3 线性代数

> **Important**
>
> **B站视频链接05 线性代数动手学深度学习v2**

1 线性代数基本知识

1.1 标量

标量在之前的学习过程中便已经知道，只是一些所谓的数据而已，譬如`1,2,3`之类的单个元素，同我们小时候学习的并无任何的不同。同样满足普通数学的四则运算法则，注意哈标量是不带箭头的哈！！

现在看一下标量支持的一些普通运算法则

1.1.1 基本运算

普通元素的加法：$c = a + b$

普通元素的减法：$c = a - b$

普通元素的乘法：$c = a \times b$

普通元素的除法：$c = a \div b$

1.1.2 长度运算

取绝对值 : $|a| = \begin{cases} a & a \geq 0 \\ -a & a \leq 0 \end{cases}$

绝对值不等式: $|a + b| \leq |a| + |b|$

绝对值 乘法：$|a \cdot b| = |a| \cdot |b|$

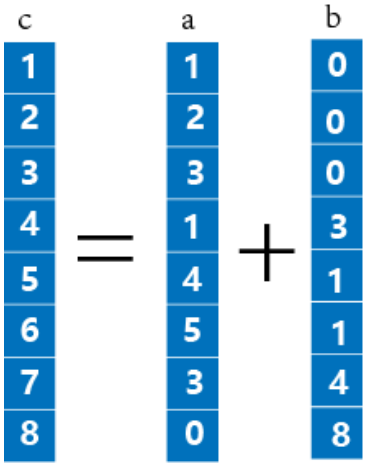
1.2 向量

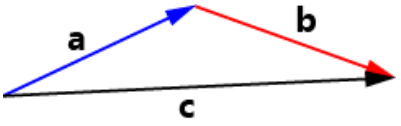
向量相对于标量而言多了一个维度的空间，向量是一个有方向和长度的东西，而标量只有长度没有方向，也可以说向量是由许许多多的标量组合而成的。

向量和小时候学习的数据是完全不相同的，它满足加减但不支持乘除，向量的乘法和除法需要满足前行后列的要求才可以对应相乘，即向量的内积。

1.2.1 基本运算

- 普通向量的加法：$\vec{c} = \vec{a} + \vec{b} \rightarrow c_i = a_i + b_i$





- 普通向量的减法：$\vec{c} = \vec{a} - \vec{b} \rightarrow c_i = a_i - b_i$ 其表现形式和加法如出一辙便不进行适当的演示了。
- 普通向量的函数：$\vec{c} = sin\vec{a}$ 这就相当于对向量里面的每一个元素都进行了sin函数的调用，即$c_i = sina_i$
- 向量和标量的相乘:$\vec{c} = a \times \vec{\alpha}$ 这就相当于将该向量的长度按照a倍进行缩放同时不改变其原本向量的方向。

- 向量之间的点乘：$\vec{\alpha}^T \cdot \vec{\beta} = \sum_i \alpha_i \cdot \beta_i$ 不难看出向量之间的点乘是有一定要求的这里我们假设向量$\vec{\alpha}$和$\vec{\beta}$的形状均为m行1列即$m \times 1$。现在将$\vec{\alpha}$进行转置则现在他的形状为$1 \times m$，此时这两个向量才可以进行相乘$(1 \times m) \cdot (m \times 1) = 1 \times 1$最终得到一个单位的数据，这就是向量之间的点乘。
- 向量点乘如果等于零:$\vec{\alpha}^T \cdot \vec{\beta} = 0$，此时可以说明向量$\vec{\alpha}$和向量$\vec{\beta}$相互正交。

1.2.2 长度运算

- 计算长度：$||\vec{a}||_2 = \sqrt{\sum_{i=1}^{m} a_i^2}$ 也就是对每个元素平方求和后开个方的事情
- 绝对值不等式(也可以说是三角形两边之和大于第三边定理):$||\vec{a} + \vec{b}|| \leq ||\vec{a}|| + ||\vec{b}||$
- 绝对值的乘法法则：$||\vec{a} \cdot \vec{b}|| = ||\vec{a}|| \cdot ||\vec{b}||$

