GAN 실습 - 적대적 생성 신경망 ¶

라이브러리 불러오기

학습 내용

- 데이터 셋 : mnist 데이터 셋
- GAN 모델을 구성하고 이를 활용하여 MNIST 이미지를 생성해 보는 것을 실습을 통해 확인해 본다.

In [ ]:

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

In [ ]:

```
import keras
from keras.layers import Dense, Dropout, Input
from keras.models import Model,Sequential
from keras.datasets import mnist
from tqdm import tqdm
from keras.layers.advanced_activations import LeakyReLU
from keras.optimizers import Adam
```

데이터 준비

In [ ]:

```
def load_data():
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
    x_train = (x_train.astype(np.float32) - 127.5)/127.5

    print("데이터 값의 범위 : ", np.min(x_train), np.max(x_train) )
    # convert shape of x_train from (60000, 28, 28) to (60000, 784)
    # 784 columns per row
    x_train = x_train.reshape(60000, 784)
    return (x_train, y_train, x_test, y_test)
```

In [ ]:

```
(X_train, y_train,X_test, y_test)=load_data()
print(X_train.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mn
ist.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz)
11493376/11490434 [==============================] - 0s 0us/step
데이터 값의 범위 :  -1.0 1.0
(60000, 784)
```

In [ ]:

```
def adam_optimizer():
    return Adam(lr=0.0002, beta_1=0.5)
```

Generator 생성

- 생성 모델
- Noise를 이용해서 가짜 이미지를 만들어낸다.

In [ ]:

```
def create_generator():
    generator=Sequential()
    generator.add(Dense(units=256,input_dim=100))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(units=512))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(units=1024))
    generator.add(LeakyReLU(0.2))

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

    generator.compile(loss='binary_crossentropy', optimizer=adam_optimizer())
    return generator

g=create_generator()
g.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               25856
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               131584
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 784)               803600
=================================================================
Total params: 1,486,352
Trainable params: 1,486,352
Non-trainable params: 0
_________________________________________________________________
```

discriminator 생성

- 구분 모델
- 진짜 이미지와 가짜 이미지를 비교한다.
  - 진짜 이미지를 1에 가깝게 간다면 진짜 이미지라고 한다.

In [ ]:

```
def create_discriminator():
    discriminator=Sequential()
    discriminator.add(Dense(units=1024,input_dim=784))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(units=512))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(units=256))
    discriminator.add(LeakyReLU(0.2))

    discriminator.add(Dense(units=1, activation='sigmoid'))

    discriminator.compile(loss='binary_crossentropy', optimizer=adam_optimizer())
    return discriminator

d =create_discriminator()
d.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 1024)              803840
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               524800
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 257
=================================================================
Total params: 1,460,225
Trainable params: 1,460,225
Non-trainable params: 0
_________________________________________________________________
```

gan

In [ ]:

```
def create_gan(discriminator, generator):
    discriminator.trainable=False
    gan_input = Input(shape=(100,))
    x = generator(gan_input)
    gan_output= discriminator(x)
    gan= Model(inputs=gan_input, outputs=gan_output)    # 레이어를 객체로 그룹화
    gan.compile(loss='binary_crossentropy', optimizer='adam')
    return gan

# g = create_generator()     # 생성자
# d = create_discriminator()  # 판별자

gan = create_gan(d,g)
gan.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
sequential (Sequential)      (None, 784)               1486352
_________________________________________________________________
sequential_1 (Sequential)    (None, 1)                 1460225
=================================================================
Total params: 2,946,577
Trainable params: 1,486,352
Non-trainable params: 1,460,225
_________________________________________________________________
```

