Week4

CH17 - 오토인코더와 GAN을 사용한 표현 학습과 생성적 학습

오토인코더:

어떤 지도 없이도 (**잠재 표현** or **코딩**이라 부르는) 입력 데이터의 밀집 표현을 학습할 수 있는 ANN

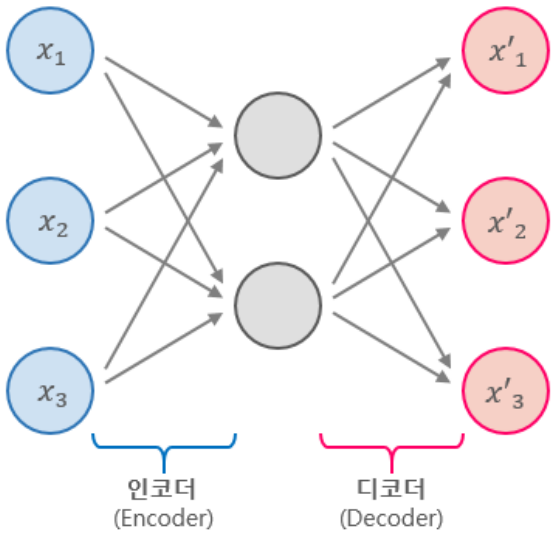
- 차원 축소 - 시각화에 유용
- 특성 추출기처럼 작동 ~ 심층 신경망의 비지도 사전훈련에 사용
- 훈련 데이터와 매우 비슷한 새로운 데이터 생성 → **생성 모델**

GAN(상대적 적대 신경망):

- 생성자 - 훈련 데이터와 비슷하게 보이는 데이터 생성
- 판별자 - 가짜 데이터와 진짜 데이터를 구별

17.1 효율적인 데이터 표현

- 패턴 찾기
- 입력 → 내부표현 : **인코더**(= 인지 네트워크)
- 내부 표현 → 출력 : **디코더**(= 생성 네트워크)



출력층의 뉴런 수 = 입력 개수

- 출력: 입력을 재구성한다 → **재구성**
- 재구성 $\neq$ 입력 → 비용 함수: **재구성 손실**

17.2 과소완전 선형 오토인코더로 PCA 수행하기

3D 데이터 셋에 PCA를 적용해 2D에 투영

```
np.random.seed(42)
tf.random.set_seed(42)

encoder = keras.models.Sequential([keras.layers.Dense(2, input_shape=[3])])
decoder = keras.models.Sequential([keras.layers.Dense(3, input_shape=[2])])
autoencoder = keras.models.Sequential([encoder, decoder])

autoencoder.compile(loss="mse", optimizer=keras.optimizers.SGD(learning_rate=0.1))
```

- 인코더, 디코더 두 개 컴포넌트로 구성
- 출력 개수 = 입력 개수
- 단순 PCA - 활성화 함수 사용 X / 비용함수 = MSE

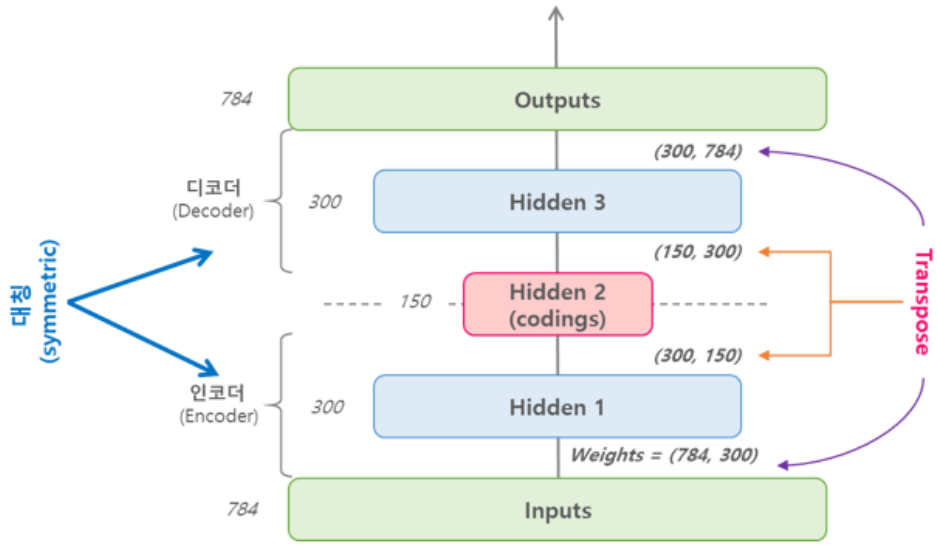
```
#위 모델을 가상의 3D 데이터셋에 훈련
#동일한 데이터셋 인코딩
history = autoencoder.fit(X_train, X_train, epochs=20) #입력과 타깃에 동일한 데이터셋 !!!
codings = encoder.predict(X_train)
```

