DL_2

2025 年 6 月 22 日

[1]:
```
import torch

print(torch.__version__)
print(torch.version.cuda)
print(torch.cuda.is_available())
```

```
2.7.1+cu128
12.8
True
```

[2]:
```
num_gpus = torch.cuda.device_count()
print("当前可用的 GPU 数量: ", num_gpus)
device = torch.device(f"cuda:{0}")
properties = torch.cuda.get_device_properties(device)
print(f"GPU {0} 的详细信息: ")
print("名称: ", properties.name)
print("显存大小: ", properties.total_memory)
```

```
当前可用的 GPU 数量:  1
GPU 0 的详细信息:
名称:  NVIDIA GeForce GTX 1650
显存大小:  4294639616
```

[3]:
```
import numpy as np

x = np.arange(12)
print(x)
print(x.shape)
print(x.size)
```

```
print(x.reshape(3, 4))
print(np.zeros((2, 3, 4)))
print(np.ones((2, 3, 4)))
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
(12,)
12
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[[[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]

 [[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]]
[[[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]

 [[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]]
```

[4]:
```
np.random.normal(0, 1, size=(3, 4))
```

[4]:
```
array([[-0.4620047 ,  1.06716902, -0.0291128 ,  1.65906922],
       [ 0.00862075,  0.07265409, -0.55022321,  1.8607617 ],
       [-0.63960364,  0.36240491,  0.56537187,  0.33692242]])
```

[5]:
```
np.array([[2, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
```

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

[6]:
```
x = np.array([1, 2, 4, 8])
y = np.array([2, 2, 2, 2])
x + y, x - y, x * y, x / y, x ** y
```

[6]:
```
(array([ 3,  4,  6, 10]),
 array([-1,  0,  2,  6]),
 array([ 2,  4,  8, 16]),
 array([0.5, 1. , 2. , 4. ]),
 array([ 1,  4, 16, 64]))
```

[7]:
```
np.exp(x)
```

[7]:
```
array([2.71828183e+00, 7.38905610e+00, 5.45981500e+01, 2.98095799e+03])
```

[8]:
```
X = np.arange(12).reshape(3, 4)
Y = np.array([[2, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
np.concatenate([X, Y], axis=0), np.concatenate([X, Y], axis=1)
```

[8]:
```
(array([[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11],
        [ 2,  1,  4,  3],
        [ 1,  2,  3,  4],
        [ 4,  3,  2,  1]]),
 array([[ 0,  1,  2,  3,  2,  1,  4,  3],
        [ 4,  5,  6,  7,  1,  2,  3,  4],
        [ 8,  9, 10, 11,  4,  3,  2,  1]]))
```

[9]:
```
X == Y
```

[9]:
```
array([[False,  True, False,  True],
       [False, False, False, False],
       [False, False, False, False]])
```

[10]:
```
X.sum()
```

[10]:
```
np.int64(66)
```

[11]:
```
a = np.arange(3).reshape(3, 1)
b = np.arange(2).reshape(1, 2)
a, b
```

[11]:
```
(array([[0],
        [1],
        [2]]),
 array([[0, 1]]))
```

[12]:
```
a + b
```

[12]:
```
array([[0, 1],
       [1, 2],
       [2, 3]])
```

[13]:
```
X[-1], X[1:3]
```

[13]:
```
(array([ 8,  9, 10, 11]),
 array([[ 4,  5,  6,  7],
        [ 8,  9, 10, 11]]))
```

[14]:
```
before = id(Y)
Y = X + Y
id(Y) == before
```

[14]:
```
False
```

[15]:
```
Z = np.zeros_like(Y)
print('id(Z):', id(Z))
Z[:] = X + Y
print('id(Z):', id(Z))
```

```
id(Z): 2243508309616
id(Z): 2243508309616
```

[16]:
```
type(X)
B = np.array(X)
type(X), type(B)
""" 在进行深度学习任务的时候，一定要注意深度学习框架定义的张量和 NumPy 张量之间可能存在的转换问题"""
```

[16]: '在进行深度学习任务的时候，一定要注意深度学习框架定义的张量和 NumPy 张量之间可能存在的转换问题'

[17]:
```
""" 深度学习存储和操作数据的主要接口是张量（n 维数组）。它提供了各种功能，包括基本数学运算、广
播、索引、切片、内存节省和转换其他 Python 对象。"""
a = np.array([3.5])
a, a.item(), float(a), int(a)
```

```
C:\Users\DaiYongle\AppData\Local\Temp\ipykernel_4392\2956111559.py:4:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is
deprecated, and will error in future. Ensure you extract a single element from
your array before performing this operation. (Deprecated NumPy 1.25.)
  a, a.item(), float(a), int(a)
```

[17]: (array([3.5]), 3.5, 3.5, 3)

[18]:
```
import os

os.makedirs(os.path.join('..', 'data'), exist_ok=True)
data_file = os.path.join('..', 'data', 'house_tiny.csv')
with open(data_file, 'w') as f:
    f.write('NumRooms,Alley,Price\n')
    f.write('NA,Pave,127500\n') # 每行表示一个数据样本
    f.write('2,NA,106000\n')
    f.write('4,NA,178100\n')
    f.write('NA,NA,140000\n')
```

