dlnd_face_generation

December 4, 2021

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
```

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

```
In [4]: 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
            transform = transforms.Compose([transforms.Resize(image_size),transforms.ToTensor()]
```

```
    dataset = datasets.ImageFolder(data_dir, transform)

    dataloader = torch.utils.data.DataLoader(dataset=dataset, batch_size=batch_size, shu

    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 [5]:
```
# Define function hyperparameters
batch_size = 32
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)
```

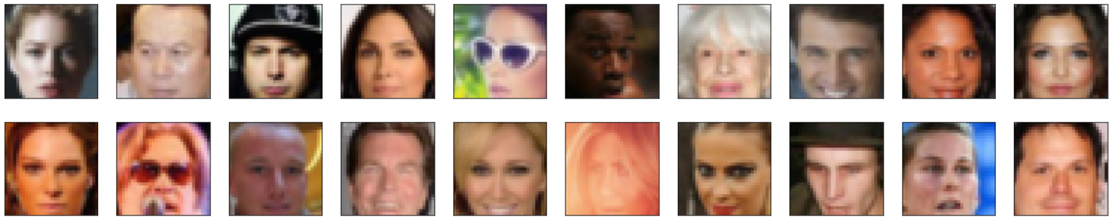
Next, you can view some images! You should seen square images of somewhat-centered faces.

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 [6]:
```
# 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 [7]:
```
# 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

    min , max = feature_range

    x = x * (max-min) + min

    return x
```

In [8]:
```
"""
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(-0.8353)
Max:  tensor(0.9059)
```

---

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 [9]: import torch.nn as nn
        import torch.nn.functional as F
```

2.2 Helper Function

```
In [10]: # helper conv function
         def conv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True):
             """Creates a convolutional layer, with optional batch normalization.
             """
             layers = []
             conv_layer = nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
                                    kernel_size=kernel_size, stride=stride, padding=padding, bia

             layers.append(conv_layer)

             if batch_norm:
                 layers.append(nn.BatchNorm2d(out_channels))
             return nn.Sequential(*layers)


         # helper deconv function
         def deconv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True
             """Creates a transpose convolutional layer, with optional batch normalization.
             """
             layers = []
             # append transpose conv layer
             layers.append(nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride, pa
             # optional batch norm layer
             if batch_norm:
                 layers.append(nn.BatchNorm2d(out_channels))
             return nn.Sequential(*layers)
```

```
In [11]: 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, stride=2, padding=1, bias=False, kernel_siz
        self.batch_norm1 = nn.BatchNorm2d(conv_dim)

        self.conv2 = nn.Conv2d(conv_dim, conv_dim*2, stride=2, padding=1, bias=False, k
        self.batch_norm2 = nn.BatchNorm2d(conv_dim*2)

        self.conv3 = nn.Conv2d(conv_dim*2, conv_dim*4, stride=2, padding=1, bias=False,
        self.batch_norm3 = nn.BatchNorm2d(conv_dim*4)

        self.conv4 = nn.Conv2d(conv_dim*4, conv_dim*8, stride=2, padding=1, bias=False,
        self.batch_norm4 = nn.BatchNorm2d(conv_dim*8)

        self.conv5 = nn.Conv2d(conv_dim*8, conv_dim*16, stride=2, padding=1, bias=False
        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)

        x = x.view(-1, self.conv_dim*4*4)

        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
```

2.3 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 [12]: 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 convolutional
                 """
                 super(Generator, self).__init__()

                 # complete init function

                 self.conv_dim = conv_dim

                 self.t_conv1 = nn.ConvTranspose2d(conv_dim, conv_dim*8, stride=2, padding=1, bi
                 self.batch_norm1 = nn.BatchNorm2d(conv_dim*8)

                 self.t_conv2 = nn.ConvTranspose2d(conv_dim*8, conv_dim*4, stride=2, padding=1,
                 self.batch_norm2 = nn.BatchNorm2d(conv_dim*4)

                 self.t_conv3 = nn.ConvTranspose2d(conv_dim*4, conv_dim*2, stride=2, padding=1,
                 self.batch_norm3 = nn.BatchNorm2d(conv_dim*2)

                 self.t_conv4 = nn.ConvTranspose2d(conv_dim*2, 3, stride=2, padding=1, bias=Fals

                 self.fc = nn.Linear(z_size, conv_dim*4)
```

```
    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)

        # output layer
        x = F.tanh(x)


        return x

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

```
Tests Passed
```

2.4 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 [13]: 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  (classname.find('Conv') != -1 or classname.find('Linear') != -1) and hasattr(m,
                 nn.init.normal_(m.weight.data, 0.0, 0.02)
```

2.5 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 [14]: """
         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 [15]: # 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)
```

```
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=True
  (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=True
  (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=Fa
  (batch_norm1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
  (t_conv2): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=F
  (batch_norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
  (t_conv3): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=Fa
  (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=Fals
  (fc): Linear(in_features=100, out_features=128, bias=True)
)
```

2.5.1 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 [16]:
```
"""
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!
```

2.6 Discriminator and Generator Losses

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

2.6.1 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.

2.6.2 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 [17]: import random
         def real_loss(D_out, smooth=False):

             batch_size = D_out.size(0)

             labels = torch.ones(batch_size)*0.9

             if train_on_gpu:
                 labels = labels.cuda()


             criterion = nn.BCELoss()

             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()

        loss = criterion(D_out.squeeze(), labels)

        return loss
```

2.7 Optimizers

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

```
In [18]: import torch.optim as optim

         d_optimizer = optim.Adam(D.parameters(), lr=0.001, betas=(0.5, 0.999))
         g_optimizer = optim.Adam(G.parameters(), lr=0.001, betas=(0.5, 0.999))
```

2.8 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 [19]: 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)
```

```python
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)

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 [20]: # set number of epochs
         n_epochs = 20


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

```
Epoch [    1/   20] | d_loss: 1.3410 | g_loss: 1.6877
Epoch [    1/   20] | d_loss: 0.8083 | g_loss: 2.5227
Epoch [    1/   20] | d_loss: 0.9431 | g_loss: 2.5018
Epoch [    1/   20] | d_loss: 1.2149 | g_loss: 3.4194
Epoch [    1/   20] | d_loss: 1.7443 | g_loss: 3.0724
Epoch [    1/   20] | d_loss: 1.7218 | g_loss: 0.7178
Epoch [    1/   20] | d_loss: 0.9594 | g_loss: 2.7744
Epoch [    1/   20] | d_loss: 1.3631 | g_loss: 1.2735
Epoch [    1/   20] | d_loss: 1.0233 | g_loss: 1.3976
Epoch [    1/   20] | d_loss: 2.0550 | g_loss: 2.2664
Epoch [    1/   20] | d_loss: 1.1259 | g_loss: 2.2083
Epoch [    1/   20] | d_loss: 0.8518 | g_loss: 2.5673
Epoch [    1/   20] | d_loss: 0.8047 | g_loss: 2.6363
Epoch [    1/   20] | d_loss: 1.0739 | g_loss: 1.6434
Epoch [    1/   20] | d_loss: 0.9607 | g_loss: 2.1572
Epoch [    1/   20] | d_loss: 0.9879 | g_loss: 3.8284
Epoch [    1/   20] | d_loss: 0.9688 | g_loss: 4.0602
Epoch [    1/   20] | d_loss: 0.8698 | g_loss: 2.1034
Epoch [    1/   20] | d_loss: 0.9104 | g_loss: 1.5537
Epoch [    1/   20] | d_loss: 1.3113 | g_loss: 3.6388
Epoch [    1/   20] | d_loss: 1.7428 | g_loss: 1.4453
```

```
Epoch [    1/   20] | d_loss: 1.1188 | g_loss: 2.3361
Epoch [    1/   20] | d_loss: 0.9638 | g_loss: 1.8358
Epoch [    1/   20] | d_loss: 1.2751 | g_loss: 1.7968
Epoch [    1/   20] | d_loss: 0.9835 | g_loss: 1.3598
Epoch [    1/   20] | d_loss: 1.0427 | g_loss: 1.8447
Epoch [    1/   20] | d_loss: 1.0476 | g_loss: 1.8864
Epoch [    1/   20] | d_loss: 0.9795 | g_loss: 1.7668
Epoch [    1/   20] | d_loss: 0.8867 | g_loss: 2.1969
Epoch [    1/   20] | d_loss: 0.8768 | g_loss: 1.6812
Epoch [    1/   20] | d_loss: 0.9383 | g_loss: 2.2353
Epoch [    1/   20] | d_loss: 0.7446 | g_loss: 2.2385
Epoch [    1/   20] | d_loss: 0.7735 | g_loss: 1.6826
Epoch [    1/   20] | d_loss: 1.3513 | g_loss: 3.3663
Epoch [    1/   20] | d_loss: 1.0557 | g_loss: 1.9687
Epoch [    1/   20] | d_loss: 0.8460 | g_loss: 2.1474
Epoch [    1/   20] | d_loss: 1.2995 | g_loss: 3.0650
Epoch [    1/   20] | d_loss: 1.1965 | g_loss: 0.9526
Epoch [    1/   20] | d_loss: 1.1108 | g_loss: 1.3406
Epoch [    1/   20] | d_loss: 0.7843 | g_loss: 2.7248
Epoch [    1/   20] | d_loss: 1.0116 | g_loss: 1.8359
Epoch [    1/   20] | d_loss: 1.1895 | g_loss: 1.8590
Epoch [    1/   20] | d_loss: 0.9884 | g_loss: 2.6506
Epoch [    1/   20] | d_loss: 1.2924 | g_loss: 3.6007
Epoch [    1/   20] | d_loss: 0.7762 | g_loss: 2.1436
Epoch [    1/   20] | d_loss: 0.9187 | g_loss: 1.8038
Epoch [    1/   20] | d_loss: 0.8312 | g_loss: 2.0580
Epoch [    1/   20] | d_loss: 1.0605 | g_loss: 1.8067
Epoch [    1/   20] | d_loss: 1.0432 | g_loss: 2.5434
Epoch [    1/   20] | d_loss: 0.9691 | g_loss: 2.4716
Epoch [    1/   20] | d_loss: 0.9586 | g_loss: 1.9098
Epoch [    1/   20] | d_loss: 0.9474 | g_loss: 1.3652
Epoch [    1/   20] | d_loss: 0.7870 | g_loss: 1.7143
Epoch [    1/   20] | d_loss: 1.1287 | g_loss: 2.3689
Epoch [    1/   20] | d_loss: 1.1794 | g_loss: 1.5965
Epoch [    1/   20] | d_loss: 0.7117 | g_loss: 2.3417
Epoch [    1/   20] | d_loss: 1.2059 | g_loss: 1.4837
Epoch [    2/   20] | d_loss: 0.9094 | g_loss: 1.6220
Epoch [    2/   20] | d_loss: 0.9585 | g_loss: 1.7712
Epoch [    2/   20] | d_loss: 0.8404 | g_loss: 2.2448
Epoch [    2/   20] | d_loss: 1.0190 | g_loss: 1.7595
Epoch [    2/   20] | d_loss: 0.9950 | g_loss: 1.5382
Epoch [    2/   20] | d_loss: 1.0275 | g_loss: 1.6162
Epoch [    2/   20] | d_loss: 0.9559 | g_loss: 2.4432
Epoch [    2/   20] | d_loss: 0.8937 | g_loss: 2.3655
Epoch [    2/   20] | d_loss: 0.9548 | g_loss: 1.4474
Epoch [    2/   20] | d_loss: 1.1028 | g_loss: 1.9325
Epoch [    2/   20] | d_loss: 1.1137 | g_loss: 1.8271
Epoch [    2/   20] | d_loss: 0.8954 | g_loss: 2.2566
```

