NLP

Transformer

Основано на NLP Course | For You

План

- seq2seq
- attention
- transformer
- BERT
- attention perfomance
- finetuning

Sequence-to-sequence

Задача

Human Translation

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

The "probability" is intuitive and is given by a human translator's expertise

Machine Translation

model parameters

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

Questions we need to answer

- **modeling**

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

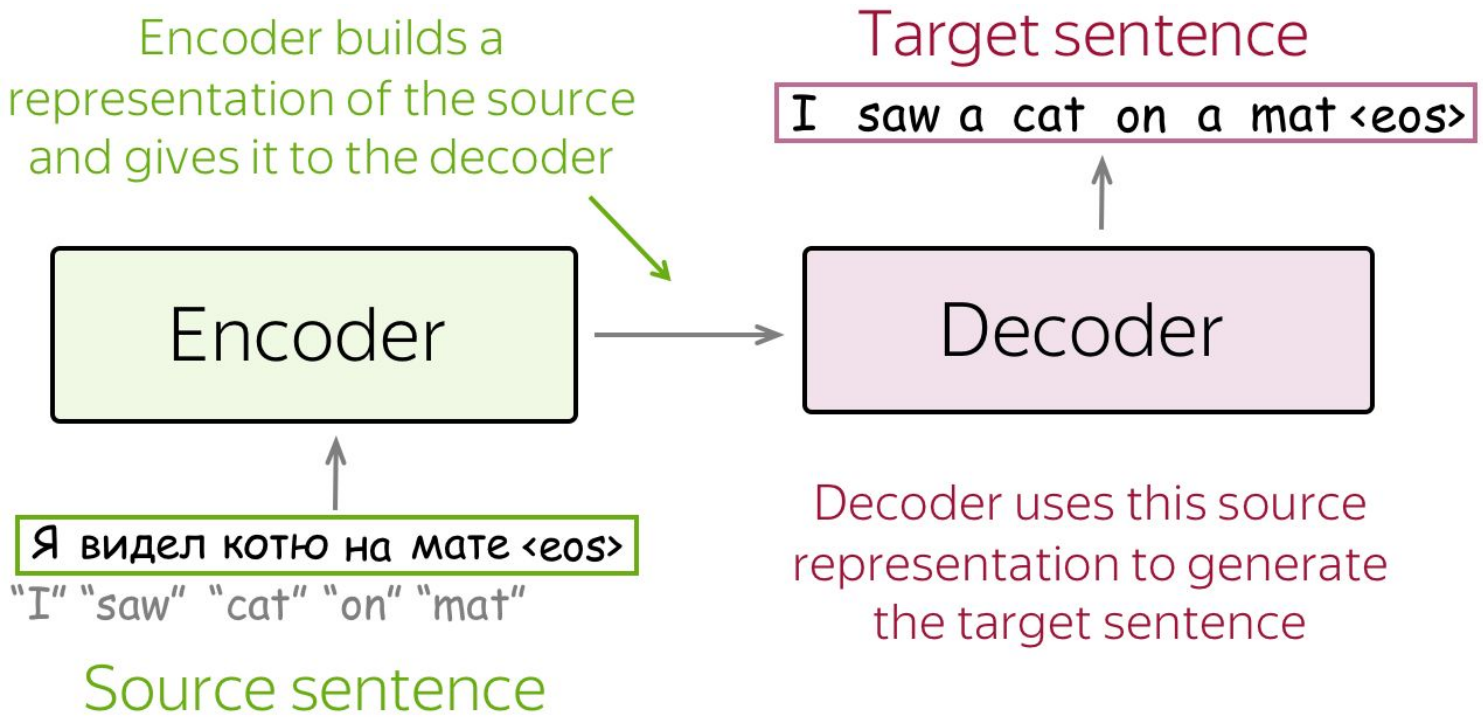
- **learning**

How to find θ?

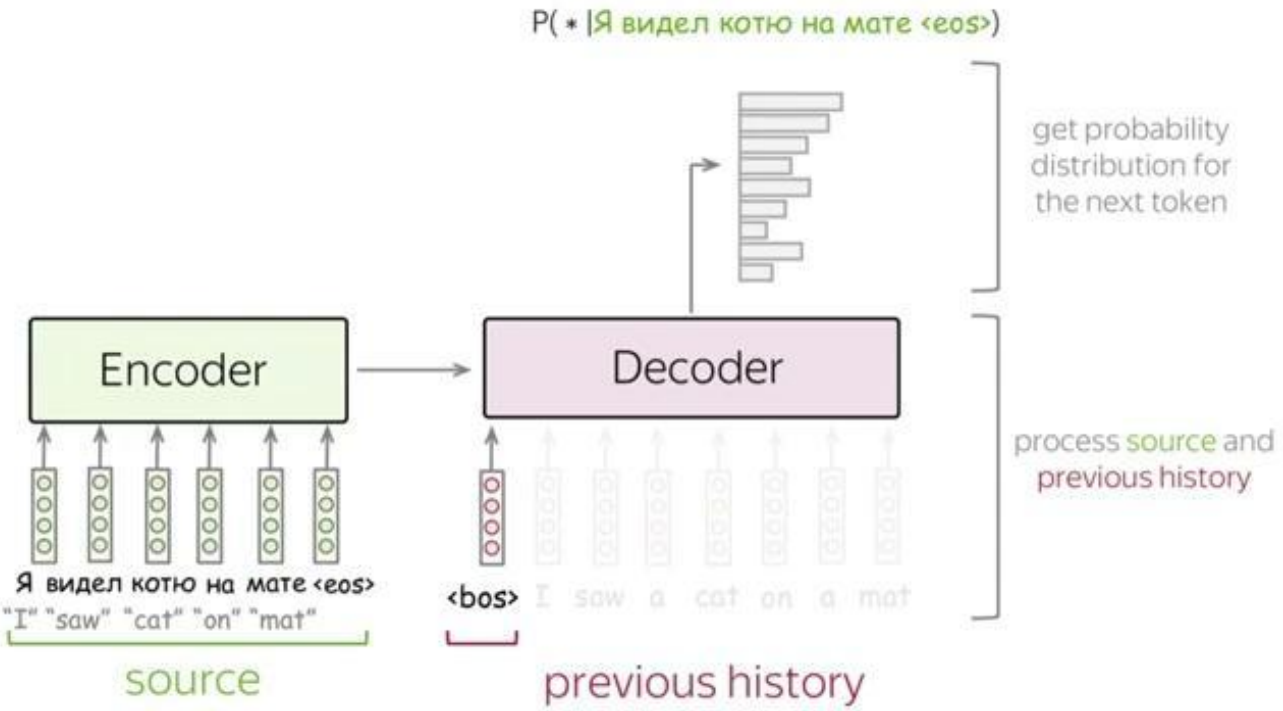
- **search**

How to find the argmax?

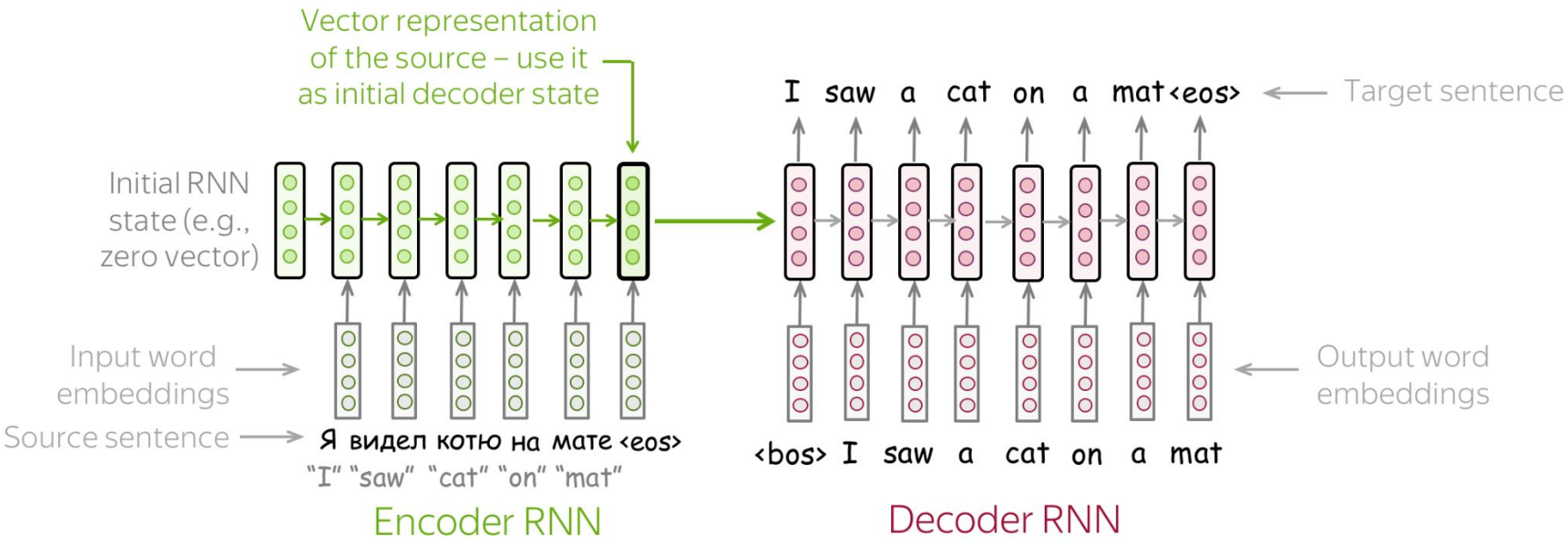
Encoder-decoder



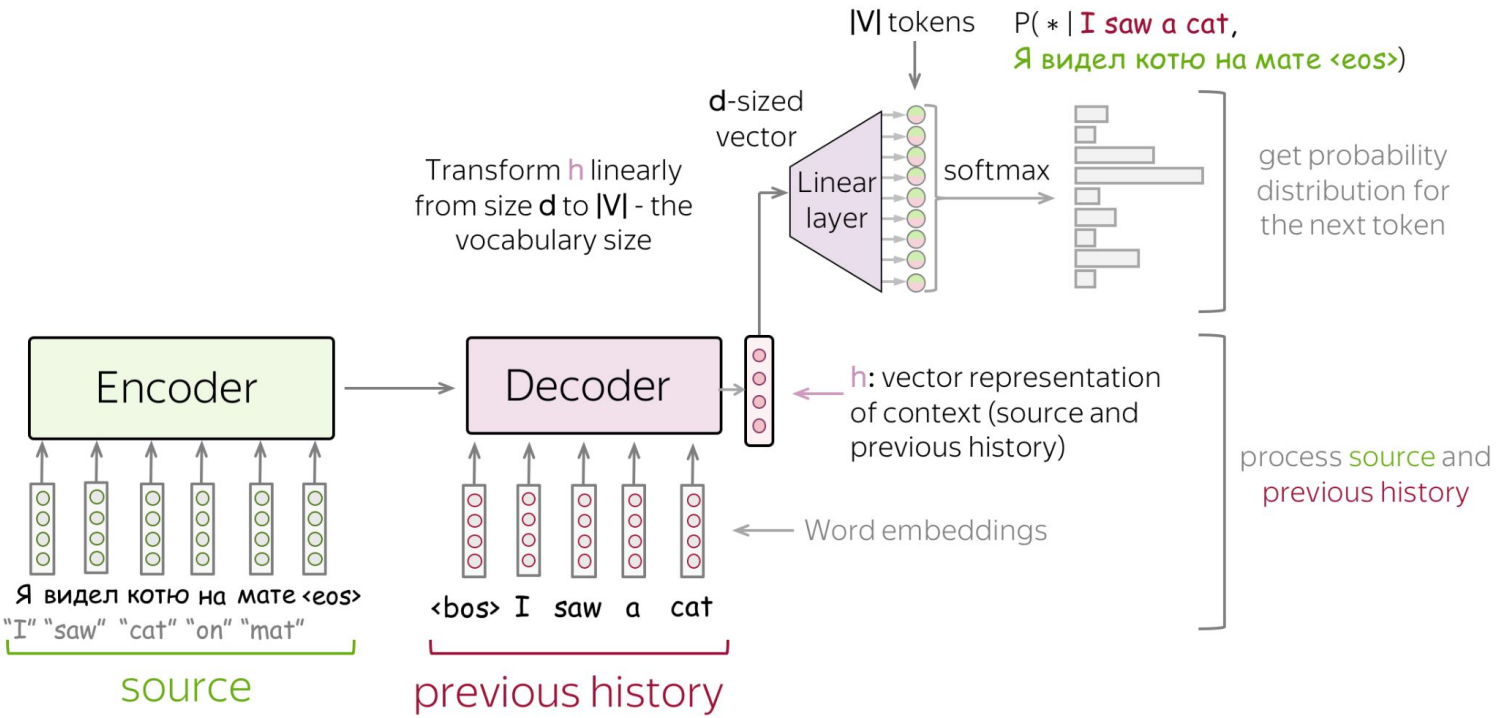
Encoder-decoder



RNN


Vector representation of the source – use it as initial decoder state
Initial RNN state (e.g., zero vector)
Input word embeddings
Source sentence
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
Encoder RNN
I saw a cat on a mat <eos>
Target sentence
<bos> I saw a cat on a mat
Output word embeddings
Decoder RNN

Как генерируем?



Как генерируем?

Source sequence:

Я видел котю на мате <eos>

"I" "saw" "cat" "on" "mat"

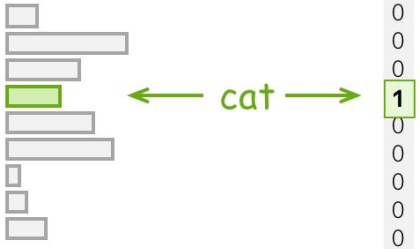
Target sequence: ← one training example

I saw a cat on a mat <eos> ← one step for this example

previous tokens

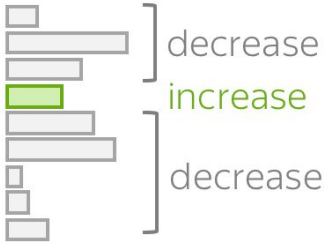
we want the model to predict this

Model prediction: p(* |I saw a, Я ... <eos>)



Target

Loss = -log (p(cat)) → min



Как найти argmax?

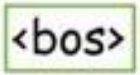
$$y' = \arg\max_y p(y|x) = \arg\max_y \prod_{t=1}^{n} p(y_t|y_{<t}, x)$$

How to find the argmax?

