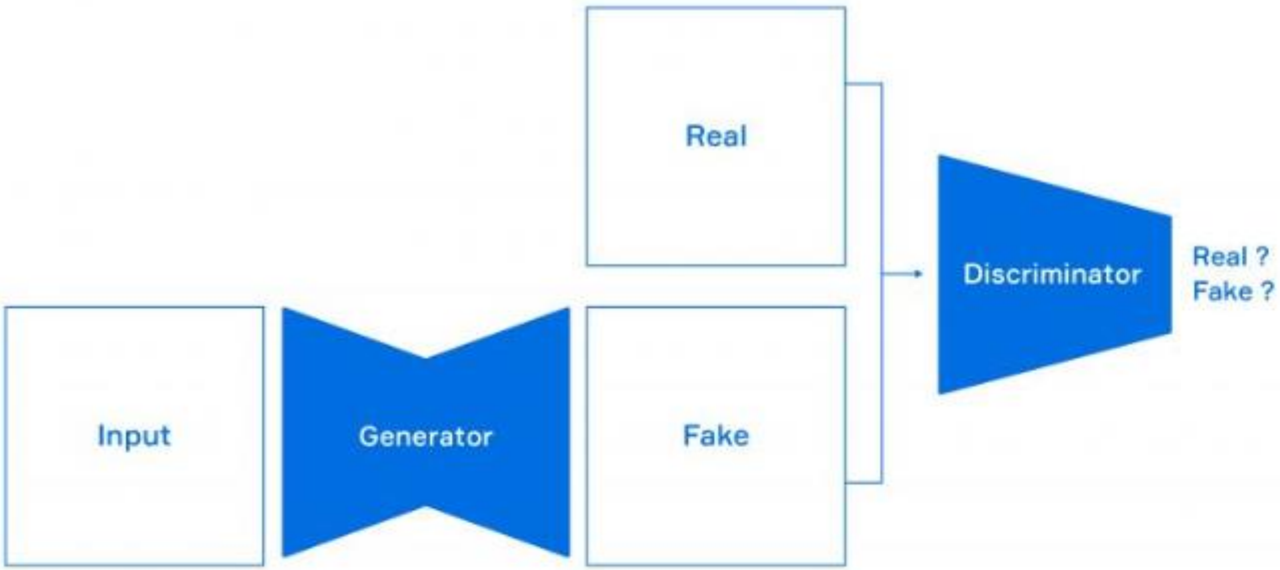
GAN

(Generative Adversarial Networks)

최우진

2018/10/16

Introduction to GAN



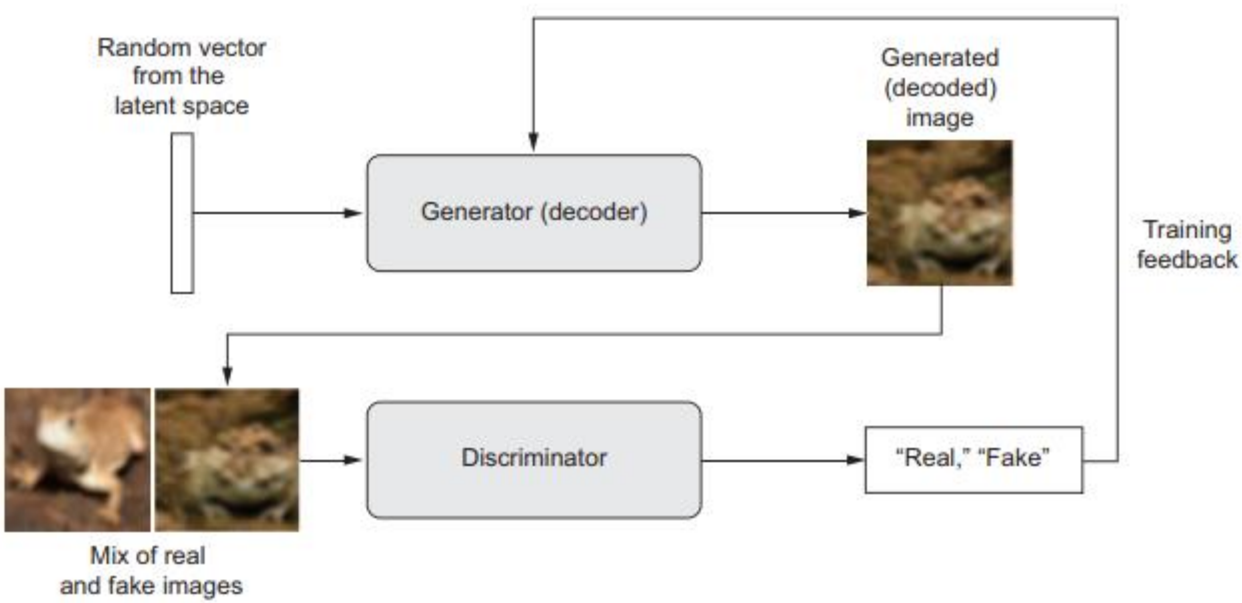
| GAN의 학습 과정 원리 〈출처: 네이버랩스〉

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

Introduction to GAN

That's what a GAN is: a forger network and an expert network, each being trained to best the other. As such, a GAN is made of two parts:

- *Generator network*—Takes as input a random vector (a random point in the latent space), and decodes it into a synthetic image
- *Discriminator network (or adversary)*—Takes as input an image (real or synthetic), and predicts whether the image came from the training set or was created by the generator network.



A schematic GAN implementation

- **A dataset of 50,000 32 x 32 RGB images**

1. A `generator` network maps vectors of shape `(latent_dim,)` to images of shape `(32, 32, 3)`.
2. A `discriminator` network maps images of shape `(32, 32, 3)` to a binary score estimating the probability that the image is real.
3. A `gan` network chains the generator and the discriminator together: `gan(x) = discriminator(generator(x))`. Thus this `gan` network maps latent space vectors to the discriminator's assessment of the realism of these latent vectors as decoded by the generator.
4. You train the discriminator using examples of real and fake images along with "real"/"fake" labels, just as you train any regular image-classification model.
5. To train the generator, you use the gradients of the generator's weights with regard to the loss of the `gan` model. This means, at every step, you move the weights of the generator in a direction that makes the discriminator more likely to classify as "real" the images decoded by the generator. In other words, you train the generator to fool the discriminator.

MNIST by GAN

```
# 설정값들을 선언합니다.
num_epoch = 100000
batch_size = 64
num_input = 28 * 28
num_latent_variable = 100
num_hidden = 128
learning_rate = 0.001
```

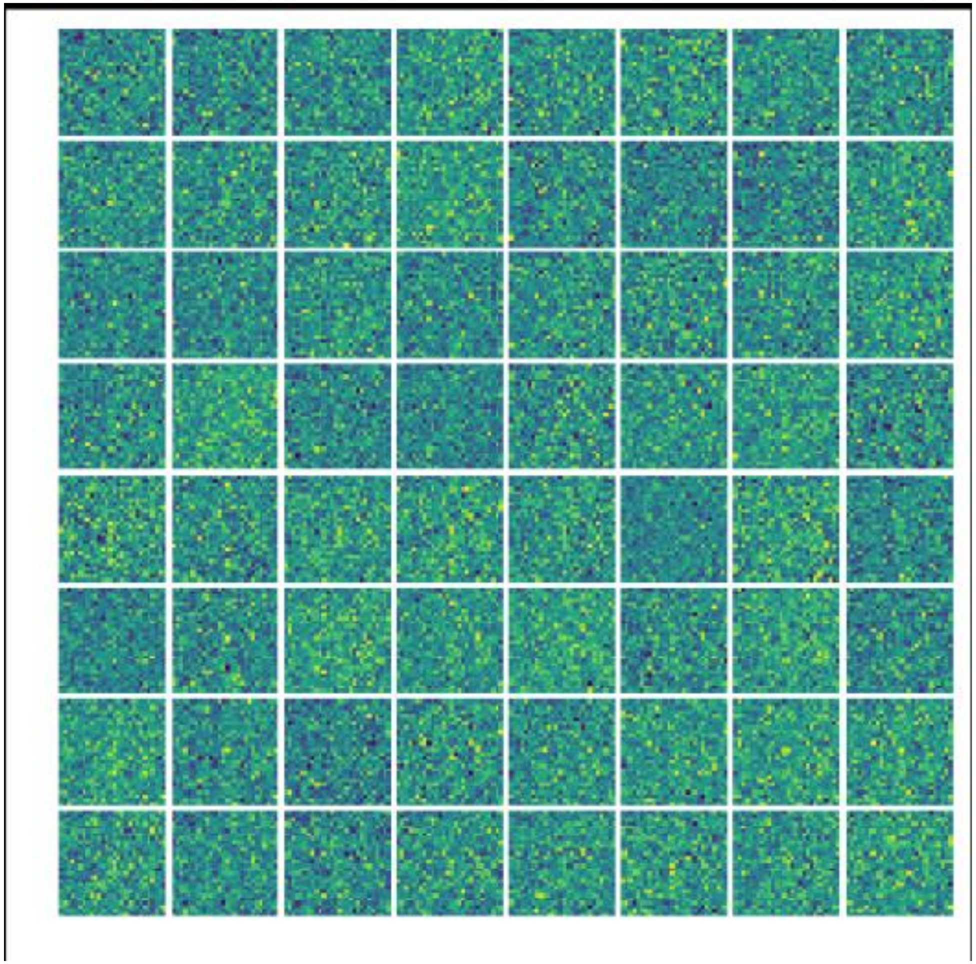
MNIST by GAN

```
# 플레이스 홀더를 선언합니다.
X = tf.placeholder(tf.float32, [None, num_input])                # 인풋 이미지
z = tf.placeholder(tf.float32, [None, num_latent_variable])      # 인풋 Latent Variable

# Generator 변수들 설정
# 100 -> 128 -> 784
with tf.variable_scope('generator'):
    # 히든 레이어 파라미터
    G_W1 = tf.Variable(tf.random_normal(shape=[num_latent_variable, num_hidden], stddev=5e-2))
    G_b1 = tf.Variable(tf.constant(0.1, shape=[num_hidden]))
    # 아웃풋 레이어 파라미터
    G_W2 = tf.Variable(tf.random_normal(shape=[num_hidden, num_input], stddev=5e-2))
    G_b2 = tf.Variable(tf.constant(0.1, shape=[num_input]))

# Discriminator 변수들 설정
# 784 -> 128 -> 1
with tf.variable_scope('discriminator'):
    # 히든 레이어 파라미터
    D_W1 = tf.Variable(tf.random_normal(shape=[num_input, num_hidden], stddev=5e-2))
    D_b1 = tf.Variable(tf.constant(0.1, shape=[num_hidden]))
    # 아웃풋 레이어 파라미터
    D_W2 = tf.Variable(tf.random_normal(shape=[num_hidden, 1], stddev=5e-2))
    D_b2 = tf.Variable(tf.constant(0.1, shape=[1]))
```

MNIST by GAN

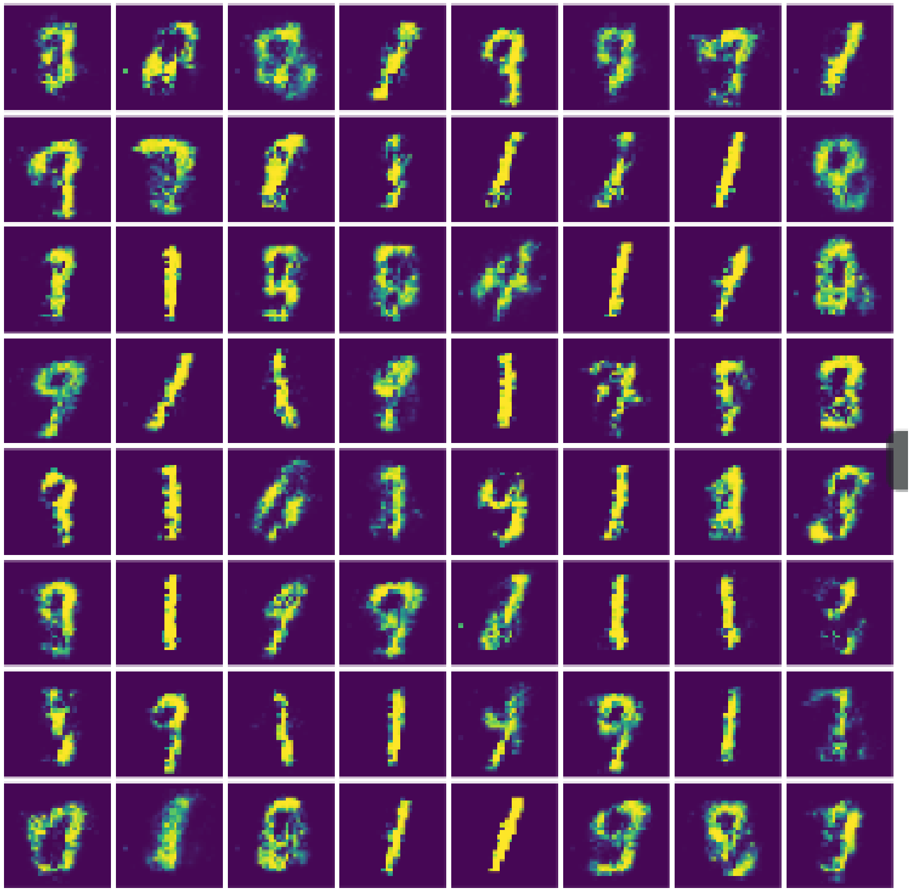
```
# Discriminator를 생성하는 함수를 정의합니다.
# Inputs:
#   X : 인풋 이미지
# Output:
#   predicted_value : Discriminator가 판단한 True(1) or Fake(0)
#   logits : sigmoid를 씌우기전의 출력값
def build_discriminator(X):
    hidden_layer = tf.nn.relu((tf.matmul(X, D_W1) + D_b1))
    logits = tf.matmul(hidden_layer, D_W2) + D_b2
    predicted_value = tf.nn.sigmoid(logits)

    return predicted_value, logits

# 생성자(Generator)를 선언합니다.
G = build_generator(z)

# 구분자(Discriminator)를 선언합니다.
D_real, D_real_logits = build_discriminator(X)  # D(x)
D_fake, D_fake_logits = build_discriminator(G)  # D(G(z))

# Discriminator의 손실 함수를 정의합니다.
d_loss_real = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=D_real_logits, labels=tf.ones_like(D_real_logits)))    # log(D(x))
d_loss_fake = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=D_fake_logits, labels=tf.zeros_like(D_fake_logits)))   # log(1-D(G(z)))
d_loss = d_loss_real + d_loss_fake  # log(D(x)) + log(1-D(G(z)))

# Generator의 손실 함수를 정의합니다.
g_loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=D_fake_logits, labels=tf.ones_like(D_fake_logits)))        # log(D(G(z))

# 전체 파라미터를 Discriminator와 관련된 파라미터와 Generator와 관련된 파라미터로 나눕니다.
tvar = tf.trainable_variables()
dvar = [var for var in tvar if 'discriminator' in var.name]
gvar = [var for var in tvar if 'generator' in var.name]

# Discriminator와 Generator의 Optimizer를 정의합니다.
d_train_step = tf.train.AdamOptimizer(learning_rate).minimize(d_loss, var_list=dvar)
g_train_step = tf.train.AdamOptimizer(learning_rate).minimize(g_loss, var_list=gvar)
```

MNIST by GAN



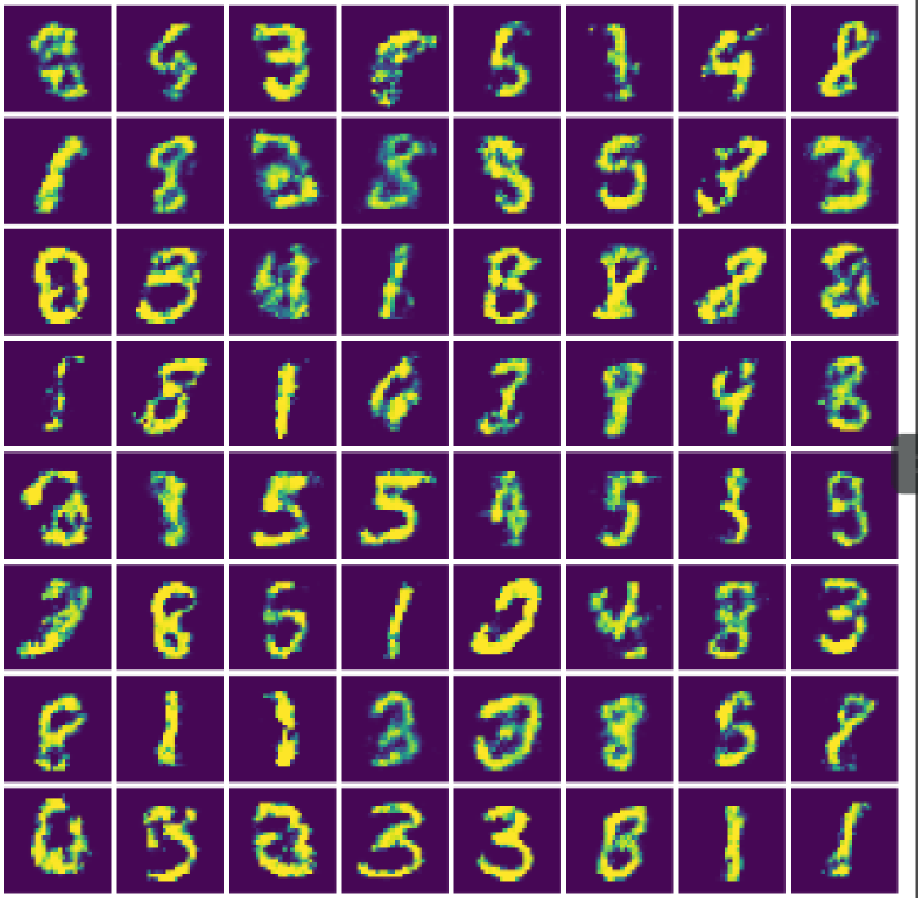
학습한 횟수: 0
=> 임의의 노이즈만 출력

MNIST by GAN



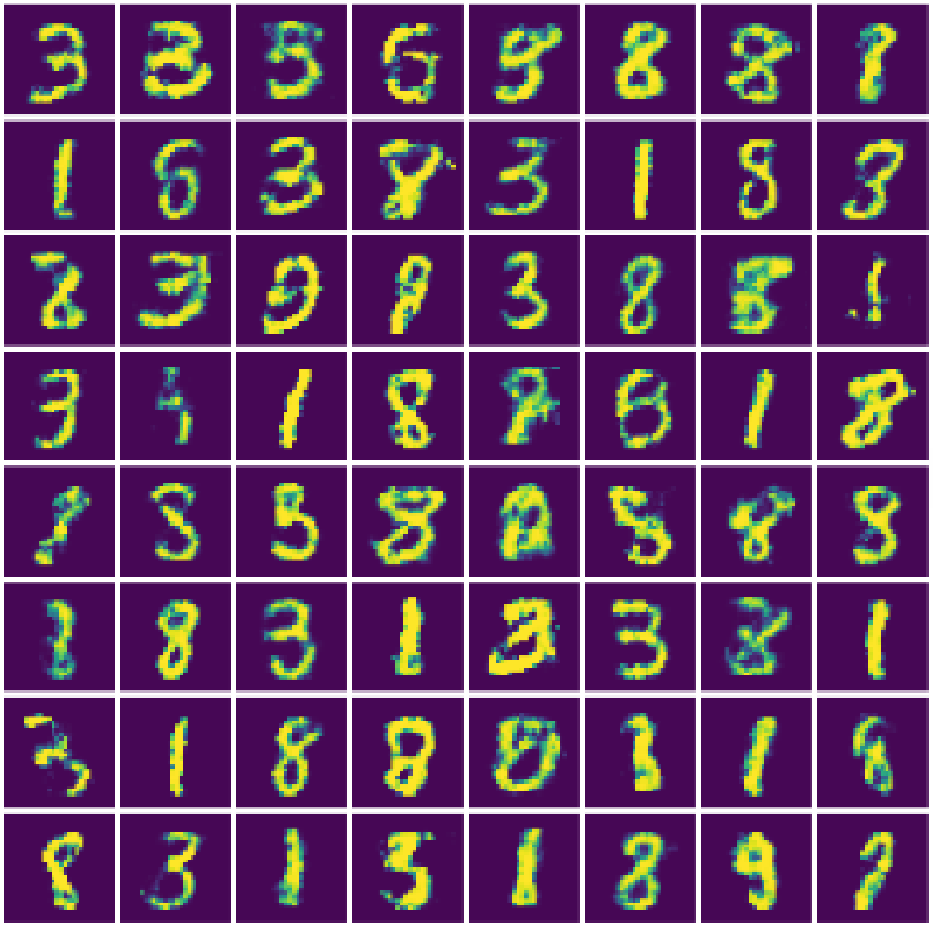
학습한 횟수: 1000

MNIST by GAN



학습한 횟수: 50000

MNIST by GAN



학습한 횟수: 100000

4장

감사합니다