[PyTorch] TRAINING A CLASSIFIER

이미지 분류 튜토리얼 ::
https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#sphx-glr-beginner-blitz-cifar10-tutoria

Data

일반적으로 이미지, 텍스트, 오디오, 비디오 등의 데이터는 파이썬 패키지를 이용해 NumPy 배열로 불러온다. 그 후 배열을 **torch.*Tensor** 로 바꾸어준다.

(파이토치는 Tensor를 써야함)

주로쓰는 라이브러리

- For images: **Pillow, OpenCV**
- For audio: **scipy** and **librosa**
- For text: raw Python, Cython based loading, or **NLTK** and **SpaCy**

torchvision

- `torchvision.datasets` **:** ImageNet, CIFAR10, MNIST 등과 같은 데이터셋을 위한 data loader
- `torch.utils.data.DataLoader` : 이미지용 데이터 변환기 (data transformer)

  → Dataset을 샘플에 쉽게 접근할 수 있도록 순회 가능한 객체(iterable)로 감싸줌

해당 튜토리얼에서는 CIFAR 10 데이터셋을 사용하였다.

CIFAR 10

- 클래스: ‘비행기(airplane)’, ‘자동차(automobile)’, ‘새(bird)’, ‘고양이(cat)’, ‘사슴(deer)’, ‘개(dog)’, ‘개구리(frog)’, ‘말(horse)’, ‘배(ship)’, ‘트럭(truck)’
- 이미지 크기: 3x32x32

  *. numpy.ndarray의 경우 “H x W x C” 구조를 가지지만, 파이토치의 텐서는 “**C x H x W**" 구조이다.

Training an image classifier

steps :

💡
1. torchvision을 사용해 CIFAR의 training/test dataset을 불러오고, nomarlize함
2. CNN 정의
3. loss function 정의
4. training data를 이용해 네트워크 훈련
5. test data를 이용해 네트워크 테스트

1. Load and normalize CIFAR10

라이브러리 불러오기

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

torchvision 데이터셋의 output은 [0, 1] 범위를 갖는 PILImage이기때문에 [-1, 1]로 정규화된 Tensor로 변환해준다.

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

```
batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)

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

- `transforms.Normalize((mean1, mean2, mean3), (std1, std2, std3))`

  image = (image - mean) / std 로 정규화 시켜줌

- `batch_size`

  batch의 크기. (e.g.) 50개 데이터가 있을 때, batch_size=10일 경우 50/10=5번의 iteration을 통해 전체 데이터셋 돌 수 있음

- `num_workers`

  — how many subprocesses to use for data loading. 0 means that the data will be loaded in the main process. (default: 0)

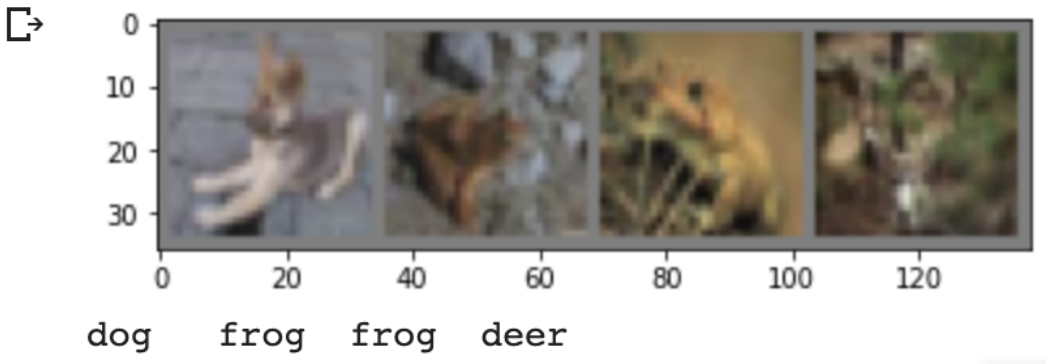
  데이터 로드 멀티 프로세싱에 대한 파라미터다. 데이터 로드 작업을 단일코어가 아닌 멀티코어로 처리할 때, 몇개의 코어를 사용할 지를 결정한다. num_workers를 설정할 때에 고려할 요소는 '학습 환경의 GPU개수, CPU개수, I/O 속도, 메모리 등'이 있다.

이미지 확인

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


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

```
    plt.show()


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

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```

>>>



*. 배치 사이즈를 4로 설정해서 데이터가 4개씩 나오는 걸 볼 수 있다.

2. Define CNN

신경망을 정의해준다.

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


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
```

```
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10) # # of class = 10

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

3. Define Loss function and Optimizer

- loss function: cross-entropy loss
- optimizer: SGD with momentum

Cross-Entropy Loss란

classification에서 자주 쓰이는 손실 함수로 주로 소프트맥스와 함께 쓰인다.

소프트맥스가 아웃풋을 0~1 사이의 확률값(전체 합=1)로 바꾸어주면 크로스-엔트로피 손실함수로 타겟(1-핫 인코딩 된)과의 오차를 구하는 것이다.

