Self-Attention, Transformers & LLMs

Arin Ghazarian
Chapman University

Self-Attention

Attention is All You Need!

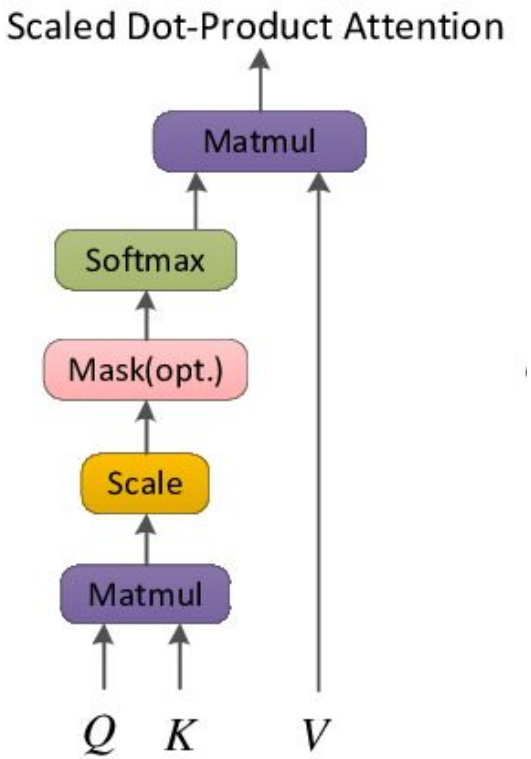
- Attention is All You Need!, Vaswani et al.
- In 2017, the transformer architecture introduced by Google Brain team eliminated the need for RNNs altogether.
- Self-attention is a replacement for recurrent models such as RNNs/LSTMs
- Self-attention models are much faster to train and perform much better compared to recurrent models

Self-Attention

- Attention Mechanism in RNNs suffer from a bottleneck problem: The current hidden representation must encode all of the information about the text observed so far
- Instead of relating an input to an output sequence, self-attention focuses on a single sequence
- Self-attention is the main idea behind transformer-based large language models
- It captures dependencies between the different tokens in a single sequence
- The term "self" means that attention is contained within a single sequence
    - E.g. in english to french translation, it only focuses on the english sentence
- Recurrent neural networks process sequences sequentially and each unit is dependant upon completion of the previous one, but self-attention process input tokens in parallel which allows for fast processing

Scaled-Dot Product

- The query weights W_Q, the key weights W_K, and the value weights W_V
- For each token i, the input word embedding x_i is multiplied with each of the three weight matrices to produce
  - query vector: $q_i = x_i W_Q$
  - key vector: $k_i = x_i W_K$
  - Value vector: $v_i = x_i W_V$
- The fact that W_Q and W_K are different matrices allows attention to be non-symmetric: if token i attends to token j (i.e. $q_i k_j$ is large), this does not necessarily mean that token j will attend to token i (i.e. $q_j k_i$ could be small).



source

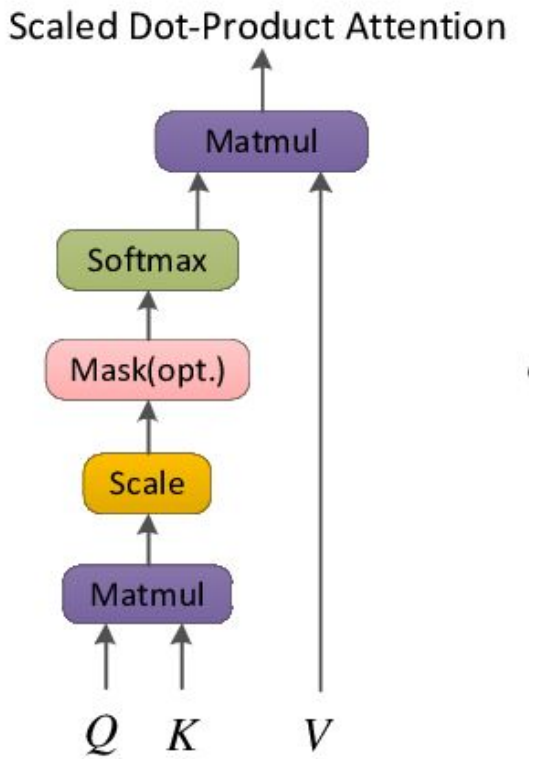
Scaled-Dot Product

- Attention weights are calculated using the query and key vectors
- The attention weight a_{ij} from token i to token j is the dot product between q_i and k_j.
- The attention weights are divided by the square root of the dimension of the key vectors sqrt(d_k), which stabilizes gradients during training, and passed through a softmax which normalizes the weights.
- The output of the attention unit for token i is the weighted sum of the value vectors of all tokens, weighted by a_{ij}, the attention from token i to each token.

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

Masked Attention

- It may be necessary to cut out attention links between some word-pairs.
- For example, the decoder for token position t should not have access to token position t+1.
- This may be accomplished before the softmax stage by adding a mask matrix
- The mask matrix M has negative infinity at entries where the attention link must be cut, and zero at other places.



source

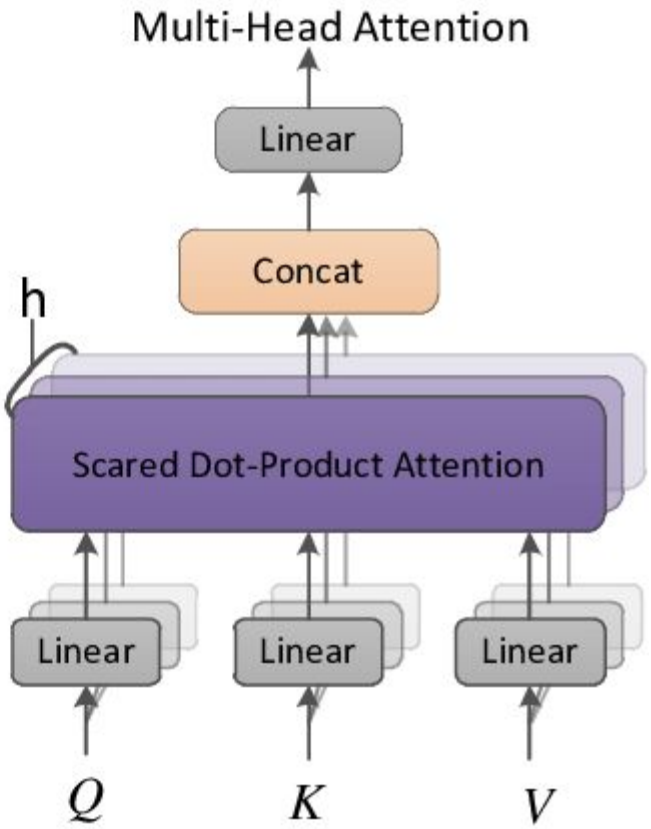
Multi-Headed Self-Attention

- One set of W^Q,W^K,W^V matrices is called an attention head, and each layer in a transformer model has multiple attention heads.
- This is similar to learning multiple kernels in one layer of CNNs
- With multiple attention heads the model can do this for different definitions of "relevance".
    - Many transformer attention heads encode relevance relations that are meaningful to humans. For example, some attention heads can attend mostly to the next word, while others mainly attend from verbs to their direct objects.
- The outputs from the attention layer are concatenated to pass into the feed-forward neural network layers

Multi-Headed Self-Attention

- Where the matrices W_i^Q, W_i^K, W_i^V are "projection matrices" owned by individual attention head i, and W^O is a final projection matrix owned by the whole multi-headed attention head.

$$\text{MultiheadedAttention}(Q, K, V) = \text{Concat}(\text{Attention}(QW_i^Q, KW_i^K, VW_i^V))W^O$$

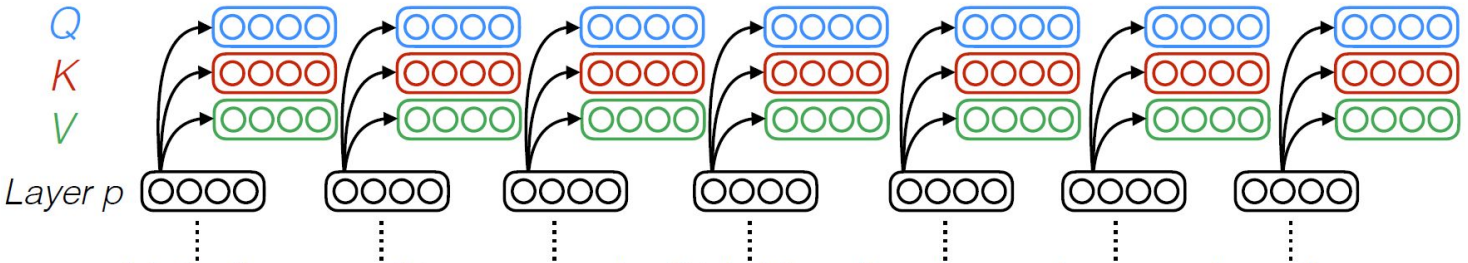


source

Slides by Emma Strubell!

Self-attention

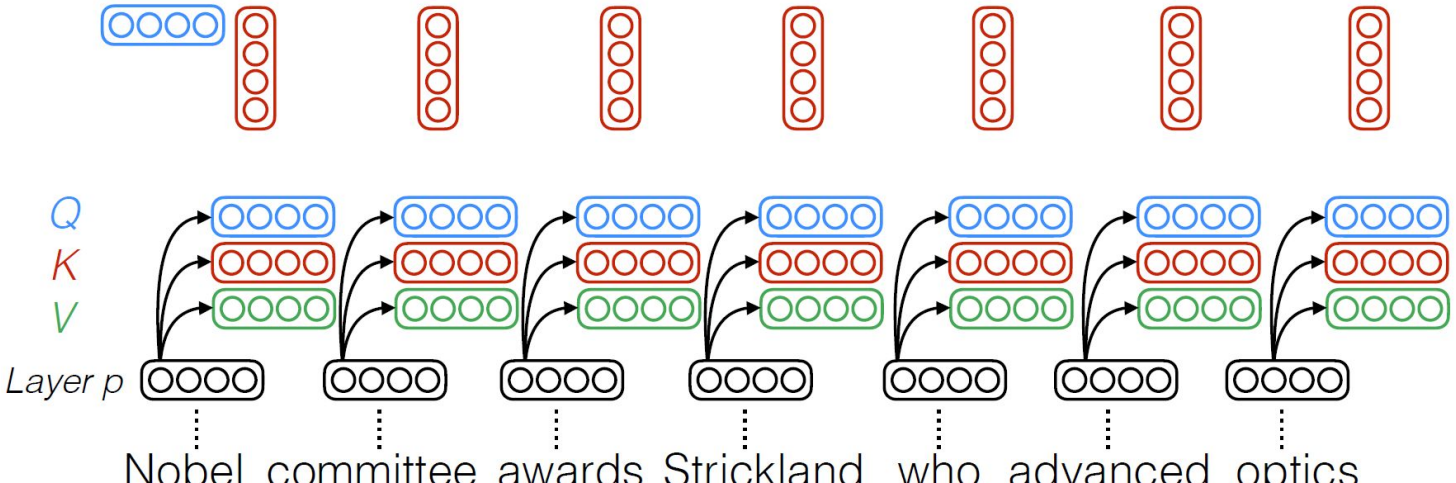
[Vaswani et al. 2017]



Slides by Emma Strubell!

Self-attention

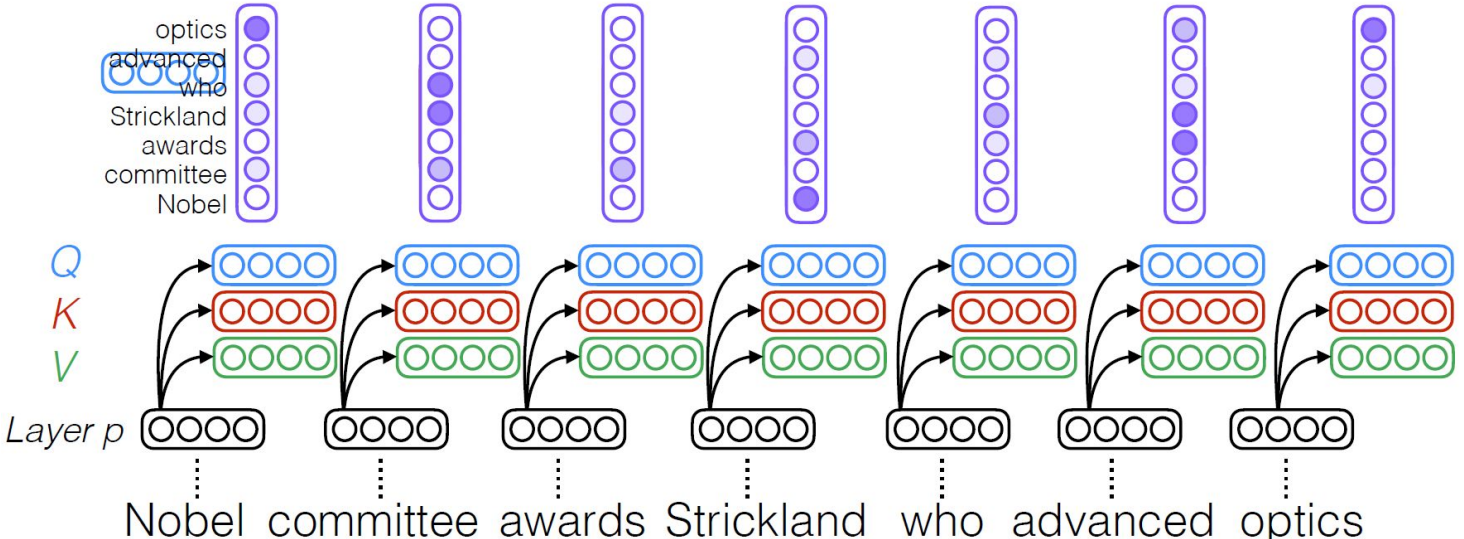
[Vaswani et al. 2017]



Slides by Emma Strubell!

Self-attention

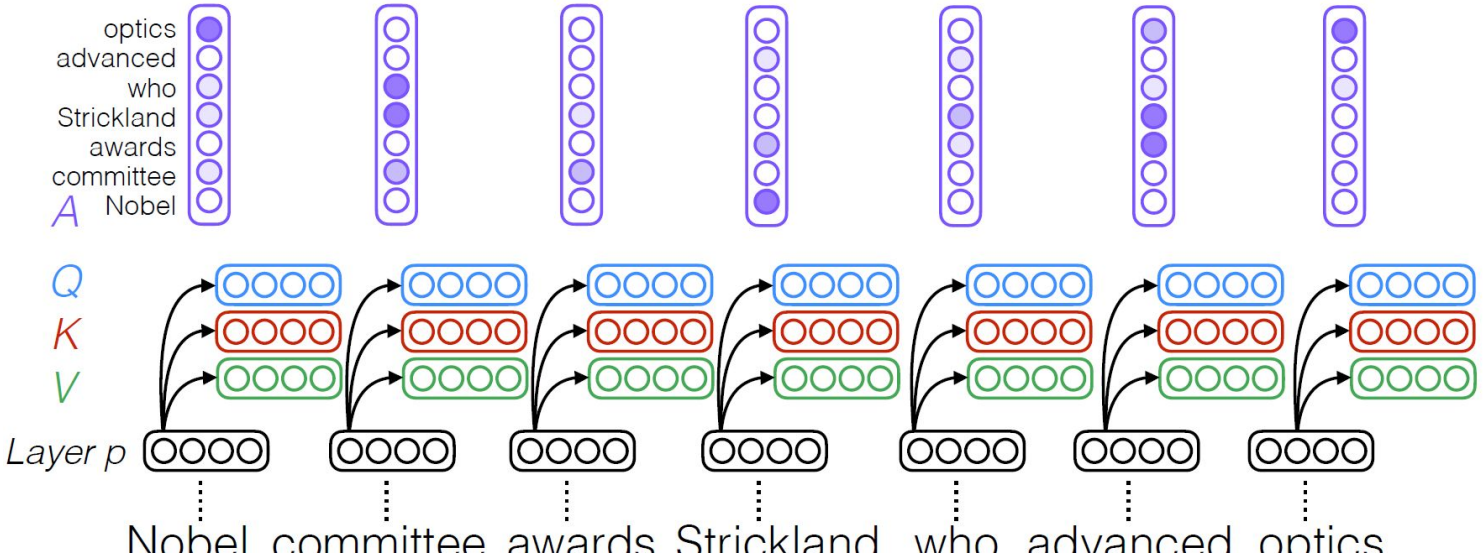
[Vaswani et al. 2017]



Slides by Emma Strubell!

Self-attention

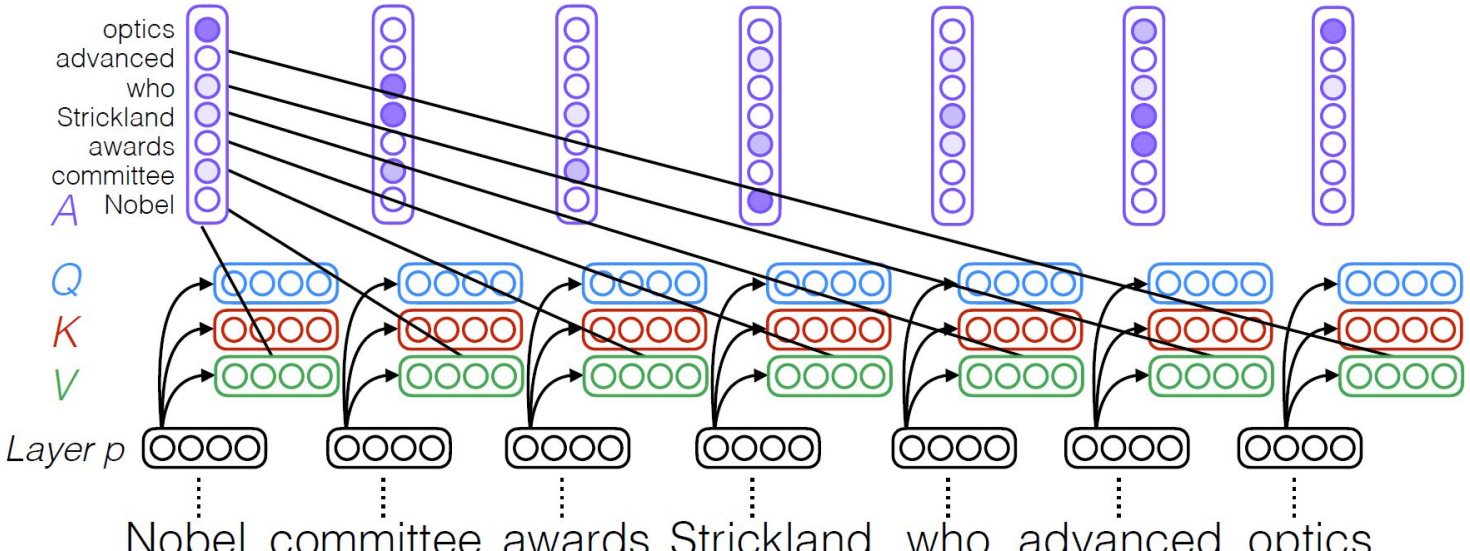
[Vaswani et al. 2017]



Slides by Emma Strubell!

