

# Лекция 1 (часть 2): Введение в работу с PyTorch

Автор: Сергей Вячеславович Макрушин e-mail: [SVMakrushin@fa.ru](mailto:SVMakrushin@fa.ru) (<mailto:SVMakrushin@fa.ru>)

Финансовый университет, 2021 г.

При подготовке лекции использованы материалы:

- ...

V 0.5 04.02.2021

## Разделы:

- [Установка PyTorch](#)
- [Тензоры и операции с ними в PyTorch](#)
  - [Создание тензоров](#)
  - [Операции с тензорами](#)
    - [Арифметические операции и математические функции:](#)
    - [Операции, изменяющие размер тензора](#)
    - [Операции агрегации](#)
    - [Матричные операции](#)
- [К оглавлению](#)

```
In [2]: # загружаем стиль для оформления презентации
from IPython.display import HTML
from urllib.request import urlopen
html = urlopen("file:./lec_v2.css")
HTML(html.read().decode('utf-8'))
```

Out[2]:

## Установка PyTorch

- [К оглавлению](#)

---

Современные инструменты для создания моделей на основе ИНС

- TensorFlow / Keras
- PyTorch

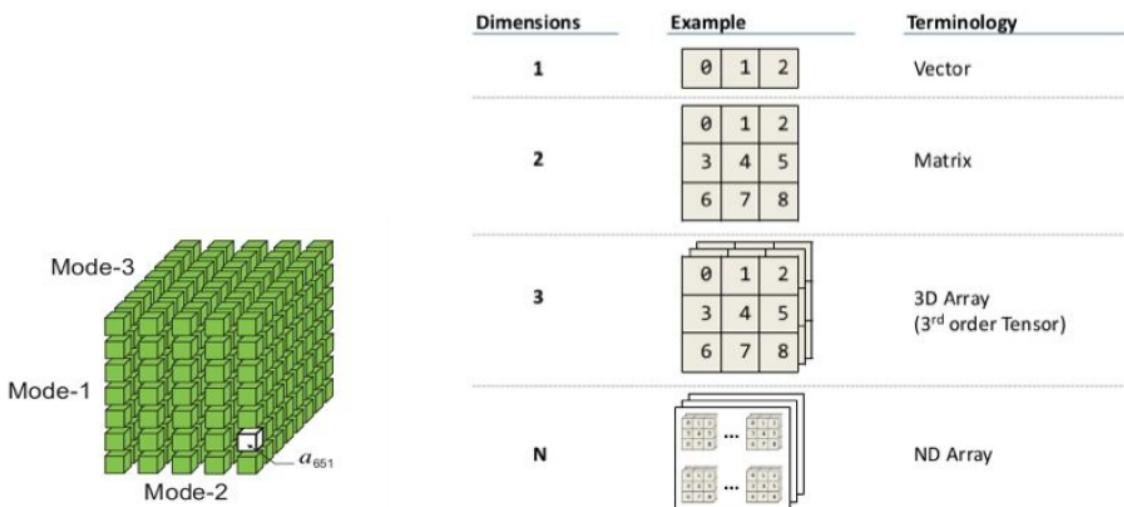
TODO: более полный обзор современного "рынка"

## Тензоры и операции с ними в PyTorch

- [К оглавлению](#)

Что понимается под тензором в TensorFlow, PyTorch и аналогичных инструментах?

- то понимают под тензором в TensorFlow, PyTorch и аналогичных инструментах.
- **Def: Тензор (в линейной алгебре)** — объект линейной алгебры, линейно преобразующий элементы одного линейного пространства в элементы другого. Частными случаями тензоров являются скаляры, векторы, билинейные формы и т. п.
  - Часто тензор представляют как многомерную таблицу  $d \times d \times \dots \times d$ , заполненную числами — компонентами тензора (где  $d$  — размерность векторного пространства, над которым задан тензор, а число размерностей совпадает рангом (валентностью) тензора). В случае ранга 2 запись тензора на письме выглядит как матрица.
  - Запись тензора в виде многомерной таблицы возможна **только после выбора базиса (системы координат)** (кроме скаляров - тензоров размерности 0). Сам тензор как "геометрическая сущность" от выбора базиса не зависит. Это можно наглядно видеть на примере вектора (тензора ранга 1) при смене системы координат:
    - при смене системы координат **компоненты вектора** (и в общем случае - тензора) **меняются** определённым образом.
    - но сам **вектор** — как "геометрическая сущность", образом которого может быть просто направленный отрезок — **при смене системы координат не изменяется**. Это же относится и к общему случаю - тензору.
  - В PyTorch, TensorFlow и аналогичных библиотеках ключевыми объектами являются **тензоры**, но:
    - **это не настоящие тензоры линейной алгебры**, а просто **многомерные таблицы**. В частности:
      - эти тензоры **не предусматривают определение базиса и возможности его изменения**.
    - для тензоров (многомерных таблиц) в PyTorch определены различные операции, важные для построения графа потока вычислений для численномоделирования ИНС и ряда других приложений.
  - Далее под тензорами мы будем иметь в виду то, что под ними понимается в PyTorch, TensorFlow и других аналогичных библиотеках.
  - **Тензоры** в TensorFlow, PyTorch и аналогичных библиотеках в очень многих аспектах **похожи на массивы NumPy**.