Жадный подход

$$\arg\max_{y} \prod_{t=1}^{n} p(y_t | y_{<t}, x) \neq \prod_{t=1}^{n} \arg\max_{y_t} p(y_t | y_{<t}, x)$$

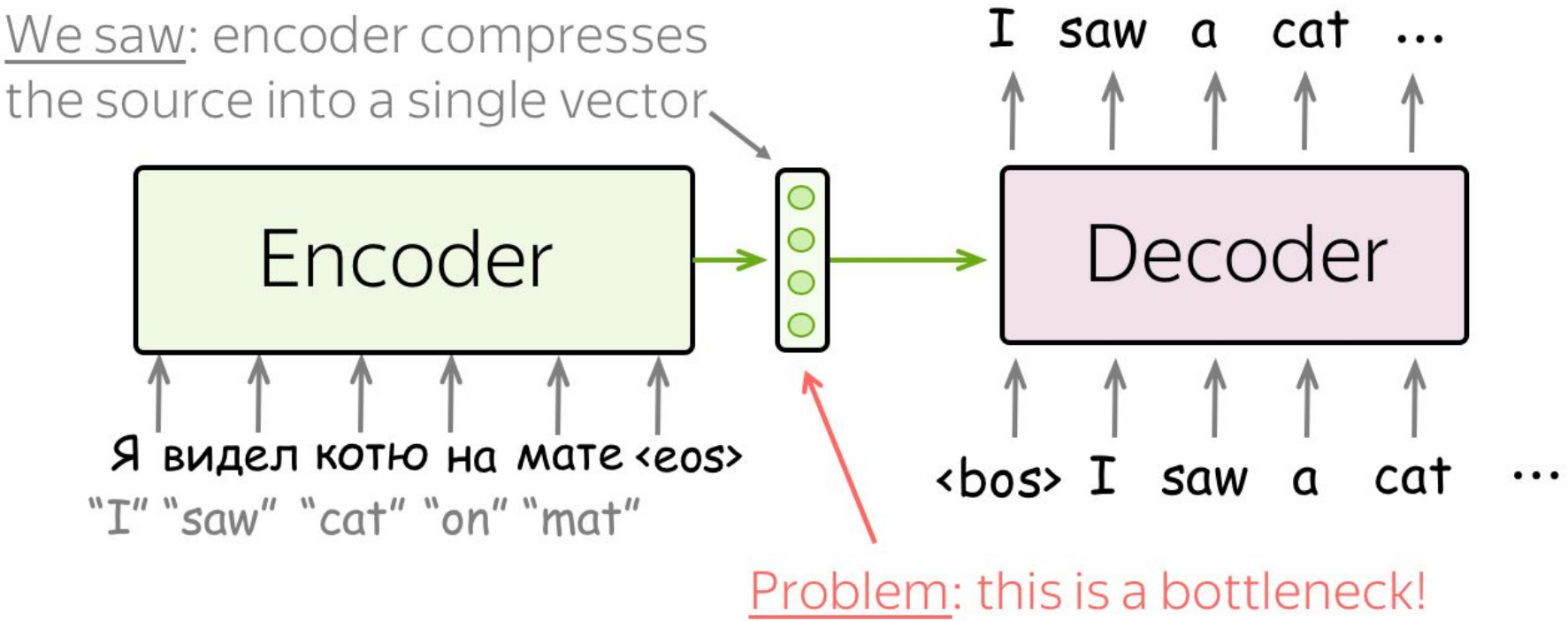
Beam search



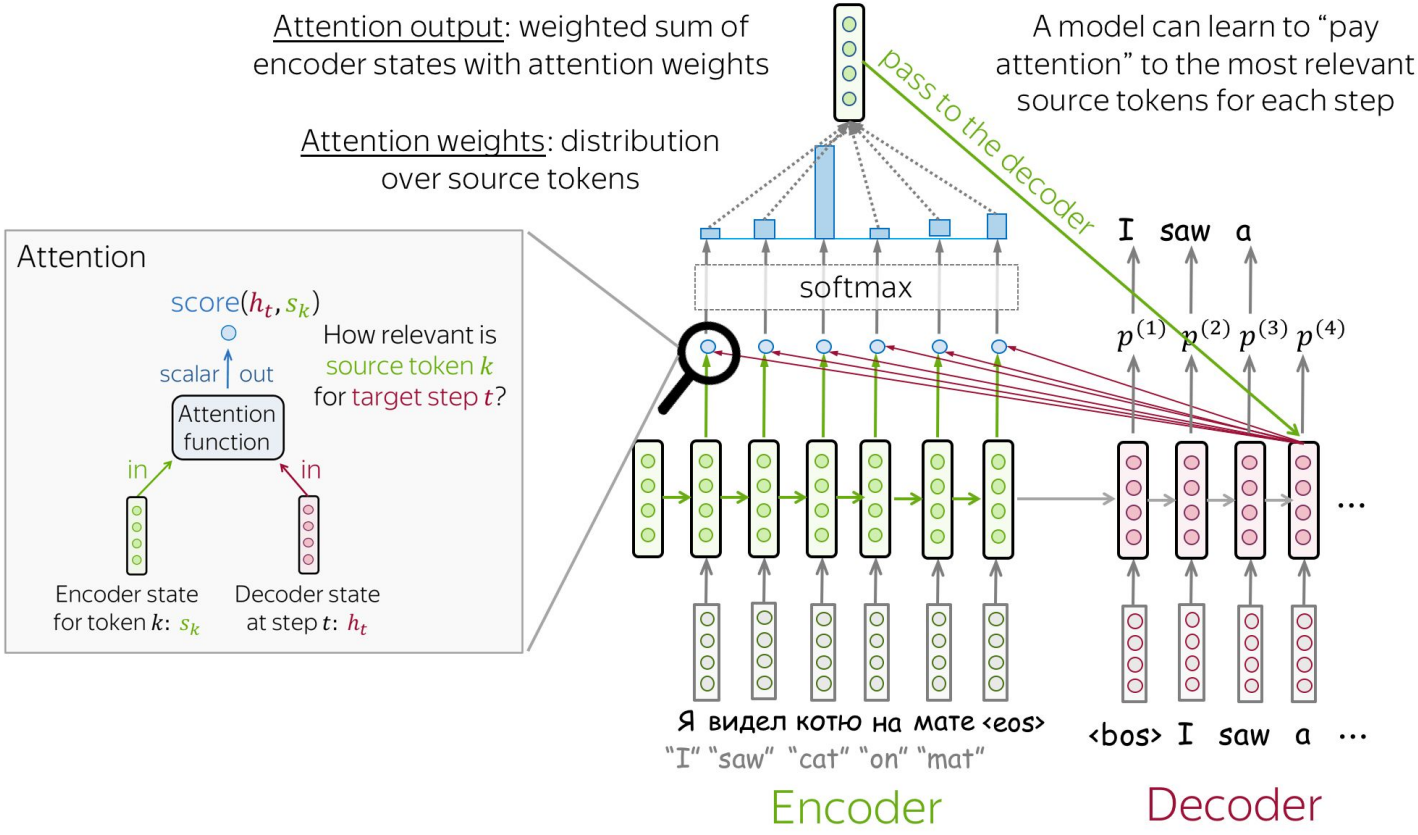
Start with the begin of sentence token or with an empty sequence