```
Epoch [    2/   20] | d_loss: 1.0349 | g_loss: 1.4165
Epoch [    2/   20] | d_loss: 0.9451 | g_loss: 1.4964
Epoch [    2/   20] | d_loss: 0.9680 | g_loss: 1.9507
Epoch [    2/   20] | d_loss: 0.9172 | g_loss: 2.5064
Epoch [    2/   20] | d_loss: 0.8907 | g_loss: 1.6503
Epoch [    2/   20] | d_loss: 1.1239 | g_loss: 2.3504
Epoch [    2/   20] | d_loss: 1.1258 | g_loss: 1.0853
Epoch [    2/   20] | d_loss: 0.8395 | g_loss: 2.0416
Epoch [    2/   20] | d_loss: 1.0543 | g_loss: 2.0583
Epoch [    2/   20] | d_loss: 1.4673 | g_loss: 1.8123
Epoch [    2/   20] | d_loss: 1.4406 | g_loss: 1.3683
Epoch [    2/   20] | d_loss: 1.1991 | g_loss: 1.5417
Epoch [    2/   20] | d_loss: 1.0116 | g_loss: 1.5197
Epoch [    2/   20] | d_loss: 1.2217 | g_loss: 1.8361
Epoch [    2/   20] | d_loss: 1.3713 | g_loss: 2.1051
Epoch [    2/   20] | d_loss: 1.2523 | g_loss: 2.1729
Epoch [    2/   20] | d_loss: 1.1620 | g_loss: 0.9532
Epoch [    2/   20] | d_loss: 1.3034 | g_loss: 1.3336
Epoch [    2/   20] | d_loss: 1.0637 | g_loss: 1.0934
Epoch [    2/   20] | d_loss: 1.1131 | g_loss: 0.6317
Epoch [    2/   20] | d_loss: 1.1693 | g_loss: 2.2442
Epoch [    2/   20] | d_loss: 0.9729 | g_loss: 2.0116
Epoch [    2/   20] | d_loss: 0.8379 | g_loss: 1.7079
Epoch [    2/   20] | d_loss: 1.0573 | g_loss: 1.7950
Epoch [    2/   20] | d_loss: 1.0828 | g_loss: 1.6619
Epoch [    2/   20] | d_loss: 0.9721 | g_loss: 1.4295
Epoch [    2/   20] | d_loss: 0.9534 | g_loss: 2.2124
Epoch [    2/   20] | d_loss: 1.1340 | g_loss: 1.7049
Epoch [    2/   20] | d_loss: 1.1888 | g_loss: 2.0104
Epoch [    2/   20] | d_loss: 0.9348 | g_loss: 1.8717
Epoch [    2/   20] | d_loss: 1.0892 | g_loss: 1.2627
Epoch [    2/   20] | d_loss: 1.2123 | g_loss: 2.3151
Epoch [    2/   20] | d_loss: 1.0010 | g_loss: 1.3907
Epoch [    2/   20] | d_loss: 1.0612 | g_loss: 2.5985
Epoch [    2/   20] | d_loss: 1.1025 | g_loss: 1.4520
Epoch [    2/   20] | d_loss: 0.8838 | g_loss: 1.9847
Epoch [    2/   20] | d_loss: 0.9856 | g_loss: 1.2092
Epoch [    2/   20] | d_loss: 0.8597 | g_loss: 2.1875
Epoch [    2/   20] | d_loss: 1.1347 | g_loss: 1.8924
Epoch [    2/   20] | d_loss: 1.5894 | g_loss: 0.7019
Epoch [    2/   20] | d_loss: 0.9852 | g_loss: 1.0994
Epoch [    2/   20] | d_loss: 1.1059 | g_loss: 1.7722
Epoch [    2/   20] | d_loss: 0.9116 | g_loss: 1.5727
Epoch [    2/   20] | d_loss: 1.3994 | g_loss: 1.3781
Epoch [    2/   20] | d_loss: 1.1416 | g_loss: 1.3420
Epoch [    3/   20] | d_loss: 1.3941 | g_loss: 2.6468
Epoch [    3/   20] | d_loss: 1.2275 | g_loss: 1.0323
Epoch [    3/   20] | d_loss: 0.9345 | g_loss: 2.0826
```

```
Epoch [    3/   20] | d_loss: 0.9573 | g_loss: 1.9348
Epoch [    3/   20] | d_loss: 1.0714 | g_loss: 1.5194
Epoch [    3/   20] | d_loss: 1.3877 | g_loss: 2.2767
Epoch [    3/   20] | d_loss: 1.1236 | g_loss: 1.8957
Epoch [    3/   20] | d_loss: 1.0790 | g_loss: 0.8222
Epoch [    3/   20] | d_loss: 0.9682 | g_loss: 1.4873
Epoch [    3/   20] | d_loss: 1.0002 | g_loss: 1.5789
Epoch [    3/   20] | d_loss: 0.9438 | g_loss: 1.4569
Epoch [    3/   20] | d_loss: 1.6300 | g_loss: 1.2499
Epoch [    3/   20] | d_loss: 0.9431 | g_loss: 0.9553
Epoch [    3/   20] | d_loss: 1.0703 | g_loss: 2.4924
Epoch [    3/   20] | d_loss: 0.9501 | g_loss: 1.8735
Epoch [    3/   20] | d_loss: 1.1096 | g_loss: 1.5766
Epoch [    3/   20] | d_loss: 1.3704 | g_loss: 1.9476
Epoch [    3/   20] | d_loss: 1.1882 | g_loss: 2.4412
Epoch [    3/   20] | d_loss: 1.0843 | g_loss: 1.1336
Epoch [    3/   20] | d_loss: 0.9643 | g_loss: 2.2593
Epoch [    3/   20] | d_loss: 0.8856 | g_loss: 1.8746
Epoch [    3/   20] | d_loss: 1.1749 | g_loss: 1.4251
Epoch [    3/   20] | d_loss: 1.0128 | g_loss: 2.2875
Epoch [    3/   20] | d_loss: 1.4404 | g_loss: 1.2730
Epoch [    3/   20] | d_loss: 0.9387 | g_loss: 1.3545
Epoch [    3/   20] | d_loss: 1.1194 | g_loss: 1.6593
Epoch [    3/   20] | d_loss: 1.2043 | g_loss: 1.2151
Epoch [    3/   20] | d_loss: 1.1007 | g_loss: 1.7584
Epoch [    3/   20] | d_loss: 1.1406 | g_loss: 1.4594
Epoch [    3/   20] | d_loss: 0.8827 | g_loss: 1.1260
Epoch [    3/   20] | d_loss: 0.8915 | g_loss: 1.5009
Epoch [    3/   20] | d_loss: 1.2010 | g_loss: 1.1046
Epoch [    3/   20] | d_loss: 1.0124 | g_loss: 1.8946
Epoch [    3/   20] | d_loss: 0.8624 | g_loss: 1.5483
Epoch [    3/   20] | d_loss: 1.1528 | g_loss: 2.2400
Epoch [    3/   20] | d_loss: 1.0025 | g_loss: 1.3095
Epoch [    3/   20] | d_loss: 1.0074 | g_loss: 1.8313
Epoch [    3/   20] | d_loss: 1.0708 | g_loss: 2.3802
Epoch [    3/   20] | d_loss: 1.1420 | g_loss: 1.3077
Epoch [    3/   20] | d_loss: 1.1235 | g_loss: 1.0105
Epoch [    3/   20] | d_loss: 1.1003 | g_loss: 1.8940
Epoch [    3/   20] | d_loss: 1.2442 | g_loss: 1.7590
Epoch [    3/   20] | d_loss: 1.0573 | g_loss: 1.5636
Epoch [    3/   20] | d_loss: 1.4217 | g_loss: 1.3573
Epoch [    3/   20] | d_loss: 0.9569 | g_loss: 1.9019
Epoch [    3/   20] | d_loss: 1.0048 | g_loss: 1.9369
Epoch [    3/   20] | d_loss: 1.3304 | g_loss: 1.5377
Epoch [    3/   20] | d_loss: 1.3715 | g_loss: 1.4302
Epoch [    3/   20] | d_loss: 1.0975 | g_loss: 1.6516
Epoch [    3/   20] | d_loss: 1.2265 | g_loss: 1.2618
Epoch [    3/   20] | d_loss: 1.0727 | g_loss: 1.6169
```

```
Epoch [    3/   20] | d_loss: 1.0492 | g_loss: 1.6460
Epoch [    3/   20] | d_loss: 1.0186 | g_loss: 1.3522
Epoch [    3/   20] | d_loss: 1.3181 | g_loss: 1.5211
Epoch [    3/   20] | d_loss: 0.9494 | g_loss: 1.6803
Epoch [    3/   20] | d_loss: 0.9418 | g_loss: 2.0692
Epoch [    3/   20] | d_loss: 0.8340 | g_loss: 1.5124
Epoch [    4/   20] | d_loss: 1.5142 | g_loss: 0.8316
Epoch [    4/   20] | d_loss: 1.1340 | g_loss: 1.8879
Epoch [    4/   20] | d_loss: 0.9550 | g_loss: 1.1428
Epoch [    4/   20] | d_loss: 0.8376 | g_loss: 1.9751
Epoch [    4/   20] | d_loss: 0.8636 | g_loss: 2.2758
Epoch [    4/   20] | d_loss: 1.0789 | g_loss: 2.1645
Epoch [    4/   20] | d_loss: 0.9338 | g_loss: 1.3286
Epoch [    4/   20] | d_loss: 1.0616 | g_loss: 0.9878
Epoch [    4/   20] | d_loss: 1.0665 | g_loss: 0.9668
Epoch [    4/   20] | d_loss: 0.9935 | g_loss: 1.9654
Epoch [    4/   20] | d_loss: 0.9366 | g_loss: 1.1873
Epoch [    4/   20] | d_loss: 0.8429 | g_loss: 2.1554
Epoch [    4/   20] | d_loss: 0.8265 | g_loss: 1.2704
Epoch [    4/   20] | d_loss: 1.1360 | g_loss: 2.0538
Epoch [    4/   20] | d_loss: 1.0645 | g_loss: 1.6111
Epoch [    4/   20] | d_loss: 1.0714 | g_loss: 1.7898
Epoch [    4/   20] | d_loss: 1.1571 | g_loss: 1.6686
Epoch [    4/   20] | d_loss: 1.3667 | g_loss: 1.1974
Epoch [    4/   20] | d_loss: 1.2227 | g_loss: 1.9541
Epoch [    4/   20] | d_loss: 1.4113 | g_loss: 1.3141
Epoch [    4/   20] | d_loss: 0.8956 | g_loss: 1.6508
Epoch [    4/   20] | d_loss: 1.0139 | g_loss: 2.0187
Epoch [    4/   20] | d_loss: 1.1382 | g_loss: 1.3817
Epoch [    4/   20] | d_loss: 1.0314 | g_loss: 1.1840
Epoch [    4/   20] | d_loss: 0.8524 | g_loss: 2.0104
Epoch [    4/   20] | d_loss: 1.2173 | g_loss: 1.1716
Epoch [    4/   20] | d_loss: 1.1469 | g_loss: 1.4327
Epoch [    4/   20] | d_loss: 1.1670 | g_loss: 1.5926
Epoch [    4/   20] | d_loss: 1.0451 | g_loss: 2.2475
Epoch [    4/   20] | d_loss: 0.9615 | g_loss: 1.6077
Epoch [    4/   20] | d_loss: 0.9233 | g_loss: 1.5778
Epoch [    4/   20] | d_loss: 0.6748 | g_loss: 1.9946
Epoch [    4/   20] | d_loss: 1.1497 | g_loss: 2.0976
Epoch [    4/   20] | d_loss: 1.1940 | g_loss: 2.2202
Epoch [    4/   20] | d_loss: 1.1119 | g_loss: 1.7764
Epoch [    4/   20] | d_loss: 0.9704 | g_loss: 1.9333
Epoch [    4/   20] | d_loss: 1.1646 | g_loss: 2.1435
Epoch [    4/   20] | d_loss: 1.0567 | g_loss: 1.1359
Epoch [    4/   20] | d_loss: 0.9990 | g_loss: 1.1512
Epoch [    4/   20] | d_loss: 1.0960 | g_loss: 1.8922
Epoch [    4/   20] | d_loss: 0.6894 | g_loss: 2.2372
Epoch [    4/   20] | d_loss: 1.0886 | g_loss: 1.1624
```

```
Epoch [    4/   20] | d_loss: 1.0955 | g_loss: 1.5626
Epoch [    4/   20] | d_loss: 1.0434 | g_loss: 1.0882
Epoch [    4/   20] | d_loss: 0.8885 | g_loss: 1.6196
Epoch [    4/   20] | d_loss: 1.1111 | g_loss: 0.9941
Epoch [    4/   20] | d_loss: 1.3397 | g_loss: 1.1548
Epoch [    4/   20] | d_loss: 0.8940 | g_loss: 1.4756
Epoch [    4/   20] | d_loss: 1.1399 | g_loss: 1.3099
Epoch [    4/   20] | d_loss: 1.0765 | g_loss: 1.7075
Epoch [    4/   20] | d_loss: 1.0793 | g_loss: 1.7735
Epoch [    4/   20] | d_loss: 1.0666 | g_loss: 2.1943
Epoch [    4/   20] | d_loss: 0.6747 | g_loss: 2.4924
Epoch [    4/   20] | d_loss: 1.1243 | g_loss: 1.1790
Epoch [    4/   20] | d_loss: 1.0284 | g_loss: 2.1970
Epoch [    4/   20] | d_loss: 1.0078 | g_loss: 1.3991
Epoch [    4/   20] | d_loss: 1.7040 | g_loss: 1.9945
Epoch [    5/   20] | d_loss: 0.8901 | g_loss: 2.4324
Epoch [    5/   20] | d_loss: 0.8928 | g_loss: 1.3646
Epoch [    5/   20] | d_loss: 0.9072 | g_loss: 2.3640
Epoch [    5/   20] | d_loss: 1.0055 | g_loss: 2.4067
Epoch [    5/   20] | d_loss: 0.8902 | g_loss: 1.3025
Epoch [    5/   20] | d_loss: 1.0305 | g_loss: 1.2618
Epoch [    5/   20] | d_loss: 1.0351 | g_loss: 2.2770
Epoch [    5/   20] | d_loss: 0.6919 | g_loss: 2.2477
Epoch [    5/   20] | d_loss: 0.9383 | g_loss: 2.7969
Epoch [    5/   20] | d_loss: 1.0097 | g_loss: 0.9729
Epoch [    5/   20] | d_loss: 0.9176 | g_loss: 1.8666
Epoch [    5/   20] | d_loss: 0.8321 | g_loss: 1.5919
Epoch [    5/   20] | d_loss: 1.1407 | g_loss: 0.8511
Epoch [    5/   20] | d_loss: 0.9464 | g_loss: 1.9345
Epoch [    5/   20] | d_loss: 0.8874 | g_loss: 1.7828
Epoch [    5/   20] | d_loss: 1.0889 | g_loss: 1.5045
Epoch [    5/   20] | d_loss: 1.1125 | g_loss: 1.7017
Epoch [    5/   20] | d_loss: 0.8589 | g_loss: 1.9159
Epoch [    5/   20] | d_loss: 1.1825 | g_loss: 1.8010
Epoch [    5/   20] | d_loss: 0.9907 | g_loss: 2.0119
Epoch [    5/   20] | d_loss: 1.0713 | g_loss: 1.1380
Epoch [    5/   20] | d_loss: 0.9846 | g_loss: 1.4710
Epoch [    5/   20] | d_loss: 0.9546 | g_loss: 2.1470
Epoch [    5/   20] | d_loss: 0.9612 | g_loss: 1.6308
Epoch [    5/   20] | d_loss: 1.2216 | g_loss: 1.3448
Epoch [    5/   20] | d_loss: 0.8024 | g_loss: 1.7310
Epoch [    5/   20] | d_loss: 0.9915 | g_loss: 1.7016
Epoch [    5/   20] | d_loss: 1.1087 | g_loss: 0.8234
Epoch [    5/   20] | d_loss: 1.0075 | g_loss: 2.0422
Epoch [    5/   20] | d_loss: 1.0165 | g_loss: 1.0873
Epoch [    5/   20] | d_loss: 1.0095 | g_loss: 1.4464
Epoch [    5/   20] | d_loss: 1.2327 | g_loss: 1.0699
Epoch [    5/   20] | d_loss: 0.9813 | g_loss: 1.5446
```

