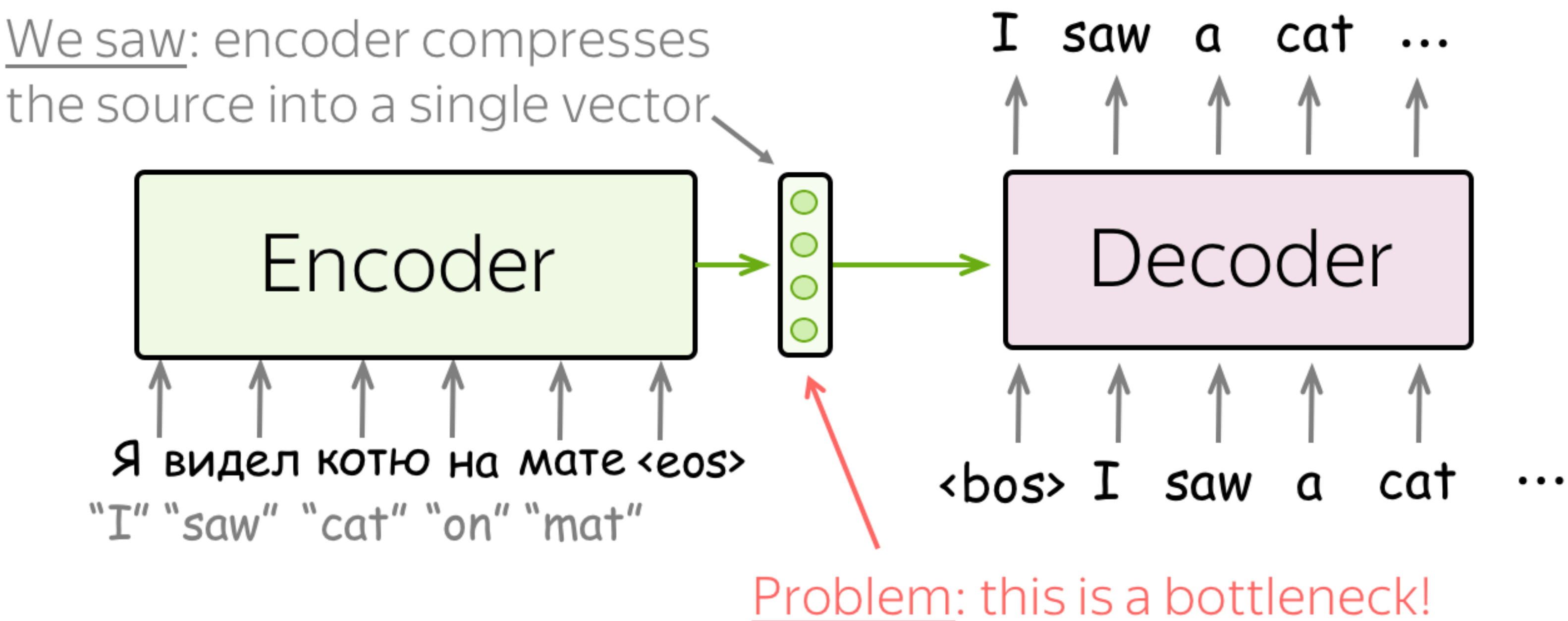
Foundations for Natural Language Processing

Transformer

Ivan Titov
(with graphics/materials from Elena Voita)

