第 3 章 Transformer 模型的注意力机制

第一眼看到这张图片时，你会注意到什么？如果给你一分钟的时间，则会把大部分时间花费在图片的哪个区域上呢（是蒲公英，还是蓝天或者小草）？你最关注的地方便是一种注意力机制，如图 3-1 所示。

图 3-1　蒲公英图

本章将探讨注意力机制的概念及如何把注意力机制应用到机器翻译实例任务上。

3.1　Transformer 模型注意力机制的概念

以机器翻译的实例为例，当将“人工智能”这 4 个汉字输入 Transformer 模型时，这 4 个字经过词嵌入和位置编码后，得到一个[1,4,512]维度的向量。这个向量随后会被传递给 Transformer 模型，如图 3-2 所示。

在 Transformer 模型框架中，最重要的部分便是注意力机制。那么，到底什么是注意力机制呢？

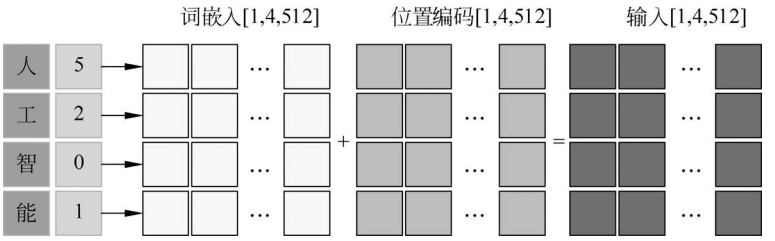


图 3-2　“人工智能”这 4 个汉字经过词嵌入和位置编码后的数据

3.1.1　Transformer 模型的自注意力机制

根据 Transformer 模型的神经网络框架，输入向量首先会被传递给注意力机制层进行注意力机制的计算。在这里，暂时抛开 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三个矩阵，对注意力机制的计算公式进行简化，让 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三个矩阵相等，即都等于输入矩阵 $\boldsymbol{X}$，这样就得到了一个新的数学公式。

$$\text{Attention}(\boldsymbol{X},\boldsymbol{X},\boldsymbol{X})=\text{Softmax}\left(\frac{\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}}{\sqrt{d_k}}\right)\boldsymbol{X} \tag{3-1}$$

式(3-1)中，d_k 是一个缩放系数，是一个常量，这里可以暂时忽略这个常量。那么，这个数学公式代表着什么意思呢？

首先输入矩阵 $\boldsymbol{X}$ 是输入序列经过词嵌入与位置编码后的数据，其矩阵维度为[1,4,512]。根据线性代数的知识，可以轻松地得到输入矩阵 $\boldsymbol{X}$ 的转置矩阵，即将输入矩阵 $\boldsymbol{X}$ 的每行顺时针旋转 90°，形成新矩阵的每列。通过矩阵转置运算，一个[1,4,512]维度的矩阵向量会变成一个[1,512,4]的新矩阵向量，如图 3-3 所示。

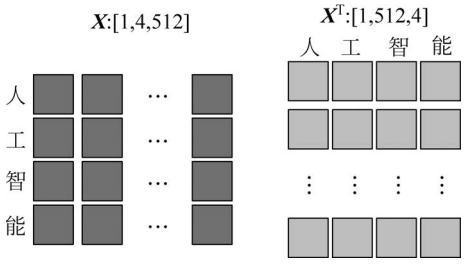


图 3-3　输入矩阵的转置矩阵

根据式(3-1)，将输入矩阵 $\boldsymbol{X}$ 与自己的转置矩阵进行注意力机制的计算称为自注意力机制。

3.1.2　Transformer 模型注意力机制中两个矩阵乘法的含义

基于线性代数的原理，可以利用矩阵乘法来处理输入矩阵与转置矩阵的乘法运算($\boldsymbol{X}\times\boldsymbol{X}^{\mathrm{T}}$)，例如，输入矩阵 $\boldsymbol{X}$ 的第 1 行的每个元素与转置矩阵 $\boldsymbol{X}^{\mathrm{T}}$ 的第 1 列中对应的元素相乘，并对结果求和，从而得到矩阵乘法的结果。类似地，可以按照同样的方式计算输入矩阵 $\boldsymbol{X}$ 的第 1 行与转置矩阵 $\boldsymbol{X}^{\mathrm{T}}$ 的第 2 列、第 3 列和第 4 列的乘积，以获得最终输出矩阵的第 1 行

数据，其他行的数据也可以通过相同的过程计算得出，如图 3-4 所示。

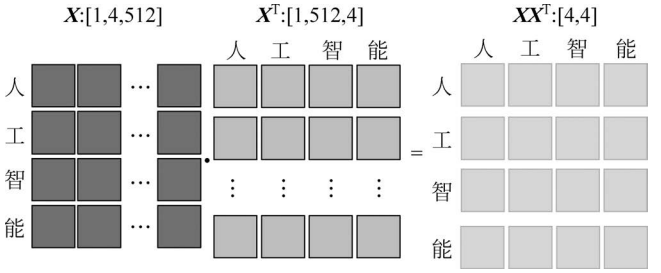


图 3-4　输入矩阵与转置矩阵的乘法运算示意图

针对第 1 行数据，可以观察到，在矩阵乘法中，“人”的词向量分别与“人工智能”这 4 个汉字的词向量进行了乘法运算。那么，这两个向量的乘积代表什么含义呢？

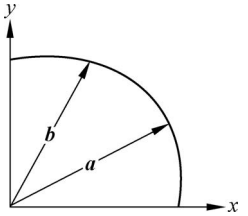


图 3-5　两个矩阵的乘法代表两个向量的夹角

两个向量的乘积也就是它们的内积，表示两个向量之间的夹角，并且还体现了一个向量在另一个向量上的投影。当投影的值越大时，说明这两个向量的相似度越高。如果两个向量的夹角为 90°甚至大于 90°，则这两个向量是线性无关的，完全没有相似性，如图 3-5 所示。

在 Transformer 模型中，这些向量代表了词向量，它们是输入单词经过词嵌入和位置编码后映射到高维空间的数值。当两个词向量的内积较大时，表示这两个单词之间的相关性较高。在 Transformer 模型预测或训练时，当关注某个单词时，应该密切关注与该单词向量内积较大的其他单词。

矩阵 $\boldsymbol{X}\times\boldsymbol{X}^{\mathrm{T}}$ 是一个方阵，在上述例子中为一个[4×4]的方阵。通过注意力机制的计算，可以得到单词“人”与单词“人工智能”这 4 个汉字之间的注意力机制数据。当然，“人”与自身的相似度最高，假设为 9，与“工”的相似度假设为 6，与“智”的相似度假设为 3，而与“能”的相似度假设为 1，如图 3-6 所示。

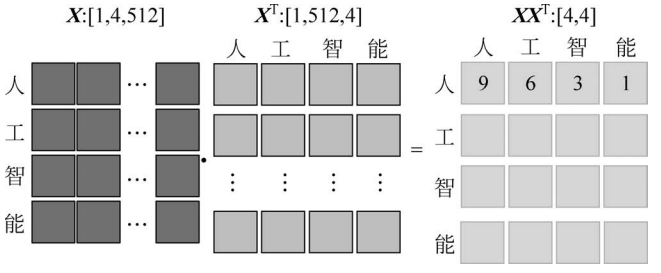


图 3-6　输入矩阵与转置矩阵的乘法结果

两个矩阵的内积数值代表了它们的相似性，数值越大表示与某个单词的相似性越高。在 Transformer 模型中，注意力机制数值较大的地方便是神经网络需要更多关注的地方。

3.1.3　Transformer 模型的 Softmax 操作

根据注意力机制的数学公式，完成注意力计算后，通常需要进行一次 Softmax 操作。Softmax 的主要作用是对模型数据进行归一化，将所有数据映射到[0,1]的范围内，并且各数据之和为 1。另外，这些数值也有另一个称谓，即权重。权重这个词大家应该不陌生，在神经网络模型中，最终的计算结果通过求和得到权重，而这些权重的总和为 1。经过 Softmax 运算后，便得到了单词“人”与其他 4 个汉字“人工智能”的注意力机制概率分布，如图 3-7 所示。

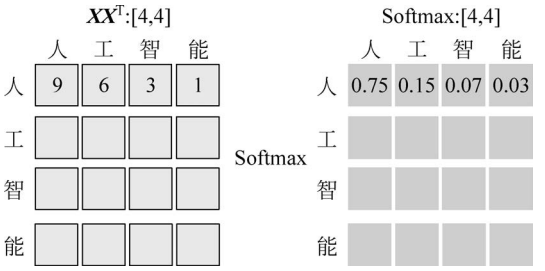


图 3-7　Transformer 模型的 Softmax 操作

当 Transformer 模型关注单词“人”时，应该将 0.75 的注意力分配给它自身，剩下的 0.15 的注意力放在汉字“工”，0.07 的注意力放在汉字“智”，0.03 的注意力放在汉字“能”（需要注意的是，这里的数值只是用于演示，并不是按照标准公式计算得来的）。当然，在 Transformer 模型的训练过程中会对这些权重进行优化。有关优化训练的详细内容将在后续章节中进行讲解。

3.1.4　Transformer 模型的注意力矩阵

在 Transformer 模型中，对于注意力机制的计算公式，将 Softmax 后的注意力矩阵与输入矩阵 $\boldsymbol{X}$ 相乘，代表着什么含义呢？如图 3-8 所示。

