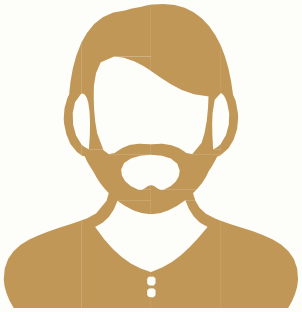
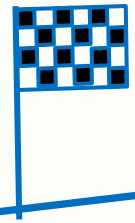
第3章 矩阵

第05讲 矩阵操作

传媒与信息工程学院
欧 新 宇

- 矩阵的定义及基本操作
- 基于矩阵的向量
- 特殊形态的矩阵
- 矩阵的四则运算
- 矩阵的秩和矩阵的迹
- 矩阵的分块
- 张量的常用操作
- 矩阵的应用





第05讲 矩阵操作

矩阵的各种操作

矩阵的基本操作

- 转置
- Frobenius 范数

矩阵的四则运算

- 矩阵的加法（标量、向量、矩阵）
- 矩阵的数乘
- 矩阵与向量的乘法
- 矩阵与矩阵的乘法

特殊运算

- 特征分解
- 奇异值分解
- Moore-Penrose伪逆
- 矩阵的迹运算
- 矩阵的秩
- 矩阵的行列式

矩阵的四则运算

第05讲 矩阵操作

矩阵的四则运算

矩阵的四则运算

矩阵的加法—矩阵与标量相加

【定义1】设存在一个$m \times n$的矩阵$A=a_{ij}$和标量b，那么矩阵A与标量b的和，记作 $A+b$，并规定其和为标量与矩阵的所有元素相加，即：$[A+b]_{ij}=a_{ij}+b$。具体而言，可公式化为：

$$A+b=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}+b$$

$$=\begin{bmatrix} a_{11}+b & a_{12}+b & \cdots & a_{1n}+b \\ a_{21}+b & a_{22}+b & \cdots & a_{2n}+b \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b & a_{m2}+b & \cdots & a_{mn}+b \end{bmatrix}$$

矩阵的四则运算

矩阵的加法—矩阵与标量相加

【例1】设$A=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$，且$b$=10，试求：$A+b$。

解：$A+b=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}+10=\begin{bmatrix}1+10 & 2+10\\ 3+10 & 4+10\end{bmatrix}=\begin{bmatrix}11 & 12\\ 13 & 14\end{bmatrix}$

```
import numpy as np
A = np.array([[1,2],[3,4]])
b = 10

print(A+b)
```

```
Last executed at 2020-05-18 18:29:17 in 4ms
```

```
[[11 12]
 [13 14]]
```

矩阵的四则运算

矩阵的加法—矩阵与向量相加

【定义2】通常，矩阵和向量是无法直接相加的。但是在深度学习中，我们也使用一些不那么准确的表达，允许矩阵与向量直接相加，得到另一个相同大小的矩阵。具体计算过程就是使向量与矩阵中的所有列(行)分别进行相加。

设$A=(a_{ij})$是$m \times n$的矩阵，

- $\boldsymbol{b}=(b_i)$ 是m维行向量，则$\boldsymbol{A}+b$的第j列的元素 $[A+b]_{:,j}=a_{:,j}+b_i$
- $\boldsymbol{b}=(b_j)$ 是n维列向量，则$\boldsymbol{A}+b$的第i行的元素 $[A+b]_{i,:}=a_{i,:}+b_j$

这种隐式复制向量到很多位置参与计算的方式，称之为**广播**（broadcasting）。广播可以提升编码的效率和运行效率，在很多算法库中得到了广泛的支持，例如常用的numpy计算库。

矩阵的四则运算

矩阵的加法——矩阵与向量相加

【例2】设$A=\begin{bmatrix}2 & 1\\ 2 & 4\end{bmatrix}$，且$b$=[10, 20]^T，$c$=[10, 20]。试求：$A+b, A+c$。

解：

$$A+b=\begin{bmatrix}2 & 1\\ 2 & 4\end{bmatrix}+\begin{bmatrix}10\\ 20\end{bmatrix}=\begin{bmatrix}2+10 & 1+10\\ 2+20 & 4+20\end{bmatrix}=\begin{bmatrix}12 & 11\\ 22 & 24\end{bmatrix}$$

$$A+c=\begin{bmatrix}2 & 1\\ 2 & 4\end{bmatrix}+[10,20]=\begin{bmatrix}2+10 & 1+20\\ 2+10 & 4+20\end{bmatrix}=\begin{bmatrix}12 & 21\\ 12 & 24\end{bmatrix}$$

```
import numpy as np
A = np.array([[2,1],[2,4]])
b = np.array([[10],[20]])
c = np.array([[10,20]])

print('A+b=\n{}'.format(A+b))
print('A+c=\n{}'.format(A+c))
```

Last executed at 2020-05-19 19:49:49 in 5ms

```
A+b=
[[12 11]
 [22 24]]
A+c=
[[12 21]
 [12 24]]
```

