01-基础数学知识

“线性代数作为数学的一个分支，广泛应用于科学和工程中。然而，因为线性代数主要是面向连续数学，而非离散数学，所以很多计算机科学家很少接触它。掌握好线性代数对于理解和从事机器学习算法相关工作是很有必要的，尤其对于深度学习算法而言。因此，在开始介绍深度学习之前，我们集中探讨一些必备的线性代数知识。”

摘录来自: [美]Ian Goodfellow（伊恩·古德费洛）. “深度学习。” Apple Books.

本节主要内容：

- 标量
- 向量
- 矩阵
- 张量
- 张量的加减法

1.1 标量

“标量 （scalar）： 一个标量就是一个单独的数，它不同于线性代数中研究的其他大部分对象（通常是多个数的数组）。”

摘录来自: [美]Ian Goodfellow（伊恩·古德费洛）. “深度学习。” Apple Books.

乍一看可能比较绕，但它只是一个很简单的概念。比如我们在 `Python` 中表示：

In [1]:

```
a = 10
b = 5
```

上面的 `Python` 变量 `a` 和 `b` 都是标量。

1.2 向量

“向量 （vector）： 一个向量是一列数。这些数是有序排列的。通过次序中的索引，我们可以确定每个单独的数。”

摘录来自: [美]Ian Goodfellow（伊恩·古德费洛）. “深度学习。” Apple Books.

假设有向量 x，x 中标量的个数被称为向量 x 的长度，记为 n。

我们使用 x_1 表示向量 x 中的第一个数，x_2 表示向量 x 中的第二个数，以此类推。

所以，一个长度为 n 的向量 x 各个元素可以被表示为 $\{x_1, x_2, \ldots, xn\}$。

> 注意：
>
> 这里从逻辑上说明时，下标被默认为从 1 开始，到 n (包含)结束。
>
> 但在实际开发中，我们的下标一般是从 0 开始的，到 $n-1$ 结束。
>
> 后面的矩阵和张量同理，就不再重复说明了。

我们试着在 `Python` 中表示向量，这里需要使用一个第三方扩展包 `Numpy` 。

`Numpy` 支持高阶大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库，是目前最受欢迎度 `Python` 数据科学类库之一。

值得一提的是， `Numpy` 引入了多维数组以及可以直接有效率地操作多维数组的函数与运算符。因此在NumPy上只要能被表示为针对数组或矩阵运算的算法，其运行效率几乎都可以与编译过的等效C语言代码一样快，极大地缓解了 `Python` 在运行效率上的焦虑。

首先，如果没有安装 `Numpy` ，先进行安装。

In [2]:

```
!pip install numpy
```

```
Looking in indexes: https://super-spider-1605756846230:****@aaronjny
-pypi.pkg.coding.net/super-spider/super-spider/simple
Requirement already satisfied: numpy in /Users/aaron/anaconda3/lib/p
ython3.7/site-packages (1.18.1)
```

引入 `Numpy` 包，惯用的方法是给 `Numpy` 设置一个别名， `np` 。

In [3]:

```
import numpy as np
```

一个元素全为 `0` 、长度为 `5` 的向量可以被表示如下：

In [4]:

```
x = np.array([0, 0, 0, 0, 0])
x
```

Out[4]:

```
array([0, 0, 0, 0, 0])
```

我们也可以使用 Numpy 随机生成一个向量：

In [5]:

```
x = np.random.normal(size=(5,))
x
```

Out[5]:

```
array([-6.80296730e-01, -2.29194137e-01,  3.14624413e-01,  1.2125843
2e+00,
        6.44775288e-04])
```

1.3 矩阵

“矩阵 （matrix）：矩阵是一个二维数组，其中的每一个元素由两个索引（而非一个）所确定。”

摘录来自: [美]Ian Goodfellow（伊恩·古德费洛）. “深度学习。” Apple Books.

