Lab5 : Let's Play GANs

311511043 李承翰

1. Introduction

   在這一次的 lab 當中我們要實作出 conditional GAN, 這部分跟上一次的 lab 有相似的地方，也就是要藉由著將標籤輸入到網路當中，讓我們可以決定出要生成怎麼樣的資料，而我們使用 ICLEVR 的圖片當作 dataset，在這一個 dataset 當中總共有 24 種幾何物體，也因此這次我們的 conditional dim 這一個變數將會是 24

   而 GAN 的全名是 Generative Adversarial Network，也就是生成對抗網路，在這一種網路模型當中有兩個子模型，分別是 generator(生成網路)、以及 discriminator(判別網路)，這兩個模型互相的訓練，當 generator 生成了新的資料後，交由 discriminator 判別這個被生成的資料像不像真實的資料，並將這個結果回饋給 generator，使其改善。藉由這樣一來一往的訓練，generator 的目標就是想盡辦法使 discriminator 無法辨別是否是生成的資料，而 discriminator 就是要盡可能地將生成的資料與真實資料區分開來。

2. Implementation details

   A. Describe how you implement your model(choice of cGAN, model architecture, loss functions)

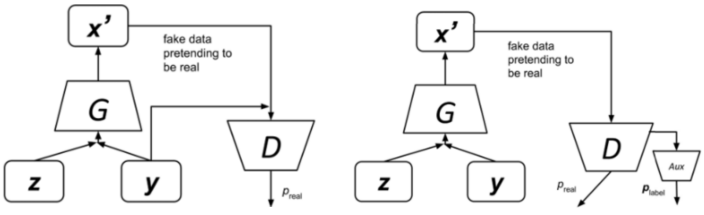
      a. Choice of cGAN

         在這次的 lab 我嘗試了兩種 cGAN, 分別是 DCGAN 以及 ACGAN

      b. Model architecture

         1. ACGAN

            ACGAN 的全稱是 Auxiliary Classifier GAN，ACGAN 的主要架構與普通的 cGAN 類似，只是她針對 discriminator 做了一些改良，除了產生一個針對生成圖片的相似度評分以外，discriminator 還多了一個 classifier 用以計算 classifier loss



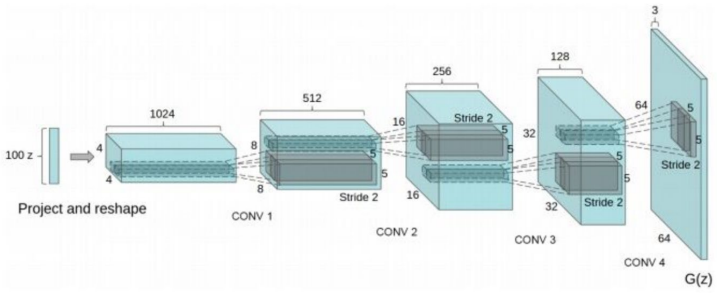
左圖是普通的 cGAN 架構，右圖是 ACGAN 的架構

在 ACGAN 的理論當中，作者認為如果我們讓模型去執行本職以外的任務，那麼應該可以提高原始的任務的性能，因此在 discriminator 旁邊多了一個 Aux 的方塊，其任務就是 classifier，作為本職以外的任務，她需要做的事情是針對生成的圖像進行分類，而本職任務就是生成圖片。

2. DCGAN

DCGAN 是一種改良版本的 GAN，不同於普通 GAN 使用全連接層，DCGAN 使用了 CNN 來取代之。

Discriminator 使用了正常的 CNN，只是將池化層都捨棄，而 Generator 使用的是反捲積層並且，在不管是 discriminator 或者 Generator 都使用了 Batch Normalization，這麼一來就可以解決在初期的時候初始化隨機性帶來的影響，提升整體訓練的穩定性。



DCGAN 的 generator 架構

c. Loss functions

1. ACGAN

在 generator 的部分，loss function 應為

$$\mathcal{L}^{(G)} = -\mathbb{E}_z \log D(g(z|y)) - \mathbb{E}_z \log p(c|g(z|y))$$

