Attention is All You Need

Reviewed by Dong Kailun, Liu Haoxing, Niu Xin

July 07, 2025

Attention is All You Need (Vaswani et al., 2017).

1 Highlights

- Model Architecture of the **encoder-decoder** structure.
- Replace recurrent layers with **multi-headed self-attention**.
- **Enhanced parallelization** compared to RNN.
- Superior Handling of **long-range dependencies** compared to RNN and CNN (constant time complexity).
- **Training ease** compared to standard recurrent or convolution networks.
- Variations and improvements.

2 The Encoder-Decoder Framework

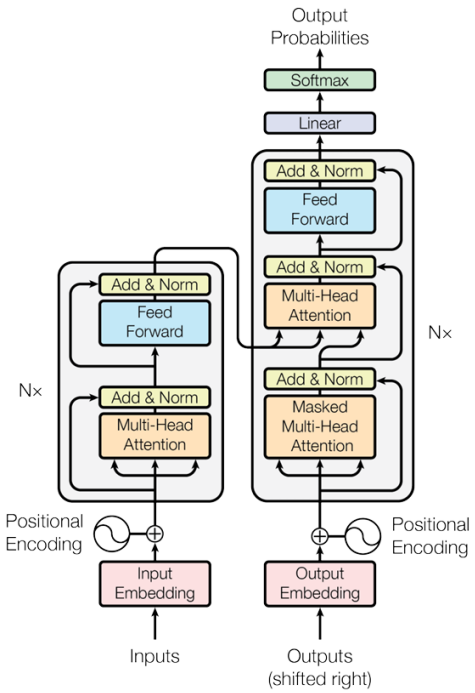

Figure 1: The Encoder-Decoder Architecture of Transformer

The Transformer model uses a proven encoder-decoder structure. However, it completely replaces recurrent layers with multi-head self-attention, enabling massive parallelization and a superior ability to handle long-range dependencies. To incorporate sequence order, positional encodings are added to the input embeddings at the very bottom of both stacks.

The model operates in two distinct phases, the encoding phase and the decoding phase. The encoder processes the entire input sentence at once. Each layer in the encoder stack uses a self-attention mechanism—where the Query (Q), Key (K), and Value (V) vectors are all derived from the same input—to build a context-aware representation for every word. The final output of the encoder is a rich set of Key and Value vectors. This set acts as a comprehensive, contextual memory of the source sentence, ready for the decoder to query.

The decoder generates the output sentence one word at a time in an auto-regressive manner. For each word it generates, the decoder performs three steps. The first step involves a masked self-attention layer that examines the words it has already generated, enabling it to comprehend the context of its partial output. The second step is the encoder-decoder Attention layer, which is the key translation step. The decoder's current output forms a Query (Q) that is sent to the encoder's final layer. It probes the Key (K) vectors from the encoder's memory to find the most relevant information. It then retrieves the corresponding Value (V) vectors to guide the prediction of the next word. The final step, similar to that in the encoder, is a final feed-forward network that processes this information to produce the actual output word.

To turn the decoder's final vector into a specific word, it's passed through a final Linear layer and then a Softmax function. This produces a probability score for every possible word in the model's vocabulary. A single word is then selected from this distribution (usually the one with the highest probability). It is this chosen word, not the probabilities, that is fed back into the decoder to begin the next generation step.

Throughout the model, each sub-layer (attention and feed-forward) is wrapped with a residual connection and layer normalization ($\text{LayerNorm}(x + \text{Sublayer}(x))$) to ensure stable and efficient training.

In the following sections, we will discuss in more detail the functionality and advantages of each method employed in this framework.

3 Scaled Dot-Product Attention

The attention function, as mentioned in the previous section, is a mapping between a query (Q) and a set of key-value pairs (K, V). All queries, keys, and values are vector-valued, and the output, the final set of values, will be weighted sums of the values based on the query and corresponding key compatibility. The particular self-attention function used in this article is the scaled dot-product attention.

Imagine we have an input sentence, and an embedding vector represents each word in it. The overall embedding matrix X consists of every single word's embedding vector (x). One way for the model to compute Q, K, and V is to learn a "weight matrix" for each of these components, W^Q, W^K and W^V so that the corresponding values for x is computed as $Q(x) = W^Q x$, $K(x) = W^K x$ and $V(x) = W^V x$. The overall Q, K, and V matrices are the results of these operations applied to the entire input sentence X as $Q = W^Q X$, $K = W^K X$, and $V = W^V X$.

