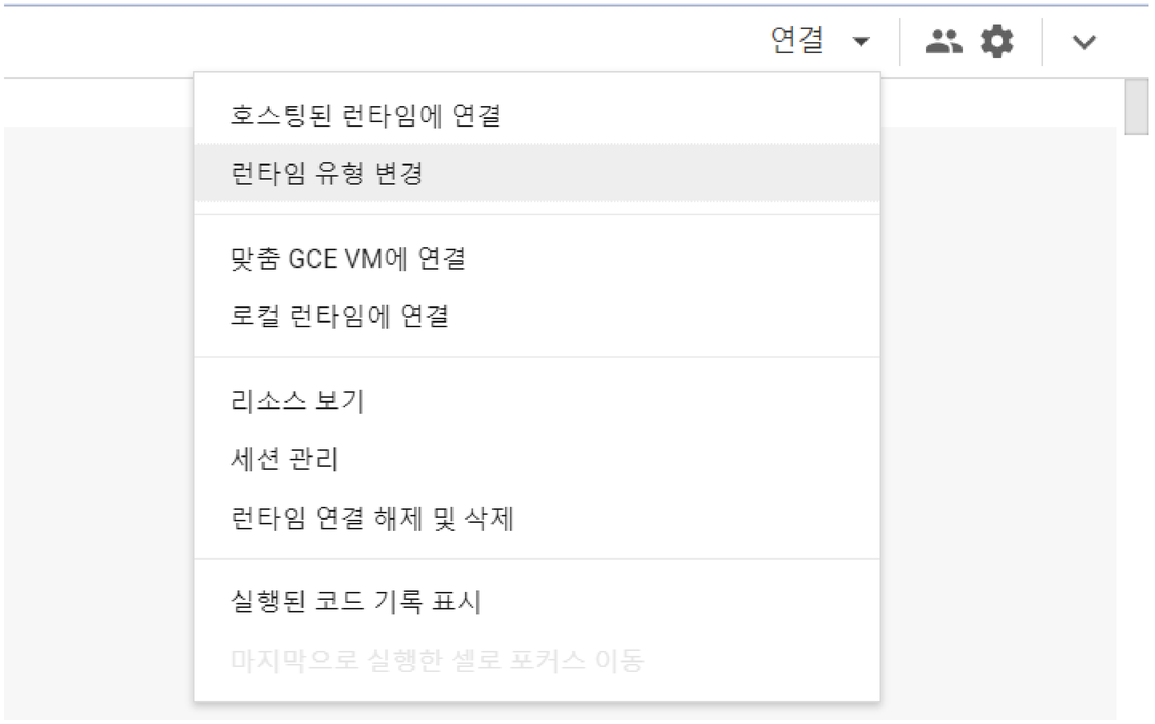
실습4 PyTorch

딥러닝 라이브러리

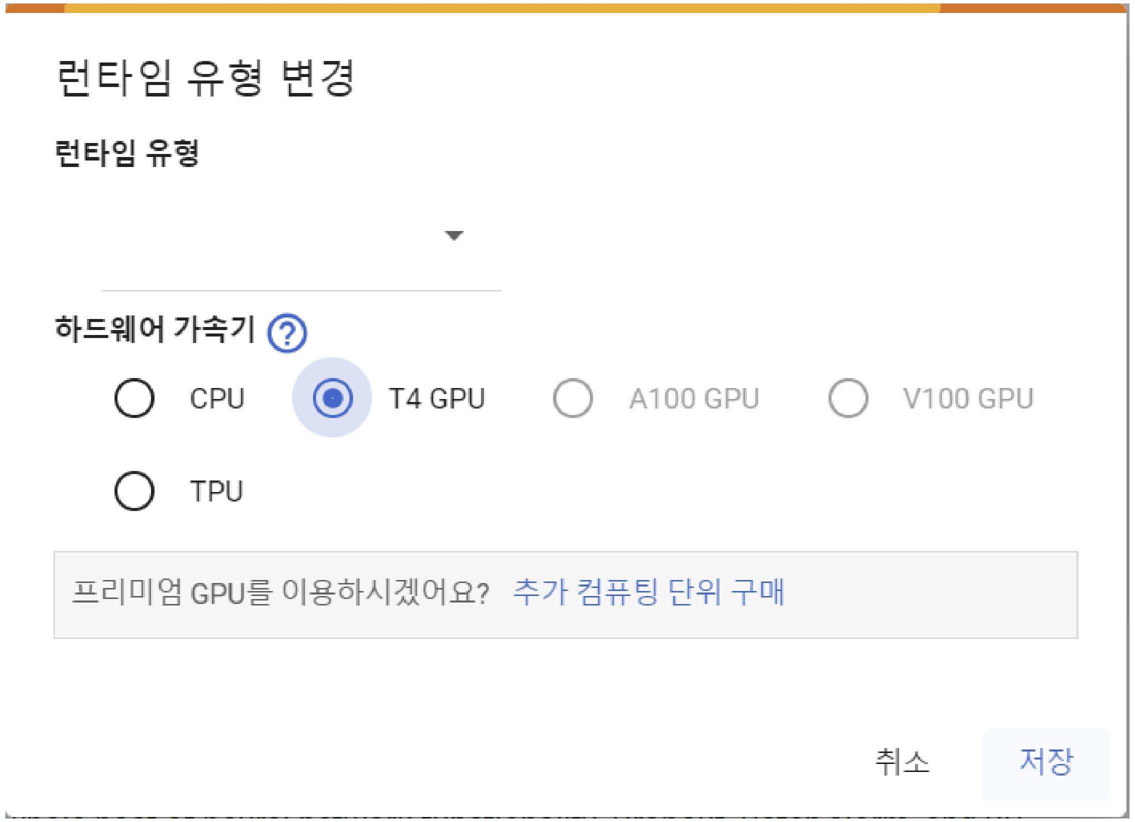
- PyTorch, TensorFlow
- 딥러닝 라이브러리 사용의 장점
  - GPU에서의 실행이 쉬움. CUDA 코드를 직접 작성할 필요 없음
  - 다양한 기능들이 high level로 구현됨

GPU 사용법

- PyTorch.ipynb 노트북실행
- 딥러닝에는 CPU 대신 GPU를 사용하는 것이 훨씬 빠름
- GPU의 많은 코어를 사용하여 병렬 처리


연결

호스팅된 런타임에 연결
런타임 유형 변경
맞춤 GCE VM에 연결
로컬 런타임에 연결
리소스 보기
세션 관리
런타임 연결 해제 및 삭제
실행된 코드 기록 표시
마지막으로 실행한 셀로 포커스 이동


