FVE3011

AI/ML/Pytorch

담당교수: 최 학남 xncui@inha.ac.kr

인하대학교

Pytorch

- http://pytorch.org/

1. Deep learning framework that puts Python first
2. Tensor Computation (like numpy) with strong GPU acceleration

Pytorch

https://github.com/pytorch/pytorch#from-source

A graph is created on the fly

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```

https://github.com/pytorch/pytorch/blob/master/docs/source/_static/img/dynamic_graph.gif

Installing PyTorch

- On the latest pip and numpy packages
- Anaconda is recommended package manager

START LOCALLY

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

PyTorch Build	Stable (1.13.1)	Preview (Nightly)		
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
Compute Platform	CUDA 11.6	CUDA 11.7	~~ROCm 5.2~~	CPU
Run this Command:	`conda install pytorch torchvision torchaudio pytorch-cuda=11.6 -c pytorch -c nvidia`			

NOTE: PyTorch LTS has been deprecated. For more information, see this blog.

CUDA-Enabled NVIDIA Quadro and NVIDIA RTX

NVIDIA Quadro and NVIDIA RTX Desktop GPUs

GPU	Compute Capability
RTX A6000	8.6
RTX A5000	8.6
RTX A4000	8.6
T1000	7.5
T600	7.5
T400	7.5
Quadro RTX 8000	7.5
Quadro RTX 6000	7.5
Quadro RTX 5000	7.5
Quadro RTX 4000	7.5
Quadro GV100	7.0
Quadro GP100	6.0
Quadro P6000	6.1
Quadro P5000	6.1
Quadro P4000	6.1
Quadro P2200	6.1
Quadro P2000	6.1
Quadro P1000	6.1
Quadro P620	6.1
Quadro P600	6.1
Quadro P400	6.1
Quadro M6000 24GB	5.2
Quadro M6000	5.2
Quadro K6000	3.5
Quadro M5000	5.2
Quadro K5200	3.5
Quadro K5000	3.0
Quadro M4000	5.2
Quadro K4200	3.0
Quadro K4000	3.0

NVIDIA Quadro and NVIDIA RTX Mobile GPUs

GPU	Compute Capability
RTX A5000	8.6
RTX A4000	8.6
RTX A3000	8.6
RTX A2000	8.6
RTX 5000	7.5
RTX 4000	7.5
RTX 3000	7.5
T2000	7.5
T1200	7.5
T1000	7.5
T600	7.5
T500	7.5
P620	6.1
P520	6.1
Quadro P5200	6.1
Quadro P4200	6.1
Quadro P3200	6.1
Quadro P5000	6.1
Quadro P4000	6.1
Quadro P3000	6.1
Quadro P2000	6.1
Quadro P1000	6.1
Quadro P600	6.1
Quadro P500	6.1
Quadro M5500M	5.2
Quadro M2200	5.2
Quadro M1200	5.0
Quadro M620	5.2
Quadro M520	5.0
Quadro K6000M	3.0

https://pytorch.org/get-started/locally/

Installing PyTorch

Prerequisites

1. Install Anaconda
2. Install CUDA, if your machine has a CUDA-enabled GPU.
3. If you want to build on Windows, Visual Studio with MSVC toolset, and NVTX are also needed. The exact requirements of those dependencies could be found out here.
4. Follow the steps described here: https://github.com/pytorch/pytorch#from-source

