1. 为什么需要学习Numpy

`NumPy`（Numerical Python的简称）是高性能科学计算和数据分析的基础包。在后面的深度学习和大模型的处理, 在底层运算的时候, 都会使用Numpy进行数据处理. 提高操作的性能.

1. 1. 高性能的并行计算

当我们在使用GPU进行运算的时候, 很多操作是可以并行处理, 但是需要数据结构的支持, 对于传统的数据结构, 比如Python中的List列表, 是无法支持并行处理的, 但是对于Numpy这种数据结构, 是支持并行计算, 可以快速提高运算效率

- Python 列表的计算方式

```
import time
# 创建两个大列表
list1 = list(range(10000000))
list2 = list(range(10000000))
# 列表相加（使用列表解析）
start_time = time.time()
result_list = [a + b for a, b in zip(list1, list2)]
print(f"列表计算时间：{time.time() - start_time} 秒")
```

> 在个人电脑上运行, 大概需要0.93秒
>
> 注意: 每个人运行的效果可能不一样(由机器的性能等相关决定)

- NumPy 数组的计算方式

```
import numpy as np
# 创建两个大数组
array1 = np.arange(10000000)
array2 = np.arange(10000000)
# 数组相加（矢量化运算）
start_time = time.time()
result_array = array1 + array2
print(f"NumPy 计算时间：{time.time() - start_time} 秒")
```

> 在个人电脑上运行, 大概需要0.03秒
>
> 注意: 每个人运行的效果可能不一样(由机器的性能等相关决定)

1. 2. 丰富的函数库与数据处理

- Python实现列表+1

```
# Python原生的list
a = [1, 2, 3, 4, 5]
# 完成如下计算
# 对a的每个元素 + 1
for i in range(5):
      a[i] = a[i] + 1
a
```

```
[2, 3, 4, 5, 6]
```

- Numpy实现数组+1

```
# 使用ndarray
import numpy as np
# 创建一个Numpy的数组类型对象
a = np.array([1, 2, 3, 4, 5])
a = a + 1 # 直接使用数组+1
a
```

```
array([2, 3, 4, 5, 6])
```

- Numpy丰富的函数

```
import numpy as np
# 生成标准正态分布的随机数
random_numbers = np.random.randn(1000000)
# 计算均值和标准差
mean = np.mean(random_numbers)
std_dev = np.std(random_numbers)
print(f"均值：{mean}, 标准差：{std_dev}")
```

```
均值：-0.00152683636167906, 标准差：0.9991470680065853
```

1. 3. Numpy的ndarray是一个通用的数据类型

在深度学习和大模型开发中，NumPy 扮演着关键角色，因为它能高效地处理和转换数据，这对于模型训练和推理至关重要。

在多媒体数据处理方面，NumPy 也显示出其独特的优势：

- **图片**：通常被表示为三维数组（高度、宽度、颜色通道），其中每个元素对应于图像的一个像素点。灰度图片则简化为二维数组。
- **视频**：视频是由一系列连续帧组成，每一帧本质上是一张图片。处理视频时，通常将每一帧作为一个独立的图像读取，然后转换成 NumPy 数组，最终整个视频可以表示为一个四维数组（帧数、高度、宽度、颜色通道）。
- **音频**：音频数据可以表示为一维数组（单声道）或二维数组（立体声或更多通道），其中数组中的元素代表连续时间点的音频信号强度。读取音频文件后，可以将整个音频信号转换为 NumPy 数组进行进一步处理。

2. Numpy的基本操作

2. 1. 环境准备

NumPy是第三方库，所以在使用 NumPy之前必须安装 NumPy 。

```
# 下面两种方法都可以，选择一种方法即可
pip install numpy
pip install numpy -i https://pypi.tuna.tsinghua.edu.cn/simple
```

```
# 验证代码
import numpy as np
print(np) # 如果没有报错，即安装成功
```

2. 2. 创建Numpy的数组对象

1. 使用array方法直接创建

```
import numpy as np
a = np.array({"a":1,"b":2}) # 接受一个列表或者元组,
a
```

> array({'a': 1, 'b': 2}, dtype=object)

2. 使用arange方法创建

```
import numpy as np
# arange 有3个参数，start: 开始值(包括) 默认为0    stop: 结束值(不包括) 必须填写 step:步长 默认为1
a = np.arange(0,10)
print(a)
b = np.arange(1,10,2)
print(b)
```