值得注意的是这种运算方法，只用于AI领域，并不符合数学规范。

矩阵的四则运算

矩阵的加法——矩阵与矩阵相加

【定义3】设存在两个$m \times n$的矩阵 $\boldsymbol{A}=a_{ij},\boldsymbol{B}=b_{ij}$，那么矩阵$\boldsymbol{A}$与$\boldsymbol{B}$的和 记作$\boldsymbol{A}+\boldsymbol{B}$，规定为：

$$\boldsymbol{A}+\boldsymbol{B}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}+\begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}$$

$$=\begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{21} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

只有当两个矩阵是同型矩阵时，这两个矩阵才能进行加法运算。

矩阵的四则运算

矩阵加法的运算规律—矩阵与矩阵相加

矩阵加法满足下列运算规律（设A, B, C都是$m \times n$的矩阵）：

1. $A + B = B + A$
2. $(A + B) + C = A + (B + C)$

设矩阵 $A=a_{ij}$，记 $-A=-a_{ij}$，$-A$ 称为矩阵A的负矩阵，显然有：

$$A + (-A) = O$$

由此规定矩阵的减法为：$A - B = A + (-B)$

矩阵的四则运算

矩阵的加法 (Python实现) —矩阵与矩阵相加

```
import numpy as np
A = np.array([[1,2],[3,4],[5,6]])
B = np.array([[10,20],[30,40],[50,60]])
print("A=\n{},\n B=\n{},\n A+B=\n{}, \n B-A=\n{}".format(A,B,A+B,B-A))
```

```
A=
[[1 2]
 [3 4]
 [5 6]],
 B=
[[10 20]
 [30 40]
 [50 60]],
```

```
 A+B=
[[11 22]
 [33 44]
 [55 66]],
```

```
 B-A=
[[ 9 18]
 [27 36]
 [45 54]]
```

矩阵的四则运算

【例3】超市年度销售总额

假设有四个超市，它们上半年和下半年的销售清单如下所示，试求这四个超市全年的销售清单。

上半年销售表	大米	面粉	食油
超市一	150	250	50
超市二	250	500	100
超市三	300	700	120
超市四	450	850	80

下半年销售表	大米	面粉	食油
超市一	180	350	60
超市二	300	550	120
超市三	350	850	150
超市四	500	850	100

将表单写成矩阵形式：

$$A=\begin{bmatrix}150 & 250 & 50\\ 250 & 500 & 100\\ 300 & 700 & 120\\ 450 & 850 & 80\end{bmatrix},\quad B=\begin{bmatrix}180 & 350 & 60\\ 300 & 550 & 120\\ 350 & 850 & 150\\ 500 & 850 & 100\end{bmatrix}$$

则全年的销售情况可以用矩阵的加法来表示：

$$C=A+B=\begin{bmatrix}150+180 & 250+350 & 50+60\\ 250+300 & 500+550 & 100+120\\ 300+350 & 700+850 & 120+150\\ 450+500 & 850+850 & 80+100\end{bmatrix}$$

矩阵的四则运算

【例3】超市年度销售总额

下面给出Python代码的实现方法：

```
import numpy as np

A = np.array([[150,250,50],[250,500,100],[300,700,120],[450,850,80]])
B = np.array([[180,350,60],[300,550,120],[350,850,150],[500,850,100]])
C = A + B
print("C=\n{}".format(C))
```

```
C=
[[ 330  600  110]
 [ 550 1050  220]
 [ 650 1550  270]
 [ 950 1700  180]]
```

基于numpy的矩阵加法也是按位相加的规则，运算过程直观、简单。但需要注意的是numpy加法运算的两个元素也必须具有相同的形态。

课堂互动一 Link

矩阵的四则运算

矩阵的乘法——矩阵与标量相乘（数量乘法）

【定义4】数 λ 与矩阵A的乘积记作 λA 或 $A\lambda$，规定为：

$$\lambda A = A\lambda = \lambda \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} \lambda a_{11} & \lambda a_{12} & \cdots & \lambda a_{1n} \\ \lambda a_{21} & \lambda a_{22} & \cdots & \lambda a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda a_{m1} & \lambda a_{m2} & \cdots & \lambda a_{mn} \end{bmatrix}$$

数乘矩阵满足下列运算规律（设 $\boldsymbol{A},\boldsymbol{B}$ 为$m \times n$矩阵，λ，μ为标量）：

- $(\lambda\mu)\boldsymbol{A} = \lambda(\mu\boldsymbol{A})$
- $(\lambda+\mu)\boldsymbol{A} = \lambda\boldsymbol{A} + \mu\boldsymbol{A}$
- $\lambda(\boldsymbol{A}+\boldsymbol{B}) = \lambda\boldsymbol{A} + \lambda\boldsymbol{B}$

矩阵相加与数乘矩阵合起来，统称为矩阵的**线性运算**。

矩阵的四则运算

【例4】计算期末总成绩

甲、乙、丙三位同学在期末考试中，4门课程的成绩分别由矩阵A给出，而他们的平时成绩则由矩阵B给出，若期末考试成绩占总成绩的90%，而平时成绩占10%，请计算这三名同学的总成绩。

$$A=\begin{bmatrix}85 & 85 & 65 & 98\\ 75 & 95 & 70 & 95\\ 80 & 70 & 76 & 92\end{bmatrix},\quad B=\begin{bmatrix}90 & 70 & 80 & 92\\ 80 & 90 & 82 & 92\\ 85 & 75 & 90 & 90\end{bmatrix}$$

