NLP方向总结-矩阵

点积Dot product

用·或• \bullet•表示，\cdot或\bullet

对于两个向量 $\vec{a} = (a_1, a_2, \dots, a_n)$ 和 $\vec{b} = (b_1, b_2, \dots, b_n)$，它们的点积表示为 $\vec{a} \cdot \vec{b}$，它等于 $\sum_{i=1}^{n} a_i b_i$，即两个向量对应位置上的元素相乘后再求和。点积的结果是一个标量（scalar），它代表了两个向量的相似程度或夹角的余弦值。

点积可以被用于计算两个向量之间的相似度

```
import torch

# 定义两个张量
x = torch.tensor([1, 2, 3])
y = torch.tensor([4, 5, 6])

# 计算点积
dot_product = torch.dot(x, y)

# 输出结果
print(dot_product)
# tensor(32)
```

需要注意的是，点积操作要求两个张量具有相同的形状。在本例中，x和y都是形状为(3,)的一维张量，因此它们可以进行点积运算。如果两个张量的形状不同，将无法进行点积运算。

要求两个张量具有相同的形状

内积inner product

内积是一种更一般的点积形式，它可以应用于更广泛的空间和函数。在向量空间中，内积可以表示为 $\langle \vec{a}, \vec{b} \rangle$，其中 $\vec{a}$ 和 $\vec{b}$ 是向量，它等于 $\sum_{i=1}^{n} a_i b_i$，与点积的定义相同。内积可以扩展到更高维空间中，比如矩阵空间和函数空间，用于描述两个对象的相似性和距离等。

内积表示两个向量的相似程度，当内积为零时，表示两个向量垂直。

也称为点积或数量积，和点积基本相似

内积可以用于描述不同特征之间的关系

哈达玛积可以用于执行逐元素的操作，例如将两个矩阵逐元素相乘得到一个新矩阵。

```
import torch

# 定义两个张量
x = torch.tensor([1, 2, 3])
y = torch.tensor([4, 5, 6])

# 计算内积
inner_product = torch.inner(x, y)

# 输出结果
print(inner_product)
# tensor(32)
```

要求两个张量具有相同的形状

外积（Outer Product）

```
import torch

# 创建两个向量
a = torch.tensor([1, 2, 3])
b = torch.tensor([4, 5, 6])

# 计算外积
c = torch.ger(a, b)

print(c)
#tensor([[ 4,  5,  6],
        [ 8, 10, 12],
        [12, 15, 18]])
```

外积是一个向量与另一个向量的转置的矩阵积。外积的结果是一个形状为$(n, m)$的矩阵，其中$n$是第一个向量的长度，$m$是第二个向量的长度。在NumPy中，可以使用numpy.outer函数实现外积运算。

外积则表示两个向量所在平面的法向量。

外积（也称向量积或叉积，叉乘）是向量运算中的一种，通常用于计算两个三维向量的垂直于这两个向量所在平面的向量。

\times 表示

哈达玛积Hadamard Product

哈达玛积是两个矩阵或向量对应位置上的元素相乘后得到的新矩阵或新向量，它的定义与点积和内积不同。对于两个相同大小的矩阵 $A$ 和 $B$，它们的哈达玛积表示为 $A \odot B$，其中 $(A \odot B){i,j} = A{i,j} \cdot B_{i,j}$，即矩阵 $A$ 和 $B$ 的对应位置上的元素相乘后得到新矩阵的对应位置上的元素。哈达玛积常常用于矩阵元素级别的运算，例如对两个矩阵进行逐元素比较、逐元素相加或相乘等。

点积和内积都是向量之间的运算，而哈达玛积是矩阵或向量元素级别的运算，矩阵乘法则是一种矩阵之间的运算。

元素乘"（element-wise multiplication）和哈达玛积（Hadamard product）在概念上是相同的，它们都是指代两个矩阵或向量的元素之间进行逐个相乘的操作。

