Autoencoder

9장 오토인코더

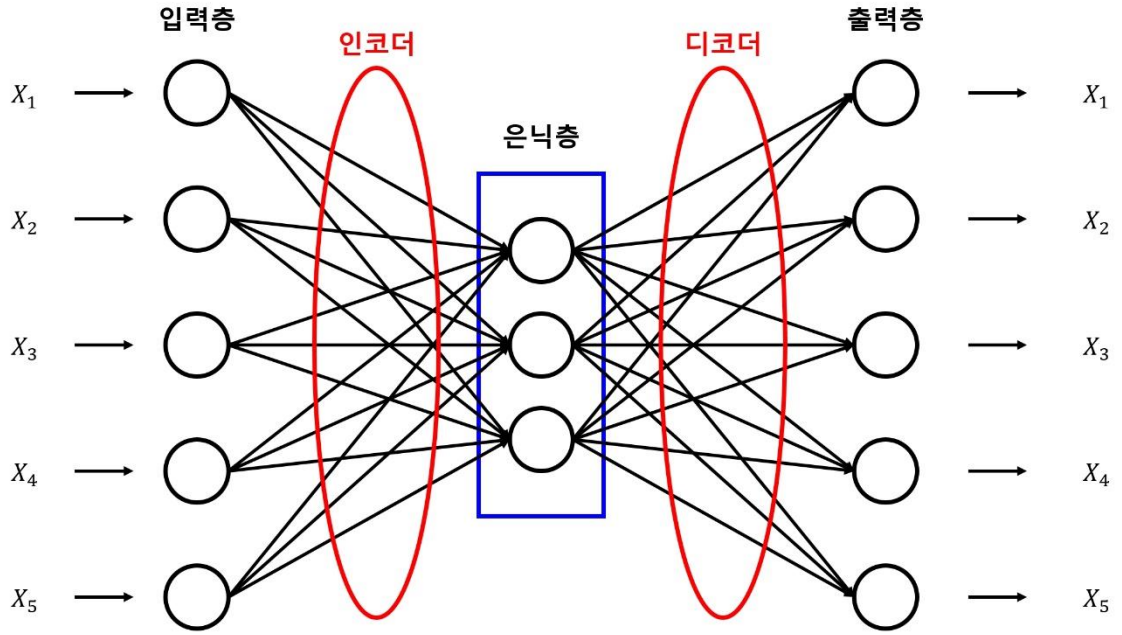
2020.08.07금 2h

오토인코더 개요

- **오토인코더(AutoEncoder)**
  - 출력값을 입력값의 근사로 하는 함수를 학습하는 비지도 학습
    - **오토인코더는 자기 자신을 재생성하는 네트워크**
      - **데이터의 숨겨진 구조를 발견하는 것이 학습목표**
    - **새로운 머신러닝의 범주로 분류하기도**
      - **자신 지도 학습(Self supervised learning)**
  - 출력이 입력과 동일하다는 점이 상이
    - **입력층으로 들어온 데이터를 인코더를 통해 은닉층으로 내보냄**
    - **은닉층의 데이터를 디코더를 통해 출력층으로 내보냄**
    - **만들어진 출력값을 입력값과 비슷해지도록 만드는 가중치를 찾아내는 것**
- **일종의 파일 압축과 유사**
  - 오토인코더는 손실 압축
- **적대적 생성 모델(Generative Adversarial Network 이하 GAN)**
  - 랜덤하게 생성된 변수를 잠재변수처럼 활용해서 새로운 이미지를 생성

오토인코더 구성

- **오토인코더는 크게 3가지 부분으로 구성**
  - 잠재 변수(은닉층)를 기준으로 하나의 대칭구조
    - **잠재 변수(Latent Vector)를 중심**
      - **일차원 벡터**
    - **입력에 가까운 부분을 인코더(Encoder)**
      - **입력에서 잠재 변수를 생성, CNN에서 특징 추출기와 비슷**
    - **출력에 가까운 부분을 디코더(Decoder)**
      - **잠재 변수를 출력**



9장 오토인코더

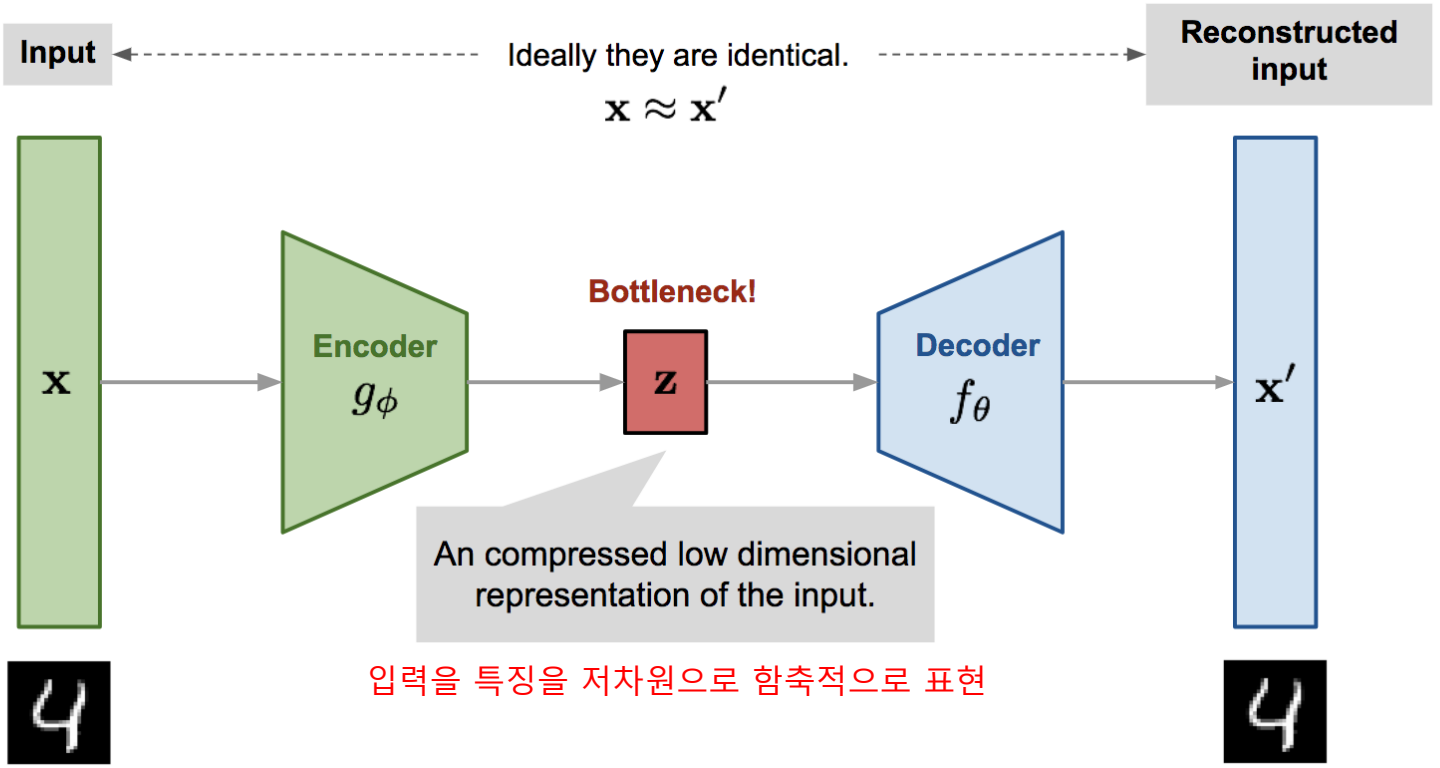
1 MNIST 손글씨 오토인코더에 적용

1h

파일

- ch9_1_auto_encoder.ipynb

MNIST 손글씨 오토인코더 적용



MNIST 손글씨 문제

- **Dense 층으로 MNIST 손글씨 오코인토더 구현**

▾ (1) 모듈 설치 및 데이터세트 확인

- 데이터는 (train_X, train_Y), (test_X, test_Y)처럼 훈련 데이터와 테스트 데이터의 튜플 쌍으로 불러 올 수 있습니다.
- 데이터를 로드한 후에 train_X와 test_X를 255.0으로 나눠서 픽셀 정규화를 하게 됩니다.
- 데이터가 잘 불러와졌는지 시각화를 통해 확인합니다.

