Machine Translation

Session 11: The Transformer

Sharid Loáiciga — July 1st, 2020

Slides credits: Alessandro Raganato
(who credits Jay Alammar, Łukasz Kaiser, Ashish Vaswani)

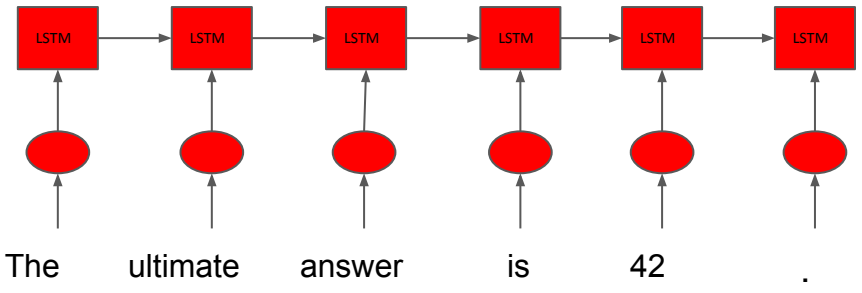
Learning Representations of Variable Length Data

- Neural Machine Translation (NMT) is a way to do Machine Translation with a *single neural network (end-to-end)*
- Sequence-to-sequence is the architecture for NMT

Learning Representations of Variable Length Data

- Neural Machine Translation (NMT) is a way to do Machine Translation with a *single neural network (end-to-end)*
- Sequence-to-sequence is the architecture for NMT

- **RNNs Everywhere**
  - Model of choice for learning variable-length representations.
  - Natural fit for sentences.
  - LSTMs, GRUs and variants dominate recurrent models.
  - At the core of seq2seq (w/ attention)



Learning Representations of Variable Length Data

- **RNNs Everywhere**
  - Model of choice for learning variable-length representations.
  - Natural fit for sentences.
  - LSTMs, GRUs and variants dominate recurrent models.
  - At the core of seq2seq (w/ attention)

- **But:**
  - Sequentiality **prohibits parallelization** within instances
  - **Long-range dependencies still tricky**, despite gating
  - RNNs (w/ sequence-aligned states) seem wasteful

Learning Representations of Variable Length Data

- **RNNs Everywhere**
  - Model of choice for learning variable-length representations.
  - Natural fit for sentences.
  - LSTMs, GRUs and variants dominate recurrent models.
  - At the core of seq2seq (w/ attention)

- **But:**
  - Sequentiality **prohibits parallelization** within instances
  - **Long-range dependencies still tricky**, despite gating
  - RNNs (w/ sequence-aligned states) seem wasteful

- Can we get rid of RNNs? with what we can replace them?

Attention

- Attention between encoder and decoder is crucial in NMT.
- **Why not use attention for representations?**

Self-Attention

- **Self-attention:**
  learn dependencies between words in the sentence and use that information to capture the internal structure of the sentence.

Self-Attention

- **Self-attention:**
  learn dependencies between words in the sentence and use that information to capture the internal structure of the sentence.

- **Self-Attention at a High Level**
  For example the following sentence is an input sentence we want to translate:
  ”The animal didn't cross the street because it was too tired”

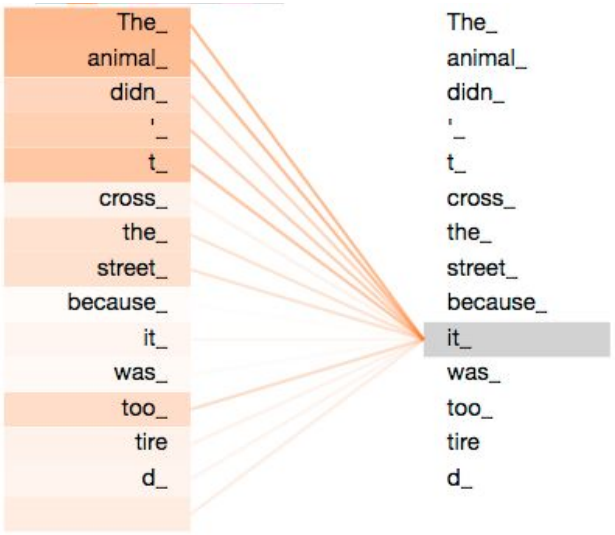
  What does “it” in this sentence refer to?

Self-Attention

- **Self-attention:**
  learn dependencies between words in the sentence and use that information to capture the internal structure of the sentence.

- **Self-Attention at a High Level**
  For example the following sentence is an input sentence we want to translate:
  ”The animal didn't cross the street because it was too tired”

  What does “it” in this sentence refer to?



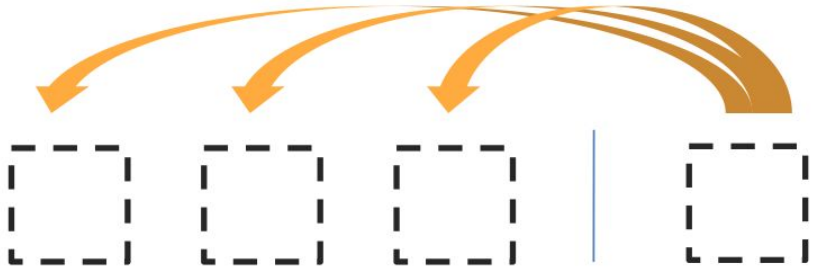
Self-Attention

- **Self-attention:**
  learn dependencies between words in the sentence and use that information to capture the internal structure of the sentence.

- **Self-Attention at a High Level**
  For example the following sentence is an input sentence we want to translate:
  ”The animal didn't cross the street because it was too tired”

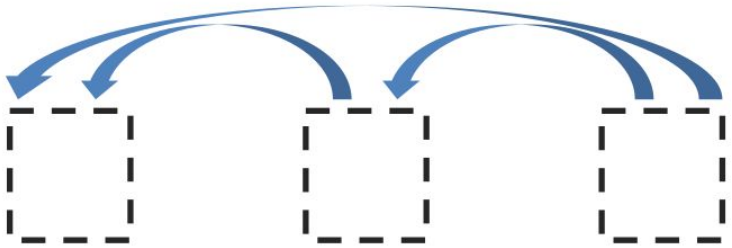
  As the model processes each word (each position in the input sequence), **self attention** allows it to look at other positions in the input sequence for **clues** that can help lead to a **better encoding for the word itself**.

Three ways of attention in encoder-decoder



Encoder-Decoder Attention

Encoder Self-Attention



MaskedDecoder Self-Attention

The Transformer

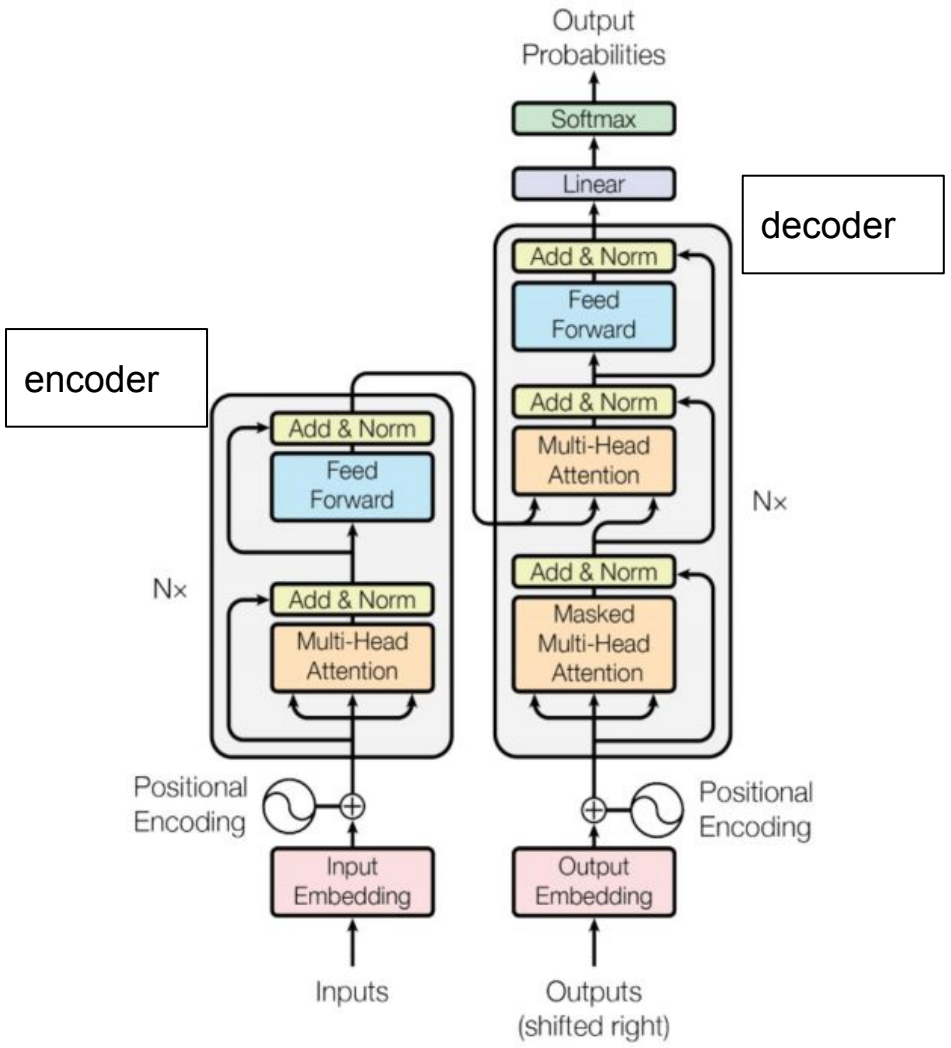
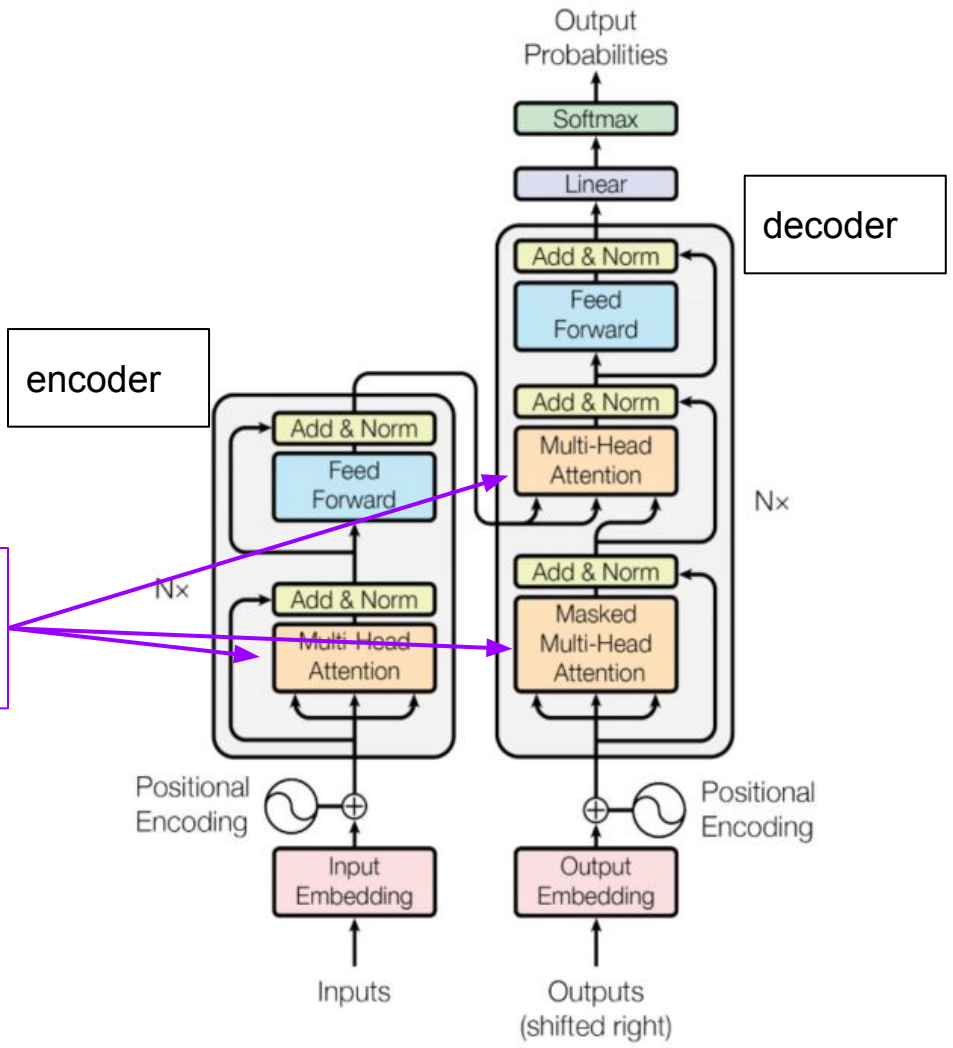


