Transformers

Thang Vu

15.01.2026

Outline

- Attention – A Recap
- Transformers
- BERT

Introduction

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

$$y' = \arg\max_y \overset{\text{model}}{p}(y|x, \overset{\text{parameters}}{\theta})$$

Questions we need to answer

- modeling

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

- learning

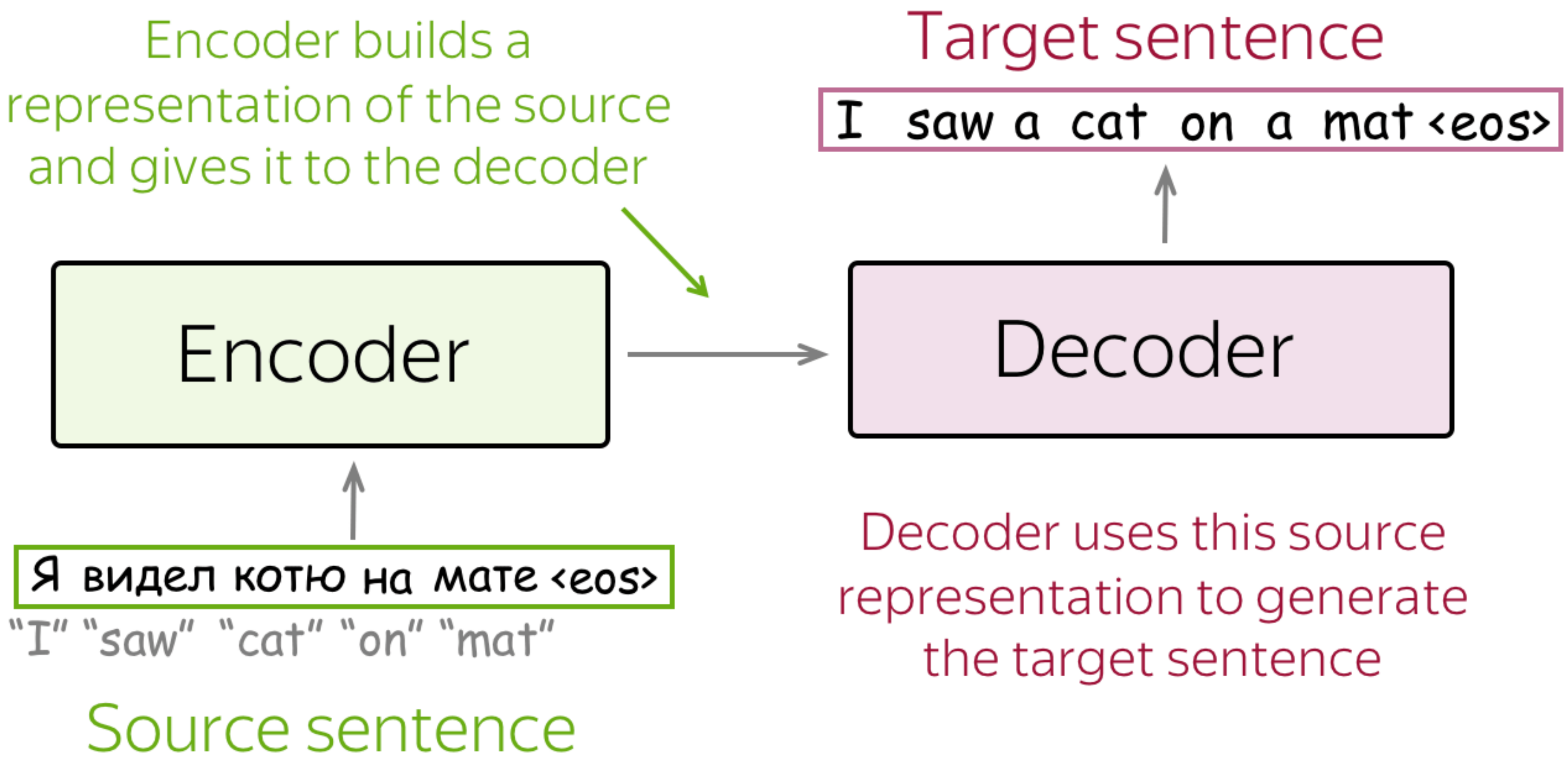
  How to find θ?

- search

  How to find the argmax?

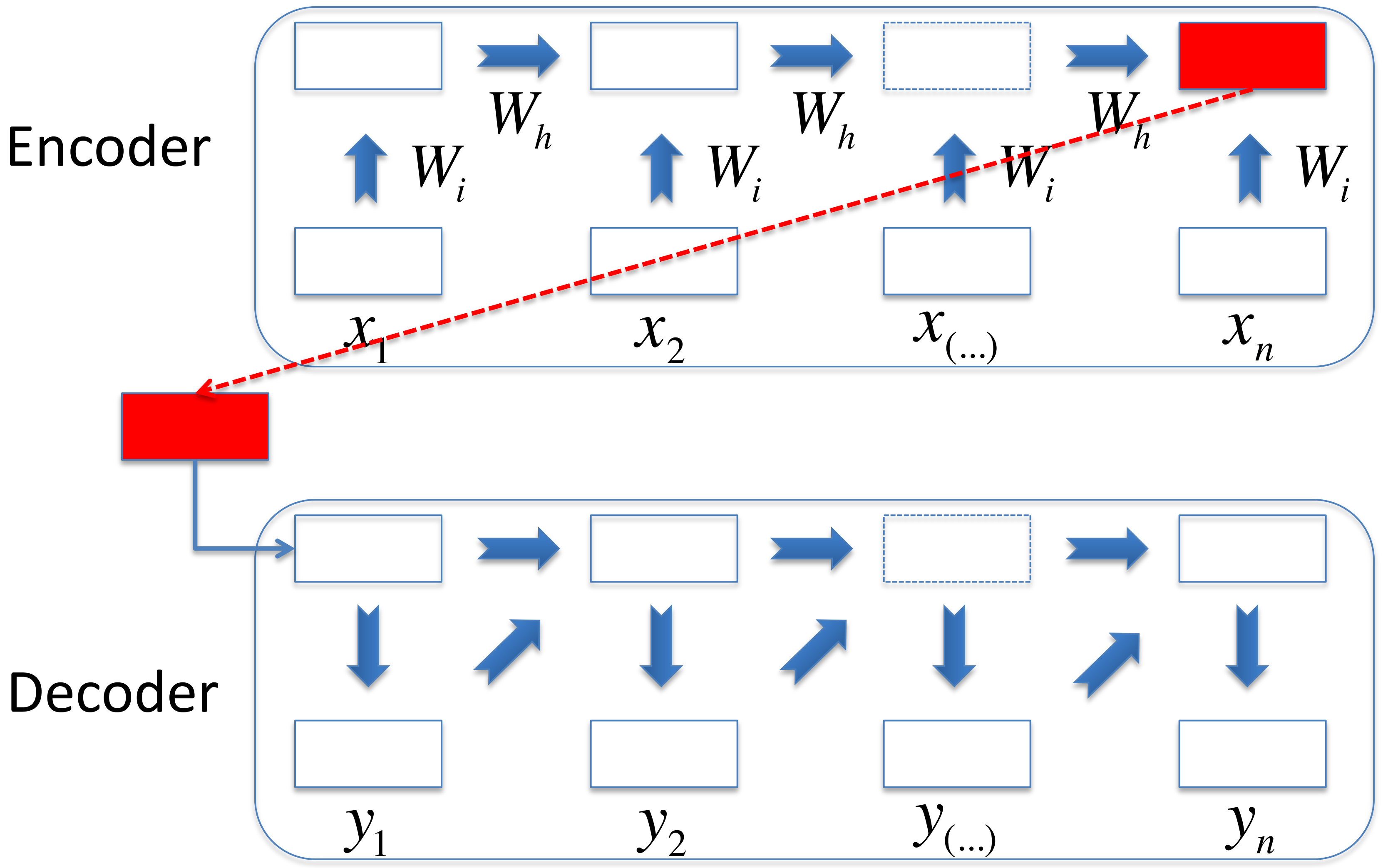
https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Introduction

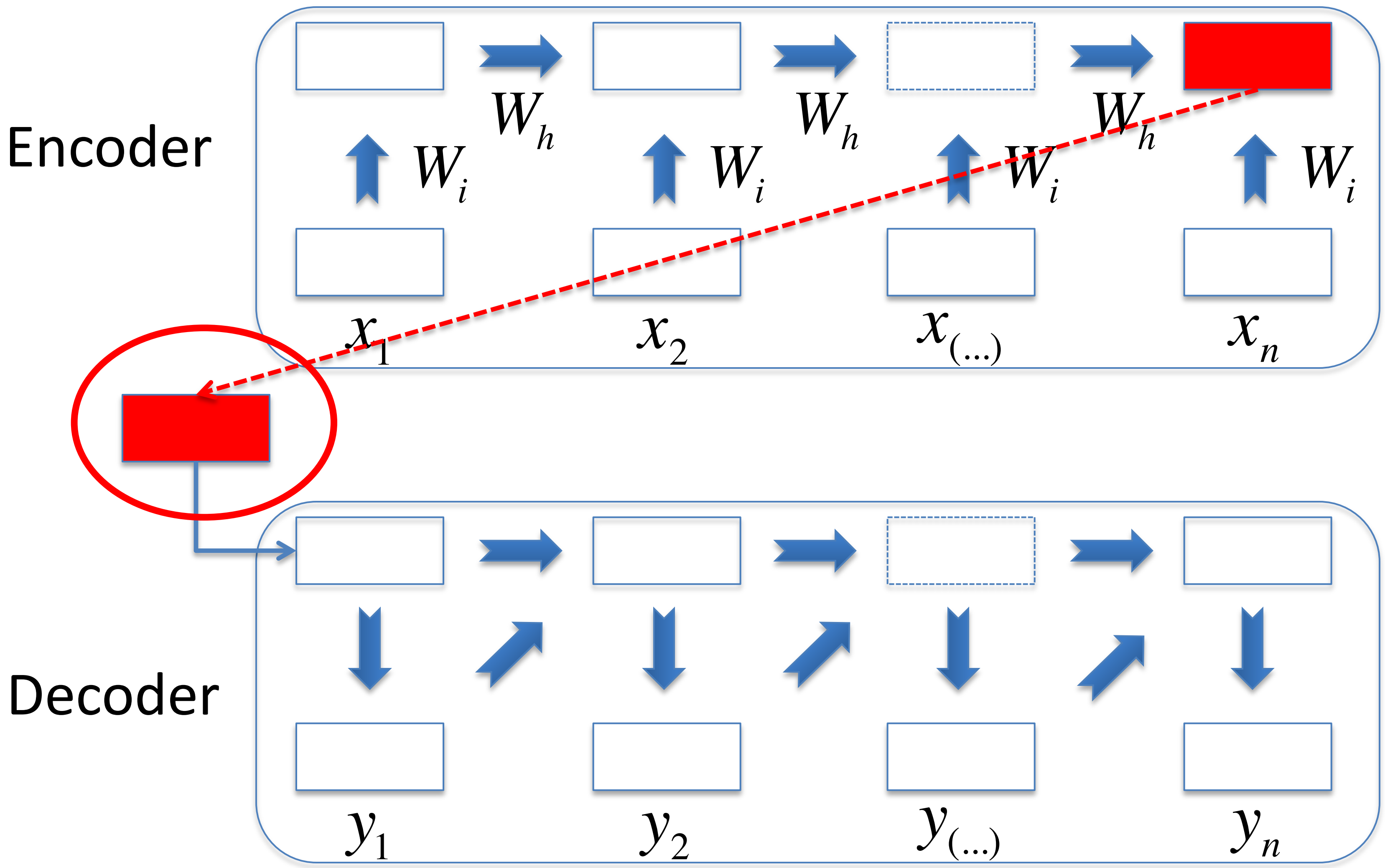

Encoder builds a representation of the source and gives it to the decoder
Target sentence
I saw a cat on a mat <eos>
Encoder
Decoder
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
Source sentence
Decoder uses this source representation to generate the target sentence

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

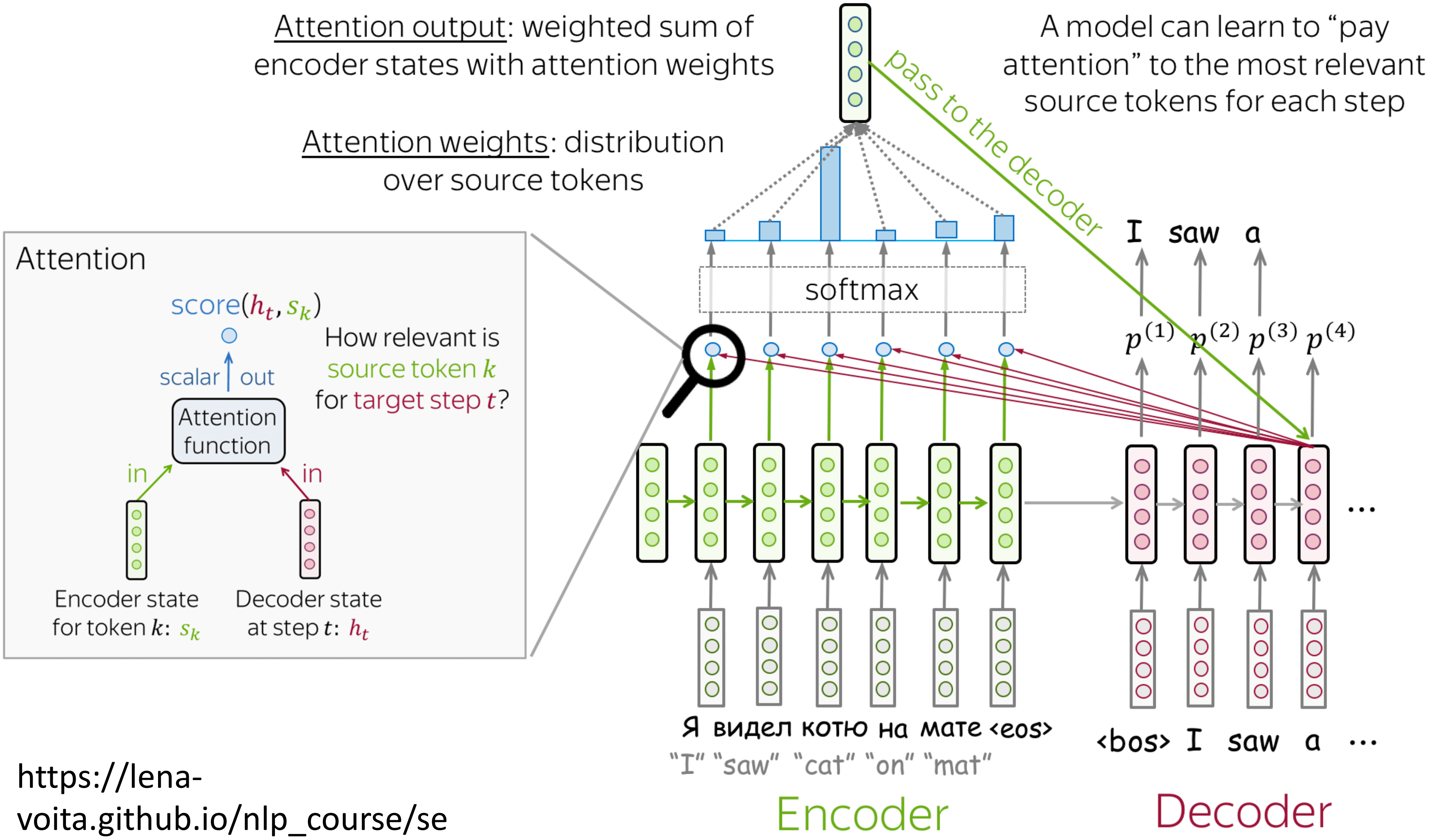
Introduction



Introduction



Attention Mechanism


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

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

Computational Process

Attention output

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

$\uparrow$ "source context for decoder step t"

