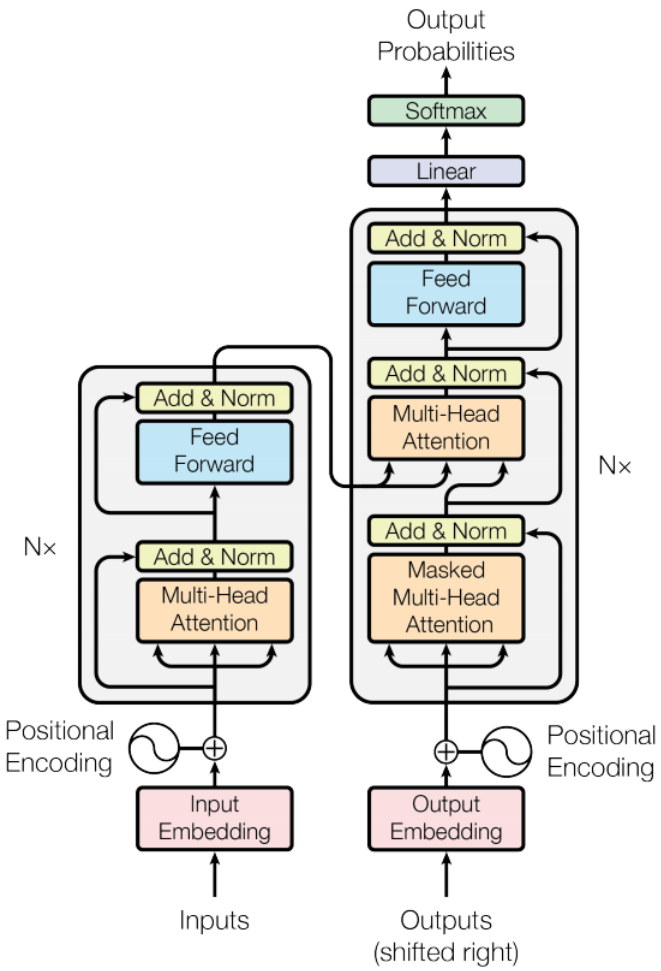
Transformer

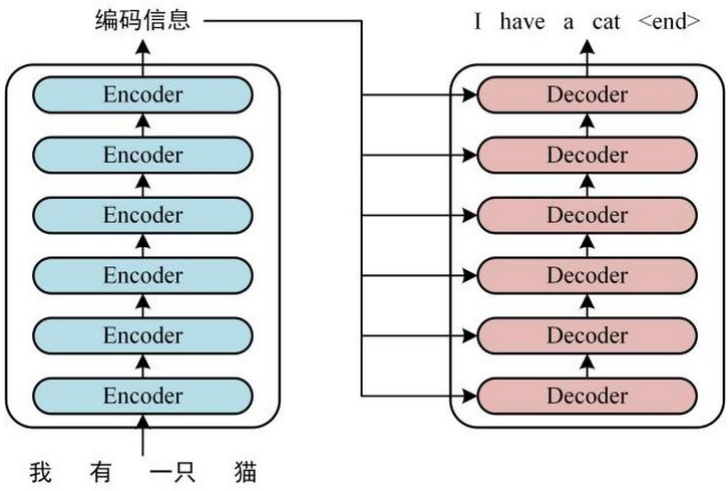
2022-01-30

1. 总体架构

模型分为Encoder和Decoder两个部分，下图为模型的架构图：

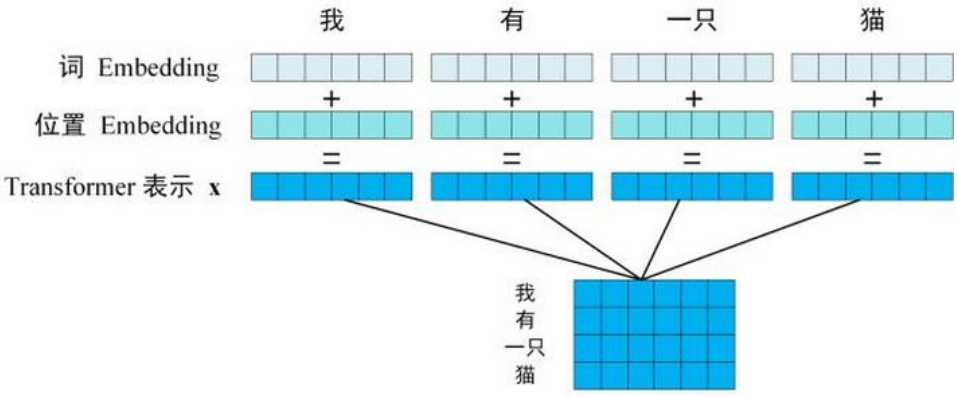


下面，以翻译为例，介绍Transformer的工作过程：



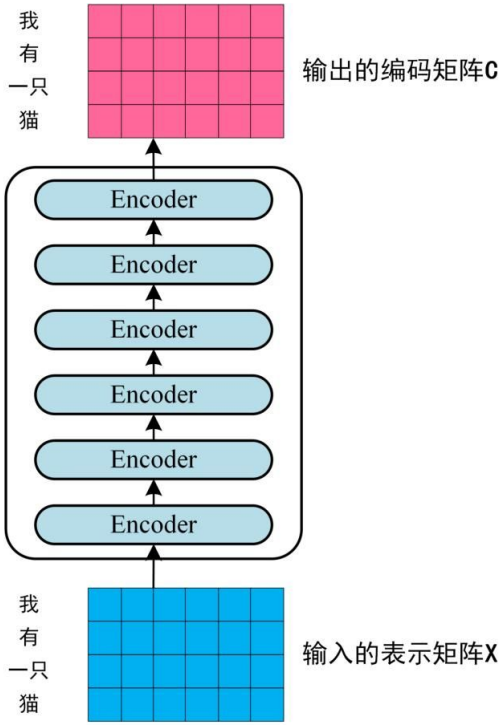
1.1. 输入

通过Input Embedding和Positional Encoding，将句子表示为编码，记为X。



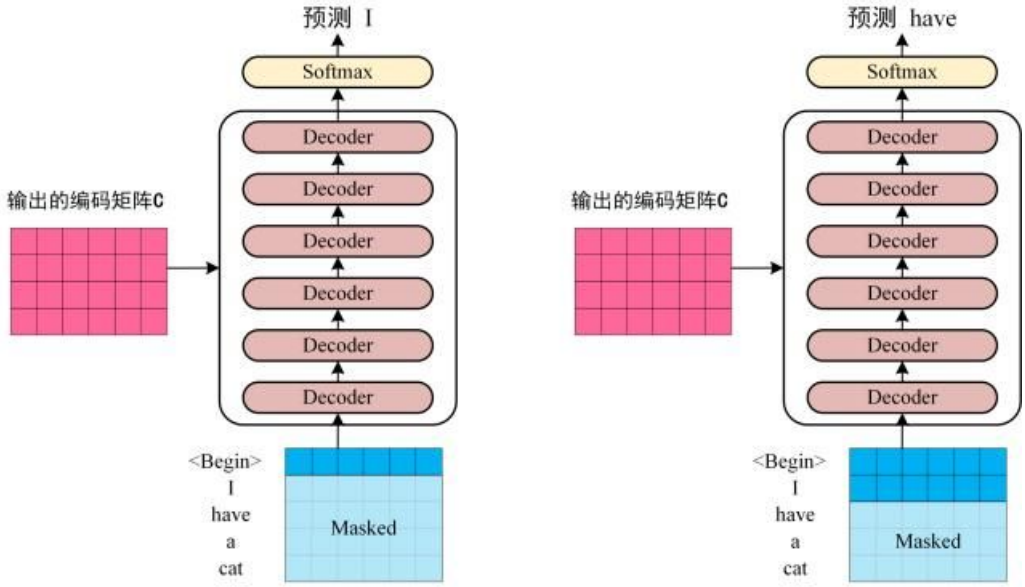
1.2. Encoder

对X，经过Encoder处理，得到单词的编码信息矩阵，记作C。该矩阵的维度与输入矩阵X完全一致。



1.3. Decoder

将Encoder输出的编码信息矩阵C传入Decoder中，该矩阵提供Decoder Attention中的K、V。Decoder会根据前i个单词翻译第i+1个单词，翻译时为了防止信息泄漏，需要进行mask操作。



2. 模型须知

2.1. 模型参数

1. batch_size

   批量大小

2. src_len

   Encoder端句子的最大长度

3. tgt_len

   Decoder端句子的最大长度

4. d_model

   词的Embedding Size

5. d_k (d_q)

   矩阵Q、K的列数

6. d_v

   矩阵V的列数

7. d_ff

   Feed Forward的隐藏层个数

8. src_vocab_size

   源端单词个数

9. tgt_vocab_size

   目标端单词个数

2.2. 输入与输出

```
enc_input           dec_input            dec_output
['我 是 中国人 P',  'S i am chinese P',   'i am chinese P E'],
['我 有 一只 猫' ,  'S i have a cat',     'i have a cat E']
```

