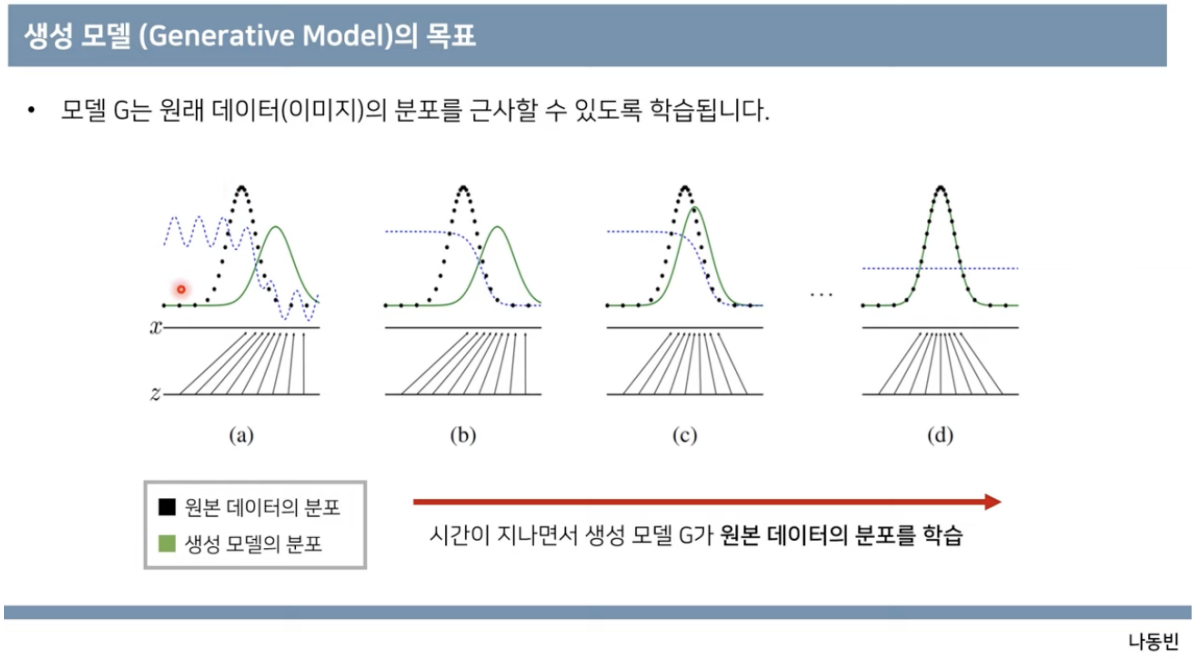
12주차 GAN

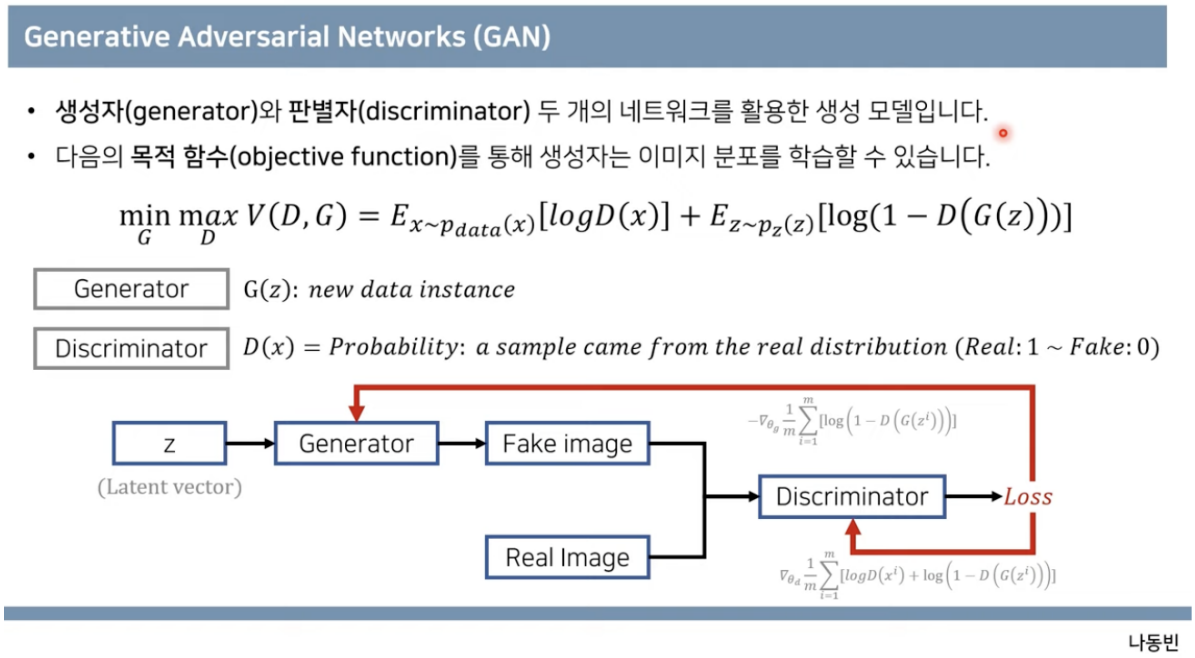
Convolution 연산 Dense 연산 → 결과 parameter값 유의

