딥러닝 프레임워크 : PyTorch

이건명
충북대학교 소프트웨어학과

학습 내용

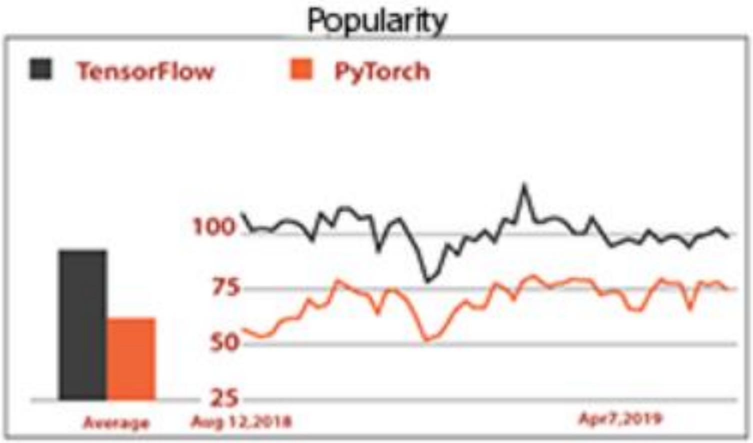
- **PyTorch 기본 이론에 대해서 알아본다.**
- **PyTorch를 이용한 딥러닝 프로그래밍 사례에 대해서 살펴본다.**

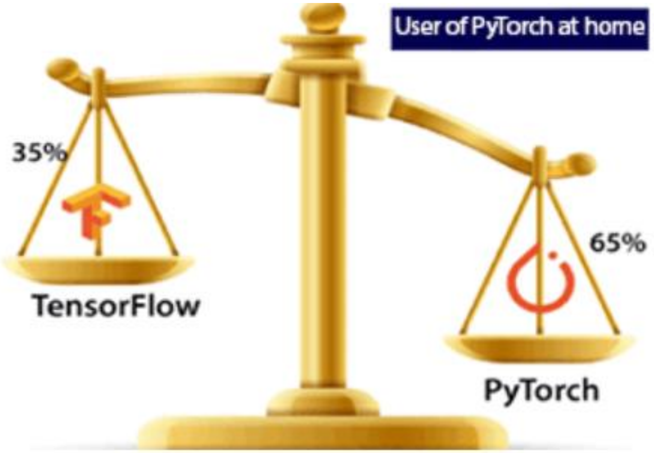
1. PyTorch

❖ PyTorch

- Python으로 기존 딥러닝 프레임워크 Torch를 구현한 것

PyTorch





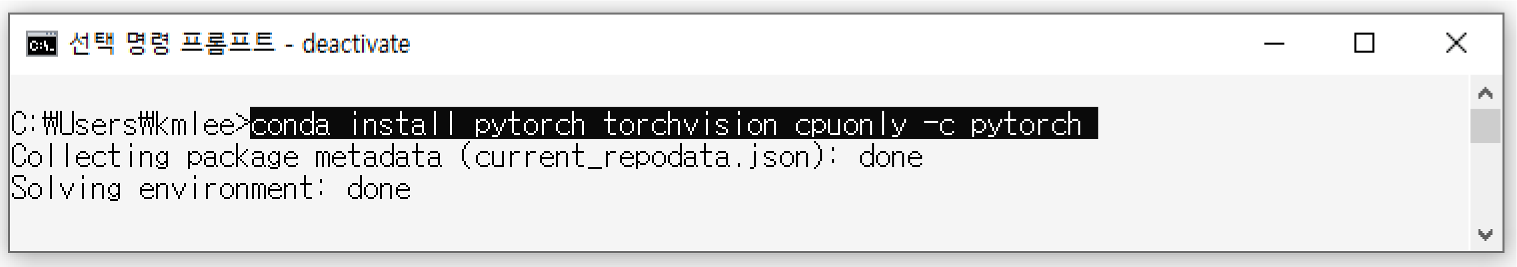
	Comparison Factors	Pass	Fail
1.	Features	TensorFlow	PyTorch
2.	Community	TensorFlow	PyTorch
3.	Level of API	TensorFlow	PyTorch
4.	Speed	PyTorch	TensorFlow
5.	Popularity	TensorFlow	PyTorch
6.	Ramp-Up Time	PyTorch	TensorFlow
7.	Coverage	TensorFlow	PyTorch
8.	Deployment	TensorFlow	PyTorch
9.	Serialization	TensorFlow	PyTorch
10.	Graph constructing and Debugging	PyTorch	TensorFlow
11.	Visualization	TensorFlow	PyTorch
12.	Architecture	PyTorch	TensorFlow
13.	Dataset	TensorFlow	PyTorch
14.	Documentation	PyTorch, TensorFlow	
15.	Device Management	TensorFlow	PyTorch
16.	Custom Extension	PyTorch	TensorFlow

https://www.javatpoint.com/pytorch-vs-tensorflow

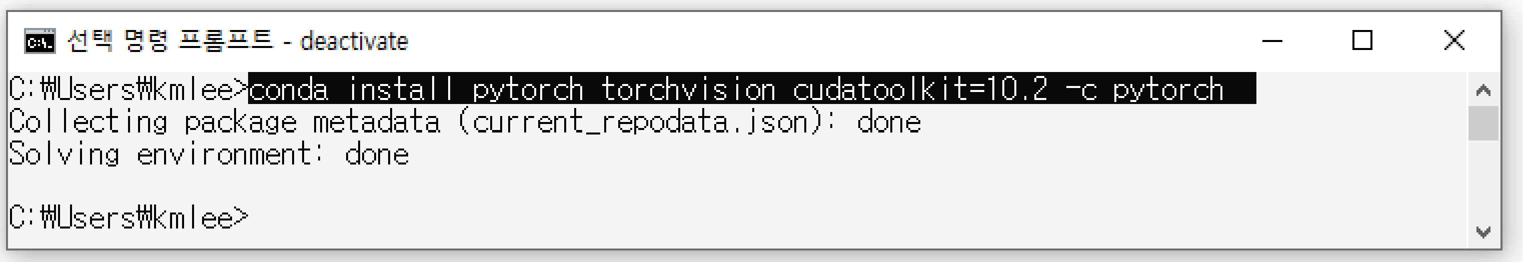
2. PyTorch 설치

❖ PyTorch 설치

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



```
C:₩Users₩kmlee>conda install pytorch torchvision cpuonly -c pytorch
Collecting package metadata (current_repodata.json): done
Solving environment: done
```



```
C:₩Users₩kmlee>conda install pytorch torchvision cudatoolkit=10.2 -c pytorch
Collecting package metadata (current_repodata.json): done
Solving environment: done

C:₩Users₩kmlee>
```

