Lab7 : Let’s Play GANs

310605005 王映勻

1. Introduction

本次實驗是要實作一個 conditional GAN 的模型，類似之前實作過的 CVAE，能依據指定的條件生成圖片。這次使用的 dataset 為 ICLEVR 的幾何物體圖片，總共有 24 種形狀顏色不一的幾何物體，因此輸入的 condition 為一個 24-dim 的 one-hot vector。

GAN(生成性對抗網絡)是包含兩個神經網絡的深度神經網絡體系結構 : 生成模型（Generator）和判別模型（Discriminator），這兩個神經網絡相互競爭以產生最好的結果。生成模型要盡可能準確地產生真實樣子的圖片，而判別模型的工作是評估生成模型的結果的真實性，利用這種「對抗性」關係，使生成模型和判別模型分別達到最好的性能。

2. Implementation details

A. Describe how you implement your model, including your choice of cGAN, model architectures, and loss functions.

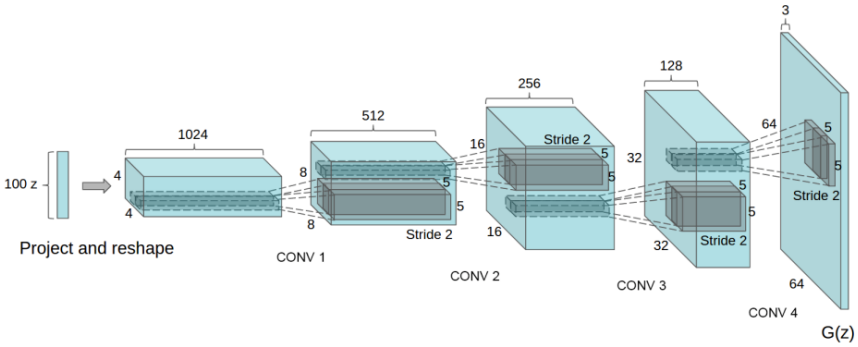
model 的架構我選擇使用 conditional GAN 加上 DCGAN，DCGAN 為 CNN 和 GAN 的結合，大幅提升了 GAN 訓練的穩定性以及生成結果質量；而加上 conditional GAN 就可以學習使用現有的類標籤生成假樣本，而不是來自未知圖像噪聲的通用樣本。

參考 pytorch 官網 DCGAN 的範例，分為 weight initialization、Generator、Discriminator、loss function 和 training 幾個部分討論：

首先 weight initialization 的部分，根據 DCGAN 的初始論文，所有模型權重應當從均值為 0，標準差為 0.02 的正態分佈中隨機初始化，而 batch normalization 的均值則要設為 1。

Generator 的目的是要將 latent vector 映射到數據空間，也就是輸入圖

像的大小(3*64*64)。首先會將 condition vector 和雜訊 z (100-dim) concatenate 起來，其中 condition vector 會先經過一個 fully connected layer 把原先 24-dim 的特徵向量轉成 200-dim 來擴充資訊，因此形成一個 300-dim 的 vector。再經過五層二維反卷積層，前四層除了二維反卷積外還包括 BatchNormalization 和 ReLU activation function。最後經過 tanh 函數處理產生 fake image，架構圖及實作流程如下：




```python
class Generator(nn.Module):
    def __init__(self, z_dim, c_dim):
        super(Generator, self).__init__()
        self.z_dim = z_dim
        self.c_dim = c_dim
        self.conditionExpand = nn.Sequential(
            nn.Linear(24, c_dim),
            # nn.ReLU(True)
        )

        ngf = 64

        self.convT1 = nn.Sequential(
            nn.ConvTranspose2d(z_dim+c_dim, ngf*8, (4, 4), stride=(1,1), padding=(0,0), bias=False),
            nn.BatchNorm2d(ngf*8),
            nn.ReLU(True)
        )
        self.convT2 = nn.Sequential(
            nn.ConvTranspose2d(ngf*8, ngf*4, (4, 4), stride=(2,2), padding=(1,1), bias=False),
            nn.BatchNorm2d(ngf*4),
            nn.ReLU(True)
        )
        self.convT3 = nn.Sequential(
            nn.ConvTranspose2d(ngf*4, ngf*2, (4, 4), stride=(2,2), padding=(1,1), bias=False),
            nn.BatchNorm2d(ngf*2),
            nn.ReLU(True)
        )
        self.convT4 = nn.Sequential(
            nn.ConvTranspose2d(ngf*2, ngf, (4, 4), stride=(2,2), padding=(1,1), bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True)
        )

        self.convT5 = nn.ConvTranspose2d(ngf, 3, (4, 4), stride=(2,2), padding=(1,1), bias=False)
        self.tanh = nn.Tanh()
```

