2019 Deep Learning and Practice
Lab 6 – InfoGAN

0756110 李東霖

May 21, 2019

1 Introduction

DCGAN can generate hand-written digit image but it can't control generator to generate image with specific digit number. To solve that situation, need use InfoGAN to learn condition from noise.

Lab's requirements as follows:

- Implement InfoGAN based on DCGAN (Deep Convolution GAN)
- Show generated images
- Plot loss curve of generator, discriminator, Q
- Plot probability curve of read datas, fake datas before updating and fake datas after updating

2 Experiment setup

2.1 Implement InfoGAN

2.1.1 Adversarial loss

The GAN has two part. One is discriminator, another is generator. Discriminator classify images x into real image and fake image. Generator yields image x from noise z.

The loss of discriminator as follows:

$$\text{loss}_D = -E_{x\sim P_{data}(x)}(\log D(x;\theta_D)) - E_{z\sim P_z(z)}(\log(1 - D(G(z;\theta_G);\theta_D))) \quad (1)$$

The loss of generator has two options as follows:

$$\text{loss}_G = -\text{loss}_D \tag{2}$$
$$= -E_{z\sim P_z(z)}(\log D(G(z;\theta_G);\theta_D)) \tag{3}$$

If loss equation of discriminator (2) goes down, it means fake images from generator are more similar as real. But $\log(1 - D(G(z;\theta_G);\theta_D))$ in equation 2 will lead to small gradient and slow convergence in generator. Because $D(G(z))$ usually is zero in early epoch and gradient near by zero is smaller than gradient near by one.

