10. 생성모델과 창작

Preview

- 인간의 생성 능력
  - 예) 아이는 오늘 겪은 일을 아빠에게 이야기하고, 처음 가본 곳의 풍경을 그림으로 그림
    - 현실 세계를 비슷하게 모방하지만 같지는 않음(의도적 왜곡, 도구 한계로 추상화 등)
- 분별 모델과 생성 모델
  - 분별 모델
    - 가족의 얼굴을 알아보고 표정을 보고 상대의 감정을 알아보는 등의 능력
    - 인공지능은 분별 능력을 중심으로 발전해 옴. 앞서 공부한 SVM, 다층 퍼셉트론, 컨볼루션 신경망, LSTM, 강화 학습은 모두 분별 모델
  - 생성 모델
    - 사람의 필체를 흉내 내는 인공지능 등. 예전에는 HMM 등의 모델을 사용
    - 2010년대부터 딥러닝 기반 생성 모델로 발전. GAN(10.3절)이 대표적임

https://www.thispersondoesnotexist.com/

그림 10-1 ProGAN 생성 모델로 만든 위조 얼굴 영상(하나는 진짜 사람 얼굴 영상)

10.1 확률적 생성 모델

확률을 사용하는 확률적 생성 모델

- 이 절은 화소 각각에 대해 독립적으로 발생 확률을 추정한 다음, 추정한 확률로 화솟값을 생성하는 매우 단순한 모델을 다룸

10.1.1 확률적 생성 모델을 MNIST에 적용

- MNIST 데이터셋에 확률적 생성 모델을 적용하는 [프로그램 10-1]

프로그램 10-1 확률 생성 모델을 MNIST 필기 숫자에 적용하기

```
01  import numpy as np
02  from tensorflow.keras.datasets import mnist
03
04  # MNIST 데이터를 읽고 0 패턴만 추출
05  (x_train,y_train),(x_test,y_test)=mnist.load_data()
06  X=x_train[np.isin(y_train,[0])]
07
08  # 화소 수준의 확률 생성 모델 구축
09  P=np.zeros((28,28,256))
10  for k in range(X.shape[0]):
11      for i in range(X.shape[1]):
12          for j in range(X.shape[2]):
13              P[i][j][X[k][i][j]]+=1
14  P=P/X.shape[0]
15
```

0 패턴에만 적용

09~14행 화소별로 0~225 명암값의 빈도를 구하고 확률로 변환

10.1.1 확률적 생성 모델을 MNIST에 적용


```
16  # 확률 생성 모델을 이용하여 20개 샘플을 생성
17  Xnew=np.zeros((20,28,28))
18  for i in range(20):
19      for r in range(28):
20          for c in range(28):
21              Xnew[i][r][c]=np.random.choice(range(256),p=P[r][c])
22
23  import matplotlib.pyplot as plt
24
25  plt.figure(figsize=(20,4))
26  for r in range(2):
27      for c in range(10):
28          plt.subplot(2,10,r*10+c+1)
29          plt.imshow(Xnew[r*10+c],cmap='gray')
30          plt.xticks([]); plt.yticks([])
31  plt.show()
```

확률 분포에 따른 난수 생성하여 화솟값 결정

생성된 샘플을 보면, 0 형태는 유지하지만 품질이 매우 낮음(화소 간의 상관 관계를 고려하지 않았기 때문)

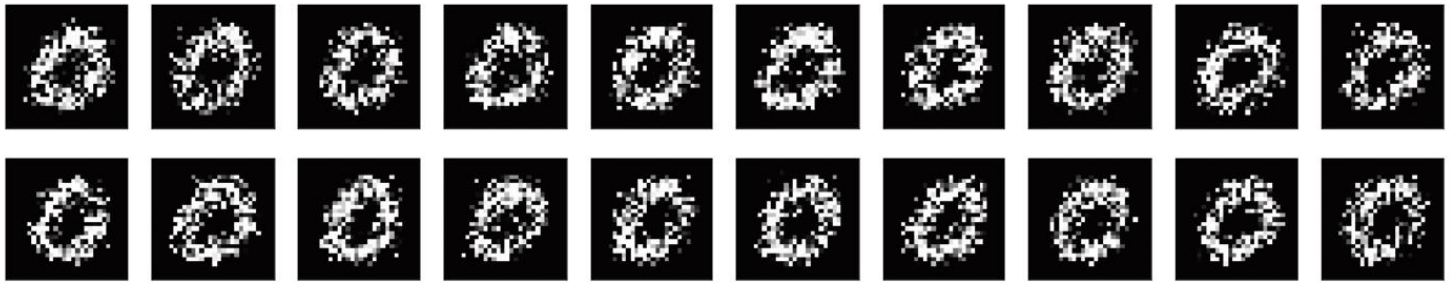
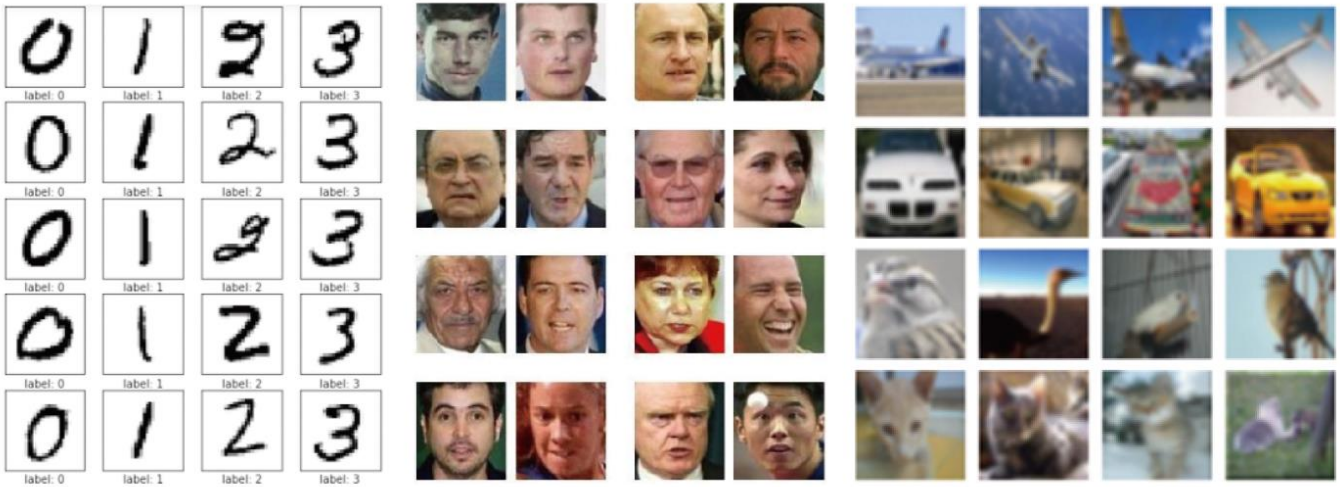


그림 10-2 화소 수준의 확률적 생성 모델이 만든 MNIST 패턴

10.1.2 현실 세계의 복잡성

- 현실 세계의 영상은 모양을 제어하는 고수준의 특징이 불분명함
  - 생성 모델을 설계하는 일이 무척 까다로움 → 오토인코더와 GAN이 새로운 길을 열어줌



(a) MNIST의 필기 숫자 (b) LFW의 얼굴 영상 (c) CIFAR-10의 자연 영상

그림 10-3 현실 세계의 패턴

- 현대적인 생성 모델은 다양한 매체를 생성함
  - 음악, 문장, 스케치, 화학구조, … https://thisxdoesnotexist.com/