### "Тензоры" в TensorFlow и аналогичных инструментах.

Тензоры в TensorFlow по логике использования и интерфейсу очень близки к `ndarray` в NumPy.

- тензор размерности 0 - скаляр
- тензор размерности 1 - вектор (одномерный массив)
- тензор размерности 2 - матрица (двухмерный массив)
- тензор размерности N - N-мерный массив

```
In [4]: import torch  
import numpy as np
```

```
In [5]: # Получение версии PyTorch  
torch.__version__
```

```
Out[5]: '1.13.1'
```

## Создание тензоров

- [к оглавлению](#)

```
In [10]: # В pytorch все основано на операциях с тензорами  
# Тензоры могут иметь:  
# 0 измерений - скаляры  
# 1 измерение - векторы  
# 2 измерения - матрицы  
# 3, 4, ... измерения - тензоры  
  
# Создание не инициализированного тензора: torch.empty(size)  
# Нужно помнить, что перед использованием такого тензора его обязательно нужно инициа-  
  
x = torch.empty(1) # scalar  
print(x)  
x = torch.empty(3) # vector, 1D  
print(x)  
x = torch.empty(2,3) # matrix, 2D  
print(x)  
x = torch.empty(2,2,3) # tensor, 3 dimensions  
print(x)  
x = torch.empty(2,2,2,3) # tensor, 4 dimensions  
print(x)  
  
tensor([1.4013e-45])  
tensor([1.4013e-45, 0.0000e+00, 1.0153e+16])  
tensor([[0., 0., 0.,  
        [0., 0., 0.]])  
tensor([[0., 0., 0.,  
        [0., 0., 0.]],  
  
        [[0., 0., 0.,  
          [0., 0., 0.]]])  
tensor([[[[nan, nan, 1.4013e-45],  
        [0.0000e+00, 2.9764e-42, 0.0000e+00]],  
  
        [[1.1210e-44, 0.0000e+00, 1.5234e+16],  
          [9.0384e-43, 1.5234e+16, 9.0384e-43]]],  
  
        [[[0.0000e+00, 0.0000e+00, 0.0000e+00],  
          [0.0000e+00, 1.5234e+16, 9.0384e-43]],  
  
        [[[2.9764e-42, 0.0000e+00, 1.4013e-45],  
          [0.0000e+00, 3.6310e+12, 9.0384e-43]]]])
```

```
In [11]: # Большинство операций с тензорами очень похожа на операции с массивами NumPy, но часи
x_np = np.empty((2,2,3))
print(x_np)
# x_np = np.empty(2,2,3) # Ошибка!
```

```
[[[1.36943342e-311 3.16202013e-322 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000 1.91824937e-076]]]
```

```
[[8.44898823e+169 2.62100026e+180 4.19936767e+175]
 [1.58871476e-076 1.25892791e-047 3.53604904e-057]]]
```

```
In [12]: # Создание тензора, заполненного случайными значениями (равномерно распределенными в [0, 1])
x = torch.rand(5, 3)
x
```

```
Out[12]: tensor([[0.8976, 0.7096, 0.8698],
 [0.1515, 0.1590, 0.3522],
 [0.7862, 0.4395, 0.8494],
 [0.6433, 0.2314, 0.2237],
 [0.1090, 0.8917, 0.1881]])
```

```
In [13]: # Создание тензоров заполненных:
# нулями:
x = torch.zeros(5, 3)
print(x)
# единицами:
x = torch.ones(5, 3)
print(x)
# тензор с единицами на главной диагонали:
x = torch.eye(5, 3)
print(x)
```

```
tensor([[0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.]])
tensor([[1., 1., 1.],
 [1., 1., 1.],
 [1., 1., 1.],
 [1., 1., 1.],
 [1., 1., 1.]])
tensor([[1., 0., 0.],
 [0., 1., 0.],
 [0., 0., 1.],
 [0., 0., 0.],
 [0., 0., 0.]])
```

```
In [14]: # определение размера тензора:
x.size()
```

```
Out[14]: torch.Size([5, 3])
```

```
In [15]: # для каждого тензора задан тип значений:
print(x.dtype) # тип заданный автоматически

# явное указание типа:
x = torch.zeros(5, 3, dtype=torch.float16)
print(x)

torch.float32
tensor([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]], dtype=torch.float16)
```

Numpy type	dtype		Torch type	Description
int64	torch.int64	torch.float	LongTensor	64 bit integer
int32	torch.int32	torch.int	IntegerTensor	32 bit signed integer
uint8	torch.uint8		ByteTensor	8 bit unsigned integer
float64 double	torch.float64	torch.double	DoubleTensor	64 bit floating point
float32	torch.float32	torch.float	FloatTensor	32 bit floating point
	torch.int16	torch.short	ShortTensor	16 bit signed integer
	torch.int8		CharTensor	6 bit signed integer

