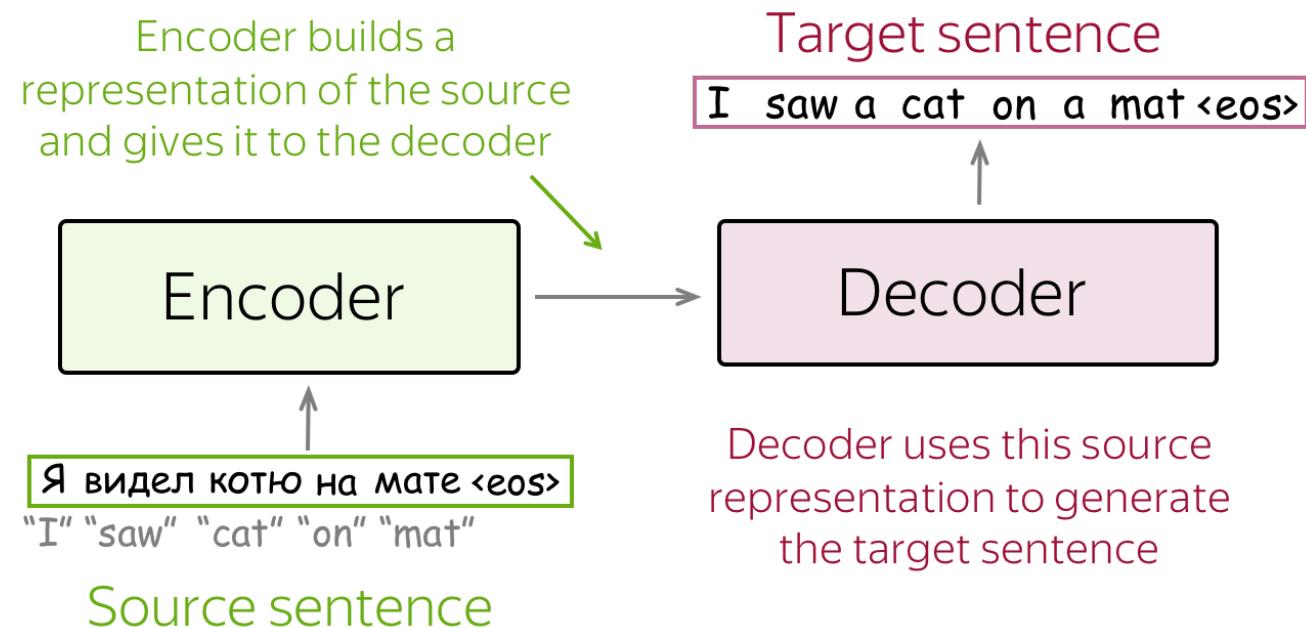
How to find the argmax?

We will just mention about these!

The Encoder-Decoder Framework

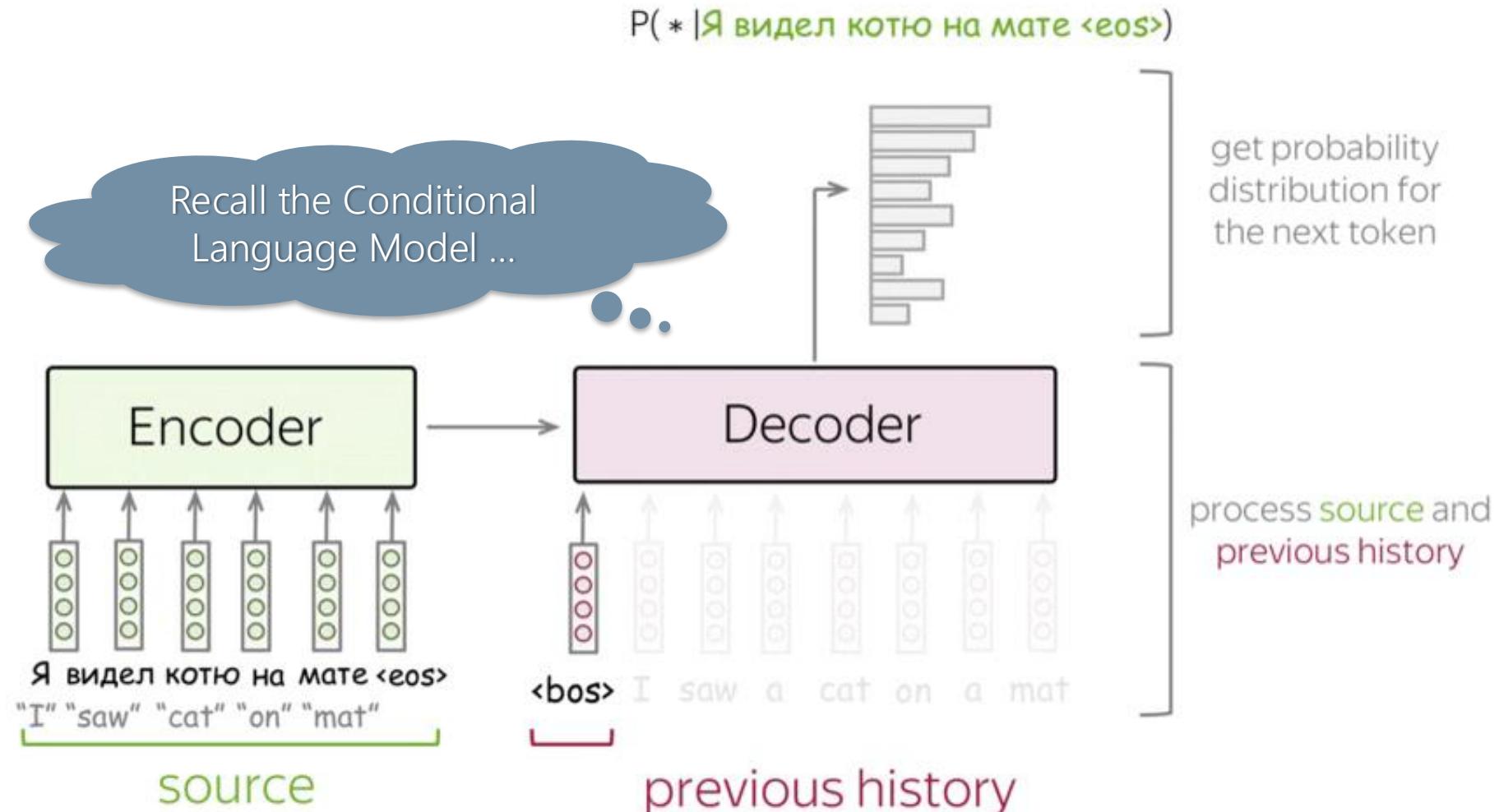
True for most of deep learning models ...

Sequence-to-sequence models as encoder-decoder architectures



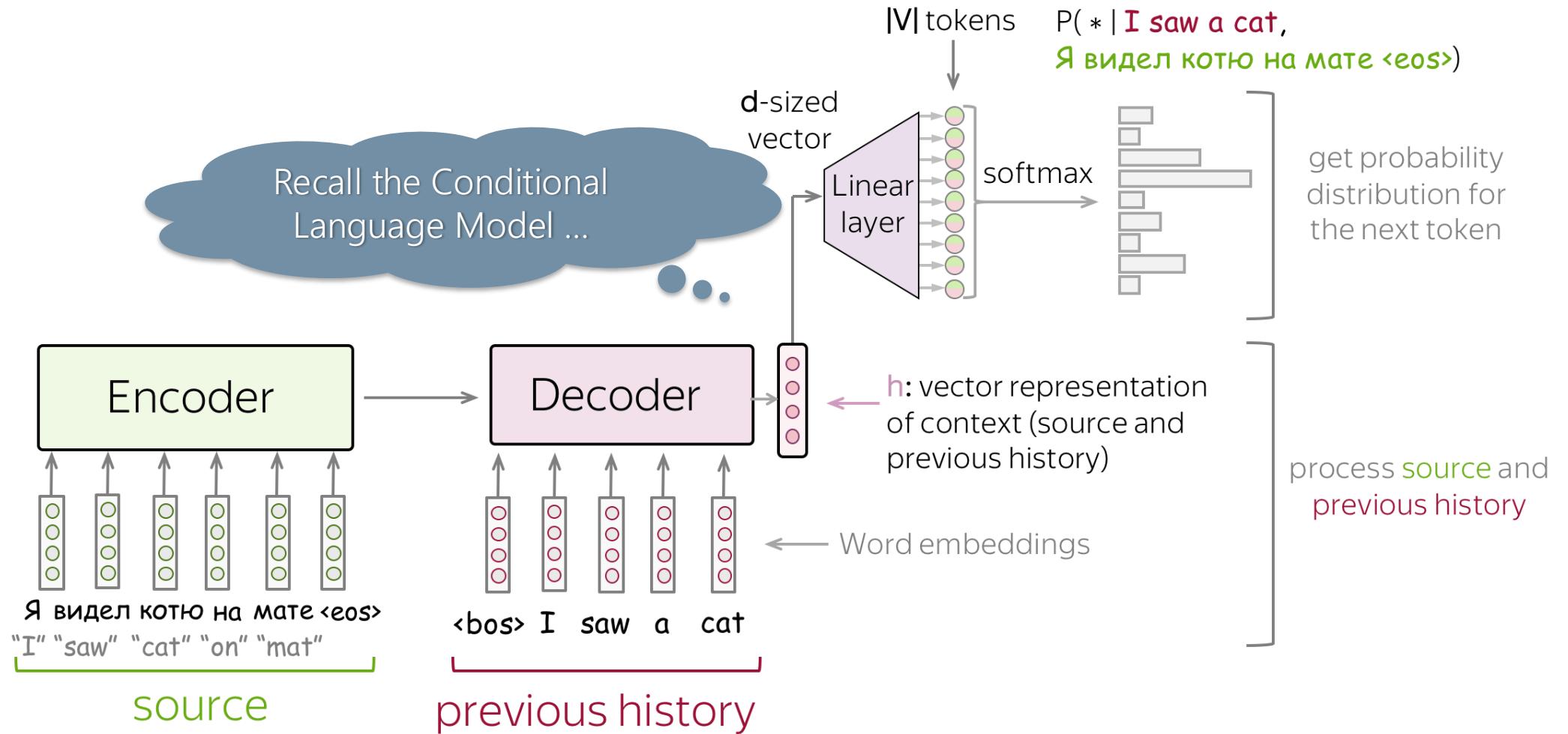
The Encoder-Decoder Framework

Sequence-to-sequence models as encoder-decoder architectures

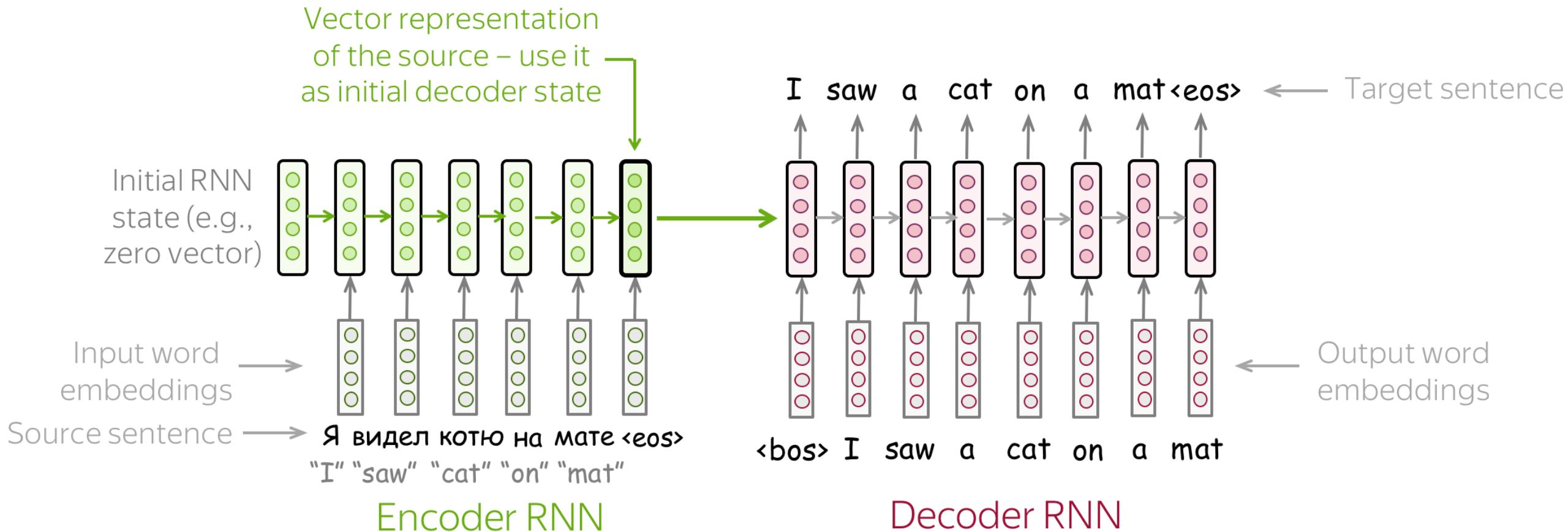


The Encoder-Decoder Framework

Sequence-to-sequence models as encoder-decoder architectures



Encoding/Decoding with Recurrent Neural Networks (LSTM)



Training Sequence to Sequence Models

Given a training sample $\langle x, y \rangle$ with input sequence $x = x_1, x_2, \dots, x_m$ and target sequence $y = y_1, y_2, \dots, y_n$,

Source sequence:

Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"

Target sequence:

I saw a cat on a mat <eos>

previous tokens we want the model
 to predict this

← one training example

← one step for this example

time t our model predicts

$$p_t = p(\cdot | y_1, y_2, \dots, y_{t-1}, x_1, x_2, \dots, x_m)$$

Using a one-hot vector for y_t we can use the cross-entropy as loss

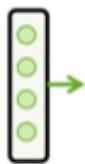
$$\text{loss}_t(p_t, y_t) = - \sum_{i=1}^{|V|} y_t^{(i)} \log(p_t^{(i)}) = -y_t^T \log(p_t)$$



Training Sequence to Sequence Models

Over the entire sequence cross-entropy becomes $-\sum_{t=0}^n y_t^T \log(p_t)$

Encoder: read source



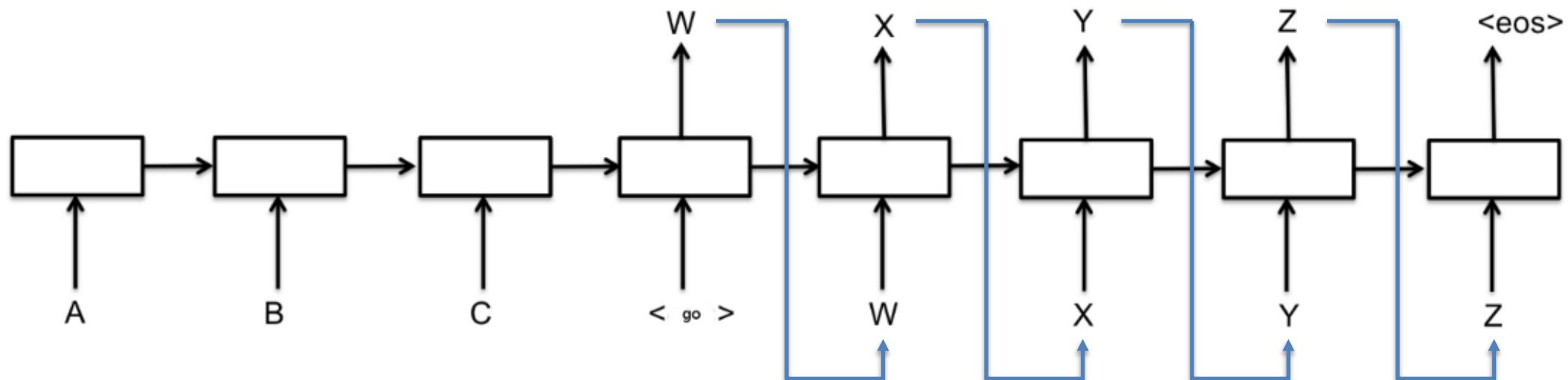
we are here
↓
Source: Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
Target: I saw a cat on a mat <eos>



Training Sequence to Sequence Models

Seq2Seq model follows a classical encoder decoder architecture

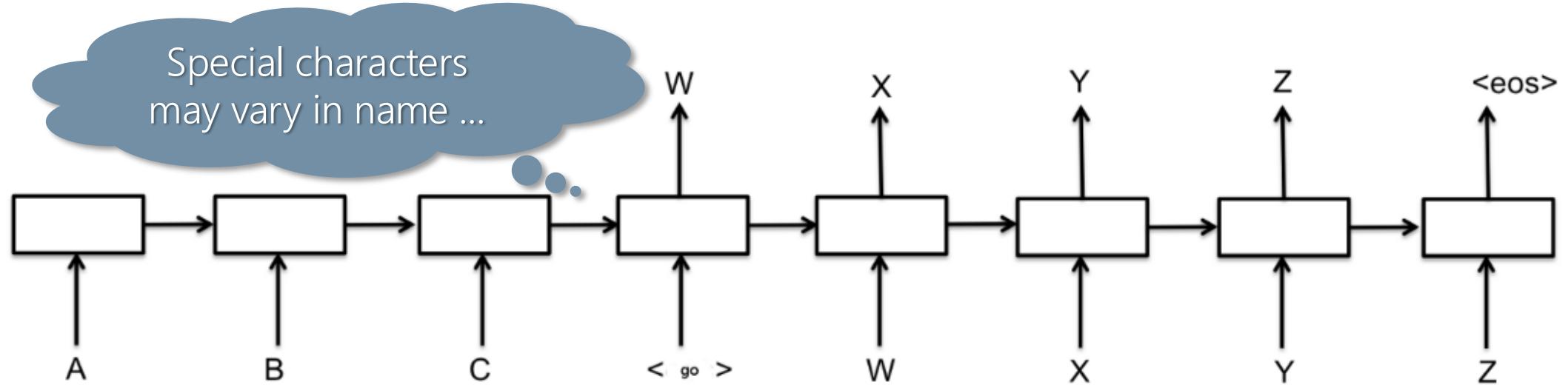
- At training time the decoder **does not** feed the output of each time step to the next; the input to the decoder time steps are the target from the training
- At inference time the decoder feeds the output of each time step as an input to the next one



Sequence to sequence learning with Neural networks: <https://arxiv.org/pdf/1409.3215.pdf>



Special Characters



<PAD>: During training, examples are fed to the network in batches. The inputs in these batches need to be the same width. This is used to pad shorter inputs to the same width of the batch

<EOS>: Needed for batching on the decoder side. It tells the decoder where a sentence ends, and it allows the decoder to indicate the same thing in its outputs as well.

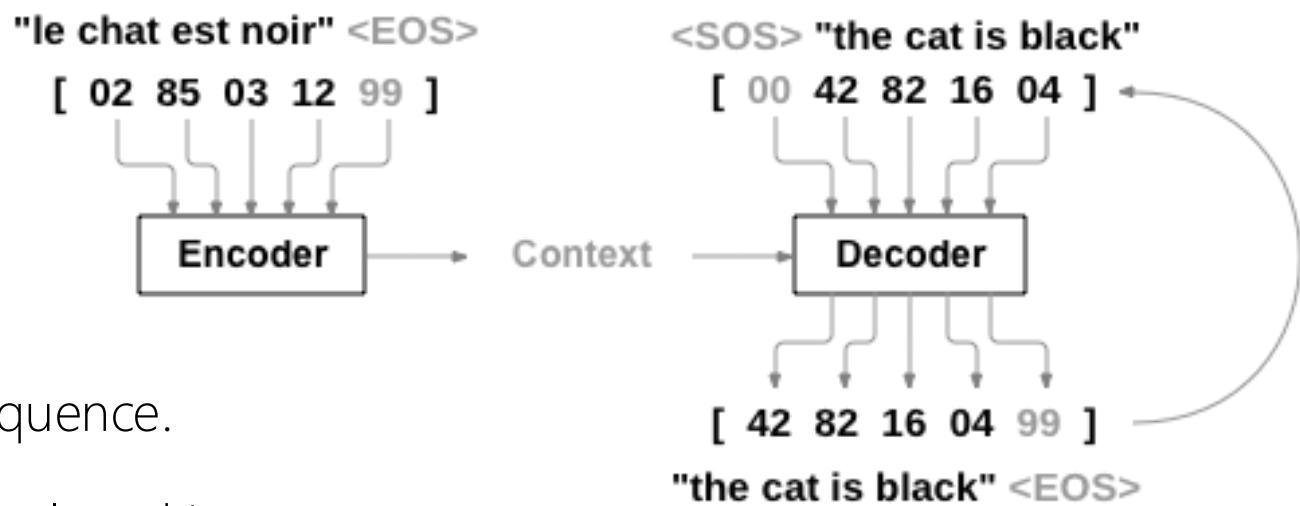
<UNK>: On real data, it can vastly improve the resource efficiency to ignore words that do not show up often enough in your vocabulary by replace those with this character.

<SOS>/<GO>: This is the input to the first time step of the decoder to let the decoder know when to start generating output.

Sequence to sequence learning with Neural networks: <https://arxiv.org/pdf/1409.3215.pdf>

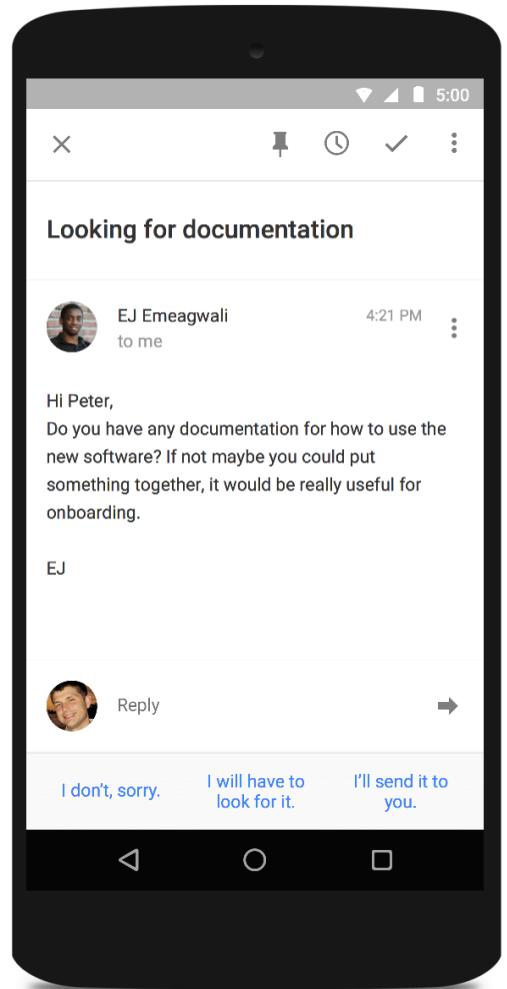
Dataset Batch Preparation

1. Sample batch_size pairs of (source_sequence, target_sequence).
2. Append <EOS> to the source_sequence
3. Prepend <SOS> to the target_sequence to obtain the target_input_sequence and append <EOS> to obtain target_output_sequence.
4. Pad up to the max_input_length (max_target_length) within the batch using the <PAD> token.
5. Encode tokens based of vocabulary (or embedding)
6. Replace out of vocabulary (OOV) tokens with <UNK>. Compute the length of each input and target sequence in the batch.

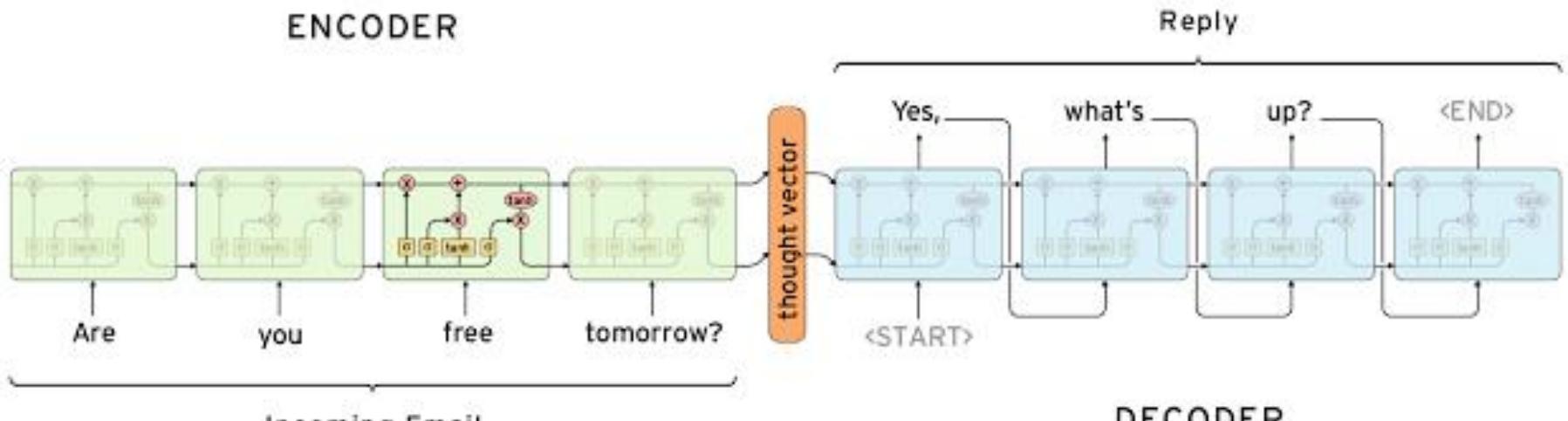


```
vocabulary = {<SOS>: 00,  
              <EOS>: 99,  
              <UNK>: 01,  
              <PAD>: 03,  
              "the": 42,  
              "is": 16,  
              ... }
```

Sequence to Sequence Training



Given $\langle S, T \rangle$ pairs, read S , and output T' that best matches T



$$p(y_1, \dots, y_{T'} | x_1, \dots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \dots, y_{t-1})$$

$$\frac{1}{|\mathcal{S}|} \sum_{(T,S) \in \mathcal{S}} \log p(T|S)$$

The Unreasonable Effectiveness of Recurrent Neural Networks: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

Greedy Decoding vs Beam Search

Once we have trained the model, i.e., learned θ , we predict a sequence $y = y_1, y_2, \dots, y_n$ given $x = x_1, x_2, \dots, x_m$ by selecting y' as

$$y' = \operatorname{argmax}_y \prod_{t=1}^n P(y_t | y_{<t}, x_1, x_2, \dots, x_m, \theta)$$

To compute the argmax over all possible sequences we can use:

- Greedy Decoding: at each step, pick the most probable token

$$y' = \operatorname{argmax}_y \prod_{t=1}^n P(y_t | y_{<t}, x_1, x_2, \dots, x_m, \theta) \approx \prod_{t=1}^n \operatorname{argmax}_{y_t} P(y_t | y_{<t}, x_1, x_2, \dots, x_m, \theta)$$

but this does not guarantee to reach the best sequence and it does not allow to backtrack from errors in early stages of classification.