Self-attention

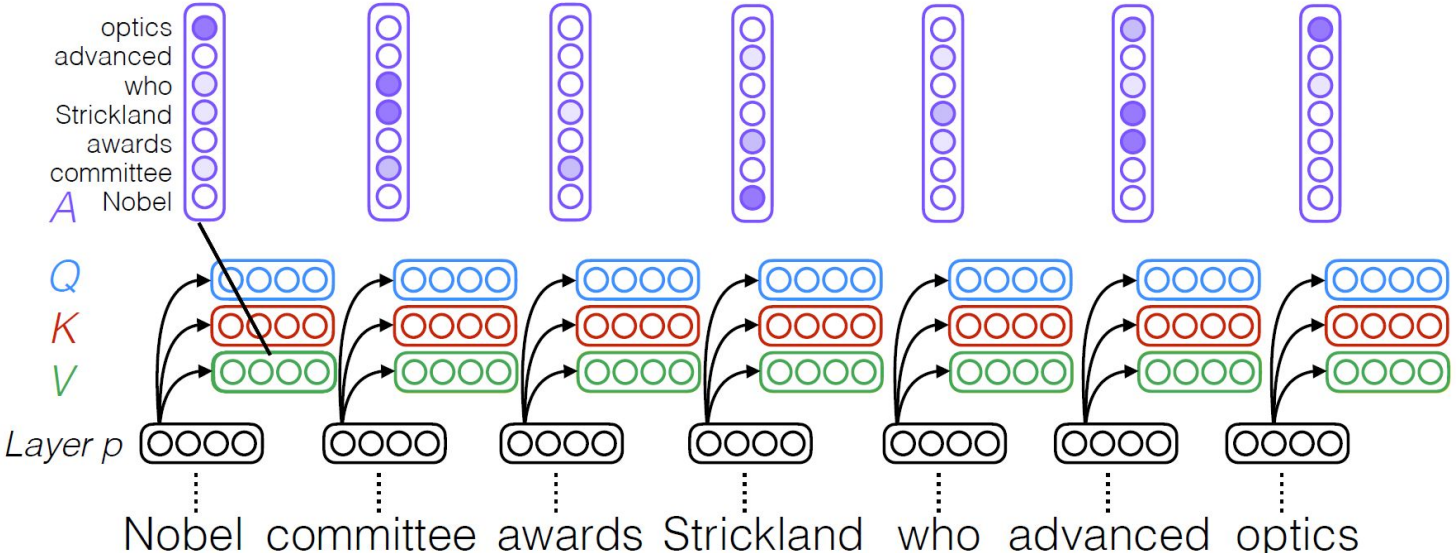
[Vaswani et al. 2017]



Slides by Emma Strubell!

Self-attention

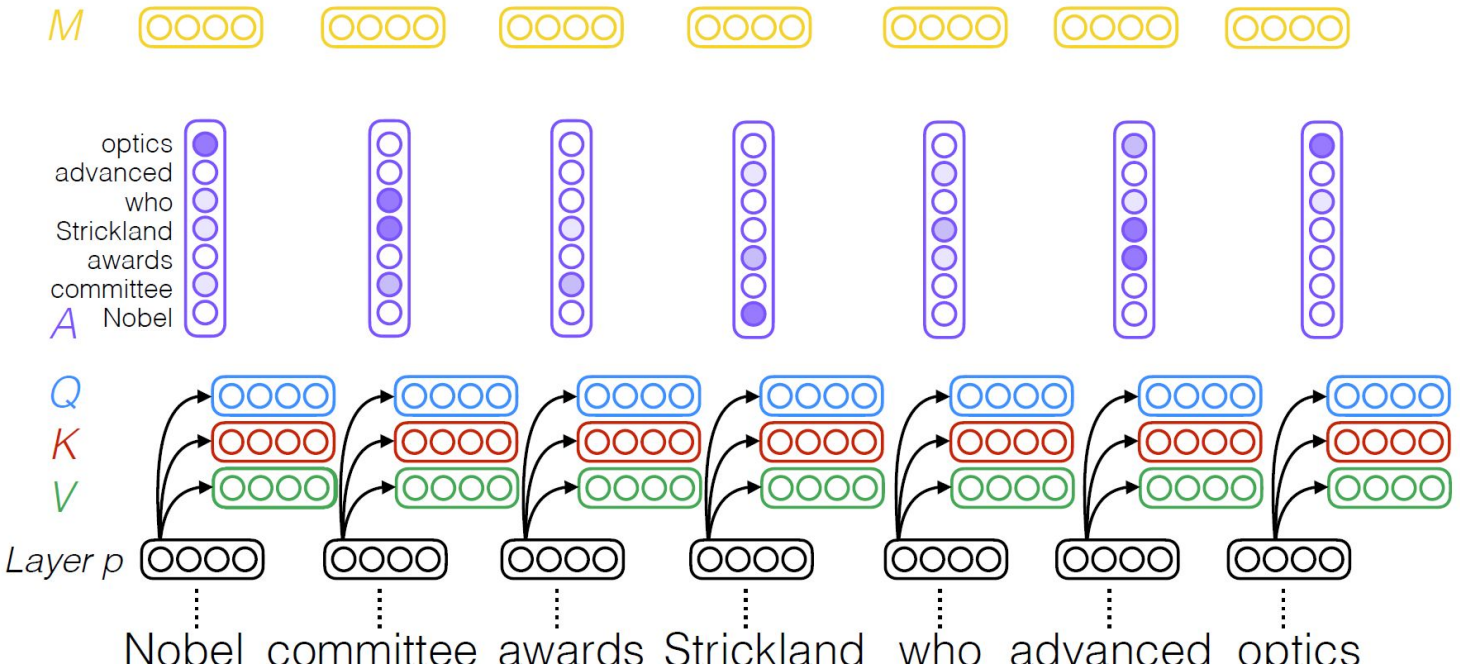
[Vaswani et al. 2017]



Slides by Emma Strubell!

Self-attention

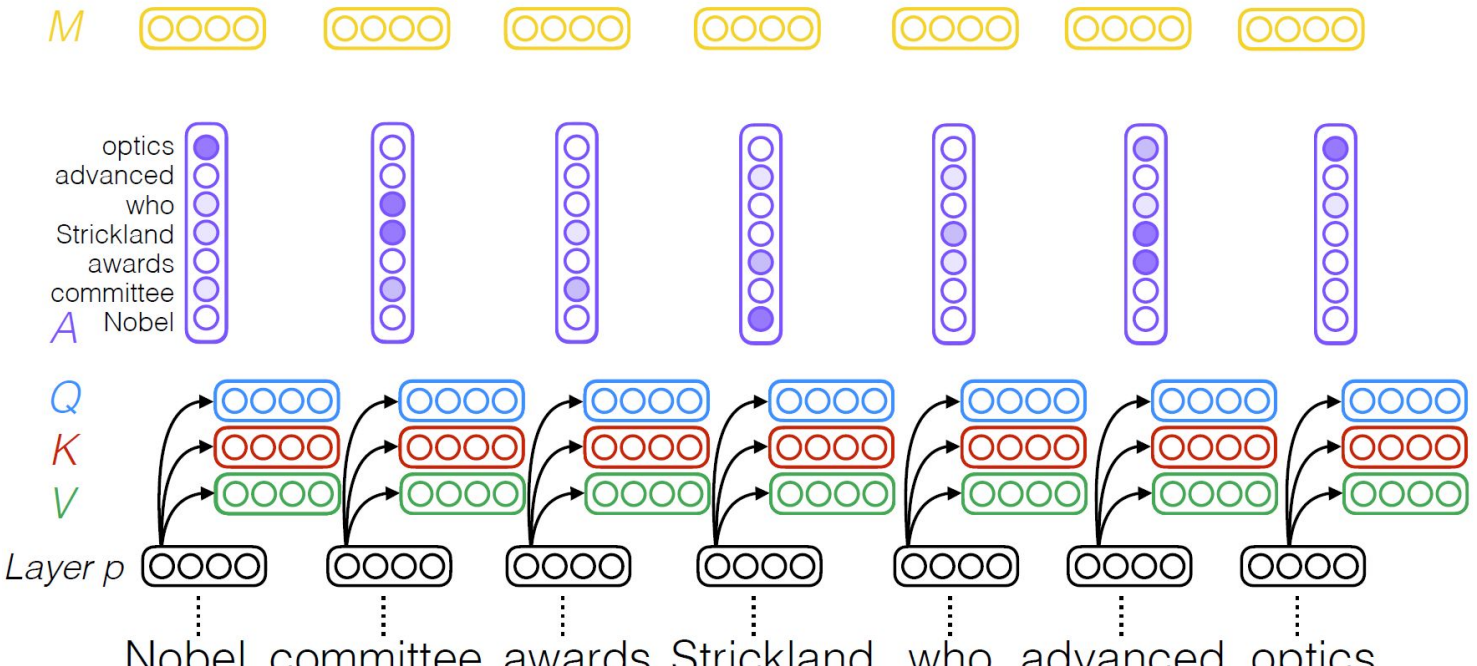
[Vaswani et al. 2017]



Slides by Emma Strubell!

Self-attention

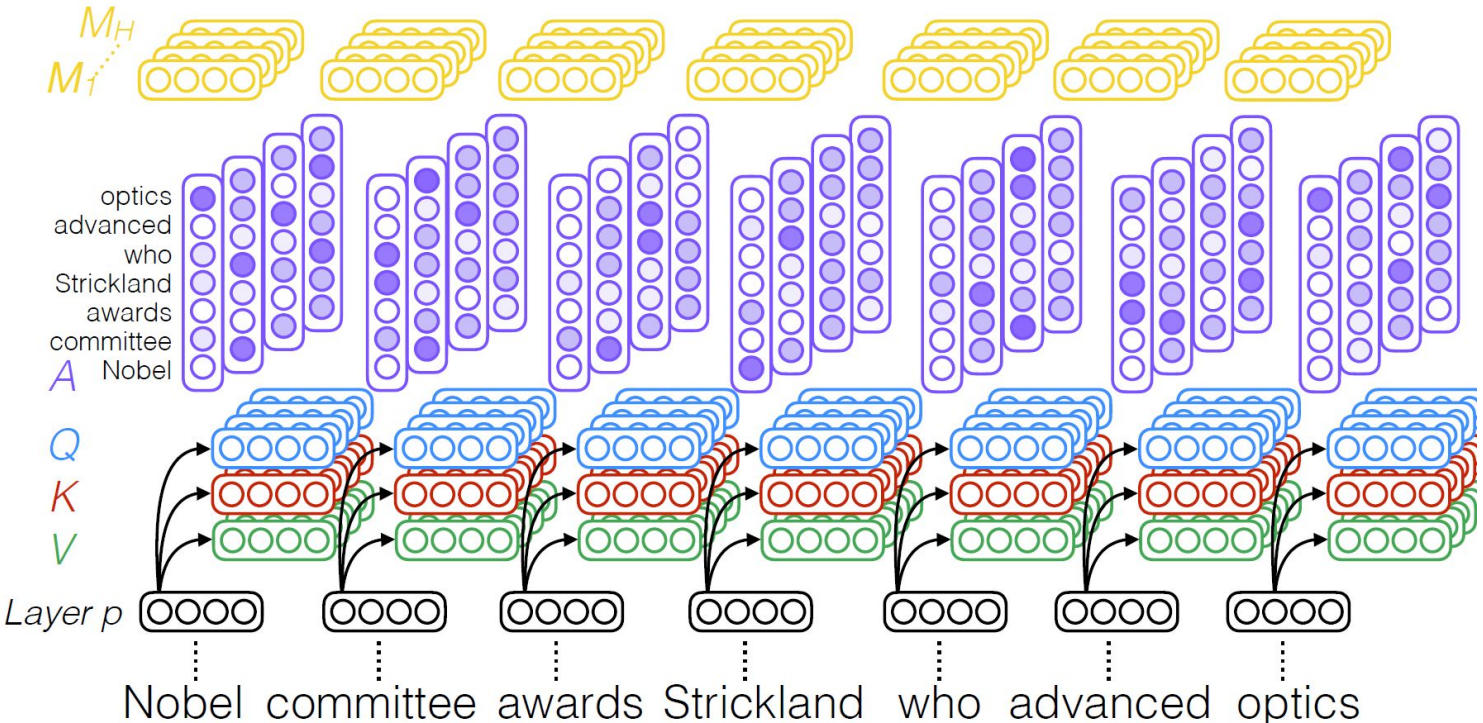
[Vaswani et al. 2017]



Slides by Emma Strubell!

[Vaswani et al. 2017]

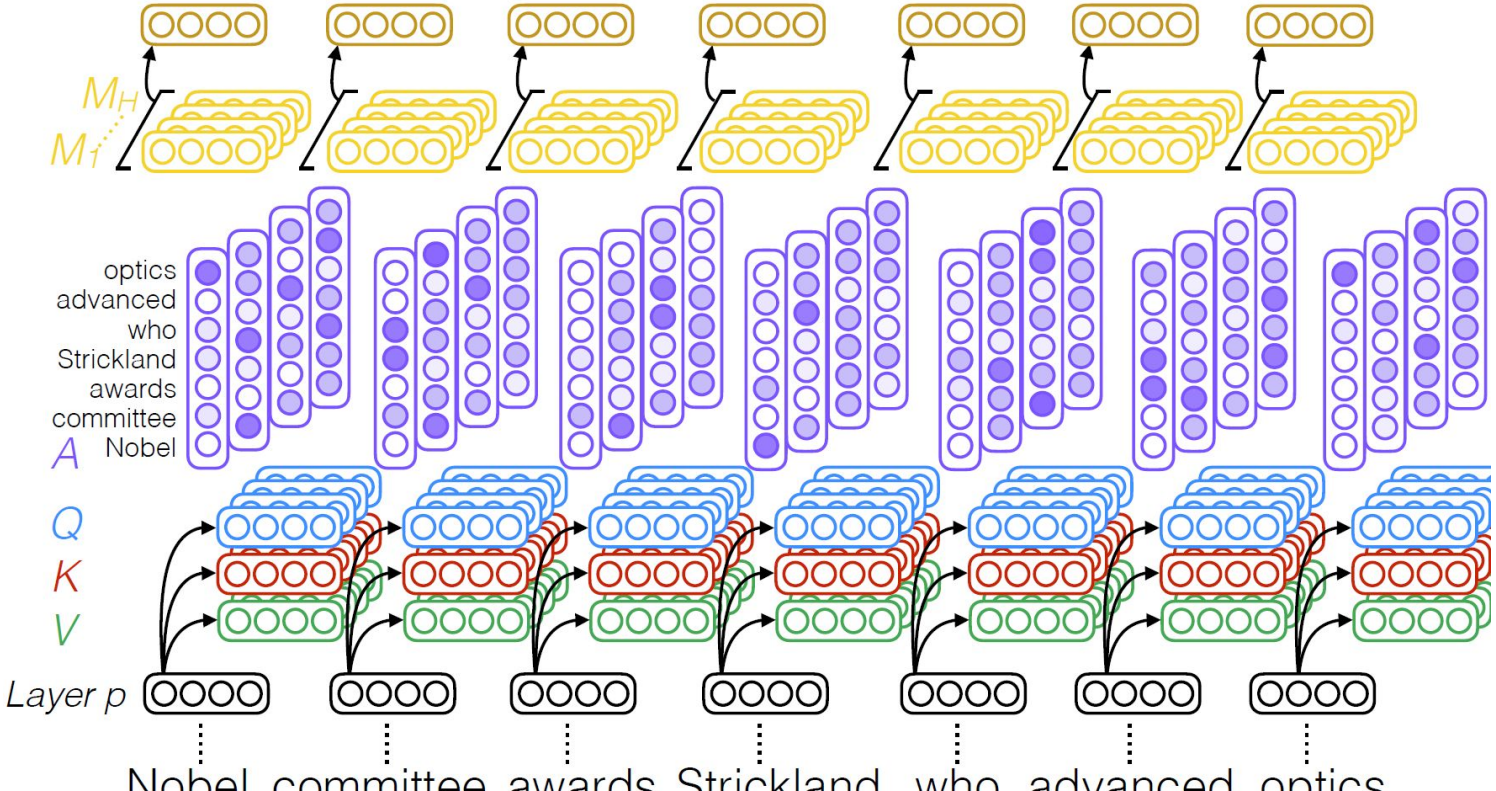
Multi-head self-attention



Slides by Emma Strubell!

