10. GAN(Generative Adversarial Nets)

NIPS 2016에서 가장 hot한 키워드를 하나 꼽으라 한다면 Generative Adversarial Network(GAN)이라고 단언할 수 있을 정도로 2014년에 Ian Goodfellow가 NIPS에서 발표한 paper에서 시작한 GAN 붐은 이제 딥러닝의 대세가 되어가고 있다.

facebook의 Yann Lecun 교수님이 근 10-20년 간에 기계학습 나온 아이디어 중 최고라고 하였으며, 이미지 생성(image generation) 외에도 natural language processing(NLP) 등 다양한 분야에서 엄청난 성과들을 보여주고 있다.

가. GAN의 개념

"Adversarial"이란 단어의 사전적 의미를 보면 대립하는, 적대하는 란 뜻을 갖는다. 대립하려면 어찌 되었든 상대가 있어야하니 GAN은 크게 두 부분으로 나누어져 있다는 것을 먼저 직관적으로 알 수 있다. Image를 만들어내는 생성자와(Generator) 이렇게 만들어진 무언가를 평가하는 평가자(Discriminator)가 있어서 서로 대립(Adversarial)하며 서로의 성능을 점차 개선해 나가자는 것이 주요 개념이다. Ian Goodfellow가 본문에서 예시로 든 것은 지폐위조범과 경찰인데 상당히 재미있고 직관적으로 이해가 되는 예이다.

지폐 위조범(Generator)은 경찰을 최대한 열심히 속이려고 하고 다른 한편에서는 경찰(Discriminator)이 이렇게 위조된 지폐를 진짜와 감별하려고(Classify) 노력한다. 이런 경쟁 속에서 두 그룹 모두 속이고 구별하는 서로의 능력이 발전하게 되고 결과적으로는 진짜 지폐와 위조 지폐를 구별할 수 없을 정도(구별할 확률 p d =0.5)에 이른다는 것.

위의 예를 수학적인 용어를 섞어 쓰면 다음과 같다. Generative model G는 우리가 갖고 있는 data x의 distribution을 알아내려고 노력합니다. 만약 G가 정확히 data distribution을 모사할 수 있다면 거기서 뽑은 sample은 완벽히 data와 구별할 수 없게 된다. 한편 discriminator model D는 현재 자기가 보고 있는 sample이 training data에서 온 것(진짜)인 지, 아니면 G로부터 만들어진 것인 지를 구별하여 각각의 경우에 대한 확률을 estimate한다.

따라서 아래 그림 10-1을 보면 알 수 있듯이 D의 입장에서는 data로부터 뽑은 sample x 는 $D(x)=1$이되고, G에 임의의 noise distribution으로부터 뽑은 input z를 넣고 만들어진 sample에 대해서는 $D(G(z))=0$이 되도록 노력한다. 즉, D는 실수할 확률을 낮추기(minimize) 위해 노력하고 반대로 G는 D가 실수할 확률을 높이기(maximize) 위해 노력하는데, 따라서 둘을 같이 놓고 보면 "minimax two-player game or minimax problem"이라 할 수 있다.

여기서 잘 살펴보면, G와 D는 사실 꼭 neural network로 만들 필요가 없습니다. 어떤 model이든 이 역할을 서로 "잘" 해줄 수만 있다면 상관이 없지만, neural network를 사용한 것이 여러모로 실제 적용 시 여러 장점이 있기에(물론 단점도 있지만) 그리고 결과도 꽤 잘 나오기에 결국 Generative Adversarial Nets가 된 것이다.

