Attention Is All You Need

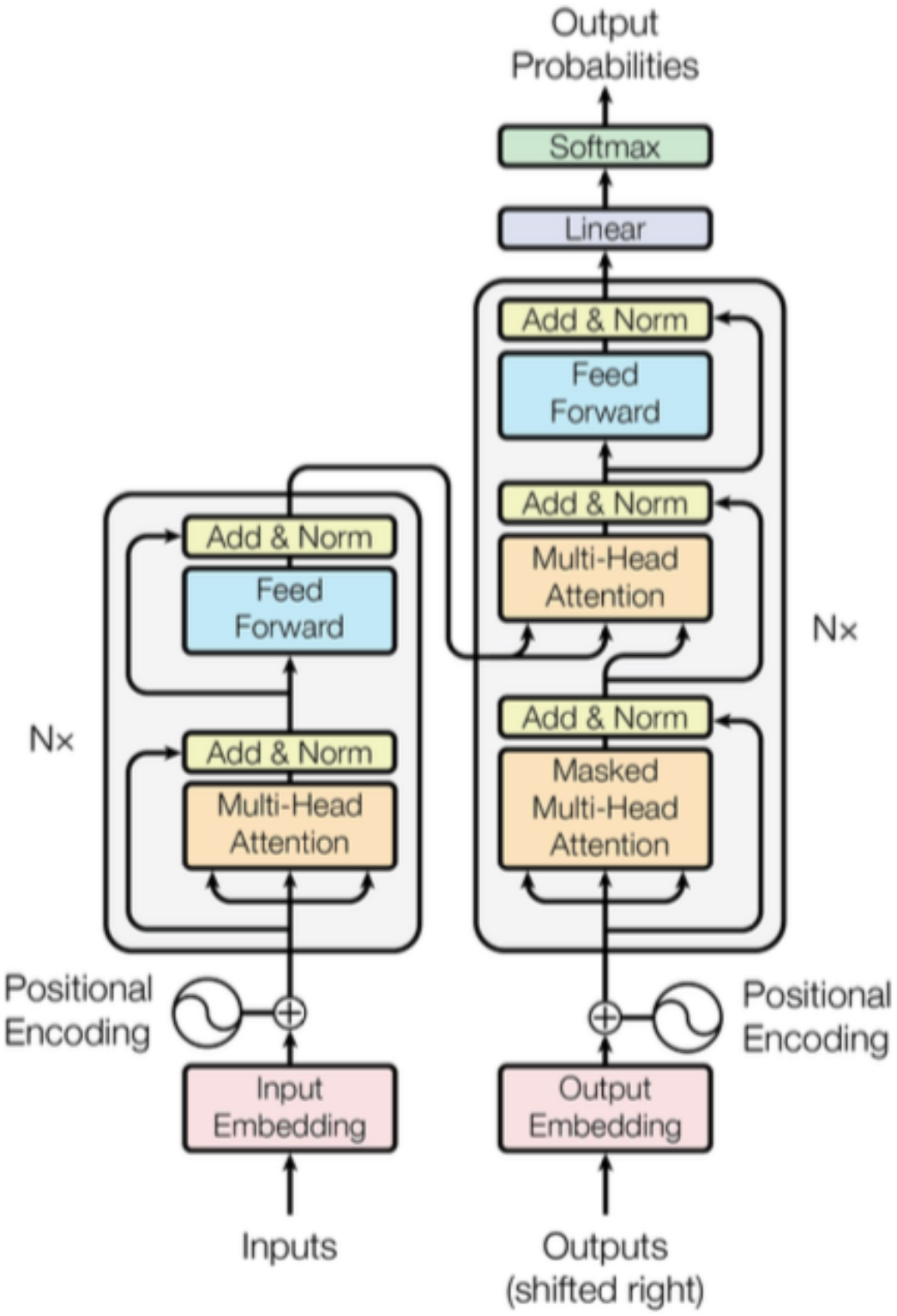


Figure 1: The Transformer - model architecture.

transformer包含两块内容：encoder和decoder

一、multi-head attention

1. scaled dot-product attention

Q、K、V的shape为：[N, time_step, hidden_size]。

Q和K经过矩阵乘法

scale（因为乘完后怕值太大，从而影响从softmax回传的梯度）

mask（主要是防止句子中当前词右侧的内容对当前词的影响，使用简单的0/1mask，主要用在self-attention上）

具体公式如下：

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

2. multi-head

先将V、K、Q切分，转变为[N, h, time_step, hidden_size/h]或[N*h, time_step, hidden_size/h]的形式，然后再进入scaled dot-product attention处理，得到的结果再concat，之后通过一个linear层。公式如下：

