CV DEEP LEARNING WITH PYTORCH

PART

MODEL TRAIN & TEST

MODEL TRAIN & TEST

◆ 학습 준비 - DATASET

❖ 학습용	모델의 최적의 W, b를 찾기 위한 데이터셋
❖ 검증용	모델의 최적화 진행 상태 즉, 학습 진행 체크 데이터셋
❖ 테스트용	모델 최적화 후 테스트에 사용할 데이터셋

MODEL TRAIN & TEST

◆ 학습 준비 - 최적화 모듈

- torch.optim 모듈
- 최적화 알고리즘에따라 모델의 W, b 업데이트 처리 클래스
- 필수 파라미터 : 모델 파라미터(model.parameters())와 학습률(LR)

torch.optim.**SGD**	• 가장 기본적인 알고리즘 • 각각의 파라미터에 대해 학습률(learning rate) 곱한 값을 사용하여 가중치 업데이트
torch.optim.**Adam**	• 학습률을 각 파라미터마다 적응적으로 조절하는 알고리즘 • 현재 그래디언트와 이전 그래디언트의 지수 가중 평균을 이용하여 가중치 업데이트
torch.optim.**RMSprop**	• 그래디언트의 제곱값의 이동 평균 이용하여 학습률 조절 알고리즘 • Adam과 유사한 방식으로 학습률 조절
torch.optim.**Adagrad**	• 각 파라미터에 대한 학습률을 조절 알고리즘 • 이전 그래디언트의 제곱의 누적 값을 사용하여 학습률 조절

MODEL TRAIN & TEST

◆ 학습 준비 - 손실/비용 처리 함수

- 모델 예측 값과 타겟 값의 차이 계산 클래스 / 함수

유형		손실함수	
회귀		torch.nn.**MSELoss**	평균 제곱 오차(Mean Squared Error) 계산
		torch.nn.**L1Loss**	평균 절대 오차(Mean Absolute Error) 계산
분류	다중	torch.nn.**CrossEntropyLoss**	교차 엔트로피 손실(Cross Entropy Loss) 계산
	이진	torch.nn.**BCELoss**	이진 교차 엔트로피 손실(Binary Cross Entropy Loss) 계산

MODEL TRAIN & TEST

◆ 성능지표 - 회귀

- 모델의 성능을 평가하는 기준

MSE	평균 제곱 오차(Mean Squared Error) 계산 제곱으로 1미만 에러는 작아지고, 그 이상은 에러가 커짐 / 스케일에 의존적
MAE	평균 절대 오차(Mean Absolute Error) 계산 특이값이 많은 경우 사용되는 성능지표 / 스케일에 의존적
RMSE	MSE값은 실제 오류의 평균보다 값이 큼, 루트를 취한 값 제곱으로 1미만 에러는 작아지고, 그 이상은 에러가 커짐 / 스케일에 의존적
R2 Squard	다른 지표들은 모델마다 값이 다르기 때문에 성능 판단이 어려움 실제 값의 분산 대비 예측값의 분산 비율로 상대적인 성능 비교 가능 1에 가까울 수록 좋은 모델

MODEL TRAIN & TEST

◆ 성능지표 - 분류

- 모델의 성능을 평가하는 기준

Accuracy	전체 데이터 중에서 정확하게 에측한 데이터 수 불균형 데이터의 경우 적합하지 않음
Precision	양성으로 판단한 것 중, 진짜 양성의 비율
Recall /Sensitivity	진짜 양성인 것들 중에서, 올바르게 양성으로 판단한 비율
F1 Score	정밀도와 재현율이 한쪽으로 치우치지 않는 수치 1에 가까울 수록 좋은 성능

MODEL TRAIN & TEST

◆ 성능지표 - 다중분류

▪ 모델의 성능을 평가하는 기준

Multi-class	- 하나의 이미지에 하나의 객체만 존재 - 2개 이상의 카테고리로 분류되는 경우
Multi-label	- 하나의 이미지에 여러 객체가 존재 - 하나의 이미지가 1개 카테고리에 속하지 않고 여러 개의 카테고리에 속함

MULTI-CLASS

MULTI-LABEL

MODEL TRAIN & TEST

◆ 성능지표 - 다중분류

micro 평가

- 모든 클래스별 성능 측정하여 평균
- 균형 클래스 적용
- 모든 클래스를 하나의 대규모 클래스로 취급

전체적인 성능 측정

- 정밀도 = (1+1+1)/(1+1+1+1) = 0.75
- 재현율 = (1 + 1 + 1)/(1+1+1+1) = 0.75
- f1 =0.75 * ((0.75*0.75)/(0.75+0.75))= 0.75
- 정확도=(1+1+1)/(1+1+1+1) = 0.75

	0	1	2
0	**1 (TP)**	1 (FP)	0 (FP)
1	0 (FP)	**1 (TP)**	0 (FP)
2	0 (FP)	0 (FP)	**1 (TP)**

MODEL TRAIN & TEST

◆ 성능지표 - 다중분류

macro 평가

- 각 클래스별 성능 측정하여 평균
- 불균형 클래스 적용
- 클래스별 동일 가중치 부여 평균

- **정밀도 : (1.0+0.5+1.0) / 3 = 8.33333.....**
  * 0 정밀도 = 1 / (1+0+0) = 1.0
  * 1 정밀도 = 1 / (1+0+1) = 0.5
  * 2 정밀도 = 1 / (0+0+1) = 1.0
- **재현율 = (0.5 + 1.0 + 1.0)/3 = 0.833333**
  * 0 재현율 = 1 / (1+1+0) = 0.5
  * 1 재현율 = 1 / (0+1+0) = 1.0
  * 2 재현율 = 1 / (0+0+1) = 1.0
- **f1 = (0.6666+0.6666+1.0)/3 = 0.77773**
  * 0 f1 = ((1.0+0.5)/(1.0+0.5))*2= 0.66666
  * 1 f1 = ((0.5+1.0)/(1.0+0.5))*2= 0.66666
  * 2 f1 = ((1.0+1.0)/(1.0+1.0))*2= 1.0

	0	1	2
0	1	1	0
1	0	1	0
2	0	0	1

MODEL TRAIN & TEST

◆ 성능지표 - 다중분류

macro 평가

- 각 클래스별 성능 측정하여 평균
- 불균형 클래스 적용
- 클래스별 동일 가중치 부여 평균

- 정확도 =
  * 0 정확도 = (1 + 2) / (1+0+2+1) = ¾=0.75
  * 1 정확도 =
  * 2 정확도 =

	0 Positive	1 Negative	2 Negative
0 Positive	1 TP	1 FN	0
1 Negative	0 FP	1 TN	0
2 Negative	0	0	1

MODEL TRAIN & TEST

◆ 성능지표 - 다중분류

macro 평가

- 각 클래스별 성능 측정하여 평균
- 불균형 클래스 적용
- 클래스별 동일 가중치 부여 평균

- 정확도 =
  * 0 정확도 = (1 + 2) / (1+0+1+2) = ¾=0.75
  * 1 정확도 = **(1 + 2) / (1+1+2+0) = ¾=0.75**
  * 2 정확도 =

	1 Positive	0 Negative	2 Negative
1 Positive	1 **TP**	0 **FN**	0
0 Negative	1 **FP**	1 **TN**	0
2 Negative	0	0	1

MODEL TRAIN & TEST

◆ 성능지표 - 다중분류

macro 평가

- 각 클래스별 성능 측정하여 평균
- 불균형 클래스 적용
- 클래스별 동일 가중치 부여 평균

- 정확도 = (0.75+0.75+1.0)/3 = 0.83333
  * 0 정확도 = (1 + 2) / (1+0+1+2) = ¾=0.75
  * 1 정확도 = (1 + 2) / (1+1+0+2) = ¾=0.75
  * 2 정확도 = (1 + 3) / (1+ 0 + 3 + 0) = 1.0

	2 Positive	0 Negative	1 Negative
2 Positive	1 TP	0 FN	0
0 Negative	0	1 TN	1
1 Negative	0 FP	0	1

PART

TORCHVISION

TORCHVISION

◆ PyTorch 비젼 라이브러리

- 파이토치에서 제공하는 **이미지 데이터셋들이 모여 있는 패키지**
- MNIST, ImageNet을 포함한 **유명한 데이터셋들을 제공**

- **모델 아키텍쳐 및 컴퓨터 비젼 위한 이미지 변환 기능 제공**
  - 유명 이미지 데이터셋 로딩 기능 : CIFAR, COCO, MNIST, ImageNet
  - 미리 학습된(pre-trained) 이미지 분류 모델 제공 : VGG, ResNet, Inception
  - 이미지 전처리 기능 제공 : Transfrom 다양한 함수 제공
  - 다양한 유틸 함수 제공 : Utils

TORCHVISION

◆ 데이터 증강 Data Augmentation

- 데이터에 변형 가하여 **데이터 규모 증가 및 변형된 다양한 데이터 케이스 학습 제공**
- 효과 : **모델 과적합(overfitting) 방지**
  - Flip (Horizontal, Vertical)
  - Random Crop
  - Shear
  - Rotate
  - Zoom
  - Blur

TORCHVISION

◆ IMAGE DATASET - BUILT-IN

- 컴퓨터 비젼 유명한 데이터 셋
- Pytorch에서 학습용으로 제공하는 데이터 셋
- https://pytorch.org/vision/stable/datasets.html

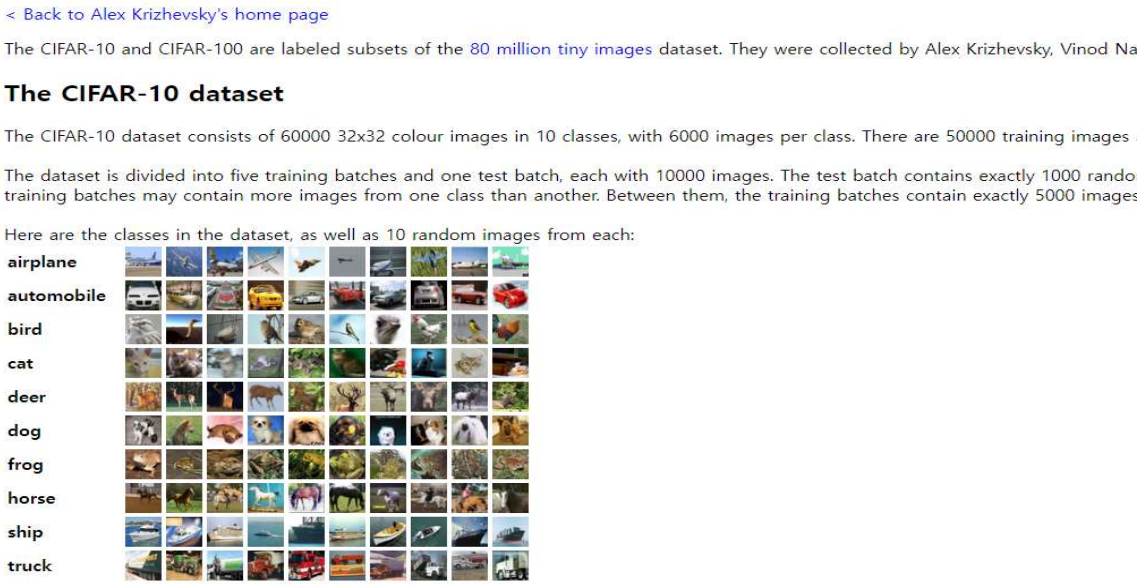
`Caltech101`(root[, target_type, transform, ...])	Caltech 101 Dataset.
`Caltech256`(root[, transform, ...])	Caltech 256 Dataset.
`CelebA`(root[, split, target_type, ...])	Large-scale CelebFaces Attributes (CelebA) Dataset Dataset.
`CIFAR10`(root[, train, transform, ...])	CIFAR10 Dataset.

TORCHVISION

◆ IMAGE DATASET - BUILT-IN

▪ CIFAR10

- 머신러닝과 컴퓨터 비전 알고리즘에 사용되는 8000만 개의 작은 이미지 데이터 세트
- 이미지 : 32 x 32 60000개
- 클래스 : 10개 (각 6000개)
- 학습용 : 50000개
- 테스트용: 10000개



< Back to Alex Krizhevsky's home page

The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Na

The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 rando training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images

Here are the classes in the dataset, as well as 10 random images from each:

TORCHVISION

◆ IMAGE DATASET - BUILT-IN

- **CIFAR10**

CLASS torchvision.datasets.CIFAR10(*root: Union[str, Path], train: bool = True, transform: Optional[Callable] = None, target_transform: Optional[Callable] = None, download: bool = False*) [SOURCE]

CIFAR10 Dataset.

Parameters:

- **root** (str or `pathlib.Path`) – Root directory of dataset where directory `cifar-10-batches-py` exists or will be saved to if download is set to True.
- **train** (*bool, optional*) – If True, creates dataset from training set, otherwise creates from test set.
- **transform** (*callable, optional*) – A function/transform that takes in a PIL image and returns a transformed version. E.g, `transforms.RandomCrop`
- **target_transform** (*callable, optional*) – A function/transform that takes in the target and transforms it.
- **download** (*bool, optional*) – If true, downloads the dataset from the internet and puts it in root directory. If dataset is already downloaded, it is not downloaded again.

TORCHVISION

◆ IMAGE DATASET - BUILT-IN

▪ CIFAR10

```
import torch
from torchvision import datasets
from torchvision.transforms import ToTensor, Lambda

ds = datasets.MNIST( root="data",
                     train=True,
                     download=True,
                     transform=ToTensor(),
                     target_transform=Lambda(lambda y:
                                             torch.zeros(10, dtype=torch.float)
                                             .scatter_(0, torch.tensor(y), value=1))
)
```

TORCHVISION

◆ IMAGE DATASET - CUSTOM

- 사용자 정의 데이터셋 관련 모듈

`DatasetFolder`(root, loader[, extensions, ...])	A generic data loader.
`ImageFolder`(root, transform, ...)	A generic data loader where the images are arranged in this way by default: .
`VisionDataset`([root, transforms, transform, ...])	Base Class For making datasets which are compatible with torchvision.

◆ IMAGE DATASET - CUSTOM

- **사용자 정의 데이터셋 모듈 - ImageFolder**

  폴더 아래 이미지 파일을 로딩하여 폴더명을 라벨로 데이터셋 생성

  폴더 구조가 정확해야 함

```
root/dog/xxx.png
root/dog/xxy.png
root/dog/[...]/xxz.png

root/cat/123.png
root/cat/nsdf3.png
root/cat/[...]/asd932_.png
```

root	: 이미지가 존재하는 폴더 경로
transform	: 이미지 변환/가공 전처리할 인스턴스
target_transform	: 타겟 변환/가공 전처리 인스턴스
loader	: 이미지 로딩 방식 설정
is_valid_file	: 이미지 파일의 유효성 검사 함수 설정
allow_empty	: 빈 폴더 허용 여부 설정 [기 False]

TORCHVISION

◆ IMAGE DATASET - CUSTOM

- **사용자 정의 데이터셋 모듈 - ImageFolder**

```
### ===> 모듈로딩
import torch
from torchvision.datasets import ImageFolder
import torchvision.transforms as transforms
import matplotlib.pyplot as plt


### ===> 이미지 데이터 존재하는 폴더 경로 설정
img_dir = '../data/img/'

### ===> 이미지 데이터셋 변환 : 지정된 경로 아래 폴더기반 데이터셋 변환
imgDS = ImageFolder(root=img_dir,
                    transform=transforms.ToTensor())
```

◆ IMAGE DATASET - CUSTOM

- **사용자 정의 데이터셋 모듈 - ImageFolder**

```
### ===> 이미지 데이터셋 확인
print(f'[Image Dataset]\n{imgDS}')
print(f'[classes]  {imgDS.classes},  {imgDS.class_to_idx}')
print(f'[Targets]  {imgDS.targets}')
print('[imgs]')
for item in imgDS.imgs : print(item)

### ===> 이미지 시각화 확인
for item in imgDS.imgs :
    img=imgDS.loader(item[0])
    plt.imshow(img)
    plt.title(f"[Label {item[1]}]")
    plt.show()
```

TORCHVISION

◆ IMAGE TRANSFORMS

- 이미지 **데이터의 전처리 및 데이터 증강** 위해 제공하는 모듈
- **torchvision.transforms.**XXX( )함수 기능

Resize	이미지 크기 조절
RandomResizedCrop	무작위 자르고 크기 조절
RandomHorizontalFlip	무작위 수평으로 뒤집기
RandomVerticalFlip	무작위 수직으로 뒤집기
ToTensor	이미지 텐서로 변환
Normalize	이미지 정규화

RandomRotation	이미지 무작위 회전
RandomCrop	이미지 무작위 자릅
Grayscale	이미지 흑백으로 변환
RandomSizedCrop	이미지 무작위 자르고 크기 조절
ColorJitter	이미지의 색상 무작위 조정

TORCHVISION

◆ IMAGE TRANSFORMS

- 이미지 데이터 전처리 기법들 구성 기능

```
from torchvision import transforms


transform = transforms.Compose(
    [
        transforms.Resize(size=(512, 512)),
        transforms.ToTensor()
    ]
)
```

원하는 변형 조합 구성

TORCHVISION

◆ IMAGE TRANSFORMS

- **이미지 데이터 전처리 - 정규화(1)**

```
from torchvision import transforms


transform = transforms.Compose(
    [
        # 0 ~ 1 범위 정규화 및 텐서화
        transforms.ToTensor()
    ]
)
```

TORCHVISION

◆ IMAGE TRANSFORMS

- **이미지 데이터 전처리 - 정규화(2)**

```
from torchvision import transforms

transform = transforms.Compose(
    [
        # 0 ~ 1 범위 정규화 및 텐서화
        transforms. Normalize( (R채널 평균, G채널 평균, B채널 평균),
                               (R채널 표준편차, G채널 표준편차, B채널 표준편차) )
    ]
)
```

TORCHVISION

◆ IMAGE TRANSFORMS

- **이미지 데이터 전처리 - 정규화(2)**

```
### ===> ImageNet이 학습한 수백만장의 이미지의 RGB 각각의 채널에 대한 평균

transform = transforms.Compose(
    [
        # 이미지의 RGB 각각의 채널에 대한 평균과 표준편차
        transforms. Normalize( (0.485, 0.456, 0.406), (0.229, 0.224, 0.225) )
    ]
)
```

TORCHVISION

◆ IMAGE TRANSFORMS

- **이미지 데이터 전처리 및 데이터셋 구성**

```
### ===> 이미지 데이터 존재하는 폴더 경로 설정
img_dir = '../data/train_img/'

### ===> 이미지 사이즈 조정 및 텐서화
preprocessing = transforms.Compose([
    transforms.Resize((50, 50)),
    transforms.ToTensor()
])

### ===> 이미지 데이터셋 변환 : 지정된 경로 아래 폴더기반 데이터셋 변환
imgDS = ImageFolder( root=img_dir,  transform=preprocessing )
```

TORCHVISION

◆ PREPROCESSING SOLUTION

- **가중 무작위 추출 : 각 클래스별 동일 개수 구성된 배치 데이터 생성**

```
def make_weights(labels, nclasses):
    labels = np.array(labels)
    weight_list=[]

    for cls in range(nclasses):
      count = len( np.where(labels == cls)[0]  )    # 각 클래스별 라벨 개수 파악

      weight = 1/count                              # 라벨 뽑힐 가중치 : 1/count, 라벨 전체 동일 할당
      weights = [weight] * count                    # 라벨 뽑힐 가중치
      weight_list+=weights                          # 데이터 로딩 시 라벨 0부터 N 까지 차례대로
                                                      나열 => 각 클래스의 가중치 일렬로 이어줌.
  return weight_list
```

TORCHVISION

◆ PREPROCESSING SOLUTION

- **가중 무작위 추출 : 각 클래스별 동일 개수 구성된 배치 데이터 생성**

```
# 이미지 데이터 불러오기
preprocessing = tr.Compose( [ tr.Resize((16,16)),  tr.ToTensor() ] )
trainset = ImageFolder(root = ../data/img',  transform= preprocessing)

# 가중치 생성 ➔ 텐서 변환
weights = make_weights( trainset.targets, len(trainset.classes) )
weights = torch.DoubleTensor(weights)
print(weights)
```

TORCHVISION

◆ PREPROCESSING SOLUTION

- **가중 무작위 추출 : 각 클래스별 동일 개수 구성된 배치 데이터 생성**

```
# 샘플러 : 배치 로딩시 자동으로 클래스에 대한 균일 분포를 갖는 배치 생성 샘플러
wsampler = WeightedRandomSampler( weights, len(weights) )
trainloader_wrs = DataLoader(trainset, batch_size=6, sampler=wsampler)


for epoch in range(5):
  for data in trainloader_wrs:
    print(data[1])
```

TORCHVISION

◆ PREPROCESSING SOLUTION

- **가중 손실함수 : 데이터가 적은 클래스에 큰 가중치 부여, 업데이트 균형**

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

# 10개 데이터
num_ins = [40, 45, 30, 62, 70, 153, 395, 46, 75, 194]

# 10개 데이터에 대한 가중치 설정 및 텐서화
weights = [1-(x/sum(num_ins)) for x in num_ins]
class_weights = torch.FloatTensor(weights).to(device)

# 가중치 nn.CrossEntropyLoss 설정
criterion = nn.CrossEntropyLoss(weight=class_weights)
```