Lab: DCGAN

Lab Objective:

In this lab, you will learn the classical form of GAN loss, and you will need to implement DCGAN, which apply convolutional neural networks to be the generator and de-convolutional neural networks to be the discriminator.

Turn in:

Report:

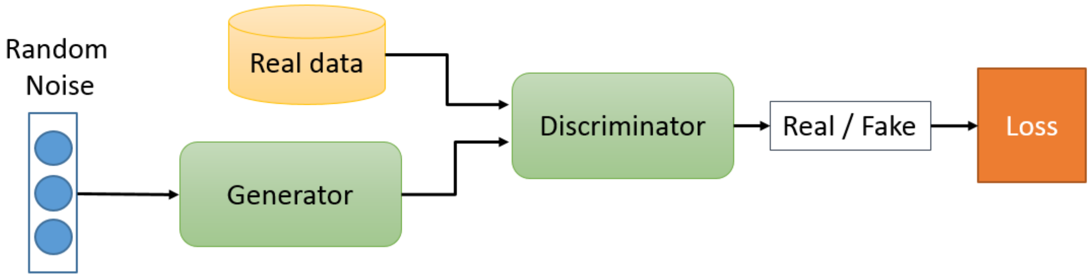
Demo:

Requirements:

1. Implement DCGAN
2. Show generated images
3. Plot the loss function of the generator and the discriminator

Implementation Details:

- Generative Adversarial Networks (GAN)
  1. GAN consists of two major components: a generator and a discriminator. The discriminator, which is a binary classifier, tries to distinguish fake inputs from real inputs, while the generator tries to fool the discriminator.



  2. The loss function of discriminator:

$$\mathcal{L}_{\mathrm{D}} = -E_{x\sim p_r}[\log D(x)] - E_{x\sim p_g}[\log(1 - D(x))]$$

  3. There are two most often used loss function of generator. The first one is:

$$\mathcal{L}_{\mathrm{G}} = -\mathcal{L}_D = \mathrm{E}_{\mathrm{x\sim p_g}}[\log(1 - D(x))]$$

The second one is:

$$\mathcal{L}_{\mathrm{G}} = E_{x\sim p_g}[-\log D(x)]$$

In this lab, you can use either of them, but you should tell me which one you used in your report.

4. The training of GAN usually suffers instability. There are a lot of papers discussing about why it happens. For more details, you can refer to the reference section.

- Deep Convolutional Generative Adversarial Networks (DCGAN)
  - DCGAN is an extension of classical GAN. The main contributions of DCGAN includes:
    1. Remove all fully-connected layers
    2. Add de-convolutional layers to generator (https://datascience.stackexchange.com/questions/6107/what-are-deconvolutional-layers)
    3. Add batch normalize layers to both generators and discriminators
    4. Replace pooling layers with stride convolutional layers
    5. For more details, please refer to the reference

- Loading Dataset

1. CelebA_aligned_reduced.h5
2. Using celebA_dataset.py to load data

```python
class CelebADataset(Dataset):
    def __init__(self, h5_path, transform=None):
        assert (os.path.isfile(h5_path))
        self.h5_path = h5_path
        self.transform = transform

        # loading the dataset into memory
        f = h5py.File(self.h5_path, "r")
        key = list(f.keys())
        print ("key list:", key)
        self.dataset = f[key[0]]
        print ("dataset loaded and its shape:", self.dataset.shape)

    def __getitem__(self, index):
        img = self.dataset[index]
        img = np.transpose(img, (1, 2, 0))
        if self.transform is not None:
            img = self.transform(img)

        return img, 0

    def __len__(self):
        return len(self.dataset)
```

3. The usage of this dataset is the same as MNIST or other datasets we used before

```python
batch_size = 64
T = transforms.Compose([transforms.ToTensor(),
                        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),])
train_loader = torch.utils.data.DataLoader(
    CelebADataset('../dataset/CelebA_aligned.h5', transform=T),
    batch_size=batch_size, shuffle=True, num_workers=1)
```