### Типы тензоров в PyTorch и массивов в NumPy

```
In [16]: x = torch.Tensor(2, 3) # аналогично torch.empty
x
```

```
Out[16]: tensor([[1.5459e+16, 9.0384e-43, 1.5460e+16],
                  [9.0384e-43, 1.5464e+16, 9.0384e-43]])
```

```
In [17]: # создание тензора из данных:
x = torch.Tensor([[0.6768, 0.5198, 0.6978],
                  [0.1581, 0.2027, 0.3723]])
x
```

```
Out[17]: tensor([[0.6768, 0.5198, 0.6978],
                  [0.1581, 0.2027, 0.3723]])
```

```
In [18]: # создание тензора из данных:
x = torch.tensor([[6, 51, 6],
                  [15, 0, 37]])
print(x, type(x), x.dtype)

x = torch.tensor([[6, 51, 6],
                  [15, 0, 37]], dtype=torch.float64)
print(x, type(x), x.dtype)

tensor([[ 6, 51,  6],
       [15,  0, 37]]) <class 'torch.Tensor'> torch.int64
tensor([[ 6., 51.,  6.],
       [15.,  0., 37.]], dtype=torch.float64) <class 'torch.Tensor'> torch.float64
```

```
In [19]: # создание тензора из массива numpy:  
a = np.array([1, 2, 3])  
x = torch.from_numpy(a)  
x
```

```
Out[19]: tensor([1, 2, 3], dtype=torch.int32)
```

```
In [20]: # Careful: If the Tensor is on the CPU (not the GPU),  
# both objects will share the same memory location, so changing one  
# will also change the other  
  
a[0] += 10  
print(a)  
print(x)
```

```
[11 2 3]  
tensor([11, 2, 3], dtype=torch.int32)
```

```
In [21]: # torch to numpy with .numpy()  
# Специфика использования общего массива данных сохраняется!  
b = x.numpy()  
b
```

```
Out[21]: array([11, 2, 3])
```

```
In [22]: # заполнение тензора значениями:  
x = torch.Tensor(2, 3)  
x.fill_(0.5) # функции, оканчивающиеся на _ меняют значение тензора слева от точки  
x
```

```
Out[22]: tensor([[0.5000, 0.5000, 0.5000],  
                 [0.5000, 0.5000, 0.5000]])
```

Справка по операциям создания тензоров тут: <https://pytorch.org/docs/stable/torch.html#creation-ops>  
(<https://pytorch.org/docs/stable/torch.html#creation-ops>).

---

## Операции с тензорами

- [К оглавлению](#)

### Арифметические операции и математические функции:

- [К оглавлению](#)

```
In [24]: x = torch.rand(2, 2)
print(f'x:\n {x}\n')
y = torch.rand(2, 2)
print(f'y:\n {y}\n')

# поэлементное сложение:
z = x + y
print(f'z = x + y:\n {z}\n')
z = torch.add(x, y)
print(f'z = torch.add(x,y):\n {z}\n')
```

```
x:
tensor([[0.8323, 0.4120],
        [0.3230, 0.0893]])

y:
tensor([[0.0645, 0.6143],
        [0.8915, 0.8552]])

z = x + y:
tensor([[0.8968, 1.0263],
        [1.2145, 0.9445]])

z = torch.add(x,y):
tensor([[0.8968, 1.0263],
        [1.2145, 0.9445]])
```

```
In [26]: y2 = y.clone().detach() # копирование содержимого тензора в новый тензор
print(f'y2:\n {y2}\n')

# операции "in place" (помещают результат в объект слева от точки) в pytorch оканчиваются точкой
y2.add_(x)
print(f'y2.add_(x) in place:\n {y2}\n')
```

```
y2:
tensor([[0.0645, 0.6143],
        [0.8915, 0.8552]])

y2.add_(x) in place:
tensor([[0.8968, 1.0263],
        [1.2145, 0.9445]])
```

In [25]: # Вычитание:

```
z = x - y
z = torch.sub(x, y)
print(f'z = torch.sub(x, y):\n {z}\n')
```

# Умножение (поэлементное):

```
z = x * y
z = torch.mul(x,y)
print(f'z = torch.mul(x,y):\n {z}\n')
```

# Деление:

```
z = x / y
z = torch.div(x,y)
print(f'z = torch.div(x,y):\n {z}\n')
```

```
z = torch.sub(x, y):
tensor([[ 0.7678, -0.2023],
       [-0.5684, -0.7658]])
```

```
z = torch.mul(x,y):
tensor([[0.0537, 0.2531],
       [0.2880, 0.0764]])
```

```
z = torch.div(x,y):
tensor([[12.9068, 0.6707],
       [ 0.3624, 0.1045]])
```

In [27]: print(f'x - y:\n {x - y}\n')

```
z = torch.abs(x - y) # поэлементный расчет модуля
print(f'z = torch.abs(x - y):\n {z}\n')
```

```
z = x - y
z.abs_()
print(f'z.abs_()):\n {z}\n')
```

```
x - y:
tensor([[ 0.7678, -0.2023],
       [-0.5684, -0.7658]])
```

```
z = torch.abs(x - y):
tensor([[0.7678, 0.2023],
       [0.5684, 0.7658]])
```

```
z.abs_():
tensor([[0.7678, 0.2023],
       [0.5684, 0.7658]])
```

