```
import torch
```

```
# Создайте случайный FloatTensor размера 3x4x5
x = torch.randn((3, 4, 5))
x
```

```
tensor([[[-0.1850, -1.8555, -0.9546,  0.1008, -0.3644],
         [ 1.2624,  0.5157,  0.2824, -1.6159, -0.3430],
         [ 0.3378,  1.8808, -0.7690, -0.7260, -1.4706],
         [-1.6866, -0.2013,  0.1146, -0.2078, -0.9165]],

        [[ 0.2289,  0.9801, -0.3466, -1.3388,  1.3150],
         [ 0.1122, -0.3463,  0.6059,  0.3741, -0.0691],
         [-1.0852,  0.9899,  0.6489,  0.1055, -1.7396],
         [-1.0937,  0.0719,  0.1627,  0.0991,  0.4062]],

        [[-0.7617,  1.2916,  1.0311, -0.8704,  0.2370],
         [-0.3298, -0.2298,  0.6691, -1.6778,  0.4143],
         [ 0.9262, -0.4755,  0.1134,  1.7685, -0.8650],
         [-0.3439, -0.5263,  0.2090,  0.6127,  0.2349]]])
```

```
# Выведите его форму (shape)
x.shape
```

```
torch.Size([3, 4, 5])
```

```
# Приведите его к форме 6 X 10
x = x.reshape(6, 10)
x
```

```
tensor([[-0.1850, -1.8555, -0.9546,  0.1008, -0.3644,  1.2624,  0.5157,  0.28
         -1.6159, -0.3430],
        [ 0.3378,  1.8808, -0.7690, -0.7260, -1.4706, -1.6866, -0.2013,  0.11
         -0.2078, -0.9165],
        [ 0.2289,  0.9801, -0.3466, -1.3388,  1.3150,  0.1122, -0.3463,  0.60
          0.3741, -0.0691],
        [-1.0852,  0.9899,  0.6489,  0.1055, -1.7396, -1.0937,  0.0719,  0.16
          0.0991,  0.4062],
        [-0.7617,  1.2916,  1.0311, -0.8704,  0.2370, -0.3298, -0.2298,  0.66
         -1.6778,  0.4143],
        [ 0.9262, -0.4755,  0.1134,  1.7685, -0.8650, -0.3439, -0.5263,  0.20
          0.6127,  0.2349]])
```

```
# Умножьте его на вектор [1, 4, 2, 2, 1] поэлементно
y = torch.Tensor([1, 4, 2, 2, 1])
res = (x.reshape(-1, 1) * y.reshape(1, -1)).reshape(6, 10, 5)
res
```

```
         [ 0.2824,  1.1296,  0.5648,  0.5648,  0.2824],
         [-1.6159, -6.4635, -3.2317, -3.2317, -1.6159],
         [-0.3430, -1.3721, -0.6861, -0.6861, -0.3430]],

        [[ 0.3378,  1.3511,  0.6756,  0.6756,  0.3378],
         [ 1.8808,  7.5232,  3.7616,  3.7616,  1.8808],
```

```
 [ 1.8808,  7.5232,  3.7616,  3.7616,  1.8808],
 [-0.7690, -3.0762, -1.5381, -1.5381, -0.7690],
 [-0.7260, -2.9041, -1.4521, -1.4521, -0.7260],
 [-1.4706, -5.8823, -2.9411, -2.9411, -1.4706],
 [-1.6866, -6.7465, -3.3732, -3.3732, -1.6866],
 [-0.2013, -0.8053, -0.4027, -0.4027, -0.2013],
 [ 0.1146,  0.4586,  0.2293,  0.2293,  0.1146],
 [-0.2078, -0.8312, -0.4156, -0.4156, -0.2078],
 [-0.9165, -3.6659, -1.8329, -1.8329, -0.9165]],

[[ 0.2289,  0.9157,  0.4579,  0.4579,  0.2289],
 [ 0.9801,  3.9203,  1.9601,  1.9601,  0.9801],
 [-0.3466, -1.3865, -0.6932, -0.6932, -0.3466],
 [-1.3388, -5.3552, -2.6776, -2.6776, -1.3388],
 [ 1.3150,  5.2601,  2.6301,  2.6301,  1.3150],
 [ 0.1122,  0.4487,  0.2244,  0.2244,  0.1122],
 [-0.3463, -1.3851, -0.6926, -0.6926, -0.3463],
 [ 0.6059,  2.4238,  1.2119,  1.2119,  0.6059],
 [ 0.3741,  1.4965,  0.7482,  0.7482,  0.3741],
 [-0.0691, -0.2766, -0.1383, -0.1383, -0.0691]],

[[-1.0852, -4.3407, -2.1704, -2.1704, -1.0852],
 [ 0.9899,  3.9598,  1.9799,  1.9799,  0.9899],
 [ 0.6489,  2.5957,  1.2979,  1.2979,  0.6489],
 [ 0.1055,  0.4220,  0.2110,  0.2110,  0.1055],
 [-1.7396, -6.9585, -3.4793, -3.4793, -1.7396],
 [-1.0937, -4.3750, -2.1875, -2.1875, -1.0937],
 [ 0.0719,  0.2877,  0.1439,  0.1439,  0.0719],
 [ 0.1627,  0.6509,  0.3254,  0.3254,  0.1627],
 [ 0.0991,  0.3966,  0.1983,  0.1983,  0.0991],
 [ 0.4062,  1.6249,  0.8124,  0.8124,  0.4062]],

[[-0.7617, -3.0466, -1.5233, -1.5233, -0.7617],
 [ 1.2916,  5.1663,  2.5831,  2.5831,  1.2916],
 [ 1.0311,  4.1246,  2.0623,  2.0623,  1.0311],
 [-0.8704, -3.4816, -1.7408, -1.7408, -0.8704],
 [ 0.2370,  0.9479,  0.4739,  0.4739,  0.2370],
 [-0.3298, -1.3191, -0.6596, -0.6596, -0.3298],
 [-0.2298, -0.9190, -0.4595, -0.4595, -0.2298],
 [ 0.6691,  2.6764,  1.3382,  1.3382,  0.6691],
 [-1.6778, -6.7112, -3.3556, -3.3556, -1.6778],
 [ 0.4143,  1.6574,  0.8287,  0.8287,  0.4143]],

[[ 0.9262,  3.7048,  1.8524,  1.8524,  0.9262],
 [-0.4755, -1.9020, -0.9510, -0.9510, -0.4755],
 [ 0.1134,  0.4536,  0.2268,  0.2268,  0.1134],
 [ 1.7685,  7.0742,  3.5371,  3.5371,  1.7685],
 [-0.8650, -3.4600, -1.7300, -1.7300, -0.8650],
 [-0.3439, -1.3756, -0.6878, -0.6878, -0.3439],
 [-0.5263, -2.1052, -1.0526, -1.0526, -0.5263],
 [ 0.2090,  0.8359,  0.4179,  0.4179,  0.2090],
 [ 0.6127,  2.4509,  1.2255,  1.2255,  0.6127],
 [ 0.2349,  0.9394,  0.4697,  0.4697,  0.2349]]])
```

```
# Умножьте тензор матрично на себя, чтобы результат был размерности 6x6
(x @ x.T).shape
```

```
torch.Size([6, 6])
```

```
# Посчитайте производную функции y = x**3 + z - 75t в точке (1, 0.5, 2)
from torch.autograd import Variable

x = Variable(torch.Tensor([1]), requires_grad=True)
z = Variable(torch.Tensor([0.5]), requires_grad=True)
t = Variable(torch.Tensor([2]), requires_grad=True)

y = x ** 3 + z - 75 * t

y.backward()

print(f"Значение производной в заданной точке:\n\
        ({float(x.grad)}, {float(z.grad)}, {float(t.grad)})")
```

```
Значение производной в заданной точке:
        (3.0, 1.0, -75.0)
```

```
# Создайте единичный тензор размера 5x6
x = torch.ones((5, 6))
x
```

