

Lecture 9: 注意力机制

注意力机制

概念: 注意力机制

- 引入:
 - 心理学认为, 人通过随意线索和不随意线索选择注意点;
 - 注意力机制中, 通过 **query** (随意线索)和 **key** (不随意线索)来有偏向性的选择输入;
 - 因为卷积、全连接、池化层都只考虑不随意线索;
 - 而注意力机制则显示的考虑随意线索;
 - 随意线索被称之为查询 (**query**), 表示我想要做什么;
 - 不随意线索就是其他背景的信息;
 - 每个输入是一个值 (**value**) 和不随意线索 (**key**) 的对;
 - 通过注意力池化层, 显示的加入查询, 并来有偏向性的选择选择某些输入;
- 概念: 非参注意力池化层
 - 概念:
 - 非参数注意力池化层 (**Non-parametric Attention Pooling Layer**) 是一种基于注意力机制的池化方法, 它不需要可学习的参数, 而是通过计算输入特征之间的相似性来动态地分配权重, 从而对特征进行加权聚合;
 - 这种方法常用于处理序列数据或空间数据, 能够捕捉数据中的重要信息;
 - 公式: 相似性

假设输入特征为 $X = \{x_1, x_2, \dots, x_N\}$, 其中 $x_i \in \mathbb{R}^d$ 是第 i 个特征向量, N 是特征的数量, d 是特征的维度。非参数注意力池化的输出 y 可以表示为:

$$y = \sum_{i=1}^N \alpha_i x_i$$

其中, α_i 是第 i 个特征的注意力权重, 计算公式为:

$$\alpha_i = \frac{\exp(s(x_i, q))}{\sum_{j=1}^N \exp(s(x_j, q))}$$

这里:

- q 是一个查询向量 (query), 通常由任务需求决定, 可以是固定的或动态生成的。
 - $s(x_i, q)$ 是相似性函数, 用于计算特征 x_i 与查询 q 之间的相似性。
- 概念: 相似函数
 - 相似函数用来衡量特征和查询之间的相关性, 可以用点积或者余弦相似度来衡量;
 - 概念: 注意力权重
 - 注意力权重为 α_i , 通过 **softmax** 函数计算, 确保所有权重的和为 1;

注意力分数

基础概念

概念：注意力分数

- 引入：

注意力分数

注意力权重

注意力分数

• 回顾：

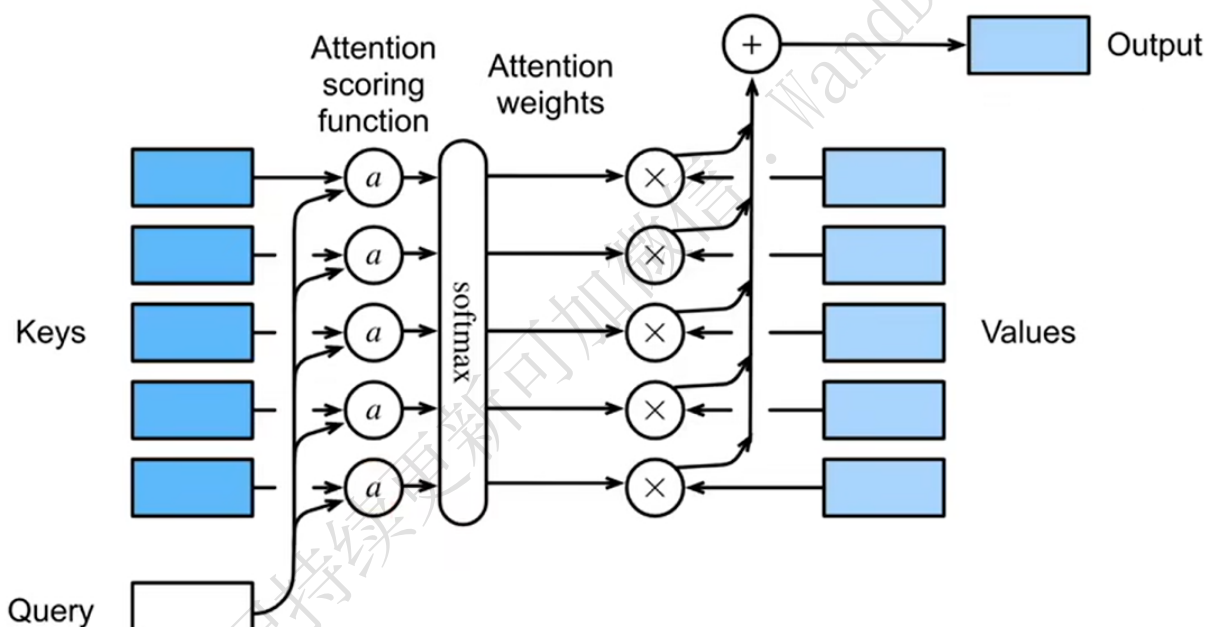
$$f(x) = \sum_i \alpha(x, x_i) y_i = \sum_{i=1}^n \text{softmax} \left(-\frac{1}{2}(x - x_i)^2 \right) y_i$$