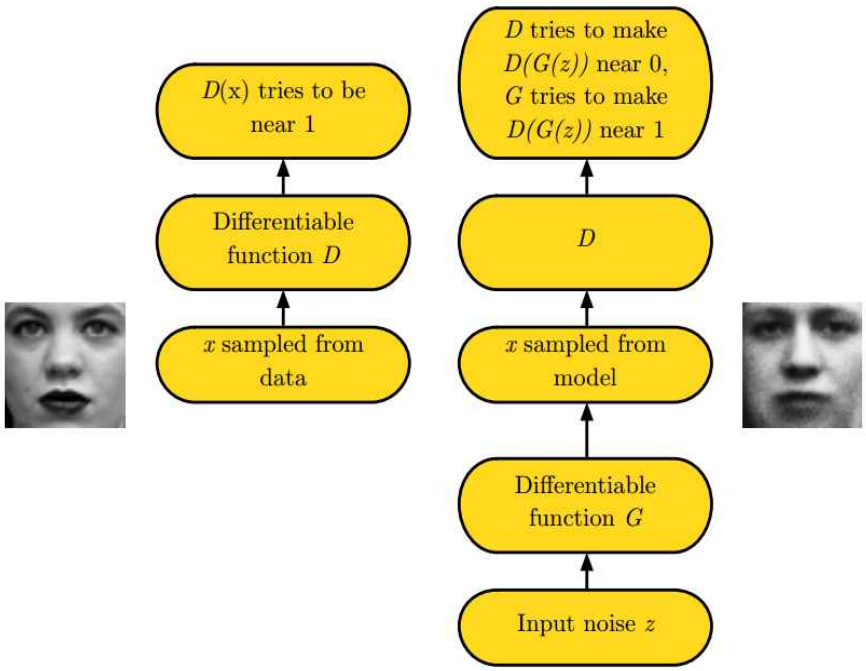

그림 10.1 GAN의 개념

나. Adversarial Nets

일단 G와 D가 둘 다 multilayer perceptrons model을 사용하면, Generator's distribution P_g over data x를 학습하기 위해 generator의 input으로 들어갈 noise variables $P_z(z)$에 대한 prior를 정의하고, data space로의 mapping을 $G(z;\theta_g)$라 표현할 수 있다. 여기서 G는 미분 가능한 함수로써 θ_g를 parameter로 갖는 multilayer perceptron이다. 한편, Discriminator 역시 multilayer perceptron으로 $D(x;\theta_d)$로 나타내며 output은 single scalar 값이 된다(확률이므로). $D(x)$는 x가 P_g가 아닌 data distribution으로부터 왔을 확률을 나타낸다. 따라서, 이를 수식으로 정리하면 다음과 같은 value function $V(G,D)$에 대한 minimax problem을 푸는 것과 같다.

$$\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_{data}(z)}[\log(1 - D(G(z)))] \quad (10\text{-}1)$$

극단적인 예시를 통해 이해해보자. 먼저 가장 이상적인 상황에서의 D입장을 생각해 보면, D는 아주 잘 구별을 하는 녀석이므로 D가 보는 sample x가 실제로 data distribution으로부터 온 녀석이라면 $D(x)=1$이므로 첫 번째 term에서 log값이 사라지고 $G(z)$가 만들어낸 녀석이라면 $D(G(z))=0$이므로 두 번째 term 역시 0으로 사라진다. 이때가 D의 입장에서

V의 "최대값"을 얻을 수 있다는 것은 자명하다. 반대로 G 의 입장에서 생각해봐도 상황은 비슷하다.

아래 그림 10-2는 논문(NIPS 2014)에서 빌려온 그림이다. 위에 설명한 내용을 아주 이해하기 좋게 잘 그려놓았는데 부연을 좀 하자면, 먼저 검은 점선이 data generating distribution, 파란 점선이 discriminator distribution, 녹색 선이 generative distribution 이다. 밑에 x와 z선은 각각 x와 z의 domain을 나타내며, 위로 뻗은 화살표가 $x = G(z)$의 mapping을 보여준다.

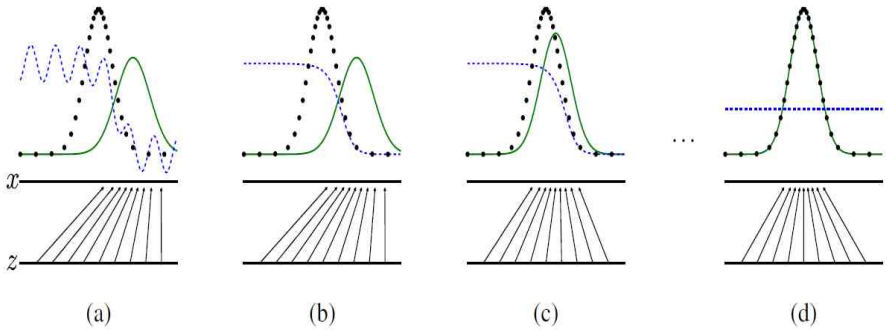

그림 10.2 학습에 따른 data generating distribution(검은색), discriminator distribution(파란색), generative distribution(녹색)의 관계

즉, 처음 시작할 때는 (a)와 같이 p_g가 p_{data}와 전혀 다르게 생긴 것을 볼 수 있고 이 상태에서 discriminator가 두 distribution을 구별하기 위해 학습을 하면 (b)와 같이 좀 더 smooth하고 잘 구별하는 distribution이 만들어진다. 이후 G가 현재 discriminator가 구별하기 어려운 방향으로 학습을 하면 (c)와 같이 좀 더 p_g가 p_{data}와 가까워지게 된다. 이런 식으로 쭉 학습을 반복하다 보면 결국에는 p_g=p_{data}가 되어 discriminator가 둘을 전혀 구별하지 못하는 즉, $D(x) = 1/2$인 상태가 된다.

논문에서 한 가지 실용적인 tip이 나오는데, 위에 value function에서 $\log(1 - D(G(z)))$부분을 G에 대해 minimize하는 대신 $\log(D(G(z)))$를 maximize하도록 G를 학습시킨다는 것이다. 나중에 저자가 밝히듯이 이 부분은 전혀 이론적인 동기로부터 수정을 한 것이 아니라 순수하게 실용적인 측면에서 적용을 하게 된 것이다. 이유도 아주 직관적인데 예를 들어 학습 초기를 생각해보면, G가 초기에는 아주 이상한 image들을 생성하기 때문에 D가 너무도 쉽게 이를 real image와 구별하게 되고 따라서 $\log(1 - D(G(z)))$ 값이 매우 saturate하여 gradient를 계산해보면 아주 작은 값이 나오기 때문에 학습이 엄청 느리게 된다. 하지만 문제를 $G = \arg max_G \log(D(G(z)))$로 바꾸게 되면, 초기에 D가 G로 나온 image를 잘 구별한다고 해도 위와 같은 문제가 생기지 않기 때문에 원래 문제와 같은 fixed point를 얻게 되면서도 stronger gradient를 줄 수 있는 상당히 괜찮은 해결방법이 된다.