이미지 시각화

In [ ]:

```
def plot_generated_images(epoch, generator, examples=100, dim=(10,10), figsize=(10,10)):
    noise= np.random.normal(loc=0, scale=1, size=[examples, 100])
    generated_images = generator.predict(noise)
    generated_images = generated_images.reshape(100,28,28)
    plt.figure(figsize=figsize)
    for i in range(generated_images.shape[0]):
        plt.subplot(dim[0], dim[1], i+1)
        plt.imshow(generated_images[i], interpolation='nearest')
        plt.axis('off')
    plt.tight_layout()
    plt.savefig('gan_generated_image %d.png' %epoch)
```

GAN 학습

In [ ]:

```
def training(epochs=1, batch_size=128):

    # 데이터 불러오기(Loading the data)
    (X_train, y_train, X_test, y_test) = load_data()
    batch_count = X_train.shape[0] / batch_size

    # Creating GAN
    generator= create_generator()
    discriminator= create_discriminator()
    gan = create_gan(discriminator, generator)

    for e in range(1,epochs+1 ):
        print("Epoch %d" %e)
        for _ in tqdm(range(batch_size)):
        #generate  random noise as an input  to  initialize the  generator
            noise= np.random.normal(0,1, [batch_size, 100])

            # Noise를 이용하여 MNIST이미지 만들기( Generate fake MNIST images from noised input )
            generated_images = generator.predict(noise)

            # Get a random set of  real images
            image_batch =X_train[np.random.randint(low=0,high=X_train.shape[0],size=batch_size)]

            # Construct different batches of  real and fake data
            X= np.concatenate([image_batch, generated_images])

            # Labels for generated and real data
            y_dis=np.zeros(2*batch_size)
            y_dis[:batch_size]=0.9

            #Pre train discriminator on  fake and real data  before starting the gan.
            discriminator.trainable=True
            discriminator.train_on_batch(X, y_dis)

            #Tricking the noised input of the Generator as real data
            noise= np.random.normal(0,1, [batch_size, 100])
            y_gen = np.ones(batch_size)

            # During the training of gan,
            # the weights of discriminator should be fixed.
            #We can enforce that by setting the trainable flag
            discriminator.trainable=False

            #training  the GAN by alternating the training of the Discriminator
            #and training the chained GAN model with Discriminator's weights freezed.
            gan.train_on_batch(noise, y_gen)

        if e == 1 or e % 20 == 0:

            plot_generated_images(e, generator)
```

