6. Generative Adversarial Nets (GAN)

6-1. GAN 개요

6-2. GAN Loss Function

6-3. GAN 학습 결과

6-4. GAN 학습 알고리즘

6-5. GAN 학습 예시 (1) - 정규분포 데이터 생성

6-6. GAN 학습 예시 (2) : 여러 개의 정규분포 데이터 생성

6-7. GAN 학습의 이론적 근거

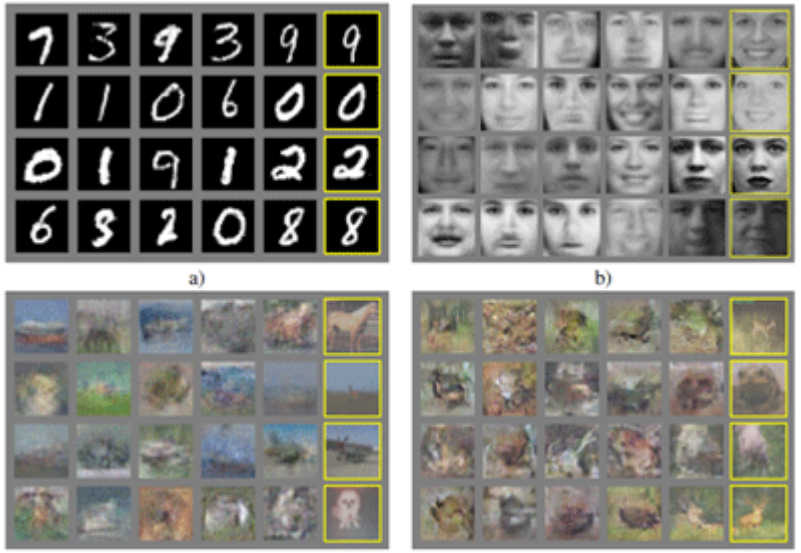
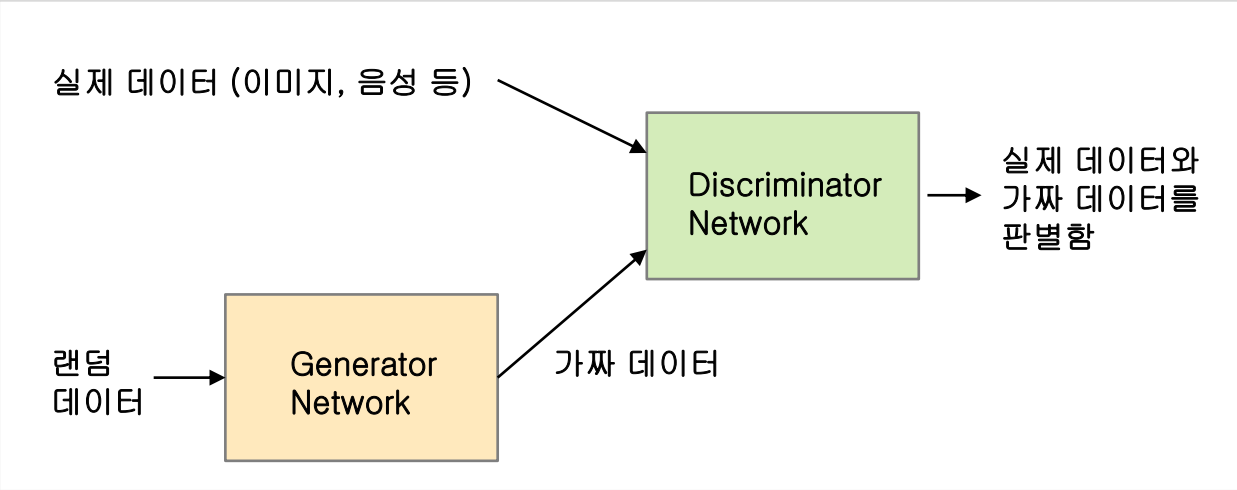
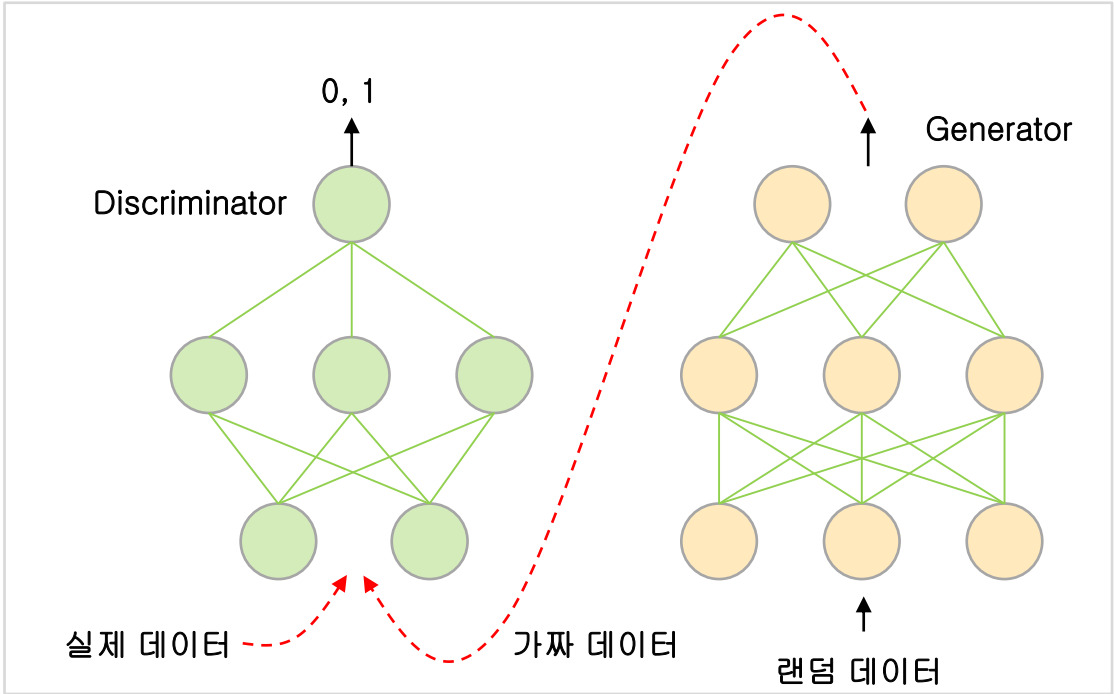