다. Intuition in GAN

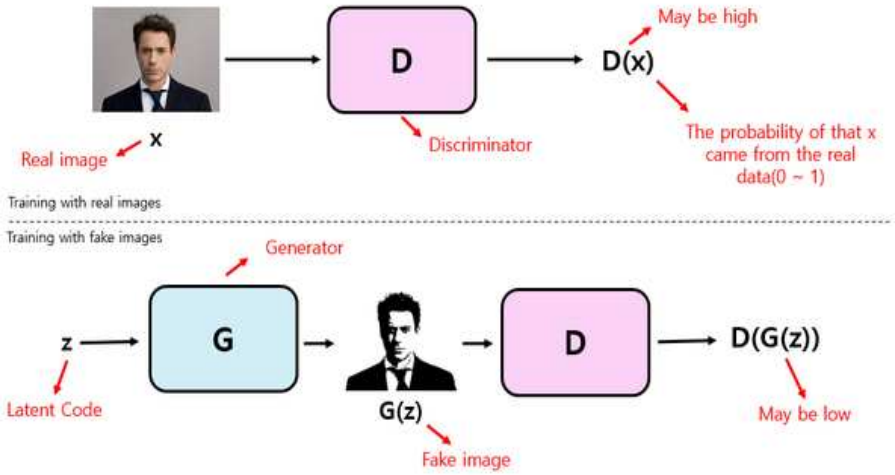

그림 10.3 GAN의 Discriminator와 Generator의 역할

직관적으로 먼저 살펴보면, 그림 10-3에서 위쪽은 진짜 이미지를 가지고 학습을 한 것이고 아래는 가짜 이미지를 가지고 학습을 한 것이다. 진짜 이미지가 들어갔을 때 discriminator는 그림 10-4와 같이 진짜 이미지를 진짜로 구별하도록 학습을 합니다.

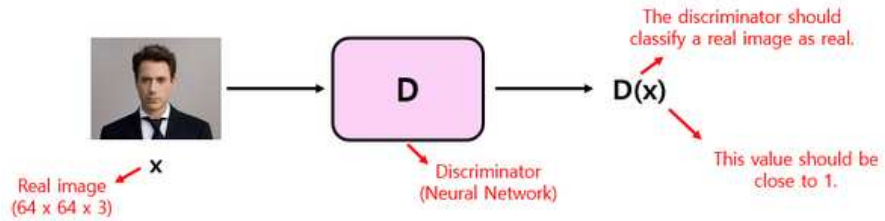

그림 10.4 Discriminator는 진짜 이미지를 진짜로 구별하도록 학습

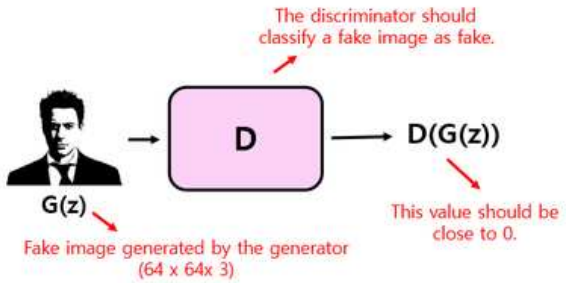

그림 10.5 Discriminator가 가짜 이미지를 가짜로 구별하도록 학습

input과 ouput 차원만 보게 된다면 input은 이미지의 고정된 벡터가 들어가게 되고 (예를 들어, 64x64x3의 컬러 영상), output 같은 경우에는 진짜, 가짜 두 가지 binary classification만 하면 되기 때문에 output의 차원은 1차원이 되고 sigmoid activation 함수를 거치게 되어 0.5를 기준으로 classification을 하게 된다. 이러한 방식으로 discriminator는 학습을 하게 된다.

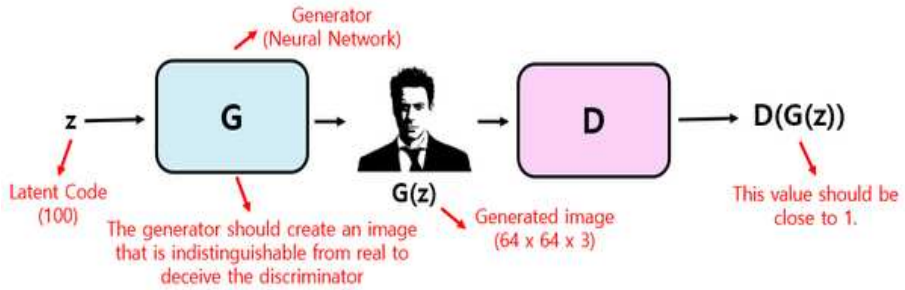

그림 10.6 Generator 학습

반대로 Generator의 경우에는 어떤 랜덤한 코드를 받아서 이미지를 생성해야 한다. 이미지를 생성을 했을 때 이미지를 가지고 discriminator를 속여야 한다. 즉, discriminator의 output이 1이 나오도록 학습을 시켜야 한다는 것이다. output이 1이 나오기 위해서는 진짜 이미지로 학습시켜야 하는데 generator의 경우에는 학습을 하면 할수록 진짜에 가까운 가짜 이미지를 생성하게 됩니다.

1) Discriminator 관점

식 (10-1)을 GAN의 목적 함수라고 부른다. 이 목적 함수를 discriminator의 경우 최대화하는 것이 목표가 된다. 수식에서 $x \sim P_{data}$라고 되어 있는 부분은 앞에서 말한 확률 밀도 함수이다. 즉, 실제 데이터의 분포를 샘플링 한다는 뜻이다. 이 말은 우리가 학습 데이터에 10만개의 사람 이미지가 있다고 한다면 거기서 x 값을 하나씩 뽑는다고 보면 된다.

