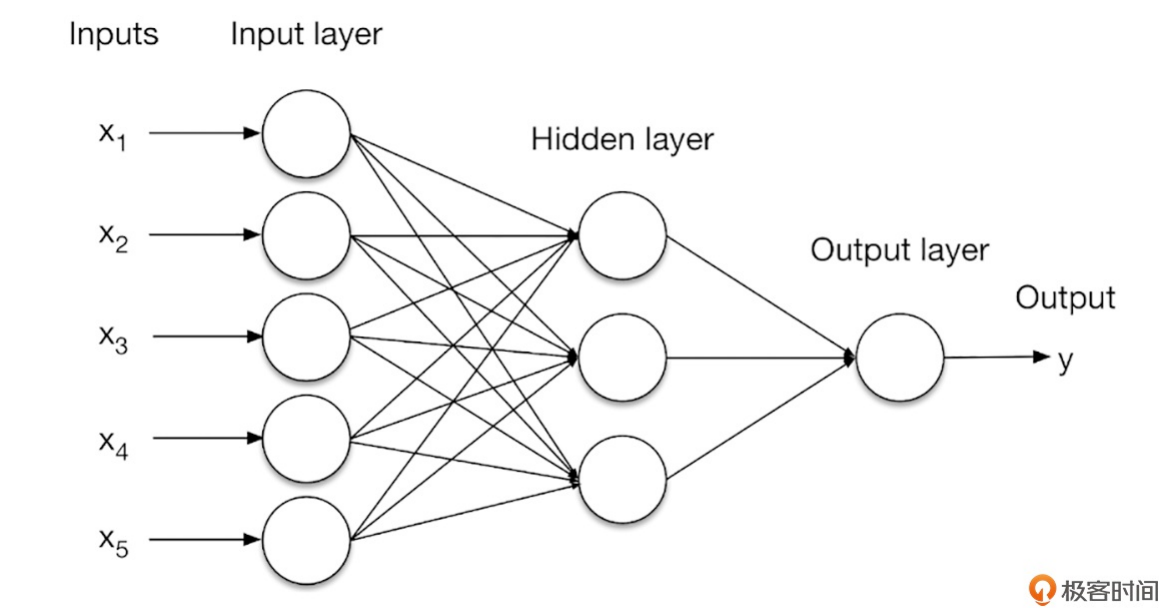


你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，**矩阵可以极大地提高计算机的运算效率**。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。



上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做**向量化（Vectorization）**，这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```
SS
X=\left[\begin{array}{l} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array}\right]
SS

W=\left[\begin{array}{ll} w_1 & w_2 \\ w_4 & w_5 \\ x_3 & w_6 \end{array}\right]
SS

H=f\left(\left[\begin{array}{l} w_1 & w_2 \\ w_4 & w_5 \\ x_3 & w_6 \end{array}\right] \cdot \left[\begin{array}{l} x_1 \\ x_2 \end{array}\right] + b\right)
SS
```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 **Arthur Cayley** 被公认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```
SSax+by=cSS

SS
\left[\begin{array}{l} a_1 x+b_1 y+c_1=0 \\ a_2 x+b_2 y+c_2=0 \end{array}\right]
SS

SS
\left[\begin{array}{l} a_{11} x_1+a_{12} x_2+\cdots+a_{1 n} x_n=b_1 \\ a_{21} x_1+a_{22} x_2+\cdots+a_{2 n} x_n=b_2 \\ \cdots \cdots \cdots \cdots \cdots \cdots \cdots \cdots \cdots \cdots \cdots \\ a_{m 1} x_1+a_{m 2} x_2+\cdots+a_{m n} x_n=b_m \end{array}\right]
SS
```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```
SS
A=\left[\begin{array}{cccc} a_{11} & a_{12} & \cdots & a_{1 n} \\ a_{21} & a_{22} & \cdots & a_{2 n} \end{array}\right]
SS
```

```
\dots & \dots & \dots & \dots \\\
a_{m1} & a_{m2} & \dots & a_{mn} \\
\end{array}\right]
\end{array}
\end{array}
```

我们把 $SAS$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $SbS$ 放入矩阵后，就是下面这样：

```
\widetilde{A}=\left[\begin{array}{cccc}
a_{11} & a_{12} & \dots & a_{1n} \\
a_{21} & a_{22} & \dots & a_{2n} \\
\dots & \dots & \dots & \dots \\
a_{m1} & a_{m2} & \dots & a_{mn}
\end{array}\right]
\end{array}
\end{array}
```

这样我们就得到了 $SAS$ 矩阵的增广矩阵 $S\widetilde{A}S$ ，可以表示为 $S(A, B)S$ ，这里的 $SBS$ 表示的是方程组常数项所构成的列向量，也就是 $m\times 1$ 的 $m$ 行 $1$ 列矩阵：

```
B=\left[\begin{array}{l}
b_1 \\
b_2 \\
\dots \\
b_m
\end{array}\right]
\end{array}
\end{array}
```

如果设 $SXS$ 为 $n\times 1$ 的 $n$ 行 $1$ 列矩阵：

```
X=\left[\begin{array}{l}
x_1 \\
x_2 \\
\dots \\
x_n
\end{array}\right]
\end{array}
\end{array}
```

那么线性方程组 $SAS$ ，就可以表示为 $SAX=BS$ 的矩阵形式。如果我们再换一种表示形式，设： $Sa_1, a_2, \dots, a_n, \beta S$ 表示增广矩阵 $S\widetilde{A}S$ 的列向量，则线性方程组 $SAS$ 又可表示为 $Sa_1x_1+a_2x_2+\dots+a_nx_n=\beta S$ 。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中，你可以用它们来简化求解，甚至可以提升计算效率，就如之前提到的神经网络的隐藏层的输出计算、图形图像的三维空间变换。在数学中也是同样的，你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多，我们下一节课再来详细讲解求解过程。

通过前面的讲解，我相信你对矩阵有了一定的了解，现在我们再回头来看看矩阵的定义吧。

矩阵的定义是：一个 $(m, n)$ 矩阵 $SAS$ ，是由 $m\times n$ 个元素组成， $m$ 和 $n$ 是实数，其中元素 $Sa_{ij}, \mathbf{i}=1, \dots, \mathbf{m}, \mathbf{j}=1, \dots, \mathbf{n}$ 按 $m$ 行 $n$ 列的矩形排布方式后可以形成矩阵 $SAS$ ：

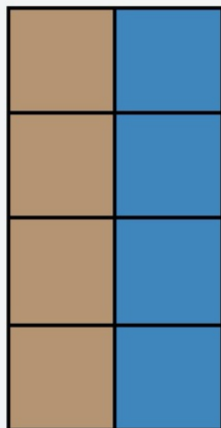
```
\begin{array}{cccc}
a_{11} & a_{12} & \dots & a_{1n} \\
a_{21} & a_{22} & \dots & a_{2n} \\
\dots & \dots & \dots & \dots \\
a_{m1} & a_{m2} & \dots & a_{mn}
\end{array}\right]
\end{array}
\end{array}
```

其中 $Sa_{ij}$ 属于实数或复数，在我们的场景中是实数 $SRS$ ，按通常的习惯， $(1, n)$ 矩阵叫做行， $(m, 1)$ 矩阵叫做列，这些特殊的矩阵叫做行或列向量。

定义完矩阵后，我接着讲一个比较有趣的概念，矩阵转换（Matrix transformation）。矩阵转换经常被用在计算机图形图像的转换中，比如，一张彩色图片从RGB角度来说说是三维的，如果要转换成灰度图片，也就是一维图片，那就要做矩阵转换。

我们来看一下矩阵转换的过程。设 $\mathbf{R}^{m\times n}$ 是实数矩阵 $(m, n)$ 的集合， $\mathbf{A} \in \mathbf{R}^{m\times n}$ 可以表示成另一种形式 $\mathbf{A} \in \mathbf{R}^{mm}$ 。我们把矩阵的 $n$ 列堆叠成一个长向量后完成转换。这个转换也叫做 $\text{reshape}$ ，其实就是重新调整原矩阵的行数、列数和维数，但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \dots & a_{1n}+b_{1n} \\
\vdots & \ddots & \vdots \\
\vdots & \ddots & \vdots \\
a_{m1}+b_{m1} & \dots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C = np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \dots, m, j=1, \dots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

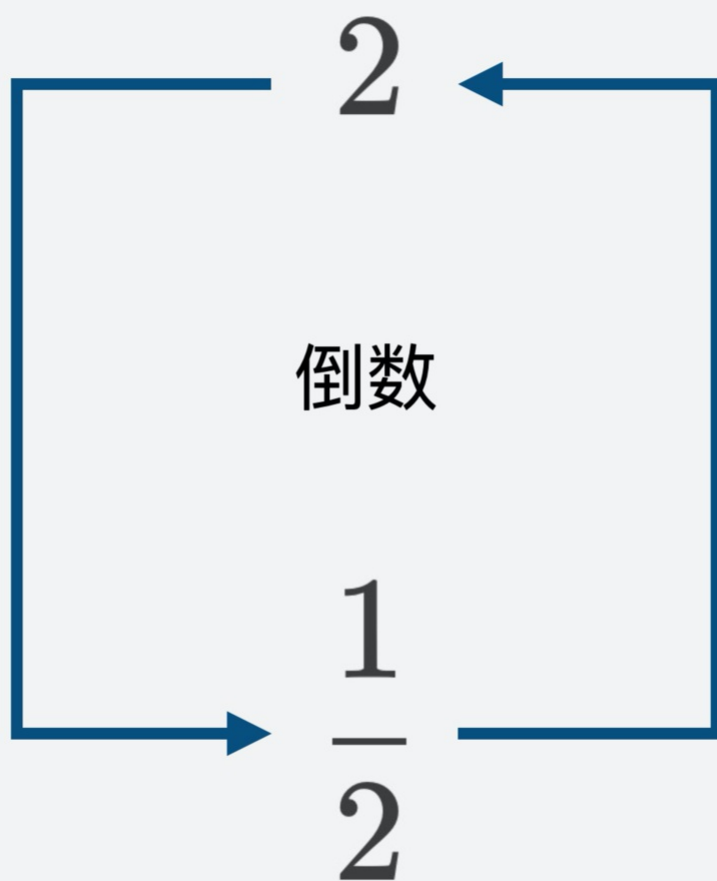
```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

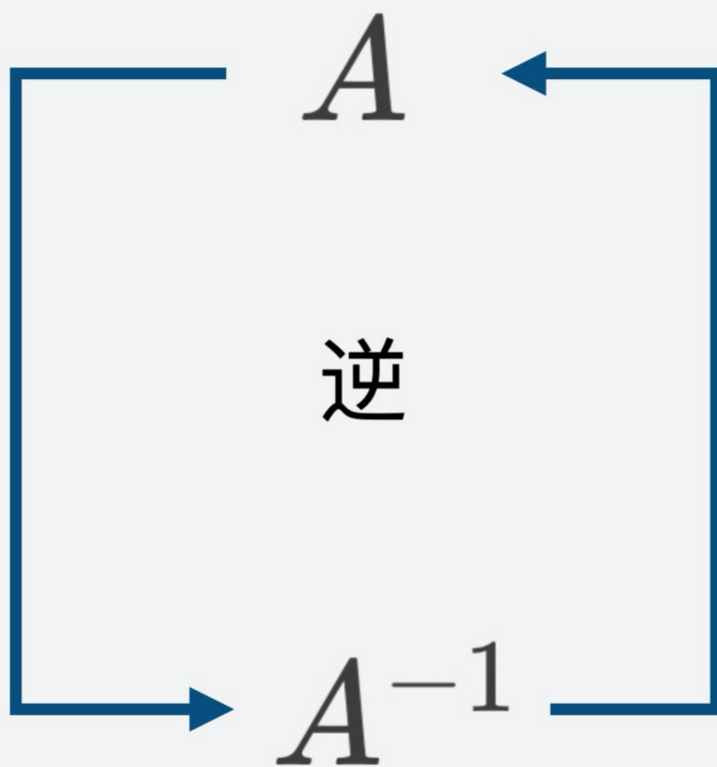
```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。





其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```
SS
A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\left[\begin{array}{cc}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{array}\right]
SS
```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```
SS
A \times A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right]
SS
```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程\$XA=BS\$。

小孩

大人

大巴

火车

$x_1$

$x_2$

3

3.5

3.2

3.6

要解\$XS\$，我们就要先计算\$A\$的逆矩阵\$A^{-1}\$：

```
$$
A^{-1}=\left[\begin{array}{cc}
3 & 3.5 \\
3.2 & 3.6
\end{array}\right]^{-1}=\frac{1}{3\times 3.6-3.5\times 3.2}\left[\begin{array}{cc}
3.6 & -3.5 \\
-3.2 & 3
\end{array}\right]
=\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
$$
```

接下来再计算\$X=BA^{-1}\$：

```
$$
\left[\begin{array}{l}
x_1 \\
x_2
\end{array}\right]=\left[\begin{array}{l}
118.4 \\
135.2
\end{array}\right]\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
=\left[\begin{array}{l}
16 \\
22
\end{array}\right]
$$
```

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least，方程次序很重要，也就是说，\$AX=BS\$和\$XA=BS\$的结果是不同的，这个一定要牢记哦！

转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把\$m\times n\$矩阵\$A\$的行列互换，得到转置矩阵\$A^T\$。

```
$$
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\ldots & \ldots & \ldots & \ldots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{array}\right]
$$
```

```
$$
A^T=\left[\begin{array}{cccc}
a_{11} & a_{21} & \ldots & a_{m1} \\
a_{12} & a_{22} & \ldots & a_{m2} \\
\ldots & \ldots & \ldots & \ldots \\
a_{1n} & a_{2n} & \ldots & a_{mn}
\end{array}\right]

```

\end{array}'right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

- 1. 矩阵和自身逆矩阵相乘得单位矩阵， $SA A^{-1}=I=A^{-1} A$ ；
- 2.  $SASBS$ 两矩阵相乘的逆，等于逆矩阵 $SBS$ 和逆矩阵 $SAS$ 相乘，这里强调一下乘的顺序很重要， $S(A B)^{-1}=B^{-1} A^{-1}$ ；
- 3.  $SABS$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $S(A+B)^{-1}\neq A^{-1}+B^{-1}$ ；
- 4. 矩阵转置的转置还是它本身， $S\left(A^T\right)^{\mathrm{T}}=A$ ；
- 5.  $SABS$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $S(A+B)^T=A^T+B^T$ ；
- 6.  $SABS$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $B$ 和转置矩阵 $A$ 的相乘，这里再次强调乘的顺序很重要， $S(A B)^T=B^T A^T$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $Sx=\left(x_{1}, \ldots, x_{10}\right)^T$ ， $Sv=\left(v_{1}, \ldots, v_{10}\right)^T$ ，如果要计算 $Sy=x^T\left(I+v v^T\right) x$ ，其中 $S$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

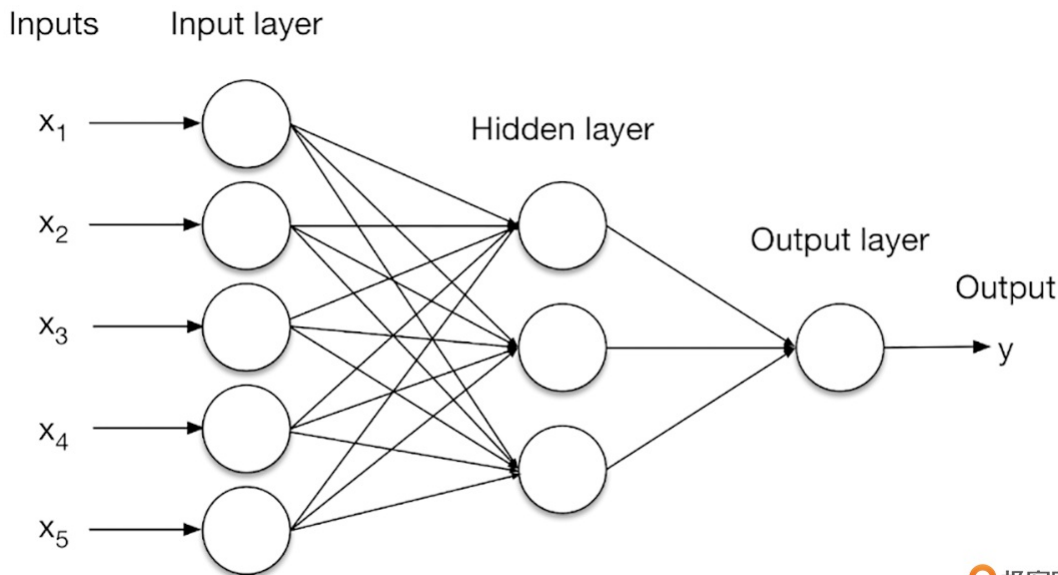
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$AX = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵 $\widetilde{A}$ ， 可以表示为\$(A, B)\$，这里的\$B\$表示的是方程组常数项所构成的列向量，也就是 $m\times 1$ 的 $m$ 行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为 $n\times 1$ 的 $n$ 行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$，就可以表示为\$AX=B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵 $\widetilde{A}$ 的列向量，则线性方程组\$A\$X=\$B\$又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由 $m\times n$ 个元素组成，  $m$ 和 $n$ 是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按 $m$ 行 $n$ 列的矩形排布方式后可以形成矩阵\$A\$：

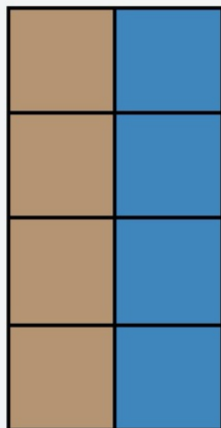
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例，  $(1, n)$ 矩阵叫做行，  $(m, 1)$ 矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设 $R^{m\times n}$ 是实数矩阵\$(m, n)\$的集合，  $A\in R^{m\times n}$ 可以表示成另一种形式  $A\in R^{mm}$ 。 我们把矩阵的 $n$ 列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \dots & a_{1n}+b_{1n} \\
\vdots & \ddots & \vdots \\
\vdots & \ddots & \vdots \\
a_{m1}+b_{m1} & \dots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \dots, m, j=1, \dots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。

\$\$AB=CS\$

但反过来B和A相乘就不行了，因为相邻阶数 $S_mS$ 不等于 $S_nS$ 。

2.哈达玛积

哈达玛积理解起来就很简单了，就是矩阵各对应元素的乘积， $c_{c_{ij}}=a_{ij} \times b_{ij}$ 。举个例子：

```
$$
C=A^{(*)} B=\left[\begin{array}{ll}
1 & 2 \\
4 & 5
\end{array}\right]\left[\begin{array}{ll}
1 & 4 \\
2 & 5
\end{array}\right]=\left[\begin{array}{cc}
1 * 1 & 2 * 4 \\
4 * 2 & 5 * 5
\end{array}\right]=\left[\begin{array}{cc}
1 & 8 \\
8 & 25
\end{array}\right]
$$
```

哈达玛积其实在数学中不常看到，不过，在编程中哈达玛积非常有用，因为它可以用来同时计算多组数据的乘积，计算效率很高。

3.克罗内克积

克罗内克积是以德国数学家利奥波德·克罗内克（Leopold Kronecker）的名字命名的。它可以应用在解线性矩阵方程和图像处理方面，当然从更时髦的角度说，它还能用在量子信息领域，我们也称之为直积或张量积。

和普通矩阵乘和哈达玛积不同的是，克罗内克积是两个任意大小矩阵间的运算，表示为 $A \times B$ ，如果 $A$ 是一个 $m \times n$ 的矩阵，而 $B$ 是一个 $p \times q$ 的矩阵，克罗内克积则是一个 $mp \times nq$ 的矩阵。

接下来我们需要定义一个在矩阵的乘法中起着特殊作用的矩阵，它就是单位矩阵。高等代数中，在求解相应的矩阵时，若添加单位矩阵，通过初等变换进行求解，往往可以使问题变得简单。按照百度百科的解释，单位矩阵如同数的乘法中的1，这种矩阵就被称为单位矩阵。它是个方阵，从左上角到右下角的对角线，也就是主对角线上的元素均为1，除此以外全都为0。

在线性代数中，大小为 $n$ 的单位矩阵就是在主对角线上均为1，而其他地方都是0的 $n \times n$ 的方阵，它用 $\mathbf{I}_n$ 表示，表达时为了方便可以忽略阶数，直接用 $\mathbf{I}$ 来表示：

```
$$
\mathbf{I}_1=[1], \mathbf{I}_2=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right], \mathbf{I}_3=\left[\begin{array}{lll}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right], \dots, \mathbf{I}_n=\left[\begin{array}{cccc}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
. & . & \dots & . \\
. & . & . & . \\
0 & 0 & \dots & 1
\end{array}\right]
$$
```

矩阵的性质

在了解了矩阵加和乘，以及单位矩阵后，我们是时候来看一看矩阵的性质了。了解矩阵的性质是进行矩阵计算的前提，就像我们小时候学加减乘除四则运算法则时那样。所以，这块内容对你来说应该不难，你作为了解就好，重点是之后的运算。

1.结合律

任意实数 $m \times n$ 矩阵 $A$ ， $n \times p$ 矩阵 $B$ ， $p \times q$ 矩阵 $C$ 之间相乘，满足结合律 $(AB)C=A(BC)$ 。这个很好理解，我就不多说了。

\$\$\forall a \in R^{\{m \times n\}}, B \in R^{\{n \times p\}}, C \in R^{\{p \times q\}} \{ (A B) C=A(B C) \$\$

2.分配律

任意实数 $m \times n$ 矩阵 $A$ 和 $B$ ， $n \times p$ 矩阵 $C$ 和 $D$ 之间相乘满足分配律 $(A+B)C=AC+BC$ ， $A(C+D)=AC+AD$ 。

\$\$\forall a \in \mathbf{R}^{\{m \times n\}}, B \in \mathbf{R}^{\{n \times p\}}, C, D \in \mathbf{R}^{\{n \times p\}} \{ (A+B) C=A C+B C, A(C+D)=A C+A D \$\$

3.单位矩阵乘

任意实数 $m \times n$ 矩阵A和单位矩阵之间的乘，等于它本身 $A$ 。

\$\$\forall a \in R^{\{m \times n\}}: \mathbf{I}\_m A=A \mathbf{I}\_n=A \$\$

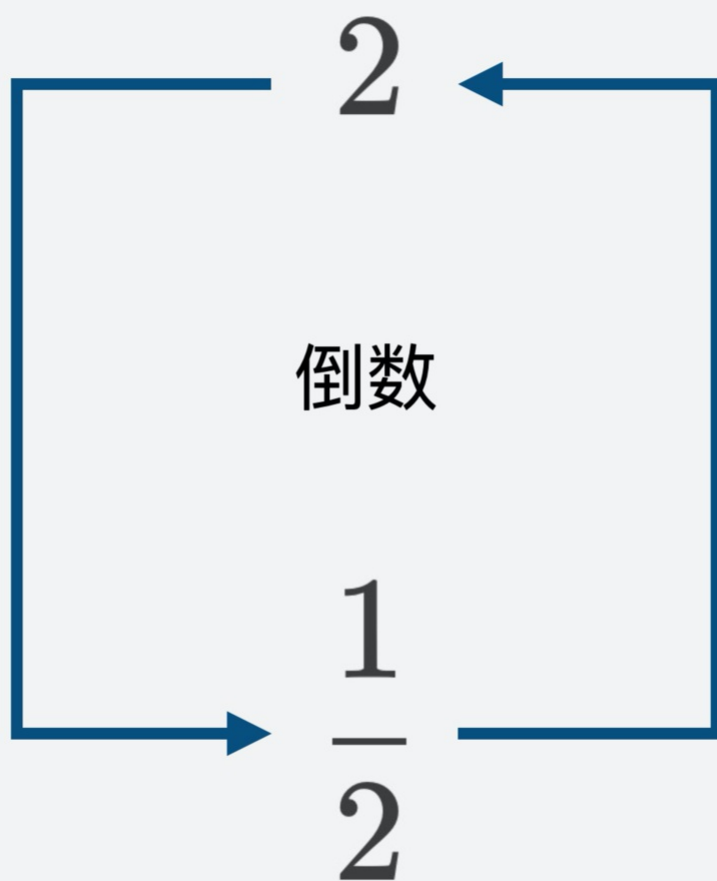
注意，这里的行和列不同， $m \neq n$ 意味着，根据矩阵乘，左乘和右乘单位矩阵也不同，也就是 $\mathbf{I}_m \neq \mathbf{I}_n$ 。

逆矩阵与转置矩阵

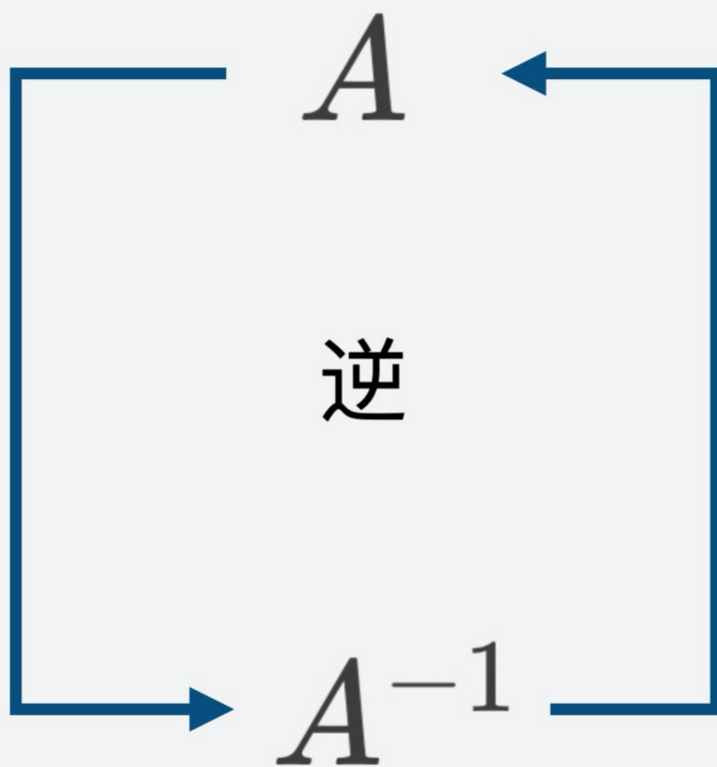
了解矩阵基本概念、运算，以及性质后，我来讲一讲矩阵应用中的两个核心内容——逆矩阵和转置矩阵。逆矩阵和转置矩阵在实际应用中大有用处，比如：坐标系中的图形变换运算。我们先来看下什么是逆矩阵。

逆矩阵

下面这个图你应该非常熟悉了，图中表现的是数字的倒数，2的倒数是 $\frac{1}{2}$ ， $\frac{1}{2}$ 的倒数是2。



其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```
SS
A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\left[\begin{array}{cc}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{array}\right]
SS
```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```
SS
A \times A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right]
SS
```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程 $XA=B$ 。

小孩	大人	大巴	火车
$x_1$	$x_2$	3	3.5
		3.2	3.6

要解 $\mathbf{X}\mathbf{S}$ ，我们就要先计算 $\mathbf{A}\mathbf{S}$ 的逆矩阵 $\mathbf{A}^{-1}\mathbf{S}$ ：

$$\frac{A^{-1}}{3 \wedge 3.5} = \left( \begin{array}{c} cc \\ 3.6 \wedge -3.5 \end{array} \right)$$

接下来再计算  $X = B A^{-1} S$ :

$$\begin{aligned} & \left\lfloor \begin{array}{l} \text{begin}\{array\}\{II\} \\ x_1\} \text{ and } x_2\} \\ \text{end}\{array\}\text{right} \end{array} \right\rfloor = \left\lfloor \begin{array}{l} \text{begin}\{array\}\{II\} \\ 118.4 \text{ and } 135.2 \\ \text{end}\{array\}\text{right} \end{array} \right\rfloor \left\lfloor \begin{array}{l} \text{begin}\{array\}\{cc\} \\ -9 \text{ and } 8.75 \\ \text{end}\{array\}\text{right} \end{array} \right\rfloor \\ & \left\lfloor \begin{array}{l} \text{begin}\{array\}\{II\} \\ 8 \text{ and } -7.5 \\ \text{end}\{array\}\text{right} \end{array} \right\rfloor = \left\lfloor \begin{array}{l} \text{begin}\{array\}\{II\} \\ 16 \text{ and } 22 \\ \text{end}\{array\}\text{right} \end{array} \right\rfloor \end{aligned}$$

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序包封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least, 方程次序很重要, 也就是说,  $SAX=BS$  和  $XA=BS$  的结果是不同的, 这个一定要牢记哦!

转置矩阵

一般伴随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把  $m \times n$  矩阵  $A$  的行列互换，得到转置矩阵  $A^T$ 。