https://pytorch.org/get-started/locally/

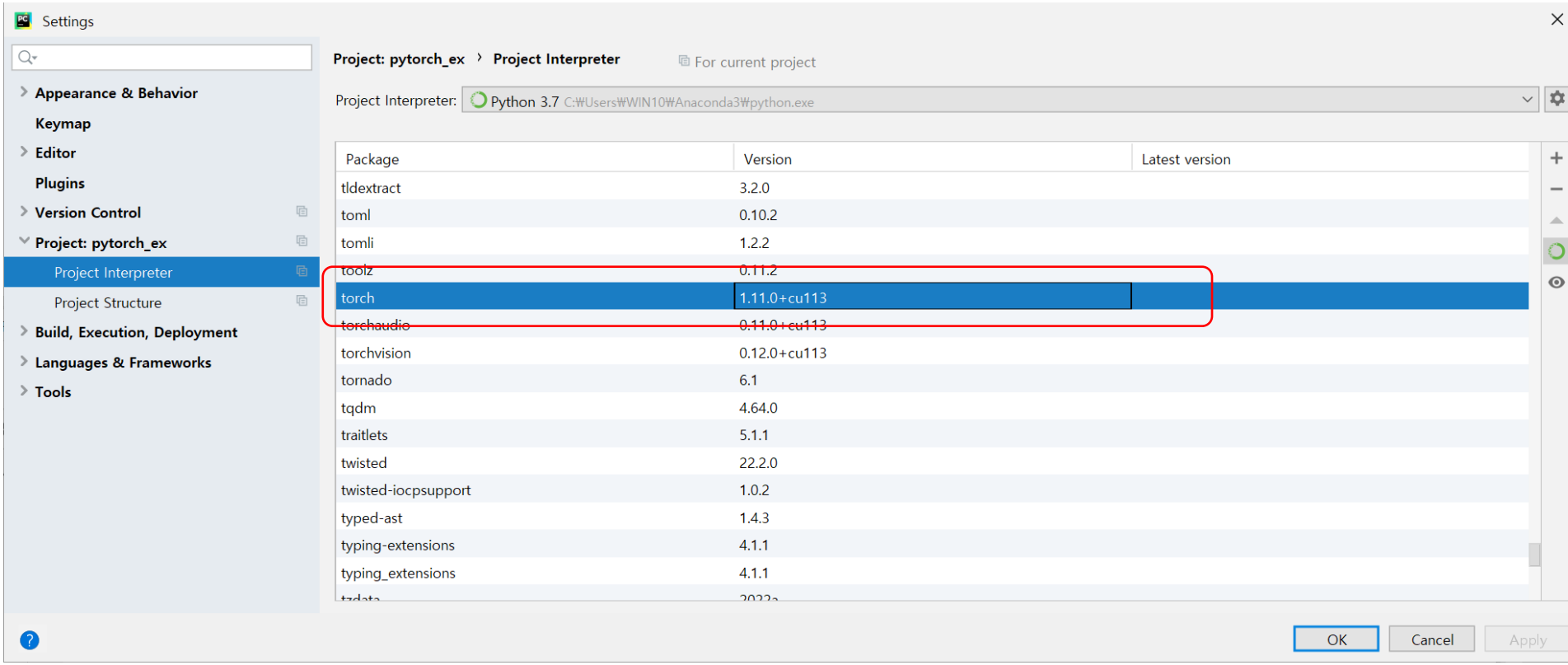
Installing PyTorch

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

```
tensor([[0.4033, 0.3859, 0.8813],
        [0.8811, 0.7242, 0.5033],
        [0.8249, 0.2634, 0.3112],
        [0.5948, 0.1092, 0.6213],
        [0.7350, 0.9898, 0.9165]])
```

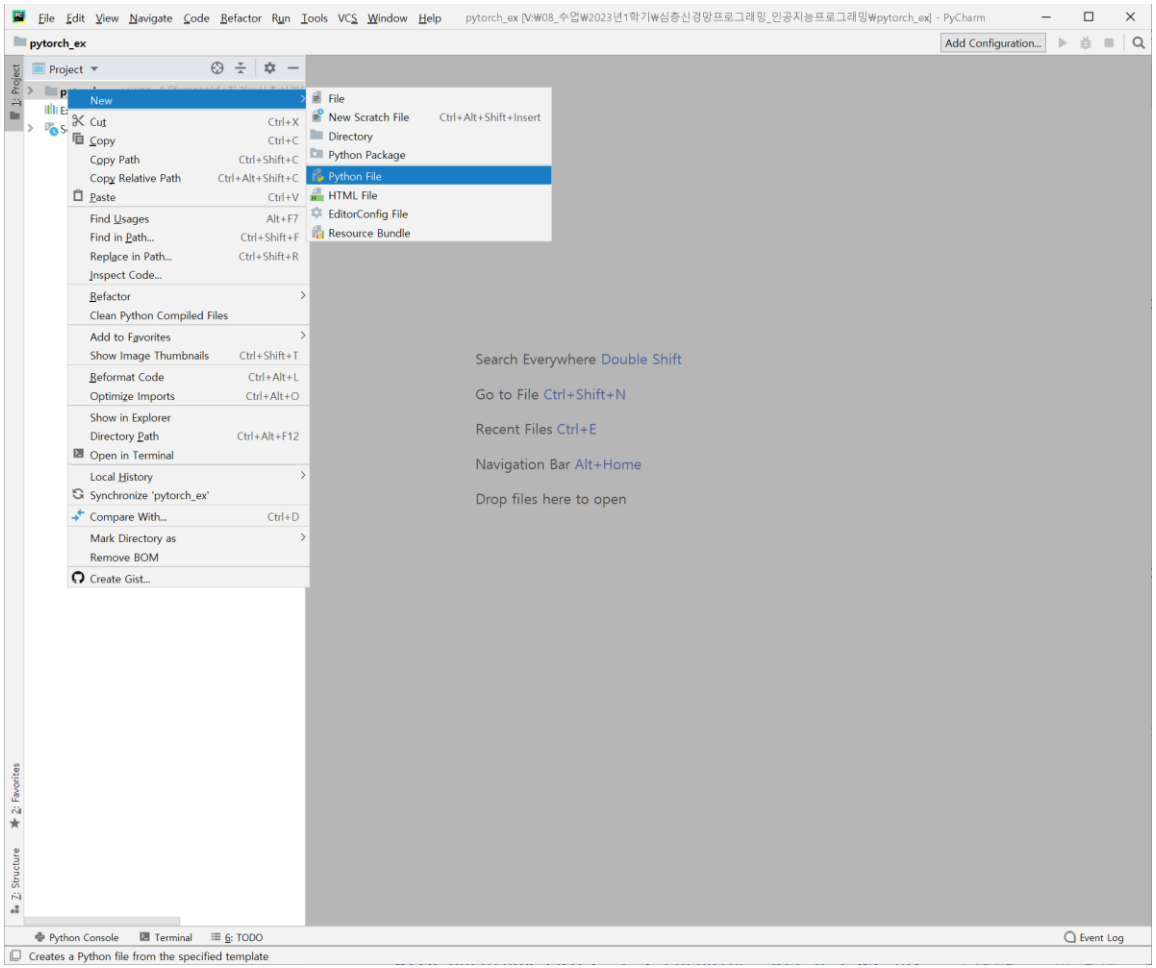
Installing pytorch(PyCharm)

