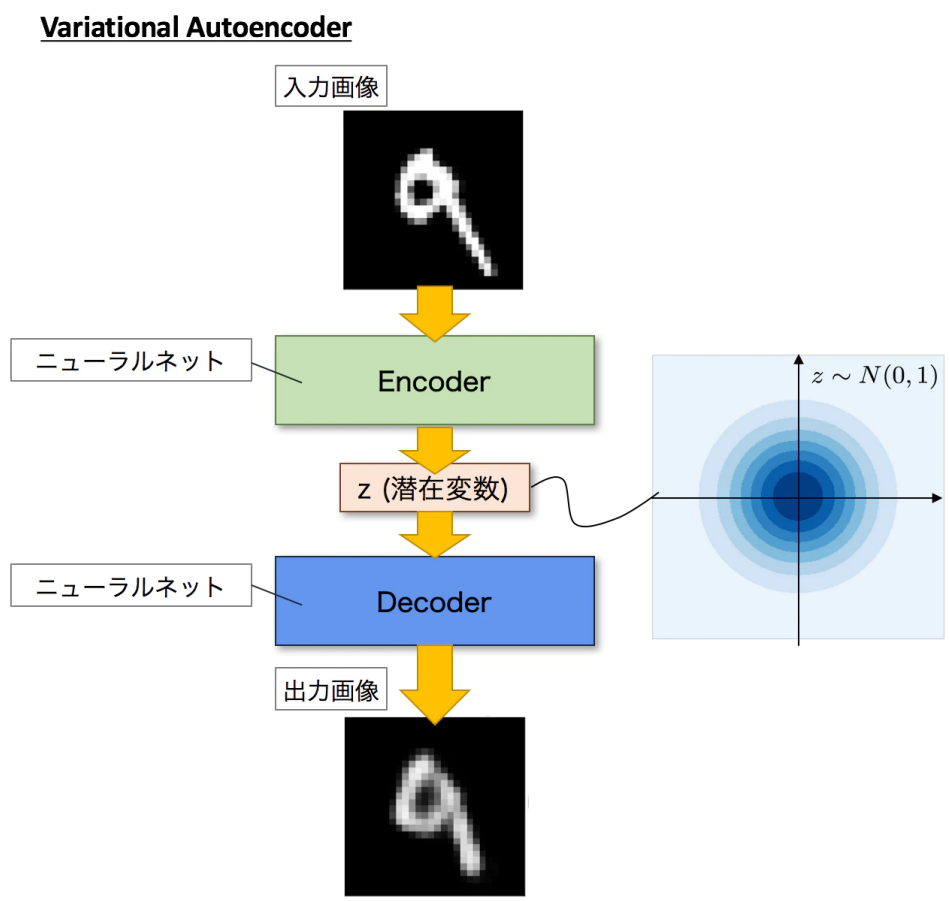
深層生成モデル（VAE・GAN）

- どんなことができるのか、手を動かしてみてみましょう

VAE (Variational Auto Encoder)

- VAEは深層生成モデルの一種で、与えられたデータと似たようなデータを幅広く生成することができます。
  - VAE全体はエンコードして小さい次元のベクトルに圧縮した後にデコードするような、入力を再構成するNNモデルとなっています。
  - エンコード先はが正規分布に近づくようなlossを入れることで、コンパクトに圧縮されたベクトルzにエンコードすることができ、またzを少し変化させると様々な画像をデコードすることができます。



