

# 你的第一個 GAN 生成手寫數字

*Chapter 3*



# Outline

**3.1 GAN的基礎：對抗訓練**

**3.2 生成器與鑑別器的目標差異**

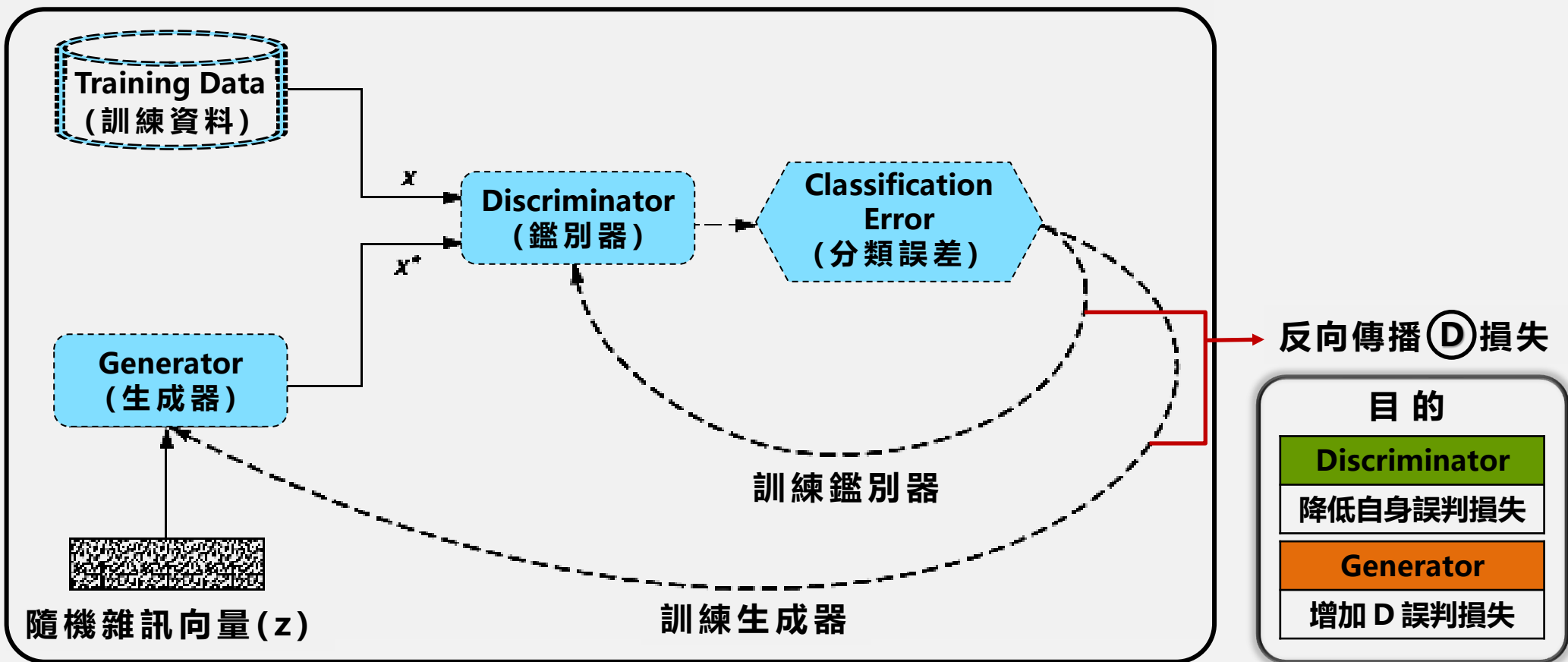
**3.3 GAN的訓練程序**

**3.4 實例：生成手寫數字**

**3.5 重點整理**

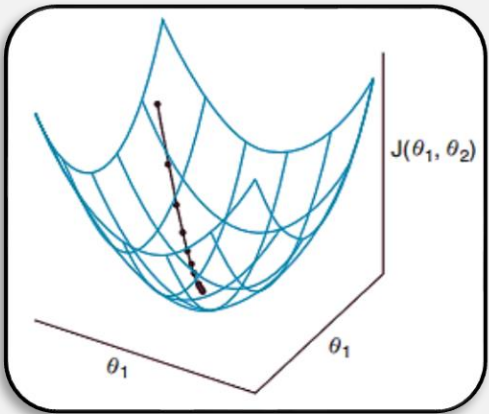
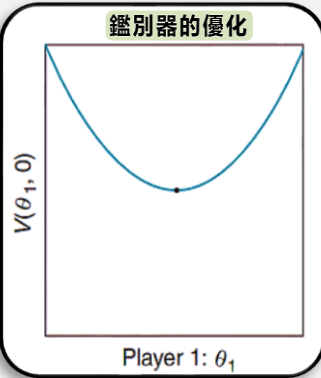
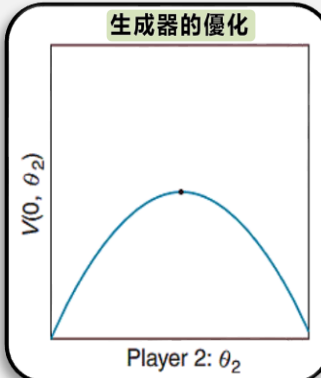
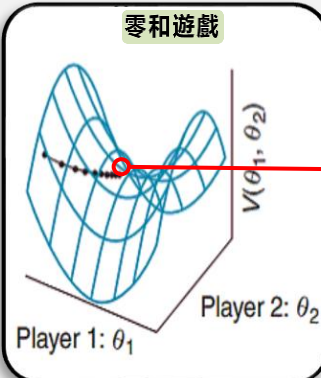
# 3.1 GAN的基礎：對抗訓練