```

\SS
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \& \dots & a_{1n} \\
a_{21} & a_{22} & \& \dots & a_{2n} \\
\& \dots & \& \dots & \& \dots \\
a_{m1} & a_{m2} & \& \dots & a_{mn}
\end{array}\right]
\SS

```

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

\end{array}\right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

1. 矩阵和自身逆矩阵相乘得单位矩阵， $SA A^{-1}=I=A^{-1} A$ ；
2.  $SASBS$ 两矩阵相乘的逆，等于逆矩阵 $SBS$ 和逆矩阵 $SAS$ 相乘，这里强调一下乘的顺序很重要， $S(A B)^{-1}=B^{-1} A^{-1}$ ；
3.  $SABS$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $S(A+B)^{-1}\neq A^{-1}+B^{-1}$ ；
4. 矩阵转置的转置还是它本身， $S\left(A^T\right)^{\mathrm{T}}=A$ ；
5.  $SABS$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $S(A+B)^T=A^T+B^T$ ；
6.  $SABS$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $B$ 和转置矩阵 $A$ 的相乘，这里再次强调乘的顺序很重要， $S(A B)^T=B^T A^T$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $S=\left(x_1, \ldots, x_{10}\right)^T$ ， $S_v=\left(v_1, \ldots, v_{10}\right)^T$ ，如果要计算 $S_y=x^T\left(I+v v^T\right) x$ ，其中 $I$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

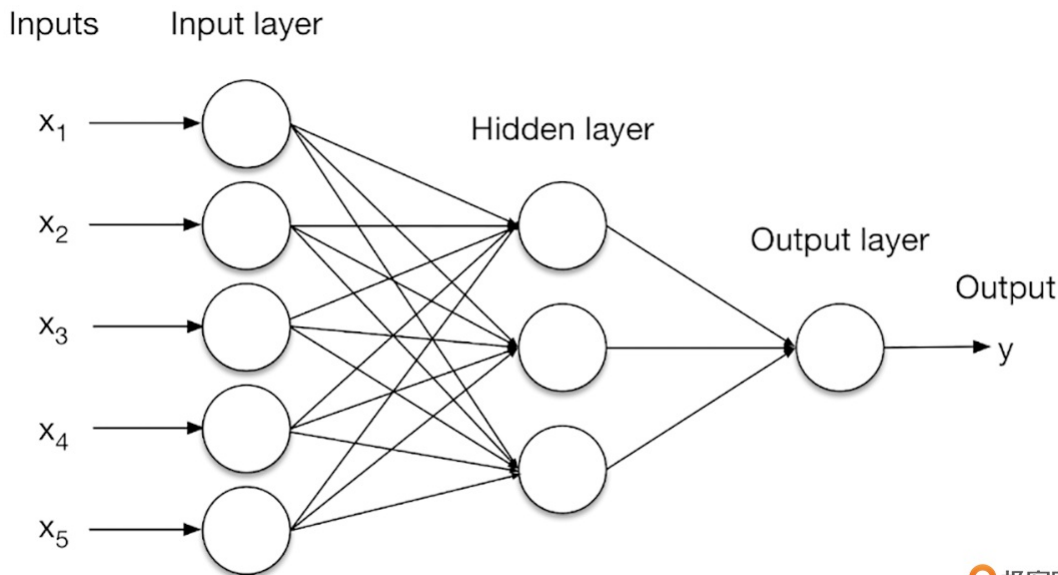
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$AX = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵\$\widetilde{A}\$， 可以表示为\$(A, B)\$， 这里的\$B\$表示的是方程组常数项所构成的列向量， 也就是\$m\times 1\$的\$m\$行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为\$n\times 1\$的\$n\$行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$， 就可以表示为\$A\$X=\$B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵\$\widetilde{A}\$的列向量， 则线性方程组\$A\$X又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、 图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由\$m\times n\$个元素组成， \$m\$和\$n\$是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按\$m\$行\$n\$列的矩形排布方式后可以形成矩阵\$A\$：

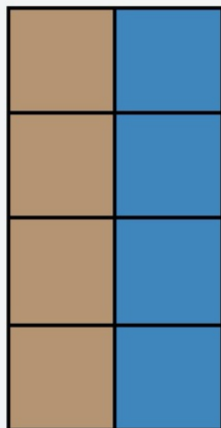
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例， \$(1, n)\$矩阵叫做行， \$(m, 1)\$矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。 矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设\$R^{m\times n}\$是实数矩阵\$(m, n)\$的集合， \$A\in R^{m\times n}\$可以表示成另一种形式 \$A\in R^{mm}\$。 我们把矩阵的\$n\$列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、 列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \cdots & a_{1n}+b_{1n} \\
\cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots \\
a_{m1}+b_{m1} & \cdots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \cdots, m, j=1, \cdots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。

\$\$AB=CS\$

但反过来B和A相乘就不行了，因为相邻阶数 $S_mS$ 不等于 $S_nS$ 。

2.哈达玛积

哈达玛积理解起来就很简单了，就是矩阵各对应元素的乘积， $c_{c_{ij}}=a_{ij} \times b_{ij}$ 。举个例子：

```
$$
C=A^{(*)} B=\left[\begin{array}{ll}
1 & 2 \\
4 & 5
\end{array}\right]\left[\begin{array}{ll}
1 & 4 \\
2 & 5
\end{array}\right]=\left[\begin{array}{cc}
1 * 1 & 2 * 4 \\
4 * 2 & 5 * 5
\end{array}\right]=\left[\begin{array}{cc}
1 & 8 \\
8 & 25
\end{array}\right]
$$
```

哈达玛积其实在数学中不常看到，不过，在编程中哈达玛积非常有用，因为它可以用来同时计算多组数据的乘积，计算效率很高。

3.克罗内克积

克罗内克积是以德国数学家利奥波德·克罗内克（Leopold Kronecker）的名字命名的。它可以应用在解线性矩阵方程和图像处理方面，当然从更时髦的角度说，它还能用在量子信息领域，我们也称之为直积或张量积。

和普通矩阵乘和哈达玛积不同的是，克罗内克积是两个任意大小矩阵间的运算，表示为 $A \times B$ ，如果 $A$ 是一个 $m \times n$ 的矩阵，而 $B$ 是一个 $p \times q$ 的矩阵，克罗内克积则是一个 $m \times p \times n \times q$ 的矩阵。

接下来我们需要定义一个在矩阵的乘法中起着特殊作用的矩阵，它就是单位矩阵。高等代数中，在求解相应的矩阵时，若添加单位矩阵，通过初等变换进行求解，往往可以使问题变得简单。按照百度百科的解释，单位矩阵如同数的乘法中的1，这种矩阵就被称为单位矩阵。它是个方阵，从左上角到右下角的对角线，也就是主对角线上的元素均为1，除此以外全都为0。

在线性代数中，大小为 $n$ 的单位矩阵就是在主对角线上均为1，而其他地方都是0的 $n \times n$ 的方阵，它用 $\mathbf{I}_n$ 表示，表达时为了方便可以忽略阶数，直接用 $\mathbf{I}$ 来表示：

```
$$
\mathbf{I}_1=[1], \mathbf{I}_2=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right], \mathbf{I}_3=\left[\begin{array}{lll}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right], \dots, \mathbf{I}_n=\left[\begin{array}{cccc}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
. & . & \dots & . \\
. & . & . & . \\
0 & 0 & \dots & 1
\end{array}\right]
$$
```

矩阵的性质

在了解了矩阵加和乘，以及单位矩阵后，我们是时候来看一看矩阵的性质了。了解矩阵的性质是进行矩阵计算的前提，就像我们小时候学加减乘除四则运算法则时那样。所以，这块内容对你来说应该不难，你作为了解就好，重点是之后的运算。

1.结合律

任意实数 $m \times n$ 矩阵 $A$ ， $n \times p$ 矩阵 $B$ ， $p \times q$ 矩阵 $C$ 之间相乘，满足结合律 $(AB)C=A(BC)$ 。这个很好理解，我就不多说了。

\$\$\forall a \in R^m \times n, B \in R^n \times p, C \in R^p \times q \{ (A B) C=A(B C)\$\$

2.分配律

任意实数 $m \times n$ 矩阵 $A$ 和 $B$ ， $n \times p$ 矩阵 $C$ 和 $D$ 之间相乘满足分配律 $(A+B)C=AC+BC$ ， $A(C+D)=AC+AD$ 。

\$\$\forall a \in \mathbf{M}\{A\}, B \in \mathbf{M}\{R\}^m \times n, C, D \in \mathbf{M}\{R\}^n \times p \{ (A+B) C=A C+B C, A(C+D)=A C+A D\$

3.单位矩阵乘

任意实数 $m \times n$ 矩阵A和单位矩阵之间的乘，等于它本身 $A$ 。

\$\$\forall a \in R^m \times n \{ \mathbf{I}\_m A=A \mathbf{I}\_n=A\$

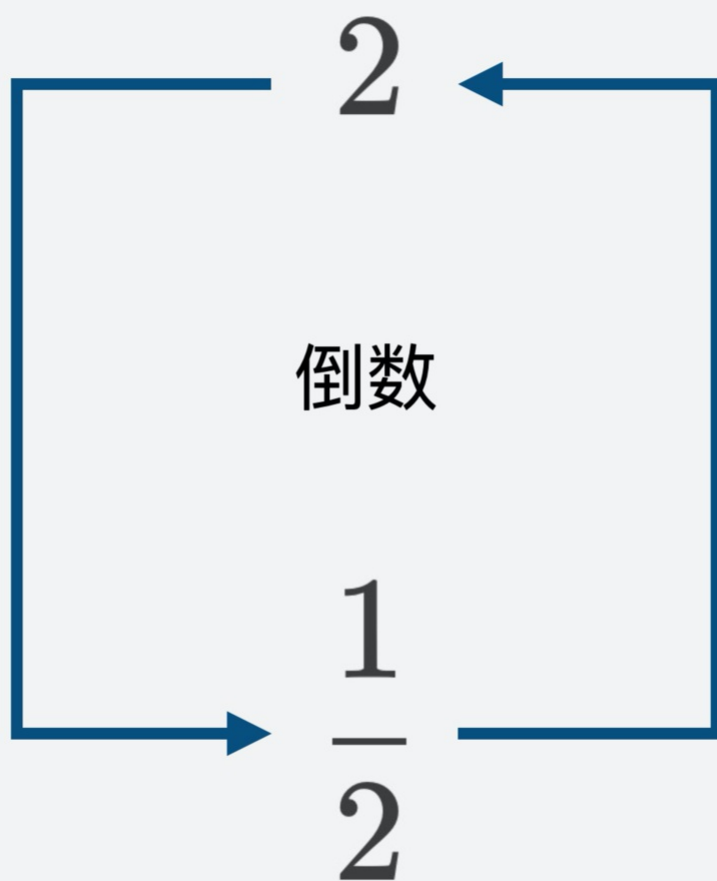
注意，这里的行和列不同， $m \neq n$ 意味着，根据矩阵乘，左乘和右乘单位矩阵也不同，也就是 $\mathbf{I}_m \neq \mathbf{I}_n$ 。

逆矩阵与转置矩阵

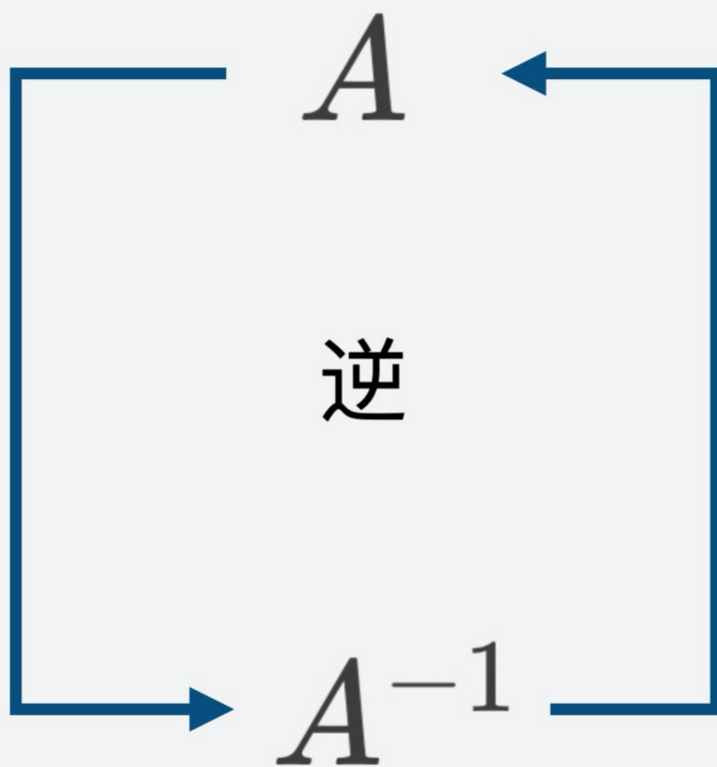
了解矩阵基本概念、运算，以及性质后，我来讲一讲矩阵应用中的两个核心内容——逆矩阵和转置矩阵。逆矩阵和转置矩阵在实际应用中大有用处，比如：坐标系中的图形变换运算。我们先来看下什么是逆矩阵。

逆矩阵

下面这个图你应该非常熟悉了，图中表现的是数字的倒数，2的倒数是1/2，1/2的倒数是2。



其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```
SS
A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\left[\begin{array}{cc}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{array}\right]
SS
```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```
SS
A \times A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]
SS
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{11}+\frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{21}=\frac{a_{11}a_{22}-a_{12}a_{21}}{a_{11}a_{22}-a_{12}a_{21}}=1
\frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{11}+\frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{22}=\frac{-a_{12}a_{11}+a_{11}a_{22}}{a_{11}a_{22}-a_{12}a_{21}}=0
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{11}+\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{21}=\frac{-a_{21}a_{11}+a_{22}a_{21}}{a_{11}a_{22}-a_{12}a_{21}}=0
\frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{12}+\frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{22}=\frac{a_{11}a_{12}-a_{12}a_{22}}{a_{11}a_{22}-a_{12}a_{21}}=-1
\end{array}\right]
SS
```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程\$XA=BS\$。

小孩          大人          大巴          火车

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 3.5 \\ 3.2 & 3.6 \end{bmatrix}$$

要解\$XS\$，我们就要先计算\$A\$的逆矩阵\$A^{-1}\$：

```
$$
A^{-1}=\left[\begin{array}{cc}
3 & 3.5 \\
3.2 & 3.6
\end{array}\right]^{-1}=\frac{1}{3\times 3.6-3.5\times 3.2}\left[\begin{array}{cc}
3.6 & -3.5 \\
-3.2 & 3
\end{array}\right]
=\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
$$
```

接下来再计算\$X=BA^{-1}\$：

```
$$
\left[\begin{array}{ll}
x_1 & x_2
\end{array}\right]=\left[\begin{array}{ll}
118.4 & 135.2
\end{array}\right]\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
=\left[\begin{array}{ll}
16 & 22
\end{array}\right]
$$
```

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序包封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least，方程次序很重要，也就是说，\$AX=BS\$和\$XA=BS\$的结果是不同的，这个一定要牢记哦！

转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把\$m\times n\$矩阵\$A\$的行列互换，得到转置矩阵\$A^T\$。

```
$$
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\ldots & \ldots & \ldots & \ldots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{array}\right]
$$
```

```
$$
A^T=\left[\begin{array}{cccc}
a_{11} & a_{21} & \ldots & a_{m1} \\
a_{12} & a_{22} & \ldots & a_{m2} \\
\ldots & \ldots & \ldots & \ldots \\
a_{1n} & a_{2n} & \ldots & a_{mn}
\end{array}\right]

```

\end{array}'right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

- 1. 矩阵和自身逆矩阵相乘得单位矩阵， $SA A^{-1}=I=A^{-1} A$ ；
- 2.  $SASBS$ 两矩阵相乘的逆，等于逆矩阵 $SBS$ 和逆矩阵 $SAS$ 相乘，这里强调一下乘的顺序很重要， $S(A B)^{-1}=B^{-1} A^{-1}$ ；
- 3.  $SABS$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $S(A+B)^{-1}\neq A^{-1}+B^{-1}$ ；
- 4. 矩阵转置的转置还是它本身， $S\left(A^T\right)^{\mathrm{T}}=A$ ；
- 5.  $SABS$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $S(A+B)^T=A^T+B^T$ ；
- 6.  $SABS$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $B$ 和转置矩阵 $A$ 的相乘，这里再次强调乘的顺序很重要， $S(A B)^T=B^T A^T$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $Sx=\left(x_{1}, \ldots, x_{10}\right)^T$ ， $Sv=\left(v_{1}, \ldots, v_{10}\right)^T$ ，如果要计算 $Sy=x^T\left(I+v v^T\right) x$ ，其中 $S$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

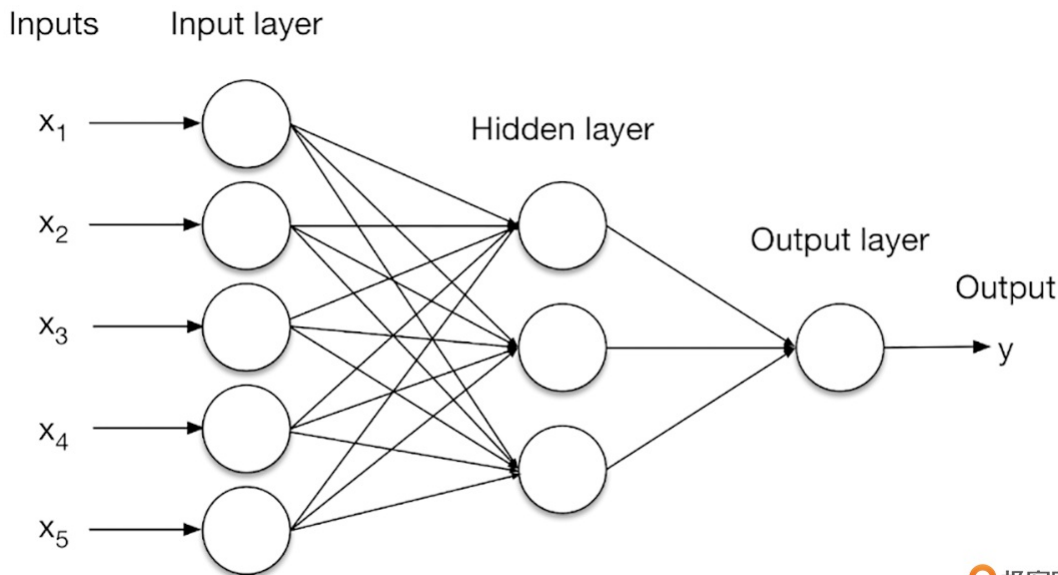
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$AX = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵 $\widetilde{A}$ ， 可以表示为\$(A, B)\$， 这里的\$B\$表示的是方程组常数项所构成的列向量， 也就是 $m\times 1$ 的 $m$ 行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为 $n\times 1$ 的 $n$ 行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$， 就可以表示为\$AX=B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵 $\widetilde{A}$ 的列向量， 则线性方程组\$A\$X=\$B\$又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、 图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由 $m\times n$ 个元素组成，  $m$ 和 $n$ 是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按 $m$ 行 $n$ 列的矩形排布方式后可以形成矩阵\$A\$：

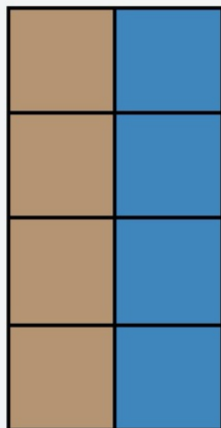
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例，  $(1, n)$ 矩阵叫做行，  $(m, 1)$ 矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。 矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设 $R^{m\times n}$ 是实数矩阵\$(m, n)\$的集合，  $A\in R^{m\times n}$ 可以表示成另一种形式  $A\in R^{mm}$ 。 我们把矩阵的 $n$ 列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、 列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \dots & a_{1n}+b_{1n} \\
\vdots & \ddots & \vdots \\
\vdots & \ddots & \vdots \\
a_{m1}+b_{m1} & \dots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \dots, m, j=1, \dots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

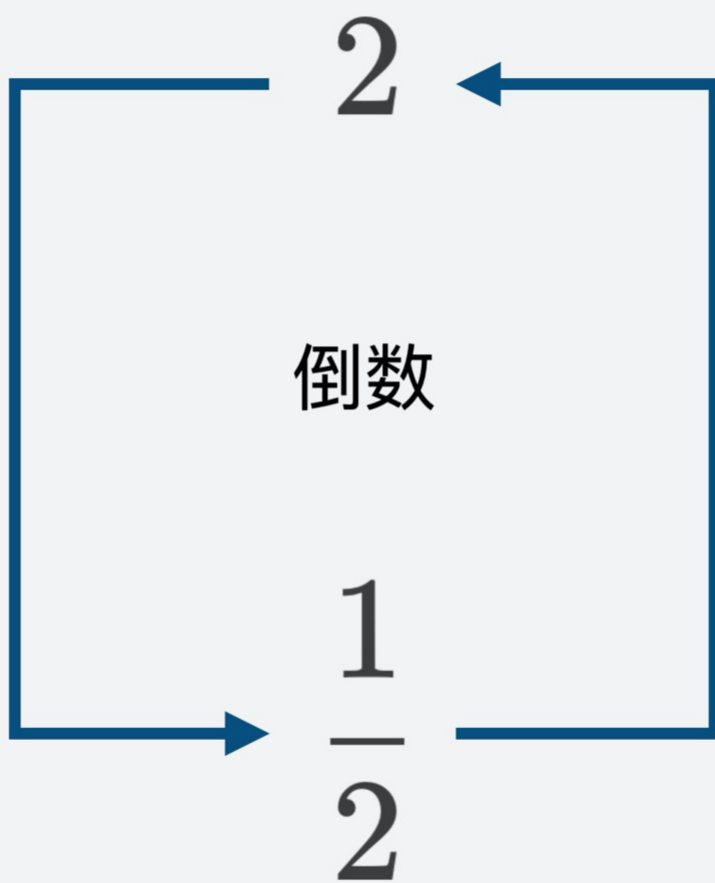
```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

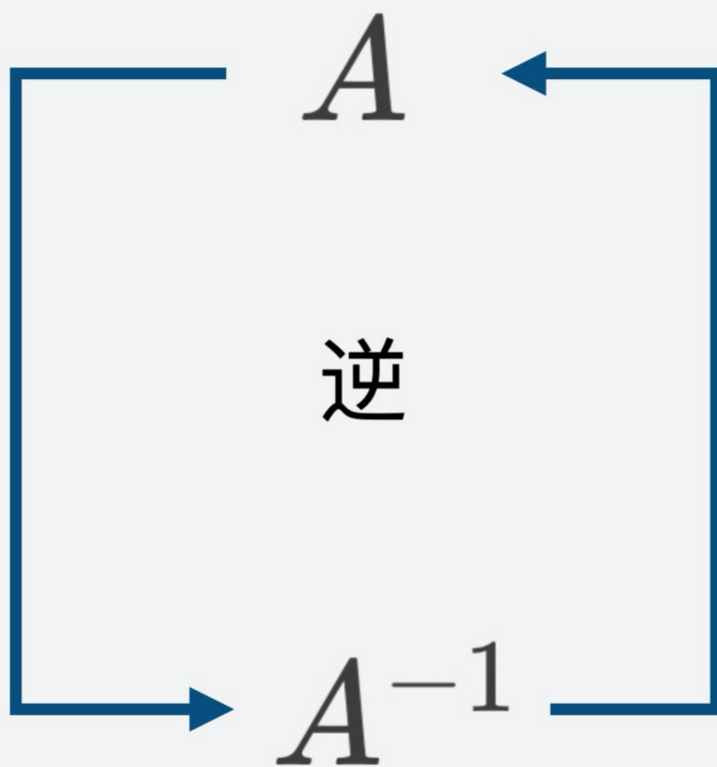
```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。





其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{SA} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```
SS
A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\left[\begin{array}{cc}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{array}\right]
SS
```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```
SS
A \times A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right]
SS
```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程\$XA=BS\$。

小孩          大人          大巴          火车

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 3 & 3.5 \\ 3.2 & 3.6 \end{bmatrix}$$

要解\$XS\$，我们就要先计算\$A\$的逆矩阵\$A^{-1}\$：

```
$$
A^{-1}=\left[\begin{array}{cc}
3 & 3.5 \\
3.2 & 3.6
\end{array}\right]^{-1}=\frac{1}{3\times 3.6-3.5\times 3.2}\left[\begin{array}{cc}
3.6 & -3.5 \\
-3.2 & 3
\end{array}\right]
=\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
$$
```

接下来再计算\$X=BA^{-1}\$：

```
$$
\left[\begin{array}{ll}
x_1 & x_2
\end{array}\right]=\left[\begin{array}{ll}
118.4 & 135.2
\end{array}\right]\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
=\left[\begin{array}{ll}
16 & 22
\end{array}\right]
$$
```

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序包封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least，方程次序很重要，也就是说，\$AX=BS\$和\$XA=BS\$的结果是不同的，这个一定要牢记哦！

转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把\$m\times n\$矩阵\$A\$的行列互换，得到转置矩阵\$A^T\$。

```
$$
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\ldots & \ldots & \ldots & \ldots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{array}\right]
$$
```

```
$$
A^T=\left[\begin{array}{cccc}
a_{11} & a_{21} & \ldots & a_{m1} \\
a_{12} & a_{22} & \ldots & a_{m2} \\
\ldots & \ldots & \ldots & \ldots \\
a_{1n} & a_{2n} & \ldots & a_{mn}
\end{array}\right]

```

\end{array}\right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

1. 矩阵和自身逆矩阵相乘得单位矩阵， $\mathbf{A} \mathbf{A}^{-1} = \mathbf{I} = \mathbf{A}^{-1} \mathbf{A}$ ；
2.  $\mathbf{A} \mathbf{B}$ 两矩阵相乘的逆，等于逆矩阵 $\mathbf{B}$ 和逆矩阵 $\mathbf{A}$ 相乘，这里强调一下乘的顺序很重要， $(\mathbf{A} \mathbf{B})^{-1} = \mathbf{B}^{-1} \mathbf{A}^{-1}$ ；
3.  $\mathbf{A} \mathbf{B}$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $(\mathbf{A} + \mathbf{B})^{-1} \neq \mathbf{A}^{-1} + \mathbf{B}^{-1}$ ；
4. 矩阵转置的转置还是它本身， $\left(\mathbf{A}^T\right)^T = \mathbf{A}$ ；
5.  $\mathbf{A} \mathbf{B}$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$ ；
6.  $\mathbf{A} \mathbf{B}$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $\mathbf{B}$ 和转置矩阵 $\mathbf{A}$ 的相乘，这里再次强调乘的顺序很重要， $(\mathbf{A} \mathbf{B})^T = \mathbf{B}^T \mathbf{A}^T$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $\mathbf{x} = \left(x_1, \dots, x_{10}\right)^T$ ， $\mathbf{S} = \left(v_1, \dots, v_{10}\right)^T$ ，如果要计算 $\mathbf{y} = \mathbf{x}^T \left(\mathbf{I} + \mathbf{v} \mathbf{v}^T\right) \mathbf{x}$ ，其中 $\mathbf{I}$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

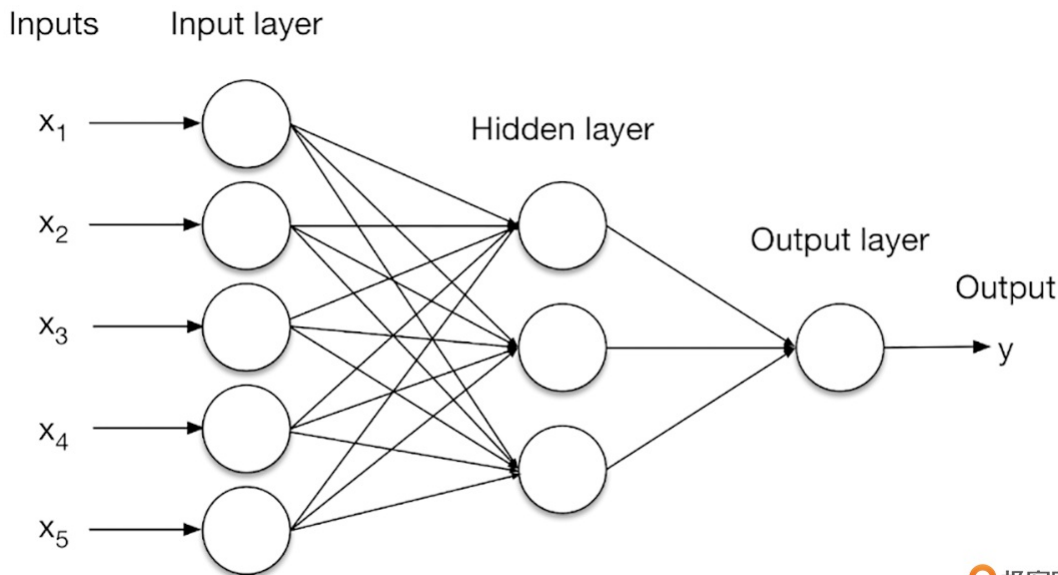
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{41} & w_{42} \\ w_{31} & w_{32} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} \\ w_{41} & w_{42} \\ w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$A \cdot X = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵 $\widetilde{A}$ ， 可以表示为\$(A, B)\$， 这里的\$B\$表示的是方程组常数项所构成的列向量， 也就是 $m\times 1$ 的 $m$ 行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为 $n\times 1$ 的 $n$ 行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$， 就可以表示为\$AX=B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵 $\widetilde{A}$ 的列向量， 则线性方程组\$A\$X=\$B\$又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、 图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由 $m\times n$ 个元素组成，  $m$ 和 $n$ 是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按\$m\$行 $n$ 列的矩形排布方式后可以形成矩阵\$A\$：

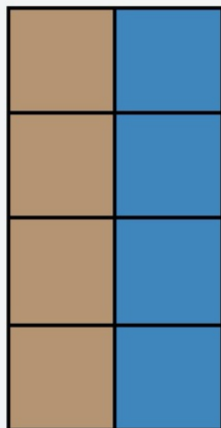
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例，  $(1, n)$ 矩阵叫做行，  $(m, 1)$ 矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。 矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设 $R^{m\times n}$ 是实数矩阵\$(m, n)\$的集合，  $A\in R^{m\times n}$ 可以表示成另一种形式  $A\in R^{mm}$ 。 我们把矩阵的 $n$ 列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、 列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \cdots & a_{1n}+b_{1n} \\
\cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots \\
a_{m1}+b_{m1} & \cdots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \cdots, m, j=1, \cdots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。

\$\$AB=CS\$

但反过来B和A相乘就不行了，因为相邻阶数 $S_mS$ 不等于 $S_nS$ 。

2.哈达玛积

哈达玛积理解起来就很简单了，就是矩阵各对应元素的乘积， $c_{c_{ij}}=a_{ij} \times b_{ij}$ 。举个例子：

```
$$
C=A^{(*)} B=\left[\begin{array}{ll}
1 & 2 \\
4 & 5
\end{array}\right]\left[\begin{array}{ll}
1 & 4 \\
2 & 5
\end{array}\right]=\left[\begin{array}{cc}
1 * 1 & 2 * 4 \\
4 * 2 & 5 * 5
\end{array}\right]=\left[\begin{array}{cc}
1 & 8 \\
8 & 25
\end{array}\right]
$$
```

哈达玛积其实在数学中不常看到，不过，在编程中哈达玛积非常有用，因为它可以用来同时计算多组数据的乘积，计算效率很高。

3.克罗内克积

克罗内克积是以德国数学家利奥波德·克罗内克（Leopold Kronecker）的名字命名的。它可以应用在解线性矩阵方程和图像处理方面，当然从更时髦的角度说，它还能用在量子信息领域，我们也称之为直积或张量积。

和普通矩阵乘和哈达玛积不同的是，克罗内克积是两个任意大小矩阵间的运算，表示为 $A \times B$ ，如果 $A$ 是一个 $m \times n$ 的矩阵，而 $B$ 是一个 $p \times q$ 的矩阵，克罗内克积则是一个 $mp \times nq$ 的矩阵。

接下来我们需要定义一个在矩阵的乘法中起着特殊作用的矩阵，它就是单位矩阵。高等代数中，在求解相应的矩阵时，若添加单位矩阵，通过初等变换进行求解，往往可以使问题变得简单。按照百度百科的解释，单位矩阵如同数的乘法中的1，这种矩阵就被称为单位矩阵。它是个方阵，从左上角到右下角的对角线，也就是主对角线上的元素均为1，除此以外全都为0。

在线性代数中，大小为 $n$ 的单位矩阵就是在主对角线上均为1，而其他地方都是0的 $n \times n$ 的方阵，它用 $\mathbf{I}_n$ 表示，表达时为了方便可以忽略阶数，直接用 $\mathbf{I}$ 来表示：