Once again генерация

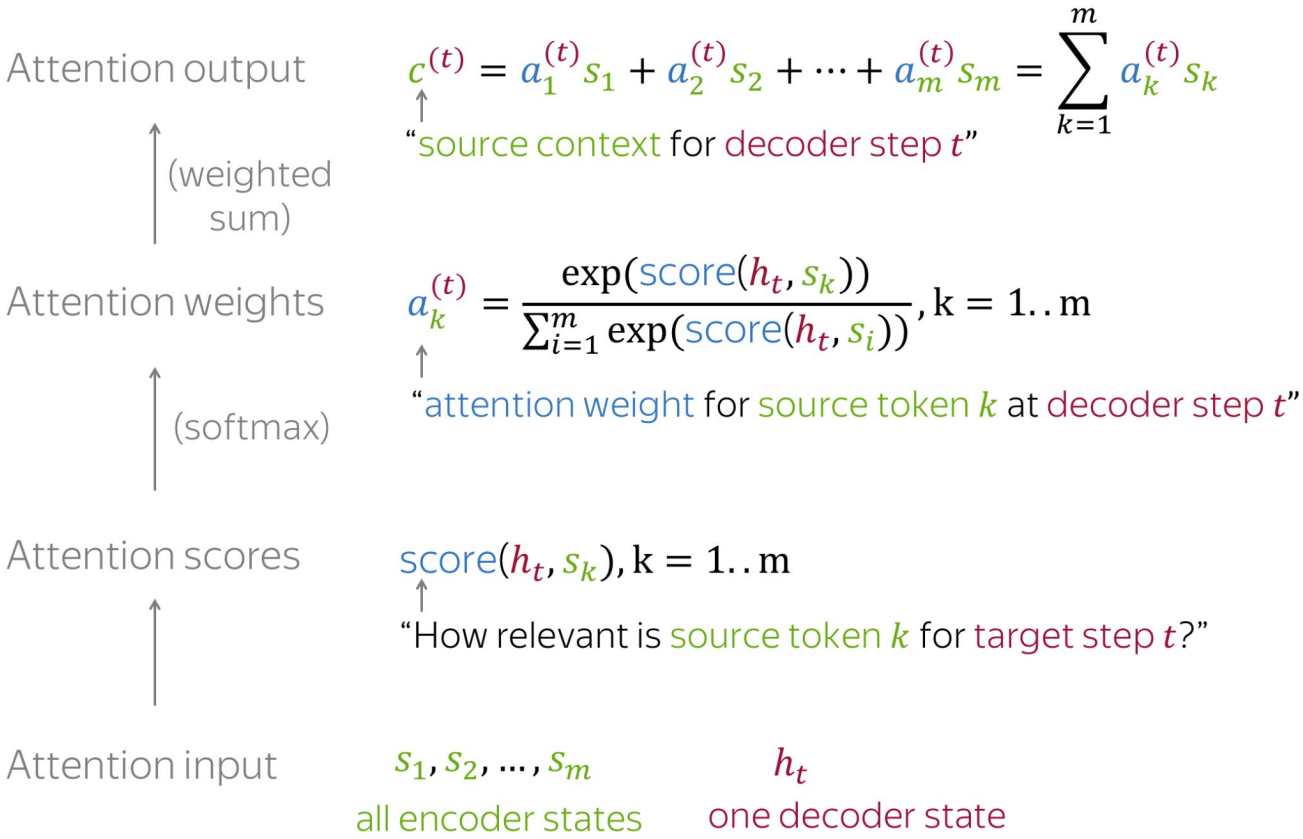
Но есть нюанс

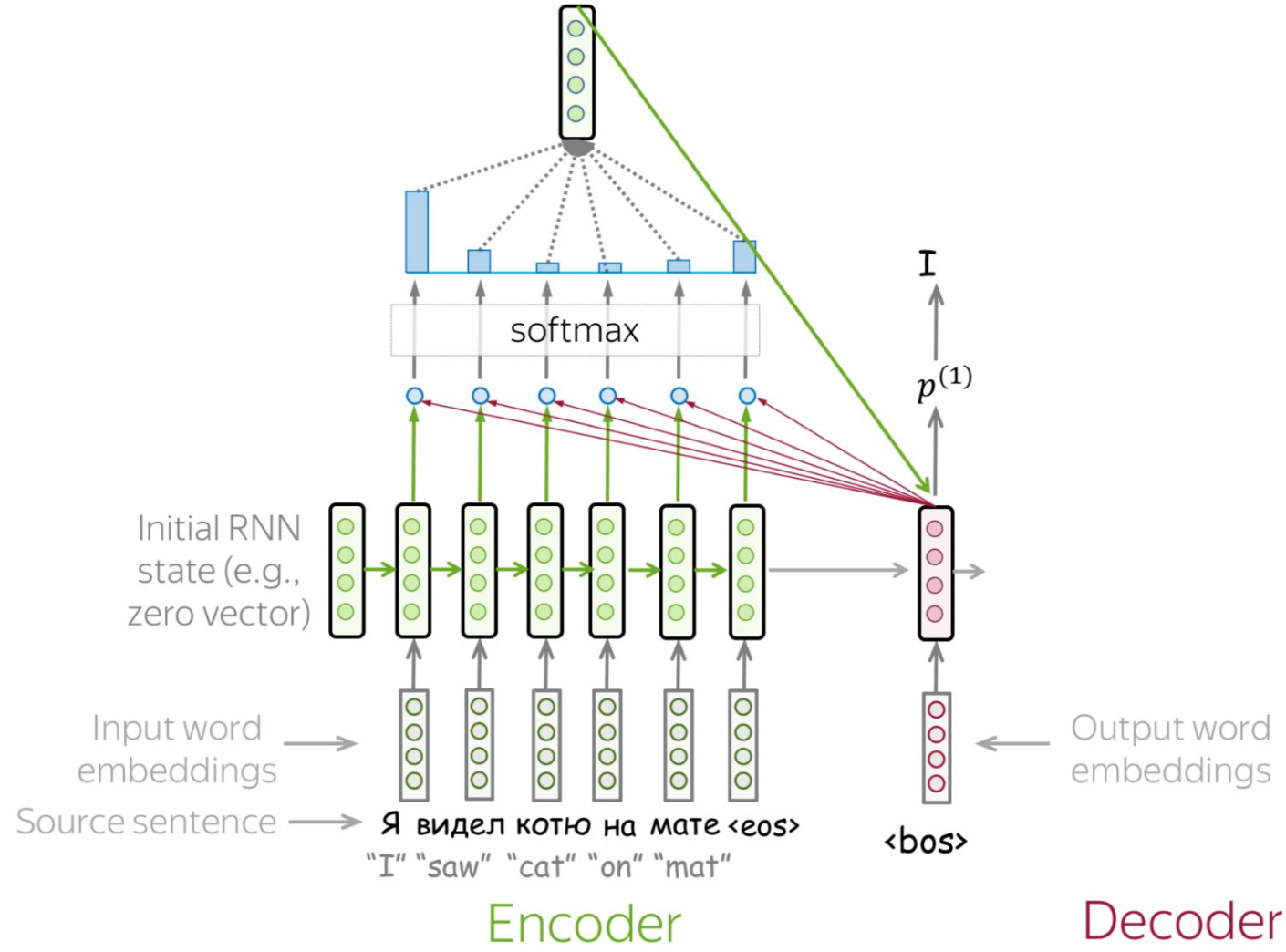


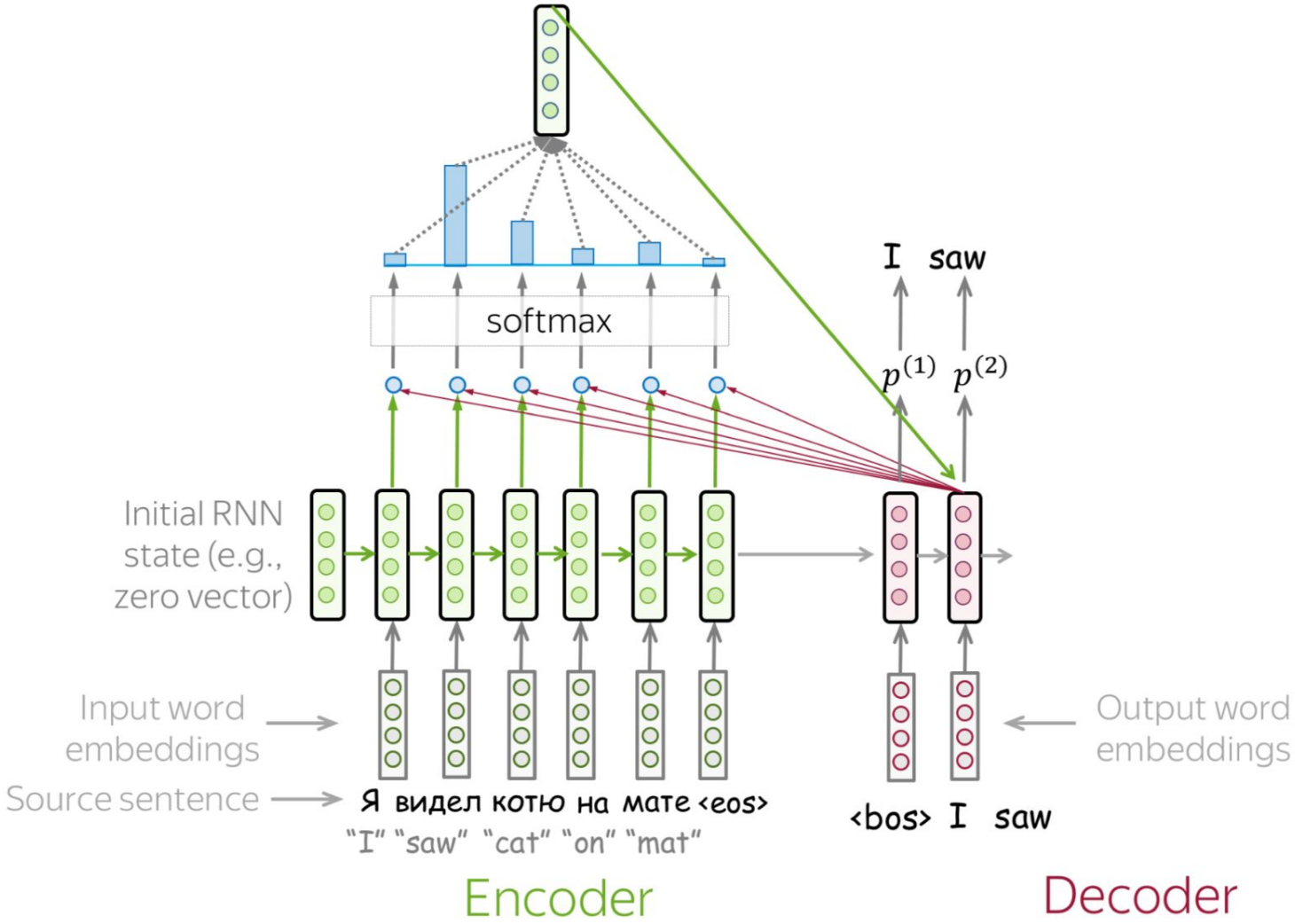
Attention

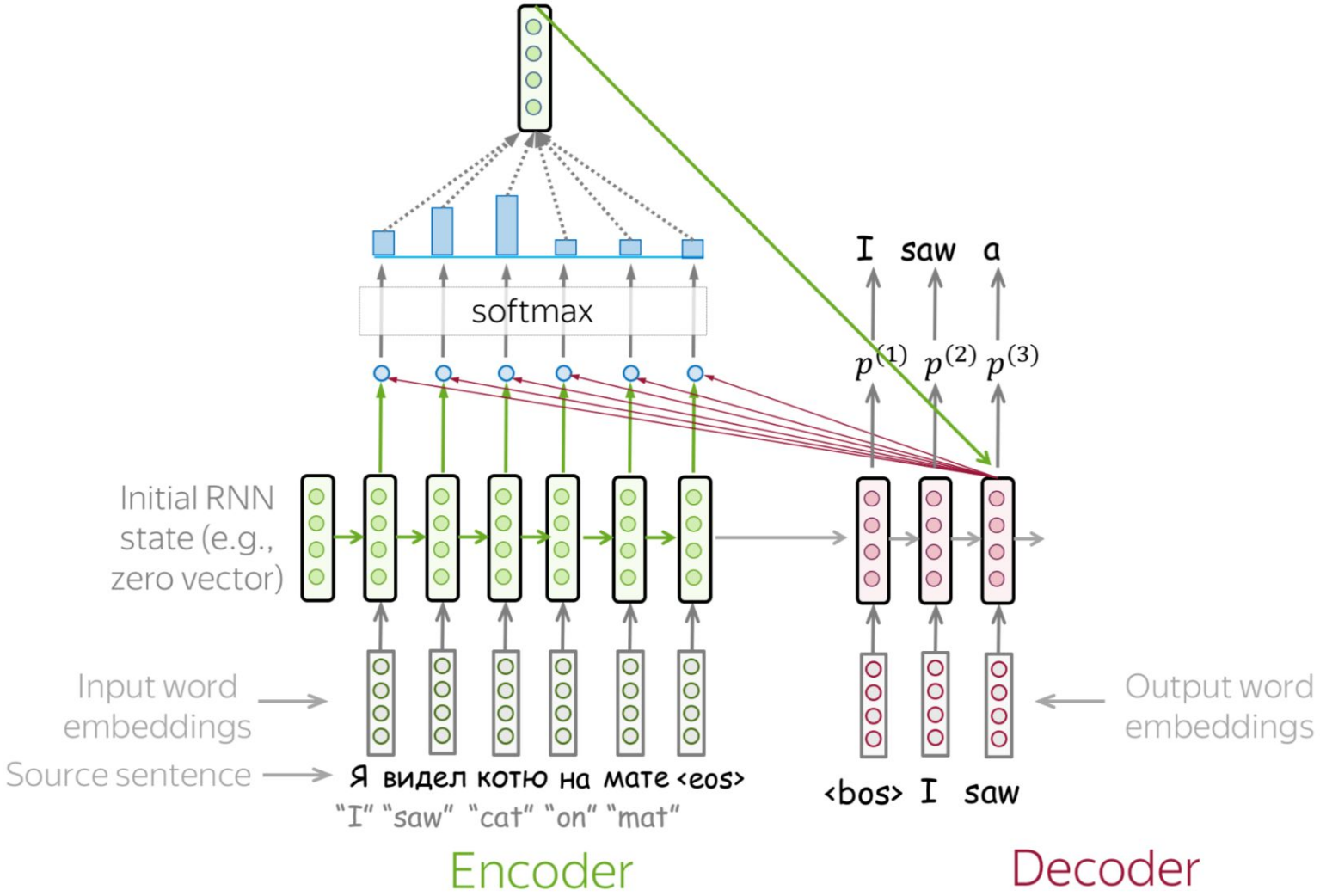

Attention output: weighted sum of encoder states with attention weights
Attention weights: distribution over source tokens
A model can learn to "pay attention" to the most relevant source tokens for each step
pass to the decoder
Attention
score(h_t, s_k)
scalar
out
Attention function
in
in
How relevant is source token k for target step t?
Encoder state for token k: s_k
Decoder state at step t: h_t
softmax
I saw a
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$ $p^{(4)}$
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
<bos> I saw a ...
Encoder
Decoder

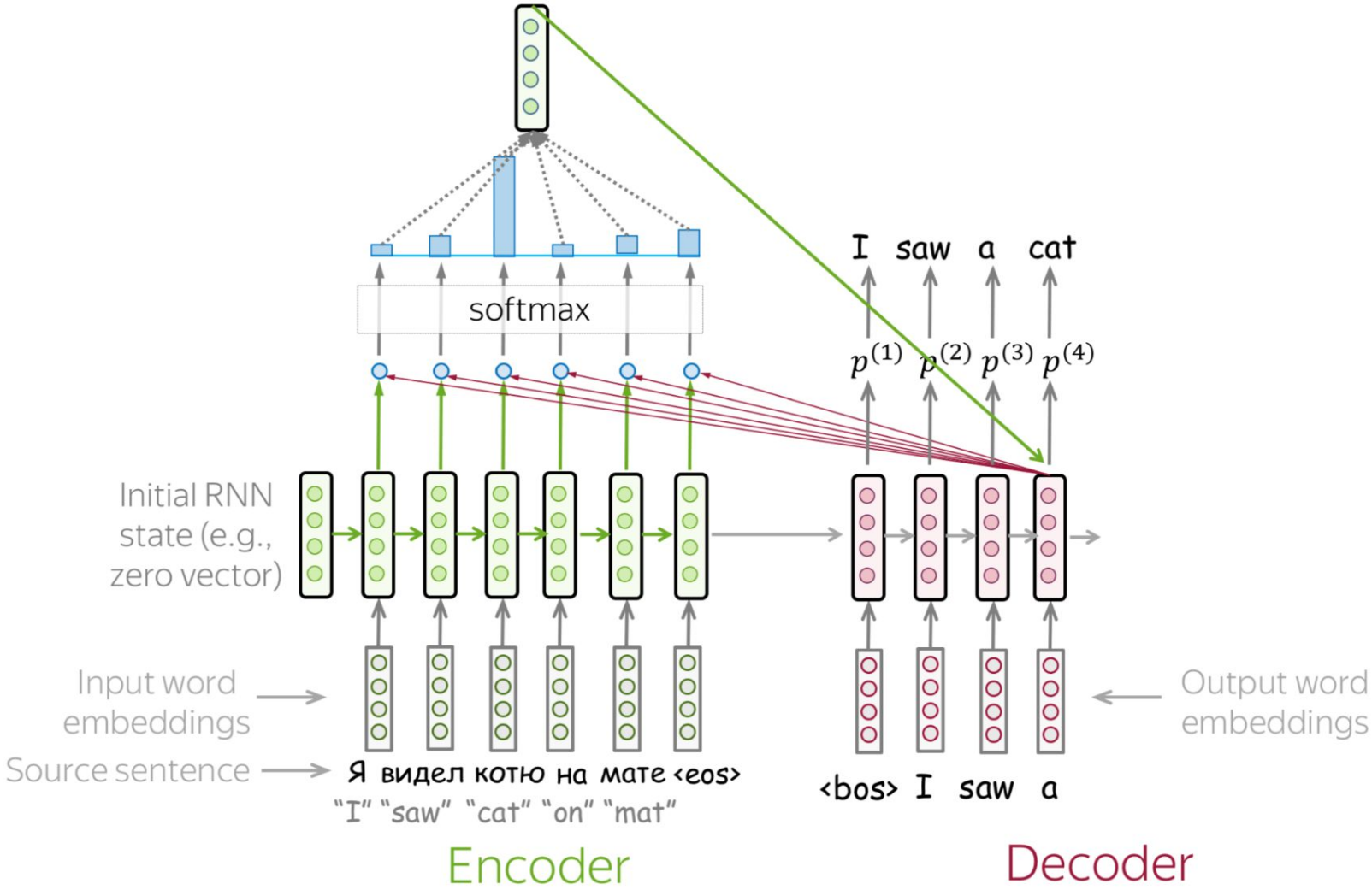
Подсчёт

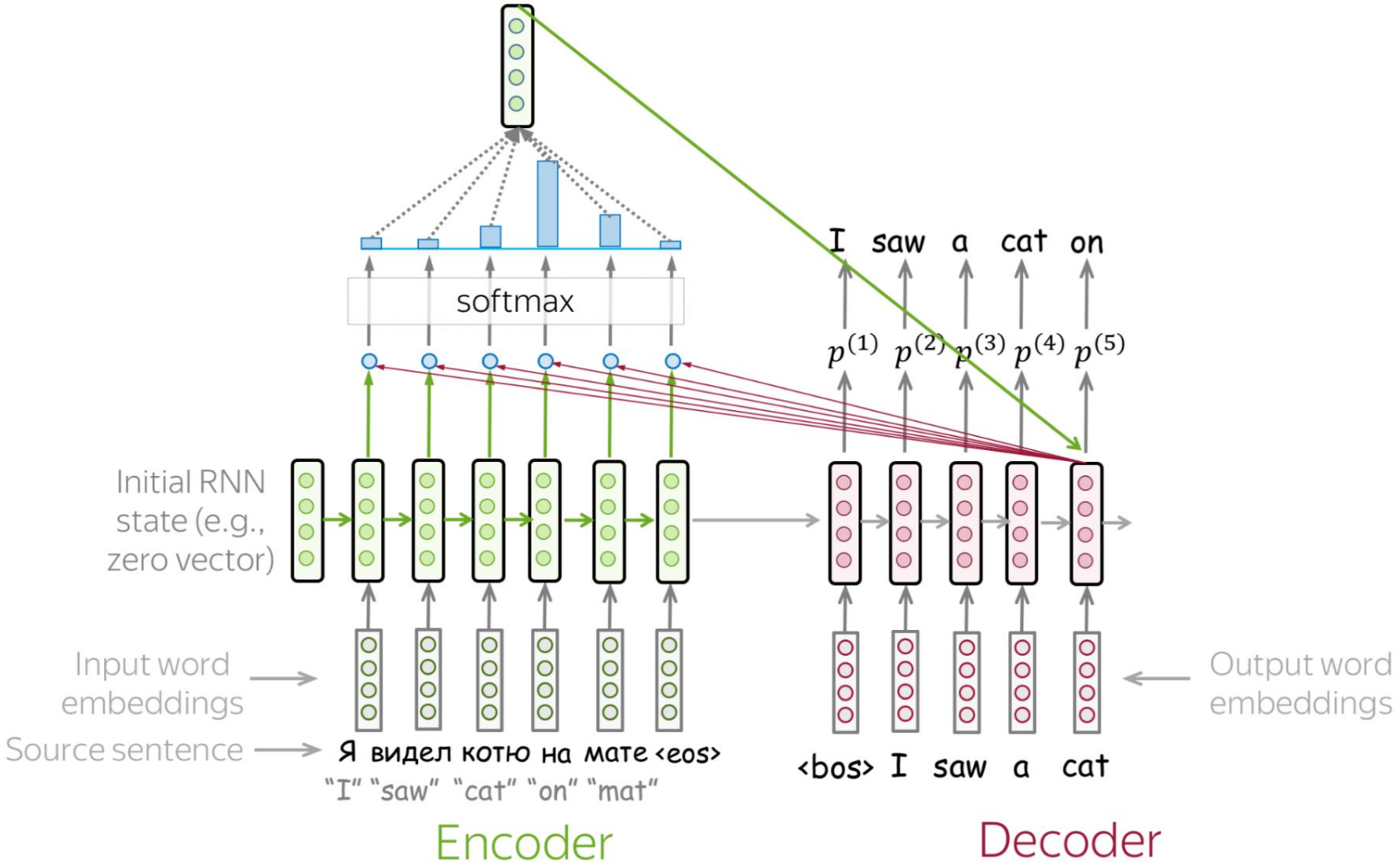

Attention output
$c^{(t)} = a_1^{(t)} s_1 + a_2^{(t)} s_2 + \cdots + a_m^{(t)} s_m = \sum_{k=1}^{m} a_k^{(t)} s_k$
"source context for decoder step t"
(weighted sum)
Attention weights
$a_k^{(t)} = \frac{\exp(\text{score}(h_t, s_k))}{\sum_{i=1}^{m} \exp(\text{score}(h_t, s_i))}, \text{k} = 1..\text{m}$
"attention weight for source token k at decoder step t"
(softmax)
Attention scores
$\text{score}(h_t, s_k), \text{k} = 1..\text{m}$
"How relevant is source token k for target step t?"
Attention input
$s_1, s_2, \ldots, s_m$
all encoder states
h_t
one decoder state

