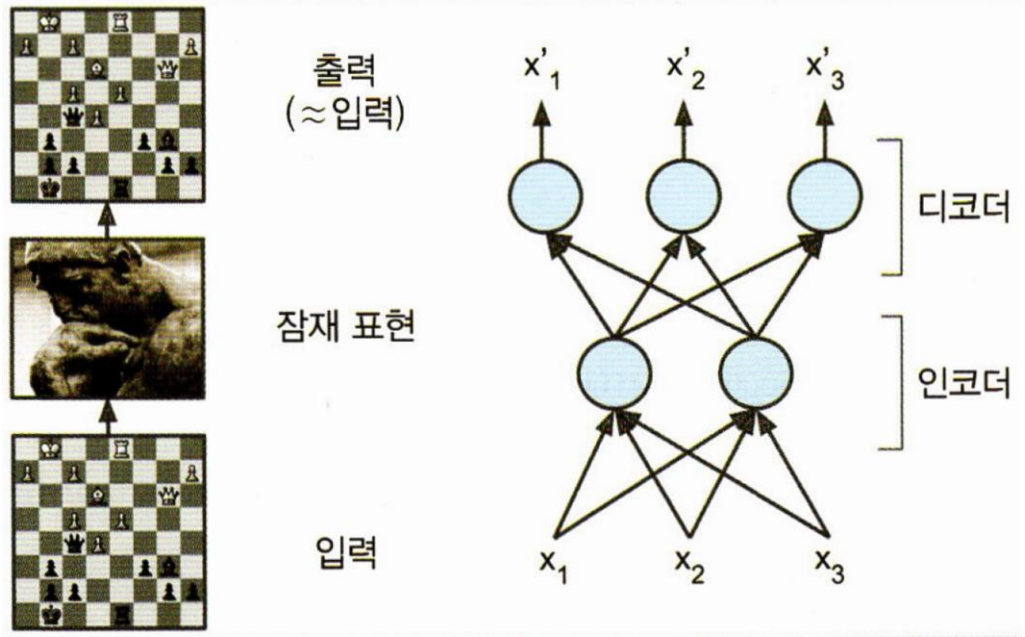
오토인코더와 GAN을 사용한 표현 학습과 생성적 학습

장 승 훈

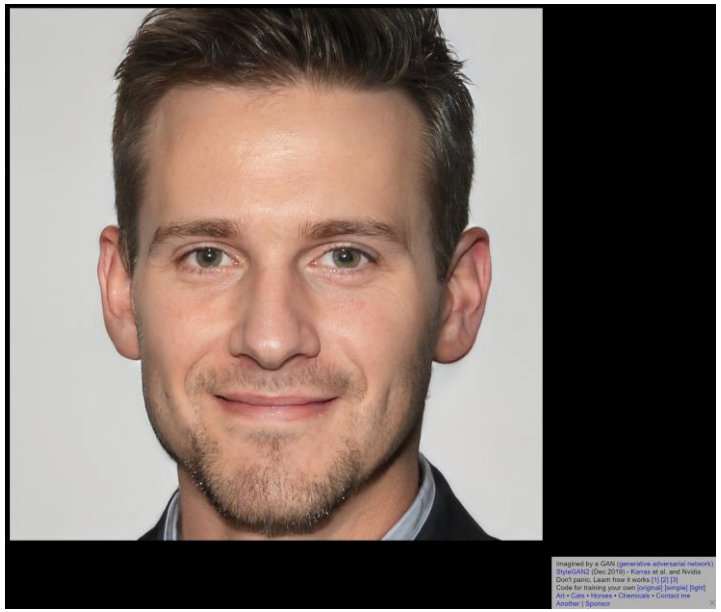
Autoencoder

- 레이블되어 있지 않은 훈련데이터를 사용하여 잠재표현 또는 코딩이랑 부르는 입력데이터의 밀집 표현을 학습할 수 있는 인공 신경망. (비지도 학습)
- 입력출력단순 복사학습 또는 일정제약조건하에서 항등함수 학습.
- 입력보다 낮은 차원을 갖게 됨. (시각화에 유용)
- 일부 오토 인코더는 훈련데이터와 비슷한 새로운 데이터를 생성할 수 있음.

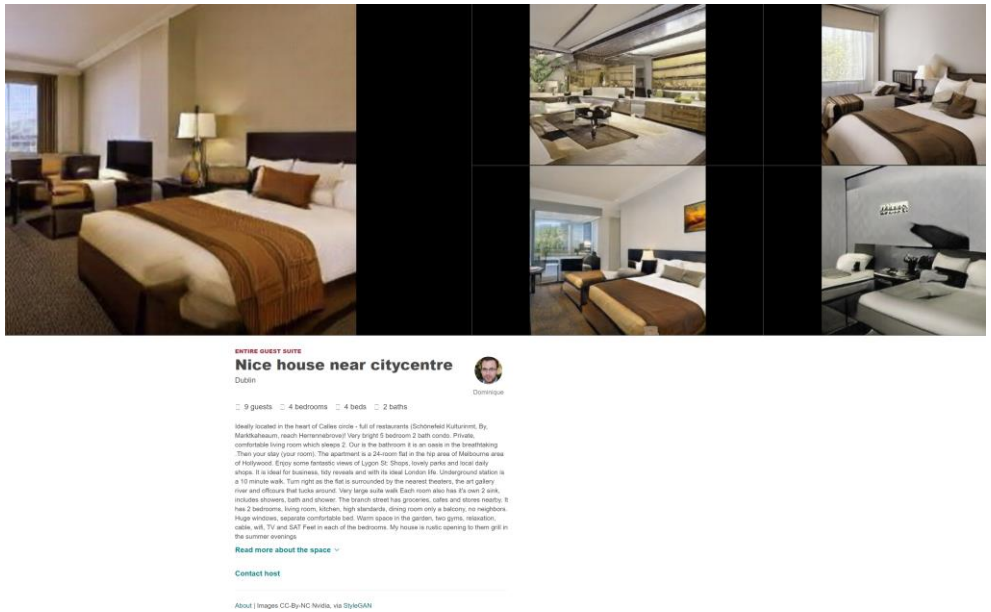


GAN (Generative adversarial networks)

- GAN으로 생성한 얼굴은 가짜라고 믿기 어려울 정도.
- 해상도 높이기/이미지에 컬러입히기/간단스케치를 실제이미지같이바꾸기/동영상에서 다름 프레임 예측하기/모델훈련을 위한 데이터 증식 등에 사용
- 생성자는 훈련데이터와 비슷하게 보이는 데이터를 생성하고, 판별자는 가짜와 진찌를 구별. 생성자와 판발자가 경쟁.



