详解Transformer中Self-Attention以及Multi-Head Attention

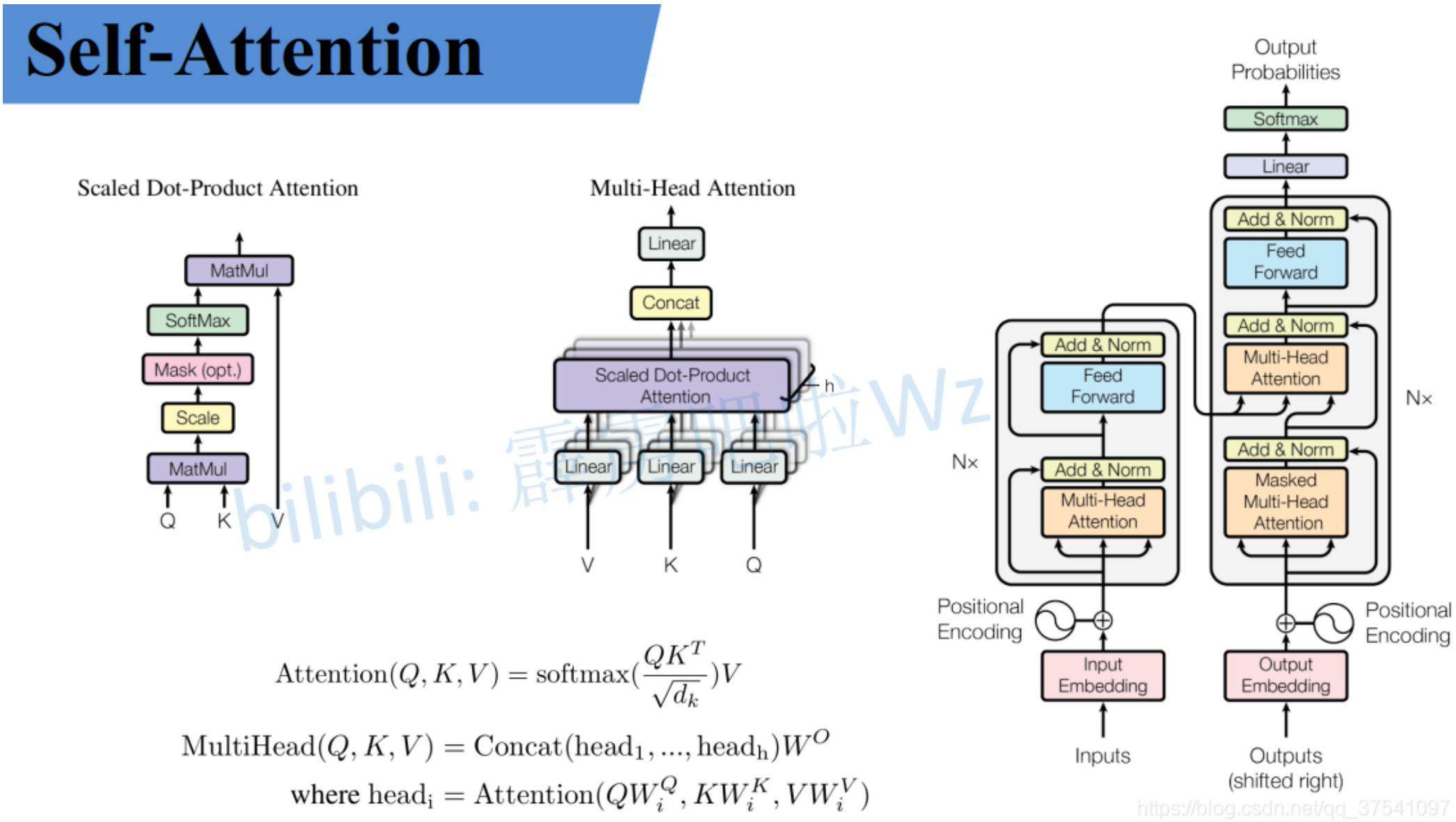
原文名称：Attention Is All You Need
原文链接：https://arxiv.org/abs/1706.03762

最近Transformer在CV领域很火，Transformer是2017年Google在 `Computation and Language` 上发表的，当时主要是针对自然语言处理领域提出的（之前的RNN模型记忆长度有限且无法并行化，只有计算完t_i时刻后的数据才能计算t_{i+1}时刻的数据，但Transformer都可以做到）。在这篇文章中作者提出了 `Self-Attention` 的概念，然后在此基础上提出 `Multi-Head Attention` ，所以本文对 `Self-Attention` 以及 `Multi-Head Attention` 的理论进行详细的讲解。在阅读本文之前，建议大家先去看下李弘毅老师讲的Transformer的内容。本文的内容是基于李弘毅老师讲的内容加上自己阅读一些源码进行的总结。