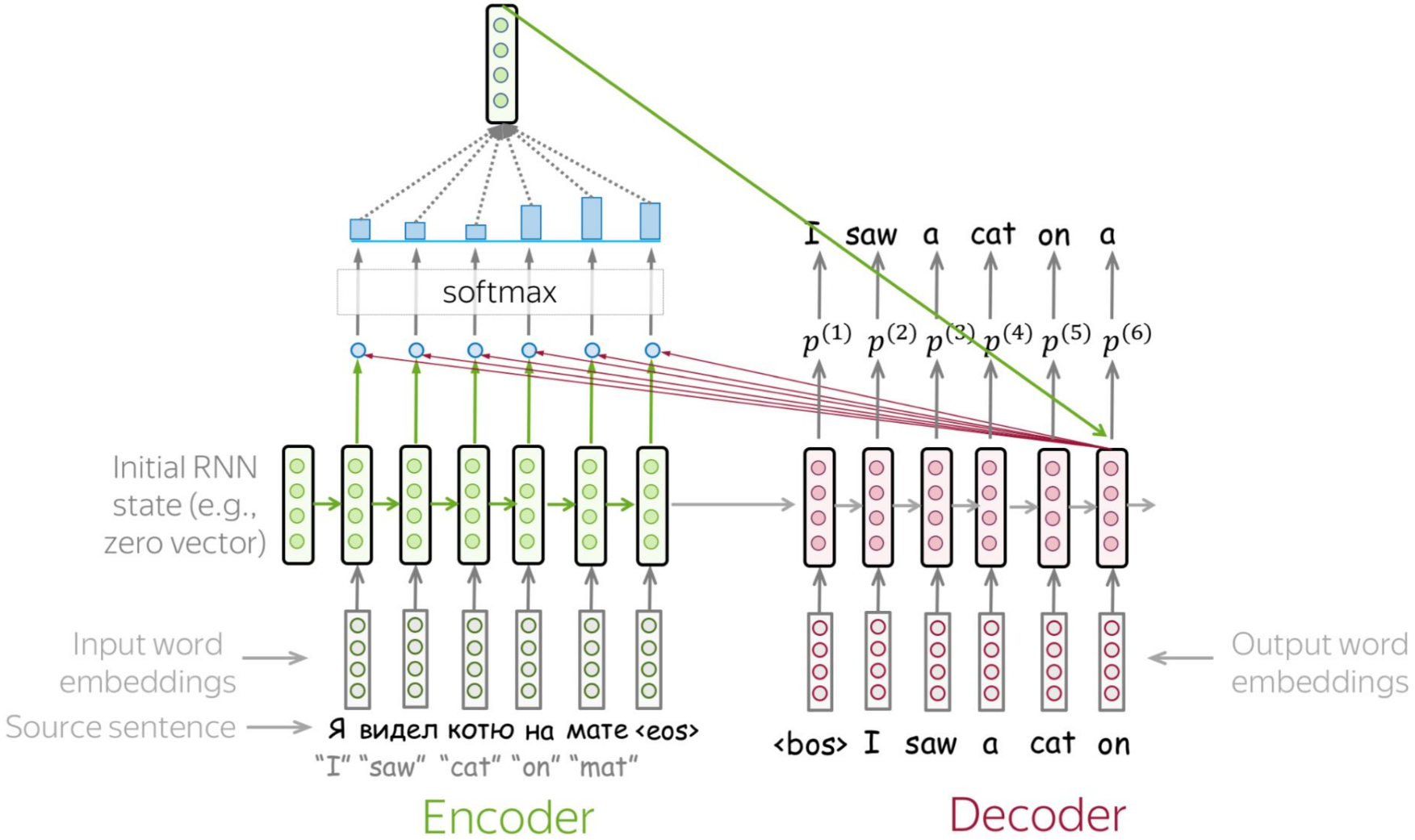

softmax
I
$p^{(1)}$
Initial RNN state (e.g., zero vector)
Input word embeddings
Source sentence
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
Output word embeddings
<bos>
Encoder
Decoder

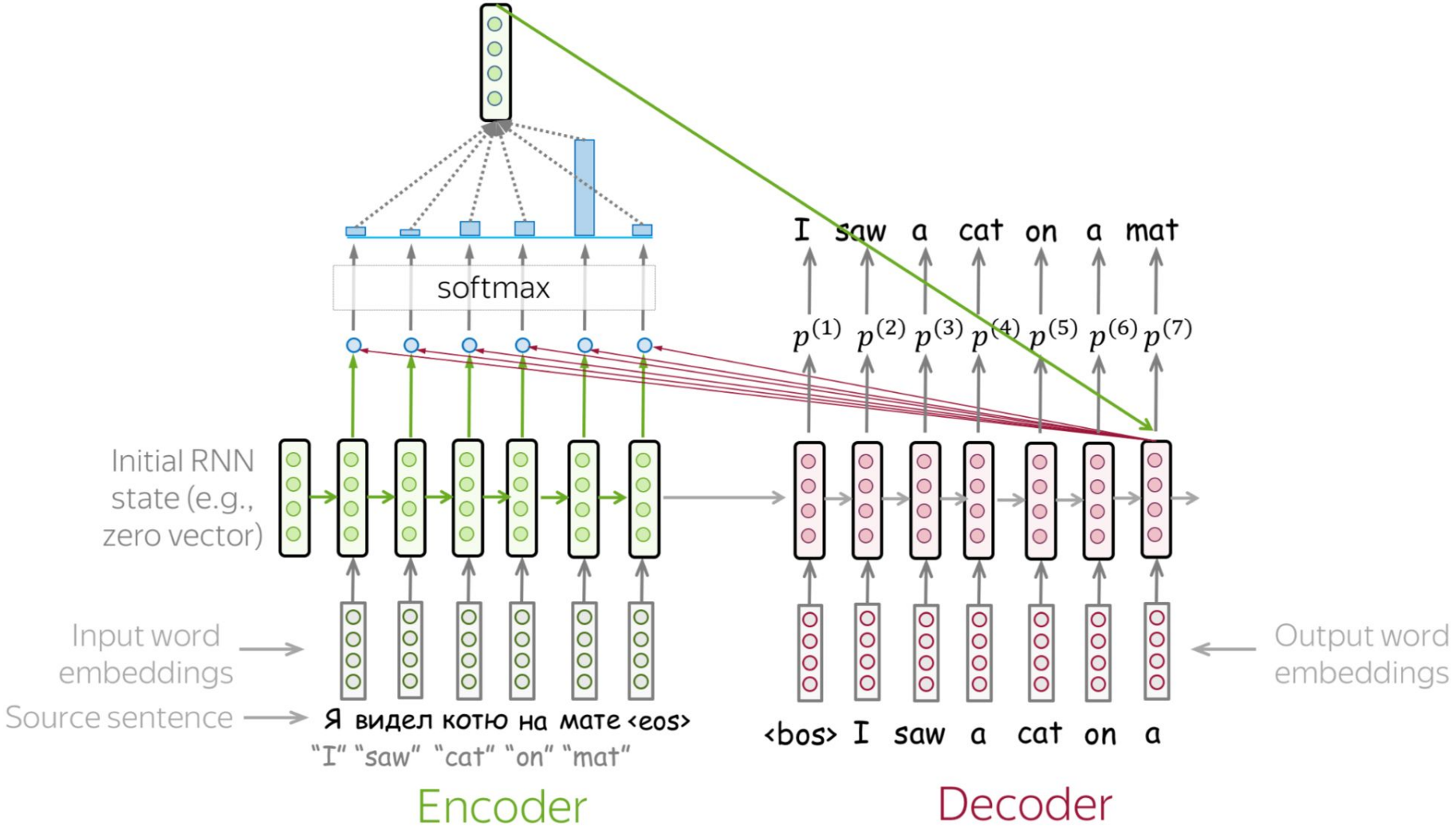

softmax
I saw
$p^{(1)}$ $p^{(2)}$
Initial RNN state (e.g., zero vector)
Input word embeddings
Source sentence
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
Output word embeddings
<bos> I saw
Encoder
Decoder

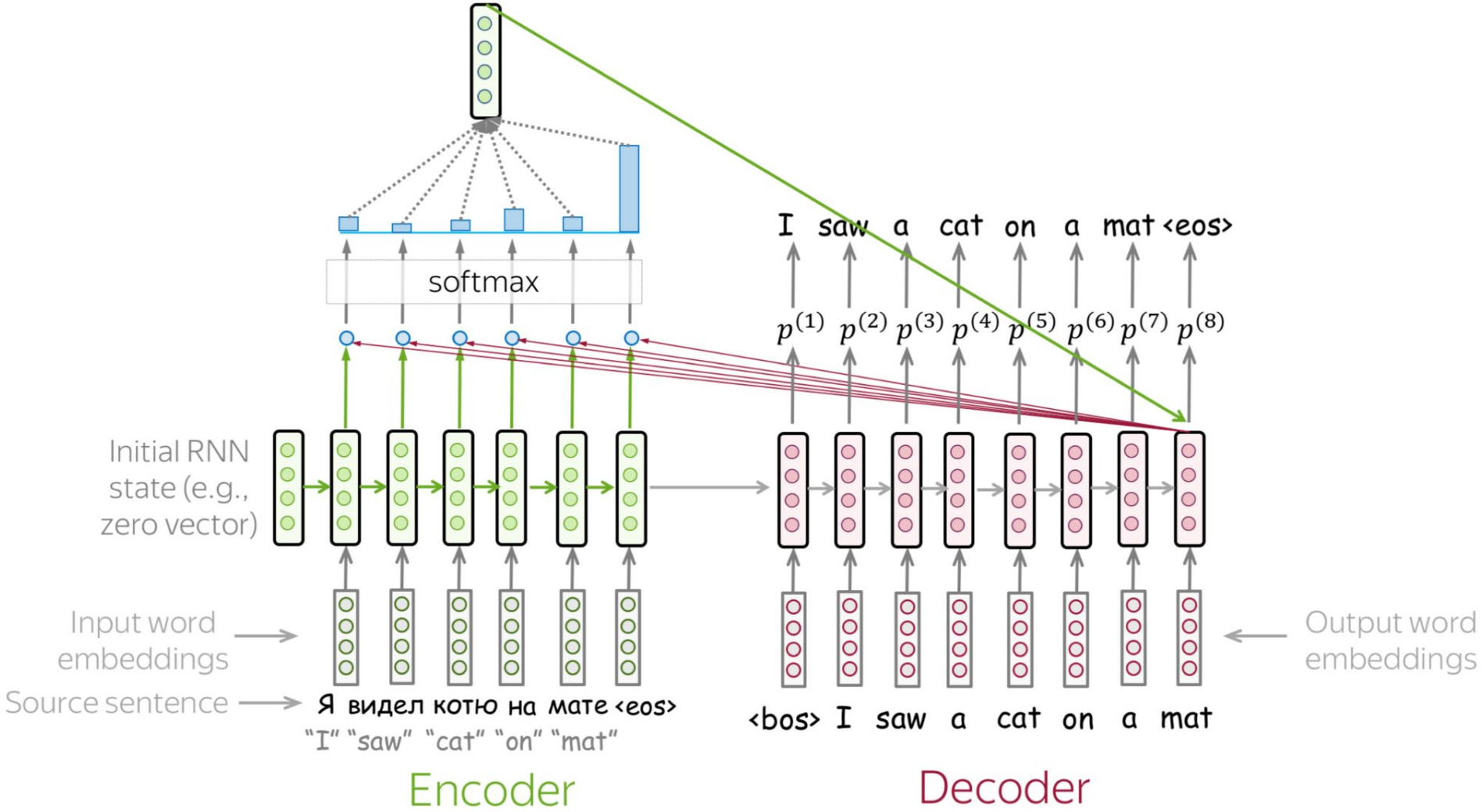

softmax
I saw a
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$
Initial RNN state (e.g., zero vector)
Input word embeddings
Source sentence
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
Output word embeddings
<bos> I saw
Encoder
Decoder


softmax
I saw a cat
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$ $p^{(4)}$
Initial RNN state (e.g., zero vector)
Input word embeddings
Source sentence
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
Output word embeddings
<bos> I saw a
Encoder
Decoder

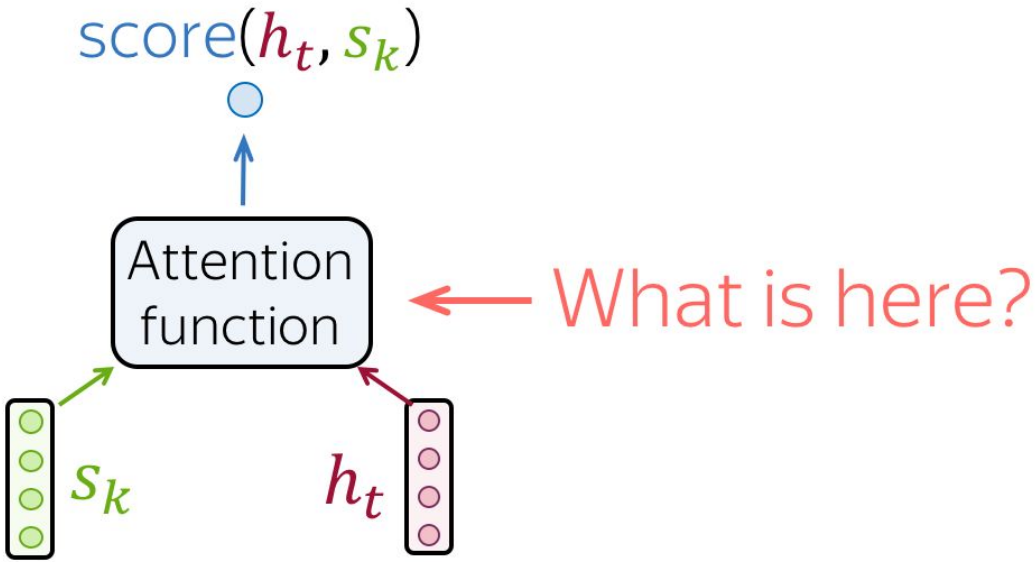

softmax
I saw a cat on
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$ $p^{(4)}$ $p^{(5)}$
Initial RNN state (e.g., zero vector)
Input word embeddings
Source sentence
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
Output word embeddings
<bos> I saw a cat
Encoder
Decoder


softmax
I saw a cat on a
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$ $p^{(4)}$ $p^{(5)}$ $p^{(6)}$
Initial RNN state (e.g., zero vector)
Input word embeddings
Source sentence
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
Output word embeddings
<bos> I saw a cat on
Encoder
Decoder


softmax
I saw a cat on a mat
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$ $p^{(4)}$ $p^{(5)}$ $p^{(6)}$ $p^{(7)}$
Initial RNN state (e.g., zero vector)
Input word embeddings
Source sentence
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
Output word embeddings
<bos> I saw a cat on a
Encoder
Decoder

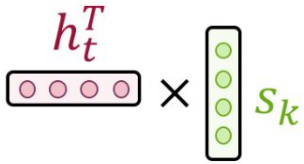

softmax
I saw a cat on a mat <eos>
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$ $p^{(4)}$ $p^{(5)}$ $p^{(6)}$ $p^{(7)}$ $p^{(8)}$
Initial RNN state (e.g., zero vector)
Input word embeddings
Source sentence
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
Output word embeddings
<bos> I saw a cat on a mat
Encoder
Decoder

А что за функция?