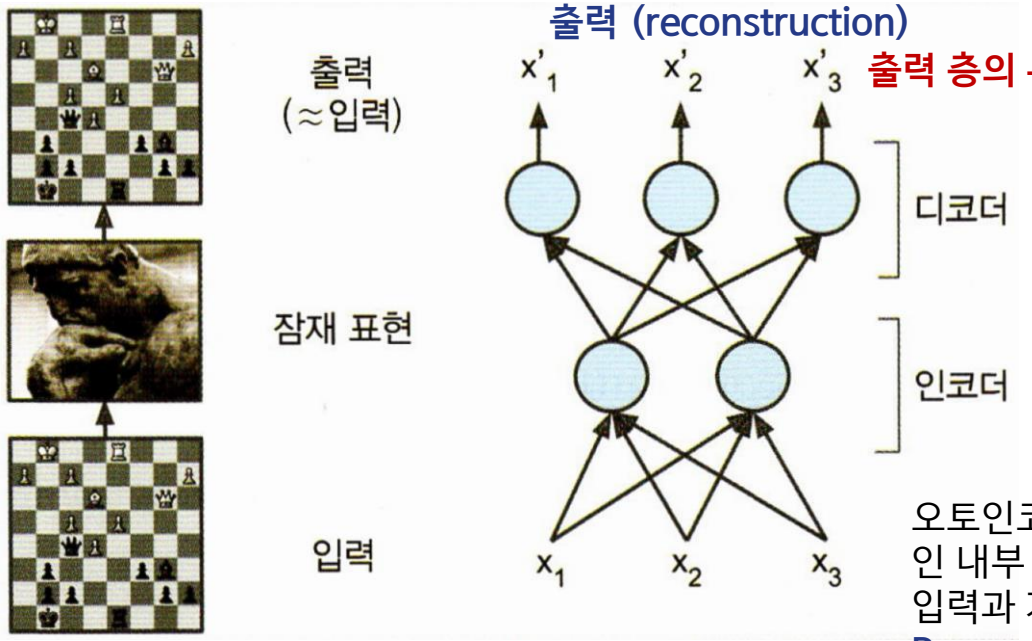
https://thispersondoesnotexist.com



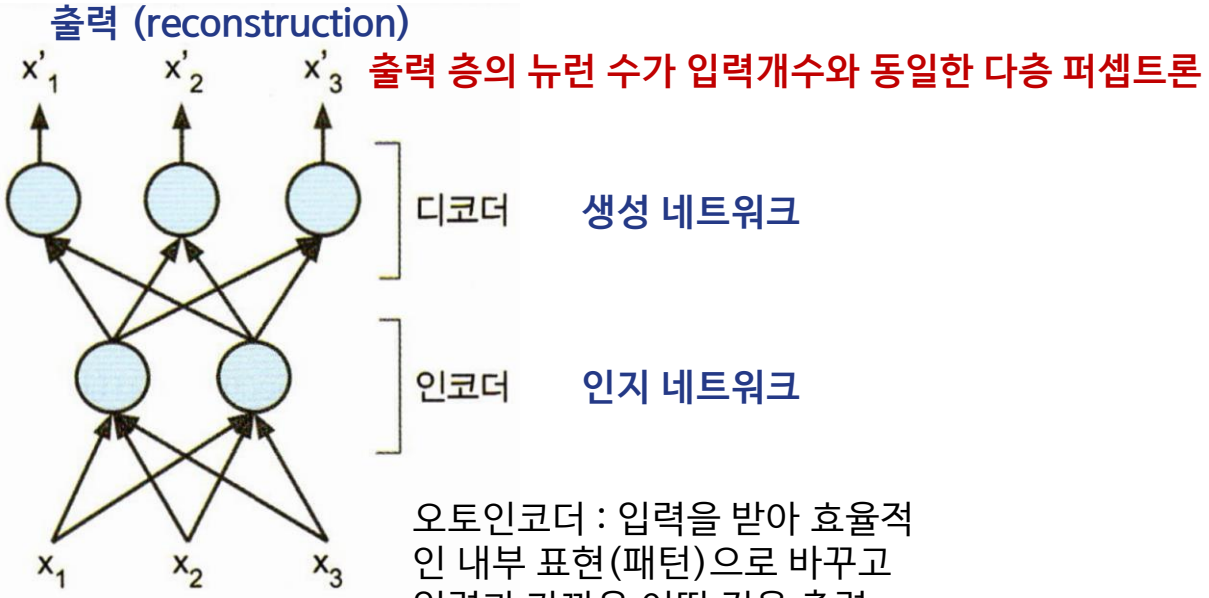
https://thisrentaldoesnotexist.com

효율적인 데이터 표현

- 40, 27, 25, 36, 81, 57, 10, 73, 19, 68
- **50**, 48, 46, 44, 42, 40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, **14**
- 긴 시퀀스를 기억하기 어려우므로 **패턴을 찾는 것이 유용**.
- **오토인코더에 제약??을 가해서** 데이터에 있는 패턴을 찾아 활용.



