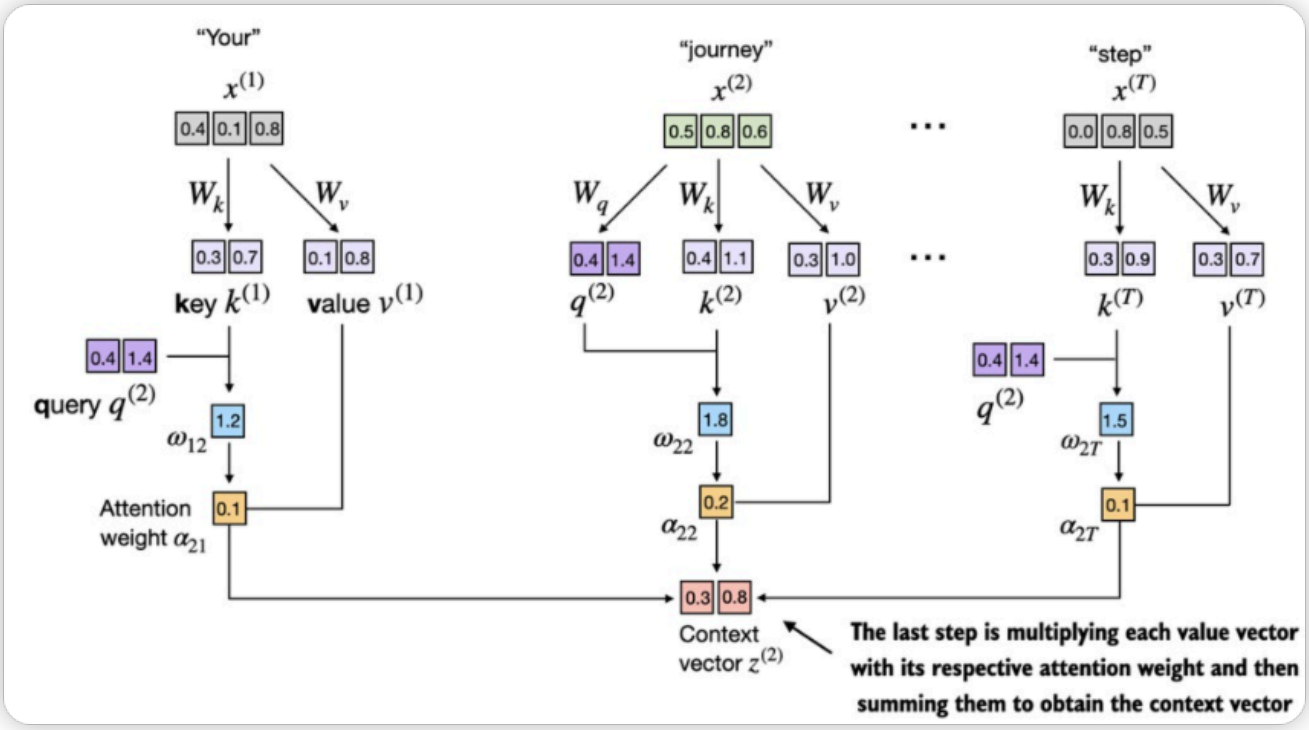
通过三个权重矩阵来转换输入向量。

- 与简单自注意力的区别：模型训练期间会更新权重矩阵，使得模型能够学习“良好”的上下文向量。

步骤

1. 初始查询 W_q ，键 W_k，值 W_v 权重矩阵。
    - 然后 `input` 分别 `矩阵乘法 @` 三个权重矩阵，得到 `queries` ， `keys` ，`values` 三个向量。
2. 计算注意力权重：通过查询 queries 和键 keys 向量
    - 不是简单自注意力直接计算输入向量之间的点积，而是通过：
        - `attn_scores = queries @ keys.T`
    - 缩放点积归一化
        - `attn_weights = torch.softmax(attn_scores / keys.shape[-1] ** 0.5, dim = -1)`
3. 计算上下文向量：通过注意力权重和值 values 向量
    - `context_vec = attn_weights @ values`

exp - 为输入元素 $x^{(2)}$ 计算上下文向量 $z^{(2)}$：



类比查询，键，值向量

- 查询向量W_q： 类似于数据库中的搜索查询。表示模型当前关注或试图理解的项目。
- 键向量W_k： 类似于数据库中的索引和搜索的键。输入序列中的每个项目都有一个关联的键。
- 值向量W_v： 类似于数据库中键值对的值。一旦模型确定哪些输入部分（键）与当前项目（查询）最相关，它就检索相应的值。

实现 Self-Attention 类

```
# 导入PyTorch的神经网络模块
import torch.nn as nn

# 继承自nn.Module
class SelfAttenton_v2(nn.Module):
    # qkv_bias：偏置向量，以提供额外的灵活性和模型的表达能力
    def __init__(self, d_in, d_out, qkv_bias=False):
```

```
        super().__init__()
        self.d_out = d_out
        # v1版本: self.W_query = nn.Parameter(torch.rand(d_in,
d_out))
                # nn.Linear 采用了比 nn.Parameter 更为复杂的权重初始化方
案，且以转置形式存储权重矩阵
        # 定义查询（Query）、键（Key）和值（Value）的线性变换层
        self.W_query = nn.Linear(d_in, d_out, bias=qkv_bias)
        self.W_key = nn.Linear(d_in, d_out, bias=qkv_bias)
        self.W_value = nn.Linear(d_in, d_out, bias=qkv_bias)

    # 前向传播函数，定义了自注意力机制的计算流程
    def forward(self, x):
        # 对输入x应用键（Key）、查询（Query）和值（Value）的线性变换
        # W_key(x) == x @ W_key
        keys = self.W_key(x)
        queries = self.W_query(x)
        values = self.W_value(x)

        # 计算查询和键的注意力分数
        attn_scores = queries @ keys.T  # 使用矩阵乘法计算注意力分数

        # 应用softmax函数对注意力分数进行归一化，得到注意力权重
        # 除以键的维度的平方根是为了进行缩放，防止梯度消失或爆炸
        attn_weights = torch.softmax(
            attn_scores / keys.shape[-1] ** 0.5, dim = -1)

        # 使用注意力权重和值（Value）计算上下文向量
        context_vec = attn_weights @ values  # 使用矩阵乘法计算加权和

        # 返回计算得到的上下文向量
        return context_vec
```