dlnd_face_generation

February 10, 2019

1 Face Generation

In this project, you'll define and train a DCGAN on a dataset of faces. Your goal is to get a generator network to generate *new* images of faces that look as realistic as possible!

The project will be broken down into a series of tasks from **loading in data to defining and training adversarial networks**. At the end of the notebook, you'll be able to visualize the results of your trained Generator to see how it performs; your generated samples should look like fairly realistic faces with small amounts of noise.

1.0.1 Get the Data

You'll be using the CelebFaces Attributes Dataset (CelebA) to train your adversarial networks.

This dataset is more complex than the number datasets (like MNIST or SVHN) you've been working with, and so, you should prepare to define deeper networks and train them for a longer time to get good results. It is suggested that you utilize a GPU for training.

1.0.2 Pre-processed Data

Since the project's main focus is on building the GANs, we've done *some* of the pre-processing for you. Each of the CelebA images has been cropped to remove parts of the image that don't include a face, then resized down to 64x64x3 NumPy images. Some sample data is show below.

If you are working locally, you can download this data by clicking here

This is a zip file that you'll need to extract in the home directory of this notebook for further loading and processing. After extracting the data, you should be left with a directory of data `processed_celeba_small/`

```
In [1]: # can comment out after executing
        !unzip processed_celeba_small.zip
```

```
unzip:  cannot find or open processed_celeba_small.zip, processed_celeba_small.zip.zip or proce
```

```
In [1]: data_dir = 'processed_celeba_small/'

        """
        DON'T MODIFY ANYTHING IN THIS CELL
```

```
"""
import pickle as pkl
import matplotlib.pyplot as plt
import numpy as np
import problem_unittests as tests
#import helper

%matplotlib inline
```

1.1 Visualize the CelebA Data

The CelebA dataset contains over 200,000 celebrity images with annotations. Since you're going to be generating faces, you won't need the annotations, you'll only need the images. Note that these are color images with 3 color channels (RGB) each.

1.1.1 Pre-process and Load the Data

Since the project's main focus is on building the GANs, we've done *some* of the pre-processing for you. Each of the CelebA images has been cropped to remove parts of the image that don't include a face, then resized down to 64x64x3 NumPy images. This *pre-processed* dataset is a smaller subset of the very large CelebA data.

> There are a few other steps that you'll need to **transform** this data and create a **DataLoader**.

Exercise: Complete the following `get_dataloader` function, such that it satisfies these requirements:

- Your images should be square, Tensor images of size `image_size x image_size` in the x and y dimension.
- Your function should return a DataLoader that shuffles and batches these Tensor images.

ImageFolder To create a dataset given a directory of images, it's recommended that you use PyTorch's ImageFolder wrapper, with a root directory `processed_celeba_small/` and data transformation passed in.

In [2]:
```
# necessary imports
import torch
from torchvision import datasets
from torchvision import transforms
```

In [3]:
```
def get_dataloader(batch_size, image_size, data_dir='processed_celeba_small/'):
    """
    Batch the neural network data using DataLoader
    :param batch_size: The size of each batch; the number of images in a batch
    :param img_size: The square size of the image data (x, y)
    :param data_dir: Directory where image data is located
    :return: DataLoader with batched data
    """
```

```
    # TODO: Implement function and return a dataloader

    image_transforms = transforms.Compose([transforms.Resize(image_size),
                                           transforms.ToTensor(),
                                          ])
    dataloader = torch.utils.data.DataLoader(datasets.ImageFolder(data_dir, transform=
                                             , shuffle=True, batch_size=batch_size)
    return dataloader
```

1.2 Create a DataLoader

Exercise: Create a DataLoader `celeba_train_loader` with appropriate hyperparameters. Call the above function and create a dataloader to view images. * You can decide on any reasonable `batch_size` parameter * Your `image_size` **must be** `32`. Resizing the data to a smaller size will make for faster training, while still creating convincing images of faces!

In [4]:
```
# Define function hyperparameters
batch_size = 20
img_size = 32

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
# Call your function and get a dataloader
celeba_train_loader = get_dataloader(batch_size, img_size)
```

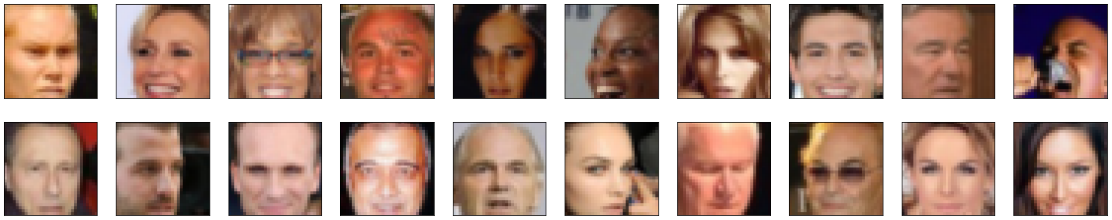
Next, you can view some images! You should seen square images of somewhat-centered faces.

Note: You'll need to convert the Tensor images into a NumPy type and transpose the dimensions to correctly display an image, suggested `imshow` code is below, but it may not be perfect.

In [5]:
```
# helper display function
def imshow(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
# obtain one batch of training images
dataiter = iter(celeba_train_loader)
images, _ = dataiter.next() # _ for no labels

# plot the images in the batch, along with the corresponding labels
fig = plt.figure(figsize=(20, 4))
plot_size=20
for idx in np.arange(plot_size):
    ax = fig.add_subplot(2, plot_size/2, idx+1, xticks=[], yticks=[])
    imshow(images[idx])
```



Exercise: Pre-process your image data and scale it to a pixel range of -1 to 1 You need to do a bit of pre-processing; you know that the output of a `tanh` activated generator will contain pixel values in a range from -1 to 1, and so, we need to rescale our training images to a range of -1 to 1. (Right now, they are in a range from 0-1.)

```
In [6]: # TODO: Complete the scale function
        def scale(x, feature_range=(-1, 1)):
            ''' Scale takes in an image x and returns that image, scaled
               with a feature_range of pixel values from -1 to 1.
               This function assumes that the input x is already scaled from 0-1.'''
            # assume x is scaled to (0, 1)
            # scale to feature_range and return scaled x

            return x*(feature_range[1] - feature_range[0]) + feature_range[0]
```

```
In [7]: """
        DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
        """
        # check scaled range
        # should be close to -1 to 1
        img = images[0]
        scaled_img = scale(img)

        print('Min: ', scaled_img.min())
        print('Max: ', scaled_img.max())
```

```
Min:  tensor(-0.9059)
Max:  tensor(0.8431)
```

---

2 Define the Model

A GAN is comprised of two adversarial networks, a discriminator and a generator.

2.1 Discriminator

Your first task will be to define the discriminator. This is a convolutional classifier like you've built before, only without any maxpooling layers. To deal with this complex data, it's suggested you use a deep network with **normalization**. You are also allowed to create any helper functions that may be useful.

Exercise: Complete the Discriminator class

- The inputs to the discriminator are 32x32x3 tensor images
- The output should be a single value that will indicate whether a given image is real or fake

```
In [8]: import torch.nn as nn
        import torch.nn.functional as F

In [9]: class Discriminator(nn.Module):

            def __init__(self, conv_dim):
                """
                Initialize the Discriminator Module
                :param conv_dim: The depth of the first convolutional layer
                """
                super(Discriminator, self).__init__()

                # complete init function
                self.conv_dim = conv_dim
                self.conv1 = nn.Conv2d(3, conv_dim, kernel_size=4, stride=2, padding=1, bias=Fa
                self.batch_norm1 = nn.BatchNorm2d(conv_dim)
                self.conv2 = nn.Conv2d(conv_dim, conv_dim*2,kernel_size=4, stride=2, padding=1
                self.batch_norm2 = nn.BatchNorm2d(conv_dim*2)
                self.conv3 = nn.Conv2d(conv_dim*2, conv_dim*4, kernel_size=4, stride=2, padding
                self.batch_norm3 = nn.BatchNorm2d(conv_dim*4)
                self.conv4 = nn.Conv2d(conv_dim*4, conv_dim*8, kernel_size=4, stride=2, padding
                self.batch_norm4 = nn.BatchNorm2d(conv_dim*8)
                self.conv5 = nn.Conv2d(conv_dim*8, conv_dim*16, kernel_size=4, stride=2, paddi
                self.fc = nn.Linear(conv_dim*4*4, 1)



            def forward(self, x):
                """
                Forward propagation of the neural network
                :param x: The input to the neural network
                :return: Discriminator logits; the output of the neural network
                """
                # define feedforward behavior
                x = F.leaky_relu(self.batch_norm1(self.conv1(x)), 0.2)
                x = F.leaky_relu(self.batch_norm2(self.conv2(x)), 0.2)
                x = F.leaky_relu(self.batch_norm3(self.conv3(x)), 0.2)
```

```
            x = F.leaky_relu(self.batch_norm4(self.conv4(x)), 0.2)
            x = self.conv5(x)
            # flatten
            x = x.view(-1, self.conv_dim*4*4)
            # final output layer
            x = F.sigmoid(self.fc(x))
            return x


        """
        DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
        """
        tests.test_discriminator(Discriminator)
```

```
Tests Passed
```

```
/home/andreili/anaconda3/envs/project_2/lib/python3.7/site-packages/torch/nn/functional.py:1332
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
```

2.2 Generator

The generator should upsample an input and generate a *new* image of the same size as our training data 32x32x3. This should be mostly transpose convolutional layers with normalization applied to the outputs.

Exercise: Complete the Generator class

- The inputs to the generator are vectors of some length `z_size`
- The output should be a image of shape 32x32x3

```
In [10]: class Generator(nn.Module):

            def __init__(self, z_size, conv_dim):
                """
                Initialize the Generator Module
                :param z_size: The length of the input latent vector, z
                :param conv_dim: The depth of the inputs to the *last* transpose convolutiona
                """
                super(Generator, self).__init__()

                # complete init function
                self.conv_dim = conv_dim
                self.t_conv1 = nn.ConvTranspose2d(conv_dim, conv_dim*8, kernel_size=4, stride=
                self.batch_norm1 = nn.BatchNorm2d(conv_dim*8)
                self.t_conv2 = nn.ConvTranspose2d(conv_dim*8, conv_dim*4, kernel_size=4, strid
                self.batch_norm2 = nn.BatchNorm2d(conv_dim*4)
                self.t_conv3 = nn.ConvTranspose2d(conv_dim*4, conv_dim*2, kernel_size=4, strid
```

```
            self.batch_norm3 = nn.BatchNorm2d(conv_dim*2)
            self.t_conv4 = nn.ConvTranspose2d(conv_dim*2, 3, kernel_size=4, stride=2, pad
            self.fc = nn.Linear(z_size, conv_dim*4)
            print('z_size', z_size)




        def forward(self, x):
            """
            Forward propagation of the neural network
            :param x: The input to the neural network
            :return: A 32x32x3 Tensor image as output
            """
            # define feedforward behavior
            batch_s = x.shape[0]
            x = self.fc(x)
            x = x.view(batch_s, self.conv_dim, 2, 2)
            x = F.relu(self.batch_norm1(self.t_conv1(x)))
            x = F.relu(self.batch_norm2(self.t_conv2(x)))
            x = F.relu(self.batch_norm3(self.t_conv3(x)))
            x = self.t_conv4(x)
            x = F.tanh(x)


            return x

    """
    DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
    """
    tests.test_generator(Generator)
```

```
z_size 25
Tests Passed
```

```
/home/andreili/anaconda3/envs/project_2/lib/python3.7/site-packages/torch/nn/functional.py:132
  warnings.warn("nn.functional.tanh is deprecated. Use torch.tanh instead.")
```

2.3 Initialize the weights of your networks

To help your models converge, you should initialize the weights of the convolutional and linear layers in your model. From reading the original DCGAN paper, they say: > All weights were initialized from a zero-centered Normal distribution with standard deviation 0.02.

So, your next task will be to define a weight initialization function that does just this!

You can refer back to the lesson on weight initialization or even consult existing model code, such as that from the `networks.py` file in CycleGAN Github repository to help you complete this function.

Exercise: Complete the weight initialization function

- This should initialize only **convolutional** and **linear** layers
- Initialize the weights to a normal distribution, centered around 0, with a standard deviation of 0.02.
- The bias terms, if they exist, may be left alone or set to 0.

```
In [11]: def weights_init_normal(m):
             """
             Applies initial weights to certain layers in a model .
             The weights are taken from a normal distribution
             with mean = 0, std dev = 0.02.
             :param m: A module or layer in a network
             """
             # classname will be something like:
             # `Conv`, `BatchNorm2d`, `Linear`, etc.
             classname = m.__class__.__name__

             # TODO: Apply initial weights to convolutional and linear layers
             if hasattr(m, 'weight') and (classname.find('Conv') != -1 or classname.find('Linea
                 nn.init.normal_(m.weight.data, 0.0, 0.02)
                 if  hasattr(m.bias, 'data'):
                     nn.init.constant_(m.bias.data, 0.0)
```

2.4 Build complete network

Define your models' hyperparameters and instantiate the discriminator and generator from the classes defined above. Make sure you've passed in the correct input arguments.

```
In [12]: """
         DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
         """
         def build_network(d_conv_dim, g_conv_dim, z_size):
             # define discriminator and generator
             D = Discriminator(d_conv_dim)
             G = Generator(z_size=z_size, conv_dim=g_conv_dim)

             # initialize model weights
             D.apply(weights_init_normal)
             G.apply(weights_init_normal)

             print(D)
             print()
             print(G)

             return D, G
```

Exercise: Define model hyperparameters

```
In [13]: # Define model hyperparams
         d_conv_dim = 32
         g_conv_dim = 32
         z_size = 100

         """
         DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
         """
         D, G = build_network(d_conv_dim, g_conv_dim, z_size)
```

```
z_size 100
Discriminator(
  (conv1): Conv2d(3, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (batch_norm1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
  (conv2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (batch_norm2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
  (conv3): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (batch_norm3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
  (conv4): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (batch_norm4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
  (conv5): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (fc): Linear(in_features=512, out_features=1, bias=True)
)

Generator(
  (t_conv1): ConvTranspose2d(32, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=
  (batch_norm1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
  (t_conv2): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=
  (batch_norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
  (t_conv3): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=
  (batch_norm3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
  (t_conv4): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=Fal
  (fc): Linear(in_features=100, out_features=128, bias=True)
)
```

2.4.1 Training on GPU

Check if you can train on GPU. Here, we'll set this as a boolean variable `train_on_gpu`. Later, you'll be responsible for making sure that >* Models, * Model inputs, and * Loss function arguments

Are moved to GPU, where appropriate.

```
In [14]: """
         DON'T MODIFY ANYTHING IN THIS CELL
         """
         import torch
```

```
        # Check for a GPU
        train_on_gpu = torch.cuda.is_available()
        if not train_on_gpu:
            print('No GPU found. Please use a GPU to train your neural network.')
        else:
            print('Training on GPU!')
```

```
Training on GPU!
```

2.5 Discriminator and Generator Losses

Now we need to calculate the losses for both types of adversarial networks.

