深度學習TensorFlow實務

GAN

Lab8

-TA-

李偉弘

廖宜健

林佑昌

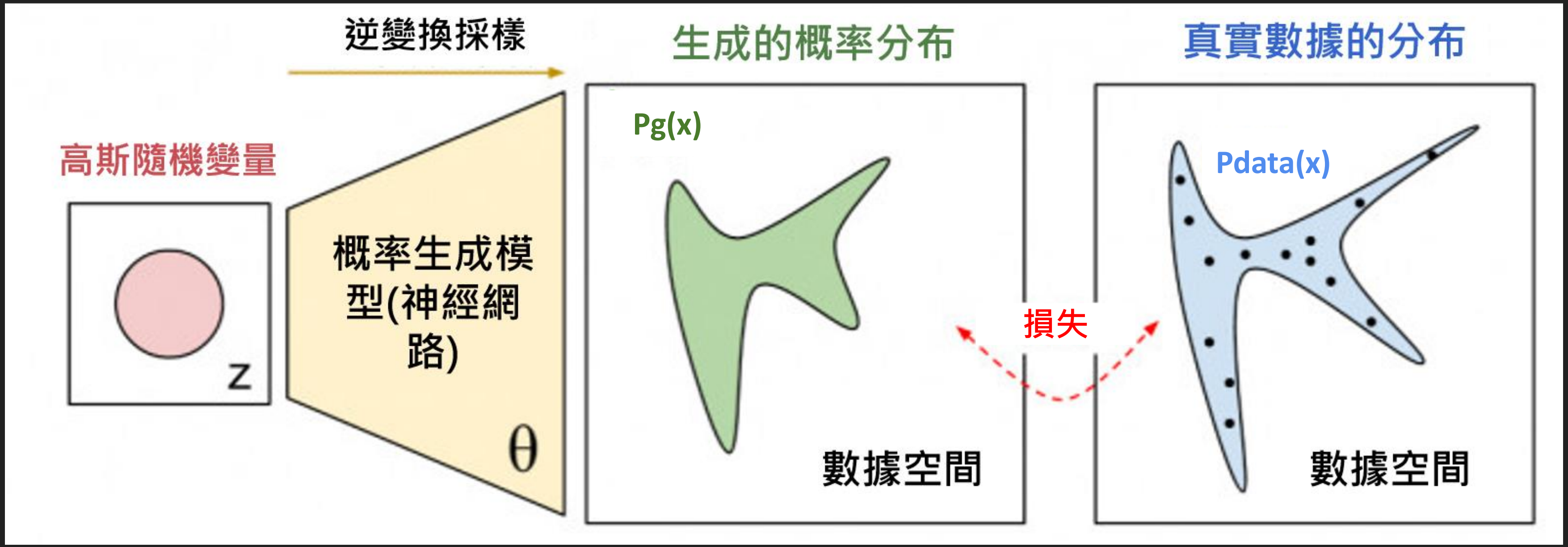
蔡明諺

彭冠偉

1. GAN 介紹

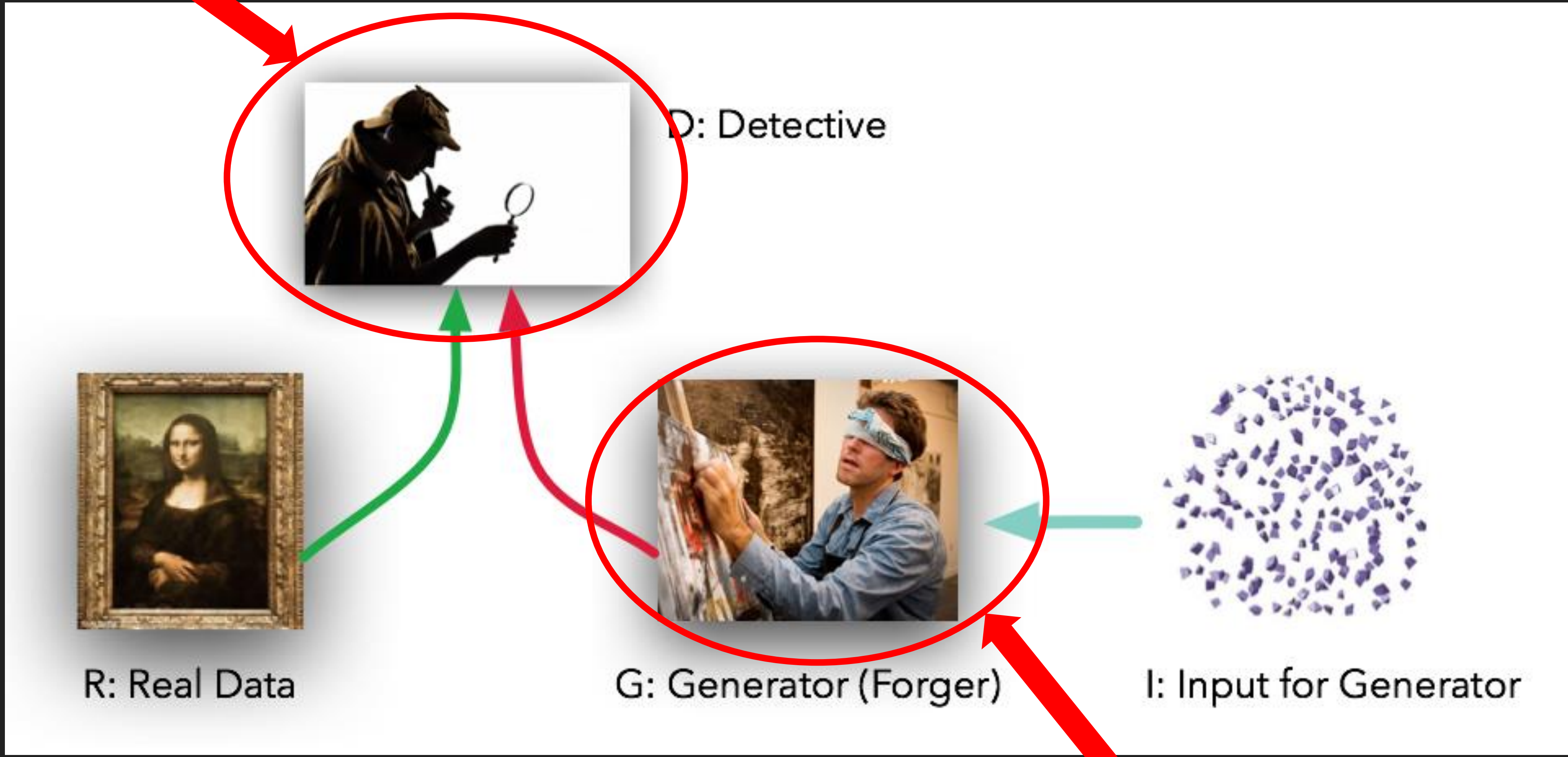
GAN概念

- 生成對抗網路(Generative adversarial networks, GAN)，其目的是為了透過模擬資料機率分布，使得這種機率分布與實際資料分布的機率統計分布一致。