```
$$
\mathbf{I}_1=[1], \mathbf{I}_2=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right], \mathbf{I}_3=\left[\begin{array}{lll}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right], \dots, \mathbf{I}_n=\left[\begin{array}{cccc}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
. & . & \dots & . \\
. & . & . & . \\
0 & 0 & \dots & 1
\end{array}\right]
$$
```

矩阵的性质

在了解了矩阵加和乘，以及单位矩阵后，我们是时候来看一看矩阵的性质了。了解矩阵的性质是进行矩阵计算的前提，就像我们小时候学加减乘除四则运算法则时那样。所以，这块内容对你来说应该不难，你作为了解就好，重点是之后的运算。

1.结合律

任意实数 $m \times n$ 矩阵 $A$ ， $n \times p$ 矩阵 $B$ ， $p \times q$ 矩阵 $C$ 之间相乘，满足结合律 $(AB)C=A(BC)$ 。这个很好理解，我就不多说了。

\$\$\forall a \in R^{\{m \times n\}}, B \in R^{\{n \times p\}}, C \in R^{\{p \times q\}} \{ (A B) C = A (B C) \$\$

2.分配律

任意实数 $m \times n$ 矩阵 $A$ 和 $B$ ， $n \times p$ 矩阵 $C$ 和 $D$ 之间相乘满足分配律 $(A+B)C=AC+BC$ ， $A(C+D)=AC+AD$ 。

\$\$\forall a \in \mathbf{R}^{\{m \times n\}}, B \in \mathbf{R}^{\{n \times p\}}, C, D \in \mathbf{R}^{\{n \times p\}} \{ (A+B) C = A C + B C, A (C+D) = A C + A D \$\$

3.单位矩阵乘

任意实数 $m \times n$ 矩阵A和单位矩阵之间的乘，等于它本身 $A$ 。

\$\$\forall a \in R^{\{m \times n\}}: \mathbf{I}\_m A = A \mathbf{I}\_n = A \$\$

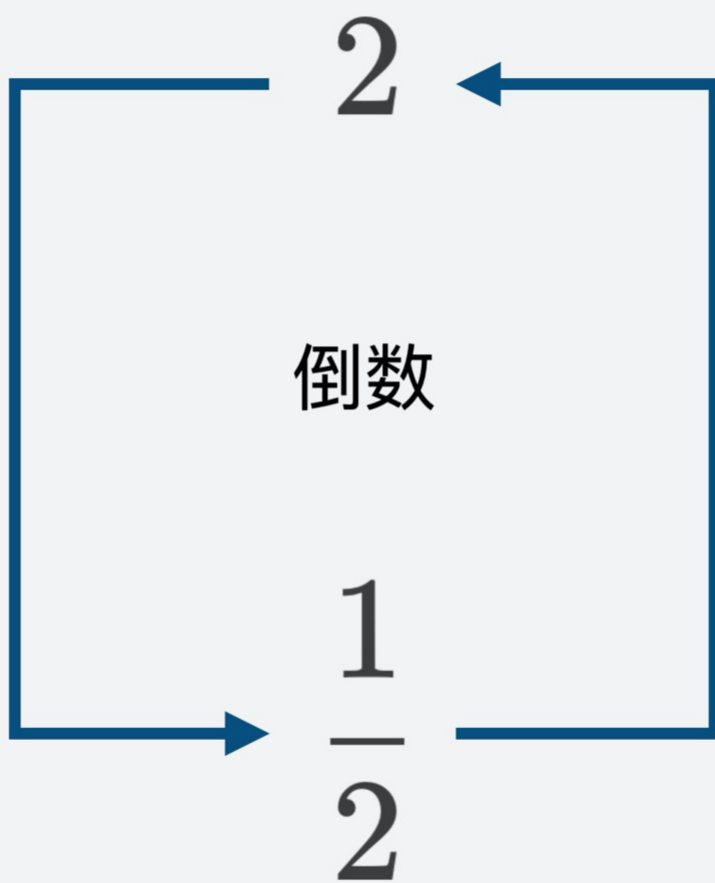
注意，这里的行和列不同， $m \neq n$ 意味着，根据矩阵乘，左乘和右乘单位矩阵也不同，也就是 $\mathbf{I}_m \neq \mathbf{I}_n$ 。

逆矩阵与转置矩阵

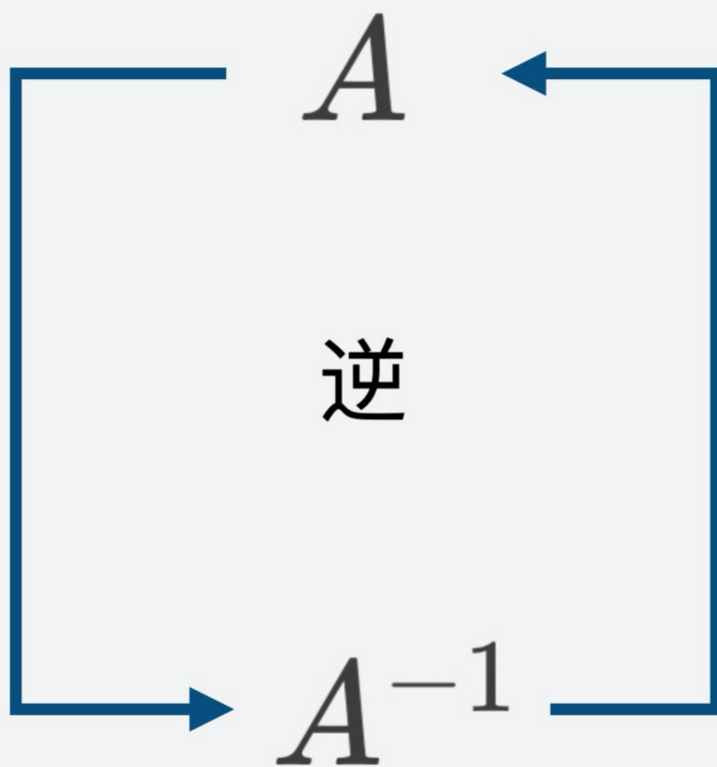
了解矩阵基本概念、运算，以及性质后，我来讲一讲矩阵应用中的两个核心内容——逆矩阵和转置矩阵。逆矩阵和转置矩阵在实际应用中大有用处，比如：坐标系中的图形变换运算。我们先来看下什么是逆矩阵。

逆矩阵

下面这个图你应该非常熟悉了，图中表现的是数字的倒数，2的倒数是 $\frac{1}{2}$ ， $\frac{1}{2}$ 的倒数是2。



其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```
SS
A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\left[\begin{array}{cc}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{array}\right]
SS
```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```
SS
A \times A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right]
SS
```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程\$XA=BS\$。

小孩          大人          大巴          火车

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 3 & 3.5 \\ 3.2 & 3.6 \end{bmatrix}$$

要解\$XS\$，我们就要先计算\$A\$的逆矩阵\$A^{-1}\$：

```
$$
A^{-1}=\left[\begin{array}{cc}
3 & 3.5 \\
3.2 & 3.6
\end{array}\right]^{-1}=\frac{1}{3\times 3.6-3.5\times 3.2}\left[\begin{array}{cc}
3.6 & -3.5 \\
-3.2 & 3
\end{array}\right]
=\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
$$
```

接下来再计算\$X=BA^{-1}\$：

```
$$
\left[\begin{array}{ll}
x_1 & x_2
\end{array}\right]=\left[\begin{array}{ll}
118.4 & 135.2
\end{array}\right]\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
=\left[\begin{array}{ll}
16 & 22
\end{array}\right]
$$
```

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least，方程次序很重要，也就是说，\$AX=BS\$和\$XA=BS\$的结果是不同的，这个一定要牢记哦！

转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把\$m\times n\$矩阵\$A\$的行列互换，得到转置矩阵\$A^T\$。

```
$$
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\ldots & \ldots & \ldots & \ldots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{array}\right]
$$
```

```
$$
A^T=\left[\begin{array}{cccc}
a_{11} & a_{21} & \ldots & a_{m1} \\
a_{12} & a_{22} & \ldots & a_{m2} \\
\ldots & \ldots & \ldots & \ldots \\
a_{1n} & a_{2n} & \ldots & a_{mn}
\end{array}\right]
$$
```

\end{array}\right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

- 1. 矩阵和自身逆矩阵相乘得单位矩阵， $SA A^{-1}=I=A^{-1} A$ ；
- 2.  $SASBS$ 两矩阵相乘的逆，等于逆矩阵 $SBS$ 和逆矩阵 $SAS$ 相乘，这里强调一下乘的顺序很重要， $S(A B)^{-1}=B^{-1} A^{-1}$ ；
- 3.  $SABS$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $S(A+B)^{-1}\neq A^{-1}+B^{-1}$ ；
- 4. 矩阵转置的转置还是它本身， $S\left(A^T\right)^{\mathrm{T}}=A$ ；
- 5.  $SABS$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $S(A+B)^T=A^T+B^T$ ；
- 6.  $SABS$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $B$ 和转置矩阵 $A$ 的相乘，这里再次强调乘的顺序很重要， $S(A B)^T=B^T A^T$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $Sx=\left(x_{1}, \ldots, x_{10}\right)^T$ ， $Sv=\left(v_{1}, \ldots, v_{10}\right)^T$ ，如果要计算 $Sy=x^T\left(I+v v^T\right) x$ ，其中 $S$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

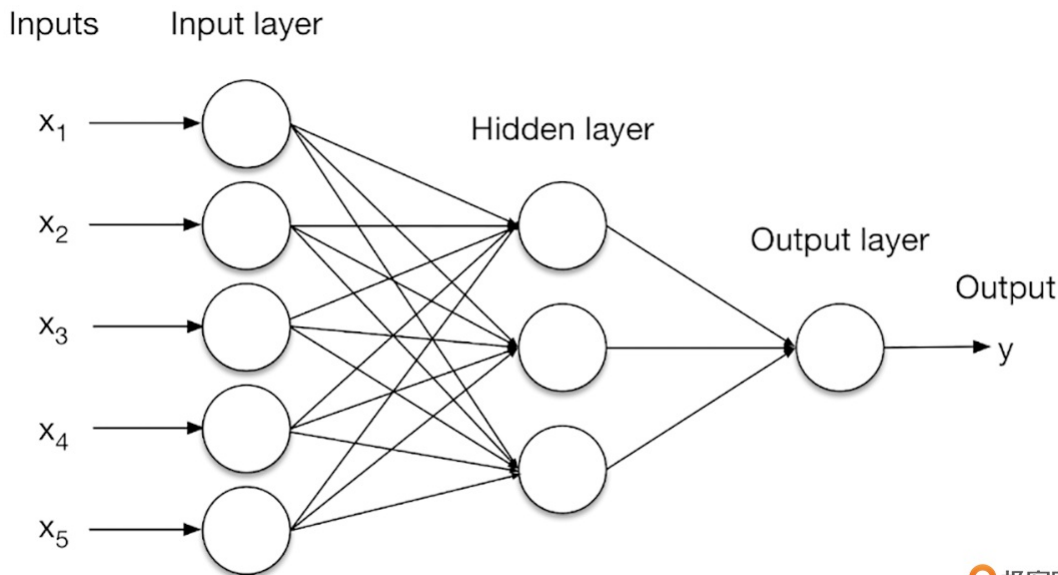
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$AX = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵 $\widetilde{A}$ ， 可以表示为\$(A, B)\$，这里的\$B\$表示的是方程组常数项所构成的列向量，也就是 $m\times 1$ 的 $m$ 行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为 $n\times 1$ 的 $n$ 行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$，就可以表示为\$AX=B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵 $\widetilde{A}$ 的列向量，则线性方程组\$A\$X=\$B\$又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由 $m\times n$ 个元素组成，  $m$ 和 $n$ 是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按\$m\$行 $n$ 列的矩形排布方式后可以形成矩阵\$A\$：

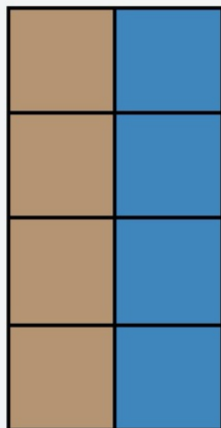
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例，  $(1, n)$ 矩阵叫做行，  $(m, 1)$ 矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设 $R^{m\times n}$ 是实数矩阵\$(m, n)\$的集合，  $A\in R^{m\times n}$ 可以表示成另一种形式  $A\in R^{mm}$ 。 我们把矩阵的 $n$ 列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \cdots & a_{1n}+b_{1n} \\
\cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots \\
a_{m1}+b_{m1} & \cdots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \cdots, m, j=1, \cdots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。

\$\$AB=CS\$

但反过来B和A相乘就不行了，因为相邻阶数 $S_mS$ 不等于 $S_nS$ 。

2.哈达玛积

哈达玛积理解起来就很简单了，就是矩阵各对应元素的乘积， $c_{c_{ij}}=a_{ij} \times b_{ij}$ 。举个例子：

```
$$
C=A^{(*)} B=\left[\begin{array}{ll}
1 & 2 \\
4 & 5
\end{array}\right]\left[\begin{array}{ll}
1 & 4 \\
2 & 5
\end{array}\right]=\left[\begin{array}{cc}
1 * 1 & 2 * 4 \\
4 * 2 & 5 * 5
\end{array}\right]=\left[\begin{array}{cc}
1 & 8 \\
8 & 25
\end{array}\right]
$$
```

哈达玛积其实在数学中不常看到，不过，在编程中哈达玛积非常有用，因为它可以用来同时计算多组数据的乘积，计算效率很高。

3.克罗内克积

克罗内克积是以德国数学家利奥波德·克罗内克（Leopold Kronecker）的名字命名的。它可以应用在解线性矩阵方程和图像处理方面，当然从更时髦的角度说，它还能用在量子信息领域，我们也称之为直积或张量积。

和普通矩阵乘和哈达玛积不同的是，克罗内克积是两个任意大小矩阵间的运算，表示为 $A \times B$ ，如果 $A$ 是一个 $m \times n$ 的矩阵，而 $B$ 是一个 $p \times q$ 的矩阵，克罗内克积则是一个 $mp \times nq$ 的矩阵。

接下来我们需要定义一个在矩阵的乘法中起着特殊作用的矩阵，它就是单位矩阵。高等代数中，在求解相应的矩阵时，若添加单位矩阵，通过初等变换进行求解，往往可以使问题变得简单。按照百度百科的解释，单位矩阵如同数的乘法中的1，这种矩阵就被称为单位矩阵。它是个方阵，从左上角到右下角的对角线，也就是主对角线上的元素均为1，除此以外全都为0。

在线性代数中，大小为 $n$ 的单位矩阵就是在主对角线上均为1，而其他地方都是0的 $n \times n$ 的方阵，它用 $\mathbf{I}_n$ 表示，表达时为了方便可以忽略阶数，直接用 $\mathbf{I}$ 来表示：

```
$$
\mathbf{I}_1=[1], \mathbf{I}_2=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right], \mathbf{I}_3=\left[\begin{array}{lll}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right], \dots, \mathbf{I}_n=\left[\begin{array}{cccc}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
. & . & \dots & . \\
. & . & . & . \\
0 & 0 & \dots & 1
\end{array}\right]
$$
```

矩阵的性质

在了解了矩阵加和乘，以及单位矩阵后，我们是时候来看一看矩阵的性质了。了解矩阵的性质是进行矩阵计算的前提，就像我们小时候学加减乘除四则运算法则时那样。所以，这块内容对你来说应该不难，你作为了解就好，重点是之后的运算。

1.结合律

任意实数 $m \times n$ 矩阵 $A$ ， $n \times p$ 矩阵 $B$ ， $p \times q$ 矩阵 $C$ 之间相乘，满足结合律 $(AB)C=A(BC)$ 。这个很好理解，我就不多说了。

\$\$\forall \text{all } A \in R^{m \times n}, B \in R^{n \times p}, C \in R^{p \times q} \{ (A B) C=A(B C) \$\$

2.分配律

任意实数 $m \times n$ 矩阵 $A$ 和 $B$ ， $n \times p$ 矩阵 $C$ 和 $D$ 之间相乘满足分配律 $(A+B)C=AC+BC$ ， $A(C+D)=AC+AD$ 。

\$\$\forall \text{all } \mathbf{A}, \mathbf{B} \in \mathbf{R}^{m \times n}, \mathbf{C}, \mathbf{D} \in \mathbf{R}^{n \times p} \{ (\mathbf{A}+\mathbf{B}) \mathbf{C}=\mathbf{A} \mathbf{C}+\mathbf{B} \mathbf{C}, \mathbf{A}(\mathbf{C}+\mathbf{D})=\mathbf{A} \mathbf{C}+\mathbf{A} \mathbf{D} \$\$

3.单位矩阵乘

任意实数 $m \times n$ 矩阵A和单位矩阵之间的乘，等于它本身 $A$ 。

\$\$\forall \text{all } A \in R^{m \times n} : \mathbf{I}\_m A=A \mathbf{I}\_n=A \$\$

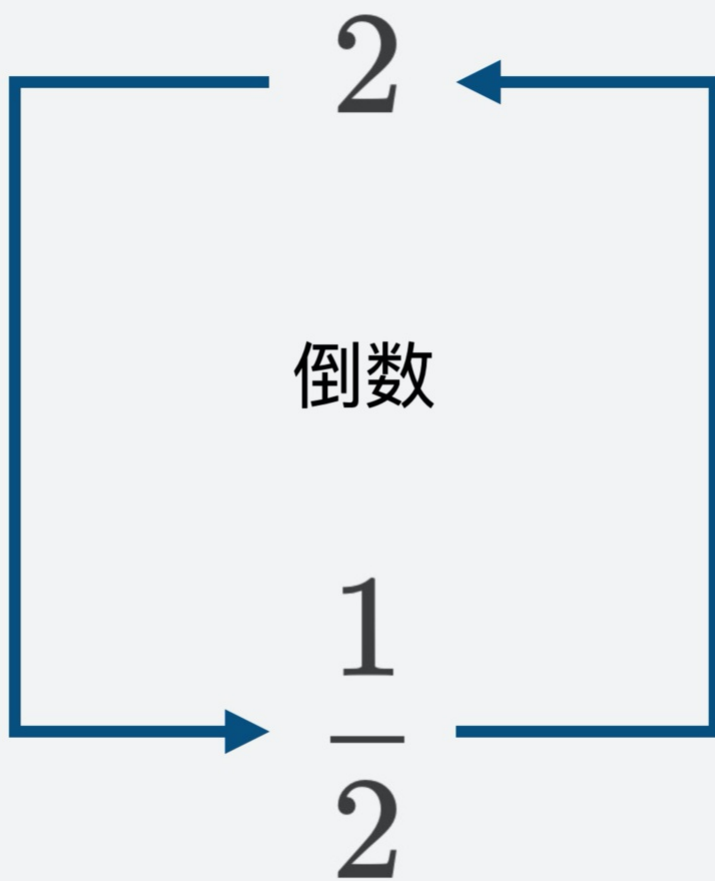
注意，这里的行和列不同， $m \neq n$ 意味着，根据矩阵乘，左乘和右乘单位矩阵也不同，也就是 $\mathbf{I}_m \neq \mathbf{I}_n$ 。

逆矩阵与转置矩阵

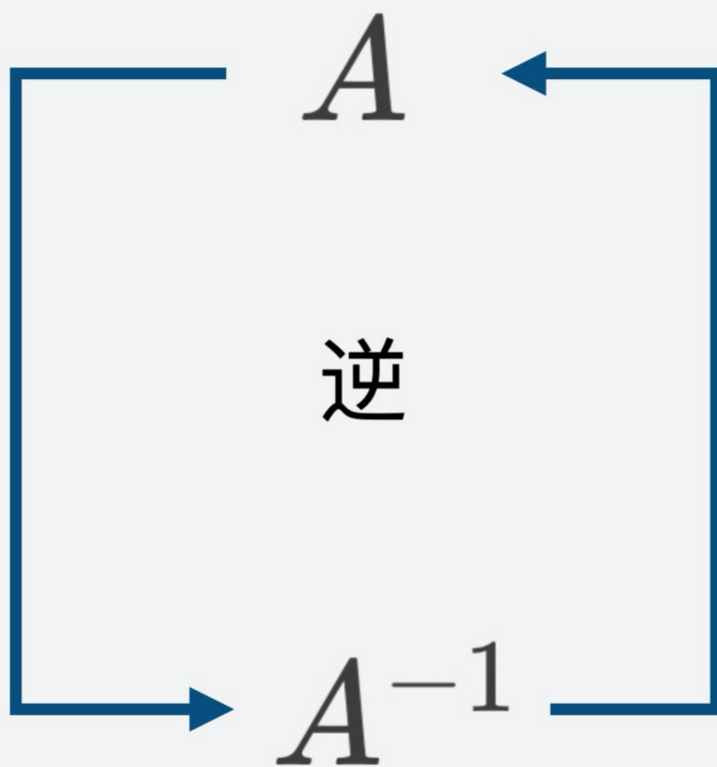
了解矩阵基本概念、运算，以及性质后，我来讲一讲矩阵应用中的两个核心内容——逆矩阵和转置矩阵。逆矩阵和转置矩阵在实际应用中大有用处，比如：坐标系中的图形变换运算。我们先来看下什么是逆矩阵。

逆矩阵

下面这个图你应该非常熟悉了，图中表现的是数字的倒数，2的倒数是 $\frac{1}{2}$ ， $\frac{1}{2}$ 的倒数是2。



其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```

A^{-1} = \frac{1}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}

```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```

A \times A^{-1} = \frac{1}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}
= \frac{1}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}} \begin{bmatrix} a_{22}a_{11} - a_{12}a_{21} & a_{22}a_{12} - a_{12}a_{22} \\ -a_{21}a_{11} + a_{11}a_{21} & -a_{21}a_{12} + a_{11}a_{22} \end{bmatrix}
= \frac{1}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}} \begin{bmatrix} a_{11}a_{22} - a_{12}a_{21} & 0 \\ 0 & a_{11}a_{22} - a_{12}a_{21} \end{bmatrix}
= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}

```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程 $XA=B$ 。

小孩	大人	大巴	火车
$x_1$	$x_2$	3	3.5
		3.2	3.6

要解 $XS$ ，我们就要先计算 $S$ 的逆矩阵 $A^{-1}$ ：

$$\begin{array}{l} A^{-1} = \left\| \begin{array}{cc} 3 & 3.5 \\ 3.2 & 3.6 \end{array} \right\| \\ \text{end{array}} \right\|^{-1} = \frac{1}{\{3\} \times 3.6 - 3.5 \times 3.2} \left\| \begin{array}{cc} 3.6 & -3.5 \\ -3.2 & 3 \end{array} \right\| \\ \text{end{array}} \right\| \\ = \frac{1}{-9 + 8.75} \left\| \begin{array}{cc} 8 & -7.5 \\ -9 & 8.75 \end{array} \right\| \\ \text{end{array}} \right\| \end{array}$$

接下来再计算  $X = B A^{-1} S$ :

$$\begin{aligned} & \left\lfloor \begin{array}{l} x_1 \\ 118.4 \\ -9.8 \\ 8 \end{array} \right\rfloor = \left\lfloor \begin{array}{l} x_2 \\ 135.2 \\ 8.75 \\ -8.75 \end{array} \right\rfloor \\ & \left\lfloor \begin{array}{l} 16 \\ 22 \end{array} \right\rfloor \end{aligned}$$

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序包封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least, 方程次序很重要, 也就是说,  $SAX=BS$  和  $SXA=BS$  的结果是不同的, 这个一定要牢记哦!

## 转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把  $m \times n$  矩阵  $A$  的行列互换，得到转置矩阵  $A^T$ 。

```

SS
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \& a_{1n} \\
a_{21} & a_{22} & \& a_{2n} \\
\& \& \& \& \\
a_{m1} & a_{m2} & \& a_{mn}
\end{array}\right]
SS

```

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

\end{array}'right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

1. 矩阵和自身逆矩阵相乘得单位矩阵， $SA A^{-1}=I=A^{-1} A$ ；
2.  $SASBS$ 两矩阵相乘的逆，等于逆矩阵 $SBS$ 和逆矩阵 $SAS$ 相乘，这里强调一下乘的顺序很重要， $S(A B)^{-1}=B^{-1} A^{-1}$ ；
3.  $SABS$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $S(A+B)^{-1}\neq A^{-1}+B^{-1}$ ；
4. 矩阵转置的转置还是它本身， $S\left(A^T\right)^{\mathrm{T}}=A$ ；
5.  $SABS$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $S(A+B)^T=A^T+B^T$ ；
6.  $SABS$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $B$ 和转置矩阵 $A$ 的相乘，这里再次强调乘的顺序很重要， $S(A B)^T=B^T A^T$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $Sx=\left(x_{1}, \ldots, x_{10}\right)^T$ ， $Sv=\left(v_{1}, \ldots, v_{10}\right)^T$ ，如果要计算 $Sy=x^T\left(I+v v^T\right) x$ ，其中 $S$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

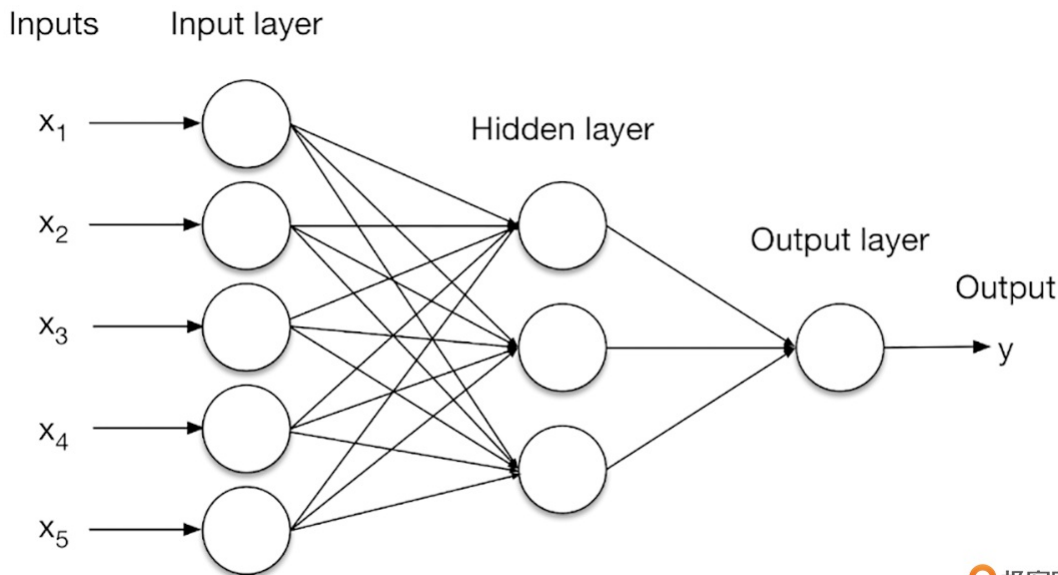
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{41} & w_{42} \\ w_{31} & w_{32} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} \\ w_{41} & w_{42} \\ w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$A \cdot X = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵 $\widetilde{A}$ ， 可以表示为\$(A, B)\$，这里的\$B\$表示的是方程组常数项所构成的列向量，也就是 $m\times 1$ 的 $m$ 行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为 $n\times 1$ 的 $n$ 行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$，就可以表示为\$AX=B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵 $\widetilde{A}$ 的列向量，则线性方程组\$A\$X=\$B\$又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由 $m\times n$ 个元素组成，  $m$ 和 $n$ 是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按 $m$ 行 $n$ 列的矩形排布方式后可以形成矩阵\$A\$：

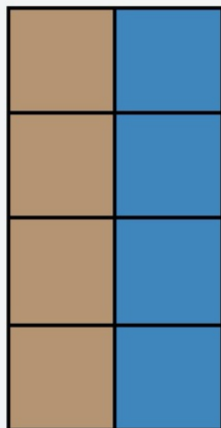
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例，  $(1, n)$ 矩阵叫做行，  $(m, 1)$ 矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设 $R^{m\times n}$ 是实数矩阵\$(m, n)\$的集合，  $A\in R^{m\times n}$ 可以表示成另一种形式  $A\in R^{mm}$ 。 我们把矩阵的 $n$ 列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \cdots & a_{1n}+b_{1n} \\
\cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots \\
a_{m1}+b_{m1} & \cdots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \cdots, m, j=1, \cdots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。

\$\$AB=CS\$

但反过来B和A相乘就不行了，因为相邻阶数 $S_mS$ 不等于 $S_nS$ 。

2.哈达玛积

哈达玛积理解起来就很简单了，就是矩阵各对应元素的乘积， $c_{c_{ij}}=a_{ij} \times b_{ij}$ 。举个例子：

```
$$
C=A^{(*)} B=\left[\begin{array}{ll}
1 & 2 \\
4 & 5
\end{array}\right]\left[\begin{array}{ll}
1 & 4 \\
2 & 5
\end{array}\right]=\left[\begin{array}{cc}
1 * 1 & 2 * 4 \\
4 * 2 & 5 * 5
\end{array}\right]=\left[\begin{array}{cc}
1 & 8 \\
8 & 25
\end{array}\right]
$$
```

哈达玛积其实在数学中不常看到，不过，在编程中哈达玛积非常有用，因为它可以用来同时计算多组数据的乘积，计算效率很高。

3.克罗内克积

克罗内克积是以德国数学家利奥波德·克罗内克（Leopold Kronecker）的名字命名的。它可以应用在解线性矩阵方程和图像处理方面，当然从更时髦的角度说，它还能用在量子信息领域，我们也称之为直积或张量积。

和普通矩阵乘和哈达玛积不同的是，克罗内克积是两个任意大小矩阵间的运算，表示为 $A \times B$ ，如果 $A$ 是一个 $m \times n$ 的矩阵，而 $B$ 是一个 $p \times q$ 的矩阵，克罗内克积则是一个 $mp \times nq$ 的矩阵。

接下来我们需要定义一个在矩阵的乘法中起着特殊作用的矩阵，它就是单位矩阵。高等代数中，在求解相应的矩阵时，若添加单位矩阵，通过初等变换进行求解，往往可以使问题变得简单。按照百度百科的解释，单位矩阵如同数的乘法中的1，这种矩阵就被称为单位矩阵。它是个方阵，从左上角到右下角的对角线，也就是主对角线上的元素均为1，除此以外全都为0。

在线性代数中，大小为 $n$ 的单位矩阵就是在主对角线上均为1，而其他地方都是0的 $n \times n$ 的方阵，它用 $\mathbf{I}_n$ 表示，表达时为了方便可以忽略阶数，直接用 $\mathbf{I}$ 来表示：

