0516003 李智嘉 lab6 report

1. Introduction

這次 lab 將實做可以讓 latent code 有可以影響結果圖片資訊的 InfoGAN model，除了原本的 Generator 以及 Discriminator，還額外加了一個 Q module，希望可以最大化 latent code 與 Generator 產生圖片的 mutual information，目的是讓輸出可以按照我們希望的輸出，並且有幾個維度可以控制一些微小的變化，而這次的 dataset 則是 MNIST 來當作練習。

2. Experiment setups

A) Implementation InfoGAN

- ### Adversarial loss

```
54    D_loss = nn.BCELoss().cuda()
```

```
94      # Discriminator
95      ## For Discriminator true term
96      real_data = y[0].to(device)
97      real_prob, _ = model_D(real_data)
98      real_prob = real_prob.view(-1, 1)
99      cls = torch.ones((batch_size, 1)).to(device)
100     loss = D_loss(real_prob, cls)
101     loss.backward()
102     dis_loss += loss.data
103     ## For Discriminator fake term
104     fake = model_G(x)
105     fake_prob, temp = model_D(fake.detach())
106     cls = torch.zeros((batch_size, 1)).to(device)
107     loss = D_loss(fake_prob.view(-1, 1), cls)
108     loss.backward()
109     dis_loss += loss.data
110     opt_D.step()
111
112     # Generator
113     opt_G.zero_grad()
114     gen_loss = 0
115     ## For Generator fake term
116     cls = torch.ones((batch_size, 1)).to(device)
117     fake = model_G(x)
118     fake_prob, temp = model_D(fake)
119     gen_loss += D_loss(fake_prob.view(-1, 1), cls)
```

Adversarial loss 是 GAN 中最重要的 loss，也就是 discriminator 與 generator 在互相對抗，對 Discriminator 來說就是要分對哪些是 real data 哪些是 fake data，而 Generator 則是希望他產生的 data 能讓 Discriminator 辨識成 true，所以剛好相反。
而這邊用的是 binary cross entropy，來算 loss

- ### Maximizing mutual information

```
55  G_cross_loss = nn.CrossEntropyLoss().cuda()
56  def G_crit_loss(x, mu, var):
57      log_loss = -0.5*(var.mul(2*np.pi)+1e-6).log() - \
58              (x-mu).pow(2).div(var.mul(2.0)+1e-6)
59      return log_loss.sum(1).mean().mul(-1.0)
```

```
## For Q term
fake_label, mu, var = model_Q(temp)
gen_loss += G_cross_loss(fake_label, label.squeeze().to(device))
con_c = con_c.to(device).view(mu.shape[0], mu.shape[1])
gen_loss += G_crit_loss(con_c, mu, var)
```

而 mutual information 根據 paper 的推導，可以變成兩項，也分別列在右方了
第一個是希望用 Q module 來 predict 這個 image 的 label 是什麼，來藉此看到先前有沒有根據 condition 來產生數字
第二個則是希望 Q 的 distribution 可以越像 Gaussian 越好

- Generate fixed noise and images

  code 如下，num 是希望產生的數字 label，是一個 list，並將其轉成對應的 one_hot 形式
  fix_noise 則是對每種數字，隨機產生一個 52 維的數值，並重複 10 次堆起來
  而我的 meaningful variable 有兩維，這邊也分別對這兩個做輸出，一個在輸出區間的時候，另一個就設成 0
  最後堆起來就可以丟進去 train 好的 Generator 產生圖片了

```python
def test(num):
    num_len = len(num)
    num = torch.tensor(num*10)
    one_hot = np.zeros((10*num_len, 10))
    one_hot[range(10*num_len), num] = 1
    dis_c = torch.from_numpy(one_hot)
    fix_noise = torch.from_numpy(np.random.randn(num_len, 52).repeat(10, axis = 0))

    c = np.linspace(0, 5, 10).reshape(1, -1)
    c = np.repeat(c, num_len, 0).reshape(-1, 1)

    c1 = np.hstack([c, np.zeros_like(c)])
    c2 = np.hstack([np.zeros_like(c), c])

    con_c = torch.from_numpy(c1)
    z = torch.cat([fix_noise, dis_c, con_c], 1).view(-1, 64, 1, 1).type(torch.FloatTensor).to(device)
    fake = model(z)
    utils.save_image(fake.data, f'1.png', nrow=10)

    con_c = torch.from_numpy(c2)
    z = torch.cat([fix_noise, dis_c, con_c], 1).view(-1, 64, 1, 1).type(torch.FloatTensor).to(device)
    fake = model(z)
    utils.save_image(fake.data, f'2.png', nrow=10)
```

