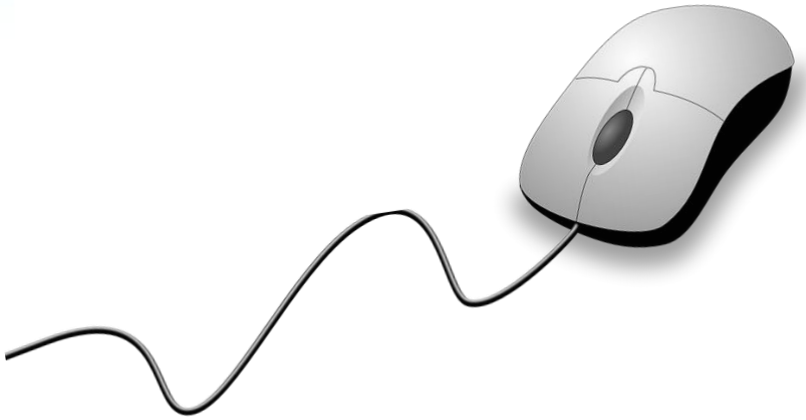
공개SW 솔루션 설치 & 활용 가이드

기타 > AI

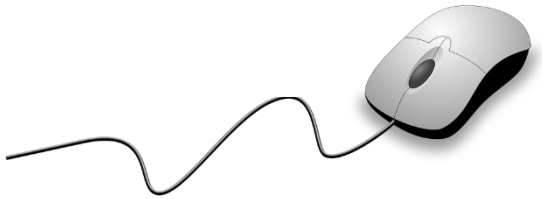


PyTorch 제대로 배워보자

How to Use Open Source Software

Open Source Software Installation & Application Guide

OPENUP
오픈소스 소프트웨어 통합지원센터
Open Source Software Support Center



CONTENTS

1. 개요
2. 기능요약
3. 실행환경
4. 설치 및 실행
5. 기능소개
6. 활용예제
7. FAQ
8. 용어정리

1. 개요

PyTorch

소개	• Python을 위한 오픈소스 머신 러닝 라이브러리 • Torch를 기반으로 하며, 자연어 처리와 같은 애플리케이션을 위해 사용 • GPU사용이 가능하기 때문에 속도가 상당히 빠르며, 사용자가 늘어나고 있는 추세 • Facebook의 인공지능 연구팀이 개발		
주요기능	• Python과 높은 호환성을 가지며, 강력한 GPU가속화를 통해 인공신경망 모델을 쉽게 구축 • 자동 미분을 사용하여 인공신경망의 역전파 단계의 연산을 자동화할 수 있도록 기능 제공		
대분류	• 기타	**소분류**	• AI
라이선스형태	• Berkeley Software Distribution (BSD)	**사전설치 솔루션**	• Python 3.x, Python 2.x
		버전	• 1.7.0 (2020년 10월 기준)
특징	• 간편한 설치, 이해와 디버깅이 쉬운 직관적이고 간결한 코드로 구성 • Define by Run 방식을 기반으로 한 실시간 결과값을 시각화 • Python Library (Numpy, Scipy, Cython etc) 와 높은 호환성을 가짐 • 자동 미분 시스템을 이용해 쉽게 Data Direct Networks (DDN) 을 구현할 수 있음		
개발회사/커뮤니티	• Facebook AI Research (https://ai.facebook.com/)		
공식 홈페이지	• https://pytorch.org/		

2. 기능요약

PyTorch

- PyTorch 주요 기능

주요기능	지원여부
32 / 64 bit OS	32 / 64 지원 (x86 / x86_64)
Pip Package	지원
Install Anaconda	지원
Install Source	지원
CPU 연산	지원
GPU 연산	지원
TPU 연산	지원

2. 기능요약

PyTorch

- PyTorch는 Python 기반의 오픈 소스 머신러닝 라이브러리로, 페이스북 인공지능 연구집단에 의해 개발되었다. 간결하고 구현이 빠르게 진행되며, TensorFlow보다 사용자가 익히기 훨씬 쉽다는 특징이 있다.

- TensorFlow는 Define-and-Run 프레임워크인 반면, PyTorch는 Define-by-Run이다.

- PyTorch는 Python 코딩과 비슷하기 때문에 언어 자체에 대한 어려움은 없으며, 선언과 동시에 데이터를 집어넣고 세션도 필요 없기 때문에, 코드가 간결하고 난이도가 낮은 편이다.

VS

3. 실행환경

PyTorch

- 하드웨어 제약이 거의 없음

주요기능	Python 2.7	Python 3.5	Python 3.6
64-bit Linux	O	O	O
CPU & GPU 지원 여부	CPU, GPU 모두 지원	CPU, GPU 모두 지원	CPU, GPU 모두 지원
Mac OS X	O	O	O
CPU & GPU 지원 여부	CPU만 지원	CPU만 지원	CPU만 지원
64-bit Windows	O	O	O
CPU & GPU 지원 여부	CPU, GPU 모두 지원	CPU, GPU 모두 지원	CPU, GPU 모두 지원

4. 설치 및 실행

세부 목차

- Windows

4.1 설치 준비 (Python 설치)

4.2 PyTorch 설치

4.3 설치 확인 및 실행

- Ubuntu (18.04 LTS)

4.1 설치 준비 (Python 설치)