3. PyTorch 기초

❖ PyTorch 기초

- **torch.Tensor( )**
  - **NumPy 배열**로 된 데이터를 PyTorch에서 다를 수 있는 **텐서**로 변환
  - 단일 자료형으로 된 다차원 배열

Data type	**dtype**	**CPU tensor**	**GPU tensor**
32-bit floating point	torch.float32 or torch.float	torch.FloatTensor	torch.cuda.FloatTensor
64-bit floating point	torch.float64 or torch.double	torch.DoubleTensor	torch.cuda.DoubleTensor
16-bit floating point	torch.float16 or torch.half	torch.HalfTensor	torch.cuda.HalfTensor
8-bit integer (unsigned)	torch.uint8	torch.ByteTensor	torch.cuda.ByteTensor
8-bit integer (signed)	torch.int8	torch.CharTensor	torch.cuda.CharTensor
16-bit integer (signed)	torch.int16 or torch.short	torch.ShortTensor	torch.cuda.ShortTensor
32-bit integer (signed)	torch.int32 or torch.int	torch.IntTensor	torch.cuda.IntTensor
64-bit integer (signed)	torch.int64 or torch.long	torch.LongTensor	torch.cuda.LongTensor
Boolean	torch.bool	torch.BoolTensor	torch.cuda.BoolTensor

[실습] PyTorch의 텐서

```
import numpy as np
Import torch

A = torch.tensor([[1., -1.], [1., -1.]])
print('A = ', A)
B = torch.tensor(np.array([[1, 2, 3], [4, 5, 6]]))
print('B = ', B)

C = torch.rand(3,3)
print('C = ', C)

D = C.numpy()
print('D = ', D)

E = B.view(1,1,2,3)
print('E = ', E)

print('sum of A = ', A.sum())
print('mean of A = ', A.mean())
```

```
A =  tensor([[ 1., -1.],
        [ 1., -1.]])
B =  tensor([[1, 2, 3],
        [4, 5, 6]], dtype=torch.int32)

C =  tensor([[0.4306, 0.4923, 0.6163],
        [0.8168, 0.6739, 0.3506],
        [0.0116, 0.2050, 0.6086]])
D =  [[0.43059546 0.49226767
0.6162876 ]
 [0.8168291  0.6738524  0.3505581 ]
 [0.01155788 0.20499885 0.60861003]]
E =  tensor([[[[1, 2, 3],
          [4, 5, 6]]]], dtype=torch.int32)
sum of A =  tensor(0.)
mean of A =  tensor(0.)
```

PyTorch 기초

❖ PyTorch 기초 – cont.

- **torch.LongTensor**( )
  - **LongTensor**로 변환

- **TensorDataSet**( )
  - **배열 쌍**을 **대응**되는 **원소**끼리 **결합**하여 하나의 **데이터 집합 생성**
  - 예. (입력 데이터, 출력 레이블)

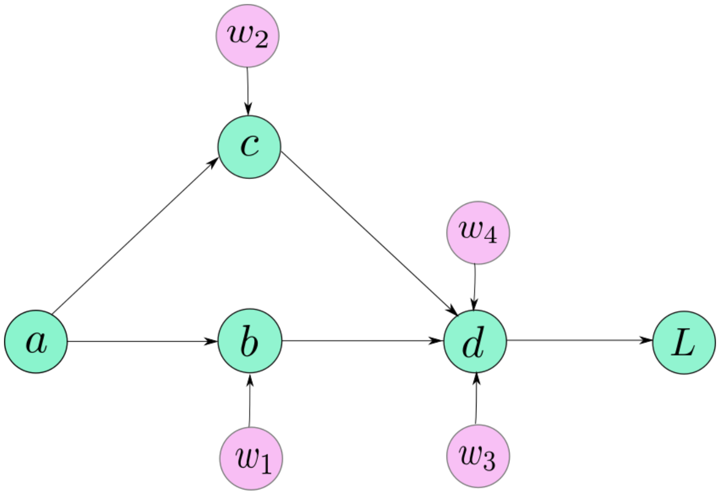
- **TensorLoader**(tensorDataset, batch_size=64, shuffle=True)
  - TensorDataSet 객체를 학습 및 추론에 사용하기 편리한 객체로 변환
    - batch_size : 신경망 가중치를 한번 수정할 때 사용하는 데이터 개수
    - suffle : 데이터 순서를 무작위로 섞을지 여부

PyTorch 기초

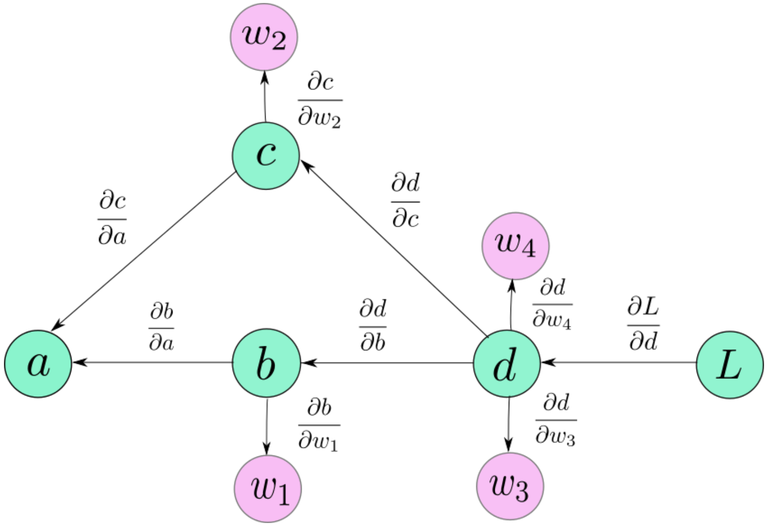
❖ **계산 그래프**(Computation graph)