Greedy Decoding vs Beam Search

Once we have trained the mode, i.e., learned θ , we predict a sequence $y = y_1, y_2, \dots, y_n$ given $x = x_1, x_2, \dots, x_m$ by selecting y' as

$$y' = \operatorname{argmax}_y \prod_{t=1}^n P(y_t | y_{<t}, x_1, x_2, \dots, x_m, \theta)$$

To compute the argmax over all possible sequences we can use:

- Beam Search: Keep track of several most probable hypotheses

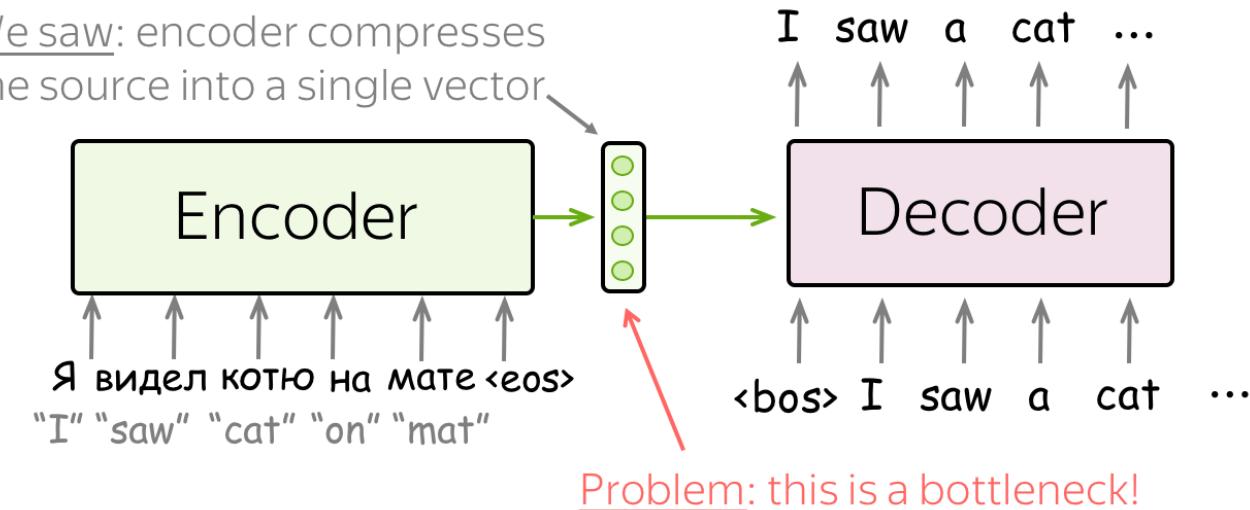
<bos>

Start with the begin of sentence token or with an empty sequence

You Need Attention!

Fixed source representation in basic sequence-to-sequence models may become a representation bottleneck as it gets suboptimal for both

- Encoder: it may be hard to compress the full sentence;
- Decoder: different information may be relevant at different steps.

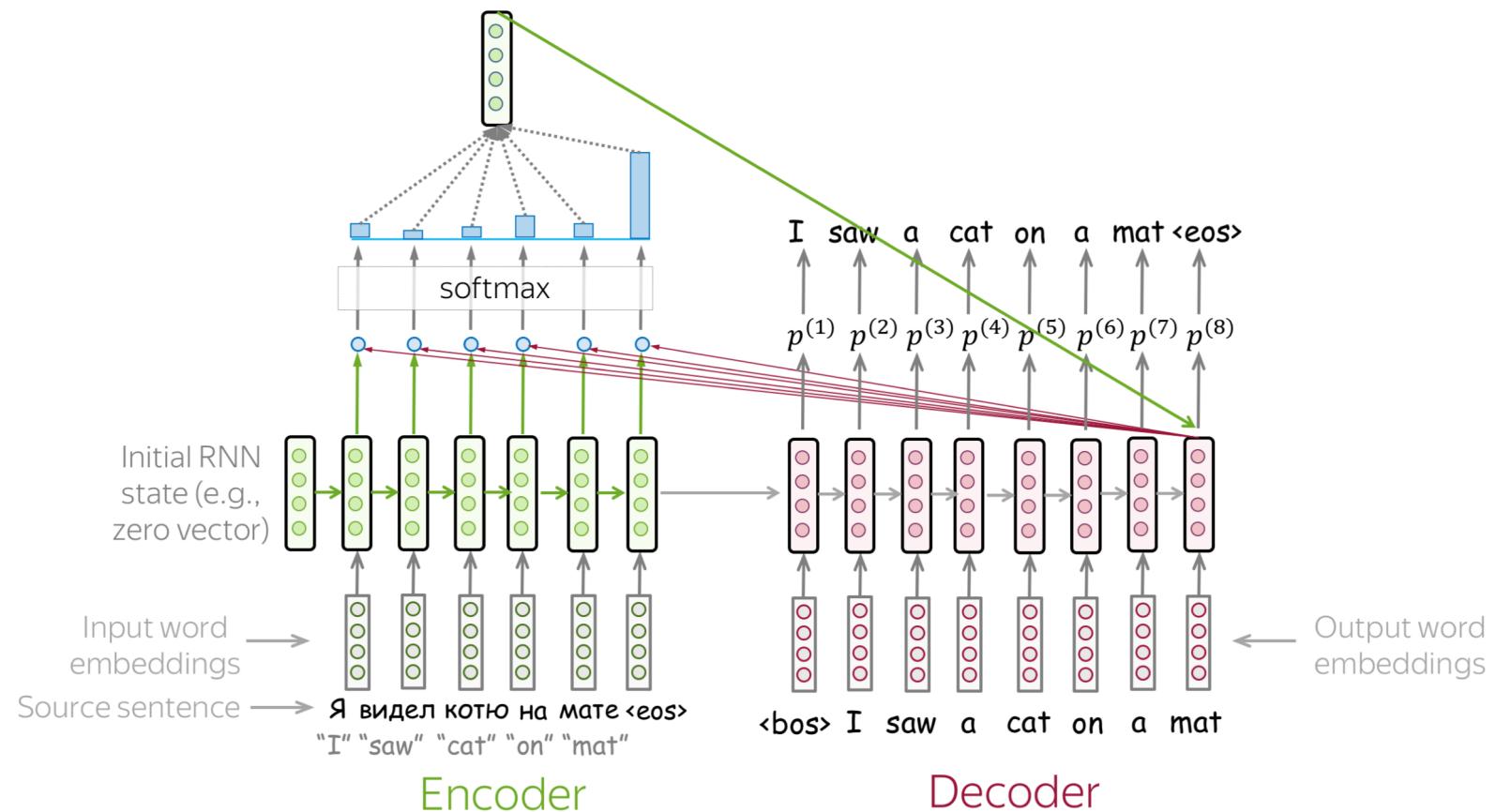


Attention let the model focus on different parts of the input

Neural Machine Translation by Jointly Learning to Align and Translate: <https://arxiv.org/pdf/1409.0473.pdf>

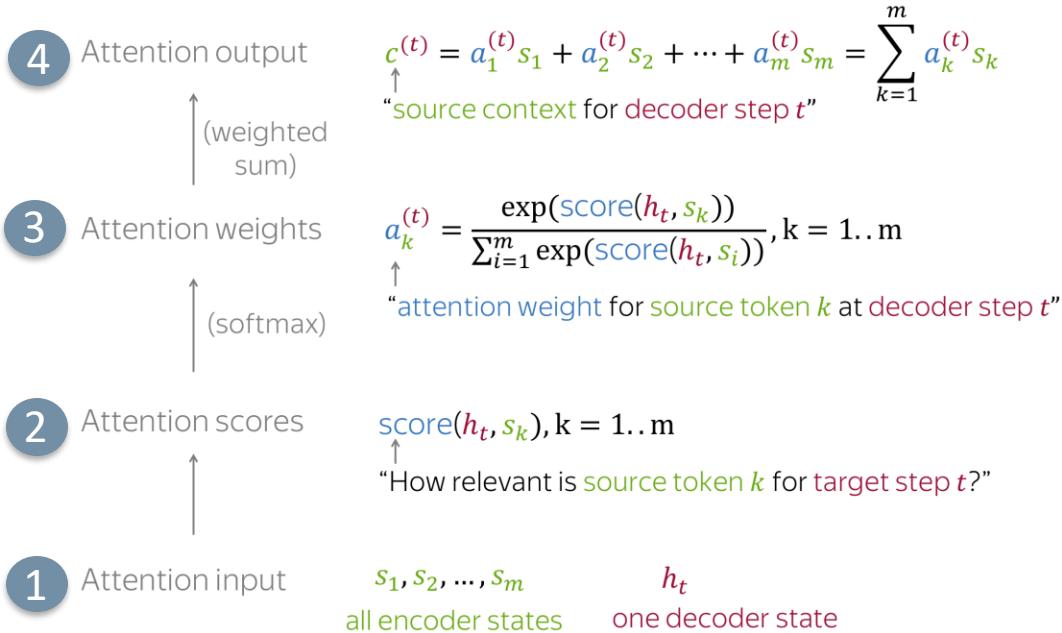
You Need Attention!

Decoder uses attention to decide which source parts are more important

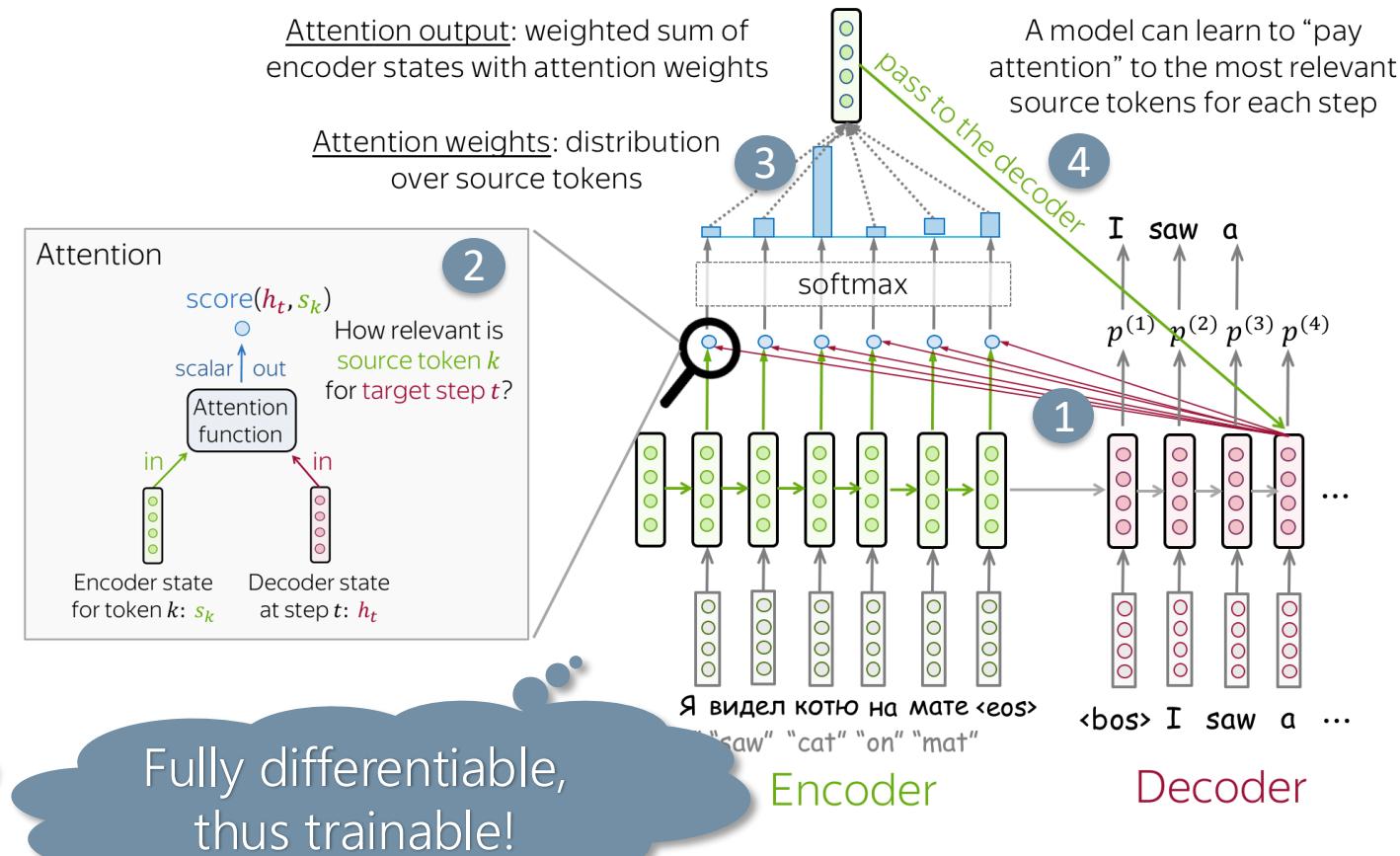


You Need Attention!

Decoder uses attention to decide which source parts are more important



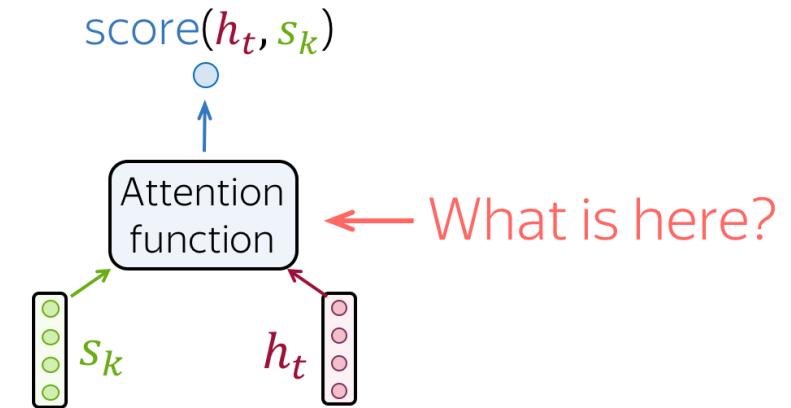
Attention scores can be computed in different ways



Attention Scores

Different mechanisms to compute attention scores have been proposed:

- Simple dot-product
- Bilinear function (aka "Luong attention")
- Multi-layer perceptron (aka "Bahdanau attention")



Dot-product

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

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

Bilinear

$$\begin{bmatrix} h_t^T \\ \vdots \end{bmatrix} \times \begin{bmatrix} W \\ \vdots \end{bmatrix} \times \begin{bmatrix} \textcolor{green}{\circ} \\ \vdots \\ \textcolor{green}{\circ} \end{bmatrix} s_k$$

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

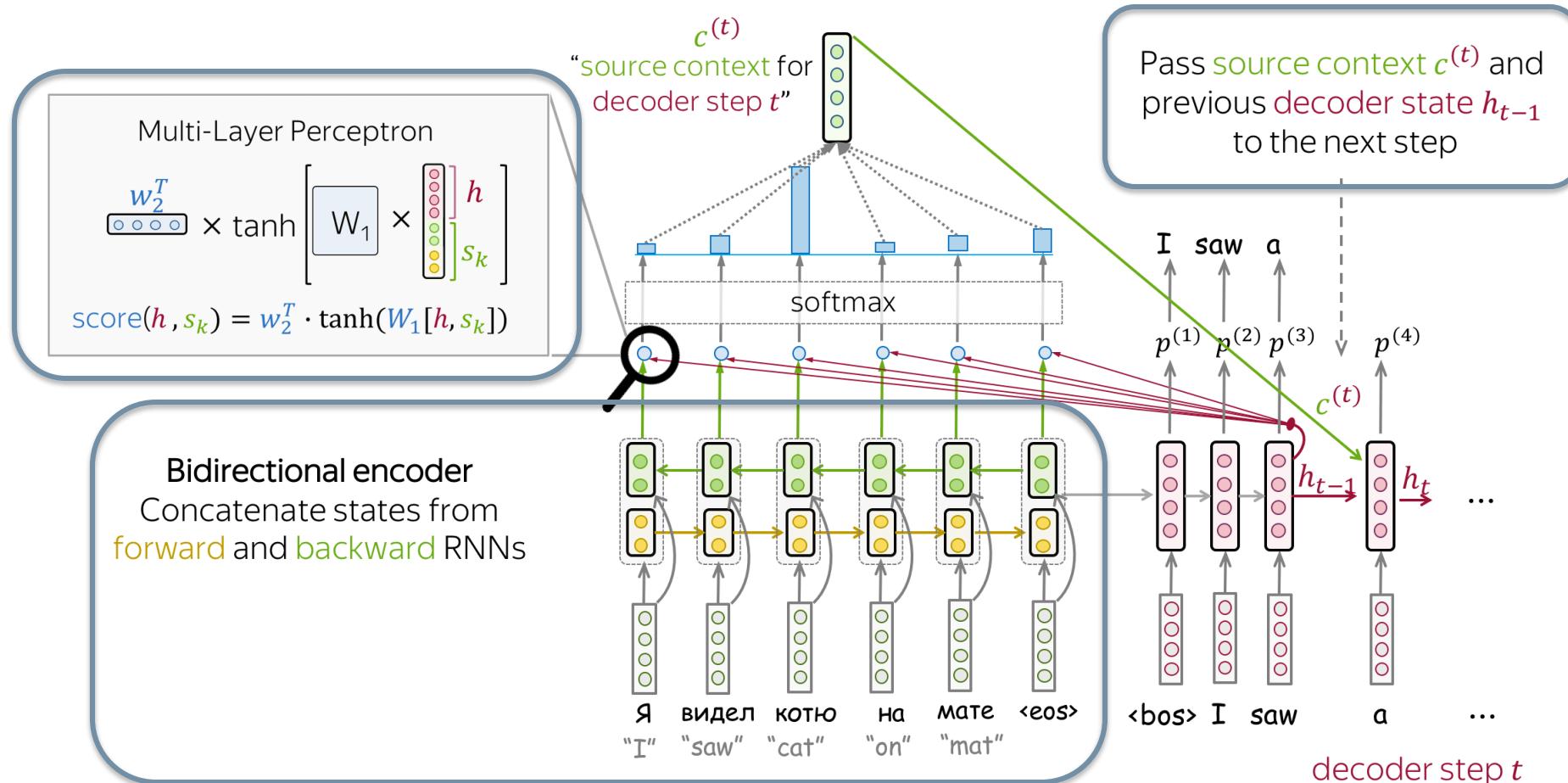
Multi-Layer Perceptron

$$\begin{bmatrix} w_2^T \\ \vdots \\ \textcolor{blue}{\circ} \end{bmatrix} \times \tanh \left[\begin{bmatrix} W_1 \\ \vdots \end{bmatrix} \times \begin{bmatrix} \textcolor{red}{\circ} \\ \vdots \\ \textcolor{red}{\circ} \end{bmatrix} h_t \right] \times \begin{bmatrix} \textcolor{green}{\circ} \\ \vdots \\ \textcolor{green}{\circ} \end{bmatrix} s_k$$

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

Bahdanau Attention Model

Proposed as part of the original Bahdanau model

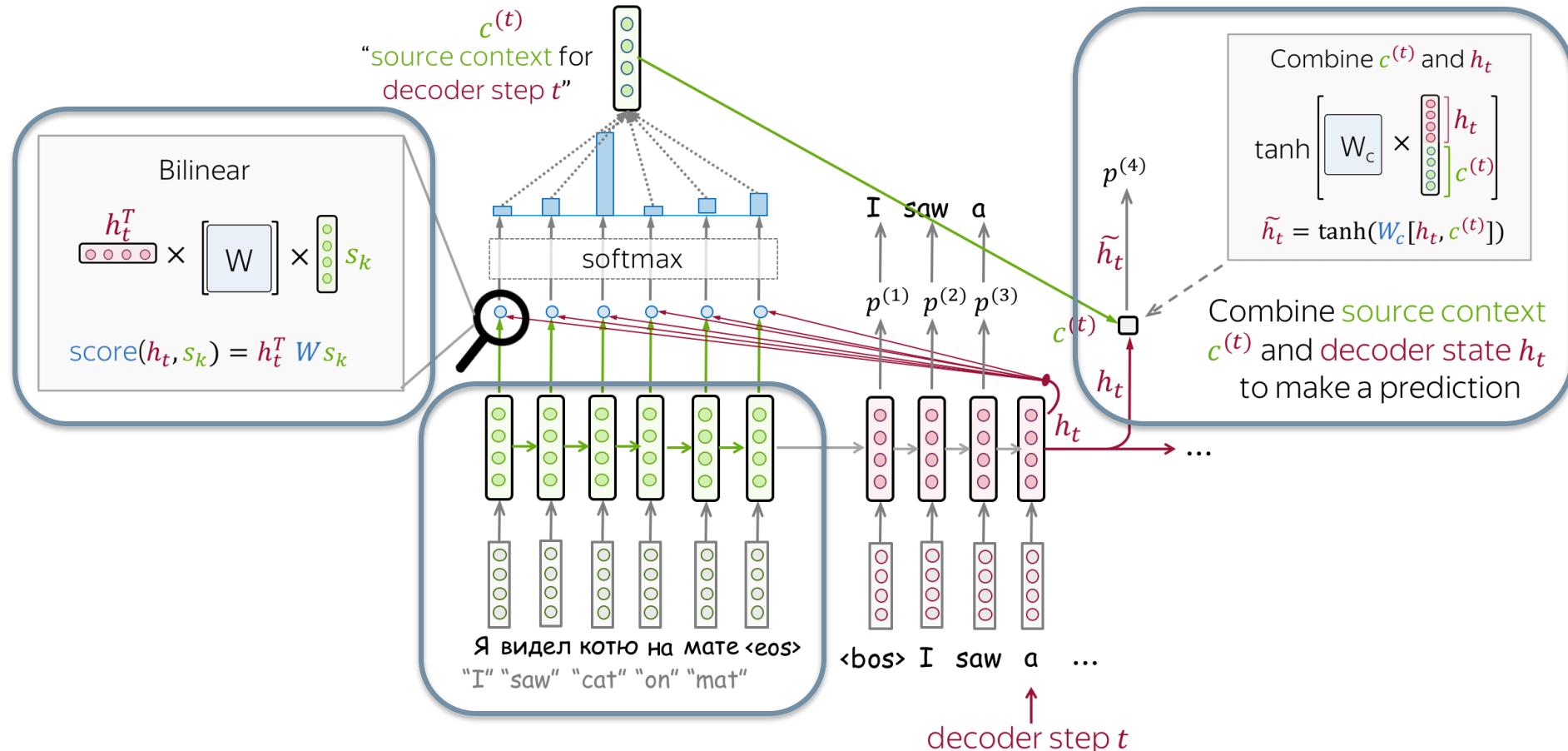


Neural Machine Translation by Jointly Learning to Align and Translate: <https://arxiv.org/pdf/1409.0473.pdf>



