# **Pytorch教程一-张量的创建**

PyTorch是一个深度学习的框架，近年来受到越来越多的深度学习爱好者的喜欢，其前身Torch是基于Lua进行编写的，后期在Facebook的支持下“幻化”成我们的PyTorch。

1. **PyTorch优点**

关于PyTorch的安装此处就不在赘述了，大家可以参考晚上的教程或者官方的文档。那么PyTorch相较于其他的深度学习框架如Tensorflow的优点有哪些呢？首先我们要知道的是Pytorch的开发目的主要还是帮助深度学习工作者快速实现自己的Idea,总结一下优点：

（1）支持 GPU

（2）动态NN

（3）与Python完美融合

（4）命令式体验

（5）扩展性强

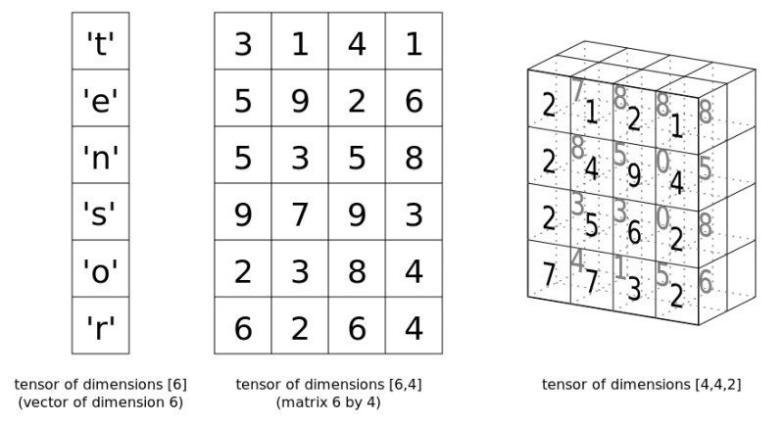
上述只是笔者认为的一些优点，至于其他的一些优点，我们可以在学习的时候亲自的体验。

1. **张量（Tensor）**

笔者学习过程中使用的PyTorch的版本为1.9.0，Python3.8.2。

为什么要提及这个张量呢？因为在PyTorch中，张量是最基础的运算单位，也就是说在PyTorch中我们所有的操作都是在操作张量。与NumPy的ndarray一样，我们可以将张量理解为多维的矩阵。而有一点不一样的是Tensor可以运行在GPU上而ndarray则仅仅能使用CPU(除了GPU的加速之外)。

下面我们举例一些Tensor的维度案例：



上述三个Tensor的维度依次为1维，2维，3维。在我们实际的工程项目中，这三类Tensor的数据常见的场景就是Vector,DataFrame和彩色三通道的图像数据。

那么在PyTorch中我们怎么来创建这常见的三种不同维度的向量呢？接下来我们进行代码演示。

1. **张量的创建**

我们一起来看一下一些常见的常量是怎么创建的：

1. 常数张量（也可以认为是0维的）

import torch

const\_dim\_tensor = torch.Tensor([1])

print(const\_dim\_tensor)

# 输出

tensor([1.])

1. 一维张量

one\_dim\_tensor = torch.Tensor([1.1,2.2,3.3])

print(one\_dim\_tensor)

# 输出

tensor([1.1000, 2.2000, 3.3000])

1. 二维张量

two\_dim\_tensor = torch.Tensor([[1, 2, 3], [4, 5, 6]])

print(two\_dim\_tensor)

# 输出

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

[4., 5., 6.]])

1. 三维张量

three\_dim\_tensor = torch.Tensor(

[[[ 1., 2., 3.],

[ 5., 6., 7.]],

[[ 9., 10., 11.],

[13., 14., 15.]]]

)

print(three\_dim\_tensor)

# 输出

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

[ 5., 6., 7.]],

[[ 9., 10., 11.],

[13., 14., 15.]]])

上述创键tensor的方式是自己传入数据的。我们可以依据PyTorch中提供的一些函数来创建tensor。接下来举例看一下：

|  |  |
| --- | --- |
| 函数 | 作用 |
| ones() | 生成全1的tensor |
| zeros() | 生成全1的tensor |
| rand()/randn() | 生成均匀/标准正态分布tensor |
| eye() | 对角tensor（对角线为1其余为0） |
| arange() | 一定步长对的tensor |
| linspace() | 均分切割的tensor |
| normal()/uniform() | 正态分布/均匀分布的tensor |
| randperm() | 随机排列的tensor |

接下来我们实际看一下这些函数的用法：

import torch  
a = torch.Tensor(3,4)  
print(a)  
print(a.type())

# 输出

tensor([[9.5511e-39, 1.0102e-38, 5.1429e-39, 9.9184e-39],

[9.0000e-39, 1.0561e-38, 1.0653e-38, 4.1327e-39],

[8.9082e-39, 9.8265e-39, 9.4592e-39, 1.0561e-38]])

torch.FloatTensor

再次运行的时候，发现是如下的结果，所以在不指定数据的时候，生成的tensor是全0的，且此时tensor的类型是float类型的。

tensor([[0., 0., 0., 0.],

[0., 0., 0., 0.],

[0., 0., 0., 0.]])