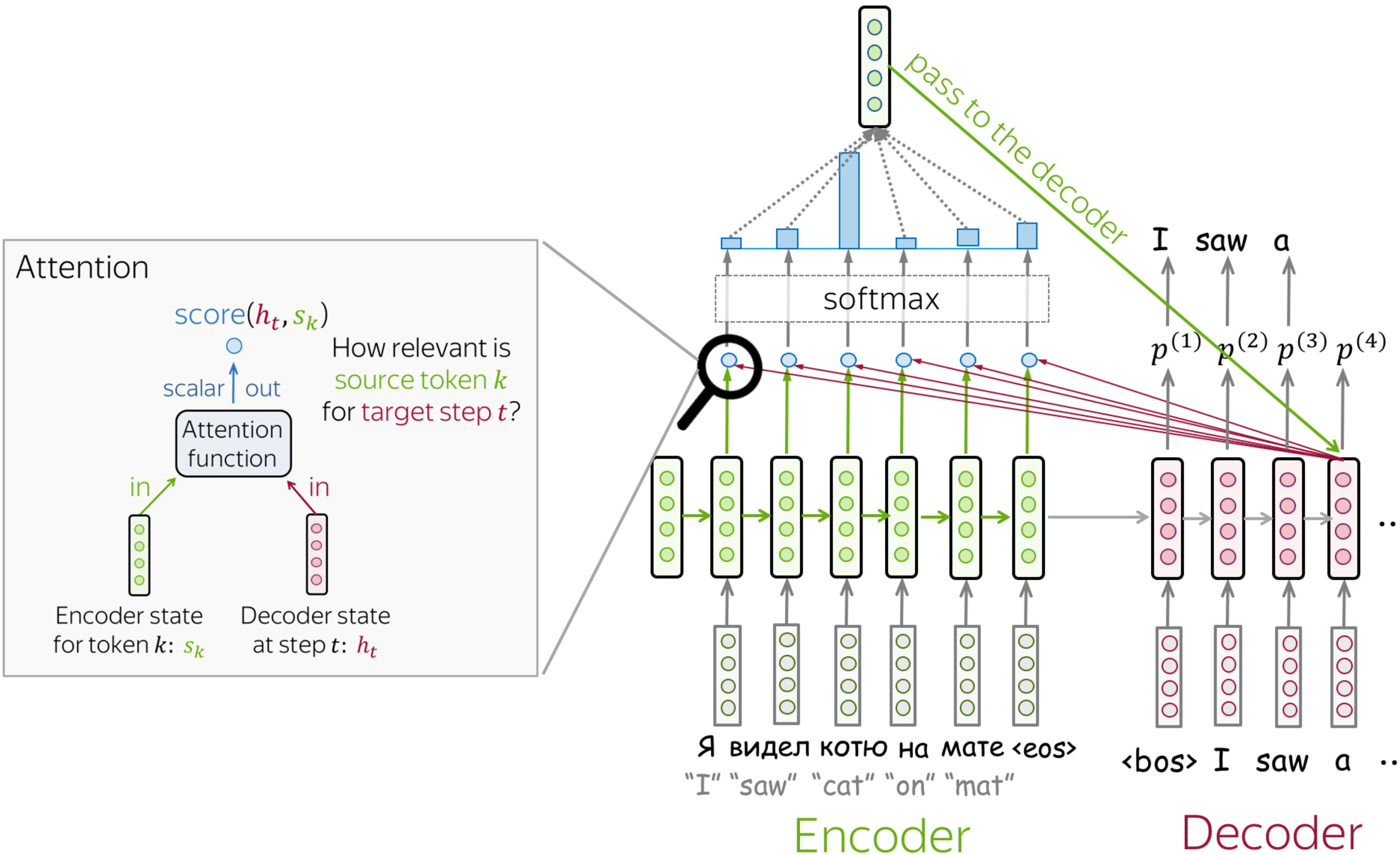
School of informatics

Recap: Vanilla encoder-decoders

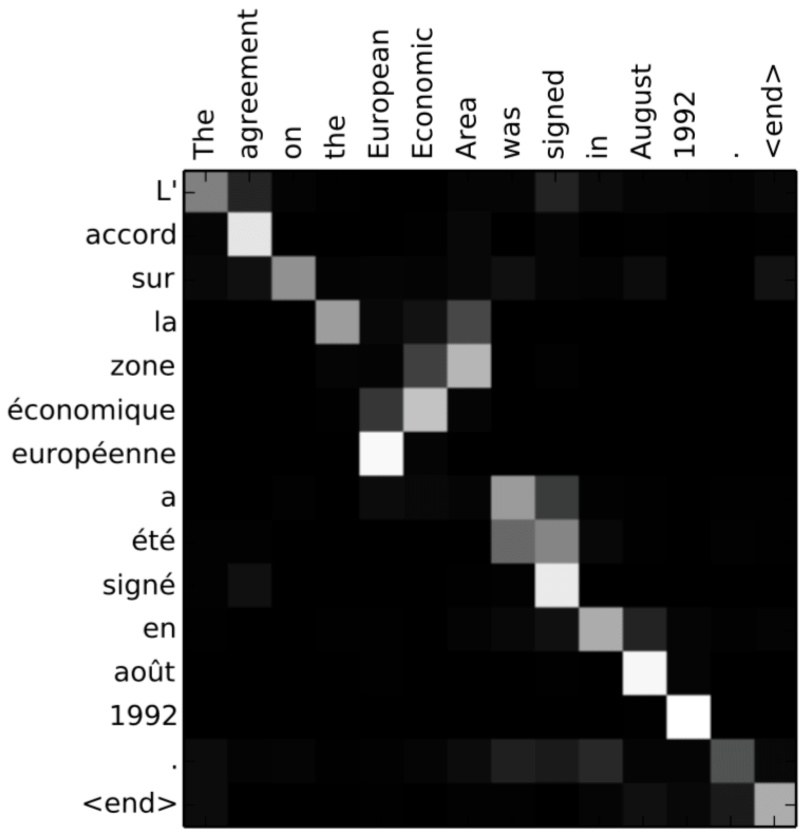


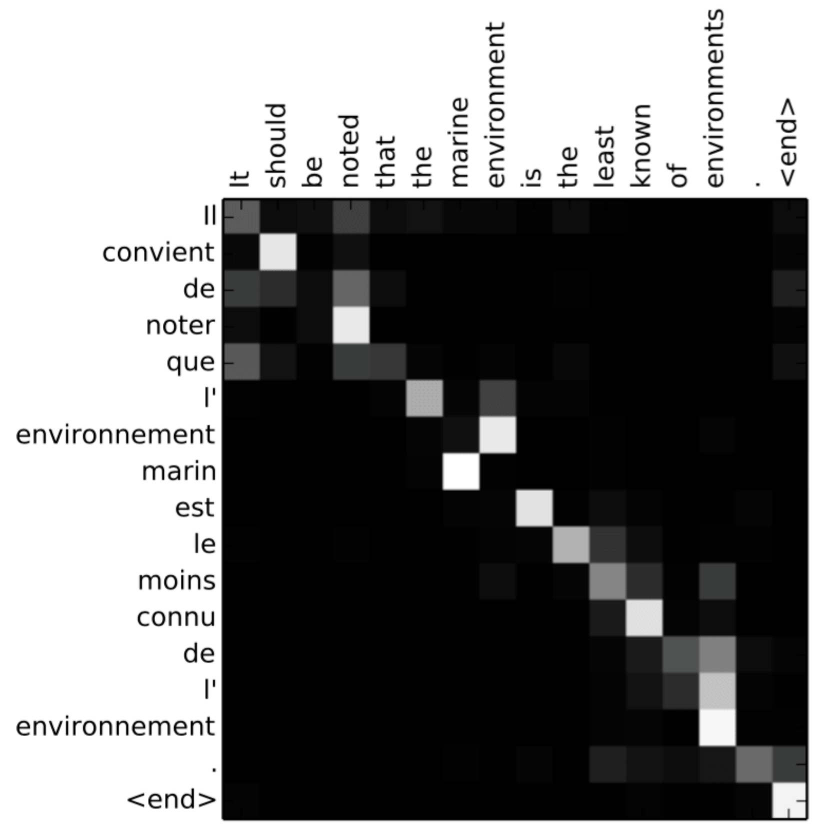
Recap: Attention

Model learns to 'pay attention' to most relevant source tokens



Recap: attention is (a bit like) alignment





Transformer

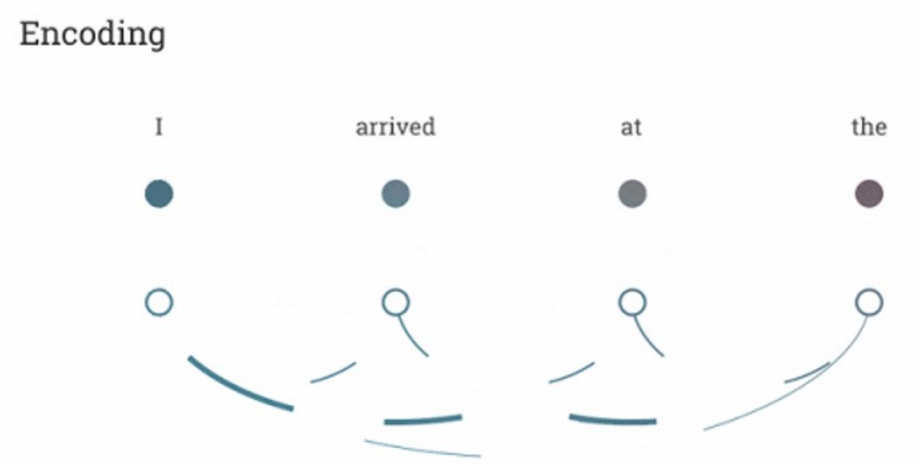
"Attention is all you need"

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

Transformer: Intuition

See animation here:
https://blog.research.google/2017/08/transformer-novel-neural-network.html

Transformer: Intuition



See animation here:
https://blog.research.google/2017/08/transformer-novel-neural-network.html

Transformer: Intuition

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

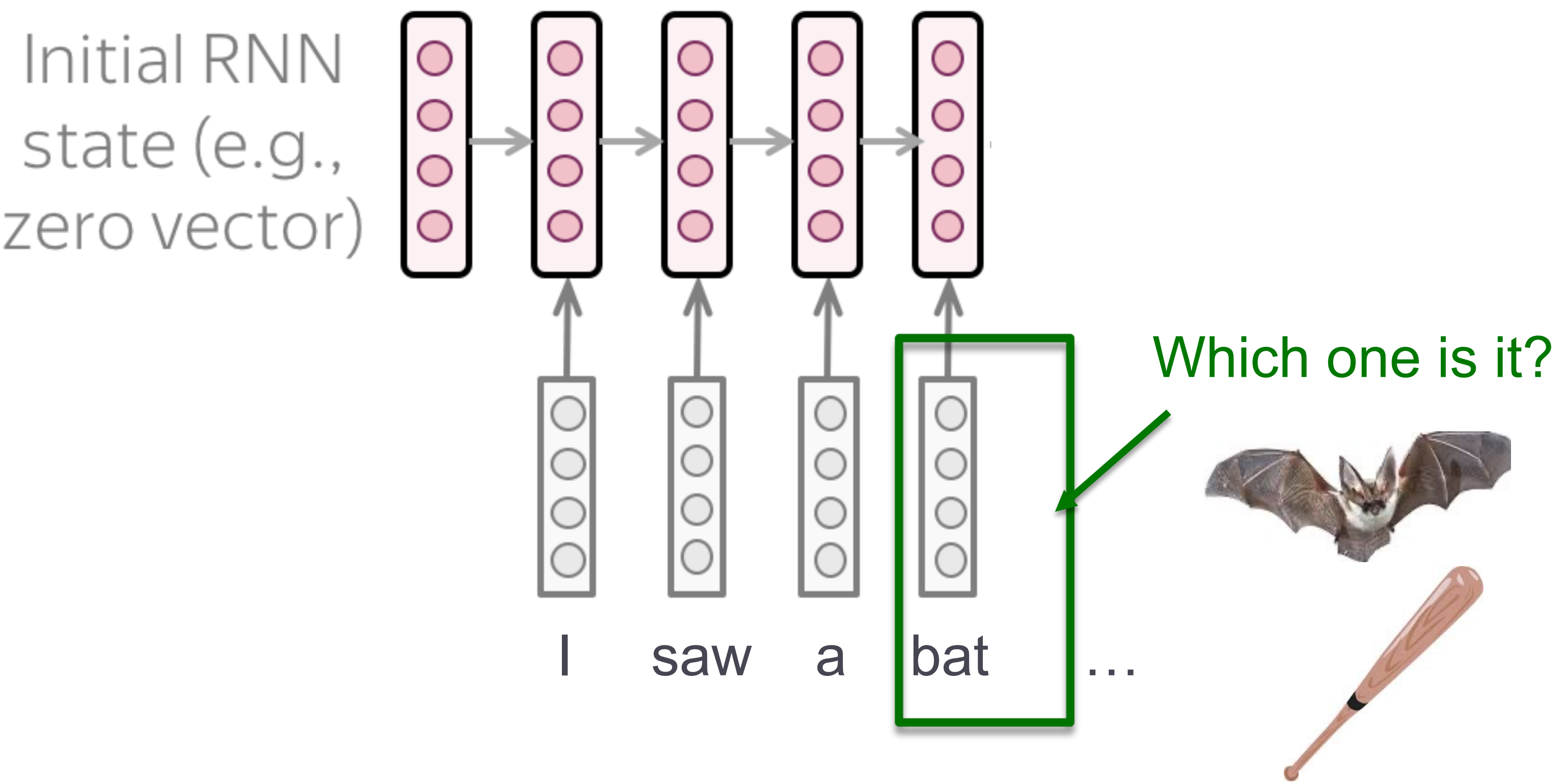
What they are doing:

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

repeat N times

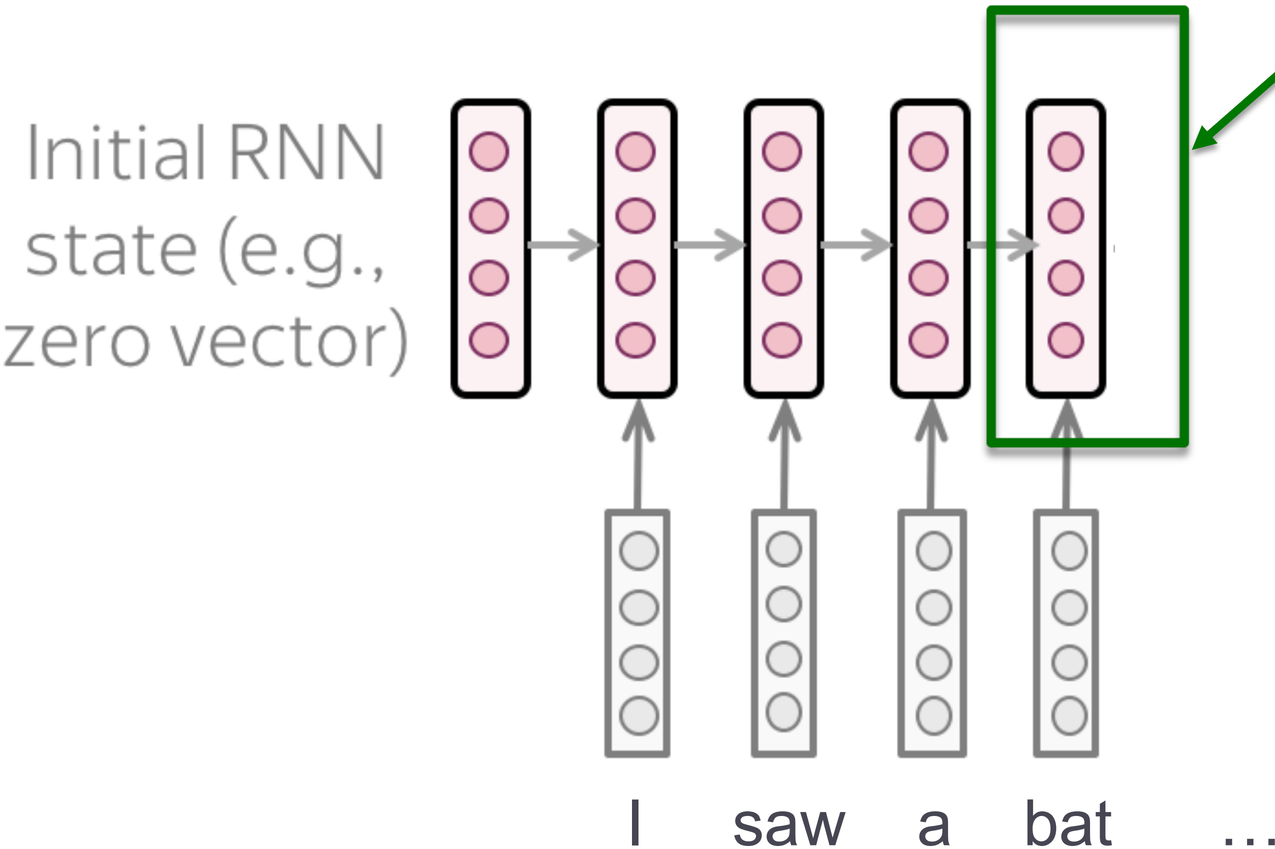
Why do we need self-attention in the encoder?

Let's recall RNNs



Why do we need self-attention in the encoder?

