In [1]:

```
### 라이브러리 및 데이터 불러오기
# 필요한 라이브러리를 불러온다.
import torch
import torch.nn as nn
from torch.optim import Adam
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from torch.autograd import Variable
import pickle

# 데이터 전처리 방식을 지정한다.
transform = transforms.Compose([
        transforms.ToTensor(), # 데이터를 PyTorch의 Tensor 형식으로 바꾼다.
        transforms.Normalize(mean=(0.5,), std=(0.5,)) # 픽셀값 0 ~ 1 -> -1 ~ 1
])

# MNIST 데이터셋을 불러온다. 지정한 폴더에 없을 경우 자동으로 다운로드한다.
mnist = datasets.MNIST(root='data', download=True, transform=transform)

# 데이터를 한번에 batch_size만큼만 가져오는 dataloader를 만든다.
dataloader = DataLoader(mnist, batch_size=60, shuffle=True)
```

나중에 다시 0~1로 바꿔주는 작업필요.

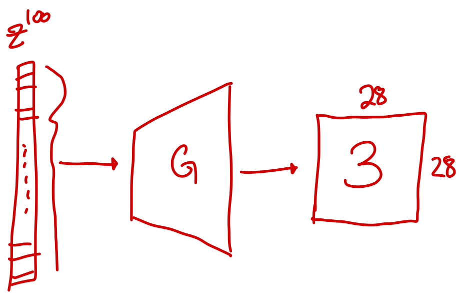
→ $\frac{img+1}{2}$

In [2]:

```
import os
import imageio

if torch.cuda.is_available():
    use_gpu = True
leave_log = True
if leave_log:
    result_dir = 'GAN_generated_images'
    if not os.path.isdir(result_dir):
        os.mkdir(result_dir)
```

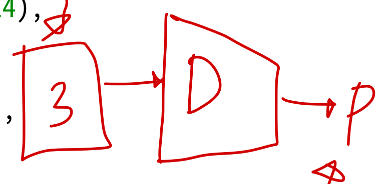
In [3]:

```python
### GAN의 생성자(Generator)
# 생성자는 랜덤 벡터 z를 입력으로 받아 가짜 이미지를 출력한다.
class Generator(nn.Module):

    # 네트워크 구조
    def __init__(self):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            nn.Linear(in_features=100, out_features=256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(in_features=256, out_features=512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(in_features=512, out_features=1024),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(in_features=1024, out_features=28*28),
            nn.Tanh())

    # (batch_size x 100) 크기의 랜덤 벡터를 받아
    # 이미지를 (batch_size x 1 x 28 x 28) 크기로 출력한다.
    def forward(self, inputs):
        return self.main(inputs).view(-1, 1, 28, 28)
```




In [4]:

```python
### GAN의 구분자(Discriminator)
# 구분자는 이미지를 입력으로 받아 이미지가 진짜인지 가짜인지 출력한다.
class Discriminator(nn.Module):

    # 네트워크 구조
    def __init__(self):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            nn.Linear(in_features=28*28, out_features=1024),
            nn.LeakyReLU(0.2, inplace=False),
            nn.Dropout(inplace=True),
            nn.Linear(in_features=1024, out_features=512),
            nn.LeakyReLU(0.2, inplace=False),
            nn.Dropout(inplace=True),
            nn.Linear(in_features=512, out_features=256),
            nn.LeakyReLU(0.2, inplace=False),
            nn.Dropout(inplace=True),
            nn.Linear(in_features=256, out_features=1),
            nn.Sigmoid())

    # (batch_size x 1 x 28 x 28) 크기의 이미지를 받아
    # 이미지가 진짜일 확률을 0~1 사이로 출력한다.
    def forward(self, inputs):
        inputs = inputs.view(-1, 28*28)
        return self.main(inputs)
```



In [5]:

```
### 생성자와 구분자 객체 만들기
G = Generator()
D = Discriminator()

if use_gpu:
    G.cuda()
    D.cuda()
```

In [6]:

```
### 손실 함수와 최적화 기법 지정하기
# Binary Cross Entropy loss
criterion = nn.BCELoss()

# 생성자의 매개 변수를 최적화하는 Adam optimizer
G_optimizer = Adam(G.parameters(), lr=0.0002, betas=(0.5, 0.999))
# 구분자의 매개 변수를 최적화하는 Adam optimizer
D_optimizer = Adam(D.parameters(), lr=0.0002, betas=(0.5, 0.999))
```

In [7]:

```
# 학습 결과 시각화하기
%matplotlib inline
from matplotlib import pyplot as plt
import numpy as np

def square_plot(data, path):
    """Take an array of shape (n, height, width) or (n, height, width , 3)
       and visualize each (height, width) thing in a grid of size approx. sqrt(n) by sqrt(n)"""

    if type(data) == list:
        data = np.concatenate(data)
    # normalize data for display
    data = (data - data.min()) / (data.max() - data.min())  -1~1 → 0~1

    # force the number of filters to be square
    n = int(np.ceil(np.sqrt(data.shape[0])))

    padding = (((0, n ** 2 - data.shape[0]) ,
                (0, 1), (0, 1))  # add some space between filters
               + ((0, 0),) * (data.ndim - 3))  # don't pad the last dimension (if there is one)
    data = np.pad(data , padding, mode='constant' , constant_values=1)  # pad with ones (white)

    # tilethe filters into an image
    data = data.reshape((n , n) + data.shape[1:]).transpose((0 , 2 , 1 , 3) + tuple(range(4 , d

    data = data.reshape((n * data.shape[1] , n * data.shape[3]) + data.shape[4:])

    plt.imsave(path, data, cmap='gray')
```

