생성 모델과 창작

Preview

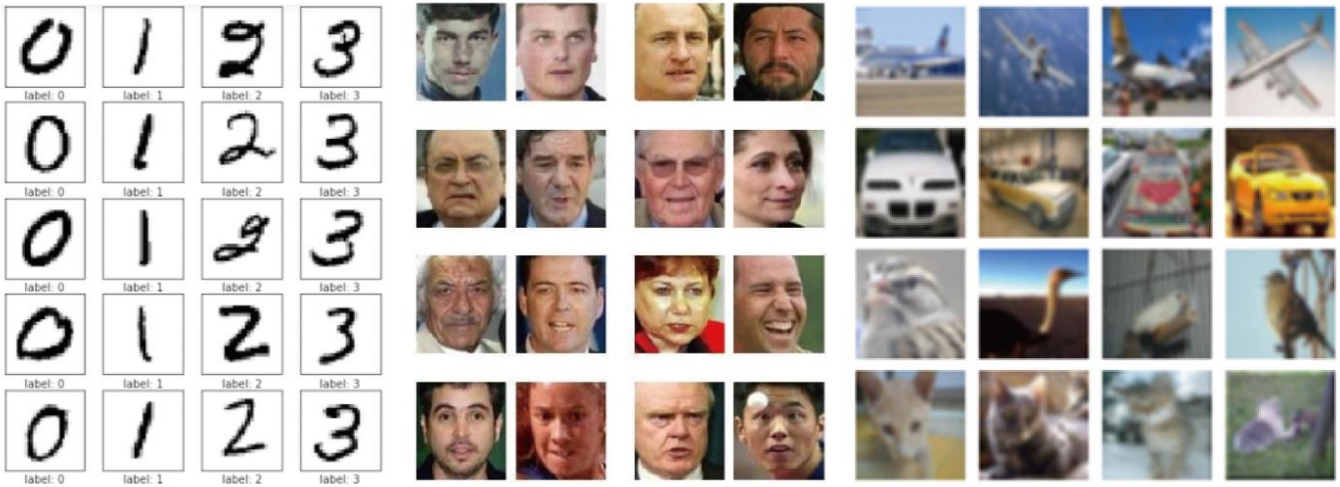
- 인간의 생성 능력
  - 예) 아이는 오늘 겪은 일을 아빠에게 이야기하고, 처음 가본 곳의 풍경을 그림으로 그림
    - 현실 세계를 비슷하게 모방하지만 같지는 않음(의도적 왜곡, 도구 한계로 추상화 등)
- 분별 모델과 생성 모델
  - 분별 모델
    - 가족의 얼굴을 알아보고 표정을 보고 상대의 감정을 알아보는 등의 능력
    - 인공지능은 분별 능력을 중심으로 발전해 옴. 앞서 공부한 SVM, 다층 퍼셉트론, 컨볼루션 신경망, LSTM, 강화 학습은 모두 분별 모델
  - 생성 모델
    - 사람의 필체를 흉내 내는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

10.1.2 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

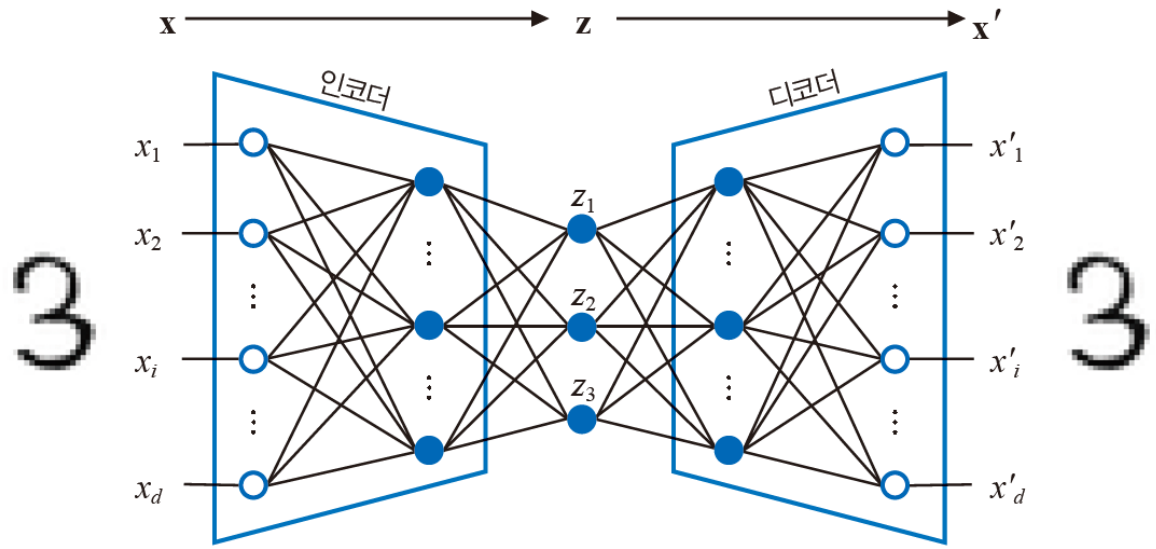


그림 10-4 오토인코더의 구조

- 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

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.init as init
import torchvision.datasets as dset
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torch.autograd import Variable
import matplotlib.pyplot as plt
%matplotlib inline

batch_size = 100
learning_rate = 0.0002
num_epoch = 1

mnist_train = dset.MNIST("./", train=True, transform=transforms.ToTensor(), target_transform=None, download=True)
mnist_test = dset.MNIST("./", train=False, transform=transforms.ToTensor(), target_transform=None, download=True)
train_loader = torch.utils.data.DataLoader(mnist_train,batch_size=batch_size, shuffle=True,num_workers=2,drop_last=True)
test_loader = torch.utils.data.DataLoader(mnist_test,batch_size=batch_size, shuffle=False,num_workers=2,drop_last=True)
```

10.2.2 오토인코더 프로그래밍

Encoder, Decoder 정의