Figure 1: The Transformer - model architecture.

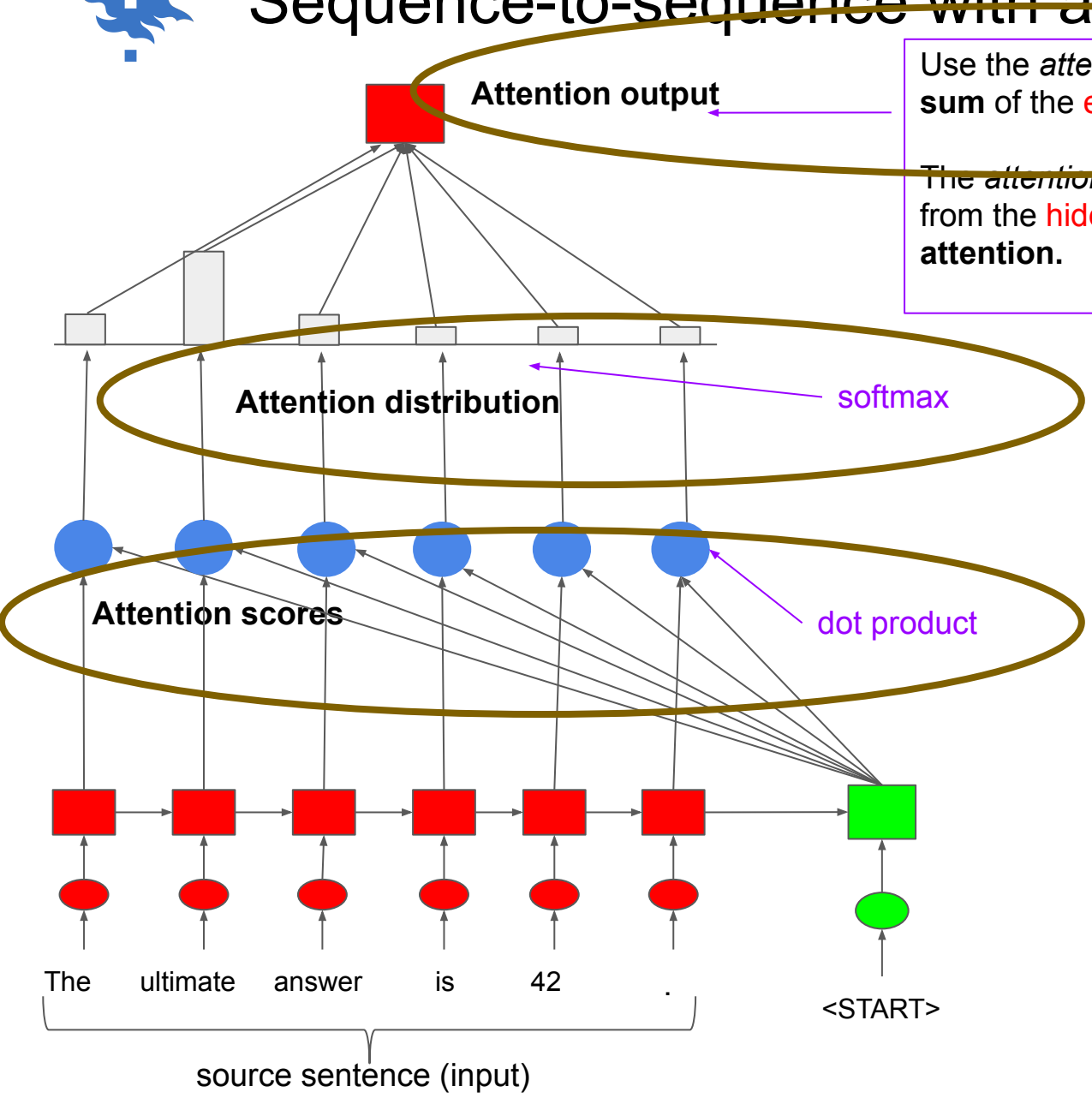
The Transformer



Here there is the self-attention mechanism

Figure 1: The Transformer - model architecture.

Sequence-to-sequence with attention



Use the *attention distribution* to take a **weighted sum** of the encoder hidden states.

The *attention output* mostly contains information from the hidden states that received **high attention.**

Self-Attention in Detail

Self-Attention in Detail

- The **first step** in calculating self-attention is to **create three vectors** from each word embedding at the encoder side.
- So for each word, we create:
  - Query vector
  - Key vector
  - Value vector
- These vectors are created by multiplying the embedding by three matrices that we trained during the training process.

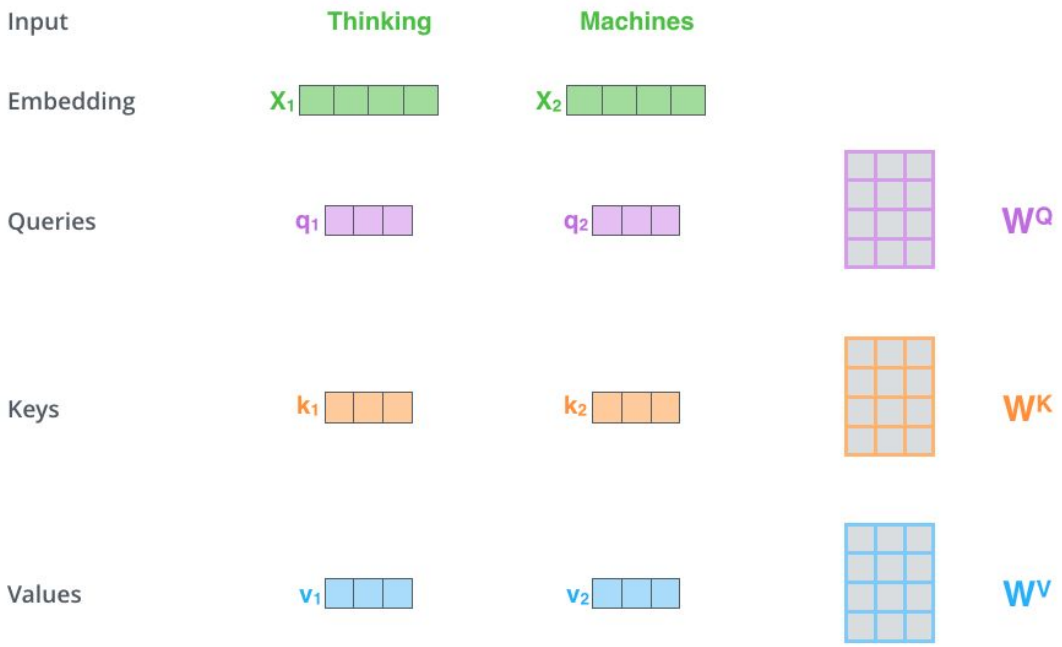
Self-Attention in Detail

- The **first step** in calculating self-attention is to **create three vectors** from each word embedding at the encoder side.
- So for each word, we create:
  - Query vector
  - Key vector
  - Value vector
- These vectors are created by multiplying the embedding by three matrices that we trained during the training process.