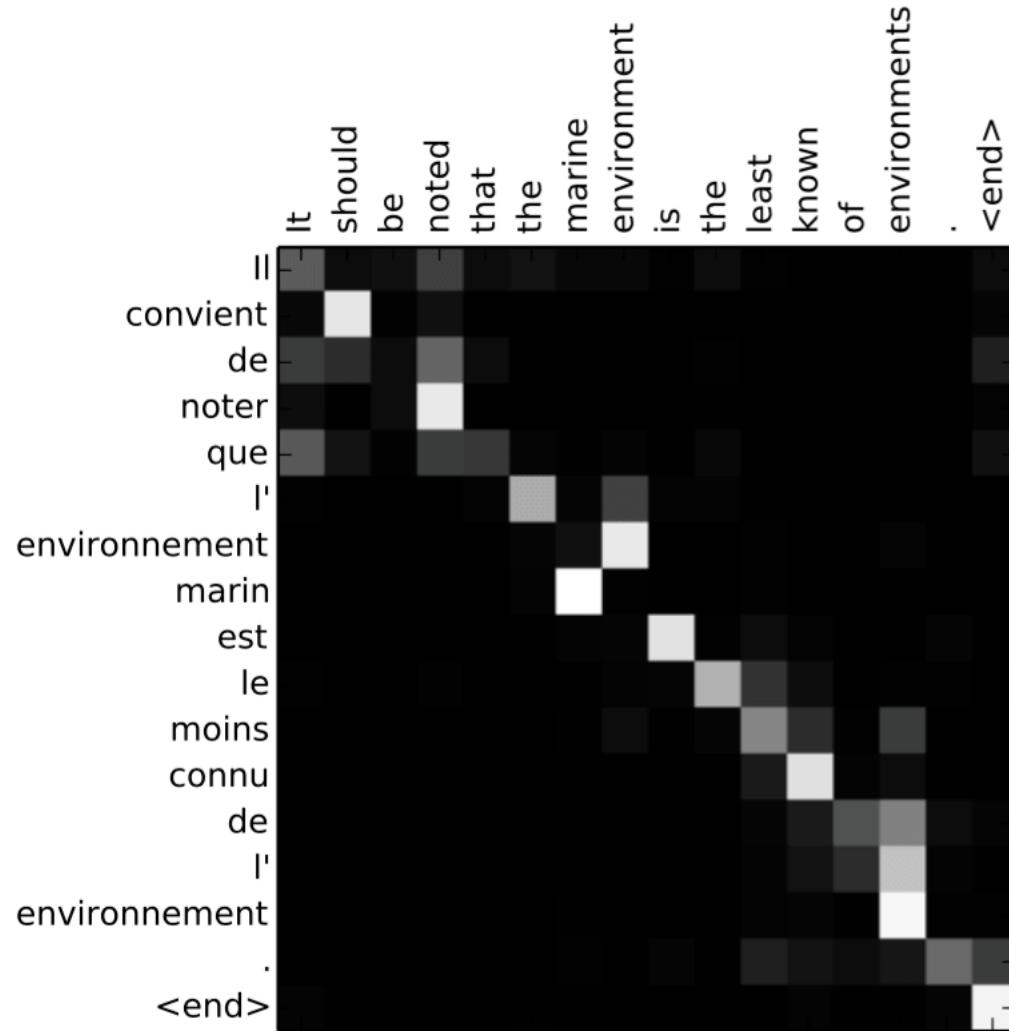
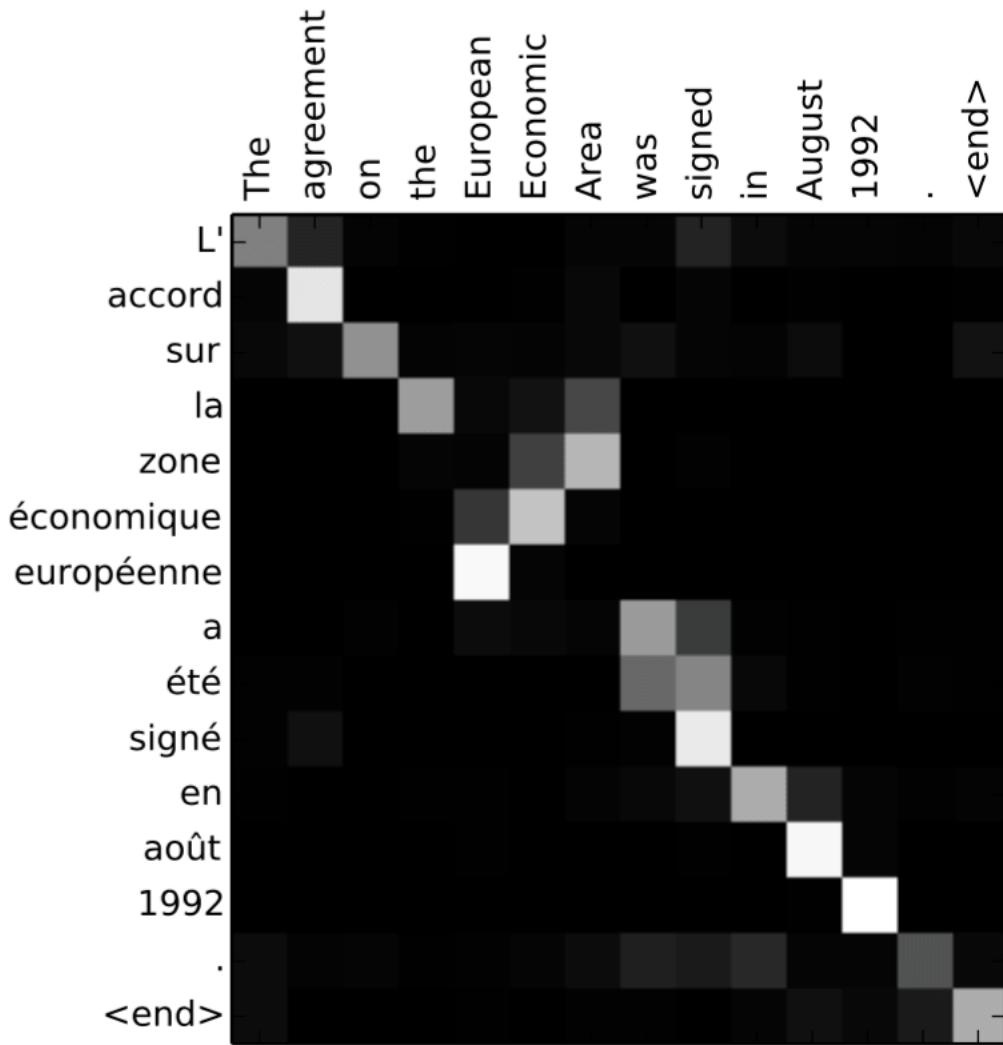
Luong Attention Model

Proposed as part of the original Luong model



Effective Approaches to Attention-based Neural Machine Translation <https://arxiv.org/abs/1508.04025>

Attention learns soft sentence alignment ...



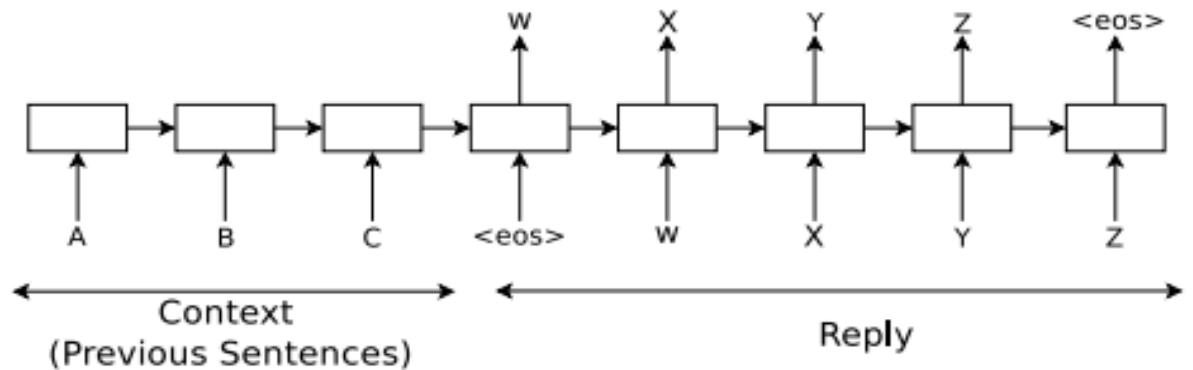
Neural Machine Translation by Jointly Learning to Align and Translate: <https://arxiv.org/pdf/1409.0473.pdf>



Generative Chatbots

We can directly apply sequence to sequence models to the conversation between two agents:

- First person utters "ABC"
- Second person replies "WXYZ"



Generative chatbots use an RNN and train it to map "ABC" to "WXYZ":

- We can borrow the model from machine translation
- A flat model simple and general
- Attention mechanisms apply as usual

A Neural Conversational Model <https://arxiv.org/pdf/1506.05869.pdf>

Chatbots Response Generation

Chatbots can be defined along at least two dimensions, *core algorithm* and *context handling*:

- Generative: encode the question into a context vector and generate the answer word by word using conditioned probability distribution over answer's vocabulary. E.g., an encoder-decoder model.
- Retrieval: rely on knowledge base of question-answer pairs. When a new question comes in, inference phase encodes it in a context vector and by using similarity measure retrieves the top-k neighbor knowledge base items.



Chatbots Response Generation

Chatbots can be defined along at least two dimensions, *core algorithm* and *context handling*:

- Single-turn: build the input vector by considering the incoming question. They may lose important information about the history of the conversation and generate irrelevant responses.

$$\{(q_i, a_i)\}$$

- Multi-turn: the input vector is built by considering a multi-turn conversational context, containing also incoming question.

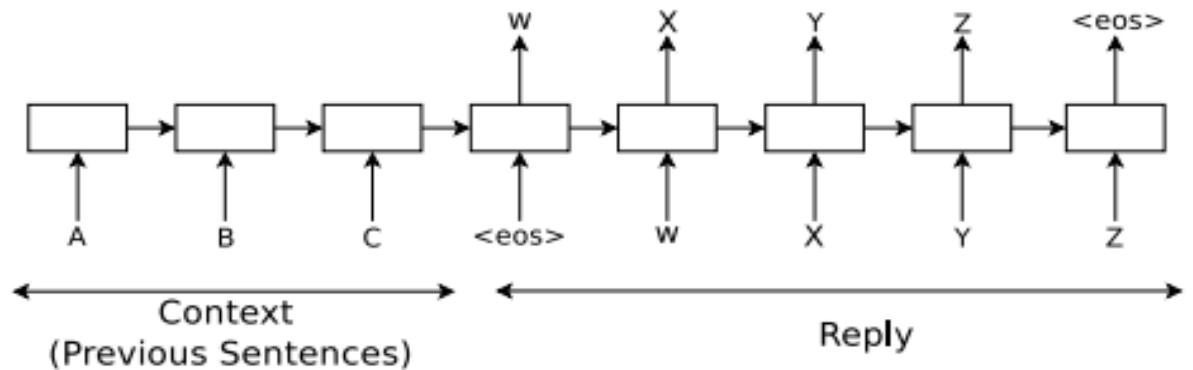
$$\{([q_{i-2}; a_{i-2}; q_{i-1}; a_{i-1}; q_i], a_i)\}$$



Generative Chatbots

We can directly apply sequence to sequence models to the conversation between two agents:

- First person utters "ABC"
- Second person replies "WXYZ"



Generative chatbots use an RNN and train it to map "ABC" to "WXYZ":

- We can borrow the model from machine translation
- A flat model simple and general
- Attention mechanisms apply as usual

How do we handle
multi turns chat?

A Neural Conversational Model <https://arxiv.org/pdf/1506.05869.pdf>

Generative Hierarchical Chatbots

We could concatenate multiple turns into a single long input sequence, however, this probably results in poor performances.

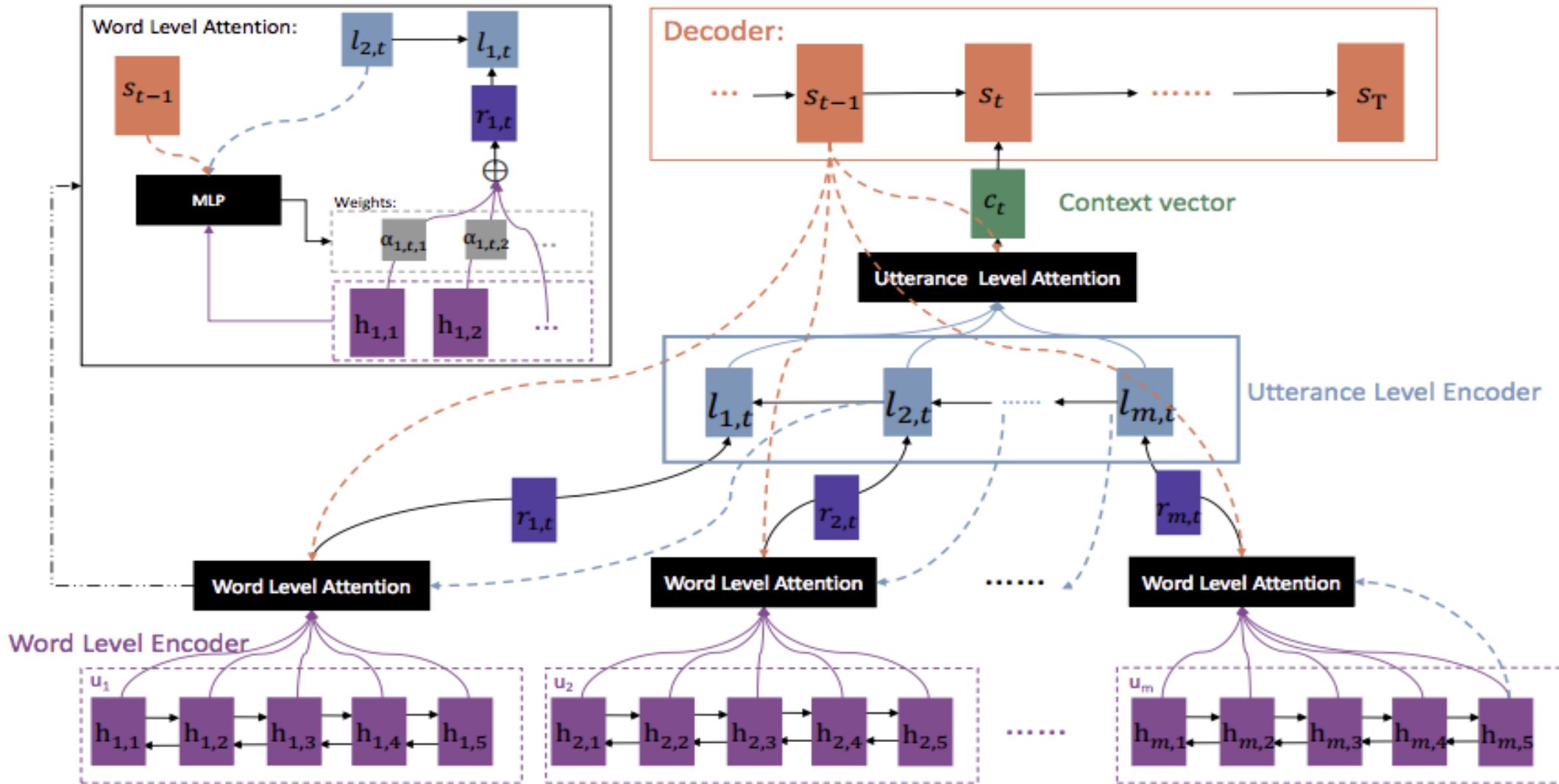
- LSTM cells often fail to catch longterm dependencies within input sequences that are longer than 100 tokens
- No explicit representation of turns can be exploited by attention mechanism

Xing et al., in 2017, extended attention mechanism from single-turn response generation to a hierarchical attention mechanism

- Hierarchical attention networks (e.g., characters -> words -> sentences)
- Generate hidden representation of a sequence from contextualized words

Hierarchical Recurrent Attention Network for Response Generation <https://arxiv.org/pdf/1701.07149.pdf>

Hierarchical Generative Multi-turn Chatbots



Hierarchical Recurrent Attention Network for Response Generation <https://arxiv.org/pdf/1701.07149.pdf>

Hierarchical Generative Multi-turn Chatbots

We can visualize hierarchical attention; darker color means more important words or utterances.



Hierarchical Recurrent Attention Network for Response Generation <https://arxiv.org/pdf/1701.07149.pdf>



Hierarchical Document Classification

Hierarchical attention networks have been used for topic classification too

- Left document denotes Science and Mathematics; model accurately localizes the words zebra, stripes, camouflage, predator and corresponding sentences.
- Right document denotes Computers and Internet; the model focuses on web, searches, browsers and their corresponding sentences.

GT: 1 Prediction: 1

why does zebras have stripes ?
what is the purpose or those stripes ?
who do they serve the zebras in the
wild life ?
this provides camouflage - predator
vision is such that it is usually difficult
for them to see complex patterns

GT: 4 Prediction: 4

how do i get rid of all the old web
searches i have on my web browser ?
i want to clean up my web browser
go to tools > options .
then click “ delete history ” and “
clean up temporary internet files . ”

Hierarchical Recurrent Attention Network for Response Generation <https://arxiv.org/pdf/1701.07149.pdf>



Hierarchical Document Classification

In Sentiment Analysis, the model can select words carrying strong sentiment like delicious, amazing, terrible and corresponding sentences.

GT: 4 Prediction: 4

pork belly = delicious .
scallops ?
i do n't .
even .
like .
scallops , and these were a-m-a-z-i-n-g .
fun and tasty cocktails .
next time i 'm in phoenix , i will go
back here .
highly recommend .

GT: 0 Prediction: 0

terrible value .
ordered pasta entree .
. \$ 16.95 good taste but size was an
appetizer size .
. no salad , no bread no vegetable .
this was .
our and tasty cocktails .
our second visit .
i will not go back .

Beyond Recurrent Neural Networks

NLP community believed **LSTMs with attention** could yield state-of-art performance on any task. But some limits were preventing this ...

Because of using LSTMs (and any Recurrent Neural Network):

- Performing inference (and training) is sequential in nature
- Parallelization at sample level is precluded by recurrence sequential nature
- Parallelization can happen at level of batch only
- Memory constraints limit batching across to many examples
- This becomes critical at longer sequence lengths ...

Here it comes another bottleneck in sequence-to-sequence modeling!



Necessity is literally the mother of invention.

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez*†
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin*‡
illia.polosukhin@gmail.com

NIPS 2017

Attention is all you need!

Google proposes to speed up training by replacing RNN (sequential in nature) with attention mechanism (parallel in nature)

	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

At each level we look at the entire sequence

Encoder

Who is doing:

- all source tokens

What they are doing:

- look at each other
- update representations

repeat N times

This happens within
prefix tokens ...

Decoder

Who is doing:

- target token at the current step

What they are doing:

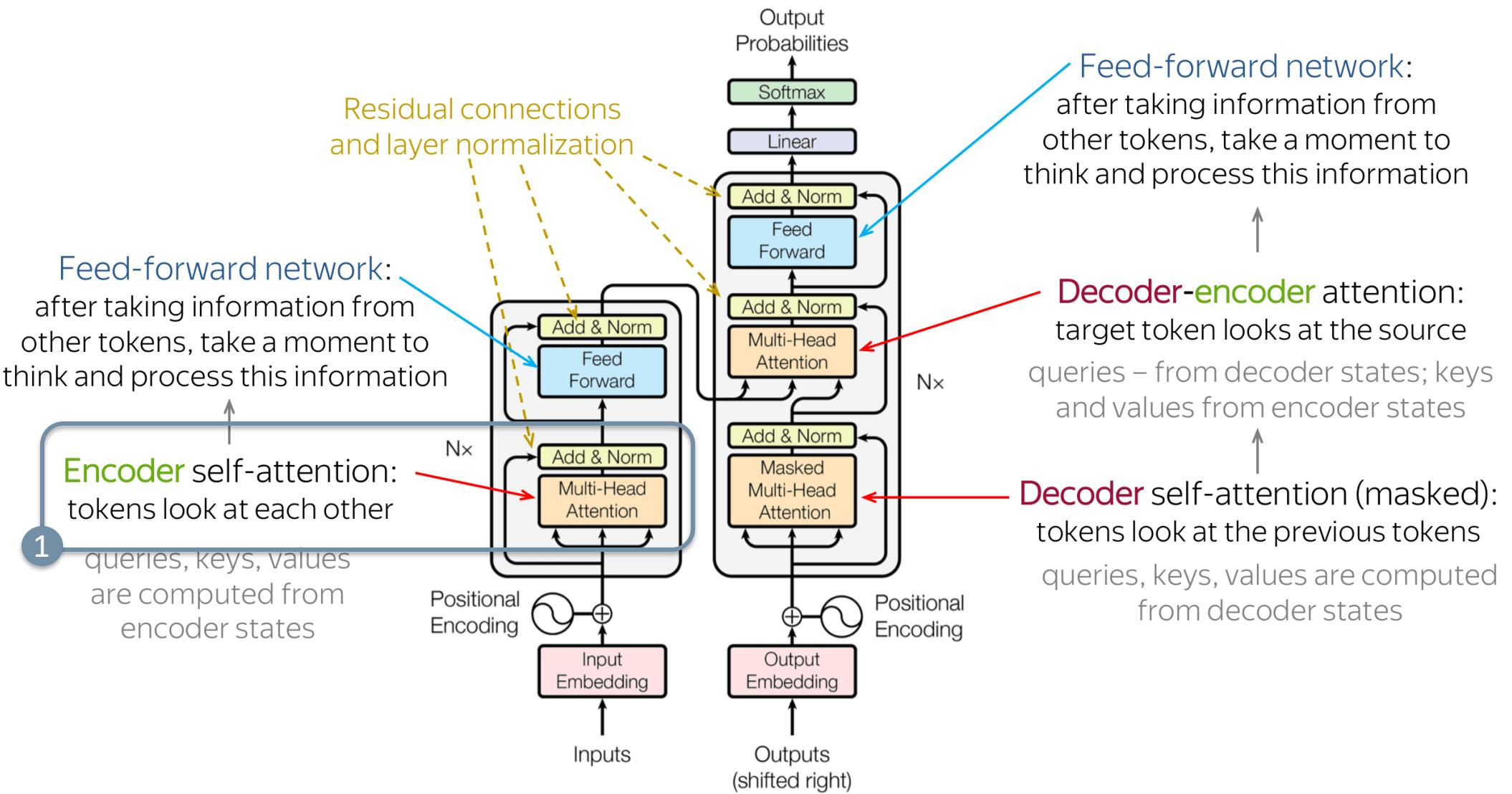
- looks at previous target tokens
- looks at source representations
- update representation

repeat N times

Transformer: A Novel Neural Network Architecture for Language Understanding <https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>



Attention is all you need!



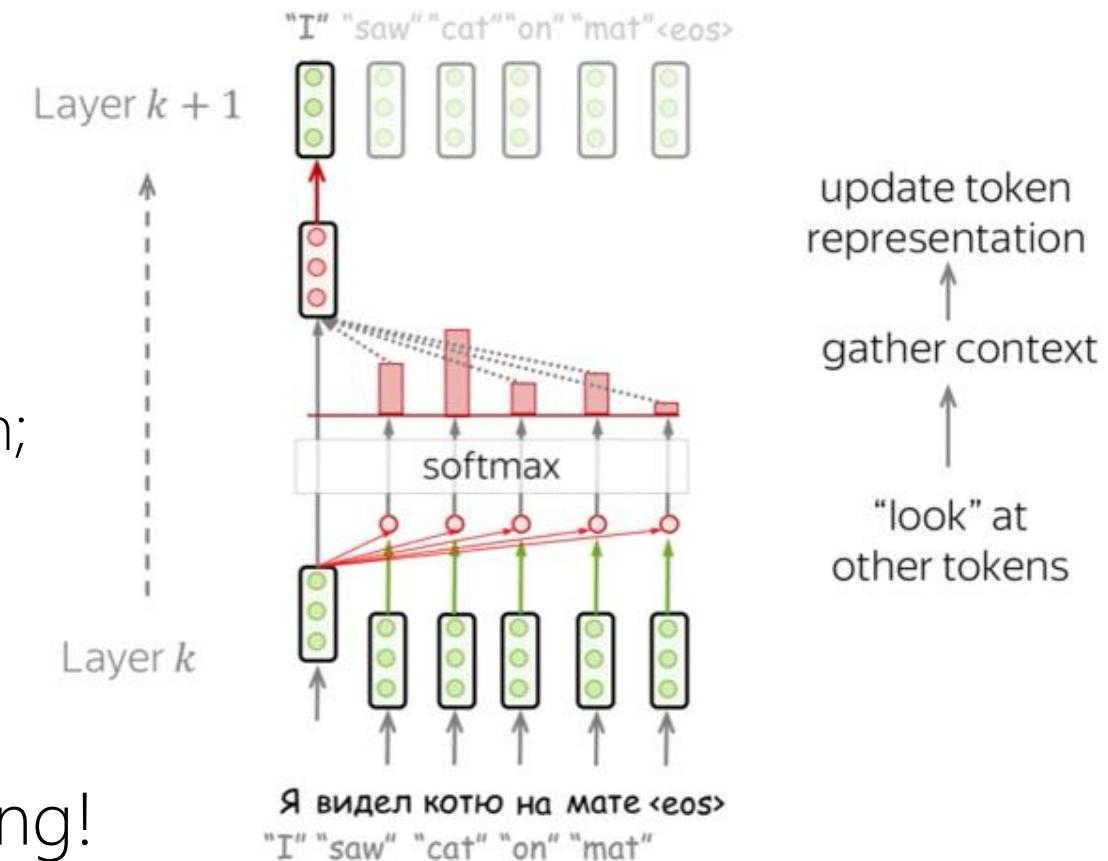
Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

The Self-Attention Idea

Self-attention operates between representations of the same nature, e.g., all encoder states in some layer.

