```
import numpy as np
import torch
import torchvision
import torch.nn as nn
import matplotlib.pyplot as plt


from helper_data import get_dataloaders_mnist
from helper_train import train_gan_v1
from helper_utils import set_deterministic, set_all_seeds
from helper_plotting import plot_multiple_training_losses
from helper_plotting import plot_generated_images


##########################
### SETTINGS
##########################

# Device
#CUDA_DEVICE_NUM = 3
#DEVICE = torch.device(f'cuda:{CUDA_DEVICE_NUM}' if torch.cuda.is_available() else 'cpu')
DEVICE = torch.device('cpu')
print('Device:', DEVICE)

# Hyperparameters
RANDOM_SEED = 42
GENERATOR_LEARNING_RATE = 0.0002
DISCRIMINATOR_LEARNING_RATE = 0.0002

NUM_EPOCHS = 5
BATCH_SIZE = 128

IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS = 28, 28, 1
```

```
Device: cpu
```

```
set_deterministic
set_all_seeds(RANDOM_SEED)


##########################
### Dataset
##########################

from torchvision import datasets
from torch.utils.data import DataLoader


custom_transforms = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize((0.5,), (0.5,))
])


train_dataset = datasets.MNIST(root='data',
                               train=True,
                               transform=custom_transforms,
                               download=True)

train_loader = DataLoader(dataset=train_dataset,
                          batch_size=BATCH_SIZE,
                          num_workers=0,
                          shuffle=True)

# Checking the dataset
for images, labels in train_loader:
    print('Image batch dimensions:', images.shape)
    print('Image label dimensions:', labels.shape)
    break
```

```
Image batch dimensions: torch.Size([128, 1, 28, 28])
Image label dimensions: torch.Size([128])
```

```
# Checking the dataset
print('Training Set:\n')
```

```
for images, labels in train_loader:
    print('Image batch dimensions:', images.size())
    print('Image label dimensions:', labels.size())
    #print(labels[:10])
    break
```

```
Training Set:

 Image batch dimensions: torch.Size([128, 1, 28, 28])
 Image label dimensions: torch.Size([128])
```

```
plt.figure(figsize=(8, 8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(torchvision.utils.make_grid(images[:64],
                                                    padding=2, normalize=True),
                        (1, 2, 0)))
plt.show()
```

1. This is the final model after tweaking a lot of parameters, initially I just tried with a simple Conv2d layer for generator which gave good results surprisingly but when I shifted to Transpose2d, it started throwing gibberish images even though the loss was good enough.

My choices of hyperparameters:

- Reduced the noise to a 7 * 7 image
- Used two Transpose Conv layers to get back the image of size 28 * 28
- In the discriminator I used Conv2d layers reducing the image dimensions by half and then passing it to a linear layer for logits.

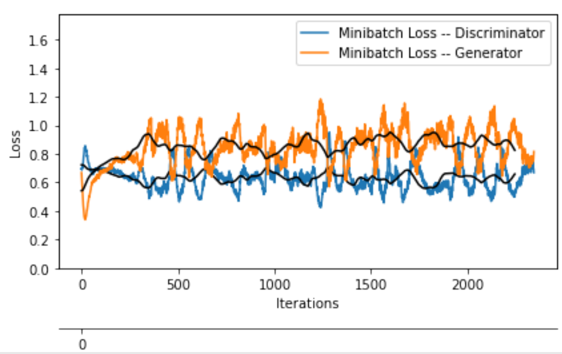
```
##########################
### MODEL
##########################
class Reshape(nn.Module):
    def __init__(self, *args):
        super().__init__()
        self.shape = args

    def forward(self, x):
        return x.view(self.shape)

class Trim(nn.Module):
    def __init__(self, *args):
        super().__init__()

    def forward(self, x):
```

```
        return x[:, :, :28, :28]

class GAN(torch.nn.Module):

    def __init__(self, latent_dim=100,
                 image_height=28, image_width=28, color_channels=1):
        super().__init__()

        self.image_height = image_height
        self.image_width = image_width
        self.color_channels = color_channels

        self.generator = nn.Sequential(
            nn.Linear(latent_dim, 7*7*color_channels),
            Reshape(-1, color_channels, 7, 7),
            nn.ConvTranspose2d(1, 16, stride=(2, 2), kernel_size=(3, 3), padding=1),
            # (f - 1) * s -2p + k
            # 7 - 1 * 2 - 2 + 3
            # 12 - 2 + 3 -> 13
            nn.LeakyReLU(0.01),
            nn.ConvTranspose2d(16, 16, stride=(2, 2), kernel_size=(6, 6), padding=1),
            nn.LeakyReLU(0.01),
            # (f - 1) * s -2p + k
            # 12 * 2 - 2 + 4
            # 24 - 2 + 6 => 28
            nn.LeakyReLU(0.01),
            nn.Flatten(),
            nn.Linear(16 * 28 * 28, image_height*image_width*color_channels),
            nn.Tanh()
        )


        self.discriminator = nn.Sequential(
            nn.Flatten(),
            # nn.Linear(image_height*image_width*color_channels, color_channels * 28 * 28 ),
            # nn.LeakyReLU(inplace=True),
            Reshape(-1, color_channels, 28 , 28 ),
            nn.Conv2d(1, 16, stride=(2, 2), kernel_size=(3, 3), padding=1),
            # 28 - 3 + 2 / 2 + 1 => 14
            nn.LeakyReLU(inplace=True),
            nn.Dropout(p=0.5),
            # Reshape(-1, color_channels, 14, 14),
            nn.Conv2d(16, 32, stride=(2, 2), kernel_size=(3, 3), padding=1),
            # (n - k + 2p) / s + 1
            # 28 - 3 + 2 / 2 + 1 => 14
            nn.LeakyReLU(inplace=True),
            nn.Dropout(p=0.5),
            nn.Flatten(),
            nn.Linear(32 * 7 * 7, 1), # outputs logits
            #nn.Sigmoid()
        )

    def generator_forward(self, z):# z has dimension NCHW
        z = torch.flatten(z, start_dim=1)
        # print(1)
        # print(z.shape)
        img = self.generator(z)
        # print(2, img.size())
        img = img.view(z.size(0),
                       self.color_channels,
                       self.image_height,
                       self.image_width)
        # print(3)
        return img

    def discriminator_forward(self, img):
        # print(img.size())
        logits = self.discriminator(img)
        # print("logits",logits.size())
        return logits


set_all_seeds(RANDOM_SEED)

model = GAN()
model.to(DEVICE)
```

```
optim_gen = torch.optim.Adam(model.generator.parameters(),
                             betas=(0.5, 0.999),
                             lr=GENERATOR_LEARNING_RATE)

optim_discr = torch.optim.Adam(model.discriminator.parameters(),
                               betas=(0.5, 0.999),
                               lr=DISCRIMINATOR_LEARNING_RATE)


log_dict = train_gan_v1(num_epochs=NUM_EPOCHS, model=model,
                        optimizer_gen=optim_gen,
                        optimizer_discr=optim_discr,
                        latent_dim=100,
                        device=DEVICE,
                        train_loader=train_loader,
                        logging_interval=100,
                        save_model='gan_mnist_01.pt')
```

```
Epoch: 001/005 | Batch 000/469 | Gen/Dis Loss: 0.6689/0.6935
Epoch: 001/005 | Batch 100/469 | Gen/Dis Loss: 0.6734/0.6993
Epoch: 001/005 | Batch 200/469 | Gen/Dis Loss: 0.7495/0.6844
Epoch: 001/005 | Batch 300/469 | Gen/Dis Loss: 0.6980/0.6845
Epoch: 001/005 | Batch 400/469 | Gen/Dis Loss: 0.9981/0.5193
Time elapsed: 2.87 min
Epoch: 002/005 | Batch 000/469 | Gen/Dis Loss: 0.6230/0.8306
Epoch: 002/005 | Batch 100/469 | Gen/Dis Loss: 0.7821/0.7036
Epoch: 002/005 | Batch 200/469 | Gen/Dis Loss: 0.6727/0.7791
Epoch: 002/005 | Batch 300/469 | Gen/Dis Loss: 0.8455/0.6527
Epoch: 002/005 | Batch 400/469 | Gen/Dis Loss: 0.8750/0.5910
Time elapsed: 5.73 min
Epoch: 003/005 | Batch 000/469 | Gen/Dis Loss: 0.9519/0.5817
Epoch: 003/005 | Batch 100/469 | Gen/Dis Loss: 0.7095/0.7558
Epoch: 003/005 | Batch 200/469 | Gen/Dis Loss: 0.8319/0.6509
Epoch: 003/005 | Batch 300/469 | Gen/Dis Loss: 1.1791/0.4425
Epoch: 003/005 | Batch 400/469 | Gen/Dis Loss: 0.9704/0.5823
Time elapsed: 8.60 min
Epoch: 004/005 | Batch 000/469 | Gen/Dis Loss: 0.8623/0.6191
Epoch: 004/005 | Batch 100/469 | Gen/Dis Loss: 0.8347/0.6021
Epoch: 004/005 | Batch 200/469 | Gen/Dis Loss: 0.9429/0.6563
Epoch: 004/005 | Batch 300/469 | Gen/Dis Loss: 0.8098/0.7262
Epoch: 004/005 | Batch 400/469 | Gen/Dis Loss: 0.9537/0.5120
Time elapsed: 11.46 min
Epoch: 005/005 | Batch 000/469 | Gen/Dis Loss: 0.8814/0.5779
Epoch: 005/005 | Batch 100/469 | Gen/Dis Loss: 0.7777/0.7122
Epoch: 005/005 | Batch 200/469 | Gen/Dis Loss: 0.8020/0.7074
Epoch: 005/005 | Batch 300/469 | Gen/Dis Loss: 0.7075/0.7666
Epoch: 005/005 | Batch 400/469 | Gen/Dis Loss: 0.9388/0.6219
Time elapsed: 14.32 min
Total Training Time: 14.32 min
```

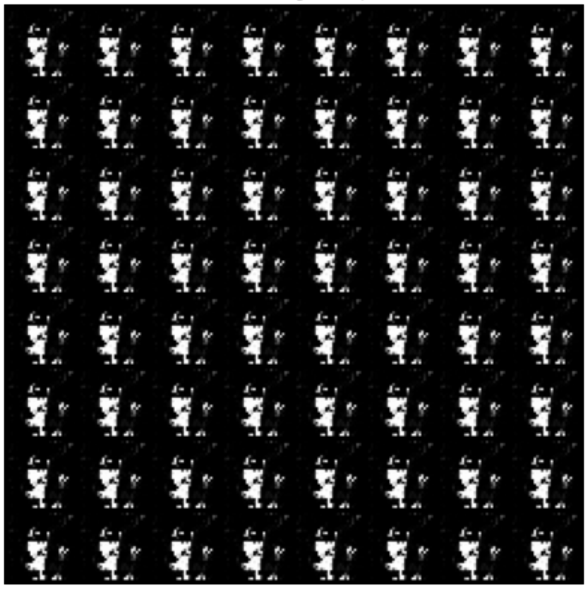
```
plot_multiple_training_losses(
    losses_list=(log_dict['train_discriminator_loss_per_batch'],
                 log_dict['train_generator_loss_per_batch']),
    num_epochs=NUM_EPOCHS,
    custom_labels_list=(' -- Discriminator', ' -- Generator')
)
```



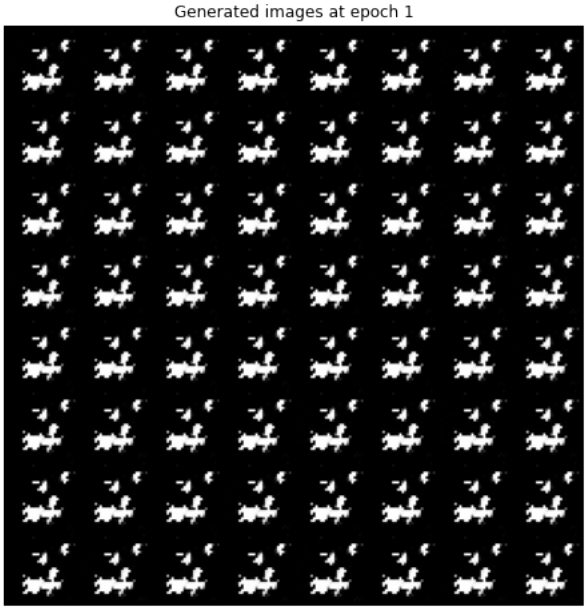

```
##########################
### VISUALIZATION
##########################

# for i in range(0, NUM_EPOCHS, 5):
for i in range(0 NUM_EPOCHS):
```

```
for i in range(0,NUM_EPOCHS):

    plt.figure(figsize=(8, 8))
    plt.axis('off')
    plt.title(f'Generated images at epoch {i}')
    plt.imshow(np.transpose(log_dict['images_from_noise_per_epoch'][i], (1, 2, 0)))
    plt.show()


plt.figure(figsize=(8, 8))
plt.axis('off')
plt.title(f'Generated images after last epoch')
plt.imshow(np.transpose(log_dict['images_from_noise_per_epoch'][-1], (1, 2, 0)))
plt.show()
```

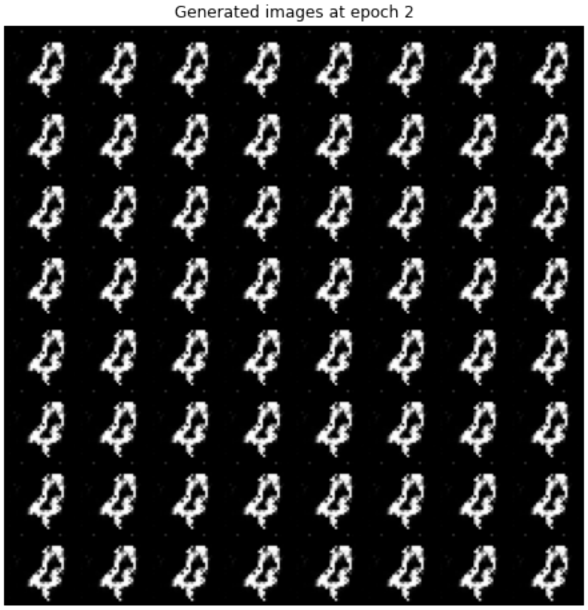
Generated images at epoch 0



Generated images at epoch 1



Generated images at epoch 2



Generated images at epoch 3