```
Epoch [    5/   20] | d_loss: 1.0009 | g_loss: 1.2067
Epoch [    5/   20] | d_loss: 1.0737 | g_loss: 1.9417
Epoch [    5/   20] | d_loss: 1.3185 | g_loss: 2.1866
Epoch [    5/   20] | d_loss: 0.9853 | g_loss: 1.0789
Epoch [    5/   20] | d_loss: 0.9530 | g_loss: 1.7885
Epoch [    5/   20] | d_loss: 0.9326 | g_loss: 2.0430
Epoch [    5/   20] | d_loss: 0.8851 | g_loss: 1.2641
Epoch [    5/   20] | d_loss: 0.9381 | g_loss: 0.8336
Epoch [    5/   20] | d_loss: 0.6952 | g_loss: 1.5381
Epoch [    5/   20] | d_loss: 1.3295 | g_loss: 2.0059
Epoch [    5/   20] | d_loss: 0.6894 | g_loss: 2.4495
Epoch [    5/   20] | d_loss: 0.9533 | g_loss: 1.6122
Epoch [    5/   20] | d_loss: 0.8554 | g_loss: 1.8900
Epoch [    5/   20] | d_loss: 1.1598 | g_loss: 1.0663
Epoch [    5/   20] | d_loss: 0.8030 | g_loss: 1.9633
Epoch [    5/   20] | d_loss: 1.0294 | g_loss: 0.9226
Epoch [    5/   20] | d_loss: 0.7200 | g_loss: 1.0300
Epoch [    5/   20] | d_loss: 1.1315 | g_loss: 1.7883
Epoch [    5/   20] | d_loss: 1.0711 | g_loss: 1.5376
Epoch [    5/   20] | d_loss: 0.9525 | g_loss: 1.3630
Epoch [    5/   20] | d_loss: 1.1085 | g_loss: 1.7579
Epoch [    5/   20] | d_loss: 0.9633 | g_loss: 1.7216
Epoch [    5/   20] | d_loss: 1.1091 | g_loss: 1.7189
Epoch [    5/   20] | d_loss: 0.9220 | g_loss: 1.5813
Epoch [    6/   20] | d_loss: 1.0488 | g_loss: 1.6270
Epoch [    6/   20] | d_loss: 0.9536 | g_loss: 1.6653
Epoch [    6/   20] | d_loss: 1.0810 | g_loss: 1.1424
Epoch [    6/   20] | d_loss: 1.2529 | g_loss: 1.1532
Epoch [    6/   20] | d_loss: 0.9694 | g_loss: 1.7206
Epoch [    6/   20] | d_loss: 0.9474 | g_loss: 1.3738
Epoch [    6/   20] | d_loss: 1.2262 | g_loss: 1.9109
Epoch [    6/   20] | d_loss: 0.7310 | g_loss: 2.4873
Epoch [    6/   20] | d_loss: 0.9578 | g_loss: 1.4205
Epoch [    6/   20] | d_loss: 0.8391 | g_loss: 2.1003
Epoch [    6/   20] | d_loss: 1.5452 | g_loss: 1.5038
Epoch [    6/   20] | d_loss: 2.3877 | g_loss: 2.6137
Epoch [    6/   20] | d_loss: 1.4421 | g_loss: 2.7658
Epoch [    6/   20] | d_loss: 0.7360 | g_loss: 1.6568
Epoch [    6/   20] | d_loss: 1.3938 | g_loss: 1.3446
Epoch [    6/   20] | d_loss: 2.1651 | g_loss: 1.6014
Epoch [    6/   20] | d_loss: 0.8241 | g_loss: 1.6208
Epoch [    6/   20] | d_loss: 0.6933 | g_loss: 2.1155
Epoch [    6/   20] | d_loss: 0.9140 | g_loss: 1.8002
Epoch [    6/   20] | d_loss: 1.1477 | g_loss: 1.6900
Epoch [    6/   20] | d_loss: 0.6583 | g_loss: 1.5776
Epoch [    6/   20] | d_loss: 0.8411 | g_loss: 1.0656
Epoch [    6/   20] | d_loss: 0.9438 | g_loss: 2.5059
Epoch [    6/   20] | d_loss: 1.2612 | g_loss: 1.1200
```

```
Epoch [    6/   20] | d_loss: 0.8428 | g_loss: 2.2863
Epoch [    6/   20] | d_loss: 0.7016 | g_loss: 2.8809
Epoch [    6/   20] | d_loss: 1.1341 | g_loss: 1.7779
Epoch [    6/   20] | d_loss: 1.0307 | g_loss: 1.5960
Epoch [    6/   20] | d_loss: 1.1288 | g_loss: 2.4794
Epoch [    6/   20] | d_loss: 1.2157 | g_loss: 2.9539
Epoch [    6/   20] | d_loss: 0.7961 | g_loss: 1.5473
Epoch [    6/   20] | d_loss: 0.7690 | g_loss: 1.9322
Epoch [    6/   20] | d_loss: 0.6972 | g_loss: 1.0251
Epoch [    6/   20] | d_loss: 1.3472 | g_loss: 1.3406
Epoch [    6/   20] | d_loss: 0.8131 | g_loss: 1.7482
Epoch [    6/   20] | d_loss: 1.0422 | g_loss: 1.4427
Epoch [    6/   20] | d_loss: 0.8503 | g_loss: 2.6571
Epoch [    6/   20] | d_loss: 1.0755 | g_loss: 2.7085
Epoch [    6/   20] | d_loss: 0.5543 | g_loss: 1.1587
Epoch [    6/   20] | d_loss: 1.3317 | g_loss: 1.4151
Epoch [    6/   20] | d_loss: 0.7668 | g_loss: 1.1039
Epoch [    6/   20] | d_loss: 1.0458 | g_loss: 1.5796
Epoch [    6/   20] | d_loss: 1.1503 | g_loss: 1.6051
Epoch [    6/   20] | d_loss: 0.9501 | g_loss: 1.3939
Epoch [    6/   20] | d_loss: 0.7846 | g_loss: 1.3245
Epoch [    6/   20] | d_loss: 0.8688 | g_loss: 2.2101
Epoch [    6/   20] | d_loss: 1.0552 | g_loss: 1.3938
Epoch [    6/   20] | d_loss: 1.1809 | g_loss: 1.1041
Epoch [    6/   20] | d_loss: 1.0089 | g_loss: 1.2802
Epoch [    6/   20] | d_loss: 1.0149 | g_loss: 1.7091
Epoch [    6/   20] | d_loss: 1.1441 | g_loss: 2.2536
Epoch [    6/   20] | d_loss: 0.6962 | g_loss: 1.9467
Epoch [    6/   20] | d_loss: 1.0782 | g_loss: 1.3300
Epoch [    6/   20] | d_loss: 0.8918 | g_loss: 1.4299
Epoch [    6/   20] | d_loss: 0.8211 | g_loss: 2.1342
Epoch [    6/   20] | d_loss: 1.2838 | g_loss: 1.9691
Epoch [    6/   20] | d_loss: 1.1194 | g_loss: 1.8925
Epoch [    7/   20] | d_loss: 0.8074 | g_loss: 1.2777
Epoch [    7/   20] | d_loss: 1.0993 | g_loss: 1.5489
Epoch [    7/   20] | d_loss: 0.8602 | g_loss: 2.4790
Epoch [    7/   20] | d_loss: 0.9960 | g_loss: 2.0519
Epoch [    7/   20] | d_loss: 1.4975 | g_loss: 1.8272
Epoch [    7/   20] | d_loss: 0.7493 | g_loss: 2.1340
Epoch [    7/   20] | d_loss: 0.8407 | g_loss: 2.2850
Epoch [    7/   20] | d_loss: 1.0513 | g_loss: 1.6050
Epoch [    7/   20] | d_loss: 0.7857 | g_loss: 2.0806
Epoch [    7/   20] | d_loss: 0.9778 | g_loss: 2.3099
Epoch [    7/   20] | d_loss: 0.9993 | g_loss: 2.7715
Epoch [    7/   20] | d_loss: 1.0938 | g_loss: 1.4697
Epoch [    7/   20] | d_loss: 0.8482 | g_loss: 2.3677
Epoch [    7/   20] | d_loss: 1.2949 | g_loss: 1.4811
Epoch [    7/   20] | d_loss: 1.1720 | g_loss: 1.7462
```

```
Epoch [    7/   20] | d_loss: 0.9126 | g_loss: 2.0487
Epoch [    7/   20] | d_loss: 1.0603 | g_loss: 1.6304
Epoch [    7/   20] | d_loss: 0.7708 | g_loss: 1.7864
Epoch [    7/   20] | d_loss: 1.0079 | g_loss: 1.8211
Epoch [    7/   20] | d_loss: 1.6345 | g_loss: 1.7260
Epoch [    7/   20] | d_loss: 0.8616 | g_loss: 2.8633
Epoch [    7/   20] | d_loss: 0.7943 | g_loss: 1.8359
Epoch [    7/   20] | d_loss: 0.9330 | g_loss: 2.0738
Epoch [    7/   20] | d_loss: 1.0051 | g_loss: 2.7588
Epoch [    7/   20] | d_loss: 0.8659 | g_loss: 1.8570
Epoch [    7/   20] | d_loss: 0.7483 | g_loss: 1.1572
Epoch [    7/   20] | d_loss: 0.8856 | g_loss: 1.8228
Epoch [    7/   20] | d_loss: 1.4415 | g_loss: 2.2207
Epoch [    7/   20] | d_loss: 0.7432 | g_loss: 2.5737
Epoch [    7/   20] | d_loss: 0.8467 | g_loss: 2.0599
Epoch [    7/   20] | d_loss: 1.1409 | g_loss: 0.7903
Epoch [    7/   20] | d_loss: 0.6581 | g_loss: 2.7223
Epoch [    7/   20] | d_loss: 0.6663 | g_loss: 2.4738
Epoch [    7/   20] | d_loss: 1.2880 | g_loss: 1.2204
Epoch [    7/   20] | d_loss: 1.1043 | g_loss: 1.5499
Epoch [    7/   20] | d_loss: 0.7580 | g_loss: 1.6071
Epoch [    7/   20] | d_loss: 0.7739 | g_loss: 1.0236
Epoch [    7/   20] | d_loss: 0.9728 | g_loss: 3.0191
Epoch [    7/   20] | d_loss: 0.7476 | g_loss: 2.4810
Epoch [    7/   20] | d_loss: 0.8540 | g_loss: 1.1241
Epoch [    7/   20] | d_loss: 0.8661 | g_loss: 2.0064
Epoch [    7/   20] | d_loss: 0.7292 | g_loss: 1.3680
Epoch [    7/   20] | d_loss: 0.7976 | g_loss: 1.5648
Epoch [    7/   20] | d_loss: 1.0501 | g_loss: 1.1225
Epoch [    7/   20] | d_loss: 1.0486 | g_loss: 1.9106
Epoch [    7/   20] | d_loss: 0.9805 | g_loss: 1.7900
Epoch [    7/   20] | d_loss: 0.8760 | g_loss: 2.0870
Epoch [    7/   20] | d_loss: 0.7017 | g_loss: 1.5833
Epoch [    7/   20] | d_loss: 0.6658 | g_loss: 2.2209
Epoch [    7/   20] | d_loss: 0.9134 | g_loss: 2.1912
Epoch [    7/   20] | d_loss: 1.4476 | g_loss: 2.7825
Epoch [    7/   20] | d_loss: 0.8492 | g_loss: 1.7715
Epoch [    7/   20] | d_loss: 0.8927 | g_loss: 3.1184
Epoch [    7/   20] | d_loss: 0.5832 | g_loss: 1.9596
Epoch [    7/   20] | d_loss: 0.9458 | g_loss: 1.6788
Epoch [    7/   20] | d_loss: 0.8262 | g_loss: 3.6936
Epoch [    7/   20] | d_loss: 0.9384 | g_loss: 1.5279
Epoch [    8/   20] | d_loss: 1.2418 | g_loss: 1.1173
Epoch [    8/   20] | d_loss: 0.7253 | g_loss: 2.0997
Epoch [    8/   20] | d_loss: 1.1337 | g_loss: 1.5788
Epoch [    8/   20] | d_loss: 0.9958 | g_loss: 1.5526
Epoch [    8/   20] | d_loss: 1.0806 | g_loss: 2.5134
Epoch [    8/   20] | d_loss: 1.0343 | g_loss: 1.9920
```