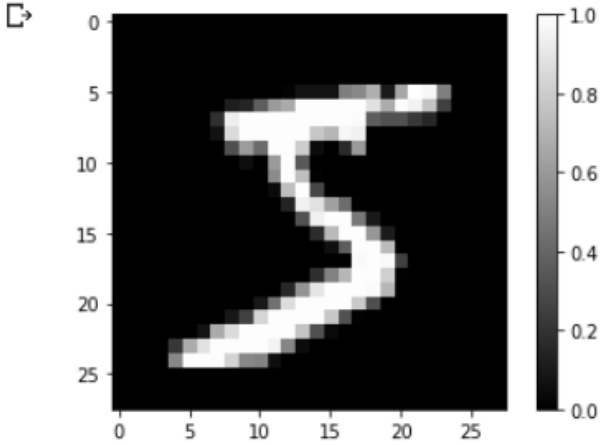
[2]
```
# 텐서플로 2 버전 선택
try:
    # %tensorflow_version only exists in Colab.
    %tensorflow_version 2.x
except Exception:
    pass
import tensorflow as tf
import numpy as np
import pandas as pd
import tensorflow_hub as hub
import matplotlib.pyplot as plt
import cv2
```

[3]
```
(train_X, train_Y), (test_X, test_Y) = tf.keras.datasets.mnist.load_data()
print(train_X.shape, train_Y.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
(60000, 28, 28) (60000,)
```

첫 이미지 확인

```
[4]  1 train_X = train_X / 255.0
     2 test_X = test_X / 255.0
     3
     4 plt.imshow(train_X[0].reshape(28, 28), cmap='gray')
     5 plt.colorbar()
     6 plt.show()
     7
     8 print(train_Y[0])
```



5

- MNIST는 Fashion MNIST처럼 가로와 세로가 각각 28픽셀인 흑백 이미지를 입력으로 하고, 0~9까지의 숫자를 출력으로 합니다. (5장과 6장 참조)

Dense 오토인코더 모델 정의

- **Flatten 레이어를 사용하는 대신**
  - train_X와 test_X의 차원을 직접 reshape() 함수로 변환
    - **입력과 출력의 형태가 같아야 하기 때문**
- **dense와 dense_2의 레이어**
  - 뉴런의 수가 같아서 대칭을 이루며
  - 각각 인코더와 디코더의 역할
  - desne_1는 잠재 변수로 뉴런의 수가 적은 것을 확인

```
[5]  1 train_X = train_X.reshape(-1, 28 * 28)
     2 test_X = test_X.reshape(-1, 28 * 28)
     3 print(train_X.shape, train_Y.shape)
```

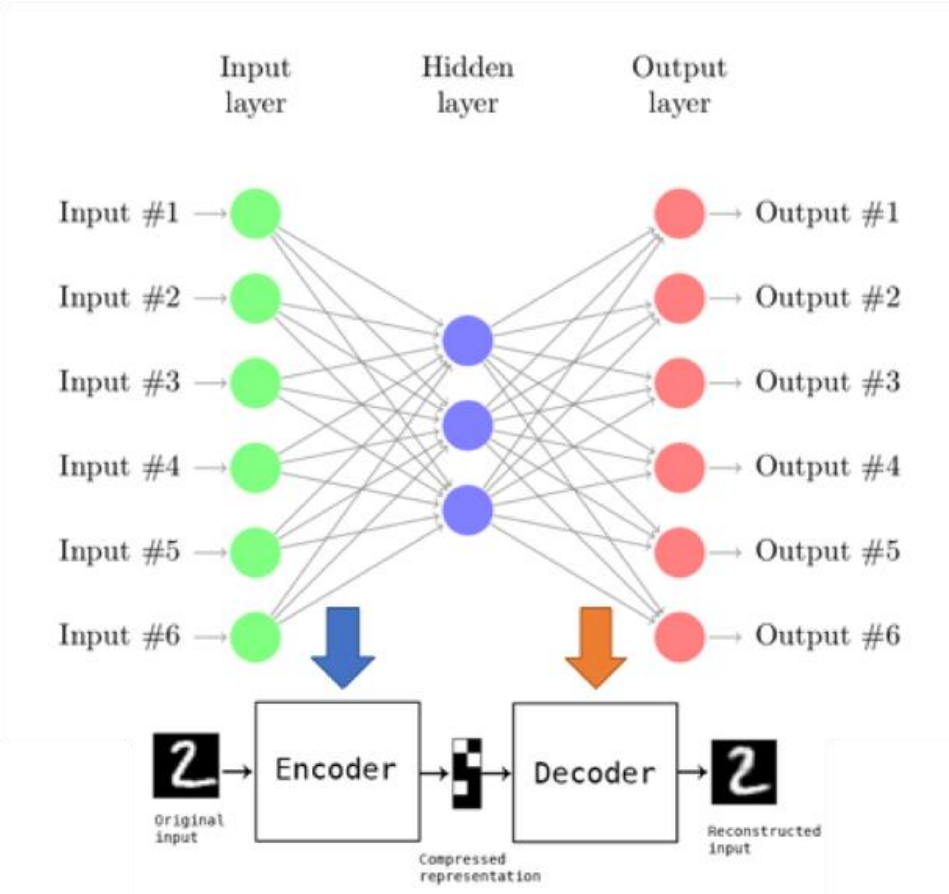
```
(60000, 784) (60000,)
```

```
[6]  1 model = tf.keras.Sequential([
     2     tf.keras.layers.Dense(784, activation='relu', input_shape=(784,)),
     3     tf.keras.layers.Dense(64, activation='relu'),
     4     tf.keras.layers.Dense(784, activation='sigmoid')
     5 ])
     6
     7 model.compile(optimizer=tf.optimizers.Adam(), loss='mse')
     8 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 784)               615440
_________________________________________________________________
dense_1 (Dense)              (None, 64)                50240
_________________________________________________________________
dense_2 (Dense)              (None, 784)               50960
=================================================================
Total params: 716,640
Trainable params: 716,640
Non-trainable params: 0
_________________________________________________________________
```

Dense 오토인코더 모델 구조

- **입력, 출력층 모두 Dense**
  - 28 x 28 = 784



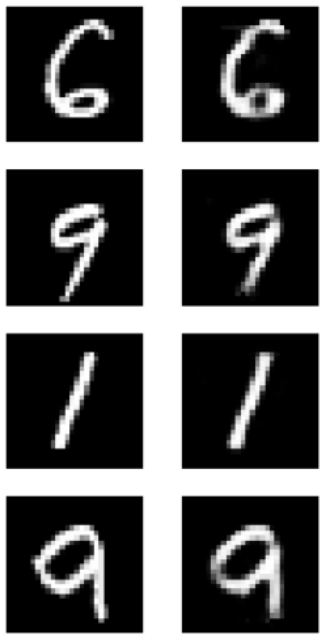
Dense 오토 인코더 모델 학습

```
1 model.fit(train_X, train_X, epochs=10, batch_size=256)
```

```
Epoch 1/10
235/235 [==============================] - 1s 3ms/step - loss: 0.0504
Epoch 2/10
235/235 [==============================] - 1s 3ms/step - loss: 0.0174
Epoch 3/10
235/235 [==============================] - 1s 3ms/step - loss: 0.0122
Epoch 4/10
235/235 [==============================] - 1s 3ms/step - loss: 0.0101
Epoch 5/10
235/235 [==============================] - 1s 3ms/step - loss: 0.0090
Epoch 6/10
235/235 [==============================] - 1s 3ms/step - loss: 0.0082
Epoch 7/10
235/235 [==============================] - 1s 4ms/step - loss: 0.0076
Epoch 8/10
235/235 [==============================] - 1s 3ms/step - loss: 0.0072
Epoch 9/10
235/235 [==============================] - 1s 4ms/step - loss: 0.0069
Epoch 10/10
235/235 [==============================] - 1s 3ms/step - loss: 0.0066
<tensorflow.python.keras.callbacks.History at 0x7f0880421128>
```