- What are the “query”, “key”, and “value” vectors?
  They’re abstractions that are useful for calculating and thinking about attention.

Self-Attention in Detail

- The **first step** in calculating self-attention is to **create three vectors** from each word embedding at the encoder side.
- So for each word, we create:
  - Query vector
  - Key vector
  - Value vector
- These vectors are created by multiplying the embedding by three matrices that we trained during the training process.



Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

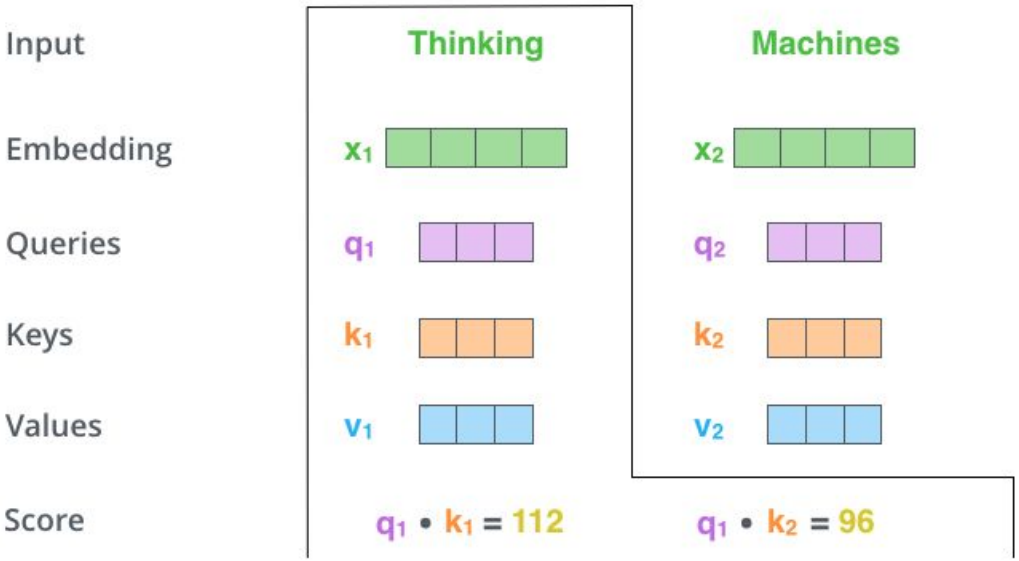
Self-Attention in Detail

- The **second step** in calculating self-attention is to **calculate a score**.
- The score determines **how much focus to place** on other parts of the input sentence as we encode a word at a certain position.
- The score is the **dot product** of the query vector with the key vector of the respective word we're scoring.

Self-Attention in Detail

- The **second step** in calculating self-attention is to **calculate a score**.
- The score determines **how much focus to place** on other parts of the input sentence as we encode a word at a certain position.
- The score is the **dot product** of the query vector with the key vector of the respective word we're scoring.

- For example: if we're processing the *self-attention* for the word in position 1 (Thinking), the first score would be the dot product of q1 and k1. The second score would be the dot product of q1 and k2, and so on...

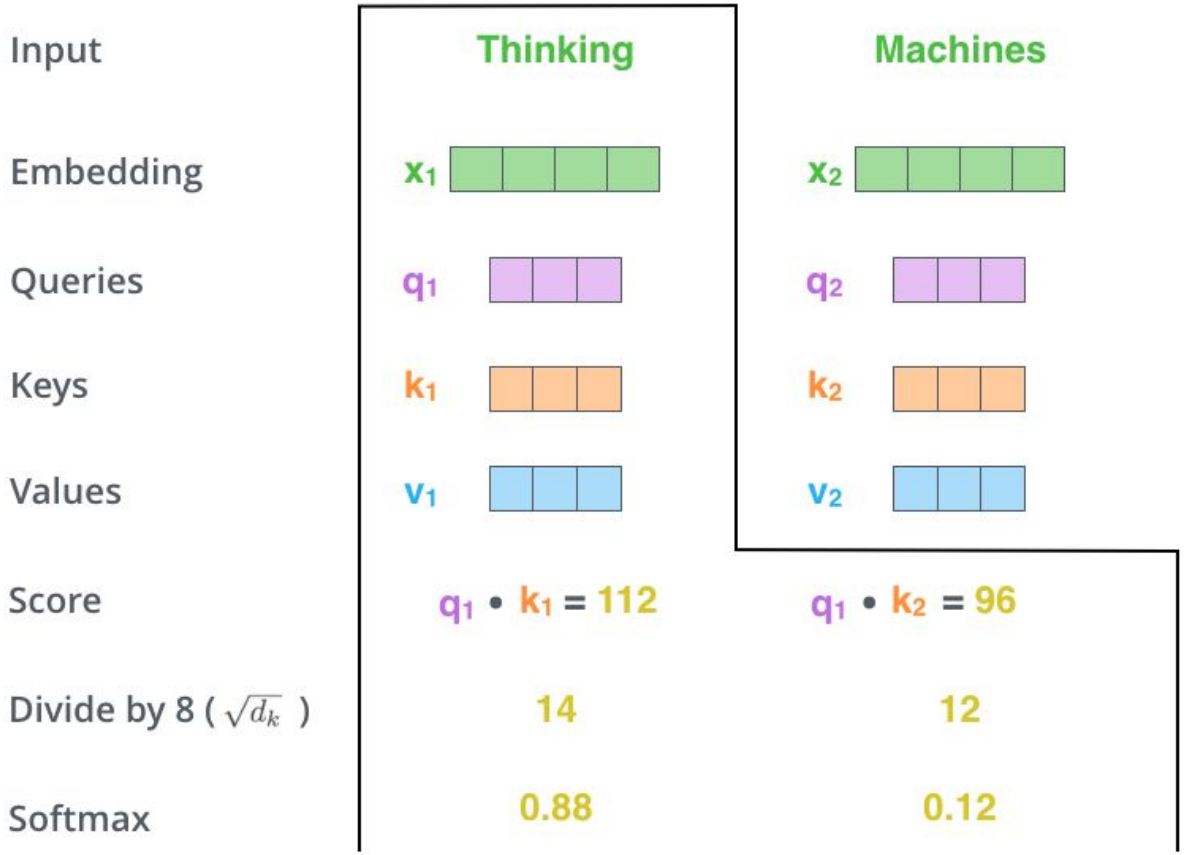


Self-Attention in Detail

- The **third and fourth steps:**
  - divide the scores by the square root of the dimension of the key vectors (for training stability reasons)
  - pass the result through a **softmax** operation. Softmax normalizes the scores so they’re all positive and add up to 1.

Self-Attention in Detail

- The **third and fourth steps:**
  - divide the scores by the square root of the dimension of the key vectors (for training stability reasons)
  - pass the result through a **softmax** operation. Softmax normalizes the scores so they're all positive and add up to 1.

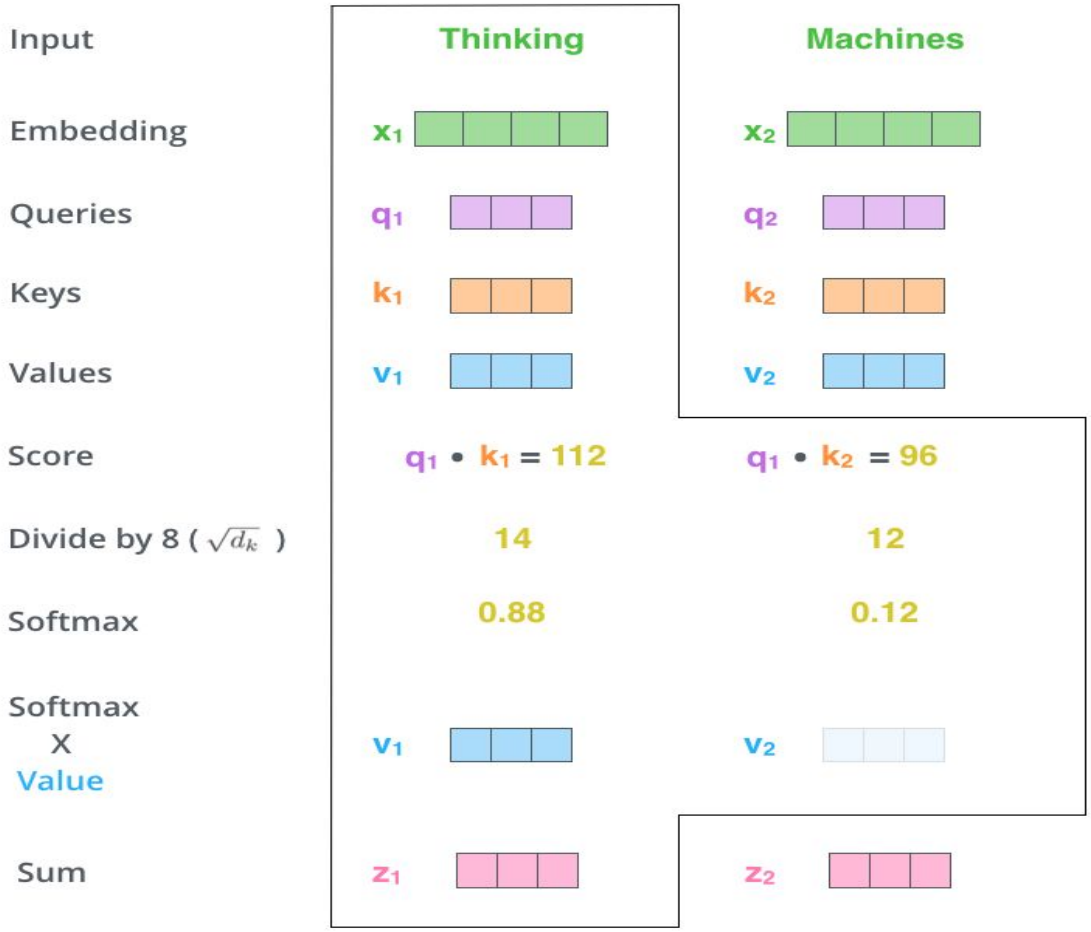


Self-Attention in Detail

- The **fifth and sixth steps:**
  - **multiply** each value **vector** by the **softmax** score
  - **sum up** the **weighted** value **vectors:** this produces the **output of the self-attention** layer at this position (e.g., for the first word).