torch.FloatTensor

1. torch.ones()

b1 = torch.ones(2,3)  
b2 = torch.ones(2,3,dtype=torch.int)  
print(b1)  
print(b1.type())  
print(b2)  
print(b2.type())

# 输出如下：

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

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

torch.FloatTensor

tensor([[1, 1, 1],

[1, 1, 1]], dtype=torch.int32)

torch.IntTensor

读者可以看到，在进行创建tensor的时候，我们可以使用dtype来指定tensor数据的类型。

1. torch.zeros()

c = torch.zeros(3,4)  
print(c)

# 输出

tensor([[0., 0., 0., 0.],

[0., 0., 0., 0.],

[0., 0., 0., 0.]])

可知生成的数据全0，一般初始化一个n维数据的时候比较常用。

1. torch.rand()

torch的rand方法和numpy的rand的方法是用法基本上是一致的，生成的随机数的tensor是存在于0和1之间的均匀分布，如下所示：

d = torch.rand(3,4)  
print(d)

# 输出

tensor([[0.9354, 0.6988, 0.4717, 0.5576],

[0.3436, 0.9680, 0.1695, 0.1074],

[0.5679, 0.0161, 0.7270, 0.5117]])

1. torch.randn()

torch的randn方法和numpy的randn的方法是用法基本上是一致的，生成的随机数的tensor是均值为0，方差为1的正态分布，如下所示：

e = torch.randn(3,4)  
print(e)

# 输出

tensor([[ 0.7104, 0.4039, -1.4383, -0.3224],

[-0.2569, 1.3015, 0.1418, -0.7603],

[-0.2319, -0.9325, 0.7468, 1.5297]])

1. torch.eyes()

f = torch.eye(3,4)  
print(f)

# 输出

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

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

[0., 0., 1., 0.]])

对于eye()来说，我们也可以只传入一个参数，torch.eye(3),输出为：

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

[0., 1., 0.],

[0., 0., 1.]])

1. torch.arange()

g = torch.arange(1,10,2)  
print(g)

# 输出

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

第三个参数2表示步长。

1. torch.linspace()

h = torch.linspace(1,10,5)  
print(h)

# 输出

tensor([ 1.0000, 3.2500, 5.5000, 7.7500, 10.0000])

第三个参数5表示最后生成的元素的个数。

1. torch.normal()

torch.normal()产生一个服从离散正态分布的张量随机数，可以指定均值和标准差。其中，标准差std是一个张量包含每个输出元素相关的正态分布标准差。

i = torch.normal(mean=torch.arange(1., 11.), std=torch.arange(1, 0, -0.1))  
print(i)

# 输出

tensor([-0.4251, 2.9629, 2.8784, 3.5575, 3.6615, 6.1325, 6.4492, 8.0842,

9.2036, 9.7753])

1. torch.normal()

j\_1 =torch.Tensor(4).uniform\_(-1,1)  
print(j\_1)

# 输出

tensor([ 0.0123, -0.3709, -0.8251, -0.8292])

j\_2 =torch.Tensor(2,3).uniform\_(-1,1)  
print(j\_2)

# 输出

tensor([[ 0.7568, -0.0081, 0.0437],

[ 0.2330, -0.8853, -0.9160]])

1. torch.randperm()

h = torch.randperm(5)  
print(h)

# 输出

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

再次运行一次，输出：

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

可以看出是随机性产生了元素为整数的tensor

另外在创建tensor的时候我们可借助numpy,即PyTorch提供了torch.from\_numpy()来创建tensor:

import numpy as np  
array = np.array([1, 2, 3, 4, 5])  
tensor\_array = torch.from\_numpy(array)  
print(tensor\_array)

# 输出

tensor([1, 2, 3, 4, 5], dtype=torch.int32)

即我们从numpy的数据中可以创建tensor。

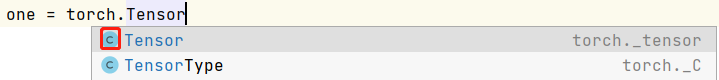
1. **Tensor和tensor**

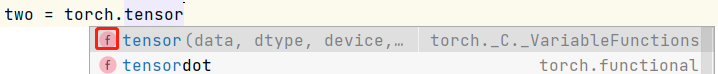
细心的小伙伴可能已经发现，使用Tensor和tensor都可以创建我们需要的张量，那有什么区别呢？

1. torch.Tensor()在PyTorch中是一个类，而tensor是一个函数它的函数定义为：

torch.tensor(data, dtype=None, device=None, requires\_grad=False)

在PyCharm中可以这样查看：





1. 我们创建一个tensor，然后打印他们的类型：

import torch  
one=torch.Tensor([1,2])  
two=torch.tensor([1,2])

print(one)  
print(two)  
print(one.type())  
print(two.type())

# 输出

tensor([1., 2.])

tensor([1, 2])

torch.FloatTensor

torch.LongTensor

可以发现它们的类型是不一样的，主要是在精度和张量的类型上有着区别。

1. **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PyTorch的张量类型相关的知识。