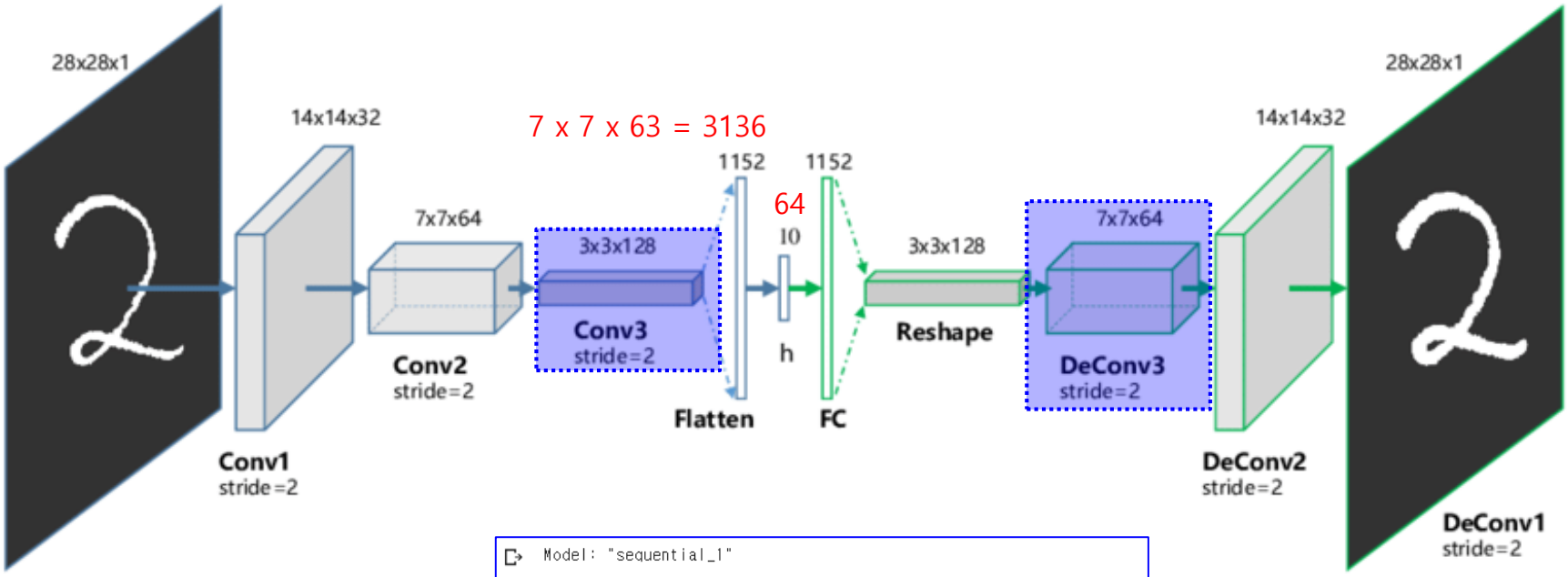
모델 시각화

```
import random

plt.figure(figsize=(4,8))
for c in range(4):
    plt.subplot(4, 2, c*2+1)
    rand_index = random.randint(0, test_X.shape[0])
    plt.imshow(test_X[rand_index].reshape(28, 28), cmap='gray')
    plt.axis('off')

    plt.subplot(4, 2, c*2+2)
    img = model.predict(np.expand_dims(test_X[rand_index], axis=0))
    plt.imshow(img.reshape(28, 28), cmap='gray')
    plt.axis('off')

plt.show()

model.evaluate(test_X, test_X)
```



```
313/313 [==============================] - 1s 2ms/step - loss: 0.0063
0.006336142309010029
```

CNN으로 MNIST 손글씨 오코인토더



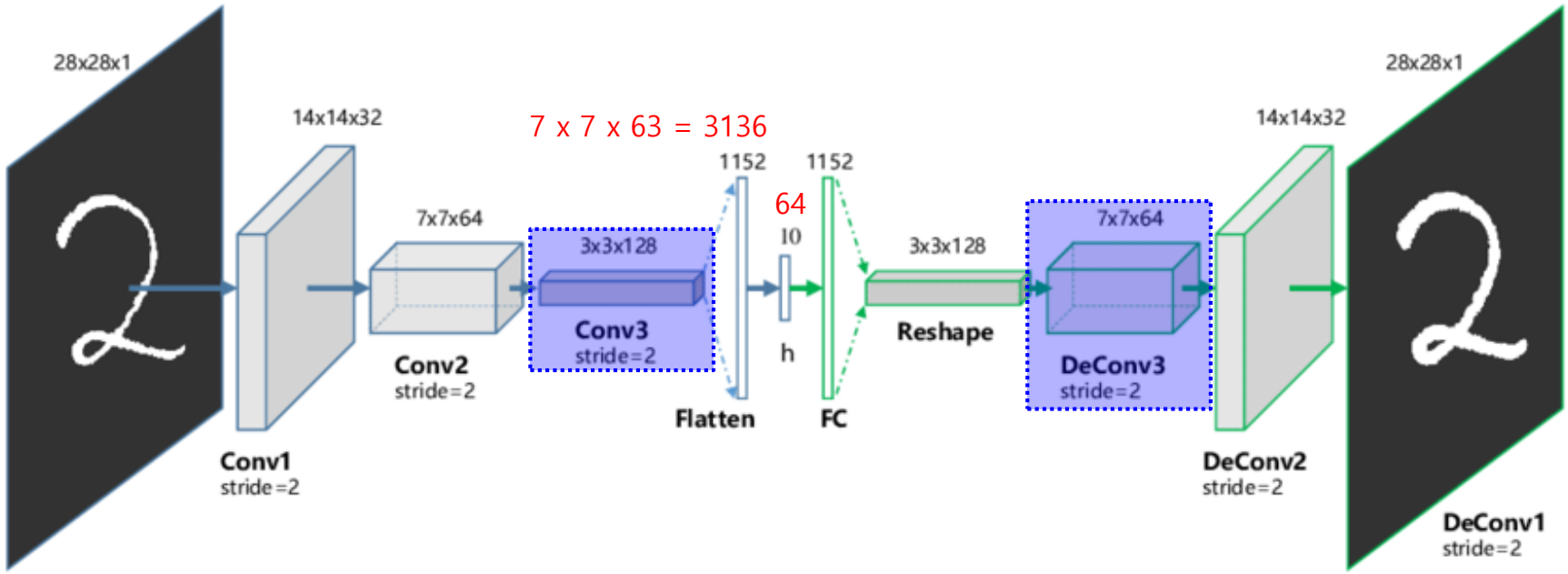

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 14, 14, 32)        160
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 7, 7, 64)          8256
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                200768
_________________________________________________________________
dense_4 (Dense)              (None, 3136)              203840
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 14, 14, 32)        8224
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 1)         129
=================================================================
Total params: 421,377
Trainable params: 421,377
Non-trainable params: 0
```

CNN으로 MNIST 손글씨 오코인토더 구현

```
train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=2, strides=(2,2), activation='relu', i
nput_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(filters=64, kernel_size=2, strides=(2,2), activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(7*7*64, activation='relu'),
    tf.keras.layers.Reshape(target_shape=(7,7,64)),
    tf.keras.layers.Conv2DTranspose(filters=32, kernel_size=2, strides=(2,2), padding='sa
me', activation='relu'),
    tf.keras.layers.Conv2DTranspose(filters=1, kernel_size=2, strides=(2,2), padding='sam
e', activation='sigmoid')
])

model.compile(optimizer=tf.optimizers.Adam(), loss='ms
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 14, 14, 32)        160
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 7, 7, 64)          8256
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                200768
_________________________________________________________________
dense_4 (Dense)              (None, 3136)              203840
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 14, 14, 32)        8224
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 1)         129
=================================================================
Total params: 421,377
Trainable params: 421,377
Non-trainable params: 0
```

CNN으로 MNIST 손글씨 오코인토더 모델




```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=2, strides=(2,2), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(filters=64, kernel_size=2, strides=(2,2), activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(7*7*64, activation='relu'),
    tf.keras.layers.Reshape(target_shape=(7,7,64)),
    tf.keras.layers.Conv2DTranspose(filters=32, kernel_size=2, strides=(2,2), padding='same', activation='relu'),
.Conv2DTranspose(filters=1, kernel_size=2, strides=(2,2), padding='same', activation='sigmoid')
    tf.keras.layers
])
```

인코더와 잠재변수

- **Conv2D 레이어를 2개**
  - kernel_size=2, strides=(2,2)로 설정해서 풀링 레이어를 쓰는 것과 같은 효과
  - Conv2D를 통과할 때마다 50%씩 감소
  - 두번째 Conv2D를 통과하면 이미지의 크지는 7X7

```
tf.keras.layers.Conv2D(filters=32, kernel_size=2, strides=(2,2), activation='relu', input_shape=(28, 28, 1)),
tf.keras.layers.Conv2D(filters=64, kernel_size=2, strides=(2,2), activation='relu'),
```

- **잠재 변수 생성**
  - Flatten()을 통과
    - **3차원의 데이터를 1차원으로**
  - 64개 뉴런의 Dense 층
    - **Dense 오토인코더와 동일한 크기로 64개의 뉴런을 가지는 Dense 레이어를 배치**

```
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(64, activation='relu'),
```

디코더 생성

- **디코더 생성**
  - 디코더는 인코더와 대칭이 되도록 다시 구성
  - 잠재변수 레이어와 연결된 레이어는 7X7 이미지를 만들기 위해 64개의 채널만큼 가지고 있는 Conv2D레이어
  - 레이어와 뉴런 수를 동일하게 만들기 위해서 Dense레이어의 뉴런 수를 7*7*64로

```
tf.keras.layers.Dense(7*7*64, activation='relu')
```

  - 1차원 데이터를 3차원으로 바꿔주기 위해 64개의 채널만큼 Reshape레이어 사용

