Transformer, BERT, ViT

Milan Straka

April 22, 2025

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

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

Attention is All You Need

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.

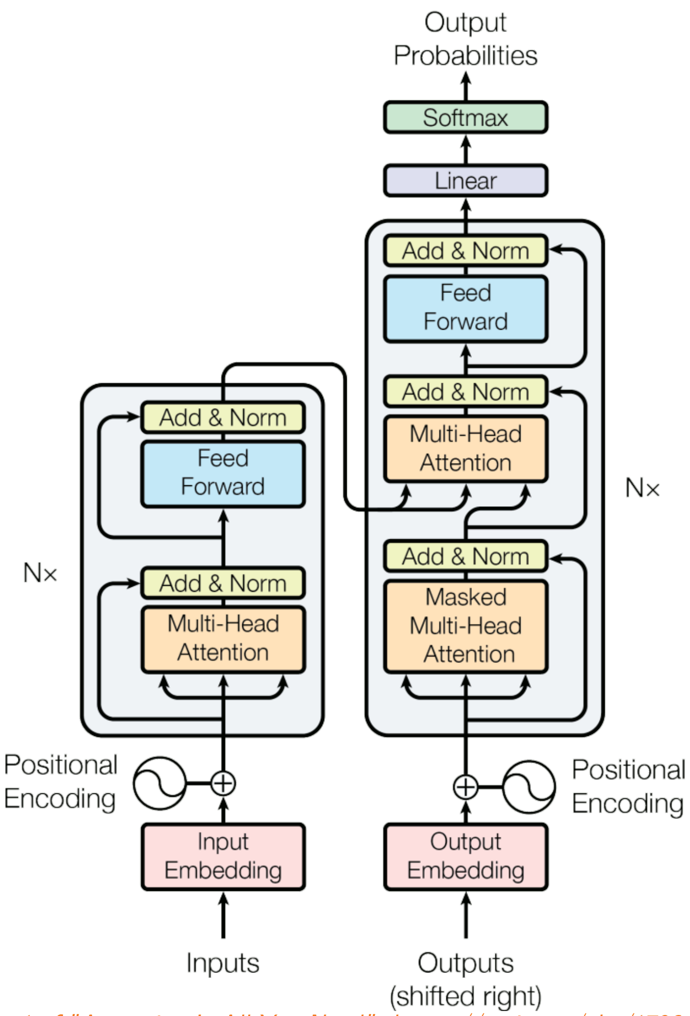
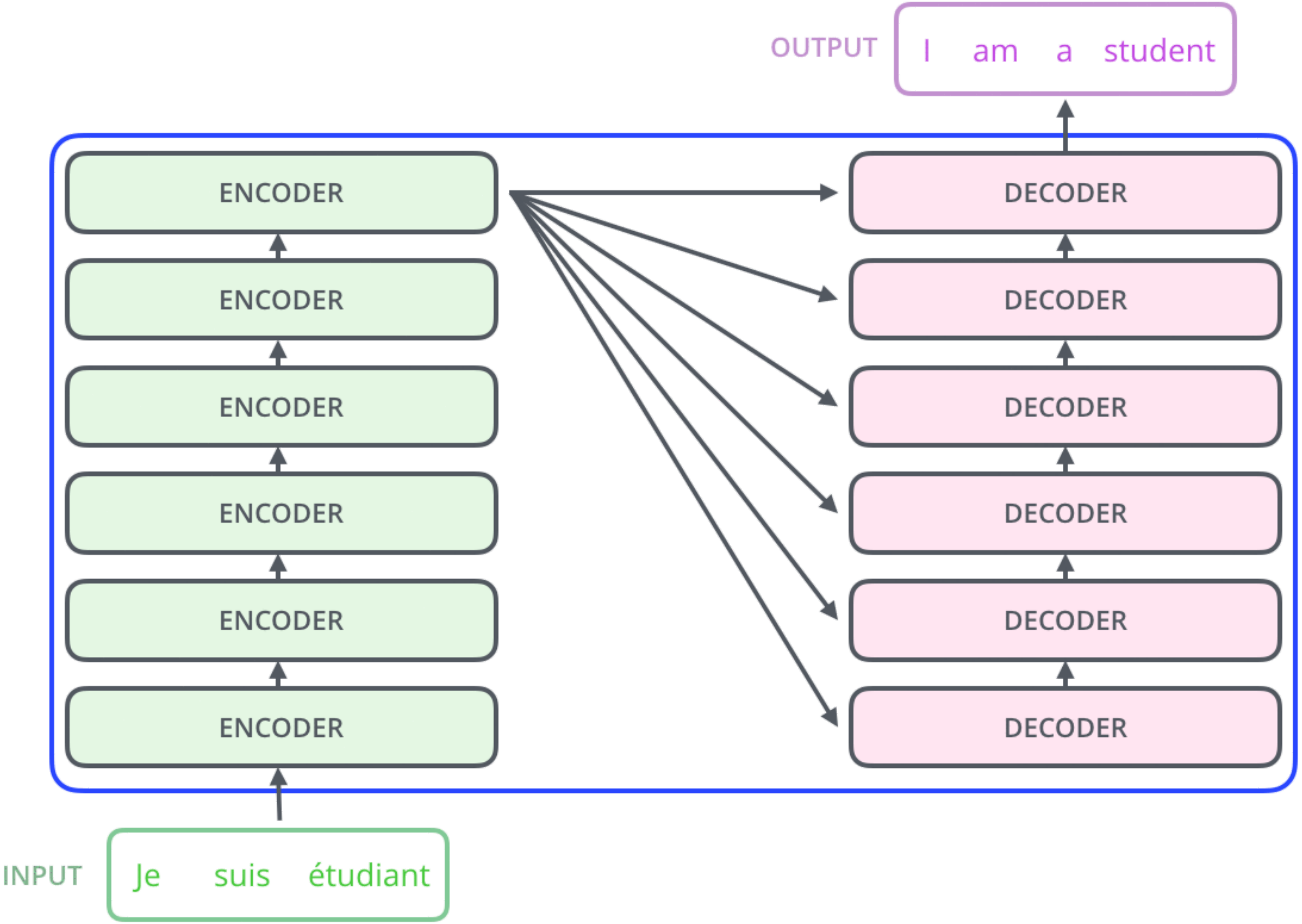
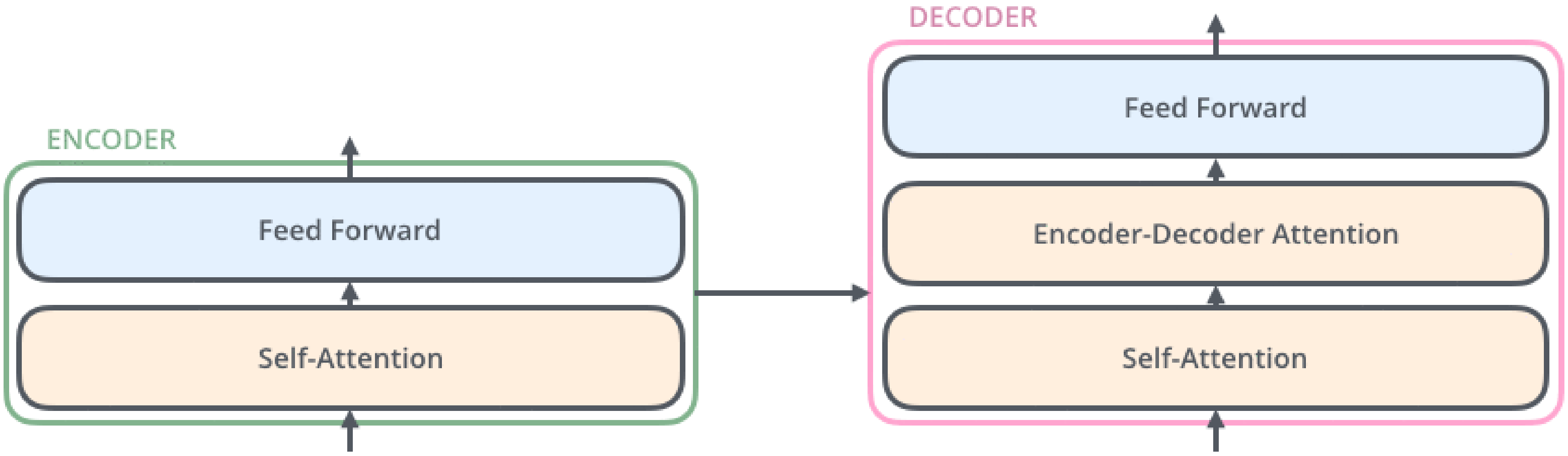


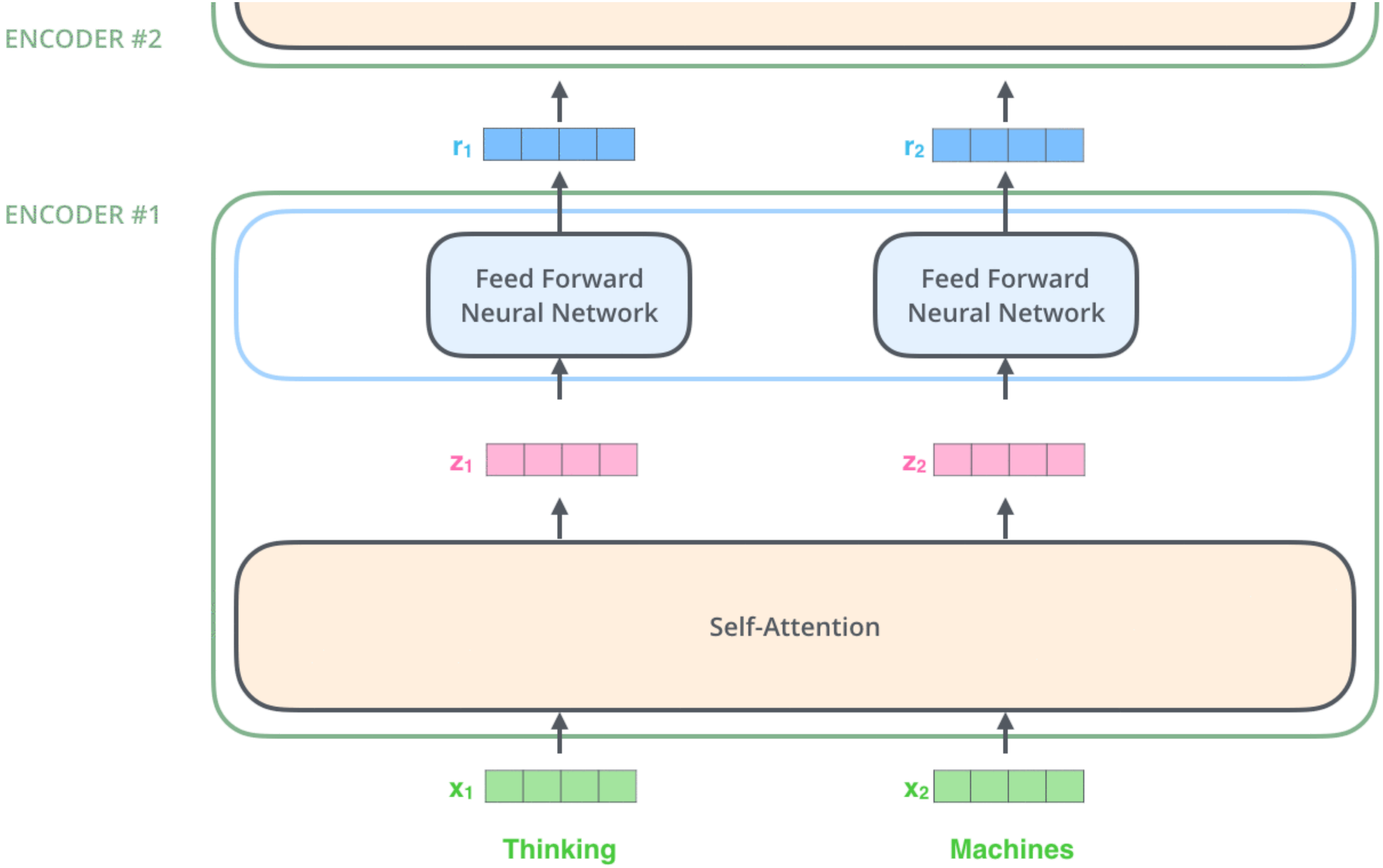
Figure 1 of "Attention Is All You Need", https://arxiv.org/abs/1706.03762



http://jalammar.github.io/images/t/The_transformer_encoder_decoder_stack.png



http://jalammar.github.io/images/t/Transformer_decoder.png



http://jalammar.github.io/images/t/encoder_with_tensors_2.png

Transformer – Self-Attention

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

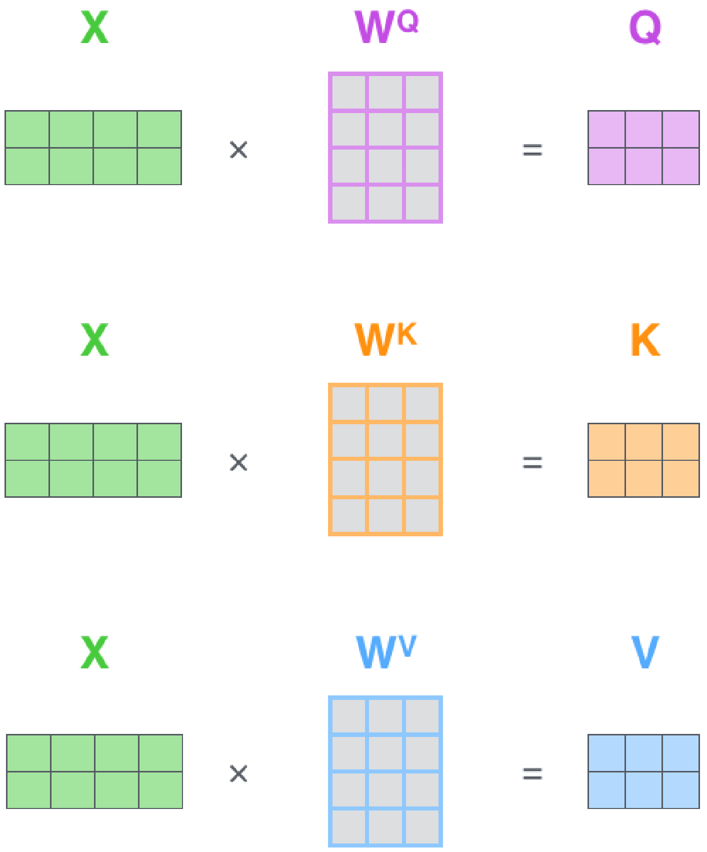
The attention module for a queries $\boldsymbol{Q} \in \mathbb{R}^{n \times d_k}$, keys $\boldsymbol{K} \in \mathbb{R}^{n \times d_k}$ and values $\boldsymbol{V} \in \mathbb{R}^{n \times d_v}$ is defined as:

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

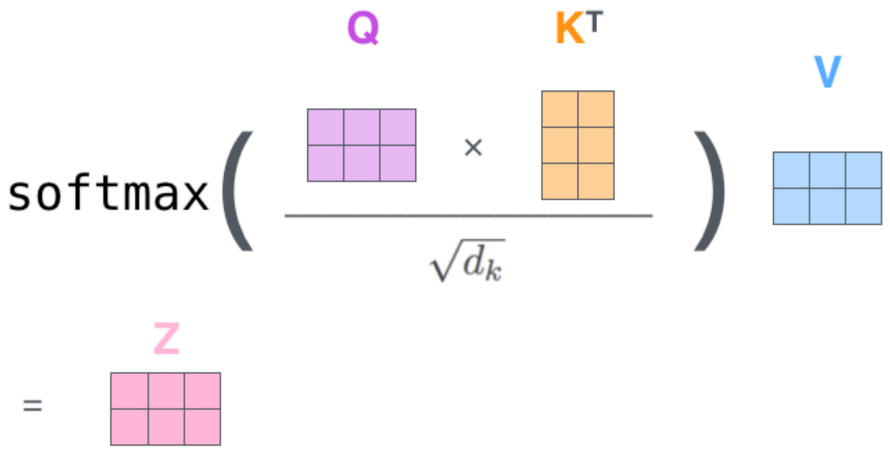
The queries, keys and values are computed from the input word representations $\boldsymbol{X}$ using a linear transformation as

$$\begin{aligned}\boldsymbol{Q} &= \boldsymbol{X}\boldsymbol{W}^Q \\ \boldsymbol{K} &= \boldsymbol{X}\boldsymbol{W}^K \\ \boldsymbol{V} &= \boldsymbol{X}\boldsymbol{W}^V\end{aligned}$$

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

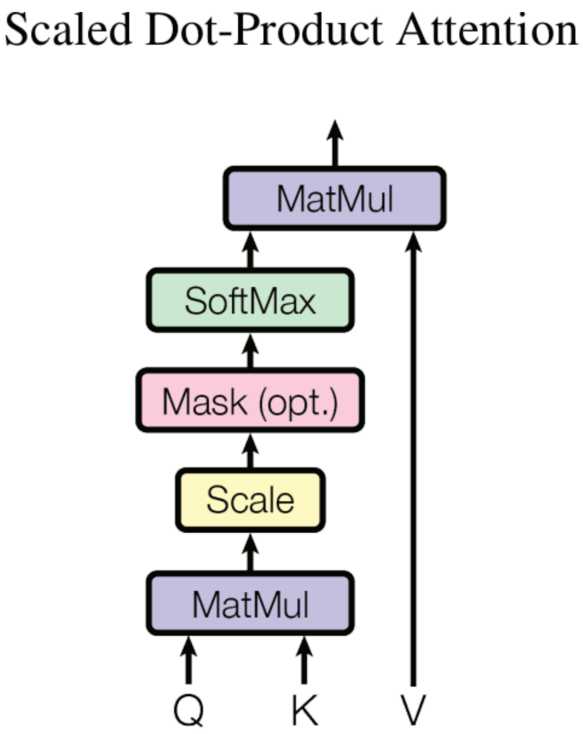


http://jalammar.github.io/images/t/self-attention-matrix-calculation.png



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

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 $\boldsymbol{W}^O$.



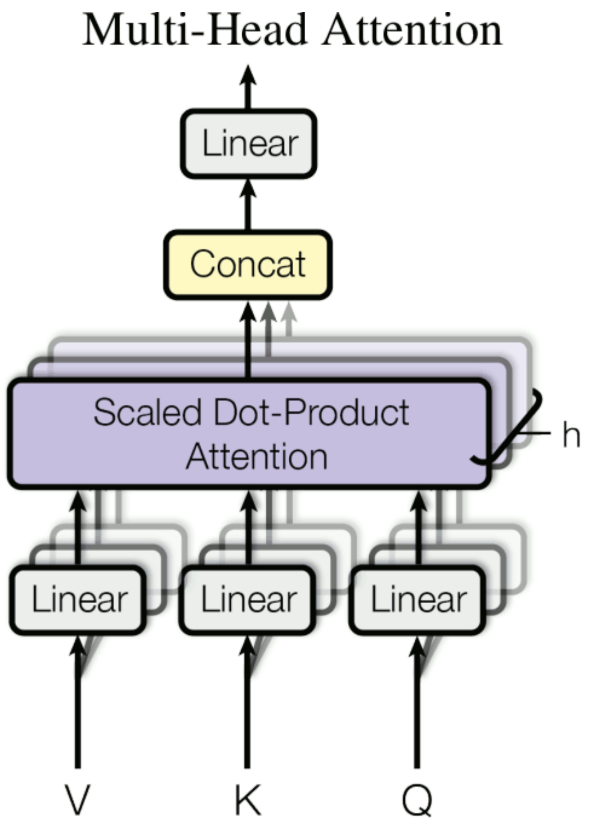
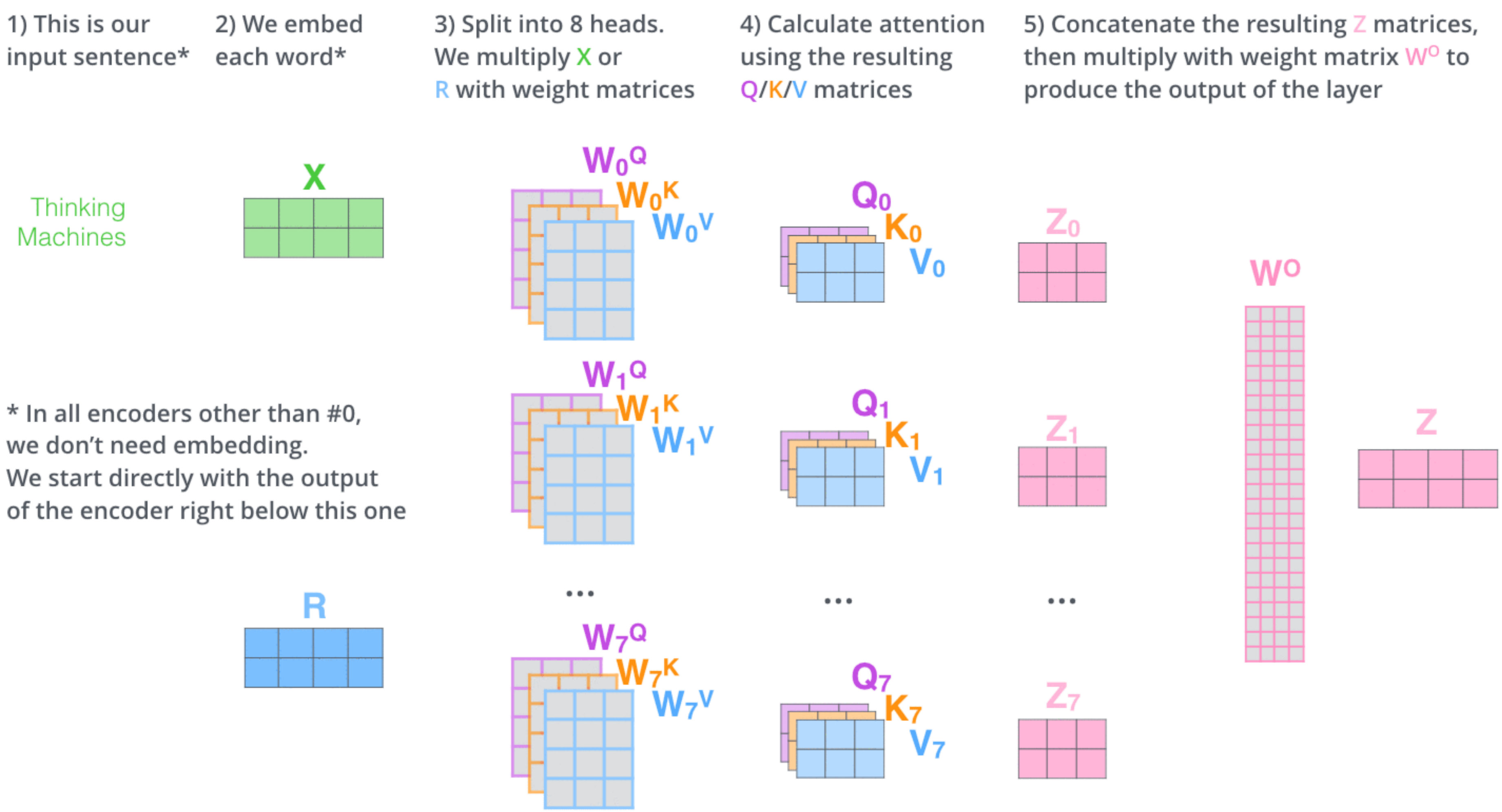
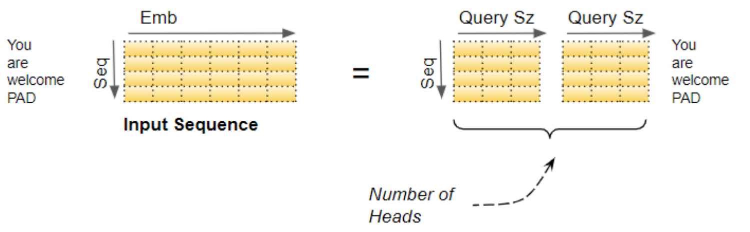


Figure 2 of "Attention Is All You Need", https://arxiv.org/abs/1706.03762

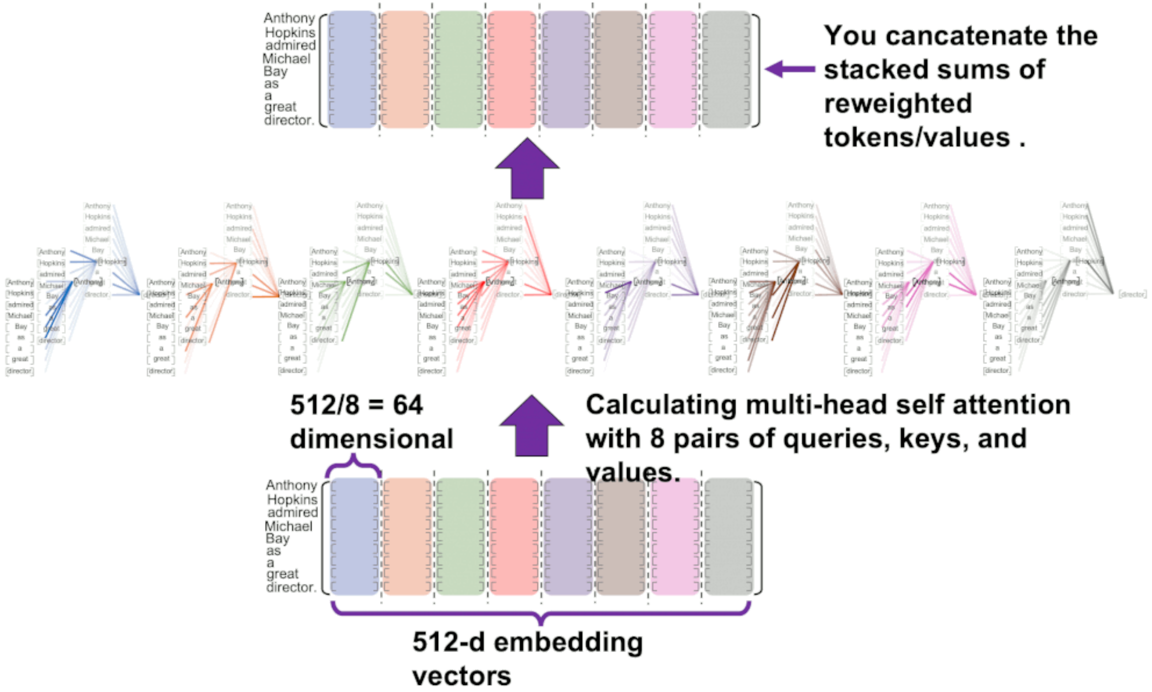


http://jalammar.github.io/images/t/transformer_multi-headed_self-attention-recap.png

When multihead attention is used, we first generate query/key/value vectors of the same dimension, and then split them into smaller pieces. Therefore, multihead attention does not increase complexity (much) and is analogous to ResNeXt/GroupNorm.



https://towardsdatascience.com/wp-content/uploads/2021/01/175EUBJLaqAMcDjgwWVh-_A.png



https://data-science-blog.com/wp-content/uploads/2022/01/mha_3-1030x608.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.

