WIKIPEDIA
The Free Encyclopedia

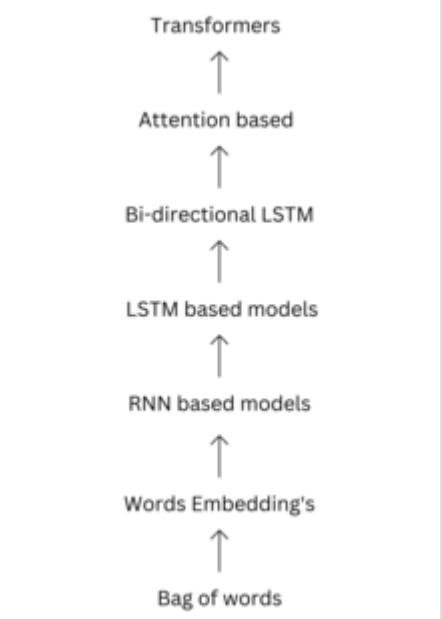
Transformer (deep learning architecture)

A **transformer** is a deep learning architecture developed by Google and based on the multi-head attention mechanism, proposed in a 2017 paper "Attention Is All You Need".[1] Text is converted to numerical representations called tokens, and each token is converted into a vector via looking up from a word embedding table.[1] At each layer, each token is then contextualized within the scope of the context window with other (unmasked) tokens via a parallel multi-head attention mechanism allowing the signal for key tokens to be amplified and less important tokens to be diminished. The transformer paper, published in 2017, is based on the softmax-based attention mechanism proposed by Bahdanau *et. al.* in 2014 for machine translation,[2][3] and the Fast Weight Controller, similar to a transformer, proposed in 1992.[4][5][6]

Transformers have the advantage of having no recurrent units, and thus requires less training time than previous recurrent neural architectures, such as long short-term memory (LSTM),[7] and its later variation has been prevalently adopted for training large language models (LLM) on large (language) datasets, such as the Wikipedia corpus and Common Crawl.[8]

This architecture is now used not only in natural language processing and computer vision,[9] but also in audio[10] and multi-modal processing. It has also led to the development of pre-trained systems, such as generative pre-trained transformers (GPTs)[11] and BERT[12] (Bidirectional Encoder Representations from Transformers).

Timeline



Timeline of natural language processing models

- In 1990, the Elman network, using a recurrent neural network, encoded each word in a training set as a vector, called a word embedding, and the whole vocabulary as a vector database, allowing it to perform such tasks as sequence-predictions that are beyond the power of a simple multilayer perceptron. A shortcoming of the static embeddings was that they didn't differentiate between multiple meanings of same-spelt words.[13]
- In 1992, the Fast Weight Controller was published by Jürgen Schmidhuber.[4] It learns to answer queries by programming the attention weights of another neural network through outer products of key vectors and value vectors called FROM and TO. The Fast Weight Controller was later shown to be equivalent to the unnormalized linear Transformer.[6][5][14][15] The terminology "learning internal spotlights of attention" was introduced in 1993.[16]
- In 1993, the IBM alignment models were used for statistical machine translation.[17]
- In 1997, a precursor of large language model, using recurrent neural networks, such as long short-term memory, was proposed.
- In 2001, a one-billion-word large text corpus, scraped from the Internet, referred to as "very very large" at the time, was used for word disambiguation.[18]

- In 2012, AlexNet demonstrated the effectiveness of large neural networks for image recognition, encouraging large artificial neural networks approach instead of older, statistical approaches.
- In 2014, a 380M-parameter seq2seq model for machine translation using two Long short-term Memory (LSTMs) networks was proposed by Sutskever at al.[19] The architecture consists of two parts. The *encoder* is an LSTM that takes in a sequence of tokens and turns it into a vector. The *decoder* is another LSTM that converts the vector into a sequence of tokens.
- In 2014, gating proved to be useful in a 130M-parameter seq2seq model, which used a simplified gated recurrent units (GRUs). Bahdanau et al[20] showed that GRUs are neither better nor worse than gated LSTMs.[21][22]
- In 2014, Bahdanau et al.[23] improved the previous seq2seq model by using an "additive" kind of attention mechanism in-between two LSTM networks. It was, however, not yet the parallelizable (scaled "dot product") kind of attention, later proposed in the 2017 transformer paper.
- In 2015, the relative performance of Global and Local (windowed) attention model architectures were assessed by Luong et al, a mixed attention architecture found to improve on the translations offered by Bahdanau's architecture, while the use of a local attention architecture reduced translation time.[24]
- In 2016, Google Translate gradually replaced the older statistical machine translation approach with the newer neural-networks-based approach that included a seq2seq model combined by LSTM and the "additive" kind of attention mechanism. They achieved a higher level of performance than the statistical approach, which took ten years to develop, in only nine months.[25][26]
- In 2017, the original (100M-sized) encoder-decoder transformer model with a faster (parallelizable or decomposable) attention mechanism was proposed in the "Attention is all you need" paper. As the model had difficulties converging, it was suggested that the learning rate should be linearly scaled up from 0 to maximal value for the first part of the training (i.e. 2% of the total number of training steps). The intent of the transformer model is to take a seq2seq model and remove its recurrent neural networks, but preserve its additive attention mechanism.[1]
- In 2018, in the ELMo paper, an entire sentence was processed before an embedding vector was assigned to each word in the sentence. A bi-directional LSTM was used to calculate such, deep contextualized embeddings for each word, improving upon the line of research from bag of words and word2vec.
- In 2018, an encoder-only transformer was used in the (more than 1B-sized) BERT model, improving upon ELMo.[27]
- In 2020, vision transformer[28] and speech-processing convolution-augmented transformer[29] outperformed recurrent neural networks, previously used for vision and speech.
- In 2020, difficulties with converging the original transformer were solved by normalizing layers *before* (instead of after) multiheaded attention by Xiong et al. This is called **pre-LN** Transformer.[30]
- In 2023, uni-directional ("autoregressive") transformers were being used in the (more than 100B-sized) GPT-3 and other OpenAI GPT models.[31][32]

Predecessors

Before transformers, predecessors of attention mechanism were added to gated recurrent neural networks, such as LSTMs and gated recurrent units (GRUs), which processed datasets sequentially. Dependency on previous token computations prevented them from being able to parallelize the attention mechanism. In 1992, fast weight controller was proposed as an alternative to recurrent neural networks that can learn "internal spotlights of attention".[16][4] In theory, the information from one token can propagate arbitrarily far down the sequence, but in practice the vanishing-gradient problem leaves the model's state at the end of a long sentence without precise, extractable information about preceding tokens.