```
tf.keras.layers.Reshape(target_shape=(7,7,64))
```

  - 마지막으로 이어지는 2개의 레이어는 Conv2DTranspose
    - **Conv2D 레이어가 하는 일의 반대되는 계산으로 이해**
  - 필터의 개수가 1인 것은 흑백인 출력 이미지와 동일

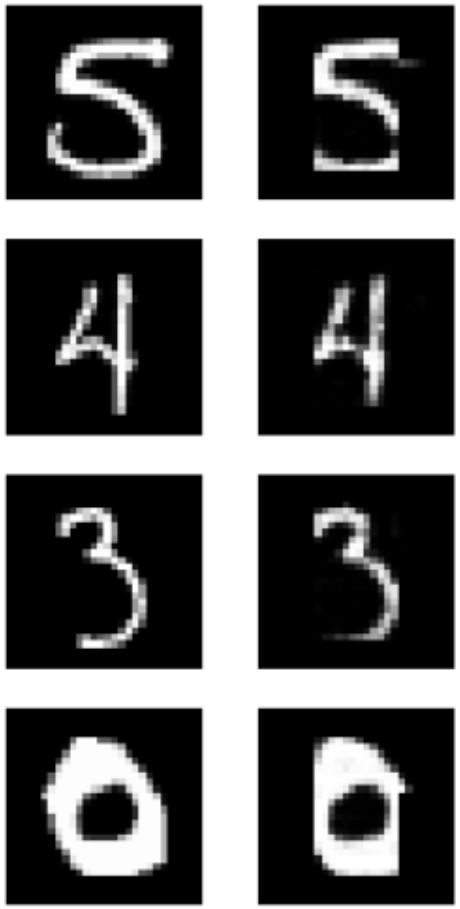
```
tf.keras.layers.Conv2DTranspose(filters=32, kernel_size=2, strides=(2,2), padding='same', activation='relu'),
tf.keras.layers.Conv2DTranspose(filters=1, kernel_size=2, strides=(2,2), padding='same', activation='sigmoid')
```

학습

```
[9]   1 model.fit(train_X, train_X, epochs=10, batch_size=256)
```

```
Epoch 1/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0763
Epoch 2/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0305
Epoch 3/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0224
Epoch 4/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0200
Epoch 5/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0190
Epoch 6/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0184
Epoch 7/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0179
Epoch 8/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0176
Epoch 9/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0174
Epoch 10/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0172
<tensorflow.python.keras.callbacks.History at 0x7f088027a5c0>
```

오토인코더의 이미지 재생성 및 모형 성능 평가



```
import random

plt.figure(figsize=(4,8))
for c in range(4):
    plt.subplot(4, 2, c*2+1)
    rand_index = random.randint(0, test_X.shape[0])
    plt.imshow(test_X[rand_index].reshape(28, 28), cmap='gray')
    plt.axis('off')

    plt.subplot(4, 2, c*2+2)
    img = model.predict(np.expand_dims(test_X[rand_index], axis=0
))
    plt.imshow(img.reshape(28, 28), cmap='gray')
    plt.axis('off')

plt.show()

model.evaluate(test_X, test_X)
```

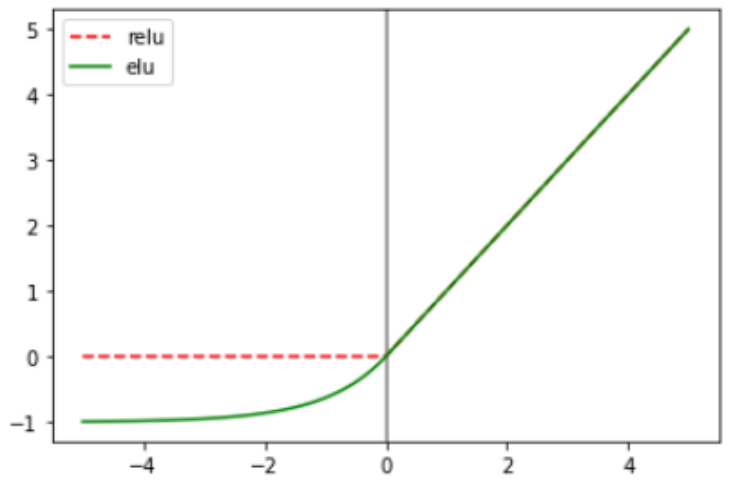
```
313/313 [==============================] - 1s 3ms/step - loss: 0.0188
0.018794177100062237
```

활성화함수(=activation) ELU 수정

- **대부분 relu를 사용**
  - relu는 양수와 0을 반환
    - **뉴런의 계산값 중 음수가 되는 결과가 많을 경우 뉴런의 출력은 무조건 0**
    - **출력은 다음 레이어의 가중치에 곱해지기 때문에 출력이 0이면 가중치의 효과를 모두 0**
  - 이러한 문제점을 해결하고자 elu 사용
- **ELU(Exponential Linear Units)**
  - elu는 0으로 수렴하지 않고 -1로 수렴
  - 2015년, Clevert et al에 의해 제안

$$f(x) = \begin{cases} x & if\ x > 0 \\ \alpha(exp(x) - 1) & if\ x \leq 0 \end{cases}$$

```
[10]  1 # 그림 9.7 출력 코드
      2 import math
      3
      4 x = np.arange(-5, 5, 0.01)
      5 relu = [0 if z < 0 else z for z in x]
      6 elu = [1.0 * (np.exp(z) - 1) if z < 0 else z for z in x]
      7
      8 # plt.axhline(0, color='gray')
      9 plt.axvline(0, color='gray')
     10 plt.plot(x, relu, 'r--', label='relu')
     11 plt.plot(x, elu, 'g-', label='elu')
     12 plt.legend()
     13 plt.show()
```



활성화함수 ELU 수정 모델

- **손실 값**
  - 1/3 감소
- **결과 그림**
  - relu와 다르게 이전의 각진 모습은 거의 찾아볼 수 없음
  - 노이즈를 제거하는 효과도 볼 수 있음

```
train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=2, strides=(2,2
), activation='elu', input_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(filters=64, kernel_size=2, strides=(2,2
), activation='elu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='elu'),
    tf.keras.layers.Dense(7*7*64, activation='elu'),
    tf.keras.layers.Reshape(target_shape=(7,7,64)),
    tf.keras.layers.Conv2DTranspose(filters=32, kernel_size=2, str
ides=(2,2), padding='same', activation='elu'),
    tf.keras.layers.Conv2DTranspose(filters=1, kernel_size=2, stri
des=(2,2), padding='same', activation='sigmoid')
])

model.compile(optimizer=tf.optimizers.Adam(), loss='mse')
model.summary()

model.fit(train_X, train_X, epochs=10, batch_size=256)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0052
0.005169130861759186
```

활성화함수 ELU 수정 모델 전 소스

```
train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=2, strides=(2,2), activation='elu', input_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(filters=64, kernel_size=2, strides=(2,2), activation='elu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='elu'),
    tf.keras.layers.Dense(7*7*64, activation='elu'),
    tf.keras.layers.Reshape(target_shape=(7,7,64)),
    tf.keras.layers.Conv2DTranspose(filters=32, kernel_size=2, strides=(2,2), padding='same', activation='elu'),
    tf.keras.layers.Conv2DTranspose(filters=1, kernel_size=2, strides=(2,2), padding='same', activation='sigmoid')
])

model.compile(optimizer=tf.optimizers.Adam(), loss='mse')
model.summary()

model.fit(train_X, train_X, epochs=10, batch_size=256)

import random

plt.figure(figsize=(4,8))
for c in range(4):
    plt.subplot(4, 2, c*2+1)
    rand_index = random.randint(0, test_X.shape[0])
    plt.imshow(test_X[rand_index].reshape(28, 28), cmap='gray')
    plt.axis('off')

    plt.subplot(4, 2, c*2+2)
    img = model.predict(np.expand_dims(test_X[rand_index], axis=0))
    plt.imshow(img.reshape(28, 28), cmap='gray')
    plt.axis('off')

plt.show()
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0052
0.005169130861759186
```