GAN概念

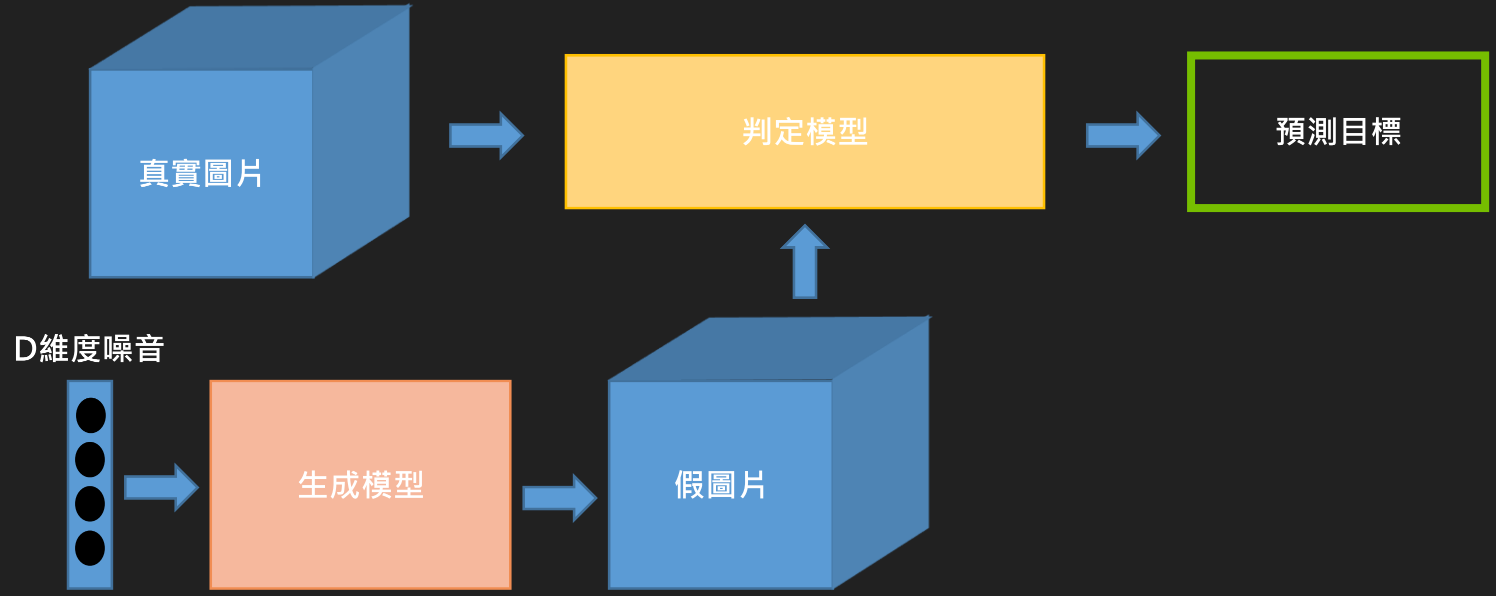
偵探(判別器)



造假者(生成器)

GAN模式

- 其目的為生成資料因而有兩個模型，一個為生成模型(Generative model)，一個為判定模型(Discriminative model)，藉由生成模型來生成新的樣本，而判定模型則是將生成樣本與實際樣本作比對。



GAN概念

- 算法的目標是令生成模型生成與真實數據幾乎沒有區別的樣本，將隨機變量生成為某一種概率分佈，也可以說概率密度函數為相等的：Pg(x)=Pdata(x)。

$$D_G^*(x) = \frac{Pdata(x)}{Pdata(x) + Pg(x)}$$

Pdata(x)：真實數據分布

Pg(x)：生成器數據分布

GAN概念

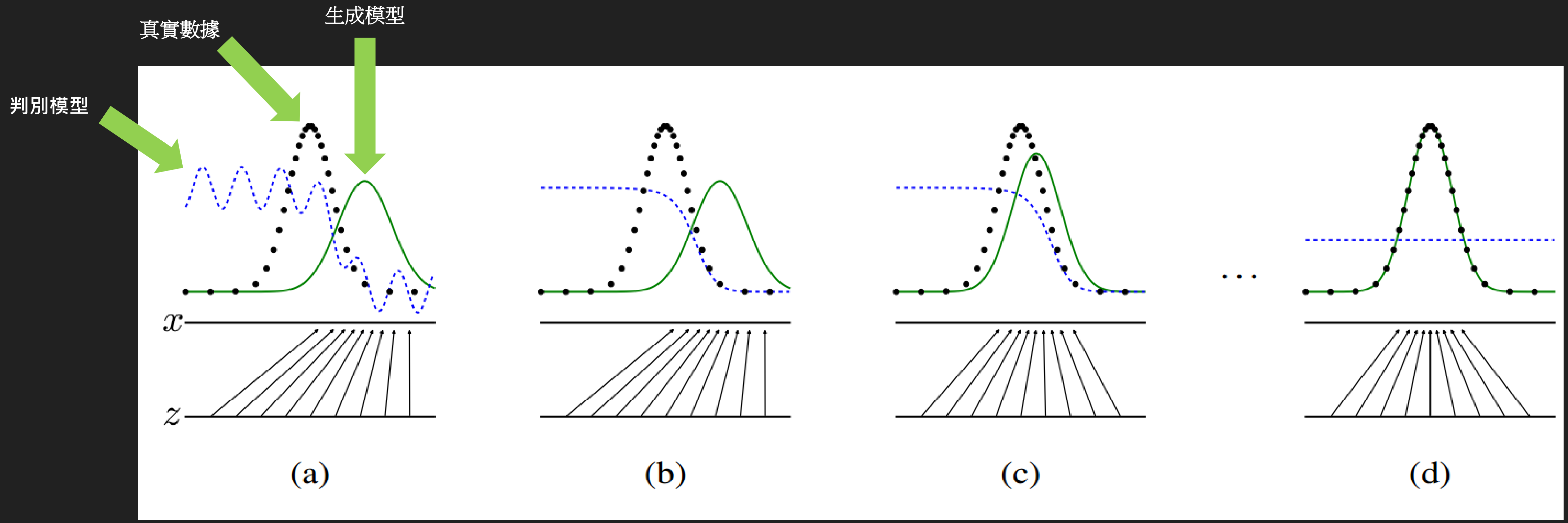
- 為了學習到生成模型在數據 x 上的分佈Pg，先定義一個先驗的輸入噪聲變量Pz(z)，然後根據G(z) 將其映射到數據空間中，其中G 為多層感知機所表徵的可微函數。
- D(x) 表示 x 來源於真實數據而不是 Pg 的概率，訓練 D 以最大化正確分辨真實樣本和生成樣本的概率，就可以通過最小化 log(1-D(G(z))) 而同時訓練 G。

$$\min_{G}\max_{D} V(D,G) = E_{x\sim Pdata(x)}[logD(x)] + E_{z\sim Pz(z)}[\log(1-D(G(z)))]$$