$\uparrow$ (weighted sum)

Attention weights

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

$\uparrow$ "attention weight for source token k at decoder step t"

$\uparrow$ (softmax)

Attention scores

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

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

$\uparrow$

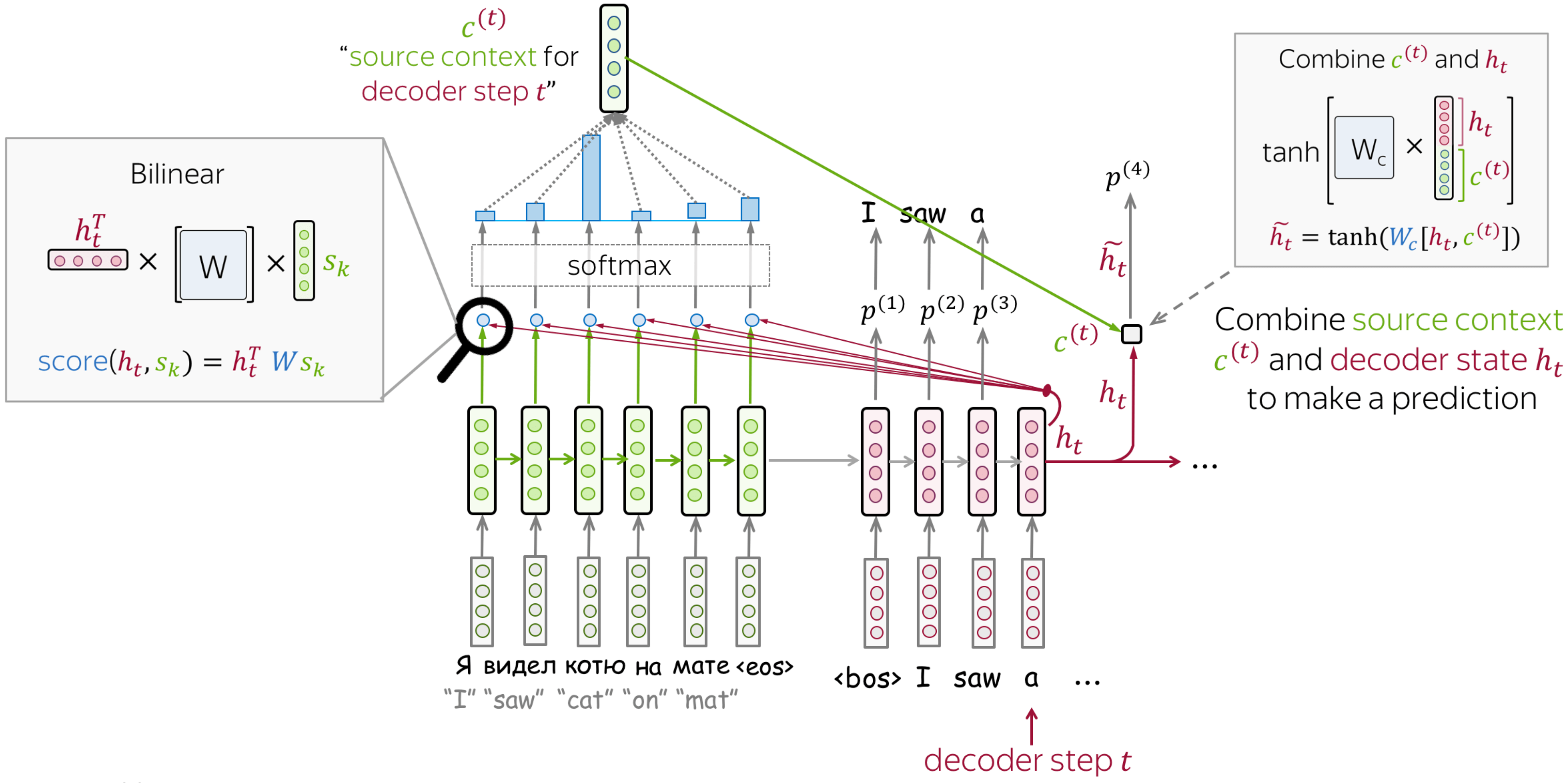
Attention input

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

h_t — one decoder state

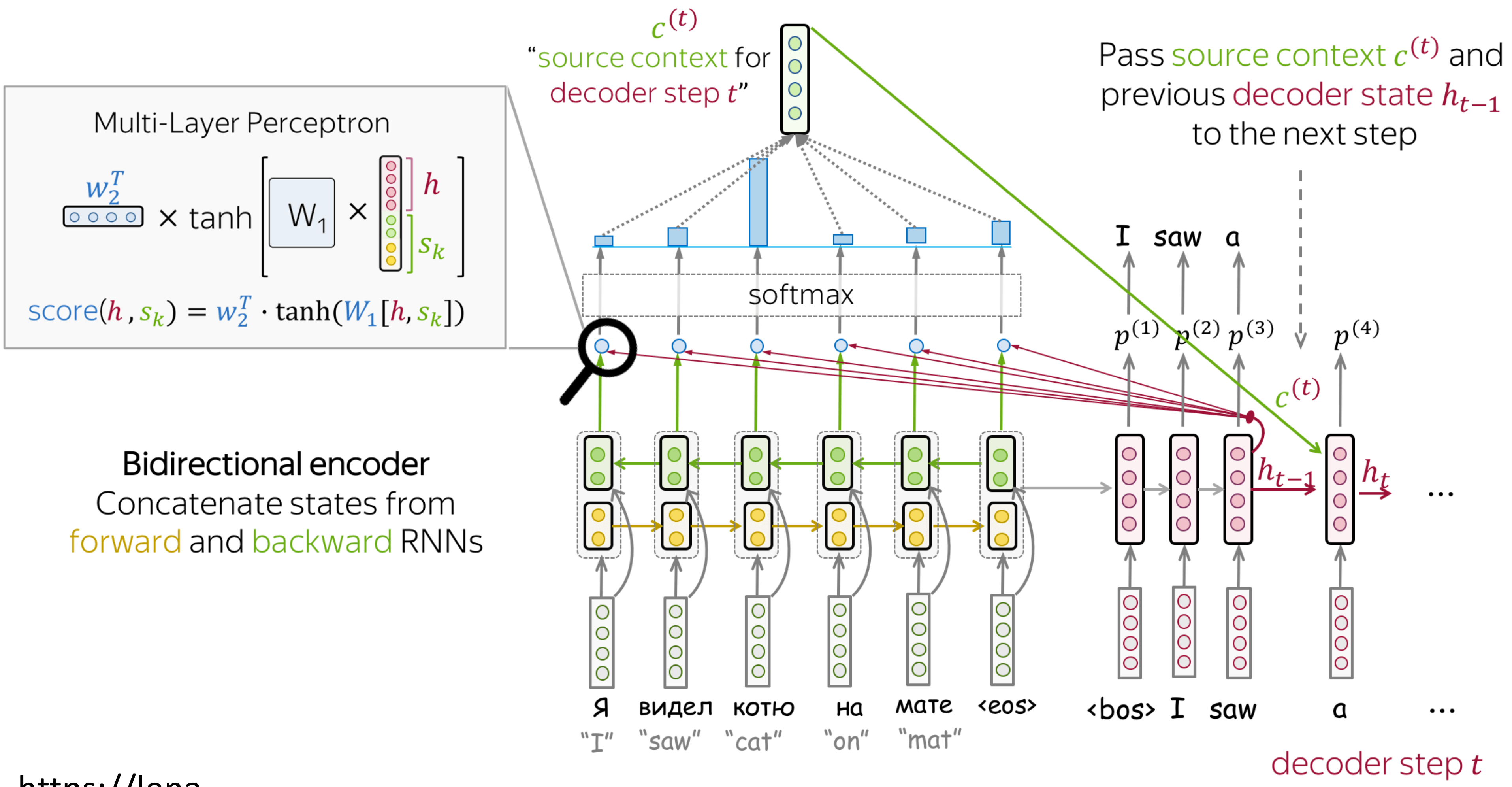
https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Luong et al 2015


$c^{(t)}$
"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
I saw a
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$
$p^{(4)}$
$\tilde{h}_t$
$c^{(t)}$
h_t
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
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
<bos> I saw a …
decoder step t

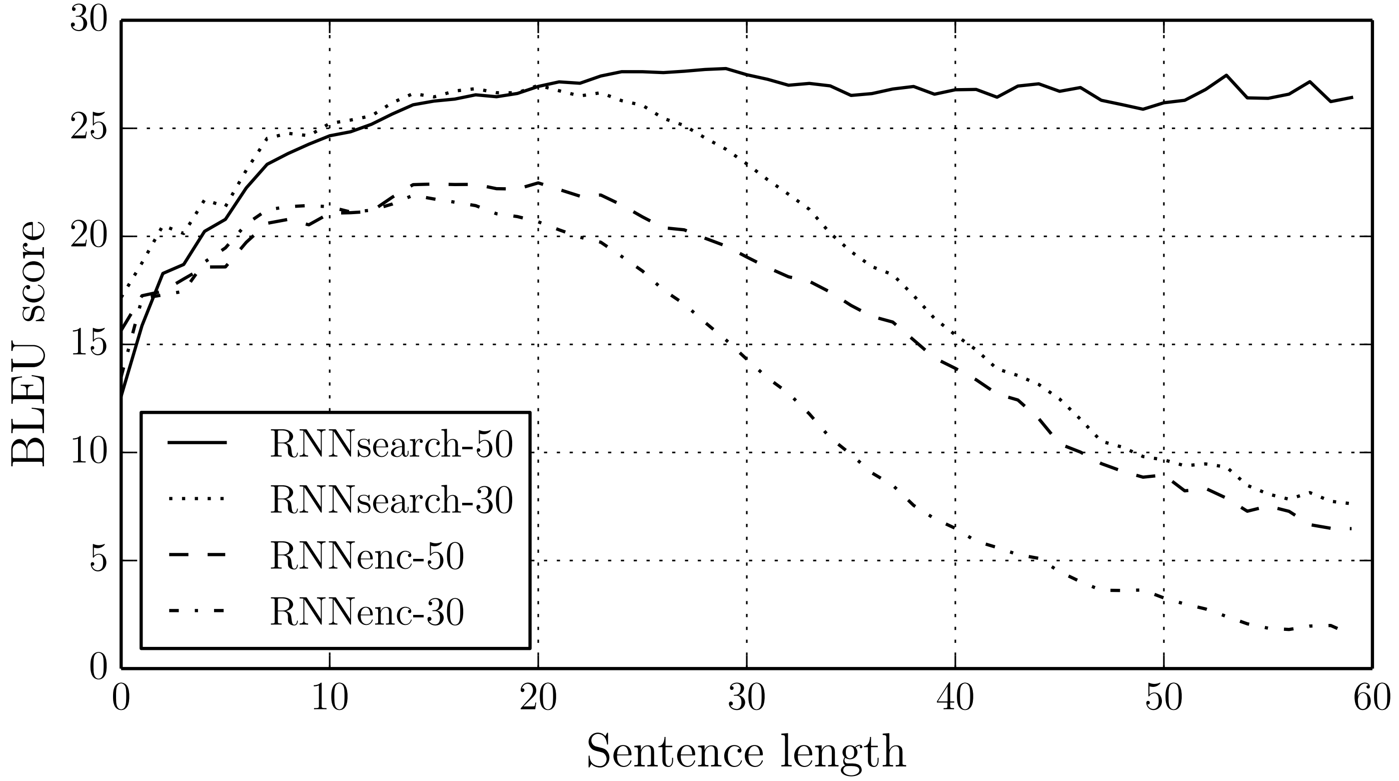
https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Bahdanau et al 2014 (2015)


Multi-Layer Perceptron
w_2^T
× tanh
W₁
×
h
s_k
$\mathrm{score}(h, s_k) = w_2^T \cdot \tanh(W_1[h, s_k])$
$c^{(t)}$
"source context for decoder step t"
Pass source context $c^{(t)}$ and previous decoder state h_{t-1} to the next step
softmax
I saw a
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$ $p^{(4)}$
$c^{(t)}$
h_{t-1}
h_t
...
Bidirectional encoder
Concatenate states from forward and backward RNNs
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
<bos> I saw a ...
decoder step t

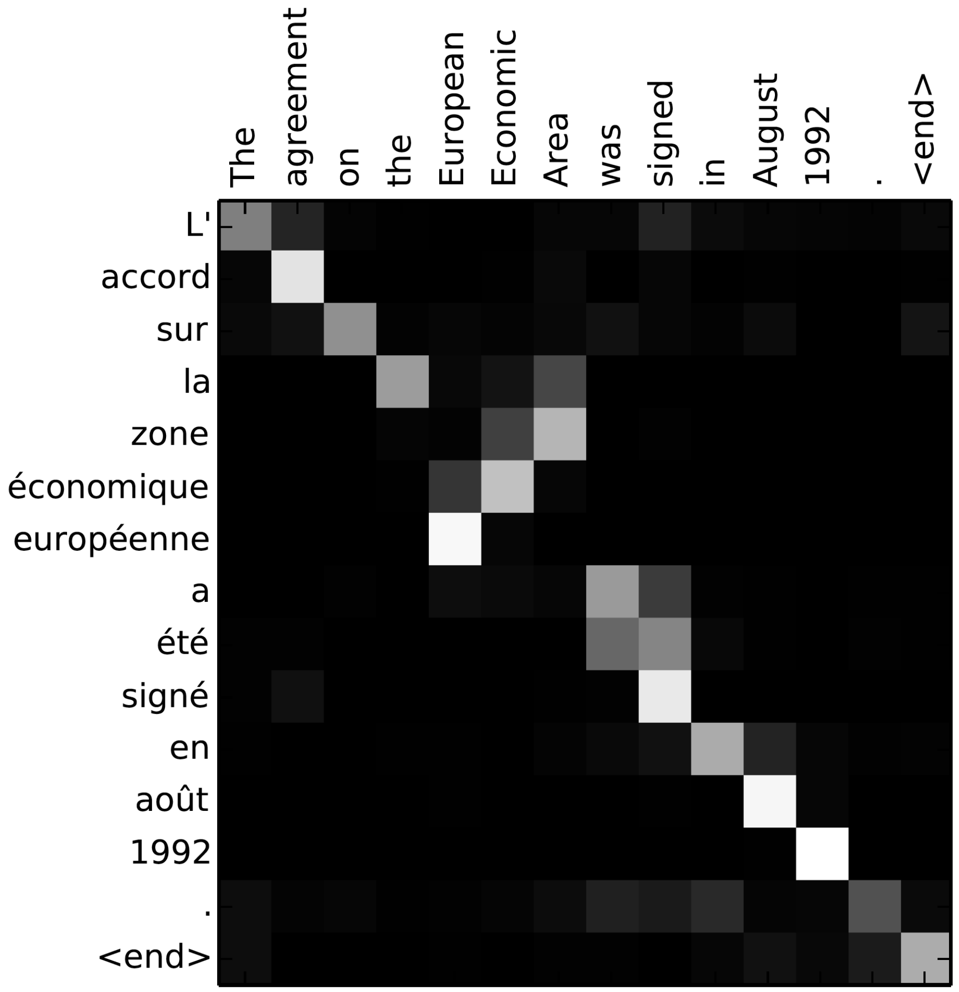
https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Attention Mechanism Improves Results