The performance of old models was enhanced by adding an attention mechanism, which allowed a model to access any preceding point along the sequence. The attention layer weighs all previous states according to a learned measure of relevance, providing relevant information about far-away tokens. This proved to be especially useful in language translation, where far-away context can be essential for the meaning of a word in a sentence. The state vector has been accessible only after the *last* English word was processed while, for example, translating it from French by a LSTM model. Although in theory such a vector retains the information about the whole original sentence, in practice the information is poorly preserved. If an attention mechanism is added, the decoder is given access to the state vectors of every input word, not just the last, and can learn attention weights that dictate how much to attend to each input state vector. The augmentation of seq2seq models with the attention mechanism was first implemented in the context of machine translation by Bahdanau, Cho, and Bengio in 2014.[2][3]

Decomposable attention

In 2016, highly parallelizable *decomposable attention* was successfully combined with a feedforward network.[33] This indicated that attention mechanisms were powerful in themselves and that sequential recurrent processing of data was not necessary to achieve the quality gains of recurrent neural networks with attention. In 2017, Vaswani et al. also proposed replacing recurrent neural networks with self-attention and started the effort to evaluate that idea.[1] Transformers, using an attention mechanism, processing all tokens simultaneously, calculated "soft" 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.

Training

Methods for stabilizing training

The plain transformer architecture had difficulty converging. In the original paper[1] the authors recommended using learning rate warmup. That is, the learning rate should linearly scale up from 0 to maximal value for the first part of the training (usually recommended to be 2% of the total number of training steps), before decaying again.

A 2020 paper found that using layer normalization *before* (instead of after) multiheaded attention and feedforward layers stabilizes training, not requiring learning rate warmup.[30]

Pretrain-finetune

Transformers typically undergo self-supervised learning involving unsupervised pretraining followed by supervised fine-tuning. Pretraining is typically done on a larger dataset than fine-tuning, due to the limited availability of labeled training data. Tasks for pretraining and fine-tuning commonly include:

- language modeling[12]
- next-sentence prediction[12]
- question answering[8]
- reading comprehension
- sentiment analysis[1]
- paraphrasing[1]

The T5 transformer paper[34] documents a large number of pretraining tasks. Some examples are:

- restoring corrupted text: `Thank you <X> me to your party <Y> week.` -> `<X> for inviting <Y> last <Z>` where the `<Z>` means "end of output".
- translation: `translate English to German: That is good.` -> `Das ist gut.`.
- judging the grammatical acceptability of a sentence (CoLA sentence): `The course is jumping well.` -> `not acceptable`.

Applications

The transformer has had great success in natural language processing (NLP), for example the tasks of machine translation and time series prediction. Many large language models such as GPT-2, GPT-3, GPT-4, Claude, BERT, XLNet, RoBERTa and ChatGPT demonstrate the ability of transformers to perform a wide variety of such NLP-related tasks, and have the potential to find real-world applications. These may include:

- machine translation
- document summarization
- document generation
- named entity recognition (NER)[35]
- biological sequence analysis
- writing computer code based on requirements expressed in natural language.
- video understanding.

In addition to the NLP applications, it has also been successful in other fields, such as computer vision, or the protein folding applications (such as AlphaFold).

As an illustrative example, *Ithaca* is an encoder-only transformer with *three* output heads. It takes as input ancient Greek inscription as sequences of characters, but with illegible characters replaced with "-". Its three output heads respectively outputs probability distributions over Greek characters, location of inscription, and date of inscription.[36]

Implementations

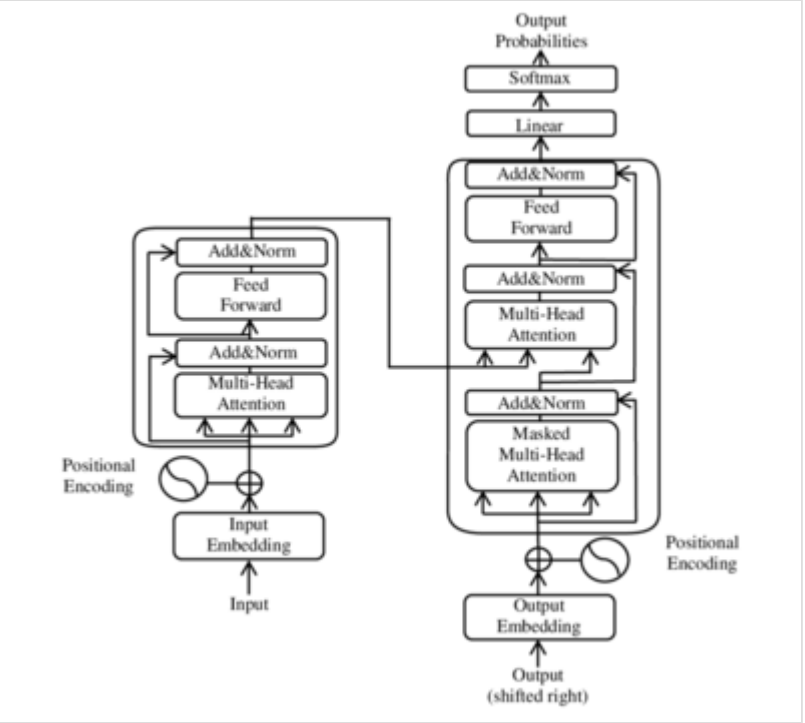
The transformer model has been implemented in standard deep learning frameworks such as TensorFlow and PyTorch.

Transformers is a library produced by Hugging Face that supplies transformer-based architectures and pretrained models.[11]

Architecture

All transformers have the same primary components:

- Tokenizers, which convert text into tokens.
- A single embedding layer, which converts tokens and positions of the tokens into vector representations.
- Transformer layers, which carry out repeated transformations on the vector representations, extracting more and more linguistic information. These consist of alternating attention and feedforward layers.
- (optional) Un-embedding layer, which converts the final vector representations back to a probability distribution over the tokens.



An illustration of main components of the transformer model from the original paper, where layer normalization was performed after multiheaded attention. In a 2020 paper it was found that placing the layer normalization in front of the multiheaded attention (instead of after) improves the training stability[30].

Transformer layers can be one of two types, *encoder* and *decoder*. In the original paper both of them were used, while later models included only one type of them. BERT is an example of an encoder-only model; GPT are decoder-only models.

Input

The input text is parsed into tokens by a tokenizer, most often a byte pair encoding tokenizer, and each token is converted into a vector via looking up from a word embedding table. Then, positional information of the token is added to the word embedding.

Encoder-decoder architecture

Like earlier seq2seq models, the original transformer model used an **encoder-decoder** architecture. The encoder consists of encoding layers that process the input tokens iteratively one layer after another, while the decoder consists of decoding layers that iteratively process the encoder's output as well as the decoder output's tokens so far.

The function of each encoder layer is to generate contextualized token representations, where each representation corresponds to a token that "mixes" information from other input tokens via self-attention mechanism. Each decoder layer contains two attention sublayers: (1) cross-attention for incorporating the

output of encoder (contextualized input token representations), and (2) self-attention for "mixing" information among the input tokens to the decoder (i.e., the tokens generated so far during inference time).[37][38]

Both the encoder and decoder layers have a feed-forward neural network for additional processing of the outputs and contain residual connections and layer normalization steps.[38]

Scaled dot-product attention

The transformer building blocks are scaled dot-product attention units. For each attention unit, the transformer model learns three weight matrices: the query weights W_Q, the key weights W_K, and the value weights W_V. For each token i, the input token representation x_i is multiplied with each of the three weight matrices to produce a query vector $q_i = x_i W_Q$, a key vector $k_i = x_i W_K$, and a value vector $v_i = x_i W_V$. 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 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. $q_i \cdot k_j$ is large), this does not necessarily mean that token j will attend to token i (i.e. $q_j \cdot k_i$ could be small). 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.