假设有 $n * m$（惯用法，表示矩阵是 n 行 m 列的） 的矩阵 $\boldsymbol{M}$, 我们使用 $\boldsymbol{M}_{i,j}$ 表示矩阵 $\boldsymbol{M}$ 第 i 行第 j 列的元素，所以左上角的元素可以被表示为 $\boldsymbol{M}_{1,1}$，右下角的元素可以被表示为 $\boldsymbol{M}_{n,m}$。

尝试使用 Numpy 创建一个随机矩阵：

In [6]:

```
m = np.random.normal(size=(3, 5))
m
```

Out[6]:

```
array([[ 0.49118024, -0.8379985 ,  0.58250048, -1.03419398,  0.28072
502],
       [-2.71479229, -0.14771702, -1.30718106,  0.29094421, -2.17916
213],
       [ 0.87394052, -1.54430192, -0.04796053, -2.05097584, -1.19206
319]])
```

左上角和右下角的元素分别为：

In [7]:

```
m[0][0],m[2][4]
```

Out[7]:

```
(0.49118024373448366, -1.192063191136317)
```

我们也可以用这种方式表达左上角和右下角的元素，对于 Numpy 的 Array 对象来说，两者是等价的：

```
m[i][j] == m[i, j]
```

In [8]:

```
m[0,0],m[2,4]
```

Out[8]:

```
(0.49118024373448366, -1.192063191136317)
```

1.4 张量

“张量 （tensor）：在某些情况下，我们会讨论坐标超过两维的数组。一般的，一个数组中的元素分布在若干维坐标的规则网格中，我们称之为张量。”

摘录来自: [美]Ian Goodfellow（伊恩·古德费洛）. “深度学习。” Apple Books.

张量的概念与向量、矩阵类似。其实我们可以更直观地去理解它：

概念	维数
标量	0
向量	1
矩阵	2
张量	>=3

对于一个3维张量，其各维度下标分别为 i, j, k 的元素表示为 $A_{i,j,k}$。

尝试使用 Numpy 生成一个随机的3维张量：

In [9]:

```
a = np.random.normal(size=(3, 4, 5))
a
```

Out[9]:

```
array([[[-1.47724692,  1.45040419,  0.28777928, -0.44289373,
         -0.53516083],
        [-0.36326611, -0.8070951 , -1.18320704, -0.46274787,
         -0.22551625],
        [-0.78442661,  0.63630508, -0.98151984,  0.75825865,
         -0.3187129 ],
        [-0.84284143, -1.91027065, -1.24614743,  0.48086974,
         -0.50506526]],

       [[-1.090708  ,  0.90377762, -0.94294048,  2.54612775,
          0.45130403],
        [-0.83583708,  0.77999195,  0.29024323, -1.39717058,
          0.2362981 ],
        [ 0.21803277,  0.18995668,  0.62341677, -1.00658737,
         -0.13480011],
        [ 0.65538761, -1.21235413,  0.39944123,  1.81018347,
         -1.93633755]],

       [[ 0.15848345, -0.171823  ,  0.49716578, -0.34148525,
          0.20683319],
        [ 0.09181793, -0.04569658,  1.62871946,  0.2153124 ,
          1.43293979],
        [-0.758154  , -0.84983143, -0.0624045 ,  1.38853805,
          1.44269434],
        [ 1.11872393, -0.66364665,  0.21946794,  1.29118661,
          2.2917813 ]]])
```

1.5 转置

“转置 （transpose）是矩阵的重要操作之一。矩阵的转置是以对角线为轴的镜像，这条从左上角到右下角的对角线被称为主对角线 （main diagonal）。”

摘录来自: [美]Ian Goodfellow（伊恩·古德费洛）. “深度学习。” Apple Books.