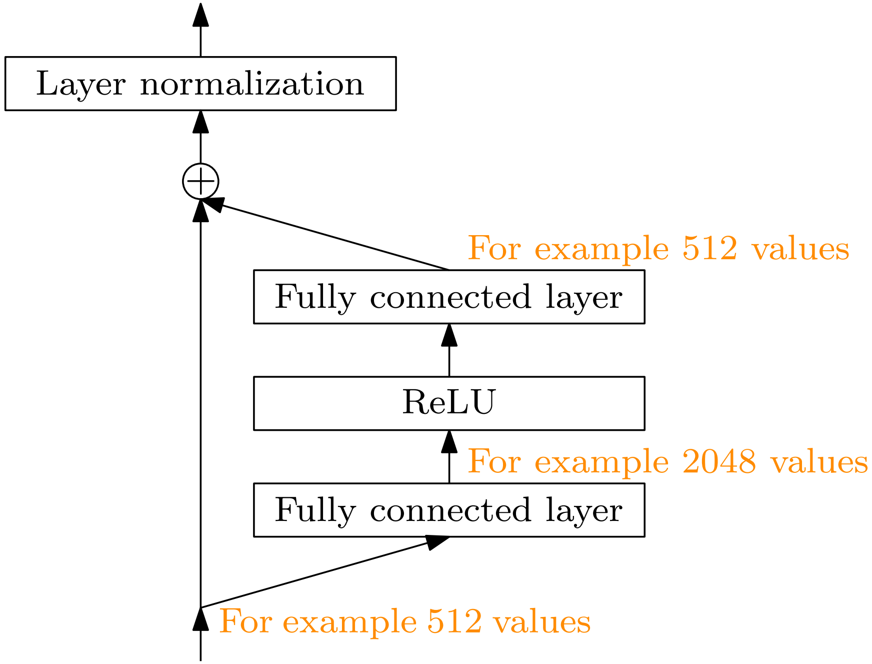
Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

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

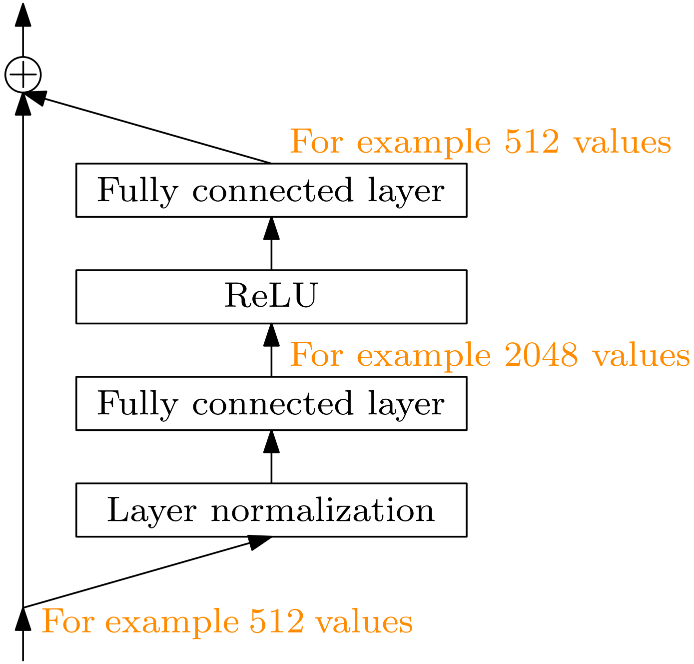
Feed Forward Networks

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.

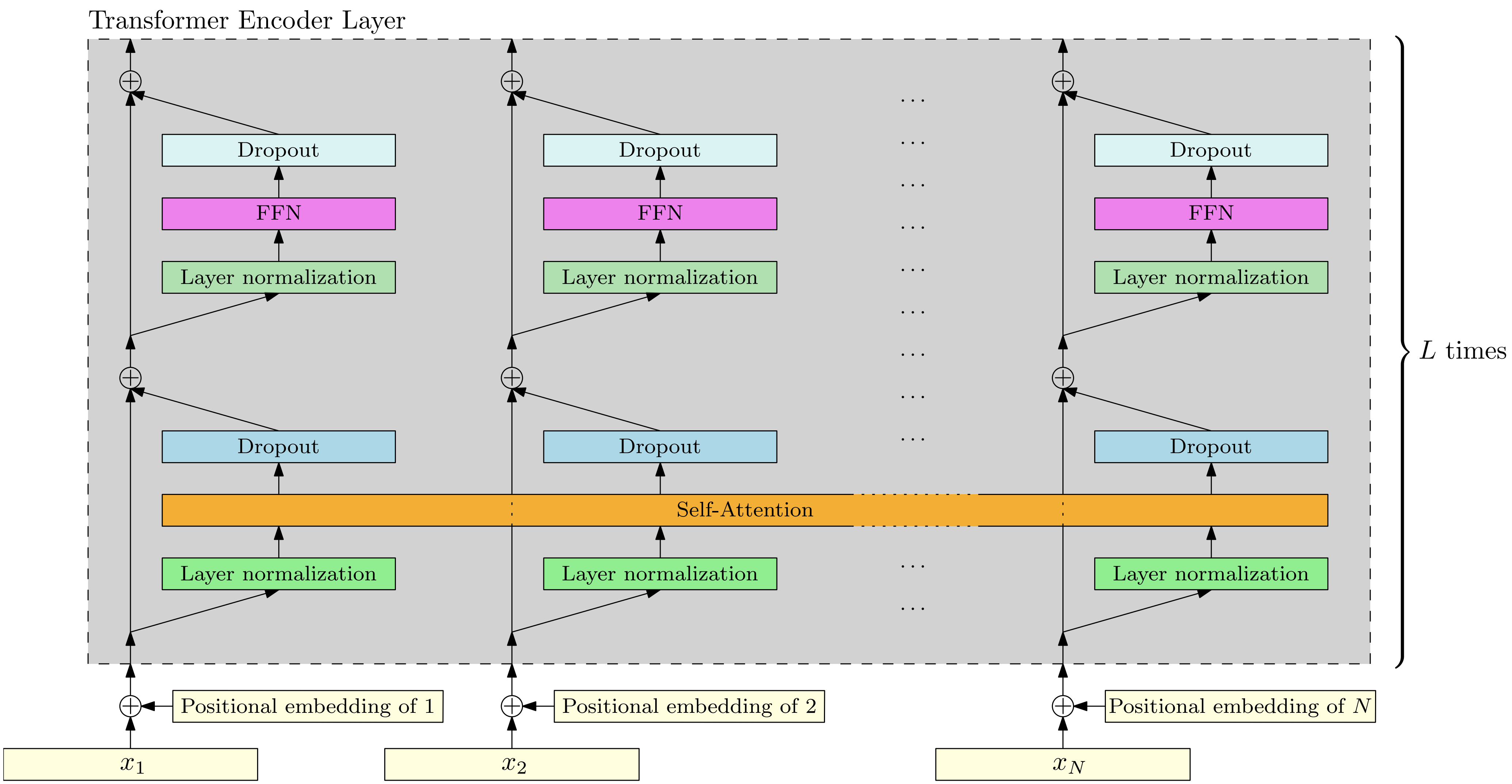
Original “Post-LN” configuration

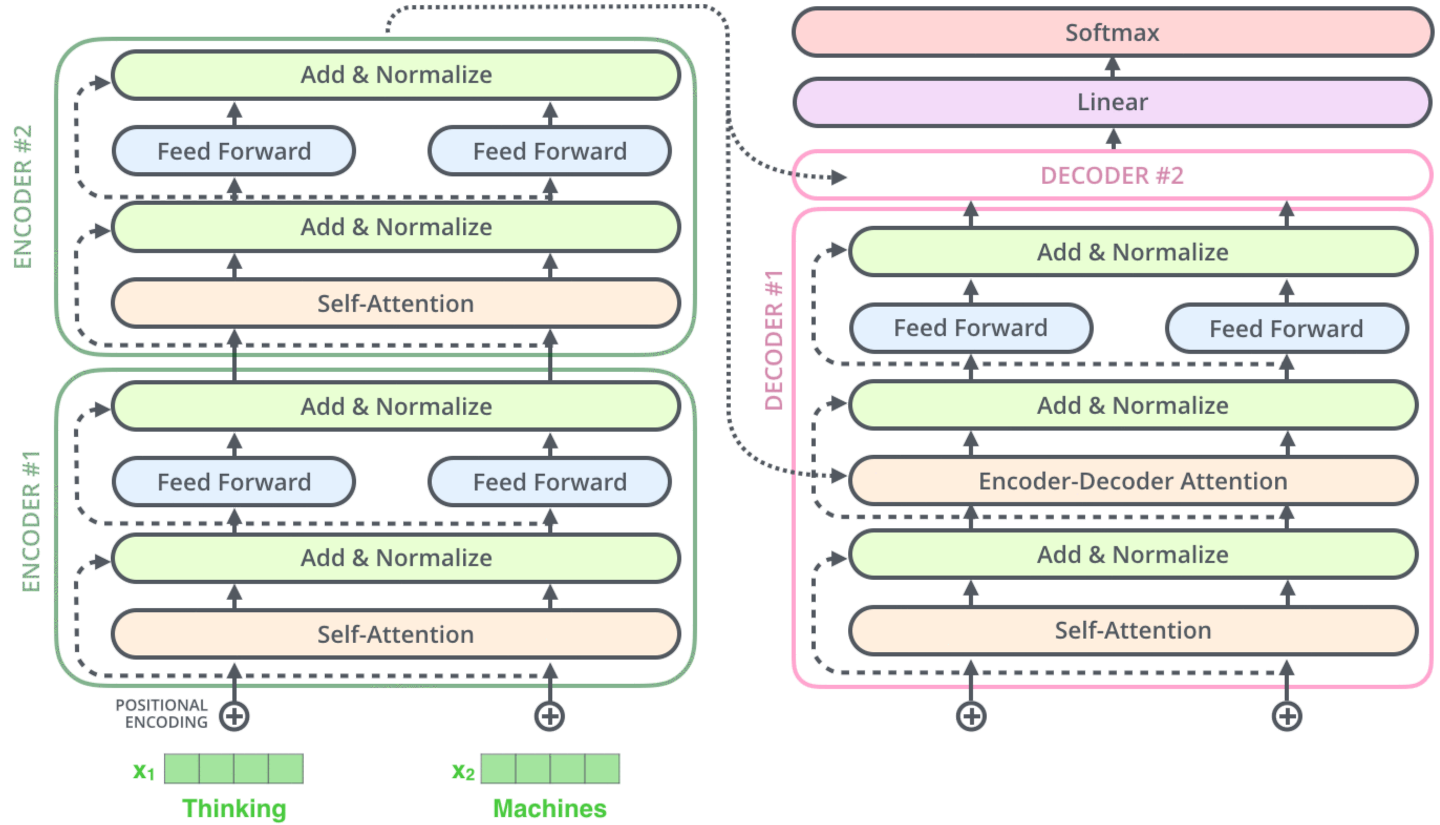


Improved “Pre-LN“ configuration since 2020



Transformer – Pre-LN Configuration





http://jalammar.github.io/images/t/transformer_resideual_layer_norm_3.png

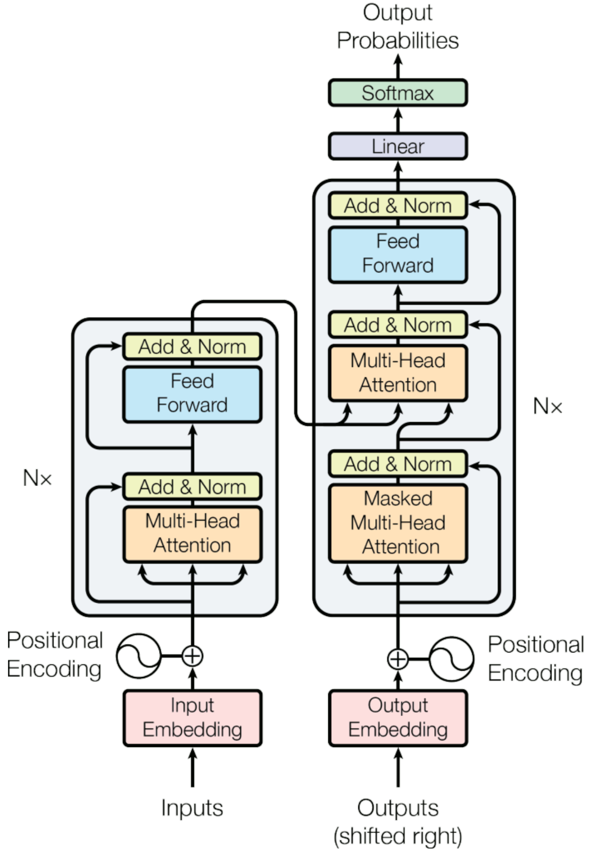


Figure 1 of "Attention Is All You Need", https://arxiv.org/abs/1706.03762

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 $\operatorname{softmax}$ calculation.

