DLP LAB6

0756138 黃伯凱

May 22, 2019

1 Introduction

這次的作業要把DCGAN改寫成InfoGAN，原本DCGAN的input只是單純的noise，但是InfoGAN的input會再加上一個class，而這個class可以幫助我們更加了解在不同domain下會讓GAN產生什麼樣不同的結果。

在paper當中有提到Fig.1，觀察第三列中有一些7有明顯的一撇，有一些沒有。所以我們沒有辦法從結果中看出不同domain下會產生出怎樣不同的結果。

Figure 1: Regular GAN result

InfoGAN會將input切成兩部分，其中一部分可以視為latent code，並且在generater後面加入了一個classifier。這一個classifier負責將generater的output分類，因此也可以分別把generater和classifier視為encoder和decoder。

有了latent code之後，我們就可以自己調整不同latent code當作input，並且觀察InfoGAN會產生怎樣不同的結果。

2 Experiment

2.1 Models

我讓Discriminater和Classifier共享Shared Layer，所以會有四個model。

```
G(
  (main): Sequential(
    (0): ConvTranspose2d(74, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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, track_running_stats=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, track_running_stats=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, track_running_stats=True)
    (11): ReLU(inplace)
    (12): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

Figure 2: Generator Model

```
Shared(
  (main): Sequential(
    (0): Conv2d(1, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=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, track_running_stats=True)
    (4): LeakyReLU(negative_slope=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, track_running_stats=True)
    (7): LeakyReLU(negative_slope=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, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace)
  )
)
```

Figure 3: Shared Layer Model

```
D(
  (main): Sequential(
    (0): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): Sigmoid()
  )
)
```

Figure 4: Discriminator Model

```
Q(
  (main): Sequential(
    (0): Linear(in_features=8192, out_features=100, bias=True)
    (1): ReLU()
    (2): Linear(in_features=100, out_features=10, bias=True)
  )
  (conv_disc): Conv2d(10, 10, kernel_size=(1, 1), stride=(1, 1))
  (conv_mu): Conv2d(10, 2, kernel_size=(1, 1), stride=(1, 1))
  (conv_var): Conv2d(10, 2, kernel_size=(1, 1), stride=(1, 1))
)
```

Figure 5: Classifier Model

Fig.5的Model即為Classifier，他可以讓我們更加了解在不同domain底下會有什麼不同的結果。

2.2 Loss

在我的InfoGAN中，我使用了62-D noise、10-D discrete latent code，2-D continuous latent code。如此一來，我的結果可以把不同特性的結果分別呈現，也可以讓特性的呈現有連續性(例如由上至下的數字線條由粗到細)。

2.2.1 Discriminator

在訓練一個InfoGAN network的時候，最優先訓練的是Discriminator，再來才是Generator和Classifier。由於Discriminator的工作是判斷Generator的output是否為real image，所以要先把real image和fake image各自餵一次給Discriminator讓他學會哪些是真哪些是假的。

```
### fake part###
z, idx = self._noise_sample(dis_c, con_c, noise, bs)
fake_x = self.G(z)
fe_out2 = self.FE(fake_x.detach())
probs_fake = self.D(fe_out2)
label.data.fill_(0)  # Set initial label all as 0
loss_fake = criterionD(probs_fake, label)
loss_fake.backward()
```

Figure 6: Fake Image Loss

```
### real part ###
optimD.zero_grad()
real_x.data.copy_(x)
fe_out1 = self.FE(real_x)
probs_real = self.D(fe_out1)
label.data.fill_(1)  # Set initial label all as 1
loss_real = criterionD(probs_real, label)
loss_real.backward()
```

Figure 7: Real Image Loss

有了fake和real的loss之後，根據Discriminator的loss function (Fig.8)，可以得到Discriminator的Loss (Fig.9)。

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

Figure 8: Discriminator Loss Function

