Attention Is All You Need

- [[📚 Attention Is All You Need]]
- **Basic Idea**
  - Can we parallelize the training of encoder-decoder models for sequence transduction tasks, because RNN based models are sequential in processing the tokens of the sequences?
  - Can we reduce the number of operations required to relate any two arbitrary tokens in the input or output sequence? In RNNs, it is O(distance between the two tokens). The memory vector in the LSTM or GRU is fixed dimensional. So, if the sequence length increases or dependencies length increases, then the same problem that is faced in encoder-decoder model of fixed length content history vector not able to capture everything, will become evident and hence, we need self-attention.

- **Model Architecture**
  - **Notations**
    - $(x_1, ..., x_n)$: input sequence token
    - $(e_1, ..., e_n)$: embedding of the input sequence (includes normal embedding + positional embedding)
    - $(z_1, ..., z_n)$: output of encoder
    - $(h_1, ..., h_n)$: hidden state of the decoder
    - $(y_1, ..., y_m)$: output sequence
    - $(g_1, ..., g_m)$: embedding of the output sequence (includes normal embedding + positional embedding)
  - **Architecture**

    🟡 **P3**

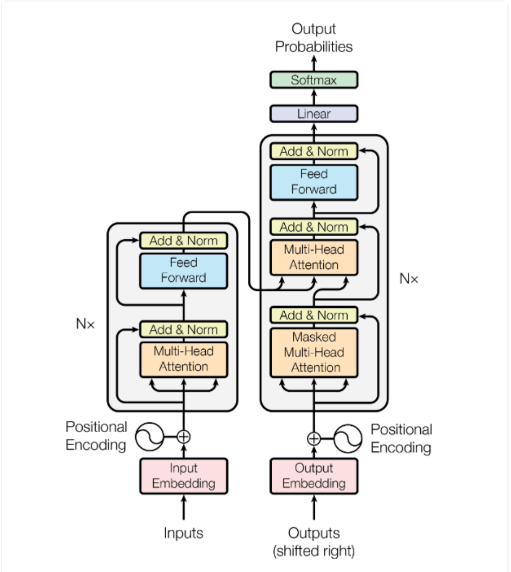


Figure 1: The Transformer - model architecture.

Encoder

🟡 **P3** Encoder: The encoder is composed of a stack of N = 6 identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, positionwise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension dmodel = 512.

Decoder

🟡 **P3** Decoder: The decoder is also composed of a stack of N = 6 identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

The masking is important in decoder because while generating sequence we cannot look into the future. 🟡 **P5** We implement this inside of scaled dot-product attention by masking out (setting to −∞) all values in the input of the softmax which correspond to illegal connections. Also, because the model is an autoregressive model i.e., the last predicted token is fed into the model again, the authors shift the output sequence by 1.

- **Scaled-Dot-Product Attention**
  Let's consider the case when the attention is between the encoder output and decoder. In this case:
  - **Query** $Q \in \mathbb{R}^{d_k}$: This is some function of h. For example, in ConvS2S, it is $Wh_i + b + g_i$
  - **Key** $K \in \mathbb{R}^{d_k}$: This is some function of z. For example, in ConvS2S, it is z_j.
  - **Value** $V \in \mathbb{R}^{d_v}$: This is some function of z. For example, in ConvS2S, it is $z_j + e_j$.
  - **Output** O: The output vector which is the weighted sum of the values.
    The general equation of attention from ConvS2S can be written as:

$$softmax(align(z_j, (Wh_i + b + g_i))(z_j + e_j)$$

where align is some function used to align the keys to values. This can be:
- **Additive** : When a MLP is used to align. (Bahdanau et al. Attention)
- **Dot-Product** : When dot product is used to align.

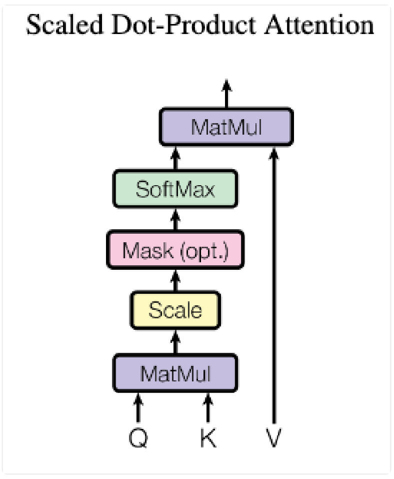
🟡 **P4** While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code. While for small values of dk the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of dk
For more variations to this align function, visit Effective Approaches to Attention-based Neural Machine Translation.