- 연산 과정을 data flow로 나타낸 그래프 구조

```
b = w1 * a
c = w2 * a
d = (w3 * b) + (w4 * c)
L = f(d)
```



forward( )



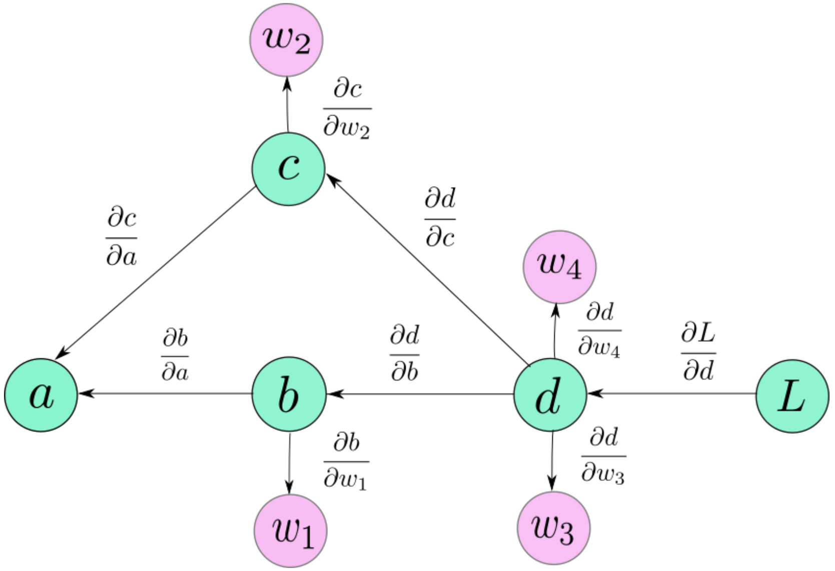
w_2
$\frac{\partial c}{\partial w_2}$
c
$\frac{\partial c}{\partial a}$
$\frac{\partial d}{\partial c}$
w_4
$\frac{\partial d}{\partial w_4}$
$\frac{\partial L}{\partial d}$
$\frac{\partial b}{\partial a}$
$\frac{\partial d}{\partial b}$
a
b
d
L
$\frac{\partial b}{\partial w_1}$
$\frac{\partial d}{\partial w_3}$
w_1
w_3

backward( )

https://towardsdatascience.com/getting-started-with-pytorch-part-1-understanding-how-automatic-differentiation-works-5008282073ec

PyTorch 기초

- ❖ **계산 그래프**(Computation graph) – cont.
  - gradient 계산
    - Computation graph를 이용한 chain rule 적용



w_2
$\frac{\partial c}{\partial w_2}$
c
$\frac{\partial c}{\partial a}$
$\frac{\partial d}{\partial c}$
w_4
$\frac{\partial d}{\partial w_4}$
$\frac{\partial L}{\partial d}$
$\frac{\partial b}{\partial a}$
$\frac{\partial d}{\partial b}$
a
b
d
L
$\frac{\partial b}{\partial w_1}$
$\frac{\partial d}{\partial w_3}$
w_1
w_3

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial d}\frac{\partial d}{\partial b}\frac{\partial b}{\partial a} + \frac{\partial L}{\partial d}\frac{\partial d}{\partial c}\frac{\partial c}{\partial a}$$

- autograd (automatic gradient)

https://towardsdatascience.com/getting-started-with-pytorch-part-1-understanding-how-automatic-differentiation-works-5008282073ec

[실습] PyTorch의 텐서

```
import torch
from torch.autograd import Variable
x = Variable(torch.tensor([[2.]]), requires_grad = True)
print('x = ', x)
print('x.data = ',  x.data)
print('x.grad = ', x.grad)
print('x.grad_fn() = ',  x.grad_fn)
y = x * x * 3
print('₩ny = ', y)
print('y.data = ',  y.data)
print('y.grad = ', y.grad)
print('y.grad_fn() = ',  y.grad_fn)
z = y**2
print('₩nz = ', z)
print('z.data = ', z.data)
print('z.grad = ', z.grad)
z.backward( )
print('₩nAfter invocation of backward()')
print('₩nx = ', x)
print('x.data = ',  x.data)
print('x.grad = ', x.grad)
print('x.grad_fn( ) = ',  x.grad_fn)
print('₩ny = ', y)
print('y.data = ',  y.data)
print('y.grad = ', y.grad)
print('y.grad_fn( ) = ',  y.grad_fn)
print('₩nz = ', z)
print('z.data = ', z.data)
print('z.grad = ', z.grad)
```

```
x =  tensor([[2.]], requires_grad=True)
x.data =  tensor([[2.]])
x.grad =  None
x.grad_fn() =  None

y =  tensor([[12.]], grad_fn=<MulBackward0>)
y.data =  tensor([[12.]])
y.grad =  None
y.grad_fn() =  <MulBackward0 object at 0x0000022A669C3508>

z =  tensor([[144.]], grad_fn=<PowBackward0>)
z.data =  tensor([[144.]])
z.grad =  None

After invocation of backward()

x =  tensor([[2.]], requires_grad=True)
x.data =  tensor([[2.]])
x.grad =  tensor([[288.]])
x.grad_fn() =  None

y =  tensor([[12.]], grad_fn=<MulBackward0>)
y.data =  tensor([[12.]])
y.grad =  None
y.grad_fn() =  <MulBackward0 object at 0x0000022A669BB188>

z =  tensor([[144.]], grad_fn=<PowBackward0>)
z.data =  tensor([[144.]])
z.grad =  None
```

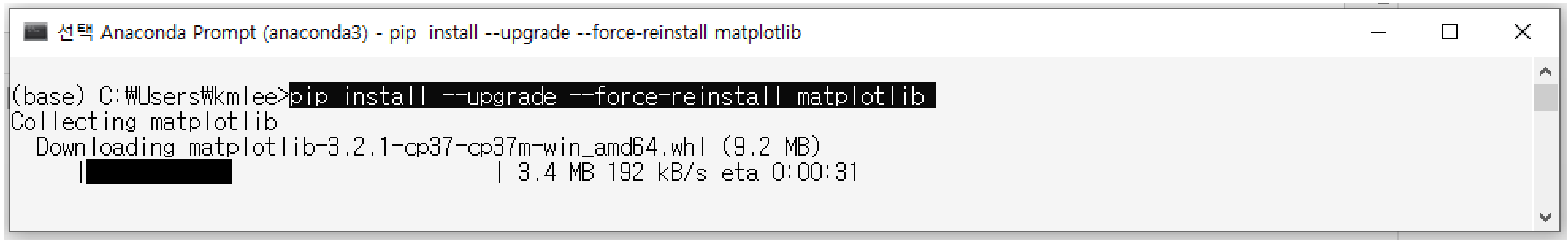
PyTorch

❖ PyTorch 기초 – cont.