Encoder-Decoder Attention

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

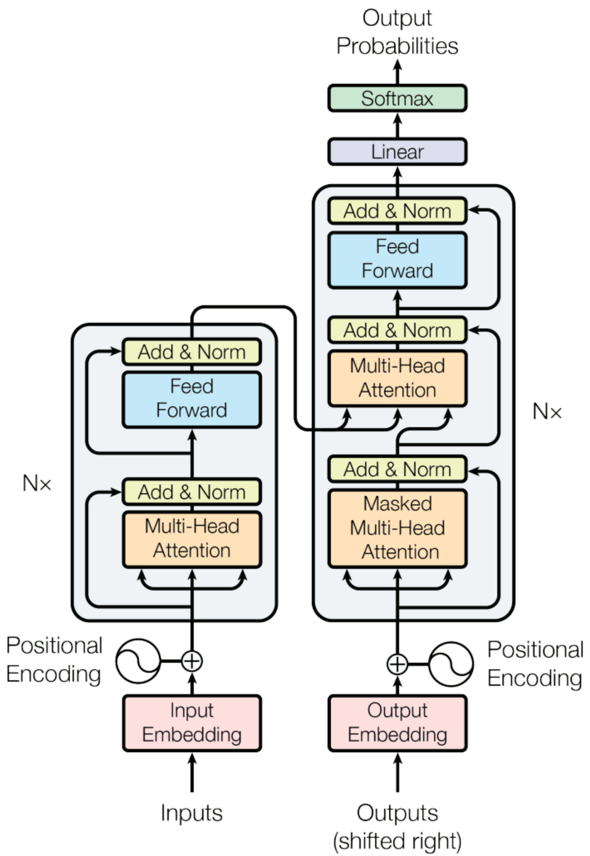
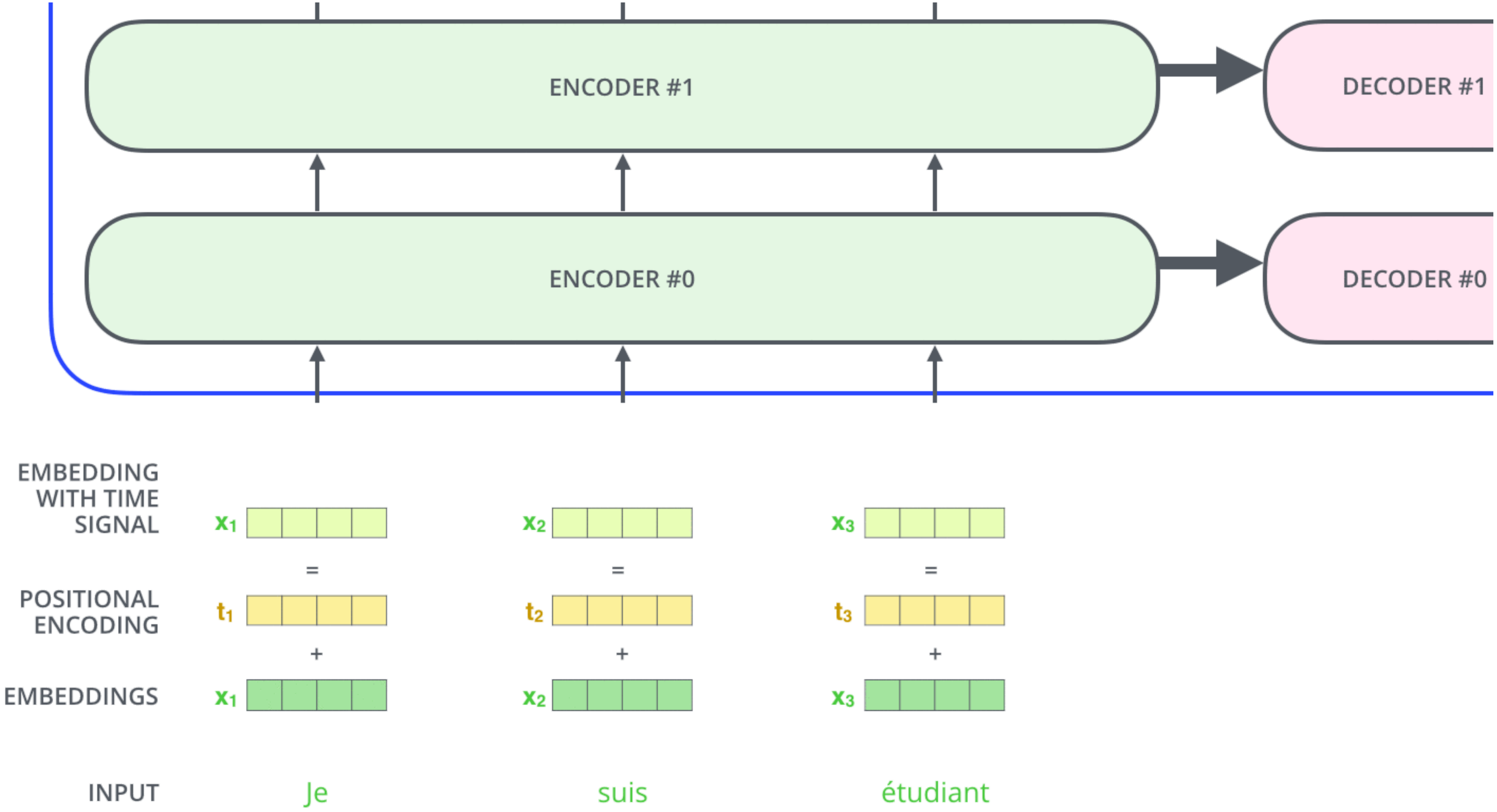


Figure 1 of "Attention Is All You Need", https://arxiv.org/abs/1706.03762

Transformer – Positional Embeddings

Transformer – Positional Embeddings



http://jalammar.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:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(pos/10000^{2i/d}\right)$$
$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d}\right)$$

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

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

Positional Embeddings

Sinusoids of different frequencies

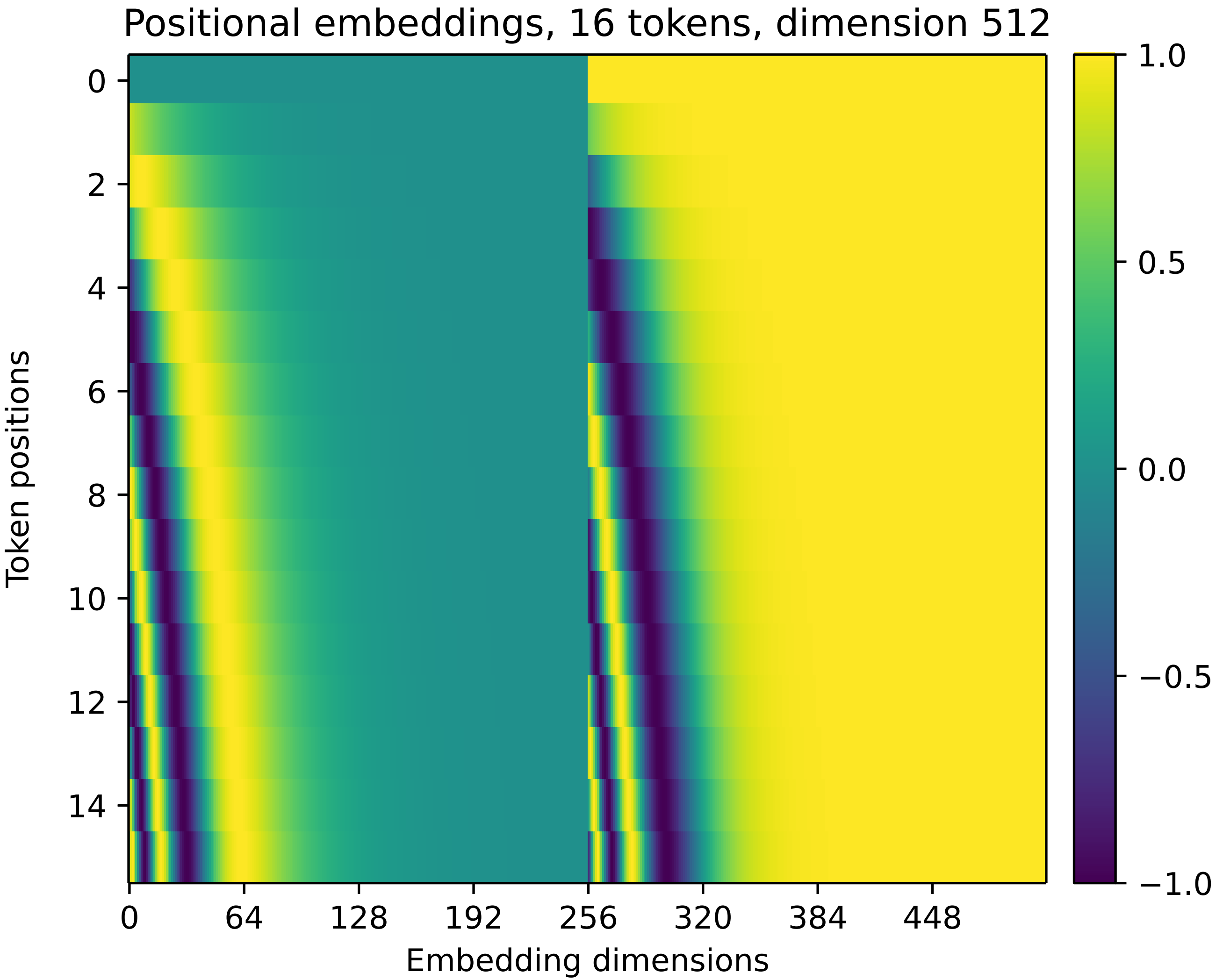
In the original description of positional embeddings (the one used on the previous slide), the sines and cosines are interleaved, so for $d = 6$, the positional embeddings would look like:

$$\mathrm{PE}_{pos} = \left(\sin\left(\frac{pos}{10000^0}\right), \cos\left(\frac{pos}{10000^0}\right), \sin\left(\frac{pos}{10000^{1/3}}\right), \cos\left(\frac{pos}{10000^{1/3}}\right), \sin\left(\frac{pos}{10000^{2/3}}\right), \cos\left(\frac{pos}{10000^{2/3}}\right)\right).$$

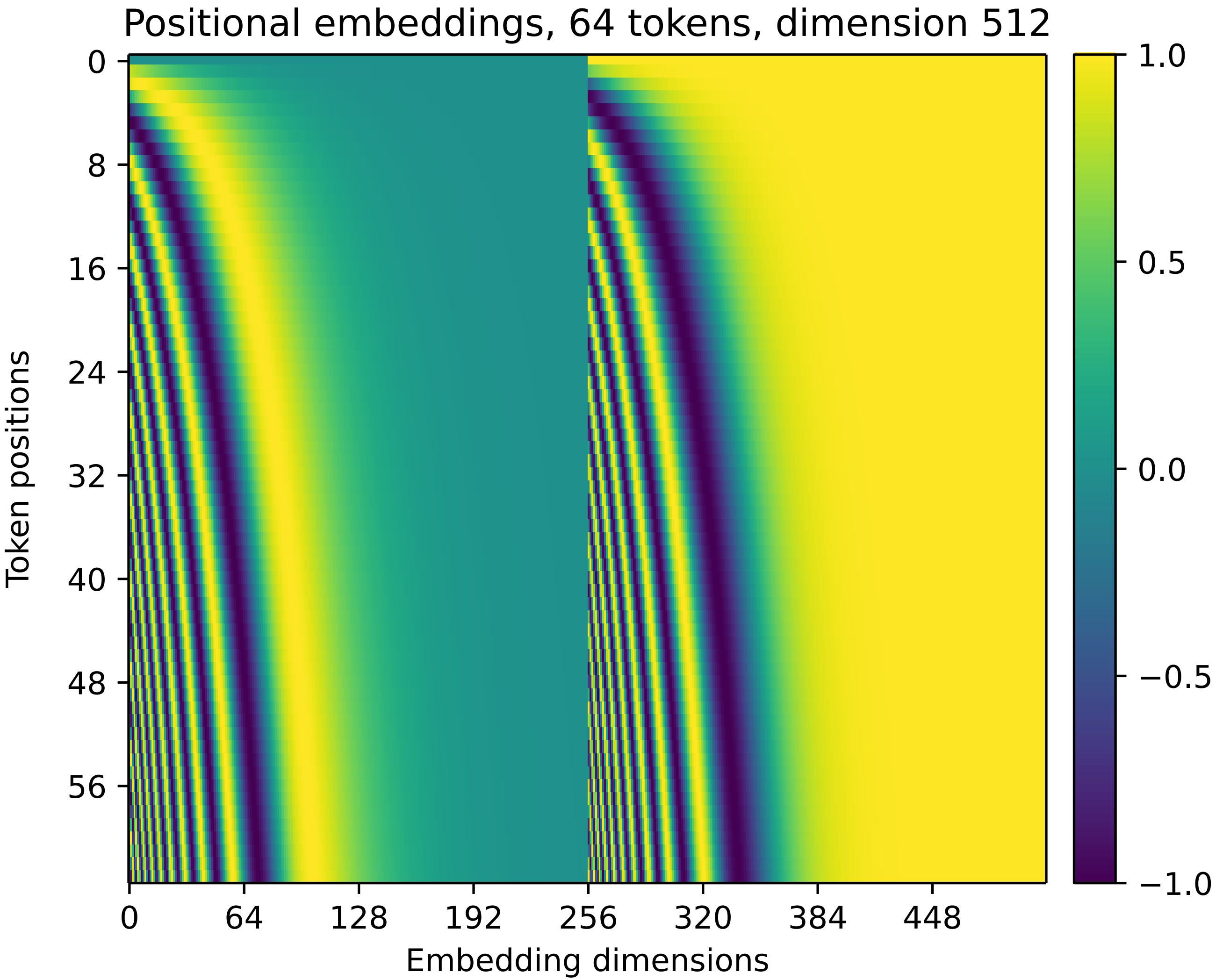
However, in practice, most implementations concatenate first all the sines and only then all the cosines:

$$\widehat{\mathrm{PE}}_{pos} = \left(\sin\left(\frac{pos}{10000^0}\right), \sin\left(\frac{pos}{10000^{1/3}}\right), \sin\left(\frac{pos}{10000^{2/3}}\right), \cos\left(\frac{pos}{10000^0}\right), \cos\left(\frac{pos}{10000^{1/3}}\right), \cos\left(\frac{pos}{10000^{2/3}}\right)\right).$$

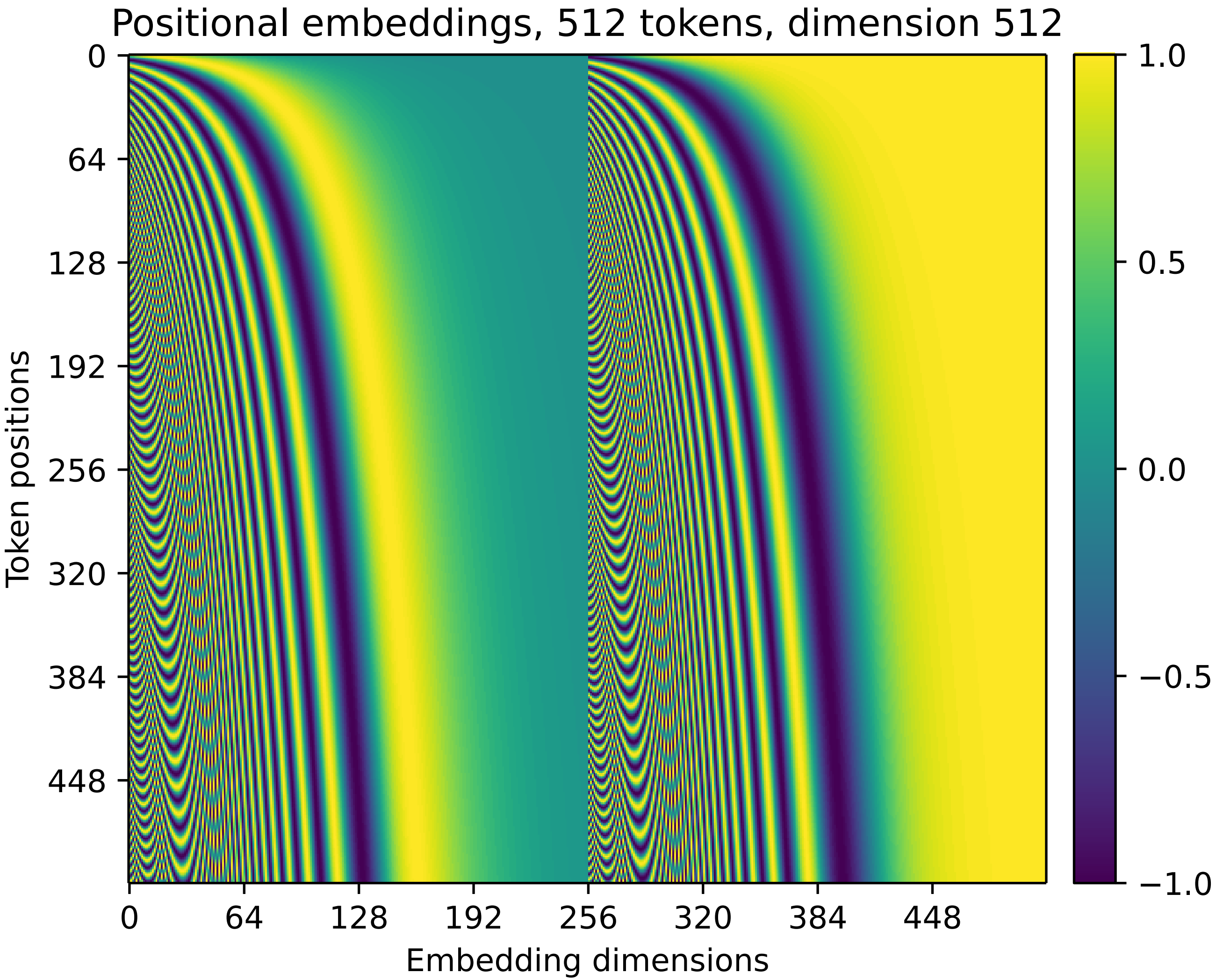
This is also how we visualize the positional embeddings on the following slides.