```
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder,self).__init__()
        self.layer1 = nn.Sequential(
                        nn.Conv2d(1,16,3,padding=1),   # batch x 16 x 28 x 28
                        nn.ReLU(),
                        nn.BatchNorm2d(16),
                        nn.Conv2d(16,32,3,padding=1),  # batch x 32 x 28 x 28
                        nn.ReLU(),
                        nn.BatchNorm2d(32),
                        nn.Conv2d(32,64,3,padding=1),  # batch x 32 x 28 x 28
                        nn.ReLU(),
                        nn.BatchNorm2d(64),
                        nn.MaxPool2d(2,2)   # batch x 64 x 14 x 14
        )
        self.layer2 = nn.Sequential(
                        nn.Conv2d(64,128,3,padding=1),  # batch x 64 x 14 x 14
                        nn.ReLU(),
                        nn.BatchNorm2d(128),
                        nn.MaxPool2d(2,2),
                        nn.Conv2d(128,256,3,padding=1),  # batch x 64 x 7 x 7
                        nn.ReLU()
        )
    def forward(self,x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(batch_size, -1)
        return out

encoder = Encoder().cuda()
```

뽑아낸 Feature를 1차원으로 reshape

```
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder,self).__init__()
        self.layer1 = nn.Sequential(
                        nn.ConvTranspose2d(256,128,3,2,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(128),
                        nn.ConvTranspose2d(128,64,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(64)
        )
        self.layer2 = nn.Sequential(
                        nn.ConvTranspose2d(64,16,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(16),
                        nn.ConvTranspose2d(16,1,3,2,1,1),
                        nn.ReLU()
        )
    def forward(self,x):
        out = x.view(batch_size,256,7,7)
        out = self.layer1(out)
        out = self.layer2(out)
        return out

decoder = Decoder().cuda()
```

받아낸 input값을 4차원으로 reshape

10.2.2 오토인코더 프로그래밍

```
parameters = list(encoder.parameters())+ list(decoder.parameters())
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(parameters, lr=learning_rate)

for i in range(num_epoch):
    for j,[image,label] in enumerate(train_loader):
        optimizer.zero_grad()

        image = Variable(image).cuda()
        output = encoder(image)
        output = decoder(output)
        loss = loss_func(output,image)

        loss.backward()
        optimizer.step()
        if j % 50 == 0:
            print("[Epcoh {}/{}] MSE Loss: {:.5f}".format(i+1,num_epoch,loss.item()))
    val_loss = 0
    for j,[image,label] in enumerate(test_loader):
        image = Variable(image,requires_grad=False).cuda()
        output = encoder(image)
        output = decoder(output)
        val_loss += loss_func(output,image).item()
    print("[Epcoh {}/{}] Validation MSE Loss: {:.5f}".format(i+1,num_epoch,val_loss/len(test_loader)))
```

인코더와 디코더의 parameters들을 합친걸 Optimizer에 넣어줌

Encoder의 output을 Decoder의 input으로 넣어줌

Decoder의 output과 image dataset간의 차이를 구함

10.2.2 오토인코더 프로그래밍

```
Encoder(
  (layer1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU()
  )
)
Decoder(
  (layer1): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (layer2): Sequential(
    (0): ConvTranspose2d(64, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ConvTranspose2d(16, 1, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
    (4): ReLU()
  )
)
```

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

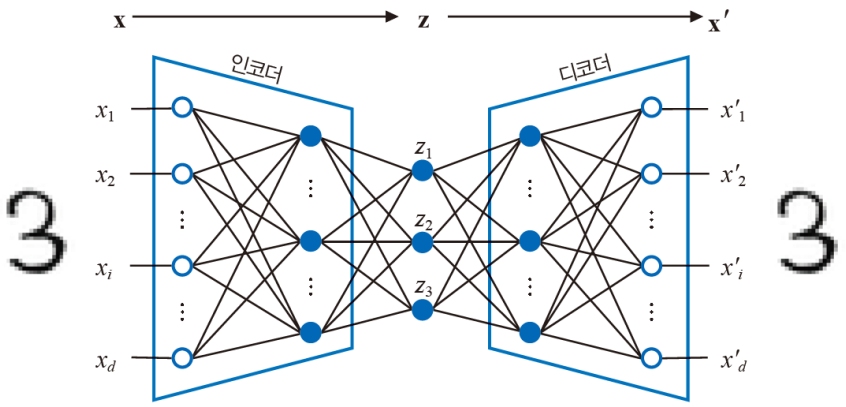


그림 10-4 오토인코더의 구조

10.2.3 생성 모델로서 오토인코더

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

```
## 생성 실험 1: 첫번째 샘플의 input image에 잡음을 섞어 새로운 샘플 생성
del encoder
del decoder

encoder = Encoder().cuda()
decoder = Decoder().cuda()

parameters = list(encoder.parameters())+ list(decoder.parameters())
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(parameters, lr=learning_rate)

for epoch in range(num_epoch):
    for j,[image,label] in enumerate(train_loader):
        noise = init.normal_(torch.FloatTensor(batch_size,1,28,28),0,0.1)
        noise = Variable(noise.cuda())
        optimizer.zero_grad()
        image = Variable(image).cuda()
        noise_image = image + noise
        output = encoder(noise_image)
        output = decoder(output)
        loss = loss_func(output,image)
        loss.backward()
        optimizer.step()
        if j % 50 == 0:
            print("[Epcoh {}/{}] MSE Loss: {:.5f}".format(epoch+1,num_epoch,loss.item()))

    val_loss = 0
    for j,[image,label] in enumerate(test_loader):
        noise = init.normal_(torch.FloatTensor(batch_size,1,28,28),0,0.1)
        noise = Variable(noise.cuda())
        optimizer.zero_grad()
        image = Variable(image,requires_grad=False).cuda()
        noise_image = image + noise
        output = encoder(noise_image)
        output = decoder(output)
        val_loss += loss_func(output,image).item()
    print("[Epcoh {}/{}] Validation MSE Loss: {:.5f}".format(epoch+1,num_epoch,val_loss/len(test_loader)))
```

10.2.3 생성 모델로서 오토인코더

- 오토인코더로 새로운 샘플 생성
  - [프로그램 10-2(b)]는 학습된 디코더로 새로운 샘플을 생성하는 프로그램
  - 생성 실험 1 시각화