2.5.1 Discriminator Losses

- For the discriminator, the total loss is the sum of the losses for real and fake images, `d_loss = d_real_loss + d_fake_loss`.
- Remember that we want the discriminator to output 1 for real images and 0 for fake images, so we need to set up the losses to reflect that.

2.5.2 Generator Loss

The generator loss will look similar only with flipped labels. The generator's goal is to get the discriminator to *think* its generated images are *real*.

Exercise: Complete real and fake loss functions **You may choose to use either cross entropy or a least squares error loss to complete the following `real_loss` and `fake_loss` functions.**

```
In [15]: import random
         def real_loss(D_out, smooth=False):
             batch_size = D_out.size(0)
             # label smoothing
             if smooth:
                 # smooth, real labels = 0.9
                 labels = torch.ones(batch_size)*0.9
             else:
                 labels = torch.ones(batch_size) # real labels = 1
             # move labels to GPU if available
             if train_on_gpu:
                 labels = labels.cuda()
             # binary cross entropy with logits loss
             criterion = nn.BCELoss()
             # calculate loss
             loss = criterion(D_out.squeeze(), labels)
             return loss
```

```
def fake_loss(D_out):
    batch_size = D_out.size(0)
    labels = torch.zeros(batch_size)
    if train_on_gpu:
        labels = labels.cuda()
    criterion = nn.BCELoss()
    # calculate loss
    loss = criterion(D_out.squeeze(), labels)
    return loss
```

2.6 Optimizers

Exercise: Define optimizers for your Discriminator (D) and Generator (G) Define optimizers for your models with appropriate hyperparameters.

```
In [16]: import torch.optim as optim

         lr=0.0005
         g_optimizer = optim.Adam(G.parameters(), lr=lr, betas=(0.3, 0.999))
         d_optimizer = optim.Adam(D.parameters(), lr=lr, betas=(0.3, 0.999))
```

2.7 Training

Training will involve alternating between training the discriminator and the generator. You'll use your functions `real_loss` and `fake_loss` to help you calculate the discriminator losses.

- You should train the discriminator by alternating on real and fake images
- Then the generator, which tries to trick the discriminator and should have an opposing loss function

Saving Samples You've been given some code to print out some loss statistics and save some generated "fake" samples.

Exercise: Complete the training function Keep in mind that, if you've moved your models to GPU, you'll also have to move any model inputs to GPU.

```
In [17]: def train(D, G, n_epochs, print_every=50):
             '''Trains adversarial networks for some number of epochs
                param, D: the discriminator network
                param, G: the generator network
                param, n_epochs: number of epochs to train for
                param, print_every: when to print and record the models' losses
                return: D and G losses'''

             # move models to GPU
```

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

# keep track of loss and generated, "fake" samples
samples = []
losses = []

# Get some fixed data for sampling. These are images that are held
# constant throughout training, and allow us to inspect the model's performance
sample_size=16
fixed_z = np.random.uniform(-1, 1, size=(sample_size, z_size))
fixed_z = torch.from_numpy(fixed_z).float()
# move z to GPU if available
if train_on_gpu:
    fixed_z = fixed_z.cuda()

# epoch training loop
for epoch in range(n_epochs):

    # batch training loop
    for batch_i, (real_images, _) in enumerate(celeba_train_loader):

        batch_size = real_images.size(0)
        real_images = scale(real_images)

        # ===============================================
        #         YOUR CODE HERE: TRAIN THE NETWORKS
        # ===============================================
        # 1. Train the discriminator on real and fake images
        if train_on_gpu:
            real_images = real_images.cuda()
        d_optimizer.zero_grad()
        D_real = D(real_images)
        d_real_loss = real_loss(D_real)
        z_flex = np.random.uniform(-1, 1, size=(batch_size, z_size))
        z_flex = torch.from_numpy(z_flex).float()
        if train_on_gpu:
            z_flex = z_flex.cuda()


        fake_images = G(z_flex)
        D_fake = D(fake_images)
        d_fake_loss = fake_loss(D_fake)

        d_loss = d_real_loss + d_fake_loss
        d_loss.backward()
        d_optimizer.step()
```

```
                # d_loss =

                # 2. Train the generator with an adversarial loss
                g_optimizer.zero_grad()
                z_flex = np.random.uniform(-1, 1, size=(batch_size, z_size))
                z_flex = torch.from_numpy(z_flex).float()
                if train_on_gpu:
                    z_flex = z_flex.cuda()
                fake_images = G(z_flex)
                D_fake = D(fake_images)
                g_loss = real_loss(D_fake, True) # use real loss to flip labels
                g_loss.backward()
                g_optimizer.step()

                # g_loss =


                # =============================================
                #              END OF YOUR CODE
                # =============================================

                # Print some loss stats
                if batch_i % print_every == 0:
                    # append discriminator loss and generator loss
                    losses.append((d_loss.item(), g_loss.item()))
                    # print discriminator and generator loss
                    print('Epoch [{:5d}/{:5d}] | d_loss: {:6.4f} | g_loss: {:6.4f}'.format
                            epoch+1, n_epochs, d_loss.item(), g_loss.item()))


            ## AFTER EACH EPOCH##
            # this code assumes your generator is named G, feel free to change the name
            # generate and save sample, fake images
            G.eval() # for generating samples
            samples_z = G(fixed_z)
            samples.append(samples_z)
            G.train() # back to training mode

        # Save training generator samples
        with open('train_samples.pkl', 'wb') as f:
            pkl.dump(samples, f)

        # finally return losses
        return losses
```

Set your number of training epochs and train your GAN!

```
In [18]: # set number of epochs
```

```
n_epochs = 10


"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
# call training function
losses = train(D, G, n_epochs=n_epochs)
```

```
Epoch [    1/   10] | d_loss: 1.4103 | g_loss: 0.8649
Epoch [    1/   10] | d_loss: 0.6414 | g_loss: 1.8388
Epoch [    1/   10] | d_loss: 1.1389 | g_loss: 1.8941
Epoch [    1/   10] | d_loss: 1.1434 | g_loss: 1.2447
Epoch [    1/   10] | d_loss: 1.2076 | g_loss: 1.2821
Epoch [    1/   10] | d_loss: 1.0673 | g_loss: 1.2187
Epoch [    1/   10] | d_loss: 1.2883 | g_loss: 1.6798
Epoch [    1/   10] | d_loss: 1.1828 | g_loss: 1.2983
Epoch [    1/   10] | d_loss: 1.2991 | g_loss: 1.2679
Epoch [    1/   10] | d_loss: 1.2314 | g_loss: 1.2728
Epoch [    1/   10] | d_loss: 1.0985 | g_loss: 1.7925
Epoch [    1/   10] | d_loss: 1.1944 | g_loss: 1.4801
Epoch [    1/   10] | d_loss: 1.2610 | g_loss: 1.2265
Epoch [    1/   10] | d_loss: 1.1630 | g_loss: 1.1105
Epoch [    1/   10] | d_loss: 1.2768 | g_loss: 1.6052
Epoch [    1/   10] | d_loss: 1.4771 | g_loss: 0.9836
Epoch [    1/   10] | d_loss: 1.1540 | g_loss: 1.0357
Epoch [    1/   10] | d_loss: 1.1258 | g_loss: 1.0343
Epoch [    1/   10] | d_loss: 1.1666 | g_loss: 1.4473
Epoch [    1/   10] | d_loss: 1.0558 | g_loss: 1.3700
Epoch [    1/   10] | d_loss: 0.9981 | g_loss: 1.9013
Epoch [    1/   10] | d_loss: 1.1576 | g_loss: 1.2420
Epoch [    1/   10] | d_loss: 1.3664 | g_loss: 1.2148
Epoch [    1/   10] | d_loss: 1.1972 | g_loss: 1.1748
Epoch [    1/   10] | d_loss: 1.1513 | g_loss: 0.8498
Epoch [    1/   10] | d_loss: 1.2951 | g_loss: 1.3923
Epoch [    1/   10] | d_loss: 1.5302 | g_loss: 1.2818
Epoch [    1/   10] | d_loss: 0.9914 | g_loss: 1.0479
Epoch [    1/   10] | d_loss: 1.9384 | g_loss: 1.7003
Epoch [    1/   10] | d_loss: 1.1495 | g_loss: 1.2644
Epoch [    1/   10] | d_loss: 1.1185 | g_loss: 1.4582
Epoch [    1/   10] | d_loss: 1.1669 | g_loss: 1.3153
Epoch [    1/   10] | d_loss: 1.2612 | g_loss: 2.0806
Epoch [    1/   10] | d_loss: 1.2118 | g_loss: 1.6748
Epoch [    1/   10] | d_loss: 1.1576 | g_loss: 1.3622
Epoch [    1/   10] | d_loss: 1.1168 | g_loss: 1.4827
Epoch [    1/   10] | d_loss: 0.9562 | g_loss: 1.0371
Epoch [    1/   10] | d_loss: 1.5098 | g_loss: 0.5003
Epoch [    1/   10] | d_loss: 0.9341 | g_loss: 1.4543
```

```
Epoch [    1/   10] | d_loss: 1.3392 | g_loss: 1.3947
Epoch [    1/   10] | d_loss: 1.6228 | g_loss: 2.3628
Epoch [    1/   10] | d_loss: 1.5827 | g_loss: 0.8523
Epoch [    1/   10] | d_loss: 0.9741 | g_loss: 1.8089
Epoch [    1/   10] | d_loss: 1.0552 | g_loss: 2.1223
Epoch [    1/   10] | d_loss: 1.4489 | g_loss: 1.4104
Epoch [    1/   10] | d_loss: 1.1968 | g_loss: 1.1674
Epoch [    1/   10] | d_loss: 1.0700 | g_loss: 1.3904
Epoch [    1/   10] | d_loss: 1.3905 | g_loss: 1.6217
Epoch [    1/   10] | d_loss: 1.0504 | g_loss: 1.3312
Epoch [    1/   10] | d_loss: 1.1043 | g_loss: 1.6325
Epoch [    1/   10] | d_loss: 1.3049 | g_loss: 1.2405
Epoch [    1/   10] | d_loss: 1.2886 | g_loss: 1.3083
Epoch [    1/   10] | d_loss: 1.1425 | g_loss: 1.2979
Epoch [    1/   10] | d_loss: 1.0488 | g_loss: 1.3812
Epoch [    1/   10] | d_loss: 0.9276 | g_loss: 0.8432
Epoch [    1/   10] | d_loss: 1.3014 | g_loss: 1.4298
Epoch [    1/   10] | d_loss: 1.0979 | g_loss: 1.3750
Epoch [    1/   10] | d_loss: 0.7403 | g_loss: 1.7325
Epoch [    1/   10] | d_loss: 1.4448 | g_loss: 1.7976
Epoch [    1/   10] | d_loss: 1.7891 | g_loss: 1.0492
Epoch [    1/   10] | d_loss: 1.3961 | g_loss: 1.1867
Epoch [    1/   10] | d_loss: 1.3773 | g_loss: 0.9567
Epoch [    1/   10] | d_loss: 1.3288 | g_loss: 1.3316
Epoch [    1/   10] | d_loss: 0.9886 | g_loss: 1.0417
Epoch [    1/   10] | d_loss: 0.8201 | g_loss: 1.4480
Epoch [    1/   10] | d_loss: 1.2818 | g_loss: 0.6758
Epoch [    1/   10] | d_loss: 1.2403 | g_loss: 1.0661
Epoch [    1/   10] | d_loss: 0.9163 | g_loss: 1.9001
Epoch [    1/   10] | d_loss: 1.1919 | g_loss: 1.2016
Epoch [    1/   10] | d_loss: 1.7347 | g_loss: 1.7580
Epoch [    1/   10] | d_loss: 0.8974 | g_loss: 1.3421
Epoch [    1/   10] | d_loss: 1.2814 | g_loss: 0.9780
Epoch [    1/   10] | d_loss: 1.2508 | g_loss: 1.2255
Epoch [    1/   10] | d_loss: 0.7616 | g_loss: 0.9957
Epoch [    1/   10] | d_loss: 0.9434 | g_loss: 2.1051
Epoch [    1/   10] | d_loss: 1.2861 | g_loss: 1.4068
Epoch [    1/   10] | d_loss: 1.5539 | g_loss: 1.8186
Epoch [    1/   10] | d_loss: 0.8659 | g_loss: 2.5097
Epoch [    1/   10] | d_loss: 1.0512 | g_loss: 1.2382
Epoch [    1/   10] | d_loss: 1.2701 | g_loss: 0.9436
Epoch [    1/   10] | d_loss: 1.1061 | g_loss: 0.9417
Epoch [    1/   10] | d_loss: 1.3371 | g_loss: 1.0656
Epoch [    1/   10] | d_loss: 1.0658 | g_loss: 1.6086
Epoch [    1/   10] | d_loss: 1.1587 | g_loss: 1.1787
Epoch [    1/   10] | d_loss: 1.1968 | g_loss: 1.3910
Epoch [    1/   10] | d_loss: 1.7940 | g_loss: 0.7069
Epoch [    1/   10] | d_loss: 1.1566 | g_loss: 1.0682
```