```
$$
\mathbf{I}_1=[1], \mathbf{I}_2=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right], \mathbf{I}_3=\left[\begin{array}{lll}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right], \dots, \mathbf{I}_n=\left[\begin{array}{cccc}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
. & . & \dots & . \\
. & . & . & . \\
0 & 0 & \dots & 1
\end{array}\right]
$$
```

矩阵的性质

在了解了矩阵加和乘，以及单位矩阵后，我们是时候来看一看矩阵的性质了。了解矩阵的性质是进行矩阵计算的前提，就像我们小时候学加减乘除四则运算法则时那样。所以，这块内容对你来说应该不难，你作为了解就好，重点是之后的运算。

1.结合律

任意实数 $m \times n$ 矩阵 $A$ ， $n \times p$ 矩阵 $B$ ， $p \times q$ 矩阵 $C$ 之间相乘，满足结合律 $(AB)C=A(BC)$ 。这个很好理解，我就不多说了。

\$\$\forall a \in R^{\{m \times n\}}, B \in R^{\{n \times p\}}, C \in R^{\{p \times q\}} \{ (A B) C = A (B C) \$\$

2.分配律

任意实数 $m \times n$ 矩阵 $A$ 和 $B$ ， $n \times p$ 矩阵 $C$ 和 $D$ 之间相乘满足分配律 $(A+B)C=AC+BC$ ， $A(C+D)=AC+AD$ 。

\$\$\forall a \in \mathbf{M}\{A\}, B \in \mathbf{M}\{R^{\{m \times n\}}\}, C, D \in \mathbf{M}\{R^{\{n \times p\}}\} \{ (A+B) C = A C + B C, A (C+D) = A C + A D \$\$

3.单位矩阵乘

任意实数 $m \times n$ 矩阵A和单位矩阵之间的乘，等于它本身 $A$ 。

\$\$\forall a \in R^{\{m \times n\}}: \mathbf{I}\_m A = A \mathbf{I}\_n = A \$\$

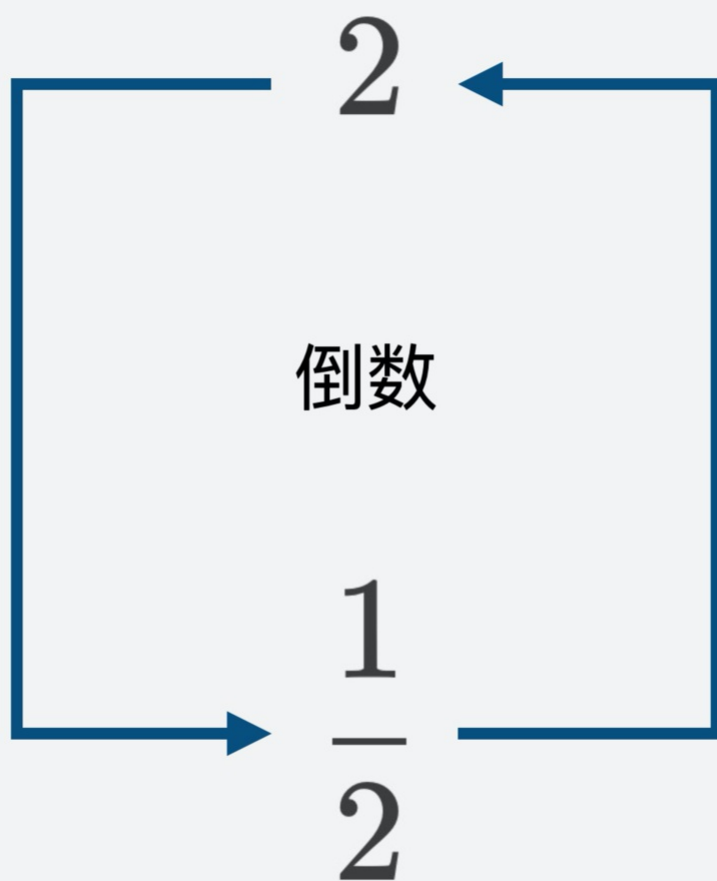
注意，这里的行和列不同， $m \neq n$ 意味着，根据矩阵乘，左乘和右乘单位矩阵也不同，也就是 $\mathbf{I}_m \neq \mathbf{I}_n$ 。

逆矩阵与转置矩阵

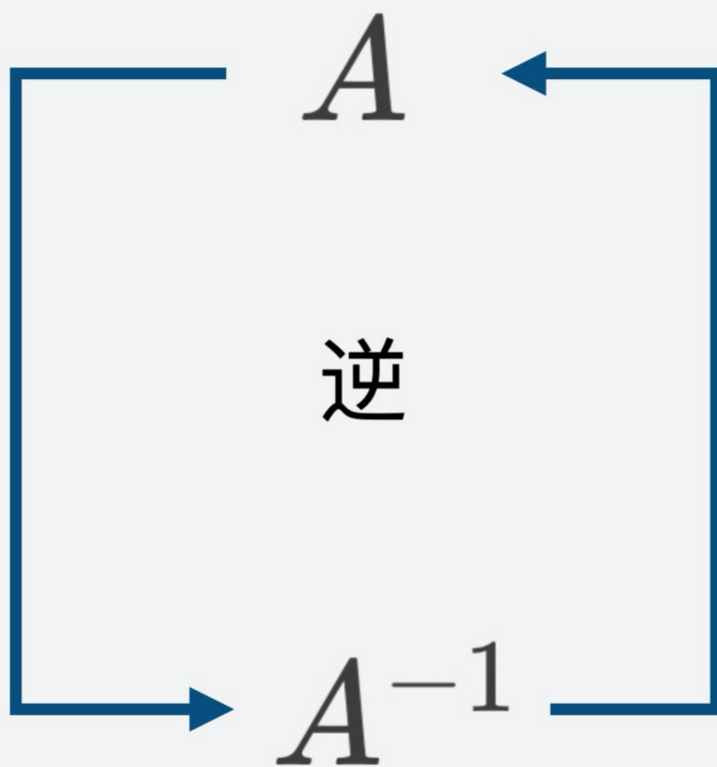
了解矩阵基本概念、运算，以及性质后，我来讲一讲矩阵应用中的两个核心内容——逆矩阵和转置矩阵。逆矩阵和转置矩阵在实际应用中大有用处，比如：坐标系中的图形变换运算。我们先来看下什么是逆矩阵。

逆矩阵

下面这个图你应该非常熟悉了，图中表现的是数字的倒数，2的倒数是 $\frac{1}{2}$ ， $\frac{1}{2}$ 的倒数是2。



其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```
SS
A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\left[\begin{array}{cc}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{array}\right]
SS
```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```
SS
A \times A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right]
SS
```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程 $XA=B$ 。

小孩	大人	大巴	火车
$x_1$	$x_2$	3	3.5
		3.2	3.6

要解 $XS$ ，我们就要先计算 $S$ 的逆矩阵 $A^{-1}$ ：

$$\begin{array}{l} A^{-1} = \left\| \begin{array}{cc} 3 & 3.5 \\ 3.2 & 3.6 \end{array} \right\| \\ \text{end{array}} \right\|^{-1} = \frac{1}{\{3\} \times 3.6 - 3.5 \times 3.2} \left\| \begin{array}{cc} 3.6 & -3.5 \\ -3.2 & 3 \end{array} \right\| \\ \text{end{array}} \right\| \\ = \frac{1}{-9 + 8.75} \left\| \begin{array}{cc} 8 & -7.5 \\ -9 & 8.75 \end{array} \right\| \\ \text{end{array}} \right\| \end{array}$$

接下来再计算  $X = B A^{-1} S$ :

$$\begin{aligned} & \left\lfloor \begin{array}{l} x_1 \\ 118.4 \\ -9.8 \\ 8 \end{array} \right\rfloor = \left\lfloor \begin{array}{l} x_2 \\ 135.2 \\ 8.75 \\ -8.75 \end{array} \right\rfloor \\ & \left\lfloor \begin{array}{l} 16 \\ 22 \end{array} \right\rfloor \end{aligned}$$

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序包封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least, 方程次序很重要, 也就是说,  $SAX=BS$  和  $SXA=BS$  的结果是不同的, 这个一定要牢记哦!

## 转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把  $m \times n$  矩阵  $A$  的行列互换，得到转置矩阵  $A^T$ 。

```

SS
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \& a_{1n} \\
a_{21} & a_{22} & \& a_{2n} \\
\& \& \& \& \\
a_{m1} & a_{m2} & \& a_{mn}
\end{array}\right]
SS

```

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

\end{array}'right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

- 1. 矩阵和自身逆矩阵相乘得单位矩阵， $SA A^{-1}=I=A^{-1} A S$ ;
- 2.  $SASBS$ 两矩阵相乘的逆，等于逆矩阵 $SBS$ 和逆矩阵 $SAS$ 相乘，这里强调一下乘的顺序很重要， $S(A B)^{-1}=B^{-1} A^{-1} S$ ;
- 3.  $SABS$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $S(A+B)^{-1}\neq A^{-1}+B^{-1} S$ ;
- 4. 矩阵转置的转置还是它本身， $S\left(A^T\right)^{\mathrm{T}}=A S$ ;
- 5.  $SABS$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $S(A+B)^T=A^T+B^T S$ ;
- 6.  $SABS$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $B$ 和转置矩阵 $A$ 的相乘，这里再次强调乘的顺序很重要， $S(A B)^T=B^T A^T S$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $Sx=\left(x_{1}, \ldots, x_{10}\right)^T S, S=\left(v_{1}, \ldots, v_{10}\right)^T T S$ ，如果要计算 $Sy=x x^T \left(I+v v^T\right) x S$ ，其中 $S$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

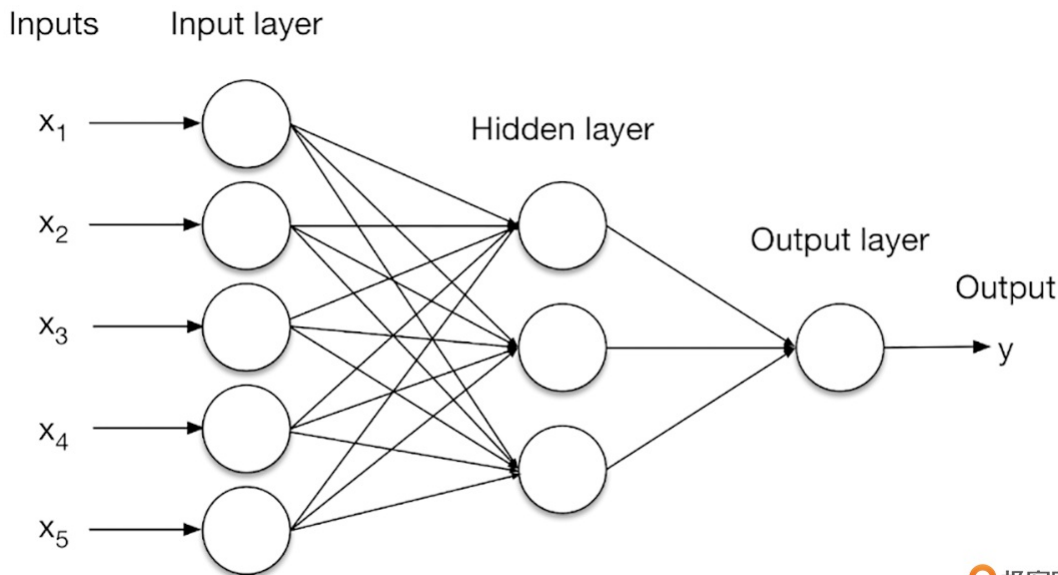
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$AX = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵\$\widetilde{A}\$， 可以表示为\$(A, B)\$， 这里的\$B\$表示的是方程组常数项所构成的列向量， 也就是\$m\times 1\$的\$m\$行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为\$n\times 1\$的\$n\$行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$， 就可以表示为\$AX=B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵\$\widetilde{A}\$的列向量， 则线性方程组\$A\$X=\$B\$又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、 图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由\$m\times n\$个元素组成， \$m\$和\$n\$是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按\$m\$行\$n\$列的矩形排布方式后可以形成矩阵\$A\$：

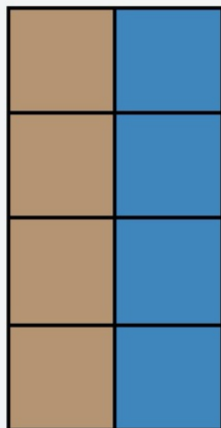
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例， \$(1, n)\$矩阵叫做行， \$(m, 1)\$矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。 矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设\$R^{m\times n}\$是实数矩阵\$(m, n)\$的集合， \$A\in R^{m\times n}\$可以表示成另一种形式 \$A\in R^{mm}\$。 我们把矩阵的\$n\$列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、 列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \dots & a_{1n}+b_{1n} \\
\vdots & \ddots & \vdots \\
\vdots & \ddots & \vdots \\
a_{m1}+b_{m1} & \dots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \dots, m, j=1, \dots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

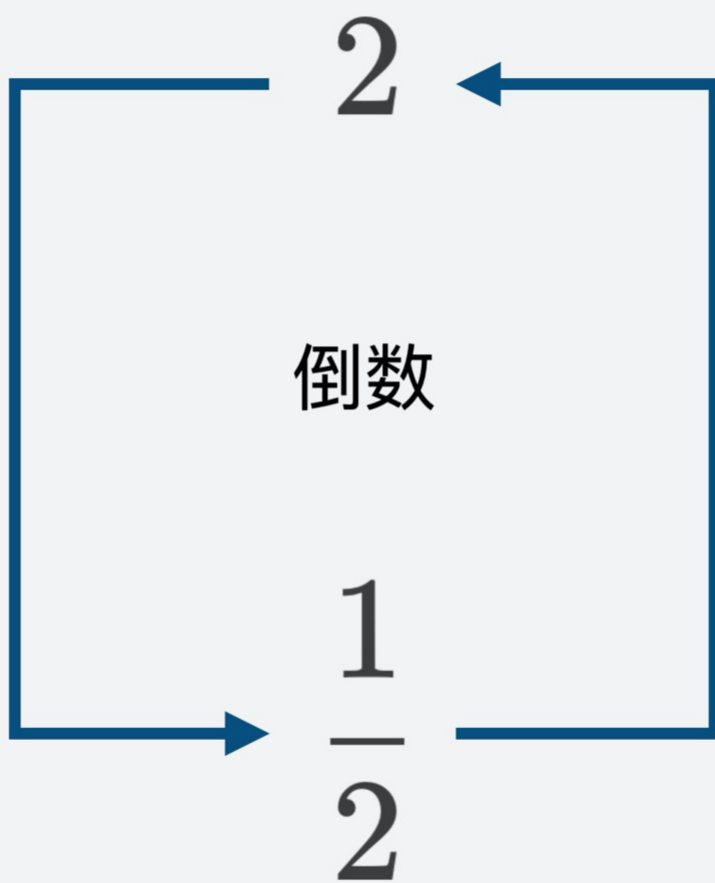
```

$$
C=A B=\left[\begin{array}{lll}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ll}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{lll}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

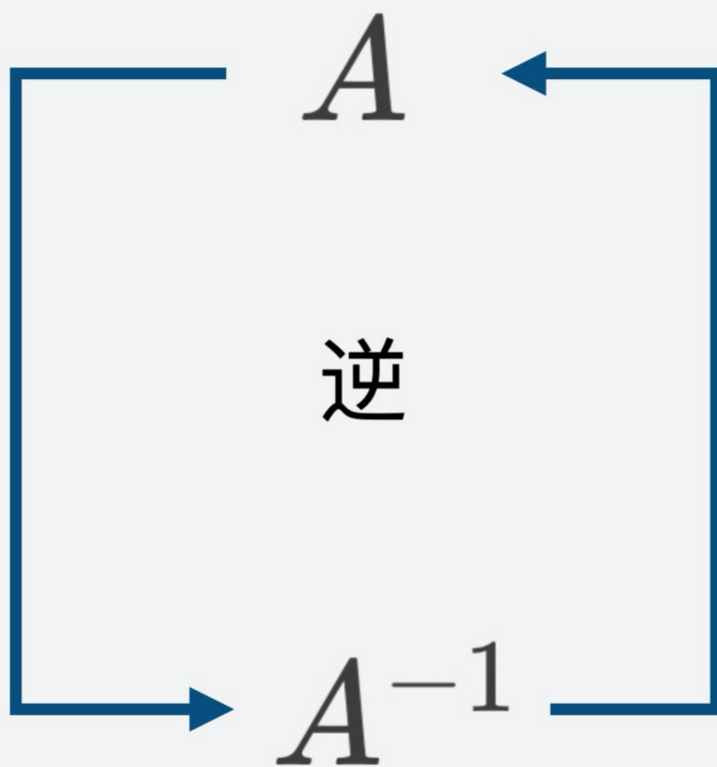
```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。





其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```
SS
A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\left[\begin{array}{cc}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{array}\right]
SS
```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```
SS
A \times A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right]
SS
```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程\$XA=BS\$。

小孩          大人          大巴          火车

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 3 & 3.5 \\ 3.2 & 3.6 \end{bmatrix}$$

要解\$XS\$，我们就要先计算\$A\$的逆矩阵\$A^{-1}\$：

```
$$
A^{-1}=\left[\begin{array}{cc}
3 & 3.5 \\
3.2 & 3.6
\end{array}\right]^{-1}=\frac{1}{3\times 3.6-3.5\times 3.2}\left[\begin{array}{cc}
3.6 & -3.5 \\
-3.2 & 3
\end{array}\right]
=\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
$$
```

接下来再计算\$X=BA^{-1}\$：

```
$$
\left[\begin{array}{ll}
x_1 & x_2
\end{array}\right]=\left[\begin{array}{ll}
118.4 & 135.2
\end{array}\right]\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
=\left[\begin{array}{ll}
16 & 22
\end{array}\right]
$$
```

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序包封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least，方程次序很重要，也就是说，\$AX=BS\$和\$XA=BS\$的结果是不同的，这个一定要牢记哦！

转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把\$m\times n\$矩阵\$A\$的行列互换，得到转置矩阵\$A^T\$。

```
$$
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\ldots & \ldots & \ldots & \ldots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{array}\right]
$$
```

```
$$
A^T=\left[\begin{array}{cccc}
a_{11} & a_{21} & \ldots & a_{m1} \\
a_{12} & a_{22} & \ldots & a_{m2} \\
\ldots & \ldots & \ldots & \ldots \\
a_{1n} & a_{2n} & \ldots & a_{mn}
\end{array}\right]

```

\end{array}'\right]
SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

1. 矩阵和自身逆矩阵相乘得单位矩阵， $SA A^{-1}=I=A^{-1} A S$ ;
2.  $SASBS$ 两矩阵相乘的逆，等于逆矩阵 $SBS$ 和逆矩阵 $SAS$ 相乘，这里强调一下乘的顺序很重要， $S(A B)^{-1}=B^{-1} A^{-1} S$ ;
3.  $SABS$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $S(A+B)^{-1}\neq A^{-1}+B^{-1} S$ ;
4. 矩阵转置的转置还是它本身， $S\left(A^T\right)^{\mathrm{T}}=A S$ ;
5.  $SABS$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $S(A+B)^T=A^T+B^T S$ ;
6.  $SABS$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $B$ 和转置矩阵 $A$ 的相乘，这里再次强调乘的顺序很重要， $S(A B)^T=B^T A^T S$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $Sx=\left(x_{1}, \ldots, x_{10}\right)^T S, S v=\left(v_{1}, \ldots, v_{10}\right)^T S$ ，如果要计算 $S y=x x^T \left(I+v v^T\right) x S$ ，其中 $S$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

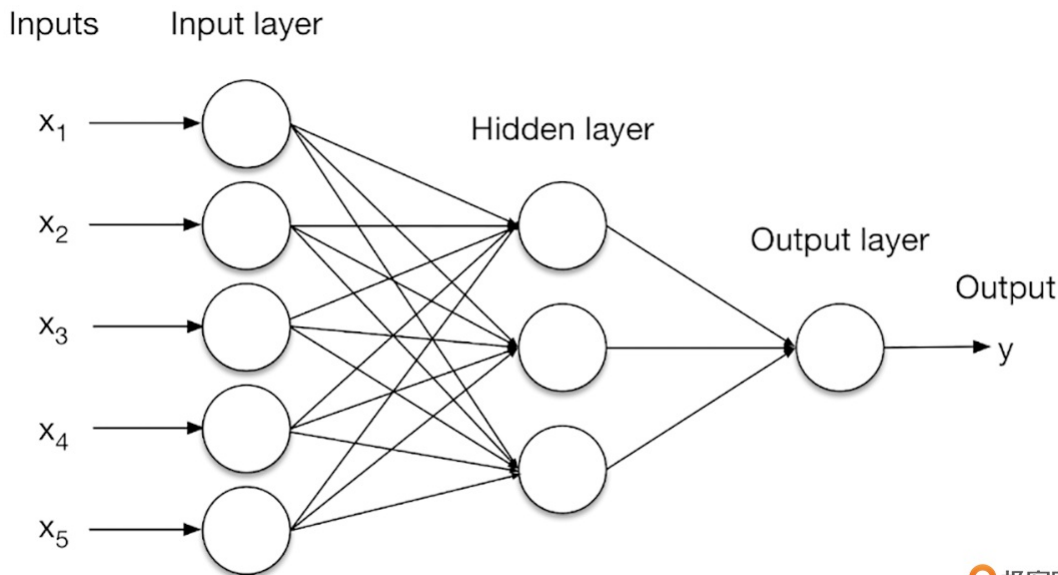
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$AX = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵\$\widetilde{A}\$， 可以表示为\$(A, B)\$， 这里的\$B\$表示的是方程组常数项所构成的列向量， 也就是\$m\times 1\$的\$m\$行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为\$n\times 1\$的\$n\$行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$， 就可以表示为\$A\$X=\$B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵\$\widetilde{A}\$的列向量， 则线性方程组\$A\$X又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、 图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由\$m\times n\$个元素组成， \$m\$和\$n\$是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按\$m\$行\$n\$列的矩形排布方式后可以形成矩阵\$A\$：

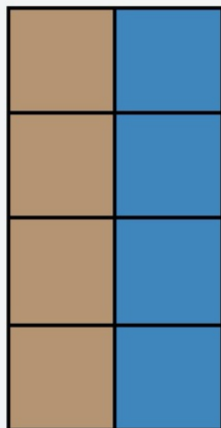
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例， \$(1, n)\$矩阵叫做行， \$(m, 1)\$矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。 矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设\$R^{m\times n}\$是实数矩阵\$(m, n)\$的集合， \$A\in R^{m\times n}\$可以表示成另一种形式 \$A\in R^{mm}\$。 我们把矩阵的\$n\$列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、 列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \cdots & a_{1n}+b_{1n} \\
\cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots \\
a_{m1}+b_{m1} & \cdots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \cdots, m, j=1, \cdots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。

\$\$AB=CS\$

但反过来B和A相乘就不行了，因为相邻阶数 $S_mS$ 不等于 $S_nS$ 。

2.哈达玛积

哈达玛积理解起来就很简单了，就是矩阵各对应元素的乘积， $c_{c_{ij}}=a_{ij} \times b_{ij}$ 。举个例子：

```
$$
C=A^{(*)} B=\left[\begin{array}{ll}
1 & 2 \\
4 & 5
\end{array}\right]\left[\begin{array}{ll}
1 & 4 \\
2 & 5
\end{array}\right]=\left[\begin{array}{cc}
1 * 1 & 2 * 4 \\
4 * 2 & 5 * 5
\end{array}\right]=\left[\begin{array}{cc}
1 & 8 \\
8 & 25
\end{array}\right]
$$
```

哈达玛积其实在数学中不常看到，不过，在编程中哈达玛积非常有用，因为它可以用来同时计算多组数据的乘积，计算效率很高。

3.克罗内克积

克罗内克积是以德国数学家利奥波德·克罗内克（Leopold Kronecker）的名字命名的。它可以应用在解线性矩阵方程和图像处理方面，当然从更时髦的角度说，它还能用在量子信息领域，我们也称之为直积或张量积。

和普通矩阵乘和哈达玛积不同的是，克罗内克积是两个任意大小矩阵间的运算，表示为 $A \times B$ ，如果 $A$ 是一个 $m \times n$ 的矩阵，而 $B$ 是一个 $p \times q$ 的矩阵，克罗内克积则是一个 $mp \times nq$ 的矩阵。

接下来我们需要定义一个在矩阵的乘法中起着特殊作用的矩阵，它就是单位矩阵。高等代数中，在求解相应的矩阵时，若添加单位矩阵，通过初等变换进行求解，往往可以使问题变得简单。按照百度百科的解释，单位矩阵如同数的乘法中的1，这种矩阵就被称为单位矩阵。它是个方阵，从左上角到右下角的对角线，也就是主对角线上的元素均为1，除此以外全都为0。

在线性代数中，大小为 $n$ 的单位矩阵就是在主对角线上均为1，而其他地方都是0的 $n \times n$ 的方阵，它用 $\mathbf{I}_n$ 表示，表达时为了方便可以忽略阶数，直接用 $\mathbf{I}$ 来表示：

```
$$
\mathbf{I}_1=[1], \mathbf{I}_2=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right], \mathbf{I}_3=\left[\begin{array}{lll}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right], \dots, \mathbf{I}_n=\left[\begin{array}{cccc}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
. & . & \dots & . \\
. & . & . & . \\
0 & 0 & \dots & 1
\end{array}\right]
$$
```

矩阵的性质

在了解了矩阵加和乘，以及单位矩阵后，我们是时候来看一看矩阵的性质了。了解矩阵的性质是进行矩阵计算的前提，就像我们小时候学加减乘除四则运算法则时那样。所以，这块内容对你来说应该不难，你作为了解就好，重点是之后的运算。

1.结合律

任意实数 $m \times n$ 矩阵 $A$ ， $n \times p$ 矩阵 $B$ ， $p \times q$ 矩阵 $C$ 之间相乘，满足结合律 $(AB)C=A(BC)$ 。这个很好理解，我就不多说了。

\$\$\forall \text{all } A \in R^{m \times n}, B \in R^{n \times p}, C \in R^{p \times q} \{ (A B) C=A(B C) \$\$

2.分配律

任意实数 $m \times n$ 矩阵 $A$ 和 $B$ ， $n \times p$ 矩阵 $C$ 和 $D$ 之间相乘满足分配律 $(A+B)C=AC+BC$ ， $A(C+D)=AC+AD$ 。

\$\$\forall \text{all } \mathbf{A}, \mathbf{B} \in \mathbf{R}^{m \times n}, \mathbf{C}, \mathbf{D} \in \mathbf{R}^{n \times p} \{ (\mathbf{A}+\mathbf{B}) \mathbf{C}=\mathbf{A} \mathbf{C}+\mathbf{B} \mathbf{C}, \mathbf{A}(\mathbf{C}+\mathbf{D})=\mathbf{A} \mathbf{C}+\mathbf{A} \mathbf{D} \$\$

3.单位矩阵乘

任意实数 $m \times n$ 矩阵A和单位矩阵之间的乘，等于它本身 $A$ 。

\$\$\forall \text{all } A \in R^{m \times n} : \mathbf{I}\_m A=A \mathbf{I}\_n=A \$\$

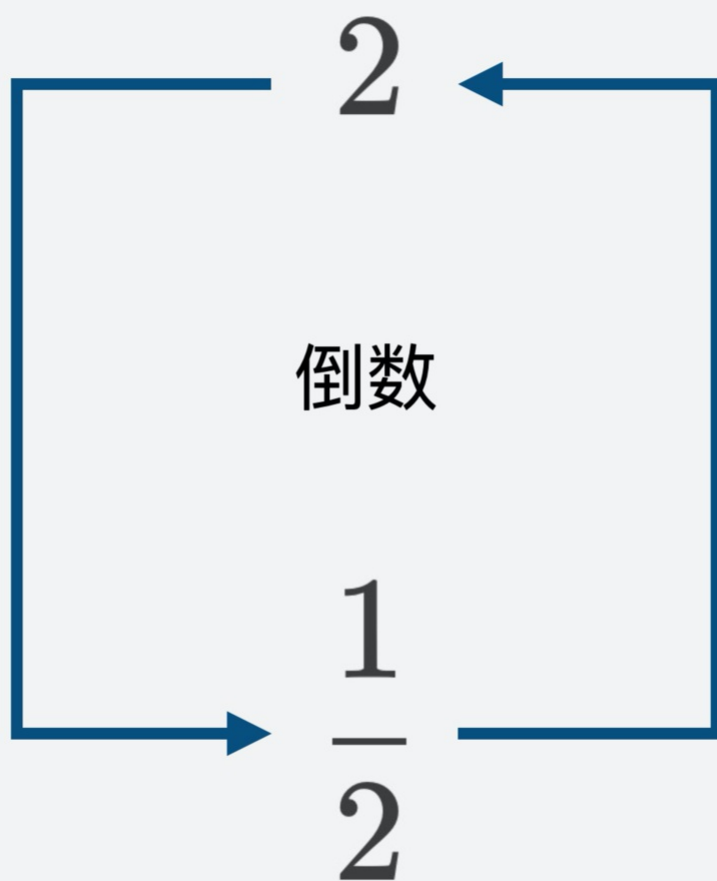
注意，这里的行和列不同， $m \neq n$ 意味着，根据矩阵乘，左乘和右乘单位矩阵也不同，也就是 $\mathbf{I}_m \neq \mathbf{I}_n$ 。

逆矩阵与转置矩阵

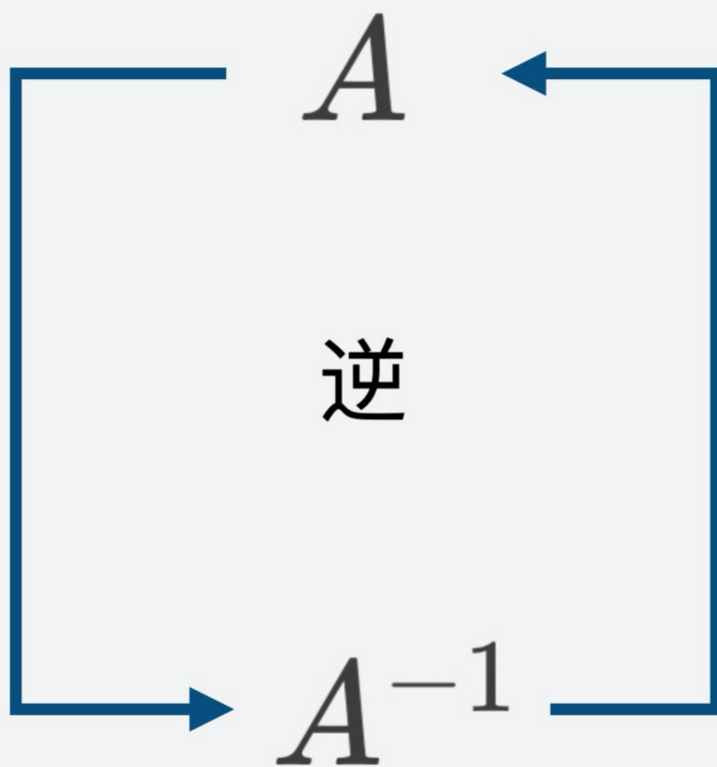
了解矩阵基本概念、运算，以及性质后，我来讲一讲矩阵应用中的两个核心内容——逆矩阵和转置矩阵。逆矩阵和转置矩阵在实际应用中大有用处，比如：坐标系中的图形变换运算。我们先来看下什么是逆矩阵。

逆矩阵

下面这个图你应该非常熟悉了，图中表现的是数字的倒数，2的倒数是 $\frac{1}{2}$ ， $\frac{1}{2}$ 的倒数是2。



其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```
SS
A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\frac{1}{a_{11} a_{22}-a_{12} a_{21}}\left[\begin{array}{cc}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{array}\right]
SS
```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```
SS
A \times A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11} a_{22}-a_{12} a_{21}} & \frac{-a_{12}}{a_{11} a_{22}-a_{12} a_{21}} \\
\frac{-a_{21}}{a_{11} a_{22}-a_{12} a_{21}} & \frac{a_{11}}{a_{11} a_{22}-a_{12} a_{21}}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11} a_{22}-a_{12} a_{21}} & \frac{-a_{12}}{a_{11} a_{22}-a_{12} a_{21}} \\
\frac{-a_{21}}{a_{11} a_{22}-a_{12} a_{21}} & \frac{a_{11}}{a_{11} a_{22}-a_{12} a_{21}}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right]
SS
```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程\$XA=BS\$。

小孩          大人          大巴          火车

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 3 & 3.5 \\ 3.2 & 3.6 \end{bmatrix}$$

要解\$XS\$，我们就要先计算\$A\$的逆矩阵\$A^{-1}\$：