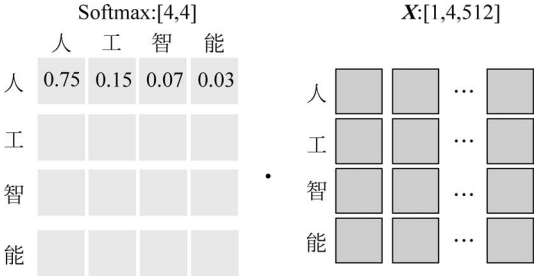


图 3-8　Transformer 模型注意力矩阵与输入矩阵相乘

根据注意力机制的计算公式（见式(3-1)），其注意力矩阵还需要与输入矩阵 $\boldsymbol{X}$ 进行乘法运算。根据矩阵乘法的规则，注意力矩阵的每行与输入矩阵 $\boldsymbol{X}$ 相乘，将得到一个新的矩

阵向量。在这个新的矩阵向量中，每个维度的数值（单词 512 维度的词嵌入向量）都是根据“人工智能”这 4 个字向量加权求和得到的。同时，汉字“人”对应的权重较大，因此在矩阵相乘时会更多地提取原始 $\boldsymbol{X}$ 矩阵中与“人”相关的信息，而权重较小的汉字则在加权求和后的过程中其信息会被减弱。这样得到的新的矩阵向量，也就是汉字“人”通过注意力机制加权求和后的矩阵表示，而且每个维度的数字都与其他汉字建立了一定的联系，如图 3-9 所示。

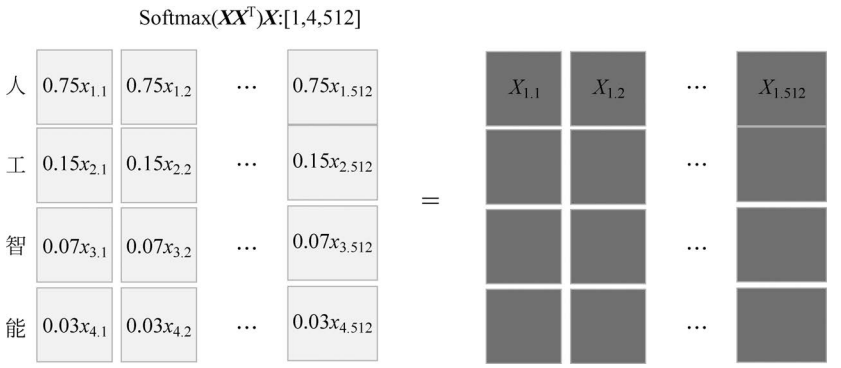


图 3-9　Transformer 模型注意力矩阵与输入矩阵相乘结果示意图

最终结果矩阵按照式(3-2)来计算：

$$\begin{cases}\text{第 1 行第 1 列：}X_{1.1}=0.75x_{1.1}+0.15x_{2.1}+0.07x_{3.1}+0.03x_{4.1}\\ \text{第 1 行第 2 列：}X_{1.2}=0.75x_{1.2}+0.15x_{2.2}+0.07x_{3.2}+0.03x_{4.2}\\ \text{第 1 行第 512 列：}X_{1.512}=0.75x_{1.512}+0.15x_{2.512}+0.07x_{3.512}+0.03x_{1.512}\end{cases}\tag{3-2}$$

同样的计算过程也可以用于计算汉字“工”和词语“智能”经过注意力机制后的新矩阵表示，从而得到一个与输入矩阵 $\boldsymbol{X}$ 维度相同的新矩阵。这个新矩阵是输入矩阵经过注意力机制加权求和后的新表示。从以上计算过程可以看出，经过注意力机制加权求和后的矩阵，每个汉字每个维度的数字都是由与其他汉字维度的数值计算而来，因此每个汉字都与其他汉字建立了联系。

Transformer 模型的训练正是针对注意力机制进行的，而且在这里将 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 都设为输入矩阵 $\boldsymbol{X}$（已知矩阵）。这样，注意力机制的计算公式中就没有未知变量了，这使 Transformer 模型无法训练出有意义的信息，因此，注意力机制的公式中需要 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三个矩阵的存在。那么 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三矩阵又是从何而来的呢？

3.2　Transformer 模型 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三矩阵

Transformer 模型中的注意力机制概念是通过 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 这 3 个矩阵的乘法将每个输入数据转换为一个新的矩阵，其中每个数据都与其他所有单词的数据进行计算，从而建立了所有输入数据之间的联系。然而，当 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三个矩阵都等于输入矩阵 $\boldsymbol{X}$，并且输入矩阵 $\boldsymbol{X}$ 是一个常量时，注意力机制的公式中就不会存在未知变量了，因此经过注意力机制后得到的结

果也将是一个常量。

然而，这样的常量数据无法传入 Transformer 神经网络模型进行相关的数据训练，因为数据本身没有未知变量，而 Transformer 模型也不知道应该训练哪些参数，因此，注意力机制失去了其本质的含义。那么该如何计算注意力呢？

3.2.1　Transformer 模型 Q、K、V 三矩阵的来历

按照机器翻译的示例，假设输入了 4 个汉字："人工智能"。这 4 个汉字经过词嵌入和位置编码后，得到一个[1，4，512]维度的矩阵向量。只有这个矩阵向量才会传递给 Transformer 模型进行训练。根据注意力机制的公式，传递给 Transformer 模型的是 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三个矩阵。

假设有一个 $\boldsymbol{W}_{\mathrm{q}}$ 矩阵，其矩阵维度为[512，512]的方阵。根据矩阵计算的规则，将输入矩阵 $\boldsymbol{X}$ 乘以 $\boldsymbol{W}_{\mathrm{q}}$ 矩阵，将会生成一个[4，512]维度的新矩阵，该矩阵称为 $\boldsymbol{Q}$ 矩阵，如图 3-10 所示。

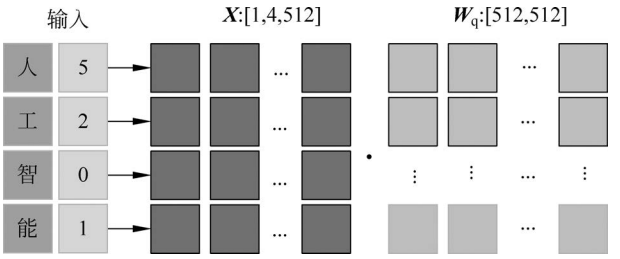


图 3-10　Transformer 模型 $\boldsymbol{Q}$ 矩阵计算方法

同样地，假设有一个 $\boldsymbol{W}_{\mathrm{k}}$ 矩阵，其矩阵维度为[512，512]的方阵。将输入矩阵 $\boldsymbol{X}$ 乘以 $\boldsymbol{W}_{\mathrm{k}}$ 矩阵，将会生成一个[4，512]维度的新矩阵，该矩阵称为 $\boldsymbol{K}$ 矩阵，如图 3-11 所示。

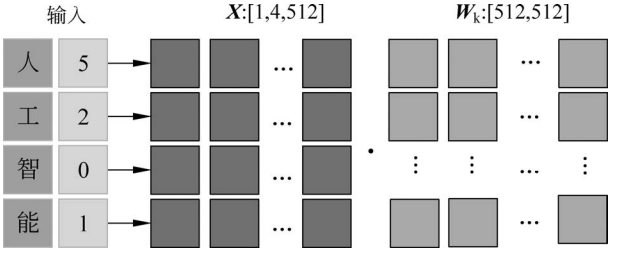


图 3-11　Transformer 模型 $\boldsymbol{K}$ 矩阵计算方法

最后，假设有一个 $\boldsymbol{W}_{\mathrm{v}}$ 矩阵，其矩阵维度为[512，512]的方阵。将输入矩阵 $\boldsymbol{X}$ 乘以 $\boldsymbol{W}_{\mathrm{v}}$ 矩阵，将会生成一个[4，512]维度的新矩阵，该矩阵称为 $\boldsymbol{V}$ 矩阵，如图 3-12 所示。

通过以上计算，就得到了 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 这 3 个矩阵，其矩阵计算公式如式(3-3)所示。

$$\boldsymbol{Q}=\boldsymbol{X}\boldsymbol{W}^{q},\quad \boldsymbol{K}=\boldsymbol{X}\boldsymbol{W}^{k},\quad \boldsymbol{V}=\boldsymbol{X}\boldsymbol{W}^{v} \tag{3-3}$$

而 $\boldsymbol{W}_{\mathrm{q}}$、$\boldsymbol{W}_{\mathrm{k}}$、$\boldsymbol{W}_{\mathrm{v}}$ 这 3 个矩阵是初始化的未知变量，因此 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 经过计算后也必然是一个未知的矩阵。当经过注意力机制的计算后，生成的矩阵也将是未知的矩阵，因此，