解：矩阵C表示总成绩，显然有：

$$C=0.9A+0.1B=\begin{bmatrix}85.5 & 83.5 & 66.5 & 97.4\\ 75.5 & 94.5 & 71.2 & 94.7\\ 80.5 & 70.5 & 77.4 & 91.8\end{bmatrix}$$

矩阵的四则运算

【例2】计算期末总成绩

甲、乙、丙三位同学在期末考试中，4门课程的成绩分别由矩阵***A***给出，而他们的平时成绩则由矩阵***B***给出，若期末考试成绩占总成绩的90%，而平时成绩占10%，请计算这三名同学的总成绩。

```
import numpy as np

A = np.array([[85,85,65,98],[75,95,70,95],[80,70,76,92]])
B = np.array([[90,70,80,92],[80,90,82,92],[82,75,90,90]])
C = 0.9*A + 0.1*B
print("C=\n{}".format(C))
```

```
C=
[[85.5 83.5 66.5 97.4]
 [75.5 94.5 71.2 94.7]
 [80.2 70.5 77.4 91.8]]
```

矩阵的四则运算

矩阵的乘法——矩阵与矩阵相乘

【定义5】设$A=(a_{ij})$ 是一个$m\times s$矩阵，$B=(b_{ij})$ 是一个$s\times n$矩阵，那么矩阵A与矩阵B的乘积是一个$m\times n$矩阵$C=(c_{ij})$，其中

$c_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+\ldots+a_{is}b_{sj}=\sum_{k=1}^{s}a_{ik}b_{kj}$ $(i=1,2,\ldots,m;\ j=1,2,\ldots,n)$，并把此乘积记作 $C=AB$。

$$\mathrm{C}=A\times\mathrm{B}=\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\\ a_{31} & a_{32}\end{bmatrix}\begin{bmatrix}b_{11} & b_{12} & b_{13}\\ b_{21} & b_{22} & b_{23}\end{bmatrix}$$

$$=\begin{bmatrix}a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22} & a_{11}b_{13}+a_{12}b_{21}\\ a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{11}+a_{12}b_{21}\\ a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{11}+a_{12}b_{21}\end{bmatrix}$$

矩阵的四则运算

矩阵乘法的要点

通过观察上面的计算公式，可以总结出矩阵乘法 $C=AB$ 的三个要点：

- **AB可乘的条件**：A的列数 = B的行数
- **AB乘积C的形状**：A的行 × B的列
- **AB乘积C的元素构成**：A的行与B的列的内积

向量可以看成是只有一列（行）的矩阵，因此矩阵和向量相乘也可以看成是矩阵与矩阵相乘的一种特例。只要满足矩阵可乘的条件即可。

矩阵的四则运算

矩阵的乘法——矩阵与矩阵相乘

【例4】求矩阵$A=\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}, B=\begin{bmatrix}1&7\\2&8\\3&9\end{bmatrix}$的乘积$AB$。

解：因为A是3×3矩阵，B是3×2矩阵，A的列数等于B的行数，所以矩阵A与B可相乘，其乘积$C=AB$是一个3×2矩阵。按定理可得：

$$C=AB=\begin{bmatrix}1&2&3\\4&5&6\\1&1&1\end{bmatrix}\begin{bmatrix}1&7\\2&8\\3&9\end{bmatrix}=\begin{bmatrix}1*1+2*2+3*3&1*7+2*8+3*9\\4*1+5*2+6*3&4*7+5*8+6*9\\1*1+1*2+1*3&1*7+1*8+1*9\end{bmatrix}$$

$$=\begin{bmatrix}14&50\\32&122\\6&24\end{bmatrix}$$

矩阵的四则运算

【例4】矩阵的乘法—矩阵与矩阵相乘

下面给出该乘积的Python代码。注意我们使用np.dot(A,B)实现矩阵乘法。

```
import numpy as np

A = np.array([[1,2,3],[4,5,6],[1,1,1]])
B = np.array([[1,7],[2,8],[3,9]])
C = np.dot(A,B)
print("C=\n{}".format(C))
```

```
C=
[[ 14  50]
 [ 32 122]
 [  6  24]]
```

值得注意的是矩阵乘法是不符合交换律的，换句话说，上例中的***BA***是没有意义的。

矩阵的四则运算

【例4】矩阵的乘法—矩阵与矩阵相乘

```
import numpy as np

A = np.array([[1,2,3],[4,5,6],[1,1,1]])
B = np.array([[1,7],[2,8],[3,9]])
C = np.dot(B,A)
print("C=\n{}".format(C))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-19-4de9c015f434> in <module>
      3 A = np.array([[1,2,3],[4,5,6],[1,1,1]])
      4 B = np.array([[1,7],[2,8],[3,9]])
----> 5 C = np.dot(B,A)
      6 print("C=\n{}".format(C))

ValueError: shapes (3,2) and (3,3) not aligned: 2 (dim 1) != 3 (dim 0)
```