This is implemented via:

- Query - asking for information;
- Key - saying that it has some information;
- Value - giving the information



The use of *Query*, *Key* and *Value* allows parallel execution and thus parallel training!

Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

Query, Key, Value ...

$$[W_Q] \times [\text{green}] = [\text{blue}]$$

Query: vector from which the attention is looking

"Hey there, do you have this information?"

$$[W_K] \times [\text{green}] = [\text{yellow}]$$

Key: vector at which the query looks to compute weights

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

$$[W_V] \times [\text{green}] = [\text{red}]$$

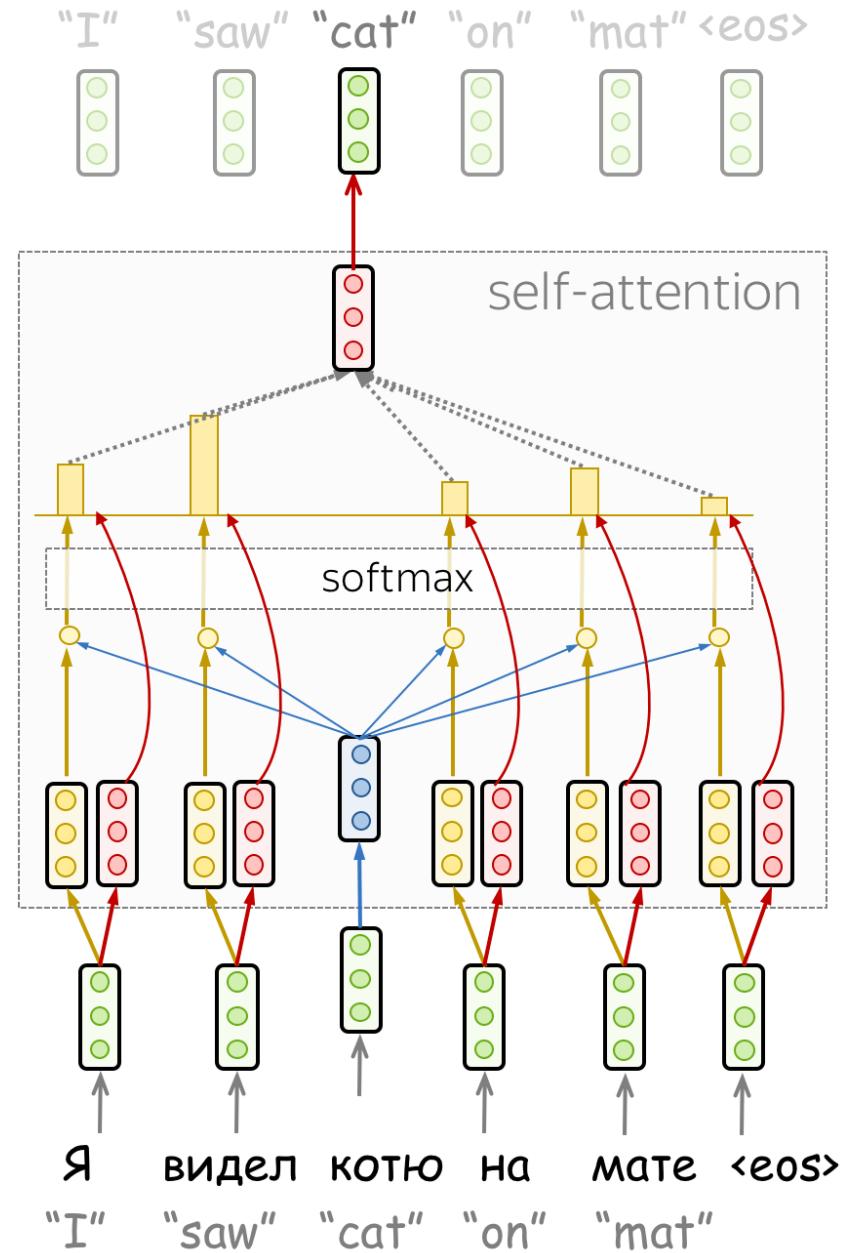
Value: their weighted sum is attention output

"Here's the information I have!"

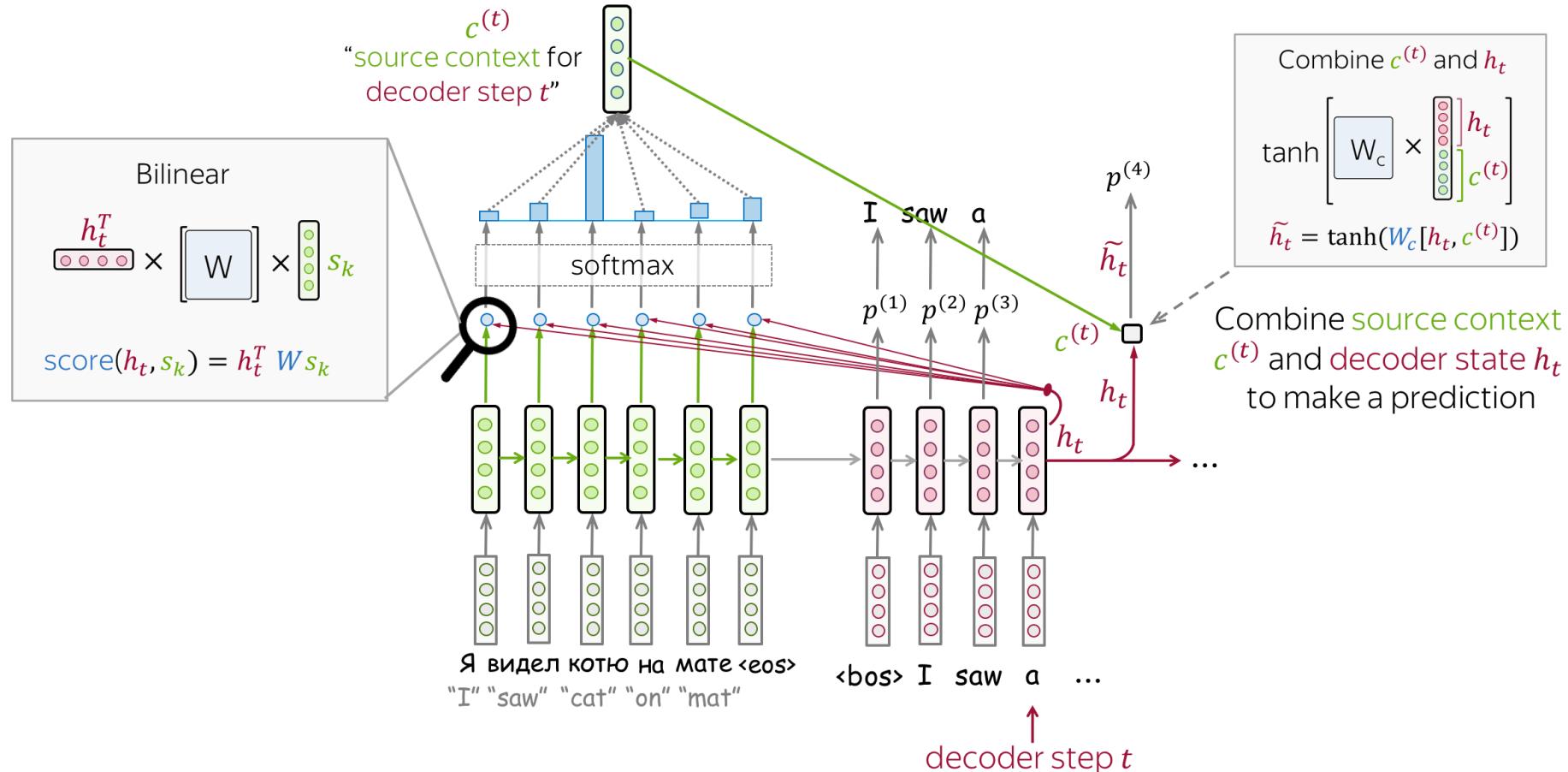
$$\text{Attention}(q, k, v) = \frac{\text{Attention weights}}{\text{softmax}\left(\frac{qk^T}{\sqrt{d_k}}\right)v}$$

from to

vector dimensionality of K, V

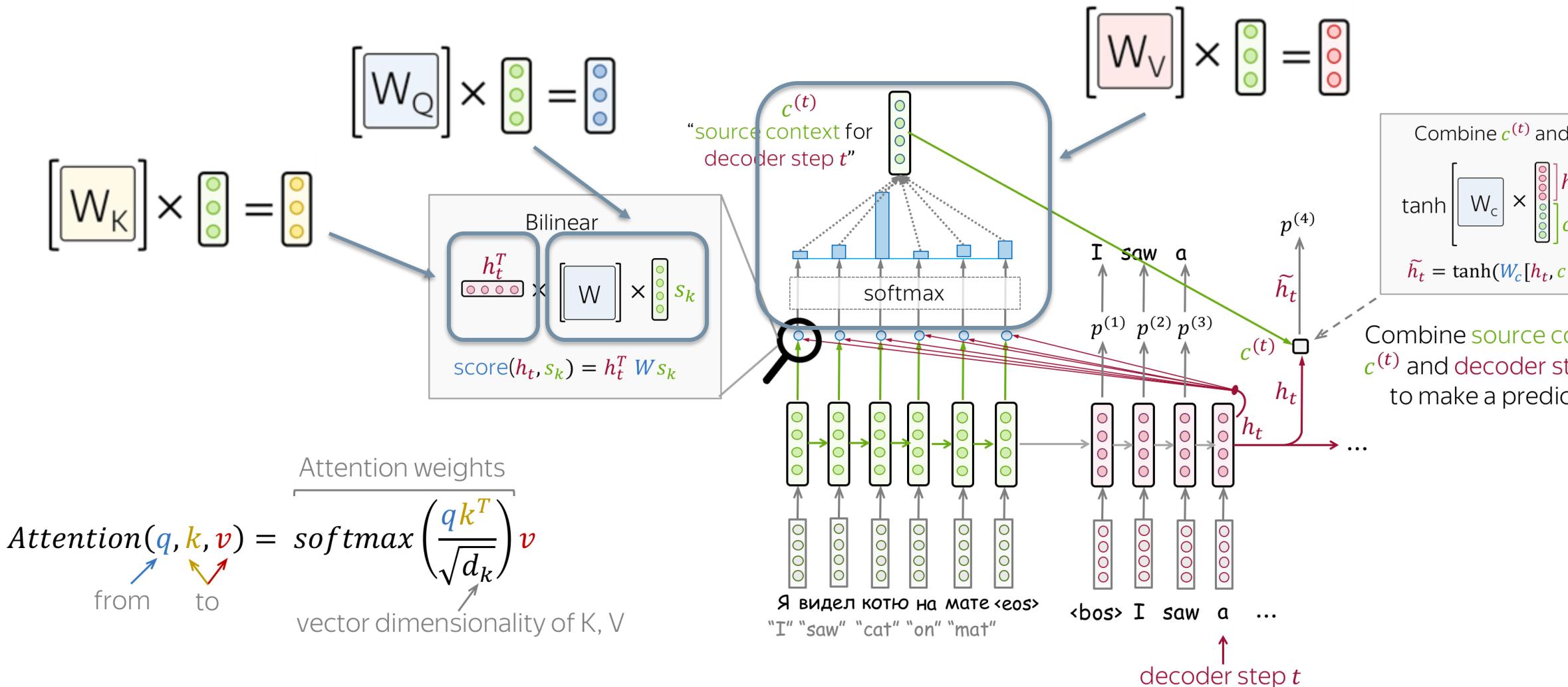


Recall Luong Attention Model



Effective Approaches to Attention-based Neural Machine Translation <https://arxiv.org/abs/1508.04025>

Recall Luong Attention Model



Effective Approaches to Attention-based Neural Machine Translation <https://arxiv.org/abs/1508.04025>

Let's Play Some Linear Algebra!

$$\begin{array}{c}
 x_1 \begin{matrix} \text{light green} \\ \text{light green} \end{matrix} \times W_Q = q_1 \begin{matrix} \text{light blue} \\ \text{light blue} \end{matrix} \\
 x_2 \begin{matrix} \text{light green} \\ \text{light green} \end{matrix} \times W_Q = q_2 \begin{matrix} \text{light blue} \\ \text{light blue} \end{matrix}
 \end{array}$$

$$\begin{array}{c}
 x_1 \begin{matrix} \text{light green} \\ \text{light green} \end{matrix} \times W_K = k_1 \begin{matrix} \text{yellow} \\ \text{yellow} \end{matrix} \\
 x_2 \begin{matrix} \text{light green} \\ \text{light green} \end{matrix} \times W_K = k_2 \begin{matrix} \text{yellow} \\ \text{yellow} \end{matrix}
 \end{array}$$

$$\begin{array}{c}
 x_1 \begin{matrix} \text{light green} \\ \text{light green} \end{matrix} \times W_V = v_1 \begin{matrix} \text{pink} \\ \text{pink} \end{matrix} \\
 x_2 \begin{matrix} \text{light green} \\ \text{light green} \end{matrix} \times W_V = v_2 \begin{matrix} \text{pink} \\ \text{pink} \end{matrix}
 \end{array}$$

Input
Embeddings

Queries

Keys

Scores

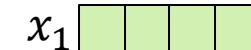
Normalize

Softmax

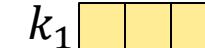
Values

Sum

Thinking

x_1 

q_1 

k_1 

$q_1 \cdot k_1 = 112$

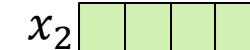
$112/\sqrt{64} = 14$

0.88

v_1 

z_1 

Machines

x_2 

q_2 

k_2 

$q_1 \cdot k_2 = 96$

$96/\sqrt{64} = 12$

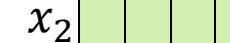
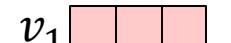
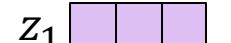
0.12

v_2 



Let's Play Some Linear Algebra!

$$\begin{array}{c} x_1 \begin{matrix} \text{Light Green} \\ \text{Light Green} \end{matrix} \times W_Q = q_1 \begin{matrix} \text{Light Blue} \\ \text{Light Blue} \end{matrix} \\ x_2 \begin{matrix} \text{Light Green} \\ \text{Light Green} \end{matrix} \end{array}$$
$$\begin{array}{c} x_1 \begin{matrix} \text{Light Green} \\ \text{Light Green} \end{matrix} \times W_K = k_1 \begin{matrix} \text{Yellow} \\ \text{Yellow} \end{matrix} \\ x_2 \begin{matrix} \text{Light Green} \\ \text{Light Green} \end{matrix} \end{array}$$
$$\begin{array}{c} x_1 \begin{matrix} \text{Light Green} \\ \text{Light Green} \end{matrix} \times W_V = v_1 \begin{matrix} \text{Red} \\ \text{Red} \end{matrix} \\ x_2 \begin{matrix} \text{Light Green} \\ \text{Light Green} \end{matrix} \end{array}$$


Input Embeddings	x_1 	Thinking	x_1 	Machines
Queries	q_1 		q_2 	
Keys		k_1 		k_2 
Scores		$q_2 \cdot k_1 = 32$		$q_2 \cdot k_2 = 64$
Normalize		$32/\sqrt{64} = 4$		$64/\sqrt{64} = 8$
Softmax		0.02		0.98
Values	v_1 		v_2 	
Sum	z_1 		z_2 	

The Illustrated Transformer <http://jalammar.github.io/illustrated-transformer/>



Let's Play Some Linear Algebra!

$$\begin{array}{c} x_1 \begin{matrix} \text{light green} \\ \text{light green} \end{matrix} \times \begin{matrix} W_Q \\ \text{light blue} \end{matrix} = q_1 \begin{matrix} \text{blue} \\ \text{blue} \end{matrix} \\ x_2 \begin{matrix} \text{light green} \\ \text{light green} \end{matrix} \times \begin{matrix} W_K \\ \text{yellow} \end{matrix} = k_1 \begin{matrix} \text{yellow} \\ \text{yellow} \end{matrix} \\ x_1 \begin{matrix} \text{light green} \\ \text{light green} \end{matrix} \times \begin{matrix} W_V \\ \text{pink} \end{matrix} = v_1 \begin{matrix} \text{pink} \\ \text{pink} \end{matrix} \\ x_2 \begin{matrix} \text{light green} \\ \text{light green} \end{matrix} \times \begin{matrix} W_V \\ \text{pink} \end{matrix} = v_2 \begin{matrix} \text{pink} \\ \text{pink} \end{matrix} \end{array}$$

→

$$softmax\left(\frac{\begin{matrix} q_1 & k_1 \\ q_2 & k_2 \end{matrix} \times \begin{matrix} \text{blue} & \text{yellow} \\ \text{blue} & \text{yellow} \end{matrix}}{\sqrt{d_{model}}}\right) \times \begin{matrix} \text{pink} \\ \text{pink} \end{matrix} = \begin{matrix} \text{purple} \\ \text{purple} \end{matrix}$$

The Illustrated Transformer <http://jalammar.github.io/illustrated-transformer/>



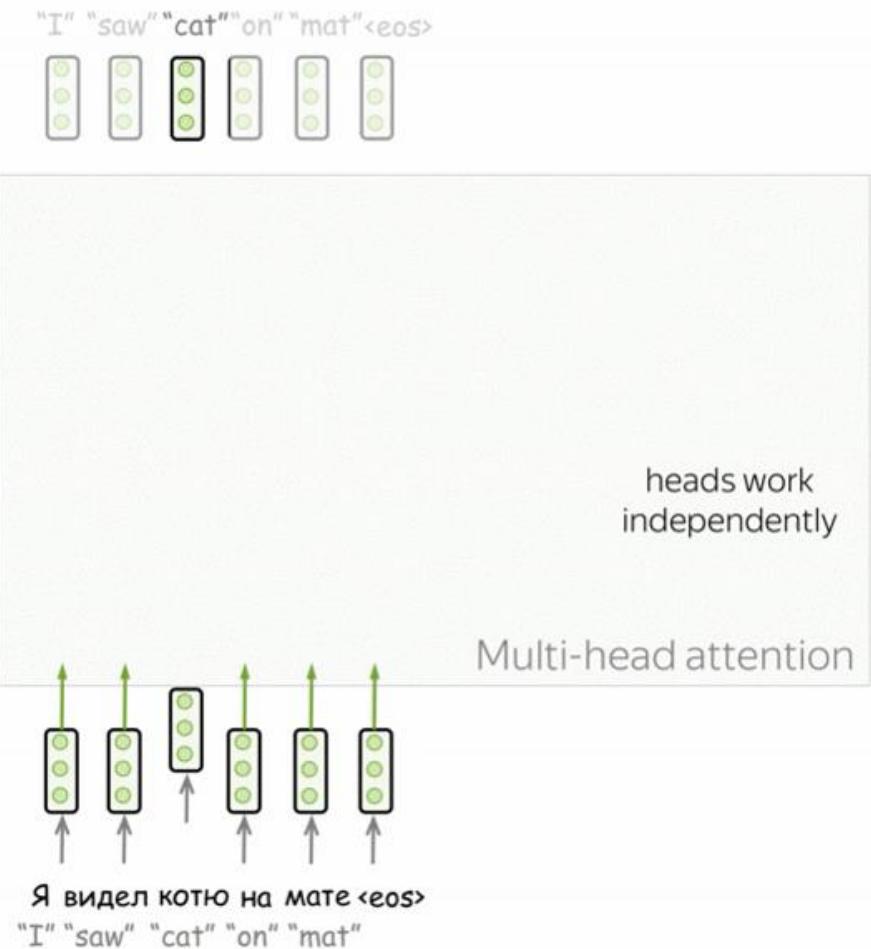
Multi-Head Attention

Attention defines the role of a word in a sentence. This, in turn, might be related to different aspects such as:

- verb inflection wrt subject in terms of gender
- verb inflection wrt subjects in terms of number
- case of objects defined by verbs
- ...

Multiple head attentions allow the model to focus on different things, both at encoding and decoding time.

Check Mark Carman
NLP lectures for this!



Multi-Head Attention

Attention defines the role of a word in a sentence. This, in turn, might be related to different aspects such as:

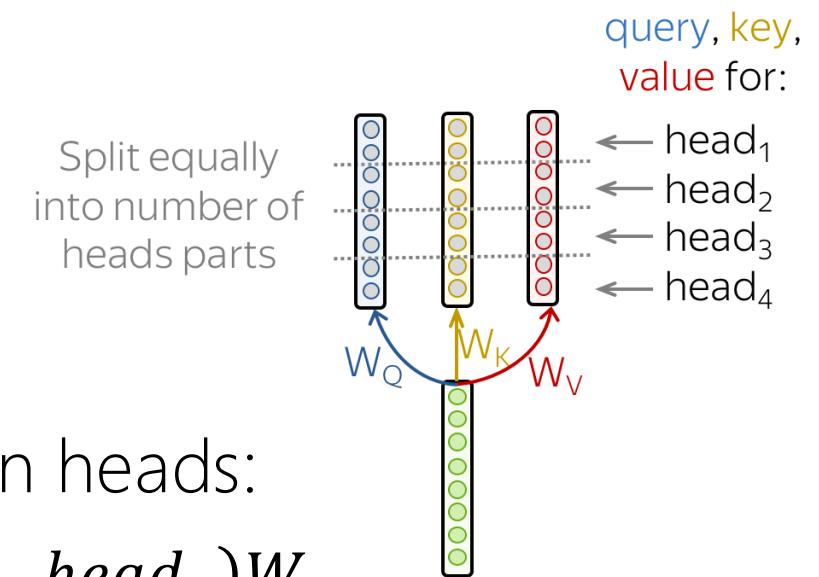
- verb inflection wrt subject in terms of gender
- verb inflection wrt subjects in terms of number
- case of objects defined by verbs

Implemented as concatenation of several attention heads:

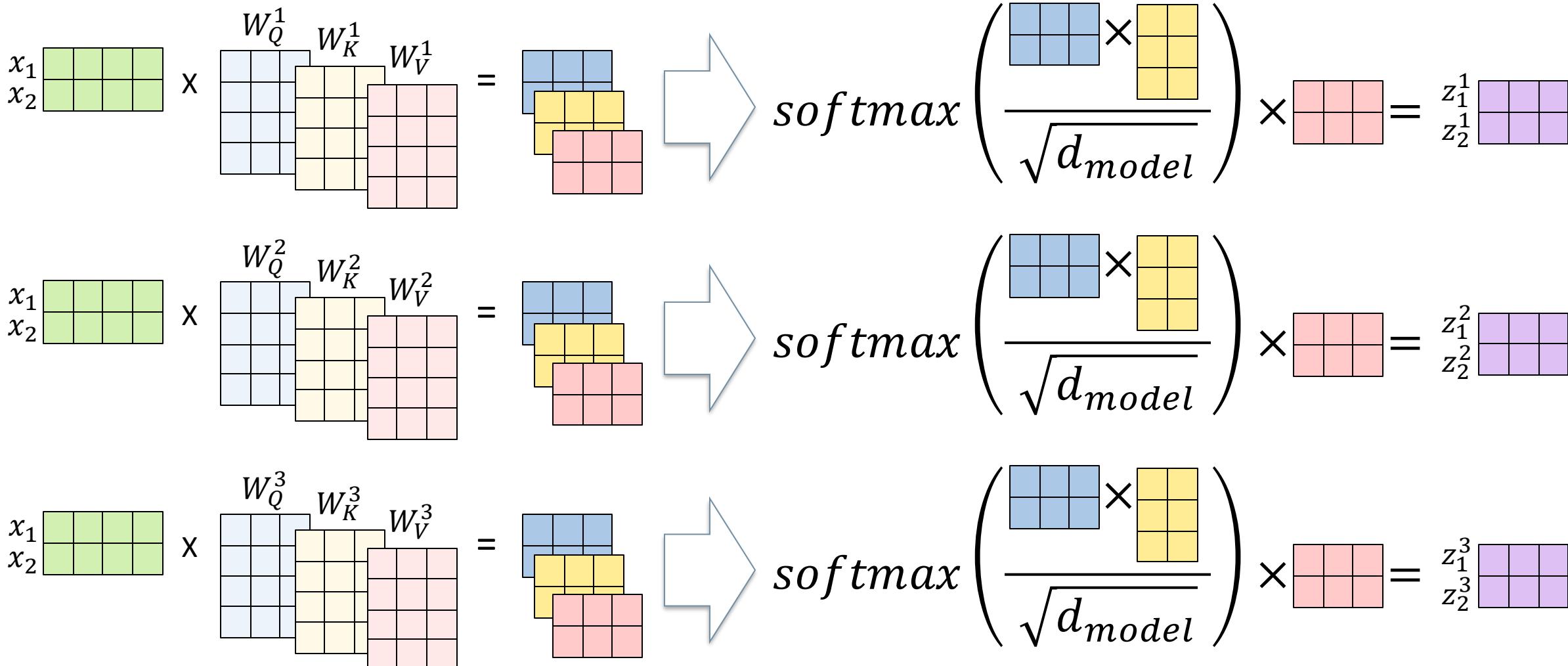
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_n)W_o$$

$$\text{head}_1 = \text{Attention}(QW_Q^i, VW_V^i, VW_V^i)$$

this way, models with one or several attention heads have the same size (i.e., model size does not increase with number of heads)

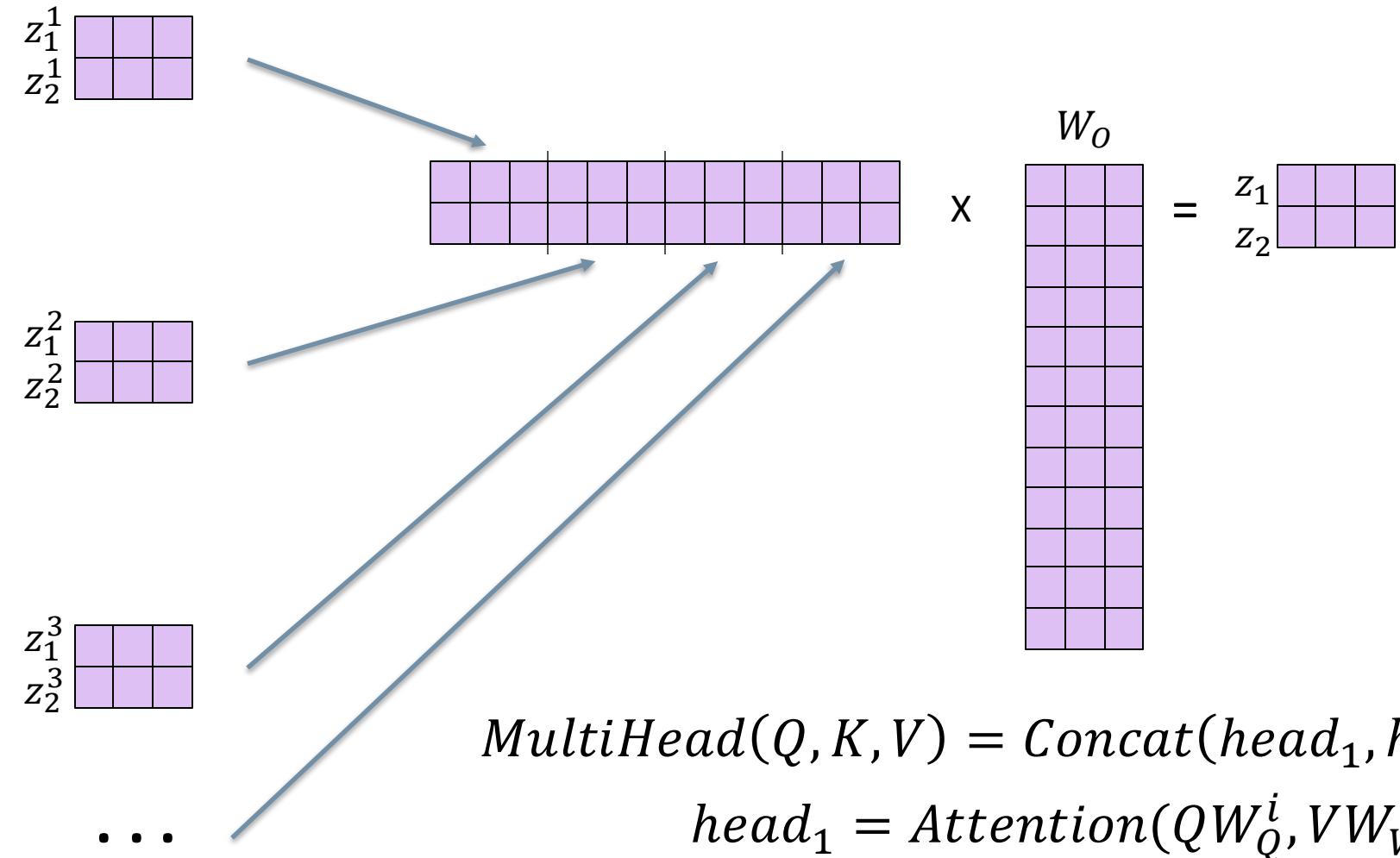


Let's Play Some Linear Algebra!