D(x)：指圖片被判斷為真實的機率

G(z)：指一個z噪音輸入到G網路，並輸出一個圖片

GAN分析



a.考慮在收斂點附近的對抗訓練：Pg 和 Pdata 已經十分相似，D 是一個局部準確的分類器。

b.在算法內部循環中訓練 D 以從數據中判別出真實樣本，該循環最終會收斂到$D(x) = \frac{Pdata(x)}{Pdata(x)+Pg(x)}$

c.隨後固定判定模型並訓練生成模型，在更新 G 之後，D 的梯度會引導 G（z）流向更可能被 D 分類為真實數據的方向

d.經過多次訓練後，如果 G 和 D 有足夠的複雜度，那麼就會到達一個均衡點。這個時候 Pg=Pdata，即生成器的概率密度函數等於真實數據的概率密度函數，也即生成的數據和真實數據是一樣的。

優化問題

- 定義一個判別模型 D 以判別樣本是不是從 Pdata(x) 分佈中取出來的。

$$\mathrm{E}_{x\sim Pdata(x)} logD(x)$$

E 指代取期望

- 最大化這一項相當於令判別模型 D 在 x 於 data 的概率密度時能準確地預測 D(x)=1。

$$D(x) = 1 \; when \; x \sim Pdata(x)$$

優化問題

- 企圖欺騙判別器的生成器 G，該項根據「負類」的對數損失函數而構建。

$$E_{z\sim Pz(x)}\log(1-D(G(z)))$$

- 因為 x<1 的對數為負，那麼如果最大化該項的值，則需要令均值 D(G(z))≈0，因此 G 並沒有欺騙 D，為了結合這兩個概念，判定模型的目標為最大化：

$$E_{x\sim Pdata(x)}\log D(x)+E_{z\sim Pz(x)}\log(1-D\big(G(z)\big))$$

優化問題

- 對於 D 而言要盡量使公式最大化（識別能力強），而對於 G 又想使之最小（生成的數據接近實際數據）。
- 整個訓練是一個迭代過程。

$$\min_{G}\max_{D} V(D,G) = E_{x\sim Pdata(x)}[logD(x)] + E_{z\sim Pz(z)}[\log(1-D(G(z)))]$$

2. 變形GAN

AttGAN

- 修圖應用

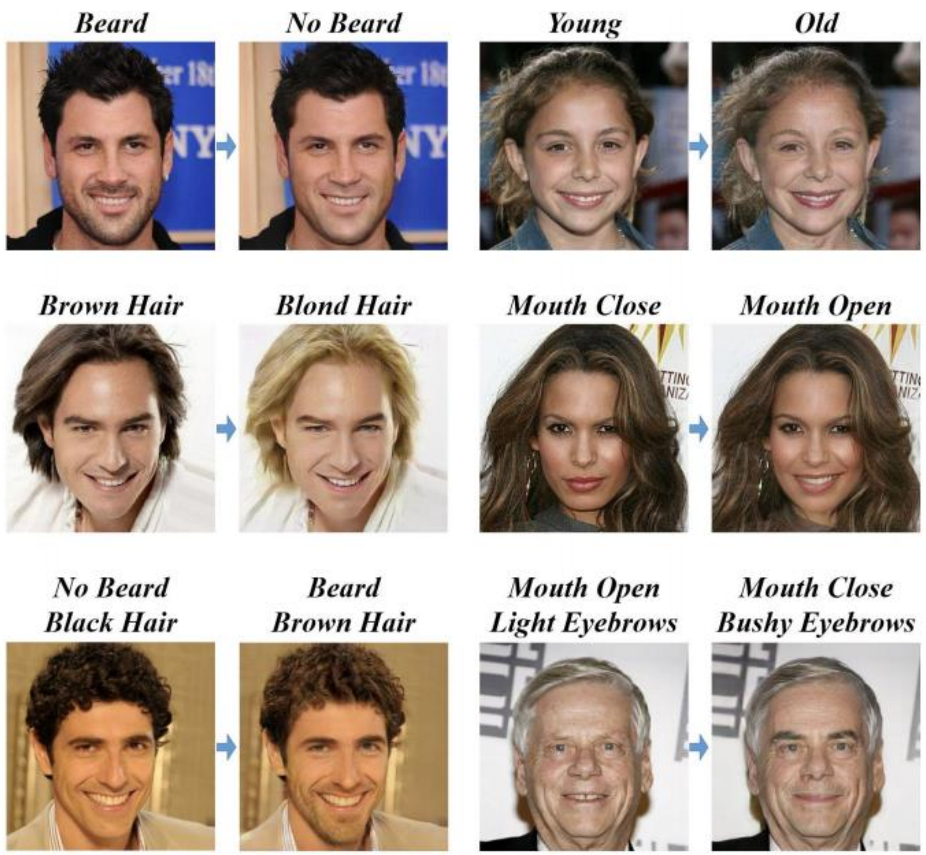


Fig. 1. Facial attribute editing results from our AttGAN. Zoom in for better resolution.

