dlnd_face_generation

March 28, 2020

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
        from torch.utils.data import DataLoader
```

```
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
```

```
    # 1. Resise the image with dimension of image_size * image_size
    # 2. Convert the numpy image to tensor
    transform = transforms.Compose([transforms.Resize(image_size),transforms.ToTensor()])

    dataset = datasets.ImageFolder(data_dir, transform)

    data_loader = DataLoader(dataset=dataset, batch_size=batch_size, shuffle=True)

    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 = 20
img_size = 32

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

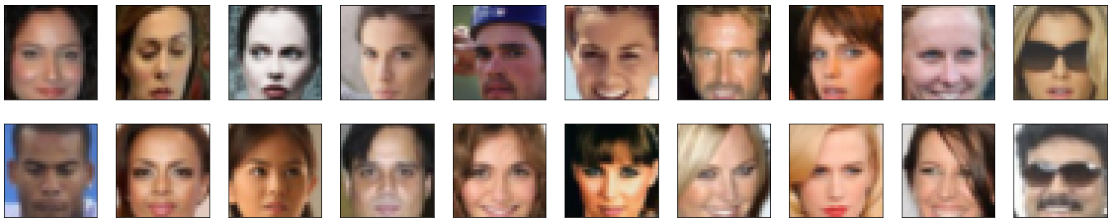
Next, you can view some images! You should seen square images of somewhat-centered faces.

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

In [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(-0.9843)
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]: # 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, out_channels,
                                    kernel_size, stride, padding, bias=False)

             # append conv layer
             layers.append(conv_layer)

             if batch_norm:
                 # append batchnorm layer
                 layers.append(nn.BatchNorm2d(out_channels))

             # using Sequential container
             return nn.Sequential(*layers)


         # helper deconv function
         def deconv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True
             """Creates a transposed-convolutional layer, with optional batch normalization.
             """
             # create a sequence of transpose + optional batch norm layers
             layers = []
             transpose_conv_layer = nn.ConvTranspose2d(in_channels, out_channels,
                                                       kernel_size, stride, padding, bias=False)
             # append transpose convolutional layer
             layers.append(transpose_conv_layer)

             if batch_norm:
                 # append batchnorm layer
                 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

        # 32x32 input
        self.conv1 = conv(3, conv_dim, 4, batch_norm=False) # first layer, no batch_nor
        # 16x16 out
        self.conv2 = conv(conv_dim, conv_dim*2, 4)
        # 8x8 out
        self.conv3 = conv(conv_dim*2, conv_dim*4, 4)
        # 4x4 out

        # final, fully-connected layer
        self.fc = nn.Linear(conv_dim*4*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

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

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

        # final output layer
        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 [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__()

                 self.conv_dim = conv_dim

                 # first, fully-connected layer
                 self.fc = nn.Linear(z_size, conv_dim*4*4*4)

                 # transpose conv layers
                 self.t_conv1 = deconv(conv_dim*4, conv_dim*2, 4)
                 self.t_conv2 = deconv(conv_dim*2, conv_dim, 4)
                 self.t_conv3 = deconv(conv_dim, 3, 4, batch_norm=False)


             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
                 """
```

```
            # fully-connected layer
            out = self.fc(x)
            # reshape with size of batch_size * depth * 4 * 4
            out = out.view(-1, self.conv_dim*4, 4, 4)

            # hidden transpose conv layers + relu
            out = F.relu(self.t_conv1(out))
            out = F.relu(self.t_conv2(out))

            # last layer + tanh activation
            out = self.t_conv3(out)
            out = F.tanh(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 [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

        # classname will be something like:
        # `Conv`, `BatchNorm2d`, `Linear`, etc.
        # TODO: Apply initial weights to convolutional and linear layers
        # for every Linear layer in a model..
        if classname.find('Linear') != -1:
            # apply a uniform distribution to the weights and a bias=0
            m.weight.data.uniform_(0.0, 1.0)
            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 [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): Sequential(
    (0): Conv2d(3, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): 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)
  (t_conv1): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (t_conv2): Sequential(
    (0): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (t_conv3): Sequential(
    (0): ConvTranspose2d(32, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
)
```

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 [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!
```

2.5 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 [30]: def real_loss(D_out):
             '''Calculates how close discriminator outputs are to being real.
                param, D_out: discriminator logits
                return: real loss'''
             loss = torch.mean((D_out-1)**2)
             return loss

         def fake_loss(D_out):
             '''Calculates how close discriminator outputs are to being fake.
                param, D_out: discriminator logits
                return: fake loss'''
             loss = torch.mean(D_out**2)
             return loss
```

2.6 Optimizers

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

```
In [31]: import torch.optim as optim

         lr=0.0002
         beta1=0.5
```

```
beta2=0.999
# 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 [32]: 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
        # =============================================
        d_optimizer.zero_grad()

        # Train with real images
        # Compute the discriminator losses on real images
        if train_on_gpu:
            real_images = real_images.cuda()

        D_real = D(real_images)
        d_real_loss = real_loss(D_real)

        # Train with fake images
        # Generate fake images
        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_images = G(z)

        # Compute the discriminator losses on fake images
        D_fake = D(fake_images)
        d_fake_loss = fake_loss(D_fake)

        # add up loss and perform backprop
        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()

        # Train with fake images and flipped labels
        # Generate fake images
        z = np.random.uniform(-1, 1, size=(batch_size, z_size))
        z = torch.from_numpy(z).float()
        if train_on_gpu:
```

```
            z = z.cuda()
        fake_images = G(z)

        # Compute the discriminator losses on fake images
        # using flipped labels!
        D_fake = D(fake_images)
        g_loss = real_loss(D_fake) # use real loss to flip labels

        # perform backprop
        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 [33]:
```
# set number of epochs
n_epochs = 10


"""
```

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

```
Epoch [    1/   10] | d_loss: 2.4301 | g_loss: 5.4698
Epoch [    1/   10] | d_loss: 2.4666 | g_loss: 1.1907
Epoch [    1/   10] | d_loss: 2.4380 | g_loss: 1.6033
Epoch [    1/   10] | d_loss: 1.6975 | g_loss: 1.9601
Epoch [    1/   10] | d_loss: 1.4815 | g_loss: 3.6159
Epoch [    1/   10] | d_loss: 1.4430 | g_loss: 0.5113
Epoch [    1/   10] | d_loss: 1.1758 | g_loss: 0.3992
Epoch [    1/   10] | d_loss: 1.7191 | g_loss: 2.2333
Epoch [    1/   10] | d_loss: 0.6551 | g_loss: 1.2663
Epoch [    1/   10] | d_loss: 0.5098 | g_loss: 0.4489
Epoch [    1/   10] | d_loss: 2.0108 | g_loss: 2.8627
Epoch [    1/   10] | d_loss: 1.3275 | g_loss: 2.0122
Epoch [    1/   10] | d_loss: 0.8080 | g_loss: 0.3247
Epoch [    1/   10] | d_loss: 1.0438 | g_loss: 0.9972
Epoch [    1/   10] | d_loss: 0.6504 | g_loss: 0.6662
Epoch [    1/   10] | d_loss: 0.3545 | g_loss: 0.3549
Epoch [    1/   10] | d_loss: 0.3929 | g_loss: 3.1667
Epoch [    1/   10] | d_loss: 0.6101 | g_loss: 1.0339
Epoch [    1/   10] | d_loss: 0.7523 | g_loss: 0.5417
Epoch [    1/   10] | d_loss: 0.8278 | g_loss: 1.3065
Epoch [    1/   10] | d_loss: 0.5695 | g_loss: 2.1941
Epoch [    1/   10] | d_loss: 0.3383 | g_loss: 2.0287
Epoch [    1/   10] | d_loss: 0.6365 | g_loss: 0.9866
Epoch [    1/   10] | d_loss: 0.3115 | g_loss: 0.8479
Epoch [    1/   10] | d_loss: 0.5624 | g_loss: 0.5255
Epoch [    1/   10] | d_loss: 0.3915 | g_loss: 0.6716
Epoch [    1/   10] | d_loss: 0.5632 | g_loss: 0.8053
Epoch [    1/   10] | d_loss: 0.2718 | g_loss: 1.1872
Epoch [    1/   10] | d_loss: 0.4458 | g_loss: 1.1765
Epoch [    1/   10] | d_loss: 0.5812 | g_loss: 1.1717
Epoch [    1/   10] | d_loss: 0.3097 | g_loss: 0.5699
Epoch [    1/   10] | d_loss: 0.3263 | g_loss: 1.5444
Epoch [    1/   10] | d_loss: 0.4281 | g_loss: 0.6642
Epoch [    1/   10] | d_loss: 0.2866 | g_loss: 1.1970
Epoch [    1/   10] | d_loss: 0.2676 | g_loss: 0.9668
Epoch [    1/   10] | d_loss: 0.3946 | g_loss: 1.5125
Epoch [    1/   10] | d_loss: 0.2080 | g_loss: 0.6221
Epoch [    1/   10] | d_loss: 0.3298 | g_loss: 0.6255
Epoch [    1/   10] | d_loss: 0.4256 | g_loss: 0.8382
Epoch [    1/   10] | d_loss: 0.6686 | g_loss: 1.3167
Epoch [    1/   10] | d_loss: 0.2904 | g_loss: 1.3104
Epoch [    1/   10] | d_loss: 0.2618 | g_loss: 1.5388
Epoch [    1/   10] | d_loss: 0.2439 | g_loss: 0.7373
```

```
Epoch [    1/   10] | d_loss: 0.2098 | g_loss: 0.9405
Epoch [    1/   10] | d_loss: 0.1820 | g_loss: 1.3970
Epoch [    1/   10] | d_loss: 0.2889 | g_loss: 0.3876
Epoch [    1/   10] | d_loss: 0.3002 | g_loss: 0.7589
Epoch [    1/   10] | d_loss: 0.2618 | g_loss: 0.3258
Epoch [    1/   10] | d_loss: 0.4060 | g_loss: 1.2993
Epoch [    1/   10] | d_loss: 0.2523 | g_loss: 0.7481
Epoch [    1/   10] | d_loss: 0.2863 | g_loss: 1.1284
Epoch [    1/   10] | d_loss: 0.7432 | g_loss: 1.4819
Epoch [    1/   10] | d_loss: 0.3180 | g_loss: 1.3886
Epoch [    1/   10] | d_loss: 0.4406 | g_loss: 1.2872
Epoch [    1/   10] | d_loss: 0.2513 | g_loss: 0.7864
Epoch [    1/   10] | d_loss: 0.3401 | g_loss: 0.6191
Epoch [    1/   10] | d_loss: 0.2608 | g_loss: 1.6422
Epoch [    1/   10] | d_loss: 0.2133 | g_loss: 0.4580
Epoch [    1/   10] | d_loss: 0.3480 | g_loss: 1.0717
Epoch [    1/   10] | d_loss: 0.2643 | g_loss: 1.0021
Epoch [    1/   10] | d_loss: 0.3369 | g_loss: 0.8871
Epoch [    1/   10] | d_loss: 0.2199 | g_loss: 0.6149
Epoch [    1/   10] | d_loss: 0.2501 | g_loss: 1.0501
Epoch [    1/   10] | d_loss: 0.4619 | g_loss: 0.7417
Epoch [    1/   10] | d_loss: 0.2571 | g_loss: 0.8041
Epoch [    1/   10] | d_loss: 0.2849 | g_loss: 0.9420
Epoch [    1/   10] | d_loss: 0.2357 | g_loss: 0.8500
Epoch [    1/   10] | d_loss: 0.2266 | g_loss: 1.7933
Epoch [    1/   10] | d_loss: 0.5859 | g_loss: 0.9404
Epoch [    1/   10] | d_loss: 0.1262 | g_loss: 1.2313
Epoch [    1/   10] | d_loss: 0.2505 | g_loss: 0.8542
Epoch [    1/   10] | d_loss: 0.3549 | g_loss: 0.7152
Epoch [    1/   10] | d_loss: 0.3030 | g_loss: 0.8172
Epoch [    1/   10] | d_loss: 0.3014 | g_loss: 0.7300
Epoch [    1/   10] | d_loss: 0.2581 | g_loss: 1.3750
Epoch [    1/   10] | d_loss: 0.2073 | g_loss: 1.2001
Epoch [    1/   10] | d_loss: 0.3518 | g_loss: 0.8672
Epoch [    1/   10] | d_loss: 0.2523 | g_loss: 0.6271
Epoch [    1/   10] | d_loss: 0.1344 | g_loss: 0.4072
Epoch [    1/   10] | d_loss: 0.2116 | g_loss: 0.9096
Epoch [    1/   10] | d_loss: 0.2509 | g_loss: 1.2891
Epoch [    1/   10] | d_loss: 0.3585 | g_loss: 1.5060
Epoch [    1/   10] | d_loss: 0.3059 | g_loss: 0.9315
Epoch [    1/   10] | d_loss: 0.2151 | g_loss: 0.7576
Epoch [    1/   10] | d_loss: 0.5953 | g_loss: 2.0661
Epoch [    1/   10] | d_loss: 0.1591 | g_loss: 1.1416
Epoch [    1/   10] | d_loss: 0.1919 | g_loss: 0.4699
Epoch [    1/   10] | d_loss: 0.1248 | g_loss: 0.6701
Epoch [    1/   10] | d_loss: 0.1413 | g_loss: 1.0504
Epoch [    1/   10] | d_loss: 0.2227 | g_loss: 0.8597
Epoch [    2/   10] | d_loss: 0.2273 | g_loss: 1.0478
```

```
Epoch [    2/   10] | d_loss: 0.1586 | g_loss: 0.9353
Epoch [    2/   10] | d_loss: 0.1221 | g_loss: 1.6211
Epoch [    2/   10] | d_loss: 0.6667 | g_loss: 0.4735
Epoch [    2/   10] | d_loss: 0.2474 | g_loss: 1.1286
Epoch [    2/   10] | d_loss: 0.1740 | g_loss: 0.9724
Epoch [    2/   10] | d_loss: 0.2673 | g_loss: 0.7514
Epoch [    2/   10] | d_loss: 0.2999 | g_loss: 0.7097
Epoch [    2/   10] | d_loss: 0.2187 | g_loss: 0.8201
Epoch [    2/   10] | d_loss: 0.2965 | g_loss: 0.5710
Epoch [    2/   10] | d_loss: 0.2771 | g_loss: 0.8025
Epoch [    2/   10] | d_loss: 0.2366 | g_loss: 0.8224
Epoch [    2/   10] | d_loss: 0.1368 | g_loss: 0.8139
Epoch [    2/   10] | d_loss: 0.3419 | g_loss: 0.7593
Epoch [    2/   10] | d_loss: 0.2078 | g_loss: 0.9907
Epoch [    2/   10] | d_loss: 0.1868 | g_loss: 0.4414
Epoch [    2/   10] | d_loss: 0.1954 | g_loss: 0.6327
Epoch [    2/   10] | d_loss: 0.3482 | g_loss: 0.9489
Epoch [    2/   10] | d_loss: 0.1813 | g_loss: 0.9793
Epoch [    2/   10] | d_loss: 0.2722 | g_loss: 0.7139
Epoch [    2/   10] | d_loss: 0.1866 | g_loss: 1.1049
Epoch [    2/   10] | d_loss: 0.1096 | g_loss: 0.5541
Epoch [    2/   10] | d_loss: 0.1406 | g_loss: 1.2488
Epoch [    2/   10] | d_loss: 0.1904 | g_loss: 0.8920
Epoch [    2/   10] | d_loss: 0.2311 | g_loss: 0.9171
Epoch [    2/   10] | d_loss: 0.3589 | g_loss: 1.2204
Epoch [    2/   10] | d_loss: 0.4547 | g_loss: 0.7026
Epoch [    2/   10] | d_loss: 0.1372 | g_loss: 1.1816
Epoch [    2/   10] | d_loss: 0.1471 | g_loss: 0.6752
Epoch [    2/   10] | d_loss: 0.1921 | g_loss: 1.1441
Epoch [    2/   10] | d_loss: 0.1407 | g_loss: 0.7794
Epoch [    2/   10] | d_loss: 0.1421 | g_loss: 1.8438
Epoch [    2/   10] | d_loss: 0.2043 | g_loss: 0.9627
Epoch [    2/   10] | d_loss: 0.3672 | g_loss: 1.2366
Epoch [    2/   10] | d_loss: 0.1329 | g_loss: 0.8022
Epoch [    2/   10] | d_loss: 0.1326 | g_loss: 0.8359
Epoch [    2/   10] | d_loss: 0.1238 | g_loss: 0.7223
Epoch [    2/   10] | d_loss: 0.2109 | g_loss: 0.9854
Epoch [    2/   10] | d_loss: 0.1687 | g_loss: 0.5950
Epoch [    2/   10] | d_loss: 0.1582 | g_loss: 1.2229
Epoch [    2/   10] | d_loss: 0.1922 | g_loss: 0.6963
Epoch [    2/   10] | d_loss: 0.1813 | g_loss: 0.7448
Epoch [    2/   10] | d_loss: 0.2164 | g_loss: 0.6855
Epoch [    2/   10] | d_loss: 0.5955 | g_loss: 0.2211
Epoch [    2/   10] | d_loss: 0.1973 | g_loss: 0.7422
Epoch [    2/   10] | d_loss: 0.1415 | g_loss: 0.7276
Epoch [    2/   10] | d_loss: 0.1406 | g_loss: 0.7571
Epoch [    2/   10] | d_loss: 0.1722 | g_loss: 1.0832
Epoch [    2/   10] | d_loss: 0.1503 | g_loss: 1.0152
```

```
Epoch [    2/   10] | d_loss: 0.1732 | g_loss: 0.8740
Epoch [    2/   10] | d_loss: 0.1998 | g_loss: 0.6524
Epoch [    2/   10] | d_loss: 0.1098 | g_loss: 0.9481
Epoch [    2/   10] | d_loss: 0.1135 | g_loss: 1.2074
Epoch [    2/   10] | d_loss: 0.1092 | g_loss: 0.8879
Epoch [    2/   10] | d_loss: 0.1303 | g_loss: 0.5540
Epoch [    2/   10] | d_loss: 0.2037 | g_loss: 1.2420
Epoch [    2/   10] | d_loss: 0.1481 | g_loss: 0.7834
Epoch [    2/   10] | d_loss: 0.1630 | g_loss: 1.0697
Epoch [    2/   10] | d_loss: 0.2416 | g_loss: 0.8787
Epoch [    2/   10] | d_loss: 0.1977 | g_loss: 0.7781
Epoch [    2/   10] | d_loss: 0.2855 | g_loss: 0.8862
Epoch [    2/   10] | d_loss: 0.1636 | g_loss: 0.5395
Epoch [    2/   10] | d_loss: 0.2160 | g_loss: 0.8314
Epoch [    2/   10] | d_loss: 0.1869 | g_loss: 0.2741
Epoch [    2/   10] | d_loss: 0.1093 | g_loss: 1.0089
Epoch [    2/   10] | d_loss: 0.0990 | g_loss: 1.3988
Epoch [    2/   10] | d_loss: 0.1473 | g_loss: 0.5828
Epoch [    2/   10] | d_loss: 0.0974 | g_loss: 0.5806
Epoch [    2/   10] | d_loss: 0.1977 | g_loss: 0.5029
Epoch [    2/   10] | d_loss: 0.1820 | g_loss: 1.0109
Epoch [    2/   10] | d_loss: 0.1286 | g_loss: 0.9972
Epoch [    2/   10] | d_loss: 0.0617 | g_loss: 0.5742
Epoch [    2/   10] | d_loss: 0.1627 | g_loss: 0.4686
Epoch [    2/   10] | d_loss: 0.2180 | g_loss: 0.4645
Epoch [    2/   10] | d_loss: 0.3335 | g_loss: 0.6608
Epoch [    2/   10] | d_loss: 0.2687 | g_loss: 0.6698
Epoch [    2/   10] | d_loss: 0.4823 | g_loss: 0.5405
Epoch [    2/   10] | d_loss: 0.1266 | g_loss: 0.5552
Epoch [    2/   10] | d_loss: 0.2666 | g_loss: 0.4675
Epoch [    2/   10] | d_loss: 0.1993 | g_loss: 1.0593
Epoch [    2/   10] | d_loss: 0.1269 | g_loss: 0.6496
Epoch [    2/   10] | d_loss: 0.1847 | g_loss: 1.0632
Epoch [    2/   10] | d_loss: 0.1026 | g_loss: 0.6414
Epoch [    2/   10] | d_loss: 0.1475 | g_loss: 0.6354
Epoch [    2/   10] | d_loss: 0.2603 | g_loss: 0.6433
Epoch [    2/   10] | d_loss: 0.1324 | g_loss: 0.7045
Epoch [    2/   10] | d_loss: 0.3273 | g_loss: 0.9276
Epoch [    2/   10] | d_loss: 0.1453 | g_loss: 0.9203
Epoch [    2/   10] | d_loss: 0.2249 | g_loss: 0.7645
Epoch [    2/   10] | d_loss: 0.1023 | g_loss: 0.9293
Epoch [    3/   10] | d_loss: 0.1477 | g_loss: 0.4435
Epoch [    3/   10] | d_loss: 0.1997 | g_loss: 1.1280
Epoch [    3/   10] | d_loss: 0.1738 | g_loss: 1.7359
Epoch [    3/   10] | d_loss: 0.1246 | g_loss: 1.2972
Epoch [    3/   10] | d_loss: 0.1483 | g_loss: 0.4876
Epoch [    3/   10] | d_loss: 0.3905 | g_loss: 0.5876
Epoch [    3/   10] | d_loss: 0.4177 | g_loss: 1.4693
```

```
Epoch [    3/   10] | d_loss: 0.0978 | g_loss: 1.2173
Epoch [    3/   10] | d_loss: 0.2828 | g_loss: 0.6652
Epoch [    3/   10] | d_loss: 0.1460 | g_loss: 1.1835
Epoch [    3/   10] | d_loss: 0.0733 | g_loss: 0.8306
Epoch [    3/   10] | d_loss: 0.2059 | g_loss: 0.7172
Epoch [    3/   10] | d_loss: 0.5297 | g_loss: 0.7692
Epoch [    3/   10] | d_loss: 0.2464 | g_loss: 0.8811
Epoch [    3/   10] | d_loss: 0.1623 | g_loss: 0.9085
Epoch [    3/   10] | d_loss: 0.0649 | g_loss: 1.0105
Epoch [    3/   10] | d_loss: 0.3517 | g_loss: 0.6047
Epoch [    3/   10] | d_loss: 0.1770 | g_loss: 1.2705
Epoch [    3/   10] | d_loss: 0.2200 | g_loss: 0.5428
Epoch [    3/   10] | d_loss: 0.3723 | g_loss: 0.6963
Epoch [    3/   10] | d_loss: 0.1455 | g_loss: 0.7673
Epoch [    3/   10] | d_loss: 0.2118 | g_loss: 0.9393
Epoch [    3/   10] | d_loss: 0.1572 | g_loss: 0.7319
Epoch [    3/   10] | d_loss: 0.1506 | g_loss: 0.6357
Epoch [    3/   10] | d_loss: 0.3206 | g_loss: 0.8373
Epoch [    3/   10] | d_loss: 0.2689 | g_loss: 1.0609
Epoch [    3/   10] | d_loss: 0.2159 | g_loss: 0.7673
Epoch [    3/   10] | d_loss: 0.2259 | g_loss: 0.5822
Epoch [    3/   10] | d_loss: 0.1987 | g_loss: 1.2263
Epoch [    3/   10] | d_loss: 0.3671 | g_loss: 1.0571
Epoch [    3/   10] | d_loss: 0.3175 | g_loss: 1.2576
Epoch [    3/   10] | d_loss: 0.2254 | g_loss: 0.5640
Epoch [    3/   10] | d_loss: 0.3605 | g_loss: 0.8751
Epoch [    3/   10] | d_loss: 0.1459 | g_loss: 0.2847
Epoch [    3/   10] | d_loss: 0.2004 | g_loss: 0.4645
Epoch [    3/   10] | d_loss: 0.1092 | g_loss: 1.0260
Epoch [    3/   10] | d_loss: 0.2184 | g_loss: 1.0101
Epoch [    3/   10] | d_loss: 0.2900 | g_loss: 0.3371
Epoch [    3/   10] | d_loss: 0.1128 | g_loss: 0.6576
Epoch [    3/   10] | d_loss: 0.1772 | g_loss: 0.5483
Epoch [    3/   10] | d_loss: 0.6985 | g_loss: 0.3466
Epoch [    3/   10] | d_loss: 0.3652 | g_loss: 2.0290
Epoch [    3/   10] | d_loss: 0.1592 | g_loss: 1.0406
Epoch [    3/   10] | d_loss: 0.1027 | g_loss: 0.7812
Epoch [    3/   10] | d_loss: 0.1011 | g_loss: 0.5741
Epoch [    3/   10] | d_loss: 0.1587 | g_loss: 0.8486
Epoch [    3/   10] | d_loss: 0.1230 | g_loss: 0.6294
Epoch [    3/   10] | d_loss: 0.2113 | g_loss: 0.6632
Epoch [    3/   10] | d_loss: 0.2070 | g_loss: 0.6264
Epoch [    3/   10] | d_loss: 0.1147 | g_loss: 0.6860
Epoch [    3/   10] | d_loss: 0.6593 | g_loss: 0.9105
Epoch [    3/   10] | d_loss: 0.2297 | g_loss: 0.5497
Epoch [    3/   10] | d_loss: 0.2411 | g_loss: 0.7682
Epoch [    3/   10] | d_loss: 0.0737 | g_loss: 0.6528
Epoch [    3/   10] | d_loss: 0.2252 | g_loss: 1.0061
```

```
Epoch [    3/   10] | d_loss: 0.3986 | g_loss: 1.1245
Epoch [    3/   10] | d_loss: 0.1073 | g_loss: 0.9194
Epoch [    3/   10] | d_loss: 0.2128 | g_loss: 0.8240
Epoch [    3/   10] | d_loss: 0.4964 | g_loss: 1.0855
Epoch [    3/   10] | d_loss: 0.1298 | g_loss: 0.8910
Epoch [    3/   10] | d_loss: 0.1139 | g_loss: 0.8290
Epoch [    3/   10] | d_loss: 0.1885 | g_loss: 0.5752
Epoch [    3/   10] | d_loss: 0.1468 | g_loss: 0.6662
Epoch [    3/   10] | d_loss: 0.1075 | g_loss: 0.6024
Epoch [    3/   10] | d_loss: 0.1842 | g_loss: 0.6351
Epoch [    3/   10] | d_loss: 0.1679 | g_loss: 0.6185
Epoch [    3/   10] | d_loss: 0.2115 | g_loss: 0.6180
Epoch [    3/   10] | d_loss: 0.2221 | g_loss: 0.9892
Epoch [    3/   10] | d_loss: 0.1570 | g_loss: 0.9202
Epoch [    3/   10] | d_loss: 0.1904 | g_loss: 0.5191
Epoch [    3/   10] | d_loss: 0.5179 | g_loss: 1.2689
Epoch [    3/   10] | d_loss: 0.0963 | g_loss: 1.1936
Epoch [    3/   10] | d_loss: 0.1591 | g_loss: 0.2346
Epoch [    3/   10] | d_loss: 0.1879 | g_loss: 0.7372
Epoch [    3/   10] | d_loss: 0.4905 | g_loss: 0.6269
Epoch [    3/   10] | d_loss: 0.1812 | g_loss: 0.8337
Epoch [    3/   10] | d_loss: 0.1223 | g_loss: 0.6924
Epoch [    3/   10] | d_loss: 0.3086 | g_loss: 0.3638
Epoch [    3/   10] | d_loss: 0.1841 | g_loss: 1.1502
Epoch [    3/   10] | d_loss: 0.2534 | g_loss: 0.6229
Epoch [    3/   10] | d_loss: 0.3765 | g_loss: 0.6009
Epoch [    3/   10] | d_loss: 0.2235 | g_loss: 0.5492
Epoch [    3/   10] | d_loss: 0.2876 | g_loss: 0.4876
Epoch [    3/   10] | d_loss: 0.1668 | g_loss: 0.7110
Epoch [    3/   10] | d_loss: 0.1749 | g_loss: 0.7771
Epoch [    3/   10] | d_loss: 0.1701 | g_loss: 1.2170
Epoch [    3/   10] | d_loss: 0.3327 | g_loss: 0.6539
Epoch [    3/   10] | d_loss: 0.2503 | g_loss: 0.7935
Epoch [    3/   10] | d_loss: 0.1867 | g_loss: 0.5411
Epoch [    3/   10] | d_loss: 0.1315 | g_loss: 1.3559
Epoch [    4/   10] | d_loss: 0.4774 | g_loss: 0.6415
Epoch [    4/   10] | d_loss: 0.2884 | g_loss: 1.1213
Epoch [    4/   10] | d_loss: 0.2329 | g_loss: 0.8128
Epoch [    4/   10] | d_loss: 0.3128 | g_loss: 0.3214
Epoch [    4/   10] | d_loss: 0.0897 | g_loss: 1.1882
Epoch [    4/   10] | d_loss: 0.2466 | g_loss: 0.7059
Epoch [    4/   10] | d_loss: 0.0892 | g_loss: 0.7516
Epoch [    4/   10] | d_loss: 0.4310 | g_loss: 1.1664
Epoch [    4/   10] | d_loss: 0.1297 | g_loss: 0.7091
Epoch [    4/   10] | d_loss: 0.1625 | g_loss: 0.4306
Epoch [    4/   10] | d_loss: 0.5555 | g_loss: 0.7452
Epoch [    4/   10] | d_loss: 0.2684 | g_loss: 0.9067
Epoch [    4/   10] | d_loss: 0.1514 | g_loss: 0.7361
```

```
Epoch [    4/   10] | d_loss: 0.1658 | g_loss: 1.1429
Epoch [    4/   10] | d_loss: 0.2808 | g_loss: 0.6010
Epoch [    4/   10] | d_loss: 0.1623 | g_loss: 0.6886
Epoch [    4/   10] | d_loss: 0.2627 | g_loss: 0.5946
Epoch [    4/   10] | d_loss: 0.2403 | g_loss: 0.7725
Epoch [    4/   10] | d_loss: 0.1517 | g_loss: 0.7087
Epoch [    4/   10] | d_loss: 0.1177 | g_loss: 1.1131
Epoch [    4/   10] | d_loss: 0.0860 | g_loss: 0.9126
Epoch [    4/   10] | d_loss: 0.3287 | g_loss: 0.6612
Epoch [    4/   10] | d_loss: 0.1845 | g_loss: 0.9595
Epoch [    4/   10] | d_loss: 0.0864 | g_loss: 0.8854
Epoch [    4/   10] | d_loss: 0.2241 | g_loss: 0.6319
Epoch [    4/   10] | d_loss: 0.1000 | g_loss: 1.1865
Epoch [    4/   10] | d_loss: 0.2102 | g_loss: 0.7519
Epoch [    4/   10] | d_loss: 0.2969 | g_loss: 0.8532
Epoch [    4/   10] | d_loss: 0.1462 | g_loss: 1.0898
Epoch [    4/   10] | d_loss: 0.1802 | g_loss: 0.9828
Epoch [    4/   10] | d_loss: 0.1989 | g_loss: 0.6293
Epoch [    4/   10] | d_loss: 0.1246 | g_loss: 0.9289
Epoch [    4/   10] | d_loss: 0.5096 | g_loss: 0.4343
Epoch [    4/   10] | d_loss: 0.1714 | g_loss: 0.6465
Epoch [    4/   10] | d_loss: 0.1394 | g_loss: 1.2927
Epoch [    4/   10] | d_loss: 0.1748 | g_loss: 0.5642
Epoch [    4/   10] | d_loss: 0.1836 | g_loss: 0.8647
Epoch [    4/   10] | d_loss: 0.2566 | g_loss: 0.3795
Epoch [    4/   10] | d_loss: 0.2775 | g_loss: 0.6541
Epoch [    4/   10] | d_loss: 0.1851 | g_loss: 1.0538
Epoch [    4/   10] | d_loss: 0.1024 | g_loss: 0.9445
Epoch [    4/   10] | d_loss: 0.2105 | g_loss: 0.4854
Epoch [    4/   10] | d_loss: 0.1903 | g_loss: 0.6250
Epoch [    4/   10] | d_loss: 0.1363 | g_loss: 0.7522
Epoch [    4/   10] | d_loss: 0.1707 | g_loss: 0.7368
Epoch [    4/   10] | d_loss: 0.0875 | g_loss: 0.9679
Epoch [    4/   10] | d_loss: 0.0984 | g_loss: 0.5300
Epoch [    4/   10] | d_loss: 0.2072 | g_loss: 0.8897
Epoch [    4/   10] | d_loss: 0.1608 | g_loss: 0.6619
Epoch [    4/   10] | d_loss: 0.3024 | g_loss: 1.3677
Epoch [    4/   10] | d_loss: 0.1429 | g_loss: 0.8275
Epoch [    4/   10] | d_loss: 0.1708 | g_loss: 0.4645
Epoch [    4/   10] | d_loss: 0.1836 | g_loss: 0.8108
Epoch [    4/   10] | d_loss: 0.1778 | g_loss: 0.6766
Epoch [    4/   10] | d_loss: 0.2087 | g_loss: 0.6569
Epoch [    4/   10] | d_loss: 0.2367 | g_loss: 0.6982
Epoch [    4/   10] | d_loss: 0.1133 | g_loss: 1.4110
Epoch [    4/   10] | d_loss: 0.1996 | g_loss: 0.7531
Epoch [    4/   10] | d_loss: 0.2099 | g_loss: 0.8082
Epoch [    4/   10] | d_loss: 0.1660 | g_loss: 0.9895
Epoch [    4/   10] | d_loss: 0.1235 | g_loss: 0.7032
```

```
Epoch [    4/   10] | d_loss: 0.1379 | g_loss: 0.5561
Epoch [    4/   10] | d_loss: 0.2295 | g_loss: 0.6665
Epoch [    4/   10] | d_loss: 0.3970 | g_loss: 0.7565
Epoch [    4/   10] | d_loss: 0.0937 | g_loss: 1.0422
Epoch [    4/   10] | d_loss: 0.2673 | g_loss: 0.5108
Epoch [    4/   10] | d_loss: 0.0997 | g_loss: 1.2693
Epoch [    4/   10] | d_loss: 0.1359 | g_loss: 0.8128
Epoch [    4/   10] | d_loss: 0.2059 | g_loss: 0.3327
Epoch [    4/   10] | d_loss: 0.1440 | g_loss: 0.5922
Epoch [    4/   10] | d_loss: 0.1055 | g_loss: 1.0923
Epoch [    4/   10] | d_loss: 0.1349 | g_loss: 1.3305
Epoch [    4/   10] | d_loss: 0.2495 | g_loss: 0.6771
Epoch [    4/   10] | d_loss: 0.0604 | g_loss: 0.9744
Epoch [    4/   10] | d_loss: 0.4490 | g_loss: 0.7523
Epoch [    4/   10] | d_loss: 0.2592 | g_loss: 0.3491
Epoch [    4/   10] | d_loss: 0.4751 | g_loss: 0.6242
Epoch [    4/   10] | d_loss: 0.1333 | g_loss: 0.4658
Epoch [    4/   10] | d_loss: 0.3729 | g_loss: 0.8569
Epoch [    4/   10] | d_loss: 0.2086 | g_loss: 0.2759
Epoch [    4/   10] | d_loss: 0.1492 | g_loss: 0.7764
Epoch [    4/   10] | d_loss: 0.1120 | g_loss: 0.4064
Epoch [    4/   10] | d_loss: 0.1335 | g_loss: 0.8220
Epoch [    4/   10] | d_loss: 0.1165 | g_loss: 0.3733
Epoch [    4/   10] | d_loss: 0.0915 | g_loss: 0.9489
Epoch [    4/   10] | d_loss: 0.1247 | g_loss: 0.4667
Epoch [    4/   10] | d_loss: 0.1666 | g_loss: 1.0101
Epoch [    4/   10] | d_loss: 0.4955 | g_loss: 1.0080
Epoch [    4/   10] | d_loss: 0.2772 | g_loss: 1.0411
Epoch [    4/   10] | d_loss: 0.0841 | g_loss: 1.1307
Epoch [    5/   10] | d_loss: 0.2474 | g_loss: 0.8288
Epoch [    5/   10] | d_loss: 0.2309 | g_loss: 1.2054
Epoch [    5/   10] | d_loss: 0.1098 | g_loss: 0.6242
Epoch [    5/   10] | d_loss: 0.1306 | g_loss: 0.8879
Epoch [    5/   10] | d_loss: 0.2407 | g_loss: 0.7988
Epoch [    5/   10] | d_loss: 0.4031 | g_loss: 0.6222
Epoch [    5/   10] | d_loss: 0.2614 | g_loss: 0.6332
Epoch [    5/   10] | d_loss: 0.1387 | g_loss: 1.0105
Epoch [    5/   10] | d_loss: 0.2142 | g_loss: 0.4708
Epoch [    5/   10] | d_loss: 0.1115 | g_loss: 1.0576
Epoch [    5/   10] | d_loss: 0.1529 | g_loss: 0.8014
Epoch [    5/   10] | d_loss: 0.2639 | g_loss: 0.5839
Epoch [    5/   10] | d_loss: 0.2179 | g_loss: 0.6905
Epoch [    5/   10] | d_loss: 0.1371 | g_loss: 1.0507
Epoch [    5/   10] | d_loss: 0.1301 | g_loss: 0.5313
Epoch [    5/   10] | d_loss: 0.1936 | g_loss: 0.5247
Epoch [    5/   10] | d_loss: 0.3599 | g_loss: 0.8976
Epoch [    5/   10] | d_loss: 0.0913 | g_loss: 0.6875
Epoch [    5/   10] | d_loss: 0.4385 | g_loss: 0.7273
```

```
Epoch [    5/   10] | d_loss: 0.2864 | g_loss: 0.7565
Epoch [    5/   10] | d_loss: 0.1661 | g_loss: 0.5950
Epoch [    5/   10] | d_loss: 0.1606 | g_loss: 1.0052
Epoch [    5/   10] | d_loss: 0.1656 | g_loss: 0.8155
Epoch [    5/   10] | d_loss: 0.2851 | g_loss: 1.1300
Epoch [    5/   10] | d_loss: 0.0831 | g_loss: 0.7821
Epoch [    5/   10] | d_loss: 0.1602 | g_loss: 1.0232
Epoch [    5/   10] | d_loss: 0.0790 | g_loss: 0.7284
Epoch [    5/   10] | d_loss: 0.2794 | g_loss: 0.8796
Epoch [    5/   10] | d_loss: 0.2048 | g_loss: 0.5011
Epoch [    5/   10] | d_loss: 0.1143 | g_loss: 1.2911
Epoch [    5/   10] | d_loss: 0.1659 | g_loss: 0.5930
Epoch [    5/   10] | d_loss: 0.1707 | g_loss: 0.8383
Epoch [    5/   10] | d_loss: 0.1793 | g_loss: 0.6010
Epoch [    5/   10] | d_loss: 0.1362 | g_loss: 0.8693
Epoch [    5/   10] | d_loss: 0.1568 | g_loss: 0.5550
Epoch [    5/   10] | d_loss: 0.2734 | g_loss: 0.8661
Epoch [    5/   10] | d_loss: 0.1622 | g_loss: 0.4789
Epoch [    5/   10] | d_loss: 0.1314 | g_loss: 0.6729
Epoch [    5/   10] | d_loss: 0.3175 | g_loss: 0.6853
Epoch [    5/   10] | d_loss: 0.1390 | g_loss: 0.5634
Epoch [    5/   10] | d_loss: 0.1285 | g_loss: 1.0927
Epoch [    5/   10] | d_loss: 0.0914 | g_loss: 0.8582
Epoch [    5/   10] | d_loss: 0.1788 | g_loss: 0.5591
Epoch [    5/   10] | d_loss: 0.2756 | g_loss: 0.6521
Epoch [    5/   10] | d_loss: 0.2118 | g_loss: 1.1018
Epoch [    5/   10] | d_loss: 0.2000 | g_loss: 0.7316
Epoch [    5/   10] | d_loss: 0.2291 | g_loss: 0.5718
Epoch [    5/   10] | d_loss: 0.0629 | g_loss: 0.6239
Epoch [    5/   10] | d_loss: 0.1874 | g_loss: 0.9876
Epoch [    5/   10] | d_loss: 0.4511 | g_loss: 0.7716
Epoch [    5/   10] | d_loss: 0.2207 | g_loss: 0.4953
Epoch [    5/   10] | d_loss: 0.3247 | g_loss: 1.1990
Epoch [    5/   10] | d_loss: 0.2556 | g_loss: 0.3931
Epoch [    5/   10] | d_loss: 0.2127 | g_loss: 0.5212
Epoch [    5/   10] | d_loss: 0.1764 | g_loss: 0.4561
Epoch [    5/   10] | d_loss: 0.1037 | g_loss: 0.6865
Epoch [    5/   10] | d_loss: 0.1750 | g_loss: 0.9300
Epoch [    5/   10] | d_loss: 0.2620 | g_loss: 0.6465
Epoch [    5/   10] | d_loss: 0.1826 | g_loss: 1.4926
Epoch [    5/   10] | d_loss: 0.2034 | g_loss: 0.7318
Epoch [    5/   10] | d_loss: 0.2959 | g_loss: 0.6150
Epoch [    5/   10] | d_loss: 0.2926 | g_loss: 0.2876
Epoch [    5/   10] | d_loss: 0.1476 | g_loss: 1.0592
Epoch [    5/   10] | d_loss: 0.1668 | g_loss: 0.5827
Epoch [    5/   10] | d_loss: 0.1988 | g_loss: 0.5553
Epoch [    5/   10] | d_loss: 0.2966 | g_loss: 0.7691
Epoch [    5/   10] | d_loss: 0.1879 | g_loss: 0.7601
```

```
Epoch [    5/   10] | d_loss: 0.1168 | g_loss: 0.5788
Epoch [    5/   10] | d_loss: 0.1001 | g_loss: 1.4578
Epoch [    5/   10] | d_loss: 0.1455 | g_loss: 0.8251
Epoch [    5/   10] | d_loss: 0.1144 | g_loss: 0.9778
Epoch [    5/   10] | d_loss: 0.2718 | g_loss: 0.6360
Epoch [    5/   10] | d_loss: 0.0961 | g_loss: 1.0583
Epoch [    5/   10] | d_loss: 0.1243 | g_loss: 0.5253
Epoch [    5/   10] | d_loss: 0.1657 | g_loss: 0.6802
Epoch [    5/   10] | d_loss: 0.1817 | g_loss: 0.6267
Epoch [    5/   10] | d_loss: 0.1758 | g_loss: 0.5368
Epoch [    5/   10] | d_loss: 0.1283 | g_loss: 0.8786
Epoch [    5/   10] | d_loss: 0.2233 | g_loss: 0.7069
Epoch [    5/   10] | d_loss: 0.1101 | g_loss: 0.8754
Epoch [    5/   10] | d_loss: 0.1427 | g_loss: 1.1811
Epoch [    5/   10] | d_loss: 0.1441 | g_loss: 0.7719
Epoch [    5/   10] | d_loss: 0.2454 | g_loss: 0.9270
Epoch [    5/   10] | d_loss: 0.2047 | g_loss: 0.5030
Epoch [    5/   10] | d_loss: 0.3016 | g_loss: 0.2074
Epoch [    5/   10] | d_loss: 0.2991 | g_loss: 0.8300
Epoch [    5/   10] | d_loss: 0.1565 | g_loss: 1.0887
Epoch [    5/   10] | d_loss: 0.2595 | g_loss: 0.6261
Epoch [    5/   10] | d_loss: 0.3179 | g_loss: 0.3387
Epoch [    5/   10] | d_loss: 0.2547 | g_loss: 0.8015
Epoch [    6/   10] | d_loss: 0.3158 | g_loss: 0.2627
Epoch [    6/   10] | d_loss: 0.4959 | g_loss: 0.5123
Epoch [    6/   10] | d_loss: 0.2004 | g_loss: 1.1703
Epoch [    6/   10] | d_loss: 0.0723 | g_loss: 1.0727
Epoch [    6/   10] | d_loss: 0.0642 | g_loss: 1.0750
Epoch [    6/   10] | d_loss: 0.2206 | g_loss: 0.3620
Epoch [    6/   10] | d_loss: 0.0984 | g_loss: 0.7246
Epoch [    6/   10] | d_loss: 0.1269 | g_loss: 0.8700
Epoch [    6/   10] | d_loss: 0.1184 | g_loss: 1.0403
Epoch [    6/   10] | d_loss: 0.2445 | g_loss: 0.7451
Epoch [    6/   10] | d_loss: 0.2100 | g_loss: 0.9143
Epoch [    6/   10] | d_loss: 0.2345 | g_loss: 0.3672
Epoch [    6/   10] | d_loss: 0.3265 | g_loss: 0.6467
Epoch [    6/   10] | d_loss: 0.1447 | g_loss: 0.2919
Epoch [    6/   10] | d_loss: 0.2972 | g_loss: 0.7911
Epoch [    6/   10] | d_loss: 0.1628 | g_loss: 1.1100
Epoch [    6/   10] | d_loss: 0.3050 | g_loss: 0.4559
Epoch [    6/   10] | d_loss: 0.1921 | g_loss: 0.5429
Epoch [    6/   10] | d_loss: 0.2377 | g_loss: 0.4386
Epoch [    6/   10] | d_loss: 0.1126 | g_loss: 1.0924
Epoch [    6/   10] | d_loss: 0.1309 | g_loss: 0.5662
Epoch [    6/   10] | d_loss: 0.0816 | g_loss: 0.7076
Epoch [    6/   10] | d_loss: 0.3325 | g_loss: 1.0033
Epoch [    6/   10] | d_loss: 0.2941 | g_loss: 0.7567
Epoch [    6/   10] | d_loss: 0.2514 | g_loss: 0.4130
```

```
Epoch [    6/   10] | d_loss: 0.6010 | g_loss: 1.3469
Epoch [    6/   10] | d_loss: 0.0697 | g_loss: 0.7611
Epoch [    6/   10] | d_loss: 0.2459 | g_loss: 0.8280
Epoch [    6/   10] | d_loss: 0.1732 | g_loss: 1.1399
Epoch [    6/   10] | d_loss: 0.2071 | g_loss: 0.6154
Epoch [    6/   10] | d_loss: 0.1150 | g_loss: 1.0817
Epoch [    6/   10] | d_loss: 0.1273 | g_loss: 0.8226
Epoch [    6/   10] | d_loss: 0.1232 | g_loss: 0.9650
Epoch [    6/   10] | d_loss: 0.3704 | g_loss: 0.3431
Epoch [    6/   10] | d_loss: 0.1581 | g_loss: 0.6082
Epoch [    6/   10] | d_loss: 0.2385 | g_loss: 1.1538
Epoch [    6/   10] | d_loss: 0.1423 | g_loss: 0.6595
Epoch [    6/   10] | d_loss: 0.1225 | g_loss: 0.9039
Epoch [    6/   10] | d_loss: 0.0793 | g_loss: 0.7533
Epoch [    6/   10] | d_loss: 0.2541 | g_loss: 0.8250
Epoch [    6/   10] | d_loss: 0.1428 | g_loss: 0.7963
Epoch [    6/   10] | d_loss: 0.1683 | g_loss: 0.5856
Epoch [    6/   10] | d_loss: 0.2142 | g_loss: 0.5564
Epoch [    6/   10] | d_loss: 0.2523 | g_loss: 0.5103
Epoch [    6/   10] | d_loss: 0.1887 | g_loss: 0.4583
Epoch [    6/   10] | d_loss: 0.1008 | g_loss: 1.0152
Epoch [    6/   10] | d_loss: 0.0998 | g_loss: 0.8952
Epoch [    6/   10] | d_loss: 0.1106 | g_loss: 0.7171
Epoch [    6/   10] | d_loss: 0.3298 | g_loss: 0.6048
Epoch [    6/   10] | d_loss: 0.2157 | g_loss: 0.6897
Epoch [    6/   10] | d_loss: 0.1449 | g_loss: 0.6521
Epoch [    6/   10] | d_loss: 0.3253 | g_loss: 0.4144
Epoch [    6/   10] | d_loss: 0.1207 | g_loss: 0.8115
Epoch [    6/   10] | d_loss: 0.1089 | g_loss: 0.8596
Epoch [    6/   10] | d_loss: 0.2256 | g_loss: 0.5652
Epoch [    6/   10] | d_loss: 0.1055 | g_loss: 0.7365
Epoch [    6/   10] | d_loss: 0.1551 | g_loss: 0.5929
Epoch [    6/   10] | d_loss: 0.0521 | g_loss: 0.6263
Epoch [    6/   10] | d_loss: 0.1659 | g_loss: 1.3364
Epoch [    6/   10] | d_loss: 0.0763 | g_loss: 0.8753
Epoch [    6/   10] | d_loss: 0.2238 | g_loss: 0.4577
Epoch [    6/   10] | d_loss: 0.1943 | g_loss: 0.5894
Epoch [    6/   10] | d_loss: 0.1684 | g_loss: 0.4842
Epoch [    6/   10] | d_loss: 0.1836 | g_loss: 0.5159
Epoch [    6/   10] | d_loss: 0.1964 | g_loss: 1.1723
Epoch [    6/   10] | d_loss: 0.3398 | g_loss: 0.6137
Epoch [    6/   10] | d_loss: 0.1893 | g_loss: 0.6728
Epoch [    6/   10] | d_loss: 0.3006 | g_loss: 0.8985
Epoch [    6/   10] | d_loss: 0.2400 | g_loss: 0.8649
Epoch [    6/   10] | d_loss: 0.1798 | g_loss: 0.6606
Epoch [    6/   10] | d_loss: 0.1383 | g_loss: 1.2096
Epoch [    6/   10] | d_loss: 0.1617 | g_loss: 0.6688
Epoch [    6/   10] | d_loss: 0.2501 | g_loss: 0.9480
```

```
Epoch [    6/   10] | d_loss: 0.1543 | g_loss: 0.8689
Epoch [    6/   10] | d_loss: 0.0905 | g_loss: 0.6488
Epoch [    6/   10] | d_loss: 0.1110 | g_loss: 1.8414
Epoch [    6/   10] | d_loss: 0.1581 | g_loss: 0.6354
Epoch [    6/   10] | d_loss: 0.1719 | g_loss: 0.5588
Epoch [    6/   10] | d_loss: 0.0640 | g_loss: 1.2909
Epoch [    6/   10] | d_loss: 0.1875 | g_loss: 0.6291
Epoch [    6/   10] | d_loss: 0.2695 | g_loss: 0.6975
Epoch [    6/   10] | d_loss: 0.2868 | g_loss: 0.8501
Epoch [    6/   10] | d_loss: 0.1298 | g_loss: 1.0008
Epoch [    6/   10] | d_loss: 0.1516 | g_loss: 0.6911
Epoch [    6/   10] | d_loss: 0.4096 | g_loss: 0.9042
Epoch [    6/   10] | d_loss: 0.0854 | g_loss: 0.8757
Epoch [    6/   10] | d_loss: 0.2215 | g_loss: 0.6278
Epoch [    6/   10] | d_loss: 0.3361 | g_loss: 1.1391
Epoch [    6/   10] | d_loss: 0.1531 | g_loss: 0.7405
Epoch [    6/   10] | d_loss: 0.3295 | g_loss: 0.5259
Epoch [    7/   10] | d_loss: 0.1385 | g_loss: 0.8274
Epoch [    7/   10] | d_loss: 0.1395 | g_loss: 1.2388
Epoch [    7/   10] | d_loss: 0.1364 | g_loss: 0.6002
Epoch [    7/   10] | d_loss: 0.1546 | g_loss: 0.9470
Epoch [    7/   10] | d_loss: 0.1578 | g_loss: 0.8266
Epoch [    7/   10] | d_loss: 0.3779 | g_loss: 0.9298
Epoch [    7/   10] | d_loss: 0.1374 | g_loss: 0.8827
Epoch [    7/   10] | d_loss: 0.3558 | g_loss: 0.8876
Epoch [    7/   10] | d_loss: 0.2201 | g_loss: 0.7260
Epoch [    7/   10] | d_loss: 0.0881 | g_loss: 0.4919
Epoch [    7/   10] | d_loss: 0.1405 | g_loss: 1.0348
Epoch [    7/   10] | d_loss: 0.2248 | g_loss: 0.8058
Epoch [    7/   10] | d_loss: 0.1891 | g_loss: 0.9303
Epoch [    7/   10] | d_loss: 0.1528 | g_loss: 0.5469
Epoch [    7/   10] | d_loss: 0.1722 | g_loss: 0.7586
Epoch [    7/   10] | d_loss: 0.1560 | g_loss: 0.6981
Epoch [    7/   10] | d_loss: 0.1793 | g_loss: 0.4502
Epoch [    7/   10] | d_loss: 0.2921 | g_loss: 0.7525
Epoch [    7/   10] | d_loss: 0.3928 | g_loss: 0.5127
Epoch [    7/   10] | d_loss: 0.3281 | g_loss: 0.7595
Epoch [    7/   10] | d_loss: 0.1293 | g_loss: 0.7405
Epoch [    7/   10] | d_loss: 0.0855 | g_loss: 1.1969
Epoch [    7/   10] | d_loss: 0.2233 | g_loss: 1.1102
Epoch [    7/   10] | d_loss: 0.0994 | g_loss: 0.6517
Epoch [    7/   10] | d_loss: 0.1631 | g_loss: 1.1098
Epoch [    7/   10] | d_loss: 0.2226 | g_loss: 0.3623
Epoch [    7/   10] | d_loss: 0.1547 | g_loss: 1.6136
Epoch [    7/   10] | d_loss: 0.1554 | g_loss: 0.8593
Epoch [    7/   10] | d_loss: 0.1643 | g_loss: 1.3281
Epoch [    7/   10] | d_loss: 0.1745 | g_loss: 0.8254
Epoch [    7/   10] | d_loss: 0.1185 | g_loss: 1.2988
```

```
Epoch [    7/   10] | d_loss: 0.4781 | g_loss: 0.6423
Epoch [    7/   10] | d_loss: 0.1356 | g_loss: 0.5968
Epoch [    7/   10] | d_loss: 0.1781 | g_loss: 0.6922
Epoch [    7/   10] | d_loss: 0.3763 | g_loss: 0.8573
Epoch [    7/   10] | d_loss: 0.1639 | g_loss: 0.9108
Epoch [    7/   10] | d_loss: 0.1843 | g_loss: 0.7089
Epoch [    7/   10] | d_loss: 0.2119 | g_loss: 1.1778
Epoch [    7/   10] | d_loss: 0.1262 | g_loss: 0.9331
Epoch [    7/   10] | d_loss: 0.2247 | g_loss: 0.4925
Epoch [    7/   10] | d_loss: 0.1804 | g_loss: 1.1096
Epoch [    7/   10] | d_loss: 0.2033 | g_loss: 0.5147
Epoch [    7/   10] | d_loss: 0.4341 | g_loss: 1.1348
Epoch [    7/   10] | d_loss: 0.1969 | g_loss: 0.8193
Epoch [    7/   10] | d_loss: 0.1894 | g_loss: 1.1997
Epoch [    7/   10] | d_loss: 0.2409 | g_loss: 0.8723
Epoch [    7/   10] | d_loss: 0.1391 | g_loss: 0.9729
Epoch [    7/   10] | d_loss: 0.0908 | g_loss: 0.8966
Epoch [    7/   10] | d_loss: 0.1415 | g_loss: 1.1076
Epoch [    7/   10] | d_loss: 0.1858 | g_loss: 0.8729
Epoch [    7/   10] | d_loss: 0.2637 | g_loss: 0.1935
Epoch [    7/   10] | d_loss: 0.1237 | g_loss: 0.5107
Epoch [    7/   10] | d_loss: 0.2608 | g_loss: 0.6994
Epoch [    7/   10] | d_loss: 0.1233 | g_loss: 0.7999
Epoch [    7/   10] | d_loss: 0.1467 | g_loss: 1.1489
Epoch [    7/   10] | d_loss: 0.2829 | g_loss: 0.4725
Epoch [    7/   10] | d_loss: 0.1392 | g_loss: 0.9044
Epoch [    7/   10] | d_loss: 0.2871 | g_loss: 0.6659
Epoch [    7/   10] | d_loss: 0.2913 | g_loss: 0.4447
Epoch [    7/   10] | d_loss: 0.2754 | g_loss: 0.4219
Epoch [    7/   10] | d_loss: 0.1105 | g_loss: 1.0876
Epoch [    7/   10] | d_loss: 0.1701 | g_loss: 0.7295
Epoch [    7/   10] | d_loss: 0.1019 | g_loss: 1.0651
Epoch [    7/   10] | d_loss: 0.1467 | g_loss: 0.6638
Epoch [    7/   10] | d_loss: 0.2019 | g_loss: 0.7584
Epoch [    7/   10] | d_loss: 0.0786 | g_loss: 0.8474
Epoch [    7/   10] | d_loss: 0.1150 | g_loss: 1.0097
Epoch [    7/   10] | d_loss: 0.2318 | g_loss: 0.6154
Epoch [    7/   10] | d_loss: 0.1219 | g_loss: 0.8068
Epoch [    7/   10] | d_loss: 0.1320 | g_loss: 1.0369
Epoch [    7/   10] | d_loss: 0.2122 | g_loss: 1.0922
Epoch [    7/   10] | d_loss: 0.2751 | g_loss: 0.7185
Epoch [    7/   10] | d_loss: 0.1783 | g_loss: 1.6686
Epoch [    7/   10] | d_loss: 0.2737 | g_loss: 0.8807
Epoch [    7/   10] | d_loss: 0.1699 | g_loss: 0.8336
Epoch [    7/   10] | d_loss: 0.1770 | g_loss: 0.7621
Epoch [    7/   10] | d_loss: 0.2066 | g_loss: 0.6356
Epoch [    7/   10] | d_loss: 0.1199 | g_loss: 0.8031
Epoch [    7/   10] | d_loss: 0.1993 | g_loss: 0.7349
```

```
Epoch [    7/   10] | d_loss: 0.2987 | g_loss: 0.8298
Epoch [    7/   10] | d_loss: 0.1924 | g_loss: 0.5520
Epoch [    7/   10] | d_loss: 0.1021 | g_loss: 0.8431
Epoch [    7/   10] | d_loss: 0.1217 | g_loss: 1.0262
Epoch [    7/   10] | d_loss: 0.1884 | g_loss: 0.7434
Epoch [    7/   10] | d_loss: 0.3342 | g_loss: 0.4955
Epoch [    7/   10] | d_loss: 0.0977 | g_loss: 1.0056
Epoch [    7/   10] | d_loss: 0.1773 | g_loss: 0.6437
Epoch [    7/   10] | d_loss: 0.1281 | g_loss: 0.3858
Epoch [    7/   10] | d_loss: 0.2777 | g_loss: 0.6247
Epoch [    7/   10] | d_loss: 0.1587 | g_loss: 0.8869
Epoch [    8/   10] | d_loss: 0.2830 | g_loss: 0.5272
Epoch [    8/   10] | d_loss: 0.1215 | g_loss: 0.8214
Epoch [    8/   10] | d_loss: 0.1530 | g_loss: 0.4799
Epoch [    8/   10] | d_loss: 0.3019 | g_loss: 0.3733
Epoch [    8/   10] | d_loss: 0.1798 | g_loss: 0.7873
Epoch [    8/   10] | d_loss: 0.2016 | g_loss: 0.9944
Epoch [    8/   10] | d_loss: 0.3275 | g_loss: 0.4837
Epoch [    8/   10] | d_loss: 0.2226 | g_loss: 0.6995
Epoch [    8/   10] | d_loss: 0.1334 | g_loss: 0.7088
Epoch [    8/   10] | d_loss: 0.1778 | g_loss: 0.7132
Epoch [    8/   10] | d_loss: 0.3167 | g_loss: 0.4873
Epoch [    8/   10] | d_loss: 0.0624 | g_loss: 0.8430
Epoch [    8/   10] | d_loss: 0.2055 | g_loss: 1.4048
Epoch [    8/   10] | d_loss: 0.0855 | g_loss: 0.8511
Epoch [    8/   10] | d_loss: 0.3090 | g_loss: 0.6850
Epoch [    8/   10] | d_loss: 0.1942 | g_loss: 0.4918
Epoch [    8/   10] | d_loss: 0.2951 | g_loss: 1.1671
Epoch [    8/   10] | d_loss: 0.0588 | g_loss: 0.8248
Epoch [    8/   10] | d_loss: 0.2112 | g_loss: 0.7197
Epoch [    8/   10] | d_loss: 0.0983 | g_loss: 1.1673
Epoch [    8/   10] | d_loss: 0.1748 | g_loss: 0.5113
Epoch [    8/   10] | d_loss: 0.1819 | g_loss: 0.6922
Epoch [    8/   10] | d_loss: 0.1174 | g_loss: 0.4914
Epoch [    8/   10] | d_loss: 0.2899 | g_loss: 1.1451
Epoch [    8/   10] | d_loss: 0.2099 | g_loss: 0.7644
Epoch [    8/   10] | d_loss: 0.1893 | g_loss: 0.6674
Epoch [    8/   10] | d_loss: 0.1389 | g_loss: 0.7280
Epoch [    8/   10] | d_loss: 0.5689 | g_loss: 0.1648
Epoch [    8/   10] | d_loss: 0.2147 | g_loss: 0.5911
Epoch [    8/   10] | d_loss: 0.1021 | g_loss: 0.9093
Epoch [    8/   10] | d_loss: 0.1905 | g_loss: 0.7978
Epoch [    8/   10] | d_loss: 0.2605 | g_loss: 0.6306
Epoch [    8/   10] | d_loss: 0.1614 | g_loss: 0.7908
Epoch [    8/   10] | d_loss: 0.1151 | g_loss: 0.8875
Epoch [    8/   10] | d_loss: 0.3020 | g_loss: 0.6308
Epoch [    8/   10] | d_loss: 0.1657 | g_loss: 0.8230
Epoch [    8/   10] | d_loss: 0.2401 | g_loss: 0.5260
```

```
Epoch [    8/   10] | d_loss: 0.1750 | g_loss: 1.0466
Epoch [    8/   10] | d_loss: 0.0873 | g_loss: 0.7665
Epoch [    8/   10] | d_loss: 0.1075 | g_loss: 1.5953
Epoch [    8/   10] | d_loss: 0.2779 | g_loss: 0.4505
Epoch [    8/   10] | d_loss: 0.1711 | g_loss: 1.1807
Epoch [    8/   10] | d_loss: 0.1408 | g_loss: 1.0568
Epoch [    8/   10] | d_loss: 0.1240 | g_loss: 0.8476
Epoch [    8/   10] | d_loss: 0.1709 | g_loss: 1.0848
Epoch [    8/   10] | d_loss: 0.3288 | g_loss: 0.5554
Epoch [    8/   10] | d_loss: 0.3065 | g_loss: 0.9607
Epoch [    8/   10] | d_loss: 0.1597 | g_loss: 0.6066
Epoch [    8/   10] | d_loss: 0.1172 | g_loss: 0.8507
Epoch [    8/   10] | d_loss: 0.2222 | g_loss: 0.6168
Epoch [    8/   10] | d_loss: 0.4205 | g_loss: 0.5520
Epoch [    8/   10] | d_loss: 0.1600 | g_loss: 0.7693
Epoch [    8/   10] | d_loss: 0.1791 | g_loss: 0.6656
Epoch [    8/   10] | d_loss: 0.1143 | g_loss: 0.8085
Epoch [    8/   10] | d_loss: 0.5420 | g_loss: 0.5565
Epoch [    8/   10] | d_loss: 0.3728 | g_loss: 0.4312
Epoch [    8/   10] | d_loss: 0.1376 | g_loss: 0.9102
Epoch [    8/   10] | d_loss: 0.1628 | g_loss: 0.8550
Epoch [    8/   10] | d_loss: 0.1685 | g_loss: 0.8326
Epoch [    8/   10] | d_loss: 0.1340 | g_loss: 1.0871
Epoch [    8/   10] | d_loss: 0.1568 | g_loss: 0.7696
Epoch [    8/   10] | d_loss: 0.1062 | g_loss: 0.7779
Epoch [    8/   10] | d_loss: 0.1111 | g_loss: 0.8803
Epoch [    8/   10] | d_loss: 0.3157 | g_loss: 0.6778
Epoch [    8/   10] | d_loss: 0.2942 | g_loss: 0.2482
Epoch [    8/   10] | d_loss: 0.2045 | g_loss: 0.8358
Epoch [    8/   10] | d_loss: 0.2589 | g_loss: 0.6240
Epoch [    8/   10] | d_loss: 0.0998 | g_loss: 0.8410
Epoch [    8/   10] | d_loss: 0.2280 | g_loss: 0.8234
Epoch [    8/   10] | d_loss: 0.0877 | g_loss: 1.2890
Epoch [    8/   10] | d_loss: 0.2438 | g_loss: 1.0254
Epoch [    8/   10] | d_loss: 0.2418 | g_loss: 0.9555
Epoch [    8/   10] | d_loss: 0.1468 | g_loss: 0.7944
Epoch [    8/   10] | d_loss: 0.2425 | g_loss: 0.5303
Epoch [    8/   10] | d_loss: 0.2102 | g_loss: 1.1475
Epoch [    8/   10] | d_loss: 0.2765 | g_loss: 0.4817
Epoch [    8/   10] | d_loss: 0.1295 | g_loss: 0.7041
Epoch [    8/   10] | d_loss: 0.1467 | g_loss: 0.9663
Epoch [    8/   10] | d_loss: 0.3805 | g_loss: 0.5545
Epoch [    8/   10] | d_loss: 0.2652 | g_loss: 0.7477
Epoch [    8/   10] | d_loss: 0.1267 | g_loss: 0.8706
Epoch [    8/   10] | d_loss: 0.1169 | g_loss: 1.0596
Epoch [    8/   10] | d_loss: 0.1409 | g_loss: 0.5338
Epoch [    8/   10] | d_loss: 0.1599 | g_loss: 0.9670
Epoch [    8/   10] | d_loss: 0.1228 | g_loss: 1.4058
```

```
Epoch [    8/   10] | d_loss: 0.3232 | g_loss: 0.4973
Epoch [    8/   10] | d_loss: 0.2261 | g_loss: 0.7411
Epoch [    8/   10] | d_loss: 0.1883 | g_loss: 0.8498
Epoch [    8/   10] | d_loss: 0.1766 | g_loss: 0.6516
Epoch [    8/   10] | d_loss: 0.1835 | g_loss: 0.8537
Epoch [    9/   10] | d_loss: 0.2518 | g_loss: 0.3888
Epoch [    9/   10] | d_loss: 0.1055 | g_loss: 1.0345
Epoch [    9/   10] | d_loss: 0.3171 | g_loss: 0.4341
Epoch [    9/   10] | d_loss: 0.3032 | g_loss: 0.5566
Epoch [    9/   10] | d_loss: 0.1556 | g_loss: 0.6965
Epoch [    9/   10] | d_loss: 0.2275 | g_loss: 0.4671
Epoch [    9/   10] | d_loss: 0.2434 | g_loss: 0.5687
Epoch [    9/   10] | d_loss: 0.6883 | g_loss: 0.4499
Epoch [    9/   10] | d_loss: 0.2590 | g_loss: 0.7662
Epoch [    9/   10] | d_loss: 0.2368 | g_loss: 0.7451
Epoch [    9/   10] | d_loss: 0.1843 | g_loss: 0.5647
Epoch [    9/   10] | d_loss: 0.1276 | g_loss: 1.1566
Epoch [    9/   10] | d_loss: 0.2946 | g_loss: 0.5720
Epoch [    9/   10] | d_loss: 0.1135 | g_loss: 0.7687
Epoch [    9/   10] | d_loss: 0.2309 | g_loss: 0.5548
Epoch [    9/   10] | d_loss: 0.1567 | g_loss: 0.5893
Epoch [    9/   10] | d_loss: 0.1078 | g_loss: 0.7713
Epoch [    9/   10] | d_loss: 0.2135 | g_loss: 0.9099
Epoch [    9/   10] | d_loss: 0.2098 | g_loss: 0.6397
Epoch [    9/   10] | d_loss: 0.1560 | g_loss: 1.2559
Epoch [    9/   10] | d_loss: 0.1024 | g_loss: 0.8338
Epoch [    9/   10] | d_loss: 0.3190 | g_loss: 0.5546
Epoch [    9/   10] | d_loss: 0.2517 | g_loss: 0.4099
Epoch [    9/   10] | d_loss: 0.0906 | g_loss: 0.8990
Epoch [    9/   10] | d_loss: 0.1616 | g_loss: 0.8387
Epoch [    9/   10] | d_loss: 0.1270 | g_loss: 0.4028
Epoch [    9/   10] | d_loss: 0.1015 | g_loss: 0.8243
Epoch [    9/   10] | d_loss: 0.2032 | g_loss: 0.5631
Epoch [    9/   10] | d_loss: 0.1714 | g_loss: 0.2421
Epoch [    9/   10] | d_loss: 0.1111 | g_loss: 1.1817
Epoch [    9/   10] | d_loss: 0.1870 | g_loss: 1.1836
Epoch [    9/   10] | d_loss: 0.0423 | g_loss: 1.1907
Epoch [    9/   10] | d_loss: 0.1483 | g_loss: 0.7542
Epoch [    9/   10] | d_loss: 0.1970 | g_loss: 1.1125
Epoch [    9/   10] | d_loss: 0.1334 | g_loss: 0.6865
Epoch [    9/   10] | d_loss: 0.2099 | g_loss: 0.7796
Epoch [    9/   10] | d_loss: 0.0918 | g_loss: 0.9243
Epoch [    9/   10] | d_loss: 0.1922 | g_loss: 0.6624
Epoch [    9/   10] | d_loss: 0.1535 | g_loss: 0.4625
Epoch [    9/   10] | d_loss: 0.1042 | g_loss: 0.8016
Epoch [    9/   10] | d_loss: 0.1258 | g_loss: 0.7736
Epoch [    9/   10] | d_loss: 0.1534 | g_loss: 0.7112
Epoch [    9/   10] | d_loss: 0.1676 | g_loss: 0.7471
```

```
Epoch [    9/   10] | d_loss: 0.0830 | g_loss: 0.9789
Epoch [    9/   10] | d_loss: 0.2609 | g_loss: 0.7458
Epoch [    9/   10] | d_loss: 0.2063 | g_loss: 1.1400
Epoch [    9/   10] | d_loss: 0.0868 | g_loss: 0.9758
Epoch [    9/   10] | d_loss: 0.1568 | g_loss: 1.0595
Epoch [    9/   10] | d_loss: 0.0851 | g_loss: 0.7437
Epoch [    9/   10] | d_loss: 0.1457 | g_loss: 1.1091
Epoch [    9/   10] | d_loss: 0.1405 | g_loss: 0.5236
Epoch [    9/   10] | d_loss: 0.0980 | g_loss: 0.7821
Epoch [    9/   10] | d_loss: 0.1330 | g_loss: 0.7897
Epoch [    9/   10] | d_loss: 0.1300 | g_loss: 0.7945
Epoch [    9/   10] | d_loss: 0.1679 | g_loss: 0.5463
Epoch [    9/   10] | d_loss: 0.1123 | g_loss: 0.9113
Epoch [    9/   10] | d_loss: 0.1114 | g_loss: 0.8900
Epoch [    9/   10] | d_loss: 0.0811 | g_loss: 0.8371
Epoch [    9/   10] | d_loss: 0.2414 | g_loss: 0.7579
Epoch [    9/   10] | d_loss: 0.1307 | g_loss: 1.2478
Epoch [    9/   10] | d_loss: 0.2421 | g_loss: 1.1101
Epoch [    9/   10] | d_loss: 0.2238 | g_loss: 0.9335
Epoch [    9/   10] | d_loss: 0.1080 | g_loss: 0.9935
Epoch [    9/   10] | d_loss: 0.2439 | g_loss: 0.7893
Epoch [    9/   10] | d_loss: 0.2005 | g_loss: 0.6268
Epoch [    9/   10] | d_loss: 0.3239 | g_loss: 0.4490
Epoch [    9/   10] | d_loss: 0.2965 | g_loss: 1.0735
Epoch [    9/   10] | d_loss: 0.1974 | g_loss: 1.0619
Epoch [    9/   10] | d_loss: 0.1032 | g_loss: 1.0186
Epoch [    9/   10] | d_loss: 0.1171 | g_loss: 0.5487
Epoch [    9/   10] | d_loss: 0.2596 | g_loss: 0.5271
Epoch [    9/   10] | d_loss: 0.2642 | g_loss: 0.7759
Epoch [    9/   10] | d_loss: 0.1902 | g_loss: 0.6500
Epoch [    9/   10] | d_loss: 0.4872 | g_loss: 0.4540
Epoch [    9/   10] | d_loss: 0.1467 | g_loss: 0.4946
Epoch [    9/   10] | d_loss: 0.1551 | g_loss: 1.0060
Epoch [    9/   10] | d_loss: 0.2785 | g_loss: 0.8039
Epoch [    9/   10] | d_loss: 0.2511 | g_loss: 0.7390
Epoch [    9/   10] | d_loss: 0.0939 | g_loss: 0.9591
Epoch [    9/   10] | d_loss: 0.1325 | g_loss: 1.2791
Epoch [    9/   10] | d_loss: 0.1880 | g_loss: 0.6625
Epoch [    9/   10] | d_loss: 0.2188 | g_loss: 0.6182
Epoch [    9/   10] | d_loss: 0.1780 | g_loss: 0.6294
Epoch [    9/   10] | d_loss: 0.1920 | g_loss: 0.4081
Epoch [    9/   10] | d_loss: 0.1494 | g_loss: 0.7603
Epoch [    9/   10] | d_loss: 0.3269 | g_loss: 0.8162
Epoch [    9/   10] | d_loss: 0.1215 | g_loss: 1.0773
Epoch [    9/   10] | d_loss: 0.1380 | g_loss: 0.5957
Epoch [    9/   10] | d_loss: 0.2353 | g_loss: 0.9644
Epoch [    9/   10] | d_loss: 0.1423 | g_loss: 1.7872
Epoch [   10/   10] | d_loss: 0.2639 | g_loss: 0.6325
```

```
Epoch [   10/   10] | d_loss: 0.1899 | g_loss: 0.5976
Epoch [   10/   10] | d_loss: 0.2060 | g_loss: 1.1658
Epoch [   10/   10] | d_loss: 0.2083 | g_loss: 1.2633
Epoch [   10/   10] | d_loss: 0.1942 | g_loss: 0.7517
Epoch [   10/   10] | d_loss: 0.1195 | g_loss: 0.4991
Epoch [   10/   10] | d_loss: 0.2320 | g_loss: 1.1770
Epoch [   10/   10] | d_loss: 0.2096 | g_loss: 0.2934
Epoch [   10/   10] | d_loss: 0.4172 | g_loss: 0.7366
Epoch [   10/   10] | d_loss: 0.1791 | g_loss: 1.0376
Epoch [   10/   10] | d_loss: 0.2247 | g_loss: 0.6173
Epoch [   10/   10] | d_loss: 0.1256 | g_loss: 0.7948
Epoch [   10/   10] | d_loss: 0.1881 | g_loss: 0.8696
Epoch [   10/   10] | d_loss: 0.1176 | g_loss: 0.7762
Epoch [   10/   10] | d_loss: 0.1910 | g_loss: 0.9401
Epoch [   10/   10] | d_loss: 0.2471 | g_loss: 0.5231
Epoch [   10/   10] | d_loss: 0.1309 | g_loss: 0.5900
Epoch [   10/   10] | d_loss: 0.2203 | g_loss: 0.7812
Epoch [   10/   10] | d_loss: 0.2118 | g_loss: 0.9183
Epoch [   10/   10] | d_loss: 0.1376 | g_loss: 0.7580
Epoch [   10/   10] | d_loss: 0.1280 | g_loss: 0.4405
Epoch [   10/   10] | d_loss: 0.0835 | g_loss: 0.7673
Epoch [   10/   10] | d_loss: 0.1619 | g_loss: 0.4090
Epoch [   10/   10] | d_loss: 0.3128 | g_loss: 0.9380
Epoch [   10/   10] | d_loss: 0.1384 | g_loss: 1.1379
Epoch [   10/   10] | d_loss: 0.0793 | g_loss: 0.9051
Epoch [   10/   10] | d_loss: 0.2776 | g_loss: 0.9072
Epoch [   10/   10] | d_loss: 0.1303 | g_loss: 0.6023
Epoch [   10/   10] | d_loss: 0.1650 | g_loss: 0.5314
Epoch [   10/   10] | d_loss: 0.1120 | g_loss: 0.6659
Epoch [   10/   10] | d_loss: 0.0916 | g_loss: 0.8271
Epoch [   10/   10] | d_loss: 0.1270 | g_loss: 0.8196
Epoch [   10/   10] | d_loss: 0.2047 | g_loss: 0.7992
Epoch [   10/   10] | d_loss: 0.3732 | g_loss: 0.5802
Epoch [   10/   10] | d_loss: 0.1762 | g_loss: 1.0259
Epoch [   10/   10] | d_loss: 0.3327 | g_loss: 0.6414
Epoch [   10/   10] | d_loss: 0.1149 | g_loss: 0.5672
Epoch [   10/   10] | d_loss: 0.2076 | g_loss: 0.9362
Epoch [   10/   10] | d_loss: 0.1088 | g_loss: 0.7051
Epoch [   10/   10] | d_loss: 0.3467 | g_loss: 0.5781
Epoch [   10/   10] | d_loss: 0.1728 | g_loss: 0.6199
Epoch [   10/   10] | d_loss: 0.0868 | g_loss: 0.6696
Epoch [   10/   10] | d_loss: 0.2924 | g_loss: 0.2529
Epoch [   10/   10] | d_loss: 0.1673 | g_loss: 0.4273
Epoch [   10/   10] | d_loss: 0.2031 | g_loss: 0.7905
Epoch [   10/   10] | d_loss: 0.1105 | g_loss: 1.1111
Epoch [   10/   10] | d_loss: 0.1646 | g_loss: 0.5381
Epoch [   10/   10] | d_loss: 0.1550 | g_loss: 0.6793
Epoch [   10/   10] | d_loss: 0.1812 | g_loss: 0.7790
```

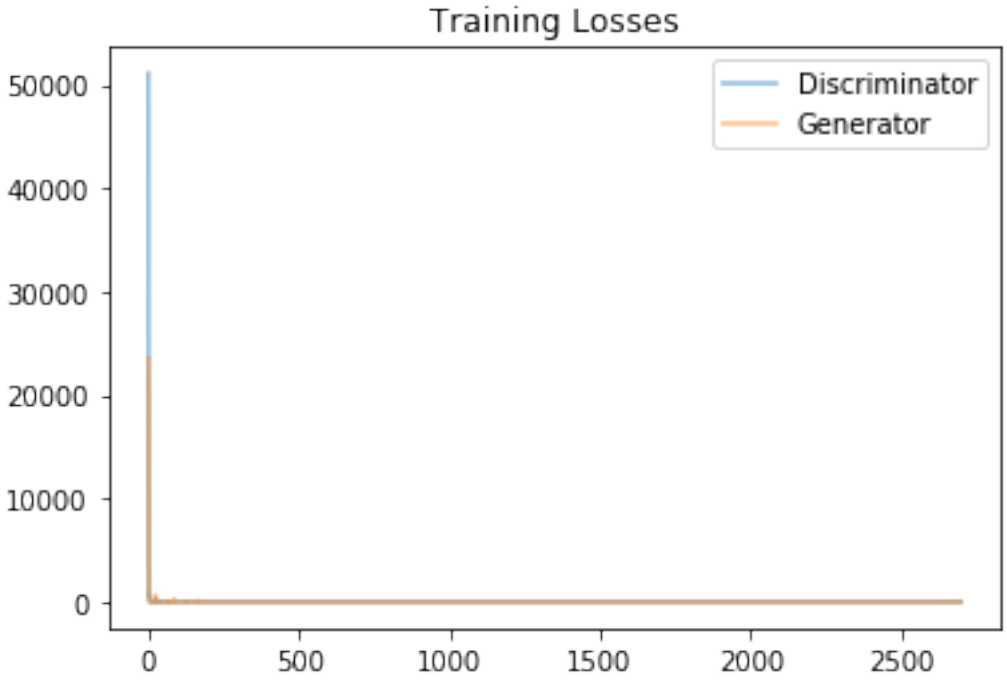
```
Epoch [   10/   10] | d_loss: 0.1825 | g_loss: 0.6480
Epoch [   10/   10] | d_loss: 0.1576 | g_loss: 0.5866
Epoch [   10/   10] | d_loss: 0.0672 | g_loss: 0.7830
Epoch [   10/   10] | d_loss: 0.0682 | g_loss: 0.9028
Epoch [   10/   10] | d_loss: 0.2121 | g_loss: 0.6101
Epoch [   10/   10] | d_loss: 0.0911 | g_loss: 0.7056
Epoch [   10/   10] | d_loss: 0.1735 | g_loss: 1.0121
Epoch [   10/   10] | d_loss: 0.2693 | g_loss: 0.6047
Epoch [   10/   10] | d_loss: 0.0686 | g_loss: 1.1836
Epoch [   10/   10] | d_loss: 0.1049 | g_loss: 1.0794
Epoch [   10/   10] | d_loss: 0.2496 | g_loss: 0.8625
Epoch [   10/   10] | d_loss: 0.3142 | g_loss: 0.4988
Epoch [   10/   10] | d_loss: 0.2109 | g_loss: 0.6246
Epoch [   10/   10] | d_loss: 0.3423 | g_loss: 0.4190
Epoch [   10/   10] | d_loss: 0.1441 | g_loss: 0.3572
Epoch [   10/   10] | d_loss: 0.1334 | g_loss: 0.5818
Epoch [   10/   10] | d_loss: 0.2551 | g_loss: 0.3828
Epoch [   10/   10] | d_loss: 0.2812 | g_loss: 0.5900
Epoch [   10/   10] | d_loss: 0.0819 | g_loss: 1.0423
Epoch [   10/   10] | d_loss: 0.0533 | g_loss: 0.8353
Epoch [   10/   10] | d_loss: 0.1274 | g_loss: 0.8721
Epoch [   10/   10] | d_loss: 0.0835 | g_loss: 1.0497
Epoch [   10/   10] | d_loss: 0.1556 | g_loss: 1.1912
Epoch [   10/   10] | d_loss: 0.1261 | g_loss: 0.3308
Epoch [   10/   10] | d_loss: 0.0782 | g_loss: 0.7864
Epoch [   10/   10] | d_loss: 0.0664 | g_loss: 1.0136
Epoch [   10/   10] | d_loss: 0.1261 | g_loss: 0.5472
Epoch [   10/   10] | d_loss: 0.3109 | g_loss: 0.9567
Epoch [   10/   10] | d_loss: 0.0955 | g_loss: 0.8497
Epoch [   10/   10] | d_loss: 0.0837 | g_loss: 1.6857
Epoch [   10/   10] | d_loss: 0.2048 | g_loss: 0.7186
Epoch [   10/   10] | d_loss: 0.1118 | g_loss: 0.9307
Epoch [   10/   10] | d_loss: 0.1397 | g_loss: 0.6554
Epoch [   10/   10] | d_loss: 0.1920 | g_loss: 0.8467
Epoch [   10/   10] | d_loss: 0.1182 | g_loss: 1.9999
Epoch [   10/   10] | d_loss: 0.0993 | g_loss: 0.9175
Epoch [   10/   10] | d_loss: 0.1585 | g_loss: 0.6710
Epoch [   10/   10] | d_loss: 0.2499 | g_loss: 0.6034
Epoch [   10/   10] | d_loss: 0.1298 | g_loss: 0.8980
Epoch [   10/   10] | d_loss: 0.1413 | g_loss: 0.9709
Epoch [   10/   10] | d_loss: 0.1940 | g_loss: 1.3362
```

2.8 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 0x7f6f18d7acf8>
```



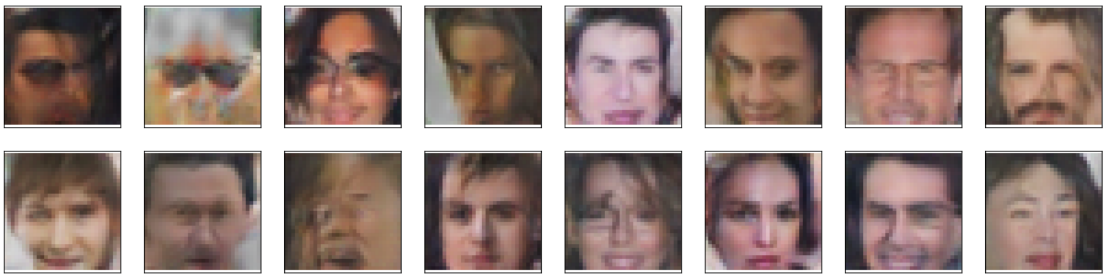
2.9 Generator samples from training

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

```
In [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)
```



2.9.1 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: I feel increasing more convolutional layers in discriminator and deconvoulational layers in generator could increase the performance. I could also train the project for more epochs. I also tried with SGD optimiser but I found , SGD has not better performance than Adam. So I keep using Adam optimiser.

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.