3. 使用zeros方法创建

   数组的元素的值全部是0

```
import numpy as np
#创建一个有3个元素的一维数组
a = np.zeros(3)
print(a)
# 创建一个2行4列的二维数组
b = np.zeros((2,4))
print(b)
```

4. 使用ones方法创建

   数组的元素的值全部是1

```
import numpy as np
#创建一个有4个元素的一维数组
a = np.ones(4)
print(a)
# 创建一个3行2列的二维数组
b = np.ones((3,2))
print(b)
```

2. 3. 查看数组的常见的属性

ndarray的属性包括 `shape` 、 `dtype` 、 `size` 和 `ndim`

- shape: 数组的形状 `ndarray.shape`
- dtype: 数组的数据类型
- size: 数组中包含的元素个数 `ndarray.size` 其大小等于各个维度的长度的乘积
- ndim: 数组的维度大小 `ndarray.ndim` 其大小等于shape包含的元素的个数

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
# 查看a的形状
print(a.shape) # 2行3列
# 查看a的维度
print(a.ndim) # 2维数组
# 查看数据类型
print(a.dtype) # int64
# 查看一共有多少个元素
print(a.size) # 6
```

2. 4. 改变数组的数据类型和形状

> 注意 对于reshape方法和astype方法都不会改变原来的数组, 只会返回一个新的数组

```
import numpy as np
# 创建一个12个元素的一维数组,并且是int64数据类型
a = np.arange(12)
print(a.dtype) # int64
b = a.reshape(2,6) # 创建一个新的数组, 形状是2行6列, 原来的数组不会改变
print(b)
b = b.astype(np.float64) #把数据类型修改为float类型, 原来的数据不会改变,
print(b.dtype)
# 把一个多维数组转换为一维数组
c = b.flatten()
print(c)
```

2. 5. ndarray数组的基本运算

NumPy中的ndarray数组非常灵活，它可以像我们日常使用的数字一样进行基本的加减乘除操作。这种操作主要分为两类：

1. **标量与数组的运算**：你可以将一个数字（标量）与一个数组进行运算。这时,这个数字会与数组中的每一个元素分别进行加、减、乘或除。例如,如果你将一个数组里的每个数字都想增加5，只需简单地把5加到这个数组上。
2. **数组与数组的运算**：两个 形状相同 的数组也可以进行对应元素之间的加减乘除操作。这意味着第一个数组的第一个元素将与第二个数组的第一个元素相加（或相减、相乘、相除），第二个元素与第二个元素运算，以此类推。

2. 5.1. 标量和ndarray数组之间的运算

darray数组之间的运算主要包括除法、乘法、加法和减法运算,在标量和ndarray数组运算的时候, 标量直接和ndarray数组中的没一个元素进行运算, 返回的ndarray的形状和原来的ndarray数组一样

```
import numpy as np
# 创建一个2行3列的二维数组
a = np.array([[1,2,3],[4,5,6]])
# 加法操作，原来的数组不会改变，返回一个新的数组
b = a + 10 # a中的每个元素都和10 操作  1+10，2+10...
print(b)
b = 10-a   # 使用10 减去a中的每个元素  10-1，10-2...
print(b)
b = 2*a # 2乘以a中的每个元素  2*1，2*2...
print(b)
b = 1/a # 1除以a中的每个元素 1/1,1/2,1/3...
print(b)
```

2. 5.2. 两个ndarray数组之间的运算

在两个ndarray数组之间的运算, 对应位置上的元素依次进行对应的运算

```
# 两个ndarray数组之间的运算
a = np.array([1,2,3])
b = np.array([4,5,6])
# 两个数组相加，返回一个新的数组
c = a+b   # 对应位置元素相加
print(c)
c=a*b
print(c)
c = a/b
print(c)
c = a**2
print(c)
```