This is exactly the same attention formula that is used in Transformers (except for the scale factor $\sqrt{d_k}$):

$$softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

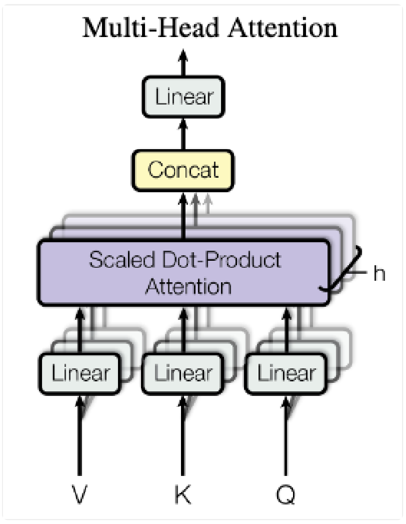
🟡 **P4** We suspect that for large values of dk, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients 4. To counteract this effect, we scale the dot products by 1√dk.
🟡 **P4** To illustrate why the dot products get large, assume that the components of q and k are independent random variables with mean 0 and variance 1. Then their dot product, q · k = ∑dk i=1 qiki, has mean 0 and variance dk .

🟡 **P4**



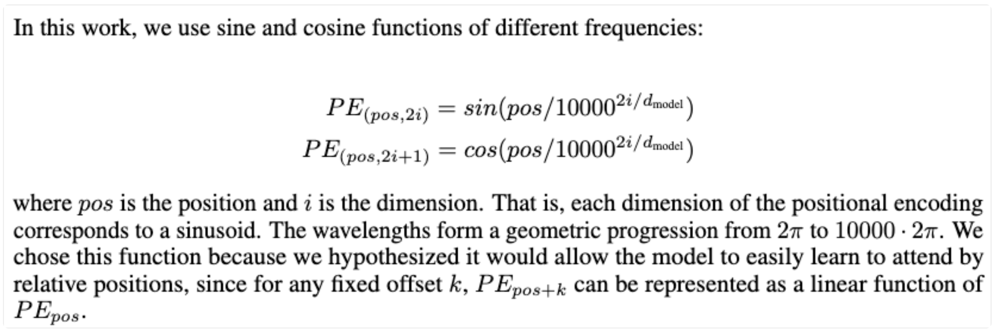
- **Multi-Head Attention**
  Basically, the same computation as above but the input is transformed (projected) by different projection matrices. Intuition: 🟡 **P5** Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

🟡 **P4**



- **Positional Embedding**

🟡 **P6**



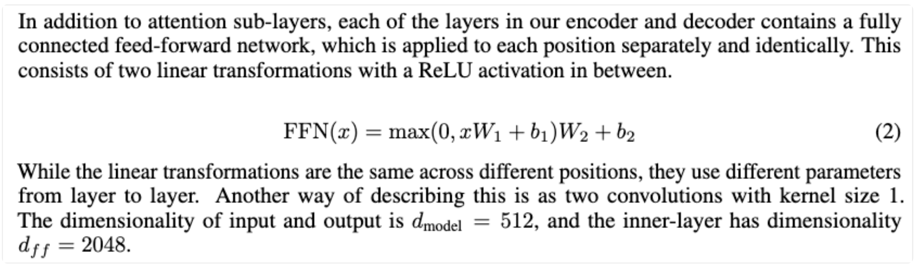
In this work, we use sine and cosine functions of different frequencies:

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

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

- **Position-wise Feed-Forward Networks**

P5


In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (2)$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

- **Self-Attention**

P6

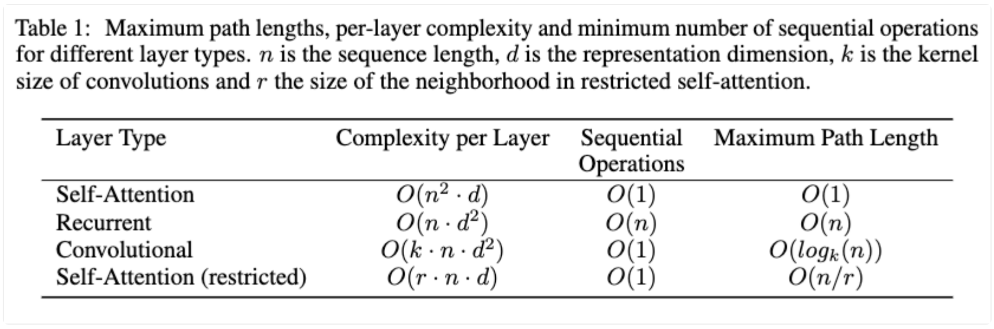


Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

The computations of the convolutional versions can be improved by using atrous (dilated) convolutions or separable convolutions. The computations for the convolution model is based on a stride of k. The computations of the self-attention, can be done by using Eq. 1 of the paper assuming $Q, K, V \in \mathbb{R}^{n \times d}$.

P6 To improve computational performance for tasks involving very long sequences, self-attention could be restricted to considering only a neighborhood of size r in6 the input sequence centered around the respective output position. This would increase the maximum path length to O(n/r).

P7 A single convolutional layer with kernel width k < n does not connect all pairs of input and output positions. Doing so requires a stack of O(n/k) convolutional layers in the case of contiguous kernels, or O(logk(n)) in the case of dilated convolutions [18], increasing the length of the longest paths between any two positions in the network. Convolutional layers are generally more expensive than recurrent layers, by a factor of k. Separable convolutions [6], however, decrease the complexity considerably, to O(k · n · d + n · d2). Even with k = n, however, the complexity of a separable convolution is equal to the combination of a self-attention layer and a point-wise feed-forward layer, the approach we take in our model.

Unlinked References