矩阵的四则运算

【例5】矩阵的乘法—矩阵与矩阵相乘（交换律）

【例5】求矩阵$A=\begin{bmatrix}-2 & 4\\ 1 & -2\end{bmatrix}, B=\begin{bmatrix}2 & 4\\ -3 & -6\end{bmatrix}$的乘积***AB***及***BA***。

解：按定义有

$$AB=\begin{bmatrix}-2 & 4\\ 1 & -2\end{bmatrix}\begin{bmatrix}2 & 4\\ -3 & -6\end{bmatrix}=\begin{bmatrix}-16 & -32\\ 8 & 16\end{bmatrix}$$

$$BA=\begin{bmatrix}2 & 4\\ -3 & -6\end{bmatrix}\begin{bmatrix}-2 & 4\\ 1 & -2\end{bmatrix}=\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$$

```python
import numpy as np

A = np.array([[-2,4],[1,-2]])
B = np.array([[2,4],[-3,-6]])
AB = np.dot(A,B)
BA = np.dot(B,A)
print("AB=\n{}".format(AB))
print("BA=\n{}".format(BA))
```

```
AB=
[[-16 -32]
 [  8  16]]
BA=
[[0 0]
 [0 0]]
```

矩阵的四则运算

【结果分析】矩阵与矩阵相乘（交换律）

根据以上例题，可以引申得到如下一些结论：

- 矩阵乘法不满足交换律，即在一般情况下$AB \neq BA$
- 若A、B满足$AB=O$，不能得出$A=O$或$B=O$的结论；
- 若$A=O$，$B=O$，若AB为同型矩阵（即A、B可乘），则$AB=O$
- 若$A\neq O$，而 $A(X-Y)=O$，不能得出$X=Y$的结论。
- 若$AC=AB$, $A\neq 0$, 不能推出$B=C$

矩阵的四则运算

矩阵与矩阵相乘的运算规律

矩阵乘法虽然不满足交换律，但仍满足下列结合律和分配率（假设运算是可行的）

- **结合律**： $(AB)C=A(BC)$
- $\lambda(AB)=(\lambda A)B=A(\lambda B)$，其中 λ 是标量
- **分配律**： $A(B+C)=AB+AC,(B+C)A=BA+CA$

对于单位矩阵 I，不难得出结论：$I_m A_{m\times n}=A_{m\times n}$，$A_{m\times n}I_n=A_{m\times n}$，或简写成：$IA=AI=A$。

可见单位矩阵 I 在矩阵乘法中的作用类似于标量 1。

矩阵的四则运算

矩阵的除法

默认情况下，矩阵是没有除法运算的，但是可以使用逆矩阵来求解矩阵的除法：

$$
\begin{aligned}
& AB = C \\
\Rightarrow\ & A^{-1}AB = A^{-1}C \\
\Rightarrow\ & B = A^{-1}C
\end{aligned}
$$

$$
\begin{aligned}
& AB = C \\
\Rightarrow\ & ABB^{-1} = CB^{-1} \\
\Rightarrow\ & A = CB^{-1}
\end{aligned}
$$

矩阵的四则运算

【例6】服装厂的生产量

有甲、乙、丙、丁4个服装厂，一个月的产量情况由下表给出，若甲厂生产8个月，乙厂生产10个月，丙厂生产5个月，而丁厂生产9个月，则共生产帽子、衣服、裤子各多少？用矩阵来描述。

品种	甲	乙	丙	丁
帽	20	4	2	7
衣	10	18	5	6
裤	5	7	16	3

解：上表可以矩阵化为：$A=\begin{bmatrix}20 & 4 & 2 & 7\\ 10 & 18 & 5 & 6\\ 5 & 7 & 16 & 3\end{bmatrix}$，$b=\begin{bmatrix}8\\ 10\\ 5\\ 9\end{bmatrix}$

则，总产量可以表示为：$C=A\times b=\begin{bmatrix}8\times 20 & 10\times 4 & 5\times 2 & 9\times 7\\ 8\times 10 & 10\times 18 & 5\times 5 & 9\times 6\\ 8\times 5 & 10\times 7 & 5\times 16 & 9\times 3\end{bmatrix}=\begin{bmatrix}273\\ 339\\ 217\end{bmatrix}$

矩阵的四则运算

【例6】服装厂的生产量

有甲、乙、丙、丁4个服装厂，一个月的产量情况由下表给出，若甲厂生产8个月，乙厂生产10个月，丙厂生产5个月，而丁厂生产9个月，则共生产帽子、衣服、裤子各多少？用矩阵来描述。

品种	甲	乙	丙	丁
帽	20	4	2	7
衣	10	18	5	6
裤	5	7	16	3

```
import numpy as np

A = np.array([[20,4,2,7],[10,18,5,6],[5,7,16,3]])
b = np.array([[8],[10],[5],[9]])
C = np.dot(A, b)
print("C=\n{}".format(C))
```

```
C=
[[273]
 [339]
 [217]]
```

矩阵的四则运算

矩阵的幂

根据矩阵的乘法法则，定义矩阵的**幂**。设A是n阶方阵，定义:

$$A^1=A,\ A^2=A^1A^1,\ \ldots,\ A^{k+1}=A^kA^1$$

其中 k 为正整数。也就是说，A^k 就是 k 个 A 连乘。显然只有**方阵**满足条件，**幂**运算才有意义。

由于矩阵乘法适合结合律，所以矩阵的幂满足以下运算规律:

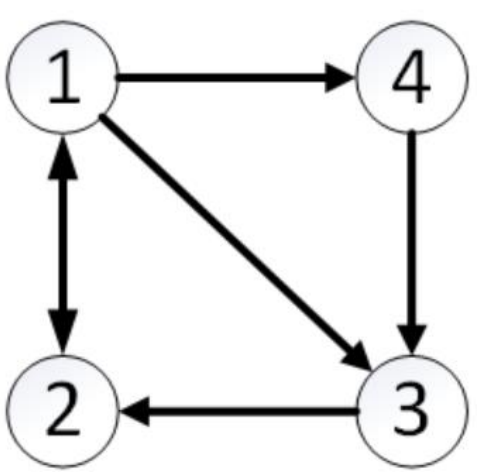
$$A^kA^l=A^{k+l},\ (A^k)^l=A^{kl}$$

其中 k、l 为正整数。又因矩阵乘法一般不满足交换律，所以对于两个 n 阶矩阵 A 与 B，一般来说 $(AB)^k\neq A^kB^k$。

矩阵的四则运算

【例7】四个城市间的单向航线

若四个城市的航向图如下所示：



$$A=(a_{ij})=\begin{bmatrix}0&1&1&1\\1&0&0&0\\0&1&0&0\\1&0&1&0\end{bmatrix},$$

$$a_{ij}=\begin{cases}1 & \text{从}i\text{市到}j\text{市有1条单向航线}\\ 0 & \text{从}i\text{市到}j\text{市没有单向航线}\end{cases}$$

若记$A^2=b_{ij}$，则b_{ij}为从 i 市经过一次中转到 j 市的单向航线条数。

$$A=(a_{ij})=\begin{bmatrix}0&1&1&1\\1&0&0&0\\0&1&0&0\\1&0&1&0\end{bmatrix},\ \text{有}A^2=\begin{bmatrix}2&1&1&0\\0&1&1&1\\1&0&0&0\\0&2&1&1\end{bmatrix}.$$

矩阵的四则运算

【例7】四个城市间的单向航线

$$A=(a_{ij})=\begin{bmatrix}0&1&1&1\\1&0&0&0\\0&1&0&0\\1&0&1&0\end{bmatrix}，有A^2=\begin{bmatrix}2&1&1&0\\0&1&1&1\\1&0&0&0\\0&2&1&1\end{bmatrix}.$$

- ✓ b_{23}=1，显示从(2)经过一次中转后到达(3)市的单向航线有一条：(2)->(1)->(3);
- ✓ b_{42}=2，显示从(4)经过一次中转后到达(2)市的单向航线有2条：(4)->(1)->(2)，(4)->(3)->(2);
- ✓ b_{11}=2，显示过(1)市的双向航线有2条：(1)->(2)->(1)，(1)->(4)->(1);
- ✓ b_{33}=0，显示(3)市没有双向航线。

矩阵的四则运算

【例7】四个城市间的单向航线(Python实现)

假设给定上面的航线矩阵A，试求从(4)市到(2)市，最多经过4次转机，有几种到达方法。

解：基本的思路是直飞(A)，一次转机（A^2），二次转机（A^3），三次转机（A^4），四次转机（A^5）之和。使用矩阵乘法显然很复杂，下面我们使用Python代码来完成这个例子。

```
import numpy as np
A = np.array([[0,1,1,1],[1,0,0,0],[0,1,0,0],[1,0,1,0]])
A2 = np.dot(A,A)
A3 = np.dot(A2,A)
A4 = np.dot(A3,A)
A5 = np.dot(A4,A)
print(A[3,1] + A2[3,1] + A3[3,1] + A4[3,1] + A5[3,1])
```

```
11
```

需要注意的是：矩阵乘法的代码语法，不能使用 A*A(元素乘)，而要使用点乘 .dot(A,B)。

矩阵的四则运算

矩阵的乘法—哈达玛积

哈达玛积(Hadamard product)：矩阵的一种乘法运算，它计算两个矩阵对应元素的乘积。因此，又称为元素对应乘积(element-wise)，元素积或Hadamard乘积。

【定义6】设$\boldsymbol{A},\boldsymbol{B}\in\boldsymbol{C}^{m\times n}$是两个$m\times n$的同型矩阵，且$\boldsymbol{A}=(a_{ij})$，$\boldsymbol{B}=(b_{ij})$，有：

$$\boldsymbol{C}=\begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1n}b_{1n} \\ a_{11}b_{11} & a_{11}b_{11} & \cdots & a_{2nb2n} \\ \vdots & \vdots & & \vdots \\ a_{m1}b_{m1} & a_{m1}b_{m1} & \cdots & a_{m1}b_{m1} \end{bmatrix}$$