实际上，"元素乘"一般指的是对矩阵或向量的所有元素逐一进行相乘的操作，而哈达玛积则是特指两个具有相同形状的矩阵逐个元素相乘得到的新矩阵。
换句话说，哈达玛积是一种特殊的元素乘法运算，它只适用于具有相同形状的矩阵或向量。在某些领域，例如图像处理，矩阵或向量元素级别的操作非常常见，而哈达玛积在这些领域中也经常被使用。

```
import torch

# 定义两个张量
x = torch.tensor([1, 2, 3])
y = torch.tensor([4, 5, 6])

# 计算哈达玛积
h_product = torch.mul(x, y)

# 输出结果
print(h_product)
# tensor([ 4, 10, 18])
```

要求两个张量具有相同的形状

.或⊙ \odot⊙

矩阵乘法

*表示, @表示矩阵乘法

矩阵乘法，如果2个元素的维度是1维，那么其结果等于dot 点积, 二维矩阵乘法就是每个向量的内积

又叫矩阵积

矩阵乘法指的是将一个矩阵的每一行与另一个矩阵的每一列进行点积，然后将结果相加得到一个新的矩阵。这个新的矩阵的行数等于第一个矩阵的行数，列数等于第二个矩阵的列数。矩阵乘法在线性代数和计算机科学中有着广泛的应用，特别是在机器学习和深度学习中，它通常用于实现神经网络的前向传播和反向传播等重要操作。

```
import torch

# 定义两个张量
x = torch.tensor([[1, 2], [3, 4], [5, 6]])
y = torch.tensor([[7, 8], [9, 10]])

# 计算矩阵乘法
m_product = torch.matmul(x, y)

# 输出结果
print(m_product)
# tensor([[ 25,  28],
        [ 57,  64],
        [ 89, 100]])
```

矩阵乘法操作要求两个矩阵满足乘法规则，即第一个矩阵的列数等于第二个矩阵的行数。在本例中，x是形状为(3, 2)的矩阵，y是形状为(2, 2)的矩阵，它们满足乘法规则，因此可以进行矩阵乘法运算。如果两个矩阵的形状不满足乘法规则，将无法进行矩阵乘法运算。

Kronecker乘积

克朗克乘积，表示⊗

也叫张量积Tensor Product

Kronecker乘积（Kronecker product）是一个在线性代数中使用的运算，它是两个矩阵的乘积的一种扩展形式。假设有两个矩阵 $A$ 和 $B$，它们的维度分别为 $m \times n$ 和 $p \times q$，则它们的Kronecker乘积表示为：

其中，$a_{ij}$ 是 $A$ 的第 $i$ 行第 $j$ 列的元素，$B$ 乘以一个矩阵就是 $B$ 中的每个元素乘以该矩阵。
Kronecker乘积可以用来描述两个矩阵的复合操作，它的性质包括分配律、结合律和乘法交换律，可以应用于线性方程组的求解、信号处理、多元时间序列分析等领域。

A \otimes B=\left[\begin{array}{ccc}
a_{11} B & \cdots & a_{1 n} B \\
\vdots & \ddots & \vdots \\
a_{m 1} B & \cdots & a_{m n} B
\end{array}\right]_{m p \times n q}

```
import torch

def kronecker_product(a, b):
    m, n = a.size()
    p, q = b.size()
    out = torch.zeros(m*p, n*q)
    for i in range(m):
        for j in range(n):
            out[i*p:(i+1)*p, j*q:(j+1)*q] = a[i,j] * b
    return out

# 示例使用
A = torch.tensor([[1, 2], [3, 4]])
B = torch.tensor([[0, 5], [6, 7]])
C = kronecker_product(A, B)
print(C)
```

```
tensor([[ 0.,  5.,  0., 10.],
        [ 6.,  7., 12., 14.],
        [ 0., 15.,  0., 20.],
        [18., 21., 24., 28.]])
```

