Lab: VAE-GAN

Lab Objective:

In this lab, you will combine adversarial loss and VAE to build a more powerful model. Precisely speaking, you will be asked to reproduce the paper "Autoencoding beyond pixels using a learned similarity metric".

Turn in:

Report:

Demo:

Requirements:

1. Implement VAE-GAN
2. Show reconstruction images and generated samples
3. Compare your results of hw3 and hw2 and discuss visual differences

Implementation Details:

- 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

```
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)
```

- VAE (beta-VAE)
  - The same concepts as we introduced before
  - Loss function:

$$\mathcal{F}(\theta, \phi, \beta; \mathbf{x}, \mathbf{z}) \geq \mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z}, \beta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta\, D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

- GAN
  - The same concepts as we introduced before
  - The loss function of discriminator:

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

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

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

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.

- VAE-GAN

1. Better distance metrics
   - Discriminator should learn better metrics than pixel-wise l2-norm or l1-norm metrics
   - Replace pixel-wise distance metrics with feature-wise distance metrics which are learned by the discriminator
   - Extracts the representations of x and $\tilde{x}$ from the *L* layer of discriminator
   - Introduce a Gaussian observation model:

     $$p(\mathrm{Dis}_l(\boldsymbol{x})|\boldsymbol{z}) = \mathcal{N}(\mathrm{Dis}_l(\boldsymbol{x})|\,\mathrm{Dis}_l(\tilde{\boldsymbol{x}}), \mathbf{I})$$

     And the original reconstruction loss becomes:

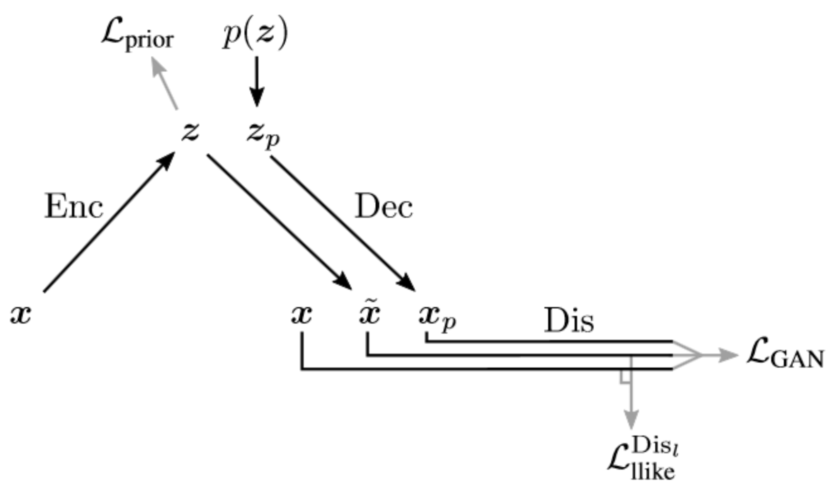
     $$\mathcal{L}_{\mathrm{llike}}^{\mathrm{Dis}_l} = -\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\mathrm{Dis}_l(\boldsymbol{x})|\boldsymbol{z})\right]$$

     In short, you should compute the l2 norm (MSE error) of the feature-wise difference between x and $\tilde{x}$, rather than the l2 norm of the pixel-wise difference

2. Adversarial loss
   - The whole autoencoder can be seen as a large generator, and we can introduce an additional neural network as discriminator. Thus, our entire system becomes a GAN
   - To improve the quality of both reconstruction images and sampling images, we take both reconstruction images and sampling images as fake inputs of the discriminator

     $$\begin{aligned}\mathcal{L}_{\mathrm{GAN}} = \ & \log(\mathrm{Dis}(\boldsymbol{x})) + \log(1 - \mathrm{Dis}(\mathrm{Dec}(\boldsymbol{z}))) \\ & + \log(1 - \mathrm{Dis}(\mathrm{Dec}(\mathrm{Enc}(\boldsymbol{x}))))\end{aligned}$$

3. VAE-GAN and its algorithm



Algorithm 1 Training the VAE/GAN model

$\boldsymbol{\theta}_{\text{Enc}}, \boldsymbol{\theta}_{\text{Dec}}, \boldsymbol{\theta}_{\text{Dis}} \leftarrow$ initialize network parameters
repeat
$\quad \boldsymbol{X} \leftarrow$ random mini-batch from dataset
$\quad \boldsymbol{Z} \leftarrow \text{Enc}(\boldsymbol{X})$
$\quad \mathcal{L}_{\text{prior}} \leftarrow D_{\text{KL}}(q(\boldsymbol{Z}|\boldsymbol{X})\|p(\boldsymbol{Z}))$
$\quad \tilde{\boldsymbol{X}} \leftarrow \text{Dec}(\boldsymbol{Z})$
$\quad \mathcal{L}_{\text{llike}}^{\text{Dis}_l} \leftarrow -\mathbb{E}_{q(\boldsymbol{Z}|\boldsymbol{X})}\left[p(\text{Dis}_l(\boldsymbol{X})|\boldsymbol{Z})\right]$
$\quad \boldsymbol{Z}_p \leftarrow$ samples from prior $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
$\quad \boldsymbol{X}_p \leftarrow \text{Dec}(\boldsymbol{Z}_p)$
$\quad \mathcal{L}_{\text{GAN}} \leftarrow \log(\text{Dis}(\boldsymbol{X})) + \log(1 - \text{Dis}(\tilde{\boldsymbol{X}})) + \log(1 - \text{Dis}(\boldsymbol{X}_p))$
$\quad$ // Update parameters according to gradients
$\quad \boldsymbol{\theta}_{\text{Enc}} \overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Enc}}}(\mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{llike}}^{\text{Dis}_l})$
$\quad \boldsymbol{\theta}_{\text{Dec}} \overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Dec}}}(\gamma \mathcal{L}_{\text{llike}}^{\text{Dis}_l} - \mathcal{L}_{\text{GAN}})$
$\quad \boldsymbol{\theta}_{\text{Dis}} \overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Dis}}} \mathcal{L}_{\text{GAN}}$
until deadline