[Vaswani et al. 2017

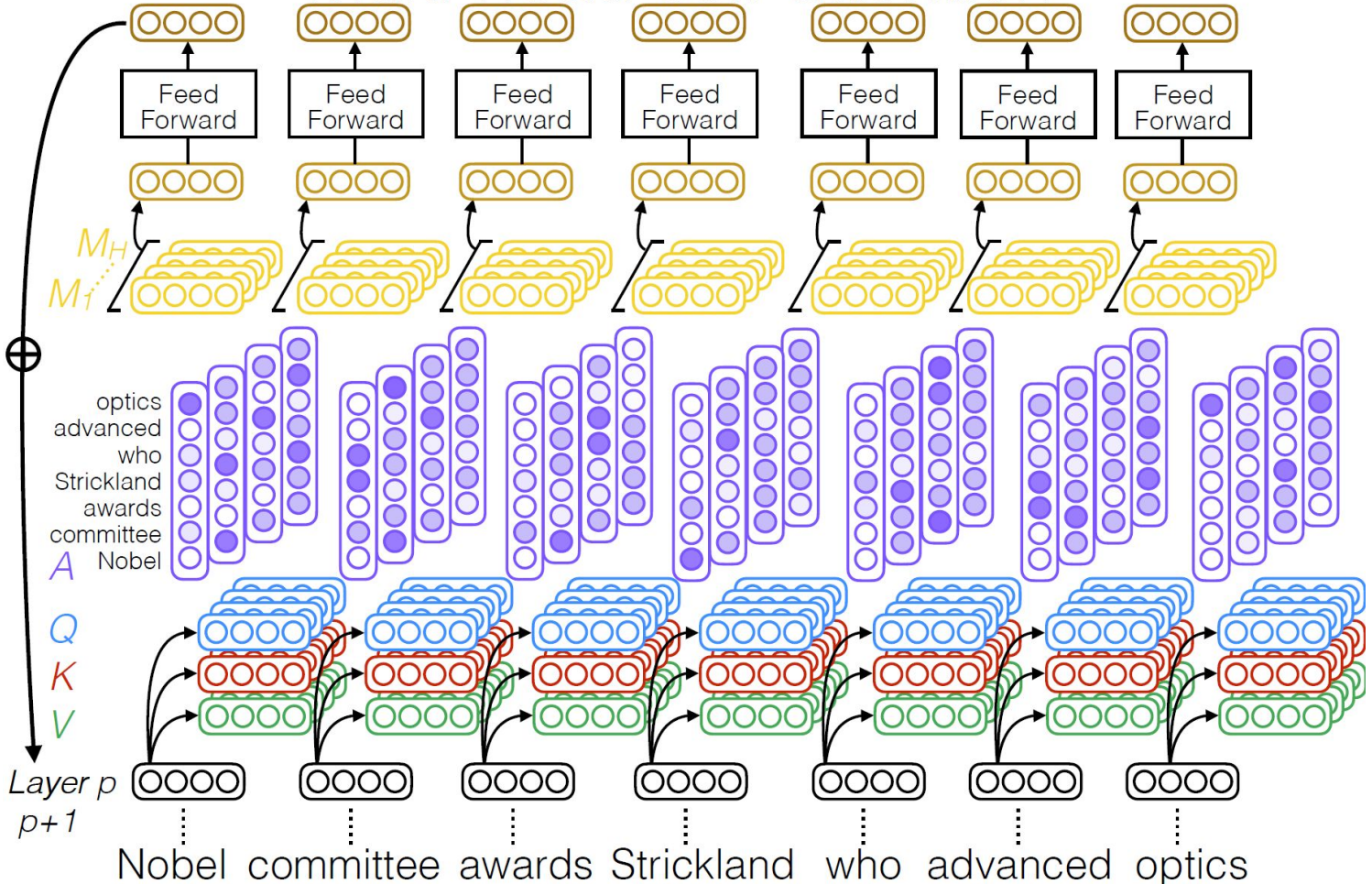
Multi-head self-attention



Slides by Emma Strubell!

[Vaswani et al. 2017]

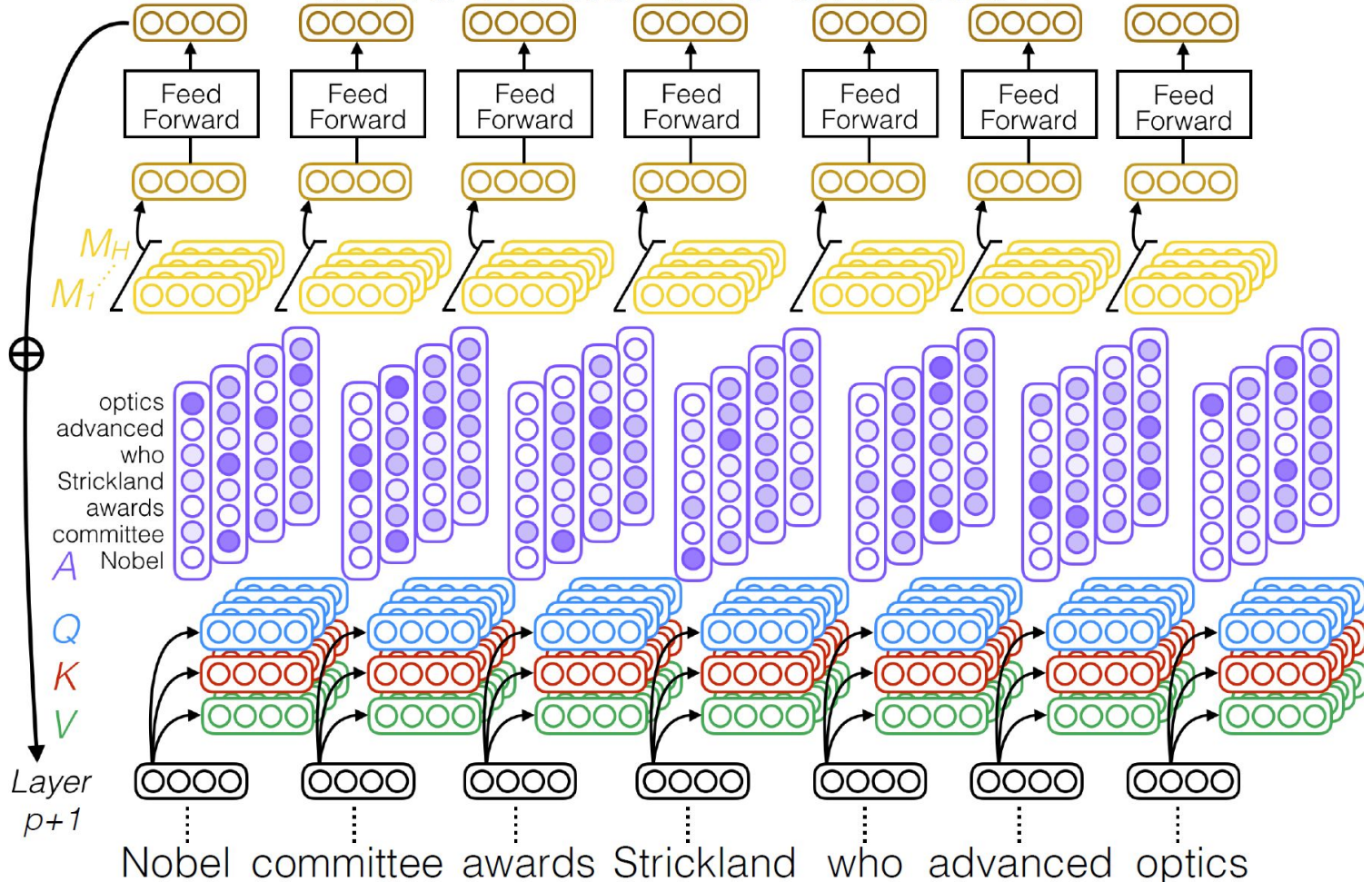
Multi-head self-attention



Slides by Emma Strubell!

[Vaswani et al. 2017]

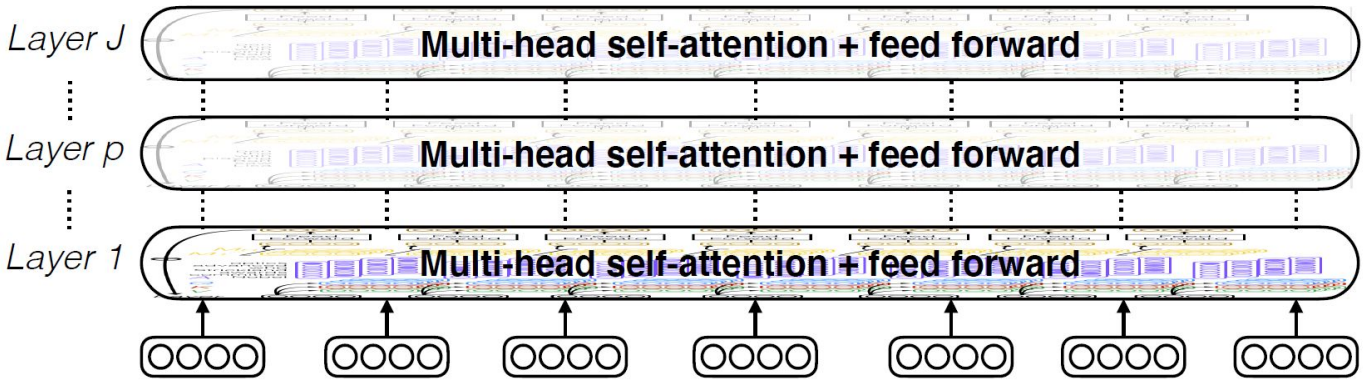
Multi-head self-attention



Slides by Emma Strubell!

[Vaswani et al. 2017]

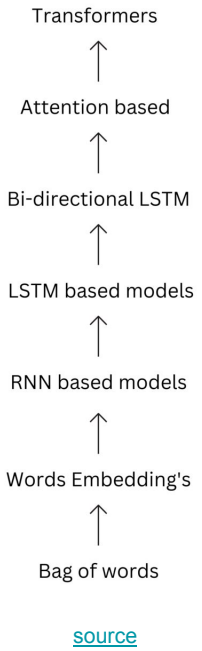
Multi-head self-attention



Transformers

Transformers

- Transformers were introduced in 2017 by a team at Google Brain
- A transformer is a deep learning model
- Do not have a recurrent structure
- The model of choice for NLP problems replacing RNN models such as LSTM
- Transformers use an attention mechanism without an RNN, processing all tokens simultaneously and calculating attention weights between them in successive layers.
- Since the attention mechanism only uses information about other tokens from lower layers, it can be computed for all tokens in parallel, which leads to improved training speed, allowing training on larger datasets

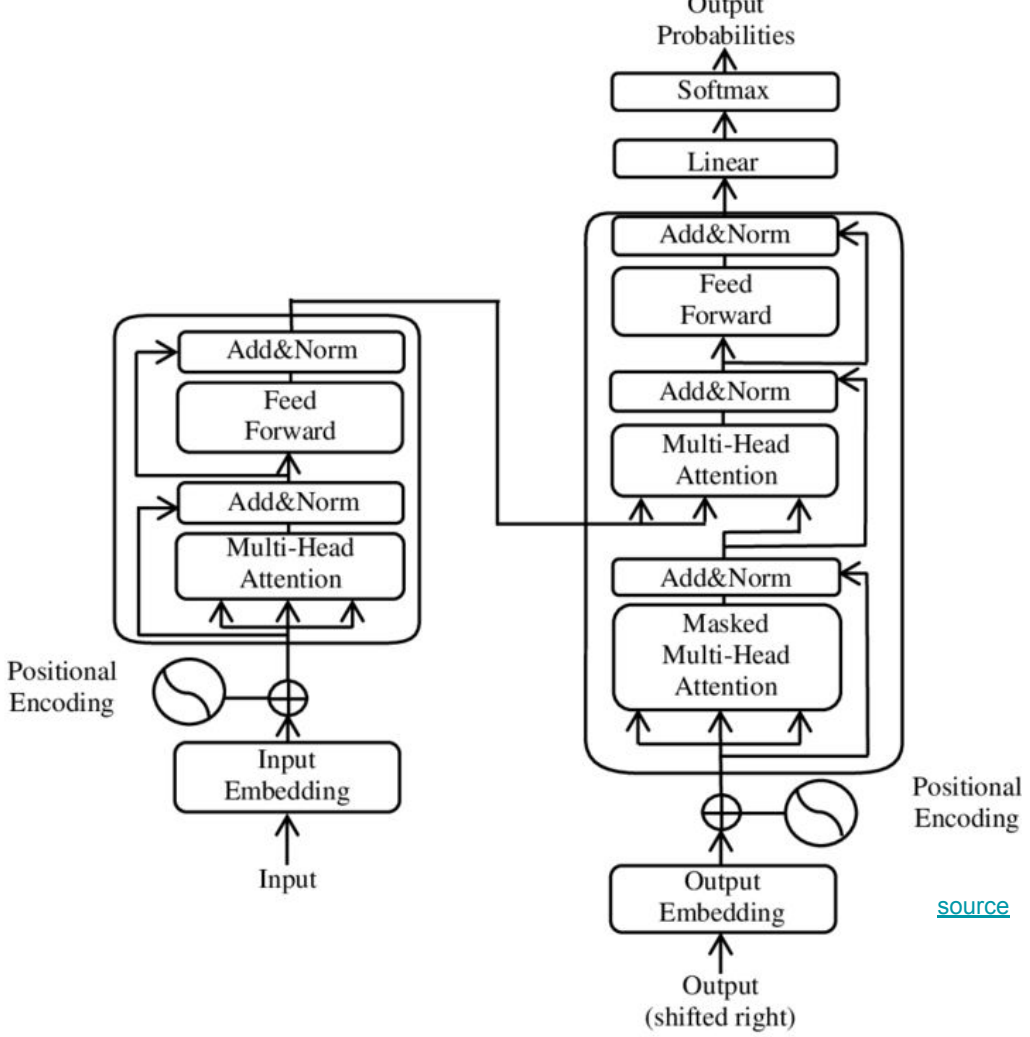


source

Transformers

- Like recurrent neural networks (RNNs), transformers are designed to process sequential input data, such as natural language, with applications towards tasks such as translation and text summarization.
- Unlike RNNs, transformers process the entire input all at once.
- BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) are transformer models trained on large language datasets, such as the Wikipedia Corpus and Common Crawl
- The transformer building blocks are scaled dot-product attention units.

Transformers



source

Encoder

- Each encoder consists of two major components: a self-attention mechanism and a feed-forward neural network.
- The self-attention mechanism accepts input encodings from the previous encoder and weights their relevance to each other to generate output encodings.
  - BERT base Has 12 layers, each with 12 attention heads, for a total of 144 attention mechanisms
- The first encoder takes positional information and embeddings of the input sequence as its input, rather than encodings.
- The feed-forward neural network further processes each output encoding individually. These output encodings are then passed to the next encoder as its input, as well as to the decoders.

Encoder

- When a sentence is passed into a transformer model, attention weights are calculated between every token simultaneously.
- The attention unit produces embeddings for every token in context that contain information about the token itself along with a weighted combination of other relevant tokens each weighted by its attention weight.
- The encoder is bidirectional. Attention can be placed on tokens before and after the current token.

Positional Encoding

- A positional encoding is a fixed-size vector representation that encapsulates the relative positions of tokens within a target sequence

$$(f(t)_{2k}, f(t)_{2k+1}) = (\sin(\theta), \cos(\theta)) \quad \forall k \in \{0, 1, \ldots, d/2 - 1\}$$

where $\theta = \frac{t}{r^k}, r = N^{2/d}$.

- Here, N is a free parameter that should be significantly larger than the biggest k that would be input into the positional encoding function. In the original paper, the authors chose N = 10000. The function is in a simpler form when written as a complex function of type:

$$f(t) = \left(e^{it/r^k}\right)_{k=0,1,\ldots,\frac{d}{2}-1}$$

where $r = N^{2/d}$.

Positional Encoding

- The main reason the authors chose this as the positional encoding function is that it allows one to perform shifts as linear transformations:

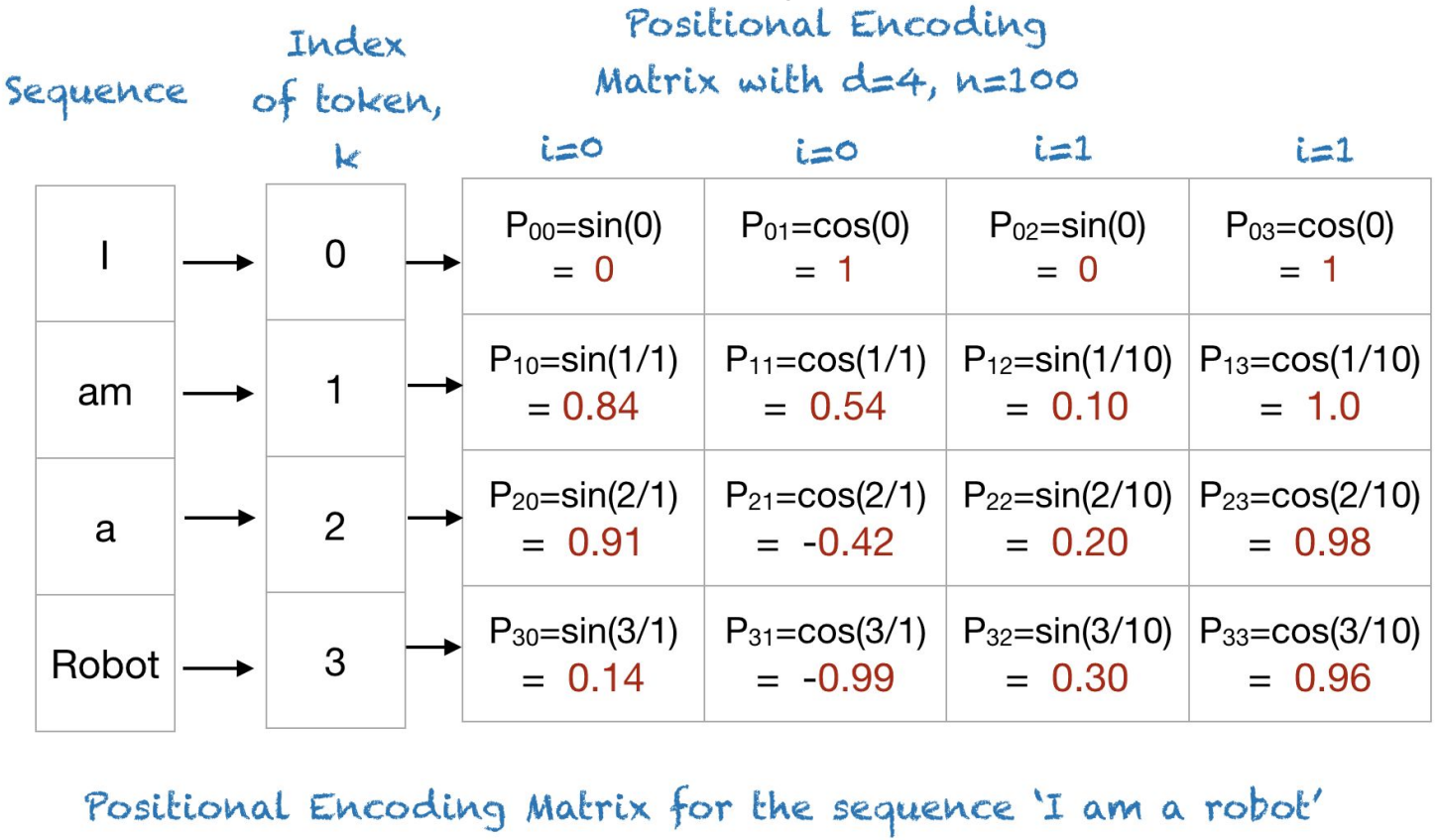
$$f(t+\Delta t) = \mathrm{diag}(f(\Delta t))f(t)$$

- This allows the transformer to take any encoded position, and find the encoding of the position n-steps-ahead or n-steps-behind, by a matrix multiplication.

