Lab 4: InfoGAN

Name : 鄭謝廷揚 StudentID : A073501

A. Introduction

In this lab, we will learn how to use InfoGAN to train MINIST. With InfoGAN, we can generate realistic images from our generator.

Lab requirements :

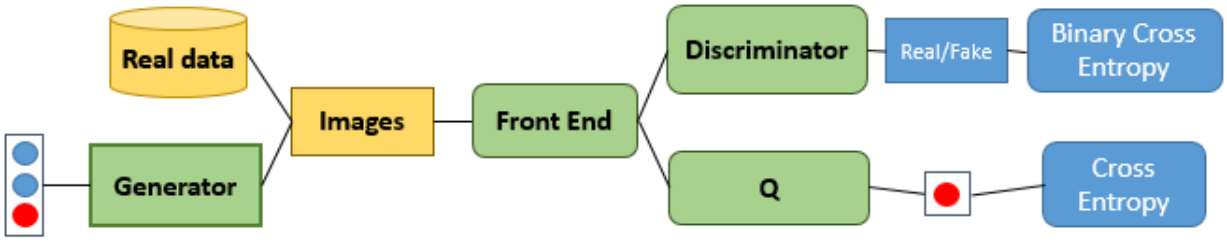
1. Modify InfoGAN
2. Maximize the mutual information between generated images and discrete one-hot vector
3. Show the generated images of the specific number
4. Plot the loss curves of the generator and the discriminator while training

B. Implementation details

1. Architectures detail

- **Overall architecture**

We separate discriminator into two part(Front End and Discriminator). With Front End, we want to see more information even can train Q better or not.



- **The setting of generator**

```
class Generator(nn.Module):
    def __init__(self, noise_size,d=128):
        super(Generator, self).__init__()

        self.main = nn.Sequential(

            nn.ConvTranspose2d(100, 1024, 4, 1, bias=False),
            nn.BatchNorm2d(1024),
            nn.ReLU(True),

            nn.ConvTranspose2d(1024, 512, 4, 2, 1, bias=False),
            nn.BatchNorm2d(512),
            nn.ReLU(True),

            nn.ConvTranspose2d(512, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.ReLU(True),

            nn.ConvTranspose2d(256, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(True),

            nn.ConvTranspose2d(128, 1, 4, 1, 1, bias=False),
            nn.Tanh()
        )
        self.noise_size = noise_size

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

- **The setting of Discriminator**

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

        self.main = nn.Sequential(
            nn.Conv2d(1024, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

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

- **The setting of Front End**

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

        self.main = nn.Sequential(

            nn.Conv2d(1, 128, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(128, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(256, 512, 4, 2, 1, bias=False),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(512, 1024, 4, 2, 1, bias=False),
            nn.BatchNorm2d(1024),
            nn.LeakyReLU(0.2, inplace=True)
        )

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

- **The setting of Q**

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

        self.main = nn.Sequential(
            nn.Linear(16384, 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()
```

2. Hyperparameters and setting

- ### Learning rate and optimizer

```python
optim_D = optim.Adam(
    [{'params':self.front_end.parameters()}, {'params':self.discriminator.parameters()}],
    lr = 0.0002, betas=(0.5, 0.99)
)
optim_G = optim.Adam(
    [{'params':self.generator.parameters()}, {'params':self.Q.parameters()}],
    lr = 0.001, betas=(0.5, 0.99)
)
```