```
Epoch [    8/   20] | d_loss: 0.7750 | g_loss: 1.1636
Epoch [    8/   20] | d_loss: 1.0802 | g_loss: 2.3683
Epoch [    8/   20] | d_loss: 1.0549 | g_loss: 2.2435
Epoch [    8/   20] | d_loss: 1.6158 | g_loss: 1.6031
Epoch [    8/   20] | d_loss: 0.7114 | g_loss: 2.2871
Epoch [    8/   20] | d_loss: 0.9751 | g_loss: 1.9100
Epoch [    8/   20] | d_loss: 0.6140 | g_loss: 2.1776
Epoch [    8/   20] | d_loss: 0.9664 | g_loss: 1.5029
Epoch [    8/   20] | d_loss: 0.6933 | g_loss: 3.1789
Epoch [    8/   20] | d_loss: 1.1110 | g_loss: 1.4896
Epoch [    8/   20] | d_loss: 0.9496 | g_loss: 1.5429
Epoch [    8/   20] | d_loss: 0.7460 | g_loss: 1.4061
Epoch [    8/   20] | d_loss: 1.0271 | g_loss: 1.7233
Epoch [    8/   20] | d_loss: 1.1428 | g_loss: 1.1510
Epoch [    8/   20] | d_loss: 1.4805 | g_loss: 1.9683
Epoch [    8/   20] | d_loss: 2.0902 | g_loss: 1.0955
Epoch [    8/   20] | d_loss: 0.8361 | g_loss: 1.1694
Epoch [    8/   20] | d_loss: 1.5236 | g_loss: 1.5805
Epoch [    8/   20] | d_loss: 0.7032 | g_loss: 1.2869
Epoch [    8/   20] | d_loss: 0.8642 | g_loss: 1.8553
Epoch [    8/   20] | d_loss: 0.8421 | g_loss: 1.2544
Epoch [    8/   20] | d_loss: 0.6808 | g_loss: 1.9333
Epoch [    8/   20] | d_loss: 0.9138 | g_loss: 0.7444
Epoch [    8/   20] | d_loss: 0.8290 | g_loss: 3.2469
Epoch [    8/   20] | d_loss: 1.1009 | g_loss: 1.3451
Epoch [    8/   20] | d_loss: 0.5889 | g_loss: 1.7209
Epoch [    8/   20] | d_loss: 0.9868 | g_loss: 2.7003
Epoch [    8/   20] | d_loss: 0.6479 | g_loss: 1.2738
Epoch [    8/   20] | d_loss: 0.7370 | g_loss: 2.4645
Epoch [    8/   20] | d_loss: 1.5040 | g_loss: 2.0487
Epoch [    8/   20] | d_loss: 1.0662 | g_loss: 1.2676
Epoch [    8/   20] | d_loss: 0.9688 | g_loss: 1.9704
Epoch [    8/   20] | d_loss: 1.3498 | g_loss: 0.9855
Epoch [    8/   20] | d_loss: 1.5557 | g_loss: 0.7540
Epoch [    8/   20] | d_loss: 1.1011 | g_loss: 1.2944
Epoch [    8/   20] | d_loss: 0.7999 | g_loss: 1.4615
Epoch [    8/   20] | d_loss: 0.8555 | g_loss: 1.2924
Epoch [    8/   20] | d_loss: 1.0586 | g_loss: 1.6241
Epoch [    8/   20] | d_loss: 1.4615 | g_loss: 1.9980
Epoch [    8/   20] | d_loss: 1.2141 | g_loss: 1.4996
Epoch [    8/   20] | d_loss: 0.6511 | g_loss: 1.9302
Epoch [    8/   20] | d_loss: 0.6358 | g_loss: 2.7789
Epoch [    8/   20] | d_loss: 0.9147 | g_loss: 2.1776
Epoch [    8/   20] | d_loss: 0.9098 | g_loss: 0.5851
Epoch [    8/   20] | d_loss: 1.2212 | g_loss: 2.6697
Epoch [    8/   20] | d_loss: 1.0504 | g_loss: 2.2149
Epoch [    8/   20] | d_loss: 0.7319 | g_loss: 1.5682
Epoch [    8/   20] | d_loss: 1.0562 | g_loss: 0.7793
```

```
Epoch [    8/   20] | d_loss: 1.1532 | g_loss: 2.3951
Epoch [    8/   20] | d_loss: 0.8775 | g_loss: 1.8595
Epoch [    8/   20] | d_loss: 1.2122 | g_loss: 1.5395
Epoch [    9/   20] | d_loss: 1.3657 | g_loss: 1.3099
Epoch [    9/   20] | d_loss: 0.5584 | g_loss: 3.1783
Epoch [    9/   20] | d_loss: 0.6522 | g_loss: 2.5068
Epoch [    9/   20] | d_loss: 1.0440 | g_loss: 1.9683
Epoch [    9/   20] | d_loss: 0.8096 | g_loss: 1.8878
Epoch [    9/   20] | d_loss: 0.8019 | g_loss: 2.2206
Epoch [    9/   20] | d_loss: 0.7028 | g_loss: 2.2059
Epoch [    9/   20] | d_loss: 1.0821 | g_loss: 2.4425
Epoch [    9/   20] | d_loss: 0.5671 | g_loss: 2.4308
Epoch [    9/   20] | d_loss: 0.5541 | g_loss: 2.9056
Epoch [    9/   20] | d_loss: 0.7006 | g_loss: 1.9342
Epoch [    9/   20] | d_loss: 0.6915 | g_loss: 2.1988
Epoch [    9/   20] | d_loss: 0.6361 | g_loss: 2.0149
Epoch [    9/   20] | d_loss: 1.1282 | g_loss: 1.3757
Epoch [    9/   20] | d_loss: 0.6713 | g_loss: 2.4115
Epoch [    9/   20] | d_loss: 0.9905 | g_loss: 1.6673
Epoch [    9/   20] | d_loss: 0.7543 | g_loss: 2.6038
Epoch [    9/   20] | d_loss: 0.8552 | g_loss: 2.0164
Epoch [    9/   20] | d_loss: 0.7924 | g_loss: 2.8267
Epoch [    9/   20] | d_loss: 0.8261 | g_loss: 2.2491
Epoch [    9/   20] | d_loss: 0.8304 | g_loss: 1.5020
Epoch [    9/   20] | d_loss: 0.7095 | g_loss: 1.9923
Epoch [    9/   20] | d_loss: 1.4415 | g_loss: 3.3428
Epoch [    9/   20] | d_loss: 1.2161 | g_loss: 0.9055
Epoch [    9/   20] | d_loss: 1.1698 | g_loss: 1.2545
Epoch [    9/   20] | d_loss: 0.7771 | g_loss: 2.0362
Epoch [    9/   20] | d_loss: 1.0175 | g_loss: 1.6533
Epoch [    9/   20] | d_loss: 1.0763 | g_loss: 1.2488
Epoch [    9/   20] | d_loss: 1.2067 | g_loss: 4.0500
Epoch [    9/   20] | d_loss: 1.4457 | g_loss: 1.0013
Epoch [    9/   20] | d_loss: 0.6138 | g_loss: 1.5349
Epoch [    9/   20] | d_loss: 0.6903 | g_loss: 2.1889
Epoch [    9/   20] | d_loss: 0.8987 | g_loss: 1.7895
Epoch [    9/   20] | d_loss: 1.1958 | g_loss: 0.6613
Epoch [    9/   20] | d_loss: 0.9407 | g_loss: 1.8264
Epoch [    9/   20] | d_loss: 0.9652 | g_loss: 1.6092
Epoch [    9/   20] | d_loss: 0.7446 | g_loss: 1.3745
Epoch [    9/   20] | d_loss: 1.0466 | g_loss: 3.2333
Epoch [    9/   20] | d_loss: 1.4456 | g_loss: 2.2510
Epoch [    9/   20] | d_loss: 0.6746 | g_loss: 1.9958
Epoch [    9/   20] | d_loss: 1.1321 | g_loss: 2.3777
Epoch [    9/   20] | d_loss: 0.8301 | g_loss: 2.3469
Epoch [    9/   20] | d_loss: 0.7412 | g_loss: 1.7729
Epoch [    9/   20] | d_loss: 1.2512 | g_loss: 1.5619
Epoch [    9/   20] | d_loss: 0.9024 | g_loss: 2.0391
```

```
Epoch [    9/   20] | d_loss: 0.6175 | g_loss: 3.1353
Epoch [    9/   20] | d_loss: 0.9978 | g_loss: 2.4990
Epoch [    9/   20] | d_loss: 0.6818 | g_loss: 2.1180
Epoch [    9/   20] | d_loss: 0.5625 | g_loss: 1.3802
Epoch [    9/   20] | d_loss: 1.0244 | g_loss: 1.7958
Epoch [    9/   20] | d_loss: 0.9174 | g_loss: 1.2820
Epoch [    9/   20] | d_loss: 0.7433 | g_loss: 1.4434
Epoch [    9/   20] | d_loss: 1.3821 | g_loss: 2.9227
Epoch [    9/   20] | d_loss: 0.7907 | g_loss: 1.0174
Epoch [    9/   20] | d_loss: 0.7007 | g_loss: 3.0953
Epoch [    9/   20] | d_loss: 0.8159 | g_loss: 2.2924
Epoch [    9/   20] | d_loss: 0.9554 | g_loss: 2.5253
Epoch [   10/   20] | d_loss: 1.0512 | g_loss: 1.0499
Epoch [   10/   20] | d_loss: 0.8143 | g_loss: 2.3451
Epoch [   10/   20] | d_loss: 0.8120 | g_loss: 2.4712
Epoch [   10/   20] | d_loss: 0.9735 | g_loss: 2.1034
Epoch [   10/   20] | d_loss: 1.0025 | g_loss: 2.0604
Epoch [   10/   20] | d_loss: 0.6702 | g_loss: 2.8112
Epoch [   10/   20] | d_loss: 0.8546 | g_loss: 1.9547
Epoch [   10/   20] | d_loss: 1.2061 | g_loss: 1.9312
Epoch [   10/   20] | d_loss: 0.9974 | g_loss: 2.7466
Epoch [   10/   20] | d_loss: 1.0738 | g_loss: 3.2056
Epoch [   10/   20] | d_loss: 0.9727 | g_loss: 1.7171
Epoch [   10/   20] | d_loss: 0.6777 | g_loss: 1.5499
Epoch [   10/   20] | d_loss: 0.7212 | g_loss: 2.3411
Epoch [   10/   20] | d_loss: 0.6318 | g_loss: 1.7620
Epoch [   10/   20] | d_loss: 0.5068 | g_loss: 2.4609
Epoch [   10/   20] | d_loss: 0.6704 | g_loss: 1.5372
Epoch [   10/   20] | d_loss: 0.6121 | g_loss: 2.9131
Epoch [   10/   20] | d_loss: 1.4172 | g_loss: 1.1579
Epoch [   10/   20] | d_loss: 0.5868 | g_loss: 2.6120
Epoch [   10/   20] | d_loss: 1.0784 | g_loss: 1.6508
Epoch [   10/   20] | d_loss: 0.8169 | g_loss: 2.2331
Epoch [   10/   20] | d_loss: 0.8817 | g_loss: 1.2021
Epoch [   10/   20] | d_loss: 0.6113 | g_loss: 1.9642
Epoch [   10/   20] | d_loss: 0.6352 | g_loss: 1.7044
Epoch [   10/   20] | d_loss: 0.6933 | g_loss: 2.1638
Epoch [   10/   20] | d_loss: 1.0306 | g_loss: 1.9590
Epoch [   10/   20] | d_loss: 1.1437 | g_loss: 2.1098
Epoch [   10/   20] | d_loss: 1.3516 | g_loss: 2.4633
Epoch [   10/   20] | d_loss: 1.0199 | g_loss: 2.3237
Epoch [   10/   20] | d_loss: 1.1409 | g_loss: 2.4938
Epoch [   10/   20] | d_loss: 0.7194 | g_loss: 1.4005
Epoch [   10/   20] | d_loss: 0.5241 | g_loss: 1.8518
Epoch [   10/   20] | d_loss: 1.0541 | g_loss: 2.3646
Epoch [   10/   20] | d_loss: 0.8564 | g_loss: 3.2623
Epoch [   10/   20] | d_loss: 1.2069 | g_loss: 1.3519
Epoch [   10/   20] | d_loss: 0.8306 | g_loss: 2.3960
```

