Lab: 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.

Turn in:

Report: 5/23 11:59 a.m.

Demo: 5/23

Dataset:

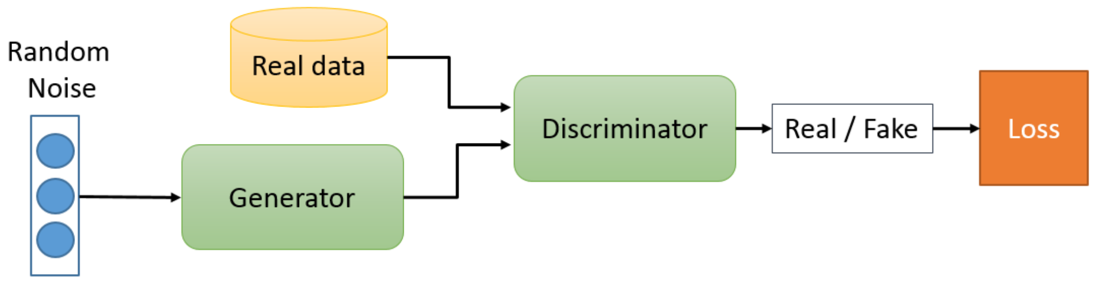
1. MNIST

Requirements:

1. Modify DCGAN (https://github.com/pytorch/examples/tree/master/dcgan) to implement InfoGAN which maximizes mutual information between generated images and discrete one hot vector
2. Show generated images (shown as in the expected results section)
3. Plot the loss curve 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:

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

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

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

The second one is:

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

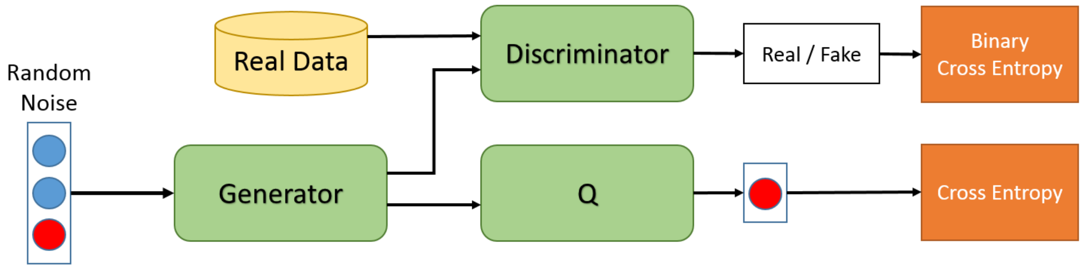
In this lab, you can use either of them, but you should tell me which one you use in your report.

4. Training of GAN usually suffers instability. There are a lot of papers discussing 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

  - Sample codes: https://github.com/pytorch/examples/tree/master/dcgan

- InfoGAN
  - Overview



  - Random noise in this lab consists of:
    1. 54-D continuous noise drawn from standard normal distribution (zero mean and unit variance)
    2. 10-D discrete one hot vector. E.q., (0, 1, 0, 0, 0, 0, 0, 0, 0, 0).
  - Maximizing mutual information to force models to use useful information

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

- Lower bound of mutual information

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

- Loss function of the generator becomes:

$$\mathcal{L}_{\mathrm{G}} = E_{x\sim p_r}[\log D(x)] + \mathrm{E}_{\mathrm{x}\sim \mathrm{p_g}}[\log(1 - D(x))] + L_I(G,Q)$$

or

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

- Sample codes:
  1. https://github.com/pianomania/infoGAN-pytorch

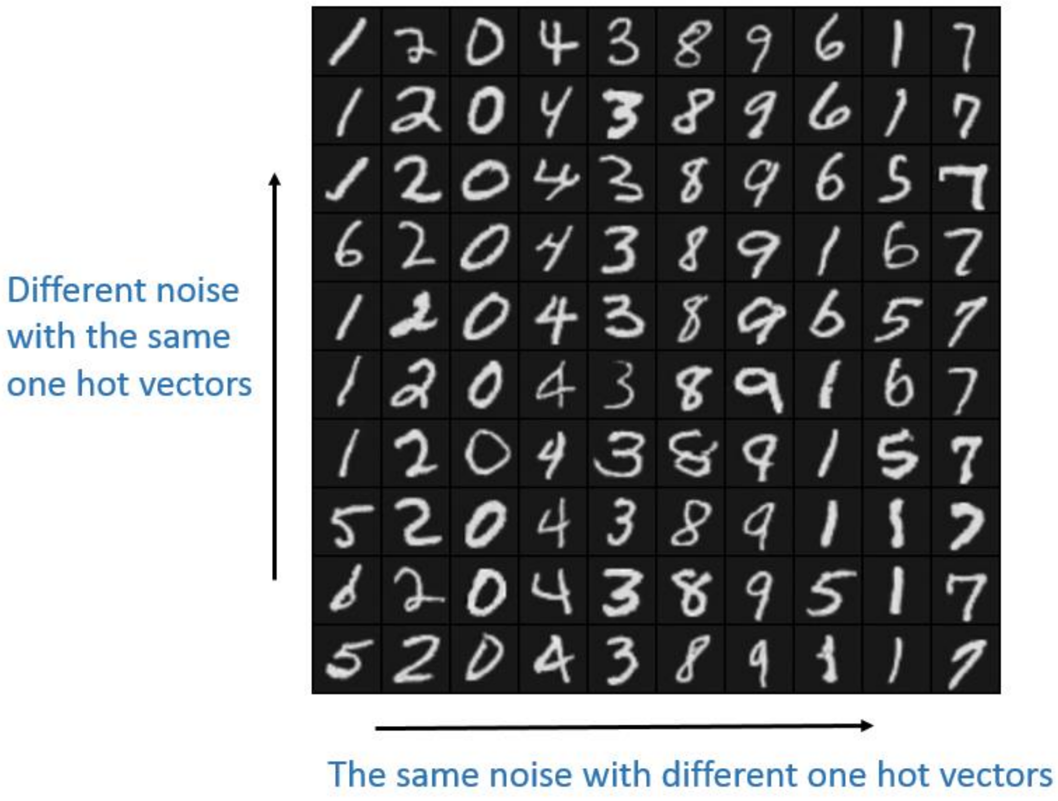
- Model Architecture.
  - Discriminator and Q can share the same feature extractor
  - Generator