```
Epoch [    1/   10] | d_loss: 1.1032 | g_loss: 1.1307
Epoch [    1/   10] | d_loss: 1.0429 | g_loss: 1.1458
Epoch [    1/   10] | d_loss: 1.4239 | g_loss: 1.7134
Epoch [    1/   10] | d_loss: 0.9672 | g_loss: 1.9387
Epoch [    1/   10] | d_loss: 0.9789 | g_loss: 1.3274
Epoch [    1/   10] | d_loss: 1.0007 | g_loss: 1.2832
Epoch [    1/   10] | d_loss: 0.8205 | g_loss: 1.0760
Epoch [    1/   10] | d_loss: 0.9490 | g_loss: 1.0498
Epoch [    1/   10] | d_loss: 1.3411 | g_loss: 0.9639
Epoch [    1/   10] | d_loss: 1.3385 | g_loss: 1.0201
Epoch [    1/   10] | d_loss: 1.2865 | g_loss: 2.0793
Epoch [    1/   10] | d_loss: 1.7690 | g_loss: 1.8706
Epoch [    1/   10] | d_loss: 1.3427 | g_loss: 0.6426
Epoch [    2/   10] | d_loss: 1.3379 | g_loss: 1.4318
Epoch [    2/   10] | d_loss: 1.3931 | g_loss: 0.7442
Epoch [    2/   10] | d_loss: 1.3189 | g_loss: 1.5368
Epoch [    2/   10] | d_loss: 1.0823 | g_loss: 1.1640
Epoch [    2/   10] | d_loss: 1.0328 | g_loss: 1.3576
Epoch [    2/   10] | d_loss: 0.9442 | g_loss: 1.1528
Epoch [    2/   10] | d_loss: 1.0617 | g_loss: 1.4182
Epoch [    2/   10] | d_loss: 1.3134 | g_loss: 1.1250
Epoch [    2/   10] | d_loss: 1.1504 | g_loss: 1.2901
Epoch [    2/   10] | d_loss: 0.8238 | g_loss: 1.6327
Epoch [    2/   10] | d_loss: 1.2482 | g_loss: 0.8293
Epoch [    2/   10] | d_loss: 0.7371 | g_loss: 0.9002
Epoch [    2/   10] | d_loss: 1.2959 | g_loss: 1.5339
Epoch [    2/   10] | d_loss: 0.9374 | g_loss: 1.5543
Epoch [    2/   10] | d_loss: 1.0167 | g_loss: 1.7638
Epoch [    2/   10] | d_loss: 1.7221 | g_loss: 1.5521
Epoch [    2/   10] | d_loss: 1.0594 | g_loss: 1.1673
Epoch [    2/   10] | d_loss: 1.0062 | g_loss: 1.4988
Epoch [    2/   10] | d_loss: 1.6373 | g_loss: 1.3845
Epoch [    2/   10] | d_loss: 1.3234 | g_loss: 1.5693
Epoch [    2/   10] | d_loss: 0.9057 | g_loss: 1.7029
Epoch [    2/   10] | d_loss: 1.2224 | g_loss: 1.0099
Epoch [    2/   10] | d_loss: 1.3510 | g_loss: 1.5625
Epoch [    2/   10] | d_loss: 1.0788 | g_loss: 1.4520
Epoch [    2/   10] | d_loss: 1.3129 | g_loss: 1.1288
Epoch [    2/   10] | d_loss: 0.9846 | g_loss: 0.9875
Epoch [    2/   10] | d_loss: 1.3664 | g_loss: 1.3223
Epoch [    2/   10] | d_loss: 1.0501 | g_loss: 1.4118
Epoch [    2/   10] | d_loss: 1.2863 | g_loss: 1.0138
Epoch [    2/   10] | d_loss: 1.3452 | g_loss: 1.2460
Epoch [    2/   10] | d_loss: 0.9673 | g_loss: 0.9498
Epoch [    2/   10] | d_loss: 1.5282 | g_loss: 0.9356
Epoch [    2/   10] | d_loss: 1.3255 | g_loss: 0.9614
Epoch [    2/   10] | d_loss: 0.8805 | g_loss: 1.4827
Epoch [    2/   10] | d_loss: 1.3921 | g_loss: 1.4464
```

```
Epoch [    2/   10] | d_loss: 0.9307 | g_loss: 0.6970
Epoch [    2/   10] | d_loss: 1.0076 | g_loss: 1.2315
Epoch [    2/   10] | d_loss: 1.3195 | g_loss: 1.4990
Epoch [    2/   10] | d_loss: 1.1246 | g_loss: 1.1201
Epoch [    2/   10] | d_loss: 1.4811 | g_loss: 1.5163
Epoch [    2/   10] | d_loss: 1.4080 | g_loss: 0.7973
Epoch [    2/   10] | d_loss: 0.9877 | g_loss: 1.4691
Epoch [    2/   10] | d_loss: 1.4308 | g_loss: 1.8648
Epoch [    2/   10] | d_loss: 1.1816 | g_loss: 1.4244
Epoch [    2/   10] | d_loss: 1.0726 | g_loss: 1.4631
Epoch [    2/   10] | d_loss: 1.3348 | g_loss: 1.5772
Epoch [    2/   10] | d_loss: 1.1720 | g_loss: 1.4212
Epoch [    2/   10] | d_loss: 1.4834 | g_loss: 0.9715
Epoch [    2/   10] | d_loss: 1.4901 | g_loss: 2.5980
Epoch [    2/   10] | d_loss: 0.9624 | g_loss: 1.0663
Epoch [    2/   10] | d_loss: 0.8093 | g_loss: 0.7809
Epoch [    2/   10] | d_loss: 1.1857 | g_loss: 1.2683
Epoch [    2/   10] | d_loss: 1.0130 | g_loss: 1.3367
Epoch [    2/   10] | d_loss: 1.1148 | g_loss: 1.5311
Epoch [    2/   10] | d_loss: 1.3640 | g_loss: 0.8682
Epoch [    2/   10] | d_loss: 1.1825 | g_loss: 1.0035
Epoch [    2/   10] | d_loss: 1.1066 | g_loss: 0.6622
Epoch [    2/   10] | d_loss: 1.1667 | g_loss: 1.0609
Epoch [    2/   10] | d_loss: 1.5087 | g_loss: 1.7658
Epoch [    2/   10] | d_loss: 1.3687 | g_loss: 0.5061
Epoch [    2/   10] | d_loss: 1.1284 | g_loss: 1.3343
Epoch [    2/   10] | d_loss: 1.7026 | g_loss: 1.6096
Epoch [    2/   10] | d_loss: 1.0271 | g_loss: 0.7906
Epoch [    2/   10] | d_loss: 0.9245 | g_loss: 0.9155
Epoch [    2/   10] | d_loss: 0.8068 | g_loss: 1.4053
Epoch [    2/   10] | d_loss: 1.1914 | g_loss: 1.6554
Epoch [    2/   10] | d_loss: 1.8400 | g_loss: 1.7792
Epoch [    2/   10] | d_loss: 0.9810 | g_loss: 1.2477
Epoch [    2/   10] | d_loss: 0.9988 | g_loss: 1.7551
Epoch [    2/   10] | d_loss: 1.1692 | g_loss: 1.4613
Epoch [    2/   10] | d_loss: 1.2089 | g_loss: 1.1251
Epoch [    2/   10] | d_loss: 1.0658 | g_loss: 1.8237
Epoch [    2/   10] | d_loss: 1.1545 | g_loss: 2.1184
Epoch [    2/   10] | d_loss: 0.8595 | g_loss: 1.6110
Epoch [    2/   10] | d_loss: 1.1333 | g_loss: 0.6485
Epoch [    2/   10] | d_loss: 0.9414 | g_loss: 1.0437
Epoch [    2/   10] | d_loss: 1.2892 | g_loss: 1.2252
Epoch [    2/   10] | d_loss: 1.5244 | g_loss: 1.1650
Epoch [    2/   10] | d_loss: 1.6249 | g_loss: 1.2196
Epoch [    2/   10] | d_loss: 1.5224 | g_loss: 1.1750
Epoch [    2/   10] | d_loss: 1.1065 | g_loss: 1.3608
Epoch [    2/   10] | d_loss: 1.1094 | g_loss: 1.6201
Epoch [    2/   10] | d_loss: 1.1512 | g_loss: 1.3448
```

```
Epoch [    2/   10] | d_loss: 1.0591 | g_loss: 1.5769
Epoch [    2/   10] | d_loss: 0.8717 | g_loss: 1.4261
Epoch [    2/   10] | d_loss: 1.1656 | g_loss: 1.1680
Epoch [    2/   10] | d_loss: 1.0802 | g_loss: 1.0869
Epoch [    2/   10] | d_loss: 2.1876 | g_loss: 0.6316
Epoch [    2/   10] | d_loss: 1.3443 | g_loss: 1.4422
Epoch [    2/   10] | d_loss: 0.9824 | g_loss: 0.7092
Epoch [    2/   10] | d_loss: 0.9545 | g_loss: 1.5944
Epoch [    2/   10] | d_loss: 1.0966 | g_loss: 1.2372
Epoch [    2/   10] | d_loss: 0.7893 | g_loss: 1.9092
Epoch [    2/   10] | d_loss: 1.5798 | g_loss: 0.5473
Epoch [    2/   10] | d_loss: 0.9107 | g_loss: 0.5006
Epoch [    2/   10] | d_loss: 0.7768 | g_loss: 0.9963
Epoch [    2/   10] | d_loss: 0.9451 | g_loss: 1.8689
Epoch [    2/   10] | d_loss: 0.9118 | g_loss: 0.9094
Epoch [    2/   10] | d_loss: 0.9395 | g_loss: 0.9856
Epoch [    2/   10] | d_loss: 1.5276 | g_loss: 2.1160
Epoch [    3/   10] | d_loss: 1.1939 | g_loss: 0.6975
Epoch [    3/   10] | d_loss: 0.8538 | g_loss: 1.8723
Epoch [    3/   10] | d_loss: 1.0590 | g_loss: 1.6050
Epoch [    3/   10] | d_loss: 0.8585 | g_loss: 1.2536
Epoch [    3/   10] | d_loss: 1.1475 | g_loss: 1.2481
Epoch [    3/   10] | d_loss: 1.0412 | g_loss: 1.7288
Epoch [    3/   10] | d_loss: 0.8366 | g_loss: 1.5531
Epoch [    3/   10] | d_loss: 0.9045 | g_loss: 1.1366
Epoch [    3/   10] | d_loss: 0.9156 | g_loss: 1.0974
Epoch [    3/   10] | d_loss: 0.9208 | g_loss: 2.5555
Epoch [    3/   10] | d_loss: 1.1739 | g_loss: 1.1167
Epoch [    3/   10] | d_loss: 1.1219 | g_loss: 1.2061
Epoch [    3/   10] | d_loss: 0.9276 | g_loss: 0.9949
Epoch [    3/   10] | d_loss: 1.0245 | g_loss: 0.8370
Epoch [    3/   10] | d_loss: 1.2531 | g_loss: 0.8387
Epoch [    3/   10] | d_loss: 1.0426 | g_loss: 1.2204
Epoch [    3/   10] | d_loss: 0.9174 | g_loss: 1.1708
Epoch [    3/   10] | d_loss: 0.9506 | g_loss: 1.8751
Epoch [    3/   10] | d_loss: 1.1347 | g_loss: 1.5218
Epoch [    3/   10] | d_loss: 0.7910 | g_loss: 1.7509
Epoch [    3/   10] | d_loss: 1.6403 | g_loss: 0.5896
Epoch [    3/   10] | d_loss: 1.0739 | g_loss: 1.3162
Epoch [    3/   10] | d_loss: 1.9050 | g_loss: 0.5380
Epoch [    3/   10] | d_loss: 1.4316 | g_loss: 0.7644
Epoch [    3/   10] | d_loss: 0.7036 | g_loss: 1.1737
Epoch [    3/   10] | d_loss: 1.0575 | g_loss: 1.6897
Epoch [    3/   10] | d_loss: 1.1357 | g_loss: 1.9642
Epoch [    3/   10] | d_loss: 1.0791 | g_loss: 1.7353
Epoch [    3/   10] | d_loss: 1.2561 | g_loss: 1.1281
Epoch [    3/   10] | d_loss: 1.3533 | g_loss: 0.9836
Epoch [    3/   10] | d_loss: 1.1515 | g_loss: 0.7911
```