BicycleGAN

- 能夠把給定的夜晚畫面合成具有不同的亮度、天空和雲的白天畫面。

CycleGan

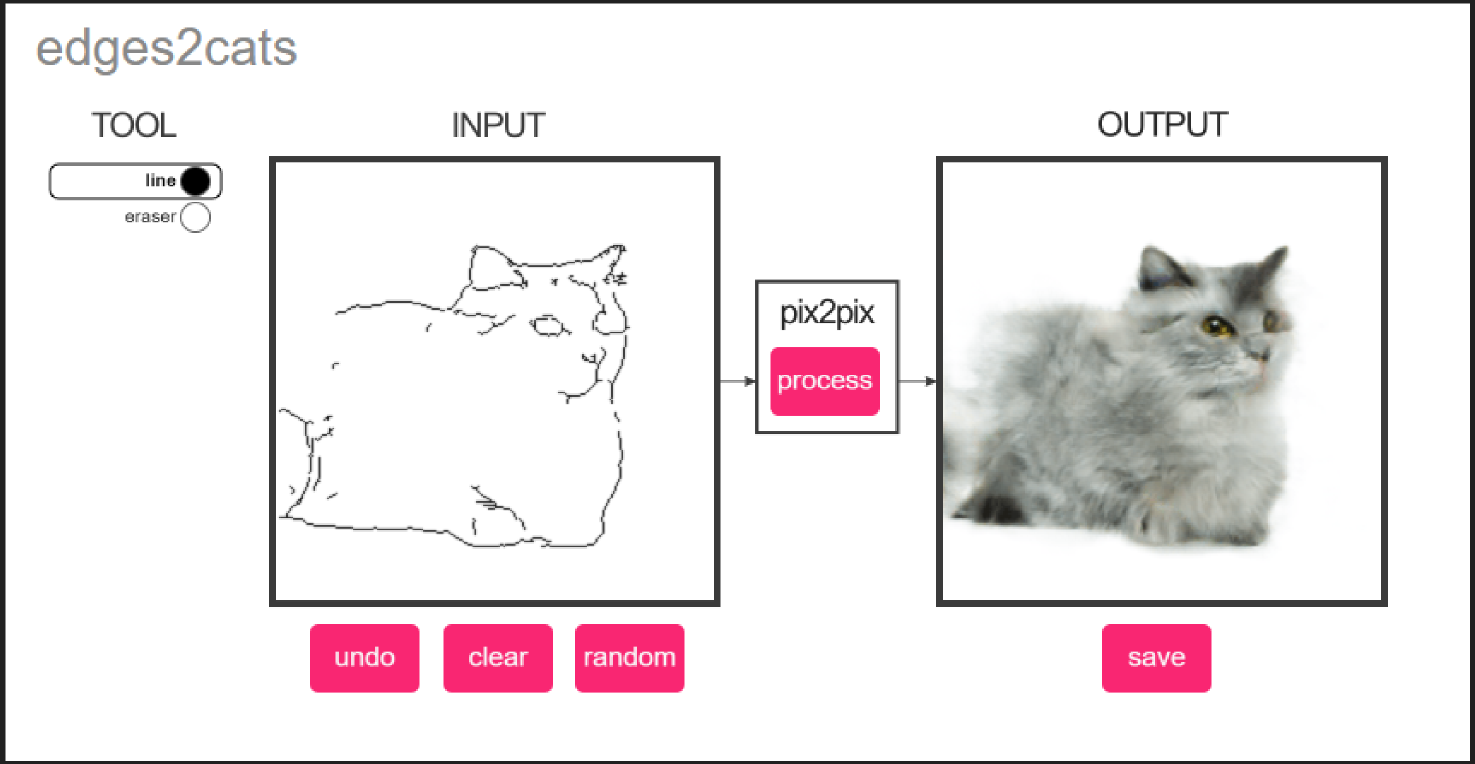
- 把馬變斑馬

DeblurGAN

- 將模糊的圖片做銳化

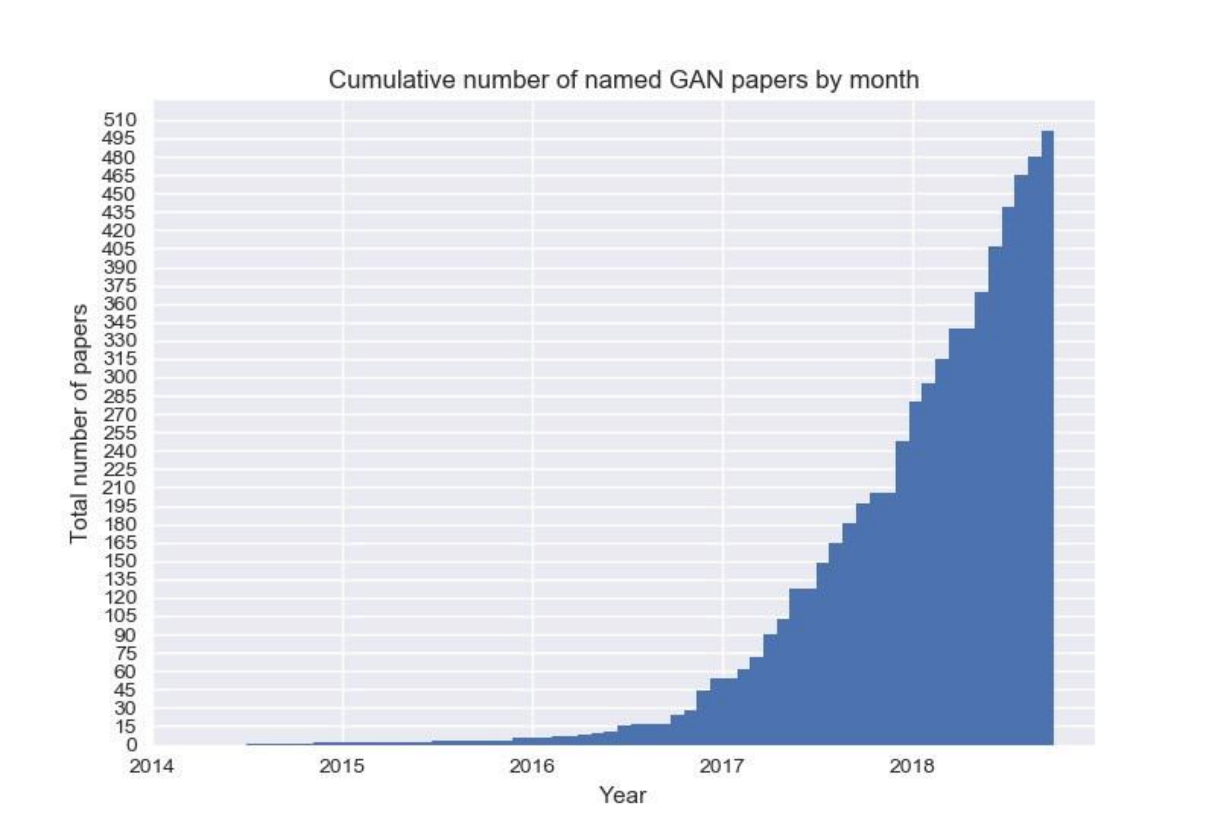
Image-to-Image Demo

- https://affinelayer.com/pixsrv/



GAN

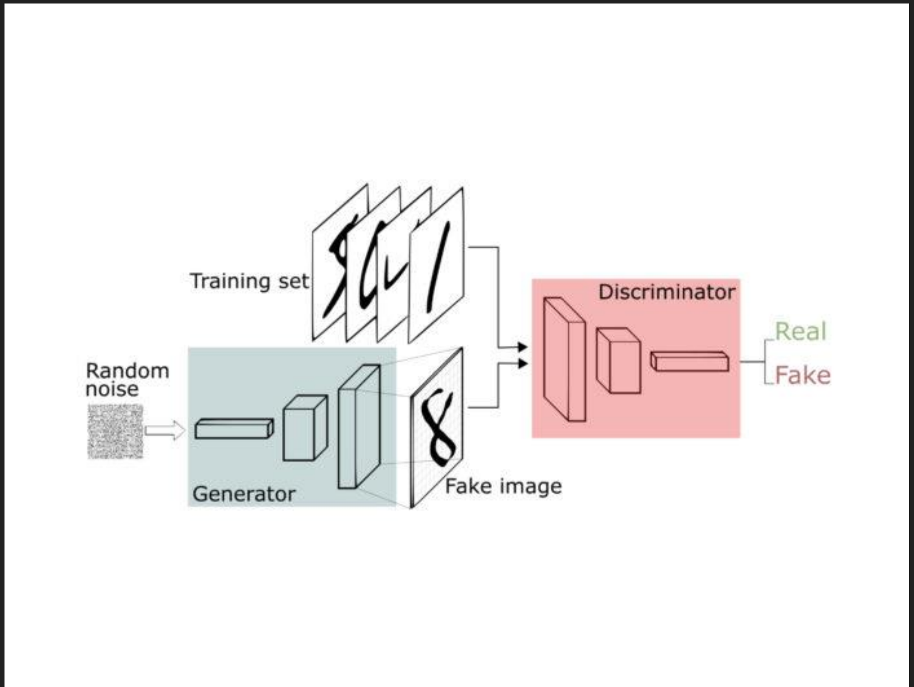
- 還有更多的應用領域，像是：電玩/醫療/資安/時尚/物理/音樂....等等，GAN 為非監督式學習帶來有趣的活力。