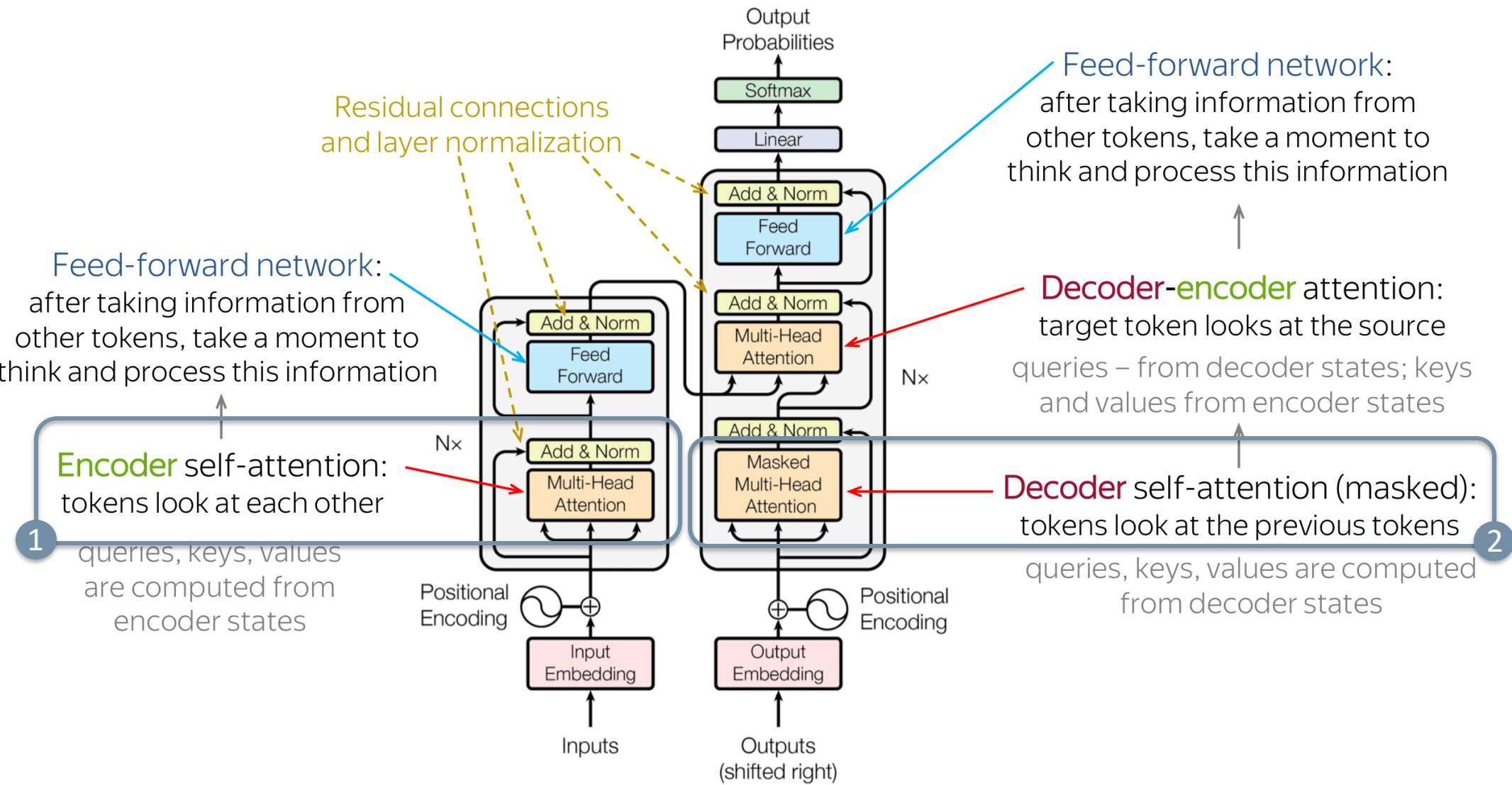


...

Let's Play Some Linear Algebra!



Attention is all you need!



Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

Masked Self-Attention

In the decoder, attention mechanism works differently at training and inference time as we should not “look ahead”:

- At inference time, we generate one token at the time as we do not know the length of the sequence (no “look-ahead” problem)
- At training time, we know already the entire output sequence and we want to process it in parallel (“look ahead” problem)



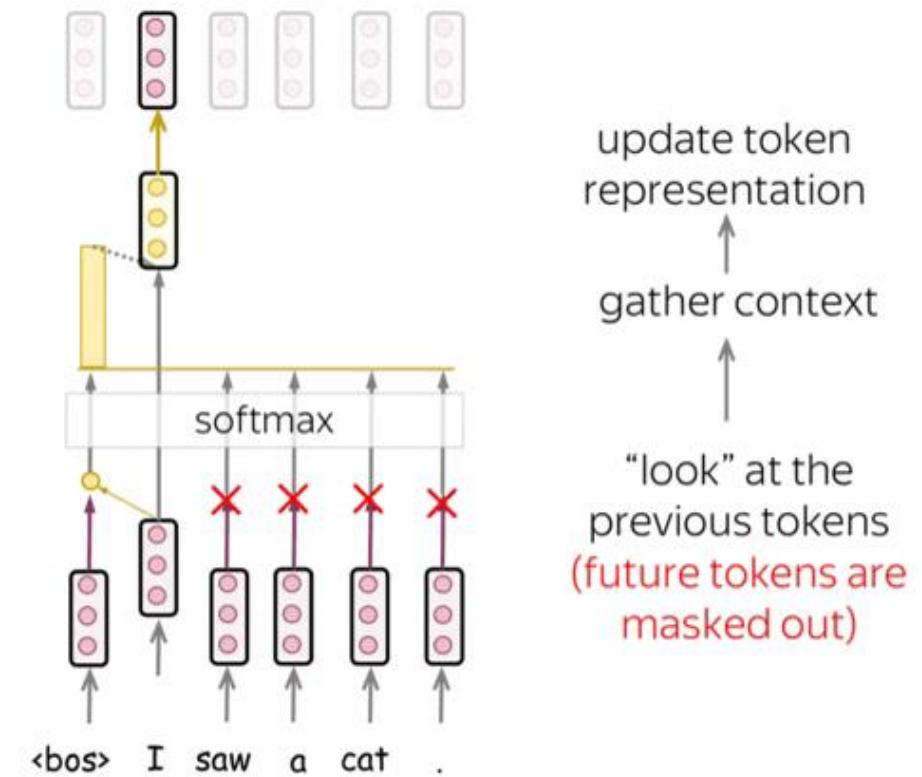
This is obtained by
masking «future tokens»
at training time

Masked Self-Attention

In the decoder, attention mechanism works differently at training and inference time as we should not “look ahead”:

- At inference time, we generate one token at the time as we do not know the length of the sequence (no “look-ahead” problem)
- At training time, we know already the entire output sequence and we want to process it in parallel (“look ahead” problem)

This is obtained by
masking «future tokens»
at training time



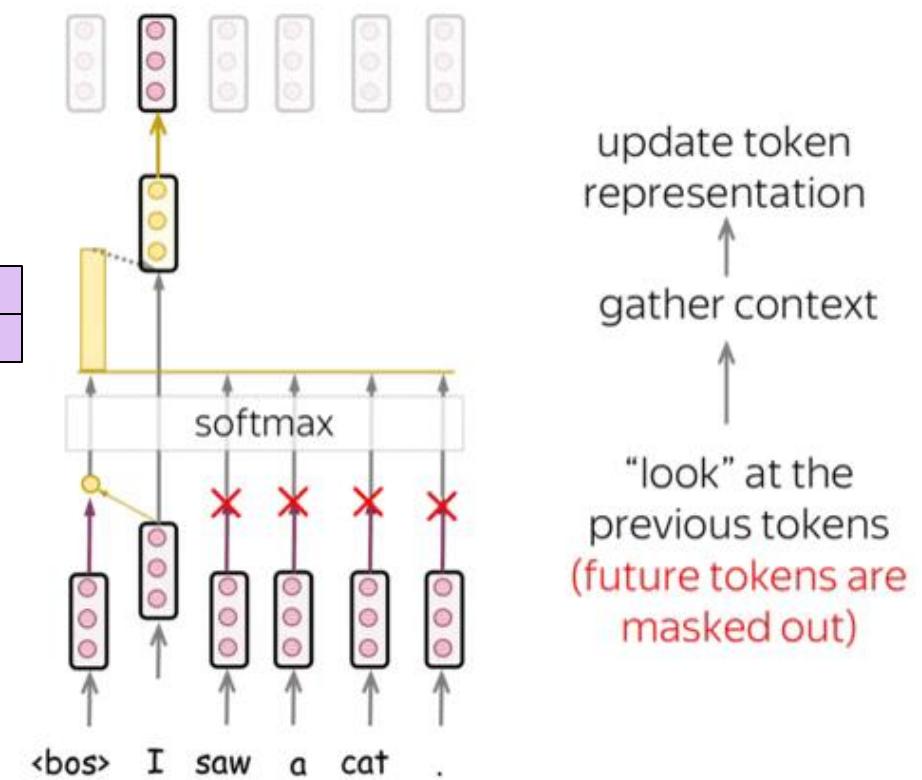
Masked Self-Attention

In the decoder, attention mechanism works differently at training and inference time as we should not “look ahead”:

$$\text{softmax} \left(\frac{\begin{pmatrix} \text{blue} & \times & \text{yellow} \\ \text{blue} & + & \text{grey} \end{pmatrix}}{\sqrt{d_{\text{model}}}} \right) \times \text{pink} = \text{purple}$$

$$\text{Mask} = \begin{bmatrix} 0 & -\infty \\ 0 & 0 \end{bmatrix}$$

This is obtained by
masking «future tokens»
at training time

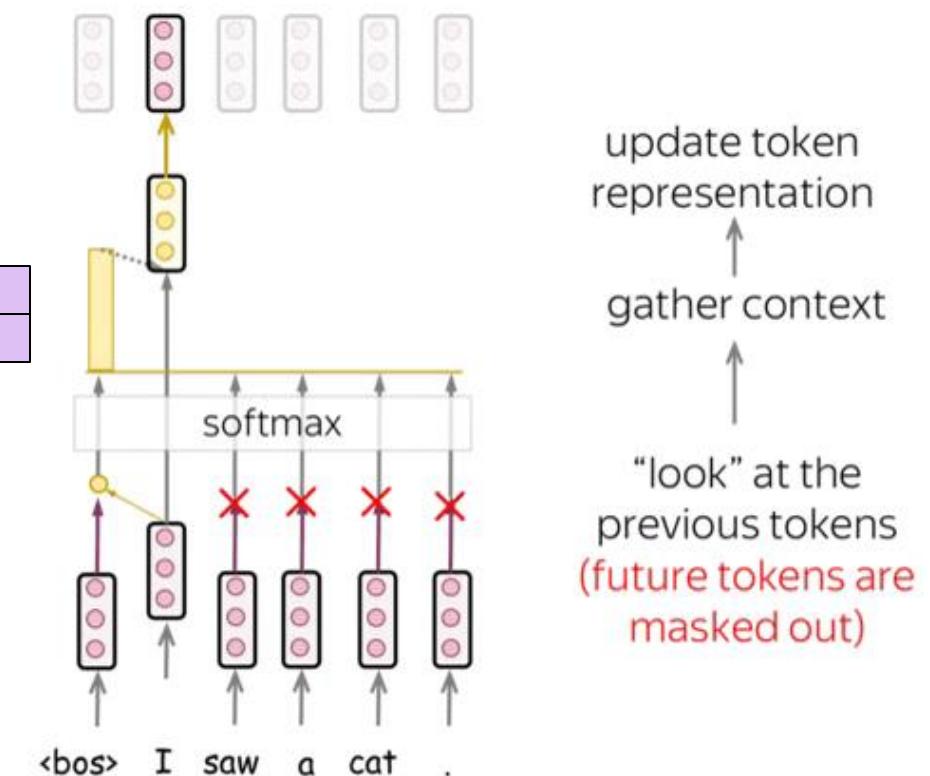


Masked Self-Attention

In the decoder, attention mechanism works differently at training and inference time as we should not "look ahead":

$$\text{softmax} \left(\frac{\begin{pmatrix} \text{blue} & \times & \text{yellow} \\ \text{blue} & + & \text{grey} \end{pmatrix}}{\sqrt{d_{\text{model}}}} \right) \times \text{pink} = \text{purple}$$

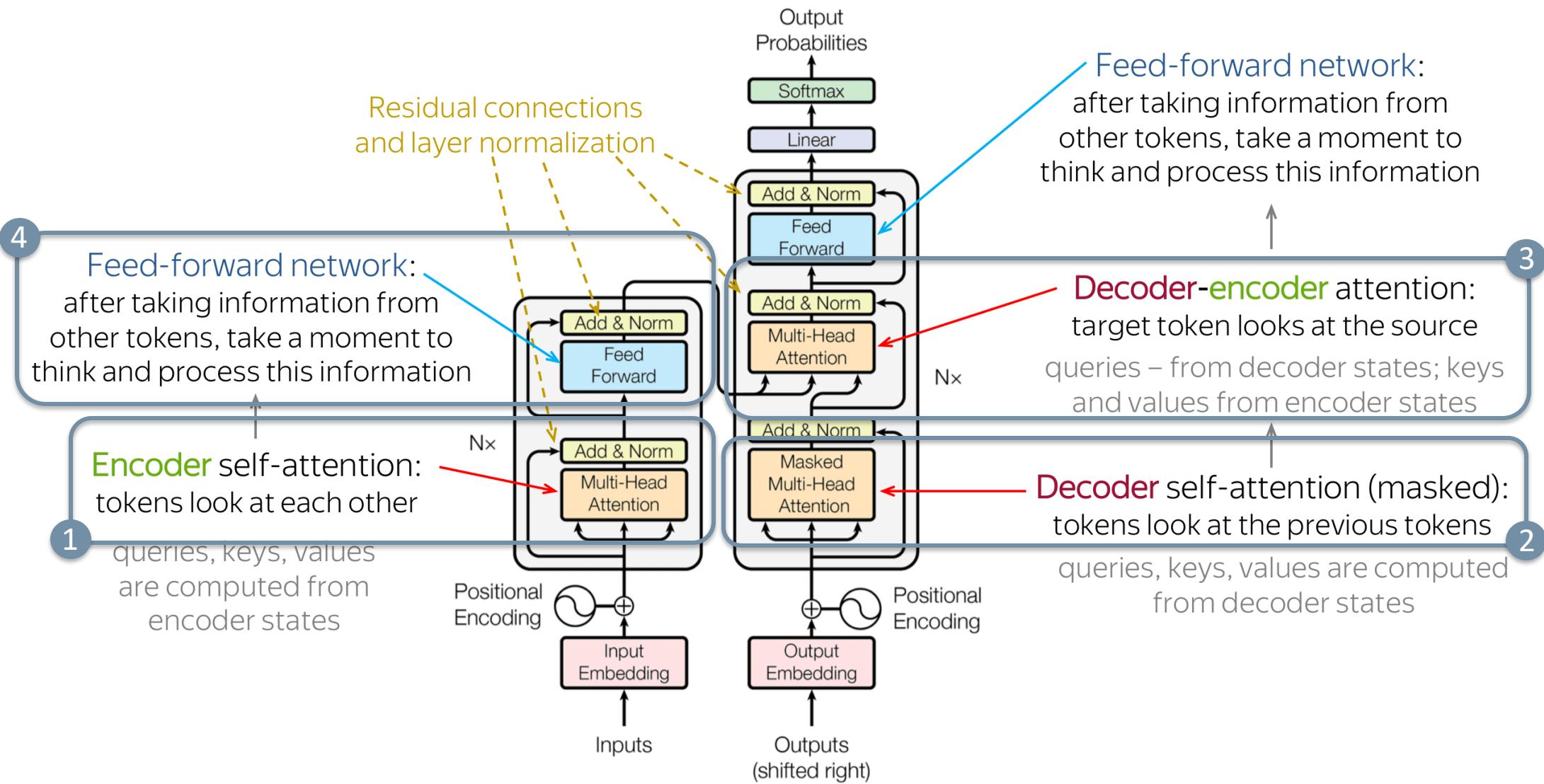
$$\text{Mask} = \begin{bmatrix} 0 & -\infty \\ 0 & 0 \end{bmatrix}$$



RNN training is $\mathcal{O}(\text{len(source)} + \text{len(target)})$

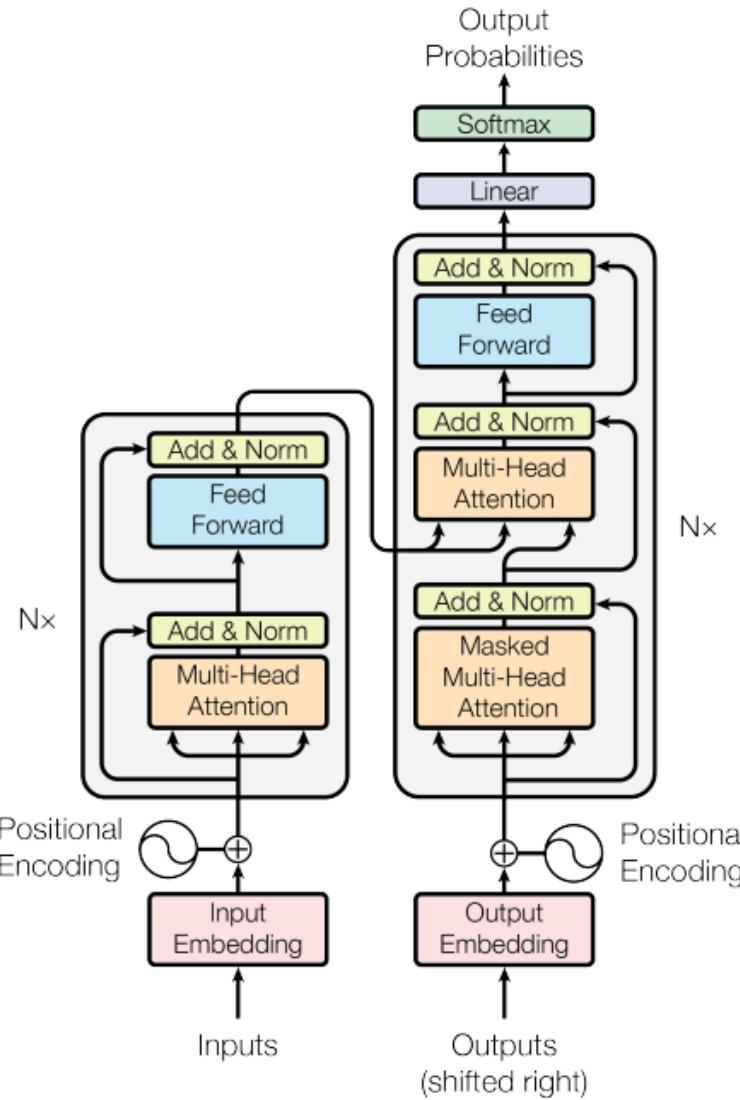
Transformer training is $\mathcal{O}(1)$ (with respect to [fixed] sequences' length)

Attention is all you need!



Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

More Transformers' Components ...



Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

More Transformers' Components ...