Neural Machine Translation by Jointly Learning to Align and Translate, Bahdanau et al 2014

Attention Mechanism Gives Insights



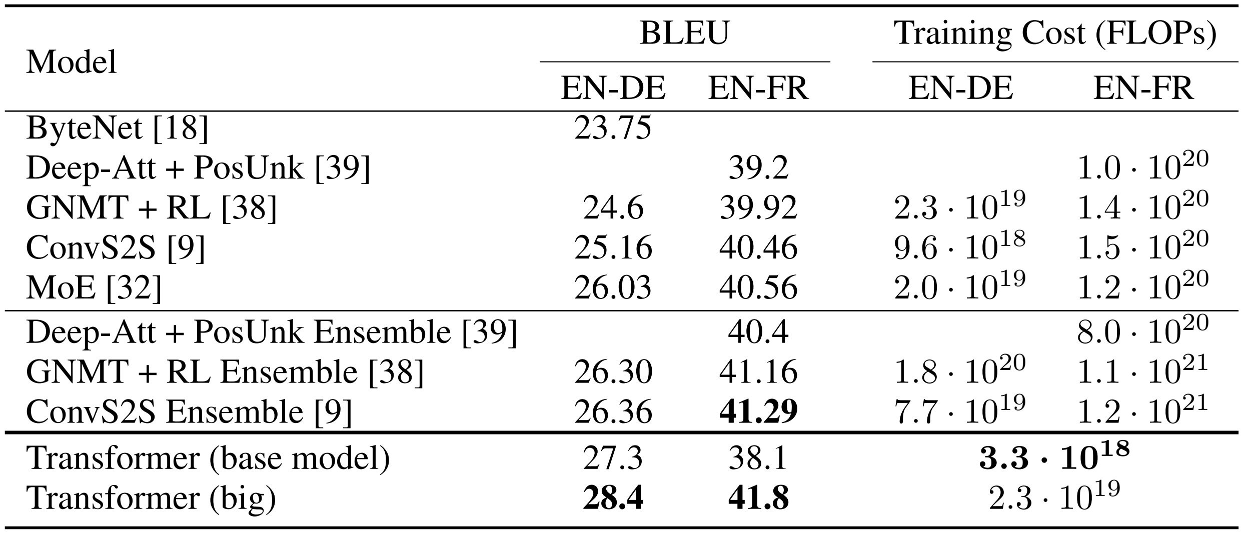
Neural Machine Translation by Jointly Learning to Align and Translate, Bahdanau et al 2014

Outline

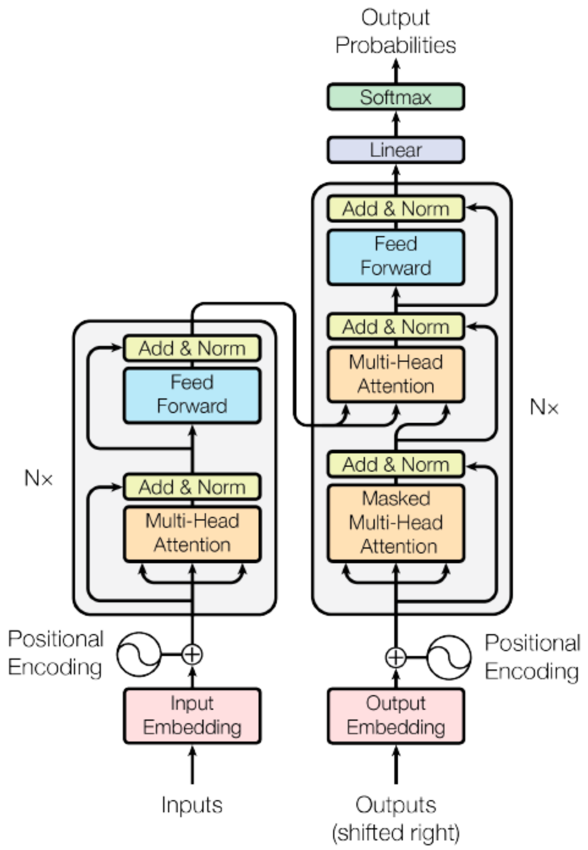
- Attention - A Recap
- Transformers
- Applications in NLP: Large language models

Attention is All You Need

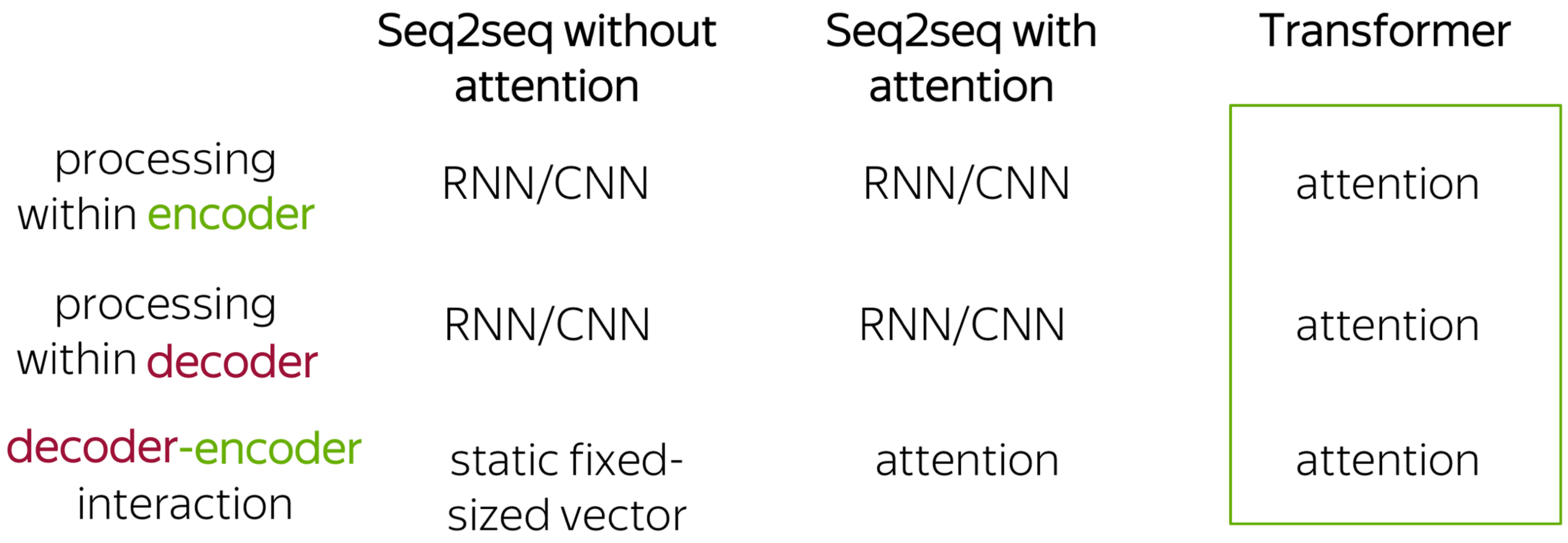
- Published by Vaswani et al 2017
- No recurrence, no convolutions
- Only attention mechanism



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**	$2.3 \cdot 10^{19}$	



Transformer



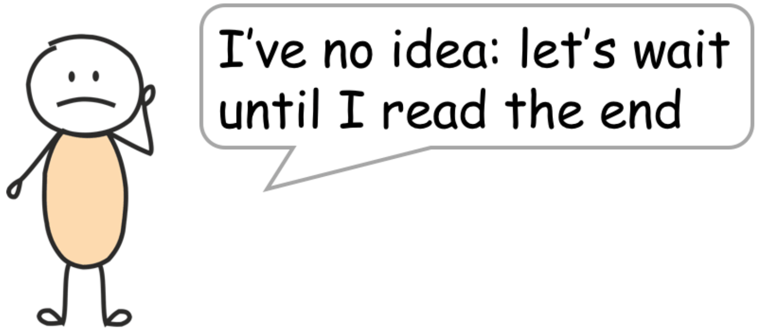
	Seq2seq without attention	Seq2seq with attention	Transformer
processing within encoder	RNN/CNN	RNN/CNN	attention
processing within decoder	RNN/CNN	RNN/CNN	attention
decoder-encoder interaction	static fixed-sized vector	attention	attention

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

Attention Is All You Need

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

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



RNNs

O(N) steps to process a sentence with length N

Transformer

Constant number of steps to process any sentence

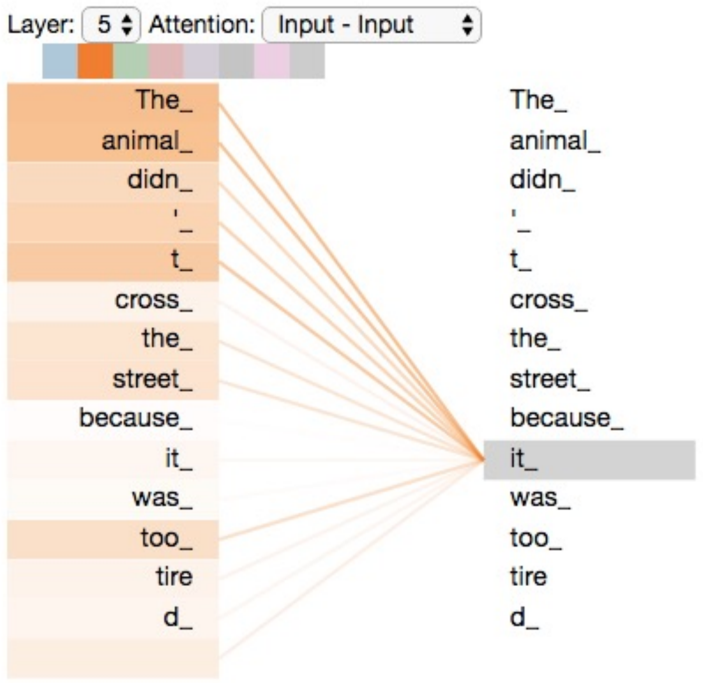
https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Transformer - Tokens Look At Each Others

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

Self Attention – One Example

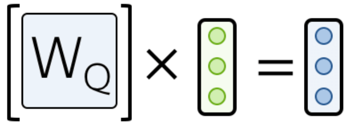
- One example:
  - When the model processes the word ***it***, self attention allows resolving coreference resolution
  - In general, self attention allows to look at clues within a sentence to better represent each word in a sentence



http://jalammar.github.io/illustrated-transformer/

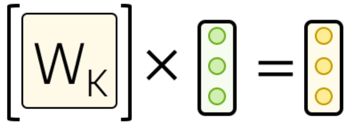
Self-Attention in Details

Each vector receives three representations ("roles")



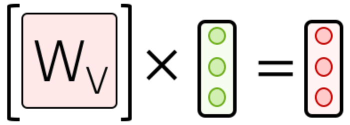
Query: vector **from** which the attention is looking

"Hey there, do you have this information?"



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

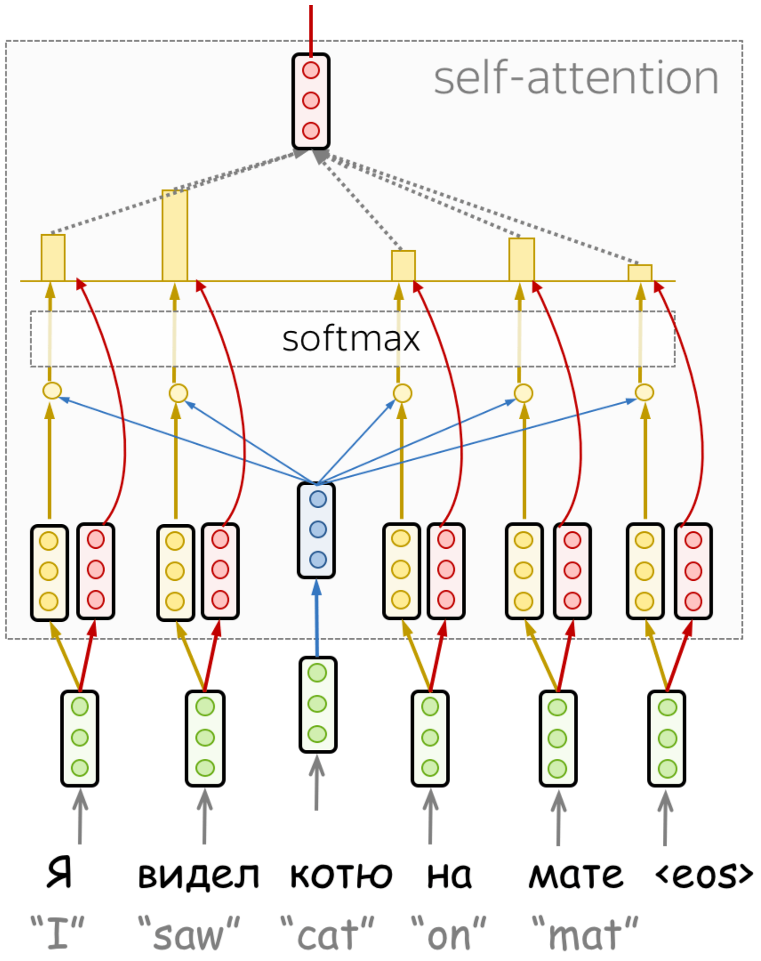
"Hi, I have this information – give me a large weight!"



Value: their weighted sum is attention output

"Here's the information I have!"