B) Loss function of generator

這邊我是直接算 fake data 的 loss 並加上 mutual information loss，如右圖

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

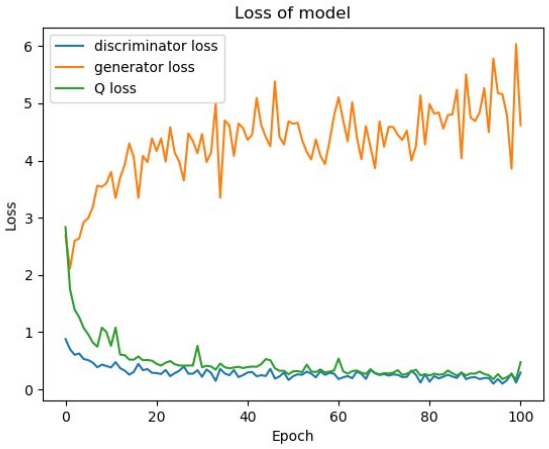
3. Results

A) Generated images

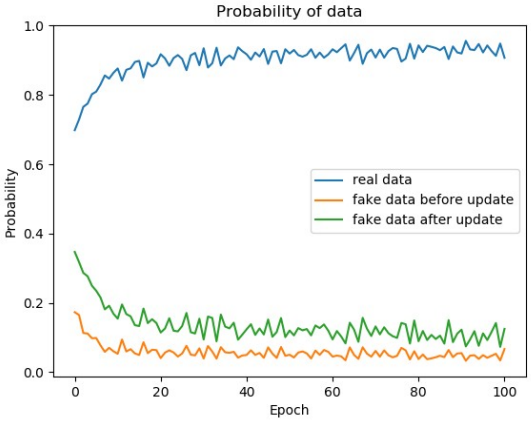
結果如右圖，從上往下是 0~9，從左往右是 meaningful variable 的變化
可以看到越左邊，筆劃越潦草且觸筆越淡，而越右邊分別慢慢有了一些變化，如 4, 7 開口漸漸合了起來，8 的下面則是越來越小

B) Training loss curves

右方上圖是 Loss 的圖
可以看到 generator 在這場比賽中是不利的，平均的 loss 越來越高，而且相當不穩定
而 Q loss 和 discriminator loss 則是越來越低，可以看作真的有學到之間的 mutual information
而這也對應了 GAN 難 train 的地方：discriminator 與 generator 的不對等問題，但這就不是 infoGAN 主要討論的重點了



再來是 Discriminator 對不同 data 的判斷機率，可以發現對於 real data 的預測越來越正確，也可以呼應到前一張圖 Discriminator loss 越來越低的結果
而 fake data 則是越來越低，比較可惜的是在最後的 Epoch，fake data 在 update 前後還是有一定的差距，也許也代表了 model 並沒有收斂，仍然在震盪中



4. Discussion

這個作業也是一番波折，其實前面寫都寫滿快的，也很快就可以開始 train，但是卻有許多問題。

第一個問題就是 input image 的大小問題，因為 Generator 出來是 64，而 MNIST 預設是 26，而當時還傻傻的直接將 true data 多經過一層 net 轉成 64，結果就是根本不能 train，後來是聽人說可以改 MNSIT 的大小才有進展的。

第二個問題就是 mode collapse，也就是 train 到一定程度 generator 就會一直輸出一樣且很糟糕的圖，原因是 discriminator 太強，對於 generator 的懲罰一不小心太強，generator 就被打飛了壞掉了，加了 weight initial 以及下面第二點改善就解決這項問題了，實在是很神奇。

第三個問題是在 train Discriminator 時，要把 fake data 在通過 Generator 完後的 backward 給砍掉，不能更新到 Generator，不然會一直 train 不起來，而我懷疑也是造成 mode collapse 的元兇之一。

第四個問題也是找最久的問題，就是一個 condition 一直輸出不了相似的圖片，最後才發現原來是因為我忘記 fix noise 要固定了，所以才會壞掉，改掉之後輸出就好很多了。

先前都聽說 GAN 很難 train，這次也真的感覺到這件事情，雖然只是在 MNIST 上而已，卻有很多問題發生，真的是獲益良多。