Discriminator 則是作為一個二分類網絡，將圖像作為輸入，並輸出該圖為真的機率。基本上與 Generator 相反，首先會把 24-dim 的 condition vector 經由 fully connected layer reshape 變成一張 1*64*64 的圖，同樣也是為了

擴充資訊，之後再與 training data 或是 Generator 的圖片做 concatenate 變成(3+1)*64*64 的圖片。經過五層二維卷積層，前四層也是除了二維卷積還包括 BatchNormalization 和 LeakyReLU activation function。最終經過 sigmoid 輸出圖片真實度的機率，實作流程如下：

```
class Discriminator(nn.Module):
    def __init__(self, img_shape, c_dim):
        super(Discriminator, self).__init__()
        self.H, self.W, self.C = img_shape
        self.conditionExpand = nn.Sequential(
            nn.Linear(24, self.H*self.W*1),
            # nn.Linear(24,self.C*self.H*self.W),
            # nn.LeakyReLU()
        )

        ndf = 64

        self.conv1 = nn.Sequential(
                nn.Conv2d(4, ndf, (4, 4), stride=(2,2), padding=(1,1), bias=False),
                nn.BatchNorm2d(ndf),
                nn.LeakyReLU()
        )
        self.conv2 = nn.Sequential(
                nn.Conv2d(ndf, ndf*2, (4, 4), stride=(2,2), padding=(1,1), bias=False),
                nn.BatchNorm2d(ndf*2),
                nn.LeakyReLU()
        )
        self.conv3 = nn.Sequential(
                nn.Conv2d(ndf*2, ndf*4, (4, 4), stride=(2,2), padding=(1,1), bias=False),
                nn.BatchNorm2d(ndf*4),
                nn.LeakyReLU()
        )
        self.conv4 = nn.Sequential(
                nn.Conv2d(ndf*4, ndf*8, (4, 4), stride=(2,2), padding=(1,1), bias=False),
                nn.BatchNorm2d(ndf*8),
                nn.LeakyReLU()
        )
        self.conv5 = nn.Conv2d(ndf*8, 1, (4, 4), stride=(1,1))
        self.sigmoid = nn.Sigmoid()
```

GAN 的 objective function 定義如下圖一，因此我們可以使用 pytorch 定義的 BCELoss (Binary Cross Entropy) (下圖二) 來作為訓練時 loss 的計算方式。

$$\min_G \max_D \mathbb{E}_{x\sim q_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = -\left[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)\right]$$

訓練過程也可分為 Discriminator 和 Generator 兩個步驟。首先 Discriminator 的部份，在訓練時會先從訓練集中產生一批真實樣本，向前傳遞給 Discriminator，計算 loss $\log D(x)$。接著，用當前的 Generator 器構造一批假樣本，將該批樣本向前傳遞給 Discriminator，計算 loss $\log(1 -$

$\mathrm{D}(G(z))$)，並透過向後傳遞累加梯度。而訓練 Generator 的部分，是先透過 Discriminator 對第一部分 Generator 的輸出進行計算 loss $log\ \mathrm{D}(G(z))$，利用反向傳播累加梯度，並透過 optimizer 更新參數。

B. Specify the hyperparameters (learning rate, epochs, etc.)

經過多種不同組合的測試，最佳的參數設定為訓練 epochs=200，learning rate=0.0001，batch_size=64，雜訊 z_dim=100，條件 c_dim=200。

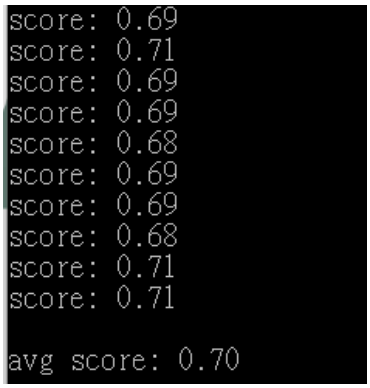
透過實驗發現，模型大多會在 100 多個 epoch 就收斂了，在後面雖然很偶爾會出現更好的分數，但應該都已經 overfitting 了。learning rate, batch_size 和 z_dim 都與官方範例的參數設置差不多，上下調整數值模型的表現都沒有變好的趨勢。而 c_dim 則是我自己設定用來擴充 condition 資訊的參數，再往上調表現也都差不多。

