# **Pytorch教程四-张量的计算**

张量是Torch中最为基本的操作原子，基本上所有的运算和操作都是基于torch的，因此关于tensor的操作也就十分的多，本次文章我们来看一下关于张量的操作相关的内容。

**一、张量的四则运算**

这里先给出在PyTorch中常用的用来张量运算的方法，加减乘除的方法如下：

torch.add(input, other, \*, alpha=1, out=None)

torch.sub(input, other, out=None) torch.mul(input, other, out=None) torch.div(input, other, out=None)

我们来一个个看看这些函数到底是怎么使用的,为此我们先创建几个张量：

t1 = torch.ones(2, 5)  
t2 = torch.arange(10).reshape(2,5)  
print(t1)  
print(t2)

# 输出

tensor([[1., 1., 1., 1., 1.],

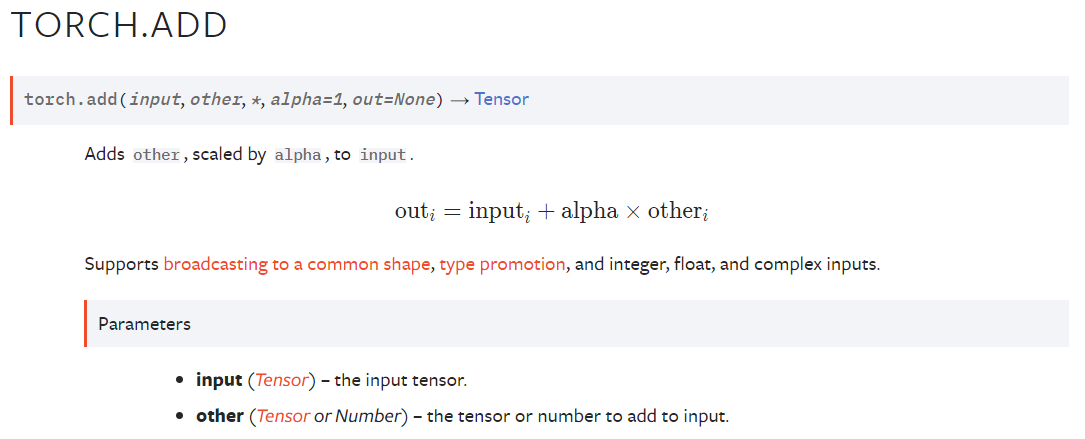
[1., 1., 1., 1., 1.]])

tensor([[0, 1, 2, 3, 4],

[5, 6, 7, 8, 9]])

1. **add()函数**

add()函数表示的是对张量进行**相加**的操作，其原型如下：



这种加法是逐元素相加，输出结果为：out=input + alpha\*other：

add\_1 = torch.add(t1,t2)  
add\_2 = torch.add(t1,t2,alpha=2)  
print(add\_1)  
print(add\_2)

# 输出

tensor([[ 1., 2., 3., 4., 5.],

[ 6., 7., 8., 9., 10.]])

tensor([[ 1., 3., 5., 7., 9.],

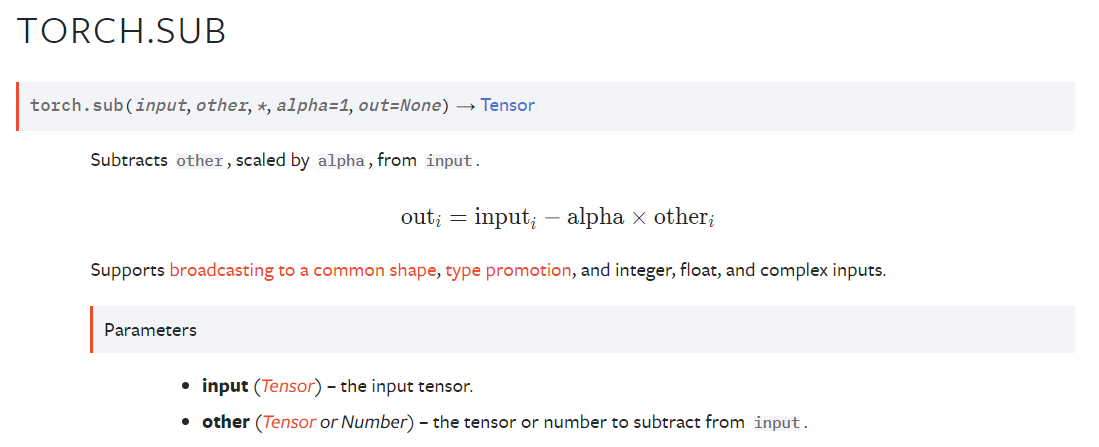
[11., 13., 15., 17., 19.]])