为矩阵A和B的Hadamard乘积，记作：$\boldsymbol{A}\odot\boldsymbol{B}$。

Hadamard乘积在图像处理中被广泛应用，例如：为一幅图片增加光照，相当于对图像矩阵中的每个像素进行变换。。

矩阵的四则运算

矩阵的乘法—哈达玛积

哈达玛积的性质：

- $A \odot 0 = 0 \odot A = 0$
- $A \odot B = B \odot A$
- $(A + B) \odot C = A \odot C + B \odot C$

课堂互动二 Link

矩阵的秩和矩阵的迹

矩阵的秩和矩阵的迹

矩阵的秩

矩阵的秩等于矩阵行阶梯型中非零行的个数，具体而言：

- 矩阵A的列秩，是指A的线性独立的纵列的极大数；
- 矩阵A的行秩，是指A的线性无关的行的极大数。

方阵的列秩和行秩总是相等的，因此它们可以称作**矩阵A的秩**，表示为：$r(A)$，$rk(A)$或rank A。对于矩阵$A_{m\times n}$，它的秩为min(m,n)，其中m, n均线性无关。

求**矩阵的秩**的方法，需要将矩阵进行**初等变换**。基于Python，我们可以使用numpy库来实现.

```
>> np.linalg.matrix_rank(A)
```

矩阵的秩和矩阵的迹

矩阵的迹

【定义】存在方阵$\boldsymbol{A}=(\boldsymbol{a_{ij}})$，它主对角线上所有元素的和称为**矩阵$\boldsymbol{A}$的迹**，记作：$\mathrm{tr}(\boldsymbol{A})=\sum_{i=1}^{n}a_{ii}$。

矩阵的迹在基于线性代数的机器学习中具有非常重要作用，特别是在很多矩阵运算中，若不使用求和符号很难进行描述，而通过矩阵乘法和迹运算符号可以很清楚地表示。

求**矩阵的迹**的方法，也可以使用numpy库来实现.

```
>> np.linalg.matrix_rank(A)
```

矩阵的秩和矩阵的迹

矩阵的迹的应用

1. 一种描述方阵的Frobenius范数的方式

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j} a_{i,j}^2} = \sqrt{\sum_{i,i} a_{i,i}^2} = \sqrt{Tr(\boldsymbol{A}\boldsymbol{A}^T)}$$

2. 迹运算在转置下不变

$$Tr(\boldsymbol{A}) = Tr(\boldsymbol{A}^T)$$

3. 标量的迹等于它本身

$$Tr(a) = a$$

4. 线性变换的迹保持不变

$$Tr(mA + nB) = mTr(A) + nTr(B)$$

矩阵的秩和矩阵的迹

矩阵的迹

5. 多个矩阵相乘得到的方阵的迹，等价于将这些矩阵中的最后一个挪到最前面之后相乘的迹 (前提是矩阵可乘)。

$$\mathrm{Tr}(\boldsymbol{ABC}) = \mathrm{Tr}(\boldsymbol{CAB}) = \mathrm{Tr}(\boldsymbol{BCA})$$

6. 更一般，即使连乘之后矩阵形态发生变化，其迹运算结果不变

$$\mathrm{Tr}\left(\prod_{i=1}^{n} \boldsymbol{F}^{(i)}\right) = \mathrm{Tr}\left(\boldsymbol{F}^{(n)} \prod_{i=1}^{n-1} \boldsymbol{F}^{(i)}\right)$$

例如：假设矩阵$\boldsymbol{A} \in R^{m\times n}$，矩阵$\boldsymbol{B} \in R^{n\times m}$，我们可以得到$\boldsymbol{AB} \in R^{m\times m}$和$\boldsymbol{BA} \in R^{n\times n}$，仍然有：

$$\mathrm{Tr}(\boldsymbol{AB}) = \mathrm{Tr}(\boldsymbol{BA})$$

课堂互动三 Link

矩阵的分块

矩阵的分块

- 将矩阵***A***分为若干个小矩阵，每一个小矩阵称为***A***的子块

$$\mathbf{A}=\left[\begin{array}{c:cc} 1 & 7 & 0 \\ 2 & 3 & 9 \\ \hdashline 3 & 8 & 1 \\ 4 & -1 & 6 \end{array}\right] \quad \left[\begin{array}{c:c:c} 1 & 7 & 0 \\ 2 & 3 & 9 \\ 3 & 8 & 1 \\ 4 & -1 & 6 \end{array}\right] \quad \left[\begin{array}{cc:c} 1 & 7 & 0 \\ \hdashline 2 & 3 & 9 \\ 3 & 8 & 1 \\ \hdashline 4 & -1 & 6 \end{array}\right] \quad \left[\begin{array}{c:cc} 1 & 7 & 0 \\ 2 & 3 & 9 \\ \hdashline 3 & 8 & 1 \\ 4 & -1 & 6 \end{array}\right]$$

- 最有用的分块方法是按行和按列分块。如下所示，

$$A=\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}=\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{bmatrix}=[\beta_1 \quad \cdots \quad \beta_n]$$

其中$\alpha_1=[a_{11} \quad a_{12} \quad \cdots \quad a_{1n}]$为行向量，$\beta_1=\begin{bmatrix} \alpha_{11} \\ \vdots \\ \alpha_{m1} \end{bmatrix}$为列向量。

