In [1]:

```
import torch
```

In [5]:

```
import numpy as np
```

torch 基本处理单元

In [4]:

```
# 返回的数组大小5x4的矩阵
torch.Tensor(5, 4)
```

Out[4]:

```
1.00000e-03 *
 -6.7953  0.0000 -6.7953  0.0000
  0.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  0.0000
[torch.FloatTensor of size 5x4]
```

In [10]:

```
# 返回的数组大小是5x4的矩阵，初始化是0~1的均匀分布
torch.rand(5, 4)
```

Out[10]:

```
 0.7977  0.1654  0.8409  0.7564
 0.3139  0.2253  0.6559  0.2096
 0.3448  0.7688  0.8343  0.9917
 0.2122  0.1920  0.1087  0.3080
 0.9847  0.0374  0.2029  0.7608
[torch.FloatTensor of size 5x4]
```

In [13]:

```
# 得到矩阵大小
a = torch.rand(5, 4)
a.size()
```

Out[13]:

```
torch.Size([5, 4])
```

In [12]:

```
# numpy 类似的返回5x4大小的矩阵
np.ones((5, 4))
```

Out[12]:

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

In [24]:

```
# numpy 和 torch.Tensor 之间的转换
a = torch.rand(5, 4)
b = a.numpy()
print(b)
```

```
[[  7.89588690e-01   7.43446290e-01   1.61940977e-01   4.80842412e-01]
 [  1.78743862e-02   1.91753581e-01   3.34077984e-01   3.44260454e-01]
 [  3.15251201e-01   1.37171671e-01   3.29199225e-01   1.57159448e-01]
 [  7.46150970e-01   8.45601916e-01   9.45035994e-01   4.01963502e-01]
 [  4.72619027e-01   1.90935910e-01   8.99520470e-04   3.92095864e-0
1]]
```

In [23]:

```
a = np.array([[3, 4], [3, 6]])
b = torch.from_numpy(a)
print(b)
```

```
 3  4
 3  6
[torch.LongTensor of size 2x2]
```

运算和numpy类似

In [14]:

```
x = torch.rand(5, 4)
y = torch.rand(5, 4)
c = 3
```

In [15]:

```
print(c * x)
```

```
 1.8998  2.4494  0.4745  2.7200
 2.7077  2.6774  2.5322  1.7018
 2.1523  1.3062  2.9104  0.8672
 2.3793  1.4136  2.4238  0.4781
 0.3026  0.1418  0.9806  2.1584
[torch.FloatTensor of size 5x4]
```

In [17]:

```
print(x + y)
```

```
 1.4478  1.1207  0.7065  1.4593
 1.5152  1.4361  1.8031  0.5893
 0.9860  0.4821  1.2809  0.5736
 1.6838  1.1084  0.9404  0.7543
 0.1237  0.1471  0.8376  1.2901
[torch.FloatTensor of size 5x4]
```

In [18]:

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

```
 1.4478  1.1207  0.7065  1.4593
 1.5152  1.4361  1.8031  0.5893
 0.9860  0.4821  1.2809  0.5736
 1.6838  1.1084  0.9404  0.7543
 0.1237  0.1471  0.8376  1.2901
[torch.FloatTensor of size 5x4]
```

In [19]:

```
# 可以直接进行操作改变原对象，x+y或者x.add()并不会改变x，但是x.add_()则会对x进行改变
x.add_(y)
```

Out[19]:

```
 1.4478  1.1207  0.7065  1.4593
 1.5152  1.4361  1.8031  0.5893
 0.9860  0.4821  1.2809  0.5736
 1.6838  1.1084  0.9404  0.7543
 0.1237  0.1471  0.8376  1.2901
[torch.FloatTensor of size 5x4]
```

In [20]:

```
print(x)
```

```
 1.4478  1.1207  0.7065  1.4593
 1.5152  1.4361  1.8031  0.5893
 0.9860  0.4821  1.2809  0.5736
 1.6838  1.1084  0.9404  0.7543
 0.1237  0.1471  0.8376  1.2901
[torch.FloatTensor of size 5x4]
```

将 torch.Tensor 放到 GPU 上

In [26]:

```
# 判断一下电脑是否支持GPU
torch.cuda.is_available()
```

Out[26]:

```
True
```

In [28]:

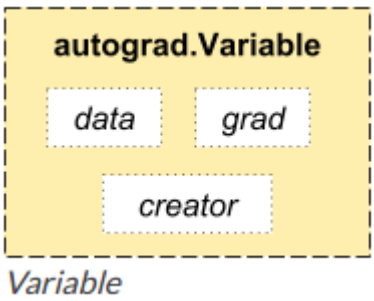
```
a = torch.rand(5, 4)
a = a.cuda()
print(a)
```

```
 0.6592  0.4923  0.6790  0.8395
 0.5175  0.1796  0.4245  0.1928
 0.1701  0.1957  0.3858  0.8376
 0.4047  0.1358  0.2725  0.5413
 0.7260  0.6121  0.6860  0.9933
[torch.cuda.FloatTensor of size 5x4 (GPU 0)]
```

torch 的自动求导功能

torch 和大部分框架一样有着自动求导功能，对象不再是 torch.Tensor，而是torch.autograd.Variable

本质上Variable和Tensor没有什么区别，不过Variable会放在一个计算图里面，可以进行前向传播和反向传播以及求导



里面的creator表示通过什么操作得到的这个Variable，grad表示反向传播的梯度

In [29]:

```
from torch.autograd import Variable
```

In [49]:

```
# requires_grad 表示是否对其求梯度，默认是False
x = Variable(torch.Tensor([3]), requires_grad=True)
y = Variable(torch.Tensor([5]), requires_grad=True)
z = 2 * x + y + 4
```

In [50]:

```
# 对 x 和 y 分别求导
z.backward()
```

In [58]:

```
# x 的导数和 y 的导数
print('dz/dx: {}'.format(x.grad.data))
print('dz/dy: {}'.format(y.grad.data))
```

```
dz/dx:
 2
[torch.FloatTensor of size 1]

dz/dy:
 1
[torch.FloatTensor of size 1]
```

神经网络部分

所依赖的主要是 torch.nn 和 torch.nn.functional

torch.nn 里面有着所有的神经网络的层的操作，其用来构建网络，只有执行一次网络的运算才执行一次

torch.nn.functional 表示的是直接对其做一次向前运算操作

In [59]:

```
from torch import nn
import torch.nn.functional as F
```

In [ ]:

```
# 基本的网络构建类模板
class net_name(nn.Module):
    def __init__(self):
        super(net_name, self).__init__()
        # 可以添加各种网络层
        self.conv1 = nn.Conv2d(3, 10, 3)
        # 具体每种层的参数可以去查看文档

    def forward(self, x):
        # 定义向前传播
        out = self.conv1(x)
        return out
```