In [8]:

```
if leave_log:
    train_hist = {}
    train_hist['D_losses'] = []
    train_hist['G_losses'] = []
    generated_images = []

z_fixed = Variable(torch.randn(5 * 5, 100), volatile=True)
if use_gpu:
    z_fixed = z_fixed.cuda()
```

```
<ipython-input-8-8b401d4d980b>:7: UserWarning: volatile was removed and now has no e
ffect. Use `with torch.no_grad():` instead.
  z_fixed = Variable(torch.randn(5 * 5, 100), volatile=True)
```

In [13]:

```python
### 모델 학습을 위한 반복문
# 데이터셋을 100번 돌며 학습한다.
for epoch in range(100):

    if leave_log:
        D_losses = []
        G_losses = []

    # 한번에 batch_size만큼 데이터를 가져온다.
    for real_data, _ in dataloader:
        batch_size = real_data.size(0)

        # 데이터를 pytorch의 변수로 변환한다.
        real_data = Variable(real_data)

        ### 구분자 학습시키기

        # 이미지가 진짜일 때 정답 값은 1이고 가짜일 때는 0이다.
        # 정답지에 해당하는 변수를 만든다.
        target_real = Variable(torch.ones(batch_size, 1))
        target_fake = Variable(torch.zeros(batch_size, 1))
```

batch_size

| 1 | 1 | ... | 1 | 1 |

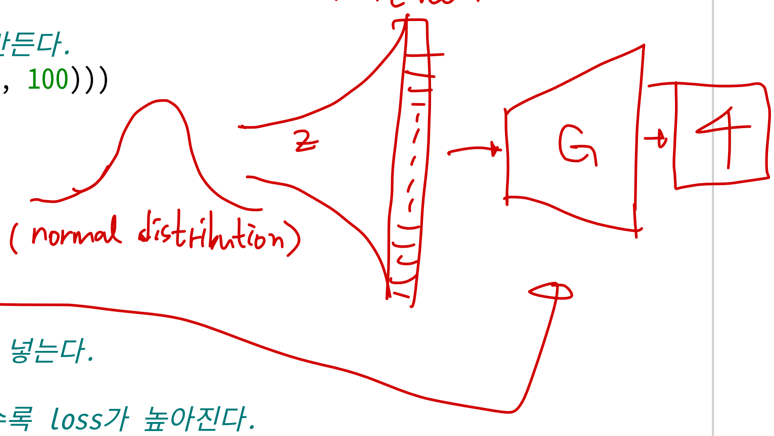
| 0 | 0 | ... | 0 | 0 |

```python
        if use_gpu:
            real_data, target_real, target_fake = real_data.cuda(), target_real.cuda(), target_

        # 진짜 이미지를 구분자에 넣는다.
        D_result_from_real = D(real_data)
        # 구분자의 출력값이 정답지인 1에서 멀수록 loss가 높아진다.
        D_loss_real = criterion(D_result_from_real, target_real)
```

D학습 1

100차원 vector

```python
        # 생성자에 입력으로 줄 랜덤 벡터 z를 만든다.
        z = Variable(torch.randn((batch_size, 100)))

        if use_gpu:
            z = z.cuda()

        # 생성자로 가짜 이미지를 생성한다.
        fake_data = G(z)
```




```python
        # 생성자가 만든 가짜 이미지를 구분자에 넣는다.
        D_result_from_fake = D(fake_data)
        # 구분자의 출력값이 정답지인 0에서 멀수록 loss가 높아진다.
        D_loss_fake = criterion(D_result_from_fake, target_fake)
```

D학습2 0

```python
        # 구분자의 loss는 두 문제에서 계산된 loss의 합이다.
        D_loss = D_loss_real + D_loss_fake
```

D학습1 D학습2

```python
        # 구분자의 매개 변수의 미분값을 0으로 초기화한다.
        D.zero_grad()
        # 역전파를 통해 매개 변수의 loss에 대한 미분값을 계산한다.
        D_loss.backward()
        # 최적화 기법을 이용해 구분자의 매개 변수를 업데이트한다.
        D_optimizer.step()

        if leave_log:
            D_losses.append(D_loss.data.item())

        # train generator G