Positional embeddings, 16 tokens, dimension 512
Token positions
0
2
4
6
8
10
12
14
Embedding dimensions
0
64
128
192
256
320
384
448
1.0
0.5
0.0
−0.5
−1.0

Transformer – Positional Embeddings



Transformer – Positional Embeddings



Transformer – Training

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 $\mathit{warmup\_steps}$ updates, the learning rate is increased linearly from zero to its target value.

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

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.

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

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

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

Transformers Ablations on En→De newtest2014 Dev

	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

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^{loss} in our case.

Transformers – Summary

- **Seq2seq** is a general architecture of producing an output sequence from an input sequence.
- It is usually trained using **teacher forcing**, and use **autoregressive decoding**.
- **Attention** allows focusing on any part of a sequence in every time step.
- **Transformer** provides more powerful sequence-to-sequence architecture and also sequence element representation architecture compared to **RNNs**, but requires **substantially more** data.
  - When data are plentiful, best models for processing text, speech, and vision data utilize the Transformer architecture (together with convolutions in the vision domain).
- In seq2seq architecture, we have both an **encoder** and the **decoder**. However, text generation (i.e., in chatbots) is usually performed by **decoder-only** models.

BERT

A year later after ELMo, at the end of 2018, a new model called BERT (standing for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers) was proposed. It is nowadays one of the most dominating approaches for pre-training word embeddings and for paragraph and document representations.

https://www.sesameworkshop.org/sites/default/files/imageservicecache/2019-03/header5120x1620_50thanniversary.png/4b00e17bb509f5c630c57c318b37d0da.webp

The BERT model computes contextualized representations using a bidirectional Transformer architecture.

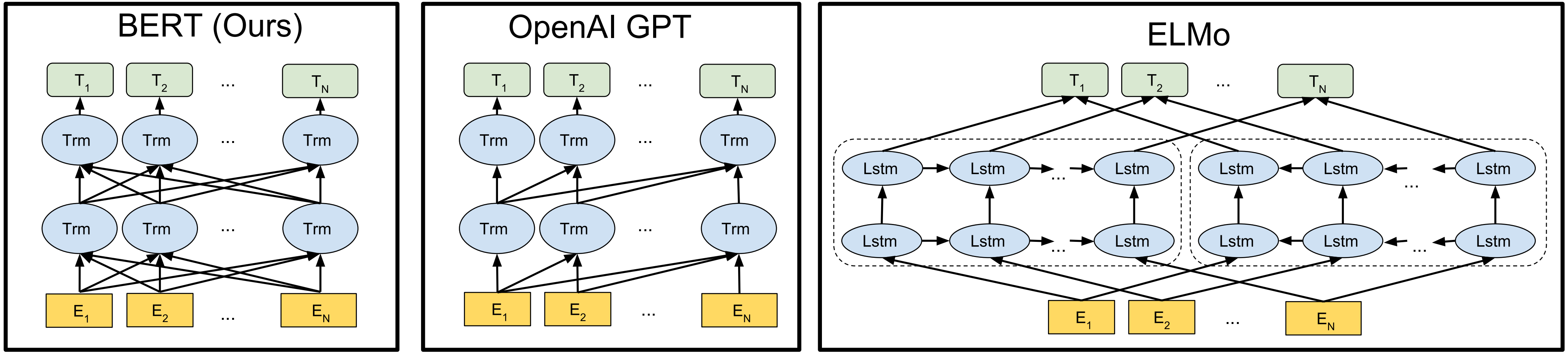
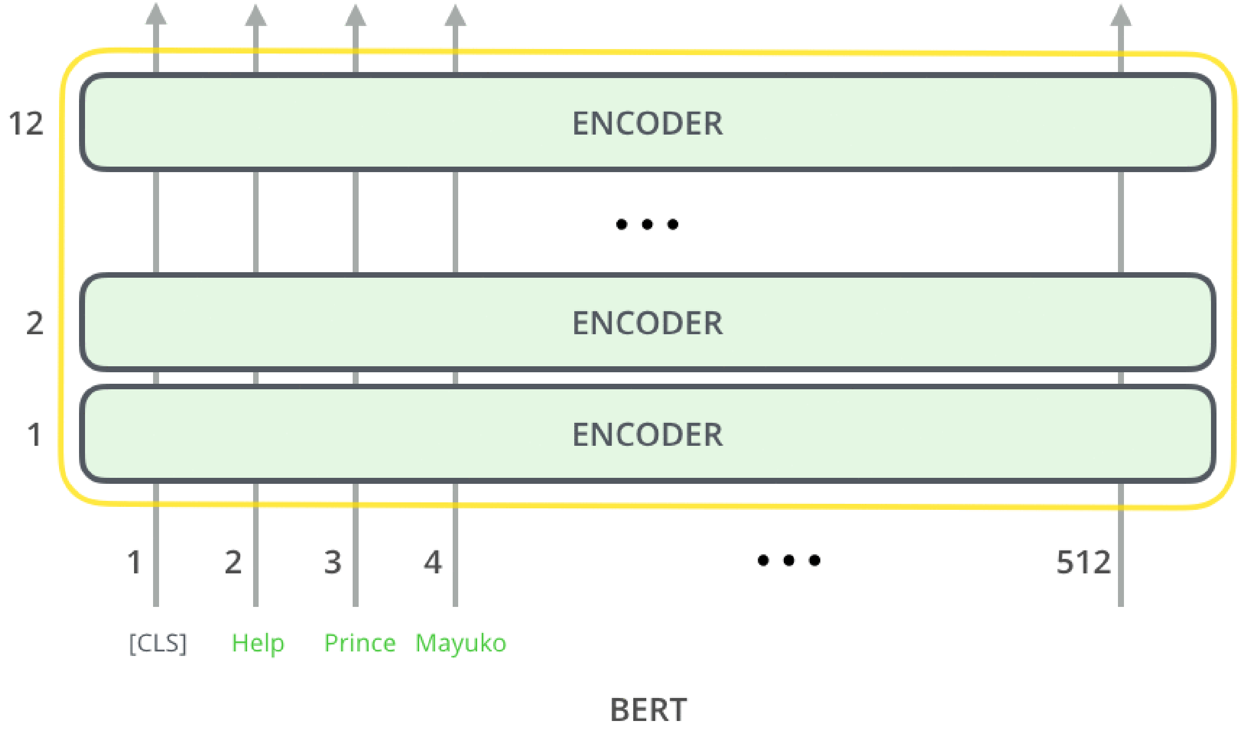


Figure 3 of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805

The baseline BERT base model consists of 12 Transformer layers:



http://jalammar.github.io/images/bert-encoders-input.png

The bidirectionality is important, but it makes training difficult.

BERT Input

The input of the BERT model is a sequence of subwords, namely their identifiers. This input represents two so-called *sentences*, but they are in fact pieces of text with hundreds of subwords (512 maximum in total). The first token is a special `CLS` token and every sentence is ended by a `SEP` token.

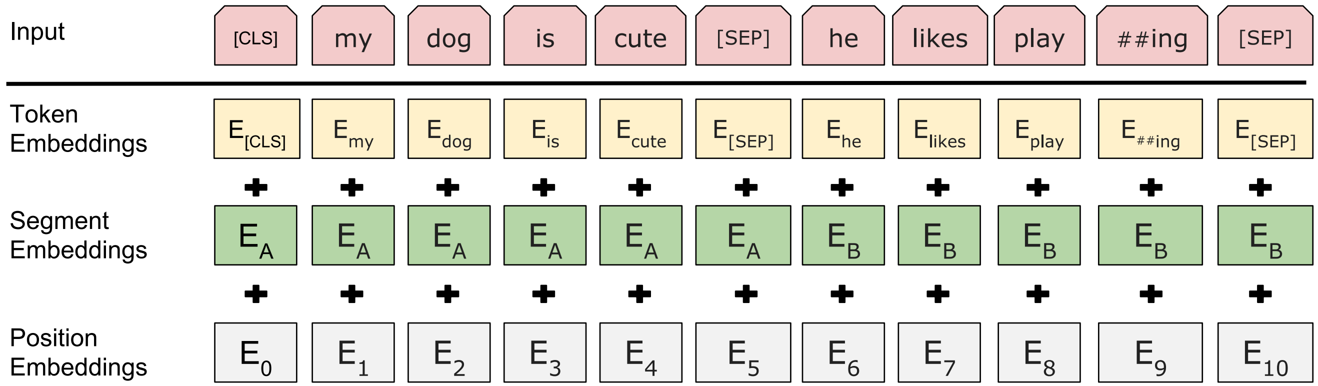


Figure 2 of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805

Every subword representation is a sum of:

- trainable subword embeddings,
- trainable positional embeddings (not the sinusoidal embeddings, but I do not know why),
- trainable segment embeddings, which indicate if a token belongs to a sentence `A` (inclusively up to its `SEP` token) or to sentence `B`.

The BERT model is pretrained using two objectives:

- **masked language model** – 15% of the input words are **masked**, and the model tries to predict them (using a head consisting of a fully connected layer with softmax activation);
  - 80% of them are replaced by a special `MASK` token;
  - 10% of them are replaced by a random word;
  - 10% of them are left intact.
- **next sentence prediction** – the model tries to predict whether the second *sentence* followed the first one in the raw corpus (using a head that on top of the `CLS` output adds a fully connected layer with tanh activation (*pooler*), followed by a softmax-activated fully connected layer with two outputs).
  - 50% of the time the second sentence is the actual next sentence;
  - 50% of the time the second sentence is a random sentence from the corpus.

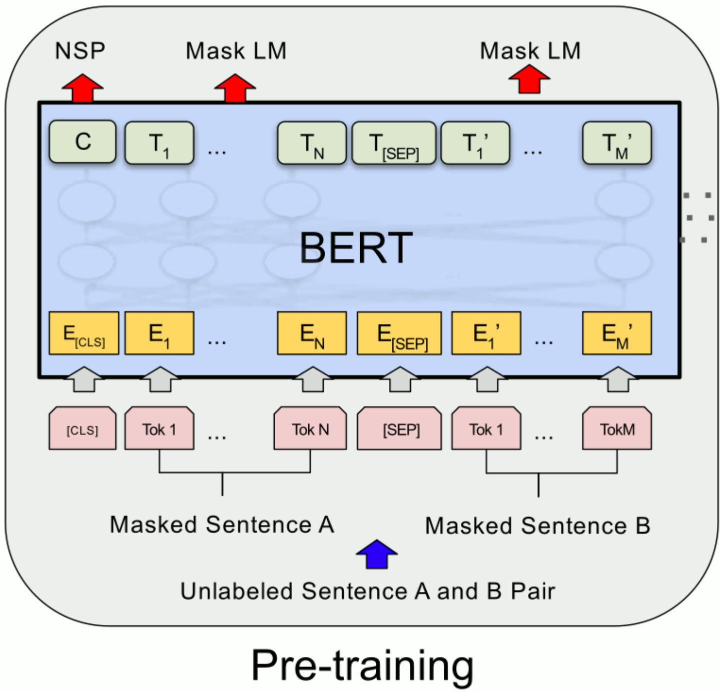


Figure 1 of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805

BERT Pretraining

For pre-training, English BookCorpus (800M words) and Wikipedia (2,500M words) are used, with a 30k WordPieces vocabulary.

Batch size is 256 sequences, each 512 subwords, giving 128k tokens per batch. Adam with learning rate 1e-4 is used, with linear learning rate warmup for the first 10k steps, followed by a linear learning rate decay to 0. Standard momentum parameters are used, and L^2 weight decay of 0.01 is utilized.

Dropout of 0.1 on all layers is used, and GELU activation is used instead of ReLU.

Furthermore, because longer sequences are quadratically more expensive, first 90% of the pre-training is performed on sequences of length 128, and only the last 10% use sequences of length 512.

Two variants are considered:

- BERT *base* with 12 layers, 12 attention heads and hidden size 768 (110M parameters),
- BERT *large* with 24 layers, 16 attention heads and hidden size 1024 (340M parameters).

ReLU multiplies the input by zero or one, depending on its value.

Dropout stochastically multiplies the input by zero or one.

Both these functionalities are merged in Gaussian error linear units (GELUs), where the input value is multiplied by $m \sim \mathrm{Bernoulli}(\Phi(x))$, where $\Phi(x) = P(x' \leq x)$ for $x' \sim \mathcal{N}(0, 1)$ is the cumulative density function of the standard normal distribution.