런타임 유형 변경

런타임 유형

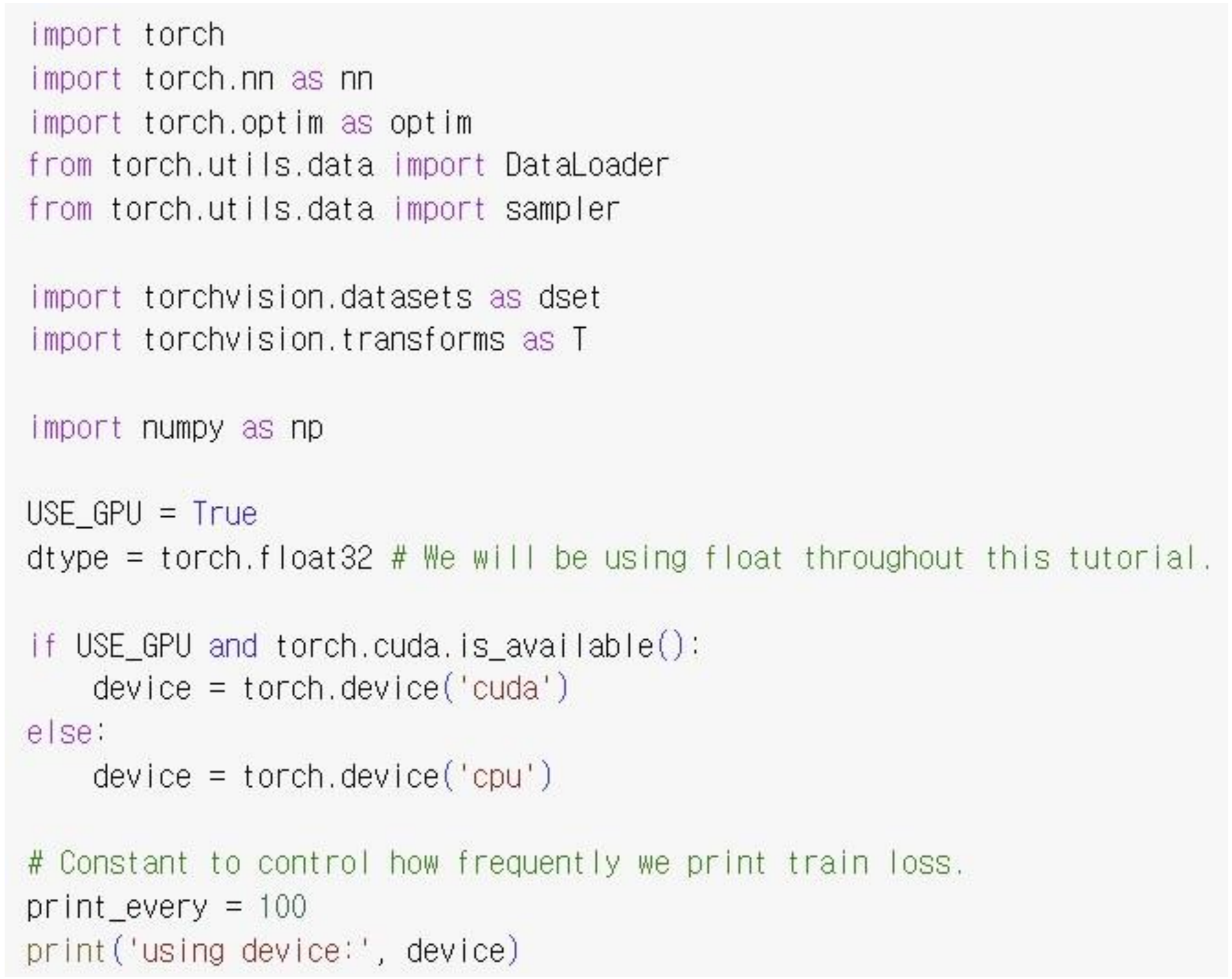
하드웨어 가속기

CPU [✓ T4 GPU] A100 GPU V100 GPU TPU

프리미엄 GPU를 이용하시겠어요? 추가 컴퓨팅 단위 구매

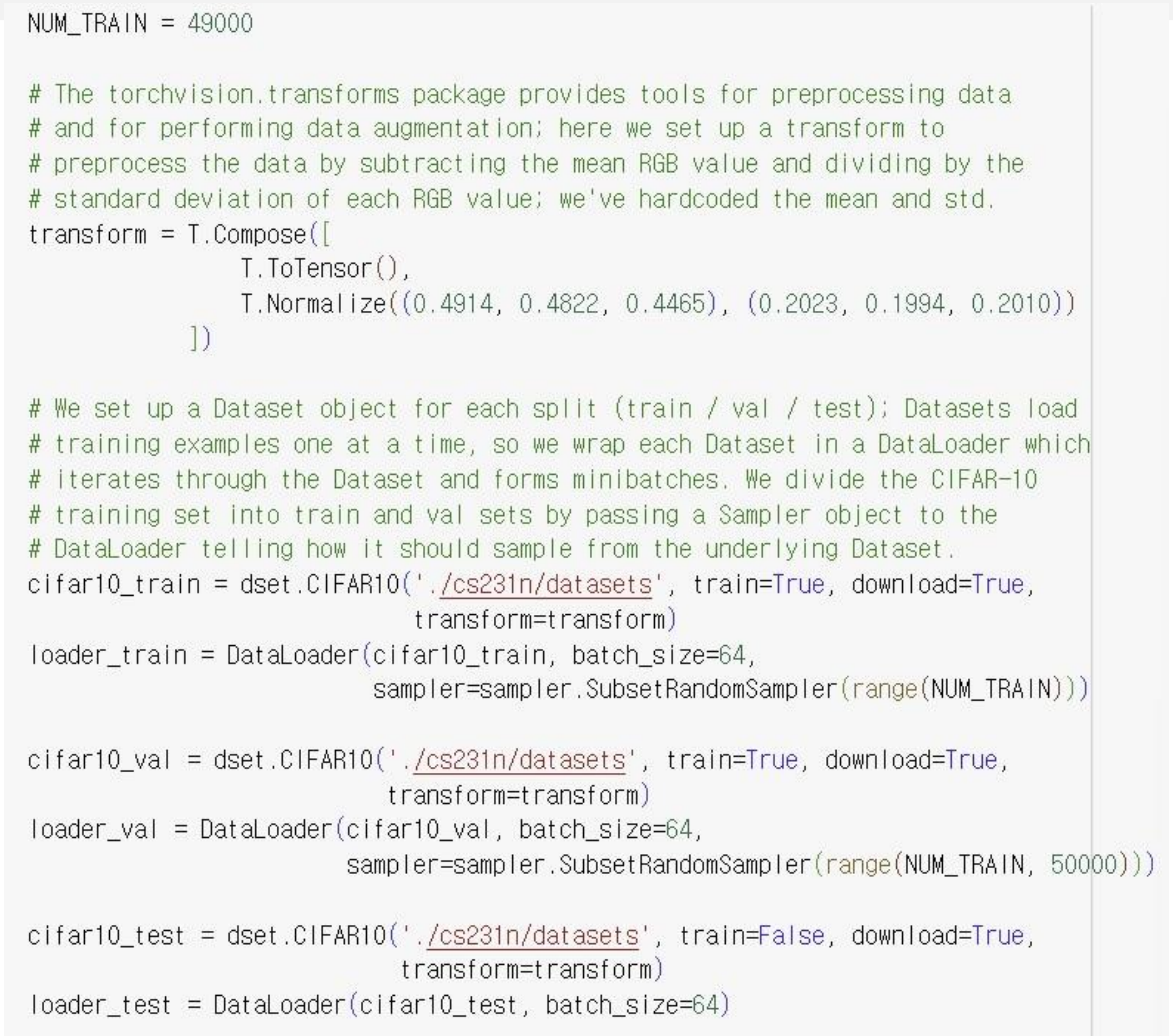
취소 저장

GPU 사용법



```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torch.utils.data import sampler

import torchvision.datasets as dset
import torchvision.transforms as T

import numpy as np

USE_GPU = True
dtype = torch.float32 # We will be using float throughout this tutorial.

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

# Constant to control how frequently we print train loss.
print_every = 100
print('using device:', device)
```

