hw3q4ab-fashion-gan

November 22, 2023

Fashion GAN

[ ]:
```
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) #Uses FashionMNIST dataset now

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

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-
images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-
images-idx3-ubyte.gz to ../data/FashionMNIST/raw/train-images-idx3-ubyte.gz

100%|      | 26421880/26421880 [00:03<00:00, 8704686.20it/s]

Extracting ../data/FashionMNIST/raw/train-images-idx3-ubyte.gz to
../data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-
labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-
labels-idx1-ubyte.gz to ../data/FashionMNIST/raw/train-labels-idx1-ubyte.gz
```

```
100%|     | 29515/29515 [00:00<00:00, 172394.90it/s]

Extracting ../data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to
../data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to
../data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz

100%|     | 4422102/4422102 [00:01<00:00, 3175625.37it/s]

Extracting ../data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to
../data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to
../data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz

100%|     | 5148/5148 [00:00<00:00, 6623397.85it/s]

Extracting ../data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to
../data/FashionMNIST/raw
```

[ ]:
```
def to_onehot(x, num_classes=10):
    assert isinstance(x, int) or isinstance(x, (torch.LongTensor, torch.cuda.
↪LongTensor))
    if isinstance(x, int):
        c = torch.zeros(1, num_classes).long()
        c[0][x] = 1
    else:
        x = x.cpu()
        c = torch.LongTensor(x.size(0), num_classes)
        c.zero_()
        c.scatter_(1, x, 1) # dim, index, src value
    return c
```

[ ]:
```
to_onehot(3)
```

[ ]:
```
tensor([[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]])
```

[ ]:
```
def get_sample_image(G, DEVICE, n_noise=100):
    img = np.zeros([280, 280])
    for j in range(10):
```

```
        c = torch.zeros([10, 10]).to(DEVICE)
        c[:, j] = 1
        z = torch.randn(10, n_noise).to(DEVICE)
        y_hat = G(z,c).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
```

0.1 Architecture

We now instantiate the generator and discriminator architectures. The generator takes a random noise vector and a one hot encoded label as input and produces an image. The discriminator takes an image and a one hot encoded label as input and produces a single value between 0 and 1. The discriminator is trained to output 1 for real images and 0 for fake images. The generator is trained to fool the discriminator by outputting images that look real.

I edited the number of classes from being included in the models, this will make the model a GAN instead of a Conditional GAN.

[ ]:
```
class Generator(nn.Module):
    def __init__(self, input_size=100, num_classes=10, image_size=28*28):
        super(Generator, self).__init__()
        self.network = nn.Sequential(
            nn.Linear(input_size, 128), # auxillary dimension for label; Got␣
 ↪rid of +num_classes
            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, c):
        x, c = x.view(x.size(0), -1), c.view(c.size(0), -1).float()
        v = x # v: [input, label] concatenated vector; Got rid of concatenation
        y_ = self.network(v)
        y_ = y_.view(x.size(0), 1, 28, 28)
        return y_
```

[ ]:
```python
class Discriminator(nn.Module):
    def __init__(self, input_size=28*28, num_classes=10, num_output=1):
        super(Discriminator, self).__init__()
        self.network = nn.Sequential(
            nn.Linear(input_size, 512), #Got rid of +num_classes
            nn.LeakyReLU(0.2),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, num_output),
            nn.Sigmoid(),
        )

    def forward(self, x, c):
        x, c = x.view(x.size(0), -1), c.view(c.size(0), -1).float()
        v = x # v: [input, label] concatenated vector; Got rid of concatenation
        y_ = self.network(v)
        return y_
```

[ ]:
```python
MODEL_NAME = 'GAN' #Changed the name
DEVICE = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

D = Discriminator().to(DEVICE) # randomly intialized
G = Generator().to(DEVICE) # randomly initialized

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
all_ones = torch.ones([batch_size, 1]).to(DEVICE) # Discriminator label: real
all_zeros = torch.zeros([batch_size, 1]).to(DEVICE) # Discriminator Label: fake
```

[ ]:
```python
images, class_labels = next(iter(data_loader))
class_labels_encoded = class_labels.view(batch_size, 1)
class_labels_encoded = to_onehot(class_labels_encoded).to(DEVICE)
print(class_labels[:10])
print(class_labels_encoded[:10])
```

```
tensor([3, 7, 9, 4, 4, 8, 2, 6, 9, 8])
tensor([[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
```

```
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
        [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]], device='cuda:0')
```