```
print(('Original Images'))
fig, axes = plt.subplots(2, 5)
fig.set_size_inches(12, 6)
for i in range(10):
    axes[i//5, i%5].imshow(image[i].cpu().detach().numpy().reshape(28, 28), cmap='gray')
plt.tight_layout()
plt.show()

print('')

print(('Auto Encoder Images'))
fig, axes = plt.subplots(2, 5)
fig.set_size_inches(12, 6)
for i in range(10):
    axes[i//5, i%5].imshow(output[i].cpu().detach().numpy().reshape(28, 28), cmap='gray')
plt.tight_layout()
plt.show()
```

10.2.3 생성 모델로서 오토인코더

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

```
## 생성 실험 2: 첫번째 샘플의 Feature 공간 표현에 잡음을 섞어 새로운 샘플 생성
del encoder
del decoder

encoder = Encoder().cuda()
decoder = Decoder().cuda()

parameters = list(encoder.parameters())+ list(decoder.parameters())
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(parameters, lr=learning_rate)

for epoch in range(num_epoch):
    for j,[image,label] in enumerate(train_loader):
        optimizer.zero_grad()
        image = Variable(image).cuda()
        output = encoder(image)
        noise = init.normal_(torch.FloatTensor(batch_size,12544),0,0.1)
        noise = Variable(noise.cuda())
        noise_output = output + noise
        output = decoder(noise_output)
        loss = loss_func(output,image)
        loss.backward()
        optimizer.step()
        if j % 50 == 0:
            print("[Epcoh {}/{}] MSE Loss: {:.5f}".format(epoch+1,num_epoch,loss.item()))

    val_loss = 0
    for j,[image,label] in enumerate(test_loader):
        optimizer.zero_grad()
        image = Variable(image,requires_grad=False).cuda()
        noise = init.normal_(torch.FloatTensor(batch_size,12544),0,0.1)
        noise = Variable(noise.cuda())
        output = encoder(image)
        noise_output = output + noise
        output = decoder(noise_output)
        val_loss += loss_func(output,image).item()
    print("[Epcoh {}/{}] Validation MSE Loss: {:.5f}".format(epoch+1,num_epoch,val_loss/len(test_loader)))
```

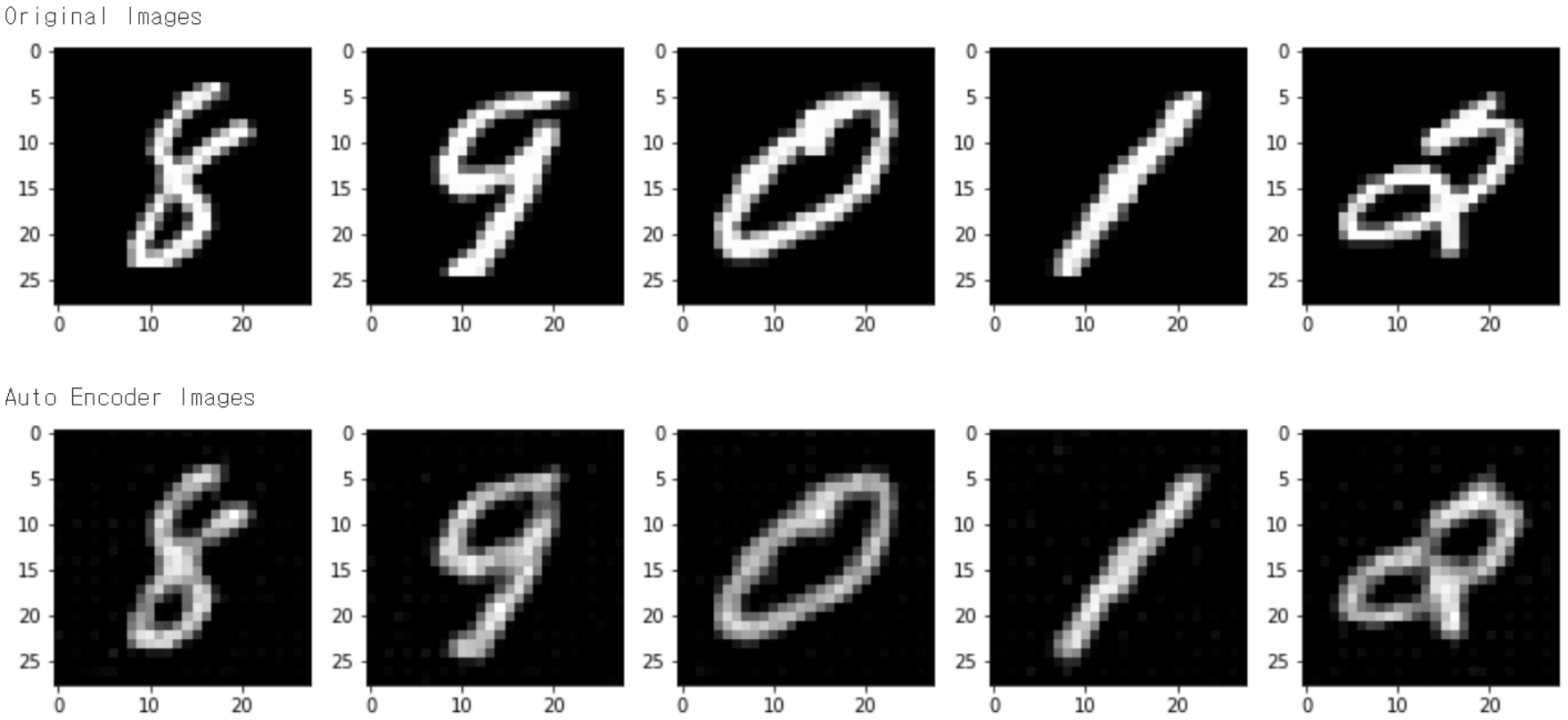
10.2.3 생성 모델로서 오토인코더

- 오토인코더로 새로운 샘플 생성
  - [프로그램 10-2(b)]는 학습된 디코더로 새로운 샘플을 생성하는 프로그램
  - 생성 실험 2 시각화 (코드 동일)