- File→ settings → Project → Project interpreter

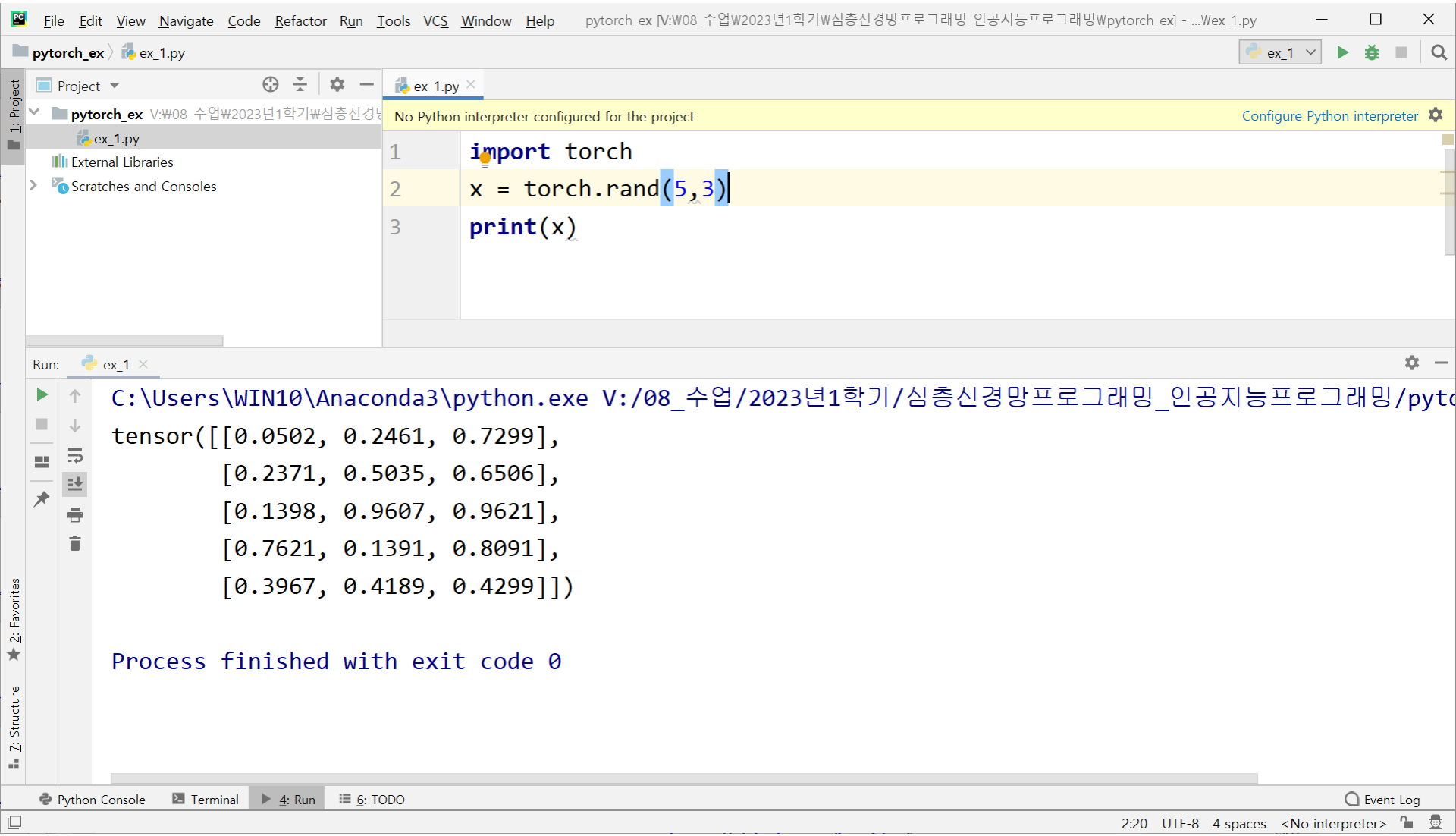


Installing pytorch(PyCharm)

- File→new project



Installing pytorch



Tensors

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

In [3]:
```
from __future__ import print_function
import torch
```

Construct a 5x3 matrix, uninitialized:

In [5]:
```
x = torch.Tensor(5,3)
print(x)
```

```
1.00000e-36 *
  0.0000  0.0000  0.0000
  0.0000  0.0000  0.0000
  0.4113  0.0000  0.0000
  0.0000  0.0001  0.0000
  1.8967  0.0000  0.0000
[torch.FloatTensor of size 5x3]
```

Tensors

Construct a randomly initialized matrix

In [9]:
```
x = torch.rand(5,3)
print(x)
```

```
 0.4381  0.1222  0.1948
 0.6345  0.0023  0.4593
 0.2548  0.3231  0.5043
 0.2990  0.9189  0.7335
 0.0187  0.8618  0.4062
[torch.FloatTensor of size 5x3]
```

Get its size

In [11]:
```
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..

Addition: syntax 1

```
In [13]: y = torch.rand(5,3)
         print(x + y)
```

```
 0.8535  0.6359  1.0222
 0.6420  0.0091  0.6165
 0.9360  0.5439  0.9757
 1.1209  1.2988  1.6813
 0.4131  1.4377  0.6229
[torch.FloatTensor of size 5x3]
```

Addition: syntax2

```
In [14]: print(torch.add(x,y))
```

```
 0.8535  0.6359  1.0222
 0.6420  0.0091  0.6165
 0.9360  0.5439  0.9757
 1.1209  1.2988  1.6813
 0.4131  1.4377  0.6229
[torch.FloatTensor of size 5x3]
```

Operations

Addition: giving an output tensor

In [16]:
```
result = torch.Tensor(5, 3)
torch.add(x, y, out=result)
print(result)
```

```
 0.8535  0.6359  1.0222
 0.6420  0.0091  0.6165
 0.9360  0.5439  0.9757
 1.1209  1.2988  1.6813
 0.4131  1.4377  0.6229
[torch.FloatTensor of size 5x3]
```

Addition: in-place

In [20]:
```
# adds x to y
y.add_(x)
print(y)
```

```
 1.2916  0.7581  1.2170