GAN 모델의 이해

https://www.youtube.com/watch?v=AVvlDmhHgC4&list=PLRx0vPvlEmdADpce8aoBhNnDaaHQN1Typ&index=9



10:00



12:23

BatchNormalization : 입력 데이터의 평균이 0 분산이 1이 되도록 재배치하는 것

1. 빠른 학습 속도
2. 가중치 초기화에 대한 민감도를 감소
3. 표준화를 대체 **
4. 기울기 소실 문제 *Gradient Descent 해결
5. **drop out 안해도 됨 → 과적합 방지**

```
import matplotlib.pyplot as plt
import numpy as np

import tensorflow as tf
import tensorflow as keras
from tensorflow.keras import initializers
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import BatchNormalization, Dense, LeakyReLU, Input
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam


(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
X_train=X_train.reshape(60000,28*28)
X_train=X_train.astype('float32')-127.5 /127.5
#성능이 더 좋은 방법을 택함 ( -1 ~ 1 사이의 값이 나올 수 있도록) -> 0~1이 아니라 표준정규분포를 띌 수 있도록 선택함
```

생성자 Generator

```
latent_dim=100
generator=Sequential()
generator.add(Dense(128,input_shape=(latent_dim,),kernel_initializer=initializers.RandomNormal(stdev=0.02),activation='relu'))
generator.add(BatchNormalization())

generator.add(Dense(256,actvation='relu'))
generator.add(BatchNormalization())

generator.add(Dense(512,activation='relu'))
generator.add(BatchNormalization())

generator.add(Dense(784,activation='tanh'))
```

Model: "sequential"

Layer (type) Output Shape Param

`dense (Dense)` `(None, 128)` `12928` (100 x 128 + 128)

`batch_normalization (BatchN ormalization)` `(None, 128)` `512`

`dense_1 (Dense)` `(None, 256)` `33024`

`batch_normalization_1 (Batc hNormalization)` `(None, 256)` `1024`

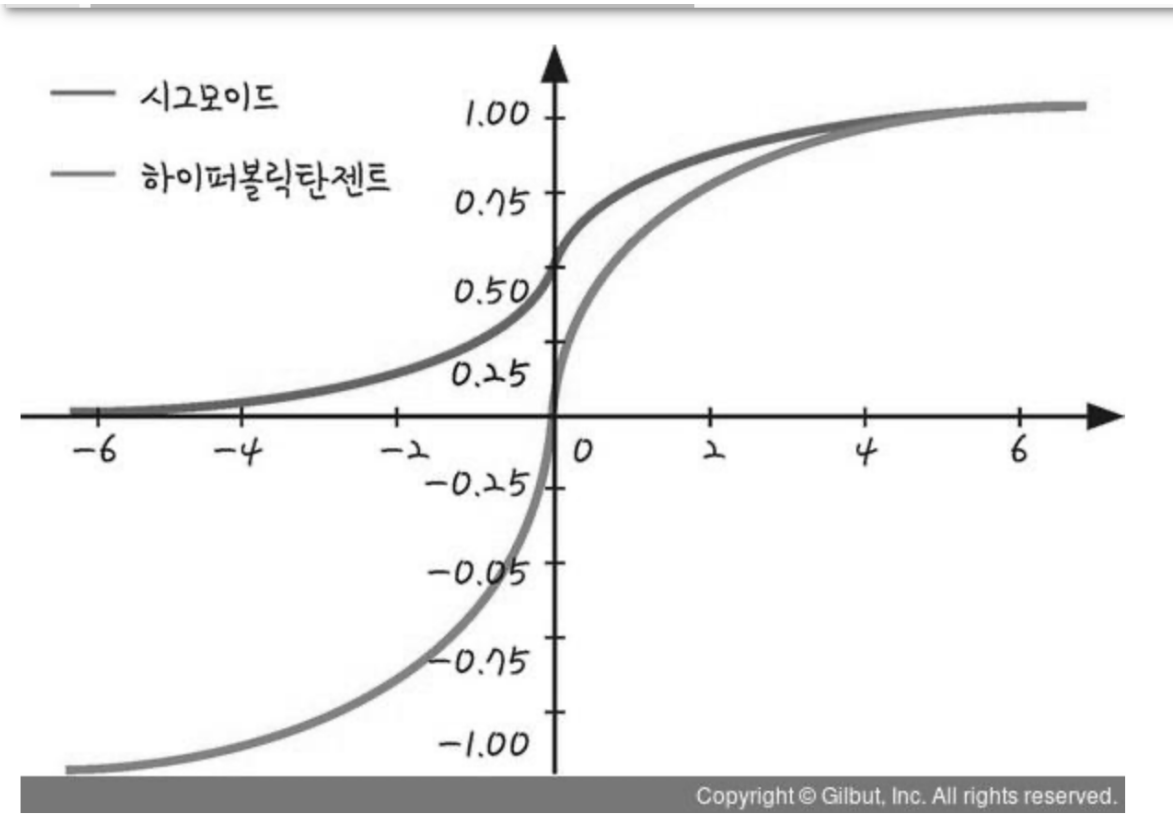
`dense_2 (Dense)` `(None, 512)` `131584`

`batch_normalization_2 (Batc hNormalization)` `(None, 512)` `2048`

`dense_3 (Dense)` `(None, 784)` `402192`

`=================================================================`

`Total params: 583,312`
`Trainable params: 581,520`
`Non-trainable params: 1,792`



판별자 Discriminator

```
discriminator=Sequential()

discriminator.add(Dense(128,input_shape=(784,),kernel_initializer=initializers.RandomNo
rmal(stddev=0.02)))
#kernel_initializer : 초기값 설정

discriminator.add(Dense(256,activation='relu'))
discriminator.add(Dense(512,activation='relu'))
discriminator.add(Dense(1,activation='sigmoid'))
```

discriminator.summary()

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 128)               100480

dense_5 (Dense)              (None, 256)               33024

dense_6 (Dense)              (None, 512)               131584

dense_7 (Dense)              (None, 1)                 513

```

```
=================================================================
Total params: 265,601
Trainable params: 265,601
Non-trainable params: 0
_________________________________________________________________
```

컴파일

```
discriminator.compile(optimizer=Adam(lr=0.0002), loss='binary_crossentropy', metrics=
['binary_accuracy'])
```

combined network

```
discriminator.trainable=False

g_input=Input(shape=(100,))
dis_output=discriminator(generator(g_input))