GAN 架構圖

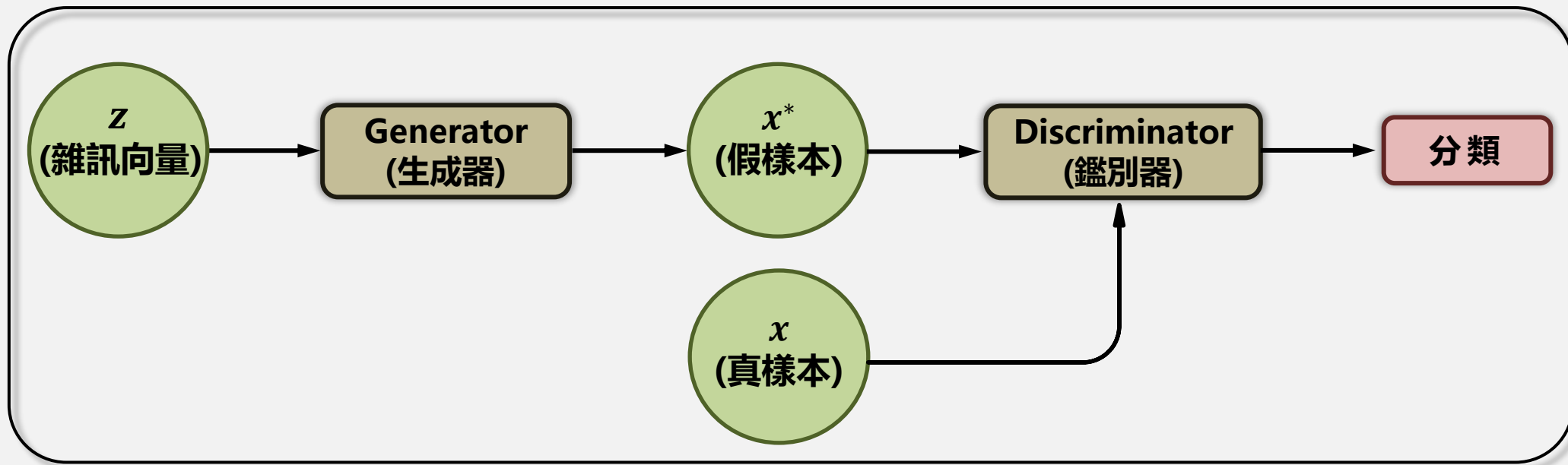


# 3.1 GAN的基礎：對抗訓練

## 傳統神經網路 vs. GAN神經網路

	傳統神經網路	GAN神經網路
損失函數	$J^{(D)}(\theta^{(D)})$ $J^{(G)}(\theta^{(G)})$	$J^{(D)}(\theta^{(D)}, \theta^{(G)})$ $J^{(G)}(\theta^{(D)}, \theta^{(G)})$
可調整參數	所有參數 $\theta^{(D)}$ 、 $\theta^{(G)}$	鑑別器 NN: $\theta^{(D)}$ 生成器 NN: $\theta^{(G)}$ <div>GAN 含兩組神經網路 只能各自調整自己的參數</div>
訓練型態	損失最小化的優化過程	比賽 (直到 NN 間達納許均衡)
示意圖		<div> <div>  <p>鑑別器的優化</p> <p>將 V 最小化</p> </div> <div>  <p>生成器的優化</p> <p>將 V 最大化</p> </div> <div>  <p>零和遊戲</p> <p>綜合損失</p> <p>收斂</p> </div> </div>

## 3.2 生成器與鑑別器的目標差異



生成器隨機取一批  
雜訊向量生成假樣本

數學式： $G(z) = x^*$

將真樣本、假樣本  
輸入鑑別器

鑑別器判斷  
輸入樣本的真偽

## 3.2 生成器與鑑別器的目標差異

### 混淆矩陣 (Confusion matrix)

#### 功能

→ 評估模型表現的優劣

#### 衡量指標

- $\text{Accuracy} = (\text{TP} + \text{TN}) / \text{TN}$
- $\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$
- $\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$

輸入	鑑別器輸出	
	接近 1 (Real)	接近 0 (Fake)
$x$ (真樣本)	TP (真陽性)	FP (偽陰性) <u>Type I error (<math>\alpha</math>)</u>
$x^*$ (假樣本)	FN (偽陽性) <u>Type II error (<math>\beta</math>)</u>	TN (真陰性)