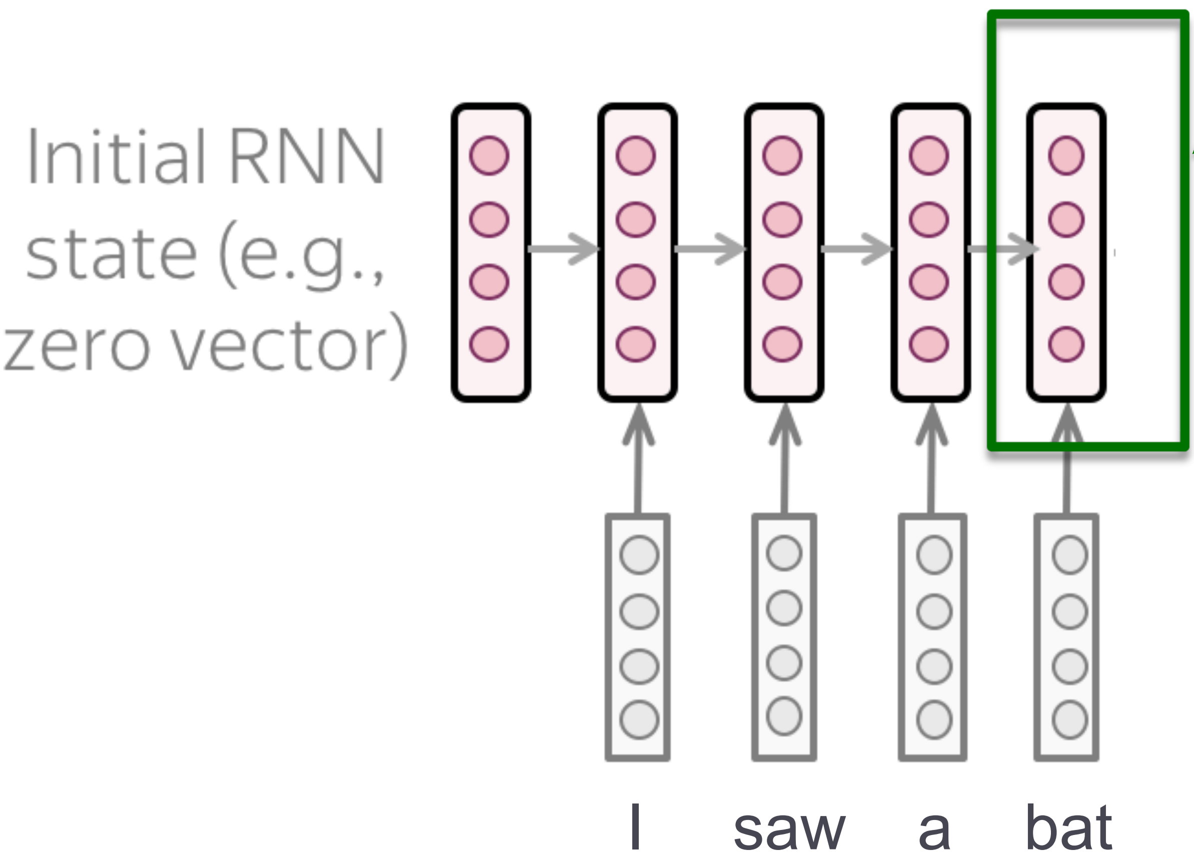
Let’s recall RNNs



This states Incorporates the prefix, but does it really help

Why do we need self-attention in the encoder?

Let’s recall RNNs



This states Incorporates the prefix, but does it really help

In RNNs we used bidirectional encoders to incorporate the information about the “future”

Why do we need self-attention in the encoder?

The self-attention is used in the Transformer to incorporate information about the context (including future context)

The process repeated multiple times, once per layer, iteratively refining the token representations

Learned end-to-end with an encoder-decoder model, so the model will learn to produce token representations useful for the decoder*

*this is going to be task-specific, e.g., representations useful for translation will be different from those for sentiment analysis or summarization

Encoder-decoder attention vs self- attention

Decoder-encoder attention is looking

- **from**: one current decoder state
- **at**: all encoder states

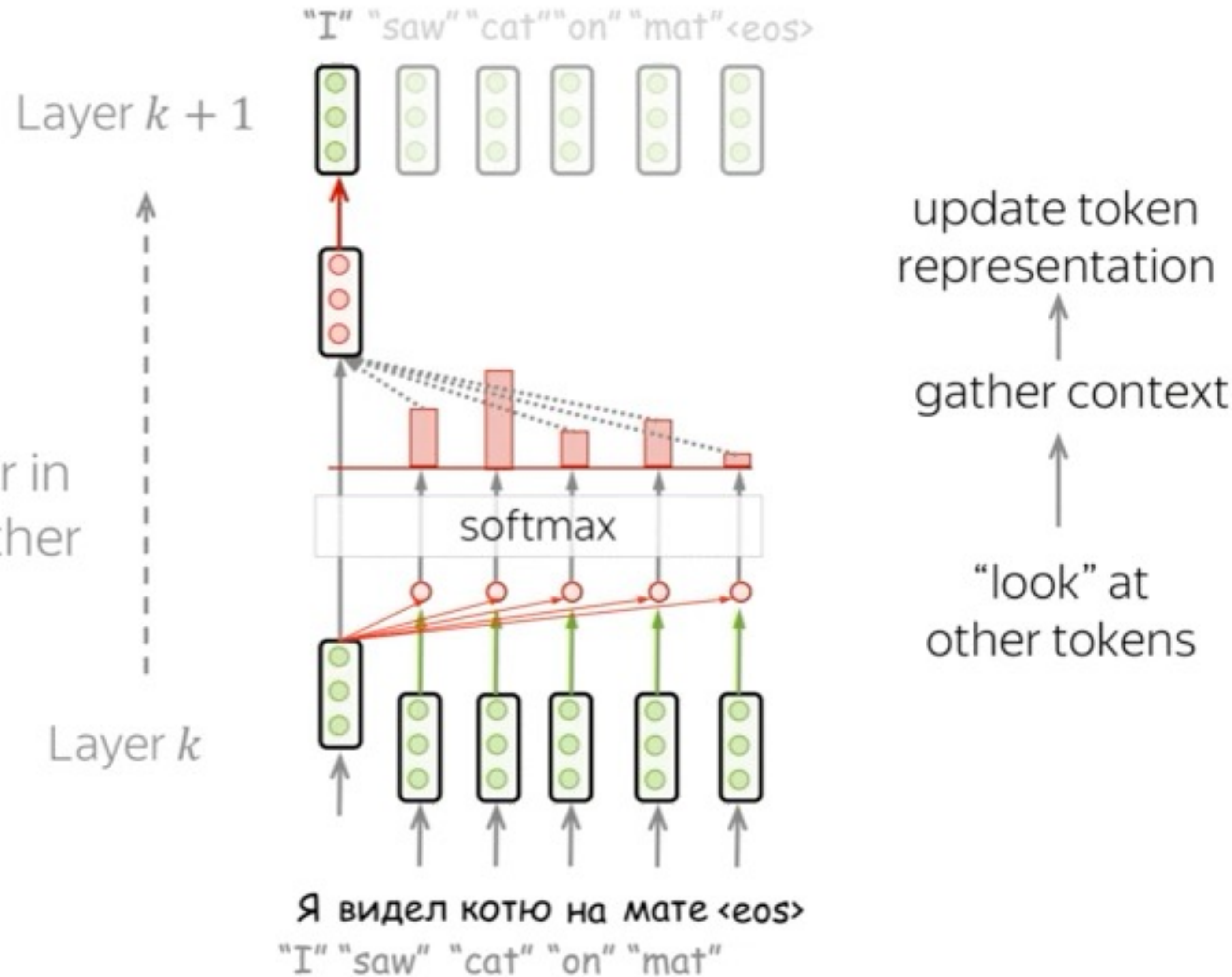
aka “cross-attention”

Self-attention is looking

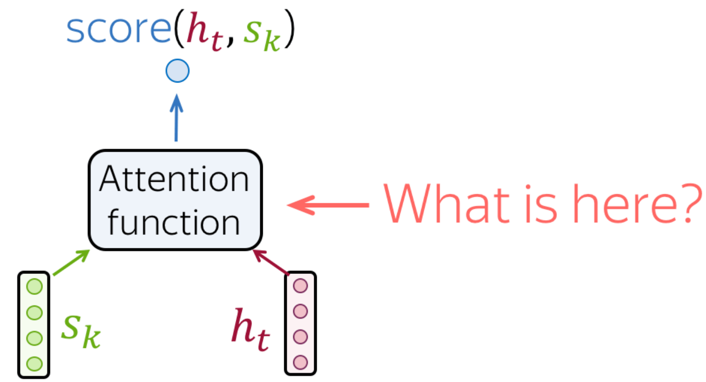
- **from**: each state from a set of states
- **at**: all other states in the same set

Self-attention

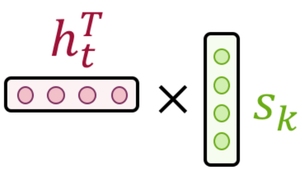
Tokens try to understand themselves better in context of each other



Recap: attention computation in encoder-decoder

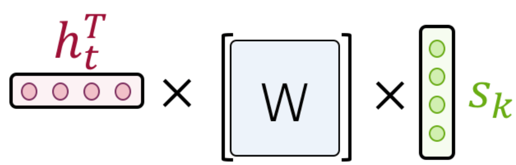


Dot-product



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

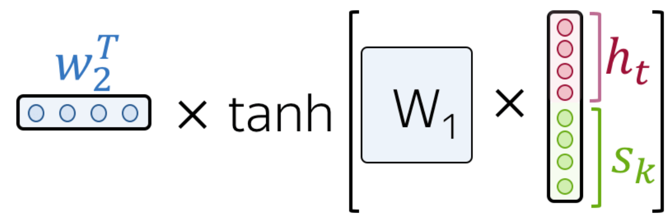
Bilinear



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

aka ‘Luong attention’

Multi-Layer Perceptron



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

aka ‘Bahdanau’ attention
(from the original paper)

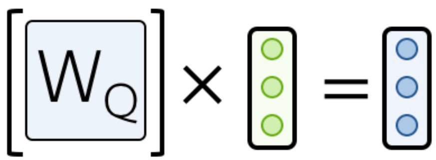
Query-Key-Value attention

In self-attention, each token plays 3 different roles and has 3 different representations (1 per role)

1. query- asking for information;
2. key - saying that it has some information;
3. value - giving the information.

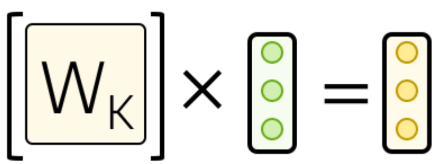
Query-Key-Value attention

They all 3 are a result of a linear transformation of the original representation



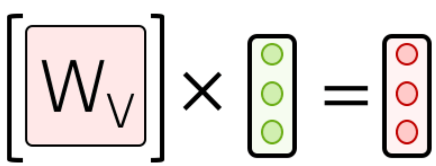
Query: vector **from** which the attention is looking

"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!"

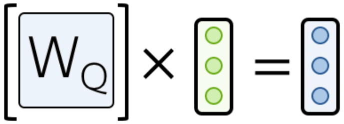


Value: their weighted sum is attention output

"Here's the information I have!"

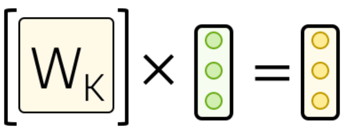
Query-Key-Value attention

Each vector receives three representations ("roles")



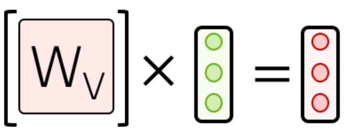
Query: vector **from** which the attention is looking

"Hey there, do you have this information?"