文章目录

前言

如果之前你有在网上找过self-attention或者transformer的相关资料，基本上都是贴的原论文中的几张图以及公式，如下图，讲的都挺抽象的，反正就是看不懂（可能我太菜的原因）。就像李弘毅老师课程里讲到的"不懂的人再怎么看也不会懂的"。那接下来本文就结合李弘毅老师课上的内容加上原论文的公式来一个个进行详解。



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

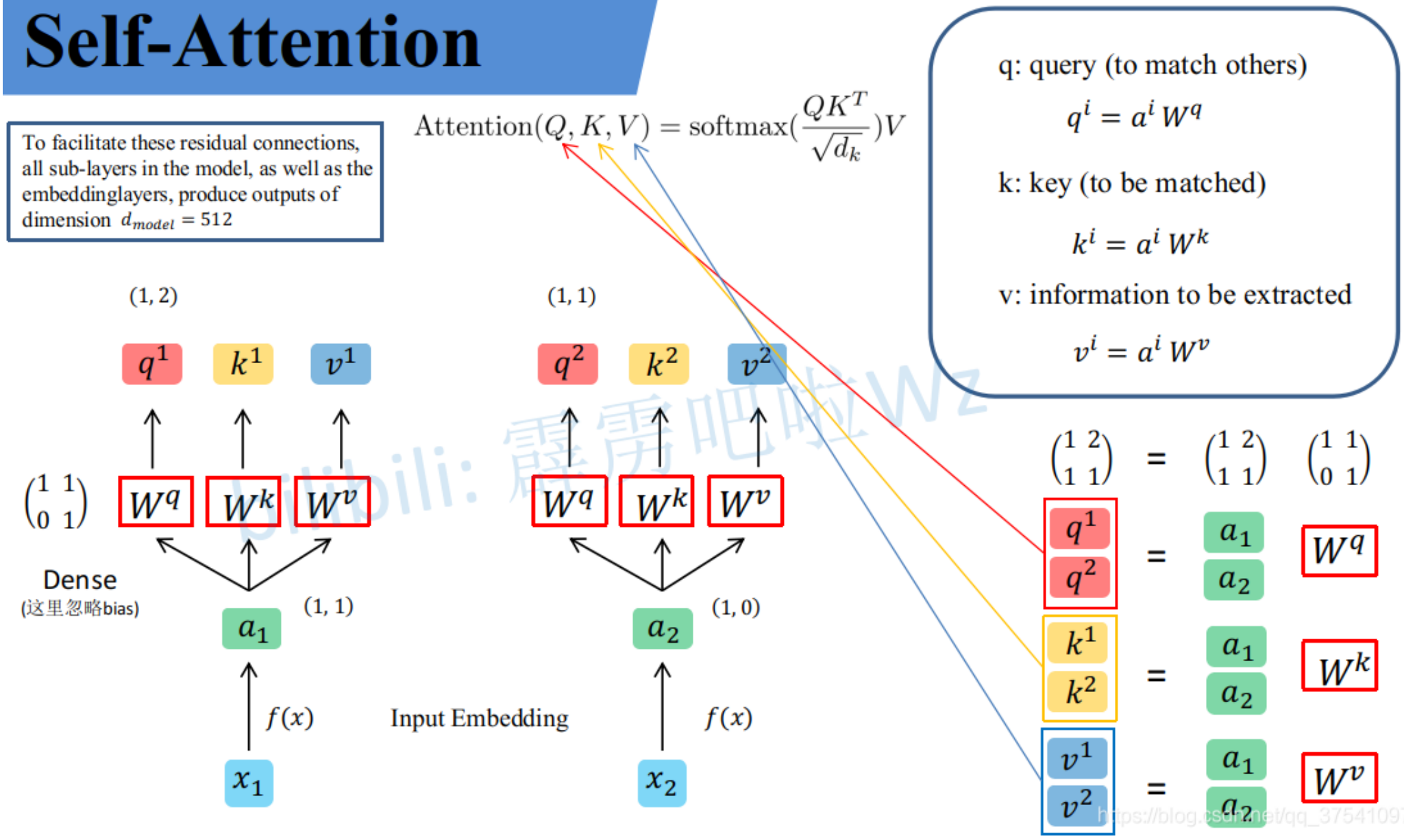
$$\text{MultiHead}(Q,K,V)=\text{Concat}(\text{head}_1,...,\text{head}_\text{h})W^O$$

$$\text{where head}_\text{i}=\text{Attention}(QW_i^Q,KW_i^K,VW_i^V)$$

Self-Attention

下面这个图是我自己画的，为了方便大家理解，假设输入的序列长度为2，输入就两个节点x_1,x_2，然后通过Input Embedding也就是图中的$f(x)$将输入映射到a_1,a_2。紧接着分别将a_1,a_2分别通过三个变换矩阵W_q,W_k,W_v（这三个参数是可训练的，是共享的）得到对应的q_i,k_i,v_i（这里在源码中是直接使用全连接层实现的，这里为了方便理解，忽略偏执）。