1. enc_input

   encoder input

2. dec_input

   decoder input

3. dec_output

   decoder output，相当于y_true

2.3. 注意

对于变量符号，为了方便起见，有重名的情况。比如在Encoder和Decoder中，Attention都有可能使用Q符号。因此，变量符号需要根据所在章节判断。

3. Encoder

3.1. 输入

3.1.1. Input Embedding

使用word2vec，对X进行词向量编码，得到X_{word_emb}。

X_{word_emb}的shape为[batch_size, src_len, d_model]。

3.1.2. Positional Encoding

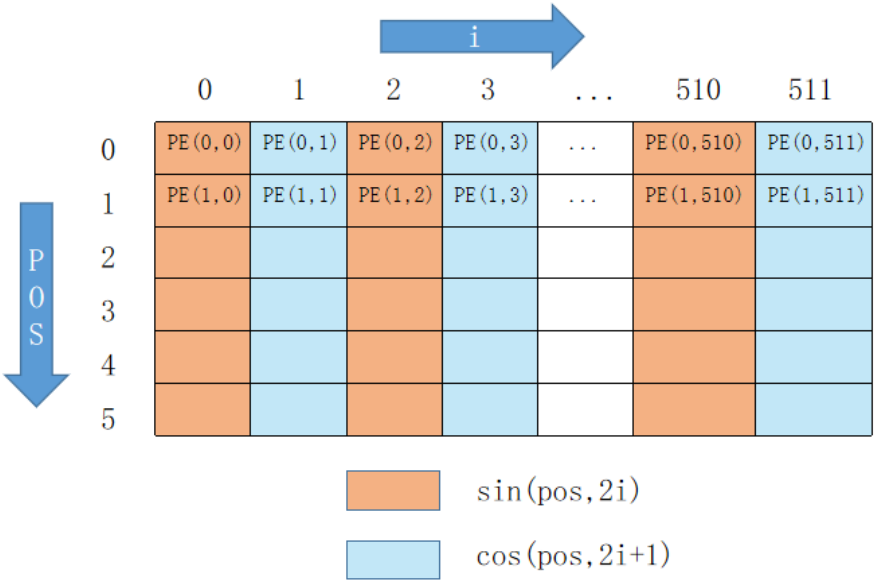
$$
\begin{aligned}
PE(pos, 2i) &= \sin\left(pos / 10000^{2i/d_{\text{modal}}}\right) \\
PE(pos, 2i+1) &= \cos\left(pos / 10000^{2i/d_{\text{model}}}\right)
\end{aligned} \tag{5}
$$

其中，pos为一句话中某词的位置，取值范围为$[0, sequence_length)$

i指词向量的维度序号，取值范围为$[0, embedding_dimension/2)$

因此，Positional Encoding结果的shape为[batch_size, src_len, d_model]。

记Positional Encoding的结果为$\mathrm{X}_{pos_encoding}$，下图为Positional Encoding的示例图：



3.1.3. 输入整合

$$\mathrm{X} = \mathrm{X}_{embedding} = \mathrm{X}_{word\_emb} + \mathrm{X}_{pos\_encoding}$$

shape为[batch_size, src_len, d_model]。

3.2. Attention

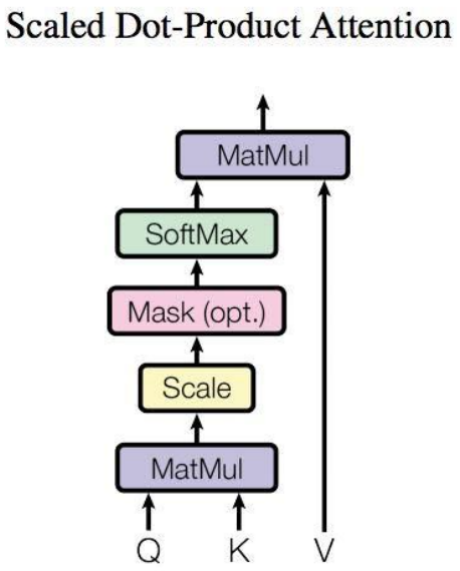
Encoder的Multi-Head Attention由多个Self-Attention组成，Self-Attention接收的是输入(单词的表示向量x组成的矩阵X)或者上一个Encoder block的输出，下面将对Self-Attention进行介绍。