17.3 적층 오토인코더

적층 오토인코더(= 심층 오토인코더): 은닉층이 여러개인 오토인코더

<구조>

- 가운데 은닉층 즉, 코딩 층을 기준으로 대칭
- 샌드위치 모양



17.3.1 케라스를 사용하여 적층 오토인코더 구현하기

```
stacked_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(30, activation="selu"),
])
stacked_decoder = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[30]),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
stacked_ae = keras.models.Sequential([stacked_encoder, stacked_decoder])
stacked_ae.compile(loss="binary_crossentropy",
                   optimizer=keras.optimizers.SGD(learning_rate=1.5), metrics=[rounded_accuracy])
history = stacked_ae.fit(X_train, X_train, epochs=20,
                         validation_data=(X_valid, X_valid))
```

17.3.2 재구성 시각화

적절하게 훈련되었는지 확인 → 입력과 출력을 비교

- 입력과 출력의 차이가 너무 크지 X

```
#이미지 출력 함수
def plot_image(image):
    plt.imshow(image, cmap="binary")
    plt.axis("off")

#재구성
def show_reconstructions(model, images=X_valid, n_images=5):
    reconstructions = model.predict(images[:n_images])
    fig = plt.figure(figsize=(n_images * 1.5, 3))
    for image_index in range(n_images):
        plt.subplot(2, n_images, 1 + image_index)
        plot_image(images[image_index])
        plt.subplot(2, n_images, 1 + n_images + image_index)
        plot_image(reconstructions[image_index])

show_reconstructions(stacked_ae)
save_fig("reconstruction_plot")
```

(위)원본 이미지 (아래)재구성

- 재구성) 식별은 가능하지만 정보를 조금 많이 잃음
- 모델을 더 오래 훈련
- 인코더, 디코더 층을 늘리기
- 코딩의 크기 늘리기

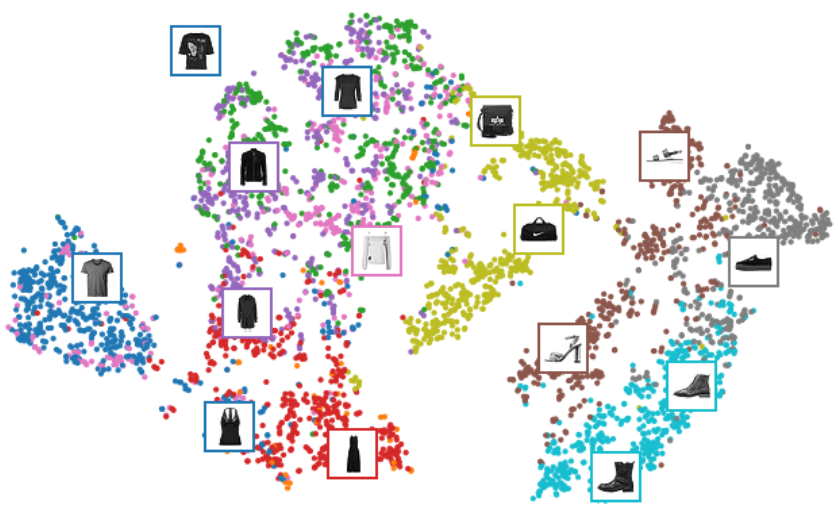
17.3.3 패션 MNIST 데이터셋 시각화

오토인토더를 사용해 차원 축소

- 샘플과 특성이 많은 대용량 데이터셋을 다룰 수 있음

```
# t-SNE 알고리즘으로 차원 축소(2차원)
np.random.seed(42)
```