矩阵***A***既可看作行向量***α***的组合，也可以看作列向量***β***的组合。

矩阵的分块

分块矩阵的运算规则

矩阵分块以后，其加减、数乘、乘法、转置等四则运算规则仍然适用。所以分块矩阵相加时，两个矩阵及其子矩阵必须保持同型；相乘时，左乘矩阵及其子矩阵的列数必须等于右乘矩阵及其子矩阵的行数。

$$\boldsymbol{A}=\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \cdots & \boldsymbol{A}_{1t} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} & \cdots & \boldsymbol{A}_{2t} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}_{r1} & \boldsymbol{A}_{r2} & \cdots & \boldsymbol{A}_{rt} \end{bmatrix} \quad \boldsymbol{B}=\begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} & \cdots & \boldsymbol{B}_{1s} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} & \cdots & \boldsymbol{B}_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{B}_{t1} & \boldsymbol{B}_{t2} & \cdots & \boldsymbol{B}_{ts} \end{bmatrix} \quad \boldsymbol{AB}=\begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} & \cdots & \boldsymbol{C}_{1s} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} & \cdots & \boldsymbol{C}_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{C}_{r1} & \boldsymbol{C}_{r2} & \cdots & \boldsymbol{C}_{rs} \end{bmatrix}$$

其中，$C_{ij}=A_{i1}B_{1j}+A_{i2}B_{2j}+\cdots+A_{it}B_{tj}=\sum_{k=1}^{t}A_{ik}B_{kj}$

矩阵的分块

分块矩阵的例子

【例8】利用分块矩阵的概念，把下列线性方程组写成向量等式。

$$\begin{cases} 2x_1 - 2x_2 + 6x_4 = -2 \\ 2x_1 - x_2 + 2x_3 + 4x_4 = -2 \\ 3x_1 - x_2 + 4x_3 + 4x_4 = -3 \end{cases}$$

解：线性方程组的矩阵可看做四个列矩阵（列向量）乘以四个行元素：

$$\begin{bmatrix} 2 & -2 & 0 & 6 \\ 2 & -1 & 2 & 4 \\ 3 & -1 & 4 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \\ -3 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix} x_1 + \begin{bmatrix} -2 \\ -1 \\ -1 \end{bmatrix} x_2, + \begin{bmatrix} 0 \\ 2 \\ 4 \end{bmatrix} x_3 + \begin{bmatrix} 6 \\ 4 \\ 4 \end{bmatrix} x_4 = \begin{bmatrix} -2 \\ -2 \\ -3 \end{bmatrix}$$

$$x_1\boldsymbol{\alpha_1} + x_2\boldsymbol{\alpha_2} + x_3\boldsymbol{\alpha_3} + x_4\boldsymbol{\alpha_4} = \mathbf{b}$$

化成了向量等式。

张量的常用操作

张量的常用操作

张量与人工智能应用

在机器学习和深度学习中，我们常常将需要处理的数据规范化为特定维度的张量。例如，在不进行批处理的时候，

- **彩色图像**可以看成是一个三维张量，分别是图像的高，图像的宽，图像的颜色通道（RGB），即，[C,H,W]。
- **视频**可以看成是一个四维张量，分别是视频的时间帧方向、每一帧的颜色通道、高和宽，即，[T,C,H,W]
- **三维场景**也可以看成是四维张量，分别是每一个像素的信息编码轴、场景的高、宽和深度，即：[C,H,W,D]

事实上，从前面的学习中我们已经知道包括标量、向量、矩阵在内的所有数据类型都可以统一成张量，这为我们进行数据处理提供了方便。在Python中，我们可以使用numpy数组来表示张量。

张量的常用操作

张量的操作

值得注意的是，数据的维度并非一成不变。以图象为例，如果输入的是灰度图，那么只需要二维张量即可，而如果是彩色图那么还需要一个单独的颜色通道。不过，对于张量的操作我们其实可以归结出一些常用的操作，例如：索引(indexing)、切片(slicing)、链接(joining)、换位(mutating)等。其中，张量的索引、切片和链接和矩阵操作基本一致。下面，我们简单介绍一下**换位**操作。

换位（mutating）：张量的换位操作类似于矩阵的转置操作。矩阵的转置是交换行和列，使得索引的次序改变。对于张量来讲，这种索引的改变，将涉及多个维度的交换。在此期间，换位操作要求只能调整索引的顺序，而每个维度内部的元素不能改变。换位操作可公式化为(仅一种)：

$$\boldsymbol{A}(c,\ i,\ j,\ k)\ =\ \boldsymbol{B}(i,\ j,\ k,\ c)$$

张量的常用操作

张量的操作—换位应用

例如，处理连续视频序列时，

- 输入视频张量$A(10,\ 3,\ 800,\ 600)$，分别表示10帧，每帧图像是3通道彩色图像，图像大小为800×600像素。
- 经过卷积神经网络提取特征后，张量A的大小变为$B\ (10,\ 128,\ 64,\ 64)$。
- 接下来需要使用循环神经网络对时间轴进行编码，我们需要先将张量的各个维度进行换位，得到张量$C(128,\ 64,\ 64,\ 10)$，再进行处理。