gan=Model(g_input,dis_output)
gan.compile(optimizer=Adam(lr=0.0002),loss='binary_crossentropy',metrics=['binary_accur
acy'])
```

Model: "model"

Layer (type) Output Shape Param

input_1 (InputLayer) [(None, 100)] 0

sequential (Sequential) (None, 784) 583312

sequential_1 (Sequential) (None, 1) 265601

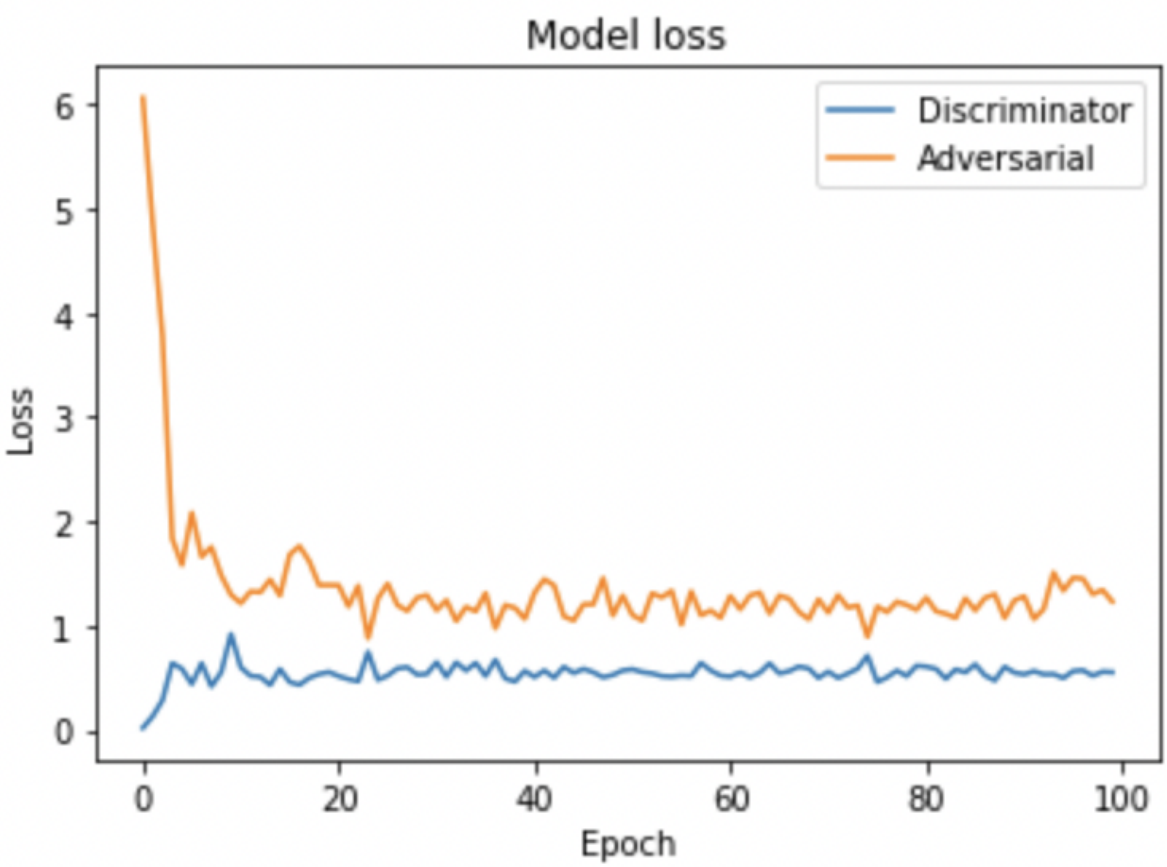
===

Total params: 848,913
Trainable params: 581,520
Non-trainable params: 267,393

```
d_loss = [] #0.5에 가까워질수록 성능이 좋음
g_loss = []

#전체적인 학습 구조로 이해하되 다양한 실험을 해보고 -> 수학적인 부분은 도구로 (아이디어가 중요하다)

for e in range(epochs):
    for i in range(len(X_train) // batch_size): #int(len(X_train) / batch_size))와 동일일
```

```
        discriminator.trainable = True

        #진짜 학습
        idx = np.random.randint(0, X_train.shape[0], batch_size) #shuffle기능과 동일
        imgs = X_train[idx]
        d_loss_real = discriminator.train_on_batch(imgs, real) #batch기준으로 학습

        #가짜학습
        noise = np.random.normal(loc=0, scale=1, size=(batch_size, latent_dim))
        fake_imgs = generator.predict_on_batch(noise)
        d_loss_fake = discriminator.train_on_batch(fake_imgs, fake)

        #batch중에 0번째꺼뽑겠다 for문 ==sigma역할과 동일하게 수행함
        d_loss_batch = 0.5 * (d_loss_real[0] + d_loss_fake[0]) #=> 평균

        #lock
        #GAN학습
        discriminator.trainable = False
        g_loss_batch = gan.train_on_batch(noise, real)

    d_loss.append(d_loss_batch)
    g_loss.append(g_loss_batch[0])
    print("epoch = %d/%d, d_loss=%.3f, g_loss=%.3f" % (e + 1, epochs, d_loss[-1], g_los
s[-1]))

    if e % 5 == 0:
        samples = 10
        fake_imgs = generator.predict(np.random.normal(loc=0, scale=1, size=(samples, l
atent_dim)))

        for k in range(samples):
            plt.subplot(2, 5, k + 1)
            plt.imshow(fake_imgs[k].reshape(28, 28), cmap="gray")
            plt.xticks([])
            plt.yticks([])
        plt.tight_layout()
        plt.show()
```

```
plt.plot(d_loss)
plt.plot(g_loss)

plt.title("Model loss")
plt.ylabel('Loss')
plt.xlabel('Epoch')

plt.legend(['Discriminator','Adversarial'])
plt.show()
```


Model loss
Discriminator
Adversarial
Loss
Epoch
0
1
2
3
4
5
6
0
20
40
60
80
100