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



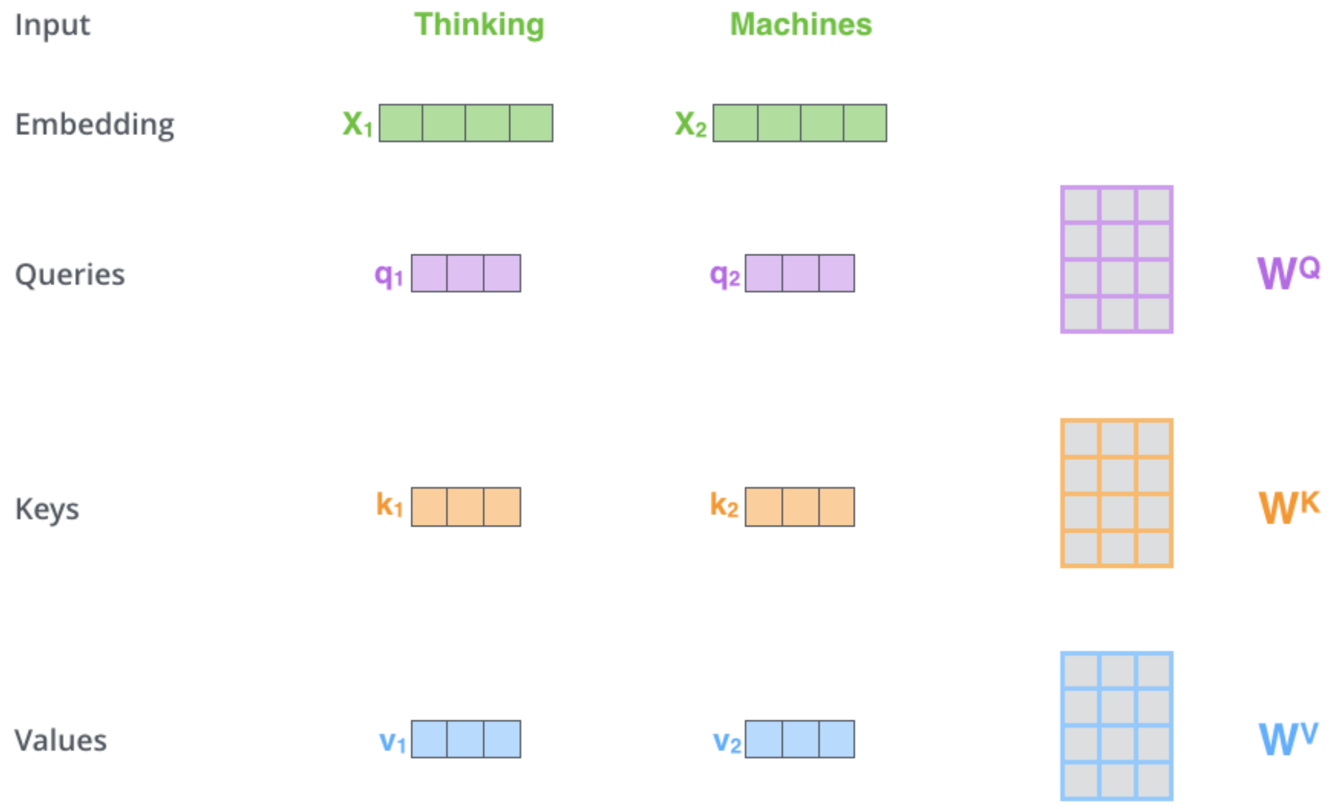
Self Attention in Details

$$Attention(\underset{\text{from}}{q}, \underbrace{k, v}_{\text{to}}) = \overbrace{softmax\left(\frac{qk^T}{\sqrt{\underset{\text{vector dimensionality of K, V}}{d_k}}}\right)}^{\text{Attention weights}} v$$

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

Self Attention in Details

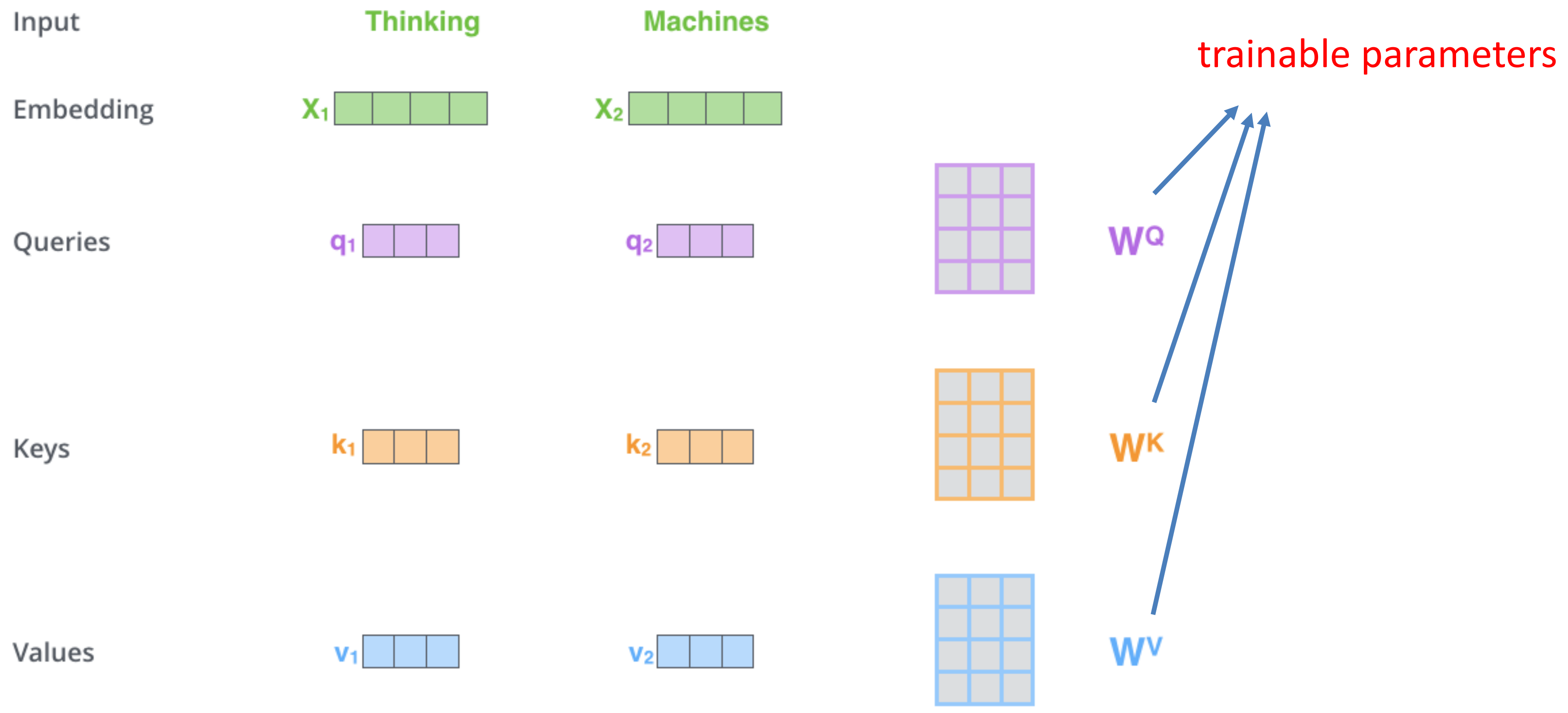
- Step 1: Create *query*, *key* and *value* for each word



http://jalammar.github.io/illustrated-transformer/

Self Attention in Details

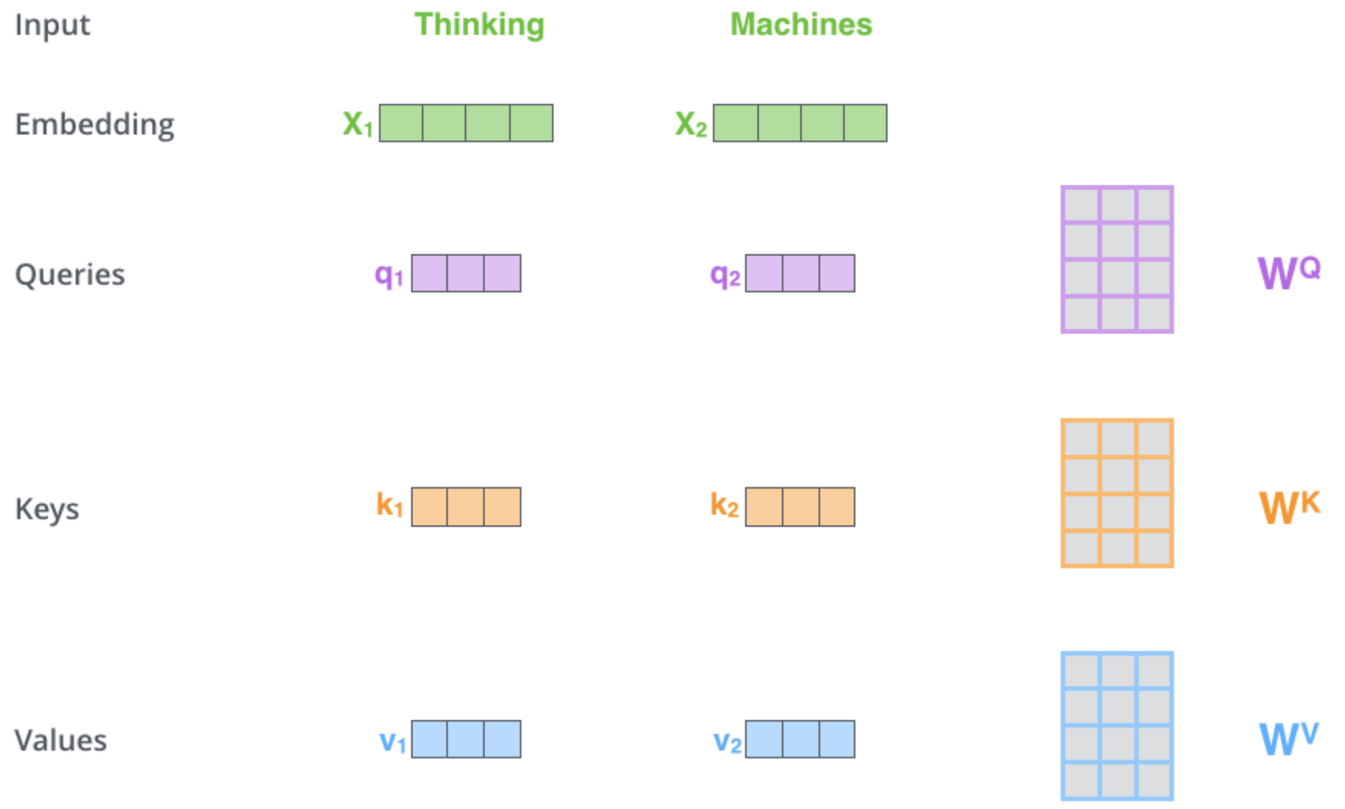
- Step 1: Create *query*, *key* and *value* for each word



http://jalammar.github.io/illustrated-transformer/

Self Attention in Details

- Step 1: Create *query*, *key* and *value* for each word



$q_i = x_i \times W^Q$

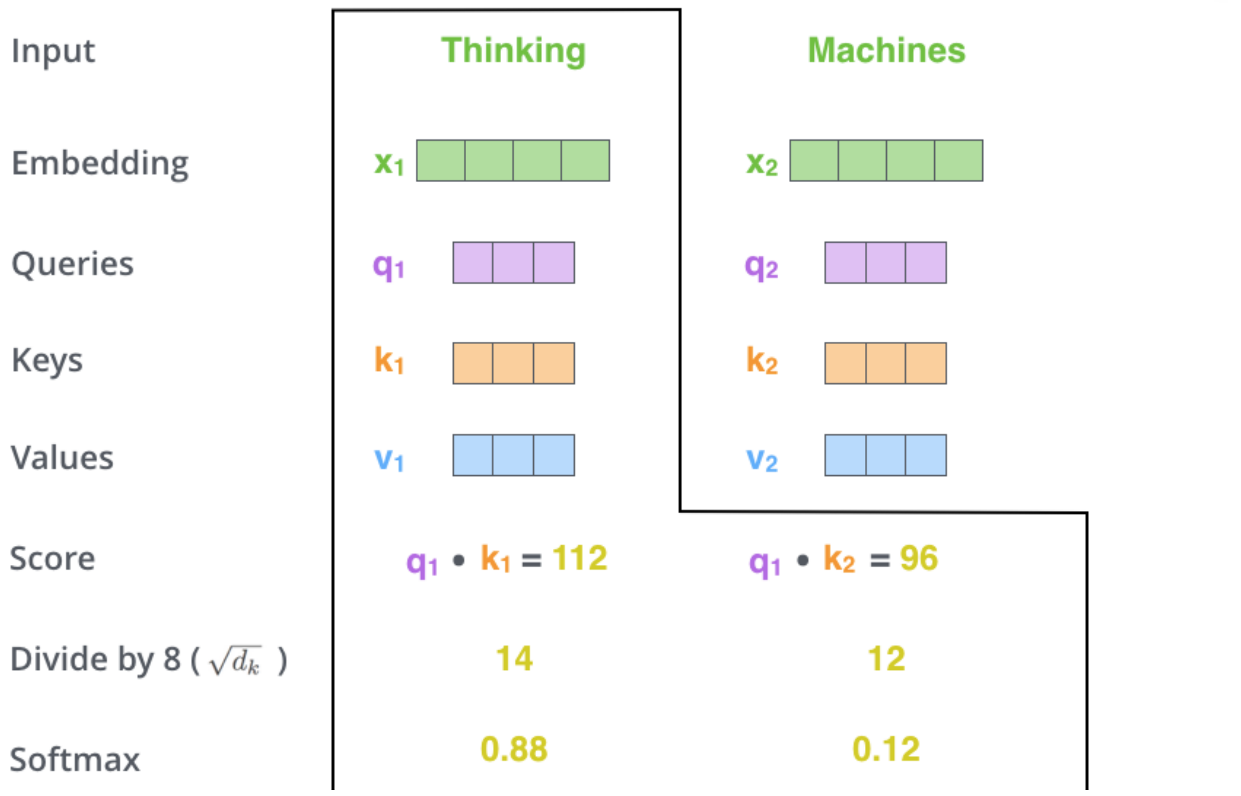
$k_i = x_i \times W^K$

$v_i = x_i \times W^V$

http://jalammar.github.io/illustrated-transformer/

Self Attention in Details

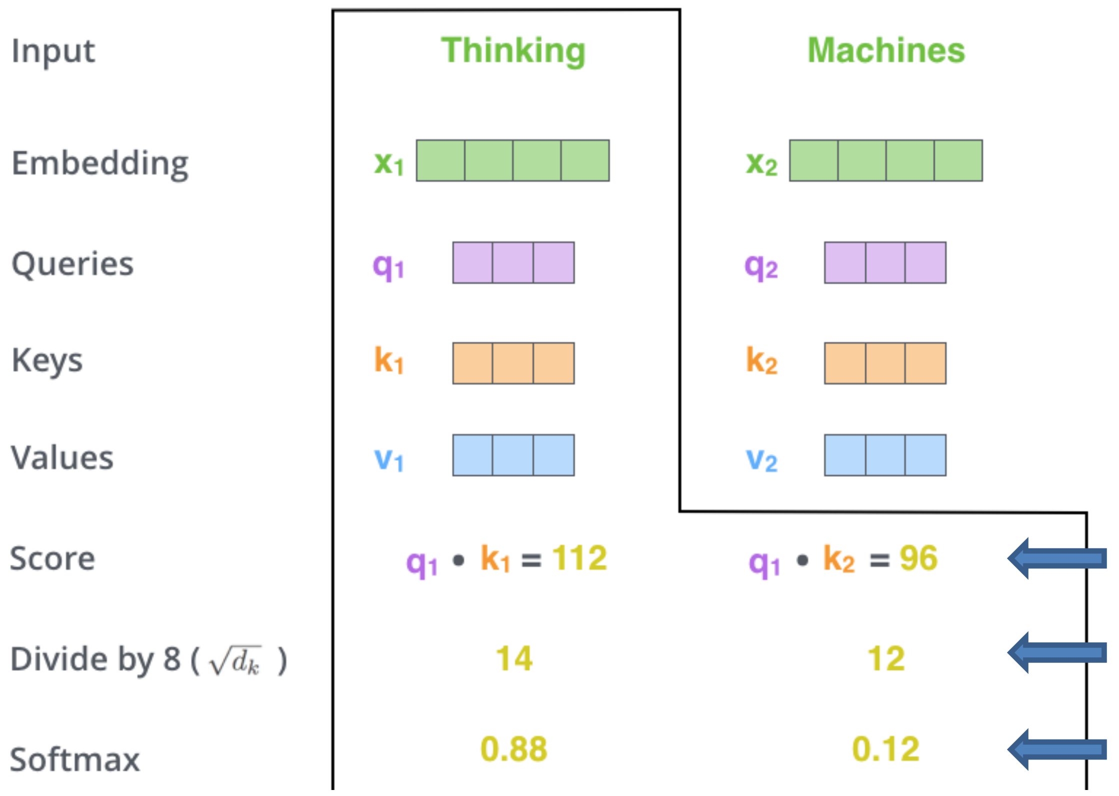
- Step 2: Calculate scores for each word with others