```
$$
A^{-1}=\left[\begin{array}{cc}
3 & 3.5 \\
3.2 & 3.6
\end{array}\right]^{-1}=\frac{1}{3\times 3.6-3.5\times 3.2}\left[\begin{array}{cc}
3.6 & -3.5 \\
-3.2 & 3
\end{array}\right]
=\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
$$
```

接下来再计算\$X=B A^{-1}\$：

```
$$
\left[\begin{array}{l}
x_1 \\
x_2
\end{array}\right]=\left[\begin{array}{l}
118.4 \\
16
\end{array}\right]\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
=\left[\begin{array}{l}
16 \\
22
\end{array}\right]
$$
```

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序包封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least，方程次序很重要，也就是说，\$AX=BS\$和\$XA=BS\$的结果是不同的，这个一定要牢记哦！

转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把\$m\times n\$矩阵\$A\$的行列互换，得到转置矩阵\$A^T\$。

```
$$
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\ldots & \ldots & \ldots & \ldots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{array}\right]
$$
```

```
$$
A^T=\left[\begin{array}{cccc}
a_{11} & a_{21} & \ldots & a_{m1} \\
a_{12} & a_{22} & \ldots & a_{m2} \\
\ldots & \ldots & \ldots & \ldots \\
a_{1n} & a_{2n} & \ldots & a_{mn}
\end{array}\right]

```

\end{array}'right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

- 1. 矩阵和自身逆矩阵相乘得单位矩阵， $SA A^{-1}=I=A^{-1} A$ ；
- 2.  $SASBS$ 两矩阵相乘的逆，等于逆矩阵 $SBS$ 和逆矩阵 $SAS$ 相乘，这里强调一下乘的顺序很重要， $S(A B)^{-1}=B^{-1} A^{-1}$ ；
- 3.  $SABS$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $S(A+B)^{-1}\neq A^{-1}+B^{-1}$ ；
- 4. 矩阵转置的转置还是它本身， $S\left(A^T\right)^{\mathrm{T}}=A$ ；
- 5.  $SABS$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $S(A+B)^T=A^T+B^T$ ；
- 6.  $SABS$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $B$ 和转置矩阵 $A$ 的相乘，这里再次强调乘的顺序很重要， $S(A B)^T=B^T A^T$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $S=\left(x_1, \ldots, x_{10}\right)^T$ ， $S_v=\left(v_1, \ldots, v_{10}\right)^T$ ，如果要计算 $S_y=x^T\left(I+v v^T\right) x$ ，其中 $S$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

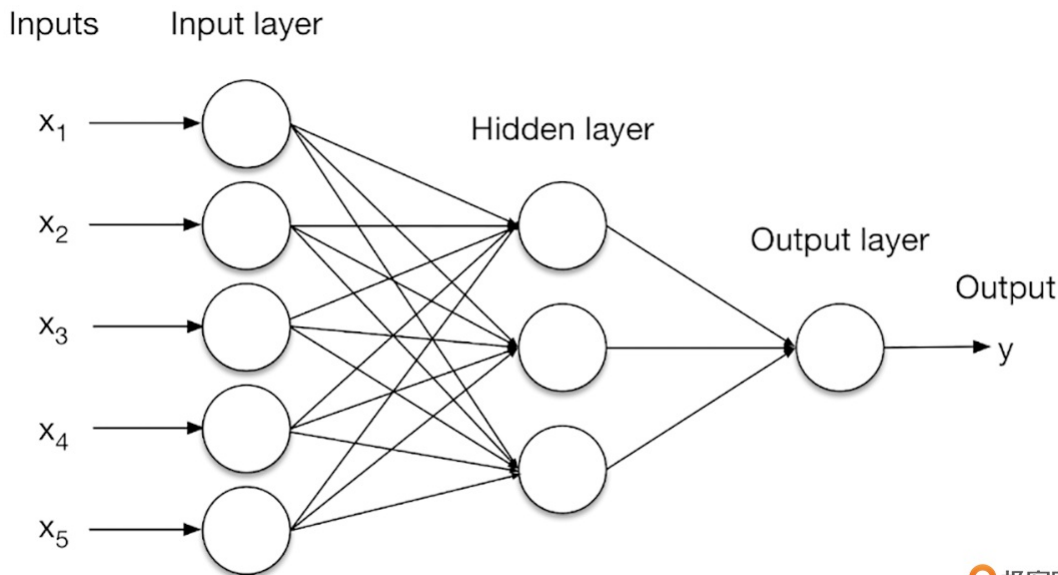
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \\ w_{51} & w_{52} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \\ w_{51} & w_{52} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + b \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$A \cdot X = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵\$\widetilde{A}\$， 可以表示为\$(A, B)\$，这里的\$B\$表示的是方程组常数项所构成的列向量，也就是\$m\times 1\$的\$m\$行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为\$n\times 1\$的\$n\$行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$，就可以表示为\$A\$X=\$B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵\$\widetilde{A}\$的列向量，则线性方程组\$A\$X又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由\$m\times n\$个元素组成， \$m\$和\$n\$是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按\$m\$行\$n\$列的矩形排布方式后可以形成矩阵\$A\$：

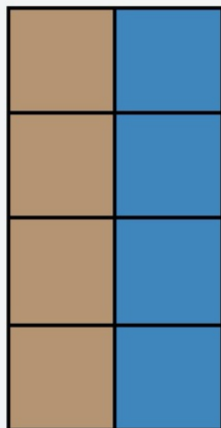
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例， \$(1, n)\$矩阵叫做行， \$(m, 1)\$矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设\$R^{m\times n}\$是实数矩阵\$(m, n)\$的集合， \$A\in R^{m\times n}\$可以表示成另一种形式 \$A\in R^{mm}\$。 我们把矩阵的\$n\$列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \dots & a_{1n}+b_{1n} \\
\vdots & \ddots & \vdots \\
\vdots & \ddots & \vdots \\
a_{m1}+b_{m1} & \dots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \dots, m, j=1, \dots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

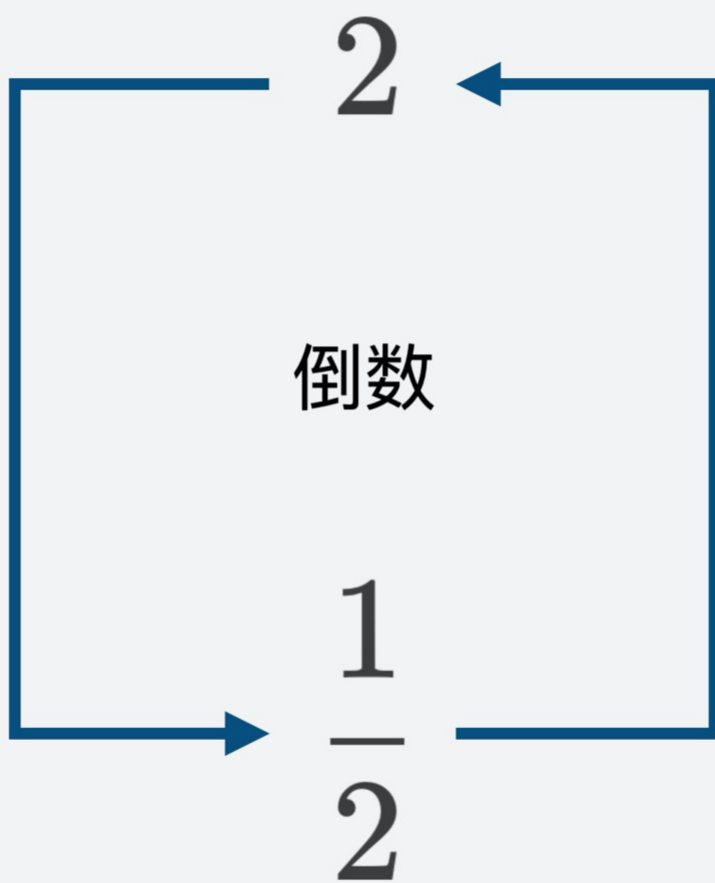
```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。





其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程\$XA=BS\$。

小孩          大人          大巴          火车

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 3 & 3.5 \\ 3.2 & 3.6 \end{bmatrix}$$

要解\$XS\$，我们就要先计算\$A\$的逆矩阵\$A^{-1}\$：

```
$$
A^{-1}=\left[\begin{array}{cc}
3 & 3.5 \\
3.2 & 3.6
\end{array}\right]^{-1}=\frac{1}{3\times 3.6-3.5\times 3.2}\left[\begin{array}{cc}
3.6 & -3.5 \\
-3.2 & 3
\end{array}\right]
=\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
$$
```

接下来再计算\$X=BA^{-1}\$：

```
$$
\left[\begin{array}{ll}
x_1 & x_2
\end{array}\right]=\left[\begin{array}{ll}
118.4 & 135.2
\end{array}\right]\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
=\left[\begin{array}{ll}
16 & 22
\end{array}\right]
$$
```

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序包封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least，方程次序很重要，也就是说，\$AX=BS\$和\$XA=BS\$的结果是不同的，这个一定要牢记哦！

转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把\$m\times n\$矩阵\$A\$的行列互换，得到转置矩阵\$A^T\$。

```
$$
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\ldots & \ldots & \ldots & \ldots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{array}\right]
$$
```

```
$$
A^T=\left[\begin{array}{cccc}
a_{11} & a_{21} & \ldots & a_{m1} \\
a_{12} & a_{22} & \ldots & a_{m2} \\
\ldots & \ldots & \ldots & \ldots \\
a_{1n} & a_{2n} & \ldots & a_{mn}
\end{array}\right]

```

\end{array}\right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

- 1. 矩阵和自身逆矩阵相乘得单位矩阵， $SA A^{-1}=I=A^{-1} A$ ；
- 2.  $SASBS$ 两矩阵相乘的逆，等于逆矩阵 $SBS$ 和逆矩阵 $SAS$ 相乘，这里强调一下乘的顺序很重要， $S(A B)^{-1}=B^{-1} A^{-1}$ ；
- 3.  $SABS$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $S(A+B)^{-1}\neq A^{-1}+B^{-1}$ ；
- 4. 矩阵转置的转置还是它本身， $S\left(A^T\right)^{\mathrm{T}}=A$ ；
- 5.  $SABS$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $S(A+B)^T=A^T+B^T$ ；
- 6.  $SABS$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $B$ 和转置矩阵 $A$ 的相乘，这里再次强调乘的顺序很重要， $S(A B)^T=B^T A^T$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $Sx=\left(x_{1}, \ldots, x_{10}\right)^T$ ， $Sv=\left(v_{1}, \ldots, v_{10}\right)^T$ ，如果要计算 $Sy=x^T\left(I+v v^T\right) x$ ，其中 $S$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

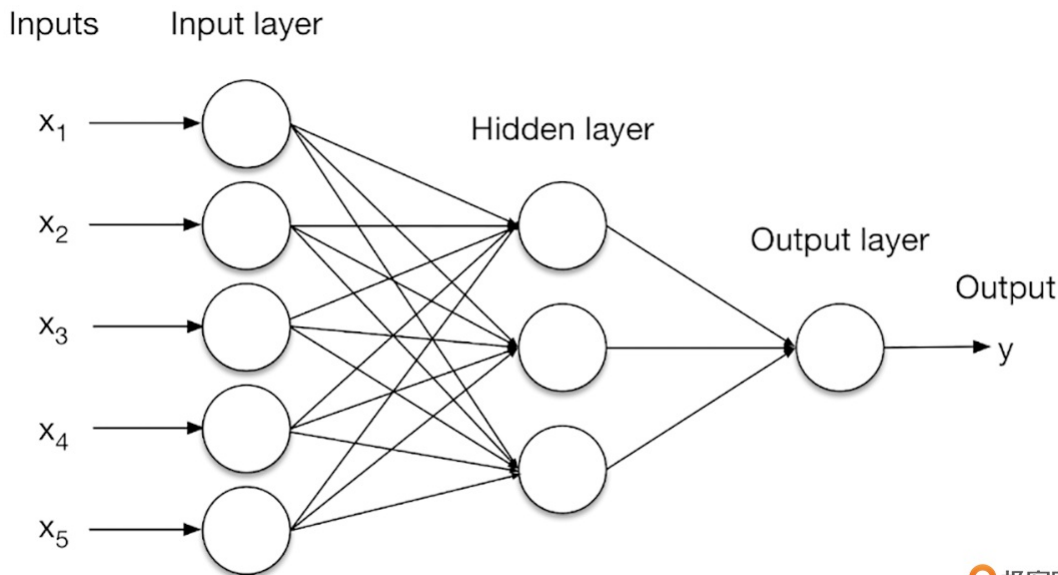
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$AX = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵\$\widetilde{A}\$， 可以表示为\$(A, B)\$， 这里的\$B\$表示的是方程组常数项所构成的列向量， 也就是\$m\times 1\$的\$m\$行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为\$n\times 1\$的\$n\$行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$， 就可以表示为\$AX=B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵\$\widetilde{A}\$的列向量， 则线性方程组\$A\$X=\$B\$又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、 图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由\$m\times n\$个元素组成， \$m\$和\$n\$是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按\$m\$行\$n\$列的矩形排布方式后可以形成矩阵\$A\$：

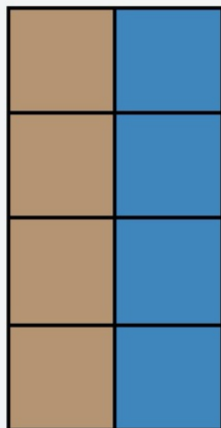
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例， \$(1, n)\$矩阵叫做行， \$(m, 1)\$矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。 矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设\$R^{m\times n}\$是实数矩阵\$(m, n)\$的集合， \$A\in R^{m\times n}\$可以表示成另一种形式 \$A\in R^{mm}\$。 我们把矩阵的\$n\$列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、 列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \cdots & a_{1n}+b_{1n} \\
\cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots \\
a_{m1}+b_{m1} & \cdots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \cdots, m, j=1, \cdots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。

\$\$AB=CS\$

但反过来B和A相乘就不行了，因为相邻阶数 $S_mS$ 不等于 $S_nS$ 。

2.哈达玛积

哈达玛积理解起来就很简单了，就是矩阵各对应元素的乘积， $c_{c_{ij}}=a_{ij} \times b_{ij}$ 。举个例子：

```
$$
C=A^{(*)} B=\left[\begin{array}{ll}
1 & 2 \\
4 & 5
\end{array}\right]\left[\begin{array}{ll}
1 & 4 \\
2 & 5
\end{array}\right]=\left[\begin{array}{cc}
1 * 1 & 2 * 4 \\
4 * 2 & 5 * 5
\end{array}\right]=\left[\begin{array}{cc}
1 & 8 \\
8 & 25
\end{array}\right]
$$
```

哈达玛积其实在数学中不常看到，不过，在编程中哈达玛积非常有用，因为它可以用来同时计算多组数据的乘积，计算效率很高。

3.克罗内克积

克罗内克积是以德国数学家利奥波德·克罗内克（Leopold Kronecker）的名字命名的。它可以应用在解线性矩阵方程和图像处理方面，当然从更时髦的角度说，它还能用在量子信息领域，我们也称之为直积或张量积。

和普通矩阵乘和哈达玛积不同的是，克罗内克积是两个任意大小矩阵间的运算，表示为 $A \times B$ ，如果 $A$ 是一个 $m \times n$ 的矩阵，而 $B$ 是一个 $p \times q$ 的矩阵，克罗内克积则是一个 $mp \times nq$ 的矩阵。

接下来我们需要定义一个在矩阵的乘法中起着特殊作用的矩阵，它就是单位矩阵。高等代数中，在求解相应的矩阵时，若添加单位矩阵，通过初等变换进行求解，往往可以使问题变得简单。按照百度百科的解释，单位矩阵如同数的乘法中的1，这种矩阵就被称为单位矩阵。它是个方阵，从左上角到右下角的对角线，也就是主对角线上的元素均为1，除此以外全都为0。

在线性代数中，大小为 $n$ 的单位矩阵就是在主对角线上均为1，而其他地方都是0的 $n \times n$ 的方阵，它用 $\mathbf{I}_n$ 表示，表达时为了方便可以忽略阶数，直接用 $\mathbf{I}$ 来表示：

```
$$
\mathbf{I}_1=[1], \mathbf{I}_2=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right], \mathbf{I}_3=\left[\begin{array}{lll}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right], \dots, \mathbf{I}_n=\left[\begin{array}{cccc}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
. & . & \dots & . \\
. & . & . & . \\
0 & 0 & \dots & 1
\end{array}\right]
$$
```

矩阵的性质

在了解了矩阵加和乘，以及单位矩阵后，我们是时候来看一看矩阵的性质了。了解矩阵的性质是进行矩阵计算的前提，就像我们小时候学加减乘除四则运算法则时那样。所以，这块内容对你来说应该不难，你作为了解就好，重点是之后的运算。

1.结合律

任意实数 $m \times n$ 矩阵 $A$ ， $n \times p$ 矩阵 $B$ ， $p \times q$ 矩阵 $C$ 之间相乘，满足结合律 $(AB)C=A(BC)$ 。这个很好理解，我就不多说了。

\$\$\forall \text{all } A \in R^{m \times n}, B \in R^{n \times p}, C \in R^{p \times q} \{ (A B) C=A(B C) \$\$

2.分配律

任意实数 $m \times n$ 矩阵 $A$ 和 $B$ ， $n \times p$ 矩阵 $C$ 和 $D$ 之间相乘满足分配律 $(A+B)C=AC+BC$ ， $A(C+D)=AC+AD$ 。

\$\$\forall \text{all } \mathbf{A}, \mathbf{B} \in \mathbf{R}^{m \times n}, \mathbf{C}, \mathbf{D} \in \mathbf{R}^{n \times p} \{ (\mathbf{A}+\mathbf{B}) \mathbf{C}=\mathbf{A} \mathbf{C}+\mathbf{B} \mathbf{C}, \mathbf{A}(\mathbf{C}+\mathbf{D})=\mathbf{A} \mathbf{C}+\mathbf{A} \mathbf{D} \$\$

3.单位矩阵乘

任意实数 $m \times n$ 矩阵A和单位矩阵之间的乘，等于它本身 $A$ 。

\$\$\forall \text{all } A \in R^{m \times n} : \mathbf{I}\_m A=A \mathbf{I}\_n=A \$\$

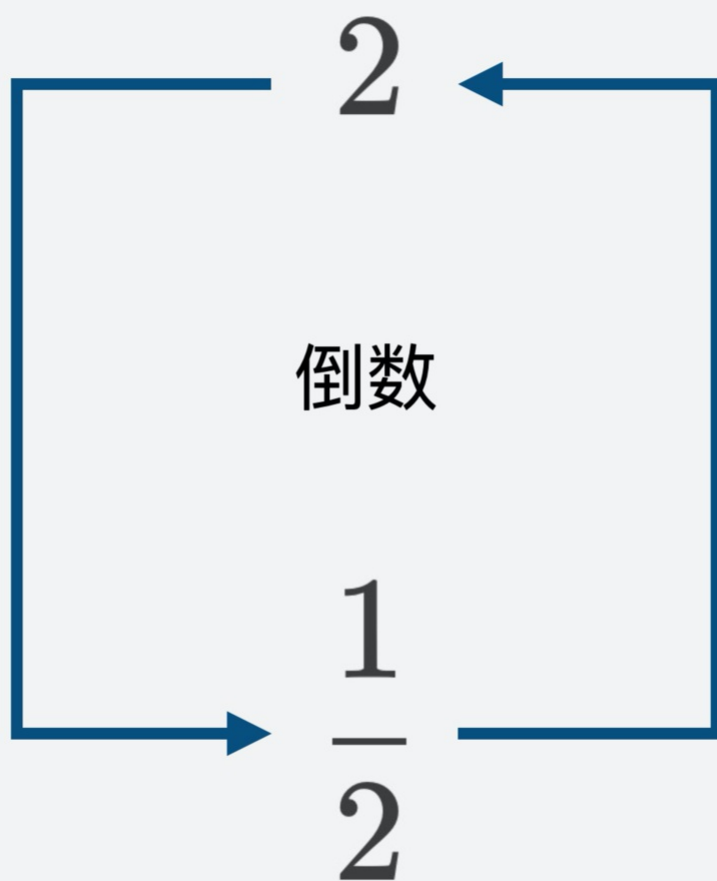
注意，这里的行和列不同， $m \neq n$ 意味着，根据矩阵乘，左乘和右乘单位矩阵也不同，也就是 $\mathbf{I}_m \neq \mathbf{I}_n$ 。

逆矩阵与转置矩阵

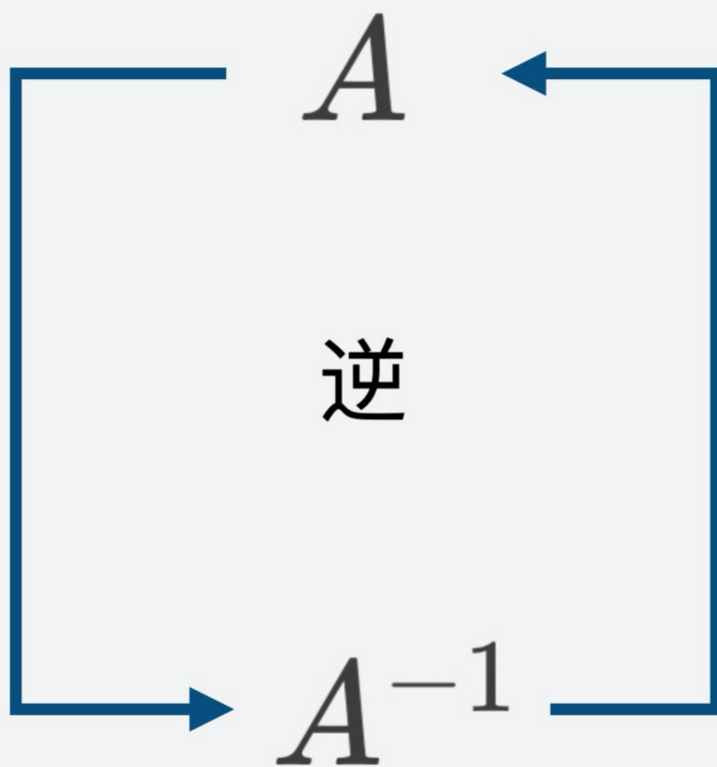
了解矩阵基本概念、运算，以及性质后，我来讲一讲矩阵应用中的两个核心内容——逆矩阵和转置矩阵。逆矩阵和转置矩阵在实际应用中大有用处，比如：坐标系中的图形变换运算。我们先来看下什么是逆矩阵。

逆矩阵

下面这个图你应该非常熟悉了，图中表现的是数字的倒数，2的倒数是 $\frac{1}{2}$ ， $\frac{1}{2}$ 的倒数是2。



其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{SA} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```
SS
A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\left[\begin{array}{cc}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{array}\right]
SS
```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```
SS
A \times A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]
SS
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} \times a_{11} + \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \times a_{12} = 1
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} \times a_{11} + \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}} \times a_{22} = 0
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} \times a_{21} + \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \times a_{22} = 0
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} \times a_{21} + \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}} \times a_{22} = 1
SS
1 & 0 \\
0 & 1
\end{array}\right]
SS
```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程 $XA=B$ 。

小孩	大人	大巴	火车
$x_1$	$x_2$	3	3.5
		3.2	3.6

要解 $XS$ ，我们就要先计算 $S$ 的逆矩阵 $A^{-1}$ ：

$$\begin{array}{l} A^{-1} = \left\| \begin{array}{cc} 3 & 3.5 \\ 3.2 & 3.6 \end{array} \right\| \\ \text{end{array}} \right\|^{-1} = \frac{1}{\{3\} \times 3.6 - 3.5 \times 3.2} \left\| \begin{array}{cc} 3.6 & -3.5 \\ -3.2 & 3 \end{array} \right\| \\ \text{end{array}} \right\| \\ -9 & 8.75 \\ 8 & -7.5 \\ \text{end{array}} \right\| \end{array}$$

接下来再计算  $X = B A^{-1} S$ :

$$\begin{aligned} & \left\lfloor \begin{array}{l} x_1 \\ 118.4 \\ -9.8 \\ 8 \end{array} \right\rfloor = \left\lfloor \begin{array}{l} x_2 \\ 135.2 \\ 8.75 \\ -8.75 \end{array} \right\rfloor \\ & \left\lfloor \begin{array}{l} 16 \\ 22 \end{array} \right\rfloor \end{aligned}$$

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序包封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least, 方程次序很重要, 也就是说,  $SAX=BS$  和  $SXA=BS$  的结果是不同的, 这个一定要牢记哦!