Self-Attention in Detail

- The **fifth and sixth steps:**
  - **multiply** each value **vector** by the **softmax** score
  - **sum up** the **weighted** value **vectors:** this produces the **output of the self-attention** layer at this position (e.g., for the first word).



Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

¿Multi-Head Attention..?

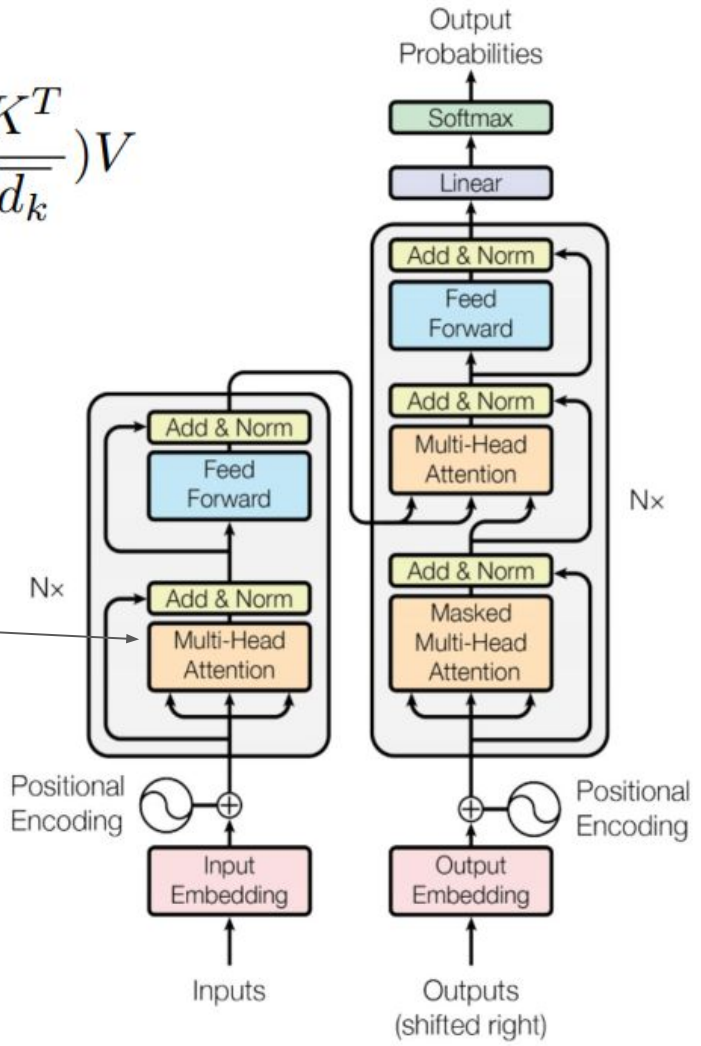


Figure 1: The Transformer - model architecture.

The Beast With Many Heads

- Further refined the *self-attention* layer by adding a mechanism called "*multi-headed*" attention:
  - It expands the model's ability to focus on different positions.
  - It gives the attention layer multiple "representation subspaces".

The Beast With Many Heads

- Further refined the *self-attention* layer by adding a mechanism called "*multi-headed*" attention:
  - It expands the model's ability to focus on different positions.
  - It gives the attention layer multiple "representation subspaces".

How it works:

- Do the same self-attention calculation outlined before, just more times with different weight matrices, ending up with different output matrices.

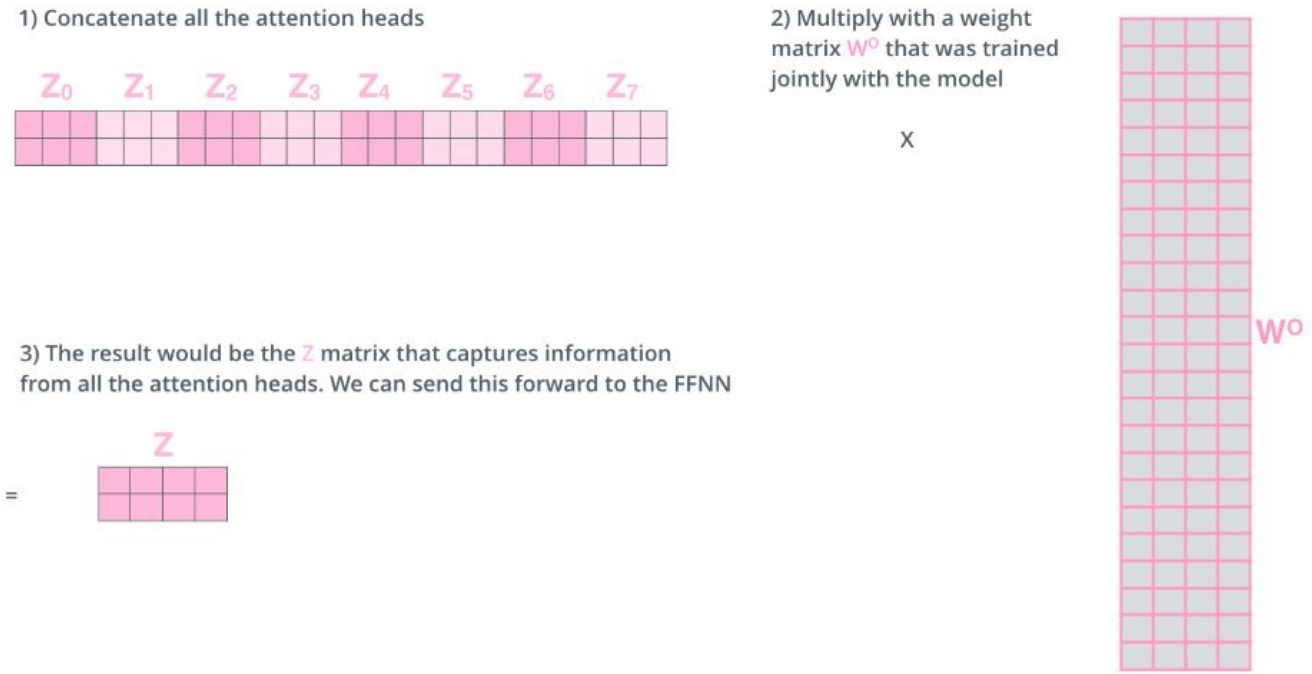
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

The Beast With Many Heads

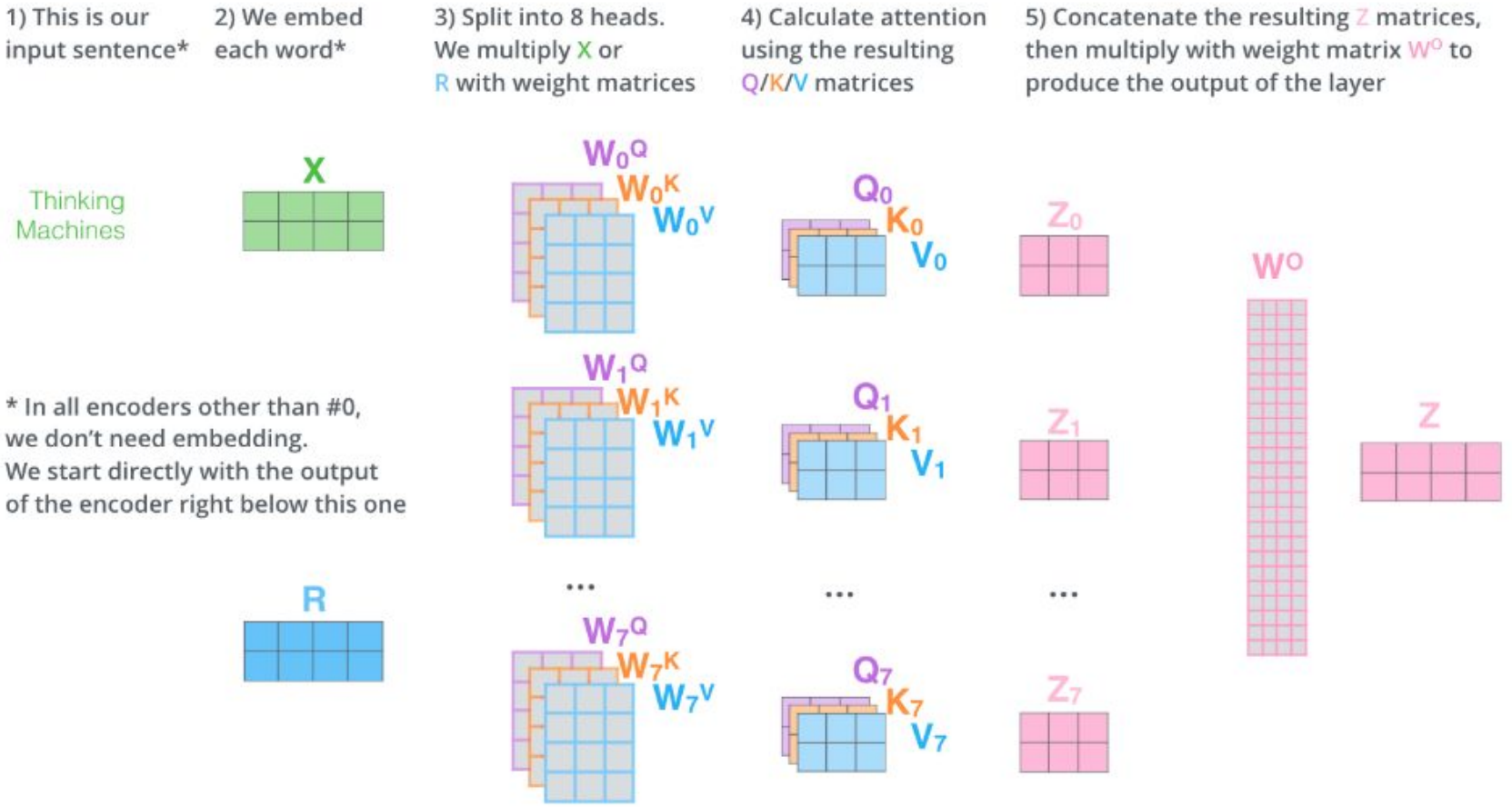
- Further refined the *self-attention* layer by adding a mechanism called "*multi-headed*" attention:
  - It expands the model's ability to focus on different positions.
  - It gives the attention layer multiple "representation subspaces".

