In [1]:

```
%matplotlib inline
```

What is PyTorch?

It’s a Python based scientific computing package targeted at two sets of audiences:

- A replacement for numpy to use the power of GPUs
- a deep learning research platform that provides maximum flexibility and speed

Getting Started

Tensors ^^^^^^^

Tensors are similar to numpy’s ndarrays, with the addition being that Tensors can also be used on a GPU to accelerate computing.

In [1]:

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

Construct a 5x3 matrix, uninitialized:

In [2]:

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

```
-9.0668e-36  3.0754e-41  0.0000e+00
 0.0000e+00 -7.2858e-29  4.5626e-41
-7.3271e-29  4.5626e-41 -1.2397e+22
 4.5626e-41 -7.3273e-29  4.5626e-41
 2.6807e-09  1.3296e+22  0.0000e+00
[torch.FloatTensor of size 5x3]
```

Construct a randomly initialized matrix

In [3]:

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

```
 0.1354  0.2071  0.0048
 0.4475  0.7896  0.8719
 0.1739  0.0013  0.9267
 0.6457  0.7901  0.4465
 0.7516  0.9782  0.1630
[torch.FloatTensor of size 5x3]
```

Get its size

In [4]:

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

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

> **Note**
>
> ``torch.Size`` is in fact a tuple, so it supports the same operations

Operations ^^^^^^^^^^ There are multiple syntaxes for operations. Let's see addition as an example

Addition: syntax 1

In [5]:

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

```
 0.6144  0.6115  0.6457
 1.3653  0.8366  1.6739
 0.5241  0.3456  1.6737
 0.6909  0.9689  1.1974
 1.3599  1.3690  0.8807
[torch.FloatTensor of size 5x3]
```

Addition: syntax 2

In [6]:

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

```
 0.6144  0.6115  0.6457
 1.3653  0.8366  1.6739
 0.5241  0.3456  1.6737
 0.6909  0.9689  1.1974
 1.3599  1.3690  0.8807
[torch.FloatTensor of size 5x3]
```

Addition: giving an output tensor

In [7]:

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

```
 0.6144  0.6115  0.6457
 1.3653  0.8366  1.6739
 0.5241  0.3456  1.6737
 0.6909  0.9689  1.1974
 1.3599  1.3690  0.8807
[torch.FloatTensor of size 5x3]
```

Addition: in-place

In [8]:

```
# adds x to y
y.add_(x)
print(y)
```

```
 0.6144  0.6115  0.6457
 1.3653  0.8366  1.6739
 0.5241  0.3456  1.6737
 0.6909  0.9689  1.1974
 1.3599  1.3690  0.8807
[torch.FloatTensor of size 5x3]
```

Note

Any operation that mutates a tensor in-place is post-fixed with an ``_`` For example: ``x.copy_(y)``, ``x.t_()``, will change ``x``.

You can use standard numpy-like indexing with all bells and whistles!

In [9]:

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

```
 0.2071
 0.7896
 0.0013
 0.7901
 0.9782
[torch.FloatTensor of size 5]
```

Read later:

100+ Tensor operations, including transposing, indexing, slicing, mathematical operations, linear algebra, random numbers, etc are described here <http://pytorch.org/docs/torch>_

Numpy Bridge

Converting a torch Tensor to a numpy array and vice versa is a breeze.

The torch Tensor and numpy array will share their underlying memory locations, and changing one will change the other.

Converting torch Tensor to numpy Array ^^

In [10]:

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

```
 1
 1
 1
 1
 1
[torch.FloatTensor of size 5]
```

In [11]:

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

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

See how the numpy array changed in value.

In [12]:

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

```
 2
 2
 2
 2
 2
[torch.FloatTensor of size 5]

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

Converting numpy Array to torch Tensor ^^ See how changing the np array changed the torch Tensor automatically

In [13]:

```
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.]

 2
 2
 2
 2
 2
[torch.DoubleTensor of size 5]
```

All the Tensors on the CPU except a CharTensor support converting to NumPy and back.

CUDA Tensors

Tensors can be moved onto GPU using the `.cuda` function.

In [14]:

```
# let us run this cell only if CUDA is available
if torch.cuda.is_available():
    x = x.cuda()
    y = y.cuda()
    x + y
```