```
import torch
import numpy as np



# Создайте случайный FloatTensor размера 3x4x5


def print_tens_info(tensor):
    print("X :\n%s\n" % tensor)
    print("X количество измерений:\n%s\n" % tensor.dim())
    print("X размеры : ",  tensor.size())
    print("X тип : %s\n" % (tensor.type()))


x = torch.arange(0,60).view(3,4,5).type(torch.float32)


print_tens_info(x)
```

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

        [[20., 21., 22., 23., 24.],
         [25., 26., 27., 28., 29.],
         [30., 31., 32., 33., 34.],
         [35., 36., 37., 38., 39.]],

        [[40., 41., 42., 43., 44.],
         [45., 46., 47., 48., 49.],
         [50., 51., 52., 53., 54.],
         [55., 56., 57., 58., 59.]]])

X количество измерений:
3

X размеры :  torch.Size([3, 4, 5])
X тип : torch.FloatTensor
```

```
# Выведите его форму (shape)


x.shape
```

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

```
# Приведите его к форме 6 X 10


xx = x.view(6,10)
xx
```

```
tensor([[ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.],
        [10., 11., 12., 13., 14., 15., 16., 17., 18., 19.],
        [20., 21., 22., 23., 24., 25., 26., 27., 28., 29.],
        [30., 31., 32., 33., 34., 35., 36., 37., 38., 39.],
        [40., 41., 42., 43., 44., 45., 46., 47., 48., 49.],
        [50., 51., 52., 53., 54., 55., 56., 57., 58., 59.]])
```

```
# Умножьте его на вектор [1, 4, 2, 2, 1] поэлементно
```

```
a = torch.FloatTensor([1, 4, 2, 2, 1])
x * a
```

```
tensor([[[  0.,   4.,   4.,   6.,   4.],
         [  5.,  24.,  14.,  16.,   9.],
         [ 10.,  44.,  24.,  26.,  14.],
         [ 15.,  64.,  34.,  36.,  19.]],

        [[ 20.,  84.,  44.,  46.,  24.],
         [ 25., 104.,  54.,  56.,  29.],
         [ 30., 124.,  64.,  66.,  34.],
         [ 35., 144.,  74.,  76.,  39.]],

        [[ 40., 164.,  84.,  86.,  44.],
         [ 45., 184.,  94.,  96.,  49.],
         [ 50., 204., 104., 106.,  54.],
         [ 55., 224., 114., 116.,  59.]]])
```

```
a = torch.FloatTensor([1, 4, 2, 2, 1])

print(torch.mv(x[0],a))
print(torch.mv(x[1],a))
print(torch.mv(x[2],a))
```

```
tensor([ 18.,  68., 118., 168.])
tensor([218., 268., 318., 368.])
tensor([418., 468., 518., 568.])
```

```
#Умножьте тензор матрично на себя, чтобы результат был размерности 6x6
```

```
print(torch.mm(xx,xx.T))
```

```
tensor([[  285.,   735.,  1185.,  1635.,  2085.,  2535.],
        [  735.,  2185.,  3635.,  5085.,  6535.,  7985.],
        [ 1185.,  3635.,  6085.,  8535., 10985., 13435.],
        [ 1635.,  5085.,  8535., 11985., 15435., 18885.],
        [ 2085.,  6535., 10985., 15435., 19885., 24335.],
        [ 2535.,  7985., 13435., 18885., 24335., 29785.]])
```

```
# Посчитайте производную функции y = x**3 + z - 75t в точке (1, 0.5, 2)
```

```
from torch.autograd import Variable
```

```
v = Variable(torch.from_numpy(np.array([1, 0.5, 2], dtype="float32")))
x = Variable(v, requires_grad = True)
print_tens_info(x)
```

```
X :
tensor([1.0000, 0.5000, 2.0000], requires_grad=True)

X количество измерений:
1

X размеры :  torch.Size([3])
X тип : torch.FloatTensor
```

```
print(x.grad)
```

```
None
```

```
y = x**3 + x- 75*x
```

```
y.backward(x)
```

```
print(x.grad)
```

```
tensor([ -71.0000,  -36.6250, -124.0000])
```

```
# Создайте единичный тензор размера 5x6
```

```
# torch.ones(5,6)
x = Variable(torch.ones(5,6), requires_grad=True)
print_tens_info(x)
```

```
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.]], requires_grad=True)

