POLITECNICO MILANO 1863

Credits for images and examples to Elena Voita's

NLP Course | For You

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

Artificial Neural Networks and Deep Learning

- Attention Mechanism and Transformers -

Matteo Matteucci, PhD (matteo.matteucci@polimi.it)

Artificial Intelligence and Robotics Laboratory

Politecnico di Milano

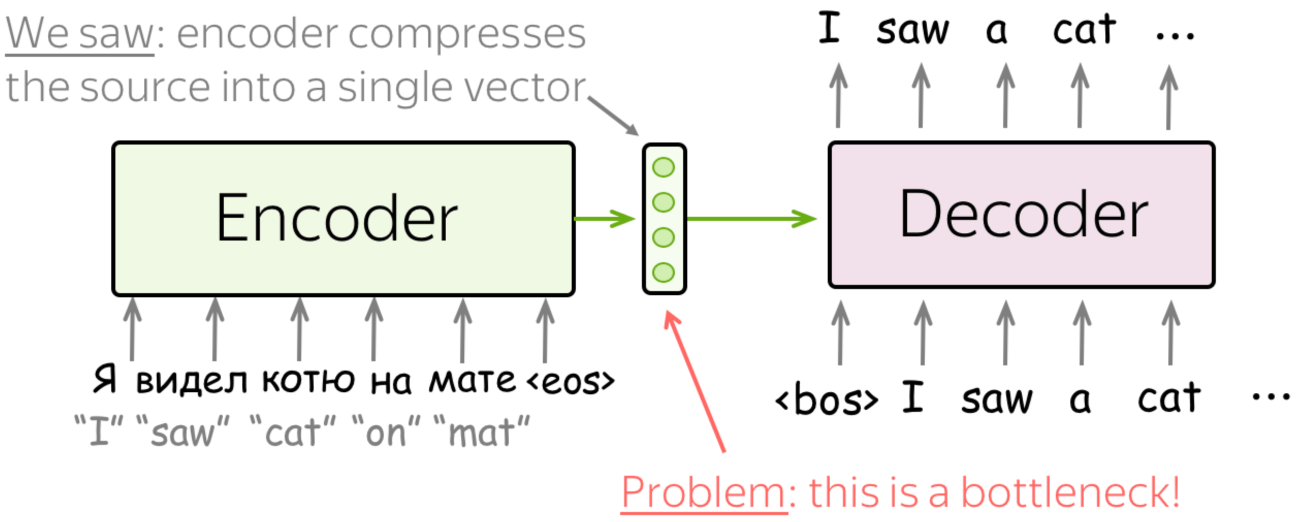
AIRLAB

ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

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.

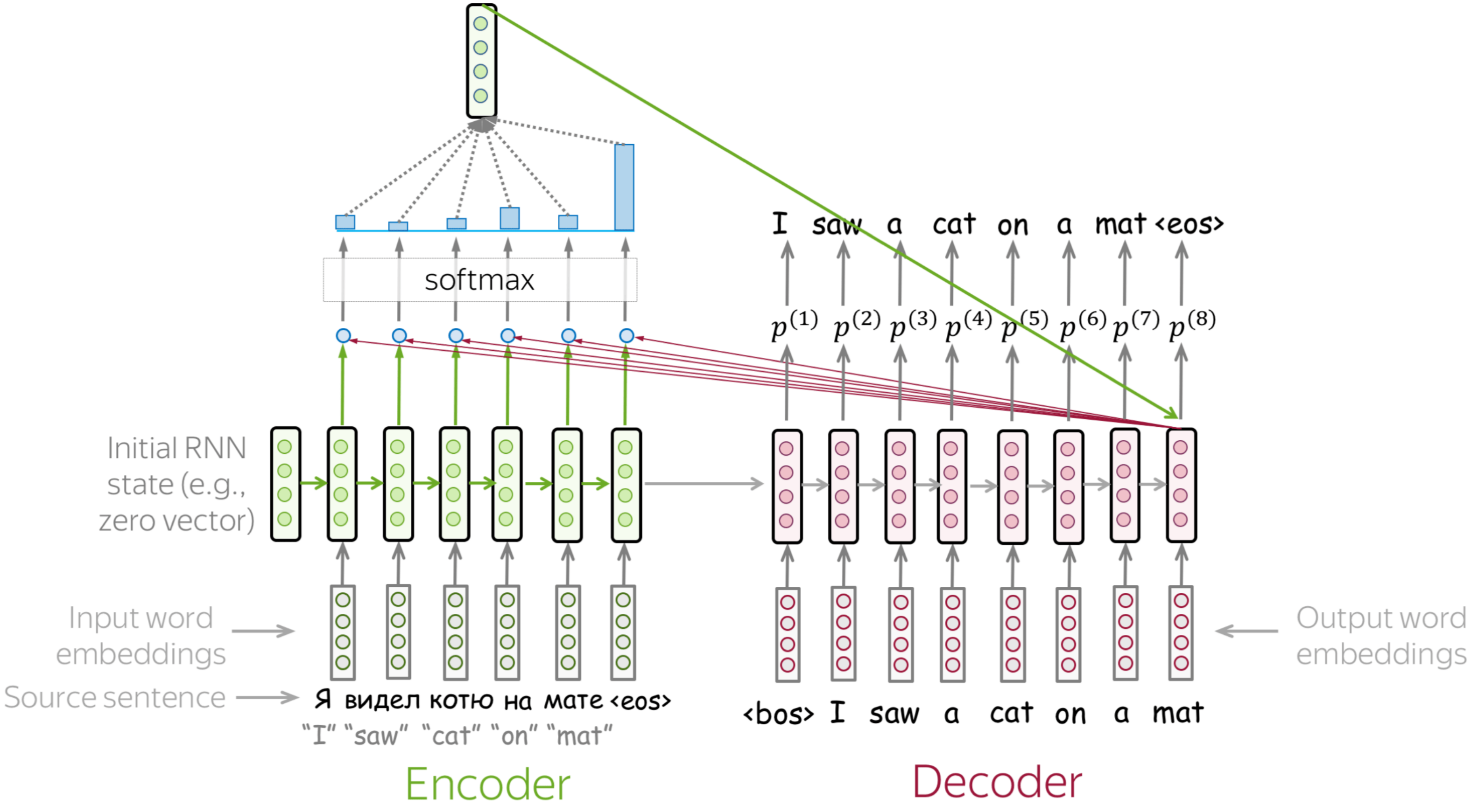


Attention let the model focus on different parts of the input

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

4 Attention output

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

"source context for decoder step t"

(weighted sum)

3 Attention weights

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

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

(softmax)

2 Attention scores

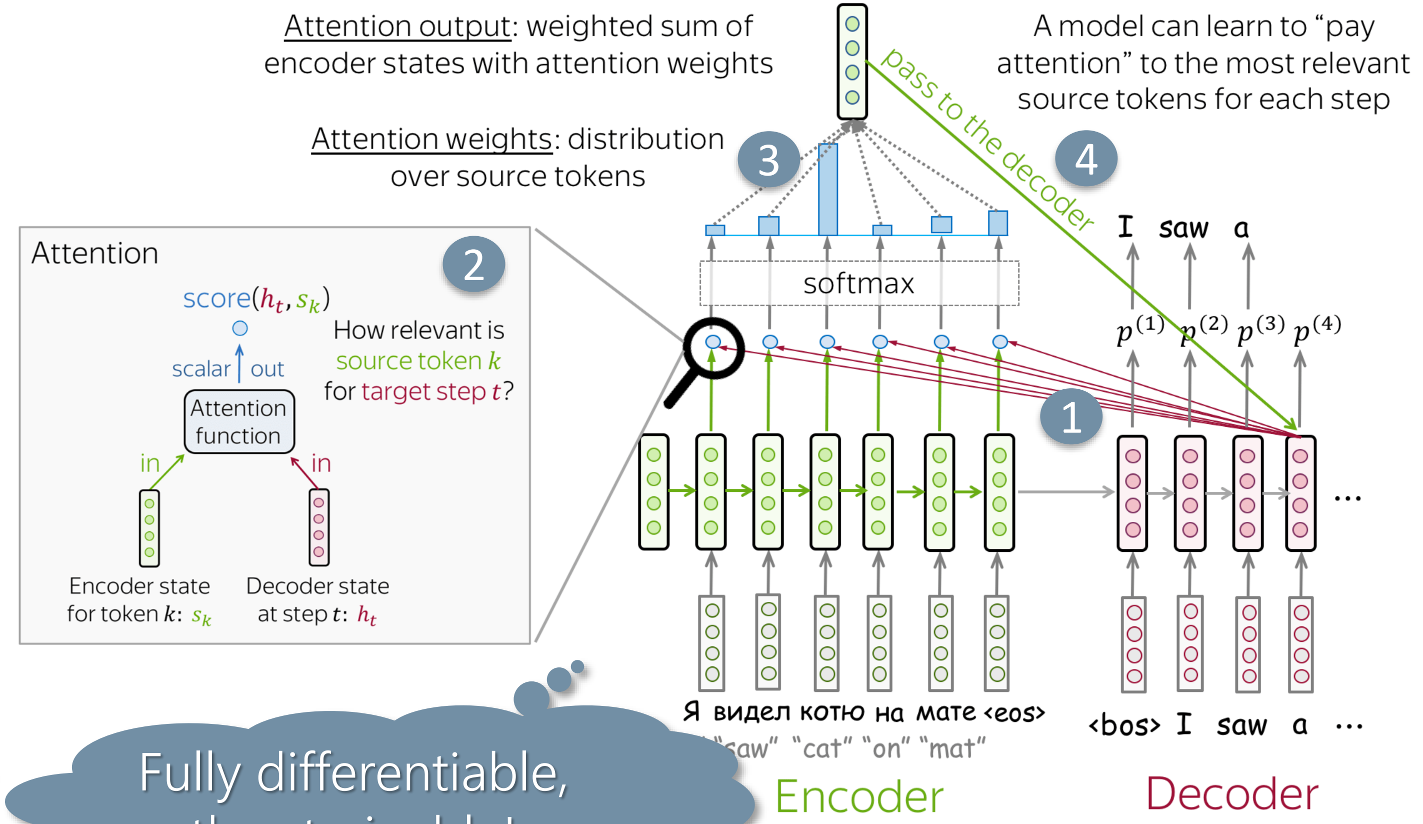
$\mathrm{score}(h_t, s_k), \mathrm{k} = 1..\mathrm{m}$

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

1 Attention input

$s_1, s_2, \dots, s_m$ all encoder states

h_t one decoder state



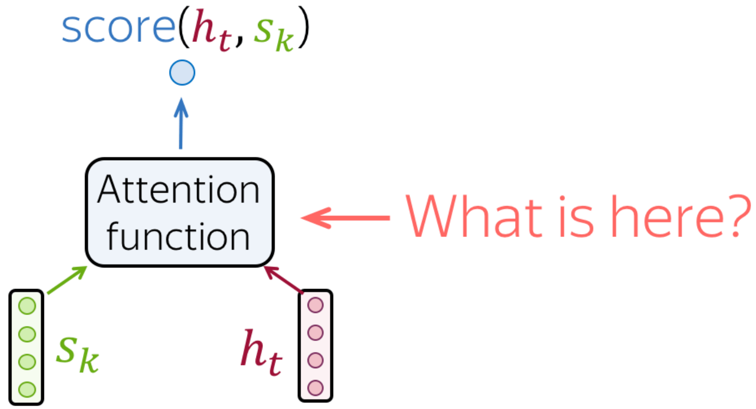
Attention scores can be computed in different ways

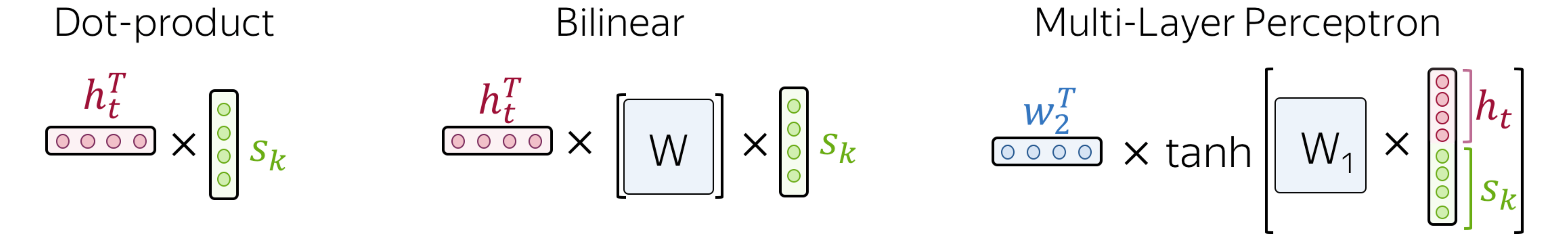
Fully differentiable, thus trainable!

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“)





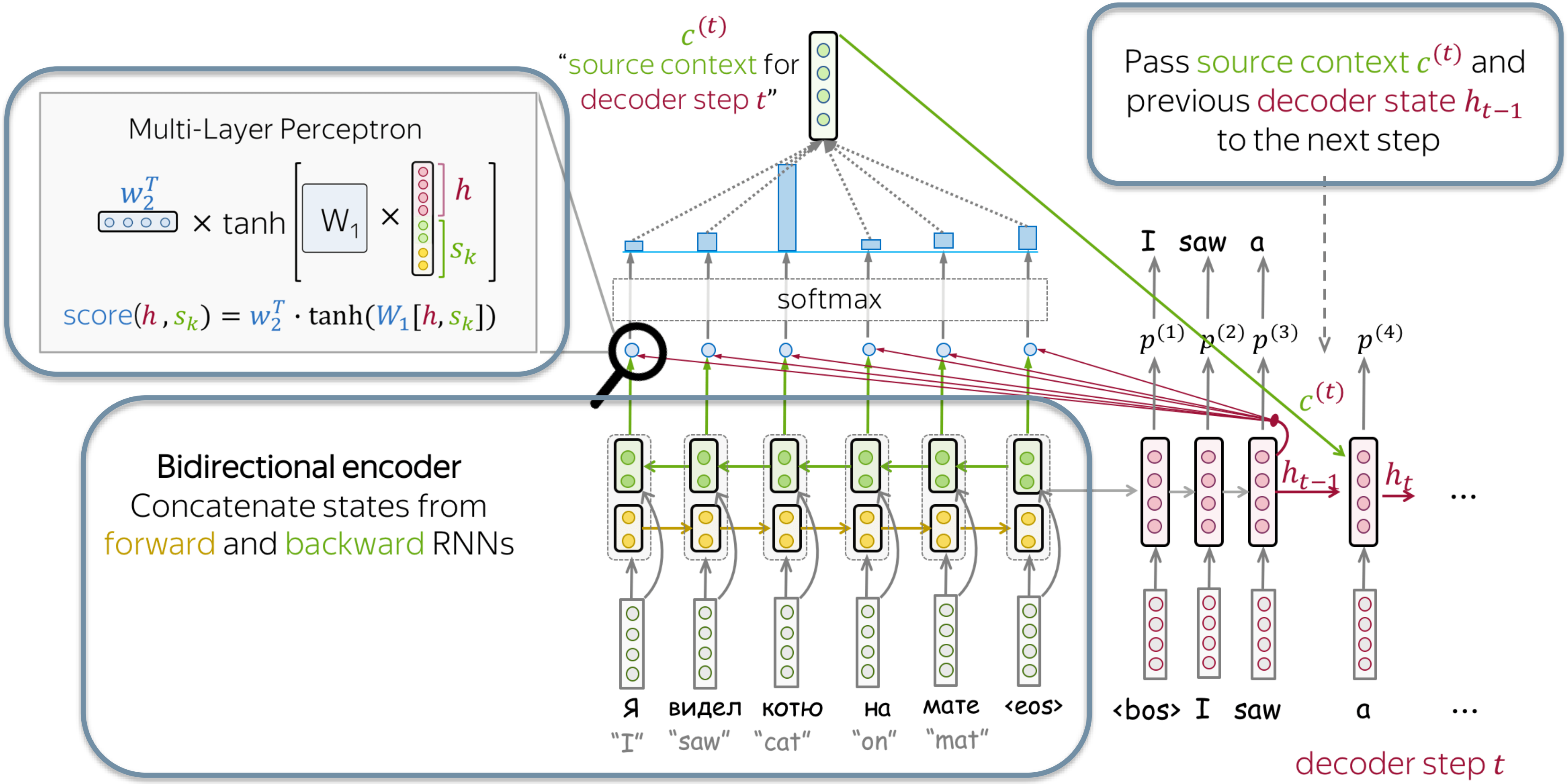
score$(h_t, s_k) = h_t^T \, s_k$

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

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

Bahdanau Attention Model

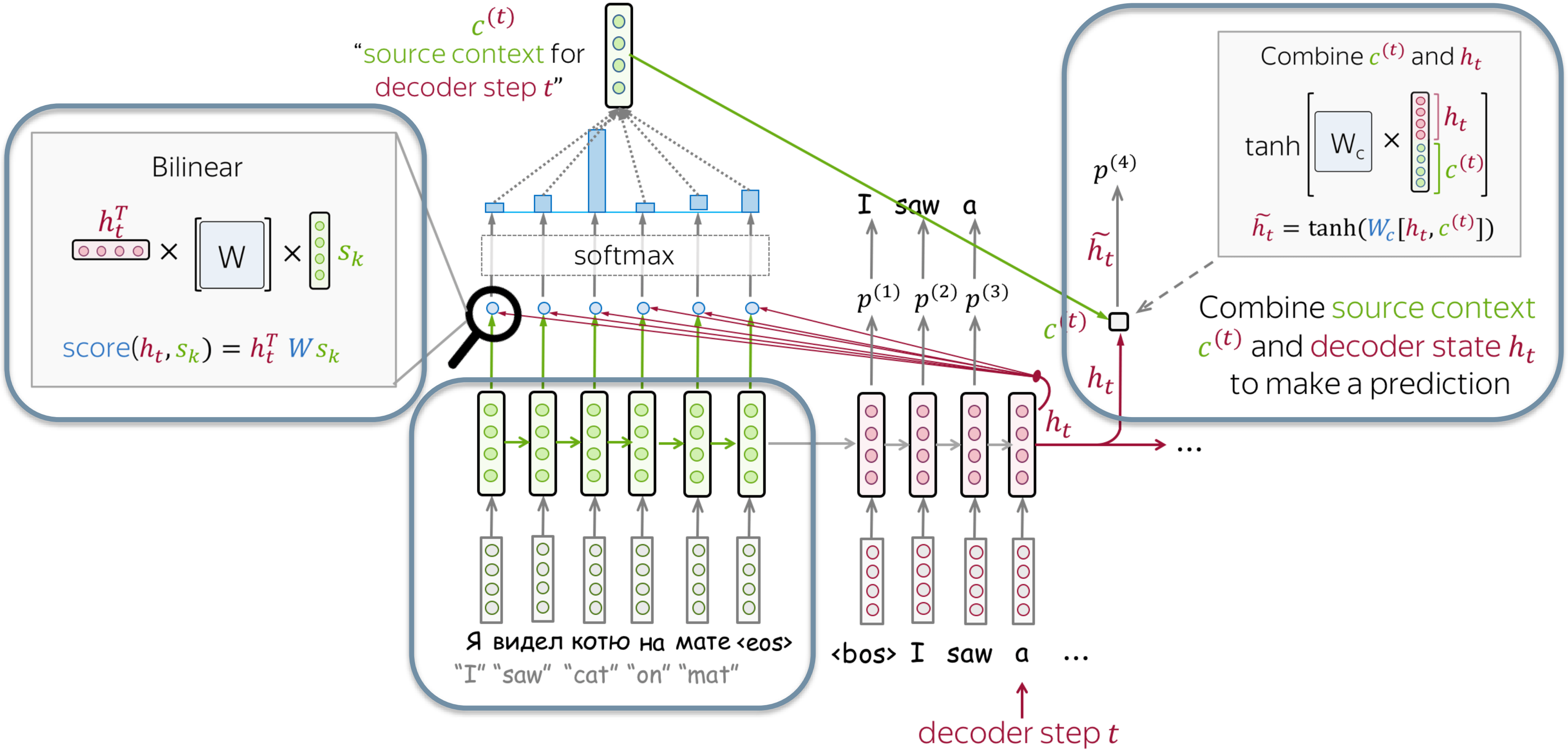
Proposed as part of the original Bahdanau model