```
model.evaluate(test_X, test_X)
```

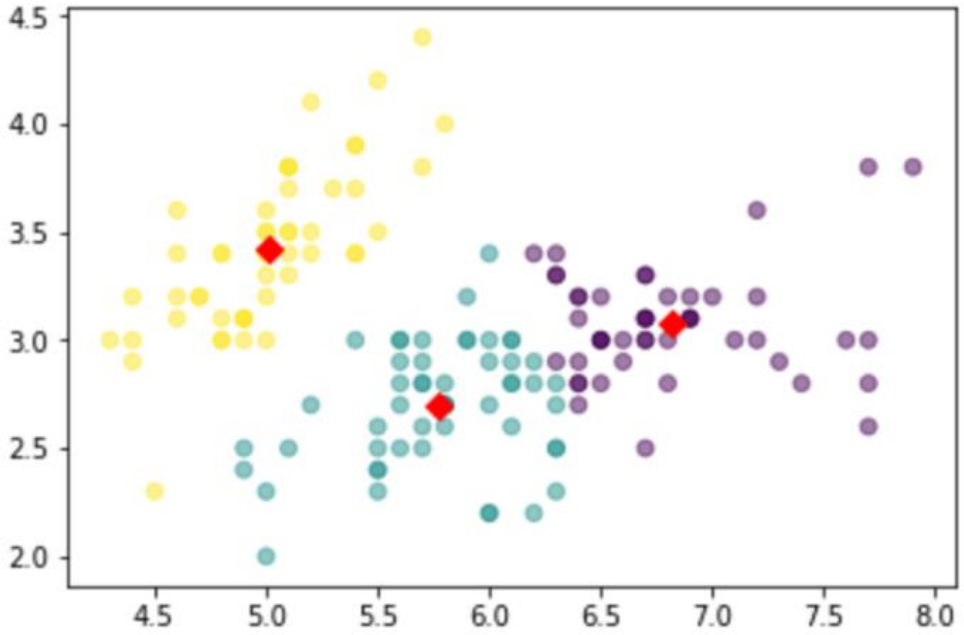
9장 오토인코더

3 클러스터링

2h

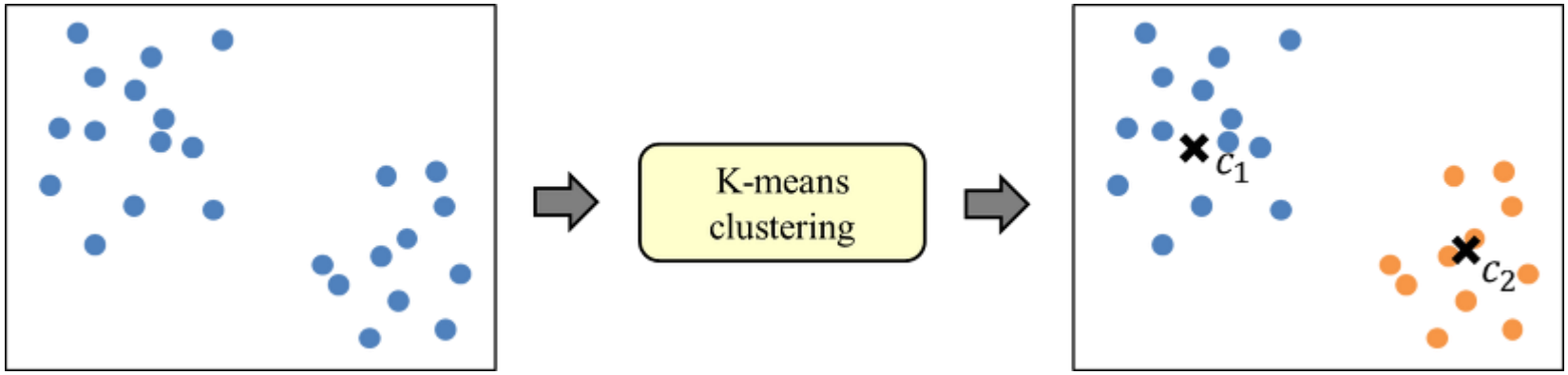
클러스터링 개요

- **대표적인 비지도학습 방법의 한 종류**
  - 비슷한 군집으로 나누는 것
  - 입력에 대한 정답이 없음
- **사례**
  - 사람의 얼굴 이미지를 몇 개의 집단으로 분류하는 것이 적절할까요?
  - 단편 소설의 장르를 몇 개로 구분해야 할까요?



K-평균 클러스터링

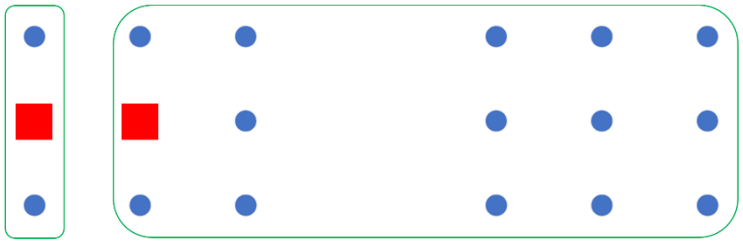
- **K-means Clustering**
  - 데이터를 입력 받아 이를 소수의 그룹으로 묶는 알고리즘
- **방법**
  - 1. 주어진 입력 중 K개의 클러스터 중심을 임의로 정한 다음
  - 2. 각 데이터와 K개의 중심과의 거리를 비교해서 가장 가까운 클러스터로 배당하고
  - 3. K개의 중심의 위치를 해당 클러스터로 옮긴 후, 이를 반복하는 알고리즘



KC 학습과정(1)

- **EM 알고리즘을 기반으로 작동**
  - EM 알고리즘은 크게 Expectation 스텝과 Maximization 스텝으로 나눔
    - **이를 수렴할 때까지 반복하는 방식**
- **군집 수 k를 2로**
  - 우선 군집의 중심(빨간색 점)을 랜덤 초기화

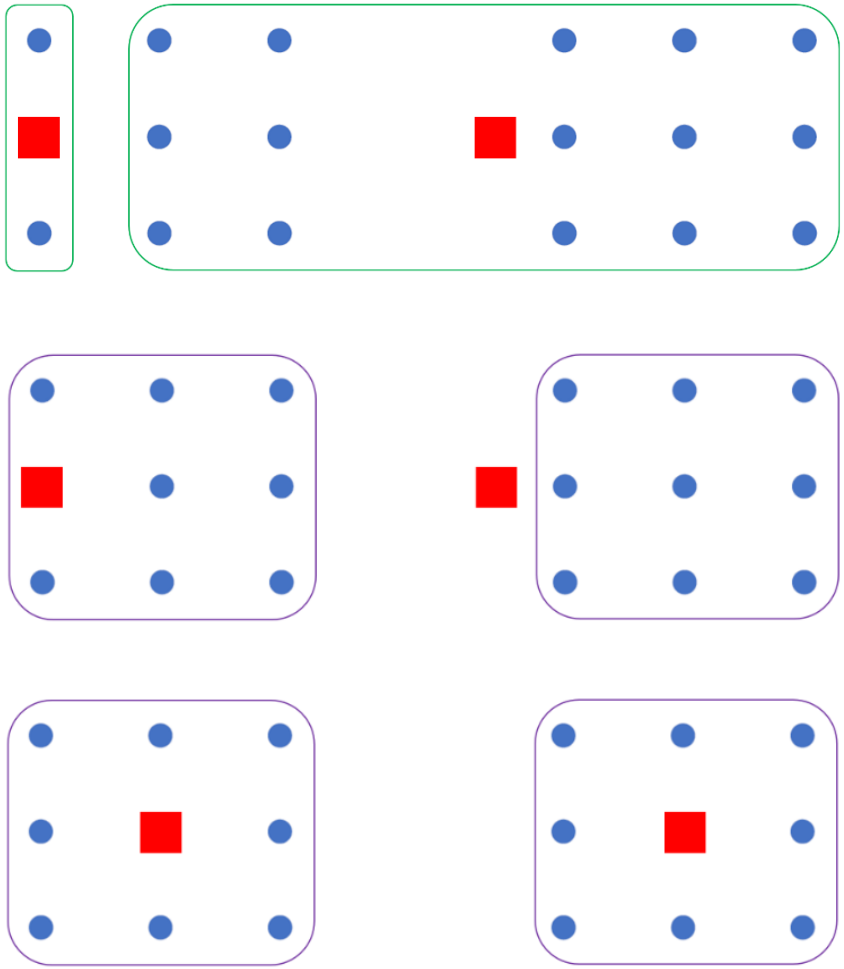
  - Expectation 스텝
    - **모든 개체들(파란색 점)을 가장 가까운 중심에 군집(녹색 박스)으로 할당**



https://ratsgo.github.io/machine%20learning/2017/04/19/KC/

KC 학습과정(2)

- **Maximization 스텝**
  - 중심을 군집 경계에 맞게 수정
- **다시 Expectation 스텝을 적용**
  - 모든 개체들을 가장 가까운 중심에 군집(보라색 박스)으로 할당
- **Maximization 스텝을 또 적용해 중심을 업데이트**
  - Expectation과 Maximization 스텝을 반복 적용해도 결과가 바뀌지 않거나(=해가 수렴)
  - 사용자가 정한 반복 수를 채우게 되면 학습이 종료



파일

- ch9_3_k-means_clustering.ipynb

K-meas 군집화 메모 코딩(1)

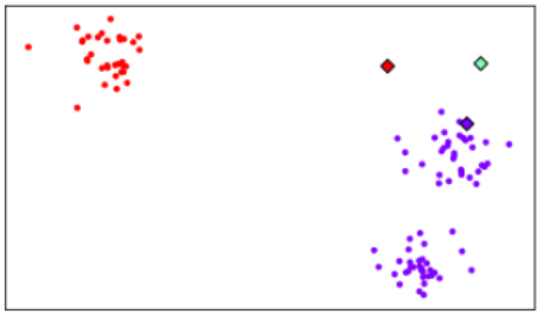
```
# 그림 9.8 출력 코드
import random
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