체스 전문가 : 실제 게임처럼 현실적인 위치에 체스말들이 있을 때 빠르게 전체 말의 위치를 외움



출력 층의 뉴런 수가 입력개수와 동일한 다층 퍼셉트론

오토인코더 : 입력을 받아 효율적인 내부 표현(패턴)으로 바꾸고 입력과 가까운 어떤 것을 출력.
Reconstruction error 체크

과소완전 선형 오토인코더로 PCA 수행하기

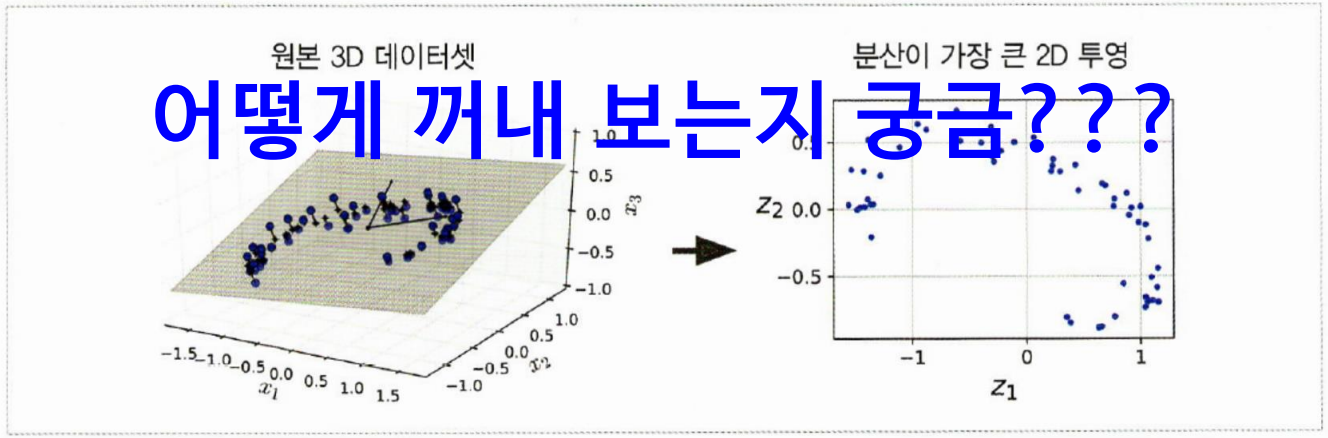
- 선형 활성화 함수만 사용하고 비용함수가 MSE라면 결국 PCA를 수행하는 것과 동일.
- 단순한 PCA를 수행하기 위해서 활성화 함수를 사용하지 않음 (모든 뉴런이 선형)

```
from tensorflow import keras

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))
```

PCA를 적용하여 2D에 투영하는 간단한 선형 오코인코더

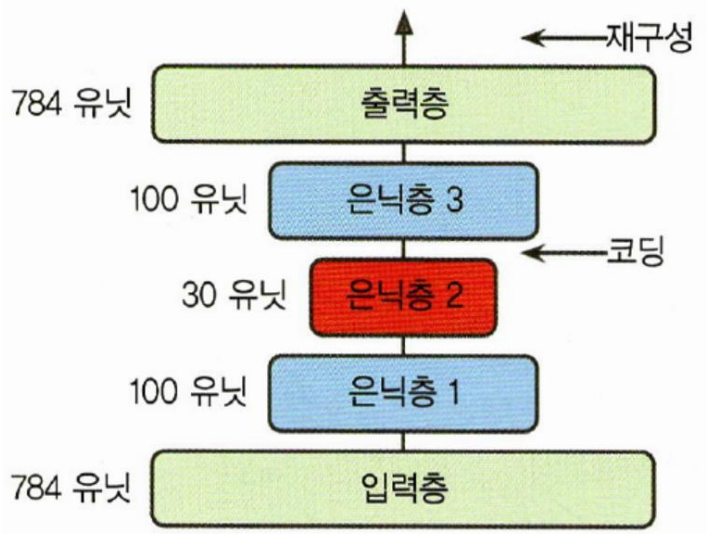
Dense 층을 가진 일반적인 sequential 모델

```
history = autoencoder.fit(X_train, X_train, epochs=20)
codings = encoder.predict(X_train)
```



적층 오토인코더

- 오토인코더도 여러 개의 은닉층을 가질 수 있음. (적층인코더 또는 심층인코더)
- 층을 추가할수록 오토인코더가 복잡한 latent feature를 학습.
- 그러나 오토인코더가 너무 강력해도 안됨. (유용한 데이터 표현을 학습하지 못함)
- MNIST input → (784-100-30-100-784)



케라스를 사용하여 적층오토인코더 구현하기

- 일반적 심층 MLP와 비슷하게 적층 오토인코더 구현 가능.
- 패션 MNIST 데이터셋, SELU 활성화 함수 사용
- 28*28 → 784 → 100 → 30 → 100 → 278 → 28*28

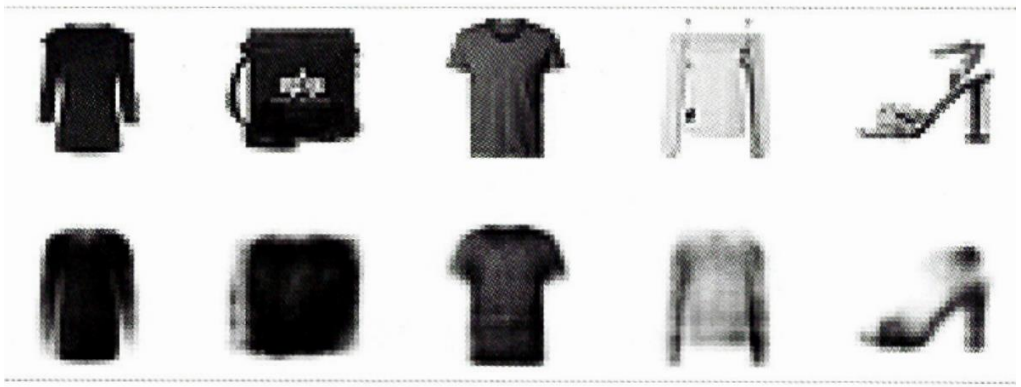
```
stacked_encoder = keras.models.Sequential([ 인코더
    keras.layers.Flatten(input_shape=[28, 28]), 28X28 픽셀 인풋을 받아서 784 벡터로 표현
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(30, activation="selu"),SELU 사용
])
stacked_decoder = keras.models.Sequential([ 디코더
    keras.layers.Dense(100, 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", ──→ 재구성 작업을 다중레이블 이진 분류 문제로 다룸.
                   optimizer=keras.optimizers.SGD(learning_rate=1.5))
history = stacked_ae.fit(X_train, X_train, epochs=10,
                         validation_data=(X_valid, X_valid))
```

재구성 시각화

- 오토인코더가 적절히 훈련되었는지 확인하는 방법은 입력과 출력을 비교.