```
# 两个ndarray数组之间的运算
a = np.array([1,2,3])
b = np.array([[4,5,6],[6,7,8]])
# 两个数组相加，返回一个新的数组
c = a+b   # 对应位置元素相加
print(c)
```

2. 5.3. 广播机制

NumPy 的广播机制是一种强大的功能，允许对不同形状的数组进行算术运算，这通常在形状相同的数组之间才是可能的。这种机制在数据科学和机器学习中非常有用，因为它允许对数据进行批量处理，而无需显式复制数据。

广播遵循一组简单的规则来应用通用函数（如加法、乘法等）：

1. **对齐维度**：从数组的尾部维度开始向前对齐，即每个数组的最后一个维度在形状上与另一个数组的最后一个维度对齐，然后是倒数第二个维度，依此类推。
2. **维度扩展**：如果一个数组在某个维度上的长度为1（或在比较中该维度不存在），而另一个数组在该维度上的长度大于1，那么首先假设前者在该维度的长度通过复制扩展到匹配后者。
3. **尺寸匹配**：两个数组能够广播，如果对于所有的维度，它们要么是相同的长度，要么其中一个是1。

如果这三条规则不能满足，NumPy 将引发一个错误，表示形状不兼容。

广播两个数组

考虑以下两个数组，形状分别为 `(4,)` 和 `(3, 4)`：

```
import numpy as np

a = np.array([1, 2, 3, 4])          # 形状为 (4,)
b = np.array([[1, 2, 3, 4],         # 形状为 (3, 4)
              [5, 6, 7, 8],
              [9, 10, 11, 12]])
```

对齐维度：`a` 形状视为 `(1, 4)` 以匹配 `b` 的 `(3, 4)`。

维度扩展：将 `a` 在第一个维度上复制扩展，从 `(1, 4)` 变为 `(3, 4)`。

现在，`a` 的每一行都与 `b` 的对应行对齐，可以直接进行元素级加法：

```
c = a + b
print(c)
```

输出结果

```
[[ 2  4  6  8]
 [ 6  8 10 12]
 [10 12 14 16]]
```

2. 5.4. 广播机制深入剖析

我们来看下面的代码具体是怎么执行的

```
# 两个ndarray数组之间的运算
a = np.array([1,2,3])
b = np.array([[4,5,6],[6,7,8]])
# 两个数组相加，返回一个新的数组
c = a+b   # 对应位置元素相加
print(c)
```

```
# 1 其中a的形状是(3,) b的形状是(2,3)
# 2 对于a的形状进行扩展变为(1,3)，b的形状是(2,3)
# 3 为了保证两个数组的维度一样，对于a的形状在行上(维度为1)进行复制，变为(2,3)
a = [[1,2,3],[1,2,3]] # (2,3)
# 4 在对应的位置进行对应的加法操作
```

2. 6. ndarray数组的索引和切片

NumPy 中的 ndarray 数组索引和切片功能非常类似于 Python 中的列表。无论是通过特定位置的元素访问还是提取数组的子集，这些操作都是简单直观的。让我们来详细了解这些技巧：

2. 6.1. 索引：从数组中选取元素

- **单个元素访问**：通过在方括号 `[]` 中放置一个整数索引来访问数组的一个元素。例如，`array[2]` 将返回数组中的第三个元素（索引从0开始）。
- **负数索引**：使用负数索引可以从数组的末尾开始访问元素。例如，`array[-1]` 是获取数组最后一个元素的快捷方式。

2. 6.2. 切片：提取数组的部分区域

- 基本切片

  通过指定起始点、终点和步长来切片。这些参数在切片操作中用冒号:分隔。例如，

  ```
  array[start:stop:step]
  ```

  会根据指定的规则提取元素。

  - `start` 是切片开始的位置。
  - `stop` 是切片结束的位置，但不包括此位置的元素。
  - `step` 是选取元素的间隔。
- **简化操作**：如果省略 `start`，则默认从头开始；如果省略 `stop`，则继续到数组末尾；如果省略 `step`，则默认步长为1。

2. 6.3. 一维数组的索引和切片

- 对于一维数组，你可以直接使用上述索引和切片方法进行操作，就像处理普通的 Python 列表一样。

2. 6.4. 多维数组的索引和切片

- **多维切片**：对于多维数组，每个维度都可以单独进行切片。例如，对于一个二维数组 `matrix`，`matrix[1:3, 0:2]` 将选取第二行到第四行（不包括第四行），以及第一列到第三列（不包括第三列）的部分。

> 注意：对于切片，不会复制新的数据，只会创建一个视图，数据还是原来的数据，如果改变切片中的数据，对于原来数组中的内容也会跟着改变，如果需要新的一份独立的数据，可以使用copy()拷贝一份数据