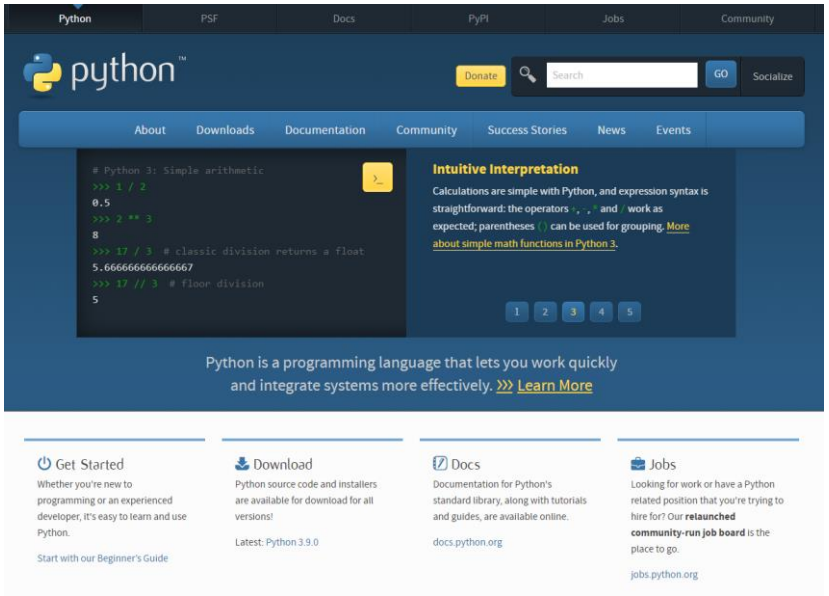
4.2 PyTorch 설치

4.3 설치 확인 및 실행

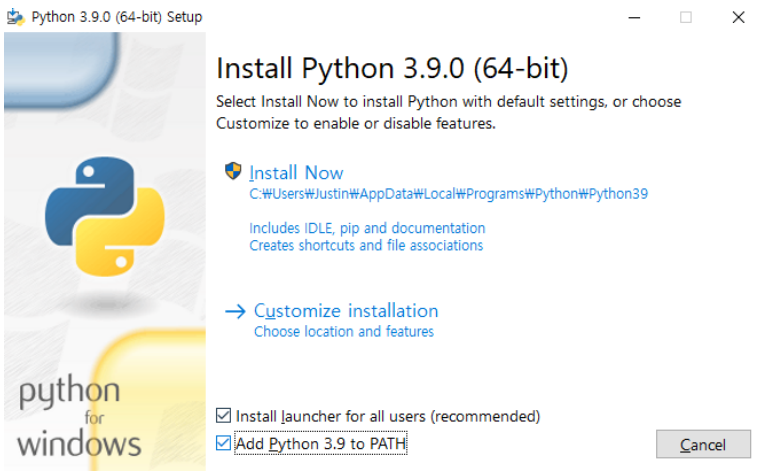
4. 설치 및 실행

4.1 설치 준비 (Python 설치) (Windows)

- Python 공식 홈페이지에서 다운로드
  - 링크 : https://www.python.org/downloads/windows/
  - Version : 3.6.x 버전을 추천 (다른 버전도 호환 가능)
  - Python 홈페이지 – Downloads – Windows 으로 접속 및 버전 선택



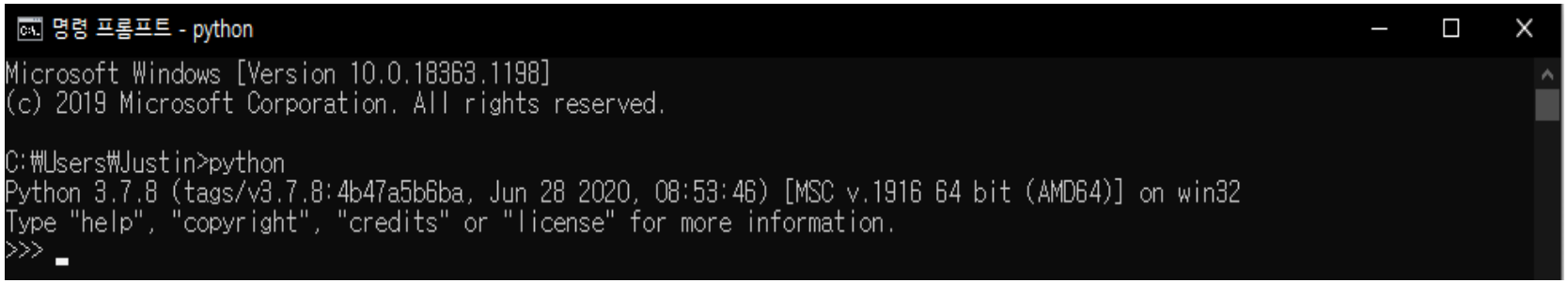
- 버전 선택 및 파일을 선택하여 다운로드 진행
  - 환경 변수 추가를 위해 Add Python 3.x to PATH 클릭
  - 이후 Install Now 버튼 및 next를 통해 설치



4. 설치 및 실행

4.1 설치 준비 (Python 설치) (Windows)

