Lab 4: InfoGAN

● Lab objective

In this lab, you will learn the classical form of GAN loss, and you will need to implement InfoGAN trained on MNIST, which adopts adversarial loss to generate realistic images and learns the disentangled representations by maximizing mutual information.

● Important Date

1. Deadline: 12/29 (Wed.) 11:59 a.m.
2. Demo date: 12/29 (Wed.)

● Turn in

1. Experimental Report (.pdf) and Source code.

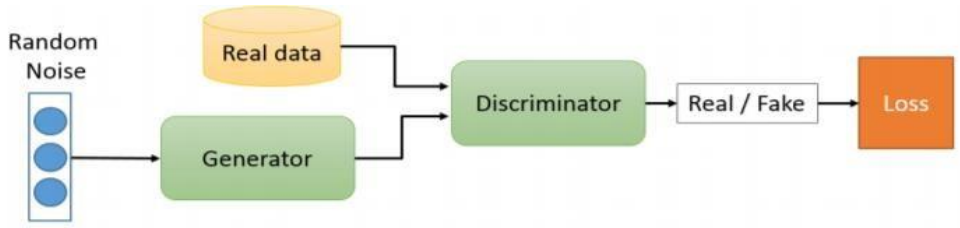
Notice: zip all files in one file and name it like DLP_LAB4_yourID_name.zip, and email it to jshuang.cs09@nycu.edu.tw with subject MTK_DLP_LAB4_yourID_name.

● Requirements

1. Modify the generator and the discriminator to InfoGAN.
2. Adopt traditional generator and discriminator loss. Maximize the mutual information between generated images and discrete one-hot vector.
3. Show the generated images of the specific number.
4. Plot the loss curves of the generator and the discriminator while training.

● Implementation details

1. Generative Adversarial Network (GAN)
   A. 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.



   B. The loss function of the discriminator

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

C. The loss function of the generator

$$\begin{cases} L_G = \log(1 - D(G(z)) \\ L_G = -\log(D(G(z)) \end{cases}$$

2. Deep Convolution Generative Adversarial Network (DCGAN)

A. DCGAN is an extension of classical GAN. The main contributions of DCGAN includes:

- Remove all fully-connected layers
- Add de-convolutional layers to generator (https://datascience.stackexchange.com/questions/6107/whataredeconvolutional-layers)
- Add batch normalize layers to both generators and discriminators
- Replace pooling layers with stride convolutional layers
- For more details, please refer to the reference

B. Sample code: https://github.com/pytorch/examples/tree/master/dcgan

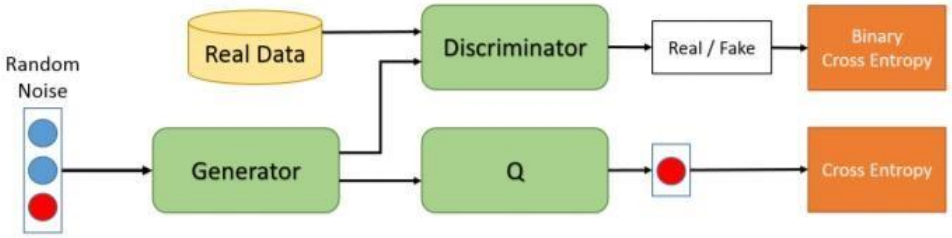
C. Sample Architecture:

You can modify this architecture to complete your assignment.

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(64, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.01)
    (6): ConvTranspose2d(256, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.01)
    (9): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): LeakyReLU(negative_slope=0.01)
    (12): ConvTranspose2d(64, 1, kernel_size=(2, 2), stride=(2, 2), bias=False)
    (13): Sigmoid()
  )
)
```