Self-Attention



其中

- q代表query，后续会去和每一个k进行匹配
- k代表key，后续会被每个q匹配
- v代表从a中提取得到的信息

假设$a_1 = (1, 1), a_2 = (1, 0), W^q = \binom{1,1}{0,1}$那么：

$$q^1 = (1, 1)\begin{pmatrix}1, 1\\ 0, 1\end{pmatrix} = (1, 2), \quad q^2 = (1, 0)\begin{pmatrix}1, 1\\ 0, 1\end{pmatrix} = (1, 1)$$

前面有说Transformer是可以并行化的，所以可以直接写成：

$$\begin{pmatrix}q^1\\ q^2\end{pmatrix} = \begin{pmatrix}1, 2\\ 1, 1\end{pmatrix}\begin{pmatrix}1, 1\\ 0, 1\end{pmatrix} = \begin{pmatrix}1, 2\\ 1, 1\end{pmatrix}$$

同理我们可以得到$\binom{k^1}{k^2}$和$\binom{v^1}{v^2}$，那么求得的$\binom{q^1}{q^2}$就是原论文中的Q，$\binom{k^1}{k^2}$就是K，$\binom{v^1}{v^2}$就是V。接着先拿q^1和每个k进行match，点乘操作，接着除以$\sqrt{d}$得到对应的α，其中d代表向量k^i的长度，在本示例中等于2，除以$\sqrt{d}$的原因在论文中的解释是“进行点乘后的数值很大，导致通过softmax后梯度变的很小”，所以通过除以$\sqrt{d}$来进行缩放。比如计算$\alpha_{1,i}$：

$$\alpha_{1,1} = \frac{q^1 \cdot k^1}{\sqrt{d}} = \frac{1 \times 1 + 2 \times 0}{\sqrt{2}} = 0.71$$
$$\alpha_{1,2} = \frac{q^1 \cdot k^2}{\sqrt{d}} = \frac{1 \times 0 + 2 \times 1}{\sqrt{2}} = 1.41$$

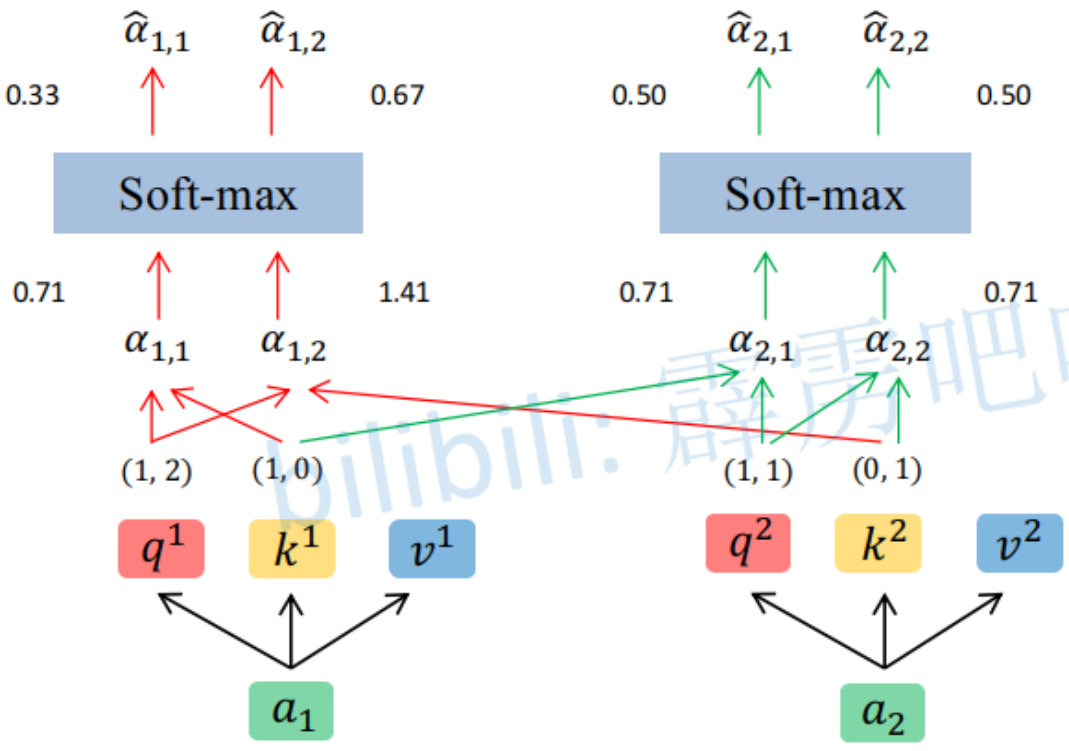
同理拿q^2去匹配所有的k能得到$\alpha_{2,i}$，统一写成矩阵乘法形式：

$$\begin{pmatrix}\alpha_{1,1} & \alpha_{1,2}\\ \alpha_{2,1} & \alpha_{2,2}\end{pmatrix} = \frac{\binom{q^1}{q^2}\binom{k^1}{k^2}^T}{\sqrt{d}}$$