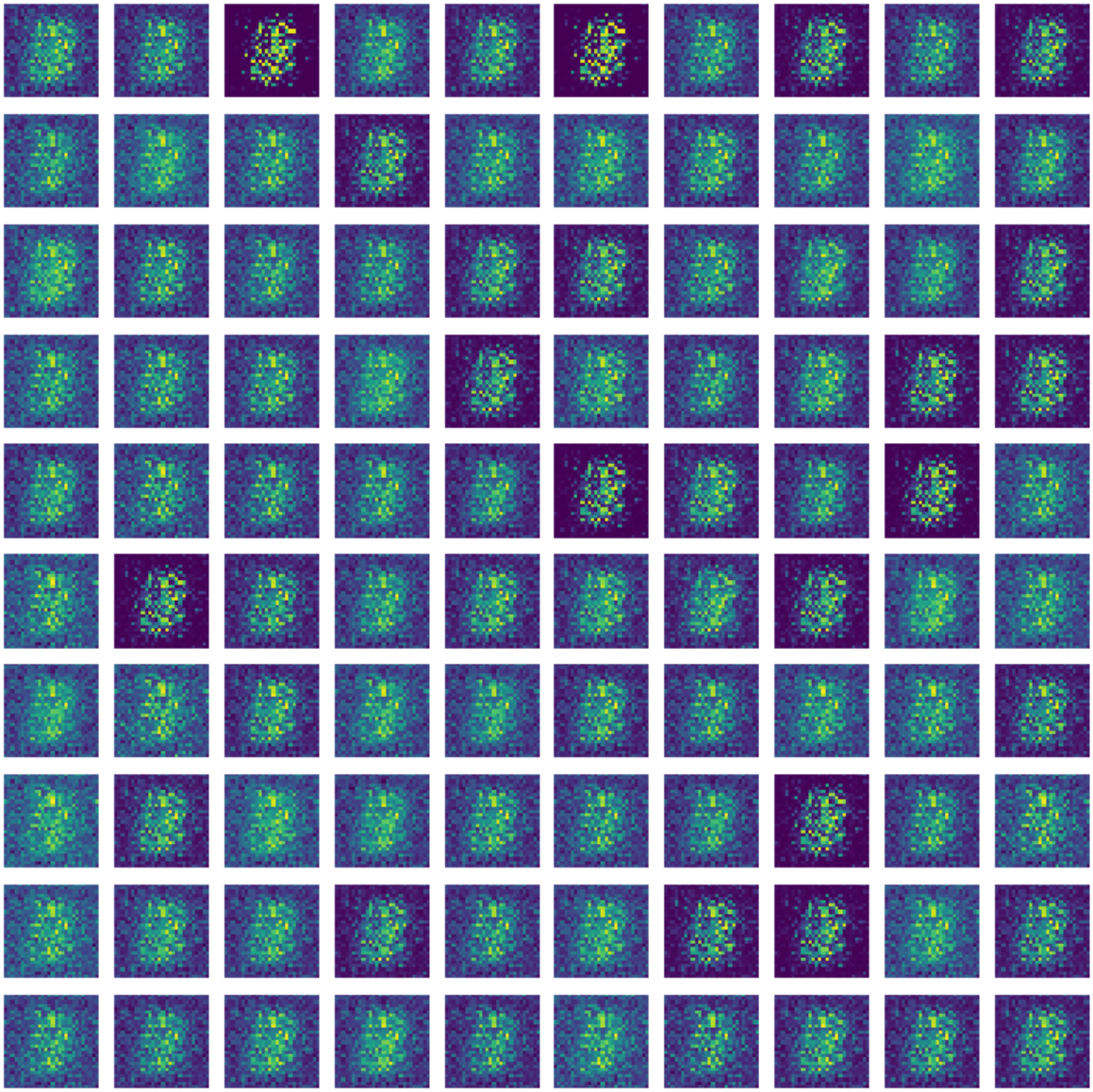
In [ ]:

```
%%time

training(200,256)
```

```
  0%|          | 0/256 [00:00<?, ?it/s]

데이터 값의 범위 :  -1.0 1.0
Epoch 1

100%|██████████| 256/256 [00:16<00:00, 15.56it/s]
  1%|          | 2/256 [00:00<00:14, 17.72it/s]

Epoch 2

100%|██████████| 256/256 [00:13<00:00, 18.67it/s]
  1%|          | 2/256 [00:00<00:13, 18.35it/s]

Epoch 3

100%|██████████| 256/256 [00:13<00:00, 18.37it/s]
  1%|          | 2/256 [00:00<00:14, 17.96it/s]
```