- 核心：

- 1. 注意力分数是 query 和 key 的相似度；
- 2. 注意力权重是注意力分数经过 softmax 结果；

- 概念：

- 不同的值输入后，计算其注意力分数，然后通过 Softmax 将其映射到 0~1 之间，并作为权重乘到不同的 Value 上；



注意力分数的两种计算

概念：Attention Scoring Function 的设计

- 方法一：Additive Attention

- 引入：

- Additive Attention（可加性注意力）是一种用于计算注意力分数的机制，常见于序列到序列（Seq2Seq）模型和 Transformer 模型中；
- 其核心思想是通过一个神经网络来计算查询（Query）和键（Key）之间的相关性；

- 核心：

- 将 query 和 key 合并起来进入一个单输出单隐藏层的 MLP；

- 概念：三个参数

- 可加性的注意力。可以学习的参数有三个：

$$\mathbf{W}_k \in \mathbb{R}^{h \times k}, \mathbf{W}_q \in \mathbb{R}^{h \times q}, \mathbf{v} \in \mathbb{R}^h$$

$$a(\mathbf{k}, \mathbf{q}) = \mathbf{v}^T \tanh(\mathbf{W}_k \mathbf{k} + \mathbf{W}_q \mathbf{q})$$

• 解释:

1. $\mathbf{W}_q$:

- 用于将查询向量 $\mathbf{q}$ 映射到一个中间向量空间。
- 维度为 d_{model} ，其中 d_{model} 是输入向量的维度，是注意力机制的隐藏维度。

2. $\mathbf{W}_k$:

- 用于将键向量 $\mathbf{k}$ 映射到与查询向量相同的中间向量空间。
- 维度与 $\mathbf{W}_q$ 相同。

3. $\mathbf{v}$:

- 用于将中间表示映射为标量注意力分数。
- 维度为 d_{model} 。

• 概念：注意力分数的计算

- 将查询向量 $\mathbf{q}$ 和键（Key）向量 $\mathbf{k}$ 拼接或相加，通过一个全连接层（通常是单层神经网络）和非线性激活函数（如 $\tanh$ ）计算出一个中间表示；

公式为：

$$e_{ij} = \mathbf{v}^T \tanh(\mathbf{W}_q \mathbf{q}_i + \mathbf{W}_k \mathbf{k}_j)$$

其中：

- e_{ij} 是查询 $\mathbf{q}_i$ 和键 $\mathbf{k}_j$ 之间的注意力分数。
- $\mathbf{W}_q$ 和 $\mathbf{W}_k$ 是可学习的权重矩阵，分别用于将查询和键映射到一个共同的向量空间。
- $\mathbf{v}$ 是一个可学习的权重向量，用于将中间表示映射为标量分数。

- 等价于将 **key** 和 **value** 合并起来后放入到一个隐藏大小为 h ，输出大小为 1 的单隐藏层 **MLP**；

• 方式二：Scaled Dot-Product Attention

• 核心：

- 直接将 **query** 和 **key** 做内积；

• 最简单的版本：

如果query和key都是同样的长度 $\mathbf{q}, \mathbf{k}_i \in \mathbb{R}^d$ ，那么可以

$$a(\mathbf{q}, \mathbf{k}_i) = \langle \mathbf{q}, \mathbf{k}_i \rangle / \sqrt{d}$$

• 向量化版本：

- $\mathbf{Q} \in \mathbb{R}^{n \times d}$, $\mathbf{K} \in \mathbb{R}^{m \times d}$, $\mathbf{V} \in \mathbb{R}^{m \times v}$
- 注意力分数: $a(\mathbf{Q}, \mathbf{K}) = \mathbf{Q}\mathbf{K}^T / \sqrt{d} \in \mathbb{R}^{n \times m}$
- 注意力池化: $f = \text{softmax}(a(\mathbf{Q}, \mathbf{K})) \mathbf{V} \in \mathbb{R}^{n \times v}$

代码实现

代码

```
import torch
from torch import nn

class AdditiveAttention(nn.Module):
    def __init__(self, key_size, query_size, num_hiddens, dropout, **kwargs):
        super(AdditiveAttention, self).__init__(**kwargs)
        # 定义可学习的参数
        self.W_k = nn.Linear(key_size, num_hiddens, bias=False) # 将键 (Key) 映射到隐藏空间
        self.W_q = nn.Linear(query_size, num_hiddens, bias=False) # 将查询 (Query) 映射到隐藏空间
        self.W_v = nn.Linear(num_hiddens, 1, bias=False) # 将隐藏状态映射为标量注意力分数
        self.dropout = nn.Dropout(dropout) # 用于防止过拟合的Dropout层

    def forward(self, queries, keys, values, valid_lens):
        # 投影查询和键到隐藏空间
        queries, keys = self.W_q(queries), self.W_k(keys) # 输入维度: [batch_size, seq_len, feature_dim]

        # 扩展维度以对齐计算 (广播机制)
        # queries.unsqueeze(2): [batch_size, query_seq_len, 1, num_hiddens]
        # keys.unsqueeze(1): [batch_size, 1, key_seq_len, num_hiddens]
        features = queries.unsqueeze(2) + keys.unsqueeze(1) # 广播相加后的维度: [batch_size, query_seq_len, key_seq_len, num_hiddens]

        features = torch.tanh(features) # 应用tanh激活函数 (非线性变换)
        scores = self.W_v(features).squeeze(-1) # 映射到标量分数并压缩最后一维: [batch_size, query_seq_len, key_seq_len]

        # 计算带掩码的注意力权重
        self.attention_weights = masked_softmax(scores, valid_lens) # 输出维度: [batch_size, query_seq_len, key_seq_len]

        # 对注意力权重应用Dropout后与值 (Value) 加权求和
        return torch.bmm(self.dropout(self.attention_weights), values) # 输出维度: [batch_size, query_seq_len, value_dim]
```