X количество измерений:
2

X размеры :  torch.Size([5, 6])
X тип : torch.FloatTensor
```

```
# Переведите его в формат numpy
```

```
# xa = x.numpy()
xa = x.detach().numpy()
```

```
print(type(xa))
print(xa)
```

```
<class 'numpy.ndarray'>
[[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.]]
```

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


from torch import nn
from torch import optim


linear = nn.Linear(6,5, bias=True)



print ('w: ', linear.weight)
print ('b: ', linear.bias)
```

```
w:  Parameter containing:
tensor([[-0.1129,  0.0189,  0.2024, -0.0260,  0.1211,  0.3760],
        [-0.2163,  0.2257, -0.3940,  0.0936, -0.2161, -0.3231],
        [-0.0043,  0.1368, -0.1738,  0.0466,  0.2290,  0.0283],
        [-0.0658, -0.1550, -0.1657,  0.2981,  0.0845, -0.2331],
        [-0.2391, -0.2676, -0.1670,  0.3814, -0.3206,  0.0161]],
       requires_grad=True)
b:  Parameter containing:
tensor([-0.1915, -0.3689,  0.0805,  0.1289,  0.3049], requires_grad=True)
```

```
w1 = linear.weight
w1
```

```
Parameter containing:
tensor([[-0.1129,  0.0189,  0.2024, -0.0260,  0.1211,  0.3760],
        [-0.2163,  0.2257, -0.3940,  0.0936, -0.2161, -0.3231],
        [-0.0043,  0.1368, -0.1738,  0.0466,  0.2290,  0.0283],
        [-0.0658, -0.1550, -0.1657,  0.2981,  0.0845, -0.2331],
        [-0.2391, -0.2676, -0.1670,  0.3814, -0.3206,  0.0161]],
       requires_grad=True)
```

```
y = x**w1 - 2 * x**2 + 5


print(x.grad)
```

```
None
```

```
y.backward(x)


print(x.grad)
```

```
     tensor([[-4.1129, -3.9811, -3.7976, -4.0260, -3.8789, -3.6240],
             [-4.2163, -3.7743, -4.3940, -3.9064, -4.2161, -4.3231],
             [-4.0043, -3.8632, -4.1738, -3.9534, -3.7710, -3.9717],
             [-4.0658, -4.1550, -4.1657, -3.7019, -3.9155, -4.2331],
             [-4.2391, -4.2676, -4.1670, -3.6186, -4.3206, -3.9839]])
```

```
x - 0.01 * x.grad
```

```
     tensor([[1.0411, 1.0398, 1.0380, 1.0403, 1.0388, 1.0362],
             [1.0422, 1.0377, 1.0439, 1.0391, 1.0422, 1.0432],
             [1.0400, 1.0386, 1.0417, 1.0395, 1.0377, 1.0397],
             [1.0407, 1.0416, 1.0417, 1.0370, 1.0392, 1.0423],
             [1.0424, 1.0427, 1.0417, 1.0362, 1.0432, 1.0398]],
            grad_fn=<SubBackward0>)
```

```
import math
from torch.nn.parameter import Parameter

class My_Linear(nn.Module):

    def __init__(self, in_F, out_F):
        super(My_Linear, self).__init__()
        self.weight = Parameter(torch.Tensor(out_F, in_F))
        self.bias   = Parameter(torch.Tensor(out_F))

        self.reset_parameters()

    def reset_parameters(self):
        for p in self.parameters():
            stdv =  1.0 / math.sqrt(p.shape[0])
            p.data.uniform_(-stdv, stdv)

    def forward(self, x):
        return  x ** self.weight.t() - 2 * x**2 + 5 + self.bias


linear = My_Linear(6, 5)


print ('w: ', linear.weight)
print ('b: ', linear.bias)
```

```
     w:  Parameter containing:
     tensor([[ 0.3952,  0.1552],
             [-0.3745, -0.1427]], requires_grad=True)
     b:  Parameter containing:
     tensor([0.4885, 0.4333], requires_grad=True)
```

```
criterion = nn.MSELoss()


optimizer = torch.optim.SGD(linear.parameters(), lr=0.01)
```

```
x = Variable(torch.randn(5), requires_grad = True)
y = Variable(torch.randn(5), requires_grad = False)

optimizer.zero_grad()
pred = linear(x)
loss = criterion(pred, y)
print('loss: ', loss.item())
```

```
loss:  nan
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/loss.py:520: UserWarning: Usi
  return F.mse_loss(input, target, reduction=self.reduction)
```

```
print ('dL/dw: ', linear.weight.grad)
print ('dL/db: ', linear.bias.grad)
```

```
dL/dw:  None
dL/db:  None
```

```
loss.backward()
```

```
print ('dL/dw: ', linear.weight.grad)
print ('dL/db: ', linear.bias.grad)
```

```
dL/dw:  tensor([[    nan,     nan,     nan,     nan,     nan,     nan],
        [    nan,     nan,     nan,     nan,     nan,     nan],
        [-2.6426, -2.5144, -0.4408, -0.5291, -1.7299, -4.3311],
        [-0.0275, -0.0255, -0.0269, -0.0260, -0.0276, -0.0271],
        [    nan,     nan,     nan,     nan,     nan,     nan]])
dL/db:  tensor([   nan,    nan, 2.0554, 1.8208,    nan])
```

```
print ('w: ', linear.weight)
print ('b: ', linear.bias)
```

```
w:  Parameter containing:
tensor([[ 0.4273,  0.3257,  0.4120,  0.2827, -0.3702,  0.2986],
        [-0.1542, -0.0264,  0.2254, -0.1505, -0.3831, -0.1738],
        [-0.2550, -0.2436,  0.2006,  0.1499, -0.1559, -0.3643],
        [-0.4126,  0.3172, -0.1963,  0.1339, -0.4461, -0.2587],
        [ 0.0511, -0.2285,  0.3618, -0.4322,  0.2126,  0.1731]],
       requires_grad=True)
b:  Parameter containing:
tensor([-0.4432, -0.1088, -0.1250, -0.1516, -0.2215], requires_grad=True)
```

```
optimizer.step()
```

```
print ('w: ', linear.weight)
print ('b: ', linear.bias)
```

```
w:  Parameter containing:
tensor([[ 0.4273,  0.3257,  0.4120,  0.2827, -0.3702,  0.2986],
        [-0.1542, -0.0264,  0.2254, -0.1505, -0.3831, -0.1738],
        [-0.2550, -0.2436,  0.2006,  0.1499, -0.1559, -0.3643],
```

```
          [-0.4126,  0.3172, -0.1963,  0.1339, -0.4461, -0.2587],
          [ 0.0511, -0.2285,  0.3618, -0.4322,  0.2126,  0.1731]],
         requires_grad=True)
b:  Parameter containing:
tensor([-0.4432, -0.1088, -0.1250, -0.1516, -0.2215], requires_grad=True)
```