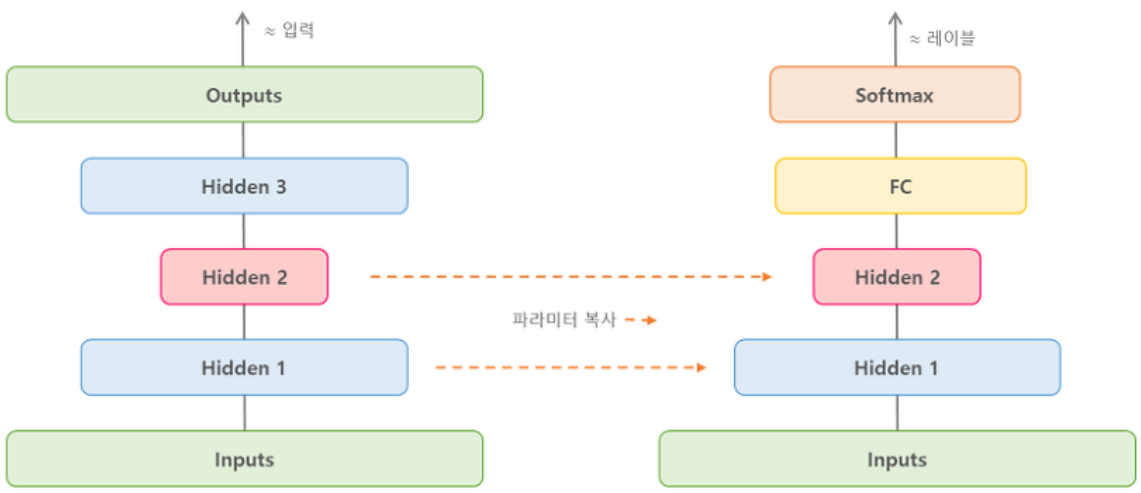
```
from sklearn.manifold import TSNE

X_valid_compressed = stacked_encoder.predict(X_valid)
tsne = TSNE()
X_valid_2D = tsne.fit_transform(X_valid_compressed)

#데이터셋 그리기
plt.scatter(X_valid_2D[:, 0], X_valid_2D[:, 1], c=y_valid, s=10, cmap="tab10")
```