我们使用了 torch.tensor() 函数来创建矩阵 A 和 B，这两个矩阵分别是 2x2 和 2x2 的矩阵。运行 kronecker_product(A, B) 函数将返回一个 4x4 的矩阵，代表了A和B的Kronecker乘积的结果。

```
import torch

# 定义两个张量
x = torch.tensor([[1, 2], [3, 4]])
y = torch.tensor([[5, 6], [7, 8]])

# 计算Kronecker乘积
k_product = torch.kron(x, y)

# 输出结果
print(k_product)
# tensor([[ 5,  6, 10, 12],
        [ 7,  8, 14, 16],
        [15, 18, 20, 24],
        [21, 24, 28, 32]])
```

Kronecker乘积操作要求两个矩阵的形状分别为(m, n)和(p, q)，则它们的Kronecker乘积的形状为(m * p, n * q)。

克罗内克积（Khatri-Rao Product）

克罗内克积是两个矩阵的列向量的哈达玛积的列向量的竖直拼接。克罗内克积的结果是一个形状为$(mn, r)$的矩阵，其中$m$和$n$分别是两个矩阵的列数，$r$是两个矩阵的行数。在NumPy中，可以使用numpy.khatri_rao函数实现克罗内克积运算。

海森伯格积（Hadamard Kronecker Product）

海森伯格积是两个矩阵的哈达玛积和Kronecker乘积的组合。海森伯格积的结果是一个形状为$(ma, nb)$的矩阵，其中$m$和$n$分别是两个矩阵的行数，$a$和$b$分别是两个矩阵的列数。在NumPy中，可以使用numpy.kron函数和numpy.multiply函数实现海森伯格积运算。

https://blog.csdn.net/Forrest97/article/details/109250728

矩阵的秩

矩阵的秩是指矩阵中线性无关的行或列的最大数目。yTorch 中计算矩阵秩的函数torch.linalg.matrix_rank()，它可以根据输入矩阵是否是Hermitian（复共轭对称）的，选择使用奇异值分解或者特征值分解来求解。

```
import torch
# 创建一个 3x4 的随机矩阵
A = torch.randn(3, 4)
# 计算矩阵的秩
r = torch.linalg.matrix_rank(A)
print(r) # 输出 3
```

```
import torch
# 创建两个 2x3 的随机矩阵
A = torch.randn(2, 3)
B = torch.randn(2, 3)
# 计算 A 和 B 的转置的矩阵乘积，得到一个 3x3 的矩阵
C = torch.matmul(A, B.t())
# 计算 C 的秩
r = torch.linalg.matrix_rank(C)
print(r) # 输出 2 或者 3，取决于 C 是否有线性相关的列
```

矩阵的秩反映了矩阵的线性相关性，也可以用来判断一个线性方程组是否有唯一解，如果一个 3x3 的矩阵的秩是 3，那么它就是满秩的，也就是说它的每一行（或每一列）都是线性无关的，那么它就可以作为系数矩阵来求解一个有三个未知数的方程组，并且有唯一解。

矩阵的秩在深度神经网络中的意义是反映了网络权重矩阵的信息量和复杂度。

矩阵的秩在数据分析中的意义是反映了数据集的维度和冗余性。一些数据分析方法，如主成分分析（PCA），就是利用降低数据集矩阵的秩来提取数据中最重要的特征，从而实现数据降维和可视化。

秩

LOW-RANK

"LOW-RANK"（低秩）通常指的是矩阵的秩（Rank）比较小，低秩矩阵在神经网络中可能反映了信息的紧凑性和结构性，可以被较少的特征向量或奇异值所表示。在很多应用中，低秩矩阵可以用来表示数据中的共性信息和规律，从而实现数据的压缩、降维和分析等目的。

满秩是指矩阵的秩等于它的行数或列数中较小的一个，也就是说矩阵的所有行（或列）都是线性无关的。 低秩是指矩阵的秩小于它的行数或列数中较小的一个，也就是说矩阵有一些线性相关的行（或列）。

rank-deficiency

在线性代数中，"rank-deficiency"（秩亏）指的是一个矩阵的秩（Rank）比它的行数或列数小，即矩阵中存在线性相关的行或列，从而导致矩阵的行列式为零，不能被逆运算。通俗地说，秩亏矩阵的行或列中存在冗余信息，可以由其他行或列表示出来，因此该矩阵包含的信息量比较少，不能完全描述数据的特征。