10.2 오토인코더

- 오토인코더는 입력 패턴과 출력 패턴이 같은 신경망
  - 사람이 레이블을 달 필요가 없는 비지도 학습
  - 고전적인 응용: 영상 압축, 잡음 제거 등
  - 딥러닝 응용: 특징 추출 또는 생성 모델
- 이 절은 오토인코더를 생성 모델로 활용

10.2.1 오토인코더의 구조와 원리

오토인코더

- 입력 패턴 **x**를 입력 받아 **x**와 똑같은 또는 유사한 **x**′를 출력하는 신경망
- 아무 제약이 없다면 은닉층의 노드 개수를 입력층과 같게 하고 모든 가중치를 1로 설정하면 됨. 하지만 이런 신경망은 무용지물

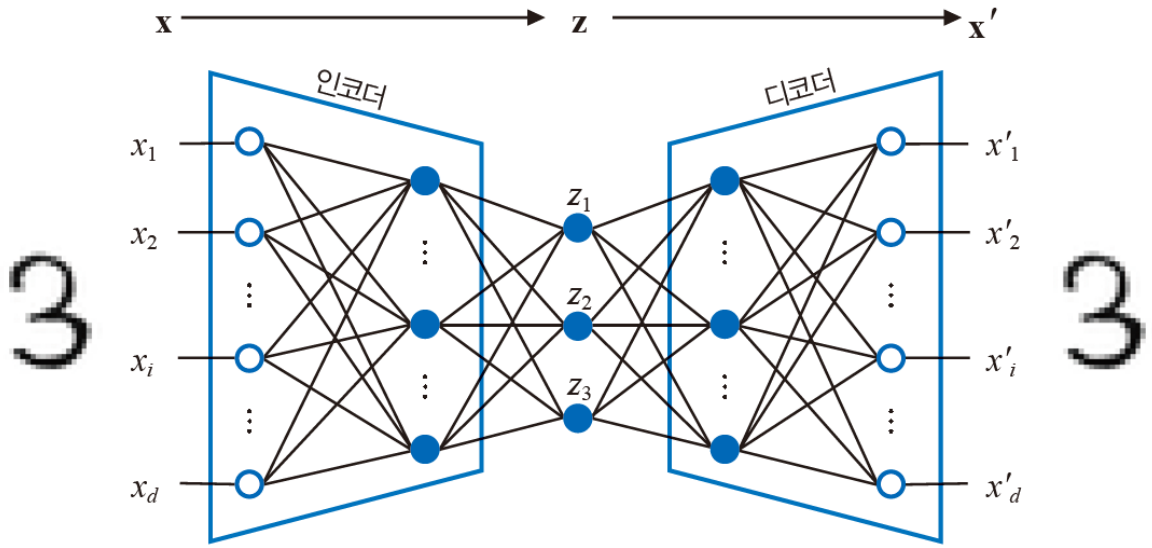


그림 10-4 오토인코더의 구조

- 실제로는 은닉층의 노드 개수를 축소하여 설계
  - 인코더는 차원을 줄이고 디코더는 차원을 회복. **z** 공간을 잠복 공간latent space이라 부름

10.2.2 오토인코더 프로그래밍

- [프로그램 10-2(a)]는 MNIST를 가지고 오토인코더를 구현

프로그램 10-2(a) MNIST를 이용한 오토인코더 구현

```
01  import numpy as np
02  from tensorflow.keras.datasets import mnist
03  from tensorflow.keras.layers import Input,Dense,Flatten,Reshape,Conv2D,
    Conv2DTranspose
04  from tensorflow.keras.models import Model
05  from tensorflow.keras import backend as K
06
07  # MNIST 데이터를 읽고 신경망에 입력할 준비
08  (x_train,y_train),(x_test,y_test)=mnist.load_data()
09  x_train=x_train.astype('float32')/255.
10  x_test=x_test.astype('float32')/255.
11  x_train=np.reshape(x_train,(len(x_train),28,28,1))
12  x_test=np.reshape(x_test,(len(x_test),28,28,1))
13
14  zdim=32 # 잠복 공간의 차원
15
```

잠복 공간 z의 차원을 32로 설정

실습

10.2.2 오토인코더 프로그래밍

신경망의 중간 결과에 접근해야 하므로
Functional API 방식으로 코딩

```
16  # 오토인코더의 인코더 부분 설계
17  encoder_input=Input(shape=(28,28,1))
18  x=Conv2D(32,(3,3),activation='relu',padding='same',strides=(1,1))(encoder_
    input)
19  x=Conv2D(64,(3,3),activation='relu',padding='same',strides=(2,2))(x)
20  x=Conv2D(64,(3,3),activation='relu',padding='same',strides=(2,2))(x)
21  x=Conv2D(64,(3,3),activation='relu',padding='same',strides=(1,1))(x)
22  x=Flatten()(x)
23  encoder_output=Dense(zdim)(x)
24  model_encoder=Model(encoder_input,encoder_output)
25  model_encoder.summary()
26
27  # 오토인코더의 디코더 부분 설계
28  decoder_input=Input(shape=(zdim,))
29  x=Dense(3136)(decoder_input)
30  x=Reshape((7,7,64))(x)
31  x=Conv2DTranspose(64,(3,3),activation='relu',padding='same',strides=(1,1))(x)
32  x=Conv2DTranspose(64,(3,3),activation='relu',padding='same',strides=(2,2))(x)
33  x=Conv2DTranspose(32,(3,3),activation='relu',padding='same',strides=(2,2))(x)
34  x=Conv2DTranspose(1,(3,3),activation='relu',padding='same',strides=(1,1))(x)
35  decoder_output=x
36  model_decoder=Model(decoder_input,decoder_output)
37  model_decoder.summary()
38
```

인코더에 해당하는 model_encoder 객체

디코더에 해당하는 model_decoder 객체

10.2.2 오토인코더 프로그래밍

```
39  # 인코더와 디코더를 결합하여 오토인코더 모델 구축
40  model_input=encoder_input
41  model_output=model_decoder(encoder_output)
42  model=Model(model_input,model_output)
43
44  # 오토인코더 학습
45  model.compile(optimizer='Adam',loss='mse')
46  model.fit(x_train,x_train,epochs=5,batch_size=128,shuffle=True,validation_
    data=(x_test,x_test))
47
48  # 복원 실험 1: x_test를 복원하는 예측 실험
49  decoded_img=model.predict(x_test)
50
51  import matplotlib.pyplot as plt
52
53  n=10
54  plt.figure(figsize=(20, 4))
55  for i in range(n):
56      plt.subplot(2, n, i+1)
57      plt.imshow(x_test[i].reshape(28, 28),cmap='gray')
58      plt.xticks([]); plt.yticks([])
59      plt.subplot(2, n, i + n+1)
60      plt.imshow(decoded_img[i].reshape(28, 28),cmap='gray')
61      plt.xticks([]); plt.yticks([])
62  plt.show()
```

인코더와 디코더를 결합한 오토인코더에 해당하는 model 객체

오토인코더의 원리에 따라 입력과 출력이 모두 x_train

학습된 오토인코더로 테스트 집합에 대해 예측을 수행

53~62행은 앞 10개 패턴의 예측 결과를 출력

10.2.2 오토인코더 프로그래밍

```
Model: "model_4"  (인코더를 담당하는 모델 encoder_model)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 7, 7, 64)          36928
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 7, 7, 64)          36928
_________________________________________________________________
flatten_2 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                100384
=================================================================
Total params: 193,056
Trainable params: 193,056
Non-trainable params: 0
_________________________________________________________________
```