## 转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把  $m \times n$  矩阵  $A$  的行列互换，得到转置矩阵  $A^T$ 。

```

SS
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \& a_{1n} \\
a_{21} & a_{22} & \& a_{2n} \\
\& \& \& \& \\
a_{m1} & a_{m2} & \& a_{mn}
\end{array}\right]
SS

```

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

\end{array}'right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

- 1. 矩阵和自身逆矩阵相乘得单位矩阵， $SA A^{-1}=I=A^{-1} A$ ；
- 2.  $SASBS$ 两矩阵相乘的逆，等于逆矩阵 $SBS$ 和逆矩阵 $SAS$ 相乘，这里强调一下乘的顺序很重要， $S(A B)^{-1}=B^{-1} A^{-1}$ ；
- 3.  $SABS$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $S(A+B)^{-1}\neq A^{-1}+B^{-1}$ ；
- 4. 矩阵转置的转置还是它本身， $S\left(A^T\right)^{\mathrm{T}}=A$ ；
- 5.  $SABS$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $S(A+B)^T=A^T+B^T$ ；
- 6.  $SABS$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $B$ 和转置矩阵 $A$ 的相乘，这里再次强调乘的顺序很重要， $S(A B)^T=B^T A^T$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $S=\left(x_1,\ldots,x_{10}\right)^T$ ， $Sv=\left(v_1,\ldots,v_{10}\right)^T$ ，如果要计算 $Sy=x^T\left(I+vv^T\right)x$ ，其中 $S$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

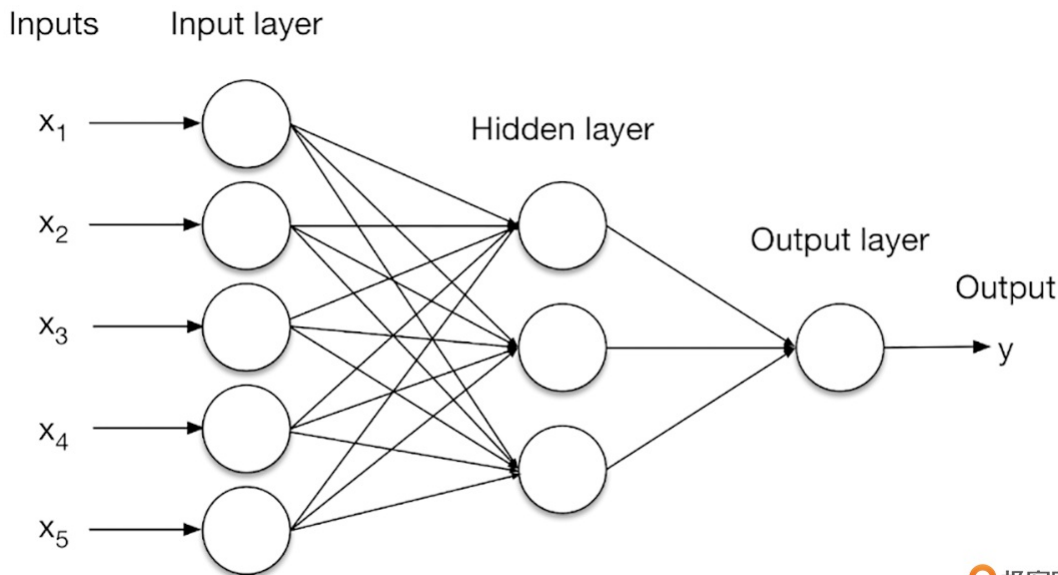
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$A \cdot X = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵\$\widetilde{A}\$， 可以表示为\$(A, B)\$， 这里的\$B\$表示的是方程组常数项所构成的列向量， 也就是\$m\times 1\$的\$m\$行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为\$n\times 1\$的\$n\$行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$， 就可以表示为\$AX=B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵\$\widetilde{A}\$的列向量， 则线性方程组\$A\$X=\$B\$又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、 图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由\$m\times n\$个元素组成， \$m\$和\$n\$是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按\$m\$行\$n\$列的矩形排布方式后可以形成矩阵\$A\$：

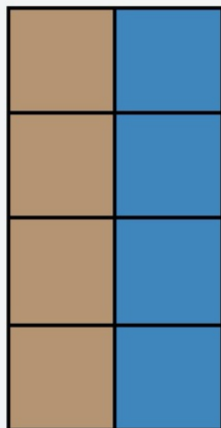
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例， \$(1, n)\$矩阵叫做行， \$(m, 1)\$矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。 矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设\$R^{m\times n}\$是实数矩阵\$(m, n)\$的集合， \$A\in R^{m\times n}\$可以表示成另一种形式 \$A\in R^{mm}\$。 我们把矩阵的\$n\$列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、 列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \cdots & a_{1n}+b_{1n} \\
\cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots \\
a_{m1}+b_{m1} & \cdots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \cdots, m, j=1, \cdots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。

\$\$AB=CS\$

但反过来B和A相乘就不行了，因为相邻阶数 $S_mS$ 不等于 $S_nS$ 。

2.哈达玛积

哈达玛积理解起来就很简单了，就是矩阵各对应元素的乘积， $c_{c_{ij}}=a_{ij} \times b_{ij}$ 。举个例子：

```
$$
C=A^{(*)} B=\left[\begin{array}{ll}
1 & 2 \\
4 & 5
\end{array}\right]\left[\begin{array}{ll}
1 & 4 \\
2 & 5
\end{array}\right]=\left[\begin{array}{cc}
1 * 1 & 2 * 4 \\
4 * 2 & 5 * 5
\end{array}\right]=\left[\begin{array}{cc}
1 & 8 \\
8 & 25
\end{array}\right]
$$
```

哈达玛积其实在数学中不常看到，不过，在编程中哈达玛积非常有用，因为它可以用来同时计算多组数据的乘积，计算效率很高。

3.克罗内克积

克罗内克积是以德国数学家利奥波德·克罗内克（Leopold Kronecker）的名字命名的。它可以应用在解线性矩阵方程和图像处理方面，当然从更时髦的角度说，它还能用在量子信息领域，我们也称之为直积或张量积。

和普通矩阵乘和哈达玛积不同的是，克罗内克积是两个任意大小矩阵间的运算，表示为 $A \times B$ ，如果 $A$ 是一个 $m \times n$ 的矩阵，而 $B$ 是一个 $p \times q$ 的矩阵，克罗内克积则是一个 $mp \times nq$ 的矩阵。

接下来我们需要定义一个在矩阵的乘法中起着特殊作用的矩阵，它就是单位矩阵。高等代数中，在求解相应的矩阵时，若添加单位矩阵，通过初等变换进行求解，往往可以使问题变得简单。按照百度百科的解释，单位矩阵如同数的乘法中的1，这种矩阵就被称为单位矩阵。它是个方阵，从左上角到右下角的对角线，也就是主对角线上的元素均为1，除此以外全都为0。

在线性代数中，大小为 $n$ 的单位矩阵就是在主对角线上均为1，而其他地方都是0的 $n \times n$ 的方阵，它用 $\mathbf{I}_n$ 表示，表达时为了方便可以忽略阶数，直接用 $\mathbf{I}$ 来表示：

```
$$
\mathbf{I}_1=[1], \mathbf{I}_2=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right], \mathbf{I}_3=\left[\begin{array}{lll}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right], \dots, \mathbf{I}_n=\left[\begin{array}{cccc}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
. & . & \dots & . \\
. & . & . & . \\
0 & 0 & \dots & 1
\end{array}\right]
$$
```

矩阵的性质

在了解了矩阵加和乘，以及单位矩阵后，我们是时候来看一看矩阵的性质了。了解矩阵的性质是进行矩阵计算的前提，就像我们小时候学加减乘除四则运算法则时那样。所以，这块内容对你来说应该不难，你作为了解就好，重点是之后的运算。

1.结合律

任意实数 $m \times n$ 矩阵 $A$ ， $n \times p$ 矩阵 $B$ ， $p \times q$ 矩阵 $C$ 之间相乘，满足结合律 $(AB)C=A(BC)$ 。这个很好理解，我就不多说了。

\$\$\forall \text{all } A \in R^{m \times n}, B \in R^{n \times p}, C \in R^{p \times q} \{ (A B) C=A(B C) \$\$

2.分配律

任意实数 $m \times n$ 矩阵 $A$ 和 $B$ ， $n \times p$ 矩阵 $C$ 和 $D$ 之间相乘满足分配律 $(A+B)C=AC+BC$ ， $A(C+D)=AC+AD$ 。

\$\$\forall \text{all } \mathbf{A}, \mathbf{B} \in \mathbf{R}^{m \times n}, \mathbf{C}, \mathbf{D} \in \mathbf{R}^{n \times p} \{ (\mathbf{A}+\mathbf{B}) \mathbf{C}=\mathbf{A} \mathbf{C}+\mathbf{B} \mathbf{C}, \mathbf{A}(\mathbf{C}+\mathbf{D})=\mathbf{A} \mathbf{C}+\mathbf{A} \mathbf{D} \$\$

3.单位矩阵乘

任意实数 $m \times n$ 矩阵A和单位矩阵之间的乘，等于它本身 $A$ 。

\$\$\forall \text{all } A \in R^{m \times n} : \mathbf{I}\_m A=A \mathbf{I}\_n=A \$\$

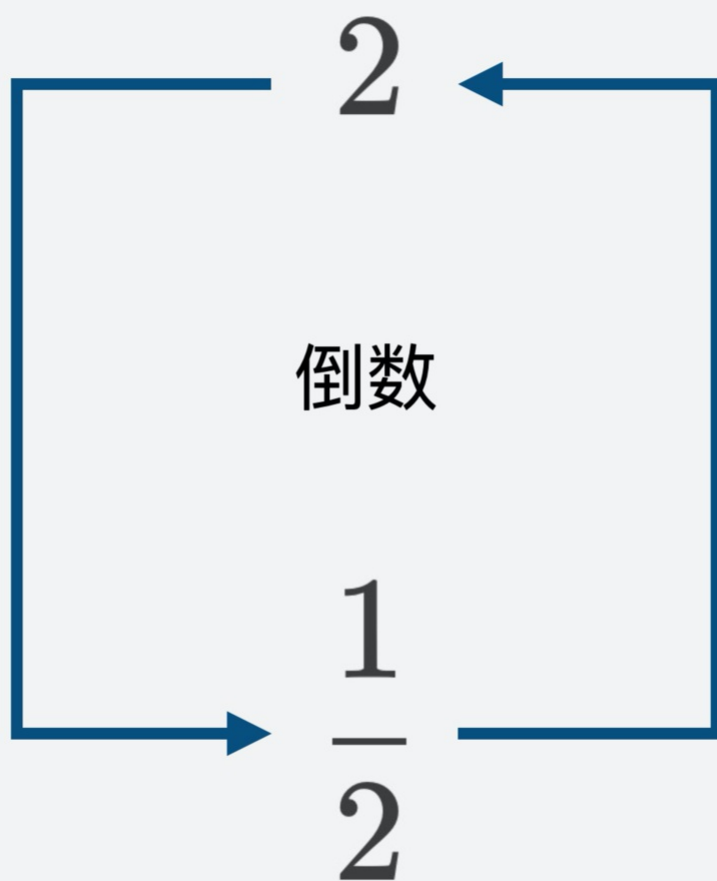
注意，这里的行和列不同， $m \neq n$ 意味着，根据矩阵乘，左乘和右乘单位矩阵也不同，也就是 $\mathbf{I}_m \neq \mathbf{I}_n$ 。

逆矩阵与转置矩阵

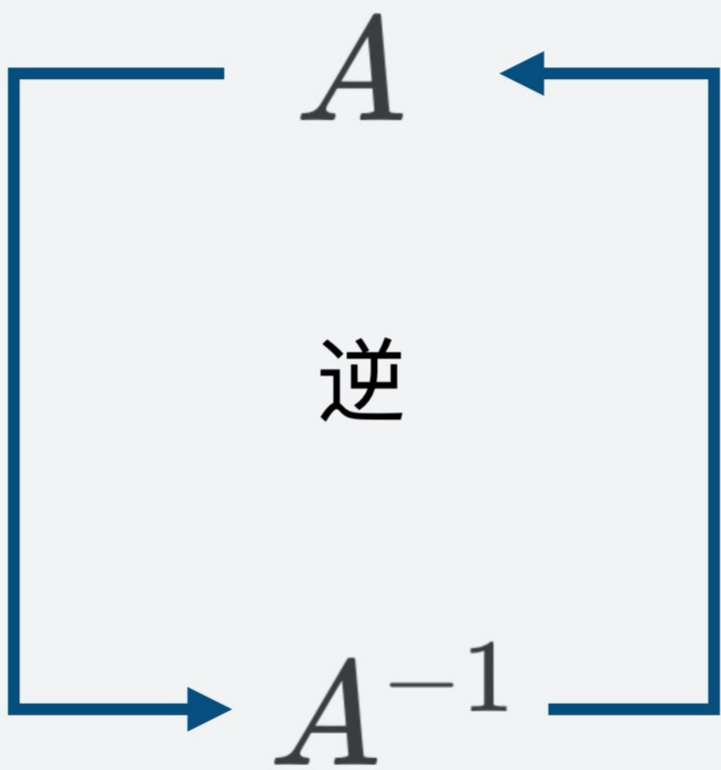
了解矩阵基本概念、运算，以及性质后，我来讲一讲矩阵应用中的两个核心内容——逆矩阵和转置矩阵。逆矩阵和转置矩阵在实际应用中大有用处，比如：坐标系中的图形变换运算。我们先来看下什么是逆矩阵。

逆矩阵

下面这个图你应该非常熟悉了，图中表现的是数字的倒数，2的倒数是 $\frac{1}{2}$ ， $\frac{1}{2}$ 的倒数是2。



其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```

A^{-1} = \frac{1}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}

```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```

A \times A^{-1} = \frac{1}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}
= \frac{1}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}} \begin{bmatrix} a_{22}a_{11} - a_{12}a_{21} & a_{22}a_{12} - a_{12}a_{22} \\ -a_{21}a_{11} + a_{11}a_{21} & -a_{21}a_{12} + a_{11}a_{22} \end{bmatrix}
= \frac{1}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}} \begin{bmatrix} a_{22}a_{11} - a_{12}a_{21} & 0 \\ 0 & a_{22}a_{11} - a_{12}a_{21} \end{bmatrix}
= \frac{1}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}} \begin{bmatrix} a_{22}a_{11} - a_{12}a_{21} & 0 \\ 0 & a_{22}a_{11} - a_{12}a_{21} \end{bmatrix}
= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}

```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程\$XA=BS\$。

小孩          大人          大巴          火车

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 3 & 3.5 \\ 3.2 & 3.6 \end{bmatrix}$$

要解\$XS\$，我们就要先计算\$A\$的逆矩阵\$A^{-1}\$：

```
$$
A^{-1}=\left[\begin{array}{cc}
3 & 3.5 \\
3.2 & 3.6
\end{array}\right]^{-1}=\frac{1}{3\times 3.6-3.5\times 3.2}\left[\begin{array}{cc}
3.6 & -3.5 \\
-3.2 & 3
\end{array}\right]
=\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
$$
```

接下来再计算\$X=BA^{-1}\$：

```
$$
\left[\begin{array}{ll}
x_1 & x_2
\end{array}\right]=\left[\begin{array}{ll}
118.4 & 135.2
\end{array}\right]\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
=\left[\begin{array}{ll}
16 & 22
\end{array}\right]
$$
```

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序包封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least，方程次序很重要，也就是说，\$AX=BS\$和\$XA=BS\$的结果是不同的，这个一定要牢记哦！

转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把\$m\times n\$矩阵\$A\$的行列互换，得到转置矩阵\$A^T\$。

```
$$
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\ldots & \ldots & \ldots & \ldots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{array}\right]
$$
```

```
$$
A^T=\left[\begin{array}{cccc}
a_{11} & a_{21} & \ldots & a_{m1} \\
a_{12} & a_{22} & \ldots & a_{m2} \\
\ldots & \ldots & \ldots & \ldots \\
a_{1n} & a_{2n} & \ldots & a_{mn}
\end{array}\right]

```

\end{array}'right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

1. 矩阵和自身逆矩阵相乘得单位矩阵， $SA A^{-1}=I=A^{-1} A$ ；
2.  $SASBS$ 两矩阵相乘的逆，等于逆矩阵 $SBS$ 和逆矩阵 $SAS$ 相乘，这里强调一下乘的顺序很重要， $S(A B)^{-1}=B^{-1} A^{-1}$ ；
3.  $SABS$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $S(A+B)^{-1}\neq A^{-1}+B^{-1}$ ；
4. 矩阵转置的转置还是它本身， $S\left(A^T\right)^{\mathrm{T}}=A$ ；
5.  $SABS$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $S(A+B)^T=A^T+B^T$ ；
6.  $SABS$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $B$ 和转置矩阵 $A$ 的相乘，这里再次强调乘的顺序很重要， $S(A B)^T=B^T A^T$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $S=\left(x_1, \ldots, x_{10}\right)^T$ ， $S_v=\left(v_1, \ldots, v_{10}\right)^T$ ，如果要计算 $S_y=x^T\left(I+v v^T\right) x$ ，其中 $S$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

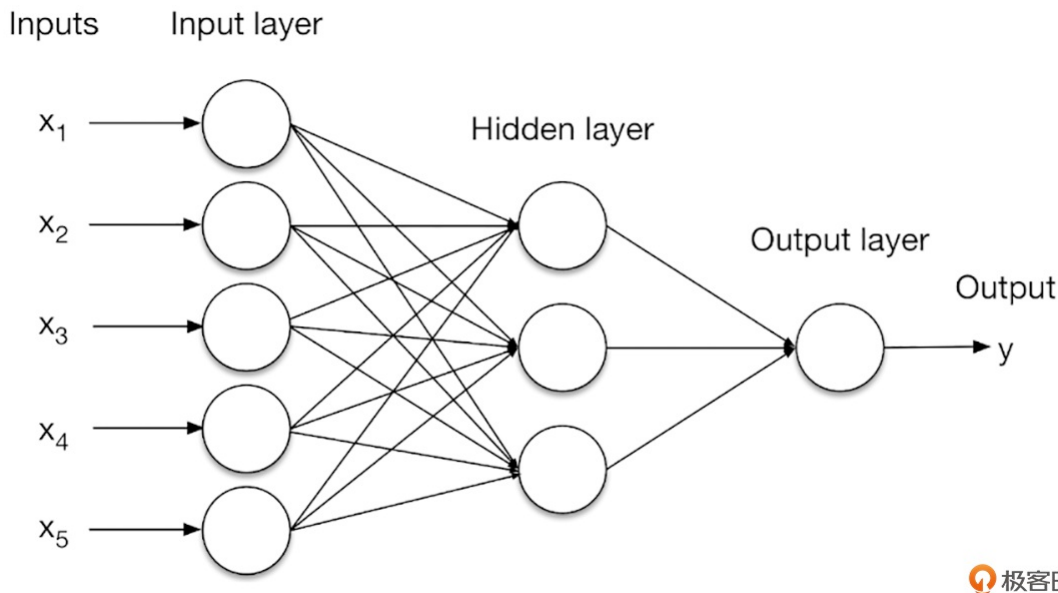
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{41} & w_{42} \\ w_{31} & w_{32} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} \\ w_{41} & w_{42} \\ w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$A \cdot X = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵 $\widetilde{A}$ ， 可以表示为\$(A, B)\$，这里的\$B\$表示的是方程组常数项所构成的列向量，也就是 $m\times 1$ 的 $m$ 行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为 $n\times 1$ 的 $n$ 行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$，就可以表示为\$A\$X=\$B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵 $\widetilde{A}$ 的列向量，则线性方程组\$A\$X=\$B\$又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由 $m\times n$ 个元素组成，  $m$ 和 $n$ 是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按 $m$ 行 $n$ 列的矩形排布方式后可以形成矩阵\$A\$：

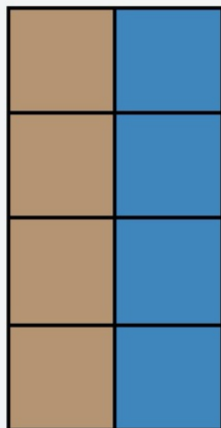
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例，  $(1, n)$ 矩阵叫做行，  $(m, 1)$ 矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设 $R^{m\times n}$ 是实数矩阵\$(m, n)\$的集合，  $A\in R^{m\times n}$ 可以表示成另一种形式  $A\in R^{mm}$ 。 我们把矩阵的 $n$ 列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \cdots & a_{1n}+b_{1n} \\
\cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots \\
a_{m1}+b_{m1} & \cdots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \cdots, m, j=1, \cdots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。

\$\$AB=CS\$

但反过来B和A相乘就不行了，因为相邻阶数 $S_mS$ 不等于 $S_nS$ 。

2.哈达玛积

哈达玛积理解起来就很简单了，就是矩阵各对应元素的乘积， $c_{c_{ij}}=a_{ij} \times b_{ij}$ 。举个例子：

```
$$
C=A^{(*)} B=\left[\begin{array}{ll}
1 & 2 \\
4 & 5
\end{array}\right]\left[\begin{array}{ll}
1 & 4 \\
2 & 5
\end{array}\right]=\left[\begin{array}{cc}
1 * 1 & 2 * 4 \\
4 * 2 & 5 * 5
\end{array}\right]=\left[\begin{array}{cc}
1 & 8 \\
8 & 25
\end{array}\right]
$$
```

哈达玛积其实在数学中不常看到，不过，在编程中哈达玛积非常有用，因为它可以用来同时计算多组数据的乘积，计算效率很高。

3.克罗内克积

克罗内克积是以德国数学家利奥波德·克罗内克（Leopold Kronecker）的名字命名的。它可以应用在解线性矩阵方程和图像处理方面，当然从更时髦的角度说，它还能用在量子信息领域，我们也称之为直积或张量积。

和普通矩阵乘和哈达玛积不同的是，克罗内克积是两个任意大小矩阵间的运算，表示为 $A \times B$ ，如果 $A$ 是一个 $m \times n$ 的矩阵，而 $B$ 是一个 $p \times q$ 的矩阵，克罗内克积则是一个 $mp \times nq$ 的矩阵。

接下来我们需要定义一个在矩阵的乘法中起着特殊作用的矩阵，它就是单位矩阵。高等代数中，在求解相应的矩阵时，若添加单位矩阵，通过初等变换进行求解，往往可以使问题变得简单。按照百度百科的解释，单位矩阵如同数的乘法中的1，这种矩阵就被称为单位矩阵。它是个方阵，从左上角到右下角的对角线，也就是主对角线上的元素均为1，除此以外全都为0。

在线性代数中，大小为 $n$ 的单位矩阵就是在主对角线上均为1，而其他地方都是0的 $n \times n$ 的方阵，它用 $\mathbf{I}_n$ 表示，表达时为了方便可以忽略阶数，直接用 $\mathbf{I}$ 来表示：

```
$$
\mathbf{I}_1=[1], \mathbf{I}_2=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right], \mathbf{I}_3=\left[\begin{array}{lll}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right], \dots, \mathbf{I}_n=\left[\begin{array}{cccc}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
. & . & \dots & . \\
. & . & . & . \\
0 & 0 & \dots & 1
\end{array}\right]
$$
```

矩阵的性质

在了解了矩阵加和乘，以及单位矩阵后，我们是时候来看一看矩阵的性质了。了解矩阵的性质是进行矩阵计算的前提，就像我们小时候学加减乘除四则运算法则时那样。所以，这块内容对你来说应该不难，你作为了解就好，重点是之后的运算。

1.结合律

任意实数 $m \times n$ 矩阵 $A$ ， $n \times p$ 矩阵 $B$ ， $p \times q$ 矩阵 $C$ 之间相乘，满足结合律 $(AB)C=A(BC)$ 。这个很好理解，我就不多说了。

\$\$\forall a \in R^{\{m \times n\}}, B \in R^{\{n \times p\}}, C \in R^{\{p \times q\}} \{ (A B) C = A (B C) \$\$

2.分配律

任意实数 $m \times n$ 矩阵 $A$ 和 $B$ ， $n \times p$ 矩阵 $C$ 和 $D$ 之间相乘满足分配律 $(A+B)C=AC+BC$ ， $A(C+D)=AC+AD$ 。

\$\$\forall a \in \mathbf{M}\{A\}, B \in \mathbf{M}\{R^{\{m \times n\}}\}, C, D \in \mathbf{M}\{R^{\{n \times p\}}\} \{ (A+B) C = A C + B C, A (C+D) = A C + A D \$\$

3.单位矩阵乘

任意实数 $m \times n$ 矩阵A和单位矩阵之间的乘，等于它本身 $A$ 。

\$\$\forall a \in R^{\{m \times n\}}: \mathbf{I}\_m A = A \mathbf{I}\_n = A \$\$

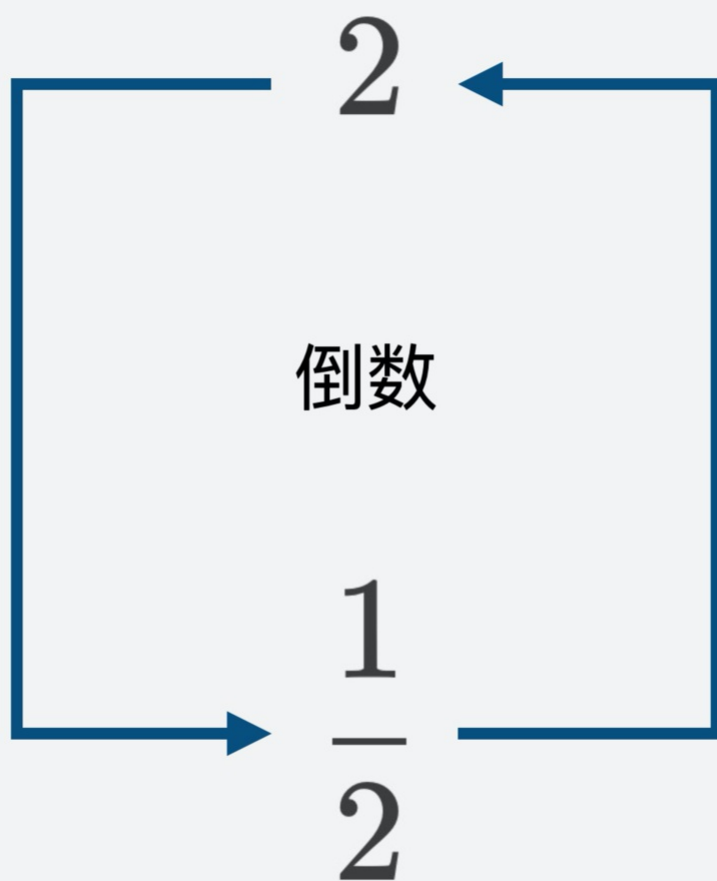
注意，这里的行和列不同， $m \neq n$ 意味着，根据矩阵乘，左乘和右乘单位矩阵也不同，也就是 $\mathbf{I}_m \neq \mathbf{I}_n$ 。

逆矩阵与转置矩阵

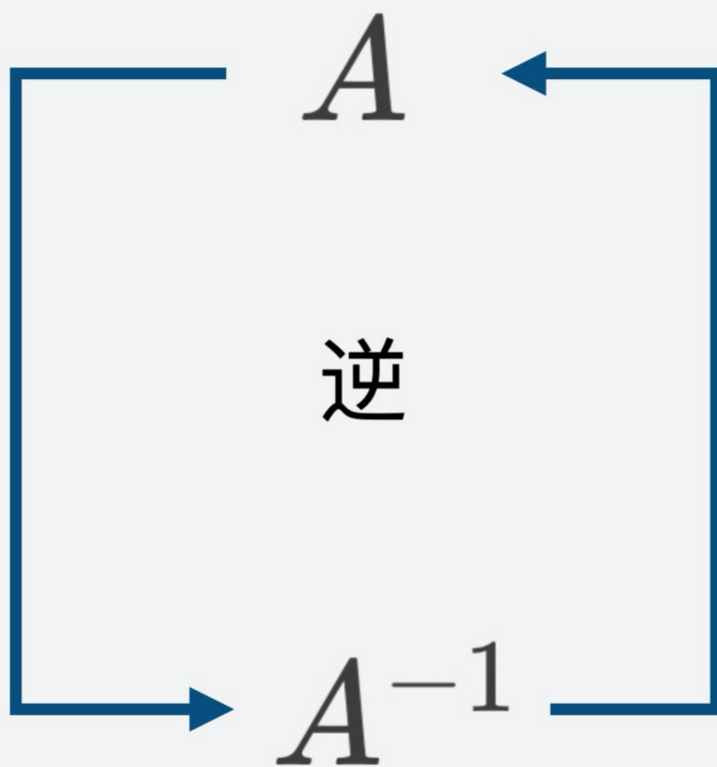
了解矩阵基本概念、运算，以及性质后，我来讲一讲矩阵应用中的两个核心内容——逆矩阵和转置矩阵。逆矩阵和转置矩阵在实际应用中大有用处，比如：坐标系中的图形变换运算。我们先来看下什么是逆矩阵。

逆矩阵

下面这个图你应该非常熟悉了，图中表现的是数字的倒数，2的倒数是 $\frac{1}{2}$ ， $\frac{1}{2}$ 的倒数是2。



其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```
SS
A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\left[\begin{array}{cc}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{array}\right]
SS
```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```
SS
A \times A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right]
SS
```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程\$XA=BS\$。

小孩          大人          大巴          火车

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 3 & 3.5 \\ 3.2 & 3.6 \end{bmatrix}$$

要解\$XS\$，我们就要先计算\$A\$的逆矩阵\$A^{-1}\$：

```
$$
A^{-1}=\left[\begin{array}{cc}
3 & 3.5 \\
3.2 & 3.6
\end{array}\right]^{-1}=\frac{1}{3\times 3.6-3.5\times 3.2}\left[\begin{array}{cc}
3.6 & -3.5 \\
-3.2 & 3
\end{array}\right]
=\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
$$
```

接下来再计算\$X=BA^{-1}\$：

```
$$
\left[\begin{array}{ll}
x_1 & x_2
\end{array}\right]=\left[\begin{array}{ll}
118.4 & 135.2
\end{array}\right]\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
=\left[\begin{array}{ll}
16 & 22
\end{array}\right]
$$
```

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序包封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least，方程次序很重要，也就是说，\$AX=BS\$和\$XA=BS\$的结果是不同的，这个一定要牢记哦！

转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把\$m\times n\$矩阵\$A\$的行列互换，得到转置矩阵\$A^T\$。

```
$$
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\ldots & \ldots & \ldots & \ldots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{array}\right]
$$
```

```
$$
A^T=\left[\begin{array}{cccc}
a_{11} & a_{21} & \ldots & a_{m1} \\
a_{12} & a_{22} & \ldots & a_{m2} \\
\ldots & \ldots & \ldots & \ldots \\
a_{1n} & a_{2n} & \ldots & a_{mn}
\end{array}\right]

```

\end{array}\right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

1. 矩阵和自身逆矩阵相乘得单位矩阵， $\mathbf{A} \mathbf{A}^{-1} = \mathbf{I} = \mathbf{A}^{-1} \mathbf{A}$ ；
2.  $\mathbf{A} \mathbf{B}$ 两矩阵相乘的逆，等于逆矩阵 $\mathbf{B}$ 和逆矩阵 $\mathbf{A}$ 相乘，这里强调一下乘的顺序很重要， $(\mathbf{A} \mathbf{B})^{-1} = \mathbf{B}^{-1} \mathbf{A}^{-1}$ ；
3.  $\mathbf{A} \mathbf{B}$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $(\mathbf{A} + \mathbf{B})^{-1} \neq \mathbf{A}^{-1} + \mathbf{B}^{-1}$ ；
4. 矩阵转置的转置还是它本身， $\left(\mathbf{A}^T\right)^T = \mathbf{A}$ ；
5.  $\mathbf{A} \mathbf{B}$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$ ；
6.  $\mathbf{A} \mathbf{B}$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $\mathbf{B}$ 和转置矩阵 $\mathbf{A}$ 的相乘，这里再次强调乘的顺序很重要， $(\mathbf{A} \mathbf{B})^T = \mathbf{B}^T \mathbf{A}^T$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_{10} \end{pmatrix}$ ， $\mathbf{S} = \begin{pmatrix} v_1 \\ \vdots \\ v_{10} \end{pmatrix}$ ，如果要计算 $\mathbf{y} = \mathbf{x}^T \begin{pmatrix} \mathbf{I} + \mathbf{v} \mathbf{v}^T \end{pmatrix} \mathbf{x}$ ，其中 $\mathbf{I}$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