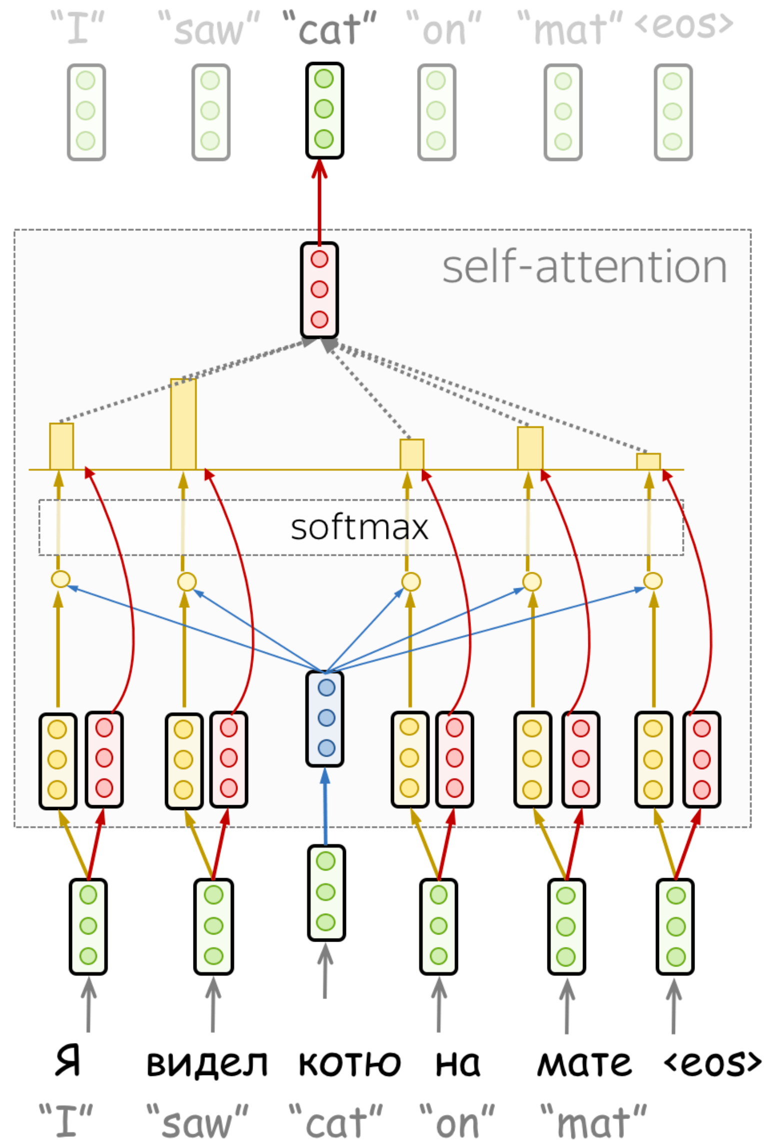
Key: vector **at** which the query looks to compute weights

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



Value: their weighted sum is attention output

"Here's the information I have!"

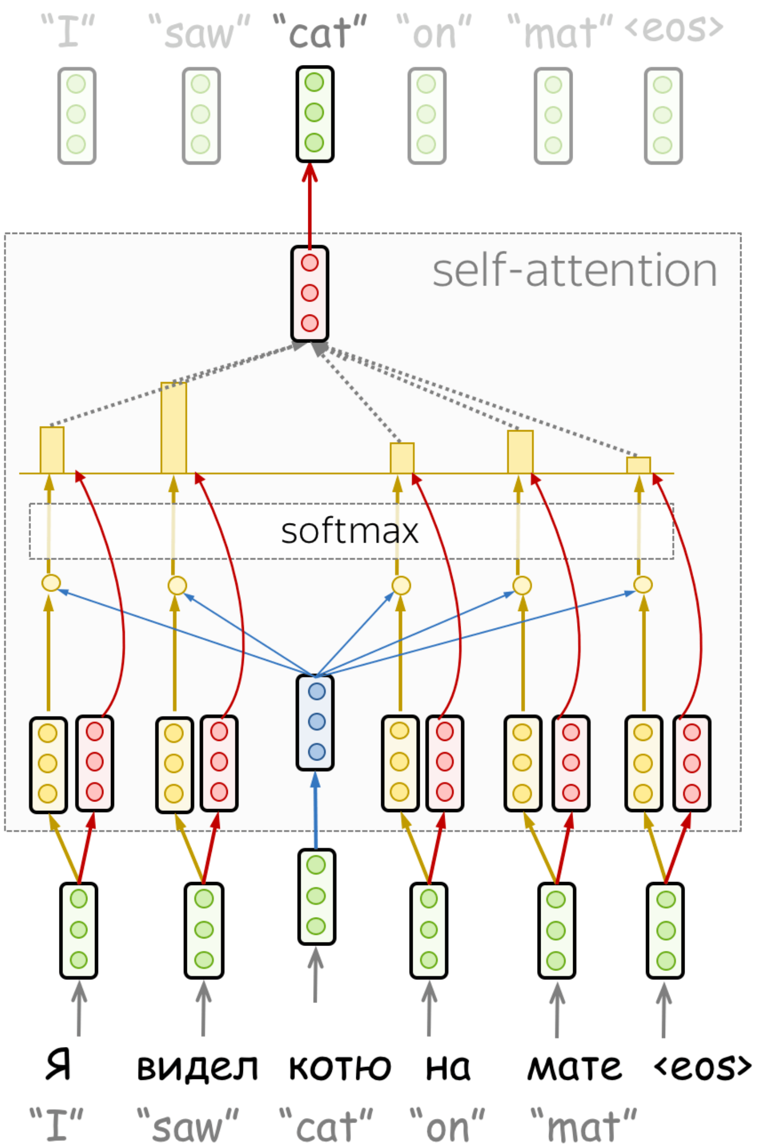


Query-Key-Value attention

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

from to

vector dimensionality of K, V



Transformer

"Attention is all you need"

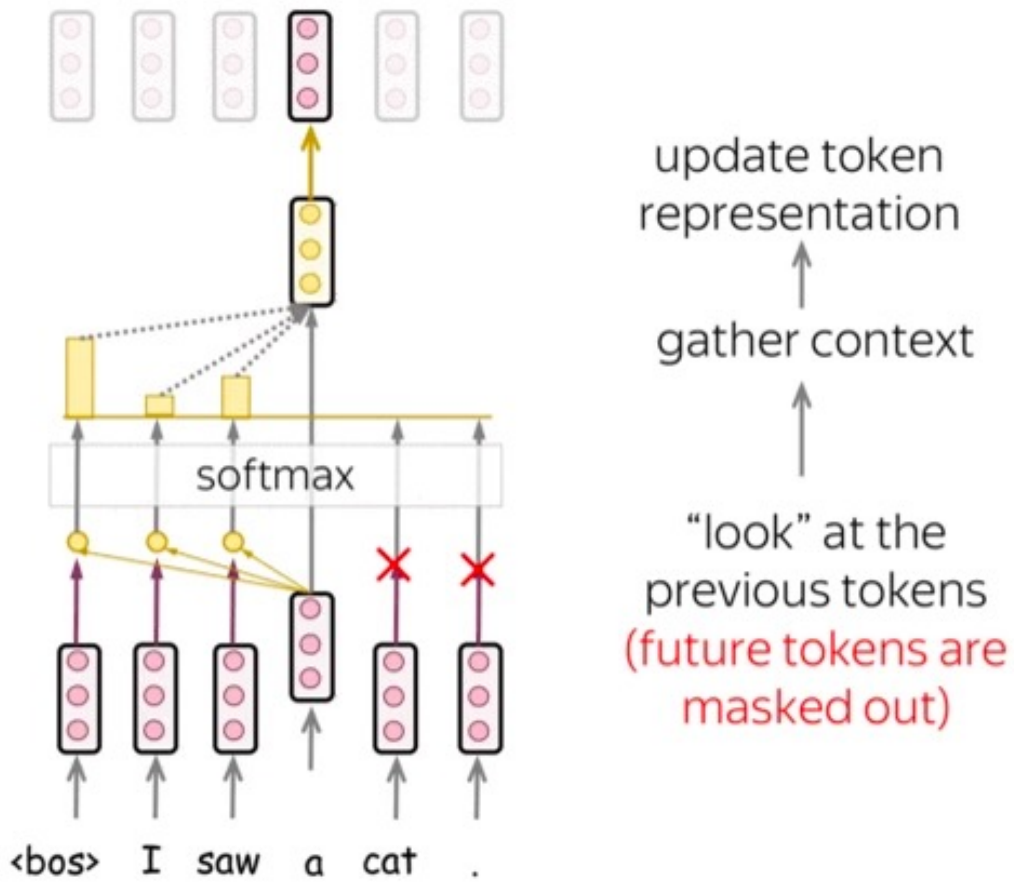
	Transformer
processing within encoder	attention
processing within decoder	$\boxed{\text{attention}}$
decoder-encoder interaction	attention

Masked Attention

At inference time, the decoder does not have access to the future (because it has not generated it yet)

The future is known in training

But training needs to be consistent with inference, so we 'mask' future tokens when training the representations in the decoder