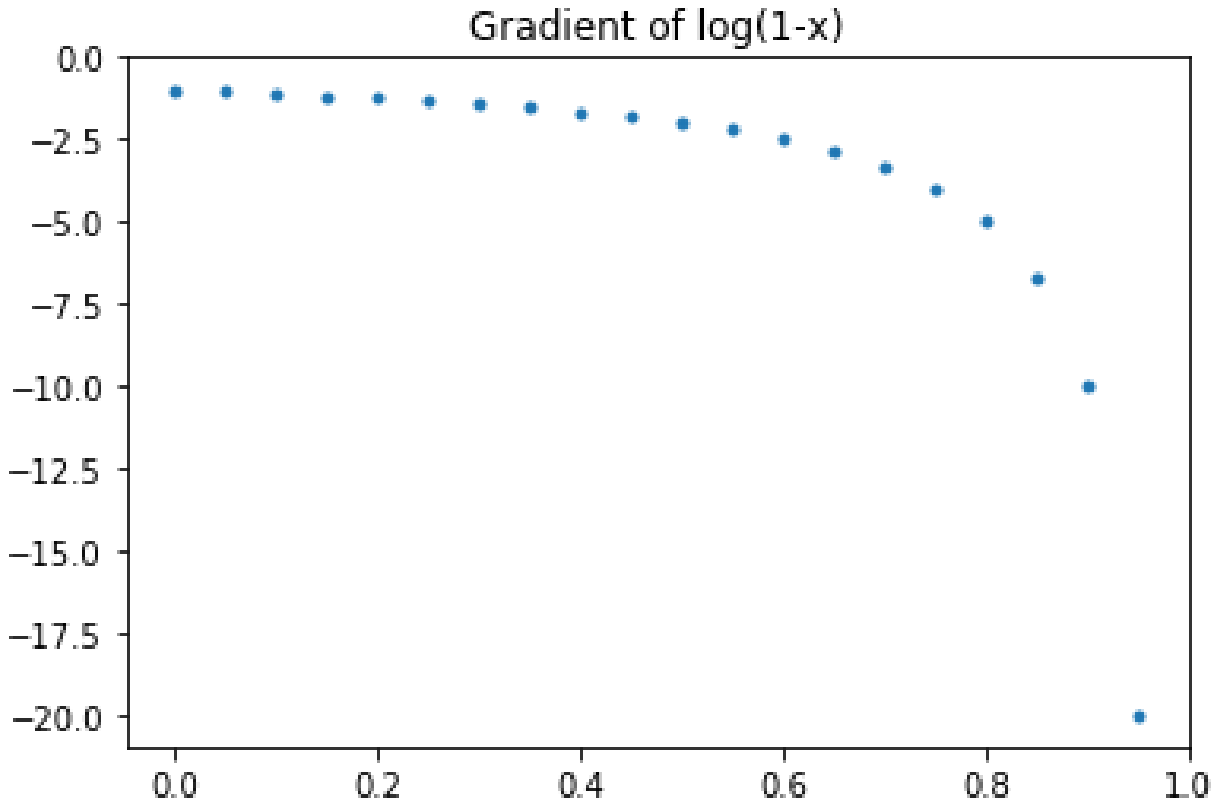

Figure 1: Gradient of log(1 - x)

Therefore, I chose equation 3 to train InfoGAN. Equation 3 converts disadvantage as advantage with gradient of $log(x)$ and zero $D(G(z)$ in begin.

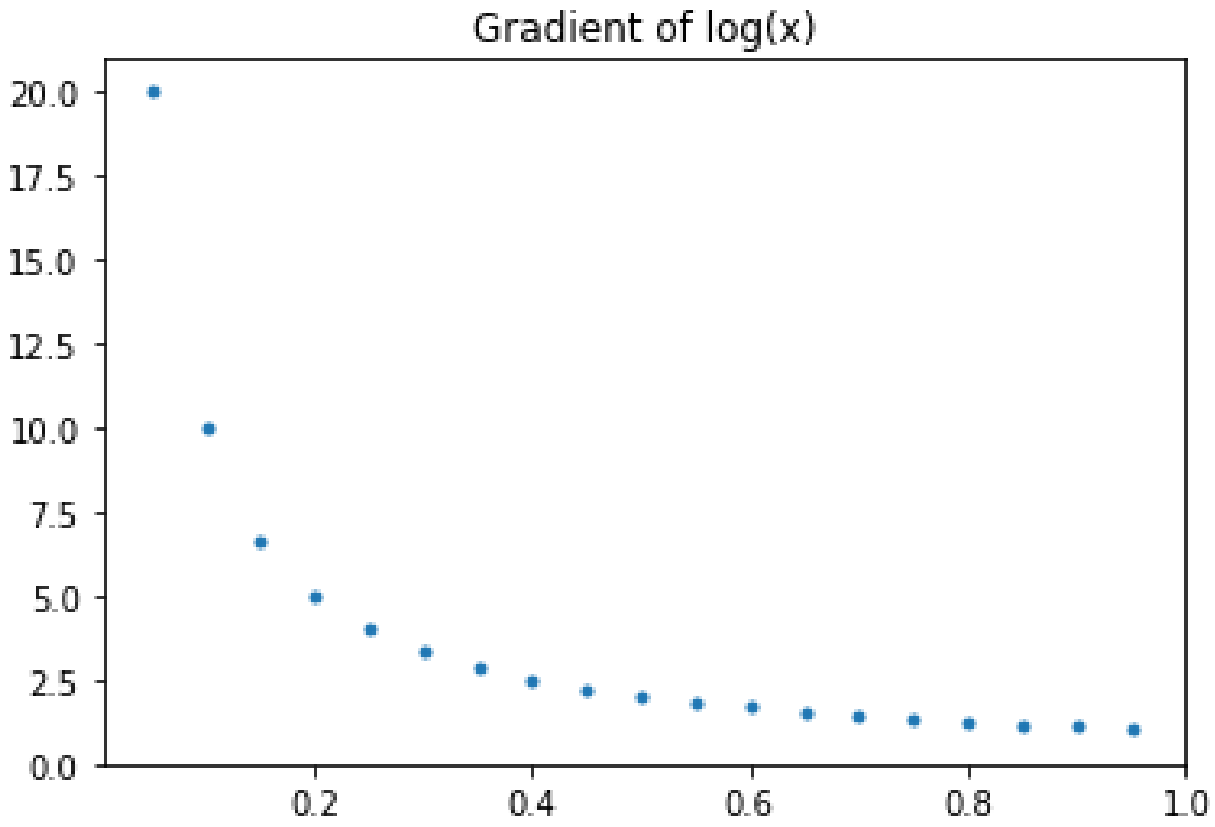

Figure 2: Gradient of log(1 - x)

