


# Seq2seq, NMT, Transformer

Milan Straka

 April 22, 2024



Charles University in Prague  
Faculty of Mathematics and Physics  
Institute of Formal and Applied Linguistics



unless otherwise stated



## Sequence-to-Sequence Architecture



Sequence-to-Sequence is a name for an architecture allowing to produce an arbitrary output sequence  $y_1, \dots, y_M$  from an input sequence  $\mathbf{x}_1, \dots, \mathbf{x}_N$ .

Unlike span labeling/CTC, no assumptions are necessary and we condition each output sequence element on all input sequence elements and all already generated output sequence elements:

$$P(\mathbf{y} \mid \mathbf{X}) = \prod_{i=1}^M P(y_i \mid \mathbf{x}_1, \dots, \mathbf{x}_N, y_1, \dots, y_{i-1}).$$



# Sequence-to-Sequence Architecture

*encoder* - zpracovává vstupní sekvenci, udělá kontext  
*decoder* - vyrobí výstupní sekvenci na základě kontextu

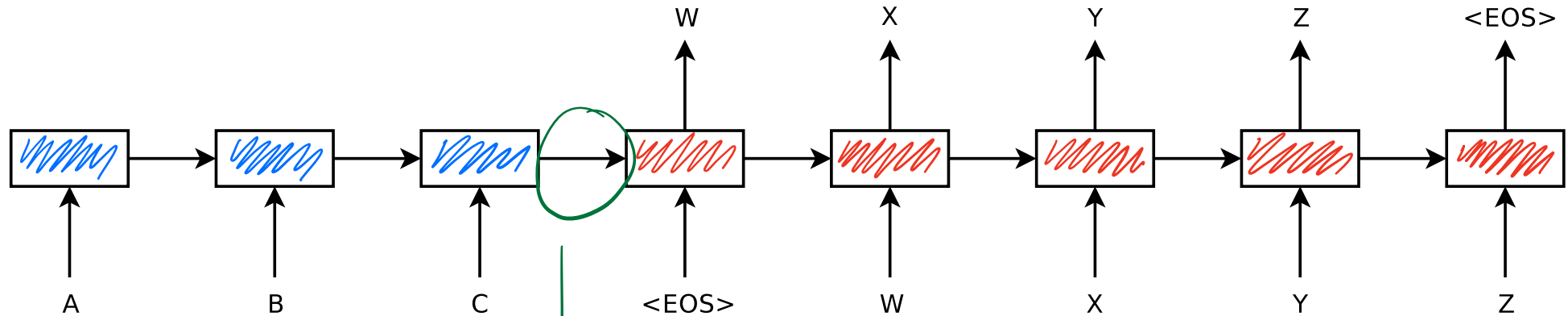
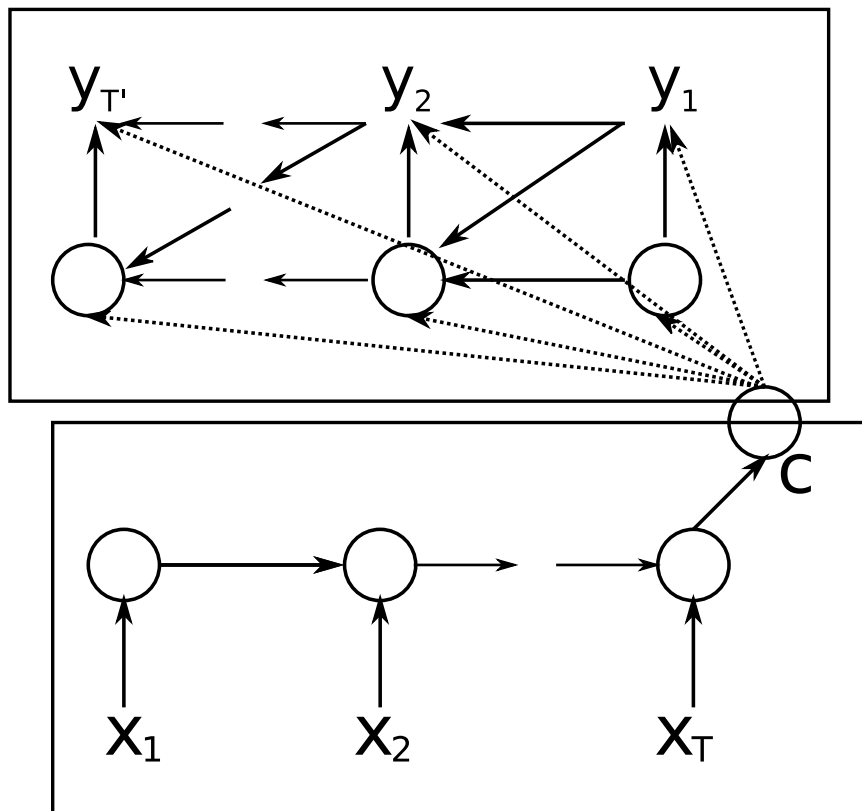


Figure 1 of "Sequence to Sequence Learning with Neural Networks", <https://arxiv.org/abs/1409.0473>

tady bych v ideální představě měl mít reprezentaci celého stavu  
 - a blbě je, že ten vektor je fixně velký...



Decoder



Encoder

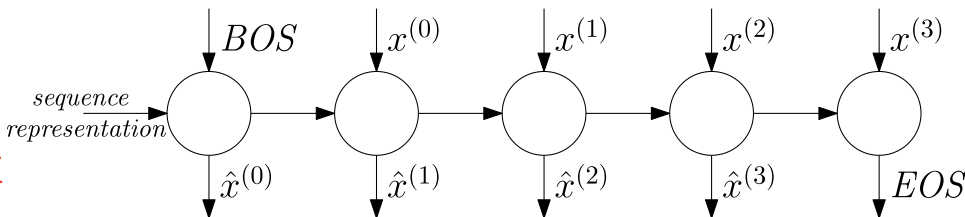
Figure 1 of "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", <https://arxiv.org/abs/1406.1078>



aby se uoíl nnd gíníron  
"históni"

Presto se to takhle first frénýje

- blý je to decoder jak velmi pomohl  
s chybami hodnot.



→ chýbnou hodnotu.

Usually, the generated logits are processed by an `arg max`, the chosen word embedded and used as next input.

výstup jednoho je vstupem druhého

tedy dostaliśmy  
distribucję...



In the decoder, we both:

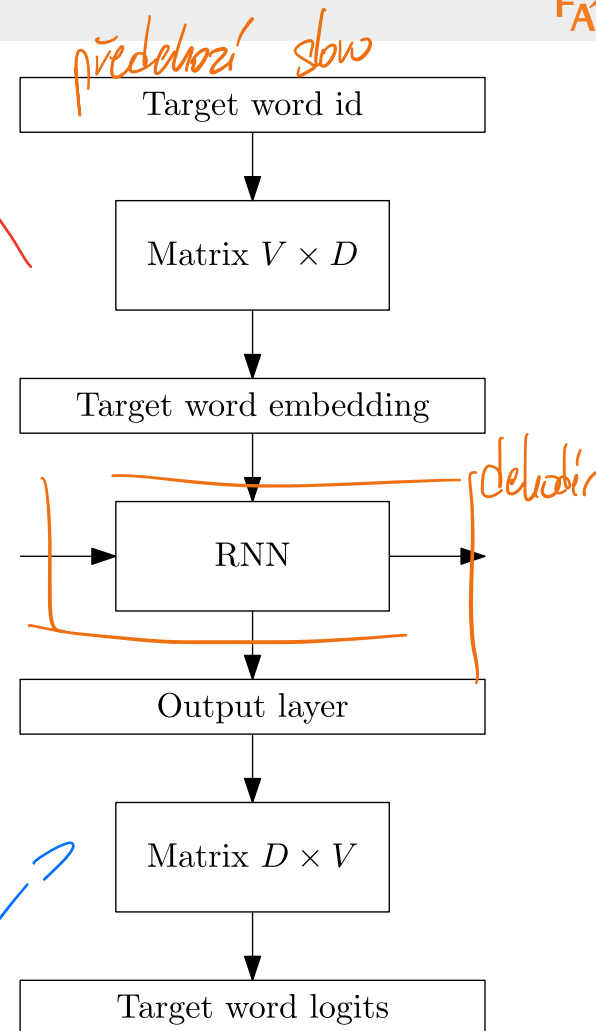
*vytvorí embedding*

- embed the previous prediction, using a matrix of size  $\mathbb{R}^{V \times D}$ , where  $V$  is the vocabulary size and  $D$  is the embedding size;
- classify the hidden state into current prediction, using a matrix of size  $\mathbb{R}^{D \times V}$ .

Both these matrices have similar meaning – they represent words in the embedding space (the first explicitly represents words by the embeddings, the second produces logits by computing weighted cosine similarity of the inputs and columns of the weight matrix).

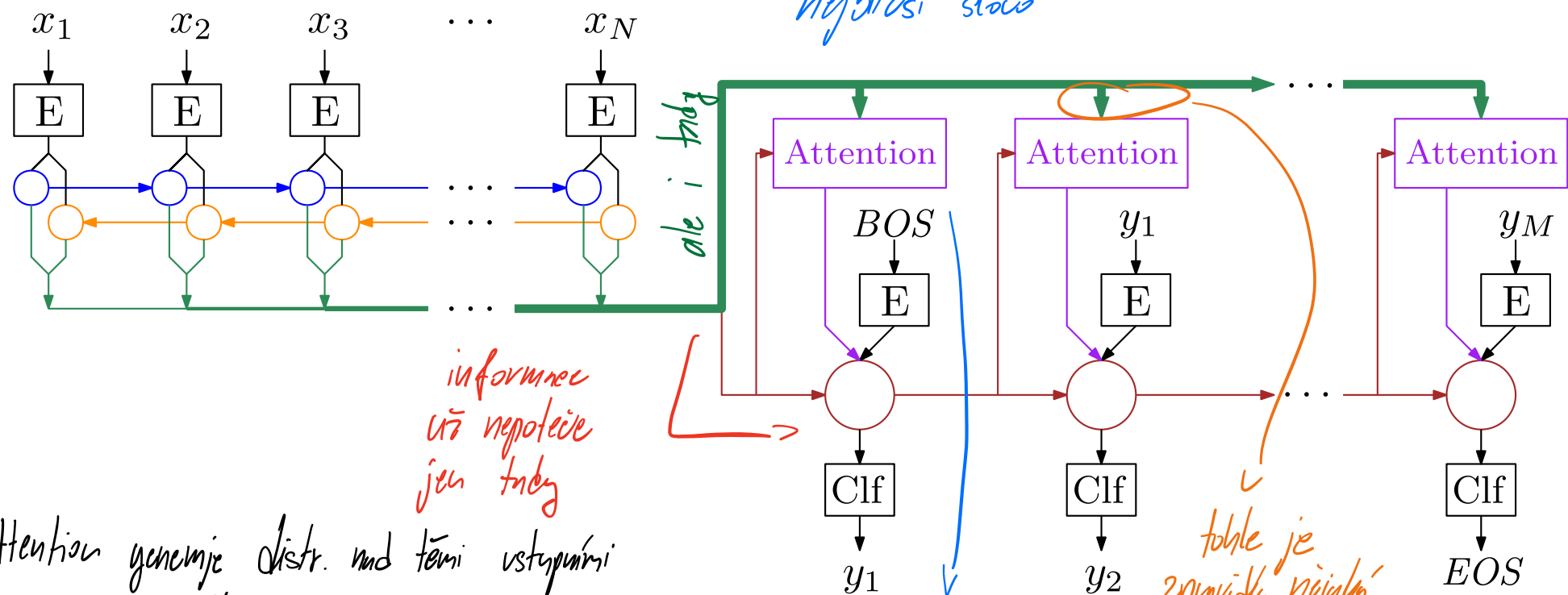
Therefore, it makes sense to **tie** these matrices, i.e., to represent one of them as a transposition of the other.

- However, while the embedding matrix should usually have constant variance per dimension, the output layer should keep the variance of the RNN output; therefore, the output layer matrix is usually the embedding matrix divided by  $\sqrt{D}$ .