3.2.1. Self-Attention

序列X与自己进行注意力计算，即序列X同时提供Q、K、V。计算方式如下所示：

$$\begin{gathered}
\mathrm{Q} = \mathrm{X}\,\mathrm{W}^{\mathrm{Q}} \\
\mathrm{K} = \mathrm{X}\,\mathrm{W}^{\mathrm{K}} \\
\mathrm{V} = \mathrm{X}\,\mathrm{W}^{\mathrm{V}} \\
\text{其中，}\ \mathrm{W}^{\mathrm{Q}}\text{、}\mathrm{W}^{\mathrm{K}}\text{、}\mathrm{W}^{\mathrm{V}}\ \text{为可学习的参数矩阵} \\
\mathrm{Q}\ \text{的}shape\text{为}[batch_size, src_len, d_k] \\
\mathrm{K}\ \text{的}shape\text{为}[batch_size, src_len, d_k] \\
\mathrm{V}\ \text{的}shape\text{为}[batch_size, src_len, d_v]
\end{gathered} \tag{6}$$

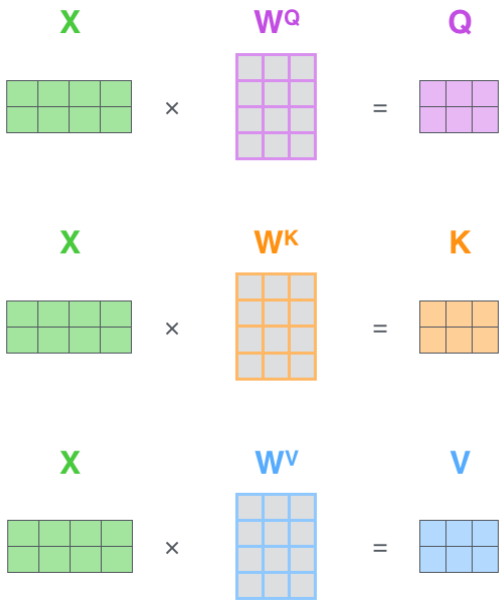
整体计算流程如下：



Self-Attention 结构

简而言之，已知X和变换矩阵W^{Q}、W^{K}、W^{V}，经过线性变换可得Q、K、V，然后再基于Q、K、V，经过点积计算和softmax计算，得到最终的输出Z。

Q、K、V生成过程：

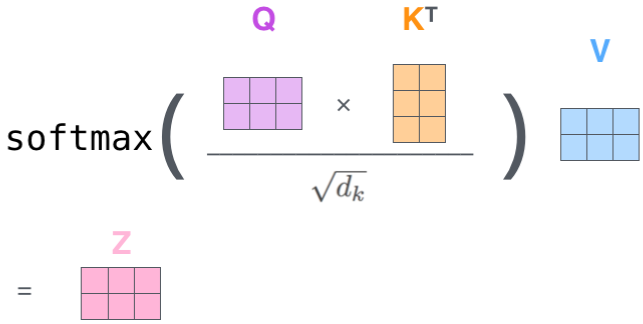


得到Q、K、V之后，就可以计算出Self-Attention的输出了。计算公式如下：

$$\text{Attention}\left(\mathrm{Q},\mathrm{K},\mathrm{V}\right)=\text{softmax}\left(\frac{\mathrm{QK}^{\mathrm{T}}}{\sqrt{d_k}}\right)\mathrm{V} \tag{7}$$

其中，d_k是矩阵Q,K的列数，即向量维度

最终结果生成过程：



其中，d_k为Q、K矩阵的列数，即向量维度，除以$\sqrt{d_k}$可以起到一定的归一化作用。

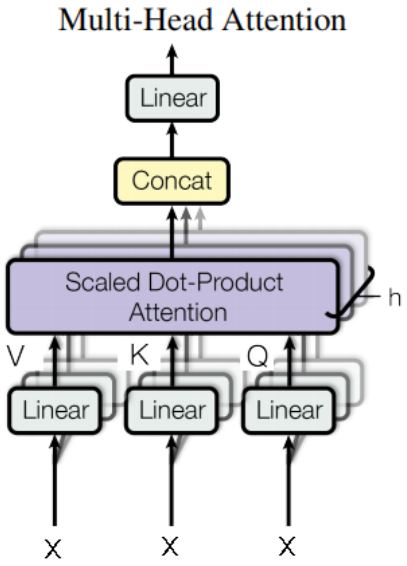
需要注意的是，在上述Self-Attention的计算过程中，通常基于mini-batch来进行计算，也就是一次计算多个句子。而一个mini-batch是由多个不等长的句子组成的，我们需要按照mini-batch中最大的句长对剩余的句子进行补齐，比如使用P作为填充字符，这个过程叫做padding。