Implement of loss from equation (1) as follows:

```
# Omit .......
                # Discriminator part
                optim_D.zero_grad()

                # Feed real image
                x, _ = batch_data

                bs = x.size(0)
                label.data.resize_(bs, 1)

                real_x = x.cuda()

                real_out = self.discriminator(self.front_end(real_x))

                # criterion_D = nn.BCELoss().cuda()
                label.data.fill_(1.0)
```

```
                loss_D_real = criterion_D(real_out, label)

                loss_D_real.backward()

                # Feed fake image from generator
                z, idx = self._noise_sample(bs)

                fake_x = self.generator(z)
                fake_out =
                ↪  self.discriminator(self.front_end(fake_x.detach()))

                # criterion_D = nn.BCELoss().cuda()
                label.data.fill_(0.0)
                loss_D_fake = criterion_D(fake_out, label)

                loss_D_fake.backward()

                optim_D.step()

                loss_D = loss_D_real + loss_D_fake
# Omit .......
```

Implement of loss from equation (3) as follows:

```
# Omit .......
                # Generator and Q part
                optim_G.zero_grad()

                # fake_x from previous `fake_x = self.generator(z)`
                fe_out = self.front_end(fake_x)
                ad_out = self.discriminator(fe_out)

                # criterion_D = nn.BCELoss().cuda()
                label.data.fill_(1.0)
                loss_G_reconstruct = criterion_D(ad_out, label)
# Omit .......
```

I can easily convert labels of discriminator as 1.0 then implement equation (3).

2.1.2 Maximizing mutual information

After deciding adversarial loss, InfoGAN needs another part different from discriminator and generator. It called Q. The Q part classify image into several cluster. In our MNIST dataset is ten clusters mapping to different digit numbers. And InfoGAN controls cluster of output

image by noise z embedded one-hot vector c. But Q need to use same feature extraction with discriminator. Because if they use different feature extraction each other, Q can't learn anything from real data. The generator and Q maybe use info which not digit number. In that situation, I can get lower loss from Q and D but I can't control output image with specific digit number.