The GELUs compute the expectation of this value, i.e.,

$$\mathrm{GELU}(x) = x \cdot \Phi(x) + 0 \cdot \big(1 - \Phi(x)\big) = x\Phi(x).$$

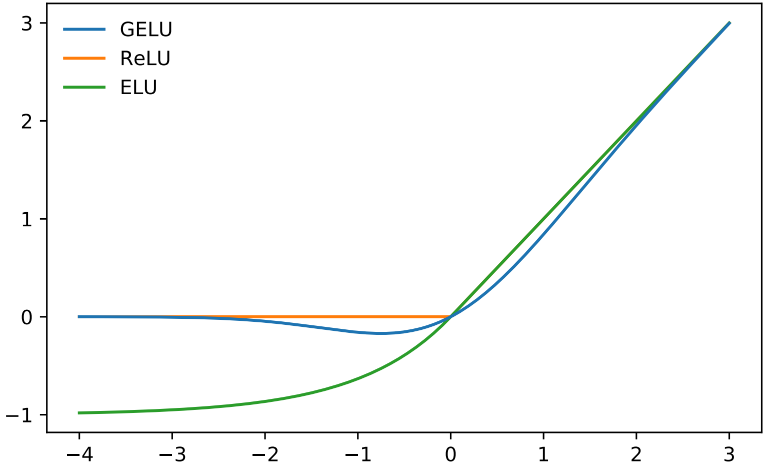


Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

Figure 1 of "Gaussian Error Linear Units (GELUs)", https://arxiv.org/abs/1606.08415

GELUs can be approximated using (no need to remember this):

$$0.5x\left(1 + \tanh\left[\sqrt{2/\pi}(x + 0.044715x^3)\right]\right) \text{ or } x\sigma(1.702x).$$

The pre-trained BERT model can be finetuned on a range of tasks:

- **sentence element representation**
  - PoS tagging
  - named entity recognition
  - …
- **sentence representation**
  - text classification
- **sentence relation representation**
  - textual entailment, aka natural language inference (the second sentence is *implied by/contradicts/has no relation to* the first sentence)
  - textual similarity
  - paraphrase detection

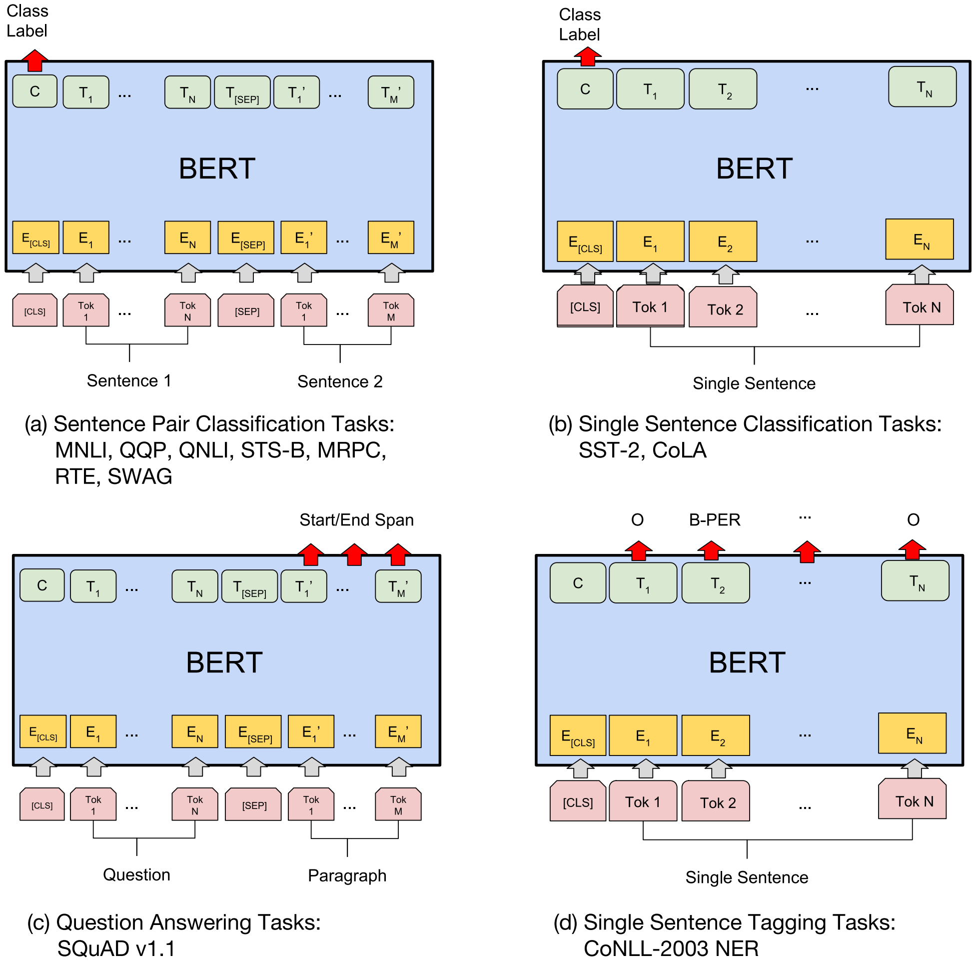


Figure 4 of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805

For finetuning, dropout 0.1 is used, usually very small number of epochs (2-4) suffice, and a good learning rate is usually one of 5e-5, 3e-5, 2e-5.

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
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

Table 1 of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805

System	Dev EM	Dev F1	Test EM	Test F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
BERT_BASE (Single)	80.8	88.5	-	-
BERT_LARGE (Single)	84.1	90.9	-	-
BERT_LARGE (Ensemble)	85.8	91.8	-	-
BERT_LARGE (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
BERT_LARGE (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Table 2: SQuAD 1.1 results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

Table 2 of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805

System	Dev EM	Dev F1	Test EM	Test F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	86.3	89.0	86.9	89.5
#1 Single - MIR-MRC (F-Net)	-	-	74.8	78.0
#2 Single - nlnet	-	-	74.2	77.1
Published				
unet (Ensemble)	-	-	71.4	74.9
SLQA+ (Single)	-		71.4	74.4
Ours				
BERT_LARGE (Single)	78.7	81.9	80.0	83.1

Table 3: SQuAD 2.0 results. We exclude entries that use BERT as one of their components.

Table 3 of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
BERT_BASE	81.6	-
BERT_LARGE	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

Table 4: SWAG Dev and Test accuracies. †Human performance is measured with 100 samples, as reported in the SWAG paper.

Table 4 of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805

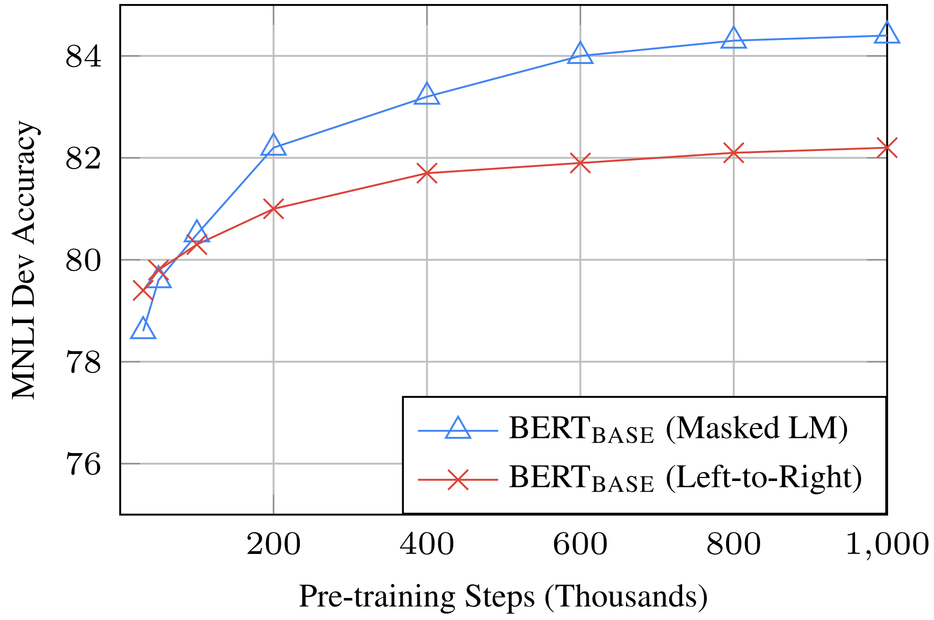


Figure 5: Ablation over number of training steps. This shows the MNLI accuracy after fine-tuning, starting from model parameters that have been pre-trained for k steps. The x-axis is the value of k.

Figure 5 of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805

Masking Rates			Dev Set Results		
MASK	SAME	RND	MNLI Fine-tune	NER Fine-tune	NER Feature-based
80%	10%	10%	84.2	95.4	94.9
100%	0%	0%	84.3	94.9	94.0
80%	0%	20%	84.1	95.2	94.6
80%	20%	0%	84.4	95.2	94.7
0%	20%	80%	83.7	94.8	94.6
0%	0%	100%	83.6	94.9	94.6

Table 8: Ablation over different masking strategies.

Table 8 of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805

Multilingual BERT

Multilingual BERT

The Multilingual BERT is pre-trained on 102-104 largest Wikipedias, including the Czech one.

There are two versions, the *cased* one has WordPieces including case, and the *uncased* one with subwords all in lower case and *without diacritics*.

Even if only very small percentage of input sentences were Czech, it works surprisingly well for Czech NLP.

Furthermore, without any explicit supervision, mBERT is able to represent the input languages in a *shared* space, allowing cross-lingual transfer.

Consider a *reading comprehension* task, where for a given paragraph and a question an answer needs to be located in the paragraph.

Then training the model in English and then directly running it on a different language works comparably to translating the data to English and then back.

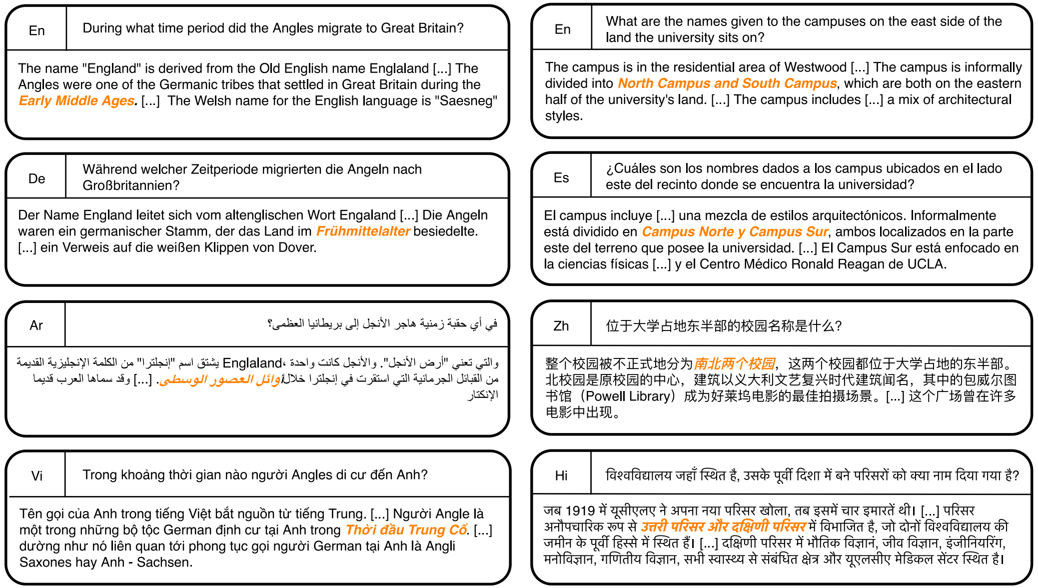


Figure 2 of "MLQA: Evaluating Cross-lingual Extractive Question Answering", https://arxiv.org/abs/1910.07475

F1 / EM	en	es	de	ar	hi	vi	zh
BERT-Large	**80.2 / 67.4**	-	-	-	-	-	-
Multilingual-BERT	77.7 / 65.2	64.3 / 46.6	57.9 / 44.3	45.7 / 29.8	43.8 / 29.7	57.1 / 38.6	57.5 / 37.3
XLM	74.9 / 62.4	**68.0 / 49.8**	**62.2 / 47.6**	**54.8 / 36.3**	48.8 / 27.3	61.4 / 41.8	61.1 / **39.6**
Translate test, BERT-L	-	65.4 / 44.0	57.9 / 41.8	33.6 / 20.4	23.8 / 18.9*	58.2 / 33.2	44.2 / 20.3
Translate train, M-BERT	-	53.9 / 37.4	62.0 / 47.5	51.8 / 33.2	**55.0 / 40.0**	**62.0 / 43.1**	**61.4** / 39.5
Translate train, XLM	-	65.2 / 47.8	61.4 / 46.7	54.0 / 34.4	50.7 / 33.4	59.3 / 39.4	59.8 / 37.9

Table 5 of "MLQA: Evaluating Cross-lingual Extractive Question Answering", https://arxiv.org/abs/1910.07475

Best Multilingual Encoders

Currently available multilingual (100+ languages) encoder models, evaluated on XNLI dataset (Multi-Genre NLI on 15 languages; models trained on English, evaluated on other languages)

Model	Parameters	XNLI (Avg)
mbert-base	178M	65.4
xlm-roberta-base		76.2
xlm-roberta-large		80.9
xlm-roberta-xl		82.3
xlm-roberta-xxl		83.1
rembert		
mt5-base		75.4
mt5-large		81.1
mt5-xl		82.9
mt5-xxl		84.5

RoBERTa (Robustly Optimized BERT)

The *next sentence prediction* was originally hypothesized to be an important factor during training of the BERT model, as indicated by ablation experiments. However, later experiments indicated removing it might improve results.

The RoBERTa authors therefore performed the following experiments:

- SEGMENT-PAIR: pair of segments with at most 512 tokens in total;
- SENTENCE-PAIR: pair of *natural sentences*, usually significantly shorter than 512 tokens;
- FULL-SENTENCES: just one segment on input with 512 tokens, can cross document boundary;
- DOC-SENTENCES: just one segment on input with 512 tokens, cannot cross document boundary.

Model	**SQuAD 1.1/2.0**	**MNLI-m**	**SST-2**	**RACE**
Our reimplementation (with NSP loss):				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
Our reimplementation (without NSP loss):				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6
$\text{BERT}_{\text{BASE}}$	88.5/76.3	84.3	92.8	64.3
$\text{XLNet}_{\text{BASE}}$ (K = 7)	–/81.3	85.8	92.7	66.1
$\text{XLNet}_{\text{BASE}}$ (K = 6)	–/81.0	85.6	93.4	66.7

Table 2 of "RoBERTa: A Robustly Optimized BERT Pretraining Approach", https://arxiv.org/abs/1907.11692

RoBERTa – Larger Batches

BERT is trained for 1M steps with a learning rate of 1e-4.