```
Discriminator(
  (main): Sequential(
    (0): Conv2d(1, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.1, inplace)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.1, inplace)
    (5): Conv2d(128, 256, kernel_size=(7, 7), stride=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.1, inplace)
  )
  (D): Sequential(
    (0): Conv2d(256, 1, kernel_size=(1, 1), stride=(1, 1))
    (1): Sigmoid()
  )
)
```

3. InfoGAN (https://arxiv.org/abs/1606.03657)
   A. Overview



   B. Random noise in this lab consists of:
      - 54-D continuous noise drawn from standard normal distribution (zero mean and unit variance)
      - 10-D discrete one hot vector. E.g., (0, 1, 0, 0, 0, 0, 0, 0, 0, 0)

   C. Maximizing mutual information to force models to use useful information

$$
\begin{aligned}
I(c; G(z,c)) &= H(c) - H(c|G(z,c)) \\
&= \mathbb{E}_{x\sim G(z,c)}[\mathbb{E}_{c'\sim P(c|x)}[\log P(c'|x)]] + H(c) \\
&= \mathbb{E}_{x\sim G(z,c)}[\underbrace{D_{\mathrm{KL}}(P(\cdot|x) \parallel Q(\cdot|x))}_{\geq 0} + \mathbb{E}_{c'\sim P(c|x)}[\log Q(c'|x)]] + H(c) \\
&\geq \mathbb{E}_{x\sim G(z,c)}[\mathbb{E}_{c'\sim P(c|x)}[\log Q(c'|x)]] + H(c)
\end{aligned}
$$

   D. Lower bound of the mutual information

$$
\begin{aligned}
L_I(G,Q) &= \boxed{E_{c\sim P(c),x\sim G(z,c)}[\log Q(c|x)] + H(c)} \\
&= E_{x\sim G(z,c)}[\mathbb{E}_{c'\sim P(c|x)}[\log Q(c'|x)]] + H(c) \\
&\leq I(c; G(z,c))
\end{aligned}
$$

   E. The loss function of the generator now becomes:

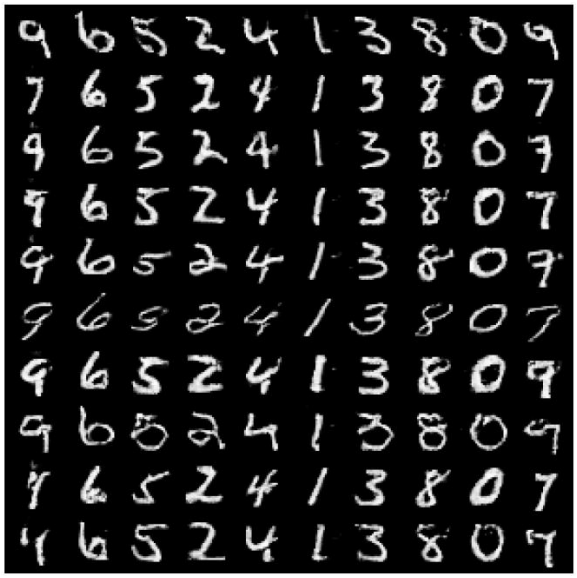
$$
\mathcal{L}_G = E_{x\sim p_g}[-\log D(x)] + L_I(G,Q)
$$

Hyper-parameters

1. noise size = 54
2. latent code size = 10
3. total epochs = 50
4. optimizer : Adam

You can adjust the hyper-parameters according to your own ideas.

Output example



Scoring Criteria

1. Report (50%)
   A. Introduction (10%)
   B. Experiment Setups (20%)
      - How did you implement Info GAN
        - Adversarial loss
        - Maximizing mutual information
      - Which loss function of generator (refer to Implementation Details C.) did you use? What's the difference?
   C. Results and discussion (20%)
      - Results of your samples
      - Training loss curves

2. Demo (50%)
   A. Generate 10 numbers with 10 different styles (noises). A column is correct if there are ≥ 7 correct numbers in that column, and your total score is decided according to the total number of correct columns. (30%)
      - 10 correct columns ---- 100%
      - 9 correct columns ---- 90%
      - ≥ 7 correct columns ---- 80%
      - ≥ 5 correct columns ---- 60%
      - Otherwise ---- 0%
   B. Questions (20%)