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.

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

Construct a 5x3 matrix, uninitialized:

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

Out:

```
tensor(1.00000e-04 *
       [[-0.0000,  0.0000,  1.5135],
        [ 0.0000,  0.0000,  0.0000],
        [ 0.0000,  0.0000,  0.0000],
        [ 0.0000,  0.0000,  0.0000],
        [ 0.0000,  0.0000,  0.0000]])
```

Construct a randomly initialized matrix:

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

Out:

```
tensor([[ 0.8291,  0.2901,  0.0616],
        [ 0.9735,  0.9243,  0.2270],
        [ 0.4104,  0.9113,  0.5121],
        [ 0.5121,  0.5446,  0.4710],
        [ 0.2666,  0.1858,  0.5197]])
```

Construct a matrix filled zeros and of dtype long:

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

Out:

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

Construct a tensor directly from data:

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

Out:

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

or create a tensor basing on existing tensor. These methods will reuse properties of the input tensor, e.g. dtype, unless new values are provided by user

```
x = x.new_ones(5, 3, dtype=torch.double)      # new_* methods take in sizes
print(x)
x = torch.randn_like(x, dtype=torch.float)    # override dtype!
print(x)                                       # result has the same size
```

Out:

```
tensor([[ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.]], dtype=torch.float64)
tensor([[-0.2183,  0.4477, -0.4053],
        [ 1.7931, -0.0048,  1.3177],
        [-1.1311,  1.0078,  0.9722],
        [-0.7771, -0.2174,  0.0412],
        [-2.1750,  1.3609, -0.3322]])
```

Get its size:

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

Out:

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

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

Operations

There are multiple syntaxes for operations. In the following example, we will take a look at the addition operation.

Addition: syntax 1

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

Out:

```
tensor([[-0.1859,  1.3970,  0.5236],
        [ 2.3854,  0.0707,  2.1970],
        [-0.3587,  1.2359,  1.8951],
        [-0.1189, -0.1376,  0.4647],
        [-1.8968,  2.0164, -0.1092]])
```

Addition: syntax 2

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

Out:

```
tensor([[-0.1859,  1.3970,  0.5236],
        [ 2.3854,  0.0707,  2.1970],
        [-0.3587,  1.2359,  1.8951],
        [-0.1189, -0.1376,  0.4647],
        [-1.8968,  2.0164, -0.1092]])
```

Addition: providing an output tensor as argument

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

Out:

```
tensor([[-0.1859,  1.3970,  0.5236],
        [ 2.3854,  0.0707,  2.1970],
        [-0.3587,  1.2359,  1.8951],
        [-0.1189, -0.1376,  0.4647],
        [-1.8968,  2.0164, -0.1092]])
```

Addition: in-place

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

Out:

```
tensor([[-0.1859,  1.3970,  0.5236],
        [ 2.3854,  0.0707,  2.1970],
        [-0.3587,  1.2359,  1.8951],
        [-0.1189, -0.1376,  0.4647],
        [-1.8968,  2.0164, -0.1092]])
```

> **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!

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

Out:

```
tensor([ 0.4477, -0.0048,  1.0078, -0.2174,  1.3609])
```

Resizing: If you want to resize/reshape tensor, you can use torch.view:

```
x = torch.randn(4, 4)
y = x.view(16)
z = x.view(-1, 8)  # the size -1 is inferred from other dimensions
print(x.size(), y.size(), z.size())
```

Out:

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

If you have a one element tensor, use .item() to get the value as a Python number

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

Out:

```
tensor([ 0.9422])
0.9422121644020081
```

Read later:

100+ Tensor operations, including transposing, indexing, slicing, mathematical operations, linear algebra, random numbers, etc., are described here.

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 a Torch Tensor to a NumPy Array

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

Out:

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

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

Out:

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

See how the numpy array changed in value.

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

Out:

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

Converting NumPy Array to Torch Tensor

See how changing the np array changed the Torch Tensor automatically

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

Out:

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

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

CUDA Tensors

Tensors can be moved onto any device using the .to method.

```
# let us run this cell only if CUDA is available
# We will use ``torch.device`` objects to move tensors in and out of GPU
if torch.cuda.is_available():
    device = torch.device("cuda")          # a CUDA device object
    y = torch.ones_like(x, device=device)  # directly create a tensor on GPU
    x = x.to(device)                        # or just use strings ``.to("cuda")``
    z = x + y
    print(z)
    print(z.to("cpu", torch.double))       # ``.to`` can also change dtype together!
```

Out:

```
tensor([ 1.9422], device='cuda:0')
tensor([ 1.9422], dtype=torch.float64)
```

Total running time of the script: (0 minutes 0.057 seconds)

[Download Python source code: tensor_tutorial.py]   [Download Jupyter notebook: tensor_tutorial.ipynb]

Gallery generated by Sphinx-Gallery

❮ Previous   Next ❯