#### 目標衝突：

Ⓓ →  $TP (D(x) \approx 1)$ 、 $TN (D(x^*) \approx 0)$

Ⓔ →  $FN (D(x^*) \approx 1)$

## 3.3 GAN的訓練程序

### 訓練演算法

For 每次訓練 do

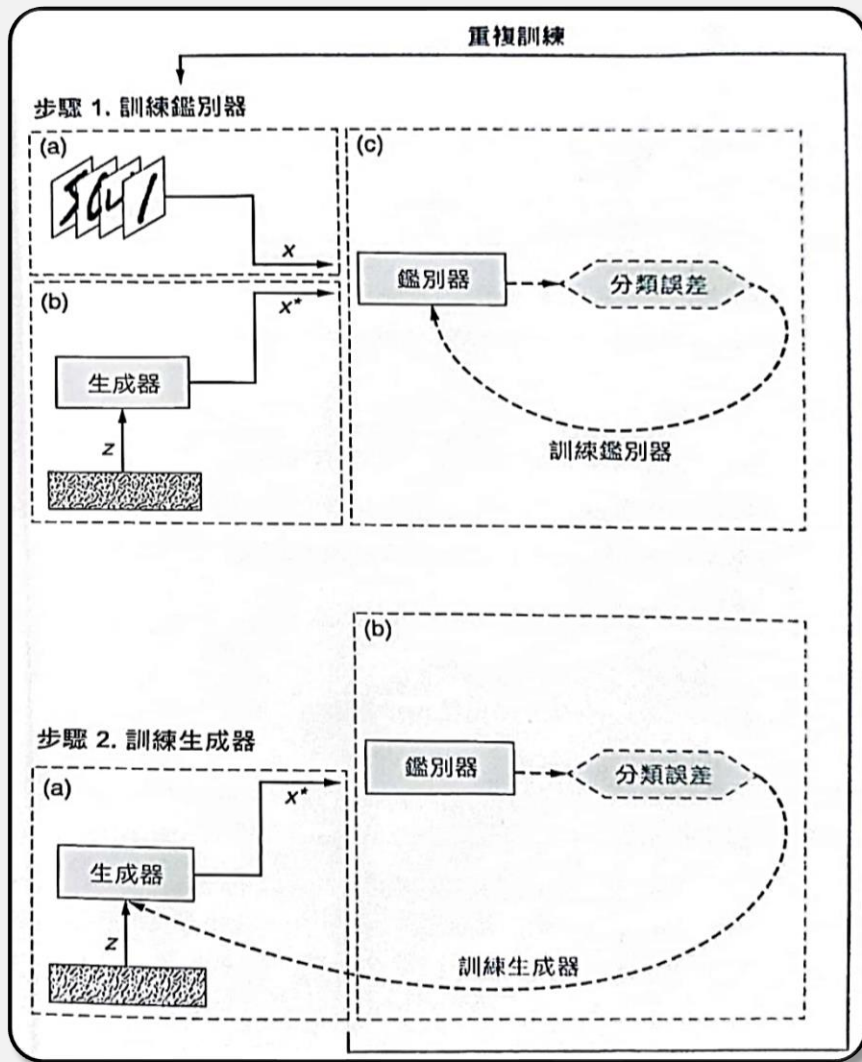
#### STEP 1 訓練鑑別器

- 從訓練集隨機取一批( $x$ )
- 取一小批( $z$ )生成( $x^*$ )  $\rightarrow G(z) = x^*$
- 計算  $D(x)$ 、 $D(x^*)$  並反向傳播總誤差  
 $\rightarrow$  調整參數  $\theta^{(D)}$ ，將分類損失最小化

