dlnd_face_generation

July 20, 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
            """
            data_transforms = transforms.Compose([transforms.Resize(image_size),
                                                  transforms.ToTensor()])
```

```
    # TODO: Implement function and return a dataloader
    image_data = datasets.ImageFolder(data_dir, transform=data_transforms)
    data_loader = torch.utils.data.DataLoader(image_data, batch_size=batch_size, shuffle

    return data_loader
```

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 = 250
        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)
```

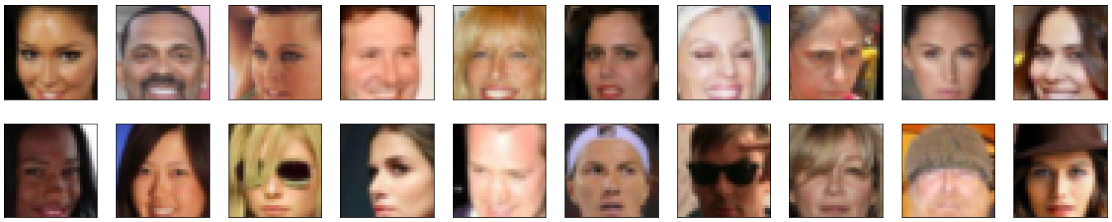
Next, you can view some images! You should seen square images of somewhat-centered faces.

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) - 1

            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(-1.)
Max:  tensor(0.7647)
```

---

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

In [10]: 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

                 #input 32 * 32 * 3, output 16 * 16 * 32
                 self.conv1 = nn.Conv2d(3, conv_dim, 4, stride=2, padding=1, bias=False)

                 #input 16 * 16 * 32, output 8 * 8 * 64
                 self.conv2 = nn.Conv2d(conv_dim, conv_dim * 2, 4, stride=2, padding=1, bias=Fal

                 self.batchnorm1 = nn.BatchNorm2d(conv_dim * 2)

                 #input 8 * 8 * 64, output 4 * 4 * 128
                 self.conv3 = nn.Conv2d(conv_dim * 2, conv_dim * 4, 4, stride=2, padding=1, bias

                 self.batchnorm2 = nn.BatchNorm2d(conv_dim * 4)

                 self.fc = nn.Linear(4 * 4 * self.conv_dim * 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
```

```
                out = F.leaky_relu(self.conv1(x), 0.2)
                out = F.leaky_relu(self.batchnorm1(self.conv2(out)), 0.2)
                out = F.leaky_relu(self.batchnorm2(self.conv3(out)), 0.2)

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

                out = self.fc(out)

                return out


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

```
Tests Passed
```

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 [11]: 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.fc = nn.Linear(z_size, 4 * 4 * self.conv_dim * 4)

                #input 4 * 4 * 128, output 8 * 8 * 64
                self.d_conv1 = nn.ConvTranspose2d(self.conv_dim * 4, self.conv_dim * 2, 4, stri
```

```
            self.batchnorm1 = nn.BatchNorm2d(self.conv_dim * 2)

            #input 8 * 8 * 64, output 16 * 16 * 32
            self.d_conv2 = nn.ConvTranspose2d(self.conv_dim * 2, self.conv_dim, 4, stride=2

            self.batchnorm2 = nn.BatchNorm2d(self.conv_dim)

            #input 16 * 16 * 32, output 32 * 32 * 3
            self.d_conv3 = nn.ConvTranspose2d(self.conv_dim, 3, 4, stride=2, padding=1, bia


        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
            out = self.fc(x)

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

            out = F.relu(self.batchnorm1(self.d_conv1(out)))
            out = F.relu(self.batchnorm2(self.d_conv2(out)))
            out = F.tanh(self.d_conv3(out))

            return out

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

```
Tests Passed
```

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 [12]: 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:
                 m.weight.data.normal_(0.0, std=0.02)
             elif classname.find('Linear') != -1:
                 m.weight.data.normal_(0.0, std=0.02)
                 m.bias.data.fill_(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 [13]: """
         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 [14]: # 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)
  (conv2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (batchnorm1): 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)
  (batchnorm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc): Linear(in_features=2048, out_features=1, bias=True)
)

Generator(
  (fc): Linear(in_features=100, out_features=2048, bias=True)
  (d_conv1): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=Fa
  (batchnorm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (d_conv2): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=Fal
  (batchnorm2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (d_conv3): ConvTranspose2d(32, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=Fals
)
```

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 [15]: """
         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 [20]: def real_loss(D_out):
             '''Calculates how close discriminator outputs are to being real.
                param, D_out: discriminator logits
                return: real loss'''
             batch_size = D_out.size(0)
             labels = torch.ones(batch_size)
             if train_on_gpu:
                 labels = labels.cuda()
             criterion = nn.BCEWithLogitsLoss()
             loss = criterion(D_out.squeeze(), labels)
             return loss

         def fake_loss(D_out):
             '''Calculates how close discriminator outputs are to being fake.
                param, D_out: discriminator logits
                return: fake loss'''
             batch_size = D_out.size(0)
             labels = torch.zeros(batch_size)
             if train_on_gpu:
                 labels = labels.cuda()
             criterion = nn.BCEWithLogitsLoss()
             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 [17]: import torch.optim as optim

         lr = 0.002
         beta1 = 0.5
         beta2 = 0.99

         # Create optimizers for the discriminator D and generator G
         d_optimizer = optim.Adam(D.parameters(), lr, [beta1, beta2])
         g_optimizer = optim.Adam(G.parameters(), lr, [beta1, beta2])
```

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 [18]: 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
        d_optimizer.zero_grad()

        if train_on_gpu:
            real_images = real_images.cuda()

        d_real = D(real_images)
        d_real_loss = real_loss(d_real)

        z = np.random.uniform(-1, 1, size=(batch_size, z_size))
        z = torch.from_numpy(z).float()
        # move x to GPU, if available
        if train_on_gpu:
            z = z.cuda()
        fake_image = G(z)

        d_fake = D(fake_image)
        d_fake_loss = fake_loss(d_fake)


        d_loss = d_real_loss + d_fake_loss

        d_loss.backward()
        d_optimizer.step()

        # 2. Train the generator with an adversarial loss
```

```
            g_optimizer.zero_grad()

            z = np.random.uniform(-1, 1, size=(batch_size, z_size))
            z = torch.from_numpy(z).float()
            # move x to GPU, if available
            if train_on_gpu:
                z = z.cuda()

            fake_image = G(z)

            g_fake = D(fake_image)

            g_loss = real_loss(g_fake)

            g_loss.backward()
            g_optimizer.step()


            # =========================================
            #              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 [21]: # set number of epochs
```

```
n_epochs = 100


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

```
Epoch [    1/  100] | d_loss: 1.3463 | g_loss: 1.6634
Epoch [    1/  100] | d_loss: 1.1512 | g_loss: 2.2143
Epoch [    1/  100] | d_loss: 1.4826 | g_loss: 1.9916
Epoch [    1/  100] | d_loss: 1.2372 | g_loss: 1.8856
Epoch [    1/  100] | d_loss: 1.2837 | g_loss: 1.3957
Epoch [    1/  100] | d_loss: 1.2444 | g_loss: 1.5703
Epoch [    1/  100] | d_loss: 1.3693 | g_loss: 1.7550
Epoch [    1/  100] | d_loss: 1.1780 | g_loss: 1.4302
Epoch [    2/  100] | d_loss: 1.2323 | g_loss: 1.2261
Epoch [    2/  100] | d_loss: 1.3586 | g_loss: 1.2334
Epoch [    2/  100] | d_loss: 1.4067 | g_loss: 1.7315
Epoch [    2/  100] | d_loss: 1.2749 | g_loss: 0.9465
Epoch [    2/  100] | d_loss: 1.2015 | g_loss: 1.1186
Epoch [    2/  100] | d_loss: 1.2783 | g_loss: 1.8469
Epoch [    2/  100] | d_loss: 1.3319 | g_loss: 1.1173
Epoch [    2/  100] | d_loss: 1.3206 | g_loss: 1.4823
Epoch [    3/  100] | d_loss: 1.2134 | g_loss: 1.3471
Epoch [    3/  100] | d_loss: 1.2695 | g_loss: 1.1931
Epoch [    3/  100] | d_loss: 1.2713 | g_loss: 0.9972
Epoch [    3/  100] | d_loss: 1.2386 | g_loss: 1.2547
Epoch [    3/  100] | d_loss: 1.2015 | g_loss: 1.5861
Epoch [    3/  100] | d_loss: 1.1600 | g_loss: 1.4207
Epoch [    3/  100] | d_loss: 1.1455 | g_loss: 1.3772
Epoch [    3/  100] | d_loss: 1.4712 | g_loss: 1.5005
Epoch [    4/  100] | d_loss: 1.1579 | g_loss: 1.0672
Epoch [    4/  100] | d_loss: 1.1568 | g_loss: 1.3745
Epoch [    4/  100] | d_loss: 1.2320 | g_loss: 1.2717
Epoch [    4/  100] | d_loss: 1.4774 | g_loss: 0.6845
Epoch [    4/  100] | d_loss: 1.1516 | g_loss: 1.1869
Epoch [    4/  100] | d_loss: 1.2790 | g_loss: 1.0060
Epoch [    4/  100] | d_loss: 1.2006 | g_loss: 1.3492
Epoch [    4/  100] | d_loss: 1.1729 | g_loss: 1.2906
Epoch [    5/  100] | d_loss: 1.2601 | g_loss: 1.2141
Epoch [    5/  100] | d_loss: 1.2107 | g_loss: 1.0780
Epoch [    5/  100] | d_loss: 1.2101 | g_loss: 1.1711
Epoch [    5/  100] | d_loss: 1.1833 | g_loss: 1.2667
Epoch [    5/  100] | d_loss: 1.2836 | g_loss: 1.5545
Epoch [    5/  100] | d_loss: 1.1991 | g_loss: 1.8337
Epoch [    5/  100] | d_loss: 1.1553 | g_loss: 1.6062
```

```
Epoch [    5/  100] | d_loss: 1.2505 | g_loss: 0.9892
Epoch [    6/  100] | d_loss: 1.1519 | g_loss: 1.3974
Epoch [    6/  100] | d_loss: 1.0695 | g_loss: 1.2369
Epoch [    6/  100] | d_loss: 1.2235 | g_loss: 1.0194
Epoch [    6/  100] | d_loss: 1.3744 | g_loss: 0.9934
Epoch [    6/  100] | d_loss: 1.1579 | g_loss: 1.7717
Epoch [    6/  100] | d_loss: 1.1288 | g_loss: 1.6101
Epoch [    6/  100] | d_loss: 1.3437 | g_loss: 1.7862
Epoch [    6/  100] | d_loss: 1.0562 | g_loss: 1.3636
Epoch [    7/  100] | d_loss: 1.0425 | g_loss: 1.3848
Epoch [    7/  100] | d_loss: 1.3969 | g_loss: 1.7576
Epoch [    7/  100] | d_loss: 1.0371 | g_loss: 1.3147
Epoch [    7/  100] | d_loss: 1.1050 | g_loss: 1.2723
Epoch [    7/  100] | d_loss: 1.1512 | g_loss: 1.2108
Epoch [    7/  100] | d_loss: 1.1291 | g_loss: 1.3868
Epoch [    7/  100] | d_loss: 1.0428 | g_loss: 1.3949
Epoch [    7/  100] | d_loss: 1.2067 | g_loss: 1.7395
Epoch [    8/  100] | d_loss: 1.1141 | g_loss: 1.1857
Epoch [    8/  100] | d_loss: 1.1551 | g_loss: 1.6084
Epoch [    8/  100] | d_loss: 1.1376 | g_loss: 2.3048
Epoch [    8/  100] | d_loss: 1.2574 | g_loss: 0.9202
Epoch [    8/  100] | d_loss: 1.2581 | g_loss: 1.4936
Epoch [    8/  100] | d_loss: 1.1432 | g_loss: 1.2344
Epoch [    8/  100] | d_loss: 1.0610 | g_loss: 1.4439
Epoch [    8/  100] | d_loss: 1.1252 | g_loss: 0.9589
Epoch [    9/  100] | d_loss: 1.1541 | g_loss: 0.9506
Epoch [    9/  100] | d_loss: 1.1406 | g_loss: 1.8924
Epoch [    9/  100] | d_loss: 1.1010 | g_loss: 0.8557
Epoch [    9/  100] | d_loss: 1.2314 | g_loss: 0.7305
Epoch [    9/  100] | d_loss: 1.0703 | g_loss: 1.4250
Epoch [    9/  100] | d_loss: 1.1569 | g_loss: 1.5996
Epoch [    9/  100] | d_loss: 1.0782 | g_loss: 1.3548
Epoch [    9/  100] | d_loss: 1.1871 | g_loss: 1.3220
Epoch [   10/  100] | d_loss: 1.1170 | g_loss: 1.7083
Epoch [   10/  100] | d_loss: 1.1446 | g_loss: 1.2094
Epoch [   10/  100] | d_loss: 1.1463 | g_loss: 1.8383
Epoch [   10/  100] | d_loss: 1.2605 | g_loss: 1.6627
Epoch [   10/  100] | d_loss: 1.0079 | g_loss: 1.5060
Epoch [   10/  100] | d_loss: 1.1192 | g_loss: 1.0493
Epoch [   10/  100] | d_loss: 0.9925 | g_loss: 1.3499
Epoch [   10/  100] | d_loss: 1.0942 | g_loss: 1.8071
Epoch [   11/  100] | d_loss: 1.0889 | g_loss: 1.4417
Epoch [   11/  100] | d_loss: 1.1137 | g_loss: 1.7545
Epoch [   11/  100] | d_loss: 1.0821 | g_loss: 1.4829
Epoch [   11/  100] | d_loss: 1.0456 | g_loss: 1.4446
Epoch [   11/  100] | d_loss: 1.0071 | g_loss: 1.1816
Epoch [   11/  100] | d_loss: 1.2615 | g_loss: 1.3476
Epoch [   11/  100] | d_loss: 0.9719 | g_loss: 1.6100
```

```
Epoch [   11/  100] | d_loss: 1.1252 | g_loss: 0.9926
Epoch [   12/  100] | d_loss: 1.2610 | g_loss: 1.8733
Epoch [   12/  100] | d_loss: 1.1533 | g_loss: 0.7717
Epoch [   12/  100] | d_loss: 1.3748 | g_loss: 2.2976
Epoch [   12/  100] | d_loss: 1.1782 | g_loss: 1.2737
Epoch [   12/  100] | d_loss: 0.9843 | g_loss: 1.5478
Epoch [   12/  100] | d_loss: 1.0073 | g_loss: 1.2313
Epoch [   12/  100] | d_loss: 1.1064 | g_loss: 1.8966
Epoch [   12/  100] | d_loss: 1.0675 | g_loss: 0.7910
Epoch [   13/  100] | d_loss: 0.9005 | g_loss: 1.9038
Epoch [   13/  100] | d_loss: 1.2851 | g_loss: 2.4841
Epoch [   13/  100] | d_loss: 1.1118 | g_loss: 1.6646
Epoch [   13/  100] | d_loss: 1.0768 | g_loss: 1.4325
Epoch [   13/  100] | d_loss: 0.9396 | g_loss: 1.1537
Epoch [   13/  100] | d_loss: 1.1147 | g_loss: 1.7198
Epoch [   13/  100] | d_loss: 0.7940 | g_loss: 1.2206
Epoch [   13/  100] | d_loss: 1.2451 | g_loss: 2.1203
Epoch [   14/  100] | d_loss: 0.9679 | g_loss: 1.4826
Epoch [   14/  100] | d_loss: 1.0167 | g_loss: 1.5571
Epoch [   14/  100] | d_loss: 1.2343 | g_loss: 2.4107
Epoch [   14/  100] | d_loss: 0.8621 | g_loss: 1.5220
Epoch [   14/  100] | d_loss: 0.9033 | g_loss: 2.4969
Epoch [   14/  100] | d_loss: 1.0244 | g_loss: 0.7491
Epoch [   14/  100] | d_loss: 0.8574 | g_loss: 1.6256
Epoch [   14/  100] | d_loss: 0.9622 | g_loss: 1.5697
Epoch [   15/  100] | d_loss: 0.9642 | g_loss: 1.0639
Epoch [   15/  100] | d_loss: 0.9906 | g_loss: 1.6249
Epoch [   15/  100] | d_loss: 0.9844 | g_loss: 2.5407
Epoch [   15/  100] | d_loss: 0.9281 | g_loss: 1.8516
Epoch [   15/  100] | d_loss: 0.7752 | g_loss: 1.7573
Epoch [   15/  100] | d_loss: 0.9765 | g_loss: 2.3117
Epoch [   15/  100] | d_loss: 0.9336 | g_loss: 1.7965
Epoch [   15/  100] | d_loss: 1.0808 | g_loss: 0.8401
Epoch [   16/  100] | d_loss: 0.8258 | g_loss: 1.6794
Epoch [   16/  100] | d_loss: 1.0735 | g_loss: 2.7760
Epoch [   16/  100] | d_loss: 0.7816 | g_loss: 2.3422
Epoch [   16/  100] | d_loss: 0.9078 | g_loss: 1.9730
Epoch [   16/  100] | d_loss: 0.8658 | g_loss: 1.4405
Epoch [   16/  100] | d_loss: 0.9576 | g_loss: 2.3504
Epoch [   16/  100] | d_loss: 1.1987 | g_loss: 2.7686
Epoch [   16/  100] | d_loss: 1.0384 | g_loss: 2.5079
Epoch [   17/  100] | d_loss: 1.0547 | g_loss: 1.1233
Epoch [   17/  100] | d_loss: 0.9044 | g_loss: 1.1286
Epoch [   17/  100] | d_loss: 0.9066 | g_loss: 1.5836
Epoch [   17/  100] | d_loss: 0.8165 | g_loss: 2.2063
Epoch [   17/  100] | d_loss: 1.0081 | g_loss: 1.5929
Epoch [   17/  100] | d_loss: 1.0989 | g_loss: 3.1860
Epoch [   17/  100] | d_loss: 0.8243 | g_loss: 1.5662
```

```
Epoch [   17/  100] | d_loss: 0.7867 | g_loss: 2.1096
Epoch [   18/  100] | d_loss: 1.2541 | g_loss: 3.0121
Epoch [   18/  100] | d_loss: 0.7574 | g_loss: 2.2147
Epoch [   18/  100] | d_loss: 1.0258 | g_loss: 2.1415
Epoch [   18/  100] | d_loss: 0.8574 | g_loss: 1.7716
Epoch [   18/  100] | d_loss: 1.1786 | g_loss: 2.8643
Epoch [   18/  100] | d_loss: 1.1527 | g_loss: 2.5876
Epoch [   18/  100] | d_loss: 0.8247 | g_loss: 1.6493
Epoch [   18/  100] | d_loss: 0.8580 | g_loss: 1.3777
Epoch [   19/  100] | d_loss: 0.7824 | g_loss: 1.5860
Epoch [   19/  100] | d_loss: 0.6774 | g_loss: 1.7150
Epoch [   19/  100] | d_loss: 0.9082 | g_loss: 1.8834
Epoch [   19/  100] | d_loss: 0.7296 | g_loss: 1.7982
Epoch [   19/  100] | d_loss: 0.8819 | g_loss: 2.2884
Epoch [   19/  100] | d_loss: 0.8154 | g_loss: 2.0283
Epoch [   19/  100] | d_loss: 1.1917 | g_loss: 0.9103
Epoch [   19/  100] | d_loss: 0.8007 | g_loss: 1.3795
Epoch [   20/  100] | d_loss: 0.8532 | g_loss: 1.2608
Epoch [   20/  100] | d_loss: 1.0059 | g_loss: 0.7671
Epoch [   20/  100] | d_loss: 0.7510 | g_loss: 2.7857
Epoch [   20/  100] | d_loss: 0.7988 | g_loss: 1.3850
Epoch [   20/  100] | d_loss: 0.8636 | g_loss: 2.0616
Epoch [   20/  100] | d_loss: 0.6995 | g_loss: 1.4391
Epoch [   20/  100] | d_loss: 0.8905 | g_loss: 1.2157
Epoch [   20/  100] | d_loss: 0.7089 | g_loss: 1.9038
Epoch [   21/  100] | d_loss: 0.8409 | g_loss: 1.3225
Epoch [   21/  100] | d_loss: 0.7740 | g_loss: 1.7542
Epoch [   21/  100] | d_loss: 0.6318 | g_loss: 2.1176
Epoch [   21/  100] | d_loss: 0.8037 | g_loss: 2.5423
Epoch [   21/  100] | d_loss: 0.8524 | g_loss: 1.3808
Epoch [   21/  100] | d_loss: 0.9020 | g_loss: 1.2971
Epoch [   21/  100] | d_loss: 1.4200 | g_loss: 0.6804
Epoch [   21/  100] | d_loss: 0.7394 | g_loss: 2.2647
Epoch [   22/  100] | d_loss: 0.9086 | g_loss: 2.5366
Epoch [   22/  100] | d_loss: 0.8511 | g_loss: 1.4684
Epoch [   22/  100] | d_loss: 0.7518 | g_loss: 1.9960
Epoch [   22/  100] | d_loss: 0.7914 | g_loss: 2.1574
Epoch [   22/  100] | d_loss: 0.7690 | g_loss: 2.5378
Epoch [   22/  100] | d_loss: 0.8339 | g_loss: 3.3619
Epoch [   22/  100] | d_loss: 0.8514 | g_loss: 3.0091
Epoch [   22/  100] | d_loss: 0.7558 | g_loss: 3.0384
Epoch [   23/  100] | d_loss: 0.6577 | g_loss: 1.6381
Epoch [   23/  100] | d_loss: 1.1078 | g_loss: 3.0613
Epoch [   24/  100] | d_loss: 0.9759 | g_loss: 1.0888
Epoch [   24/  100] | d_loss: 0.8211 | g_loss: 3.0261
Epoch [   25/  100] | d_loss: 0.7602 | g_loss: 2.0365
Epoch [   25/  100] | d_loss: 0.6162 | g_loss: 1.7457
Epoch [   25/  100] | d_loss: 0.5642 | g_loss: 2.6791
```

```
Epoch [   25/  100] | d_loss: 0.6611 | g_loss: 1.8048
Epoch [   25/  100] | d_loss: 0.6604 | g_loss: 2.0063
Epoch [   25/  100] | d_loss: 0.9489 | g_loss: 3.1161
Epoch [   25/  100] | d_loss: 0.6557 | g_loss: 2.0544
Epoch [   25/  100] | d_loss: 0.6181 | g_loss: 2.5664
Epoch [   26/  100] | d_loss: 0.6111 | g_loss: 3.3296
Epoch [   26/  100] | d_loss: 0.7730 | g_loss: 1.2698
Epoch [   26/  100] | d_loss: 0.5896 | g_loss: 2.9068
Epoch [   26/  100] | d_loss: 0.8551 | g_loss: 2.8068
Epoch [   26/  100] | d_loss: 0.6791 | g_loss: 2.2489
Epoch [   26/  100] | d_loss: 0.6975 | g_loss: 1.5449
Epoch [   26/  100] | d_loss: 0.9712 | g_loss: 2.7984
Epoch [   26/  100] | d_loss: 0.6828 | g_loss: 3.0461
Epoch [   27/  100] | d_loss: 0.6475 | g_loss: 1.8263
Epoch [   27/  100] | d_loss: 1.0800 | g_loss: 3.3213
Epoch [   27/  100] | d_loss: 1.1622 | g_loss: 3.7866
Epoch [   27/  100] | d_loss: 0.5307 | g_loss: 3.3342
Epoch [   27/  100] | d_loss: 1.2773 | g_loss: 3.7205
Epoch [   27/  100] | d_loss: 0.5098 | g_loss: 1.6942
Epoch [   27/  100] | d_loss: 0.6351 | g_loss: 1.8601
Epoch [   27/  100] | d_loss: 0.5854 | g_loss: 2.7818
Epoch [   28/  100] | d_loss: 0.6215 | g_loss: 2.3320
Epoch [   28/  100] | d_loss: 0.5387 | g_loss: 1.8994
Epoch [   28/  100] | d_loss: 0.6721 | g_loss: 1.7518
Epoch [   28/  100] | d_loss: 0.5083 | g_loss: 2.5056
Epoch [   28/  100] | d_loss: 0.3875 | g_loss: 3.6135
Epoch [   28/  100] | d_loss: 0.5948 | g_loss: 2.3809
Epoch [   28/  100] | d_loss: 0.5876 | g_loss: 1.9803
Epoch [   28/  100] | d_loss: 0.5229 | g_loss: 2.2409
Epoch [   29/  100] | d_loss: 0.5996 | g_loss: 1.9493
Epoch [   29/  100] | d_loss: 0.6997 | g_loss: 1.3991
Epoch [   29/  100] | d_loss: 0.6698 | g_loss: 2.0500
Epoch [   29/  100] | d_loss: 0.5684 | g_loss: 2.5680
Epoch [   29/  100] | d_loss: 1.0364 | g_loss: 3.3032
Epoch [   29/  100] | d_loss: 0.5808 | g_loss: 3.0184
Epoch [   29/  100] | d_loss: 0.5681 | g_loss: 1.7958
Epoch [   29/  100] | d_loss: 0.5702 | g_loss: 2.4255
Epoch [   30/  100] | d_loss: 0.4343 | g_loss: 2.0392
Epoch [   30/  100] | d_loss: 0.6200 | g_loss: 2.1745
Epoch [   30/  100] | d_loss: 0.5979 | g_loss: 1.7929
Epoch [   30/  100] | d_loss: 0.6096 | g_loss: 2.6489
Epoch [   30/  100] | d_loss: 1.4366 | g_loss: 3.7204
Epoch [   30/  100] | d_loss: 0.4237 | g_loss: 2.4681
Epoch [   30/  100] | d_loss: 0.9141 | g_loss: 1.2616
Epoch [   30/  100] | d_loss: 0.6541 | g_loss: 1.4750
Epoch [   31/  100] | d_loss: 0.5565 | g_loss: 1.9269
Epoch [   31/  100] | d_loss: 0.6057 | g_loss: 1.9882
Epoch [   31/  100] | d_loss: 1.2230 | g_loss: 1.8084
```

```
Epoch [   31/  100] | d_loss: 0.5311 | g_loss: 2.8243
Epoch [   31/  100] | d_loss: 0.7885 | g_loss: 2.2921
Epoch [   31/  100] | d_loss: 0.4408 | g_loss: 2.5822
Epoch [   31/  100] | d_loss: 0.3437 | g_loss: 2.9612
Epoch [   31/  100] | d_loss: 0.8231 | g_loss: 2.0779
Epoch [   32/  100] | d_loss: 0.5180 | g_loss: 2.0594
Epoch [   32/  100] | d_loss: 0.4921 | g_loss: 2.2184
Epoch [   32/  100] | d_loss: 0.5118 | g_loss: 1.7169
Epoch [   32/  100] | d_loss: 0.4384 | g_loss: 2.4953
Epoch [   32/  100] | d_loss: 0.5102 | g_loss: 2.3845
Epoch [   32/  100] | d_loss: 0.6575 | g_loss: 2.1702
Epoch [   32/  100] | d_loss: 0.4594 | g_loss: 2.9328
Epoch [   32/  100] | d_loss: 0.4929 | g_loss: 2.9381
Epoch [   33/  100] | d_loss: 1.5056 | g_loss: 4.8884
Epoch [   33/  100] | d_loss: 0.5745 | g_loss: 1.9888
Epoch [   33/  100] | d_loss: 0.8352 | g_loss: 1.8097
Epoch [   33/  100] | d_loss: 0.5602 | g_loss: 3.1447
Epoch [   33/  100] | d_loss: 0.6755 | g_loss: 1.8919
Epoch [   33/  100] | d_loss: 0.4045 | g_loss: 2.1681
Epoch [   33/  100] | d_loss: 0.6498 | g_loss: 3.5502
Epoch [   33/  100] | d_loss: 0.5623 | g_loss: 3.0902
Epoch [   34/  100] | d_loss: 0.6842 | g_loss: 3.7292
Epoch [   34/  100] | d_loss: 1.5921 | g_loss: 0.2115
Epoch [   34/  100] | d_loss: 0.5702 | g_loss: 1.7903
Epoch [   34/  100] | d_loss: 0.6746 | g_loss: 3.4043
Epoch [   34/  100] | d_loss: 1.6729 | g_loss: 0.5088
Epoch [   34/  100] | d_loss: 0.4591 | g_loss: 2.5124
Epoch [   34/  100] | d_loss: 0.5086 | g_loss: 1.3436
Epoch [   34/  100] | d_loss: 0.5797 | g_loss: 3.5700
Epoch [   35/  100] | d_loss: 0.4466 | g_loss: 2.7006
Epoch [   35/  100] | d_loss: 0.5964 | g_loss: 2.5475
Epoch [   35/  100] | d_loss: 0.6450 | g_loss: 2.3249
Epoch [   35/  100] | d_loss: 0.5139 | g_loss: 2.6193
Epoch [   35/  100] | d_loss: 0.8060 | g_loss: 3.8520
Epoch [   35/  100] | d_loss: 0.3378 | g_loss: 3.2881
Epoch [   35/  100] | d_loss: 0.6265 | g_loss: 1.8577
Epoch [   35/  100] | d_loss: 0.3890 | g_loss: 2.3443
Epoch [   36/  100] | d_loss: 0.6307 | g_loss: 2.6396
Epoch [   36/  100] | d_loss: 0.2887 | g_loss: 2.6886
Epoch [   36/  100] | d_loss: 0.4120 | g_loss: 3.1350
Epoch [   36/  100] | d_loss: 0.7488 | g_loss: 1.4611
Epoch [   36/  100] | d_loss: 0.3889 | g_loss: 2.2324
Epoch [   36/  100] | d_loss: 0.3520 | g_loss: 2.7685
Epoch [   36/  100] | d_loss: 2.9291 | g_loss: 0.3941
Epoch [   36/  100] | d_loss: 0.4505 | g_loss: 2.4305
Epoch [   37/  100] | d_loss: 0.5711 | g_loss: 2.6194
Epoch [   37/  100] | d_loss: 0.4093 | g_loss: 2.7405
Epoch [   37/  100] | d_loss: 0.6553 | g_loss: 3.0579
```

```
Epoch [   37/  100] | d_loss: 0.5199 | g_loss: 4.1506
Epoch [   37/  100] | d_loss: 0.4049 | g_loss: 3.1049
Epoch [   37/  100] | d_loss: 0.4575 | g_loss: 3.1757
Epoch [   37/  100] | d_loss: 0.5890 | g_loss: 1.5932
Epoch [   37/  100] | d_loss: 0.4314 | g_loss: 2.1532
Epoch [   38/  100] | d_loss: 0.4224 | g_loss: 2.8294
Epoch [   38/  100] | d_loss: 0.4308 | g_loss: 1.7658
Epoch [   38/  100] | d_loss: 0.5625 | g_loss: 2.4040
Epoch [   38/  100] | d_loss: 0.4914 | g_loss: 2.7880
Epoch [   38/  100] | d_loss: 0.5489 | g_loss: 3.5586
Epoch [   38/  100] | d_loss: 0.4733 | g_loss: 2.6055
Epoch [   38/  100] | d_loss: 0.5929 | g_loss: 2.1889
Epoch [   38/  100] | d_loss: 0.2778 | g_loss: 2.5833
Epoch [   39/  100] | d_loss: 0.5346 | g_loss: 1.6246
Epoch [   39/  100] | d_loss: 0.6039 | g_loss: 2.0710
Epoch [   39/  100] | d_loss: 0.3759 | g_loss: 2.5267
Epoch [   39/  100] | d_loss: 1.1509 | g_loss: 0.3717
Epoch [   39/  100] | d_loss: 0.4416 | g_loss: 3.2651
Epoch [   39/  100] | d_loss: 0.6168 | g_loss: 2.7128
Epoch [   39/  100] | d_loss: 0.4683 | g_loss: 1.8364
Epoch [   39/  100] | d_loss: 0.4775 | g_loss: 1.8699
Epoch [   40/  100] | d_loss: 0.3603 | g_loss: 2.7327
Epoch [   40/  100] | d_loss: 0.3798 | g_loss: 2.7021
Epoch [   40/  100] | d_loss: 0.5484 | g_loss: 2.0367
Epoch [   40/  100] | d_loss: 0.4522 | g_loss: 2.2309
Epoch [   40/  100] | d_loss: 0.4111 | g_loss: 2.4755
Epoch [   40/  100] | d_loss: 0.5655 | g_loss: 3.0069
Epoch [   40/  100] | d_loss: 1.0718 | g_loss: 1.6773
Epoch [   40/  100] | d_loss: 0.4468 | g_loss: 2.9412
Epoch [   41/  100] | d_loss: 0.3715 | g_loss: 2.8342
Epoch [   41/  100] | d_loss: 0.6220 | g_loss: 2.0734
Epoch [   41/  100] | d_loss: 0.4136 | g_loss: 3.5442
Epoch [   41/  100] | d_loss: 0.5151 | g_loss: 2.2643
Epoch [   41/  100] | d_loss: 0.4002 | g_loss: 3.2514
Epoch [   41/  100] | d_loss: 0.3903 | g_loss: 1.8688
Epoch [   41/  100] | d_loss: 0.7106 | g_loss: 2.4779
Epoch [   41/  100] | d_loss: 0.6286 | g_loss: 3.6170
Epoch [   42/  100] | d_loss: 0.3378 | g_loss: 2.7065
Epoch [   42/  100] | d_loss: 0.7425 | g_loss: 4.6189
Epoch [   42/  100] | d_loss: 0.3912 | g_loss: 3.0257
Epoch [   42/  100] | d_loss: 0.3995 | g_loss: 3.0272
Epoch [   42/  100] | d_loss: 1.0240 | g_loss: 1.4317
Epoch [   42/  100] | d_loss: 0.4668 | g_loss: 2.0681
Epoch [   42/  100] | d_loss: 1.0254 | g_loss: 1.1544
Epoch [   42/  100] | d_loss: 0.4563 | g_loss: 2.3131
Epoch [   43/  100] | d_loss: 0.4637 | g_loss: 2.4836
Epoch [   43/  100] | d_loss: 0.9003 | g_loss: 6.3969
Epoch [   43/  100] | d_loss: 0.3508 | g_loss: 2.6288
```

```
Epoch [   43/  100] | d_loss: 0.4506 | g_loss: 3.2600
Epoch [   43/  100] | d_loss: 0.3480 | g_loss: 3.1622
Epoch [   43/  100] | d_loss: 0.2942 | g_loss: 3.0141
Epoch [   43/  100] | d_loss: 0.4093 | g_loss: 2.7065
Epoch [   43/  100] | d_loss: 1.1016 | g_loss: 6.7204
Epoch [   44/  100] | d_loss: 1.4669 | g_loss: 2.7338
Epoch [   44/  100] | d_loss: 0.5506 | g_loss: 2.4791
Epoch [   44/  100] | d_loss: 0.4354 | g_loss: 3.7846
Epoch [   44/  100] | d_loss: 0.4020 | g_loss: 2.3912
Epoch [   44/  100] | d_loss: 0.6646 | g_loss: 1.8423
Epoch [   44/  100] | d_loss: 0.4974 | g_loss: 3.6553
Epoch [   44/  100] | d_loss: 0.4848 | g_loss: 2.8643
Epoch [   44/  100] | d_loss: 0.3263 | g_loss: 3.9130
Epoch [   45/  100] | d_loss: 0.4191 | g_loss: 3.3431
Epoch [   45/  100] | d_loss: 0.4561 | g_loss: 3.3451
Epoch [   45/  100] | d_loss: 0.4152 | g_loss: 2.9621
Epoch [   45/  100] | d_loss: 0.5466 | g_loss: 1.6248
Epoch [   45/  100] | d_loss: 0.3563 | g_loss: 3.5099
Epoch [   45/  100] | d_loss: 0.5681 | g_loss: 2.1892
Epoch [   45/  100] | d_loss: 0.2963 | g_loss: 4.1916
Epoch [   45/  100] | d_loss: 0.5164 | g_loss: 1.2436
Epoch [   46/  100] | d_loss: 0.3226 | g_loss: 3.0903
Epoch [   46/  100] | d_loss: 0.5856 | g_loss: 4.7733
Epoch [   46/  100] | d_loss: 0.3139 | g_loss: 2.2108
Epoch [   46/  100] | d_loss: 0.3112 | g_loss: 3.8652
Epoch [   46/  100] | d_loss: 0.6123 | g_loss: 1.2414
Epoch [   46/  100] | d_loss: 0.3812 | g_loss: 3.6960
Epoch [   46/  100] | d_loss: 0.5322 | g_loss: 2.9843
Epoch [   46/  100] | d_loss: 0.3270 | g_loss: 3.4577
Epoch [   47/  100] | d_loss: 0.5216 | g_loss: 2.5954
Epoch [   47/  100] | d_loss: 0.4991 | g_loss: 4.3134
Epoch [   47/  100] | d_loss: 0.5711 | g_loss: 4.7258
Epoch [   47/  100] | d_loss: 0.5605 | g_loss: 2.8129
Epoch [   47/  100] | d_loss: 0.6285 | g_loss: 1.6258
Epoch [   47/  100] | d_loss: 0.3195 | g_loss: 3.4208
Epoch [   47/  100] | d_loss: 0.3270 | g_loss: 3.0008
Epoch [   47/  100] | d_loss: 0.3254 | g_loss: 2.4742
Epoch [   48/  100] | d_loss: 0.3300 | g_loss: 3.0817
Epoch [   48/  100] | d_loss: 0.7322 | g_loss: 4.6882
Epoch [   48/  100] | d_loss: 0.5228 | g_loss: 3.6498
Epoch [   48/  100] | d_loss: 0.2210 | g_loss: 3.4480
Epoch [   48/  100] | d_loss: 0.3236 | g_loss: 3.1288
Epoch [   48/  100] | d_loss: 0.4231 | g_loss: 3.4813
Epoch [   48/  100] | d_loss: 0.4962 | g_loss: 3.0190
Epoch [   48/  100] | d_loss: 0.5535 | g_loss: 2.5004
Epoch [   49/  100] | d_loss: 0.4559 | g_loss: 2.5484
Epoch [   49/  100] | d_loss: 0.2670 | g_loss: 2.9976
Epoch [   49/  100] | d_loss: 0.5178 | g_loss: 1.7294
```

```
Epoch [   49/  100] | d_loss: 0.5815 | g_loss: 3.4366
Epoch [   49/  100] | d_loss: 0.5732 | g_loss: 2.6162
Epoch [   49/  100] | d_loss: 0.4537 | g_loss: 2.9586
Epoch [   49/  100] | d_loss: 0.4058 | g_loss: 4.1578
Epoch [   49/  100] | d_loss: 0.2592 | g_loss: 3.5598
Epoch [   50/  100] | d_loss: 1.2199 | g_loss: 4.8777
Epoch [   50/  100] | d_loss: 0.5346 | g_loss: 3.1878
Epoch [   50/  100] | d_loss: 0.4959 | g_loss: 3.4814
Epoch [   50/  100] | d_loss: 0.5077 | g_loss: 3.3122
Epoch [   50/  100] | d_loss: 0.5145 | g_loss: 4.1317
Epoch [   50/  100] | d_loss: 0.7051 | g_loss: 1.3441
Epoch [   50/  100] | d_loss: 0.3046 | g_loss: 3.3507
Epoch [   50/  100] | d_loss: 0.4842 | g_loss: 2.4238
Epoch [   51/  100] | d_loss: 0.4298 | g_loss: 4.0537
Epoch [   51/  100] | d_loss: 0.7676 | g_loss: 1.9265
Epoch [   51/  100] | d_loss: 0.3146 | g_loss: 3.7383
Epoch [   51/  100] | d_loss: 0.2760 | g_loss: 3.4987
Epoch [   51/  100] | d_loss: 0.2619 | g_loss: 2.9446
Epoch [   51/  100] | d_loss: 0.5951 | g_loss: 3.5504
Epoch [   51/  100] | d_loss: 0.6279 | g_loss: 2.3238
Epoch [   51/  100] | d_loss: 0.3707 | g_loss: 3.3334
Epoch [   52/  100] | d_loss: 0.2934 | g_loss: 3.5700
Epoch [   52/  100] | d_loss: 1.1911 | g_loss: 6.3821
Epoch [   52/  100] | d_loss: 0.3875 | g_loss: 2.9417
Epoch [   52/  100] | d_loss: 0.2651 | g_loss: 4.3812
Epoch [   52/  100] | d_loss: 0.5362 | g_loss: 3.6999
Epoch [   52/  100] | d_loss: 0.2624 | g_loss: 3.8989
Epoch [   52/  100] | d_loss: 0.3066 | g_loss: 2.8618
Epoch [   52/  100] | d_loss: 0.6815 | g_loss: 2.0460
Epoch [   53/  100] | d_loss: 0.4938 | g_loss: 3.2992
Epoch [   53/  100] | d_loss: 0.5295 | g_loss: 4.6129
Epoch [   53/  100] | d_loss: 5.7868 | g_loss: 0.2134
Epoch [   53/  100] | d_loss: 0.5170 | g_loss: 2.1399
Epoch [   53/  100] | d_loss: 0.6377 | g_loss: 1.4478
Epoch [   53/  100] | d_loss: 0.3483 | g_loss: 2.3446
Epoch [   53/  100] | d_loss: 0.3331 | g_loss: 3.5436
Epoch [   53/  100] | d_loss: 0.5230 | g_loss: 3.5544
Epoch [   54/  100] | d_loss: 0.3812 | g_loss: 2.0564
Epoch [   54/  100] | d_loss: 0.2895 | g_loss: 2.7092
Epoch [   54/  100] | d_loss: 2.3194 | g_loss: 7.2933
Epoch [   54/  100] | d_loss: 0.3293 | g_loss: 3.4685
Epoch [   54/  100] | d_loss: 0.3833 | g_loss: 3.5774
Epoch [   54/  100] | d_loss: 0.2730 | g_loss: 3.5905
Epoch [   54/  100] | d_loss: 0.2529 | g_loss: 3.0277
Epoch [   54/  100] | d_loss: 1.2541 | g_loss: 0.8886
Epoch [   55/  100] | d_loss: 0.4000 | g_loss: 2.6284
Epoch [   55/  100] | d_loss: 0.4662 | g_loss: 2.4988
Epoch [   55/  100] | d_loss: 0.5534 | g_loss: 3.5979
```

```
Epoch [   55/  100] | d_loss: 0.3158 | g_loss: 2.7663
Epoch [   55/  100] | d_loss: 0.3595 | g_loss: 3.7289
Epoch [   55/  100] | d_loss: 0.4380 | g_loss: 1.9298
Epoch [   55/  100] | d_loss: 0.2514 | g_loss: 3.5236
Epoch [   55/  100] | d_loss: 0.3568 | g_loss: 3.8662
Epoch [   56/  100] | d_loss: 0.2991 | g_loss: 3.4006
Epoch [   56/  100] | d_loss: 0.3932 | g_loss: 3.0118
Epoch [   56/  100] | d_loss: 0.8674 | g_loss: 5.1609
Epoch [   56/  100] | d_loss: 0.3258 | g_loss: 4.9003
Epoch [   56/  100] | d_loss: 0.4137 | g_loss: 1.6678
Epoch [   56/  100] | d_loss: 0.3342 | g_loss: 3.0576
Epoch [   56/  100] | d_loss: 0.3747 | g_loss: 1.9991
Epoch [   56/  100] | d_loss: 0.6877 | g_loss: 3.5992
Epoch [   57/  100] | d_loss: 0.5730 | g_loss: 4.1813
Epoch [   57/  100] | d_loss: 0.3758 | g_loss: 3.1505
Epoch [   57/  100] | d_loss: 0.3469 | g_loss: 2.4177
Epoch [   57/  100] | d_loss: 0.2808 | g_loss: 3.1055
Epoch [   57/  100] | d_loss: 0.3621 | g_loss: 2.6262
Epoch [   57/  100] | d_loss: 0.2515 | g_loss: 4.1284
Epoch [   57/  100] | d_loss: 0.4006 | g_loss: 2.8214
Epoch [   57/  100] | d_loss: 0.3980 | g_loss: 4.2547
Epoch [   58/  100] | d_loss: 0.3216 | g_loss: 3.5403
Epoch [   58/  100] | d_loss: 0.2975 | g_loss: 3.1200
Epoch [   58/  100] | d_loss: 0.4508 | g_loss: 2.8874
Epoch [   58/  100] | d_loss: 0.1592 | g_loss: 4.0312
Epoch [   58/  100] | d_loss: 0.8296 | g_loss: 3.0719
Epoch [   58/  100] | d_loss: 0.2897 | g_loss: 3.1062
Epoch [   58/  100] | d_loss: 0.4543 | g_loss: 5.1245
Epoch [   58/  100] | d_loss: 0.2615 | g_loss: 2.9711
Epoch [   59/  100] | d_loss: 0.3505 | g_loss: 3.7642
Epoch [   59/  100] | d_loss: 0.1536 | g_loss: 3.3853
Epoch [   59/  100] | d_loss: 0.3930 | g_loss: 1.8324
Epoch [   59/  100] | d_loss: 0.3375 | g_loss: 2.2846
Epoch [   59/  100] | d_loss: 0.7101 | g_loss: 4.9777
Epoch [   59/  100] | d_loss: 0.4004 | g_loss: 4.5589
Epoch [   59/  100] | d_loss: 0.2824 | g_loss: 4.2661
Epoch [   59/  100] | d_loss: 1.5462 | g_loss: 7.5970
Epoch [   60/  100] | d_loss: 0.7643 | g_loss: 1.5272
Epoch [   60/  100] | d_loss: 0.3739 | g_loss: 2.4942
Epoch [   60/  100] | d_loss: 0.2787 | g_loss: 3.0744
Epoch [   60/  100] | d_loss: 0.2047 | g_loss: 3.4298
Epoch [   60/  100] | d_loss: 0.5056 | g_loss: 3.4717
Epoch [   60/  100] | d_loss: 0.2570 | g_loss: 3.2066
Epoch [   60/  100] | d_loss: 0.2546 | g_loss: 3.3939
Epoch [   60/  100] | d_loss: 0.3334 | g_loss: 3.3866
Epoch [   61/  100] | d_loss: 0.3421 | g_loss: 3.4512
Epoch [   61/  100] | d_loss: 0.5209 | g_loss: 5.1146
Epoch [   61/  100] | d_loss: 1.8500 | g_loss: 0.8296
```

```
Epoch [   61/  100] | d_loss: 0.3556 | g_loss: 4.0053
Epoch [   61/  100] | d_loss: 0.1966 | g_loss: 3.9478
Epoch [   61/  100] | d_loss: 0.3198 | g_loss: 2.7479
Epoch [   61/  100] | d_loss: 0.2838 | g_loss: 3.7822
Epoch [   61/  100] | d_loss: 0.4181 | g_loss: 5.7134
Epoch [   62/  100] | d_loss: 0.2680 | g_loss: 4.4645
Epoch [   62/  100] | d_loss: 0.2184 | g_loss: 3.6046
Epoch [   62/  100] | d_loss: 0.3641 | g_loss: 4.2819
Epoch [   62/  100] | d_loss: 0.3552 | g_loss: 3.7460
Epoch [   62/  100] | d_loss: 0.1959 | g_loss: 3.3642
Epoch [   62/  100] | d_loss: 0.3966 | g_loss: 2.3995
Epoch [   62/  100] | d_loss: 0.4494 | g_loss: 3.2528
Epoch [   62/  100] | d_loss: 0.3090 | g_loss: 3.5403
Epoch [   63/  100] | d_loss: 0.4750 | g_loss: 4.9447
Epoch [   63/  100] | d_loss: 0.2803 | g_loss: 3.1912
Epoch [   63/  100] | d_loss: 0.1857 | g_loss: 4.4522
Epoch [   63/  100] | d_loss: 0.2244 | g_loss: 3.5815
Epoch [   63/  100] | d_loss: 0.4584 | g_loss: 4.1925
Epoch [   63/  100] | d_loss: 0.4761 | g_loss: 1.8449
Epoch [   63/  100] | d_loss: 0.2050 | g_loss: 2.7203
Epoch [   63/  100] | d_loss: 2.7570 | g_loss: 6.2705
Epoch [   64/  100] | d_loss: 0.4219 | g_loss: 3.0702
Epoch [   64/  100] | d_loss: 0.2825 | g_loss: 3.0643
Epoch [   64/  100] | d_loss: 0.2213 | g_loss: 4.6294
Epoch [   64/  100] | d_loss: 0.2615 | g_loss: 3.6094
Epoch [   64/  100] | d_loss: 0.2253 | g_loss: 2.4502
Epoch [   64/  100] | d_loss: 0.5622 | g_loss: 4.6923
Epoch [   64/  100] | d_loss: 0.2216 | g_loss: 3.5034
Epoch [   64/  100] | d_loss: 0.2867 | g_loss: 3.0465
Epoch [   65/  100] | d_loss: 0.1793 | g_loss: 4.8114
Epoch [   65/  100] | d_loss: 0.8908 | g_loss: 1.8066
Epoch [   65/  100] | d_loss: 0.3419 | g_loss: 2.6825
Epoch [   65/  100] | d_loss: 0.1864 | g_loss: 3.5497
Epoch [   65/  100] | d_loss: 0.3565 | g_loss: 3.3184
Epoch [   65/  100] | d_loss: 0.2981 | g_loss: 2.8204
Epoch [   65/  100] | d_loss: 0.3025 | g_loss: 3.9729
Epoch [   65/  100] | d_loss: 0.3843 | g_loss: 4.2373
Epoch [   66/  100] | d_loss: 0.2431 | g_loss: 4.6669
Epoch [   66/  100] | d_loss: 0.3950 | g_loss: 4.2827
Epoch [   66/  100] | d_loss: 0.3376 | g_loss: 3.2716
Epoch [   66/  100] | d_loss: 4.3476 | g_loss: 5.6472
Epoch [   66/  100] | d_loss: 0.3812 | g_loss: 4.1855
Epoch [   66/  100] | d_loss: 0.7408 | g_loss: 5.3606
Epoch [   66/  100] | d_loss: 1.1536 | g_loss: 2.2095
Epoch [   66/  100] | d_loss: 0.2751 | g_loss: 3.4255
Epoch [   67/  100] | d_loss: 0.1693 | g_loss: 4.0131
Epoch [   67/  100] | d_loss: 0.4122 | g_loss: 3.6398
Epoch [   67/  100] | d_loss: 0.1977 | g_loss: 2.9714
```

```
Epoch [   67/  100] | d_loss: 0.5346 | g_loss: 5.1485
Epoch [   67/  100] | d_loss: 0.3984 | g_loss: 2.9659
Epoch [   67/  100] | d_loss: 0.3442 | g_loss: 3.9654
Epoch [   67/  100] | d_loss: 0.2456 | g_loss: 4.2413
Epoch [   67/  100] | d_loss: 0.1934 | g_loss: 4.3529
Epoch [   68/  100] | d_loss: 0.6219 | g_loss: 3.9163
Epoch [   68/  100] | d_loss: 0.3165 | g_loss: 4.1121
Epoch [   68/  100] | d_loss: 0.2880 | g_loss: 4.2079
Epoch [   68/  100] | d_loss: 0.3551 | g_loss: 3.0486
Epoch [   68/  100] | d_loss: 0.3882 | g_loss: 4.3513
Epoch [   68/  100] | d_loss: 0.2219 | g_loss: 4.3765
Epoch [   68/  100] | d_loss: 0.3325 | g_loss: 4.0926
Epoch [   68/  100] | d_loss: 0.2240 | g_loss: 4.0944
Epoch [   69/  100] | d_loss: 0.3651 | g_loss: 4.8154
Epoch [   69/  100] | d_loss: 0.1997 | g_loss: 4.0754
Epoch [   69/  100] | d_loss: 0.3236 | g_loss: 3.0363
Epoch [   69/  100] | d_loss: 0.3793 | g_loss: 2.5287
Epoch [   69/  100] | d_loss: 0.2409 | g_loss: 5.0140
Epoch [   69/  100] | d_loss: 0.4581 | g_loss: 5.4271
Epoch [   69/  100] | d_loss: 0.4977 | g_loss: 3.1136
Epoch [   69/  100] | d_loss: 0.2242 | g_loss: 3.4017
Epoch [   70/  100] | d_loss: 0.2665 | g_loss: 3.6591
Epoch [   70/  100] | d_loss: 0.4624 | g_loss: 5.9595
Epoch [   70/  100] | d_loss: 0.2273 | g_loss: 5.1778
Epoch [   70/  100] | d_loss: 0.6153 | g_loss: 2.9398
Epoch [   70/  100] | d_loss: 0.2161 | g_loss: 3.2183
Epoch [   70/  100] | d_loss: 0.3939 | g_loss: 2.9871
Epoch [   70/  100] | d_loss: 0.2642 | g_loss: 3.6311
Epoch [   70/  100] | d_loss: 0.2201 | g_loss: 2.3645
Epoch [   71/  100] | d_loss: 0.2073 | g_loss: 3.6989
Epoch [   71/  100] | d_loss: 0.3771 | g_loss: 1.5940
Epoch [   71/  100] | d_loss: 0.2420 | g_loss: 3.9911
Epoch [   71/  100] | d_loss: 0.1645 | g_loss: 4.5329
Epoch [   71/  100] | d_loss: 0.3868 | g_loss: 6.2015
Epoch [   71/  100] | d_loss: 0.2732 | g_loss: 4.3122
Epoch [   71/  100] | d_loss: 0.4090 | g_loss: 1.9443
Epoch [   71/  100] | d_loss: 0.1929 | g_loss: 3.7187
Epoch [   72/  100] | d_loss: 0.2139 | g_loss: 4.6632
Epoch [   72/  100] | d_loss: 0.4920 | g_loss: 3.0722
Epoch [   72/  100] | d_loss: 0.3793 | g_loss: 3.1116
Epoch [   72/  100] | d_loss: 0.4977 | g_loss: 1.8532
Epoch [   72/  100] | d_loss: 0.2807 | g_loss: 3.7154
Epoch [   72/  100] | d_loss: 0.2002 | g_loss: 4.9606
Epoch [   72/  100] | d_loss: 0.1546 | g_loss: 4.1851
Epoch [   72/  100] | d_loss: 0.4561 | g_loss: 4.9607
Epoch [   73/  100] | d_loss: 0.6156 | g_loss: 1.6505
Epoch [   73/  100] | d_loss: 0.2479 | g_loss: 3.1470
Epoch [   73/  100] | d_loss: 0.3055 | g_loss: 3.9191
```

```
Epoch [   73/  100] | d_loss: 0.1956 | g_loss: 2.8829
Epoch [   73/  100] | d_loss: 0.2115 | g_loss: 4.0036
Epoch [   73/  100] | d_loss: 0.2556 | g_loss: 3.7014
Epoch [   73/  100] | d_loss: 0.1799 | g_loss: 4.8288
Epoch [   73/  100] | d_loss: 0.5238 | g_loss: 4.7527
Epoch [   74/  100] | d_loss: 0.5490 | g_loss: 4.4224
Epoch [   74/  100] | d_loss: 0.1025 | g_loss: 4.5674
Epoch [   74/  100] | d_loss: 0.3593 | g_loss: 1.9670
Epoch [   74/  100] | d_loss: 1.2346 | g_loss: 2.9036
Epoch [   74/  100] | d_loss: 0.2076 | g_loss: 4.0436
Epoch [   74/  100] | d_loss: 0.2137 | g_loss: 3.6466
Epoch [   74/  100] | d_loss: 0.2485 | g_loss: 3.3115
Epoch [   74/  100] | d_loss: 0.1540 | g_loss: 3.6811
Epoch [   75/  100] | d_loss: 0.1740 | g_loss: 3.9411
Epoch [   75/  100] | d_loss: 0.1776 | g_loss: 3.1542
Epoch [   75/  100] | d_loss: 0.2386 | g_loss: 3.2417
Epoch [   75/  100] | d_loss: 0.1918 | g_loss: 3.1367
Epoch [   75/  100] | d_loss: 1.4765 | g_loss: 2.2190
Epoch [   75/  100] | d_loss: 0.2291 | g_loss: 3.6601
Epoch [   75/  100] | d_loss: 0.1504 | g_loss: 3.0366
Epoch [   75/  100] | d_loss: 0.2853 | g_loss: 4.5325
Epoch [   76/  100] | d_loss: 0.3877 | g_loss: 4.4774
Epoch [   76/  100] | d_loss: 0.2662 | g_loss: 4.2749
Epoch [   76/  100] | d_loss: 0.1261 | g_loss: 4.4301
Epoch [   76/  100] | d_loss: 0.2735 | g_loss: 4.6280
Epoch [   76/  100] | d_loss: 0.3098 | g_loss: 5.2678
Epoch [   76/  100] | d_loss: 0.5413 | g_loss: 3.9988
Epoch [   76/  100] | d_loss: 0.2418 | g_loss: 4.1321
Epoch [   76/  100] | d_loss: 0.1996 | g_loss: 4.3794
Epoch [   77/  100] | d_loss: 0.6680 | g_loss: 6.7504
Epoch [   77/  100] | d_loss: 0.3560 | g_loss: 2.9001
Epoch [   77/  100] | d_loss: 0.1524 | g_loss: 4.6202
Epoch [   77/  100] | d_loss: 0.0975 | g_loss: 3.8968
Epoch [   77/  100] | d_loss: 0.4294 | g_loss: 2.1608
Epoch [   77/  100] | d_loss: 0.4125 | g_loss: 2.2632
Epoch [   77/  100] | d_loss: 0.2138 | g_loss: 3.5654
Epoch [   77/  100] | d_loss: 0.2403 | g_loss: 3.9033
Epoch [   78/  100] | d_loss: 0.1899 | g_loss: 4.6374
Epoch [   78/  100] | d_loss: 0.2460 | g_loss: 3.4026
Epoch [   78/  100] | d_loss: 0.5880 | g_loss: 0.9231
Epoch [   78/  100] | d_loss: 0.2667 | g_loss: 3.5015
Epoch [   78/  100] | d_loss: 0.2423 | g_loss: 4.7476
Epoch [   78/  100] | d_loss: 0.1817 | g_loss: 4.6032
Epoch [   78/  100] | d_loss: 0.2395 | g_loss: 4.0621
Epoch [   78/  100] | d_loss: 0.1892 | g_loss: 3.5651
Epoch [   79/  100] | d_loss: 0.1868 | g_loss: 4.6093
Epoch [   79/  100] | d_loss: 0.1299 | g_loss: 4.6902
Epoch [   79/  100] | d_loss: 0.1391 | g_loss: 3.9885
```

```
Epoch [   79/  100] | d_loss: 0.2781 | g_loss: 3.6085
Epoch [   79/  100] | d_loss: 0.1988 | g_loss: 3.7422
Epoch [   79/  100] | d_loss: 0.2270 | g_loss: 3.0279
Epoch [   79/  100] | d_loss: 0.2747 | g_loss: 3.1942
Epoch [   79/  100] | d_loss: 0.2323 | g_loss: 4.0217
Epoch [   80/  100] | d_loss: 0.4828 | g_loss: 2.3829
Epoch [   80/  100] | d_loss: 1.9782 | g_loss: 0.3932
Epoch [   80/  100] | d_loss: 0.2143 | g_loss: 4.1448
Epoch [   80/  100] | d_loss: 0.1750 | g_loss: 4.7500
Epoch [   80/  100] | d_loss: 0.3161 | g_loss: 3.6624
Epoch [   80/  100] | d_loss: 0.2116 | g_loss: 3.4116
Epoch [   80/  100] | d_loss: 0.7588 | g_loss: 6.6919
Epoch [   80/  100] | d_loss: 0.6132 | g_loss: 5.3639
Epoch [   81/  100] | d_loss: 0.3828 | g_loss: 2.9728
Epoch [   81/  100] | d_loss: 0.1248 | g_loss: 3.9062
Epoch [   81/  100] | d_loss: 0.1687 | g_loss: 4.7522
Epoch [   81/  100] | d_loss: 0.2383 | g_loss: 4.6725
Epoch [   81/  100] | d_loss: 0.2109 | g_loss: 3.3142
Epoch [   81/  100] | d_loss: 0.3195 | g_loss: 4.7139
Epoch [   81/  100] | d_loss: 0.1428 | g_loss: 3.8695
Epoch [   81/  100] | d_loss: 1.4313 | g_loss: 0.2902
Epoch [   82/  100] | d_loss: 0.3565 | g_loss: 2.9765
Epoch [   82/  100] | d_loss: 0.2746 | g_loss: 3.3334
Epoch [   82/  100] | d_loss: 0.1664 | g_loss: 4.4453
Epoch [   82/  100] | d_loss: 0.3162 | g_loss: 5.4170
Epoch [   82/  100] | d_loss: 6.7000 | g_loss: 6.0830
Epoch [   82/  100] | d_loss: 0.2702 | g_loss: 2.6358
Epoch [   82/  100] | d_loss: 0.2654 | g_loss: 3.8596
Epoch [   82/  100] | d_loss: 0.6421 | g_loss: 5.3422
Epoch [   83/  100] | d_loss: 1.3092 | g_loss: 0.8690
Epoch [   83/  100] | d_loss: 0.1889 | g_loss: 3.9618
Epoch [   83/  100] | d_loss: 0.1321 | g_loss: 3.3944
Epoch [   83/  100] | d_loss: 0.2172 | g_loss: 5.5358
Epoch [   83/  100] | d_loss: 0.2018 | g_loss: 3.2408
Epoch [   83/  100] | d_loss: 0.1989 | g_loss: 3.7805
Epoch [   83/  100] | d_loss: 0.1450 | g_loss: 5.0454
Epoch [   83/  100] | d_loss: 0.2559 | g_loss: 4.4778
Epoch [   84/  100] | d_loss: 0.2450 | g_loss: 3.6853
Epoch [   84/  100] | d_loss: 0.1599 | g_loss: 4.8324
Epoch [   84/  100] | d_loss: 0.4750 | g_loss: 4.4238
Epoch [   84/  100] | d_loss: 0.2360 | g_loss: 3.0627
Epoch [   84/  100] | d_loss: 0.2541 | g_loss: 3.6654
Epoch [   84/  100] | d_loss: 0.3566 | g_loss: 4.0485
Epoch [   84/  100] | d_loss: 1.4913 | g_loss: 1.4167
Epoch [   84/  100] | d_loss: 0.1710 | g_loss: 3.9441
Epoch [   85/  100] | d_loss: 0.1514 | g_loss: 4.4408
Epoch [   85/  100] | d_loss: 0.1949 | g_loss: 3.8379
Epoch [   85/  100] | d_loss: 0.3319 | g_loss: 5.7453
```

```
Epoch [   85/  100] | d_loss: 0.5641 | g_loss: 8.1855
Epoch [   85/  100] | d_loss: 0.2179 | g_loss: 3.8207
Epoch [   85/  100] | d_loss: 0.1867 | g_loss: 4.9312
Epoch [   85/  100] | d_loss: 3.5199 | g_loss: 1.0735
Epoch [   85/  100] | d_loss: 0.1555 | g_loss: 3.8586
Epoch [   86/  100] | d_loss: 0.4336 | g_loss: 5.9419
Epoch [   86/  100] | d_loss: 0.2107 | g_loss: 3.8787
Epoch [   86/  100] | d_loss: 0.1892 | g_loss: 5.2887
Epoch [   86/  100] | d_loss: 0.1830 | g_loss: 4.1988
Epoch [   86/  100] | d_loss: 0.7119 | g_loss: 3.1355
Epoch [   86/  100] | d_loss: 0.1079 | g_loss: 3.9316
Epoch [   86/  100] | d_loss: 0.2332 | g_loss: 3.7208
Epoch [   86/  100] | d_loss: 0.1528 | g_loss: 4.6443
Epoch [   87/  100] | d_loss: 0.1903 | g_loss: 3.9222
Epoch [   87/  100] | d_loss: 0.5884 | g_loss: 5.0299
Epoch [   87/  100] | d_loss: 0.1308 | g_loss: 4.5685
Epoch [   87/  100] | d_loss: 0.1749 | g_loss: 4.3336
Epoch [   87/  100] | d_loss: 0.2089 | g_loss: 4.1504
Epoch [   87/  100] | d_loss: 0.2156 | g_loss: 2.9345
Epoch [   87/  100] | d_loss: 0.2281 | g_loss: 3.6003
Epoch [   87/  100] | d_loss: 0.3279 | g_loss: 4.9982
Epoch [   88/  100] | d_loss: 0.3364 | g_loss: 2.9522
Epoch [   88/  100] | d_loss: 1.3800 | g_loss: 6.7697
Epoch [   88/  100] | d_loss: 0.2237 | g_loss: 3.5726
Epoch [   88/  100] | d_loss: 0.2063 | g_loss: 5.0111
Epoch [   88/  100] | d_loss: 0.2074 | g_loss: 4.0910
Epoch [   88/  100] | d_loss: 0.3844 | g_loss: 2.8703
Epoch [   88/  100] | d_loss: 0.1269 | g_loss: 4.5933
Epoch [   88/  100] | d_loss: 0.5319 | g_loss: 5.5147
Epoch [   89/  100] | d_loss: 0.3165 | g_loss: 4.2684
Epoch [   89/  100] | d_loss: 0.2281 | g_loss: 3.3425
Epoch [   89/  100] | d_loss: 0.1682 | g_loss: 5.0370
Epoch [   89/  100] | d_loss: 0.2248 | g_loss: 4.0059
Epoch [   89/  100] | d_loss: 0.2681 | g_loss: 4.0371
Epoch [   89/  100] | d_loss: 0.2586 | g_loss: 4.3963
Epoch [   89/  100] | d_loss: 0.1721 | g_loss: 3.4416
Epoch [   89/  100] | d_loss: 0.1957 | g_loss: 4.1662
Epoch [   90/  100] | d_loss: 0.3383 | g_loss: 4.9469
Epoch [   90/  100] | d_loss: 2.4406 | g_loss: 0.7878
Epoch [   90/  100] | d_loss: 0.1472 | g_loss: 3.7941
Epoch [   90/  100] | d_loss: 0.1444 | g_loss: 3.4512
Epoch [   90/  100] | d_loss: 0.1262 | g_loss: 3.9419
Epoch [   90/  100] | d_loss: 0.7780 | g_loss: 5.5077
Epoch [   90/  100] | d_loss: 0.2518 | g_loss: 3.2352
Epoch [   90/  100] | d_loss: 0.2507 | g_loss: 3.3418
Epoch [   91/  100] | d_loss: 0.1519 | g_loss: 3.9803
Epoch [   91/  100] | d_loss: 0.2745 | g_loss: 3.1507
Epoch [   91/  100] | d_loss: 0.1324 | g_loss: 4.2553
```

```
Epoch [   91/  100] | d_loss: 0.1281 | g_loss: 4.7575
Epoch [   91/  100] | d_loss: 0.2036 | g_loss: 4.3959
Epoch [   91/  100] | d_loss: 0.1791 | g_loss: 4.3165
Epoch [   91/  100] | d_loss: 0.2443 | g_loss: 3.9419
Epoch [   91/  100] | d_loss: 0.1714 | g_loss: 4.9631
Epoch [   92/  100] | d_loss: 0.2932 | g_loss: 3.9933
Epoch [   92/  100] | d_loss: 0.2717 | g_loss: 3.7212
Epoch [   92/  100] | d_loss: 0.1835 | g_loss: 4.2718
Epoch [   92/  100] | d_loss: 0.1793 | g_loss: 4.8001
Epoch [   92/  100] | d_loss: 4.9194 | g_loss: 7.2753
Epoch [   92/  100] | d_loss: 0.2245 | g_loss: 4.1824
Epoch [   92/  100] | d_loss: 0.1956 | g_loss: 3.1795
Epoch [   92/  100] | d_loss: 0.2948 | g_loss: 4.1452
Epoch [   93/  100] | d_loss: 0.2145 | g_loss: 3.4069
Epoch [   93/  100] | d_loss: 0.3107 | g_loss: 3.5253
Epoch [   93/  100] | d_loss: 0.2635 | g_loss: 3.1947
Epoch [   93/  100] | d_loss: 0.2067 | g_loss: 4.6473
Epoch [   93/  100] | d_loss: 0.1595 | g_loss: 4.4851
Epoch [   93/  100] | d_loss: 0.2117 | g_loss: 3.7633
Epoch [   93/  100] | d_loss: 0.1794 | g_loss: 4.3689
Epoch [   93/  100] | d_loss: 0.1785 | g_loss: 4.5826
Epoch [   94/  100] | d_loss: 0.1775 | g_loss: 4.6417
Epoch [   94/  100] | d_loss: 0.2678 | g_loss: 3.6052
Epoch [   94/  100] | d_loss: 0.1732 | g_loss: 3.9290
Epoch [   94/  100] | d_loss: 0.1667 | g_loss: 4.7930
Epoch [   94/  100] | d_loss: 0.3092 | g_loss: 4.9342
Epoch [   94/  100] | d_loss: 0.1672 | g_loss: 4.2565
Epoch [   94/  100] | d_loss: 0.2169 | g_loss: 5.0066
Epoch [   94/  100] | d_loss: 0.1995 | g_loss: 4.2080
Epoch [   95/  100] | d_loss: 0.1657 | g_loss: 4.5222
Epoch [   95/  100] | d_loss: 0.1461 | g_loss: 4.6757
Epoch [   95/  100] | d_loss: 0.2666 | g_loss: 4.6944
Epoch [   95/  100] | d_loss: 0.2029 | g_loss: 4.2820
Epoch [   95/  100] | d_loss: 0.1641 | g_loss: 5.5857
Epoch [   95/  100] | d_loss: 0.1821 | g_loss: 3.1288
Epoch [   95/  100] | d_loss: 0.2786 | g_loss: 4.5932
Epoch [   95/  100] | d_loss: 0.1871 | g_loss: 4.1119
Epoch [   96/  100] | d_loss: 0.1879 | g_loss: 4.7203
Epoch [   96/  100] | d_loss: 0.1263 | g_loss: 4.9872
Epoch [   96/  100] | d_loss: 0.2229 | g_loss: 4.6656
Epoch [   96/  100] | d_loss: 0.0891 | g_loss: 4.3621
Epoch [   96/  100] | d_loss: 0.1886 | g_loss: 4.0613
Epoch [   96/  100] | d_loss: 0.6142 | g_loss: 4.1794
Epoch [   96/  100] | d_loss: 0.2029 | g_loss: 4.3575
Epoch [   96/  100] | d_loss: 0.1863 | g_loss: 4.1715
Epoch [   97/  100] | d_loss: 0.1508 | g_loss: 4.4210
Epoch [   97/  100] | d_loss: 0.1922 | g_loss: 5.1181
Epoch [   97/  100] | d_loss: 0.1432 | g_loss: 5.0716
```

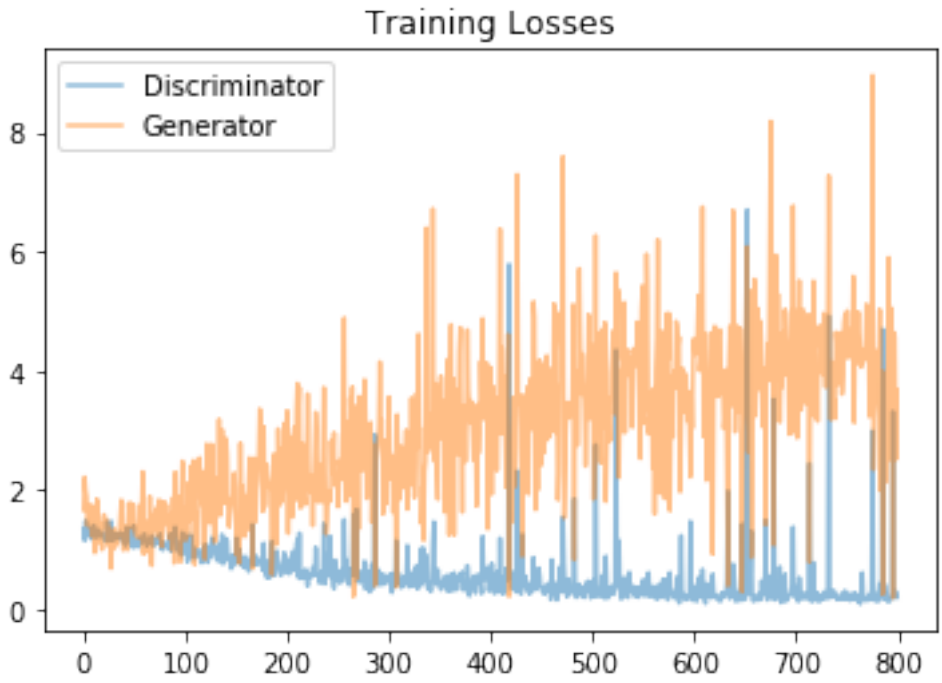
```
Epoch [   97/  100] | d_loss: 0.2304 | g_loss: 4.7674
Epoch [   97/  100] | d_loss: 0.2511 | g_loss: 3.2267
Epoch [   97/  100] | d_loss: 0.1612 | g_loss: 4.5041
Epoch [   97/  100] | d_loss: 0.2092 | g_loss: 4.8726
Epoch [   97/  100] | d_loss: 2.9901 | g_loss: 8.9585
Epoch [   98/  100] | d_loss: 0.4603 | g_loss: 2.3412
Epoch [   98/  100] | d_loss: 0.3035 | g_loss: 3.2172
Epoch [   98/  100] | d_loss: 0.1358 | g_loss: 3.4706
Epoch [   98/  100] | d_loss: 0.2484 | g_loss: 3.6779
Epoch [   98/  100] | d_loss: 0.1459 | g_loss: 4.3410
Epoch [   98/  100] | d_loss: 0.1334 | g_loss: 4.0782
Epoch [   98/  100] | d_loss: 0.2223 | g_loss: 5.0292
Epoch [   98/  100] | d_loss: 0.4365 | g_loss: 1.9929
Epoch [   99/  100] | d_loss: 0.2566 | g_loss: 3.9935
Epoch [   99/  100] | d_loss: 4.6991 | g_loss: 0.2505
Epoch [   99/  100] | d_loss: 0.1639 | g_loss: 3.9764
Epoch [   99/  100] | d_loss: 0.4163 | g_loss: 2.2402
Epoch [   99/  100] | d_loss: 0.3076 | g_loss: 2.1310
Epoch [   99/  100] | d_loss: 0.2670 | g_loss: 2.9067
Epoch [   99/  100] | d_loss: 0.1251 | g_loss: 4.3272
Epoch [   99/  100] | d_loss: 0.1955 | g_loss: 5.8984
Epoch [  100/  100] | d_loss: 0.1393 | g_loss: 2.5174
Epoch [  100/  100] | d_loss: 0.1728 | g_loss: 5.0582
Epoch [  100/  100] | d_loss: 0.2144 | g_loss: 3.2229
Epoch [  100/  100] | d_loss: 3.3162 | g_loss: 0.1878
Epoch [  100/  100] | d_loss: 0.2849 | g_loss: 4.6388
Epoch [  100/  100] | d_loss: 0.1982 | g_loss: 4.5688
Epoch [  100/  100] | d_loss: 0.2930 | g_loss: 2.5134
Epoch [  100/  100] | d_loss: 0.1976 | g_loss: 3.6776
```

2.8 Training loss

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

```
In [22]: 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[22]: <matplotlib.legend.Legend at 0x7f89e831a4e0>
```



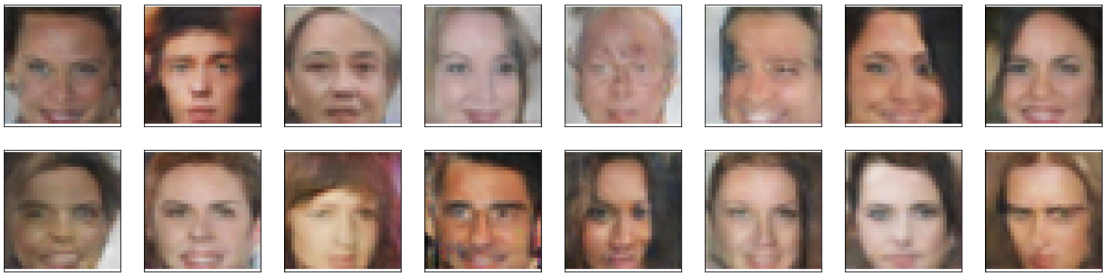
2.9 Generator samples from training

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

```
In [23]: # 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 [24]: # Load samples from generator, taken while training
         with open('train_samples.pkl', 'rb') as f:
             samples = pkl.load(f)

In [25]: _ = 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: My model uses BCELLoss, I would experiment with other loss functions and also optimize the Discriminator and Generator classes

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.