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

def show_reconstructions(model, n_images=5):
    reconstructions = model.predict(X_valid[: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(X_valid[image_index])
        plt.subplot(2, n_images, 1 + n_images + image_index)
        plot_image(reconstructions[image_index])

show_reconstructions(stacked_ae)
```

Validation 과 reconstruction 을 그리기



원본이미지

재구성이미지

식별이 가능하긴 하지만 정보를 많이 잃어버렸음.
층을 늘리거나 코딩의 크기를 늘려서 모델을 잘 훈련하면 좋을 수 있지만, 반대로 유익한 패턴을 학습하지 못한채 완벽한 재구성 이미지 만을 만들 수 있음.

패션 MNIST 데이터셋 시각화 AE를 사용한 차원축소

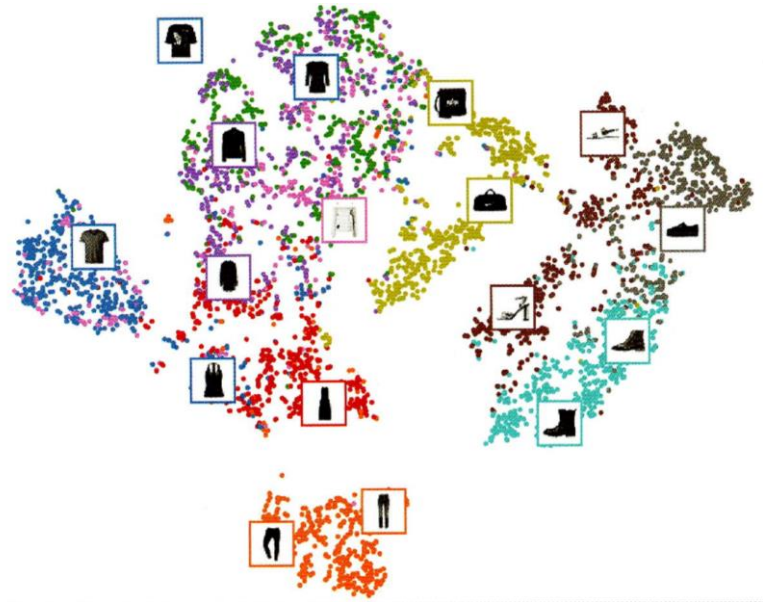
- 데이터셋의 차원축소.
- 다른 차원축소 알고리즘만큼 좋은 결과를 주진 못함
- 그러나 오토인코더는 샘플과 특성이 많은 대용량 데이터셋을 다룰 수 있다는 점이 강점.
- 오토인코더를 사용하여 적절한 수준으로 차원을 축소하고, 이를 다시 다른 차원축소방법으로 축소할 수도 있음.

```
from sklearn.manifold import TSNE

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

MNIST-30차원까지 축소

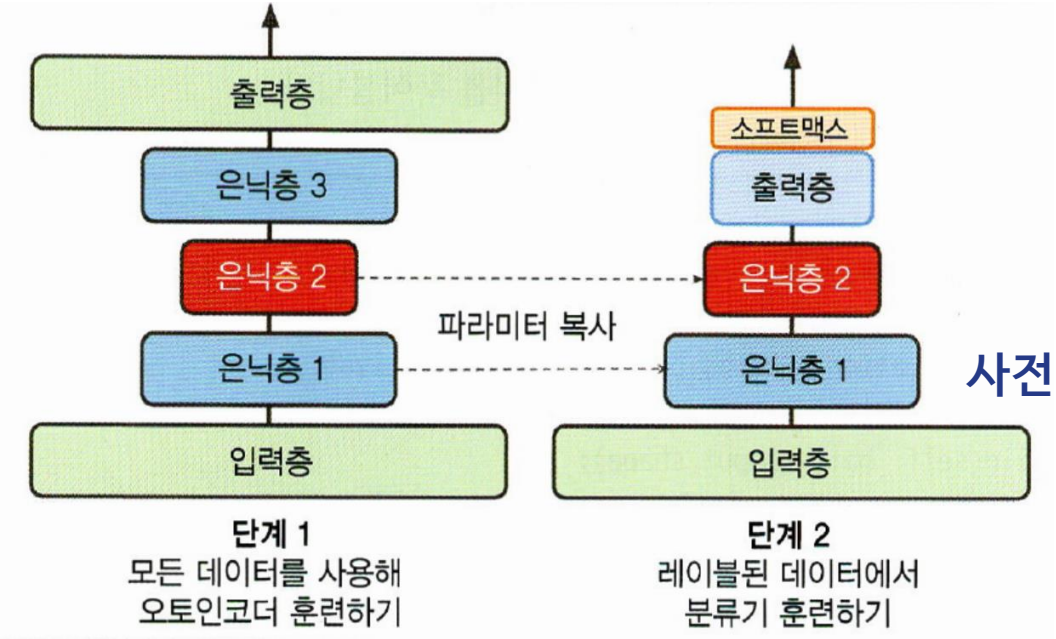
t-SNE-2차원으로 축소

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



적층 오토인코더를 사용한 비지도 사전훈련

- 레이블된 훈련데이터가 많지 않은 복잡한 지도학습문제의 경우, 비슷한 문제를 학습한 신경망을 찾아 하위층을 재사용하는 것도 좋은 방법. (전이학습?)
- 저수준의 특성을 학습할 필요가 없어, 적은 훈련 데이터를 사용해 고성능 모델을 훈련할 수 있음.
- AE를 통해서 찾은 latent feature 를 레이블 데이터에 대한 회귀문제에 활용.