CIFAR-10 로드



```
NUM_TRAIN = 49000

# The torchvision.transforms package provides tools for preprocessing data
# and for performing data augmentation; here we set up a transform to
# preprocess the data by subtracting the mean RGB value and dividing by the
# standard deviation of each RGB value; we've hardcoded the mean and std.
transform = T.Compose([
                T.ToTensor(),
                T.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
            ])

# We set up a Dataset object for each split (train / val / test); Datasets load
# training examples one at a time, so we wrap each Dataset in a DataLoader which
# iterates through the Dataset and forms minibatches. We divide the CIFAR-10
# training set into train and val sets by passing a Sampler object to the
# DataLoader telling how it should sample from the underlying Dataset.
cifar10_train = dset.CIFAR10('./cs231n/datasets', train=True, download=True,
                             transform=transform)
loader_train = DataLoader(cifar10_train, batch_size=64,
                          sampler=sampler.SubsetRandomSampler(range(NUM_TRAIN)))

cifar10_val = dset.CIFAR10('./cs231n/datasets', train=True, download=True,
                           transform=transform)
loader_val = DataLoader(cifar10_val, batch_size=64,
                        sampler=sampler.SubsetRandomSampler(range(NUM_TRAIN, 50000)))

cifar10_test = dset.CIFAR10('./cs231n/datasets', train=False, download=True,
                            transform=transform)
loader_test = DataLoader(cifar10_test, batch_size=64)
```

추상화 수준

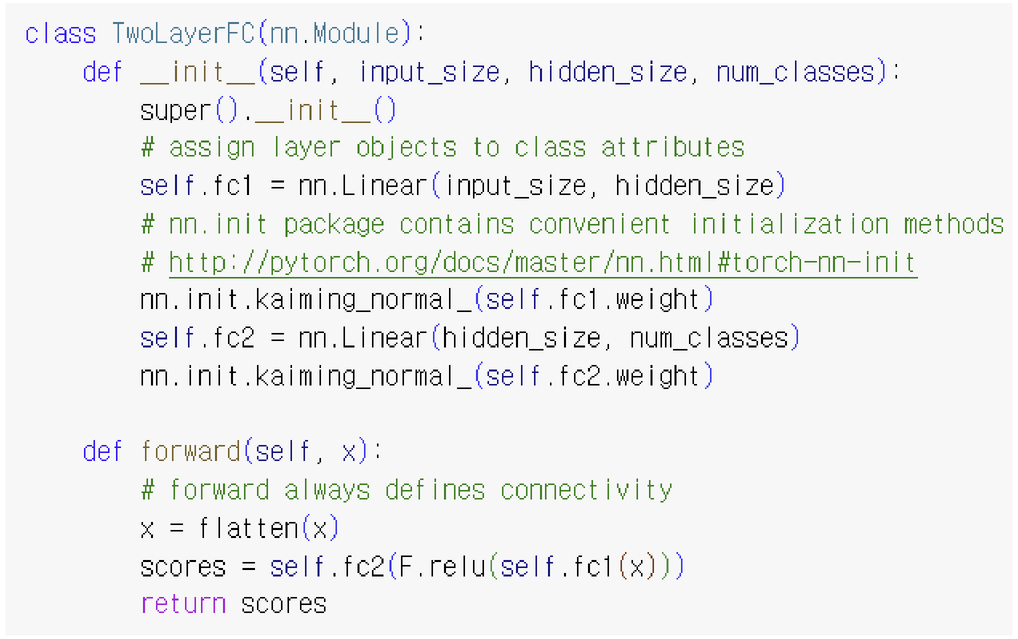
- PyTorch는 다양한 추상화 수준의 API를 제공
  - low-level API: 직접 세세한 것까지 일일이 신경쓰며 구현하는 경우
  - high-level API: 이미 잘 구현된 함수 등을 단순히 쉽게 활용하는 경우
- 3가지 정도의 추상화 수준으로 나뉨
  - Barebones Pytorch
  - nn.Module API
  - nn.Sequential API

추상화 수준

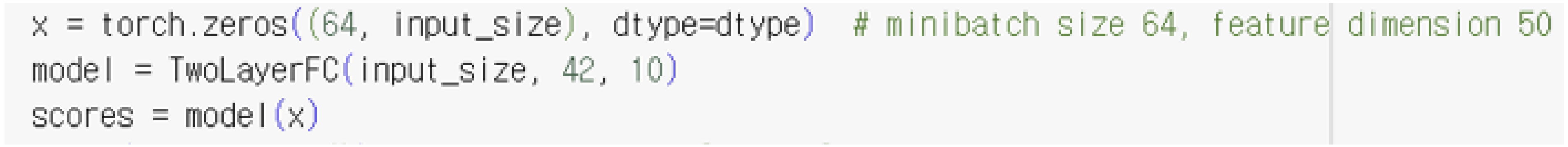
- 추상화 수준 장단점 비교
  - low-level: 사용하기 불편하지만 자유도가 높음
  - high-level: 사용하기 쉽고 편리하지만 자유도가 비교적 떨어짐
- 각 응용에 따라 필요한 추상화 수준 API를 선택해서 사용
- 본 수업에서는 nn.Module API 및 nn.Sequential API를 설명
  - 과제는 기본적으로 nn.Sequential API를 사용

Module API 사용법

- nn.Module API는 여러 편리한 기능을 제공
  – 예) RMSProp, Adagrad 등의 파라미터 업데이트 방법들을 구현하는 torch.optim 패키지 제공
- 다음과 같이 사용
  – 1. nn.Module을 서브클래스화
  – 2. 생성자 init()에서 필요한 모든 계층을 클래스 속성으로 정의
  – 3. forward() 메서드에서 신경망의 연결성 정의

