DATA
SCIENCE LAB

Week 2
Lecture

23.07.19 / 9기 조의현

0. INTRO

Pytorch Lightning : ~ 2 weeks!

WEEK 2 : PytorchLightning & Tensorboard (23/07/19)

WEEK 3 : Train with Huggingface Libaray (23/07/26)

DATA
SCIENCE LAB

MNIST AutoEncoder with PytorchLightning

23.07.19 / 9기 조의현

1. Lightning

Pytorch Lightning

깃허브 링크

https://github.com/Lightning-AI/lightning

딥러닝 모델 구축 & 학습을 보조하는 최신 라이브러리!

1. Lightning

기존 Torch 모델 구축 & 학습과정

1. 데이터 (.json, .csv)를 Dataset 형태로 불러와 Dataloader 배치로 분할

- Dataset : Pd.Dataframe과 비슷, 하지만 데이터와 레이블을 Dict. 형태로 저장
- Dataloader : Dataset 형태의 데이터를 배치 (Batch) 단위로 분할해 학습

```
print(datasets)
```

```
DatasetDict({
    train: Dataset({
        features: ['laws_service_id', 'fact', 'laws_service'],
        num_rows: 161192
    })
})
```

```
datasets['train'][0]
```

```
{'laws_service_id': 32,
 'fact': '피고인은 2018. 8. 9. 23:33경 술을 마신 상태로 경산시 사동에 있
놓고 잠을 자고 있다는 112 신고를 받고 현장에 출동한 경산경찰서 C파출소
3회에 걸쳐 음주측정기에 입김을 불어 넣는 방법으로 음주측정에 응할 것을
 'laws_service': '도로교통법 제148조의2 제2항,도로교통법 제44조 제2항'}
```

1. Lightning

데이터 받아오는 과정

```
import pytorch_lightning as pl
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from pytorch_lightning.loggers import TensorBoardLogger
import os

# MNIST 데이터 받아오기
transform=transforms.Compose([transforms.ToTensor()])
train_dataset = datasets.MNIST('./', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST('./', train=False, download=True, transform=transform)
```

```
#1: LSUN #2: LSUNClass #3: ImageFolder #4:
#5: FakeData #6: CocoCaptions #7: CocoDetec
#10: EMNIST #11: FashionMNIST #12: QMNIST #
#15: StanfordCars #16: STL10 #17: SUN397 #1
#20: SEMEION #21: Omniglot #22: SBU #23: Fl
#25: Flowers102 #26: VOCSegmentation #27: V
#30: Caltech101 #31: Caltech256 #32: CelebA
#35: VisionDataset #36: USPS #37: Kinetics
#40: Places365 #41: Kitti #42: INaturalist
#45: KittiFlow #46: Sintel #47: FlyingChair
#50: Food101 #51: DTD #52: FER2013 #53: GTS
#55: OxfordIIITPet #56: PCAM #57: Country21
#60: RenderedSST2 #61: Kitti2012Stereo #62:
#65: CREStereo #66: FallingThingsStereo #67
#70: ETH3DStereo
```

Torchvision에서 기본 제공하는 MNIST 데이터를 받아옵니다.

- Train/test pre-labeling + transform 함수로 바로 전환 가능!

1. Lightning

Train/Test/Val

```python
from torch.utils.data import random_split

total_length = len(train_dataset)

train_ratio = 0.8
train_length = int(total_length * train_ratio)
val_length = total_length - train_length

train_dataset, val_dataset = random_split(train_dataset, [train_length, val_length])
```

random_split : Dataset 데이터에 대한 train_test_split 역할 수행

```python
print(f'# of train dataset : {len(train_dataset)}')
print(f'# of test dataset : {len(test_dataset)}')
```

```
# of train dataset : 60000
# of test dataset : 10000
```

```python
print(f'# of train dataset : {len(train_dataset)}')
print(f'# of val dataset : {len(val_dataset)}')
```

```
# of train dataset : 48000
# of val dataset : 12000
```

1. Lightning

Dataloader 배치 분할

딥러닝 모델은 batch_size에 따라 데이터를 분할한 후,
해당 batch만큼 학습한 이후 loss function을 update합니다.

ex) batch_size = 128,
48,000개의 데이터를 ÷ 128 = 375개의 batch로 나눈 후,
각 batch에 있는 128개의 데이터 학습 후 loss update.