```

```
        ### 생성자 학습시키기

        # 생성자에 입력으로 줄 랜덤 벡터 z를 만든다.
        z = Variable(torch.randn((batch_size, 100)))

        if use_gpu:
            z = z.cuda()

        # 생성자로 가짜 이미지를 생성한다. 가짜이미지 한번더 생성
        fake_data = G(z)
        # 생성자가 만든 가짜 이미지를 구분자에 넣는다.
        D_result_from_fake = D(fake_data)
        # 생성자의 입장에서 구분자의 출력값이 1에서 멀수록 loss가 높아진다.
        G_loss = criterion(D_result_from_fake, target_real)

        # 생성자의 매개 변수의 미분값을 0으로 초기화한다.
        G.zero_grad()
        # 역전파를 통해 매개 변수의 loss에 대한 미분값을 계산한다.
        G_loss.backward()
        # 최적화 기법을 이용해 생성자의 매개 변수를 업데이트한다.
        G_optimizer.step()

        if leave_log:
            G_losses.append(G_loss.data.item())
    if leave_log:
        true_positive_rate = (D_result_from_real > 0.5).float().mean().data.item()
        true_negative_rate = (D_result_from_fake < 0.5).float().mean().data.item()
        base_message = ("Epoch: {epoch:<3d} D Loss: {d_loss:<8.6} G Loss: {g_loss:<8.6} "
                        "True Positive Rate: {tpr:<5.1%} True Negative Rate: {tnr:<5.1%}"
                       )
        message = base_message.format(
                    epoch=epoch,
                    d_loss=sum(D_losses)/len(D_losses),
                    g_loss=sum(G_losses)/len(G_losses),
                    tpr=true_positive_rate,
                    tnr=true_negative_rate
        )
        print(message)

    if leave_log:
        fake_data_fixed = G(z_fixed)
        image_path = result_dir + '/epoch{}.png'.format(epoch)
        square_plot(fake_data_fixed.view(25, 28, 28).cpu().data.numpy(), path=image_path)
        generated_images.append(image_path)

    if leave_log:
        train_hist['D_losses'].append(torch.mean(torch.FloatTensor(D_losses)))
        train_hist['G_losses'].append(torch.mean(torch.FloatTensor(G_losses)))

torch.save(G.state_dict(), "gan_generator.pkl")
torch.save(D.state_dict(), "gan_discriminator.pkl")
with open('gan_train_history.pkl', 'wb') as f:
    pickle.dump(train_hist, f)

generated_image_array = [imageio.imread(generated_image) for generated_image in generated_image
imageio.mimsave(result_dir + '/GAN_generation.gif', generated_image_array, fps=5)
```

```
Epoch: 0    D Loss: 0.675597 G Loss: 2.4401   True Positive Rate: 86.7% True Negati
ve Rate: 100.0%
```

```
Epoch: 1   D Loss: 0.62179  G Loss: 2.40634  True Positive Rate: 81.7% True Negati
ve Rate: 98.3%
Epoch: 2   D Loss: 0.756454 G Loss: 1.994    True Positive Rate: 71.7% True Negati
ve Rate: 93.3%
Epoch: 3   D Loss: 0.901167 G Loss: 1.6013   True Positive Rate: 61.7% True Negati
ve Rate: 68.3%
Epoch: 4   D Loss: 0.966424 G Loss: 1.40884  True Positive Rate: 70.0% True Negati
ve Rate: 80.0%
Epoch: 5   D Loss: 1.06734  G Loss: 1.21014  True Positive Rate: 78.3% True Negati
ve Rate: 83.3%
Epoch: 6   D Loss: 1.10436  G Loss: 1.14163  True Positive Rate: 63.3% True Negati
ve Rate: 75.0%
Epoch: 7   D Loss: 1.12438  G Loss: 1.10729  True Positive Rate: 73.3% True Negati
ve Rate: 68.3%
Epoch: 8   D Loss: 1.1447   G Loss: 1.08023  True Positive Rate: 63.3% True Negati
ve Rate: 90.0%
Epoch: 9   D Loss: 1.16998  G Loss: 1.04394  True Positive Rate: 61.7% True Negati
ve Rate: 76.7%
Epoch: 10  D Loss: 1.18633  G Loss: 1.01089  True Positive Rate: 51.7% True Negati
ve Rate: 81.7%
Epoch: 11  D Loss: 1.20961  G Loss: 0.973225 True Positive Rate: 61.7% True Negati
ve Rate: 80.0%
Epoch: 12  D Loss: 1.2198   G Loss: 0.962458 True Positive Rate: 56.7% True Negati
ve Rate: 76.7%
Epoch: 13  D Loss: 1.23816  G Loss: 0.92789  True Positive Rate: 46.7% True Negati
ve Rate: 73.3%
Epoch: 14  D Loss: 1.24362  G Loss: 0.924061 True Positive Rate: 50.0% True Negati
ve Rate: 78.3%
Epoch: 15  D Loss: 1.2499   G Loss: 0.912016 True Positive Rate: 40.0% True Negati
ve Rate: 70.0%
Epoch: 16  D Loss: 1.26288  G Loss: 0.892861 True Positive Rate: 56.7% True Negati
ve Rate: 70.0%
Epoch: 17  D Loss: 1.26077  G Loss: 0.893253 True Positive Rate: 51.7% True Negati
ve Rate: 80.0%
Epoch: 18  D Loss: 1.26579  G Loss: 0.888503 True Positive Rate: 58.3% True Negati
