1. Generative Adversarial Network(GAN) applied to MNIST dataset.(10 pt) Train a GAN model for the MNIST dataset. A GAN model is composed of a generator and a discriminator competing with each other.

(a) (12pt) Use two multi-layer perceptions each with 4 linear layers for generator and discriminator. The input to the generator is a random vector of length 100. Use LeakyReLU with negative slope of 0.2 as your activation for the hidden layers. Use learning rate of 0.0002 and regularization technique of your choice. Train the model and generate some new image by passing in random vectors to the generator, using the plot_digits() function from last week's homework reference to visualize them.

In [1]:
```
from torch.optim import SGD, Adam
import torch.nn.functional as F
import random
from tqdm import tqdm
import math
from sklearn.model_selection import train_test_split
from torch import nn
import torch
import torch.nn as rftui90o-p=[\
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.autograd import Variable
from torchvision.utils import save_image

transform = transforms.Compose([transforms.ToTensor(),transforms.Normalize(m

train_dataset = datasets.MNIST(root='./mnist_data/', train=True, transform=t
test_dataset = datasets.MNIST(root='./mnist_data/', train=False, transform=t

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=1

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
/global/common/software/nersc/shasta2105/pytorch/1.9.0/lib/python3.8/site-p
ackages/torchvision-0.10.0a0+300a8a4-py3.8-linux-x86_64.egg/torchvision/dat
asets/mnist.py:498: UserWarning: The given NumPy array is not writeable, an
d PyTorch does not support non-writeable tensors. This means you can write
to the underlying (supposedly non-writeable) NumPy array using the tensor.
You may want to copy the array to protect its data or make it writeable bef
ore converting it to a tensor. This type of warning will be suppressed for
the rest of this program. (Triggered internally at  ../torch/csrc/utils/ten
sor_numpy.cpp:174.)
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
```

In [2]:

```
class Generator(nn.Module):
    def __init__(self, g_input_dim, g_output_dim):
        super(Generator, self).__init__()
        self.fc1 = nn.Linear(g_input_dim, 256)
        self.fc2 = nn.Linear(self.fc1.out_features, self.fc1.out_features*2)
        self.fc3 = nn.Linear(self.fc2.out_features, self.fc2.out_features*2)
        self.fc4 = nn.Linear(self.fc3.out_features, g_output_dim)

    # forward method
    def forward(self, x):
        x = F.leaky_relu(self.fc1(x), 0.2)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = F.leaky_relu(self.fc3(x), 0.2)
        return torch.tanh(self.fc4(x))

class Discriminator(nn.Module):
    def __init__(self, d_input_dim):
        super(Discriminator, self).__init__()
        self.fc1 = nn.Linear(d_input_dim, 1024)
        self.fc2 = nn.Linear(self.fc1.out_features, self.fc1.out_features//2
        self.fc3 = nn.Linear(self.fc2.out_features, self.fc2.out_features//2
        self.fc4 = nn.Linear(self.fc3.out_features, 1)

    # forward method
    def forward(self, x):
        x = F.leaky_relu(self.fc1(x), 0.2)
        x = F.dropout(x, 0.3)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = F.dropout(x, 0.3)
        x = F.leaky_relu(self.fc3(x), 0.2)