10.2.2 오토인코더 프로그래밍

```
Model: "model_5" (디코더를 담당하는 모델 decoder_model)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 32)]              0
_________________________________________________________________
dense_4 (Dense)              (None, 3136)              103488
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 7, 7, 64)          36928
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 14, 14, 64)        36928
_________________________________________________________________
conv2d_transpose_6 (Conv2DTr (None, 28, 28, 32)        18464
_________________________________________________________________
conv2d_transpose_7 (Conv2DTr (None, 28, 28, 1)         289
=================================================================
Total params: 196,097
Trainable params: 196,097
Non-trainable params: 0
_________________________________________________________________
```

10.2.2 오토인코더 프로그래밍

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 121s 2ms/sample - loss: 0.0209 -
val_loss: 0.0064
Epoch 2/5
60000/60000 [==============================] - 127s 2ms/sample - loss: 0.0056 -
val_loss: 0.0049
Epoch 3/5
60000/60000 [==============================] - 100s 2ms/sample - loss: 0.0045 -
val_loss: 0.0043
Epoch 4/5
60000/60000 [==============================] - 136s 2ms/sample - loss: 0.0041 -
val_loss: 0.0039
Epoch 5/5
60000/60000 [==============================] - 134s 2ms/sample - loss: 0.0038 -
val_loss: 0.0037
```

원래 샘플

오토인코더가 예측한 샘플

10.2.2 오토인코더 프로그래밍

Functional API 방식으로 코딩

- 오토인코더에서는 신경망의 중간 층의 결과에 접근할 필요가 있어 Function API 방식으로 코딩해야 함

NOTE Sequential 방식과 Functional API 방식

이 책은 처음부터 Sequential 방식을 사용해왔다. 아래 표의 왼쪽은 C-C-FC 구조의 신경망을 설계하는 Sequential 방식의 코드를 보여준다. 이 방식에서는 중간 결과를 따로 빼내어 사용할 방법이 없다. 따라서 출력이 하나이면 충분한 상황에서 사용한다. Functional API에서는 예제처럼 x1, x2, x3이라는 서로 다른 객체에 중간 결과를 저장할 수 있다. 따라서 중간 결과에 접근하여 또 다른 데이터 흐름을 쉽게 만들 수 있다. Functional API에서는 데이터의 흐름을 여러 줄기로 나누어 신경망이 여러 개의 데이터를 출력하게 만들 수 있다.

Sequential 방식	Functional API 방식
model=Sequential() model.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)) model.add(Conv2D(64,(3,3),activation='relu') model.add(Flatten()) model.add(Dense(10,activation='softmax'))	input=Input(shape=(28,28,1)) x1=Conv2D(32,(3,3),activation='relu')(input) x2=Conv2D(64,(3,3),activation='relu')(x1) x3=Flatten()(x2) output=Dense(10)(x3) model=Model(input,output)

10.2.3 생성 모델로서 오토인코더

- [프로그램 10-2(a)]의 32차원의 잠복 공간의 의미
  - 28*28(=784) 차원을 32차원으로 축소
  - 원래 패턴을 아주 비슷하게 복원하므로 잠복 공간은 원래 패턴을 충실하게 표현하는 고수준 특징으로 간주할 수 있음
    - 예를 들어, 첫번째 차원 z_1은 획의 둥근 정도, 두번째 차원 z_2는 획의 두께 등
  - 따라서 디코더를 떼어내고 인코더 부분만 취하여 특징 추출기로 활용 가능. 뒤에 다층 퍼셉트론 또는 SVM을 붙이면 훌륭한 필기 숫자 인식기가 됨
  - 현대 딥러닝은 오토인코더를 사용하지 않더라도 높은 성능을 달성할 수 있어 특징 추출기로 활용하는 사례가 줄고 있음
  - 대신 생성 모델로 많이 사용함

10.2.3 생성 모델로서 오토인코더

오토인코더로 새로운 샘플 생성

- [프로그램 10-2(b)]는 학습된 디코더로 새로운 샘플을 생성하는 프로그램

프로그램 10-2(b) 학습된 오토인코더를 생성 모델로 활용

```
63
64  # 생성 실험 1: 첫 번째 샘플의 잠복 공간 표현에 잡음을 섞어 새로운 샘플 생성
65  x0=x_test[0]
66  z=model_encoder.predict(x0.reshape(1,28,28,1))
67  print(np.round(z,3))
68  zz=np.zeros((20,zdim))
69  for i in range(20):
70      zz[i]=z[0]+(i-10)/10.0
71  generated_img=model_decoder.predict(zz)
72
73  plt.figure(figsize=(20, 4))
74  for i in range(20):
75      plt.subplot(2,10,i+1)
76      plt.imshow(generated_img[i].reshape(28,28),cmap='gray')
77      plt.xticks([]); plt.yticks([])
78      plt.title('noise='+str((i-10)/10.0))
79  plt.show()
80
```

65~66행은 첫번째 샘플에 대해 인코더로 잠복 공간의 점을 예측

68~70행은 잠복 공간의 점에 잡음을 섞음

71행은 잡음 섞인 점에 대해 디코더로 샘플 생성

10.2.3 생성 모델로서 오토인코더

```
81  # 생성 실험 2: 같은 부류의 두 샘플 사이를 보간하여 새로운 샘플 생성
82  x4_6=np.array((x_test[4],x_test[6]))
83  z=model_encoder.predict(x4_6)
84  zz=np.zeros((20,zdim))
85  for i in range(20):
86      alpha=i/(20.0-1.0)
87      zz[i]=(1.0-alpha)*z[0]+alpha*z[1]
88  generated_img=model_decoder.predict(zz)
89
90  plt.figure(figsize=(20, 4))
91  for i in range(20):
92      plt.subplot(2,10,i+1)
93      plt.imshow(generated_img[i].reshape(28,28),cmap='gray')
94      plt.xticks([]); plt.yticks([])
95      plt.title('alpha='+str(round(i/(20.0-1.0),3)))
96  plt.show()
```

82~83행은 4번과 6번 샘플 대해 인코더로 잠복 공간의 점을 예측(둘 다 4 패턴)

84~87행은 두 잠복 공간의 점을 잇는 선분 상에서 20개 점 생성

88행은 사잇점에 대해 디코더로 샘플 생성

10.2.3 생성 모델로서 오토인코더

```
[[ 0.038  0.622 -0.264 -1.344  1.494  0.236  0.754  1.339 -1.158 -0.343
  -0.24  -1.562  0.122  0.368  0.663 -1.2   -0.882 -0.922 -0.963  0.847
   0.655  0.391 -0.223 -0.717  0.748 -0.652  0.8   -1.007 -0.086 -0.533
   0.59   0.65 ]]
```

32차원 잠복 공간 상의 점의 좌표

생성 실험 1의 결과

생성 실험 2의 결과

10.2.3 생성 모델로서 오토인코더

- [프로그램 10-2(b)]가 생성한 샘플의 품질 평가
  - 실험 1: 7과 비슷한 샘플이 생성됨. 잡음이 0.3 근방에서 패턴이 왜곡되기 시작하고 잡음이 더 커지면 형편없는 모양이 됨
  - 실험 2: 서로 다른 모양의 4 패턴이 서서히 변함을 확인. Alpha가 0.5 근방에서 획이 끊어져 품질이 떨어지는 현상
- 결론적으로
  - 오토인코더는 생성 모델로서 가능성이 있음
  - 잠복 공간의 점들 중에 품질이 떨어지는 것이 다수 있음

10.2.4 2차원 잠복 공간 관찰

- [프로그램 10-3]은 2차원 잠복 공간에서 테스트 집합의 분포를 시각화
  - 시각화를 위해 2차원으로 축소

프로그램 10-3 오토인코더의 2차원 잠복 공간을 시각화

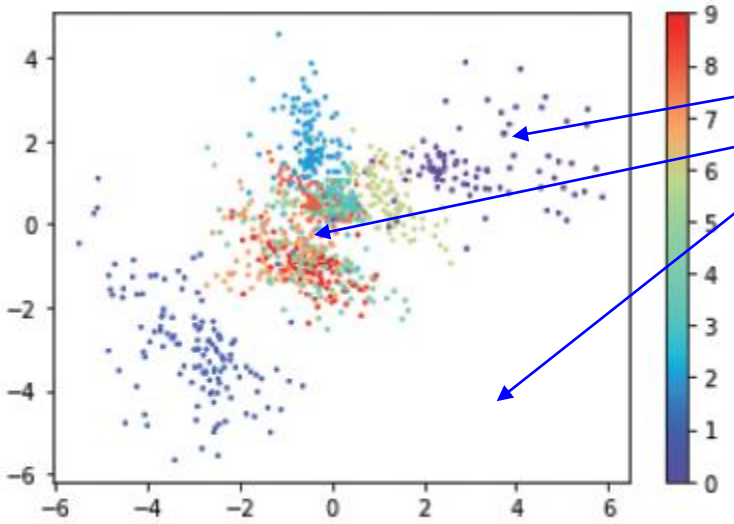
```
01
…   [프로그램 10-2(a)]의 01~12행
12
13
14  zdim=2    # 시각화를 위해 2차원으로 설정
15
16
…   [프로그램 10-2(a)]의 16~46행
46
47
```

10.2.4 2차원 잠복 공간 관찰