1.3 矩阵

矩阵是一个非常重要的研究对象对于线性代数而言，同时在考研数学中线性代数主要研究的也是各个矩阵的变换等等，因此矩阵的计算以及研究在深度学习的领域也非常重要的。

在这里我们需要理解，矩阵对于空间变换的影响、矩阵的特征值和特征向量、矩阵的乘法、以及一些特殊矩阵的性质等等问题

1.3.1 基本运算

- 矩阵的加法：$C = A + B \rightarrow C_{ij} = A_{ij} + B_{ij}$
- 矩阵的减法：$C = A - B \rightarrow C_{ij} = A_{ij} - B_{ij}$

- 矩阵的乘法：

$$C = A \cdot B \rightarrow \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{3} A_{1i} \times B_{i1} & \sum_{i=1}^{3} A_{1i} \times B_{i2} & \sum_{i=1}^{3} A_{1i} \times B \\ \sum_{i=1}^{3} A_{2i} \times B_{i1} & \sum_{i=1}^{3} A_{2i} \times B_{i2} & \sum_{i=1}^{3} A_{2i} \times B \\ \sum_{i=1}^{3} A_{3i} \times B_{i1} & \sum_{i=1}^{3} A_{3i} \times B_{i2} & \sum_{i=1}^{3} A_{3i} \times B \end{bmatrix}$$

  矩阵乘法遵守前行后列的要求。

- 矩阵的函数：$C = sinA \rightarrow C_{ij} = sinA_{ij}$

1.3.2 矩阵性质

- 正交矩阵：$A \cdot A^T = 1$，并且其举证内部的任意一行均正交
- 特征值和特征向量：$A\vec{x} = \lambda\vec{x}$，无论矩形A如何进行处理总有一个方向$\vec{x}$不会发生方向的变化仅仅只会发生长度的改变。同时实对称矩阵必定有特征向量。

矩阵这一部分比较难懂因此可以在网络上面多多学习，我这里推荐知名的数学博主$3Blue1Brown$的视频进行学习

视频链接：线性代数的本质

---

2 线性代数的代码实现

> 注意下面所有的代码能够运行的前提是，自己的计算机已经正确配置好了`pytorch`环境，如果环境没有配置好请一定要耐心配置好后在继续，环境配置视频。

环境配置完毕后需要引入自己的库，下面就不写这些头文件了

```
import torch
import matplotlib.pyplot as plt
import cv2 as cv
import numpy as np
```

```
In [22]: import torch
         import cv2 as cv
         import matplotlib.pyplot as plt
         import numpy as np
```

引入成功后便可以进行代码的编写啦！

2.1 数组的创建

2.1.1 标量的创建

标量就是一个数字也可以说是一个数据，这个的创建是依托下面的代码进行实现

```
x = torch.tensor([3.0]) #创建一个标量x
```

在编译器里不难看出他的`size`仅仅唯`1`

```
In [6]: x = torch.tensor([3.0])
        x , x.shape
Out[6]: (tensor([3.]), torch.Size([1]))
```

2.1.2 向量的创建

```
y = torch.arange(4) #创建一个向量 y
```

在编译器中可以看到他的size是`4`

```
In [14]: y = torch.arange(4)
         y , y.shape
Out[14]: (tensor([0, 1, 2, 3]), torch.Size([4]))
```

同时我们可以对向量中各个元素进行适当的索引或是提取的操作

```
y[0] , y[1] , y[2] , y[3] #逐个提取其中元素
```

下面是他的数组索引情况，他是从0号开始逐个进行索引的

```
In [16]: y[0] , y[1] , y[2] , y[3]
Out[16]: (tensor(0), tensor(1), tensor(2), tensor(3))
```

2.1.3 矩阵的创建

矩阵的创建类型非常的多，可以创建全`1`的矩阵、全`0`的矩阵、连续自然数的矩阵等等。

```
A = torch.ones(20).reshape(4,5)   #全1
B = torch.zeros(20).reshape(4,5)  #全0
C= torch.arange(20).reshape(4,5) #连续自然数
```