- 出典: Variational Autoencoder徹底解説
- 目的関数

$$Loss = 再構成誤差 + zのはみだし具合(KLダイバージェンス)$$

- 時間がかかるのでepoch数を減らして実行してみます

In [1]:

```
import argparse
import torch
import torch.utils.data
from torch import nn, optim
from torch.nn import functional as F
from torchvision import datasets, transforms
from torchvision.utils import save_image


parser = argparse.ArgumentParser(description='VAE MNIST Example')
parser.add_argument('--batch-size', type=int, default=128, metavar='N',
                    help='input batch size for training (default: 128)')
parser.add_argument('--epochs', type=int, default=8, metavar='N',
                    help='number of epochs to train (default: 10)')
parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='disables CUDA training')
parser.add_argument('--seed', type=int, default=1, metavar='S',
                    help='random seed (default: 1)')
```

```
parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                    help='how many batches to wait before logging training status')
args = parser.parse_args("".split())

args.cuda = not args.no_cuda and torch.cuda.is_available()

torch.manual_seed(args.seed)

device = torch.device("cuda" if args.cuda else "cpu")

kwargs = {'num_workers': 1, 'pin_memory': True} if args.cuda else {}
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.ToTensor()),
    batch_size=args.batch_size, shuffle=True, **kwargs)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.ToTensor()),
    batch_size=args.batch_size, shuffle=True, **kwargs)


class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()

        self.fc1 = nn.Linear(784, 400)
        self.fc21 = nn.Linear(400, 20)
        self.fc22 = nn.Linear(400, 20)
        self.fc3 = nn.Linear(20, 400)
        self.fc4 = nn.Linear(400, 784)

    def encode(self, x):
        h1 = F.relu(self.fc1(x))
        return self.fc21(h1), self.fc22(h1)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5*logvar)
        eps = torch.randn_like(std)
        return mu + eps*std

    def decode(self, z):
        h3 = F.relu(self.fc3(z))
        return torch.sigmoid(self.fc4(h3))

    def forward(self, x):
        mu, logvar = self.encode(x.view(-1, 784))
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar


model = VAE().to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)


# Reconstruction + KL divergence losses summed over all elements and batch
def loss_function(recon_x, x, mu, logvar):
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction='sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD


def train(epoch):
    model.train()
    train_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.to(device)
        optimizer.zero_grad()
        recon_batch, mu, logvar = model(data)
        loss = loss_function(recon_batch, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]¥tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader),
                loss.item() / len(data)))

    print('====> Epoch: {} Average loss: {:.4f}'.format(
          epoch, train_loss / len(train_loader.dataset)))


def test(epoch):
    model.eval()
    test_loss = 0
    with torch.no_grad():
        for i, (data, _) in enumerate(test_loader):
            data = data.to(device)
```

```
            recon_batch, mu, logvar = model(data)
            test_loss += loss_function(recon_batch, data, mu, logvar).item()
            if i == 0:
                n = min(data.size(0), 8)
                comparison = torch.cat([data[:n],
                                        recon_batch.view(args.batch_size, 1, 28, 28)[:n]])
                save_image(comparison.cpu(),
                         'reconstruction_' + str(epoch) + '.png', nrow=n)

    test_loss /= len(test_loader.dataset)
    print('====> Test set loss: {:.4f}'.format(test_loss))


for epoch in range(1, args.epochs + 1):
    train(epoch)
    test(epoch)
    with torch.no_grad():
        sample = torch.randn(64, 20).to(device)
        sample = model.decode(sample).cpu()
        save_image(sample.view(64, 1, 28, 28),
                   'sample_' + str(epoch) + '.png')
```

```
Train Epoch: 1 [0/60000 (0%)]   Loss: 550.595764
Train Epoch: 1 [1280/60000 (2%)]        Loss: 287.982056
Train Epoch: 1 [2560/60000 (4%)]        Loss: 242.801651
Train Epoch: 1 [3840/60000 (6%)]        Loss: 213.623199
Train Epoch: 1 [5120/60000 (9%)]        Loss: 216.030853
Train Epoch: 1 [6400/60000 (11%)]       Loss: 216.303207
Train Epoch: 1 [7680/60000 (13%)]       Loss: 203.971207
Train Epoch: 1 [8960/60000 (15%)]       Loss: 197.527985
Train Epoch: 1 [10240/60000 (17%)]      Loss: 199.671753
Train Epoch: 1 [11520/60000 (19%)]      Loss: 192.269043
Train Epoch: 1 [12800/60000 (21%)]      Loss: 186.938980
Train Epoch: 1 [14080/60000 (23%)]      Loss: 179.695877
Train Epoch: 1 [15360/60000 (26%)]      Loss: 168.194870
Train Epoch: 1 [16640/60000 (28%)]      Loss: 165.249542
Train Epoch: 1 [17920/60000 (30%)]      Loss: 166.312332
Train Epoch: 1 [19200/60000 (32%)]      Loss: 162.762482
Train Epoch: 1 [20480/60000 (34%)]      Loss: 167.461792
Train Epoch: 1 [21760/60000 (36%)]      Loss: 154.407196
Train Epoch: 1 [23040/60000 (38%)]      Loss: 153.276062
Train Epoch: 1 [24320/60000 (41%)]      Loss: 149.537003
Train Epoch: 1 [25600/60000 (43%)]      Loss: 154.969391
Train Epoch: 1 [26880/60000 (45%)]      Loss: 156.321213
Train Epoch: 1 [28160/60000 (47%)]      Loss: 148.482239
Train Epoch: 1 [29440/60000 (49%)]      Loss: 149.441269
Train Epoch: 1 [30720/60000 (51%)]      Loss: 144.996063
Train Epoch: 1 [32000/60000 (53%)]      Loss: 143.773865
Train Epoch: 1 [33280/60000 (55%)]      Loss: 143.741455
Train Epoch: 1 [34560/60000 (58%)]      Loss: 145.463394
Train Epoch: 1 [35840/60000 (60%)]      Loss: 142.603760
Train Epoch: 1 [37120/60000 (62%)]      Loss: 138.537796
Train Epoch: 1 [38400/60000 (64%)]      Loss: 136.863266
Train Epoch: 1 [39680/60000 (66%)]      Loss: 141.317612
Train Epoch: 1 [40960/60000 (68%)]      Loss: 135.600006
Train Epoch: 1 [42240/60000 (70%)]      Loss: 137.085205
Train Epoch: 1 [43520/60000 (72%)]      Loss: 135.763840
Train Epoch: 1 [44800/60000 (75%)]      Loss: 130.929977
Train Epoch: 1 [46080/60000 (77%)]      Loss: 138.923218
Train Epoch: 1 [47360/60000 (79%)]      Loss: 137.344742
Train Epoch: 1 [48640/60000 (81%)]      Loss: 135.910385
Train Epoch: 1 [49920/60000 (83%)]      Loss: 129.528473
Train Epoch: 1 [51200/60000 (85%)]      Loss: 132.102951
Train Epoch: 1 [52480/60000 (87%)]      Loss: 134.186996
Train Epoch: 1 [53760/60000 (90%)]      Loss: 126.278847
Train Epoch: 1 [55040/60000 (92%)]      Loss: 132.283127
Train Epoch: 1 [56320/60000 (94%)]      Loss: 132.642441
Train Epoch: 1 [57600/60000 (96%)]      Loss: 129.768478
Train Epoch: 1 [58880/60000 (98%)]      Loss: 134.149750
====> Epoch: 1 Average loss: 164.8554
====> Test set loss: 128.1345
Train Epoch: 2 [0/60000 (0%)]   Loss: 123.857483
Train Epoch: 2 [1280/60000 (2%)]        Loss: 130.576630
Train Epoch: 2 [2560/60000 (4%)]        Loss: 120.633873
Train Epoch: 2 [3840/60000 (6%)]        Loss: 128.934464
Train Epoch: 2 [5120/60000 (9%)]        Loss: 129.286057
Train Epoch: 2 [6400/60000 (11%)]       Loss: 126.731430
Train Epoch: 2 [7680/60000 (13%)]       Loss: 126.215179
Train Epoch: 2 [8960/60000 (15%)]       Loss: 127.598969