The attention calculation for all tokens can be expressed as one large matrix calculation using the softmax function, which is useful for training due to computational matrix operation optimizations that quickly compute matrix operations. The matrices Q, K and V are defined as the matrices where the ith rows are vectors q_i, k_i, and v_i respectively. Then we can represent the attention as

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

where softmax is taken over the horizontal axis.

Multi-head 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. While each attention head attends to the tokens that are relevant to each token, multiple attention heads allow the model to do this for different definitions of "relevance". In addition, the influence field representing relevance can become progressively dilated in successive layers. 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.[39] The computations for each attention head can be performed in parallel, which allows for fast processing. The outputs for the attention layer are concatenated to pass into the feed-forward neural network layers.

Concretely, let the multiple attention heads be indexed by i, then we have

$$\text{MultiheadedAttention}(Q, K, V) = \text{Concat}_{i \in [\#heads]}(\text{Attention}(XW_i^Q, XW_i^K, XW_i^V))W^O$$

where the matrix X is the concatenation of word embeddings, and 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.

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 M that is $-\infty$ at entries where the attention link must be cut, and 0 at other places:

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

For example, the following mask matrix is used in autoregressive modeling:

$$M = \begin{bmatrix} 0 & -\infty & -\infty & \dots & -\infty \\ 0 & 0 & -\infty & \dots & -\infty \\ 0 & 0 & 0 & \dots & -\infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

In words, it means that each token can pay attention to itself, and every token before it, but not any after it.

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. 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.

The first encoder takes positional information and embeddings of the input sequence as its input, rather than encodings. The positional information is necessary for the transformer to make use of the order of the sequence, because no other part of the transformer makes use of this.[1]

The encoder is bidirectional. Attention can be placed on tokens before and after the current token. Tokens are used instead of words to account for polysemy.

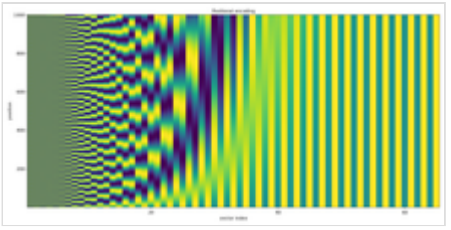
Positional encoding

A positional encoding is a fixed-size vector representation that encapsulates the relative positions of tokens within a target sequence: it provides the transformer model with information about *where* the words are in the input sequence.

The positional encoding is defined as a function of type $f : \mathbb{R} \to \mathbb{R}^d; d \in \mathbb{Z}, d > 0$, where d is a positive even integer. The full positional encoding – as defined in the original paper – is given by the equation:

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

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



A diagram of a sinusoidal positional encoding with parameters $N = 10000, d = 100$

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,[1] the authors chose $N = 10000$.

The function is in a simpler form when written as a complex function of type $f : \mathbb{R} \to \mathbb{C}^{d/2}$

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

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

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

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

where $\Delta t \in \mathbb{R}$ is the distance one wishes to shift. 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.

By taking a linear sum, any convolution can also be implemented as linear transformations:

$$\sum_j c_j f(t + \Delta t_j) = \left(\sum_j c_j \, \text{diag}(f(\Delta t_j))\right) f(t)$$

for any constants c_j. This allows the transformer to take any encoded position and find a linear sum of the encoded locations of its neighbors. This sum of encoded positions, when fed into the attention mechanism, would create attention weights on its neighbors, much like what happens in a convolutional neural network language model. In the author's words, "we hypothesized it would allow the model to easily learn to attend by relative position."

In typical implementations, all operations are done over the real numbers, not the complex numbers, but since complex multiplication can be implemented as real 2-by-2 matrix multiplication, this is a mere notational difference.

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*.[1][38]

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.[1] This allows for autoregressive text generation. For all attention heads, attention can't be placed on following tokens. The last decoder is followed by a final linear transformation and softmax layer, to produce the output probabilities over the vocabulary.

All members of OpenAI's GPT series have a decoder-only architecture.

Terminology

In large language models, the terminology is somewhat different than the terminology used in the original Transformer paper:[40]

- "encoder only": full encoder, full decoder.
- "encoder-decoder": full encoder, autoregressive decoder.
- "decoder only": autoregressive encoder, autoregressive decoder.

Here "autoregressive" means that a mask is inserted in the attention head to zero out all attention from one token to all tokens following it, as described in the "masked attention" section.

Generally, Transformer-based language models are of two types: causal (or "autoregressive") and masked. The GPT series is causal and decoder only. BERT is masked and encoder only.[41][42] The T5 series is encoder-decoder, with a full encoder and autoregressive decoder.[34]

Subsequent work

Alternative activation functions

The original transformer uses ReLU activation function. Other activation functions were developed, such as SwiGLU.[43]

Alternative positional encodings

Transformers may use other positional encoding methods than sinusoidal.[44]

RoPE

RoPE (rotary positional embedding),[45] is best explained by considering a list of 2-dimensional vectors $[(x_1^{(1)}, x_1^{(2)}), (x_2^{(1)}, x_2^{(2)}), (x_3^{(1)}, x_3^{(2)}), \dots]$. Now pick some angle θ. Then RoPE encoding is

$$\mathrm{RoPE}\left(x_m^{(1)}, x_m^{(2)}, m\right) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} x_m^{(1)} \\ x_m^{(2)} \end{pmatrix} = \begin{pmatrix} x_m^{(1)} \cos m\theta - x_m^{(2)} \sin m\theta \\ x_m^{(2)} \cos m\theta + x_m^{(1)} \sin m\theta \end{pmatrix}$$

Equivalently, if we write the 2-dimensional vectors as complex numbers $z_m := x_m^{(1)} + i x_m^{(2)}$, then RoPE encoding is just multiplication by an angle:

$$\mathbf{RoPE}\big(z_m, m\big) = e^{im\theta} z_m$$

For a list of $2n$-dimensional vectors, a RoPE encoder is defined by a sequence of angles $\theta^{(1)}, \dots, \theta^{(n)}$. Then the RoPE encoding is applied to each pair of coordinates.

The benefit of RoPE is that the dot-product between two vectors depends on their relative location only:

$$\mathbf{RoPE}\big(x, m\big)^T \mathbf{RoPE}\big(y, n\big) = \mathbf{RoPE}\big(x, m+k\big)^T \mathbf{RoPE}\big(y, n+k\big)$$

for any integer k.