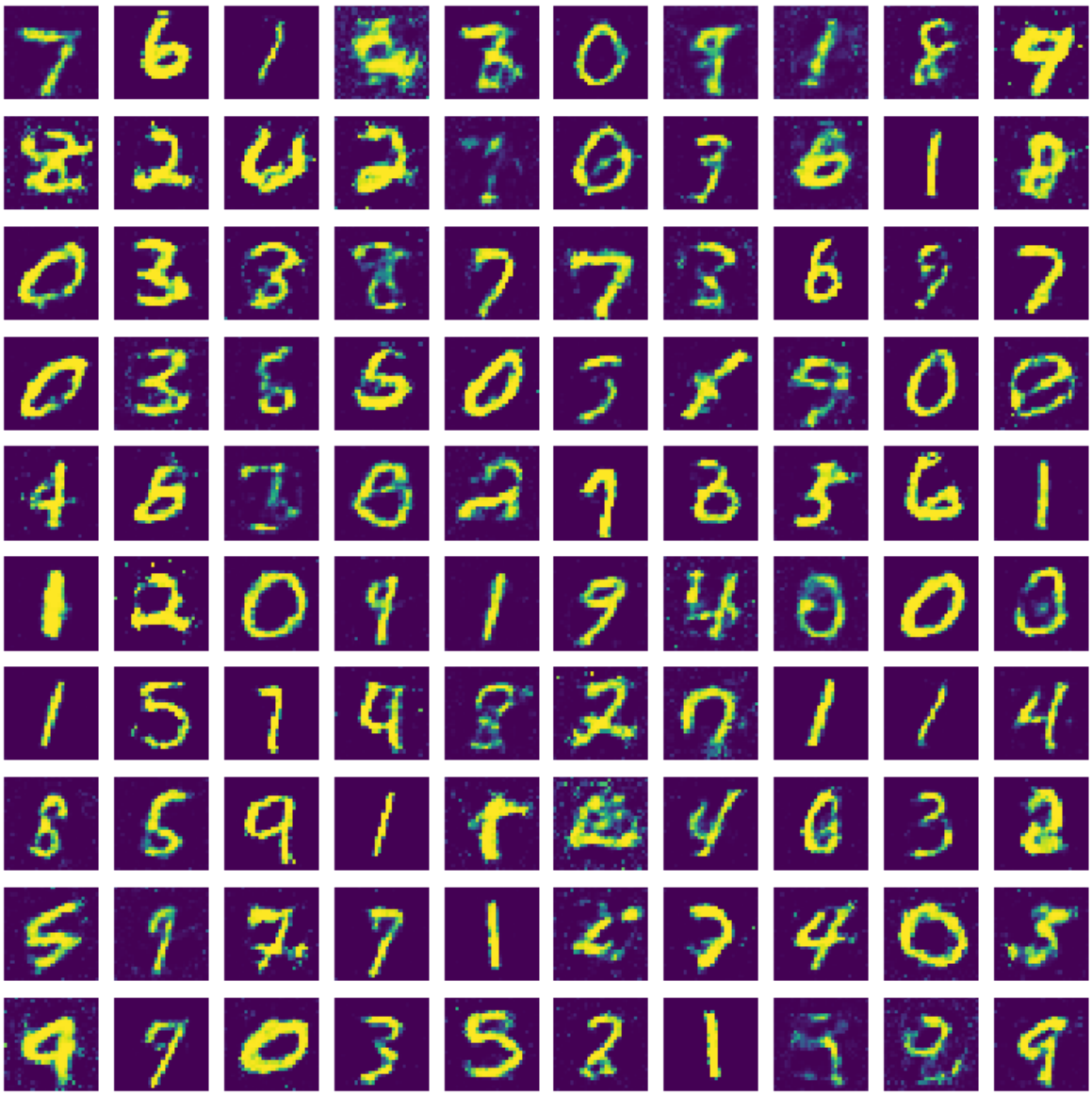
In [ ]:

```
from IPython.display import Image, display

display(Image("./gan_generated_image 1.png"))
```