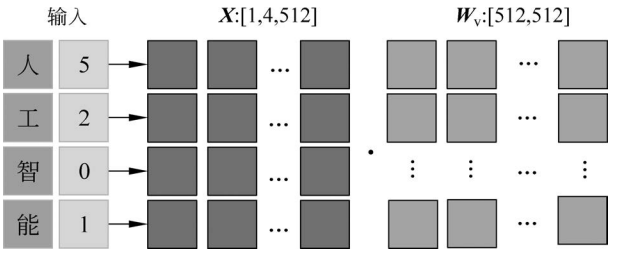


图 3-12 Transformer 模型 V 矩阵计算方法

Transformer 模型的任务就是训练和优化 W_q、W_k、W_v 这 3 个矩阵，以确保经过注意力机制后的矩阵符合预期的输出。

3.2.2 Transformer 模型 Q、K、V 矩阵注意力机制的运算

根据 3.2.1 节得到的 Q、K、V 三个矩阵，就可以使用注意力机制的计算公式来计算注意力。Q 矩阵的维度为[4,512]，而矩阵 K 经过转置后得到一个新的矩阵，其维度为[512,4]。通过将 Q 矩阵乘以 K 矩阵的转置，便得到了注意力机制的注意力矩阵，其维度为[4,4]，如图 3-13 所示。

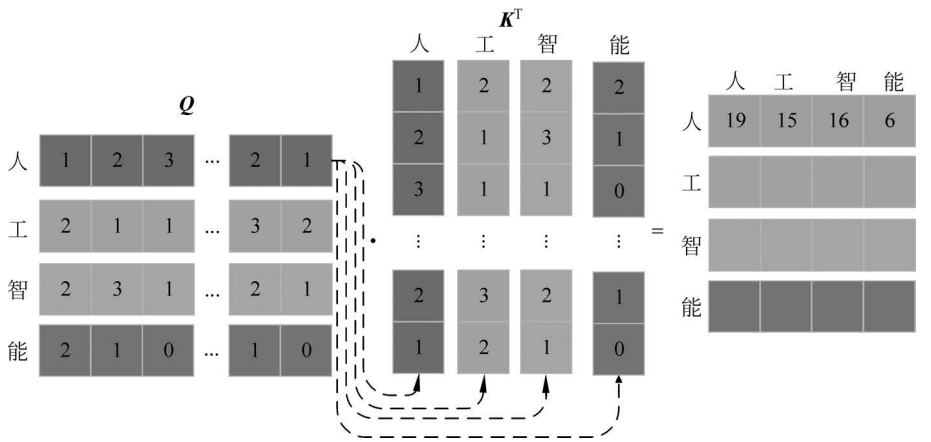


图 3-13 Transformer 模型 Q 矩阵乘以 K 矩阵的转置

当然，在计算注意力之前还会除以一个缩放系数$\sqrt{d_k}$，并对结果进行 Softmax 计算，从而得到“人工智能”这 4 个汉字分别与汉字“人”“工”“智”“能”之间的权重值。值得注意的是，每行权重值的和等于 1。这也解释了为什么需要进行 Pad Mask 的原因，因为模型希望真正存在数据的地方具有权重值，而被掩码的地方权重值为 0。

最后，将注意力机制的矩阵与矩阵 V 相乘，得到一个新的矩阵，其中，注意力矩阵的维度为[4,4]，而矩阵 V 的维度为[4,512]。通过矩阵乘法，得到的新矩阵的维度仍然是[4,512]，这个新矩阵就是经过加权求和后的输入矩阵 X 新向量表示，如图 3-14 所示。

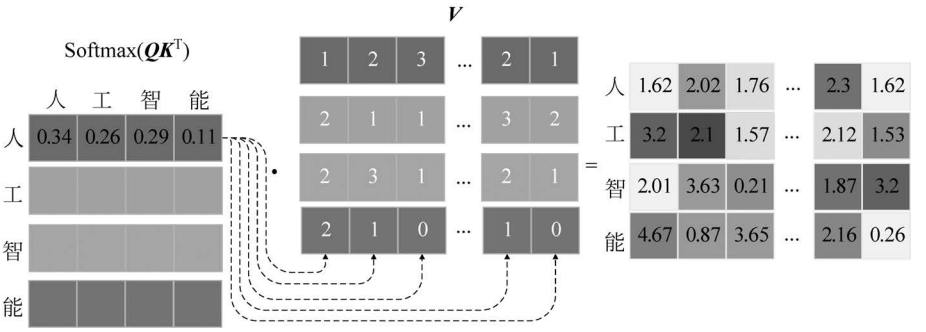


图 3-14　Transformer 模型注意力矩阵乘以 $\boldsymbol{V}$ 矩阵

在整个计算过程中，未知变量是 $\boldsymbol{W}_{\mathrm{q}}$、$\boldsymbol{W}_{\mathrm{k}}$、$\boldsymbol{W}_{\mathrm{v}}$，而 Transformer 模型通过优化不同注意力机制的权重来优化这些未知变量。

3.3　Transformer 模型注意力机制中的缩放点积

从注意力机制的计算公式可以看出，在计算注意力矩阵时，除以一个缩放系数 $\sqrt{d_k}$。那么为什么需要这个缩放系数呢？如果不使用这个系数，则会有什么问题？

3.3.1　Transformer 模型注意力机制的问题

梯度消失问题：神经网络的权重会按照损失的梯度进行更新，但在某些情况下，梯度可能非常小，从而有效地阻止了权重的更新。这会导致神经网络无法进行有效训练，这通常被称为梯度消失问题。

数据 Softmax 操作：假设有一个正态分布，Softmax 的值在很大程度上取决于标准差。如果标准差很大，则 Softmax 函数将只有一个峰值，其他值都趋向于 0。为了更好地可视化这个问题，可以随机生成一些数据，并进行代码的可视化操作，其代码如下：

```
#第 3 章/3.3.1/创建均值为 0、标准差为 100 的正态分布
import torch
import numpy as np
import torch.nn as nn
import matplotlib.pyplot as plt
a = np.random.normal(0,100,size=(20000))        #创建均值为 0、标准差为 100 的数据
plt.hist(a)
plt.show()                                      #可视化
```

代码执行后，其输出如图 3-15 所示。

其 Softmax 操作的代码如下：

```
#第 3 章/3.3.1/创建均值为 0、标准差为 100 的正态分布，并执行 Softmax 操作
import torch
import numpy as np
```

```
import torch.nn as nn
import matplotlib.pyplot as plt
a = np.random.normal(0,100,size=(20000))      #创建均值为 0、标准差为 100 的正态分布
attn = nn.Softmax(dim=-1)(torch.from_numpy(a))          #执行 Softmax 操作
plt.plot(attn)
plt.show()                                                #可视化
```

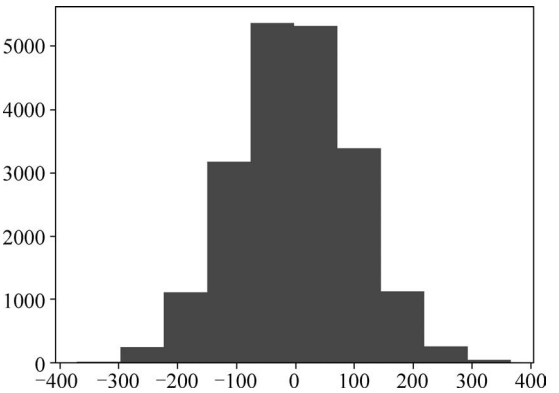


图 3-15　均值为 0、标准差为 100 的正态分布数据

利用上述数据进行一层 Softmax 操作，并可视化经过 Softmax 后的数据。从可视结果可以看出，在 Softmax 操作后只存在一个有效的值 1，其他值都为 0。这意味着注意力权重消失，模型很难进行有效学习，如图 3-16 所示。

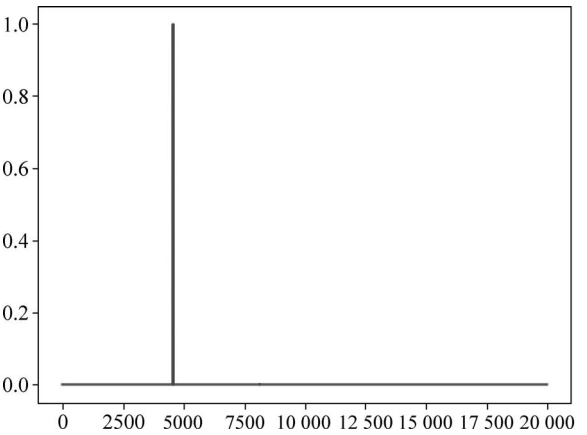


图 3-16　均值为 0、标准差为 100 的数据经过 Softmax 可视化

3.3.2　Transformer 模型注意力机制的缩放点积

缩放点积操作的目的是确保点积的值不会因为向量维度的增加而变得过大。通过引入缩放因子，可以将点积的值控制在一个合适的范围内，使 Softmax 函数能够更好地处理注意力权重。可以通过创建一个均值为 0，标准差为 100 的正态分布，并将其标准差缩放为 1

来说明这一点，其代码如下：

```
#第 3 章/3.3.2/创建一个均值为 0，标准差为 100 的正态分布，并将其标准差缩放为 1
import torch
import numpy as np
import torch.nn as nn
import matplotlib.pyplot as plt
a = np.random.normal(0, 100, size=(20000))     #创建一个均值为 0，标准差为 100 的数据
b = a / 100
#创建一个包含两个子图的画布
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
#绘制 a 的直方图
ax1.hist(a)
ax1.set_title('Histogram of a')
#绘制 b 的直方图
ax2.hist(b)
ax2.set_title('Histogram of b')
#调整子图之间的间距
plt.tight_layout()
#显示图形
plt.show()
```