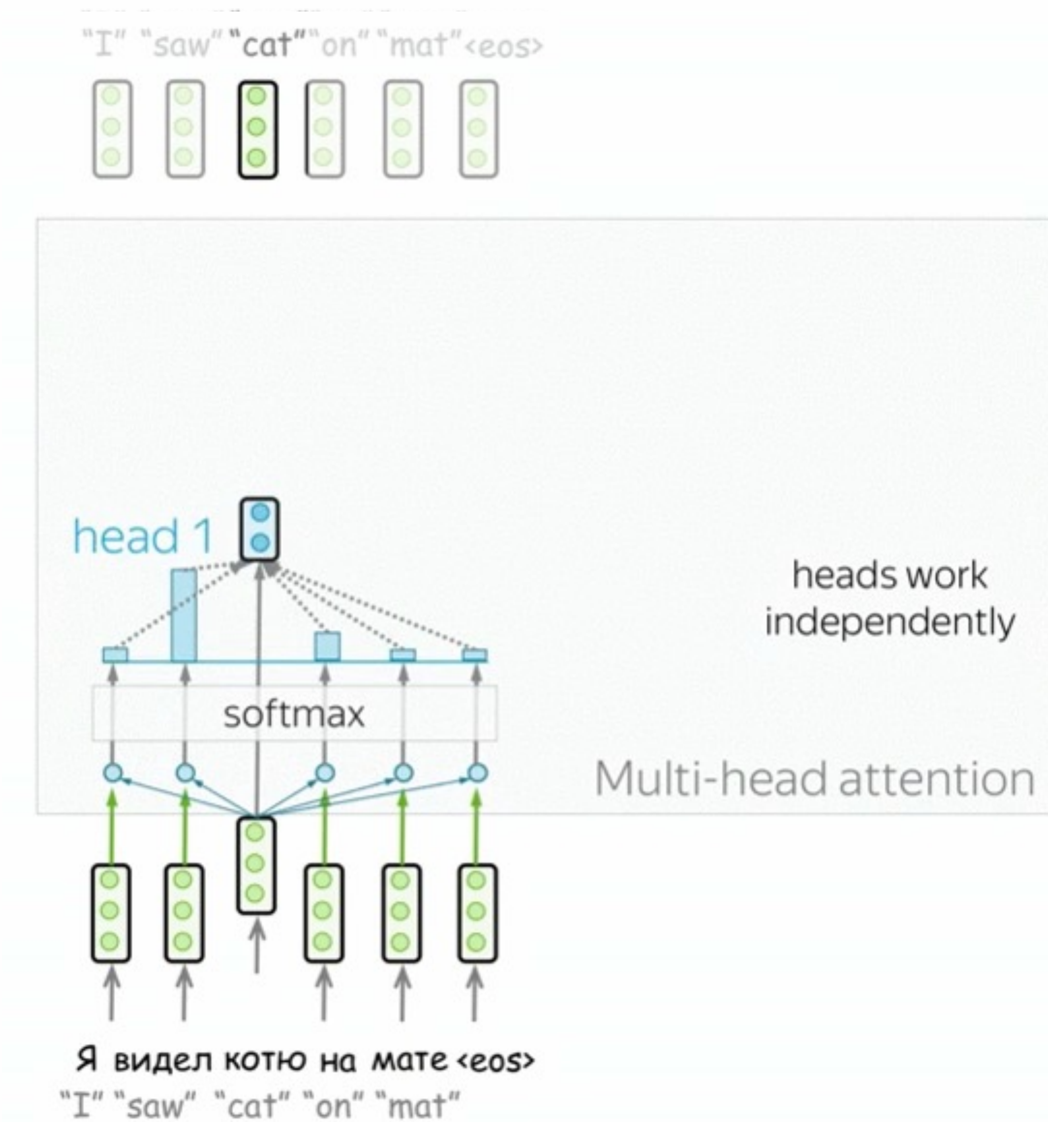


(you can think of this as a trick enabling fast training)

Multi-Head Attention

Each head specializes on a different relation

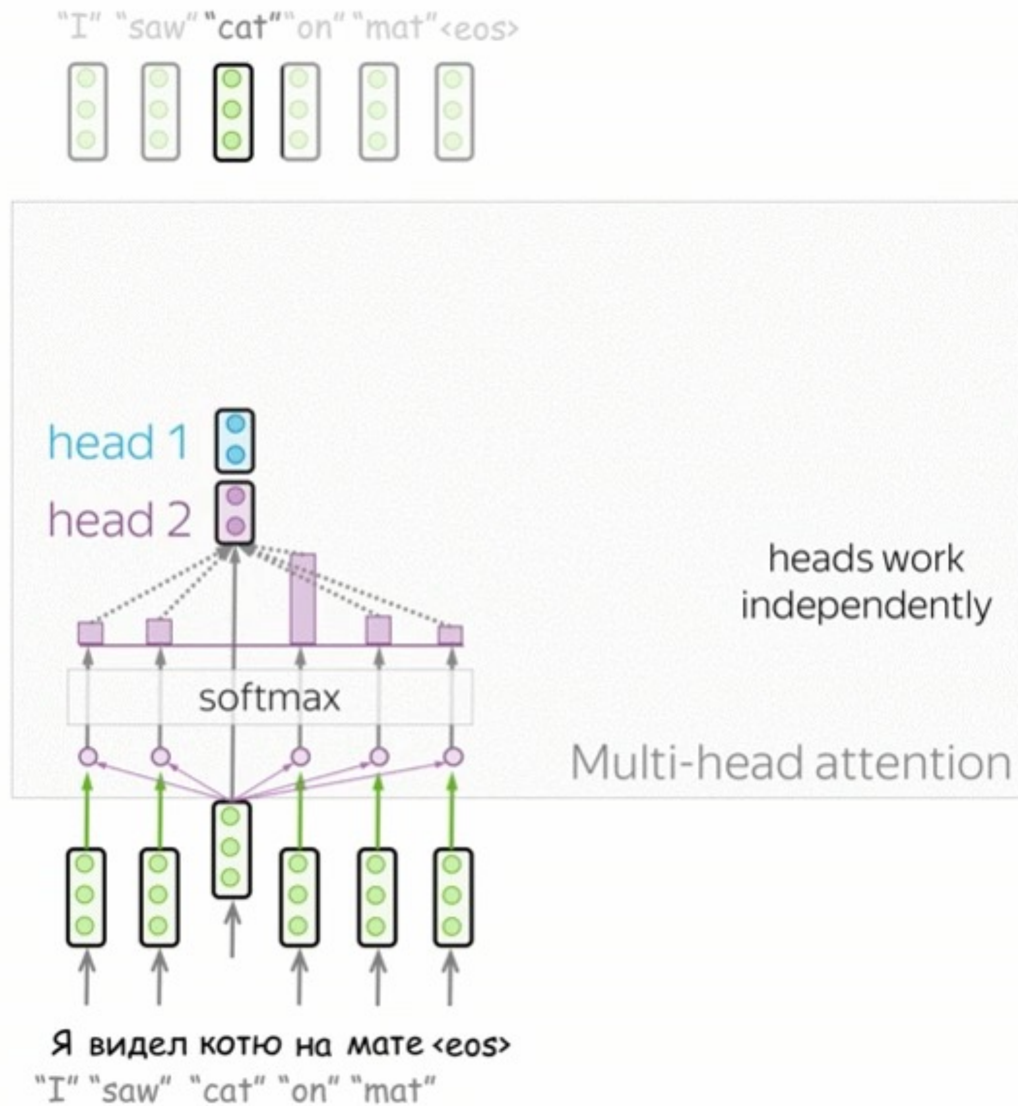
Intuition: there are different relations between words in a sentence (e.g., subject 'affects' a verb in a different way than its object)



Multi-Head Attention

Each head specializes on a different relation

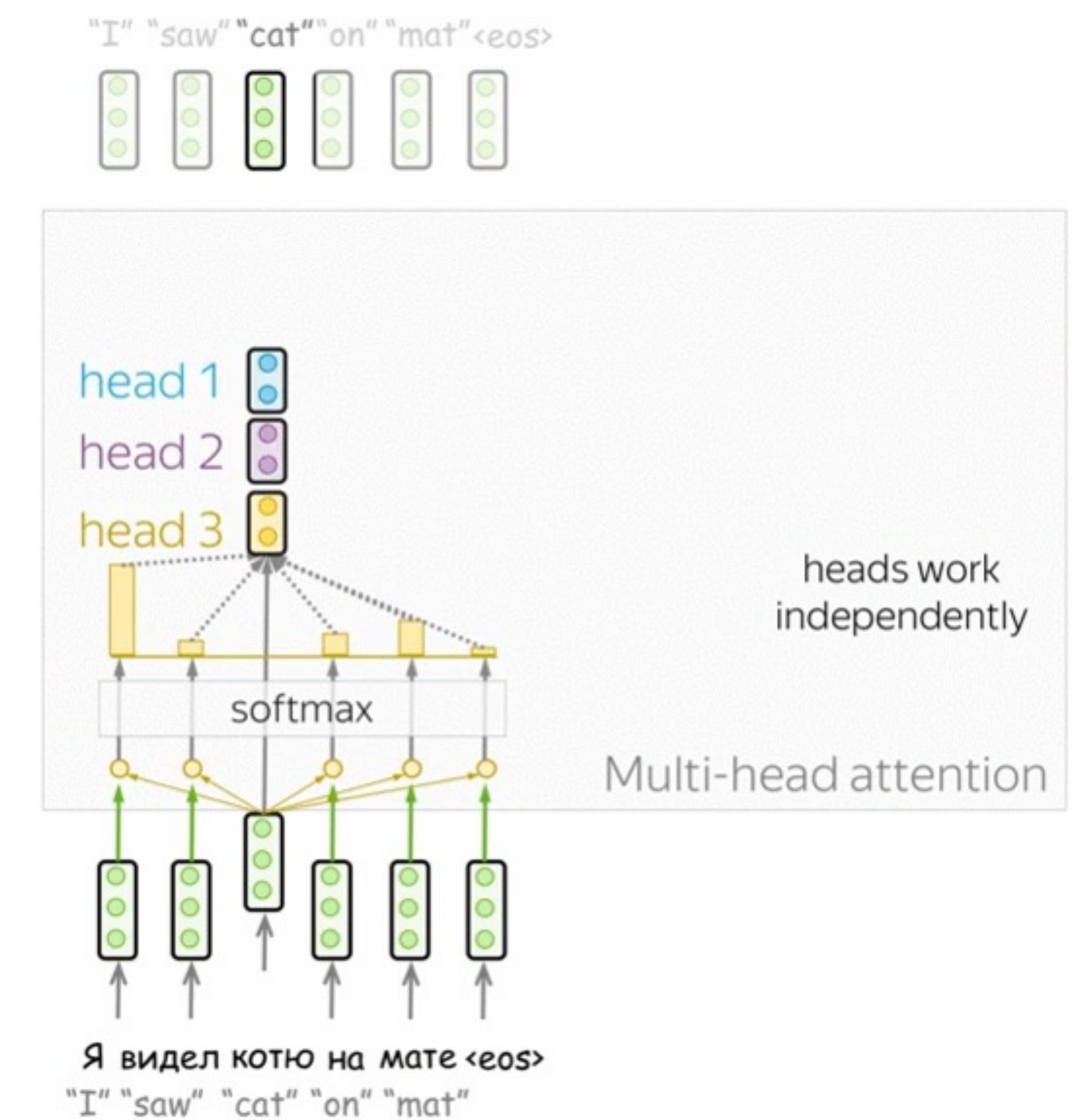
Intuition: there are different relations between words in a sentence (e.g., subject 'affects' a verb in a different way than its object)