张量的常用操作

张量的操作

首先，我们创建一个4维的随机数组成的张量用于模拟一组图像数据，其格式为[num, height, weight, channel]。

```
import numpy as np

bs = list()
for i in range(0, 10):
    a = np.random.randint(low=0, high=255, size=[100, 200, 3])
    bs.append(a[np.newaxis, :])
imgs = np.concatenate(bs, axis=0)
print(imgs.shape)
```

```
Last executed at 2020-06-03 11:19:25 in 9ms
```

```
(10, 100, 200, 3)
```

张量的常用操作

张量的操作

- 调整张量各个维度的顺序

```
A = imgs.transpose(3,0,1,2)
print(A.shape)
```

```
Last executed at 2020-06-03 11:45:40 in 5ms
```

```
(3, 10, 100, 200)
```

- 利用张量的索引进行切片

```
a = imgs[2,:,:,:]
b = A[:,2,:,:].transpose(1,2,0)

print(b.shape)
print(a.shape)
```

```
Last executed at 2020-06-03 11:41:25 in 5ms
```

```
(100, 200, 3)
(100, 200, 3)
```

张量的常用操作

张量的操作

- 张量的链接（将两个样本合并成到一个张量）

```
New = np.stack((a,b))
print(New.shape)
```

```
Last executed at 2020-06-03 11:39:03 in 10m
(2, 100, 200, 3)
```

- 将样本拉成一行n列

```
img = a.reshape(1,-1)
print(img.shape)
```

```
Last executed at 2020-06-03 11:39:46 in 5ms
(1, 60000)
```

张量的常用操作

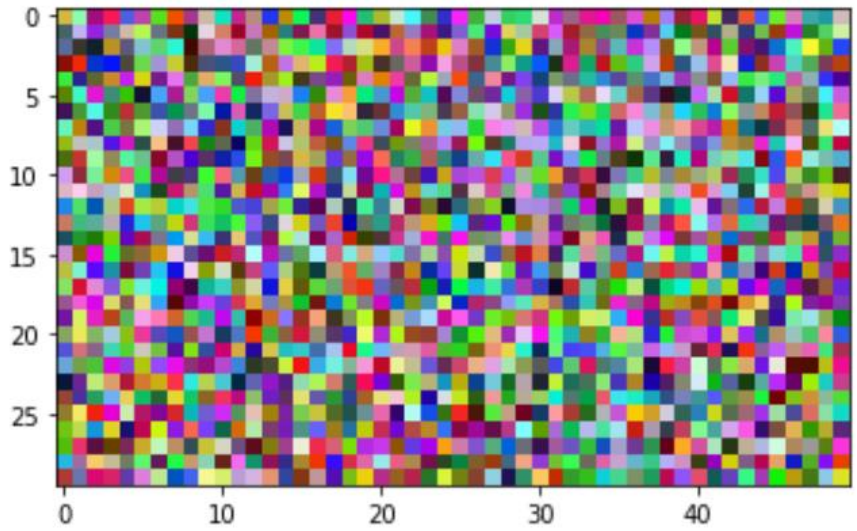
张量的操作

- 显示样本

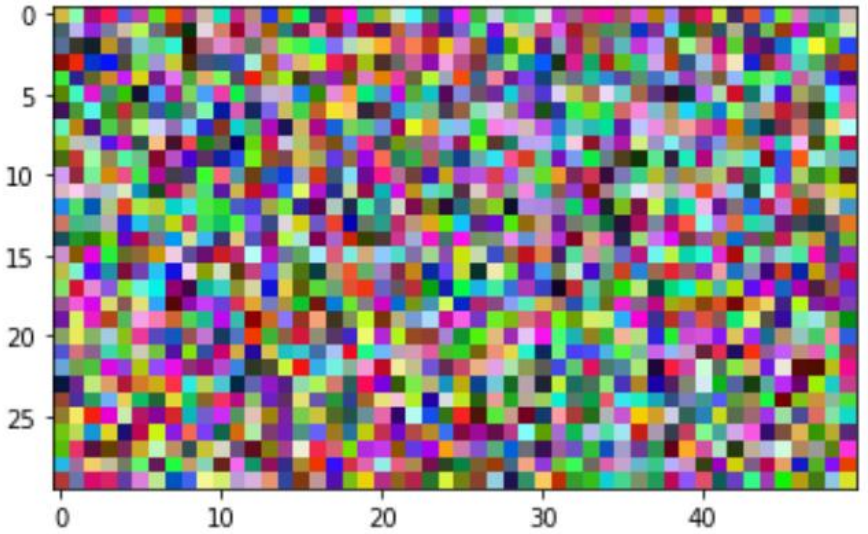
```
import matplotlib.pyplot as plt
plt.imshow(a[20:50,30:80,:])
plt.imshow(b[20:50,30:80,:])
```

Last executed at 2020-06-03 11:43:30 in 134ms

<matplotlib.image.AxesImage at 0x1d8bc801a88>



<matplotlib.image.AxesImage at 0x1d8bcc6f508>



课堂互动四 Link

读万卷书 行万里路 只为最好的修炼

QQ：14777591 (宇宙骑士)

Email：ouxinyu@alumni.hust.edu.cn

Tel：18687840023