- 명령 프롬프트 창을 관리자 권한을 통해 실행
  - python 이라고 입력하여 파이썬 실행 유무 확인



4.2 PyTorch 설치 (Windows)

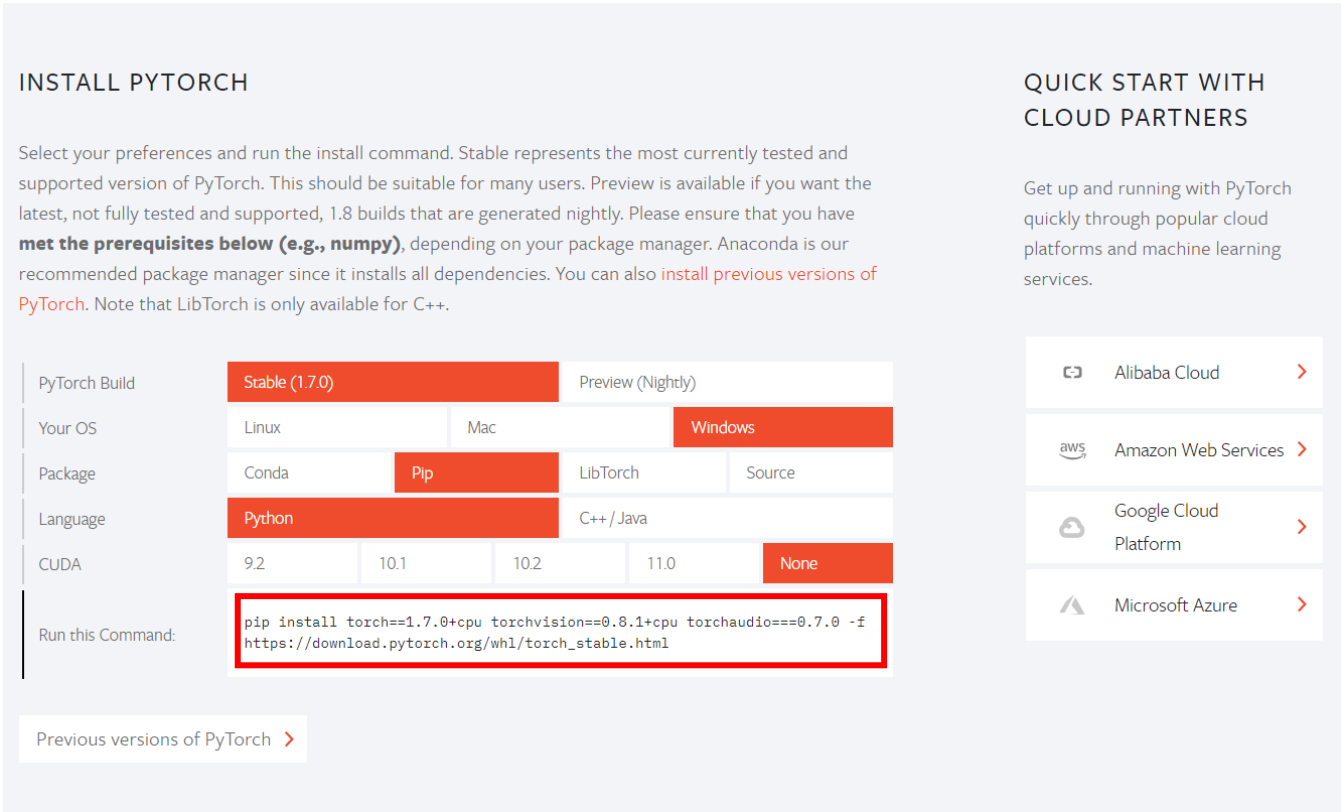
PyTorch

- PyTorch 공식 홈페이지 접속
  - INSTALL PYTORCH 항목에 본인에게 해당되는 내용 선택
    - PyTorch Build : Stable(1.7.0) 버전 설치 권장
    - Your OS : 본인이 이용하고 있는 운영체제를 선택 (Windows 선택)
    - Package : Python을 설치했으므로 Pip 선택, 만약 아나콘다를 설치하였으면 Conda 선택
    - Language : Python을 설치했으므로 Python을 선택,
      만약 C++, Java를 선택하였으면 C++, Java를 선택
    - CUDA : 딥러닝 모델을 학습할 때, GPU를 이용하는 경우 CUDA를 설치하여 버전을 선택,
      만약 CPU로 딥러닝 모델을 학습할 경우, None을 선택
    - Run this Command : 해당되는 내용을 복사하여 명령 프롬프트창에 입력하여 실행

4. 설치 및 실행

4.2 PyTorch 설치 (Windows)

- PyTorch 공식 홈페이지 접속 화면



출력되는 빨간색 네모 안의 코드를 명령프롬프트 창에 입력하여 실행

4. 설치 및 실행

4.3 PyTorch 설치 확인 및 실행 (Windows)

- PyTorch을 Python 콘솔에서 실행

```
Python 3.7.8 (tags/v3.7.8:4b47a5b6ba, Jun 28 2020, 08:53:46) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.__version__
'1.6.0+cu101'
```

4.1 설치 준비 (Python 설치) (Ubuntu 18.04 LTS)

PyTorch

- Terminal 을 통해 Python 설치

 - 명령어

  - sudo apt update
  - sudo apt install software-properties-common
  - sudo add-apt-repository ppa:deadsnakes/ppa
  - sudo apt update
  - sudo apt install python 3.6 # Python 3.6 version
  - sudo apt install python 3.7 # Python 3.7 version

4. 설치 및 실행

4.1 설치 준비 (Python 설치) (Ubuntu 18.04 LTS)