# Attention



distancu najbližší slovo, ja' mu chci dať najbližší slovo

informácie už nepotečie je to tady

Attention generuje distr. nad tými vstupnými slovami podľa softmaxu.

vyberá, ktorú informáciu chce vziať

takže je zhrnutá najbližší časť toho vstupu



# Bahdanau Attention

As another input during decoding, we add context vector  $c_i$ :

*minulý výstup*  

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$
*minulý stav*      *kontext výstup*

We compute the context vector as a weighted combination of source sentence encoded outputs:

$$c_i = \sum_j \alpha_{ij} h_j$$

The weights  $\alpha_{ij}$  are softmax of  $e_{ij}$  over  $j$ ,

$$\alpha_i = \text{softmax}(e_i),$$

with  $e_{ij}$  being *i := časová známka*

*výběv matice*  
*tohle je jistě  $\alpha[h_j || s_{1:t}]$*   
*konkretizace*

$$e_{ij} = v^T \tanh(Vh_j + Ws_{i-1} + b).$$

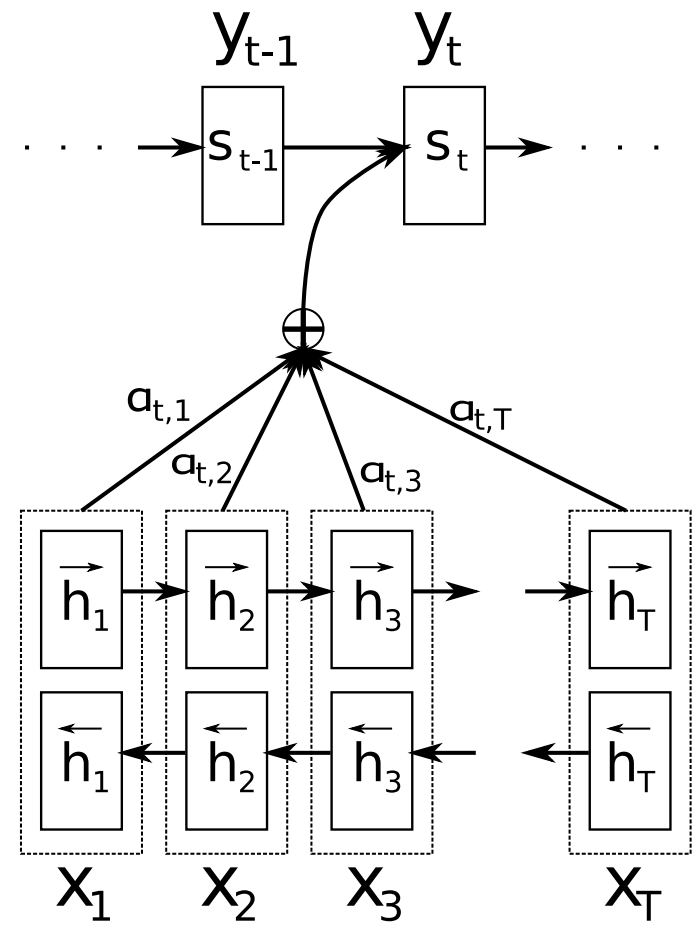


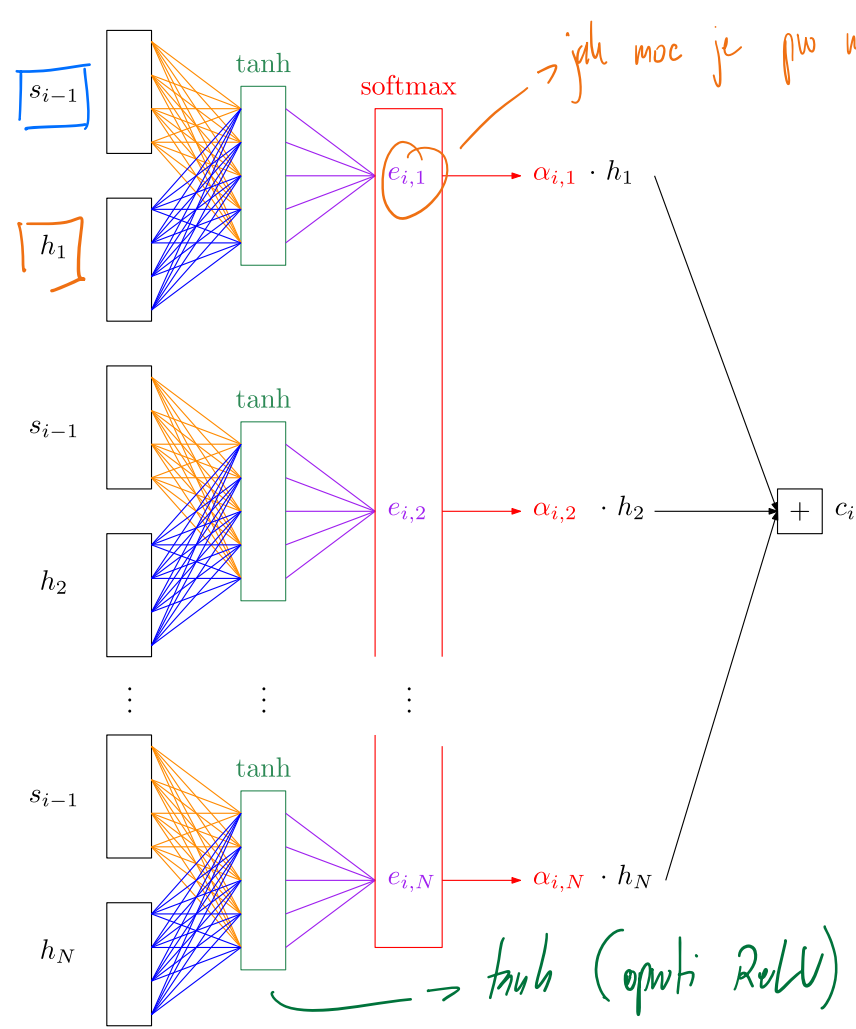
Figure 1 of "Neural Machine Translation by Jointly Learning to Align and Translate", <https://arxiv.org/abs/1409.0473>



# Bahdanau Attention Implementation

aktuální stav

vstupní sekv.



jaka moc je pro mě dané slovo relevantní

Uděl by to bylo více, teď by gradient jednoduše po té největší hodnotě.

To je důvod, proč jsme fthy šklovali 1/255

→ tanh (oproti ReLU) zjistíš, že vstupy mají mezi sebou rozdíl řádově v jednotkách

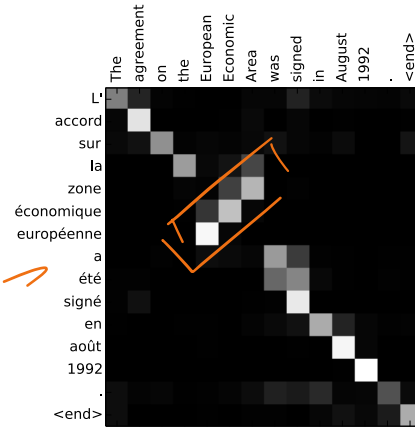


Attention pro EN -> FR překlád

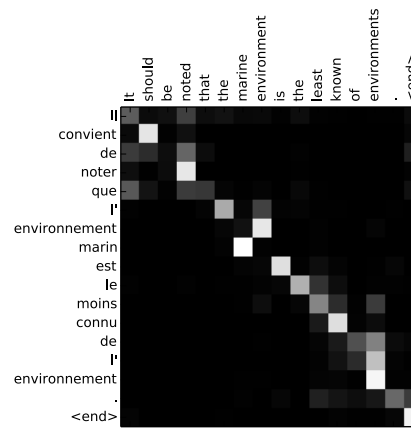
aniž by ta attention  
byla vůbec rozumná

tady je vidět že  
ta francouzštinu má ty  
slova obměněná...

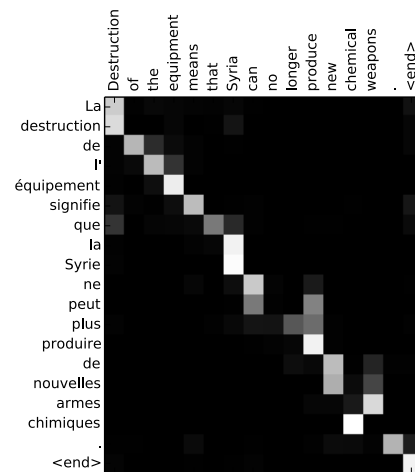
a transformace s tím  
nemá problém...



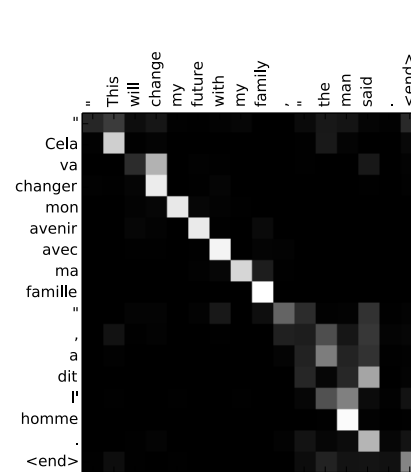
(a)



(b)



(c)



(d)

ukazuje, kam se chce  
hledat, když překládá.

Je zajímavý, jak často se  
může hledat na první jedno  
slovo

Figure 3 of "Neural Machine Translation by Jointly Learning to Align and Translate", <https://arxiv.org/abs/1409.0473>



In the described *Bahdanau* (or *additive*) attention, we performed

$$e_{ij} = \mathbf{v}^\top \tanh(\mathbf{V}\mathbf{h}_j \overset{\text{tohle je to aditivum}}{\oplus} \mathbf{W}\mathbf{s}_{i-1} + \mathbf{b}).$$

There are however other methods how  $\mathbf{V}\mathbf{h}_j$  and  $\mathbf{W}\mathbf{s}_{i-1}$  can be combined, most notably the *Luong* (or *dot-product*) attention, which uses just a dot product:

$$e_{ij} = (\mathbf{V}\mathbf{h}_j)^T (\mathbf{W}\mathbf{s}_{i-1}).$$

The latter is easier to implement, but may sometimes be more difficult to train (scaling helps a bit, wait for the Transformer self-attention description); both approaches are used in quite a few papers.

tohle se trochu hůř trénuje, protože z toho padají divnější hodnoty.  
Výsledky mají docela podobný



# Subword Units

Translate **subword units** instead of words. The subword units can be generated in several ways, the most commonly used are: *- ideálně častý buďen jednotky, nečastý přijde složit z jednotek.*

- **BPE** Using the *byte pair encoding* algorithm. Start with individual characters plus a special end-of-word symbol •. Then, merge the most occurring symbol pair  $A, B$  by a new symbol  $AB$ , with the symbol pair never crossing word boundary (so that the end-of-word symbol cannot be inside a subword).

Considering text with words *low, lowest, newer, wider*, a possible sequence of merges:

*• bude vždycky poslední od slovy, nebudu reprezentovat sub 2 více slov*

*> takže delším, než zuplín svůj slovník. Myslelka jak Huffman encoding, přestože toby je jen "similarity"*

$r \bullet \rightarrow r\bullet$	<i>lowee slova</i>	<i>low •</i>
$l o \rightarrow lo$		<i>lowest •</i>
$lo w \rightarrow low$		<i>newer •</i>
$e r\bullet \rightarrow er\bullet$		<i>wider •</i>

The BPE algorithm is executed on the training data, and it generates the resulting dictionary, merging rules, and training data encoded using this dictionary.



# Subword Units

– takže väčšie mutual-information medzi slovami. Ty s týmto MI pak spojili.

- **Wordpieces:** Given a text divided into subwords, we can compute unigram probability of every subword, and then get the likelihood of the text under a unigram language model by multiplying the probabilities of the subwords in the text.

When we have only a text and a subword dictionary, we divide the text in a greedy fashion, iteratively choosing the longest existing subword.

When constructing the subwords, we again start with individual characters (compared to BPE, we have a *start-of-word* character instead of an *end-of-word* character), and then repeatedly join such a pair of subwords that increases the unigram language model likelihood the most. *sequences of merges must be the same! bahn vs. bahm.* *keďže pátichm* *kojiment slova mezi* *jazyky*

- In the original implementation, the input data were once in a while “repared” (retokenized) in a greedy fashion with the up-to-date dictionary. However, the recent implementations do not seem to do it – but they retokenize the training data with the final dictionary, contrary to the BPE approach. *> i treba po častom to stroj. A sčítá sa ten* *prostor pre sta překládání jazyky.*

For both approaches, usually quite little subword units are used (32k-64k), often generated on the union of the two vocabularies of the source and target languages (the so-called *joint BPE* or *shared wordpieces*). *Protože existují slova, které se nemají překládat*



Both the BPE and the WordPieces give very similar results; the biggest difference is that during the inference:

- for BPE, the sequence of merges must be performed in the same order as during the construction of the BPE (because we use the output of BPE as training data),
- for Wordpieces, it is enough to find longest matches from the subword dictionary (because we reprocessed the training data with the final dictionary);
- note that the above difference is mostly artificial – if we reparsed the training data in the BPE approach, we could also perform “greedy tokenization”.

Of course, the two algorithms also differ in the way how they choose the pair of subwords to merge.

Both algorithms are implemented in quite a few libraries, most notably the `sentencepiece` library and the Hugging Face `tokenizers` package.