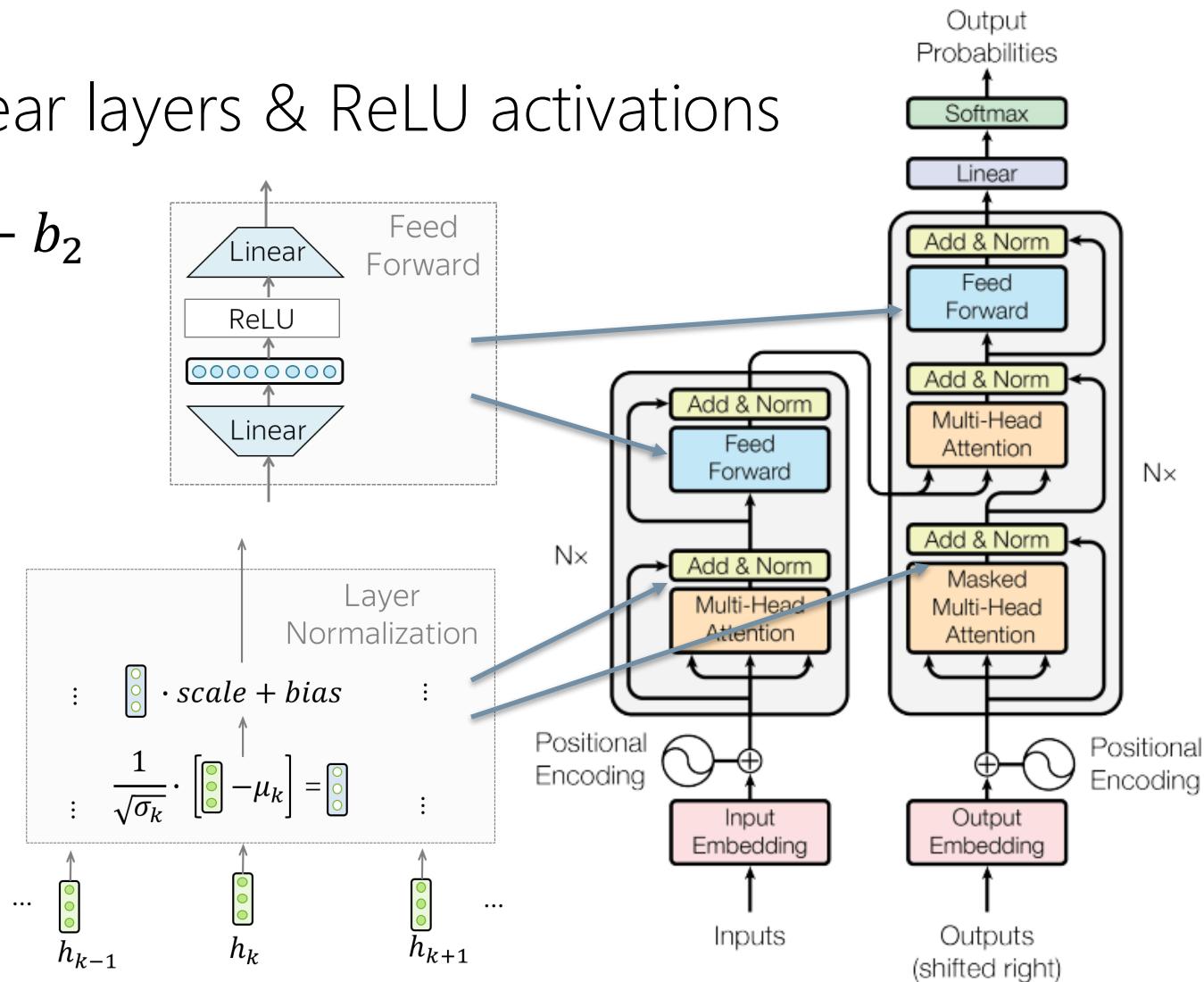
Feed forward blocks with two linear layers & ReLU activations

$$FFN(x) = \max(0, x_{W_1} + b_1) W_2 + b_2$$

Layer normalization

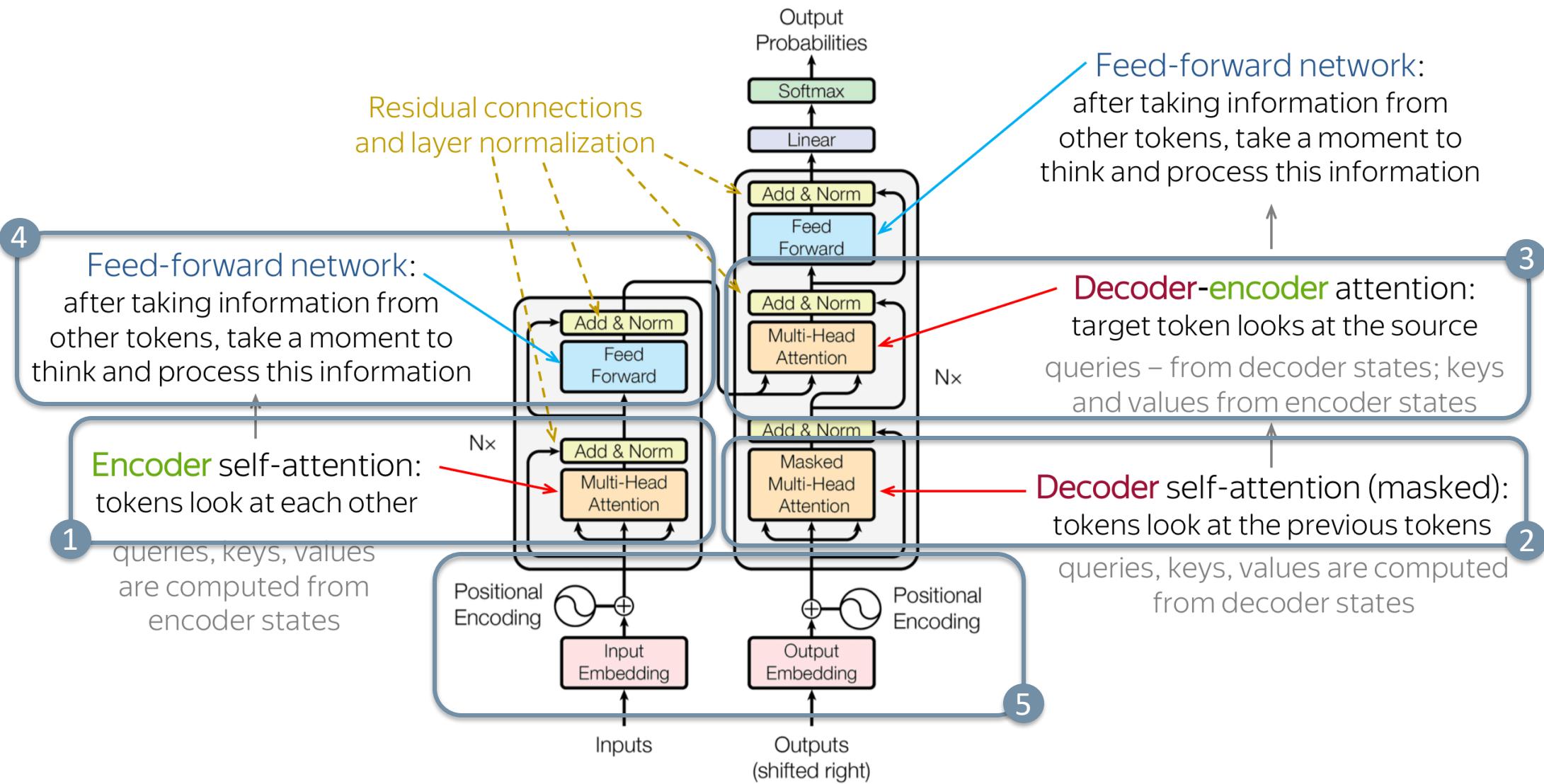
- Normalizes each single vector representation of examples in a batch independently
- Applies *scale* and *bias* globally, which are trainable layer level parameters

Residual connections ...



Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

Attention is all you need!



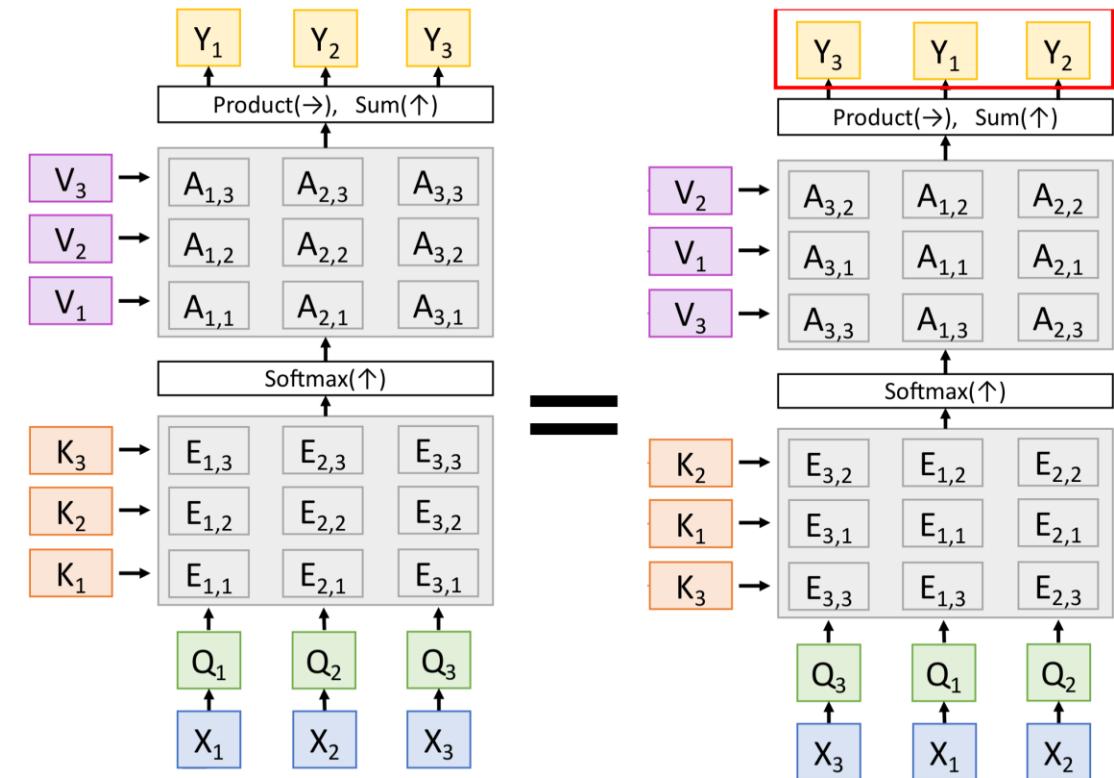
Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

Self-Attention Permutation Invariance

The Self-Attention mechanism is permutation invariant by nature as it does not depend on the position nor the order of words in the sequence

If you change the order of words, this has no impact on the attention values, but just on their order

Positional encoding is used to make self-attention depend also on the position of the input



Positional Encoding

A token input representation is the sum of two embeddings:

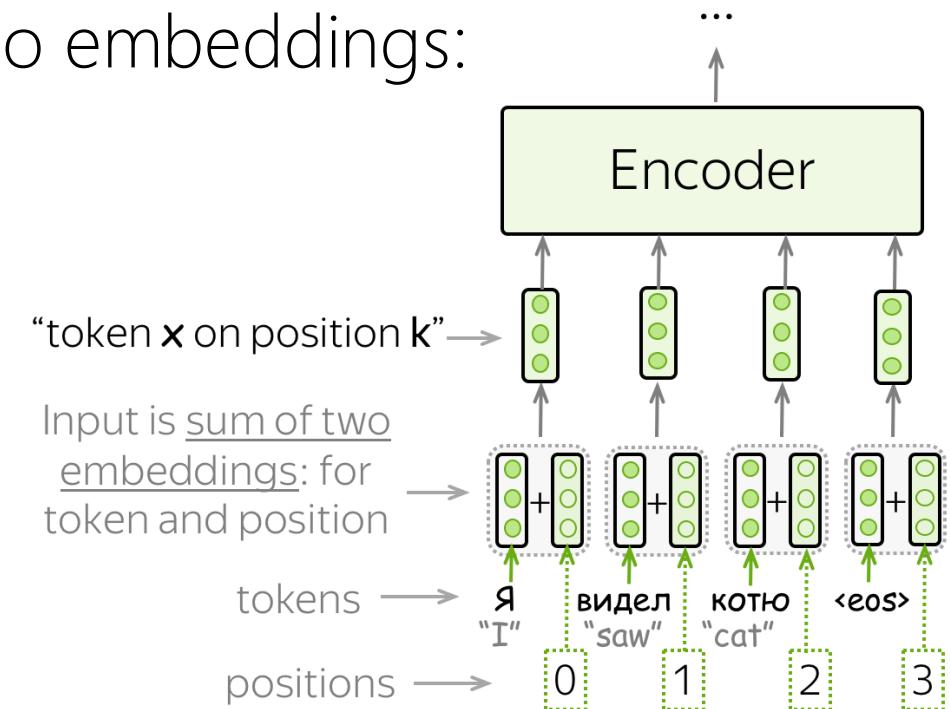
- for tokens (as we always do)
- for positions (needed for this model)

Positional embeddings can be learned, but
Transformer uses fixed positional encodings:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

where pos is position, i is the vector dimension, and d_{model} the input size.
Authors have used learned encodings too ...



Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

Positional Encoding

A token input representation is **the sum** of two embeddings:

- for tokens (as we always do)
- for positions (needed for this model)

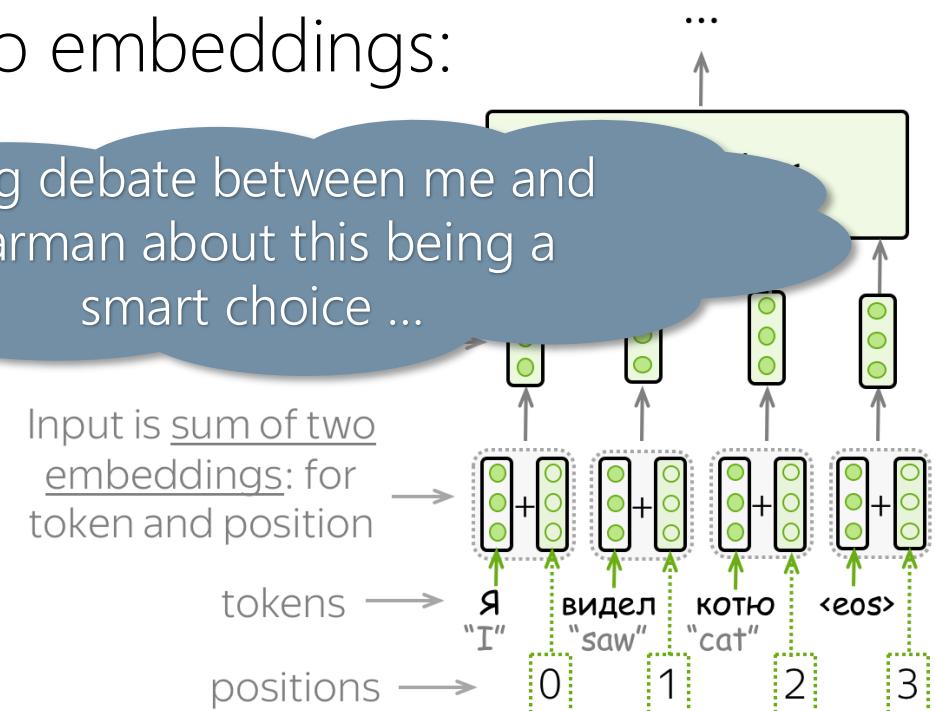
Long debate between me and Carman about this being a smart choice ...

Positional embeddings can be learned, but Transformer uses fixed positional encodings:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

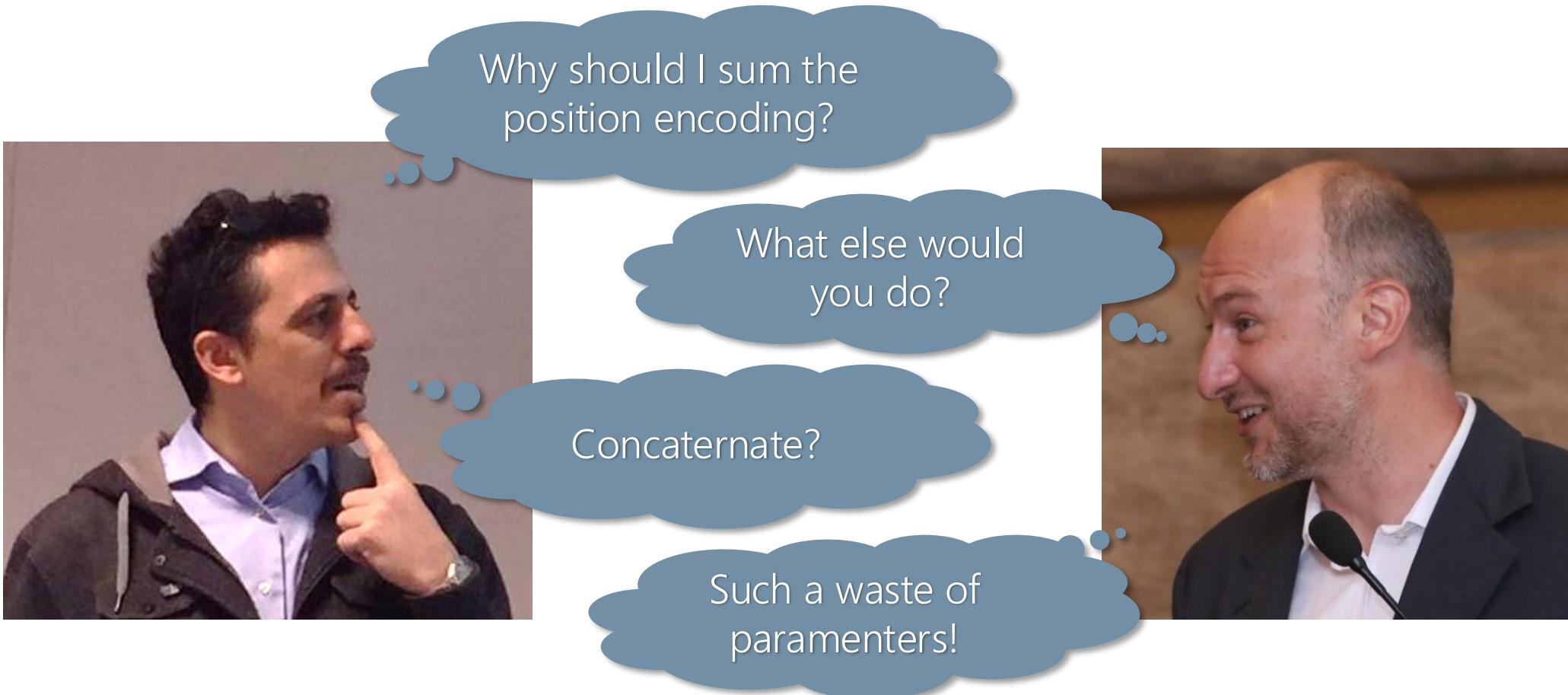
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

where pos is position, i is the vector dimension, and d_{model} the input size.
Authors have used learned encodings too ...



Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

Summation vs Concatenation



Adding vs. concatenating positional embeddings & Learned positional encodings <https://youtu.be/M2ToEXF6Olw>

Positional Encoding

A token input representation is **the sum** of two embeddings:

- for tokens (as we always do)
- for positions (needed for this model)

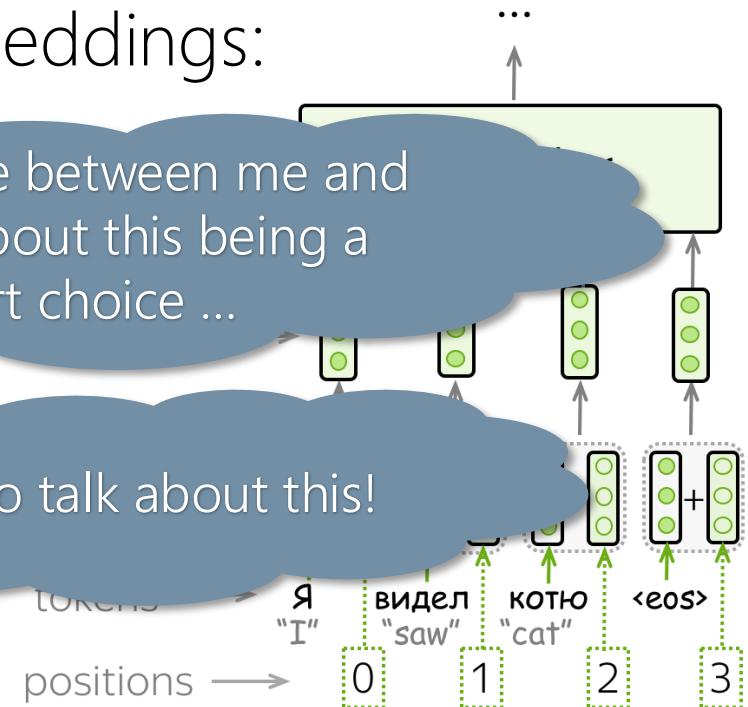
Long debate between me and Carman about this being a smart choice ...

Positional embeddings can be learned, but Transformer uses fixed positional encoding

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

where pos is position, i is the vector dimension, and d_{model} the input size.
Authors have used learned encodings too ...



Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

Intuition Behind Transformer Position Encoding

Consider at the binary representation of a position, i.e., a number

0 :	0 0 0 0	8 :	1 0 0 0
1 :	0 0 0 1	9 :	1 0 0 1
2 :	0 0 1 0	10 :	1 0 1 0
3 :	0 0 1 1	11 :	1 0 1 1
4 :	0 1 0 0	12 :	1 1 0 0
5 :	0 1 0 1	13 :	1 1 0 1
6 :	0 1 1 0	14 :	1 1 1 0
7 :	0 1 1 1	15 :	1 1 1 1

- The LBS is alternating on every number
- The second-lowest bit is rotating on every two
- Frequency halves the next position, and so on.

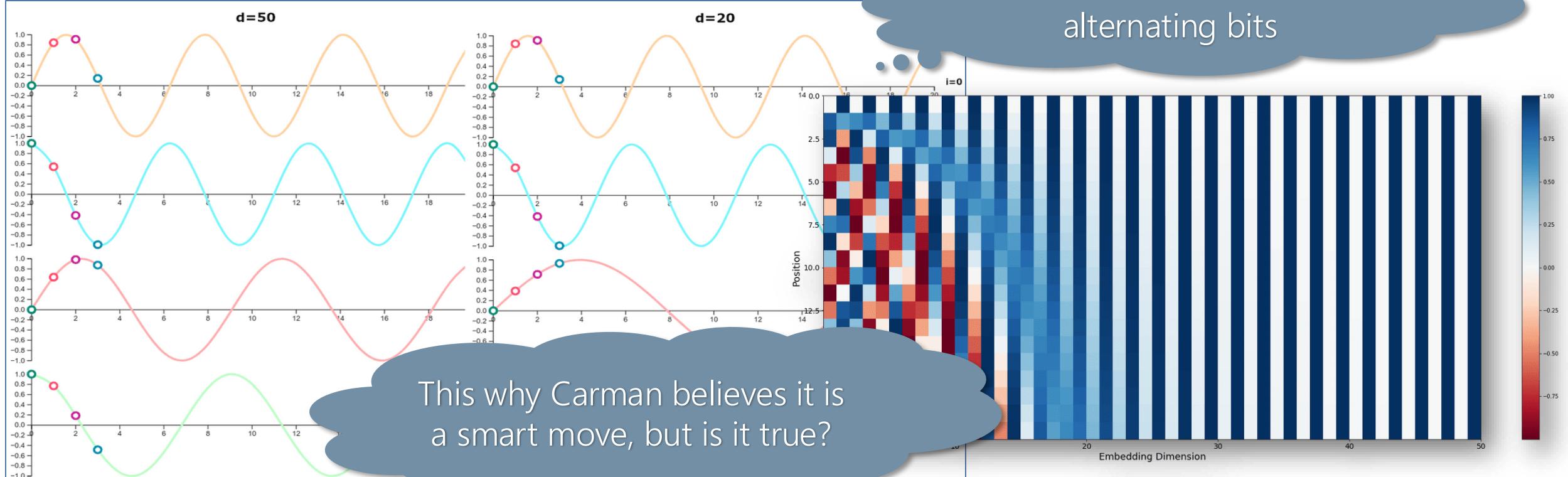


Let's pos be the position in an input sequence and $PE_{pos} \in \mathbb{R}^{d_{model}}$ its encoding. The encoding function $f(pos): \mathbb{N} \rightarrow \mathbb{R}^{d_{model}}$ is defined as

$$PE_{pos}^{(i)} = f(pos)^{(i)} = \begin{cases} \sin(\omega_k \cdot pos), & i = 2k \\ \cos(\omega_k \cdot pos), & i = 2k + 1 \end{cases} \quad \omega_k = \frac{1}{10000^{2k/d_{model}}}$$

Transformer Architecture: The Positional Encoding https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Visualizing Transformer Position Encoding



p_0	p_1	p_2	p_3	
0.000	0.841	0.909	0.141	$i=0$
1.000	0.540	-0.416	-0.990	$i=1$
0.000	0.638	0.983	0.875	$i=2$
1.000	0.770	0.186	-0.484	$i=3$

p_0	p_1	p_2	p_3	
0.000	0.841	0.909	0.141	$i=0$
1.000	0.540	-0.416	-0.990	$i=1$
0.000	0.388	0.715	0.930	$i=2$
1.000	0.922	0.699	0.368	$i=3$

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

Positional encoding visualization <https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers>

Position Encoding and Relative Positioning

For every sine-cosine pair for frequency ω_k find a linear transformation $M \in \mathbb{R}^{2 \times 2}$, independent of pos , where the following equation holds

$$M \begin{bmatrix} \sin(\omega_k \cdot pos) \\ \cos(\omega_k \cdot pos) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k \cdot (pos + k)) \\ \cos(\omega_k \cdot (pos + k)) \end{bmatrix}$$

Let $M \in \mathbb{R}^{2 \times 2}$ we want to find u_1, v_1, u_2, v_2 so that

$$\begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \end{bmatrix} \begin{bmatrix} \sin(\omega_k \cdot pos) \\ \cos(\omega_k \cdot pos) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k \cdot (pos + k)) \\ \cos(\omega_k \cdot (pos + k)) \end{bmatrix}$$

By the addition theorem

$$\begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \end{bmatrix} \begin{bmatrix} \sin(\omega_k \cdot pos) \\ \cos(\omega_k \cdot pos) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k \cdot pos) \cos(\omega_k \cdot k) + \cos(\omega_k \cdot pos) \sin(\omega_k \cdot k) \\ \cos(\omega_k \cdot pos) \cos(\omega_k \cdot k) - \sin(\omega_k \cdot pos) \sin(\omega_k \cdot k) \end{bmatrix}$$

Linear Relationships in the Transformer's Positional Encoding <https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/>