接着对每一行即$(\alpha_{1,1}, \alpha_{1,2})$和$(\alpha_{2,1}, \alpha_{2,2})$分别进行softmax处理得到$(\hat{\alpha}_{1,1}, \hat{\alpha}_{1,2})$和$(\hat{\alpha}_{2,1}, \hat{\alpha}_{2,2})$，这里的$\hat{\alpha}$相当于计算得到针对每个$v$的权重。到这我们就完成了$\text{Attention}(Q, K, V)$公式中$\text{softmax}(\frac{QK^T}{\sqrt{d_k}})$部分。

Self-Attention

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



Scaled Dot-Product Attension:

$$\alpha_{1,i}=q^1\cdot k^i/\sqrt{d}$$

$$\alpha_{2,i}=q^2\cdot k^i/\sqrt{d}$$

(d is the dim of k)

.....

$$\begin{pmatrix}0.71 & 1.41\\ 0.71 & 0.71\end{pmatrix}=\begin{pmatrix}1 & 2\\ 1 & 1\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}/\ 1.41$$

$$\begin{pmatrix}\alpha_{1,1} & \alpha_{1,2}\\ \alpha_{2,1} & \alpha_{2,2}\end{pmatrix}=\begin{matrix}q^1\\ q^2\end{matrix}\ \ k^1\ k^2\ /\sqrt{d}$$

Soft-max（行）

$$\begin{pmatrix}\hat{\alpha}_{1,1} & \hat{\alpha}_{1,2}\\ \hat{\alpha}_{2,1} & \hat{\alpha}_{2,2}\end{pmatrix}\quad\begin{pmatrix}0.33 & 0.67\\ 0.50 & 0.50\end{pmatrix}$$

上面已经计算得到α，即针对每个v的权重，接着进行加权得到最终结果：

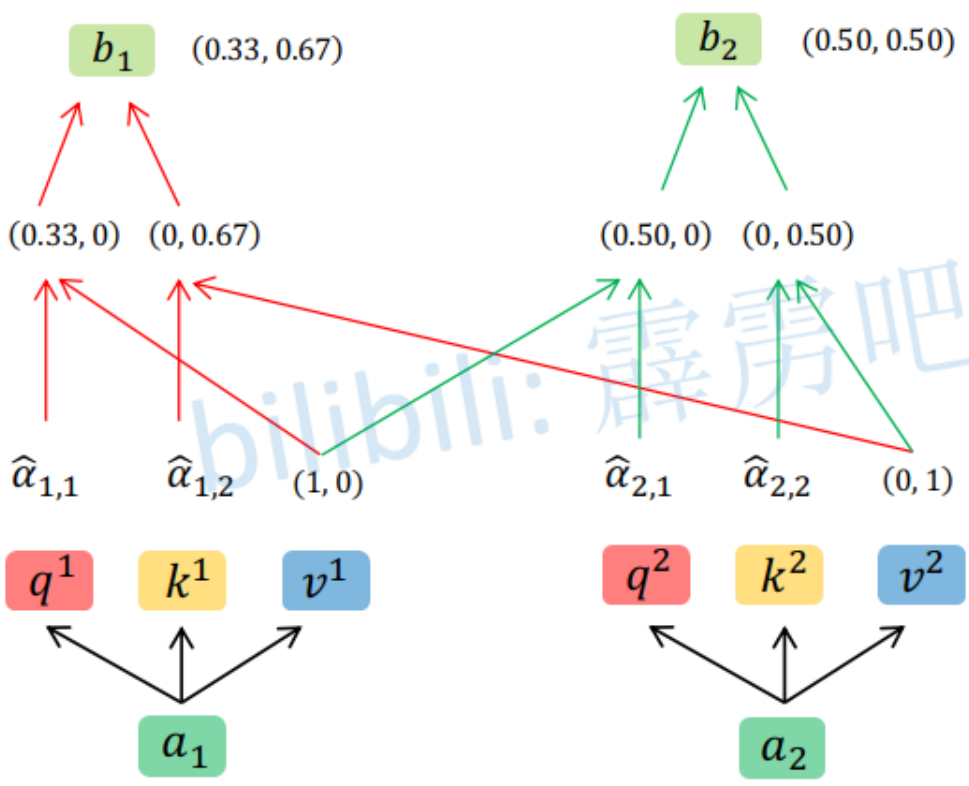
$$b_1=\hat{\alpha}_{1,1}\times v^1+\hat{\alpha}_{1,2}\times v^2=(0.33,0.67)$$
$$b_2=\hat{\alpha}_{2,1}\times v^1+\hat{\alpha}_{2,2}\times v^2=(0.50,0.50)$$