ve Rate: 61.7%
Epoch: 19  D Loss: 1.27449  G Loss: 0.878842 True Positive Rate: 73.3% True Negati
ve Rate: 66.7%
Epoch: 20  D Loss: 1.27637  G Loss: 0.874628 True Positive Rate: 61.7% True Negati
ve Rate: 83.3%
Epoch: 21  D Loss: 1.27423  G Loss: 0.876046 True Positive Rate: 48.3% True Negati
ve Rate: 56.7%
Epoch: 22  D Loss: 1.27934  G Loss: 0.866685 True Positive Rate: 60.0% True Negati
ve Rate: 60.0%
Epoch: 23  D Loss: 1.27966  G Loss: 0.865575 True Positive Rate: 53.3% True Negati
ve Rate: 83.3%
Epoch: 24  D Loss: 1.2798   G Loss: 0.864793 True Positive Rate: 45.0% True Negati
ve Rate: 71.7%
Epoch: 25  D Loss: 1.27903  G Loss: 0.868448 True Positive Rate: 56.7% True Negati
ve Rate: 63.3%
Epoch: 26  D Loss: 1.28216  G Loss: 0.86383  True Positive Rate: 61.7% True Negati
ve Rate: 66.7%
Epoch: 27  D Loss: 1.28537  G Loss: 0.857613 True Positive Rate: 45.0% True Negati
ve Rate: 66.7%
Epoch: 28  D Loss: 1.28281  G Loss: 0.862414 True Positive Rate: 46.7% True Negati
ve Rate: 71.7%
Epoch: 29  D Loss: 1.28035  G Loss: 0.864972 True Positive Rate: 55.0% True Negati
ve Rate: 71.7%
Epoch: 30  D Loss: 1.27817  G Loss: 0.870954 True Positive Rate: 55.0% True Negati
ve Rate: 65.0%
Epoch: 31  D Loss: 1.27735  G Loss: 0.870821 True Positive Rate: 58.3% True Negati
```

```
ve Rate: 85.0%
Epoch: 32  D Loss: 1.27742  G Loss: 0.870563 True Positive Rate: 56.7% True Negati
ve Rate: 65.0%
Epoch: 33  D Loss: 1.28021  G Loss: 0.867131 True Positive Rate: 73.3% True Negati
ve Rate: 75.0%
Epoch: 34  D Loss: 1.27597  G Loss: 0.874014 True Positive Rate: 61.7% True Negati
ve Rate: 70.0%
Epoch: 35  D Loss: 1.27769  G Loss: 0.870908 True Positive Rate: 63.3% True Negati
ve Rate: 66.7%
Epoch: 36  D Loss: 1.27822  G Loss: 0.872324 True Positive Rate: 53.3% True Negati
ve Rate: 61.7%
Epoch: 37  D Loss: 1.27767  G Loss: 0.86911  True Positive Rate: 68.3% True Negati
ve Rate: 70.0%
Epoch: 38  D Loss: 1.27412  G Loss: 0.87391  True Positive Rate: 73.3% True Negati
ve Rate: 60.0%
Epoch: 39  D Loss: 1.27743  G Loss: 0.869919 True Positive Rate: 53.3% True Negati
ve Rate: 68.3%
Epoch: 40  D Loss: 1.2753   G Loss: 0.878193 True Positive Rate: 48.3% True Negati
ve Rate: 65.0%
Epoch: 41  D Loss: 1.27803  G Loss: 0.86842  True Positive Rate: 56.7% True Negati
ve Rate: 65.0%
Epoch: 42  D Loss: 1.28235  G Loss: 0.87137  True Positive Rate: 58.3% True Negati
ve Rate: 71.7%
Epoch: 43  D Loss: 1.27834  G Loss: 0.872655 True Positive Rate: 56.7% True Negati
ve Rate: 81.7%
Epoch: 44  D Loss: 1.2806   G Loss: 0.869648 True Positive Rate: 53.3% True Negati
ve Rate: 68.3%
Epoch: 45  D Loss: 1.27753  G Loss: 0.871985 True Positive Rate: 51.7% True Negati
ve Rate: 46.7%
Epoch: 46  D Loss: 1.27931  G Loss: 0.868883 True Positive Rate: 68.3% True Negati
ve Rate: 63.3%
Epoch: 47  D Loss: 1.28316  G Loss: 0.866255 True Positive Rate: 51.7% True Negati
ve Rate: 71.7%
Epoch: 48  D Loss: 1.27679  G Loss: 0.871868 True Positive Rate: 73.3% True Negati
ve Rate: 61.7%
Epoch: 49  D Loss: 1.28183  G Loss: 0.866442 True Positive Rate: 65.0% True Negati
ve Rate: 68.3%
Epoch: 50  D Loss: 1.27979  G Loss: 0.870687 True Positive Rate: 51.7% True Negati
ve Rate: 70.0%
Epoch: 51  D Loss: 1.27697  G Loss: 0.873522 True Positive Rate: 48.3% True Negati
ve Rate: 63.3%
Epoch: 52  D Loss: 1.28354  G Loss: 0.867277 True Positive Rate: 63.3% True Negati
ve Rate: 85.0%
Epoch: 53  D Loss: 1.27663  G Loss: 0.876767 True Positive Rate: 61.7% True Negati
ve Rate: 76.7%
Epoch: 54  D Loss: 1.27448  G Loss: 0.877351 True Positive Rate: 61.7% True Negati
ve Rate: 65.0%
Epoch: 55  D Loss: 1.28     G Loss: 0.869467 True Positive Rate: 60.0% True Negati
ve Rate: 68.3%
Epoch: 56  D Loss: 1.27695  G Loss: 0.871021 True Positive Rate: 75.0% True Negati
ve Rate: 68.3%
Epoch: 57  D Loss: 1.28288  G Loss: 0.862851 True Positive Rate: 61.7% True Negati
ve Rate: 78.3%
Epoch: 58  D Loss: 1.27752  G Loss: 0.872078 True Positive Rate: 65.0% True Negati
ve Rate: 61.7%