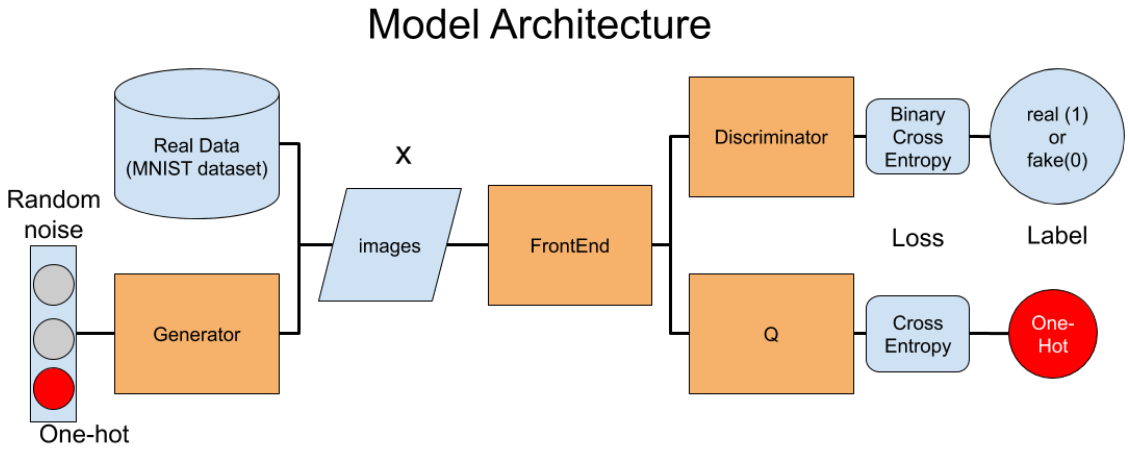

Figure 3: Architecture of my implement

Implement of model architecture as follows :

```
class Generator(nn.Module):
    def __init__(self, noise_size, feature_size):
        super(Generator, self).__init__()

        self.main = nn.Sequential(

            nn.ConvTranspose2d(noise_size, feature_size*8, 4, 1,
            ↪  bias=False),
            nn.BatchNorm2d(feature_size*8),
            nn.ReLU(True),

            nn.ConvTranspose2d(feature_size*8, feature_size*4, 4, 2, 1,
            ↪  bias=False),
            nn.BatchNorm2d(feature_size*4),
            nn.ReLU(True),

            nn.ConvTranspose2d(feature_size*4, feature_size*2, 4, 2, 1,
            ↪  bias=False),
            nn.BatchNorm2d(feature_size*2),
```

```
            nn.ReLU(True),

            nn.ConvTranspose2d(feature_size*2, feature_size*1, 2, 2, 1,
            ↪  bias=False),
            nn.BatchNorm2d(feature_size*1),
            nn.ReLU(True),

            nn.ConvTranspose2d(feature_size*1, 1, 1, 1, 1, bias=False),
            nn.Tanh()
            #nn.Sigmoid()
        )
        self.noise_size = noise_size

    def forward(self, x):
        return self.main(x)

class FrontEnd(nn.Module):
    def __init__(self, feature_size):
        super(FrontEnd, self).__init__()

        self.main = nn.Sequential(

            nn.Conv2d(1, feature_size*1, 1, 1, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(feature_size*1, feature_size*2, 2, 2, 1, bias=False),
            nn.BatchNorm2d(feature_size*2),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(feature_size*2, feature_size*4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(feature_size*4),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(feature_size*4, feature_size*8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(feature_size*8),
            nn.LeakyReLU(0.2, inplace=True)
        )

    def forward(self, x):
        return self.main(x)

class Discriminator(nn.Module):
    def __init__(self, feature_size):
        super(Discriminator, self).__init__()
```

```
        self.main = nn.Sequential(
            nn.Conv2d(feature_size*8, 1, 4, 1, bias=False),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.main(x)
        #return self.main(x).view(-1, 1)
        return x.view(-1,1)

class Q(nn.Module):
    def __init__(self, feature_size):
        super(Q, self).__init__()

        self.main = nn.Sequential(
            nn.Linear(feature_size*8*4*4, 100, bias=True),
            nn.ReLU(),
            nn.Linear(100, 10, bias=True)
        )

    def forward(self, x):
        x = x.view(x.size(0), -1)
        return self.main(x).squeeze()
```

And The loss of Q is cross entropy. Implement as follows (also part of previous section) :

```
# Omit .......
                # Generator and Q part
                optim_G.zero_grad()

                # fake_x from previous `fake_x = self.generator(z)`
                fe_out = self.front_end(fake_x)
                ad_out = self.discriminator(fe_out)

                # criterion_D = nn.BCELoss().cuda()
                label.data.fill_(1.0)
                loss_G_reconstruct = criterion_D(ad_out, label)

                c = self.Q(fe_out)

                # criterion_Q = nn.CrossEntropyLoss().cuda()
                loss_Q = criterion_Q(c, idx)

                loss_G = loss_G_reconstruct + loss_Q
```

```
                loss_G.backward()

                optim_G.step()

                fake_x = self.generator(z)
                fake_out_after =
                 ↪  self.discriminator(self.front_end(fake_x.detach()))
# Omit .......
```