- ### Batch size and Epoch

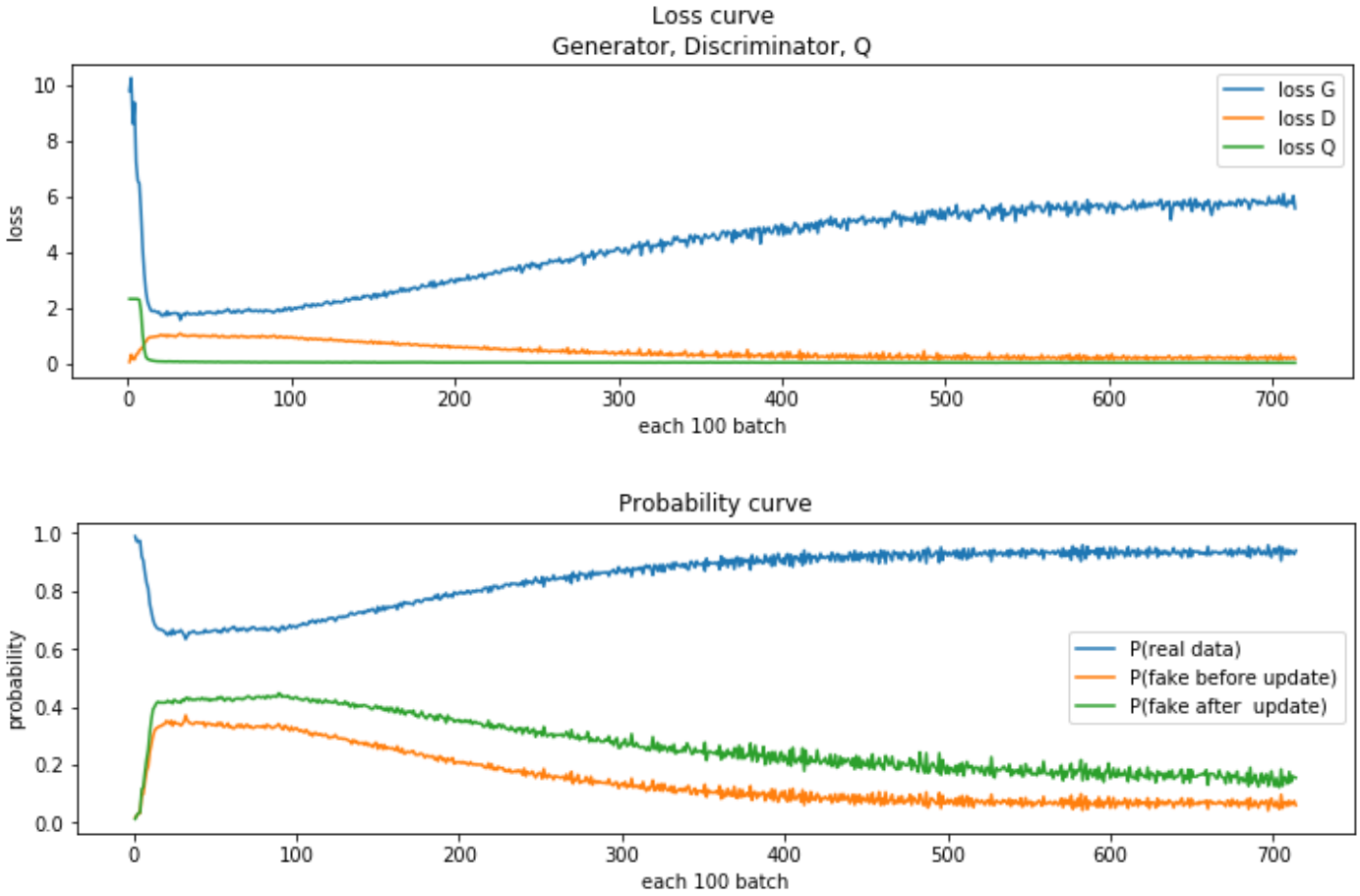
  Batch size is setting to 100, and Epoch is decided by training result.

C. Results and discussion

1. Results of all MNIST numbers

Our model can clearly generate six numbers (6, 1, 9, 7, 8, 4).

2. Discussion



At beginning of training process, loss of G, D and Q is balanced, and every part of model can be trained well. However, with the training process going generator is too much stronger than D. Therefore generator can not be trained well. In order to train D, Q and G simultaneously, maybe we can try different learning rate of D and Q.