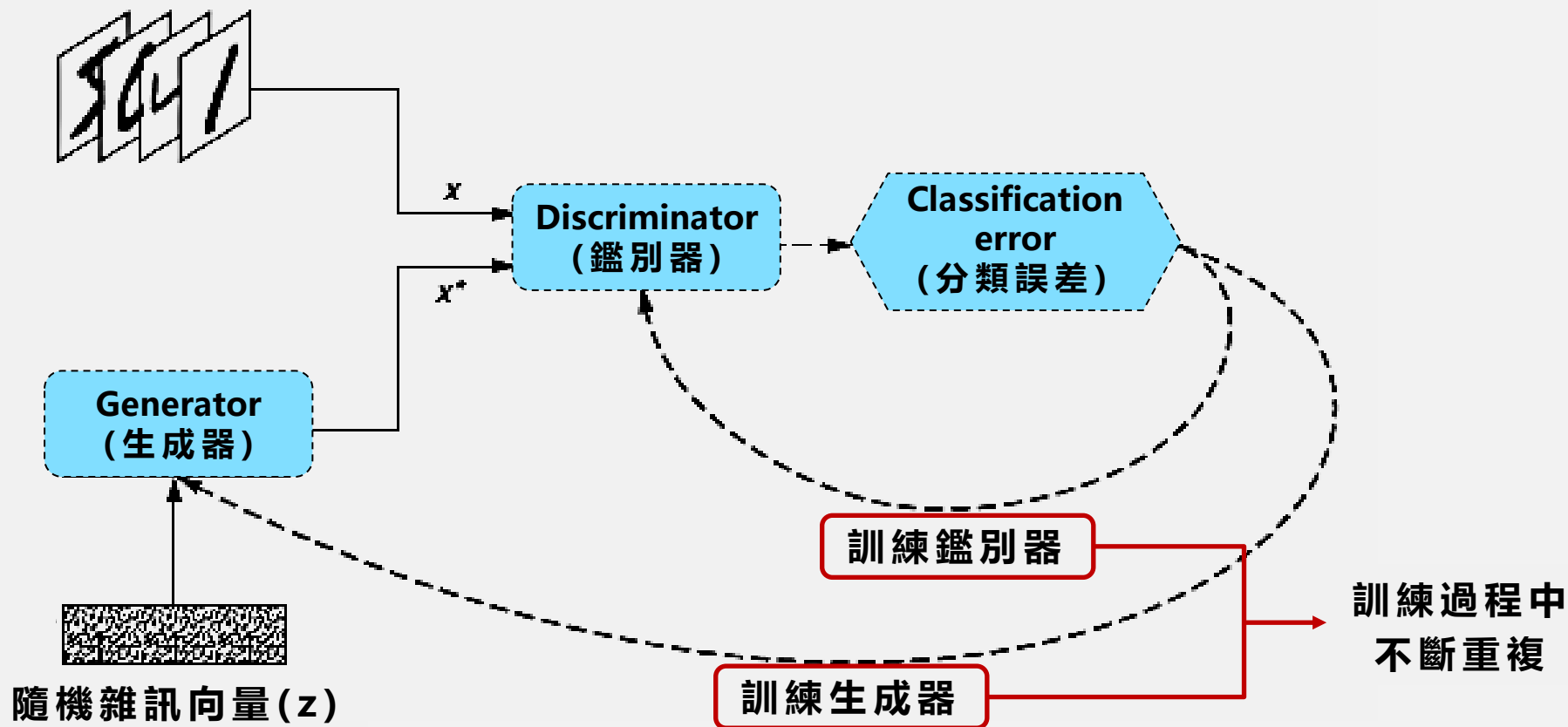
#### STEP 2 訓練生成器

- 取一小批( $z$ )生成( $x^*$ )  $\rightarrow G(z) = x^*$
- 計算  $D(x^*)$  並反向傳播誤差  
 $\rightarrow$  調整參數  $\theta^{(G)}$ ，將分類損失最大化

End for



### 3.4 實例：生成手寫數字





## 3.4 實例：生成手寫數字

### 流程

**STEP 1** 匯入模組並設定模型輸入層維度

**STEP 2** 定義函式

- `build_generator ()`
- `build_discriminator ()`
- `build_gan ()`

**STEP 3** 建立並編譯模型

**STEP 4** 定義訓練函式

- `train ()`

**STEP 5** 定義產生圖片的函式

- `sample_images ()`

**STEP 6** 開始訓練

**FINISH** 輸出結果、產生圖片

## 3.4 實例：生成手寫數字

### 匯入模組

```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np

from keras.datasets import mnist
from keras.layers import Dense, Flatten, Reshape
from keras.layers import LeakyReLU
from keras.models import Sequential
from keras.optimizers import Adam
```

### 設定模型輸入層維度

```
img_rows = 28
img_cols = 28
channels = 1 #MNIST圖像皆為灰階

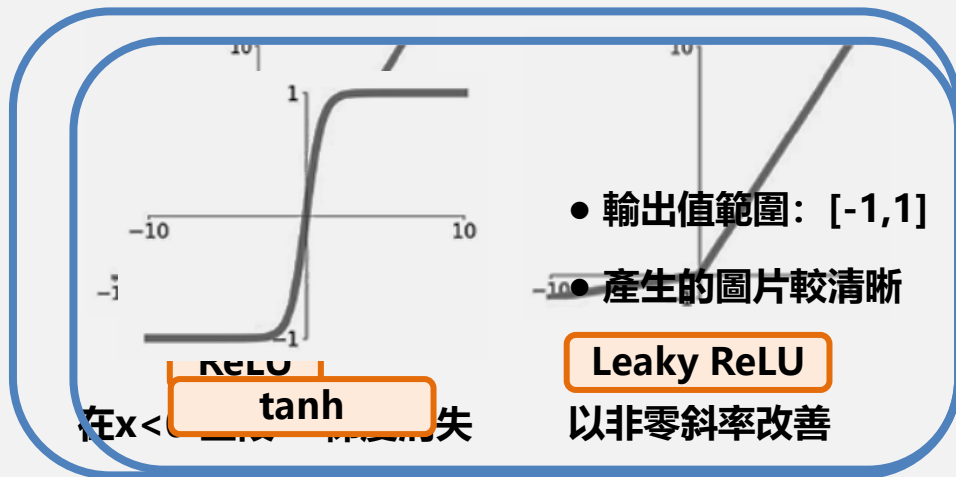
img_shape = (img_rows, img_cols, channels) # 輸入圖像的尺寸

z_dim = 100 #指定生成器隨機向量的長度
```