사전훈련된 층을 동결 건조 ^^

가중치 묶기

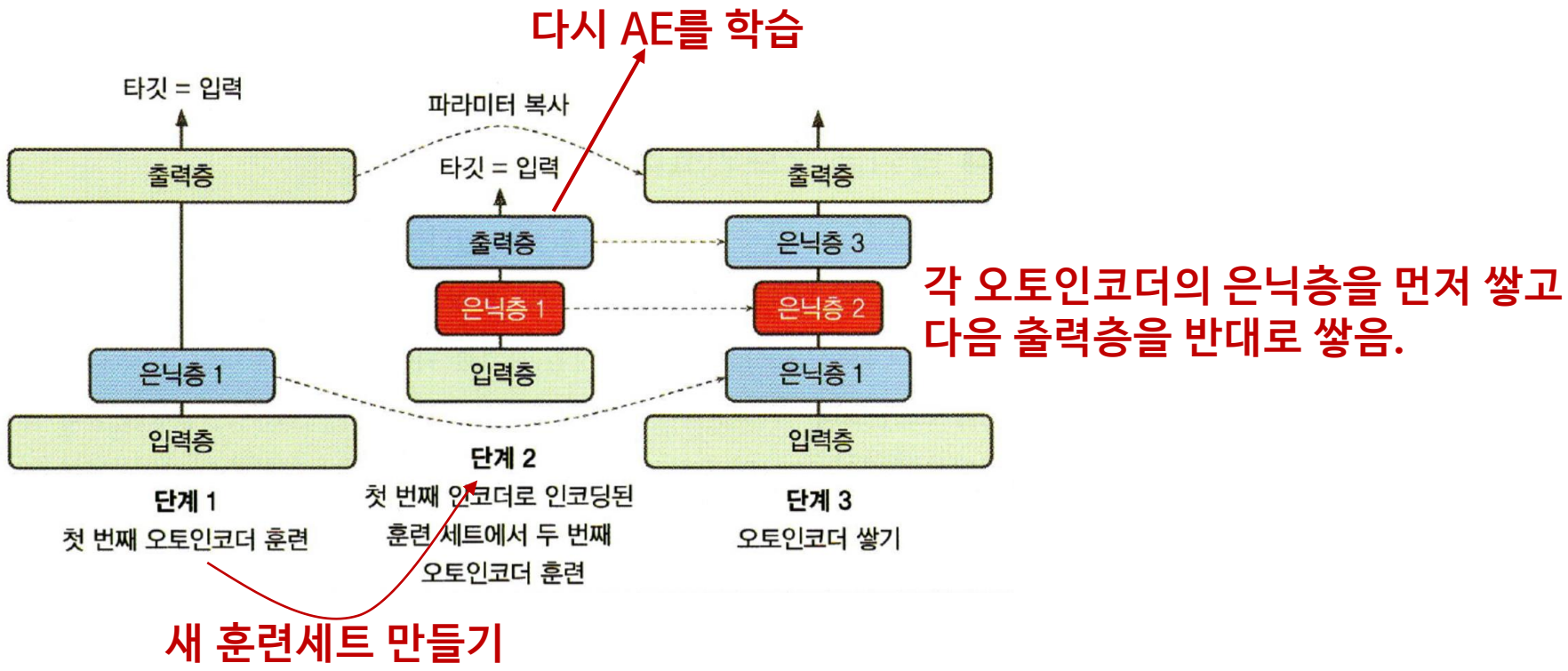
- AE가 완벽히 대칭일 때, 디코더의 가중차와 인코더의 가중치를 묶는 것이 일반적.
- 이렇게 하면 가중치의 수를 절반으로 줄여서, 속도를 높이고 과적합을 방지.

```
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", initializer="zeros",
                                      shape=[self.dense.input_shape[-1]])
        super().build(batch_input_shape)
    def call(self, inputs):
        z = tf.matmul(inputs, self.dense.weights[0], transpose_b=True)
        return self.activation(z + self.biases)
```

이부분은 잘 모르겠음. 684 p는 더 공부!!

한 번에 오토인코더 한 개씩 훈련하기

- 한번에 전체 오토인코더를 훈련하는 것보다 아래 그림과 같이 오토인코더 하나를 훈련하고 이를 쌓아 올려서 한 개의 적층 오토인코더를 만들 수 있음.
- 탐욕적 방식의 층별 훈련 (greedy layerwise training)



합성곱 오토인코더

- 이미지의 경우 AE가 좋은 성능을 내지 못함. CNN이 DNN보다 훨씬 성능이 좋음.
- 이미지에 대한 AE를 만들려면 CAE를 만들어야 함.
- 전형적으로 입력에서 공간방향의 차원(커널 크기)을 줄이고 깊이(특성맵의 개수)는 늘림.
- 디코더는 거꾸로 함.

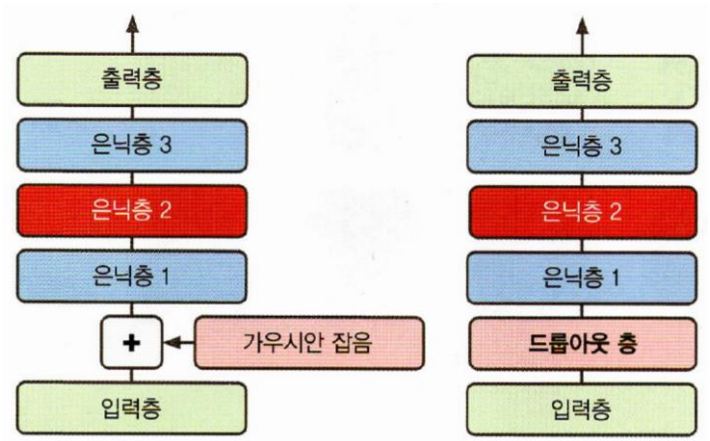
```
conv_encoder = keras.models.Sequential([
    keras.layers.Reshape([28, 28, 1], input_shape=[28, 28]),
    keras.layers.Conv2D(16, kernel_size=3, padding="same", activation="selu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(32, kernel_size=3, padding="same", activation="selu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(64, kernel_size=3, padding="same", activation="selu"),
    keras.layers.MaxPool2D(pool_size=2)
])
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])
```

패션 MNIST 데이터셋에 대한 간단한 합성곱 오토인코더

잡음제거 AE

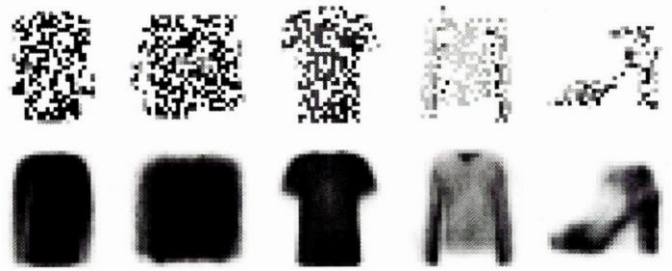
- 잡음을 추가하거나 잡음이 없는 원본 입력을 복원하도록 훈련.
- 얀 르쿤의 1987년 석사 논문에 언급.
- 파스칼 빈센트의 2010년 논문에 적층잡음제거 오토인코더를 소개. Journal of Machine Learning Research 11 (2010): 3371 – 3408