ALiBi

ALiBi (Attention with Linear Biases)[46] is not a *replacement* for the positional encoder on the original transformer. Instead, it is an *additional* positional encoder that is directly plugged into the attention mechanism. Specifically, the ALiBi attention mechanism is

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

Here, s is a real number ("scalar"), and B is the *linear bias* matrix defined by

$$B = \begin{pmatrix} 0 & 1 & 2 & 3 & \cdots \\ -1 & 0 & 1 & 2 & \cdots \\ -2 & -1 & 0 & 1 & \cdots \\ -3 & -2 & -1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

in other words, $B_{i,j} = j - i$.

ALiBi allows pretraining on short context windows, then finetuning on longer context windows. Since it is directly plugged into the attention mechanism, it can be combined with any positional encoder that is plugged into the "bottom" of the entire network (which is where the sinusoidal encoder on the original transformer, as well as RoPE and many others, are located).

Relative Position Encodings

Relative Position Encodings[47] is similar to ALiBi, but more generic:

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

where B is a Toeplitz matrix, that is, $B_{i,j} = B_{i',j'}$ whenever $i - j = i' - j'$.

Efficient implementation

FlashAttention

FlashAttention[48] is an algorithm that implements the transformer attention mechanism efficiently on a GPU. It performs matrix multiplications in blocks, such that each block fits within the cache of a GPU, and by careful management of the blocks it minimizes data copying between GPU caches (as data movement is slow).

An improved version, FlashAttention-2,[49][50][51] was developed to cater to the rising demand for language models capable of handling longer context lengths. It offers enhancements in work partitioning and parallelism, enabling it to achieve up to 230 TFLOPs/s on A100 GPUs (FP16/BF16), a 2x speed increase over the original FlashAttention.

Key advancements in FlashAttention-2 include the reduction of non-matmul FLOPs, improved parallelism over the sequence length dimension, better work partitioning between GPU warps, and added support for head dimensions up to 256 and multi-query attention (MQA) and grouped-query attention (GQA).

Benchmarks revealed FlashAttention-2 to be up to 2x faster than FlashAttention and up to 9x faster than a standard attention implementation in PyTorch. Future developments include optimization for new hardware like H100 GPUs and new data types like FP8.

Multi-Query Attention

Multi-Query Attention changes the multiheaded attention mechanism.[52] Whereas normally,

$$\text{MultiheadedAttention}(Q, K, V) = \text{Concat}_{i \in [\#heads]}(\text{Attention}(XW_i^Q, XW_i^K, XW_i^V))W^O$$

with Multi-Query Attention, there is just one W^K, W^V, thus:

$$\text{MultiQueryAttention}(Q, K, V) = \text{Concat}_{i \in [\#heads]}(\text{Attention}(XW_i^Q, XW^K, XW^V))W^O$$

This has a neutral effect on model quality and training speed, but increases inference speed.

Speculative decoding

Transformers are used in large language models for autoregressive sequence generation: generating a stream of text, one token at a time. However, in most settings, decoding from a language models is memory-bound, meaning that we have spare compute power available. Speculative decoding[53][54] uses this spare compute power by computing several tokens in parallel. Similarly to speculative execution in CPUs, future tokens are computed concurrently, by speculating on the value of previous tokens, and are later discarded if it turns out the speculation was incorrect.

Specifically, consider a transformer model like GPT-3 with a context window size of 512. To generate an entire context window autoregressively with greedy decoding, it must be run for 512 times, each time generating a token $x_1, x_2, \ldots, x_{512}$. However, if we had some educated guess for the values of these tokens, we could verify all of them in parallel, in one run of the model, by checking that each x_t is indeed the token with the largest log-likelihood in the t-th output.

In speculative decoding, a smaller model or some other simple heuristic is used to generate a few speculative tokens that are subsequently verified by the larger model. For example, suppose a small model generated four speculative tokens: $\tilde{x_1}, \tilde{x_2}, \tilde{x_3}, \tilde{x_4}$. These tokens are run through the larger model, and only $\tilde{x_1}$ and $\tilde{x_2}$ are accepted. The same run of the large model already generated a new token x_3 to replace $\tilde{x_3}$, and $\tilde{x_4}$ is completely discarded. The process then repeats (starting from the 4th token) until all tokens are generated.

For non-greedy decoding, similar ideas apply, except the speculative tokens are accepted or rejected stochastically, in a way that guarantees the final output distribution is the same as if speculative decoding was not used.[53][55]

Sub-quadratic transformers

Training transformer-based architectures can be expensive, especially for long inputs.[56] Alternative architectures include the Reformer (which reduces the computational load from $O(N^2)$ to $O(N \ln N)$[56]), or models like ETC/BigBird (which can reduce it to $O(N)$)[57] where N is the length of the sequence. This is done using locality-sensitive hashing and reversible layers.[58][59]

Ordinary transformers require a memory size that is quadratic in the size of the context window. Attention-free transformers[60] reduce this to a linear dependence while still retaining the advantages of a transformer by linking the key to the value.

Long Range Arena (2020)[61] is a standard benchmark for comparing the behavior of transformer architectures over long inputs.

Random Feature Attention (2021)[62] uses Fourier random features:

$$\varphi(x) = \frac{1}{\sqrt{D}}[\cos\langle w_1, x\rangle, \sin\langle w_1, x\rangle, \cdots \cos\langle w_D, x\rangle, \sin\langle w_D, x\rangle]^T$$

where $w_1, \ldots, w_D$ are independent samples from the normal distribution $N(0, \sigma^2 I)$. This choice of parameters satisfy $\mathbb{E}[\langle \varphi(x), \varphi(y)\rangle] = e^{\frac{\|x-y\|^2}{2\sigma^2}}$, or

$$e^{\langle x,y\rangle/\sigma^2} = \mathbb{E}[\langle e^{\|x\|^2/2\sigma^2}\varphi(x), e^{\|y\|^2/2\sigma^2}\varphi(y)\rangle] \approx \langle e^{\|x\|^2/2\sigma^2}\varphi(x), e^{\|y\|^2/2\sigma^2}\varphi(y)\rangle$$

Consequently, the one-headed attention, with one query, can be written as

$$\text{Attention}(q, K, V) = \text{softmax}\left(\frac{qK^{\mathrm{T}}}{\sqrt{d_k}}\right)V \approx \frac{\varphi(q)^T \sum_i e^{\|k_i\|^2/2\sigma^2}\varphi(k_i)v_i^T}{\varphi(q)^T \sum_i e^{\|k_i\|^2/2\sigma^2}\varphi(k_i)}$$

where $\sigma = d_K^{1/4}$. Similarly for multiple queries, and for multiheaded attention.

This approximation can be computed in linear time, as we can compute the matrix $\varphi(k_i)v_i^T$ first, then multiply it with the query. In essence, we have managed to obtain a more precise version of

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

Performer (2022)[63] uses the same Random Feature Attention, but $w_1, \dots, w_D$ are first independently sampled from the normal distribution $N(0, \sigma^2 I)$, then they are Gram-Schmidt processed.

Multimodality

Transformers can also be used/adapted for modalities (input or output) beyond just text, usually by finding a way to "tokenize" the modality.

