

AI principle HW3

Code https://github.com/Benchangatrul284/ML-practice/tree/main/AI_principle_HW3

第七組 110511068 蔡雅婷 110511164 張語楹 109261007 李紹穎

Part 1: GAN

GAN 全名為 Generative Adversarial Network 生成式對抗神經網路，具有兩個模型，其中一個是負責學習資料分布的 generator，另一個是來分辨真偽的 discriminator。

我們的任務是讓 generator 能夠在 discriminator 的監督下產生更好的花朵圖片，讓 discriminator 在一次次更新中努力不要被 generator 混淆，所以 GAN 具有 minimax game 的特性。

在程式中我們可以先透過產生 normal 雜訊輸入到 generator 產生圖片，再將產生的圖片和真實圖片都放進 discriminator 中辨別真偽，discriminator 最後一層是 dense(1)再過 tanh 讓數值在[-1, 1]間，接著在 crossentropy 終將 from_logits = True，這樣可以讓他自動將分布轉成機率分布的 sigmoid，讓數值在[0,1]間，所以 discriminator 最後是會產生數值而已。

```
model.add(Dense(1))
model.add(Activation('tanh'))
```

```
cross_entropy =
tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

接著是最重要的 loss 計算和 gradient 計算，在 GAN 中我們使用的方法是 BinaryCrossEntropy，其實也很好理解，discriminator 在輸入真圖片時會產生一個真結果，這些真結果應該要很靠近 1，並 loss 計算時將真結果和全都是 1 進行 crossentropy，而假圖片輸出形成假結果時，應該要和 0 進行 crossentropy，generator 接著會將假結果和 1 進行 crossentropy，因為 generator 會希望假結果非常靠近 1，兩者接著再做 gradient descent。

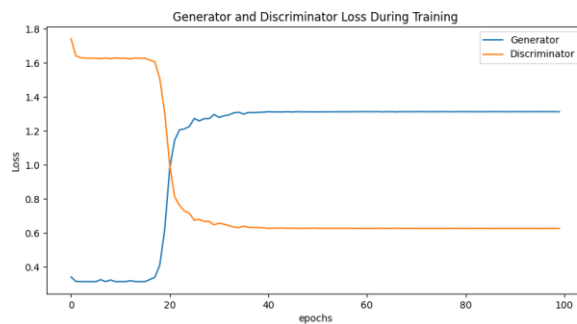
```
def generator_loss(fake_output):
    """
    The generator wants the discriminator to label the generated
    samples as valid (ones)
    """
    return cross_entropy(tf.ones_like(fake_output), fake_output)