Epoch: 59  D Loss: 1.28368  G Loss: 0.86175  True Positive Rate: 45.0% True Negati
ve Rate: 73.3%
Epoch: 60  D Loss: 1.28211  G Loss: 0.867707 True Positive Rate: 53.3% True Negati
ve Rate: 83.3%
Epoch: 61  D Loss: 1.28513  G Loss: 0.860972 True Positive Rate: 66.7% True Negati
ve Rate: 80.0%
```

```
Epoch: 62  D Loss: 1.2834   G Loss: 0.862185 True Positive Rate: 55.0% True Negati
ve Rate: 60.0%
Epoch: 63  D Loss: 1.27846  G Loss: 0.868512 True Positive Rate: 50.0% True Negati
ve Rate: 71.7%
Epoch: 64  D Loss: 1.28168  G Loss: 0.863583 True Positive Rate: 51.7% True Negati
ve Rate: 86.7%
Epoch: 65  D Loss: 1.2817   G Loss: 0.861713 True Positive Rate: 48.3% True Negati
ve Rate: 66.7%
Epoch: 66  D Loss: 1.2831   G Loss: 0.864848 True Positive Rate: 55.0% True Negati
ve Rate: 70.0%
Epoch: 67  D Loss: 1.2829   G Loss: 0.861824 True Positive Rate: 63.3% True Negati
ve Rate: 65.0%
Epoch: 68  D Loss: 1.28819  G Loss: 0.855423 True Positive Rate: 48.3% True Negati
ve Rate: 50.0%
Epoch: 69  D Loss: 1.28134  G Loss: 0.863536 True Positive Rate: 43.3% True Negati
ve Rate: 65.0%
Epoch: 70  D Loss: 1.28313  G Loss: 0.860871 True Positive Rate: 68.3% True Negati
ve Rate: 60.0%
Epoch: 71  D Loss: 1.28711  G Loss: 0.857076 True Positive Rate: 55.0% True Negati
ve Rate: 70.0%
Epoch: 72  D Loss: 1.28596  G Loss: 0.858733 True Positive Rate: 53.3% True Negati
ve Rate: 66.7%
Epoch: 73  D Loss: 1.28542  G Loss: 0.855485 True Positive Rate: 53.3% True Negati
ve Rate: 66.7%
Epoch: 74  D Loss: 1.28934  G Loss: 0.853749 True Positive Rate: 58.3% True Negati
ve Rate: 58.3%
Epoch: 75  D Loss: 1.29019  G Loss: 0.852499 True Positive Rate: 70.0% True Negati
ve Rate: 70.0%
Epoch: 76  D Loss: 1.28586  G Loss: 0.854936 True Positive Rate: 60.0% True Negati
ve Rate: 71.7%
Epoch: 77  D Loss: 1.28598  G Loss: 0.856553 True Positive Rate: 43.3% True Negati
ve Rate: 81.7%
Epoch: 78  D Loss: 1.29109  G Loss: 0.85129  True Positive Rate: 51.7% True Negati
ve Rate: 75.0%
Epoch: 79  D Loss: 1.2907   G Loss: 0.848362 True Positive Rate: 46.7% True Negati
ve Rate: 71.7%
Epoch: 80  D Loss: 1.29118  G Loss: 0.85118  True Positive Rate: 55.0% True Negati
ve Rate: 78.3%
Epoch: 81  D Loss: 1.28922  G Loss: 0.850126 True Positive Rate: 58.3% True Negati
ve Rate: 75.0%
Epoch: 82  D Loss: 1.28609  G Loss: 0.855902 True Positive Rate: 65.0% True Negati
ve Rate: 70.0%
Epoch: 83  D Loss: 1.28652  G Loss: 0.85632  True Positive Rate: 56.7% True Negati
ve Rate: 60.0%
Epoch: 84  D Loss: 1.29146  G Loss: 0.850562 True Positive Rate: 51.7% True Negati
ve Rate: 76.7%

Epoch: 85  D Loss: 1.294    G Loss: 0.844387 True Positive Rate: 65.0% True Negati
ve Rate: 78.3%
Epoch: 86  D Loss: 1.29125  G Loss: 0.849909 True Positive Rate: 51.7% True Negati
ve Rate: 68.3%
Epoch: 87  D Loss: 1.29092  G Loss: 0.847983 True Positive Rate: 45.0% True Negati
ve Rate: 73.3%
Epoch: 88  D Loss: 1.29201  G Loss: 0.848245 True Positive Rate: 40.0% True Negati
ve Rate: 68.3%
Epoch: 89  D Loss: 1.29433  G Loss: 0.843447 True Positive Rate: 53.3% True Negati
ve Rate: 76.7%
Epoch: 90  D Loss: 1.29687  G Loss: 0.84468  True Positive Rate: 63.3% True Negati
ve Rate: 65.0%
Epoch: 91  D Loss: 1.29631  G Loss: 0.840685 True Positive Rate: 63.3% True Negati
ve Rate: 65.0%
```

```
Epoch: 92  D Loss: 1.29476  G Loss: 0.841257 True Positive Rate: 51.7% True Negati
ve Rate: 83.3%
Epoch: 93  D Loss: 1.29734  G Loss: 0.841508 True Positive Rate: 48.3% True Negati
ve Rate: 76.7%
Epoch: 94  D Loss: 1.29597  G Loss: 0.840039 True Positive Rate: 58.3% True Negati
ve Rate: 81.7%
Epoch: 95  D Loss: 1.29261  G Loss: 0.845269 True Positive Rate: 71.7% True Negati
ve Rate: 65.0%
Epoch: 96  D Loss: 1.29588  G Loss: 0.841494 True Positive Rate: 63.3% True Negati
ve Rate: 71.7%
Epoch: 97  D Loss: 1.29863  G Loss: 0.839284 True Positive Rate: 65.0% True Negati
ve Rate: 66.7%
Epoch: 98  D Loss: 1.29274  G Loss: 0.845982 True Positive Rate: 58.3% True Negati
ve Rate: 56.7%
Epoch: 99  D Loss: 1.29441  G Loss: 0.84067  True Positive Rate: 58.3% True Negati
ve Rate: 66.7%
```