In [28]: torch.cos(x) # поэлементный расчет cos

```
# x.cos_()
```

Out[28]: tensor([[0.6732, 0.9163],
 [0.9483, 0.9960]])

In [29]: torch.sigmoid(x) # расчет сигмоиды

```
# x.sigmoid_()
```

Out[29]: tensor([[0.6968, 0.6016],
 [0.5801, 0.5223]])

Справка по математическим операциям тут: <https://pytorch.org/docs/stable/torch.html#math-operations> (<https://pytorch.org/docs/stable/torch.html#math-operations>)

## Операции, изменяющие размер тензора:

- [к оглавлению](#)

```
In [30]: # Операции среза (slicing) (работает аналогично NumPy):  
x = torch.rand(5,3)  
print(x)  
print(x[1, 1]) # элемент с индексом 1, 1 (результат: тензор размерности 0!)  
print(x[:, 0]) # все строки, столбец 0  
print(x[1, :]) # строка 1, все столбцы  
  
tensor([[0.8006, 0.9254, 0.9848],  
        [0.2028, 0.2815, 0.9052],  
        [0.6770, 0.7310, 0.5299],  
        [0.2704, 0.1688, 0.5423],  
        [0.6423, 0.8804, 0.7158]])  
tensor(0.2815)  
tensor([0.8006, 0.2028, 0.6770, 0.2704, 0.6423])  
tensor([0.2028, 0.2815, 0.9052])
```

```
In [27]: # тензор размерности 0 (скаляр), все равно остается типом 'torch.Tensor':  
print(x[1,1], type(x[1,1]))  
  
tensor(0.5943) <class 'torch.Tensor'>
```

```
In [25]: # получение самого значения из тензора размерности 0:  
print(x[1,1].item())  
  
0.6895393133163452
```

```
In [33]: # итерирование по тензору:  
for v in x[1, :]:  
    print(v.item())  
  
0.20279234647750854  
0.2814783453941345  
0.9052202105522156
```

```
In [34]: # Изменение формы тензора (reshape) с помощью torch.view():  
x = torch.randn(4, 4) # матрица 4 на 4  
print(x, x.size(), '\n')  
  
tensor([[-0.8115,  0.7329, -0.7435,  2.6998],  
       [-0.0119, -2.0350,  0.5019,  1.1513],  
       [ 0.9889,  0.9159, -0.0941,  0.9879],  
       [ 0.3529,  1.0743, -0.5272, -0.8844]]) torch.Size([4, 4])
```

```
In [35]: y = x.view(16) # вектор из 16 компонент
print(y, y.size(), '\n')
z = x.view(2, 2, 4) # тензор 2 на 2 на 4
print(z, z.size(), '\n')

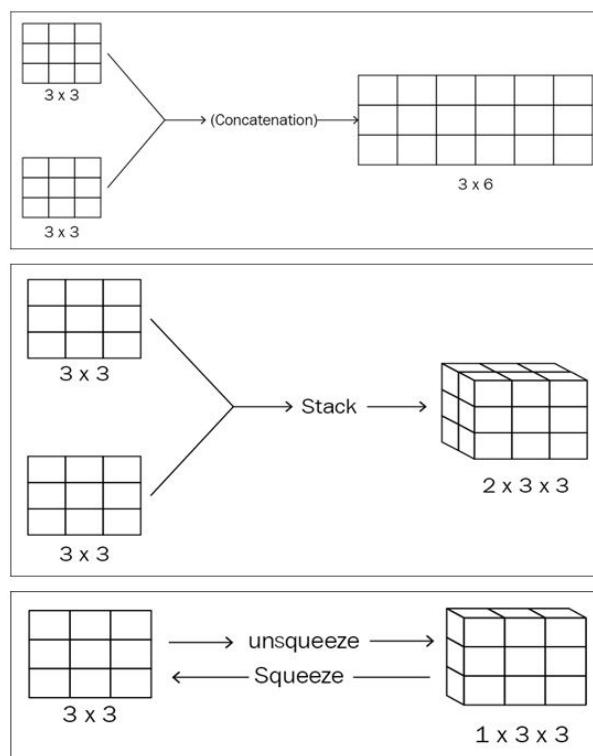
tensor([-0.8115,  0.7329, -0.7435,  2.6998, -0.0119, -2.0350,  0.5019,  1.1513,
       0.9889,  0.9159, -0.0941,  0.9879,  0.3529,  1.0743, -0.5272, -0.8844]) torch.Size([16])

tensor([[[[-0.8115,  0.7329, -0.7435,  2.6998],
          [-0.0119, -2.0350,  0.5019,  1.1513]],

         [[ 0.9889,  0.9159, -0.0941,  0.9879],
          [ 0.3529,  1.0743, -0.5272, -0.8844]]]) torch.Size([2, 2, 4])
```

```
In [31]: t = x.view(-1, 8) # размер -1 означает, что размерность этой компоненты будет подобрена
print(t, t.size())

tensor([[-1.2505,  0.1728, -0.1401,  0.2515,  0.7455, -1.1262,  0.1048, -0.5705],
       [ 1.5765, -0.3304,  0.8056, -2.1390, -1.0988,  1.0378,  1.0964,  1.6614]]) torch.Size([2, 8])
```



