Transformers

Bryan Pardo

Deep Learning

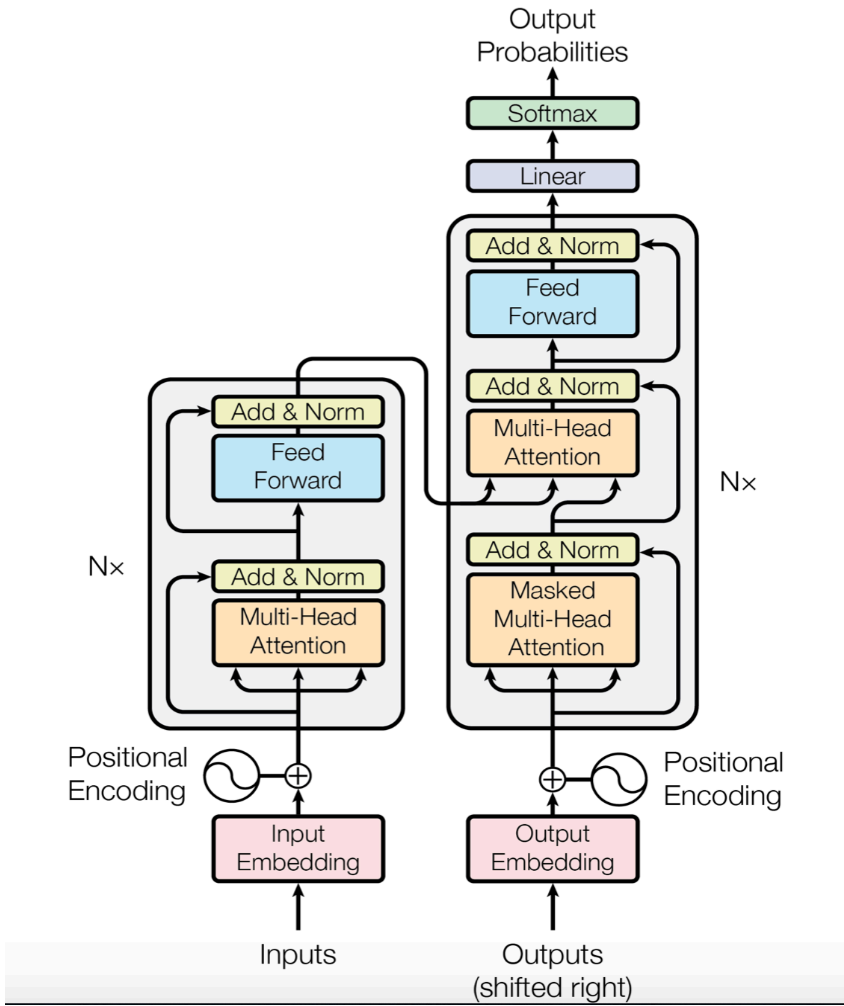
Northwestern University

We’re ready for Transformers!

Pay Attention!

- Transformers introduced in 2017
- Use attention
- Do NOT use recurrent layers
- Do NOT use convolutional layers
- ..Hence the title of the paper that introduced them

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). **Attention is all you need**. In *Advances in neural information processing systems*(pp. 5998-6008).



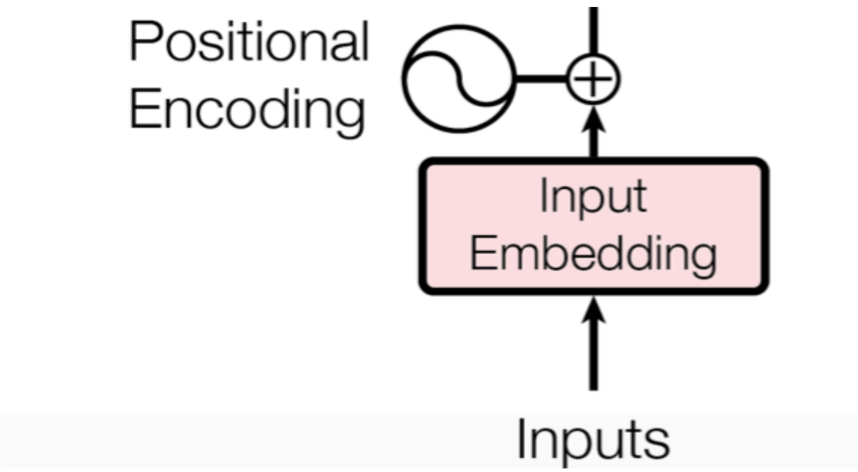
Features of Transformers

- They use residual connections to allow deeper networks to fine-tune as appropriate

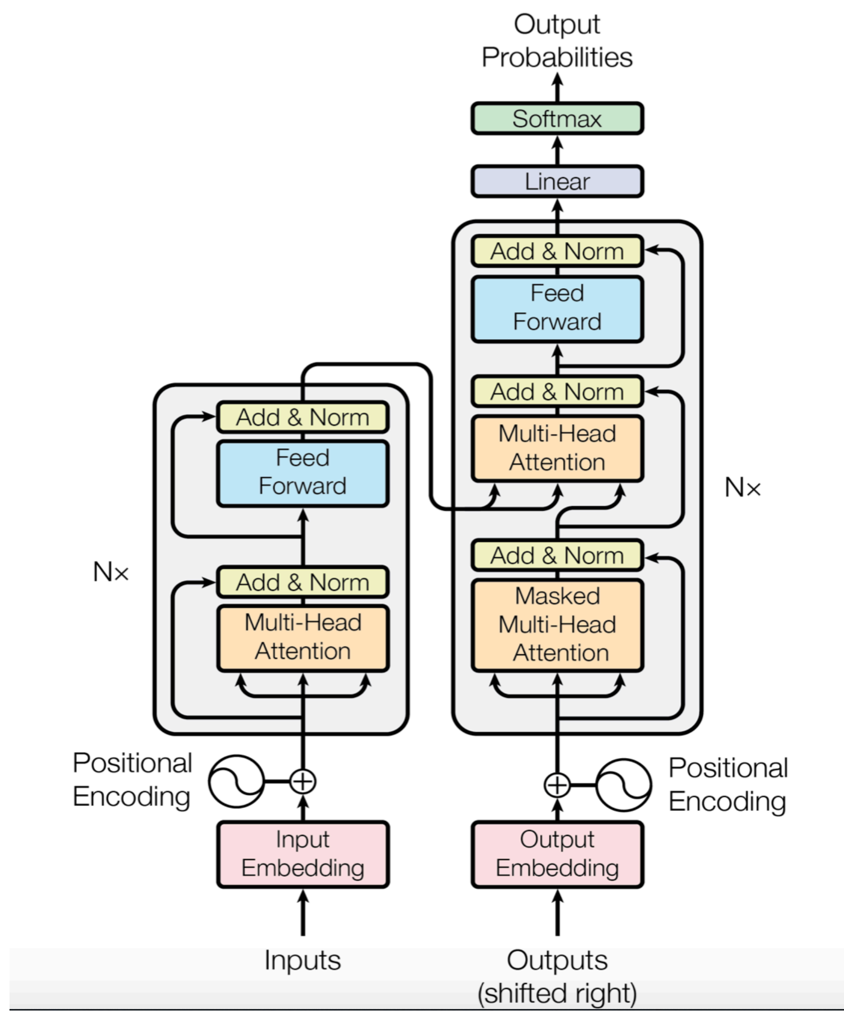
- They use attention in multiple places in both the encoder and decoder

- People often mistake them for automobiles

Step-by-step



Positional
Encoding
Input
Embedding
Inputs



Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)

The Input sequence

- Let's build a representation that takes the input sequence and learns relationships between input tokens (words represented by their embeddings)

Embedding:	0	734	912	733	43	1

Input sentence: [START] Thinking machines think quickly [STOP]

Positional encoding

- In an RNN, the recurrence encodes the order implicitly.
- In a Transformer, relatedness between words is handled by self-attention.
- If we’re not using recursion to implicitly encode order, how does the system tell the difference between these two sentences?

  Bob hands Maria the ball.

  Maria hands Bob the ball.

Positional encoding

- Lots of ways to go
  - e.g. just number the items.
- They chose to implicitly encode position by adding the values of a bunch of sinewaves to the embeddings
- Honestly, I'm not sure why they did it this way, rather than just appending an index number to the embedding

Positional encoding

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

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

$i =$ the index of a dimension in a single input embedding vector x
$d_{model} =$ the total number of dimensions in the embedding
pos = the position in the sequence of the input embedding vector

A concrete example

For example, for word w at position $pos \in [0, L-1]$ in the input sequence $w = (w_0, \cdots, w_{L-1})$, with 4-dimensional embedding e_w, and $d_{model} = 4$, the operation would be

$$\begin{aligned} e'_w &= e_w + \left[sin\left(\frac{pos}{10000^0}\right), cos\left(\frac{pos}{10000^0}\right), sin\left(\frac{pos}{10000^{2/4}}\right), cos\left(\frac{pos}{10000^{2/4}}\right) \right] \\ &= e_w + \left[sin\left(pos\right), cos\left(pos\right), sin\left(\frac{pos}{100}\right), cos\left(\frac{pos}{100}\right) \right] \end{aligned}$$

where the formula for positional encoding is as follows

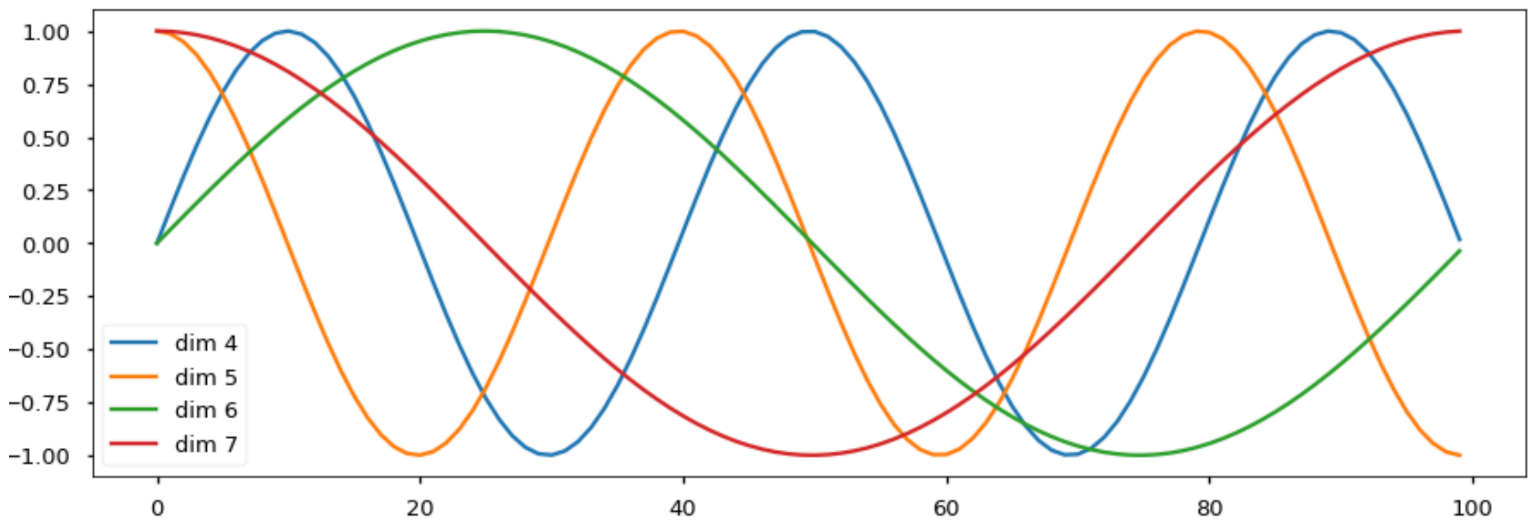
$$\mathrm{PE}(pos, 2i) = sin\left(\frac{pos}{10000^{2i/d_{model}}}\right),$$

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

with $d_{model} = 512$ (thus $i \in [0, 255]$) in the original paper.