统一写成矩阵乘法形式：

$$\begin{pmatrix}b_1\\ b_2\end{pmatrix}=\begin{pmatrix}\hat{\alpha}_{1,1} & \hat{\alpha}_{1,2}\\ \hat{\alpha}_{2,1} & \hat{\alpha}_{2,2}\end{pmatrix}\begin{pmatrix}v^1\\ v^2\end{pmatrix}$$

Self-Attention

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



$$b^1=\sum_i\hat{\alpha}_{1,i}\times v^i$$

$$b^2=\sum_i\hat{\alpha}_{2,i}\times v^i$$

$$\begin{pmatrix}0.33 & 0.67\\ 0.50 & 0.50\end{pmatrix}=\begin{pmatrix}0.33 & 0.67\\ 0.50 & 0.50\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$$

$$\begin{matrix}b_1\\ b_2\end{matrix}=\begin{pmatrix}\hat{\alpha}_{1,1} & \hat{\alpha}_{1,2}\\ \hat{\alpha}_{2,1} & \hat{\alpha}_{2,2}\end{pmatrix}\begin{matrix}v^1\\ v^2\end{matrix}$$

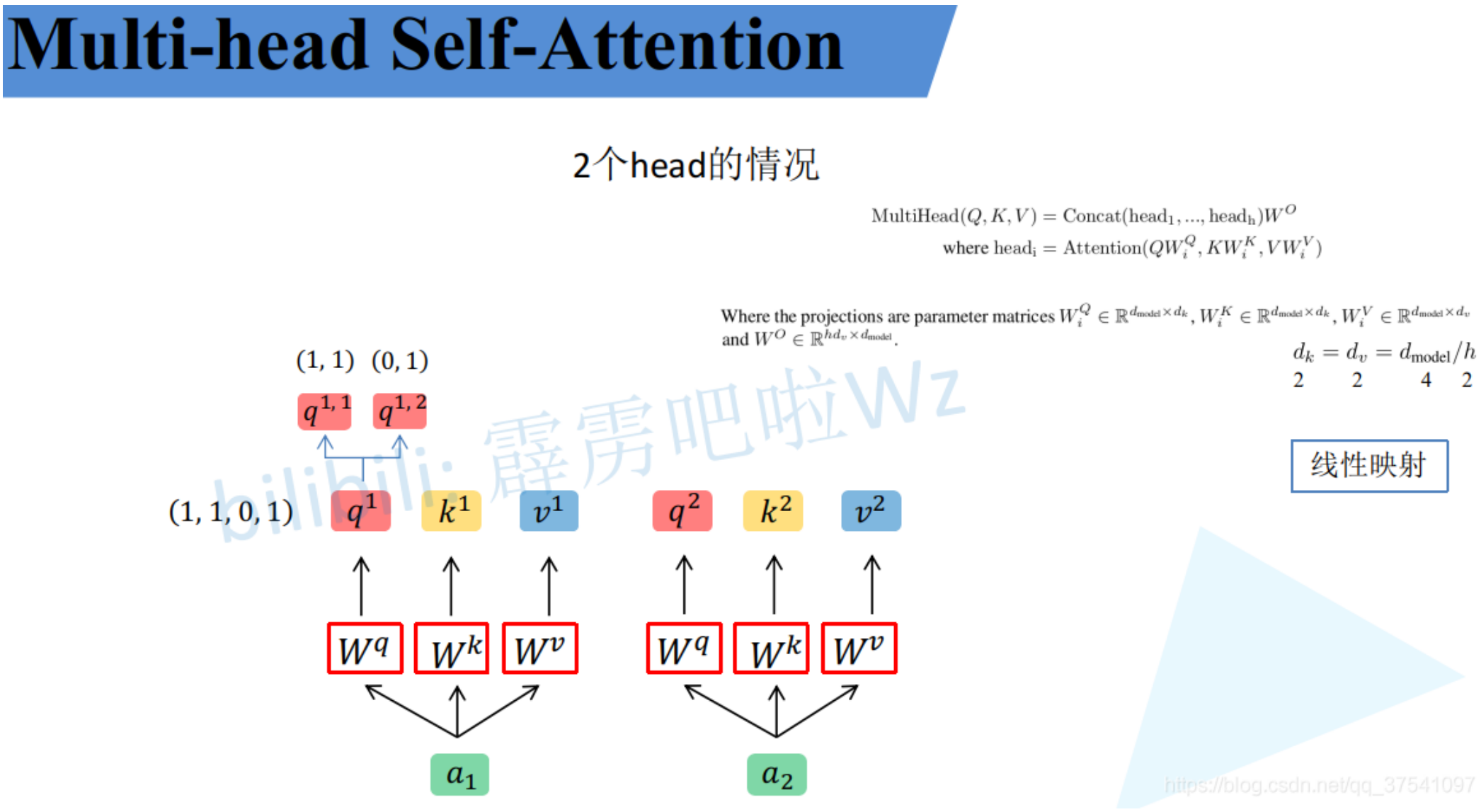
到这， `Self-Attention` 的内容就讲完了。总结下来就是论文中的一个公式：