Train Epoch: 2 [10240/60000 (17%)]      Loss: 124.811150
Train Epoch: 2 [11520/60000 (19%)]      Loss: 120.234749
Train Epoch: 2 [12800/60000 (21%)]      Loss: 125.330246
Train Epoch: 2 [14080/60000 (23%)]      Loss: 125.615234
Train Epoch: 2 [15360/60000 (26%)]      Loss: 124.513947
Train Epoch: 2 [16640/60000 (28%)]      Loss: 126.385956
Train Epoch: 2 [17920/60000 (30%)]      Loss: 122.737869
Train Epoch: 2 [19200/60000 (32%)]      Loss: 123.886749
Train Epoch: 2 [20480/60000 (34%)]      Loss: 122.428024
Train Epoch: 2 [21760/60000 (36%)]      Loss: 123.816864
Train Epoch: 2 [23040/60000 (38%)]      Loss: 123.283905
Train Epoch: 2 [24320/60000 (41%)]      Loss: 122.145889
Train Epoch: 2 [25600/60000 (43%)]      Loss: 118.836884
Train Epoch: 2 [26880/60000 (45%)]      Loss: 118.756752
Train Epoch: 2 [28160/60000 (47%)]      Loss: 125.290443
Train Epoch: 2 [29440/60000 (49%)]      Loss: 122.523354
Train Epoch: 2 [30720/60000 (51%)]      Loss: 117.786148
Train Epoch: 2 [32000/60000 (53%)]      Loss: 119.198685
Train Epoch: 2 [33280/60000 (55%)]      Loss: 124.256905
Train Epoch: 2 [34560/60000 (58%)]      Loss: 117.114510
Train Epoch: 2 [35840/60000 (60%)]      Loss: 115.025963
Train Epoch: 2 [37120/60000 (62%)]      Loss: 119.339828
```

```
Train Epoch: 2 [38400/60000 (64%)]      Loss: 120.265221
Train Epoch: 2 [39680/60000 (66%)]      Loss: 121.589767
Train Epoch: 2 [40960/60000 (68%)]      Loss: 115.960037
Train Epoch: 2 [42240/60000 (70%)]      Loss: 116.858063
Train Epoch: 2 [43520/60000 (72%)]      Loss: 120.354919
Train Epoch: 2 [44800/60000 (75%)]      Loss: 119.434906
Train Epoch: 2 [46080/60000 (77%)]      Loss: 114.847801
Train Epoch: 2 [47360/60000 (79%)]      Loss: 115.614342
Train Epoch: 2 [48640/60000 (81%)]      Loss: 119.164261
Train Epoch: 2 [49920/60000 (83%)]      Loss: 121.876266
Train Epoch: 2 [51200/60000 (85%)]      Loss: 121.933479
Train Epoch: 2 [52480/60000 (87%)]      Loss: 121.828873
Train Epoch: 2 [53760/60000 (90%)]      Loss: 115.843834
Train Epoch: 2 [55040/60000 (92%)]      Loss: 114.530930
Train Epoch: 2 [56320/60000 (94%)]      Loss: 122.686127
Train Epoch: 2 [57600/60000 (96%)]      Loss: 113.210793
Train Epoch: 2 [58880/60000 (98%)]      Loss: 121.891304
====> Epoch: 2 Average loss: 122.0216
====> Test set loss: 116.0042
Train Epoch: 3 [0/60000 (0%)]   Loss: 117.037003
Train Epoch: 3 [1280/60000 (2%)]        Loss: 114.776764
Train Epoch: 3 [2560/60000 (4%)]        Loss: 114.757584
Train Epoch: 3 [3840/60000 (6%)]        Loss: 119.504852
Train Epoch: 3 [5120/60000 (9%)]        Loss: 113.168205
Train Epoch: 3 [6400/60000 (11%)]       Loss: 119.137169
Train Epoch: 3 [7680/60000 (13%)]       Loss: 113.456680
Train Epoch: 3 [8960/60000 (15%)]       Loss: 113.398407
Train Epoch: 3 [10240/60000 (17%)]      Loss: 113.233376
Train Epoch: 3 [11520/60000 (19%)]      Loss: 116.266098
Train Epoch: 3 [12800/60000 (21%)]      Loss: 116.503098
Train Epoch: 3 [14080/60000 (23%)]      Loss: 115.268875
Train Epoch: 3 [15360/60000 (26%)]      Loss: 112.991455
Train Epoch: 3 [16640/60000 (28%)]      Loss: 116.597580
Train Epoch: 3 [17920/60000 (30%)]      Loss: 116.905830
Train Epoch: 3 [19200/60000 (32%)]      Loss: 109.774872
Train Epoch: 3 [20480/60000 (34%)]      Loss: 113.501366
Train Epoch: 3 [21760/60000 (36%)]      Loss: 113.274178
Train Epoch: 3 [23040/60000 (38%)]      Loss: 112.505630
Train Epoch: 3 [24320/60000 (41%)]      Loss: 112.741570
Train Epoch: 3 [25600/60000 (43%)]      Loss: 118.447845
Train Epoch: 3 [26880/60000 (45%)]      Loss: 120.662521
Train Epoch: 3 [28160/60000 (47%)]      Loss: 111.600677
Train Epoch: 3 [29440/60000 (49%)]      Loss: 115.883194
Train Epoch: 3 [30720/60000 (51%)]      Loss: 116.404755
Train Epoch: 3 [32000/60000 (53%)]      Loss: 115.922226
Train Epoch: 3 [33280/60000 (55%)]      Loss: 116.282509
Train Epoch: 3 [34560/60000 (58%)]      Loss: 111.363457
Train Epoch: 3 [35840/60000 (60%)]      Loss: 116.081230
Train Epoch: 3 [37120/60000 (62%)]      Loss: 120.048325
Train Epoch: 3 [38400/60000 (64%)]      Loss: 116.040108
Train Epoch: 3 [39680/60000 (66%)]      Loss: 116.887070
Train Epoch: 3 [40960/60000 (68%)]      Loss: 113.031456
Train Epoch: 3 [42240/60000 (70%)]      Loss: 113.499130
Train Epoch: 3 [43520/60000 (72%)]      Loss: 113.330200
Train Epoch: 3 [44800/60000 (75%)]      Loss: 113.574554
Train Epoch: 3 [46080/60000 (77%)]      Loss: 114.805374
Train Epoch: 3 [47360/60000 (79%)]      Loss: 114.781822
Train Epoch: 3 [48640/60000 (81%)]      Loss: 109.210091
Train Epoch: 3 [49920/60000 (83%)]      Loss: 112.337555
Train Epoch: 3 [51200/60000 (85%)]      Loss: 114.106903
Train Epoch: 3 [52480/60000 (87%)]      Loss: 112.466087
Train Epoch: 3 [53760/60000 (90%)]      Loss: 114.904022
Train Epoch: 3 [55040/60000 (92%)]      Loss: 112.291801
Train Epoch: 3 [56320/60000 (94%)]      Loss: 114.647423
Train Epoch: 3 [57600/60000 (96%)]      Loss: 112.924667
Train Epoch: 3 [58880/60000 (98%)]      Loss: 111.304520
====> Epoch: 3 Average loss: 114.5006
====> Test set loss: 111.4008
Train Epoch: 4 [0/60000 (0%)]   Loss: 112.052094
Train Epoch: 4 [1280/60000 (2%)]        Loss: 111.231529
Train Epoch: 4 [2560/60000 (4%)]        Loss: 112.165092
Train Epoch: 4 [3840/60000 (6%)]        Loss: 115.582779
Train Epoch: 4 [5120/60000 (9%)]        Loss: 108.364258
Train Epoch: 4 [6400/60000 (11%)]       Loss: 113.595276
Train Epoch: 4 [7680/60000 (13%)]       Loss: 109.402222
Train Epoch: 4 [8960/60000 (15%)]       Loss: 116.161217
Train Epoch: 4 [10240/60000 (17%)]      Loss: 117.132889
Train Epoch: 4 [11520/60000 (19%)]      Loss: 112.577927
Train Epoch: 4 [12800/60000 (21%)]      Loss: 112.747391
Train Epoch: 4 [14080/60000 (23%)]      Loss: 115.483231
Train Epoch: 4 [15360/60000 (26%)]      Loss: 117.257347
Train Epoch: 4 [16640/60000 (28%)]      Loss: 110.050247
Train Epoch: 4 [17920/60000 (30%)]      Loss: 108.694374
Train Epoch: 4 [19200/60000 (32%)]      Loss: 109.969170
Train Epoch: 4 [20480/60000 (34%)]      Loss: 111.543533
Train Epoch: 4 [21760/60000 (36%)]      Loss: 115.150169
Train Epoch: 4 [23040/60000 (38%)]      Loss: 108.276428
Train Epoch: 4 [24320/60000 (41%)]      Loss: 110.876335
Train Epoch: 4 [25600/60000 (43%)]      Loss: 110.621185
Train Epoch: 4 [26880/60000 (45%)]      Loss: 109.089874
Train Epoch: 4 [28160/60000 (47%)]      Loss: 110.446648
Train Epoch: 4 [29440/60000 (49%)]      Loss: 111.123375
Train Epoch: 4 [30720/60000 (51%)]      Loss: 111.446587
Train Epoch: 4 [32000/60000 (53%)]      Loss: 108.972908
Train Epoch: 4 [33280/60000 (55%)]      Loss: 110.763618
Train Epoch: 4 [34560/60000 (58%)]      Loss: 114.331444
Train Epoch: 4 [35840/60000 (60%)]      Loss: 110.222778
Train Epoch: 4 [37120/60000 (62%)]      Loss: 111.312355
Train Epoch: 4 [38400/60000 (64%)]      Loss: 111.650116
Train Epoch: 4 [39680/60000 (66%)]      Loss: 113.275818
Train Epoch: 4 [40960/60000 (68%)]      Loss: 111.395370
Train Epoch: 4 [42240/60000 (70%)]      Loss: 113.030807
Train Epoch: 4 [43520/60000 (72%)]      Loss: 111.500259
Train Epoch: 4 [44800/60000 (75%)]      Loss: 110.398109
Train Epoch: 4 [46080/60000 (77%)]      Loss: 108.656761
Train Epoch: 4 [47360/60000 (79%)]      Loss: 108.651413
Train Epoch: 4 [48640/60000 (81%)]      Loss: 105.984497
```