从结果上来看还是很好理解的，上述的操作还有一个等价的操作方式：

print(t1+t2)  
print(t1+2\*t2)

1. **sub()函数**

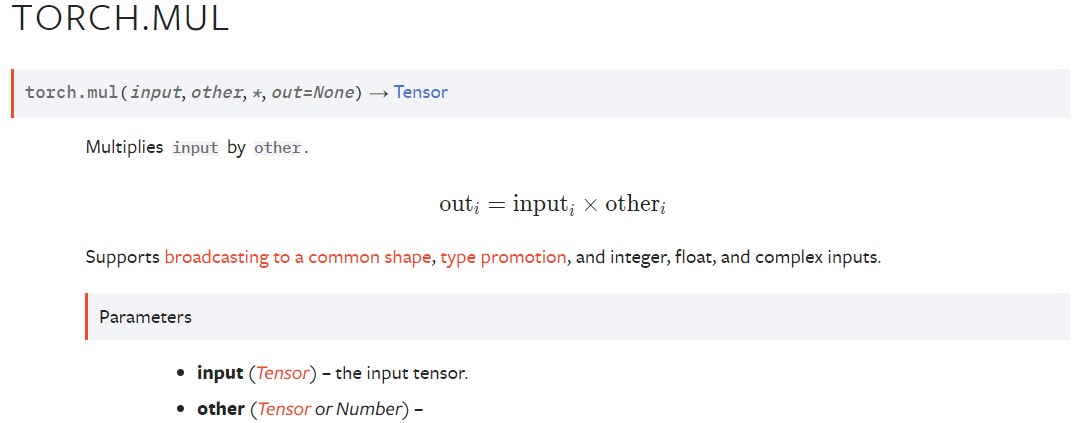
sub()函数表示的是对张量进行**相减**的操作，其原型如下：



加和减基本是类似的，这里我就不进行代码的演示了。

1. **mul()函数**

mul()函数表示的是对张量进行**相乘**的操作，其原型如下：



当然了这个乘法是支持广播操作的，我们来看几个小例子：

mul\_1 = torch.mul(t1+1,t2)  
t3 = torch.randn(4, 1)  
t4 = torch.randn(1, 4)  
mul\_2 = torch.mul(t3,t4)  
print(mul\_1)  
print(t3)  
print(t4)  
print(mul\_2)

# 输出

tensor([[ 0., 2., 4., 6., 8.],

[10., 12., 14., 16., 18.]])

tensor([[ 1.2307],

[ 0.5598],

[-2.1564],

[ 0.4440]])

tensor([[ 1.1551, -0.5289, -1.0828, 1.4717]])