X, Y = make_blobs(random_state=6)
pred_Y = np.zeros_like(Y)

fig = plt.figure(figsize=(6,12))

centers = []
for c in range(3):
    center_X = random.random() * (max(X[:,0]) - min(X[:,0])) + min(X[:,0])
    center_Y = random.random() * (max(X[:,1]) - min(X[:,1])) + min(X[:,1])
    centers.append([center_X, center_Y])
centers = np.array(centers)
prev_centers = []
```

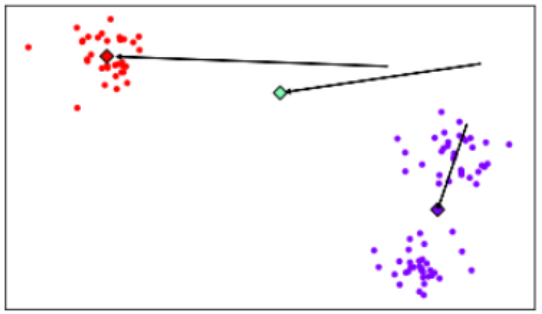
K-meas 군집화 메모 코딩(2)

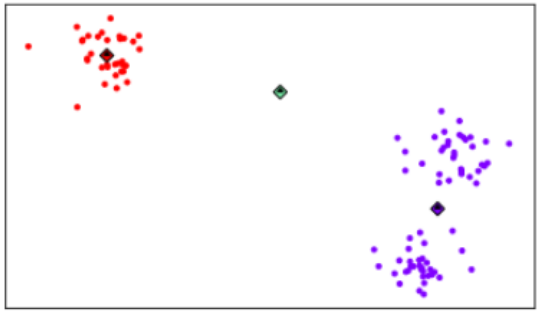
```
for t in range(3):
    for i in range(len(X)):
        min_dist = 9999
        center = -1
        for c in range(3):
            dist = ((X[i,0] - centers[c,0]) ** 2 + (X[i,1] - centers[c,1]) ** 2) ** 0.5
            if dist < min_dist:
                min_dist = dist
                center = c
        pred_Y[i] = center

    ax = fig.add_subplot(3, 1, t+1)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.scatter(X[:,0], X[:,1], marker='.', c=pred_Y, cmap='rainbow')
    ax.scatter(centers[:,0], centers[:,1], marker='D', c=range(3), cmap='rainbow', edgecolors=(0,0,0,1))

    if len(prev_centers) != 0:
        for c in range(3):
            ax.arrow(prev_centers[c,0], prev_centers[c,1], (centers[c,0]-
prev_centers[c,0])*0.95, (centers[c,1]-
prev_centers[c,1])*0.95, head_width=0.25, head_length=0.2, fc='k', ec='k')

    # update center
    prev_centers = np.copy(centers)
    for c in range(3):
        count = len(pred_Y[pred_Y == c])
        centers[c,0] = sum(X[pred_Y == c, 0]) / (count+1e-6)
        centers[c,1] = sum(X[pred_Y == c, 1]) / (count+1e-6)

plt.show()
```

K-means 군집화 과정

- **다이아몬드 3개**
  - 군집의 중심
  - 계산과정에서 중심이 이동







잠재 변수의 클러스터링

- **잠재 변수**
  - 데이터의 가장 압축된 표현
    - **손실 압축이지만 중요하지 않거나 세부 정보를 잃어버릴 확률이 높음**
  - 효과적으로 복원할 수 있도록 중요한 정보만 포함
- **잠재 변수를 이용해 클러스터링**
  - 바로 이전 코드(p341 예제 9-9)의 잠재 변수를 사용
    - **다음 모델의 4번째 층(첨자로 3)의 Dense 층의 출력을 사용**

```
train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=2, strides=(2,2), activation='elu',
                           input_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(filters=64, kernel_size=2, strides=(2,2), activation='elu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='elu'),
    tf.keras.layers.Dense(7*7*64, activation='elu'),
    tf.keras.layers.Reshape(target_shape=(7,7,64)),
    tf.keras.layers.Conv2DTranspose(filters=32, kernel_size=2, strides=(2,2),
                                    padding='same', activation='elu'),
    tf.keras.layers.Conv2DTranspose(filters=1, kernel_size=2, strides=(2,2),
                                    padding='same', activation='sigmoid')
])
```

예제 9-9(1)

```
import tensorflow as tf
import numpy as np
import pandas as pd
import tensorflow_hub as hub
import matplotlib.pyplot as plt

(train_X, train_Y), (test_X, test_Y) = tf.keras.datasets.mnist.load_data()
print(train_X.shape, train_Y.shape)



train_X = train_X / 255.0
test_X = test_X / 255.0

plt.imshow(train_X[0].reshape(28, 28), cmap='gray')
plt.colorbar()
plt.show()

print(train_Y[0])
```

예제 9-9(2)

```
train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=2, strides=(2,2), activation='elu',
                           input_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(filters=64, kernel_size=2, strides=(2,2), activation='elu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='elu'),
    tf.keras.layers.Dense(7*7*64, activation='elu'),
    tf.keras.layers.Reshape(target_shape=(7,7,64)),
    tf.keras.layers.Conv2DTranspose(filters=32, kernel_size=2, strides=(2,2),
                                    padding='same', activation='elu'),
    tf.keras.layers.Conv2DTranspose(filters=1, kernel_size=2, strides=(2,2),
                                    padding='same', activation='sigmoid')
])

model.compile(optimizer=tf.optimizers.Adam(), loss='mse')
model.fit(train_X, train_X, epochs=20, batch_size=256)
```

64차원의 MNIST 잠재 변수를 군집화

- **한 줄로 모델을 생성**
  - 훈련 데이터를 64차원의 잠재 변수로 생성

```
latent_vector_model = tf.keras.Model(inputs=model.input, outputs=model.layers[3].output)
```

```
[9]  1 # 9.11 잠재변수 추출 모델 정의 및 실행
     2 latent_vector_model = tf.keras.Model(inputs=model.input, outputs=model.layers[3].output)
     3 latent_vector = latent_vector_model.predict(train_X)
     4 print(latent_vector.shape)
     5 print(latent_vector[0])
```

```
(60000, 64)
[20.945282   14.54987    10.659601    3.7661126  -1.         19.961151
  5.2667227   6.481052   -0.99999994 -0.9999997   9.664746   19.717037
 -0.9999999  20.866062   -1.         -1.         -0.99999994 25.393711
 17.573387   -1.         -0.99999994 14.655176   17.32514    -0.9997823
 12.707492   -1.         16.836151   20.489754   10.260367   -1.
 -1.         -0.9999999  -0.99999946 -0.9999994   4.3679724   3.704837
 12.431805   28.099548    7.449969   14.744117    8.839078   -1.
 -0.9999996  15.726482   -0.99999964  9.310612   -0.6123714  18.305103
  2.991678   -0.99997157  9.960504   12.770576    3.5120966  30.884506
  7.594349   -0.99999994 -1.         16.374226   27.325478    6.6651373
 -0.99999994 18.528429   18.58517    10.224054  ]
```

사이킷런의 K-평균 클러스터링 알고리즘 사용

- **인자 n_clusters**
  - 나눌 군집 수
- **인자 n_init**
  - 초기 중심의 위치를 다르게 선택해서 n번 테스트한 가장 좋은 결과를 저장
    - **초기 중심위치 시도 횟수. 디폴트는 10이고 10개의 무작위 중심위치 목록 중 가장 좋은 값을 선택**
- **인자 random_state**
  - 난수 시드 번호

셀의 수행 시간 측정, Wall time: 실제 소요 시간, CPU times는 멀티 코어 사용시 모든 코어의 계산 시간을 합쳐서 표시

```
[10]  1 # 9.12 사이킷 런의 K-평균 클러스터링 알고리즘 사용
      2 %%time
      3 from sklearn.cluster import KMeans
      4
      5 kmeans = KMeans(n_clusters=10, n_init=10, random_state=42)
      6 kmeans.fit(latent_vector)

      CPU times: user 13.9 s, sys: 3.23 s, total: 17.2 s
      Wall time: 13.4 s
```

잠재 변수로 알고리즘을 학습

계산 결과

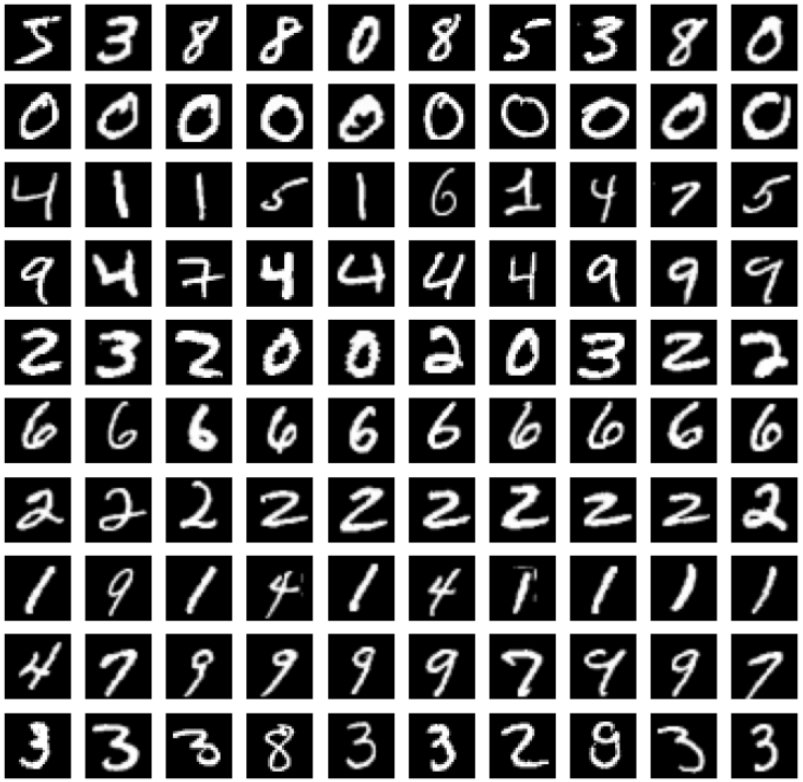
- **labels_**
  - 각 데이터가 0부터 9사이의 어떤 클러스터에 속하는지에 대한 정보가 저장
- **cluster_centers_**
  - 각 클러스터의 중심 좌표가 저장
    - **64차원이기 때문에 이 좌표가 각각 무엇을 의미하는지 알기 어려움**

