```
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
```

什么是PyTorch?

PyTorch是基于Python的科学计算库，包括两个特点：

- 可以认为PyTorch是可以很好利用GPUs的NumPy
- 一个提供最大灵活性和速度的深度学习研究平台

让我们开始吧！

Torch张量与NumPy的ndarrays很像，方便使用GPU进行加速运算。

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

创建一个未初始化的5x3矩阵

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

```
tensor([[-4.2757e+15,  4.5706e-41, -4.2757e+15],
        [ 4.5706e-41,  1.3283e-08,  1.7186e-04],
        [ 1.0801e-05,  1.6822e-04,  5.2480e-08],
        [ 8.2647e+20,  8.4948e+20,  1.3150e+22],
        [ 3.3497e-09,  4.2657e-08,  5.3735e-05]])
```

创建一个随机初始化矩阵

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

```
tensor([[0.5440, 0.5674, 0.5301],
        [0.5652, 0.4678, 0.6458],
        [0.0494, 0.4971, 0.5074],
        [0.7826, 0.4193, 0.5726],
        [0.5966, 0.4346, 0.1485]])
```

创建一个long数据类型零矩阵

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

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

直接从数据创建张量

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

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

从已有的张量中创建张量。这些方法会使用输入张量的一些属性，例如数据类型dtype，但是不会使用已有张量的数据值

```
x = x.new_ones(5, 3, dtype=torch.double)      # new_*方法按照sizes尺寸创建tensor
print(x)

x = torch.randn_like(x, dtype=torch.float)    # 更改数据类型，重载数据类型dtype!
print(x)                                      # 两种创建方式的结果拥有相同的尺寸
```

```
tensor([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]], dtype=torch.float64)
tensor([[-0.4980,  1.1899, -0.4753],
        [-1.6577, -0.1259, -1.9142],
        [-1.0422, -0.6243,  0.3144],
        [-0.4593, -0.5345,  0.7847],
        [ 1.4574,  1.7541,  1.5835]])
```

得到数据的size

```
print(x.size())
```

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

记录

``torch.Size``实际上是一个元组, 所以该变量都支持所有的元组操作。

操作 有多个torch.tensor的语法操作。接下来的例子中，我们看一下加法操作

加法: 语法1

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

```
tensor([[ 0.0255,  1.4141, -0.0656],
        [-1.0872,  0.3904, -1.0431],
        [-0.4227, -0.1648,  0.4838],
        [-0.1308,  0.2492,  1.0986],
        [ 1.5024,  2.7532,  1.8516]])
```

加法：语法2

```
print(torch.add(x, y))
```

```
tensor([[ 0.0255,  1.4141, -0.0656],
        [-1.0872,  0.3904, -1.0431],
        [-0.4227, -0.1648,  0.4838],
        [-0.1308,  0.2492,  1.0986],
        [ 1.5024,  2.7532,  1.8516]])
```

加法：设置输出张量为加法函数变量

```
result = torch.empty(5, 3)
torch.add(x, y, out=result)
print(result)
```

```
tensor([[ 0.0255,  1.4141, -0.0656],
        [-1.0872,  0.3904, -1.0431],
        [-0.4227, -0.1648,  0.4838],
        [-0.1308,  0.2492,  1.0986],
        [ 1.5024,  2.7532,  1.8516]])
```

加法：内置

```
# 将x加到y上，改变y的值
y.add_(x)
print(y)
```

```
tensor([[ 0.0255,  1.4141, -0.0656],
        [-1.0872,  0.3904, -1.0431],
        [-0.4227, -0.1648,  0.4838],
        [-0.1308,  0.2492,  1.0986],
        [ 1.5024,  2.7532,  1.8516]])
```

笔记

对于张量运算而言，在对应的操作之后加上``_``可是以张量为类主体实现任意的操作。 例如: ``x.copy_(y)``, ``x.t_()``, 这些操作会改变 ``x``的值.

你也可以使用标准的类NumPy索引使用所有的修饰符!

```
print(x[:, 1])
```

```
tensor([ 1.1899, -0.1259, -0.6243, -0.5345,  1.7541])
```

张量尺寸变化：如果你想更改张量的尺寸，可以使用`torch.view`:

```
x = torch.randn(4, 4)
y = x.view(16)
z = x.view(-1, 8)  # 这里的-1将会按照其他维度的大小而自动调整
print(x.size(), y.size(), z.size())
```

```
torch.Size([4, 4]) torch.Size([16]) torch.Size([2, 8])
```

如果你有只有一个元素的张量，可以使用`.item()`来获取这个张量的值。

```
x = torch.randn(1)
print(x)
print(x.item())
```

```
tensor([-0.7095])
-0.7094669938087463
```

进一步阅读:

对于100+的张量操作，包括转置、索引、分割、数学操作、线性代数、随机数等等。 可以参考`这里 <https://pytorch.org/docs/torch>`.

与NumPy的桥梁

将Torch张量转换为NumPy阵列是很容易的，反之也是一样的

Torch张量和NumPy阵列共享内存，因此改变一个将会改变另外一个。

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

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

```
b = a.numpy()
print(b)
```

```
[ 1.  1.  1.  1.  1.]
```

来看看numpy阵列是如何改变Torch张量的。

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

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

将NumPy阵列转换为Torch张量

看看np阵列是如何自动改变Torch张量的。

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

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

所有的CPU上张量除了字符张量都支持Torch张量和Numpy阵列的相互转换。

CUDA张量

张量可以使用`.to`方法实现数据传输到任意设备上，当然包括GPU。

```
# CUDA可用的话，if中的代码才会运行
# 我们使用``torch.device``来实现张量移入和移出GPU
if torch.cuda.is_available():
    device = torch.device("cuda")          # 一个CUDA设备对象
    y = torch.ones_like(x, device=device)  # 在GPU上直接创建张量tensor
    x = x.to(device)                       # 或者使用命令``.to("cuda")``将tensor传入GPU
    z = x + y
    print(z)
    print(z.to("cpu", torch.double))       # ``.to``也可以改变数类型!
```

```
tensor([0.2905], device='cuda:0')
tensor([0.2905], dtype=torch.float64)
```