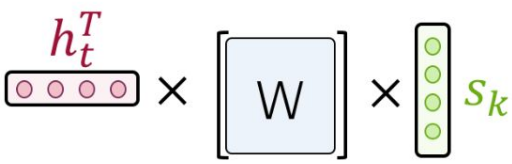
В целом любая

Dot-product



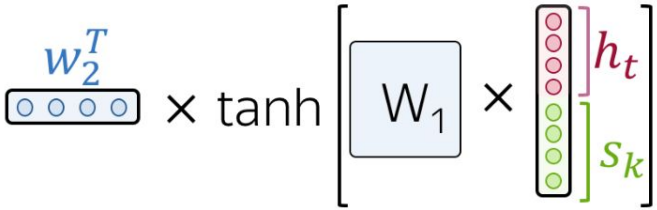
$$\text{score}(h_t, s_k) = h_t^T \; s_k$$

Bilinear



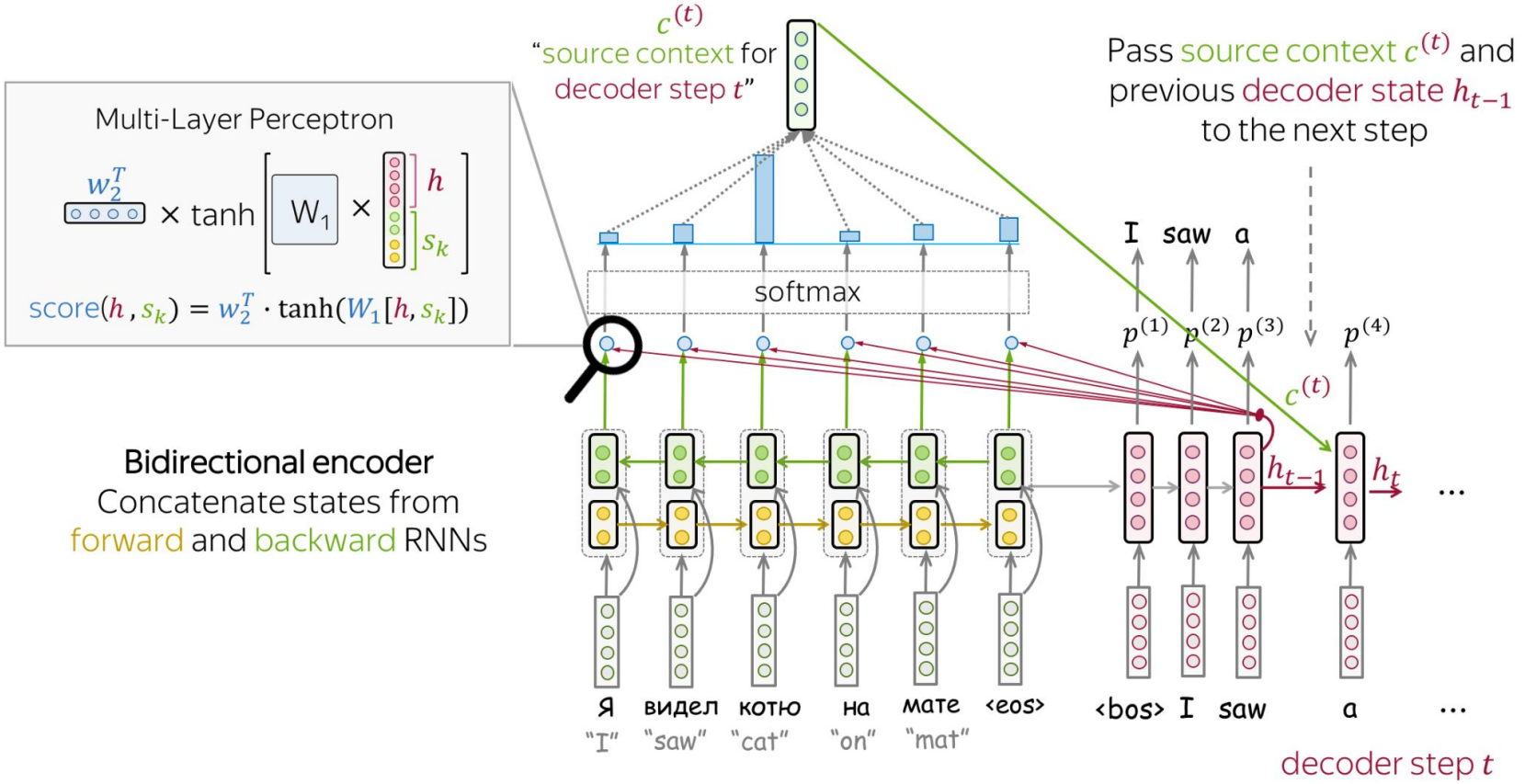
$$\text{score}(h_t, s_k) = h_t^T \; W s_k$$

Multi-Layer Perceptron

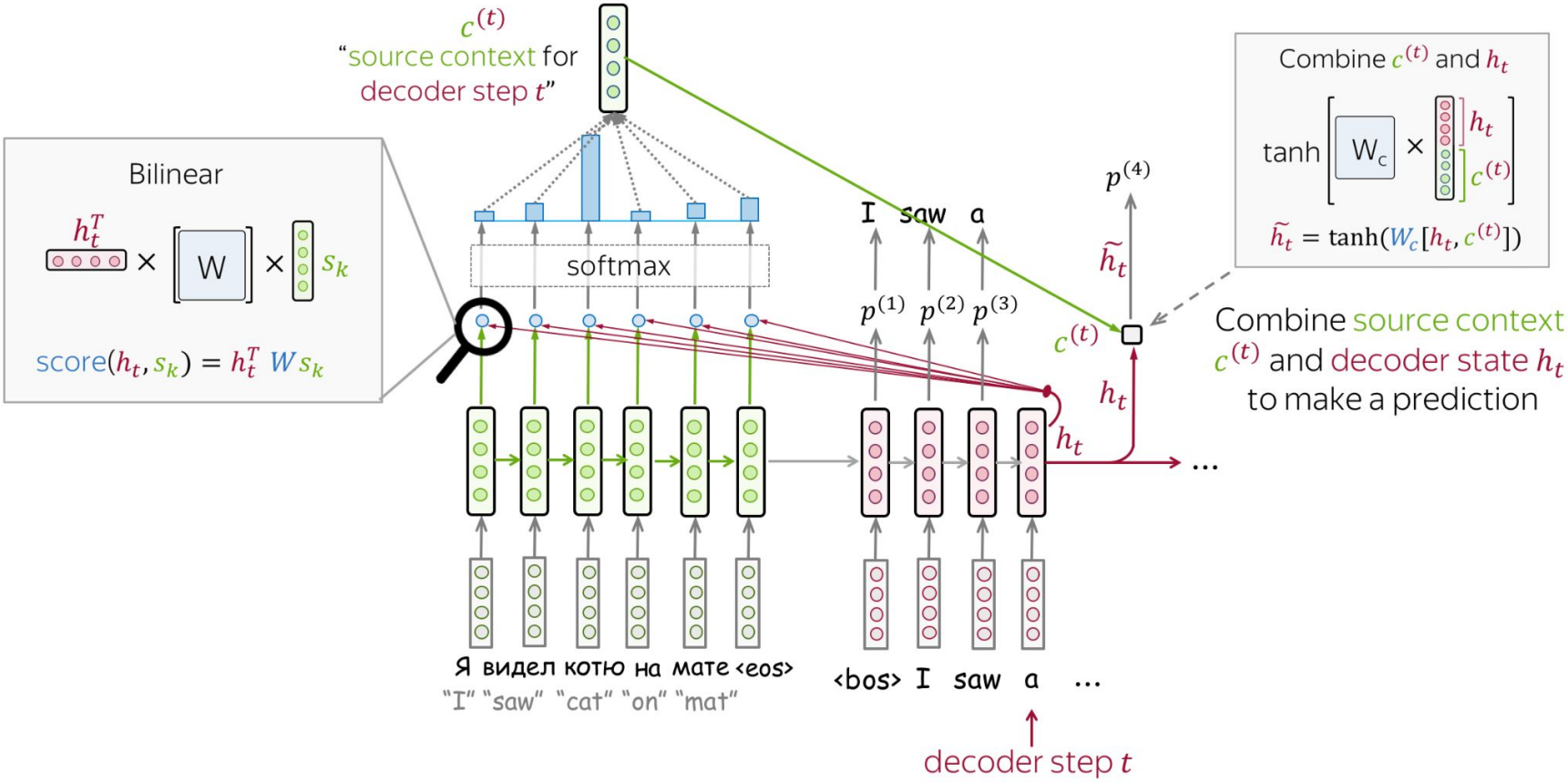


$$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1[h_t, s_k])$$

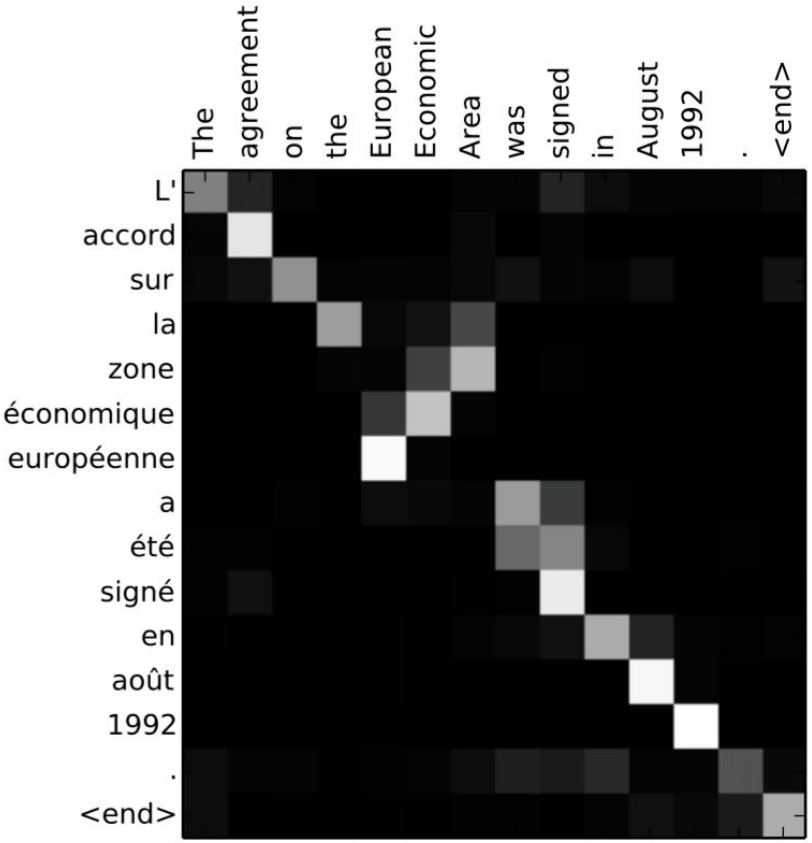
Bahdanau model

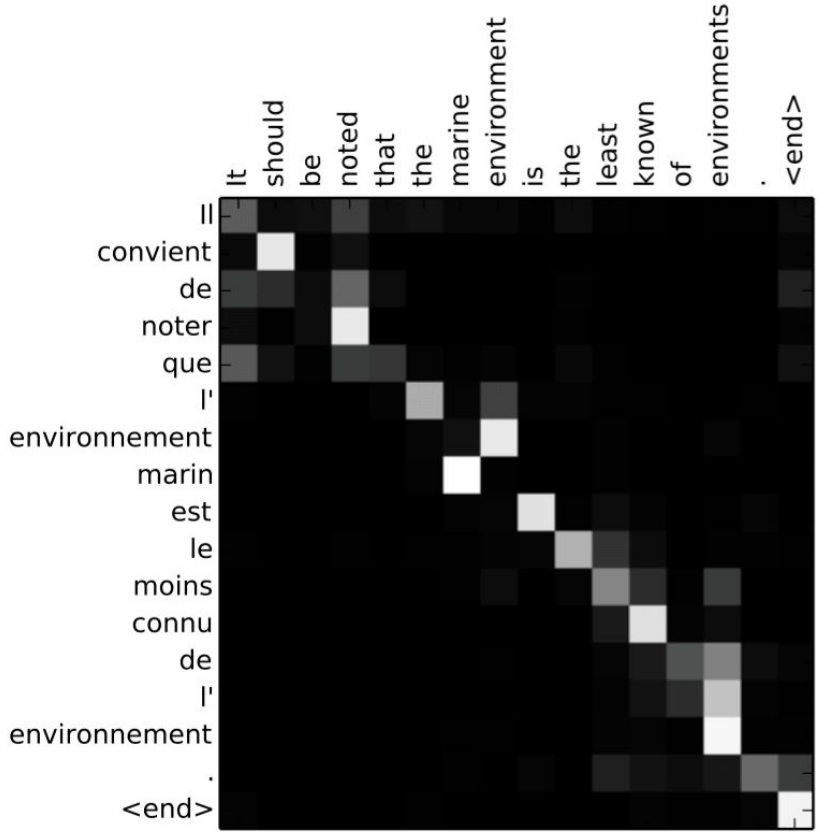


Luong model


$c^{(t)}$
"source context for decoder step t"
Bilinear
h_t^T × W × s_k
score$(h_t, s_k) = h_t^T W s_k$
softmax
Combine $c^{(t)}$ and h_t
tanh W_c × h_t $c^{(t)}$
$\tilde{h}_t = \tanh(W_c[h_t, c^{(t)}])$
Combine source context $c^{(t)}$ and decoder state h_t to make a prediction
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$ $p^{(4)}$
I saw a
$\tilde{h}_t$
$c^{(t)}$
h_t
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
<bos> I saw a ...
decoder step t

Alignment


The
agreement
on
the
European
Economic
Area
was
signed
in
August
1992
.
<end>
L'
accord
sur
la
zone
économique
européenne
a
été
signé
en
août
1992
.
<end>


It
should
be
noted
that
the
marine
environment
is
the
least
known
of
environments
.
<end>
Il
convient
de
noter
que
l'
environnement
marin
est
le
moins
connu
de
l'
environnement
.
<end>

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

Что хотим от трансформера?