Discriminator는 진짜 데이터를 받았을 경우 1에 가까운 값을 내놔야 하니까 그것을 수식적으로 $\log D(x)$ 값을 최대화 하게끔 표현한다. $D(x)$는 0에서 1사이의 값을 내보내야 하고, 가장 최대값은 $D(x)$가 1일 때는 0, 0일 때는 마이너스 무한대로 최소가 된다.

목적 함수 수식 오른쪽 $z \sim P_z(z)$의 경우에 z는 랜덤한 값이다. 아래 그림에 보면 z가 generator의 input으로 들어가게 된다, 가우시안 분포표, 즉 정규 분포표에서 백차원짜리 벡터를 샘플링하게 된다. random한 벡터를 G에게 줬을 때 가짜 이미지를 생성하고, $G(z)$값을 D가 받았을 때는 0에 가까운 값을 줘야한다. 수식적으로 log(1-x) 꼴로 나타나게 됩니다.

2) Generator 관점

목적 함수 수식에서, 왼쪽의 $x \sim P_{data}$의 부분에는 generator가 관여할 수 없는 부분이다. 아래 그림을 보면 discriminator가 학습하는데는 generator함수가 쓰이지 않았기 때문에 관

여할 수 없다. 따라서 오른쪽 $z \sim P_z(z)$ 부분을 최소화해야 한다. discriminator와 반대로 $\log(D(G(z)))$가 1이 되서 최소화를 만들어야 합니다.

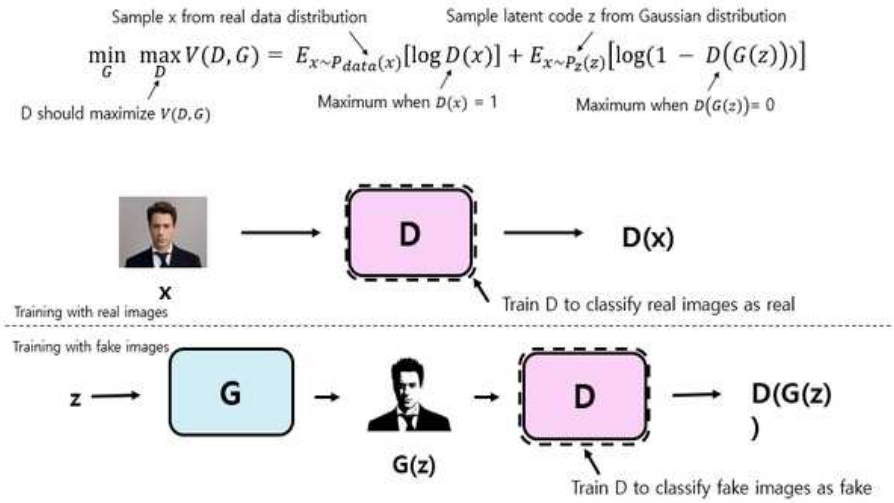

그림 10.7 Discriminator 관점

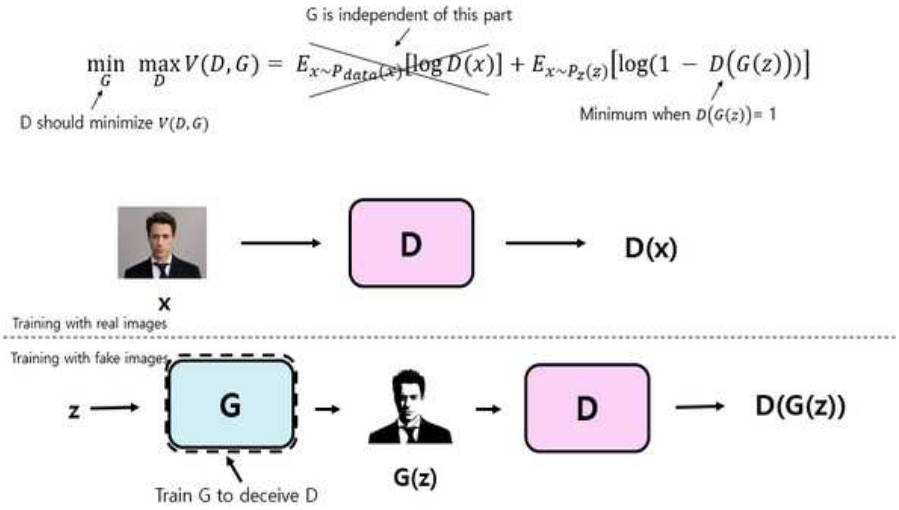

그림 10.8 Generator 관점

라. GAN 구현

1) 데이터 load

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)
```

2) 하이퍼파라미터 정의

```
total_epoch = 100
batch_size = 100
learning_rate = 0.0002
# 신경망 레이어 구성 옵션
n_hidden = 256
n_input = 28 * 28
n_noise = 128  # 생성기의 입력 값으로 사용할 노이즈의 크기
```

3) 신경망 모델 구성

```
#GAN 도 Unsupervised 학습이므로 Autoencoder 처럼 Y 를 사용하지 않음
X = tf.placeholder(tf.float32, [None, n_input])
Z = tf.placeholder(tf.float32, [None, n_noise])  # 노이즈 Z를 입력값으로 사용

# 생성기 신경망에 사용하는 변수들
G_W1 = tf.Variable(tf.random_normal([n_noise, n_hidden], stddev=0.01))
G_b1 = tf.Variable(tf.zeros([n_hidden]))
G_W2 = tf.Variable(tf.random_normal([n_hidden, n_input], stddev=0.01))
G_b2 = tf.Variable(tf.zeros([n_input]))

# 판별기 신경망에 사용하는 변수들
D_W1 = tf.Variable(tf.random_normal([n_input, n_hidden], stddev=0.01))
D_b1 = tf.Variable(tf.zeros([n_hidden]))
# 판별기의 최종 결과값은 얼마나 진짜와 가깝냐를 판단하는 한 개의 스칼라값
D_W2 = tf.Variable(tf.random_normal([n_hidden, 1], stddev=0.01))
D_b2 = tf.Variable(tf.zeros([1]))

#생성기(G) 신경망을 구성
def generator(noise_z):
    hidden = tf.nn.relu(tf.matmul(noise_z, G_W1) + G_b1)
    output = tf.nn.sigmoid(tf.matmul(hidden, G_W2) + G_b2)
    return output
```

```
# 판별기(D) 신경망을 구성
def discriminator(inputs):
    hidden = tf.nn.relu(tf.matmul(inputs, D_W1) + D_b1)
    output = tf.nn.sigmoid(tf.matmul(hidden, D_W2) + D_b2)
    return output

# 랜덤한 노이즈(Z)를 만듬
def get_noise(batch_size, n_noise):
    return np.random.normal(size=(batch_size, n_noise))

# 노이즈를 이용해 랜덤한 이미지를 생성
G = generator(Z)
# 노이즈를 이용해 생성한 이미지가 진짜 이미지인지 판별한 값을 구합니다.
D_gene = discriminator(G)
# 진짜 이미지를 이용해 판별한 값을 구합니다.
D_real = discriminator(X)
```

4) 손실함수 정의 및 최적화

GAN 모델의 최적화는 loss_G 와 loss_D 를 최대화 하는 것임. 다만 loss_D와 loss_G는 서로 연관관계가 있기 때문에 두 개의 손실 값이 항상 같이 증가하는 경향을 보이지는 않을 것이다. loss_D가 증가하려면 loss_G는 하락해야하고, loss_G가 증가하려면 loss_D는 하락해야하는 경쟁관계에 있기 때문이다. 논문의 수식에 따른 다음 로직을 보면 loss_D 를 최대화하기 위해서는 D_gene 값을 최소화하게 된다. 이것은 판별기는 생성기가 만들어낸 이미지가 가짜라고 판단하도록 판별기 신경망을 학습시킨다.

- 판별기에 진짜 이미지를 넣었을 때에도 최대값을: tf.log(D_real)
- 가짜 이미지를 넣었을 때에도 최대값을: tf.log(1 - D_gene) 갖도록 학습시키기 때문이다.

```
loss_D = tf.reduce_mean(tf.log(D_real) + tf.log(1 - D_gene))
#반면 loss_G 를 최대화하기 위해서는 D_gene 값을 최대화하게 되는데,
#이것은 가짜 이미지를 넣었을 때, 판별기가 최대한 실제 이미지라고 판단하도록
#생성기 신경망을 학습시킨다.
#논문에서는 loss_D 와 같은 수식으로 최소화 하는 생성기를 찾지만, 결국 D_gene
#값을 최대화하는 것이므로 다음과 같이 사용할 수 있다.
loss_G = tf.reduce_mean(tf.log(D_gene))

# loss_D 를 구할 때는 판별기 신경망에 사용되는 변수만 사용하고,
# loss_G 를 구할 때는 생성기 신경망에 사용되는 변수만 사용하여 최적화
D_var_list = [D_W1, D_b1, D_W2, D_b2]
G_var_list = [G_W1, G_b1, G_W2, G_b2]
```

```
#GAN 논문의 수식에 따르면 loss 를 극대화 해야하지만, minimize 하는 최적화
#함수를 사용하기 때문에 최적화 하려는 loss_D 와 loss_G 에 음수 부호를 붙인다.
train_D=
tf.train.AdamOptimizer(learning_rate).minimize(-loss_D, var_list=D_var_list)
train_G=
tf.train.AdamOptimizer(learning_rate).minimize(-loss_G, var_list=G_var_list)
```

5) 신경망 모델 학습

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())

total_batch = int(mnist.train.num_examples/batch_size)
loss_val_D, loss_val_G = 0, 0

for epoch in range(total_epoch):
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        noise = get_noise(batch_size, n_noise)
        # 판별기와 생성기 신경망을 각각 학습
        _, loss_val_D = sess.run([train_D, loss_D],
                                 feed_dict={X: batch_xs, Z: noise})
        _, loss_val_G = sess.run([train_G, loss_G],
                                 feed_dict={Z: noise})

        print('Epoch:', '%04d' % epoch, 'D loss: {:.4}'.format(loss_val_D),
              'G loss: {:.4}'.format(loss_val_G))

        #학습이 되어가는 모습을 보기 위해 주기적으로 이미지를 생성하여 저장
        if epoch == 0 or (epoch + 1) % 10 == 0:
          sample_size = 10
          noise = get_noise(sample_size, n_noise)
          samples = sess.run(G, feed_dict={Z: noise})

          fig, ax = plt.subplots(1, sample_size, figsize=(sample_size, 1))
          for i in range(sample_size):
              ax[i].set_axis_off()
              ax[i].imshow(np.reshape(samples[i], (28, 28)))
          plt.savefig('samples/{}.png'.format(str(epoch).zfill(3)),
                      bbox_inches='tight')
          plt.close(fig)
print('최적화 완료!')
```

< Example 10-1 > GAN을 이용한 MNIST 숫자 합성

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)

#옵션 설정
total_epoch = 100
batch_size = 100
learning_rate = 0.0002
# 신경망 레이어 구성 옵션
n_hidden = 256
n_input = 28 * 28
n_noise = 128  # 생성기의 입력값으로 사용할 노이즈의 크기

#신경망 모델 구성
# GAN 도 Unsupervised 학습이므로 Autoencoder 처럼 Y 를 사용하지 않습니다.
X = tf.placeholder(tf.float32, [None, n_input])
# 노이즈 Z를 입력값으로 사용합니다.
Z = tf.placeholder(tf.float32, [None, n_noise])

# 생성기 신경망에 사용하는 변수들입니다.
G_W1 = tf.Variable(tf.random_normal([n_noise, n_hidden], stddev=0.01))
G_b1 = tf.Variable(tf.zeros([n_hidden]))
G_W2 = tf.Variable(tf.random_normal([n_hidden, n_input], stddev=0.01))
G_b2 = tf.Variable(tf.zeros([n_input]))

# 판별기 신경망에 사용하는 변수들입니다.
D_W1 = tf.Variable(tf.random_normal([n_input, n_hidden], stddev=0.01))
D_b1 = tf.Variable(tf.zeros([n_hidden]))
# 판별기의 최종 결과값은 얼마나 진짜와 가깝냐를 판단하는 한 개의 스칼라값입니다.
D_W2 = tf.Variable(tf.random_normal([n_hidden, 1], stddev=0.01))
D_b2 = tf.Variable(tf.zeros([1]))

# 생성기(G) 신경망을 구성합니다.
def generator(noise_z):
    hidden = tf.nn.relu(tf.matmul(noise_z, G_W1) + G_b1)
    output = tf.nn.sigmoid(tf.matmul(hidden, G_W2) + G_b2)
    return output

# 판별기(D) 신경망을 구성합니다.
def discriminator(inputs):
    hidden = tf.nn.relu(tf.matmul(inputs, D_W1) + D_b1)
    output = tf.nn.sigmoid(tf.matmul(hidden, D_W2) + D_b2)
    return output

# 랜덤한 노이즈(Z)를 만듭니다.
def get_noise(batch_size, n_noise):
    return np.random.normal(size=(batch_size, n_noise))
```

< Example 10-1 > GAN을 이용한 MNIST 숫자 합성 (계속)

```
# 노이즈를 이용해 랜덤한 이미지를 생성합니다.
G = generator(Z)
# 노이즈를 이용해 생성한 이미지가 진짜 이미지인지 판별한 값을 구합니다.
D_gene = discriminator(G)
# 진짜 이미지를 이용해 판별한 값을 구합니다.
D_real = discriminator(X)

loss_D = tf.reduce_mean(tf.log(D_real) + tf.log(1 - D_gene))
loss_G = tf.reduce_mean(tf.log(D_gene))

D_var_list = [D_W1, D_b1, D_W2, D_b2]
G_var_list = [G_W1, G_b1, G_W2, G_b2]

# 최적화 하려는 loss_D 와 loss_G 에 음수 부호를 붙여줍니다.
train_D= tf.train.AdamOptimizer(learning_rate).minimize(-loss_D, var_list=D_var_list)
train_G= tf.train.AdamOptimizer(learning_rate).minimize(-loss_G, var_list=G_var_list)

#신경망 모델 학습
sess = tf.Session()
sess.run(tf.global_variables_initializer())

total_batch = int(mnist.train.num_examples/batch_size)
loss_val_D, loss_val_G = 0, 0

for epoch in range(total_epoch):
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        noise = get_noise(batch_size, n_noise)

        # 판별기와 생성기 신경망을 각각 학습시킵니다.
        _, loss_val_D = sess.run([train_D, loss_D],
                                 feed_dict={X: batch_xs, Z: noise})
        _, loss_val_G = sess.run([train_G, loss_G],
                                 feed_dict={Z: noise})

    print('Epoch:', '%04d' % epoch,
          'D loss: {:.4}'.format(loss_val_D),
          'G loss: {:.4}'.format(loss_val_G))

    #학습이 되어가는 모습을 보기 위해 주기적으로 이미지를 생성하여 저장
    if epoch == 0 or (epoch + 1) % 10 == 0:
        sample_size = 10
        noise = get_noise(sample_size, n_noise)
        samples = sess.run(G, feed_dict={Z: noise})

        fig, ax = plt.subplots(1, sample_size, figsize=(sample_size, 1))
        for i in range(sample_size):
            ax[i].set_axis_off()
            ax[i].imshow(np.reshape(samples[i], (28, 28)))
        plt.savefig('samples/{}.png'.format(str(epoch).zfill(3)), bbox_inches='tight')
        plt.close(fig)
print('최적화 완료!')
```

< Example 10-2 > GAN을 이용한 원하는 MNIST 숫자 합성

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)

#옵션 설정
total_epoch = 100
batch_size = 100
n_hidden = 256
n_input = 28 * 28
n_noise = 128
n_class = 10

#신경망 모델 구성
X = tf.placeholder(tf.float32, [None, n_input])
# 노이즈와 실제 이미지에, 그에 해당하는 숫자에 대한 정보를 넣어주기 위해 사용
Y = tf.placeholder(tf.float32, [None, n_class])
Z = tf.placeholder(tf.float32, [None, n_noise])

def generator(noise, labels):
    with tf.variable_scope('generator'):
        # noise 값에 labels 정보를 추가합니다.
        inputs = tf.concat([noise, labels], 1)
        # TensorFlow 에서 제공하는 유틸리티 함수를 이용해 신경망 구성
        hidden = tf.layers.dense(inputs, n_hidden, activation=tf.nn.relu)
        output = tf.layers.dense(hidden, n_input,  activation=tf.nn.sigmoid)
    return output


def discriminator(inputs, labels, reuse=None):
    with tf.variable_scope('discriminator') as scope:
        # 노이즈에서 생성한 이미지와 실제 이미지를 판별하는 모델의 변수를 동일하게 하
        #  기 위해,이전에 사용되었던 변수를 재사용
        if reuse:
            scope.reuse_variables()
        inputs = tf.concat([inputs, labels], 1)
        hidden = tf.layers.dense(inputs, n_hidden, activation=tf.nn.relu)
        output = tf.layers.dense(hidden, 1, activation=None)
    return output

def get_noise(batch_size, n_noise):
    return np.random.uniform(-1., 1., size=[batch_size, n_noise])

# 생성 모델과 판별 모델에 Y 즉, labels 정보를 추가하여
# labels 정보에 해당하는 이미지를 생성할 수 있도록 유도합니다.
G = generator(Z, Y)
D_real = discriminator(X, Y)
D_gene = discriminator(G, Y, True)
```

< Example 10-2 > GAN을 이용한 원하는 MNIST 숫자 합성(계속)

```
# 진짜 이미지를 판별하는 D_real 값은 1에 가깝도록,
# 가짜 이미지를 판별하는 D_gene 값은 0에 가깝도록 하는 손실 함수
loss_D_real = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
                            logits=D_real, labels=tf.ones_like(D_real)))
loss_D_gene = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
                            logits=D_gene, labels=tf.zeros_like(D_gene)))
# loss_D_real 과 loss_D_gene 을 더한 뒤 이 값을 최소화 하도록 최적화합니다.
loss_D = loss_D_real + loss_D_gene
# 가짜 이미지를 진짜에 가깝게 만들도록 생성망을 학습시키기 위해, D_gene 을 최대한 1에
# 가깝도록 만드는 손실함수
loss_G = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
                            logits=D_gene, labels=tf.ones_like(D_gene)))

# TensorFlow 에서 제공하는 유틸리티 함수를 이용해
# discriminator 와 generator scope 에서 사용된 변수들을 쉽게 가져올 수 있다.
vars_D = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,
                                scope='discriminator')
vars_G = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,
                                scope='generator')
train_D = tf.train.AdamOptimizer().minimize(loss_D, var_list=vars_D)
train_G = tf.train.AdamOptimizer().minimize(loss_G, var_list=vars_G)

#신경망 모델 학습
sess = tf.Session()
sess.run(tf.global_variables_initializer())
total_batch = int(mnist.train.num_examples/batch_size)
loss_val_D, loss_val_G = 0, 0

for epoch in range(total_epoch):
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        noise = get_noise(batch_size, n_noise)
        _, loss_val_D = sess.run([train_D, loss_D],
                                 feed_dict={X: batch_xs, Y: batch_ys, Z: noise})
        _, loss_val_G = sess.run([train_G, loss_G],
                                 feed_dict={Y: batch_ys, Z: noise})
    print('Epoch:', '%04d' % epoch,'D loss: {:.4}'.format(loss_val_D),
          'G loss: {:.4}'.format(loss_val_G))

    #학습이 되어가는 모습을 보기 위해 주기적으로 레이블에 따른 이미지를 생성하여 저장
    if epoch == 0 or (epoch + 1) % 10 == 0:
        sample_size = 10
        noise = get_noise(sample_size, n_noise)
        samples = sess.run(G, feed_dict={Y: mnist.test.labels[:sample_size], Z: noise})

        fig, ax = plt.subplots(2, sample_size, figsize=(sample_size, 2))
        for i in range(sample_size):
            ax[0][i].set_axis_off()
            ax[1][i].set_axis_off()
            ax[0][i].imshow(np.reshape(mnist.test.images[i], (28, 28)))
            ax[1][i].imshow(np.reshape(samples[i], (28, 28)))
        plt.savefig('samples2/{}.png'.format(str(epoch).zfill(3)), bbox_inches='tight')
        plt.close(fig)
print('최적화 완료!')
```

