1 條件式 GAN - CGAN

1.1 介紹

在上一節的 GAN 介紹裡面，我們只檢測了圖畫的真假，不管畫 0 還是畫 1，只要畫得像就算是真，但我們總是有需要可以產生特定數字的時候，那該怎麼辦呢？很簡單，就是在讓創作者畫畫的時候，順便把『條件』輸入進去即可，而我們的鑑賞者在鑑賞的時候，也需要依照條件來鑑賞『真假』，整個模型會變成下面這樣

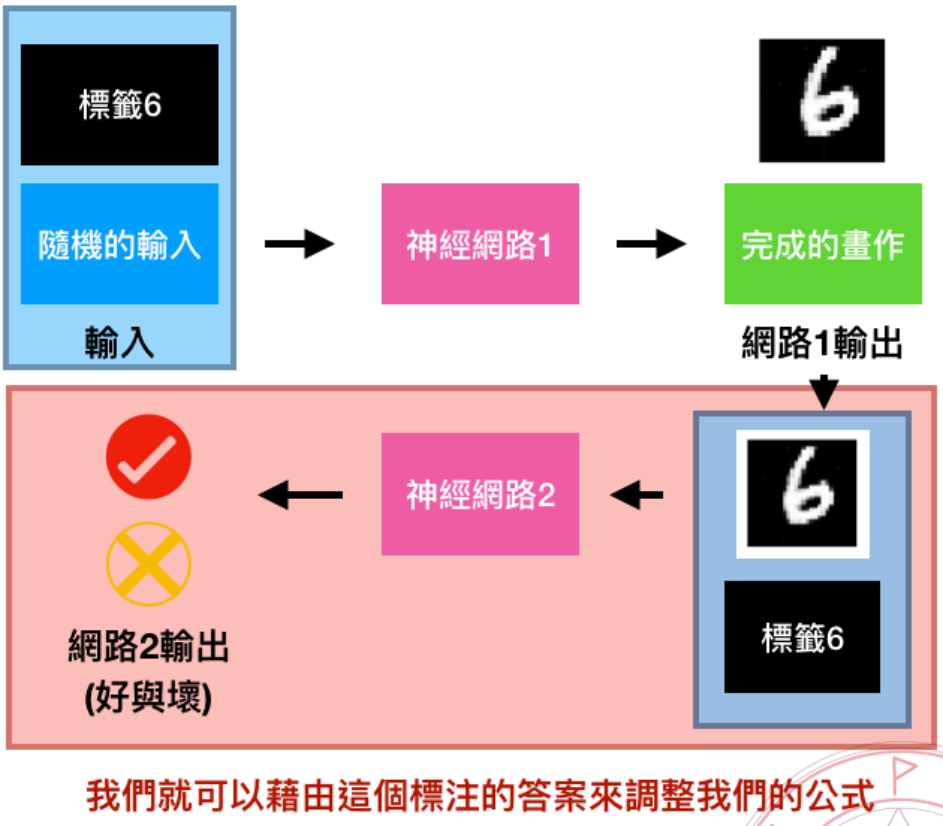


圖: CGAN 架構

1.2 ✓ Step1. 準備資料集

和之前一樣我們先準備 MNIST 資料

```
[程式]: from keras.datasets import mnist
        import numpy as np
        %matplotlib inline
        # 我們會使用到一些內建的資料庫, MAC 需要加入以下兩行, 才不會把對方的 ssl 憑證視為無效
        import ssl
        ssl._create_default_https_context = ssl._create_unverified_context
```

```
Using TensorFlow backend.
```

```
[程式]: # 回傳值: ((訓練特徵, 訓練目標), (測試特徵, 測試目標))
        (x_train, y_train),(x_test, y_test) = mnist.load_data()
```

跟之前一樣是 60000 筆 28 × 28 的黑白圖片

```
[程式]: x_train.shape
```

```
[輸出]: (60000, 28, 28)
```

一樣把資料處理，對 x 處理到-1-1 的區間

```
[程式]: from keras.utils import np_utils
        # reshape 讓他從 32 * 32 變成 784 * 1 的一維陣列
        # 讓我們標準化到-1~1 區間
        x_train_shaped = (x_train - 127.5)/127.5
        x_test_shaped = (x_test - 127.5)/127.5
```

這裡比較特別的是我們等等必須把標籤一起傳進去，讓『標籤』和『隨機的一堆亂數 (我們之前提到的靈感)』傳進去一起判斷真假，於是我們必須對於 y_train 也處理過，如何處理呢?