代码执行后，两种数据分布的直方图完全相同，只是数据的标准差不同(一种是 100，另一种是 1)，如图 3-17 所示。

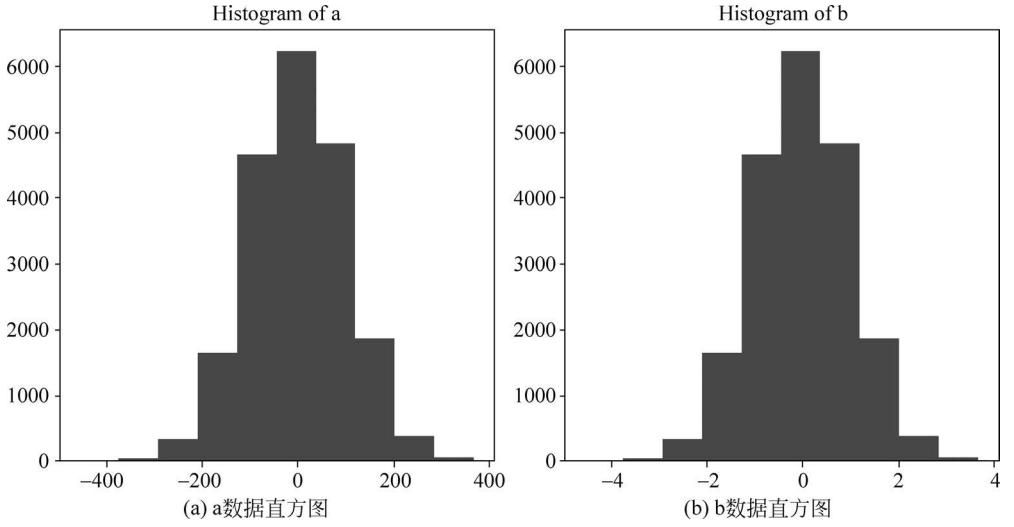


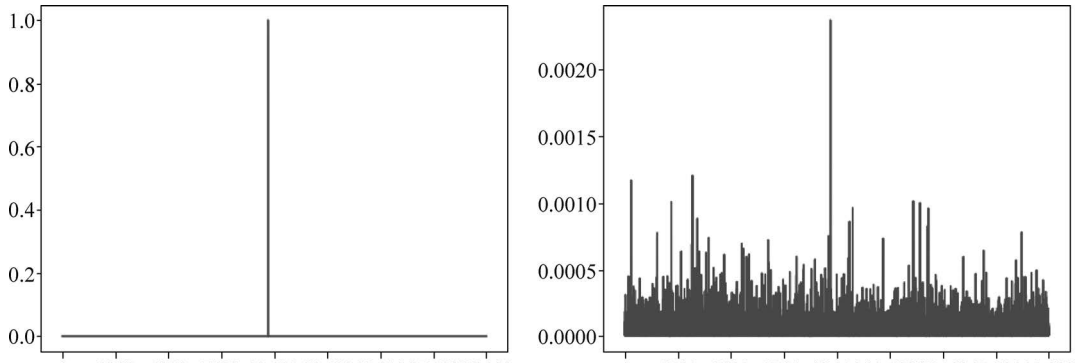
图 3-17　均值为 0 的数据可视化

然后观察这两种数据经过 Softmax 操作后的数值，代码如下：

```
#第 3 章/3.3.2/Softmax 操作
attn = nn.Softmax(dim=-1)(torch.from_numpy(a))          #Softmax 操作
plt.plot(attn)
```

```
plt.show()
attn_b = nn.Softmax(dim=-1)(torch.from_numpy(b))     #Softmax 操作
plt.plot(attn_b)
plt.show()
```

代码执行后如图 3-18 所示，可以看到，经过缩放点积后的数据，在经过 Softmax 操作后呈现出更加分散、分布更加均匀的特点，不再局限于单个数值，这有助于有效地改善模型的学习机制，避免梯度消失或者梯度爆炸问题。



(a) 数据标准差为100　　(b) 数据标准差为1

图 3-18　数据 Softmax 操作可视化

3.4　Transformer 模型注意力机制的代码实现过程

按照标准的注意力机制公式来编写 Transformer 模型最关键的注意力机制的代码，其代码如下：

```
#第 3 章/3.4/注意力机制代码实现过程-第一部分
import torch
import torch.nn as nn
import math
import numpy as np
import matplotlib.pyplot as plt
input = torch.LongTensor([[5,2,1,0,0],[1,3,1,4,0]])     #初始化一个变量
src_vocab_size = 10                                      #输入词表长度
d_model = 512                                            #word-embedding 维度
d_k = d_v = 64                                           #多头注意力机制维度
word_embr = word_emb(input)                  #[2,5,512],输入数据经过词嵌入
pes = pe(word_embr).transpose(0, 1)          #[2,5,512],输入数据添加位置编码
enc_pad_mask = get_attn_pad_mask(input,input)            #[2,5,5] 获取 Pad Mask 矩阵
```

首先，初始化输入一个 LongTensor 变量，其矩阵维度为[2,5]，其中，2 代表输入两个句子，5 代表每个句子有 5 个汉字。假设在矩阵中，数字 0 代表 Pad Mask，然后引入 word_emb()函数和 pe()函数，这些函数是第 2 章中介绍的词嵌入和位置编码函数。输入数据需

要经过词嵌入和位置编码后才能传递给 Transformer 的注意力机制模块。经过以上操作后，数据的维度变为[2,5,512]。由于输入数据存在 Pad Mask，所以代码使用 get_attn_pad_mask()函数获取 Pad Mask 矩阵，以便在进行注意力机制计算时使用。如果是解码器的输入，则需要计算 Sequence Mask 矩阵。

```
#第 3 章/3.4/注意力机制代码实现过程-第二部分
class ScaledDotProductAttention(nn.Module):
    def __init__(self):
        super(ScaledDotProductAttention, self).__init__()
    def forward(self, Q, K, V, attn_mask):
        #Q: [batch_size, len_q, d_k]                    #[2,5,512] 定义 Q 矩阵
        #K: [batch_size, len_k, d_k]                    #[2,5,512]定义 K 矩阵
        #V: [batch_size, len_v(=len_k), d_v]            #[2,5,512] 定义 V 矩阵
        #attn_mask: [batch_size, seq_len, seq_len]      #[2,5,5] 定义 mask 矩阵
        #scores:[batch_size,len_q,len_k]                #[2,5,5]注意力矩阵
        #根据注意力机制计算公式计算 Q 乘以 K 的转置矩阵，再除以根号下 d_k
        scores = torch.matmul(Q, K.transpose(-1, -2)) / np.sqrt(d_k)
        if attn_mask is not None: #若存在 mask，则添加 mask 矩阵
            scores.masked_fill_(attn_mask, -1e9)       #[2,5,5]
        attn = nn.Softmax(dim=-1)(scores)       #对最后一个维度(v)执行 Softmax 操作
        #result: [batch_size, len_q, d_v]               #[2,5,512]
        #根据注意力机制的公式，注意力矩阵再乘以 V 矩阵
        result = torch.matmul(attn, V)                 #[2,5,512]
        return result, attn                            #attn 注意力矩阵(用于可视化)
```

接下来，建立一个注意力机制计算函数，该函数接受 4 个参数，分别是 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三个矩阵，以及掩码矩阵。$\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三个矩阵是通过将输入矩阵 $\boldsymbol{X}$ 经过线性变换得到的新矩阵表示，其维度仍然是[2, 5, 512]，而掩码矩阵的维度为[2, 5, 5]。

根据注意力机制的计算公式，让矩阵 $\boldsymbol{Q}$ 乘以 $\boldsymbol{K}$ 矩阵的转置，并除以一个缩放系数$\sqrt{d_k}$，得到注意力矩阵，其维度为[2 ,5, 5]。在这一步中，还需要判断是否存在掩码矩阵。如果存在掩码矩阵，则需要将掩码的位置置为一个很小的负无穷数，这样在进行 Softmax 计算时，掩码的位置的数值就会变为 0，而其他没有被掩码的位置的数值和为 1。最后，根据注意力机制的计算公式，将计算得到的注意力矩阵乘以输入矩阵 $\boldsymbol{V}$，就得到了最终的结果矩阵。

建立完成注意力机制实现函数后，可以初始化一下数据，并进行注意力机制的可视化操作，其代码如下：

```
#第 3 章/3.4/注意力机制代码输出可视化
Q = K = V = X                                          #初始化 Q、K、V 矩阵，使它们都等于 X
self_attention = ScaledDotProductAttention()           #初始化注意力机制函数
atten_result, atten = self_attention(Q,K,V,enc_pad_mask)      #计算注意力机制
print('atten_result',atten_result)                     #打印输出结果
print('atten_result',atten_result.shape)               #打印最终输出的数据形状
#数据可视化
import matplotlib.pyplot as plt
#将注意力矩阵的形状从 [batch_size, len_q, len_k] 转换为 [len_q, len_k]
```

```
attention_matrix = atten[0].detach().NumPy()
#绘制热力图
plt.imshow(attention_matrix, cmap='viridis', interpolation='nearest')
#添加颜色条
plt.colorbar()
#添加坐标轴标签
plt.xlabel('len_k')
plt.ylabel('len_q')
#显示图形
plt.show()
```