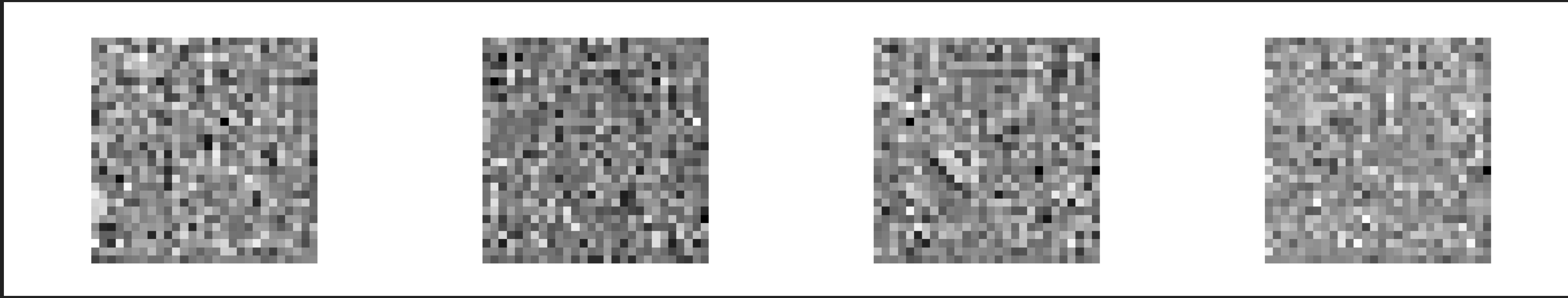


2. 實作GAN

GAN

- 透過 MNIST 資料集實現 GAN 效果。





程式解析

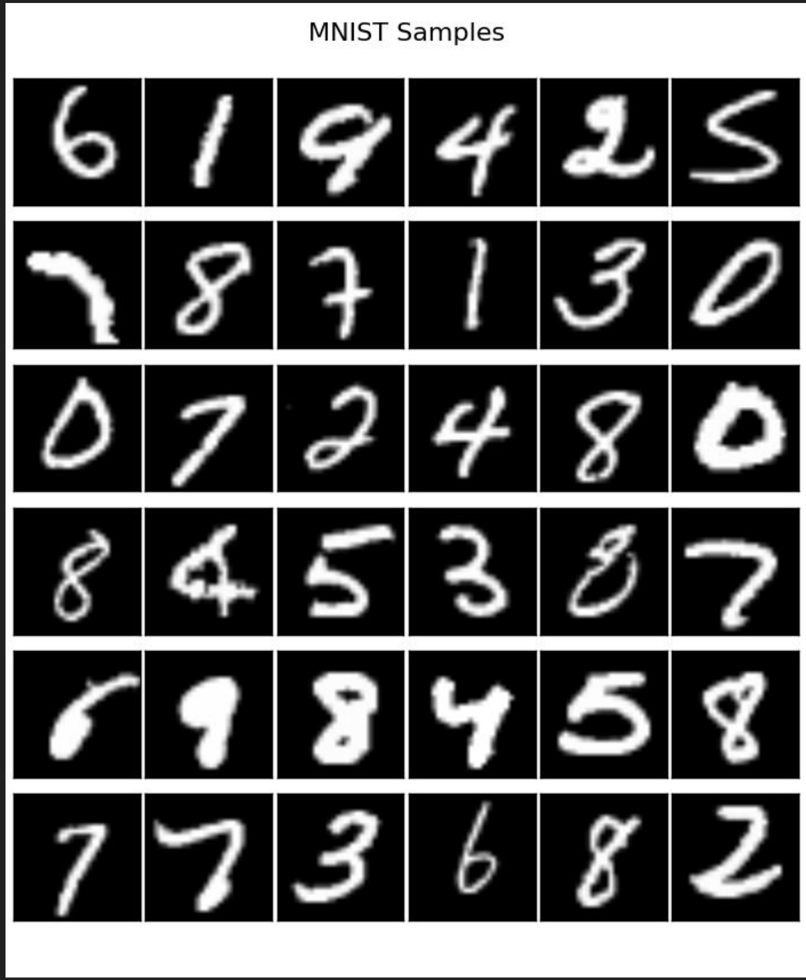
- MNIST的內容就是手寫的數字0–9（圖片）

```
import tensorflow as tf
import numpy as np
import datetime
import matplotlib.pyplot as plt

# Read the dataset
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/")
```

導 入 物
件包

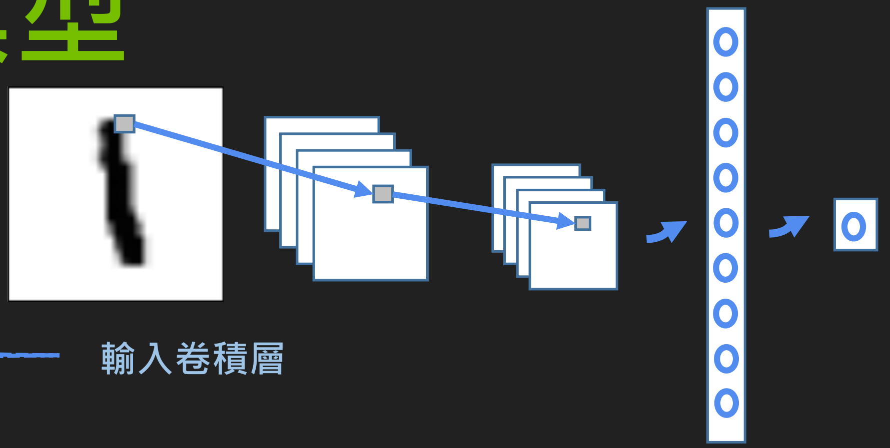
將 MNIST 載 入
到MNIST_data
目錄下



程式解析-判定模型

- 判定模型目的就是要分辨真假資料，在這個task上就是給一張圖片，然後輸出一個「相似度」的分數—越高表示這張圖片越像從真的dataset出來，反之則是由工匠偽造的。

