Attention Score

一维样本

腰围：51、56、58 **(Key)**

体重：40、43、48 **(Value)**

如果腰围是57，怎么预测体重？因为有56和58，所以通常都会去取平均值$f(57)=\frac{f(56)+f(58)}{2}=\frac{43+48}{2}$

因为57刚好是56和58的平均数，所以给的权重都是0.5。但是我们没有用上其他的（Key、Value）

假设用α（q，k：）来表示q与k对应的注意力权重，则体重预测值f（q）为：

$f(\mathbf{q},(\mathbf{k}_1,\mathbf{v}_1),\ldots,(\mathbf{k}_3,\mathbf{v}_3))=\sum_{i=1}^{3}\alpha(\mathbf{q},\mathbf{k}_i)\mathbf{v}_i\in\mathbb{R}^v,$

α是任意能刻画相关性的函数，但需要归一化，我们以基于高斯核的注意力分数为例（包括Softmax函数）：

$\alpha(\mathbf{q},\mathbf{k}_i)=\mathrm{softmax}(\frac{1}{2}\dot{(}q-k_i)^2)=\frac{\exp(a(\mathbf{q},\mathbf{k}_i))}{\sum_{j=1}^{3}\exp(a(\mathbf{q},\mathbf{k}_j))}\in\mathbb{R}.$

对于 $k=51$：$\quad\alpha(57,51)=\frac{\exp(-18)}{\exp(-18)+\exp(-0.5)+\exp(-0.5)}\approx 0$

对于 $k=56$：$\quad\alpha(57,56)=\frac{\exp(-0.5)}{\exp(-18)+\exp(-0.5)+\exp(-0.5)}\approx\frac{1}{2}$

对于 $k=58$：$\quad\alpha(57,58)=\frac{\exp(-0.5)}{\exp(-18)+\exp(-0.5)+\exp(-0.5)}\approx\frac{1}{2}$

$f(57)=0\cdot 40+\frac{1}{2}\cdot 43+\frac{1}{2}\cdot 48=\frac{43+48}{2}=45.5$

注意力分数

Softmax：$a(\mathbf{q},\mathbf{k})=\frac{\exp(a_i)}{\sum_{j=1}^{n}\exp(a_j)}$

加性模型 (Additive Attention)：$a(\mathbf{q},\mathbf{k})=\mathbf{v}^T\cdot\tanh(\mathbf{W}_q\mathbf{q}+\mathbf{W}_k\mathbf{k})$，$\mathbf{W}_q$和$\mathbf{W}_k$是可学习的权重矩阵

缩放点积模型 (Scaled Dot-Product Attention)：$a(\mathbf{q},\mathbf{k})=\frac{\mathbf{q}\cdot\mathbf{k}^T}{\sqrt{d_k}}$，$d_k$是$Key$的长度

二维样本

（腰围，胸围）：（（51，70），（56，82），（56，82））**(Key)**

（体重，身高）：（（40，155），（43，159），（48，162））**(Value)**

假设现在给出的Query是二维的：（（57，83），（55，76））

以点积模型为例 : $a(\mathbf{q},\mathbf{k})=\mathbf{q}\cdot\mathbf{k}^T$

Q·K^T·V为$2\cdot 2$矩阵，为了缓解梯度消失，还会除d_k(Scale)

自注意力

Q、K、V是同一个矩阵（单独一章）

Attention Score Code

```python
import math
import torch
from torch import nn
from d2l import torch as d2l

def mask_softmax(X, valid_lens):
    """ 如果没有给定有效长度(valid_lens)，直接在最后一个维度上执行softmax """
    if valid_lens is None:
        return nn.functional.softmax(X, dim=-1)
    else:
        shape = X.shape
        if valid_lens.dim() == 1:
            # 如果有效长度是一维,则扩展
            valid_lens = torch.repeat_interleave(valid_lens, shape[1])
        else:
            # 不是一维则展平成一维
            valid_lens = valid_lens.reshape(-1)
        # On the last axis, replace masked elements with a very large negative (-1e6)
        X = d2l.sequence_mask(X.reshape(-1, shape[-1]), valid_lens,
                              value=-1e6)
        # 对遮蔽后的张量恢复原来的形状并进行softmax运算
        return nn.functional.softmax(X.reshape(shape), dim=-1)

test_mask_softmax_init = torch.rand(2,2,4) # Two dim, Four lines, Three columns
print('test_mask_softmax:',test_mask_softmax_init)

# 第一个的只保留前两列，第二个的只保留前三列，后面的全被mask掉
print('test_mask_softmax:',mask_softmax(test_mask_softmax_init, torch.tensor([2,3])))
# 更细致的划分mask
print('test_mask_softmax:',mask_softmax(test_mask_softmax_init, torch.tensor([[1,3],
[2,4]])))

# Markdown那几个公式
class AdditiveAttention(nn.Module):
    """加性注意力"""
    def __init__(self, key_size, query_size, num_hiddens, dropout, **kwargs):
        super(AdditiveAttention, self).__init__(**kwargs)
        self.W_k = nn.Linear(key_size, num_hiddens, bias=False)
        self.W_q = nn.Linear(query_size, num_hiddens, bias=False)
        self.w_v = nn.Linear(num_hiddens, 1, bias=False)
        self.dropout = nn.Dropout(dropout)

    def forward(self, queries, keys, values, valid_lens):
        queries, keys = self.W_q(queries), self.W_k(keys)
        # 为了计算相似度得分，将查询扩展一个维度，使其形状与键相匹配
        features = queries.unsqueeze(2) + keys.unsqueeze(1)
        features = torch.tanh(features)
        scores = self.w_v(features).squeeze(-1)
        self.attention_weights = mask_softmax(scores, valid_lens)
        # values的shape: (batch_size, num_queries, 1, num_values, embed_size)
```

```python
        # scores的shape: (batch_size, num_queries, num_keys)
        # attention_weights的shape: (batch_size, num_queries, num_keys)
        out = torch.bmm(self.dropout(self.attention_weights), values)
        return out.squeeze(1)


queries, keys = torch.normal(0, 1, (2,1,20)), torch.ones((2,10,2))
values = torch.arange(40, dtype=torch.float32).reshape(1,10,4).repeat(2,1,1)
valid_lens = torch.tensor([2,6])
attention = AdditiveAttention(key_size=2, query_size=20, num_hiddens=8, dropout=0.1)
attention.eval()
attention(queries, keys, values, valid_lens)

d2l.show_heatmaps(attention.attention_weights.reshape((1,1,2,10)),
                  xlabel='Keys', ylabel='Queries')
d2l.plt.show()

class DotProductAttention(nn.Module):
    """缩放点积注意力"""
    def __init__(self, dropout, **kwargs):
        super(DotProductAttention, self).__init__(**kwargs)
        self.dropout = nn.Dropout(dropout)

    def forward(self, queries, keys, values, valid_lens=None):
        d = queries.shape[-1]
        scores = torch.bmm(queries, keys.transpose(1,2)) / math.sqrt(d)
        self.attention_weights = mask_softmax(scores, valid_lens)
        return torch.bmm(self.dropout(self.attention_weights), values)

queries = torch.normal(0,1,(2,1,2))
attention = DotProductAttention(dropout=0.5)
attention.eval()
attention(queries, keys, values, valid_lens)
d2l.show_heatmaps(attention.attention_weights.reshape((1,1,2,10)),
                  xlabel='Keys', ylabel='Queries')
d2l.plt.show()
```