```
Epoch [   10/   20] | d_loss: 0.7389 | g_loss: 1.7327
Epoch [   10/   20] | d_loss: 0.7164 | g_loss: 0.9886
Epoch [   10/   20] | d_loss: 0.8201 | g_loss: 2.0292
Epoch [   10/   20] | d_loss: 0.7873 | g_loss: 2.0382
Epoch [   10/   20] | d_loss: 1.5792 | g_loss: 1.2192
Epoch [   10/   20] | d_loss: 0.6874 | g_loss: 1.1817
Epoch [   10/   20] | d_loss: 1.5140 | g_loss: 1.4178
Epoch [   10/   20] | d_loss: 1.1923 | g_loss: 1.4457
Epoch [   10/   20] | d_loss: 1.0105 | g_loss: 2.7703
Epoch [   10/   20] | d_loss: 0.7807 | g_loss: 1.7882
Epoch [   10/   20] | d_loss: 0.7691 | g_loss: 1.9226
Epoch [   10/   20] | d_loss: 1.2896 | g_loss: 2.5494
Epoch [   10/   20] | d_loss: 0.8661 | g_loss: 4.0807
Epoch [   10/   20] | d_loss: 0.9140 | g_loss: 2.4824
Epoch [   10/   20] | d_loss: 1.4228 | g_loss: 2.1107
Epoch [   10/   20] | d_loss: 1.3455 | g_loss: 1.6029
Epoch [   10/   20] | d_loss: 0.6939 | g_loss: 3.2343
Epoch [   10/   20] | d_loss: 1.2694 | g_loss: 0.8288
Epoch [   10/   20] | d_loss: 0.8768 | g_loss: 1.9760
Epoch [   10/   20] | d_loss: 0.8660 | g_loss: 2.6614
Epoch [   10/   20] | d_loss: 0.9840 | g_loss: 1.8546
Epoch [   11/   20] | d_loss: 1.0583 | g_loss: 1.1410
Epoch [   11/   20] | d_loss: 1.0229 | g_loss: 1.2665
Epoch [   11/   20] | d_loss: 1.3764 | g_loss: 1.1308
Epoch [   11/   20] | d_loss: 0.8307 | g_loss: 1.8961
Epoch [   11/   20] | d_loss: 0.7443 | g_loss: 2.3635
Epoch [   11/   20] | d_loss: 0.6494 | g_loss: 2.9373
Epoch [   11/   20] | d_loss: 0.8008 | g_loss: 2.6747
Epoch [   11/   20] | d_loss: 0.6632 | g_loss: 1.3517
Epoch [   11/   20] | d_loss: 0.8178 | g_loss: 2.4439
Epoch [   11/   20] | d_loss: 0.8109 | g_loss: 1.7945
Epoch [   11/   20] | d_loss: 0.7457 | g_loss: 2.5001
Epoch [   11/   20] | d_loss: 1.0695 | g_loss: 1.5002
Epoch [   11/   20] | d_loss: 0.7995 | g_loss: 1.8809
Epoch [   11/   20] | d_loss: 0.4994 | g_loss: 2.7949
Epoch [   11/   20] | d_loss: 1.3293 | g_loss: 1.9701
Epoch [   11/   20] | d_loss: 1.1794 | g_loss: 1.2487
Epoch [   11/   20] | d_loss: 0.7724 | g_loss: 2.1893
Epoch [   11/   20] | d_loss: 1.1044 | g_loss: 2.0210
Epoch [   11/   20] | d_loss: 1.2017 | g_loss: 1.2567
Epoch [   11/   20] | d_loss: 1.1828 | g_loss: 1.7897
Epoch [   11/   20] | d_loss: 1.0272 | g_loss: 1.7311
Epoch [   11/   20] | d_loss: 0.7241 | g_loss: 1.6426
Epoch [   11/   20] | d_loss: 1.0612 | g_loss: 1.8093
Epoch [   11/   20] | d_loss: 0.9432 | g_loss: 3.3945
Epoch [   11/   20] | d_loss: 1.0017 | g_loss: 1.6286
Epoch [   11/   20] | d_loss: 0.6023 | g_loss: 2.5788
Epoch [   11/   20] | d_loss: 0.9020 | g_loss: 1.9261
```

```
Epoch [   11/   20] | d_loss: 0.7430 | g_loss: 2.5351
Epoch [   11/   20] | d_loss: 0.6537 | g_loss: 3.9969
Epoch [   11/   20] | d_loss: 0.7229 | g_loss: 2.8053
Epoch [   11/   20] | d_loss: 1.1724 | g_loss: 2.9201
Epoch [   11/   20] | d_loss: 0.7322 | g_loss: 1.7223
Epoch [   11/   20] | d_loss: 0.7636 | g_loss: 2.1063
Epoch [   11/   20] | d_loss: 0.6407 | g_loss: 3.8544
Epoch [   11/   20] | d_loss: 1.3011 | g_loss: 1.2367
Epoch [   11/   20] | d_loss: 0.8339 | g_loss: 2.4530
Epoch [   11/   20] | d_loss: 1.1833 | g_loss: 2.5081
Epoch [   11/   20] | d_loss: 0.7424 | g_loss: 2.0304
Epoch [   11/   20] | d_loss: 0.5180 | g_loss: 1.8731
Epoch [   11/   20] | d_loss: 0.7543 | g_loss: 2.2796
Epoch [   11/   20] | d_loss: 0.7932 | g_loss: 0.9287
Epoch [   11/   20] | d_loss: 1.0160 | g_loss: 1.8964
Epoch [   11/   20] | d_loss: 0.6824 | g_loss: 2.6888
Epoch [   11/   20] | d_loss: 0.5437 | g_loss: 2.6340
Epoch [   11/   20] | d_loss: 0.5948 | g_loss: 2.4756
Epoch [   11/   20] | d_loss: 0.9285 | g_loss: 2.2105
Epoch [   11/   20] | d_loss: 0.5390 | g_loss: 2.7158
Epoch [   11/   20] | d_loss: 0.8970 | g_loss: 2.4838
Epoch [   11/   20] | d_loss: 1.1088 | g_loss: 1.8450
Epoch [   11/   20] | d_loss: 1.0986 | g_loss: 1.1351
Epoch [   11/   20] | d_loss: 0.9969 | g_loss: 1.3477
Epoch [   11/   20] | d_loss: 0.8520 | g_loss: 2.0100
Epoch [   11/   20] | d_loss: 0.9686 | g_loss: 1.7592
Epoch [   11/   20] | d_loss: 1.0549 | g_loss: 1.9004
Epoch [   11/   20] | d_loss: 0.8198 | g_loss: 2.1203
Epoch [   11/   20] | d_loss: 0.9208 | g_loss: 2.0379
Epoch [   11/   20] | d_loss: 0.8502 | g_loss: 1.3376
Epoch [   12/   20] | d_loss: 0.8349 | g_loss: 1.6693
Epoch [   12/   20] | d_loss: 0.9142 | g_loss: 2.2833
Epoch [   12/   20] | d_loss: 0.9239 | g_loss: 2.6598
Epoch [   12/   20] | d_loss: 0.7118 | g_loss: 1.8293
Epoch [   12/   20] | d_loss: 0.9603 | g_loss: 1.6718
Epoch [   12/   20] | d_loss: 0.9519 | g_loss: 2.5433
Epoch [   12/   20] | d_loss: 2.1665 | g_loss: 1.0754
Epoch [   12/   20] | d_loss: 1.4691 | g_loss: 1.1451
Epoch [   12/   20] | d_loss: 0.6288 | g_loss: 2.1011
Epoch [   12/   20] | d_loss: 0.9008 | g_loss: 2.6876
Epoch [   12/   20] | d_loss: 0.6861 | g_loss: 2.4360
Epoch [   12/   20] | d_loss: 0.6669 | g_loss: 2.6263
Epoch [   12/   20] | d_loss: 0.6634 | g_loss: 2.2835
Epoch [   12/   20] | d_loss: 0.5989 | g_loss: 1.7570
Epoch [   12/   20] | d_loss: 0.8619 | g_loss: 1.9932
Epoch [   12/   20] | d_loss: 0.8641 | g_loss: 2.2038
Epoch [   12/   20] | d_loss: 1.3999 | g_loss: 2.7109
Epoch [   12/   20] | d_loss: 0.9189 | g_loss: 1.6294
```

```
Epoch [   12/   20] | d_loss: 0.6185 | g_loss: 1.7377
Epoch [   12/   20] | d_loss: 0.7670 | g_loss: 1.5876
Epoch [   12/   20] | d_loss: 0.6118 | g_loss: 2.4857
Epoch [   12/   20] | d_loss: 1.0499 | g_loss: 2.1376
Epoch [   12/   20] | d_loss: 0.9243 | g_loss: 1.1888
Epoch [   12/   20] | d_loss: 0.6717 | g_loss: 0.9377
Epoch [   12/   20] | d_loss: 0.6794 | g_loss: 3.0587
Epoch [   12/   20] | d_loss: 0.7906 | g_loss: 1.6634
Epoch [   12/   20] | d_loss: 1.1769 | g_loss: 1.5246
Epoch [   12/   20] | d_loss: 0.7740 | g_loss: 1.4364
Epoch [   12/   20] | d_loss: 0.8622 | g_loss: 3.1335
Epoch [   12/   20] | d_loss: 0.6151 | g_loss: 2.7097
Epoch [   12/   20] | d_loss: 1.2825 | g_loss: 2.1065
Epoch [   12/   20] | d_loss: 0.7534 | g_loss: 2.0775
Epoch [   12/   20] | d_loss: 0.5110 | g_loss: 3.4616
Epoch [   12/   20] | d_loss: 0.9491 | g_loss: 1.0655
Epoch [   12/   20] | d_loss: 0.8709 | g_loss: 3.5972
Epoch [   12/   20] | d_loss: 0.7070 | g_loss: 2.6101
Epoch [   12/   20] | d_loss: 0.7544 | g_loss: 2.9295
Epoch [   12/   20] | d_loss: 0.8358 | g_loss: 2.3395
Epoch [   12/   20] | d_loss: 0.6654 | g_loss: 2.6731
Epoch [   12/   20] | d_loss: 0.7862 | g_loss: 1.7343
Epoch [   12/   20] | d_loss: 0.8280 | g_loss: 1.6613
Epoch [   12/   20] | d_loss: 0.7375 | g_loss: 2.1069
Epoch [   12/   20] | d_loss: 0.8701 | g_loss: 1.2782
Epoch [   12/   20] | d_loss: 0.9104 | g_loss: 1.8193
Epoch [   12/   20] | d_loss: 1.1433 | g_loss: 1.7419
Epoch [   12/   20] | d_loss: 1.1412 | g_loss: 1.2031
Epoch [   12/   20] | d_loss: 0.5641 | g_loss: 2.2382
Epoch [   12/   20] | d_loss: 0.8329 | g_loss: 1.6403
Epoch [   12/   20] | d_loss: 0.6120 | g_loss: 2.5593
Epoch [   12/   20] | d_loss: 0.8342 | g_loss: 1.3488
Epoch [   12/   20] | d_loss: 0.6363 | g_loss: 1.3277
Epoch [   12/   20] | d_loss: 1.6017 | g_loss: 1.7677
Epoch [   12/   20] | d_loss: 0.4985 | g_loss: 2.1848
Epoch [   12/   20] | d_loss: 0.8673 | g_loss: 1.6973
Epoch [   12/   20] | d_loss: 1.1007 | g_loss: 2.0663
Epoch [   12/   20] | d_loss: 1.7783 | g_loss: 2.9400
Epoch [   12/   20] | d_loss: 0.8479 | g_loss: 1.3612
Epoch [   13/   20] | d_loss: 1.0811 | g_loss: 1.8548
Epoch [   13/   20] | d_loss: 1.0037 | g_loss: 1.5085
Epoch [   13/   20] | d_loss: 0.7087 | g_loss: 1.9632
Epoch [   13/   20] | d_loss: 1.0003 | g_loss: 2.5180
Epoch [   13/   20] | d_loss: 1.2275 | g_loss: 1.5070
Epoch [   13/   20] | d_loss: 0.9931 | g_loss: 2.4093
Epoch [   13/   20] | d_loss: 0.7622 | g_loss: 1.7991
Epoch [   13/   20] | d_loss: 0.7369 | g_loss: 1.7393
Epoch [   13/   20] | d_loss: 0.6865 | g_loss: 1.4352
```