- Terminal 을 실행하여 명령어를 통해
  - python3.6 혹은 python3.7 이라고 입력하여 파이썬 실행 유무 확인

```
justin@Justin:~$ python3.6
Python 3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

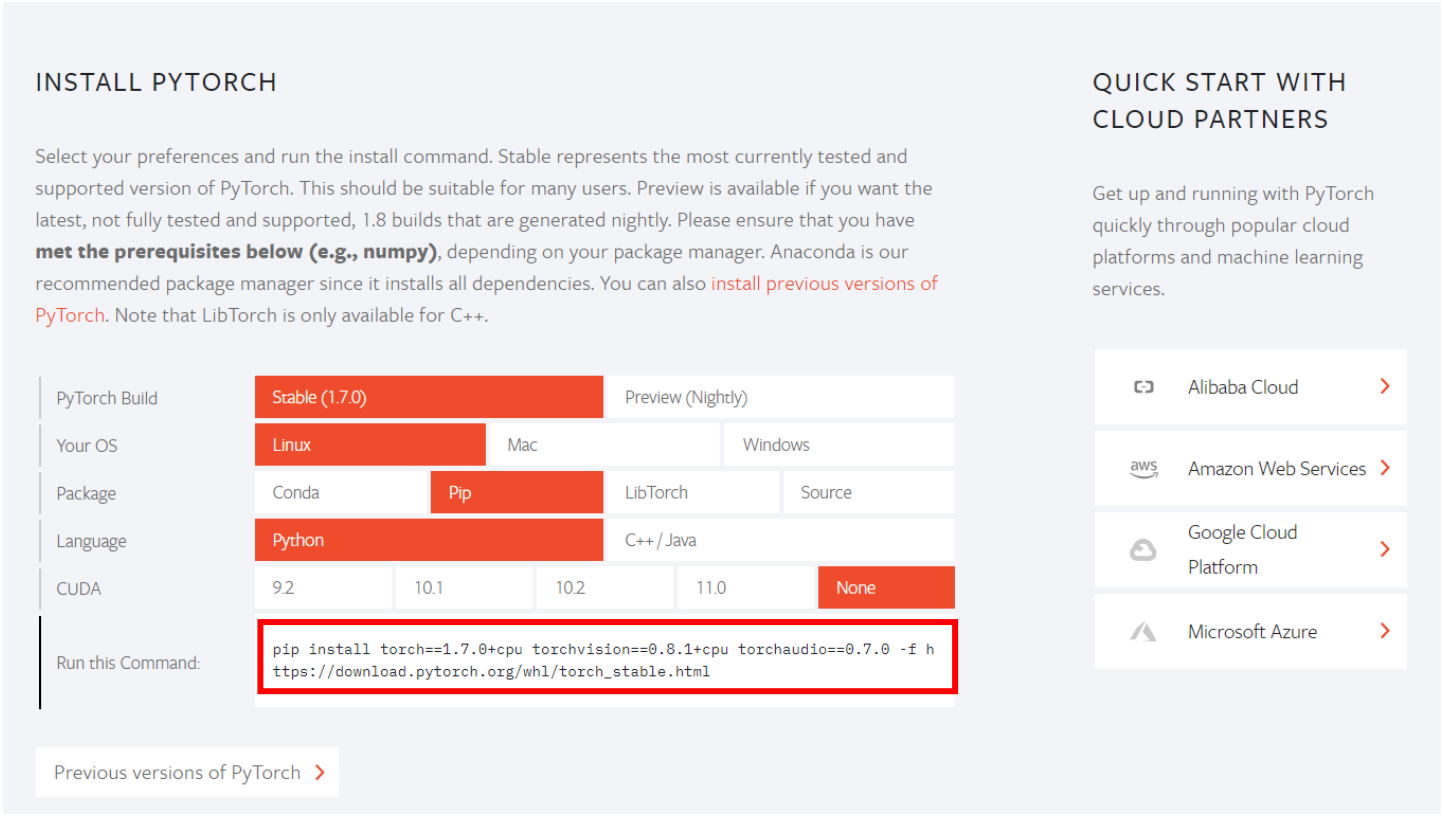
4.2 PyTorch 설치 (Ubuntu 18.04 LTS)

- PyTorch 공식 홈페이지 접속
  - INSTALL PYTORCH 항목에 본인에게 해당되는 내용 선택
    - PyTorch Build : Stable(1.7.0) 버전 설치 권장
    - Your OS : 본인이 이용하고 있는 운영체제를 선택 (Linux 선택)
    - Package : Python을 설치했으므로 Pip 선택, 만약 아나콘다를 설치하였으면 Conda 선택
    - Language : Python을 설치했으므로 Python을 선택,
      만약 C++, Java를 선택하였으면 C++, Java를 선택
    - CUDA : 딥러닝 모델을 학습할 때, GPU를 이용하는 경우 CUDA를 설치하여 버전을 선택,
      만약 CPU로 딥러닝 모델을 학습할 경우, None을 선택
    - Run this Command : 해당되는 내용을 복사하여 Terminal에 입력하여 실행

4. 설치 및 실행

4.2 PyTorch 설치 (Ubuntu 18.04 LTS)

- PyTorch 공식 홈페이지 접속 화면



출력되는 빨간색 네모 안의 코드를 터미널 창에 입력하여 실행

4. 설치 및 실행

4.3 PyTorch 설치 확인 및 실행 (Ubuntu 18.04 LTS)

- PyTorch을 터미널창에 설치하는 과정

```
(py36) justin@Justin:~$ pip install torch==1.7.0+cu101 torchvision==0.8.1+cu101 torchaudio==0.7.0 -f https://download.pytorch.org/whl/torch_stable.html
Looking in links: https://download.pytorch.org/whl/torch_stable.html
Collecting torch==1.7.0+cu101
  Downloading https://download.pytorch.org/whl/cu101/torch-1.7.0%2Bcu101-cp36-cp36m-linux_x86_64.whl (735.3 MB)
     |████████████████████████████████| 735.3 MB 105 kB/s
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None)) after connection broken by 'ProtocolError('Connection aborted.', ConnectionResetError(104, 'Connection reset by peer'))': /simple/torchvision/
Collecting torchvision==0.8.1+cu101
  Downloading https://download.pytorch.org/whl/cu101/torchvision-0.8.1%2Bcu101-cp36-cp36m-linux_x86_64.whl (12.8 MB)
     |████████████████████████████████| 12.8 MB 6.1 MB/s
Collecting torchaudio==0.7.0
  Downloading torchaudio-0.7.0-cp36-cp36m-manylinux1_x86_64.whl (7.6 MB)
     |████████████████████████████████| 7.6 MB 5.2 MB/s
Collecting dataclasses
  Using cached dataclasses-0.7-py3-none-any.whl (18 kB)
Collecting typing-extensions
  Downloading typing_extensions-3.7.4.3-py3-none-any.whl (22 kB)
Collecting numpy
  Downloading numpy-1.19.4-cp36-cp36m-manylinux2010_x86_64.whl (14.5 MB)
     |████████████████████████████████| 14.5 MB 5.1 MB/s
Processing ./.cache/pip/wheels/6e/9c/ed/4499c9865ac1002697793e0ae05ba6be33553d098f3347fb94/future-0.18.2-py3-none-any.whl
Collecting pillow>=4.1.1
  Downloading Pillow-8.0.1-cp36-cp36m-manylinux1_x86_64.whl (2.2 MB)
     |████████████████████████████████| 2.2 MB 5.6 MB/s
Installing collected packages: dataclasses, typing-extensions, numpy, future, torch, pillow, torchvision, torchaudio
Successfully installed dataclasses-0.7 future-0.18.2 numpy-1.19.4 pillow-8.0.1 torch-1.7.0+cu101 torchaudio-0.7.0 torchvision-0.8.1+cu101 typing-extensions-3.7.4.3
```

- PyTorch을 Python 콘솔에서 실행