# Google NMT — první zajímavý model google translate

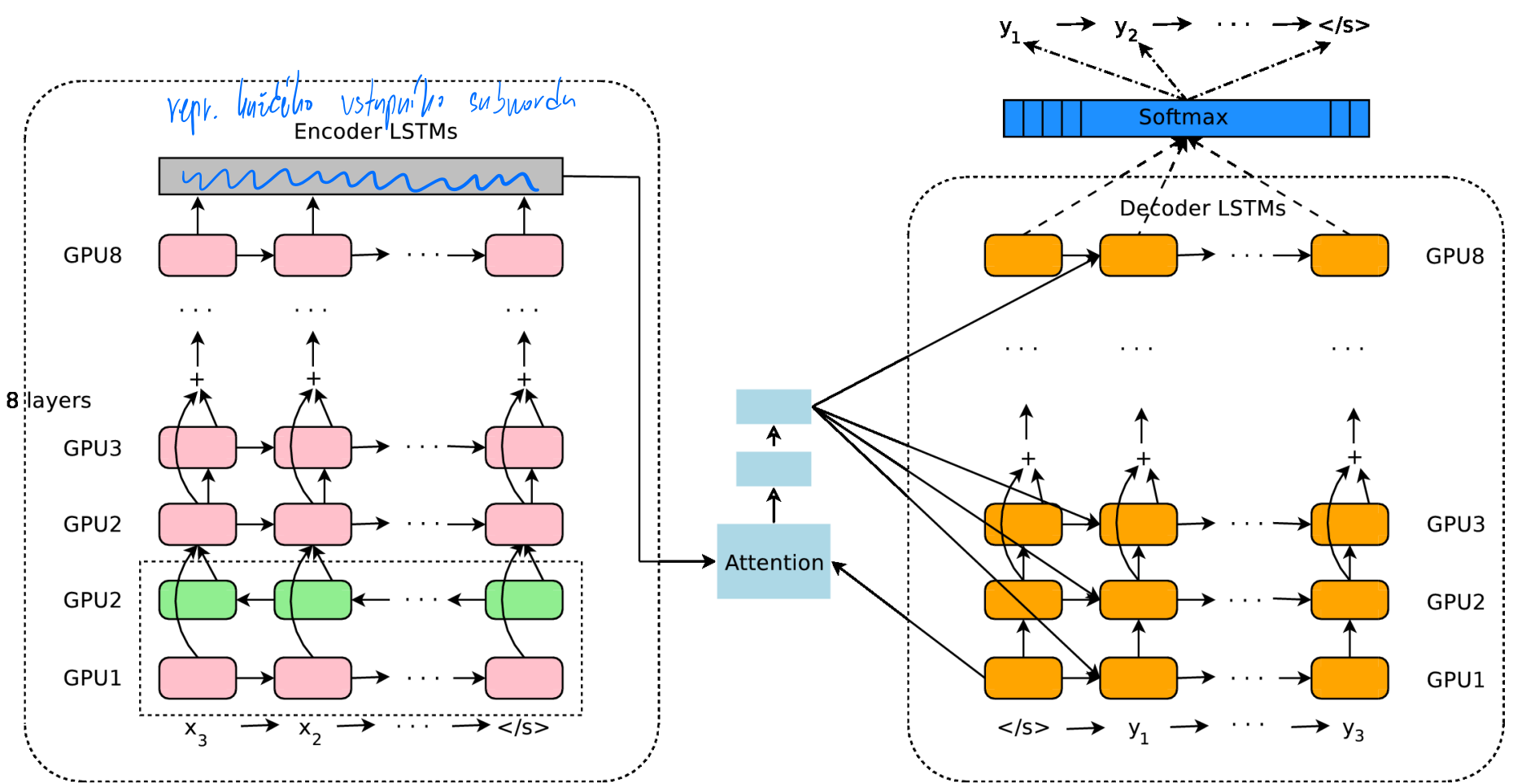
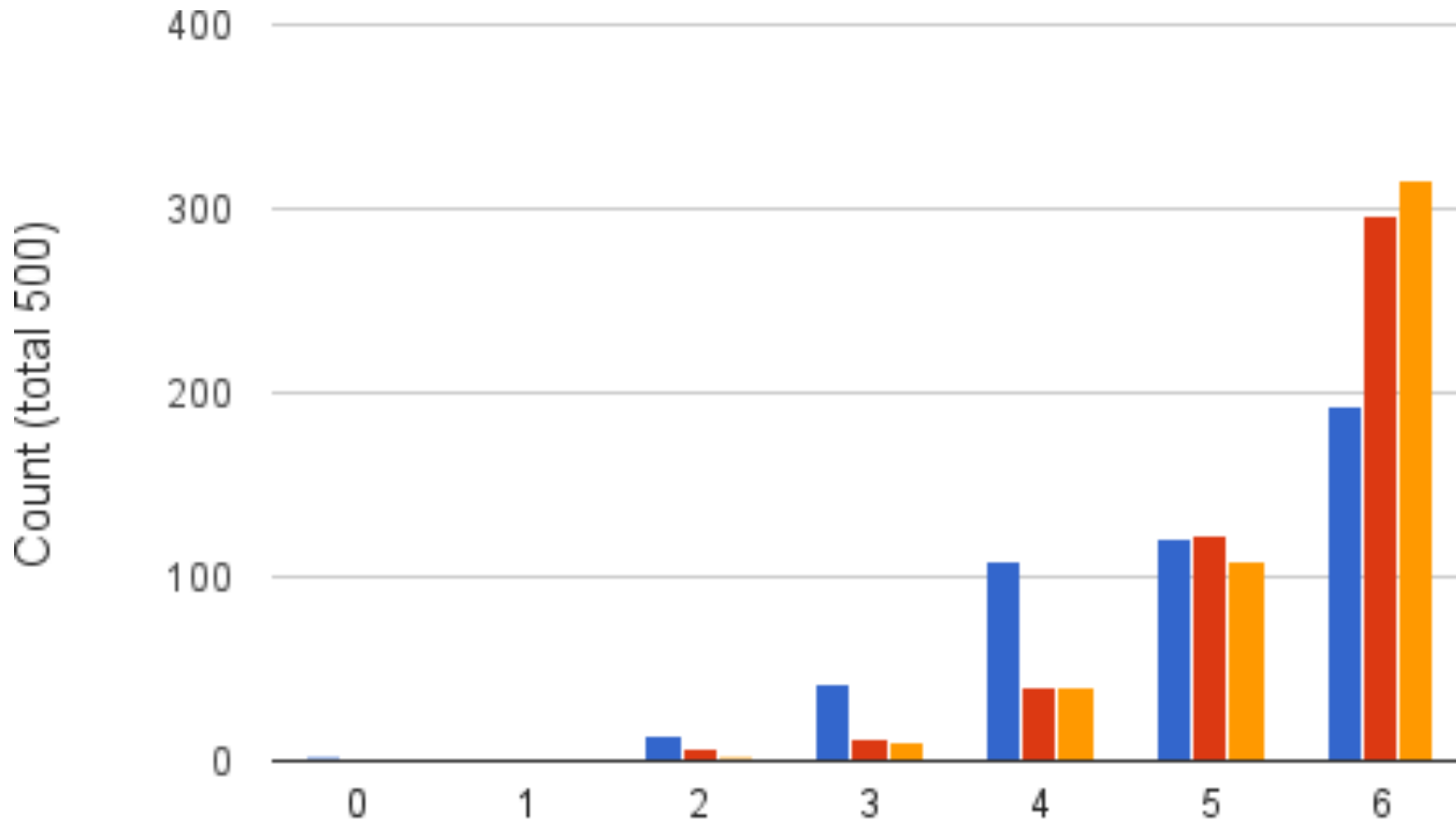


Figure 1 of "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", <https://arxiv.org/abs/1609.08144>





PBMT - GNMT - Human

Figure 6 of "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", <https://arxiv.org/abs/1609.08144>



# Beyond one Language Pair

decoder →  
encoder →



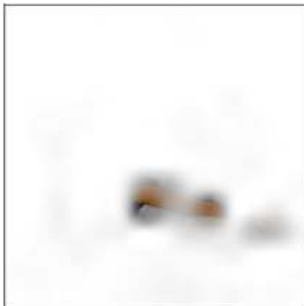
Fig. 5. A selection of evaluation results, grouped by human rating.

Figure 5 of "Show and Tell: Lessons learned from the 2015 MSCOCO...", <https://arxiv.org/abs/1609.06647>



# Beyond one Language Pair

attention  
wie abzeichnen



What vegetable is the dog  
chewing on?

MCB: carrot

GT: carrot



What kind of dog is this?

MCB: husky

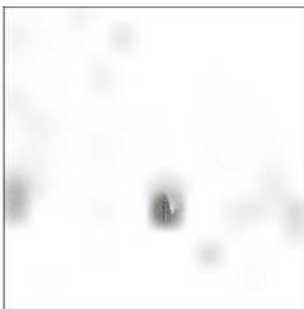
GT: husky



What kind of flooring does  
the room have?

MCB: carpet

GT: carpet



What color is the traffic  
light?

MCB: green

GT: green



Is this an urban area?

MCB: yes

GT: yes



Where are the buildings?

MCB: in background

GT: on left

Figure 6 of "Multimodal Compact Bilinear Pooling for VQA and Visual Grounding", <https://arxiv.org/abs/1606.01847>



# Multilingual and Unsupervised Translation

Many attempts at multilingual translation.

→ proto čím bližší jazyky, tým lepší

- Individual encoders and decoders, **shared attention**.
- Shared encoders and decoders.

Surprisingly, even unsupervised translation is attempted lately. By unsupervised we understand settings where we have access to large monolingual corpora, but no parallel data.

In 2019, the best unsupervised systems were on par with the best 2014 supervised systems.

		WMT-14			
		fr-en	en-fr	de-en	en-de
Unsupervised	Proposed system	33.5	36.2	27.0	22.5
	<i>detok. SacreBLEU*</i>	33.2	33.6	26.4	21.2
Supervised	WMT best*	35.0	35.8	29.0	20.6 <sup>†</sup>
	Vaswani et al. (2017)	-	41.0	-	28.4
	Edunov et al. (2018)	-	45.6	-	35.0

a trénuje se  
na každý pár

Table 3: Results of the proposed method in comparison to different supervised systems (BLEU).

Table 3 of "An Effective Approach to Unsupervised Machine Translation", <https://arxiv.org/abs/1902.01313>

Reálně ale stačí jeden velký model, který umí encodeovat všechny jazyky. Naproti Microsoft přistupuje přes pár jazyků.



Existují i přístupy, kdy se třebaje uspořádat bez pravidelného textu  
a pouze se speciální (ortogonální) transformací wordů prostor  
tak, aby dostal stejná slova v cílovém jazyce.



For some sequence processing tasks, *sequential* processing (as performed by recurrent neural networks) of its elements might be too restrictive.

Instead, we may want to be able to combine sequence elements independently on their distance.

Such processing is allowed in the **Transformer** architecture, originally proposed for neural machine translation in 2017 in *Attention is All You Need* paper.

*Celé se to vymýšlelo, aby som sa zbavil sekvencového zapamätania*  $\circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \dots$



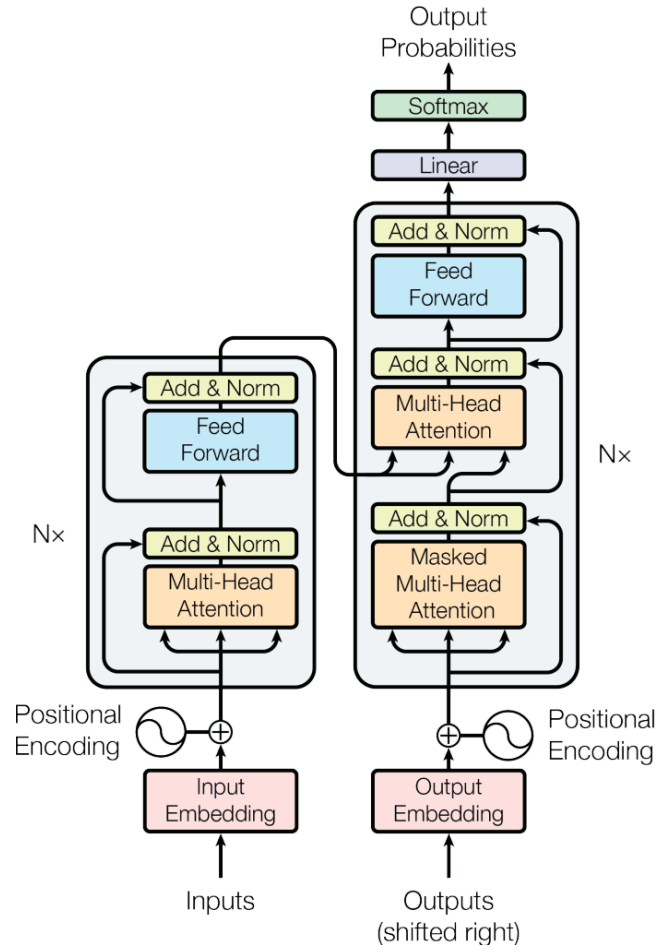
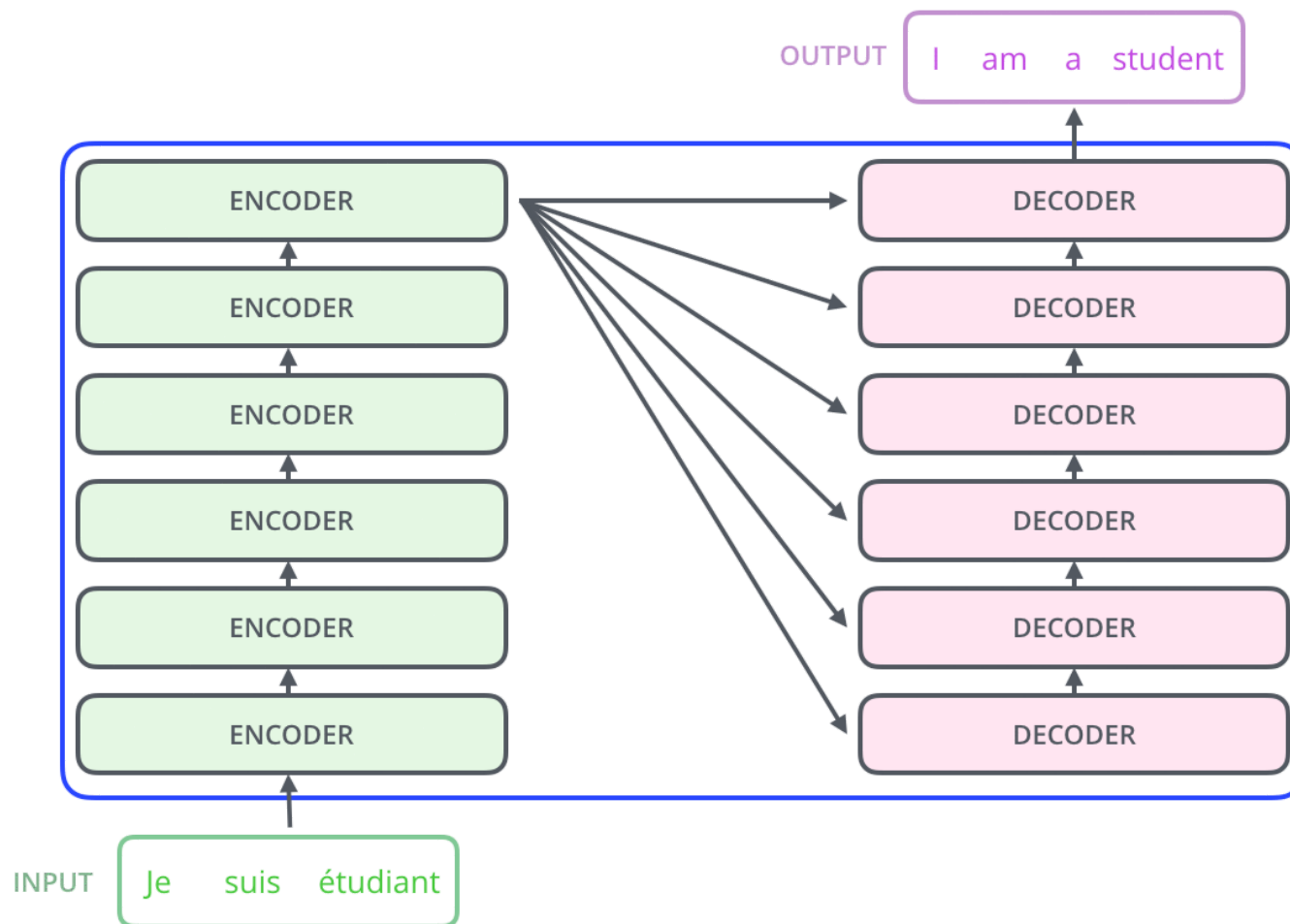


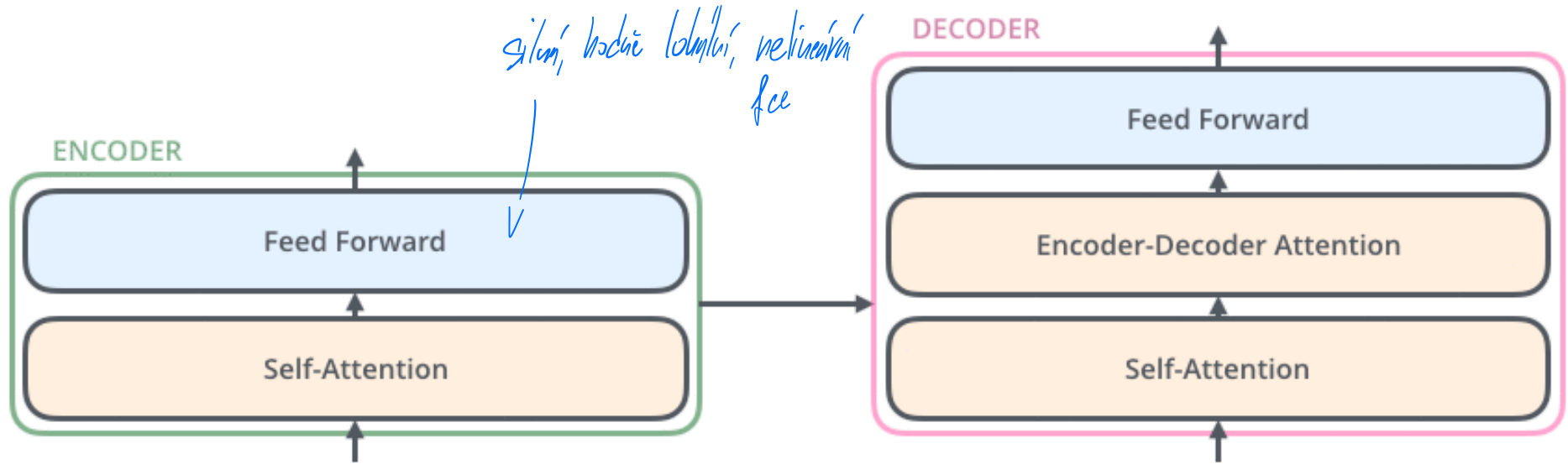
Figure 1 of "Attention Is All You Need", <https://arxiv.org/abs/1706.03762>





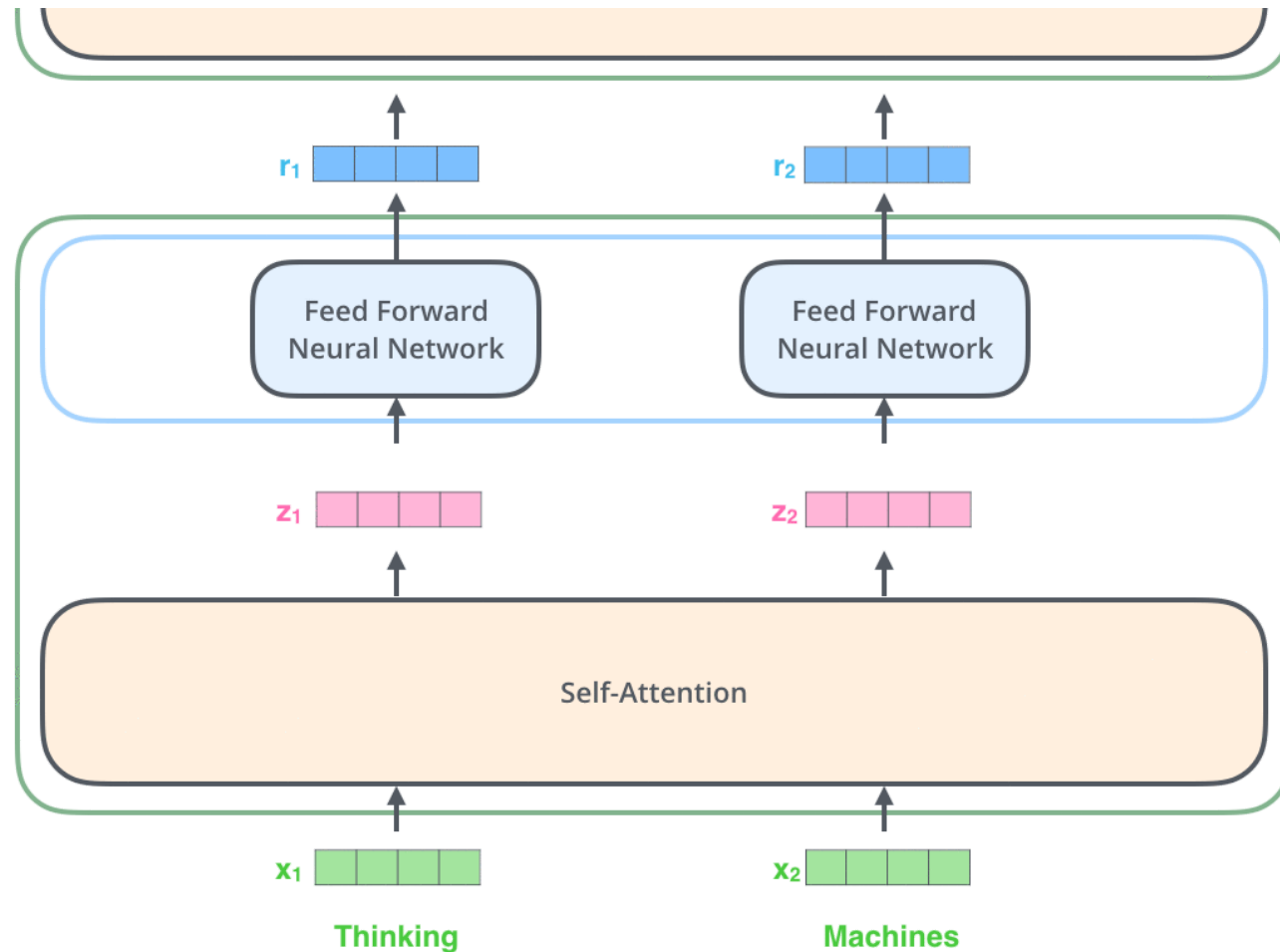
[http://jalammar.github.io/images/t/The\\_transformer\\_encoder\\_decoder\\_stack.png](http://jalammar.github.io/images/t/The_transformer_encoder_decoder_stack.png)





[http://jalammar.github.io/images/t/Transformer\\_decoder.png](http://jalammar.github.io/images/t/Transformer_decoder.png)





*obří hroužence* →

[http://jalamar.github.io/images/t/encoder\\_with\\_tensors\\_2.png](http://jalamar.github.io/images/t/encoder_with_tensors_2.png)



Assume that we have a sequence of  $n$  words represented using a matrix  $\mathbf{X} \in \mathbb{R}^{n \times d}$ .

The attention module for queries  $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$ , keys  $\mathbf{K} \in \mathbb{R}^{n \times d_k}$  and values  $\mathbf{V} \in \mathbb{R}^{n \times d_v}$  is defined as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} \right) \mathbf{V}.$$

*to jak me spolu funguje*  
*kombinuji se obkrmady skalárním součinem*  
*Chápej jako logiky všech dvojic*  
*softmax děláme po řádcích*  
 $q^T h = \cos \varphi \cdot \|q\| \cdot \|h\|$

The queries, keys and values are computed from the input word representations  $\mathbf{X}$  using a linear transformation as

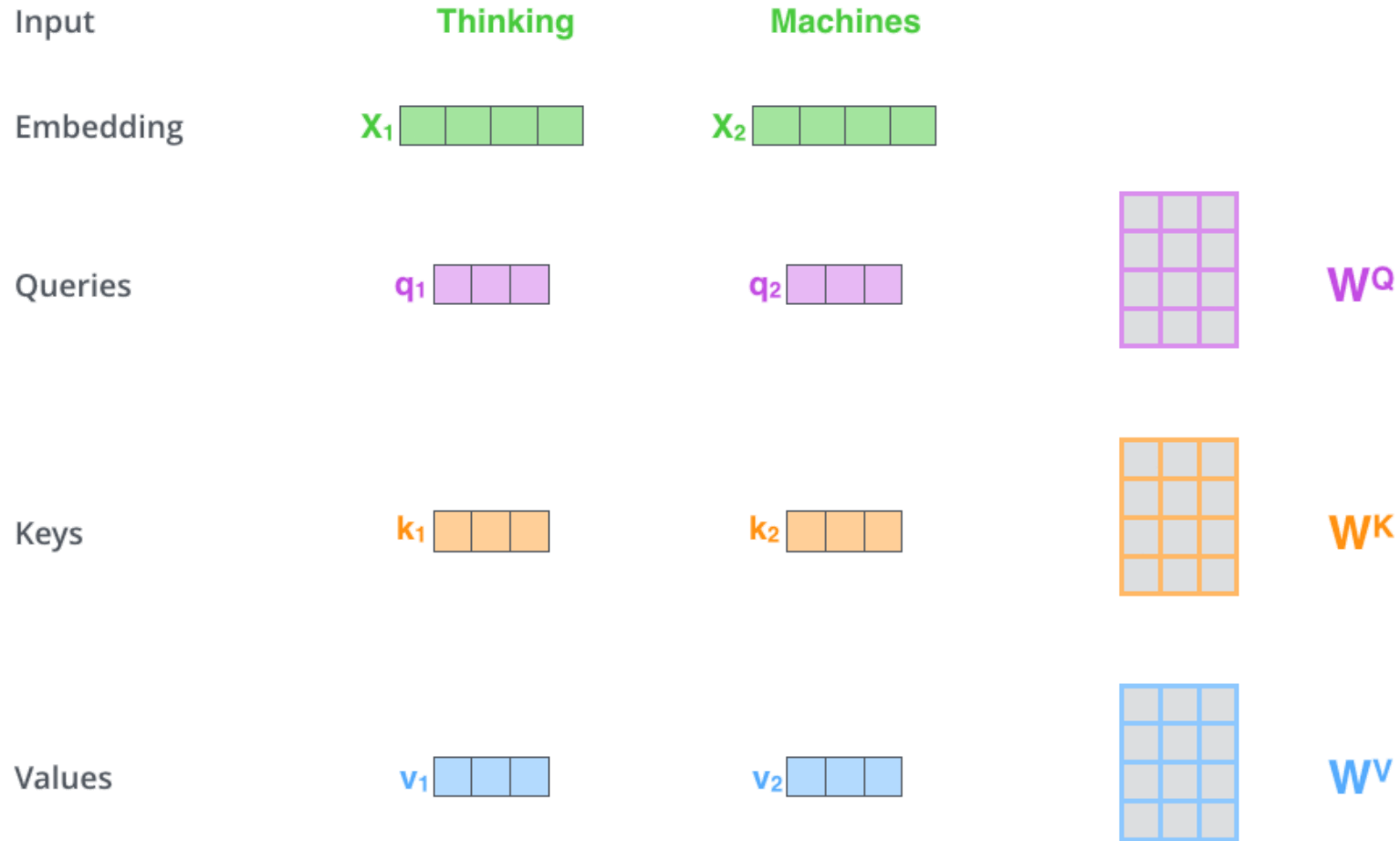
*Self-attention, protože předtím slovo attendovalo, teď attenduje jednotlivé slova (sub-slova)*

$$\begin{aligned} \mathbf{Q} &= \mathbf{X}\mathbf{W}^Q \rightarrow \text{každý slovo říká, co hledám} \\ \mathbf{K} &= \mathbf{X}\mathbf{W}^K \rightarrow \text{každý slovo říká, co je} \\ \mathbf{V} &= \mathbf{X}\mathbf{W}^V \rightarrow \text{každý slovo říká, co to je za hodnota} \end{aligned}$$

*je to všechno lin. fun. vstupů*

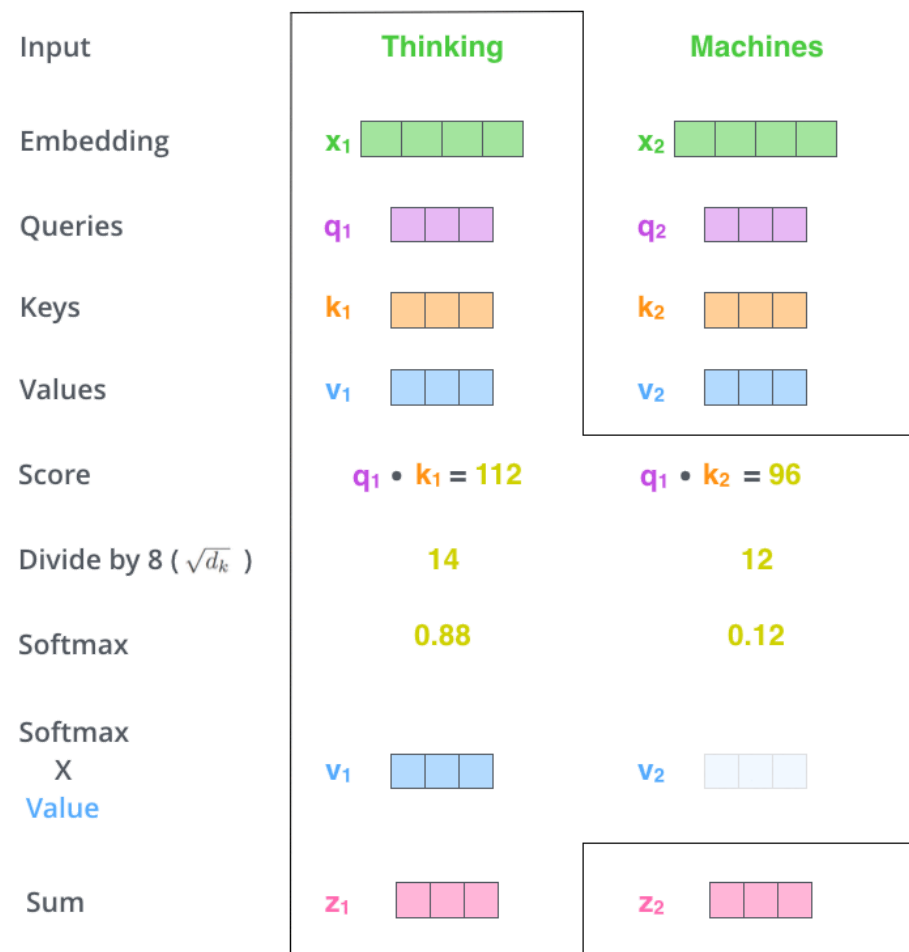
for trainable weight matrices  $\mathbf{W}^Q, \mathbf{W}^K \in \mathbb{R}^{d \times d_k}$  and  $\mathbf{W}^V \in \mathbb{R}^{d \times d_v}$ .





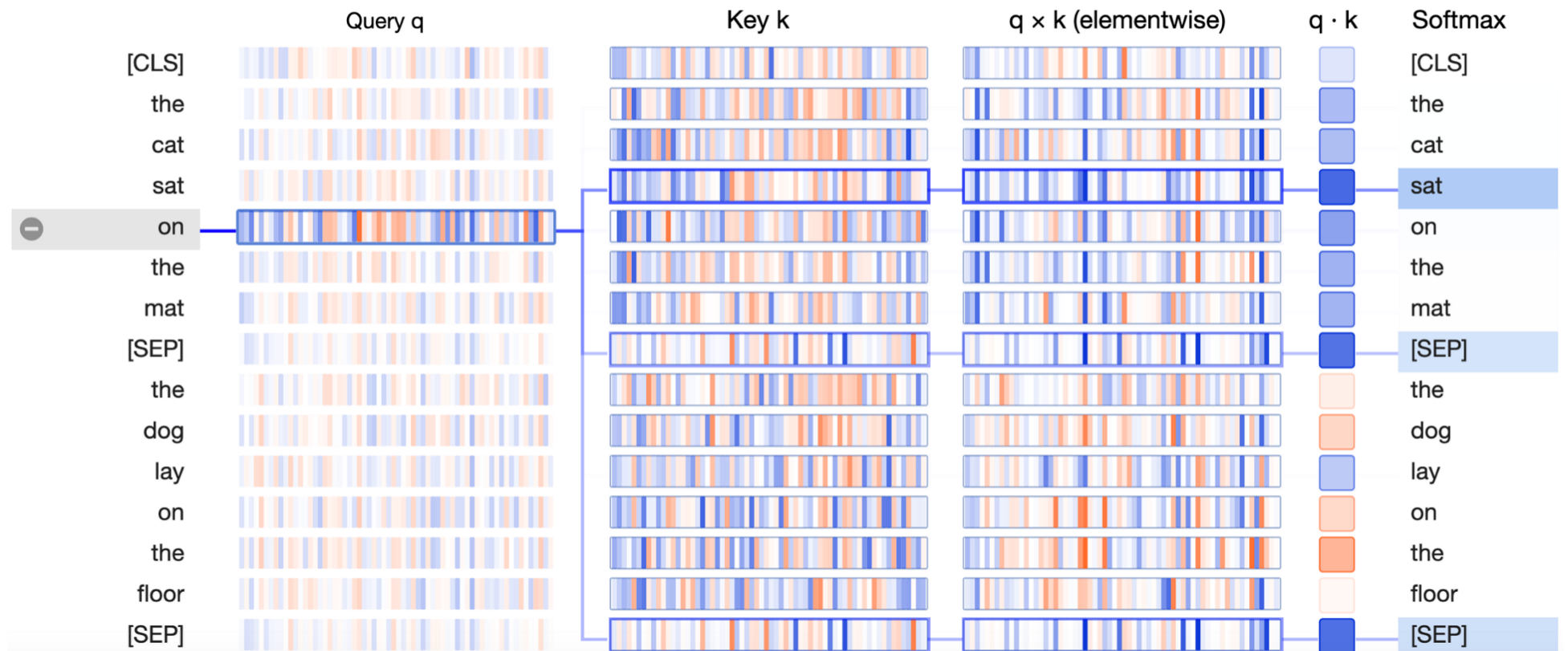
[http://jalammar.github.io/images/t/transformer\\_self\\_attention\\_vectors.png](http://jalammar.github.io/images/t/transformer_self_attention_vectors.png)





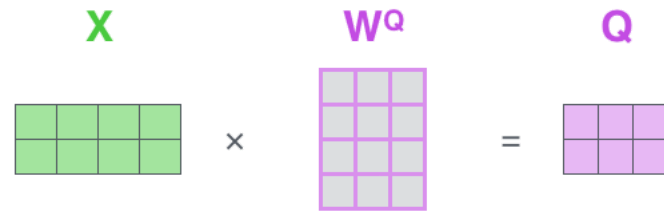
<http://jalamar.github.io/images/t/self-attention-output.png>

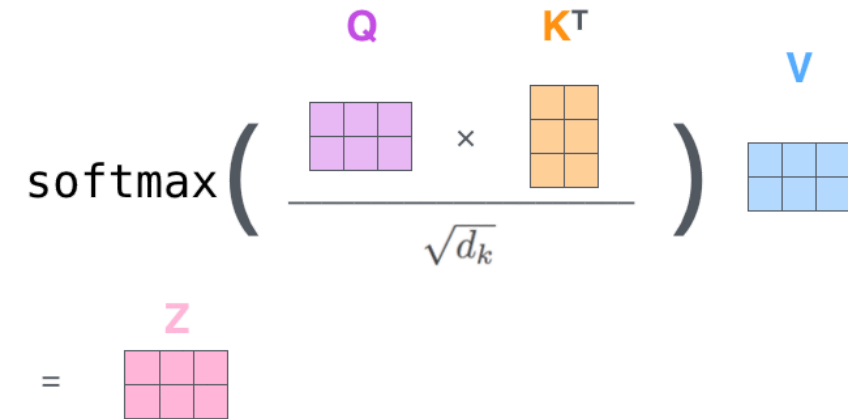




[https://miro.medium.com/max/2000/1\\*jBsFVNOOcJ-I3tsLVgni\\_w.png](https://miro.medium.com/max/2000/1*jBsFVNOOcJ-I3tsLVgni_w.png)





$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) \times V = Z$$


<http://jalammar.github.io/images/t/self-attention-matrix-calculation-2.png>

<http://jalammar.github.io/images/t/self-attention-matrix-calculation.png>

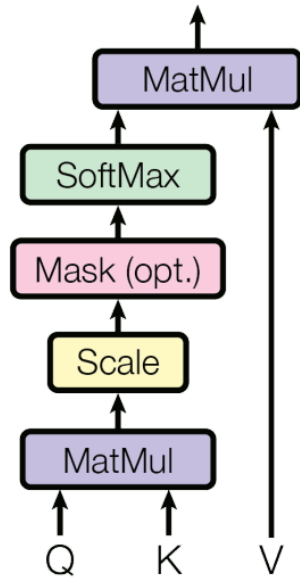


# Transformer – Multihead Attention

Multihead attention is used in practice. Instead of using one huge attention, we split queries, keys and values to several groups (similar to how ResNeXt works), compute the attention in each of the groups separately, concatenate the results and multiply them by a matrix  $W^O$ .

Scaled Dot-Product Attention

- Často mě zajímá  
více queries kromě  
jedné?  
„zajímá mě pohled  
vpředu, vzadu...“



Abych se z toho  
lokálního feature  
spíše dostal zpět  
do globálního

Multi-Head Attention

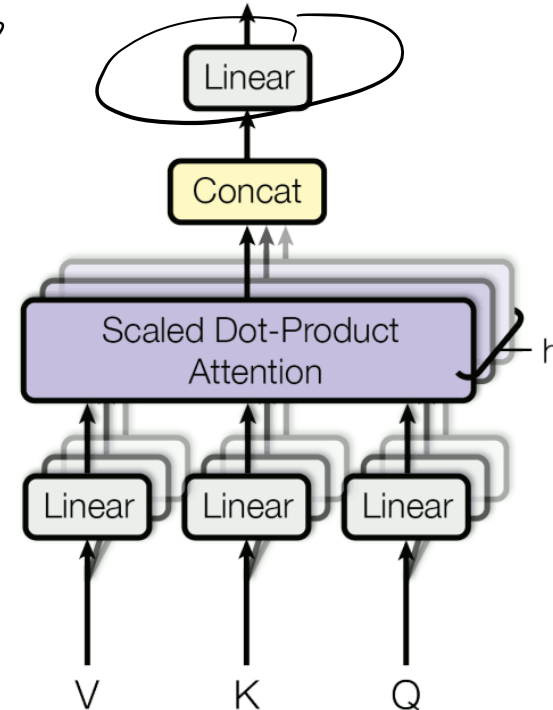
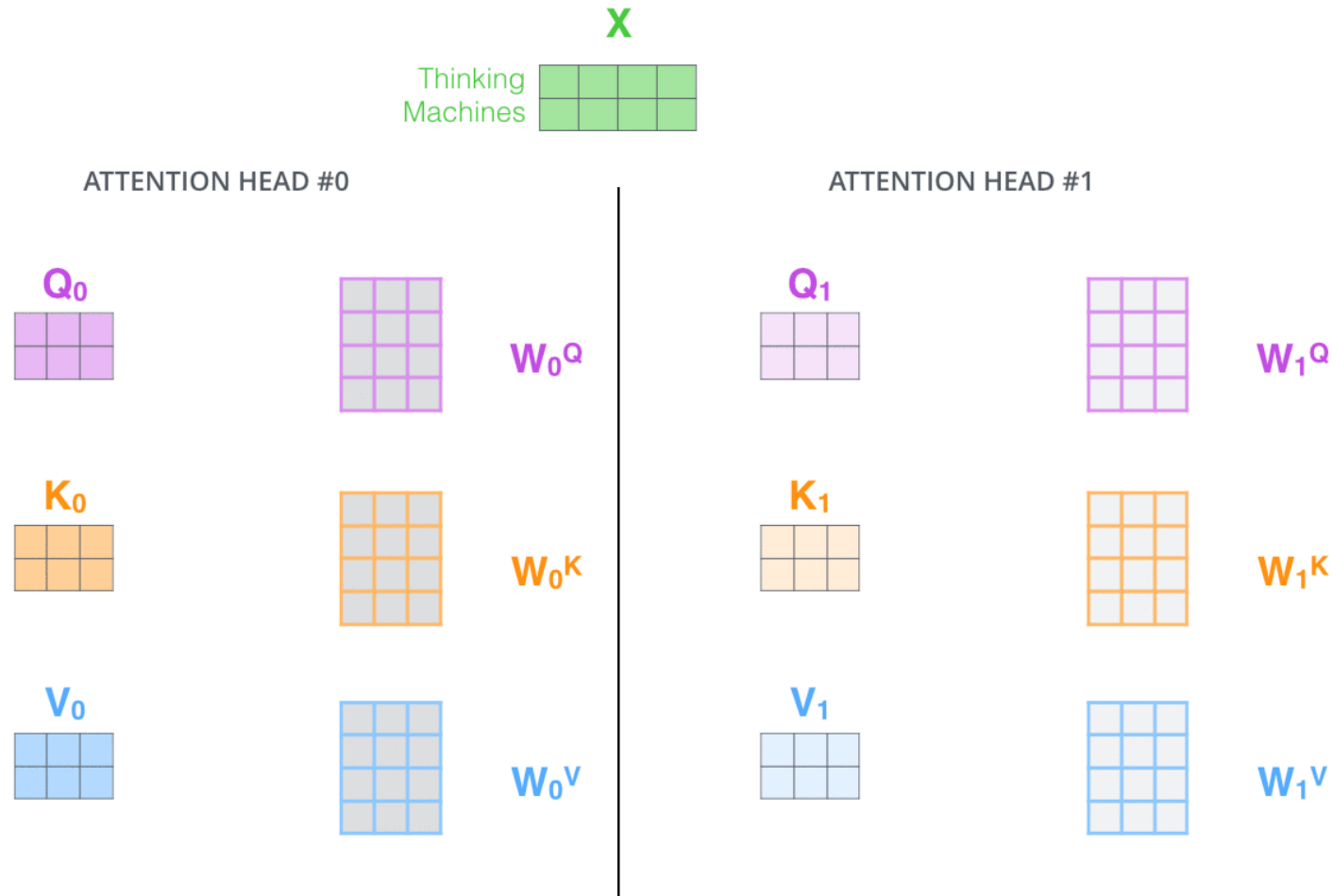


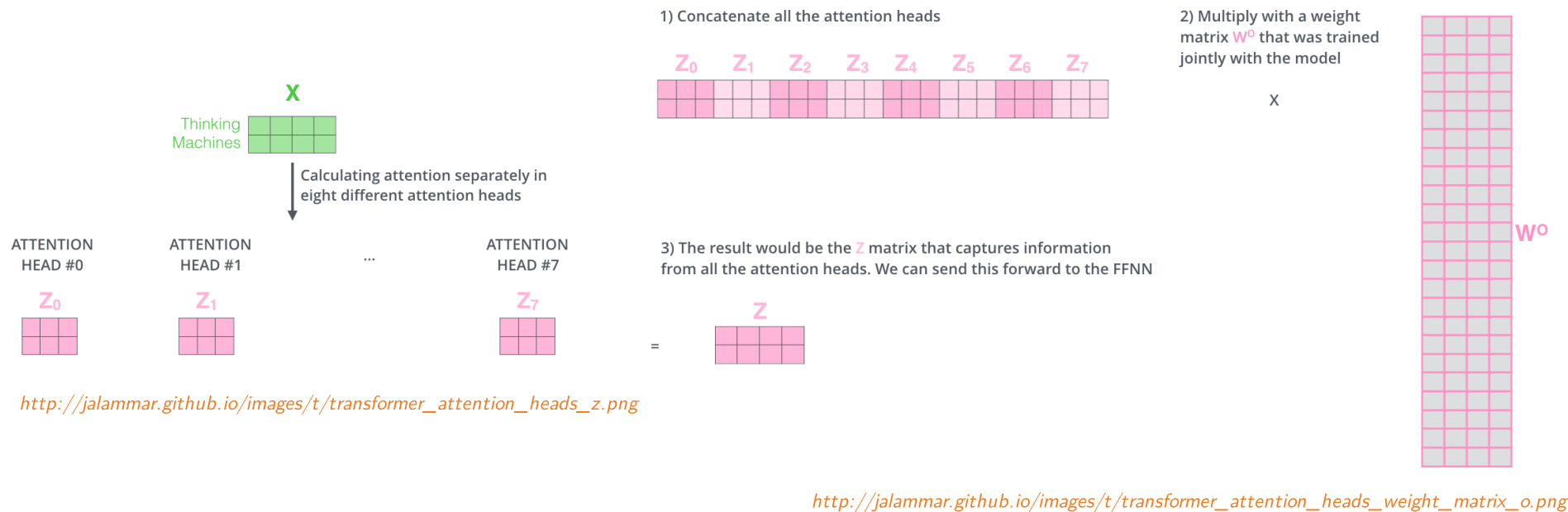
Figure 2 of "Attention Is All You Need", <https://arxiv.org/abs/1706.03762>





[http://jalammar.github.io/images/t/transformer\\_attention\\_heads\\_qkv.png](http://jalammar.github.io/images/t/transformer_attention_heads_qkv.png)







# Transformer – Multihead Attention

1) This is our input sentence\*

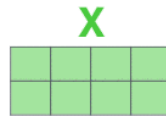
2) We embed each word\*

3) Split into 8 heads. We multiply  $X$  or  $R$  with weight matrices

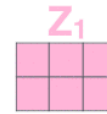
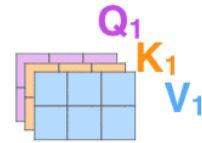
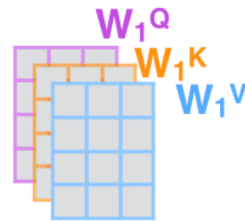
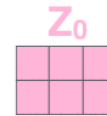
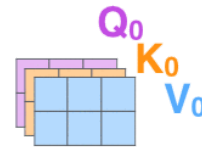
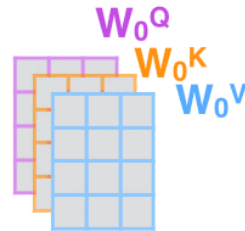
4) Calculate attention using the resulting  $Q/K/V$  matrices

5) Concatenate the resulting  $Z$  matrices, then multiply with weight matrix  $W^O$  to produce the output of the layer

Thinking  
Machines



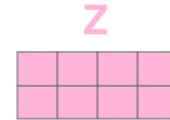
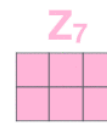
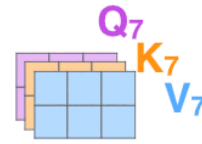
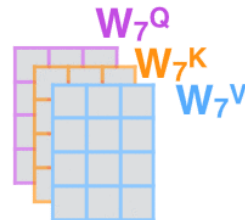
\* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



...

...

...



[http://jalammar.github.io/images/t/transformer\\_multi-headed\\_self-attention-recap.png](http://jalammar.github.io/images/t/transformer_multi-headed_self-attention-recap.png)



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

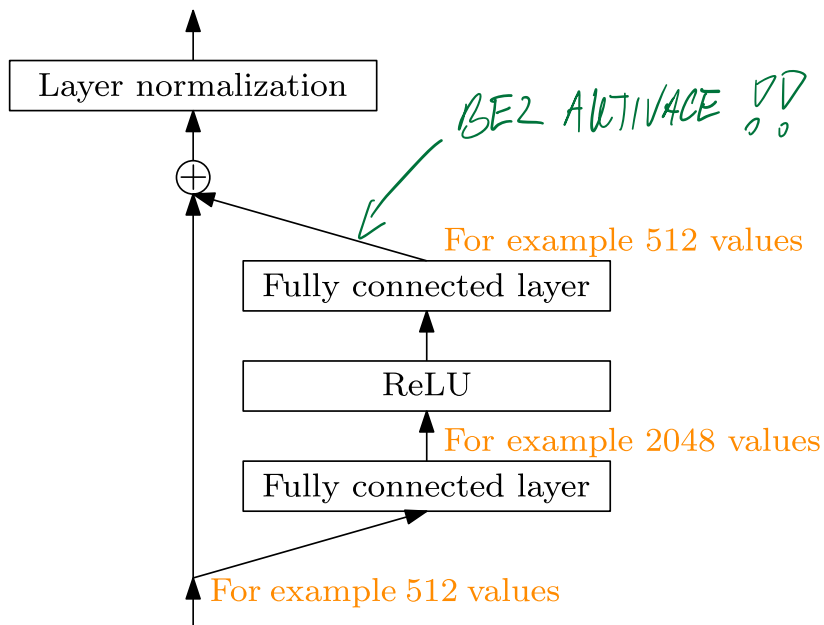
Table 1 of "Attention Is All You Need", <https://arxiv.org/abs/1706.03762>



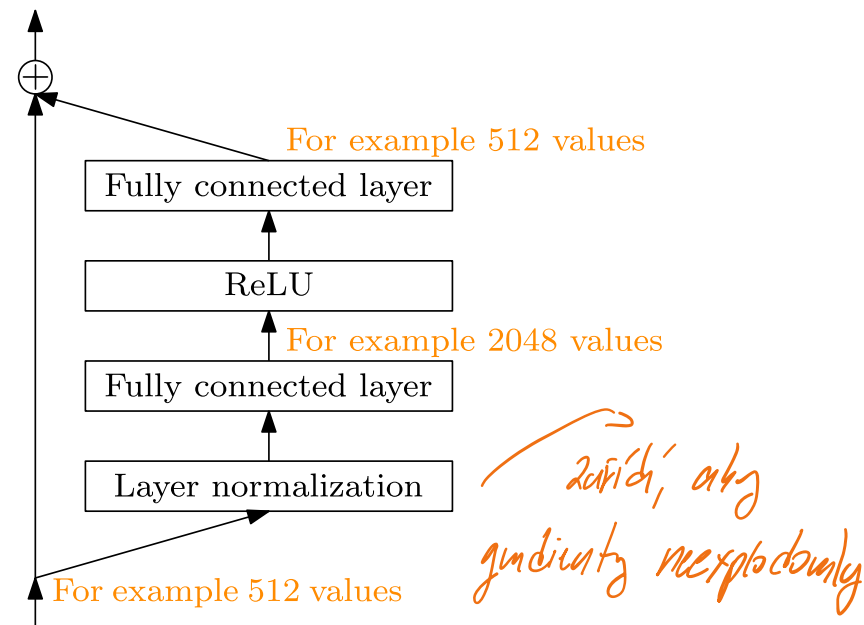
## Feed Forward Networks

The self-attention is complemented with FFN layers, which is a fully connected ReLU layer with four times as many hidden units as inputs, followed by another fully connected layer without activation.

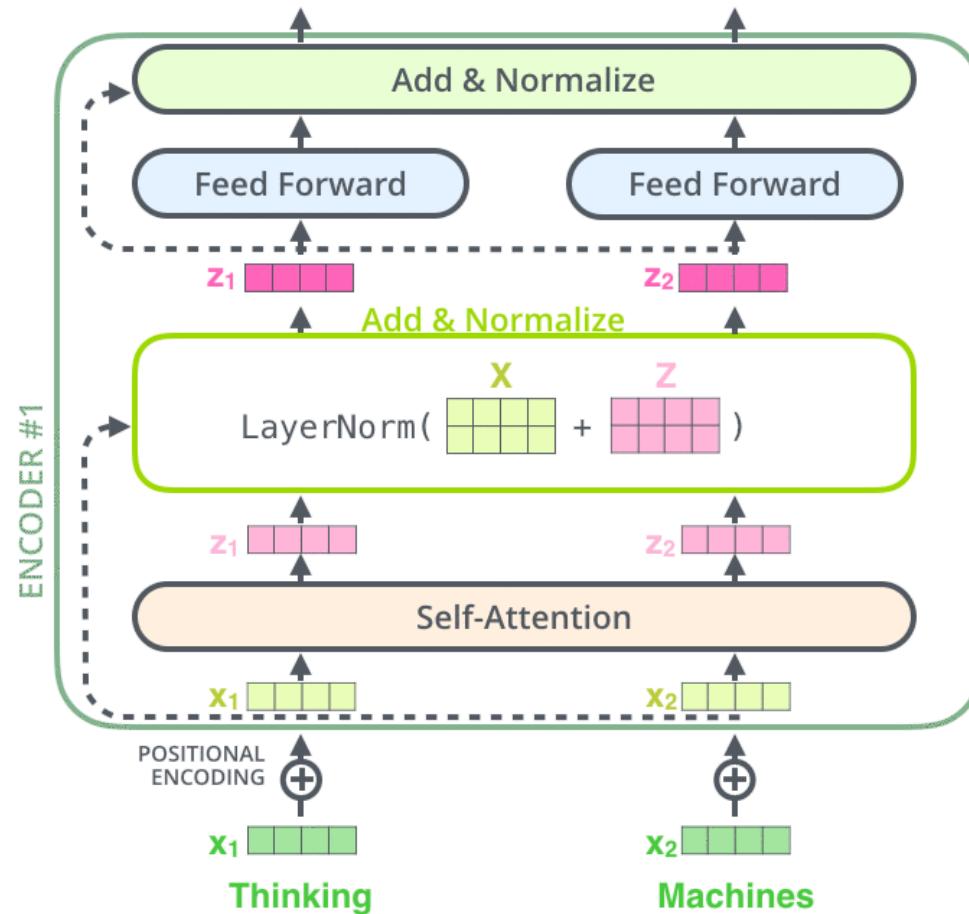
Original “Post-LN” configuration



Improved “Pre-LN“ configuration since 2020



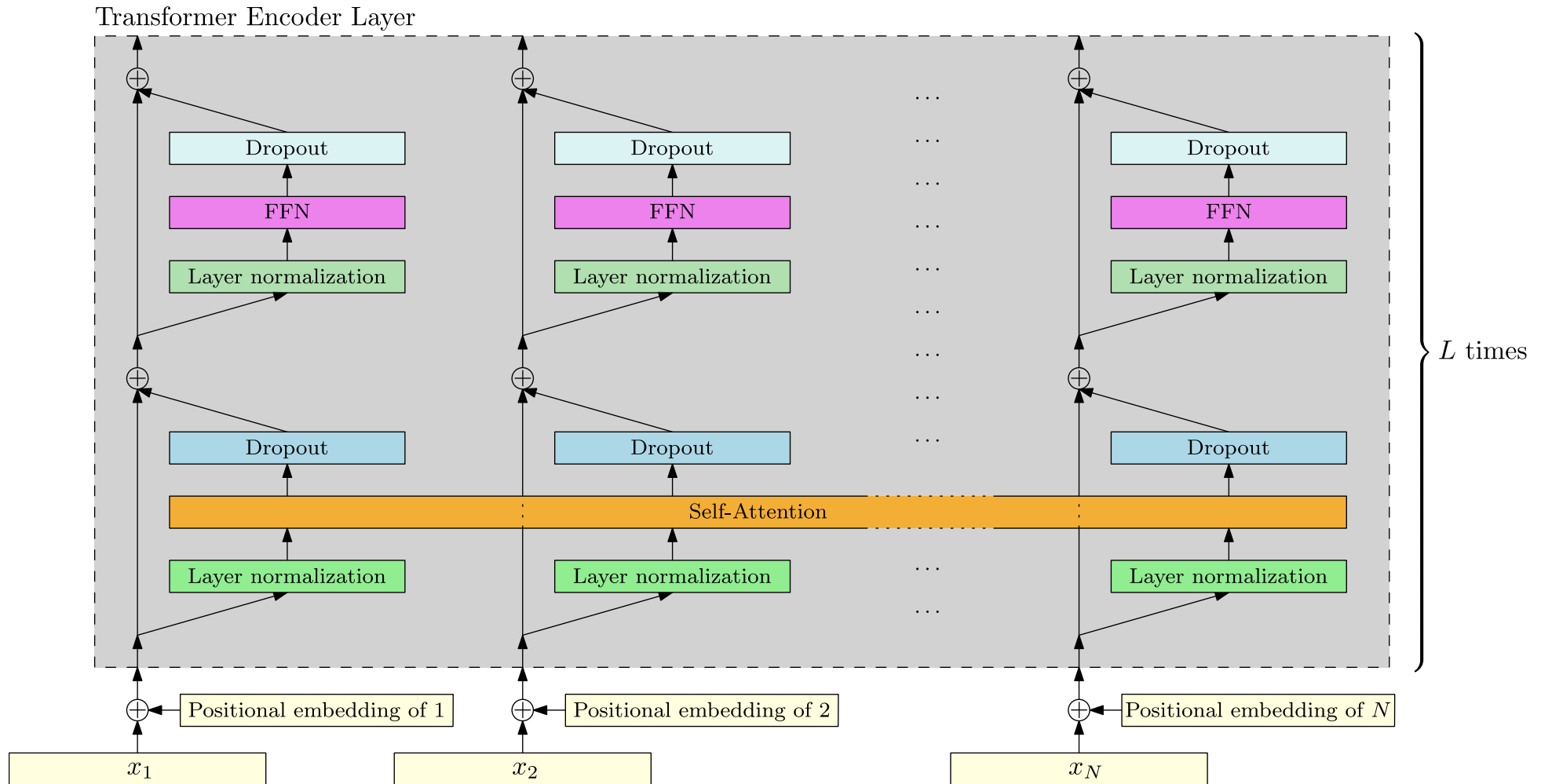




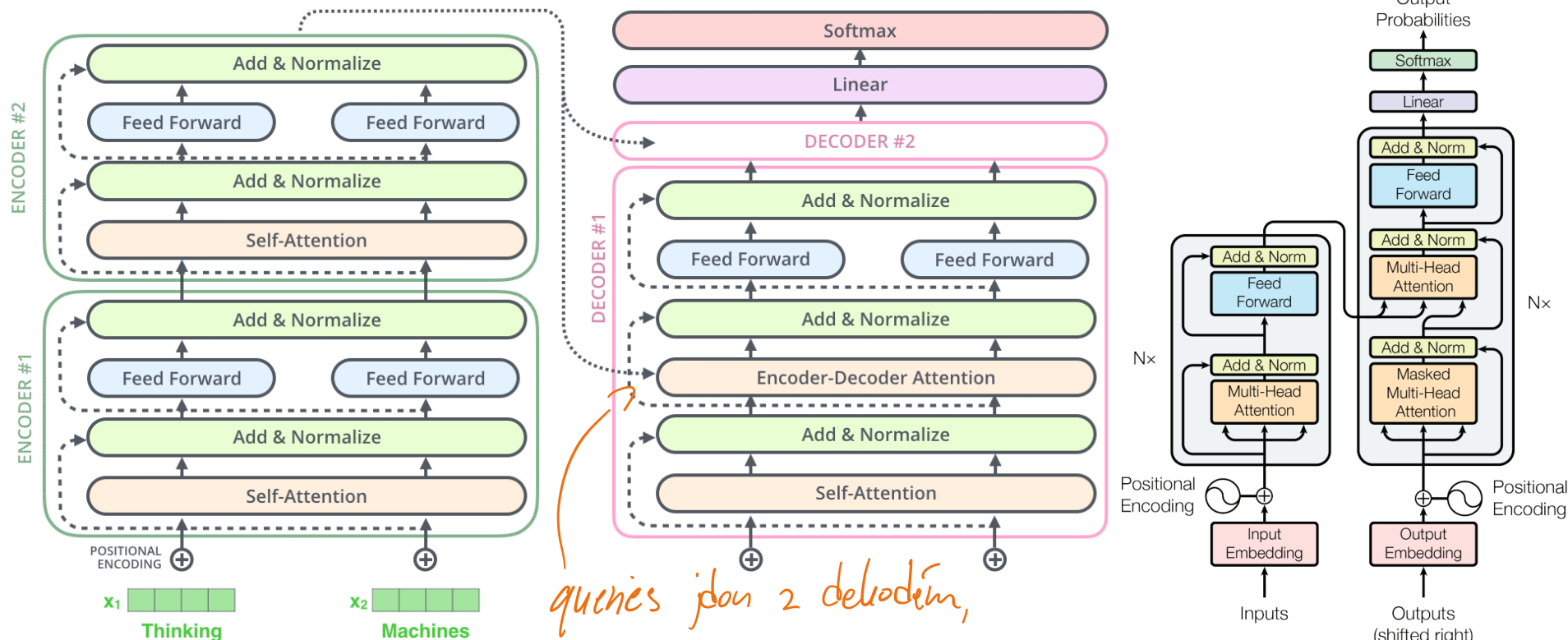
[http://jalammar.github.io/images/t/transformer\\_residual\\_layer\\_norm\\_2.png](http://jalammar.github.io/images/t/transformer_residual_layer_norm_2.png)



# Transformer – Pre-LN Configuration







[http://jalamar.github.io/images/t/transformer\\_residual\\_layer\\_norm\\_3.png](http://jalamar.github.io/images/t/transformer_residual_layer_norm_3.png)

Figure 1 of "Attention Is All You Need",  
<https://arxiv.org/abs/1706.03762>

queries jdon 2 dekodém,  
 h,v jdon 2 enkodém



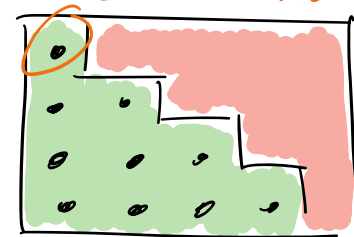
Attention nikdy nemůže uvažovat do budoucnosti !

## Masked Self-Attention

During decoding, the self-attention must attend only to earlier positions in the output sequence.

This is achieved by **masking** future positions, i.e., zeroing their weights out, which is usually implemented by setting them to  $-\infty$  before the softmax calculation.

slow nemůže vidět scan budoucnost upředu.



## Encoder-Decoder Attention

In the encoder-decoder attentions, the *queries* comes from the decoder, while the *keys* and the *values* originate from the encoder.

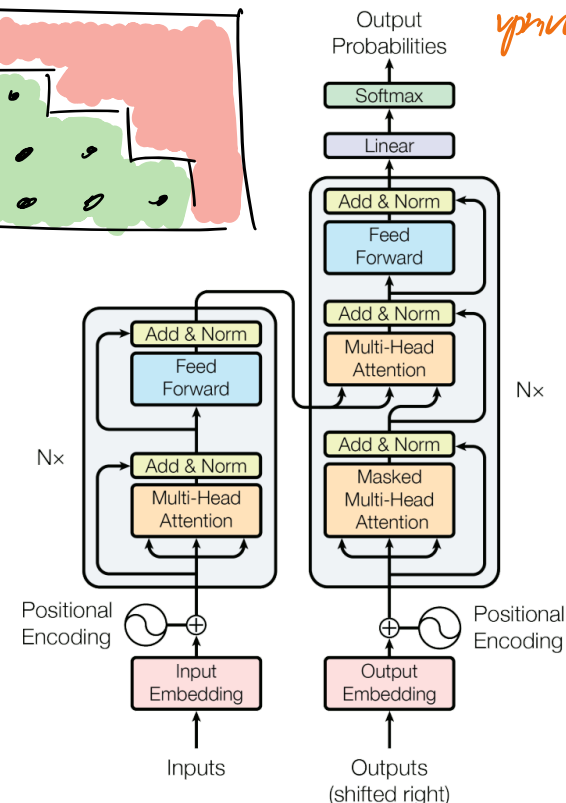
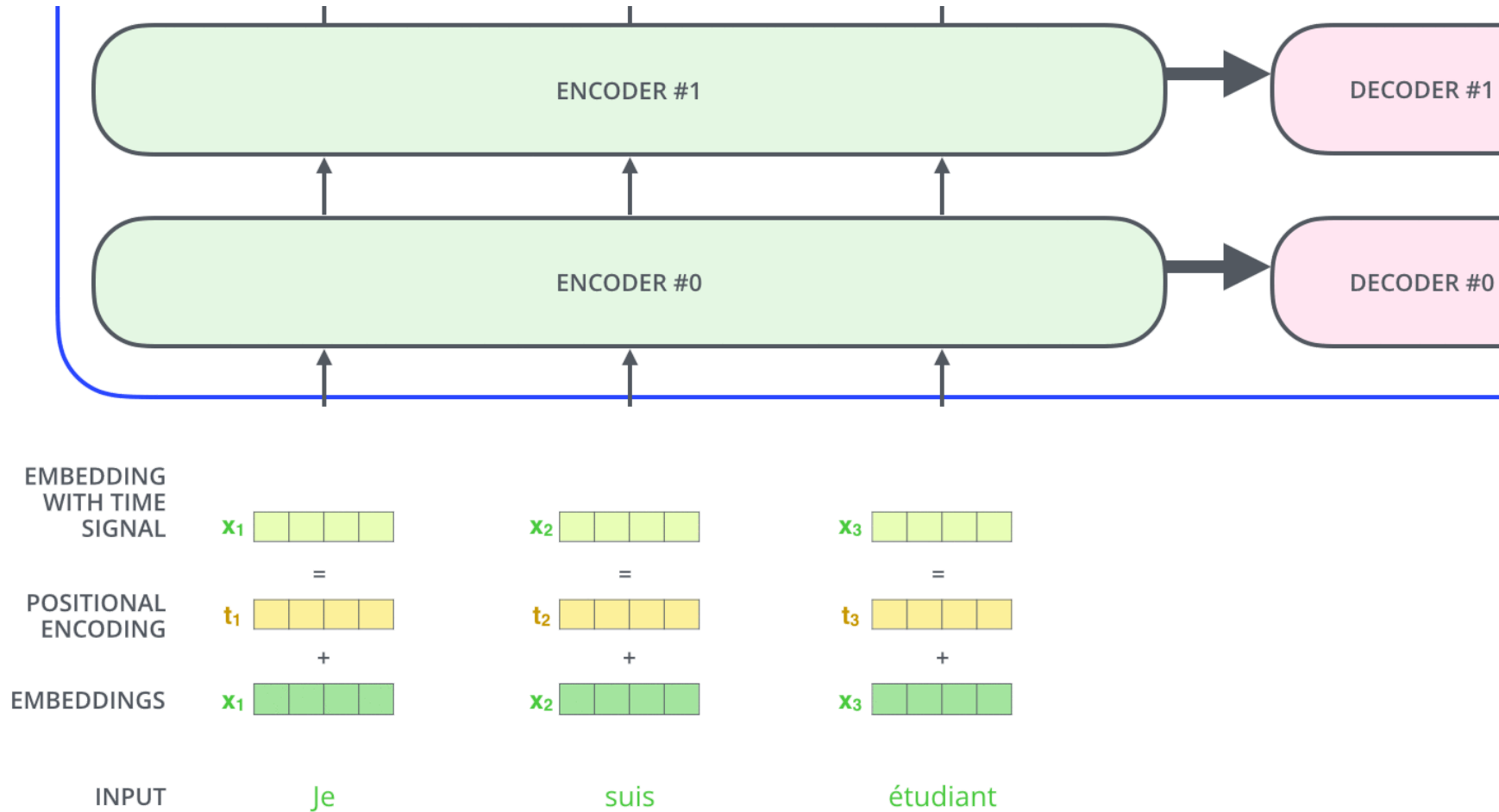


Figure 1 of "Attention Is All You Need", <https://arxiv.org/abs/1706.03762>



# Transformer – Positional Embedding



[http://jalamar.github.io/images/t/transformer\\_positional\\_encoding\\_vectors.png](http://jalamar.github.io/images/t/transformer_positional_encoding_vectors.png)



## Positional Embeddings

We need to encode positional information (which was implicit in RNNs).

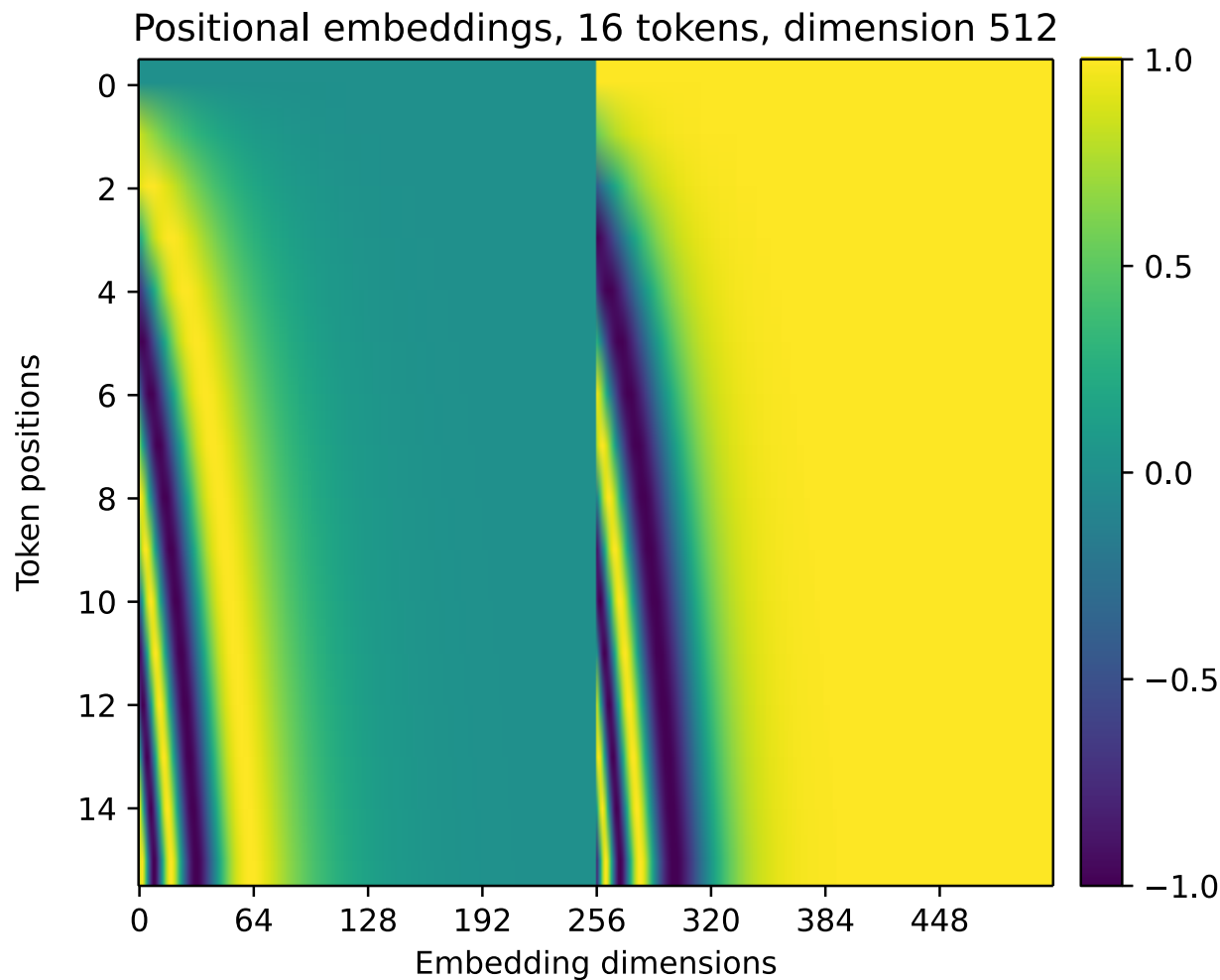
- Learned embeddings for every position.
- Sinusoids of different frequencies:

$$\begin{aligned}\text{PE}_{(pos, 2i)} &= \sin(pos/10000^{2i/d}) \\ \text{PE}_{(pos, 2i+1)} &= \cos(pos/10000^{2i/d})\end{aligned}$$

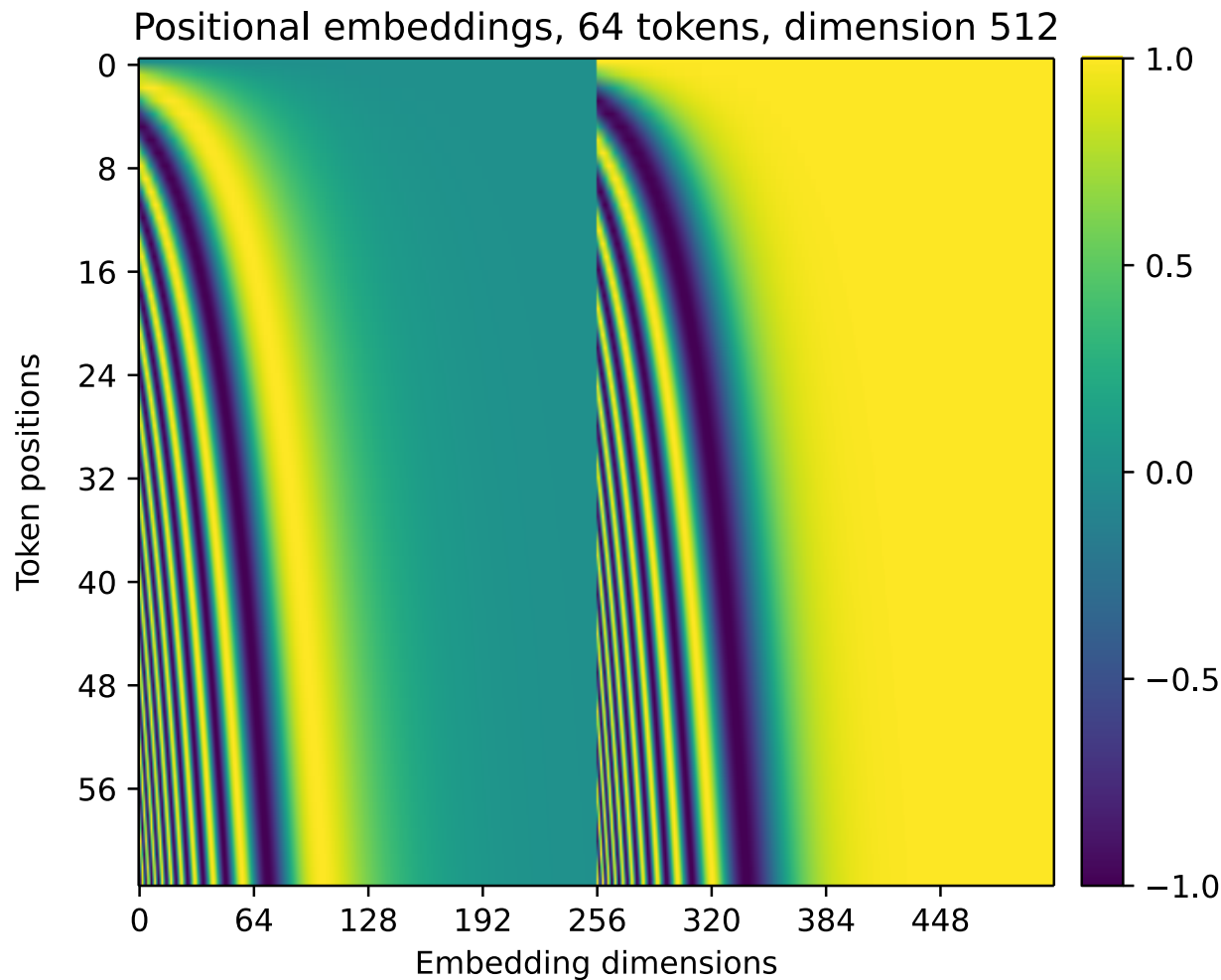
This choice of functions should allow the model to attend to relative positions, since for any fixed  $k$ ,  $\text{PE}_{pos+k}$  is a linear function of  $\text{PE}_{pos}$ , because

$$\begin{aligned}\text{PE}_{(pos+k, 2i)} &= \sin((pos+k)/10000^{2i/d}) \\ &= \sin(pos/10000^{2i/d}) \cdot \cos(k/10000^{2i/d}) + \cos(pos/10000^{2i/d}) \cdot \sin(k/10000^{2i/d}) \\ &= \text{offset}_{(k, 2i)} \cdot \text{PE}_{(pos, 2i)} + \text{offset}_{(k, 2i+1)} \cdot \text{PE}_{(pos, 2i+1)}.\end{aligned}$$

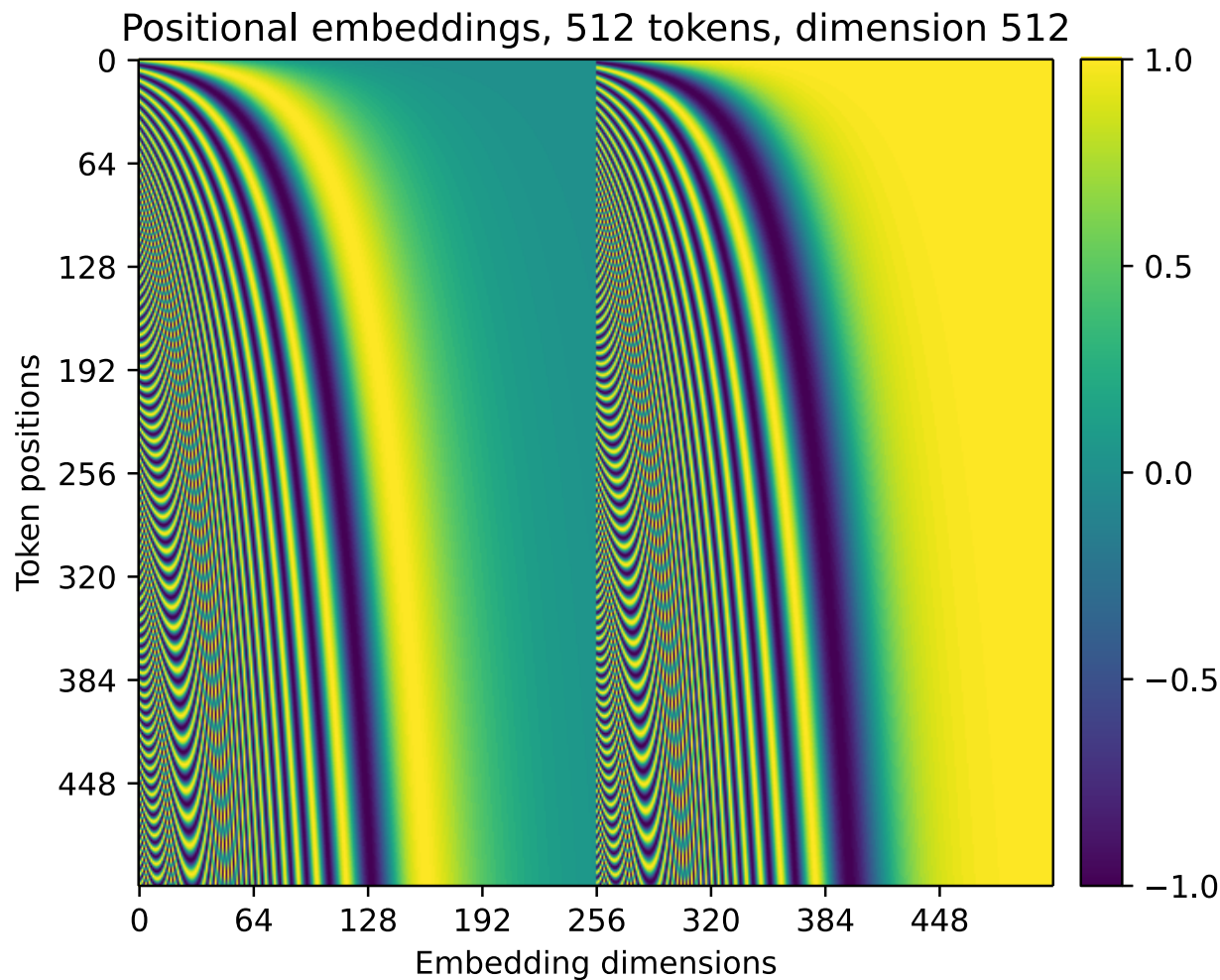














## Regularization

The network is regularized by:

- dropout of input embeddings,
- dropout of each sub-layer, just before it is added to the residual connection (and then normalized),
- label smoothing.

Default dropout rate and also label smoothing weight is 0.1.

## Parallel Execution

Because of the *masked attention*, training can be performed in parallel.

However, inference is still sequential.



## Optimizer

Adam optimizer (with  $\beta_2 = 0.98$ , smaller than the default value of 0.999) is used during training, with the learning rate decreasing proportionally to inverse square root of the step number.

## Warmup

Furthermore, during the first *warmup\_steps* updates, the learning rate is increased linearly from zero to its target value.

$$\text{learning\_rate} = \frac{1}{\sqrt{d_{\text{model}}}} \min \left( \frac{1}{\sqrt{\text{step\_num}}}, \frac{\text{step\_num}}{\text{warmup\_steps}} \cdot \frac{1}{\sqrt{\text{warmup\_steps}}} \right).$$

In the original paper, 4000 warmup steps were proposed.

Note that the goal of warmup is mostly to prevent divergence early in training; the Pre-LN configuration usually trains well even without warmup.

→ tedy chci být 2<sup>n</sup> step\_num

$$= \frac{1}{\sqrt{\text{step\_num}}}$$

tohle bylo inženýrsky  
zmysleno.



Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the 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	<b>41.29</b>	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	<b><math>3.3 \cdot 10^{18}</math></b>	
Transformer (big)	<b>28.4</b>	<b>41.8</b>	$2.3 \cdot 10^{19}$	

Table 2 of "Attention Is All You Need", <https://arxiv.org/abs/1706.03762>

Wordpieces were constructed using BPE with a shared vocabulary of about 37k tokens.



	$N$	$d_{\text{model}}$	$d_{\text{ff}}$	$h$	$d_k$	$d_v$	$P_{\text{drop}}$	$\epsilon_{ls}$	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$	
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65	
(A)					1	512	512				5.29	24.9	
					4	128	128				5.00	25.5	
					16	32	32				4.91	25.8	
					32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58	
					32					5.01	25.4	60	
(C)	2									6.11	23.7	36	
	4									5.19	25.3	50	
	8									4.88	25.5	80	
		256				32	32				5.75	24.5	28
		1024				128	128				4.66	26.0	168
			1024								5.12	25.4	53
			4096								4.75	26.2	90
(D)									0.0	5.77	24.6		
									0.2	4.95	25.5		
									0.0	4.67	25.3		
									0.2	5.47	25.7		
(E)	positional embedding instead of sinusoids									4.92	25.7		
big	6	1024	4096	16					0.3	300K	<b>4.33</b>	<b>26.4</b>	213

Table 4 of "Attention Is All You Need", <https://arxiv.org/abs/1706.03762>

The PPL is *perplexity per wordpiece*, where perplexity is  $e^{H(P)}$ , i.e.,  $e^{\text{loss}}$  in our case.



**Main Takeaway** *potrebujn ale fakt hvoozuě mæ dat, aby se to model učil ten tush.*  
Generally, Transformer provides more powerful sequence-to-sequence architecture and also sequence element representation architecture than RNNs, but requires **substantially more** data.

## 3D Visualization of a Decoder-only Model

On <https://bbycroft.net/llm> you can find a 3D visualization with the description of the Transformer computation steps of several GPT models. The GPT models are language models (they estimate conditional probability of a word given its previous context), and therefore consist purely of the decoder part of a Transformer (so they do not contain neither an encoder nor encoder-decoder attention; consequently, all their self-attentions are masked).