PyTorch张量与操作

1.PyTorch张量

张量类似于Numpy的ndarray，另外tersors也可用于GPU以加速计算。

```
from __future__ inport print_function
import torch
```

torch.empty():

构造一个未初始化的矩阵：

```
x = torch.empty(5, 3)
print(x)
```

输出：

```
tensor([[0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.]])
```

torch.rand():

构造一个随机初始化的矩阵：

```
x = torch.rand(5, 3)
print(x)
print(type(x))
```

输出：

```
tensor([[0.4810, 0.0878, 0.4352],
 [0.7106, 0.1466, 0.2440],
 [0.9014, 0.1964, 0.2801],
 [0.1658, 0.8228, 0.1042],
 [0.0480, 0.1602, 0.2322]])
<class 'torch.Tensor'>
```

torch.zeros():

构造一个以0填充且数据类型dtype为long的矩阵：

```
x = torch.zeros(5, 3, dtype=torch.long)
print(x)
```

输出：

```
tensor([[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0],
 [0, 0, 0],
```

```
  [0, 0, 0]])
```

torch.tensor():

直接从数据构造tensor：

```
x = torch.tensor([5.5, 3])
print(x)
```

输出：

```
tensor([5.5000, 3.0000])
```

可以根据已有的tensor变量创建新的tensor变量。此处新tensor变量保留原tensor的尺寸大小、数值类型等属性，除非是重新定义这些属性。

tensor.new_ones():

new_*()方法需要输入尺寸大小

```
tensor1 = torch.tensor([5.5, 3])
# 显示定义新的尺寸为5*3，数值类型为torch.double
tensor2 = tensor1.newones(5, 3, dtype=torch.double)
print(tensor2)
```

输出：

```
tensor([[1., 1., 1.],
 [1., 1., 1.],
 [1., 1., 1.],
 [1., 1., 1.],
 [1., 1., 1.]], dtype=torch.float64)
```

torch.randn_like(old_tensor):

保留相同尺寸大小生成随机的tensor

```
tensor3 = torch.randn_like(tensor, dtype=torch.float) # 修改参数类型
print(tensor3)
```

输出：

```
tensor([[-0.4491, -0.2634, -0.0040],
 [-0.1624, 0.4475, -0.8407],
 [-0.6539, -1.2772, 0.6060],
 [ 0.2304, 0.0879, -0.3876],
 [ 1.2900, -0.7475, -1.8212]])
```

输出结果是根据上个方法声明的tensor2变量来声明的新变量，尺寸不变。

tensor.size():

获取tensor的尺寸大小：

```
print(tensor3.size())
```

输出：

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

注：torch.Size实际上是元组（tuple）类型，所以支持所有的元组操作。

2.PyTorch操作

加法操作的实现：

（1）+运算符

（2）torch.add(tensor1, tensor2, [out=tensor3])

（3）tensor1.add_(tensor) # 直接修改tensor变量

```
import torch
x = torch.rand(5, 3)
y = torch.rand(5, 3)
print('加法的第一种实现：+')
print(x+y)
print('加法的第二种实现：torch.add()')
# print(torch.add(x, y))
result = torch.empty(5, 3) # 创建一个用于填充结果的张量
torch.add(x, y, out=result) # 添加输出张量作为参数
 print(result)
 print('加法的第三种实现：tensor1.add_(tensor2)')
 print(x.add_(y)) # 直接改变x的值
 print(x)
```

输出：

```
加法的第一种实现：+
tensor([[1.2263, 1.5124, 0.8025],
 [1.1695, 1.3850, 0.5301],
 [1.0961, 0.6277, 0.0963],
 [0.3961, 0.3045, 0.7070],
 [1.3090, 0.3670, 1.0136]])
加法的第二种实现：torch.add()
tensor([[1.2263, 1.5124, 0.8025],
 [1.1695, 1.3850, 0.5301],
  [1.0961, 0.6277, 0.0963],
  [0.3961, 0.3045, 0.7070],
  [1.3090, 0.3670, 1.0136]])
 加法的第三种实现：tensor1.add_(tensor2)
 tensor([[1.2263, 1.5124, 0.8025],
  [1.1695, 1.3850, 0.5301],
  [1.0961, 0.6277, 0.0963],
```

```
 [0.3961, 0.3045, 0.7070],
 [1.3090, 0.3670, 1.0136]])
tensor([[1.2263, 1.5124, 0.8025],
 [1.1695, 1.3850, 0.5301],
 [1.0961, 0.6277, 0.0963],
 [0.3961, 0.3045, 0.7070],
 [1.3090, 0.3670, 1.0136]])
```

