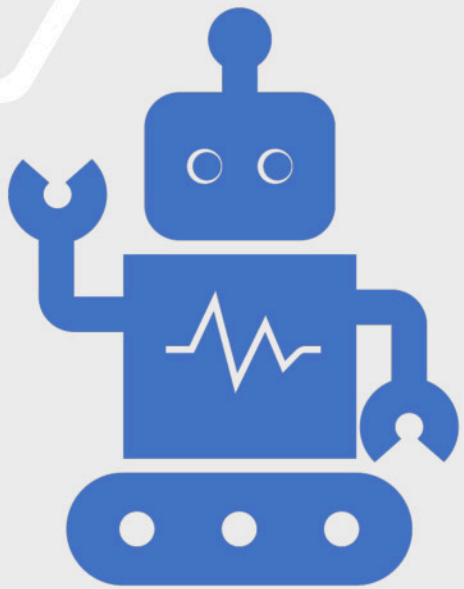
2020.08.24. ~ 2020.08.28.

경상대학교 407동 202호

Day 05

초보자를 위한 빅 데이터/ 머신러닝



Index

오늘의 할 일 요약

- AutoEncoder 를 사용한 비지도학습을 익힌다.
- 합성곱 신경망(CNN)을 익힌다.
- 옷 데이터를 사용해 AutoEncoder로 이미지 생성을 익힌다.

AutoEncoder

0. Hyperparameter 설정

```
leaning_rate = 1e-4
batch_size = 64
epochs = 20
```

learning_rate : 최적화 함수의 학습 율
batch_size: 한번에 몇개의 데이터를 학습 하는지 결정
epochs : 전체 데이터를 몇 회 반복 학습 하는지 결정

1. Python Package import

```
from tensorflow import keras
import tensorflow as tf
from tensorflow.keras import layers as l

import numpy as np
from tqdm import tqdm
import matplotlib.pyplot as plt
from time import time
```

```
import tensorflow as tf
# “tensorflow” 라는 이름의 모듈을 불러와 “tf” 라는 약칭으로 사용함.

from tensorflow import keras
# “tensorflow” 라는 모듈에서 ”keras” 라는 모듈을 불러옴
```

2. Load Dataset

```
mnist = keras.datasets.fashion_mnist
(train_x, train_y), (test_x, test_y) = mnist.load_data()

print("Train image shape :", train_x.shape)
print("Test image shape :", test_x.shape)
print("Train label shape :",train_y.shape)
print("Test label shape :",test_y.shape)
```

```
Train image shape : (60000, 28, 28)
Test image shape : (10000, 28, 28)
Train label shape : (60000,)
Test label shape : (10000,)
```

```
keras.datasets.mnist.load_data()
# keras 모듈 내부의 mnist dataset을 불러옴.
# 불러온 데이터는 train_{x,y}, test_{x,y}로 4가지 Numpy 변수로 가지고 옴.
# Numpy 변수의 경우 *.shape를 통해 데이터 형태를 받을 수 있음
```

3. Custom Dataset Preprocessing : 1/4 Start

```
# validation / training Split Done

train_dataset = tf.data.Dataset.from_tensor_slices((train_x, train_y))
valid_dataset = tf.data.Dataset.from_tensor_slices((val_x, val_y))
test_dataset = tf.data.Dataset.from_tensor_slices((test_x, test_y))
```

커스텀 데이터 로더를 생성 tf.data.Dataset이

```
*_dataset = tf.data.Dataset.from_tensor_slices((*_x, *_y))
```

tf.data.Dataset.from_tensor_slices 함수를 사용해 데이터,라벨을 입력하여 커스텀 데이터 셋을 생성

3. Custom Dataset Preprocessing : 2/4

```
def pre_processing(image,label):
    image = tf.cast(image,dtype=tf.float32) # cast to fp32
    image /=255.0 # normalization
    return image,label

train_dataset = train_dataset.map(pre_processing)
valid_dataset = valid_dataset.map(pre_processing)
test_dataset = test_dataset.map(pre_processing)
```

custom dataset 과 전처리 함수를 연결 하는 과정
Dataset.map 을 사용해 전처리 함수와 데이터 셋을 연결 할 수 있음
이미지 자체로 다뤄야 하기 때문에 reshape를 제외