### Операции изменения размера матриц

```
In [36]: x1 = torch.rand(2,3)
print(x1)
y1 = torch.rand(2,3)
print(y1)

tensor([[0.9701,  0.8984,  0.9965],
       [0.6626,  0.2660,  0.0166]])
tensor([[0.6247,  0.0287,  0.6157],
       [0.0007,  0.2299,  0.0752]])
```

```
In [42]: # Конкатенация (concatenation):
```

```
print('x1', x1, x1.size())
print('y1', y1, y1.size())

# Concatenates 2 tensors on zeroth dimension:
concat1 = torch.cat((x1, y1))
print(concat1, concat1.size())

# Concatenates 2 tensors on zeroth dimension
x = torch.rand(2,3)
concat2 = torch.cat((x1, y1), dim=0)
print(concat2, concat2.size())

# Concatenates 2 tensors on first dimension
x = torch.rand(2,3)
concat3 = torch.cat((x1, y1), dim=1)
print(concat3, concat3.size())

x1 tensor([[0.9701, 0.8984, 0.9965],
           [0.6626, 0.2660, 0.0166]]) torch.Size([2, 3])
y1 tensor([[0.6247, 0.0287, 0.6157],
           [0.0007, 0.2299, 0.0752]]) torch.Size([2, 3])
tensor([[9.7007e-01, 8.9845e-01, 9.9646e-01, 6.2469e-01, 2.8705e-02, 6.1566e-01],
        [6.6263e-01, 2.6602e-01, 1.6638e-02, 7.1949e-04, 2.2992e-01, 7.5181e-02]]) torch.Size([2, 6])
```

```
In [44]: # Разбиение тензора (split):
```

```
print(x1)

splitted1 = x1.split(split_size=1, dim=0)
print(splitted1, splitted1[0].size())      # 2 tensors of 2x2 and 1x2 size

splitted2 = x1.split(split_size=2, dim=1)
print(splitted2[0], splitted2[0].size(), '\n', splitted2[1], splitted2[1].size())

tensor([[0.9701, 0.8984, 0.9965],
        [0.6626, 0.2660, 0.0166]])
(tensor([[0.9701, 0.8984, 0.9965]]), tensor([[0.6626, 0.2660, 0.0166]])) torch.Size
([1, 3])
```

```
In [45]: # stack:
```

```
print(x1)
print(y1)

stacked1 = torch.stack((x1, y1), dim=0)
print(stacked1, stacked1.size()) # Возвращаем тензор: 2(в результате стак!) x 2 x 3

tensor([[0.9701, 0.8984, 0.9965],
        [0.6626, 0.2660, 0.0166]])
tensor([[0.6247, 0.0287, 0.6157],
        [0.0007, 0.2299, 0.0752]])
tensor([[9.7007e-01, 8.9845e-01, 9.9646e-01],
        [6.6263e-01, 2.6602e-01, 1.6638e-02]],

       [[6.2469e-01, 2.8705e-02, 6.1566e-01],
        [7.1949e-04, 2.2992e-01, 7.5181e-02]]) torch.Size([2, 2, 3])
```

```
In [36]: stacked2 = torch.stack((x1, y1), dim=1)
print(stacked2, stacked2.size()) # Возвращаем тензор: 2 x 2(в результате stack!) x 3

tensor([[ [0.3964, 0.6470, 0.5792],
          [0.0626, 0.9867, 0.1250]],

         [[0.1353, 0.8764, 0.6003],
          [0.7733, 0.3105, 0.3181]]]) torch.Size([2, 2, 3])
```

```
In [46]: #squeeze and unsqueeze
x2 = torch.rand(3, 2, 1) # a tensor of size 3x2x1
print(x2, x2.size())
squeezed1 = x2.squeeze()
print(squeezed1, squeezed1.size()) # remove the 1 sized dimension

tensor([[[0.9539],
          [0.2437]],

         [[0.9619],
          [0.6110]],

         [[0.2962],
          [0.2813]]]) torch.Size([3, 2, 1])
tensor([[0.9539, 0.2437],
        [0.9619, 0.6110],
        [0.2962, 0.2813]]) torch.Size([3, 2])
```

```
In [38]: x3 = torch.rand(3)
print(x3)

with_fake_dimension1 = x3.unsqueeze(0)
print(with_fake_dimension1, with_fake_dimension1.size()) # added a fake zeroth dimension

with_fake_dimension2 = x3.unsqueeze(1)
print(with_fake_dimension2, with_fake_dimension2.size()) # added a fake zeroth dimension

tensor([0.3728, 0.6864, 0.4883])
tensor([[0.3728, 0.6864, 0.4883]]) torch.Size([1, 3])
tensor([[0.3728],
        [0.6864],
        [0.4883]]) torch.Size([3, 1])
```

```
In [47]: # Распространение (broadcasting) - так же как в NumPy:

t1 = torch.arange(1.0, 5.0)
t3 = torch.arange(0.0, 3.0)
print(t1, t3)
tm = t1.ger(t3)
print(tm, tm.size())

tensor([1., 2., 3., 4.]) tensor([0., 1., 2.])
tensor([[0., 1., 2.],
        [0., 2., 4.],
        [0., 3., 6.],
        [0., 4., 8.]]) torch.Size([4, 3])
```

```
In [48]: t1 * tm
```

```
-----  
RuntimeError                                     Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_18664\1190549674.py in <module>  
----> 1 t1 * tm
```