对$\mathrm{Q}\,\mathrm{K}^{\mathrm{T}}$的结果进行mask，即将padding位置的结果置为-inf，那么，再进行softmax计算时便不受影响。

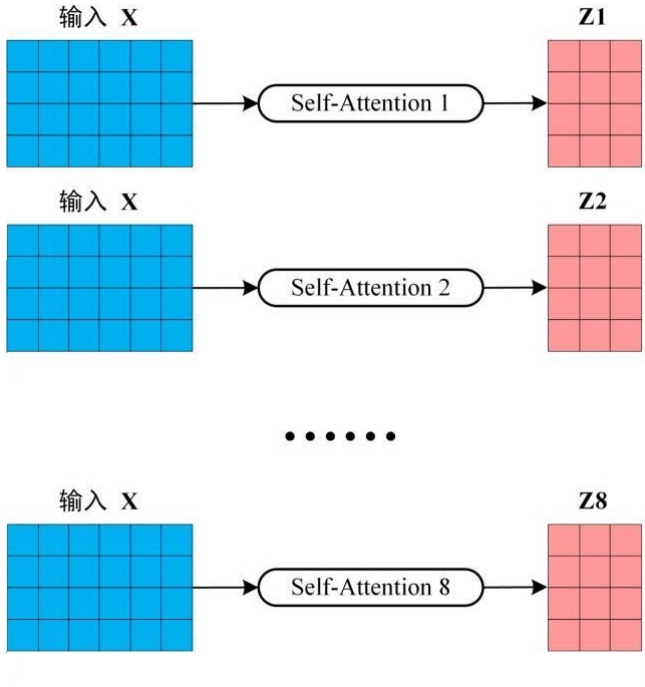
3.2.2. Multi-head Attention

多个Self-Attention的组合，即定义多组Q、K、V，每个Self-Attention生成一个输出，将多个输出进行Concat操作后，通过一次Linear变换，得到最终输出。

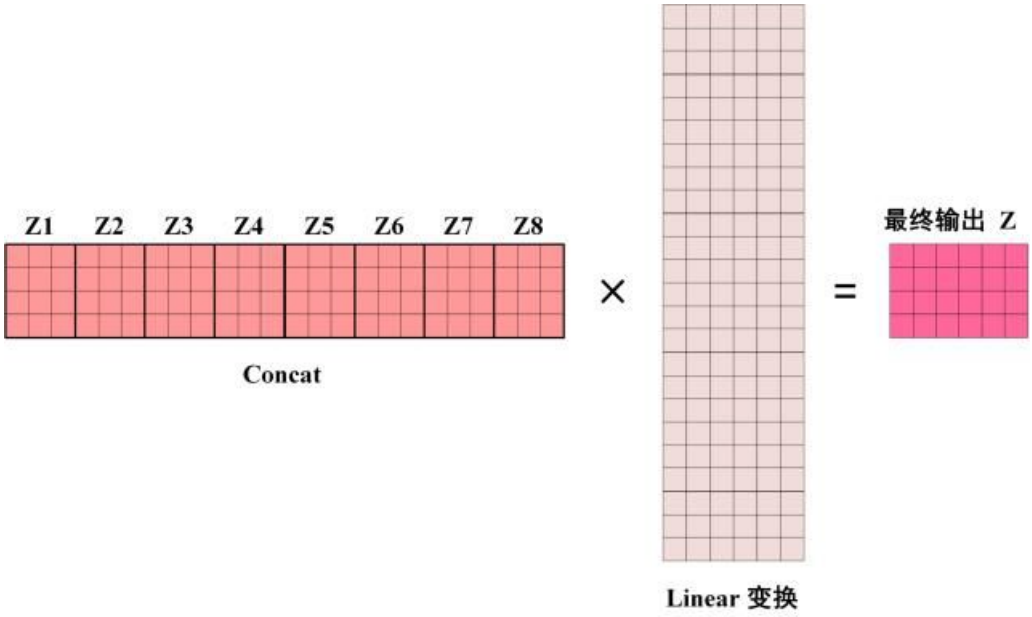
Multi-Head Attention架构图如下：



首先将输入X进行线性变换得到Q、K、V，然后分别传递到 h 个不同的 Self-Attention 中，计算得到 h 个输出矩阵Z。下图是 h=8 时候的情况，此时会得到 8 个输出矩阵Z。



