Deep Learning and Practice Lab5

311605011

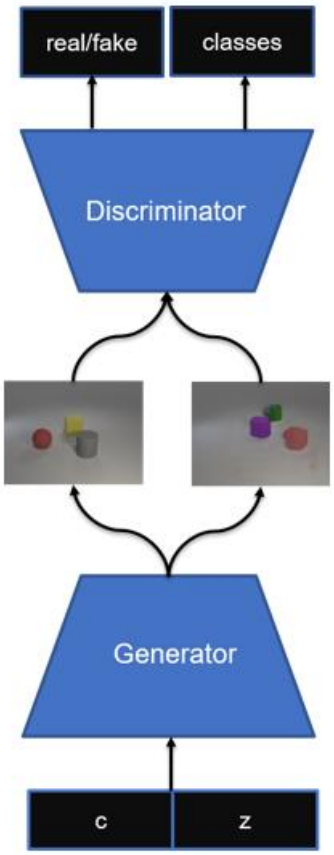
黃品振

1. Introduction:

這次的作業是要實作一個 conditional GAN，利用 Generator 加上不同的 condition 去生成我們想要的圖片。這次的訓練資料是 ICLEVR 的資料，是由 24 種不同的幾何圖形構成的圖片，而我們的目標是生成指定情況的圖片。

2. Implementation details:

A. How I implement my model :



上圖是這次 CGAN 的流程圖，一般的 GAN 是由 noise 經過 Generator 生成新的圖片，而我們這次的流程是要再將 condition

的資訊和 noise concate 在一起送入 Generator，生成圖片之後要將生成的假圖片和 condition 送到 Discriminator 判斷是真是假，也順便讓 Discriminator 擁有確認送入圖片與正確的 condition 是否有對應的能力，除此之外，再將真正的圖片與 condition 送到 Discriminator 去訓練，讓其有能力判定送入的圖片是真的是真的，或者是真的是假的。而 Generator 的部分，我們會將生成的圖片送入 Discriminator 來去預測，然後再全為真的狀況做 loss 的計算，去將 Generator 訓練到可以生成不錯的圖片。

B. Generator and Discriminator:

- What kind of GAN did I choose:
  因為 GAN 相當難訓練，所以我在這次的實驗中我使用的 GAN 是架構比較簡單的 DCGAN，如果訓練失敗也比較好找出問題的原因。
- Generator :
  我所使用的 Generator model 設為可以決定 latent vector(z)以及另外一個維度 c，這個維度 c 是由一個全連接層將原本為 24 維的 one hot vector 變成我們想要的維度 c，接著我們會將 latent vector 以及經由全連接層得到的 vector concate 在一起當作神經網路的輸入(此時輸入維度為 c+z)，得到生成的圖片。
- Discriminator:
  也是有一個全連接層，將 condition 的 24 維 mapping 到 1*64*64，再將圖片(3*64*64)與經過全連接層的 condition concate 在一起當作神經網路的輸入，最後就可以得到一個範圍在(0, 1)之間的評分。
- Training:
  訓練的時候我們分成兩個部分:Discriminator 以及 Generator，我們會先訓練 Discriminator，接著再訓練 Generator。首先是 Discriminator，Discriminator 的 Loss 分為兩項，第一項是將 condition 以及 real image 送進 Discriminator 得到的分數與真正應該得到的分數(1)做 Binary Cross Entropy 計算得到，第二項則是將假照片與 condition 送進 Discriminator 得到的分數與真正應該得到的分數(0)做做 Binary Cross Entropy 計算得到，再將兩個相加，即可得到完整的 Discriminator loss function 做後續的 backpropagation。
  再來是 Generator，Generator 的 Loss 則只有一項，是將生成的圖片與 condition 輸入進 Discriminator 裡面評分，再與希望可以得到的分數(1，希望能被判成是正確的機率越大越好)做 Binary Cross Entropy 計算得到，接著就可以做後續的 backpropagation 來訓練 Generator 來產生更好的圖片了。

```python
for epoch in range(1,1 + args.epochs):
    total_loss_g=0
    total_loss_d=0
    for i,(images,cond) in enumerate(dataloader):
        g_model.train()
        d_model.train()
        batch_size=len(images)
        images = images.to(device)
        cond = cond.to(device)

        real = torch.ones(batch_size).to(device)
        fake = torch.zeros(batch_size).to(device)
        """
        train discriminator
        """
        optimizer_D.zero_grad()

        # for fake images
        z = torch.randn(batch_size, args.z_dim).to(device)
        gen_imgs = g_model(z, cond)
        pred = d_model(gen_imgs.detach(), cond)
        loss_fake = criterion(pred, fake)


        # for real images
        pred = d_model(images, cond)
        loss_real = criterion(pred, real)

        loss_d = loss_fake + loss_real
        loss_d.backward()
        optimizer_D.step()
```

```python
        """
        train generator
        """
        for _ in range(4):
            optimizer_G.zero_grad()

            z = torch.randn(batch_size,args.z_dim).to(device)
            gen_imgs = g_model(z, cond)
            predicts = d_model(gen_imgs,cond)
            loss_g = criterion(predicts,real)

            loss_g.backward()
            optimizer_G.step()


        print(f'epoch{epoch} {i}/{len(dataloader)}  loss_g: {loss_g.item():.3f}  loss_d: {loss_d.item():.3f}')
        total_loss_g += loss_g.item()
        total_loss_d += loss_d.item()
```

正常流程的 GAN 訓練方法應該是將 Discriminator 訓練到一個程度之後再開始訓練，我這邊讓 Generator 比 Discriminator 多 train 4 倍，讓其收斂得更快一些，成效也還不錯。

- The hyperparameter:

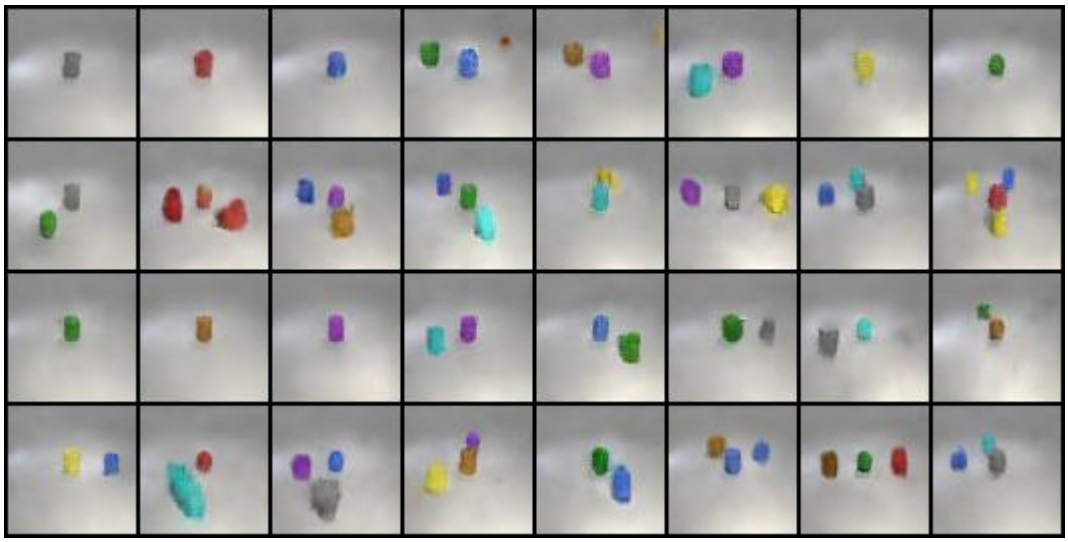
```python
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--lr', default=0.0002, type=float, help='learning rate')
    parser.add_argument('--z_dim', default=100, type=int, help='')
    parser.add_argument('--c_dim', default=100, type=int, help='')
    parser.add_argument('--epochs', default=300, type=int, help='')
    parser.add_argument('--batch_size', default=256, type=int, help='batch size')

    args = parser.parse_args()
    return args
```

3. Results and discussion:

A. The results based on testing data:

- Test:

testing score: 0.71



- New test:

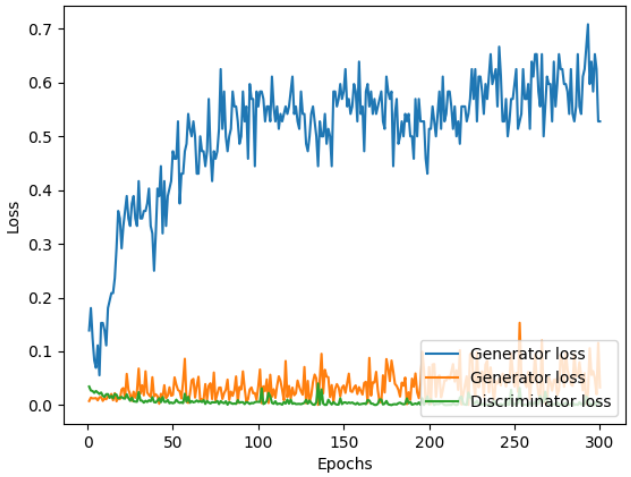
New testing score: 0.65

B. Discuss:

我後來有調整 condition 的維度以及 batch size，得到的 test 結果分別如下:

```python
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--lr', default=0.0002, type=float, help='learning rate')
    parser.add_argument('--z_dim', default=100, type=int, help='')
    parser.add_argument('--c_dim', default=100, type=int, help='')
    parser.add_argument('--epochs', default=300, type=int, help='')
    parser.add_argument('--batch_size', default=256, type=int, help='batch size')

    args = parser.parse_args()
    return args
```

Loss、accuracy 圖:

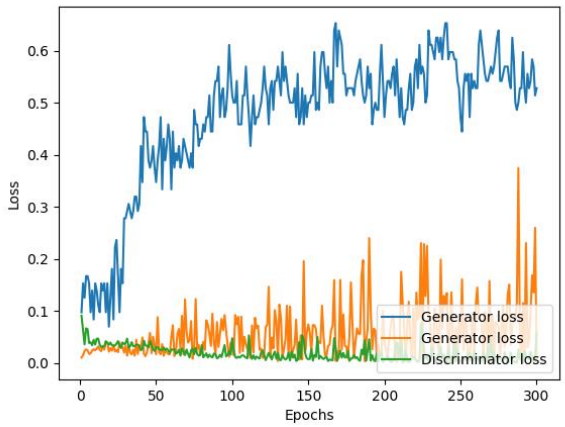


(藍色線是 accuracy，圖例的字為誤植)

Highest testing accuracy: 0.71

```python
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--lr', default=0.0002, type=float, help='learning rate')
    parser.add_argument('--z_dim', default=100, type=int, help='')
    parser.add_argument('--c_dim', default=300, type=int, help='')
    parser.add_argument('--epochs', default=300, type=int, help='')
    parser.add_argument('--batch_size', default=512, type=int, help='batch size')

    args = parser.parse_args()
    return args
```

Loss、accuracy 圖:



(藍色線是 accuracy，圖例的字為誤植)

Highest testing accuracy: 0.68

從結果發現，在 batch size 為 256 的時候效果是比 512 好，且在 Condition 的維度似乎也會影響實驗結果。在第二張圖中的 Generator Loss 震盪也比較大。