TENSOR

一、概要

张量与NumPy的ndarrays相似，不同之处在于张量可以在GPU或其他硬件加速器上运行。事实上，张量和NumPy数组经常可以共享相同的底层内存，从而消除了复制数据的需要。

二、内容

一、初始化

从python中初始化

```
data = [[1, 2],[3, 4]]
x_data = torch.tensor(data)
print(x_data)
```

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

从numpy中初始化

```
data = [[1, 2],[3, 4]]
print(type(data[0][0]))
x_data = torch.tensor(data)
print(x_data.dtype)
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
print(x_np)
```

```
<class 'int'>
torch.int64
tensor([[1, 2],
        [3, 4]], dtype=torch.int32)
```

从numpy中初始化时，tensor默认是32位的int。

注意：CPU上的张量和NumPy数组可以共享它们的底层内存位置，改变其中一个会改变另一个。

tensor=>np

```
t = torch.ones(5)
print(f"t: {t}")
n = t.numpy()
print(f"n: {n}")

t.add_(1)
print(f"t: {t}")
print(f"n: {n}")
```

```
t: tensor([1., 1., 1., 1., 1.])
n: [1. 1. 1. 1. 1.]
t: tensor([2., 2., 2., 2., 2.])
n: [2. 2. 2. 2. 2.]
```

np=>tensor

```
n = np.ones(5)
t = torch.from_numpy(n)
print(f"t: {t}")
print(f"n: {n}")
np.add(n, 1, out=n)
print(f"t: {t}")
print(f"n: {n}")
```

```
t: tensor([1., 1., 1., 1., 1.], dtype=torch.float64)
n: [1. 1. 1. 1. 1.]
t: tensor([2., 2., 2., 2., 2.], dtype=torch.float64)
n: [2. 2. 2. 2. 2.]
```

复制另一个tensor的型状

```
data = [[1, 2],[3, 4]]
x_data = torch.tensor(data)
x_ones = torch.ones_like(x_data)
print(f"\n {x_ones} \n")
x_rand = torch.rand_like(x_data, dtype=torch.float)
print(f"\n {x_rand} \n")
```

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


tensor([[0.6708, 0.9273],
        [0.3637, 0.1464]])
```

> 新的张量保留了参数张量的属性：形状，数据类型（shape, datatype），除非显式地覆盖。

从函数中初始化

```
rand_tensor = torch.rand(3,2)
ones_tensor = torch.ones(3,2)
print(rand_tensor)
print(ones_tensor)
```

```
tensor([[0.8623, 0.9927],
        [0.7194, 0.5140],
        [0.5389, 0.0119]])
tensor([[1., 1.],
        [1., 1.],
        [1., 1.]])
```

二、tensor的属性

tensor属性描述了它们的形状、数据类型和存储它们的设备。

```python
tensor = torch.rand(3, 4)
print(f"Shape: {tensor.shape}")
print(f"Datatype: {tensor.dtype}")
print(f"Device: {tensor.device}")
print(f"转置: {tensor.T}")
```

```
Shape: torch.Size([3, 4])
Datatype: torch.float32
Device: cpu
转置: tensor([[0.9781, 0.8337, 0.1247],
        [0.4495, 0.1493, 0.8012],
        [0.8006, 0.6573, 0.5372],
        [0.4269, 0.0887, 0.9392]])
```

三、Tensor操作（函数）

有100多种张量运算，包括转置、索引、切片、数学运算、线性代数、随机采样等。这些运算都可以在GPU上运行（通常比在CPU上更快）。

```python
if torch.cuda.is_available():
   tensor = tensor.to('cuda')
   print(f"Device tensor is stored on: {tensor.device}")
```

按元素乘法和矩阵乘法

```python
tensor = torch.ones(3,3)
tensor[:,1] = 0
print(tensor)

# 按元素乘法
print(f"tensor.mul(tensor) \n {tensor.mul(tensor)}")
print(f"tensor * tensor \n {tensor * tensor}")
# 矩阵乘法
print(f"tensor.matmul(tensor.T) \n {tensor.matmul(tensor.T)}")
print(f"tensor @ tensor.T \n {tensor @ tensor.T}")
```

```
tensor([[1., 0., 1.],
        [1., 0., 1.],
        [1., 0., 1.]])
tensor.mul(tensor)
 tensor([[1., 0., 1.],
        [1., 0., 1.],
        [1., 0., 1.]])
tensor * tensor
 tensor([[1., 0., 1.],
        [1., 0., 1.],
        [1., 0., 1.]])
tensor.matmul(tensor.T)
 tensor([[2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.]])
tensor @ tensor.T
 tensor([[2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.]])
```

原地操作（In-place operations）

```
tensor = torch.ones(3,3)
tensor[:,1] = 0
print(tensor)
tensor.add_(5)
print(tensor)
```

```
tensor([[1., 0., 1.],
        [1., 0., 1.],
        [1., 0., 1.]])
tensor([[6., 5., 6.],
        [6., 5., 6.],
        [6., 5., 6.]])
```

> 原地操作会改变内容。