Position Encoding and Relative Positioning

From the previous we derive the following two equations

$$u_1 \sin(\omega_k \cdot pos) + v_1 \cos(\omega_k \cdot pos) = \sin(\omega_k \cdot pos) \cos(\omega_k \cdot k) + \cos(\omega_k \cdot pos) \sin(\omega_k \cdot k)$$

$$u_2 \sin(\omega_k \cdot pos) + v_2 \cos(\omega_k \cdot pos) = \cos(\omega_k \cdot pos) \cos(\omega_k \cdot k) - \sin(\omega_k \cdot pos) \sin(\omega_k \cdot k)$$

by solving these equations, we get the following

$$u_1 = \cos(\omega_k \cdot k) \quad v_1 = \sin(\omega_k \cdot k)$$

$$u_2 = -\sin(\omega_k \cdot k) \quad v_2 = \cos(\omega_k \cdot k)$$

The transformation matrix is thus independent from *pos* (it is a rotation)

$$M = \begin{bmatrix} \cos(\omega_k \cdot k) & \sin(\omega_k \cdot k) \\ -\sin(\omega_k \cdot k) & \cos(\omega_k \cdot k) \end{bmatrix}$$

Linear Relationships in the Transformer's Positional Encoding <https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/>

Position Encoding and Relative Positioning

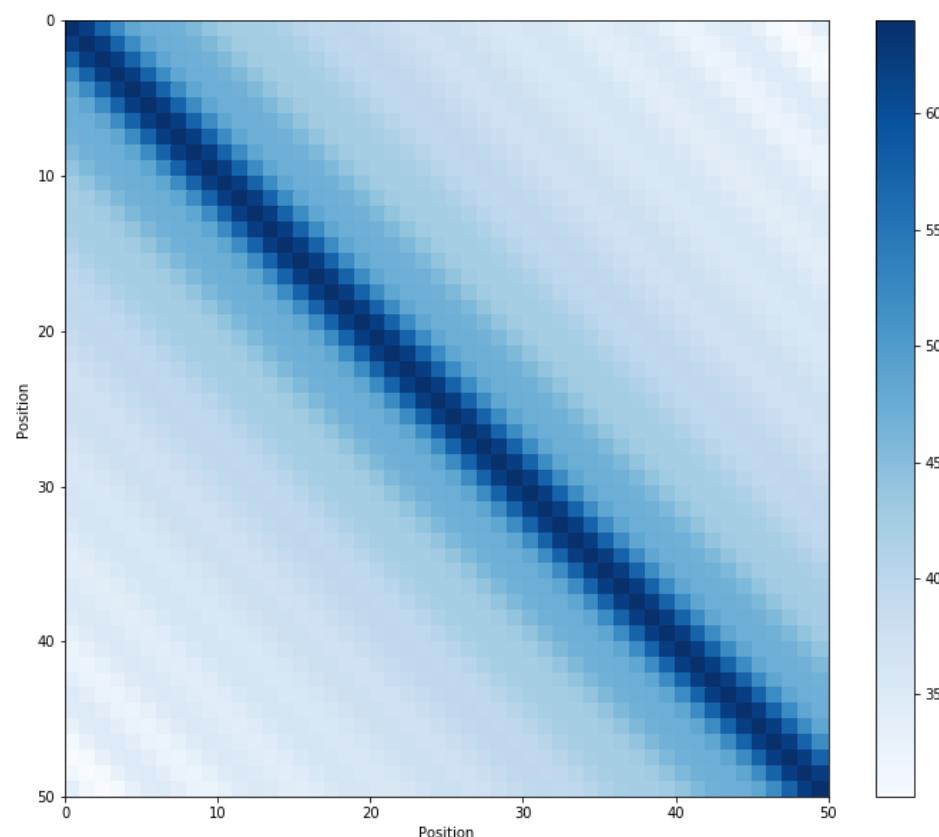
From the previous we have

$$u_1 \sin(\omega_k \cdot pos) + v_1 \cos(\omega_k \cdot pos)$$

$$u_2 \sin(\omega_k \cdot pos) + v_2 \cos(\omega_k \cdot pos)$$

by solving these equations we get

The transformation



and the relative position encoding is given by

$$\begin{aligned} & u_1 \sin(\omega_k \cdot pos) + v_1 \cos(\omega_k \cdot pos) \\ & u_2 \sin(\omega_k \cdot pos) + v_2 \cos(\omega_k \cdot pos) \end{aligned}$$

This transformation is a rotation (it is a rotation)

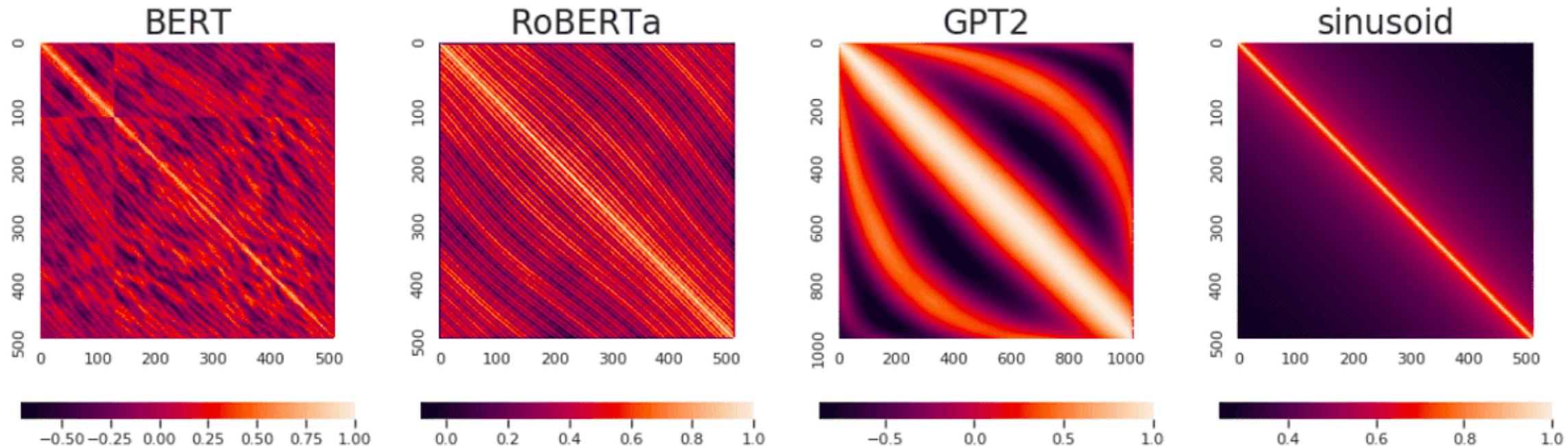
Neighboring time-steps distance are symmetrical and decay smoothly!

Linear Relationships in the Transformer's Positional Encoding <https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/>

Learning Positional Embeddings

This why I believe it is not such a smart move! ;-)

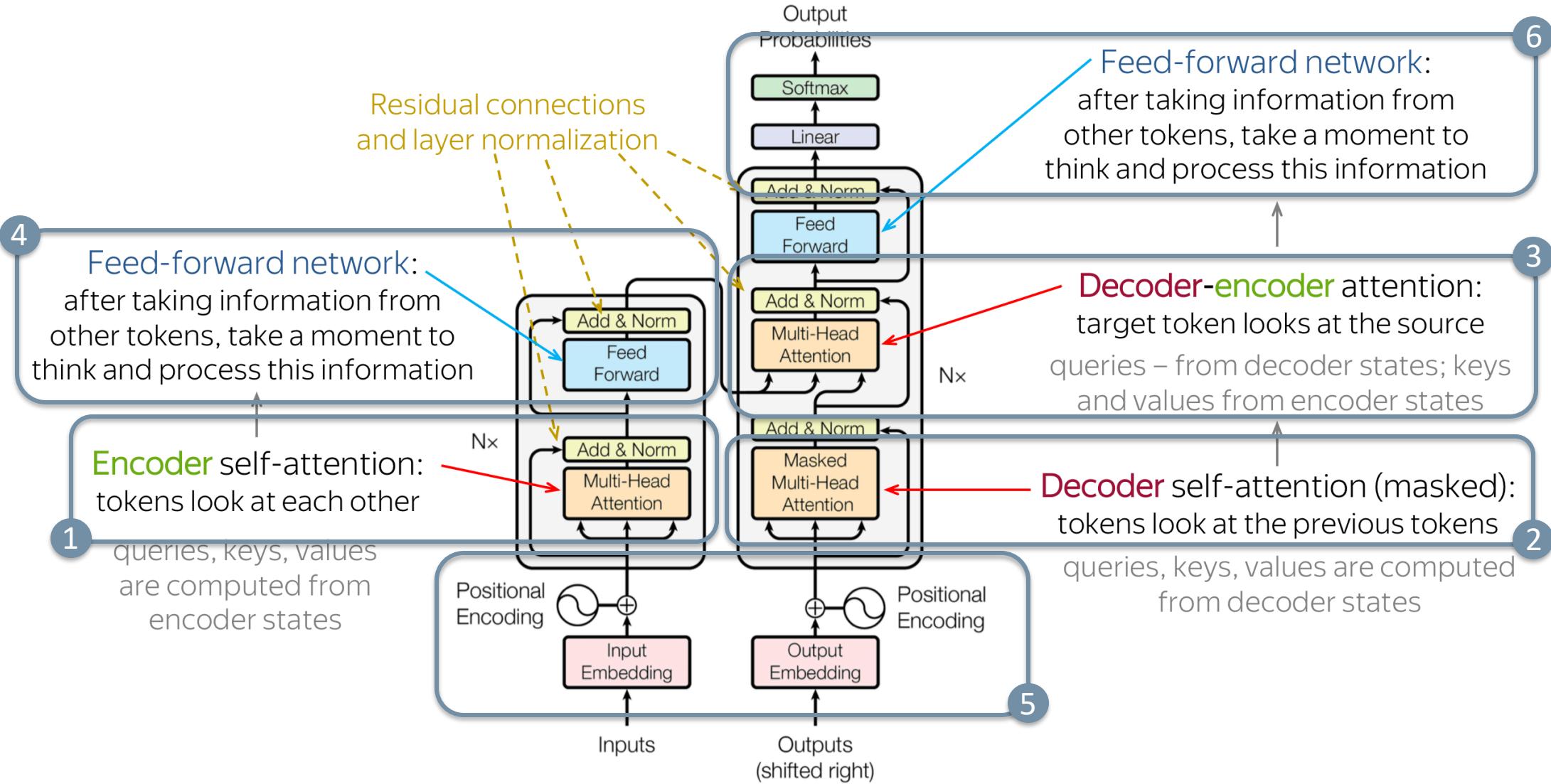
Nevertheless, state of the art Transformers (BERT, RoBERTa, GPT-2, ...) learn the positional encoding instead of using a fixed one



Moreover, some of them use summation, others use concatenation ...

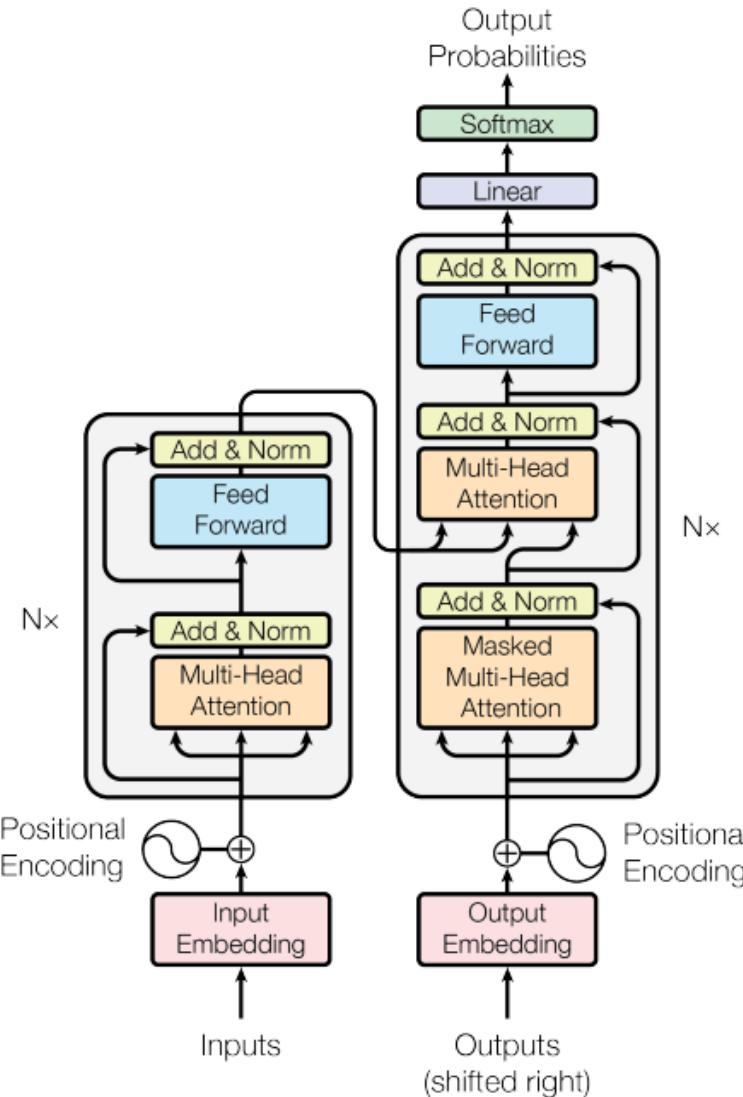
What Do Position Embeddings Learn? An Empirical Study of Pre-Trained Language Model Positional Encoding <https://arxiv.org/abs/2010.04903>

Attention is all you need!



Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

Transformer Complexity



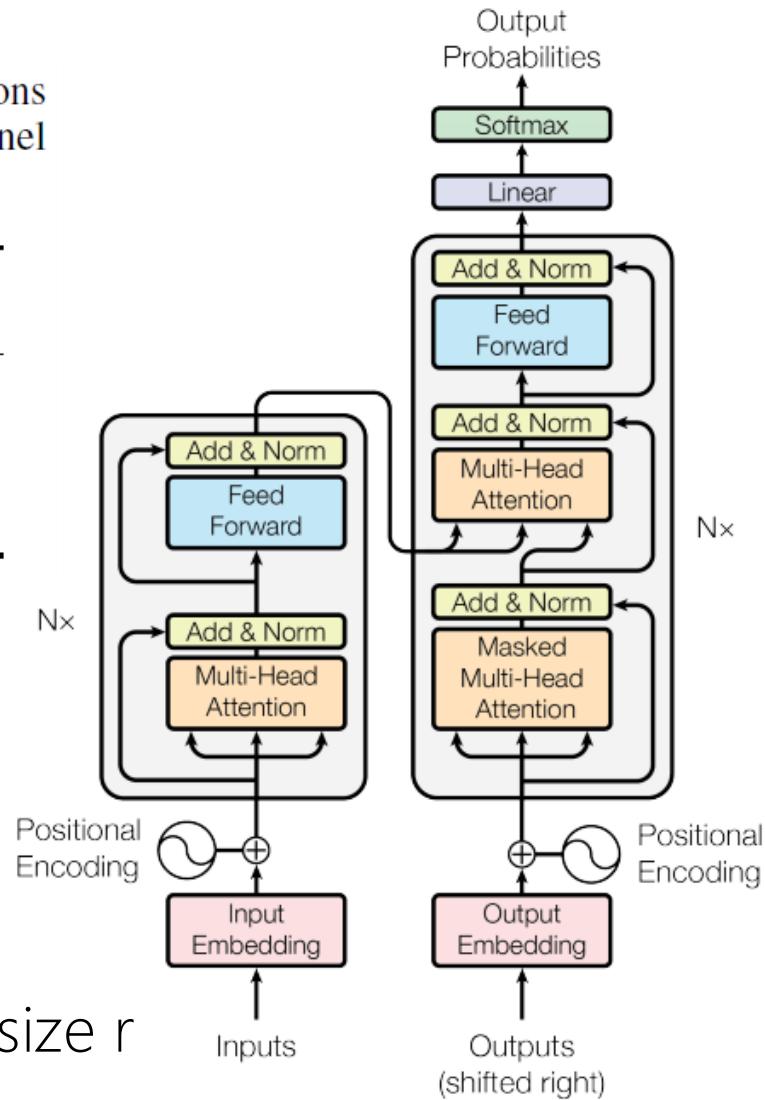
Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

Transformer Complexity

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

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

- Self-Attention has $O(1)$ maximum path length (capture long range dependency easily)
- When $n < d$, Self-Attention has lower complexity than a recurrent layer
- We can always restrict attention to a neighborhood of size r



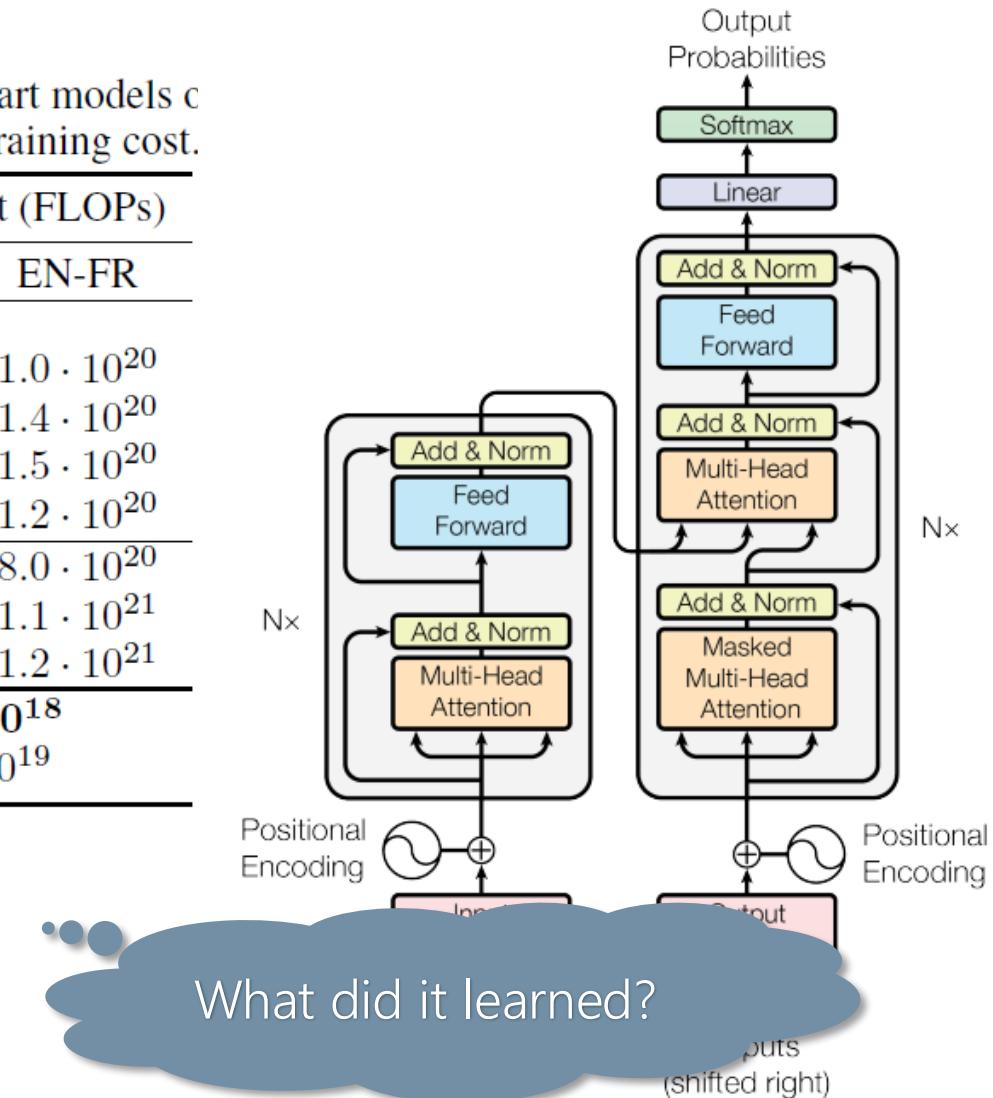
Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

Transformer Performance

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8	$2.3 \cdot 10^{19}$	

- EN-to-DE: new state-of-the-art
- EN-to-FR: new single-model state-of-the-art



Attention Is All You Need <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>

Transformer Heads are Interpretable

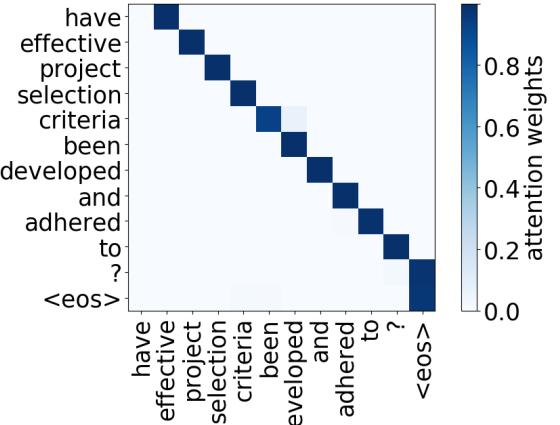
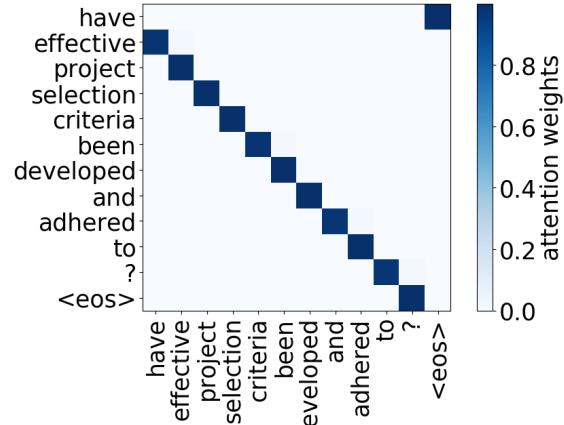
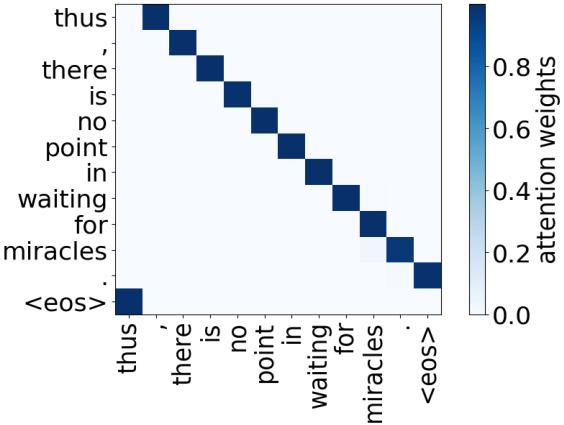
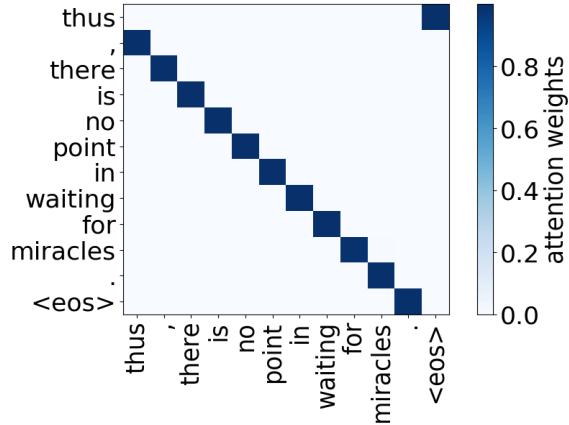
By looking at how much, on average, different heads "contribute" to generated translations it turns out only a small number are important and they play interpretable "roles":

- Positional: attend to a token's immediate neighbors, and the model has several such heads (usually 2-3 looking at the previous and 2 looking at the next ones)
- Syntactic: learned to track some major syntactic relations in the sentence (subject-verb, verb-object, etc.)
- Rare tokens: the most important head on the first layer attends to the least frequent tokens in a sentence (this is true for models trained on different language pairs!)

