

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: 12/12

Demo: 12/13

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. Download dataset from google drive

(https://drive.google.com/file/d/0B512lwbc5_ZFcVFfeXBOTzZ4WkE/view?usp=sharing)

2. Using celebA_dataset.py to load data

```
1 class CelebADataset(Dataset):
2     def __init__(self, h5_path, transform=None):
3         assert (os.path.isfile(h5_path))
4         self.h5_path = h5_path
5         self.transform = transform
6
7         # loading the dataset into memory
8         f = h5py.File(self.h5_path, "r")
9         key = list(f.keys())
10        print ("key list:", key)
11        self.dataset = f[key[0]]
12        print ("dataset loaded and its shape:", self.dataset.shape)
13
14    def __getitem__(self, index):
15        img = self.dataset[index]
16        img = np.transpose(img, (1, 2, 0))
17        if self.transform is not None:
18            img = self.transform(img)
19
20        return img, 0
21
22    def __len__(self):
23        return len(self.dataset)
```

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

```
1 batch_size = 64
2 T = transforms.Compose([transforms.ToTensor(),
3                           transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),])
4 train_loader = torch.utils.data.DataLoader(
5     CelebADataset('../dataset/CelebA_aligned.h5', transform=T),
6     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:

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

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

$$\mathcal{L}_G = -\mathcal{L}_D = E_{x \sim p_g}[\log(1 - D(x))]$$

The second one is:

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

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:

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

And the original reconstruction loss becomes:

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

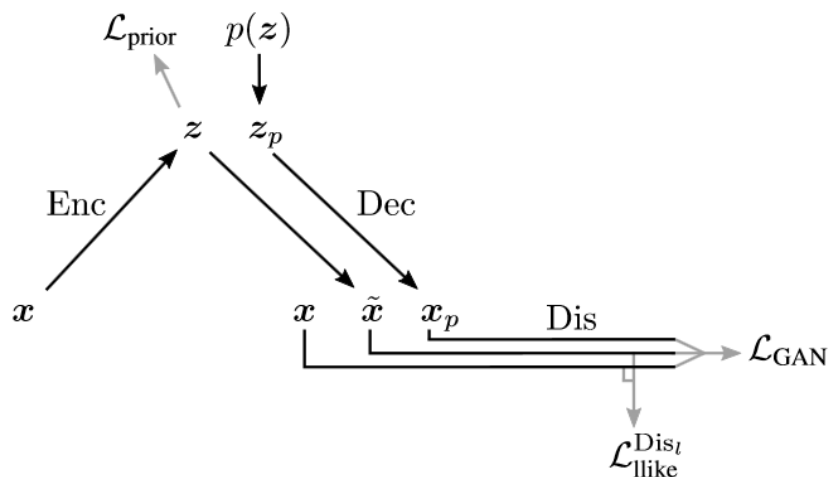
In short, you should compute the l2 norm of the feature-wise difference between x and \tilde{x} , rather than the l2 norm of the pixel-wise difference

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

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

3. VAE-GAN and its algorithm



Algorithm 1 Training the VAE/GAN model

 $\theta_{\text{Enc}}, \theta_{\text{Dec}}, \theta_{\text{Dis}} \leftarrow$ initialize network parameters**repeat** $\mathbf{X} \leftarrow$ random mini-batch from dataset $\mathbf{Z} \leftarrow \text{Enc}(\mathbf{X})$ $\mathcal{L}_{\text{prior}} \leftarrow D_{\text{KL}}(q(\mathbf{Z}|\mathbf{X})||p(\mathbf{Z}))$ $\tilde{\mathbf{X}} \leftarrow \text{Dec}(\mathbf{Z})$ $\mathcal{L}_{\text{llike}}^{\text{Dis}_l} \leftarrow -\mathbb{E}_{q(\mathbf{Z}|\mathbf{X})} [p(\text{Dis}_l(\mathbf{X})|\mathbf{Z})]$ $\mathbf{Z}_p \leftarrow$ samples from prior $\mathcal{N}(\mathbf{0}, \mathbf{I})$ $\mathbf{X}_p \leftarrow \text{Dec}(\mathbf{Z}_p)$ $\mathcal{L}_{\text{GAN}} \leftarrow \log(\text{Dis}(\mathbf{X})) + \log(1 - \text{Dis}(\tilde{\mathbf{X}}))$
 $+ \log(1 - \text{Dis}(\mathbf{X}_p))$

// Update parameters according to gradients

 $\theta_{\text{Enc}} \stackrel{+}{\leftarrow} -\nabla_{\theta_{\text{Enc}}} (\mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{llike}}^{\text{Dis}_l})$ $\theta_{\text{Dec}} \stackrel{+}{\leftarrow} -\nabla_{\theta_{\text{Dec}}} (\gamma \mathcal{L}_{\text{llike}}^{\text{Dis}_l} - \mathcal{L}_{\text{GAN}})$ $\theta_{\text{Dis}} \stackrel{+}{\leftarrow} -\nabla_{\theta_{\text{Dis}}} \mathcal{L}_{\text{GAN}}$ **until** deadline

4. TBD

● 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)
Reshape	Reshape	Conv3				(batch size, 256*8*8)
Fc1	FC	Reshape				(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. Learning rate : 3e-4
3. Batch size: 64
4. Hidden code size: 2048
5. Gamma : 5
6. Beta = 15
7. Alpha = 0.1

● Rule of Thumb

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



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]:

(Please write all your report in the same file)

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 (reconstruction and generated samples) and hw2 and discuss visual differences (25%, 35%)
5. Demo (20%)