```
# 一维数组的索引和切片
a = np.array([2,4,5,3,1,6,8])
# 获取数组的第一个元素，索引从0开始
print(a[0])
# 获取数组的2-5个元素
b = a[1:5] # 包括开始位置，不包括解释位置
print(b)
# 从第二个元素开始，每隔一个元素获取一个元素
b = a[1::2] # 元素是同一份，b只是数据的一份视图
print(b)
b[0]=100 # 修改b中的元素，a中的元素也会修改
print(a)
b = b.copy() # 重新复制一份数据，a和b之间的数据没有关系
b[0]=500
```

```
print(b)
print(a)
```

```
# 多维数组的索引和切片
# 多维数组的索引和切片
a = np.arange(24)
a = a.reshape(2,4,3) # 创建一个3维数组,
# 获取到第一个4行3列的二维数组
print(a[0])
# 获取到第二个4行3列数组中的第二行
print(a[1,1])
# 获取到第二个4行3列数组中的第二列
print(a[0,:,1]) # 中括号第一个值代表第一个维度，第二个值代表第二个维度，第三个值 代表第三个维度
```

```
 # 多维数组的索引和切片
a = np.arange(24)
a = a.reshape(2,4,3) # 创建一个3维数组,
b = a[:,1:3,0:2] # 使用切片获取对应的元素
print(b)
```

2. 7. ndarray数组的统计方法

可以通过数组上的一组数学函数对整个数组或某个轴向的数据进行统计计算。主要包括如下统计方法：

- `mean`：计算算术平均数，零长度数组的mean为NaN。
- `std` 和 `var`：计算标准差和方差，自由度可调（默认为n）。
- `sum`：对数组中全部或某轴向的元素求和，零长度数组的sum为0。
- `max` 和 `min`：计算最大值和最小值。
- `argmin` 和 `argmax`：分别为最大和最小元素的索引。
- `cumsum`：计算所有元素的累加。
- `cumprod`：计算所有元素的累积。

```
# ndarray数组的统计方法
a = np.arange(1,25)
a = a.reshape(4,6)
print(a.mean()) # 计算平均值
print(np.mean(a)) # 计算平均值
print(a.std(),a.var()) # 计算标准差和方差
print(a.sum()) # 求和计算
print(a.max(),a.min()) # 计算最大值和最小值
print(a.argmin(),a.argmax()) # 计算最小值的位置，最大值的位置
print(a.cumsum())# 计算累计求和
print(a.cumprod()) # 计算累计乘积
```

```
# ndarray数组的统计方法
a = np.arange(1,25)
a = a.reshape(4,6)
print(a)
print("="*50)
print(a.mean(axis=0)) # 根据列进行平均值计算
print(a.mean(axis=1)) # 根据行进行平均值计算
print(a.std(axis=0), a.var(axis=1)) # 根据列进行标准差计算，行进行方差计算
print(a.sum(axis=0)) # 根据列进行求和计算
```

2. 8. 随机数np.random

主要介绍创建ndarray随机数组以及随机打乱顺序、随机选取元素等相关操作的方法。

2. 8.1. 创建随机ndarray数组

创建随机ndarray数组主要包含设置随机种子、均匀分布和正态分布三部分内容

```
# 1 设置随机种子,种子一样, 创建的随机数是一样的
np.random.seed(0)
# 2 创建一个均匀分布的数组, 指定创建数组的维度
a = np.random.rand(2,3) # 元素的取值[0,1)的一个2行3列的数组
print(a)
print("="*10)
# 3 创建一个数值范围在[1,10)直接的2行3列的数组
b = np.random.randint(low=1,high=10,size=(2,3))
print(b)
print("="*10)
# 4 生成均匀分布随机数, 指定随机数取值范围和数组形状
c = np.random.uniform(low=-1,high=1,size=(4,6))
print(c)
print('='*10)
# 5 生成正态分布随机数, 指定均值loc和方差scale
d = np.random.normal(loc=0,scale=1,size=(400,700))
print(c)
print(c.mean(),c.var())
```

2. 8.2. 随机打乱ndarray数组顺序

```
# 一维数组随机打乱
a = np.arange(10)
# 直接改变原来的数组, 数据顺序会随机打乱
np.random.shuffle(a)
print(a)
```