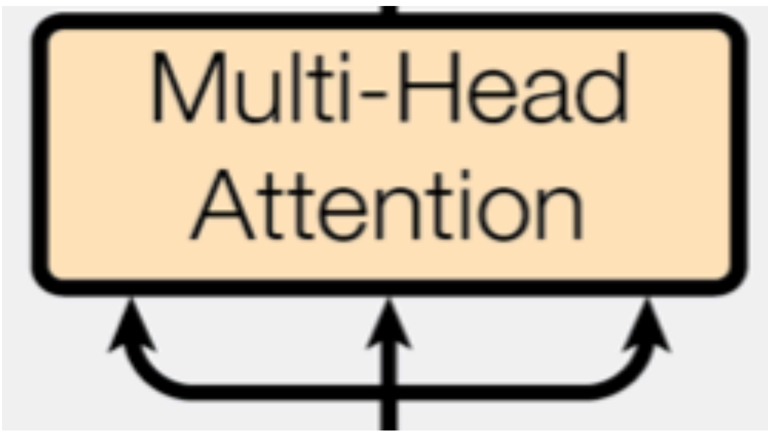
Thanks for the example, anonymous contributor on Stack Overflow!
https://datascience.stackexchange.com/questions/51065/what-is-the-positional-encoding-in-the-transformer-model

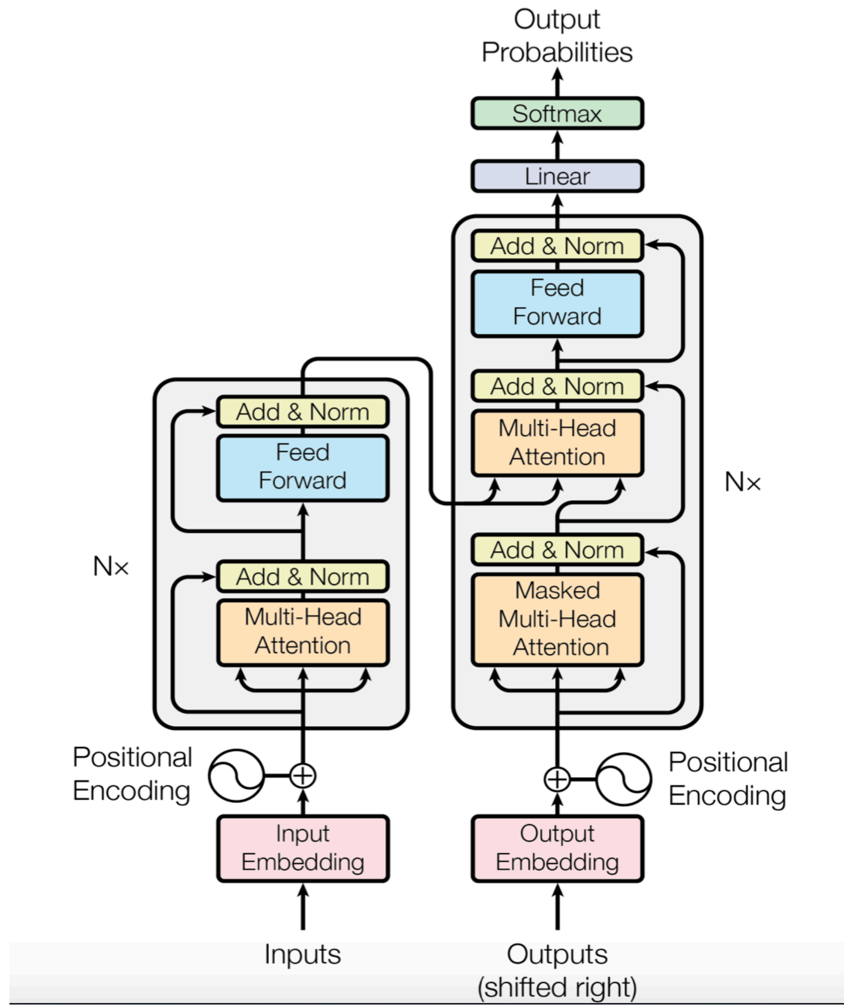
Some positional encodings visualized



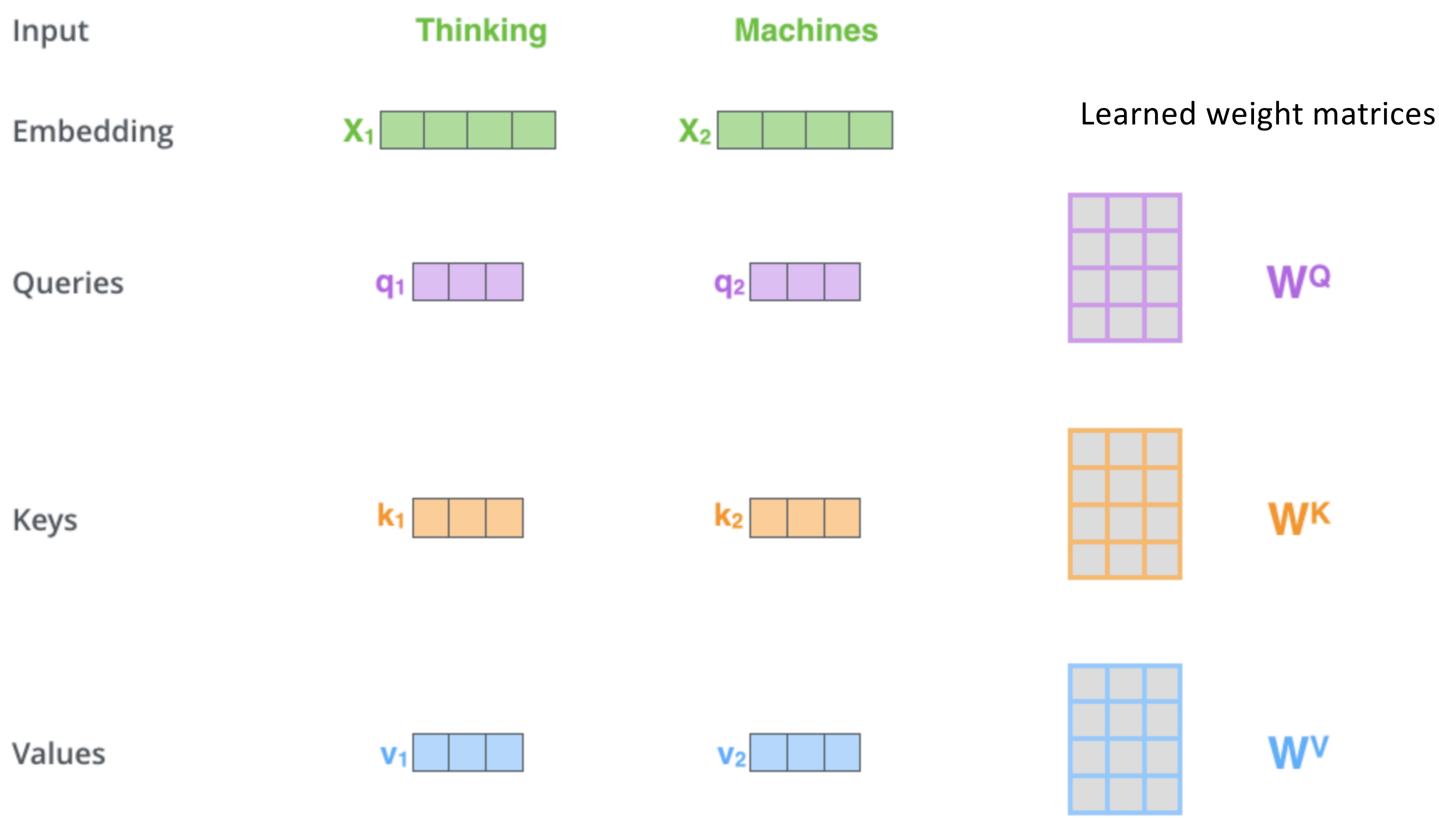
https://nlp.seas.harvard.edu/2018/04/03/attention.html#decoder

Step-by-step


Multi-Head
Attention

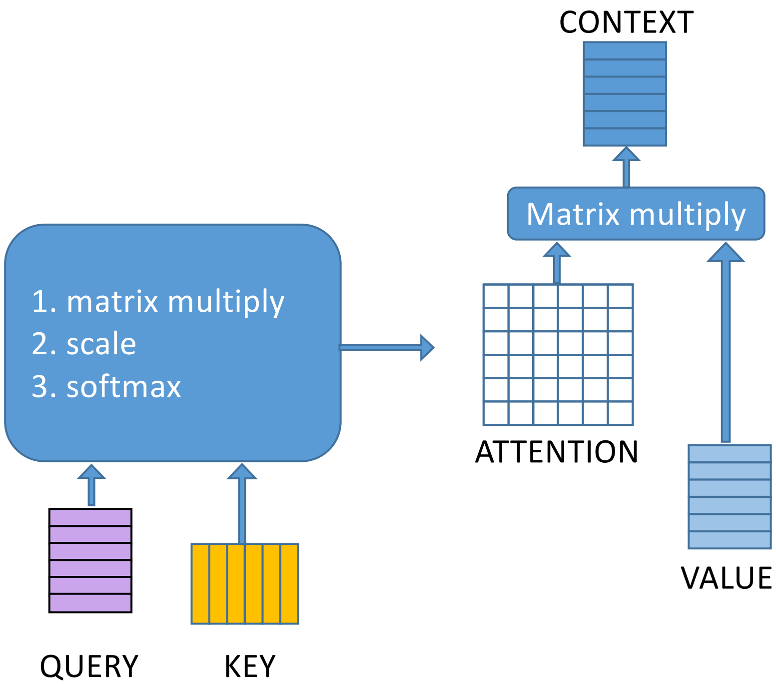

Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)