2.1.3 Generate noise and image

When training, I generated different noise with random cluster index which size is batch size. When evaluation or image generated, I generated same noise with ten cluster index from 0 to 9.

```
# Omit .......
    def _noise_eval(self):
        cluster_num = 10

        # gen all cluster one hot vectors
        idx = np.arange(cluster_num)
        c = np.zeros((cluster_num, cluster_num))
        c[range(cluster_num), idx] = 1.0

        # gen torch (#cluster, #noise, 1, 1) with same noise
        z = torch.cat([
            torch.FloatTensor(1, self.noise_size -
             ↪  cluster_num).uniform_(-1.0, 1.0).expand(cluster_num,
             ↪  self.noise_size - cluster_num),
            torch.Tensor(c)
        ] , 1).view(-1, self.noise_size, 1, 1)

        return z.cuda(), torch.LongTensor(idx).cuda()

    def _noise_sample(self, batch_size=None):
        if batch_size is None:
            batch_size = self.batch_size
        # gen condition c
        cluster_num = 10
        idx = np.random.randint(cluster_num, size=batch_size)
        c = np.zeros((batch_size, cluster_num))
        c[range(batch_size), idx] = 1.0

        # gen torch (batch, #noise, 1, 1)
```

```
        z = torch.cat([
            torch.FloatTensor(batch_size, self.noise_size -
            ↪  cluster_num).uniform_(-1.0, 1.0),
            torch.Tensor(c)
        ] , 1).view(-1, self.noise_size, 1, 1)

        return z.cuda(), torch.LongTensor(idx).cuda()
# Omit .......
```

And I generate fixed result with modify random seed in pytorch.

```
# Omit .......
    def evaluation(self, noise_num=10, seed=None):

        if seed is None:
            import time
            seed = time.time()

        self.generator.eval()

        x = []
        torch.manual_seed(seed)
        for i in range(noise_num):
            z, _ = self._noise_eval()
            x.append( self.generator(z) )

        x = torch.cat(x, dim=0)

        self.generator.train()

        return x.detach().cpu()
# Omit .......
```

2.2 Loss function of generator

According previous implement, my loss of generator as follows:

$$\text{Loss}_{G,Q} = -E_{z\sim P_z(z)}(\log D(G(z;\theta_G);\theta_D)) - E_{c\sim P_c(c),z'\sim P_{z'}(z')}(\log Q(c|G(z;\theta_G);\theta_Q)) \quad (4)$$