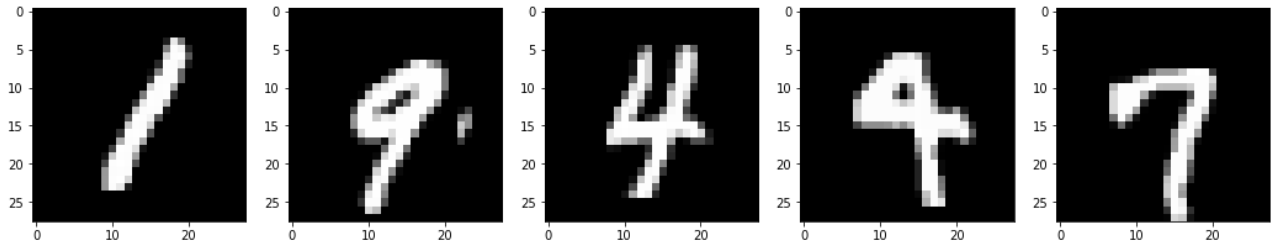
```
Train Epoch: 4 [49920/60000 (83%)]      Loss: 112.042656
Train Epoch: 4 [51200/60000 (85%)]      Loss: 108.484085
Train Epoch: 4 [52480/60000 (87%)]      Loss: 112.930649
Train Epoch: 4 [53760/60000 (90%)]      Loss: 115.461166
Train Epoch: 4 [55040/60000 (92%)]      Loss: 112.013062
Train Epoch: 4 [56320/60000 (94%)]      Loss: 106.776459
Train Epoch: 4 [57600/60000 (96%)]      Loss: 115.344437
Train Epoch: 4 [58880/60000 (98%)]      Loss: 109.698456
====> Epoch: 4 Average loss: 111.4363
====> Test set loss: 109.4620
Train Epoch: 5 [0/60000 (0%)]   Loss: 111.082909
Train Epoch: 5 [1280/60000 (2%)]        Loss: 113.060677
Train Epoch: 5 [2560/60000 (4%)]        Loss: 113.950851
Train Epoch: 5 [3840/60000 (6%)]        Loss: 115.257408
Train Epoch: 5 [5120/60000 (9%)]        Loss: 109.494102
Train Epoch: 5 [6400/60000 (11%)]       Loss: 112.074158
Train Epoch: 5 [7680/60000 (13%)]       Loss: 111.627899
Train Epoch: 5 [8960/60000 (15%)]       Loss: 108.465790
Train Epoch: 5 [10240/60000 (17%)]      Loss: 110.462448
Train Epoch: 5 [11520/60000 (19%)]      Loss: 109.300110
Train Epoch: 5 [12800/60000 (21%)]      Loss: 109.709923
Train Epoch: 5 [14080/60000 (23%)]      Loss: 111.102646
Train Epoch: 5 [15360/60000 (26%)]      Loss: 109.874336
Train Epoch: 5 [16640/60000 (28%)]      Loss: 110.200691
Train Epoch: 5 [17920/60000 (30%)]      Loss: 113.683266
Train Epoch: 5 [19200/60000 (32%)]      Loss: 117.210953
Train Epoch: 5 [20480/60000 (34%)]      Loss: 107.724884
Train Epoch: 5 [21760/60000 (36%)]      Loss: 112.185410
Train Epoch: 5 [23040/60000 (38%)]      Loss: 109.684418
Train Epoch: 5 [24320/60000 (41%)]      Loss: 108.938889
Train Epoch: 5 [25600/60000 (43%)]      Loss: 113.837997
Train Epoch: 5 [26880/60000 (45%)]      Loss: 112.577629
Train Epoch: 5 [28160/60000 (47%)]      Loss: 111.305290
Train Epoch: 5 [29440/60000 (49%)]      Loss: 109.659409
Train Epoch: 5 [30720/60000 (51%)]      Loss: 107.261696
Train Epoch: 5 [32000/60000 (53%)]      Loss: 106.404953
Train Epoch: 5 [33280/60000 (55%)]      Loss: 104.664246
Train Epoch: 5 [34560/60000 (58%)]      Loss: 111.269928
Train Epoch: 5 [35840/60000 (60%)]      Loss: 107.945465
Train Epoch: 5 [37120/60000 (62%)]      Loss: 107.384857
Train Epoch: 5 [38400/60000 (64%)]      Loss: 109.551552
Train Epoch: 5 [39680/60000 (66%)]      Loss: 114.005356
Train Epoch: 5 [40960/60000 (68%)]      Loss: 110.215042
Train Epoch: 5 [42240/60000 (70%)]      Loss: 110.454414
Train Epoch: 5 [43520/60000 (72%)]      Loss: 107.744980
Train Epoch: 5 [44800/60000 (75%)]      Loss: 109.709808
Train Epoch: 5 [46080/60000 (77%)]      Loss: 109.344131
Train Epoch: 5 [47360/60000 (79%)]      Loss: 108.325424
Train Epoch: 5 [48640/60000 (81%)]      Loss: 113.565521
Train Epoch: 5 [49920/60000 (83%)]      Loss: 106.110939
Train Epoch: 5 [51200/60000 (85%)]      Loss: 110.159134
Train Epoch: 5 [52480/60000 (87%)]      Loss: 113.429108
Train Epoch: 5 [53760/60000 (90%)]      Loss: 107.978783
Train Epoch: 5 [55040/60000 (92%)]      Loss: 113.720276
Train Epoch: 5 [56320/60000 (94%)]      Loss: 109.164856
Train Epoch: 5 [57600/60000 (96%)]      Loss: 111.169739
Train Epoch: 5 [58880/60000 (98%)]      Loss: 109.429985
====> Epoch: 5 Average loss: 109.7120
====> Test set loss: 108.2358
Train Epoch: 6 [0/60000 (0%)]   Loss: 112.188278
Train Epoch: 6 [1280/60000 (2%)]        Loss: 108.195358
Train Epoch: 6 [2560/60000 (4%)]        Loss: 112.676460
Train Epoch: 6 [3840/60000 (6%)]        Loss: 105.103088
Train Epoch: 6 [5120/60000 (9%)]        Loss: 105.141190
Train Epoch: 6 [6400/60000 (11%)]       Loss: 109.508141
Train Epoch: 6 [7680/60000 (13%)]       Loss: 108.504074
Train Epoch: 6 [8960/60000 (15%)]       Loss: 107.786743
Train Epoch: 6 [10240/60000 (17%)]      Loss: 111.271736
Train Epoch: 6 [11520/60000 (19%)]      Loss: 108.949448
Train Epoch: 6 [12800/60000 (21%)]      Loss: 106.638344
Train Epoch: 6 [14080/60000 (23%)]      Loss: 113.686966
Train Epoch: 6 [15360/60000 (26%)]      Loss: 105.156342
Train Epoch: 6 [16640/60000 (28%)]      Loss: 111.776360
Train Epoch: 6 [17920/60000 (30%)]      Loss: 110.850121
Train Epoch: 6 [19200/60000 (32%)]      Loss: 109.335625
Train Epoch: 6 [20480/60000 (34%)]      Loss: 108.057861
Train Epoch: 6 [21760/60000 (36%)]      Loss: 109.381905
Train Epoch: 6 [23040/60000 (38%)]      Loss: 110.732544
Train Epoch: 6 [24320/60000 (41%)]      Loss: 109.711472
Train Epoch: 6 [25600/60000 (43%)]      Loss: 108.162476
Train Epoch: 6 [26880/60000 (45%)]      Loss: 104.834122
Train Epoch: 6 [28160/60000 (47%)]      Loss: 108.348869
Train Epoch: 6 [29440/60000 (49%)]      Loss: 105.496819
Train Epoch: 6 [30720/60000 (51%)]      Loss: 111.441025
Train Epoch: 6 [32000/60000 (53%)]      Loss: 104.859467
Train Epoch: 6 [33280/60000 (55%)]      Loss: 108.636429
Train Epoch: 6 [34560/60000 (58%)]      Loss: 111.124359
Train Epoch: 6 [35840/60000 (60%)]      Loss: 106.693352
Train Epoch: 6 [37120/60000 (62%)]      Loss: 109.697281
Train Epoch: 6 [38400/60000 (64%)]      Loss: 107.597961
Train Epoch: 6 [39680/60000 (66%)]      Loss: 104.770401
Train Epoch: 6 [40960/60000 (68%)]      Loss: 109.415764
Train Epoch: 6 [42240/60000 (70%)]      Loss: 107.780846
Train Epoch: 6 [43520/60000 (72%)]      Loss: 106.667442
Train Epoch: 6 [44800/60000 (75%)]      Loss: 111.479347
Train Epoch: 6 [46080/60000 (77%)]      Loss: 103.173615
Train Epoch: 6 [47360/60000 (79%)]      Loss: 106.479256
Train Epoch: 6 [48640/60000 (81%)]      Loss: 108.134995
Train Epoch: 6 [49920/60000 (83%)]      Loss: 106.888885
Train Epoch: 6 [51200/60000 (85%)]      Loss: 107.642944
Train Epoch: 6 [52480/60000 (87%)]      Loss: 108.485054
Train Epoch: 6 [53760/60000 (90%)]      Loss: 102.809845