```
(py36) justin@Justin:~$ python
Python 3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.__version__
'1.7.0+cu101'
>>>
```

5. 기능소개

세부 목차

5.1 연산 정의하기

5.2 자동미분 계산하기

5. 기능소개

5.1 연산 정의하기

- Tensor를 자유롭게 다루기 위해 필요한 기본 연산 정의 및 실습하기
  - import torch 로 PyTorch 라이브러리를 불러옴
  - Scalar, Vector, Matrix, Tensor 형식의 자료구조를 정의함
  - 정의된 자료구조에 대해 기초 연산을 정의하여 계산

```
In [1]: import torch
```

Scalar

```
In [2]: scalar1 = torch.tensor([1.])
        print(scalar1)
tensor([1.])

In [3]: scalar2 = torch.tensor([3.])
        print(scalar2)
tensor([3.])

In [4]: add_scalar = scalar1 + scalar2
        print(add_scalar)
tensor([4.])

In [5]: sub_scalar = scalar1 - scalar2
        print(sub_scalar)
tensor([-2.])

In [6]: mul_scalar = scalar1 * scalar2
        print(mul_scalar)
tensor([3.])

In [7]: div_scalar = scalar1 / scalar2
        print(div_scalar)
tensor([0.3333])
```

```
In [8]: torch.add(scalar1, scalar2)
Out[8]: tensor([4.])

In [9]: torch.sub(scalar1, scalar2)
Out[9]: tensor([-2.])

In [10]: torch.mul(scalar1, scalar2)
Out[10]: tensor([3.])

In [11]: torch.div(scalar1, scalar2)
Out[11]: tensor([0.3333])
```

Scalar 자료구조에 대한 연산 정의 및 실행 예제

PyTorch

Vector

```
In [12]: vector1 = torch.tensor([1., 2., 3.])
         print(vector1)
tensor([1., 2., 3.])

In [13]: vector2 = torch.tensor([4., 5., 6.])
         print(vector2)
tensor([4., 5., 6.])

In [14]: add_vector = vector1 + vector2
         print(add_vector)
tensor([5., 7., 9.])

In [15]: sub_vector = vector1 - vector2
         print(sub_vector)
tensor([-3., -3., -3.])

In [16]: mul_vector = vector1 * vector2
         print(mul_vector)
tensor([ 4., 10., 18.])

In [17]: div_vector = vector1 / vector2
         print(div_vector)
tensor([0.2500, 0.4000, 0.5000])
```

```
In [18]: torch.add(vector1, vector2)
Out[18]: tensor([5., 7., 9.])

In [19]: torch.sub(vector1, vector2)
Out[19]: tensor([-3., -3., -3.])

In [20]: torch.mul(vector1, vector2)
Out[20]: tensor([ 4., 10., 18.])

In [21]: torch.div(vector1, vector2)
Out[21]: tensor([0.2500, 0.4000, 0.5000])

In [22]: torch.dot(vector1, vector2)
Out[22]: tensor(32.)
```

Vector 자료구조에 대한 연산 정의 및 실행 예제

PyTorch

Matrix

```
In [23]: matrix1 = torch.tensor([[1., 2.], [3., 4.]])
         print(matrix1)

tensor([[1., 2.],
        [3., 4.]])
```

```
In [24]: matrix2 = torch.tensor([[5., 6.], [7., 8.]])
         print(matrix2)

tensor([[5., 6.],
        [7., 8.]])
```

```
In [25]: sum_matrix = matrix1 + matrix2
         print(sum_matrix)

tensor([[ 6.,  8.],
        [10., 12.]])
```

```
In [26]: sub_matrix = matrix1 - matrix2
         print(sub_matrix)

tensor([[-4., -4.],
        [-4., -4.]])
```

```
In [27]: mul_matrix = matrix1 * matrix2
         print(mul_matrix)

tensor([[ 5., 12.],
        [21., 32.]])
```

```
In [28]: div_matrix = matrix1 / matrix2
         print(div_matrix)

tensor([[0.2000, 0.3333],
        [0.4286, 0.5000]])
```

```
In [29]: torch.add(matrix1, matrix2)

Out[29]: tensor([[ 6.,  8.],
                 [10., 12.]])
```

```
In [30]: torch.sub(matrix1, matrix2)

Out[30]: tensor([[-4., -4.],
                 [-4., -4.]])
```

```
In [31]: torch.mul(matrix1, matrix2)

Out[31]: tensor([[ 5., 12.],
                 [21., 32.]])
```

```
In [32]: torch.div(matrix1, matrix2)

Out[32]: tensor([[0.2000, 0.3333],
                 [0.4286, 0.5000]])
```

```
In [33]: torch.matmul(matrix1, matrix2)

Out[33]: tensor([[19., 22.],
                 [43., 50.]])
```

Matrix 자료구조에 대한 연산 정의 및 실행 예제

PyTorch

Tensor