```
Epoch [    3/   10] | d_loss: 0.8330 | g_loss: 0.9250
Epoch [    3/   10] | d_loss: 0.9304 | g_loss: 0.9170
Epoch [    3/   10] | d_loss: 0.8616 | g_loss: 1.1445
Epoch [    3/   10] | d_loss: 0.9966 | g_loss: 1.3215
Epoch [    3/   10] | d_loss: 1.5068 | g_loss: 1.4020
Epoch [    3/   10] | d_loss: 1.3179 | g_loss: 1.5114
Epoch [    3/   10] | d_loss: 0.8582 | g_loss: 1.4847
Epoch [    3/   10] | d_loss: 0.9187 | g_loss: 0.9491
Epoch [    3/   10] | d_loss: 0.7239 | g_loss: 1.0730
Epoch [    3/   10] | d_loss: 1.1304 | g_loss: 1.5026
Epoch [    3/   10] | d_loss: 0.6660 | g_loss: 1.2360
Epoch [    3/   10] | d_loss: 1.2375 | g_loss: 1.1830
Epoch [    3/   10] | d_loss: 0.8281 | g_loss: 1.0219
Epoch [    3/   10] | d_loss: 0.7693 | g_loss: 1.1429
Epoch [    3/   10] | d_loss: 1.1582 | g_loss: 1.3646
Epoch [    3/   10] | d_loss: 1.2366 | g_loss: 1.6084
Epoch [    3/   10] | d_loss: 0.8740 | g_loss: 1.6241
Epoch [    3/   10] | d_loss: 0.9928 | g_loss: 0.9537
Epoch [    3/   10] | d_loss: 0.8618 | g_loss: 1.2965
Epoch [    3/   10] | d_loss: 1.2395 | g_loss: 0.9222
Epoch [    3/   10] | d_loss: 1.0520 | g_loss: 1.7028
Epoch [    3/   10] | d_loss: 0.8895 | g_loss: 0.8827
Epoch [    3/   10] | d_loss: 1.0895 | g_loss: 0.8075
Epoch [    3/   10] | d_loss: 0.9151 | g_loss: 0.9287
Epoch [    3/   10] | d_loss: 1.0853 | g_loss: 0.7727
Epoch [    3/   10] | d_loss: 1.7984 | g_loss: 2.6502
Epoch [    3/   10] | d_loss: 0.6735 | g_loss: 1.6411
Epoch [    3/   10] | d_loss: 0.8963 | g_loss: 1.5371
Epoch [    3/   10] | d_loss: 0.8960 | g_loss: 0.7179
Epoch [    3/   10] | d_loss: 0.8318 | g_loss: 0.5750
Epoch [    3/   10] | d_loss: 1.1109 | g_loss: 1.6622
Epoch [    3/   10] | d_loss: 1.6628 | g_loss: 1.0108
Epoch [    3/   10] | d_loss: 0.7987 | g_loss: 1.0814
Epoch [    3/   10] | d_loss: 0.7154 | g_loss: 1.2933
Epoch [    3/   10] | d_loss: 1.0257 | g_loss: 1.0376
Epoch [    3/   10] | d_loss: 1.0146 | g_loss: 1.2451
Epoch [    3/   10] | d_loss: 0.9969 | g_loss: 1.2037
Epoch [    3/   10] | d_loss: 0.7804 | g_loss: 0.9637
Epoch [    3/   10] | d_loss: 1.3120 | g_loss: 0.7019
Epoch [    3/   10] | d_loss: 2.0856 | g_loss: 2.5503
Epoch [    3/   10] | d_loss: 0.9744 | g_loss: 1.1674
Epoch [    3/   10] | d_loss: 1.9452 | g_loss: 0.7028
Epoch [    3/   10] | d_loss: 0.8743 | g_loss: 1.7334
Epoch [    3/   10] | d_loss: 1.6635 | g_loss: 0.7133
Epoch [    3/   10] | d_loss: 1.6023 | g_loss: 2.0029
Epoch [    3/   10] | d_loss: 1.0845 | g_loss: 2.5748
Epoch [    3/   10] | d_loss: 1.1788 | g_loss: 0.8825
Epoch [    3/   10] | d_loss: 1.0841 | g_loss: 0.9637
```

```
Epoch [    3/   10] | d_loss: 1.0472 | g_loss: 1.2540
Epoch [    3/   10] | d_loss: 0.6032 | g_loss: 1.0978
Epoch [    3/   10] | d_loss: 0.9984 | g_loss: 1.1032
Epoch [    3/   10] | d_loss: 1.1737 | g_loss: 1.1946
Epoch [    3/   10] | d_loss: 1.1041 | g_loss: 3.0604
Epoch [    3/   10] | d_loss: 1.1370 | g_loss: 0.8613
Epoch [    3/   10] | d_loss: 1.8401 | g_loss: 1.3623
Epoch [    3/   10] | d_loss: 0.7785 | g_loss: 2.2317
Epoch [    3/   10] | d_loss: 0.6003 | g_loss: 1.9250
Epoch [    3/   10] | d_loss: 1.0168 | g_loss: 1.1982
Epoch [    3/   10] | d_loss: 0.5025 | g_loss: 1.7267
Epoch [    3/   10] | d_loss: 0.6585 | g_loss: 1.6590
Epoch [    3/   10] | d_loss: 0.9350 | g_loss: 1.3754
Epoch [    3/   10] | d_loss: 1.3864 | g_loss: 1.4820
Epoch [    3/   10] | d_loss: 1.1537 | g_loss: 1.0857
Epoch [    3/   10] | d_loss: 0.7770 | g_loss: 1.2987
Epoch [    3/   10] | d_loss: 0.6280 | g_loss: 1.2153
Epoch [    3/   10] | d_loss: 1.1406 | g_loss: 1.7510
Epoch [    3/   10] | d_loss: 0.9211 | g_loss: 1.0967
Epoch [    3/   10] | d_loss: 1.4530 | g_loss: 1.0076
Epoch [    3/   10] | d_loss: 1.0602 | g_loss: 1.2380
Epoch [    4/   10] | d_loss: 0.9532 | g_loss: 1.6827
Epoch [    4/   10] | d_loss: 0.9183 | g_loss: 1.4903
Epoch [    4/   10] | d_loss: 1.2641 | g_loss: 0.9632
Epoch [    4/   10] | d_loss: 1.2330 | g_loss: 2.4446
Epoch [    4/   10] | d_loss: 1.1490 | g_loss: 2.1710
Epoch [    4/   10] | d_loss: 0.7996 | g_loss: 0.8543
Epoch [    4/   10] | d_loss: 0.9126 | g_loss: 2.2447
Epoch [    4/   10] | d_loss: 1.3259 | g_loss: 1.4373
Epoch [    4/   10] | d_loss: 1.2079 | g_loss: 0.9897
Epoch [    4/   10] | d_loss: 1.3487 | g_loss: 1.8678
Epoch [    4/   10] | d_loss: 1.2319 | g_loss: 1.4863
Epoch [    4/   10] | d_loss: 1.7939 | g_loss: 1.9478
Epoch [    4/   10] | d_loss: 1.0301 | g_loss: 1.1384
Epoch [    4/   10] | d_loss: 0.7094 | g_loss: 1.2593
Epoch [    4/   10] | d_loss: 1.2279 | g_loss: 0.9519
Epoch [    4/   10] | d_loss: 0.7810 | g_loss: 1.7262
Epoch [    4/   10] | d_loss: 0.8896 | g_loss: 1.2655
Epoch [    4/   10] | d_loss: 0.8726 | g_loss: 1.4518
Epoch [    4/   10] | d_loss: 1.0130 | g_loss: 1.1444
Epoch [    4/   10] | d_loss: 0.6001 | g_loss: 2.5749
Epoch [    4/   10] | d_loss: 0.8966 | g_loss: 1.0271
Epoch [    4/   10] | d_loss: 0.8443 | g_loss: 2.4946
Epoch [    4/   10] | d_loss: 0.6677 | g_loss: 2.0419
Epoch [    4/   10] | d_loss: 0.8628 | g_loss: 1.5613
Epoch [    4/   10] | d_loss: 1.4877 | g_loss: 1.6583
Epoch [    4/   10] | d_loss: 1.2863 | g_loss: 0.7654
Epoch [    4/   10] | d_loss: 0.5247 | g_loss: 2.2302
```

```
Epoch [    4/   10] | d_loss: 0.4518 | g_loss: 1.8348
Epoch [    4/   10] | d_loss: 0.5122 | g_loss: 2.2591
Epoch [    4/   10] | d_loss: 0.9537 | g_loss: 1.4999
Epoch [    4/   10] | d_loss: 0.9248 | g_loss: 1.6518
Epoch [    4/   10] | d_loss: 1.0471 | g_loss: 1.5773
Epoch [    4/   10] | d_loss: 0.4747 | g_loss: 1.5146
Epoch [    4/   10] | d_loss: 1.0194 | g_loss: 1.4899
Epoch [    4/   10] | d_loss: 0.9436 | g_loss: 0.6121
Epoch [    4/   10] | d_loss: 0.8733 | g_loss: 1.4771
Epoch [    4/   10] | d_loss: 1.3106 | g_loss: 1.3174
Epoch [    4/   10] | d_loss: 0.7846 | g_loss: 1.7301
Epoch [    4/   10] | d_loss: 0.9719 | g_loss: 1.5384
Epoch [    4/   10] | d_loss: 1.2102 | g_loss: 1.2036
Epoch [    4/   10] | d_loss: 0.5000 | g_loss: 2.1715
Epoch [    4/   10] | d_loss: 1.2394 | g_loss: 1.2792
Epoch [    4/   10] | d_loss: 0.9689 | g_loss: 1.1481
Epoch [    4/   10] | d_loss: 0.8266 | g_loss: 1.0324
Epoch [    4/   10] | d_loss: 0.8996 | g_loss: 2.3515
Epoch [    4/   10] | d_loss: 0.8213 | g_loss: 0.5968
Epoch [    4/   10] | d_loss: 1.8528 | g_loss: 2.2124
Epoch [    4/   10] | d_loss: 0.6160 | g_loss: 2.1925
Epoch [    4/   10] | d_loss: 1.0917 | g_loss: 2.3224
Epoch [    4/   10] | d_loss: 0.7951 | g_loss: 0.9747
Epoch [    4/   10] | d_loss: 1.0053 | g_loss: 2.6722
Epoch [    4/   10] | d_loss: 1.0174 | g_loss: 1.6424
Epoch [    4/   10] | d_loss: 1.5945 | g_loss: 1.6615
Epoch [    4/   10] | d_loss: 1.4528 | g_loss: 0.8363
Epoch [    4/   10] | d_loss: 1.6926 | g_loss: 2.3870
Epoch [    4/   10] | d_loss: 0.8564 | g_loss: 0.9154
Epoch [    4/   10] | d_loss: 0.9711 | g_loss: 1.1711
Epoch [    4/   10] | d_loss: 0.8593 | g_loss: 1.0469
Epoch [    4/   10] | d_loss: 1.1585 | g_loss: 1.5057
Epoch [    4/   10] | d_loss: 1.2417 | g_loss: 1.2665
Epoch [    4/   10] | d_loss: 0.6405 | g_loss: 0.4508
Epoch [    4/   10] | d_loss: 1.0384 | g_loss: 0.9953
Epoch [    4/   10] | d_loss: 0.7906 | g_loss: 1.4927
Epoch [    4/   10] | d_loss: 1.0335 | g_loss: 1.7483
Epoch [    4/   10] | d_loss: 1.1060 | g_loss: 1.2970
Epoch [    4/   10] | d_loss: 1.2898 | g_loss: 2.6675
Epoch [    4/   10] | d_loss: 1.3971 | g_loss: 0.8059
Epoch [    4/   10] | d_loss: 1.1067 | g_loss: 1.4219
Epoch [    4/   10] | d_loss: 0.6596 | g_loss: 1.2327
Epoch [    4/   10] | d_loss: 0.9183 | g_loss: 1.0875
Epoch [    4/   10] | d_loss: 1.2820 | g_loss: 1.5328
Epoch [    4/   10] | d_loss: 1.0861 | g_loss: 1.0447
Epoch [    4/   10] | d_loss: 0.8615 | g_loss: 1.5066
Epoch [    4/   10] | d_loss: 0.9028 | g_loss: 1.3933
Epoch [    4/   10] | d_loss: 0.7344 | g_loss: 1.1209
```

```
Epoch [    4/   10] | d_loss: 0.5735 | g_loss: 0.9192
Epoch [    4/   10] | d_loss: 0.8148 | g_loss: 1.6641
Epoch [    4/   10] | d_loss: 0.8113 | g_loss: 1.5773
Epoch [    4/   10] | d_loss: 1.2705 | g_loss: 1.9547
Epoch [    4/   10] | d_loss: 0.4782 | g_loss: 1.5067
Epoch [    4/   10] | d_loss: 1.2484 | g_loss: 2.0060
Epoch [    4/   10] | d_loss: 1.3035 | g_loss: 1.5368
Epoch [    4/   10] | d_loss: 1.5325 | g_loss: 0.7558
Epoch [    4/   10] | d_loss: 0.8324 | g_loss: 1.2056
Epoch [    4/   10] | d_loss: 1.0244 | g_loss: 1.3175
Epoch [    4/   10] | d_loss: 0.7272 | g_loss: 1.3613
Epoch [    4/   10] | d_loss: 1.3069 | g_loss: 2.0744
Epoch [    4/   10] | d_loss: 0.7670 | g_loss: 1.2373
Epoch [    4/   10] | d_loss: 1.1195 | g_loss: 1.8027
Epoch [    4/   10] | d_loss: 0.9227 | g_loss: 1.3825
Epoch [    4/   10] | d_loss: 1.7787 | g_loss: 2.0837
Epoch [    4/   10] | d_loss: 0.7396 | g_loss: 1.4075
Epoch [    4/   10] | d_loss: 0.9625 | g_loss: 2.6159
Epoch [    4/   10] | d_loss: 0.8395 | g_loss: 1.3581
Epoch [    4/   10] | d_loss: 0.8313 | g_loss: 2.0187
Epoch [    4/   10] | d_loss: 0.8822 | g_loss: 1.2271
Epoch [    4/   10] | d_loss: 0.6529 | g_loss: 2.0047
Epoch [    4/   10] | d_loss: 1.3928 | g_loss: 1.1244
Epoch [    4/   10] | d_loss: 0.8067 | g_loss: 1.8966
Epoch [    4/   10] | d_loss: 0.8510 | g_loss: 1.9897
Epoch [    5/   10] | d_loss: 0.9166 | g_loss: 0.9109
Epoch [    5/   10] | d_loss: 0.5274 | g_loss: 1.7477
Epoch [    5/   10] | d_loss: 0.5547 | g_loss: 2.2752
Epoch [    5/   10] | d_loss: 0.8210 | g_loss: 0.7183
Epoch [    5/   10] | d_loss: 0.4445 | g_loss: 2.1942
Epoch [    5/   10] | d_loss: 0.7507 | g_loss: 1.0824
Epoch [    5/   10] | d_loss: 1.3123 | g_loss: 0.7929
Epoch [    5/   10] | d_loss: 0.9258 | g_loss: 0.9372
Epoch [    5/   10] | d_loss: 1.9030 | g_loss: 1.7763
Epoch [    5/   10] | d_loss: 1.4073 | g_loss: 2.0853
Epoch [    5/   10] | d_loss: 0.5200 | g_loss: 1.6764
Epoch [    5/   10] | d_loss: 0.5155 | g_loss: 0.8750
Epoch [    5/   10] | d_loss: 0.4792 | g_loss: 2.2765
Epoch [    5/   10] | d_loss: 1.0491 | g_loss: 1.5939
Epoch [    5/   10] | d_loss: 0.9230 | g_loss: 0.6559
Epoch [    5/   10] | d_loss: 1.0197 | g_loss: 1.0866
Epoch [    5/   10] | d_loss: 0.9938 | g_loss: 1.1983
Epoch [    5/   10] | d_loss: 0.7105 | g_loss: 1.0754
Epoch [    5/   10] | d_loss: 0.7726 | g_loss: 1.2404
Epoch [    5/   10] | d_loss: 1.1384 | g_loss: 0.7565
Epoch [    5/   10] | d_loss: 0.6885 | g_loss: 1.6991
Epoch [    5/   10] | d_loss: 0.9113 | g_loss: 1.6230
Epoch [    5/   10] | d_loss: 1.3084 | g_loss: 0.8324
```