17.3.4 적층 오토인코더를 사용한 비지도 사전훈련

- 신경망 재사용

  기존의 네트워크에서 학습한 특성 감지 기능을 재사용



17.3.5 가중치 묶기

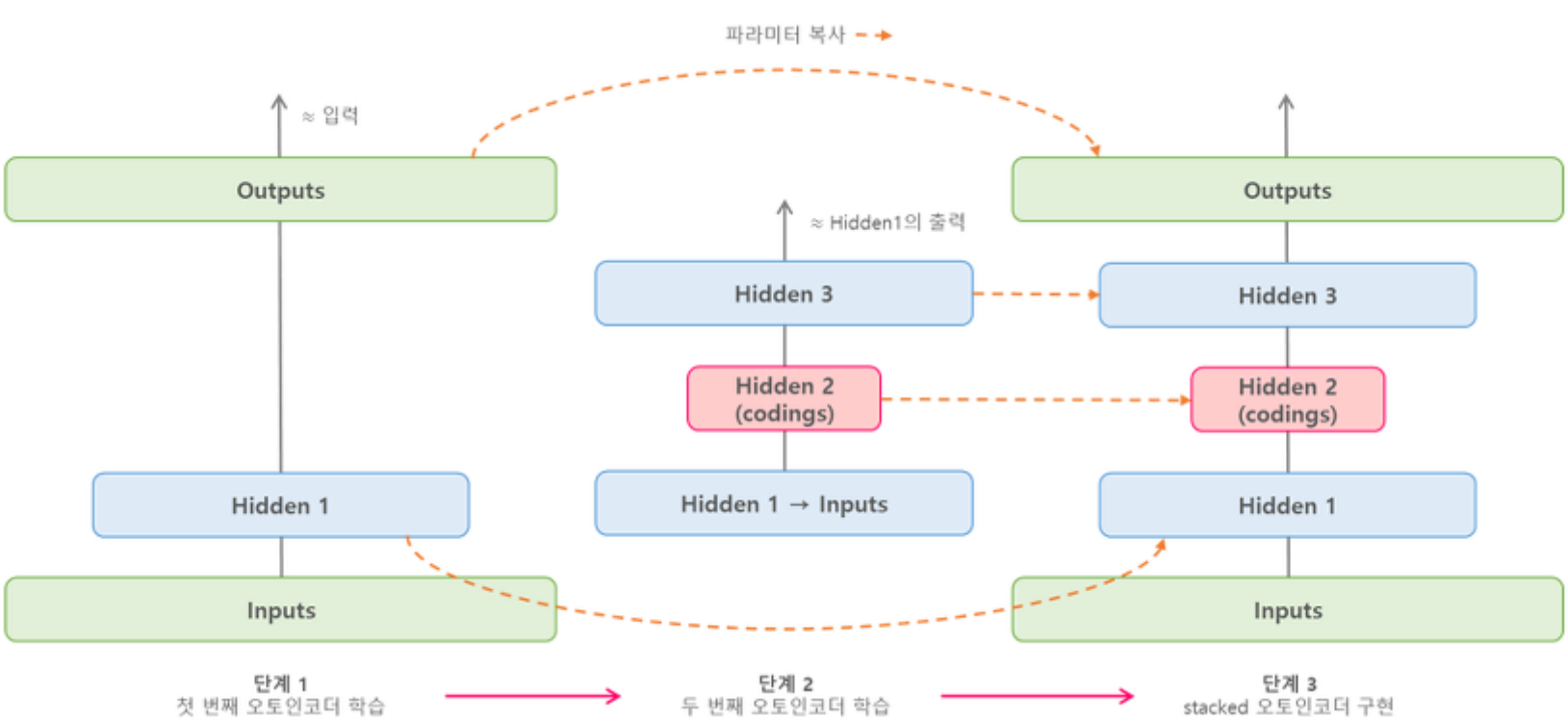
오토인코더가 완벽한 대칭일 때, 디코더의 가중치와 인코더의 가중치를 묶음

- 가중치의 수가 절반으로 줄어듦
- 훈련 속도 상승
- 과대적합 위험 줄어듦

```
#사용자 정의층 - 가중치 묶기
class DenseTranspose(keras.layers.Layer):
    def __init__(self, dense, activation=None, **kwargs):
        self.dense = dense
        self.activation = keras.activations.get(activation)
        super().__init__(**kwargs)
    def build(self, batch_input_shape):
        self.biases = self.add_weight(name="bias",
                                      shape=[self.dense.input_shape[-1]],
                                      initializer="zeros")
```

```
        super().build(batch_input_shape)
    def call(self, inputs):
        z = tf.matmul(inputs, self.dense.weights[0], transpose_b=True) #전치된 가중치 사용
        return self.activation(z + self.biases)

#새로운 적층 오토인코더 생성
dense_1 = keras.layers.Dense(100, activation="selu")
dense_2 = keras.layers.Dense(30, activation="selu")

tied_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    dense_1,
    dense_2
])

tied_decoder = keras.models.Sequential([
    DenseTranspose(dense_2, activation="selu"),
    DenseTranspose(dense_1, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])

tied_ae = keras.models.Sequential([tied_encoder, tied_decoder])
```

17.3.6 한번에 오토인코더 한 개씩 훈련하기



탐욕적 방식의 층별 훈련

17.4 합성곱 오토인코더

- 이미지를 다룰 때 사용

```
#일반적인 CNN
#인코더 - 높이와 너비(공간방향의 차원)는 축소 / 깊이(특성 맵 갯수)는 늘림
conv_encoder = keras.models.Sequential([
    keras.layers.Reshape([28, 28, 1], input_shape=[28, 28]),
    keras.layers.Conv2D(16, kernel_size=3, padding="SAME", activation="selu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(32, kernel_size=3, padding="SAME", activation="selu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(64, kernel_size=3, padding="SAME", activation="selu"),
    keras.layers.MaxPool2D(pool_size=2)
])
#디코더 - 거꾸로 동작(전치 합성곱층 사용)
conv_decoder = keras.models.Sequential([
    keras.layers.Conv2DTranspose(32, kernel_size=3, strides=2, padding="VALID", activation="selu",
                                 input_shape=[3, 3, 64]),
    keras.layers.Conv2DTranspose(16, kernel_size=3, strides=2, padding="SAME", activation="selu"),
    keras.layers.Conv2DTranspose(1, kernel_size=3, strides=2, padding="SAME", activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
conv_ae = keras.models.Sequential([conv_encoder, conv_decoder])
```

