GAN(Generative Adversarial Networks)

학습 내용

01. GAN의 구분모델과 생성모델

02. GAN에 대한 이해

03. GAN의 구성

04. GAN의 사용예

05. Tensorflow를 이용한 GAN 구현

01. GAN의 구분모델과 생성모델

- 어떤 값을 분류하거나 예측하는데 초점이 맞춰져 있다.
  - 이런 모델을 구분 모델(Discriminative Model)이라 한다.
- 생성모델(Generative Model) : 주어진 트레이닝 데이터의 특징을 학습 후, 트레이닝 데이터와 유사한 모델 생성

02 GAN 이해해 보기

- GAN:Generative Adversarial Networks 약자이다.
- 게임 이론(Game Theory)의 minimax two-player 게임의 구조를 이용해서 생성한 모델
- GAN 모델은 GoodFellow, Ian, et al의 'Generative adversarial nets.'라는 제목의 2014년도의 논문에서 최초 제안
- GAN은 생성형 인공지능 방식으로 실제 데이터(real data)와 비슷한 확률 분포를 가지는 허구 데이터(fake data)를 생성
- 허구 데이터는 GAN에서 만들어진 데이터이기 때문에 생성 데이터라고 한다.
- DNN은 레이블이 있는 정보를 학습하는 지도학습 방식이지만, GAN은 레이블이 없는 정보를 다루는 비지도 학습.

GAN에서의 입력 데이터는 무작위 잡음. 출력 데이터는 입력 데이터보다 높은 차원으로 구성

- GAN의 직관적인 이해
  - 경찰(구분자)와 위조 지폐 생성범(생성자)
  - 경찰의 역할 : 위조 지폐 생성범이 생성한 위조지폐와 진짜 지폐를 구분할 수 있도록 최대한 노력
  - 위조범(생성자) : 경찰을 속이기 위해 최대한 진짜 지폐와 구분이 되지 않는 위조지폐를 생성하기 위해 노력
  - 경찰의 학습과 위조범의 학습이 계속 진행해 가면 경찰이 위조지폐 생성범이 생성한 위조지폐와 진짜 지폐를 50% 확률로 구분할 수 있게 되는 균형점에서 학습이 종료됨.

결과 : 원본 데이터와 유사한 데이터의 분포를 학습하게 된다.

- 학습을 마친 GAN에 새로운 무작위 잡음을 입력하면 학습한 실제 데이터와 유사한 형태의 허구 데이터 출력
- 예를 들어 필기체나 숫자나 사람의 얼굴을 학습시키면 학습시킨 것과 같은 필기체 숫자나 사람 얼굴이 나옴.

03. GAN의 구성

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

GAN의 구성 예 - 사람의 얼굴

- 생성자 G는 임의의 잠재 변수(노이즈값)을 입력 받아, 가짜 이미지를 생성(Fake Image)를 생성
- 구분자 D는 진짜 이미지이면 1, 가짜 이미지이면 0의 값을 출력하는 것을 목표로 한다.
  - 즉 구분자(D)는 진짜 이미지, 가짜 이미지를 입력받는다.
- 생성자 G와 구분자 D는 임의의 머신 러닝 모델(예를 들면 소프트 맥스, SVM등)을 사용하여 구현한다.
  - 일반적으로 인공신경망을 이용하여 구현한다.

04. GAN의 사용예

In [1]:

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

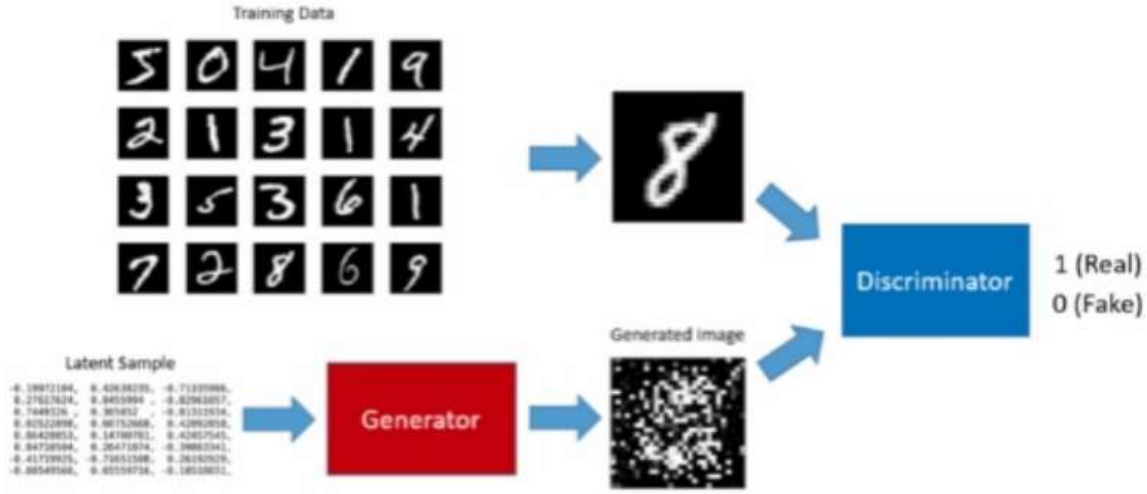
In [2]:

```
!pwd
```

```
/content
```

In [3]:

```
1 display(Image(filename="/content/img/GAN00.png"))
```

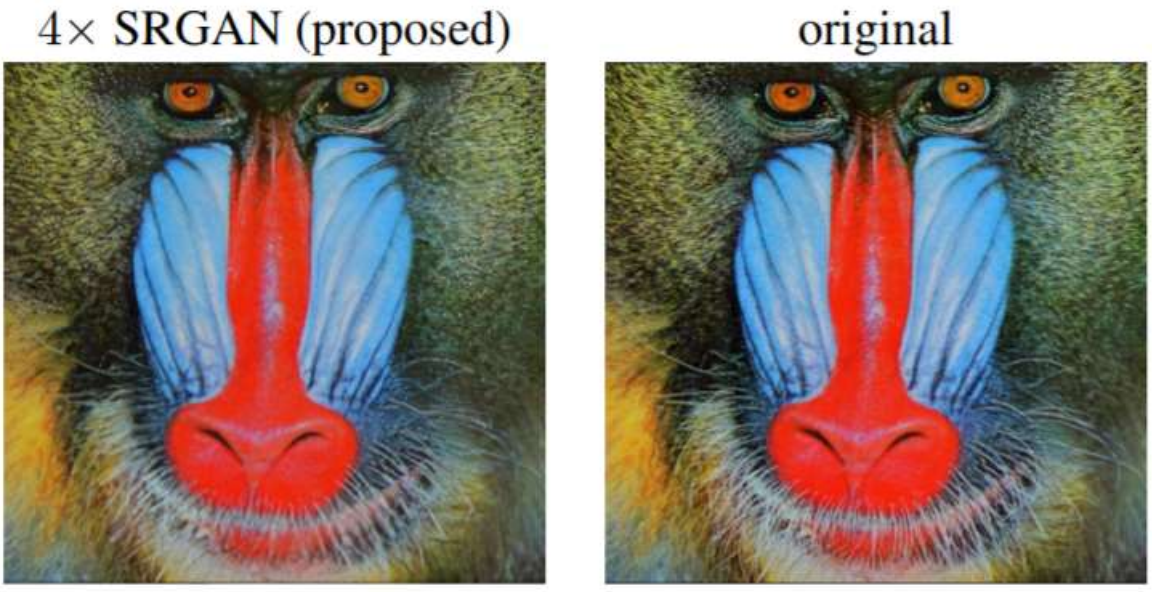


슈퍼 해상도 이미지

- 작은 이미지에서 고해상도 이미지를 만들어 낸다.
- https://arxiv.org/pdf/1609.04802.pdf (https://arxiv.org/pdf/1609.04802.pdf)