```
D_loss = loss_real + loss_fake
```

Figure 9: Discriminator Loss

2.2.2 Classifier

接下來是訓練Classifier，這部分有兩個參數要訓練，discrete和continuous。discrete latent code可以分辨出不同的數字，continuous latent code可以讓結果有連續性的呈現。所以discrete要和label算loss，continuous要和數字分布的mean, variance算loss。

10中的fe_out是shared layer的output，idx是每一筆data的label。

```
q_logits, q_mu, q_var = self.Q(fe_out)
class_ = torch.LongTensor(idx).cuda()
target = Variable(class_)
dis_loss = criterionQ_dis(q_logits, target)
con_loss = criterionQ_con(con_c, q_mu, q_var)*0.1
```

Figure 10: Classifier Loss

2.2.3 Generator

最後才是訓練Generator，由於我們都希望Generator的output可以看起來更像real image，所以產生一堆圖片並經過Discriminator之後，我們希望output可以越接近1越好。於是Generator的loss要拿D(G(z))和全部都是1的向量一起算。這邊稱這個步驟為Reconstruction。 根據上述說明，我用

```
fe_out = self.FE(fake_x)
probs_fake = self.D(fe_out)
label.data.fill_(1.0)
reconstruct_loss = criterionD(probs_fake, label)
```

Figure 11: Reconstruction Loss

的Genrator Loss Function是Fig.12，只是因為我有加上continuous和discrete latent code，所以要在後面加上這兩個的loss(如圖Fig.13)

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

Figure 12: Generator Loss Function

```
G_loss = reconstruct_loss + dis_loss + con_loss
```

Figure 13: Generator Loss

2.3 Generate noise and images

由於前面已經把network都訓練好了，所以我只要再丟亂數進我的model就可以自然產生逼近real image的圖片。

Fig.14是我產生noise的方式，而one_hot實際上就是label。

```
fix_noise = torch.Tensor(100, 62).uniform_(-1, 1)

noise.data.copy_(fix_noise)
dis_c.data.copy_(torch.Tensor(one_hot))
```

Figure 14: Generate Noise

Fig.15和Fig.16是我分別依據latent code c1 和c2 產生的結果圖。

```
con_c.data.copy_(torch.from_numpy(c1))
z = torch.cat([noise, dis_c, con_c], 1).view(-1, 74, 1, 1)
x_save = self.G(z)
```

Figure 15: Generate image related to latent code c1