```
In [34]: tensor1 = torch.tensor([[[1., 2.], [3., 4.]], [[5., 6.], [7., 8.]]])
         print(tensor1)

tensor([[[1., 2.],
         [3., 4.]],

        [[5., 6.],
         [7., 8.]]])
```

```
In [35]: tensor2 = torch.tensor([[[9., 10.], [11., 12.]], [[13., 14.], [15., 16.]]])
         print(tensor2)

tensor([[[ 9., 10.],
         [11., 12.]],

        [[13., 14.],
         [15., 16.]]])
```

```
In [36]: sum_tensor = tensor1 + tensor2
         print(sum_tensor)

tensor([[[10., 12.],
         [14., 16.]],

        [[18., 20.],
         [22., 24.]]])
```

```
In [37]: sub_tensor = tensor1 - tensor2
         print(sub_tensor)

tensor([[[-8., -8.],
         [-8., -8.]],

        [[-8., -8.],
         [-8., -8.]]])
```

```
In [38]: mul_tensor = tensor1 * tensor2
         print(mul_tensor)

tensor([[[  9.,  20.],
         [ 33.,  48.]],

        [[ 65.,  84.],
         [105., 128.]]])
```

```
In [39]: div_tensor = tensor1 / tensor2
         print(div_tensor)

tensor([[[0.1111, 0.2000],
         [0.2727, 0.3333]],

        [[0.3846, 0.4286],
         [0.4667, 0.5000]]])
```

```
In [40]: torch.matmul(tensor1, tensor2)
Out[40]: tensor([[[ 31.,  34.],
                  [ 71.,  78.]],

                 [[155., 166.],
                  [211., 226.]]])
```

```
In [41]: torch.add(tensor1, tensor2)
Out[41]: tensor([[[10., 12.],
                  [14., 16.]],

                 [[18., 20.],
                  [22., 24.]]])
```

```
In [42]: torch.sub(tensor1, tensor2)
Out[42]: tensor([[[-8., -8.],
                  [-8., -8.]],

                 [[-8., -8.],
                  [-8., -8.]]])
```

```
In [43]: torch.mul(tensor1, tensor2)
Out[43]: tensor([[[  9.,  20.],
                  [ 33.,  48.]],

                 [[ 65.,  84.],
                  [105., 128.]]])
```

```
In [44]: torch.div(tensor1, tensor2)
Out[44]: tensor([[[0.1111, 0.2000],
                  [0.2727, 0.3333]],

                 [[0.3846, 0.4286],
                  [0.4667, 0.5000]]])
```

```
In [45]: tensor1
Out[45]: tensor([[[1., 2.],
                  [3., 4.]],

                 [[5., 6.],
                  [7., 8.]]])
```

```
In [46]: tensor2
Out[46]: tensor([[[ 9., 10.],
                  [11., 12.]],

                 [[13., 14.],
                  [15., 16.]]])
```

```
In [47]: torch.matmul(tensor1, tensor2)
Out[47]: tensor([[[ 31.,  34.],
                  [ 71.,  78.]],

                 [[155., 166.],
                  [211., 226.]]])
```

Matrix 자료구조에 대한 연산 정의 및 실행 예제

In [48]:
```
import torch

if torch.cuda.is_available():
    DEVICE = torch.device('cuda')
else:
    DEVICE = torch.device('cpu')

BATCH_SIZE = 64
INPUT_SIZE = 1000
HIDDEN_SIZE = 100
OUTPUT_SIZE = 10

x = torch.randn(BATCH_SIZE,
                INPUT_SIZE,
                device = DEVICE,
                dtype = torch.float,
                requires_grad = False)

y = torch.randn(BATCH_SIZE,
                OUTPUT_SIZE,
                device = DEVICE,
                dtype = torch.float,
                requires_grad = False)

w1 = torch.randn(INPUT_SIZE,
                 HIDDEN_SIZE,
                 device = DEVICE,
                 dtype = torch.float,
                 requires_grad = True)

w2 = torch.randn(HIDDEN_SIZE,
                 OUTPUT_SIZE,
                 device = DEVICE,
                 dtype = torch.float,
                 requires_grad = True)

learning_rate = 1e-6
```

자동미분 계산하기 코드 (1)

자동미분 계산하기 코드 (2)

```
for t in range(1, 501):
    y_pred = x.mm(w1).clamp(min = 0).mm(w2)

    loss = (y_pred - y).pow(2).sum()
    if t % 100 == 0:
        print("Iteration: ", t, "Wt", "Loss: ", loss.item())
    loss.backward()

    with torch.no_grad():
        w1 -= learning_rate * w1.grad
        w2 -= learning_rate * w2.grad

        w1.grad.zero_()
        w2.grad.zero_()
```

```
Iteration:  100         Loss:  527.76318359375
Iteration:  200         Loss:  3.209841728210449
Iteration:  300         Loss:  0.03574322536587715
Iteration:  400         Loss:  0.0007254641968756914
Iteration:  500         Loss:  8.230483217630535e-05
```

- PyTorch를 이용해 역전파를 자동으로 계산하는 Autograd 실습하기
  - w1, w2 값이 역전파에 의해 업데이트 되는 내용의 스크립트 파일

6. 활용예제

세부 목차

6.1 손글씨 숫자를 인식하는 인공신경망 모델 구축하기 (환경 설정)

6.2 손글씨 숫자 인식을 위해 데이터 셋을 불러오기

6.3 학습 모델 정의

6.4 모델 학습 진행

6.5 모델 평가 진행

6. 활용예제

6.1 손글씨 숫자를 인식하는 인공신경망 모델 구축하기 (환경 설정)

PyTorch

In [1]:
```
''' 1. Module Import '''
import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import transforms, datasets
```

활용 예제에 관련된 모듈 불러오기

In [2]:
```
''' 2. 딥러닝 모델을 설계할 때 활용하는 장비 확인 '''
if torch.cuda.is_available():
    DEVICE = torch.device('cuda')
else:
    DEVICE = torch.device('cpu')

print('Using PyTorch version:', torch.__version__, ' Device:', DEVICE)
```

연산 장치 설정 (CPU 혹은 GPU)