- 分析:

理论公式:

$$e_{ij} = v^T \tanh(W_q q_i + W_k k_j)$$

代码实现:

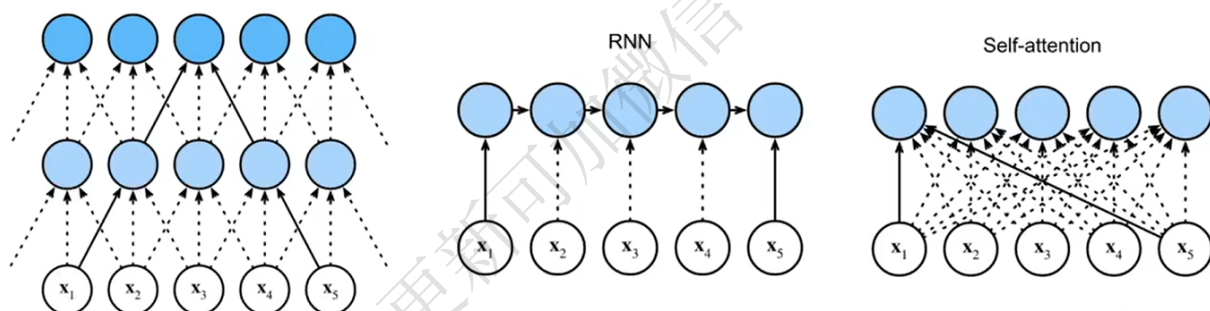
- `W_q` 和 `W_k` 对应公式中的线性变换。
- `W_v` 对应公式中的向量 `v` (通过线性层实现)。
- `features = queries + keys` 对应 `W_q q_i + W_k k_j` (广播相加)。

自注意力

区别

概念: 自注意力和其他架构区别

- 区别: CNN、RNN、自注意力



- 区别: 性能

	CNN	RNN	自注意力
计算复杂度	$O(knd^2)$	$O(nd^2)$	$O(n^2d)$
并行度	$O(n)$	$O(1)$	$O(n)$
最长路径	$O(n/k)$	$O(n)$	$O(1)$

- 自注意力在序列非常长是, 其时间复杂度很高, 是 n 的平方;
- 自注意力的并行度很好;
- 其最长路径是 $O(1)$;

- 总结:

- 自注意力比较适合比较长的文本, 其可以无视距离、看比较长的文本;
- 但即使可以看的比较长, 其计算性能特别高;

基础概念

概念: 自注意力

- 概念:

- 自注意力机制是一种**序列内特征交互**的机制，它通过计算序列中每个元素与其他元素的相关性，动态地聚合全局信息。自注意力池化层将 x_i 当做 **key**，**value**，**query** 来对序列抽取特征；

- 特点:

- 1. **捕捉长程依赖**: 不受序列长度限制，直接建模任意两个位置的关系；
- 2. **动态权重分配**: 根据内容相关性，自动调整注意力权重，而非固定模式；
- 3. **并行计算**: 摆脱 **RNN** 的时序依赖，支持高效并行化；

- 计算:

- (0) 将某一个输入的 X_i 通过 $f(x)$ 将其嵌入到高纬度；
- (1) 特征映射:

将输入序列 $X \in \mathbb{R}^{n \times d}$ 通过线性变换得到查询 (Query)、键 (Key)、值 (Value) :

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

其中 W_Q, W_K, W_V 是可学习参数矩阵。

- (2) 注意力权重计算:

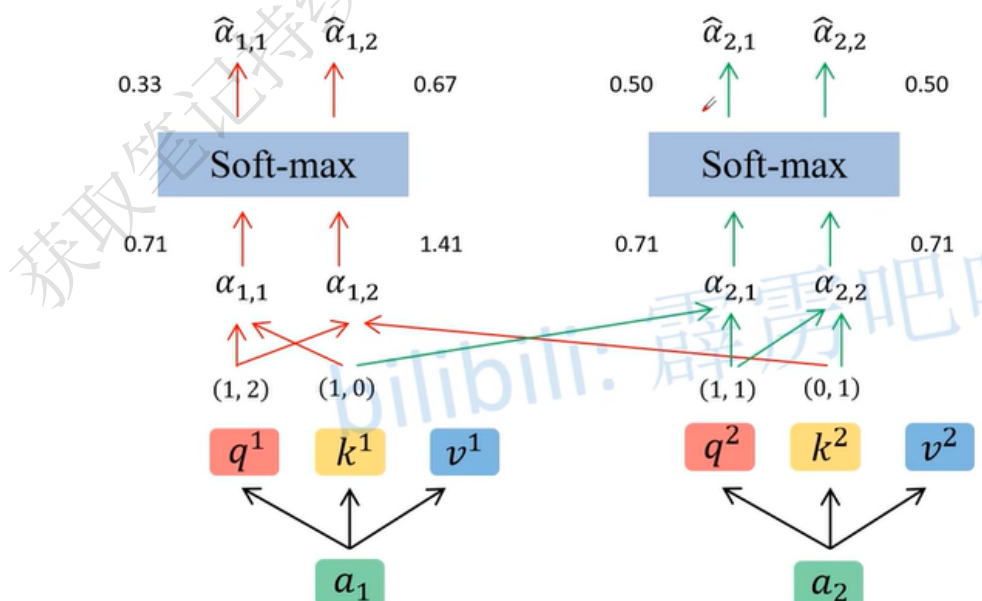
- 公式:

通过点积计算相关性，并归一化为概率分布:

$$A = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right)$$

其中 d_k 是缩放因子，用于防止点积值过大导致梯度消失。

- 图示:



- 这里计算得到的 α 帽就是每一个 V 所对应的权重，权重越大、也就越关注某一个 V ；

- (3) 特征聚合:

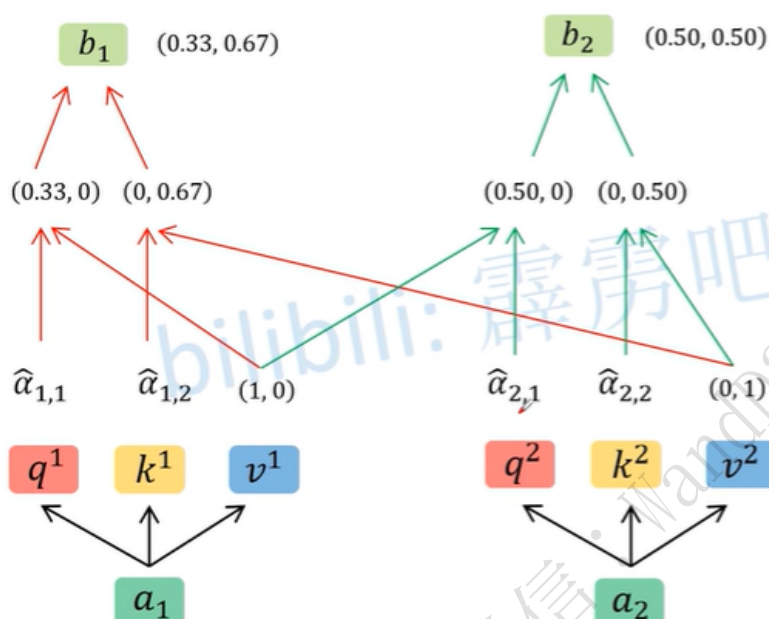
- 公式:

使用注意力权重对值进行加权求和：

$$Z = AV$$

输出 $Z \in \mathbb{R}^{n \times d}$ 是每个位置聚合全局信息后的新表示。

• 解释：



• 公式：点积注意力

点积注意力是自注意力机制的核心，其计算方式为：

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

- **点积**：计算查询和键的相似度
- **缩放因子**： $\sqrt{d_k}$ 用于防止点积值过大导致梯度消失
- **softmax**：将相似度转换为概率分布

概念：Q、K、V 的含义

• 概念：Q、K、V 的含义

• **Query**（查询）

- **角色**：主动寻找信息的主体；
- **类比**：相当于搜索引擎中的“搜索词”；
- **作用**：表示当前需要关注的目标位置（如 NLP 中的当前词）；

• **Key**

- **角色**：被匹配的信息索引；
- **类比**：相当于网页的“关键词标签”；
- **作用**：表示各位置的匹配特征（如 NLP 中其他词的关联特征）

• **Value**

- **角色**：实际携带的信息内容，表示机器认为它实际上学习到的信息；
- **类比**：相当于网页的"正文内容"；
- **作用**：包含需要被聚合的原始信息（如 NLP 中其他词的语义信息）；