The RoBERTa authors also considered larger batches (with linearly larger learning rate).

bsz	steps	lr	ppl	MNLI-m	SST-2
256	1M	1e-4	3.99	84.7	92.7
2K	125K	7e-4	**3.68**	**85.2**	**92.9**
8K	31K	1e-3	3.77	84.6	92.8

Table 3: Perplexity on held-out training data (*ppl*) and development set accuracy for base models trained over BOOKCORPUS and WIKIPEDIA with varying batch sizes (*bsz*). We tune the learning rate (*lr*) for each setting. Models make the same number of passes over the data (epochs) and have the same computational cost.

Table 3 of "RoBERTa: A Robustly Optimized BERT Pretraining Approach", https://arxiv.org/abs/1907.11692

The RoBERTa model, **R**obustly **o**ptimized **BERT** **a**pproach, is trained with dynamic masking, FULL-SENTENCES without NSP, large 8k minibatches and byte-level BPE with 50k subwords.

Model	**data**	**bsz**	**steps**	**SQuAD** (v1.1/2.0)	**MNLI-m**	**SST-2**
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

Table 4: Development set results for RoBERTa as we pretrain over more data (16GB → 160GB of text) and pretrain for longer (100K → 300K → 500K steps). Each row accumulates improvements from the rows above. RoBERTa matches the architecture and training objective of $\text{BERT}_{\text{LARGE}}$. Results for $\text{BERT}_{\text{LARGE}}$ and $\text{XLNet}_{\text{LARGE}}$ are from Devlin et al. (2019) and Yang et al. (2019), respectively. Complete results on all GLUE tasks can be found in the Appendix.

Table 4 of "RoBERTa: A Robustly Optimized BERT Pretraining Approach", https://arxiv.org/abs/1907.11692

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
$\text{BERT}_{\text{LARGE}}$	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
$\text{XLNet}_{\text{LARGE}}$	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**	**91.3**	-
Ensembles on test (from leaderboard as of July 25, 2019)										
ALICE	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	**96.8**	**93.0**	67.8	91.6	**90.4**	88.4
RoBERTa	**90.8/90.2**	**98.9**	90.2	**88.2**	96.7	92.3	67.8	**92.2**	89.0	**88.5**

Table 5: Results on GLUE. All results are based on a 24-layer architecture. $\text{BERT}_{\text{LARGE}}$ and $\text{XLNet}_{\text{LARGE}}$ results are from Devlin et al. (2019) and Yang et al. (2019), respectively. RoBERTa results on the development set are a median over five runs. RoBERTa results on the test set are ensembles of *single-task* models. For RTE, STS and MRPC we finetune starting from the MNLI model instead of the baseline pretrained model. Averages are obtained from the GLUE leaderboard.

Table 5 of "RoBERTa: A Robustly Optimized BERT Pretraining Approach", https://arxiv.org/abs/1907.11692

Model	SQuAD 1.1		SQuAD 2.0	
	EM	F1	EM	F1
Single models on dev, w/o data augmentation				
$\text{BERT}_{\text{LARGE}}$	84.1	90.9	79.0	81.8
$\text{XLNet}_{\text{LARGE}}$	**89.0**	94.5	86.1	88.8
RoBERTa	88.9	**94.6**	**86.5**	**89.4**
Single models on test (as of July 25, 2019)				
$\text{XLNet}_{\text{LARGE}}$			$86.3^\dagger$	$89.1^\dagger$
RoBERTa			86.8	89.8
XLNet + SG-Net Verifier			$\mathbf{87.0}^\dagger$	$\mathbf{89.9}^\dagger$

Table 6 of "RoBERTa: A Robustly Optimized BERT Pretraining Approach", https://arxiv.org/abs/1907.11692

Model	Accuracy	Middle	High
Single models on test (as of July 25, 2019)			
$\text{BERT}_{\text{LARGE}}$	72.0	76.6	70.1
$\text{XLNet}_{\text{LARGE}}$	81.7	85.4	80.2
RoBERTa	**83.2**	**86.5**	**81.3**

Table 7: Results on the RACE test set. $\text{BERT}_{\text{LARGE}}$ and $\text{XLNet}_{\text{LARGE}}$ results are from Yang et al. (2019).

Table 7 of "RoBERTa: A Robustly Optimized BERT Pretraining Approach", https://arxiv.org/abs/1907.11692

One Transformer to Rule Them All

The Transformer architecture has been spreading and has become almost omnipresent in tasks where data is available in abundance, in which case it usually delivers superior performance.

Large Language Models

- GPT: Transformer decoder-only model, 2018, ~150M parameters
- GPT-2: Transformer decoder-only model, 2019, ~1.5B parameters
- GPT-3: Transformer decoder-only model, May 2020, ~175B parameters
- GPT-4: Transformer decoder-only model, March 2023, ???; $100M
- Gopher: Transformer decoder-only model, Dec 2021, ~280B parameters
- XGLM: multilingual Transformer decoder-only model, Dec 2021, ~7.5B parameters
  - 30 languages from 16 language families
- Megatron-Turing NLP: Transformer decoder-only model, Jan 2022, ~530B parameters
  - trained using 2240 A100
- PaLM: Transformer decoder-only model, Apr 2022, ~580B parameters
  - trained using 6144 TPUv4; 22% training data non-English

PaLM Jokes Explanations

Explaining a Joke

```
I will explain these jokes:
(1) The problem with kleptomaniacs is that they always take things literally.
Explanation: This joke is wordplay. Someone who "takes things literally" is someone who doesn't fully understand social cues and context, which is a negative trait.
But the definition of kleptomania is someone who literally takes things.
(2) Always borrow money from a pessimist. They'll never expect it back.
Explanation: Most people expect you to pay them back when you borrow money, however a pessimist is someone who always assumes the worst, so if you borrow money from
them, they will expect that you won't pay them back anyways.
```

Input: I was supposed to start writing the paper at 5:00 PM. But then I started playing with this cool new language model for 10 minutes. 10 minutes later, it's suddenly 9:30 PM!
Model Output: This joke is about how time flies when you're having fun. The person was supposed to start writing a paper at 5:00 PM, but they started playing with a language model instead. They thought they were only playing with it for 10 minutes, but it was actually 4.5 hours.

Input: I think I understand why me and my partner are such a good match. I get excited when I smell food and become super alert when the doorbell rings. My girlfriend Shelley becomes ecstatic when she sees a person who she just saw 5 hours ago.
Model Output: This joke is about dogs. Dogs get excited when they smell food and when the doorbell rings. The girlfriend is excited to see someone she just saw 5 hours ago, which is a trait of dogs.

Input: I was going to fly to visit my family on April 6th. My mom said "Oh great, your step dad's poetry reading is that night!" So now I'm flying in on April 7th.
Model Output: The joke is that the speaker's mother is trying to get them to go to their step dad's poetry reading, but the speaker doesn't want to go, so they are changing their flight to the day after the poetry reading.

Input: I tried 10,000 random restarts of my neural network, but I was accused of overfitting. I guess no good seed goes unpunished.
Model Output: This joke is a pun. A neural network is a computer program that can learn from data. A "seed" is a number that is used to initialize a random number generator. A "good seed" is a number that produces a good random number generator. The phrase "no good deed goes unpunished" means that if you do something good, you will be punished for it.

Input: Reading "War and Peace" doesn't scare me. I once had to review the CL for my intern's whole project on his final day!
Model Output: 'War and Peace' is a very long book, so reading it is a daunting task. However, the speaker is saying they had to review the code for their intern's project on their final day, which is even more daunting.

Input: What's the difference between a zebra and an umbrella? One is a striped animal related to horses, another is a device you use to stop the rain from falling on you.
Model Output: This joke is an anti-joke. The joke is that the answer is obvious, and the joke is that you were expecting a funny answer.

Figure 19 of "PaLM: Scaling Language Modeling with Pathways", https://arxiv.org/abs/2204.02311

Language Models and Instruction Finetuning

There are nowadays many large language models of various sizes, some of them open-source:

- Llama 1, 2, 3, 4: by Meta, publicly available
- Mistral, Mixtral: by Mistral AI (founded in France), some models available
- Claude 1, 2, 3, 3.5, 3.7: by Anthropic
- Gemma and Gemini: by Alphabet; previously PaLM and LaMDA
- BLOOM, OPT, Falcon, ...

See the Large Language Models course https://ufal.mff.cuni.cz/courses/npfl140 if you want to know more.

Pre-trained Encoder-Decoder Models

Various pre-trained encoder-decoder models are available:

- BART, 2019, ~200M parameters
- T5, Oct 2019, up to ~11B parameters
  - REALM, Feb 2020, ~330M, uses explicit retrieval from a large knowledge base

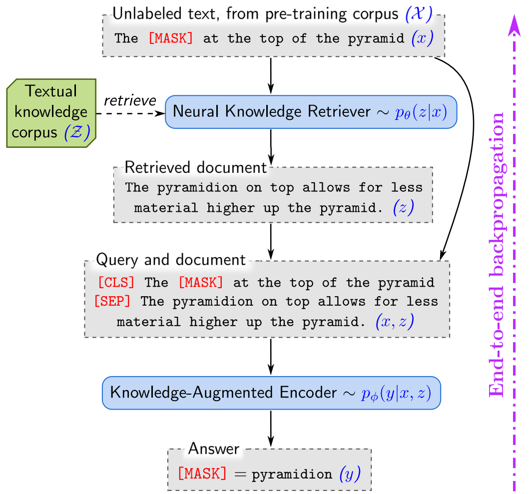


Figure 1 of "REALM: Retrieval-Augmented Language Model Pre-Training", https://arxiv.org/pdf/2002.05202.pdf

- mT5, Oct 2020, ~100 languages, up to ~13B parameters
  - sizes small (300M), base (582M), large (1.23B), xl (3.74B), xxl (12.9B)
- ByT5, May 2021, byte-based, ~100 languages, same sizes as mT5

Instruction Following aka Chatbots

The large language models can be finetuned for conversational interactions, which is called **instruction finetuning**.

The original approach was to use the so-called Reinforcement Learning from Human Feedback (RLHF); lately, other approaches like DPO have appeared.

- ChatGPT: OpenAI; together with DeepMind the creator of RLHF
- Gemini: Alphabet
- Llama 4 Instruct
- Claude 3.7 Sonnet
- DeepSeek-V3-0324, DeepSeek-R1: current best open-source models (as of April 22, 2025, according to https://lmarena.ai/?leaderboard)

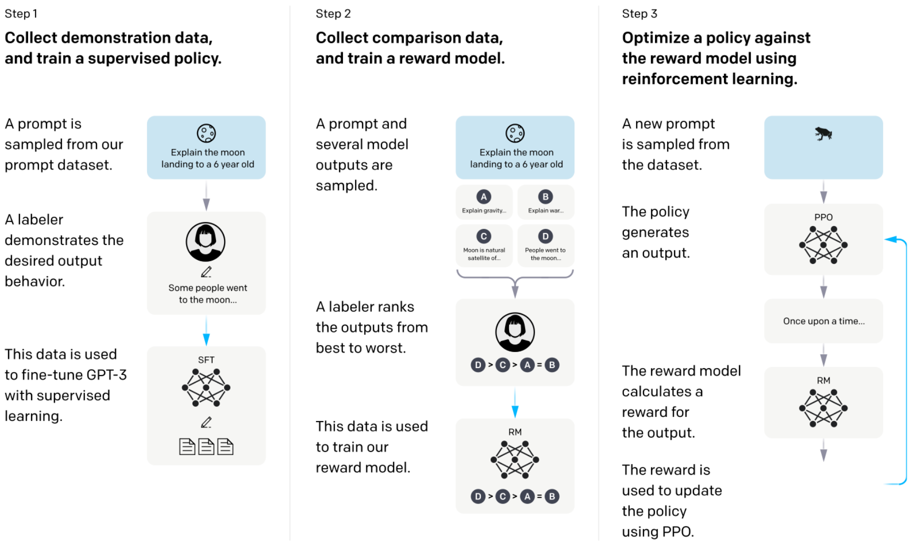


Figure 2 of "Training language models to follow instructions with human feedback", https://arxiv.org/abs/2203.02155

Visual Transformer

In Oct 2020, an influential paper

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

proposed processing of images using ViT a variant of the Transformers architecture (**Vi**sual **T**ransformer, a Pre-LN Transformer with GELU activations):

- trainable 1D positional embeddings are added to linear projection of fixed-size patches;
- the MLP is exactly FFN (expansion factor 4, GELU);
- when finetuning on larger images, positional embeddings are first linearly interpolated from the original to the new resolution.

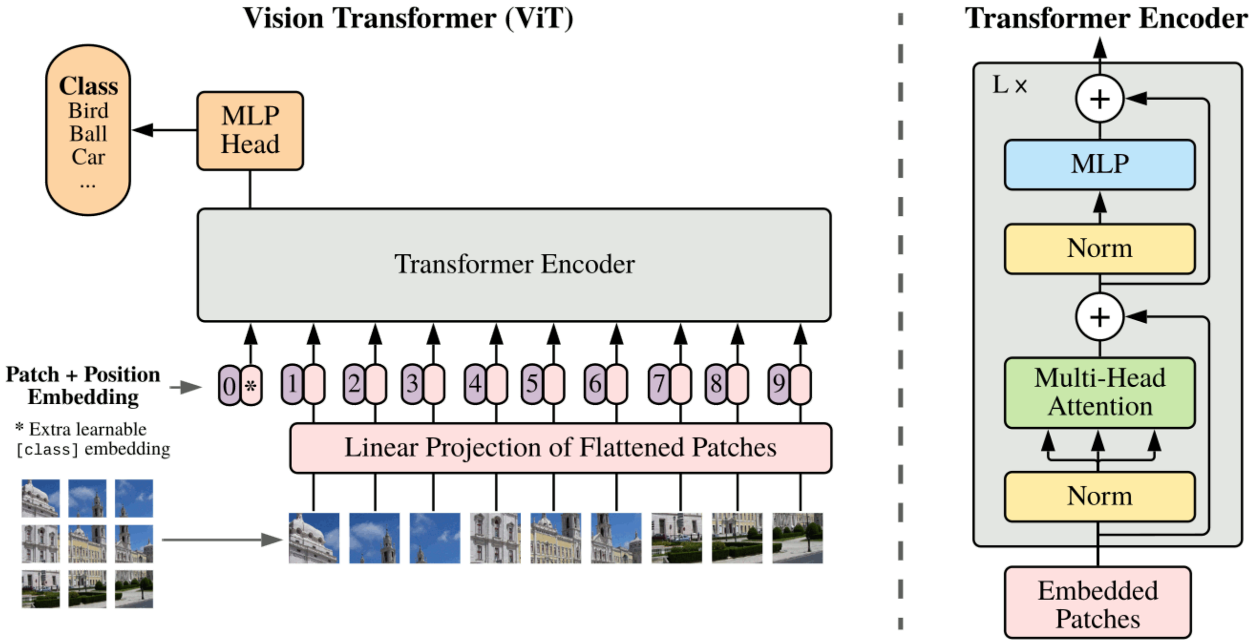


Figure 1 of "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", https://arxiv.org/abs/2010.11929

Image Recognition with Transformers

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1 of "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", https://arxiv.org/abs/2010.11929

The ViT architecture surpasses convolutional models like EfficientNet when pre-trained on very large data (~300M images); however, training only on ImageNet1k delivered worse results (77.9% top-1 accuracy).