```
Using PyTorch version: 1.6.0+cu101  Device: cuda
```

In [3]:
```
BATCH_SIZE = 32
EPOCHS = 10
```

인공신경망 모델이 학습될 때 학습되는 횟수 및 단위 설정

6. 활용예제

6.2 손글씨 숫자 인식을 위해 데이터 셋을 불러오기

PyTorch

- 손글씨 숫자 인식 데이터 셋 다운로드 및 불러오기

In [4]:
```
''' 3. MNIST 데이터 다운로드 (Train set, Test set 분리하기) '''
train_dataset = datasets.MNIST(root = "../data/MNIST",
                               train = True,
                               download = True,
                               transform = transforms.ToTensor())

test_dataset = datasets.MNIST(root = "../data/MNIST",
                              train = False,
                              transform = transforms.ToTensor())

train_loader = torch.utils.data.DataLoader(dataset = train_dataset,
                                           batch_size = BATCH_SIZE,
                                           shuffle = True)

test_loader = torch.utils.data.DataLoader(dataset = test_dataset,
                                          batch_size = BATCH_SIZE,
                                          shuffle = False)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../data/MNIST₩MNIST₩raw₩train-images-idx3-ubyte.gz
100.1%
Extracting ../data/MNIST₩MNIST₩raw₩train-images-idx3-ubyte.gz to ../data/MNIST₩MNIST₩raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../data/MNIST₩MNIST₩raw₩train-labels-idx1-ubyte.gz
113.5%
Extracting ../data/MNIST₩MNIST₩raw₩train-labels-idx1-ubyte.gz to ../data/MNIST₩MNIST₩raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ../data/MNIST₩MNIST₩raw₩t10k-images-idx3-ubyte.gz
100.4%
Extracting ../data/MNIST₩MNIST₩raw₩t10k-images-idx3-ubyte.gz to ../data/MNIST₩MNIST₩raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ../data/MNIST₩MNIST₩raw₩t10k-labels-idx1-ubyte.gz
```

- 손글씨 숫자를 인식하는 학습 모델 설계하기

In [7]:
```
''' 6. Multi Layer Perceptron (MLP) 모델 설계하기 '''
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 10)

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = self.fc1(x)
        x = F.sigmoid(x)
        x = self.fc2(x)
        x = F.sigmoid(x)
        x = self.fc3(x)
        x = F.log_softmax(x, dim = 1)
        return x
```

In [8]:
```
''' 7. Optimizer, Objective Function 설정하기 '''
model = Net().to(DEVICE)
optimizer = torch.optim.SGD(model.parameters(), lr = 0.01, momentum = 0.5)
criterion = nn.CrossEntropyLoss()

print(model)
```

```
Net(
  (fc1): Linear(in_features=784, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=10, bias=True)
)
```

6. 활용예제

6.3 학습 모델 정의

PyTorch

- 손글씨 숫자를 인식하기 위해 설계한 인공신경망 모델을 학습하는 함수 정의하기

In [9]:
```
''' 8. MLP 모델 학습을 진행하며 학습 데이터에 대한 모델 성능을 확인하는 함수 정의 '''
def train(model, train_loader, optimizer, log_interval):
    model.train()
    for batch_idx, (image, label) in enumerate(train_loader):
        image = image.to(DEVICE)
        label = label.to(DEVICE)
        optimizer.zero_grad()
        output = model(image)
        loss = criterion(output, label)
        loss.backward()
        optimizer.step()

        if batch_idx % log_interval == 0:
            print("Train Epoch: {} [{}/{} ({:.0f}%)]₩tTrain Loss: {:.6f}".format(
                epoch, batch_idx * len(image),
                len(train_loader.dataset), 100. * batch_idx / len(train_loader),
                loss.item()))
```

6. 활용예제

6.3 학습 모델 정의

- 학습이 완료된 인공신경망 모델을 검증하는 함수 정의하기

In [10]:
```
''' 9. 학습되는 과정 속에서 검증 데이터에 대한 모델 성능을 확인하는 함수 정의 '''
def evaluate(model, test_loader):
    model.eval()
    test_loss = 0
    correct = 0

    with torch.no_grad():
        for image, label in test_loader:
            image = image.to(DEVICE)
            label = label.to(DEVICE)
            output = model(image)
            test_loss += criterion(output, label).item()
            prediction = output.max(1, keepdim = True)[1]
            correct += prediction.eq(label.view_as(prediction)).sum().item()

    test_loss /= len(test_loader.dataset)
    test_accuracy = 100. * correct / len(test_loader.dataset)
    return test_loss, test_accuracy
```

6. 활용예제

6.4 모델 학습 진행

PyTorch

- 손글씨 숫자를 인식하기 위해 설계한 인공신경망 모델을 학습하기

In [11]:
```
''' 10. MLP 학습 실행하며 Train, Test set의 Loss 및 Test set Accuracy 확인하기 '''
for epoch in range(1, EPOCHS + 1):
    train(model, train_loader, optimizer, log_interval = 200)
    test_loss, test_accuracy = evaluate(model, test_loader)
    print("₩n[EPOCH: {}], ₩tTest Loss: {:.4f}, ₩tTest Accuracy: {:.2f} % ₩n".format(
        epoch, test_loss, test_accuracy))
```

```
c:₩users₩justin₩101₩lib₩site-packages₩torch₩nn₩functional.py:1625: UserWarning: nn.functional.sigmoid
is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
Train Epoch: 1 [0/60000 (0%)]   Train Loss: 2.270418
Train Epoch: 1 [6400/60000 (11%)]       Train Loss: 2.314499
Train Epoch: 1 [12800/60000 (21%)]      Train Loss: 2.337222
Train Epoch: 1 [19200/60000 (32%)]      Train Loss: 2.327368
Train Epoch: 1 [25600/60000 (43%)]      Train Loss: 2.300396
Train Epoch: 1 [32000/60000 (53%)]      Train Loss: 2.318115
Train Epoch: 1 [38400/60000 (64%)]      Train Loss: 2.284231
Train Epoch: 1 [44800/60000 (75%)]      Train Loss: 2.300245
Train Epoch: 1 [51200/60000 (85%)]      Train Loss: 2.254090
Train Epoch: 1 [57600/60000 (96%)]      Train Loss: 2.202278