How it works:

- Do the same self-attention calculation outlined before, just more times with different weight matrices, ending up with different output matrices.

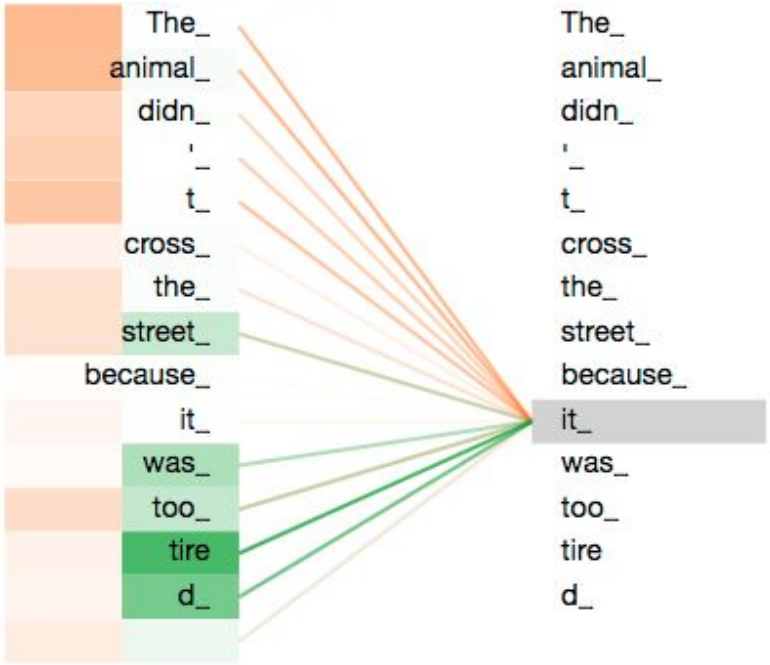


The Beast With Many Heads



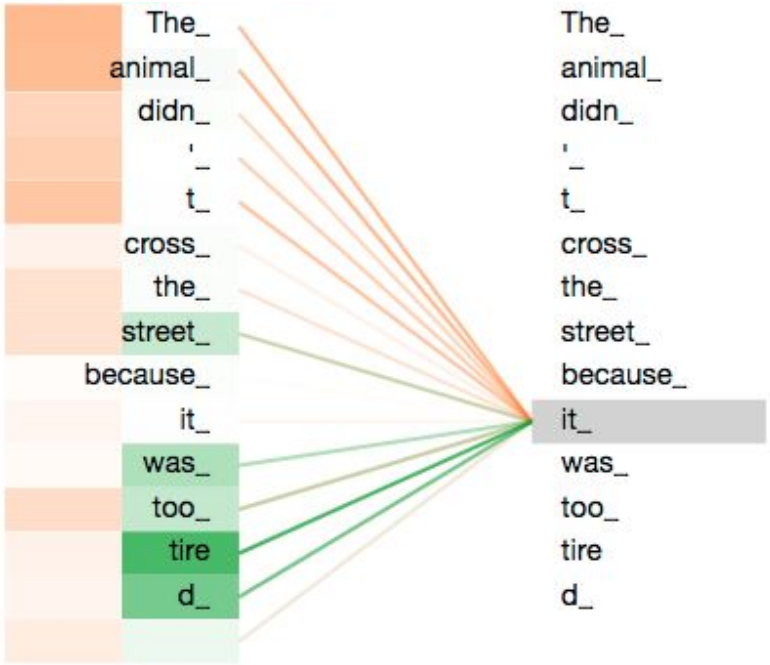
The Beast With Many Heads

- for the word "it", one attention head is focusing most on "the animal", while another one is focusing on "tired".

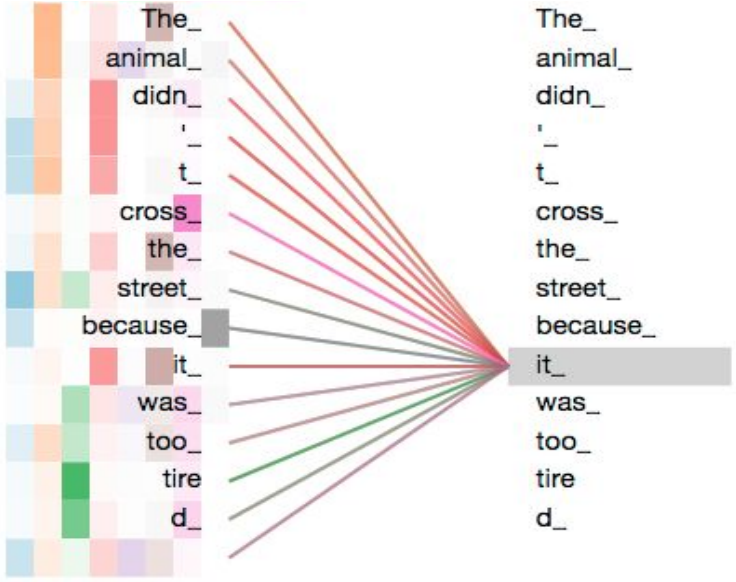


The Beast With Many Heads

- for the word "it", one attention head is focusing most on "the animal", while another one is focusing on "tired".



- Visualizing all the attention heads, however, things can be harder to interpret...



Representing The Order of The Sequence Using Positional Encoding

- How to account for the order of the words in the input sequence?

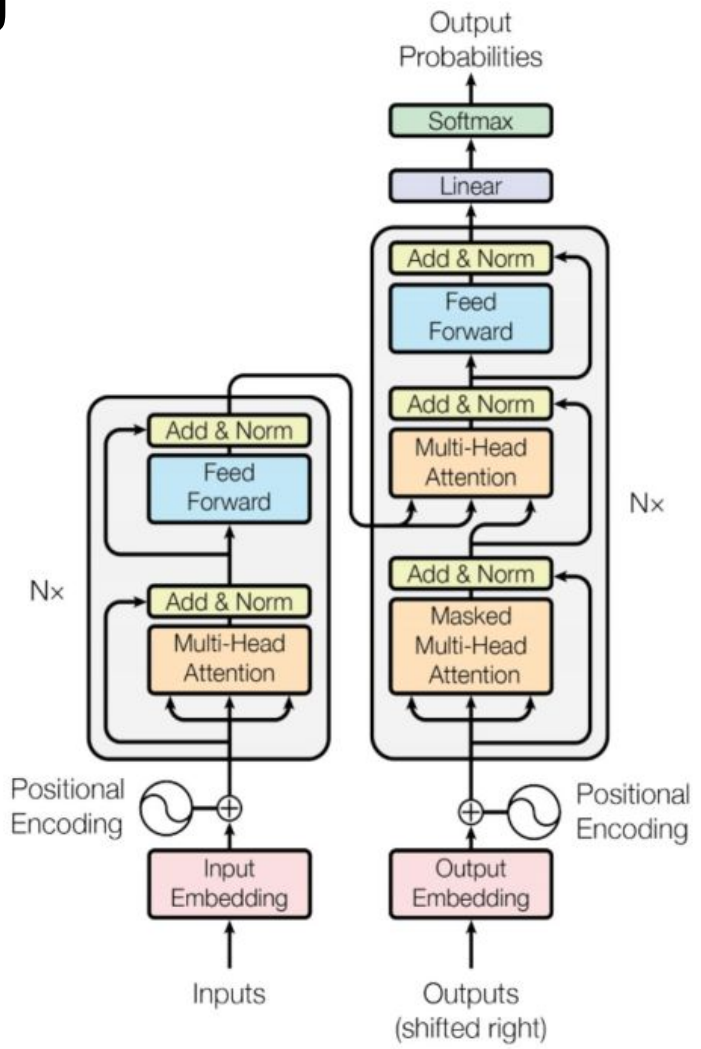


Figure 1: The Transformer - model architecture.

Representing The Order of The Sequence Using Positional Encoding

- How to account for the order of the words in the input sequence?
  Positional Encoding!

  It gives the advantage of being able to scale to unseen lengths of sequences (e.g. if the trained model is asked to translate a sentence longer than any of those in the training set).