```
print(('Original Images'))
fig, axes = plt.subplots(2, 5)
fig.set_size_inches(12, 6)
for i in range(10):
    axes[i//5, i%5].imshow(image[i].cpu().detach().numpy().reshape(28, 28), cmap='gray')
plt.tight_layout()
plt.show()

print('')

print(('Auto Encoder Images'))
fig, axes = plt.subplots(2, 5)
fig.set_size_inches(12, 6)
for i in range(10):
    axes[i//5, i%5].imshow(output[i].cpu().detach().numpy().reshape(28, 28), cmap='gray')
plt.tight_layout()
plt.show()
```

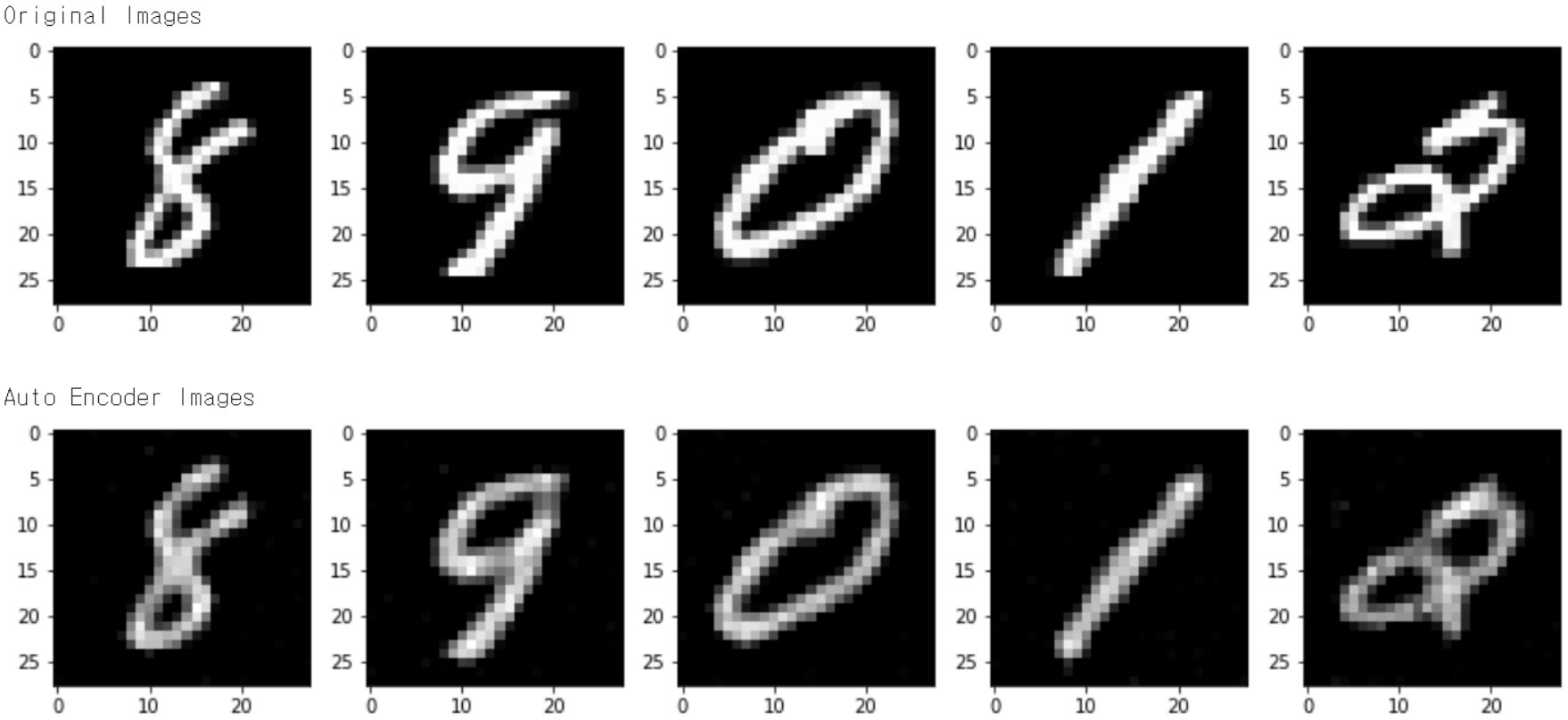
10.2.3 생성 모델로서 오토인코더

- 실험 1의 결과



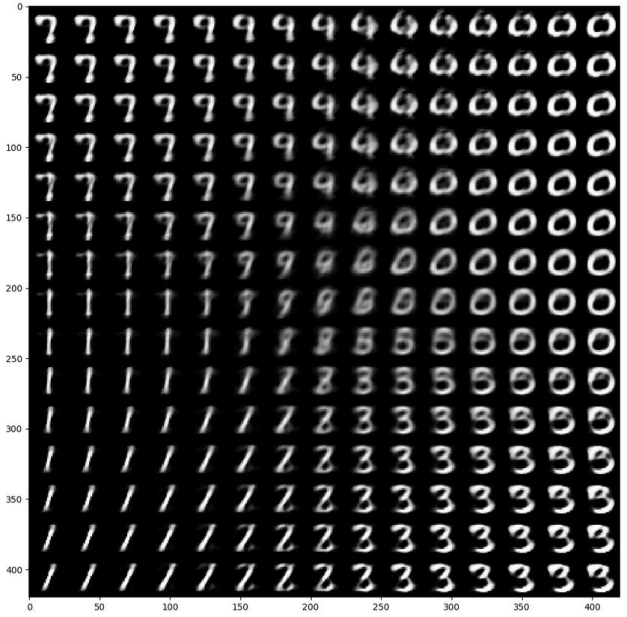
10.2.3 생성 모델로서 오토인코더

- 실험 2의 결과



10.2.3 생성 모델로서 오토인코더

- [프로그램 10-2(b)]가 생성한 샘플의 품질 평가
  - 실험 1: 7과 비슷한 샘플이 생성됨. 잡음을 심하게 넣으면, 패턴이 왜곡되기 시작하고 잡음이 제대로 decode가 됐다고 보기 힘든 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