注：可以改变tensor的变量操作都带有一个后缀“_”，例如x.copy_(y), x.t_()都可以改变x变量。

对于tensor的访问，和Numpy对数组类似，可以**使用索引来访问某一维度的数据**，例如：

```
# 访问tensor3第一列数据
tensor3 = tensor([[ 0.1000, 0.1325, 0.0461],
 [ 0.4731, 0.4523, -0.7517],
 [ 0.2995, -0.9576, 1.4906],
 [ 1.0461, 0.7557, -0.0187],
 [ 2.2446, -0.3473, -1.0873]])
print(tensor3[:, 0])
```

输出：

```
tensor([0.1000, 0.4731, 0.2995, 1.0461, 2.2446])
```

torch.view():

修改tensor的尺寸

```
x = torch.randn(4, 4)
y = x.view(16)
# -1 表示除给定维度外的其余维度的乘积
z = x.view(-1, 8) # 此处的8意味着每个维度8个数据
print('x={}\ny={}\nz={}'.format(str(x), str(y), str(z)))
print(x.size(), y.size(), z.size())
```

输出：

```
x=tensor([[-2.3745e-02, 1.8577e-01, -3.0376e-01, -1.3960e-01],
 [ 2.1698e-01, 1.0112e+00, 1.8753e-01, -2.6876e-01],
 [-6.4817e-01, -1.0856e+00, -6.1193e-01, 1.5208e+00],
 [ 1.1180e+00, -5.8861e-01, 3.6519e-05, 9.9426e-01]])
y=tensor([-2.3745e-02, 1.8577e-01, -3.0376e-01, -1.3960e-01, 2.1698e-01,
 1.0112e+00, 1.8753e-01, -2.6876e-01, -6.4817e-01, -1.0856e+00,
 -6.1193e-01, 1.5208e+00, 1.1180e+00, -5.8861e-01, 3.6519e-05,
 9.9426e-01])
```

```
z=tensor([[-2.3745e-02, 1.8577e-01, -3.0376e-01, -1.3960e-01, 2.1698e-01,
  1.0112e+00, 1.8753e-01, -2.6876e-01],
  [-6.4817e-01, -1.0856e+00, -6.1193e-01, 1.5208e+00, 1.1180e+00,
  -5.8861e-01, 3.6519e-05, 9.9426e-01]])
 torch.Size([4, 4]) torch.Size([16]) torch.Size([2, 8])
```

tensor.item()

tensor中仅有一个元素，可以利用此方法获取类似python中整形类型的数值

```
x = torch.randn(1) # 返回一个填充了正态分布中随机数的张量，均值=0 方差=1
print(x)
print(x.item())
```

输出：

```
tensor([0.4549])
0.4549027979373932
```

注：更多运算操作：

https://pytorch.org/docs/stable/torch.html

3.Tensor和Numpy数组的转换

Tensor和Numpy数组可以相互转换，并且两者转换后共享在CPU下的内存空间，即改变其中一个数值，另一个变量随之改变。

tensor转换为Numpy数组

tensor.numpy()

实现tensor转换为Numpy数组

```
a = torch.ones(5)
print(a)
b = a.numpy()
print(b)
```

输出：

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

两个变量共享同一个内存空间：

修改tensor变量a，则对应的numpy数组b的变化：

```
a.add_(1)
print(a)
print(b)
```

输出：

```
tensor([2., 2., 2., 2., 2.])
```

```
[2. 2. 2. 2. 2.]
```

numpy数组转为tensor

torch.from_numpy(numpy_array)

```
import numpy as np
a = np.ones(5)
b = torch.from_numpy(a)
np.add(a, 1, out=a)
print(a)
print(b)
```

输出：

```
[2. 2. 2. 2. 2.]
tensor([2., 2., 2., 2., 2.], dtype=torch.float64)
```

在CPU上，除了CharTensor外的所有Tensor类型变量，都支持和Numpy数组的相互转换操作

4.CUDA张量

Tensors可以通过.to(device)方法转换到不同设备上（即CPU和GPU）。

```
# 当CUDA可用时，可以运行下方代码，采用torch.device()方法来
# 改变tensors是否在GPU上进行计算操作
if torch.cuda.is_available():
device = torch.device("cuda") # 定义一个CUDA设备对象
y = torch.ones_like(x, device=device) # 显示创建在GPU上的一个tensor
x = x.to(device) # 也可以采用.to("cuda")
z = x + y
print(z)
print(z.to("cpu", torch.double)) # .to()方法也可以改变数值类型
```

输出结果，第一个为在GPU上的结果，打印变量会有device='cuda:0'.第二个变量是在cpu上的变量

```
tensor([1.4549], device='cuda:0')
tensor([1.4549], dtype=torch.float64)
```