• 总结：

W_Q	如何提出问题（关注方向）	注意力聚焦区域
W_K	如何组织可检索信息	信息检索效率
W_V	如何保留有用信息	信息传递保真度

• 关系：

通过Q和K的矩阵乘法计算相似度：

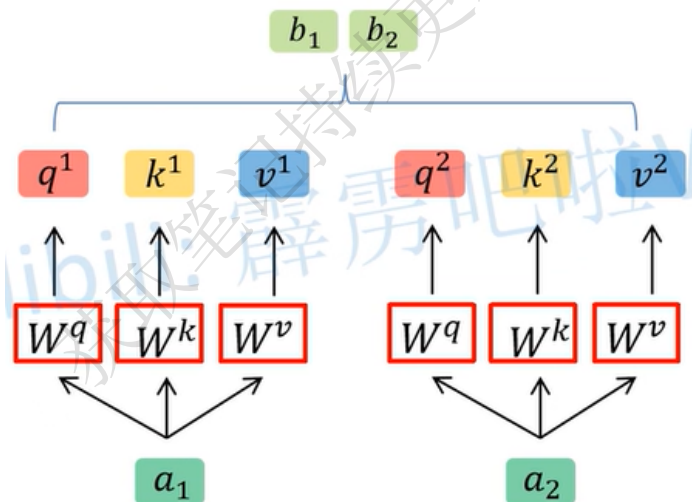
$$A = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right)$$

- **Q的列方向**：表示每个查询的特征维度
- **K的行方向**：表示每个键的特征维度
- **点积结果**：反映查询与键的匹配程度

多头自注意力机制

概念：多头自注意力机制

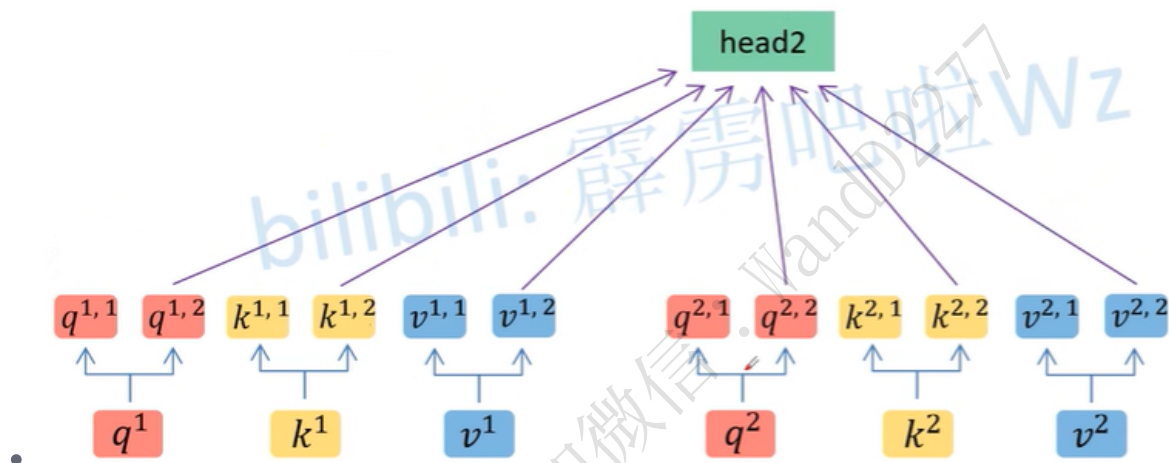
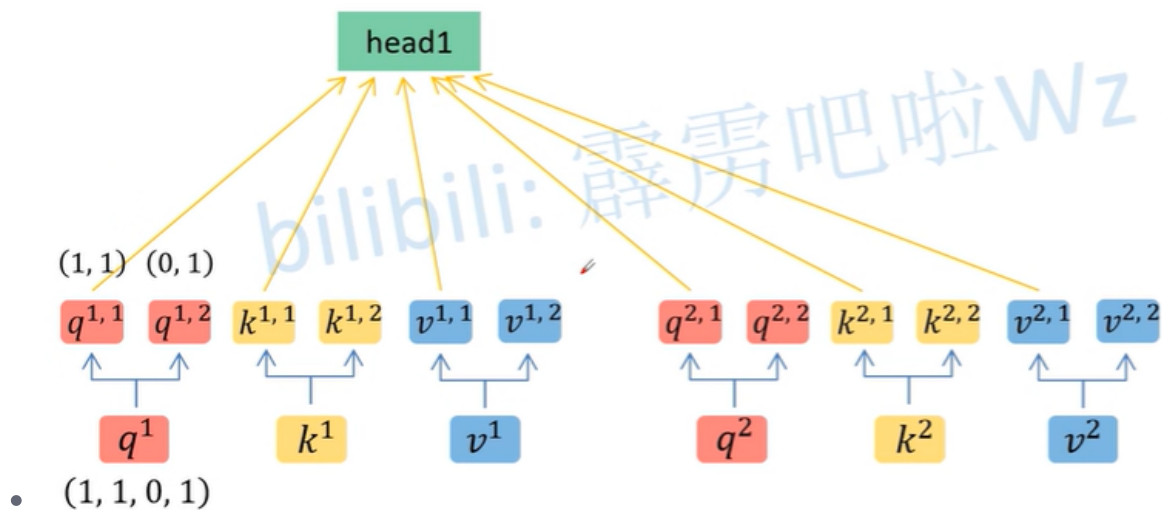
• 图示：1 个 head 的情况