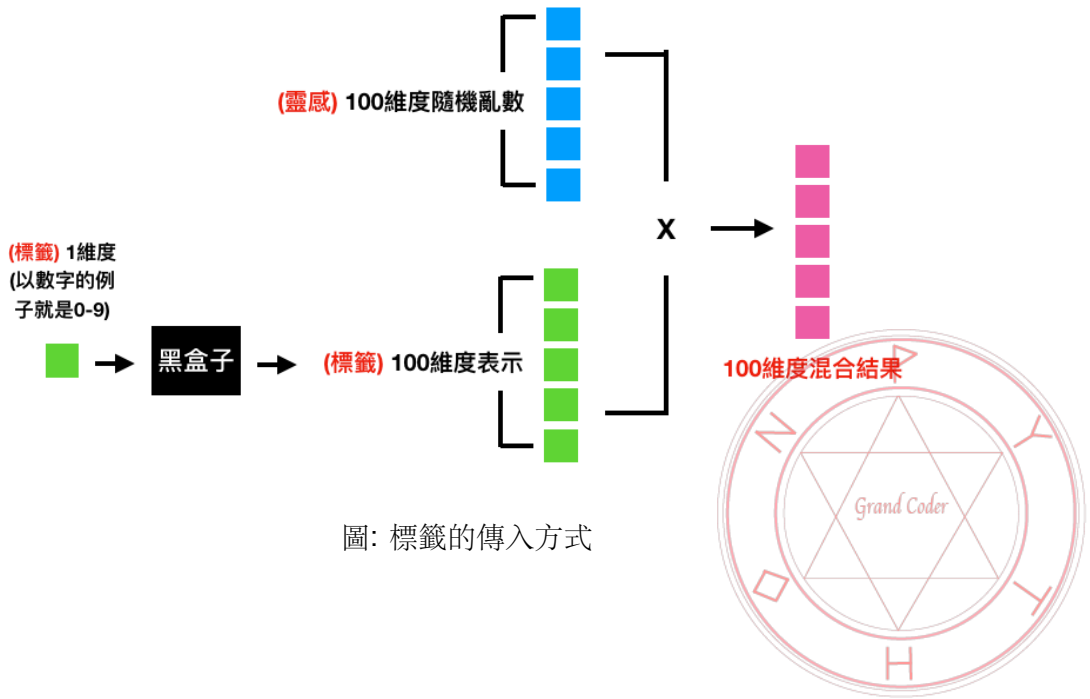


圖: 標籤的傳入方式

問題這個黑盒子是什麼呢? 熟悉深度學習基礎的同學應該也猜到了，就是我們的 Embedding 層，而 Embedding 層厲害的地方就是，雖然我們不知道每個標籤的對應表示是什麼，但是我們可以用深度學習來找出這個公式，也就是『標籤』- 『公式 (利用學習得出)』- 『Embedding』