그림 출처 : Ian J. Goodfellow et, al., 2014, Generative Adversarial Nets

GAN 개요

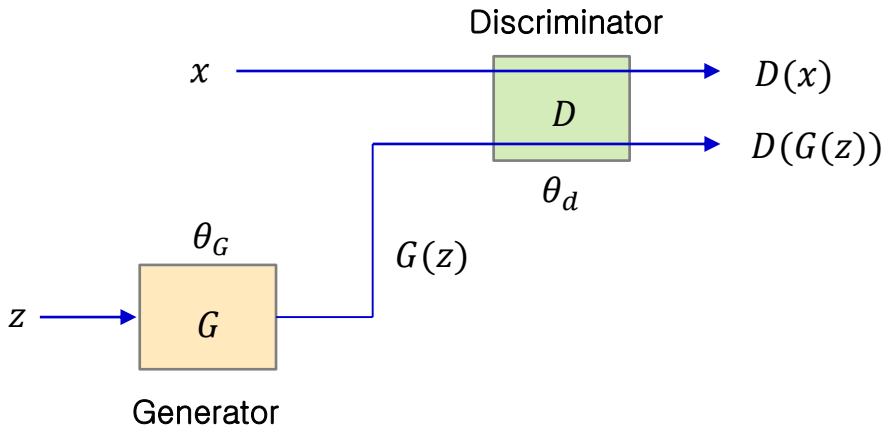
- GAN은 2014년 Ian Goodfellow가 발표한 것으로 Unsupervised Learning 방식으로 이미지, 문서, 음성 등의 데이터를 생성하는 알고리즘이다.
- 논문 : Ian J. Goodfellow et, al., 2014, Generative Adversarial Nets. GAN은 최근 발표된 딥러닝 알고리즘 중 가장 흥미로운 아이디어로 평가 받고 있다.
- GAN은 Discriminator N/W와 Generator N/W로 구성되어 있다. Generator는 랜덤 데이터 (Noise)를 입력 받아 실제 데이터 (Real Data)와 유사한 가짜 데이터 (Fake Data)를 생성하고, Discriminator는 Real Data와 Fake Data를 데이터를 구별한다.
- Generator는 실제 데이터와 동일한 차원의 데이터를 출력하고, Discriminator는 0 ~ 1 사잇값 (Sigmoid)을 출력한다. Discriminator에 Real Data가 들어오면 1에 가까운 값이 출력되고, Fake Data가 들어오면 0에 가까운 값이 출력된다.
- Discriminator는 Real과 Fake 데이터를 구별하도록 학습하고, Generator는 Discriminator가 구별하지 못하도록 실제 같은 가짜 데이터를 생성하도록 학습한다.
- GAN 알고리즘은 Generator와 Discriminator가 경쟁을 통해 각자 최선의 목적을 달성할 수 있도록 학습하고, 서로 균형 (Nash Equilibrium)을 이룬 상태에서 학습이 완료된다. Nash 균형 상태에 도달하면 Discriminator는 실제와 가짜를 잘 구별하지 못하므로 0.5에 가까운 값을 출력한다.
- GAN은 실제와 유사한 학습 데이터를 생성할 수 있으므로, 금융 데이터 시뮬레이션 등에 활용할 수 있다. (10장 주가 시계열 시뮬레이션 참조)





GAN 학습을 위한 Loss function

- Discriminator와 Generator를 학습시키기 위해서는 아래와 같은 Loss function $(V(D,G))$을 이용한다.
- $V(D,G)$가 최대가 되도록 Discriminator의 weight와 bias (θ_d)를 업데이트하고, $V(D,G)$가 최소가 되도록 Generator의 weight와 bias (θ_G)를 업데이트한다.
- Discriminator는 실제 데이터인 x가 들어오면 그 출력인 $D(x)$가 1에 가까운 값이 나오도록 θ_d를 업데이트하고, 가짜 데이터인 $G(z)$가 들어오면 그 출력인 $D(G(z))$가 0에 가까운 값이 나오도록 θ_d를 업데이트한다.
- Generator는 Discriminator의 $D(G(z))$가 1에 가까운 값이 나오도록 θ_G를 업데이트한다.
- 학습이 완료되면 Generator는 랜덤 노이즈 데이터를 받아서 실제 데이터와 유사한 $G(z)$를 생성한다. Discriminator는 x를 받거나, $G(z)$를 받아도 실제와 가짜를 구별하지 못하는 상태가 된다. 즉, $D(x) = 0.5, D(G(z)) = 0.5$ 인 상태가 된다.
- Discriminator가 x와 $G(z)$를 구별하지 못한다는 것은 $x = G(z)$ 라는 것을 의미한다. 즉, 실제 데이터 (x)와 유사한 가짜 데이터 $(G(z))$가 생성된 것이다.
- GAN은 다른 신경망과 달리 최적화 (min or max)를 위해 학습하는 것이 아니라, D와 G가 $V(D,G)$를 가지고 min-max 게임을 하고 상호 균형에 도달하도록 학습시키는 알고리즘이다. (결과적으로는 G가 게임의 승자임).