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

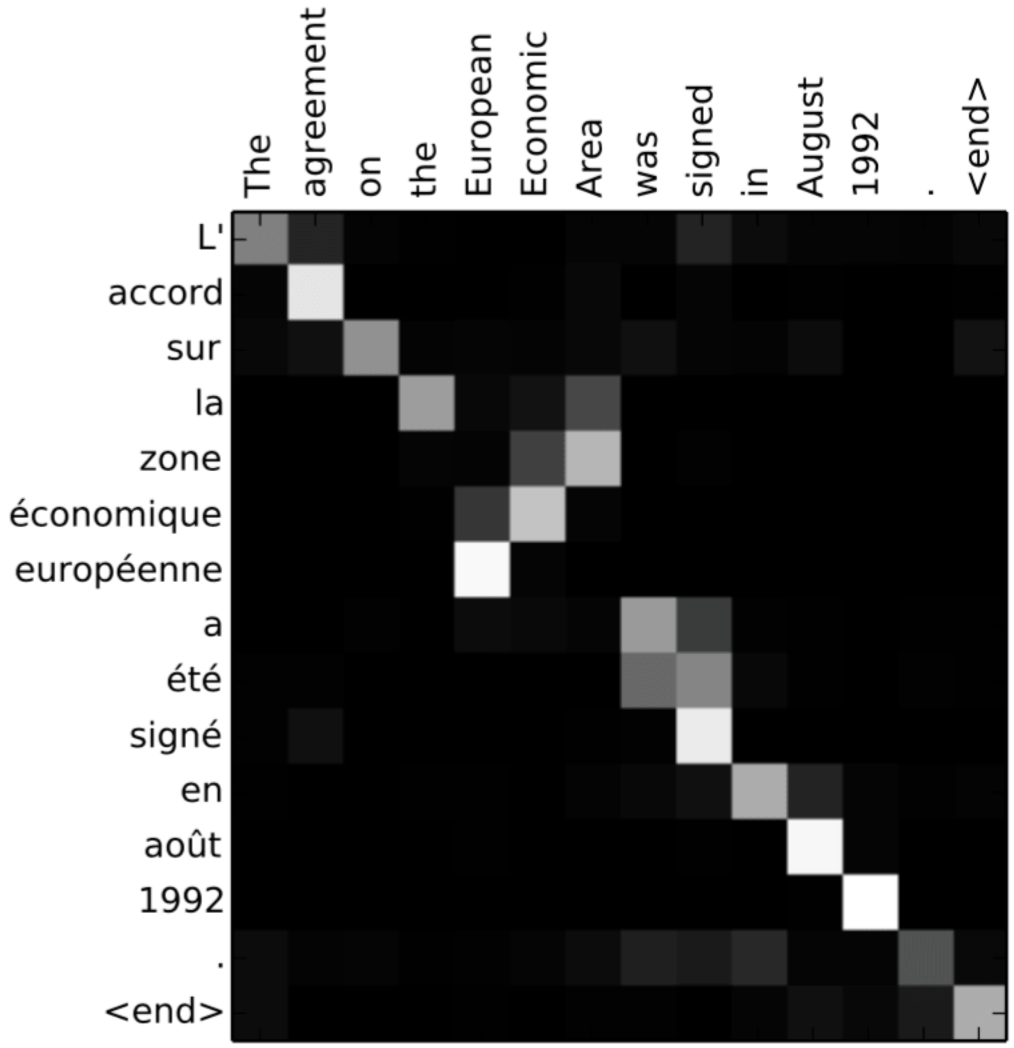
Luong Attention Model

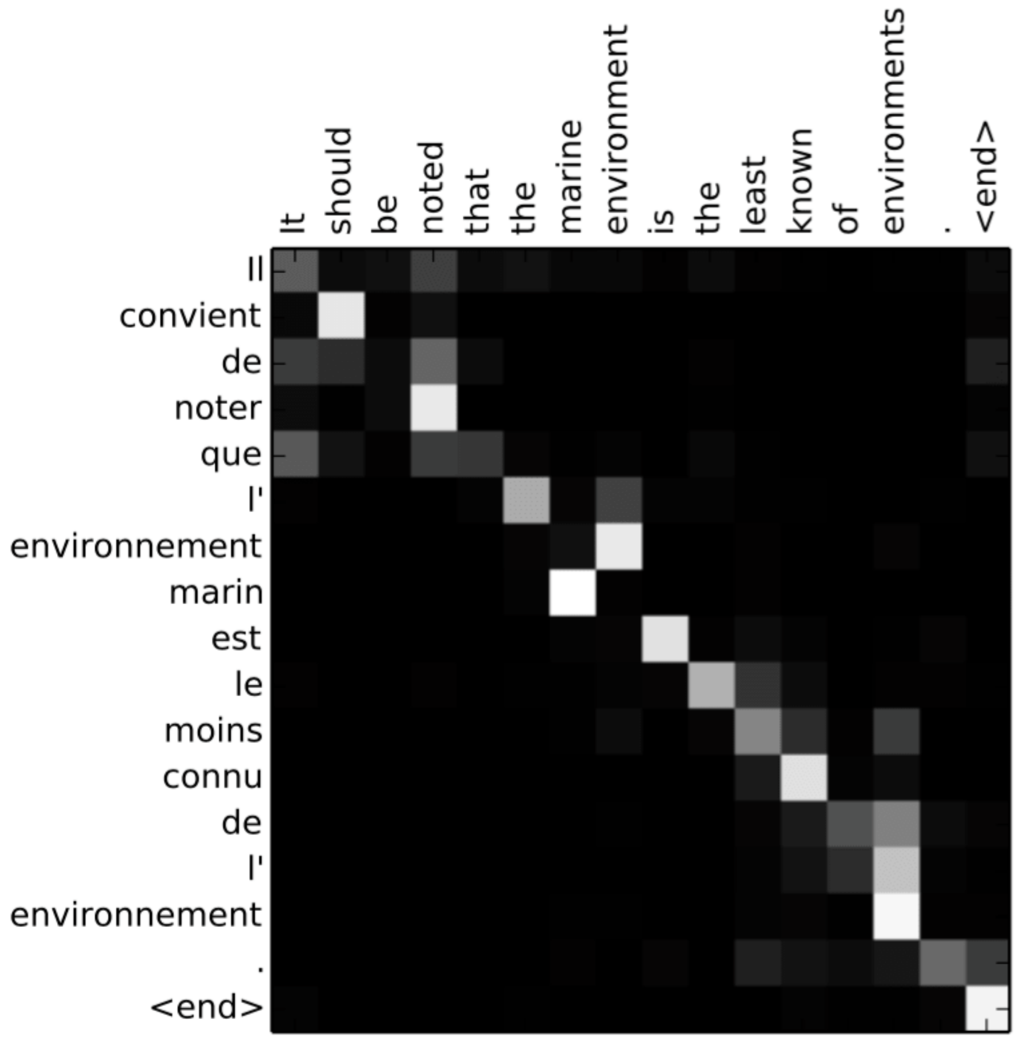
Proposed as part of the original Luong model



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

Attention learns soft sentence alignment …



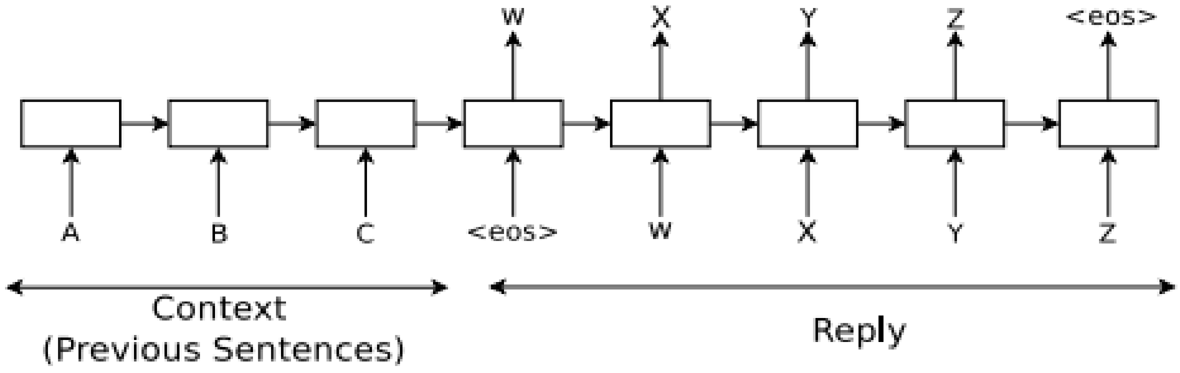


Neural Machine Translation by Jointly Learning to Align and Translate: https://arxiv.org/pdf/1409.0473.pdf

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*](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.

We focus on these here!

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.

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

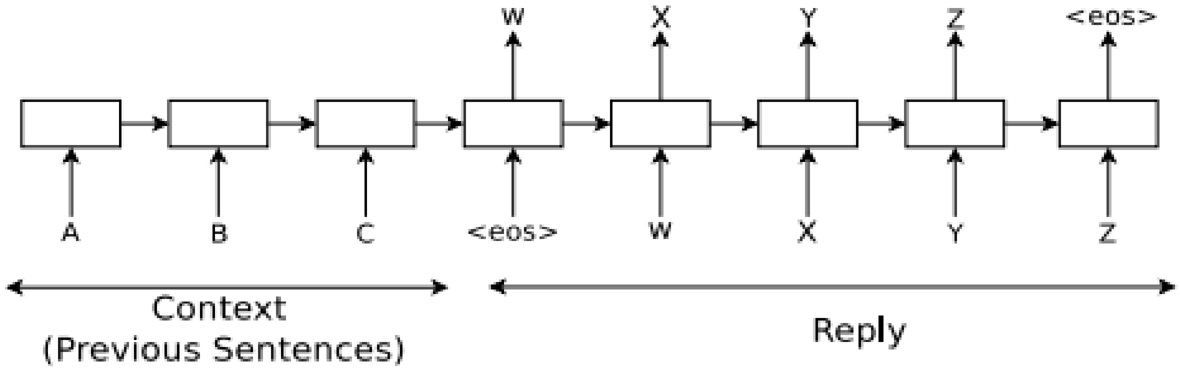
- Multi-turn: the input vector is built by considering a multi-turn conversational context, containing also incoming question.

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

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

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

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

This happens within prefix tokens …

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!

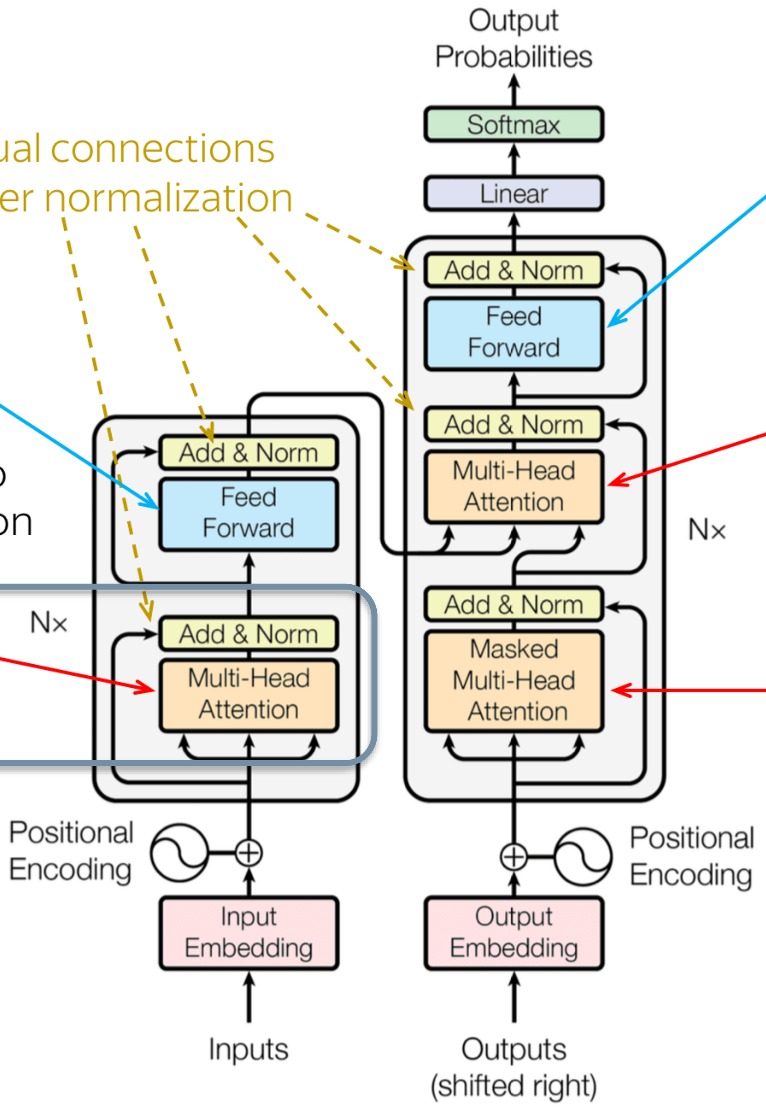
Residual connections
and layer normalization

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

1

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

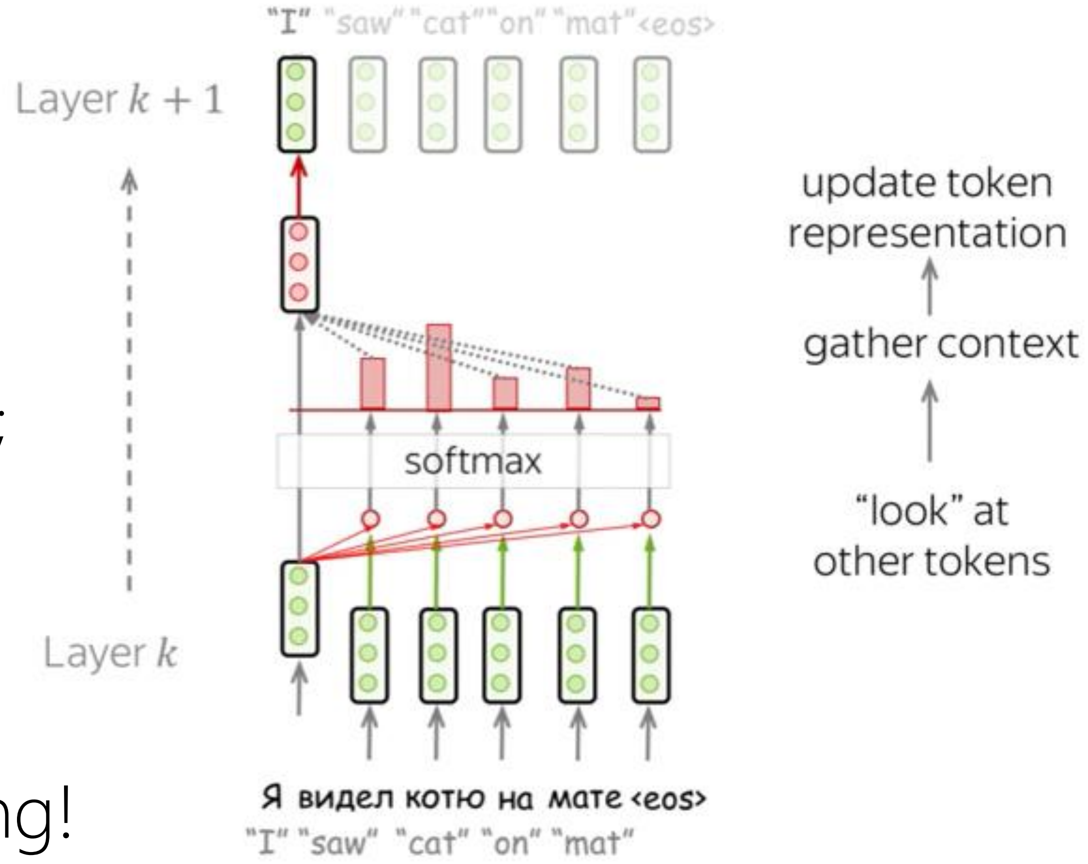
Decoder self-attention (masked):
tokens look at the previous tokens

queries, keys, values are computed
from decoder states

Attention Is All You Need https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

The Self-Attention Idea

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



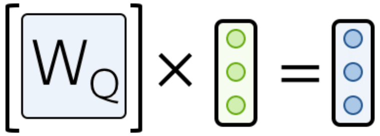
This is implemented via:

- 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!

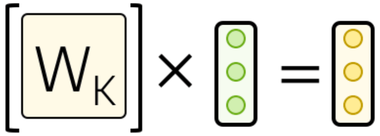
Attention Is All You Need https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Query, Key, Value …



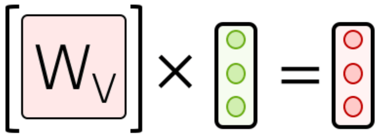
Query: vector **from** which the attention is looking

"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!"