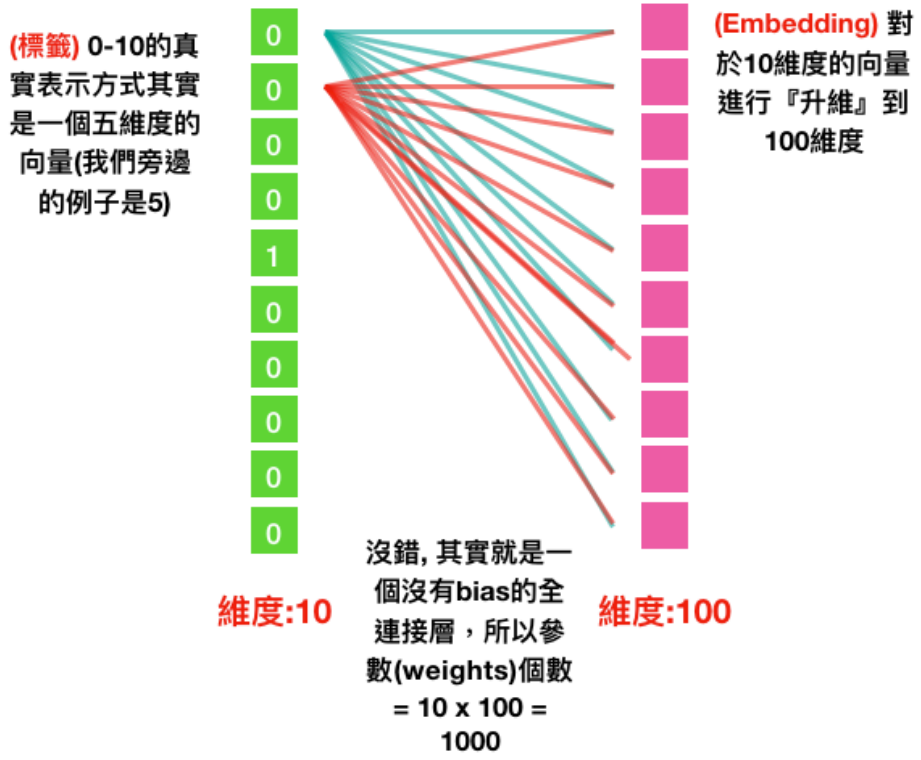


圖: Embedding 層

而我們 keras 的 Embedding 層要求的輸入長這樣子

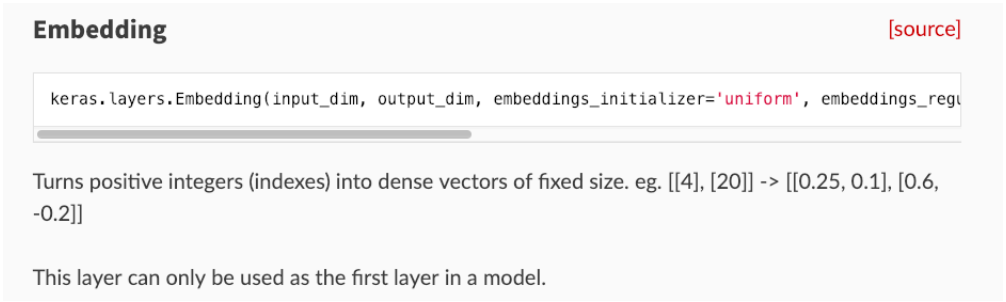
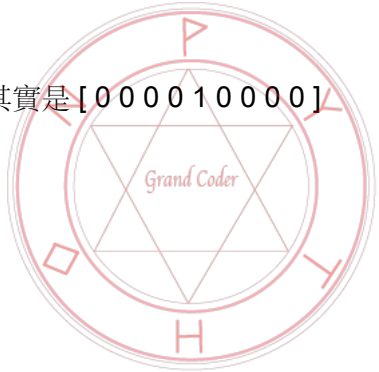


圖: Keras Embedding

所以你必須把輸入從 5 轉化成為 [5], 實際上 Keras 會知道表示其實是 [0 0 0 0 1 0 0 0 0]

```
[程式]: print('原本的 y_train shape:', y_train.shape)
        y_train
```



```
原本的 y_train shape: (60000,)
```

```
[輸出]: array([5, 0, 4, …, 5, 6, 8], dtype=uint8)
```

完成轉化！你可以觀察一下上面的表示形式和下面的表示形式

```
[程式]: # 這裡 reshape 比較特別的參數是-1
        # nparray 的 reshape 容許你填入一個-1
        # -1 是指會自動幫你算出-1 該等於多少
        # ex (2, 3) reshape (-1, 1) 相當於 reshape 成 (6, 1)
        y_train = y_train.reshape(-1, 1)
        print('後來的 y_train shape:', y_train.shape)
        y_train
```

```
後來的 y_train shape: (60000, 1)
```

```
[輸出]: array([[5],
               [0],
               [4],
               …,
               [5],
               [6],
               [8]], dtype=uint8)
```