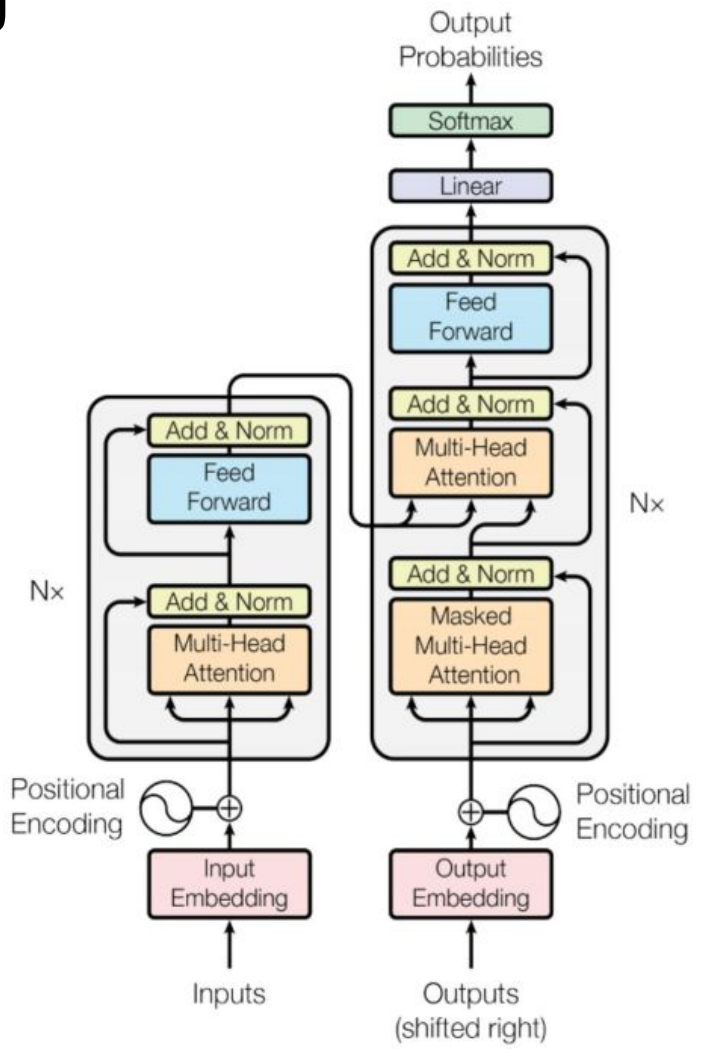
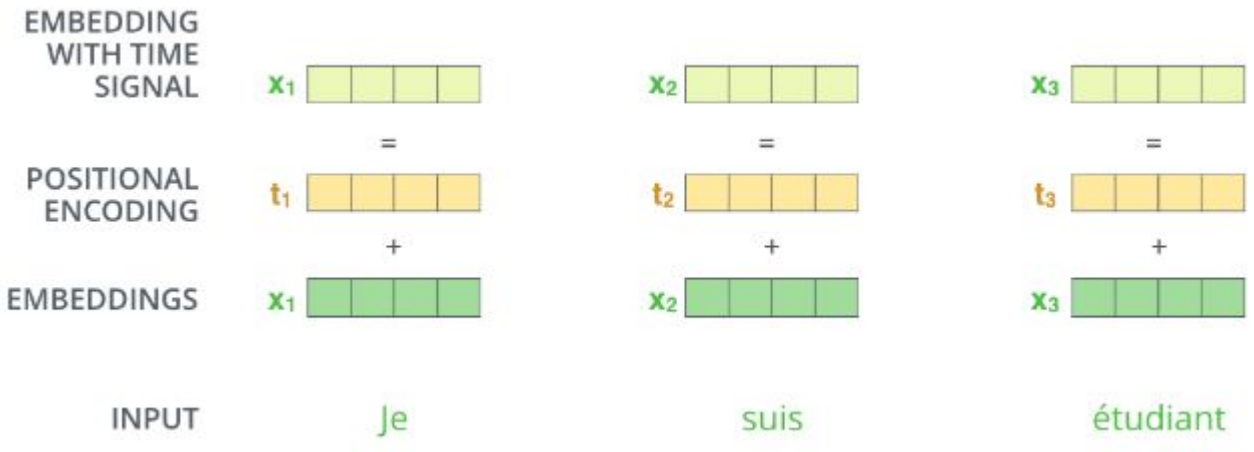


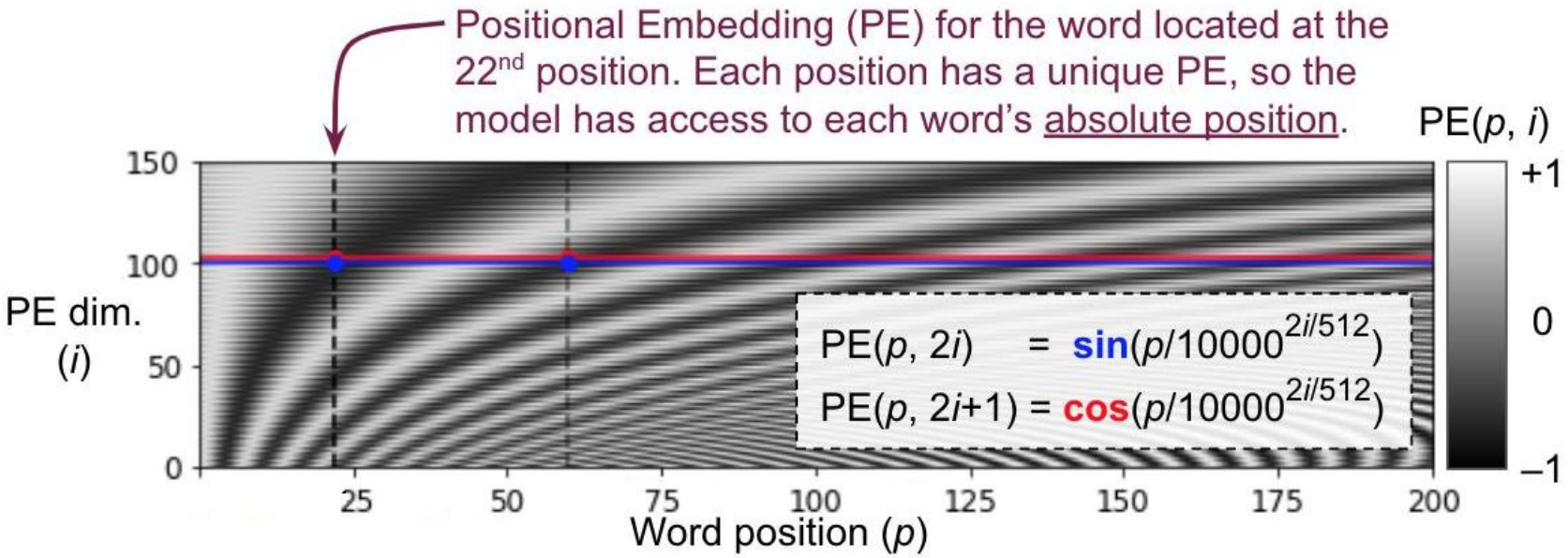
Figure 1: The Transformer - model architecture.

Representing The Order of The Sequence Using Positional Encoding

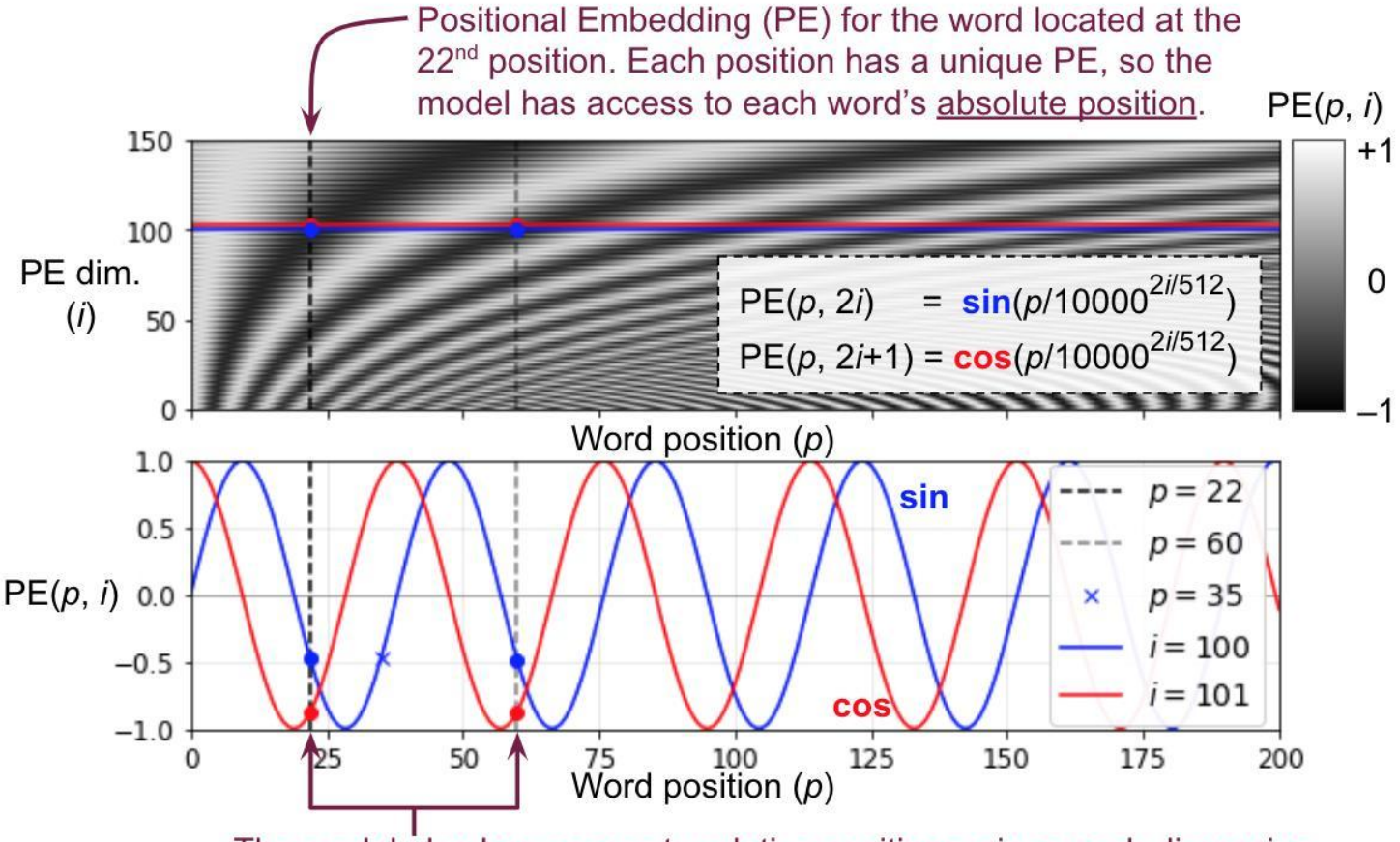
- **Positional Encoding:**
- The transformer **adds a vector to each input embedding**.
  These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence



Representing The Order of The Sequence Using Positional Encoding



Representing The Order of The Sequence Using Positional Encoding



Attention is Cheap!