Vision transformers[28] adapt the transformer to computer vision by breaking down input images as a series of patches, turning them into vectors, and treating them like tokens in a standard transformer.

Conformer[29] and later Whisper[64] follow the same pattern for speech recognition, first turning the speech signal into a spectrogram, which is then treated like an image, i.e. broken down into a series of patches, turned into vectors and treated like tokens in a standard transformer.

Perceivers by Andrew Jaegle et al. (2021)[65][66] can learn from large amounts of heterogeneous data.

Regarding image *outputs*, Peebles et al introduced a diffusion transformer (DiT) which facilitates use of the transformer architecture for diffusion-based image production.[67] Also, Google released a transformer-centric image generator called "Muse" based on parallel decoding and masked generative transformer technology.[68] (Transformers played a less-central role with prior image-producing technologies,[69] albeit still a significant one.[70])

See also

- Perceiver – Machine learning algorithm for non-textual data
- BERT (language model) – Language model developed by Google
- GPT-3 – 2020 text-generating language model
- GPT-4 – 2023 text-generating language model
- ChatGPT – Chatbot and virtual assistant developed by OpenAI
- Wu Dao – Chinese multimodal artificial intelligence program
- Vision transformer – Machine learning algorithm for vision processing
- BLOOM (language model) – Open-access multilingual language model

References

1. Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N; Kaiser, Łukasz; Polosukhin, Illia (2017). "Attention is All you Need" (https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf) (PDF). *Advances in Neural Information Processing Systems*. **30**. Curran Associates, Inc.

2. Bahdanau; Cho, Kyunghyun; Bengio, Yoshua (September 1, 2014). "Neural Machine Translation by Jointly Learning to Align and Translate". arXiv:1409.0473 (https://arxiv.org/abs/1409.0473) [cs.CL (https://arxiv.org/archive/cs.CL)].
3. Luong, Minh-Thang; Pham, Hieu; Manning, Christopher D. (August 17, 2015). "Effective Approaches to Attention-based Neural Machine Translation". arXiv:1508.04025 (https://arxiv.org/abs/1508.04025) [cs.CL (https://arxiv.org/archive/cs.CL)].
4. Schmidhuber, Jürgen (1992). "Learning to control fast-weight memories: an alternative to recurrent nets". *Neural Computation*. **4** (1): 131–139. doi:10.1162/neco.1992.4.1.131 (https://doi.org/10.1162%2Fneco.1992.4.1.131). S2CID 16683347 (https://api.semanticscholar.org/CorpusID:16683347).
5. Schlag, Imanol; Irie, Kazuki; Schmidhuber, Jürgen (2021). "Linear Transformers Are Secretly Fast Weight Programmers". *ICML 2021*. Springer. pp. 9355–9366.
6. Katharopoulos, Angelos; Vyas, Apoorv; Pappas, Nikolaos; Fleuret, François (2020). "Transformers are RNNs: Fast autoregressive Transformers with linear attention" (https://paperswithcode.com/paper/a-decomposable-attention-model-for-natural). *ICML 2020*. PMLR. pp. 5156–5165.
7. Hochreiter, Sepp; Schmidhuber, Jürgen (1 November 1997). "Long Short-Term Memory". *Neural Computation*. **9** (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735 (https://doi.org/10.1162%2Fneco.1997.9.8.1735). ISSN 0899-7667 (https://www.worldcat.org/issn/0899-7667). PMID 9377276 (https://pubmed.ncbi.nlm.nih.gov/9377276). S2CID 1915014 (https://api.semanticscholar.org/CorpusID:1915014).
8. "Better Language Models and Their Implications" (https://openai.com/blog/better-language-models/). *OpenAI*. 2019-02-14. Archived (https://web.archive.org/web/20201219132206/https://openai.com/blog/better-language-models/) from the original on 2020-12-19. Retrieved 2019-08-25.
9. He, Cheng (31 December 2021). "Transformer in CV" (https://towardsdatascience.com/transformer-in-cv-bbdb58bf335e). *Transformer in CV*. Towards Data Science. Archived (https://web.archive.org/web/20230416160902/https://towardsdatascience.com/transformer-in-cv-bbdb58bf335e?gi=78319b60873f) from the original on 16 April 2023. Retrieved 19 June 2021.
10. Radford, Alec; Jong Wook Kim; Xu, Tao; Brockman, Greg; McLeavey, Christine; Sutskever, Ilya (2022). "Robust Speech Recognition via Large-Scale Weak Supervision". arXiv:2212.04356 (https://arxiv.org/abs/2212.04356) [eess.AS (https://arxiv.org/archive/eess.AS)].
11. Wolf, Thomas; Debut, Lysandre; Sanh, Victor; Chaumond, Julien; Delangue, Clement; Moi, Anthony; Cistac, Pierric; Rault, Tim; Louf, Remi; Funtowicz, Morgan; Davison, Joe; Shleifer, Sam; von Platen, Patrick; Ma, Clara; Jernite, Yacine; Plu, Julien; Xu, Canwen; Le Scao, Teven; Gugger, Sylvain; Drame, Mariama; Lhoest, Quentin; Rush, Alexander (2020). "Transformers: State-of-the-Art Natural Language Processing". *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. pp. 38–45. doi:10.18653/v1/2020.emnlp-demos.6 (https://doi.org/10.18653%2Fv1%2F2020.emnlp-demos.6). S2CID 208117506 (https://api.semanticscholar.org/CorpusID:208117506).
12. "Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing" (http://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html). *Google AI Blog*. 2 November 2018. Archived (https://web.archive.org/web/20210113211449/https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html) from the original on 2021-01-13. Retrieved 2019-08-25.
13. Elman, Jeffrey L. (March 1990). "Finding Structure in Time" (http://doi.wiley.com/10.1207/s15516709cog1402_1). *Cognitive Science*. **14** (2): 179–211. doi:10.1207/s15516709cog1402_1 (https://doi.org/10.1207%2Fs15516709cog1402_1). S2CID 2763403 (https://api.semanticscholar.org/CorpusID:2763403).