- model.train( )
  - 신경망 모델을 **학습 모드**로 전환
- model.eval( )
  - 신경망 모델을 **추론 모델**로 전환
- optimizer.zero_grad( )
  - **역전파 오차(그레디언트)** 계산의 **초기화**
- with.torch.no_grad( )
  - **추론 과정**에서는 **그레디언트** 계산 **불필요**

❖ Anaconda에 설치된 패키지와 윈도우 설치 패키지 충돌시

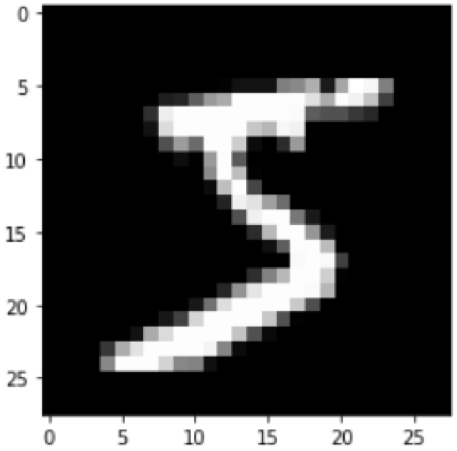
- Anaconda 환경에서 재설치


선택 Anaconda Prompt (anaconda3) - pip install --upgrade --force-reinstall matplotlib

```
(base) C:₩Users₩kmlee>pip install --upgrade --force-reinstall matplotlib
Collecting matplotlib
  Downloading matplotlib-3.2.1-cp37-cp37m-win_amd64.whl (9.2 MB)
     |███████                   | 3.4 MB 192 kB/s eta 0:00:31
```

[실습] PyTorch의 MLP 프로그래밍

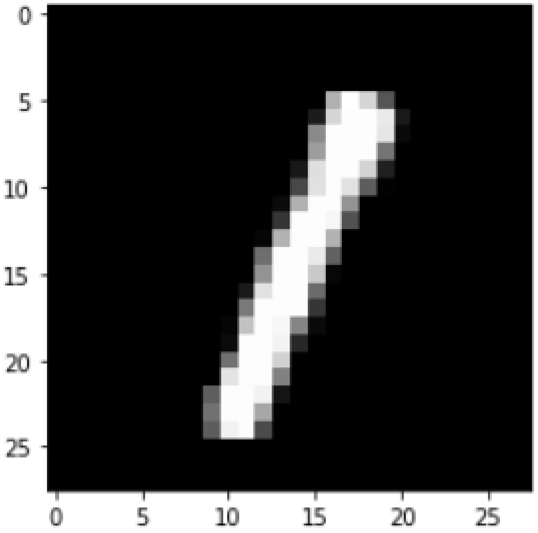
```
#-*- coding: utf-8 -*-

from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, cache=True)

X = mnist.data/255
y = mnist.target

import matplotlib.pyplot as plt
plt.imshow(X[0].reshape(28,28), cmap='gray')
plt.show( )
print("이미지 레이블 : { }".format(y[0]))

import torch
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=1/7, random_state=0)
X_train = torch.Tensor(X_train)
X_test = torch.Tensor(X_test)
y_train = torch.LongTensor(list(map(int, y_train)))
y_test = torch.LongTensor(list(map(int, y_test)))

ds_train = TensorDataset(X_train, y_train)
ds_test = TensorDataset(X_test, y_test)

loader_train = DataLoader(ds_train, batch_size=64, shuffle=True)
loader_test = DataLoader(ds_test, batch_size=64, shuffle=False)
```



```
이미지 레이블 : 5
```