In [4]:

```
1 display(Image(filename="/content/img/GAN03.png"))
```

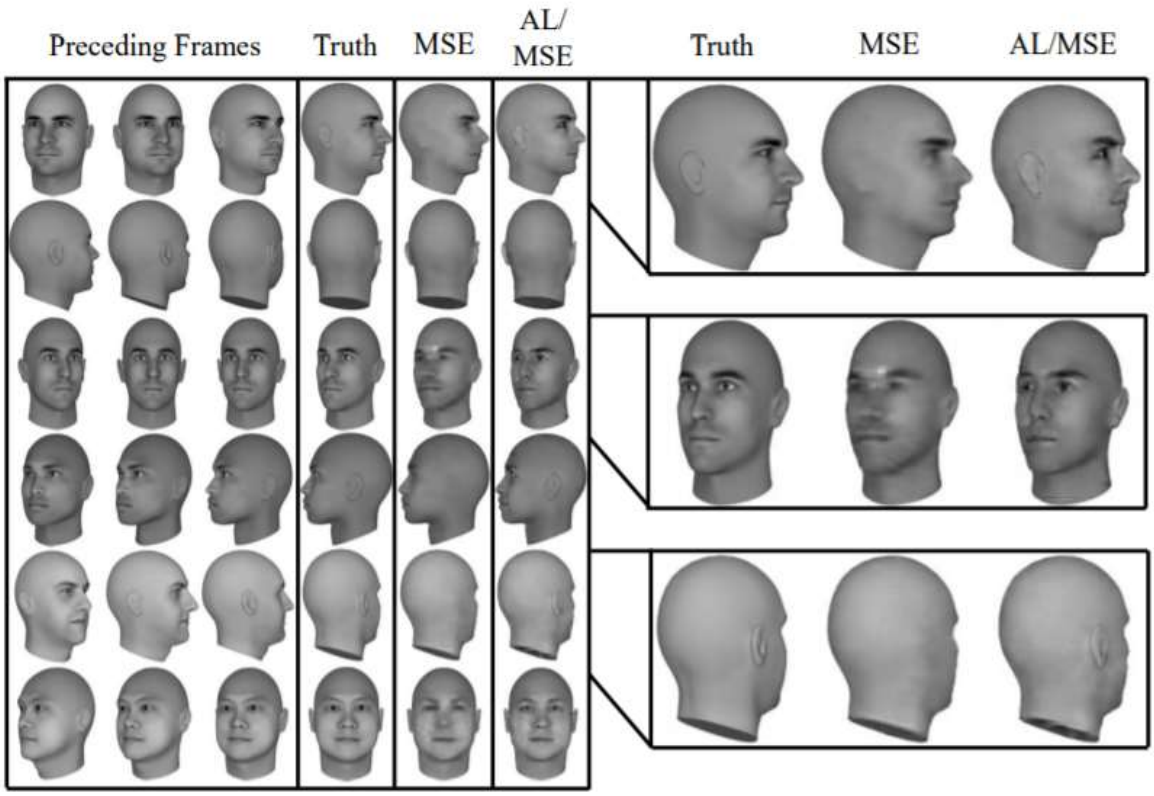


동영상의 다음 프레임 예측

- https://arxiv.org/pdf/1511.06380.pdf (https://arxiv.org/pdf/1511.06380.pdf)