在實作當中，我們使用 pytorch 的 BCELoss 來執行

在 discriminator 當中，loss function 應為

$$\mathcal{L}^{(D)} = -\mathbb{E}_{x\sim p_{data}} \log D(x) - \mathbb{E}_z \log[1-D(G(z|y))] - \mathbb{E}_{x\sim p_{data}} p(c|x) - \mathbb{E}_z \log p(c|g(z|y))$$

針對圖片真偽度的評分，我們使用 BCELoss

classifier 的部分則是使用 CrossEntropyLoss

但是在實驗中發現，因為 Pytorch 的 CrossEntropyLoss 當中含有 softmax，對模型表現造成了不太好的影響，因此後來我也改用 BCELoss

2. DCGAN

在 generator 的部分是

$$\mathcal{L}^{(G)} = -\mathbb{E}_z \log D(g(z|y)) - \mathbb{E}_z \log p(c|g(z|y))$$

也因此與 ACGAN 相同，我們使用 BCELoss

而在 discriminator 端，我們沒有 Aux 的輔助任務，因此是:

$$\mathcal{L}^{(D)} = -\mathbb{E}_{x\sim p_{data}}\log D(x) - \mathbb{E}_z \log[1-D(G(z|y))]$$

而實作中就使用 BCELoss

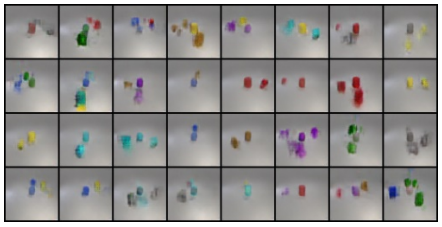
B. Specify the hyperparameters
   1. Learning rate for generator: 0.0001
   2. Learning rate for discriminator:0.0002
   3. Optimizer for generator: adam, betas = [0.5,0.999]
   4. Optimizer for discriminator: sgd, momentum = 0.9
   5. Batch size = 32
   6. Epoch = 300
   7. Dropout rate 0.5

3. Result and Discussion

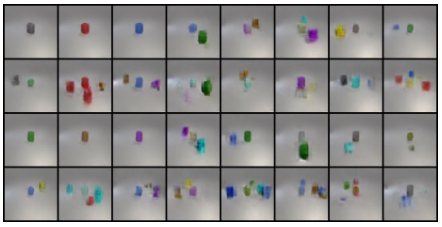
A. Show your results based on testing data(including images predicted)

```
get a better accuracy: 75.0
get a better new accuracy: 83.33333333333334
get a better accuracy: 76.38888888888889
get a better accuracy: 77.77777777777779
get a better accuracy: 79.16666666666666
get a better accuracy: 81.94444444444444
```

	ACGAN	DCGAN
Best acc(test.json)	81.944	73.12
Best acc(new_test.json)	83.33	75.66



ACGAN new_test.json



ACGAN test.json

B. Discuss the results of different models architectures

在我嘗試的兩種模型當中有 DCGAN 以及 ACGAN，而在討論 A 的圖表當中我們可以看到，ACGAN 跟 DCGAN 相比，是 ACGAN 的效果表現較好，這也符合一開始我的預期，畢竟 ACGAN 除了生成圖片的任務以外，還有一個輔助性的任務，以期可以提高他生成圖片這一個任務的正確率。

而在 loss fcn 的差異當中，本來在原論文中，ACGAN 是使用 CrossEntropy 來當中他輔助任務的損失函數，只是因為在 pytorch 當中將 argmax 也包含在了 CrossEntropy 當中，這使得本來有明確分類的資料標籤反而失去了其意義，因此造成了訓練結果不佳，所以我將 CrossEntropy 換成了

BinaryCrossEntropy 來解決這一個問題。而 DCGAN 的部分，我有觀察到在這次模型當中如果不是使用 adam 這一個 optimizer，而是使用 RMSProp 那可以有效的改善模型不穩定的問題。而在助教提供的 github 當中我也使用了其中的幾個 tips 來試圖讓模型的表現更好，例如:

1. 分別對 generator 及 discriminator 使用不同的 learning rate
2. 針對 discriminator 使用 sgd 的 optimizer
3. 以一定的機率去交換 fake label 還有 true label