[EPOCH: 1],     Test Loss: 0.0697,      Test Accuracy: 29.01 %

Train Epoch: 2 [0/60000 (0%)]   Train Loss: 2.242800
Train Epoch: 2 [6400/60000 (11%)]       Train Loss: 2.179223
Train Epoch: 2 [12800/60000 (21%)]      Train Loss: 2.138005
Train Epoch: 2 [19200/60000 (32%)]      Train Loss: 2.170197
Train Epoch: 2 [25600/60000 (43%)]      Train Loss: 1.979321
Train Epoch: 2 [32000/60000 (53%)]      Train Loss: 1.977249
Train Epoch: 2 [38400/60000 (64%)]      Train Loss: 1.841060
Train Epoch: 2 [44800/60000 (75%)]      Train Loss: 1.578944
Train Epoch: 2 [51200/60000 (85%)]      Train Loss: 1.333097
Train Epoch: 2 [57600/60000 (96%)]      Train Loss: 1.292717

[EPOCH: 2],     Test Loss: 0.0405,      Test Accuracy: 59.94 %

Train Epoch: 3 [0/60000 (0%)]   Train Loss: 1.350040
Train Epoch: 3 [6400/60000 (11%)]       Train Loss: 1.108275
```

- 학습이 완료될 때마다 평가를 진행하며 최종으로 약 90% 수준의 정확도를 확인할 수 있음

```
[EPOCH: 8],     Test Loss: 0.0113,      Test Accuracy: 89.59 %

Train Epoch: 9 [0/60000 (0%)]   Train Loss: 0.403923
Train Epoch: 9 [6400/60000 (11%)]       Train Loss: 0.178136
Train Epoch: 9 [12800/60000 (21%)]      Train Loss: 0.458371
Train Epoch: 9 [19200/60000 (32%)]      Train Loss: 0.349120
Train Epoch: 9 [25600/60000 (43%)]      Train Loss: 0.428086
Train Epoch: 9 [32000/60000 (53%)]      Train Loss: 0.580726
Train Epoch: 9 [38400/60000 (64%)]      Train Loss: 0.105964
Train Epoch: 9 [44800/60000 (75%)]      Train Loss: 0.393156
Train Epoch: 9 [51200/60000 (85%)]      Train Loss: 0.165475
Train Epoch: 9 [57600/60000 (96%)]      Train Loss: 0.313095

[EPOCH: 9],     Test Loss: 0.0108,      Test Accuracy: 89.92 %

Train Epoch: 10 [0/60000 (0%)]  Train Loss: 0.578675
Train Epoch: 10 [6400/60000 (11%)]      Train Loss: 0.174825
Train Epoch: 10 [12800/60000 (21%)]     Train Loss: 0.413739
Train Epoch: 10 [19200/60000 (32%)]     Train Loss: 0.182302
Train Epoch: 10 [25600/60000 (43%)]     Train Loss: 0.223432
Train Epoch: 10 [32000/60000 (53%)]     Train Loss: 0.610788
Train Epoch: 10 [38400/60000 (64%)]     Train Loss: 0.158630
Train Epoch: 10 [44800/60000 (75%)]     Train Loss: 0.335066
Train Epoch: 10 [51200/60000 (85%)]     Train Loss: 0.237495
Train Epoch: 10 [57600/60000 (96%)]     Train Loss: 0.469383

[EPOCH: 10],    Test Loss: 0.0105,      Test Accuracy: 90.33 %
```

PyTorch

Q&A PyTorch는 TensorFlow보다 장점이 무엇인가요?

PyTorch 와 TensorFlow의 가장 큰 차이점은 바로 딥러닝을 구현하는 방식이 다르다는 점입니다. TensorFlow는 Placeholder를 선언하고, 이것으로 계산 그래프를 만들어서 코드를 실행하는 직관적이지 않지만, PyTorch는 선언 및 데이터를 입력하는 방식이여서 코드가 간결하고 난이도가 쉬운 장점이 있습니다.

Q&A PyTorch에서 유용한 패키지는 무엇이 있나요?

PyTorch에는 다양한 패키지들을 통해 인공신경망 모델을 쉽게 구축할 수 있습니다. 예를 들어, autograd 패키지를 이용하여 자동 미분을 계산하고, nn 패키지를 통해 인공신경망 모델을 쉽게 구현하며, optim 패키지를 이용하여 코드 1줄로 최적화 기법을 쉽게 적용할 수 있습니다.

8. 용어정리

PyTorch

용어	설명
파이썬	1991년 프로그래머인 귀도 반 로섬 (Guido van Rossum)이 발표한 고급 프로그래밍 언어로, 플랫폼 독립적이며 인터프리터식, 객체지향적, 동적타이핑(dynamically typed) 대화형 언어
PyTorch	Python을 위한 오픈소스 머신 러닝 라이브러리
Define-and-Run	인공신경망 모델을 정의하며, 이는 고정된 형태로 데이터를 입력받는 형식
Define-by-Run	인공신경망 모델이 역동적으로 정의되며, 쉽고 다양한 방식으로 모델을 정의 및 실험하는 형식
인공신경망	기계학습과 인지과학에서 생물학의 신경망에서 영감을 얻은 통계학적 학습 알고리즘
BATCH_SIZE	인공신경망 모델을 학습할 때 오차값을 계산하여 가중치를 업데이트하기 위해 계산되는 데이터 개수 단위
EPOCHS	전체 데이터에 대해서 인공신경망 모델을 학습할 때 학습을 진행하는 전체 횟수
LEARNING_RATE	BATCH_SIZE를 통해 계산된 오차값을 활용하여 가중치값을 업데이트 할 때, 업데이트 되는 단위를 조정하는 매개변수

Open Source Software Installation & Application Guide

이 저작물은 크리에이티브 커먼즈 [저작자표시 – 비영리 – 동일조건 변경허락 2.0 대한민국 라이선스]에 따라 이용하실 수 있습니다.