http://jalammar.github.io/illustrated-transformer/

Self Attention in Details

- Step 2: Calculate scores for each word with others



scoring function

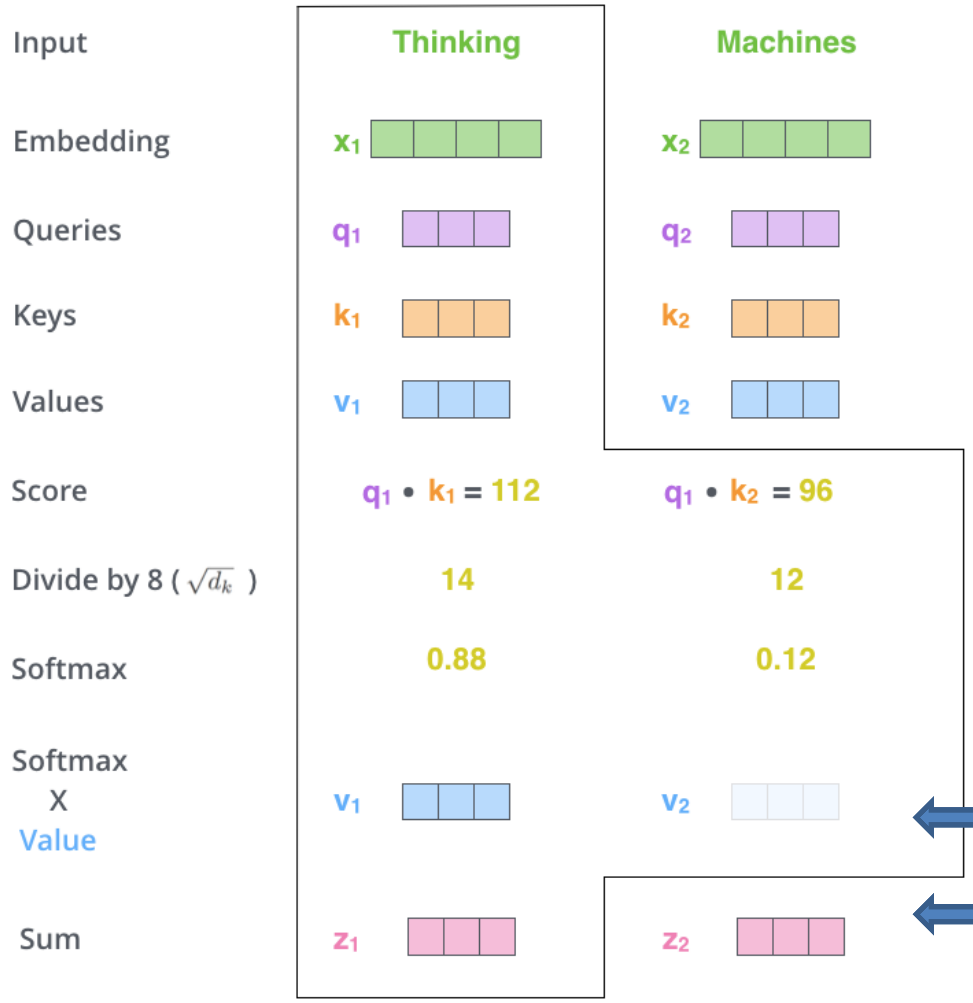
trick to have more stable gradients

normalization step

http://jalammar.github.io/illustrated-transformer/

Self Attention in Details

- Step 3: Compute the values and the output



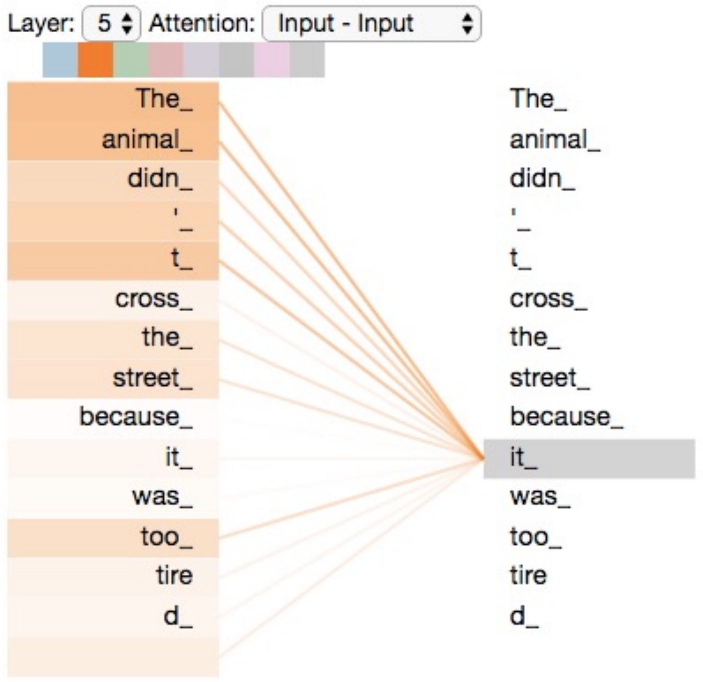
compute all the values

sum up the weighted value vectors

http://jalammar.github.io/illustrated-transformer/

Self Attention – One Example

- One example:
  - When the model processes the word ***it***, self attention allows resolving coreference resolution
  - In general, self attention allows to look at clues within a sentence to better represent each word in a sentence



http://jalammar.github.io/illustrated-transformer/

Transformer

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

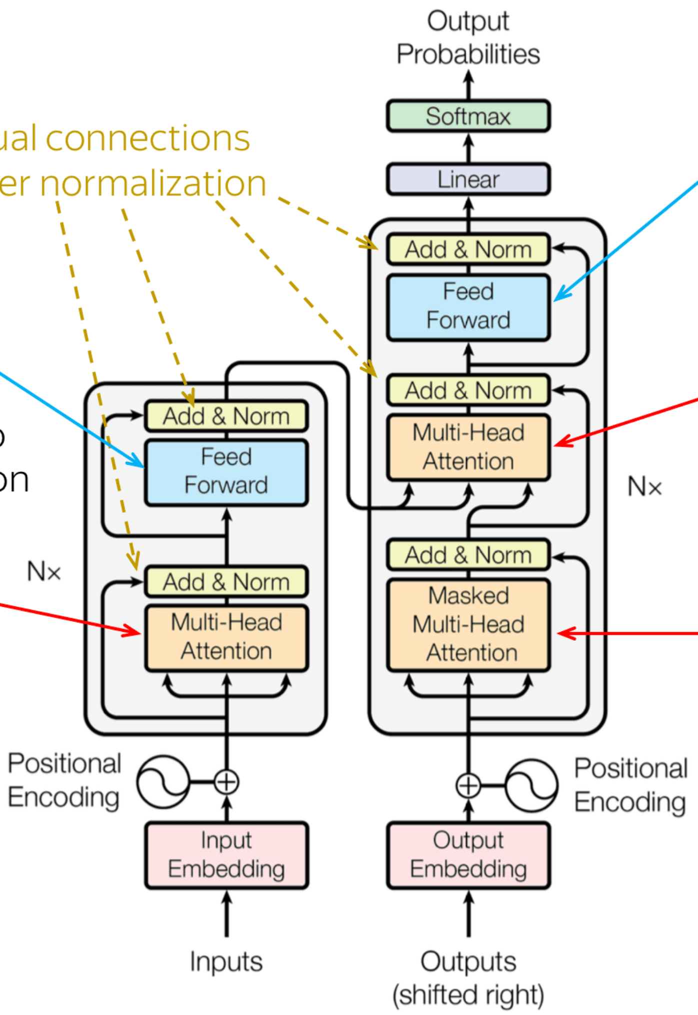
Feed-forward network: after taking information from other tokens, take a moment to think and process this information

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

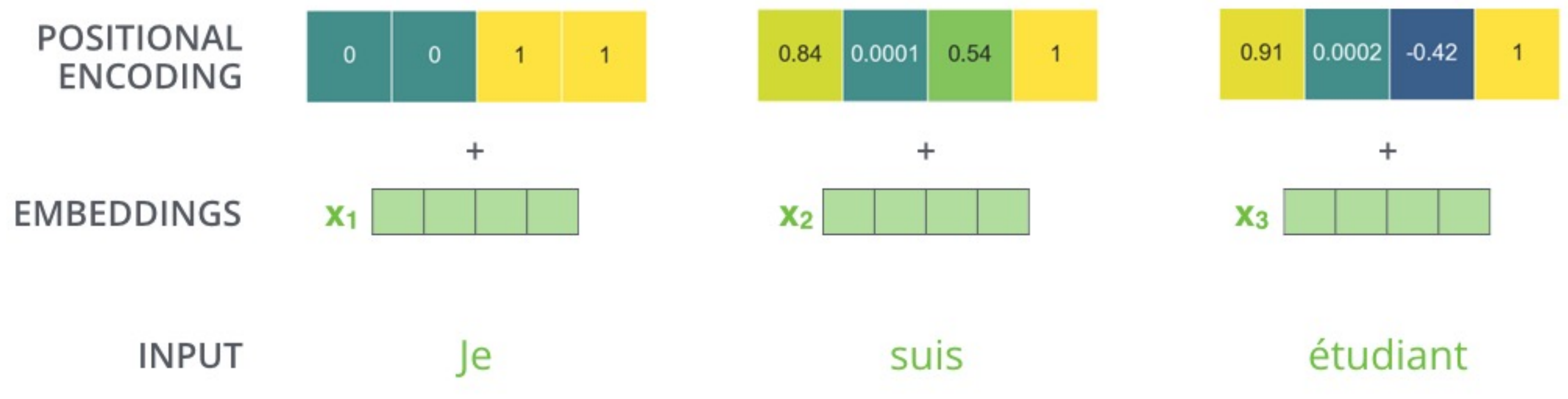
queries, keys, values are computed from decoder states