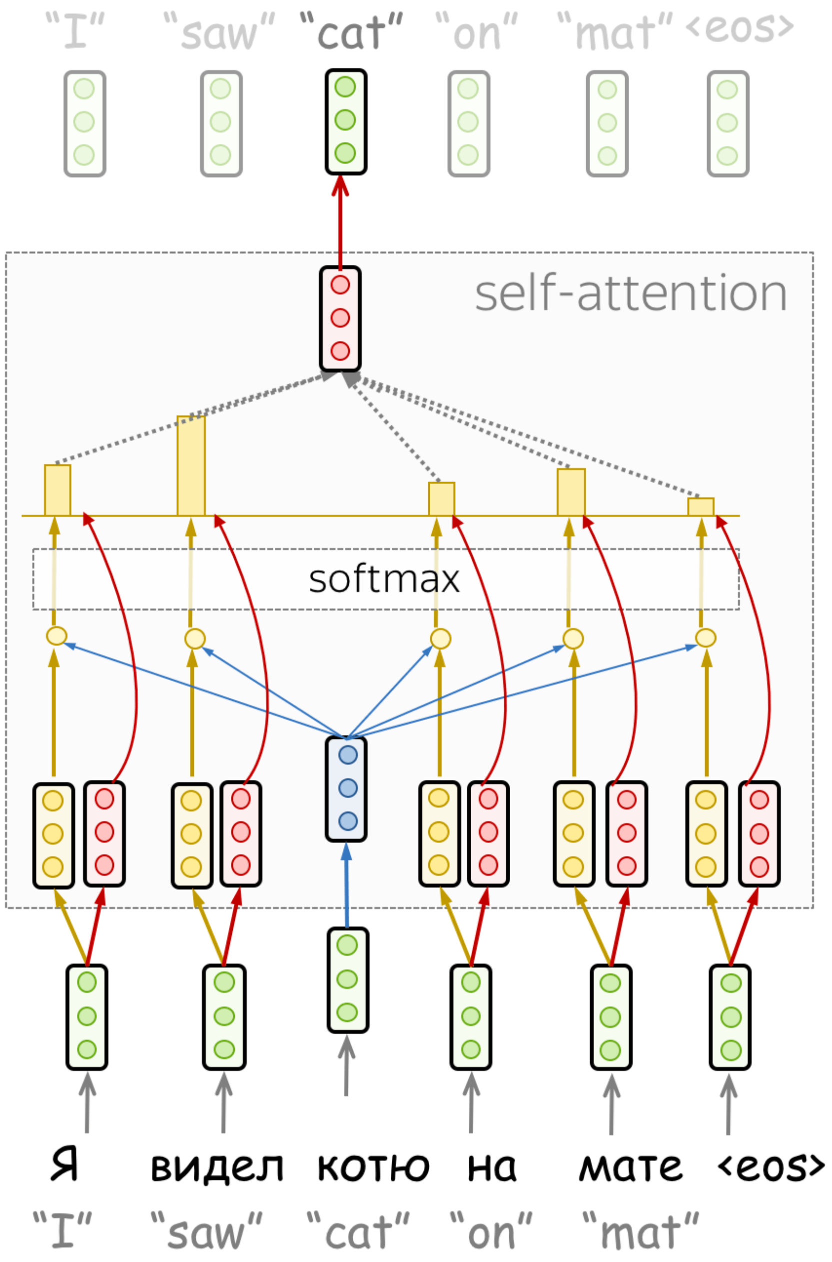


Value: their weighted sum is attention output

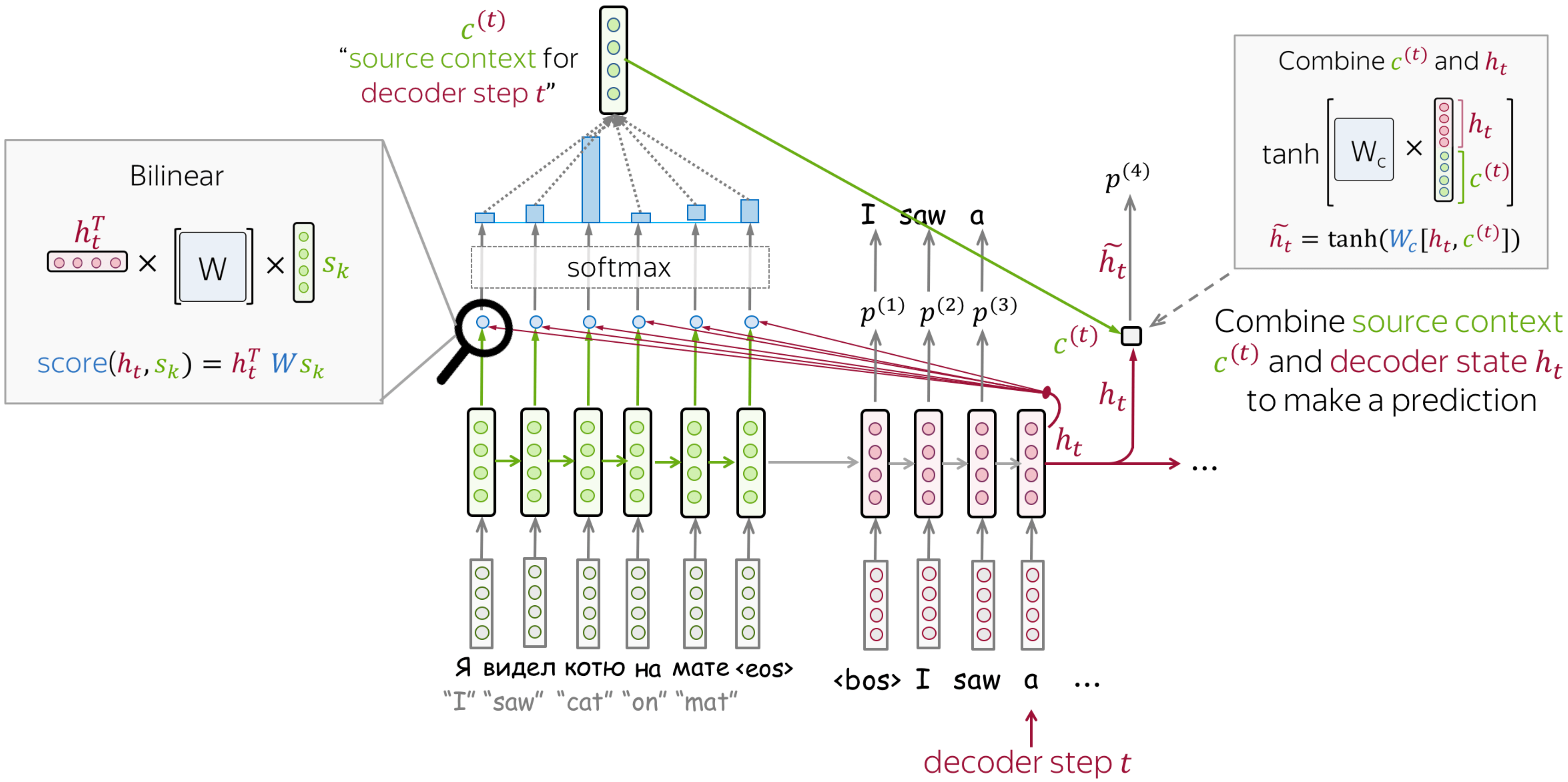
"Here's the information I have!"

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

from (q) — to (k, v) — vector dimensionality of K, V (d_k)

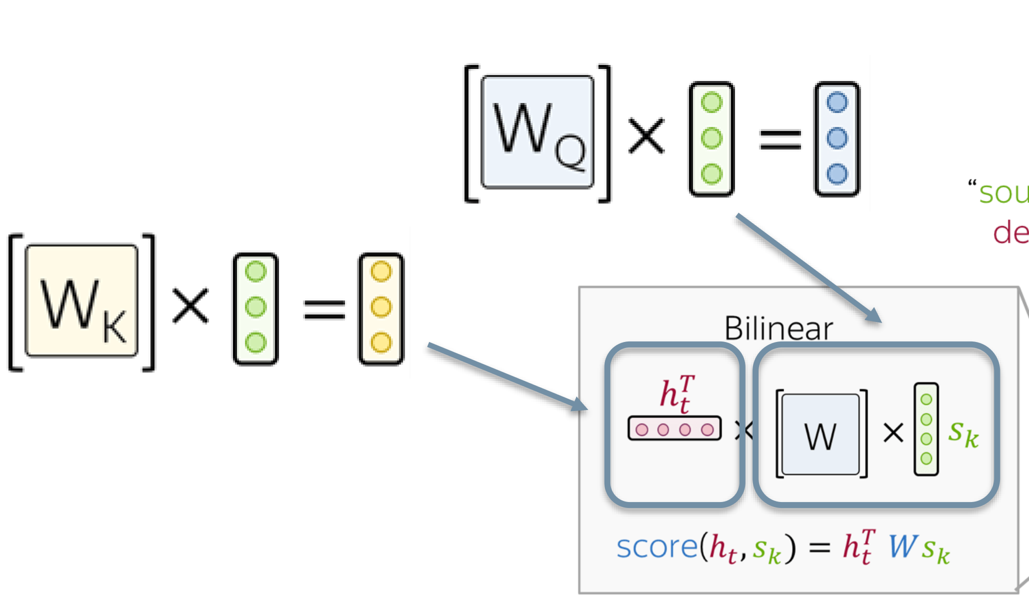


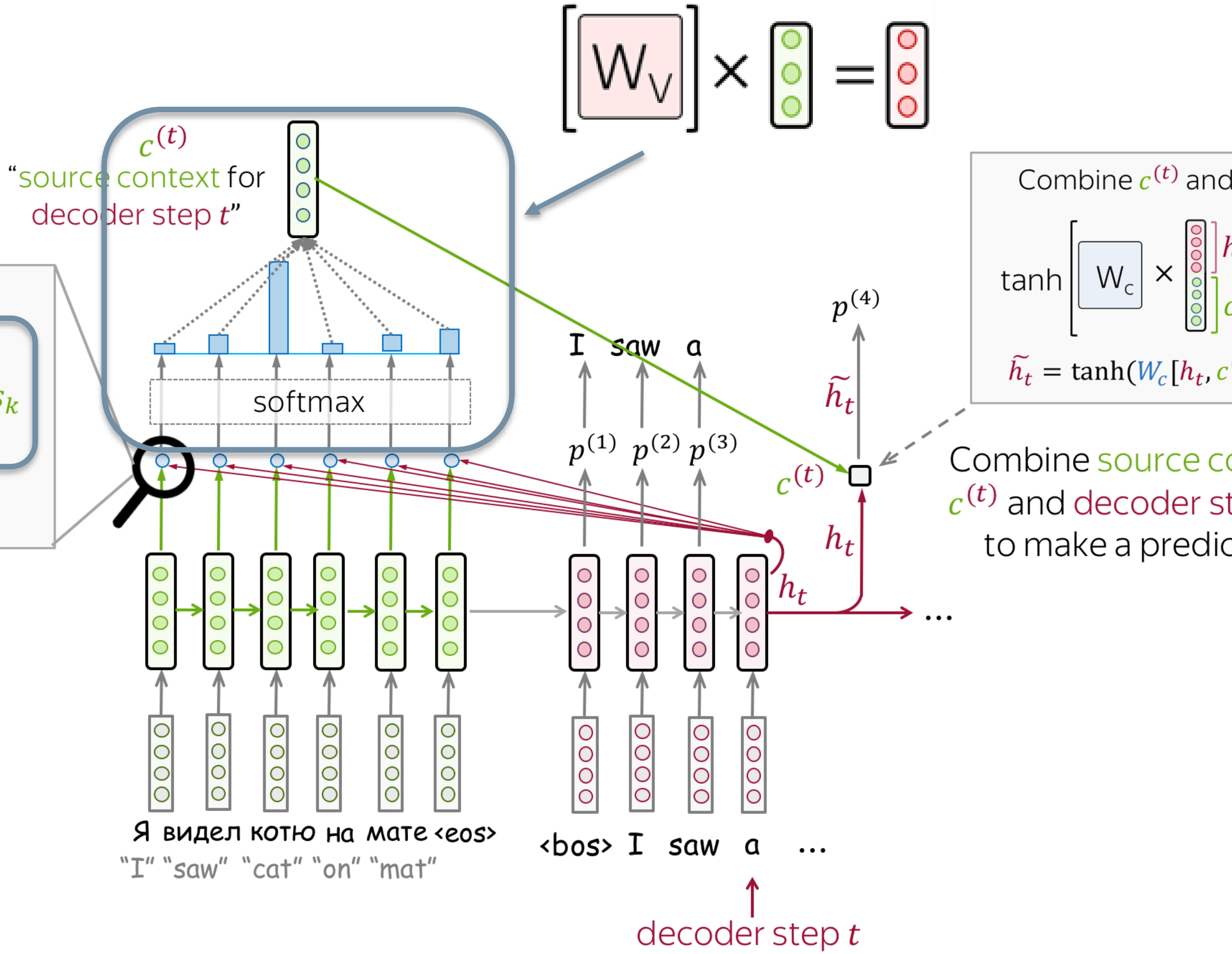
Recall Luong Attention Model



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

Recall Luong Attention Model





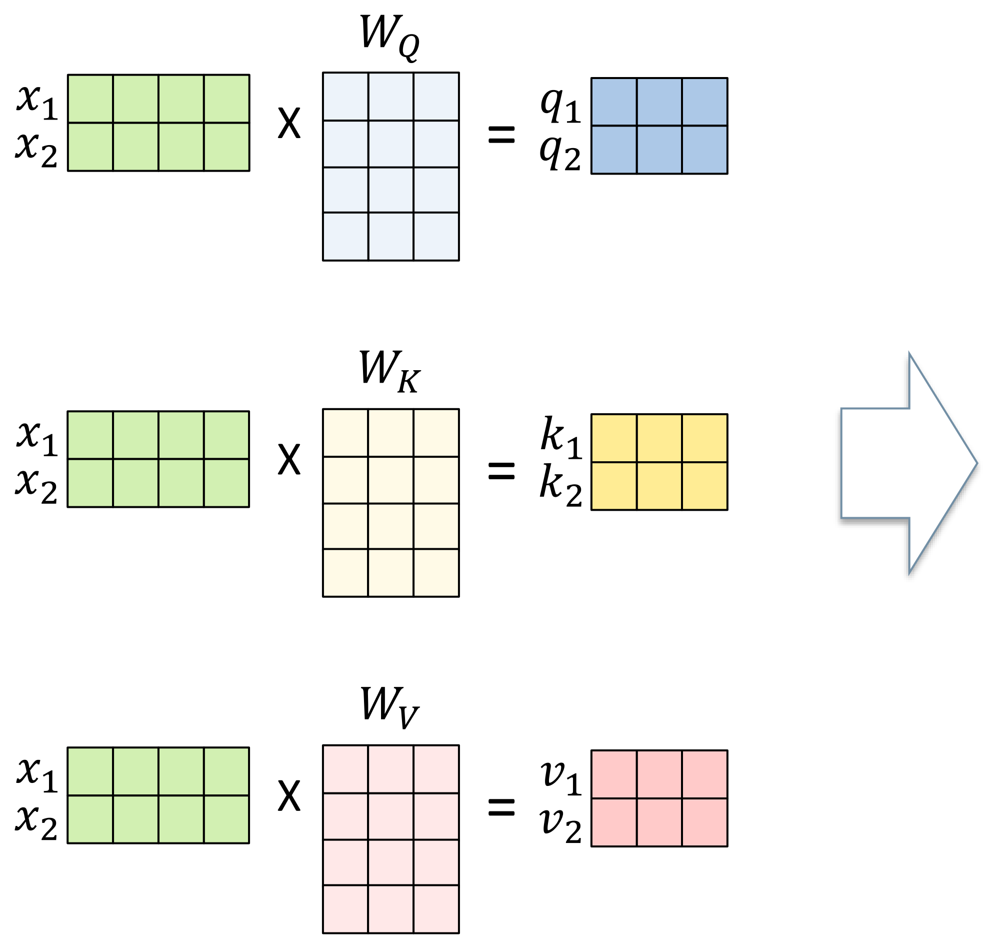
$$Attention(q, k, v) = \overbrace{softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)}^{\text{Attention weights}} v$$

from → q; to → k, v

d_k: vector dimensionality of K, V

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

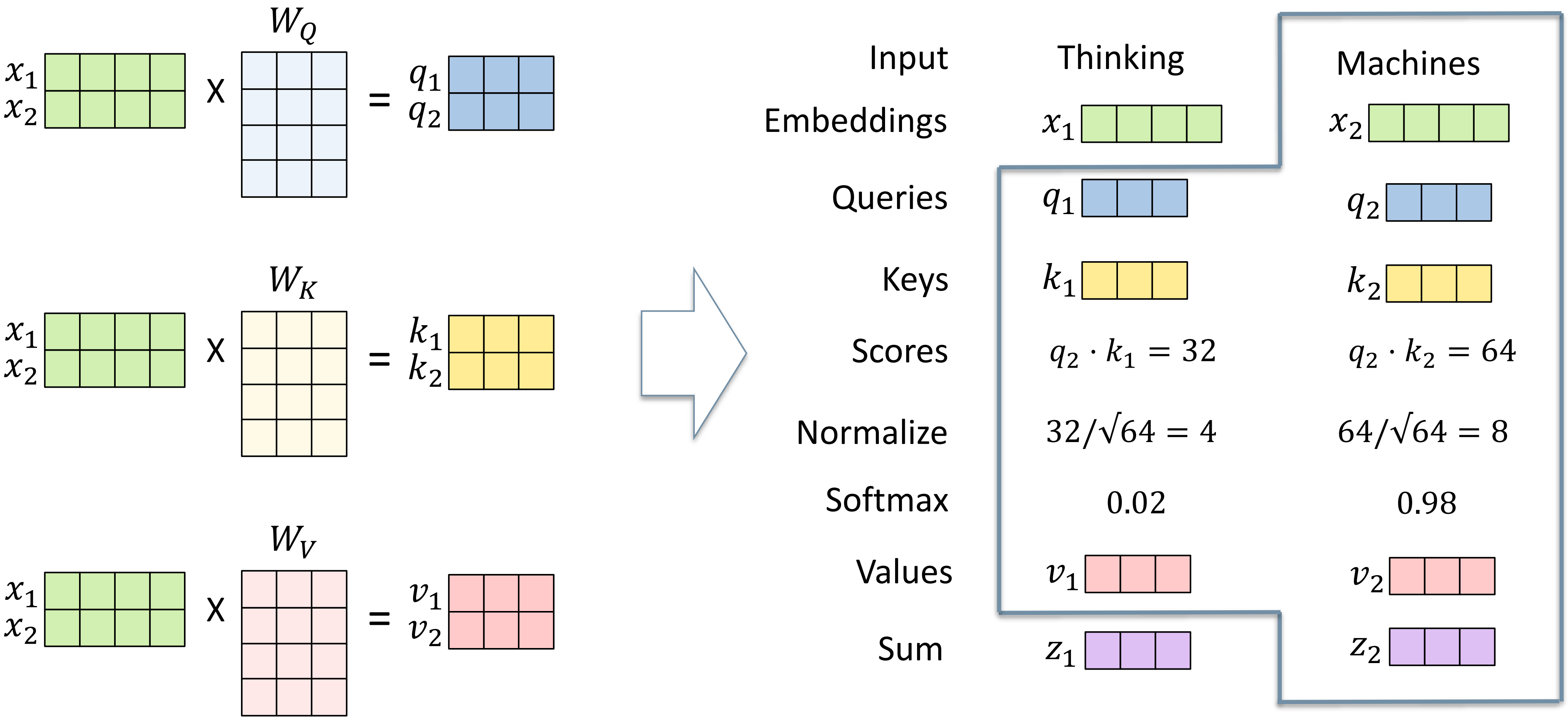
Let's Play Some Linear Algebra!



Input	Thinking	Machines
Embeddings	x_1	x_2
Queries	q_1	q_2
Keys	k_1	k_2
Scores	$q_1 \cdot k_1 = 112$	$q_1 \cdot k_2 = 96$
Normalize	$112/\sqrt{64} = 14$	$96/\sqrt{64} = 12$
Softmax	0.88	0.12
Values	v_1	v_2
Sum	z_1	

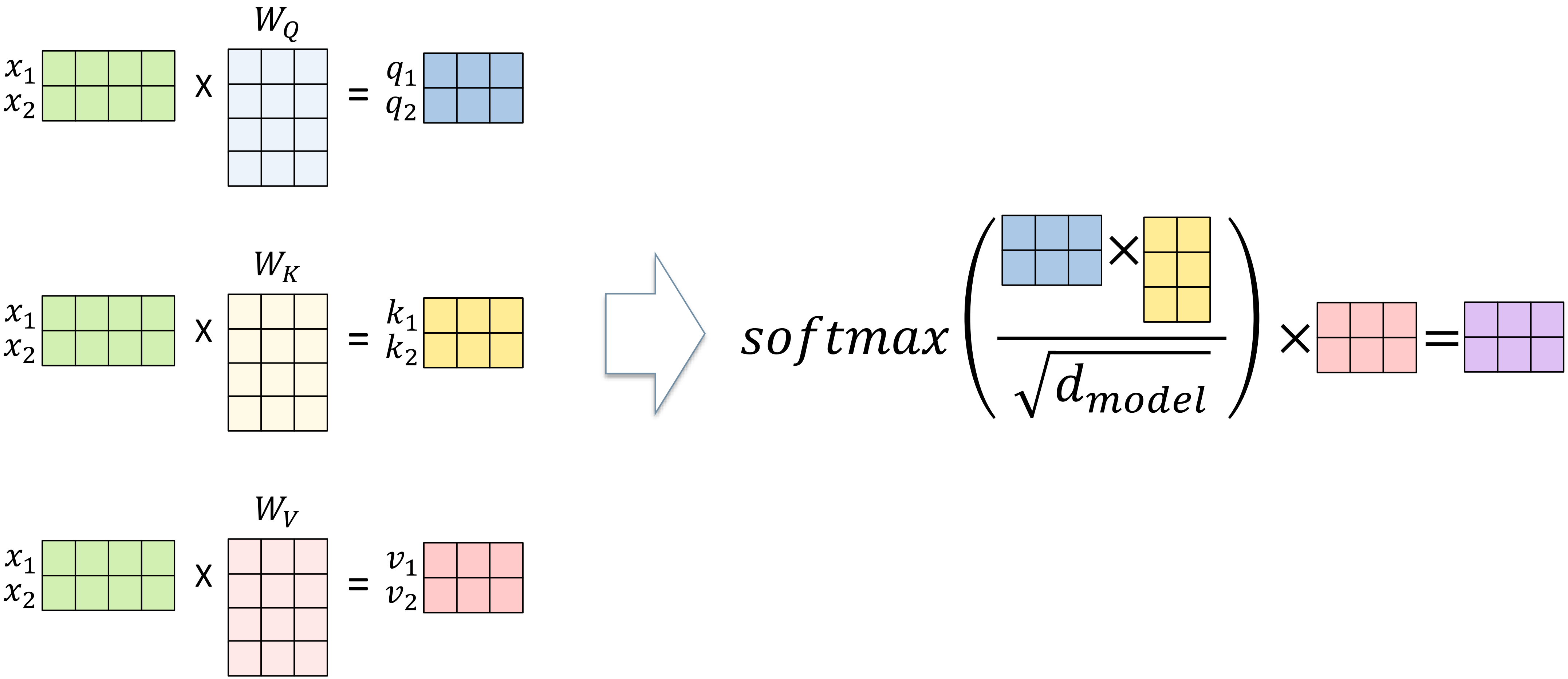
The Illustrated Transformer http://jalammar.github.io/illustrated-transformer/

Let's Play Some Linear Algebra!



