GAN(Generative Adversarial Networks)

In [7]:

```
import os, warnings
# 경고 메시지 무시하거나 숨길때(ignore), 다시보이게(default)
# warnings.filterwarnings(action='default')
warnings.filterwarnings(action='ignore')
from IPython.display import display, Image
```

01 데이터 가져오기

In [8]:

```
# https://arxiv.org/abs/1406.2661
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)
```

```
Extracting ./mnist/data/train-images-idx3-ubyte.gz
Extracting ./mnist/data/train-labels-idx1-ubyte.gz
Extracting ./mnist/data/t10k-images-idx3-ubyte.gz
Extracting ./mnist/data/t10k-labels-idx1-ubyte.gz
```

02 기본 변수 설정

In [9]:

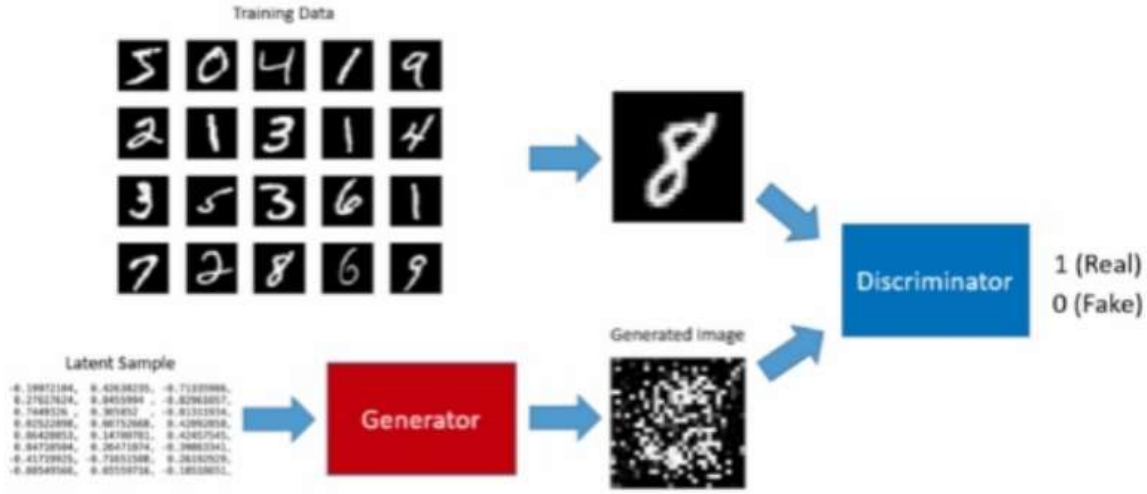
```
total_epoch = 100       # epoch 수 설정
batch_size = 100        # 배치 사이즈
learning_rate = 0.0002  # 학습률

# 신경망 레이어 구성 옵션
n_hidden = 256          # 은닉층 노드
n_input = 28 * 28       # 입력
n_noise = 128           # 생성기의 입력값으로 사용할 노이즈의 크기
```

03 신경망 모델 구성

In [11]:

```
display(Image(filename="../img/gan01.png"))
```



- https://go-hard.tistory.com/1 (https://go-hard.tistory.com/1) 참조

노이즈 입력(Z), 실제 입력(X)

In [13]:

```
# 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])
```

생성자 신경망 변수 선언

- n_noise : 128, n_hidden : 256
- G_W1, G_b1 : 첫번째 입력에서 은닉층으로 보내는 가중치와 편향변수
- G_W2, G_b2 : 출력층에서 사용할 편향변수

In [15]:

```
# 신경망 레이어 구성 옵션
# n_noise = 128          # 생성기의 입력값으로 사용할 노이즈의 크기
# n_hidden = 256         # 은닉층 노드

# n_input = 28 * 28      # 입력
```

In [14]:

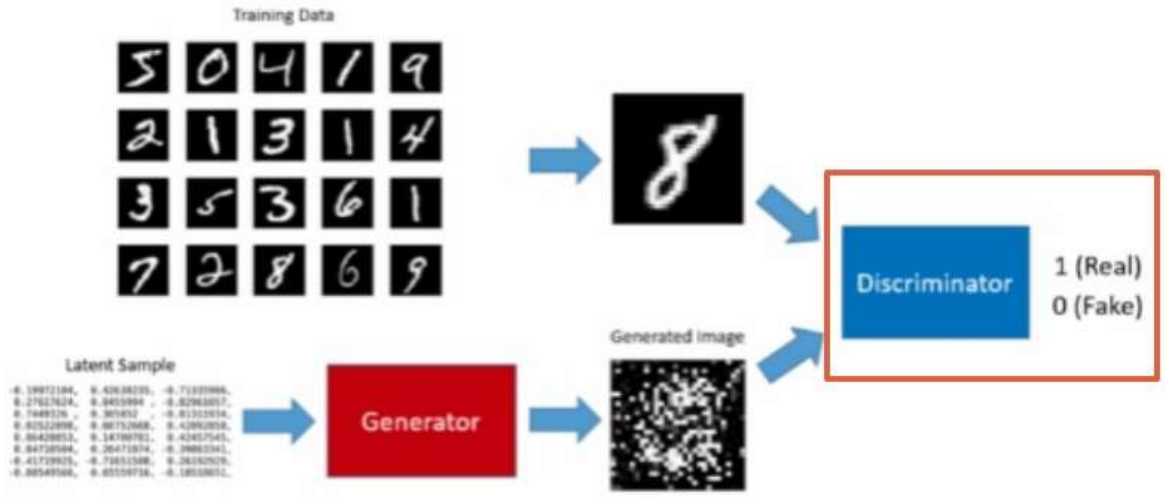
```
G_W1 = tf.Variable(tf.random_normal([n_noise, n_hidden], stddev=0.01))
G_b1 = tf.Variable(tf.zeros([n_hidden]))
G_W2 = tf.Variable(tf.random_normal([n_hidden, n_input], stddev=0.01))
G_b2 = tf.Variable(tf.zeros([n_input]))
```

```
WARNING:tensorflow:From C:\Anaconda3\lib\site-packages\tensorflow\python\fra
mework\op_def_library.py:263: colocate_with (from tensorflow.python.framewor
k.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

판별자 신경망 변수 선언

In [17]:

```
display(Image(filename="../img/gan02.png"))
```



In [19]:

```
# 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]))
```

04 신경망 구성(생성자(G), 구분자(D))

생성자(G) 신경망 구성

- 무작위 생성한 노이즈를 받아, 가중치와 편향을 반영하여 은닉층 구성.
- sigmoid 함수를 이용하여 최종 결과값 0~1 사이의 값 반환

In [20]:

```
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
```

구분자(D) 신경망 구성

- 구분자 신경망 구성, 가중치와 편향을 반영한 데이터 출력
- sigmoid 함수를 이용하여 최종 결과값 0~1 사이의 값 반환

In [22]:

```
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
```

노이즈 생성 함수

In [23]:

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

노이즈를 이용한 랜덤한 이미지 생성 및 판별망 신경망 이용 판별값 구하기

In [24]:

```
# 노이즈를 이용해 랜덤한 이미지를 생성합니다.
# Z에는 실행 시, noise가 입력됨.
G = generator(Z)
# 노이즈를 이용해 생성한 이미지가 진짜 이미지인지 판별한 값을 구합니다.
D_fake = discriminator(G)  # 가짜 이미지 넣기
# 진짜 이미지를 이용해 판별한 값을 구합니다.
D_real = discriminator(X)  # 진짜 이미지 넣기
```

In [26]:

```
print(G)
print(D_fake)
print(D_real)
```

```
Tensor("Sigmoid:0", shape=(?, 784), dtype=float32)
Tensor("Sigmoid_1:0", shape=(?, 1), dtype=float32)
Tensor("Sigmoid_2:0", shape=(?, 1), dtype=float32)
```

- GAN은 생성자(Generator) : 구분자가 1로 예측하도록 하는 것을 목표로 학습시킴.
- GAN은 구분자(Discriminator) : 진짜 데이터를 받으면 1로 가짜 데이터를 받으면 0으로 예측하도록 학습시킴.

05 GAN 모델의 최적화

- loss_G와 loss_D를 최대화 하는 것. 단, 서로의 손실이 연관되어 있어, 두 손실값이 같이 증가가 어려움.
- loss_D를 최대화하기 위해서는 D_gene값을 최소화시킴.
- 판별기에 진짜 이미지를 넣었을 때에도 최대값을 : tf.log(D_real)
- 가짜 이미지를 넣었을 때에도 최대값을 : tf.log(1 - D_gene)

In [27]:

```
loss_D = tf.reduce_mean(tf.log(D_real) + tf.log(1 - D_fake))
```

- loss_G(생성자 손실)를 최대화하기 위해서는 D_gene값을 최대화 한다.
- 가짜 이미지를 넣었을 때, 판별기가 최대한 실제 이미지라고 판단하도록 생성기 신경망을 학습

In [28]:

```
# 결국 D_gene 값을 최대화하는 것이므로 다음과 같이 사용할 수 있습니다.
loss_G = tf.reduce_mean(tf.log(D_fake))
```

loss_D 를 구할 때는 판별기 신경망에 사용되는 변수만 사용

loss_G 를 구할 때는 생성기 신경망에 사용되는 변수만 사용

In [30]:

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

In [31]:

```
# GAN 논문의 수식에 따르면 loss 를 극대화 해야하지만, minimize 하는 최적화 함수를 사용하기 때
# 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)

```

06 모델학습

In [32]:

```
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
```

In [33]:

```
%%time

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!')
```

```
Epoch: 0000 D loss: -0.4394 G loss: -1.997
Epoch: 0001 D loss: -0.4831 G loss: -2.333
Epoch: 0002 D loss: -0.1296 G loss: -3.186
Epoch: 0003 D loss: -0.6119 G loss: -1.394
Epoch: 0004 D loss: -0.3188 G loss: -2.003
Epoch: 0005 D loss: -0.2194 G loss: -2.416
Epoch: 0006 D loss: -0.1212 G loss: -3.344
Epoch: 0007 D loss: -0.1976 G loss: -2.758
Epoch: 0008 D loss: -0.4429 G loss: -2.5
Epoch: 0009 D loss: -0.3757 G loss: -2.493
Epoch: 0010 D loss: -0.4561 G loss: -2.554
Epoch: 0011 D loss: -0.4461 G loss: -2.234
Epoch: 0012 D loss: -0.3111 G loss: -2.569
Epoch: 0013 D loss: -0.323 G loss: -2.285
Epoch: 0014 D loss: -0.367 G loss: -2.661
Epoch: 0015 D loss: -0.4759 G loss: -2.396
Epoch: 0016 D loss: -0.3834 G loss: -2.566
Epoch: 0017 D loss: -0.2517 G loss: -2.641
Epoch: 0018 D loss: -0.2633 G loss: -2.608
Epoch: 0019 D loss: -0.3133 G loss: -2.693
Epoch: 0020 D loss: -0.3645 G loss: -2.71
Epoch: 0021 D loss: -0.3366 G loss: -2.698
```