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:

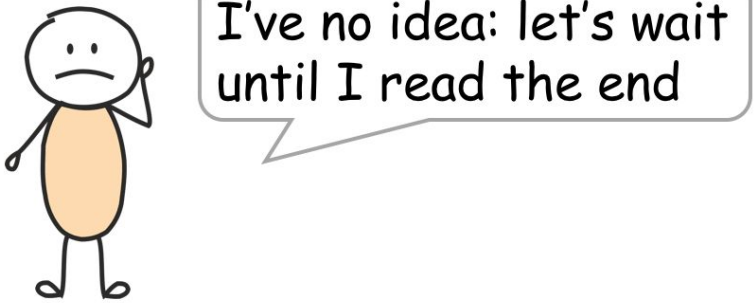
- looks at previous target tokens
- looks at source representations
- update representation

repeat N times

Почему RNN будет работать хуже?

I arrived at the bank after crossing the ... ...street? ...river?

What does **bank** mean in this sentence?

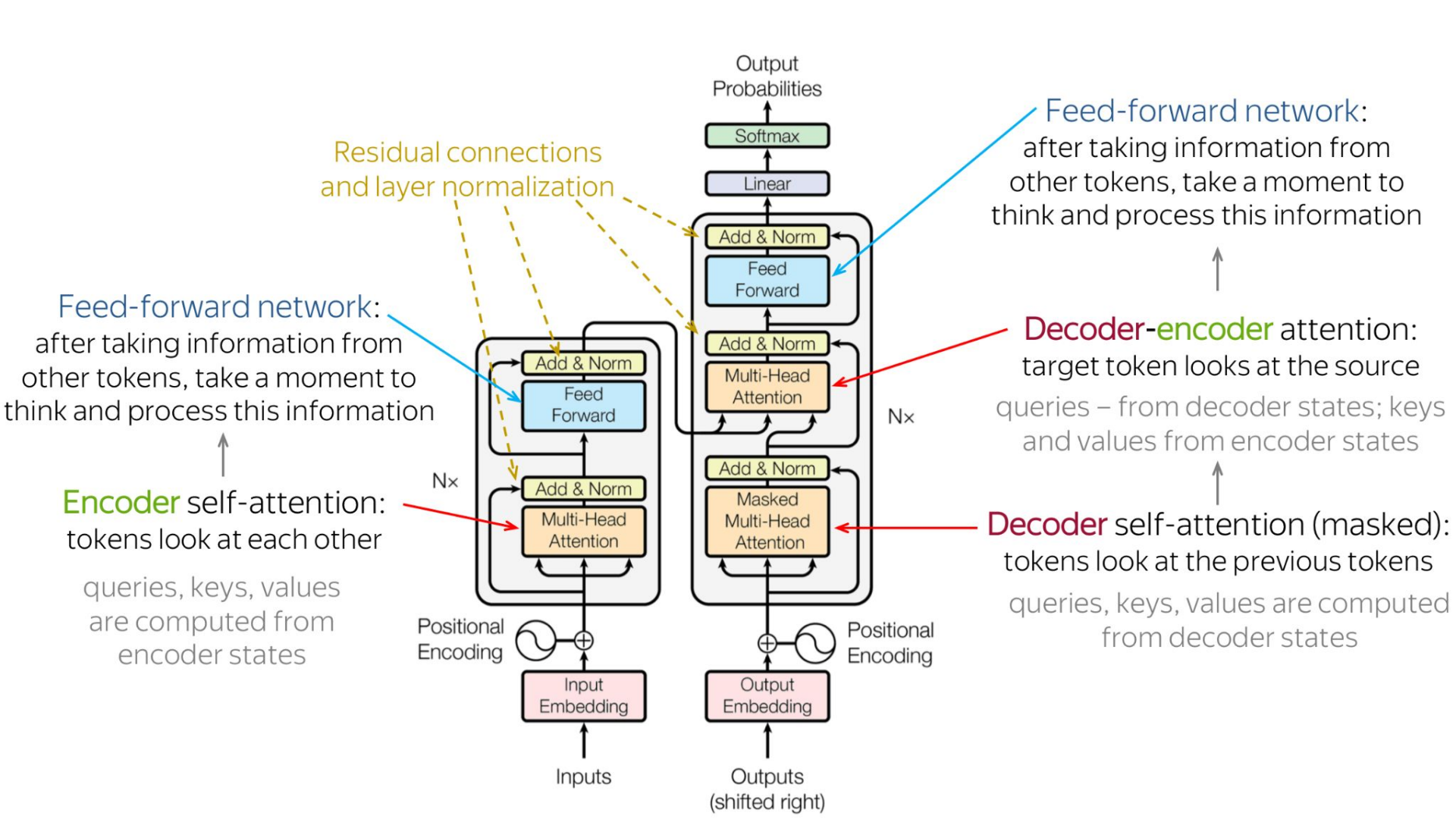


RNNs

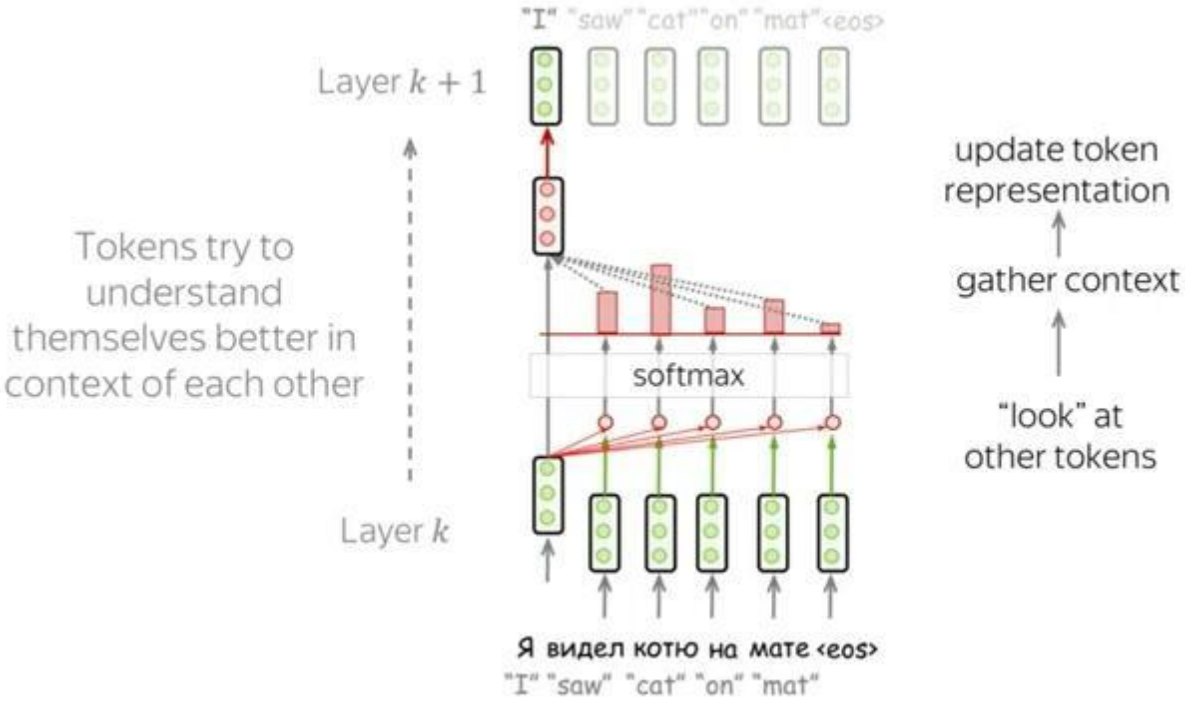
O(N) steps to process a

Transformer

Constant number of steps


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
Residual connections and layer normalization
Feed-forward network: after taking information from other tokens, take a moment to think and process this information
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
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

Self-attention



Self-attention

Each vector receives three representations ("roles")

$W_Q \times = $ Query: vector **from** which the attention is looking

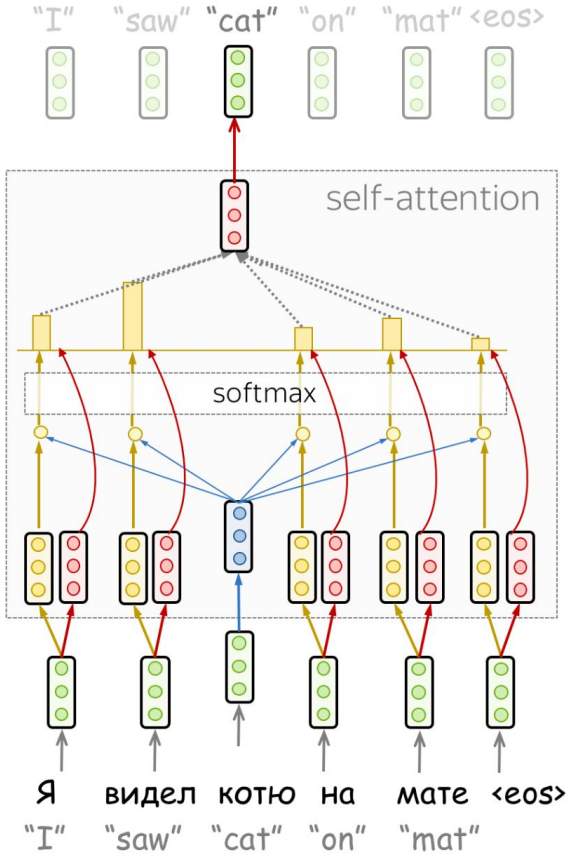
"Hey there, do you have this information?"

$W_K \times = $ Key: vector **at** which the query looks to compute weights

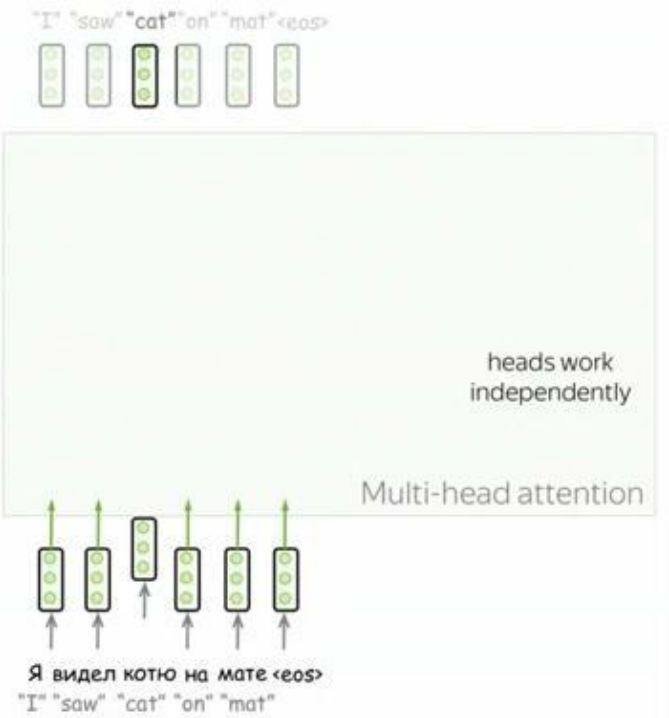
"Hi, I have this information – give me a large weight!"

$W_V \times = $ Value: their weighted sum is attention output

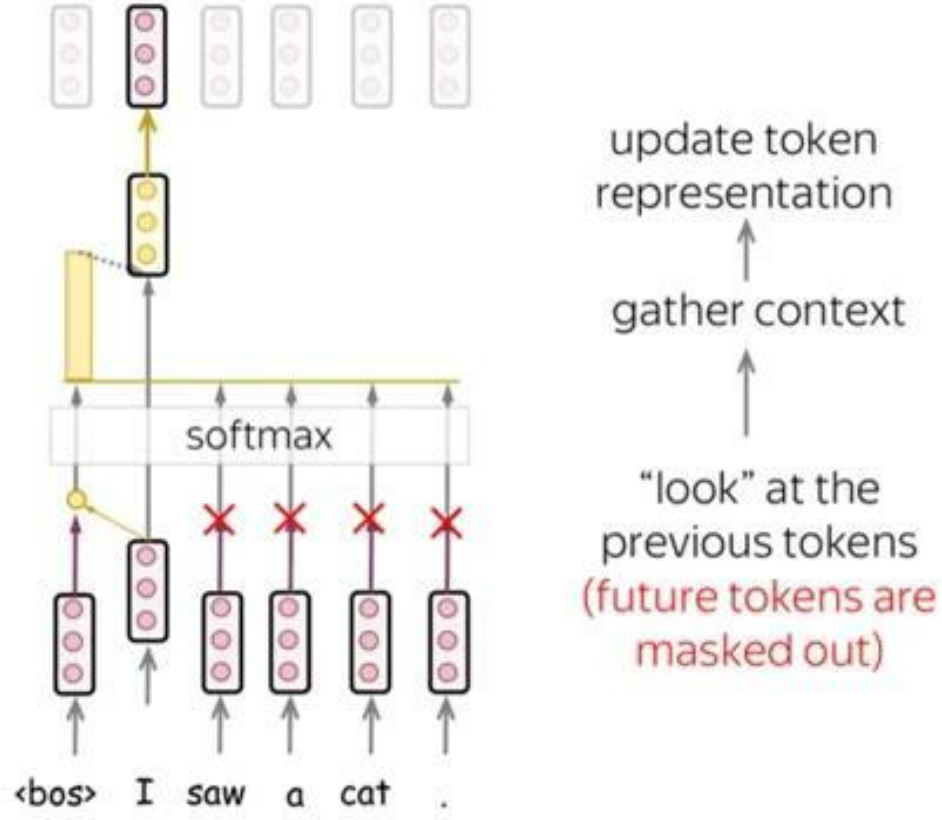
"Here's the information I have!"