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

Let's Play Some Linear Algebra!



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

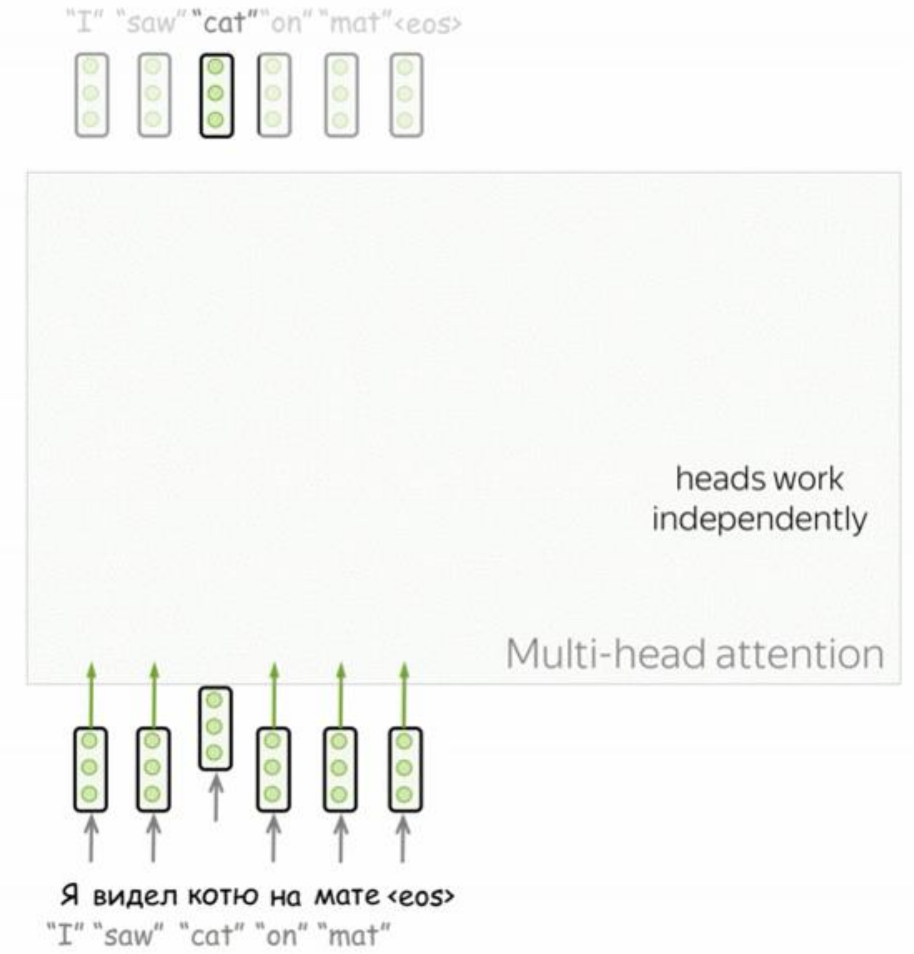
Multi-Head Attention

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 defines by verbs
- …

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

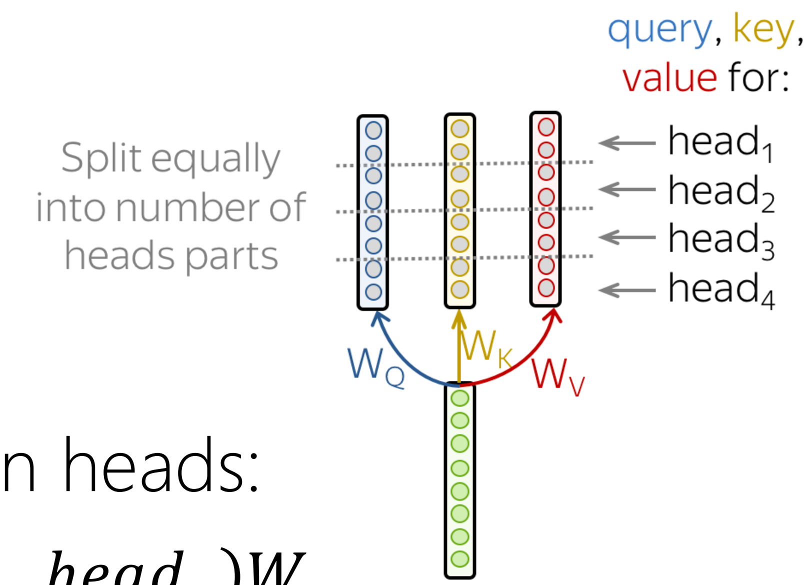
Check Mark Carman lecture for this!



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 defines by verbs



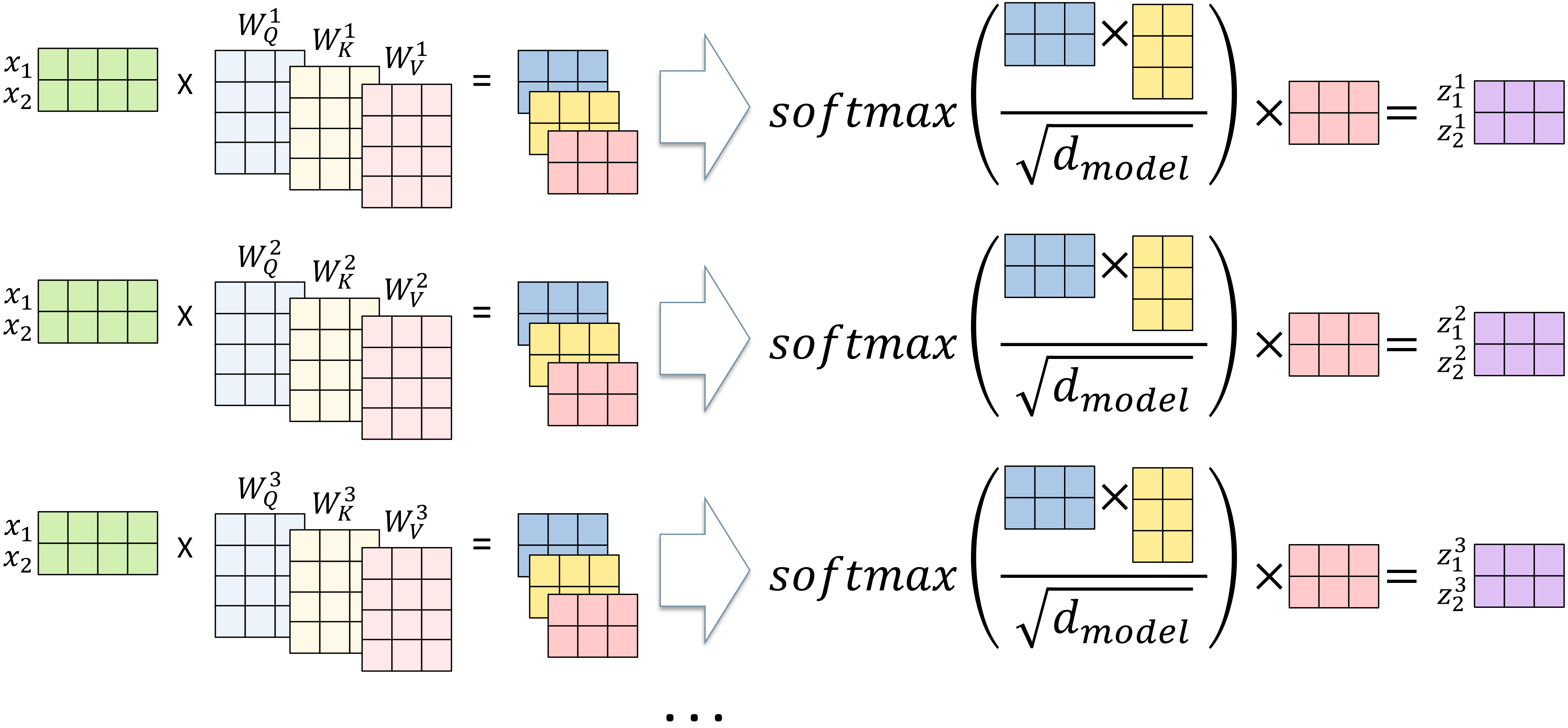
Implemented as concatenation of several attention heads:

$$MultiHead(Q,K,V) = Concat(head_1, head_2, \dots, head_n)W_o$$

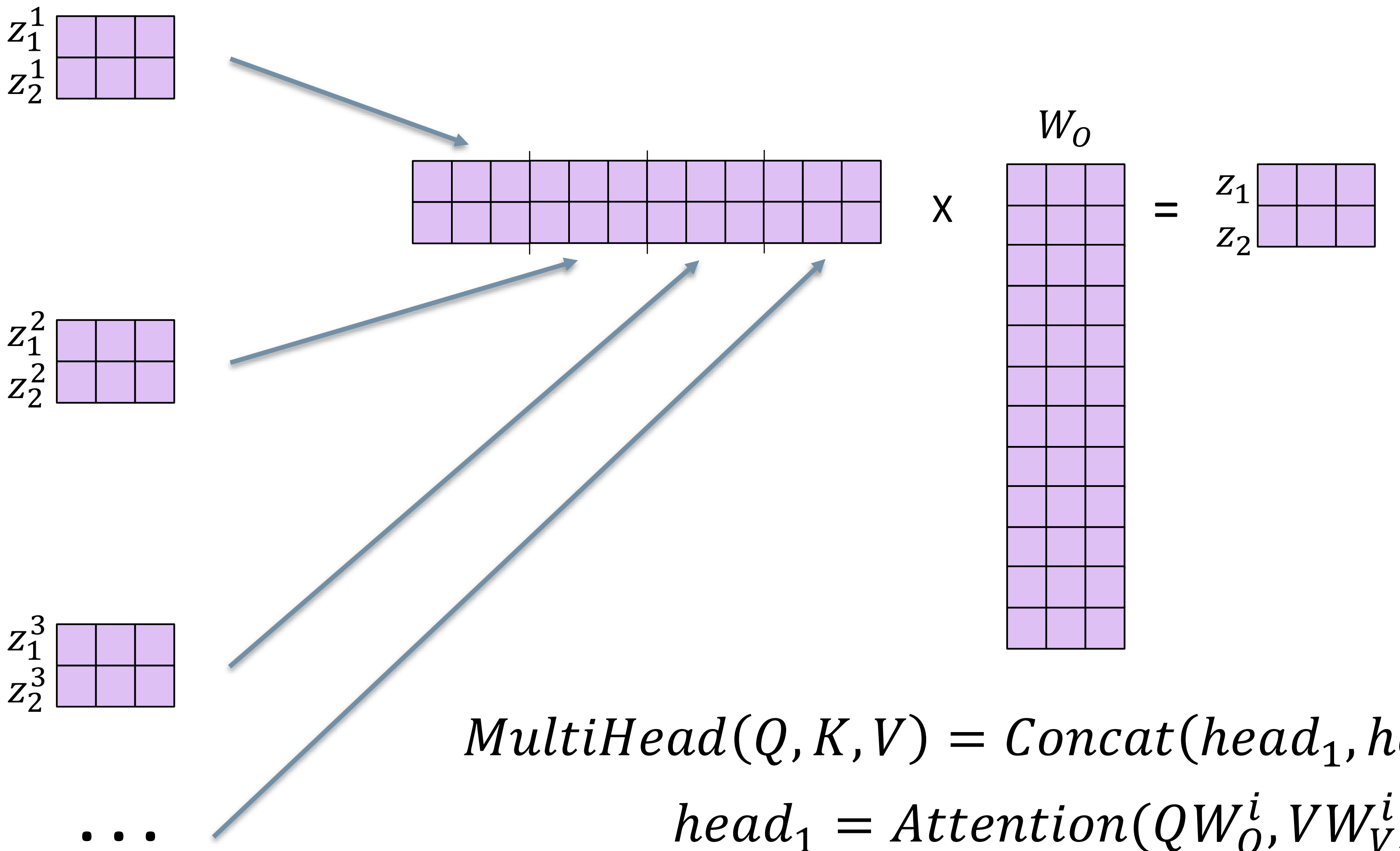
$$head_1 = Attention(QW_Q^i, VW_V^i, VW_V^i)$$

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!



$$MultiHead(Q,K,V) = Concat(head_1, head_2, \dots, head_n)W_o$$

$$head_1 = Attention(QW_Q^i, VW_V^i, VW_V^i)$$

Attention is all you need!

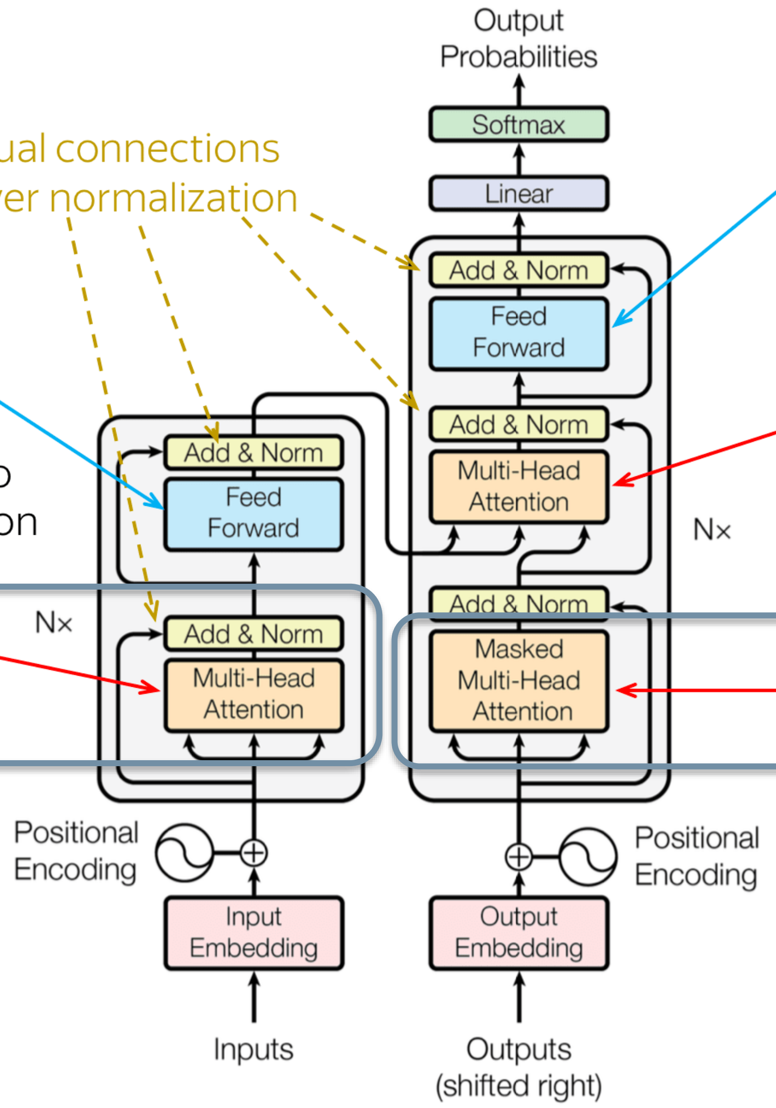
Residual connections
and layer normalization

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

1

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

2

queries, keys, values are computed
from decoder states

Attention Is All You Need https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-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)

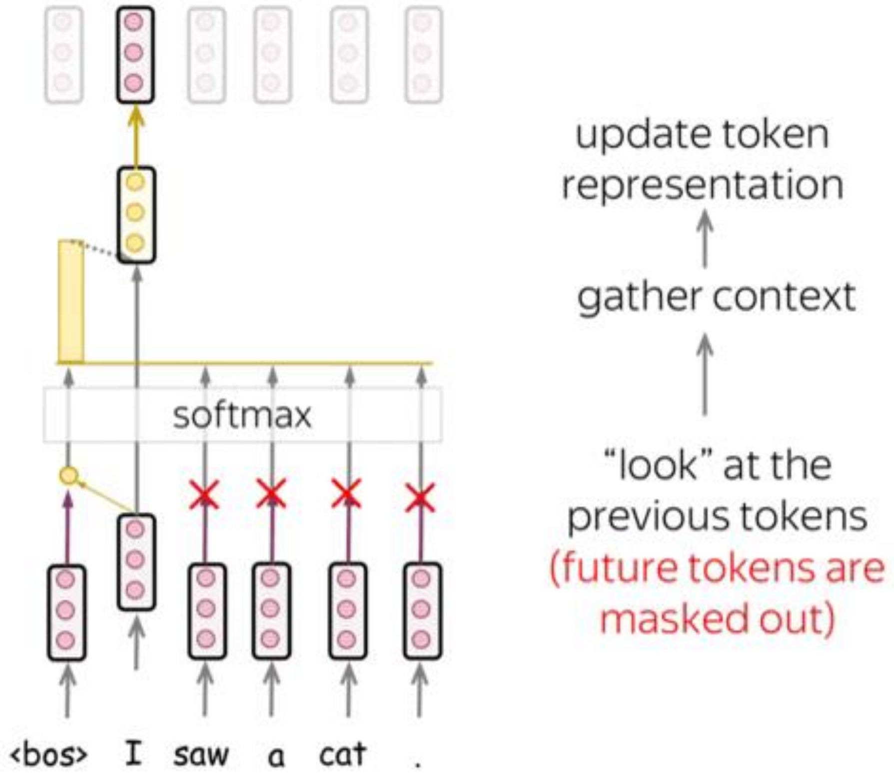
This is obtained by masking «future tokens» at training time

Masked Self-Attention

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

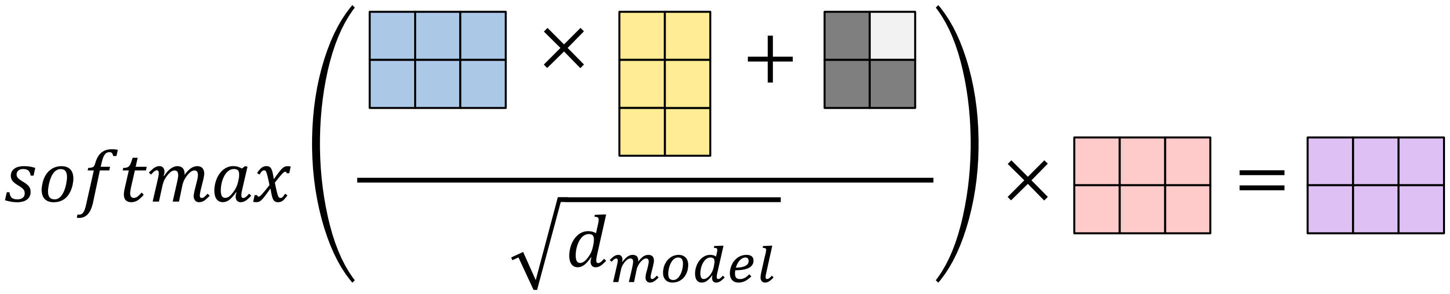
- At inference time, we generate one token at the time as we do not know the length of the sequence (no “look-ahead” problem)
- At training time, we know already the entire output sequence and we want to process it in parallel (“look ahead” problem)