 1.2765  0.0114  1.0757
 1.1908  0.8670  1.4800
 1.4199  2.2177  2.4148
 0.4318  2.2995  1.0290
[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.

Operations

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

```
In [22]: print(x[:, 1])

 0.1222
 0.0023
 0.3231
 0.9189
 0.8618
[torch.FloatTensor of size 5]
```

- Read later: 100+ Tensor operations, including transposing, indexing, slicing, mathem atical operations, linear algebra, random numbers, etc are described here http://pyt orch.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.

Convertng torch Tensor to numpy Array

In [23]:
```
a = torch.ones(5)
print(a)
```

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

In [25]:
```
b = a.numpy()
print(b)
```

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

Numpy Bridge

See how the numpy array changed in value.

In [26]:
```
a.add_(1)
print(a)
print(b)
```

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

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

Numpy Bridge

Covering numpy Array to torch Tensor

See how changing the np array changedthe torch Tensor atomatically

In [27]:
```
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 Char Tensor support converting to Numpy and back.

CUDA Tensors

Tensors can be moved onto GPU using the .cude function..

In [32]:
```
# let us run this cell only if CUDA is available
if torch.cuda.is_available():
    x = x.cuda()
    y = y.cuda()
    x + y
    print(x + y)
```

```
 1.7296  0.8803  1.4118
 1.9110  0.0137  1.5350
 1.4456  1.1901  1.9842
 1.7189  3.1366  3.1483
 0.4506  3.1613  1.4352
[torch.cuda.FloatTensor of size 5x3 (GPU 0)]
```

- References: http://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html#tensors

CUDA Tensors