14. Choromanski, Krzysztof; Likhosherstov, Valerii; Dohan, David; Song, Xingyou; Gane, Andreea; Sarlos, Tamas; Hawkins, Peter; Davis, Jared; Mohiuddin, Afroz; Kaiser, Lukasz; Belanger, David; Colwell, Lucy; Weller, Adrian (2020). "Rethinking Attention with Performers". arXiv:2009.14794 (https://arxiv.org/abs/2009.14794) [cs.CL (https://arxiv.org/archive/cs.CL)].
15. Schmidhuber, Juergen (26 March 2021). "26 March 1991: Neural nets learn to program neural nets with fast weights—the first Transformer variants. 2021-: New stuff!" (https://web.archive.org/web/20231205213949/https://people.idsia.ch/~juergen/fast-weight-programmer-1991-transformer.html). IDSIA, Switzerland. Archived from the original (https://people.idsia.ch/~juergen/fast-weight-programmer-1991-transformer.html) on 5 Dec 2023. Retrieved 29 Dec 2023.
16. Schmidhuber, Jürgen (1993). "Reducing the ratio between learning complexity and number of time-varying variables in fully recurrent nets". *ICANN 1993*. Springer. pp. 460–463.
17. Brown, Peter F. (1993). "The mathematics of statistical machine translation: Parameter estimation". *Computational Linguistics* (19): 263–311.
18. Banko, Michele; Brill, Eric (2001). "Scaling to very very large corpora for natural language disambiguation" (https://doi.org/10.3115%2F1073012.1073017). *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics - ACL '01*. Morristown, NJ, USA: Association for Computational Linguistics: 26–33. doi:10.3115/1073012.1073017 (https://doi.org/10.3115%2F1073012.1073017). S2CID 6645623 (https://api.semanticscholar.org/CorpusID:6645623).
19. Sutskever, Ilya; Vinyals, Oriol; Le, Quoc V (2014). "Sequence to Sequence Learning with Neural Networks" (https://proceedings.neurips.cc/paper_files/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html). *Advances in Neural Information Processing Systems*. **27**. Curran Associates, Inc. arXiv:1409.3215 (https://arxiv.org/abs/1409.3215).
20. Cho, Kyunghyun; van Merrienboer, Bart; Bahdanau, Dzmitry; Bengio, Yoshua (2014). "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches" (https://dx.doi.org/10.3115/v1/w14-4012). *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Stroudsburg, PA, USA: Association for Computational Linguistics: 103–111. arXiv:1409.1259 (https://arxiv.org/abs/1409.1259). doi:10.3115/v1/w14-4012 (https://doi.org/10.3115%2Fv1%2Fw14-4012). S2CID 11336213 (https://api.semanticscholar.org/CorpusID:11336213).
21. Chung, Junyoung; Gulcehre, Caglar; Cho, KyungHyun; Bengio, Yoshua (2014). "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". arXiv:1412.3555 (https://arxiv.org/abs/1412.3555) [cs.NE (https://arxiv.org/archive/cs.NE)].
22. Gruber, N.; Jockisch, A. (2020), "Are GRU cells more specific and LSTM cells more sensitive in motive classification of text?", *Frontiers in Artificial Intelligence*, **3**: 40, doi:10.3389/frai.2020.00040 (https://doi.org/10.3389%2Ffrai.2020.00040), PMC 7861254 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7861254), PMID 33733157 (https://pubmed.ncbi.nlm.nih.gov/33733157), S2CID 220252321 (https://api.semanticscholar.org/CorpusID:220252321)
23. Bahdanau, Dzmitry; Cho, Kyunghyun; Bengio, Yoshua (2014-09-01). "Neural Machine Translation by Jointly Learning to Align and Translate". arXiv:1409.0473 (https://arxiv.org/abs/1409.0473) [cs.CL (https://arxiv.org/archive/cs.CL)].
24. "Google Scholar" (https://scholar.google.com/scholar_lookup?arxiv_id=1508.04025). *scholar.google.com*. Retrieved 2023-08-13.
25. Lewis-Kraus, Gideon (2016-12-14). "The Great A.I. Awakening" (https://web.archive.org/web/20230524052626/https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html). *The New York Times*. ISSN 0362-4331 (https://www.worldcat.org/issn/0362-4331). Archived from the original (https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html) on 24 May 2023. Retrieved 2023-06-22.

26. Wu, Yonghui; et al. (2016-09-01). "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". arXiv:1609.08144 (https://arxiv.org/abs/1609.08144) [cs.CL (https://arxiv.org/archive/cs.CL)].
27. Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina (11 October 2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". arXiv:1810.04805v2 (https://arxiv.org/abs/1810.04805v2) [cs.CL (https://arxiv.org/archive/cs.CL)].
28. Dosovitskiy, Alexey; Beyer, Lucas; Kolesnikov, Alexander; Weissenborn, Dirk; Zhai, Xiaohua; Unterthiner, Thomas; Dehghani, Mostafa; Minderer, Matthias; Heigold, Georg; Gelly, Sylvain; Uszkoreit, Jakob (2021-06-03). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". arXiv:2010.11929 (https://arxiv.org/abs/2010.11929) [cs.CV (https://arxiv.org/archive/cs.CV)].
29. Gulati, Anmol; Qin, James; Chiu, Chung-Cheng; Parmar, Niki; Zhang, Yu; Yu, Jiahui; Han, Wei; Wang, Shibo; Zhang, Zhengdong; Wu, Yonghui; Pang, Ruoming (2020). "Conformer: Convolution-augmented Transformer for Speech Recognition". arXiv:2005.08100 (https://arxiv.org/abs/2005.08100) [eess.AS (https://arxiv.org/archive/eess.AS)].
30. Xiong, Ruibin; Yang, Yunchang; He, Di; Zheng, Kai; Zheng, Shuxin; Xing, Chen; Zhang, Huishuai; Lan, Yanyan; Wang, Liwei; Liu, Tie-Yan (2020-06-29). "On Layer Normalization in the Transformer Architecture". arXiv:2002.04745 (https://arxiv.org/abs/2002.04745) [cs.LG (https://arxiv.org/archive/cs.LG)].
31. "Improving language understanding with unsupervised learning" (https://openai.com/research/language-unsupervised). *openai.com*. June 11, 2018. Archived (https://web.archive.org/web/20230318210736/https://openai.com/research/language-unsupervised) from the original on 2023-03-18. Retrieved 2023-03-18.
32. *finetune-transformer-lm* (https://github.com/openai/finetune-transformer-lm), OpenAI, June 11, 2018, retrieved 2023-05-01
33. "Papers with Code – A Decomposable Attention Model for Natural Language Inference" (https://paperswithcode.com/paper/a-decomposable-attention-model-for-natural). *paperswithcode.com*.
34. Raffel, Colin; Shazeer, Noam; Roberts, Adam; Lee, Katherine; Narang, Sharan; Matena, Michael; Zhou, Yanqi; Li, Wei; Liu, Peter J. (2020-01-01). "Exploring the limits of transfer learning with a unified text-to-text transformer" (https://dl.acm.org/doi/abs/10.5555/3455716.3455856). *The Journal of Machine Learning Research*. **21** (1): 140:5485–140:5551. arXiv:1910.10683 (https://arxiv.org/abs/1910.10683). ISSN 1532-4435 (https://www.worldcat.org/issn/1532-4435).
35. Kariampuzha, William; Alyea, Gioconda; Qu, Sue; Sanjak, Jaleal; Mathé, Ewy; Sid, Eric; Chatelaine, Haley; Yadaw, Arjun; Xu, Yanji; Zhu, Qian (2023). "Precision information extraction for rare disease epidemiology at scale" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9972634). *Journal of Translational Medicine*. **21** (1): 157. doi:10.1186/s12967-023-04011-y (https://doi.org/10.1186%2Fs12967-023-04011-y). PMC 9972634 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9972634). PMID 36855134 (https://pubmed.ncbi.nlm.nih.gov/36855134).
36. Assael, Yannis; Sommerschield, Thea; Shillingford, Brendan; Bordbar, Mahyar; Pavlopoulos, John; Chatzipanagiotou, Marita; Androutsopoulos, Ion; Prag, Jonathan; de Freitas, Nando (March 2022). "Restoring and attributing ancient texts using deep neural networks" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8907065). *Nature*. **603** (7900): 280–283. Bibcode:2022Natur.603..280A (https://ui.adsabs.harvard.edu/abs/2022Natur.603..280A). doi:10.1038/s41586-022-04448-z (https://doi.org/10.1038%2Fs41586-022-04448-z). ISSN 1476-4687 (https://www.worldcat.org/issn/1476-4687). PMC 8907065 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8907065). PMID 35264762 (https://pubmed.ncbi.nlm.nih.gov/35264762).

37. "Sequence Modeling with Neural Networks (Part 2): Attention Models" (https://indico.io/blog/sequence-modeling-neural-networks-part2-attention-models/). *Indico*. 2016-04-18. Archived (https://web.archive.org/web/20201021203352/https://indico.io/blog/sequence-modeling-neural-networks-part2-attention-models/) from the original on 2020-10-21. Retrieved 2019-10-15.
38. Alammar, Jay. "The Illustrated Transformer" (http://jalammar.github.io/illustrated-transformer/). *jalammar.github.io*. Archived (https://web.archive.org/web/20201018061610/https://jalammar.github.io/illustrated-transformer/) from the original on 2020-10-18. Retrieved 2019-10-15.
39. Clark, Kevin; Khandelwal, Urvashi; Levy, Omer; Manning, Christopher D. (August 2019). "What Does BERT Look at? An Analysis of BERT's Attention" (https://www.aclweb.org/anthology/W19-4828). *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Florence, Italy: Association for Computational Linguistics: 276–286. arXiv:1906.04341 (https://arxiv.org/abs/1906.04341). doi:10.18653/v1/W19-4828 (https://doi.org/10.18653%2Fv1%2FW19-4828). Archived (https://web.archive.org/web/20201021211357/https://www.aclweb.org/anthology/W19-4828/) from the original on 2020-10-21. Retrieved 2020-05-20.
40. LeCun, Yann (Apr 28, 2023). "A survey of LLMs with a practical guide and evolutionary tree" (https://web.archive.org/web/20230623012310/https://twitter.com/ylecun/status/1651762787373428736?lang=en). *Twitter*. Archived from the original (https://twitter.com/ylecun/status/1651762787373428736?lang=en) on 23 Jun 2023. Retrieved 2023-06-23.
41. "Masked language modeling" (https://huggingface.co/docs/transformers/tasks/masked_language_modeling). *huggingface.co*. Retrieved 2023-10-05.
42. "Causal language modeling" (https://huggingface.co/docs/transformers/tasks/language_modeling). *huggingface.co*. Retrieved 2023-10-05.
43. Shazeer, Noam (2020-02-01). "GLU Variants Improve Transformer". arXiv:2002.05202 (https://arxiv.org/abs/2002.05202) [cs.LG (https://arxiv.org/archive/cs.LG)].
44. Dufter, Philipp; Schmitt, Martin; Schütze, Hinrich (2022-06-06). "Position Information in Transformers: An Overview" (https://doi.org/10.1162%2Fcoli_a_00445). *Computational Linguistics*. **48** (3): 733–763. arXiv:2102.11090 (https://arxiv.org/abs/2102.11090). doi:10.1162/coli_a_00445 (https://doi.org/10.1162%2Fcoli_a_00445). ISSN 0891-2017 (https://www.worldcat.org/issn/0891-2017). S2CID 231986066 (https://api.semanticscholar.org/CorpusID:231986066).
45. Su, Jianlin; Lu, Yu; Pan, Shengfeng; Murtadha, Ahmed; Wen, Bo; Liu, Yunfeng (2021-04-01). "RoFormer: Enhanced Transformer with Rotary Position Embedding". arXiv:2104.09864 (https://arxiv.org/abs/2104.09864) [cs.CL (https://arxiv.org/archive/cs.CL)].
46. Press, Ofir; Smith, Noah A.; Lewis, Mike (2021-08-01). "Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation". arXiv:2108.12409 (https://arxiv.org/abs/2108.12409) [cs.CL (https://arxiv.org/archive/cs.CL)].
47. Shaw, Peter; Uszkoreit, Jakob; Vaswani, Ashish (2018). "Self-Attention with Relative Position Representations". arXiv:1803.02155 (https://arxiv.org/abs/1803.02155) [cs.CL (https://arxiv.org/archive/cs.CL)].
48. Dao, Tri; Fu, Dan; Ermon, Stefano; Rudra, Atri; Ré, Christopher (2022-12-06). "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness" (https://proceedings.neurips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html). *Advances in Neural Information Processing Systems*. **35**: 16344–16359. arXiv:2205.14135 (https://arxiv.org/abs/2205.14135).
49. "Stanford CRFM" (https://crfm.stanford.edu/2023/07/17/flash2.html). *crfm.stanford.edu*. Retrieved 2023-07-18.
50. "FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning" (https://princeton-nlp.github.io/flash-atttention-2/). *Princeton NLP*. 2023-06-17. Retrieved 2023-07-18.