[19]:
```
import pandas as pd

data = pd.read_csv(data_file)
print(data)
```

```
   NumRooms Alley   Price
0       NaN  Pave  127500
1       2.0   NaN  106000
2       4.0   NaN  178100
3       NaN   NaN  140000
```

[20]:
```
inputs, outputs = data.iloc[:, 0:1], data.iloc[:, 2]
inputs = inputs.fillna(inputs.mean())
""" 方法 mean() 是列取均值的意思, """
print(inputs)
```

```
   NumRooms
0       3.0
1       2.0
2       4.0
3       3.0
```

[21]:
```
import numpy as np
x = np.array(3.0)
y = np.array(2.0)
"""numpy 的一些整型、浮点型的数据类型和 python 原生的区别在于更适合批量计算"""
x + y, x * y, x / y, x ** y
```

[21]: (np.float64(5.0), np.float64(6.0), np.float64(1.5), np.float64(9.0))

[22]:
```
x = np.arange(4)
x
```

[22]: array([0, 1, 2, 3])

[23]:
```
x[3]
```

[23]: np.int64(3)

[24]:
```
len(x)
```

[24]: 4

[25]:
```
""" 用张量表示一个向量（一个轴）时, shape 访问向量的长度得到的即是张量沿该轴的长度（维数）"""
x.shape
```

[25]: (4,)

[26]:
```
A = np.arange(20).reshape(5, 4)
A
```

[26]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19]])
```

[27]:
```
""" 矩阵的转置"""
A.T
```

[27]:
```
array([[ 0,  4,  8, 12, 16],
       [ 1,  5,  9, 13, 17],
       [ 2,  6, 10, 14, 18],
       [ 3,  7, 11, 15, 19]])
```

[28]:
```
B = np.array([[1, 2, 3], [2, 0, 4], [3, 4, 5]])
B
```

[28]:
```
array([[1, 2, 3],
       [2, 0, 4],
       [3, 4, 5]])
```

[29]:
```
B == B.T
```

[29]:
```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

[30]:
```
""" 向量是标量的升级，矩阵是向量的升级，张量也是矩阵的升级"""
X = np.arange(24).reshape(2, 3, 4)
X
```

[30]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

[31]:
```
A = np.arange(20).reshape(5, 4)
B = A.copy()
A, A + B
```

[31]:
```
(array([[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11],
        [12, 13, 14, 15],
        [16, 17, 18, 19]]),
 array([[ 0,  2,  4,  6],
        [ 8, 10, 12, 14],
        [16, 18, 20, 22],
        [24, 26, 28, 30],
        [32, 34, 36, 38]]))
```

[32]:
```
A * B
```

[32]:
```
array([[  0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121],
       [144, 169, 196, 225],
       [256, 289, 324, 361]])
```

[33]:
```
a = 2
X = np.arange(24).reshape(2, 3, 4)
a + X, (a * X).shape
```

[33]:
```
(array([[[ 2,  3,  4,  5],
         [ 6,  7,  8,  9],
         [10, 11, 12, 13]],

        [[14, 15, 16, 17],
         [18, 19, 20, 21],
         [22, 23, 24, 25]]]),
 (2, 3, 4))
```

[34]:
```
x = np.arange(4)
x, x.sum()
```

[34]:
```
(array([0, 1, 2, 3]), np.int64(6))
```

[35]:
```
A.shape, A.sum()
```

[35]:
```
((5, 4), np.int64(190))
```

[36]:
```
A_sum_axis0 = A.sum(axis=0)
"""axis 就是选择沿着张量的哪一条轴进行求和的意思（有多少个轴，简单来说就可以通过观察多少层嵌套来确认）"""
A_sum_axis0, A_sum_axis0.shape
```

[36]:
```
(array([40, 45, 50, 55]), (4,))
```

[37]:
```
A_sum_axis1 = A.sum(axis=1)
A_sum_axis1, A_sum_axis1.shape
```

[37]:
```
(array([ 6, 22, 38, 54, 70]), (5,))
```

[38]:
```
A.sum(axis=(0, 1))
```

[38]:
```
np.int64(190)
```

[39]:
```
A.mean(), A.sum() / A.size
```

[39]:
```
(np.float64(9.5), np.float64(9.5))
```

[40]:
```
A, A.shape, A.mean(axis=0), A.sum(axis=0) / A.shape[0]
```

[40]:
```
(array([[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11],
        [12, 13, 14, 15],
        [16, 17, 18, 19]]),
 (5, 4),
 array([ 8.,  9., 10., 11.]),
 array([ 8.,  9., 10., 11.]))
```

[41]:
```
sum_A = A.sum(axis=1, keepdims=True)
"""keepdims 表示是否保持轴数不变"""
sum_A
```

[41]:
```
array([[ 6],
       [22],
       [38],
       [54],
       [70]])
```

[42]:
```
""" 按行计算"""
A.cumsum(axis=0)
```

[42]:
```
array([[ 0,  1,  2,  3],
       [ 4,  6,  8, 10],
       [12, 15, 18, 21],
       [24, 28, 32, 36],
       [40, 45, 50, 55]])
```