GAN의 Loss function

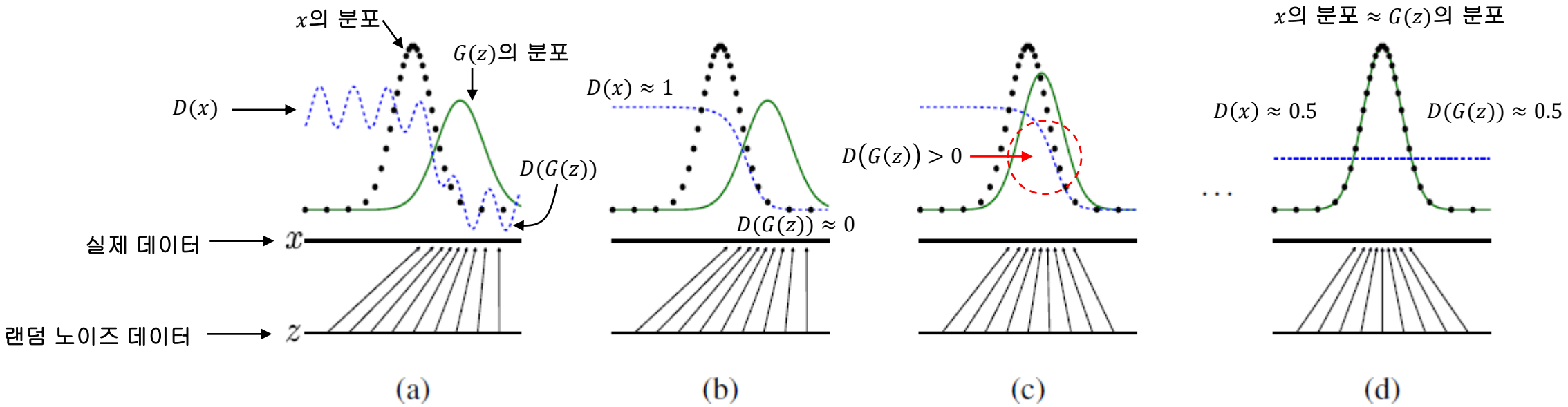
$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1-D(G(z)))]$$

- Discriminator는 $\max_D V(D,G)$로 학습한다. $D(x) = 1$ 이고 $D(G(z)) = 0$ 일 때가 최대이다.
- Real Data (x)가 들어오면 1을 출력하고, Fake Data $(G(z))$가 들어오면 0을 출력하도록 학습한다.
- Generator는 $\min_G V(G)$로 학습한다. $D(G(z)) = 1$ 일 때가 최소이다.
- Discriminator에 Fake Data $(G(z))$가 들어오면 1을 출력하도록 학습한다.

GAN 학습의 결과

- 학습이 안된 상태 (a, b)에서 실제 데이터 x의 분포가 아래와 같이 정규분포이고, 임의의 랜덤 데이터 z로 만든 가짜 데이터 $G(z)$의 분포가 오른쪽으로 치우친 분포라면, $D(x)$는 1에 가까운 값이 출력되고, $D(G(z))$는 0에 가까운 값이 출력된다. 즉, D는 진짜 (x)와 가짜 $(G(z))$를 잘 구별하고, G는 진짜 같은 가짜를 잘 만들지 못하고 있다.
- 학습이 진행되면 (c) 가짜 데이터 $G(z)$의 분포가 점점 x 분포와 유사해 지고 $D(G(z))$ 값도 점차 커지고, $D(x)$ 값은 점차 작아진다.
- 학습이 완료되면 (d) x와 $G(z)$의 분포가 잘 일치하고 $D(x) = D(G(z)) = 0.5$로 수렴한다. 즉, 임의의 랜덤 데이터를 G에 입력하면 x와 유사한 분포 특성을 갖는 데이터가 출력된다.

학습이 완료된 후 D의 출력 : $D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \rightarrow p_g(x) = p_{data}(x)$ 일 때 $D^*(x) = 0.5$ 가 된다.



GAN 학습 알고리즘

- Goodfellow 논문에 소개된 pseudo code는 아래와 같다. Discriminator를 k번 학습시키고, Generator를 1 번 학습 시킨다 (학습 데이터의 유형에 따라 k 조절).
- Gradient Descent는 SGD, (mini) Batch update, Momentum, RMSProp, Adam 등의 알고리즘을 모두 사용할 수 있다.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Sample minibatch of m examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$$

$\longleftarrow \max_{D} E_{x \sim p_{data}(x)}[logD(x)] + E_{z \sim p_z(z)}[log(1 - D(G(z)))]$

end for

- Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$$

$\longleftarrow \min_{G} E_{z \sim p_z(z)}[log(1 - D(G(z)))]$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

아래 함수를 이용하면 초기 학습 효과가 개선된다.