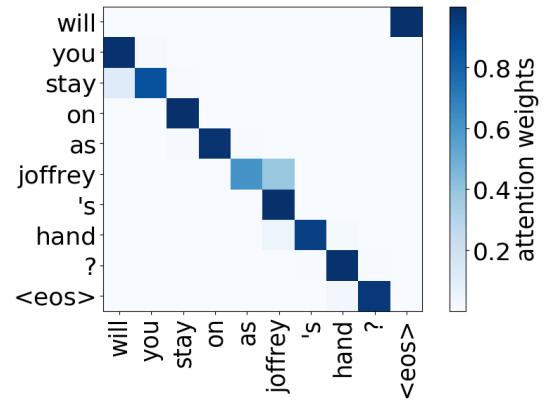
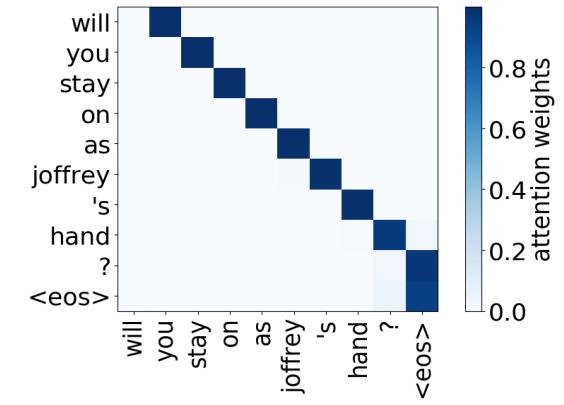
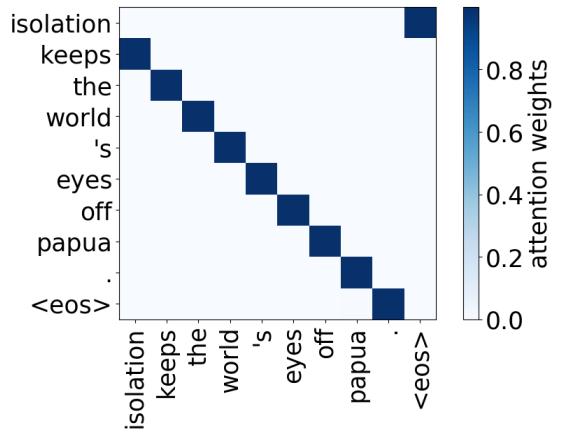
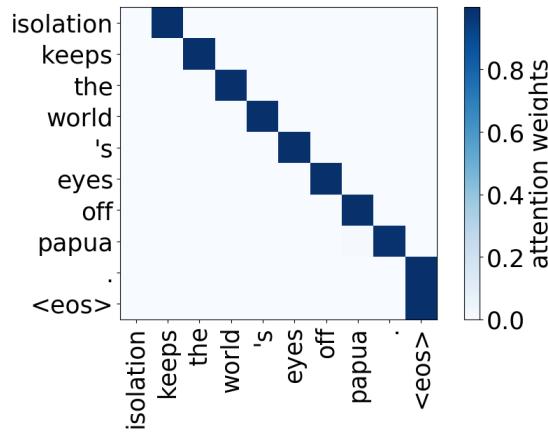


Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned https://lena-voita.github.io/posts/acl19_heads.html

Positional Heads



Model Trained on WMT EN-DE

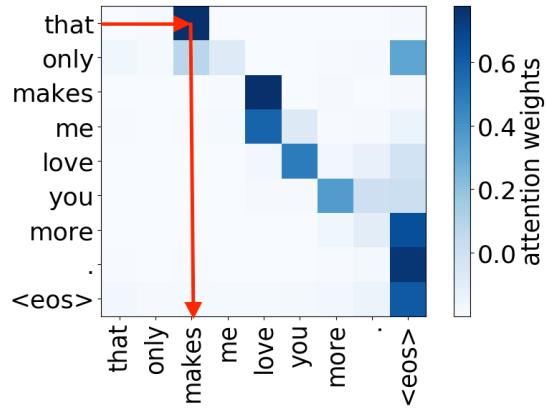


Model Trained on WMT EN-RU

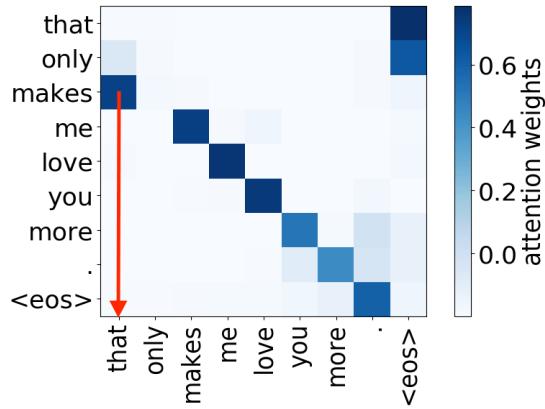
Model Trained on OpenSubtitles EN-RU

Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned https://lena-voita.github.io/posts/acl19_heads.html

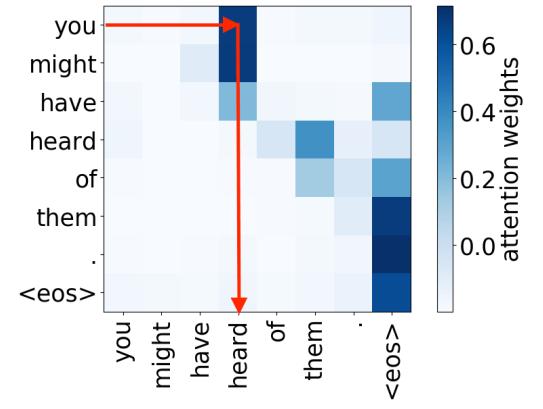
Syntactic Heads



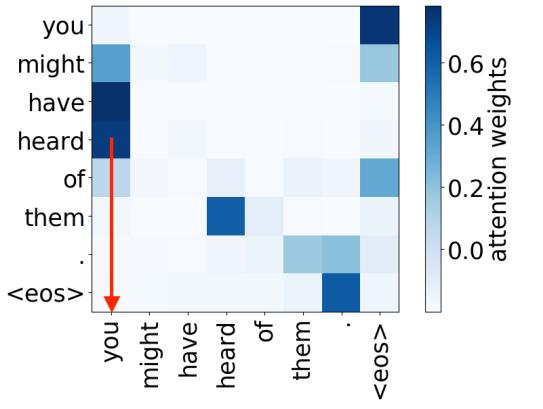
Subject->Verb



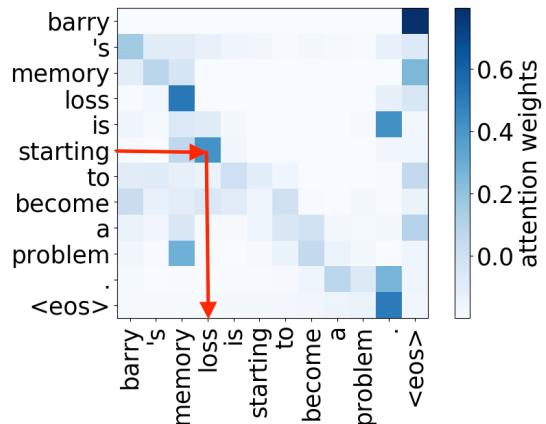
Verb->Subject



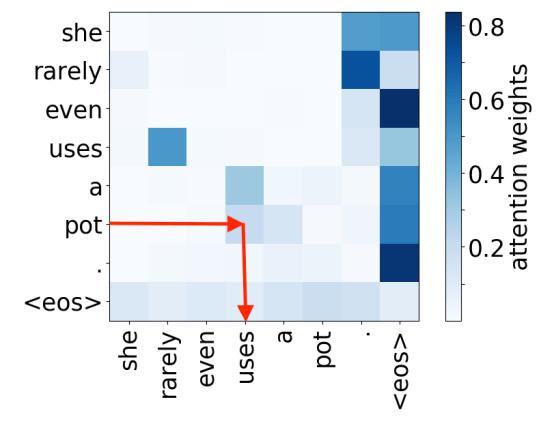
Subject->Verb



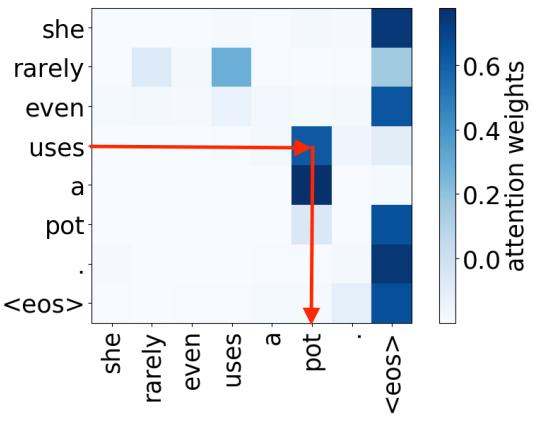
Verb->Subject



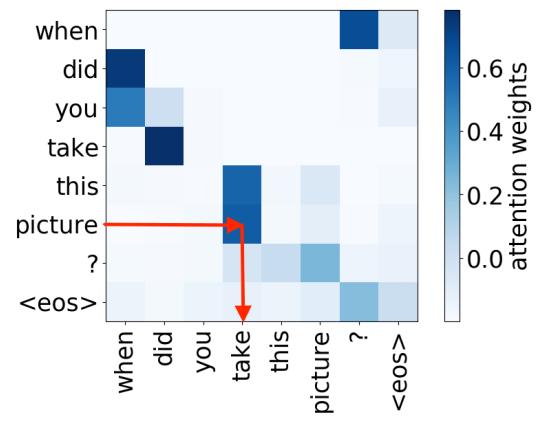
Verb -> Subject



Object -> Verb



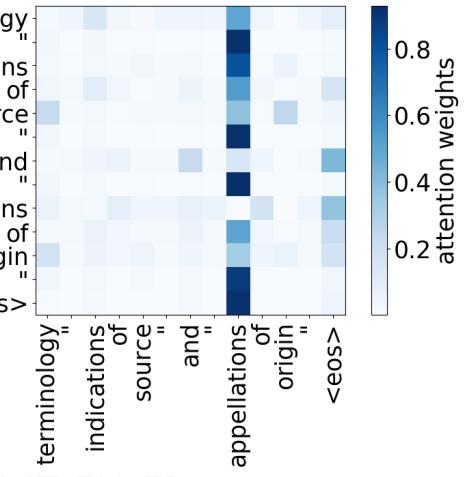
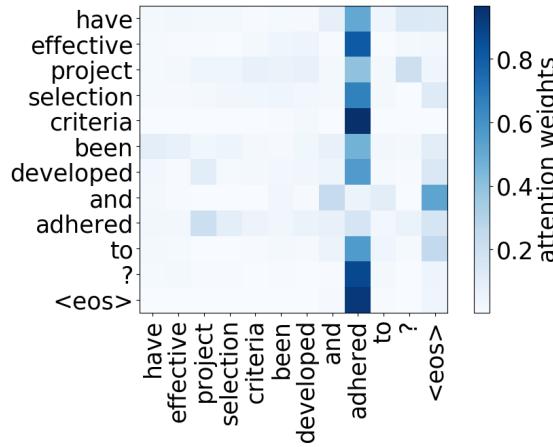
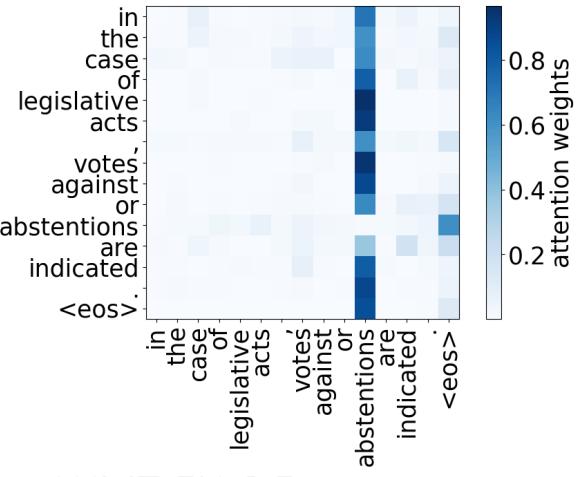
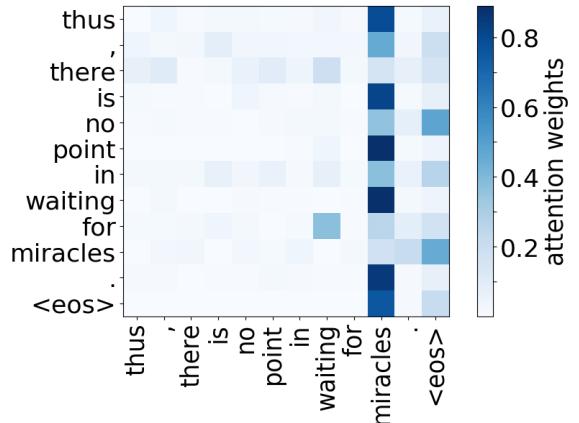
Verb -> Object



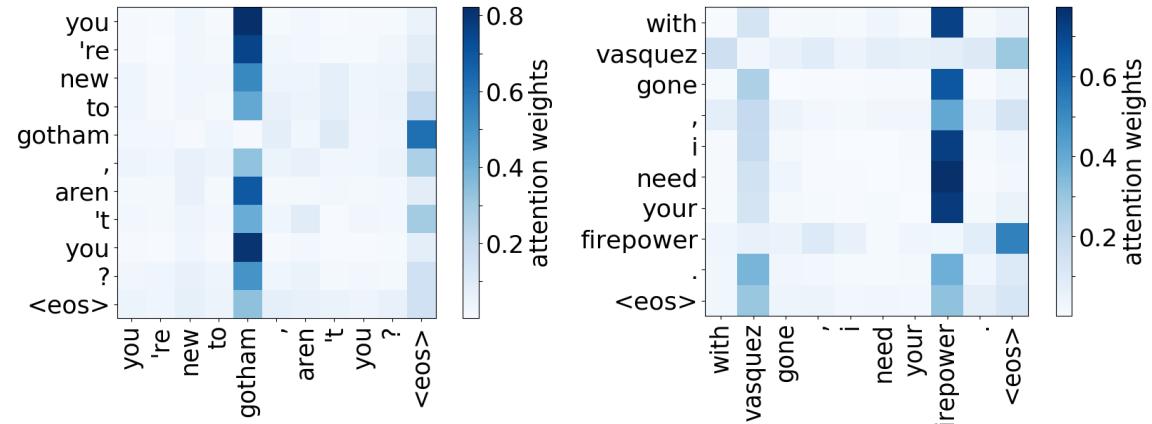
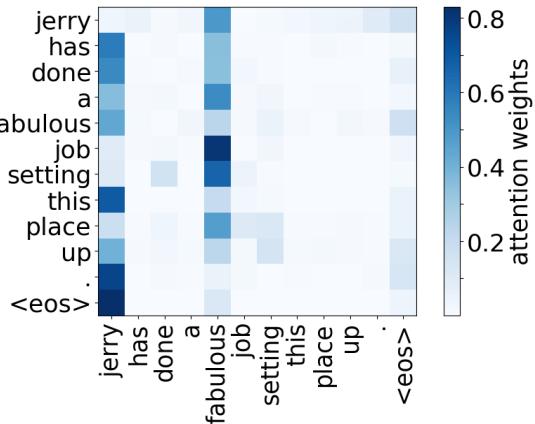
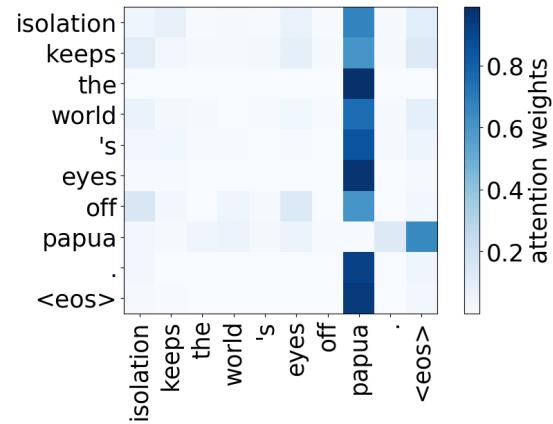
Object -> Verb

Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned https://lena-voita.github.io/posts/acl19_heads.html

Rare Tokens Heads



Model Trained on WMT EN-DE

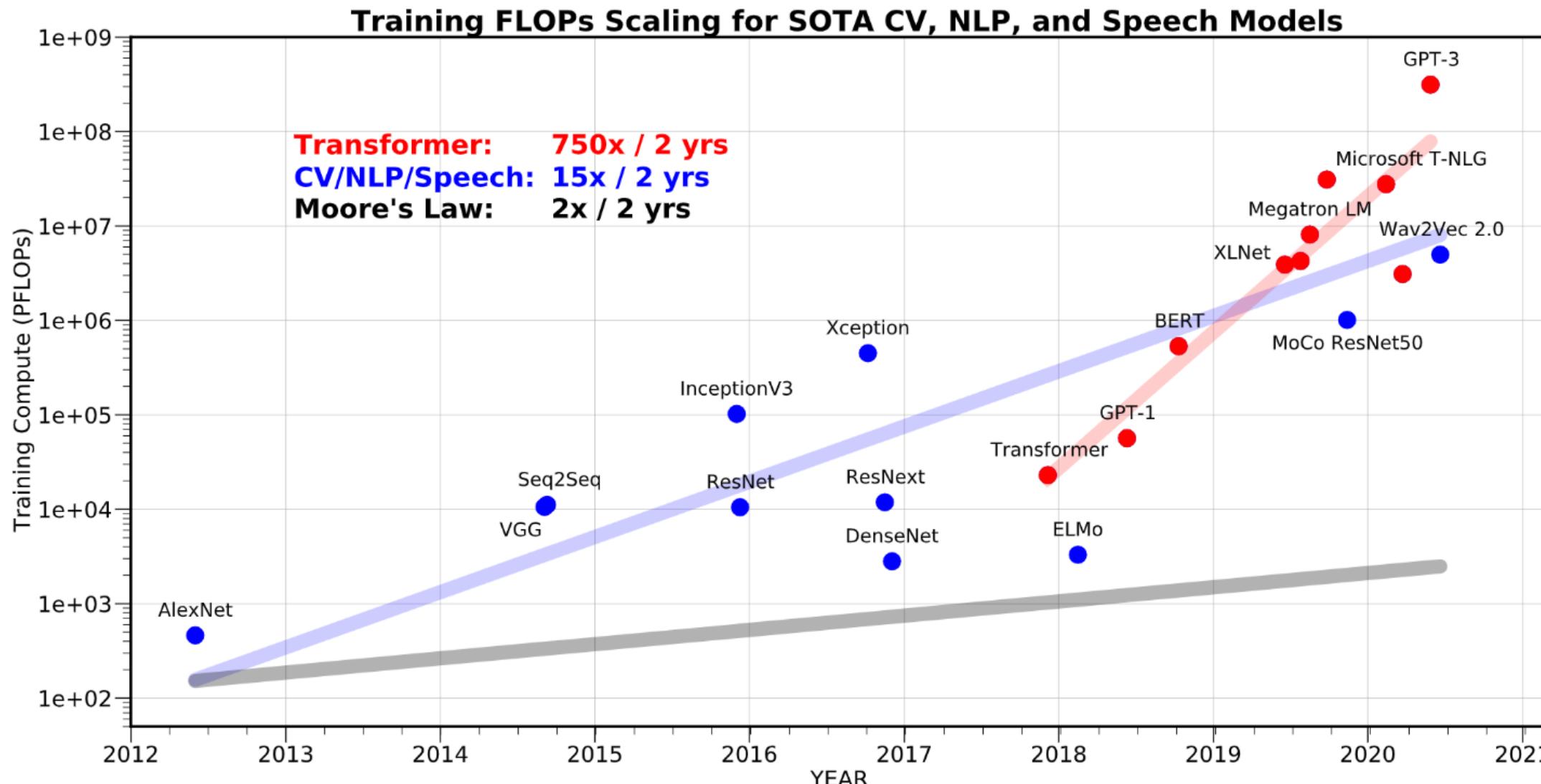


Model Trained on WMT EN-RU

Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned https://lena-voita.github.io/posts/acl19_heads.html

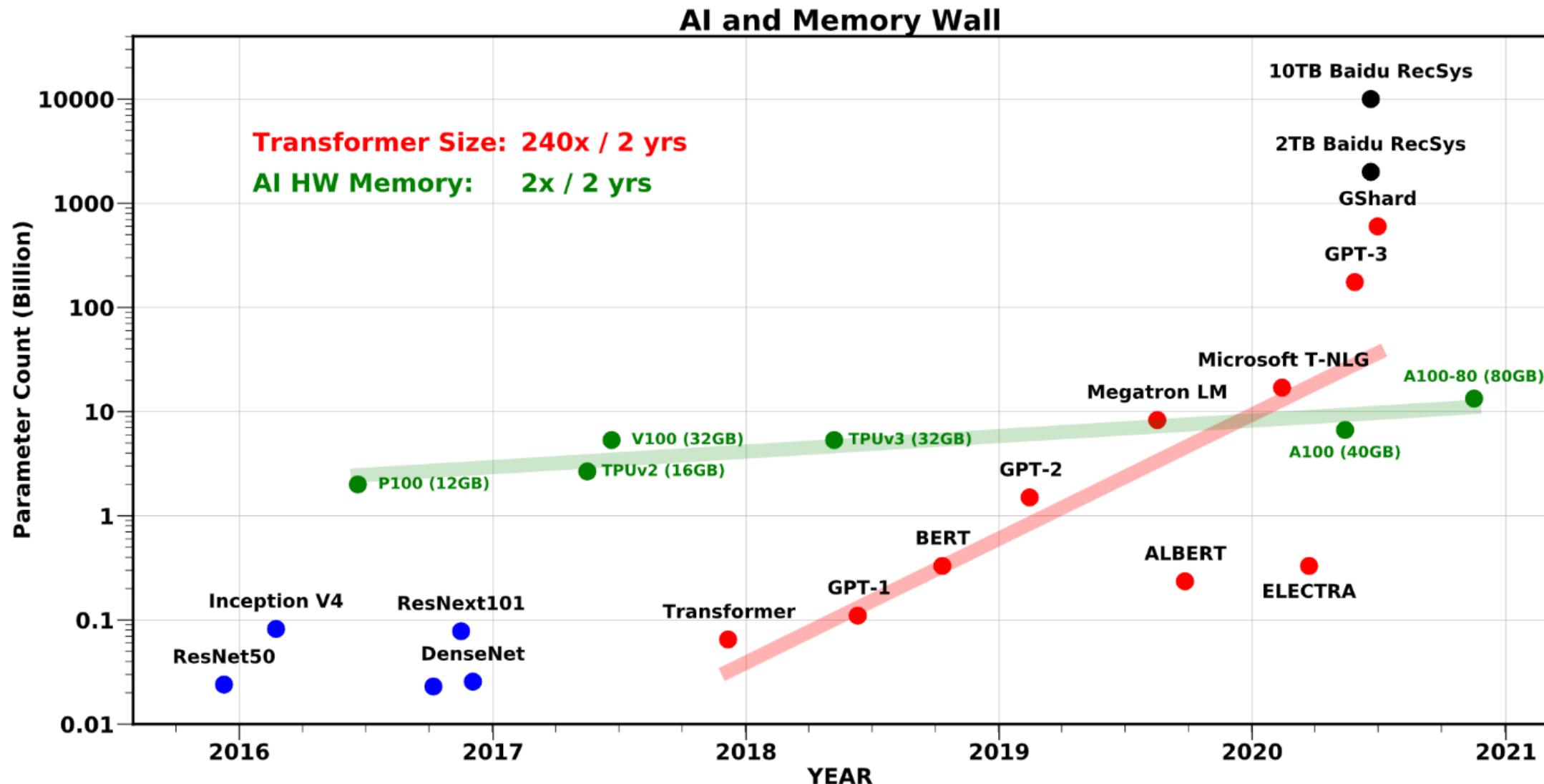
Model Trained on OpenSubtitles EN-RU

Are there any limits for Transformers?



AI and Memory Wall <https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8>

Are there any limits for Transformers?



AI and Memory Wall <https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8>

Acknowledgements

Amazing images and content taken from Elena Voita's NLP Course

NLP Course | For You

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

Step by step implementation of Transformers

- *Text classification with Transformer*
[https://keras.io/examples/nlp/text classification with transformer/](https://keras.io/examples/nlp/text_classification_with_transformer/)
- *English-to-Spanish translation with a sequence-to-sequence Transformer*
[https://keras.io/examples/nlp/neural machine translation with transformer/](https://keras.io/examples/nlp/neural_machine_translation_with_transformer/)
- Neural machine translation with a Transformer and Keras
<https://www.tensorflow.org/text/tutorials/transformer?hl=en>
- *The Annotated Transformer* <http://nlp.seas.harvard.edu/annotated-transformer/>



Acknowledgements

Slides material taken from following blogs/papers (order of appearance):

- *The Unreasonable Effectiveness of Recurrent Neural Networks:*
<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
- *Sequence to sequence learning with Neural networks:* <https://arxiv.org/pdf/1409.3215.pdf>
- *Neural Machine Translation by Jointly Learning to Align and Translate:*
<https://arxiv.org/pdf/1409.0473.pdf>
- *Effective Approaches to Attention-based Neural Machine Translation*
<https://arxiv.org/abs/1508.04025>
- *A Neural Conversational Model* <https://arxiv.org/pdf/1506.05869.pdf>
- *Hierarchical Recurrent Attention Network for Response Generation*
<https://arxiv.org/pdf/1701.07149.pdf>
- *Transformer: A Novel Neural Network Architecture for Language Understanding*
<https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>
- *Attention Is All You Need* <https://arxiv.org/abs/1706.03762>



Acknowledgements

Slides material taken from following blogs/papers (continued):

- *The Illustrated Transformer* <http://jalammar.github.io/illustrated-transformer/>
- *Transformer Architecture: The Positional Encoding*
https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
- *Positional encoding visualization*
<https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers>
- *Linear Relationships in the Transformer's Positional Encoding* <https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/>
- *What Do Position Embeddings Learn? An Empirical Study of Pre-Trained Language Model Positional Encoding* <https://arxiv.org/abs/2010.04903>
- *Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned* https://lena-voita.github.io/posts/acl19_heads.html
- *AI and Memory Wall* <https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8>



Evaluation of Day 3 Lecture

<https://forms.office.com/e/hYp13wzWJN>

- Now it is closed, I will open it at the end
- 20 minutes to complete
- 32 questions 1 point each
- Each question 4 choices only 1 correct
- Best and worst (at class level) removed





POLITECNICO
MILANO 1863

Credits for images and examples to Elena Voita's
NLP Course | For You
https://lena-voita.github.io/nlp_course.html

Advanced Deep Learning

- Attention and Transformers -

Matteo Matteucci, PhD (matteo.matteucci@polimi.it)
Artificial Intelligence and Robotics Laboratory
Politecnico di Milano