```
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder,self).__init__()
        self.layer1 = nn.Sequential(
                        nn.Conv2d(1,16,3,padding=1),   # batch x 16 x 28 x 28
                        nn.ReLU(),
                        nn.BatchNorm2d(16),
                        nn.Conv2d(16,32,3,padding=1),  # batch x 32 x 28 x 28
                        nn.ReLU(),
                        nn.BatchNorm2d(32),
                        nn.Conv2d(32,64,3,padding=1),  # batch x 32 x 28 x 28
                        nn.ReLU(),
                        nn.BatchNorm2d(64),
                        nn.MaxPool2d(2,2)   # batch x 64 x 14 x 14
        )
        self.layer2 = nn.Sequential(
                        nn.Conv2d(64,128,3,padding=1),  # batch x 64 x 14 x 14
                        nn.ReLU(),
                        nn.BatchNorm2d(128),
                        nn.MaxPool2d(2,2),
                        nn.Conv2d(128,256,3,padding=1),  # batch x 64 x 7 x 7
                        nn.ReLU()
        )
        self.fc = nn.Linear(12544,2)
    def forward(self,x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(batch_size, -1)
        out = self.fc(out)
        return out
```

Fc를 마지막에 붙여 2차원 분포값만 남김
Decoder에서는 반대로 맨앞단에 Linear(2,12544)

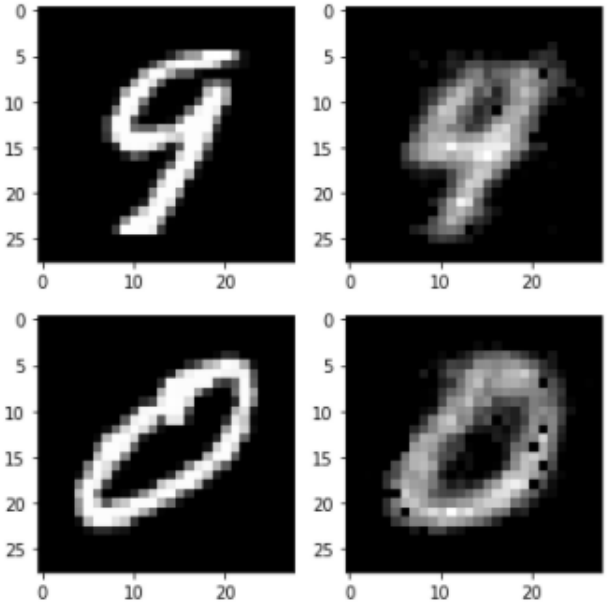
10.2.4 2차원 잠복 공간 관찰

기존 트레이닝 코드(epoch문 안)

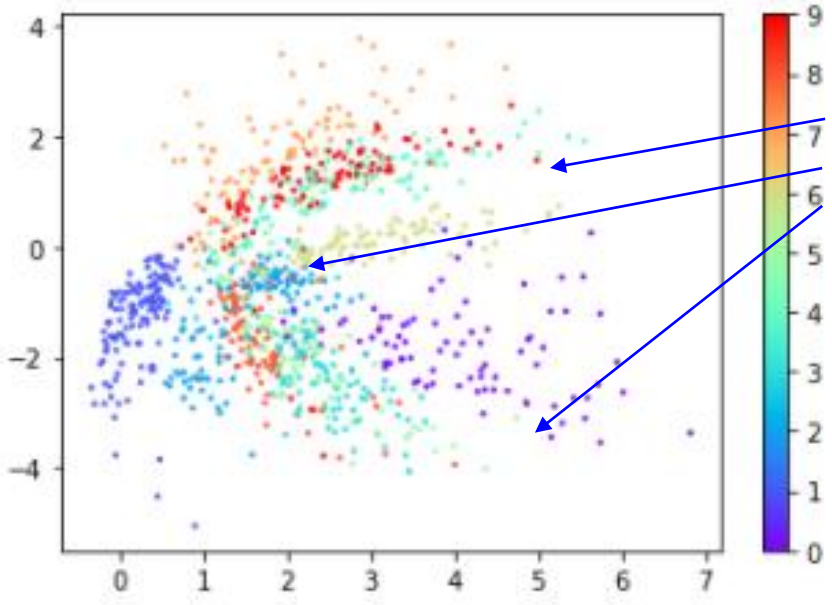
```
if(i+1==num_epoch):
    for j,[image,label] in enumerate(test_loader):
      image = Variable(image,requires_grad=False).cuda()
      encoded_output = encoder(image)
      if(j == 0):
        encoded = encoded_output
        labels = label
      elif(encoded.size(0)<1000):
        encoded = torch.cat((encoded_output,encoded),0)
        labels = torch.cat((label,labels),0)
```