秩亏矩阵在实际应用中很常见，例如在图像处理和机器学习中，数据矩阵中可能存在噪声、缺失值或冗余特征，导致矩阵秩降低。在这种情况下，我们需要采用一些方法来处理秩亏矩阵，以提高数据的表现力和预测准确率。

常用的秩亏矩阵处理方法包括：主成分分析（PCA）、奇异值分解（SVD）、核矩阵分解（Kernel Matrix Factorization）、低秩矩阵补全（Low-Rank Matrix Completion）等。这些方法可以用来降维、去除噪声、填补缺失值或提取数据的共性信息等，从而提高数据的表现力和预测准确率。

full rank

在线性代数中，"full rank"（满秩）指的是一个矩阵的秩（Rank）等于它的行数或列数中的较小值，即矩阵中的所有行或列都是线性独立的，可以唯一地表示出矩阵的所有元素。满秩矩阵通常具有很好的性质和应用，如可逆、有唯一解、唯一表示等。

满秩矩阵在实际应用中也很常见，例如在线性回归、最小二乘法、矩阵分解等问题中，通常需要矩阵满秩的条件来保证算法的有效性和稳定性。
如果一个矩阵不是满秩矩阵，那么它就是秩亏矩阵（Rank-deficient Matrix），其中至少存在一个线性相关的行或列。秩亏矩阵的性质和应用通常会受到一定的限制和影响，因此在处理实际问题时需要注意矩阵的秩和满秩性质。

Intrinsic dimension

Intrinsic dimension（本征维度）是指数据集中所包含的最少信息所需要的维数，或者说是数据集中实际独立变化的方向的数量。它是指数据集固有的本质维数，不受数据在高维空间中的嵌入方式或表示方式的影响。具体来说，如果一个数据集可以被嵌入到一个 $d$ 维空间中，而且在这个空间中，数据集的每个样本点只需要 $k$ 个坐标来描述，那么数据集的内在维数就是 $k$。

本征维度对于数据分析和机器学习来说是非常重要的，因为它可以帮助我们更好地理解数据集的本质结构，并且可以减少计算和存储的开销。例如，当处理高维数据时，内在维度可以帮助我们找到数据中真正有用的特征，从而提高模型的性能和准确度。另外，内在维度还可以用于数据压缩、可视化和降维等领域。

矩阵的秩和本征维度有一定的关系，但也有区别

矩阵的秩是指矩阵的列空间（或行空间）的维度，也就是说矩阵可以表示多少个线性无关的向量。 本征维度是指数据集或信号的最小表示所需要的变量个数，也就是说数据集或信号可以用多少个线性无关的特征来描述。

如果一个矩阵是由一个数据集或信号构成的，那么它的秩就等于它的本征维度。如果一个矩阵不是由一个数据集或信号构成的，那么它的秩就不一定等于它的本征维度。 例如，一个单位矩阵（对角线上都是 1 的方阵）的秩就等于它的行数（或列数），但它没有本征维度这个概念。

intrinsic rank

Intrinsic rank（内在秩）是指一个数据集中最重要的线性无关的特征的数量，它可以被看作是数据集的内在维度的一个度量。内在秩的概念通常用于描述低秩矩阵近似问题中，其中我们希望通过一个低秩矩阵来近似一个高秩矩阵，以减少存储和计算的开销。

具体来说，如果一个数据集可以表示为一个秩为 $k$ 的矩阵的线性组合，而且这个 $k$ 是数据集中最重要的线性无关特征的数量，那么这个数据集的内在秩就是 $k$。换句话说，内在秩是指能够用来描述数据集最重要信息的最少的线性无关特征数量。

内在秩和内在维度的概念有些类似，但内在秩更多地关注于矩阵的结构和线性无关特征的数量，而内在维度则更多地关注于数据集的实际维度和独立变化方向的数量。在某些情况下，内在秩和内在维度可能是相同的，但在其他情况下，它们可能会有所不同。