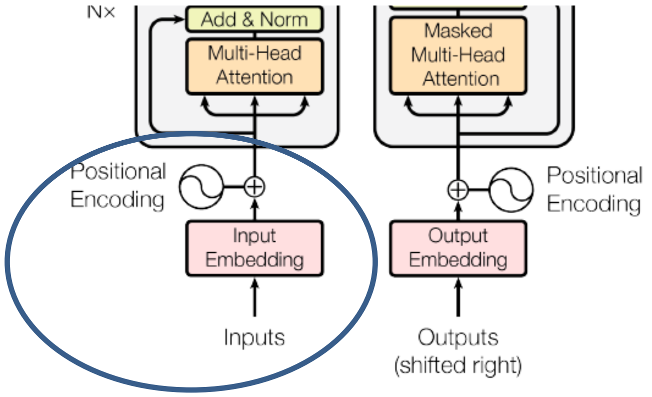


https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Positional Encoding

- Encode the relative position of words to each other in a sequence

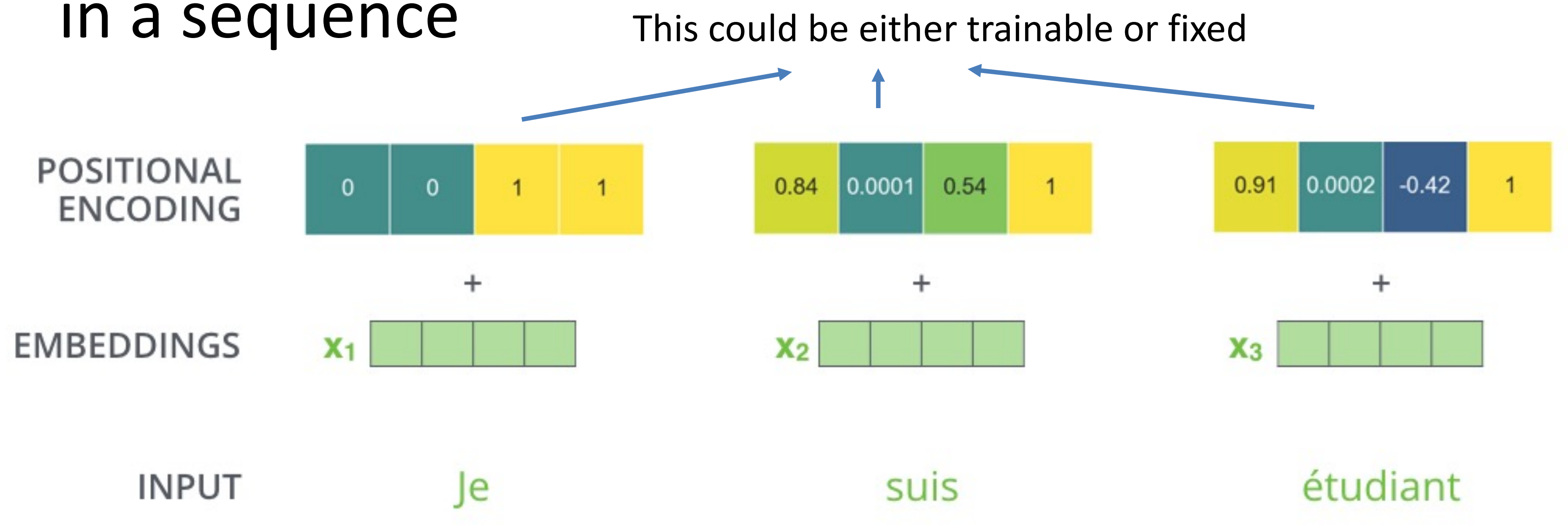


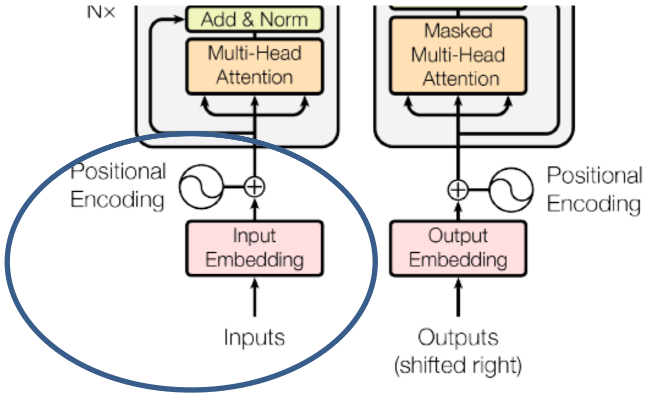


http://jalammar.github.io/illustrated-transformer/

Positional Encoding

- Encode the relative position of words to each other in a sequence





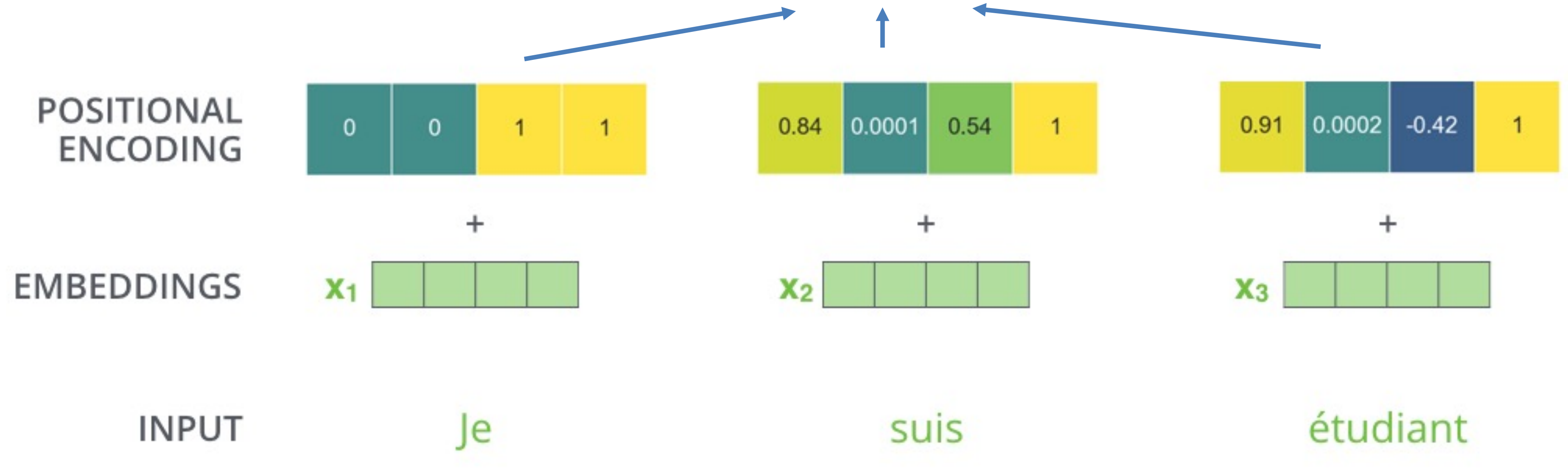
http://jalammar.github.io/illustrated-transformer/

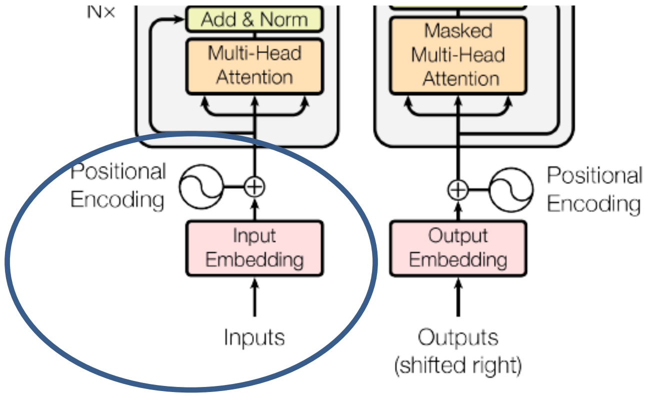
Positional Encoding

- Encode the relative position of words to each other in a sequence

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

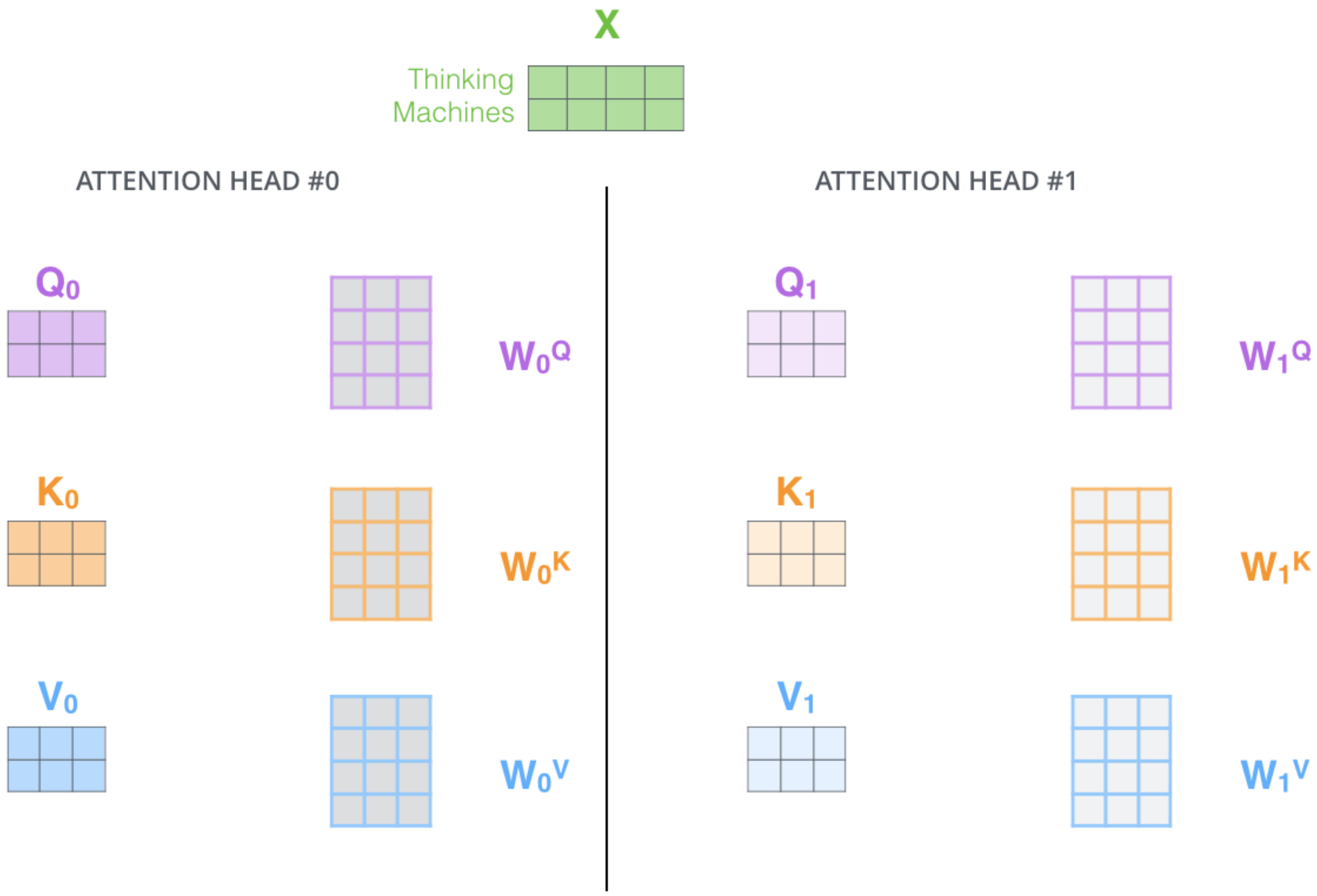
or position embeddings





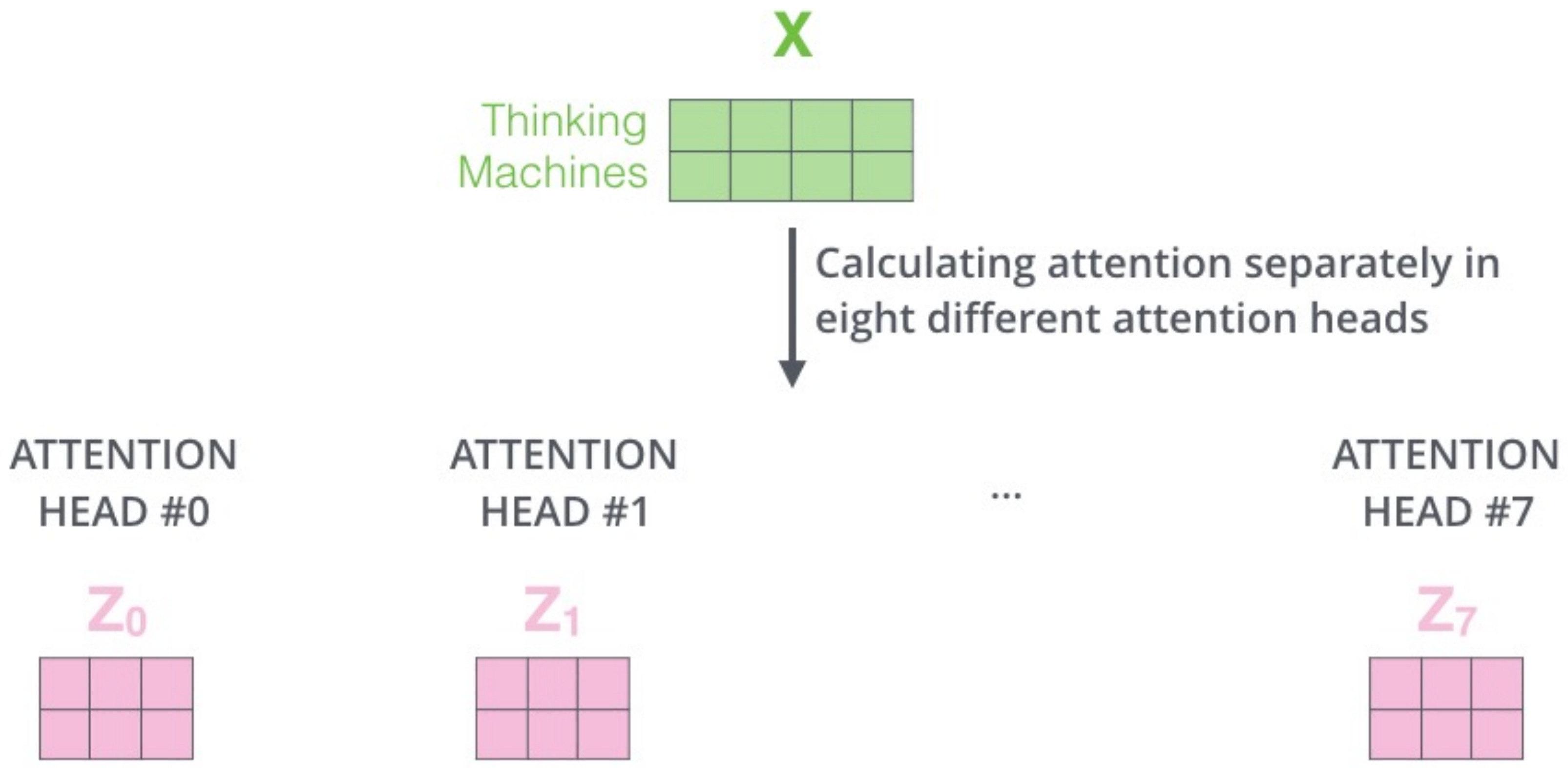
http://jalammar.github.io/illustrated-transformer/

Multi-head Attention



http://jalammar.github.io/illustrated-transformer/

Multi-head Attention



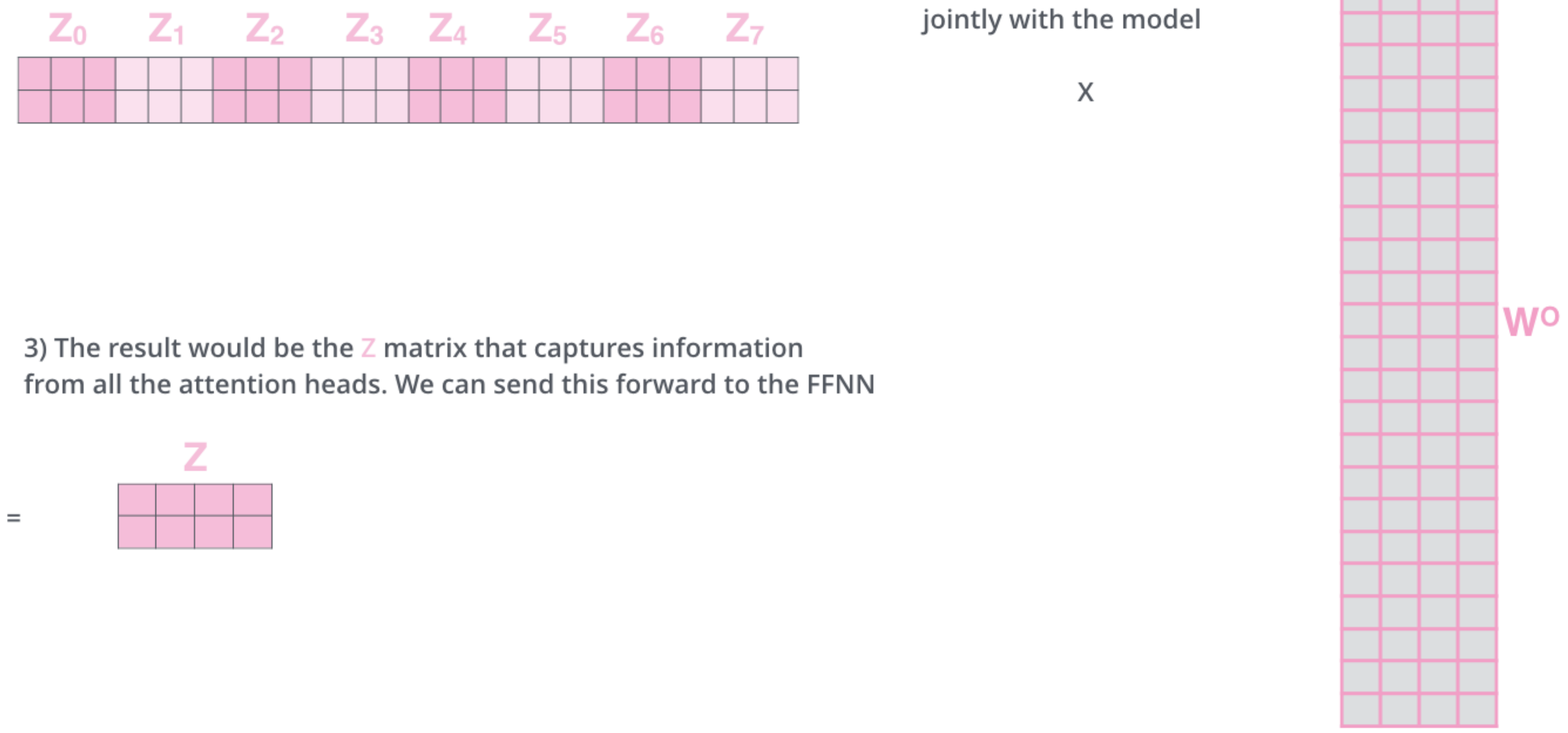
http://jalammar.github.io/illustrated-transformer/