- Model Architecture.
  - Generator

Name	Type	Input	Kernel Size	Stride	Input Shape
Deconv1	Transpose convolution	Input noise	4	1	(batch size, nz, 1, 1)
BN1	Batch Norm	Deconv1			(batch size, ngf*8, 4, 4)
ReLU1	ReLU	BN1			(batch size, ngf*8, 4, 4)
Deconv2	Transpose convolution	Relu1	4	2	(batch size, ngf*8, 4, 4)
BN2	Batch Norm	Deconv2			(batch size, ngf*4, 8, 8)
ReLU2	ReLU	BN2			(batch size, ngf*4, 8, 8)
Deconv3	Transpose convolution	ReLU2	4	2	(batch size, ngf*4, 8, 8)
BN3	Batch Norm	Deconv3			(batch size, ngf*2, 16, 16)
ReLU3	ReLU	BN3			(batch size, ngf*2, 16, 16)
Deconv4	Transpose convolution	ReLU3	4	2	(batch size, ngf*2, 16, 16)
BN4	Batch Norm	Deconv4			(batch size, ngf, 32, 32)
ReLU4	ReLU	BN4			(batch size, ngf, 32, 32)
Deconv5	Transpose convolution	ReLU4	4	2	(batch size, ngf, 32, 32)
Output	Tanh	Deconv5			(batch size, 3, 64, 64)

- Discriminator

Type	Input	Kernel Size	Stride	Padding	Input Shape	Type
Conv1	Convolution	Input noise	4	2	1	(batch size, 3, 64, 64)
LeakyReLU1	LeakyReLU	Conv1				(batch size, ndf, 32, 32)
Conv2	Convolution	LeakyReLU1	4	2	1	(batch size, ndf, 32, 32)
BN2	Batch Norm	Conv2				(batch size, ndf*2, 16, 16)
LeakyReLU2	LeakyReLU	BN2				(batch size, ndf*2, 16, 16)
Conv3	Convolution	LeakyReLU2	4	2	1	(batch size, ndf*2, 16, 16)
BN3	Batch Norm	Conv3				(batch size, ndf*4, 8, 8)
LeakyReLU3	LeakyReLU	BN3				(batch size, ndf*4, 8, 8)
Conv4	Convolution	LeakyReLU3	4	2	1	(batch size, ndf*4, 8, 8)
BN4	Batch Norm	Conv4				(batch size, ndf*8, 4, 4)
LeakyReLU4	LeakyReLU	BN4				(batch size, ndf*8, 4, 4)
Conv5	Convolution	LeakyReLU4	4	1	0	(batch size, ndf*8, 4, 4)
Output	Sigmoid	Conv5				(batch size, 1, 1, 1)

- Hyper-parameters
  1. Batch size: 64
  2. Learning rate: 2e-4
  3. nz = 100
  4. ngf = 64
  5. ndf = 64
  6. Total epochs = 25
  7. Optimizer: Adam

Reference:

1. Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in neural information processing systems*. 2014. http://papers.nips.cc/paper/5423-generative-adversarial-nets
2. Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." *arXiv preprint arXiv:1511.06434* (2015). https://arxiv.org/abs/1511.06434
3. A quick introduce to de-convolutional neural networks: https://datascience.stackexchange.com/questions/6107/what-are-deconvolutional-layers
4. Arjovsky, Martin, Soumith Chintala, and Léon Bottou. "Wasserstein gan." *arXiv preprint arXiv:1701.07875* (2017). https://arxiv.org/abs/1701.07875
5. Arjovsky, Martin, and Léon Bottou. "Towards principled methods for training generative adversarial networks." *arXiv preprint arXiv:1701.04862* (2017). https://arxiv.org/abs/1701.04862
6. Pytorch DCGAN example: https://github.com/pytorch/examples/tree/master/dcgan

Report Spec [black: Demo, Gray: No Demo]:

1. Introduction (15%, 15%)
2. Experiment setups: (20%, 20%)
   A. How you implement DCGAN
   B. Which loss function of generator you used
3. Results:
   A. Results of your samples (20%, 30%)
4. Discussion (25%, 35%)
5. Demo (20%)