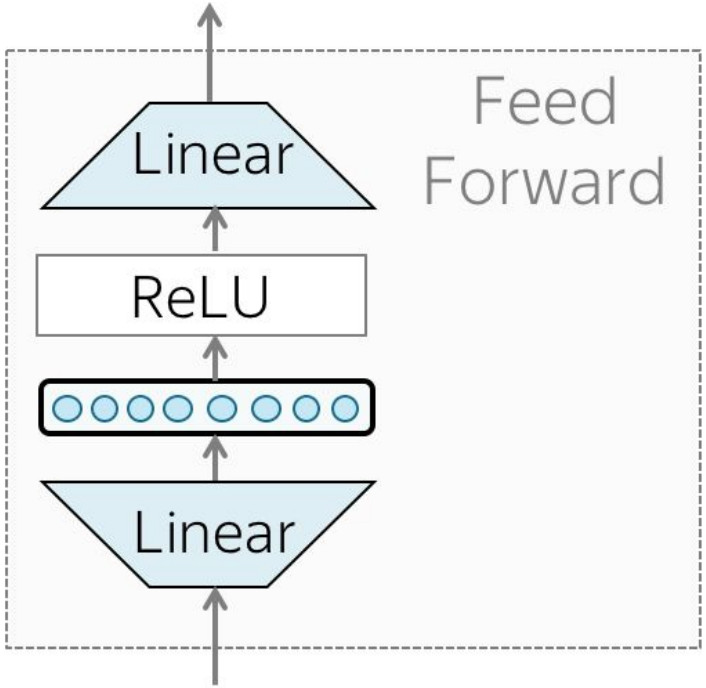
Multi-head attention



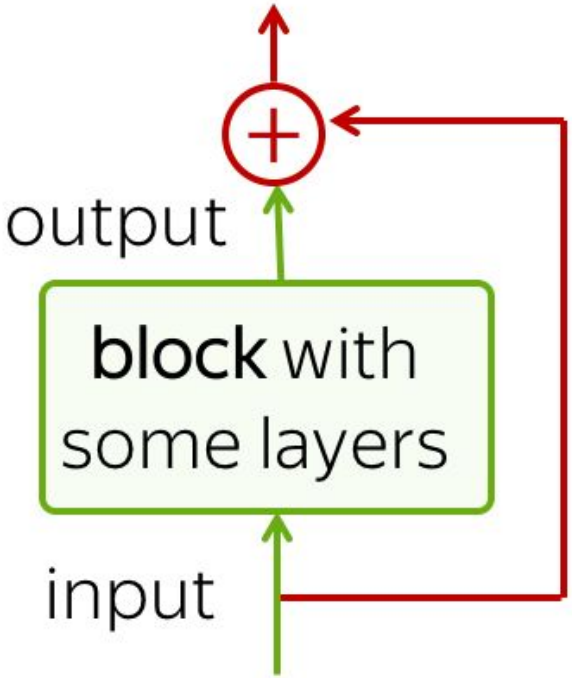
Masked attention



FFN

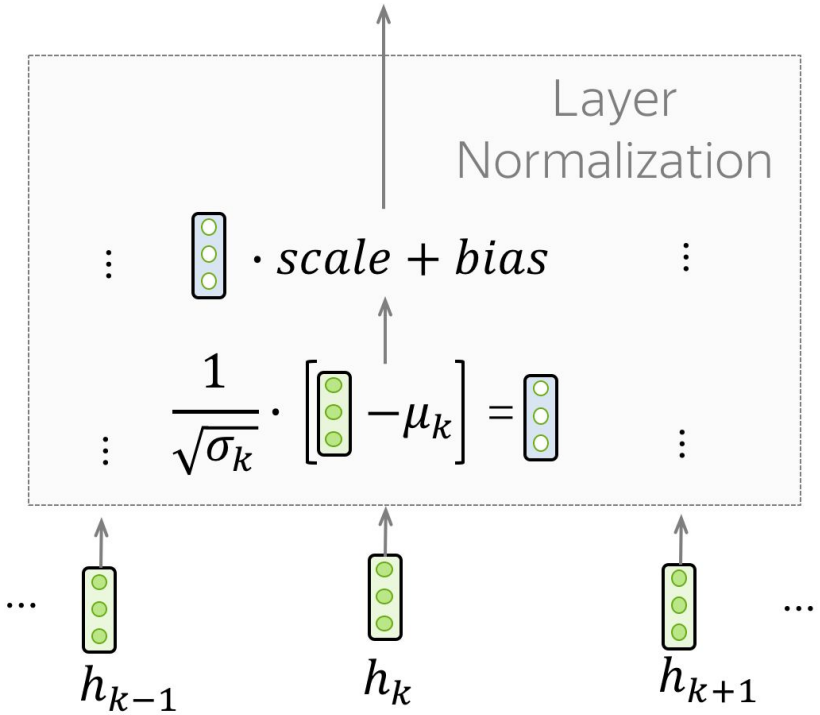


Residual connection

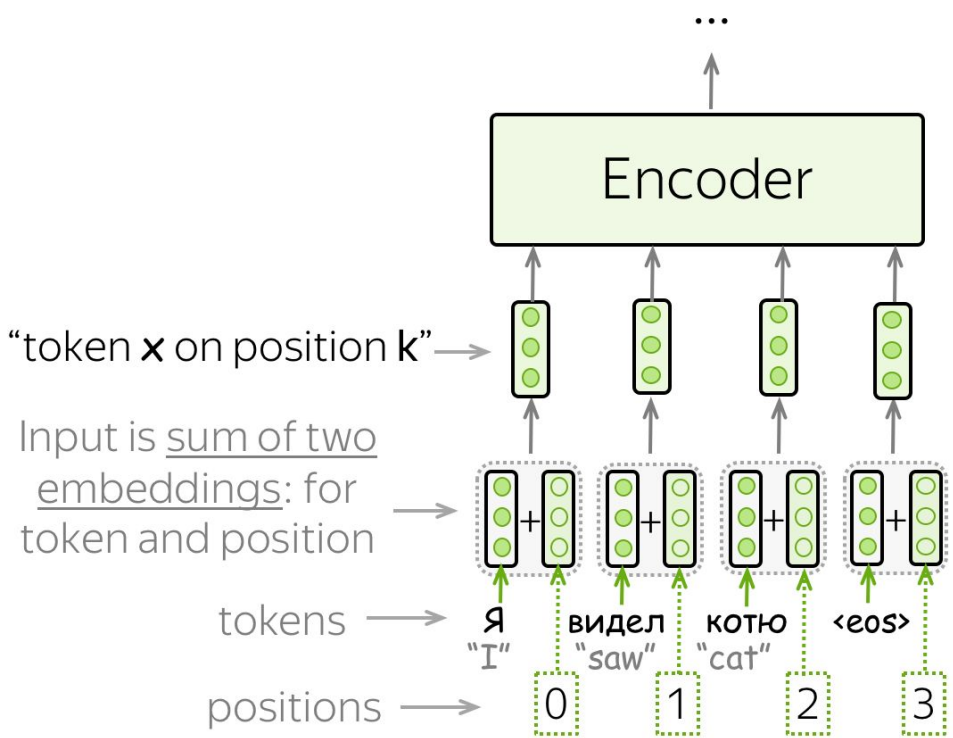


Residual connection: add a block's input to its output

Layer norm


Layer
Normalization
$\cdot\, scale + bias$
$\frac{1}{\sqrt{\sigma_k}} \cdot [\quad -\mu_k] =$
h_{k-1}
h_k
h_{k+1}

Positional encoding



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

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

BPE (byte-pair encoding)

Initial vocabulary:
characters

↓

Split each word
into characters

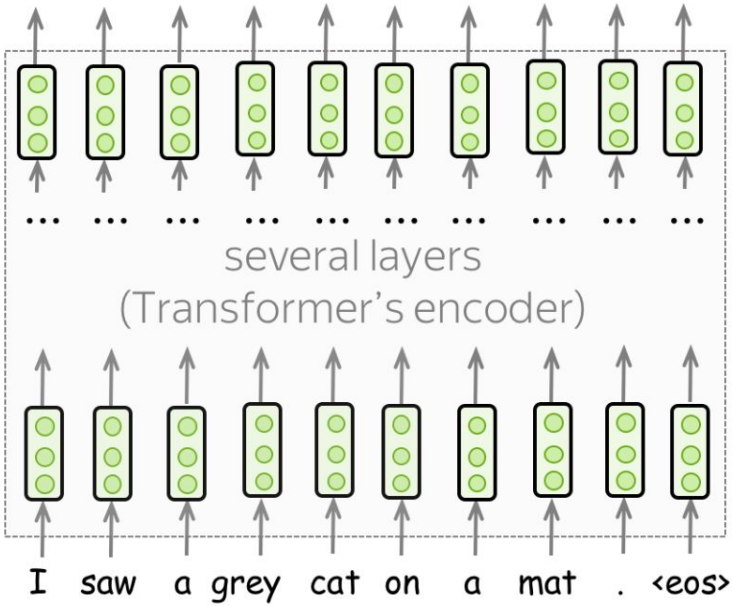
Words in the data:

word	count
c a t	4
m a t	5
m a t s	2
m a t e	3
a t e	3
e a t	2

Current merge table:

(empty)

BERT



Model architecture:

- Transformer's encoder

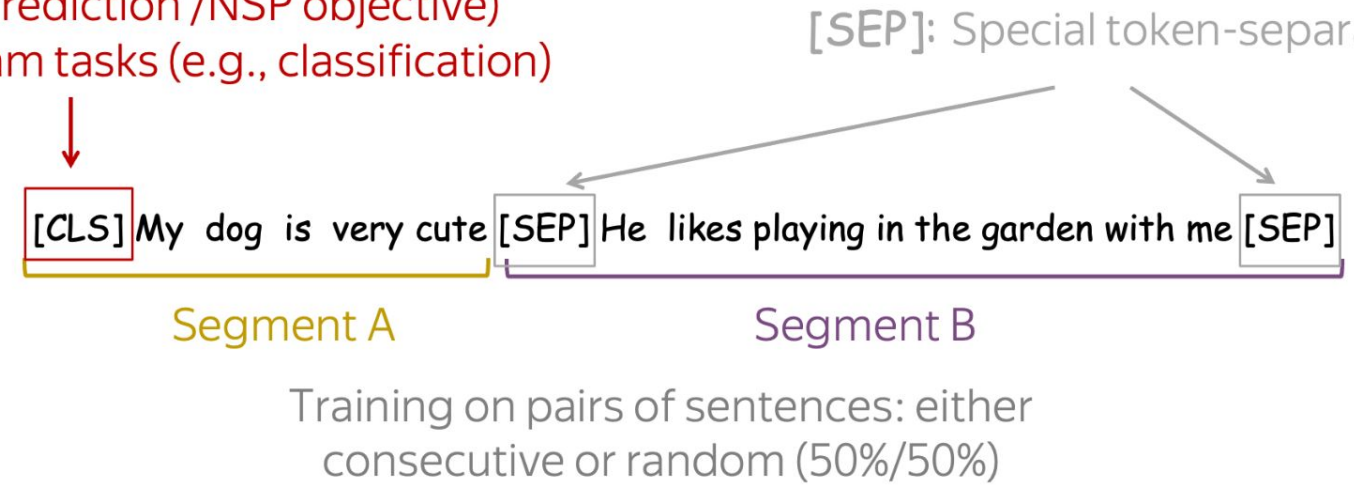
What is special about it:

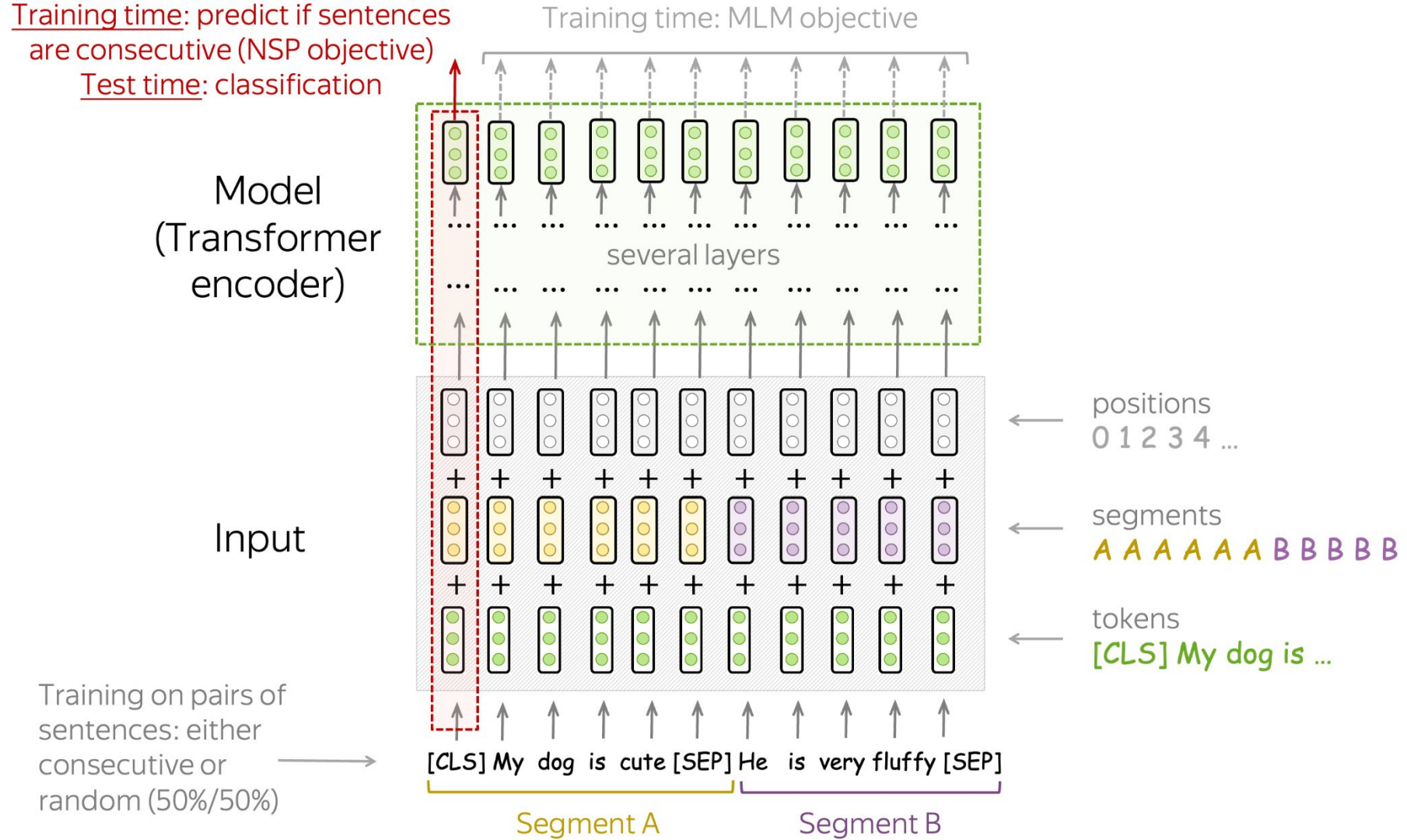
- Training objectives
  - MLM: Masked language modeling
  - NSP: Next sentence prediction
- The way it is used
  - No task-specific models