```
Epoch [    5/   10] | d_loss: 0.5023 | g_loss: 2.1223
Epoch [    5/   10] | d_loss: 1.7493 | g_loss: 1.3027
Epoch [    5/   10] | d_loss: 1.1548 | g_loss: 0.6824
Epoch [    5/   10] | d_loss: 0.8755 | g_loss: 1.5680
Epoch [    5/   10] | d_loss: 1.0124 | g_loss: 1.3306
Epoch [    5/   10] | d_loss: 0.7783 | g_loss: 2.3941
Epoch [    5/   10] | d_loss: 0.9790 | g_loss: 1.3543
Epoch [    5/   10] | d_loss: 0.4920 | g_loss: 1.3563
Epoch [    5/   10] | d_loss: 0.4713 | g_loss: 2.4572
Epoch [    5/   10] | d_loss: 0.7678 | g_loss: 1.2969
Epoch [    5/   10] | d_loss: 0.8048 | g_loss: 1.9051
Epoch [    5/   10] | d_loss: 1.3612 | g_loss: 1.1191
Epoch [    5/   10] | d_loss: 1.4942 | g_loss: 0.7424
Epoch [    5/   10] | d_loss: 1.6317 | g_loss: 1.2699
Epoch [    5/   10] | d_loss: 0.4619 | g_loss: 1.9289
Epoch [    5/   10] | d_loss: 1.1223 | g_loss: 1.7001
Epoch [    5/   10] | d_loss: 0.8188 | g_loss: 0.9223
Epoch [    5/   10] | d_loss: 0.7488 | g_loss: 0.6590
Epoch [    5/   10] | d_loss: 1.0117 | g_loss: 1.9842
Epoch [    5/   10] | d_loss: 0.6902 | g_loss: 1.8735
Epoch [    5/   10] | d_loss: 0.8938 | g_loss: 1.1908
Epoch [    5/   10] | d_loss: 1.3954 | g_loss: 0.8841
Epoch [    5/   10] | d_loss: 0.7160 | g_loss: 2.3099
Epoch [    5/   10] | d_loss: 1.0774 | g_loss: 3.4963
Epoch [    5/   10] | d_loss: 1.1672 | g_loss: 0.8723
Epoch [    5/   10] | d_loss: 0.6665 | g_loss: 2.3460
Epoch [    5/   10] | d_loss: 0.6310 | g_loss: 1.1578
Epoch [    5/   10] | d_loss: 0.5426 | g_loss: 1.8361
Epoch [    5/   10] | d_loss: 0.6256 | g_loss: 1.9651
Epoch [    5/   10] | d_loss: 0.7586 | g_loss: 1.6411
Epoch [    5/   10] | d_loss: 1.0595 | g_loss: 1.5612
Epoch [    5/   10] | d_loss: 0.7740 | g_loss: 0.8654
Epoch [    5/   10] | d_loss: 0.7464 | g_loss: 1.8950
Epoch [    5/   10] | d_loss: 0.5281 | g_loss: 0.8100
Epoch [    5/   10] | d_loss: 1.2437 | g_loss: 0.6387
Epoch [    5/   10] | d_loss: 0.9890 | g_loss: 2.3341
Epoch [    5/   10] | d_loss: 0.6741 | g_loss: 1.9735
Epoch [    5/   10] | d_loss: 0.9407 | g_loss: 1.5267
Epoch [    5/   10] | d_loss: 0.6912 | g_loss: 0.8558
Epoch [    5/   10] | d_loss: 1.0265 | g_loss: 2.3864
Epoch [    5/   10] | d_loss: 0.7701 | g_loss: 0.9044
Epoch [    5/   10] | d_loss: 0.4345 | g_loss: 1.6829
Epoch [    5/   10] | d_loss: 0.6971 | g_loss: 3.1617
Epoch [    5/   10] | d_loss: 0.6683 | g_loss: 2.2199
Epoch [    5/   10] | d_loss: 1.2863 | g_loss: 1.2639
Epoch [    5/   10] | d_loss: 1.0823 | g_loss: 2.5284
Epoch [    5/   10] | d_loss: 0.7077 | g_loss: 1.9231
Epoch [    5/   10] | d_loss: 0.6878 | g_loss: 1.5830
```

```
Epoch [    5/   10] | d_loss: 1.4067 | g_loss: 1.3835
Epoch [    5/   10] | d_loss: 1.0378 | g_loss: 2.3387
Epoch [    5/   10] | d_loss: 1.6075 | g_loss: 2.7685
Epoch [    5/   10] | d_loss: 0.8502 | g_loss: 1.4110
Epoch [    5/   10] | d_loss: 0.9213 | g_loss: 2.3204
Epoch [    5/   10] | d_loss: 0.5483 | g_loss: 1.4494
Epoch [    5/   10] | d_loss: 1.3129 | g_loss: 3.2201
Epoch [    5/   10] | d_loss: 0.3741 | g_loss: 3.1199
Epoch [    5/   10] | d_loss: 0.9282 | g_loss: 2.1041
Epoch [    5/   10] | d_loss: 0.7695 | g_loss: 2.2916
Epoch [    5/   10] | d_loss: 0.4838 | g_loss: 1.3339
Epoch [    5/   10] | d_loss: 1.0026 | g_loss: 1.1537
Epoch [    5/   10] | d_loss: 1.0132 | g_loss: 1.1944
Epoch [    5/   10] | d_loss: 1.4133 | g_loss: 1.5574
Epoch [    5/   10] | d_loss: 0.7990 | g_loss: 1.4073
Epoch [    5/   10] | d_loss: 1.0526 | g_loss: 1.3659
Epoch [    5/   10] | d_loss: 0.9425 | g_loss: 0.8701
Epoch [    5/   10] | d_loss: 0.5832 | g_loss: 3.0287
Epoch [    5/   10] | d_loss: 0.3876 | g_loss: 0.9847
Epoch [    5/   10] | d_loss: 1.7602 | g_loss: 1.2398
Epoch [    5/   10] | d_loss: 1.5685 | g_loss: 2.7442
Epoch [    5/   10] | d_loss: 0.7033 | g_loss: 0.9335
Epoch [    5/   10] | d_loss: 1.2229 | g_loss: 1.7393
Epoch [    5/   10] | d_loss: 0.9503 | g_loss: 1.0868
Epoch [    5/   10] | d_loss: 1.1870 | g_loss: 1.2888
Epoch [    5/   10] | d_loss: 1.0710 | g_loss: 1.6763
Epoch [    5/   10] | d_loss: 0.7037 | g_loss: 3.3399
Epoch [    5/   10] | d_loss: 0.9383 | g_loss: 2.7474
Epoch [    5/   10] | d_loss: 0.7966 | g_loss: 1.4184
Epoch [    6/   10] | d_loss: 1.7135 | g_loss: 1.8710
Epoch [    6/   10] | d_loss: 0.9887 | g_loss: 2.0725
Epoch [    6/   10] | d_loss: 0.7877 | g_loss: 0.7019
Epoch [    6/   10] | d_loss: 1.1165 | g_loss: 0.8605
Epoch [    6/   10] | d_loss: 0.5470 | g_loss: 1.5494
Epoch [    6/   10] | d_loss: 1.8400 | g_loss: 1.4453
Epoch [    6/   10] | d_loss: 0.4436 | g_loss: 3.1697
Epoch [    6/   10] | d_loss: 1.5085 | g_loss: 0.8224
Epoch [    6/   10] | d_loss: 0.7772 | g_loss: 2.1140
Epoch [    6/   10] | d_loss: 0.5046 | g_loss: 1.6985
Epoch [    6/   10] | d_loss: 0.4173 | g_loss: 2.0933
Epoch [    6/   10] | d_loss: 1.0846 | g_loss: 1.4411
Epoch [    6/   10] | d_loss: 0.9765 | g_loss: 1.1312
Epoch [    6/   10] | d_loss: 0.6386 | g_loss: 2.0008
Epoch [    6/   10] | d_loss: 0.6914 | g_loss: 1.9655
Epoch [    6/   10] | d_loss: 1.5439 | g_loss: 1.0561
Epoch [    6/   10] | d_loss: 0.8500 | g_loss: 1.0383
Epoch [    6/   10] | d_loss: 0.7134 | g_loss: 2.3271
Epoch [    6/   10] | d_loss: 1.8959 | g_loss: 1.5858
```

```
Epoch [    6/   10] | d_loss: 1.8578 | g_loss: 0.8888
Epoch [    6/   10] | d_loss: 0.9584 | g_loss: 0.7639
Epoch [    6/   10] | d_loss: 0.3678 | g_loss: 1.1231
Epoch [    6/   10] | d_loss: 0.6728 | g_loss: 2.0519
Epoch [    6/   10] | d_loss: 1.5112 | g_loss: 0.6726
Epoch [    6/   10] | d_loss: 1.2238 | g_loss: 2.6889
Epoch [    6/   10] | d_loss: 1.1194 | g_loss: 1.3246
Epoch [    6/   10] | d_loss: 0.5974 | g_loss: 3.1711
Epoch [    6/   10] | d_loss: 0.6485 | g_loss: 1.0877
Epoch [    6/   10] | d_loss: 1.1482 | g_loss: 1.9406
Epoch [    6/   10] | d_loss: 0.7956 | g_loss: 1.9278
Epoch [    6/   10] | d_loss: 1.1536 | g_loss: 2.1339
Epoch [    6/   10] | d_loss: 0.5177 | g_loss: 2.0163
Epoch [    6/   10] | d_loss: 1.0867 | g_loss: 1.5767
Epoch [    6/   10] | d_loss: 0.7795 | g_loss: 1.5803
Epoch [    6/   10] | d_loss: 0.7052 | g_loss: 1.0140
Epoch [    6/   10] | d_loss: 0.9675 | g_loss: 1.4741
Epoch [    6/   10] | d_loss: 0.5911 | g_loss: 1.4588
Epoch [    6/   10] | d_loss: 0.6503 | g_loss: 2.1733
Epoch [    6/   10] | d_loss: 0.6057 | g_loss: 2.2397
Epoch [    6/   10] | d_loss: 0.3294 | g_loss: 2.6504
Epoch [    6/   10] | d_loss: 0.7181 | g_loss: 1.7079
Epoch [    6/   10] | d_loss: 0.9719 | g_loss: 2.2286
Epoch [    6/   10] | d_loss: 1.0388 | g_loss: 1.5492
Epoch [    6/   10] | d_loss: 0.9217 | g_loss: 1.2295
Epoch [    6/   10] | d_loss: 0.9972 | g_loss: 2.5132
Epoch [    6/   10] | d_loss: 0.8831 | g_loss: 2.3872
Epoch [    6/   10] | d_loss: 1.4501 | g_loss: 2.1100
Epoch [    6/   10] | d_loss: 0.7529 | g_loss: 1.1389
Epoch [    6/   10] | d_loss: 0.5522 | g_loss: 2.7998
Epoch [    6/   10] | d_loss: 1.4511 | g_loss: 2.3681
Epoch [    6/   10] | d_loss: 0.9207 | g_loss: 1.8509
Epoch [    6/   10] | d_loss: 1.0050 | g_loss: 2.5140
Epoch [    6/   10] | d_loss: 1.9369 | g_loss: 1.9291
Epoch [    6/   10] | d_loss: 0.6214 | g_loss: 1.3777
Epoch [    6/   10] | d_loss: 0.6676 | g_loss: 1.2284
Epoch [    6/   10] | d_loss: 0.9400 | g_loss: 1.0463
Epoch [    6/   10] | d_loss: 0.7218 | g_loss: 2.0530
Epoch [    6/   10] | d_loss: 1.0717 | g_loss: 1.9356
Epoch [    6/   10] | d_loss: 1.3256 | g_loss: 1.3650
Epoch [    6/   10] | d_loss: 0.9959 | g_loss: 1.0122
Epoch [    6/   10] | d_loss: 1.0244 | g_loss: 0.9735
Epoch [    6/   10] | d_loss: 0.6719 | g_loss: 1.7960
Epoch [    6/   10] | d_loss: 0.7088 | g_loss: 1.9801
Epoch [    6/   10] | d_loss: 0.6638 | g_loss: 1.5116
Epoch [    6/   10] | d_loss: 0.8131 | g_loss: 2.7799
Epoch [    6/   10] | d_loss: 0.4599 | g_loss: 1.6609
Epoch [    6/   10] | d_loss: 0.5289 | g_loss: 2.1029
```