## 3.4 實例：生成手寫數字

### 實作生成器

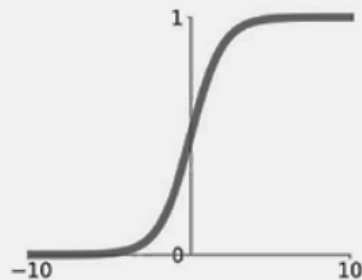
```
def build_generator(img_shape, z_dim):  
    model = Sequential()  
  
    model.add(Dense(128, input_dim=z_dim)) # 建立全連接層(隱藏層)  
  
    model.add(LeakyReLU(alpha=0.01)) # 隱藏層激活函數: Leaky ReLU  
  
    model.add(Dense(28 * 28 * 1, activation='tanh')) # 輸出層激活函數: tanh  
  
    model.add(Reshape(img_shape)) # 重塑輸出圖片的維度  
  
    return model
```



## 3.4 實例：生成手寫數字

### 實作鑑別器

```
def build_discriminator(img_shape):  
    model = Sequential()  
  
    model.add(Flatten(input_shape=img_shape)) # 將輸入圖像攤平成一維  
  
    model.add(Dense(128)) # 建立全連接層(隱藏層)  
  
    model.add(LeakyReLU(alpha=0.01)) # 激活函數: Leaky ReLU  
  
    model.add(Dense(1, activation='sigmoid')) # 激活函數: sigmoid  
  
    return model
```



sigmoid

- 輸出值範圍:  $[0,1]$
- 代表輸入圖片為真的機率

## 3.4 實例：生成手寫數字

### 建立與編譯 GAN

#### # 定義 GAN 函式

```
def build_gan(generator, discriminator):
```

```
    model = Sequential()
```

```
    model.add(generator)
    model.add(discriminator)
```



# 結合生成器與鑑別器

```
    return model
```

## 3.4 實例：生成手寫數字

### 建立與編譯 GAN

#### # 建立和編譯鑑別器

```
discriminator = build_discriminator(img_shape)
discriminator.compile(loss= 'binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])
```

#### # 鎖定鑑別器參數以便訓練生成器

```
discriminator.trainable = False
```

#### # 建立生成器

```
generator = build_generator(img_shape, z_dim)
```

#### # 建立和編譯 GAN 模型用以訓練生成器

```
gan = build_gan(generator, discriminator)
gan.compile(loss= 'binary_crossentropy', optimizer=Adam())
```

#### 二元交叉熵

- 評估二分類時  
預測與實際損失的差距
- 越小越好

## 3.4 實例：生成手寫數字

### 撰寫GAN訓練函式

# 定義用來儲存損失率、準確率及迭代次數的陣列

```
losses = []
```

```
accuracies = []
```

```
iteration_checkpoints = []
```

```
def train(iterations, batch_size, sample_interval):
```

```
(X_train, _, _) = mnist.load_data() # 載入MNIST資料集
```

```
X_train = X_train / 127.5 - 1.0 # 將灰階像素質由 [0, 255] 轉換為 [-1, 1]
```

```
X_train = np.expand_dims(X_train, axis=3)
```

```
real = np.ones((batch_size, 1)) # 將真圖片標籤設為 1
```

```
fake = np.zeros((batch_size, 1)) # 將假圖片標籤設為 0
```

#### Note

(X\_train, Y\_train), (X\_test, Y\_test)

將此三個不會用到的變數以底線表示

生成器輸出層激勵函數: tanh

## 3.4 實例：生成手寫數字

### 撰寫GAN訓練函式

**for** iteration in range(iterations):

**# 隨機挑一批真圖片**

idx = np.random.randint(0, X\_train.shape[0], batch\_size)

imgs = X\_train[idx]

**# 生成一批假圖片**

z = np.random.normal(0, 1, (batch\_size, 100))

gen\_imgs = generator.predict(z)

從標準常態分布中取樣

**# 訓練鑑別器**

d\_loss\_real = discriminator.train\_on\_batch(imgs, real)

d\_loss\_fake = discriminator.train\_on\_batch(gen\_imgs, fake)

d\_loss, accuracy = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

鑑別器較佳的輸出

- 真陽性
- 真陰性



## 3.4 實例：生成手寫數字

### 撰寫 GAN 訓練函式

```
z = np.random.normal(0, 1, (batch_size, 100))
```

從標準常態分布中取樣

```
# 訓練生成器
```

```
g_loss = gan.train_on_batch(z, real)
```

```
if (iteration + 1) % sample_interval == 0:
```

```
# 紀錄損失率及準確率
```

```
losses.append((d_loss, g_loss))
```

```
accuracies.append(100.0 * accuracy)
```

```
iteration_checkpoints.append(iteration + 1)
```

```
# 顯示訓練過程
```

```
print("%d [D loss: %f, acc.: %.2f%%] [G loss: %f]" %  
      (iteration + 1, d_loss, 100.0 * accuracy, g_loss))
```

```
# 生成並顯示圖片
```

```
sample_images(generator)
```