程式解析-判定模型



```python
def discriminator(images, reuse_variables=None):
    with tf.variable_scope(tf.get_variable_scope(), reuse=reuse_variables) as scope:

        # First convolutional and pool layers
        # This finds 32 different 5 x 5 pixel features
        d_w1 = tf.get_variable('d_w1', [5, 5, 1, 32], initializer=tf.truncated_normal_initializer(stddev=0.02))
        d_b1 = tf.get_variable('d_b1', [32], initializer=tf.constant_initializer(0))
        d1 = tf.nn.conv2d(input=images, filter=d_w1, strides=[1, 1, 1, 1], padding='SAME')
        d1 = d1 + d_b1
        d1 = tf.nn.relu(d1)
        d1 = tf.nn.avg_pool(d1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

        # Second convolutional and pool layers
        # This finds 64 different 5 x 5 pixel features
        d_w2 = tf.get_variable('d_w2', [5, 5, 32, 64], initializer=tf.truncated_normal_initializer(stddev=0.02))
        d_b2 = tf.get_variable('d_b2', [64], initializer=tf.constant_initializer(0))
        d2 = tf.nn.conv2d(input=d1, filter=d_w2, strides=[1, 1, 1, 1], padding='SAME')
        d2 = d2 + d_b2
        d2 = tf.nn.relu(d2)
        d2 = tf.nn.avg_pool(d2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

        # First fully connected layer
        d_w3 = tf.get_variable('d_w3', [7 * 7 * 64, 1024], initializer=tf.truncated_normal_initializer(stddev=0.02))
        d_b3 = tf.get_variable('d_b3', [1024], initializer=tf.constant_initializer(0))
        d3 = tf.reshape(d2, [-1, 7 * 7 * 64])
        d3 = tf.matmul(d3, d_w3)
        d3 = d3 + d_b3
        d3 = tf.nn.relu(d3)

        # Second fully connected layer
        d_w4 = tf.get_variable('d_w4', [1024, 1], initializer=tf.truncated_normal_initializer(stddev=0.02))
        d_b4 = tf.get_variable('d_b4', [1], initializer=tf.constant_initializer(0))
        d4 = tf.matmul(d3, d_w4) + d_b4

        # d4 contains unscaled values
        return d4
```

輸入卷積層

卷積層

隱藏層

輸出層

程式解析-判定模型

```
d_w1 = tf.get_variable('d_w1', [5, 5, 1, 32], initializer=tf.truncated_normal_initializer(stddev=0.02))
d_b1 = tf.get_variable('d_b1', [32], initializer=tf.constant_initializer(0))
d1 = tf.nn.conv2d(input=images, filter=d_w1, strides=[1, 1, 1, 1], padding='SAME')
d1 = d1 + d_b1
d1 = tf.nn.relu(d1)
d1 = tf.nn.avg_pool(d1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
```

設 定 初 始 變 量 - filter 與 bias

建立2-D捲基層OP

建立神經元 f(wx+b)

```
d_w2 = tf.get_variable('d_w2', [5, 5, 32, 64], initializer=tf.truncated_normal_initializer(stddev=0.02))
d_b2 = tf.get_variable('d_b2', [64], initializer=tf.constant_initializer(0))
d2 = tf.nn.conv2d(input=d1, filter=d_w2, strides=[1, 1, 1, 1], padding='SAME')
d2 = d2 + d_b2
d2 = tf.nn.relu(d2)
d2 = tf.nn.avg_pool(d2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
```

設 定 初 始 變 量 - filter 與 bias

建立2-D捲基層OP

建立神經元 f(wx+b)

```
d_w3 = tf.get_variable('d_w3', [7 * 7 * 64, 1024], initializer=tf.truncated_normal_initializer(stddev=0.02))
d_b3 = tf.get_variable('d_b3', [1024], initializer=tf.constant_initializer(0))
d3 = tf.reshape(d2, [-1, 7 * 7 * 64])
d3 = tf.matmul(d3, d_w3)
d3 = d3 + d_b3
d3 = tf.nn.relu(d3)
```

設 定 初 始 變 量 - weight 與 bias

轉換為全連結層

建立神經元 f(wx+b)

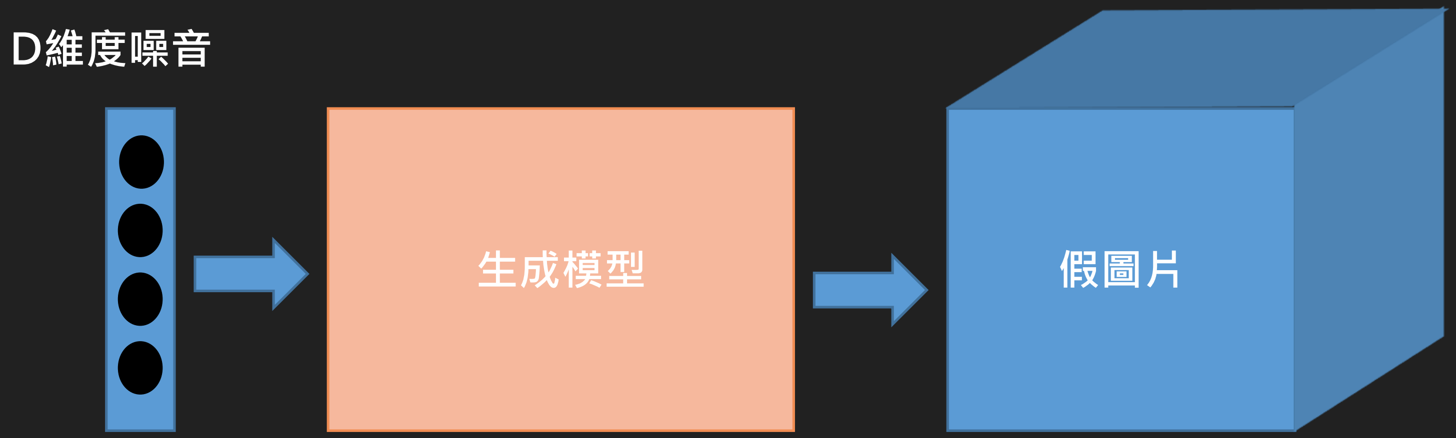
```
d_w4 = tf.get_variable('d_w4', [1024, 1], initializer=tf.truncated_normal_initializer(stddev=0.02))
d_b4 = tf.get_variable('d_b4', [1], initializer=tf.constant_initializer(0))
d4 = tf.matmul(d3, d_w4) + d_b4
```

設 定 初 始 變 量 - weight 與 bias

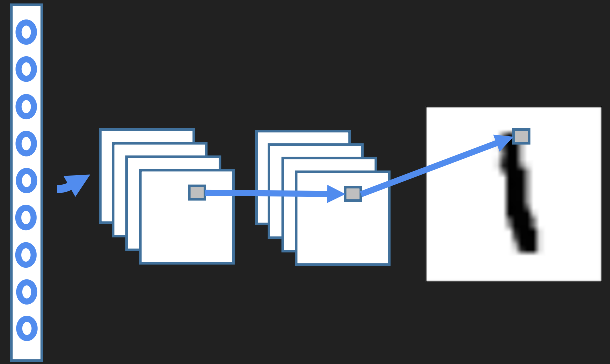
建立輸出神經元 f(wx+b)