Multi-Head Attention

Each head specializes on a different relation

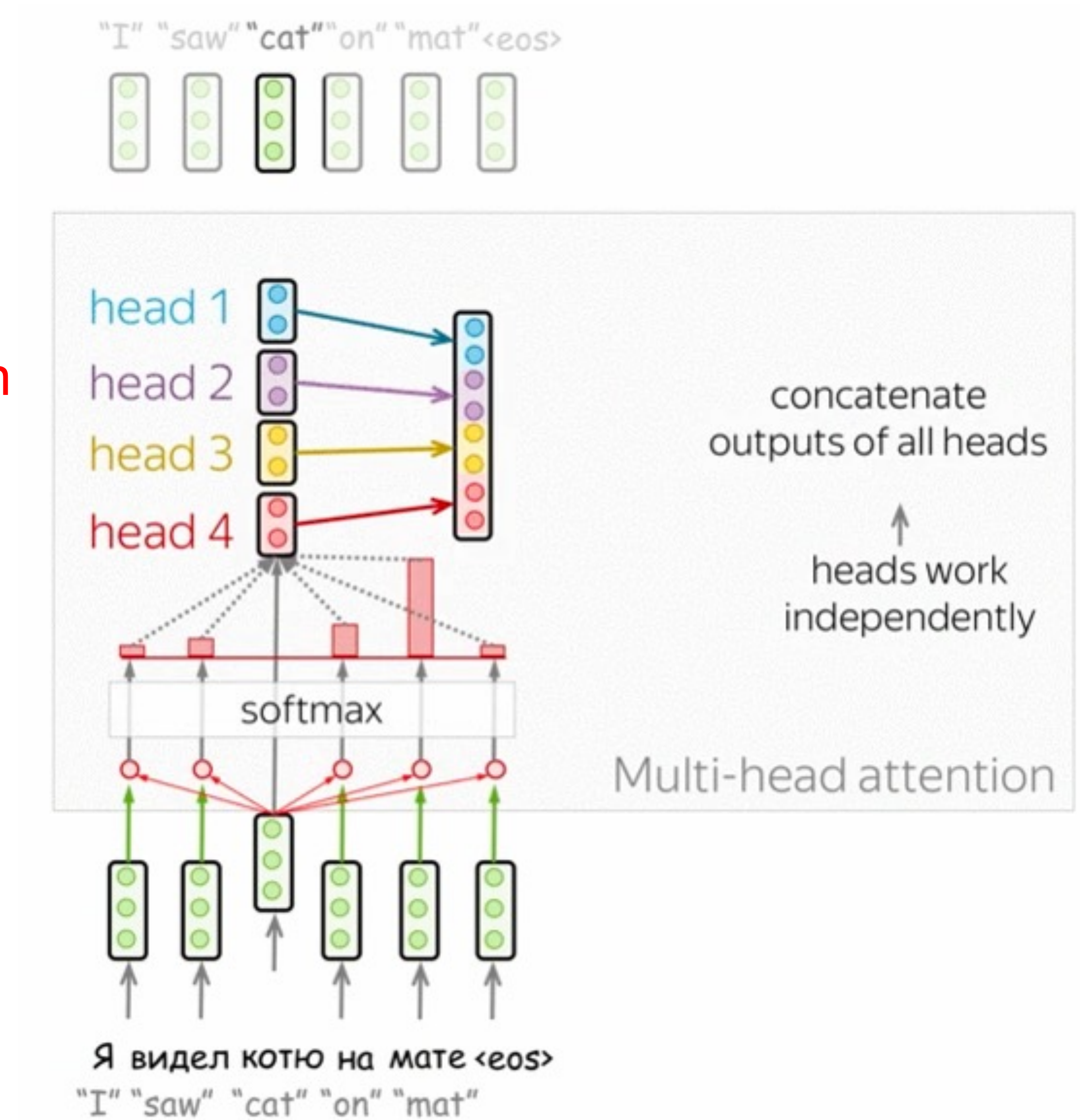
Intuition: there are different relations between words in a sentence (e.g., subject ‘affects’ a verb in a different way than its object)



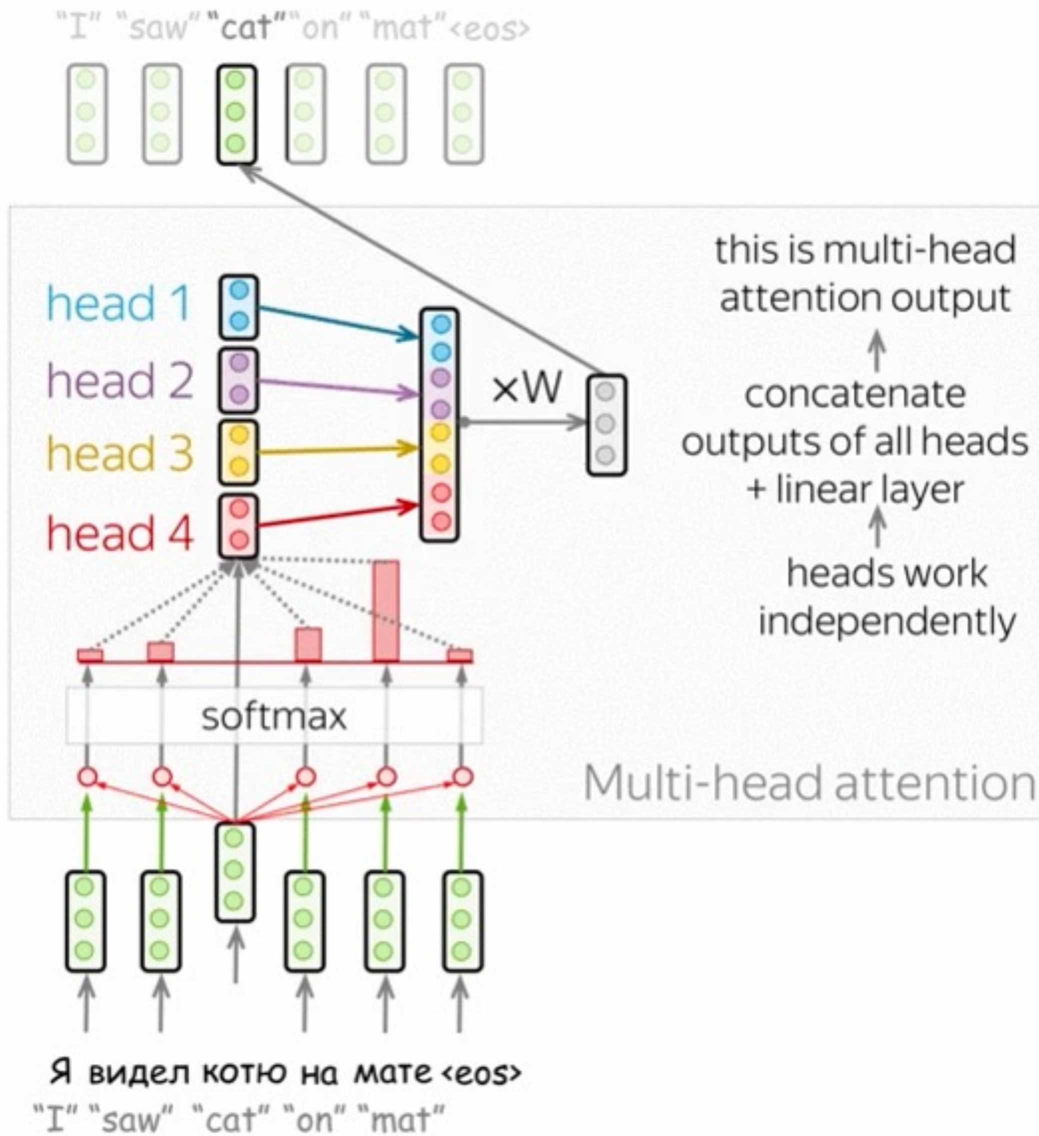
Multi-Head Attention

Each head specializes on a different relation

Intuition: there are different relations between words in a sentence (e.g., subject ‘affects’ a verb in a different way than its object)



Multi-Head Attention



Each heads performs independent QKV attention (with their own head-specific parameters, i.e. W_k, W_q, W_v matrices)

Multi-Head Attention

Let's say *Q* – is the set of states we look **from**
K - is the set of states we look **at**
V - is the set of states we **take values from** (usually *K*=*V*)

In encoder self-attention *Q*=*K*=*V*; *they are represented as matrices (states = rows)*

Multi-Head Attention

Let's say Q – is the set of states we look **from**
K - is the set of states we look **at**
V - is the set of states we **take values from** (usually $K=V$)

In encoder self-attention $Q=K=V$; *they are represented as matrices (states = rows)*

Each head has its own head-specific parameters, i.e. W_k, W_q, W_v matrices

$$\mathrm{head}_i = \mathrm{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

Multi-Head Attention

Let's say Q – is the set of states we look **from**
K - is the set of states we look **at**
V - is the set of states we **take values from** (usually K=V)

In encoder self-attention Q=K=V; *they are represented as matrices (states = rows)*

Each head has its own head-specific parameters, i.e. W_k, W_q, W_v matrices

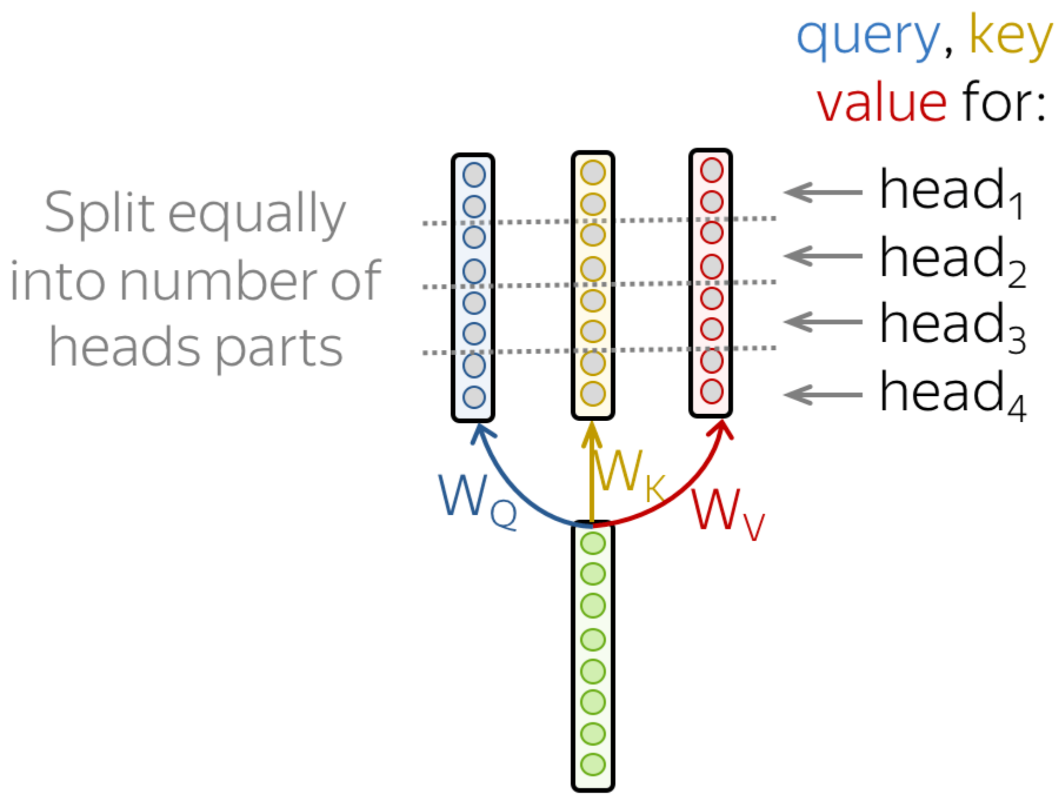
$$\mathrm{head}_i = \mathrm{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

Then, the results are concatenated

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_n)W_o$$

Multi-Head Attention

In practice:



Multi-Head Attention

Each head has its own head-specific parameters, i.e. W_k, W_q, W_v matrices

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_n) W_o,$$

$$\mathrm{head}_i = \mathrm{Attention}(Q W_Q^i, K W_K^i, V W_V^i)$$

If we treat softmax scores as ‘constant’ (of course, they are not constant): the result is a linear function of the token representations.