드롭 아웃 층이 있는 일반적인 적층 AE (드롭아웃은 훈련하는 동안에만 활성화, GaussianNoise 층도 마찬가지)



가우시안 잡음(왼쪽) 또는 드롭아웃을 사용한 잡음제거 오토 인코더

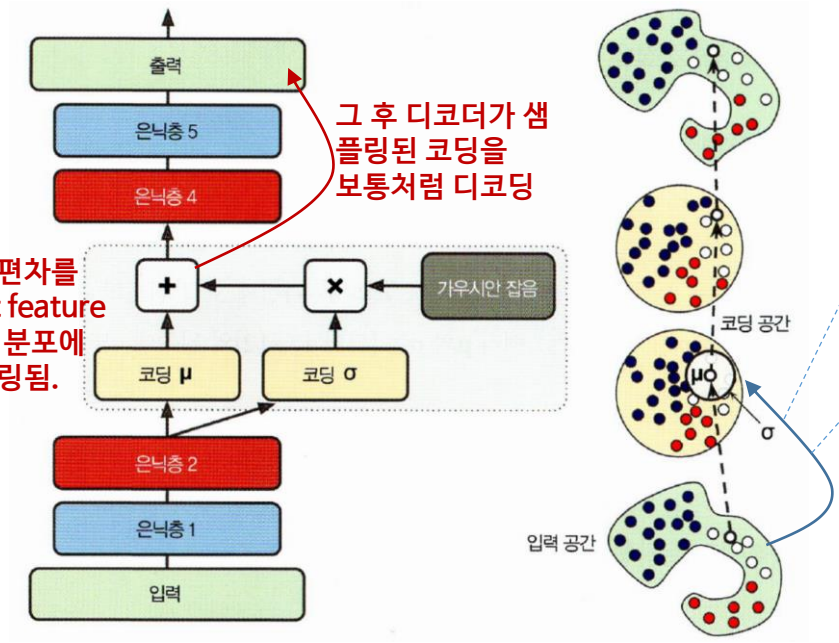
```
dropout_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(30, activation="selu")
])
dropout_decoder = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu",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



잡음 섞인 이미지(위)와 재구성된 이미지(아래)

변이형 오토인코더 (VAE)

- 2014년 다이데릭 킹마와 맥스 웰링이 변이형 오토인코더를 소개했음.
- 확률적 오토인코더. 즉 훈련이 끝난 후에도 출력이 부분적으로 우연에 의해 결정.
- 생성 오토인코더. 마치 훈련 세트에서 샘플링 된 것 같은 새로운 샘플을 생성할 수 있음.
- RBM과 유사하지만 훈련이 더 쉽고 샘플링 과정이 훨씬 빠름.
- 효율적인 근사 베이즈 추론 방법인 변분 베이즈 추론(variational Bayesian inference)

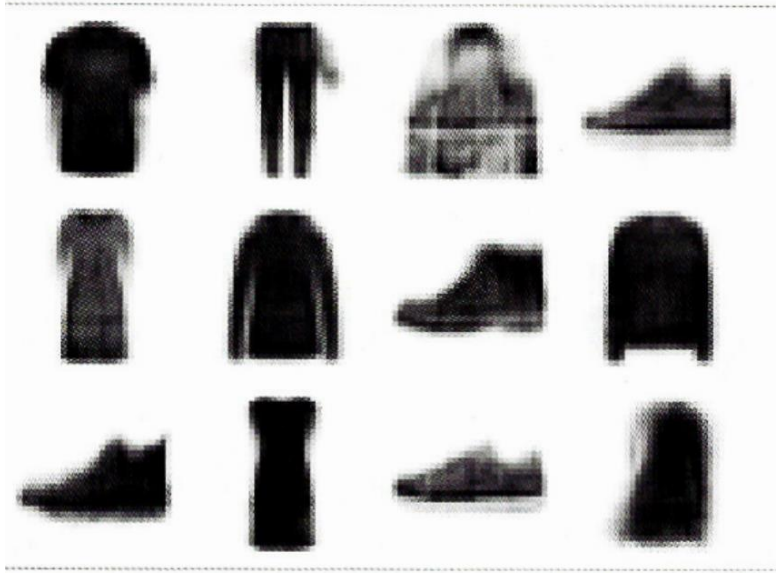


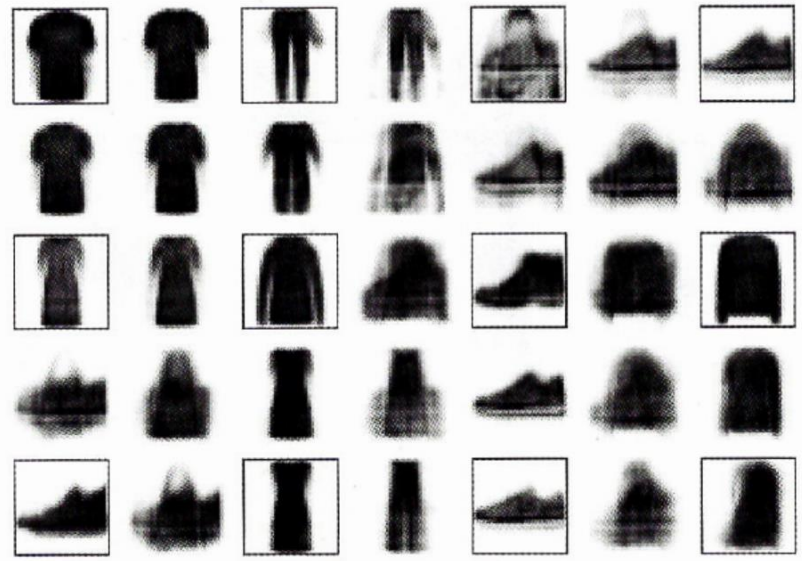
- 입력이 매우 복잡한 분포를 가지더라도 간단한 가우시안 분포에서 샘플링 된 것 처럼 코딩을 만드는 경향이 있음.
- 코딩을 가우시안 샘플들의 군집처럼 보이도록 코딩 공간 안으로 점진적으로 이동
- 따라서 VAE는 훈련이 끝난 뒤 새로운 샘플을 쉽게 생성.

VAE 활용 패션 MNIST 이미지 생성.

- VAE 사용하여 패션의류처럼 보이는 이미지를 생성.
- 가우시안 분포에서 랜덤한 코딩을 샘플링하여 디코딩하는 것이 전부.
- VAE는 시맨틱 보간(interpolation) 수행도 가능.
- 지난 몇년간 VAE가 널리 쓰였지만 GAN이 훨씬 실제 같이 또렷한 이미지를 만들어줘서 인기가 더 많음.

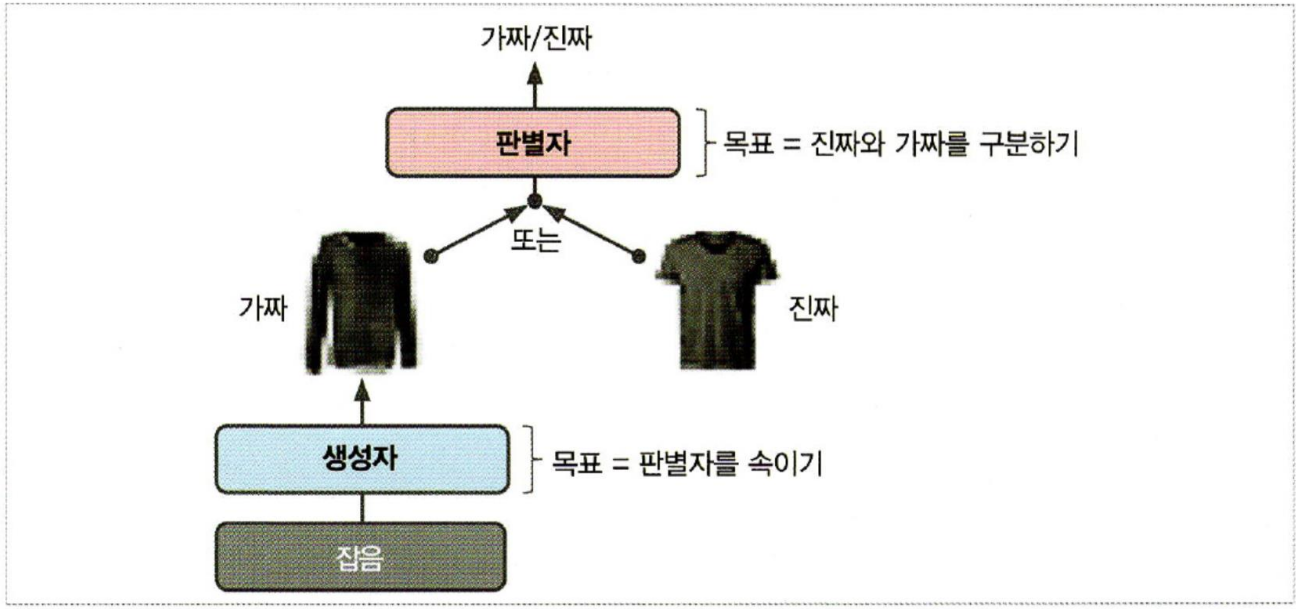
```
codings = tf.random.normal(shape=[12, codings_size])
images = variational_decoder(codings).numpy()
```





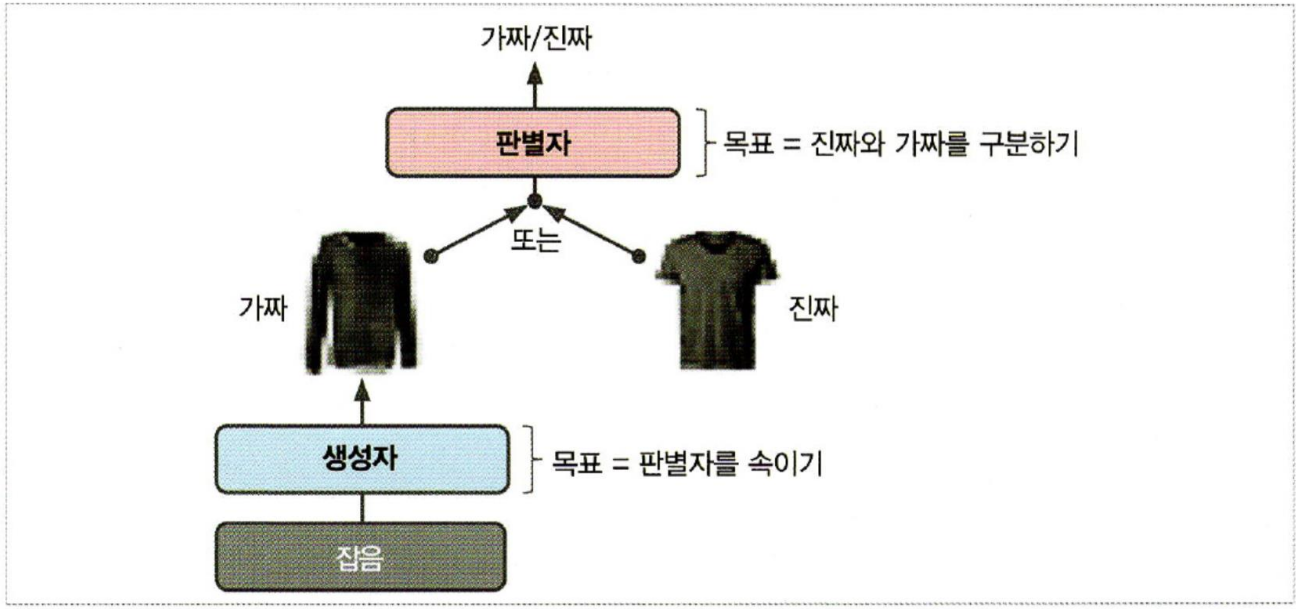
생성적 적대 신경망

- 이언 굿펠로 등이 2014년 논문에서 제안했음.
- 신경망을 서로 겨루게 하고 경쟁을 통해 신경망을 향상하는 것을 기대함.
- 생성자, 랜덤한 분포를 입력으로 받고 이미지와 같은 데이터를 출력, 랜덤한 입력은 생성할 이미지의 latent featur로 생각할 수 있음. 생성자는 VAE의 디코더와 같은 기능을 제공. 가우시안 노이즈를 통해 새로운 이미지를 출력. **판별자를 속일만큼 진짜 같은 이미지를 만듦.**
- 판별자, 생성자에서 얻은 가짜 이미지나 훈련세트에서 추출한 진짜 이미지를 입력으로 받아 이미지가 가짜인지 진짜인지 구분. **진짜와 가짜를 구분.**
- **GAN은 다른 목표를 가진 두 네트워크로 구성되므로 일반적인 신경망 처럼 훈련할 수 없음.**



생성적 적대 신경망

- GAN은 다른 목표를 가진 두 네트워크로 구성되므로 일반적인 신경망 처럼 훈련할 수 없음.
- 첫번째 단계 – **판별자를 훈련**
  - 훈련세트에서 **실제 이미지 배치**를 샘플링 하고 생성자에서 생성한 **동일한 수의 가짜 이미지를 합침**. **가짜 이미지의 레이블은 0**으로 세팅하고 **진짜 이미지는 1**로 세팅.
  - 판별자는 **이진 크로스 엔트로피를 사용**해 한 스텝 동안 이렇게 레이블된 배치로 훈련. 역전파는 **판별자의 가중치만 최적화**함.
- 두번째 단계 – **생성자를 훈련**
  - **생성자**를 사용해 **다른 가짜 이미지 배치를 만듦**. 다시 **판별자를 사용해 진짜/가짜 판별**. 이번에는 **배치에 진짜 이미지를 추가하지 않고 레이블을 모두 1(진짜)로 세팅**. 다시 말해 생성자가 판별자가 진짜라고 (잘못) 믿을 이미지를 만들어야 함!!!
  - 이 단계에서는 **판별자의 가중치를 동결**하는 것이 중요함. **역전파는 생성자의 가중치**에만 영향을 미침.



패션 MNIST 데이터셋으로 간단한 GAN 실행

- 생성자 – 오토인코더의 디코더와 비슷.
- 판별자는 일반적인 이진 분류기
- 훈련이 일반적인반복이 아니기 때문에 fit() 메서드를 사용할 수 없음.

```
codings_size = 30

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

하나의 노드

이진분류기

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

gan 모델을 컴파일하기 전에 판별자가 훈련되지 않도록 설정

패션 MNIST 데이터셋으로 간단한 GAN 실행

- 훈련이 일반적인반복이 아니기 때문에 fit() 메서드를 사용할 수 없음.
- 이를 위해 먼저 이미지를 순회하는 dataset 을 만들어야 함.

```
batch_size = 32
dataset = tf.data.Dataset.from_tensor_slices(X_train).shuffle(1000)
dataset = dataset.batch(batch_size, drop_remainder=True).prefetch(1)
```

```
def train_gan(gan, dataset, batch_size, codings_size, n_epochs=50):
    generator, discriminator = gan.layers
    for epoch in range(n_epochs):
        for X_batch in dataset:
            # 단계 1 - 판별자 훈련
            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)
            # 단계 2 - 생성자 훈련
            noise = tf.random.normal(shape=[batch_size, codings_size])
            y2 = tf.constant([[1.]] * batch_size)
            discriminator.trainable = False

            gan.train_on_batch(noise, y2)

train_gan(gan, dataset, batch_size, codings_size)
```

가우시안 노이즈를 생성자에 주입하여 가짜 이미지 생성.

동일한 개수의 진짜 이미지를 합쳐서 배치 완성

Y1은 가짜 이미지는 0, 진짜 이미지는 1로 설정.

판별자를 훈련.

판별자를 훈련.

가우시안 노이즈 주입

판별자가 가짜 이미지를 진짜로 믿게 만들기 위해서 타겟 y2를 1로 지정

패션 MNIST 데이터셋으로 간단한 GAN 실행

- 생성된 이미지를 출력하면 **첫 번째 에포크가 끝난 후에 벌써 패션 MNIST 처럼 보이는 이미지**를 볼 수 있음.
- 그러나 이보다 더 좋은 이미지는 생성되지 않음. 그 이유는??

GAN 훈련의 어려움.

- 훈련 과정에서 생성자와 판별자는 끊임없이 서로 앞서려고 노력.
- 제로섬 게임.
- 훈련이 진행됨에 따라 내시 균형 (Nash equilibrium) 이라 부르는 상태에 다다름. (서로 전략을 수정하지 않는 상태)
- 모두 도로 오른쪽으로 운전, 포식자는 먹이를 쫓고 먹잇감은 도망치기.
- GAN은 하나의 내시 균형에만 도달할 수 있음을 증명. → 생성자가 완벽하게 가짜이미지를 생성, 판별자는 추측만 가능.
- 충분히 오래 훈련을 하면 이런 상태에 도달할 지 모르나, 어떤것도 이를 보장하지 않음.

- 가장 큰 어려움은 모드 붕괴. (생성자의 출력의 다양성 감소)
- 생성자가 다른 클래스보다 신발을 그럴싸 하게 만들어 낸다고 가정하면, 판별자를 더 속이기 위해서 더 더양한 신발 이미지를 만들도록 유도함. 생성자는 점진적으로 다른 이미지를 생성하는 방법을 잊게 됨. 그 동안 판별자가 보게 될 유일한 가짜 이미지는 신발이 됨.
- 그러면 다시 판별자도 다른 클래스의 가짜 이미지를 구별하는 방법을 잊어버림. 판별자가 진짜 신발 주에서 가짜를 구별하게 되면 생성자는 다른 클래스로 옮겨가야 함. 셔츠로 옮겨가서 잘 훈련하면 신발에 대해서는 잊어버릴 것. 그리고 판별자도 뒤따라 갈 것.
- 생성자와 판별자가 지속적으로 서로에게 영향을 주기 때문에 파마리터 변동이 크고 불안정. 안정적으로 훈련되다가도 갑자기 발산 할 수 있음.

- 이러한 요인들로 인해서 GAN 하이퍼 파라미터는 매우 민감하게 작동. 세부 튜닝을 위해서는 많은 노력이 필요.

➔ 이 분야는 매우 활발히 연구 중이며 GAN의 역학은 아직 완벽하게 파악하지 못했음.

심층 합성곱 GAN
ProGAN
StyleGAN