Train Epoch: 6 [55040/60000 (92%)]      Loss: 108.560776
Train Epoch: 6 [56320/60000 (94%)]      Loss: 109.021576
Train Epoch: 6 [57600/60000 (96%)]      Loss: 112.878906
Train Epoch: 6 [58880/60000 (98%)]      Loss: 108.373764
====> Epoch: 6 Average loss: 108.5484
```

```
====> Test set loss: 107.3431
Train Epoch: 7 [0/60000 (0%)]   Loss: 108.115082
Train Epoch: 7 [1280/60000 (2%)]        Loss: 109.145676
Train Epoch: 7 [2560/60000 (4%)]        Loss: 107.465958
Train Epoch: 7 [3840/60000 (6%)]        Loss: 109.015228
Train Epoch: 7 [5120/60000 (9%)]        Loss: 107.590248
Train Epoch: 7 [6400/60000 (11%)]       Loss: 108.954834
Train Epoch: 7 [7680/60000 (13%)]       Loss: 109.174400
Train Epoch: 7 [8960/60000 (15%)]       Loss: 102.631935
Train Epoch: 7 [10240/60000 (17%)]      Loss: 107.866821
Train Epoch: 7 [11520/60000 (19%)]      Loss: 109.493500
Train Epoch: 7 [12800/60000 (21%)]      Loss: 104.674149
Train Epoch: 7 [14080/60000 (23%)]      Loss: 108.831528
Train Epoch: 7 [15360/60000 (26%)]      Loss: 104.159973
Train Epoch: 7 [16640/60000 (28%)]      Loss: 106.838104
Train Epoch: 7 [17920/60000 (30%)]      Loss: 107.023399
Train Epoch: 7 [19200/60000 (32%)]      Loss: 104.524567
Train Epoch: 7 [20480/60000 (34%)]      Loss: 104.836525
Train Epoch: 7 [21760/60000 (36%)]      Loss: 109.801254
Train Epoch: 7 [23040/60000 (38%)]      Loss: 109.171799
Train Epoch: 7 [24320/60000 (41%)]      Loss: 106.634048
Train Epoch: 7 [25600/60000 (43%)]      Loss: 108.324341
Train Epoch: 7 [26880/60000 (45%)]      Loss: 108.384560
Train Epoch: 7 [28160/60000 (47%)]      Loss: 111.276222
Train Epoch: 7 [29440/60000 (49%)]      Loss: 111.967995
Train Epoch: 7 [30720/60000 (51%)]      Loss: 102.527031
Train Epoch: 7 [32000/60000 (53%)]      Loss: 105.052475
Train Epoch: 7 [33280/60000 (55%)]      Loss: 106.157318
Train Epoch: 7 [34560/60000 (58%)]      Loss: 108.033531
Train Epoch: 7 [35840/60000 (60%)]      Loss: 110.111763
Train Epoch: 7 [37120/60000 (62%)]      Loss: 106.192932
Train Epoch: 7 [38400/60000 (64%)]      Loss: 108.618652
Train Epoch: 7 [39680/60000 (66%)]      Loss: 106.823578
Train Epoch: 7 [40960/60000 (68%)]      Loss: 105.517220
Train Epoch: 7 [42240/60000 (70%)]      Loss: 108.897141
Train Epoch: 7 [43520/60000 (72%)]      Loss: 107.661545
Train Epoch: 7 [44800/60000 (75%)]      Loss: 110.947922
Train Epoch: 7 [46080/60000 (77%)]      Loss: 107.206192
Train Epoch: 7 [47360/60000 (79%)]      Loss: 108.167450
Train Epoch: 7 [48640/60000 (81%)]      Loss: 107.913597
Train Epoch: 7 [49920/60000 (83%)]      Loss: 109.134621
Train Epoch: 7 [51200/60000 (85%)]      Loss: 109.836945
Train Epoch: 7 [52480/60000 (87%)]      Loss: 107.283005
Train Epoch: 7 [53760/60000 (90%)]      Loss: 105.503799
Train Epoch: 7 [55040/60000 (92%)]      Loss: 108.322746
Train Epoch: 7 [56320/60000 (94%)]      Loss: 109.833153
Train Epoch: 7 [57600/60000 (96%)]      Loss: 101.845757
Train Epoch: 7 [58880/60000 (98%)]      Loss: 104.239166
====> Epoch: 7 Average loss: 107.6947
====> Test set loss: 106.7860
Train Epoch: 8 [0/60000 (0%)]   Loss: 105.993454
Train Epoch: 8 [1280/60000 (2%)]        Loss: 106.482254
Train Epoch: 8 [2560/60000 (4%)]        Loss: 104.721931
Train Epoch: 8 [3840/60000 (6%)]        Loss: 103.486046
Train Epoch: 8 [5120/60000 (9%)]        Loss: 103.258614
Train Epoch: 8 [6400/60000 (11%)]       Loss: 109.474426
Train Epoch: 8 [7680/60000 (13%)]       Loss: 106.652878
Train Epoch: 8 [8960/60000 (15%)]       Loss: 104.647041
Train Epoch: 8 [10240/60000 (17%)]      Loss: 109.504974
Train Epoch: 8 [11520/60000 (19%)]      Loss: 106.101479
Train Epoch: 8 [12800/60000 (21%)]      Loss: 108.611481
Train Epoch: 8 [14080/60000 (23%)]      Loss: 103.320038
Train Epoch: 8 [15360/60000 (26%)]      Loss: 110.706955
Train Epoch: 8 [16640/60000 (28%)]      Loss: 110.568359
Train Epoch: 8 [17920/60000 (30%)]      Loss: 102.253708
Train Epoch: 8 [19200/60000 (32%)]      Loss: 107.956505
Train Epoch: 8 [20480/60000 (34%)]      Loss: 100.032623
Train Epoch: 8 [21760/60000 (36%)]      Loss: 107.028885
Train Epoch: 8 [23040/60000 (38%)]      Loss: 110.031487
Train Epoch: 8 [24320/60000 (41%)]      Loss: 108.483749
Train Epoch: 8 [25600/60000 (43%)]      Loss: 104.010208
Train Epoch: 8 [26880/60000 (45%)]      Loss: 109.660843
Train Epoch: 8 [28160/60000 (47%)]      Loss: 109.877090
Train Epoch: 8 [29440/60000 (49%)]      Loss: 110.470001
Train Epoch: 8 [30720/60000 (51%)]      Loss: 105.574669
Train Epoch: 8 [32000/60000 (53%)]      Loss: 110.672394
Train Epoch: 8 [33280/60000 (55%)]      Loss: 106.926361
Train Epoch: 8 [34560/60000 (58%)]      Loss: 109.081535
Train Epoch: 8 [35840/60000 (60%)]      Loss: 107.863922
Train Epoch: 8 [37120/60000 (62%)]      Loss: 104.871292
Train Epoch: 8 [38400/60000 (64%)]      Loss: 106.573280
Train Epoch: 8 [39680/60000 (66%)]      Loss: 109.174339
Train Epoch: 8 [40960/60000 (68%)]      Loss: 105.720230
Train Epoch: 8 [42240/60000 (70%)]      Loss: 108.013596
Train Epoch: 8 [43520/60000 (72%)]      Loss: 107.756058
Train Epoch: 8 [44800/60000 (75%)]      Loss: 104.752014
Train Epoch: 8 [46080/60000 (77%)]      Loss: 105.799866
Train Epoch: 8 [47360/60000 (79%)]      Loss: 107.535027
Train Epoch: 8 [48640/60000 (81%)]      Loss: 103.296608
Train Epoch: 8 [49920/60000 (83%)]      Loss: 109.784622
Train Epoch: 8 [51200/60000 (85%)]      Loss: 107.876450
Train Epoch: 8 [52480/60000 (87%)]      Loss: 112.150925
Train Epoch: 8 [53760/60000 (90%)]      Loss: 107.237457
Train Epoch: 8 [55040/60000 (92%)]      Loss: 108.191856
Train Epoch: 8 [56320/60000 (94%)]      Loss: 106.435379
Train Epoch: 8 [57600/60000 (96%)]      Loss: 108.785271
Train Epoch: 8 [58880/60000 (98%)]      Loss: 104.415237
====> Epoch: 8 Average loss: 107.0709
====> Test set loss: 106.3582
```