In [78]:

```python
from IPython.display import Image
import cv2
import numpy as np
```

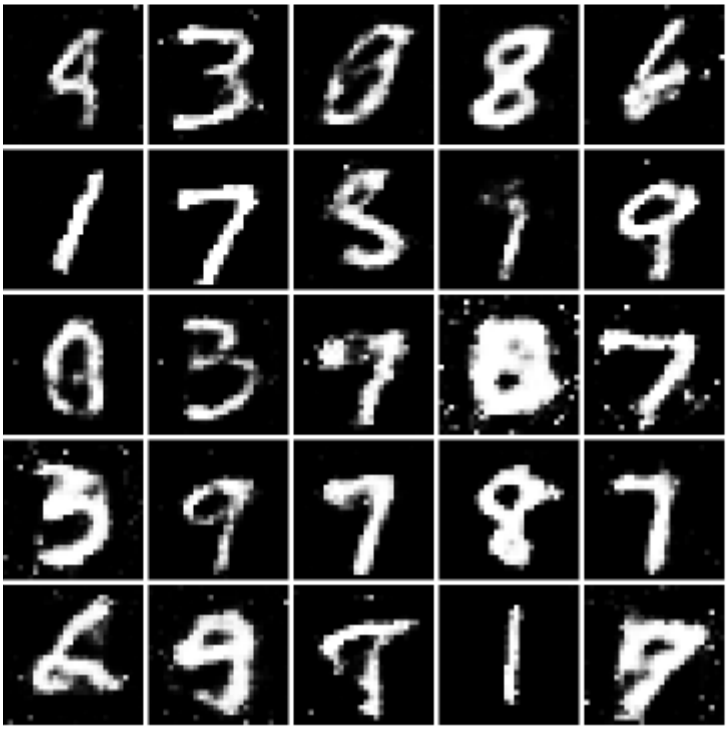
Out[78]:

True

In [83]:

```python
img = cv2.imread('./GAN_generated_images/epoch49.png')
img = cv2.resize(img, None, fx=2.5, fy=2.5, interpolation=cv2.INTER_AREA)
cv2.imwrite('./GAN_generated_images/epoch49.png',img)
Image(filename='./GAN_generated_images/epoch49.png')
```

Out[83]:

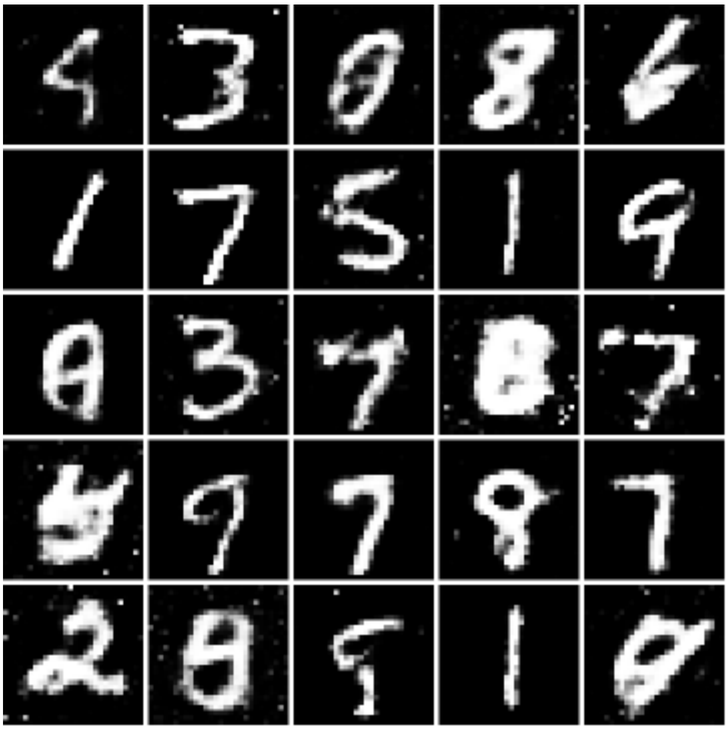


49

In [82]:

```
img = cv2.imread('./GAN_generated_images/epoch59.png')
img = cv2.resize(img, None, fx=2.5, fy=2.5, interpolation=cv2.INTER_AREA)
cv2.imwrite('./GAN_generated_images/epoch59.png',img)
Image(filename='./GAN_generated_images/epoch59.png')
```

Out[82]:

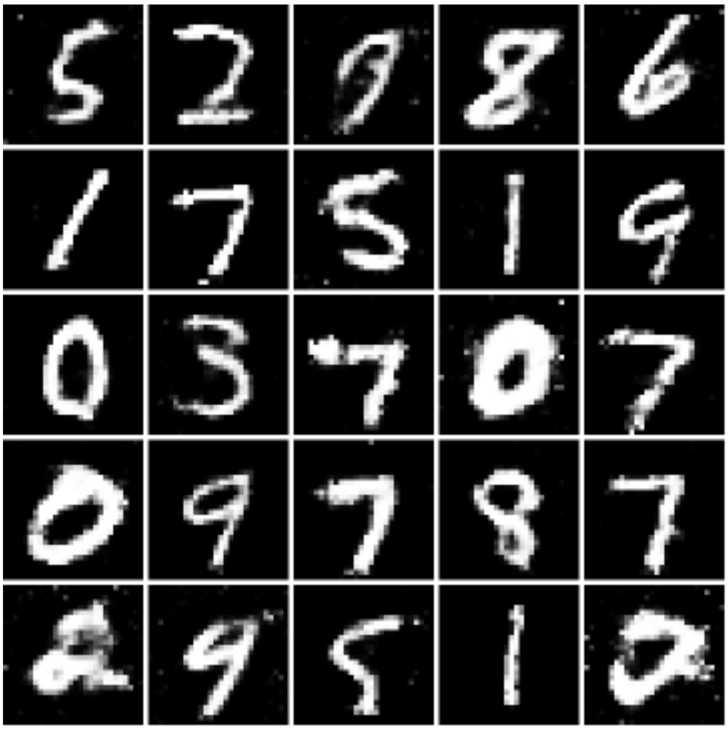


59

In [80]:

```
img = cv2.imread('./GAN_generated_images/epoch73.png')
img = cv2.resize(img, None, fx=2.5, fy=2.5, interpolation=cv2.INTER_AREA)
cv2.imwrite('./GAN_generated_images/epoch73.png',img)
Image(filename='./GAN_generated_images/epoch73.png')
```

Out[80]:

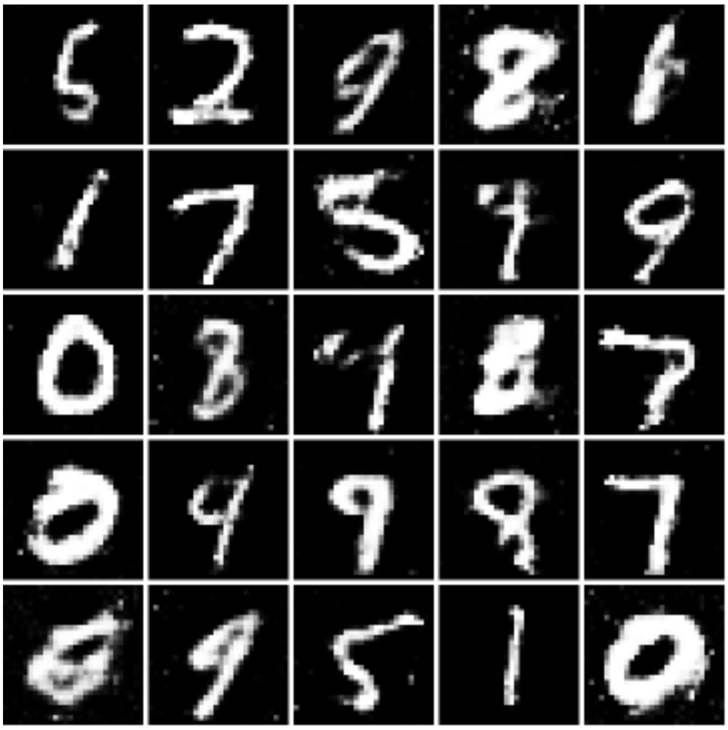


73

In [84]:

```
img = cv2.imread('./GAN_generated_images/epoch89.png')
img = cv2.resize(img, None, fx=2.5, fy=2.5, interpolation=cv2.INTER_AREA)
cv2.imwrite('./GAN_generated_images/epoch89.png',img)
Image(filename='./GAN_generated_images/epoch89.png')
```