通俗点来说，就是把一个矩阵的行变成列、列变成行。

以一个 $2 * 3$ 的矩阵为例，矩阵 M 表示如下：

$$M = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix}$$

其转置矩阵 M^T 表示如下：

$$M^T = \begin{bmatrix} 0 & 3 \\ 1 & 4 \\ 2 & 5 \end{bmatrix}$$

尝试使用 `Numpy` 实现：

In [10]:

```
m = np.array([[0, 1, 2], [3, 4, 5]])
print('矩阵 m :')
m
```

矩阵 m :

Out[10]:

```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [11]:

```
mt = np.transpose(m)
print('矩阵 m 的转置:')
mt
```

矩阵 m 的转置:

Out[11]:

```
array([[0, 3],
       [1, 4],
       [2, 5]])
```

再转置回去也是一样的，经过两次转置的矩阵和原矩阵是相等的：

In [12]:

```
m == np.transpose(mt)
```

Out[12]:

```
array([[ True,  True,  True],
       [ True,  True,  True]])
```

再讨论一下标量和向量的转置。

标量可以看做只有一个元素的矩阵，因此它的转置就是它本身。

向量可以看作是只有一行（行向量）或一列（列向量）的矩阵，因此对向量的转置就是把行向量和列向量的互相转换。

三维及以上的张量我们一般不讨论转置问题，如果讨论，也是将其中的两维当做矩阵来讨论转置。

1.6 矩阵加减法

对于一个 $n * m$ 的 矩阵 M，我们称矩阵 M 的 `shape` 为 `(n, m)`。

比如：

In [13]:

```
m = np.array([[0, 1, 2], [3, 4, 5]])
m.shape
```

Out[13]:

```
(2, 3)
```

严格来说，只有 `shape` 相同矩阵才可以做加减法运算。

对矩阵做加减法，实际上是对矩阵上相同位置的元素做加减法：

$$C = A + B$$

实际运算：

$$C_{i,j} = A_{i,j} + B_{i,j}$$

其中$0 < i <= n, 0 < j <= m$。

In [14]:

```
a = np.random.normal(size=(2, 3))
b = np.random.normal(size=(2, 3))
c = a + b
a,b,c
```

Out[14]:

```
(array([[-1.98188627,  0.64308176,  1.63826943],
        [ 0.4741988 , -1.14586692,  1.9500476 ]]),
 array([[ 3.23050371, -2.09977565, -0.45927353],
        [-0.03677667,  1.54541431,  0.35497293]]),
 array([[ 1.24861744, -1.45669389,  1.1789959 ],
        [ 0.43742213,  0.39954739,  2.30502053]]))
```

但实际上，矩阵和标量或符合条件的向量也是可以做加减法的，这种方法叫做 广播 。

矩阵和标量做加、减、乘、除运算，相当于是矩阵中各个元素分别和标量做运算。以加法举例，即：

$$C = A + b$$

等价于：

$$C_{i,j} = A_{i,j} + b$$

其中$0 < i <= n, 0 < j <= m$。

使用 `Numpy` 演示：

In [15]:

```
a = np.array([[1, 1, 1], [2, 2, 2]])
b = 3
c = a+b
c
```

Out[15]:

```
array([[4, 4, 4],
       [5, 5, 5]])
```

注意，这里以矩阵和标量的运算举例，但显然，这种规则也适用于更高纬度的张量，故张量与标量的运算法则与商数相同。

矩阵和向量的加减运算，需要满足一定的条件。简单来说，对于 $n * m$ 的矩阵：

- 当向量为行向量时，向量的长度需要为 m。
- 当向量为列向量时，向量的长度需要为 n。（如果使用 `Numpy` 运算，此时需要将向量 reshape 为 $n * 1$ 的矩阵。）

使用 `Numpy` 演示：

In [16]:

```
# 矩阵和行向量
a = np.random.normal(size=(2, 3))
b = np.random.normal(size=(3,))
c = a + b
c
```

Out[16]:

```
array([[-1.16777634, -1.13722631,  1.07140355],
       [ 1.05301046,  0.92230674,  1.00003219]])
```

In [17]:

```
# 矩阵和列向量
a = np.random.normal(size=(2, 3))
b = np.random.normal(size=(2,))
b = np.reshape(b, (2, 1))
c = a + b
c
```

Out[17]:

```
array([[ 1.10841252,  0.90262979, -0.07237292],
       [-1.20199719,  0.28281933, -0.44196774]])
```

值得一提的是，这里虽然是说矩阵和向量的加减运算，但抽象一下可以发现，我们讲的实际上是高维张量和低维张量之间的运算。因此，此计算规则也可以推广到张量计算中去。

以 `Numpy` 举例：

In [18]:

```
# 五维张量和行向量
a = np.random.normal(size=(2, 1, 1, 2, 3))
b = np.random.normal(size=(3,))
c = a + b
c
```

Out[18]:

```
array([[[[[-0.99144413,  0.69572448, -1.91827681],
          [-0.21011665, -0.4784677 , -2.73886853]]]],



       [[[[ 1.02164992,  0.26722761, -2.45975445],
          [-0.62432421,  0.37397614, -1.78144773]]]]])
```

1.7 矩阵乘法

“矩阵乘法是矩阵运算中最重要的操作之一。两个矩阵 A 和 B 的矩阵乘积 （matrix product）是第三个矩阵 C 。为了使乘法可被定义，矩阵 A 的列数必须和矩阵 B 的行数相等。如果矩阵 A 的形状是 $m * n$，矩阵 B 的形状是 $n * p$，那么矩阵 C 的形状是 $m * p$。”

摘录来自: [美]Ian Goodfellow（伊恩·古德费洛）. “深度学习。” Apple Books.

矩阵 A 和 B 的乘积可以表示为：

$$C = AB$$

其实际计算规则是用 A $(m * n)$ 的每一行，去与 B $(n * p)$中的每一列中的对应位置的数相乘（所以上面会要求A的列数要等于B的行数），并对乘积求和（即求A中所有行向量和B中所有列向量求点积），构建出新的矩阵。数学表达式如下：

$$C_{i,j} = \sum_{k}^{n} A_{i,k} * B_{k,j}$$

In [19]:

```
a = np.array([[1, 1], [2, 2], [3, 3]])
b = np.array([[1, 1, 1], [2, 2, 2]])
c = a.dot(b)
c
```

Out[19]:

```
array([[3, 3, 3],
       [6, 6, 6],
       [9, 9, 9]])
```

矩阵乘法满足分配律，即：

$$A(B + C) = AB + AC$$

In [20]:

```
a = np.array([[1, 1, 1], [1, 1, 1]])
b = np.array([[2, 2], [2, 2], [2, 2]])
c = np.array([[3, 3], [3, 3], [3, 3]])
np.dot(a, b+c) == np.dot(a, b)+np.dot(a, c)
```

Out[20]:

```
array([[ True,  True],
       [ True,  True]])
```

矩阵乘法也满足结合律，即：

$$A(BC) = (AB)C$$

In [21]:

```
a = np.array([[1, 1, 1], [1, 1, 1]])
b = np.array([[2, 2], [2, 2], [2, 2]])
c = np.array([[3, 3, 3], [3, 3, 3]])
np.dot(a, np.dot(b, c)) == np.dot(np.dot(a, b), c)
```

Out[21]:

```
array([[ True,  True,  True],
       [ True,  True,  True]])
```

需要注意，矩阵乘法并不满足交换律。

其实很明显，为了使乘法可被定义，矩阵 A 的列数必须和矩阵 B 的行数相等。交换之后不一定能满足此条件。

矩阵乘积的转置：

$$(AB)^T = B^T A^T$$

In [22]:

```
a = np.array([[1, 1, 1], [1, 1, 1]])
b = np.array([[2, 2], [2, 2], [2, 2]])
np.transpose(np.dot(a, b)) == np.dot(np.transpose(b), np.transpose(a))
```

Out[22]:

```
array([[ True,  True],
       [ True,  True]])
```