```
con_c.data.copy_(torch.from_numpy(c2))
z = torch.cat([noise, dis_c, con_c], 1).view(-1, 74, 1, 1)
x_save = self.G(z)
```

Figure 16: Generate image related to latent code c2

3 Result

3.1 Results of my Samples

我總共train了150個epoch，結果如下。

(a) Latent Code 1 Result (b) Latent Code 2 Result

Figure 17: Result

Fig.18是利用存下來的weight以及latent code c1生成單一數字的結果。

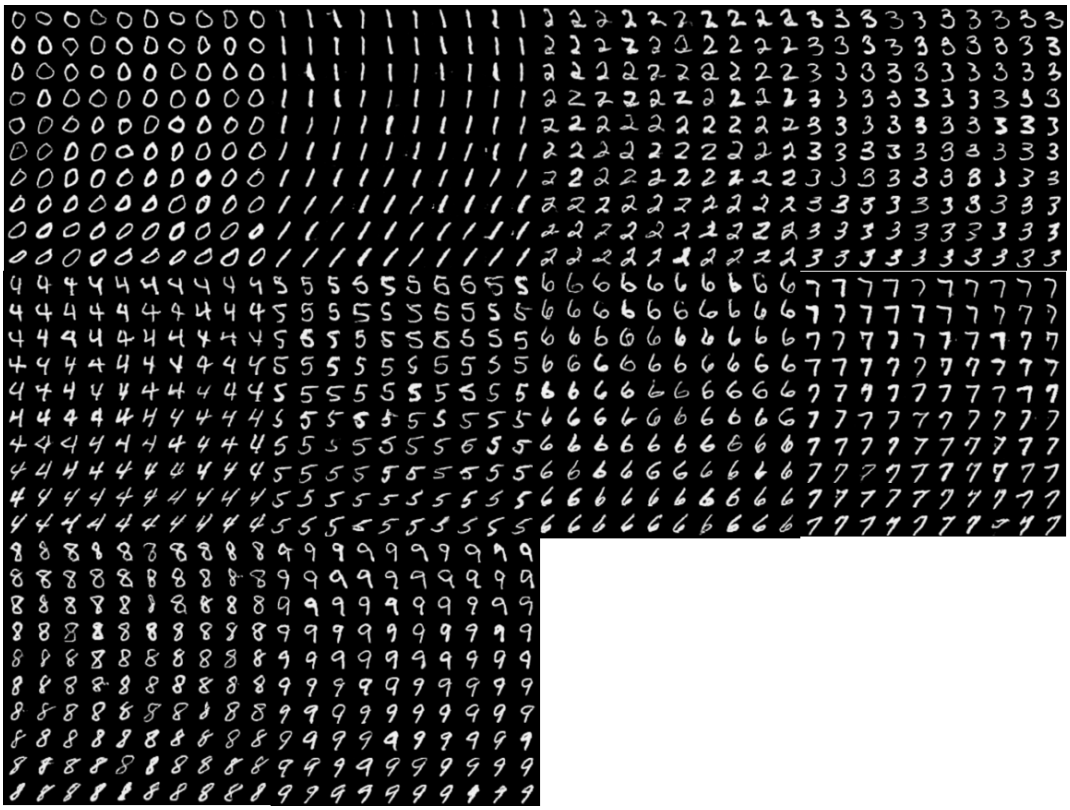

Figure 18: Latent Code 1 Result

Fig.19是利用存下來的weight以及latent code c2生成單一數字的結果。

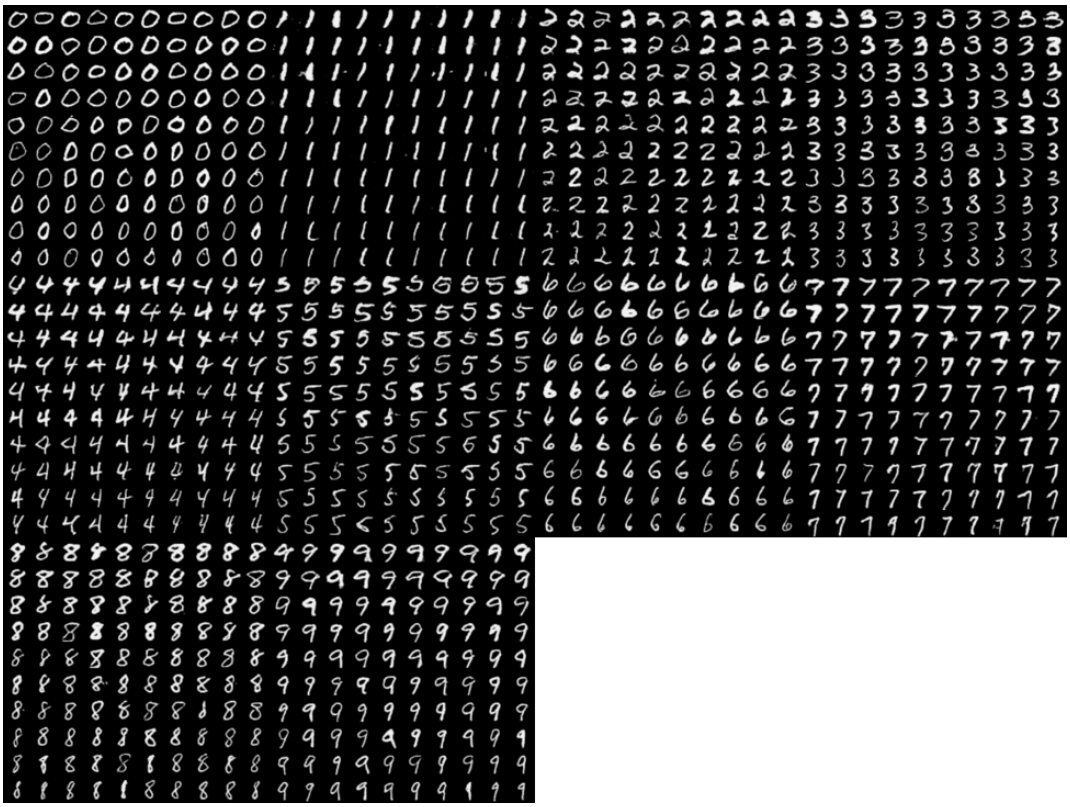

Figure 19: Latent Code 2 Result

3.2 Training Loss Curves

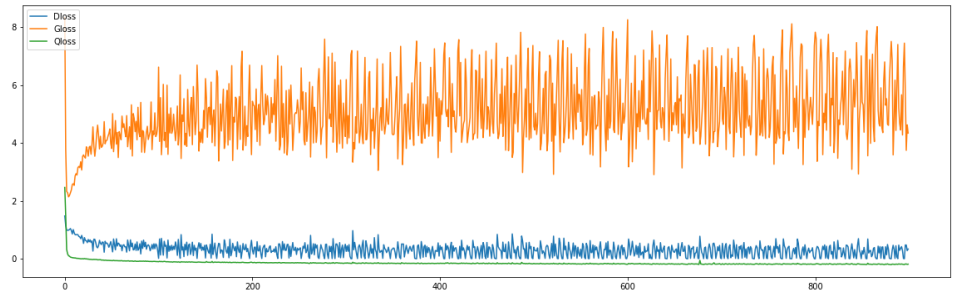

Figure 20: Loss

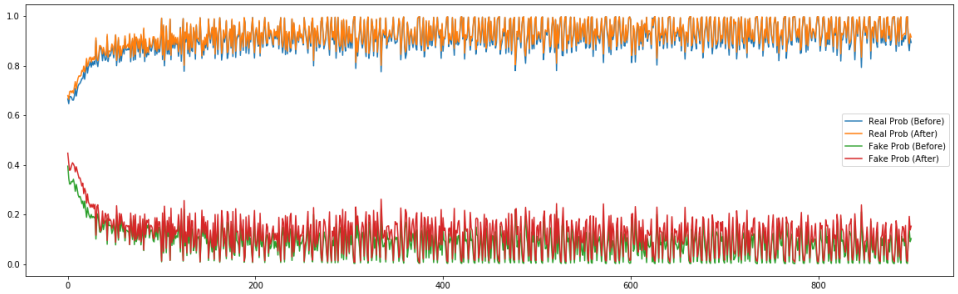

Figure 21: Probability

4 Discussion

這次作業從DCGAN改寫成InfoGAN，而且助教也有提供sample code，所以要更改的部分其實不多。主要出現問題的地方是result image的呈現上。

一開始沒注意到作業要求是要按照pdf中的結果呈現，所以我train了很多次如下圖的結果。

Figure 22: Result Image

後來發現同樣的數字要呈現一行，所以更改了Latent Code的生成方式。

```
c = np.linspace(-1, 1, 10).reshape(1, -1)
c = np.repeat(c, 10, 0).reshape(-1, 1)
zeros = np.zeros(c.shape)
c1 = np.hstack([c, zeros])
c2 = np.hstack([zeros, c])
```

Figure 23: Original Latent Code

```
c = np.linspace(-1, 1, 10).reshape(1, -1)
c = np.repeat(c, 10).reshape(-1, 1)
zeros = np.zeros(c.shape)
c1 = np.hstack([c, zeros])
c2 = np.hstack([zeros, c])
```

Figure 24: New Latent Code

也更改了One Hot Code的生成方式。

```
idx = np.arange(10).repeat(10)
one_hot = np.zeros((100, 10))
one_hot[range(100), idx] = 1
```

Figure 25: Original One Hot Code

```
one_hot = np.zeros((100, 10))
for i in range(100):
    one_hot[i, i%10] = 1
```

Figure 26: New One Hot Code