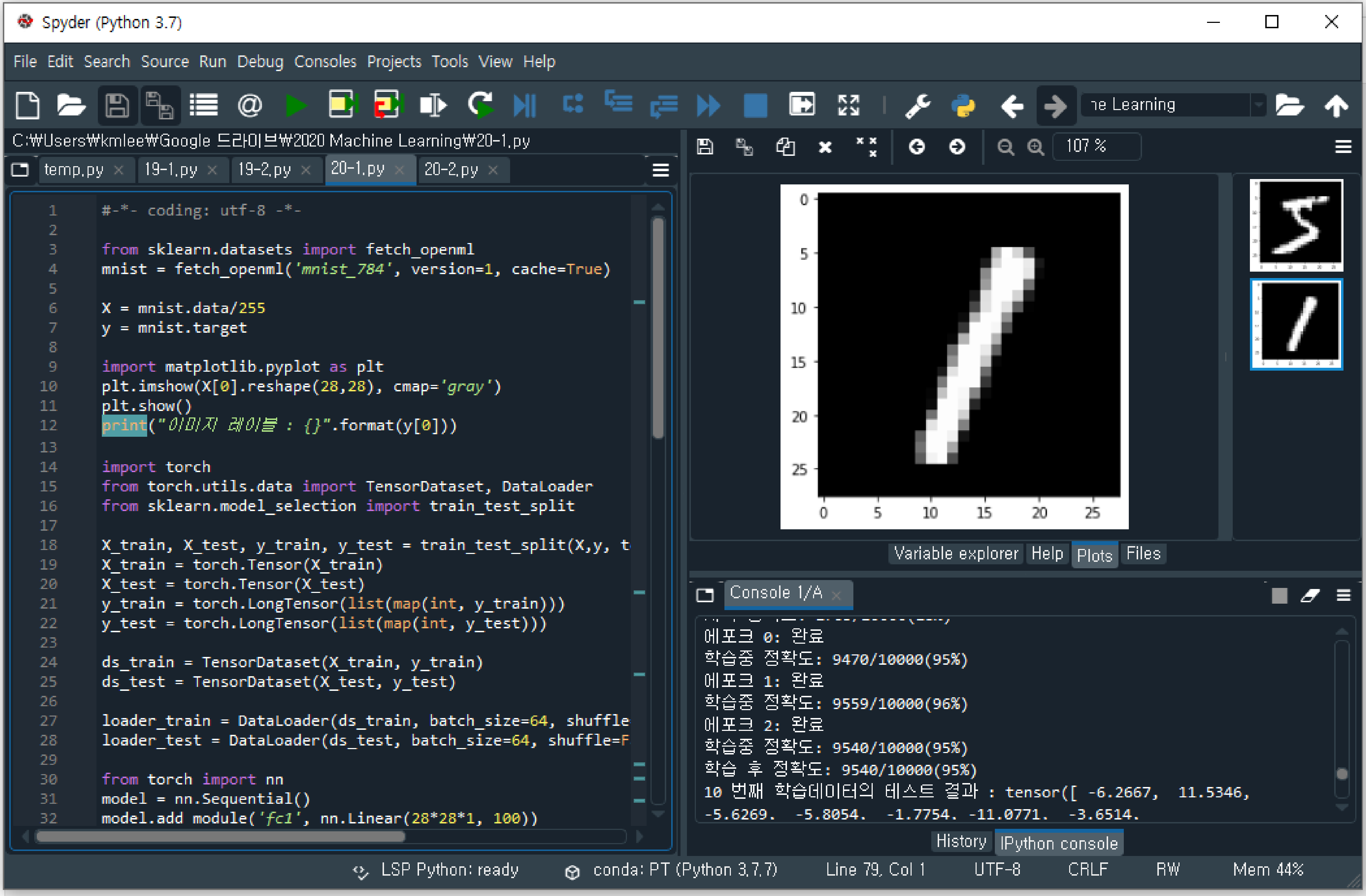
```
from torch import nn
model = nn.Sequential( )
model.add_module('fc1', nn.Linear(28*28*1, 100))        # 모델 구성
model.add_module('relu1', nn.ReLU())
model.add_module('fc2', nn.Linear(100,100))
model.add_module('relu2', nn.ReLU())
model.add_module('fc3', nn.Linear(100,10))

from torch import optim
loss_fn = nn.CrossEntropyLoss( )   # 손실 함수
optimizer = optim.Adam(model.parameters( ), lr=0.01)

def train(epoch):
    model.train( )  # 학습 모드로 변환
    for data, targets in loader_train:
        optimizer.zero_grad( )  # 그레디언트 초기화
        outputs = model(data)
        loss = loss_fn(outputs, targets)
        loss.backward( )
        optimizer.step( )
    print('에포크 {}: 완료'.format(epoch))

def test(head):
    model.eval( )   # 테스트 모드로 변환
    correct = 0
    with torch.no_grad( ):
        for data, targets in loader_test:
            outputs = model(data)
            _, predicted = torch.max(outputs.data, 1)
            correct += predicted.eq(targets.data.view_as(predicted)).sum()
    data_num = len(loader_test.dataset)
    print('{ } 정확도: { }/{ }({:.0f}%)'.format(head, correct, data_num, 100.*correct/data_num))
```

```
test('시작')
for epoch in range(3):
    train(epoch)
    test('학습중')
test('학습 후')

index = 10  # 테스트 데이터 중에서 확인해볼 데이터의 인덱스
model.eval( )  # 모델 테스트 모드로 전환
data = X_test[index]
output = model(data)   # 모델 적용
print('{ } 번째 학습데이터의 테스트 결과 : { }'.format(index,output))
_, predicted = torch.max(output.data, 0)
print('{ }번째 데이터의 예측 : { }'.format(index, predicted))
X_test_show = (X_test[index]).numpy()
plt.imshow(X_test_show.reshape(28,28), cmap='gray')
print( ' 실제 레이블: { }'.format(y_test[index]))
```



```
시작 정확도: 796/10000(8%)
에포크 0: 완료
학습중 정확도: 9429/10000(94%)
에포크 1: 완료
학습중 정확도: 9514/10000(95%)
에포크 2: 완료
학습중 정확도: 9589/10000(96%)
학습 후 정확도: 9589/10000(96%)
10 번째 학습데이터의 테스트 결과 : tensor([-18.3571,  22.7998, -12.3894, -21.2029,  -4.9429, -20.4559, -11.2541,
6.2497,  -1.3856, -11.9634], grad_fn=<AddBackward0>)
10번째 데이터의 예측 : 1
실제 레이블 : 1
```