$\max_{G} E_{z \sim p_z(z)}[logD(G(z))]$

GAN 학습 예시 (1) : 정규분포 데이터 생성

- 정규분포에서 x를 샘플링한 후 x와 유사한 분포 특성을 갖는 데이터 $G(z)$를 생성해 보고, x와 $G(z)$의 유사성을 확인해 본다 (육안 및 KL divergence).
- x는 표준정규분포에서 1,000개 샘플링하고, z는 $-1 \sim +1$ 사이의 임의 랜덤값 1,000개를 샘플링한다. x는 1차원이고, z는 임의 차원의 데이터이다.
- Discriminator의 입력층 뉴런은 1개, 은닉층 뉴런은 8개, 출력층 뉴런은 1개를 배치한다. 은닉층의 활성함수는 ReLu를 사용하고, 출력층은 이진분류 (real or fake)를 위해 Sigmoid를 사용한다. 은닉층 뉴런 개수는 임의로 설정할 수 있으나, 입력층 뉴런은 입력 데이터의 차원, 출력층 뉴런은 1개로 설정해야 한다.
- Generator의 입력층 뉴런은 8개, 은닉층 뉴런은 4개, 출력층 뉴런은 1개를 배치한다. 입력층과 은닉층은 임의로 배치해도 되지만 출력층 뉴런 개수는 Discriminator의 입력층 뉴런 개수와 같아야 한다. 은닉층의 활성함수는 ReLu를 사용하고, 출력층 활성함수는 Linear를 사용한다.
- Optimizer는 Adam을 사용하고, learning rate = 0.0001, k = 1을 사용하여 mini-batch 방식으로 10,000번 학습한다.

x 데이터의 예시

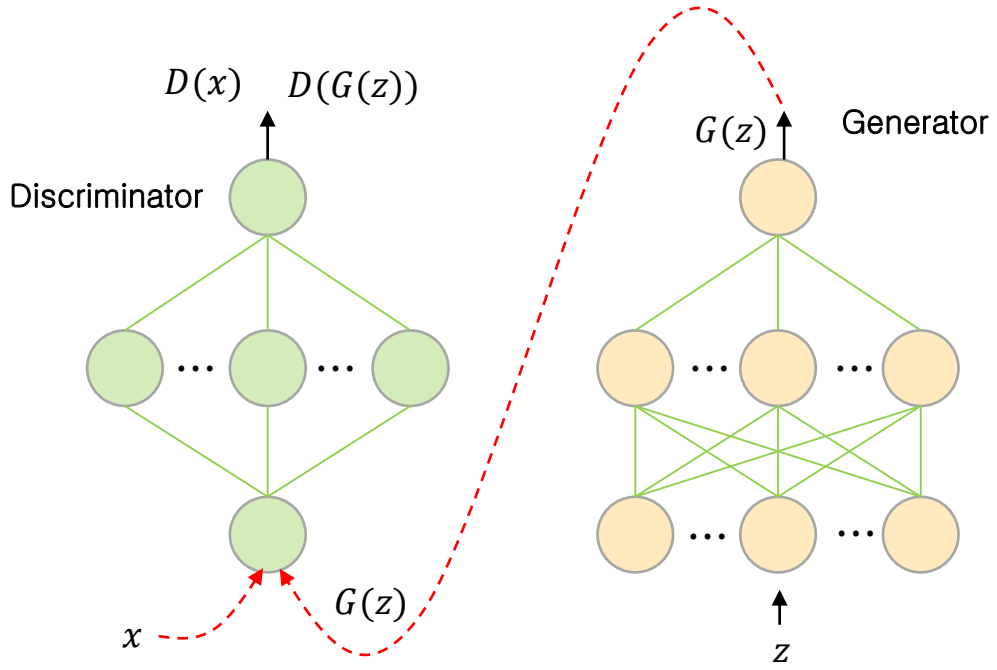
```
[-2.49e-03,  9.26e-01,  1.15e+00,  2.96e-01,  1.37e+00,  1.23e+00,
  1.29e+00, -1.72e+00, -4.34e-01,  3.91e-01,  7.83e-01,  5.30e-01,
 -7.06e-01, -2.51e-01,  1.85e-02,  5.04e-03,  9.25e-01,  9.32e-02 ...]
```

z 데이터의 예시

```
[[-0.58,  0.35,  0.86, ...,  0.58, -0.67, -0.14],
 [ 0.01,  0.05,  0.81, ..., -0.22, -0.59,  0.13],
 [-0.05,  0.89, -0.16, ..., -0.03, -0.02,  0.18]]
```

생성된 G(z) 데이터의 예시

```
[-6.56e-01,  6.41e-02, -7.06e-01, -8.61e-01,  1.62e+00, -3.10e-01,
  4.13e-01, -1.21e+00,  5.91e-01, -2.84e-01, -9.49e-01,  3.34e-01,
 -8.57e-01,  9.67e-01,  1.51e-01, -5.23e-03,  1.15e+00,  3.69e-01 ...]
```



(실습 파일 : 6-1.GAN(Normal).py)