Self Attention: Query – Key – Value



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

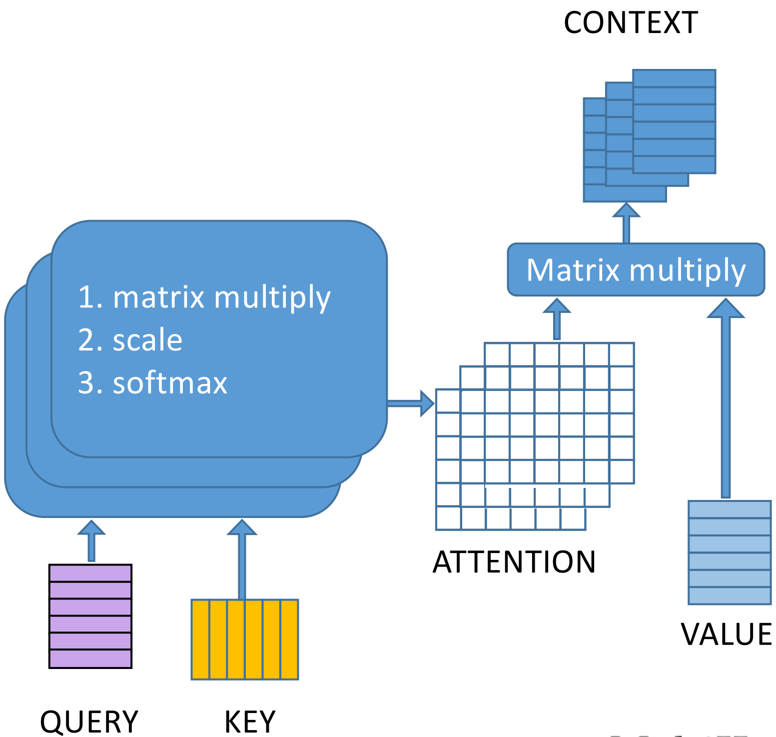
Self Attention: Query – Key – Value



- QUERY: Built from embedded input sequence
- KEY: Same as query
- VALUE: Same as query
- ATTENTION: "relatedness" between pairs of words
- CONTEXT: The sequence of context values

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

Multi-head Self Attention



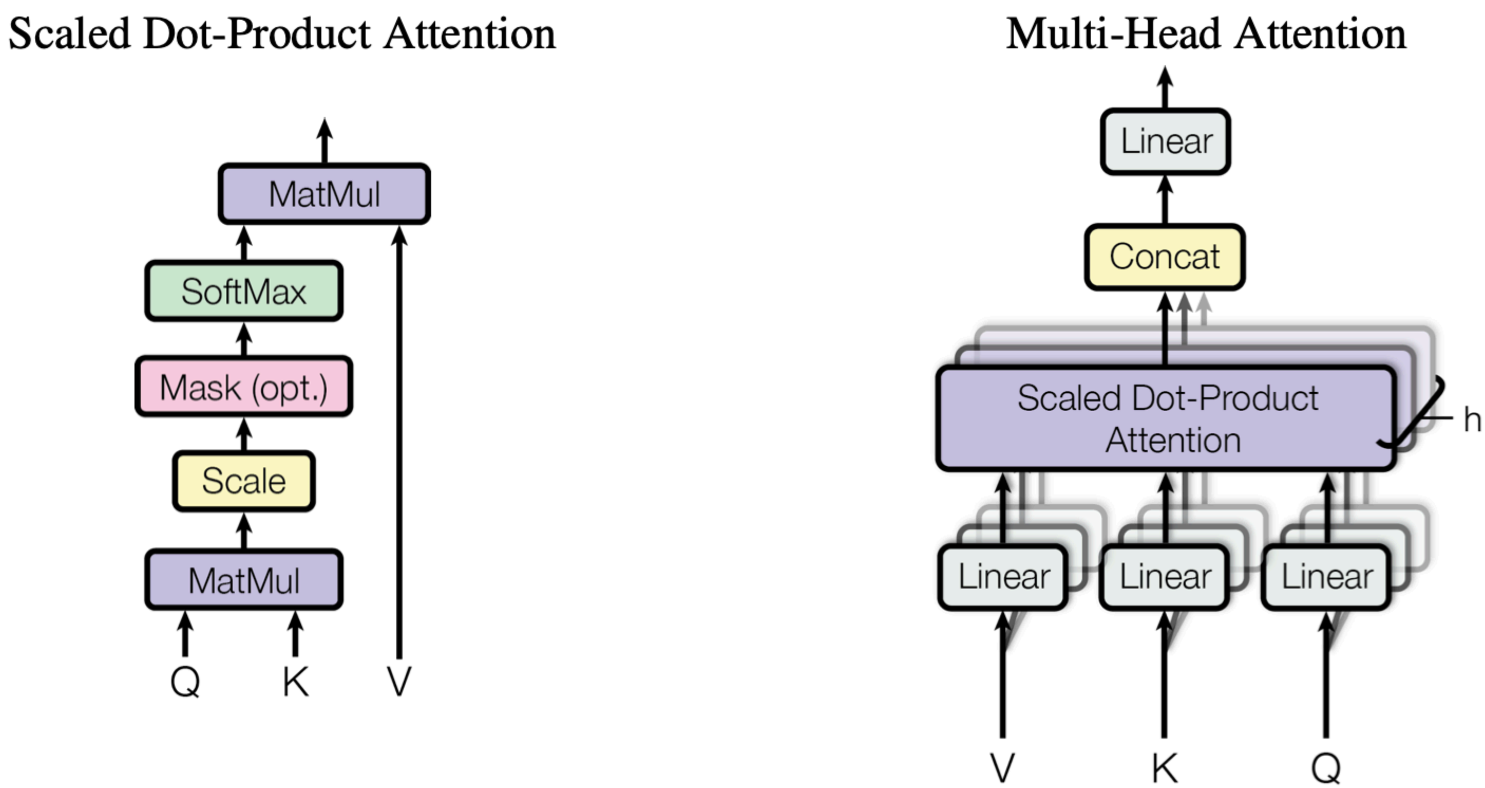
- Each attention head will have unique learned weight matrices for the query (Q), key (K), and value (V), where *i* is the index of each attention head

$$W_i^Q, W_i^K, W_i^V$$

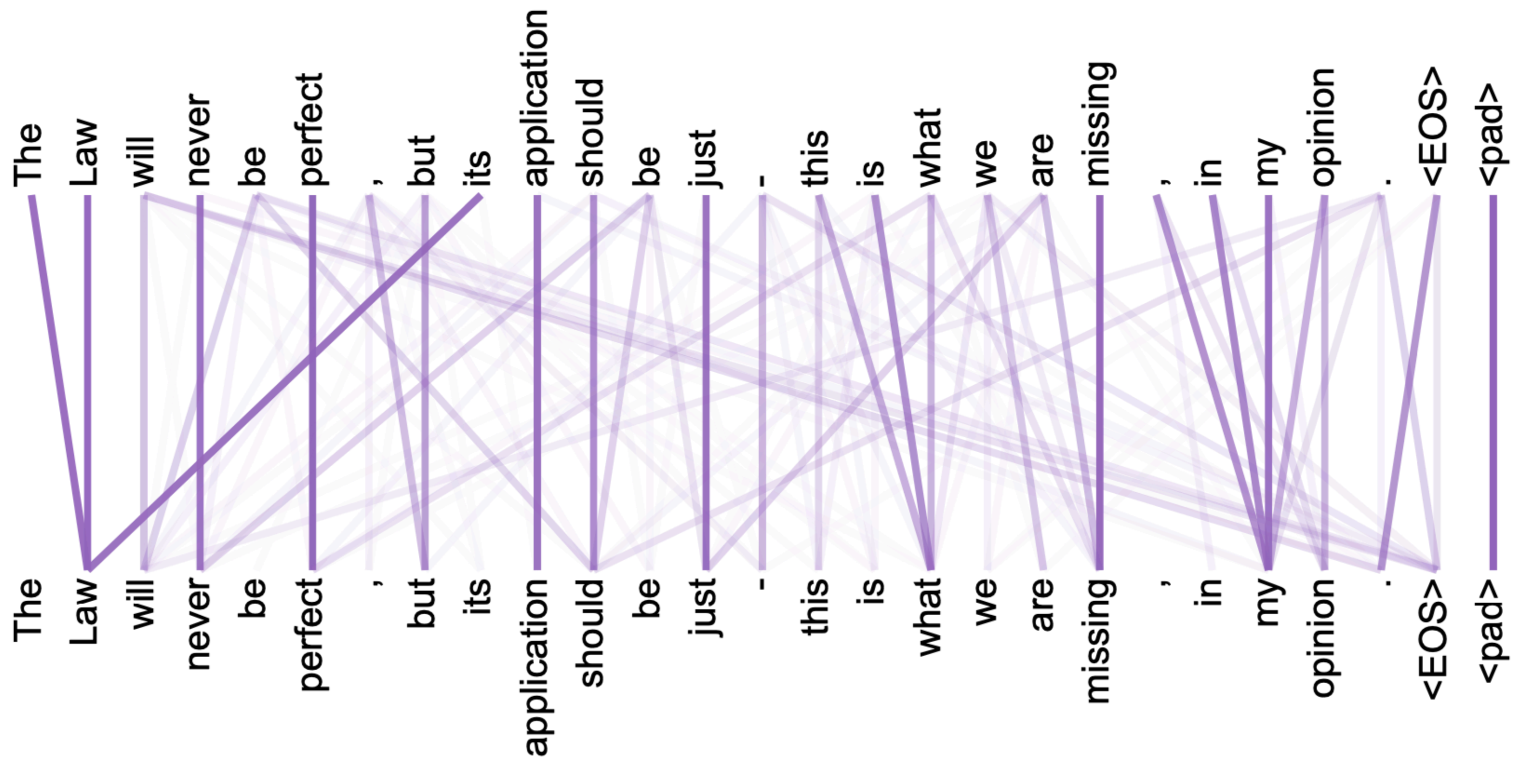
- There's also output weight to learn for the multi-head layer

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Here’s the figure from the paper



Self-attention: one head, all words



Self attention: Multiple heads, one word

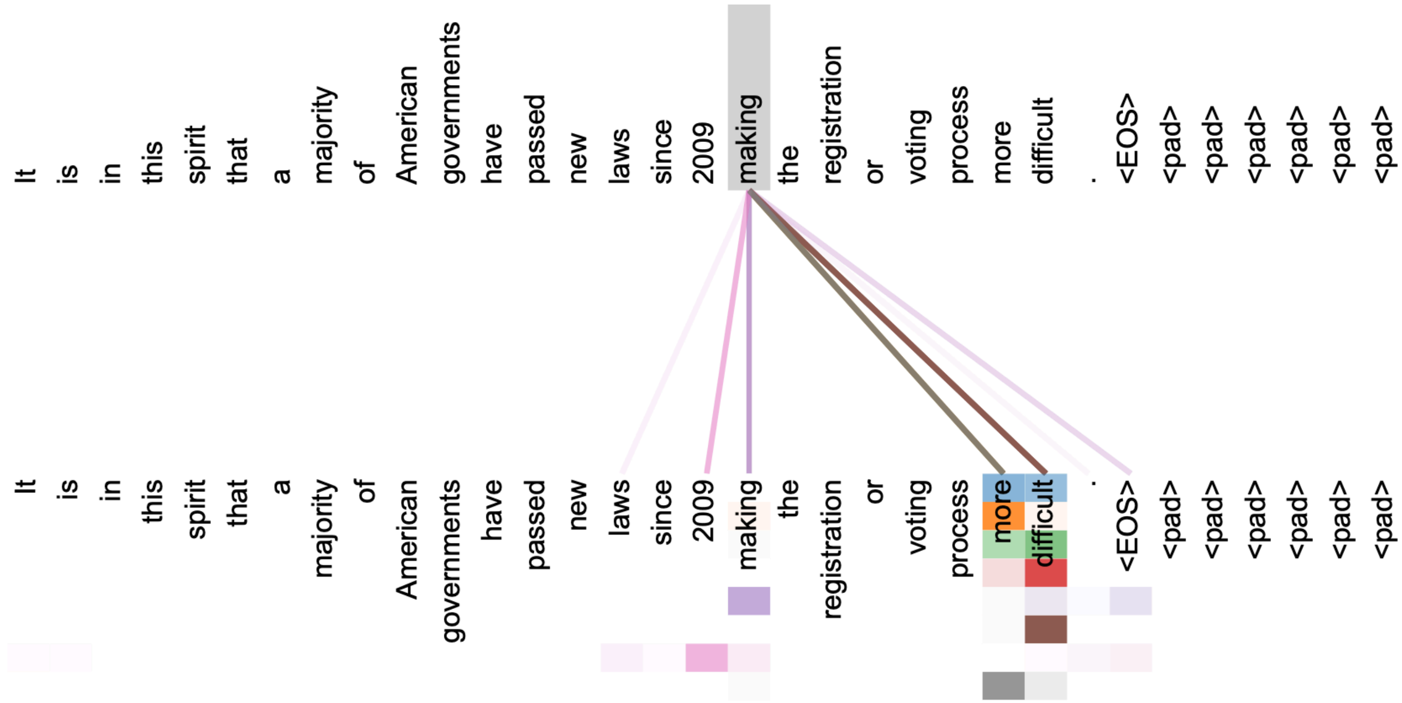
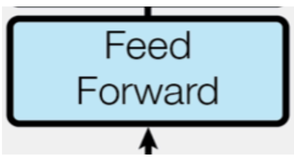


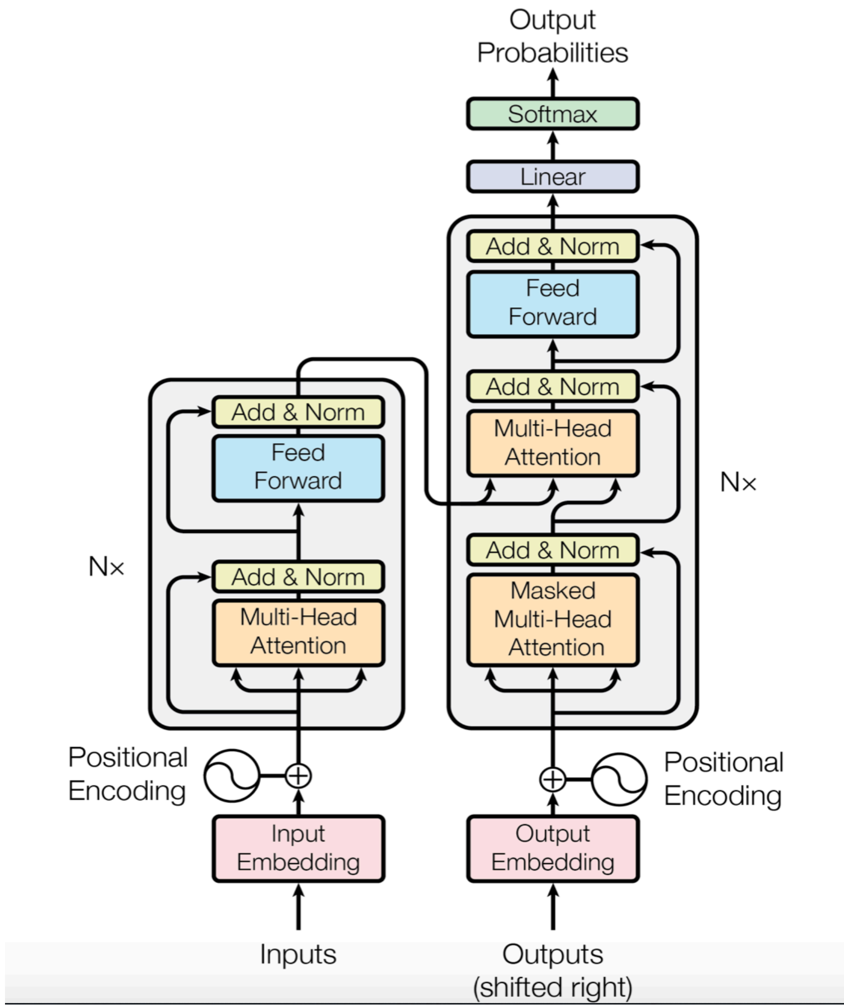
Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

Step-by-step



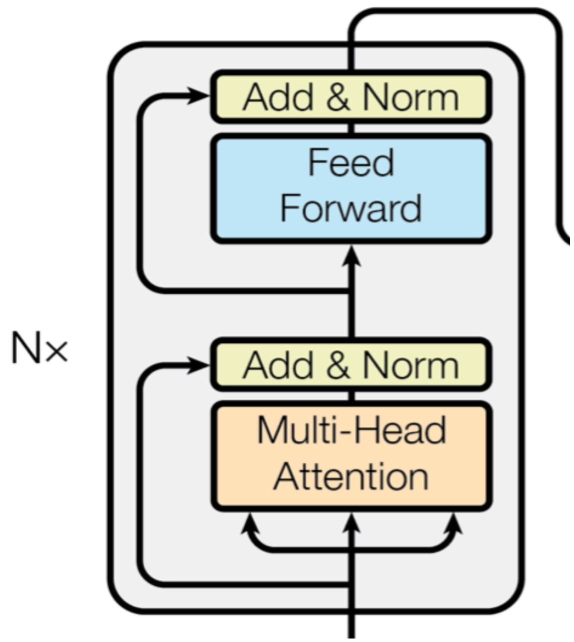
$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Two linear layers, with a ReLU in between

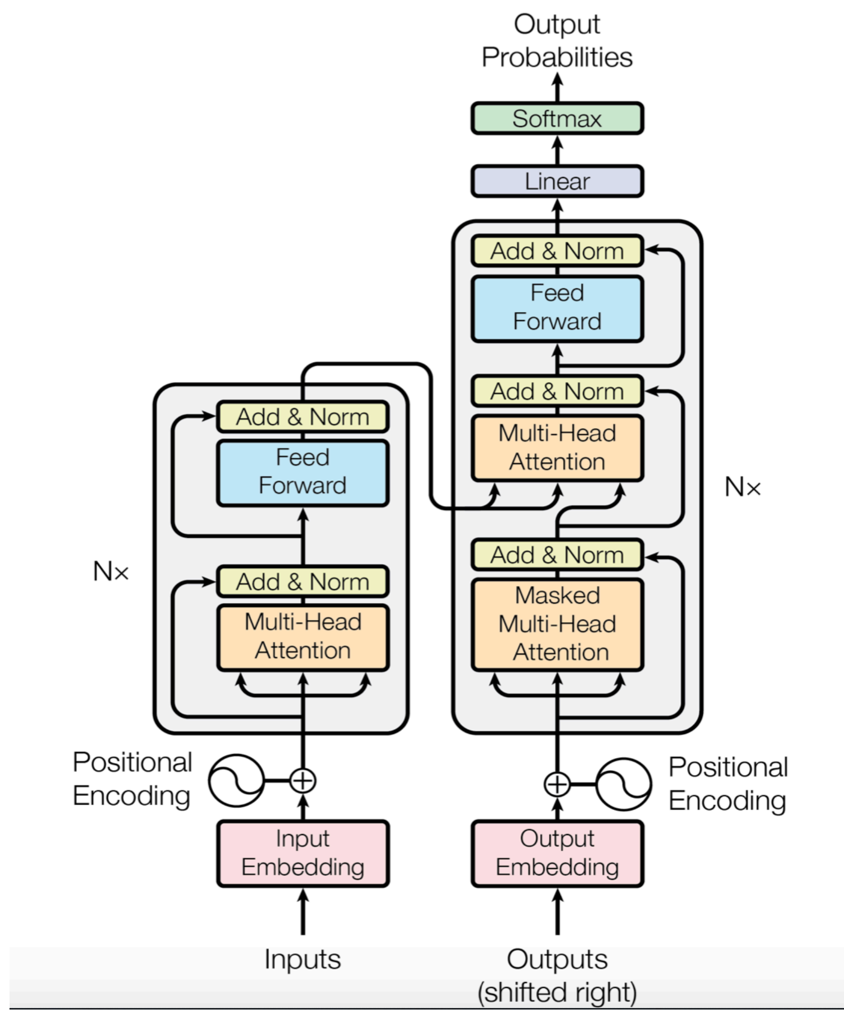


Step-by-step

One encoder block

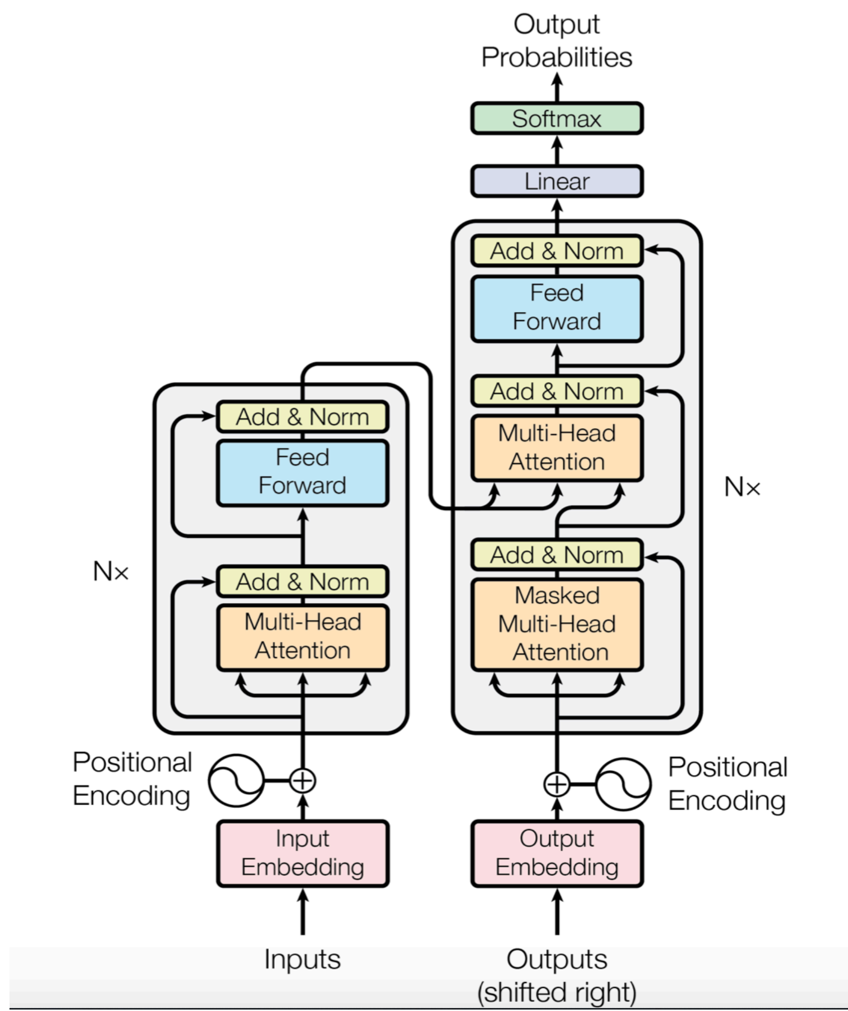


Nx: The number of encoder blocks IN SEQUENCE

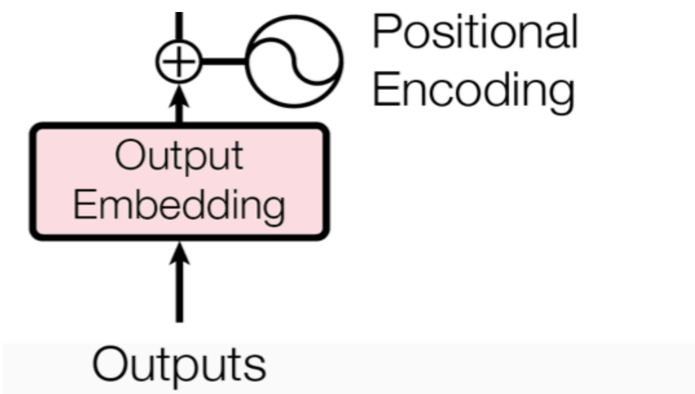


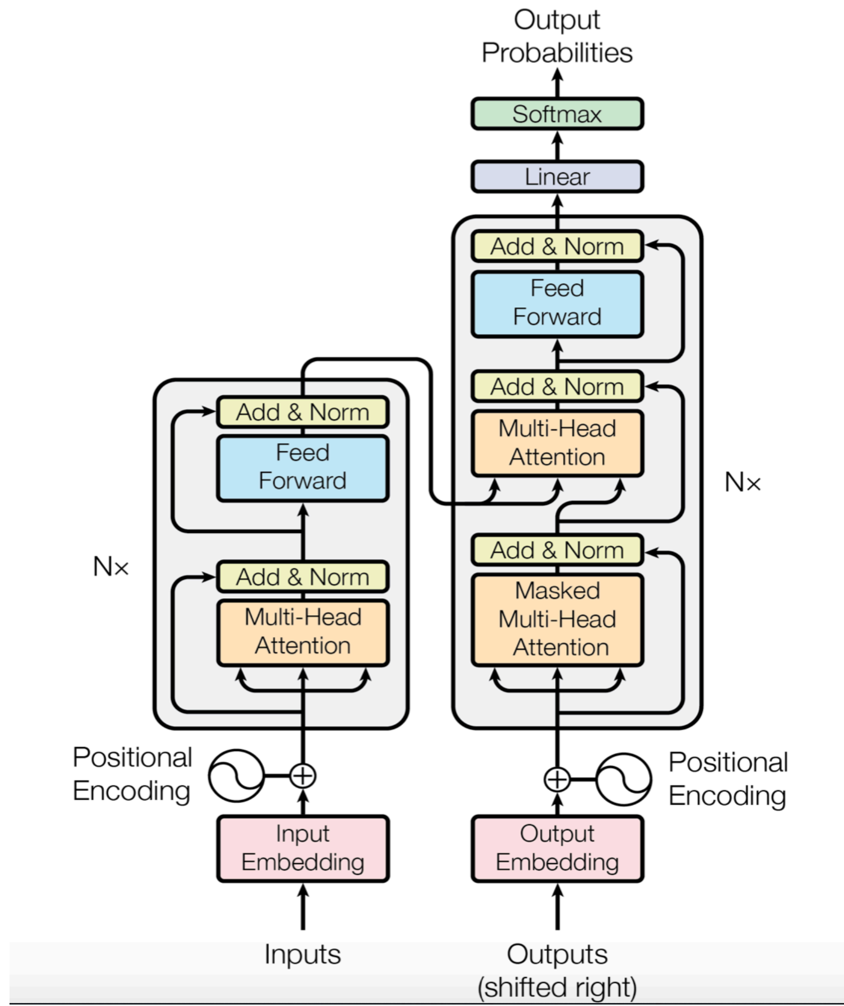
Step-by-step

- Run the encoder on the ENTIRE input language sequence.
- The decoder outputs tokens one-at-a-time, feeding the previous token output into the model to generate the next token.
- The next decoder output is conditioned on the ENTIRE sequence of encoder outputs + the previous decoder output.

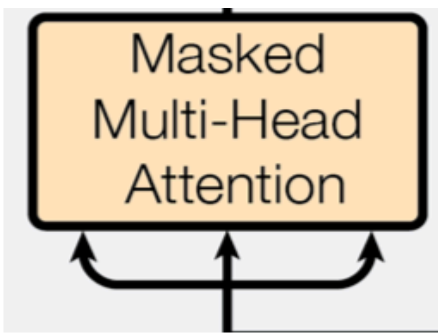


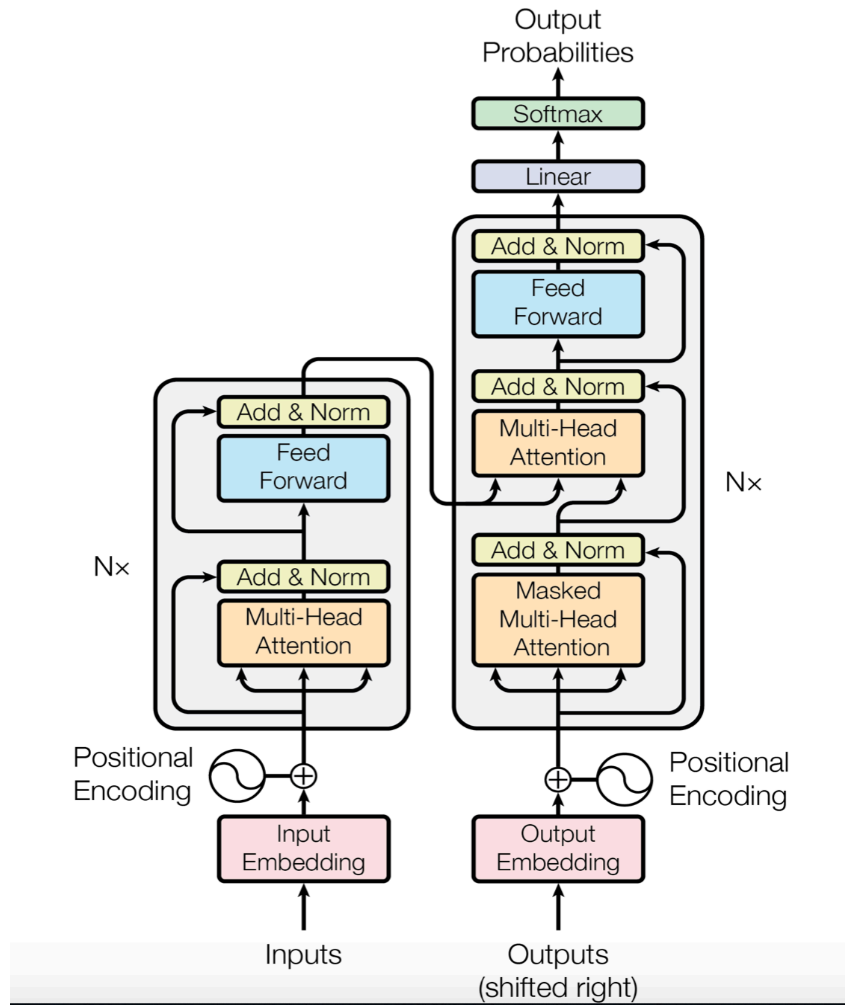
Step-by-step


Positional
Encoding
Output
Embedding
Outputs


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)

Step-by-step


Masked
Multi-Head
Attention

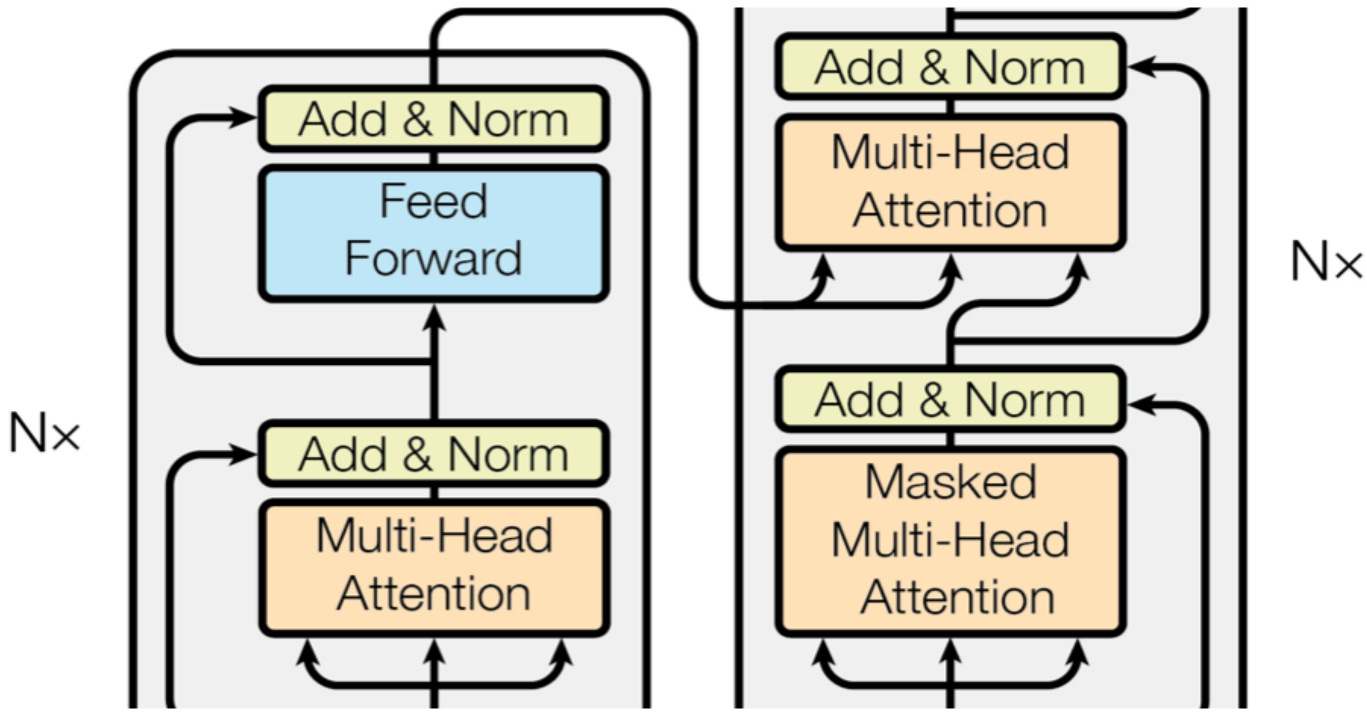

Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)

Masked attention

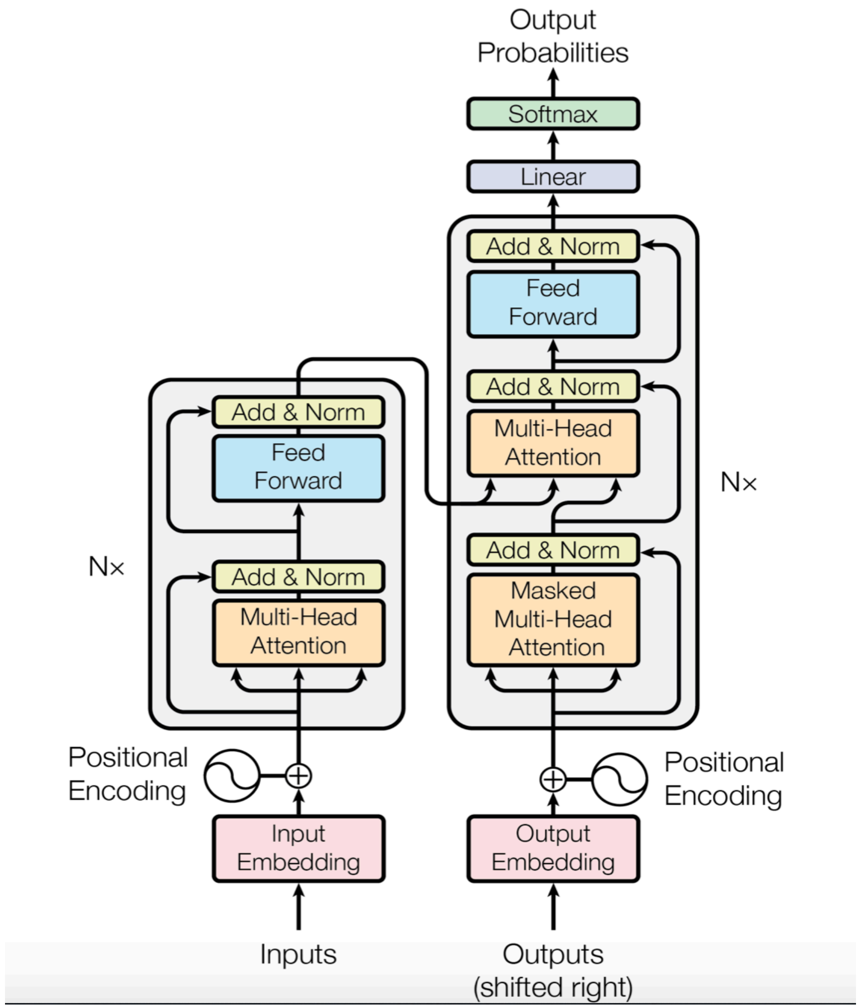
- Don’t let the attention ”look ahead” to sequence elements the system hasn’t generated yet
- Apply a “mask” matrix with 0 everywhere you’re not allowed to look and 1 everywhere else
- Do element-wise multiplication to the value vector

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

3 variants of attention



That’s the whole model



Transformer encoding: parallelizable

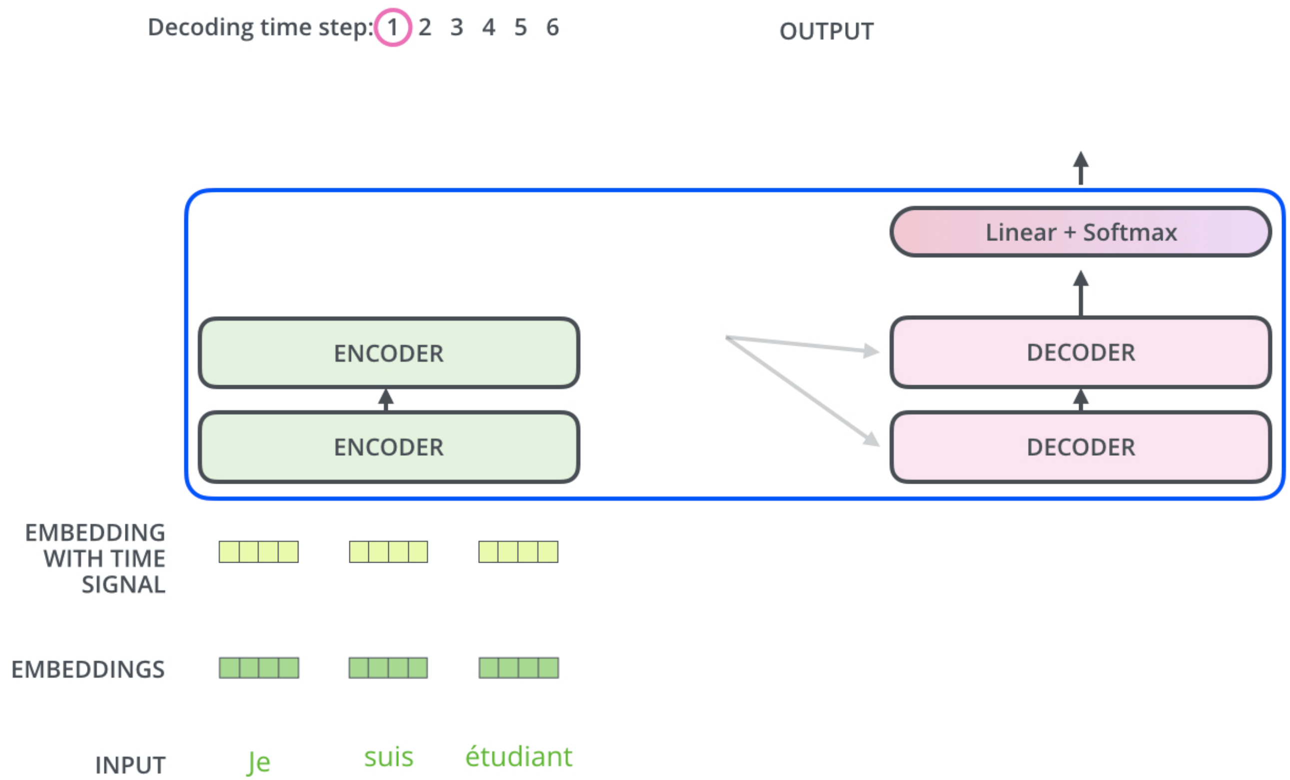


Image from https://jalammar.github.io/illustrated-transformer/

Transformer decoding: autoregressive

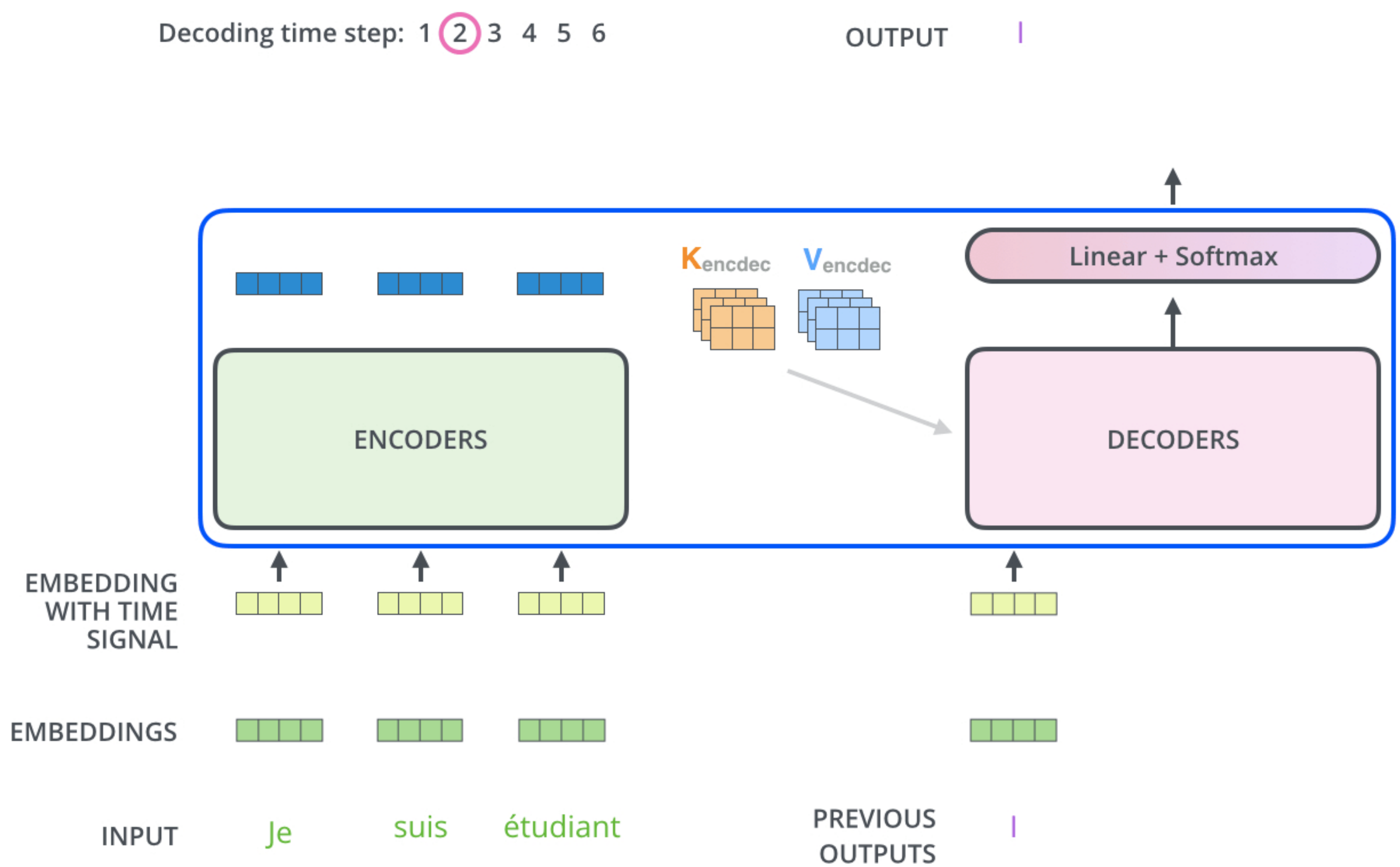


Image from https://.github.io/illustrated-transformer/

State-of-the-art language translation

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

OK I'm convinced….attention is great but…

- What if I'm doing (single) language modeling instead of translation?
- Do I really need this whole encoder-decoder framework?
- No. You don't (more on this in a minute)

Training a good language model is…

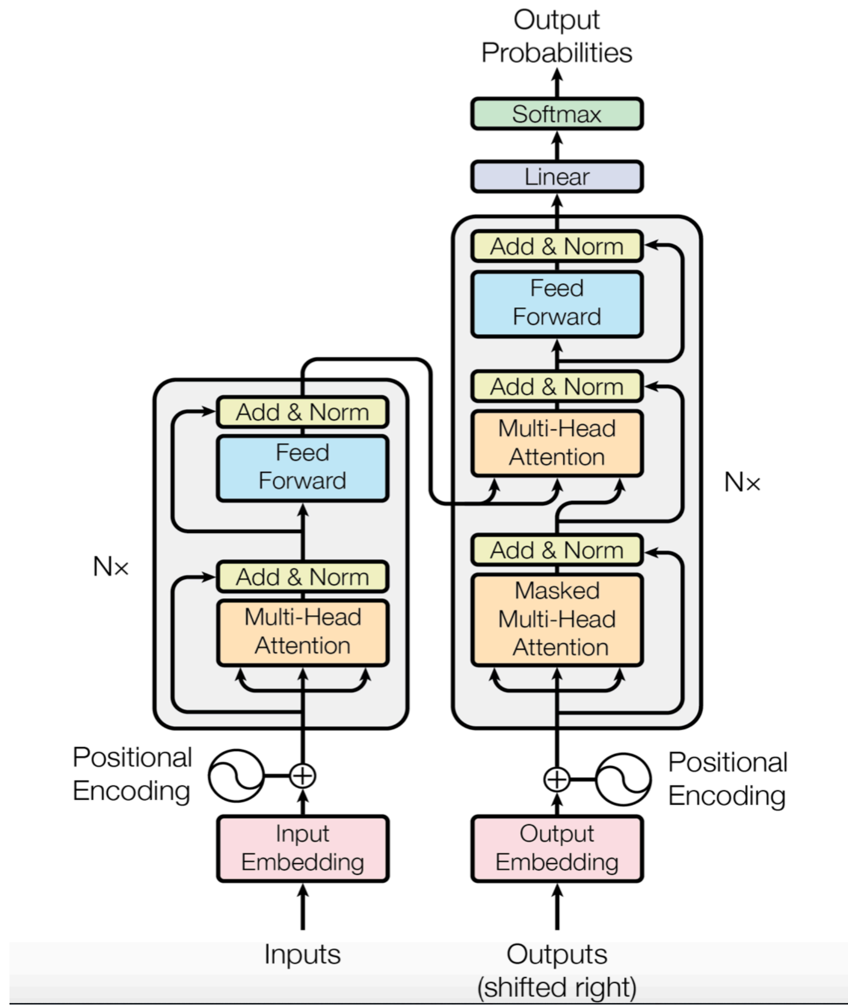
- Slow
- Requires lots of data
- Not every task has enough labeled data
- Requires lots of computing resources
- Not everyone has enough computing resources

Let’s use TRANSFER LEARNING!

- Train a general model on some task using a huge dataset for a long time
- Fine-tune the trained model on a variety of related “downstream” tasks
- Allows re-use of architecture
- Allows training on tasks where there is relatively little data

Use half of a Transformer

- Get rid of the encoder
- Use the decoder as a language model
- Note that this is an autoregressive model



GPT: Generative Pre-Training

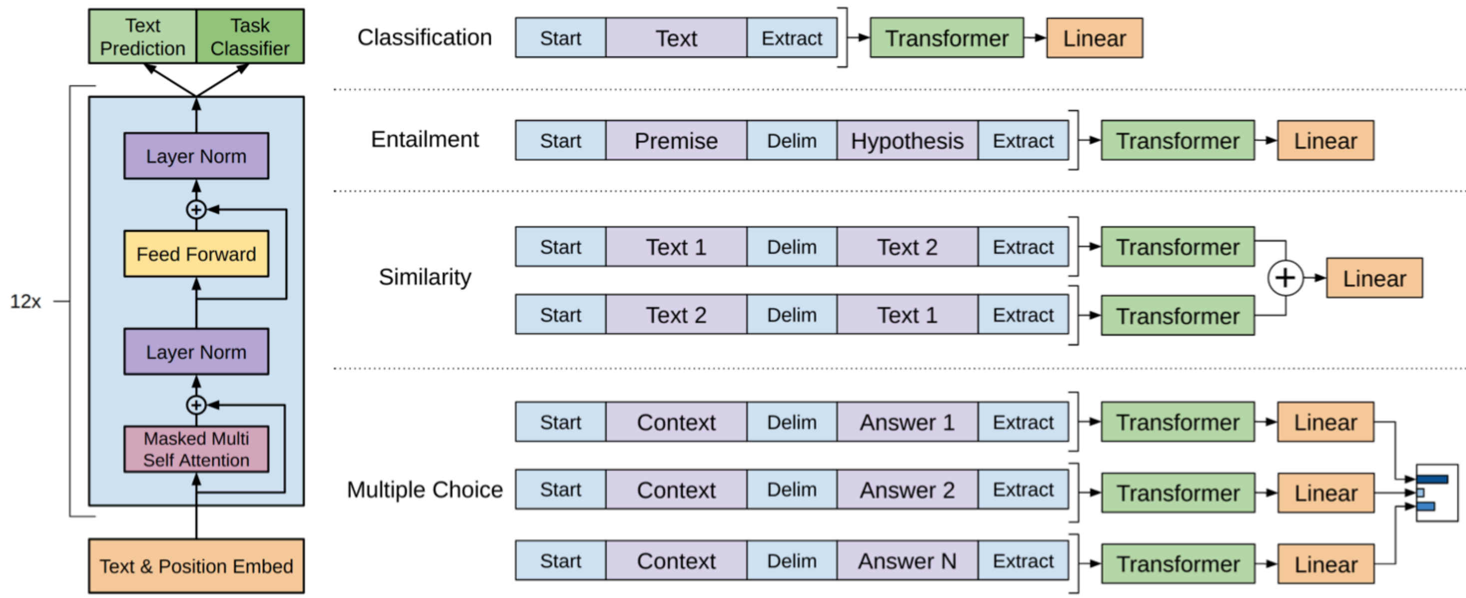


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

GPT was state of the art in 2018

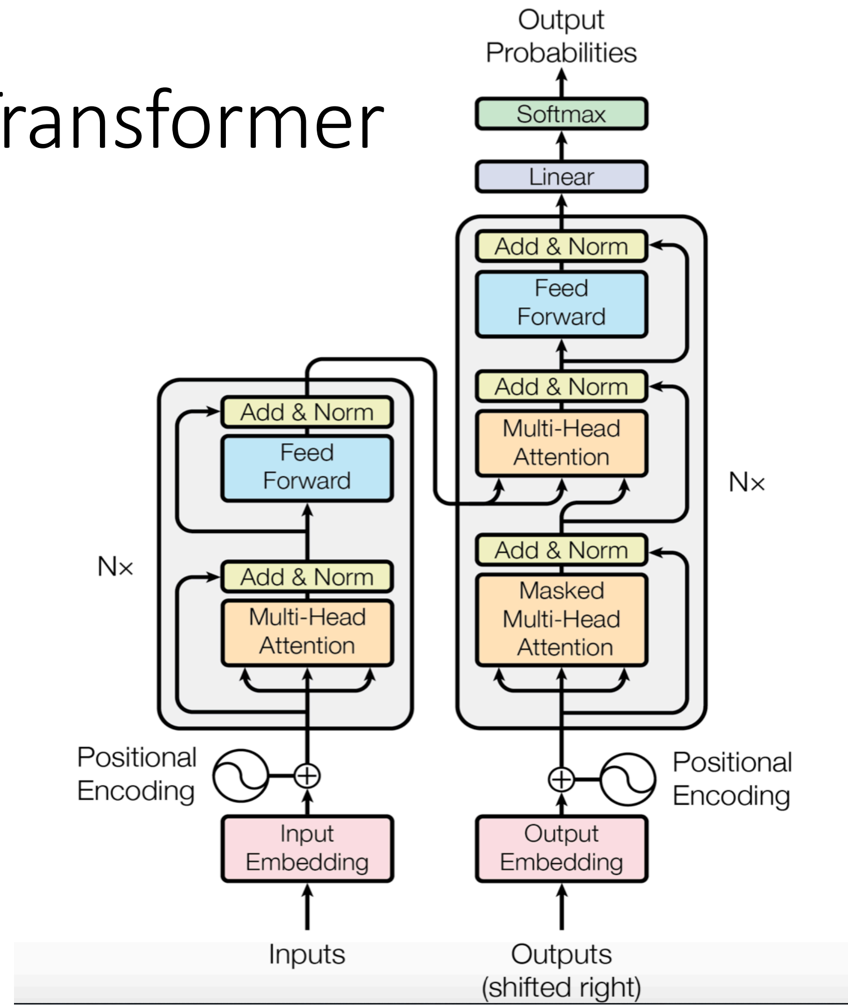
DATASET	TASK	SOTA	OURS
SNLI	Textual Entailment	89.3	**89.9**
MNLI Matched	Textual Entailment	80.6	**82.1**
MNLI Mismatched	Textual Entailment	80.1	**81.4**
SciTail	Textual Entailment	83.3	**88.3**
QNLI	Textual Entailment	82.3	**88.1**
RTE	Textual Entailment	**61.7**	56.0
STS-B	Semantic Similarity	81.0	**82.0**
QQP	Semantic Similarity	66.1	**70.3**
MRPC	Semantic Similarity	**86.0**	82.3
RACE	Reading Comprehension	53.3	**59.0**
ROCStories	Commonsense Reasoning	77.6	**86.5**
COPA	Commonsense Reasoning	71.2	**78.6**
SST-2	Sentiment Analysis	**93.2**	91.3
CoLA	Linguistic Acceptability	35.0	**45.4**
GLUE	Multi Task Benchmark	68.9	**72.8**

BERT
Bidirectional **E**ncoder **R**epresentations from **T**ransformers

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Use the OTHER half of a Transformer

- Get rid of the decoder
- Use the encoder as a language model



Input encoding to BERT

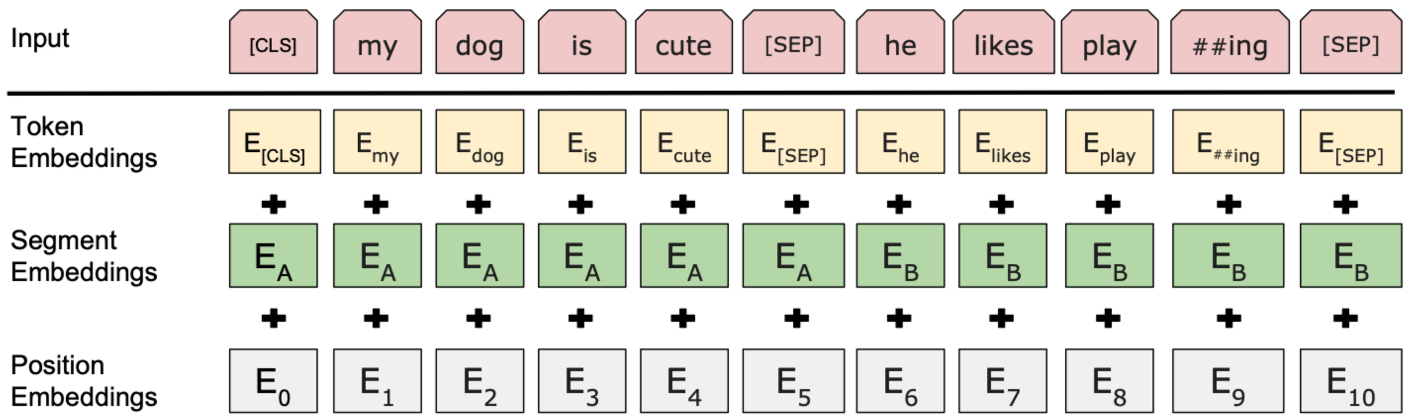


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, tl
tion embeddings and the position embeddings.

Masked word prediction training

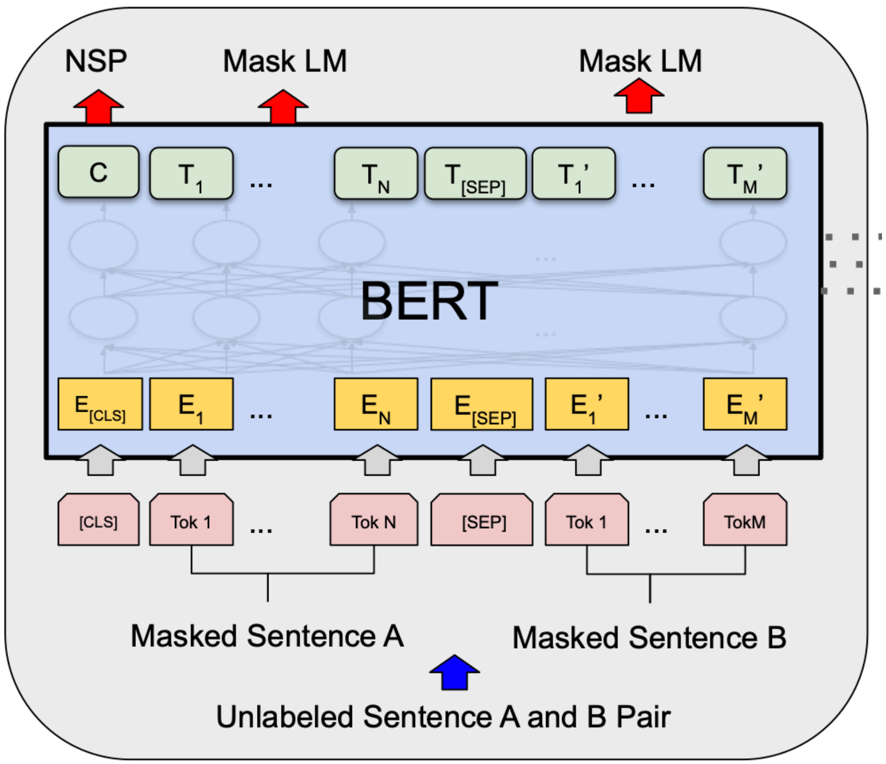
- Randomly cover up a word in a sentence

Original sentence: My dog has fleas.

Training sentences:

[MASK] dog has fleas.
My [MASK] has fleas.
My dog [MASK] fleas.
My dog has [MASK].

Model learns to expect [MASK]. Here's the fix.



- 80% of the time: Replace the word with the `[MASK]` token, e.g., `my dog is hairy` → `my dog is [MASK]`
- 10% of the time: Replace the word with a random word, e.g., `my dog is hairy` → `my dog is apple`
- 10% of the time: Keep the word unchanged, e.g., `my dog is hairy` → `my dog is hairy`. The purpose of this is to bias the representation towards the actual observed word.

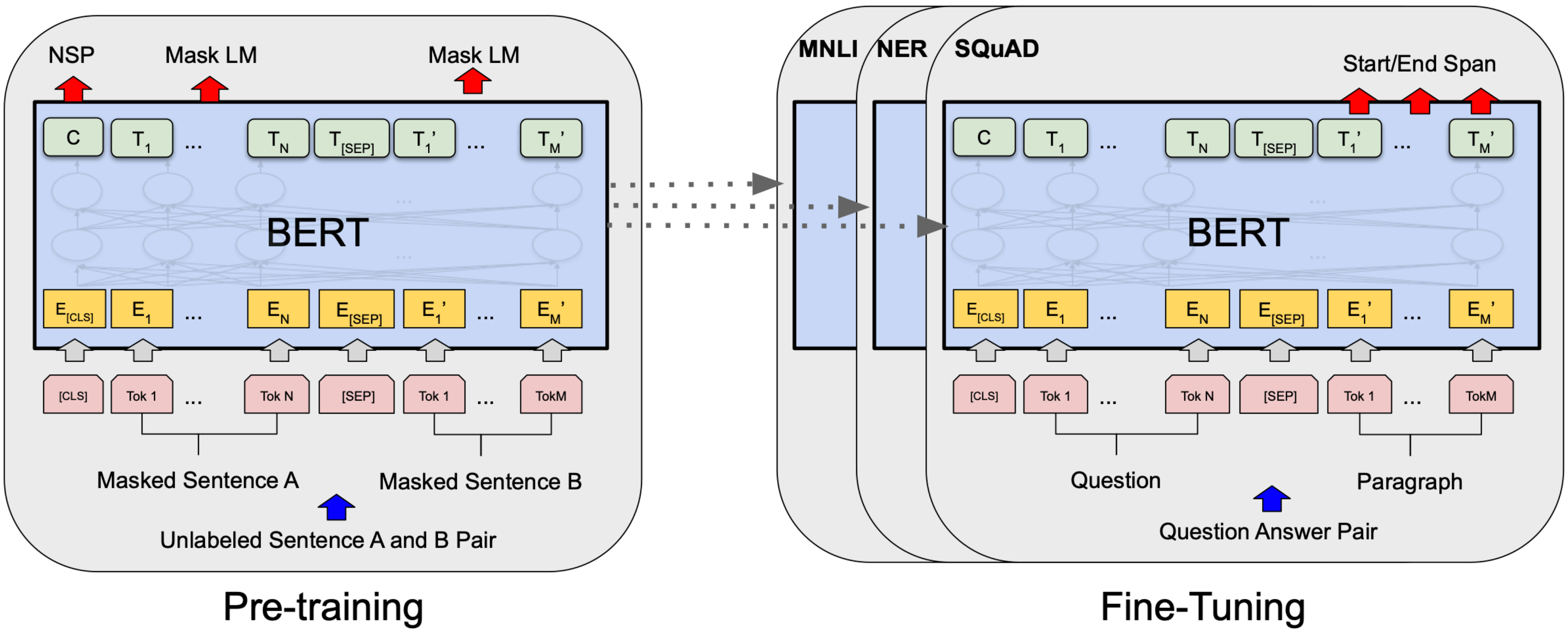


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

Language Tasks

MNLI Multi-Genre Natural Language Inference is a large-scale, crowdsourced entailment classification task (Williams et al., 2018). Given a pair of sentences, the goal is to predict whether the second sentence is an *entailment*, *contradiction*, or *neutral* with respect to the first one.

QQP Quora Question Pairs is a binary classification task where the goal is to determine if two questions asked on Quora are semantically equivalent (Chen et al., 2018).

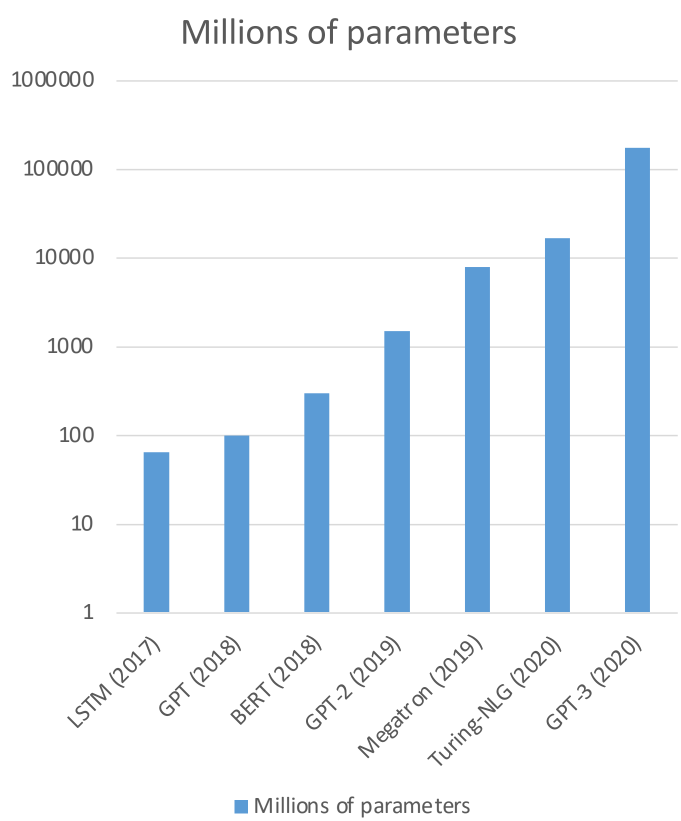
QNLI Question Natural Language Inference is a version of the Stanford Question Answering Dataset (Rajpurkar et al., 2016) which has been converted to a binary classification task (Wang et al., 2018a). The positive examples are (question, sentence) pairs which do contain the correct answer, and the negative examples are (question, sentence) from the same paragraph which do not contain the answer.

SST-2 The Stanford Sentiment Treebank is a binary single-sentence classification task consisting of sentences extracted from movie reviews with human annotations of their sentiment (Socher et al., 2013).

General Language Understanding Evaluation

System	MNLI-(m/mm)	QQP	QNLI	SST-2
	392k	363k	108k	67k
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4
OpenAI GPT	82.1/81.4	70.3	87.4	91.3
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**

Bigger is better?



OpenAI: GPT3: 175 billion parameters

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Agarwal, S. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Microsoft: Turing-NLG: 20 billion parameters

(February 2020) https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/

NVIDIA:Megatron: 8 billion

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*.

OpenAI: GPT-2: 1.5 billion parameters

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9.

Google: BERT (Transformer): 300 million parameters

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

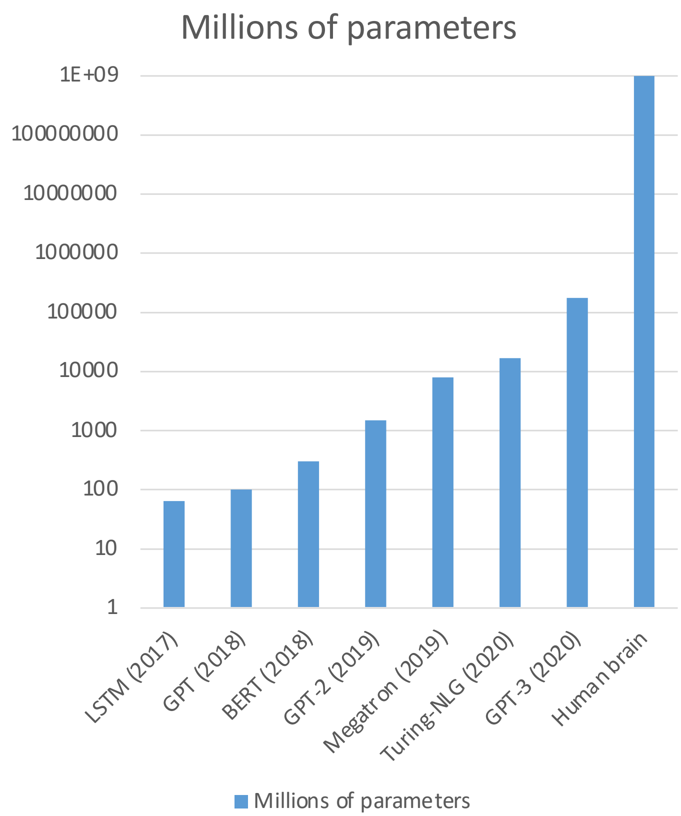
OpenAI: GPT (Transformer): 100 million parameters

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.

Google: Seq2Seq LSTM: 65 million parameters

Britz, D., Goldie, A., Luong, M. T., & Le, Q. (2017). Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*.

Bigger is better?



Human Brain: 10^15 connections
https://www.nature.com/articles/d41586-019-02208-0

OpenAI: GPT3: 175 billion parameters
Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Agarwal, S. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Microsoft: Turing-NLG: 20 billion parameters
(February 2020) https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/

NVIDIA:Megatron: 8 billion
Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*.

OpenAI: GPT-2: 1.5 billion parameters
Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9.

Google: BERT (Transformer): 300 million parameters
Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

OpenAI: GPT (Transformer): 100 million parameters
Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.

Google: Seq2Seq LSTM: 65 million parameters
Britz, D., Goldie, A., Luong, M. T., & Le, Q. (2017). Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*.