tensor([[ 1.4217, -0.6509, -1.3326, 1.8112],

[ 0.6467, -0.2961, -0.6062, 0.8239],

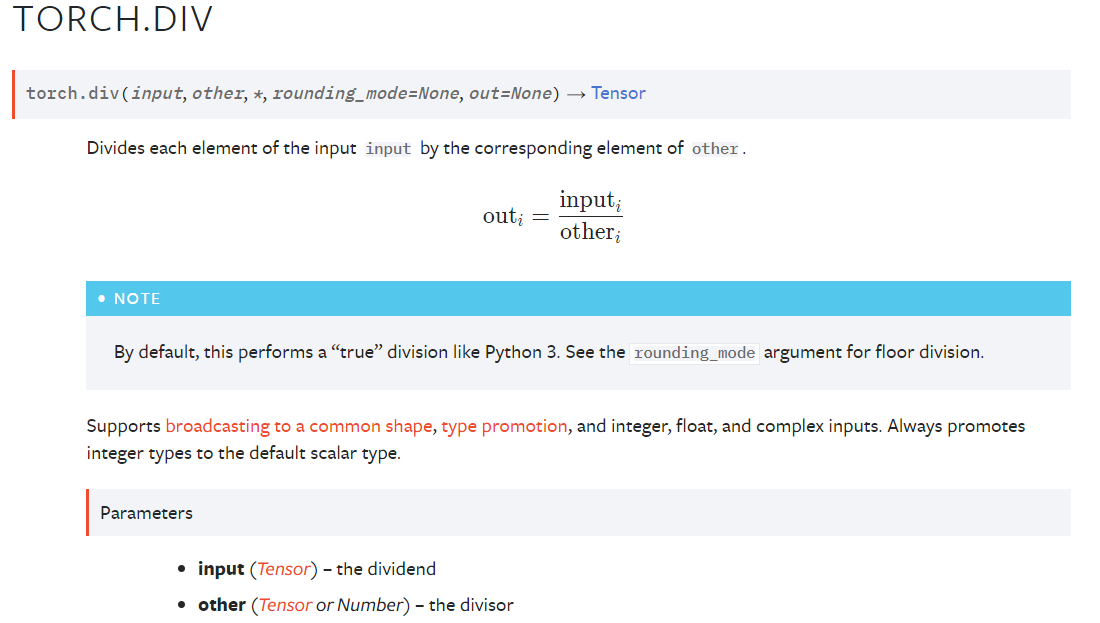
[-2.4910, 1.1405, 2.3350, -3.1735],

[ 0.5129, -0.2348, -0.4808, 0.6534]])

输出的结果还是很好理解的，

1. **div()函数**

div()函数表示的是对张量进行**相除**的操作，其原型如下：



我们知道进行除法的时候，会涉及到一些四舍五入的问题，在div()函数中使用参数rounding\_mode来进行控制，这里我们看一下官网的例子：

a = torch.tensor([[-0.3711, -1.9353, -0.4605, -0.2917],  
 [ 0.1815, -1.0111, 0.9805, -1.5923],  
 [ 0.1062, 1.4581, 0.7759, -1.2344],  
 [-0.1830, -0.0313, 1.1908, -1.4757]])  
b = torch.tensor([ 0.8032, 0.2930, -0.8113, -0.2308])  
div\_1 = torch.div(a,2)  
div\_2 = torch.div(a, b, rounding\_mode='trunc')  
div\_3 = torch.div(a, b, rounding\_mode='floor')  
print(div\_1)  
print(div\_2)  
print(div\_3)

# 输出

tensor([[-0.1856, -0.9676, -0.2303, -0.1459],

[ 0.0908, -0.5056, 0.4902, -0.7962],

[ 0.0531, 0.7290, 0.3880, -0.6172],

[-0.0915, -0.0157, 0.5954, -0.7379]])

tensor([[-0., -6., 0., 1.],

[ 0., -3., -1., 6.],

[ 0., 4., -0., 5.],

[-0., -0., -1., 6.]])

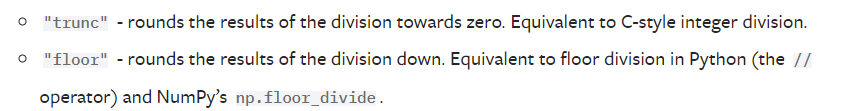
tensor([[-1., -7., 0., 1.],

[ 0., -4., -2., 6.],

[ 0., 4., -1., 5.],

[-1., -1., -2., 6.]])