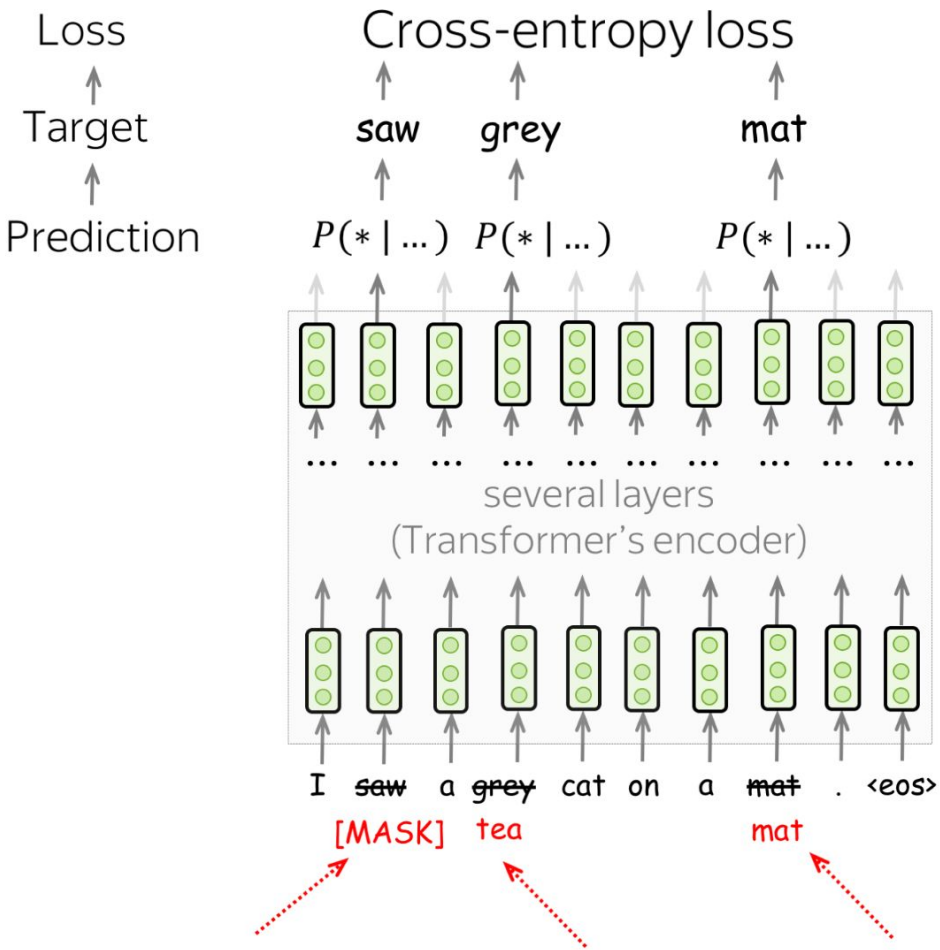
Input

[CLS]: Special token

- Training time: predict if sentences are consecutive or not (Next Sentence Prediction /NSP objective)
- Test time: downstream tasks (e.g., classification)




Training time: predict if sentences are consecutive (NSP objective)
Test time: classification
Training time: MLM objective
Model (Transformer encoder)
several layers
Input
positions
0 1 2 3 4 ...
segments
A A A A A A B B B B B
tokens
[CLS] My dog is ...
Training on pairs of sentences: either consecutive or random (50%/50%)
[CLS] My dog is cute [SEP] He is very fluffy [SEP]
Segment A
Segment B



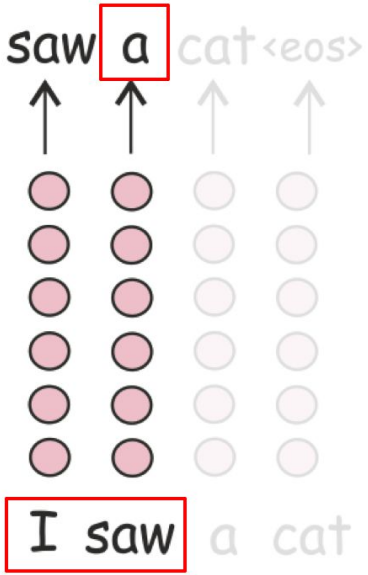
At each training step:

- pick randomly 15% of tokens
- replace each of the chosen tokens with something
- predict original chosen tokens

- **[MASK]**, with p = 80%
- Random token, with p = 10%
- Original token, with p = 10%

Language Modeling

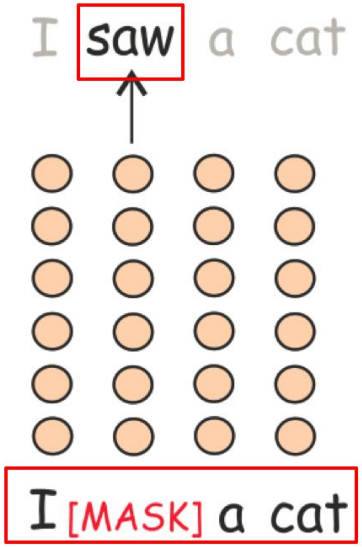
- Target: next token
- Prediction: $P(* \mid \text{I saw})$



left-to-right, does not see future

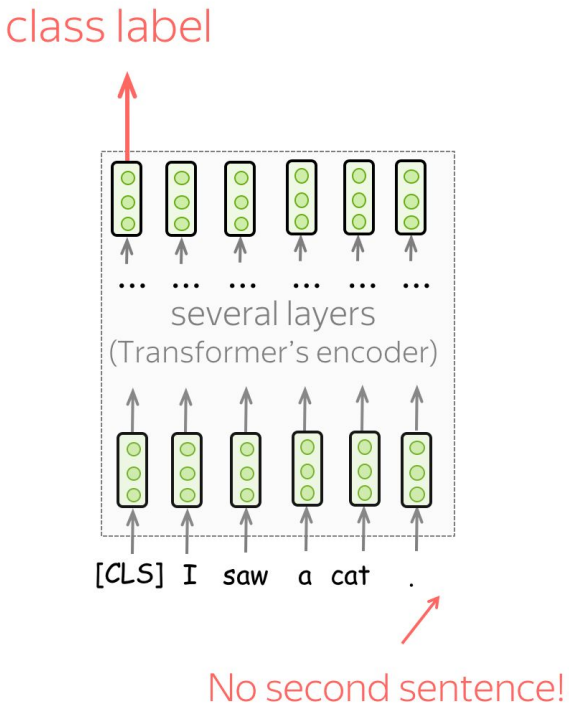
Masked Language Modeling

- Target: current token (the true one)
- Prediction: $P(* \mid \text{I [MASK] a cat})$

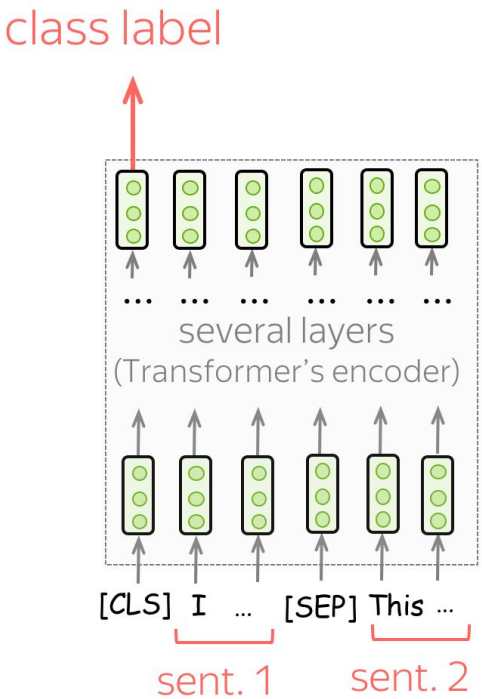


sees the whole text, but something is corrupted

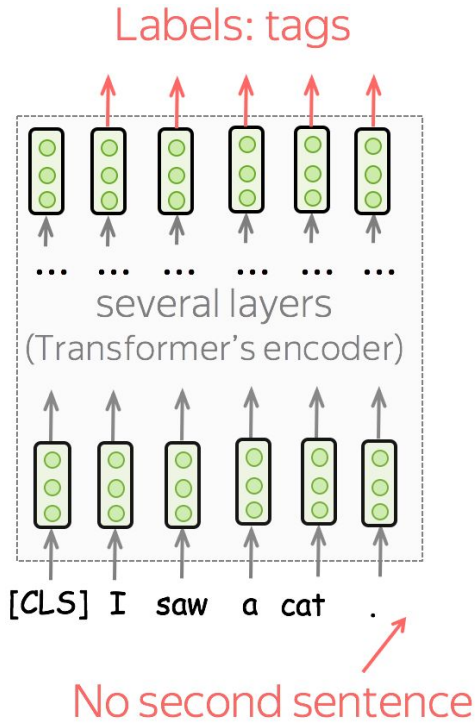
Sequence classification



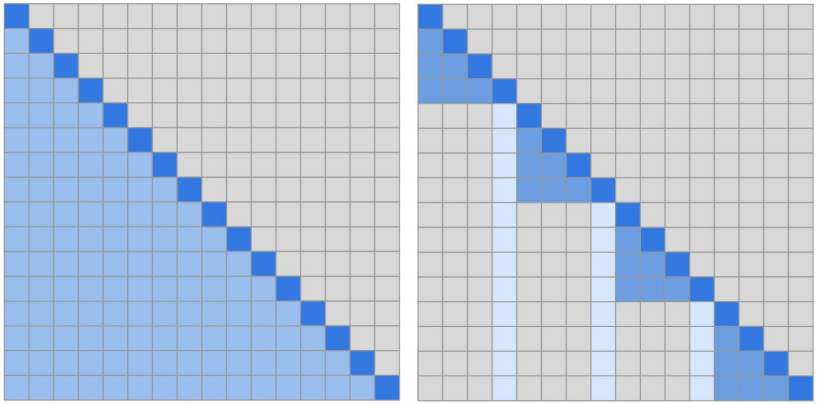
Relevance (e.g.)

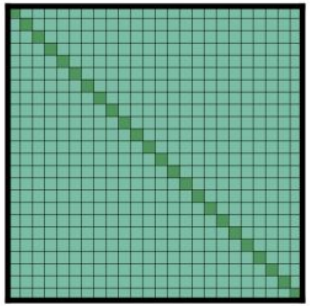


NER (named entity recognition)

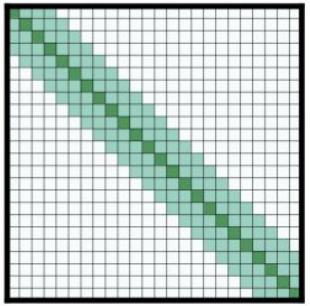


Разные маски attention

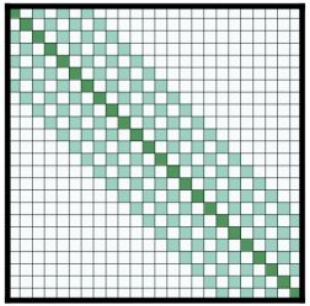




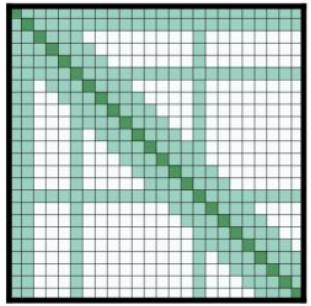
(a) Full n^2 attention



(b) Sliding window attention

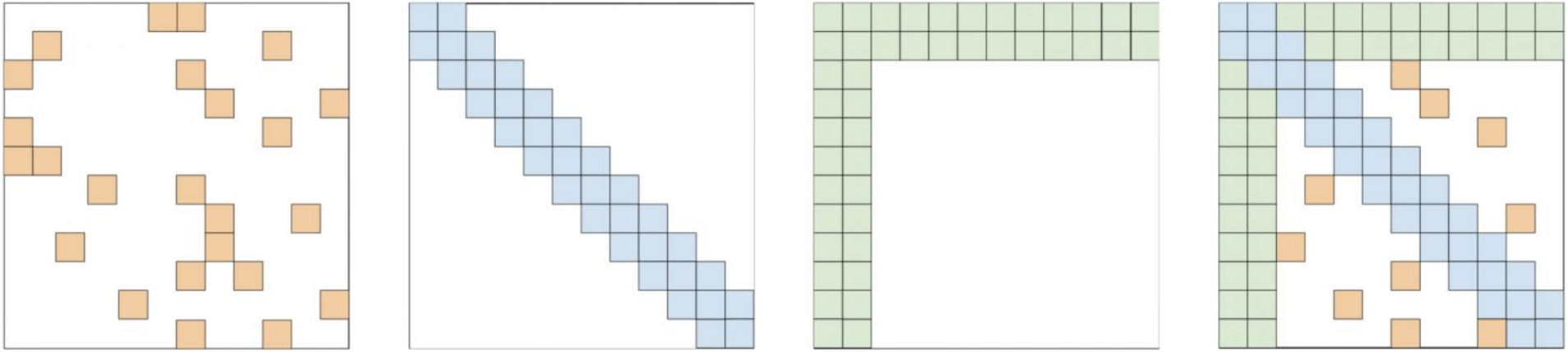


(c) Dilated sliding window



(d) Global+sliding window

Разные маски attention



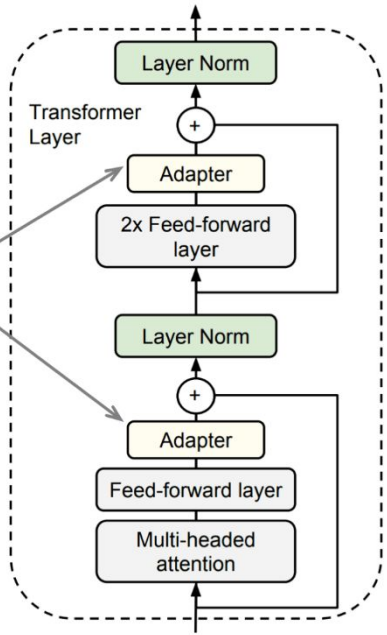
(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

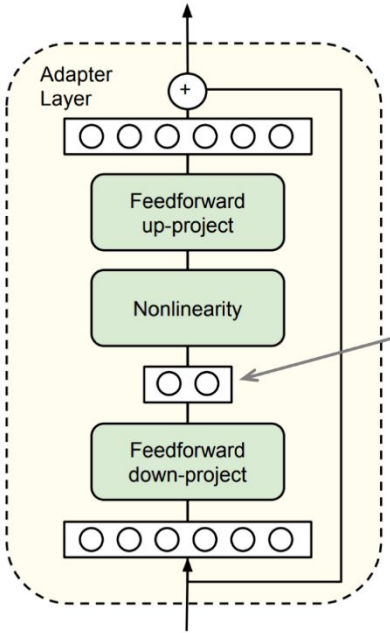
Модификации подсчёта внимания

$$\mathrm{softmax}(Q_{n\times d}K^T_{n\times d})V_{n\times d} \approx Q'_{n\times r}(K'^T_{n\times r}V_{n\times d})$$

Адаптеры

Only these are trained, everything else is fixed and is the same for all tasks





Small hidden size, i.e. an adaptor has only a few parameters (which is good!)

LoRA (Low Rank Adaptation)