< Example 10-3 > DCGAN을 이용한 MNIST 숫자 합성

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data')

tf.reset_default_graph()
batch_size = 64
n_noise = 64
X_in = tf.placeholder(dtype=tf.float32, shape=[None, 28, 28], name='X')
noise = tf.placeholder(dtype=tf.float32, shape=[None, n_noise])
keep_prob = tf.placeholder(dtype=tf.float32, name='keep_prob')
is_training = tf.placeholder(dtype=tf.bool, name='is_training')

def lrelu(x):
    return tf.maximum(x, tf.multiply(x, 0.2))
def binary_cross_entropy(x, z):
    eps = 1e-12
    return (-(x * tf.log(z + eps) + (1. - x) * tf.log(1. - z + eps)))
def discriminator(img_in, reuse=None, keep_prob=keep_prob):
     activation = lrelu
     with tf.variable_scope("discriminator", reuse=reuse):
         x = tf.reshape(img_in, shape=[-1, 28, 28, 1])
         x = tf.layers.conv2d(x, kernel_size=5, filters=64, strides=2, padding='same',
                activation=activation)
         x = tf.layers.dropout(x, keep_prob)
         x = tf.layers.conv2d(x, kernel_size=5, filters=64, strides=1, padding='same',
                activation=activation)
         x = tf.layers.dropout(x, keep_prob)
         x = tf.layers.conv2d(x, kernel_size=5, filters=64, strides=1, padding='same',
                activation=activation)
         x = tf.layers.dropout(x, keep_prob)
         x = tf.contrib.layers.flatten(x)
         x = tf.layers.dense(x, units=128, activation=activation)
         x = tf.layers.dense(x, units=1, activation=tf.nn.sigmoid)
         return x
def generator(z, keep_prob=keep_prob, is_training=is_training):
     activation = lrelu
     momentum = 0.99
     with tf.variable_scope("generator", reuse=None):
         x = z
         d1 = 4
         d2 = 1
         x = tf.layers.dense(x, units=d1 * d1 * d2, activation=activation)
         x = tf.layers.dropout(x, keep_prob)
         x=tf.contrib.layers.batch_norm(x, is_training=is_training, decay=momentum)
         x = tf.reshape(x, shape=[-1, d1, d1, d2])
         x = tf.image.resize_images(x, size=[7, 7])
         x = tf.layers.conv2d_transpose(x, kernel_size=5, filters=64, strides=2,
                padding='same', activation=activation)
         x = tf.layers.dropout(x, keep_prob)
         x = tf.contrib.layers.batch_norm(x, is_training=is_training,
                decay=momentum)
         x = tf.layers.conv2d_transpose(x, kernel_size=5, filters=64, strides=2,
                padding='same', activation=activation)
         x = tf.layers.dropout(x, keep_prob)
         x = tf.contrib.layers.batch_norm(x, is_training=is_training, decay=momentum)
         x = tf.layers.conv2d_transpose(x, kernel_size=5, filters=64, strides=1, padding='same',
                activation=activation)
         x = tf.layers.dropout(x, keep_prob)
         x = tf.contrib.layers.batch_norm(x, is_training=is_training, decay=momentum)
         x = tf.layers.conv2d_transpose(x, kernel_size=5, filters=1, strides=1, padding='same',
                activation=tf.nn.sigmoid)
         return x
```

< Example 10-3 > DCGAN을 이용한 MNIST 숫자 합성(계속)

```
g = generator(noise, keep_prob, is_training)
d_real = discriminator(X_in)
d_fake = discriminator(g, reuse=True)

vars_g = [var for var in tf.trainable_variables() if var.name.startswith("generator")]
vars_d = [var for var in tf.trainable_variables() if var.name.startswith("discriminator")]

d_reg = tf.contrib.layers.apply_regularization(tf.contrib.layers.l2_regularizer(1e-6), vars_d)
g_reg = tf.contrib.layers.apply_regularization(tf.contrib.layers.l2_regularizer(1e-6), vars_g)

loss_d_real = binary_cross_entropy(tf.ones_like(d_real), d_real)
loss_d_fake = binary_cross_entropy(tf.zeros_like(d_fake), d_fake)
loss_g = tf.reduce_mean(binary_cross_entropy(tf.ones_like(d_fake), d_fake))
loss_d = tf.reduce_mean(0.5 * (loss_d_real + loss_d_fake))

update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
with tf.control_dependencies(update_ops):
    optimizer_d = tf.train.RMSPropOptimizer(learning_rate=0.00015).minimize(loss_d + d_reg,
var_list=vars_d)
    optimizer_g = tf.train.RMSPropOptimizer(learning_rate=0.00015).minimize(loss_g + g_reg,
var_list=vars_g)

sess = tf.Session()
sess.run(tf.global_variables_initializer())
for i in range(60000):
    train_d = True
    train_g = True
    keep_prob_train = 0.6  # 0.5

    n = np.random.uniform(0.0, 1.0, [batch_size, n_noise]).astype(np.float32)
    batch = [np.reshape(b, [28, 28]) for b in mnist.train.next_batch(batch_size=batch_size)[0]]

    d_real_ls, d_fake_ls, g_ls, d_ls = sess.run([loss_d_real, loss_d_fake, loss_g, loss_d],
            feed_dict={X_in: batch, noise: n, keep_prob: keep_prob_train, is_training: True})

    d_real_ls = np.mean(d_real_ls)
    d_fake_ls = np.mean(d_fake_ls)
    g_ls = g_ls
    d_ls = d_ls

    if g_ls * 1.5 < d_ls:
        train_g = False
        pass
    if d_ls * 2 < g_ls:
        train_d = False
        pass

    if train_d:
        sess.run(optimizer_d, feed_dict={noise: n, X_in: batch, keep_prob: keep_prob_train,
            is_training: True})
    if train_g:
        sess.run(optimizer_g, feed_dict={noise: n, keep_prob: keep_prob_train, is_training:
            True})
```