首先采用函数创建一个比较大的向量，然后将向量的形状进行修改变成矩阵即可完成创建

```
In [17]: A = torch.ones(20).reshape(4,5)
         B = torch.zeros(20).reshape(4,5)
         C = torch.arange(20).reshape(4,5)
         A , B , C
Out[17]: (tensor([[1., 1., 1., 1., 1.],
                  [1., 1., 1., 1., 1.],
                  [1., 1., 1., 1., 1.],
                  [1., 1., 1., 1., 1.]]),
          tensor([[0., 0., 0., 0., 0.],
                  [0., 0., 0., 0., 0.],
                  [0., 0., 0., 0., 0.],
                  [0., 0., 0., 0., 0.]]),
          tensor([[ 0,  1,  2,  3,  4],
                  [ 5,  6,  7,  8,  9],
                  [10, 11, 12, 13, 14],
                  [15, 16, 17, 18, 19]]))
```

2.2 线性代数的基本运算

2.2.1 标量的基本运算

由于标量的基本运算之前已经有过提到因此这次就不详细的进行说明仅仅将代码和运行截图列下。

```
a = torch.tensor([3.0])
b = torch.tensor([1.0])
c = a + b
d = a - b
e = a * b
f = a / b
```

```
In [19]:
a = torch.tensor([3.0])
b = torch.tensor([1.0])
c = a + b
d = a - b
e = a * b
f = a / b
a , b , c , d , e , f
```

```
Out[19]: (tensor([3.]),
 tensor([1.]),
 tensor([4.]),
 tensor([2.]),
 tensor([3.]),
 tensor([3.]))
```

2.2.2 向量的点乘

在上面说到，向量的点乘就是两个长度应该相同的两个向量进行对应元素相乘后相加，同时要求前一个向量是躺着的，后一个向量是竖着的。

```
x = torch.arange(5)
y = torch.arange(5)
z = torch.dot(x,y)
x , y , z
```

```
In [62]:
x = torch.arange(5)
y = torch.arange(5)
z = torch.dot(x,y)
x , y , z
```

```
Out[62]: (tensor([0, 1, 2, 3, 4]), tensor([0, 1, 2, 3, 4]), tensor(30))
```

如果想要矩阵进行转置则可以使用下面的代码命令进行实现

```
x.T
y.T
z.T
```

```
In [67]:
x , x.T ,  y , y.T , z , z.T
```

```
Out[67]: (tensor([0, 1, 2, 3, 4]),
 tensor([0, 1, 2, 3, 4]),
 tensor([0, 1, 2, 3, 4]),
 tensor([0, 1, 2, 3, 4]),
 tensor(30),
 tensor(30))
```

虽然图片上不大看的出来，但是确实是进行了转置，可以用矩阵进行查看

```
C , C.T
```

```
In [69]: C , C.T
```

```
Out[69]: (tensor([[ 0,  1,  2,  3,  4],
                  [ 5,  6,  7,  8,  9],
                  [10, 11, 12, 13, 14],
                  [15, 16, 17, 18, 19]]),
          tensor([[ 0,  5, 10, 15],
                  [ 1,  6, 11, 16],
                  [ 2,  7, 12, 17],
                  [ 3,  8, 13, 18],
                  [ 4,  9, 14, 19]]))
```

使用矩阵效果就非常的明显了。

2.2.3 矩阵和向量的乘法

矩阵和向量的乘法主要是下面的这几个代码，不过要理解的话需要阅读相对应的线性代数的教材才可以。

```
A = torch.arange(12).reshape(3,4)
B = torch.zeros(12).reshape(3,4)
C = torch.tensor([[1,2,3],[9,8,7],[2,5,3],[2,5,6]])
x = torch.arange(4)
y = torch.ones(4)
z = torch.zeros(4)

e1 = torch.mv(A,x)
e2 = torch.mm(A,C)

A , B , C ,x , y , z , e1 ,e2
```

- `mm`：矩阵 × 矩阵
- `mv`：矩阵 × 向量

In [83]:
```
A = torch.arange(12).reshape(3,4)
B = torch.zeros(12).reshape(3,4)
C = torch.tensor([[1,2,3],[9,8,7],[2,5,3],[2,5,6]])
x = torch.arange(4)
y = torch.ones(4)
z = torch.zeros(4)

e1 = torch.mv(A,x)
e2 = torch.mm(A,C)

A , B , C ,x , y , z , e1 ,e2
```