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



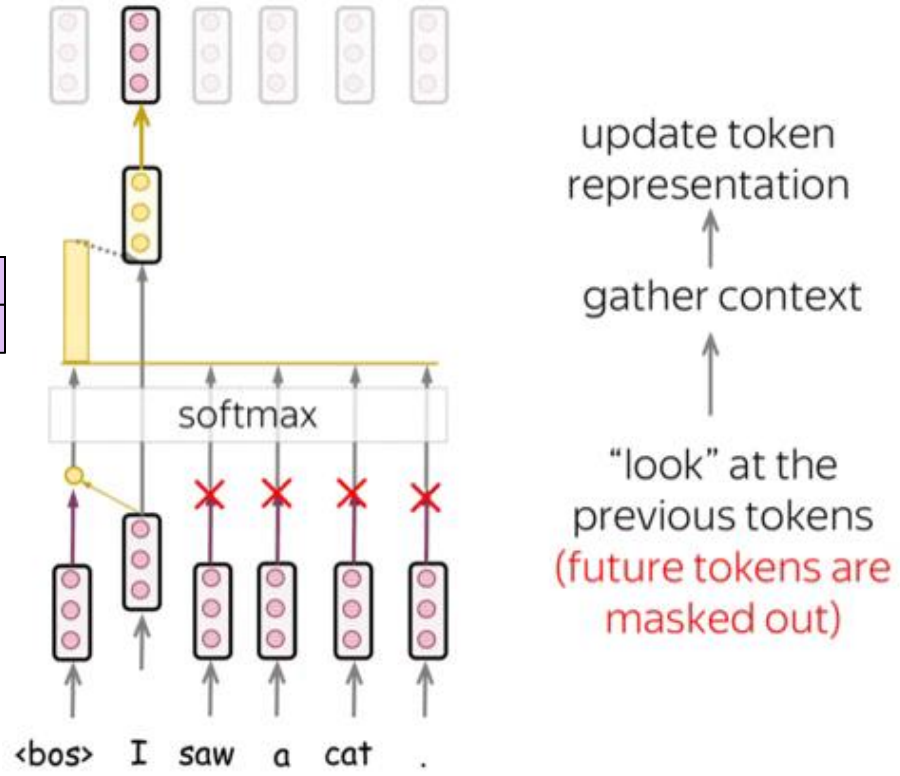
Masked Self-Attention

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



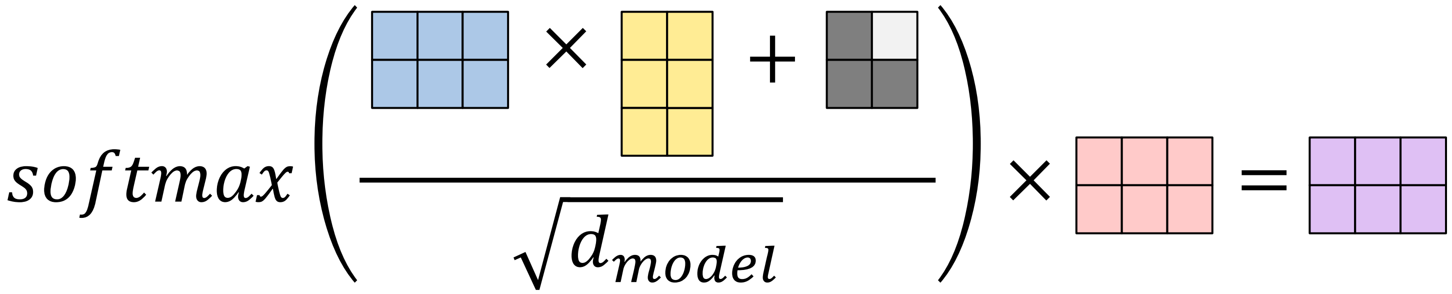
$$Mask = \begin{bmatrix} 0 & -\inf \\ 0 & 0 \end{bmatrix}$$

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

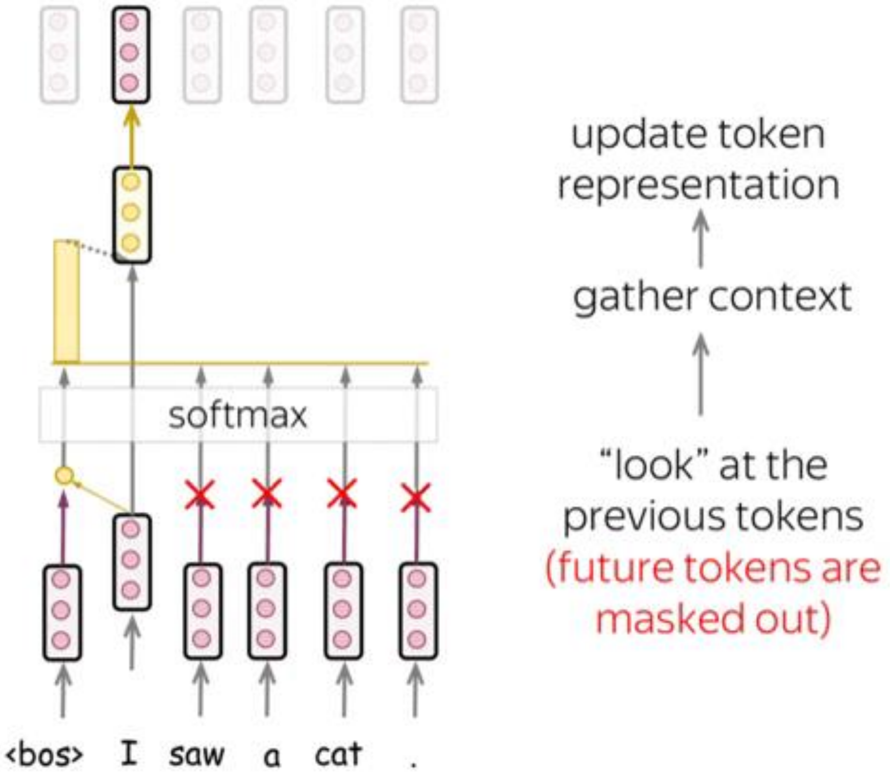


Masked Self-Attention

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

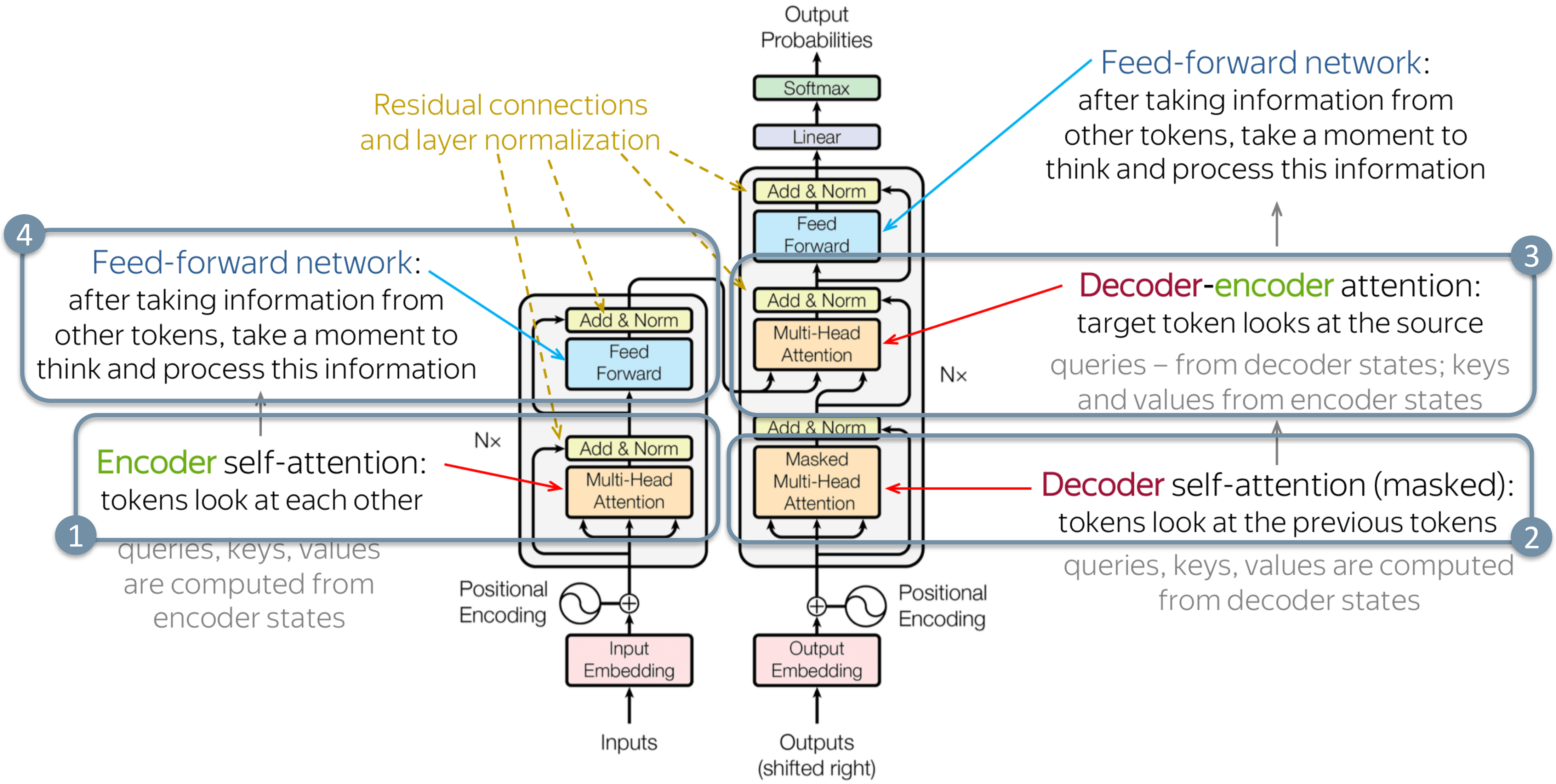


$$Mask = \begin{bmatrix} 0 & -\inf \\ 0 & 0 \end{bmatrix}$$



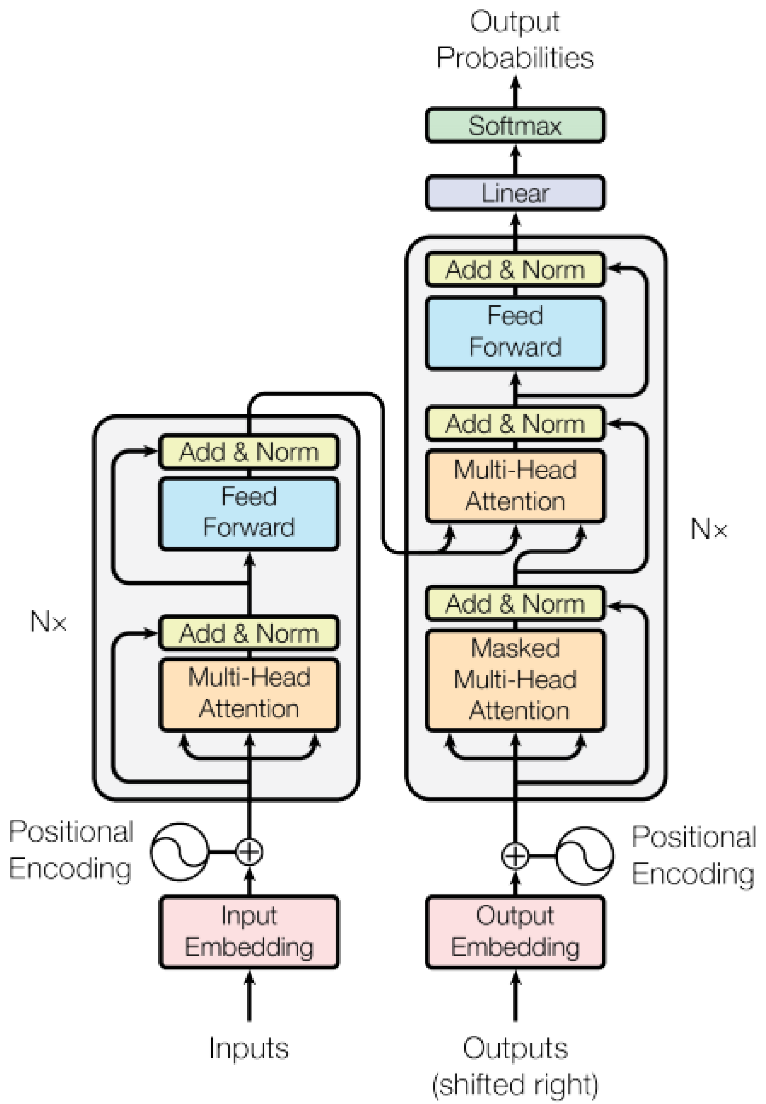
RNN training is <u>*O(len(source) + len(target))*</u>
Transformer training is <u>*O(1) (with respect to [fixed] sequences' length)*</u>

Attention is all you need!



Attention Is All You Need [*https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf*](https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf)

More Transformers' Components …



Attention Is All You Need https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

More Transformers' Components …

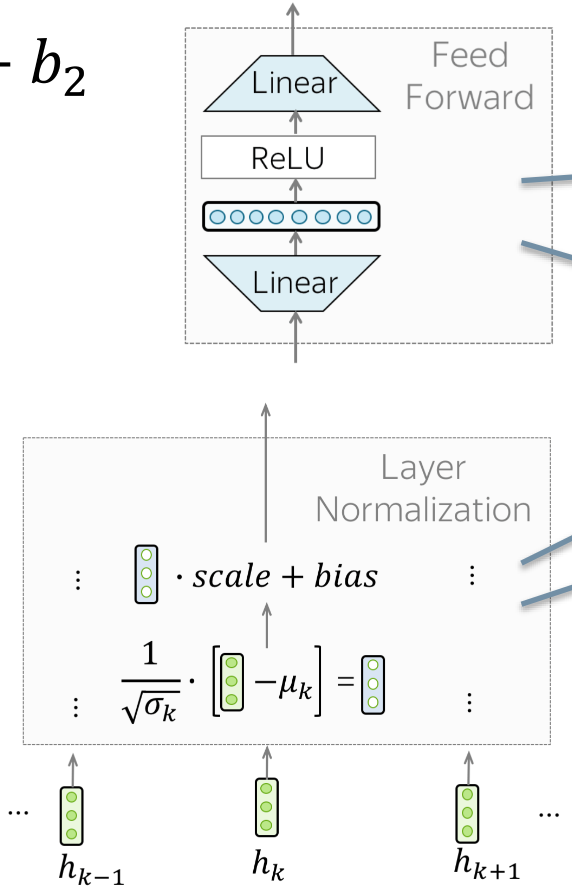
Feed forward blocks with two linear layers & ReLU activations

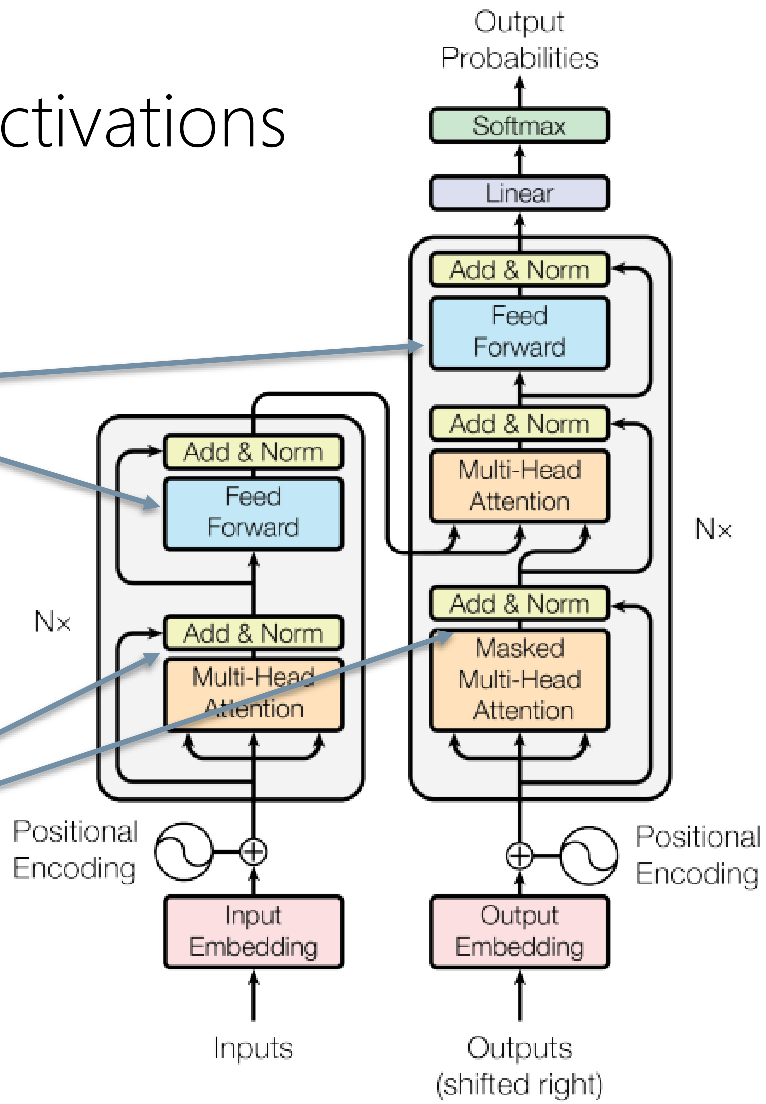
$$FFN(x) = \max\left(0, x_{W_1} + b_1\right) W_2 + b_2$$

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 …

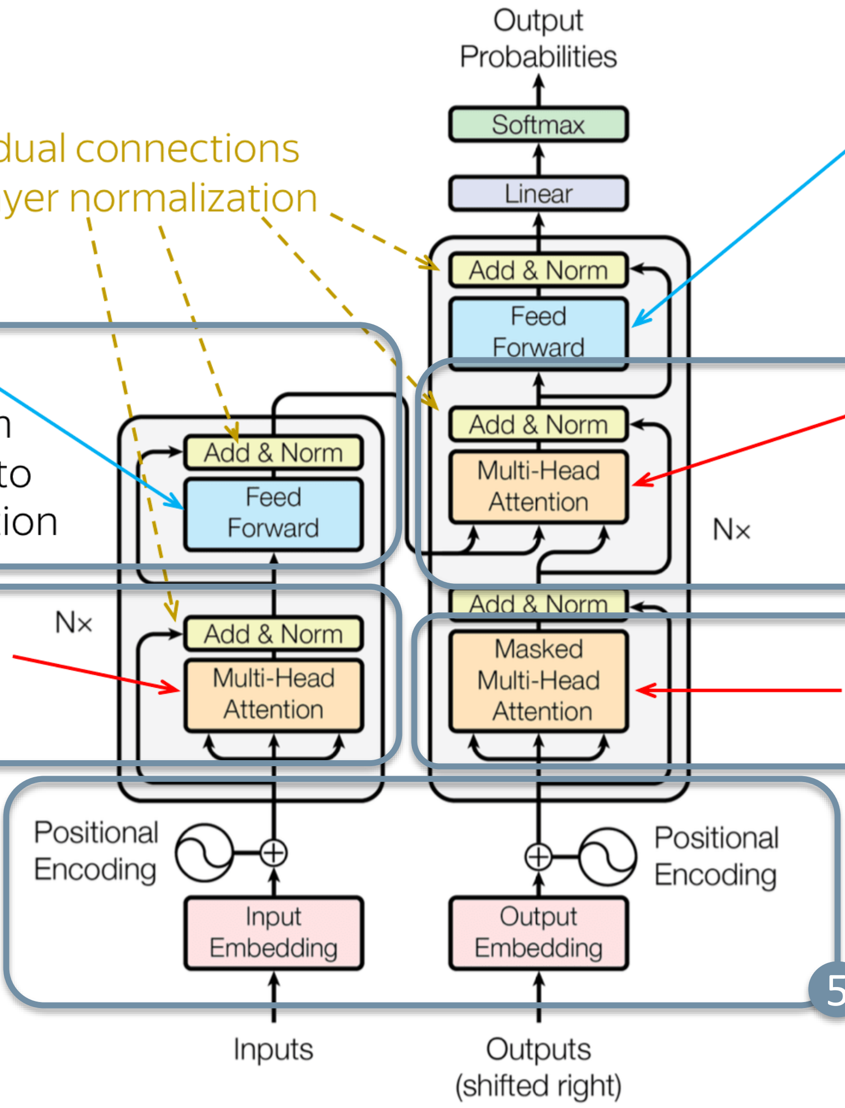




Attention Is All You Need https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Attention is all you need!