然后，将8 个输出矩阵Z进行Concat后得到一个矩阵，再将该矩阵进行线性变换后，得到最终输出。如下图所示：



可以看到 Multi-Head Attention 输出的矩阵Z与其输入的矩阵X的维度是一样的。

3.2.3. add

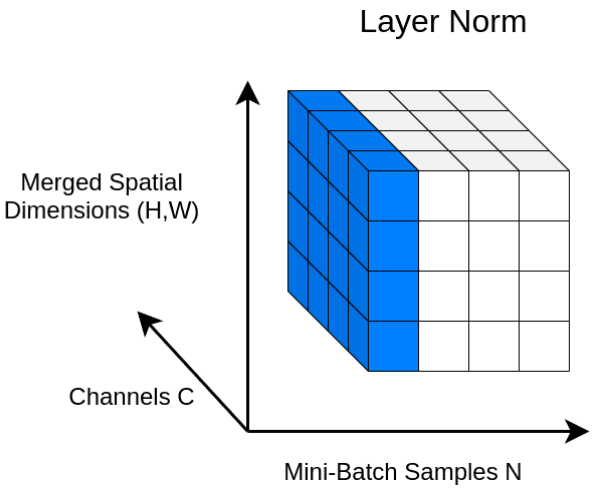
残差连接。

$$\mathrm{X} = \mathrm{X} + \mathrm{Multi\text{-}headAttention}(\mathrm{Q}, \mathrm{K}, \mathrm{V})$$

3.2.4. norm

对X进行Layer Normlization处理，即$\mathrm{X} = \mathrm{Layer\ Normalization}(\mathrm{X})$。

其过程如下图所示，其中，N为batch_size大小，C相当于词的个数，Merged Spatial Dimensions相当于词向量。



3.3. 线性层

3.3.1. Feed Forward

两层的线性变换，第一层使用ReLU作为激活函数，第二层不使用激活函数。对应公式如下：

$$\mathrm{X} = \mathrm{Linear}(\mathrm{ReLU}(\mathrm{Linear}(\mathrm{X}))) \tag{8}$$

线性变换后，再经过残差连接和Layer Norm处理，得到最终的输出，且维度与输入X的维度一致。

3.3.2. add

对线性变化的结果进行残差连接。

$$\mathrm{X} = \mathrm{X} + \mathrm{Linear}(\mathrm{ReLU}(\mathrm{Linear}(\mathrm{X})))$$

3.3.3. norm

对X进行Layer Normlization处理。

$$\mathrm{X} = \mathrm{Layer\ Normalization}(\mathrm{X})$$

3.4. Encoder block总结

通过上面描述的 Multi-Head Attention,、Feed Forward、 Add & Norm 操作可以构造出一个 Encoder block。Encoder block 接收输入矩阵$\mathrm{X}_{batch_size \times src_len \times d_model}$，并输出一个矩阵$\mathrm{O}_{batch_size \times src_len \times d_model}$。通过多个 Encoder block 叠加就可以组成 Encoder。

第一个 Encoder block 的输入为句子单词的表示向量矩阵，后续 Encoder block 的输入是前一个 Encoder block 的输出，最后一个 Encoder block 输出的矩阵就是**编码信息矩阵** C，这一矩阵后续会用到 Decoder 中。

4. Decoder

该结构的主要功能点如下：

1. 包含两个Multi-Head Attention
    1. Masked Multi-Head Attention
    2. Multi-Head Attention：K、V由Encoder的编码信息矩阵C提供，而Q使用Masked Multi-Head Attention的输出进行计算。
2. 最后的softmax层计算预测概率

4.1. 输入

目标序列的前序序列，后续简称dec input。

4.1.1. Output Embedding

对dec_input进行word2vec词向量编码，后续简称$\mathrm{X_dec}_{word_emb}$。

4.1.2. Positional Encoding

同Encoder的Positional Encoding，后续简称$\mathrm{X_dec}_{pos_encoding}$。

4.1.3. 输入整合

$$\mathrm{X\_dec} = \mathrm{X\_dec}_{embedding} = \mathrm{X\_dec}_{word\_emb} + \mathrm{X\_dec}_{pos\_encoding}$$