```
out_img = torch.squeeze(output.cpu().data)
fig = [plt.figure() for _ in range(5)]
rows = 1
cols = 2
for i in range(5):
    ax1 = fig[i].add_subplot(rows, cols, 1)
    ax1.imshow(torch.squeeze(image[i].cpu()).data.numpy(),cmap='gray')
    ax2 = fig[i].add_subplot(rows, cols, 2)
    ax2.imshow(out_img[i].numpy(),cmap='gray')

plt.show()

n=1000
encoded = encoded.cpu().detach().numpy()
sc = plt.scatter(encoded[:,0],encoded[:,1],s=2,c=labels,cmap='rainbow')
plt.colorbar(sc)
```

10.2.4 2차원 잠복 공간 관찰



잠복 공간을 2차원으로
축소하여 품질 저하



같은 부류가 군집화
밀집도가 높은 공간이 존재
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

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

10.3.1 동기와 원리

GAN의 원리

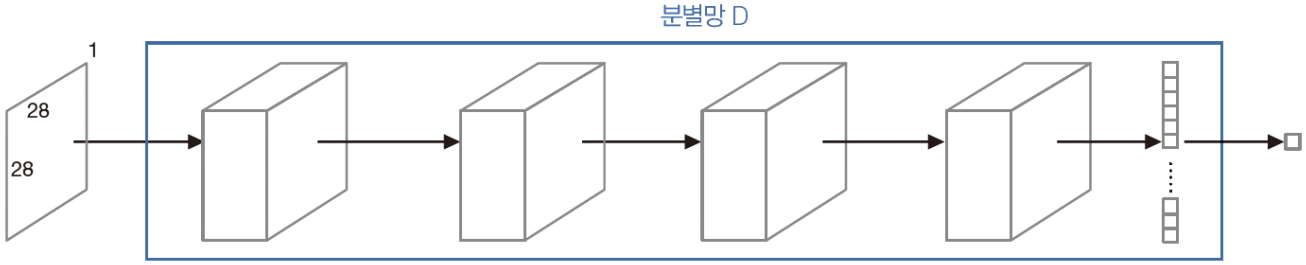
- 생성망 G와 분별망 D라는 두 개의 대립 관계의 신경망을 사용
  - G는 D를 속일 수 있을 정도로 품질이 높는 가짜 샘플을 생성
  - D는 G가 만든 가짜 샘플을 높은 정확률로 맞힘

그림 10-5 생성 적대 신경망에서 생성망과 분별망의 적대적 관계

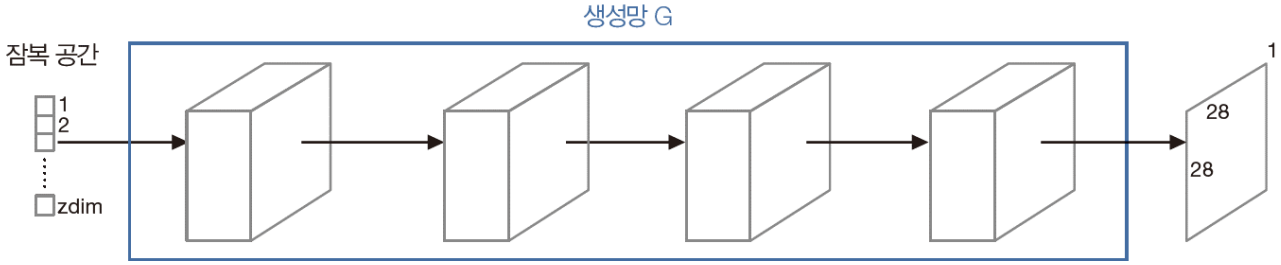
- 위조지폐범과 경찰에 비유
  - 현실 세계와 달리 위조지폐범에 해당하는 생성망이 승리해야 함

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



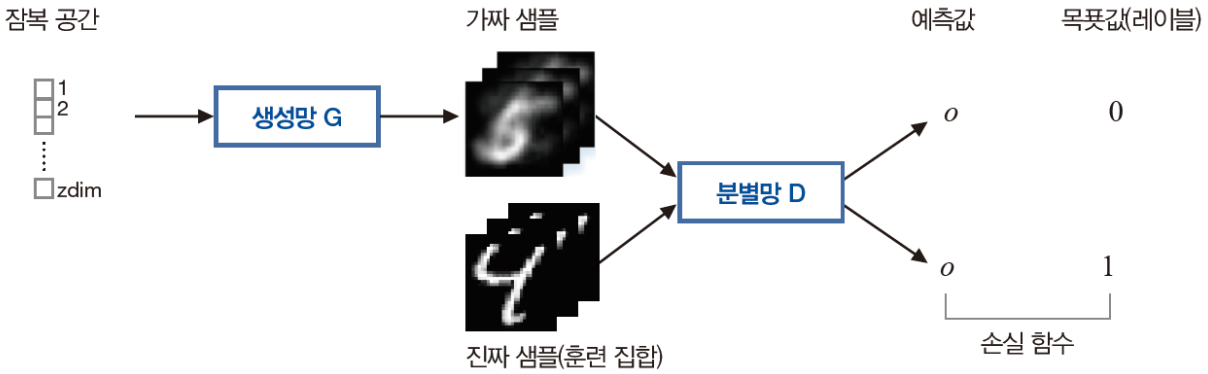
(b) 생성망 G

그림 10-6 생성 적대 신경망의 분별망과 생성망의 구조

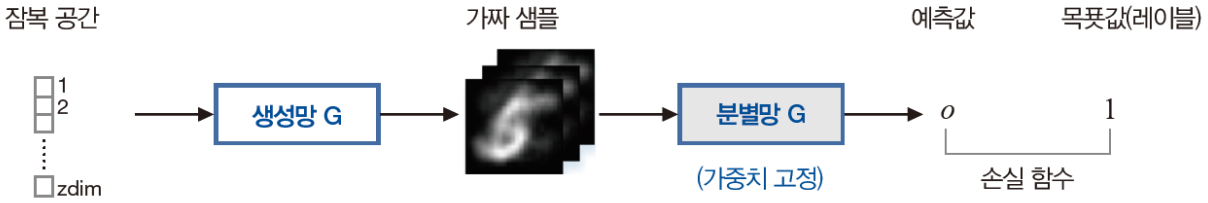
10.3.1 동기와 원리

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]