```
48  decoded_img=model.predict(x_test)
49
50  import matplotlib.pyplot as plt
51
52  # 테스트 샘플에 대해 복원 품질 평가
53  n = 10
54  plt.figure(figsize=(20, 4))
55  for i in range(n):
56      plt.subplot(2, n, i+1)
57      plt.imshow(x_test[i].reshape(28, 28),cmap='gray')
58      plt.xticks([]); plt.yticks([])
59      plt.subplot(2, n, i + n+1)
60      plt.imshow(decoded_img[i].reshape(28, 28),cmap='gray')
61      plt.xticks([]); plt.yticks([])
62  plt.show()
63
64  # 테스트 집합의 분포를 2차원 잠복 공간에 시각화
65  n=1000
66  xx=x_test[0:n]
67  z=model_encoder.predict(xx)
68  sc=plt.scatter(z[:,0],z[:,1],s=2,c=y_test[0:n],cmap='rainbow')
69  plt.colorbar(sc)
```

10.2.4 2차원 잠복 공간 관찰

잠복 공간을 2차원으로
축소하여 품질 저하



같은 부류가 군집화
원점 부근에 밀집
안쓰는 공간이 많음

10.3 생성 적대 신경망

- 2014년에 굿펠로는 생성 적대 신경망(GAN)을 발표
  - GAN(generative adversarial network)은 2개의 신경망이 적대적인 관계에서 학습하는 생성 모델
  - 이후 개량된 GAN이 여럿 발표되는데, 현재 ProGAN이 가장 뛰어남
    - [그림 10-1]은 ProGAN이 생성한 가짜 얼굴로서, 세번째만 핀란드 산나 마린 수상의 진짜 얼굴

10.3.1 동기와 원리

GAN의 원리

- 생성망 G와 분별망 D라는 두 개의 대립 관계의 신경망을 사용
  - G는 D를 속일 수 있을 정도로 품질이 높는 가짜 샘플을 생성
  - D는 G가 만든 가짜 샘플을 높은 정확률로 맞힘

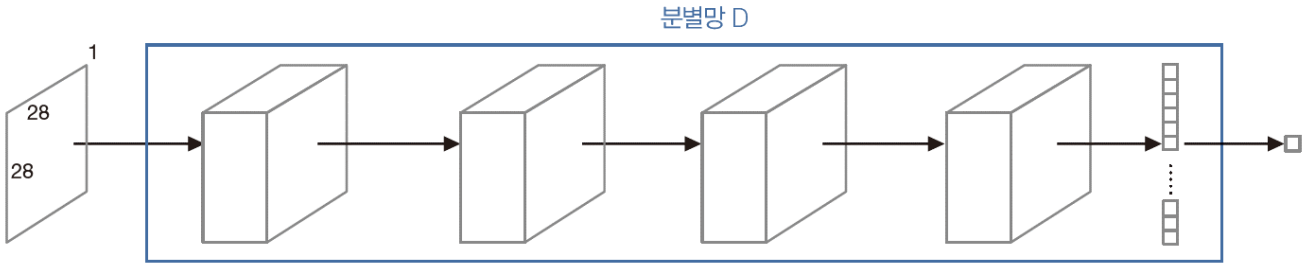
그림 10-5 생성 적대 신경망에서 생성망과 분별망의 적대적 관계

- 위조지폐범과 경찰에 비유
  - 현실 세계와 달리 위조지폐범에 해당하는 생성망이 승리해야 함

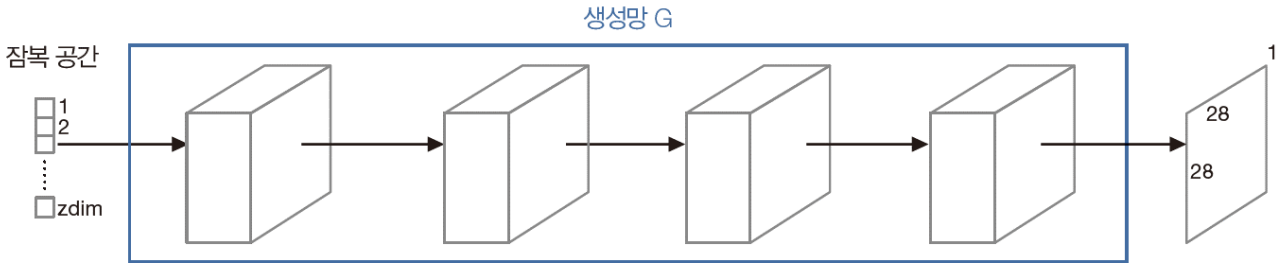
10.3.1 동기와 원리

구조(MNIST를 예로 사용하여 설명)

- 분별망 D
  - 입력은 28*28 영상. 출력 노드는 1개(1은 진짜, 0은 가짜, 활성 함수로 sigmoid 사용)
- 생성망 G
  - 입력은 zdim-차원의 잠복 공간의 한 점의 좌표. 출력은 28*28 영상
- 오토인코더와 비슷하여 구조를 코딩하는 일은 쉬움



(a) 분별망 D



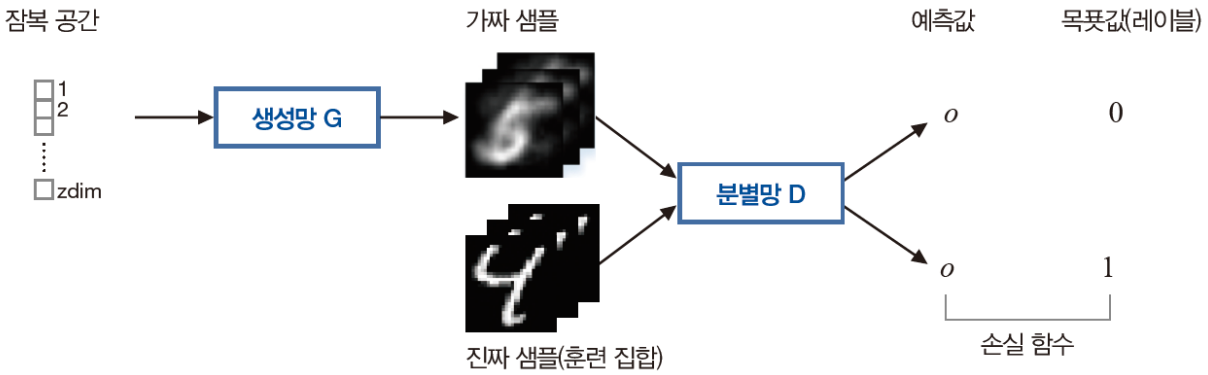
(b) 생성망 G

그림 10-6 생성 적대 신경망의 분별망과 생성망의 구조

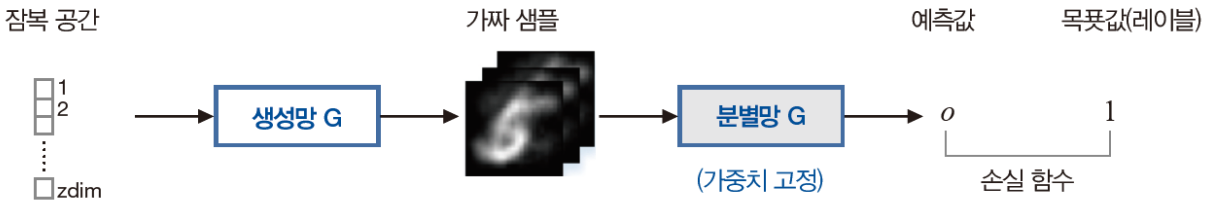
10.3.1 동기와 원리

학습

- 분별망의 학습: 2부류(진짜와 가짜) 분류에 해당하므로 비교적 쉬움([그림 10-7(a)])
- 생성망의 학습은 복잡([그림 10-7(b)])
  - G가 생성한 가짜 샘플에 레이블 1을 붙여 학습. 즉 분별망을 속이는 학습
  - 이때 G의 가중치를 고정하고 학습해야 함. 왜?



(a) 분별망 학습



(b) 생성망 학습

그림 10-7 생성 적대 신경망의 분별망과 생성망의 학습

10.3.2 생성 적대 신경망의 프로그래밍

- MNIST를 가지고 GAN을 구현하는 [프로그램 10-4]

실습

프로그램 10-4 MNIST데이터를 활용한 생성 적대 신경망의 구현