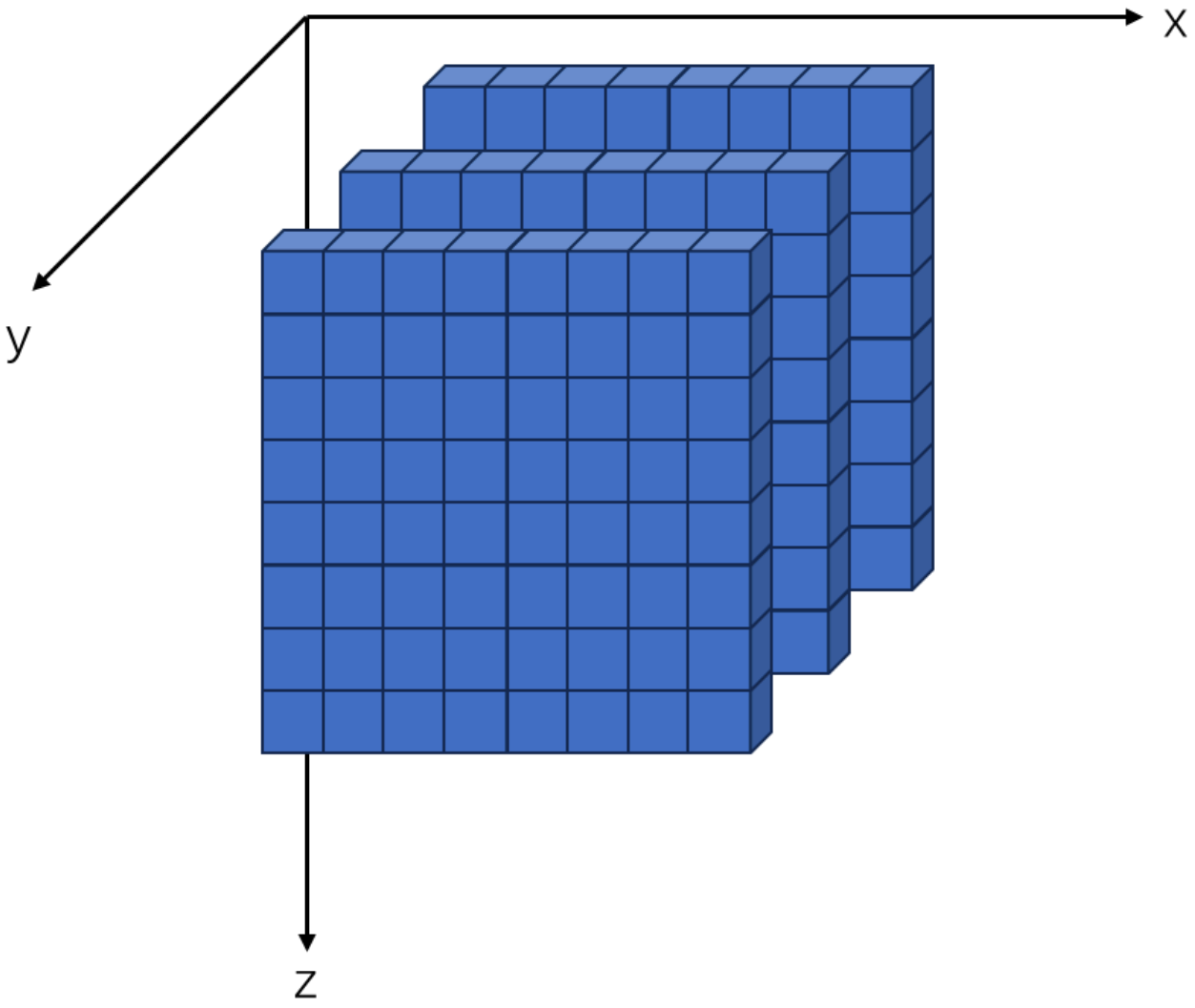
Out[83]:
```
(tensor([[ 0,  1,  2,  3],
         [ 4,  5,  6,  7],
         [ 8,  9, 10, 11]]),
 tensor([[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]]),
 tensor([[1, 2, 3],
         [9, 8, 7],
         [2, 5, 3],
         [2, 5, 6]]),
 tensor([0, 1, 2, 3]),
 tensor([1., 1., 1., 1.]),
 tensor([0., 0., 0., 0.]),
 tensor([14, 38, 62]),
 tensor([[ 19,  33,  31],
         [ 75, 113, 107],
         [131, 193, 183]]))
```