Given matrices Q, K, and V, the scaled dot-product attention is defined as:

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

The input consists of queries and keys of dimension d_k, and values of dimension d_v. We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values.

- $QK^\top$: This is a dot product between the Query matrix and the transposed Key matrix. For a single word's query vector (q), it's essentially calculating a dot product with every other word's key vector (k). A higher dot product means the key is more relevant or "answers" the query better. This step produces the raw attention scores.

- $\frac{...}{\sqrt{d_k}}$: The scores are scaled down by the square root of the dimension of the key vectors.

- softmax(...): The softmax function is applied to the scaled scores, turning them into positive weights that all sum to 1. These weights represent how much attention each word should pay to every other word.

- $...V$: Finally, these softmax weights are multiplied by the Value matrix. This step effectively amplifies the Value vectors of the words that are most relevant (have high attention weights) and drowns out the irrelevant ones.

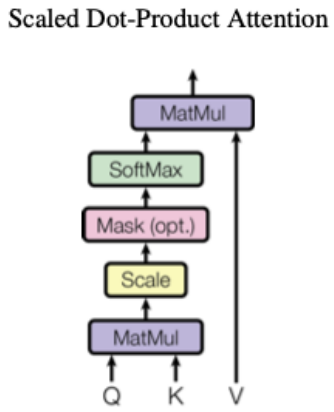

Figure 2: Scaled Dot-Product Attention

3.1 The use for dividing by $\sqrt{d_k}$

1. **Preventing large dot product values:**
   When the dimensionality d_k is large, the dot products between queries and keys tend to have large magnitudes. For example, if q and k are standard normally distributed vectors, their dot product has a variance proportional to d_k. Without scaling, the values of $QK^\top$ become very large.

   Feeding these large values into the softmax function results in very sharp distributions, where one element is close to 1 and the rest are near 0.

2. **Gradient vanishing in softmax:**
   When the input to softmax has large magnitudes, it pushes the function into regions where gradients become very small, making learning difficult due to vanishing gradients.

3.2 Softmax Function

Given a vector:

$$\mathbf{z} = [z_1, z_2, \ldots, z_n]$$

The softmax function is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \quad \text{for } i = 1, \ldots, n$$

Properties:

- All outputs are non-negative: $\text{softmax}(z_i) \geq 0$

- All outputs sum to 1: $\sum_{i=1}^{n} \text{softmax}(z_i) = 1$

Purpose of Softmax:

- Converts any real-valued vector into a probability distribution;
- Larger input values correspond to higher probabilities (closer to 1);
- Smaller input values correspond to lower probabilities (closer to 0).

4 Multi-Head Attention

In the context of self-attention, the calculation is similar to finding the answer to the specific question posted by the set of Q, K, and V. In multi-head attention, each "head" is like a parallel line of inquiry. Each head has its own learned weight matrices and specializes in capturing a different type of contextual relationship.

The idea is to project these queries, keys, and values h times through separate learned linear transformations into lower-dimensional subspaces. Specifically, the queries, keys, and values are linearly mapped to d_k, d_k, and d_v dimensions. Each head then independently performs scaled dot-product attention:

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

After computing attention outputs for each head (each of size d_v), the results are concatenated to form a single vector of size $h \cdot d_v$, and this is linearly transformed once more with a projection matrix $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ to return to the original model dimension. This process is summarized as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \quad \text{where} \quad \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

In typical Transformer implementations, the number of heads h is set to 8, and $d_k = d_v = d_{\text{model}}/h$. For example, in models like BERT or the original Transformer, the model dimension is typically $d_{\text{model}} = 512$. If the number of attention heads is set to $h = 8$, then each head operates on vectors of size $d_k = d_v = 64$, since $512/8 = 64$. The outputs of all 8 heads are then concatenated to form a single vector of dimension $8 \times 64 = 512$, which matches the original d_{model}.