```
01  import numpy as np
02  from tensorflow.keras.datasets import mnist
03  from tensorflow.keras.layers import Input,Activation,Dense,Flatten,Reshape,
    Conv2D,Conv2DTranspose,Dropout,BatchNormalization,UpSampling2D
04  from tensorflow.keras.models import Model
05  from tensorflow.keras import backend as K
06  from tensorflow.keras.losses import mse
07  import matplotlib.pyplot as plt
08
09  (x_train,y_train),(x_test,y_test)=mnist.load_data()
10  x_train = (x_train.astype('float32')/255.0)*2.0-1.0      # [-1,1] 구간
11  x_test = (x_test.astype('float32')/255.0)*2.0-1.0
12  x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
13  x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
14
15  batch_siz=64
16  epochs=5000
17  dropout_rate=0.4
18  batch_norm=0.9
19  zdim=100                                                 # 잠복 공간의 차원
20
```

09~13행은 MNIST 읽고 신경망에 입력할 수 있는 형태로 변환

잠복 공간의 차원을 100으로 설정

10.3.2 생성 적대 신경망의 프로그래밍

```
21  discriminator_input=Input(shape=(28, 28, 1))          # 분별망 D 설계
22  x=Conv2D(64,(5,5),activation='relu',padding='same',strides=(2,2))
    (discriminator_input)
23  x=Dropout(dropout_rate)(x)
24  x=Conv2D(64,(5,5),activation='relu',padding='same',strides=(2,2))(x)
25  x=Dropout(dropout_rate)(x)
26  x=Conv2D(128,(5,5),activation='relu',padding='same',strides=(2,2))(x)
27  x=Dropout(dropout_rate)(x)
28  x=Conv2D(128,(5,5),activation='relu',padding='same',strides=(1,1))(x)
29  x=Dropout(dropout_rate)(x)
30  x=Flatten()(x)
31  discriminator_output=Dense(1,activation='sigmoid')(x)
32  discriminator=Model(discriminator_input,discriminator_output)
33
34  generator_input=Input(shape=(zdim,))                  # 생성망 G 설계
35  x=Dense(3136)(generator_input)
36  x=BatchNormalization(momentum=batch_norm)(x)
37  x=Activation('relu')(x)
38  x=Reshape((7,7,64))(x)
39  x=UpSampling2D()(x)
40  x=Conv2D(128,(5,5),padding='same')(x)
41  x=BatchNormalization(momentum=batch_norm)(x)
42  x=Activation('relu')(x)
43  x=UpSampling2D()(x)
44  x=Conv2D(64,(5,5),padding='same')(x)
45  x=BatchNormalization(momentum=batch_norm)(x)
46  x=Activation('relu')(x)
47  x=Conv2D(64,(5,5),padding='same')(x)
48  x=BatchNormalization(momentum=batch_norm)(x)
49  x=Activation('relu')(x)
50  x=Conv2D(1,(5,5),activation='tanh',padding='same')(x)
51  generator_output=x
52  generator=Model(generator_input,generator_output)
```

진짜(1)와 가짜(0)를 구별하려고 sigmoid 사용

분별망 D를 만들어 discriminator 객체에 저장

생성망 G를 만들어 generator 객체에 저장

10.3.2 생성 적대 신경망의 프로그래밍

분별망의 학습 설정