生成器的目的

生成出近似真圖片的擬真樣本

## 3.4 實例：生成手寫數字

### 生成並顯示圖片

```
def sample_images(generator, image_grid_rows=4, image_grid_columns=4):  
    z = np.random.normal(0, 1, (image_grid_rows * image_grid_columns, z_dim)) # 取一組隨機雜訊  
    gen_imgs = generator.predict(z) # 用這組隨機雜訊生成圖片  
    gen_imgs = 0.5 * gen_imgs + 0.5 # 將圖片像素質由 [-1, 1] 轉換為 [0, 1]  
    fig, axs = plt.subplots(image_grid_rows, image_grid_columns, figsize=(4, 4),  
                             sharey=True, sharex=True) # 設定圖片網格  
  
    cnt = 0  
    for i in range(image_grid_rows):  
        for j in range(image_grid_columns):  
            axs[i, j].imshow(gen_imgs[cnt, :, :, 0], cmap= 'gray' )  
            axs[i, j].axis('off' )  
            cnt += 1
```

## 3.4 實例：生成手寫數字

### 訓練模型

# 設定超參數

iterations = 20000

batch\_size = 128

sample\_interval = 1000

# 用指定的迭代次數訓練GAN

train(iterations, batch\_size, sample\_interval)

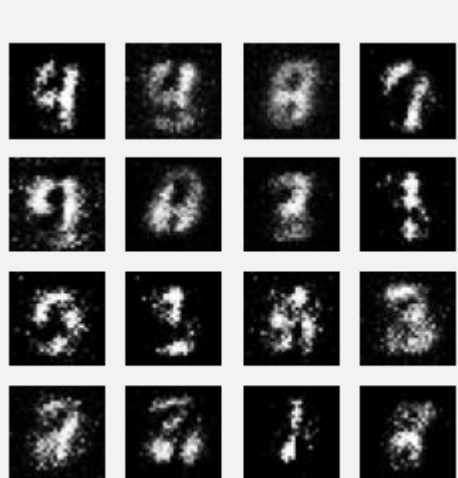
## 3.4 實例：生成手寫數字

### 訓練結果

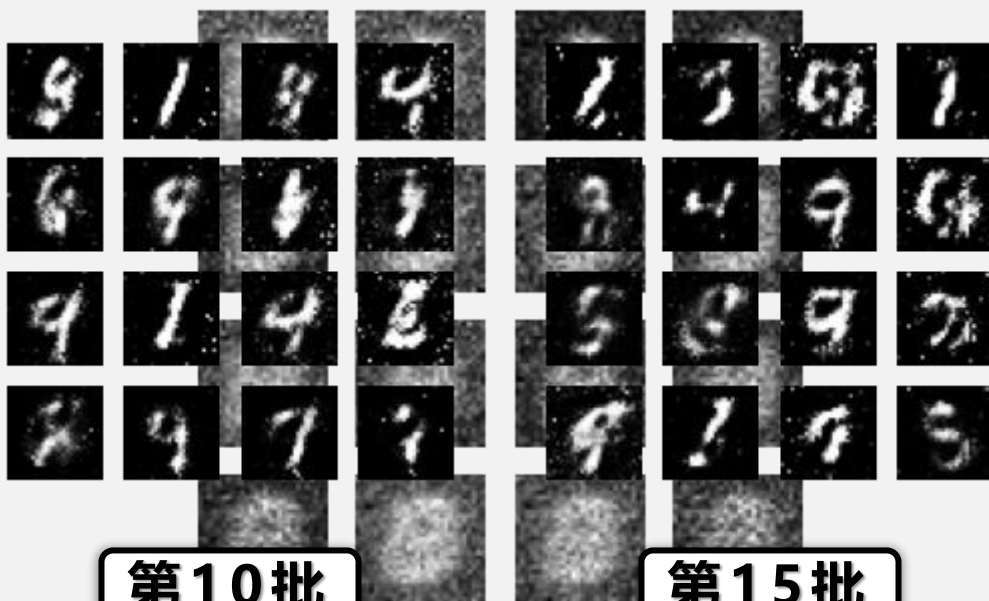
```
1000 [D loss: 0.020129, acc.: 100.00%] [G loss: 4.307247]
2000 [D loss: 0.127398, acc.: 95.31%] [G loss: 3.321208]
3000 [D loss: 0.265902, acc.: 90.23%] [G loss: 3.658431]
4000 [D loss: 0.354625, acc.: 86.33%] [G loss: 3.475443]
5000 [D loss: 0.190033, acc.: 92.19%] [G loss: 5.315524]
6000 [D loss: 0.219283, acc.: 90.62%] [G loss: 4.656998]
7000 [D loss: 0.135775, acc.: 94.92%] [G loss: 5.939697]
8000 [D loss: 0.168335, acc.: 91.80%] [G loss: 4.860357]
9000 [D loss: 0.208104, acc.: 91.02%] [G loss: 4.896073]
10000 [D loss: 0.424239, acc.: 83.98%] [G loss: 4.404410]
11000 [D loss: 0.411975, acc.: 83.98%] [G loss: 3.948246]
12000 [D loss: 0.406832, acc.: 81.64%] [G loss: 3.000345]
13000 [D loss: 0.392198, acc.: 82.42%] [G loss: 3.617300]
14000 [D loss: 0.380430, acc.: 82.42%] [G loss: 3.345677]
15000 [D loss: 0.349196, acc.: 83.98%] [G loss: 3.862607]
16000 [D loss: 0.302157, acc.: 88.67%] [G loss: 2.660031]
17000 [D loss: 0.344069, acc.: 83.98%] [G loss: 3.083309]
18000 [D loss: 0.319945, acc.: 85.94%] [G loss: 3.481219]
19000 [D loss: 0.590636, acc.: 75.39%] [G loss: 2.744638]
20000 [D loss: 0.442096, acc.: 76.56%] [G loss: 2.529825]
```