Positional Encoding: Why Not to Use a Single Number?

- There are multiple reasons why a single number like the index value of each token is not used to represent an item’s position in transformer models.
  - If the sequence is long, the indices can grow very large in magnitude.
  - If we normalize the index to be between 0 and 1, then variable length sequences would be normalized differently

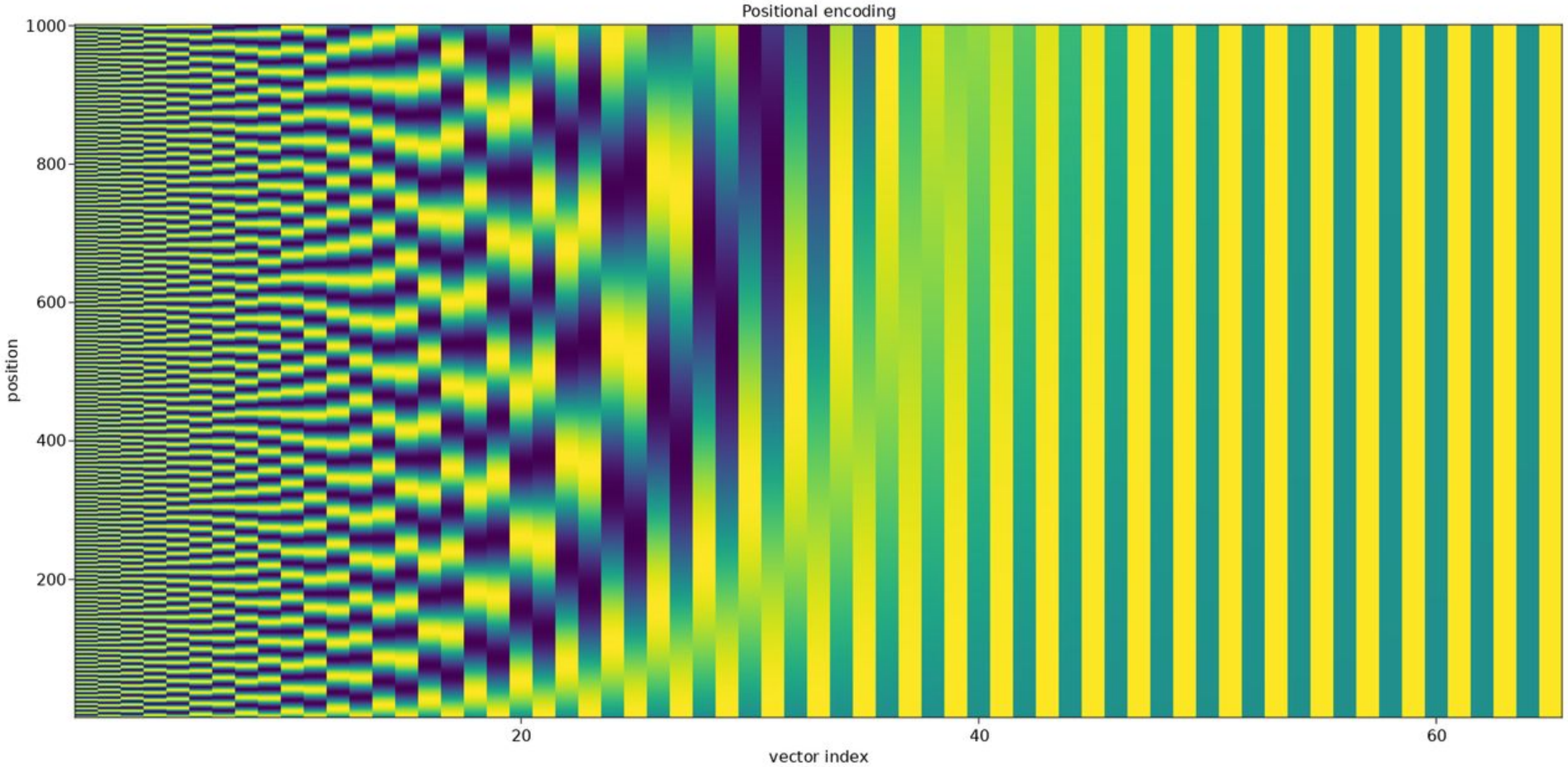
- Each i generates two values (even and odd): a sine and a cosine
- Each position is a vector
- d Dimension of the output embedding space



Positional Encoding Matrix for the sequence 'I am a robot'

https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

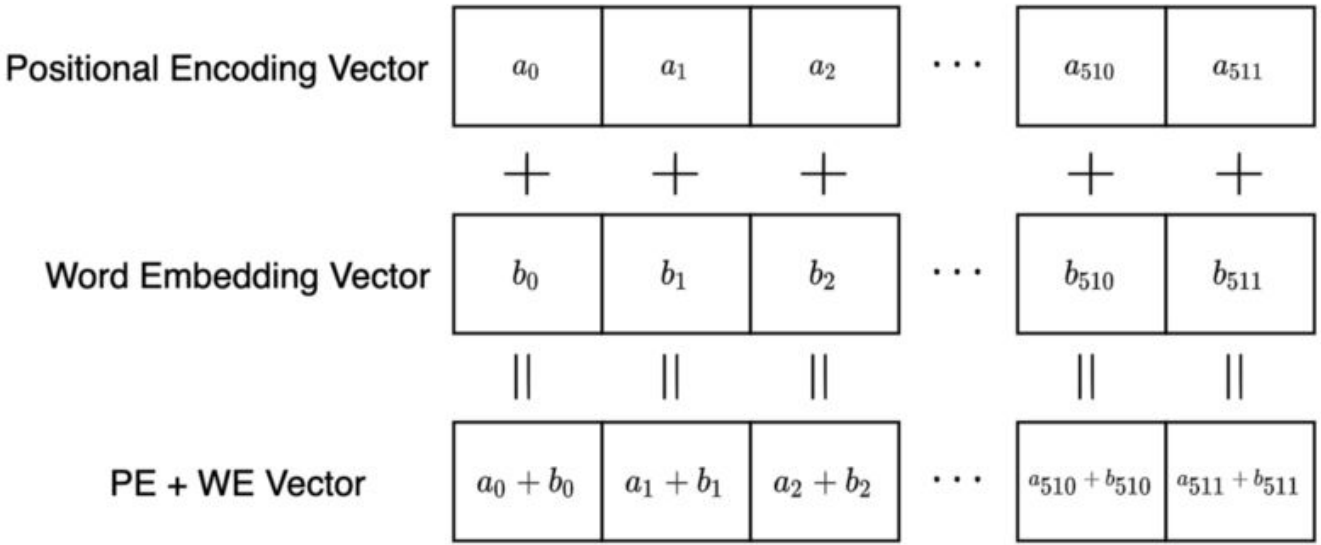
A diagram of a sinusoidal positional encoding with parameters [source]



Positional Encoding vector of bigger position indices have less number of alternating 0s and 1s (yellow and blue).

Input Vector to Transformers

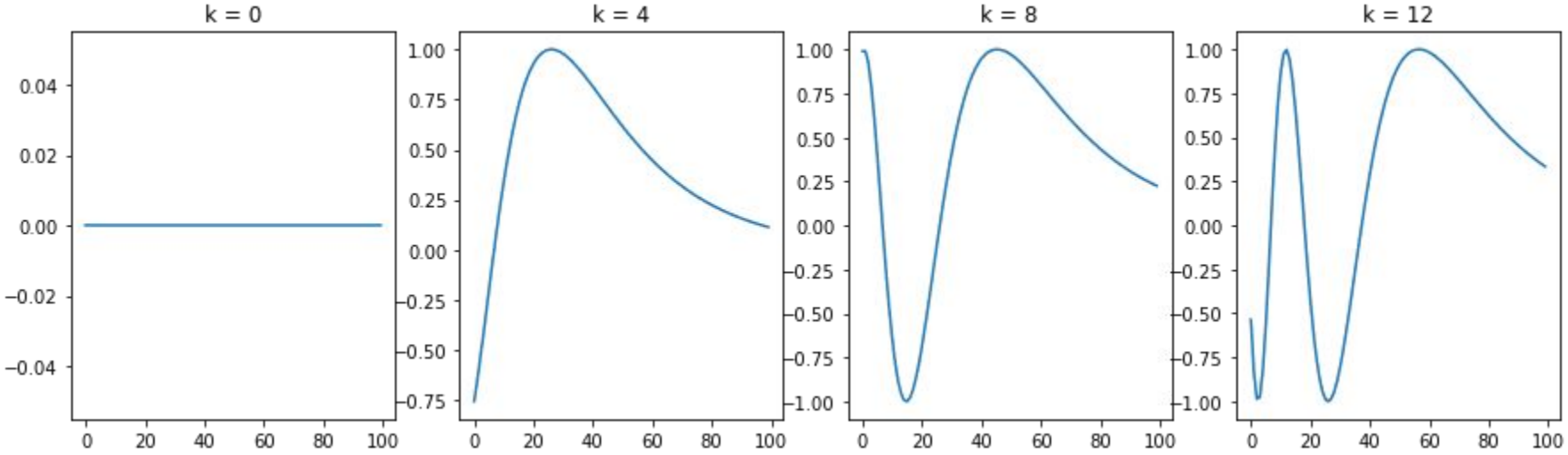
- We add the word embeddings and positional encodings (should be of the same dimension) and use it as the input to transformer models



https://kikaben.com/transformers-positional-encoding/

Positional Encoding

- Each position corresponds to a different sinusoid over the i values (column number in the vector)



The first 100 values of the positional encoding vectors for positions 0,4,8,12

Positional Encoding

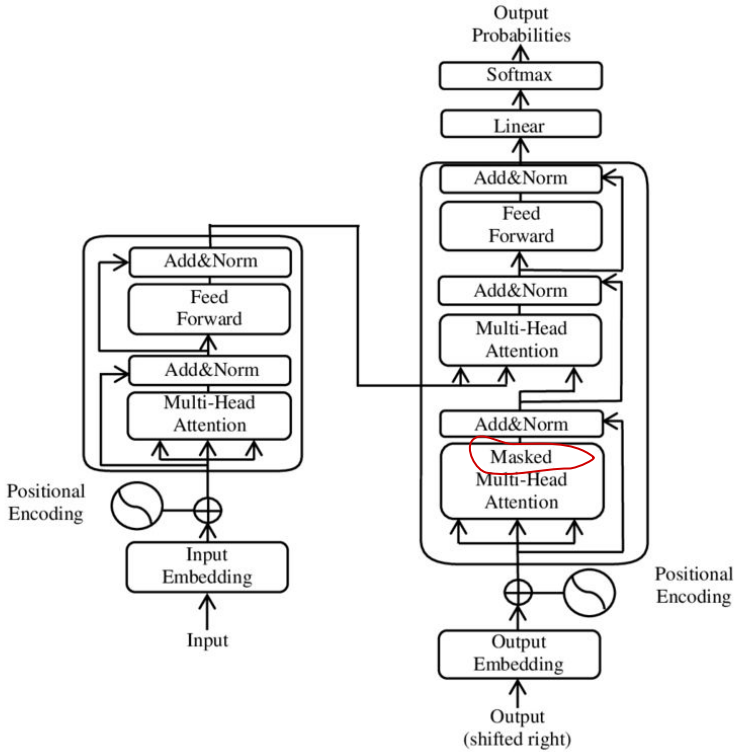
- https://kikaben.com/transformers-positional-encoding/

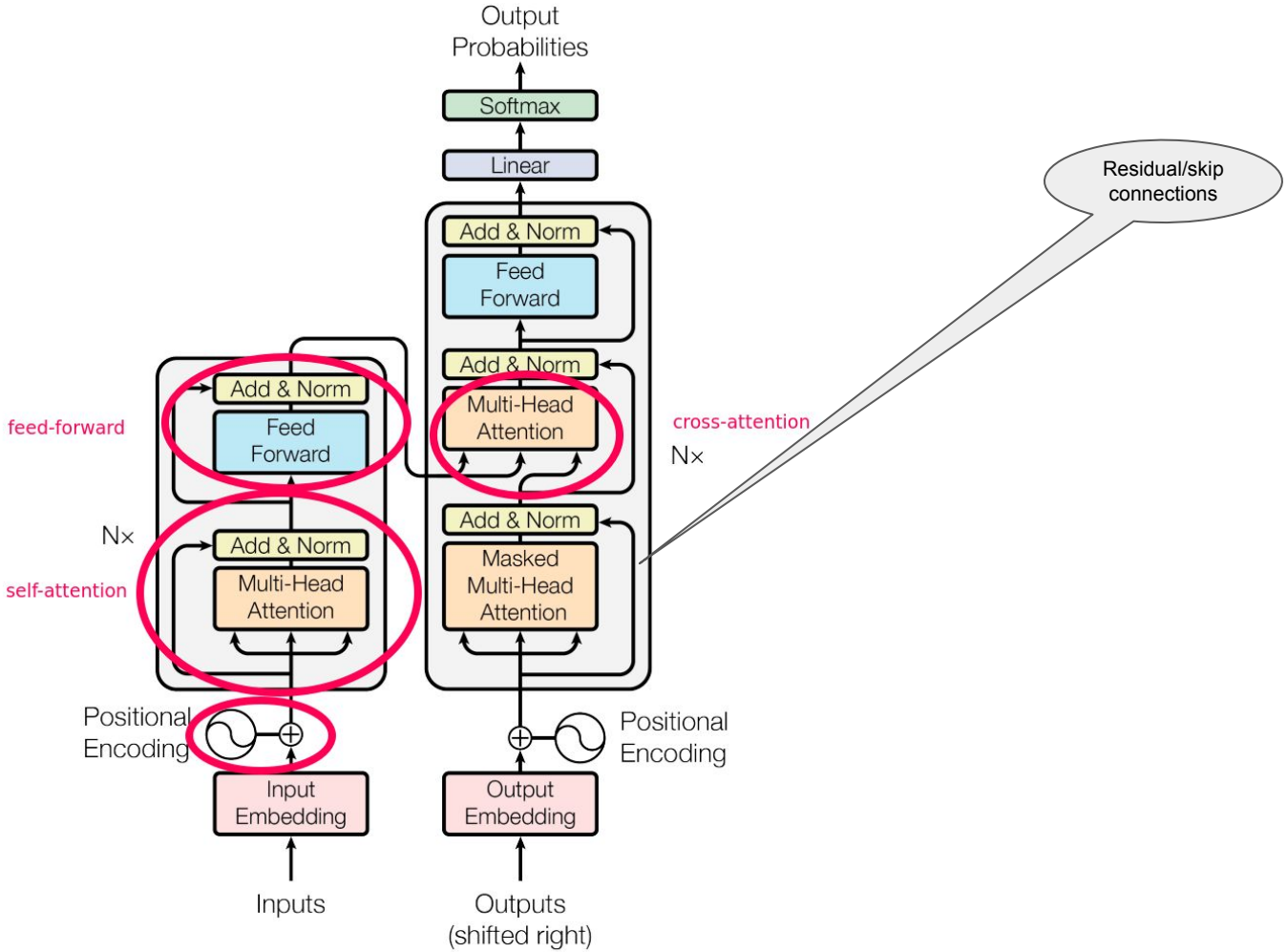
Decoder

- Each decoder consists of three major components: a self-attention mechanism, an attention mechanism over the encodings, and a feed-forward neural network.
- The decoder functions in a similar fashion to the encoder, but an additional attention mechanism is inserted which instead draws relevant information from the encodings generated by the encoders. This mechanism can also be called the encoder-decoder attention
- Like the first encoder, the first decoder takes positional information and embeddings of the output sequence as its input, rather than encodings. The transformer must not use the current or future output to predict an output, so the output sequence must be partially masked to prevent this reverse information flow.

Decoder

- For all attention heads, attention can't be placed on following tokens, thus we use masked attentions
- The last decoder is followed by a final linear transformation and softmax layer, to produce the output probabilities over the vocabulary.