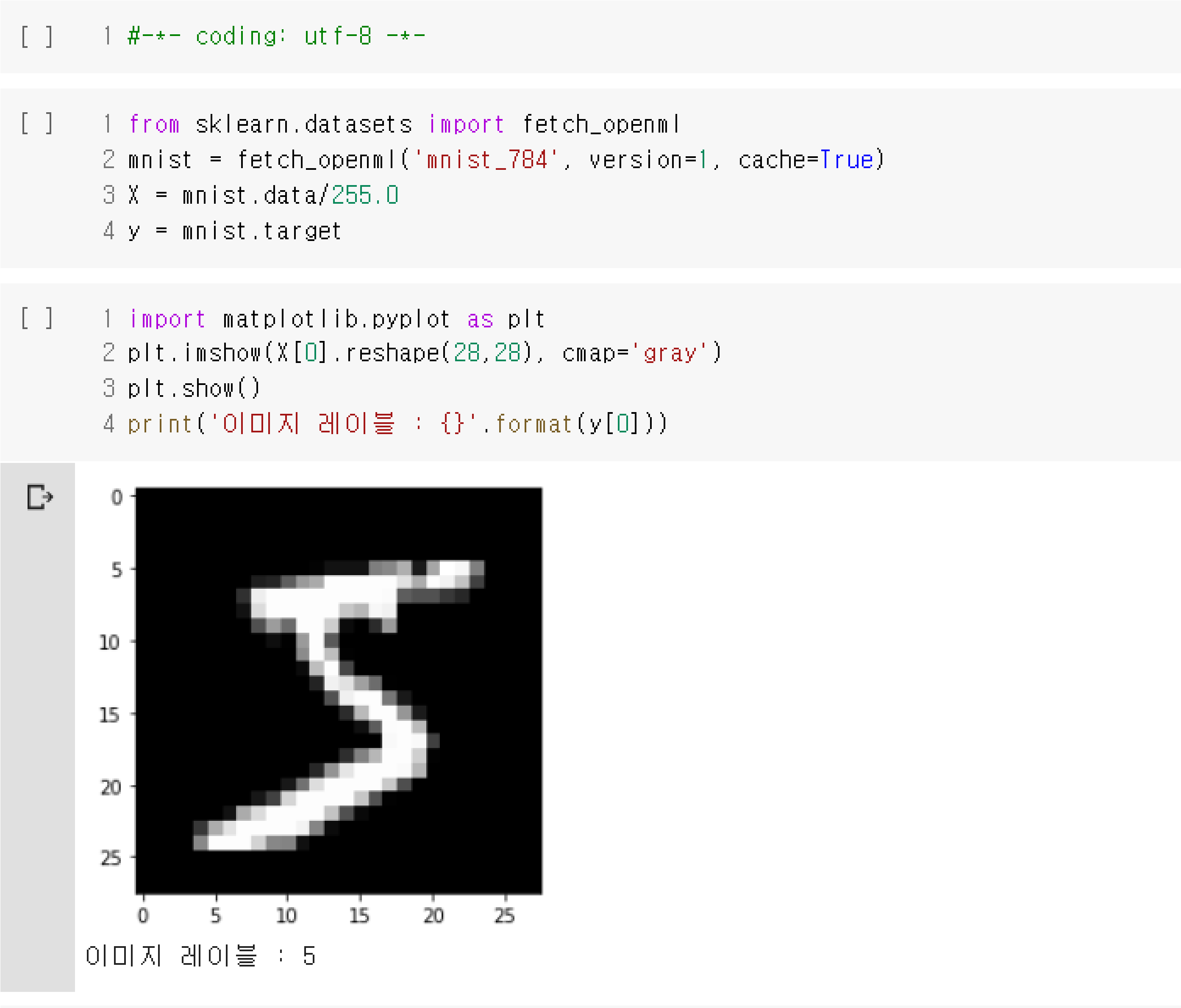
[실습] Spyder에서 PyTorch 실행



```python
#-*- coding: utf-8 -*-

from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, cache=True)

X = mnist.data/255
y = mnist.target

import matplotlib.pyplot as plt
plt.imshow(X[0].reshape(28,28), cmap='gray')
plt.show()
print("이미지 레이블 : {}".format(y[0]))

import torch
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, t
X_train = torch.Tensor(X_train)
X_test = torch.Tensor(X_test)
y_train = torch.LongTensor(list(map(int, y_train)))
y_test = torch.LongTensor(list(map(int, y_test)))

ds_train = TensorDataset(X_train, y_train)
ds_test = TensorDataset(X_test, y_test)

loader_train = DataLoader(ds_train, batch_size=64, shuffle
loader_test = DataLoader(ds_test, batch_size=64, shuffle=F

from torch import nn
model = nn.Sequential()
model.add_module('fc1', nn.Linear(28*28*1, 100))
```

```
에포크 0: 완료
학습중 정확도: 9470/10000(95%)
에포크 1: 완료
학습중 정확도: 9559/10000(96%)
에포크 2: 완료
학습중 정확도: 9540/10000(95%)
학습 후 정확도: 9540/10000(95%)
10 번째 학습데이터의 테스트 결과 : tensor([ -6.2667,  11.5346,
-5.6269.  -5.8054.  -1.7754. -11.0771.  -3.6514.
```

[실습] Colab에서 PyTorch 실행

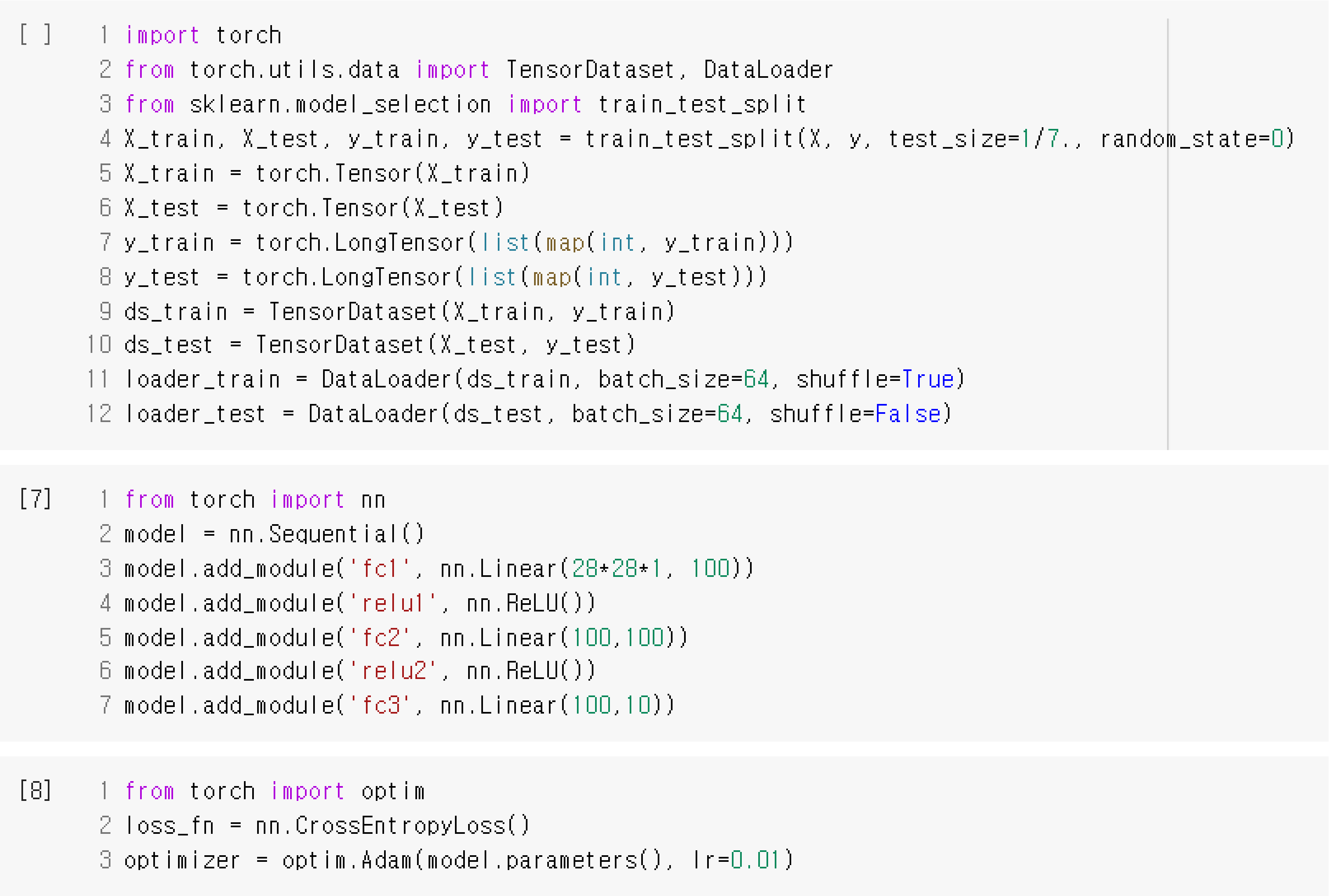


```python
#-*- coding: utf-8 -*-
```

```python
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, cache=True)
X = mnist.data/255.0
y = mnist.target
```

```python
import matplotlib.pyplot as plt
plt.imshow(X[0].reshape(28,28), cmap='gray')
plt.show()
print('이미지 레이블 : {}'.format(y[0]))
```

```
이미지 레이블 : 5
```