식은 다음과 같다 $CE = -\sum t_i log(s_i)$

아래는 binary classification(C=2)에서의 cross entropy 함수이다.

$l = -(ylog(p) + (1-y)log(1-p))$

- p: the predicted probability
- y is the indicator (0 or 1 in the case of binary classification)

*. 파이토치의 크로스 엔트로피는 소프트맥스와 합쳐져있기때문에 굳이 소프트맥스 레이어를 추가할 필요가 없다고한다.

```
import torch.optim as optim

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

4. Train the Network

단순히 데이터를 반복해서 신경망에 입력하고, optimize하며 신경망을 학습시킬 수 있다.

```
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

        # 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(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0

print('Finished Training')
```

>>>

```
[1,  2000] loss: 2.235
[1,  4000] loss: 1.911
[1,  6000] loss: 1.714
[1,  8000] loss: 1.612
[1, 10000] loss: 1.537
[1, 12000] loss: 1.491
[2,  2000] loss: 1.444
[2,  4000] loss: 1.385
[2,  6000] loss: 1.396
[2,  8000] loss: 1.360
[2, 10000] loss: 1.319
[2, 12000] loss: 1.308
Finished Training
```

Save model

```
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
```

5. Test the network on the test data

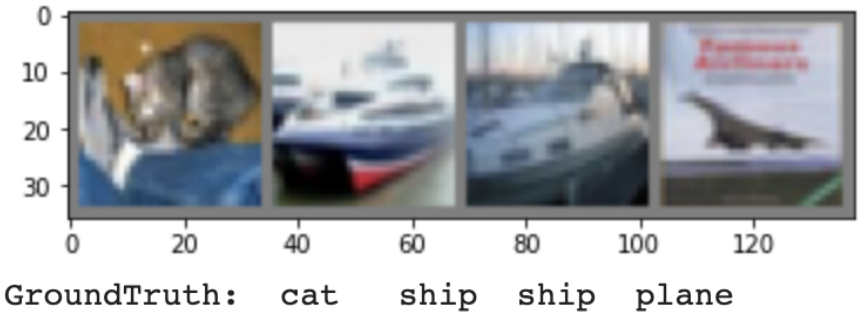
위에서 학습용 데이터셋을 2번 반복하여 신경망을 학습시켰는데, 학습이 제대로 되었는지 확인할 필요가 있다.

신경망이 예측한 출력과 정답(Ground-truth)를 비교한 후 만일 예측값이 맞다면 샘플을 ‘correct predictions’에 추가한다.

test data 확인하기

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

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

>>>



저장했던 모델 불러오기

```
net = Net()
net.load_state_dict(torch.load(PATH))
```

예측하기

```
outputs = net(images)
```

예측한 값은 각 클래스에 대한 확률을 나타낸다. 따라서 가장 높은 확률을 가지는 클래스의 index를 추출하여 확인한다.

```
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join(f'{classes[predicted[j]]:5s}'
                              for j in range(4)))
```

>>>

```
Predicted:  cat   deer  horse frog
```

~~cat 말고 다 틀렸는데요…~~

전체 데이터셋에 대해 확인하기

```
correct = 0
total = 0
# since we're not training, we don't need to calculate the gradients for our outputs
with torch.no_grad():
    for data in testloader:
        images, labels = data
        # calculate outputs by running images through the network
        outputs = net(images)
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

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

- **with.torch.no_grad():**

  미분을 하지 않겠다는 의미. 학습된 모델이 제대로 작동하는지 확인하기 위해 gradient를 사용하지 않는, 즉 back propagation을 하지 않는 것. 마찬가지로 image의 차원을 조절해 준 다음 model에 전달함

>>>

```
Accuracy of the network on the 10000 test images: 53 %
```

각 class에 대한 예측값 계산하여 어떤 클래스를 잘 분류하고 어떤 클래스를 잘 못 분류하는지 확인

```
# prepare to count predictions for each class
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}
```

```
# again no gradients needed
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)
        # collect the correct predictions for each class
        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1


# print accuracy for each class
for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```

>>>

```
Accuracy for class: plane is 75.0 %
Accuracy for class: car   is 74.9 %
Accuracy for class: bird  is 23.0 %
Accuracy for class: cat   is 34.2 %
Accuracy for class: deer  is 35.8 %
Accuracy for class: dog   is 60.9 %
Accuracy for class: frog  is 68.2 %
Accuracy for class: horse is 51.2 %
Accuracy for class: ship  is 58.5 %
Accuracy for class: truck is 50.6 %
```

Training on GPU

GPU 환경에서 신경망을 학습한다. 이를 위해 네트워크를 GPU로 옮겨주어야한다.

우선, CUDA를 사용할 수 있다면 첫번째 CUDA 장치를 사용하도록 한다.

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

```
print(device) # cuda:0
```

device를 설정했다면, 아래의 코드를 활용해서 network와 데이터(input & target)을 device(GPU)로 보내주어야한다

```
net.to(device)

inputs, labels = data[0].to(device), data[1].to(device)
```