        x = F.dropout(x, 0.3)
        return torch.sigmoid(self.fc4(x))
```

In [3]:

```
# build network
d_input_dim = 784
g_input_dim = 100


G = Generator(g_input_dim = g_input_dim, g_output_dim = d_input_dim).to(devi
D = Discriminator(d_input_dim).to(device)
# loss
criterion = nn.BCELoss()
# optimizer
lr = 0.0002
G_optimizer = Adam(G.parameters(), lr = lr)
D_optimizer = Adam(D.parameters(), lr = lr)
```

In [4]:
```
def D_train(x):
    #=======================Train the discriminator=======================#

    D.zero_grad()


    # train discriminator on real
    x_real, y_real = x.view(-1, 784), torch.ones(100, 1)
    x_real, y_real = Variable(x_real).to(device), Variable(y_real).to(device

    D_output = D(x_real)
    D_real_loss = criterion(D_output, y_real)
    D_real_score = D_output

    # train discriminator on fake
    z = Variable(torch.randn(100, 100).to(device))
    x_fake, y_fake = G(z), Variable(torch.zeros(100, 1).to(device))

    D_output = D(x_fake)
    D_fake_loss = criterion(D_output, y_fake)
    D_fake_score = D_output

    # gradient backprop & optimize ONLY D's parameters
    D_loss = D_real_loss + D_fake_loss
    D_loss.backward()
    D_optimizer.step()

    return  D_loss.data.item()
def G_train(x):
    #=====================Train the generator=====================#
    G.zero_grad()


    z = Variable(torch.randn(100, 100).to(device))
    y = Variable(torch.ones(100, 1).to(device))

    G_output = G(z)
    D_output = D(G_output)
    G_loss = criterion(D_output, y)

    # gradient backprop & optimize ONLY G's parameters
    G_loss.backward()
    G_optimizer.step()

    return G_loss.data.item()
```

In [5]:
```
n_epoch = 200
for epoch in range(1, n_epoch+1):
    D_losses, G_losses = [], []
    for batch_idx, (x, _) in enumerate(train_loader):
        D_losses.append(D_train(x))
        G_losses.append(G_train(x))

    print('[%d/%d]: loss_d: %.3f, loss_g: %.3f' % (
            (epoch), n_epoch, torch.mean(torch.FloatTensor(D_losses)), torch
```

```
[1/200]: loss_d: 0.857, loss_g: 3.055
[2/200]: loss_d: 1.156, loss_g: 1.558
[3/200]: loss_d: 0.918, loss_g: 1.981
[4/200]: loss_d: 0.688, loss_g: 2.440
[5/200]: loss_d: 0.506, loss_g: 2.866
[6/200]: loss_d: 0.510, loss_g: 2.842
[7/200]: loss_d: 0.590, loss_g: 2.555
[8/200]: loss_d: 0.531, loss_g: 2.656
[9/200]: loss_d: 0.652, loss_g: 2.333
[10/200]: loss_d: 0.707, loss_g: 2.143
[11/200]: loss_d: 0.781, loss_g: 1.909
[12/200]: loss_d: 0.829, loss_g: 1.785
[13/200]: loss_d: 0.813, loss_g: 1.830
[14/200]: loss_d: 0.781, loss_g: 1.962
[15/200]: loss_d: 0.839, loss_g: 1.816
[16/200]: loss_d: 0.832, loss_g: 1.795
[17/200]: loss_d: 0.864, loss_g: 1.740
[18/200]: loss_d: 0.844, loss_g: 1.768
[19/200]: loss_d: 0.861, loss_g: 1.722
[20/200]: loss_d: 0.881, loss_g: 1.706
[21/200]: loss_d: 0.903, loss_g: 1.628
[22/200]: loss_d: 0.937, loss_g: 1.560
[23/200]: loss_d: 0.943, loss_g: 1.550
[24/200]: loss_d: 0.972, loss_g: 1.467
[25/200]: loss_d: 1.002, loss_g: 1.380
[26/200]: loss_d: 0.986, loss_g: 1.427
[27/200]: loss_d: 1.002, loss_g: 1.410
[28/200]: loss_d: 1.045, loss_g: 1.321
[29/200]: loss_d: 1.073, loss_g: 1.242
[30/200]: loss_d: 1.089, loss_g: 1.205
[31/200]: loss_d: 1.088, loss_g: 1.226
[32/200]: loss_d: 1.067, loss_g: 1.282
[33/200]: loss_d: 1.067, loss_g: 1.267