让输入矩阵 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 都等于输入矩阵 $\boldsymbol{X}$，并将 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三个矩阵与 Pad Mask 矩阵一起传递给注意力机制函数。在这里，可以打印出经过注意力机制计算后的结果矩阵及其形状。由于注意力机制不改变输入矩阵的维度，所以经过注意力机制的计算后，矩阵的维度仍然是[2,5,512]，其输出如下：

```
atten_result tensor([[
        [ 1.2291, 1.3626, 2.1573, ..., 3.4164, -1.2502, 0.5883],
        [ 0.6696, 0.3658, 2.0497, ..., -0.1866, 0.7938, 0.1699],
        [ 1.8404, -0.1952, 0.9078, ..., 0.0304, 1.6829, -0.0272],
        [ 1.7838, -0.1680, 0.9630, ..., 0.0200, 1.6399, -0.0177],
        [ 1.8168, -0.1839, 0.9308, ..., 0.0261, 1.6650, -0.0233]],
       [[ 1.1720, 0.8458, 0.2195, ..., 0.0304, 1.6828, -0.0272],
        [ 1.5178, 0.0592, 0.9711, ..., -0.1349, -0.4596, 0.7501],
        [ 1.8346, -0.1861, 0.9018, ..., 0.0304, 1.6829, -0.0272],
        [ 0.9450, -0.2576, 1.6519, ..., 0.2529, -0.5878, 1.4342],
        [ 1.3575, -0.2256, 1.3054, ..., 0.1499, 0.4633, 0.7577]]],
      grad_fn=<UnsafeViewBackward>)
atten_result torch.Size([2, 5, 512])
```

当然，也可以对注意力机制的矩阵进行可视化，使用热力图表示，不同颜色表示注意力权重的大小，颜色越亮表示权重越大，说明注意力数值越大。代码可视化执行后，其输出如图 3-19 所示。

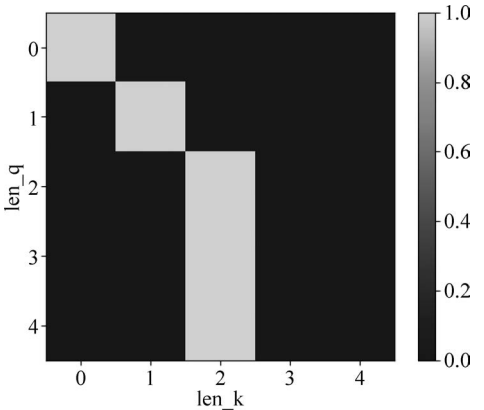


图 3-19　Transformer 模型注意力矩阵可视化

3.5　Transformer 模型多头注意力机制

相比于多头注意力机制，3.4 节中介绍的注意力机制可以看作其中的一头。那么为什么需要多头注意力机制呢？实际上，这与我们对人、事、物的评估类似，不同的人对同一个人或同一件事可能会有不同的看法。如果只听从某个人的意见，则必然会偏离事实，而如果多人评价同一件事情，综合多个人的观点，就能更接近真相。多头注意力机制也是基于同样的道理，它使用多个矩阵来关注相同的输入矩阵，并最终通过综合多个头的权重信息来获取最终的输出权重，从而获得更有效的注意力。那么在 Transformer 模型中，多头注意力机制是如何处理的呢？

3.5.1　Transformer 模型多头注意力机制的计算公式

仍以机器翻译为例。输入 Transformer 模型的仍然是“人工智能”这 4 个汉字，通过词嵌入和位置编码后，生成一个维度为[1，4，512]的输入向量矩阵 $\boldsymbol{X}$。只有经过这样的处理，输入矩阵 $\boldsymbol{X}$ 才会被传递到 Transformer 模型中，并进行多头注意力机制的计算，其多头注意力机制的计算公式如下：

$$\begin{cases}\text{MultiHead}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})=\text{Concat}(\text{head}_1,\text{head}_2,\cdots,\text{head}_n)\boldsymbol{W}^{\boldsymbol{O}}\\ \text{where head}_i=\text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^{\boldsymbol{Q}},\boldsymbol{K}\boldsymbol{W}_i^{\boldsymbol{K}},\boldsymbol{V}\boldsymbol{W}_i^{\boldsymbol{V}})\end{cases} \tag{3-4}$$

式(3-4)中，$\boldsymbol{W}_i^{\boldsymbol{Q}}\in\mathbf{R}^{d_{\text{model}}\times d_k}$，$\boldsymbol{W}_i^{\boldsymbol{K}}\in\mathbf{R}^{d_{\text{model}}\times d_k}$，$\boldsymbol{W}_i^{\boldsymbol{V}}\in\mathbf{R}^{d_{\text{model}}\times d_v}$，$\boldsymbol{W}^{\boldsymbol{O}}\in\mathbf{R}^{hd_v\times d_{\text{model}}}$，而多头的维度如式(3-5)所示，都是 64。

$$d_k=d_v=\frac{d_{\text{model}}}{h}=64 \tag{3-5}$$

根据多头注意力机制的计算公式，每个头可以看作一个注意力机制。在式(3-4)中，有 3 个变量，这 3 个变量的计算公式如下：

$$\begin{cases}\boldsymbol{Q}_i=\boldsymbol{Q}\boldsymbol{W}_i^{\boldsymbol{Q}}\\ \boldsymbol{K}_i=\boldsymbol{K}\boldsymbol{W}_i^{\boldsymbol{K}}\\ \boldsymbol{V}_i=\boldsymbol{V}\boldsymbol{W}_i^{\boldsymbol{V}}\end{cases} \tag{3-6}$$

将它们代入注意力机制的公式中，就可以得到每个头的注意力计算公式，其公式如下：

$$\text{head}_i=\text{Attention}(\boldsymbol{Q}_i,\boldsymbol{K}_i,\boldsymbol{V}_i)=\text{Softmax}\left(\frac{\boldsymbol{Q}_i\boldsymbol{K}_i^{\mathrm{T}}}{\sqrt{d_k}}\right)\boldsymbol{V}_i \tag{3-7}$$

其中，i 表示多头注意力机制的头数，在 Transformer 模型中，将其定义为 8。

3.5.2　Transformer 模型 $\boldsymbol{Q}_i$、$\boldsymbol{K}_i$、$\boldsymbol{V}_i$ 的来历

通常情况下，$\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三个矩阵都等于输入矩阵 $\boldsymbol{X}$，其维度为[4,512]，因此，假设有一个 $\boldsymbol{W}_{\boldsymbol{q}_0}$ 矩阵，其维度为[512,64]，将其与输入矩阵 $\boldsymbol{Q}$ 相乘($\boldsymbol{Q}\times\boldsymbol{W}_{\boldsymbol{q}_0}$)，得到一个新的矩阵 $\boldsymbol{Q}_0$，其

维度为[4,64]，如图 3-20 所示。

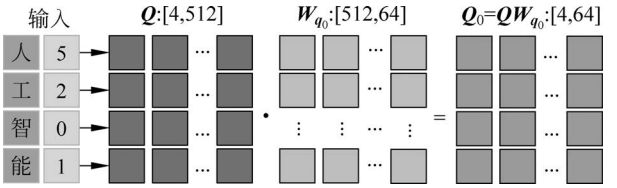


图 3-20　Transformer 模型 $\boldsymbol{Q}_0$ 矩阵的计算过程

同理，假设有一个 $\boldsymbol{W}_{k_0}$ 矩阵，其维度为[512,64]，将其与输入矩阵 $\boldsymbol{K}$ 相乘($\boldsymbol{K}\times\boldsymbol{W}_{k_0}$)，得到一个新的矩阵 $\boldsymbol{K}_0$，其维度为[4,64]，如图 3-21 所示。

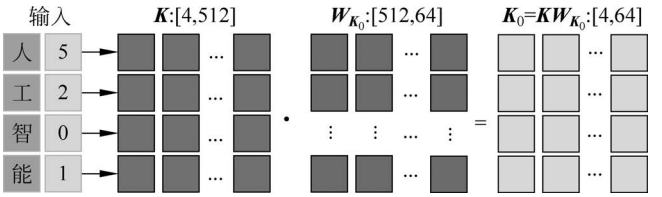


图 3-21　Transformer 模型 $\boldsymbol{K}_0$ 矩阵的计算过程

假设有一个 $\boldsymbol{W}_{v_0}$ 矩阵，其维度为[512,64]，将其与输入矩阵 $\boldsymbol{V}$ 相乘($\boldsymbol{V}\times\boldsymbol{W}_{v_0}$)，得到一个新的矩阵 $\boldsymbol{V}_0$，其维度为[4,64]，如图 3-22 所示。

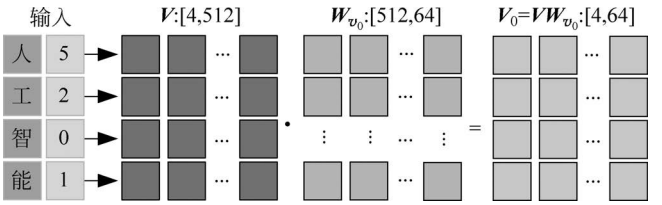


图 3-22　Transformer 模型 $\boldsymbol{V}_0$ 矩阵的计算过程

通过以上运算，就得到了多头注意力机制中第 1 个头的 $\boldsymbol{Q}_0$、$\boldsymbol{K}_0$、$\boldsymbol{V}_0$ 三矩阵，然后根据多头注意力机制的计算公式，就可以得到经过注意力机制计算后的 $\boldsymbol{Z}_0$ 矩阵，其维度为[4,64]。计算公式如下：

$$\boldsymbol{Z}_0=\text{head}_0=\text{Attention}(\boldsymbol{Q}_0,\boldsymbol{K}_0,\boldsymbol{V}_0)=\text{Softmax}\left(\frac{\boldsymbol{Q}_0\boldsymbol{K}_0^{\mathrm{T}}}{\sqrt{d_k}}\right)\boldsymbol{V}_0 \tag{3-8}$$