1.3 ✓ Step2. 建立創作家

我們做一個跟我們以前反向的深度網路，神經元隨著層數越來越大，最後的神經元數目要等於你要創作的圖片的維度 (28 x 28)，我們希望最後的輸出是在 -1 和 1 區間，所以我們使用 tanh 函數當成我們最後的激活函數

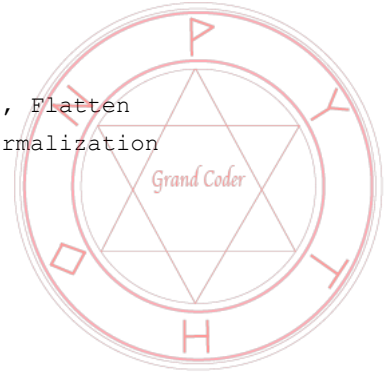
基本上就是跟之前 GAN 做一樣的事，不過你發現我們最後把 GAN 的 generator 合併上我們的標籤 Embedding，完成我們最後想要的創作家

```
[程式]: img_shape = (28, 28)
        random_dim = 100
```

我們先做一個和之前一樣的創作家，不過這不是我們最後使用的創作家

```
[程式]: from keras.layers import Input
        from keras.models import Model, Sequential
        from keras.layers.core import Reshape, Dense, Dropout, Flatten
        from keras.layers import Embedding, multiply, BatchNormalization

        # 這裡跟我們 GAN 的創作家一模一樣
```

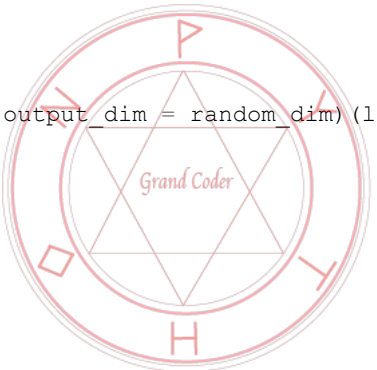


```
generator = Sequential()
generator.add(Dense(256, input_dim=random_dim, activation='relu'))
generator.add(BatchNormalization())
generator.add(Dense(512, activation='relu'))
generator.add(BatchNormalization())
generator.add(Dense(784, activation='tanh'))
generator.add(Reshape(img_shape))
generator.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 256)               25856
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dense_5 (Dense)              (None, 512)               131584
_________________________________________________________________
batch_normalization_4 (Batch (None, 512)               2048
_________________________________________________________________
dense_6 (Dense)              (None, 784)               402192
_________________________________________________________________
reshape_2 (Reshape)          (None, 28, 28)            0
=================================================================
Total params: 562,704
Trainable params: 561,168
Non-trainable params: 1,536
_________________________________________________________________
```

把剛剛的創作家的輸入 [noise] 成兩個輸入 [noise, label]。這就是我們等等要使用的創作家了！注意我們一樣沒 compile，因為不需要直接訓練他

[程式]:
```
# 這裡必需使用 Model 來做比較複雜的模型
noise = Input(shape=(random_dim,))
# 我們的標籤輸入 輸入只有一個維度 (1, ) 的, 是為了讓小括號被當成 tuple
# 否則會被當成普通的 ()
label = Input(shape=(1,), dtype='int32')
# 使用 Embedding 得到 100 向量
# input_dim = 10 -> 0~9
# output_dim = 100 -> 100 維度的向量
# 接著使用 Flatten 把 (1, 100) 轉化成為 (100)
label_embedding = Flatten()(Embedding(input_dim = 10, output_dim = random_dim)(label))
# 把靈感和標籤乘起來
model_input = multiply([noise, label_embedding])
# 經過我們上面的 Generator 做出圖片
```



```
        img = generator(model_input)
        # 完整的 generator
        # inputs = [noise, label] -> [靈感, 標籤]
        # outputs = img -> 創作圖片
        cgenerator = Model([noise, label], img)
        cgenerator.summary()
```

```
_________________________________________________________________________________________
_____________
Layer (type)                 Output Shape              Param #     Connected to
=========================================================================================
============
input_4 (InputLayer)         (None, 1)                 0
_________________________________________________________________________________________
_____________
embedding_2 (Embedding)      (None, 1, 100)            1000        input_4[0][0]
_________________________________________________________________________________________
_____________
input_3 (InputLayer)         (None, 100)               0
_________________________________________________________________________________________
_____________
flatten_2 (Flatten)          (None, 100)               0           embedding_2[0][0]
_________________________________________________________________________________________
_____________
multiply_2 (Multiply)        (None, 100)               0           input_3[0][0]
                                                                   flatten_2[0][0]
_________________________________________________________________________________________
_____________
sequential_3 (Sequential)    (None, 28, 28)            562704      multiply_2[0][0]
=========================================================================================
============
Total params: 563,704
Trainable params: 562,168
Non-trainable params: 1,536
_________________________________________________________________________________________
_____________
```