In [5]:

```
1 display(Image(filename="/content/img/GAN02.png"))
```

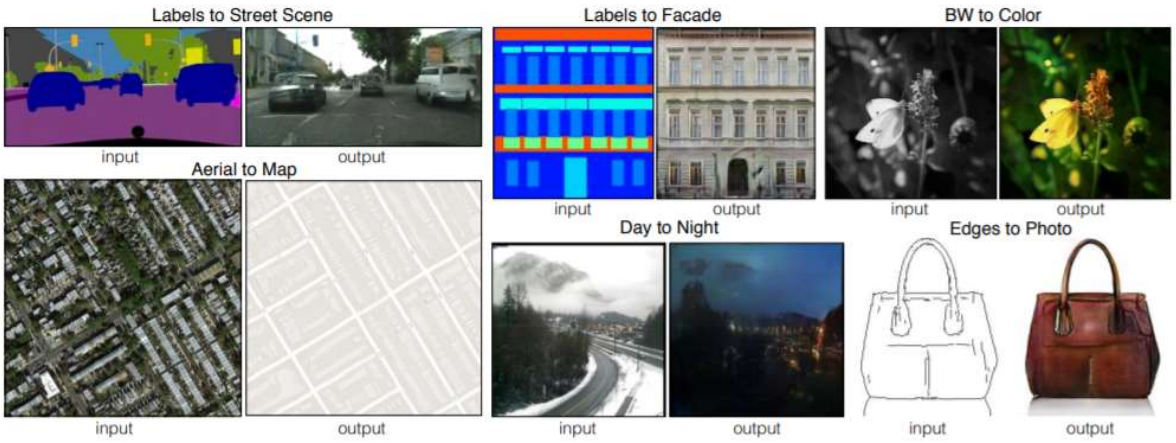


이미지를 다른 이미지로 변환

- https://arxiv.org/pdf/1611.07004.pdf (https://arxiv.org/pdf/1611.07004.pdf)

In [6]:

```
1 display(Image(filename="/content/img/GAN04.png"))
```



텍스트로 이미지 생성하기 : https://arxiv.org/pdf/1605.05396.pdf (https://arxiv.org/pdf/1605.05396.pdf)

불필요 제거 - 필요없는 부분 제거

In [7]:

```
1 display(Image(filename="/content/img/GAN05.png"))
```

훈련 데이터 생성(https://arxiv.org/pdf/1707.03124.pdf (https://arxiv.org/pdf/1707.03124.pdf))

새 애니메이션 캐릭터 만들기(https://arxiv.org/pdf/1708.05509.pdf (https://arxiv.org/pdf/1708.05509.pdf))

사진으로부터 3D 모델 생성

기타 GAN의 종류

- 바닐라 GAN
- 조건부 GAN : 사용자가 원하는 레이블로 이미지를 만든다.
- InfoGAN : 명시적 지도 훈련 없이도 필요한 레이블의 이미지를 생성할 수 있다.

GAN의 단점

- GAN이 생성한 이미지에는 계산, 원근감, 글로벌 구조와 같은 몇 가지 단점이 존재한다.
- 현재 모델을 개선하기 위해 단점들이 광범위하게 연구되고 있음.

05 Tensorflow를 활용한 Gan구현

01 데이터 가져오기

In [8]:

```
1 %tensorflow_version 1.x
```

```
TensorFlow 1.x selected.
```

In [9]:

```
# 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)
```

```
WARNING:tensorflow:From <ipython-input-9-4a88a2be3f8a>:7: read_data_sets
(from tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated
and will be removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/
models.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/contr
ib/learn/python/learn/datasets/mnist.py:260: maybe_download (from tensorfl
ow.contrib.learn.python.learn.datasets.base) is deprecated and will be rem
oved in a future version.
Instructions for updating:
Please write your own downloading logic.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/contr
ib/learn/python/learn/datasets/base.py:252: _internal_retry.<locals>.wrap.
<locals>.wrapped_fn (from tensorflow.contrib.learn.python.learn.datasets.b
ase) is deprecated and will be removed in a future version.
Instructions for updating:
Please use urllib or similar directly.
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/contr
ib/learn/python/learn/datasets/mnist.py:262: extract_images (from tensorfl
ow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be re
moved in a future version.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting ./mnist/data/train-images-idx3-ubyte.gz
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/contr
ib/learn/python/learn/datasets/mnist.py:267: extract_labels (from tensorfl
ow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be re
moved in a future version.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting ./mnist/data/train-labels-idx1-ubyte.gz
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/contr
ib/learn/python/learn/datasets/mnist.py:110: dense_to_one_hot (from tensor
flow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be
removed in a future version.
Instructions for updating:
Please use tf.one_hot on tensors.
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting ./mnist/data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting ./mnist/data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/contr
ib/learn/python/learn/datasets/mnist.py:290: DataSet.__init__ (from tensor
flow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be
removed in a future version.
Instructions for updating:
```

```
Please use alternatives such as official/mnist/dataset.py from tensorflow/
models.
```