GAN 학습 예시 (1) : 정규분포 데이터 생성

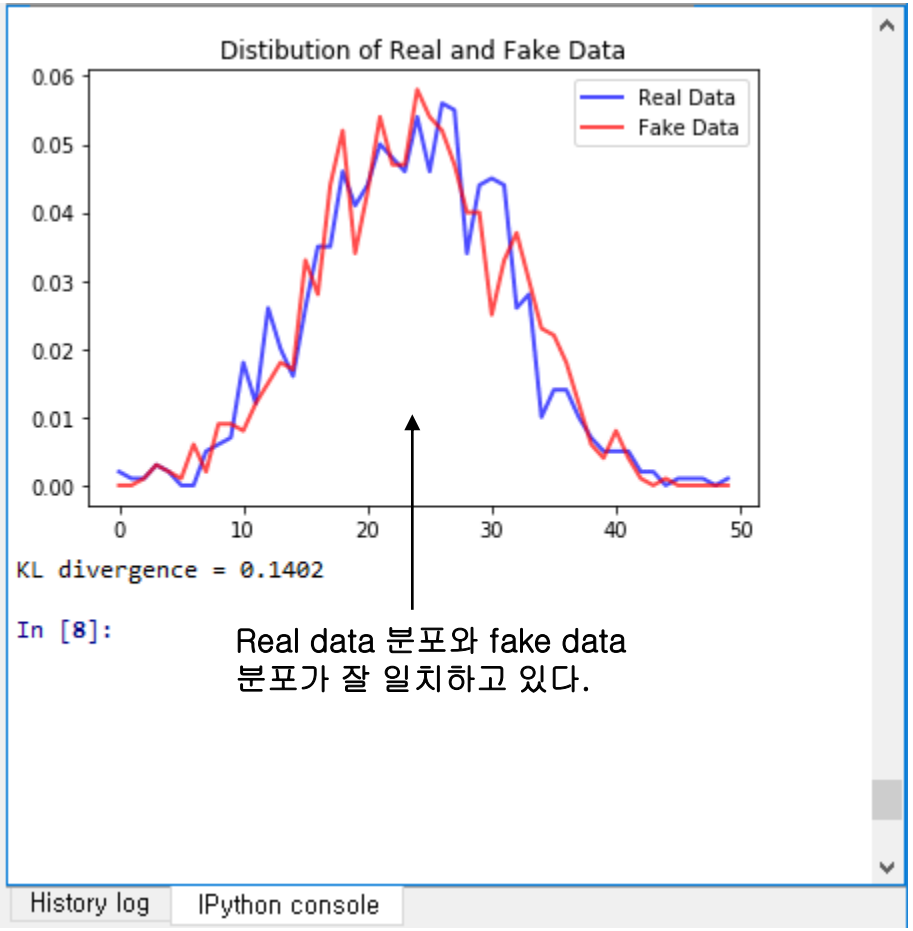
- Goodfellow의 pseudo code를 이용하고 이전 페이지의 조건대로 학습한 결과는 아래와 같다. Real data 분포와 fake data 분포가 잘 일치하고 있다.
- 이 방식을 이용하면 임의의 분포를 갖는 데이터를 무한히 생성할 수 있다. 예를 들어 정규분포와는 다른 실제 시장의 주가 수익률 분포를 모방하여 실제와 유사한 주가 데이터를 무한히 생성할 수 있고, 다양한 금융 모델들에 대한 시뮬레이션 등 많은 부분에 활용할 수 있다.

```python
# Generative Adversarial Network (GAN) 예시
# 기본 GAN (original GAN) 알고리즘을 이용하여 정규분포에서 샘플링한 데이터 (Real Data)와
# 유사한 Fake Data를 생성한다. Real Data와 Fake Data의 분포가 잘 일치하는지 확인한다.
#
# 원 논문 : Ian J. Goodfellow, et, al., 2014, Generative Adversarial Nets.
# 위 논문의 Psedudo code (Section 4. Algorithm 1)를 위주로 하였음.
# 참고 : Original GAN은 unstable, mode collapse 등의 문제가 있으며, 최근 변형된 알고리즘이
#       많이 개발되어 있다. 여기서는 원 알고리즘을 사용하였으므로 결과가 다소 unstable 할 수 있다
#
# 2018.9.10, 아마추어퀀트 (조성현)
# ------------------------------------------------------------------------------------
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# 정규분포로부터 데이터를 샘플링한다
realData = np.random.normal(size=1000)
realData = realData.reshape(realData.shape[0], 1)

# 데이터 P, Q에 대한 KL divergence를 계산한다.
def KL(P, Q):
    # 두 데이터의 분포를 계산한다
    histP, binsP = np.histogram(P, bins=50)
    histQ, binsQ = np.histogram(Q, bins=binsP)

    # 두 분포를 pdf로 만들기 위해 normalization한다.
    histP = histP / (np.sum(histP) + 1e-8)
```