3. Custom Dataset Preprocessing : 3/4

```
train_dataset = train_dataset.shuffle(train_data_size).batch(batch_size)
test_dataset = test_dataset.batch(batch_size)
valid_dataset = valid_dataset.batch(batch_size)
```

Dataset 설정을 진행

```
train_dataset = train_dataset.shuffle(train_data_size).batch(batch_size)
# Dataset.shuffle(버퍼사이즈)를 사용해 데이터를 순서를 임의로 만듬
# Dataset.batch(배치사이즈)를 사용해 데이터 순서를 몇 개씩 가져 올지 설정
```

Shuffle, Batch둘 다 dataset을 반환함

3. Custom Dataset Preprocessing : 4/4 End

```
print(train_dataset.element_spec)
print(test_dataset.element_spec)
print(valid_dataset.element_spec)
```

Dataset 설정을 진행

shuffle, batch둘다 dataset을 반환함, dataset.element_spec 으로 출력되는 데이터 형태를 확인 할 수 있음

4. Model Build : Encoder 정의

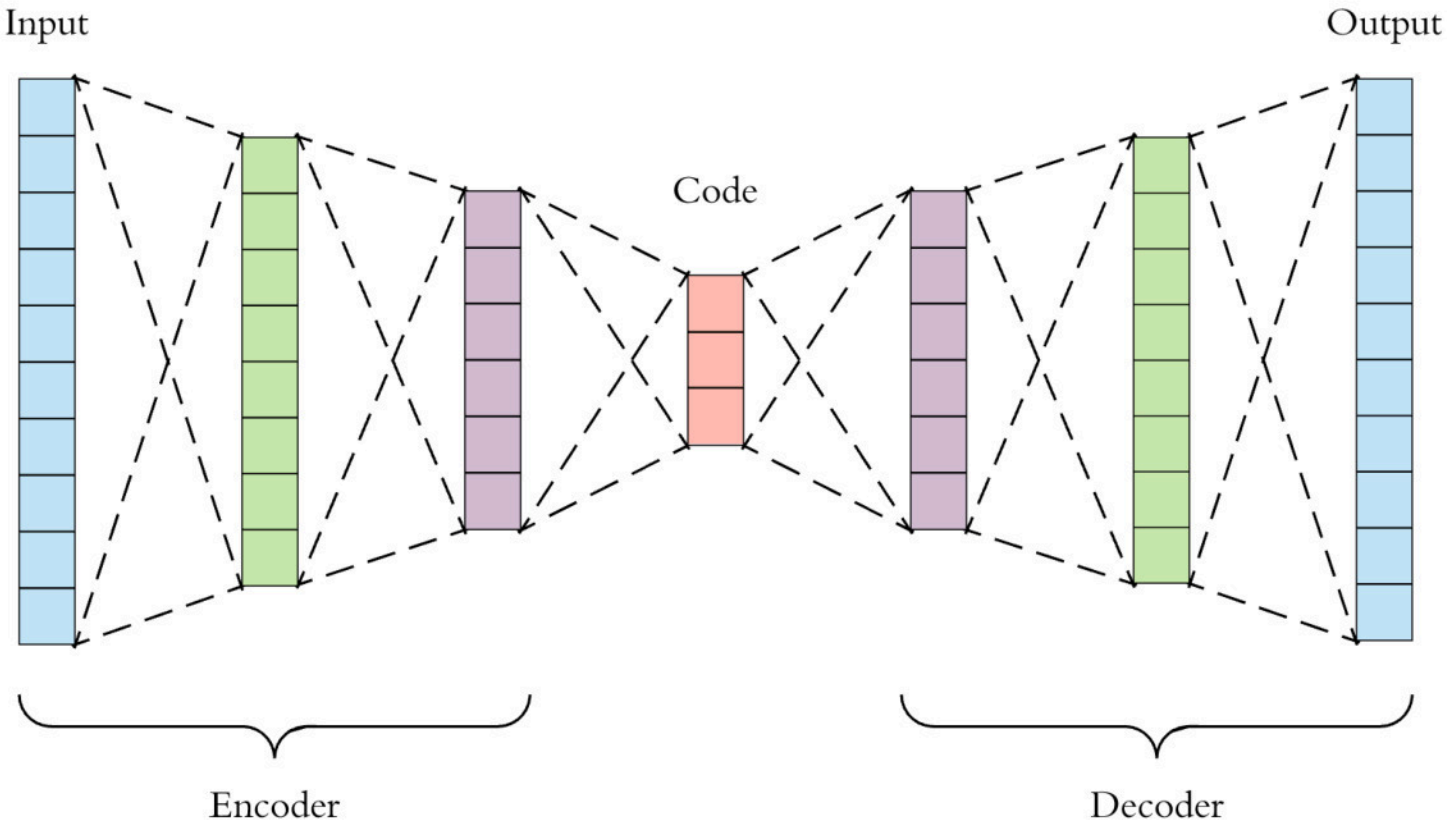
```
layers = [
    # Encoder
    l.Conv2D(filters=32, kernel_size=3, strides=1, padding='same'),
    l.ReLU(),
    l.MaxPool2D(pool_size=(2,2)),

    l.Conv2D(filters=64, kernel_size=3, strides=1, padding='same'),
    l.ReLU(),
    l.MaxPool2D(pool_size=(2,2)),

    l.Conv2D(filters=128, kernel_size=3, strides=1, padding='same'),
    l.ReLU(),
    # 이어짐
```