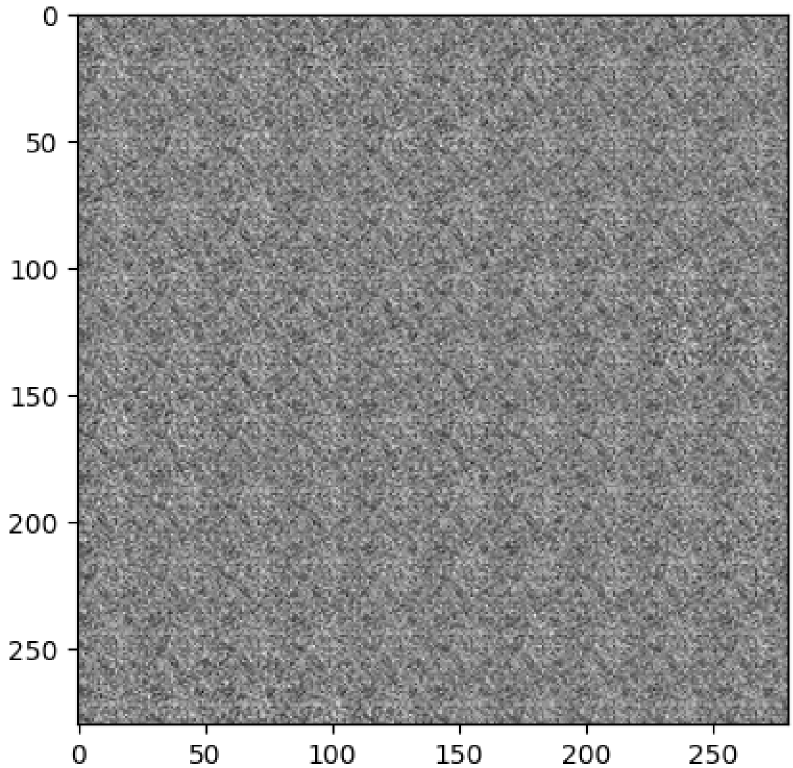
[ ]:
```
# a directory to save the generated images
if not os.path.exists('samples'):
    os.makedirs('samples')

for epoch in range(max_epoch):
    for idx, (images, class_labels) in enumerate(data_loader):
        # Training Discriminator
        x = images.to(DEVICE)
        class_labels = class_labels.view(batch_size, 1) # add singleton␣
↪dimension so batch_size x 1
        class_labels = to_onehot(class_labels).to(DEVICE)
        x_outputs = D(x, class_labels) # input includes labels
        D_x_loss = criterion(x_outputs, all_ones) # Discriminator loss for real␣
↪images

        z = torch.randn(batch_size, n_noise).to(DEVICE)
        z_outputs = D(G(z, class_labels), class_labels) # input to both␣
↪generator and discriminator includes labels
        D_z_loss = criterion(z_outputs, all_zeros) # 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, class_labels), class_labels)
        G_loss = -1 * criterion(z_outputs, all_zeros) # Generator loss is␣
↪negative discriminator 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: 1.4167184829711914, G Loss: -0.7008953094482422
Epoch: 0/10, Step: 500, D Loss: 1.198230504989624, G Loss: -0.44864386320114136
Epoch: 1/10, Step: 1000, D Loss: 1.2353107929229736, G Loss: -0.5327345132827759
Epoch: 1/10, Step: 1500, D Loss: 1.2166428565979004, G Loss: -0.5280460119247437
Epoch: 2/10, Step: 2000, D Loss: 1.304343581199646, G Loss: -0.4828684329986572
Epoch: 2/10, Step: 2500, D Loss: 1.2642494440078735, G Loss: -0.5151442289352417
Epoch: 3/10, Step: 3000, D Loss: 1.257765531539917, G Loss: -0.5388797521591187
Epoch: 3/10, Step: 3500, D Loss: 1.4073083400726318, G Loss: -0.7102686166763306
Epoch: 4/10, Step: 4000, D Loss: 1.3161141872406006, G Loss: -0.5851697325706482
Epoch: 4/10, Step: 4500, D Loss: 1.2726058959960938, G Loss: -0.5268336534500122
Epoch: 5/10, Step: 5000, D Loss: 1.3868470191955566, G Loss: -0.5979888439178467
Epoch: 5/10, Step: 5500, D Loss: 1.3306419849395752, G Loss: -0.5476356148719788
Epoch: 6/10, Step: 6000, D Loss: 1.3461498022079468, G Loss: -0.7852540016174316
Epoch: 6/10, Step: 6500, D Loss: 1.2957665920257568, G Loss: -0.5773029327392578
Epoch: 7/10, Step: 7000, D Loss: 1.3240413665771484, G Loss: -0.658687949180603
Epoch: 8/10, Step: 7500, D Loss: 1.3452906608581543, G Loss: -0.6307313442230225
Epoch: 8/10, Step: 8000, D Loss: 1.3202133178710938, G Loss: -0.6333258152008057
Epoch: 9/10, Step: 8500, D Loss: 1.4612188339233398, G Loss: -0.6147671937942505
Epoch: 9/10, Step: 9000, D Loss: 1.390981674194336, G Loss: -0.6171368360519409
```