```
Epoch [    6/   10] | d_loss: 0.3911 | g_loss: 1.8151
Epoch [    6/   10] | d_loss: 0.9728 | g_loss: 2.3865
Epoch [    6/   10] | d_loss: 1.4331 | g_loss: 3.1804
Epoch [    6/   10] | d_loss: 1.7477 | g_loss: 1.9537
Epoch [    6/   10] | d_loss: 0.3743 | g_loss: 2.4501
Epoch [    6/   10] | d_loss: 1.4811 | g_loss: 1.0417
Epoch [    6/   10] | d_loss: 1.6020 | g_loss: 1.2445
Epoch [    6/   10] | d_loss: 0.8938 | g_loss: 1.9133
Epoch [    6/   10] | d_loss: 0.4381 | g_loss: 0.5416
Epoch [    6/   10] | d_loss: 0.6358 | g_loss: 2.3125
Epoch [    6/   10] | d_loss: 0.5751 | g_loss: 1.2688
Epoch [    6/   10] | d_loss: 0.7420 | g_loss: 1.5336
Epoch [    6/   10] | d_loss: 1.0336 | g_loss: 2.1683
Epoch [    6/   10] | d_loss: 0.8788 | g_loss: 0.7717
Epoch [    6/   10] | d_loss: 0.9776 | g_loss: 1.0694
Epoch [    6/   10] | d_loss: 0.9040 | g_loss: 1.4337
Epoch [    6/   10] | d_loss: 1.1538 | g_loss: 1.1800
Epoch [    6/   10] | d_loss: 0.4627 | g_loss: 2.2263
Epoch [    6/   10] | d_loss: 0.9503 | g_loss: 2.6689
Epoch [    6/   10] | d_loss: 0.4158 | g_loss: 1.6264
Epoch [    6/   10] | d_loss: 1.1428 | g_loss: 2.1964
Epoch [    6/   10] | d_loss: 0.9313 | g_loss: 2.2854
Epoch [    6/   10] | d_loss: 1.1153 | g_loss: 1.6482
Epoch [    6/   10] | d_loss: 1.0487 | g_loss: 1.1411
Epoch [    6/   10] | d_loss: 1.3513 | g_loss: 1.8494
Epoch [    6/   10] | d_loss: 0.9619 | g_loss: 1.3019
Epoch [    6/   10] | d_loss: 1.5933 | g_loss: 1.0450
Epoch [    6/   10] | d_loss: 0.4145 | g_loss: 1.3245
Epoch [    6/   10] | d_loss: 0.9160 | g_loss: 0.6952
Epoch [    6/   10] | d_loss: 0.7498 | g_loss: 1.3424
Epoch [    6/   10] | d_loss: 0.5712 | g_loss: 1.8602
Epoch [    6/   10] | d_loss: 1.1057 | g_loss: 1.5860
Epoch [    6/   10] | d_loss: 0.3822 | g_loss: 0.9132
Epoch [    7/   10] | d_loss: 0.7271 | g_loss: 1.6217
Epoch [    7/   10] | d_loss: 0.7827 | g_loss: 0.6475
Epoch [    7/   10] | d_loss: 1.2350 | g_loss: 1.7717
Epoch [    7/   10] | d_loss: 1.2344 | g_loss: 1.8899
Epoch [    7/   10] | d_loss: 1.0409 | g_loss: 0.7569
Epoch [    7/   10] | d_loss: 0.9128 | g_loss: 1.0516
Epoch [    7/   10] | d_loss: 0.5883 | g_loss: 2.0602
Epoch [    7/   10] | d_loss: 0.4755 | g_loss: 1.8545
Epoch [    7/   10] | d_loss: 1.7880 | g_loss: 2.1898
Epoch [    7/   10] | d_loss: 1.1128 | g_loss: 2.2575
Epoch [    7/   10] | d_loss: 1.0296 | g_loss: 1.5661
Epoch [    7/   10] | d_loss: 1.3867 | g_loss: 1.5386
Epoch [    7/   10] | d_loss: 1.4130 | g_loss: 1.7142
Epoch [    7/   10] | d_loss: 1.6220 | g_loss: 2.7156
Epoch [    7/   10] | d_loss: 1.0585 | g_loss: 1.7828
```

```
Epoch [    7/   10] | d_loss: 0.2785 | g_loss: 1.3857
Epoch [    7/   10] | d_loss: 0.7628 | g_loss: 2.1358
Epoch [    7/   10] | d_loss: 0.4366 | g_loss: 0.9935
Epoch [    7/   10] | d_loss: 1.3216 | g_loss: 1.2990
Epoch [    7/   10] | d_loss: 1.0972 | g_loss: 2.9182
Epoch [    7/   10] | d_loss: 1.1646 | g_loss: 1.5439
Epoch [    7/   10] | d_loss: 1.4908 | g_loss: 1.0536
Epoch [    7/   10] | d_loss: 0.7951 | g_loss: 1.2971
Epoch [    7/   10] | d_loss: 0.9910 | g_loss: 3.5327
Epoch [    7/   10] | d_loss: 0.7094 | g_loss: 1.2395
Epoch [    7/   10] | d_loss: 1.8504 | g_loss: 1.4879
Epoch [    7/   10] | d_loss: 0.6332 | g_loss: 2.1100
Epoch [    7/   10] | d_loss: 0.8184 | g_loss: 3.0048
Epoch [    7/   10] | d_loss: 0.9481 | g_loss: 1.4509
Epoch [    7/   10] | d_loss: 1.3892 | g_loss: 1.4813
Epoch [    7/   10] | d_loss: 0.4275 | g_loss: 1.7704
Epoch [    7/   10] | d_loss: 1.8681 | g_loss: 2.8185
Epoch [    7/   10] | d_loss: 0.9678 | g_loss: 0.8730
Epoch [    7/   10] | d_loss: 0.9722 | g_loss: 2.7843
Epoch [    7/   10] | d_loss: 0.8836 | g_loss: 2.4510
Epoch [    7/   10] | d_loss: 0.8343 | g_loss: 1.9504
Epoch [    7/   10] | d_loss: 0.8136 | g_loss: 1.6529
Epoch [    7/   10] | d_loss: 1.5739 | g_loss: 0.7590
Epoch [    7/   10] | d_loss: 0.6997 | g_loss: 1.1344
Epoch [    7/   10] | d_loss: 0.4370 | g_loss: 1.9532
Epoch [    7/   10] | d_loss: 0.6037 | g_loss: 1.8138
Epoch [    7/   10] | d_loss: 0.5307 | g_loss: 2.5431
Epoch [    7/   10] | d_loss: 1.1014 | g_loss: 1.7913
Epoch [    7/   10] | d_loss: 0.6164 | g_loss: 0.9741
Epoch [    7/   10] | d_loss: 0.5860 | g_loss: 1.0301
Epoch [    7/   10] | d_loss: 1.0810 | g_loss: 1.2147
Epoch [    7/   10] | d_loss: 0.4272 | g_loss: 1.4698
Epoch [    7/   10] | d_loss: 1.0323 | g_loss: 2.2823
Epoch [    7/   10] | d_loss: 0.9049 | g_loss: 1.7446
Epoch [    7/   10] | d_loss: 0.9012 | g_loss: 1.2777
Epoch [    7/   10] | d_loss: 1.2145 | g_loss: 1.2961
Epoch [    7/   10] | d_loss: 0.6243 | g_loss: 2.4842
Epoch [    7/   10] | d_loss: 0.8566 | g_loss: 2.6523
Epoch [    7/   10] | d_loss: 0.6723 | g_loss: 1.5318
Epoch [    7/   10] | d_loss: 0.5809 | g_loss: 1.6777
Epoch [    7/   10] | d_loss: 0.7234 | g_loss: 1.8118
Epoch [    7/   10] | d_loss: 1.0768 | g_loss: 1.3019
Epoch [    7/   10] | d_loss: 0.2490 | g_loss: 2.1094
Epoch [    7/   10] | d_loss: 0.8475 | g_loss: 1.6520
Epoch [    7/   10] | d_loss: 0.5549 | g_loss: 1.8005
Epoch [    7/   10] | d_loss: 1.3352 | g_loss: 1.1700
Epoch [    7/   10] | d_loss: 0.8281 | g_loss: 1.4018
Epoch [    7/   10] | d_loss: 0.9985 | g_loss: 2.7314
```

```
Epoch [    7/   10] | d_loss: 1.1129 | g_loss: 1.2226
Epoch [    7/   10] | d_loss: 1.2877 | g_loss: 3.1141
Epoch [    7/   10] | d_loss: 1.0375 | g_loss: 2.3318
Epoch [    7/   10] | d_loss: 0.7099 | g_loss: 1.2404
Epoch [    7/   10] | d_loss: 0.8821 | g_loss: 1.0305
Epoch [    7/   10] | d_loss: 0.5517 | g_loss: 0.8410
Epoch [    7/   10] | d_loss: 1.1531 | g_loss: 1.7852
Epoch [    7/   10] | d_loss: 0.9795 | g_loss: 2.2440
Epoch [    7/   10] | d_loss: 1.4185 | g_loss: 3.6400
Epoch [    7/   10] | d_loss: 1.5970 | g_loss: 1.5769
Epoch [    7/   10] | d_loss: 0.6078 | g_loss: 2.7808
Epoch [    7/   10] | d_loss: 0.8604 | g_loss: 1.4773
Epoch [    7/   10] | d_loss: 0.2924 | g_loss: 2.0738
Epoch [    7/   10] | d_loss: 1.0308 | g_loss: 1.5060
Epoch [    7/   10] | d_loss: 0.7151 | g_loss: 0.9210
Epoch [    7/   10] | d_loss: 0.6067 | g_loss: 1.7722
Epoch [    7/   10] | d_loss: 1.1525 | g_loss: 1.2840
Epoch [    7/   10] | d_loss: 0.4157 | g_loss: 2.5741
Epoch [    7/   10] | d_loss: 0.7669 | g_loss: 1.1074
Epoch [    7/   10] | d_loss: 0.4336 | g_loss: 2.8630
Epoch [    7/   10] | d_loss: 0.5781 | g_loss: 1.3904
Epoch [    7/   10] | d_loss: 0.5227 | g_loss: 1.0615
Epoch [    7/   10] | d_loss: 1.3942 | g_loss: 3.0023
Epoch [    7/   10] | d_loss: 0.7205 | g_loss: 2.1578
Epoch [    7/   10] | d_loss: 0.5515 | g_loss: 1.7875
Epoch [    7/   10] | d_loss: 0.4051 | g_loss: 1.9601
Epoch [    7/   10] | d_loss: 0.3578 | g_loss: 1.7808
Epoch [    7/   10] | d_loss: 0.3140 | g_loss: 2.0839
Epoch [    7/   10] | d_loss: 0.7981 | g_loss: 1.4898
Epoch [    7/   10] | d_loss: 1.2857 | g_loss: 2.4852
Epoch [    7/   10] | d_loss: 1.8538 | g_loss: 1.2343
Epoch [    7/   10] | d_loss: 0.7544 | g_loss: 1.0284
Epoch [    7/   10] | d_loss: 0.9640 | g_loss: 1.1378
Epoch [    7/   10] | d_loss: 0.8477 | g_loss: 3.3688
Epoch [    7/   10] | d_loss: 0.8704 | g_loss: 3.5881
Epoch [    7/   10] | d_loss: 0.9585 | g_loss: 2.4453
Epoch [    7/   10] | d_loss: 0.4768 | g_loss: 1.6425
Epoch [    8/   10] | d_loss: 2.9378 | g_loss: 2.9804
Epoch [    8/   10] | d_loss: 0.6097 | g_loss: 1.6343
Epoch [    8/   10] | d_loss: 0.4979 | g_loss: 2.0315
Epoch [    8/   10] | d_loss: 1.1367 | g_loss: 0.8854
Epoch [    8/   10] | d_loss: 0.6236 | g_loss: 2.7084
Epoch [    8/   10] | d_loss: 1.6523 | g_loss: 1.7214
Epoch [    8/   10] | d_loss: 0.4980 | g_loss: 2.2070
Epoch [    8/   10] | d_loss: 0.3637 | g_loss: 1.5246
Epoch [    8/   10] | d_loss: 1.1293 | g_loss: 1.3270
Epoch [    8/   10] | d_loss: 0.3432 | g_loss: 2.8539
Epoch [    8/   10] | d_loss: 0.4973 | g_loss: 2.1048
```

```
Epoch [    8/   10] | d_loss: 0.5901 | g_loss: 1.6526
Epoch [    8/   10] | d_loss: 1.3476 | g_loss: 0.6706
Epoch [    8/   10] | d_loss: 0.5845 | g_loss: 2.2904
Epoch [    8/   10] | d_loss: 0.3091 | g_loss: 1.5830
Epoch [    8/   10] | d_loss: 0.9885 | g_loss: 1.6297
Epoch [    8/   10] | d_loss: 0.5983 | g_loss: 2.2942
Epoch [    8/   10] | d_loss: 0.5554 | g_loss: 1.3681
Epoch [    8/   10] | d_loss: 1.0854 | g_loss: 2.4241
Epoch [    8/   10] | d_loss: 0.5710 | g_loss: 2.3656
Epoch [    8/   10] | d_loss: 0.4387 | g_loss: 2.7304
Epoch [    8/   10] | d_loss: 1.2052 | g_loss: 1.8789
Epoch [    8/   10] | d_loss: 1.2074 | g_loss: 1.0615
Epoch [    8/   10] | d_loss: 0.6868 | g_loss: 1.5824
Epoch [    8/   10] | d_loss: 0.4926 | g_loss: 3.4491
Epoch [    8/   10] | d_loss: 0.4231 | g_loss: 1.5217
Epoch [    8/   10] | d_loss: 1.0428 | g_loss: 2.1677
Epoch [    8/   10] | d_loss: 0.2972 | g_loss: 1.2946
Epoch [    8/   10] | d_loss: 0.8041 | g_loss: 3.1981
Epoch [    8/   10] | d_loss: 0.8686 | g_loss: 1.7269
Epoch [    8/   10] | d_loss: 0.5085 | g_loss: 2.6916
Epoch [    8/   10] | d_loss: 0.3462 | g_loss: 2.3134
Epoch [    8/   10] | d_loss: 1.7360 | g_loss: 2.9243
Epoch [    8/   10] | d_loss: 0.9660 | g_loss: 1.7745
Epoch [    8/   10] | d_loss: 0.7703 | g_loss: 2.1716
Epoch [    8/   10] | d_loss: 1.0452 | g_loss: 3.7102
Epoch [    8/   10] | d_loss: 1.0933 | g_loss: 3.0051
Epoch [    8/   10] | d_loss: 0.7725 | g_loss: 2.0359
Epoch [    8/   10] | d_loss: 0.6906 | g_loss: 3.4893
Epoch [    8/   10] | d_loss: 0.8943 | g_loss: 1.7152
Epoch [    8/   10] | d_loss: 1.0338 | g_loss: 1.9187
Epoch [    8/   10] | d_loss: 0.7761 | g_loss: 1.3549
Epoch [    8/   10] | d_loss: 1.1011 | g_loss: 2.1753
Epoch [    8/   10] | d_loss: 0.8745 | g_loss: 2.4948
Epoch [    8/   10] | d_loss: 0.7267 | g_loss: 3.8147
Epoch [    8/   10] | d_loss: 0.9170 | g_loss: 1.9722
Epoch [    8/   10] | d_loss: 0.3825 | g_loss: 2.7486
Epoch [    8/   10] | d_loss: 0.4469 | g_loss: 2.1458
Epoch [    8/   10] | d_loss: 1.1951 | g_loss: 1.5294
Epoch [    8/   10] | d_loss: 2.3043 | g_loss: 0.7101
Epoch [    8/   10] | d_loss: 0.7455 | g_loss: 1.4687
Epoch [    8/   10] | d_loss: 0.2634 | g_loss: 1.5691
Epoch [    8/   10] | d_loss: 1.0769 | g_loss: 1.0944
Epoch [    8/   10] | d_loss: 0.8411 | g_loss: 1.3633
Epoch [    8/   10] | d_loss: 0.6130 | g_loss: 1.8845
Epoch [    8/   10] | d_loss: 0.5217 | g_loss: 2.2191
Epoch [    8/   10] | d_loss: 0.3655 | g_loss: 3.1503
Epoch [    8/   10] | d_loss: 1.2430 | g_loss: 1.7690
Epoch [    8/   10] | d_loss: 0.4233 | g_loss: 1.2055
```