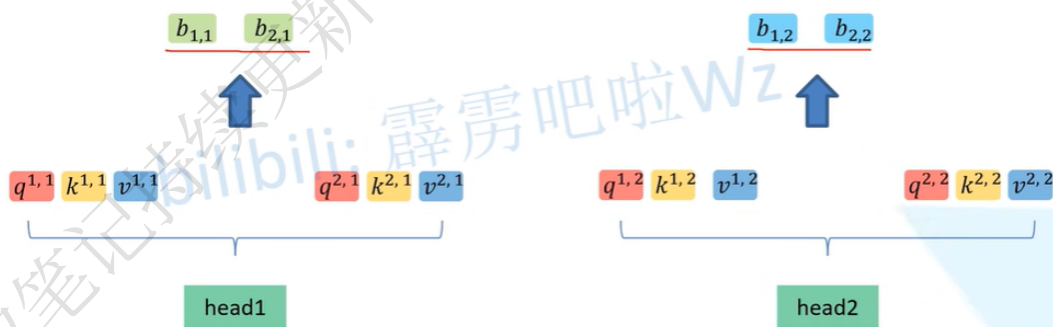
• 步骤：多头自注意力机制

• 第一步：2 个 head 的情况

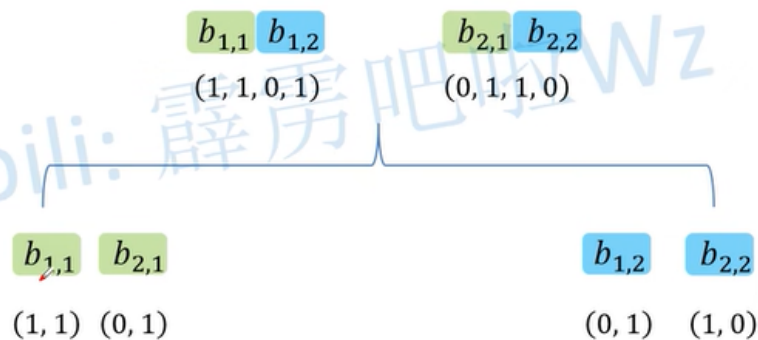
- 将所有均分出来的第一个部分计算给 head1，将另一部分计算给 head2；



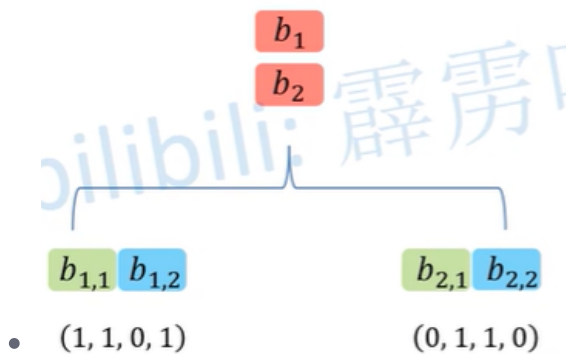
- 第二步：两个 head 计算
 - 分别按照自注意力的机制，计算 b 的值；



