딥러닝 홀로서기#15

주제 : How to Write Well-Organized DL Code from Scratch(Live Coding)

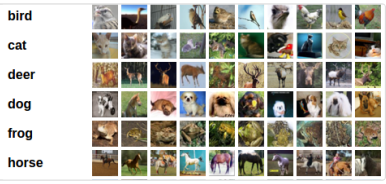
링크 :

[#15.Lab] How to Write Well-Organized DL Code from Scratch(Live Coding) - 딥러닝 홀로서기
실습자료 링크 : https://github.com/heartcored98/Standalone-DeepLearning/blob/master/Lec3/Lab4_write_pretty_DL_code.ipynb자료 저장소 링크 : https://github.com/heartcored98...
https://youtu.be/lh2Ed-b5l28

1. TRAINING A CLASSIFIER

- Pytorch 사이트의 공식 튜토리얼
  - Image data 10 way Classification
  - 링크

Training a Classifier - PyTorch Tutorials 1.7.1 documentation
Now you might be thinking, Generally, when you have to deal with image, text, audio or video data, you can use standard python packages that load data into a numpy array. Then you can convert this array into a torch.*Tensor .
https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html



- 샘플 코드에서 Editing한 부분은 붉은색으로 표시
- Import Package

```
import torch
import torchvision
import torchvision.transforms as transforms
```

- Transform 함수
  - ToTensor : Image File을 Torch Tensor로 변환 (제일 어두운 점 0 - 제일 밝은 점 1로 변환)
  - Normalize : Image Tensor의 RGB 각 채널을 Normalize해주는 것
    - 앞의 (0.5, 0.5,0.5) : RGB채널의 평균 점 0.5를 중심으로 Distribution을 Shift하여 -0.5 ~ 0.5의 Range로 세팅
    - 뒤의 (0.5, 0.5, 0.5) : Stdev (표준편차)를 0.5로 잡은 것

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainset, valset = torch.utils.data.random_split(trainset, [40000, 10000])
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)
valloader = torch.utils.data.DataLoader(valset, batch_size=4, shuffle=False)
```

```
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)


classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

- Image Inspection (=Inspect Dataset)
  - Image들이 잘 Showing되는지 테스트
  - Ground Truth 체크 : True Y가 잘 나오는 지 테스트
  - Type & Shape Check : 4가지 요소가 있는 것을 알 수 있는데, 이는 위의 코드에서 Batch_size
    - 첫번째 요소 : 해당 Batch에서 건네받은 Tensor → 사진 sample의 dimension
    - 두번째 요소 : 1이었을 땐 흑백, 3은 RGB Scale
    - 세번째, 네번째 요소 : 32 x 32
  - Label Check
    - Label shape은 차원을 나타냄
    - Labels은 어떤 사진인지에 대한 Class 값

```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))

print(type(images), images.shape)
print(type(labels), labels.shape, labels)
```

- Define a Convolutional Neural Network
  - 모델을 작성

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self, in_dim, out_dim, hid_dim, n_layer, act):
        super(MLP, self).__init__()
        self.in_dim = in_dim
        self.out_dim = out_dim
        slef.hid_dim = hid_dim
        self.n_layer = n_layer
        self.act = act

        self.fc = nn.Linear(self.in_dim, self.hid_dim)
        self.linears = nn.ModuleList()
```

```
        for i in range(self.n_layer-1)
            self.linears.append(nn.Linear(self.hid_dim, self.hid_dim))
        self.fc2 = nn.Linear(self.hid_dim, self.out_dim)

        if self.act == 'relu':
            self.act = nn.ReLU()
        else:
            raise RunTimeError

    def forward(self, x):
        x = self.act(self.fc(x))
        for fc in self.linears:
            x = self.act(fc(x))
        x = self.fc2(x)
        return x

model = MLP(3072, 10, 100, 2, 'relu')
print(model)
```

- Define Loss Function and Optimizer (Sample Code)

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

- Train the network (Sample Code)

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        print(inputs.shape) #original shape
        inputs = inputs.view(-1, 3072) #view : reshaping function
        print(inputs.shape)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

- Test the network on the test data (Sample Code)