程式解析-生成模型

- 生成模型的目的是要偽造圖片，因此輸出入跟偵探是相反的，若是訓練地非常完美，就可以不斷地輸出跟真實手寫數字相差無幾的圖片。



程式解析-生成模型



```python
def generator(z, batch_size, z_dim):
    with tf.variable_scope(tf.get_variable_scope(), reuse=reuse_variables) as scope:
        # From z_dim to 56*56 dimension
        g_w1 = tf.get_variable('g_w1', [z_dim, 3136], dtype=tf.float32, initializer=tf.truncated_normal_initializer(stddev=0.02))
        g_b1 = tf.get_variable('g_b1', [3136], initializer=tf.truncated_normal_initializer(stddev=0.02))
        g1 = tf.matmul(z, g_w1) + g_b1
        g1 = tf.reshape(g1, [-1, 56, 56, 1])
        g1 = tf.contrib.layers.batch_norm(g1, epsilon=1e-5, scope='bn1')
        g1 = tf.nn.relu(g1)

        # Generate 50 features
        g_w2 = tf.get_variable('g_w2', [3, 3, 1, z_dim/2], dtype=tf.float32, initializer=tf.truncated_normal_initializer(stddev=0.02))
        g_b2 = tf.get_variable('g_b2', [z_dim/2], initializer=tf.truncated_normal_initializer(stddev=0.02))
        g2 = tf.nn.conv2d(g1, g_w2, strides=[1, 2, 2, 1], padding='SAME')
        g2 = g2 + g_b2
        g2 = tf.contrib.layers.batch_norm(g2, epsilon=1e-5, scope='bn2')
        g2 = tf.nn.relu(g2)
        g2 = tf.image.resize_images(g2, [56, 56])

        # Generate 25 features
        g_w3 = tf.get_variable('g_w3', [3, 3, z_dim/2, z_dim/4], dtype=tf.float32, initializer=tf.truncated_normal_initializer(stddev=0.02))
        g_b3 = tf.get_variable('g_b3', [z_dim/4], initializer=tf.truncated_normal_initializer(stddev=0.02))
        g3 = tf.nn.conv2d(g2, g_w3, strides=[1, 2, 2, 1], padding='SAME')
        g3 = g3 + g_b3
        g3 = tf.contrib.layers.batch_norm(g3, epsilon=1e-5, scope='bn3')
        g3 = tf.nn.relu(g3)
        g3 = tf.image.resize_images(g3, [56, 56])

        # Final convolution with one output channel
        g_w4 = tf.get_variable('g_w4', [1, 1, z_dim/4, 1], dtype=tf.float32, initializer=tf.truncated_normal_initializer(stddev=0.02))
        g_b4 = tf.get_variable('g_b4', [1], initializer=tf.truncated_normal_initializer(stddev=0.02))
        g4 = tf.nn.conv2d(g3, g_w4, strides=[1, 2, 2, 1], padding='SAME')
        g4 = g4 + g_b4
        g4 = tf.sigmoid(g4)

        # Dimensions of g4: batch_size x 28 x 28 x 1
        return g4
```

輸入層

卷積層

卷積層

輸出層

程式解析-生成模型

```
g_w1 = tf.get_variable('g_w1', [z_dim, 3136], dtype=tf.float32, initializer=tf.truncated_normal_initializer(stddev=0.02))
g_b1 = tf.get_variable('g_b1', [3136], initializer=tf.truncated_normal_initializer(stddev=0.02))
g1 = tf.matmul(z, g_w1) + g_b1
g1 = tf.reshape(g1, [-1, 56, 56, 1])
g1 = tf.contrib.layers.batch_norm(g1, epsilon=1e-5, scope='bn1')
g1 = tf.nn.relu(g1)
```

設定初始變量-weight 與 bias

建立神經元 f(wx+b)

將張量重組成圖片大小 56*56*1

每一層做Batch Normalization

```
g_w2 = tf.get_variable('g_w2', [3, 3, 1, z_dim/2], dtype=tf.float32, initializer=tf.truncated_normal_initializer(stddev=0.02))
g_b2 = tf.get_variable('g_b2', [z_dim/2], initializer=tf.truncated_normal_initializer(stddev=0.02))
g2 = tf.nn.conv2d(g1, g_w2, strides=[1, 2, 2, 1], padding='SAME')
g2 = g2 + g_b2
g2 = tf.contrib.layers.batch_norm(g2, epsilon=1e-5, scope='bn2')
g2 = tf.nn.relu(g2)
g2 = tf.image.resize_images(g2, [56, 56])
```

設定初始變量-filter 與 bias

建立2-D捲基層OP

建立神經元 f(wx+b)

將張量重組成圖片大小 56*56

```
g_w3 = tf.get_variable('g_w3', [3, 3, z_dim/2, z_dim/4], dtype=tf.float32, initializer=tf.truncated_normal_initializer(stddev=0.02))
g_b3 = tf.get_variable('g_b3', [z_dim/4], initializer=tf.truncated_normal_initializer(stddev=0.02))
g3 = tf.nn.conv2d(g2, g_w3, strides=[1, 2, 2, 1], padding='SAME')
g3 = g3 + g_b3
g3 = tf.contrib.layers.batch_norm(g3, epsilon=1e-5, scope='bn3')
g3 = tf.nn.relu(g3)
g3 = tf.image.resize_images(g3, [56, 56])
```

設定初始變量-filter 與 bias

建立2-D捲基層OP

建立神經元 f(wx+b)

將張量重組成圖片大小 56*56

```
g_w4 = tf.get_variable('g_w4', [1, 1, z_dim/4, 1], dtype=tf.float32, initializer=tf.truncated_normal_initializer(stddev=0.02))
g_b4 = tf.get_variable('g_b4', [1], initializer=tf.truncated_normal_initializer(stddev=0.02))
g4 = tf.nn.conv2d(g3, g_w4, strides=[1, 2, 2, 1], padding='SAME')
g4 = g4 + g_b4
g4 = tf.sigmoid(g4)
```

設定初始變量-filter 與 bias

建立神經元 f(wx+b)， sigmoid 將輸出落在 0 與 1 之間， 即將圖片視為黑與白

程式解析

- Gz = 生成假圖、Dx = 真圖的分數、Dg = 假圖的分數

```
# Define the plceholder and the graph
z_dimensions = 100

# For generator, one image for a batch
z_batch = np.random.normal(0, 1, [1, z_dimensions])

tf.reset_default_graph()
batch_size = 50

z_placeholder = tf.placeholder(tf.float32, [None, z_dimensions], name='z_placeholder')
# z_placeholder is for feeding input noise to the generator

x_placeholder = tf.placeholder(tf.float32, shape = [None,28,28,1], name='x_placeholder')
# x_placeholder is for feeding input images to the discriminator

Gz = generator(z_placeholder, batch_size, z_dimensions)
# Gz holds the generated images

Dx = discriminator(x_placeholder)
# Dx will hold discriminator prediction probabilities
# for the real MNIST images

Dg = discriminator(Gz, reuse_variables=True)
# Dg will hold discriminator prediction probabilities for generated images
```

定義維度

產生常態分佈亂數

重新定義圖形

定義每次訓練多少樣本

設置輸入噪音佔位符

設置輸入真實圖片佔位符

透過 generate function 得到假圖

將真實圖片進入訓練

將假圖片進入分辨

程式解析-損失

- Dx是真圖分數，Dg是假圖分數
- 分別對這兩個分數與真值(1)和假值(0)取cross_entropy

```
Let x = logits, z = labels, the result is:

z * -log(sigmoid(x)) + (1 - z) * -log(1 - sigmoid(x))
```

```
# Two Loss Functions for discriminator
d_loss_real = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits = Dx, labels = tf.ones_like(Dx)))
d_loss_fake = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits = Dg, labels = tf.zeros_like(Dg)))

# Loss function for generator
g_loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits = Dg, labels = tf.ones_like(Dg)))
```