- 对其进行拼接：



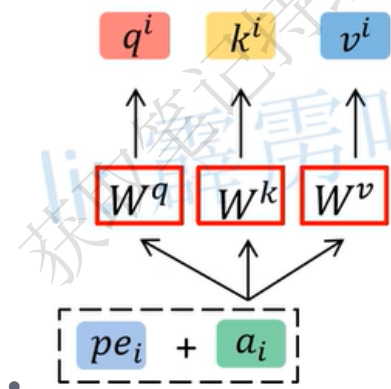
- 第三步：使用 W^O 来进行融合



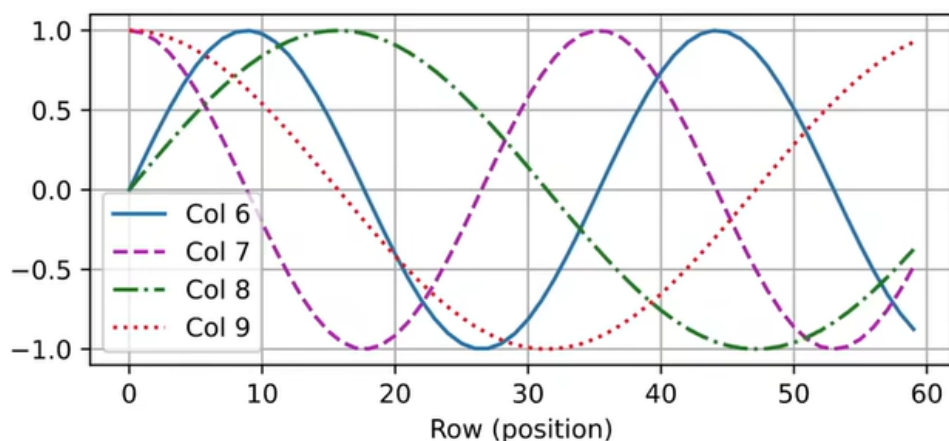
位置编码

概念： 自注意力的位置编码

- 引入：
 - 跟 CNN/RNN 不同，自注意力并没有记录位置信息；
 - 因此可以使用位置编码，将位置信息注入到输入里；
 - 自注意力机制本身是**排列不变**的（Permutation Invariant），即输入序列的顺序变化不会影响输出；
 - 然而，在大多数任务（如 NLP）中，序列顺序是至关重要的语义信息。因此，需要显式地引入位置编码来保留序列的顺序信息；
 - 其不改变注意力机制本身，而是将位置编码信息放入到输入里面；
 - 比如假设长度为 n 的序列，那么使用位置编码矩阵也是同样大小，将这个 P 加到输入 x 里面，作为自编码输入；
- 核心：
 - 位置编码在输入中加入位置信息，使得自注意力能够记忆位置信息；
- 概念：
 - 位置编码通过余弦（Cos）和正弦（Sin）函数，将序列的位置信息嵌入到模型中；



- 这种编码方式的核心思想是利用三角函数的周期性和线性变换特性，捕捉位置间的相对关系，同时支持模型泛化到不同长度的序列；
- 图示：实际的效果



概念：为什么使用 Cos/Sin 进行编码

- 通过三角函数的加法公式，位置 $pos + k$ 的编码可以表示为 pos 和 k 编码的线性组合。例如：

$$\sin\left(\frac{pos + k}{\lambda}\right) = \sin\left(\frac{pos}{\lambda}\right) \cos\left(\frac{k}{\lambda}\right) + \cos\left(\frac{pos}{\lambda}\right) \sin\left(\frac{k}{\lambda}\right)$$

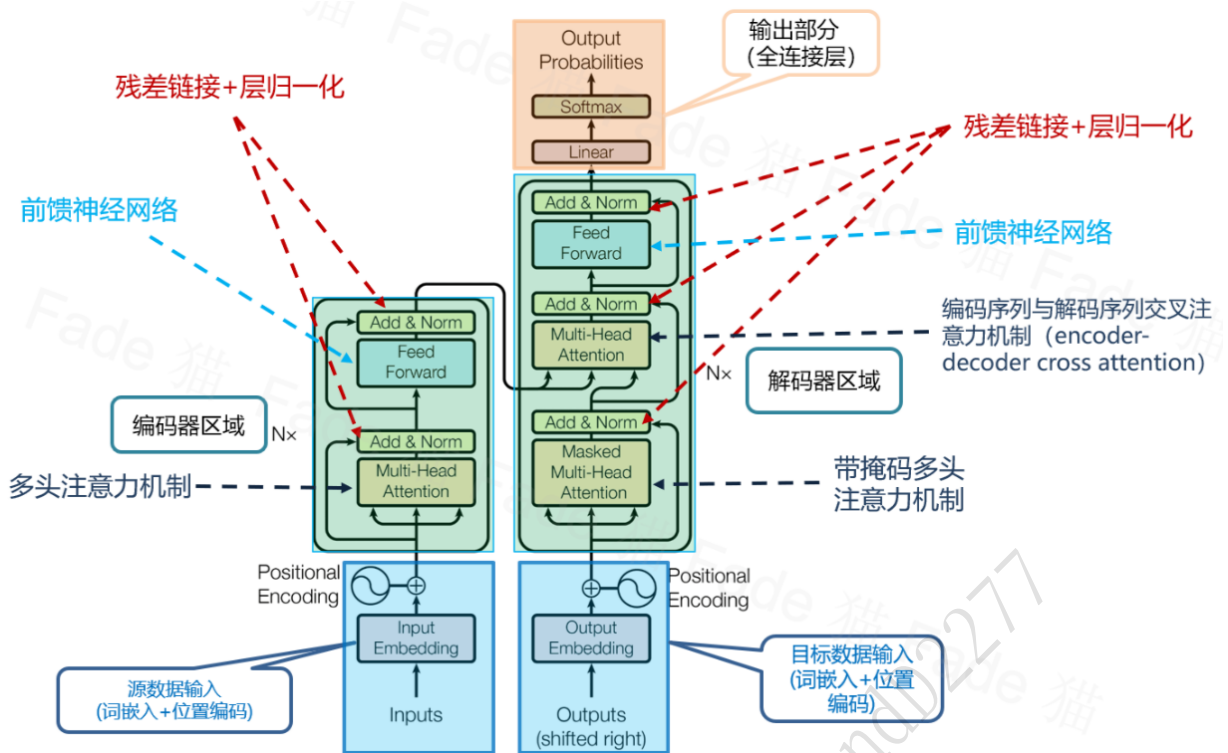
- 这种性质允许模型通过简单的线性变换学习相对位置偏移 k ，而无需显式存储所有可能的相对位置组合；
- 并且，位置编码不依赖训练数据中的最大序列长度，而是通过数学公式动态生成。这使得模型能够处理训练时未见的超长序列，例如在推理阶段处理比训练数据更长的文本，相比可学习的位置嵌入（Learned Positional Embedding），基于 Cos/Sin 的编码具有更好的外推性。例如，即使测试时序列长度远超训练时的最大值，编码依然有效；