```
Epoch [    8/   10] | d_loss: 0.4025 | g_loss: 2.7103
Epoch [    8/   10] | d_loss: 1.1004 | g_loss: 0.8154
Epoch [    8/   10] | d_loss: 0.6026 | g_loss: 3.3108
Epoch [    8/   10] | d_loss: 1.3533 | g_loss: 1.8102
Epoch [    8/   10] | d_loss: 0.7400 | g_loss: 1.7952
Epoch [    8/   10] | d_loss: 1.1139 | g_loss: 1.8628
Epoch [    8/   10] | d_loss: 1.5291 | g_loss: 4.0243
Epoch [    8/   10] | d_loss: 0.4095 | g_loss: 1.2706
Epoch [    8/   10] | d_loss: 0.6405 | g_loss: 1.8724
Epoch [    8/   10] | d_loss: 0.7140 | g_loss: 1.4117
Epoch [    8/   10] | d_loss: 0.7494 | g_loss: 2.5444
Epoch [    8/   10] | d_loss: 0.8427 | g_loss: 1.3891
Epoch [    8/   10] | d_loss: 0.9890 | g_loss: 2.3182
Epoch [    8/   10] | d_loss: 0.7655 | g_loss: 1.5913
Epoch [    8/   10] | d_loss: 1.2147 | g_loss: 2.3298
Epoch [    8/   10] | d_loss: 0.5738 | g_loss: 2.9920
Epoch [    8/   10] | d_loss: 0.9584 | g_loss: 2.6382
Epoch [    8/   10] | d_loss: 0.5580 | g_loss: 1.0629
Epoch [    8/   10] | d_loss: 0.4603 | g_loss: 2.4527
Epoch [    8/   10] | d_loss: 0.8099 | g_loss: 1.4474
Epoch [    8/   10] | d_loss: 0.4595 | g_loss: 1.1441
Epoch [    8/   10] | d_loss: 0.9995 | g_loss: 1.4232
Epoch [    8/   10] | d_loss: 1.8931 | g_loss: 2.1789
Epoch [    8/   10] | d_loss: 0.9322 | g_loss: 0.8737
Epoch [    8/   10] | d_loss: 0.8810 | g_loss: 1.6864
Epoch [    8/   10] | d_loss: 0.4863 | g_loss: 1.7784
Epoch [    8/   10] | d_loss: 0.8927 | g_loss: 1.1257
Epoch [    8/   10] | d_loss: 0.4231 | g_loss: 1.0993
Epoch [    8/   10] | d_loss: 1.4181 | g_loss: 2.1841
Epoch [    8/   10] | d_loss: 0.2415 | g_loss: 1.2264
Epoch [    8/   10] | d_loss: 0.3946 | g_loss: 2.1632
Epoch [    8/   10] | d_loss: 0.6945 | g_loss: 2.0483
Epoch [    8/   10] | d_loss: 1.0866 | g_loss: 1.5370
Epoch [    8/   10] | d_loss: 1.2064 | g_loss: 1.9515
Epoch [    8/   10] | d_loss: 0.7395 | g_loss: 0.7345
Epoch [    8/   10] | d_loss: 1.0343 | g_loss: 0.5984
Epoch [    8/   10] | d_loss: 1.0966 | g_loss: 2.9855
Epoch [    8/   10] | d_loss: 0.8260 | g_loss: 2.2644
Epoch [    8/   10] | d_loss: 1.0635 | g_loss: 1.6769
Epoch [    8/   10] | d_loss: 1.3239 | g_loss: 2.5696
Epoch [    8/   10] | d_loss: 1.0901 | g_loss: 1.4004
Epoch [    9/   10] | d_loss: 0.9321 | g_loss: 3.1480
Epoch [    9/   10] | d_loss: 1.1239 | g_loss: 2.5243
Epoch [    9/   10] | d_loss: 0.9767 | g_loss: 2.8008
Epoch [    9/   10] | d_loss: 1.4319 | g_loss: 1.2321
Epoch [    9/   10] | d_loss: 0.3834 | g_loss: 2.3047
Epoch [    9/   10] | d_loss: 0.5417 | g_loss: 1.7258
Epoch [    9/   10] | d_loss: 0.7926 | g_loss: 1.8377
```

```
Epoch [    9/   10] | d_loss: 0.4733 | g_loss: 2.9957
Epoch [    9/   10] | d_loss: 0.9174 | g_loss: 2.3099
Epoch [    9/   10] | d_loss: 1.2035 | g_loss: 2.5183
Epoch [    9/   10] | d_loss: 1.5566 | g_loss: 0.7784
Epoch [    9/   10] | d_loss: 0.8621 | g_loss: 1.1806
Epoch [    9/   10] | d_loss: 1.4247 | g_loss: 1.7312
Epoch [    9/   10] | d_loss: 1.9501 | g_loss: 1.9274
Epoch [    9/   10] | d_loss: 0.8708 | g_loss: 1.5998
Epoch [    9/   10] | d_loss: 0.6810 | g_loss: 2.6343
Epoch [    9/   10] | d_loss: 0.6387 | g_loss: 1.6795
Epoch [    9/   10] | d_loss: 0.5452 | g_loss: 1.4072
Epoch [    9/   10] | d_loss: 0.1708 | g_loss: 2.2024
Epoch [    9/   10] | d_loss: 0.2877 | g_loss: 2.0244
Epoch [    9/   10] | d_loss: 0.4922 | g_loss: 3.0152
Epoch [    9/   10] | d_loss: 0.9586 | g_loss: 1.0167
Epoch [    9/   10] | d_loss: 1.6191 | g_loss: 2.0786
Epoch [    9/   10] | d_loss: 1.7914 | g_loss: 2.9861
Epoch [    9/   10] | d_loss: 0.5655 | g_loss: 1.3467
Epoch [    9/   10] | d_loss: 0.4552 | g_loss: 3.0542
Epoch [    9/   10] | d_loss: 1.0322 | g_loss: 2.0347
Epoch [    9/   10] | d_loss: 1.1033 | g_loss: 1.8657
Epoch [    9/   10] | d_loss: 0.4539 | g_loss: 2.7818
Epoch [    9/   10] | d_loss: 0.7398 | g_loss: 1.5730
Epoch [    9/   10] | d_loss: 1.3243 | g_loss: 0.6444
Epoch [    9/   10] | d_loss: 0.5161 | g_loss: 1.3848
Epoch [    9/   10] | d_loss: 0.3724 | g_loss: 1.9004
Epoch [    9/   10] | d_loss: 0.5955 | g_loss: 1.5537
Epoch [    9/   10] | d_loss: 1.0551 | g_loss: 1.1622
Epoch [    9/   10] | d_loss: 0.5970 | g_loss: 1.9078
Epoch [    9/   10] | d_loss: 0.2359 | g_loss: 1.6797
Epoch [    9/   10] | d_loss: 0.8216 | g_loss: 1.7419
Epoch [    9/   10] | d_loss: 1.1218 | g_loss: 1.4767
Epoch [    9/   10] | d_loss: 0.5889 | g_loss: 1.8578
Epoch [    9/   10] | d_loss: 0.1825 | g_loss: 1.9152
Epoch [    9/   10] | d_loss: 0.6550 | g_loss: 1.2600
Epoch [    9/   10] | d_loss: 1.2630 | g_loss: 2.0470
Epoch [    9/   10] | d_loss: 0.6999 | g_loss: 1.4144
Epoch [    9/   10] | d_loss: 2.2633 | g_loss: 3.4960
Epoch [    9/   10] | d_loss: 1.1029 | g_loss: 0.7504
Epoch [    9/   10] | d_loss: 0.6944 | g_loss: 1.8402
Epoch [    9/   10] | d_loss: 0.5141 | g_loss: 2.0686
Epoch [    9/   10] | d_loss: 0.1364 | g_loss: 2.7845
Epoch [    9/   10] | d_loss: 0.9466 | g_loss: 2.5263
Epoch [    9/   10] | d_loss: 1.0108 | g_loss: 2.2659
Epoch [    9/   10] | d_loss: 0.7348 | g_loss: 2.0750
Epoch [    9/   10] | d_loss: 0.6608 | g_loss: 1.5533
Epoch [    9/   10] | d_loss: 0.7609 | g_loss: 1.9102
Epoch [    9/   10] | d_loss: 0.9414 | g_loss: 2.5171
```

```
Epoch [    9/   10] | d_loss: 0.3575 | g_loss: 1.9324
Epoch [    9/   10] | d_loss: 0.8971 | g_loss: 1.8939
Epoch [    9/   10] | d_loss: 0.6388 | g_loss: 2.2678
Epoch [    9/   10] | d_loss: 1.3157 | g_loss: 1.3496
Epoch [    9/   10] | d_loss: 1.4573 | g_loss: 2.8529
Epoch [    9/   10] | d_loss: 0.3875 | g_loss: 2.4588
Epoch [    9/   10] | d_loss: 0.2258 | g_loss: 1.3081
Epoch [    9/   10] | d_loss: 0.6827 | g_loss: 0.5956
Epoch [    9/   10] | d_loss: 1.1255 | g_loss: 1.9232
Epoch [    9/   10] | d_loss: 0.7694 | g_loss: 2.6143
Epoch [    9/   10] | d_loss: 0.3538 | g_loss: 1.7059
Epoch [    9/   10] | d_loss: 1.0517 | g_loss: 2.6027
Epoch [    9/   10] | d_loss: 0.6363 | g_loss: 1.5649
Epoch [    9/   10] | d_loss: 1.1753 | g_loss: 2.1410
Epoch [    9/   10] | d_loss: 1.2480 | g_loss: 1.0059
Epoch [    9/   10] | d_loss: 0.8798 | g_loss: 1.7851
Epoch [    9/   10] | d_loss: 0.5981 | g_loss: 2.4322
Epoch [    9/   10] | d_loss: 0.5923 | g_loss: 1.9523
Epoch [    9/   10] | d_loss: 0.4431 | g_loss: 2.5481
Epoch [    9/   10] | d_loss: 0.8771 | g_loss: 2.0252
Epoch [    9/   10] | d_loss: 1.3419 | g_loss: 2.3829
Epoch [    9/   10] | d_loss: 0.9360 | g_loss: 2.7776
Epoch [    9/   10] | d_loss: 1.4222 | g_loss: 1.0133
Epoch [    9/   10] | d_loss: 0.8483 | g_loss: 1.1853
Epoch [    9/   10] | d_loss: 1.7422 | g_loss: 3.7323
Epoch [    9/   10] | d_loss: 1.5545 | g_loss: 1.6867
Epoch [    9/   10] | d_loss: 0.7861 | g_loss: 2.8806
Epoch [    9/   10] | d_loss: 0.8485 | g_loss: 2.3086
Epoch [    9/   10] | d_loss: 0.4572 | g_loss: 1.4642
Epoch [    9/   10] | d_loss: 0.3658 | g_loss: 2.7106
Epoch [    9/   10] | d_loss: 1.2979 | g_loss: 1.2219
Epoch [    9/   10] | d_loss: 0.8354 | g_loss: 2.3110
Epoch [    9/   10] | d_loss: 0.3110 | g_loss: 1.4825
Epoch [    9/   10] | d_loss: 0.5451 | g_loss: 1.2895
Epoch [    9/   10] | d_loss: 0.3220 | g_loss: 2.5254
Epoch [    9/   10] | d_loss: 2.1999 | g_loss: 0.9681
Epoch [    9/   10] | d_loss: 0.3138 | g_loss: 3.5914
Epoch [    9/   10] | d_loss: 1.0146 | g_loss: 0.4605
Epoch [    9/   10] | d_loss: 0.8892 | g_loss: 1.2834
Epoch [    9/   10] | d_loss: 0.8127 | g_loss: 2.4615
Epoch [    9/   10] | d_loss: 1.4326 | g_loss: 1.1746
Epoch [    9/   10] | d_loss: 0.5522 | g_loss: 1.9938
Epoch [    9/   10] | d_loss: 0.6867 | g_loss: 2.7799
Epoch [    9/   10] | d_loss: 1.5875 | g_loss: 0.7334
Epoch [    9/   10] | d_loss: 0.5499 | g_loss: 1.4200
Epoch [   10/   10] | d_loss: 2.8665 | g_loss: 1.4005
Epoch [   10/   10] | d_loss: 0.2722 | g_loss: 1.4458
Epoch [   10/   10] | d_loss: 0.8834 | g_loss: 0.9132
```