17.5 순환 오토인코더

- 시계열, 텍스트와 같은 시퀀스에 대한 오토인코더 생성

```
#인코더: 시퀀스-투-벡터
recurrent_encoder = keras.models.Sequential([
    keras.layers.LSTM(100, return_sequences=True, input_shape=[28, 28]),
    keras.layers.LSTM(30)
])
#디코더: 벡터-투-시퀀스
recurrent_decoder = keras.models.Sequential([
    keras.layers.RepeatVector(28, input_shape=[30]), #타임스텝마다 입력 벡터를 주입하기 위해
    keras.layers.LSTM(100, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(28, activation="sigmoid"))
])
recurrent_ae = keras.models.Sequential([recurrent_encoder, recurrent_decoder])
```

- ~ 순환 오토인코더 - 과소완전
- **과대완전 오토인코더**를 만드는 방법 - 입력 크기만큼 혹은 입력 크기보다 큰 코딩층

17.6 잡음 제거 오토인코더

- 입력에 잡음 추가
- 잡음이 없는 원본 입력을 복원하는 훈련
- **적층 잡음 제거 오토인코더**
  - 순수한 가우시안 잡음

```
denoising_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.GaussianNoise(0.2),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(30, activation="selu")
])
denoising_decoder = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[30]),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
denoising_ae = keras.models.Sequential([denoising_encoder, denoising_decoder])
```

  - 드롭아웃 - 무작위 입력 추출

```
dropout_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(30, activation="selu")
])
dropout_decoder = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[30]),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
dropout_ae = keras.models.Sequential([dropout_encoder, dropout_decoder])
```

17.7 희소 오토인코더

- **희소**
- 비용함수에 적절한 항 추가 - 오토인코더가 코딩 층에서 활성화되는 뉴런 수 감소
- 코딩을 0과 1 사이의 값으로 제한
  - 시그모이드 활성화 함수 사용
  - 큰 코딩 층 사용(ex. 300개 유닛을 가진 층)

```
sparse_l1_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(300, activation="sigmoid"),
    keras.layers.ActivityRegularization(l1=1e-3)  # 이전 층에 `activity_regularizer=keras.regularizers.l1(1e-3)`를 추가하는 것과 같음
])
sparse_l1_decoder = keras.models.Sequential([
```

```
    keras.layers.Dense(100, activation="selu", input_shape=[300]),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
sparse_l1_ae = keras.models.Sequential([sparse_l1_encoder, sparse_l1_decoder])
```