```
dataiter = iter(testloader)
images, labels = dataiter.next()

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
```

- Test Whole Dataset (Sample Code)

```
# ===== Measure Test Accuracy ==== #
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        images = images.view(-1, 3072)
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

- Validation Set Accuracy (Sample Code)

```
# ==== Measure Val Accuracy / Val Loss ==== #
correct = 0
total = 0
with torch.no_grad():
    for data in valloader:
        images, labels = data
        images = images.view(-1, 3072)
        outputs = net(images)

        loss = criterion(outputs, labels)
        val_loss += loss.item()
        print(loss) #Loss of Each Batch

        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print(len(valloader))
    val_loss = val_loss / len(valloader)
    print(val_loss)
    acc = 100 * correct / total

print('Accuracy of the network on the 10000 test images: {:2.2f}% Loss: {:2.2f}%'.format(acc, val_loss))
```

- Experiment

```
def experiment(args):

    net = MLP(args.in_dim, args.out_dim, args.hid_dim, args.n_layer, args.act)
    net.cuda()
    print(net)

    #위의 Optimizer Code를 복붙 후 숫자들을 변수명으로 교체
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=args.lr, momentum=args.mm)

    #위의 Training Code를 복붙
    for epoch in range(args.epoch):  # loop over the dataset multiple times

        running_loss = 0.0
        train_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data
            inputs = inputs.view(-1, 3072) #view : reshaping function
            inputs = inputs.cuda()
            labels = labels.cuda()

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
```

```
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            train_loss += loss.item()
            if i % 2000 == 1999:    # print every 2000 mini-batches
                print('[%d, %5d] loss: %.3f' %
                      (epoch + 1, i + 1, running_loss / 2000))
                running_loss = 0.0

        # ==== Validation ==== #
        correct = 0
        total = 0
        val_loss = 0
        with torch.no_grad():
            for data in valloader:
                images, labels = data
                images = images.view(-1, 3072)

                images = images.cuda()
                labels = labels.cuda()
                outputs = net(images)

                loss = criterion(outputs, labels)
                val_loss += loss.item()
                print(loss) #Loss of Each Batch

                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

            print(len(valloader))
            val_loss = val_loss / len(valloader)
            print(val_loss)
            val_acc = 100 * correct / total

            print('Epoch: {}, Trian Loss: {}, Val Loss: {}, Val Acc: {}'.format(epoch, train_loss, val_loss, val_acc))

    # ==== Evaluation ==== #
    correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            images, labels = data
            images = images.view(-1, 3072)
            images = images.cuda()
            labels = labels.cuda()

            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        test_acc = 100 * correct / total

    return train_loss, val_loss, val_acc, test_acc
```

- Argparse

```
import argparse

seed = 123
np.random.seed(seed)
torch.manual_seed(seed)


parser = argparse.ArgumentParser()
args = parser.parse_args("")

args.n_layer = 5
args.in_dim = 3072
args.out_dim = 10
args.hid_dim = 100
args.act = 'relu'
args.lr = 0.001
args.mm = 0.9
args.epoch = 2
```

```
list_var1 = [4, 5, 6]
list_var2 = [50, 100, 150]

for var1 in list_var1 :
    for var2 in list_var2 :
        args.n_layer = var1
        args.hid_dim = var2
        result = experiment(args)
        print(result)
```

- Assignment
  - 위의 코드를 활용하여 하이퍼파라미터 값을 튜닝하여 Accuracy를 높이기
  - Regularization (L2, Dropout 등)을 파이토치 검색하여 Example 을 참고하여 직접 적용하고, 사용한 것과 사용하지 않은 것을 비교분석
  - 중요! Test Accuracy를 절대 보지 않고, Validation Set의 Accuracy를 보고 개선!