def discriminator_loss(real_output, fake_output):
    """
    The discriminator wants to classify real samples as valid and fake
    samples as invalid (zeros)
    """
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
```

with laten_dim = 10, epochs = 100

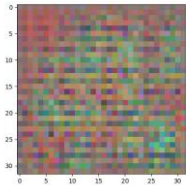
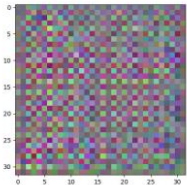
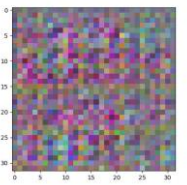
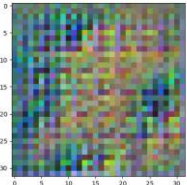
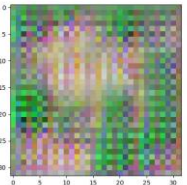
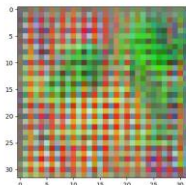
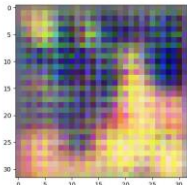
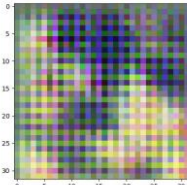
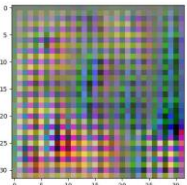
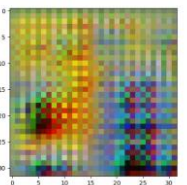
最後嘗試出最好的結果: batch_size = 16, lr = 1e-4, time for 1 epoch = 40sec

loss history of training:



可以看到在 20 epochs 時具有交叉點，在接近 40 epochs 時，兩者的 loss 飽和

images generated every 10 epochs

epoch 10	epoch 20	epoch 30	epoch 40	epoch 50
				
epoch 60	epoch 70	epoch 80	epoch 90	epoch 100
				

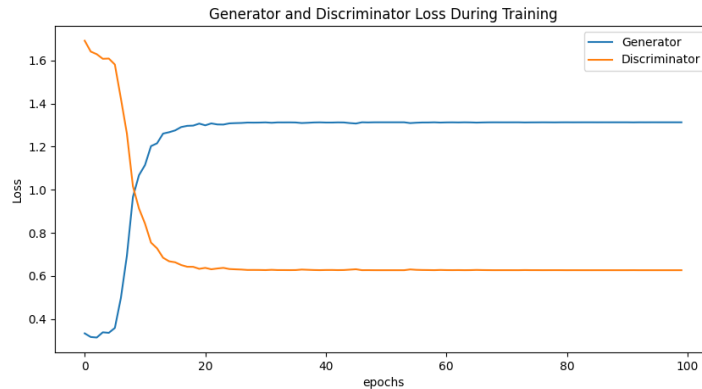
大概 50 epochs 後才有辦法看出一些花朵的特色，因為 latent_dim 的大小只有 10，變化量很小所以圖片中的顏色才會趨向於同色塊格子狀的感覺。

在訓練時有嘗試很多不同的 hyperparameter 和 model，發現在顯存足夠的情況下使用較大的 batch 訓練時間會快速很多，但 epoch 數量固定 100，GAN 的模型需要較多的更新，後來發現將 batch 調整至 16 較適合。

with laten_dim = 256, epochs = 100

最後嘗試出最好的結果: batch_size = 16, lr = 1e-4, time for 1 epoch = 40sec

loss history of training:



images generated every 10 epochs

epoch 10	epoch 20	epoch 30	epoch 40	epoch 50
epoch 60	epoch 70	epoch 80	epoch 90	epoch 100

在 60 epochs 比較知道自己真的在 train 花朵而不適普通的色塊 XD。

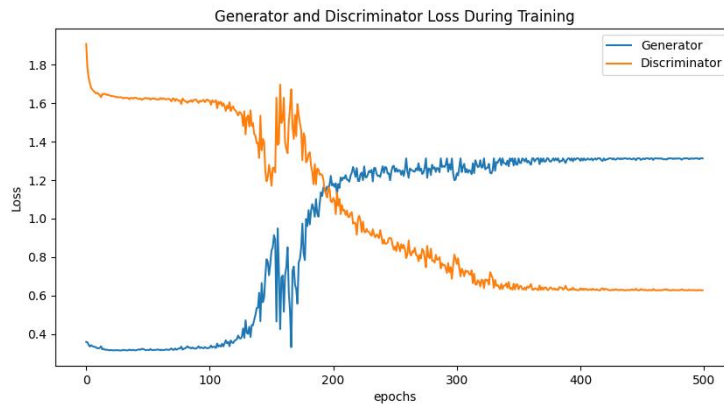
相較於 latent_dim = 10，latent_dim = 256 具有以下特色:

1. loss 較快飽和(覺得可能的原因是，具有較多的變數輸入，會使模型看到比較複雜的變量，從而加快 loss 更新)