Mult-Head Attention only weights and transforms them

Have you noticed something strange?
(think of properties of linear transformations)

Multi-Head Attention

Each head has its own head-specific parameters, i.e. W_k, W_q, W_v matrices

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_n) W_o,$$

$$\mathrm{head}_i = \mathrm{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

Token representations are multiplied through W_v x W_o

Two linear transformations are still a linear transformation

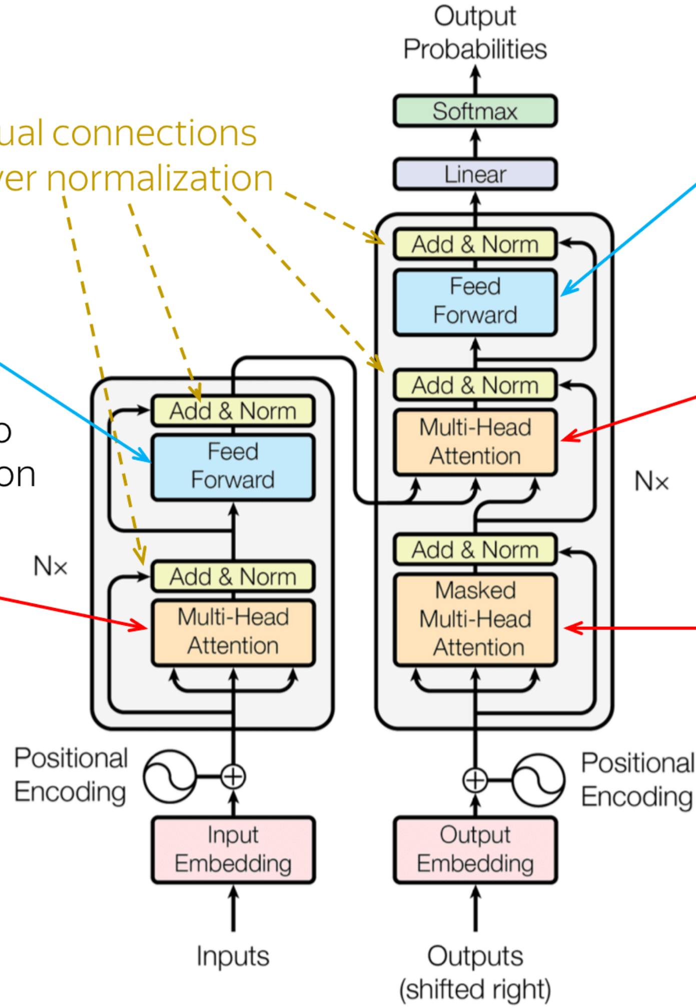
Transformer architecture

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

Encoder self-attention: tokens look at each other

queries, keys, values are computed from encoder states

Residual connections and layer normalization



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

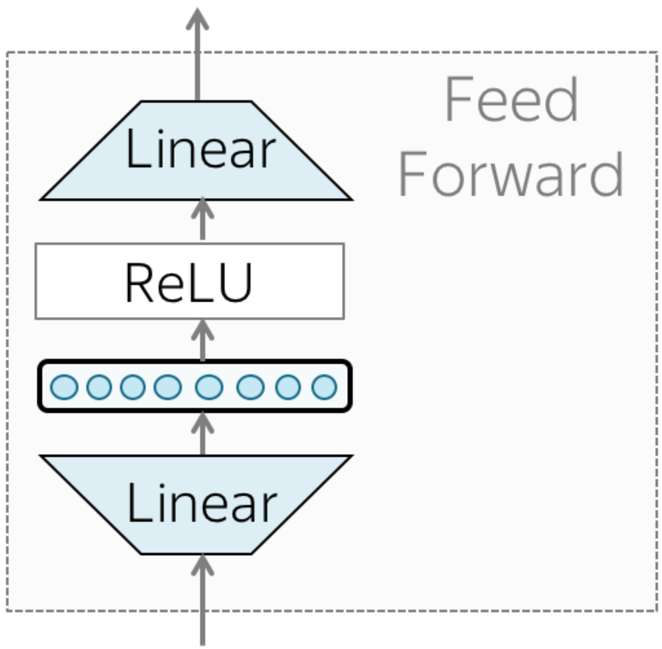
Decoder-encoder attention: target token looks at the source

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

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

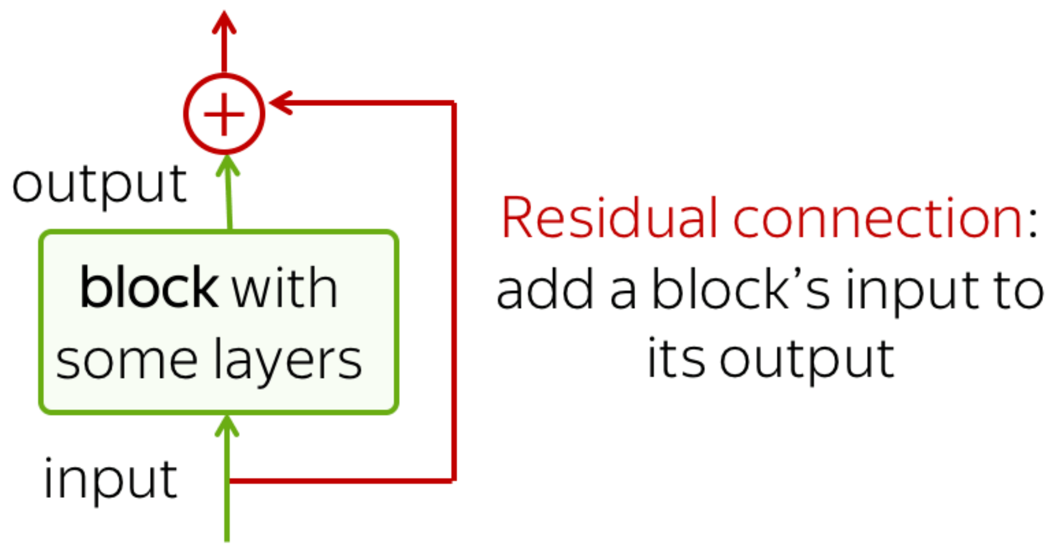
queries, keys, values are computed from decoder states

Feedforward blocks



$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Residual connections



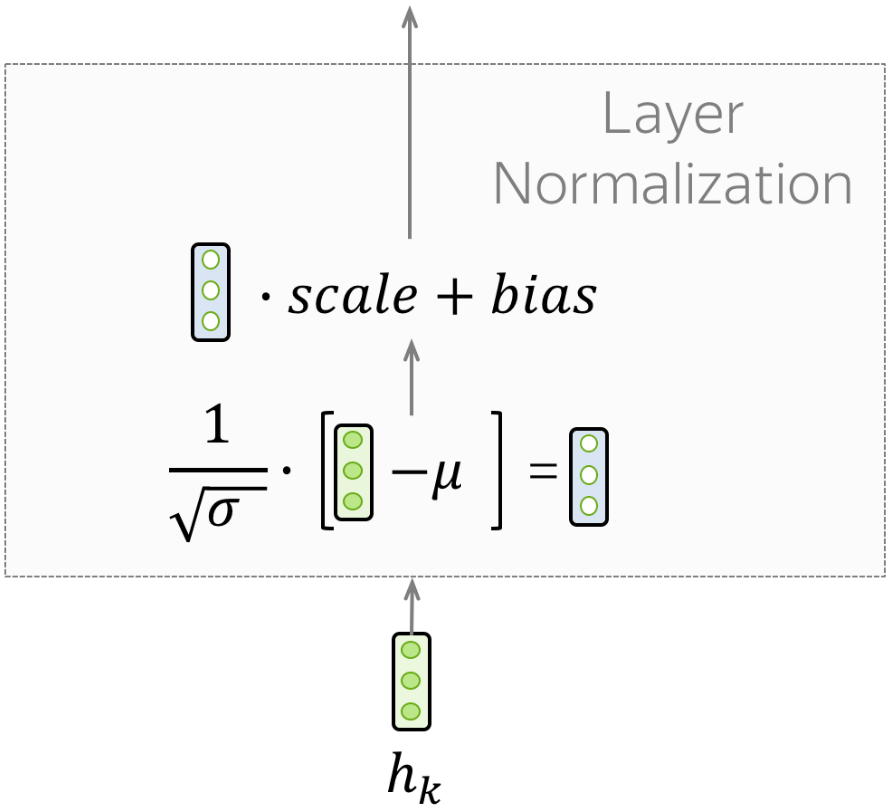
Recall, we considered them for CNNs

Enable learning of deep architectures (i.e. many layers)

With residuals, non-adjacent modules 'communicate' between each other through the 'residual channel' or a module can directly send information to the top level

Layer Normalization

LayerNorm improves training stability (but there are alternatives to LayerNorm)



μ - is the mean of a token representation h_k (across its dimension)