## 3.4 實例：生成手寫數字

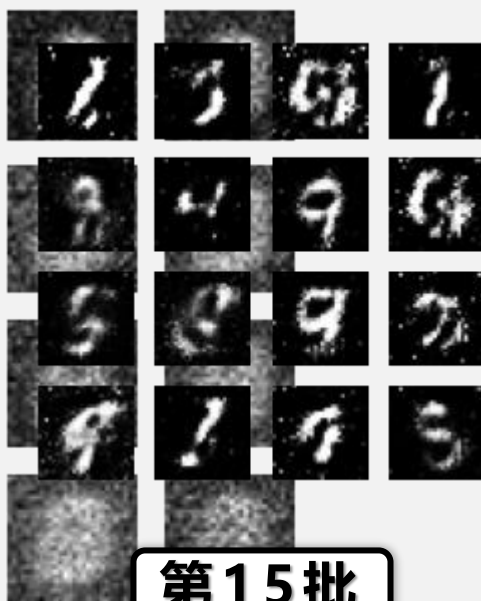
輸出圖片



第5批



第10批



第15批



第20批

第1批

## 3.4 實例：生成手寫數字

### 繪製鑒別器和生成器的訓練損失

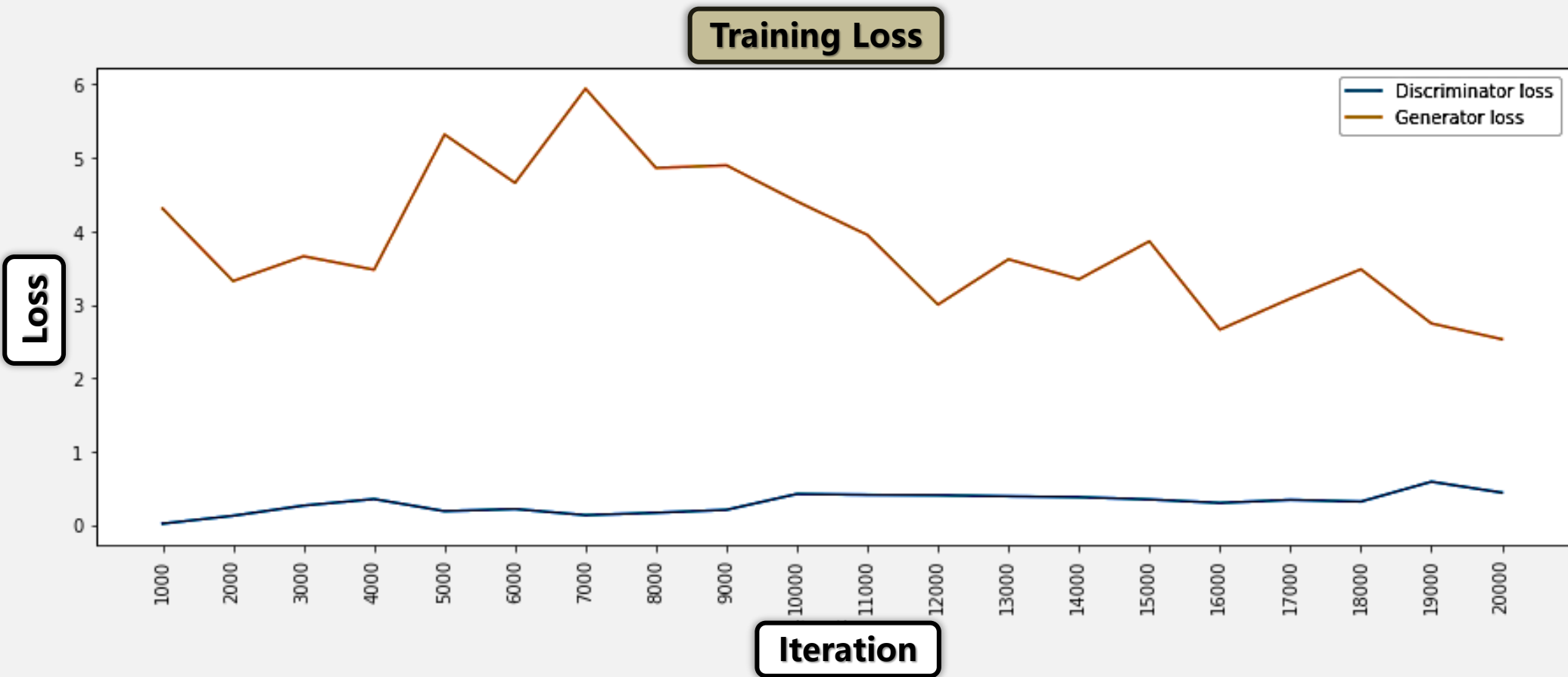
```
losses = np.array(losses)

plt.figure(figsize=(15, 5))
plt.plot(iteration_checkpoints, losses.T[0], label="Discriminator loss")
plt.plot(iteration_checkpoints, losses.T[1], label="Generator loss")

plt.xticks(iteration_checkpoints, rotation=90)

plt.title("Training Loss")
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.legend()
```