1.4 ✓ Step3. 建立鑑賞家

一樣先建立一個跟我們之前 GAN 一樣的鑑賞家

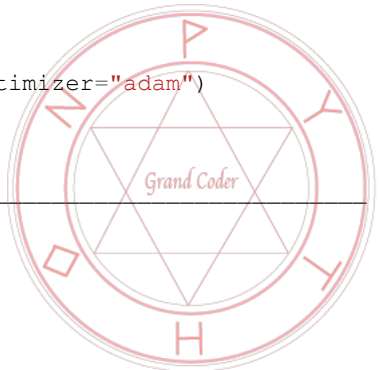
```
[程式]: discriminator = Sequential()
        discriminator.add(Dense(512, input_dim=784,
                                activation='relu'))
        discriminator.add(Dropout(0.25))
```

```
        discriminator.add(Dense(256, activation='relu'))
        discriminator.add(Dropout(0.25))
        discriminator.add(Dense(128, activation='relu'))
        discriminator.add(Dropout(0.25))
        discriminator.add(Dense(1, activation='sigmoid'))
        discriminator.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_8 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 129
=================================================================
Total params: 566,273
Trainable params: 566,273
Non-trainable params: 0
_________________________________________________________________
```

跟上面完全一樣，除了原本的要判斷的圖片，我們順便把標籤傳進去，但由於這次進去的是 28 × 28 的圖片，所以我們的標籤也應該 Embedding 成相對的維度，也就是 784 維度

[程式]:
```
img = Input(shape=img_shape)
# 得到標籤的 784 維度表示
label = Input(shape=(1,), dtype='int32')
label_embedding = Flatten()(Embedding(input_dim=10, output_dim=784)(label))
flat_img = Flatten()(img)
model_input = multiply([flat_img, label_embedding])
validity = discriminator(model_input)
cdiscriminator = Model([img, label], validity)
cdiscriminator.compile(loss='binary_crossentropy', optimizer="adam")
cdiscriminator.summary()
```




```
_________________________________________________________________
_____________
```