```
53
54 discriminator.compile(optimizer='Adam',loss='binary_crossentropy',metrics
   =['accuracy'])
55
56 discriminator.trainable=False
57 gan_input=Input(shape=(zdim,))
58 gan_output=discriminator(generator(gan_input))
59 gan=Model(gan_input,gan_output)
60 gan.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['accuracy'])
61     분별망 학습을 하는 함수
62 def train_discriminator(x_train):
63     c=np.random.randint(0,x_train.shape[0],batch_siz)
64     real=x_train[c]
65     discriminator.train_on_batch(real,np.ones((batch_siz,1)))
66
67     p=np.random.normal(0,1,(batch_siz,zdim))
68     fake=generator.predict(p)
69     discriminator.train_on_batch(fake,np.zeros((batch_siz,1)))
70     생성망 학습을 하는 함수
71 def train_generator():
72     p=np.random.normal(0,1,(batch_siz,zdim))
73     gan.train_on_batch(p,np.ones((batch_siz,1)))
74
```

56~60행은 생성망의 학습 설정([그림 10-7(b)] 참조)

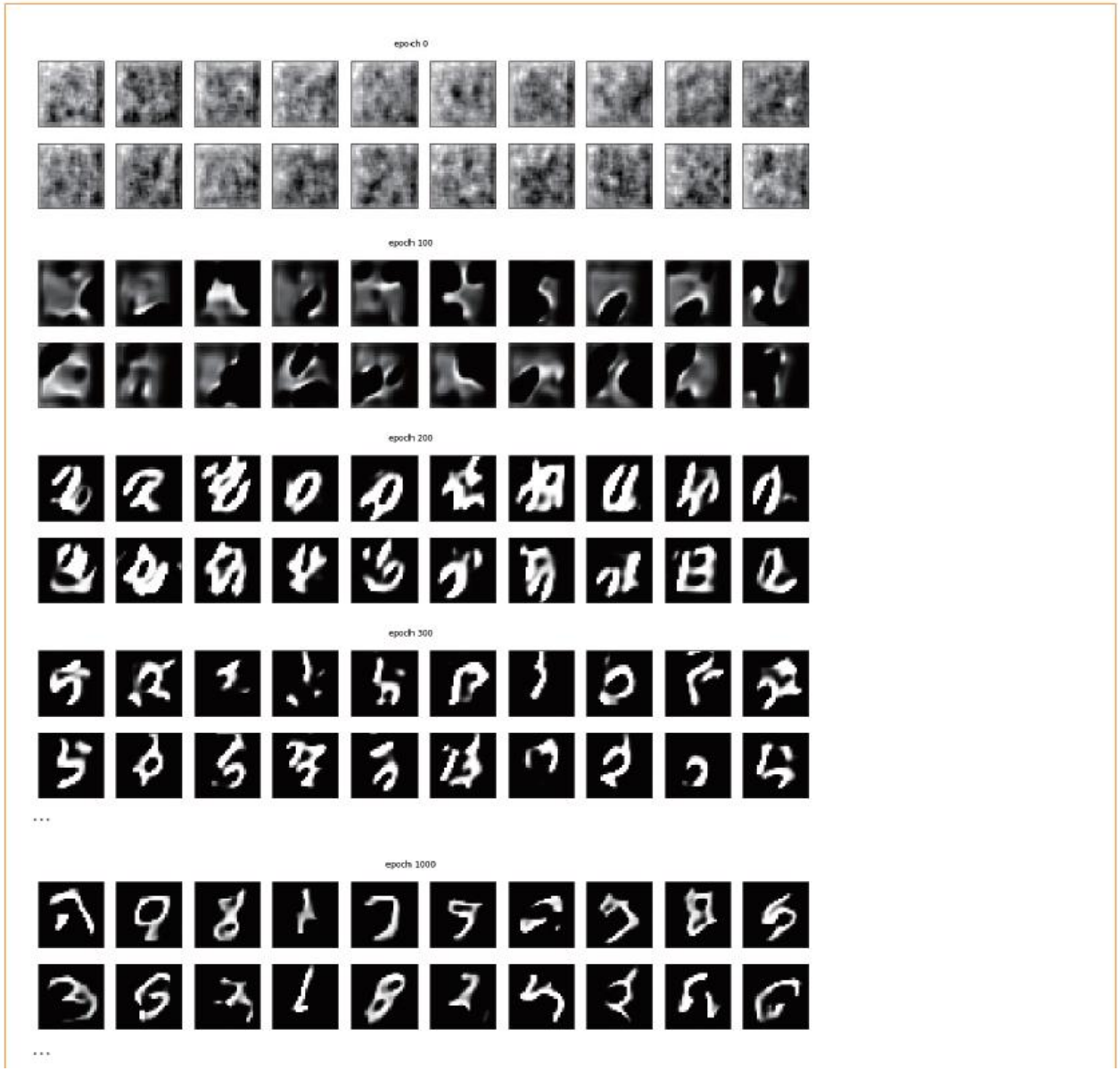
63~65행은 진짜 샘플에 레이블 1을 붙이고 학습

67~69행은 가짜 샘플에 레이블 0을 붙이고 학습

10.3.2 생성 적대 신경망의 프로그래밍

```
75  for i in range(epochs+1):                  # 학습을 수행
76      train_discriminator(x_train)  분별망 학습을 호출
77      train_generator()             생성망 학습을 호출
78      if(i%100==0):                          # 학습 도중 100세대마다 중간 상황 출력
79          plt.figure(figsize=(20, 4))
80          plt.suptitle('epoch '+str(i))
81          for k in range(20):
82              plt.subplot(2,10,k+1)
83              img=generator.predict(np.random.normal(0,1,(1,zdim)))
84              plt.imshow(img[0].reshape(28,28),cmap='gray')
85              plt.xticks([]); plt.yticks([])
86          plt.show()
87
88  imgs=generator.predict(np.random.normal(0,1,(50,zdim)))
89  plt.figure(figsize=(20,10))   # 학습을 마친 후 50개 샘플을 생성하여 출력
90  for i in range(50):
91      plt.subplot(5,10,i+1)
92      plt.imshow(imgs[i].reshape(28,28),cmap='gray')
93      plt.xticks([]); plt.yticks([])
```

10.3.2 생성 적대 신경망의 프로그래밍



10.3.2 생성 적대 신경망의 프로그래밍

학습을 마친 모델로 생성한 50개 샘플
(데이터가 긴 획으로 구성된다는 사실을
제대로 학습. 반 정도가 쓸만한 패턴)

10.4 응용 시나리오: 인공지능 패션 디자인

생성 모델을 활용하는 디자인 비즈니스

- 디자이너의 임무는 꾸준히 새로운 디자인을 생성하여 수익을 최대화
- 아이디어가 고갈되었을 때 생성 모델이 도움이 됨
  - 생성 모델로 디자인을 생성한 다음 쓸만한 것을 골라냄
- 디자이너가 없는 비즈니스도 가능
  - 고객이 생성 모델을 조작하여 고유한 디자인을 설계
  - 자동으로 생산 라인(로봇과 3D 프린터 구비)에 전달되어 생산이 이루어지고 자동 배송 시스템으로 넘어감
  - 세상에 단 하나뿐인 사용자 맞춤형 패션 제품

10.4.1 fashion MNIST 데이터로 생성 적대 신경망 학습

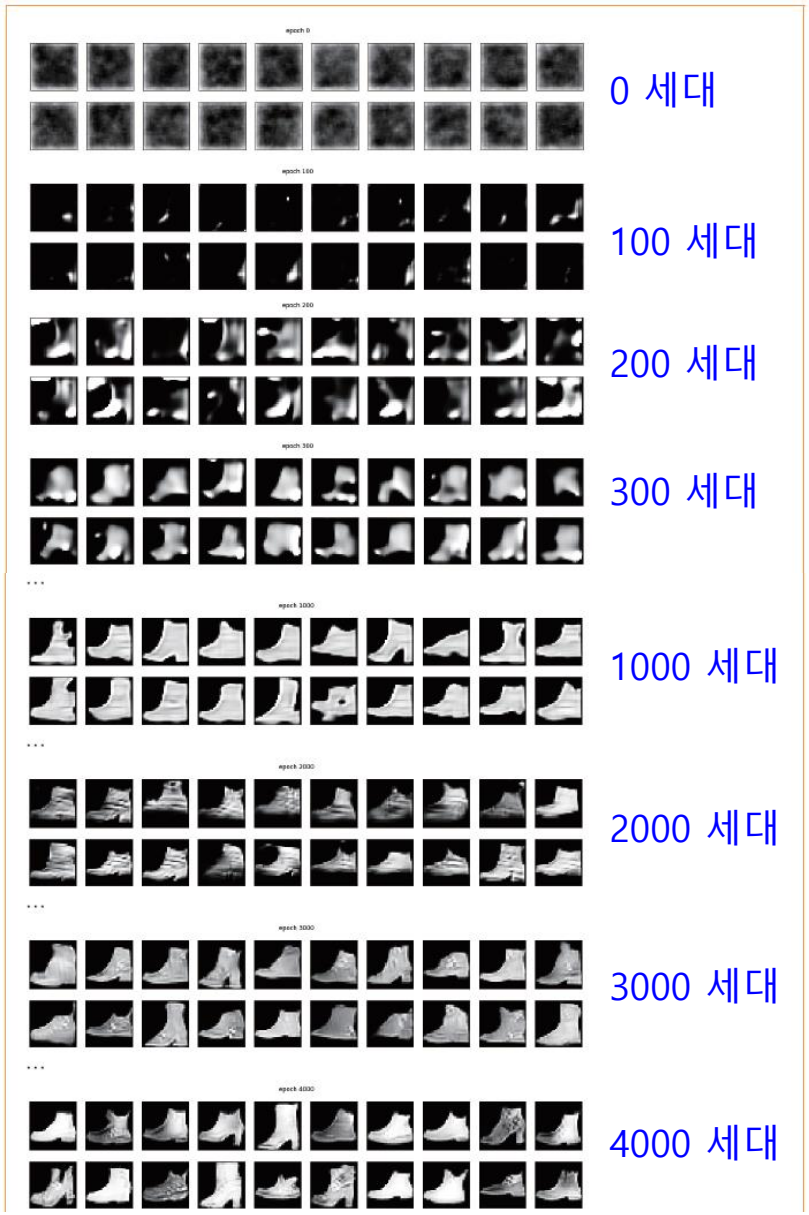
- fashion MNIST로 GAN을 학습하는 [프로그램 10-5(a)]
  - [프로그램 10-4]와 거의 비슷

프로그램 10-5(a) fashion MNIST 데이터를 활용한 생성 적대 신경망의 학습