4. We modify the loss function of encoder a little bit:

   Loss function for the encoder: $\beta * \text{L}_{\text{prior}} + L_{llike}^{Dis_l}$

- Model Architecture
  - Size of input images: (batch size, 3, 64, 64)
  - Size of Latent codes (Gaussian noise): (batch size, 2048)
  - Encoder

Name	Type	Input	Kernel size	Stride	Padding	Output shape
Conv1	Conv	Input images	5	2	2	(batch size, 64, 32, 32)
Conv2	Conv	Conv1	5	2	2	(batch size, 128, 16, 16)
Conv3	Conv	Conv2	5	2	2	(batch size, 256, 8, 8)
Flatten	Flatten	Conv3				(batch size, 256*8*8)
Fc1	FC	Flatten				(batch size, 2048)
BN1	BN	Fc1				(batch, 2048)
ReLU1	ReLU	BN1				(batch, 2048)
Mean	FC	ReLU1				(batch, 2048)
Fc2	FC	Conv3	0			(batch size, 2048)
BN2	BN	Fc2				(batch, 2048)

ReLU2	ReLU	BN2				(batch, 2048)
Log Var	FC	ReLU2				(batch, 2048)

- Decoder

Name	Type	Input	Kernel size	Stride	Padding	Output shape
Fc1	FC	Latent codes				(batch size, 8*8*256)
BN1	BN	Fc1				(batch size, 8*8*256)
ReLU1	ReLU	BN1				(batch size, 8*8*256)
Reshape	Reshape	ReLU1				(batch size, 256, 8, 8)
Deconv1	Deconv	Reshape	5	2	2	(batch size, 256, 16, 16)
Deconv2	Deconv	Deconv1	5	2	2	(batch size, 128, 32, 32)
Deconv3	Deconv	Deconv2	5	2	2	(batch size, 32, 64, 64)
Deconv4	Deconv	Deconv3	5	1	2	(batch size, 3, 64, 64)
Tanh()	Tanh	Deconv4				(batch size, 3, 64, 64)

```
output = self.deconv1(preprocessed_codes, output_size=(bs, 256, 16, 16))
output = self.deconv2(output, output_size=(bs, 128, 32, 32))
output = self.deconv3(output, output_size=(bs, 32, 64, 64))
output = self.deconv4(output, output_size=(bs, 3, 64, 64))
```

- Discriminator

Name	Type	Input	Kernel size	Stride	Padding	Output shape
Conv1	Conv	Real/fake images	5	1	2	(batch size, 32, 64, 64)
Conv2	Conv	Conv1	5	2	2	(batch size, 128, 32, 32)
Conv3	Conv	Conv2	5	2	2	(batch size, 256, 16, 16)
Conv4	Conv	Conv3	5	2	2	(batch size, 256, 8, 8)
Reshape	Reshape	Conv4				(batch size, 256*8*8)
Fc1	FC	Reshape				(batch size, 1024)
ELU1	ELU	Fc1				(batch size, 1024)
Fc2	FC	ELU1				(batch size, 1)
Sigmoid	Sigmoid	Fc2				(batch size, 1)

- Hyper-parameters
  1. Max epochs : 50
  2. RMSprop
  3. Learning rate for the encoder and the decoder: 3e-4
  4. Learning rate for the discriminator: 3e-5

5. Batch size: 64
6. Hidden code size: 2048
7. Gamma : 5 (enhance the reconstruction loss for the decoder)
8. Beta: 15 (enhance the reconstruction loss for the encoder)

- Rule of Thumb
  - You may need 3~5 hours to implement this homework
  - Total training time will cost you around 10 hours

Reference:

1. Larsen, Anders Boesen Lindbo, et al. "Autoencoding beyond pixels using a learned similarity metric." *arXiv preprint arXiv:1512.09300* (2015). https://arxiv.org/pdf/1512.09300.pdf

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

1. Introduction (15%, 15%)
2. Experiment setups:
   A. How you implement VAE-GAN (10%, 10%)
   B. Which loss function of generator you used (10%, 10%)
3. Results:
   A. Show reconstruction images and generated samples (20%, 30%)
4. Discussion
   A. Compare your results of hw3 and hw2 and discuss visual differences (25%, 35%)
5. Demo (20%)