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

Multi-Head Attention

刚刚已经聊完了Self-Attention模块，接下来再来看看Multi-Head Attention模块，实际使用中基本使用的还是Multi-Head Attention模块。原论文中说使用多头注意力机制能够联合来自不同head部分学习到的信息。 `Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.` 其实只要懂了Self-Attention模块Multi-Head Attention模块就非常简单了。

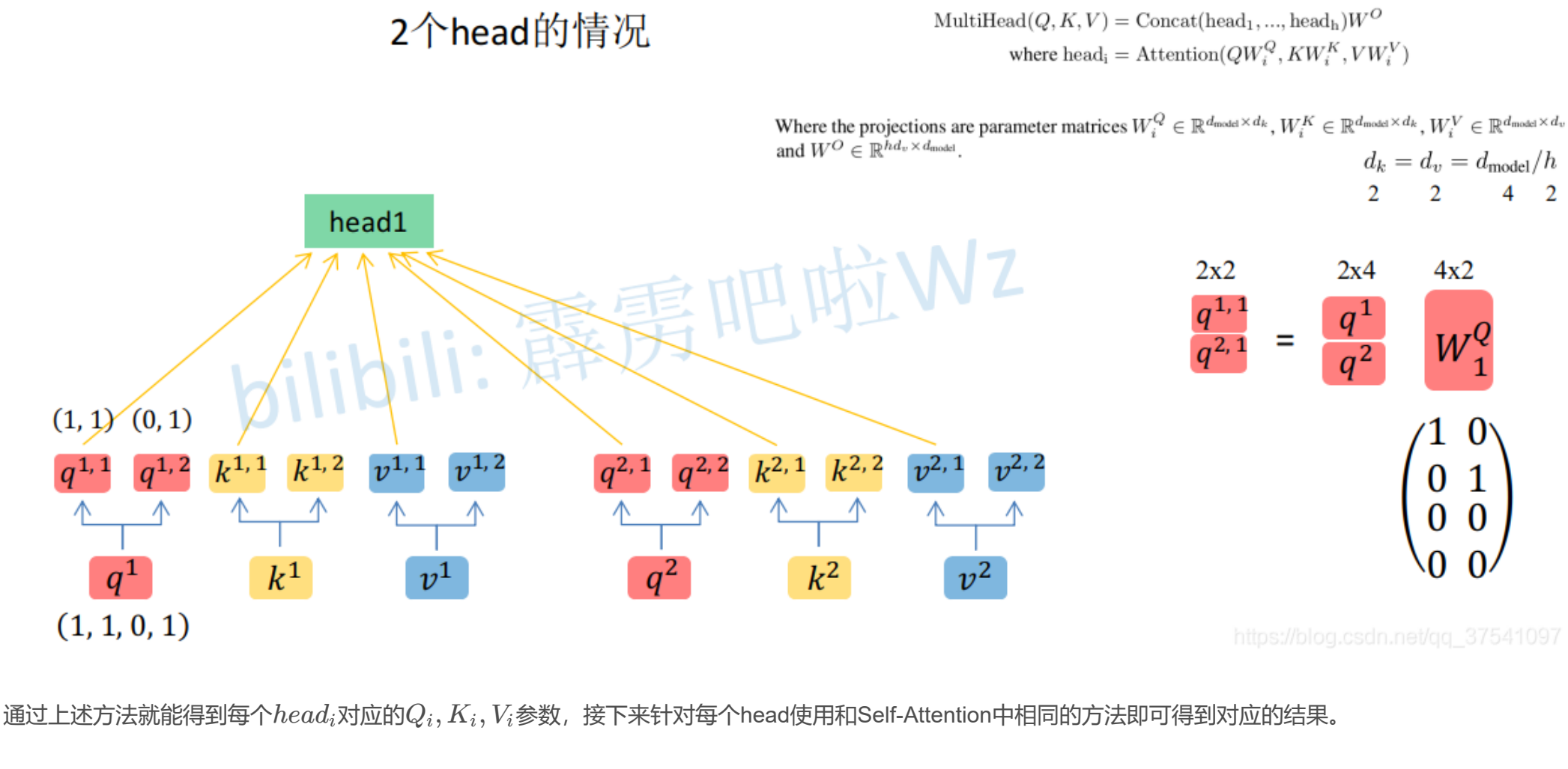
首先还是和Self-Attention模块一样将a_i分别通过W^q, W^k, W^v得到对应的q^i, k^i, v^i，然后再根据使用的head的数目h进一步把得到的q^i, k^i, v^i均分成h份。比如下图中假设$h = 2$然后q^1拆分成$q^{1,1}$和$q^{1,2}$，那么$q^{1,1}$就属于head1，$q^{1,2}$属于head2。