同样的方法，假设有一个 $\boldsymbol{W}_{q_1}$ 矩阵，$\boldsymbol{W}_{k_1}$ 与 $\boldsymbol{W}_{v_1}$ 矩阵。使其与输入 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三矩阵分别做乘法，就得到了多头注意力机制中的第 2 个头的 $\boldsymbol{Q}_1$、$\boldsymbol{K}_1$、$\boldsymbol{V}_1$ 三矩阵。再根据多头注意力机制的计算公式，就可以得到经过注意力机制计算后的 $\boldsymbol{Z}_1$ 矩阵，其维度为[4,64]。计算公式如下：

$$\boldsymbol{Z}_1=\text{head}_1=\text{Attention}(\boldsymbol{Q}_1,\boldsymbol{K}_1,\boldsymbol{V}_1)=\text{Softmax}\left(\frac{\boldsymbol{Q}_1\boldsymbol{K}_1^{\mathrm{T}}}{\sqrt{d_k}}\right)\boldsymbol{V}_1 \tag{3-9}$$

当然，由于是多头注意力机制，其头数为 8，因此假设有 8 个 $\boldsymbol{W}_q$ 矩阵，输入矩阵 $\boldsymbol{Q}$ 乘以

8 个 $\boldsymbol{W}_q$ 矩阵，便得到了 8 个 $\boldsymbol{Q}$ 矩阵，分别是$[\boldsymbol{Q}_0,\boldsymbol{Q}_1,\cdots,\boldsymbol{Q}_7]$，计算公式如下：

$$[\boldsymbol{Q}_0,\boldsymbol{Q}_1,\cdots,\boldsymbol{Q}_7]=\boldsymbol{Q}\boldsymbol{W}_i^Q,\quad i=[0,1,\cdots,7] \tag{3-10}$$

$\boldsymbol{W}_k$ 矩阵同样也有 8 个，输入矩阵 $\boldsymbol{K}$ 乘以 8 个 $\boldsymbol{W}_k$ 矩阵，便得到了 8 个 $\boldsymbol{K}$ 矩阵，分别是$[\boldsymbol{K}_0,\boldsymbol{K}_1,\cdots,\boldsymbol{K}_7]$，计算公式如下：

$$[\boldsymbol{K}_0,\boldsymbol{K}_1,\cdots,\boldsymbol{K}_7]=\boldsymbol{K}\boldsymbol{W}_i^K,\quad i=[0,1,\cdots,7] \tag{3-11}$$

$\boldsymbol{W}_v$ 矩阵同样也有 8 个，输入矩阵 $\boldsymbol{V}$ 乘以 8 个 $\boldsymbol{W}_v$ 矩阵，便得到了 8 个 $\boldsymbol{V}$ 矩阵，分别是$[\boldsymbol{V}_0,\boldsymbol{V}_1,\cdots,\boldsymbol{V}_7]$，计算公式如下：

$$[\boldsymbol{V}_0,\boldsymbol{V}_1,\cdots,\boldsymbol{V}_7]=\boldsymbol{V}\boldsymbol{W}_i^V,\quad i=[0,1,\cdots,7] \tag{3-12}$$

通过以上的计算，便得到了多头注意力机制中 8 个头的 $\boldsymbol{Q}_i$、$\boldsymbol{K}_i$、$\boldsymbol{V}_i$ 三矩阵。

3.5.3　Transformer 模型多头注意力机制的计算

根据注意力机制的计算公式，可以计算每个头经过注意力机制后的矩阵，一共得到 8 个矩阵，分别是$[\boldsymbol{Z}_0,\boldsymbol{Z}_1,\cdots,\boldsymbol{Z}_7]$。每个 $\boldsymbol{Z}$ 矩阵的维度都为[4,64]，如图 3-23 所示。

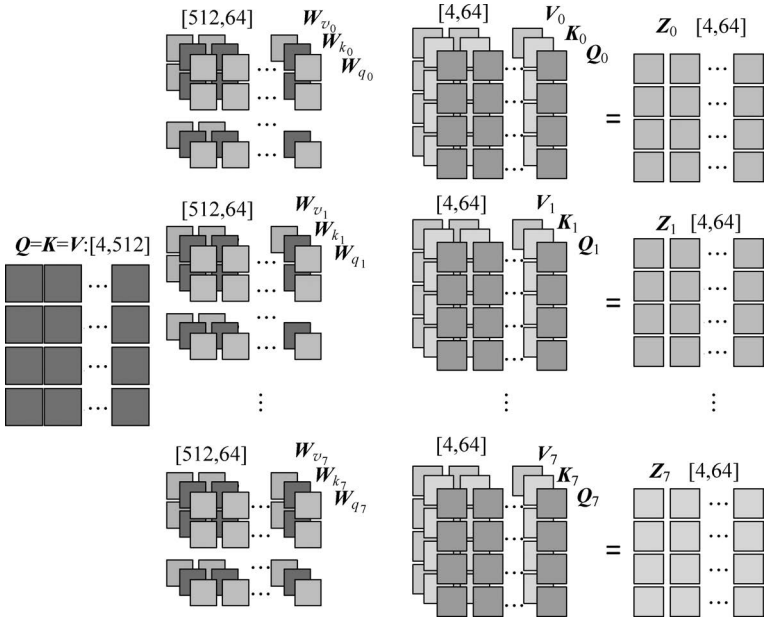


图 3-23　Transformer 模型多头注意力机制的计算过程

根据多头注意力机制的计算公式（见式(3-4)），使用 Concat 方法将$[\boldsymbol{Z}_0,\boldsymbol{Z}_1,\cdots,\boldsymbol{Z}_7]$这 8 个输出矩阵合并在一起，得到一个新的矩阵 $\boldsymbol{Z}$，其维度为[4,512]，其计算公式如下：

$$\boldsymbol{Z}=\mathrm{Concat}(\mathrm{head}_1,\mathrm{head}_2,\cdots,\mathrm{head}_n) \tag{3-13}$$

假设有一个矩阵 $\boldsymbol{W}_o$，其维度为[512,512]，将输出矩阵 $\boldsymbol{Z}$ 与 $\boldsymbol{W}_o$ 矩阵相乘，得到经过多头注意力机制后的矩阵，矩阵维度仍然是[4,512]，其计算公式如下：

$$\mathrm{MultiHead}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})=\boldsymbol{Z}\boldsymbol{W}_o \tag{3-14}$$

当然，在这里假设 batch-size 为 1，所以实际的矩阵维度为[1,4,512]，通过以上的计算过程便得到了最终多头注意力机制的结果，如图 3-24 所示。

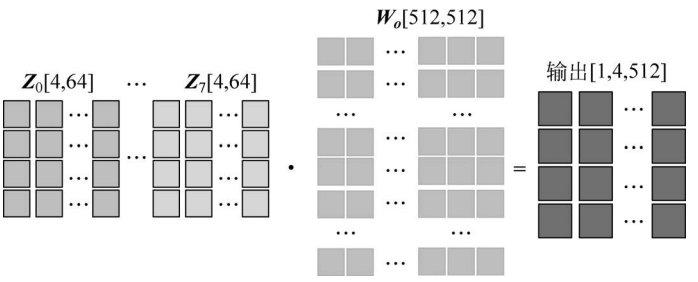


图 3-24　Transformer 模型多头注意力机制的计算结果

本节所讲解的多头注意力机制，其中未知参数是 W_o 矩阵及 8 个 W_q、W_k、W_v 矩阵，而 Transformer 模型对以上未知矩阵进行了训练与优化。

3.6　Transformer 模型多头注意力机制的代码实现

与注意力机制的代码相比，多头注意力机制的代码是在原有基础上计算多个头的 Q、K、V 矩阵，然后进行注意力机制的计算。

3.6.1　Transformer 模型多头注意力机制的代码

多头注意力机制的代码如下：

```
#第 3 章/3.6.1/多头注意力机制代码实现-第一部分
import torch
import torch.nn as nn
import math
import numpy as np
import matplotlib.pyplot as plt
input = torch.LongTensor([[5,2,1,0,0],[1,3,1,4,0]])
src_vocab_size = 10
d_model = 512
word_embr = word_emb(input)                              #词嵌入
word_embr = word_embr.transpose(0, 1)
d_k = d_v = 64
pes = pe(word_embr).transpose(0, 1)                      #位置编码 [2,5,512]
enc_pad_mask = get_attn_pad_mask(input,input)            #Pad Mask 矩阵 [2,5,5]
n_heads = 8
input_Q = input_K = input_V = pes
```

输入数据仍然是 LongTensor([[5,2,1,0,0], [1,3,1,4,0]])变量，经过词嵌入和位置编码后，将输出数据传递给多头注意力机制，其中数字“0”代表 Pad Mask。最后，将输入 Q、K、V 三个矩阵都设为输入矩阵 X，有了 Q、K、V 三个矩阵，就可以分别计算多头注意力机制