```
# Vanilla GAN
import torch
import torch.nn as nn
import torch.utils as utils
import torch.nn.init as init
from torch.autograd import Variable
import torchvision.utils as v_utils
import torchvision.datasets as dset
import torchvision.transforms as transforms
import numpy as np
import matplotlib.pyplot as plt
from collections import OrderedDict

# Set Hyperparameters
epochs = 250
batch_size = 512
learning_rate = 0.0002
z_size = 50
middle_size = 200
mnist_train = dset.MNIST("./", train=True, transform=transforms.ToTensor(), target_transform=None, download=True)
train_loader = torch.utils.data.DataLoader(dataset=mnist_train,batch_size=batch_size,shuffle=True,drop_last=True)
```

10.3.2 생성 적대 신경망의 프로그래밍

- MNIST를 가지고 GAN을 구현하는 [프로그램 10-4]

fcBlock 정의

```
class fcBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(fcBlock, self).__init__()
        self.body = nn.Sequential(
          nn.Linear(in_channels, out_channels),
          nn.LeakyReLU(0.2)
        )
    def forward(self, x):
        out = self.body(x)
        return out
```

Random noise z를 input으로 받아서 1x28x28 이미지 생성

```
# Generator receives random noise z and create 1x28x28 image
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.n_features = 50
        self.n_out = 784
        self.fc0 = fcBlock(self.n_features,256)
        self.fc1 = fcBlock(256,512)
        self.fc2 = fcBlock(512,1024)
        self.fc3 = fcBlock(1024,self.n_out)
    def forward(self, x):
        x = self.fc0(x)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        x = x.view(-1, 1, 28, 28)
        return x

generator=Generator().cuda()
print(generator)
```

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.n_in = 784
        self.n_out = 1
        self.fc0 = fcBlock(self.n_in,1024)
        self.fc1 = fcBlock(1024,512)
        self.fc2 = fcBlock(512,256)
        self.fc3 = fcBlock(256,self.n_out)
        self.dropout = nn.Dropout(0.3)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = x.view(-1, 784)
        x = self.dropout(self.fc0(x))
        x = self.dropout(self.fc1(x))
        x = self.dropout(self.fc2(x))
        x = self.fc3(x)
        x = self.sigmoid(x)
        return x

discriminator=Discriminator().cuda()
print(discriminator)
```

진짜(1)와 가짜(0)를 구별하려고 sigmoid 사용

10.3.2 생성 적대 신경망의 프로그래밍

- MNIST를 가지고 GAN을 구현하는 [프로그램 10-4]

```
loss_func = nn.BCELoss()
gen_optim = torch.optim.Adam(generator.parameters(), lr=learning_rate)
dis_optim = torch.optim.Adam(discriminator.parameters(), lr=learning_rate)

ones_label = Variable(torch.ones(batch_size,1)).cuda()
zeros_label = Variable(torch.zeros(batch_size,1)).cuda()

generator.train()
discriminator.train()

# train
for epoch in range(epochs):
    for _,(image,label) in enumerate(train_loader):
        image = Variable(image).cuda()

        # discriminator
        dis_optim.zero_grad()
        z = Variable(init.normal_(torch.Tensor(batch_size,z_size),mean=0,std=0.1)).cuda()
        gen_fake = generator(z)
        dis_fake = discriminator(gen_fake)

        dis_real = discriminator(image)
        dis_loss = torch.sum(loss_func(dis_fake,zeros_label)) + torch.sum(loss_func(dis_real,ones_label))
        dis_loss.backward(retain_graph=True)
        dis_optim.step()

        # generator
        gen_optim.zero_grad()
        z = Variable(init.normal_(torch.Tensor(batch_size,z_size),mean=0,std=0.1)).cuda()
        gen_fake = generator(z)
        dis_fake = discriminator(gen_fake)

        gen_loss = torch.sum(loss_func(dis_fake,ones_label)) # fake classified as real
        gen_loss.backward()
        gen_optim.step()
```

Generator/Discriminator
각각에 대해 Optimizer 정의

Random Noise z 생성 후 generator에서 fake_image 생성

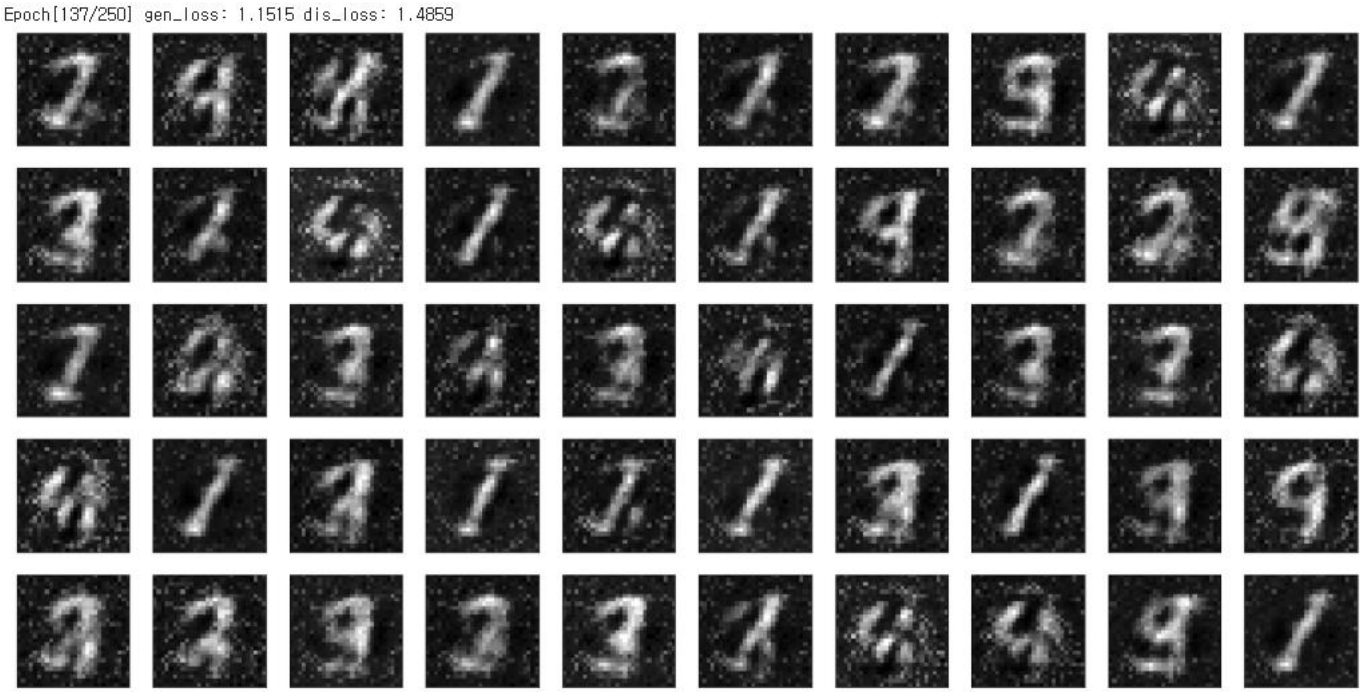
Fake label / Real label 사용해서 더한 결과간 차이 학습