02 기본 옵션 설정

In [10]:

```
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
```

02. 신경망 모델 구성

- 노이즈를 이용하여 데이터 생성
- 비지도학습이므로 Y가 없음

In [11]:

```
# 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
- 첫번째 가중치와 편향은 은닉층으로 출력하기 위한 변수
- 두 번째 가중치와 편향은 출력층에서 사용할 변수
- 두 번째 가중치의 변수 크기는 실제 이미지의 크기와 같아야 함.

In [12]:

```
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 신경망

- n_input = 28 * 28 # 입력
- n_hidden = 256 # 은닉층 노드
- 이곳에서 사용하는 변수는 공유함.(실제 이미지 판별, 생성한 이미지 판별)

In [13]:

```
# 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]))
```

2-1 생성자(G) 신경망 구성

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

In [14]:

```
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
```

2-2 구분자(D) 신경망 구성

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

In [15]:

```
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
```

In [16]:

```
# 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)

# 진짜 이미지를 이용해 판별한 값을 구합니다.
D_real = discriminator(X)
```

- 경찰을 학습(구분자가 가짜라고 판단하도록 하는 손실값)시킬 때,
  - D_real은 1에 가까워야 함.
  - 가짜 이미지 판별값(D_fake)은 0에 가까워야 함.

GAN의 모델의 최적화

- loss_G(생성자)와 loss_D(구분자)를 최대화 하는 것.
  - 단, 서로의 손실이 연관되어 있어, 두 손실값이 같이 증가가 어려움.
- loss_D를 최대화하기 위해서는 D_fake값을 최소화시킴.
- 판별기에 진짜 이미지를 넣었을 때에도 최대값을 : tf.log(D_real)
- 가짜 이미지를 넣었을 때에도 최대값을 : tf.log(1 - D_fake)

In [17]:

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

- D_fake(가짜 이미지 판별값)은 1에 가깝게 만들면 된다.(위조 지폐범의 학습)

- 위조 지폐범의 학습은 D_fake(가짜)값을 최대화 한다.
- 가짜 이미지를 넣었을 때, 판별기가 최대한 실제 이미지라고 판단하도록 생성기 신경망을 학습

In [18]:

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

In [19]:

```
# 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]
```

즉 GAN의 학습은 loss_D와 loss_G 를 최대화 하는것.

In [20]:

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

2-3 생성자의 입력인 노이즈 발생을 위한 노이즈 생성 함수

In [21]:

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

03 모델 학습

In [22]:

```
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
```

- samples 폴더 생성 필요

In [24]:

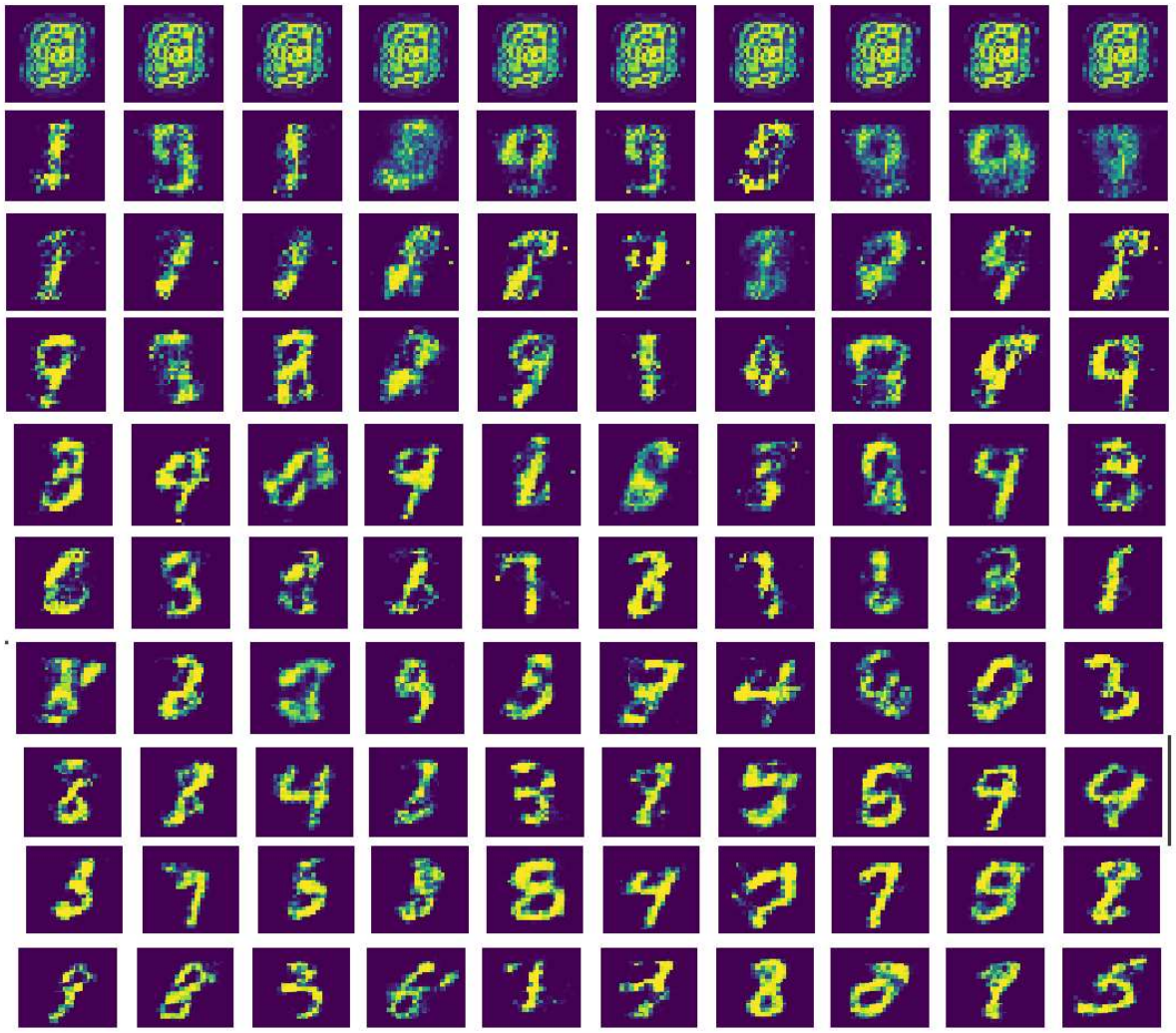
```
%%time

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)       # 노이즈 생성.
        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: 0022 D loss: -0.4852 G loss: -2.574
Epoch: 0023 D loss: -0.2561 G loss: -2.732
Epoch: 0024 D loss: -0.4315 G loss: -3.044
Epoch: 0025 D loss: -0.5236 G loss: -2.432
Epoch: 0026 D loss: -0.4617 G loss: -2.525
Epoch: 0027 D loss: -0.3115 G loss: -2.765
Epoch: 0028 D loss: -0.5227 G loss: -2.257
Epoch: 0029 D loss: -0.4608 G loss: -2.26
Epoch: 0030 D loss: -0.5712 G loss: -2.314
Epoch: 0031 D loss: -0.454 G loss: -2.794
Epoch: 0032 D loss: -0.5061 G loss: -2.753
Epoch: 0033 D loss: -0.4824 G loss: -2.462
Epoch: 0034 D loss: -0.663 G loss: -2.504
Epoch: 0035 D loss: -0.455 G loss: -2.7
Epoch: 0036 D loss: -0.4232 G loss: -2.625
Epoch: 0037 D loss: -0.5037 G loss: -2.545
Epoch: 0038 D loss: -0.5141 G loss: -2.2
Epoch: 0039 D loss: -0.4947 G loss: -2.224
Epoch: 0040 D loss: -0.5483 G loss: -2.286
Epoch: 0041 D loss: -0.4829 G loss: -2.161
```

10에폭마다의 이미지 생성된 것을 하나의 이미지로 정리해 보면 아래와 같다.

In [25]:

```
1 display(Image(filename="/content/img/GAN06.png"))
```



In [ ]:

```
1
```