的矩阵。当然，还使用了 get_attn_pad_mask 方法获取 Pad Mask 矩阵。

```
#第 3 章/3.6.1/多头注意力机制代码实现-第二部分
class MultiHeadAttention(nn.Module):
    def __init__(self):
        super(MultiHeadAttention, self).__init__()
        #初始化注意力机制中需要的 Wq、Wk、Wv 及 Wo 矩阵，矩阵维度为[512,512]
        self.W_Q = nn.Linear(d_model, d_k *n_heads, bias=False)     #[512,512]
        self.W_K = nn.Linear(d_model, d_k *n_heads, bias=False)     #[512,512]
        self.W_V = nn.Linear(d_model, d_v *n_heads, bias=False)     #[512,512]
        self.W_O = nn.Linear(n_heads *d_v, d_model, bias=False)     #[512,512]
    def forward(self, input_Q, input_K, input_V, attn_mask):
        #输入 Q、K、V 三矩阵，矩阵维度都为 [2,5,512]
        #input_Q: [batch_size, len_q, d_model]                      #[2,5,512]
        #input_K: [batch_size, len_k, d_model]                      #[2,5,512]
        #input_V: [batch_size, len_v(=len_k), d_model]              #[2,5,512]
        #计算 Pad Mask 矩阵，矩阵维度为[2,5,5]
        #attn_mask: [batch_size, seq_len, seq_len]                  #[2,5,5]
        #设置 residual，便于计算残差连接
        residual, batch_size = input_Q, input_Q.size(0)             #[2,5,512]
        #B: batch_size, S:seq_len, D: dim
        #(B,S,D)-proj-> (B,S,D_new)-
        #split->(B,S,Head,W)-trans->(B,Head,S,W)
        #获取多头注意力机制中的 8 个 Q、K、V 矩阵，每个矩阵的维度都为[2,8,5,64]
        #Q: [batch_size, n_heads, len_q, d_k]                       #[2,8,5,64]
        Q = self.W_Q(input_Q).view(batch_size, -1, n_heads, d_k).transpose(1, 2)
        #K: [batch_size, n_heads, len_k, d_k]                       #[2,8,5,64]
        K = self.W_K(input_K).view(batch_size, -1, n_heads, d_k).transpose(1, 2)
        #V: [batch_size, n_heads, len_v(=len_k), d_v] #[2,8,5,64]
        V = self.W_V(input_V).view(batch_size, -1, n_heads, d_v).transpose(1, 2)
        #attn_mask:[batch_size,seq_len,seq_len] ->->->->
        #->->->->->->[batch_size,n_heads,seq_len,seq_len]
        #重复 8 次，得到 8 个头的 Pad Mask 矩阵
        attn_mask = attn_mask.unsqueeze(1).repeat(1, n_heads, 1, 1)   #[2,8,5,5]
        #result:[batch_size,n_heads,len_q,d_v]                      #[2,8,5,64]
        #attn:[batch_size,n_heads,len_q, len_k]                     #[2,8,5,5]
        #计算多头注意力机制
        result, attn = ScaledDotProductAttention()(Q, K, V, attn_mask)

        #result:[batch_size,n_heads,len_q,d_v]->[batch_size,len_q,n_heads *d_v]
        #contat heads #result 2*5*512
        result = result.transpose(1, 2).reshape(batch_size, -1, n_heads *d_v)
        #乘以 Wo 矩阵，合并 8 个头的输出数据
        output = self.W_O(result) #[batch_size, len_q, d_model]    #[2,5,512]
        #执行残差连接后，再做一次数据归一化操作，就可以传递给下一层的神经网络模型了
        return nn.LayerNorm(d_model)(output + residual), attn      #[2,5,512]
```

这里定义了一个 MultiHeadAttention 多头注意力机制函数，并在初始化部分定义了 4 个线性变换矩阵，分别是 $\boldsymbol{W}_q$、$\boldsymbol{W}_k$、$\boldsymbol{W}_v$、$\boldsymbol{W}_o$。这 4 个矩阵是 Transformer 模型需要训练的未

知参数矩阵，每个矩阵的维度为[512，512]。每个头的 $\boldsymbol{W}_{q_i}$、$\boldsymbol{W}_{k_i}$、$\boldsymbol{W}_{v_i}$ 矩阵的维度都为[512，64]。在代码实现中，8个头的 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 矩阵一起计算，因此 $\boldsymbol{W}_q$、$\boldsymbol{W}_k$、$\boldsymbol{W}_v$ 矩阵的维度都为[512，512]，其计算公式如下：

$$\boldsymbol{W}_q = \text{Concat}(\boldsymbol{W}_{q_0}, \boldsymbol{W}_{q_1}, \cdots, \boldsymbol{W}_{q_7}) \tag{3-15}$$

$$\boldsymbol{W}_k = \text{Concat}(\boldsymbol{W}_{k_0}, \boldsymbol{W}_{k_1}, \cdots, \boldsymbol{W}_{k_7}) \tag{3-16}$$

$$\boldsymbol{W}_v = \text{Concat}(\boldsymbol{W}_{v_0}, \boldsymbol{W}_{v_1}, \cdots, \boldsymbol{W}_{v_7}) \tag{3-17}$$

式(3-15)、式(3-16)、式(3-17)中 $\boldsymbol{W}_{q_i}$、$\boldsymbol{W}_{k_i}$、$\boldsymbol{W}_{v_i}$ 矩阵维度都为[512，64]，因此 $\boldsymbol{W}_q$、$\boldsymbol{W}_k$、$\boldsymbol{W}_v$ 的数据维度都为[512，512]。

实现函数接受4个参数，分别是输入矩阵 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 和 Pad Mask 矩阵。$\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三矩阵的维度与输入矩阵 $\boldsymbol{X}$ 的维度相同，均为[2，5，512]，而 Pad Mask 的矩阵维度为[2，5，5]。

(1) 2：代表输入两个句子。

(2) 5：代表每个句子中有5个汉字。

(3) 512：词嵌入的数据维度。

然后使用变量 residual 保存输入矩阵和变量 batch_size。residual 变量是为了 Transformer 模型中的残差连接操作，关于这点稍后会在后续章节中进行说明。接下来，使用输入矩阵 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 与 $\boldsymbol{W}_q$、$\boldsymbol{W}_k$、$\boldsymbol{W}_v$ 矩阵进行线性变换，得到了8个头的 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三矩阵，每个头的矩阵维度都为[2，5，64]，而代码中的 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 三个矩阵的维度为[batch_size，n_heads，len_q，d_v]，对于这个例子来讲，维度为[2,8,5,64]。

(1) 2：Batch Size，输入两个句子。

(2) 8：head num，一共有8个头。

(3) 5：len_q，每个句子中有5个汉字(这里包含 Pad Mask)。

(4) 64：d_v，每个头的词嵌入维度。

由于采用了多头注意力机制，所以在计算注意力机制之前，需要将 Pad Mask 矩阵拆分成多个头的 Pad Mask。首先，Pad Mask 的矩阵维度为[2,5,5]，利用 unsqueeze(1)函数增加一个维度，此时矩阵维度为[2,1,5,5]，然后将第二维度复制8次，便得到了多头注意力机制的 Pad Mask 矩阵，其维度为[2,8,5,5]。

得到8个头的 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 矩阵后，就可以使用注意力机制的代码来计算注意力机制了。将输入矩阵 $\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 及 Pad Mask 矩阵传递给3.4节介绍的注意力机制函数，计算注意力，并将结果返回。注意力机制不会改变输入矩阵的维度，因此输出矩阵的维度依然为[2,8,5,64]。

最后，通过 transpose 函数对结果变量的第二维度、第三维度进行转换，使矩阵维度变为[2,5,8,64]，然后通过 reshape 函数对多头的输出维度进行转换，将矩阵维度转换为[2,5,512]。这样就得到了经过多头注意力机制的结果。再根据多头注意力机制的公式，将结果矩阵通过 $\boldsymbol{W}_o$ 矩阵进行一次线性变换，其矩阵维度仍为[2,5,512]。

代码到这里为止，多头注意力机制的代码已经完整实现。在这里，只需返回结果变量，但是根据 Transformer 模型的架构，每次数据通过一个模块时都需要进行残差连接和数据

归一化操作，因此最后一行代码执行了残差连接与数据归一化操作，以便传递给下一层的神经网络。

最后，初始化多头注意力机制函数，并传递相关的输入矩阵。可以打印输出结果矩阵和结果矩阵维度。可以看到，经过多头注意力机制后，矩阵的维度保持不变，依然为[2，5，512]，输出如下：

```
m_head_atten_result tensor([[
        [-0.1605, 1.2608, -0.6856, ..., 0.5043, -1.6757, 1.2376],
        [-0.3512, -0.1073, 1.3214, ..., 1.4473, -0.8715, 0.8557],
        [ 1.5346, -0.9605, 1.6794, ..., 0.3909, 0.6375, 1.9169],
        [-1.2765, -0.8159, -0.1087, ..., 0.5661, 1.1516, 1.4668],
        [-1.9795, -0.4878, -0.8385, ..., 0.5787, 1.1683, 1.4647]],
       [[ 0.7732, 0.2312, 0.8715, ..., 0.2745, 0.9548, 2.0081],
        [ 2.0831, 0.4099, -0.6964, ..., 0.5363, 0.5820, 0.6014],
        [ 1.5541, -1.0806, 1.6650, ..., 0.2298, 0.9527, 1.9963],
        [ 0.0689, -0.8965, -0.3886, ..., 0.4475, 1.1963, 0.9367],
        [-2.0188, -0.6554, -0.8316, ..., 0.4140, 1.4297, 1.6245]]],
      grad_fn=<NativeLayerNormBackward>)
torch.Size([2, 5, 512])
```