[34/200]: loss_d: 1.061, loss_g: 1.283
[35/200]: loss_d: 1.054, loss_g: 1.295
[36/200]: loss_d: 1.090, loss_g: 1.224
[37/200]: loss_d: 1.109, loss_g: 1.188
[38/200]: loss_d: 1.120, loss_g: 1.182
[39/200]: loss_d: 1.124, loss_g: 1.156
[40/200]: loss_d: 1.108, loss_g: 1.192
[41/200]: loss_d: 1.120, loss_g: 1.166
[42/200]: loss_d: 1.128, loss_g: 1.150
[43/200]: loss_d: 1.128, loss_g: 1.151
[44/200]: loss_d: 1.119, loss_g: 1.162
[45/200]: loss_d: 1.136, loss_g: 1.133
[46/200]: loss_d: 1.150, loss_g: 1.121
[47/200]: loss_d: 1.146, loss_g: 1.116
[48/200]: loss_d: 1.166, loss_g: 1.077
[49/200]: loss_d: 1.165, loss_g: 1.083
[50/200]: loss_d: 1.159, loss_g: 1.098
[51/200]: loss_d: 1.175, loss_g: 1.052
[52/200]: loss_d: 1.182, loss_g: 1.062
[53/200]: loss_d: 1.167, loss_g: 1.089
[54/200]: loss_d: 1.184, loss_g: 1.051
[55/200]: loss_d: 1.167, loss_g: 1.072
[56/200]: loss_d: 1.192, loss_g: 1.038
[57/200]: loss_d: 1.193, loss_g: 1.029
[58/200]: loss_d: 1.194, loss_g: 1.033
[59/200]: loss_d: 1.203, loss_g: 1.014
```

```
[60/200]: loss_d: 1.209, loss_g: 1.008
[61/200]: loss_d: 1.214, loss_g: 0.992
[62/200]: loss_d: 1.218, loss_g: 0.987
[63/200]: loss_d: 1.220, loss_g: 0.983
[64/200]: loss_d: 1.218, loss_g: 0.998
[65/200]: loss_d: 1.213, loss_g: 0.998
[66/200]: loss_d: 1.220, loss_g: 1.003
[67/200]: loss_d: 1.216, loss_g: 0.994
[68/200]: loss_d: 1.208, loss_g: 1.016
[69/200]: loss_d: 1.222, loss_g: 0.988
[70/200]: loss_d: 1.227, loss_g: 0.974
[71/200]: loss_d: 1.218, loss_g: 0.998
[72/200]: loss_d: 1.218, loss_g: 0.988
[73/200]: loss_d: 1.225, loss_g: 0.978
[74/200]: loss_d: 1.233, loss_g: 0.961
[75/200]: loss_d: 1.243, loss_g: 0.955
[76/200]: loss_d: 1.232, loss_g: 0.959
[77/200]: loss_d: 1.234, loss_g: 0.971
[78/200]: loss_d: 1.239, loss_g: 0.957
[79/200]: loss_d: 1.241, loss_g: 0.952
[80/200]: loss_d: 1.239, loss_g: 0.968
[81/200]: loss_d: 1.236, loss_g: 0.956
[82/200]: loss_d: 1.236, loss_g: 0.951
[83/200]: loss_d: 1.238, loss_g: 0.963
[84/200]: loss_d: 1.246, loss_g: 0.959
[85/200]: loss_d: 1.235, loss_g: 0.957
[86/200]: loss_d: 1.245, loss_g: 0.950
[87/200]: loss_d: 1.250, loss_g: 0.942
[88/200]: loss_d: 1.250, loss_g: 0.943
[89/200]: loss_d: 1.261, loss_g: 0.931
[90/200]: loss_d: 1.255, loss_g: 0.937
[91/200]: loss_d: 1.253, loss_g: 0.934
[92/200]: loss_d: 1.263, loss_g: 0.927
[93/200]: loss_d: 1.257, loss_g: 0.926
[94/200]: loss_d: 1.260, loss_g: 0.923
[95/200]: loss_d: 1.261, loss_g: 0.928
[96/200]: loss_d: 1.263, loss_g: 0.923
[97/200]: loss_d: 1.263, loss_g: 0.909
[98/200]: loss_d: 1.260, loss_g: 0.927
[99/200]: loss_d: 1.258, loss_g: 0.933
[100/200]: loss_d: 1.258, loss_g: 0.925
[101/200]: loss_d: 1.258, loss_g: 0.922
[102/200]: loss_d: 1.263, loss_g: 0.921
[103/200]: loss_d: 1.259, loss_g: 0.916
[104/200]: loss_d: 1.266, loss_g: 0.913
[105/200]: loss_d: 1.267, loss_g: 0.907
[106/200]: loss_d: 1.265, loss_g: 0.913
[107/200]: loss_d: 1.263, loss_g: 0.922
[108/200]: loss_d: 1.258, loss_g: 0.917
[109/200]: loss_d: 1.263, loss_g: 0.926
[110/200]: loss_d: 1.268, loss_g: 0.912
[111/200]: loss_d: 1.262, loss_g: 0.920
[112/200]: loss_d: 1.265, loss_g: 0.914
[113/200]: loss_d: 1.266, loss_g: 0.919
[114/200]: loss_d: 1.260, loss_g: 0.923
[115/200]: loss_d: 1.276, loss_g: 0.894
[116/200]: loss_d: 1.261, loss_g: 0.918
[117/200]: loss_d: 1.274, loss_g: 0.904