Transformer

架构

概念：Transformer

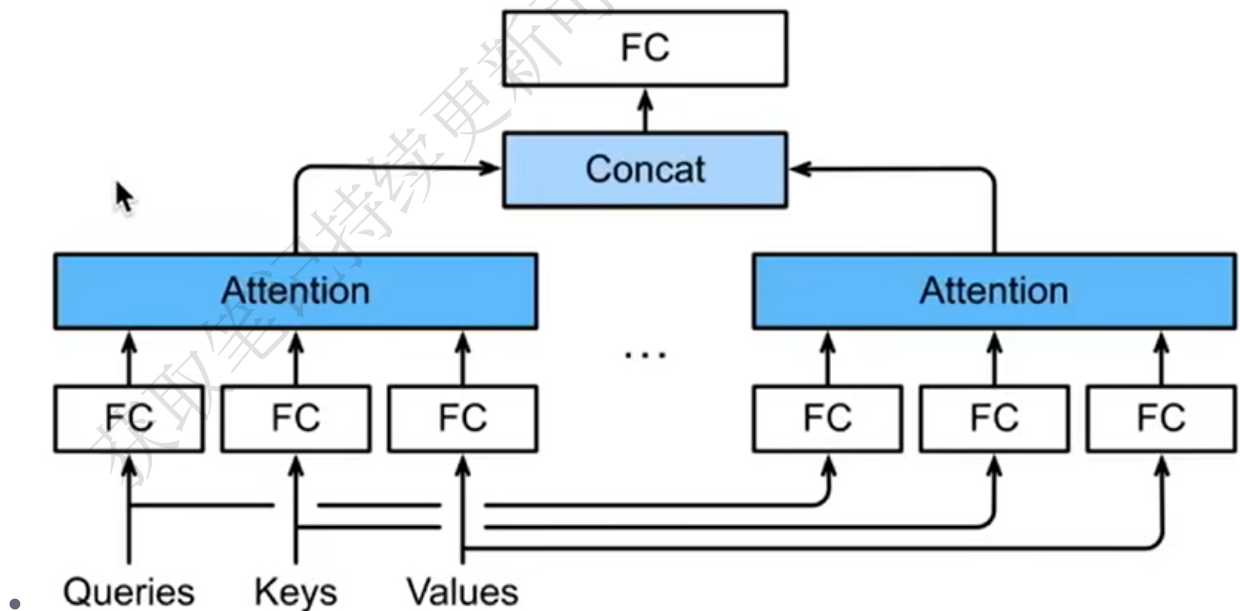
- 概念：Transformer 架构
 - 基于 编码器-解码器 架构来处理序列对；
 - 跟使用注意力的 seq2seq 不同，Transformer 是纯基于自注意力；
- 图示：



多头注意力

概念：多头注意力

- 概念：多头注意力
 - 对同一 (key, value, query)，希望抽取不同的信息，例如短距离关系和长距离关系；

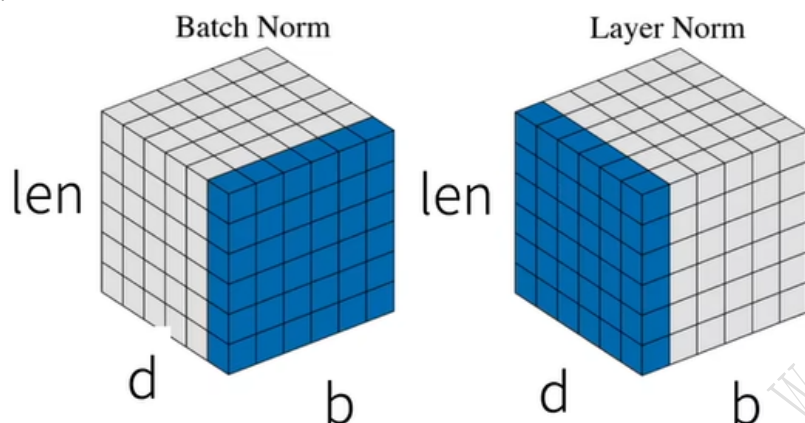


- 对每一个 Attention 做一个全连接，出来之后将其 Concat，然后再进行全连接；
- 概念：有掩码的多头注意力
 - 解码器对序列中一个元素输出时，不应该考虑该元素之后的元素；
 - 可以通过掩码来实现；
 - 也就是计算 x_i 输出时，假装当前序列长度为；

层归一化

概念：层归一化

- 概念：
 - Transformer 中不能使用 Batch Norm，而是要使用 Layer Norm；
 - 因为 Batch Norm 对每一个通道的特征进行归一化，可如何做 NLP 时，序列的长度会变，每次输入内容之后、会产生不稳定；
 - 而层归一化对每个样本里的元素进行归一化，因此其对于样本的长度的处理更加稳定一点；
- 图示：



信息传递

概念：信息传递

- 编码器中的输出 $y_1, \dots, y_n$
- 将其作为解码中第 i 个Transformer块中多头注意力的key和value
 - 它的query来自目标序列
- 意味着编码器和解码器中块的个数和输出维度都是一样的