后面两个输出是存在差异的，主要是因为rounding\_mode参数作用：



即“trunc”是C风格的除法，后者“floor”是np.floor\_divide类的除法。

还有类似的相关函数如：torch.addcdiv()、torch.addcmul()。这里就不在赘述了。

**二、比较运算**

Torch中提供了大量的可应用于张量的比较运算的函数：

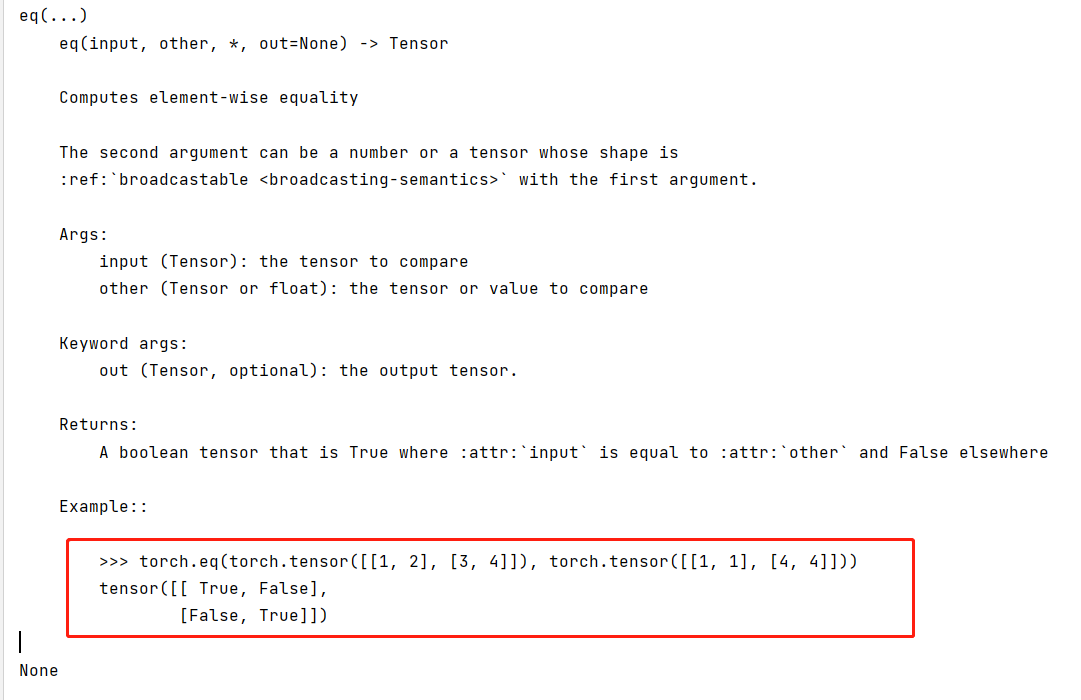
|  |  |
| --- | --- |
| **函数名称** | **函数功能说明** |
| Torch.eq() | 比较张量的元素是否相等 |
| Torch.equal() | 比较张量的元素是否相等且形状是否一样 |
| Torch.allclose() | 比较张量的元素是否接近(阈值控制接近程度) |
| Torch.gt() | 比较张量的元素是否大于 |
| Torch.ge() | 比较张量的元素是否大于等于 |
| Torch.lt() | 比较张量的元素是否小于 |
| Torch.le() | 比较张量的元素是否小于等于 |
| Torch.ne() | 比较张量的元素是否不等于 |

注意：上述比较的结果返回的都是tensor。这里我就不一一举例来说明各个函数的使用方法了，仅看看eq()和equal()函数：

对了，我们在不会使用一个函数的时候，我们可以使用help()的方式来查看它们的详情：

print(help(torch.eq))

输出如下，可以看出某一个函数的用法：



t\_eq\_1 = torch.tensor([1, 2, 3])  
t\_eq\_2 = torch.tensor([1, 1, 3])  
print(torch.eq(t\_eq\_1,t\_eq\_2))  
print(t\_eq\_1.equal(t\_eq\_2))  
print(torch.equal(t\_eq\_1, t\_eq\_2))

# 输出

tensor([ True, False, True])

False

False

除了上述两大类的之外，PyTorch还提供了很多其他的用于张量计算的函数，比如：

1. 三角函数运算(cos,sin...)
2. 对数函数(log,log1p,log2,log10)
3. 指数函数(exp)
4. 幂函数(pow)
5. 其他函数(erf,sign,clamp...)

这些函数也比较常用，这里就不在赘述了。大家自己百度或者查找官网相关文档，或者在使用的时候再查找也是可以的，毕竟那么多的API也是很那记住的啊。（笑哭）

**三、总结**

以上就是本次的文章的全部内容，下次文章我们来看一下Torch中关于自动求导的相关知识，就当做自我复习了。