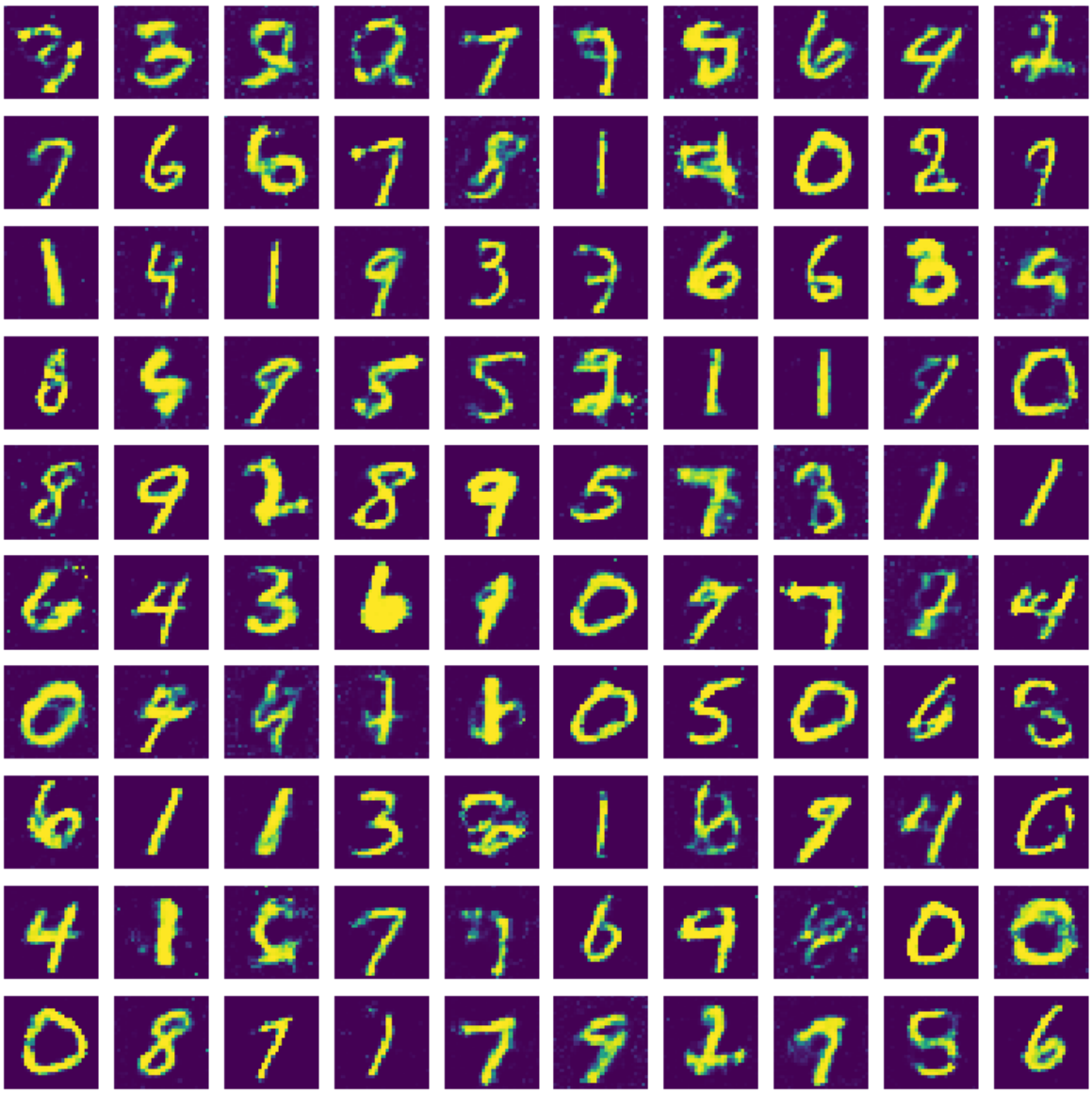
In [ ]:

```
display(Image("./gan_generated_image 60.png"))
```