```
01  import numpy as np
02  from tensorflow.keras.datasets import fashion_mnist
03  from tensorflow.keras.layers import Input,Activation,Dense,Flatten,Reshape,C
    onv2D,Conv2DTranspose,Dropout,BatchNormalization,UpSampling2D,LeakyReLU
04  from tensorflow.keras.models import Model
05  from tensorflow.keras import backend as K
06  from tensorflow.keras.losses import mse
07  import matplotlib.pyplot as plt
08
09  (x_train,y_train),(x_test,y_test)=fashion_mnist.load_data()
10  x_train=x_train[np.isin(y_train,[9])]              # 9번 부류는 ankle boot
11  x_train = (x_train.astype('float32')/255.0)*2.0-1.0  # [-1,1] 구간
12  x_test = (x_test.astype('float32')/255.0)*2.0-1.0
13  x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
14  x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
15
16
 …  [프로그램 10-4]의 15~93행
94
```

ankle boot 부류로 국한하여 학습

10.4.1 fashion MNIST 데이터로 생성 적대 신경망 학습



10.4.1 fashion MNIST 데이터로 생성 적대 신경망 학습

학습된 모델로 50개
샘플을 생성해봄

10.4.2 생성된 패턴의 품질 평가

- 대체적인 품질 평가
  - [프로그램 10-5(a)]는 그럴싸한 샘플을 생성해줌
- 체계적인 평가 방법 1: 유사한 진짜 샘플과 비교하기

프로그램 10-5(b) 생성된 샘플에 대해 훈련 집합에서 가장 가까운 샘플 찾기