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{\text{model}}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

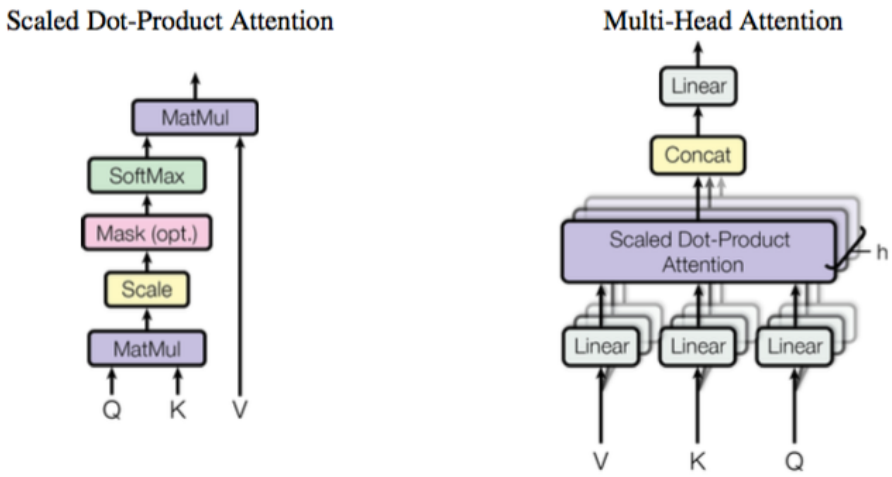


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

3. 三种attention
    1. encoder-decoder attention：使用multi-head attention，输入为encoder的输出和decoder的self-attention输出，其中encoder的self-attention作为key and value，decoder的self-attention作为query
    2. encoder self-attention：使用multi-head attention，输入的Q、K、V都是一样的（input embedding and positional embedding）
    3. decoder self-attention：使用masked multi-head attention，输入的Q、K、V都是一样的（output embedding and positional embedding）

4. 使用self-attention的好处

主要考虑计算复杂度、并行性、长依赖路径长度（比如某句话的第一个词和译文最后一个词的信息传递路径）。

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

restricted self-attention表示将一句话切分成长度为r的n/r块，再在每块里做attention。

二、FFN

用了两层Dense层，activation用的都是Relu。可以看成是两层的1*1的1d-convolution。hidden_size变化为：512->2048->512

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

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 $d_{ff} = 2048$.

三、embedding and softmax

1. embedding：input embedding、positional embedding和softmax前的linear层共享一个weight matrix，每个embedding layer的输出均乘上sqrt(hidden_size)

四、positional embedding

文中说训练的positional embedding与直接使用固定的正弦曲线的效果一致，并且如果直接用固定的位置信息（词的位置从1到time_step），也是一样的。下面主要讲一下使用正弦曲线作为positional embedding。

考虑词序维度(pos)和embedding维度(i)，如果embedding维度位置为偶数，则用sin，否则用cos。pos决定了相位，而i决定了周期/频率。每个位置的embedding值都可以由其余位置线性表出(比如sin(a+b)=sin(a)cos(b)+cos(a)sin(b))。

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 $2\pi$ 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 $PE_{pos}$.

We also experimented with using learned positional embeddings [9] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

五、optimizer

We used the Adam optimizer [19] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \tag{3}$$

This corresponds to increasing the learning rate linearly for the first $warmup\_steps$ training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used $warmup\_steps = 4000$.

六、regularization

用了三个方法：

1. residual dropout

**Residual Dropout** We apply dropout [32] to the output of each sub-layer, before it is added to the sub-layer input and normalized. In addition, we apply dropout to the sums of the embeddings and the positional encodings in both the encoder and decoder stacks. For the base model, we use a rate of $P_{drop} = 0.1$.

2. attention dropout

**Attention Dropout** Query to key attentions are structurally similar to hidden-to-hidden weights in a feed-forward network, albeit across positions. The softmax activations yielding attention weights can then be seen as the analogue of hidden layer activations. A natural possibility is to extend dropout [32] to attention. We implement attention dropout by dropping out attention weights as,

$$\text{Attention}(Q, K, V) = \text{dropout}(\text{softmax}(\frac{QK^T}{\sqrt{d}}))V$$

In addition to residual dropout, we found attention dropout to be beneficial for our parsing experiments.

3. label smoothing

**Label Smoothing** During training, we employed label smoothing of value $\epsilon_{ls} = 0.1$ [34]. This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.