程式解析-損失

```
# Two Loss Functions for discriminator
d_loss_real = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits = Dx, labels = tf.ones_like(Dx)))
d_loss_fake = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits = Dg, labels = tf.zeros_like(Dg)))

# Loss function for generator
g_loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits = Dg, labels = tf.ones_like(Dg)))

# Get the varaibles for different network
tvars = tf.trainable_variables()

d_vars = [var for var in tvars if 'd_' in var.name]
g_vars = [var for var in tvars if 'g_' in var.name]

print([v.name for v in d_vars])
print([v.name for v in g_vars])


# Train the discriminator
d_trainer_fake = tf.train.AdamOptimizer(0.0003).minimize(d_loss_fake, var_list=d_vars)
d_trainer_real = tf.train.AdamOptimizer(0.0003).minimize(d_loss_real, var_list=d_vars)

# Train the generator
g_trainer = tf.train.AdamOptimizer(0.0001).minimize(g_loss, var_list=g_vars)
```

Dx：最大化 (1)
Dg：最小化(0)

讓假圖分數和1(tf.ones_like) 差別最小化，提高判別對假圖的分數

返回的是需要訓練的變量列表

取得不同變量的名稱

Adam optimization優化器，將損失最小化

程式解析-訓練

```python
saver = tf.train.Saver()

sess = tf.Session()
sess.run(tf.global_variables_initializer())
# Train generator and discriminator together
for i in range(100000):
    real_image_batch = mnist.train.next_batch(batch_size)[0].reshape([batch_size, 28, 28, 1])
    z_batch = np.random.normal(0, 1, size=[batch_size, z_dimensions])

    # Train discriminator on both real and fake images
    _, __, dLossReal, dLossFake = sess.run([d_trainer_real, d_trainer_fake, d_loss_real, d_loss_fake],
                                           {x_placeholder: real_image_batch, z_placeholder: z_batch})

    # Train generator
    z_batch = np.random.normal(0, 1, size=[batch_size, z_dimensions])
    _ = sess.run(g_trainer, feed_dict={z_placeholder: z_batch})

    # if i % 10 == 0:
    #     # Update TensorBoard with summary statistics
    #     z_batch = np.random.normal(0, 1, size=[batch_size, z_dimensions])
    #     summary = sess.run(merged, {z_placeholder: z_batch, x_placeholder: real_image_batch})
    #     writer.add_summary(summary, i)

    if i % 1000 == 0:
        # Save the model every 1000 iteration
        save_path = saver.save(sess, "./tmp/model{}.ckpt".format(i))
        print("Model saved in file: %s" % save_path)

    if i % 100 == 0:
        # Every 100 iterations, show a generated image
        print("Iteration:", i, "at", datetime.datetime.now())
        z_batch = np.random.normal(0, 1, size=[1, z_dimensions])
        generated_images = generator(z_placeholder, 1, z_dimensions)
        images = sess.run(generated_images, {z_placeholder: z_batch})
        plt.imshow(images[0].reshape([28, 28]), cmap='Greys')
        plt.savefig("./img/image{}.png".format(i))

        # Show discriminator's estimate
        im = images[0].reshape([1, 28, 28, 1])
        result = discriminator(x_placeholder)
        estimate = sess.run(result, {x_placeholder: im})
        print("Estimate:", estimate)
```

提供了變量、模型(也稱圖Graph)的保存和恢復模型

建立初始化變數 Op

取 batch_size 數量的圖片，並將重組為28*28*1

定義常態亂數 z

開始訓練判別器，並返回損失值

將常態分佈亂數餵入生成器，並開始訓練生成器與返回其損失數值

每迭代1000次將模型儲存在 tmp 資料夾底下

每迭代100次，將生成器生成的圖片存到 img 資料夾底下

將生成圖片使用判別器判別，並返回其判別數值

Jupyter開啟程式

- 在 cmd 視窗，輸入以下指令

```
> activate tensorflow
```

```
C:\>activate tensorflow

(tensorflow) C:\>
```

Jupyter開啟程式

- 在 cmd 視窗，輸入以下指令

```
> jupyter notebook
```

```
C:\>activate tensorflow

(tensorflow) C:\>jupyter notebook
```

Jupyter開啟程式

- 將程式式碼貼入並執行。

In [1]:

```python
import tensorflow as tf
import numpy as np
import datetime
import matplotlib.pyplot as plt

# Read the dataset
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/")
```

```
WARNING:tensorflow:From <ipython-input-1-26916c86d9aa>:8: read_data_sets (from tensorflow.contrib.learn.python.learn.datasets.mnist) is d
eprecated and will be removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
WARNING:tensorflow:From /home/user/anaconda3/envs/tensorflow/lib/python2.7/site-packages/tensorflow/contrib/learn/python/learn/datasets/m
nist.py:260: maybe_download (from tensorflow.contrib.learn.python.learn.datasets.base) is deprecated and will be removed in a future vers
ion.
Instructions for updating:
Please write your own downloading logic.
WARNING:tensorflow:From /home/user/anaconda3/envs/tensorflow/lib/python2.7/site-packages/tensorflow/contrib/learn/python/learn/datasets/m
nist.py:262: extract_images (from tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a future ver
sion.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data/train-images-idx3-ubyte.gz
WARNING:tensorflow:From /home/user/anaconda3/envs/tensorflow/lib/python2.7/site-packages/tensorflow/contrib/learn/python/learn/datasets/m
nist.py:267: extract_labels (from tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a future ver
sion.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From /home/user/anaconda3/envs/tensorflow/lib/python2.7/site-packages/tensorflow/contrib/learn/python/learn/datasets/m
```

In [2]:

```python
def discriminator(images, reuse_variables=tf.AUTO_REUSE):
    with tf.variable_scope(tf.get_variable_scope(), reuse=reuse_variables) as scope:

        # First convolutional and pool layers
        # This finds 32 different 5 x 5 pixel features
        d_w1 = tf.get_variable('d_w1', [5, 5, 1, 32], initializer=tf.truncated_normal_initializer(stdde
        d_b1 = tf.get_variable('d_b1', [32], initializer=tf.constant_initializer(0))
        d1 = tf.nn.conv2d(input=images, filter=d_w1, strides=[1, 1, 1, 1], padding='SAME')
        d1 = d1 + d_b1
        d1 = tf.nn.relu(d1)
        d1 = tf.nn.avg_pool(d1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

        # Second convolutional and pool layers
        # This finds 64 different 5 x 5 pixel features
        d_w2 = tf.get_variable('d_w2', [5, 5, 32, 64], initializer=tf.truncated_normal_initializer(stdc
        d_b2 = tf.get_variable('d_b2', [64], initializer=tf.constant_initializer(0))
```

Reference

- 思源，GAN完整理論推導與實現，機器之心[Link]
- Jonbrouner,，Generative Adversarial Networks for Beginners，GitHub[Link]
- Hindupuravinash，The GAN Zoo，GitHub[Link]

-END-