[118/200]: loss_d: 1.271, loss_g: 0.909
```

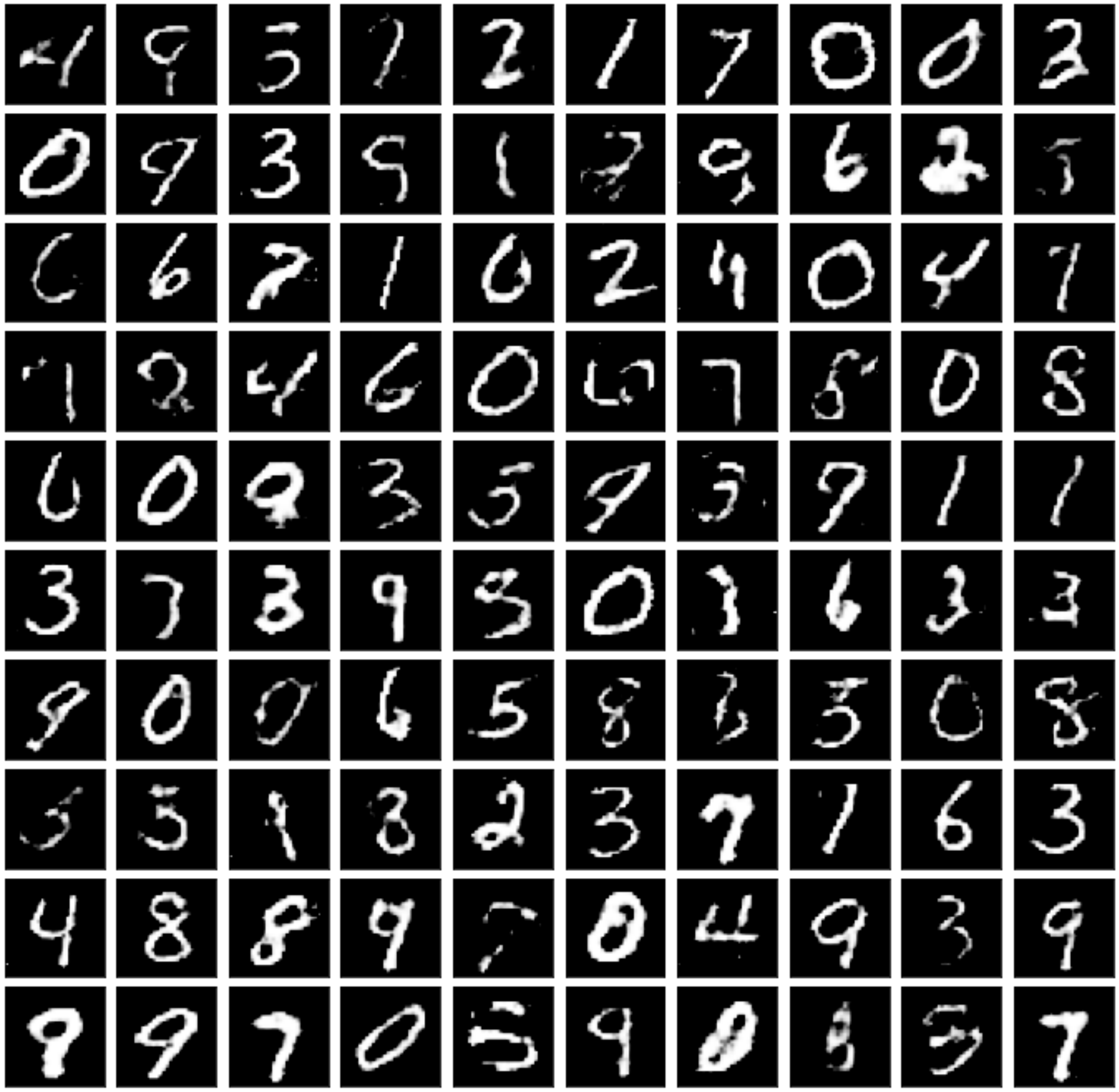
```
[119/200]: loss_d: 1.275, loss_g: 0.902
[120/200]: loss_d: 1.278, loss_g: 0.895
[121/200]: loss_d: 1.274, loss_g: 0.893
[122/200]: loss_d: 1.266, loss_g: 0.918
[123/200]: loss_d: 1.273, loss_g: 0.902
[124/200]: loss_d: 1.271, loss_g: 0.899
[125/200]: loss_d: 1.275, loss_g: 0.898
[126/200]: loss_d: 1.267, loss_g: 0.911
[127/200]: loss_d: 1.275, loss_g: 0.891
[128/200]: loss_d: 1.273, loss_g: 0.902
[129/200]: loss_d: 1.279, loss_g: 0.894
[130/200]: loss_d: 1.280, loss_g: 0.898
[131/200]: loss_d: 1.281, loss_g: 0.882
[132/200]: loss_d: 1.272, loss_g: 0.910
[133/200]: loss_d: 1.272, loss_g: 0.891
[134/200]: loss_d: 1.278, loss_g: 0.886
[135/200]: loss_d: 1.275, loss_g: 0.908
[136/200]: loss_d: 1.273, loss_g: 0.904
[137/200]: loss_d: 1.270, loss_g: 0.909
[138/200]: loss_d: 1.273, loss_g: 0.901
[139/200]: loss_d: 1.284, loss_g: 0.882
[140/200]: loss_d: 1.283, loss_g: 0.888
[141/200]: loss_d: 1.283, loss_g: 0.890
[142/200]: loss_d: 1.280, loss_g: 0.892
[143/200]: loss_d: 1.281, loss_g: 0.892
[144/200]: loss_d: 1.283, loss_g: 0.888
[145/200]: loss_d: 1.282, loss_g: 0.883
[146/200]: loss_d: 1.283, loss_g: 0.887
[147/200]: loss_d: 1.282, loss_g: 0.882
[148/200]: loss_d: 1.282, loss_g: 0.887
[149/200]: loss_d: 1.281, loss_g: 0.891
[150/200]: loss_d: 1.286, loss_g: 0.883
[151/200]: loss_d: 1.285, loss_g: 0.873
[152/200]: loss_d: 1.285, loss_g: 0.878
[153/200]: loss_d: 1.279, loss_g: 0.883
[154/200]: loss_d: 1.281, loss_g: 0.885
[155/200]: loss_d: 1.281, loss_g: 0.882
[156/200]: loss_d: 1.280, loss_g: 0.892
[157/200]: loss_d: 1.285, loss_g: 0.876
[158/200]: loss_d: 1.282, loss_g: 0.884
[159/200]: loss_d: 1.287, loss_g: 0.873
[160/200]: loss_d: 1.283, loss_g: 0.878
[161/200]: loss_d: 1.288, loss_g: 0.873
[162/200]: loss_d: 1.283, loss_g: 0.875
[163/200]: loss_d: 1.290, loss_g: 0.872
[164/200]: loss_d: 1.281, loss_g: 0.882
[165/200]: loss_d: 1.284, loss_g: 0.879
[166/200]: loss_d: 1.285, loss_g: 0.879
[167/200]: loss_d: 1.286, loss_g: 0.876
[168/200]: loss_d: 1.281, loss_g: 0.885
[169/200]: loss_d: 1.289, loss_g: 0.872
[170/200]: loss_d: 1.291, loss_g: 0.873
[171/200]: loss_d: 1.285, loss_g: 0.873
[172/200]: loss_d: 1.290, loss_g: 0.864
[173/200]: loss_d: 1.287, loss_g: 0.875
[174/200]: loss_d: 1.285, loss_g: 0.883
[175/200]: loss_d: 1.285, loss_g: 0.879
[176/200]: loss_d: 1.285, loss_g: 0.878
[177/200]: loss_d: 1.285, loss_g: 0.881
```

```
[178/200]: loss_d: 1.284, loss_g: 0.877
[179/200]: loss_d: 1.292, loss_g: 0.865
[180/200]: loss_d: 1.291, loss_g: 0.865
[181/200]: loss_d: 1.289, loss_g: 0.869
[182/200]: loss_d: 1.293, loss_g: 0.867
[183/200]: loss_d: 1.291, loss_g: 0.874
[184/200]: loss_d: 1.291, loss_g: 0.883
[185/200]: loss_d: 1.292, loss_g: 0.860
[186/200]: loss_d: 1.291, loss_g: 0.869
[187/200]: loss_d: 1.300, loss_g: 0.852
[188/200]: loss_d: 1.288, loss_g: 0.871
[189/200]: loss_d: 1.294, loss_g: 0.875
[190/200]: loss_d: 1.284, loss_g: 0.880
[191/200]: loss_d: 1.289, loss_g: 0.875
[192/200]: loss_d: 1.290, loss_g: 0.863
[193/200]: loss_d: 1.290, loss_g: 0.864
[194/200]: loss_d: 1.294, loss_g: 0.857
[195/200]: loss_d: 1.296, loss_g: 0.857
[196/200]: loss_d: 1.294, loss_g: 0.871
[197/200]: loss_d: 1.290, loss_g: 0.866
[198/200]: loss_d: 1.296, loss_g: 0.853