```
95
96  # 훈련 집합 x_train에서 img와 가장 가까운 영상을 찾아주는 함수
97  def most_similar(img,x_train):
98      vmin=1.0e10
99      for i in range(len(x_train)):
100         dist=np.mean(np.abs(img-x_train[i]))
101         if dist<vmin:
102             imin,vmin=i,dist
103     return x_train[imin]
104
105 # 50개의 영상에 대해 가장 가까운 영상을 찾아 보여줌
106 plt.figure(figsize=(20,10))
107 for k in range(50):
108     plt.subplot(5,10,k+1)
109     plt.imshow(most_similar(imgs[k],x_train).reshape(28,28),cmap='gray')
110     plt.xticks([]); plt.yticks([])
111 plt.show()
```

10.4.2 생성된 패턴의 품질 평가

[프로그램 10-5(a)]로 생성된 가짜 샘플과 가장 비슷한 진짜 샘플

10.4.2 생성된 패턴의 품질 평가

비교하기 편하도록 나란히 배치

(a) 생성 적대 생성망이 생성한 샘플([프로그램 10-5(a)]의 출력)

(b) 훈련 집합 x_train에서 가장 가까운 샘플([프로그램 10-5(b)]의 출력)

그림 10-8 생성 적대 신경망이 생성한 샘플과 훈련 집합에서 가장 가까운 샘플

각 쌍은 유사하지만 같지는 않음
굽을 만들거나,
굽 두께에 변화를 주거나,
바닥에 패임을 추가하거나,
명암 변화를 적용 하는 등의 변화
→ 디자인에 새로운 요소 가미

10.4.2 생성된 패턴의 품질 평가

체계적인 평가 방법 2: 사람을 대상으로 생성 모델 평가

- 진짜와 가짜를 반씩 섞은 후 사람에게 진위 구별하게 하는 실험
  - 50% 정확률이라면 사람을 완벽하게 속인 셈
  - [그림 10-9]는 여러 모델에 대한 정확률(점수)

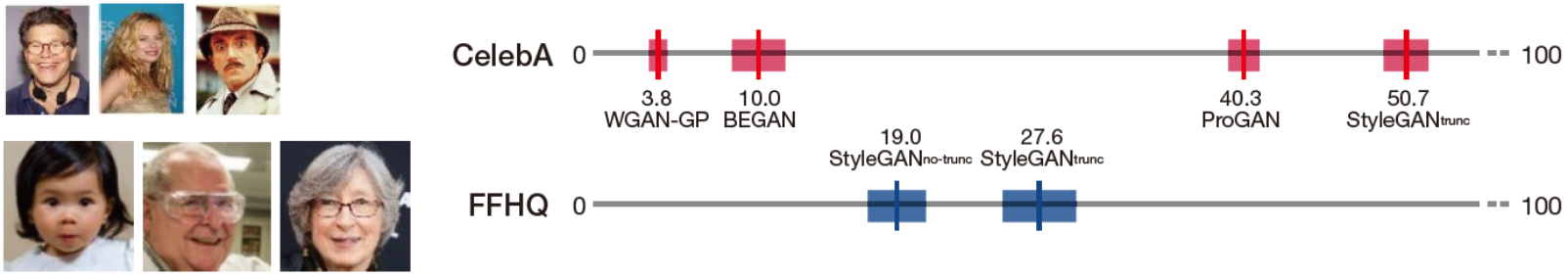


그림 10-9 생성 모델을 평가한 점수[Zhou2019]

10.4.2 생성된 패턴의 품질 평가

- Which face is real?
  - https://whichfaceisreal.com에 접속하면 자신이 평가자가 되어 실험해볼 수 있음

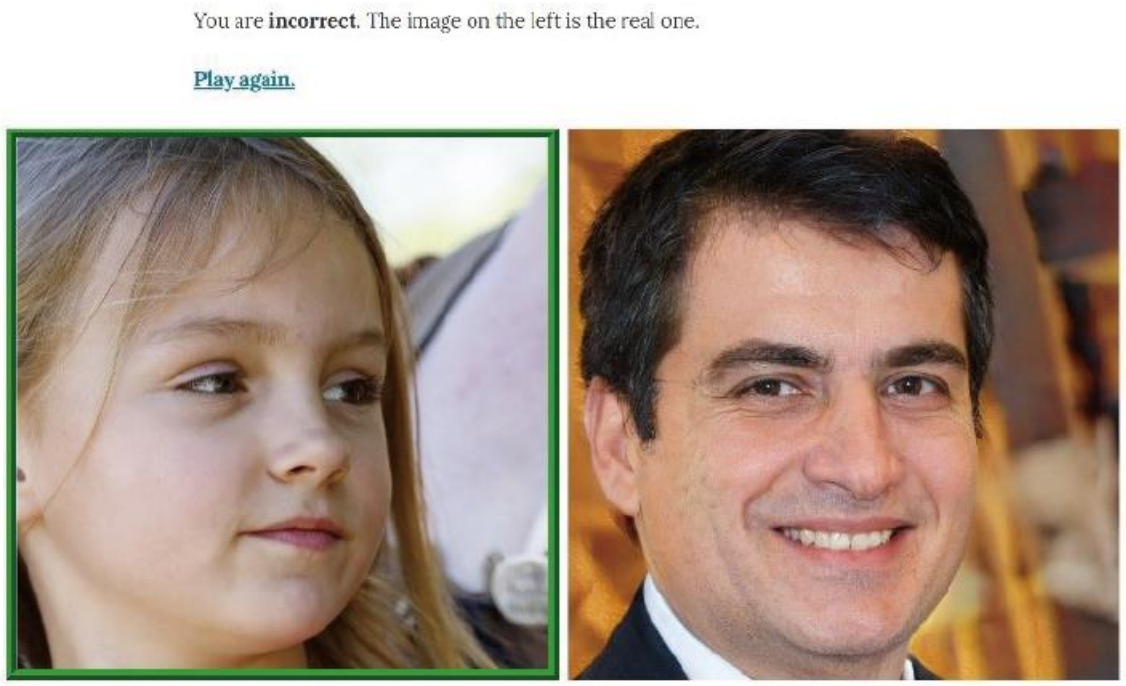


그림 10-10 'Which face is real?' 사이트에서 생성 모델 평가

10.5 생성 모델의 발전과 인공지능 창작

- 영상과 문학, 음악에서 큰 발전
  - 인공지능이 창작한 그림으로 전시회 열림
  - 시범적으로 시나 소설을 쓰는 인공지능 탄생
  - 인공지능이 작곡한 곡을 웹을 통해 사람들이 감상

10.5.1 생성 적대 신경망의 발전

간략한 역사

- 굿펠로의 2014년의 첫 GAN은 완전연결구조를 사용
- 완전연결층을 컨볼루션층으로 대치한 DCGAN(deep convolutional GAN)
- 모양을 제어하는 특징을 명시적으로 추가한 InfoGAN([그림 10-11])
- 저해상도에서 고해상도로 진행하는 ProGAN(Progressive GAN)([그림 10-12])
- 사진을 입력하고 화풍을 지정하면 해당 화풍으로 변환해주는 CycleGAN([그림 10-13])

10.5.1 생성 적대 신경망의 발전

그림 10-11 InfoGAN의 해석 가능한 모양 특징(왼쪽 c_1: 부류, 중간 c_2: 획 기울음, 오른쪽 c_3: 획 두께)[Chen2016]

그림 10-13 CycleGAN의 화풍 변환[Zhu2017]

10.5.1 생성 적대 신경망의 발전

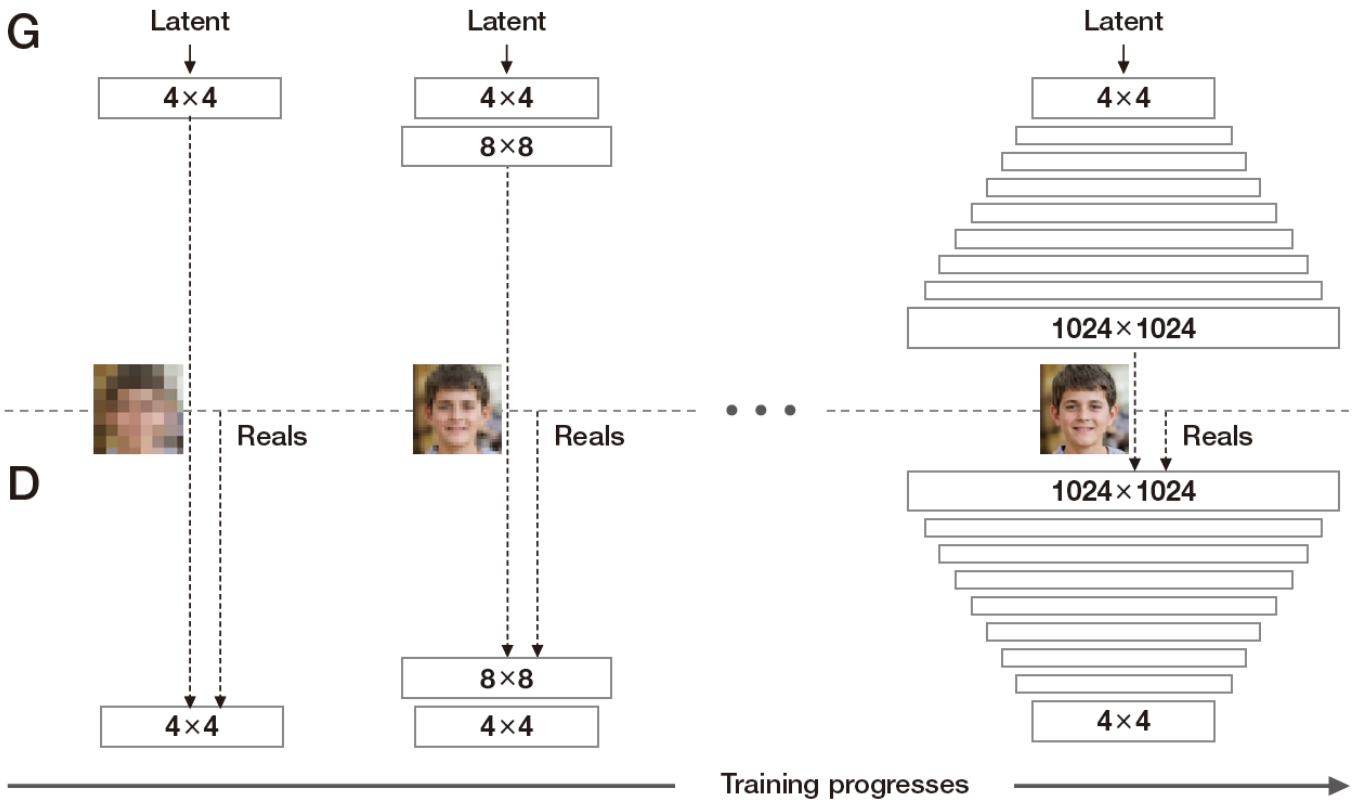


그림 10-12 고품질 영상을 생성하기 위한 ProGAN의 전략[Karras2018]

10.5.2 순서열 생성 모델의 발전

- 텍스트와 음악 생성에 적용
  - 기사 작성
    - "연합뉴스 국내 최초 머신러닝 AI 날씨 기사 선보여"(2020년 5월 4일자 한국기자협회의 기사)
  - 텍스트 생성에 사용되는 seq2seq 모델
    - 주의$_{attention}$ 기능 없는 단점

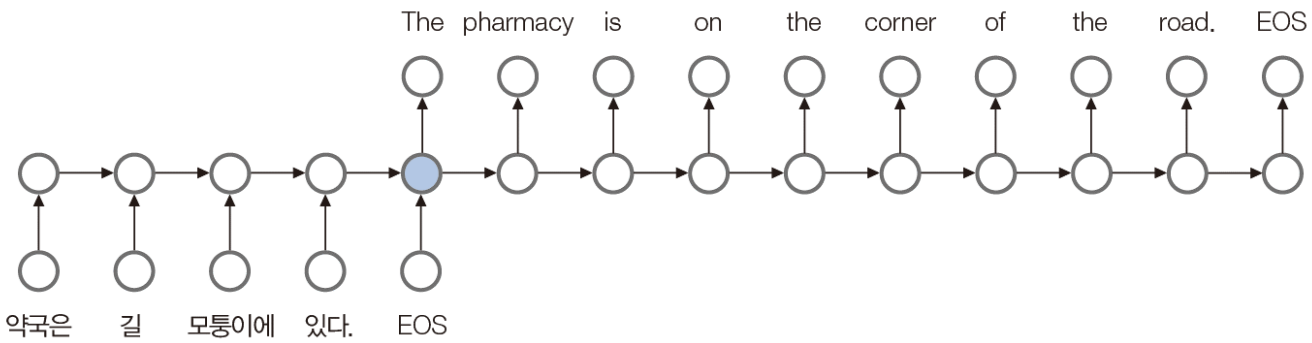


그림 10-14 seq2seq의 동작 원리[Sutskever2014]

  - 2017년에 주의 기능에 집중하는 트랜스포머 모델이 발표됨
    - 구글의 BERT
    - OpenAI 재단의 GPT-3