**RuntimeError:** The size of tensor a (4) must match the size of tensor b (3) at non-singleton dimension 1

```
In [49]: t1.size(), tm.size()
```

```
Out[49]: (torch.Size([4]), torch.Size([4, 3]))
```

```
In [50]: print(t1.unsqueeze(1))  
print(tm)
```

```
tensor([[1.],  
       [2.],  
       [3.],  
       [4.]])  
tensor([[0., 1., 2.],  
       [0., 2., 4.],  
       [0., 3., 6.],  
       [0., 4., 8.]])
```

```
In [43]: t1.unsqueeze(1) * tm
```

```
Out[43]: tensor([[ 0.,  1.,  2.],  
                  [ 0.,  4.,  8.],  
                  [ 0.,  9., 18.],  
                  [ 0., 16., 32.]])
```

```
In [44]: t1.unsqueeze(1).size(), tm.size()
```

```
Out[44]: (torch.Size([4, 1]), torch.Size([4, 3]))
```

## Операции агрегации:

- [К оглавлению](#)

```
In [45]: mat = torch.tensor(  
                      [[0., 1., 2.],  
                       [0., 2., 4.],  
                       [0., 3., 6.],  
                       [0., 4., 8.]])
```

```
# суммирование по всем элементам:  
mat.sum()
```

```
Out[45]: tensor(30.)
```

```
In [46]: # суммирование по оси 0:  
mat.sum(dim=0)
```

```
Out[46]: tensor([ 0., 10., 20.])
```

```
In [55]: # суммирование по оси 1:  
mat.sum(dim=1)
```

```
Out[55]: tensor([ 3.,  6.,  9., 12.])
```

```
In [56]: # получение среднего значения:  
  
print(mat.mean())  
print(mat.mean(dim=0))  
print(mat.mean(dim=1))
```

```
tensor(2.5000)  
tensor([0.0000, 2.5000, 5.0000])  
tensor([1., 2., 3., 4.])
```

## Матричные операции:

- [к оглавлению](#)

```
In [57]: # Умножение (поэлементное):
```

```
x = torch.rand(2, 2)  
print(f'x:\n {x}\n')  
y = torch.rand(2, 2)  
print(f'y:\n {y}\n')  
  
z = x * y  
print(f'z = torch.mul(x,y):\n {z}\n')  
z = torch.mul(x,y)  
print(f'z = torch.mul(x,y):\n {z}\n')
```

```
x:  
tensor([[0.7255, 0.9459],  
       [0.0180, 0.9965]])
```

```
y:  
tensor([[0.5618, 0.6862],  
       [0.2016, 0.1258]])
```

```
z = torch.mul(x,y):  
tensor([[0.4076, 0.6491],  
       [0.0036, 0.1253]])
```

```
z = torch.mul(x,y):  
tensor([[0.4076, 0.6491],  
       [0.0036, 0.1253]])
```

In [58]: # Умножение матрицы на вектор:

```
# torch.mv(input, vec, out=None) → Tensor
# Performs a matrix-vector product of the matrix input and the vector vec.
# If input is a (n \times m)(n\times m) tensor, vec is a 1-D tensor of size mm , out will be a 1-D tensor of size nn.

mat = torch.randn(2, 3)
print(mat, mat.size())
vec = torch.randn(3)
print(vec, vec.size())
res = torch.mv(mat, vec)
print(res, res.size())

tensor([[ 0.2450, -0.9958,  0.3320],
        [-1.1015,  1.3226,  3.2153]]) torch.Size([2, 3])
tensor([-0.0979,  2.0161, -0.1482]) torch.Size([3])
tensor([-2.0808,  2.2977]) torch.Size([2])
```

In [59]: # Умножение матрицы на матрицу:

```
# torch.mm(input, mat2, out=None) → Tensor
# Performs a matrix multiplication of the matrices input and mat2.
# If input is a (nxm) tensor, mat2 is a (mxp) tensor, out will be a (nxp) tensor.

mat1 = torch.randn(2, 3)
print(mat1, mat1.size())
mat2 = torch.randn(3, 3)
print(mat2, mat2.size())
res = torch.mm(mat1, mat2)
print(res, res.size())

tensor([[ -1.6114,   0.1754,  -0.1479],
        [ -0.8400,  -1.4489,  -0.1735]]) torch.Size([2, 3])
tensor([[ -0.2815,   0.7116,  -1.1436],
        [  0.1062,   0.8233,   1.0840],
        [  0.4111,   1.1746,  -0.1564]]) torch.Size([3, 3])
tensor([[  0.4115,  -1.1759,   2.0561],
        [  0.0113,  -1.9944,  -0.5829]]) torch.Size([2, 3])
```