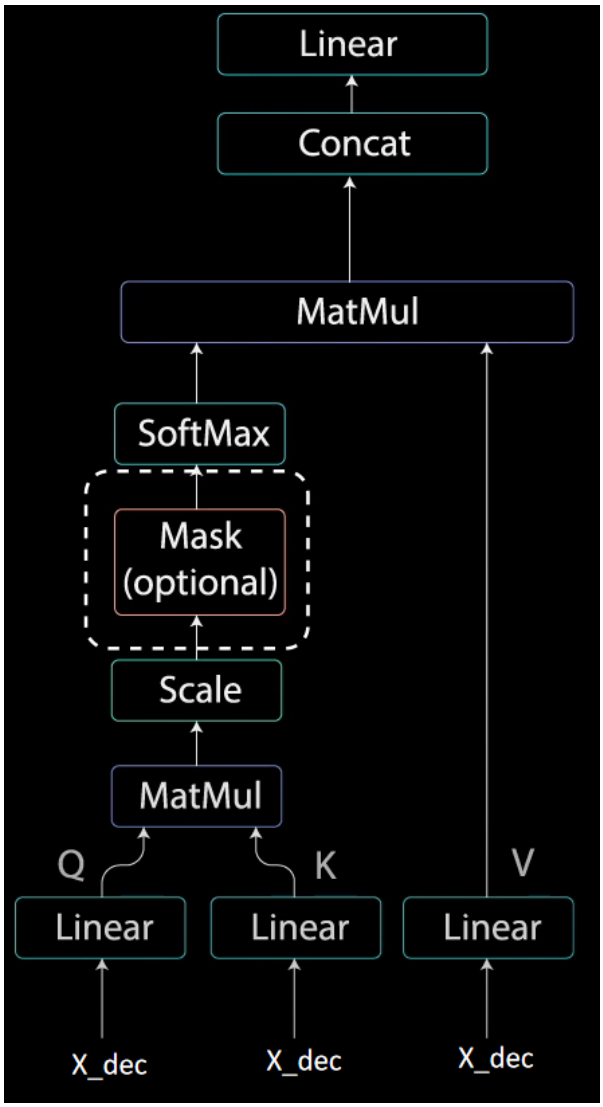
shape为[batch_size, tgt_len, d_model]。

4.2. Masked Attention

4.2.1. Masked Multi-Head Attention

对X_dec，经过三次线性变换，得到Q、K、V。然后进行Self-Attention操作，首先通过$\frac{Q\times K^T}{\sqrt{d_k}}$得到Scaled Scores，Scaled Scores是[tgt_len, tgt_len]的矩阵，接下来，在softmax之前，要对Scaled Scores进行**sequence mask**以及**padding mask**，前者防止信息泄漏，后者消除padding字符的影响。

Masked Multi-Head Attention整体流程图如下：



mask过程如下，在final mask中，数值为1的将被替换为-inf：

sequence mask

	S	I	am	chinese	P
S	0	1	1	1	1
I		0	1	1	1
am			0	1	1
chinese				0	1
P					0

+

padding mask

	S	I	am	chinese	P
S	F	F	F	F	T
I	F	F	F	F	T
am	F	F	F	F	T
chinese	F	F	F	F	T
P	F	F	F	F	T

=

final mask

	S	I	am	chinese	P
S	0	1	1	1	1
I		0	1	1	1
am			0	1	1
chinese				0	1
P					1

使用final mask矩阵，对scaled scores矩阵进行mask：

scaled scores

	S	I	am	chinese	P
S	0.6	0.1	0.1	0.1	0.1
I	0.1	0.6	0.1	0.1	0.1
am	0.2	0.1	0.3	0.2	0.2
chinese	0.3	0.1	0.1	0.3	0.2
P	0.4	0.2	0.2	0.1	0.1

+ final mask

	S	I	am	chinese	P
S	0	-inf	-inf	-inf	-inf
I		0	-inf	-inf	-inf
am			0	-inf	-inf
chinese				0	-inf
P					-inf

= masked scores

	S	I	am	chinese	P
S	0.6	-inf	-inf	-inf	-inf
I	0.1	0.6	-inf	-inf	-inf
am	0.2	0.1	0.3	-inf	-inf
chinese	0.3	0.1	0.1	0.3	-inf
P	0.4	0.2	0.2	0.1	-inf

基于masked scores进行softmax计算，可以将-inf变为0，得到的矩阵即为每个词之间的权重：

Softmax(

0.6	**-inf**	**-inf**	**-inf**	-inf
0.1	0.6	-inf	-inf	-inf
0.2	0.1	0.3	-inf	-inf
0.3	0.1	0.1	0.3	-inf
0.4	0.2	0.2	0.1	-inf

