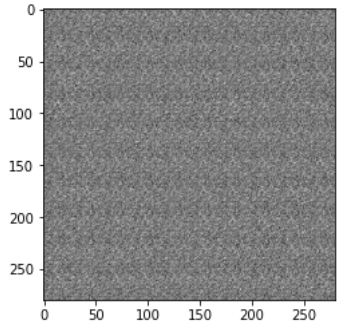
```
import torch
from torchvision import datasets
import torch.nn as nn
from torch.utils.data import DataLoader
from torchvision import transforms
import numpy as np
import os
from matplotlib.pyplot import imsave
import matplotlib.pyplot as plt
import matplotlib.image as mpimg


# A transform to convert the images to tensor and normalize their RGB values
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize(mean=[0.5], std=[0.5])]
)

data = datasets.FashionMNIST(root='../data/', train=True, transform=transform, download=True)

batch_size = 64
data_loader = DataLoader(dataset=data, batch_size=batch_size, shuffle=True, drop_last=True)


def get_sample_image(G, DEVICE, n_noise=100):
    img = np.zeros([280, 280])
    for j in range(10):
        z = torch.randn(10, n_noise).to(DEVICE)
        y_hat = G(z).view(10, 28, 28)
        result = y_hat.cpu().data.numpy()
        img[j*28:(j+1)*28] = np.concatenate([x for x in result], axis=-1)
    return img


class Generator(nn.Module):
    def __init__(self, input_size=100,image_size=28*28):
        super(Generator, self).__init__()
        self.network = nn.Sequential(
            nn.Linear(input_size, 128), # auxillary dimension for label
            nn.LeakyReLU(0.2),
            nn.Linear(128, 256),
            nn.BatchNorm1d(256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 512),
            nn.BatchNorm1d(512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 1024),
            nn.BatchNorm1d(1024),
            nn.LeakyReLU(0.2),
            nn.Linear(1024, image_size),
            nn.Tanh()
        )

    def forward(self, x):
        x = x.view(x.size(0), -1)
        #v = torch.cat((x, 1)) # v: [input, label] concatenated vector
        y_ = self.network(x)
        y_ = y_.view(x.size(0), 1, 28, 28)
        return y_
```

```
class Discriminator(nn.Module):
    def __init__(self, input_size=28*28, num_output=1):
        super(Discriminator, self).__init__()
        self.network = nn.Sequential(
            nn.Linear(input_size, 512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, num_output),
            nn.Sigmoid(),
        )

    def forward(self, x):
        x = x.view(x.size(0), -1)
        # v = torch.cat((x), 1) # v: [input, label] concatenated vector
        y_ = self.network(x)
        return y_


MODEL_NAME = 'GAN'
DEVICE = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

D = Discriminator().to(DEVICE)
G = Generator().to(DEVICE)

max_epoch = 10
step = 0
n_noise = 100 # size of noise vector

criterion = nn.BCELoss()
D_opt = torch.optim.Adam(D.parameters(), lr=0.0002, betas=(0.5, 0.999))
G_opt = torch.optim.Adam(G.parameters(), lr=0.0002, betas=(0.5, 0.999))

# We will denote real images as 1s and fake images as 0s
# This is why we needed to drop the last batch of the data loader
D_labels = torch.ones([batch_size, 1]).to(DEVICE) # Discriminator label: real
D_fakes = torch.zeros([batch_size, 1]).to(DEVICE) # Discriminator Label: fake


# a directory to save the generated images
if not os.path.exists('samples'):
    os.makedirs('samples')

for epoch in range(max_epoch):
    for idx, (images, _) in enumerate(data_loader):
        # Training Discriminator
        x = images.to(DEVICE)
        x_outputs = D(x) # input includes labels
        D_x_loss = criterion(x_outputs, D_labels) # Discriminator loss for real images

        z = torch.randn(batch_size, n_noise).to(DEVICE)
        z_outputs = D(G(z)) # input to both generator and discriminator includes labels
        D_z_loss = criterion(z_outputs, D_fakes) # Discriminator loss for fake images
        D_loss = D_x_loss + D_z_loss # Total Discriminator loss

        D.zero_grad()
        D_loss.backward()
        D_opt.step()

        # Training Generator
        z = torch.randn(batch_size, n_noise).to(DEVICE)
        z_outputs = D(G(z))
        G_loss = -1 * criterion(z_outputs, D_fakes) # Generator loss is negative disciminator loss

        G.zero_grad()
        G_loss.backward()
        G_opt.step()

        if step % 500 == 0:
            print('Epoch: {}/{}, Step: {}, D Loss: {}, G Loss: {}'.format(epoch, max_epoch, step, D_loss.item(), G_loss.item()))

        if step % 1000 == 0:
            G.eval()
            img = get_sample_image(G, DEVICE, n_noise)
            imsave('samples/{}_step{}.jpg'.format(MODEL_NAME, str(step).zfill(3)), img, cmap='gray')
            G.train()
        step += 1
```