[ ]

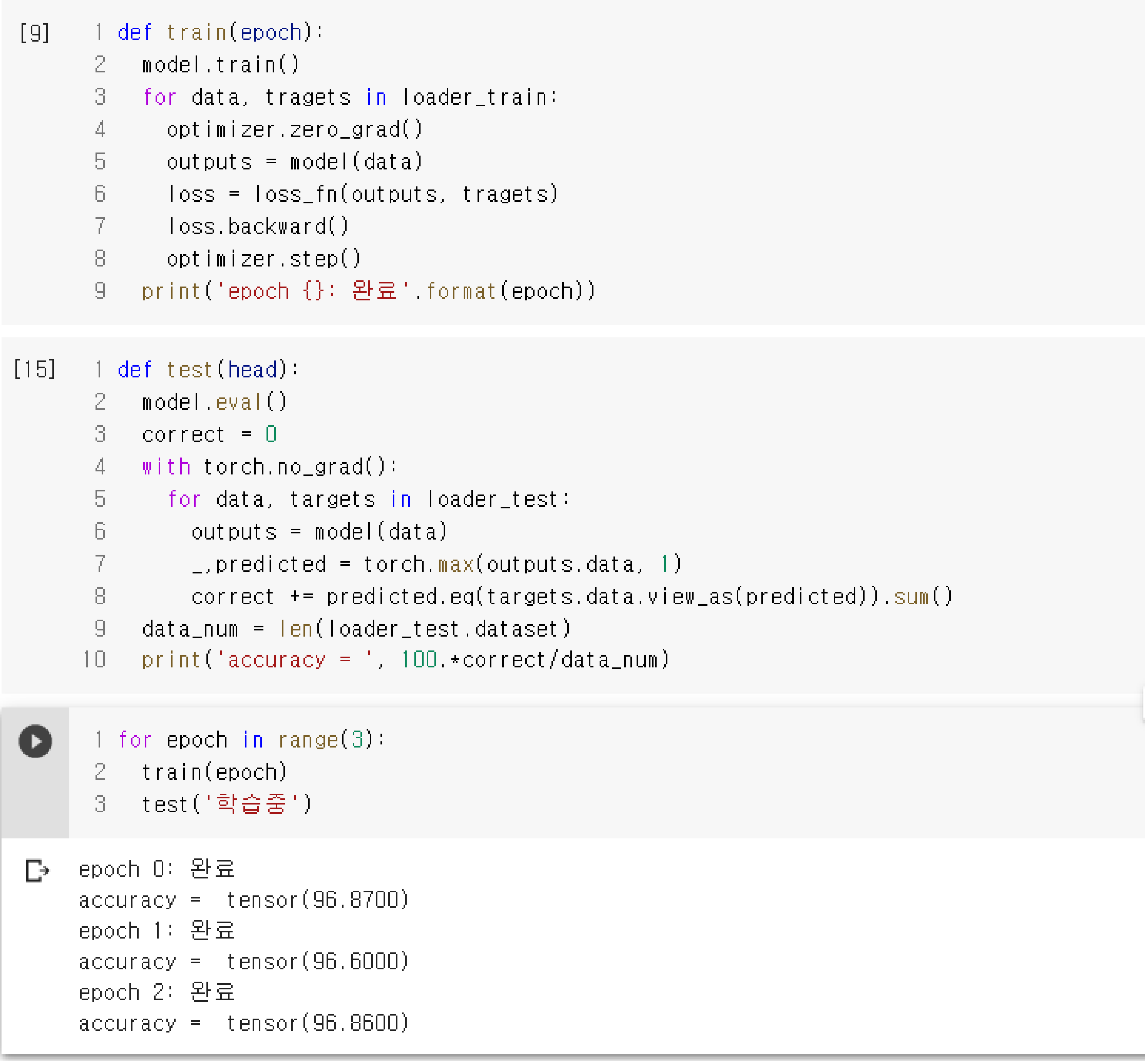
```python
import torch
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/7., random_state=0)
X_train = torch.Tensor(X_train)
X_test = torch.Tensor(X_test)
y_train = torch.LongTensor(list(map(int, y_train)))
y_test = torch.LongTensor(list(map(int, y_test)))
ds_train = TensorDataset(X_train, y_train)
ds_test = TensorDataset(X_test, y_test)
loader_train = DataLoader(ds_train, batch_size=64, shuffle=True)
loader_test = DataLoader(ds_test, batch_size=64, shuffle=False)
```