Residual connections
and layer normalization



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

4
Feed-forward network:
after taking information from
other tokens, take a moment to
think and process this information

3
Decoder-encoder attention:
target token looks at the source

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

1
Encoder self-attention:
tokens look at each other

queries, keys, values
are computed from
encoder states

2
Decoder self-attention (masked):
tokens look at the previous tokens

queries, keys, values are computed
from decoder states

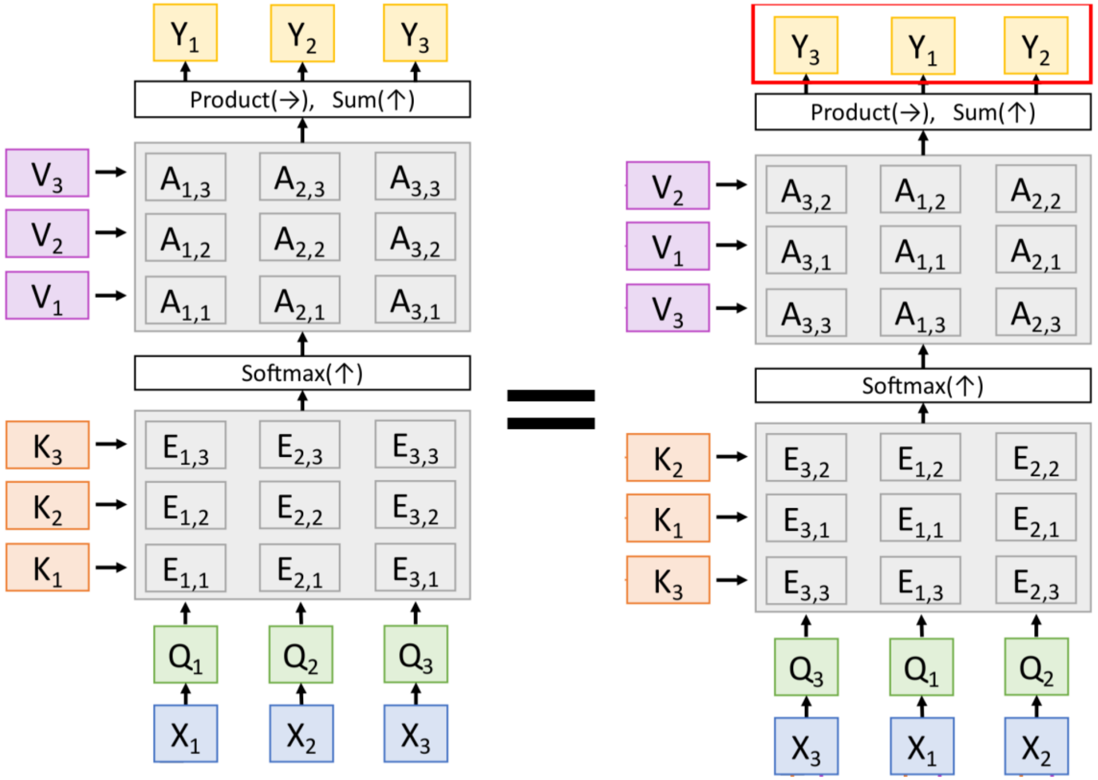
Attention Is All You Need https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

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

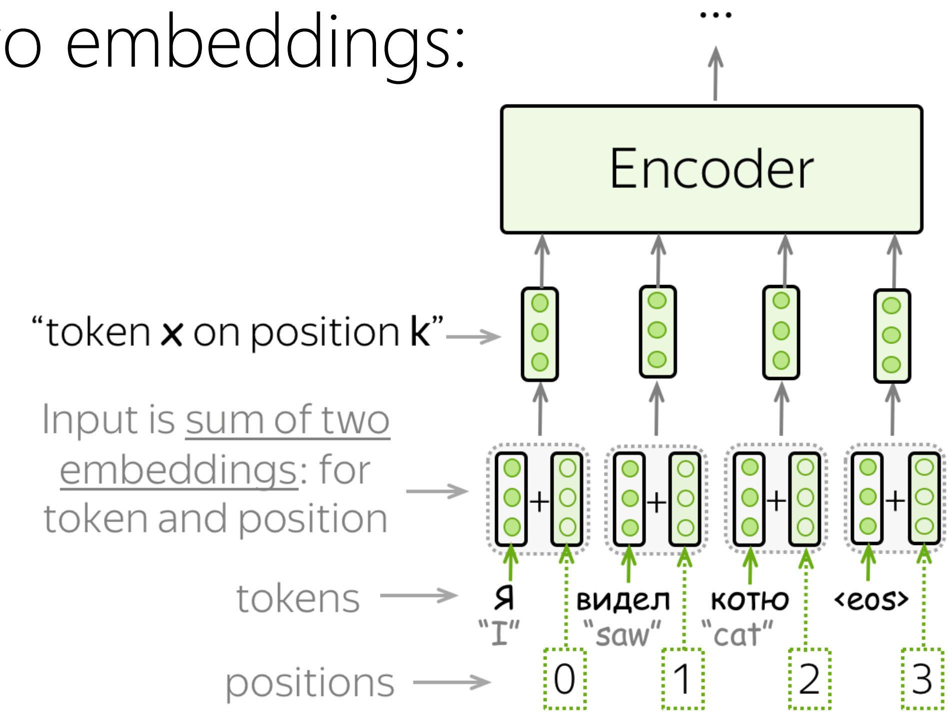
Positional encoding is used to make self-attention depend also on the position of the input



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:

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

where *pos* is position, *i* is the vector dimension, and d_{model} the input size.

Authors have tried learned encodings but did not improve ...

Attention Is All You Need https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Positional Encoding

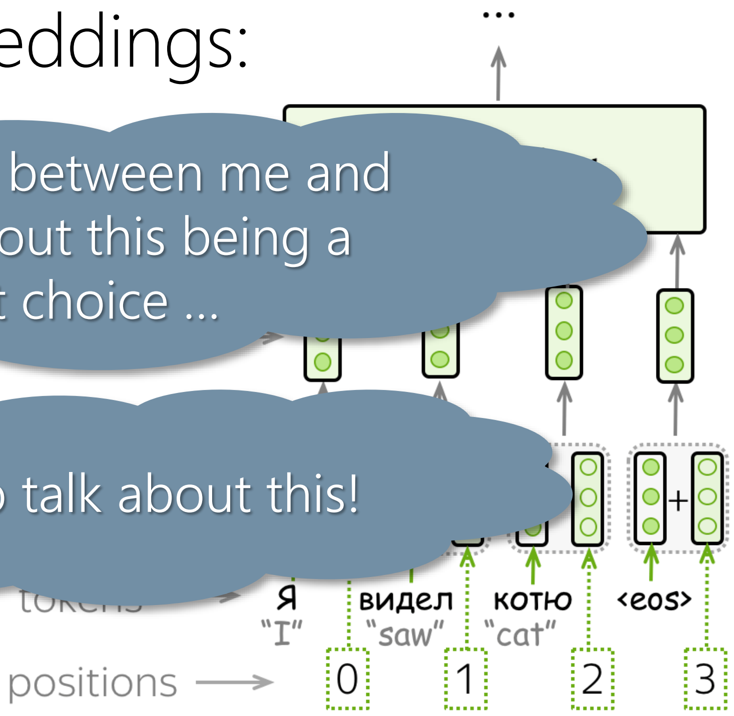
A token input representation is the sum of two embeddings:

- 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 encoding

… not to talk about this!



$$\boxed{\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_{model}}\right) \\ PE_{(pos,2i+1)} &= cos\left(pos/10000^{2i/d_{model}}\right) \end{aligned}}$$

where *pos* is position, *i* is the vector dimension, and d_{model} the input size.

Authors have tried learned encodings but did not improve …

Attention Is All You Need https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-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.

Binary digits are a waste in the land of float …

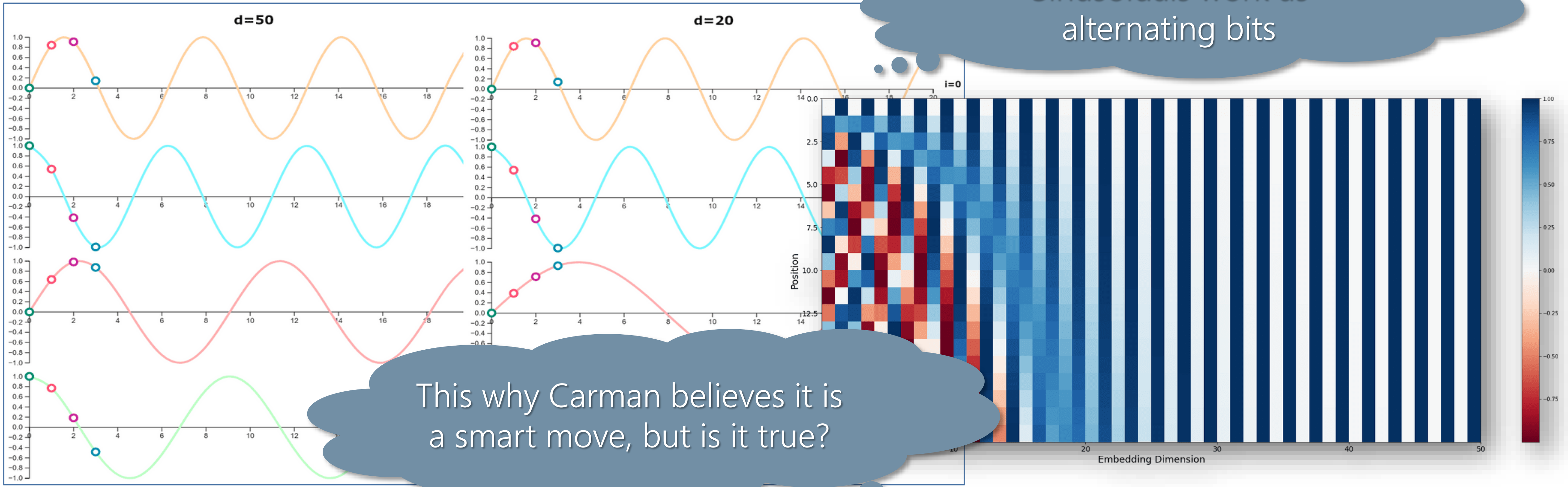
Let's $\boldsymbol{pos}$ be the position in an input sequence and $\boldsymbol{PE_{pos}} \in \mathbb{R}^{\boldsymbol{d_{model}}}$ its encoding. The encoding function $\boldsymbol{f(pos)}: \mathbb{N} \rightarrow \mathbb{R}^{\boldsymbol{d_{model}}}$ is defined as

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

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

Visualizing Transformer Position Encoding

Sinusoidals work as alternating bits



This why Carman believes it is a smart move, but is it true?

p0	p1	p2	p3	
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

		p2	p3	
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

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

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

The transformation matrix is thus independent from $\boldsymbol{pos}$ (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/

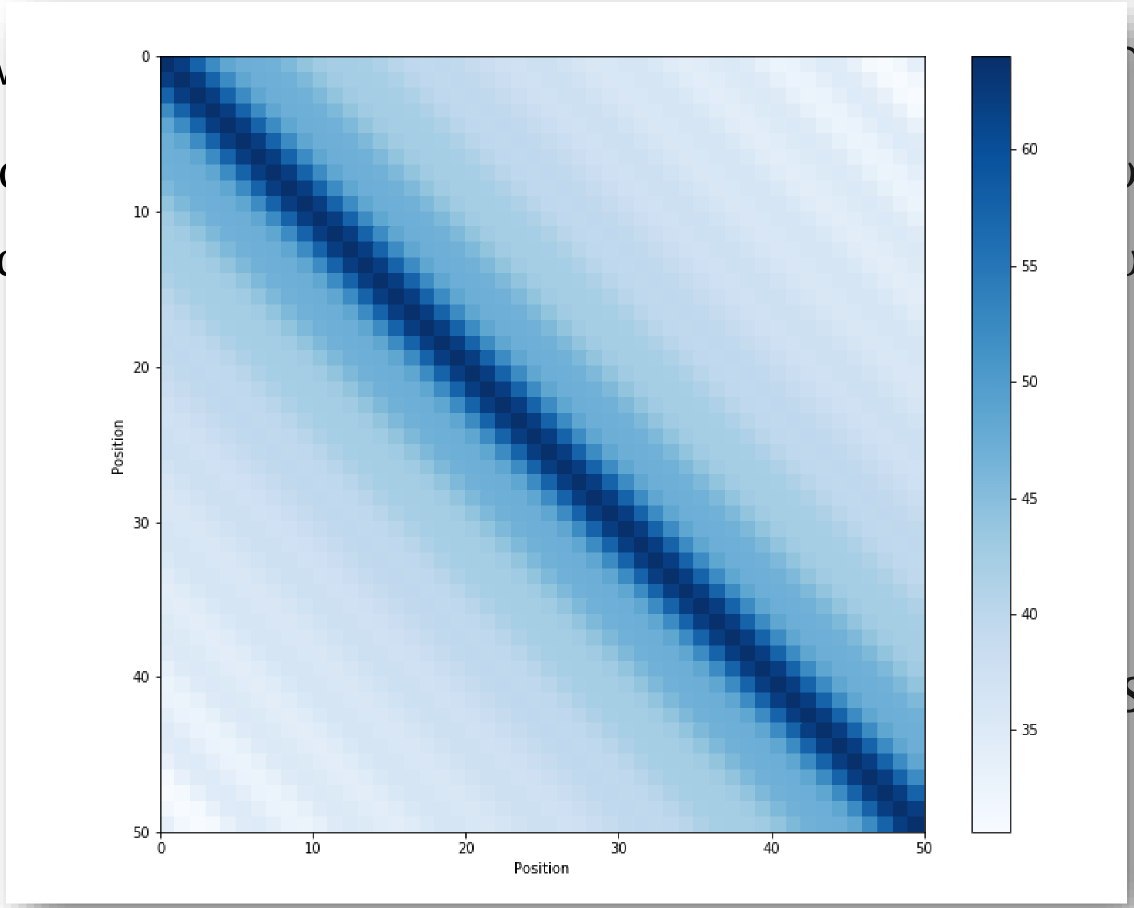
Position Encoding and Relative Positioning

From the previous v… …ns

$u_1 \sin(\omega_k \cdot pos) + v_1$ … $_k \cdot pos)\, sin(\omega_k \cdot k)$

$u_2 \sin(\omega_k \cdot pos) + v_2$ … $_k \cdot pos)\, sin(\omega_k \cdot k)$

by solving these eq…



The transformation … s (it is a rotation)

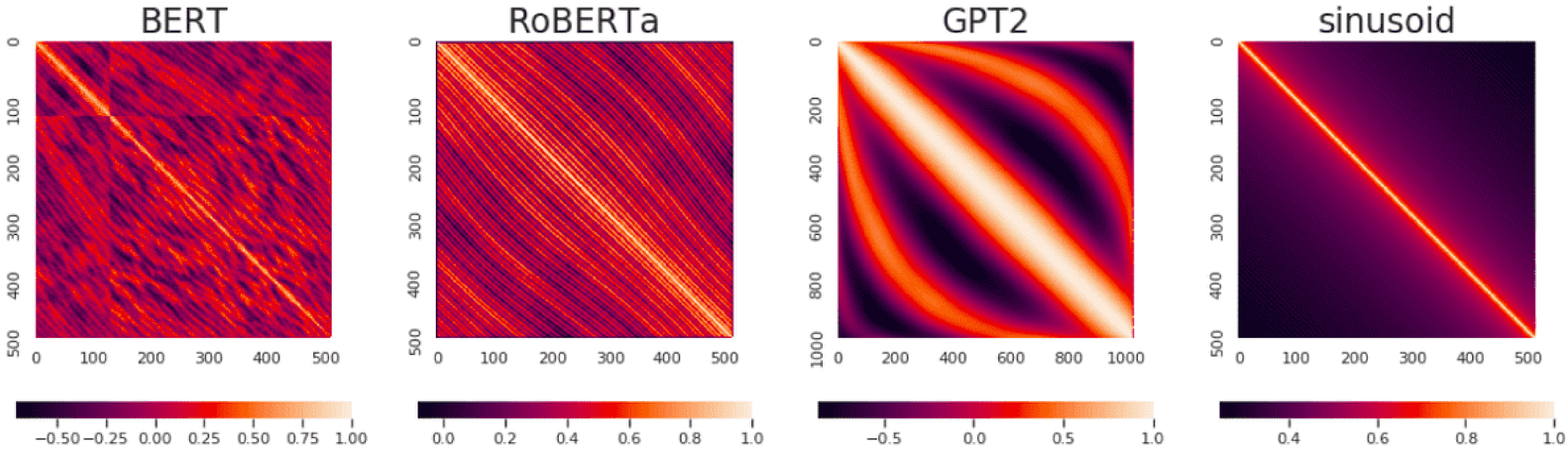
Neighboring time-steps distance are symmetrical and decay smoothly!

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! ;-)

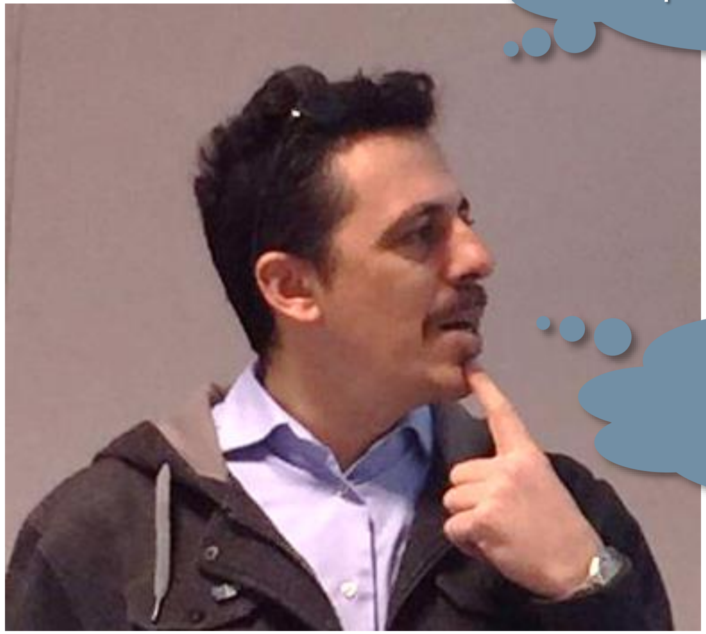
Nevertheless, state of the art Transformers (BERT, RoBERTa, GPT-2, …) learn the positional encoding instead of using a fixed one



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

Summation vs Concatenation



Why should I sum the position encoding?