GAN 학습 예시 (1) : 정규분포 데이터 생성

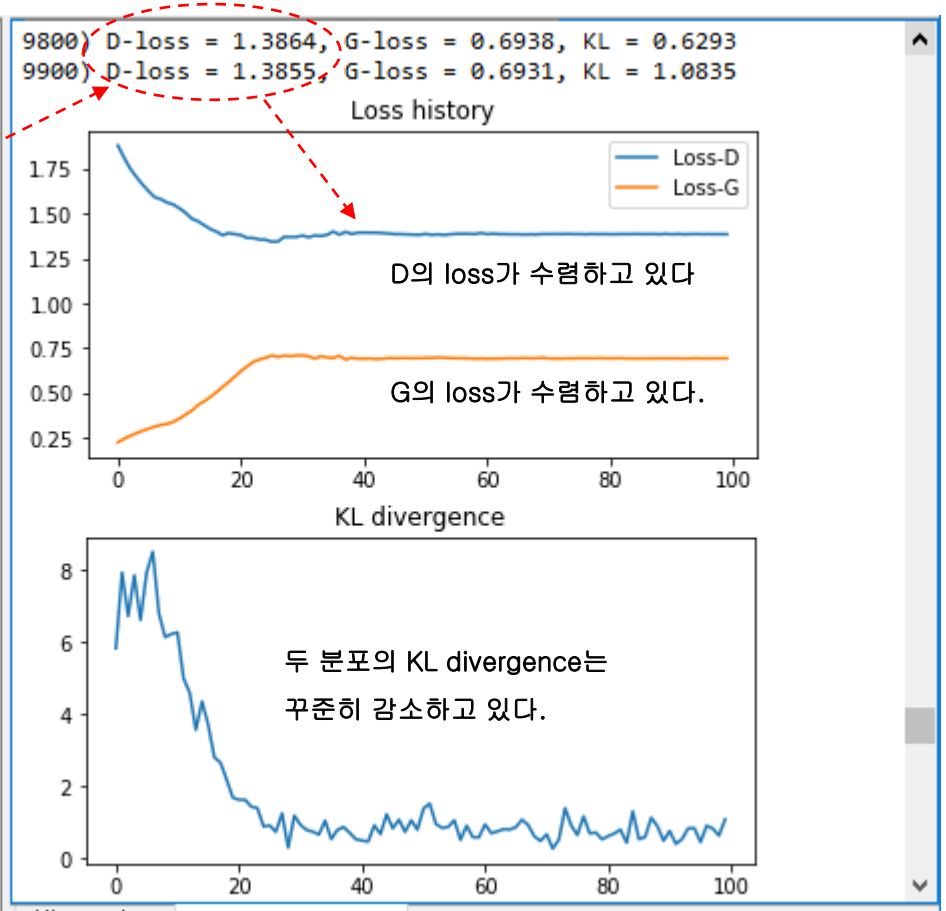
(실습 파일 : 6-1.GAN(Normal).py)
(실습 파일 : 6-2.GAN(slim).py)

- 학습 진행 과정을 관찰해 보면, Discriminator의 loss는 감소하다가 점차 증가하면서 어딘가로 수렴하고 있고 (균형점), Generator의 loss도 꾸준히 감소하여 균형점에 수렴하는 것을 볼 수 있다. Discriminator는 점차 진짜와 가짜를 구별하지 못하는 상태가 되고, Generator는 점차 진짜같은 가짜를 만들어내고 있는 것이다.
- Real data와 fake data 분포의 KL divergence를 관찰해 보면 학습이 진행될수록 점차 작아지는 것을 볼 수 있다. 두 분포가 점점 비슷해지고 있는 것이다.

```
100 nBatchSize = int(realData.shape[0] / nBatchCnt)  # 블록 당 Size
101 nK = 2              # Discriminator 학습 횟수 (위 논문에서는 nK = 1을 사용하였음)
102 k = 0
103 for i in range(10000):
104     # Mini-batch 방식으로 학습한다
105     np.random.shuffle(realData)
106
107     for n in range(nBatchCnt):
108         # input 데이터를 Mini-batch 크기에 맞게 자른다
109         nFrom = n * nBatchSize
110         nTo = n * nBatchSize + nBatchSize
111
112         # 마지막 루프이면 nTo는 input 데이터의 끝까지.
113         if n == nBatchCnt - 1:
114             nTo = realData.shape[0]
115
116         # 학습 데이터를 준비한다
117         bx = realData[nFrom : nTo]
118         bz = getNoise(m=bx.shape[0], n=nGInput)
119
120         if k < nK:
121             # Discriminator를 nK-번 학습한다.
122             _, lossDHist = sess.run([trainD, D_loss], feed_dict={x : bx, z : bz})
123             k += 1
124         else:
125             # Generator를 1-번 학습한다.
126             _, lossGHist = sess.run([trainG, G_loss], feed_dict={x : bx, z : bz})
127             k = 0
```

G를 학습할수록 D-loss는
log(4) = 1.3863 으로 수렴한다.
(GAN의 이론적 근거 편 참조)

```
9800) D-loss = 1.3864, G-loss = 0.6938, KL = 0.6293
9900) D-loss = 1.3855, G-loss = 0.6931, KL = 1.0835
```



GAN 학습 예시 (1) : 정규분포 데이터 생성

(실습 파일 : 6-1.GAN(Normal).py)
(실습 파일 : 6-2.GAN(slim).py)

- 학습을 완료한 후 Generator에 임의의 랜덤 데이터를 넣어 $G(z)$를 생성하고 $G(z)$를 Discriminator에 넣으면 Discriminator는 0.5에 가까운 값을 출력한다.
- 이것은 G(z)가 50%의 확률로 real data일 수도 있고, fake data일 수도 있다는 것으로, Real인지 fake인지 잘 모르겠다는 의미이다. 즉, fake data가 real data를 많이 닮아있다는 것을 의미한다. 학습이 완료된 후 150번 라인으로 fake data를 무한히 생성할 수 있다.

```
136
137 plt.figure(figsize=(6, 3))
138 plt.plot(histLossD, label='Loss-D')
139 plt.plot(histLossG, label='Loss-G')
140 plt.legend()
141 plt.title("Loss history")
142 plt.show()
143
144 plt.figure(figsize=(6, 3))
145 plt.plot(histKL)
146 plt.title("KL divergence")
147 plt.show()
148
149 # real data 분포 (p)와 fake data 분포 (q)를 그려본다
150 fakeData = sess.run(Gz, feed_dict={z : getNoise(m=realData.shape[0], n=nGInput)})
151 p, q, kld = KL(realData, fakeData)
152 plt.plot(p, color='blue', linewidth=2.0, alpha=0.7, label='Real Data')
153 plt.plot(q, color='red', linewidth=2.0, alpha=0.7, label='Fake Data')
154 plt.legend()
155 plt.title("Distibution of Real and Fake Data")
156 plt.show()
157 print("KL divergence = %.4f" % kld)
158
159 # Fake Data를 Discriminator에 넣었을 때 출력값을 확인해 본다.
160 outputD = sess.run(Dx, feed_dict={x : fakeData})
161 print(outputD.T)
162 sess.close()
163
```

임의의 z로 생성한 $G(z)$가 real data일 확률이 모두 0.5로 출력되고 있다.