[199/200]: loss_d: 1.296, loss_g: 0.858
[200/200]: loss_d: 1.293, loss_g: 0.885
```

In [14]:

```
import matplotlib.pyplot as plt
def plot_digits(data):
    fig, ax = plt.subplots(10, 10, figsize=(12, 12),
                           subplot_kw=dict(xticks=[], yticks=[]))
    fig.subplots_adjust(hspace=0.1, wspace=0.1)
    for i, axi in enumerate(ax.flat):
        im = axi.imshow(data[i].reshape(28, 28), cmap=plt.get_cmap('gray'))
        im.set_clim(0, 1)

with torch.no_grad():
    z = Variable(torch.randn(100, 100).to(device))
    generated_img = G(z).detach().cpu().numpy()
    plot_digits(generated_img)
```



(b) (8pt) Use two CNNs each with 4 convolutional blocks for generator and discriminator. Each convolutional block is composed of a convolution layer, a batch normalization and a LeakyReLU with negative slope of 0.2 activation function. The input to the generator is a random vector of length 100. Use learning rate of 0.0002. Train the model and generate some new image by passing in random vectors to the generator, using the plot_digits() function from last week's homework reference to visualize them. Does the generated image look more like real image

In [22]:
```
class Generator(nn.Module):
    def __init__(self, in_ch=100,kernel_size=4,out=64):
        super(Generator, self).__init__()
        self.gen = nn.Sequential(
                                nn.ConvTranspose2d(in_ch, 4 * out, kernel_si
                                nn.BatchNorm2d(4 * out),
                                nn.ReLU(),
                                nn.ConvTranspose2d(4 * out, 2 * out, kernel_
                                nn.BatchNorm2d(2 * out),
                                nn.ReLU(),
                                nn.ConvTranspose2d(2 * out, 1 * out,kernel_s
                                nn.BatchNorm2d(1 * out),
                                nn.ReLU(),
                                nn.ConvTranspose2d(out, 1,kernel_size,stride
                                nn.Tanh(),
                                )

    def forward(self, z):
        z = z.view(-1, 100, 1, 1)
        return self.gen(z)


class Discriminator(nn.Module):
    def __init__(self,out=64,kernel_size = 4):
        super(Discriminator, self).__init__()
        self.dis = nn.Sequential(
            nn.Conv2d(1, out, kernel_size, stride=2,padding=3, bias=False),
            nn.LeakyReLU(0.2),
            nn.Conv2d(out, 2 * out, kernel_size, stride=2, padding=1, bias=F
            nn.BatchNorm2d(2 * out),
            nn.LeakyReLU(0.2),
            nn.Conv2d(2*out, 4 *out,kernel_size, stride=2, padding=1, bias=F
            nn.BatchNorm2d(4 *out),
            nn.LeakyReLU(0.2),
            nn.Conv2d(4*out, 1,kernel_size, stride=1, padding=0, bias=False)
            nn.Sigmoid()

        )

    def forward(self, x):
        x = x.view(-1, 1, 28,28)
        return self.dis(x).view(-1)
```

In [23]:
```
# build network
G = Generator().to(device)
D = Discriminator().to(device)
# loss
criterion = nn.BCELoss()
# optimizer
lr = 0.0002
G_optimizer = Adam(G.parameters(), lr = lr)
D_optimizer = Adam(D.parameters(), lr = lr)
```

In [26]:
```
def D_train(x):
    #======================Train the discriminator======================#


    real_label = 1
    fake_label = 1- real_label
    num_shape = x.shape[0]