看到这里，如果读过原论文的人肯定有疑问，论文中不是写的通过W_i^Q, W_i^K, W_i^V映射得到每个head的Q_i, K_i, V_i吗：

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

但我在github上看的一些源码中就是简单的进行均分，其实也可以将W_i^Q, W_i^K, W_i^V设置成对应值来实现均分，比如下图中的Q通过W_1^Q就能得到均分后的Q_1。



通过上述方法就能得到每个$head_i$对应的Q_i, K_i, V_i参数，接下来针对每个head使用和Self-Attention中相同的方法即可得到对应的结果。

$$\text{Attention}(Q_i, K_i, V_i) = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{d_k}})V_i$$

Multi-head Self-Attention

2个head的情况

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

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 $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

$$d_k = d_v = d_{\text{model}}/h$$
2 2 4 2

$b_{1,1}$ $b_{2,1}$ $b_{1,2}$ $b_{2,2}$

$q^{1,1}$ $k^{1,1}$ $v^{1,1}$ $q^{2,1}$ $k^{2,1}$ $v^{2,1}$

$q^{1,2}$ $k^{1,2}$ $v^{1,2}$ $q^{2,2}$ $k^{2,2}$ $v^{2,2}$

head1 head2

接着将每个head得到的结果进行concat拼接，比如下图中$b_{1,1}$（$head_1$得到的b_1）和$b_{1,2}$（$head_2$得到的b_1）拼接在一起，$b_{2,1}$（$head_1$得到的b_2）和$b_{2,2}$（$head_2$得到的b_2）拼接在一起。

Multi-head Self-Attention

2个head的情况

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

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 $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

$$d_k = d_v = d_{\text{model}}/h$$
2 2 4 2

Concat

$b_{1,1}$ $b_{1,2}$ $b_{2,1}$ $b_{2,2}$

(1, 1, 0, 1) (0, 1, 1, 0)

$b_{1,1}$ $b_{2,1}$ $b_{1,2}$ $b_{2,2}$

(1, 1) (0, 1) (0, 1) (1, 0)

接着将拼接后的结果通过W^O（可学习的参数）进行融合，如下图所示，融合后得到最终的结果b_1, b_2。

Multi-head Self-Attention

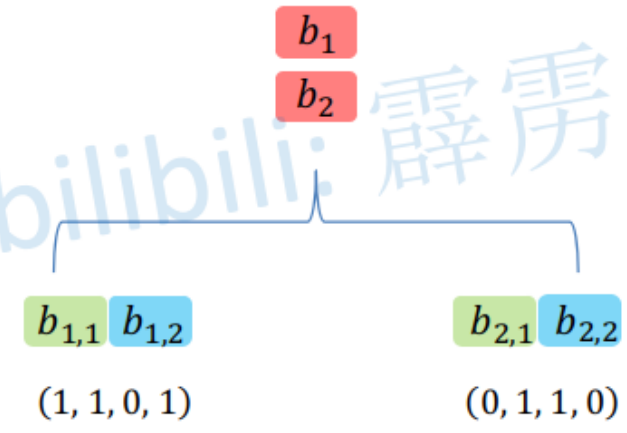
2个head的情况

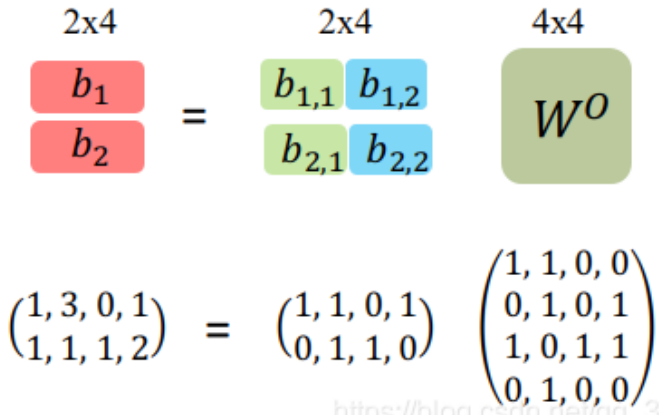
Fused

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

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 $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