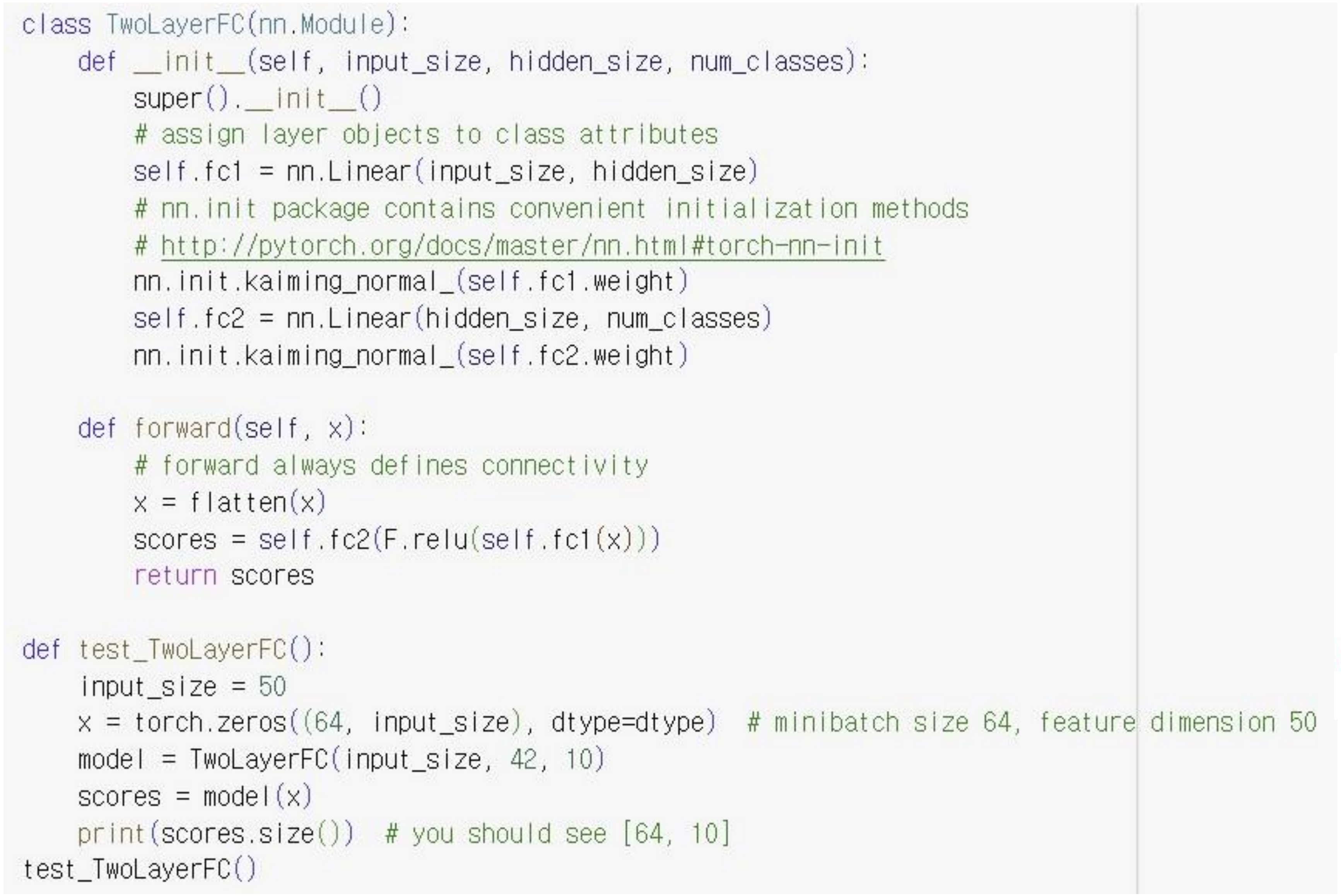


```
class TwoLayerFC(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super().__init__()
        # assign layer objects to class attributes
        self.fc1 = nn.Linear(input_size, hidden_size)
        # nn.init package contains convenient initialization methods
        # http://pytorch.org/docs/master/nn.html#torch-nn-init
        nn.init.kaiming_normal_(self.fc1.weight)
        self.fc2 = nn.Linear(hidden_size, num_classes)
        nn.init.kaiming_normal_(self.fc2.weight)

    def forward(self, x):
        # forward always defines connectivity
        x = flatten(x)
        scores = self.fc2(F.relu(self.fc1(x)))
        return scores
```

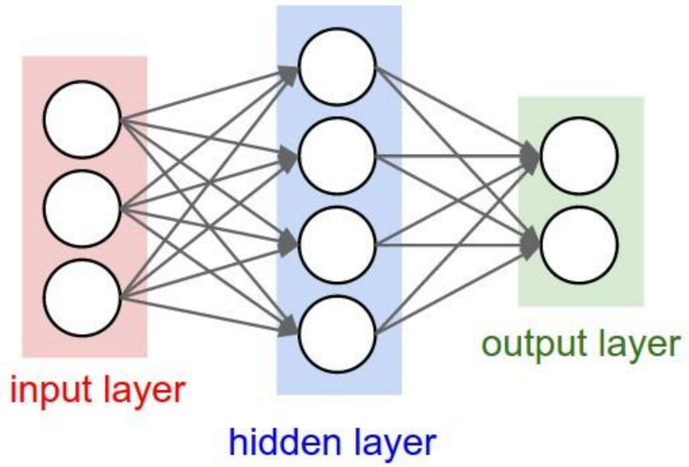


```
x = torch.zeros((64, input_size), dtype=dtype)  # minibatch size 64, feature dimension 50
model = TwoLayerFC(input_size, 42, 10)
scores = model(x)
```

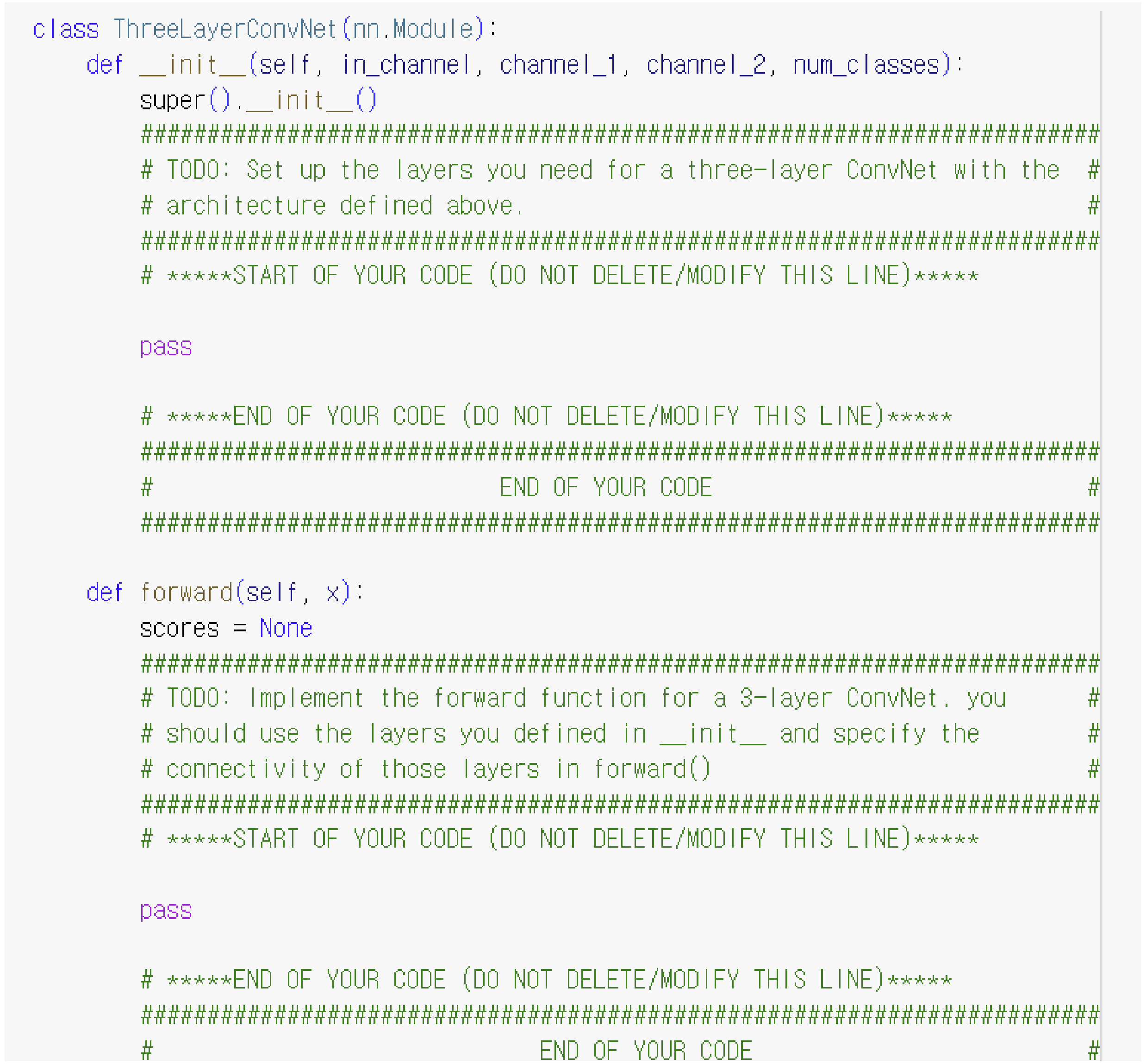
Module API 사용법: 완전연결 신경망 예



```python
class TwoLayerFC(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super().__init__()
        # assign layer objects to class attributes
        self.fc1 = nn.Linear(input_size, hidden_size)
        # nn.init package contains convenient initialization methods
        # http://pytorch.org/docs/master/nn.html#torch-nn-init
        nn.init.kaiming_normal_(self.fc1.weight)
        self.fc2 = nn.Linear(hidden_size, num_classes)
        nn.init.kaiming_normal_(self.fc2.weight)

    def forward(self, x):
        # forward always defines connectivity
        x = flatten(x)
        scores = self.fc2(F.relu(self.fc1(x)))
        return scores

def test_TwoLayerFC():
    input_size = 50
    x = torch.zeros((64, input_size), dtype=dtype)  # minibatch size 64, feature dimension 50
    model = TwoLayerFC(input_size, 42, 10)
    scores = model(x)
    print(scores.size())  # you should see [64, 10]
test_TwoLayerFC()
```