AutoEncoder의 Ecoder 파트 정의

4. Model Build : AutoEncoder



AutoEncoder로 출력 값을 입력 값의 근사로 하는 함수를 학습하는 비지도 학습이다.
입력을 Encoder로 압축하고 압축된 유닛(Code)를 Decoder로 복원함으로써 유닛이 입력한 데이터의 특징을 학습하게 됨.

4. Model Build : Decoder 정의

```
    # Decoder 전 페이지에서 계속
    l.Conv2DTranspose(filters=64, kernel_size=4, strides=2, padding='same'),
    l.ReLU(),
    l.Conv2DTranspose(filters=32, kernel_size=4, strides=2, padding='same'),
    l.ReLU(),
    # Interpolation
    l.Conv2D(filters=16, kernel_size=3, strides=1, padding='same'), l.ReLU(),
    l.Conv2D(filters=1, kernel_size=3, strides=1, padding='same'),
    l.Activation(tf.nn.sigmoid)
]
```

AutoEncoder의 Decoder 파트 정의

4. Model Build : AutoEncoder Model 생성

```
model = keras.Sequential(layers)
model.build(input_shape=(None,28,28,1))
loss_funcion = keras.losses.MeanAbsoluteError()
optimizer = keras.optimizers.Adam(
            learning_rate=leaning_rate)
train_loss =  keras.metrics.Mean(name='train_loss')
test_loss = keras.metrics.Mean(name='test_loss')
model.summary()
```

5. Logging using TensorBoard

```
summary_writer = tf.summary.create_file_writer(
                    logdir='tensorboard/logs{}'.format(time()),
                    filename_suffix='AutoEncoder')
```

학습 과정 중 출력되는 loss, 정확도, 입출력 데이터 등의 시각화하여 볼 수 있는 함수
tf.summary.create_file_writer()를 통해 summary_writer를 생성, 이를 통해
데이터를 입력 할 수 있음

시각화 가능한 데이터는 다음과 같음

- 스칼라
- 모델 그래프 형태
- 2D, 3D 데이터
- 히스토그램 등

6. Train step build

```
def train_step(images):
    with tf.GradientTape() as tape:
        pred = model(images,training=True)
        loss =loss_funcion(images,pred)

    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients,model.trainable_variables))
    train_loss(loss)
    return pred
```

학습을 위한 함수

6. Test step build

```
def test_step(images):
    pred = model(images, training=False)
    loss = loss_funcion(images, pred)

    test_loss(loss)
    return pred
```

검증을 위한 함수

7. Training

```
from tqdm.notebook import tqdm

for images in test_dataset.take(1):
    sample_images = images

for epoch in range(epochs):
    TRAIN(+ LOGGING)
    TEST(+ LOGGING)
```

```
for images in test_dataset.take(1):
    sample_images = image
```

를 통해 전 과정에 동일한 샘플 데이터를 하나 받아 둔다. 데이터 셋의 take(개수) 함수를 사용하면 원하는 개수 (단위는 배치)의 데이터를 받아 올 수 있다.