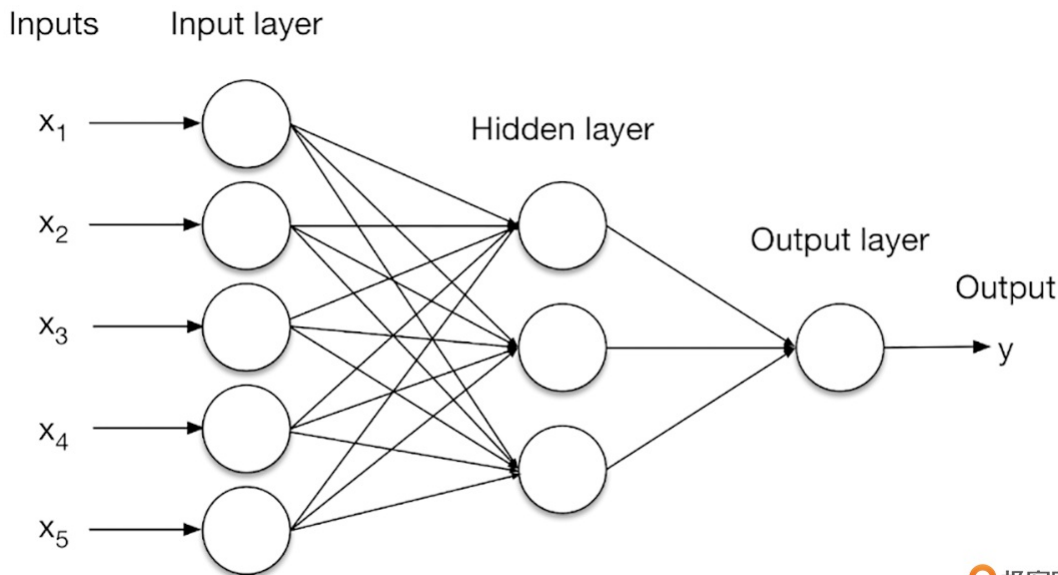
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$A \cdot X = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵\$\widetilde{A}\$， 可以表示为\$(A, B)\$，这里的\$B\$表示的是方程组常数项所构成的列向量，也就是\$m\times 1\$的\$m\$行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为\$n\times 1\$的\$n\$行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$，就可以表示为\$A\$X=\$B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵\$\widetilde{A}\$的列向量，则线性方程组\$A\$X又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由\$m\times n\$个元素组成， \$m\$和\$n\$是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按\$m\$行\$n\$列的矩形排布方式后可以形成矩阵\$A\$：

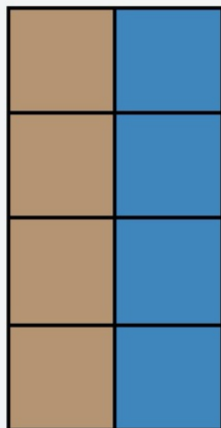
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例， \$(1, n)\$矩阵叫做行， \$(m, 1)\$矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设\$R^{m\times n}\$是实数矩阵\$(m, n)\$的集合， \$A\in R^{m\times n}\$可以表示成另一种形式 \$A\in R^{mm}\$。 我们把矩阵的\$n\$列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \cdots & a_{1n}+b_{1n} \\
\cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots \\
a_{m1}+b_{m1} & \cdots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \cdots, m, j=1, \cdots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。

\$\$AB=CS\$

但反过来B和A相乘就不行了，因为相邻阶数 $S_mS$ 不等于 $S_nS$ 。

2.哈达玛积

哈达玛积理解起来就很简单了，就是矩阵各对应元素的乘积， $c_{c_{ij}}=a_{ij} \times b_{ij}$ 。举个例子：

```
$$
C=A^{(*)} B=\left[\begin{array}{ll}
1 & 2 \\
4 & 5
\end{array}\right]\left[\begin{array}{ll}
1 & 4 \\
2 & 5
\end{array}\right]=\left[\begin{array}{cc}
1 * 1 & 2 * 4 \\
4 * 2 & 5 * 5
\end{array}\right]=\left[\begin{array}{cc}
1 & 8 \\
8 & 25
\end{array}\right]
$$
```

哈达玛积其实在数学中不常看到，不过，在编程中哈达玛积非常有用，因为它可以用来同时计算多组数据的乘积，计算效率很高。

3.克罗内克积

克罗内克积是以德国数学家利奥波德·克罗内克（Leopold Kronecker）的名字命名的。它可以应用在解线性矩阵方程和图像处理方面，当然从更时髦的角度说，它还能用在量子信息领域，我们也称之为直积或张量积。

和普通矩阵乘和哈达玛积不同的是，克罗内克积是两个任意大小矩阵间的运算，表示为 $A \times B$ ，如果 $A$ 是一个 $m \times n$ 的矩阵，而 $B$ 是一个 $p \times q$ 的矩阵，克罗内克积则是一个 $m \times p \times n \times q$ 的矩阵。

接下来我们需要定义一个在矩阵的乘法中起着特殊作用的矩阵，它就是单位矩阵。高等代数中，在求解相应的矩阵时，若添加单位矩阵，通过初等变换进行求解，往往可以使问题变得简单。按照百度百科的解释，单位矩阵如同数的乘法中的1，这种矩阵就被称为单位矩阵。它是个方阵，从左上角到右下角的对角线，也就是主对角线上的元素均为1，除此以外全都为0。

在线性代数中，大小为 $n$ 的单位矩阵就是在主对角线上均为1，而其他地方都是0的 $n \times n$ 的方阵，它用 $\mathbf{I}_n$ 表示，表达时为了方便可以忽略阶数，直接用 $\mathbf{I}$ 来表示：

```
$$
\mathbf{I}_1=[1], \mathbf{I}_2=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right], \mathbf{I}_3=\left[\begin{array}{lll}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right], \dots, \mathbf{I}_n=\left[\begin{array}{cccc}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
. & . & \dots & . \\
. & . & . & . \\
0 & 0 & \dots & 1
\end{array}\right]
$$
```

矩阵的性质

在了解了矩阵加和乘，以及单位矩阵后，我们是时候来看一看矩阵的性质了。了解矩阵的性质是进行矩阵计算的前提，就像我们小时候学加减乘除四则运算法则时那样。所以，这块内容对你来说应该不难，你作为了解就好，重点是之后的运算。

1.结合律

任意实数 $m \times n$ 矩阵 $A$ ， $n \times p$ 矩阵 $B$ ， $p \times q$ 矩阵 $C$ 之间相乘，满足结合律 $(AB)C=A(BC)$ 。这个很好理解，我就不多说了。

\$\$\forall a \in R^{\{m \times n\}}, B \in R^{\{n \times p\}}, C \in R^{\{p \times q\}} \{ (A B) C = A (B C) \$\$

2.分配律

任意实数 $m \times n$ 矩阵 $A$ 和 $B$ ， $n \times p$ 矩阵 $C$ 和 $D$ 之间相乘满足分配律 $(A+B)C=AC+BC$ ， $A(C+D)=AC+AD$ 。

\$\$\forall a \in \mathbf{M}\{A\}, B \in \mathbf{M}\{R^{\{m \times n\}}\}, C, D \in \mathbf{M}\{R^{\{n \times p\}}\} \{ (A+B) C = A C + B C, A (C+D) = A C + A D \$\$

3.单位矩阵乘

任意实数 $m \times n$ 矩阵A和单位矩阵之间的乘，等于它本身 $A$ 。

\$\$\forall a \in R^{\{m \times n\}}: \mathbf{I}\_m A = A \mathbf{I}\_n = A \$\$

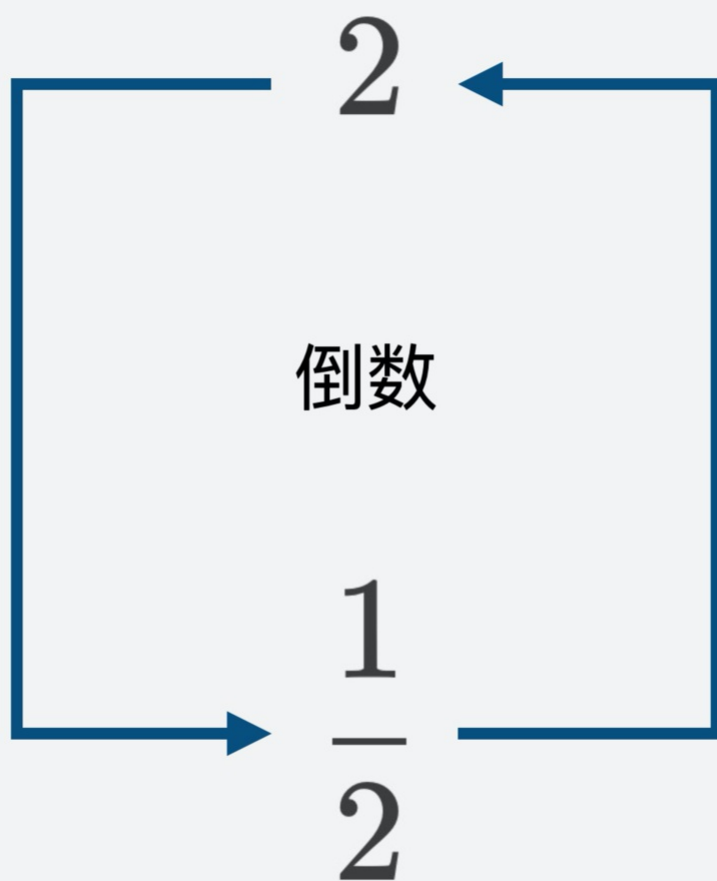
注意，这里的行和列不同， $m \neq n$ 意味着，根据矩阵乘，左乘和右乘单位矩阵也不同，也就是 $\mathbf{I}_m \neq \mathbf{I}_n$ 。

逆矩阵与转置矩阵

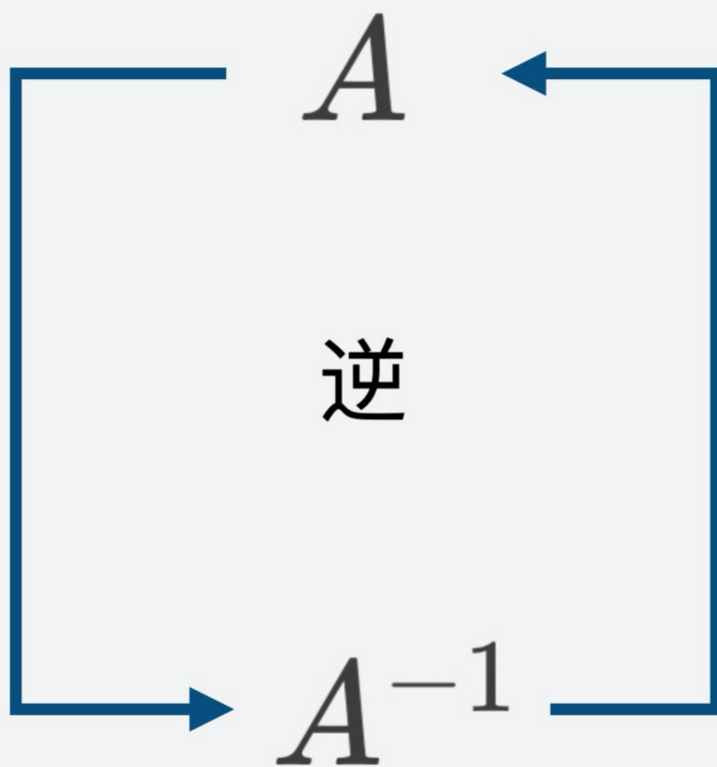
了解矩阵基本概念、运算，以及性质后，我来讲一讲矩阵应用中的两个核心内容——逆矩阵和转置矩阵。逆矩阵和转置矩阵在实际应用中大有用处，比如：坐标系中的图形变换运算。我们先来看下什么是逆矩阵。

逆矩阵

下面这个图你应该非常熟悉了，图中表现的是数字的倒数，2的倒数是 $\frac{1}{2}$ ， $\frac{1}{2}$ 的倒数是2。



其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```
SS
A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\left[\begin{array}{cc}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{array}\right]
SS
```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```
SS
A \times A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]
SS
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{11}+\frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{21}=\frac{a_{11}a_{22}-a_{12}a_{21}}{a_{11}a_{22}-a_{12}a_{21}}=1
\frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{11}+\frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{22}=\frac{-a_{12}a_{11}+a_{11}a_{22}}{a_{11}a_{22}-a_{12}a_{21}}=0
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{11}+\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{21}=\frac{-a_{21}a_{11}+a_{22}a_{21}}{a_{11}a_{22}-a_{12}a_{21}}=0
\frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{12}+\frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{22}=\frac{a_{11}a_{12}-a_{12}a_{22}}{a_{11}a_{22}-a_{12}a_{21}}=0
\frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{11}+\frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{22}=\frac{-a_{12}a_{11}+a_{11}a_{22}}{a_{11}a_{22}-a_{12}a_{21}}=1
SS
```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程\$XA=BS\$。

小孩          大人          大巴          火车

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 3.5 \\ 3.2 & 3.6 \end{bmatrix}$$

要解\$XS\$，我们就要先计算\$A\$的逆矩阵\$A^{-1}\$：

```
$$
A^{-1}=\left[\begin{array}{cc}
3 & 3.5 \\
3.2 & 3.6
\end{array}\right]^{-1}=\frac{1}{3\times 3.6-3.5\times 3.2}\left[\begin{array}{cc}
3.6 & -3.5 \\
-3.2 & 3
\end{array}\right]
=\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
$$
```

接下来再计算\$X=BA^{-1}\$：

```
$$
\left[\begin{array}{ll}
x_1 & x_2
\end{array}\right]=\left[\begin{array}{ll}
118.4 & 135.2
\end{array}\right]\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
=\left[\begin{array}{ll}
16 & 22
\end{array}\right]
$$
```

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least，方程次序很重要，也就是说，\$AX=BS\$和\$XA=BS\$的结果是不同的，这个一定要牢记哦！

转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把\$m\times n\$矩阵\$A\$的行列互换，得到转置矩阵\$A^T\$。

```
$$
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\ldots & \ldots & \ldots & \ldots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{array}\right]
$$
```

```
$$
A^T=\left[\begin{array}{cccc}
a_{11} & a_{21} & \ldots & a_{m1} \\
a_{12} & a_{22} & \ldots & a_{m2} \\
\ldots & \ldots & \ldots & \ldots \\
a_{1n} & a_{2n} & \ldots & a_{mn}
\end{array}\right]

```

\end{array}\right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

1. 矩阵和自身逆矩阵相乘得单位矩阵， $AA^{-1}=I=A^{-1}A$ ；
2.  $AB$ 两矩阵相乘的逆，等于逆矩阵 $B$ 和逆矩阵 $A$ 相乘，这里强调一下乘的顺序很重要， $(AB)^{-1}=B^{-1}A^{-1}$ ；
3.  $AB$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $(A+B)^{-1}\neq A^{-1}+B^{-1}$ ；
4. 矩阵转置的转置还是它本身， $\left(A^T\right)^T=A$ ；
5.  $AB$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $(A+B)^T=A^T+B^T$ ；
6.  $AB$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $B$ 和转置矩阵 $A$ 的相乘，这里再次强调乘的顺序很重要， $(AB)^T=B^TA^T$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $x=\left(x_1,\ldots,x_{10}\right)^T$ ， $S=\left(v_1,\ldots,v_{10}\right)^T$ ，如果要计算 $y=x^T\left(I+v\cdot^T\right)x$ ，其中 $I$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

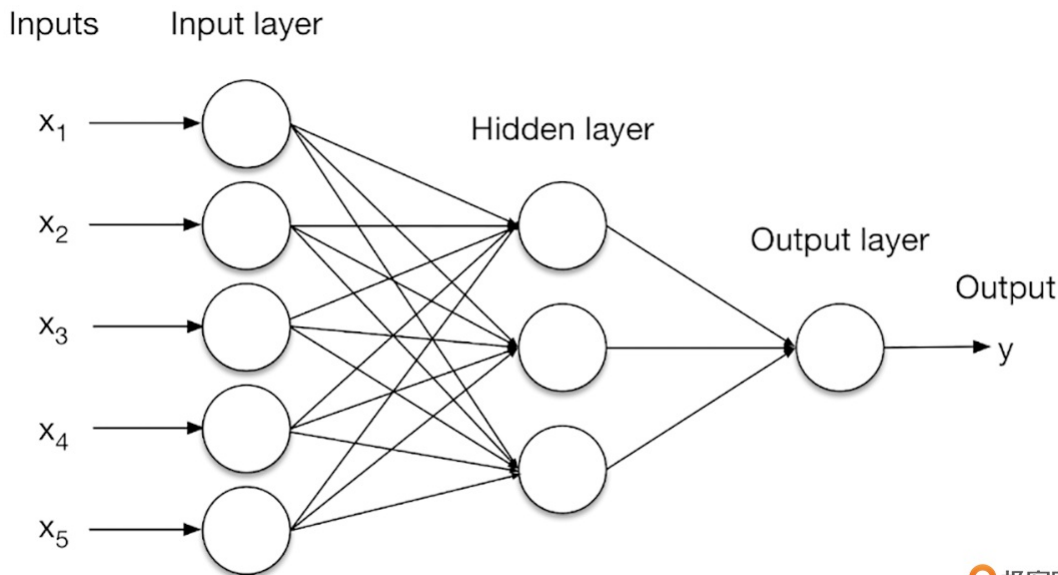
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$AX = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵 $\widetilde{A}$ ， 可以表示为\$(A, B)\$， 这里的\$B\$表示的是方程组常数项所构成的列向量， 也就是 $m\times 1$ 的 $m$ 行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为 $n\times 1$ 的 $n$ 行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$， 就可以表示为\$AX=B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵 $\widetilde{A}$ 的列向量， 则线性方程组\$A\$X=\$B\$又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、 图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解， 我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由 $m\times n$ 个元素组成，  $m$ 和 $n$ 是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按 $m$ 行 $n$ 列的矩形排布方式后可以形成矩阵\$A\$：

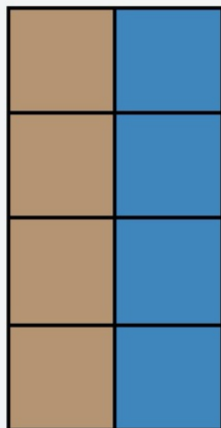
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例，  $(1, n)$ 矩阵叫做行，  $(m, 1)$ 矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后， 我接着讲一个比较有趣的概念， 矩阵转换（Matrix transformation）。 矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。 设 $R^{m\times n}$ 是实数矩阵\$(m, n)\$的集合，  $A\in R^{m\times n}$ 可以表示成另一种形式  $A\in R^{mm}$ 。 我们把矩阵的 $n$ 列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、 列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \cdots & a_{1n}+b_{1n} \\
\cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots \\
a_{m1}+b_{m1} & \cdots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \cdots, m, j=1, \cdots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。

\$\$AB=CS\$

但反过来B和A相乘就不行了，因为相邻阶数 $S_mS$ 不等于 $S_nS$ 。

2.哈达玛积

哈达玛积理解起来就很简单了，就是矩阵各对应元素的乘积， $c_{c_{ij}}=a_{ij} \times b_{ij}$ 。举个例子：

```
$$
C=A^{(*)} B=\left[\begin{array}{ll}
1 & 2 \\
4 & 5
\end{array}\right]\left[\begin{array}{ll}
1 & 4 \\
2 & 5
\end{array}\right]=\left[\begin{array}{cc}
1 * 1 & 2 * 4 \\
4 * 2 & 5 * 5
\end{array}\right]=\left[\begin{array}{cc}
1 & 8 \\
8 & 25
\end{array}\right]
$$
```

哈达玛积其实在数学中不常看到，不过，在编程中哈达玛积非常有用，因为它可以用来同时计算多组数据的乘积，计算效率很高。

3.克罗内克积

克罗内克积是以德国数学家利奥波德·克罗内克（Leopold Kronecker）的名字命名的。它可以应用在解线性矩阵方程和图像处理方面，当然从更时髦的角度说，它还能用在量子信息领域，我们也称之为直积或张量积。

和普通矩阵乘和哈达玛积不同的是，克罗内克积是两个任意大小矩阵间的运算，表示为 $A \times B$ ，如果 $A$ 是一个 $m \times n$ 的矩阵，而 $B$ 是一个 $p \times q$ 的矩阵，克罗内克积则是一个 $m \times p \times n \times q$ 的矩阵。

接下来我们需要定义一个在矩阵的乘法中起着特殊作用的矩阵，它就是单位矩阵。高等代数中，在求解相应的矩阵时，若添加单位矩阵，通过初等变换进行求解，往往可以使问题变得简单。按照百度百科的解释，单位矩阵如同数的乘法中的1，这种矩阵就被称为单位矩阵。它是个方阵，从左上角到右下角的对角线，也就是主对角线上的元素均为1，除此以外全都为0。

在线性代数中，大小为 $n$ 的单位矩阵就是在主对角线上均为1，而其他地方都是0的 $n \times n$ 的方阵，它用 $\mathbf{I}_n$ 表示，表达时为了方便可以忽略阶数，直接用 $\mathbf{I}$ 来表示：

```
$$
\mathbf{I}_1=[1], \mathbf{I}_2=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right], \mathbf{I}_3=\left[\begin{array}{lll}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right], \dots, \mathbf{I}_n=\left[\begin{array}{cccc}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
. & . & \dots & . \\
. & . & . & . \\
0 & 0 & \dots & 1
\end{array}\right]
$$
```

矩阵的性质

在了解了矩阵加和乘，以及单位矩阵后，我们是时候来看一看矩阵的性质了。了解矩阵的性质是进行矩阵计算的前提，就像我们小时候学加减乘除四则运算法则时那样。所以，这块内容对你来说应该不难，你作为了解就好，重点是之后的运算。

1.结合律

任意实数 $m \times n$ 矩阵 $A$ ， $n \times p$ 矩阵 $B$ ， $p \times q$ 矩阵 $C$ 之间相乘，满足结合律 $(AB)C=A(BC)$ 。这个很好理解，我就不多说了。

\$\$\forall a \in R^m, B \in R^{n \times p}, C \in R^{p \times q} \{ (A B) C=A(B C) \$\$

2.分配律

任意实数 $m \times n$ 矩阵 $A$ 和 $B$ ， $n \times p$ 矩阵 $C$ 和 $D$ 之间相乘满足分配律 $(A+B)C=AC+BC$ ， $A(C+D)=AC+AD$ 。

\$\$\forall a \in \mathbf{R}^m, B \in \mathbf{R}^{n \times p}, C, D \in \mathbf{R}^{n \times p} \{ (A+B) C=A C+B C, A(C+D)=A C+A D \$\$

3.单位矩阵乘

任意实数 $m \times n$ 矩阵A和单位矩阵之间的乘，等于它本身 $A$ 。

\$\$\forall a \in R^{m \times n} : \mathbf{I}\_m A=A \mathbf{I}\_n=A \$\$

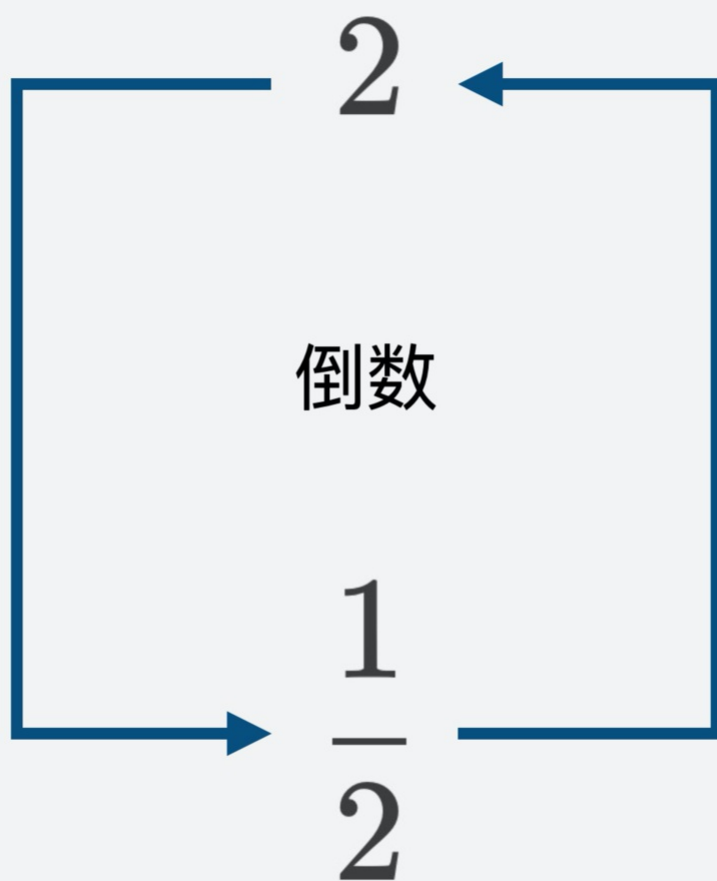
注意，这里的行和列不同， $m \neq n$ 意味着，根据矩阵乘，左乘和右乘单位矩阵也不同，也就是 $\mathbf{I}_m \neq \mathbf{I}_n$ 。

逆矩阵与转置矩阵

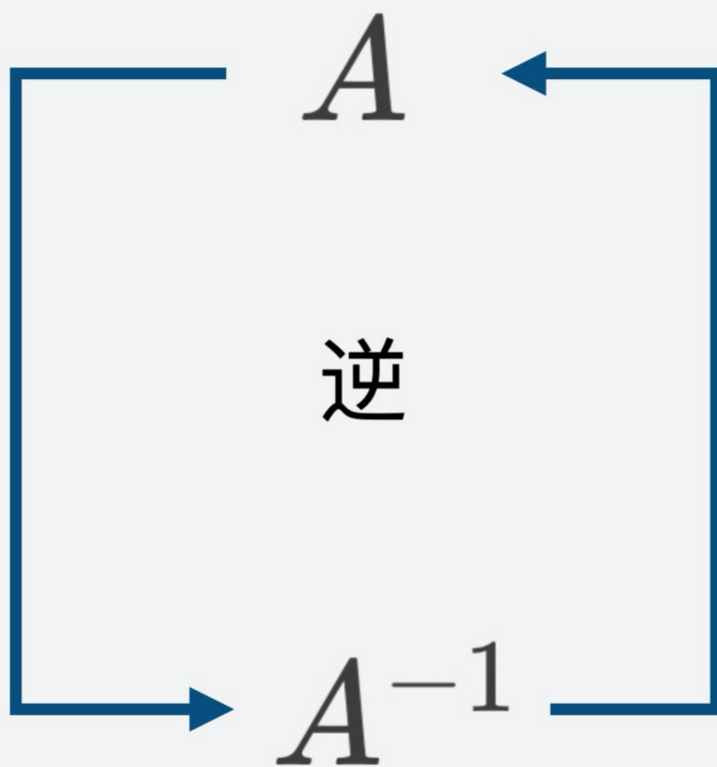
了解矩阵基本概念、运算，以及性质后，我来讲一讲矩阵应用中的两个核心内容——逆矩阵和转置矩阵。逆矩阵和转置矩阵在实际应用中大有用处，比如：坐标系中的图形变换运算。我们先来看下什么是逆矩阵。

逆矩阵

下面这个图你应该非常熟悉了，图中表现的是数字的倒数，2的倒数是 $\frac{1}{2}$ ， $\frac{1}{2}$ 的倒数是2。



其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```
SS
A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\left[\begin{array}{cc}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{array}\right]
SS
```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```
SS
A \times A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]
SS
A \times A^{-1}=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right]
SS
```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人？

首先，我们设置一些矩阵，组成线性方程\$XA=BS\$。

小孩          大人          大巴          火车

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 3 & 3.5 \\ 3.2 & 3.6 \end{bmatrix}$$

要解\$XS\$，我们就要先计算\$A\$的逆矩阵\$A^{-1}\$：

```
$$
A^{-1}=\left[\begin{array}{cc}
3 & 3.5 \\
3.2 & 3.6
\end{array}\right]^{-1}=\frac{1}{3\times 3.6-3.5\times 3.2}\left[\begin{array}{cc}
3.6 & -3.5 \\
-3.2 & 3
\end{array}\right]
=\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
$$
```

接下来再计算\$X=BA^{-1}\$：

```
$$
\left[\begin{array}{ll}
x_1 & x_2
\end{array}\right]=\left[\begin{array}{ll}
118.4 & 135.2
\end{array}\right]\left[\begin{array}{cc}
-9 & 8.75 \\
8 & -7.5
\end{array}\right]
=\left[\begin{array}{ll}
16 & 22
\end{array}\right]
$$
```

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least，方程次序很重要，也就是说，\$AX=BS\$和\$XA=BS\$的结果是不同的，这个一定要牢记哦！

转置矩阵

一般件随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把\$m\times n\$矩阵\$A\$的行列互换，得到转置矩阵\$A^T\$。

```
$$
A=\left[\begin{array}{cccc}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\ldots & \ldots & \ldots & \ldots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{array}\right]
$$
```

```
$$
A^T=\left[\begin{array}{cccc}
a_{11} & a_{21} & \ldots & a_{m1} \\
a_{12} & a_{22} & \ldots & a_{m2} \\
\ldots & \ldots & \ldots & \ldots \\
a_{1n} & a_{2n} & \ldots & a_{mn}
\end{array}\right]

```

\end{array}\right]

SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

1. 矩阵和自身逆矩阵相乘得单位矩阵， $\mathbf{A} \mathbf{A}^{-1} = \mathbf{I} = \mathbf{A}^{-1} \mathbf{A}$ ；
2.  $\mathbf{A} \mathbf{B}$  两矩阵相乘的逆，等于逆矩阵  $\mathbf{B}$  和逆矩阵  $\mathbf{A}$  相乘，这里强调一下乘的顺序很重要， $(\mathbf{A} \mathbf{B})^{-1} = \mathbf{B}^{-1} \mathbf{A}^{-1}$ ；
3.  $\mathbf{A} \mathbf{B}$  两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $(\mathbf{A} + \mathbf{B})^{-1} \neq \mathbf{A}^{-1} + \mathbf{B}^{-1}$ ；
4. 矩阵转置的转置还是它本身， $\left(\mathbf{A}^{\mathrm{T}}\right)^{\mathrm{T}} = \mathbf{A}$ ；
5.  $\mathbf{A} \mathbf{B}$  两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $(\mathbf{A} + \mathbf{B})^{\mathrm{T}} = \mathbf{A}^{\mathrm{T}} + \mathbf{B}^{\mathrm{T}}$ ；
6.  $\mathbf{A} \mathbf{B}$  两矩阵相乘后的转置矩阵，等于转置矩阵  $\mathbf{B}$  和转置矩阵  $\mathbf{A}$  的相乘，这里再次强调乘的顺序很重要， $(\mathbf{A} \mathbf{B})^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}} \mathbf{A}^{\mathrm{T}}$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $\mathbf{x} = \left(x_1, \dots, x_{10}\right)^{\mathrm{T}}$ ， $\mathbf{S} = \left(v_1, \dots, v_{10}\right)^{\mathrm{T}}$ ，如果要计算 $\mathbf{y} = \mathbf{x}^{\mathrm{T}} \left(\mathbf{I} + \mathbf{v} \mathbf{v}^{\mathrm{T}}\right) \mathbf{x}$ ，其中 $\mathbf{I}$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