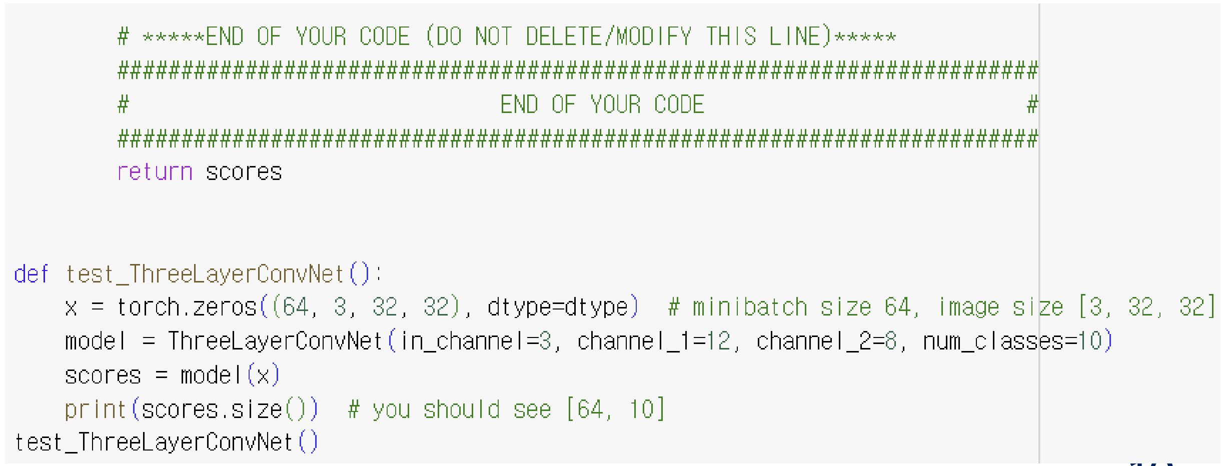


Module API 사용법: 컨볼루션 신경망 예



```
class ThreeLayerConvNet(nn.Module):
    def __init__(self, in_channel, channel_1, channel_2, num_classes):
        super().__init__()
        ########################################################################
        # TODO: Set up the layers you need for a three-layer ConvNet with the  #
        # architecture defined above.                                          #
        ########################################################################
        # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

        pass

        # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
        ########################################################################
        #                          END OF YOUR CODE                            #
        ########################################################################

    def forward(self, x):
        scores = None
        ########################################################################
        # TODO: Implement the forward function for a 3-layer ConvNet. you     #
        # should use the layers you defined in __init__ and specify the        #
        # connectivity of those layers in forward()                            #
        ########################################################################
        # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

        pass

        # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
        ########################################################################
        #                             END OF YOUR CODE                         #
```

- 2의 제로 패딩, 5x5 필터의 컨볼루션 계층
- ReLU
- 1의 제로 패딩, 3x3 필터의 컨볼루션 계층
- ReLU
- num_classes 개의 클래스 점수로의 완전 연결계층



```
        # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
        ########################################################################
        #                             END OF YOUR CODE                         #
        ########################################################################
        return scores


def test_ThreeLayerConvNet():
    x = torch.zeros((64, 3, 32, 32), dtype=dtype)  # minibatch size 64, image size [3, 32, 32]
    model = ThreeLayerConvNet(in_channel=3, channel_1=12, channel_2=8, num_classes=10)
    scores = model(x)
    print(scores.size())  # you should see [64, 10]
test_ThreeLayerConvNet()
```

Module API 사용법: 컨볼루션 신경망 예

- 생성자 __init__
  - nn.Linear, nn.Conv2d 등의 클래스들을 활용
- forward 메서드
  - self.conv1, F.relu, flatten, self.fc 등의 메서드/함수를 활용

정확도 계산

```python
def check_accuracy_part34(loader, model):
    if loader.dataset.train:
        print('Checking accuracy on validation set')
    else:
        print('Checking accuracy on test set')
    num_correct = 0
    num_samples = 0
    model.eval()  # set model to evaluation mode
    with torch.no_grad():
        for x, y in loader:
            x = x.to(device=device, dtype=dtype)  # move to device, e.g. GPU
            y = y.to(device=device, dtype=torch.long)
            scores = model(x)
            _, preds = scores.max(1)
            num_correct += (preds == y).sum()
            num_samples += preds.size(0)
        acc = float(num_correct) / num_samples
        print('Got %d / %d correct (%.2f)' % (num_correct, num_samples, 100 * acc))
```

학습

```python
def train_part34(model, optimizer, epochs=1):
    """
    Train a model on CIFAR-10 using the PyTorch Module API.

    Inputs:
    - model: A PyTorch Module giving the model to train.
    - optimizer: An Optimizer object we will use to train the model
    - epochs: (Optional) A Python integer giving the number of epochs to train for

    Returns: Nothing, but prints model accuracies during training.
    """
    model = model.to(device=device)  # move the model parameters to CPU/GPU
    for e in range(epochs):
        for t, (x, y) in enumerate(loader_train):
            model.train()  # put model to training mode
            x = x.to(device=device, dtype=dtype)  # move to device, e.g. GPU
            y = y.to(device=device, dtype=torch.long)

            scores = model(x)
            loss = F.cross_entropy(scores, y)

            # Zero out all of the gradients for the variables which the optimizer
            # will update.
            optimizer.zero_grad()

            # This is the backwards pass: compute the gradient of the loss with
            # respect to each  parameter of the model.
            loss.backward()
```