3.6.2　Transformer 模型多头注意力矩阵可视化

得到多头注意力矩阵后，可以按照以下代码进行热力图的可视化操作，其可视化代码如下：

```
#第 3 章/3.6.2/Transformer 模型多头注意力矩阵可视化
import matplotlib.pyplot as plt
#将注意力矩阵的形状从 [batch_size, n_heads, len_q, len_k]
#转换为 [n_heads, len_q, len_k]
attention_matrices = attn.detach().NumPy()          #转换到 NumPy 数据格式
n_heads = attention_matrices.shape[1]               #获取注意力机制头数
#可视化多头注意力机制
n_rows = (n_heads + 1) //2
n_cols = min(2, n_heads)
fig, axes = plt.subplots(n_rows, n_cols, figsize=(12, 8))
for i in range(n_heads):
   attention_matrix = attention_matrices[0, i]      #可视化第 1 个 batch
   #[1, i]可视化第 2 个 batch
    row = i //n_cols
    col = i % n_cols
    #在当前子图中绘制热力图
    im = axes[row, col].imshow(attention_matrix, cmap='hot', interpolation=
'nearest')
    axes[row, col].set_xlabel('len_k')
    axes[row, col].set_ylabel('len_q')
```

```
    fig.colorbar(im, ax=axes)
    plt.tight_layout()
    plt.show()
```

由于是多头注意力机制，所以代码会生成 8 个头的注意力矩阵（矩阵维度为[2，8，5，5]），并且此处的 Batch Size 为 2，因此会有 16 个注意力矩阵。首先，使用代码 attention_matrix＝attention_matrices[0，i]来可视化第 1 个 batch 的 8 个注意力矩阵。从注意力矩阵可以看出，颜色越深的地方表示对应位置的注意力权重越大。注意，在第 1 个 batch 后面的两个数字“0”处为 Pad Mask，经过 Softmax 操作后，对应的注意力机制权重为 0，可视化结果如图 3-25 所示。

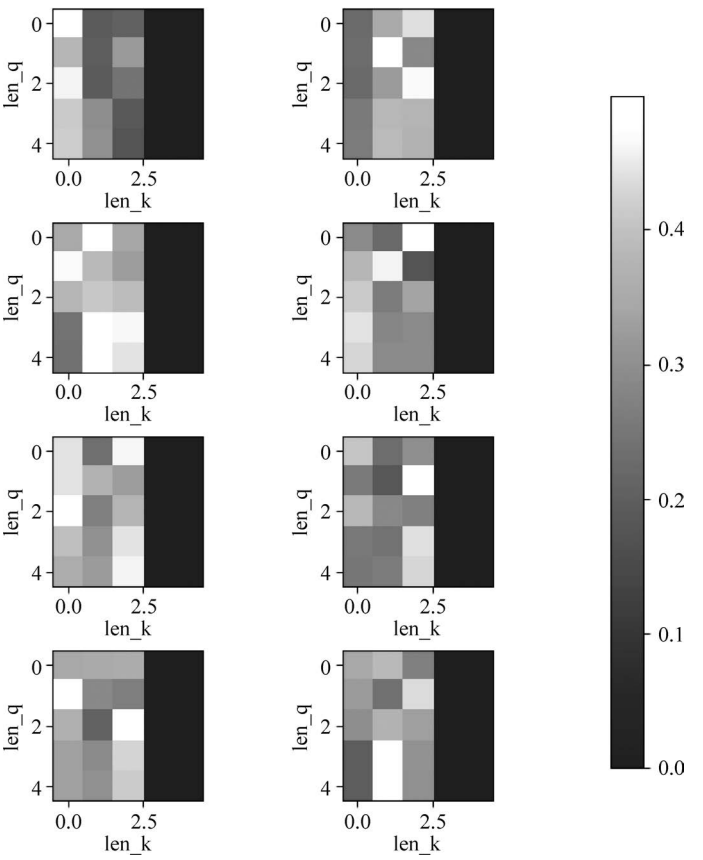


图 3-25　Transformer 模型多头注意力矩阵可视化结果(1)

接下来，可以通过修改代码 attention_matrix ＝ attention_matrices[1，i]来可视化第 2 个 batch 的多头注意力矩阵热力图。从可视化结果可以看出，注意力权重也随着颜色变深而增大。同样，在第 2 个 batch 后面的一个数字“0”处为 Pad Mask，经过 Softmax 操作后，对应的注意力机制权重为 0，可视化结果如图 3-26 所示。

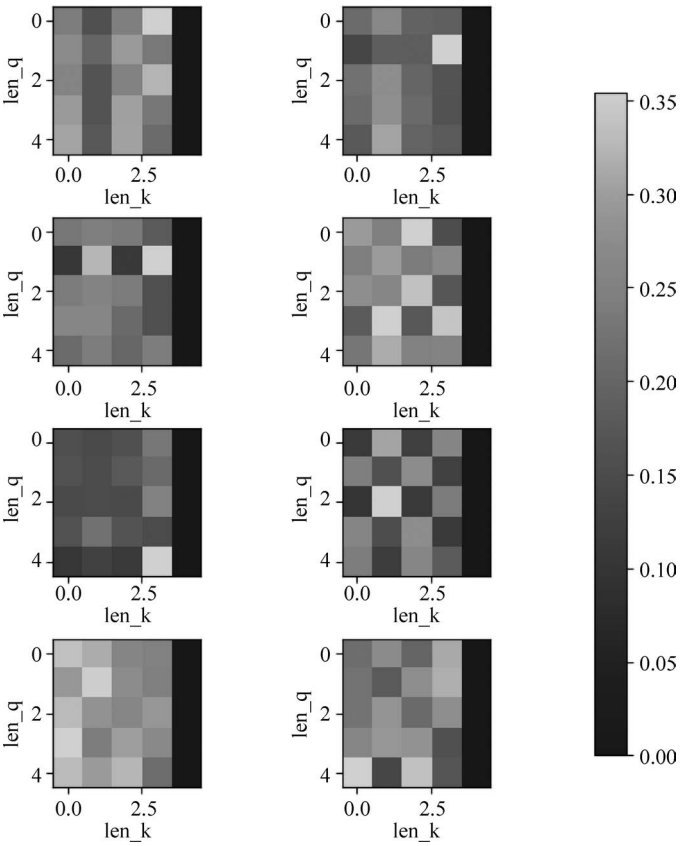


图 3-26　Transformer 模型多头注意力矩阵可视化结果(2)

3.7　本章总结

本章主要介绍了 Transformer 模型中的注意力机制和多头注意力机制的概念和实现，以及其在自然语言处理中的应用。

首先，介绍了注意力机制的概念。注意力机制是指在处理序列数据时，模型能够自动学习不同位置的权重，从而选择性地关注与当前任务相关的信息。在 Transformer 模型中，注意力机制是通过计算输入序列中各个位置之间的相关性来实现的。

接下来，介绍了自注意力机制。自注意力机制是指在计算注意力权重时，只关注输入序列本身，而不需要引入其他的上下文信息。自注意力机制可以有效地捕捉输入序列中长距依赖的信息，是 Transformer 模型中的核心组件之一。

在自注意力机制的基础上，本章进一步地介绍了注意力机制中的两个矩阵乘法操作，其中，第 1 个矩阵乘法操作用于计算输入序列中各个位置之间的相关性，也称为点积注意力；第 2 个矩阵乘法操作用于计算输入序列中各个位置的注意力权重，也称为 Softmax 操作。

Softmax操作是将点积注意力得到的权重矩阵进行归一化处理，使每个位置的权重值在0～1，并且所有位置的权重值之和为1。Softmax操作可以使模型更加关注与当前任务相关的信息，同时也可以避免梯度消失的问题。

在计算注意力权重时，本章介绍了使用缩放点积注意力的原因。由于点积注意力在计算过程中会产生较大的值，从而导致Softmax操作后的梯度变小，进而影响模型的训练效果。为了解决这个问题，在计算点积注意力时引入了一个缩放因子，使点积注意力的值在一个合适的范围内，从而提高了模型的训练稳定性。

接下来，本章介绍了多头注意力机制。多头注意力机制是将自注意力机制扩展到多个子空间中，每个子空间都有自己的注意力权重，从而使模型能够更好地捕捉输入序列中的多种语义信息。多头注意力机制可以提高模型的表达能力和泛化能力，是Transformer模型中的另一个核心组件之一。

最后，本章介绍了注意力机制和多头注意力机制的代码实现。在实现过程中，需要计算$\boldsymbol{Q}$矩阵、$\boldsymbol{K}$矩阵和$\boldsymbol{V}$矩阵，其中$\boldsymbol{Q}$矩阵用于计算注意力权重，$\boldsymbol{K}$矩阵和$\boldsymbol{V}$矩阵用于计算输出序列。在计算$\boldsymbol{Q}$矩阵、$\boldsymbol{K}$矩阵和$\boldsymbol{V}$矩阵时，需要使用线性变换和激活函数等操作，以提高模型的表达能力。