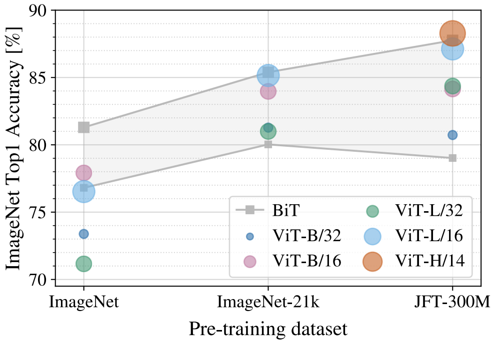


Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

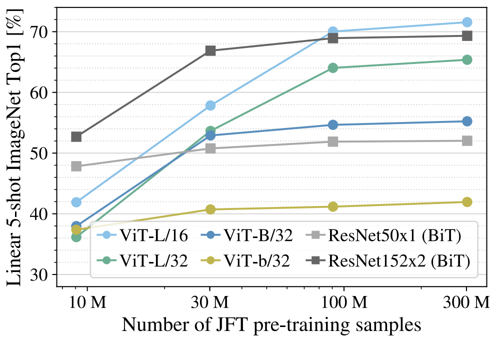


Figure 4: Linear few-shot evaluation on ImageNet versus pre-training size. ResNets perform better with smaller pre-training datasets but plateau sooner than ViT, which performs better with larger pre-training. ViT-b is ViT-B with all hidden dimensions halved.

Figures 3 and 4 of "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", https://arxiv.org/abs/2010.11929

Trainable 1D positional embeddings are used.

The authors consider also 2D positional embeddings, which are a concatenation of trainable 1D positional embeddings for each dimension, but the results are very comparable.

Pos. Emb.	Default/Stem	Every Layer	Every Layer-Shared
No Pos. Emb.	0.61382	N/A	N/A
1-D Pos. Emb.	0.64206	0.63964	0.64292
2-D Pos. Emb.	0.64001	0.64046	0.64022
Rel. Pos. Emb.	0.64032	N/A	N/A

Table 8: Results of the ablation study on positional embeddings with ViT-B/16 model evaluated on ImageNet 5-shot linear.

Table 8 of "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", https://arxiv.org/abs/2010.11929

DeiT

An improved training with a variety of augmentation (DeiT architecture, Dec 2020) resulted in performance close to EfficientNet when trained only on ImageNet1k data (83.1% vs 84.7% top-1 accuracy).

DeiT III: Revenge of the ViT (Apr 2022) has presented simplified training procedure, achieving results analogous to EfficientNetV2 on ImageNet1k (85.2% vs 85.7% top-1 accuracy) and ImageNet21k (87.2% vs 87.3% top-1 accuracy).

Image Recognition with Transformers

When data is limited (*"only" 1M images*), an efficient approach to train a ViT is a BERT-like masking, which was proposed in Nov 2021 paper

Masked Autoencoders Are Scalable Vision Learners.

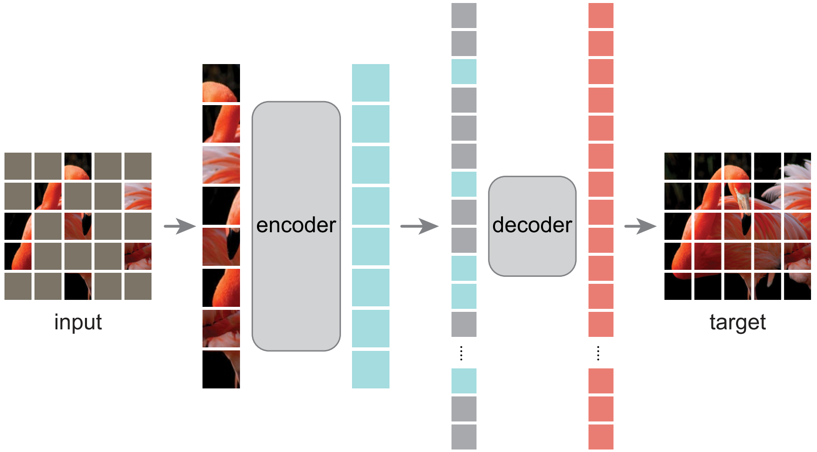


Figure 1 of "Masked Autoencoders Are Scalable Vision Learners", https://arxiv.org/abs/2111.06377

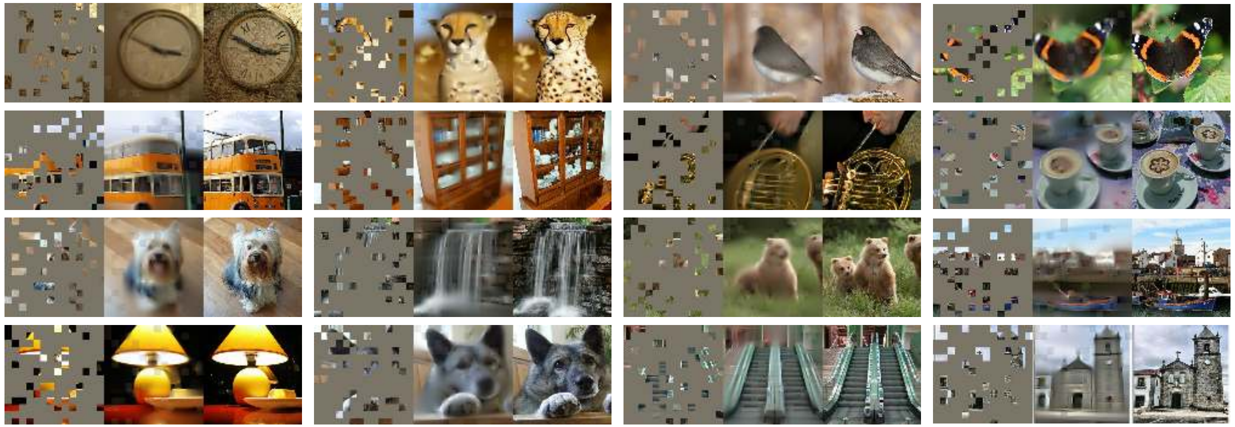


Figure 2 of "Masked Autoencoders Are Scalable Vision Learners", https://arxiv.org/abs/2111.06377

This MAE architecture reaches 86.9% top-1 accuracy on ImageNet1k-only training on images of size 224, and 87.8% on images of size 448.

As of April 2025, the current second best model on ImageNet is BASIC-L trained with Lion optimizer. The image encoder in BASIC-L is CoAtNet-7, an architecture combining MBConv in first stages and relative pre-activated 2D self-attention in later stages, with GELU activations everywhere. The image encoder has 2.4B parameters, it is trained on 6.6B noisy image-text pairs using a batch size of 65536 images in ~7k TPUv4/days, and achieves 91.1% top-1 accuracy.

It also achieves 88.3% zero-shot accuracy on ImageNet, i.e., when no ImageNet training data is used for training nor finetuning.

The current best model OmniVec pre-trains a model processing multiple modalities (images, depth maps, videos, 3D point clouds, audio, text) using domain-specific Transformer-based encoders and then processed by shared Transformer layers. Masked pretraining similar to MAE is employed (reaching 88.6% on ImageNet), and the model can be finetuned on individual datasets afterwards (92.4% top-1 ImageNet accuracy).

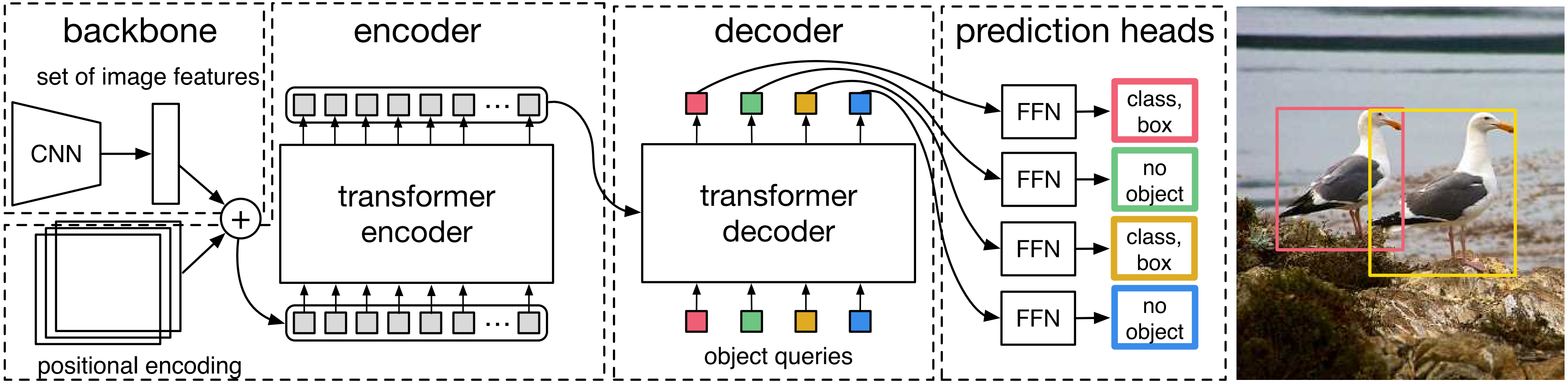


Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "`no object`" class.

Figure 2 of "End-to-End Object Detection with Transformers", https://arxiv.org/pdf/2005.12872.pdf

- During training, we pair the predictions and gold objects (padded with "`no object`"s to the same length) using a maximum-weight bipartite matching algorithm – the Hungarian algorithm. The matching is based on both the classification and regression losses.
- The encoder uses fixed sine positional encodings added to every self-attention layer. The x and y axes are encoded independently and concatenated.

spatial pos. enc. encoder	spatial pos. enc. decoder	output pos. enc. decoder	AP	Δ	AP_{50}	Δ
none	none	learned at input	32.8	-7.8	55.2	-6.5
sine at input	sine at input	learned at input	39.2	-1.4	60.0	-1.6
learned at attn.	learned at attn.	learned at attn.	39.6	-1.0	60.7	-0.9
none	sine at attn.	learned at attn.	39.3	-1.3	60.3	-1.4
sine at attn.	sine at attn.	learned at attn.	**40.6**	-	**61.6**	-

Table 3 of "End-to-End Object Detection with Transformers", https://arxiv.org/pdf/2005.12872.pdf

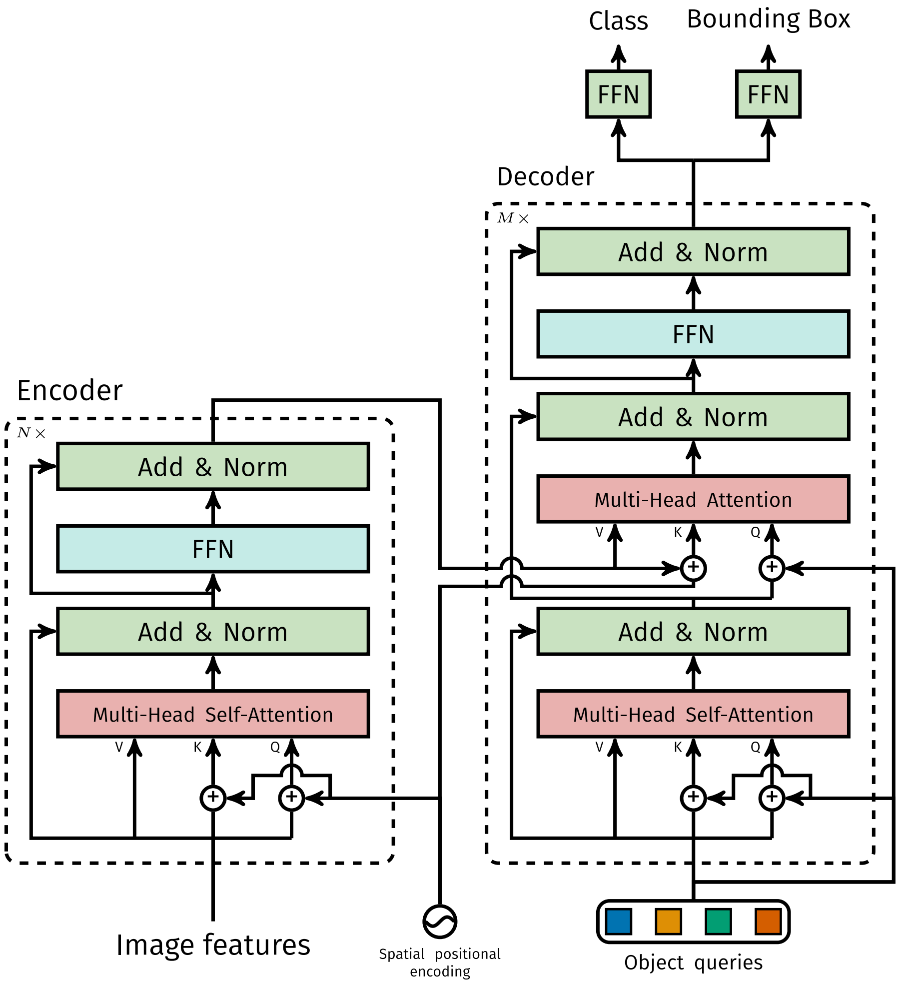


Figure 10 of "End-to-End Object Detection with Transformers", https://arxiv.org/pdf/2005.12872.pdf