What else would you do?

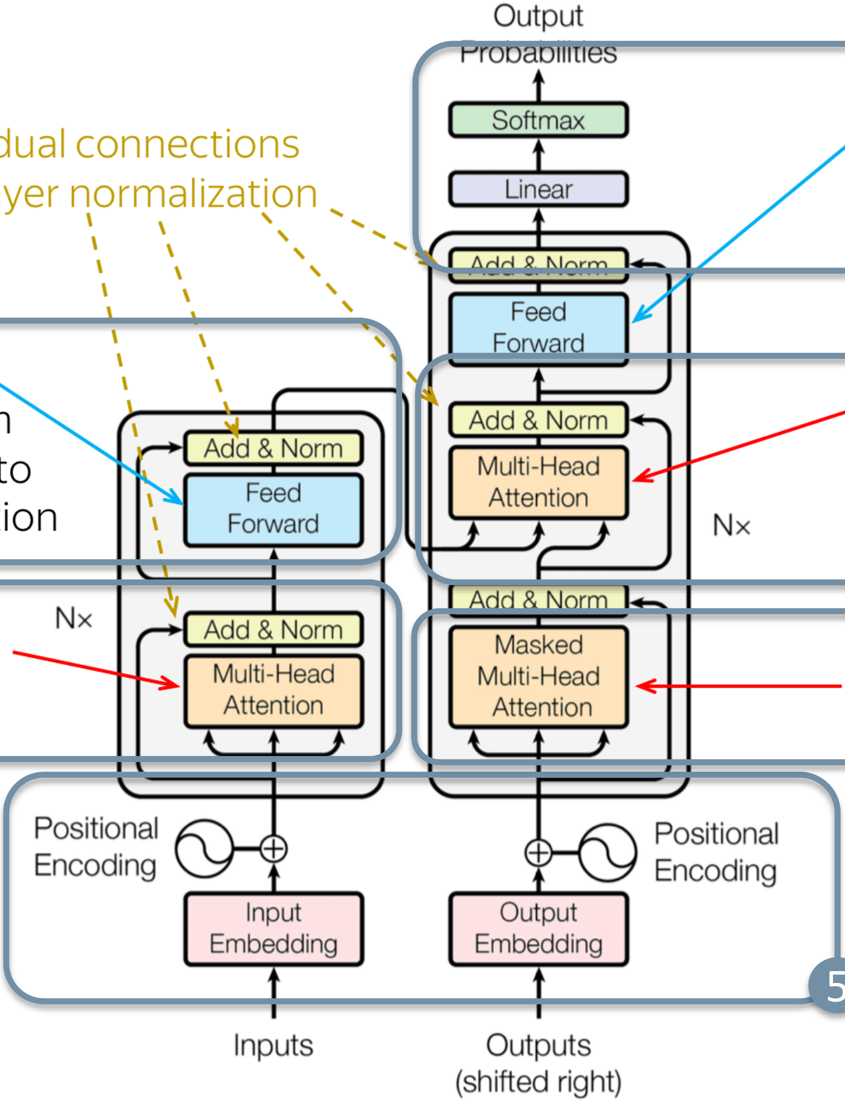
Concaternate?

Such a waste of paramenters!

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

Attention is all you need!

Residual connections and layer normalization



1 **Encoder self-attention:** tokens look at each other

queries, keys, values are computed from encoder states

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

queries, keys, values are computed from decoder states

3 **Decoder-encoder attention:** target token looks at the source

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

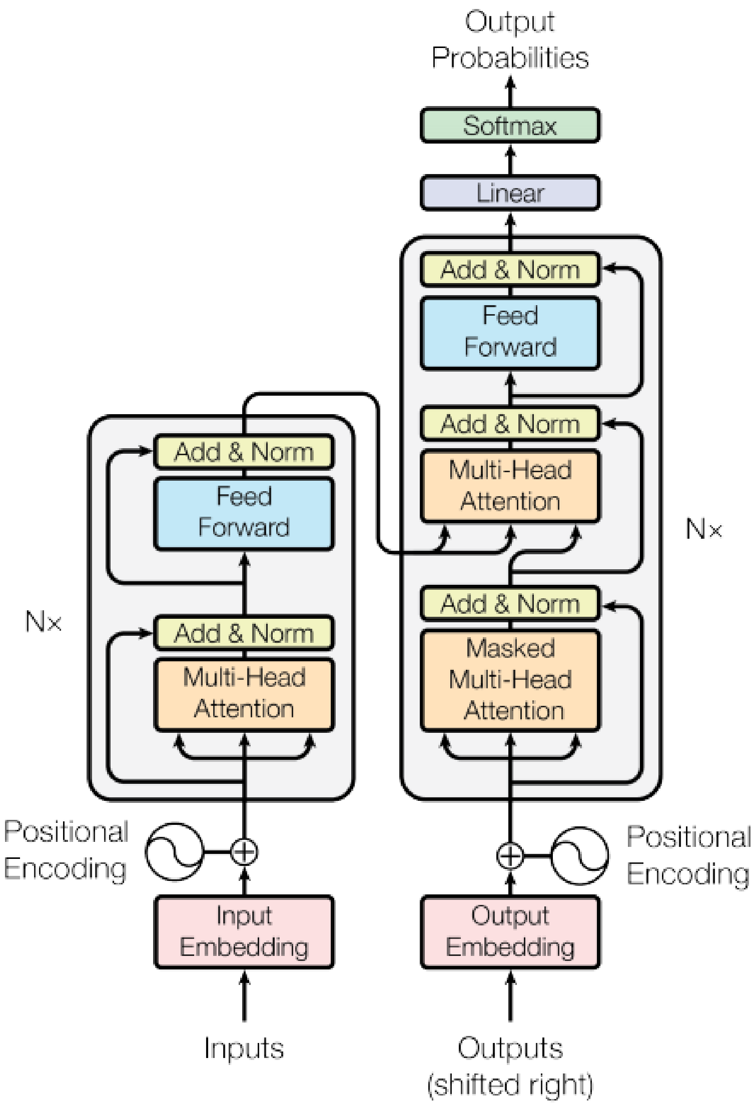
4 **Feed-forward network:** after taking information from other tokens, take a moment to think and process this information

5

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

Attention Is All You Need https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Transformer Complexity



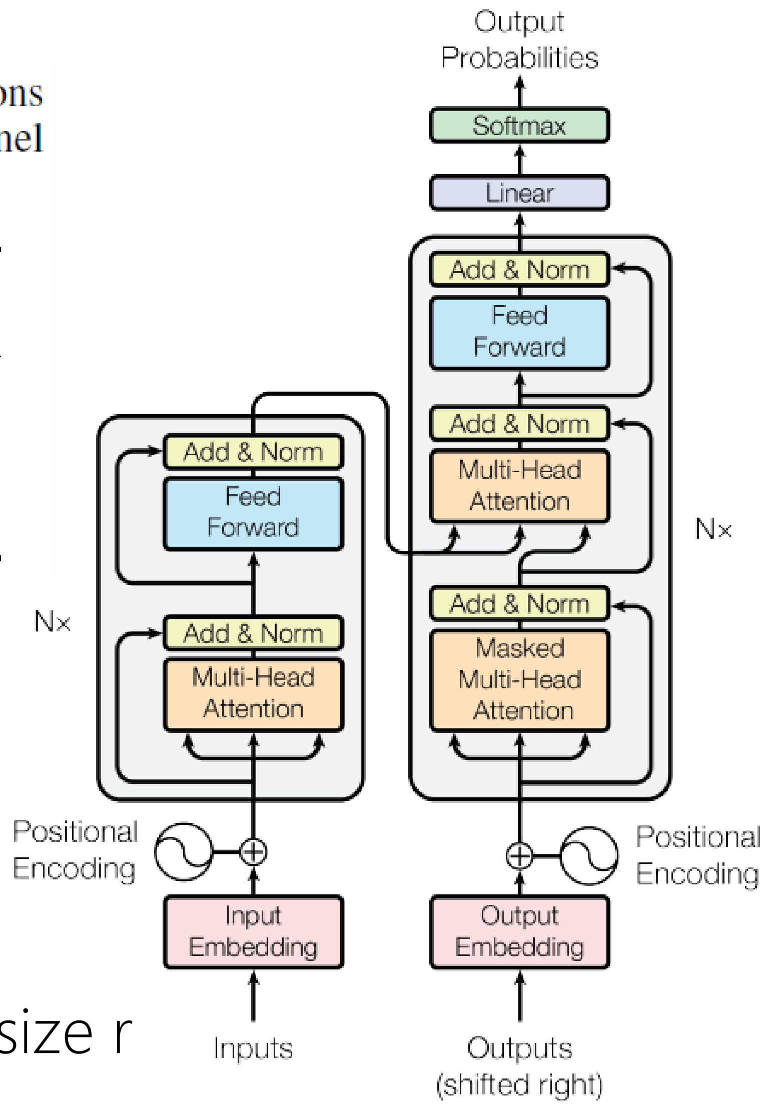
Attention Is All You Need https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

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

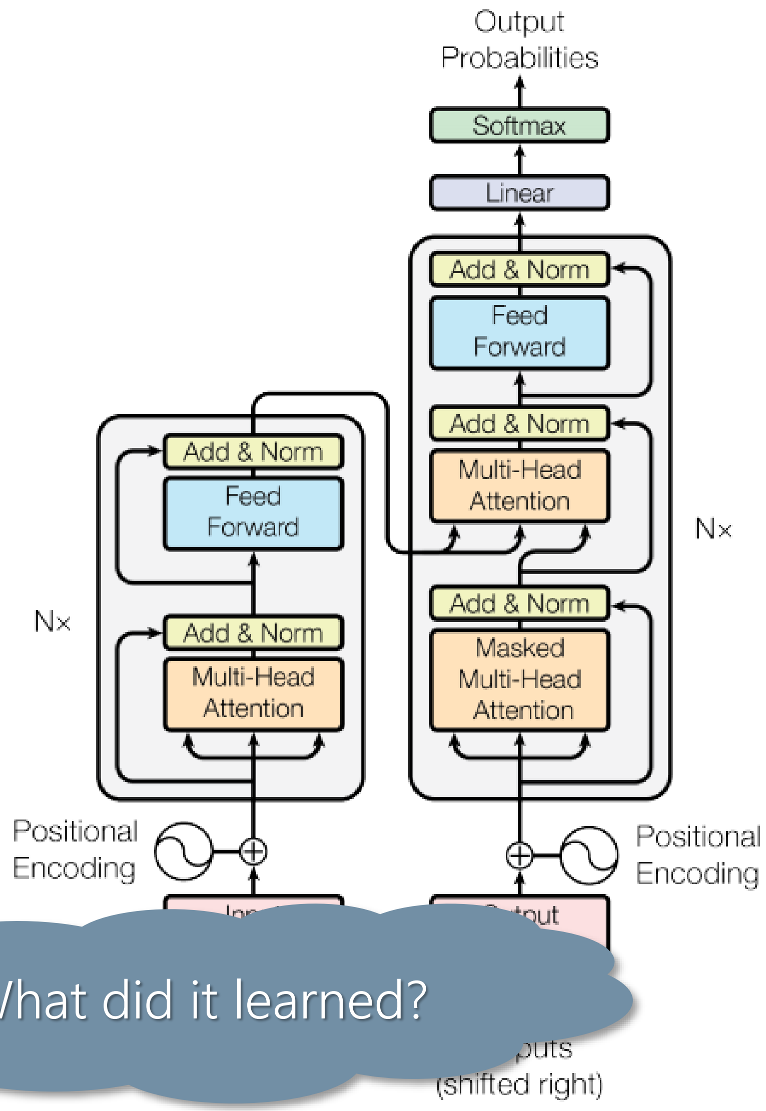


Attention Is All You Need https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Transformer Performance

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models c
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	$\mathbf{3.3 \cdot 10^{18}}$	
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

What did it learned?

Attention Is All You Need https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Transformer Heads are Interpretable

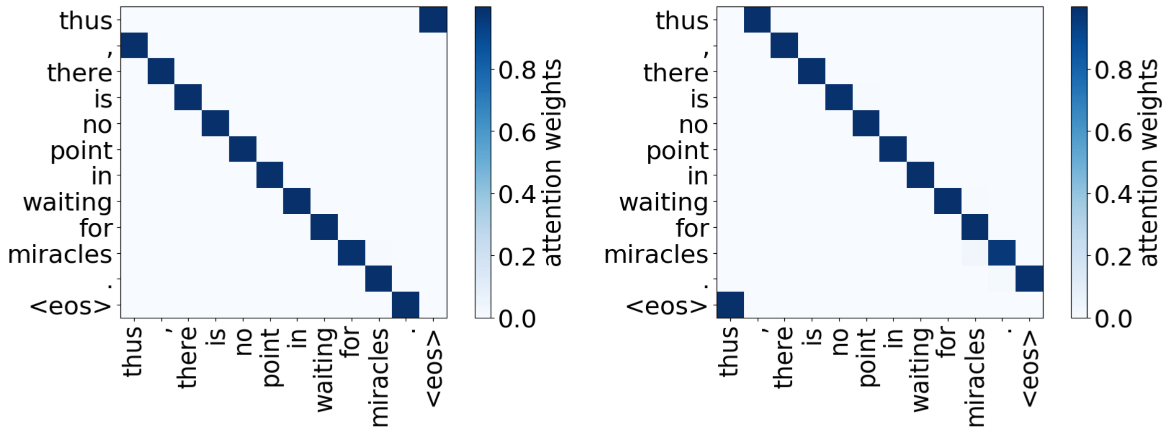
By looking at how much, on average, different heads "contribute" to generated translations it turns out only a small number are important and the play interpretable "roles“:

- 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!)