51. "Introducing Together AI Chief Scientist Tri Dao, as he releases FlashAttention-2 to speed up model training and inference" (https://together.ai/blog/tri-dao-flash-attention). *TOGETHER*. Retrieved 2023-07-18.
52. Chowdhery, Aakanksha; Narang, Sharan; Devlin, Jacob; Bosma, Maarten; Mishra, Gaurav; Roberts, Adam; Barham, Paul; Chung, Hyung Won; Sutton, Charles; Gehrmann, Sebastian; Schuh, Parker; Shi, Kensen; Tsvyashchenko, Sasha; Maynez, Joshua; Rao, Abhishek (2022-04-01). "PaLM: Scaling Language Modeling with Pathways". arXiv:2204.02311 (https://arxiv.org/abs/2204.02311) [cs.CL (https://arxiv.org/archive/cs.CL)].
53. Leviathan, Yaniv; Kalman, Matan; Matias, Yossi (2023-05-18), *Fast Inference from Transformers via Speculative Decoding*, arXiv:2211.17192 (https://arxiv.org/abs/2211.17192)
54. Fu, Yao (2023-12-13). "Towards 100x Speedup: Full Stack Transformer Inference Optimization" (https://yaofu.notion.site/Towards-100x-Speedup-Full-Stack-Transformer-Inference-Optimization-43124c3688e14cffaf2f1d6cbdf26c6c).
55. Chen, Charlie; Borgeaud, Sebastian; Irving, Geoffrey; Lespiau, Jean-Baptiste; Sifre, Laurent; Jumper, John (2023-02-02), *Accelerating Large Language Model Decoding with Speculative Sampling*, arXiv:2302.01318 (https://arxiv.org/abs/2302.01318)
56. Kitaev, Nikita; Kaiser, Łukasz; Levskaya, Anselm (2020). "Reformer: The Efficient Transformer". arXiv:2001.04451 (https://arxiv.org/abs/2001.04451) [cs.LG (https://arxiv.org/archive/cs.LG)].
57. "Constructing Transformers For Longer Sequences with Sparse Attention Methods" (https://ai.googleblog.com/2021/03/constructing-transformers-for-longer.html). *Google AI Blog*. 25 March 2021. Archived (https://web.archive.org/web/20210918150757/https://ai.googleblog.com/2021/03/constructing-transformers-for-longer.html) from the original on 2021-09-18. Retrieved 2021-05-28.
58. "Tasks with Long Sequences – Chatbot" (https://www.coursera.org/lecture/attention-models-in-nlp/tasks-with-long-sequences-suzNH). *Coursera*. Archived (https://web.archive.org/web/20201026130457/https://www.coursera.org/lecture/attention-models-in-nlp/tasks-with-long-sequences-suzNH) from the original on 2020-10-26. Retrieved 2020-10-22.
59. "Reformer: The Efficient Transformer" (http://ai.googleblog.com/2020/01/reformer-efficient-transformer.html). *Google AI Blog*. 16 January 2020. Archived (https://web.archive.org/web/20201022210019/https://ai.googleblog.com/2020/01/reformer-efficient-transformer.html) from the original on 2020-10-22. Retrieved 2020-10-22.
60. Zhai, Shuangfei; Talbott, Walter; Srivastava, Nitish; Huang, Chen; Goh, Hanlin; Zhang, Ruixiang; Susskind, Josh (2021-09-21). "An Attention Free Transformer". arXiv:2105.14103 (https://arxiv.org/abs/2105.14103) [cs.LG (https://arxiv.org/archive/cs.LG)].
61. Tay, Yi; Dehghani, Mostafa; Abnar, Samira; Shen, Yikang; Bahri, Dara; Pham, Philip; Rao, Jinfeng; Yang, Liu; Ruder, Sebastian; Metzler, Donald (2020-11-08). "Long Range Arena: A Benchmark for Efficient Transformers". arXiv:2011.04006 (https://arxiv.org/abs/2011.04006) [cs.LG (https://arxiv.org/archive/cs.LG)].
62. Peng, Hao; Pappas, Nikolaos; Yogatama, Dani; Schwartz, Roy; Smith, Noah A.; Kong, Lingpeng (2021-03-19). "Random Feature Attention". arXiv:2103.02143 (https://arxiv.org/abs/2103.02143) [cs.CL (https://arxiv.org/archive/cs.CL)].
63. Choromanski, Krzysztof; Likhosherstov, Valerii; Dohan, David; Song, Xingyou; Gane, Andreea; Sarlos, Tamas; Hawkins, Peter; Davis, Jared; Belanger, David; Colwell, Lucy; Weller, Adrian (2020-09-30). "Masked Language Modeling for Proteins via Linearly Scalable Long-Context Transformers". arXiv:2006.03555 (https://arxiv.org/abs/2006.03555) [cs.LG (https://arxiv.org/archive/cs.LG)].

64. Radford, Alec; Kim, Jong Wook; Xu, Tao; Brockman, Greg; McLeavey, Christine; Sutskever, Ilya (2022). "Robust Speech Recognition via Large-Scale Weak Supervision". arXiv:2212.04356 (https://arxiv.org/abs/2212.04356) [eess.AS (https://arxiv.org/archive/eess.AS)].
65. Jaegle, Andrew; Gimeno, Felix; Brock, Andrew; Zisserman, Andrew; Vinyals, Oriol; Carreira, Joao (2021-06-22). "Perceiver: General Perception with Iterative Attention". arXiv:2103.03206 (https://arxiv.org/abs/2103.03206) [cs.CV (https://arxiv.org/archive/cs.CV)].
66. Jaegle, Andrew; Borgeaud, Sebastian; Alayrac, Jean-Baptiste; Doersch, Carl; Ionescu, Catalin; Ding, David; Koppula, Skanda; Zoran, Daniel; Brock, Andrew; Shelhamer, Evan; Hénaff, Olivier (2021-08-02). "Perceiver IO: A General Architecture for Structured Inputs & Outputs". arXiv:2107.14795 (https://arxiv.org/abs/2107.14795) [cs.LG (https://arxiv.org/archive/cs.LG)].
67. Peebles, William; Xie, Saining (March 2, 2023). "Scalable Diffusion Models with Transformers". arXiv:2212.09748 (https://arxiv.org/abs/2212.09748) [cs.CV (https://arxiv.org/archive/cs.CV)].
68. "Google AI Unveils Muse, a New Text-to-Image Transformer Model" (https://www.infoq.com/news/2023/01/google-muse-text-to-image/). *InfoQ*.
69. "Using Diffusion Models to Create Superior NeRF Avatars" (https://blog.metaphysic.ai/muse-googles-super-fast-text-to-image-model-abandons-latent-diffusion-for-transformers/,%20https://blog.metaphysic.ai/muse-googles-super-fast-text-to-image-model-abandons-latent-diffusion-for-transformers/). January 5, 2023.
70. Islam, Arham (November 14, 2022). "How Do DALL·E 2, Stable Diffusion, and Midjourney Work?" (https://www.marktechpost.com/2022/11/14/how-do-dall%c2%b7e-2-stable-diffusion-and-midjourney-work/).

Further reading

- Alexander Rush, The Annotated transformer (https://nlp.seas.harvard.edu/2018/04/03/attention.html) Archived (https://web.archive.org/web/20210922093841/https://nlp.seas.harvard.edu/2018/04/03/attention.html) 2021-09-22 at the Wayback Machine, Harvard NLP group, 3 April 2018
- Phuong, Mary; Hutter, Marcus (2022), *Formal Algorithms for Transformers*, arXiv:2207.09238 (https://arxiv.org/abs/2207.09238)
- Ferrando, Javier; Sarti, Gabriele; Bisazza, Arianna; Costa-jussà, Marta R. (2024-05-01), *A Primer on the Inner Workings of Transformer-based Language Models* (http://arxiv.org/abs/2405.00208), doi:10.48550/arXiv.2405.00208 (https://doi.org/10.48550%2FarXiv.2405.00208), retrieved 2024-05-17
- Hubert Ramsauer *et al.* (2020), "Hopfield Networks is All You Need" (https://arxiv.org/abs/2008.02217) Archived (https://web.archive.org/web/20210918150812/https://arxiv.org/abs/2008.02217) 2021-09-18 at the Wayback Machine, preprint submitted for ICLR 2021. arXiv:2008.02217; see also authors' blog (https://ml-jku.github.io/hopfield-layers/) Archived (https://web.archive.org/web/20210918150757/https://ml-jku.github.io/hopfield-layers/) 2021-09-18 at the Wayback Machine

  – Discussion of the effect of a transformer layer as equivalent to a Hopfield update, bringing the input closer to one of the fixed points (representable patterns) of a continuous-valued Hopfield network

---