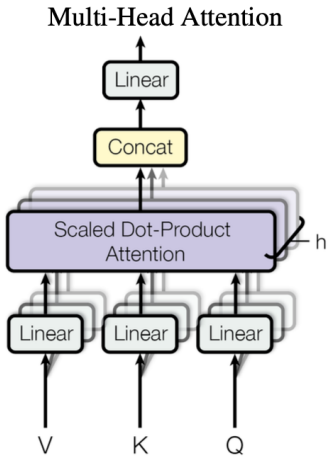

Figure 3: Multi-Head Attention

This design offers two main advantages:

- It allows the model to attend to information from multiple representation subspaces at different positions in parallel, enhancing its ability to capture diverse patterns in the input.
- Since each head works on a lower-dimensional space, the total computational cost remains approximately the same as using a single full-dimensional attention mechanism.

5 Batch Norm vs. Layer Norm

2D input: Batch Norm normalize each column (feature); Layer Norm normalize each row of matrix (sample)

For 3D input ($\mathbf{T} \in \mathbb{R}^{n \times m \times k}$, k: number of features), each sample is the embeddings of a sequence of sentence. Batch Norm normalize each Normalize each page ($T_i \in \mathbb{R}^{m \times n}$); Layer Norm normalize each layer ($T_j \in \mathbb{R}^{n \times k}$)

Why Layer Norm is better than batch norm (for transformer)?

- It does not rely on batch statistics.
- Works well with variable-length sequences.
- It provides consistent behavior across training and inference.
- It can be applied before or after a sublayer, helps with training stability and deeper models

6 positional encoding

"Positional encodings" are added to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. For any fixed offset k, PE_{pos+k} can be represented as a linear function of $PEpos$. It may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

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

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

pos is the position and i is the dimension.

Other paper on positional encoding: [1]

7 Advantages of Self-Attention

7.1 Computational complexity

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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

Figure 4: Complexity Table

The complexity discussion can be separated into three categories: Complexity per layer, Sequential Operations, and Maximum Path Length.

Complexity per layer refers to the amount of computation required to process a sequence in one layer. In Self-Attention, the n^2 comes from the fact that every word needs to calculate an attention score with every other word in the sequence. This can be reduced if we restrict the attention to a smaller neighborhood. In the other two types, the complexity is linear with respect to length n, as they traverse through the sequence once. The main computational burden lies on the model operations on the embeddings d^2. The problem is more model-specific than input-specific.

The sequential operations measure the number of operations that must be performed in order, which is the primary bottleneck for parallelization. For the self-attention transformer, since the model can see the entire sequence at once, the core attention calculation is just a set of matrix multiplications that can be executed in a single, constant-time operation. This enables massive parallelization, significantly reducing training time on suitable hardware. Similarly, the CNN can apply its convolution filters simultaneously. However, an RNN is defined by its sequential nature. To process the n-th word, it must first have the result from the previous word. This creates a computational chain of length n that cannot start from the middle.

The maximum Path Length measures the maximum distance information has to travel to get from any one point in the sequence to another. Shorter paths make it easier to learn long-range dependencies. The self-attention mechanism is designed to address this problem, creating a direct, weighted connection between every pair of words in the sequence. The path length is always a single step. If we restrict the neighborhood to a size of r, then at each step, we can access at most a distance of r apart, giving a total of n/r steps to access the farthest point in the sequence. In RNN, for information to travel from the first word to the n-th word, it must pass through the hidden state of every single intermediate word. The path length is therefore linear with the distance n. In a standard CNN, a single convolution layer can only connect words within its kernel size k. To connect words that are far apart, information must be passed through a stack of multiple convolution layers. The path length is therefore proportional to the distance divided by the kernel size, n/k. For dilated convolutions, where the spacing between the kernel elements increases exponentially with each layer, the number of layers needed satisfies $k^L = n$, giving $L = \log_k(n)$.

7.2 Enhanced Parallelization

By dispensing with the sequential nature of RNNs, the Transformer architecture allows for significantly more parallelization during training. This results in faster training times and the ability to train on significantly larger datasets.

7.3 Long-range dependencies

Self-attention mechanisms enable the model to directly connect words across long distances in a sequence, overcoming the limitations of RNNs and CNNs in capturing long-range dependencies. The path length for relating any two positions in the sequence is constant.

7.4 Training ease

The combination of parallelization and a more direct way of learning dependencies often results in a more straightforward and quicker training process compared to recurrent or convolutional networks to achieve state-of-the-art results.

8 Variations and Improvements

References

[1] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.