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \rightarrow p_g(x) = p_{data}(x)$$

```
 0.50446224 0.48668286 0.50807625 0.5026261  0.49537635 0.5037954
 0.5213943  0.5038679  0.5030706  0.5030106  0.49624577 0.4956136
 0.5095958  0.49576408 0.50133026 0.50406265 0.48747805 0.51511174
 0.4879986  0.50326127 0.50618875 0.50951374 0.4871182  0.49836624
 0.5015312  0.50395566 0.49102062 0.50349104 0.5034095  0.502525
 0.505661   0.5046839  0.5044195  0.50508636 0.5034677  0.49249122
 0.50469303 0.5046607  0.48945034 0.5042712  0.49462858 0.5024765
 0.49580264 0.5059264  0.49170113 0.4981994  0.49657112 0.495147
 0.5055813  0.50453633 0.50040555 0.49787262 0.4934293  0.5027963
 0.50434756 0.504158   0.502575   0.50861114 0.49931026 0.49124566
 0.50361484 0.50247085 0.5028836  0.5029686  0.50439227 0.5072432
 0.50280297 0.4935034  0.50383145 0.5082691  0.511482   0.49808702
 0.50871843 0.49727893 0.50295144 0.500372   0.50495106 0.48957375
 0.5127157  0.50372845 0.49559104 0.5031764  0.5045577  0.5009951
 0.49542865 0.5081626  0.5037979  0.50293946 0.50096166 0.49553367
 0.48646578 0.50713634 0.49551097 0.49284706 0.50601315 0.5040159
 0.502451   0.48744395 0.50203323 0.4904965  0.50273895 0.49348852
 0.49878863 0.5056532  0.48876026 0.49508983 0.49565068 0.49413276
 0.49272537 0.5056712  0.5045847  0.49570936 0.49780786 0.4986278
 0.49495575 0.5029684  0.5035278  0.49518627 0.50336367 0.5049615
 0.50471294 0.50455034 0.50411004 0.49525294 0.4956786  0.50499856
 0.5046183  0.48703885 0.502488   0.5045463  0.49526462 0.49930942
 0.5047978  0.48621055 0.4975406  0.48997682 0.5032202  0.50347227
 0.49663487 0.502863   0.4916655  0.5102526  0.49481484 0.5046687
 0.50116026 0.49884978 0.5046469  0.5057215  0.50593156 0.4953141
 0.4958258  0.5033606  0.4951989  0.5078959  0.4880764  0.48715526
 0.5112362  0.5058866  0.502954   0.5107963 ]]

In [12]:
```

History log | IPython console

(실습 파일 : 6-3.GAN(4-dist).py)

GAN 학습 예시 (2) : 여러 개의 정규분포 데이터 생성

- 주어진 4 개의 정규분포 데이터를 흉내내서 이와 유사한 분포를 갖는 데이터를 생성한다.
- 아래 예시는 평균이 다른 지점에서 표준편차가 0.5인 4개의 정규분포를 데이터를 생성하고, 이와 유사한 분포의 데이터를 생성한 결과이다.
- Fake Data (빨간색 점)가 Real Data (파란색 점)에 잘 겹쳐지는 것을 확인할 수 있다.