现在基本的一些代码都已经学的差不多了，现在开始矩阵按照某一固定轴进行求和

2.3 矩阵的按轴求和

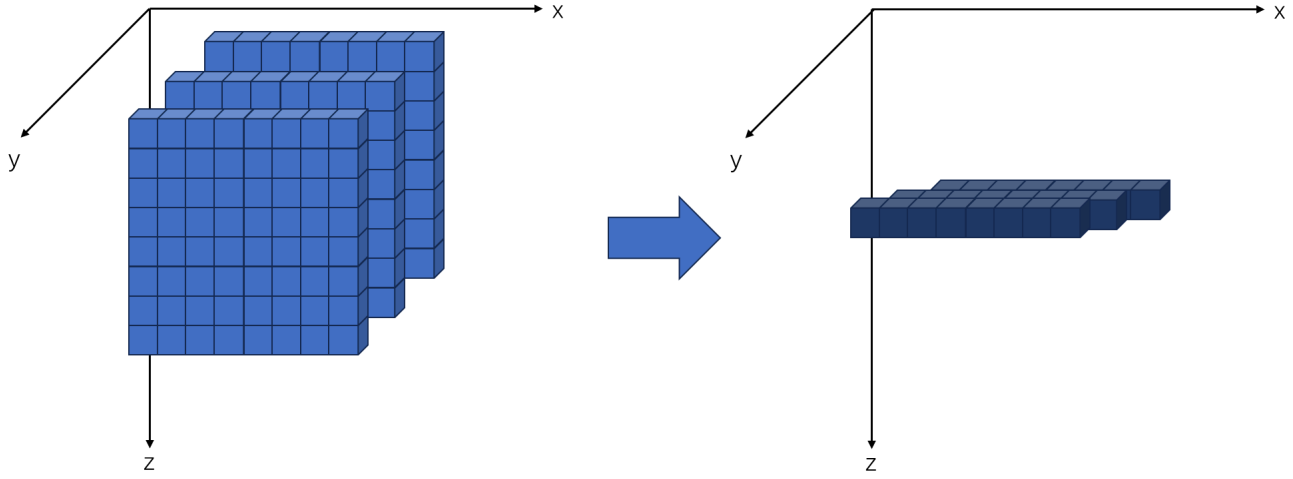
矩阵按照固定轴求和，可以看作是空间的压缩(坍塌)处理，以一个三维的数组为例子

假设有一个数组是一个$3 \times 8 \times 8$大小的矩阵，那么他总共有三个轴的方向分别为x轴、y轴、z轴，这三个轴就是所谓的按轴求和的那个轴，我们用图像来比较直观的表示一下下。



所谓的$axis = 0$或者$axis = 1$仅仅表示了第几个轴而已

假设我想要然他按照z轴进行求和，则和z轴有关其他直接向着z轴的方向累加



其他的方向也是如此，现在便可以进行代码的展示

```
A = torch.arange(24).reshape(2,3,4)
a0 =torch.sum(A,axis = 0)
a1 = torch.sum(A,axis = 1)
a2 = torch.sum(A,axis = 2)
A , a0 , a1 , a2
```

In [92]:
```
A = torch.arange(24).reshape(2,3,4)
a0 =torch.sum(A,axis = 0)
a1 = torch.sum(A,axis = 1)
a2 = torch.sum(A,axis = 2)
A , a0 , a1 , a2
```

Out[92]:
```
(tensor([[[ 0,  1,  2,  3],
          [ 4,  5,  6,  7],
          [ 8,  9, 10, 11]],

         [[12, 13, 14, 15],
          [16, 17, 18, 19],
          [20, 21, 22, 23]]]),
 tensor([[12, 14, 16, 18],
         [20, 22, 24, 26],
         [28, 30, 32, 34]]),
 tensor([[12, 15, 18, 21],
         [48, 51, 54, 57]]),
 tensor([[ 6, 22, 38],
         [54, 70, 86]]))
```

可以看到情况就是图片的样子，按照那个轴进行坍塌，也可以简单的表示为直接删去了所求轴的维度。