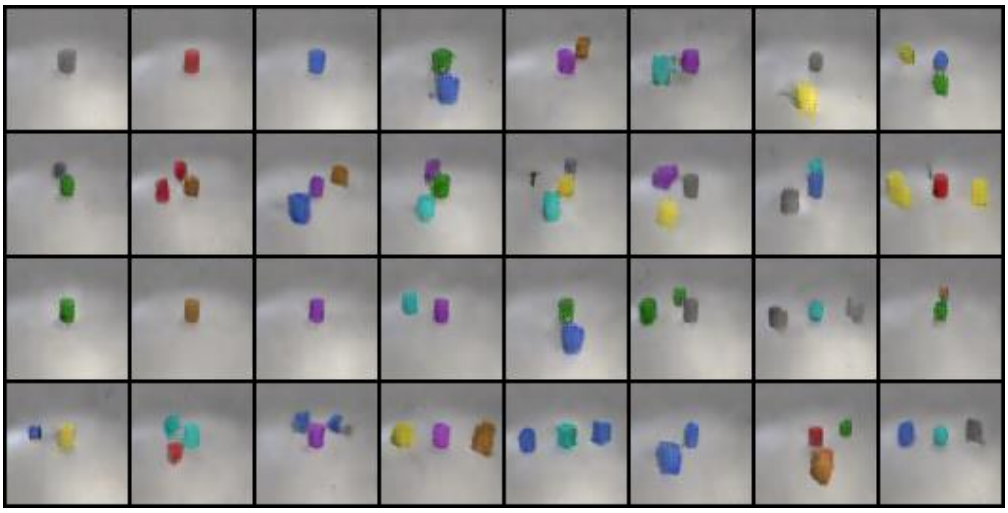
3. Results and discussion

A. Show your results based on the testing data. (including images)

(1) test.json

Score：0.70



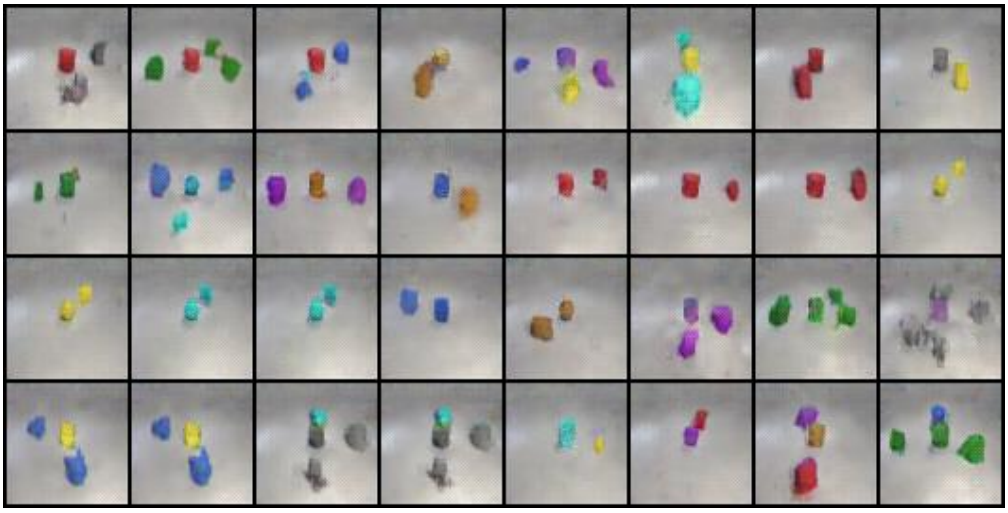


(2) new_test.json

Score：0.71

```
score: 0.70
score: 0.71
score: 0.70
score: 0.70
score: 0.71
score: 0.71
score: 0.71
score: 0.71
score: 0.70
score: 0.70

avg score: 0.71
```



B. Discuss the results of different models architectures.

模型架構的部分，除了經典的 DCGAN，我也有嘗試 ACGAN 及 WGAN。ACGAN 的特點在於 Discriminator 的輸出除了圖片的真實度還包括其 class

label，照理來說可以升成更高質量的圖片，但訓練 500 epochs 分數大概只有 0.25 左右，可能是因為類別太多難以收斂，且訓練過程兩者的 loss 震盪都很大，結果圖片如下：

WGAN 主要做了四個改變，捨棄 Discriminator 最後一層的 sigmoid 函數；loss 不取 log；將 Discriminator 更新後的參數絕對值限制在一個固定常數的範圍；不用基於動量的 optimizer (Adam)，改用 RMSProp。透過簡單的改變試圖使 GAN 訓練更穩定並解決 collapse mode 等問題，但我實作測試後效果和原先的 DCGAN 是差不多的，測試分數也都落在大概 0.65~0.7 之間，並沒有明顯的改善，因此這邊就不特別放結果圖。