In [60]: mat1.mm(mat2)

Out[60]: tensor([[ 0.4115, -1.1759, 2.0561],
 [ 0.0113, -1.9944, -0.5829]])

In [61]: # Outer product of 2 vectors

```
vec1 = torch.arange(1, 4)    # Size 3
print(vec1, vec1.size())
vec2 = torch.arange(1, 3)    # Size 2
print(vec2, vec2.size())
res = torch.ger(vec1, vec2) # vec1 - рассматривается как вектор-столбец; vec2 - рассматривается как вектор-строка
print(res, res.size()) # Size 3x2

tensor([1, 2, 3]) torch.Size([3])
tensor([1, 2]) torch.Size([2])
tensor([[1, 2],
        [2, 4],
        [3, 6]]) torch.Size([3, 2])
```

In [62]: vec1.ger(vec2)

Out[62]: tensor([[1, 2],
 [2, 4],
 [3, 6]])

## Функция matmul

- Matrix product of two tensors: `torch.matmul(input, other, out=None) → Tensor`
- The behavior depends on the dimensionality of the tensors as follows:
  - If **both tensors are 1-dimensional**, the **dot product (scalar) is returned**.
  - If **both arguments are 2-dimensional**, the **matrix-matrix product is returned**.
  - If the **first argument is 1-dimensional and the second argument is 2-dimensional**, a 1 is prepended to its dimension for the purpose of the matrix multiply. After the matrix multiply, the prepended dimension is removed.
  - If the **first argument is 2-dimensional and the second argument is 1-dimensional**, the matrix-vector product is returned.
  - If **both arguments are at least 1-dimensional and at least one argument is N-dimensional (where N > 2)**, then a batched matrix multiply is returned. If the first argument is 1-dimensional, a 1 is prepended to its dimension for the purpose of the batched matrix multiply and removed after. If the second argument is 1-dimensional, a 1 is appended to its dimension for the purpose of the batched matrix multiple and removed after. The non-matrix (i.e. batch) dimensions are broadcasted (and thus must be broadcastable). For example, if input is a  $j \times 1 \times n \times m$  tensor and other is a  $k \times m \times p$  tensor, out will be an  $j \times k \times n \times p$  tensor.

```
In [63]: torch.arange(4)
```

```
Out[63]: tensor([0, 1, 2, 3])
```

```
In [64]: torch.arange(3*4).view(3, 4)
```

```
Out[64]: tensor([[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7],
                 [ 8,  9, 10, 11]])
```

```
In [65]: # vector x vector
tensor1 = torch.randn(3)
print(tensor1, tensor1.size())
tensor2 = torch.randn(3)
print(tensor2, tensor2.size())
res = torch.matmul(tensor1, tensor2)
print(res, res.size()) # результат: скаляр
```

```
tensor([-2.5219, -1.2146,  1.3170]) torch.Size([3])
tensor([ 0.2303,  0.9512, -0.0546]) torch.Size([3])
tensor(-1.8081) torch.Size([])
```

```
In [66]: # Вызов функции matmul можно выполнять с помощью оператора @:
tensor1 @ tensor2
```

```
Out[66]: tensor(-1.8081)
```

```
In [72]: # vector x matrix
tensor1 = torch.arange(3*4).view(3, 4)
print(tensor1, tensor1.size())
tensor2 = torch.arange(3)
print(tensor2, tensor2.size())

res = torch.matmul(tensor2, tensor1)
print(res, res.size())

tensor([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]]) torch.Size([3, 4])
tensor([0, 1, 2]) torch.Size([3])
tensor([20, 23, 26, 29]) torch.Size([4])
```

```
In [71]: # unu:
tensor2 @ tensor1
```

```
Out[71]: tensor([20, 23, 26, 29])
```

```
In [279]: # matrix x vector
tensor1 = torch.arange(3*4).view(3, 4)
print(tensor1, tensor1.size())
tensor2 = torch.arange(4)
print(tensor2, tensor2.size())

res = torch.matmul(tensor1, tensor2)
print(res, res.size())

tensor([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]]) torch.Size([3, 4])
tensor([0, 1, 2, 3]) torch.Size([4])
tensor([14, 38, 62]) torch.Size([3])
```

```
In [68]: tensor1 @ tensor2
```

```
Out[68]: tensor([14, 38, 62])
```