[7]

```python
from torch import nn
model = nn.Sequential()
model.add_module('fc1', nn.Linear(28*28*1, 100))
model.add_module('relu1', nn.ReLU())
model.add_module('fc2', nn.Linear(100,100))
model.add_module('relu2', nn.ReLU())
model.add_module('fc3', nn.Linear(100,10))
```

[8]

```python
from torch import optim
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```



[9]

```python
def train(epoch):
  model.train()
  for data, tragets in loader_train:
    optimizer.zero_grad()
    outputs = model(data)
    loss = loss_fn(outputs, tragets)
    loss.backward()
    optimizer.step()
  print('epoch {}: 완료'.format(epoch))
```

[15]

```python
def test(head):
  model.eval()
  correct = 0
  with torch.no_grad():
    for data, targets in loader_test:
      outputs = model(data)
      _,predicted = torch.max(outputs.data, 1)
      correct += predicted.eq(targets.data.view_as(predicted)).sum()
  data_num = len(loader_test.dataset)
  print('accuracy = ', 100.*correct/data_num)
```

```python
for epoch in range(3):
  train(epoch)
  test('학습중')
```

```
epoch 0: 완료
accuracy =  tensor(96.8700)
epoch 1: 완료
accuracy =  tensor(96.6000)
epoch 2: 완료
accuracy =  tensor(96.8600)
```

[실습] CNN 모델을 이용한 MNIST 데이터 분류

```
#-*- coding: utf-8 -*-
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, cache=True)
X = mnist.data
y = mnist.target

import torch
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/7, random_state=0)
X_train = torch.Tensor(X_train)
X_test = torch.Tensor(X_test)
y_train = torch.LongTensor(list(map(int, y_train)))
y_test = torch.LongTensor(list(map(int, y_test)))

import torch.nn as nn
import torch.nn.functional as F
from torch import optim
from torch.autograd import Variable

X_train = X_train.view(-1, 1,28,28).float()
X_test = X_test.view(-1,1,28,28).float()
print(X_train.shape)
print(X_test.shape)

train = TensorDataset(X_train, y_train)
test = TensorDataset(X_test, y_test)
BATCH_SIZE = 32
loader_train = DataLoader(train, batch_size = BATCH_SIZE, shuffle = False)
loader_test = DataLoader(test, batch_size = BATCH_SIZE, shuffle = False)
```

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__( )
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        self.conv2 = nn.Conv2d(32, 32, kernel_size=5)
        self.conv3 = nn.Conv2d(32,64, kernel_size=5)
        self.fc1 = nn.Linear(3*3*64, 256)
        self.fc2 = nn.Linear(256, 10)

        self.loss_fn = nn.CrossEntropyLoss( )
        self.optimizer = optim.Adam(self.parameters( ), lr=0.01)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(F.max_pool2d(self.conv2(x), 2))
        x = F.dropout(x, p=0.5, training=self.training)
        x = F.relu(F.max_pool2d(self.conv3(x),2))
        x = F.dropout(x, p=0.5, training=self.training)
        x = x.view(-1,3*3*64 )
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size,
           stride=1, padding=0, dilation=1, groups=1,
           bias=True, padding_mode='zeros')
```

```
def fit(model, loader_train):
    optimizer = torch.optim.Adam(model.parameters( ))
    error = nn.CrossEntropyLoss( )
    EPOCHS = 1
    model.train( )
    for epoch in range(EPOCHS):
        correct = 0
        for batch_idx, (X_batch, y_batch) in enumerate(loader_train):
            var_X_batch = Variable(X_batch).float( )
            var_y_batch = Variable(y_batch)
            optimizer.zero_grad( )
            output = model(var_X_batch)
            loss = error(output, var_y_batch)
            loss.backward( )
            optimizer.step( )
            predicted = torch.max(output.data, 1)[1]
            correct += (predicted == var_y_batch).sum( )
            if batch_idx % 50 == 0:
                print('에포크 : {} [{}/{} ({:.0f}%)]₩t 손실함수 : {:.6f}₩t Accuracy:{:.3f}%'.format(
                    epoch, batch_idx*len(X_batch), len(loader_train),
                    100.*batch_idx / len(loader_train),
                    loss.data,
                    correct*100./ (BATCH_SIZE*(batch_idx+1))))
```

```
def evaluate(model):
    correct = 0
    for test_imgs, test_labels in loader_test:
        test_imgs = Variable(test_imgs).float()
        output = model(test_imgs)
        predicted = torch.max(output,1)[1]
        correct += (predicted == test_labels).sum()
    print("테스트 데이터 정확도: {:.3f}% ".format( float(correct) /
(len(loader_test)*BATCH_SIZE)))

cnn = CNN( )
evaluate(cnn)
fit(cnn, loader_train)
cnn.eval( )  # 모델 테스트 모드로 전환
evaluate(cnn)

index = 10  # 테스트 데이터 중에서 확인해볼 데이터의 인덱스
data = X_test[index].view(-1, 1,28,28).float( )
output = cnn(data)   # 모델 적용
print('{} 번째 학습데이터의 테스트 결과 : {}'.format(index, output))
_, predicted = torch.max(output, 1)
print('{}번째 데이터의 예측 : {}'.format(index, predicted.numpy()))
print('실제 레이블 : {}'.format(y_test[index]))
```

```
테스트 데이터 정확도: 0.101%
에포크 : 0 [0/1875 (0%)]     손실함수 : 16.765696        Accuracy:6.250%
에포크 : 0 [1600/1875 (3%)]                손실함수 : 1.837372         Accuracy:21.691%
                                          :
에포크 : 0 [59200/1875 (99%)]              손실함수 : 0.256999         Accuracy:86.894%
테스트 데이터 정확도: 0.930%
10 번째 학습데이터의 테스트 결과 : tensor([[-9.7553e+00, -1.5448e-03, -9.4535e+00, -9.9060e+00, -8.7322e+00,
-8.8163e+00, -1.0269e+01, -7.7631e+00, -7.7663e+00, -8.7147e+00]], grad_fn=<LogSoftmaxBackward>)
10번째 데이터의 예측 : [1]
실제 레이블 : 1
```

실습

1. PyTorch 환경을 구성한다.
2. 실습 프로그래밍 예제를 PyTorch와 Colab 환경에서 직접 실행해 본다.