Pytorch

一、张量

张量在数学概念中是多维数组，但在Pytorch中不仅仅表示为多维数组，它还是自动求导的关键。说到张量与自动求导，必须先知道Variable这一数据类型，Variable是Pytorch 0.4.0版本之前的重要数据类型，但之后Variable已经并入到了Tensor，但了解Variable是有必要的。

1.1 Tensor与Variable

1、在Pytorch0.4.0之前，Variable 是torch.autograd中的数据类型，主要用于封装Tensor,进行自动求导，它主要有五个属性（），**data, grad, grad_fn, requires_grad, is_leaf**.

data: 被包装的Tensor

grad：data的梯度

grad_fn: 记录创建tensor的函数function，是自动求导的关键

requires_grad: 指示是否需要梯度

is_leaf: 指示是否需要叶子节点（张量），叶子节点会在计算图中要用到

2、在pytorch0.4.0之后，Variable并入到了Tensor，增加了三个与数据相关的属性，**dtype, shape, device**

dtype: 张量的数据类型，如torch.FloatTensor, torch.cuda.FloatTensor

shape：张量的形状, 如（64，3，224，224）

device：张量所在的设备，GPU/CPU，是加速的关键

1.2 创建张量

1.2.1 直接创建

（1）通过torch.tensor()创建：

功能：它是将data直接创建成了tensor

```
torch.tensor(data
             ,dtype = None
             ,device = None
             ,requires_grad = False
             ,pin_memory = False)
```

- data: 数据，可以是list，numpy
- dtype：数据类型，默认与data保持一致
- device：所在设备，cuda/cpu
- requires_grad：是否需要计算梯度
- pin_memory：是否存于锁页内存，通常设置为False

```
import torch
import numpy as np

# ====================================  example 1  ========================
# 通过torch.tensor创建张量
flag = True
# flag = False
if flag:
    arr = np.ones((3, 3))
    print('ndarray的数据类型：', arr.dtype)
    print(arr)

    t = torch.tensor(arr, device='cuda')  # 张量放在GPU上,可能会需要一点耗时，因为从cpu转到GPU上会需要一点时间
    # t = torch.tensor(arr)

    print(t)
```

```
ndarray的数据类型： float64
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
tensor([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]], device='cuda:0', dtype=torch.float64) # 0是GPU的编号

Process finished with exit code 0
```

（2）torch.from_numpy(ndarray):

功能：从numpy创建tensor

注意：**从torch.from_numpy创建的tensor与原ndarray共享内存，当改变其中的一个数据，另一个也将会被改动**

```
# ====================================  example 2  ========================
# 通过torch.from_numpy创建张量
flag = True
# flag = False
if flag:
    arr = np.array([[1, 2, 3], [4, 5, 6]])
    t = torch.from_numpy(arr)
    # print('numpy array:', arr)
```

```
    # print('tensor:', t)

    # 修改array的数据
    # print('\n修改arr')
    # arr[0, 0] = 0
    # print('numpy array:', arr)
    # print('tensor:', t)  # 两者都发生变化

    # 修改tensor
    print('\n修改arr')
    t[0, 0] = -1
    print('numpy array:', arr)
    print('tensor:', t)  # 两者都发生变化
```

####1.2.2、依据数值创建

(1) torch. zeros()

功能：依据size创建全0张量

```
torch.zeros(*size
            ,out = None
            ,dtype = None
            ,layout = torch.strided
            ,device = None
            ,requires_grad = False)
```

- size:张量的形状，如（3, 3）、（3,224,224）
- out:输出的张量
- layout：内存中的布局形式，有strided,sparse_coo等
- device：所在设备，gpu/cpu
- requires_grad：是否需要梯度

```
# ======================================  example 3  ========================
# 通过torch.zeros创建张量
flag = True
# flag = False
if flag:
    out_t = torch.tensor([1])

    t = torch.zeros((3, 3), out=out_t)

    print(t, '\n', out_t)
    print(id(t), id(out_t), id(t) == id(out_t))
```

(2) 通过torch.zeros_like()

功能：根据input形状创建全0张量

```
torch.zeros_like(input
                 ,dtype = None
                 ,layout = None
                 ,device = None
                 ,requires_grad = False)
```

- input：创建与input相同形状的全0张量
- dtype：数据类型
- layout：内存中的布局形式

(3) torch.ones() --类比zeros

(4) torch.ones_like()

(5) 通过torch.full() --自定义数值

```
torch.full(size
           ,fill_value  # 张量的形状
           ,out = None  # 张量的值
           ,dtype = None
           ,layout = torch.strided
           ,device = None
           ,requires_grad = False)
```

```
# ====================================  example 4  ========================
# 通过torch.zeros创建张量
flag = True
# flag = False
if flag:
    t = torch.full((3, 3), 10)  # 自定义全10 张量
    print(t)
```