```
# 多维数字的随机打乱, 只会打乱第一维的顺序
a = np.arange(12)
a = a.reshape(3,4)     # 改变为一个3行4列的数组
np.random.shuffle(a) # 直接打乱原来的数组, 行会发生改变
print(a)
```

2. 8.3. 随机选取元素

```
a = np.arange(12)
a = a.reshape(3,4)     # 改变为一个3行4列的数组
# 只能从一维数组中进行选取
np.random.choice(a.flatten(), size=(2,2))
```

2. 9. 内积和外积

在 NumPy 中，内积和外积是两种基本的向量运算，它们在数学、物理、工程、数据科学等多个领域都有广泛的应用。这些运算不仅用于基本的数学计算，还在解决实际问题中发挥着重要作用。

2. 9.1. 内积

内积，或称点积（Dot Product），是两个向量的元素对应相乘后求和的结果，结果是一个标量。在 NumPy 中，内积可以通过 `np.dot(a, b)` 或 `a.dot(b)` 进行计算，适用于一维和多维数组。

```
# 一维数组的内积
# 创建两个一维数组
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
# 计算内积
inner_product = np.dot(a, b)
print("内积结果:", inner_product)
# 计算逻辑 1*4+2*5+3*6=32
```

```
# 二维数组的内积
import numpy as np
# 创建一个 3x4 的二维数组
a = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])

# 创建一个 4x1 的二维数组（列向量）
b = np.array([[1],[2],[3],[4]])

# 计算内积
inner_product_2d = np.dot(a, b)
print("内积结果:\n", inner_product_2d)
# 计算规则: 1*1+2*2+3*3+4*4=30
# 计算规则: 5*1+6*2+7*3+8*4=70
# 计算规则: 9*1+10*2+11*3+12*4=110
```

> 在多维数组计算内积的时候, 要求第一个数组a (M,K)的最后一个维度 需要和第二个数组b(K,N)的倒数第二个维度保持一致
>
> 计算后的形状是(M,N)

2. 9.2. 外积

外积，或张量积（Outer Product），是两个向量的每个元素对应相乘，结果是一个矩阵。在 NumPy 中，外积通过 `np.outer(a, b)` 计算。

```
# 向量定义
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
# 计算外积
outer_product = np.outer(a, b)
print("外积: \n", outer_product)
# 计算规则 1*4, 1*5, 1*6
# 计算规则 2*4, 2*5, 2*6
# 计算规则 3*4, 3*5, 3*6
```

2. 10. 维度转换

NumPy 的 `transpose` 方法是用于调整数组的维度顺序的一个非常有用的工具。在数学、数据处理、图像处理和机器学习等领域中，这种能力特别重要，因为它可以帮助我们根据需要重新排列多维数据的维度。

应用场景

1. **图像处理**：
    - 在处理图像数据时，经常需要将图像数组的维度进行转换，以满足特定库或算法的输入要求。例如，从 `(高度，宽度，颜色通道)` 转换到 `(颜色通道，高度，宽度)`。
2. **数据科学**：
    - 在处理多维数据集时，经常需要调整数据的维度以匹配特定分析工具或数据模型的预期输入格式。
3. **机器学习**：
    - 在神经网络中，可能需要对输入数据的维度进行重排，以符合网络层期望的输入结构。特别是在使用不同框架（如 TensorFlow 和 PyTorch）时，它们对数据维度的要求可能不同。
4. **数值计算**：
    - 在进行线性代数操作时，有时需要转置矩阵来执行特定的矩阵运算。

```
# 二维数组转置
# 创建一个二维数组
a = np.array([[1, 2, 3], [4, 5, 6]])
print("原始数组:\n", a)
# 转置数组
a_transposed = np.transpose(a)
print("转置后的数组:\n", a_transposed)
```

```
# 三维数组维度重排
# 考虑一个三维数组，我们需要调整其维度的顺序。例如，我们有一个形状为 (2, 3, 4) 的数组，我们想要调整为 (4, 2, 3) 的形状。
# 创建一个三维数组
b = np.arange(24).reshape(2, 3, 4)
print("原始数组形状:", b.shape)
# 重排维度
# b_transposed = b.transpose(2,0,1)
b_transposed = np.transpose(b, axes=(2, 0, 1))
print("重排后的数组形状:", b_transposed.shape)
```

2. 11. 文件保存

NumPy提供了 `save` 和 `load` 接口，直接将数组保存成文件(保存为.npy格式)，或者从.npy文件中读取数组。