Multi-head Attention

1) Concatenate all the attention heads

2) Multiply with a weight matrix W^O that was trained jointly with the model

3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN



http://jalammar.github.io/illustrated-transformer/

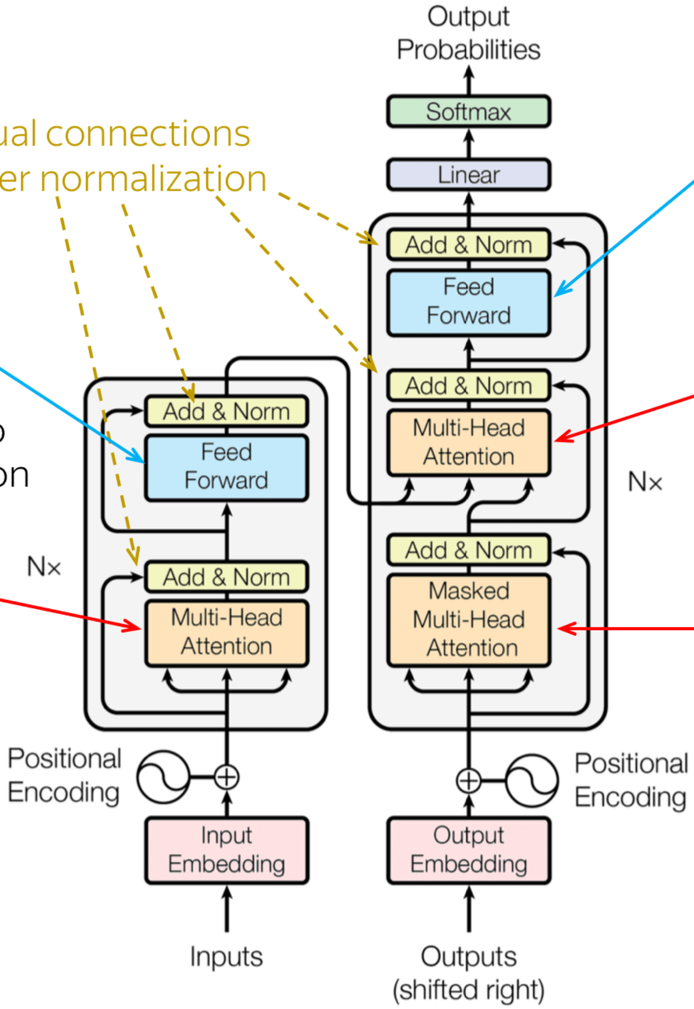
Transformer

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

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

Transformer: Advantages and Challenges

Complexity per Layer

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Parallelization

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Handling Long-range Dependencies

- Learning long-range dependencies is a key challenge in many sequence to sequence tasks
- The length of the path between any combination of positions in the input and output is one key factor

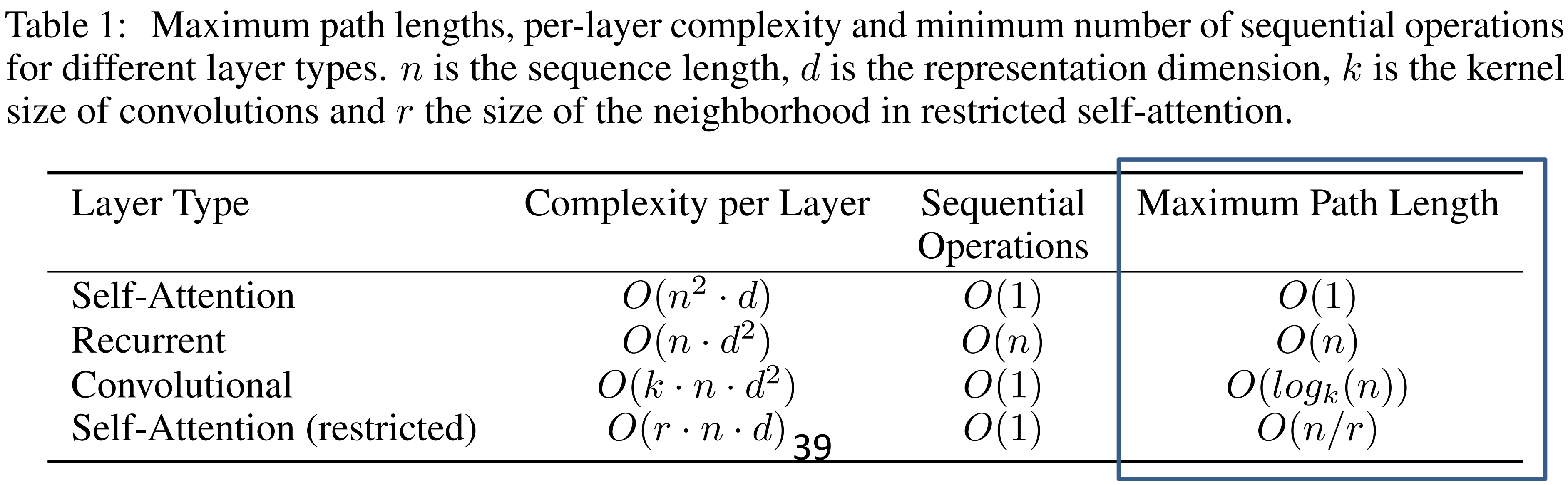
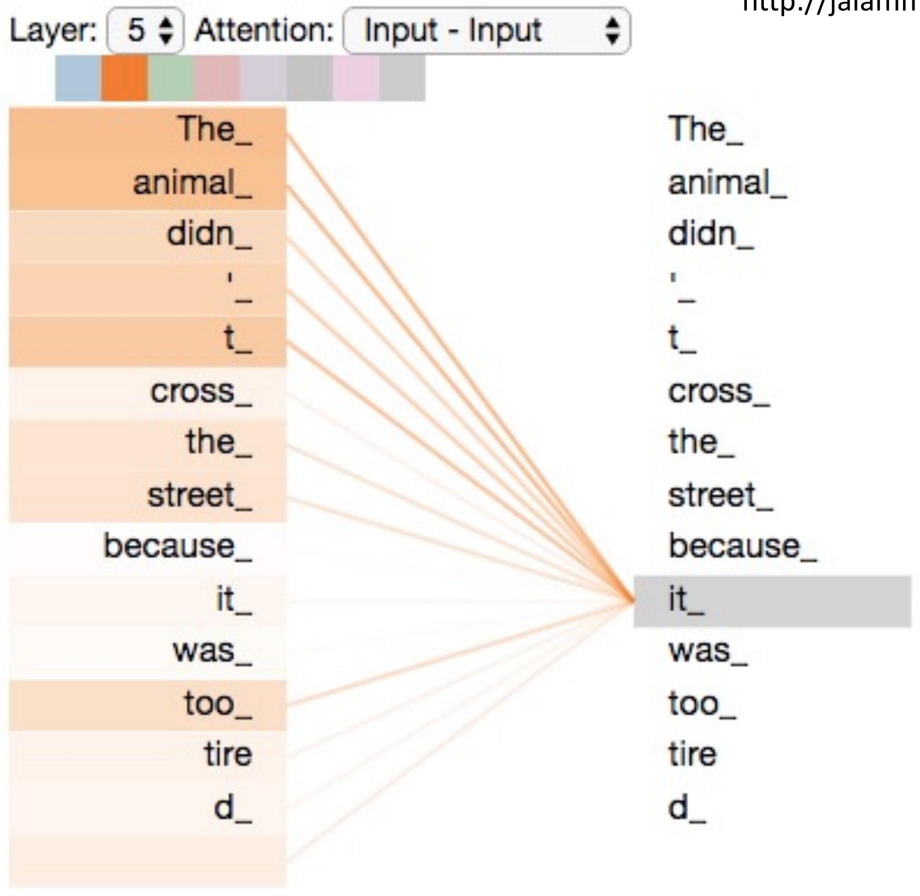


Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

More Interpretable Models

http://jalammar.github.io/illustrated-transformer/



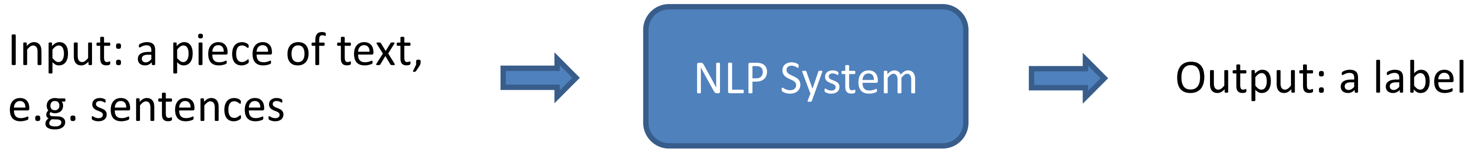
- More in https://github.com/jessevig/bertviz

Outline

- Attention – A Recap
- Transformers
- Applications in NLP: BERT

Text Classification

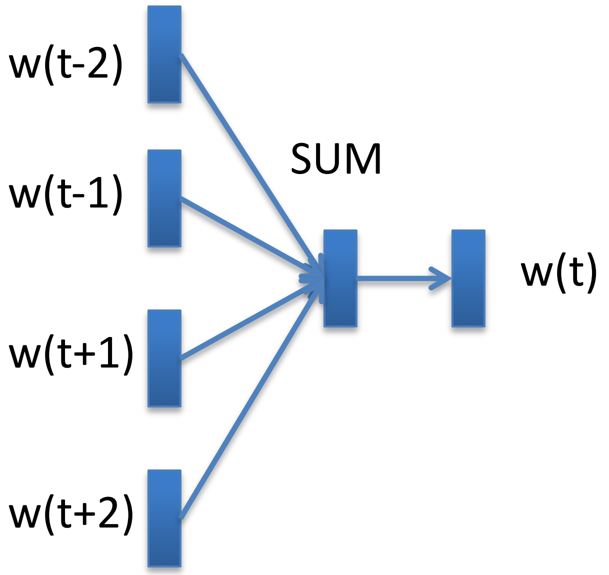
- Problem settings:



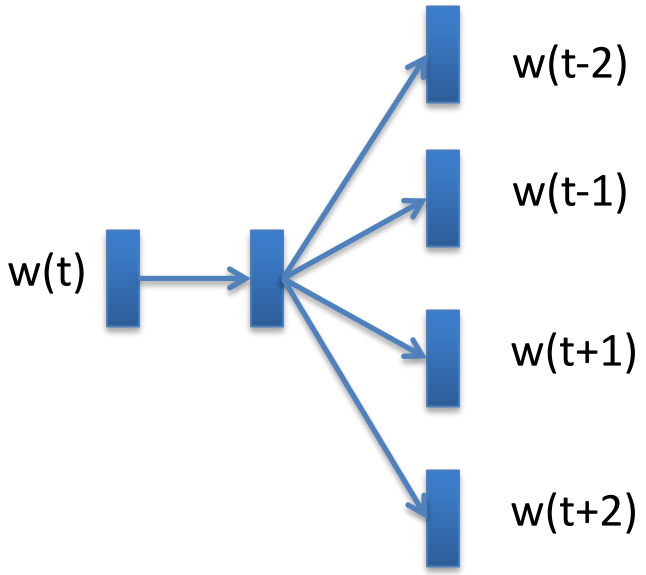
- Some examples in a sentiment analysis task:
  - This movie is great $\Rightarrow$ positive
  - There are many meaningless points in the documentary film $\Rightarrow$ negative
- Modern NLP methods:
  - Leverage word embeddings (word $\Rightarrow$ vectors)
  - Input text $\Rightarrow$ matrices

Word Embeddings

- Word2vec - proposed by T. Mikolov 2013
- https://code.google.com/p/word2vec/



CBOW



Skip-gram

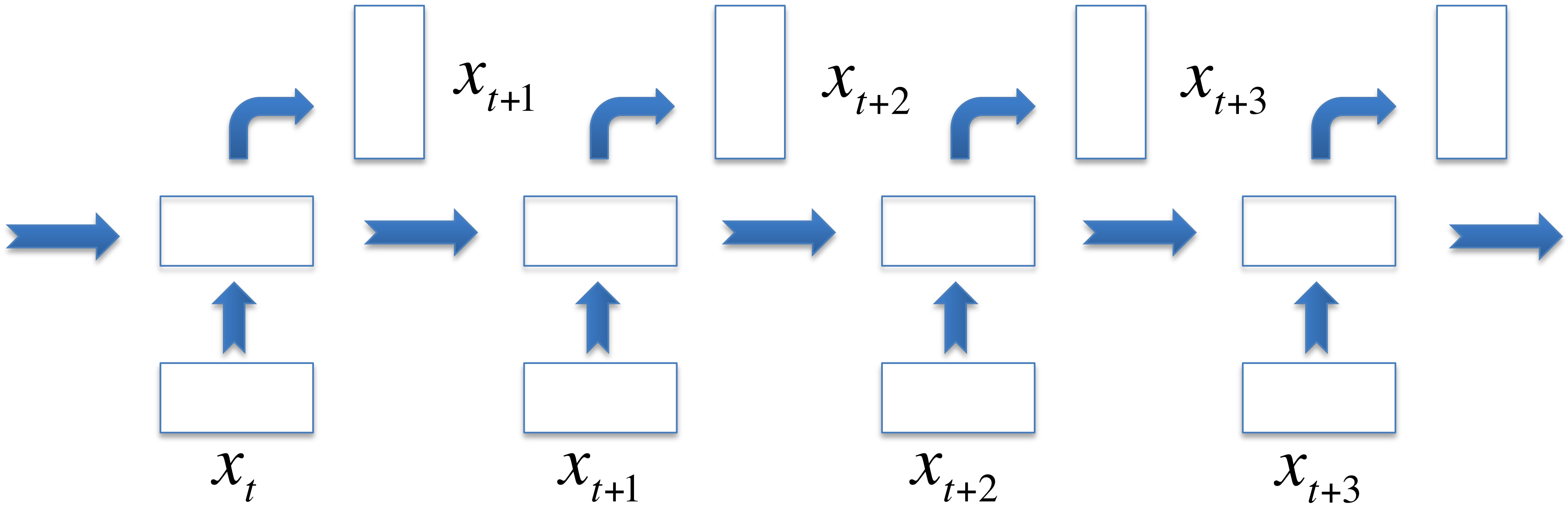
Word Embeddings

- Words have multiple senses, e.g.
  - She keeps all her money in a ***bank***
  - She prefers to just sit on a ***bank*** and relax ..
- i.e. *one word : one vector* is not a good approach