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_12 (InputLayer)        (None, 1)                 0
__________________________________________________________________________________________________
input_11 (InputLayer)        (None, 28, 28)            0
__________________________________________________________________________________________________
embedding_6 (Embedding)      (None, 1, 784)            7840        input_12[0][0]
__________________________________________________________________________________________________
flatten_10 (Flatten)         (None, 784)               0           input_11[0][0]
__________________________________________________________________________________________________
flatten_9 (Flatten)          (None, 784)               0           embedding_6[0][0]
__________________________________________________________________________________________________
multiply_6 (Multiply)        (None, 784)               0           flatten_10[0][0]
                                                                   flatten_9[0][0]
__________________________________________________________________________________________________
sequential_4 (Sequential)    (None, 1)                 566273      multiply_6[0][0]
==================================================================================================
Total params: 574,113
Trainable params: 574,113
Non-trainable params: 0
__________________________________________________________________________________________________
```

1.5 ✓ Step4. 組合網路

別忘了我們訓練 Generator 的時候需要一個打分數對象，於是我們必須把 Discriminator 接在後面打分數，這裡要注意一下，打分數的時候要順便把 label 也帶上去打分數，我們把上面的圖再貼一次！這個整個組合起來就是我們訓練創作家的方式

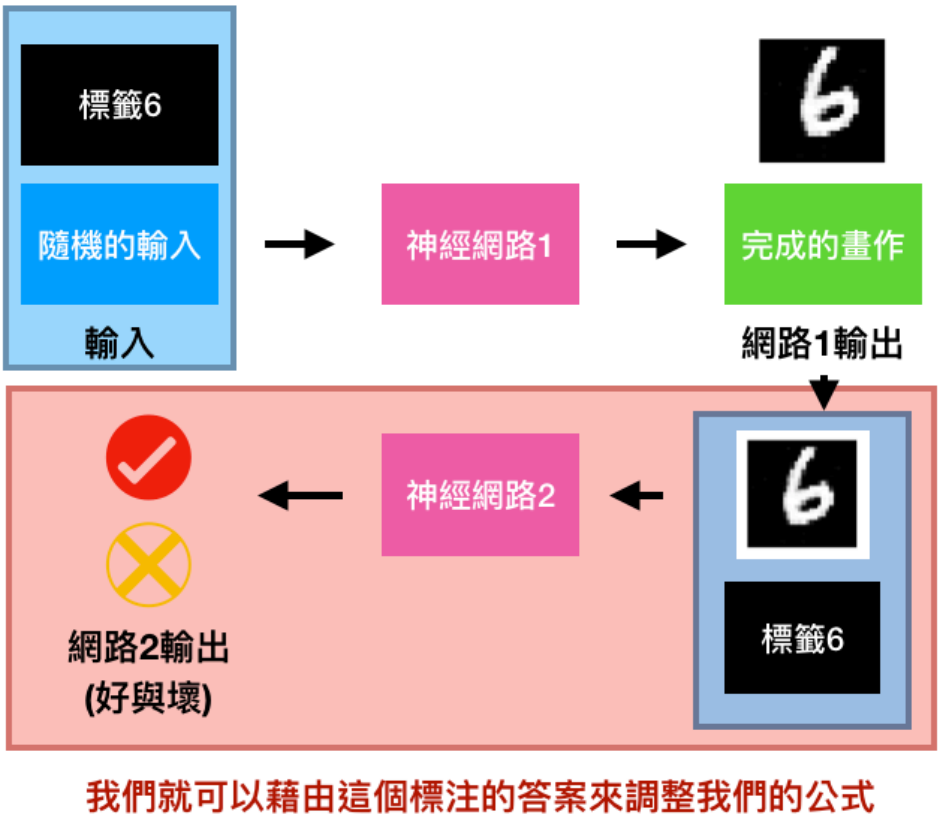
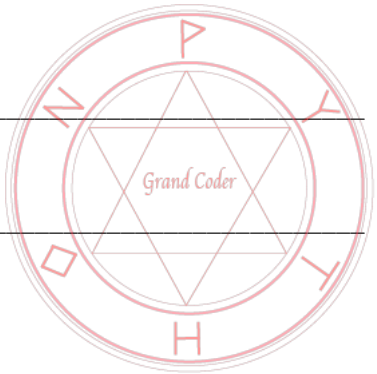


圖: 組合網路來訓練 Generator

[程式]:
```
# 記得在組合網路的時候必須讓鑑賞家保持不動!
cdiscriminator.trainable = False
cgan_input = Input(shape=(random_dim,))
cgan_label = Input(shape=(1, ))
x = cgenerator([cgan_input, cgan_label])
cgan_output = cdiscriminator([x, cgan_label])
cgan = Model(inputs=[cgan_input, cgan_label], outputs=cgan_output)
cgan.compile(loss='binary_crossentropy', optimizer="adam")
cgan.summary()
```

```
_________________________________________________________________________________________
____________
Layer (type)                 Output Shape              Param #     Connected to
=========================================================================================
============
input_5 (InputLayer)         (None, 100)               0
_________________________________________________________________________________________
____________
input_6 (InputLayer)         (None, 1)                 0
_________________________________________________________________________________________
```