```
# 保存数组到 .npy 文件
# 创建一个示例数组
array_to_save = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# 保存数组到文件
np.save('saved_array.npy', array_to_save)
print("Array has been saved to 'saved_array.npy'")
```

```
# 从文件加载数组
loaded_array = np.load('saved_array.npy')
print("Loaded array:")
print(loaded_array)
```

3. Numpy的应用

3. 1. 深度学习中的矩阵和向量运算

在深度学习中,数据通常以矩阵的形式表示，例如神经网络的权重和输入数据。使用 NumPy 进行这些矩阵的计算是非常高效的。下面是一个简单的示例，说明如何使用 NumPy 来模拟一个基本的前向传播过程，即一个神经网络层的计算。

需求说明:

假设我们有一个简单的全连接层（也称为密集层），它执行以下计算：

$output = activation(W \cdot input + b)$

其中 W 是权重矩阵，b 是偏置向量，activation是激活函数（如ReLU）。

```
import numpy as np
# 定义一个激活函数relu
def relu(x):
      return np.maximum(0, x)
# 生成示例数据(模拟数据)
input_data = np.random.rand(10, 5)   # 10个输入样本，每个样本5个特征

# 初始化权重和偏置
W = np.random.rand(5, 3)   # 将5个输入特征转换为3个输出特征
b = np.random.rand(3)      # 对应于3个输出特征的偏置

# 计算网络层的输出
z = np.dot(input_data, W) + b
output = relu(z)   # 应用ReLU激活函数
print("网络输出:\n", output)
```

3. 2. 图像处理中的数组操作

NumPy 可以用来处理图像数据，例如进行图像的翻转、缩放等操作。这些操作本质上是对数组进行操作。

需求说明:

- **图像翻转**：水平翻转和垂直翻转。
- **图像缩放**：使用简单的重采样方法来缩放图像。

> 需要先安装依赖
>
> pip install Pillow
>
> pip install Pillow -i **https://pypi.tuna.tsinghua.edu.cn/simple**

```
import numpy as np
from PIL import Image
# 加载图像并转换为numpy数组
image = Image.open('a.jpg')
image_array = np.array(image)
```

```
# 获取图片的数组属性
print(image_array.shape)
# 水平翻转
horizontally_flipped = np.fliplr(image_array)
# 垂直翻转
vertically_flipped = np.flipud(image_array)
# 图像缩放（简单重采样，每隔一个像素取样）
scaled_image = image_array[::2, ::2]
# 显示原始和变换后的图像
# Image.fromarray(horizontally_flipped).show()
# Image.fromarray(vertically_flipped).show()
Image.fromarray(scaled_image).show()
```

4. 作业

4. 1. 作业题 1：内积和外积的运算

题目描述： 给定两个向量 `v1 = [1, 2, 3]` 和 `v2 = [4, 5, 6]`，请完成以下任务：

- 计算并打印这两个向量的内积。
- 计算并打印这两个向量的外积。

4. 2. 作业题 2：多维数组的维度转换

题目描述： 创建一个形状为 `(4, 5, 6)` 的三维数组，使用随机数填充此数组。完成以下任务：

- 将数组转置为 `(5, 6, 4)` 的形状，并打印结果。
- 将转置后的数组进一步转换为 `(6, 5, 4)` 的形状，并打印结果。

4. 3. 作业题 3：数组的切片操作

题目描述： 创建一个形状为 `(8, 10)` 的二维数组，使用顺序整数填充（从1开始）。完成以下任务：

- 提取第 3 行的所有元素并打印。
- 提取第 5 列的所有元素并打印。
- 提取中间的 4x5 子数组并打印。

4. 4. 作业题 4：数组的广播机制

题目描述： 定义一个形状为 `(5, 1)` 的列向量和一个形状为 `(1, 5)` 的行向量，使用简单的连续整数填充。执行以下任务：

- 使用广播机制将这两个向量相加，并打印结果。
- 将结果矩阵与一个形状为 `(5, 5)` 的全1矩阵相乘，并打印结果。