기존 : 데이터 학습 후 바로 loss update -> batch_size = 1

※ 만일 375.3333와 같이 남는 데이터가 있다면 묶어서 따로 학습합니다.

```
train_params = {'batch_size': TRAIN_BATCH_SIZE,
                'shuffle': True,
                'num_workers': 8,
                'pin_memory' :True
                }

test_params = {'batch_size': VALID_BATCH_SIZE,
                'num_workers': 8,
                'pin_memory' :True
                }

val_params = {'batch_size': VALID_BATCH_SIZE,
                'num_workers': 8,
                'pin_memory' :True
                }

train_loader = DataLoader(train_dataset, **train_params)
test_loader = DataLoader(test_dataset, **test_params)
val_loader = DataLoader(val_dataset, **val_params)
```

1. Lightning

기존 Torch 모델 구축 & 학습과정

2. 딥러닝 모델을 nn.module 함수를 통해 구축
    - 전체적인 모델 구조 + forward 함수 포함
3. Loss function과 Optimizer 정의
4. Training step과 backpropagation을 for loop으로 정의
    - Validation step이 있다면 추가적으로 for loop 확장
5. Test step도 동일하기 for loop구조로 evaluation 진행

1. Lightning

기존 Torch 모델 구축 & 학습과정

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(16, 32)
        self.fc2 = nn.Linear(32, 2)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```python
model = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# 모델 학습
for epoch in range(10):  # 전체 데이터셋에 대해 10번 반복
    for data, labels in dataloader:
        # 모델 예측
        outputs = model(data)
        # 손실 계산
        loss = criterion(outputs, labels)
        # 역전파 및 최적화
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

1. Lightning

단점 & 한계점

1. 모델 구축 & loss function & optimizer & training step 구현이 따로따로임
2. 모델 최적화의 불편함 -> parameter도 각자 정리
3. 복잡한 코드 구성

➔ 하나의 class 내에서 모델 구축 + loss & optim을 정의하고,

➔ Parameter는 Dictionary 형태로 입력

➔ GPU, TPU 연동 및 Tensorboard와 같은 프레임워크와 연동

=> Pytorch Lightning!

2. Autoencoder

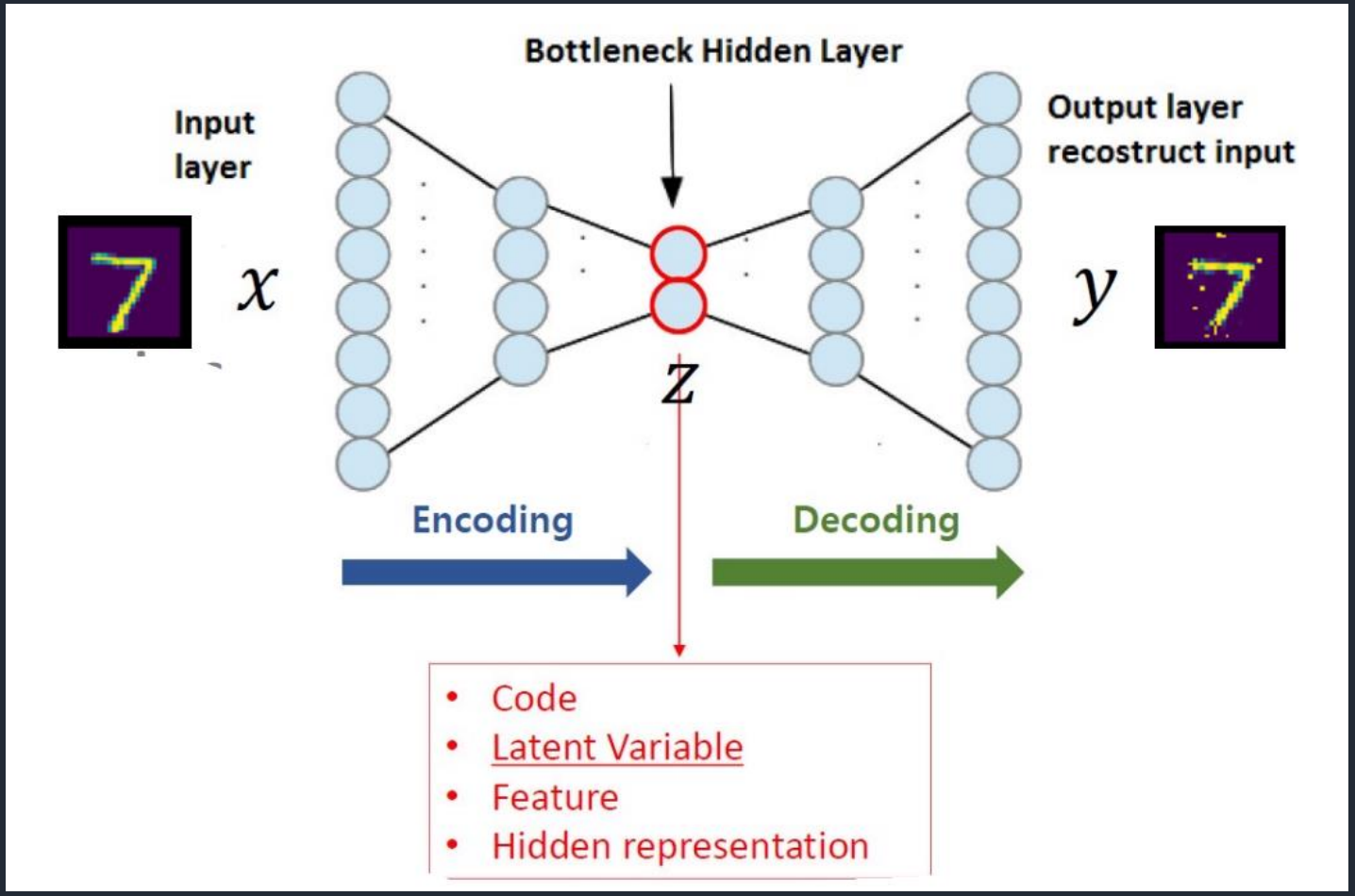
Pytorch Lightning

Pytorch Lightning은 nn.module로 대표되는 pytorch 딥러닝 프레임워크 구축 과정을 보조하는, 딥러닝 모델 개발을 쉽고 유연하게 만들어주는 보조 라이브러리

- Minimal running speed
- Models become hardware agonistic
- Easier to reproduce
- Able to Distribute

2. Autoencoder

Autoencoder



2. Autoencoder

Autoencoder

오토인코더란, 입력 벡터를 encoder를 통해 차원을 축소하고, 축소한 벡터를 decoder를 통해 재구성해 벡터의 특성을 학습하는 비지도학습 알고리즘이다.

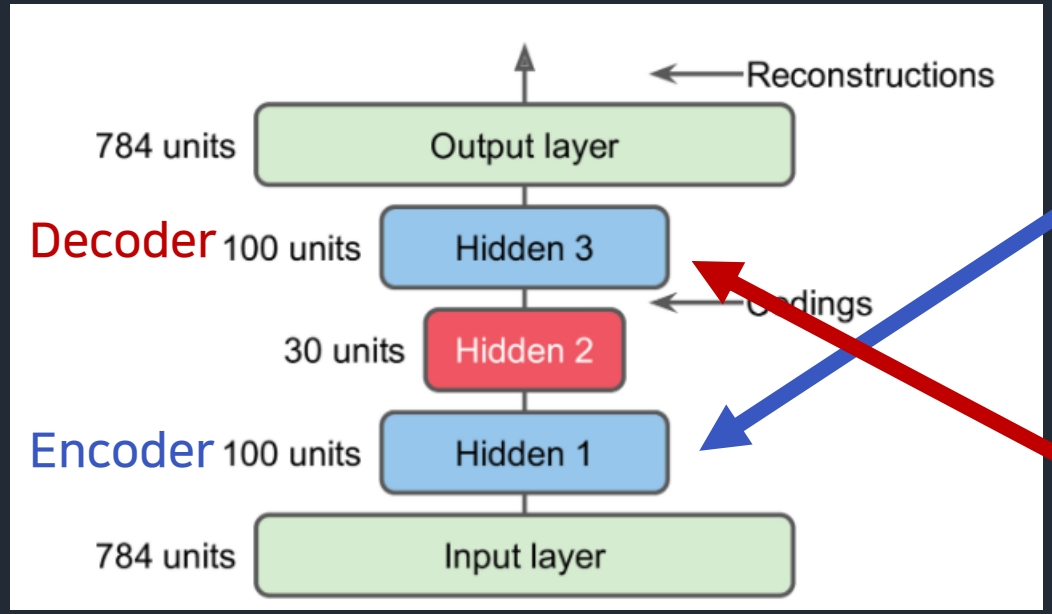
Encoder : 데이터 (X)을 latent vector (Z)으로 reduce (Unsupervised)

Decoder : latent vector (Z)을 다시 데이터 (X)으로 복원 (Supervised)

➔ 데이터 압축, 복원, 차원의 저주 해소, 노이즈 제거에 사용!

2. Autoencoder

Stacked Autoencoder




```
self.encoder = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size = 3, padding=1),
    nn.LeakyReLU(),
    nn.Conv2d(32, 64, kernel_size=3, stride = 2, padding=1),
    nn.LeakyReLU(),
    nn.Conv2d(64, 64, kernel_size = 3, stride = 2, padding = 1),
    nn.LeakyReLU(),
    nn.Flatten(),
    nn.Linear(64 * 7 * 7, 2),
    nn.LeakyReLU()
)

self.decoder = nn.Sequential(
    nn.Linear(2, 64 * 7 * 7),
    nn.Unflatten(1, (64, 7, 7)),
    nn.LeakyReLU(),
    nn.ConvTranspose2d(64, 64, kernel_size=3, stride = 2, padding = 1, output_padding = 1),
    nn.LeakyReLU(),
    nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, padding=1, output_padding=1),
    nn.LeakyReLU(),
    nn.ConvTranspose2d(32, 1, kernel_size=3, padding=1)
)
```

2. Autoencoder

Encoder

$\therefore\ O = \frac{(W - K + 2P)}{S} + 1$ (W = height, K = filter size, P = padding size, S = stride size)

```
nn.Conv2d(1, 32, kernel_size = 3, padding=1),
nn.LeakyReLU(),
nn.Conv2d(32, 64, kernel_size=3, stride = 2, padding=1),
nn.LeakyReLU(),
nn.Conv2d(64, 64, kernel_size = 3, stride = 2, padding = 1),
nn.LeakyReLU(),
nn.Flatten(),
nn.Linear(64 * 7 * 7, 2),
nn.LeakyReLU()
```

1. 32개의 3 x 3 커널 (28 – 3 + 1 * 2)/1 + 1 = 28 -> 이미지 크기 변화 X (28, 28)
2. 64개의 3 x 3 커널 (28 – 3 + 1 * 2)/2 + 1 = 14 -> 이미지 절반으로 감소 (14, 14)
3. 2번과 동일 = 7 -> 이미지 절반으로 감소 (7, 7)
4. 2차원 이미지 1차원으로 변환 (64 * 7 * 7 = 3136)

2. Autoencoder

Decoder

$\therefore W = (O - 1) * S - 2P + K + o$ (o = output padding size)

```
    nn.Linear(2, 64 * 7 * 7),
    nn.Unflatten(1, (64, 7, 7)),
    nn.LeakyReLU(),
    nn.ConvTranspose2d(64, 64, kernel_size=3, stride = 2, padding = 1, output_padding = 1),
    nn.LeakyReLU(),
    nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, padding=1, output_padding=1),
    nn.LeakyReLU(),
    nn.ConvTranspose2d(32, 1, kernel_size=3, padding=1)
)
```

1. 1차원 이미지 2차원으로 변환 (3136 -> 64 * 7 * 7)
2. 64개의 3x3 커널 (7 – 1) * 2 – 2 * 1 + 3 + 1 = 14 -> 이미지 두배로 증가 (14 x 14)
3. 2번 동일 -> 이미지 두배로 증가 (28 x 28)
4. 32개의 3x3 커널 -> 이미지 크기 동일 (28 x 28)

➔ 이미지 크기 & 특성 복원!

DATA SCIENCE LAB

발표자 조의현
E-mail: cuihyun12@yonsei.ac.kr