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

- Discriminator and Q

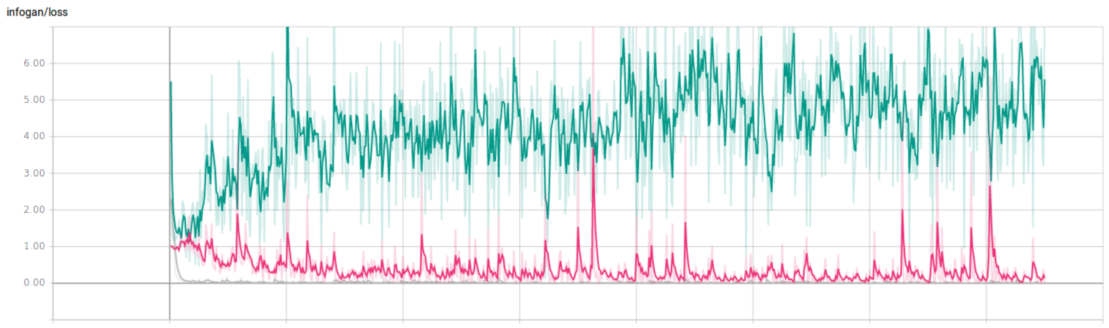
```
_netD(
  (main): Sequential(
    (0): Conv2d(1, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(0.2, inplace)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (4): LeakyReLU(0.2, inplace)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
    (7): LeakyReLU(0.2, inplace)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
    (10): LeakyReLU(0.2, inplace)
  )
  (discriminator): Sequential(
    (0): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): Sigmoid()
  )
  (Q): Sequential(
    (0): Linear(in_features=8192, out_features=100, bias=True)
    (1): ReLU()
    (2): Linear(in_features=100, out_features=10, bias=True)
  )
)
```

- Expected Outputs
  - Generated images

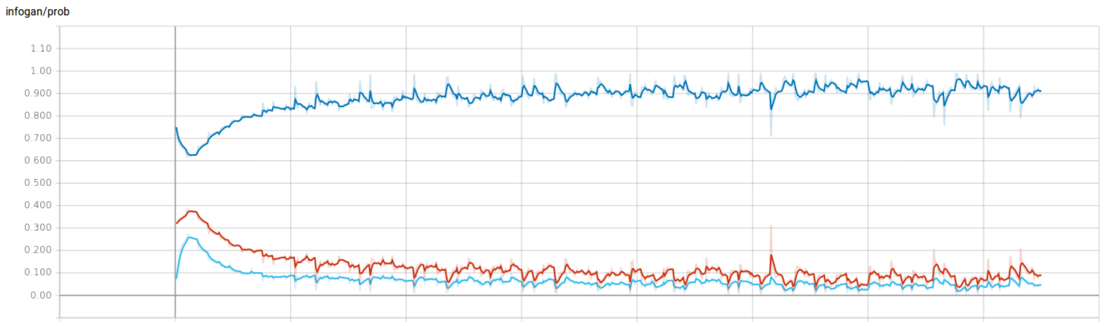

Different noise with the same one hot vectors
The same noise with different one hot vectors

■ Training loss curves (every 100 batch steps)

1. Loss of the generator, the discriminator, and the Q



2. Probability of real data, fake data before updating G and after updating G



● Hyper-parameters

1. Batch size: 64
2. Learning rate for the discriminator: 2e-4
3. Learning rate for the generator and Q: 1e-3
4. nz = 64
5. c_size (size of meaningful codes) = 10
6. ngf = 64
7. ndf = 64
8. Total epochs = 80
9. 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

Report Spec:

1. Introduction (5%)
2. Experiment setups: (20%)
   A. How you implement InfoGAN
      i. Adversarial loss
      ii. Maximizing mutual information
      iii. How you generate fixed noise and images
   B. Which loss function of generator you used
3. Results (30%):
   A. Results of your samples (shown as in the expected results section)
   B. Training loss curves
4. Discussion (15%)
5. Demo (30%)
   A. Given a label, you have to generate corresponding images