    ## update the discriminator
    D_output = D(x).to(device)
    y_real = torch.full((num_shape,), real_label, device=device)
    D_real_loss = criterion(D_output.to(torch.float), y_real.to(torch.float)

    z = torch.randn(num_shape, 100,1,1,device=device)

    D_fake_outputs = G(z).detach()
    D_fake = D(D_fake_outputs)
    y_fake = torch.full((num_shape,), fake_label, device=device)
    D_fake_loss = criterion(D_fake.to(torch.float), y_fake.to(torch.float))

    D.zero_grad()
    D_loss = D_real_loss + D_fake_loss
    D_loss.backward()
    D_optimizer.step()
    return D_loss.data.item()

def G_train(x):
    #======================Train the generator======================#
    ## update the generator

    real_label = 1
    fake_label = 1- real_label
    num_shape = x.shape[0]
    z = torch.randn(num_shape, 100, 1, 1, device=device)


    D_output_ = D(G(z))
    fake_labels = torch.full((num_shape,), real_label, device=device)
    G_loss = criterion(D_output_.to(torch.float), fake_labels.to(torch.float

    G.zero_grad()
    G_loss.backward()
    G_optimizer.step()

    return G_loss.data.item()
```

In [27]:
```
n_epoch = 200
for epoch in range(1, n_epoch+1):
    D_losses, G_losses = [], []
    for batch_idx, (x, _) in enumerate(train_loader):
        x = x.to(device)

        D_losses.append(D_train(x))
        G_losses.append(G_train(x))

    print('[%d/%d]: loss_d: %.3f, loss_g: %.3f' % (
            (epoch), n_epoch, torch.mean(torch.FloatTensor(D_losses)), torch
```

```
[1/200]: loss_d: 0.017, loss_g: 7.797
[2/200]: loss_d: 0.040, loss_g: 7.690
[3/200]: loss_d: 0.027, loss_g: 7.026
[4/200]: loss_d: 0.036, loss_g: 7.794
[5/200]: loss_d: 0.071, loss_g: 6.183
[6/200]: loss_d: 0.180, loss_g: 4.766
[7/200]: loss_d: 0.238, loss_g: 4.215
[8/200]: loss_d: 0.262, loss_g: 3.729
[9/200]: loss_d: 0.310, loss_g: 3.370
[10/200]: loss_d: 0.333, loss_g: 3.329
[11/200]: loss_d: 0.288, loss_g: 3.205
[12/200]: loss_d: 0.302, loss_g: 3.258
[13/200]: loss_d: 0.274, loss_g: 3.211
[14/200]: loss_d: 0.385, loss_g: 3.165
[15/200]: loss_d: 0.342, loss_g: 3.040
[16/200]: loss_d: 0.333, loss_g: 2.964
[17/200]: loss_d: 0.379, loss_g: 2.943
[18/200]: loss_d: 0.431, loss_g: 2.865
[19/200]: loss_d: 0.408, loss_g: 2.807
[20/200]: loss_d: 0.385, loss_g: 2.741
[21/200]: loss_d: 0.366, loss_g: 2.766
[22/200]: loss_d: 0.431, loss_g: 2.767
[23/200]: loss_d: 0.436, loss_g: 2.772
[24/200]: loss_d: 0.432, loss_g: 2.740
[25/200]: loss_d: 0.396, loss_g: 2.835
[26/200]: loss_d: 0.447, loss_g: 2.735
[27/200]: loss_d: 0.374, loss_g: 2.670
[28/200]: loss_d: 0.427, loss_g: 2.812
[29/200]: loss_d: 0.437, loss_g: 2.847
[30/200]: loss_d: 0.401, loss_g: 2.697
[31/200]: loss_d: 0.379, loss_g: 2.823
[32/200]: loss_d: 0.401, loss_g: 2.869
[33/200]: loss_d: 0.384, loss_g: 2.853
[34/200]: loss_d: 0.385, loss_g: 2.918
[35/200]: loss_d: 0.408, loss_g: 2.907
[36/200]: loss_d: 0.380, loss_g: 2.890
[37/200]: loss_d: 0.365, loss_g: 2.986
[38/200]: loss_d: 0.347, loss_g: 2.916
[39/200]: loss_d: 0.397, loss_g: 3.025
[40/200]: loss_d: 0.352, loss_g: 3.007
[41/200]: loss_d: 0.391, loss_g: 3.041
[42/200]: loss_d: 0.352, loss_g: 3.016
[43/200]: loss_d: 0.355, loss_g: 3.191
[44/200]: loss_d: 0.369, loss_g: 3.013
[45/200]: loss_d: 0.336, loss_g: 3.130
[46/200]: loss_d: 0.316, loss_g: 3.198
[47/200]: loss_d: 0.320, loss_g: 3.160