- 모델에 벌칙 부과 - **희소 손실**

  사용자 정의 규제 - 희소 오토인코더 생성

17.8 변이형 오토인코더

- **확률적 오토인코더**

  훈련이 끝난 후에도 출력이 부분적으로 우연에 의해 셩정

- **생성 오토인코더**

  훈련 세트에서 샘플링된 것 같은 새로운 샘플 생성 가능

17.8.1 패션 MNIST 이미지 생성하기

- 시멘틱 보간

17.9 상대적 적대 신경망

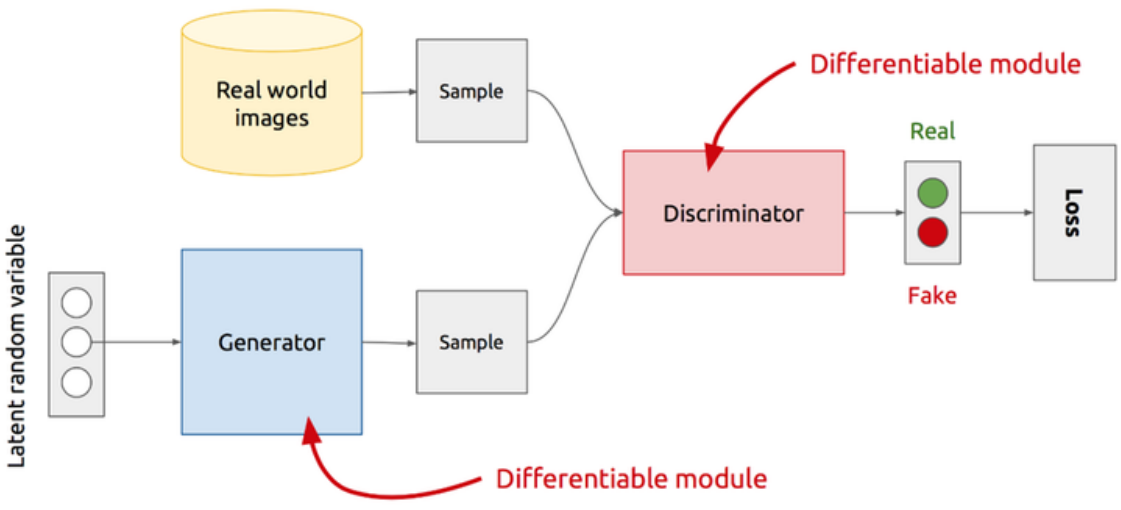
<생성자>

- 랜덤한 분포를 입력으로(생성할 이미지의 잠재 표현 즉, 코딩)
- 이미지 같은 데이터 출력
- 변이형 오토인코더의 디코더와 같은 기능 제공

→ 판별자를 속일 만큼 진짜 같은 이미지를 만드는 것이 목표

<판별자>

- 생성자에서 얻은 가짜 혹은 훈련셋에서 추출한 진짜를 입력으로
- 이미지가 가짜인지 진짜인지 구분 !




```
codings_size = 30

#생성자 생성
generator = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[codings_size]),
    keras.layers.Dense(150, activation="selu"),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])

#판별자 생성
discriminator = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(150, activation="selu"),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(1, activation="sigmoid")
])

#GAN 모델 생성
```

```
gan = keras.models.Sequential([generator, discriminator])

discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")
discriminator.trainable = False
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")

#fit()사용 불가 -> 사용자 함수 생성
def train_gan(gan, dataset, batch_size, codings_size, n_epochs=50):
    generator, discriminator = gan.layers
    for epoch in range(n_epochs):
        print("Epoch {}/{}".format(epoch + 1, n_epochs))
        for X_batch in dataset:
            # phase 1 - training the discriminator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            generated_images = generator(noise)
            X_fake_and_real = tf.concat([generated_images, X_batch], axis=0)
            y1 = tf.constant([[0.]] * batch_size + [[1.]] * batch_size)
            discriminator.trainable = True
            discriminator.train_on_batch(X_fake_and_real, y1)
            # phase 2 - training the generator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            y2 = tf.constant([[1.]] * batch_size)
            discriminator.trainable = False
            gan.train_on_batch(noise, y2)
        plot_multiple_images(generated_images, 8)
        plt.show()
```

17.9.1 GAN 훈련의 어려움

제로섬 게임 - **내시 균형** 상태

모드 붕괴 : 생성자의 출력의 다양성이 줄어들 때

<해결책>

- 경험 재생
- 미니배치 판별

17.9.2 심층 합성곱 GAN

안정적인 합성곱 GAN 구축하기

- (판별자) 풀링 층 → 스트라이드 합성곱으로 변경
- (생성자) 풀링 층 → 전치 합성곱으로 변경
- 생성자, 판별자에 배치 정규화 사용(출력층, 입력층 제외)
- 생성자의 모든 층을 ReLU함수 사용(탄젠트 함수 사용해야하는 층 제외)
- 판별자의 모든 층은 LeakyReLU 사용

```
codings_size = 100
generator = keras.models.Sequential([
    keras.layers.Dense(7 * 7 * 128, input_shape=[codings_size]),
    keras.layers.Reshape([7, 7, 128]),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(64, kernel_size=5, strides=2, padding="SAME",
                                 activation="selu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(1, kernel_size=5, strides=2, padding="SAME",
                                 activation="tanh"),
])
discriminator = keras.models.Sequential([
    keras.layers.Conv2D(64, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2),
                        input_shape=[28, 28, 1]),
    keras.layers.Dropout(0.4),
    keras.layers.Conv2D(128, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2)),
    keras.layers.Dropout(0.4),
    keras.layers.Flatten(),
    keras.layers.Dense(1, activation="sigmoid")
])
gan = keras.models.Sequential([generator, discriminator])
```