[43]:
```
A / sum_A
```

[43]:
```
array([[0.        , 0.16666667, 0.33333333, 0.5       ],
       [0.18181818, 0.22727273, 0.27272727, 0.31818182],
       [0.21052632, 0.23684211, 0.26315789, 0.28947368],
       [0.22222222, 0.24074074, 0.25925926, 0.27777778],
       [0.22857143, 0.24285714, 0.25714286, 0.27142857]])
```

[44]:
```
y = np.ones(4)
x, y, np.dot(x, y)
""" 和数学中一样，向量的乘积（点乘）是对应相乘再相加"""
```

[44]:
```
'和数学中一样，向量的乘积（点乘）是对应相乘再相加'
```

[45]:
```
np.sum(x * y)
```

[45]:
```
np.float64(6.0)
```

[46]:
```
""" 矩阵-向量积的原理和线性代数矩阵的乘法也是一个道理，要求也是一样"""
A.shape, x.shape, np.dot(A, x)
```

[46]:
```
((5, 4), (4,), array([ 14,  38,  62,  86, 110]))
```

[47]:
```
""" 矩阵-矩阵乘法也和线性代数的说法一致"""
B = np.ones(shape=(4, 3))
```

```
np.dot(A, B)
```

[47]:
```
array([[ 6.,  6.,  6.],
       [22., 22., 22.],
       [38., 38., 38.],
       [54., 54., 54.],
       [70., 70., 70.]])
```

[48]:
```
u = np.array([3, -4])
"""L2 范数（更常见）的意思就是数组内部的元素的平方和开根号"""
u, np.linalg.norm(u)
```

[48]:
```
(array([ 3, -4]), np.float64(5.0))
```

[49]:
```
"""L1 范数（不太常见）的意思就是内部元素绝对值相加"""
np.abs(u).sum()
```

[49]:
```
np.int64(7)
```

[50]:
```
"""Frobenius 范数满足向量范数的所有性质，它就像是矩阵形向量的 L2 范数"""
np.linalg.norm(np.ones((4, 9)))
```

[50]:
```
np.float64(6.0)
```

[51]:
```
%matplotlib inline
from matplotlib import pyplot as plt
from matplotlib_inline import backend_inline
def f(x):
    return 3 * x ** 2 - 4 * x

def numerical_lim(f, x, h):
    return (f(x + h) - f(x)) / h

h = 0.1
for i in range(5):
    print(f'h={h:.5f}, numerical limit={numerical_lim(f, 1, h):.5f}')
    h *= 0.1
```

```
h=0.10000, numerical limit=2.30000
```

```
h=0.01000, numerical limit=2.03000
h=0.00100, numerical limit=2.00300
h=0.00010, numerical limit=2.00030
h=0.00001, numerical limit=2.00003
```

[52]:
```
""" 使用 svg 格式在 Jupyter 中显示绘图"""
def use_svg_display():
    backend_inline.set_matplotlib_formats('svg')

""" 设置 matplotlib 的图表大小"""
def set_figsize(figsize=(3.5, 2.5)):
    use_svg_display()
    plt.rcParams['figure.figsize'] = figsize

""" 设置 matplotlib 的轴"""
def set_axes(axes, xlabel, ylabel, xlim, ylim, xscale, yscale, legend):
    axes.set_xlabel(xlabel)
    axes.set_ylabel(ylabel)
    axes.set_xscale(xscale)
    axes.set_yscale(yscale)
    axes.set_xlim(xlim)
    axes.set_ylim(ylim)
    if legend:
        axes.legend(legend)
    axes.grid()

""" 绘制数据点"""
def plot(X, Y=None, xlabel=None, ylabel=None, legend=None, xlim=None,␣
 ↪ylim=None, xscale='linear', yscale='linear', fmts=('-', 'm--', 'g-.', 'r:'),␣
 ↪figsize=(3.5, 2.5), axes=None):
    if legend is None:
        legend = []
    set_figsize(figsize)
    axes = axes if axes else plt.gca()
    # 如果 X 有一个轴, 输出 True
    def has_one_axis(X):
        return (hasattr(X, "ndim") and X.ndim == 1 or isinstance(X, list)
```

```
    and not hasattr(X[0], "__len__"))
    if has_one_axis(X):
        X = [X]
    if Y is None:
        X, Y = [[]] * len(X), X
    elif has_one_axis(Y):
        Y = [Y]
    if len(X) != len(Y):
        X = X * len(Y)
    axes.cla()
    for x, y, fmt in zip(X, Y, fmts):
        if len(x):
            axes.plot(x, y, fmt)
        else:
            axes.plot(y, fmt)
    set_axes(axes, xlabel, ylabel, xlim, ylim, xscale, yscale, legend)
```

[53]:
```
x = np.arange(0, 3, 0.1)
print(x)
plot(x, [f(x), 2 * x - 3], 'x', 'f(x)', legend=['f(x)', 'Tangent line (x=1)'])
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7
 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9]
```