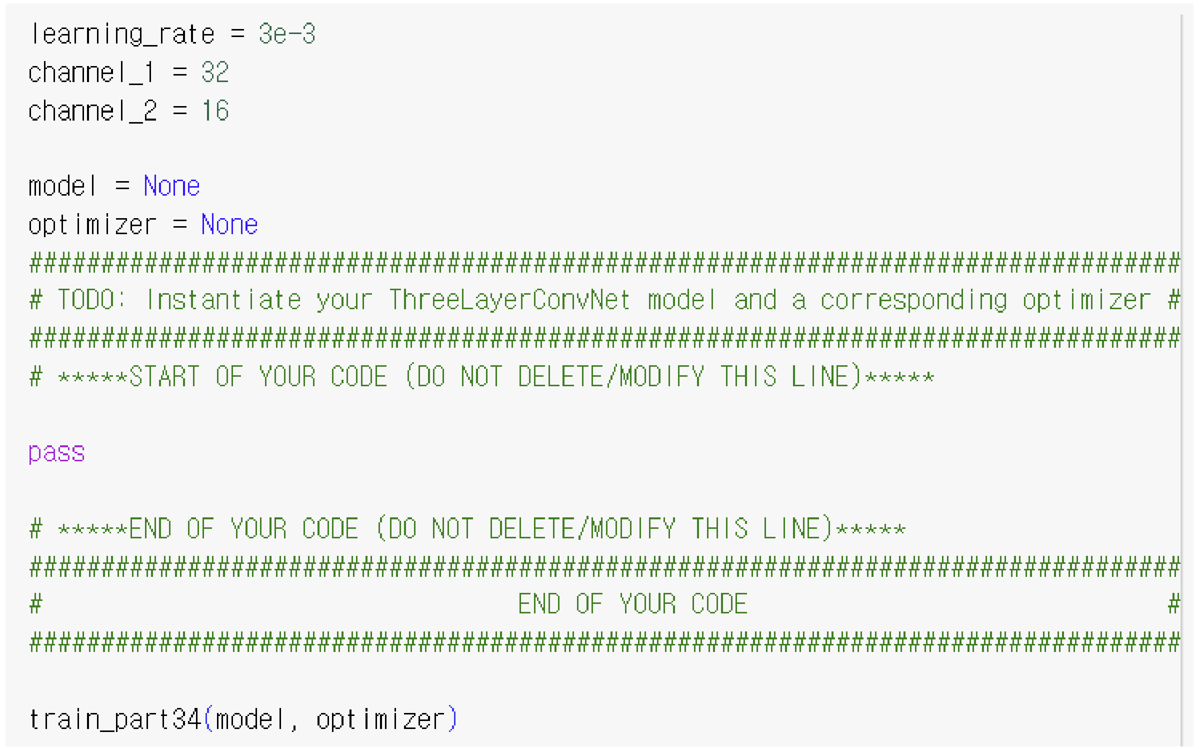
```python
            # Actually update the parameters of the model using the gradients
            # computed by the backwards pass.
            optimizer.step()

            if t % print_every == 0:
                print('Iteration %d, loss = %.4f' % (t, loss.item()))
                check_accuracy_part34(loader_val, model)
                print()
```

2계층 신경망 훈련

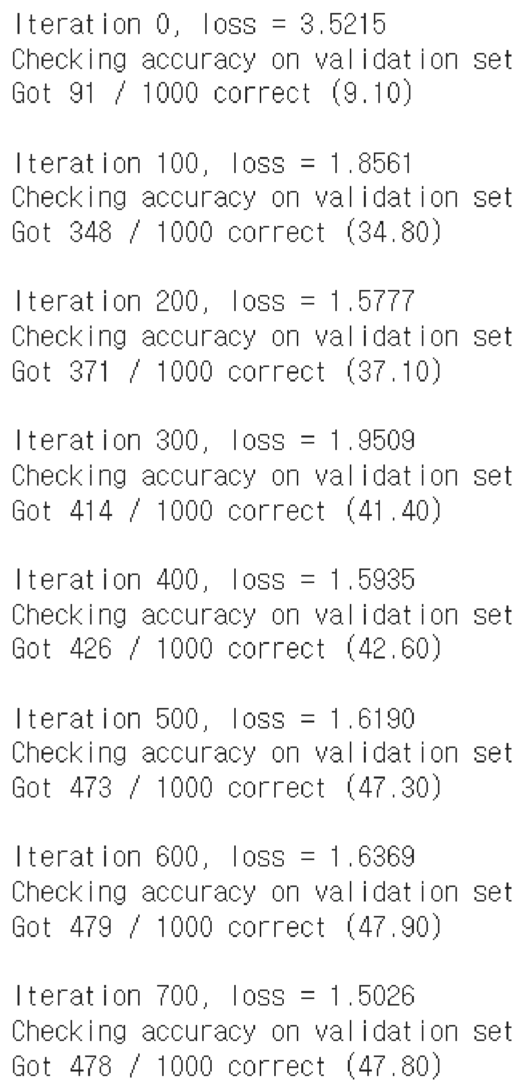
```
hidden_layer_size = 4000
learning_rate = 1e-2
model = TwoLayerFC(3 * 32 * 32, hidden_layer_size, 10)
optimizer = optim.SGD(model.parameters(), lr=learning_rate)

train_part34(model, optimizer)
```

```
Iteration 0, loss = 2.9489
Checking accuracy on validation set
Got 154 / 1000 correct (15.40)

Iteration 100, loss = 2.4830
Checking accuracy on validation set
Got 326 / 1000 correct (32.60)

Iteration 200, loss = 2.0254
Checking accuracy on validation set
Got 374 / 1000 correct (37.40)

Iteration 300, loss = 1.8018
Checking accuracy on validation set
Got 389 / 1000 correct (38.90)

Iteration 400, loss = 2.0877
Checking accuracy on validation set
Got 416 / 1000 correct (41.60)

Iteration 500, loss = 1.7459
Checking accuracy on validation set
Got 422 / 1000 correct (42.20)

Iteration 600, loss = 1.8678
Checking accuracy on validation set
Got 434 / 1000 correct (43.40)

Iteration 700, loss = 1.6545
Checking accuracy on validation set
Got 446 / 1000 correct (44.60)
```

3계층 컨볼루션 신경망 훈련



```python
learning_rate = 3e-3
channel_1 = 32
channel_2 = 16

model = None
optimizer = None
################################################################################
# TODO: Instantiate your ThreeLayerConvNet model and a corresponding optimizer #
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

pass

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
################################################################################
#                                 END OF YOUR CODE                             #
################################################################################

train_part34(model, optimizer)
```



```
Iteration 0, loss = 3.5215
Checking accuracy on validation set
Got 91 / 1000 correct (9.10)

Iteration 100, loss = 1.8561
Checking accuracy on validation set
Got 348 / 1000 correct (34.80)

Iteration 200, loss = 1.5777
Checking accuracy on validation set
Got 371 / 1000 correct (37.10)

Iteration 300, loss = 1.9509
Checking accuracy on validation set
Got 414 / 1000 correct (41.40)

Iteration 400, loss = 1.5935
Checking accuracy on validation set
Got 426 / 1000 correct (42.60)

Iteration 500, loss = 1.6190
Checking accuracy on validation set
Got 473 / 1000 correct (47.30)

Iteration 600, loss = 1.6369
Checking accuracy on validation set
Got 479 / 1000 correct (47.90)

Iteration 700, loss = 1.5026
Checking accuracy on validation set
Got 478 / 1000 correct (47.80)
```