```
[11]  1 # 9.13 계산 결과 확인
      2 print(kmeans.labels_)
      3 print(kmeans.cluster_centers_.shape)
      4 print(kmeans.cluster_centers_[0])
```

```
[0 1 2 ... 0 5 0]
(10, 64)
[19.006199   14.133522   19.437984   14.363924   -0.99999976 15.086731
 12.164242    4.609981   -0.9999996  -0.9999778  13.655131   18.674732
 -0.99999964 16.505865   -0.9999992  -0.9999998  -0.99999744 17.62626
 10.783649   -0.9999998  -0.9999987  12.441184   10.400169   -0.97614855
 15.397683   -0.9999997  13.849438   16.583647   15.790642   -0.9999998
 -0.99999964 -0.9999988  -0.9999981  -0.9999987   2.891747   12.03423
 22.559164   19.77699    12.093555   16.41429    15.831373   -0.99999994
 -0.9999984  18.39989    -0.99999815 13.873233   14.1866     14.655701
  2.3089197  -0.99936515 18.188889   12.895048    5.8652954  11.887303
 12.380737   -0.99999964 -0.9999993  19.815802   28.578535    5.3411593
 -0.99999934 15.889366   15.790815   10.368467  ]
```

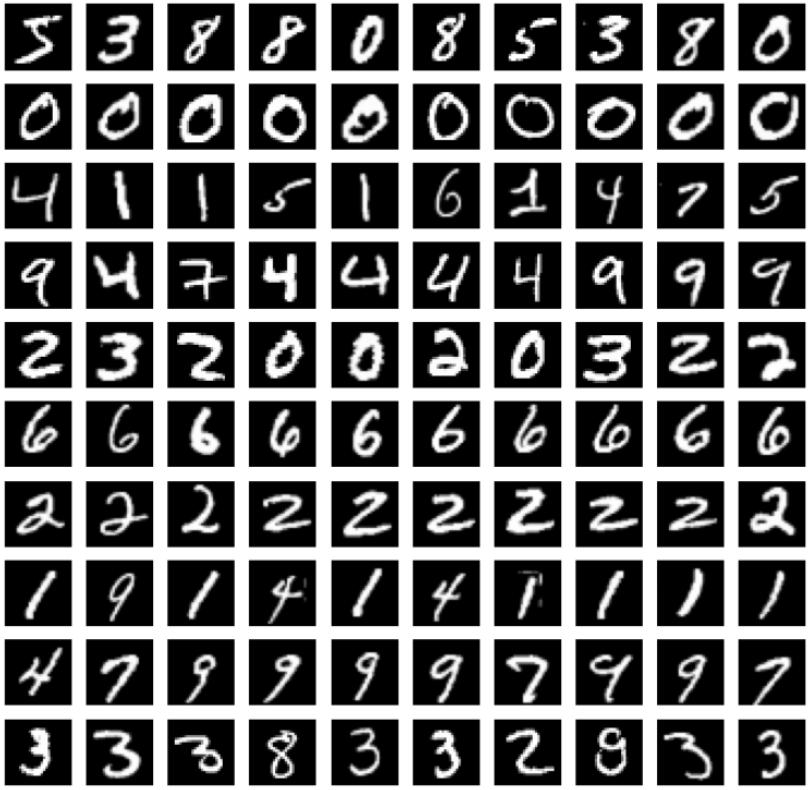
클러스터링 결과 출력

- **각 클러스터에 속하는 이미지가 어떤 것인지 출력**
  - 행은 클러스터 번호
    - **0~9**
- **각 행은 동일한 군집**
  - 숫자가 다르면서 같은 클러스터로 분류된 이미지들이 문제
    - **첫 군집이 틀린 것이 많음**
  - n_init를 늘려서 시도
- **여전히 클러스터링 결과를 시각화를 해야 문제**
  - 이를 시행하려면 2차원 또는 3차원의 잠재변수가 가진 자원을 축소
  - t-SNE가 해결책



클러스터링 결과 출력 소스

```
# 9.14 클러스터링 결과 출력
plt.figure(figsize=(12,12))

for i in range(10):
    images = train_X[kmeans.labels_ == i]
    for c in range(10):
        plt.subplot(10, 10, i*10+c+1)
        plt.imshow(images[c].reshape(28,28),
                   cmap='gray')
        plt.axis('off')

plt.show()
```



시각화 도구 t-SNE

- **군집화 및 강력한 시각화 도구**
  - 고차원의 데이터를 저차원(주로 2차원 혹은 3차원)의 시각화를 위한 데이터로 변환
    - **토론토 대학의 제프리 힌튼 교수 참여 연구**

K-평균 클러스터링: 클러스터를 계산하기 위
한 단위로 중심과 각 데이터의 거리를 계산

- **SNE: Stochastic Neighbor Embedding**
  - t-SNE는 각 데이터의 유사도를 정의
    - **원래 공간에서 유사도와 저차원 공간에서의 유사도가 비슷해지도록 학습**
  - 고차원과 저차원에서 확률 값을 구한 후
    - **저차원 확률 값이 고차원에 가까워지도록 학습**
  - 유사도는 확률적(Stochastic)으로 표현
    - **t는 t-분포**

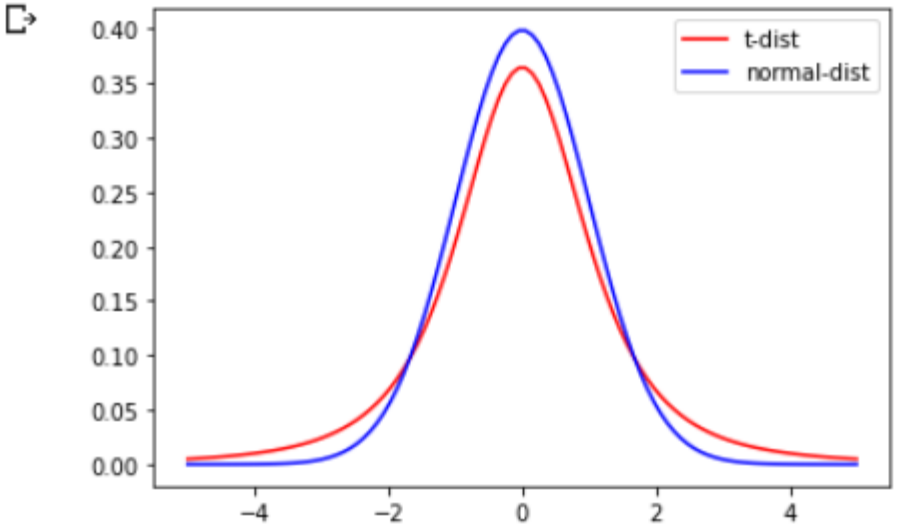
거리를 확률로 표현: 데이터를 중심으로 다른
데이터의 거리에 대한 t-분포의 확률로 치환

가까운 거리를 확률이 높아지고, 먼 거리는 확률이 낮아짐

t-분포와 정규분포의 모양 차이

- **t-분포**
  - 정규분포보다 더 낮고 꼬리가 좀 더 두꺼운 분포
- **거리를 확률로 표현**
  - 데이터 하나를 중심으로 다른 데이터를 거리에 대한 t-분포의 확률로 대체시키는 것