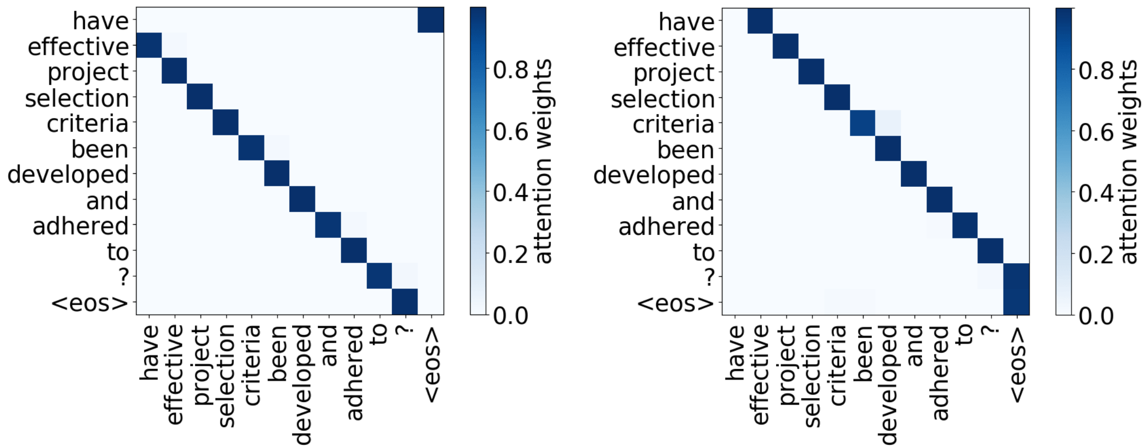
Remaining ones can be pruned

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

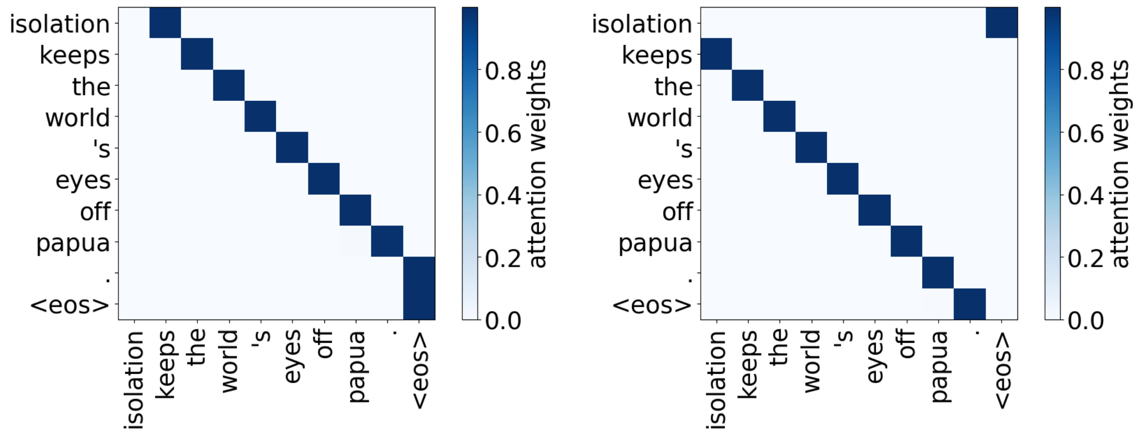
Positional Heads



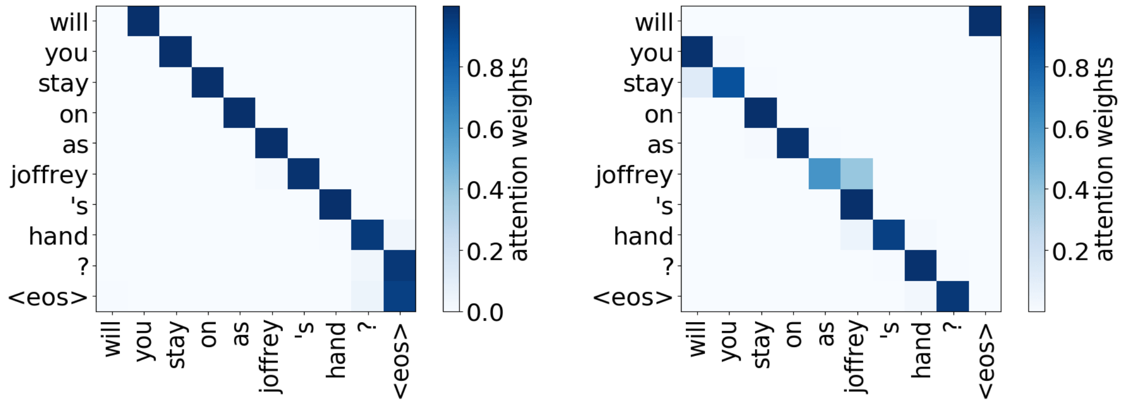
Model Trained on WMT EN-DE



Model Trained on WMT EN-FR



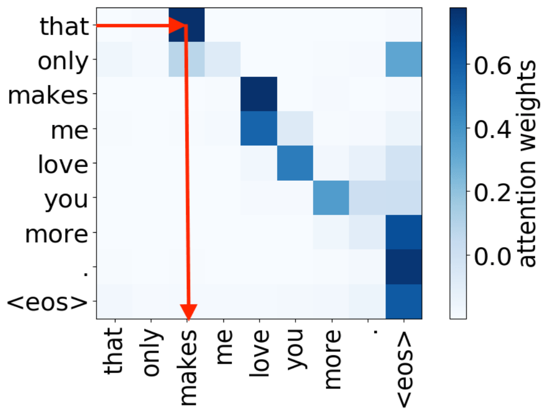
Model Trained on WMT EN-RU



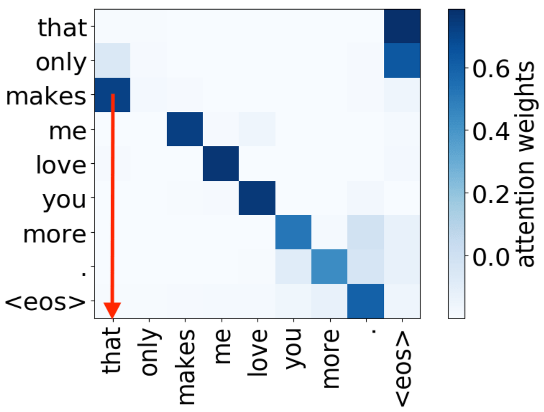
Model Trained on OpenSubtitles EN-RU

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

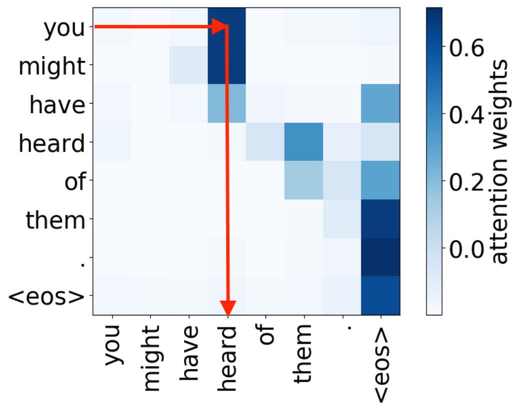
Syntactic Heads



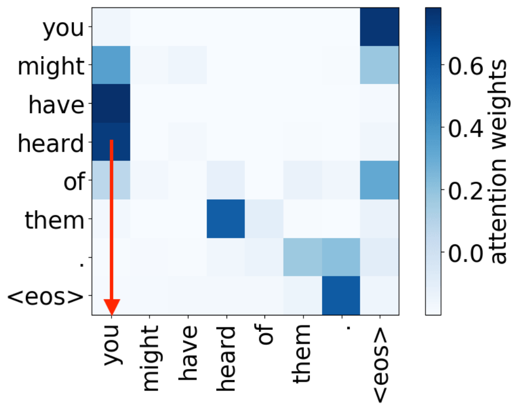
Subject->Verb



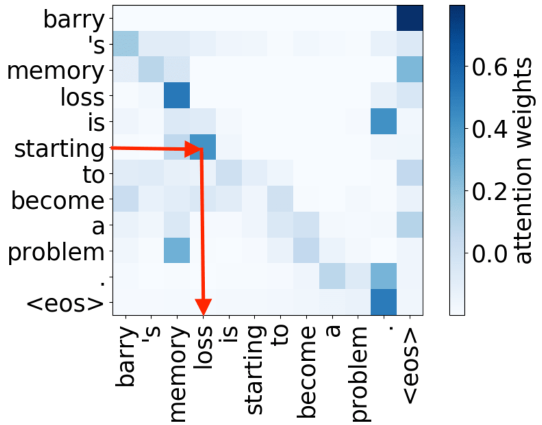
Verb->Subject



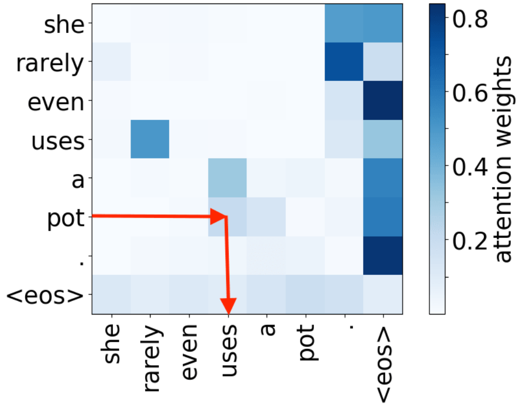
Subject->Verb



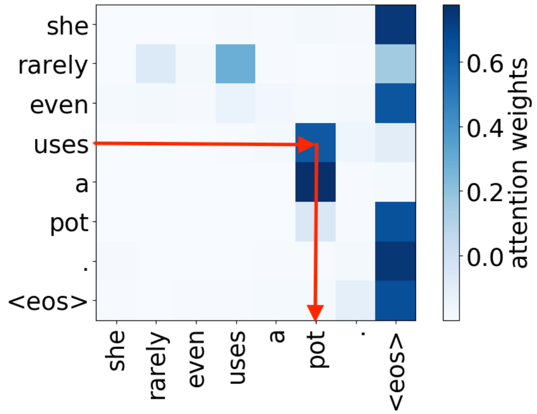
Verb->Subject



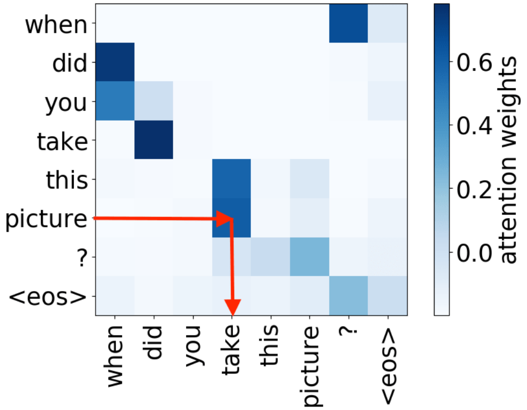
Verb -> Subject



Object -> Verb



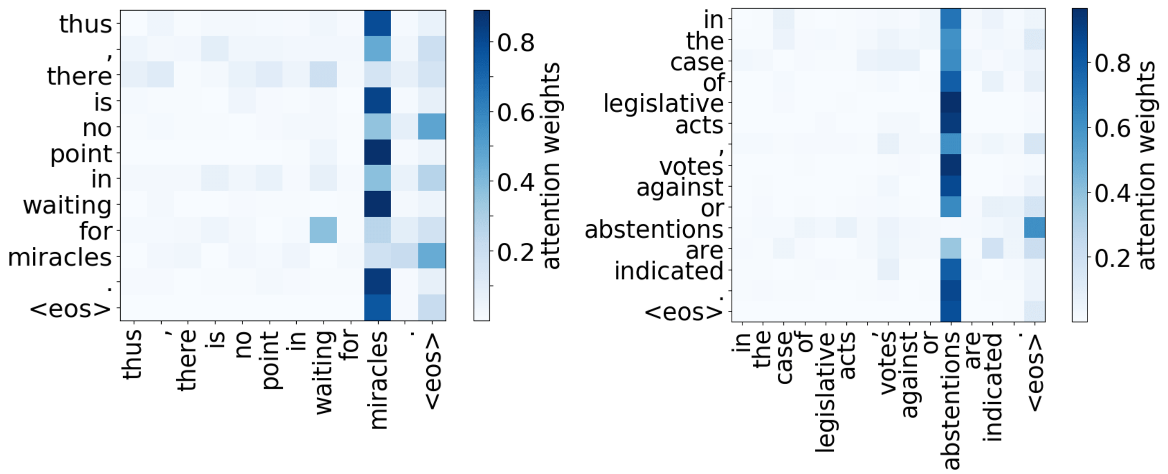
Verb -> Object



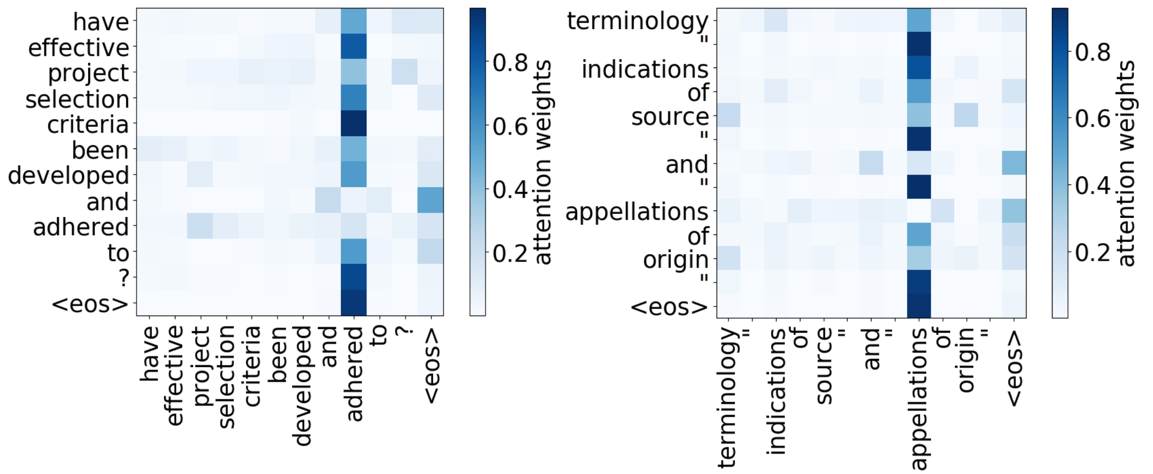
Object -> Verb

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

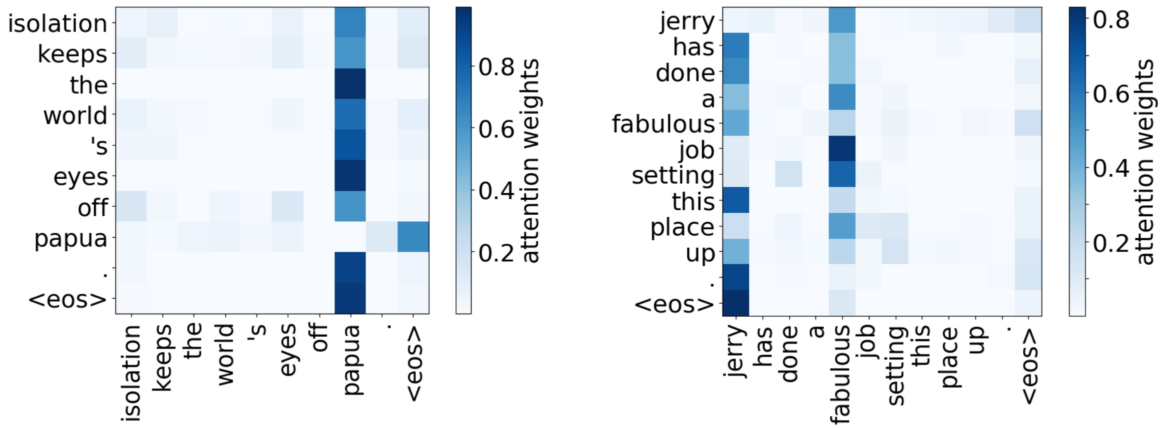
Rare Tokens Heads



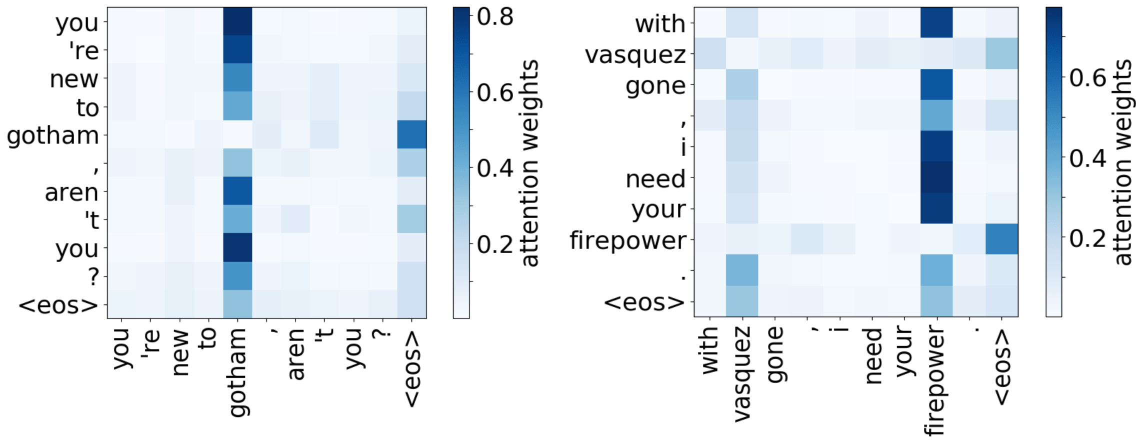
Model Trained on WMT EN-DE



Model Trained on WMT EN-FR



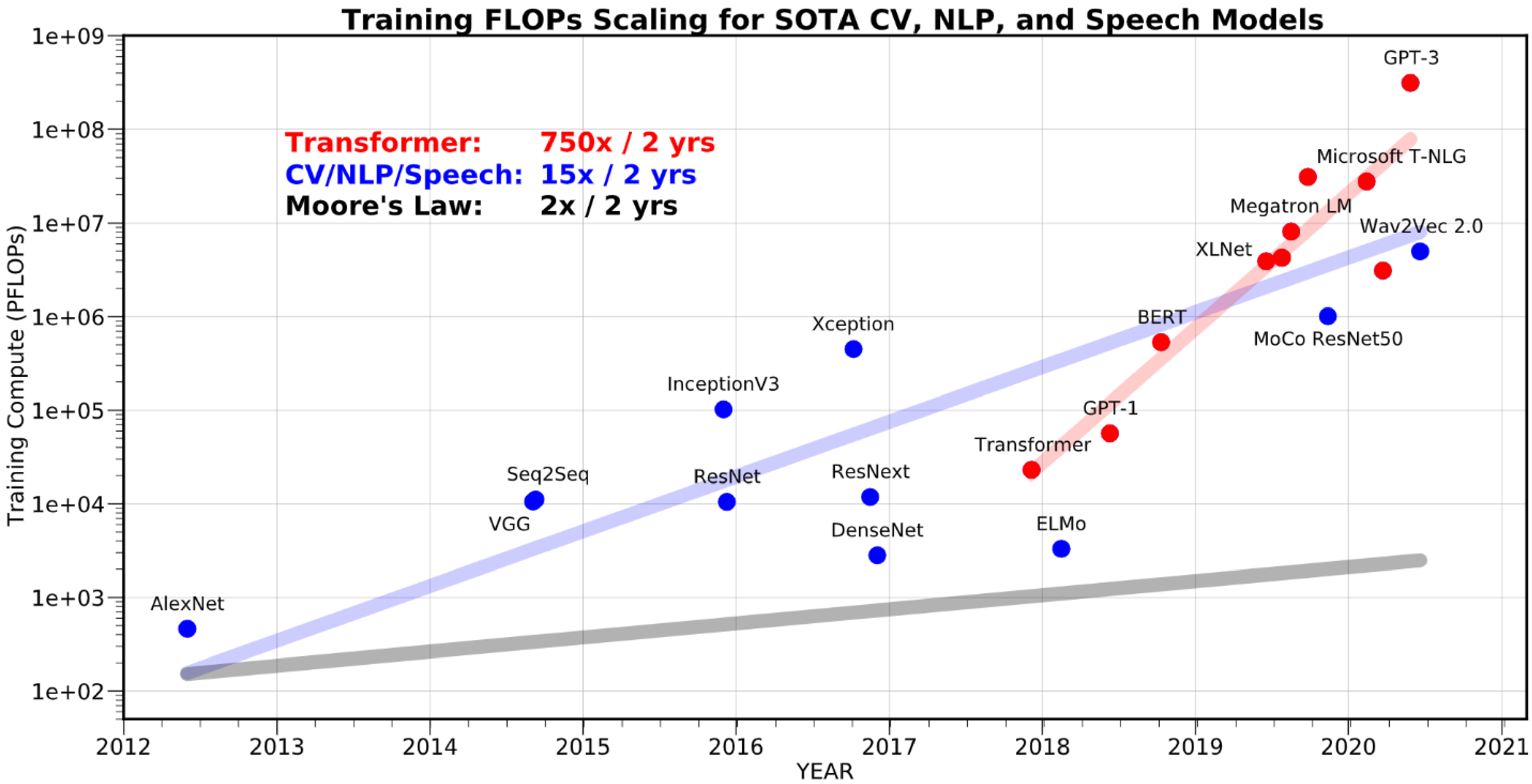
Model Trained on WMT EN-RU



Model Trained on OpenSubtitles EN-RU

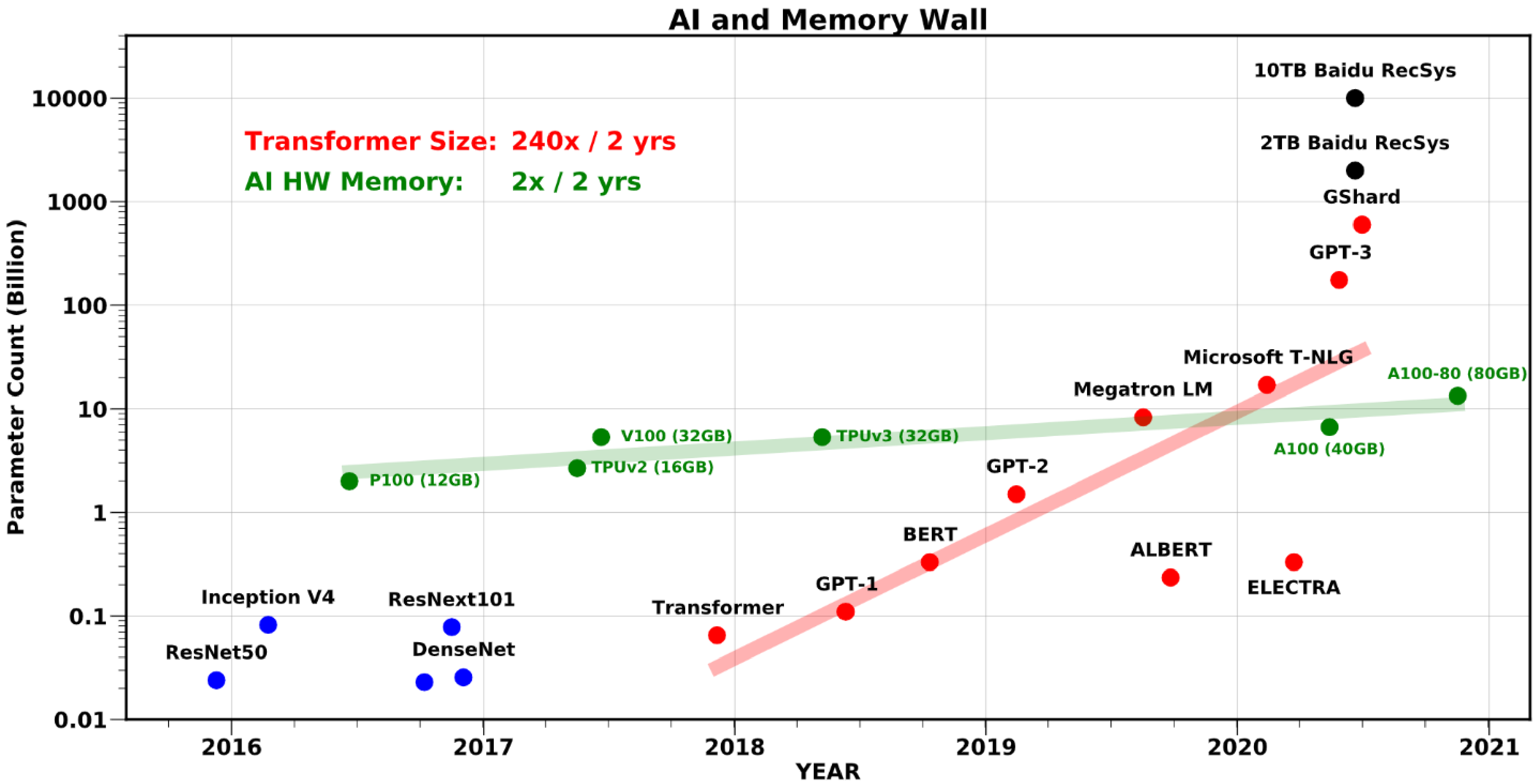
Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned https://lena-voita.github.io/posts/acl19_heads.html

Are there any limits for Transformers?



AI and Memory Wall [*https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8*](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*](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*](https://www.tensorflow.org/text/tutorials/transformer?hl=en)
- *The Annotated Transformer* [*http://nlp.seas.harvard.edu/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