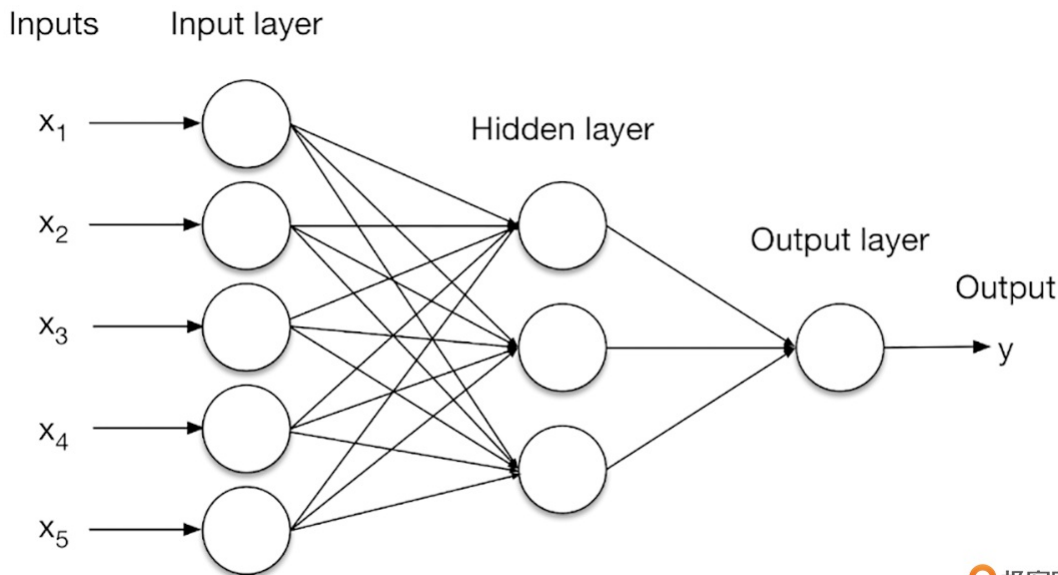
欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“矩阵”。

在开始学习之前，我想先问你个问题，你觉得，学习矩阵有什么用呢？你可以先自己想一想。之后我们讲任何一个知识的时候，你都可以从这个角度出发，自己先思考一下，这样有助于你对所学内容理解得更深刻。

对于刚才那个问题，我的答案很简单，就一句话，从我们程序员的角度去理解的话，矩阵可以极大地提高计算机的运算效率。怎么说呢？我给你举一个例子。在机器学习中（特别是深度学习，或者更具体一点，神经网络），并行计算是非常昂贵的。





极客时间

上图是一个典型的神经网络架构，在这时候，矩阵就能发挥用武之地了，计算 $S$ 隐藏层输出的公式是： $S = f(W \cdot x + b)$ ，其中 $W$ 是权重矩阵， $f$ 是激活函数， $b$ 是偏差， $x$ 是输入层矩阵。而这个计算过程就叫做向量化（Vectorization），这也是GPU在深度学习中非常重要的原因，因为GPU非常擅长做类似矩阵乘之类的运算。

```


$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$


$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$


$$H = f \left( \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$


```

不过，矩阵也不仅仅局限于神经网络的应用，同时它也可以用在计算机图形图像的应用中，比如，三维物体从取景到屏幕的显示，就需要经历一系列的空间变换，才能生成二维图像显示在显示器上。在这个计算过程中，我们都需要用到矩阵。

矩阵是非常实用的，但它正式作为数学中的研究对象出现，其实是在行列式的研究发展起来之后。英国数学家 Arthur Cayley 被认为矩阵论的创立人，他提出的矩阵概念可能来自于行列式。但我相信另一种说法，提出矩阵是为了更简单地表达线性方程组，也就是说，矩阵是线性方程组的另一种表达。

## 矩阵的基本概念

线性方程组的概念很简单，上节我们已经简单提过。你在小学或中学肯定也学过二元一次方程和二元一次方程组。

```


$$ax + by = c$$


$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$


```

在这样一个方程组中， $a_1$ 、 $a_2$ 、 $b_1$ 、 $b_2$ 不能同时为0。当我们把二元一次方程组再扩展一下，变成多元一次方程组时，我们就能得到线性方程组的一般表达，即 $AX=BS$ 。

```


$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$


```

于是，这个线性方程组的所有系数就构成了一个 $m \times n$ 的 $m$ 行 $n$ 列矩阵：

```


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


```

我们把 $A$ 称为该方程组的系数矩阵，而当我们把等式右边的常数 $B$ 放入矩阵后，就是下面这样：

```


$$AX = B$$


```

```
\widetilde{A}=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} & b_{1} \\a_{21} & a_{22} & \ldots & a_{2n} & b_{2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn} & b_{m}\end{array}\right]
```

这样我们就得到了\$A\$矩阵的增广矩阵 $\widetilde{A}$ ， 可以表示为\$(A, B)\$，这里的\$B\$表示的是方程组常数项所构成的列向量，也就是 $m\times 1$ 的 $m$ 行\$1\$列矩阵：

```
B=\left[\begin{array}{l}b_{1} \\b_{2} \\ \cdots \\b_{m}\end{array}\right]
```

如果设\$X\$为 $n\times 1$ 的 $n$ 行\$1\$列矩阵：

```
X=\left[\begin{array}{l}x_{1} \\x_{2} \\ \cdots \\x_{n}\end{array}\right]
```

那么线性方程组\$A\$X=\$B\$，就可以表示为\$AX=B\$的矩阵形式。如果我们再换一种表示形式， 设：\$a\_{11},a\_{12},\ldots,a\_{1n},\beta\$表示增广矩阵 $\widetilde{A}$ 的列向量，则线性方程组\$A\$X=\$B\$又可表示为\$a\_{11}x\_{1}+a\_{12}x\_{2}+\cdots+a\_{1n}x\_{n}=\beta\$。

线性方程组的矩阵和向量形式都是线性方程组的其他表达形式。在工作中， 你可以用它们来简化求解， 甚至可以提升计算效率， 就如之前提到的神经网络的隐藏层的输出计算、图形图像的三维空间变换。在数学中也是同样的， 你可以经常运用它们来简化求解。具体线性方程组求解的内容比较多， 我们下一节课再来详细讲解求解过程。

通过前面的讲解，我相信你对矩阵有了一定的了解， 现在我们再回头来看看矩阵的定义吧。

矩阵的定义是： 一个\$(m, n)\$矩阵\$A\$， 是由 $m\times n$ 个元素组成，  $m$ 和 $n$ 是实数， 其中元素\$a\_{ij}, i=1, \ldots, m, j=1, \ldots, n\$按\$m\$行 $n$ 列的矩形排布方式后可以形成矩阵\$A\$：

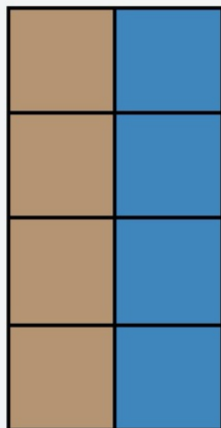
```
A=\left[\begin{array}{cccc}a_{11} & a_{12} & \ldots & a_{1n} \\a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\a_{m1} & a_{m2} & \ldots & a_{mn}\end{array}\right]
```

其中\$a\_{ij}\$属于实数或复数， 在我们的场景中是实数\$R\$， 按通常的惯例，  $(1, n)$ 矩阵叫做行，  $(m, 1)$ 矩阵叫做列， 这些特殊的矩阵叫做行或列向量。

定义完矩阵后，我接着讲一个比较有趣的概念，矩阵转换（Matrix transformation）。矩阵转换经常被用在计算机图形图像的转换中， 比如， 一张彩色图片从RGB角度来说说是三维的， 如果要转换成灰度图片， 也就是一维图片， 那就要做矩阵转换。

我们来看一下矩阵转换的过程。设 $R^{m\times n}$ 是实数矩阵\$(m, n)\$的集合，  $A\in R^{m\times n}$ 可以表示成另一种形式  $A\in R^{mm}$ 。我们把矩阵的 $n$ 列堆叠成一个长向量后完成转换。这个转换也叫做reshape， 其实就是重新调整原矩阵的行数、列数和维数， 但是元素个数不变。

$$A \in \mathbb{R}^{4 \times 2}$$



转换



a

## 矩阵的运算

了解了矩阵的基本定义后，我们才能进入矩阵的运算环节，就是矩阵的加和乘。

加运算很简单，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{m \times n}$ 的加运算其实就是矩阵各自元素的加。

```

$$
A+B=\left[\begin{array}{ccc}
a_{11}+b_{11} & \dots & a_{1n}+b_{1n} \\
\vdots & \ddots & \vdots \\
\vdots & \ddots & \vdots \\
a_{m1}+b_{m1} & \dots & a_{mn}+b_{mn}
\end{array}\right] \in \mathbb{R}^{m \times n}
$$

```

我推荐你使用NumPy的einsum来高效地做这类运算，因为它在速度和内存效率方面通常可以超越我们常见的array函数。

```
C= np.einsum('il, lj', A, B)
```

接下来，我们一起来看看矩阵的乘。这里你需要注意，矩阵的乘和通常意义上“数之间的乘”不同，矩阵的乘有多种类型，这里我讲三种最普遍，也是在各领域里用得最多的矩阵乘。

### 1.普通矩阵乘

普通矩阵乘是应用最广泛的矩阵乘，两个矩阵 $A \in \mathbb{R}^{m \times n}$ ， $B \in \mathbb{R}^{n \times k}$ ，普通矩阵则乘可以表示为 $C=A B \in \mathbb{R}^{m \times k}$ ， $C$ 中元素的计算规则是矩阵 $A$ 、 $B$ 对应两两元素乘积之和。

```

$$
c_{ij}=\sum_{k=1}^n a_{ik} b_{kj}, i=1, \dots, m, j=1, \dots, l
$$

```

我们举例来说明。 $C$ 的第一个元素 $c_{11}=a_{11} \times b_{11}+a_{12} \times b_{21}+a_{13} \times b_{31}=1 \times 1+2 \times 2+3 \times 3$ 。

```

$$
C=A B=\left[\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right]\left[\begin{array}{ccc}
1 & 4 \\
2 & 5 \\
3 & 6
\end{array}\right]=\left[\begin{array}{ccc}
1 \times 1+2 \times 2+3 \times 3 & 1 \times 4+2 \times 5+3 \times 6 \\
4 \times 1+5 \times 2+6 \times 3 & 4 \times 4+5 \times 5+6 \times 6
\end{array}\right]=\left[\begin{array}{cc}
14 & 32 \\
32 & 77
\end{array}\right]
$$

```

这里需要特别注意的是，只有相邻阶数匹配的矩阵才能相乘，例如，一个 $n \times k$ 矩阵 $A$ 和一个 $k \times m$ 矩阵 $B$ 相乘，最后得出 $n \times m$ 矩阵 $C$ ，而这里的 $k$ 就是相邻阶数。

\$\$AB=CS\$

但反过来B和A相乘就不行了，因为相邻阶数 $S_mS$ 不等于 $S_nS$ 。

2.哈达玛积

哈达玛积理解起来就很简单了，就是矩阵各对应元素的乘积， $c_{c_{ij}}=a_{ij} \times b_{ij}$ 。举个例子：

```
$$
C=A^{(*)} B=\left[\begin{array}{ll}
1 & 2 \\
4 & 5
\end{array}\right]\left[\begin{array}{ll}
1 & 4 \\
2 & 5
\end{array}\right]=\left[\begin{array}{cc}
1 * 1 & 2 * 4 \\
4 * 2 & 5 * 5
\end{array}\right]=\left[\begin{array}{cc}
1 & 8 \\
8 & 25
\end{array}\right]
$$
```

哈达玛积其实在数学中不常看到，不过，在编程中哈达玛积非常有用，因为它可以用来同时计算多组数据的乘积，计算效率很高。

3.克罗内克积

克罗内克积是以德国数学家利奥波德·克罗内克（Leopold Kronecker）的名字命名的。它可以应用在解线性矩阵方程和图像处理方面，当然从更时髦的角度说，它还能用在量子信息领域，我们也称之为直积或张量积。

和普通矩阵乘和哈达玛积不同的是，克罗内克积是两个任意大小矩阵间的运算，表示为 $A \times B$ ，如果 $A$ 是一个 $m \times n$ 的矩阵，而 $B$ 是一个 $p \times q$ 的矩阵，克罗内克积则是一个 $mp \times nq$ 的矩阵。

接下来我们需要定义一个在矩阵的乘法中起着特殊作用的矩阵，它就是单位矩阵。高等代数中，在求解相应的矩阵时，若添加单位矩阵，通过初等变换进行求解，往往可以使问题变得简单。按照百度百科的解释，单位矩阵如同数的乘法中的1，这种矩阵就被称为单位矩阵。它是个方阵，从左上角到右下角的对角线，也就是主对角线上的元素均为1，除此以外全都为0。

在线性代数中，大小为 $n$ 的单位矩阵就是在主对角线上均为1，而其他地方都是0的 $n \times n$ 的方阵，它用 $\mathbf{I}_n$ 表示，表达时为了方便可以忽略阶数，直接用 $\mathbf{I}$ 来表示：

```
$$
\mathbf{I}_1=[1], \mathbf{I}_2=\left[\begin{array}{ll}
1 & 0 \\
0 & 1
\end{array}\right], \mathbf{I}_3=\left[\begin{array}{lll}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right], \dots, \mathbf{I}_n=\left[\begin{array}{cccc}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
. & . & \dots & . \\
. & . & . & . \\
0 & 0 & \dots & 1
\end{array}\right]
$$
```

矩阵的性质

在了解了矩阵加和乘，以及单位矩阵后，我们是时候来看一看矩阵的性质了。了解矩阵的性质是进行矩阵计算的前提，就像我们小时候学加减乘除四则运算法则时那样。所以，这块内容对你来说应该不难，你作为了解就好，重点是之后的运算。

1.结合律

任意实数 $m \times n$ 矩阵 $A$ ， $n \times p$ 矩阵 $B$ ， $p \times q$ 矩阵 $C$ 之间相乘，满足结合律 $(AB)C=A(BC)$ 。这个很好理解，我就不多说了。

\$\$\forall a \in R^{\{m \times n\}}, B \in R^{\{n \times p\}}, C \in R^{\{p \times q\}} \{ (A B) C = A (B C) \$\$

2.分配律

任意实数 $m \times n$ 矩阵 $A$ 和 $B$ ， $n \times p$ 矩阵 $C$ 和 $D$ 之间相乘满足分配律 $(A+B)C=AC+BC$ ， $A(C+D)=AC+AD$ 。

\$\$\forall a \in \mathbf{R}^{\{m \times n\}}, B \in \mathbf{R}^{\{n \times p\}}, C, D \in \mathbf{R}^{\{n \times p\}} \{ (A+B) C = A C + B C, A (C+D) = A C + A D \$\$

3.单位矩阵乘

任意实数 $m \times n$ 矩阵A和单位矩阵之间的乘，等于它本身 $A$ 。

\$\$\forall a \in R^{\{m \times n\}}: \mathbf{I}\_m A = A \mathbf{I}\_n = A \$\$

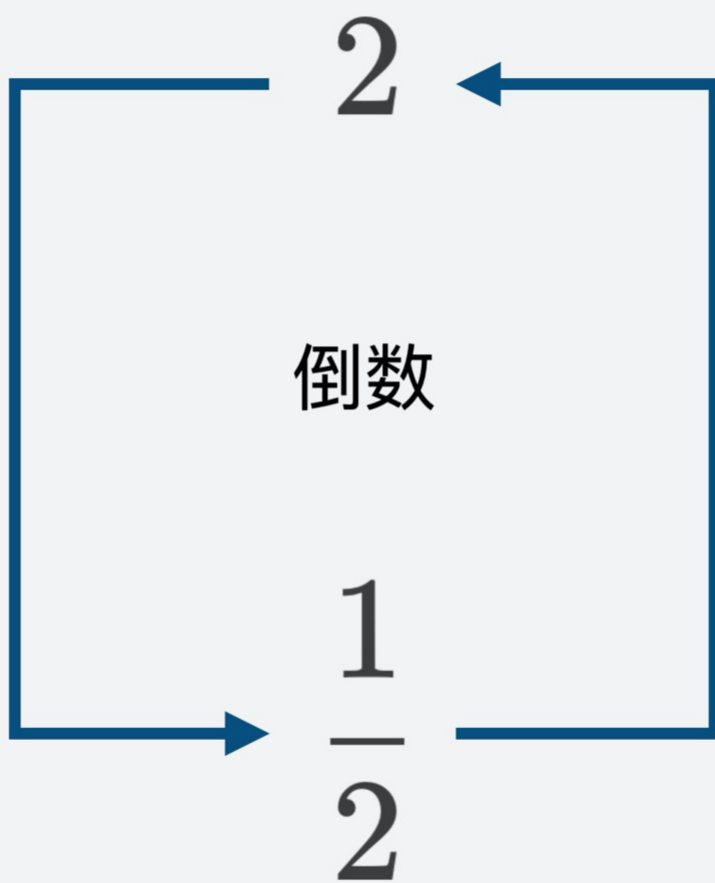
注意，这里的行和列不同， $m \neq n$ 意味着，根据矩阵乘，左乘和右乘单位矩阵也不同，也就是 $\mathbf{I}_m \neq \mathbf{I}_n$ 。

逆矩阵与转置矩阵

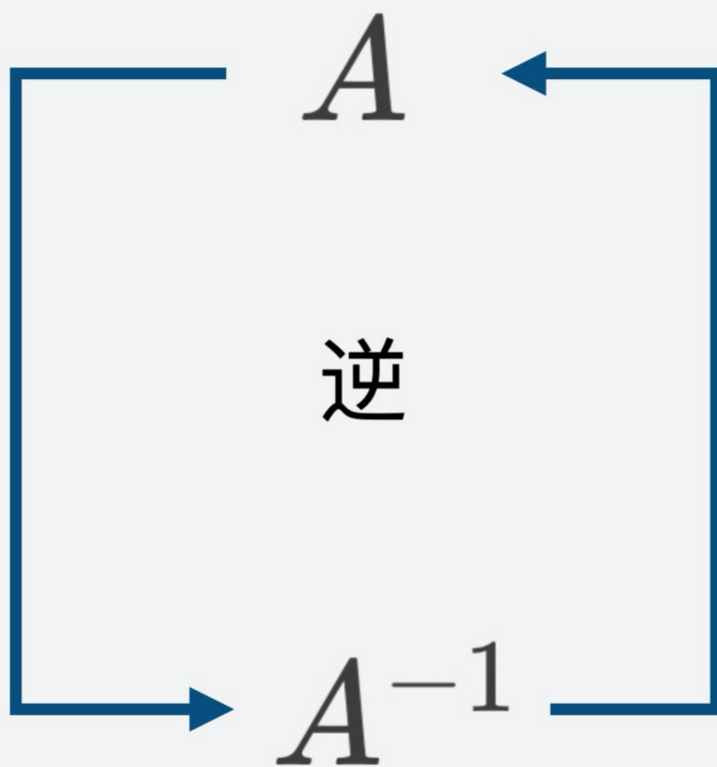
了解矩阵基本概念、运算，以及性质后，我来讲一讲矩阵应用中的两个核心内容——逆矩阵和转置矩阵。逆矩阵和转置矩阵在实际应用中大有用处，比如：坐标系中的图形变换运算。我们先来看下什么是逆矩阵。

逆矩阵

下面这个图你应该非常熟悉了，图中表现的是数字的倒数，2的倒数是 $\frac{1}{2}$ ， $\frac{1}{2}$ 的倒数是2。



其实逆矩阵也有着类似的概念，只不过是写法不一样，我们会把逆矩阵写成 $A^{-1}$ 。那为什么不是 $\frac{1}{A}$ 呢？那是因为数字1无法被矩阵除。



我们知道， $S$ 乘以它的倒数 $\frac{1}{S}$ 等于 $1$ 。同样的道理， $SA$ 乘以它的逆矩阵 $A^{-1}$ 就等于单位矩阵，即 $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ （ $\mathbf{I}$ 即单位矩阵），反过来也一样， $\mathbf{A}^{-1} \times \mathbf{A} = \mathbf{I}$ 。

为方便你理解，我用一个 $2 \times 2$ 矩阵 $A$ 来解释一下逆矩阵的算法。首先，我们交换 $a_{11}$ 和 $a_{22}$ 的位置，然后在 $a_{12}$ 和 $a_{21}$ 前加上负号，最后除以行列式 $a_{11}a_{22} - a_{12}a_{21}$ 。

```
SS
A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\frac{1}{a_{11}a_{22}-a_{12}a_{21}}\left[\begin{array}{cc}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{array}\right]
SS
```

那我们该如何验证这不是正解呢？

方法其实很简单，记得刚才的公式就行， $\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$ 。现在我们就代入公式来验证一下， $A$ 和它的逆矩阵相乘，通过刚才的算法最终得出的结果是单位矩阵。

```
SS
A \times A^{-1}=\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]\left[\begin{array}{ll}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}\right]^{-1}=\left[\begin{array}{ll}
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}} \\
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}
\end{array}\right]
SS
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{11}+\frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{21}=\frac{a_{11}a_{22}-a_{12}a_{21}}{a_{11}a_{22}-a_{12}a_{21}}=1
\frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{11}+\frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{22}=\frac{-a_{11}a_{12}+a_{11}a_{22}}{a_{11}a_{22}-a_{12}a_{21}}=\frac{a_{11}(a_{22}-a_{12})}{a_{11}a_{22}-a_{12}a_{21}}=\frac{a_{11}}{a_{11}}=1
\frac{-a_{21}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{11}+\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{21}=\frac{-a_{11}a_{21}+a_{21}a_{22}}{a_{11}a_{22}-a_{12}a_{21}}=\frac{a_{21}(a_{22}-a_{12})}{a_{11}a_{22}-a_{12}a_{21}}=\frac{a_{21}}{a_{11}}\times\frac{a_{11}}{a_{11}}=\frac{a_{21}}{a_{11}}\times\frac{a_{11}}{a_{11}}=1
\frac{a_{11}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{12}+\frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{22}=\frac{a_{11}a_{12}-a_{12}a_{22}}{a_{11}a_{22}-a_{12}a_{21}}=\frac{a_{12}(a_{11}-a_{22})}{a_{11}a_{22}-a_{12}a_{21}}=\frac{a_{12}}{a_{11}}\times\frac{a_{11}-a_{22}}{a_{11}}=\frac{a_{12}}{a_{11}}\times\frac{a_{11}}{a_{11}}-\frac{a_{12}}{a_{11}}\times\frac{a_{22}}{a_{11}}=\frac{a_{12}}{a_{11}}-\frac{a_{12}a_{22}}{a_{11}a_{22}}=\frac{a_{12}}{a_{11}}-\frac{a_{12}}{a_{11}}=0
\frac{a_{22}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{12}+\frac{-a_{12}}{a_{11}a_{22}-a_{12}a_{21}}\times a_{22}=\frac{a_{11}a_{22}+a_{12}a_{22}}{a_{11}a_{22}-a_{12}a_{21}}=\frac{a_{22}(a_{11}+a_{12})}{a_{11}a_{22}-a_{12}a_{21}}=\frac{a_{22}}{a_{11}}\times\frac{a_{11}+a_{12}}{a_{11}}=\frac{a_{22}}{a_{11}}\times\frac{a_{11}}{a_{11}}+\frac{a_{22}}{a_{11}}\times\frac{a_{12}}{a_{11}}=\frac{a_{22}}{a_{11}}+\frac{a_{12}a_{22}}{a_{11}a_{22}}=\frac{a_{22}}{a_{11}}+\frac{a_{12}}{a_{11}}=\frac{a_{22}+a_{12}}{a_{11}}=\frac{a_{22}}{a_{11}}+\frac{a_{12}}{a_{11}}=1
SS
```

这里有一点需要特别说明，不是每一个矩阵都是可逆的。如果一个矩阵是可逆的，那这个矩阵我们叫做**非奇异矩阵**，如果一个矩阵是不可逆的，那这个矩阵我们就叫做**奇异矩阵**，而且如果一个矩阵可逆，那它的逆矩阵必然是唯一的。

还记得行列式 $a_{11}a_{22} - a_{12}a_{21}$ 吗？如果我们要证明矩阵是可逆的，只要证明行列式不等于零就行。更高阶的逆矩阵的算法也是一样的原理。

最后，我想通过一个现实生活中的案例来让你更多地了解逆矩阵。

一个旅游团由孩子和大人组成，去程他们一起做大巴，每个孩子的票价\$3\$元，大人票价\$3.2\$元，总共花费\$118.4\$元。回程一起做火车，每个孩子的票价\$3.5\$元，大人票价\$3.6\$元，总共花费

\$135.2\$元。请问旅游团里有多少小孩和大人?

首先，我们设置一些矩阵，组成线性方程 $XA=B$ 。

小孩	大人	大巴	火车
$x_1$	$x_2$	3	3.5
		3.2	3.6

要解 $\mathbf{X}\mathbf{S}$ ，我们就要先计算 $\mathbf{A}\mathbf{S}$ 的逆矩阵 $\mathbf{A}^{-1}\mathbf{S}$ ：

$$\frac{A^{-1}}{3 \wedge 3.5} \left( \frac{3 \wedge 3.6}{3.6 \wedge -3.5} \right)^{-1} = \frac{1}{3 \wedge 3.6 \wedge 3.5 \wedge 3.2} \left( \frac{3.2 \wedge 3}{-9 \wedge 8.75} \right)^{-1}$$

接下来再计算  $X = B A^{-1} S$ :

$$\begin{aligned} & \left\lfloor \begin{array}{l} \text{begin}\{array\}\{II\} \\ x_1\} \text{ and } x_2\} \\ \text{end}\{array\}\text{right} \end{array} \right\rfloor = \left\lfloor \begin{array}{l} \text{begin}\{array\}\{II\}, \\ 118.4 \text{ and } 135.2 \\ \text{end}\{array\}\text{right} \end{array} \right\rfloor \left\lfloor \begin{array}{l} \text{begin}\{array\}\{cc\} \\ -9 \text{ and } 8.75 \\ \text{end}\{array\}\text{right} \end{array} \right\rfloor \\ & \left\lfloor \begin{array}{l} 8 \text{ and } -7.5 \\ \text{end}\{array\}\text{right} \end{array} \right\rfloor = \left\lfloor \begin{array}{l} \text{begin}\{array\}\{II\}, \\ 16 \text{ and } 22 \\ \text{end}\{array\}\text{right} \end{array} \right\rfloor \end{aligned}$$

最终，我们得出这个旅游团有16个小孩和22个大人。

这也是解线性方程组的一种方法，类似这样的计算被广泛应用在各领域中，比如建筑工程、游戏和动画的3D效果上。虽然现在有很多程序包封装了这类数学计算的底层实现，但如果你能很好地理解这些概念，就可以为编程或算法调优打下坚实的基础。

Last but not least, 方程次序很重要, 也就是说,  $SAX=BS$  和  $SXA=BS$  的结果是不同的, 这个一定要牢记哦!

转置矩阵

一般伴随逆矩阵之后出现的就是转置矩阵。在计算机图形图像处理中，如果要对一个物体进行旋转、平移、缩放等操作，就要对描述这个物体的所有矩阵进行运算，矩阵转置就是这类运算之一，而矩阵的转置在三维空间中的解释就相当于“得到关于某个点对称的三维立体”。所以，转置矩阵的定义很简单。

将矩阵的行列互换，得到的新矩阵就叫做转置矩阵（transpose）。转置矩阵的行列式不变。我们把  $m \times n$  矩阵  $A$  的行列互换，得到转置矩阵  $A^T$ 。

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

\end{array}'\right]
SS

最后，为了方便你理解，我们再总结一下逆矩阵和转置矩阵的性质。你不用死记硬背，重在理解。

1. 矩阵和自身逆矩阵相乘得单位矩阵， $SA\ A^{-1}=I=A^{-1}\ A$ ;
2.  $SASBS$ 两矩阵相乘的逆，等于逆矩阵 $SBS$ 和逆矩阵 $SAS$ 相乘，这里强调一下乘的顺序很重要， $S(A\ B)^{-1}=B^{-1}\ A^{-1}$ ;
3.  $SABS$ 两矩阵相加后的逆矩阵，不等于各自逆矩阵的相加， $S(A+B)^{-1}\neq A^{-1}+B^{-1}$ ;
4. 矩阵转置的转置还是它本身， $S\left(A^T\right)^{\mathrm{T}}=A$ ;
5.  $SABS$ 两矩阵相加后的转置矩阵，等于各自转置矩阵的相加， $S(A+B)^T=A^T+B^T$ ;
6.  $SABS$ 两矩阵相乘后的转置矩阵，等于转置矩阵 $B$ 和转置矩阵 $A$ 的相乘，这里再次强调乘的顺序很重要， $S(A\ B)^T=B^T\ A^T$ 。

本节小结

好了，到这里矩阵这一讲就结束了，最后我再带你总结一下前面讲解的内容。

今天的知识，你只需要知道矩阵是线性方程组的另一种表达，了解和掌握矩阵的定义和性质就足够了。当然，矩阵还有很多内容，但我认为掌握了我讲的这些内容后，就为以后的一些矩阵应用场景打下了坚实的数学基础，也是下一讲的解线性方程组的前置知识。



线性代数练习场

对于10维列向量 $Sx=\left(x_{1}, \ldots, x_{10}\right)^T S, \quad S=\left(v_{1}, \ldots, v_{10}\right)^T$ ，如果要计算 $Sy=x^T\left(I+v v^T\right) x S$ ，其中 $S$ 是10阶单位矩阵。你会怎么做？

友情提醒，这里有多种方式解题。你能不能找到一个最简单的方法来解决这道题？虽然结果很重要，但我想说的是过程更重要，而且往往解题过程不同，从计算机角度来说，运算的效率会有极大的不同。

欢迎你在留言区晒出你的运算过程和结果。如果有收获，也欢迎你把这篇文章分享给你的朋友。