```
Epoch [   13/   20] | d_loss: 0.8622 | g_loss: 3.1865
Epoch [   13/   20] | d_loss: 0.8305 | g_loss: 1.3625
Epoch [   13/   20] | d_loss: 0.6657 | g_loss: 1.7578
Epoch [   13/   20] | d_loss: 0.9207 | g_loss: 3.3307
Epoch [   13/   20] | d_loss: 0.5324 | g_loss: 2.3157
Epoch [   13/   20] | d_loss: 0.7678 | g_loss: 2.7930
Epoch [   13/   20] | d_loss: 0.7884 | g_loss: 2.6841
Epoch [   13/   20] | d_loss: 0.6324 | g_loss: 2.4080
Epoch [   13/   20] | d_loss: 0.9327 | g_loss: 2.4433
Epoch [   13/   20] | d_loss: 0.7720 | g_loss: 1.5235
Epoch [   13/   20] | d_loss: 0.8538 | g_loss: 1.4712
Epoch [   13/   20] | d_loss: 0.6654 | g_loss: 2.2614
Epoch [   13/   20] | d_loss: 0.7834 | g_loss: 3.4943
Epoch [   13/   20] | d_loss: 0.6431 | g_loss: 2.2454
Epoch [   13/   20] | d_loss: 0.6543 | g_loss: 2.5686
Epoch [   13/   20] | d_loss: 0.8663 | g_loss: 1.6647
Epoch [   13/   20] | d_loss: 0.7846 | g_loss: 1.6436
Epoch [   13/   20] | d_loss: 0.9141 | g_loss: 1.6811
Epoch [   13/   20] | d_loss: 0.9476 | g_loss: 2.6255
Epoch [   13/   20] | d_loss: 0.8612 | g_loss: 2.3644
Epoch [   13/   20] | d_loss: 0.6452 | g_loss: 1.9365
Epoch [   13/   20] | d_loss: 0.8770 | g_loss: 2.4096
Epoch [   13/   20] | d_loss: 0.7452 | g_loss: 2.6177
Epoch [   13/   20] | d_loss: 0.7976 | g_loss: 1.4218
Epoch [   13/   20] | d_loss: 1.0160 | g_loss: 2.4934
Epoch [   13/   20] | d_loss: 1.0639 | g_loss: 2.4466
Epoch [   13/   20] | d_loss: 0.7034 | g_loss: 2.1513
Epoch [   13/   20] | d_loss: 1.4920 | g_loss: 2.6873
Epoch [   13/   20] | d_loss: 1.3166 | g_loss: 1.5136
Epoch [   13/   20] | d_loss: 0.6524 | g_loss: 1.3054
Epoch [   13/   20] | d_loss: 1.1444 | g_loss: 1.7232
Epoch [   13/   20] | d_loss: 0.9726 | g_loss: 1.8137
Epoch [   13/   20] | d_loss: 0.8719 | g_loss: 1.7782
Epoch [   13/   20] | d_loss: 0.6694 | g_loss: 1.9829
Epoch [   13/   20] | d_loss: 0.9114 | g_loss: 1.7468
Epoch [   13/   20] | d_loss: 1.0380 | g_loss: 2.9024
Epoch [   13/   20] | d_loss: 0.8718 | g_loss: 2.0894
Epoch [   13/   20] | d_loss: 0.5736 | g_loss: 1.8192
Epoch [   13/   20] | d_loss: 1.2947 | g_loss: 2.0543
Epoch [   13/   20] | d_loss: 1.3861 | g_loss: 2.2540
Epoch [   13/   20] | d_loss: 0.8377 | g_loss: 2.8135
Epoch [   13/   20] | d_loss: 0.5706 | g_loss: 3.2303
Epoch [   13/   20] | d_loss: 1.2595 | g_loss: 2.1464
Epoch [   13/   20] | d_loss: 0.7651 | g_loss: 2.3553
Epoch [   13/   20] | d_loss: 1.1259 | g_loss: 2.8996
Epoch [   13/   20] | d_loss: 0.6878 | g_loss: 1.7747
Epoch [   13/   20] | d_loss: 1.1499 | g_loss: 2.2945
Epoch [   13/   20] | d_loss: 0.9429 | g_loss: 2.1055
```

```
Epoch [   14/   20] | d_loss: 1.2740 | g_loss: 1.8179
Epoch [   14/   20] | d_loss: 0.9777 | g_loss: 1.5269
Epoch [   14/   20] | d_loss: 0.7362 | g_loss: 2.0298
Epoch [   14/   20] | d_loss: 0.7028 | g_loss: 1.9201
Epoch [   14/   20] | d_loss: 0.8644 | g_loss: 2.1528
Epoch [   14/   20] | d_loss: 0.9921 | g_loss: 1.1206
Epoch [   14/   20] | d_loss: 0.7099 | g_loss: 1.7517
Epoch [   14/   20] | d_loss: 0.7797 | g_loss: 2.5758
Epoch [   14/   20] | d_loss: 0.8123 | g_loss: 3.0986
Epoch [   14/   20] | d_loss: 1.0557 | g_loss: 1.9375
Epoch [   14/   20] | d_loss: 1.0205 | g_loss: 1.1121
Epoch [   14/   20] | d_loss: 0.5840 | g_loss: 2.9331
Epoch [   14/   20] | d_loss: 0.6891 | g_loss: 1.6905
Epoch [   14/   20] | d_loss: 0.6430 | g_loss: 2.3173
Epoch [   14/   20] | d_loss: 0.7533 | g_loss: 1.9212
Epoch [   14/   20] | d_loss: 0.7147 | g_loss: 2.8714
Epoch [   14/   20] | d_loss: 1.0837 | g_loss: 3.0381
Epoch [   14/   20] | d_loss: 0.5956 | g_loss: 2.4516
Epoch [   14/   20] | d_loss: 0.5916 | g_loss: 1.1084
Epoch [   14/   20] | d_loss: 1.7104 | g_loss: 1.9082
Epoch [   14/   20] | d_loss: 0.7259 | g_loss: 1.9869
Epoch [   14/   20] | d_loss: 0.7081 | g_loss: 1.9809
Epoch [   14/   20] | d_loss: 0.8266 | g_loss: 2.2862
Epoch [   14/   20] | d_loss: 1.1873 | g_loss: 1.1504
Epoch [   14/   20] | d_loss: 0.6501 | g_loss: 2.5667
Epoch [   14/   20] | d_loss: 0.5962 | g_loss: 1.8413
Epoch [   14/   20] | d_loss: 0.5375 | g_loss: 2.4155
Epoch [   14/   20] | d_loss: 0.7838 | g_loss: 3.0000
Epoch [   14/   20] | d_loss: 0.4923 | g_loss: 3.4247
Epoch [   14/   20] | d_loss: 0.8401 | g_loss: 1.6430
Epoch [   14/   20] | d_loss: 0.7199 | g_loss: 1.8397
Epoch [   14/   20] | d_loss: 0.5308 | g_loss: 1.4427
Epoch [   14/   20] | d_loss: 0.9072 | g_loss: 1.3427
Epoch [   14/   20] | d_loss: 0.9753 | g_loss: 3.2659
Epoch [   14/   20] | d_loss: 1.0735 | g_loss: 1.4396
Epoch [   14/   20] | d_loss: 1.1187 | g_loss: 2.8004
Epoch [   14/   20] | d_loss: 0.9423 | g_loss: 2.4187
Epoch [   14/   20] | d_loss: 0.7413 | g_loss: 2.4858
Epoch [   14/   20] | d_loss: 1.3018 | g_loss: 1.6220
Epoch [   14/   20] | d_loss: 0.9945 | g_loss: 1.1113
Epoch [   14/   20] | d_loss: 0.5971 | g_loss: 1.2396
Epoch [   14/   20] | d_loss: 0.5772 | g_loss: 2.3390
Epoch [   14/   20] | d_loss: 0.8171 | g_loss: 2.5582
Epoch [   14/   20] | d_loss: 1.1975 | g_loss: 1.0512
Epoch [   14/   20] | d_loss: 0.9063 | g_loss: 2.7857
Epoch [   14/   20] | d_loss: 0.5283 | g_loss: 2.8671
Epoch [   14/   20] | d_loss: 0.8424 | g_loss: 1.4731
Epoch [   14/   20] | d_loss: 0.5966 | g_loss: 2.4843
```

```
Epoch [   14/   20] | d_loss: 0.9153 | g_loss: 2.7967
Epoch [   14/   20] | d_loss: 0.9535 | g_loss: 2.0965
Epoch [   14/   20] | d_loss: 0.7422 | g_loss: 1.5148
Epoch [   14/   20] | d_loss: 0.8377 | g_loss: 2.5290
Epoch [   14/   20] | d_loss: 1.0618 | g_loss: 1.1808
Epoch [   14/   20] | d_loss: 0.9344 | g_loss: 1.3381
Epoch [   14/   20] | d_loss: 0.5994 | g_loss: 1.4026
Epoch [   14/   20] | d_loss: 1.3936 | g_loss: 2.4075
Epoch [   14/   20] | d_loss: 1.0295 | g_loss: 0.7976
Epoch [   15/   20] | d_loss: 0.5931 | g_loss: 2.0979
Epoch [   15/   20] | d_loss: 0.5902 | g_loss: 1.8700
Epoch [   15/   20] | d_loss: 0.6703 | g_loss: 2.5686
Epoch [   15/   20] | d_loss: 0.8310 | g_loss: 2.6152
Epoch [   15/   20] | d_loss: 0.8090 | g_loss: 2.1511
Epoch [   15/   20] | d_loss: 0.7203 | g_loss: 2.4799
Epoch [   15/   20] | d_loss: 0.5369 | g_loss: 1.5427
Epoch [   15/   20] | d_loss: 0.8177 | g_loss: 2.5534
Epoch [   15/   20] | d_loss: 0.6210 | g_loss: 2.0961
Epoch [   15/   20] | d_loss: 1.2136 | g_loss: 1.6601
Epoch [   15/   20] | d_loss: 0.6623 | g_loss: 2.1093
Epoch [   15/   20] | d_loss: 0.6578 | g_loss: 2.3748
Epoch [   15/   20] | d_loss: 0.8960 | g_loss: 1.2108
Epoch [   15/   20] | d_loss: 0.9907 | g_loss: 1.9481
Epoch [   15/   20] | d_loss: 0.8998 | g_loss: 1.9503
Epoch [   15/   20] | d_loss: 0.8386 | g_loss: 1.8928
Epoch [   15/   20] | d_loss: 1.0743 | g_loss: 2.1755
Epoch [   15/   20] | d_loss: 1.1427 | g_loss: 2.5023
Epoch [   15/   20] | d_loss: 0.5590 | g_loss: 2.5243
Epoch [   15/   20] | d_loss: 0.4692 | g_loss: 4.0102
Epoch [   15/   20] | d_loss: 1.1474 | g_loss: 1.8025
Epoch [   15/   20] | d_loss: 1.2782 | g_loss: 2.0035
Epoch [   15/   20] | d_loss: 0.7952 | g_loss: 3.4628
Epoch [   15/   20] | d_loss: 0.6008 | g_loss: 3.2108
Epoch [   15/   20] | d_loss: 0.6210 | g_loss: 1.8595
Epoch [   15/   20] | d_loss: 0.6277 | g_loss: 3.1413
Epoch [   15/   20] | d_loss: 0.6632 | g_loss: 1.3418
Epoch [   15/   20] | d_loss: 0.5386 | g_loss: 2.1345
Epoch [   15/   20] | d_loss: 0.8168 | g_loss: 3.5440
Epoch [   15/   20] | d_loss: 1.0598 | g_loss: 1.9390
Epoch [   15/   20] | d_loss: 0.9321 | g_loss: 2.2803
Epoch [   15/   20] | d_loss: 0.6276 | g_loss: 2.4134
Epoch [   15/   20] | d_loss: 0.5423 | g_loss: 1.6639
Epoch [   15/   20] | d_loss: 0.8474 | g_loss: 2.1702
Epoch [   15/   20] | d_loss: 0.5195 | g_loss: 2.8451
Epoch [   15/   20] | d_loss: 0.6933 | g_loss: 2.1381
Epoch [   15/   20] | d_loss: 1.0315 | g_loss: 2.8259
Epoch [   15/   20] | d_loss: 0.6441 | g_loss: 1.7524
Epoch [   15/   20] | d_loss: 1.4911 | g_loss: 1.6668
```

```
Epoch [   15/   20] | d_loss: 1.0644 | g_loss: 2.9486
Epoch [   15/   20] | d_loss: 0.6567 | g_loss: 2.8730
Epoch [   15/   20] | d_loss: 0.6012 | g_loss: 3.8355
Epoch [   15/   20] | d_loss: 0.5520 | g_loss: 2.2558
Epoch [   15/   20] | d_loss: 0.6177 | g_loss: 2.7650
Epoch [   15/   20] | d_loss: 0.7465 | g_loss: 2.6714
Epoch [   15/   20] | d_loss: 1.1038 | g_loss: 1.4019
Epoch [   15/   20] | d_loss: 0.5039 | g_loss: 2.2442
Epoch [   15/   20] | d_loss: 0.8809 | g_loss: 0.7716
Epoch [   15/   20] | d_loss: 0.7663 | g_loss: 2.7180
Epoch [   15/   20] | d_loss: 1.5398 | g_loss: 2.3821
Epoch [   15/   20] | d_loss: 1.0225 | g_loss: 3.1515
Epoch [   15/   20] | d_loss: 0.6483 | g_loss: 2.4153
Epoch [   15/   20] | d_loss: 0.6063 | g_loss: 3.0700
Epoch [   15/   20] | d_loss: 0.7834 | g_loss: 0.7998
Epoch [   15/   20] | d_loss: 0.9736 | g_loss: 1.3293
Epoch [   15/   20] | d_loss: 0.9427 | g_loss: 2.3545
Epoch [   15/   20] | d_loss: 0.6711 | g_loss: 2.2321
Epoch [   16/   20] | d_loss: 0.8058 | g_loss: 2.3772
Epoch [   16/   20] | d_loss: 0.6469 | g_loss: 1.4919
Epoch [   16/   20] | d_loss: 0.6250 | g_loss: 2.6132
Epoch [   16/   20] | d_loss: 0.5387 | g_loss: 3.0888
Epoch [   16/   20] | d_loss: 0.7706 | g_loss: 2.3900
Epoch [   16/   20] | d_loss: 0.5155 | g_loss: 1.3384
Epoch [   16/   20] | d_loss: 0.6055 | g_loss: 2.0808
Epoch [   16/   20] | d_loss: 1.0702 | g_loss: 1.1748
Epoch [   16/   20] | d_loss: 1.0125 | g_loss: 1.3581
Epoch [   16/   20] | d_loss: 0.5400 | g_loss: 3.1761
Epoch [   16/   20] | d_loss: 0.4674 | g_loss: 3.4390
Epoch [   16/   20] | d_loss: 0.8151 | g_loss: 3.4030
Epoch [   16/   20] | d_loss: 0.7489 | g_loss: 2.8332
Epoch [   16/   20] | d_loss: 1.0263 | g_loss: 3.0564
Epoch [   16/   20] | d_loss: 0.5276 | g_loss: 4.0774
Epoch [   16/   20] | d_loss: 0.8042 | g_loss: 1.4352
Epoch [   16/   20] | d_loss: 1.2452 | g_loss: 1.8593
Epoch [   16/   20] | d_loss: 0.7803 | g_loss: 4.5005
Epoch [   16/   20] | d_loss: 0.8892 | g_loss: 2.3830
Epoch [   16/   20] | d_loss: 0.6096 | g_loss: 3.2828
Epoch [   16/   20] | d_loss: 0.7968 | g_loss: 3.4958
Epoch [   16/   20] | d_loss: 0.7609 | g_loss: 2.8309
Epoch [   16/   20] | d_loss: 1.0750 | g_loss: 1.7746
Epoch [   16/   20] | d_loss: 0.7031 | g_loss: 2.7837
Epoch [   16/   20] | d_loss: 0.8447 | g_loss: 1.5464
Epoch [   16/   20] | d_loss: 0.4187 | g_loss: 2.4520
Epoch [   16/   20] | d_loss: 0.6950 | g_loss: 2.0474
Epoch [   16/   20] | d_loss: 0.5338 | g_loss: 2.6289
Epoch [   16/   20] | d_loss: 1.6015 | g_loss: 3.0962
Epoch [   16/   20] | d_loss: 1.2246 | g_loss: 2.8718
```

```
Epoch [   16/   20] | d_loss: 1.1026 | g_loss: 3.6262
Epoch [   16/   20] | d_loss: 0.7506 | g_loss: 1.6233
Epoch [   16/   20] | d_loss: 0.7807 | g_loss: 2.3179
Epoch [   16/   20] | d_loss: 0.7641 | g_loss: 1.4597
Epoch [   16/   20] | d_loss: 0.4897 | g_loss: 2.4759
Epoch [   16/   20] | d_loss: 0.6194 | g_loss: 1.1375
Epoch [   16/   20] | d_loss: 0.5781 | g_loss: 1.3922
Epoch [   16/   20] | d_loss: 1.1199 | g_loss: 1.1769
Epoch [   16/   20] | d_loss: 1.2359 | g_loss: 2.3224
Epoch [   16/   20] | d_loss: 0.5815 | g_loss: 2.1328
Epoch [   16/   20] | d_loss: 1.1934 | g_loss: 2.0734
Epoch [   16/   20] | d_loss: 0.9385 | g_loss: 2.6502
Epoch [   16/   20] | d_loss: 0.5644 | g_loss: 2.1051
Epoch [   16/   20] | d_loss: 0.8662 | g_loss: 1.5439
Epoch [   16/   20] | d_loss: 0.5457 | g_loss: 1.1409
Epoch [   16/   20] | d_loss: 0.7046 | g_loss: 1.7774
Epoch [   16/   20] | d_loss: 0.8389 | g_loss: 2.6264
Epoch [   16/   20] | d_loss: 0.7247 | g_loss: 2.9109
Epoch [   16/   20] | d_loss: 0.8420 | g_loss: 3.0485
Epoch [   16/   20] | d_loss: 0.9493 | g_loss: 3.6272
Epoch [   16/   20] | d_loss: 0.7761 | g_loss: 2.3473
Epoch [   16/   20] | d_loss: 0.5224 | g_loss: 2.6953
Epoch [   16/   20] | d_loss: 0.8810 | g_loss: 2.0656
Epoch [   16/   20] | d_loss: 1.1909 | g_loss: 2.6195
Epoch [   16/   20] | d_loss: 0.7435 | g_loss: 2.3253
Epoch [   16/   20] | d_loss: 0.8058 | g_loss: 2.1462
Epoch [   16/   20] | d_loss: 0.5116 | g_loss: 3.2302
Epoch [   17/   20] | d_loss: 0.5442 | g_loss: 1.4460
Epoch [   17/   20] | d_loss: 0.5413 | g_loss: 3.3535
Epoch [   17/   20] | d_loss: 0.5469 | g_loss: 2.8652
Epoch [   17/   20] | d_loss: 0.8004 | g_loss: 2.1881
Epoch [   17/   20] | d_loss: 0.6837 | g_loss: 2.2428
Epoch [   17/   20] | d_loss: 0.7177 | g_loss: 2.4053
Epoch [   17/   20] | d_loss: 0.7028 | g_loss: 3.2445
Epoch [   17/   20] | d_loss: 0.5630 | g_loss: 3.1835
Epoch [   17/   20] | d_loss: 0.7838 | g_loss: 3.1248
Epoch [   17/   20] | d_loss: 0.9326 | g_loss: 1.8044
Epoch [   17/   20] | d_loss: 0.6480 | g_loss: 2.0256
Epoch [   17/   20] | d_loss: 0.6951 | g_loss: 1.7096
Epoch [   17/   20] | d_loss: 0.8858 | g_loss: 3.1829
Epoch [   17/   20] | d_loss: 0.5658 | g_loss: 2.6337
Epoch [   17/   20] | d_loss: 0.8150 | g_loss: 2.2932
Epoch [   17/   20] | d_loss: 0.8911 | g_loss: 2.3870
Epoch [   17/   20] | d_loss: 0.6673 | g_loss: 3.2768
Epoch [   17/   20] | d_loss: 0.5074 | g_loss: 2.0498
Epoch [   17/   20] | d_loss: 0.9728 | g_loss: 2.7461
Epoch [   17/   20] | d_loss: 0.6860 | g_loss: 3.4419
Epoch [   17/   20] | d_loss: 0.5877 | g_loss: 1.8639
```

```
Epoch [   17/   20] | d_loss: 0.6915 | g_loss: 1.8448
Epoch [   17/   20] | d_loss: 0.6018 | g_loss: 3.1527
Epoch [   17/   20] | d_loss: 0.5820 | g_loss: 3.0992
Epoch [   17/   20] | d_loss: 0.8517 | g_loss: 2.4713
Epoch [   17/   20] | d_loss: 0.8019 | g_loss: 1.7650
Epoch [   17/   20] | d_loss: 0.6783 | g_loss: 1.8447
Epoch [   17/   20] | d_loss: 1.1697 | g_loss: 1.6909
Epoch [   17/   20] | d_loss: 1.3140 | g_loss: 1.9916
Epoch [   17/   20] | d_loss: 0.7275 | g_loss: 2.1595
Epoch [   17/   20] | d_loss: 0.6965 | g_loss: 2.0113
Epoch [   17/   20] | d_loss: 0.5738 | g_loss: 2.9806
Epoch [   17/   20] | d_loss: 0.3984 | g_loss: 2.4970
Epoch [   17/   20] | d_loss: 0.8555 | g_loss: 2.3000
Epoch [   17/   20] | d_loss: 0.5670 | g_loss: 3.7274
Epoch [   17/   20] | d_loss: 0.9608 | g_loss: 2.5820
Epoch [   17/   20] | d_loss: 0.8849 | g_loss: 3.2872
Epoch [   17/   20] | d_loss: 0.7786 | g_loss: 1.5698
Epoch [   17/   20] | d_loss: 0.7081 | g_loss: 1.9964
Epoch [   17/   20] | d_loss: 0.9466 | g_loss: 2.5659
Epoch [   17/   20] | d_loss: 0.6524 | g_loss: 2.5837
Epoch [   17/   20] | d_loss: 0.5438 | g_loss: 1.3879
Epoch [   17/   20] | d_loss: 0.6652 | g_loss: 1.5976
Epoch [   17/   20] | d_loss: 0.6218 | g_loss: 2.9459
Epoch [   17/   20] | d_loss: 0.9822 | g_loss: 1.5928
Epoch [   17/   20] | d_loss: 0.5721 | g_loss: 2.6007
Epoch [   17/   20] | d_loss: 0.5330 | g_loss: 1.8301
Epoch [   17/   20] | d_loss: 0.6160 | g_loss: 2.3702
Epoch [   17/   20] | d_loss: 1.1669 | g_loss: 2.4051
Epoch [   17/   20] | d_loss: 0.6798 | g_loss: 2.0011
Epoch [   17/   20] | d_loss: 1.3019 | g_loss: 2.9528
Epoch [   17/   20] | d_loss: 0.5732 | g_loss: 2.1153
Epoch [   17/   20] | d_loss: 0.7787 | g_loss: 3.2821
Epoch [   17/   20] | d_loss: 0.9609 | g_loss: 2.0011
Epoch [   17/   20] | d_loss: 0.5489 | g_loss: 2.7939
Epoch [   17/   20] | d_loss: 0.8883 | g_loss: 2.0143
Epoch [   17/   20] | d_loss: 0.7772 | g_loss: 2.3836
Epoch [   18/   20] | d_loss: 0.7827 | g_loss: 2.3279
Epoch [   18/   20] | d_loss: 0.6529 | g_loss: 2.5776
Epoch [   18/   20] | d_loss: 1.0637 | g_loss: 3.3406
Epoch [   18/   20] | d_loss: 0.7554 | g_loss: 2.2833
Epoch [   18/   20] | d_loss: 1.6443 | g_loss: 2.3834
Epoch [   18/   20] | d_loss: 0.5132 | g_loss: 3.2191
Epoch [   18/   20] | d_loss: 0.5532 | g_loss: 3.2277
Epoch [   18/   20] | d_loss: 0.7034 | g_loss: 2.5730
Epoch [   18/   20] | d_loss: 1.0738 | g_loss: 2.2131
Epoch [   18/   20] | d_loss: 1.0019 | g_loss: 1.6097
Epoch [   18/   20] | d_loss: 1.0275 | g_loss: 2.6202
Epoch [   18/   20] | d_loss: 0.7459 | g_loss: 2.0738
```

```
Epoch [   18/   20] | d_loss: 0.8917 | g_loss: 2.8130
Epoch [   18/   20] | d_loss: 0.7123 | g_loss: 2.4916
Epoch [   18/   20] | d_loss: 1.3290 | g_loss: 1.9162
Epoch [   18/   20] | d_loss: 0.8120 | g_loss: 3.9262
Epoch [   18/   20] | d_loss: 1.0496 | g_loss: 3.0416
Epoch [   18/   20] | d_loss: 0.9931 | g_loss: 1.4097
Epoch [   18/   20] | d_loss: 0.7568 | g_loss: 1.5599
Epoch [   18/   20] | d_loss: 0.6217 | g_loss: 2.8971
Epoch [   18/   20] | d_loss: 0.5928 | g_loss: 3.5630
Epoch [   18/   20] | d_loss: 0.8999 | g_loss: 1.1092
Epoch [   18/   20] | d_loss: 0.4923 | g_loss: 3.8426
Epoch [   18/   20] | d_loss: 0.7802 | g_loss: 2.8676
Epoch [   18/   20] | d_loss: 0.8009 | g_loss: 3.7081
Epoch [   18/   20] | d_loss: 0.7012 | g_loss: 2.5884
Epoch [   18/   20] | d_loss: 1.1585 | g_loss: 3.7037
Epoch [   18/   20] | d_loss: 0.7057 | g_loss: 2.7349
Epoch [   18/   20] | d_loss: 0.8640 | g_loss: 1.7636
Epoch [   18/   20] | d_loss: 0.5208 | g_loss: 3.2720
Epoch [   18/   20] | d_loss: 0.5190 | g_loss: 2.4661
Epoch [   18/   20] | d_loss: 0.7642 | g_loss: 1.9514
Epoch [   18/   20] | d_loss: 0.5865 | g_loss: 2.8906
Epoch [   18/   20] | d_loss: 0.6109 | g_loss: 3.4088
Epoch [   18/   20] | d_loss: 0.5690 | g_loss: 1.9385
Epoch [   18/   20] | d_loss: 0.6005 | g_loss: 2.6348
Epoch [   18/   20] | d_loss: 0.6583 | g_loss: 2.8178
Epoch [   18/   20] | d_loss: 0.7407 | g_loss: 1.3495
Epoch [   18/   20] | d_loss: 0.9958 | g_loss: 2.0336
Epoch [   18/   20] | d_loss: 0.7320 | g_loss: 3.2602
Epoch [   18/   20] | d_loss: 0.8400 | g_loss: 2.0187
Epoch [   18/   20] | d_loss: 0.6358 | g_loss: 3.6595
Epoch [   18/   20] | d_loss: 0.6716 | g_loss: 3.4455
Epoch [   18/   20] | d_loss: 0.6674 | g_loss: 2.1010
Epoch [   18/   20] | d_loss: 0.9175 | g_loss: 2.0061
Epoch [   18/   20] | d_loss: 0.7876 | g_loss: 2.4048
Epoch [   18/   20] | d_loss: 1.0223 | g_loss: 2.0445
Epoch [   18/   20] | d_loss: 0.6393 | g_loss: 2.1354
Epoch [   18/   20] | d_loss: 0.6395 | g_loss: 2.2431
Epoch [   18/   20] | d_loss: 0.7933 | g_loss: 2.1143
Epoch [   18/   20] | d_loss: 0.5078 | g_loss: 3.0505
Epoch [   18/   20] | d_loss: 0.7487 | g_loss: 2.0351
Epoch [   18/   20] | d_loss: 0.6201 | g_loss: 2.6515
Epoch [   18/   20] | d_loss: 0.6549 | g_loss: 3.0692
Epoch [   18/   20] | d_loss: 0.6596 | g_loss: 2.7784
Epoch [   18/   20] | d_loss: 0.6770 | g_loss: 1.3617
Epoch [   18/   20] | d_loss: 0.5012 | g_loss: 3.7108
Epoch [   19/   20] | d_loss: 0.7839 | g_loss: 3.0450
Epoch [   19/   20] | d_loss: 0.9493 | g_loss: 2.9600
Epoch [   19/   20] | d_loss: 0.8853 | g_loss: 1.4898
```