	FLOPs
Self-Attention	$O(\text{length}^2 \cdot \text{dim})$
RNN (LSTM)	$O(\text{length} \cdot \text{dim}^2)$
Convolution	$O(\text{length} \cdot \text{dim}^2 \cdot \text{kernel\_width})$

FLOPs: floating point operations per second is a measure of computer performance.

Attention is Cheap!

	FLOPs	
Self-Attention	$O(\text{length}^2 \cdot \text{dim})$	$= 4 \cdot 10^9$
RNN (LSTM)	$O(\text{length} \cdot \text{dim}^2)$	$= 16 \cdot 10^9$
Convolution	$O(\text{length} \cdot \text{dim}^2 \cdot \text{kernel\_width})$	$= 6 \cdot 10^9$

length=1000 dim=1000 kernel_width=3

Machine Translation Results: WMT-14

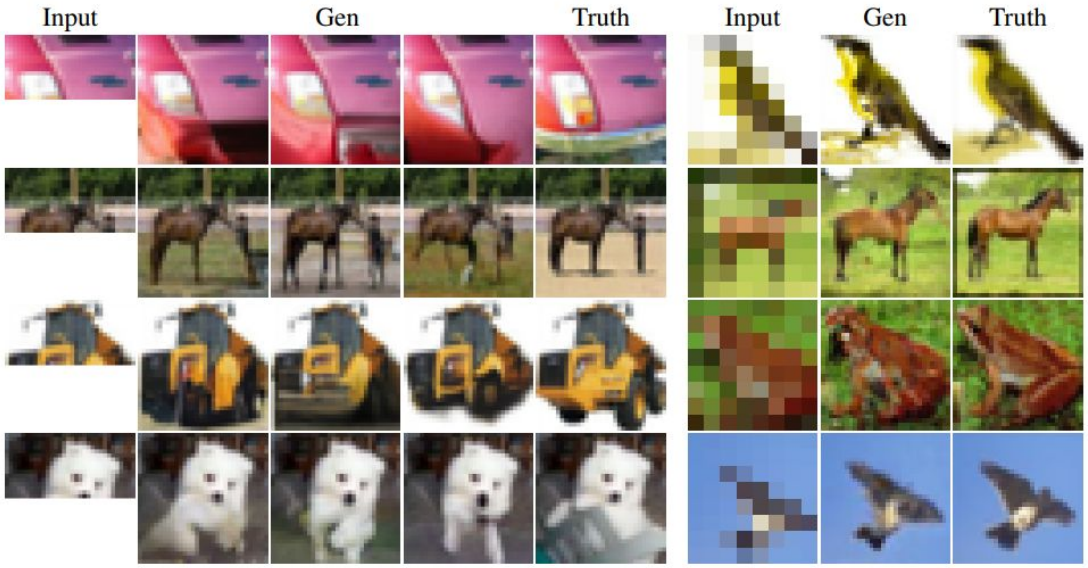
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [17]	23.75			
Deep-Att + PosUnk [37]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [36]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [31]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [37]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [36]	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	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Coreference resolution (Winograd schemas)

Sentence	Google Translate	Transformer
The cow ate the hay because it was **delicious**.	La vache mangeait le foin parce **qu'elle** était délicieuse.	La vache a mangé le foin parce **qu'il** était délicieux.
The cow ate the hay because it was **hungry**.	La vache mangeait le foin parce **qu'elle** avait faim.	La vache mangeait le foin parce **qu'elle** avait faim.
The women stopped drinking the wines because they were **carcinogenic**.	Les femmes ont cessé de boire les vins parce **qu'ils** étaient cancérogènes.	Les femmes ont cessé de boire les vins parce **qu'ils** étaient cancérigènes.
The women stopped drinking the wines because they were **pregnant**.	Les femmes ont cessé de boire les vins parce **qu'ils** étaient enceintes.	Les femmes ont cessé de boire les vins parce **qu'elles** étaient enceintes.
The city councilmen refused the female demonstrators a permit because they **advocated** violence.	Les conseillers municipaux ont refusé aux femmes manifestantes un permis parce **qu'ils** préconisaient la violence.	Le conseil municipal a refusé aux manifestantes un permis parce **qu'elles** prônaient la violence.
The city councilmen refused the female demonstrators a permit because they **feared** violence.	Les conseillers municipaux ont refusé aux femmes manifestantes un permis parce **qu'ils** craignaient la violence	Le conseil municipal a refusé aux manifestantes un permis parce **qu'elles** craignaient la violence.*

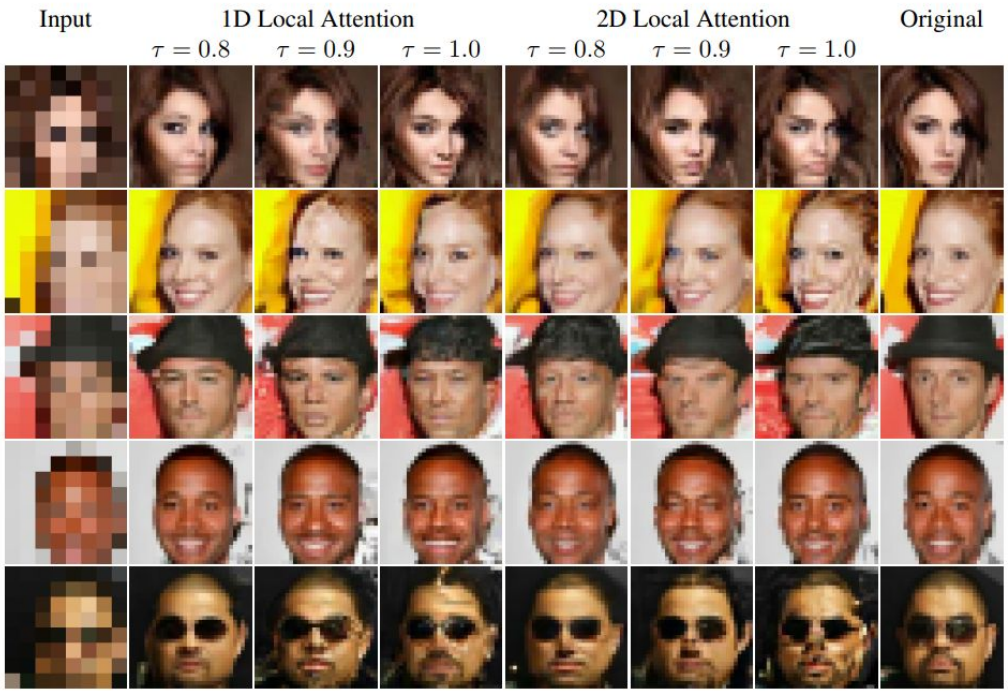
Self-Attention Everywhere

- Image generation



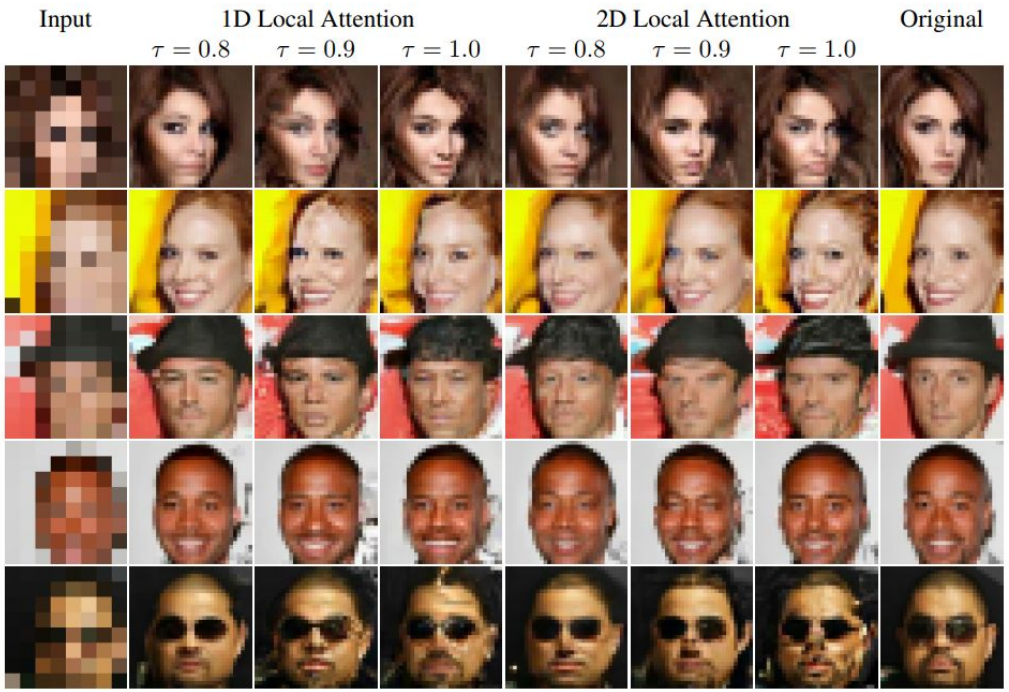
Self-Attention Everywhere

- Image generation



Self-Attention Everywhere

- Image generation



- Music generation:
  - https://magenta.tensorflow.org/music-transformer

NN Hyperparameters

- Regularization
- Loss function

[Slav Petrov, 2015]