3 Experimental results

3.1 Generated image from sample

Figure 4: Result, x-axis is different condition, y-axis is different noise

3.2 Training Loss curve

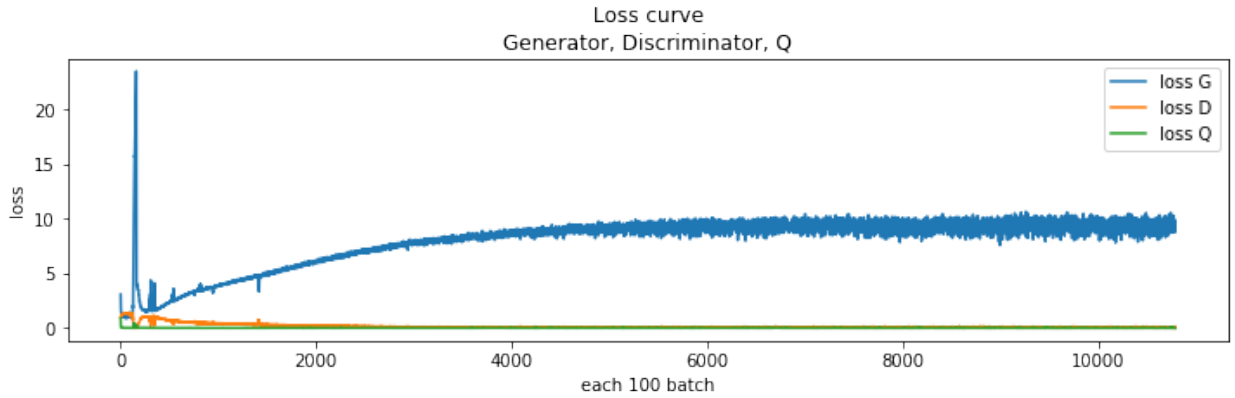

Figure 5: Loss curve of Generator, Discriminator, Q

3.3 Training Probability curve

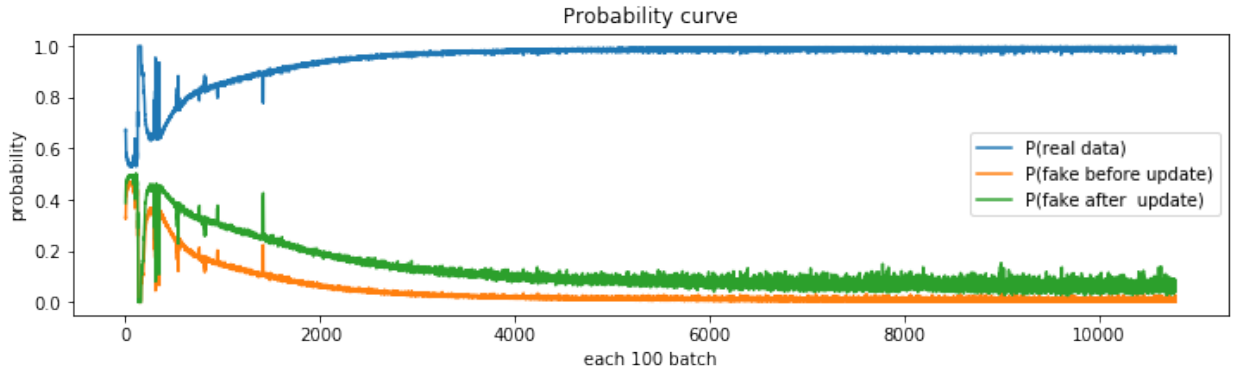

Figure 6: Probability of real data, fake data before G updating and fake data after G updating

4 Discussion

4.1 Broken InfoGAN

When I does this lab, I find easily get a broken InfoGAN in final. But they looks good in begining. After special point, model usually can't recovery.

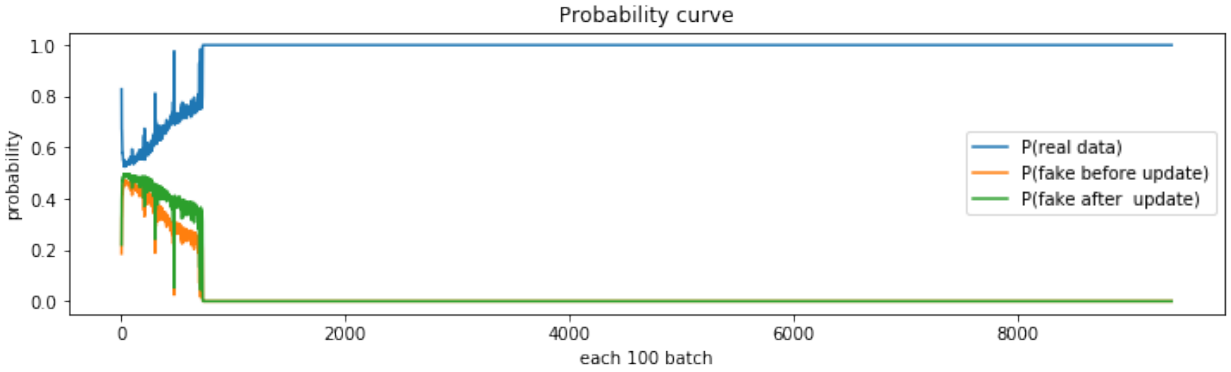

Figure 7: Probability of real data, fake data before G updating and fake data after G updating

I find discriminator became too strong too powerful lead to generator can't adversary discriminator. Because the green line can't greater than orange line after that broken point.