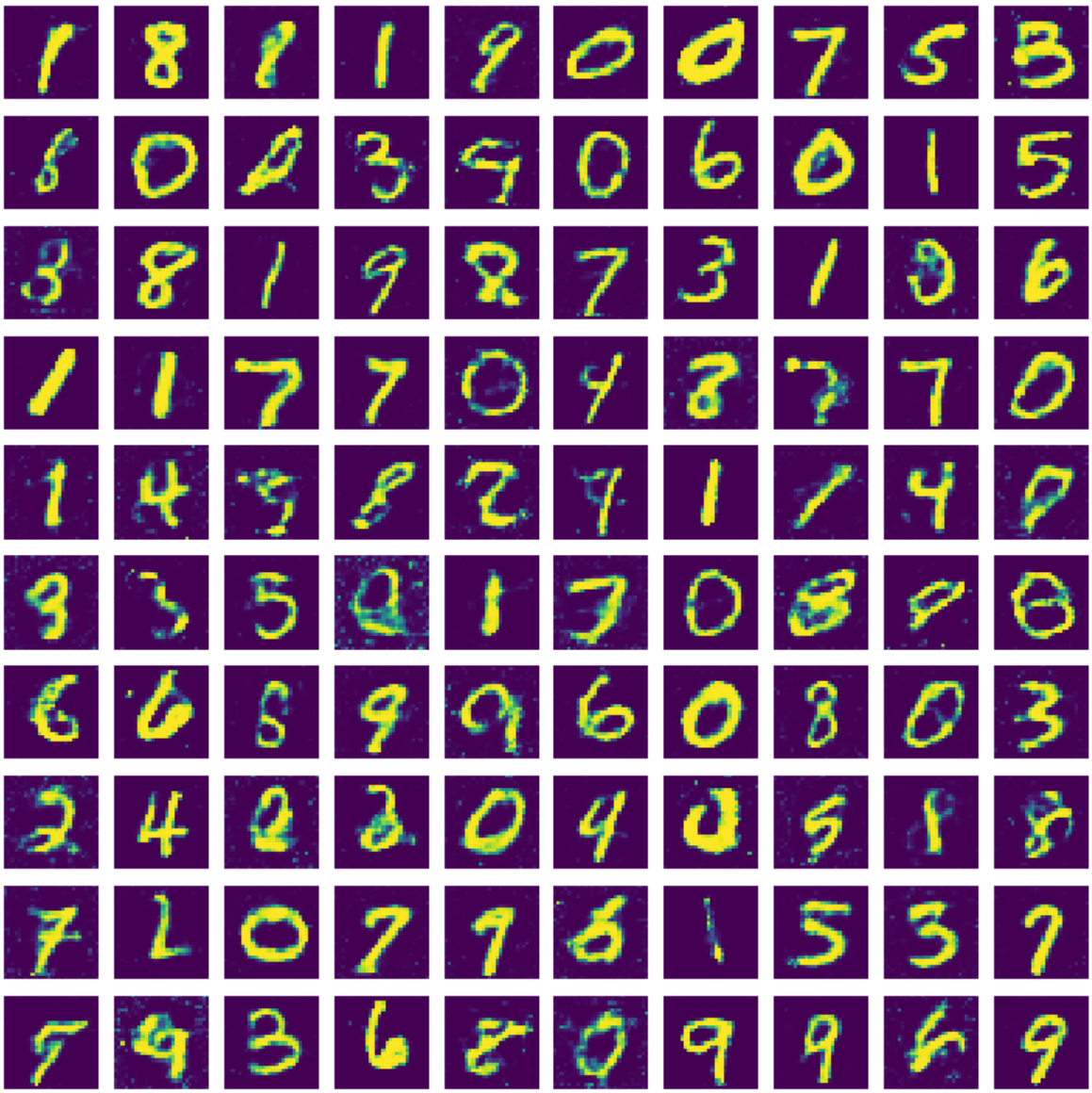
In [ ]:

```
display(Image("./gan_generated_image 100.png"))
```



In [ ]:

```
display(Image("./gan_generated_image 200.png"))
```



REF

- https://towardsdatascience.com/writing-your-first-generative-adversarial-network-with-keras-2d16fd8d4889 (https://towardsdatascience.com/writing-your-first-generative-adversarial-network-with-keras-2d16fd8d4889)
- https://medium.com/datadriveninvestor/generative-adversarial-network-gan-using-keras-ce1c05cfdfd3 (https://medium.com/datadriveninvestor/generative-adversarial-network-gan-using-keras-ce1c05cfdfd3)

In [ ]: