dlnd_face_generation

July 27, 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
```

```
In [54]: 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
             """
             transform = transforms.Compose([transforms.Resize(image_size),
                                             transforms.ToTensor()])
```

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

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

    data_loader = torch.utils.data.DataLoader(dataset=dataset, batch_size=batch_size, s

    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 [55]: # Define function hyperparameters
         batch_size = 64
         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)
```

```
In [6]: celeba_train_loader
```

```
Out[6]: <torch.utils.data.dataloader.DataLoader at 0x7f550a0da320>
```

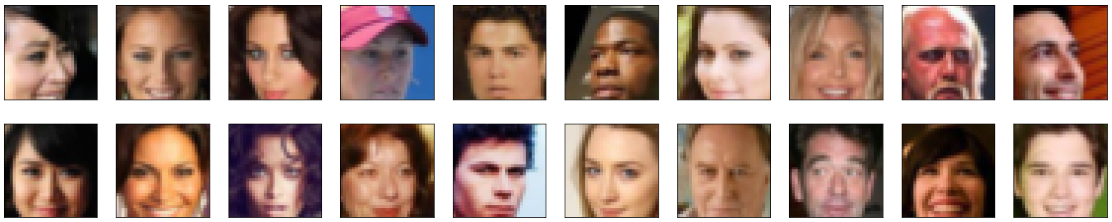
Next, you can view some images! You should seen square images of somewhat-centered faces.

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 [56]: # 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 [57]: # 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_mum, max_mum = feature_range
             x = x * (max_mum - min_mum) + min_mum
             return x
```

```
In [58]: """
         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())

         # imshow(scaled_img)
```

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

---

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 [10]: import torch.nn as nn
         import torch.nn.functional as F

In [59]: def conv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True):
             """Creates a convolutional layer, with onormalization.
             """
             layers = []
             conv_layers = nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
                                     kernel_size=kernel_size, stride=stride, padding=padding, bia

             layers.append(conv_layers)

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

In [81]: 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__()
                 self.conv_dim = conv_dim
                 # complete init function
                 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, batch_norm = False)
                 # 4x4 out

                 # final, fully-connected layer
         #         ?self.conv4 = conv(conv_dim*4*4*4, 1, 4, stride=1, batch_norm=False)
                 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
            x = F.leaky_relu(self.conv1(x), 0.2)
            x = F.leaky_relu(self.conv2(x), 0.2)
            x = F.leaky_relu(self.conv3(x), 0.2)

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

            # final output layer
            x = self.fc(x)
            return x


    """
    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 [82]: def deconv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True
             layers = []
             layers.append(nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride, pa
             if batch_norm:
                 layers.append(nn.BatchNorm2d(out_channels))
             return nn.Sequential(*layers)
```

```
In [83]: 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

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

            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
            """
            # define feedforward behavior
            x = self.fc(x)
            x = x.view(-1, self.conv_dim*4, 4, 4) # (batch_size, depth, 4, 4)

            x = F.relu(self.t_conv1(x))
            x = F.relu(self.t_conv2(x))


            # last layer: tanh activation instead of relu
            x = self.t_conv3(x)
            x = F.tanh(x)

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

```
Tests Passed
```

2.3 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 [84]: def weights_init_normal(m):
             """
             Applies initial weights to certain layers in a model .
             The weights are taken from a normal distribution
             with mean = 0, std dev = 0.02.
             :param m: A module or layer in a network
             """
             # classname will be something like:
             # `Conv`, `BatchNorm2d`, `Linear`, etc.
             classname = m.__class__.__name__

             # TODO: Apply initial weights to convolutional and linear layers

             if hasattr(m, 'weight') and (classname.find('Conv') != -1 or classname.find('Linear
                 nn.init.normal_(m.weight.data, 0.0, 0.02)
                 if hasattr(m.bias, 'data'):
                     nn.init.constant_(m.bias.data, 0.0)
         #     if classname.find('Linear') != -1:
         #         # apply a centered, uniform distribution to the weights
         #         m.weight.data.normal_(0.0, 0.02)
         #     if classname.find('Conv') != -1:
         #         # apply a centered, uniform distribution to the weights
         #         m.weight.data.normal_(0.0, 0.02)
         #     if classname.find('BatchNorm2d') != -1:
         #         # apply a centered, uniform distribution to the weights
         #         m.weight.data.normal_(0.0, 0.02)
```

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 [85]: """
         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 [86]: # 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)
  )
  (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 [87]:
```
"""
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 [88]: 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 [97]: import torch.optim as optim

         lr = 0.0005
         beta1=0.3
         beta2=0.999 # default value

         # 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 [98]: 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()

# 1. Train with real images

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

img_real = D(real_images)
the_real_loss = real_loss(img_real)

# 2. 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
img_fake = D(fake_images)
the_fake_loss = fake_loss(img_fake)
d_loss = the_real_loss + the_fake_loss

d_loss.backward()
d_optimizer.step()

# 2. Train the generator with an adversarial loss
g_optimizer.zero_grad()

# 1. 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!
the_fake = D(fake_images)
g_loss = real_loss(the_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 [104]: # set number of epochs
          n_epochs = 25


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

```
Epoch [    1/   25] | d_loss: 1.3224 | g_loss: 1.7945
Epoch [    1/   25] | d_loss: 1.2067 | g_loss: 1.9597
```

```
Epoch [    1/   25] | d_loss: 1.2808 | g_loss: 0.8919
Epoch [    1/   25] | d_loss: 1.3186 | g_loss: 0.6927
Epoch [    1/   25] | d_loss: 1.4569 | g_loss: 1.9841
Epoch [    1/   25] | d_loss: 0.9712 | g_loss: 1.1209
Epoch [    1/   25] | d_loss: 1.0303 | g_loss: 1.4423
Epoch [    1/   25] | d_loss: 1.0650 | g_loss: 1.0975
Epoch [    1/   25] | d_loss: 1.0579 | g_loss: 1.1430
Epoch [    1/   25] | d_loss: 0.9722 | g_loss: 1.5128
Epoch [    1/   25] | d_loss: 1.4057 | g_loss: 0.7196
Epoch [    1/   25] | d_loss: 1.1642 | g_loss: 1.7977
Epoch [    1/   25] | d_loss: 1.1210 | g_loss: 1.0927
Epoch [    1/   25] | d_loss: 1.4886 | g_loss: 2.4212
Epoch [    1/   25] | d_loss: 1.2285 | g_loss: 1.6128
Epoch [    1/   25] | d_loss: 1.2489 | g_loss: 0.5743
Epoch [    1/   25] | d_loss: 0.8787 | g_loss: 1.0831
Epoch [    1/   25] | d_loss: 0.9834 | g_loss: 1.5673
Epoch [    1/   25] | d_loss: 1.0172 | g_loss: 1.2111
Epoch [    1/   25] | d_loss: 1.1474 | g_loss: 1.9503
Epoch [    1/   25] | d_loss: 1.0811 | g_loss: 1.7227
Epoch [    1/   25] | d_loss: 1.0886 | g_loss: 1.4207
Epoch [    1/   25] | d_loss: 1.1360 | g_loss: 1.9629
Epoch [    1/   25] | d_loss: 0.9686 | g_loss: 1.3053
Epoch [    1/   25] | d_loss: 1.4836 | g_loss: 2.4028
Epoch [    1/   25] | d_loss: 1.0127 | g_loss: 1.3357
Epoch [    1/   25] | d_loss: 1.0126 | g_loss: 1.1162
Epoch [    1/   25] | d_loss: 0.9278 | g_loss: 1.4437
Epoch [    1/   25] | d_loss: 0.9413 | g_loss: 1.3536
Epoch [    2/   25] | d_loss: 0.9001 | g_loss: 1.2449
Epoch [    2/   25] | d_loss: 0.7309 | g_loss: 1.5911
Epoch [    2/   25] | d_loss: 0.8014 | g_loss: 1.9567
Epoch [    2/   25] | d_loss: 0.9403 | g_loss: 1.1963
Epoch [    2/   25] | d_loss: 1.0929 | g_loss: 1.2056
Epoch [    2/   25] | d_loss: 1.0049 | g_loss: 1.5775
Epoch [    2/   25] | d_loss: 1.0544 | g_loss: 0.8798
Epoch [    2/   25] | d_loss: 1.0259 | g_loss: 1.1660
Epoch [    2/   25] | d_loss: 1.1267 | g_loss: 0.9626
Epoch [    2/   25] | d_loss: 0.9463 | g_loss: 1.3275
Epoch [    2/   25] | d_loss: 1.1180 | g_loss: 1.7909
Epoch [    2/   25] | d_loss: 1.2220 | g_loss: 0.8616
Epoch [    2/   25] | d_loss: 0.9140 | g_loss: 1.8549
Epoch [    2/   25] | d_loss: 0.7363 | g_loss: 1.9312
Epoch [    2/   25] | d_loss: 1.0451 | g_loss: 2.0558
Epoch [    2/   25] | d_loss: 0.9190 | g_loss: 1.0109
Epoch [    2/   25] | d_loss: 0.7919 | g_loss: 1.9791
Epoch [    2/   25] | d_loss: 1.4281 | g_loss: 0.3527
Epoch [    2/   25] | d_loss: 0.8140 | g_loss: 1.3211
Epoch [    2/   25] | d_loss: 0.8850 | g_loss: 1.5322
Epoch [    2/   25] | d_loss: 0.8550 | g_loss: 1.4036
```

```
Epoch [    2/   25] | d_loss: 0.7646 | g_loss: 2.0198
Epoch [    2/   25] | d_loss: 0.8438 | g_loss: 1.3938
Epoch [    2/   25] | d_loss: 0.7757 | g_loss: 2.0302
Epoch [    2/   25] | d_loss: 0.9158 | g_loss: 1.7773
Epoch [    2/   25] | d_loss: 0.9431 | g_loss: 2.1555
Epoch [    2/   25] | d_loss: 1.1650 | g_loss: 0.6625
Epoch [    2/   25] | d_loss: 0.7899 | g_loss: 2.1726
Epoch [    2/   25] | d_loss: 1.1114 | g_loss: 2.6586
Epoch [    3/   25] | d_loss: 1.0972 | g_loss: 1.7165
Epoch [    3/   25] | d_loss: 1.0141 | g_loss: 2.5228
Epoch [    3/   25] | d_loss: 0.8571 | g_loss: 1.8775
Epoch [    3/   25] | d_loss: 1.1214 | g_loss: 2.3486
Epoch [    3/   25] | d_loss: 1.2610 | g_loss: 0.5458
Epoch [    3/   25] | d_loss: 2.5922 | g_loss: 0.4998
Epoch [    3/   25] | d_loss: 0.9783 | g_loss: 2.3577
Epoch [    3/   25] | d_loss: 0.7684 | g_loss: 1.4425
Epoch [    3/   25] | d_loss: 0.8775 | g_loss: 1.4085
Epoch [    3/   25] | d_loss: 0.9968 | g_loss: 1.4545
Epoch [    3/   25] | d_loss: 0.8240 | g_loss: 2.1955
Epoch [    3/   25] | d_loss: 1.2673 | g_loss: 2.2116
Epoch [    3/   25] | d_loss: 0.9089 | g_loss: 1.6314
Epoch [    3/   25] | d_loss: 0.8682 | g_loss: 2.0147
Epoch [    3/   25] | d_loss: 0.7976 | g_loss: 1.6071
Epoch [    3/   25] | d_loss: 0.9728 | g_loss: 1.8782
Epoch [    3/   25] | d_loss: 0.9569 | g_loss: 1.3374
Epoch [    3/   25] | d_loss: 1.1188 | g_loss: 2.5079
Epoch [    3/   25] | d_loss: 0.8544 | g_loss: 1.6994
Epoch [    3/   25] | d_loss: 0.9541 | g_loss: 2.1330
Epoch [    3/   25] | d_loss: 0.8324 | g_loss: 2.0198
Epoch [    3/   25] | d_loss: 1.0632 | g_loss: 2.3242
Epoch [    3/   25] | d_loss: 0.8795 | g_loss: 2.3237
Epoch [    3/   25] | d_loss: 0.8741 | g_loss: 1.8696
Epoch [    3/   25] | d_loss: 0.7589 | g_loss: 1.9652
Epoch [    3/   25] | d_loss: 0.7711 | g_loss: 2.4629
Epoch [    3/   25] | d_loss: 0.7760 | g_loss: 2.6254
Epoch [    3/   25] | d_loss: 0.7255 | g_loss: 2.9287
Epoch [    3/   25] | d_loss: 0.8766 | g_loss: 1.7550
Epoch [    4/   25] | d_loss: 1.0831 | g_loss: 1.5575
Epoch [    4/   25] | d_loss: 1.0142 | g_loss: 0.9153
Epoch [    4/   25] | d_loss: 0.8365 | g_loss: 1.4881
Epoch [    4/   25] | d_loss: 0.6476 | g_loss: 2.3232
Epoch [    4/   25] | d_loss: 0.9045 | g_loss: 1.4097
Epoch [    4/   25] | d_loss: 1.4063 | g_loss: 0.7984
Epoch [    4/   25] | d_loss: 1.0070 | g_loss: 1.3247
Epoch [    4/   25] | d_loss: 0.9686 | g_loss: 0.8035
Epoch [    4/   25] | d_loss: 0.9820 | g_loss: 1.9409
Epoch [    4/   25] | d_loss: 0.7509 | g_loss: 1.8716
Epoch [    4/   25] | d_loss: 0.8070 | g_loss: 1.2510
```

```
Epoch [    4/   25] | d_loss: 0.9841 | g_loss: 2.4305
Epoch [    4/   25] | d_loss: 0.9620 | g_loss: 1.1323
Epoch [    4/   25] | d_loss: 0.8722 | g_loss: 2.0515
Epoch [    4/   25] | d_loss: 1.0788 | g_loss: 1.0199
Epoch [    4/   25] | d_loss: 1.0547 | g_loss: 2.1554
Epoch [    4/   25] | d_loss: 0.5843 | g_loss: 2.0384
Epoch [    4/   25] | d_loss: 0.6528 | g_loss: 2.2050
Epoch [    4/   25] | d_loss: 0.8876 | g_loss: 1.1717
Epoch [    4/   25] | d_loss: 0.6892 | g_loss: 2.4319
Epoch [    4/   25] | d_loss: 1.1132 | g_loss: 3.5846
Epoch [    4/   25] | d_loss: 0.6675 | g_loss: 2.4004
Epoch [    4/   25] | d_loss: 1.4943 | g_loss: 0.6479
Epoch [    4/   25] | d_loss: 0.9402 | g_loss: 1.1314
Epoch [    4/   25] | d_loss: 0.6619 | g_loss: 2.2351
Epoch [    4/   25] | d_loss: 1.6277 | g_loss: 1.4622
Epoch [    4/   25] | d_loss: 1.1746 | g_loss: 3.1993
Epoch [    4/   25] | d_loss: 0.7359 | g_loss: 2.1440
Epoch [    4/   25] | d_loss: 1.6602 | g_loss: 1.1188
Epoch [    5/   25] | d_loss: 1.2689 | g_loss: 0.7366
Epoch [    5/   25] | d_loss: 1.0250 | g_loss: 1.0175
Epoch [    5/   25] | d_loss: 1.3164 | g_loss: 1.0823
Epoch [    5/   25] | d_loss: 1.1163 | g_loss: 1.0096
Epoch [    5/   25] | d_loss: 0.8803 | g_loss: 1.7759
Epoch [    5/   25] | d_loss: 1.0314 | g_loss: 1.4347
Epoch [    5/   25] | d_loss: 0.6584 | g_loss: 2.5280
Epoch [    5/   25] | d_loss: 0.7013 | g_loss: 2.3332
Epoch [    5/   25] | d_loss: 0.7131 | g_loss: 2.6398
Epoch [    5/   25] | d_loss: 0.7210 | g_loss: 3.0478
Epoch [    5/   25] | d_loss: 0.9690 | g_loss: 0.8212
Epoch [    5/   25] | d_loss: 0.8985 | g_loss: 2.2607
Epoch [    5/   25] | d_loss: 0.6073 | g_loss: 2.0451
Epoch [    5/   25] | d_loss: 0.7901 | g_loss: 1.9611
Epoch [    5/   25] | d_loss: 1.0675 | g_loss: 1.4172
Epoch [    5/   25] | d_loss: 0.6392 | g_loss: 2.0430
Epoch [    5/   25] | d_loss: 1.1612 | g_loss: 2.7944
Epoch [    5/   25] | d_loss: 1.4057 | g_loss: 0.4653
Epoch [    5/   25] | d_loss: 0.7325 | g_loss: 1.9711
Epoch [    5/   25] | d_loss: 0.4050 | g_loss: 2.9230
Epoch [    5/   25] | d_loss: 0.9460 | g_loss: 2.0799
Epoch [    5/   25] | d_loss: 0.5612 | g_loss: 2.5274
Epoch [    5/   25] | d_loss: 0.7427 | g_loss: 1.7110
Epoch [    5/   25] | d_loss: 1.0554 | g_loss: 1.0590
Epoch [    5/   25] | d_loss: 0.6718 | g_loss: 1.9669
Epoch [    5/   25] | d_loss: 0.8398 | g_loss: 1.7281
Epoch [    5/   25] | d_loss: 0.7957 | g_loss: 2.1787
Epoch [    5/   25] | d_loss: 0.7667 | g_loss: 1.3290
Epoch [    5/   25] | d_loss: 0.9807 | g_loss: 1.2771
Epoch [    6/   25] | d_loss: 1.1792 | g_loss: 1.0938
```

```
Epoch [    6/   25] | d_loss: 0.6580 | g_loss: 2.4267
Epoch [    6/   25] | d_loss: 0.9308 | g_loss: 1.2594
Epoch [    6/   25] | d_loss: 0.5683 | g_loss: 2.0245
Epoch [    6/   25] | d_loss: 0.7718 | g_loss: 1.4332
Epoch [    6/   25] | d_loss: 0.4786 | g_loss: 2.6295
Epoch [    6/   25] | d_loss: 1.3024 | g_loss: 1.2195
Epoch [    6/   25] | d_loss: 0.5599 | g_loss: 1.6705
Epoch [    6/   25] | d_loss: 0.7223 | g_loss: 1.9725
Epoch [    6/   25] | d_loss: 0.9992 | g_loss: 2.9368
Epoch [    6/   25] | d_loss: 0.7842 | g_loss: 1.4538
Epoch [    6/   25] | d_loss: 0.7683 | g_loss: 1.8183
Epoch [    6/   25] | d_loss: 0.4775 | g_loss: 2.4214
Epoch [    6/   25] | d_loss: 0.5329 | g_loss: 2.1226
Epoch [    6/   25] | d_loss: 0.7835 | g_loss: 1.9402
Epoch [    6/   25] | d_loss: 0.9299 | g_loss: 2.3530
Epoch [    6/   25] | d_loss: 1.2980 | g_loss: 1.6836
Epoch [    6/   25] | d_loss: 0.5269 | g_loss: 2.5642
Epoch [    6/   25] | d_loss: 0.7050 | g_loss: 2.2691
Epoch [    6/   25] | d_loss: 1.3012 | g_loss: 0.7287
Epoch [    6/   25] | d_loss: 0.8881 | g_loss: 1.4727
Epoch [    6/   25] | d_loss: 1.8324 | g_loss: 3.3346
Epoch [    6/   25] | d_loss: 0.5809 | g_loss: 2.9552
Epoch [    6/   25] | d_loss: 0.8332 | g_loss: 1.9620
Epoch [    6/   25] | d_loss: 0.4242 | g_loss: 2.3370
Epoch [    6/   25] | d_loss: 0.3876 | g_loss: 2.9865
Epoch [    6/   25] | d_loss: 0.5254 | g_loss: 2.1742
Epoch [    6/   25] | d_loss: 0.9685 | g_loss: 3.4392
Epoch [    6/   25] | d_loss: 0.5409 | g_loss: 3.5997
Epoch [    7/   25] | d_loss: 0.7437 | g_loss: 3.5776
Epoch [    7/   25] | d_loss: 0.9610 | g_loss: 2.7316
Epoch [    7/   25] | d_loss: 0.4984 | g_loss: 2.7956
Epoch [    7/   25] | d_loss: 0.4531 | g_loss: 3.7499
Epoch [    7/   25] | d_loss: 1.9318 | g_loss: 4.6667
Epoch [    7/   25] | d_loss: 0.6546 | g_loss: 2.1840
Epoch [    7/   25] | d_loss: 0.6108 | g_loss: 2.4436
Epoch [    7/   25] | d_loss: 0.4081 | g_loss: 2.8425
Epoch [    7/   25] | d_loss: 0.7117 | g_loss: 1.4366
Epoch [    7/   25] | d_loss: 1.1100 | g_loss: 0.5794
Epoch [    7/   25] | d_loss: 0.8595 | g_loss: 3.3110
Epoch [    7/   25] | d_loss: 0.6328 | g_loss: 1.6328
Epoch [    7/   25] | d_loss: 1.1840 | g_loss: 0.6509
Epoch [    7/   25] | d_loss: 1.6071 | g_loss: 3.0516
Epoch [    7/   25] | d_loss: 1.3163 | g_loss: 0.8444
Epoch [    7/   25] | d_loss: 0.8582 | g_loss: 1.9493
Epoch [    7/   25] | d_loss: 0.7781 | g_loss: 2.7456
Epoch [    7/   25] | d_loss: 1.3507 | g_loss: 0.6299
Epoch [    7/   25] | d_loss: 1.1062 | g_loss: 3.5373
Epoch [    7/   25] | d_loss: 0.6312 | g_loss: 3.3355
```

```
Epoch [    7/   25] | d_loss: 0.4368 | g_loss: 2.6112
Epoch [    7/   25] | d_loss: 2.1935 | g_loss: 0.4815
Epoch [    7/   25] | d_loss: 0.4238 | g_loss: 2.4323
Epoch [    7/   25] | d_loss: 0.5043 | g_loss: 3.7120
Epoch [    7/   25] | d_loss: 1.1264 | g_loss: 0.9735
Epoch [    7/   25] | d_loss: 0.8790 | g_loss: 1.7364
Epoch [    7/   25] | d_loss: 0.9281 | g_loss: 2.1627
Epoch [    7/   25] | d_loss: 0.8930 | g_loss: 2.6168
Epoch [    7/   25] | d_loss: 0.5616 | g_loss: 2.0554
Epoch [    8/   25] | d_loss: 0.4895 | g_loss: 2.7601
Epoch [    8/   25] | d_loss: 1.5438 | g_loss: 1.0488
Epoch [    8/   25] | d_loss: 0.6820 | g_loss: 2.0958
Epoch [    8/   25] | d_loss: 0.5221 | g_loss: 3.2265
Epoch [    8/   25] | d_loss: 1.0660 | g_loss: 1.6699
Epoch [    8/   25] | d_loss: 0.9020 | g_loss: 2.7364
Epoch [    8/   25] | d_loss: 0.6572 | g_loss: 1.8820
Epoch [    8/   25] | d_loss: 0.7258 | g_loss: 2.4374
Epoch [    8/   25] | d_loss: 0.7190 | g_loss: 3.3956
Epoch [    8/   25] | d_loss: 0.7674 | g_loss: 1.6064
Epoch [    8/   25] | d_loss: 1.1084 | g_loss: 1.0321
Epoch [    8/   25] | d_loss: 0.8232 | g_loss: 2.5703
Epoch [    8/   25] | d_loss: 0.8296 | g_loss: 1.3722
Epoch [    8/   25] | d_loss: 0.5597 | g_loss: 2.3714
Epoch [    8/   25] | d_loss: 0.9570 | g_loss: 3.9136
Epoch [    8/   25] | d_loss: 0.4669 | g_loss: 3.8070
Epoch [    8/   25] | d_loss: 0.9383 | g_loss: 3.1419
Epoch [    8/   25] | d_loss: 0.8169 | g_loss: 1.5481
Epoch [    8/   25] | d_loss: 0.4651 | g_loss: 2.9823
Epoch [    8/   25] | d_loss: 0.4271 | g_loss: 2.5208
Epoch [    8/   25] | d_loss: 0.3694 | g_loss: 3.9017
Epoch [    8/   25] | d_loss: 0.7208 | g_loss: 2.4869
Epoch [    8/   25] | d_loss: 0.9978 | g_loss: 3.4897
Epoch [    8/   25] | d_loss: 0.2672 | g_loss: 2.9104
Epoch [    8/   25] | d_loss: 0.7970 | g_loss: 3.4869
Epoch [    8/   25] | d_loss: 0.6889 | g_loss: 3.4449
Epoch [    8/   25] | d_loss: 1.0111 | g_loss: 3.7416
Epoch [    8/   25] | d_loss: 1.0424 | g_loss: 2.4244
Epoch [    8/   25] | d_loss: 0.5638 | g_loss: 3.1369
Epoch [    9/   25] | d_loss: 0.8951 | g_loss: 2.2968
Epoch [    9/   25] | d_loss: 0.4911 | g_loss: 2.9843
Epoch [    9/   25] | d_loss: 0.9408 | g_loss: 2.3306
Epoch [    9/   25] | d_loss: 1.0166 | g_loss: 2.1733
Epoch [    9/   25] | d_loss: 1.0281 | g_loss: 3.2834
Epoch [    9/   25] | d_loss: 0.4721 | g_loss: 4.2102
Epoch [    9/   25] | d_loss: 0.8181 | g_loss: 2.4817
Epoch [    9/   25] | d_loss: 0.6277 | g_loss: 3.4045
Epoch [    9/   25] | d_loss: 1.1795 | g_loss: 3.7972
Epoch [    9/   25] | d_loss: 0.5851 | g_loss: 1.9979
```

```
Epoch [    9/   25] | d_loss: 0.5379 | g_loss: 2.9928
Epoch [    9/   25] | d_loss: 0.5853 | g_loss: 3.3485
Epoch [    9/   25] | d_loss: 0.4091 | g_loss: 2.9270
Epoch [    9/   25] | d_loss: 0.4806 | g_loss: 3.8282
Epoch [    9/   25] | d_loss: 0.8251 | g_loss: 2.3748
Epoch [    9/   25] | d_loss: 0.8302 | g_loss: 2.9938
Epoch [    9/   25] | d_loss: 0.7289 | g_loss: 2.2750
Epoch [    9/   25] | d_loss: 0.4940 | g_loss: 2.8615
Epoch [    9/   25] | d_loss: 0.4294 | g_loss: 3.2232
Epoch [    9/   25] | d_loss: 0.4963 | g_loss: 3.1413
Epoch [    9/   25] | d_loss: 1.2986 | g_loss: 2.2535
Epoch [    9/   25] | d_loss: 0.9671 | g_loss: 2.6557
Epoch [    9/   25] | d_loss: 0.7108 | g_loss: 3.2600
Epoch [    9/   25] | d_loss: 0.4557 | g_loss: 4.3068
Epoch [    9/   25] | d_loss: 0.9506 | g_loss: 2.2191
Epoch [    9/   25] | d_loss: 0.4891 | g_loss: 3.0513
Epoch [    9/   25] | d_loss: 0.5922 | g_loss: 1.4702
Epoch [    9/   25] | d_loss: 0.8996 | g_loss: 4.2142
Epoch [    9/   25] | d_loss: 0.4640 | g_loss: 3.0067
Epoch [   10/   25] | d_loss: 1.0859 | g_loss: 3.1877
Epoch [   10/   25] | d_loss: 2.7158 | g_loss: 4.0206
Epoch [   10/   25] | d_loss: 0.4266 | g_loss: 3.4244
Epoch [   10/   25] | d_loss: 0.7143 | g_loss: 1.3494
Epoch [   10/   25] | d_loss: 0.3869 | g_loss: 3.1906
Epoch [   10/   25] | d_loss: 0.5862 | g_loss: 3.4599
Epoch [   10/   25] | d_loss: 0.9617 | g_loss: 3.9414
Epoch [   10/   25] | d_loss: 0.7193 | g_loss: 2.9607
Epoch [   10/   25] | d_loss: 0.4125 | g_loss: 3.1189
Epoch [   10/   25] | d_loss: 0.5680 | g_loss: 5.1510
Epoch [   10/   25] | d_loss: 1.0532 | g_loss: 1.9323
Epoch [   10/   25] | d_loss: 0.4179 | g_loss: 3.3130
Epoch [   10/   25] | d_loss: 1.2033 | g_loss: 4.2260
Epoch [   10/   25] | d_loss: 0.5314 | g_loss: 3.1393
Epoch [   10/   25] | d_loss: 0.3298 | g_loss: 3.5619
Epoch [   10/   25] | d_loss: 0.9750 | g_loss: 1.8018
Epoch [   10/   25] | d_loss: 0.4320 | g_loss: 3.8762
Epoch [   10/   25] | d_loss: 1.0164 | g_loss: 1.0815
Epoch [   10/   25] | d_loss: 1.6219 | g_loss: 2.8562
Epoch [   10/   25] | d_loss: 0.3825 | g_loss: 3.1119
Epoch [   10/   25] | d_loss: 0.5119 | g_loss: 3.0684
Epoch [   10/   25] | d_loss: 0.8502 | g_loss: 3.2630
Epoch [   10/   25] | d_loss: 0.3381 | g_loss: 4.0727
Epoch [   10/   25] | d_loss: 0.8249 | g_loss: 4.1866
Epoch [   10/   25] | d_loss: 0.4270 | g_loss: 3.6523
Epoch [   10/   25] | d_loss: 0.7584 | g_loss: 1.9482
Epoch [   10/   25] | d_loss: 0.6701 | g_loss: 2.0114
Epoch [   10/   25] | d_loss: 0.8382 | g_loss: 2.5523
Epoch [   10/   25] | d_loss: 0.3734 | g_loss: 3.4183
```

```
Epoch [   11/   25] | d_loss: 0.4641 | g_loss: 3.6765
Epoch [   11/   25] | d_loss: 0.6241 | g_loss: 2.3287
Epoch [   11/   25] | d_loss: 0.5994 | g_loss: 4.6144
Epoch [   11/   25] | d_loss: 0.5457 | g_loss: 2.9184
Epoch [   11/   25] | d_loss: 0.3331 | g_loss: 2.5244
Epoch [   11/   25] | d_loss: 0.5803 | g_loss: 2.1346
Epoch [   11/   25] | d_loss: 0.9817 | g_loss: 2.2451
Epoch [   11/   25] | d_loss: 0.5852 | g_loss: 1.6954
Epoch [   11/   25] | d_loss: 1.4257 | g_loss: 1.2366
Epoch [   11/   25] | d_loss: 1.0413 | g_loss: 3.0451
Epoch [   11/   25] | d_loss: 0.2931 | g_loss: 3.1472
Epoch [   11/   25] | d_loss: 1.1297 | g_loss: 1.8596
Epoch [   11/   25] | d_loss: 0.5649 | g_loss: 2.8571
Epoch [   11/   25] | d_loss: 0.7043 | g_loss: 2.7052
Epoch [   11/   25] | d_loss: 1.2742 | g_loss: 3.3141
Epoch [   11/   25] | d_loss: 0.9534 | g_loss: 3.2300
Epoch [   11/   25] | d_loss: 0.5992 | g_loss: 2.6029
Epoch [   11/   25] | d_loss: 0.7148 | g_loss: 2.6658
Epoch [   11/   25] | d_loss: 0.3933 | g_loss: 3.7766
Epoch [   11/   25] | d_loss: 0.2519 | g_loss: 3.1139
Epoch [   11/   25] | d_loss: 0.5915 | g_loss: 1.9835
Epoch [   11/   25] | d_loss: 0.8419 | g_loss: 1.4719
Epoch [   11/   25] | d_loss: 0.7306 | g_loss: 2.1625
Epoch [   11/   25] | d_loss: 0.6788 | g_loss: 2.9249
Epoch [   11/   25] | d_loss: 0.4615 | g_loss: 2.6040
Epoch [   11/   25] | d_loss: 0.2741 | g_loss: 3.8164
Epoch [   11/   25] | d_loss: 0.3294 | g_loss: 3.0394
Epoch [   11/   25] | d_loss: 0.5459 | g_loss: 2.8343
Epoch [   11/   25] | d_loss: 0.2657 | g_loss: 4.1911
Epoch [   12/   25] | d_loss: 1.1804 | g_loss: 2.7864
Epoch [   12/   25] | d_loss: 0.3446 | g_loss: 3.6484
Epoch [   12/   25] | d_loss: 0.7167 | g_loss: 3.5864
Epoch [   12/   25] | d_loss: 0.8088 | g_loss: 4.0141
Epoch [   12/   25] | d_loss: 0.6065 | g_loss: 2.6942
Epoch [   12/   25] | d_loss: 0.3137 | g_loss: 2.6627
Epoch [   12/   25] | d_loss: 0.6023 | g_loss: 2.4546
Epoch [   12/   25] | d_loss: 1.3735 | g_loss: 0.8534
Epoch [   12/   25] | d_loss: 1.1980 | g_loss: 2.6693
Epoch [   12/   25] | d_loss: 0.9752 | g_loss: 3.4800
Epoch [   12/   25] | d_loss: 0.8483 | g_loss: 2.0274
Epoch [   12/   25] | d_loss: 0.8420 | g_loss: 3.1963
Epoch [   12/   25] | d_loss: 0.4578 | g_loss: 3.8758
Epoch [   12/   25] | d_loss: 0.3533 | g_loss: 3.5983
Epoch [   12/   25] | d_loss: 0.9934 | g_loss: 3.0841
Epoch [   12/   25] | d_loss: 0.9240 | g_loss: 1.7644
Epoch [   12/   25] | d_loss: 0.4255 | g_loss: 3.2173
Epoch [   12/   25] | d_loss: 0.6934 | g_loss: 3.3474
Epoch [   12/   25] | d_loss: 0.5048 | g_loss: 3.6865
```

```
Epoch [   12/   25] | d_loss: 0.5790 | g_loss: 3.2494
Epoch [   12/   25] | d_loss: 0.6305 | g_loss: 3.7795
Epoch [   12/   25] | d_loss: 0.2859 | g_loss: 4.3573
Epoch [   12/   25] | d_loss: 0.8581 | g_loss: 2.6691
Epoch [   12/   25] | d_loss: 0.9266 | g_loss: 3.4522
Epoch [   12/   25] | d_loss: 1.4980 | g_loss: 1.2961
Epoch [   12/   25] | d_loss: 0.4149 | g_loss: 2.9138
Epoch [   12/   25] | d_loss: 0.4965 | g_loss: 1.3899
Epoch [   12/   25] | d_loss: 1.5176 | g_loss: 0.8478
Epoch [   12/   25] | d_loss: 0.9210 | g_loss: 2.6157
Epoch [   13/   25] | d_loss: 1.5762 | g_loss: 4.0243
Epoch [   13/   25] | d_loss: 1.1320 | g_loss: 4.5531
Epoch [   13/   25] | d_loss: 0.3383 | g_loss: 3.0725
Epoch [   13/   25] | d_loss: 0.8715 | g_loss: 4.3717
Epoch [   13/   25] | d_loss: 0.2877 | g_loss: 3.4268
Epoch [   13/   25] | d_loss: 0.8506 | g_loss: 4.1571
Epoch [   13/   25] | d_loss: 0.5197 | g_loss: 2.4341
Epoch [   13/   25] | d_loss: 0.6730 | g_loss: 2.4975
Epoch [   13/   25] | d_loss: 0.6182 | g_loss: 4.7904
Epoch [   13/   25] | d_loss: 1.1466 | g_loss: 4.7465
Epoch [   13/   25] | d_loss: 0.5417 | g_loss: 3.3832
Epoch [   13/   25] | d_loss: 1.4309 | g_loss: 0.3579
Epoch [   13/   25] | d_loss: 0.6964 | g_loss: 1.8617
Epoch [   13/   25] | d_loss: 0.6284 | g_loss: 1.7956
Epoch [   13/   25] | d_loss: 0.7542 | g_loss: 3.6478
Epoch [   13/   25] | d_loss: 0.6914 | g_loss: 1.7066
Epoch [   13/   25] | d_loss: 0.6228 | g_loss: 3.5868
Epoch [   13/   25] | d_loss: 3.3130 | g_loss: 1.5508
Epoch [   13/   25] | d_loss: 0.5239 | g_loss: 4.2288
Epoch [   13/   25] | d_loss: 0.8017 | g_loss: 5.0026
Epoch [   13/   25] | d_loss: 0.3035 | g_loss: 2.8267
Epoch [   13/   25] | d_loss: 1.8748 | g_loss: 4.5590
Epoch [   13/   25] | d_loss: 0.5230 | g_loss: 1.9090
Epoch [   13/   25] | d_loss: 0.2469 | g_loss: 4.1873
Epoch [   13/   25] | d_loss: 0.5898 | g_loss: 1.5591
Epoch [   13/   25] | d_loss: 0.5093 | g_loss: 3.3670
Epoch [   13/   25] | d_loss: 0.3436 | g_loss: 4.3081
Epoch [   13/   25] | d_loss: 0.6868 | g_loss: 1.9316
Epoch [   13/   25] | d_loss: 0.7688 | g_loss: 2.0613
Epoch [   14/   25] | d_loss: 1.6653 | g_loss: 3.8131
Epoch [   14/   25] | d_loss: 0.3629 | g_loss: 3.4135
Epoch [   14/   25] | d_loss: 4.4270 | g_loss: 0.3945
Epoch [   14/   25] | d_loss: 0.4621 | g_loss: 4.2350
Epoch [   14/   25] | d_loss: 0.8590 | g_loss: 4.0473
Epoch [   14/   25] | d_loss: 0.5326 | g_loss: 4.5430
Epoch [   14/   25] | d_loss: 0.6941 | g_loss: 3.6253
Epoch [   14/   25] | d_loss: 0.5418 | g_loss: 3.3585
Epoch [   14/   25] | d_loss: 0.6706 | g_loss: 3.1552
```

```
Epoch [   14/   25] | d_loss: 0.3330 | g_loss: 2.7862
Epoch [   14/   25] | d_loss: 0.6727 | g_loss: 2.5572
Epoch [   14/   25] | d_loss: 0.6412 | g_loss: 3.0511
Epoch [   14/   25] | d_loss: 1.3237 | g_loss: 2.0026
Epoch [   14/   25] | d_loss: 1.1602 | g_loss: 4.8368
Epoch [   14/   25] | d_loss: 0.4986 | g_loss: 3.1987
Epoch [   14/   25] | d_loss: 0.6451 | g_loss: 2.5922
Epoch [   14/   25] | d_loss: 0.4600 | g_loss: 4.5182
Epoch [   14/   25] | d_loss: 0.5943 | g_loss: 2.6143
Epoch [   14/   25] | d_loss: 0.6053 | g_loss: 4.1437
Epoch [   14/   25] | d_loss: 0.2415 | g_loss: 3.5471
Epoch [   14/   25] | d_loss: 0.4252 | g_loss: 3.8096
Epoch [   14/   25] | d_loss: 0.8866 | g_loss: 3.4161
Epoch [   14/   25] | d_loss: 0.4106 | g_loss: 2.9596
Epoch [   14/   25] | d_loss: 0.1529 | g_loss: 3.5857
Epoch [   14/   25] | d_loss: 3.1488 | g_loss: 0.2666
Epoch [   14/   25] | d_loss: 0.4709 | g_loss: 3.7891
Epoch [   14/   25] | d_loss: 0.8425 | g_loss: 2.3589
Epoch [   14/   25] | d_loss: 0.7138 | g_loss: 3.2197
Epoch [   14/   25] | d_loss: 0.6338 | g_loss: 2.7577
Epoch [   15/   25] | d_loss: 1.1620 | g_loss: 4.2212
Epoch [   15/   25] | d_loss: 1.3550 | g_loss: 3.3675
Epoch [   15/   25] | d_loss: 0.8754 | g_loss: 2.6375
Epoch [   15/   25] | d_loss: 0.7011 | g_loss: 2.3818
Epoch [   15/   25] | d_loss: 0.4671 | g_loss: 2.2342
Epoch [   15/   25] | d_loss: 0.4887 | g_loss: 2.4628
Epoch [   15/   25] | d_loss: 0.3090 | g_loss: 4.1145
Epoch [   15/   25] | d_loss: 1.1085 | g_loss: 5.0137
Epoch [   15/   25] | d_loss: 0.8640 | g_loss: 1.4786
Epoch [   15/   25] | d_loss: 0.3202 | g_loss: 3.2382
Epoch [   15/   25] | d_loss: 1.2750 | g_loss: 2.9883
Epoch [   15/   25] | d_loss: 0.8901 | g_loss: 3.5102
Epoch [   15/   25] | d_loss: 0.5768 | g_loss: 2.1998
Epoch [   15/   25] | d_loss: 0.7609 | g_loss: 4.0056
Epoch [   15/   25] | d_loss: 0.2714 | g_loss: 3.2211
Epoch [   15/   25] | d_loss: 0.7417 | g_loss: 2.2375
Epoch [   15/   25] | d_loss: 0.3110 | g_loss: 3.7779
Epoch [   15/   25] | d_loss: 0.7191 | g_loss: 3.4930
Epoch [   15/   25] | d_loss: 0.9117 | g_loss: 1.5163
Epoch [   15/   25] | d_loss: 2.0480 | g_loss: 0.8409
Epoch [   15/   25] | d_loss: 0.4508 | g_loss: 3.6812
Epoch [   15/   25] | d_loss: 0.1648 | g_loss: 4.3796
Epoch [   15/   25] | d_loss: 0.2071 | g_loss: 3.6995
Epoch [   15/   25] | d_loss: 0.6412 | g_loss: 2.3771
Epoch [   15/   25] | d_loss: 0.6815 | g_loss: 2.3032
Epoch [   15/   25] | d_loss: 0.6217 | g_loss: 2.1370
Epoch [   15/   25] | d_loss: 0.7171 | g_loss: 1.8029
Epoch [   15/   25] | d_loss: 1.2345 | g_loss: 4.0065
```

```
Epoch [   15/   25] | d_loss: 0.2256 | g_loss: 4.0646
Epoch [   16/   25] | d_loss: 0.1754 | g_loss: 3.7737
Epoch [   16/   25] | d_loss: 0.2686 | g_loss: 4.0839
Epoch [   16/   25] | d_loss: 0.6173 | g_loss: 2.7758
Epoch [   16/   25] | d_loss: 0.8210 | g_loss: 2.4101
Epoch [   16/   25] | d_loss: 0.8247 | g_loss: 3.5395
Epoch [   16/   25] | d_loss: 0.3556 | g_loss: 4.5371
Epoch [   16/   25] | d_loss: 1.4419 | g_loss: 3.3574
Epoch [   16/   25] | d_loss: 0.2953 | g_loss: 3.2672
Epoch [   16/   25] | d_loss: 0.4823 | g_loss: 4.3531
Epoch [   16/   25] | d_loss: 0.7852 | g_loss: 5.0105
Epoch [   16/   25] | d_loss: 1.4341 | g_loss: 4.1420
Epoch [   16/   25] | d_loss: 0.5262 | g_loss: 2.6281
Epoch [   16/   25] | d_loss: 0.6593 | g_loss: 3.1655
Epoch [   16/   25] | d_loss: 0.4822 | g_loss: 2.2697
Epoch [   16/   25] | d_loss: 0.3292 | g_loss: 4.8258
Epoch [   16/   25] | d_loss: 0.5333 | g_loss: 4.2409
Epoch [   16/   25] | d_loss: 0.7935 | g_loss: 2.6052
Epoch [   16/   25] | d_loss: 0.2777 | g_loss: 3.0945
Epoch [   16/   25] | d_loss: 0.0896 | g_loss: 3.5561
Epoch [   16/   25] | d_loss: 0.3863 | g_loss: 3.3601
Epoch [   16/   25] | d_loss: 0.6252 | g_loss: 1.9313
Epoch [   16/   25] | d_loss: 0.2223 | g_loss: 3.8739
Epoch [   16/   25] | d_loss: 0.6187 | g_loss: 2.1921
Epoch [   16/   25] | d_loss: 0.3777 | g_loss: 3.6630
Epoch [   16/   25] | d_loss: 0.7921 | g_loss: 1.2270
Epoch [   16/   25] | d_loss: 0.3681 | g_loss: 2.6027
Epoch [   16/   25] | d_loss: 0.6686 | g_loss: 1.8288
Epoch [   16/   25] | d_loss: 0.4006 | g_loss: 3.6846
Epoch [   16/   25] | d_loss: 1.1037 | g_loss: 1.1237
Epoch [   17/   25] | d_loss: 0.6687 | g_loss: 2.8748
Epoch [   17/   25] | d_loss: 1.6123 | g_loss: 0.8761
Epoch [   17/   25] | d_loss: 0.5069 | g_loss: 3.5090
Epoch [   17/   25] | d_loss: 0.3533 | g_loss: 2.3808
Epoch [   17/   25] | d_loss: 0.6722 | g_loss: 1.9441
Epoch [   17/   25] | d_loss: 0.6544 | g_loss: 1.7748
Epoch [   17/   25] | d_loss: 0.3149 | g_loss: 3.9894
Epoch [   17/   25] | d_loss: 0.7477 | g_loss: 2.6981
Epoch [   17/   25] | d_loss: 0.5407 | g_loss: 4.0689
Epoch [   17/   25] | d_loss: 1.0751 | g_loss: 2.8671
Epoch [   17/   25] | d_loss: 0.5915 | g_loss: 3.7430
Epoch [   17/   25] | d_loss: 0.2713 | g_loss: 3.7921
Epoch [   17/   25] | d_loss: 0.4640 | g_loss: 3.8016
Epoch [   17/   25] | d_loss: 0.4803 | g_loss: 3.7395
Epoch [   17/   25] | d_loss: 0.4937 | g_loss: 2.9785
Epoch [   17/   25] | d_loss: 0.4450 | g_loss: 4.1447
Epoch [   17/   25] | d_loss: 0.2251 | g_loss: 3.8204
Epoch [   17/   25] | d_loss: 0.5568 | g_loss: 2.2814
```

```
Epoch [   17/   25] | d_loss: 0.6525 | g_loss: 2.3921
Epoch [   17/   25] | d_loss: 0.2889 | g_loss: 3.6680
Epoch [   17/   25] | d_loss: 0.8009 | g_loss: 4.0203
Epoch [   17/   25] | d_loss: 0.9961 | g_loss: 5.5215
Epoch [   17/   25] | d_loss: 0.3773 | g_loss: 3.6729
Epoch [   17/   25] | d_loss: 0.2907 | g_loss: 3.3095
Epoch [   17/   25] | d_loss: 0.6645 | g_loss: 2.0581
Epoch [   17/   25] | d_loss: 0.6706 | g_loss: 2.0233
Epoch [   17/   25] | d_loss: 0.7534 | g_loss: 3.4083
Epoch [   17/   25] | d_loss: 0.5158 | g_loss: 2.5914
Epoch [   17/   25] | d_loss: 0.4938 | g_loss: 5.3324
Epoch [   18/   25] | d_loss: 0.7910 | g_loss: 5.1756
Epoch [   18/   25] | d_loss: 0.1237 | g_loss: 4.2513
Epoch [   18/   25] | d_loss: 0.4722 | g_loss: 3.5052
Epoch [   18/   25] | d_loss: 0.6341 | g_loss: 2.1086
Epoch [   18/   25] | d_loss: 0.4791 | g_loss: 2.5090
Epoch [   18/   25] | d_loss: 0.7977 | g_loss: 1.8993
Epoch [   18/   25] | d_loss: 0.7270 | g_loss: 2.8938
Epoch [   18/   25] | d_loss: 0.6668 | g_loss: 2.4071
Epoch [   18/   25] | d_loss: 0.5581 | g_loss: 2.5230
Epoch [   18/   25] | d_loss: 0.5792 | g_loss: 5.4908
Epoch [   18/   25] | d_loss: 0.4291 | g_loss: 5.2808
Epoch [   18/   25] | d_loss: 0.6103 | g_loss: 2.3483
Epoch [   18/   25] | d_loss: 0.2719 | g_loss: 4.5747
Epoch [   18/   25] | d_loss: 0.2916 | g_loss: 3.6058
Epoch [   18/   25] | d_loss: 0.5554 | g_loss: 2.6455
Epoch [   18/   25] | d_loss: 1.9448 | g_loss: 0.7794
Epoch [   18/   25] | d_loss: 0.7774 | g_loss: 4.8525
Epoch [   18/   25] | d_loss: 0.4964 | g_loss: 4.2131
Epoch [   18/   25] | d_loss: 1.4057 | g_loss: 3.3911
Epoch [   18/   25] | d_loss: 0.5065 | g_loss: 3.8028
Epoch [   18/   25] | d_loss: 0.6181 | g_loss: 4.3123
Epoch [   18/   25] | d_loss: 0.3071 | g_loss: 3.8087
Epoch [   18/   25] | d_loss: 0.6045 | g_loss: 3.4497
Epoch [   18/   25] | d_loss: 0.5180 | g_loss: 2.5890
Epoch [   18/   25] | d_loss: 0.5752 | g_loss: 3.0247
Epoch [   18/   25] | d_loss: 0.9883 | g_loss: 1.4855
Epoch [   18/   25] | d_loss: 0.4427 | g_loss: 3.8484
Epoch [   18/   25] | d_loss: 0.3629 | g_loss: 3.1622
Epoch [   18/   25] | d_loss: 0.5080 | g_loss: 4.0292
Epoch [   19/   25] | d_loss: 0.4428 | g_loss: 2.9147
Epoch [   19/   25] | d_loss: 1.0626 | g_loss: 1.2725
Epoch [   19/   25] | d_loss: 0.6916 | g_loss: 3.8045
Epoch [   19/   25] | d_loss: 0.2151 | g_loss: 4.5540
Epoch [   19/   25] | d_loss: 0.5552 | g_loss: 2.5401
Epoch [   19/   25] | d_loss: 0.7853 | g_loss: 3.9799
Epoch [   19/   25] | d_loss: 0.2212 | g_loss: 2.8417
Epoch [   19/   25] | d_loss: 0.2408 | g_loss: 4.3624
```

```
Epoch [   19/   25] | d_loss: 0.4617 | g_loss: 4.9019
Epoch [   19/   25] | d_loss: 0.4443 | g_loss: 2.8048
Epoch [   19/   25] | d_loss: 1.6607 | g_loss: 3.1570
Epoch [   19/   25] | d_loss: 0.4231 | g_loss: 3.2258
Epoch [   19/   25] | d_loss: 0.3600 | g_loss: 4.5336
Epoch [   19/   25] | d_loss: 0.4987 | g_loss: 5.1531
Epoch [   19/   25] | d_loss: 0.6064 | g_loss: 2.0285
Epoch [   19/   25] | d_loss: 0.3481 | g_loss: 3.0593
Epoch [   19/   25] | d_loss: 0.5373 | g_loss: 3.6042
Epoch [   19/   25] | d_loss: 0.5375 | g_loss: 2.1702
Epoch [   19/   25] | d_loss: 0.2074 | g_loss: 3.3811
Epoch [   19/   25] | d_loss: 2.9526 | g_loss: 0.7368
Epoch [   19/   25] | d_loss: 0.5512 | g_loss: 2.2673
Epoch [   19/   25] | d_loss: 0.5164 | g_loss: 3.9549
Epoch [   19/   25] | d_loss: 0.5335 | g_loss: 4.7594
Epoch [   19/   25] | d_loss: 0.8800 | g_loss: 4.5716
Epoch [   19/   25] | d_loss: 0.5375 | g_loss: 2.0711
Epoch [   19/   25] | d_loss: 0.6248 | g_loss: 1.7581
Epoch [   19/   25] | d_loss: 0.8340 | g_loss: 3.0437
Epoch [   19/   25] | d_loss: 0.2312 | g_loss: 3.3476
Epoch [   19/   25] | d_loss: 0.2001 | g_loss: 4.5779
Epoch [   20/   25] | d_loss: 0.7221 | g_loss: 3.1492
Epoch [   20/   25] | d_loss: 0.5761 | g_loss: 1.7088
Epoch [   20/   25] | d_loss: 0.5249 | g_loss: 2.2915
Epoch [   20/   25] | d_loss: 0.8937 | g_loss: 5.7232
Epoch [   20/   25] | d_loss: 0.6530 | g_loss: 2.0018
Epoch [   20/   25] | d_loss: 0.3441 | g_loss: 5.0030
Epoch [   20/   25] | d_loss: 0.5037 | g_loss: 5.3238
Epoch [   20/   25] | d_loss: 0.2838 | g_loss: 3.7775
Epoch [   20/   25] | d_loss: 1.0047 | g_loss: 3.6078
Epoch [   20/   25] | d_loss: 0.5396 | g_loss: 2.3478
Epoch [   20/   25] | d_loss: 0.6174 | g_loss: 2.6924
Epoch [   20/   25] | d_loss: 0.4379 | g_loss: 5.3347
Epoch [   20/   25] | d_loss: 0.2492 | g_loss: 4.0627
Epoch [   20/   25] | d_loss: 1.4821 | g_loss: 2.2532
Epoch [   20/   25] | d_loss: 0.9539 | g_loss: 4.4431
Epoch [   20/   25] | d_loss: 1.6923 | g_loss: 1.8337
Epoch [   20/   25] | d_loss: 0.5409 | g_loss: 2.0382
Epoch [   20/   25] | d_loss: 0.5300 | g_loss: 3.3061
Epoch [   20/   25] | d_loss: 0.2686 | g_loss: 4.7126
Epoch [   20/   25] | d_loss: 0.6105 | g_loss: 3.5000
Epoch [   20/   25] | d_loss: 0.2702 | g_loss: 3.8773
Epoch [   20/   25] | d_loss: 0.5078 | g_loss: 2.9080
Epoch [   20/   25] | d_loss: 0.9561 | g_loss: 4.2052
Epoch [   20/   25] | d_loss: 0.3292 | g_loss: 2.6044
Epoch [   20/   25] | d_loss: 0.4680 | g_loss: 3.1079
Epoch [   20/   25] | d_loss: 0.6475 | g_loss: 2.0309
Epoch [   20/   25] | d_loss: 1.1999 | g_loss: 1.4462
```

```
Epoch [   20/   25] | d_loss: 0.3301 | g_loss: 4.4069
Epoch [   20/   25] | d_loss: 0.2704 | g_loss: 3.9770
Epoch [   21/   25] | d_loss: 0.3090 | g_loss: 4.9901
Epoch [   21/   25] | d_loss: 0.6219 | g_loss: 2.8575
Epoch [   21/   25] | d_loss: 0.3310 | g_loss: 4.3716
Epoch [   21/   25] | d_loss: 0.8598 | g_loss: 3.3896
Epoch [   21/   25] | d_loss: 0.8530 | g_loss: 2.9290
Epoch [   21/   25] | d_loss: 0.3822 | g_loss: 5.5122
Epoch [   21/   25] | d_loss: 0.6085 | g_loss: 2.7530
Epoch [   21/   25] | d_loss: 0.3241 | g_loss: 3.1284
Epoch [   21/   25] | d_loss: 0.5368 | g_loss: 3.5807
Epoch [   21/   25] | d_loss: 0.3062 | g_loss: 5.7253
Epoch [   21/   25] | d_loss: 0.6000 | g_loss: 3.2763
Epoch [   21/   25] | d_loss: 0.9640 | g_loss: 2.5999
Epoch [   21/   25] | d_loss: 0.7771 | g_loss: 1.9834
Epoch [   21/   25] | d_loss: 0.4495 | g_loss: 4.1657
Epoch [   21/   25] | d_loss: 0.4065 | g_loss: 2.4771
Epoch [   21/   25] | d_loss: 0.4144 | g_loss: 2.8759
Epoch [   21/   25] | d_loss: 0.2861 | g_loss: 3.5223
Epoch [   21/   25] | d_loss: 0.5160 | g_loss: 5.0965
Epoch [   21/   25] | d_loss: 0.3200 | g_loss: 3.4238
Epoch [   21/   25] | d_loss: 2.2122 | g_loss: 6.0178
Epoch [   21/   25] | d_loss: 0.6509 | g_loss: 5.3611
Epoch [   21/   25] | d_loss: 0.2656 | g_loss: 3.2819
Epoch [   21/   25] | d_loss: 0.2030 | g_loss: 3.2862
Epoch [   21/   25] | d_loss: 1.2800 | g_loss: 4.6918
Epoch [   21/   25] | d_loss: 0.9475 | g_loss: 4.1283
Epoch [   21/   25] | d_loss: 0.3189 | g_loss: 3.9940
Epoch [   21/   25] | d_loss: 0.3012 | g_loss: 4.2987
Epoch [   21/   25] | d_loss: 2.1522 | g_loss: 4.2016
Epoch [   21/   25] | d_loss: 0.2505 | g_loss: 3.8864
Epoch [   22/   25] | d_loss: 0.8158 | g_loss: 3.5298
Epoch [   22/   25] | d_loss: 0.2268 | g_loss: 4.5189
Epoch [   22/   25] | d_loss: 0.6538 | g_loss: 2.2362
Epoch [   22/   25] | d_loss: 0.2957 | g_loss: 2.9779
Epoch [   22/   25] | d_loss: 0.2964 | g_loss: 3.4131
Epoch [   22/   25] | d_loss: 0.8939 | g_loss: 3.6668
Epoch [   22/   25] | d_loss: 0.4666 | g_loss: 4.4953
Epoch [   22/   25] | d_loss: 0.6465 | g_loss: 4.8625
Epoch [   22/   25] | d_loss: 0.2019 | g_loss: 3.5818
Epoch [   22/   25] | d_loss: 0.2092 | g_loss: 4.7175
Epoch [   22/   25] | d_loss: 0.6346 | g_loss: 2.6952
Epoch [   22/   25] | d_loss: 0.6110 | g_loss: 3.9519
Epoch [   22/   25] | d_loss: 0.6015 | g_loss: 2.1370
Epoch [   22/   25] | d_loss: 0.5945 | g_loss: 3.2726
Epoch [   22/   25] | d_loss: 0.6539 | g_loss: 4.0094
Epoch [   22/   25] | d_loss: 0.7198 | g_loss: 2.1072
Epoch [   22/   25] | d_loss: 0.2357 | g_loss: 3.4842
```

```
Epoch [   22/   25] | d_loss: 0.2645 | g_loss: 3.6557
Epoch [   22/   25] | d_loss: 0.4338 | g_loss: 3.3982
Epoch [   22/   25] | d_loss: 0.6116 | g_loss: 2.6400
Epoch [   22/   25] | d_loss: 1.0643 | g_loss: 4.6544
Epoch [   22/   25] | d_loss: 1.0412 | g_loss: 1.1405
Epoch [   22/   25] | d_loss: 0.2933 | g_loss: 3.7097
Epoch [   22/   25] | d_loss: 0.5543 | g_loss: 3.6445
Epoch [   22/   25] | d_loss: 0.6051 | g_loss: 2.1737
Epoch [   22/   25] | d_loss: 0.4642 | g_loss: 5.8002
Epoch [   22/   25] | d_loss: 0.2670 | g_loss: 3.7488
Epoch [   22/   25] | d_loss: 0.6037 | g_loss: 5.5825
Epoch [   22/   25] | d_loss: 0.6211 | g_loss: 3.0375
Epoch [   23/   25] | d_loss: 1.0029 | g_loss: 1.0001
Epoch [   23/   25] | d_loss: 0.2160 | g_loss: 4.1260
Epoch [   23/   25] | d_loss: 0.6551 | g_loss: 1.6277
Epoch [   23/   25] | d_loss: 0.4905 | g_loss: 2.7496
Epoch [   23/   25] | d_loss: 0.4233 | g_loss: 3.9319
Epoch [   23/   25] | d_loss: 0.2678 | g_loss: 4.3175
Epoch [   23/   25] | d_loss: 0.3269 | g_loss: 4.9034
Epoch [   23/   25] | d_loss: 0.5646 | g_loss: 2.9554
Epoch [   23/   25] | d_loss: 0.2456 | g_loss: 4.2105
Epoch [   23/   25] | d_loss: 0.2376 | g_loss: 3.9803
Epoch [   23/   25] | d_loss: 0.1777 | g_loss: 4.0612
Epoch [   23/   25] | d_loss: 0.1920 | g_loss: 4.0527
Epoch [   23/   25] | d_loss: 0.1747 | g_loss: 4.7146
Epoch [   23/   25] | d_loss: 0.1927 | g_loss: 4.4462
Epoch [   23/   25] | d_loss: 0.5223 | g_loss: 2.4612
Epoch [   23/   25] | d_loss: 0.6180 | g_loss: 2.3359
Epoch [   23/   25] | d_loss: 0.5491 | g_loss: 2.6965
Epoch [   23/   25] | d_loss: 0.4177 | g_loss: 3.6262
Epoch [   23/   25] | d_loss: 3.9546 | g_loss: 0.8627
Epoch [   23/   25] | d_loss: 0.8527 | g_loss: 3.0355
Epoch [   23/   25] | d_loss: 0.5377 | g_loss: 2.8601
Epoch [   23/   25] | d_loss: 0.3654 | g_loss: 4.1284
Epoch [   23/   25] | d_loss: 0.6191 | g_loss: 2.7422
Epoch [   23/   25] | d_loss: 0.6027 | g_loss: 3.9642
Epoch [   23/   25] | d_loss: 1.3064 | g_loss: 4.4668
Epoch [   23/   25] | d_loss: 0.3440 | g_loss: 4.3484
Epoch [   23/   25] | d_loss: 0.6304 | g_loss: 4.6019
Epoch [   23/   25] | d_loss: 0.3084 | g_loss: 3.9597
Epoch [   23/   25] | d_loss: 0.2668 | g_loss: 4.1604
Epoch [   24/   25] | d_loss: 1.4058 | g_loss: 5.2760
Epoch [   24/   25] | d_loss: 1.4663 | g_loss: 3.9969
Epoch [   24/   25] | d_loss: 0.4113 | g_loss: 5.1446
Epoch [   24/   25] | d_loss: 0.0992 | g_loss: 4.1437
Epoch [   24/   25] | d_loss: 0.7790 | g_loss: 3.5404
Epoch [   24/   25] | d_loss: 0.7471 | g_loss: 1.3238
Epoch [   24/   25] | d_loss: 0.6410 | g_loss: 2.8327
```

```
Epoch [   24/   25] | d_loss: 0.3300 | g_loss: 3.9938
Epoch [   24/   25] | d_loss: 0.7599 | g_loss: 0.9941
Epoch [   24/   25] | d_loss: 0.3037 | g_loss: 4.8934
Epoch [   24/   25] | d_loss: 0.4578 | g_loss: 3.3605
Epoch [   24/   25] | d_loss: 0.9477 | g_loss: 4.1304
Epoch [   24/   25] | d_loss: 0.6121 | g_loss: 3.8773
Epoch [   24/   25] | d_loss: 0.2876 | g_loss: 3.8488
Epoch [   24/   25] | d_loss: 0.3848 | g_loss: 4.2041
Epoch [   24/   25] | d_loss: 0.4328 | g_loss: 3.0680
Epoch [   24/   25] | d_loss: 0.3621 | g_loss: 3.6556
Epoch [   24/   25] | d_loss: 0.2148 | g_loss: 4.1505
Epoch [   24/   25] | d_loss: 0.1951 | g_loss: 4.1228
Epoch [   24/   25] | d_loss: 0.8326 | g_loss: 2.8255
Epoch [   24/   25] | d_loss: 0.6476 | g_loss: 2.3222
Epoch [   24/   25] | d_loss: 0.4507 | g_loss: 2.7513
Epoch [   24/   25] | d_loss: 1.5017 | g_loss: 4.6215
Epoch [   24/   25] | d_loss: 0.7903 | g_loss: 3.5556
Epoch [   24/   25] | d_loss: 0.4833 | g_loss: 3.6149
Epoch [   24/   25] | d_loss: 0.4219 | g_loss: 2.9928
Epoch [   24/   25] | d_loss: 0.2992 | g_loss: 4.5471
Epoch [   24/   25] | d_loss: 0.4662 | g_loss: 2.8564
Epoch [   24/   25] | d_loss: 0.6117 | g_loss: 3.1026
Epoch [   25/   25] | d_loss: 0.9729 | g_loss: 3.6510
Epoch [   25/   25] | d_loss: 0.1509 | g_loss: 3.9491
Epoch [   25/   25] | d_loss: 0.5905 | g_loss: 2.8750
Epoch [   25/   25] | d_loss: 0.4800 | g_loss: 2.5020
Epoch [   25/   25] | d_loss: 2.0657 | g_loss: 1.2673
Epoch [   25/   25] | d_loss: 0.5853 | g_loss: 1.7653
Epoch [   25/   25] | d_loss: 0.5697 | g_loss: 2.9874
Epoch [   25/   25] | d_loss: 0.2038 | g_loss: 3.5822
Epoch [   25/   25] | d_loss: 0.9425 | g_loss: 6.5755
Epoch [   25/   25] | d_loss: 0.3542 | g_loss: 4.9652
Epoch [   25/   25] | d_loss: 0.2410 | g_loss: 4.9449
Epoch [   25/   25] | d_loss: 0.7201 | g_loss: 2.5143
Epoch [   25/   25] | d_loss: 0.4740 | g_loss: 5.8464
Epoch [   25/   25] | d_loss: 0.2811 | g_loss: 3.7509
Epoch [   25/   25] | d_loss: 0.2698 | g_loss: 5.5968
Epoch [   25/   25] | d_loss: 0.4094 | g_loss: 4.7452
Epoch [   25/   25] | d_loss: 0.5188 | g_loss: 2.5190
Epoch [   25/   25] | d_loss: 0.6432 | g_loss: 2.6290
Epoch [   25/   25] | d_loss: 0.5162 | g_loss: 2.5768
Epoch [   25/   25] | d_loss: 0.9033 | g_loss: 4.7972
Epoch [   25/   25] | d_loss: 0.5022 | g_loss: 3.4584
Epoch [   25/   25] | d_loss: 0.8725 | g_loss: 4.0959
Epoch [   25/   25] | d_loss: 0.1976 | g_loss: 4.3805
Epoch [   25/   25] | d_loss: 0.1205 | g_loss: 3.4966
Epoch [   25/   25] | d_loss: 0.5578 | g_loss: 4.4902
Epoch [   25/   25] | d_loss: 0.1309 | g_loss: 3.4457
```

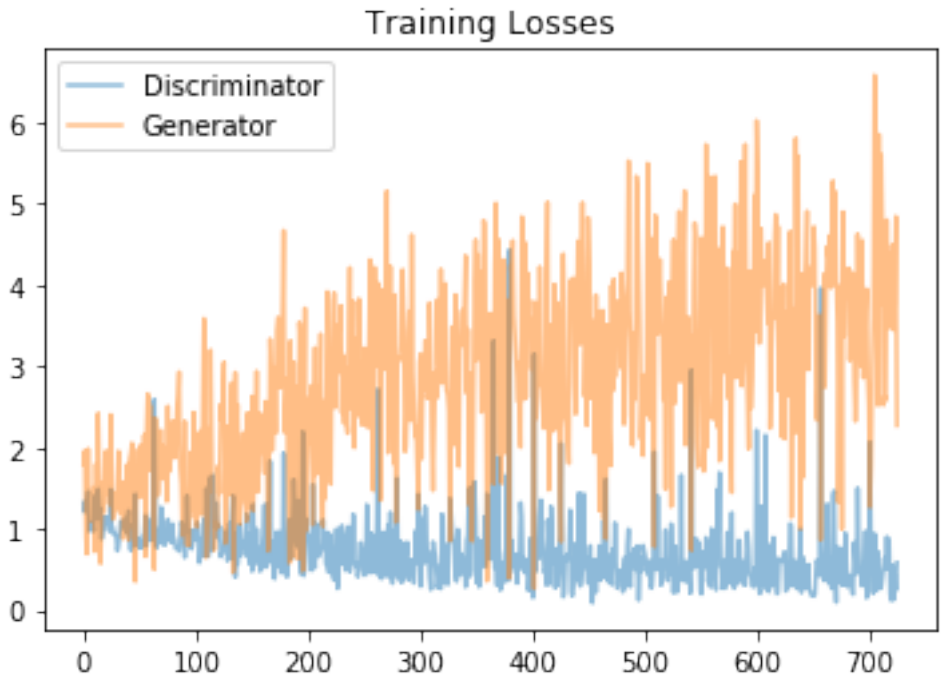
```
Epoch [   25/   25] | d_loss: 0.4452 | g_loss: 3.7630
Epoch [   25/   25] | d_loss: 0.2451 | g_loss: 4.8398
Epoch [   25/   25] | d_loss: 0.5866 | g_loss: 2.2704
```

2.8 Training loss

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

```
In [105]: 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[105]: <matplotlib.legend.Legend at 0x7f54d682a780>
```



2.9 Generator samples from training

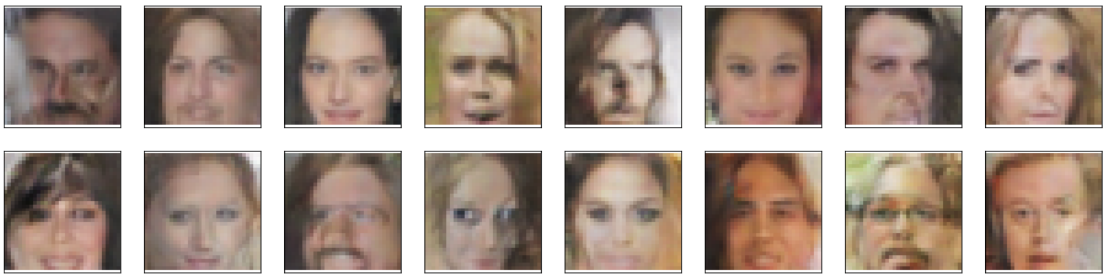
View samples of images from the generator, and answer a question about the strengths and weaknesses of your trained models.

```
In [106]: # helper function for viewing a list of passed in sample images
          def view_samples(epoch, samples):
```

```
    fig, axes = plt.subplots(figsize=(16,4), nrows=2, ncols=8, sharey=True, sharex=Tru
    for ax, img in zip(axes.flatten(), samples[epoch]):
        img = img.detach().cpu().numpy()
        img = np.transpose(img, (1, 2, 0))
        img = ((img + 1)*255 / (2)).astype(np.uint8)
        ax.xaxis.set_visible(False)
        ax.yaxis.set_visible(False)
        im = ax.imshow(img.reshape((32,32,3)))
```

In [107]:
```
# Load samples from generator, taken while training
with open('train_samples.pkl', 'rb') as f:
    samples = pkl.load(f)
```

In [109]:
```
_ = view_samples(-1, samples)
```



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

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

Answer: (Write your answer in this cell). * As stated in the cell above, the dataset is a bias one mostly made of the whites celeb. And also have the same patterns, or may the same standards of picture representation. If more datasets of variety are used, I suggest images that look closely as humans will be generated.

- The network architecture I Implemented is of 3 CNN layers and a Fully connected one at the output; I have used fewer layers initially, and i noticed the later(the one used) model size made better images than before. Generally, deeper networks count. I used, suggestively, fewer deep CNN because the datasets data are of lower resolution and that there are no much features as adding deeper layers might affect the model.
- I tested several number of epochs and found out that if I use more number of epochs there are less sculptured generated images. Adam is the best choice for GAN's as well as other architectures from many sources i have read and seen in previous project in the program.

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.