## 3.4 實例：生成手寫數字



## 3.4 實例：生成手寫數字

### 繪製鑒別器的準確率

```
accuracies = np.array(accuracies)

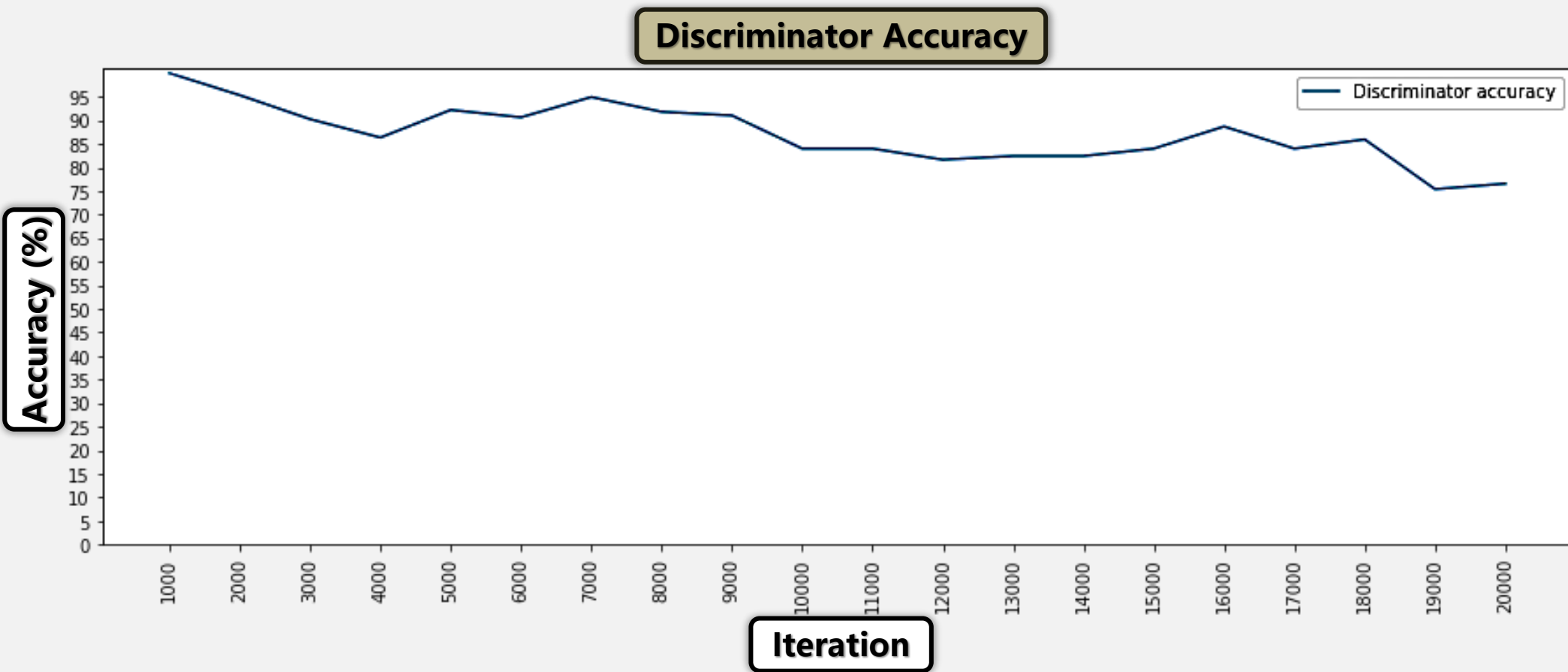
plt.figure(figsize=(15, 5))
plt.plot(iteration_checkpoints, accuracies, label="Discriminator accuracy")

plt.xticks(iteration_checkpoints, rotation=90)
plt.yticks(range(0, 100, 5))

plt.title("Discriminator Accuracy")
plt.xlabel("Iteration")
plt.ylabel("Accuracy (%)")
plt.legend()
```



## 3.4 實例：生成手寫數字



## 3.5 重點整理

### 重點

- 1 生成器的目標：生成出近似真圖片的擬真樣本  
鑑別器的目標：盡可能準確判斷圖片的真偽
- 2 GAN的兩個神經網路僅能各自調整自己的參數 $\theta$ 
  - 生成器： $J^{(G)}(\theta^{(G)}, \theta^{(D)}) \rightarrow \theta^{(G)}$
  - 鑑別器： $J^{(G)}(\theta^{(G)}, \theta^{(D)}) \rightarrow \theta^{(D)}$