[48/200]: loss_d: 0.329, loss_g: 3.186
[49/200]: loss_d: 0.364, loss_g: 3.189
[50/200]: loss_d: 0.310, loss_g: 3.158
[51/200]: loss_d: 0.307, loss_g: 3.149
[52/200]: loss_d: 0.318, loss_g: 3.270
[53/200]: loss_d: 0.315, loss_g: 3.273
[54/200]: loss_d: 0.326, loss_g: 3.290
[55/200]: loss_d: 0.308, loss_g: 3.311
[56/200]: loss_d: 0.307, loss_g: 3.254
[57/200]: loss_d: 0.279, loss_g: 3.354
[58/200]: loss_d: 0.329, loss_g: 3.294
[59/200]: loss_d: 0.293, loss_g: 3.398
```

```
[60/200]: loss_d: 0.271, loss_g: 3.365
[61/200]: loss_d: 0.294, loss_g: 3.431
[62/200]: loss_d: 0.312, loss_g: 3.370
[63/200]: loss_d: 0.256, loss_g: 3.430
[64/200]: loss_d: 0.282, loss_g: 3.494
[65/200]: loss_d: 0.320, loss_g: 3.417
[66/200]: loss_d: 0.281, loss_g: 3.384
[67/200]: loss_d: 0.238, loss_g: 3.503
[68/200]: loss_d: 0.294, loss_g: 3.468
[69/200]: loss_d: 0.291, loss_g: 3.516
[70/200]: loss_d: 0.250, loss_g: 3.531
[71/200]: loss_d: 0.257, loss_g: 3.498
[72/200]: loss_d: 0.265, loss_g: 3.684
[73/200]: loss_d: 0.275, loss_g: 3.523
[74/200]: loss_d: 0.244, loss_g: 3.593
[75/200]: loss_d: 0.295, loss_g: 3.580
[76/200]: loss_d: 0.249, loss_g: 3.654
[77/200]: loss_d: 0.243, loss_g: 3.616
[78/200]: loss_d: 0.254, loss_g: 3.699
[79/200]: loss_d: 0.243, loss_g: 3.706
[80/200]: loss_d: 0.248, loss_g: 3.702
[81/200]: loss_d: 0.291, loss_g: 3.718
[82/200]: loss_d: 0.229, loss_g: 3.707
[83/200]: loss_d: 0.258, loss_g: 3.700
[84/200]: loss_d: 0.253, loss_g: 3.672
[85/200]: loss_d: 0.234, loss_g: 3.732
[86/200]: loss_d: 0.212, loss_g: 3.807
[87/200]: loss_d: 0.260, loss_g: 3.700
[88/200]: loss_d: 0.220, loss_g: 3.864
[89/200]: loss_d: 0.236, loss_g: 3.815
[90/200]: loss_d: 0.243, loss_g: 3.895
[91/200]: loss_d: 0.212, loss_g: 3.842
[92/200]: loss_d: 0.264, loss_g: 3.819
[93/200]: loss_d: 0.197, loss_g: 3.909
[94/200]: loss_d: 0.223, loss_g: 3.864
[95/200]: loss_d: 0.210, loss_g: 3.979
[96/200]: loss_d: 0.239, loss_g: 3.875
[97/200]: loss_d: 0.283, loss_g: 3.867
[98/200]: loss_d: 0.222, loss_g: 3.837
[99/200]: loss_d: 0.224, loss_g: 3.862
[100/200]: loss_d: 0.209, loss_g: 4.006
[101/200]: loss_d: 0.184, loss_g: 4.002
[102/200]: loss_d: 0.230, loss_g: 4.071
[103/200]: loss_d: 0.172, loss_g: 4.057
[104/200]: loss_d: 0.242, loss_g: 3.916
[105/200]: loss_d: 0.190, loss_g: 4.040
[106/200]: loss_d: 0.191, loss_g: 4.187
[107/200]: loss_d: 0.192, loss_g: 4.171
[108/200]: loss_d: 0.166, loss_g: 4.125
[109/200]: loss_d: 0.187, loss_g: 4.194
[110/200]: loss_d: 0.222, loss_g: 4.109
[111/200]: loss_d: 0.206, loss_g: 4.099
[112/200]: loss_d: 0.185, loss_g: 4.219
[113/200]: loss_d: 0.217, loss_g: 4.124
[114/200]: loss_d: 0.180, loss_g: 4.152
[115/200]: loss_d: 0.194, loss_g: 4.128
[116/200]: loss_d: 0.207, loss_g: 4.186
[117/200]: loss_d: 0.169, loss_g: 4.264
[118/200]: loss_d: 0.181, loss_g: 4.357
```

```
[119/200]: loss_d: 0.169, loss_g: 4.294
[120/200]: loss_d: 0.188, loss_g: 4.346
[121/200]: loss_d: 0.221, loss_g: 4.333
[122/200]: loss_d: 0.155, loss_g: 4.241
[123/200]: loss_d: 0.160, loss_g: 4.428
[124/200]: loss_d: 0.191, loss_g: 4.354
[125/200]: loss_d: 0.177, loss_g: 4.360
[126/200]: loss_d: 0.200, loss_g: 4.229
[127/200]: loss_d: 0.141, loss_g: 4.458
[128/200]: loss_d: 0.211, loss_g: 4.475
[129/200]: loss_d: 0.154, loss_g: 4.381
[130/200]: loss_d: 0.144, loss_g: 4.588