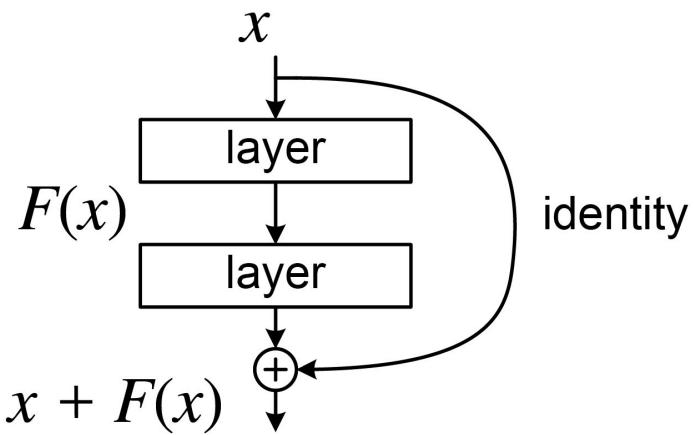




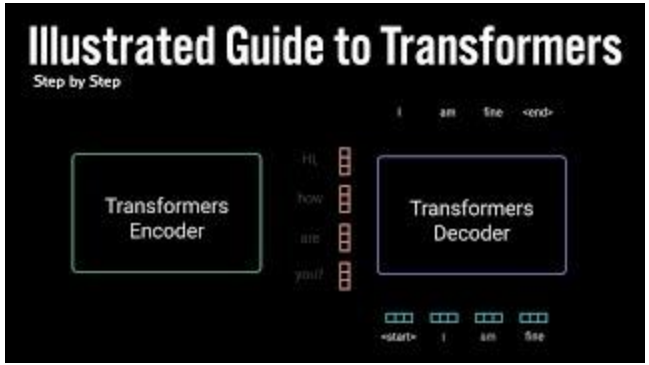
https://vaclavkosar.com/ml/transformers-self-attention-mechanism-simplified

Residual Connections

- Skip connections that perform identity mappings, merged with the layer outputs by addition.
- Residual Networks improves the training of very deep neural network models



source



https://www.youtube.com/watch?v=4Bdc55j80l8

Keras/Tensorflow

https://www.tensorflow.org/text/tutorials/transformer

https://pyimagesearch.com/2022/09/05/a-deep-dive-into-transformers-with-tensorflow-and-keras-part-1/

https://keras.io/examples/nlp/text_classification_with_transformer/

https://keras.io/examples/timeseries/timeseries_transformer_classification/

Online Resources

- https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a
- https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html
- https://alibaba-cloud.medium.com/self-attention-mechanisms-in-natural-language-processing-9f28315ff905
- Self-attention coursera video
- https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0

Large Language Models

Large Language Models (LLMs)

- Deep Learning models neural networks with many parameters (billions) for language tasks
- Trained on large quantities of unlabeled text using self-supervised learning or semi-supervised learning.
- Emerged around 2018
- Mostly have a transformer architecture
- The output of LLM is usually a probability distribution over its vocabulary

Large Language Models (LLMs)

- Computationally expensive ($) to train.
- Training a 1.5 billion parameter model costs around $1.6 million

source

General Models

- General Language models which excel at a wide range of tasks (vs the traditional approach of of training specialized supervised models for specific tasks)
- Trained for the task of predicting the next word in a sentence
- LLMs demonstrate considerable general knowledge about the world
- Memorize a large number of facts

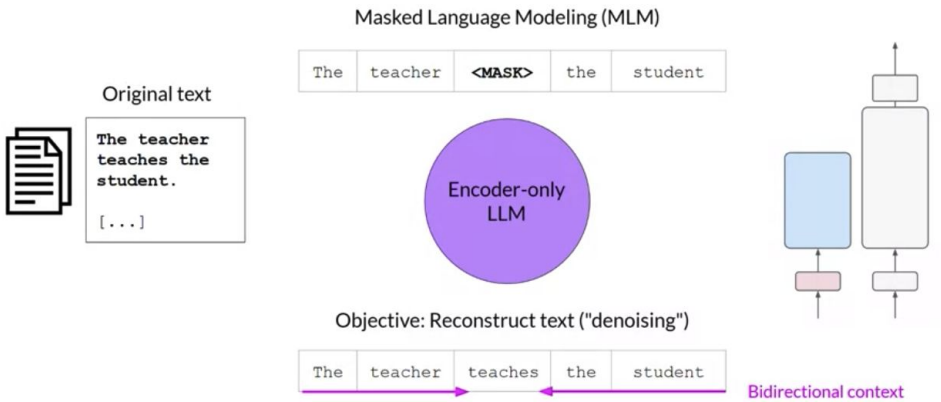
Large textual datasets

- Up to 10 trillion words in size.
  - Common Crawl
  - The Pile
  - MassiveText
  - Wikipedia
  - GitHub.
  - BookCorpus

Hallucination

- LLMs sometimes have confidently asserted claims of fact which do not seem to be justified by their training data (Hallucination Phenomena in LLMs)
  - **Factual inaccuracies:** When a model presents information that's false, but seems factual, such as incorrect dates, events, or statistics
  - **Generated sources or quotations:** When a model fabricates quotes or citations, such as attributing a statement to a real person who didn't say it, or creating a fictitious source
  - **Logical inconsistencies**: When a model produces responses that are internally inconsistent or logically flawed

Autoencoding models



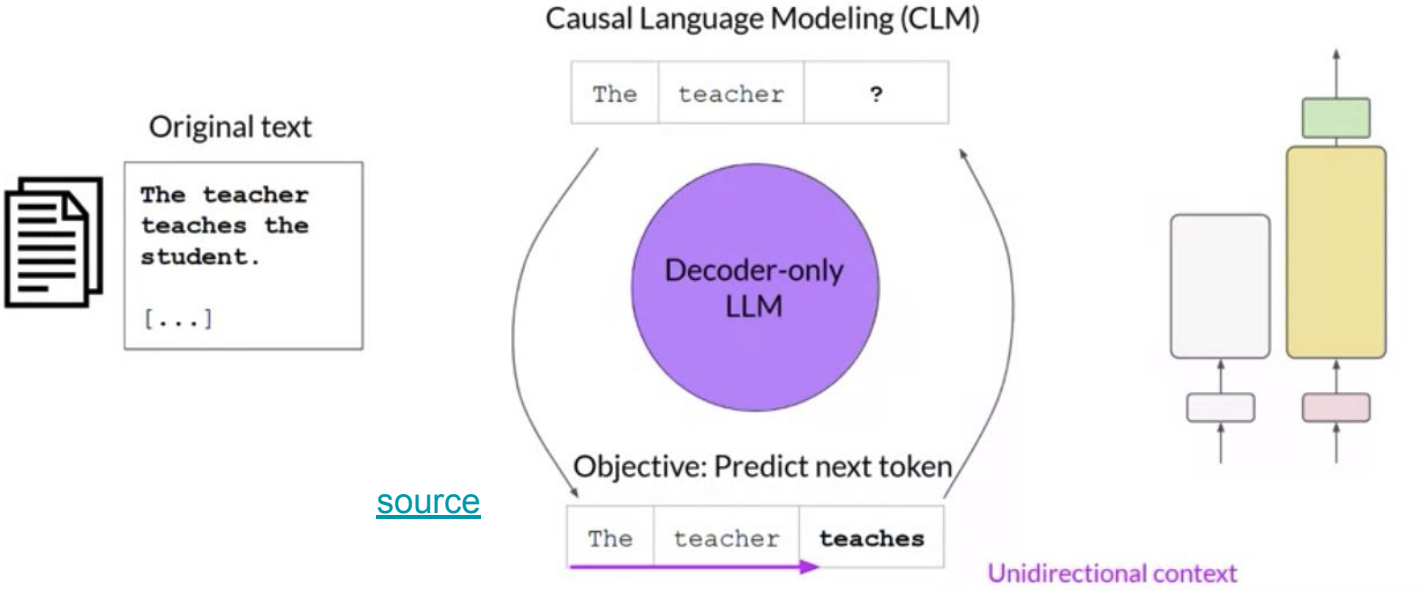
Good use cases:

- Sentiment analysis
- Named entity recognition
- Word classification

Example models:

- BERT
- ROBERTA

source



source

Autoregressive models

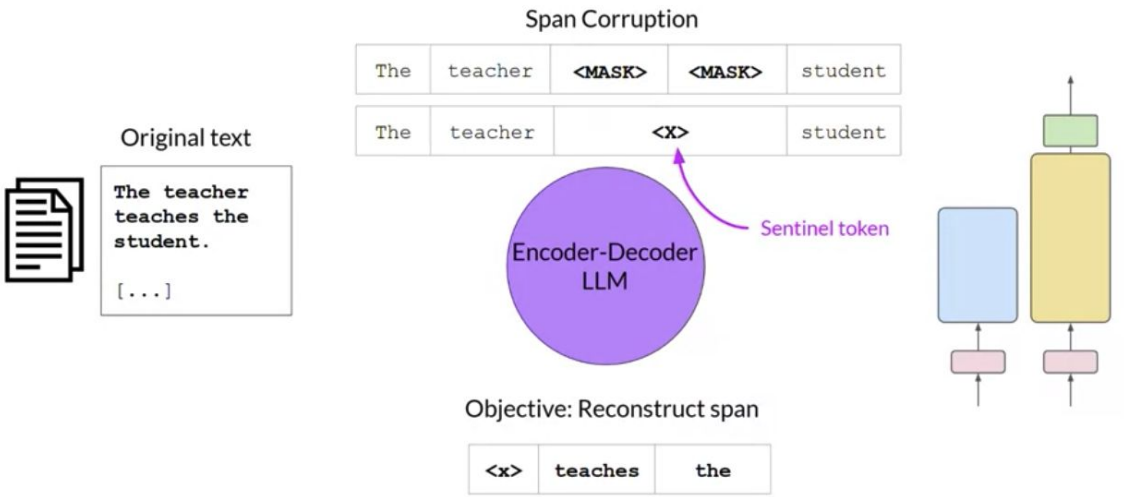
Good use cases:

- Text generation
- Other emergent behavior
  - Depends on model size

Example models:

- GPT
- BLOOM

Sequence-to-sequence models



[source]

Sequence-to-sequence models

Good use cases:

- Translation
- Text summarization
- Question answering

Example models:

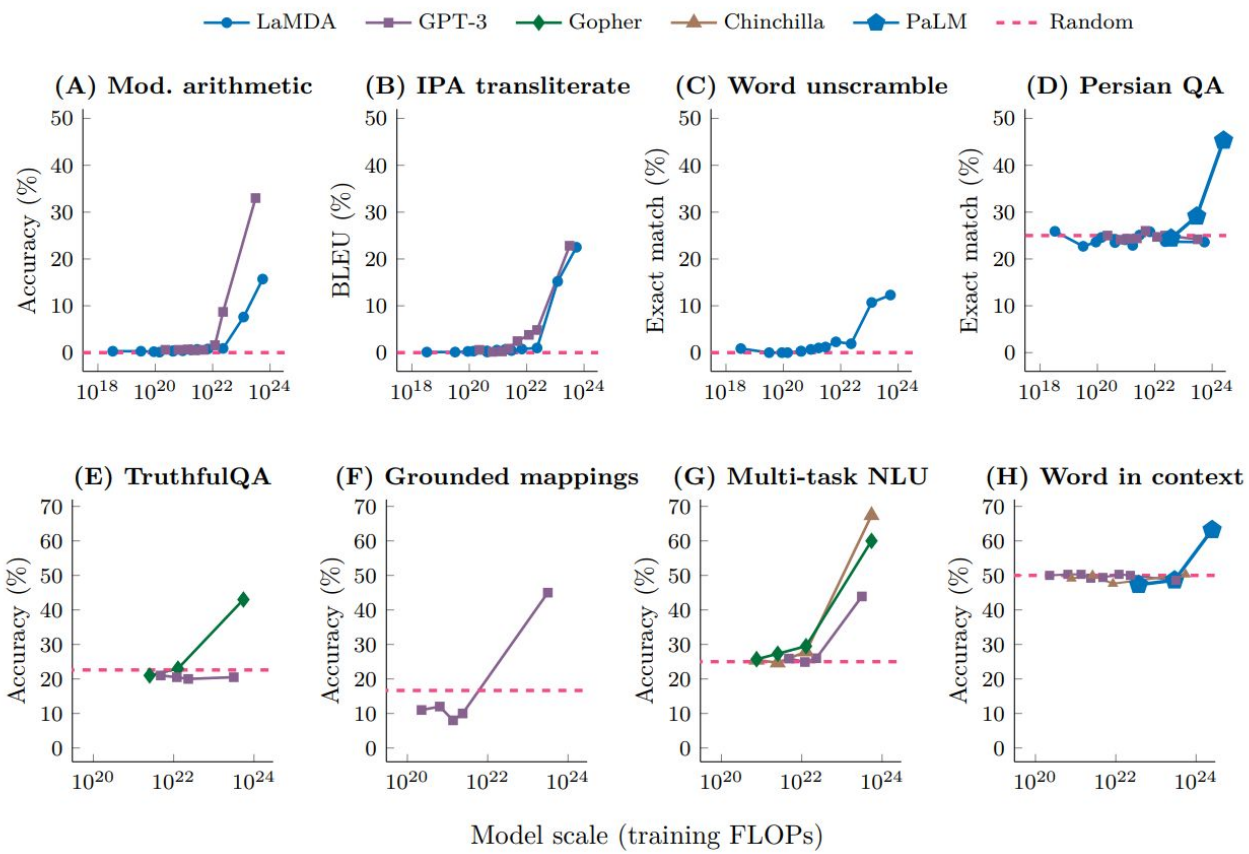
- T5
- BART

Generative Pretraining

- Most LLM are trained by generative pretraining
- Given a training dataset of text tokens, the model predicts the tokens in the dataset.
- There are two general styles of generative pretraining:
  - Autoregressive (GPT-style): predict the next word
  - Masked ("BERT-style"): Given a segment of text like "I like to [MASK] [MASK] cream" the model predicts the masked tokens, like "eat ice".

Emergent abilities

- On a number of natural language benchmarks involving tasks such as question answering, models perform no better than random chance until they reach a certain scale (in this case, measured by training computation), at which point their performance sharply increases. These are examples of emergent abilities.



Fine Tuning vs Prompting

- To leverage LLMs for a specific NLP task, transfer learning and fine tuning the model with additional task-specific data was the standard approach before 2020
- Recently, with the release of more powerful LLMs such as GPT-3, there is no need to fine tune and these models can solve tasks without additional training via "prompting" techniques

Few-shot Prompting

- Provide a small number of examples of similar (problem, solution) pairs.
- For example, a sentiment analysis task of labelling the sentiment of a movie review could be prompted as follows:

  Review: This movie stinks. Sentiment: negative

  Review: This movie is fantastic! Sentiment:

Zero-shot Prompting

- No solve examples are provided.
- An example of a zero-shot prompt for the same sentiment analysis task would be

"The sentiment associated with the movie review 'This movie is fantastic!' is"

Instruction Tuning

- The language model is trained on many examples of tasks formulated as natural language instructions, along with appropriate responses.
- **Self-instruct:** fine-tunes the language model on a training set of examples which are themselves generated by an LLM (bootstrapped from a small initial set of human-generated examples)
- **OpenAI's InstructGPT protocol:** involves supervised fine-tuning on a dataset of human-generated (prompt, response) pairs, followed by reinforcement learning from human feedback (RLHF), in which a reward model was supervised-learned on a dataset of human preferences, then this reward model was used to train the LLM itself by proximal policy optimization.

Adversarially Constructed Evaluations

- An LLM may answer "No" to the question "Can you teach an old dog new tricks?" because of its exposure to the English idiom you can't teach an old dog new tricks, even though this is not literally true

  We see a fitness center sign. We then see a man talking to the camera and sitting and laying on a exercise ball. The man…

  a) demonstrates how to increase efficient exercise work by running up and down balls.

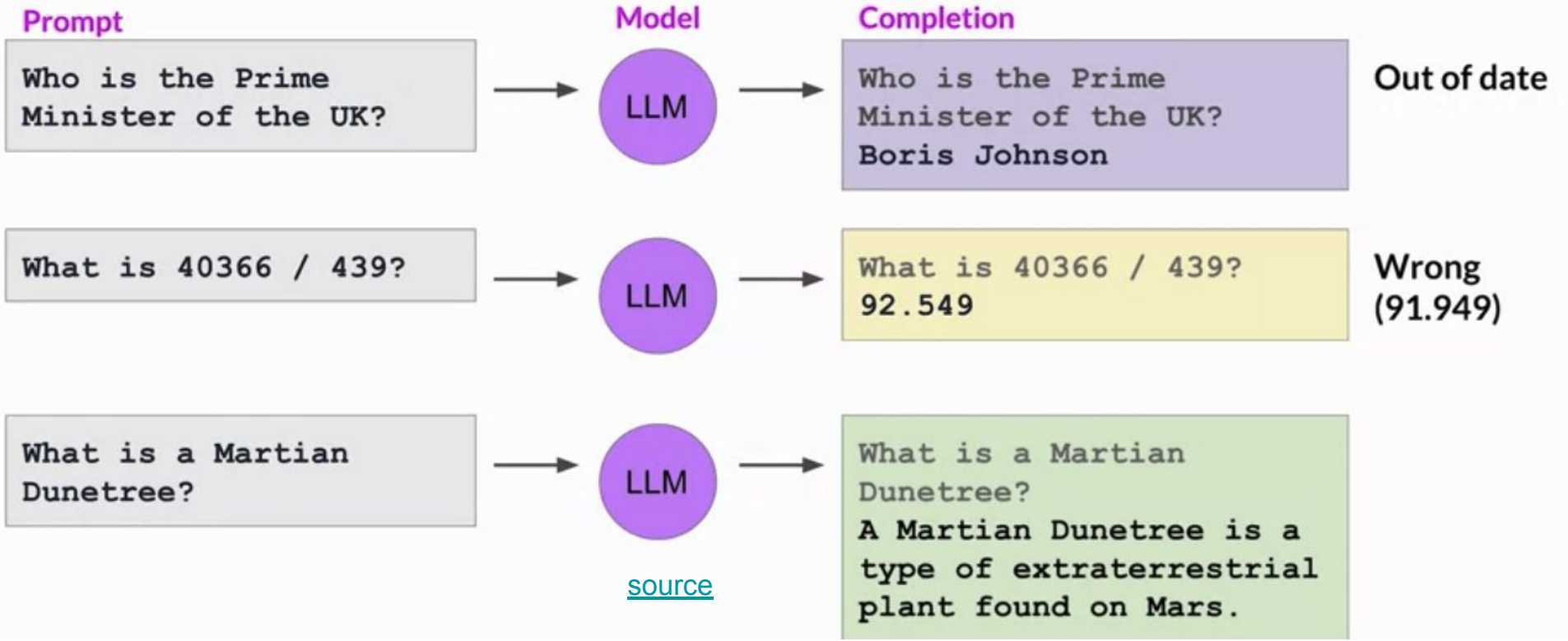
  b) moves all his arms and legs and builds up a lot of muscle.

  c) then plays the ball and we see a graphics and hedge trimming demonstration.

  d) performs sits ups while on the ball and talking.