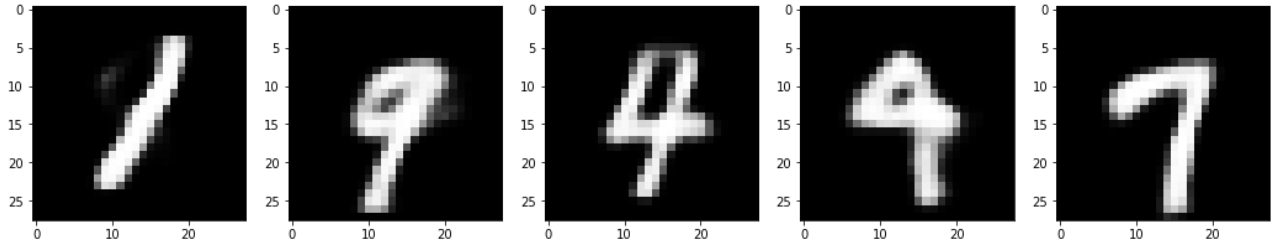
In [2]:

```
N = 5
data, _ = iter(test_loader).next()
data = data[:N].to(device)
```

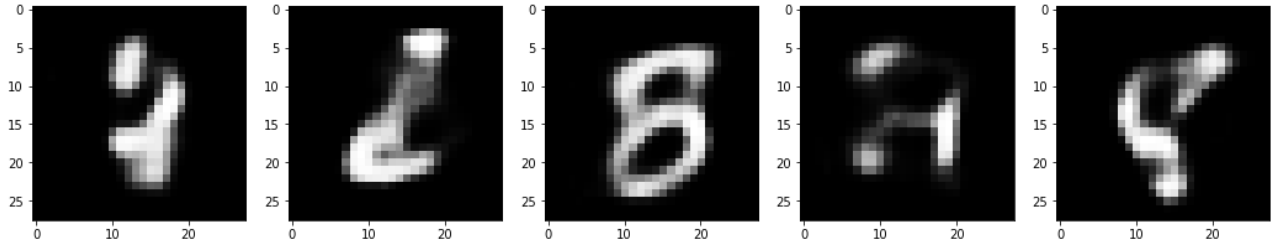
In [3]:
```
import matplotlib.pyplot as plt

data_np = data.detach().cpu().numpy()

plt.figure(figsize=(18,18))
for i in range(N):
  plt.subplot(1,N,i+1); plt.imshow(data_np[i].squeeze(), "gray")
plt.show()
```