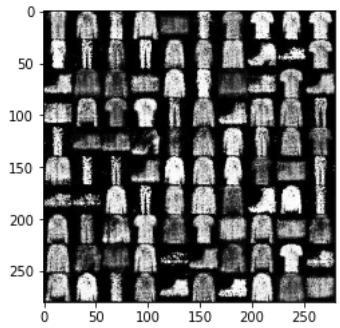
```
Epoch: 0/10, Step: 0, D Loss: 1.4120373725891113, G Loss: -0.7062667608261108
Epoch: 0/10, Step: 500, D Loss: 1.1913490295410156, G Loss: -0.5061103701591492
Epoch: 1/10, Step: 1000, D Loss: 1.1233105659484863, G Loss: -0.593504786491394
Epoch: 1/10, Step: 1500, D Loss: 1.2383441925048828, G Loss: -0.45383721590042114
Epoch: 2/10, Step: 2000, D Loss: 1.2721788883209229, G Loss: -0.4557284116744995
Epoch: 2/10, Step: 2500, D Loss: 1.3356597423553467, G Loss: -0.5755120515823364
Epoch: 3/10, Step: 3000, D Loss: 1.331081748008728, G Loss: -0.5234083533287048
Epoch: 3/10, Step: 3500, D Loss: 1.38785719871521, G Loss: -0.627362847328186
Epoch: 4/10, Step: 4000, D Loss: 1.3459150791168213, G Loss: -0.5192853808403015
Epoch: 4/10, Step: 4500, D Loss: 1.3248074054718018, G Loss: -0.5603094696998596
Epoch: 5/10, Step: 5000, D Loss: 1.3582115173339844, G Loss: -0.6155641078948975
Epoch: 5/10, Step: 5500, D Loss: 1.2880699634552002, G Loss: -0.5496246814727783
Epoch: 6/10, Step: 6000, D Loss: 1.3480061292648315, G Loss: -0.6984512805938721
Epoch: 6/10, Step: 6500, D Loss: 1.2495630979537964, G Loss: -0.5555623173713684
Epoch: 7/10, Step: 7000, D Loss: 1.299135446548462, G Loss: -0.5941945314407349
Epoch: 8/10, Step: 7500, D Loss: 1.3808751106262207, G Loss: -0.5168596506118774
Epoch: 8/10, Step: 8000, D Loss: 1.3835018873214722, G Loss: -0.6137067675590515
Epoch: 9/10, Step: 8500, D Loss: 1.370305061340332, G Loss: -0.6274106502532959
Epoch: 9/10, Step: 9000, D Loss: 1.32705557346344, G Loss: -0.6069644689559937
```

Now let's plot these images. At first, the generator just produces noise (as we expect).

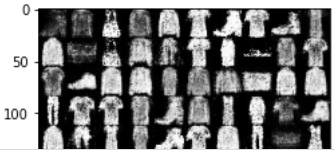
```
img = mpimg.imread('samples/GAN_step000.jpg')
imgplot = plt.imshow(img)
plt.show()
```




```
img = mpimg.imread('samples/GAN_step5000.jpg')
imgplot = plt.imshow(img)
plt.show()
```




```
img = mpimg.imread('samples/GAN_step9000.jpg')
imgplot = plt.imshow(img)
plt.show()
```


0
50
100