NN Hyperparameters

- Regularization
- Loss function

- Dimensions
- Activation function
- Initialization
- Adagrad
- Dropout

[Slav Petrov, 2015]

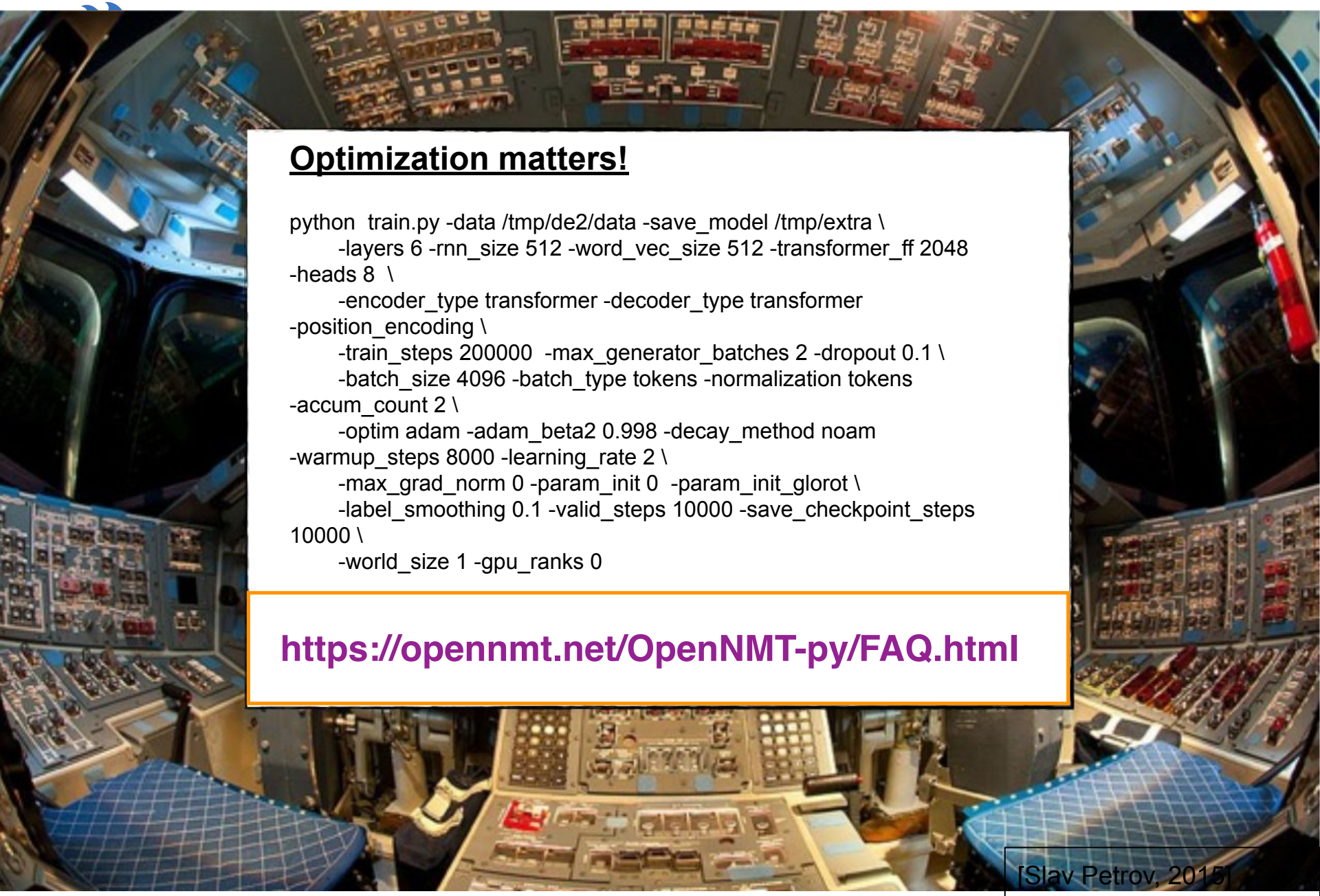
NN Hyperparameters

- Regularization
- Loss function

- Dimensions
- Activation functi
- Initialization
- Adagrad
- Dropout

- Mini-batch size
- Initial learning rate
- Learning rate schedule
- Momentum
- Stopping time
- Parameter averaging

[Slav Petrov, 2015]



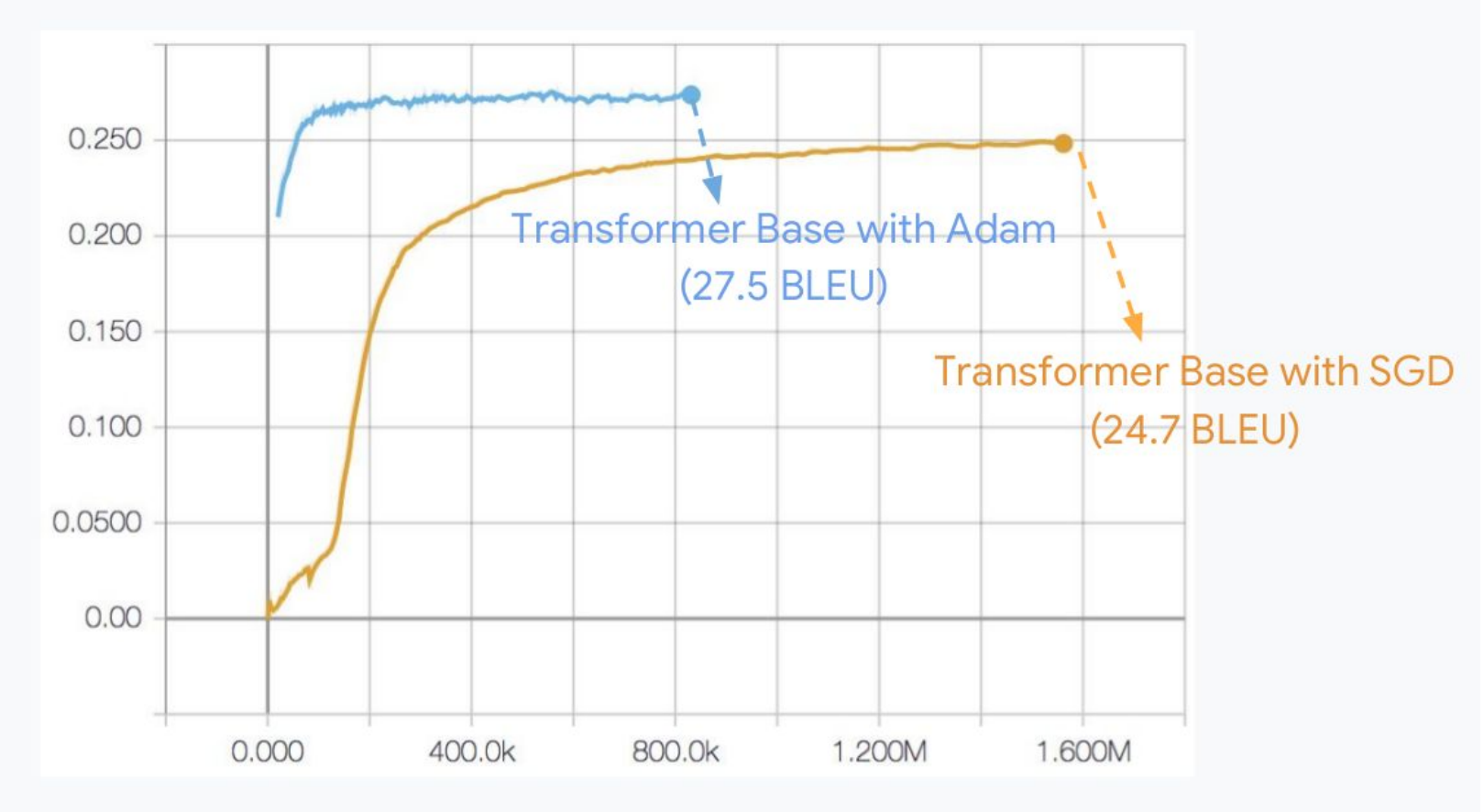
Optimization matters!

```
python  train.py -data /tmp/de2/data -save_model /tmp/extra \
        -layers 6 -rnn_size 512 -word_vec_size 512 -transformer_ff 2048
-heads 8  \
        -encoder_type transformer -decoder_type transformer
-position_encoding \
        -train_steps 200000  -max_generator_batches 2 -dropout 0.1 \
        -batch_size 4096 -batch_type tokens -normalization tokens
-accum_count 2 \
        -optim adam -adam_beta2 0.998 -decay_method noam
-warmup_steps 8000 -learning_rate 2 \
        -max_grad_norm 0 -param_init 0  -param_init_glorot \
        -label_smoothing 0.1 -valid_steps 10000 -save_checkpoint_steps
10000 \
        -world_size 1 -gpu_ranks 0
```

https://opennmt.net/OpenNMT-py/FAQ.html

[Slav Petrov, 2015]

NN Hyperparameters



[Orhan Firat, 2018]

Ablations

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{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	**4.33**	**26.4**	213

Readings

- Transformer model:
  Vaswani, Ashish, et al. (2017)
  Attention Is All You Need
  https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

  Martin Popel, Ondřej Bojar (2018)
  Training Tips for the Transformer Model
  https://arxiv.org/pdf/1804.00247.pdf

  A Raganato, J Tiedemann (2018)
  An Analysis of Encoder Representations in Transformer-Based Machine Translation
  https://www.aclweb.org/anthology/W18-5431

  Alexander M. Rush (2018)
  The Annotated Transformer
  https://aclweb.org/anthology/W18-2509

  Niki Parmar et al. (2018)
  Image Transformer
  https://arxiv.org/pdf/1802.05751.pdf

  Cheng-Zhi Anna Huang et al. (2018)
  Music Transformer
  https://arxiv.org/pdf/1809.04281.pdf