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

```
# Переведите его в формат numpy
x.numpy()
```

```
array([[1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.]], dtype=float32)
```

```
# Давайте теперь пооптимизируем: возьмите функцию y = x**w1 - 2 * x**2 + 5
# Посчитайте
```

```
from torch import nn
from torch import optim
```

```
w1 = Variable(torch.randn(1), requires_grad=True)
x = Variable(torch.Tensor([2]), requires_grad=False)
y = Variable(torch.Tensor([4]), requires_grad=False)

def func(x, w1):
    return x ** w1 - 2 * x ** 2 + 5

print(y, x, w1)
```

```
tensor([4.]) tensor([2.]) tensor([-1.8543], requires_grad=True)
```

```
criterion = nn.MSELoss()
optimizer = torch.optim.SGD([w1], lr=0.001)
```

```
for epoch in range(201):
    optimizer.zero_grad()
    pred = func(x, w1)
    loss = criterion(pred, y)
    if epoch % 25 == 0:
        print(f"epoch {epoch}: loss {float(loss)}")
    loss.backward()
    optimizer.step()
```

```
epoch 0: loss 45.20446014404297
epoch 25: loss 45.03105545043945
epoch 50: loss 44.841468811035156
epoch 75: loss 44.63336181640625
epoch 100: loss 44.40394592285156
epoch 125: loss 44.149845123291016
epoch 150: loss 43.86696243286133
epoch 175: loss 43.55023193359375
epoch 200: loss 43.19338607788086
```

```
print(y, x, w1)
```

```
tensor([4.]) tensor([2.]) tensor([-1.2210], requires_grad=True)
```

```

```

✓ 0 сек. выполнено в 15:06