Out[84]:

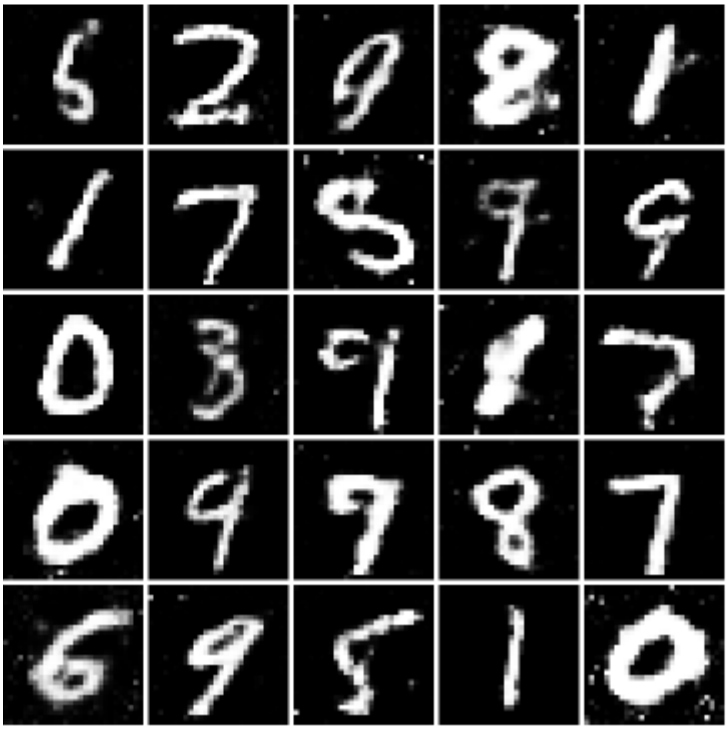


89

In [81]:

```
img = cv2.imread('./GAN_generated_images/epoch93.png')
img = cv2.resize(img, None, fx=2.5, fy=2.5, interpolation=cv2.INTER_AREA)
cv2.imwrite('./GAN_generated_images/epoch93.png',img)
Image(filename='./GAN_generated_images/epoch93.png')
```

Out[81]:

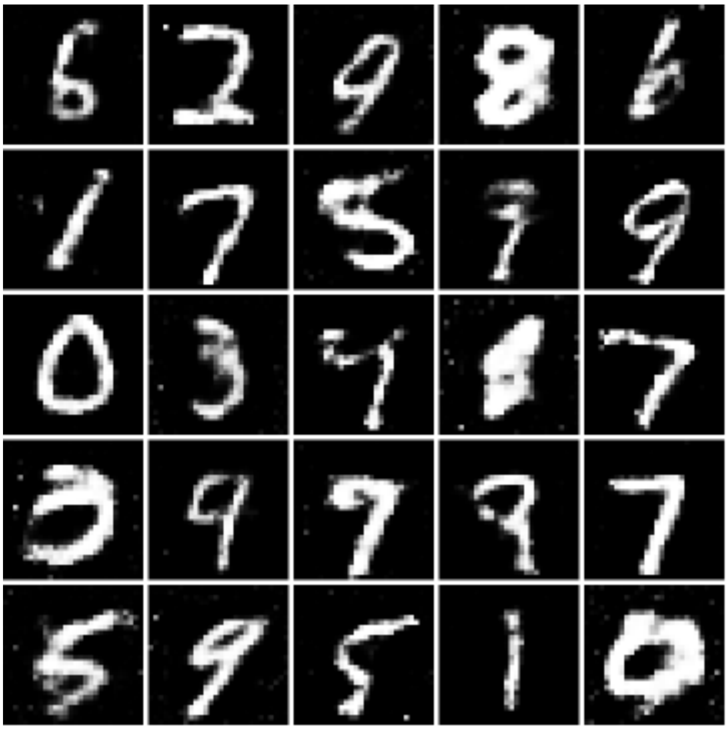


93

In [85]:

```
img = cv2.imread('./GAN_generated_images/epoch99.png')
img = cv2.resize(img, None, fx=2.5, fy=2.5, interpolation=cv2.INTER_AREA)
cv2.imwrite('./GAN_generated_images/epoch99.png',img)
Image(filename='./GAN_generated_images/epoch99.png')
```

Out[85]:



99

In [ ]:

```

```