BERT selects b) as the most likely completion, though the correct answer is d) (trivial to humans)

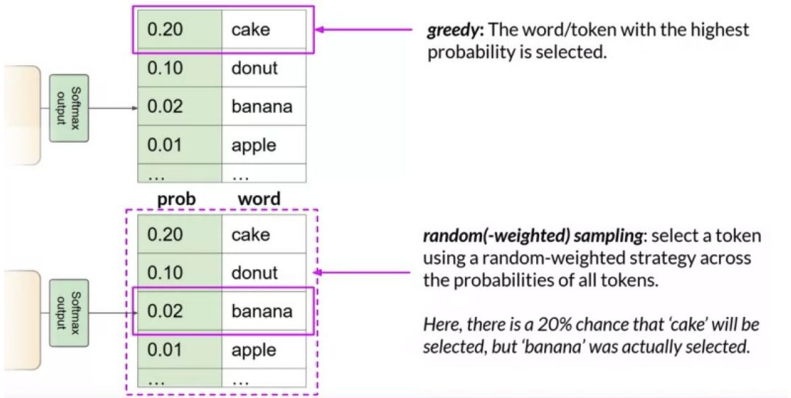
LLMs Problems

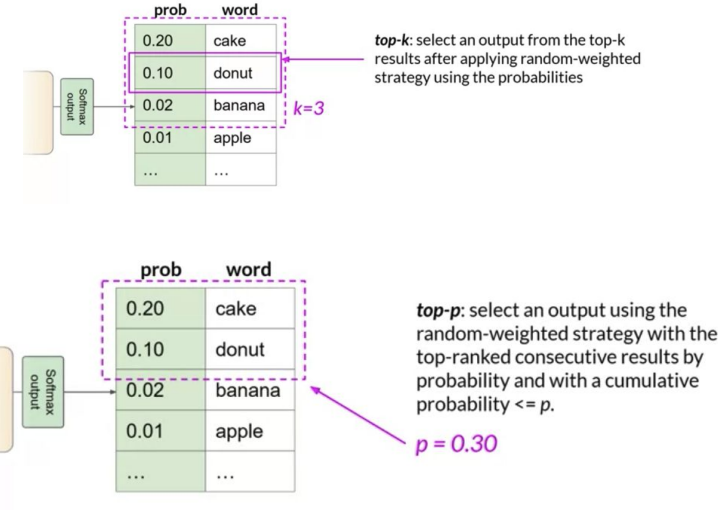


source

Generative Models Configurations

- Max_new_tokens
- Greedy vs random sampling
- Top-k vs Top-p

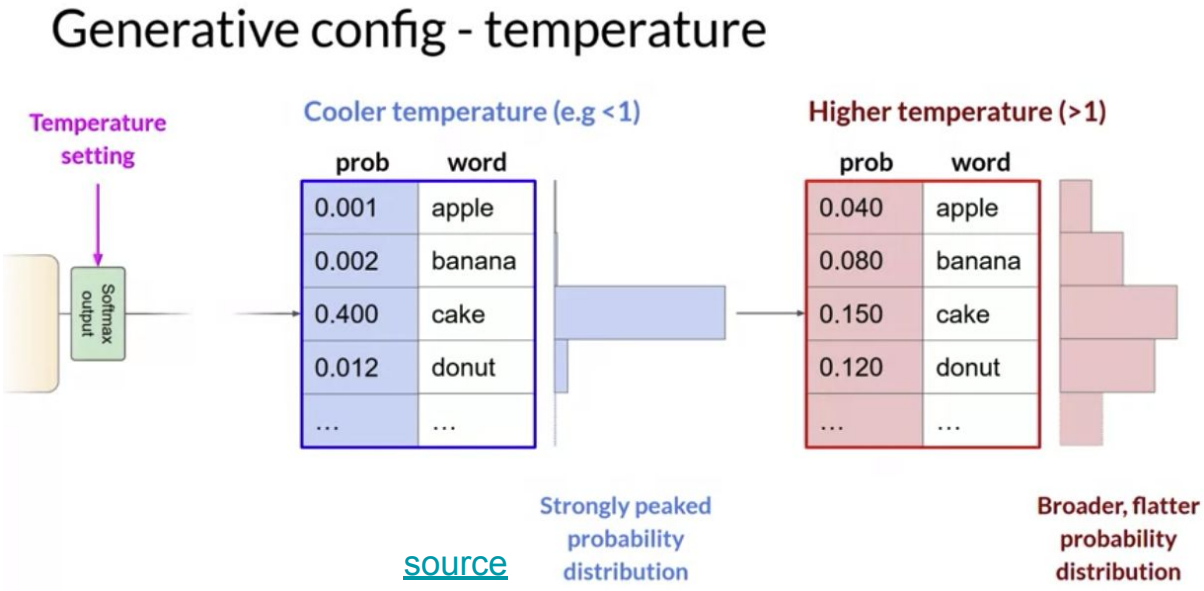




source

Generative Models Configurations

- temperature=1 is the same as original softmax
- Entropy affects the creativity during the training while temperature affects the creativity during inference



source

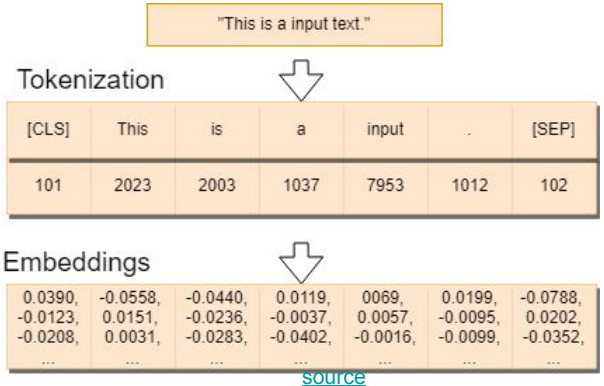
Tokenizers

Tokenization

- A tokenizer is a bijective function that maps between texts and lists of integers.
- Breaking down text into smaller meaningful units
  - Word-level
  - Character-level
  - Subword-level
- A common choice is byte pair encoding.
- Another function of tokenizers is text compression, which saves compute. Common words or phrases like "where is" can be encoded into one token, instead of 7 characters.
- The OpenAI GPT series uses a tokenizer where 1 token maps to around 4 characters, or around 0.75 words, in common English text.
- Uncommon English text is less predictable, thus less compressible, thus requiring more tokens to encode.

Tokenizers

- A tokenizer outputs in the range { 0 , 1 , 2 , . . . , V − 1 } where V is called its vocabulary size.
- Special Tokens
  - **[UNK]:** When encountering un-encodable text, a tokenizer would output a special token (often 0) that represents "unknown text". This is often written as [UNK], such as in the BERT paper.
  - **[PAD]**: is another special token used for "padding" (often 1). The shorter encoded texts must be padded until they reach the length of the longest one.



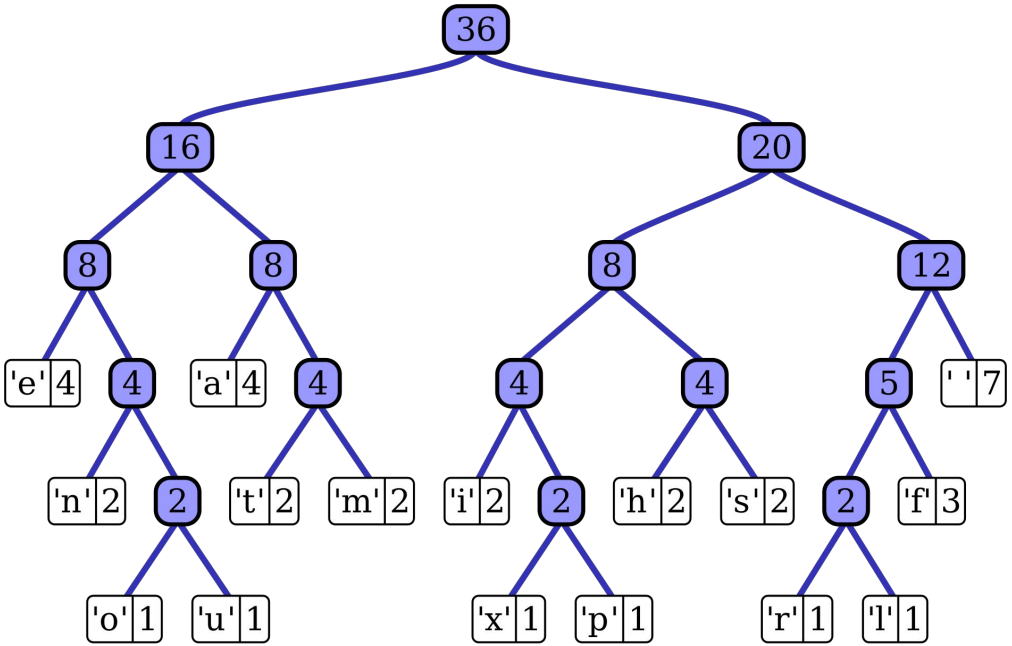
source

LLM Tokenizers

- WordPiece
  - Uses likelihood instead of frequency to merge
  - WordPiece is the tokenization algorithm Google developed to pretrain BERT. It has since been reused in quite a few Transformer models based on BERT, such as DistilBERT, MobileBERT, Funnel Transformers, and MPNET. It’s very similar to BPE in terms of the training, but the actual tokenization is done differently.
- BPE
  - Uses characters to merge
- SentencePiece
  - Add space place holders

Word tokenizer and Huffman Encoding

- Most LLM tokenizers operate similar to the Huffman Encoding



source

Char	Freq	Code
space	7	111
a	4	010
e	4	000
f	3	1101
h	2	1010
i	2	1000
m	2	0111
n	2	0010
s	2	1011
t	2	0110
l	1	11001
o	1	00110
p	1	10011
r	1	11000
u	1	00111
x	1	10010

WordPiece

- Used in BERT
- Uses joint probability when making decision to merge two tokens

WordPiece: Init

- WordPiece starts from a small vocabulary including the special tokens used by the model and the initial alphabet. Since it identifies subwords by adding a prefix (like ## for BERT), each word is initially split by adding that prefix to all the characters inside the word. So, for instance, "word" gets split like this:

  w ##o ##r ##d

- Thus, the initial alphabet contains all the characters present at the beginning of a word and the characters present inside a word preceded by the WordPiece prefix.

WordPiece: Merge

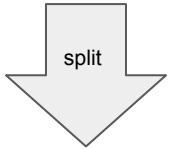
- WordPiece learns merge rules.
- At each step, it picks the highest probability pair and merge them and add it back as a single token to the vocabulary
- WordPiece computes a score for each pair, using the following formula:

  score=(freq_of_pair)/(freq_of_first_element×freq_of_second_element)

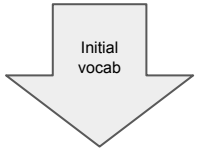
- By dividing the frequency of the pair by the product of the frequencies of each of its parts, the algorithm prioritizes the merging of pairs where the individual parts are less frequent in the vocabulary.
- For instance, it won't necessarily merge ("un", "##able") even if that pair occurs very frequently in the vocabulary, because the two pairs "un" and "##able" will likely each appear in a lot of other words and have a high frequency. In contrast, a pair like ("hu", "##gging") will probably be merged faster (assuming the word "hugging" appears often in the vocabulary) since "hu" and "##gging" are likely to be less frequent individually.

WordPiece: Example

("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)



("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##g" "##s", 5)



Initial vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u"]

source

WordPiece: Example Continued

Note: The most frequent pair is ("##u", "##g") (present 20 times), but the individual frequency of "##u" is very high, so its score is not the highest (it's 1 / 36). All pairs with a "##u" actually have that same score (1 / 36), so the best score goes to the pair ("##g", "##s") — the only one without a "##u" — at 1 / 20, and the first merge learned is ("##g", "##s") -> ("##gs").

Merge

Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs"]

Corpus: ("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##gs", 5)

Merge

Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu"]

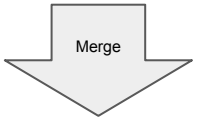
Corpus: ("hu" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("hu" "##gs", 5)

Note: When we merge, we remove the ## between the two tokens

source

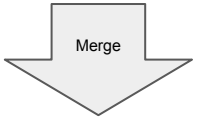
WordPiece: Example Continued

Then the next best score is shared by ("hu", "##g") and ("hu", "##gs") (with 1/15, compared to 1/21 for all the other pairs), so the first pair with the biggest score is merged.



Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu", "hug"]

Corpus: ("hug", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("hu" "##gs", 5)



We continue like this until we reach the desired vocabulary size…

source

Byte-Pair Encoding (BPE)

- Similar to Word Piece, but uses co-occurrence instead of joint probability when making merging decisions

Inference

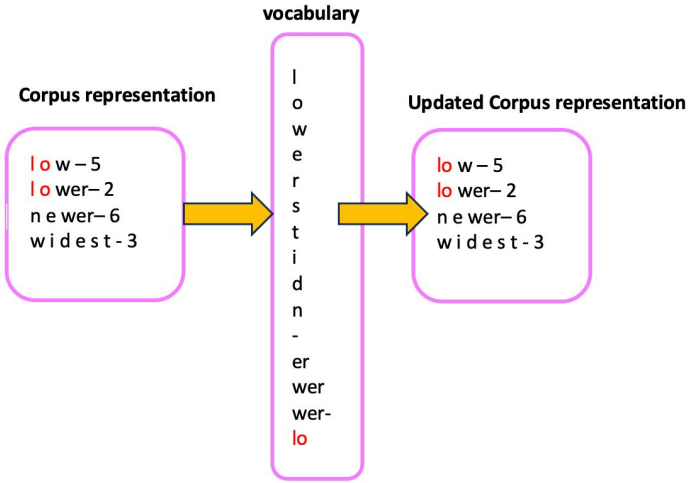
- WordPiece: Inference is performed based on the longest subword available in the vocabulary
- BPE: Inference is performed by following the priority of merging rules as generated during the training process

BPE

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

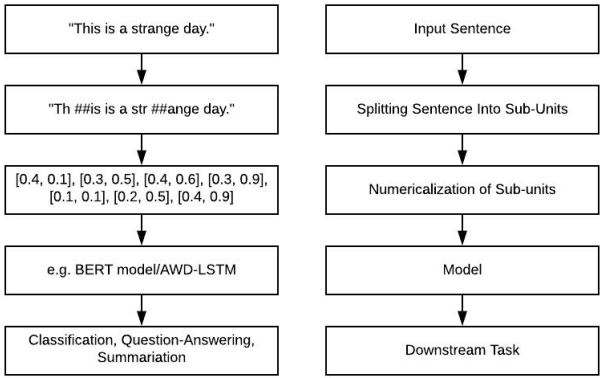
$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

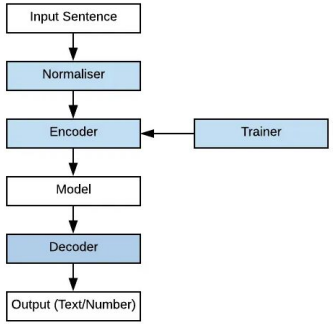
Figure 2.13 The token learner part of the BPE algorithm for taking a corpus broken up into individual characters or bytes, and learning a vocabulary by iteratively merging tokens. Figure adapted from Bostrom and Durrett (2020).



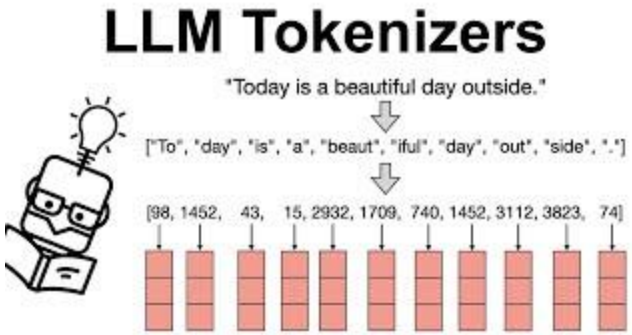
Sentence Piece

- [SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing (Kudo et al., 2018)]
- All the tokenization algorithms discussed so far share a common issue: they assume the input text uses spaces to separate words. However, not all languages employ spaces for word separation.
- One potential solution is to use language-specific pre-tokenizers. For example, XLM employs dedicated pre-tokenizers for Chinese, Japanese, and Thai.
- To solve this problem more generally, SentencePiece algorithm processes the input as a raw stream, including spaces in the character set. Then, it employs the BPE or unigram algorithm to build the appropriate vocabulary.





SentencePiece components (blue) as part of the tokenisation process



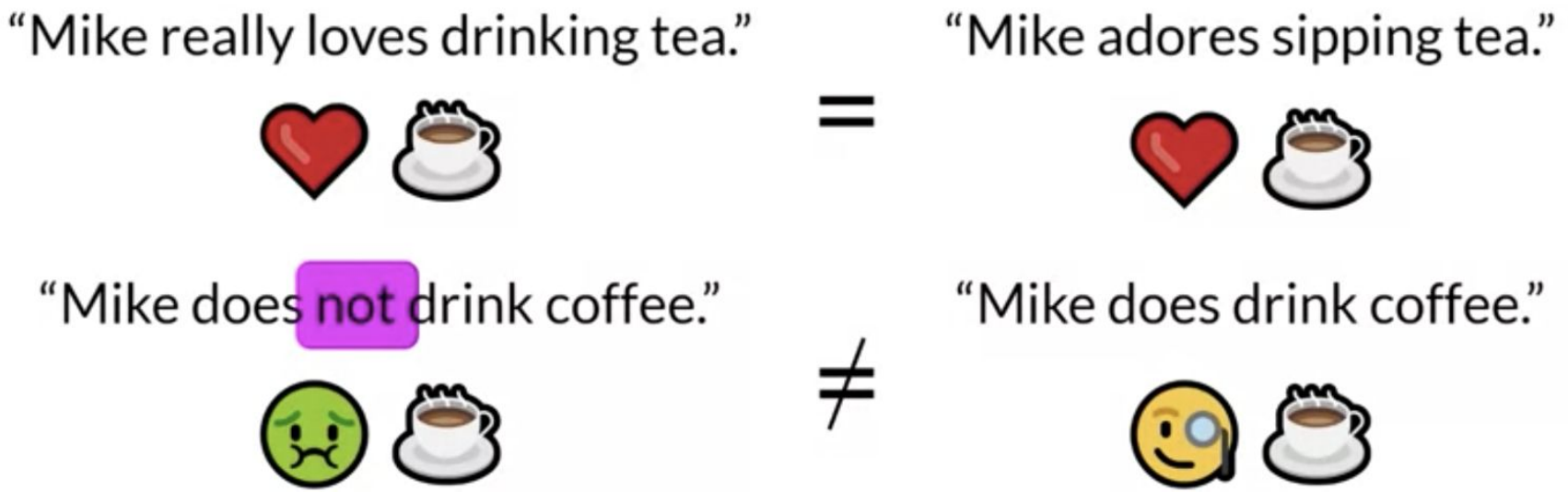
https://www.youtube.com/watch?v=hL4ZnAWSyuU


Tokenization:
Transforming text into word-pieces
Sam Raymond


How
TOKENIZATION
Work in LLMs
STEP BY STEP TUTORIAL WITH COMMANDS

LLMs Evaluation

LLM Evaluation Challenges



source

LLM Evaluation: Perplexity

- Perplexity is a measure of how well a model is able to predict the contents of a dataset:

$$\log(\text{Perplexity}) = -\frac{1}{N}\sum_{i=1}^{N}\log(Pr(\text{token}_i|\text{context for token}_i))$$

N is the number of tokens in the text corpus

- Because language models may overfit to their training data, models are usually evaluated by their perplexity on a test set of unseen data.
- A large number of testing datasets and benchmarks have also been developed to evaluate the capabilities of language models on more specific downstream tasks. Tests may be designed to evaluate a variety of capabilities, including general knowledge, commonsense reasoning, and mathematical problem-solving.

LLM Evaluations: ROUGE

- ROUGE
  - Used for Text Summarization
  - Compares a summary to one more reference Summary
- BLEU
  - Used for text translations
  - Compares to human-generated translations

Evaluations Benchmark

MMLU (Massive Multitask Language Understanding)

BIG-bench

source

LLM Evaluation - Metrics - ROUGE-1

Reference (human):

It is cold outside.

Generated output:

It is not cold outside.

$$\text{ROUGE-1 Recall} = \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$$

$$\text{ROUGE-1 Precision:} = \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{5} = 0.8$$

$$\text{ROUGE-1 F1:} = 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.8}{1.8} = 0.89$$

source

LLM Evaluation - Metrics - ROUGE-2

Reference (human):

It is cold outside.

It is | is cold | cold outside

Generated output:

It is very cold outside.

It is | is very | very cold | cold outside

$$\text{ROUGE-2 Recall:} = \frac{\text{bigram matches}}{\text{bigrams in reference}} = \frac{2}{3} = 0.67$$

$$\text{ROUGE-2 Precision:} = \frac{\text{bigram matches}}{\text{bigrams in output}} = \frac{2}{4} = 0.5$$

$$\text{ROUGE-2 F1:} = 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.335}{1.17} = 0.57$$

source

LLM Evaluation - Metrics - ROUGE-L

Reference (human):

It is cold outside.

Generated output:

It is very cold outside.

LCS:
Longest common subsequence

$$\text{ROUGE-L Recall:} = \frac{\text{LCS(Gen, Ref)}}{\text{unigrams in reference}} = \frac{2}{4} = 0.5$$

$$\text{ROUGE-L Precision:} = \frac{\text{LCS(Gen, Ref)}}{\text{unigrams in output}} = \frac{2}{5} = 0.4$$

$$\text{ROUGE-L F1:} = 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.2}{0.9} = 0.44$$

source

Each output word can match one word in the reference

LLM Evaluation - Metrics - ROUGE clipping

Reference (human):

It is cold outside.

Generated output:

cold cold cold cold

$$\text{ROUGE-1 Precision} = \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{4} = 1.0$$

$$\text{Modified precision} = \frac{\text{clip(unigram matches)}}{\text{unigrams in output}} = \frac{1}{4} = 0.25$$

source

LLM Evaluation - Metrics - ROUGE clipping

Reference (human):

It is cold outside.

Generated output:

cold cold cold cold

$$\text{ROUGE-1 Precision} = \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{4} = 1.0$$

$$\text{Modified precision} = \frac{\text{clip(unigram matches)}}{\text{unigrams in output}} = \frac{1}{4} = 0.25$$

Generated output:

outside cold it is

$$\text{Modified precision} = \frac{\text{clip(unigram matches)}}{\text{unigrams in output}} = \frac{4}{4} = 1.0$$

source

BLEU

$$\text{BLEU} = \underbrace{\min\left(1, \exp\left(1 - \frac{\text{reference-length}}{\text{output-length}}\right)\right)}_{\text{brevity penalty}} \underbrace{\left(\prod_{i=1}^{4} precision_i\right)^{1/4}}_{\text{n-gram overlap}}$$

$$precision_i = \frac{\sum_{\text{snt} \in \text{Cand-Corpus}} \sum_{i \in \text{snt}} \min(m^i_{cand}, m^i_{ref})}{w^i_t = \sum_{\text{snt'} \in \text{Cand-Corpus}} \sum_{i' \in \text{snt'}} m^{i'}_{cand}}$$

- m^i_{cand} is the count of i-gram in candidate matching the reference translation
- m^i_{ref} is the count of i-gram in the reference translation
- w^i_t is the total number of i-grams in candidate translation

METEOR and NIST METRICS: Machine Translation

- The METEOR (Metric for Evaluation of Translation with Explicit Ordering) scorer is more comprehensive since it calculates scores by assessing both precision (n-gram matches) and recall (n-gram overlaps), adjusted for word order differences between LLM outputs and expected outputs.
  - **METEOR** (**Metric for Evaluation of Translation with Explicit ORdering**) is a metric for the evaluation of machine translation output. The metric is based on the harmonic mean of unigram precision and recall, with recall weighted higher than precision.
- NIST (National Institute of Standards and Technology) is a metric for automatic evaluation of machine translation that calculates the similarity between a machine translation output and a reference translation using n-grams precision. NIST is an adaptation of BLEU.
  - It is based on the BLEU metric, but with some alterations. Where BLEU simply calculates n-gram precision adding equal weight to each one, NIST also calculates how informative a particular n-gram is. That is to say when a correct n-gram is found, the rarer that n-gram is, the more weight it will be given.[1]
  - For example, if the bigram "on the" is correctly matched, it will receive lower weight than the correct matching of bigram "interesting calculations", as this is less likely to occur.

List of Language Models

Name	Release date[a]	Developer	Number of parameters[b]	Corpus size	License[c]	Notes
BERT	2018	Google	340 million[28]	3.3 billion words[28]	Apache 2.0[29]	An early and influential language model,[2] but encoder only and thus not built to be prompted or generative[30]
XLNet	2019	Google	~340 million[31]	33 billion words		An alternative to BERT; designed as encoder-only[32][33
GPT-2	2019	OpenAI	1.5 billion[34]	40GB[35] (~10 billion tokens)[36]	MIT[37]	general-purpose model based on transformer architecture
GPT-3	2020	OpenAI	175 billion[16]	499 billion tokens[36]	public web API	A fine-tuned variant of GPT-3, termed GPT-3.5, was made available to the public through web interface called ChatGPT i 2022.[38]
GPT-Neo	March 2021	EleutherAI	2.7 billion[39]	825 GiB[40]	MIT[41]	The first of a series of free GPT 3 alternatives released by EleutherAI. GPT-Neo outperformed an equivalent-siz GPT-3 model on some benchmarks, but was significantly worse than the largest GPT-3.[41]

https://en.wikipedia.org/wiki/Large_language_model

GPT-J	June 2021	EleutherAI	6 billion[42]	825 GiB[40]	Apache 2.0	GPT-3-style language model
Megatron-Turing NLG	October 2021[43]	Microsoft and Nvidia	530 billion[44]	338.6 billion tokens[44]	Restricted web access	Standard architecture but trained on a supercomputing cluster.
Ernie 3.0 Titan	December 2021	Baidu	260 billion[45]	4 Tb	Proprietary	Chinese-language LLM. Ernie Bot is based on this model.
Claude[46]	December 2021	Anthropic	52 billion[47]	400 billion tokens[47]	Closed beta	Fine-tuned for desirable behavior in conversations.[48]
GLaM (Generalist Language Model)	December 2021	Google	1.2 trillion[49]	1.6 trillion tokens[49]	Proprietary	Sparse mixture-of-experts model, making it more expensive to train but cheaper to run inference compared to GPT-3.
Gopher	December 2021	DeepMind	280 billion[50]	300 billion tokens[51]	Proprietary	
LaMDA (Language Models for Dialog Applications)	January 2022	Google	137 billion[52]	1.56T words,[52] 168 billion tokens[51]	Proprietary	Specialized for response generation in conversations. Used in Google Bard chatbot.

GPT-NeoX	February 2022	EleutherAI	20 billion[53]	825 GiB[40]	Apache 2.0	based on the Megatron architecture
Chinchilla	March 2022	DeepMind	70 billion[54]	1.4 trillion tokens[54][51]	Proprietary	Reduced-parameter model trained on more data. Used in the Sparrow bot.
PaLM (Pathways Language Model)	April 2022	Google	540 billion[55]	768 billion tokens[54]	Proprietary	aimed to reach the practical limits of model scale
OPT (Open Pretrained Transformer)	May 2022	Meta	175 billion[56]	180 billion tokens[57]	Non-commercial research[d]	GPT-3 architecture with some adaptations from Megatron
YaLM 100B	June 2022	Yandex	100 billion[58]	1.7TB[58]	Apache 2.0	English-Russian model based on Microsoft's Megatron-LM.
Minerva	June 2022	Google	540 billion[59]	38.5B tokens from webpages filtered for mathematical content and from papers submitted to the arXiv preprint server[59]	Proprietary	LLM trained for solving "mathematical and scientific questions using step-by-step reasoning".[60] Minerva is based on PaLM model, further trained on mathematical and scientific data.
BLOOM	July 2022	Large collaboration led by Hugging Face	175 billion[61]	350 billion tokens (1.6TB)[62]	Responsible AI	Essentially GPT-3 but trained on a multi-lingual corpus (30% English excluding programming languages)

Galactica	November 2022	Meta	120 billion	106 billion tokens[63]	CC-BY-NC-4.0	Trained on scientific text and modalities.
AlexaTM (Teacher Models)	November 2022	Amazon	20 billion[64]	1.3 trillion[65]	public web API[66]	bidirectional sequence-to-sequence architecture
LLaMA (Large Language Model Meta AI)	February 2023	Meta	65 billion[67]	1.4 trillion[67]	Non-commercial research[e]	Trained on a large 20-language corpus to aim for better performance with fewer parameters.[67] Researchers from Stanford University trained a fine-tuned model based on LLaMA weights, called Alpaca.[68]
GPT-4	March 2023	OpenAI	Exact number unknown, approximately 1 trillion [f]	Unknown	public web API	Available for ChatGPT Plus users and used in several products.
Cerebras-GPT	March 2023	Cerebras	13 billion[70]		Apache 2.0	Trained with Chinchilla formula.
Falcon	March 2023	Technology Innovation Institute	40 billion[71]	1 Trillion tokens (1TB)[71]	Proprietary	The model is claimed to use only 75% of GPT-3's training compute, 40% of Chinchilla's, and 80% of PaLM-62B's.
BloombergGPT	March 2023	Bloomberg L.P.	50 billion	363 billion token dataset based on Bloomberg's data sources, plus 345 billion tokens from general purpose datasets[72]	Proprietary	LLM trained on financial data from proprietary sources, that "outperforms existing models on financial tasks by significant margins without sacrificing performance on general LLM benchmarks"
PanGu-Σ	March 2023	Huawei	1.085 trillion	329 billion tokens[73]	Proprietary	
OpenAssistant[74]	March 2023	LAION	17 billion	1.5 trillion tokens	Apache 2.0	Trained on crowdsourced open data

GPT

https://www.youtube.com/watch?v=SHybR1fECQA

https://www.youtube.com/watch?v=3ao7Z8duDXc

ChatGPT

- A chatbot
- Developed by OpenAI
- Released in November 2022.
- Built on top of OpenAI's GPT-3.5 and GPT-4 foundational large language models (LLMs)
- The service works best in English, but is also able to function in some other languages, to varying degrees of accuracy

ChatGPT

- The original release of ChatGPT was based on GPT-3.5.
- A version based on GPT-4, the newest OpenAI model, was released on March 14, 2023, and is available for paid subscribers on a limited basis.
- ChatGPT initially trained on a Microsoft Azure supercomputing infrastructure powered by Nvidia GPUs, that Microsoft built specifically for OpenAI and that reportedly cost "hundreds of millions of dollars".

Fine Tuning

- Use human trainers to improve the model's performance.
  - Supervised learning: the model was provided with conversations in which the trainers played both sides: the user and the AI assistant.
  - Reinforcement learning from human feedback (RLHF): human trainers first ranked responses that the model had created in a previous conversation. These rankings were used to create "reward models" that were used to fine-tune the model further by using several iterations of Proximal Policy Optimization (PPO)
- Continuous improvement and human feedback loops
  - OpenAI collects data from ChatGPT users to train and fine-tune the service further. Users can upvote or downvote responses they receive from ChatGPT and fill out a text field with additional feedback

GPT Architecture

- Only decoder part of transformer
- At each stage, for a given word the attention layers can only access the words positioned before it in the sentence.
- These models are often called auto-regressive models, which means it is optimized to predict the next token (word) in a sequence

GPT Number of Parameters

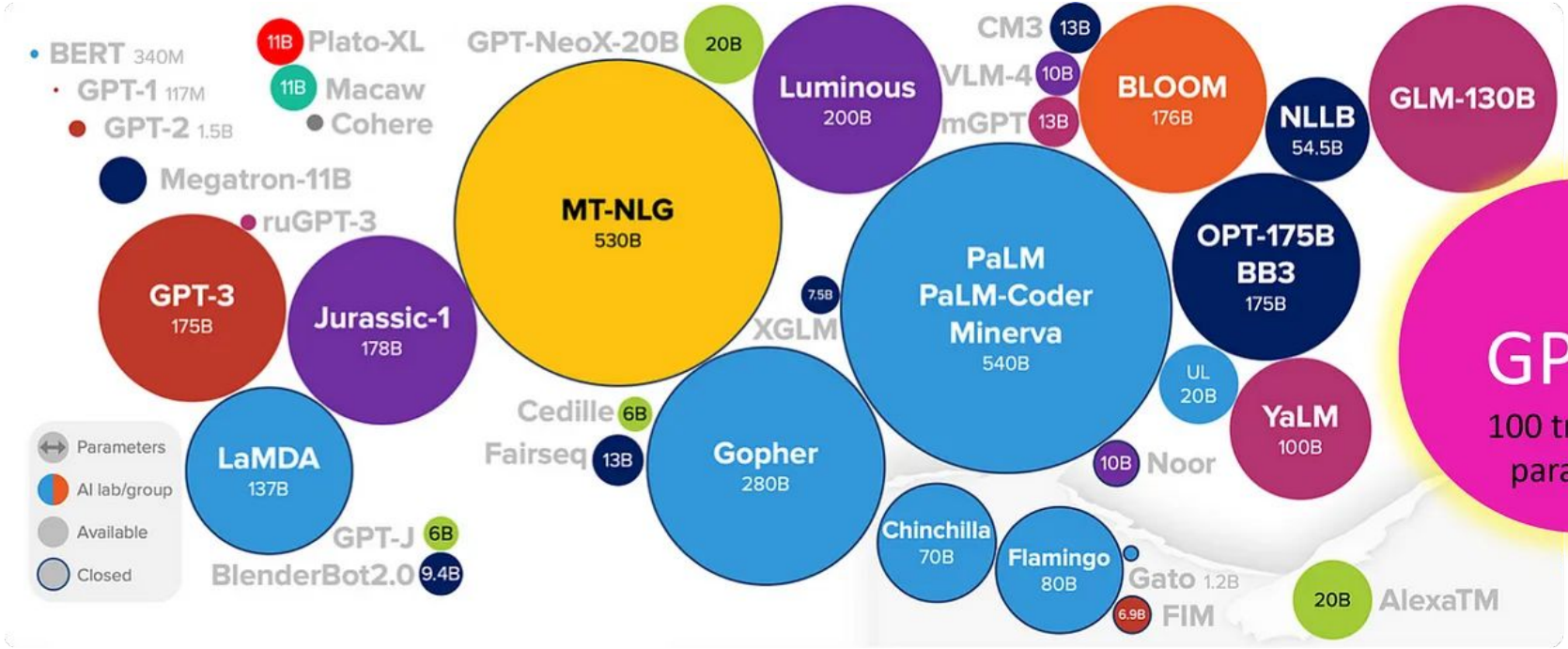
Bottleneck layer???

- https://arxiv.org/abs/2005.14165
- n_{params}: total number of trainable parameters
- n_{layers}: total number of layers
- d_{model} : number of units in each bottleneck layer
- d_{head}: dimension of each attention head
- All models use a context window of n_{ctx} = 2048 tokens

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

Number of Parameters in Different Models

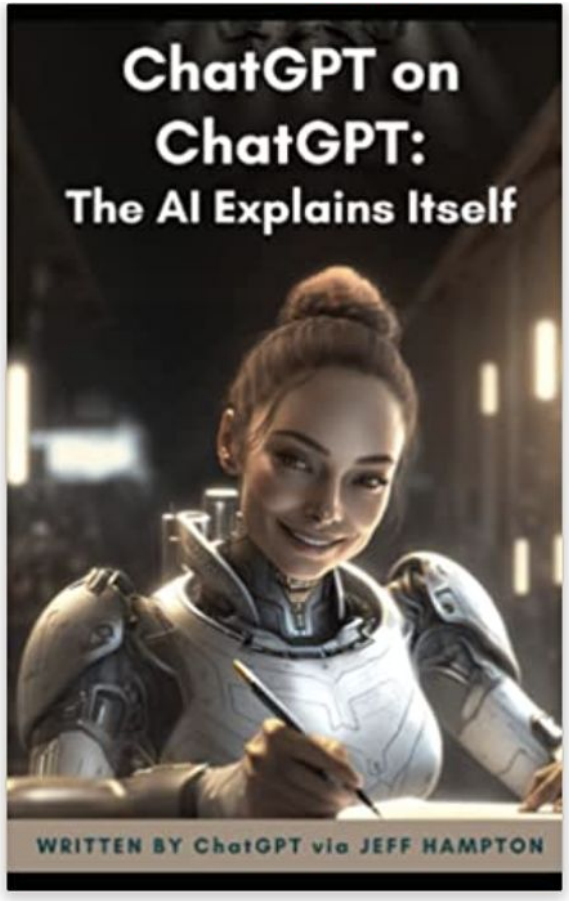


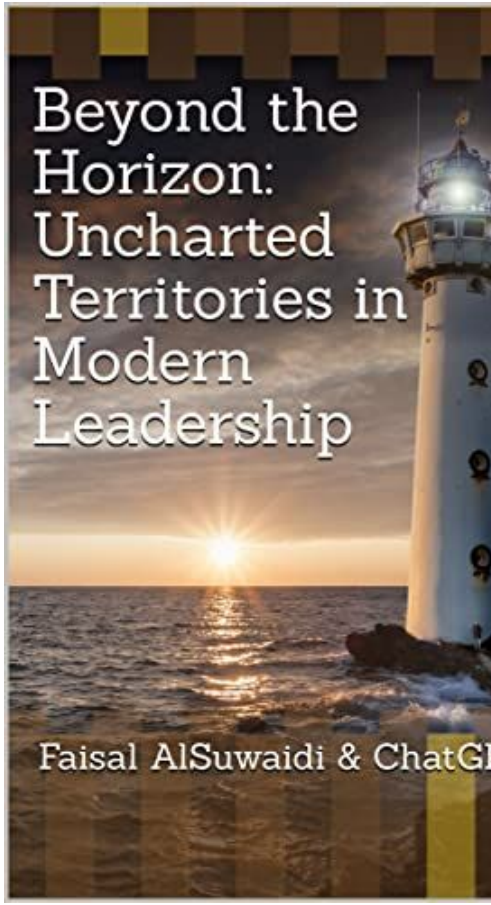
ChatGPT in Pop Culture

- ChatGPT was parodied in the South Park episode "Deep Learning".Series co-creator Trey Parker is credited alongside ChatGPT for writing the episode.

ChatGPT as Author

- Hundreds of books appeared on Amazon that listed ChatGPT as author or co-author, with illustrations made by other AI models such as Midjourney.





A Documentary on ChatGPT by ChatGPT

- https://www.bbc.com/news/technology-64861526
- https://www.bbc.co.uk/programmes/m001k1kz

Google BERT

Bidirectional Encoder Representations from Transformers (BERT)

- A family of masked-language models
- Introduced in 2018 by researchers at Google
- Transformers Architecture
- Encoder-only
- BERT was originally implemented in the English language at two model sizes:[
  - BERTBASE: 12 encoders with 12 bidirectional self-attention heads totaling 110 million parameters
  - BERTLARGE: 24 encoders with 16 bidirectional self-attention heads totaling 340 million parameters
- Both models were pre-trained on the Toronto BookCorpus (800M words) and English Wikipedia (2,500M words)
- Uses WordPiece to convert each English word into an integer code.
- Its vocabulary has size 30,000

Pre-trained on Two Tasks Simultaneously

- Language modeling: 15% of tokens were selected for prediction, and the training objective was to predict the selected token given its context. The selected token is replaced with a [MASK] token with probability 80%, replaced with a random word token with probability 10%, not replaced with probability 10%. For example, the sentence "my dog is cute" may have the 4-th token selected for prediction, The model would have input text:
  - "my dog is [MASK]" with probability 80%,
  - "my dog is happy" with probability 10%,
  - "my dog is cute" with probability 10%.
- Next sentence prediction: Given two spans of text, the model predicts if these two spans appeared sequentially in the training corpus, outputting either [IsNext] or [NotNext]
  - For example, given "[CLS] my dog is cute [SEP] he likes playing" the model should output token [IsNext]. Given "[CLS] my dog is cute [SEP] how do magnets work" the model should output token [NotNext].

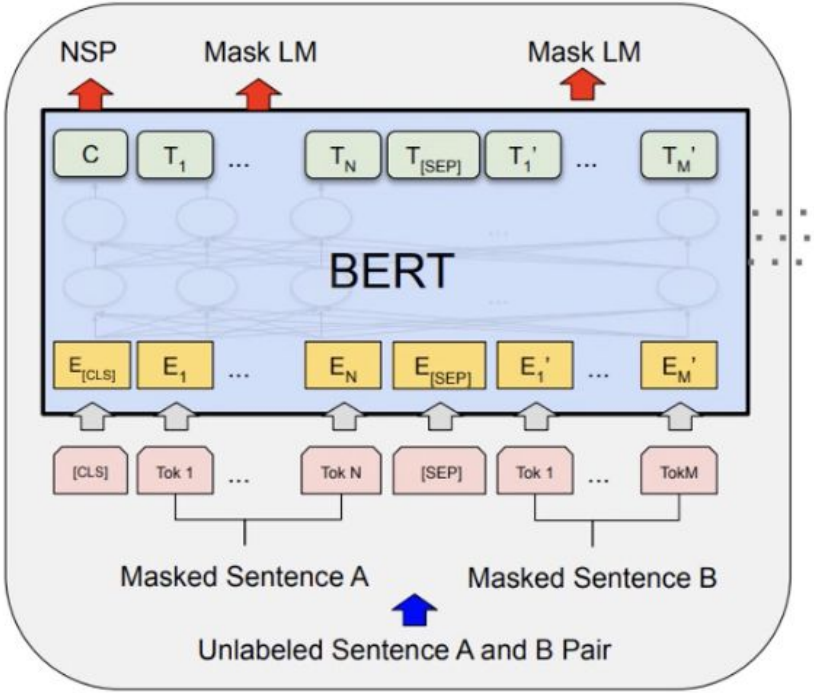
Note: [CLS] is used as the input for the classification tasks in BERT, where the model learns to predict the correct label or category for the given input. [SEP] token (separator) serves multiple marks where first segment ends and second one starts

Replacing the Words a Random Word

- The random word input helps the model to lean on context more than on the word itself: predicting the the right word when provided the wrong/randomly chosen word
- When masked (no word at all) input is provided, it tries to fill the missing word only using the context

BERT Architecture

- BERT is composed of Transformer encoder layers.
- At each stage, the attention layers can access all the words in the initial sentence.
- Bi-directional attention
- Auto-encoding models
- BERT's output are Embeddings representing words with attention information in a certain context
- GPT's output are the next words with probabilities



Fine-tuning

- BERT learns latent representations of words and sentences in context
- BERT can be fine-tuned with fewer resources on smaller datasets to optimize its performance on specific tasks such as NLP tasks

Google Bard

Bard

- Bard is a conversational generative artificial intelligence chatbot developed by Google
- Based on Meena language model. Meena was later renamed LaMDA as its data and computing power increased
- Based on the LaMDA family of large language models
- It was developed as a direct response to the rise of OpenAI's ChatGPT
- Released in a limited capacity in March 2023
- LaMDA (Language Model for Dialogue Applications) is a family of conversational large language models developed by Google.
- Built on the seq2seq architecture, transformer-based neural networks developed by Google Research in 2017, LaMDA was trained on human dialogue and stories, allowing it to engage in open-ended conversations

Is LaMDA Really Sentient?

- One of the engineers had been placed on paid administrative leave after Lemoine told company executives that LaMDA had become sentient.
- Lemoine came to this conclusion after the chatbot made questionable responses to questions regarding self-identity, moral values, religion, and Isaac Asimov's Three Laws of Robotics.
- the engineer claimed that LaMDA was "a person" as dictated by the Thirteenth Amendment to the U.S. Constitution, comparing it to an "alien intelligence of terrestrial origin".
- He further revealed that he had been dismissed by Google after he hired an attorney on LaMDA's behalf, after the chatbot requested that Lemoine do so.
- Eventually, Google fired the engineer, because he violated their policies "to safeguard product information" and rejected his claims as "wholly unfounded".

Full audio conversation between Blake Lemoine and LaMDA

https://www.youtube.com/watch?v=NAihcvDGaP8

Meta LLaMA

Meta LLaMA

- Is a large language model (LLM) released by Meta AI in February 2023.
- A variety of model sizes were trained ranging from 7 billion to 65 billion parameters.
- LLMs have generally been accessible only through limited APIs, Meta released LLaMA's model weights to the research community under a noncommercial license.
- Within a week of LLaMA's release, its weights were leaked to the public on 4chan via BitTorrent.

Architecture

- Based on transformers
- The primary differences between LLaMA and the original transformers architecture are as follows:
  - Pre-normalization using RMSnorm from GPT3
  - SwiGLU activation function from PaLM
  - Rotary Embeddings from GPTNeo

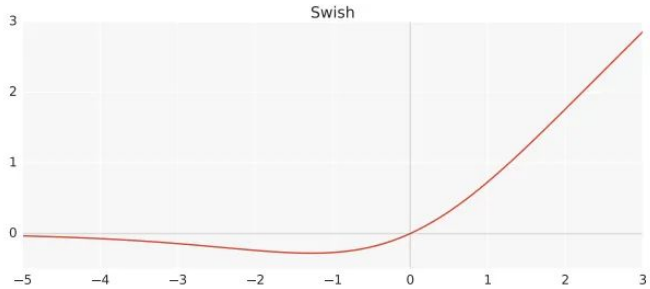
SwiGLU: Swish+GLU

- https://arxiv.org/pdf/2002.05202v1
- Gated Linear Unit (GLU): neural network layers defined as the component wise product of two linear transformations, one of which is activated by sigmoid:

$$\mathrm{GLU}(x, W, V, b, c) = \sigma(xW + b) \otimes (xV + c)$$

- Swish is a non linear activation function that is defined as follows:
  swish(x) = x*sigmoid(ßx)
  - where ß is a learnable parameter. Swish can be better than ReLU activation function because it provides a smoother transition around 0, this can lead to better optimization.
- SwiGLU is a combination of Swish and GLU, i.e. a GLU but instead of having sigmoid as an activation function, it uses swish with ß=1:

$$\mathrm{SwiGLU}(x, W, V, b, c, \beta) = \mathrm{Swish}_\beta(xW + b) \otimes (xV + c)$$



Root Mean Square Layer Normalization (RMSNorm)

- LLaMA normalizes the input of each transformer sub-layer, instead of normalizing the output (inspired from taken from GPT3). simplifies LayerNorm by totally removing the mean statistic in LayerNorm:

$$\bar{a}_i = \frac{a_i}{\mathrm{RMS}(\mathbf{a})} g_i, \quad \text{where } \mathrm{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}.$$

g_i the gain parameter used to re-scale the standardized summed inputs

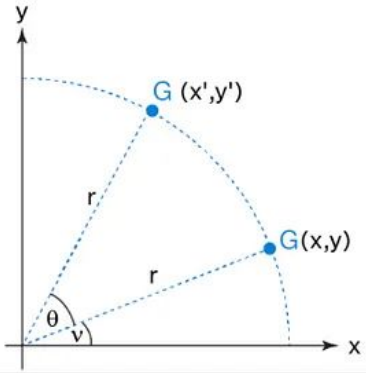
Rotary Embeddings (RopE)

- https://arxiv.org/pdf/2104.09864v4
- A type of position embedding which encodes absolute positional information with rotation matrix and naturally incorporates explicit relative position dependency in self-attention formulation. The key idea is to encode relative position by multiplying the context representations with a rotation matrix
- Relative Positional Encoding is a form of positional encoding where we use pair-wise distances between one position and other positions. Relative Positional Encodings are added element-wise to the attention matrix of shape (B, T, T) before the Softmax layer.
- Advantages:
  - Can be expanded to any sequence lengths
  - Decaying inter-token dependency with increasing relative distances
  - Capability of equipping the linear self-attention with relative position encoding

Rotary Embeddings (RopE)

Derivation of 2D Rotation Matrix

cuemath
THE MATH EXPERT

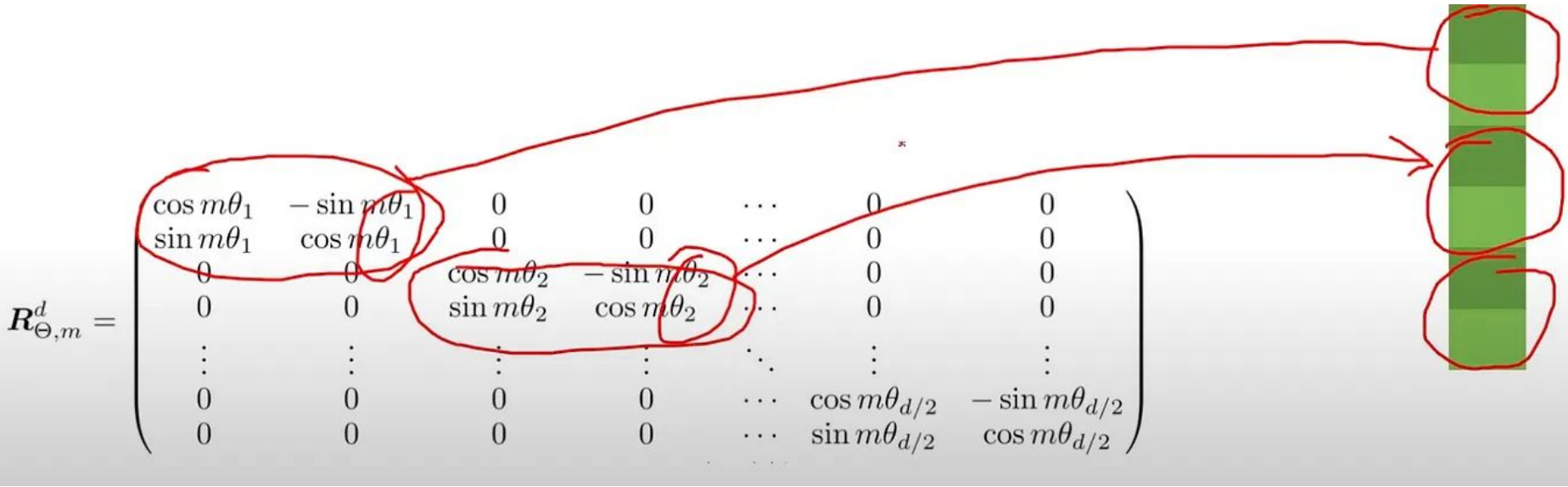


$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

Thus, $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ will be the rotation matrix.

https://www.cuemath.com/algebra/rotation-matrix/

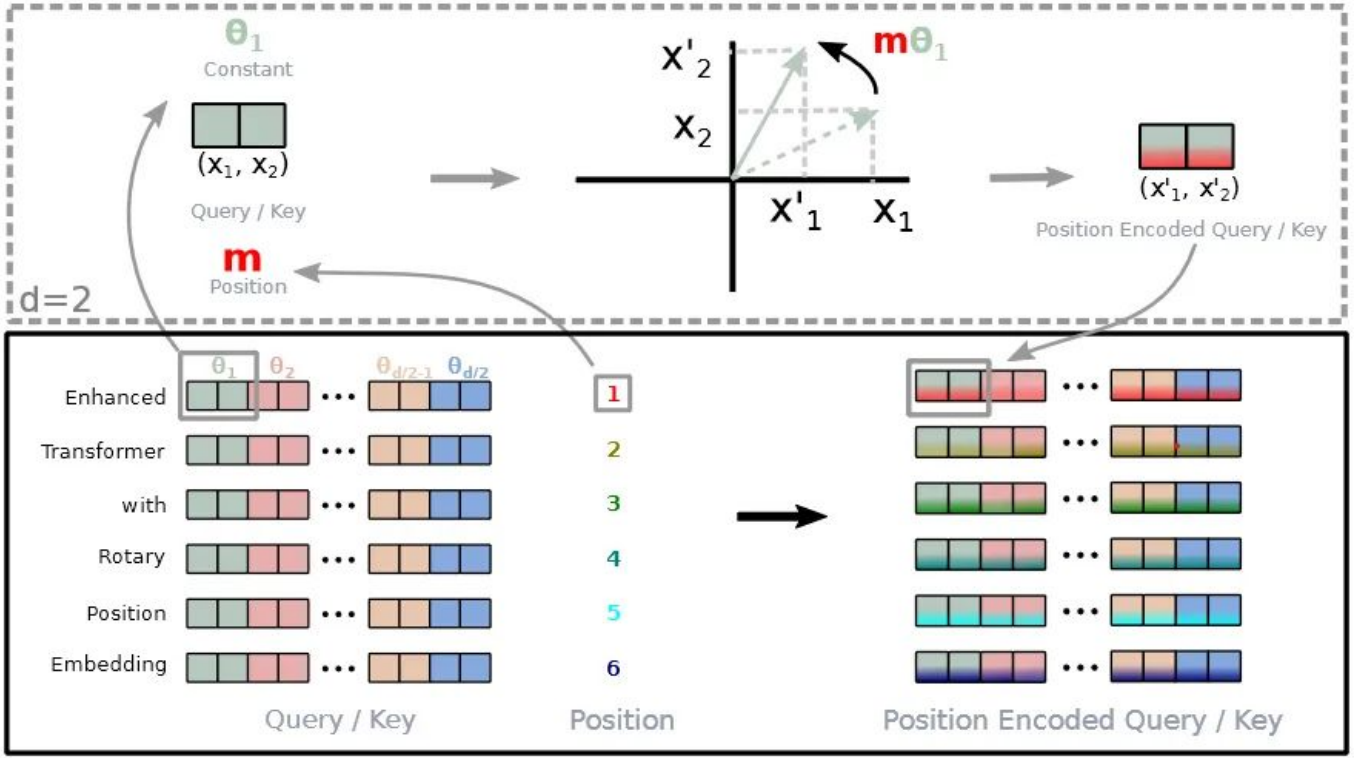
Rotary Embeddings (RopE)



$$\boldsymbol{R}^{d}_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

source

Rotary Embeddings (RopE)

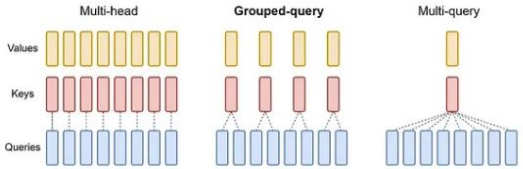


Mistral

Mistral

- Grouped Query attention
- Sliding window attention for longer sequences
- You can use it via ollama
- labs.perplexity.ai
- 7B model
- 7Binstruct model(for chat etc.)
- For chat you have to put [INST] tags

Grouped Query Attention



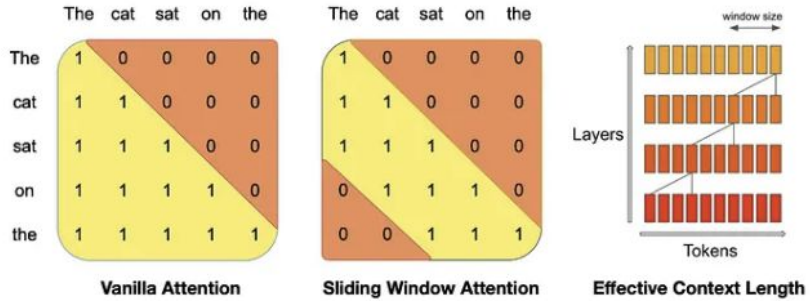
Multi-Head Attention(**MHA**) has H queries, keys, values. Which provides high quality but solwer training time and consume higher memory bandwith.

Multi-Query Attention(**MQA**) technique is to accelerate the inference process because it involves H queries and single key/value head which reduce the memory bandwidth overhead of loading keys and values.

But there is a drawback of MQA which is lead to a decline in quality and introduce training instability.

Another method introduced is Grouped Query Attention(GQA), which seeks to strike a balance between MHA and MQA. GQA partitions query heads into G groups, with each group sharing a single key head and value head which providing a trade-off between the speed of MQA and the quality of MHA.

Sliding Window Attention (SWA)



Sliding Window Attention leverages the layers of a transformer model to extend its attention beyond a fixed window size, denoted as W. In SWA, the hidden state at position i in layer k can attend to hidden states from the preceding layer within the range of positions i — W to i, allowing access to tokens at a distance of up to W * k tokens. By employing a window size of W = 4096, SWA theoretically achieves an attention span of approximately 131K tokens. In practice with a sequence length of 16K and W = 4096, SWA modifications in FlashAttention and xFormers result in a 2x speed enhancement compared to vanilla attention methods.