[131/200]: loss_d: 0.184, loss_g: 4.355
[132/200]: loss_d: 0.168, loss_g: 4.434
[133/200]: loss_d: 0.172, loss_g: 4.418
[134/200]: loss_d: 0.140, loss_g: 4.549
[135/200]: loss_d: 0.157, loss_g: 4.566
[136/200]: loss_d: 0.183, loss_g: 4.493
[137/200]: loss_d: 0.144, loss_g: 4.582
[138/200]: loss_d: 0.140, loss_g: 4.591
[139/200]: loss_d: 0.178, loss_g: 4.513
[140/200]: loss_d: 0.194, loss_g: 4.488
[141/200]: loss_d: 0.145, loss_g: 4.616
[142/200]: loss_d: 0.168, loss_g: 4.471
[143/200]: loss_d: 0.170, loss_g: 4.583
[144/200]: loss_d: 0.149, loss_g: 4.686
[145/200]: loss_d: 0.123, loss_g: 4.695
[146/200]: loss_d: 0.139, loss_g: 4.624
[147/200]: loss_d: 0.156, loss_g: 4.680
[148/200]: loss_d: 0.188, loss_g: 4.550
[149/200]: loss_d: 0.121, loss_g: 4.761
[150/200]: loss_d: 0.161, loss_g: 4.631
[151/200]: loss_d: 0.161, loss_g: 4.724
[152/200]: loss_d: 0.132, loss_g: 4.764
[153/200]: loss_d: 0.159, loss_g: 4.740
[154/200]: loss_d: 0.127, loss_g: 4.799
[155/200]: loss_d: 0.146, loss_g: 4.752
[156/200]: loss_d: 0.151, loss_g: 4.807
[157/200]: loss_d: 0.155, loss_g: 4.647
[158/200]: loss_d: 0.152, loss_g: 4.869
[159/200]: loss_d: 0.129, loss_g: 4.827
[160/200]: loss_d: 0.170, loss_g: 4.765
[161/200]: loss_d: 0.113, loss_g: 4.884
[162/200]: loss_d: 0.127, loss_g: 4.872
[163/200]: loss_d: 0.132, loss_g: 4.906
[164/200]: loss_d: 0.160, loss_g: 4.798
[165/200]: loss_d: 0.151, loss_g: 4.770
[166/200]: loss_d: 0.123, loss_g: 4.993
[167/200]: loss_d: 0.126, loss_g: 4.884
[168/200]: loss_d: 0.162, loss_g: 4.948
[169/200]: loss_d: 0.132, loss_g: 4.842
[170/200]: loss_d: 0.124, loss_g: 5.053
[171/200]: loss_d: 0.126, loss_g: 5.017
[172/200]: loss_d: 0.151, loss_g: 4.943
[173/200]: loss_d: 0.120, loss_g: 4.890
[174/200]: loss_d: 0.081, loss_g: 5.233
[175/200]: loss_d: 0.146, loss_g: 5.032
[176/200]: loss_d: 0.176, loss_g: 4.917
[177/200]: loss_d: 0.102, loss_g: 4.961
```

```
[178/200]: loss_d: 0.125, loss_g: 5.072
[179/200]: loss_d: 0.158, loss_g: 5.005
[180/200]: loss_d: 0.130, loss_g: 4.922
[181/200]: loss_d: 0.097, loss_g: 5.196
[182/200]: loss_d: 0.163, loss_g: 4.876
[183/200]: loss_d: 0.085, loss_g: 5.220
[184/200]: loss_d: 0.186, loss_g: 4.949
[185/200]: loss_d: 0.091, loss_g: 5.101
[186/200]: loss_d: 0.130, loss_g: 5.167
[187/200]: loss_d: 0.113, loss_g: 5.195
[188/200]: loss_d: 0.105, loss_g: 5.202
[189/200]: loss_d: 0.156, loss_g: 5.092
[190/200]: loss_d: 0.130, loss_g: 5.097
[191/200]: loss_d: 0.089, loss_g: 5.228
[192/200]: loss_d: 0.125, loss_g: 5.180
[193/200]: loss_d: 0.111, loss_g: 5.114
[194/200]: loss_d: 0.116, loss_g: 5.195
[195/200]: loss_d: 0.142, loss_g: 5.183
[196/200]: loss_d: 0.087, loss_g: 5.403
[197/200]: loss_d: 0.128, loss_g: 5.205
[198/200]: loss_d: 0.100, loss_g: 5.283
[199/200]: loss_d: 0.112, loss_g: 5.287
[200/200]: loss_d: 0.161, loss_g: 5.001
```

In [28]:

```
def plot_digits(data):
    fig, ax = plt.subplots(10, 10, figsize=(12, 12),
                           subplot_kw=dict(xticks=[], yticks=[]))
    fig.subplots_adjust(hspace=0.1, wspace=0.1)
    for i, axi in enumerate(ax.flat):
        im = axi.imshow(data[i].reshape(28, 28), cmap=plt.get_cmap('gray'))
        im.set_clim(0, 1)

with torch.no_grad():
    z = Variable(torch.randn(100, 100).to(device))
    generated_img = G(z).detach().cpu().numpy()
    plot_digits(generated_img)
```