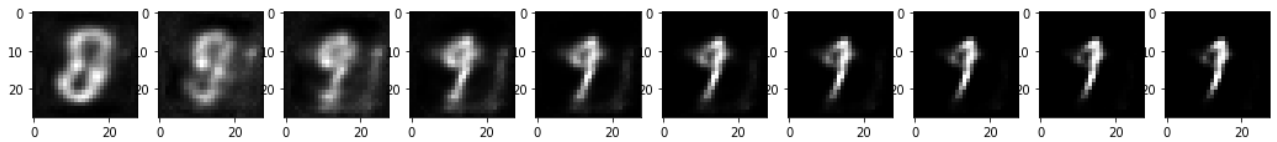
In [4]:
```
recon_batch, mu, logvar = model(data)
recon_batch = recon_batch.view(N, 28, 28).detach().cpu().numpy()

plt.figure(figsize=(18,18))
for i in range(N):
  plt.subplot(1,N,i+1); plt.imshow(recon_batch[i], "gray")
plt.show()
```



In [5]:
```
N = 5

sample = torch.randn(N, 20).to(device)
sample = model.decode(sample)

sample = sample.view(N, 28, 28).detach().cpu().numpy()


plt.figure(figsize=(18,18))
for i in range(N):
  plt.subplot(1,N,i+1); plt.imshow(sample[i], "gray")
plt.show()
```



In [6]:
```
a = torch.randn(20).to(device)
b = torch.randn(20).to(device)

N = 10
sample = torch.stack([r*a/float(N) + (1.-r)*b/float(N) for r in range(N)])
sample = model.decode(sample)
sample = sample.view(N, 28, 28).detach().cpu().numpy()

plt.figure(figsize=(18,18))
for i in range(N):
  plt.subplot(1,N,i+1); plt.imshow(sample[i], "gray")
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

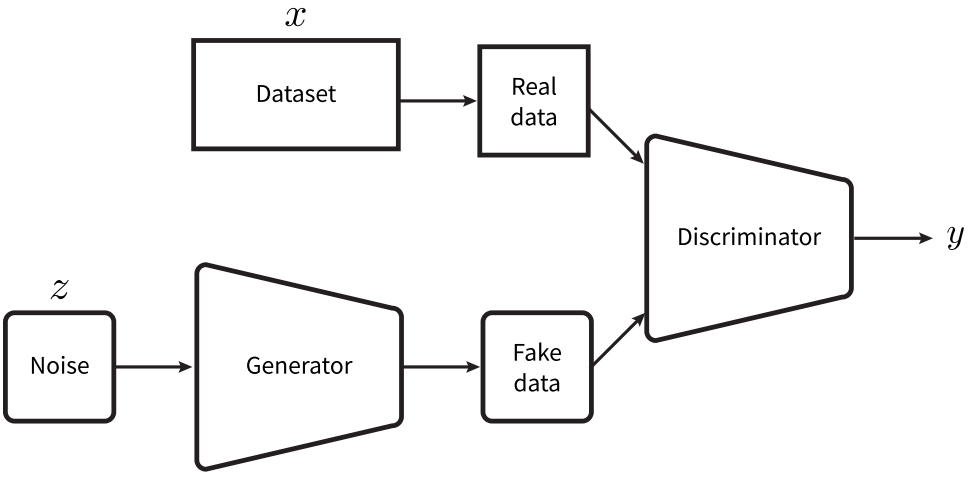
In [ ]:

In [ ]:

In [ ]:

GAN (Generative Adversarial Networks)

- GANは深層生成モデルの一種で、与えられたデータと遜色ない本物のような画像を作ることができます。
  - GANの全体は生成器(G:Generator)と識別器(D:Disctiminator)の二つからなっており、生成器は識別器をだませるように、識別器は本物のデータと生成されたデータを見分けられるように、いたちごっこのような学習を行います。



出典: https://www.bigdata-navi.com/aidrops/2519/

- 目的関数
  - 1つの式に表すと以下のようになります(ミニマックス問題)

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

  - 識別器の目的関数 (D=1: 本物と判断, D=0: 偽物と判断)

$$\max_{\phi} E_{pd(x)}[\ln D(\phi; x)] + E_{p(z)}[\ln(1 - D(\phi; G(\theta; z))]$$

    - 識別器は本物は本物、偽物は偽物と識別できるようにする

$$\min_{\theta} E_{p(z)}[\ln(1 - D(\phi; G(\theta; z))]$$

  - 生成器の目的関数
    - 生成器は、本物だと識別されるようにする
- ref: http://aidiary.hatenablog.com/entry/20180304/1520172429

- 時間がかかるのでepoch数を減らして実行してみます

In [7]:
```
import os
import pickle
import numpy as np
```

```
from tqdm import tqdm
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from torchvision.utils import save_image, make_grid

import matplotlib.pyplot as plt

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)


def initialize_weights(model):
    for m in model.modules():
        if isinstance(m, nn.Conv2d):
            m.weight.data.normal_(0, 0.02)
            m.bias.data.zero_()
        elif isinstance(m, nn.ConvTranspose2d):
            m.weight.data.normal_(0, 0.02)
            m.bias.data.zero_()
        elif isinstance(m, nn.Linear):
            m.weight.data.normal_(0, 0.02)
            m.bias.data.zero_()


class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()

        self.fc = nn.Sequential(
            nn.Linear(62, 1024),
            nn.BatchNorm1d(1024),
            nn.ReLU(),
            nn.Linear(1024, 128 * 7 * 7),
            nn.BatchNorm1d(128 * 7 * 7),
            nn.ReLU(),
        )

        self.deconv = nn.Sequential(
            nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 1, kernel_size=4, stride=2, padding=1),
            nn.Sigmoid(),
        )

        initialize_weights(self)

    def forward(self, input):
        x = self.fc(input)
        x = x.view(-1, 128, 7, 7)
        x = self.deconv(x)
        return x


class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()

        self.conv = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2),
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2),
        )

        self.fc = nn.Sequential(
            nn.Linear(128 * 7 * 7, 1024),
            nn.BatchNorm1d(1024),
            nn.LeakyReLU(0.2),
            nn.Linear(1024, 1),
            nn.Sigmoid(),
        )

        initialize_weights(self)

    def forward(self, input):
        x = self.conv(input)
        x = x.view(-1, 128 * 7 * 7)
        x = self.fc(x)
        return x

# Generatorのサイズテスト
G = Generator()
```

```
input = torch.rand((32, 62))
out = G(input)
print(out.size())
torch.Size([32, 1, 28, 28])

# Discriminatorのサイズテスト
D = Discriminator()
input = torch.rand((32, 1, 28, 28))
out = D(input)
print(out.size())
torch.Size([32, 1])

# hyperparameters
batch_size = 128
lr = 0.0002
z_dim = 62
num_epochs = 10  # 25
sample_num = 16
log_dir = './'

# initialize network
G = Generator().to(device)
D = Discriminator().to(device)

# optimizer
G_optimizer = optim.Adam(G.parameters(), lr=lr, betas=(0.5, 0.999))
D_optimizer = optim.Adam(D.parameters(), lr=lr, betas=(0.5, 0.999))