```
Epoch: 0022 D loss: -0.4076 G loss: -2.211
Epoch: 0023 D loss: -0.3359 G loss: -2.892
Epoch: 0024 D loss: -0.4418 G loss: -2.35
Epoch: 0025 D loss: -0.3847 G loss: -2.666
Epoch: 0026 D loss: -0.3455 G loss: -2.887
Epoch: 0027 D loss: -0.3744 G loss: -2.603
Epoch: 0028 D loss: -0.498 G loss: -2.35
Epoch: 0029 D loss: -0.5374 G loss: -2.465
Epoch: 0030 D loss: -0.5066 G loss: -2.49
Epoch: 0031 D loss: -0.5452 G loss: -2.206
Epoch: 0032 D loss: -0.4928 G loss: -2.498
Epoch: 0033 D loss: -0.6053 G loss: -2.24
Epoch: 0034 D loss: -0.5987 G loss: -2.406
Epoch: 0035 D loss: -0.7317 G loss: -2.226
Epoch: 0036 D loss: -0.604 G loss: -2.486
Epoch: 0037 D loss: -0.5141 G loss: -2.353
Epoch: 0038 D loss: -0.5338 G loss: -2.236
Epoch: 0039 D loss: -0.5697 G loss: -2.033
Epoch: 0040 D loss: -0.7049 G loss: -2.202
Epoch: 0041 D loss: -0.6726 G loss: -2.047
Epoch: 0042 D loss: -0.7254 G loss: -2.085
Epoch: 0043 D loss: -0.5568 G loss: -2.385
Epoch: 0044 D loss: -0.6436 G loss: -1.922
Epoch: 0045 D loss: -0.6978 G loss: -2.618
Epoch: 0046 D loss: -0.8081 G loss: -1.784
Epoch: 0047 D loss: -0.7397 G loss: -1.863
Epoch: 0048 D loss: -0.6723 G loss: -2.029
Epoch: 0049 D loss: -0.8234 G loss: -1.516
Epoch: 0050 D loss: -0.7471 G loss: -1.995
Epoch: 0051 D loss: -0.779 G loss: -2.029
Epoch: 0052 D loss: -0.6543 G loss: -1.99
Epoch: 0053 D loss: -0.7047 G loss: -2.061
Epoch: 0054 D loss: -0.9042 G loss: -1.917
Epoch: 0055 D loss: -0.8167 G loss: -2.095
Epoch: 0056 D loss: -0.6696 G loss: -1.938
Epoch: 0057 D loss: -0.8221 G loss: -1.863
Epoch: 0058 D loss: -0.7519 G loss: -2.008
Epoch: 0059 D loss: -0.8165 G loss: -1.903
Epoch: 0060 D loss: -0.691 G loss: -1.917
Epoch: 0061 D loss: -0.7551 G loss: -1.858
Epoch: 0062 D loss: -0.6572 G loss: -2.212
Epoch: 0063 D loss: -0.8745 G loss: -1.932
Epoch: 0064 D loss: -0.7544 G loss: -1.884
Epoch: 0065 D loss: -0.8797 G loss: -1.919
Epoch: 0066 D loss: -0.7726 G loss: -1.991
Epoch: 0067 D loss: -0.7727 G loss: -2.042
Epoch: 0068 D loss: -0.7823 G loss: -2.01
Epoch: 0069 D loss: -0.878 G loss: -1.864
Epoch: 0070 D loss: -0.7444 G loss: -1.955
Epoch: 0071 D loss: -0.6237 G loss: -2.14
Epoch: 0072 D loss: -1.004 G loss: -1.82
Epoch: 0073 D loss: -0.6804 G loss: -1.923
Epoch: 0074 D loss: -0.7347 G loss: -1.729
Epoch: 0075 D loss: -0.7332 G loss: -1.79
Epoch: 0076 D loss: -0.9578 G loss: -1.589
Epoch: 0077 D loss: -0.7974 G loss: -2.085
Epoch: 0078 D loss: -0.7918 G loss: -1.963
Epoch: 0079 D loss: -0.7984 G loss: -1.814
Epoch: 0080 D loss: -0.7293 G loss: -1.99
Epoch: 0081 D loss: -0.7594 G loss: -1.775
Epoch: 0082 D loss: -0.8765 G loss: -1.645
```

```
Epoch: 0083 D loss: -0.6829 G loss: -1.896
Epoch: 0084 D loss: -0.815 G loss: -1.938
Epoch: 0085 D loss: -0.6896 G loss: -1.874
Epoch: 0086 D loss: -0.782 G loss: -1.999
Epoch: 0087 D loss: -0.7771 G loss: -1.794
Epoch: 0088 D loss: -0.95 G loss: -1.655
Epoch: 0089 D loss: -0.8118 G loss: -1.772
Epoch: 0090 D loss: -0.9103 G loss: -1.803
Epoch: 0091 D loss: -0.7695 G loss: -2.069
Epoch: 0092 D loss: -0.7436 G loss: -1.826
Epoch: 0093 D loss: -0.7155 G loss: -1.781
Epoch: 0094 D loss: -0.7494 G loss: -1.871
Epoch: 0095 D loss: -0.6962 G loss: -1.867
Epoch: 0096 D loss: -0.8299 G loss: -1.641
Epoch: 0097 D loss: -0.8731 G loss: -1.915
Epoch: 0098 D loss: -0.7524 G loss: -1.835
Epoch: 0099 D loss: -0.7041 G loss: -2.074
최적화 완료!
Wall time: 8min 1s
```

In [ ]:

```
1
```