```
Epoch [   10/   10] | d_loss: 0.2268 | g_loss: 2.5759
Epoch [   10/   10] | d_loss: 1.2287 | g_loss: 3.0111
Epoch [   10/   10] | d_loss: 0.9425 | g_loss: 2.0335
Epoch [   10/   10] | d_loss: 0.8316 | g_loss: 1.3084
Epoch [   10/   10] | d_loss: 0.5183 | g_loss: 1.6841
Epoch [   10/   10] | d_loss: 0.6788 | g_loss: 1.2244
Epoch [   10/   10] | d_loss: 0.5956 | g_loss: 2.6149
Epoch [   10/   10] | d_loss: 0.9978 | g_loss: 0.7222
Epoch [   10/   10] | d_loss: 0.9038 | g_loss: 2.6062
Epoch [   10/   10] | d_loss: 0.3941 | g_loss: 1.4206
Epoch [   10/   10] | d_loss: 0.5781 | g_loss: 2.4631
Epoch [   10/   10] | d_loss: 0.7909 | g_loss: 2.0529
Epoch [   10/   10] | d_loss: 0.6050 | g_loss: 1.3431
Epoch [   10/   10] | d_loss: 0.8335 | g_loss: 1.7802
Epoch [   10/   10] | d_loss: 1.3020 | g_loss: 1.0596
Epoch [   10/   10] | d_loss: 0.9748 | g_loss: 2.1436
Epoch [   10/   10] | d_loss: 0.3386 | g_loss: 1.8713
Epoch [   10/   10] | d_loss: 1.4429 | g_loss: 1.0957
Epoch [   10/   10] | d_loss: 0.7899 | g_loss: 2.1241
Epoch [   10/   10] | d_loss: 0.4298 | g_loss: 2.6489
Epoch [   10/   10] | d_loss: 0.4039 | g_loss: 1.1619
Epoch [   10/   10] | d_loss: 2.0091 | g_loss: 2.3962
Epoch [   10/   10] | d_loss: 1.1100 | g_loss: 1.8406
Epoch [   10/   10] | d_loss: 0.4590 | g_loss: 1.3664
Epoch [   10/   10] | d_loss: 0.2132 | g_loss: 1.0531
Epoch [   10/   10] | d_loss: 1.2777 | g_loss: 1.9357
Epoch [   10/   10] | d_loss: 0.8609 | g_loss: 1.8953
Epoch [   10/   10] | d_loss: 0.6688 | g_loss: 1.0325
Epoch [   10/   10] | d_loss: 1.1353 | g_loss: 1.8040
Epoch [   10/   10] | d_loss: 0.2493 | g_loss: 3.0940
Epoch [   10/   10] | d_loss: 0.6083 | g_loss: 1.8781
Epoch [   10/   10] | d_loss: 0.7405 | g_loss: 1.7332
Epoch [   10/   10] | d_loss: 0.7258 | g_loss: 1.9775
Epoch [   10/   10] | d_loss: 0.2663 | g_loss: 1.9508
Epoch [   10/   10] | d_loss: 2.8762 | g_loss: 1.4337
Epoch [   10/   10] | d_loss: 0.4392 | g_loss: 0.8164
Epoch [   10/   10] | d_loss: 1.8846 | g_loss: 1.8776
Epoch [   10/   10] | d_loss: 0.6628 | g_loss: 0.8596
Epoch [   10/   10] | d_loss: 0.5123 | g_loss: 1.0918
Epoch [   10/   10] | d_loss: 0.4224 | g_loss: 2.1514
Epoch [   10/   10] | d_loss: 0.6825 | g_loss: 0.9565
Epoch [   10/   10] | d_loss: 1.9469 | g_loss: 1.2477
Epoch [   10/   10] | d_loss: 0.8199 | g_loss: 3.2338
Epoch [   10/   10] | d_loss: 1.0098 | g_loss: 1.3360
Epoch [   10/   10] | d_loss: 0.2010 | g_loss: 2.5307
Epoch [   10/   10] | d_loss: 0.8502 | g_loss: 2.8041
Epoch [   10/   10] | d_loss: 0.5666 | g_loss: 2.4284
Epoch [   10/   10] | d_loss: 0.8848 | g_loss: 1.2131
```

```
Epoch [   10/   10] | d_loss: 0.5240 | g_loss: 2.0538
Epoch [   10/   10] | d_loss: 0.6282 | g_loss: 2.7748
Epoch [   10/   10] | d_loss: 1.1031 | g_loss: 0.5856
Epoch [   10/   10] | d_loss: 0.7199 | g_loss: 2.4936
Epoch [   10/   10] | d_loss: 1.3112 | g_loss: 2.5385
Epoch [   10/   10] | d_loss: 0.7069 | g_loss: 2.3468
Epoch [   10/   10] | d_loss: 0.1247 | g_loss: 1.1054
Epoch [   10/   10] | d_loss: 0.5445 | g_loss: 2.8792
Epoch [   10/   10] | d_loss: 0.2235 | g_loss: 1.0629
Epoch [   10/   10] | d_loss: 1.4685 | g_loss: 2.6151
Epoch [   10/   10] | d_loss: 0.9871 | g_loss: 1.1990
Epoch [   10/   10] | d_loss: 0.5267 | g_loss: 1.2767
Epoch [   10/   10] | d_loss: 0.5458 | g_loss: 2.7297
Epoch [   10/   10] | d_loss: 1.1603 | g_loss: 2.0369
Epoch [   10/   10] | d_loss: 0.9981 | g_loss: 1.2787
Epoch [   10/   10] | d_loss: 0.3944 | g_loss: 3.5747
Epoch [   10/   10] | d_loss: 1.0510 | g_loss: 3.8643
Epoch [   10/   10] | d_loss: 0.8093 | g_loss: 0.7995
Epoch [   10/   10] | d_loss: 0.5618 | g_loss: 1.3025
Epoch [   10/   10] | d_loss: 0.7568 | g_loss: 1.9658
Epoch [   10/   10] | d_loss: 1.1235 | g_loss: 1.5379
Epoch [   10/   10] | d_loss: 0.7860 | g_loss: 1.9740
Epoch [   10/   10] | d_loss: 0.7712 | g_loss: 3.9938
Epoch [   10/   10] | d_loss: 0.6742 | g_loss: 2.1079
Epoch [   10/   10] | d_loss: 0.7714 | g_loss: 3.6654
Epoch [   10/   10] | d_loss: 0.9547 | g_loss: 1.1724
Epoch [   10/   10] | d_loss: 1.0860 | g_loss: 1.6711
Epoch [   10/   10] | d_loss: 1.1004 | g_loss: 1.0528
Epoch [   10/   10] | d_loss: 1.7372 | g_loss: 1.1823
Epoch [   10/   10] | d_loss: 0.4231 | g_loss: 2.1299
Epoch [   10/   10] | d_loss: 0.2075 | g_loss: 2.2749
Epoch [   10/   10] | d_loss: 1.2048 | g_loss: 2.5277
Epoch [   10/   10] | d_loss: 1.0253 | g_loss: 0.8609
Epoch [   10/   10] | d_loss: 0.5808 | g_loss: 1.6271
Epoch [   10/   10] | d_loss: 0.2108 | g_loss: 4.5083
Epoch [   10/   10] | d_loss: 0.7078 | g_loss: 2.0553
Epoch [   10/   10] | d_loss: 1.1755 | g_loss: 1.6351
Epoch [   10/   10] | d_loss: 0.1570 | g_loss: 2.6834
Epoch [   10/   10] | d_loss: 0.8121 | g_loss: 1.6935
Epoch [   10/   10] | d_loss: 1.5536 | g_loss: 1.3697
Epoch [   10/   10] | d_loss: 1.4764 | g_loss: 2.8801
Epoch [   10/   10] | d_loss: 1.0495 | g_loss: 2.5536
Epoch [   10/   10] | d_loss: 1.3047 | g_loss: 0.4902
Epoch [   10/   10] | d_loss: 0.9184 | g_loss: 1.0453
Epoch [   10/   10] | d_loss: 1.1294 | g_loss: 0.8916
Epoch [   10/   10] | d_loss: 0.8984 | g_loss: 2.5977
Epoch [   10/   10] | d_loss: 0.9602 | g_loss: 1.7041
Epoch [   10/   10] | d_loss: 0.7802 | g_loss: 2.6553
```

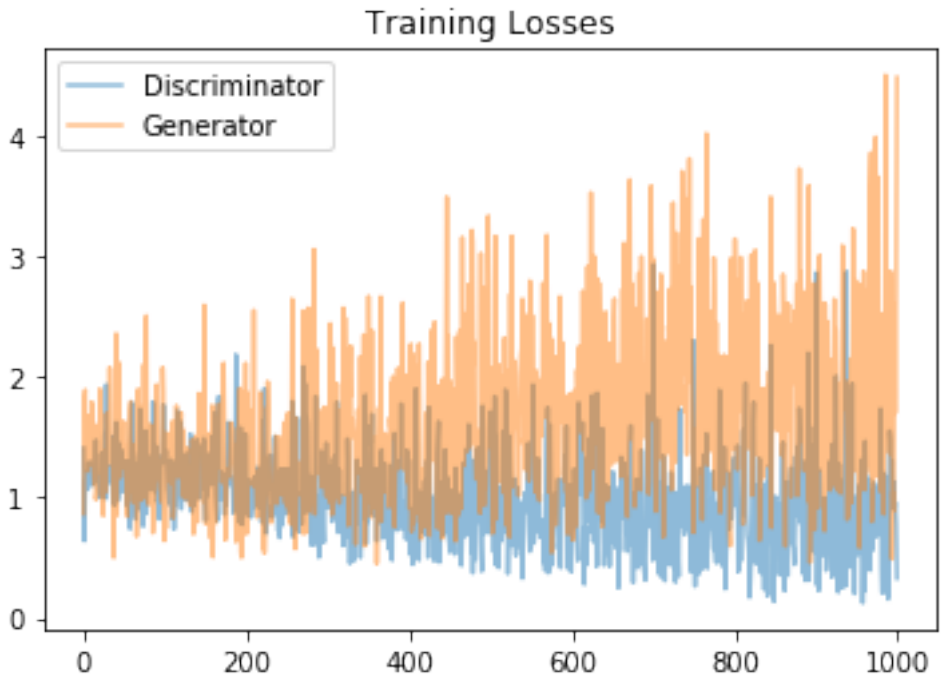
```
Epoch [   10/   10] | d_loss: 0.3297 | g_loss: 4.4885
```

2.8 Training loss

Plot the training losses for the generator and discriminator, recorded after each epoch.

```
In [19]: fig, ax = plt.subplots()
         losses = np.array(losses)
         plt.plot(losses.T[0], label='Discriminator', alpha=0.5)
         plt.plot(losses.T[1], label='Generator', alpha=0.5)
         plt.title("Training Losses")
         plt.legend()
```

```
Out[19]: <matplotlib.legend.Legend at 0x7f34e72916d8>
```



2.9 Generator samples from training

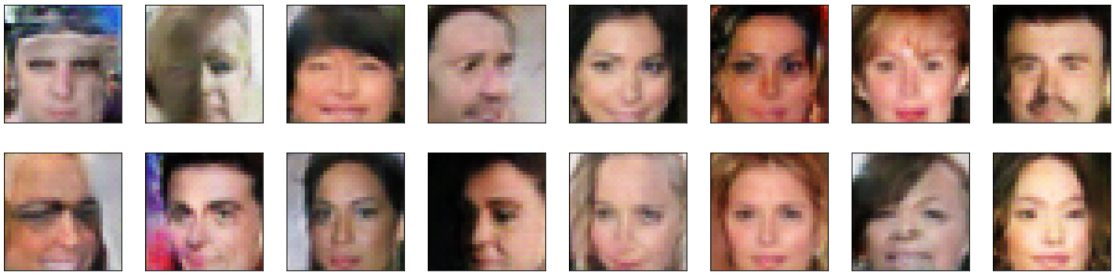
View samples of images from the generator, and answer a question about the strengths and weaknesses of your trained models.

```
In [20]: # helper function for viewing a list of passed in sample images
         def view_samples(epoch, samples):
             fig, axes = plt.subplots(figsize=(16,4), nrows=2, ncols=8, sharey=True, sharex=Tru
             for ax, img in zip(axes.flatten(), samples[epoch]):
```

```
        img = img.detach().cpu().numpy()
        img = np.transpose(img, (1, 2, 0))
        img = ((img + 1)*255 / (2)).astype(np.uint8)
        ax.xaxis.set_visible(False)
        ax.yaxis.set_visible(False)
        im = ax.imshow(img.reshape((32,32,3)))
```

```
In [21]: # Load samples from generator, taken while training
         with open('train_samples.pkl', 'rb') as f:
             samples = pkl.load(f)
```

```
In [22]: _ = view_samples(-1, samples)
```



2.9.1 Question: What do you notice about your generated samples and how might you improve this model?

When you answer this question, consider the following factors: * The dataset is biased; it is made of "celebrity" faces that are mostly white * Model size; larger models have the opportunity to learn more features in a data feature space * Optimization strategy; optimizers and number of epochs affect your final result

Answer: (Write your answer in this cell)

The GAN generated pictures of faces. I want to mention that they are under different angles and pictures are not damaged because of that. I might train for more than 10 epochs to further improve but there is a limit of epochs that depends of the spread between generator and discriminator batch size when no further improvement occurs. Model is deep as generator and as well as discriminator and they work at maximum capacity considering in the input size and output size.

Definitely, more variety of faces might help to train neural network better and generate a new type of faces.

Model size matters we have to ensure that our models reconginze and generate faces correctly. Deep models allow to catch some more characteristrics of the faces.

I used suggested beta1 0.3 and it generated more types of faces than beta1 0.5 which was suggested by the paper. Adam is the best choice for GAN's as well as other architectures. Number of epochs is a critical component of GAN's. Especially spread betwenn batch size of a generator and a descriminator.

2.9.2 Submitting This Project

When submitting this project, make sure to run all the cells before saving the notebook. Save the notebook file as "dlnd_face_generation.ipynb" and save it as a HTML file under "File" -> "Download as". Include the "problem_unittests.py" files in your submission.