```
[13]  1 # 그림 9.9 출력 코드
      2 import scipy as sp
      3 t_dist = sp.stats.t(2.74)
      4 normal_dist = sp.stats.norm()
      5
      6 x = np.linspace(-5, 5, 100)
      7 t_pdf = t_dist.pdf(x)
      8 normal_pdf = normal_dist.pdf(x)
      9 plt.plot(x, t_pdf, c='red', label='t-dist')
     10 plt.plot(x, normal_pdf, c='blue', label='normal-dist')
     11 plt.legend()
     12 plt.show()
```



sklearn.manifold의 TSNE

- **sklearn.manifold**
  - 고차원의 공간에 분포하는 데이터를 쉽게 볼 수 있는 사물 형태로 분포되도록 학습시키는 다양체 학습(manifold learning)을 위한 여러 알고리즘 제공
  - 그 중하나가 t-SNE
- **TSNE 인자**
  - n_components: 저차원의 수를 의미, 2차원 공간이기 때문에 2
  - learning_rate: 학습률로 10에서 1000사이
  - perplexity: 알고리즘 계산에서 고려할 최근접 이웃의 숫자, 보통 5-50
  - random_state: 랜덤 초기화 숫자
- **메소드 fit_transform() 결과값을 반환**
  - 학습과 변환 과정을 동시에 진행

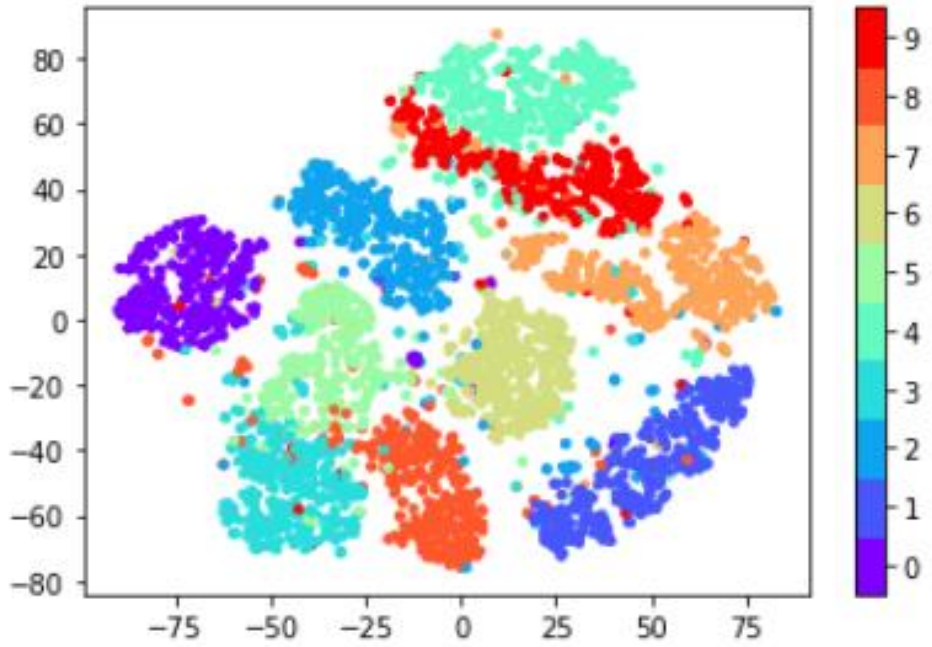
```
# 9.15 사이킷 런의 t-SNE 사용
%%time
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, learning_rate=100, perplexity=15, random_state=0)
tsne_vector = tsne.fit_transform(latent_vector[:5000])
```

결과는 위에서 지정한 2차원 좌표

계산 속도가 느려 앞 5,000개의 데이터만 학습

출력 이미지의 클러스터링

- **뭉친 것이 군집화 결과**
  - 같은 색상은 정답 분류에 의한 색상
    - **이미지 라벨에 따라 같은 숫자끼리 비교적 잘 뭉쳐있는 것을 확인**



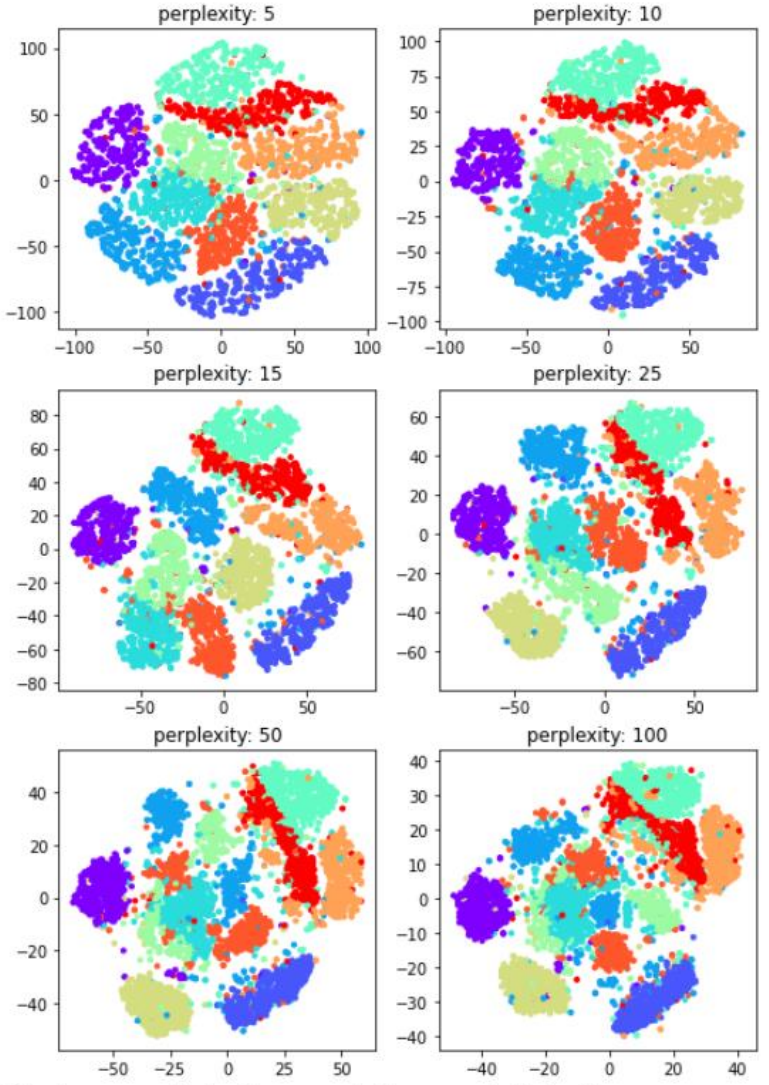

```
# 9.15 사이킷 런의 t-SNE 사용
%%time
from sklearn.manifold import TSNE
```

```
CPU times: user 1min 1s, sys: 46.5 ms, total: 1min 1s
Wall time: 32.9 s
```

```
tsne = TSNE(n_components=2, learning_rate=100, perplexity=15, random_state=0)
tsne_vector = tsne.fit_transform(latent_vector[:5000])

cmap = plt.get_cmap('rainbow', 10)
fig = plt.scatter(tsne_vector[:,0], tsne_vector[:,1], marker='.', c=train_Y[:5000], cmap=cmap)
cb = plt.colorbar(fig, ticks=range(10))
n_clusters = 10
tick_locs = (np.arange(n_clusters) + 0.5)*(n_clusters-1)/n_clusters
cb.set_ticks(tick_locs)
cb.set_ticklabels(range(10))

plt.show()
```

다양한 perplexity 설정

- 인자 perplexity는 다른 하이퍼패러미터처럼 여러 번의 실험이 필요
  - Perplexity가 높아질수록 뭉치는 클러스터도 있지만, 뒤섞이는 클러스터도 보이는 것으로 볼 때 최적의 값을 찾기 위한 여러 시도가 필요
- K-means와 다르게 클러스터링 수는 필요 없음




```
# 9.16 다양한 perplexity 인수에 대한 t-SNE 결과
%%time

perplexities = [5, 10, 15, 25, 50, 100]
plt.figure(figsize=(8,12))

for c in range(6):
    tsne = TSNE(n_components=2, learning_rate=100,
            perplexity=perplexities[c], random_state=0)
    tsne_vector = tsne.fit_transform(latent_vector[:5000])

    plt.subplot(3, 2, c+1)
    plt.scatter(tsne_vector[:,0], tsne_vector[:,1],
           marker='.', c=train_Y[:5000], cmap='rainbow')
    plt.title('perplexity: {0}'.format(perplexities[c]))

plt.show()
```

```
CPU times: user 7min 16s, sys: 1.19 s, total: 7min 17s
Wall time: 3min 52s
```

t-SNE 클러스터 위에 MNIST 이미지 표시 구현

```
# 9.17 t-SNE 클러스터 위에 MNIST 이미지 표시
from matplotlib.offsetbox import TextArea, DrawingArea,
       OffsetImage, AnnotationBbox

plt.figure(figsize=(16,16))

tsne = TSNE(n_components=2, learning_rate=100,
            perplexity=15, random_state=0)
tsne_vector = tsne.fit_transform(latent_vector[:5000])

ax = plt.subplot(1, 1, 1)
ax.scatter(tsne_vector[:,0], tsne_vector[:,1], marker='.',
           c=train_Y[:5000], cmap='rainbow')
for i in range(200):
    imagebox = OffsetImage(train_X[i].reshape(28,28))
    ab = AnnotationBbox(imagebox, (tsne_vector[i,0],
          tsne_vector[i,1]), frameon=False, pad=0.0)
    ax.add_artist(ab)

ax.set_xticks([])
ax.set_yticks([])
plt.show()
```

이미지나 텍스트 등 주석을 그래프 위에 표시하기 위한 주석 상자를 그리는 함수

t-SNE 클러스터 위에 MNIST 이미지 표시 결과