# loss
criterion = nn.BCELoss()

# load dataset
transform = transforms.Compose([
    transforms.ToTensor()
])

dataset = datasets.MNIST('../data', train=True, download=True, transform=transform)
data_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True, drop_last=True)

def train(D, G, criterion, D_optimizer, G_optimizer, data_loader):
    # 訓練モードへ
    D.train()
    G.train()

    # 本物のラベルは1
    y_real = torch.ones(batch_size, 1)
    # 偽物のラベルは0
    y_fake = torch.zeros(batch_size, 1)

    y_real = y_real.to(device)
    y_fake = y_fake.to(device)

    D_running_loss = 0
    G_running_loss = 0
    for batch_idx, (real_images, _) in enumerate(tqdm(data_loader)):
        z = torch.rand((batch_size, z_dim))
        real_images, z = real_images.to(device), z.to(device)

        # Discriminatorの更新
        D_optimizer.zero_grad()

        # Discriminatorにとって本物画像の認識結果は1（本物）に近いほどよい
        # E[log(D(x))]
        D_real = D(real_images)
        D_real_loss = criterion(D_real, y_real)

        # DiscriminatorにとってGeneratorが生成した偽物画像の認識結果は0（偽物）に近いほどよい
        # E[log(1 - D(G(z)))]
        # fake_imagesを通じて勾配がGに伝わらないようにdetach()して止める
        fake_images = G(z)
        D_fake = D(fake_images.detach())
        D_fake_loss = criterion(D_fake, y_fake)

        # 2つのlossの和を最小化する
        D_loss = D_real_loss + D_fake_loss
        D_loss.backward()
        D_optimizer.step()
        D_running_loss += D_loss.item()

        # Generatorの更新
        z = torch.rand((batch_size, z_dim))
        z = z.to(device)

        G_optimizer.zero_grad()

        # GeneratorにとってGeneratorが生成した画像の認識結果は1（本物）に近いほどよい
        # E[log(D(G(z)))
```

```
        fake_images = G(z)
        D_fake = D(fake_images)
        G_loss = criterion(D_fake, y_real)
        G_loss.backward()
        G_optimizer.step()
        G_running_loss += G_loss.item()

    D_running_loss /= len(data_loader)
    G_running_loss /= len(data_loader)

    return D_running_loss, G_running_loss

def generate(epoch, G, log_dir='logs'):
    G.eval()

    os.makedirs(log_dir, exist_ok=True)

    # 生成のもとになる乱数を生成
    sample_z = torch.rand((64, z_dim))
    sample_z = sample_z.to(device)

    # Generatorでサンプル生成
    with torch.no_grad():
        samples = G(sample_z).to('cpu')
    save_image(samples, os.path.join(log_dir, 'epoch_%03d.png' % (epoch)))

history = {}
history['D_loss'] = []
history['G_loss'] = []
for epoch in range(num_epochs):
    D_loss, G_loss = train(D, G, criterion, D_optimizer, G_optimizer, data_loader)

    print('epoch %d, D_loss: %.4f G_loss: %.4f' % (epoch + 1, D_loss, G_loss))
    history['D_loss'].append(D_loss)
    history['G_loss'].append(G_loss)

    # 特定のエポックでGeneratorから画像を生成してモデルも保存
    if epoch == 0 or epoch == 9 or epoch == 24:
        generate(epoch + 1, G, log_dir)
        torch.save(G.state_dict(), os.path.join(log_dir, 'G_%03d.pth' % (epoch + 1)))
        torch.save(D.state_dict(), os.path.join(log_dir, 'D_%03d.pth' % (epoch + 1)))

# 学習履歴を保存
with open(os.path.join(log_dir, 'history.pkl'), 'wb') as f:
    pickle.dump(history, f)
```

```
cuda
torch.Size([32, 1, 28, 28])
torch.Size([32, 1])
100%|██████████| 468/468 [00:13<00:00, 35.50it/s]
  0%|          | 0/468 [00:00<?, ?it/s]
epoch 1, D_loss: 0.8255 G_loss: 1.3022
100%|██████████| 468/468 [00:13<00:00, 34.07it/s]
  1%|          | 4/468 [00:00<00:14, 32.97it/s]
epoch 2, D_loss: 1.0237 G_loss: 1.1171
100%|██████████| 468/468 [00:13<00:00, 33.71it/s]
  1%|          | 4/468 [00:00<00:13, 33.16it/s]
epoch 3, D_loss: 1.0485 G_loss: 1.0855
100%|██████████| 468/468 [00:13<00:00, 33.63it/s]
  1%|          | 4/468 [00:00<00:13, 33.36it/s]
epoch 4, D_loss: 1.0479 G_loss: 1.0947
100%|██████████| 468/468 [00:13<00:00, 33.62it/s]
  1%|          | 4/468 [00:00<00:14, 32.89it/s]
epoch 5, D_loss: 1.0379 G_loss: 1.1054
100%|██████████| 468/468 [00:13<00:00, 33.71it/s]
  1%|          | 4/468 [00:00<00:13, 34.56it/s]
epoch 6, D_loss: 1.0215 G_loss: 1.1376
100%|██████████| 468/468 [00:14<00:00, 33.10it/s]
  1%|          | 4/468 [00:00<00:13, 33.88it/s]
epoch 7, D_loss: 0.9936 G_loss: 1.1692
100%|██████████| 468/468 [00:13<00:00, 33.69it/s]
  1%|          | 4/468 [00:00<00:13, 33.27it/s]
epoch 8, D_loss: 0.9674 G_loss: 1.2082
100%|██████████| 468/468 [00:13<00:00, 33.70it/s]
  1%|          | 4/468 [00:00<00:14, 33.02it/s]
epoch 9, D_loss: 0.9452 G_loss: 1.2532
100%|██████████| 468/468 [00:13<00:00, 33.63it/s]
epoch 10, D_loss: 0.9130 G_loss: 1.2955
```

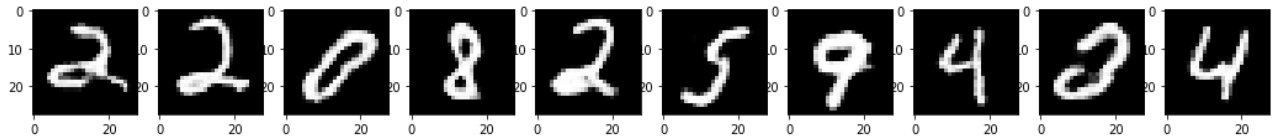
In [8]:

```
sample_z = torch.rand((64, z_dim))
sample_z = sample_z.to(device)

# Generatorでサンプル生成
with torch.no_grad():
    samples = G(sample_z).to('cpu')

samples = samples.view(64, 28, 28).detach().cpu().numpy()
```

In [9]:
```
print(samples.shape)
```

```
(64, 28, 28)
```

In [10]:
```
plt.figure(figsize=(18,18))
samples = samples[:N]
for i in range(N):
  plt.subplot(1,N,i+1); plt.imshow(samples[i], "gray")
plt.show()
```



In [ ]:
```

```