```
__________
model_1 (Model)                 (None, 28, 28)       563704      input_5[0][0]
                                                                 input_6[0][0]
__________________________________________________________________________________________
__________
model_2 (Model)                 (None, 1)            574113      model_1[1][0]
                                                                 input_6[0][0]
==========================================================================================
============
Total params: 1,137,817
Trainable params: 562,168
Non-trainable params: 575,649
__________________________________________________________________________________________
__________
```

1.6 ✓ Step5. 開始訓練

一樣對於下面的 cell 執行數次，在 loss 差不多的時候停下結果

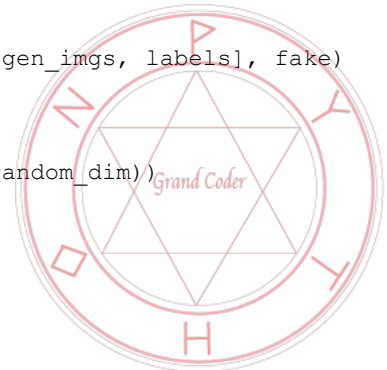
```
[程式]: batch_size = 200
        epoch_count = 100

        for epoch in range(0, epoch_count):
            for batch_count in range(0, 300):
                idx = np.random.randint(0, x_train.shape[0], batch_size)
                imgs = x_train_shaped[idx]
                # 不一樣的點在這裡！我們有把訓練資料的標籤拿出來
                labels = y_train[idx]

                valid = np.ones((batch_size, 1))
                fake = np.zeros((batch_size, 1))
                # 步驟 0: 讓創作家製造出 fake image
                noise = np.random.normal(0, 1, (batch_size, random_dim))
                # 跟普通 GAN 不一樣, 帶入了標籤部分
                gen_imgs = cgenerator.predict([noise, labels])

                # 步驟 1: 讓鑑賞家鑑賞對的 image
                d_loss_real = cdiscriminator.train_on_batch([imgs, labels], valid)
                # 步驟 2: 讓鑑賞家鑑賞錯的 image
                d_loss_fake = cdiscriminator.train_on_batch([gen_imgs, labels], fake)
                d_loss = (d_loss_real + d_loss_fake) / 2

                noise = np.random.normal(0, 1, (batch_size, random_dim))
                # 步驟 3: 訓練創作家的創作能力
```



```
            g_loss = cgan.train_on_batch([noise, labels], valid)

        if (epoch + 1) % 10 == 0:
            dash = "-" * 15
            print(dash, "epoch", epoch + 1, dash)
            print("Discriminator loss:", d_loss)
            print("Generator loss:", g_loss)
```

```
--------------- epoch 10 ---------------
Discriminator loss: 0.5831745862960815
Generator loss: 1.2670963
--------------- epoch 20 ---------------
Discriminator loss: 0.5439196825027466
Generator loss: 1.1932579
--------------- epoch 30 ---------------
Discriminator loss: 0.5193800926208496
Generator loss: 1.1459067
--------------- epoch 40 ---------------
Discriminator loss: 0.588961124420166
Generator loss: 1.0838479
--------------- epoch 50 ---------------
Discriminator loss: 0.5306670665740967
Generator loss: 1.1740686
--------------- epoch 60 ---------------
Discriminator loss: 0.5842047929763794
Generator loss: 1.2247632
--------------- epoch 70 ---------------
Discriminator loss: 0.5230212807655334
Generator loss: 1.2086267
--------------- epoch 80 ---------------
Discriminator loss: 0.5432819128036499
Generator loss: 1.2568197
--------------- epoch 90 ---------------
Discriminator loss: 0.5319997072219849
Generator loss: 1.2006314
--------------- epoch 100 ---------------
Discriminator loss: 0.5373293161392212
Generator loss: 1.269484
```

1.7 ✓ Step6. 訓練結果

你可以看到我們這裡就可以根據我們想要的 label 產生我們想要的數字了！ 而且也挺有模有樣的！

```
[程式]: import matplotlib.pyplot as plt
        %matplotlib inline
        noise = np.random.normal(0, 1, (10, random_dim))
        sampled_labels = np.arange(0, 10).reshape(-1, 1)

        gen_imgs = cgenerator.predict([noise, sampled_labels])

        # Rescale images 0 - 1
        gen_imgs = 0.5 * gen_imgs + 0.5
        gen_imgs = gen_imgs.reshape(10, 28, 28)
        plt.figure(figsize = (14, 14))
        # range(0, 10) 產生出十種不同 label(0 - 9) 針對性的產生我們要的數字
        for i in range(0, 10):
            plt.subplot(1, 10, i + 1)
            plt.axis("off")
            plt.imshow(gen_imgs[i], cmap='gray')
```