Discriminator에서 준 낸 결과를 기반으로 차이 학습

10.3.2 생성 적대 신경망의 프로그래밍

```
# eval
print("Epoch[{}/{}] gen_loss: {:.4f} dis_loss: {:.4f}".format(epoch,epochs,gen_loss.data,dis_loss.data))
plt.figure(figsize=(20,10))
gen_fake = gen_fake.cpu().detach().numpy()
for i in range(50):
  plt.subplot(5,10,i+1)
  plt.imshow(gen_fake[i,:,:,:].reshape(28,28),cmap='gray')
  plt.xticks([]);plt.yticks([])
plt.show()
```

Fake generated image plot하기



10.4 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

10.4.1 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])
- 영상간 변화 task에 대한 범용적 모델인 Pix2pix (Image to Image Translation)

10.4.1 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]

10.4.1 생성 적대 신경망의 발전

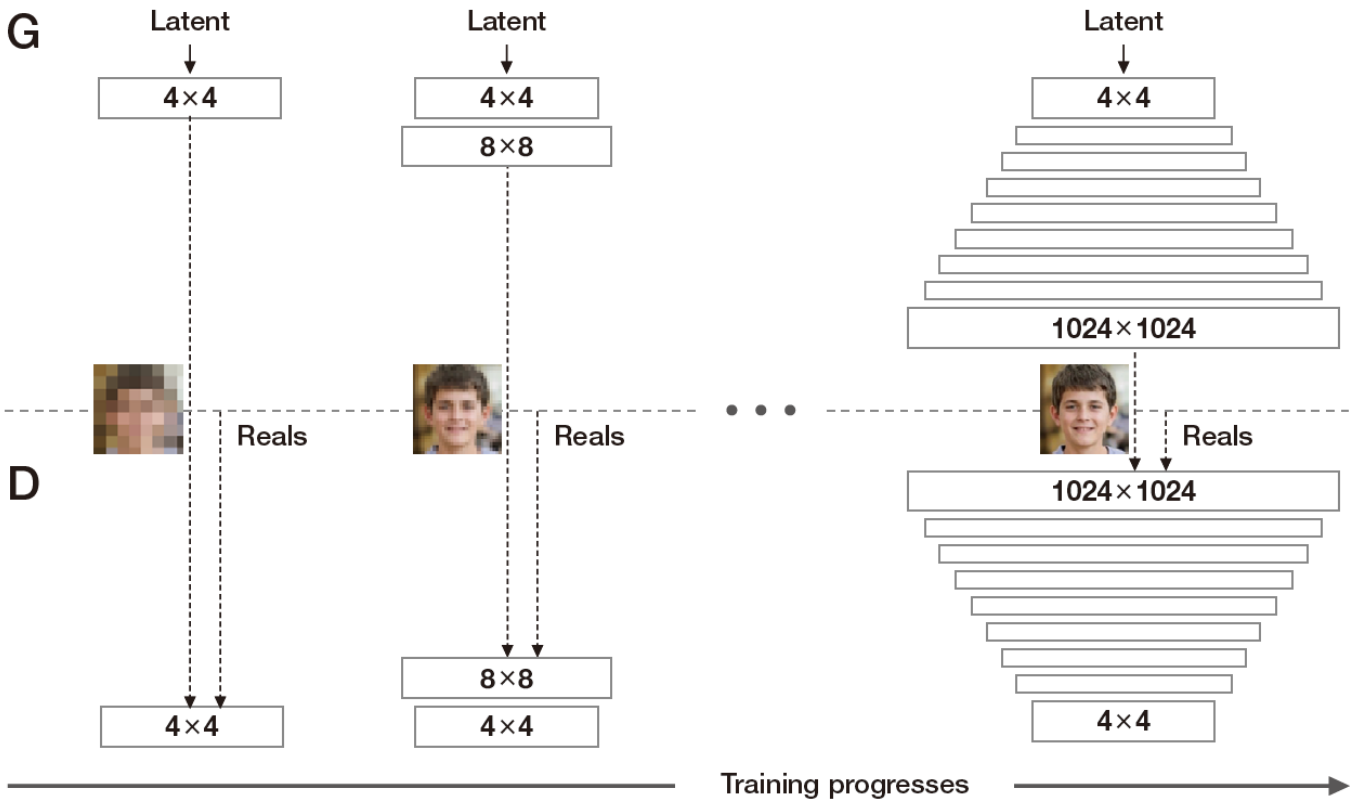


그림 10-12 고품질 영상을 생성하기 위한 ProGAN의 전략[Karras2018]

https://medium.com/@EricKuy/progressive-gan-dd7cadb17dc4

10.4.1 생성 적대 신경망의 발전

- Pix2pix (Phillip, 2017)
  - 대부분의 영상변환에 범용적으로 동작하는 GAN 기반 영상 변환 딥러닝 모델 제안
  - 발표자료 [Link]