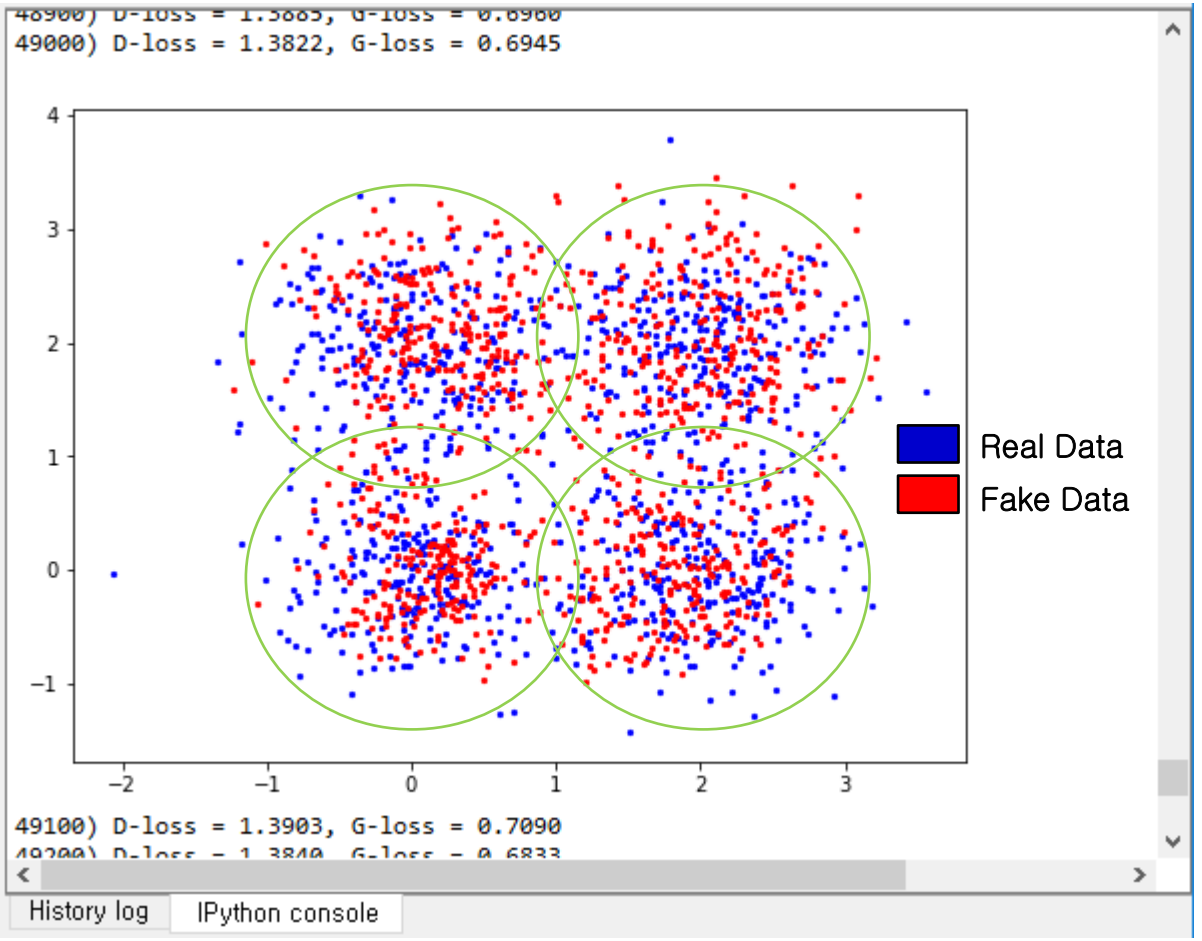
```python
# 데이터 세트를 생성한다.
def createDataSet(n):
    xy = []
    for i in range(n):
        r = np.random.random()
        if r < 0.25:
            x = np.random.normal(0.0, 0.5)
            y = np.random.normal(0.0, 0.5)
        elif r < 0.50:
            x = np.random.normal(0.0, 0.5)
            y = np.random.normal(2.0, 0.5)
        elif r < 0.75:
            x = np.random.normal(2.0, 0.5)
            y = np.random.normal(0.0, 0.5)
        else:
            x = np.random.normal(2.0, 0.5)
            y = np.random.normal(2.0, 0.5)
        xy.append([x, y])

    return pd.DataFrame(xy, columns=['x', 'y'])

# 데이터 세트를 생성한다
ds = createDataSet(n=1000)
realData = np.array(ds)
nDataRow = realData.shape[0]
nDataCol = realData.shape[1]

nGInput = 10
nGHidden = 128
```

주어진 4개의 정규 분포 (Real Data)



참고 사항 : GAN 학습의 이론적 근거

- G를 고정한 상태에서 Loss 함수인 $V(D,G)$ 를 D에 대해 학습하면 아래와 같이 $D^*(x)$가 되어, $p_g(x) \to p_{data}(x)$ 일때 $D^*(x) = 0.5$가 된다.
- $D^*(x)$ 를 $V(D,G)$에 대입하면 $p_{data}(x)$ 와 $p_g(x)$ 의 KL divergence 혹은 Jensen-Shannon divergence로 표현된다.
- G를 학습하면 ($V^*(D,G)$ 가 최소가 되도록 하면) $JSD(p_{data}||p_g)$ 가 최소가 되므로, $p_{data}(x)$ 와 $p_g(x)$ 의 분포가 유사해 진다.
- G를 학습하면 $JSD(p_{data}||p_g) \to 0$ 에 수렴하고, $V^*(D,G) \to -log4 = -1.3863$ 에 수렴한다 (Global minimum).
  → 예시 (1) 에서는 D-loss가 1.3864, 1.3855 부근으로 수렴하고 있다.

$$V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1-D(G(z)))]$$

$$= E_{x\sim p_{data}(x)}[logD(x)] + E_{x\sim p_g(x)}[log(1-D(x))]$$

$$= \int_x \left[p_{data}(x)logD(x) + p_g(x)log(1-D(x))\right] dx$$

이 부분을 maximize (fixed G)

$$\max_D V(D,G) \to \frac{\partial[\,]}{\partial D} = 0 \to \frac{p_{data}(x)}{D(x)} - \frac{p_g(x)}{1-D(x)} = 0$$

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}, \quad p_g(x) \to p_{data}(x), \quad D^*(x) \to \frac{1}{2}$$

G를 고정하고 D를 학습한 결과

$$Let, \quad p_{data}(x) = d, \quad p_g(x) = g \qquad \int_x p_{data}(x)dx = 1$$

$$V^*(D,G) = \int_x d \cdot log\frac{d}{d+g}dx + \int_x g \cdot log\frac{g}{d+g}dx$$

$$= -log4 + \int_x d \cdot log\frac{2d}{d+g}dx + \int_x g \cdot log\frac{2g}{d+g}dx$$

$$= -log4 + KL\left(p_{data}||\frac{p_{data}+p_g}{2}\right) + KL\left(p_g||\frac{p_{data}+p_g}{2}\right)$$

$$= -log4 + 2 \cdot JSD(p_{data}||p_g)$$

$$JSD(p||q) = \frac{1}{2}KL\left(p||\frac{p+q}{2}\right) + \frac{1}{2}KL\left(q||\frac{p+q}{2}\right)$$ ← Jensen-Shannon Divergence

$$\min_G V^*(D,G) \to \min_G JSD(p_{data}||p_g)$$ ← G를 학습하면 두 분포의 JSD가 최소가 된다.