Sequential API 사용법

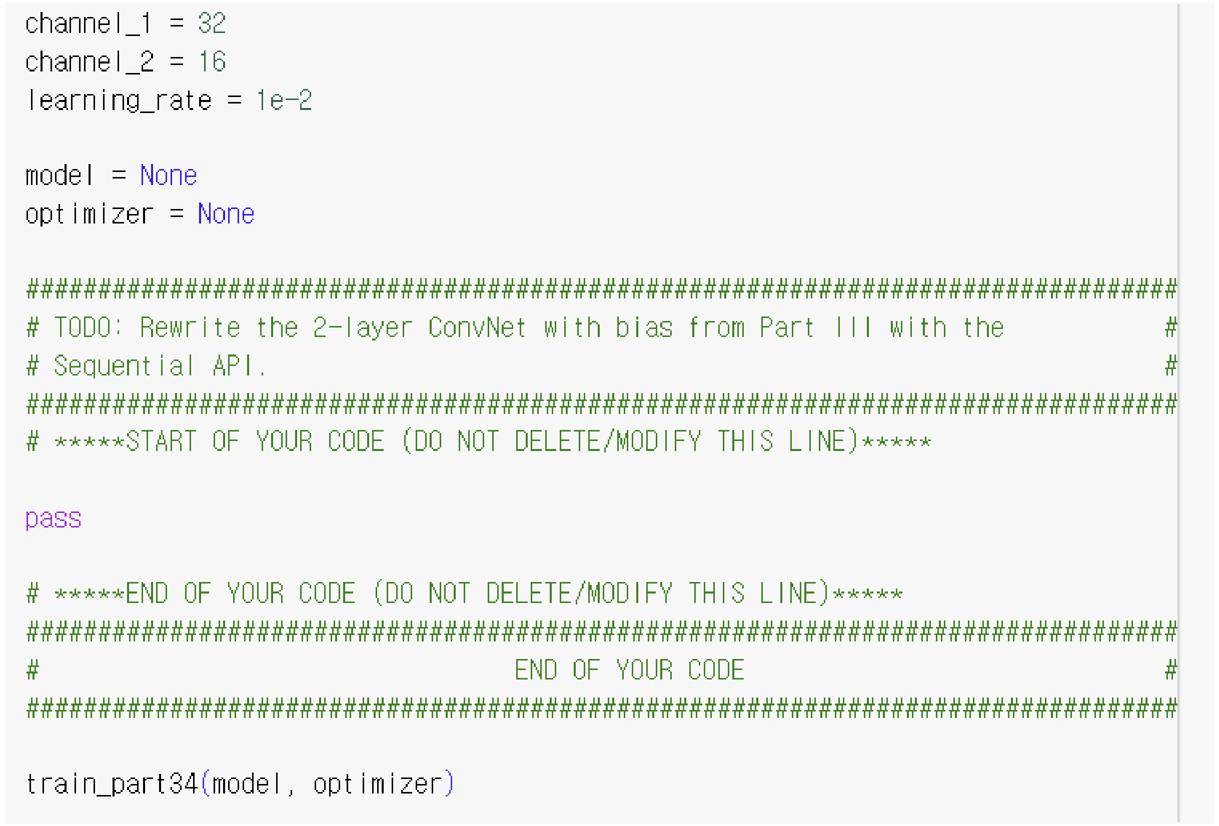
- Module API의 경우, 3단계를 거쳐야 했음
  - nn.Module의 서브클래스 생성
  - __init__ 생성자에서 클래스 속성에 계층을 하나씩 할당
  - forward() 메서드에서 각 계층을 하나씩 호출
- Sequential API (nn.Sequential)는 위 단계들을 하나로 통합
  - nn.Module API만큼 유연하지는 않음
  - feed-forward 보다 더 복잡한 구조 지정 어려움
  - 그럼에도 실제 사례에서 많이 사용됨

Sequential API 사용법: 2계층 신경망

```python
# We need to wrap `flatten` function in a module in order to stack it
# in nn.Sequential
class Flatten(nn.Module):
    def forward(self, x):
        return flatten(x)

hidden_layer_size = 4000
learning_rate = 1e-2

model = nn.Sequential(
    Flatten(),
    nn.Linear(3 * 32 * 32, hidden_layer_size),
    nn.ReLU(),
    nn.Linear(hidden_layer_size, 10),
)

# you can use Nesterov momentum in optim.SGD
optimizer = optim.SGD(model.parameters(), lr=learning_rate,
                      momentum=0.9, nesterov=True)

train_part34(model, optimizer)
```

```
Iteration 0, loss = 2.3169
Checking accuracy on validation set
Got 184 / 1000 correct (18.40)

Iteration 100, loss = 2.1072
Checking accuracy on validation set
Got 407 / 1000 correct (40.70)

Iteration 200, loss = 1.7676
Checking accuracy on validation set
Got 413 / 1000 correct (41.30)

Iteration 300, loss = 1.4692
Checking accuracy on validation set
Got 442 / 1000 correct (44.20)

Iteration 400, loss = 1.7728
Checking accuracy on validation set
Got 424 / 1000 correct (42.40)

Iteration 500, loss = 1.7285
Checking accuracy on validation set
Got 443 / 1000 correct (44.30)

Iteration 600, loss = 2.0262
Checking accuracy on validation set
Got 437 / 1000 correct (43.70)

Iteration 700, loss = 1.8091
Checking accuracy on validation set
Got 436 / 1000 correct (43.60)
```

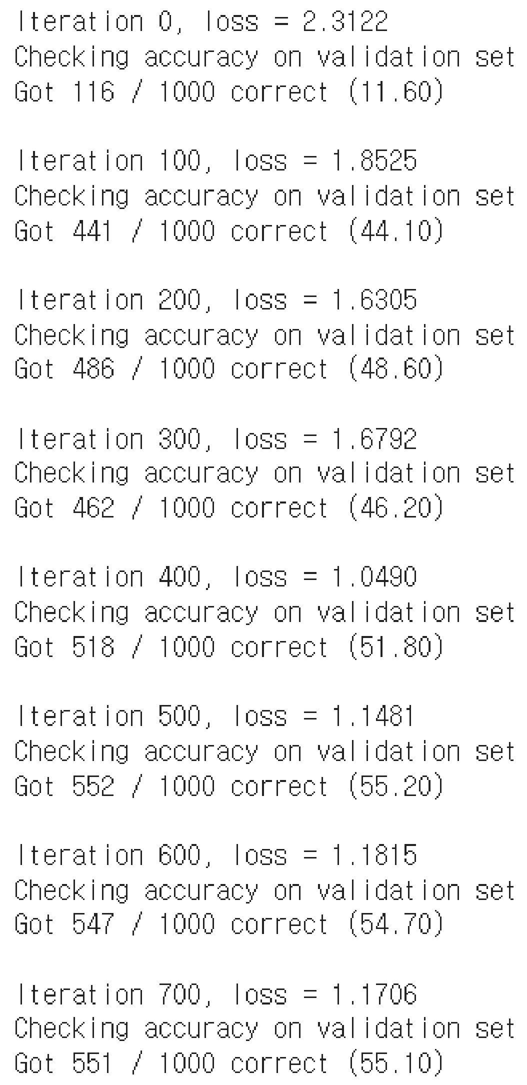
Sequential API 사용법: 3층 컨볼루션 신경망



```
channel_1 = 32
channel_2 = 16
learning_rate = 1e-2

model = None
optimizer = None

################################################################################
# TODO: Rewrite the 2-layer ConvNet with bias from Part III with the           #
# Sequential API.                                                              #
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

pass

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
################################################################################
#                                 END OF YOUR CODE                             #
################################################################################

train_part34(model, optimizer)
```