) =

	S	I	am	chinese	P
S	1	0	0	0	0
I	0.38	0.62	-inf	-inf	-inf
am	0.33	0.3	0.37	-inf	-inf
chinese	0.27	0.23	0.23	0.27	-inf
P	0.3	0.24	0.24	0.22	-inf

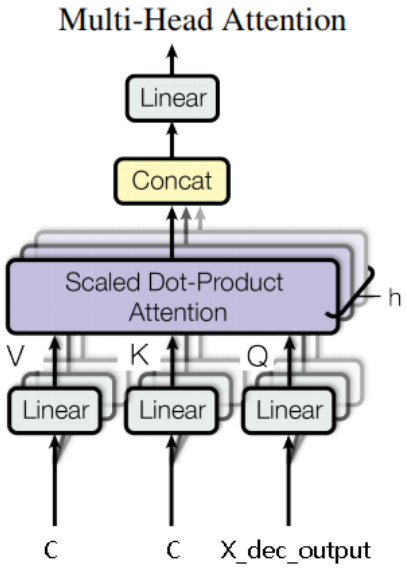
最后，对权重矩阵再进行scale，然后与V相乘得到输出结果。

将输出结果做残差连接及Layer Normlization后，输入到下一步的Multi-Head Attention。

将该步的结果记为$\mathrm{X_dec_output}$

4.3. Multi-Head Attention

Encoder的输出结果（矩阵C）提供K、V，Masked Attention的结果提供Q，再次进行Self-Attention操作。



该步Q的shape为[batch_size, tgt_len, d_k]，K的shape为[batch_size, src_len, d_k]，那么，$\mathrm{Q} \times \mathrm{K}^{\mathrm{T}}$的shape为[batch_size, tgt_len, src_len]，该步骤仅对K进行padding mask，如下图所示：

	我	是	中国人	P
S				
I				
am				
chinese				
P				

最后一行蓝色部分未进行padding mask，但是对最后一列必须进行padding mask。

4.4. 线性层

Feed Forward、add、norm与前文类似，不再赘述。

最后输出的结果，其shape为[batch_size, tgt_len, d_model]，与最初的输入X_dec的shape一致。

5. 结果输出

5.1. Linear

将Decoder的输出，通过Linear操作映射为[batch_size, tgt_len, tgt_vocab_size]的矩阵。

5.2. Softmax

通过Softmax计算出概率，然后

5.3. Loss

使用交叉熵损失函数计算loss。

6. 预测

dec_input的第一个字符为S，然后逐字预测，将概率最大的词加入到dec_input，如果该词是E，则结束。

```
def greedy_decoder(model, enc_input, start_symbol):
    # 根据enc_input获取enc_outputs
    enc_outputs, enc_self_attns = model.encoder(enc_input)
    dec_input = torch.zeros(1, 0).type_as(enc_input.data)
    terminal = False
    next_symbol = start_symbol
    while not terminal:
        # 将上一步预测的最大概率的词，作为dec_input

 dec_input=torch.cat([dec_input,torch.tensor([[next_symbol]],dtype=enc_input
.dtype)],-1)
        dec_outputs, _, _ = model.decoder(dec_input, enc_input, enc_outputs)
        projected = model.projection(dec_outputs)
```

```
        prob = projected.squeeze(0).max(dim=-1, keepdim=False)[1]
        next_word = prob.data[-1]
        next_symbol = next_word
        if next_symbol == word2idx["E"]:
            terminal = True
    return dec_input
```

7. 参考文档

1. The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)
2. wmathor/nlp-tutorial: Natural Language Processing Tutorial for Deep Learning Researchers (github.com)
3. Transformer详解 - mathor (wmathor.com)
4. Transformer的PyTorch实现 - mathor (wmathor.com)
5. In-layer normalization techniques for training very deep neural networks | AI Summer (theaisummer.com)
6. Transformer模型详解（图解最完整版） - 知乎 (zhihu.com)
7. BERT大火却不懂Transformer？读这一篇就够了 - 知乎 (zhihu.com)
8. Transformer论文逐段精读【论文精读】哔哩哔哩bilibili
9. Transformer从零详细解读(可能是你见过最通俗易懂的讲解)哔哩哔哩bilibili
10. Transformer代码(源码)从零解读(Pytorch版本)哔哩哔哩bilibili
11. Transformer的PyTorch实现哔哩哔哩bilibili
12. 从语言模型到Seq2Seq：Transformer如戏，全靠Mask - 科学空间|Scientific Spaces