$$d_k = d_v = d_{\text{model}}/h$$
$$2 \quad 2 \quad 4 \quad 2$$





到这， `Multi-Head Attention` 的内容就讲完了。总结下来就是论文中的两个公式：

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

Positional Encoding

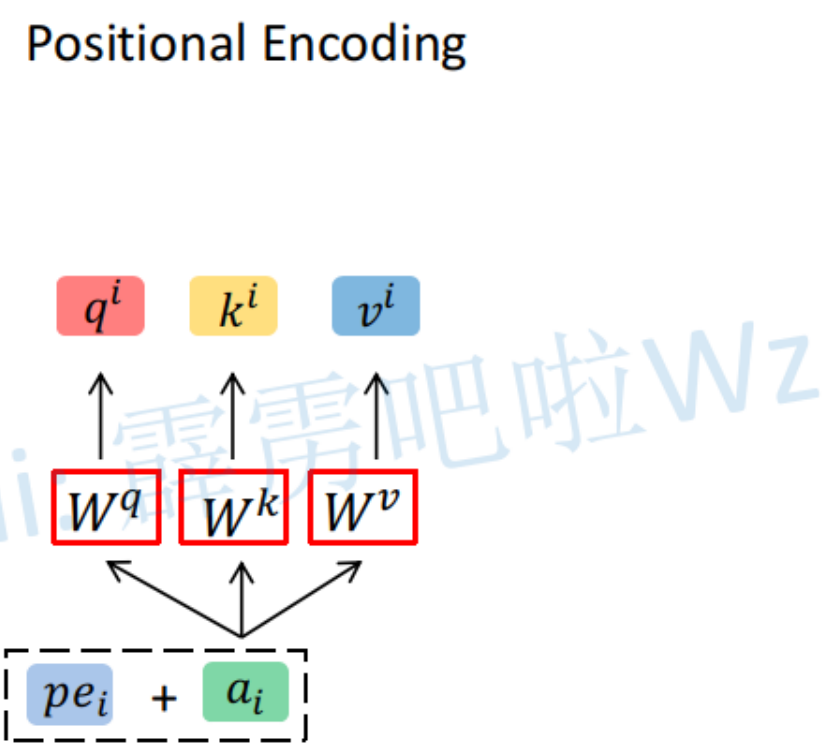
如果仔细观察刚刚讲的Self-Attention和Multi-Head Attention模块，在计算中是没有考虑到位置信息的。假设在Self-Attention模块中，输入a_1, a_2, a_3得到b_1, b_2, b_3。对于a_1而言，a_2和a_3离它都是一样近的而且没有先后顺序。假设将输入的顺序改为a_1, a_3, a_2，对结果b_1是没有任何影响的。下面是使用Pytorch做的一个实验，首先使用 `nn.MultiheadAttention` 创建一个 `Self-Attention` 模块（ `num_heads=1` ），注意这里在正向传播过程中直接传入QKV，接着创建两个顺序不同的QKV变量t1和t2（主要是将q^2, k^2, v^2和q^3, k^3, v^3的顺序换了下），分别将这两个变量输入Self-Attention模块进行正向传播。

```
import torch
import torch.nn as nn


m = nn.MultiheadAttention(embed_dim=2, num_heads=1)

t1 = [[[1., 2.],   # q1, k1, v1
       [2., 3.],   # q2, k2, v2
       [3., 4.]]]  # q3, k3, v3

t2 = [[[1., 2.],   # q1, k1, v1
       [3., 4.],   # q3, k3, v3
       [2., 3.]]]  # q2, k2, v2

q, k, v = torch.as_tensor(t1), torch.as_tensor(t1), torch.as_tensor(t1)
print("result1: \n", m(q, k, v))

q, k, v = torch.as_tensor(t2), torch.as_tensor(t2), torch.as_tensor(t2)
print("result2: \n", m(q, k, v))
```

对比结果可以发现，即使调换了q^2, k^2, v^2和q^3, k^3, v^3的顺序，但对于b_1是没有影响的。

```
result1:
 (tensor([[[0.5651, 0.0727],
          [0.8517, 0.1088],
          [1.1383, 0.1448]]], grad_fn=<AddBackward0>), tensor([[[1.]],

         [[1.]],

         [[1.]]], grad_fn=<DivBackward0>))
result2:
 (tensor([[[0.5651, 0.0727],
          [1.1383, 0.1448],
          [0.8517, 0.1088]]], grad_fn=<AddBackward0>), tensor([[[1.]],

         [[1.]],

         [[1.]]], grad_fn=<DivBackward0>))
```


为了引入位置信息，在原论文中引入了位置编码 positional encodings 。 To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. 如下图所示，位置编码是直接加在输入的$a = \{a_1, ..., a_n\}$中的，即$pe = \{pe_1, ..., pe_n\}$和$a = \{a_1, ..., a_n\}$拥有相同的维度大小。关于位置编码在原论文中有提出两种方案，一种是原论文中使用的固定编码，即论文中给出的 sine and cosine functions 方法，按照该方法可计算出位置编码；另一种是可训练的位置编码，作者说尝试了两种方法发现结果差不多（但在ViT论文中使用的是可训练的位置编码）。



到这，关于Self-Attention、Multi-Head Attention以及位置编码的内容就全部讲完了，如果有讲的不对的地方希望大家指出。

相关推荐