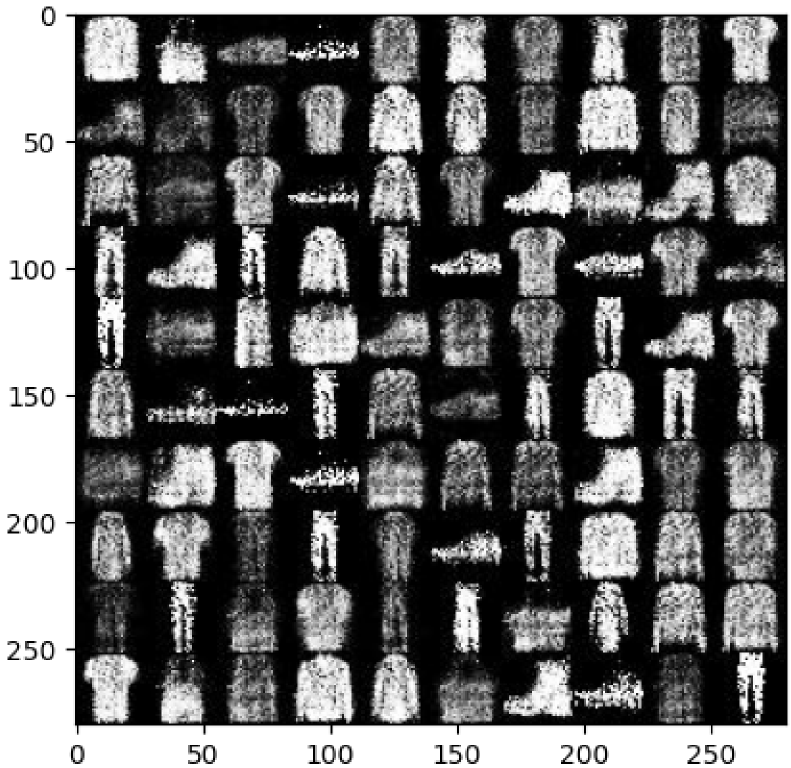
Now let's plot these images. At first, the generator just produces noise (as we expect).

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

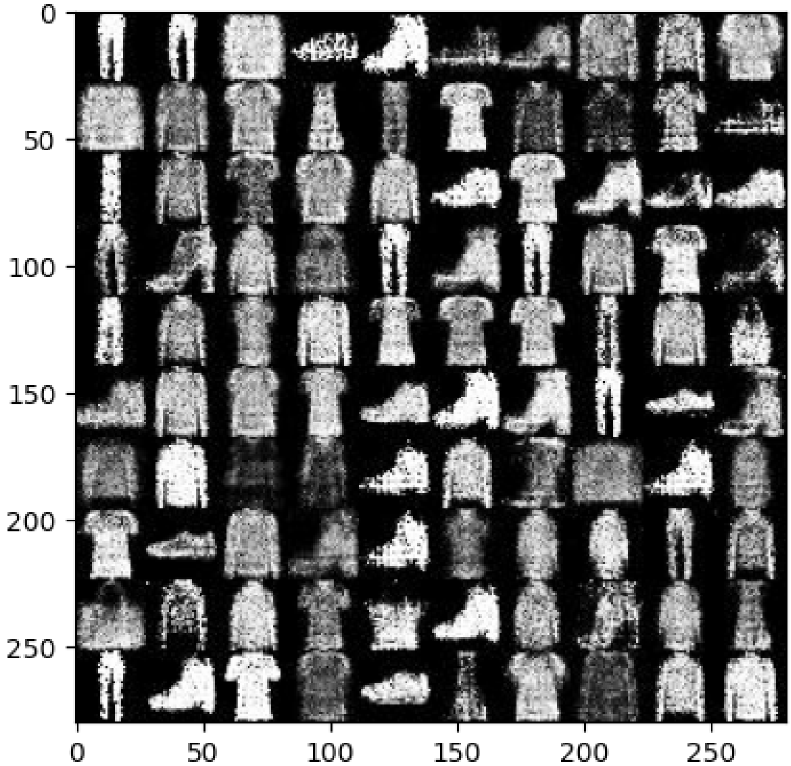


But then it gets better.

```
[ ]: 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
150
200
250
0
50
100
150
200
250