In [73]: # batched matrix x broadcasted vector

```
tensor1 = torch.randn(10, 3, 4)
print(tensor1, tensor1.size(), '\n-----')
tensor2 = torch.randn(4)
print(tensor2, tensor2.size(), '\n-----')
res = torch.matmul(tensor1, tensor2)
print(res, res.size())

tensor([[[ -0.0254,   0.7122,  -2.5060,   0.5180],
         [ 0.1458,  -0.8855,  -0.8308,   1.5698],
         [-0.7911,   0.8250,  -0.9246,  -0.1922]],

        [[-0.3259,  -0.8213,   1.5900,  -0.1392],
         [-0.5806,  -0.1336,  -2.4994,  -0.2150],
         [-0.1507,  -1.1597,   0.4157,   0.5377]],

        [[ 0.1935,  -0.6100,  -0.0840,  -0.2509],
         [ 0.2435,   0.0852,  -0.7656,  -0.5838],
         [-1.1525,   1.2272,  -0.6801,   1.1422]],

        [[ 1.5298,   0.4393,   1.0724,  -1.3998],
         [ 0.5910,   1.0698,   0.0492,  -0.4684],
         [-0.5860,   0.3775,  -0.4975,   0.1747]],

        [[ 0.6234,  -1.6125,   0.0581,  -0.9023],
         [ 0.8492,   0.4678,   2.2095,  -0.4018],
         [-0.4183,   1.1057,   0.4946,   0.2117]],

        [[-0.4135,  -1.1620,  -0.4104,   0.7465],
         [-1.5399,   0.3171,   0.6196,  -1.0964],
         [-1.4138,  -0.7486,  -0.2011,  -0.4922]],

        [[-0.1190,   0.5037,   1.6496,   0.6041],
         [ 0.0585,   1.0287,  -0.0672,   0.6690],
         [-0.2793,  -0.8093,  -1.3930,  -0.2564]],

        [[-0.2652,   0.3194,   0.1951,   0.4310],
         [-0.9333,  -2.0235,  -0.0407,   1.4903],
         [-0.2456,  -0.3495,   0.5053,   0.7812]],

        [[ 0.5721,  -0.2114,  -2.9636,   0.6201],
         [-1.0270,   0.1474,   0.7315,  -0.6116],
         [ 1.0629,   0.8071,   0.5145,   0.4959]],

        [[ 0.0760,  -0.5637,  -0.7976,   0.0616],
         [-1.1659,  -1.4374,  -0.7419,   0.4414],
         [-0.4875,  -0.7246,   0.1285,  -0.3838]]]) torch.Size([10, 3, 4])

-----
tensor([ 0.1706,  -0.5846,   0.1843,  -1.3673]) torch.Size([4])

-----
tensor([[-1.5908,  -1.7570,  -0.5249],
       [ 0.9079,  -0.1876,  -0.0064],
       [ 0.7172,   0.6489,  -2.6011],
       [ 2.1158,   0.1250,  -0.6511],
       [ 2.2935,   0.8279,  -0.9160],
       [-0.4876,   1.1653,   0.8324],
       [-0.8368,  -1.5186,   0.5194],
       [-0.7854,  -1.0215,  -0.8126],
       [-1.1729,   0.7098,  -0.8738],
       [ 0.1113,  -0.0989,   0.8890]]) torch.Size([10, 3])
```

In [74]: # batched matrix x batched matrix

```
tensor1 = torch.randn(10, 3, 4)
print(tensor1.size(), '\n-----')
tensor2 = torch.randn(10, 4, 5)
print(tensor2.size(), '\n-----')
res = torch.matmul(tensor1, tensor2)
print(res.size())
```

```
torch.Size([10, 3, 4])
-----
```

```
torch.Size([10, 4, 5])
-----
```

```
torch.Size([10, 3, 5])
```

In [75]: # batched matrix x broadcasted matrix

```
tensor1 = torch.randn(10, 3, 4)
print(tensor1.size(), '\n-----')
tensor2 = torch.randn(4, 5)
print(tensor2, tensor2.size(), '\n-----')
res = torch.matmul(tensor1, tensor2)
print(res.size())
```

```
torch.Size([10, 3, 4])
-----
```

```
tensor([[ 2.3362,  0.1706,  0.4528,  0.8596,  1.0453],
       [-0.0351,  0.2272, -0.4056, -0.2502,  0.9134],
       [-1.5136,  0.4217,  1.3492,  0.6766,  0.9003],
       [-1.6946, -1.3057,  1.5708,  0.5506, -2.0987]]) torch.Size([4, 5])
```

```
-----
```

```
torch.Size([10, 3, 5])
```

## Спасибо за внимание!

### Технический раздел:

- И Введение в искусственные нейронные сети
  - Базовые понятия и история
- И Машинное обучение и концепция глубокого обучения
- И Почему глубокое обучение начало приносить плоды и активно использоваться только после 2010 г?
  - Производительность оборудования
  - Доступность наборов данных и тестов
  - Алгоритмические достижения в области глубокого обучения
    - Улучшенные подходы к регуляризации
    - Улучшенные схемы инициализации весов
    - (повтор) Усовершенствованные методы градиентного спуска
- Обратное распространение ошибки
  - Оптимизация
    - Стохастический градиентный спуск
    - Усовершенствованные методы градиентного спуска
- Введение в PyTorch

next **Q:** qs line

next **A:** an line

next **Note:** an line

next **Def:** df line

next **Ex:** ex line

next **+ pl** line

next **- mn** line

next **± plmn** line

next **⇒ hn** line