```
Epoch [   19/   20] | d_loss: 0.9659 | g_loss: 1.4864
Epoch [   19/   20] | d_loss: 0.6830 | g_loss: 2.5309
Epoch [   19/   20] | d_loss: 0.9917 | g_loss: 2.5348
Epoch [   19/   20] | d_loss: 1.4953 | g_loss: 1.9250
Epoch [   19/   20] | d_loss: 0.4410 | g_loss: 2.9916
Epoch [   19/   20] | d_loss: 0.5552 | g_loss: 2.8080
Epoch [   19/   20] | d_loss: 0.6311 | g_loss: 1.8596
Epoch [   19/   20] | d_loss: 0.6951 | g_loss: 1.2085
Epoch [   19/   20] | d_loss: 0.6199 | g_loss: 1.3496
Epoch [   19/   20] | d_loss: 0.6987 | g_loss: 2.5058
Epoch [   19/   20] | d_loss: 0.7674 | g_loss: 3.3816
Epoch [   19/   20] | d_loss: 0.6044 | g_loss: 2.0638
Epoch [   19/   20] | d_loss: 0.6181 | g_loss: 3.1093
Epoch [   19/   20] | d_loss: 0.7304 | g_loss: 2.0611
Epoch [   19/   20] | d_loss: 0.5230 | g_loss: 3.5598
Epoch [   19/   20] | d_loss: 0.6729 | g_loss: 2.2826
Epoch [   19/   20] | d_loss: 0.6523 | g_loss: 2.9797
Epoch [   19/   20] | d_loss: 0.6964 | g_loss: 1.0485
Epoch [   19/   20] | d_loss: 0.9815 | g_loss: 1.7044
Epoch [   19/   20] | d_loss: 0.4850 | g_loss: 3.0402
Epoch [   19/   20] | d_loss: 0.8843 | g_loss: 3.0263
Epoch [   19/   20] | d_loss: 0.6680 | g_loss: 1.6425
Epoch [   19/   20] | d_loss: 0.6254 | g_loss: 2.5503
Epoch [   19/   20] | d_loss: 0.8843 | g_loss: 2.9107
Epoch [   19/   20] | d_loss: 0.5059 | g_loss: 2.3863
Epoch [   19/   20] | d_loss: 0.7862 | g_loss: 2.0949
Epoch [   19/   20] | d_loss: 0.8642 | g_loss: 1.2420
Epoch [   19/   20] | d_loss: 0.5017 | g_loss: 2.2498
Epoch [   19/   20] | d_loss: 0.9011 | g_loss: 2.4869
Epoch [   19/   20] | d_loss: 0.7912 | g_loss: 3.2564
Epoch [   19/   20] | d_loss: 0.9036 | g_loss: 2.3072
Epoch [   19/   20] | d_loss: 0.6537 | g_loss: 2.4892
Epoch [   19/   20] | d_loss: 0.9856 | g_loss: 1.8230
Epoch [   19/   20] | d_loss: 0.6974 | g_loss: 3.5683
Epoch [   19/   20] | d_loss: 0.8720 | g_loss: 1.7102
Epoch [   19/   20] | d_loss: 0.5744 | g_loss: 4.1938
Epoch [   19/   20] | d_loss: 0.7319 | g_loss: 2.0310
Epoch [   19/   20] | d_loss: 0.6079 | g_loss: 3.2692
Epoch [   19/   20] | d_loss: 0.6811 | g_loss: 1.4666
Epoch [   19/   20] | d_loss: 0.9788 | g_loss: 3.3136
Epoch [   19/   20] | d_loss: 0.8530 | g_loss: 2.5627
Epoch [   19/   20] | d_loss: 0.6760 | g_loss: 1.1650
Epoch [   19/   20] | d_loss: 0.6712 | g_loss: 2.0366
Epoch [   19/   20] | d_loss: 0.8386 | g_loss: 3.1648
Epoch [   19/   20] | d_loss: 0.6143 | g_loss: 1.8054
Epoch [   19/   20] | d_loss: 0.7175 | g_loss: 1.5347
Epoch [   19/   20] | d_loss: 0.7970 | g_loss: 1.2423
Epoch [   19/   20] | d_loss: 0.5639 | g_loss: 1.9605
```

```
Epoch [   19/   20] | d_loss: 0.9214 | g_loss: 1.6524
Epoch [   19/   20] | d_loss: 0.5339 | g_loss: 2.6060
Epoch [   19/   20] | d_loss: 0.6773 | g_loss: 2.3793
Epoch [   19/   20] | d_loss: 1.0604 | g_loss: 2.6288
Epoch [   19/   20] | d_loss: 0.5649 | g_loss: 2.2232
Epoch [   19/   20] | d_loss: 0.5027 | g_loss: 2.1210
Epoch [   20/   20] | d_loss: 0.7218 | g_loss: 2.3684
Epoch [   20/   20] | d_loss: 1.1098 | g_loss: 2.8287
Epoch [   20/   20] | d_loss: 0.5236 | g_loss: 2.5820
Epoch [   20/   20] | d_loss: 0.4824 | g_loss: 2.2719
Epoch [   20/   20] | d_loss: 1.1892 | g_loss: 1.8896
Epoch [   20/   20] | d_loss: 1.6209 | g_loss: 0.6663
Epoch [   20/   20] | d_loss: 0.8145 | g_loss: 3.1186
Epoch [   20/   20] | d_loss: 0.6474 | g_loss: 1.9422
Epoch [   20/   20] | d_loss: 0.5925 | g_loss: 2.1051
Epoch [   20/   20] | d_loss: 1.0092 | g_loss: 1.1114
Epoch [   20/   20] | d_loss: 0.6464 | g_loss: 4.2825
Epoch [   20/   20] | d_loss: 0.5464 | g_loss: 2.0075
Epoch [   20/   20] | d_loss: 0.6921 | g_loss: 2.9085
Epoch [   20/   20] | d_loss: 1.1326 | g_loss: 2.5260
Epoch [   20/   20] | d_loss: 0.7264 | g_loss: 2.0625
Epoch [   20/   20] | d_loss: 0.4747 | g_loss: 3.3205
Epoch [   20/   20] | d_loss: 0.8937 | g_loss: 1.9346
Epoch [   20/   20] | d_loss: 0.7077 | g_loss: 2.2547
Epoch [   20/   20] | d_loss: 0.4992 | g_loss: 3.5804
Epoch [   20/   20] | d_loss: 0.6453 | g_loss: 2.9467
Epoch [   20/   20] | d_loss: 0.6017 | g_loss: 3.2960
Epoch [   20/   20] | d_loss: 0.4938 | g_loss: 2.2505
Epoch [   20/   20] | d_loss: 1.0161 | g_loss: 2.1228
Epoch [   20/   20] | d_loss: 0.8263 | g_loss: 2.2612
Epoch [   20/   20] | d_loss: 1.2367 | g_loss: 2.3257
Epoch [   20/   20] | d_loss: 1.5403 | g_loss: 2.5413
Epoch [   20/   20] | d_loss: 0.6304 | g_loss: 1.1448
Epoch [   20/   20] | d_loss: 0.7135 | g_loss: 2.2574
Epoch [   20/   20] | d_loss: 0.5186 | g_loss: 2.7045
Epoch [   20/   20] | d_loss: 0.8641 | g_loss: 1.6471
Epoch [   20/   20] | d_loss: 0.7141 | g_loss: 2.2868
Epoch [   20/   20] | d_loss: 0.6510 | g_loss: 1.2789
Epoch [   20/   20] | d_loss: 1.0150 | g_loss: 1.8404
Epoch [   20/   20] | d_loss: 1.3549 | g_loss: 3.7053
Epoch [   20/   20] | d_loss: 0.5677 | g_loss: 2.0687
Epoch [   20/   20] | d_loss: 0.5162 | g_loss: 2.8573
Epoch [   20/   20] | d_loss: 0.6188 | g_loss: 2.1754
Epoch [   20/   20] | d_loss: 0.8311 | g_loss: 2.6876
Epoch [   20/   20] | d_loss: 0.5172 | g_loss: 2.0483
Epoch [   20/   20] | d_loss: 1.2065 | g_loss: 2.5250
Epoch [   20/   20] | d_loss: 0.7892 | g_loss: 2.3359
Epoch [   20/   20] | d_loss: 0.7492 | g_loss: 2.1295
```

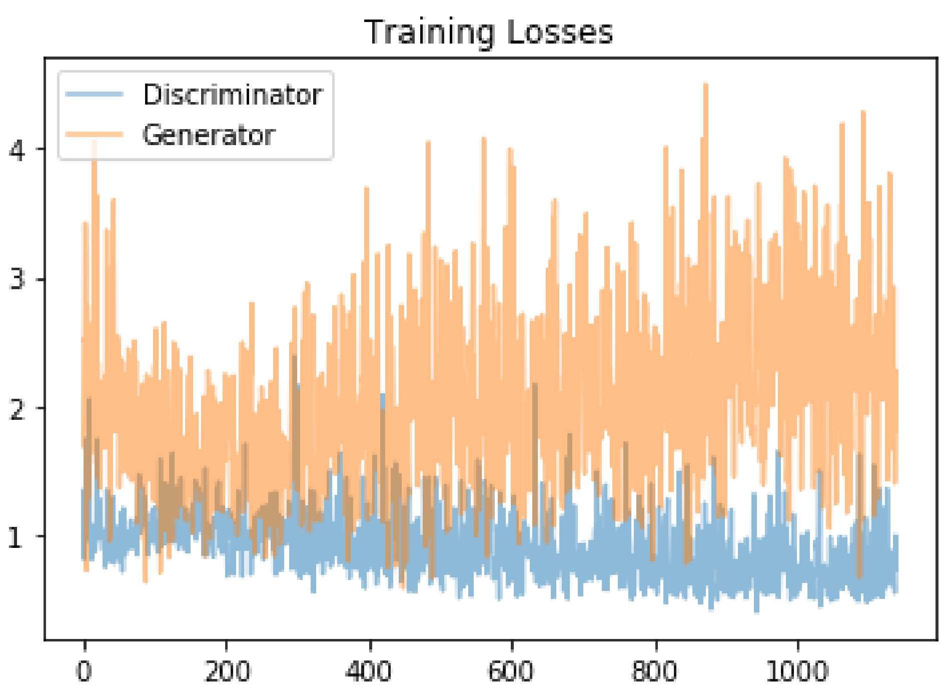
```
Epoch [   20/   20] | d_loss: 0.5869 | g_loss: 2.5117
Epoch [   20/   20] | d_loss: 1.3604 | g_loss: 2.8163
Epoch [   20/   20] | d_loss: 0.8007 | g_loss: 1.4255
Epoch [   20/   20] | d_loss: 0.8116 | g_loss: 2.4831
Epoch [   20/   20] | d_loss: 0.8342 | g_loss: 2.8697
Epoch [   20/   20] | d_loss: 0.6238 | g_loss: 3.8051
Epoch [   20/   20] | d_loss: 0.8757 | g_loss: 2.1380
Epoch [   20/   20] | d_loss: 0.8475 | g_loss: 2.8795
Epoch [   20/   20] | d_loss: 0.6528 | g_loss: 2.4820
Epoch [   20/   20] | d_loss: 0.6074 | g_loss: 1.7928
Epoch [   20/   20] | d_loss: 0.6357 | g_loss: 1.6599
Epoch [   20/   20] | d_loss: 0.6990 | g_loss: 2.9249
Epoch [   20/   20] | d_loss: 0.5469 | g_loss: 1.3986
Epoch [   20/   20] | d_loss: 0.9948 | g_loss: 2.2753
Epoch [   20/   20] | d_loss: 0.7320 | g_loss: 1.8144
```

2.9 Training loss

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

In [21]:
```
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[21]: <matplotlib.legend.Legend at 0x7fdd3c16deb8>



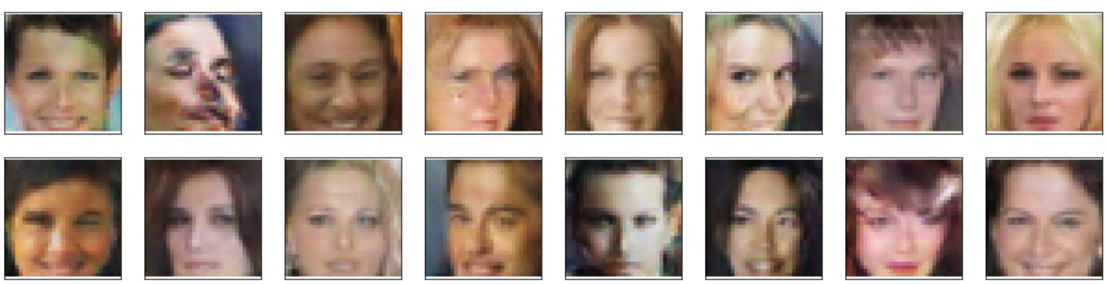
2.10 Generator samples from training

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

```
In [22]: # 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=True
             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 [23]: # Load samples from generator, taken while training
         with open('train_samples.pkl', 'rb') as f:
             samples = pkl.load(f)
```

```
In [24]: _ = view_samples(-1, samples)
```



2.10.1 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: - 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 characteristics of the faces.
```

-I used suggested beta1 0.5 and it generated more types of faces than beta1 0.1 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.
```

2.10.2 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.