(6) torch.full_like()

(7) 通过torch.arange() --等差数列的方法，[start，end)左闭右开

功能：创建等差的一维张量

```
torch.arange(start=0  # 起始值
             ,end  # 结束值
             ,step=1  # 数列公差，默认为1
             ,out=None
             ,dtype=None
             ,layout=torch.strided
             ,device=None
             ,requires_grad=False)
```

```
# 通过torch.zeros创建张量
flag = True
# flag = False
if flag:
    t = torch.arange(2, 10, 2)
    print(t)
```

（8）通过torch.linspace()创造均分数列，[start, end]闭区间

```
torch.linspace(start  # 起始值
               ,end  # 结束值
               ,step=100  # 数列长度
               ,out=None
               ,dtype=None
               ,layout=torch.strided
               ,device=None
               ,requires_grad=False)
```

```
# 通过torch.linspace创建张量
flag = True
# flag = False
if flag:
    t = torch.linspace(2, 10, 5)
    print(t)
```

（9）torch.logspace()

功能：创建对数均分的一维张量

注意：长度为steps，底为base

```
torch.logspace(start  # 起始值
               ,end  # 结束值
               ,step=100  # 数列长度
               ,base=10.0  # 底
               ,out=None
               ,dtype=None
               ,layout=torch.strided
               ,device=None
               ,requires_grad=False)
```

（10）torch.eye()

功能：创建单位对角矩阵

1.2.3、依据概率创建

(1) torch.normal()

功能：生成正态分布（高斯分布）的张量

```
torch.normal(mean  # 均值
             ,std  # 标准差
             ,out=None)
```

注意：此方式有四种模式

mean为标量，std为标量

mean为标量，std为张量

mean为张量，std为标量

mean为张量，std为张量

```
# 通过torch.normal创建张量
flag = True
# flag = False
if flag:
    # mean:张量，std：张量
    mean = torch.arange(1, 5, dtype=torch.float)
    std = torch.arange(1, 5, dtype=torch.float)
    t_normal = torch.normal(mean, std)
    print('mean:{}\nstd:{}'.format(mean, std))
    print(t_normal)
```

```
mean:tensor([1., 2., 3., 4.])
std:tensor([1., 2., 3., 4.])
tensor([2.0285, 2.0979, 0.5410, 7.9715])
```

其中2.0285是由均值为1，标准差为1的正态分布生成的张量；7.9715是由均值为4，标准差为4的正态分布生成的张量

```
 # mean:标量， std:标量
    t_normal = torch.normal(0, 1, size=(5,))  # 这里需要额外设置size
    print(t_normal)
```

```
# mean：张量， std:标量
    mean = torch.arange(1, 5, dtype=torch.float)
    std = 1
    t_normal = torch.normal(mean, std)
    print('mean:{}\nstd:{}'.format(mean, std))
    print(t_normal)
```

```
mean:tensor([1., 2., 3., 4.])
std:1
tensor([0.3336, 2.8148, 2.6217, 2.9895])
```

```
# mean:标量， std:张量
    mean = 0
    std = torch.arange(1, 5, dtype=torch.float)
    t_normal = torch.normal(mean, std)
    print('mean:{}\nstd:{}'.format(mean, std))
    print(t_normal)
```

(2) torch.randn()

(3) torch.randn_like（）

功能： 生成标准正态分布

- size：张量的形状

```
torch.randn(*size
            ,out=None
            ,dtype=None
            ,layout=torch.strided
            ,device=None
            ,requires_grad=False)
```

(4) torch.rand()

(5) torch.rand_like()

功能：在区间[0, 1)上，生成均匀分布

(6) torch.randint()

(7) torch.randint_like()

功能：区间[low, high)生成整数均匀分布

size: 张量的形状

(8) torch.randperm()

功能：生成从0- n-1的随机排列

(9) torch.bernoulli()

功能：根据input为概率，生成伯努利分布

1.2.4 简答

1、张量与矩阵、向量、标量的关系是怎么样的?

- 在数学上，张量是一个多维数组，它是标量、向量、矩阵的高维拓展。标量就是0维的张量，向量就是一维的张量，矩阵是二维的张量
- tensor是pytorch最基本的操作对象，表示的是一个多维的矩阵。
  tensor与numpy相对应，可与numpy的ndarray相互转换。但pytorch可以再GPU上运行，ndarray只能在CPU上运行。

2、Variable"赋予"张量什么功能?

Variable赋予了tensor封装与自动求导功能。

3、实现torch.normal( )创建张量的四种模式