Contextual Embeddings

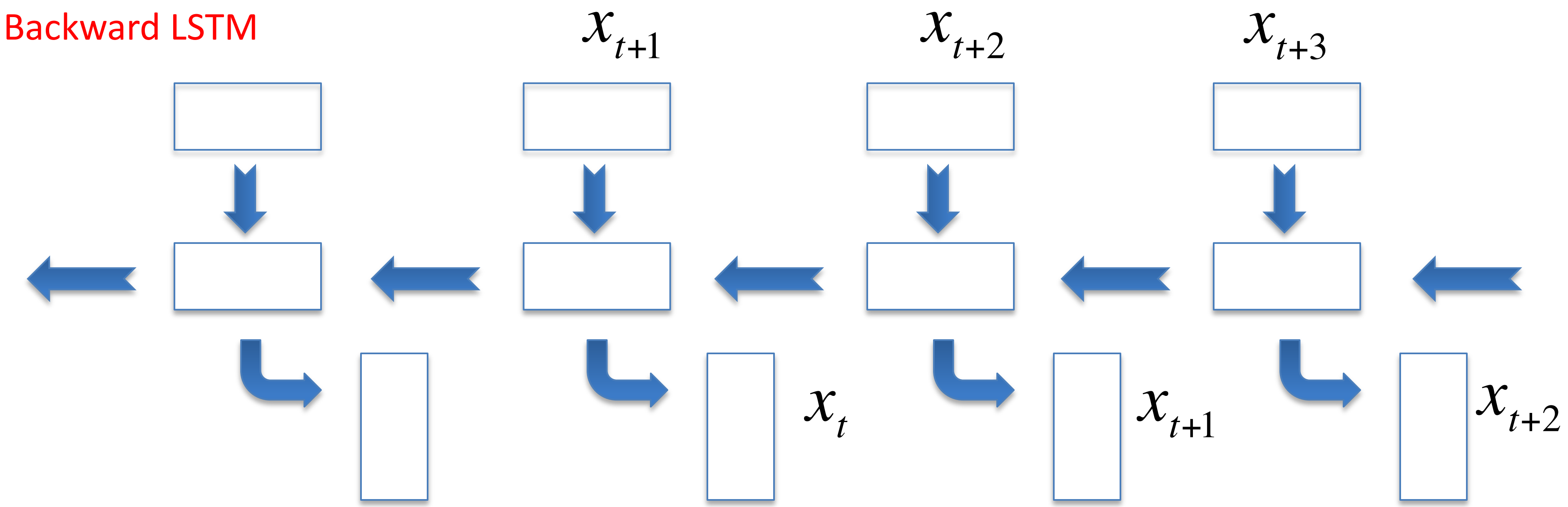
- Instead of saving for each word a vector, save a model that can dynamically generate for each word a vector depending on its context
- Deep contextualized word representations
  Peters et al 2018
  - The authors named their model - **ELMO**

ELMO - BiLSTM

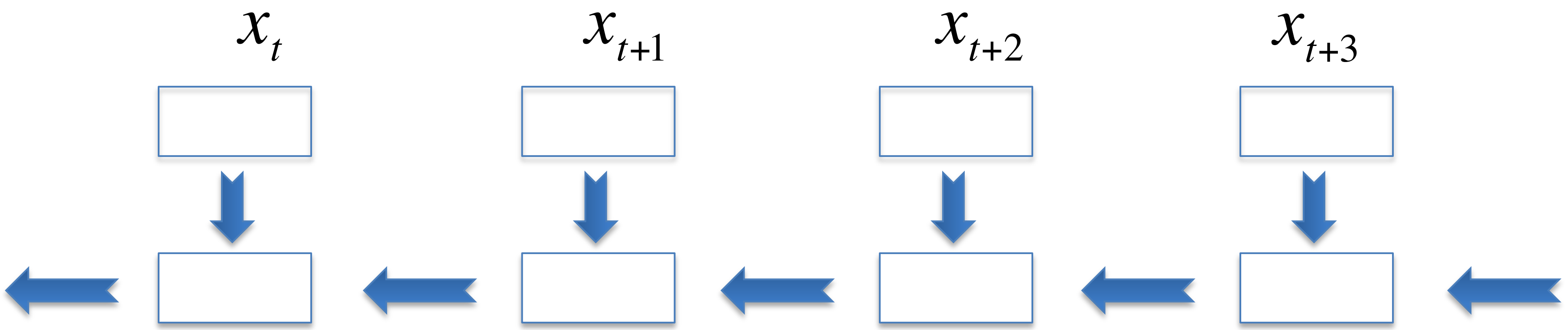


Forward LSTM

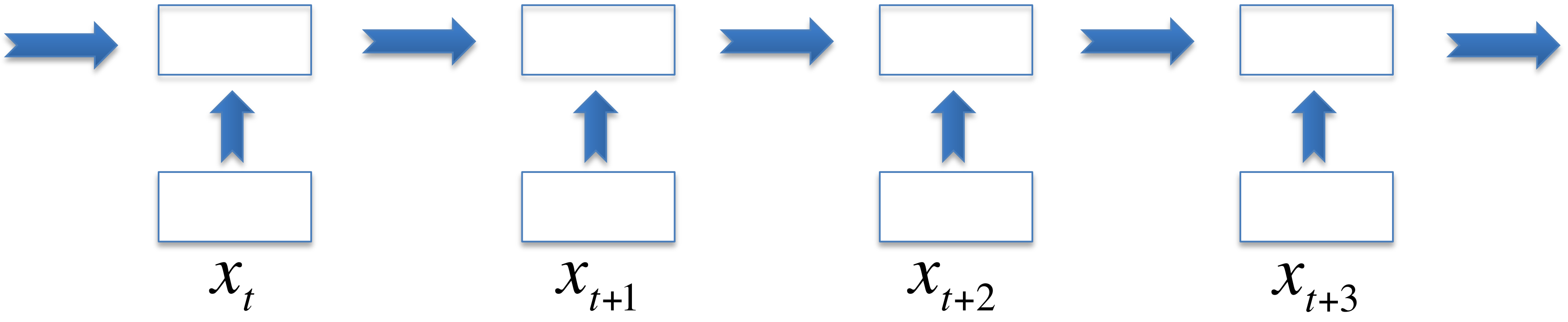
ELMO - BiLSTM



ELMO - BiLSTM

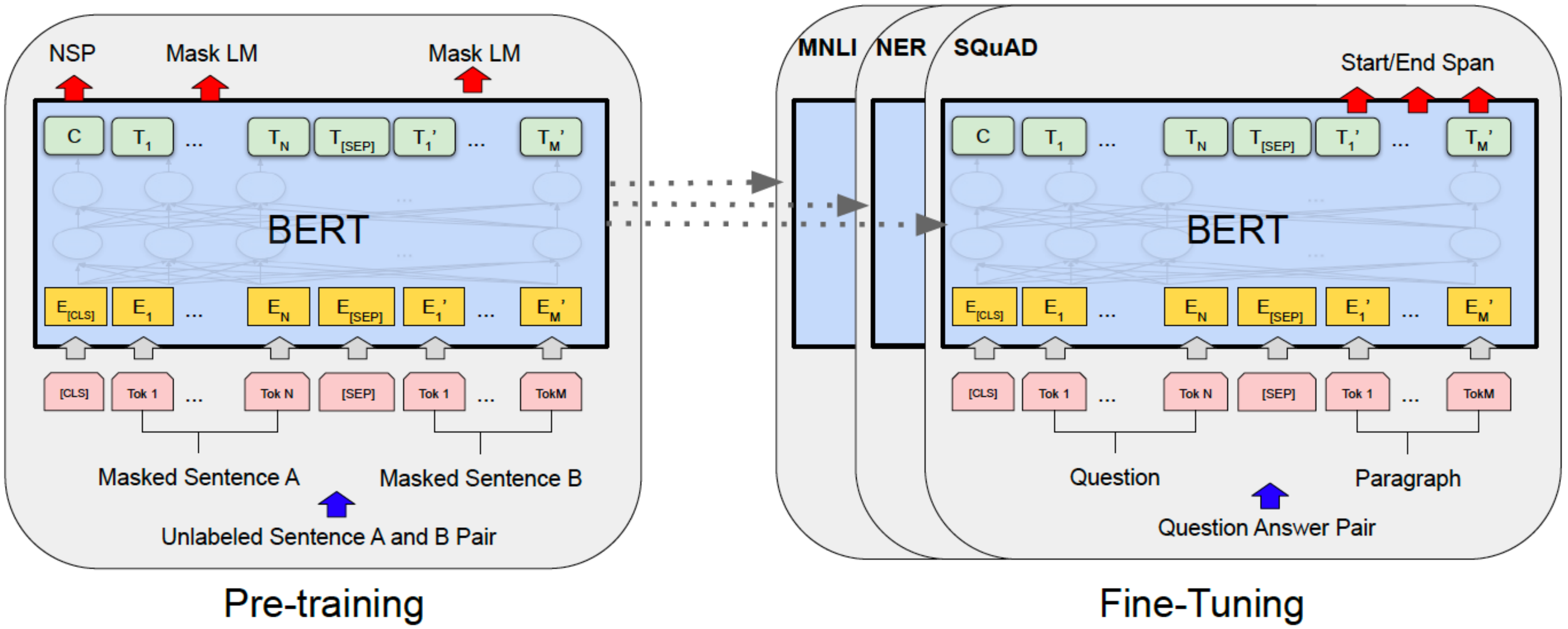


concatenation of the hidden states $\Rightarrow$ contextualized embeddings



BERT

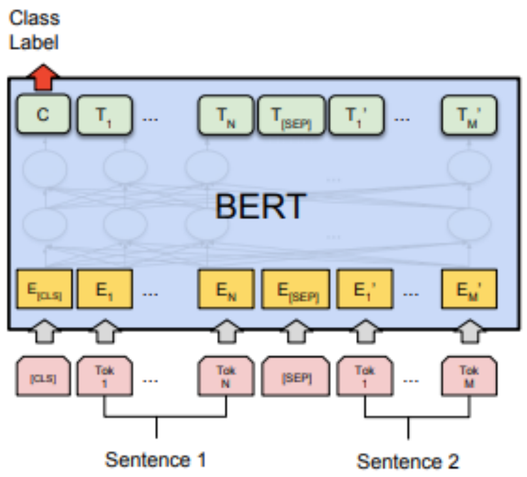
- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al 2019



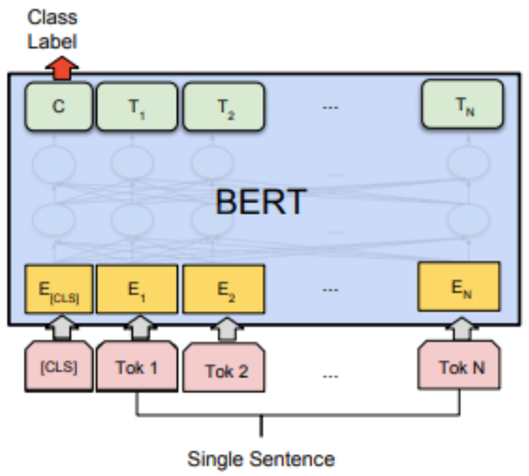
Contextual Embedding - BERT

- Replace the BiLSTM layers with a Transformer
- Two crucial changes in the objectives:
  - Masked language model
    - Randomly mask some percentages of tokens, e.g. 15%
  - Next sentence prediction task
- Pretrain it with a very large amount of training data and save the pre-trained model for further apps
- Pretrained BERT can be seen as a powerful feature extractor

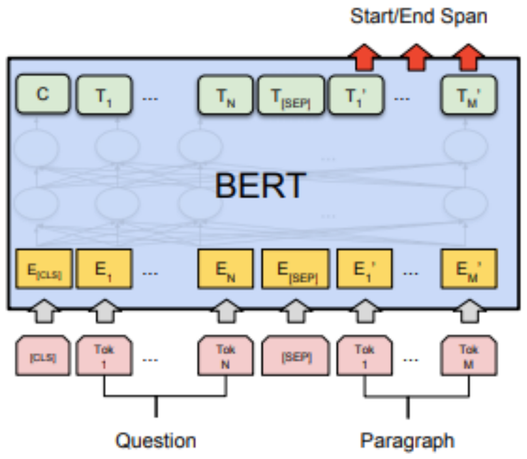
Fine-tuning BERT – Some Applications



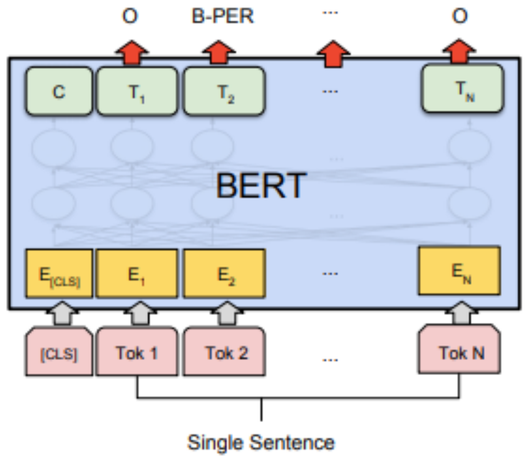
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA



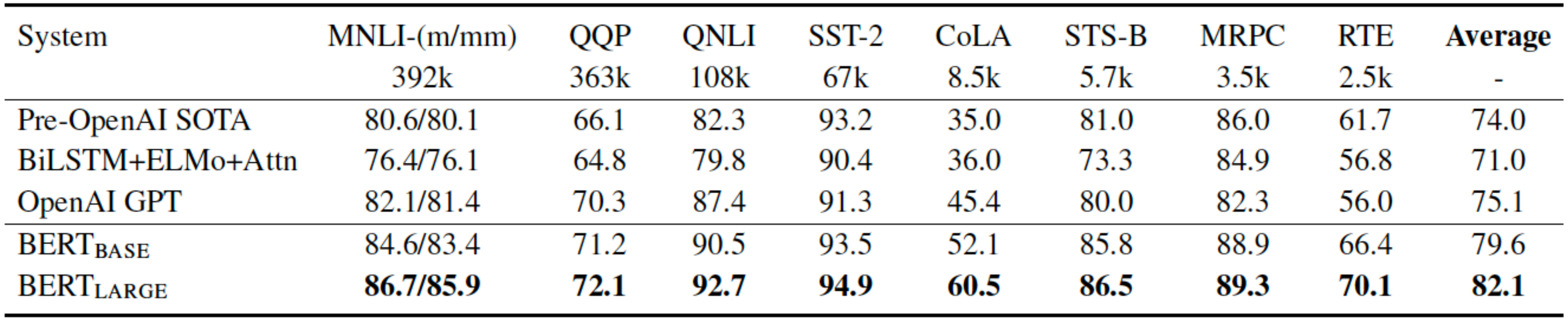
(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Fine-tuning BERT – Some Results

- Some results on GLUE benchmark dataset



System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT$_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT$_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

- Many NLP systems are based on pretrained BERT
  – It works pretty well on a wide range of domains
  – It requires less training data then training a system from scratch

Thanks for listening!