7. Training(*TRAIN*)

```
pbar = tqdm(train_dataset,total=(train_data_size//batch_size)+1)
for i, images in enumerate(pbar):
    train_pred = train_step(images)
    pbar.set_description('[%d/%d] Loss: %0.4f'% (epoch+1, epochs,
                                                 train_loss.result()))

    if i % 10 == 0:
        LOGGING
```

tqdm()은 반복의 진행상황을 출력한다.
enumerate(pbar)는 각 반복의 반환 값이 {반복 횟수, 데이터}로 나오게 한다.

if i % 10 == 0 # 학습 과정 중 매 10회 마다 발생한 데이터를 tensorboard로 넘겨 시각화 함
매번 log를 남기게 하면 속도가 느려지는 문제가 발생하기 때문에 횟수를 지정 해야 함

7. Training(*LOGGING*)

```
sample_pred = test_step(sample_images)
itr = optimizer.iterations.numpy()
with summary_writer.as_default():
    tf.summary.scalar('Train/loss(MAE)',train_loss.result(), step=itr)
    tf.summary.image('Train/Image',train_pred, step=itr)
    train_loss.reset_states()

    tf.summary.scalar('Sample/loss(MAE)',test_loss.result(), step=itr)
    tf.summary.image('Sample/Image',sample_pred, step=itr)
    test_loss.reset_states()
```

앞서 생성한 샘플 데이터를 사용해 샘플 이미지를 하나 출력, 학습과 샘플 이미지를 Tensorboard로 시각화 하고 학습, 샘플 이미지의 손실 값(loss)도 Tensorboard를 통해 그래프로 시각화

tf.summary.{scalar,image}를 통해 스칼라 값과 이미지를 TensorBoard로 보낼 수 있으며, 'Sample/loss(MAE)' 와 같은 이름을 설정해 각 값을 구분 할 수 있음

8. Test(*TEST + LOGGING*)

```
for i,images in enumerate(test_dataset):
    test_pred = test_step(images)
    if i % 3 == 0:
        with summary_writer.as_default():
            tf.summary.scalar('Test/loss(MAE)',test_loss.result(),step=opt
            imizer.iterations.numpy())
            tf.summary.image('Test/Image',test_pred,step=optimizer.iterati
                ons.numpy())
            test_loss.reset_states()
```

모든 테스트 데이터 셋을 사용한 테스트

9. SAVE & LOAD using h5 format

```
# SAVE
filepath='autoencoder.h5'
model.save(filepath)

# LOAD
model = None
model = keras.models.load_model(filepath)
```

```
def make_grid(images):
    images = []
    for i in range(0,64,8):
        im_list = [pred[index] for index in range(i,i+8)]
        images.append(tf.concat(im_list,1))
    return tf.concat(images,0)
```

Load 된 모델의 테스트를 위한 함수

9. SAVE & LOAD using h5 format

```
pred = model.predict(sample_images)

image = make_grid(pred)
tf.keras.preprocessing.image.save_img('pred_image.png',image)

image = plt.imread('pred_image.png')
plt.imshow(image,cmap=plt.cm.binary_r)
plt.show()
```

불러온 모델에 샘플 데이터를 입력하여 시각화 한다.
make_grid를 통해 출력된 64개 (1배치) 이미지를 8*8 배열로 만들어
tf.keras.preprocessing.image.save_img를 통해 'pred_image.png' 이름으로
저장한다.

저장된 이미지는 matplotlib를 사용해 시각화

Project

25

Project 목적

저해상도 이미지 이미지를 고해상도 이미지로 업스케일링 하기

[Hint]

1. 전처리 함수에서 저해상도 이미지 생성하기
2. 손실 함수(loss) 변경
3. 머신러닝 모델 수정 (입력 크기를 모든 레이어에서 같게 만들기)
4. Train,Test step 입력 인자 변경