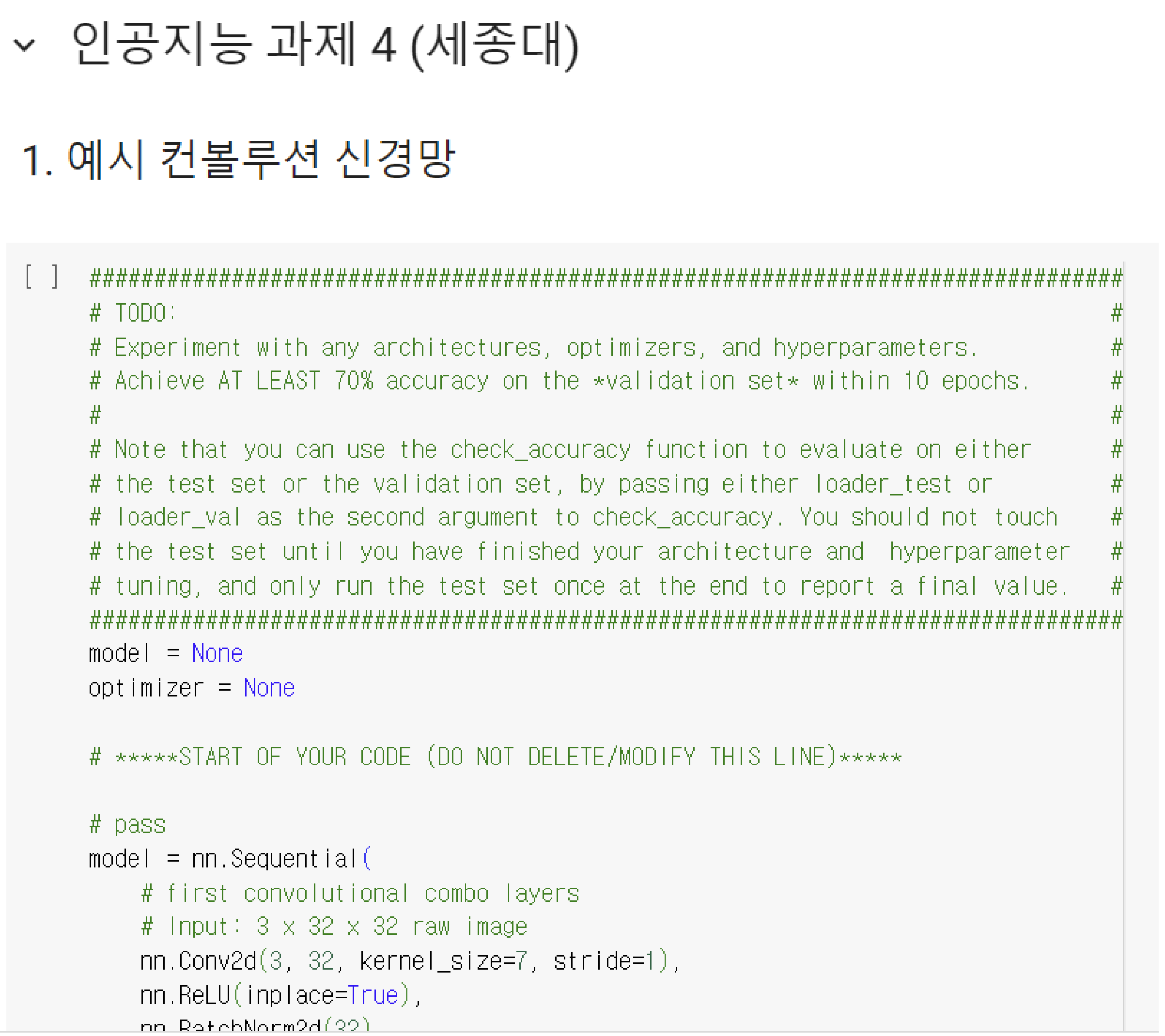
- 구조
  - 2의 제로 패딩, 5x5 필터의 컨볼루션 계층
  - ReLU
  - 1의 제로 패딩, 3x3 필터의 컨볼루션 계층
  - ReLU
  - num_classes 개의 클래스 점수로의 완전연결 계층
- PyTorch의 기본 가중치 초기화 사용
- 0.9의 모멘텀 파라미터를 가진 네스테로프 경사하강법 사용

Sequential API 사용법: 3층 컨볼루션 신경망



```
Iteration 0, loss = 2.3122
Checking accuracy on validation set
Got 116 / 1000 correct (11.60)

Iteration 100, loss = 1.8525
Checking accuracy on validation set
Got 441 / 1000 correct (44.10)

Iteration 200, loss = 1.6305
Checking accuracy on validation set
Got 486 / 1000 correct (48.60)

Iteration 300, loss = 1.6792
Checking accuracy on validation set
Got 462 / 1000 correct (46.20)

Iteration 400, loss = 1.0490
Checking accuracy on validation set
Got 518 / 1000 correct (51.80)

Iteration 500, loss = 1.1481
Checking accuracy on validation set
Got 552 / 1000 correct (55.20)

Iteration 600, loss = 1.1815
Checking accuracy on validation set
Got 547 / 1000 correct (54.70)

Iteration 700, loss = 1.1706
Checking accuracy on validation set
Got 551 / 1000 correct (55.10)
```

과제4



인공지능 과제 4 (세종대)

1. 예시 컨볼루션 신경망

```
[ ] ################################################################################
    # TODO:                                                                        #
    # Experiment with any architectures, optimizers, and hyperparameters.          #
    # Achieve AT LEAST 70% accuracy on the *validation set* within 10 epochs.      #
    #                                                                              #
    # Note that you can use the check_accuracy function to evaluate on either      #
    # the test set or the validation set, by passing either loader_test or         #
    # loader_val as the second argument to check_accuracy. You should not touch    #
    # the test set until you have finished your architecture and  hyperparameter   #
    # tuning, and only run the test set once at the end to report a final value.   #
    ################################################################################
    model = None
    optimizer = None

    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

    # pass
    model = nn.Sequential(
        # first convolutional combo layers
        # Input: 3 x 32 x 32 raw image
        nn.Conv2d(3, 32, kernel_size=7, stride=1),
        nn.ReLU(inplace=True),
```

과제$_4$ 수행을 위해 실행해야 하는 코드 블럭들

- 구글 드라이브 연동 코드 블럭 수행
- 런타임 유형 연결 변경 (CPU -> GPU)
- "GPU" GPU 코드 블럭 실행
- "Part I. Preparation" CIFAR-10 로드" 코드 블럭 실행
- "PyTorch Tensors: Flatten Function" 코드 블럭 실행
- "Module API: Check Accuracy" 코드 블럭 실행
- "Module API: Training Loop" 코드 블럭 실행
- 그 후 "인공지능 과제$_4$" 부분으로 가 1,2,3번 코드 블럭 작성 후 실행