σ - is the standard deviation, computed analogously

scale and *bias* are trainable parameters

One layer – *residual stream* for a token

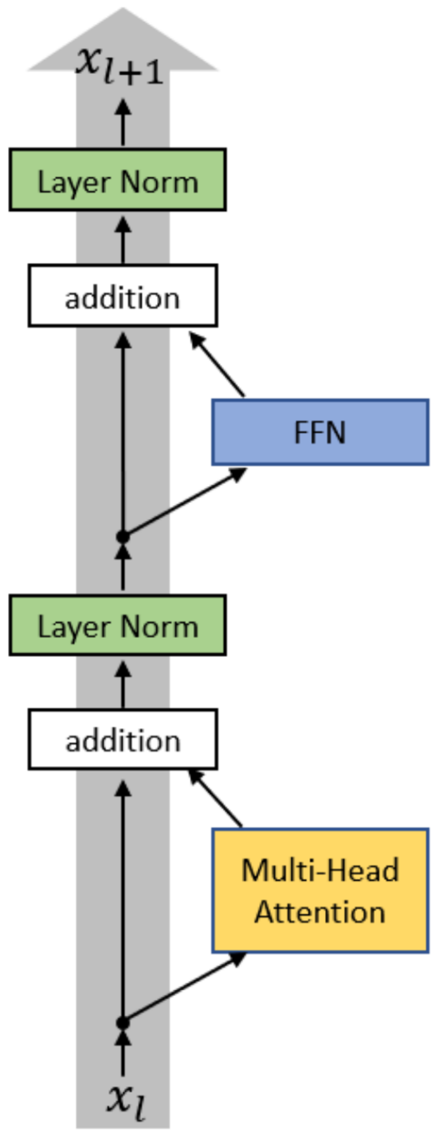
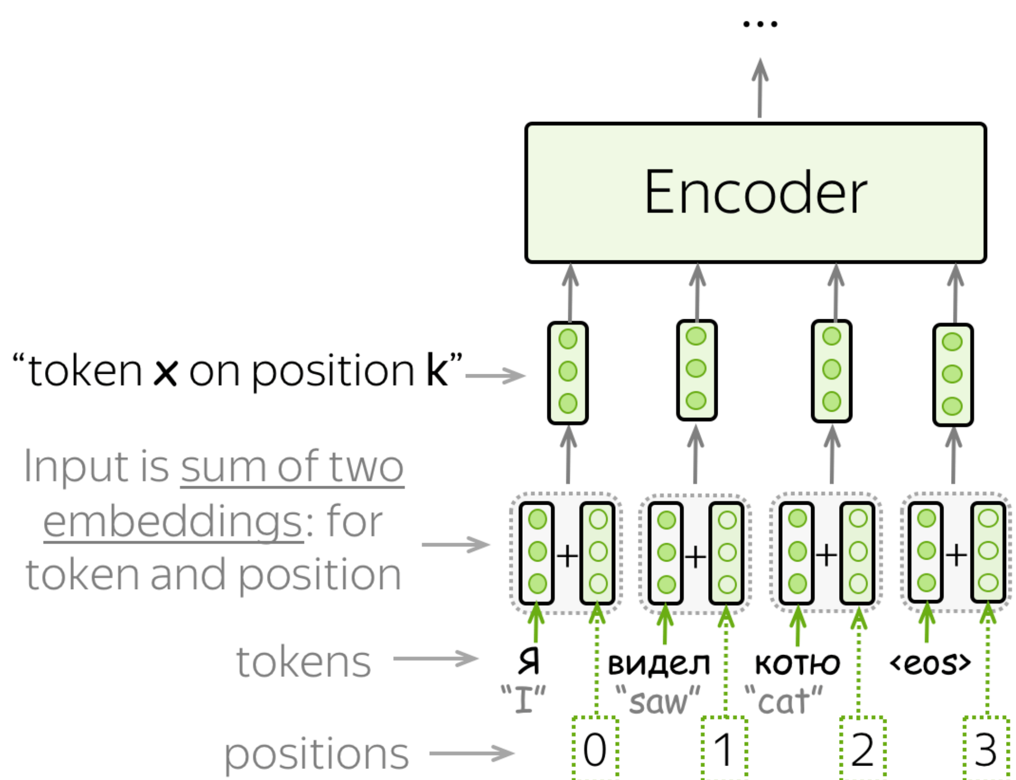


Image from Xiong et al., 2020

Positional Encoding

Transformers (unlike RNNs) do not have a notion of order of the tokens. To incorporate information about the order, we use ‘*position embeddings*’



In the simplest form, position embeddings are just a collection of vectors, one per a position (like word embeddings: one per word type). But better approaches exist

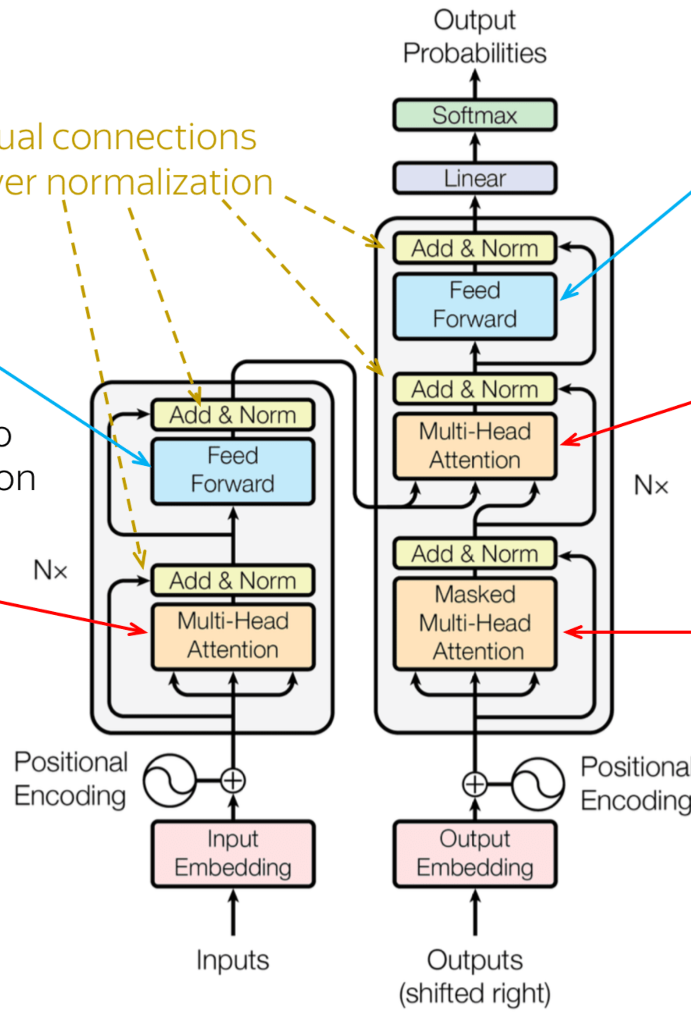
We should now know every block!

Residual connections
and layer normalization

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

Encoder self-attention:
tokens look at each other

queries, keys, values
are computed from
encoder states



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

Decoder-encoder attention:
target token looks at the source

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

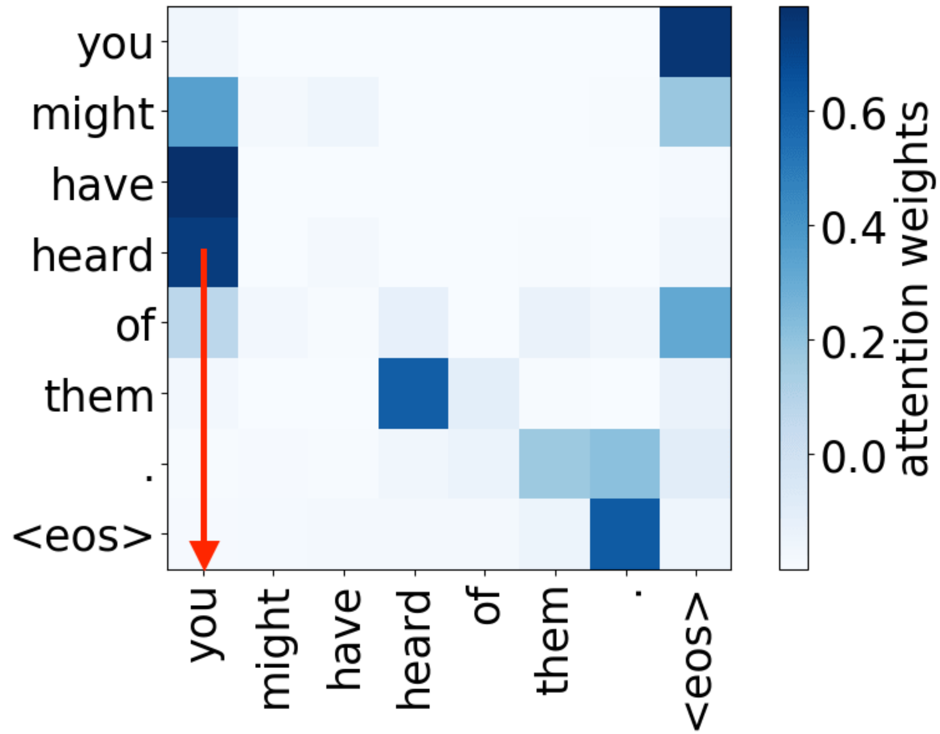
Decoder self-attention (masked):
tokens look at the previous tokens

queries, keys, values are computed
from decoder states

Interpretability

Recall, individual heads focus on different tokens (have their individual attention modules)

It turns out many heads have interpretable roles



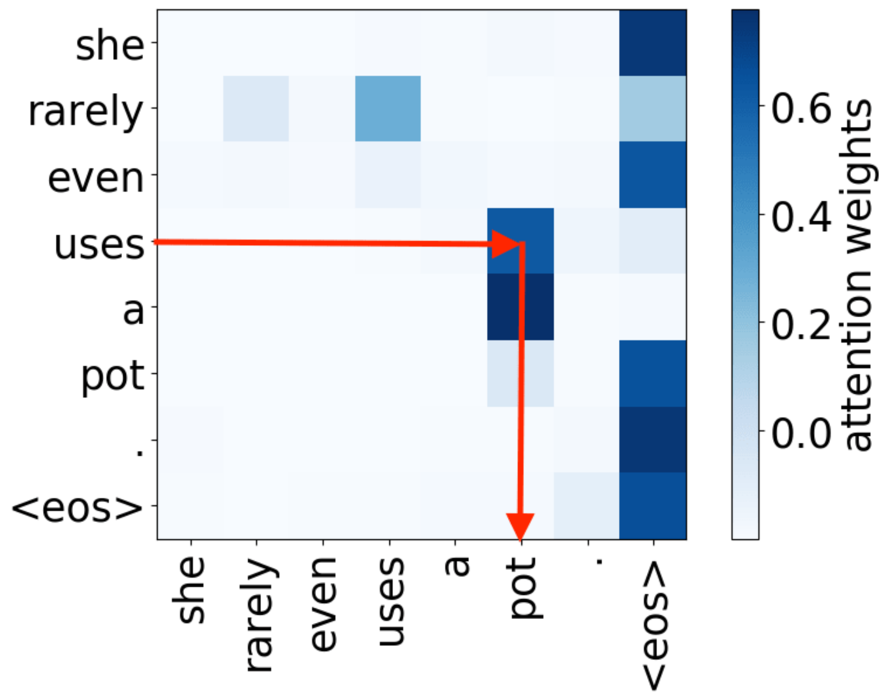
verb->subject

Encoder of a machine translation Transformer model

Interpretability

Recall, individual heads focus on different tokens (have their individual attention modules)

It turns out many heads have interpretable roles



verb->object

Encoder of a machine translation Transformer model

Take-aways

- Transformer is the architecture which powers most of state-of-the-art models in NLP and beyond

- The key component is attention (make sure you understand it)
  … but other components (residual connections, feed-forward, position embeddings, and layer norm) play important roles

- Heads can learn specialized and interpretable functions

- Lots of linearity in Transformers