2. 產生的圖片的色彩較混合，楊淇向 epoch 80 一樣，比較不會看出是一塊塊的色塊拼湊出來的

picture size = (128,3128, 3)

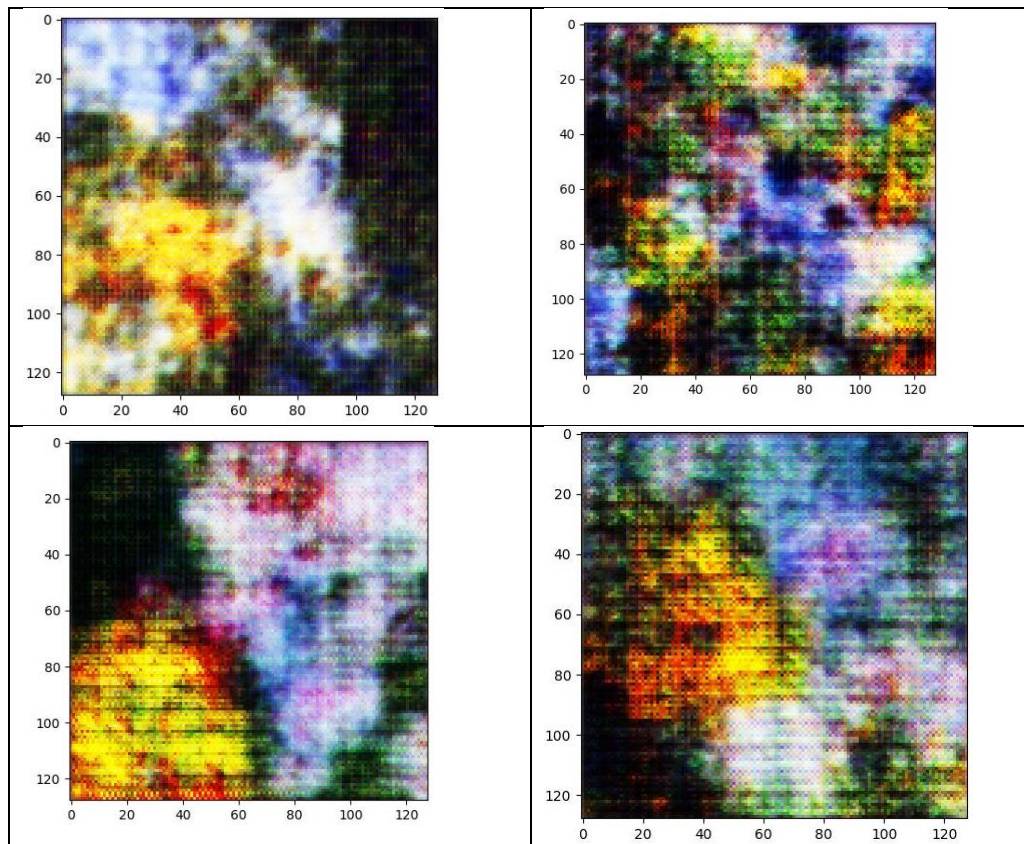
with latent = 1024, batch size = 64, epochs = 500, lr = 1e-5

Generator and Discriminator Loss During Training



從 loss 的結果圖中可以發現大概在 250 epochs 時，generator 的 loss 就沒有再傷生趨近於飽和，在 300epochs 時，discriminator 的 loss 沒有再下降，大概在 200 epochs 時 generator 和 discriminator 來到了交叉點，而從所有產生的結果可以看出，model 大概在 250 左右以後產生的圖像內容都差不多，如下。

4 張結果圖



可以從結果圖看出，generator 學會圖片中需要有黃色和粉色擠在一起(花)、綠色(葉子)包圍黃色和粉色、綠色在旁邊的是類似藍天的白色和藍色，很驚訝他真的可以學到一定的分布。

Part 2: VAE

Goal:

$$g_{\phi}(z|x) = P(x|z)$$

因此我們用 KL divergence 來算兩個分布的距離，一番推導後：

$$D_{KL}(g_{\phi}(z|x)||P(z|x)) = \log P(x) + D_{KL}(g_{\phi}(z|x)||P(z)) - E_{z \sim g_{\phi}(z|x)} \log f_{\theta}(x|z)$$

where $D_{KL}(P||Q) = \int p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$

整理一下上面的式子：

$$E_{z \sim g_{\phi}(z|x)} \log f_{\theta}(x|z) - D_{KL}(g_{\phi}(z|x)||P(z)) = \log P(x) - D_{KL}(g_{\phi}(z|x)||P(z|x))$$

$\log P(x)$ 是我們的 log likelihood 是我們最終要 maximize 的值，而

$-D_{KL}(g_{\phi}(z|x)||P(z|x))$ 是我們 reconstruction 後 $g_{\phi}(z|x)$ 和 prior $P(z|x)$ 的距離。

我們的 loss function 為：

$$\begin{aligned} -\mathcal{L}_{\theta, \phi}(x) \text{ (ELBO)} &= -\log P(x) + D_{KL}(g_{\phi}(z|x)||P(z|x)) \\ &= -E_{z \sim g_{\phi}(z|x)} \log f_{\theta}(x|z) + D_{KL}(g_{\phi}(z|x)||P(z)) \end{aligned}$$




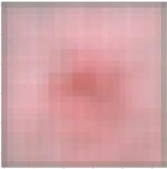
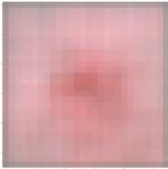

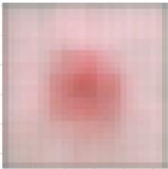


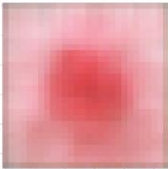
ELBO 的意思為 evidence lower bound 因為他是 $\log P(x)$ 的最小值。

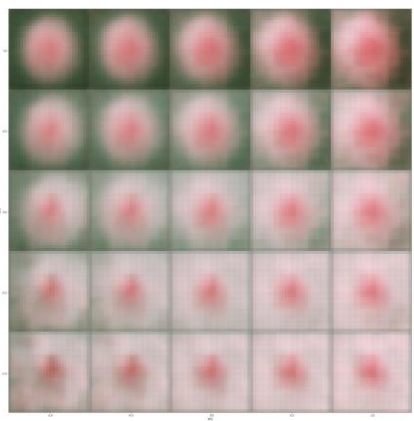
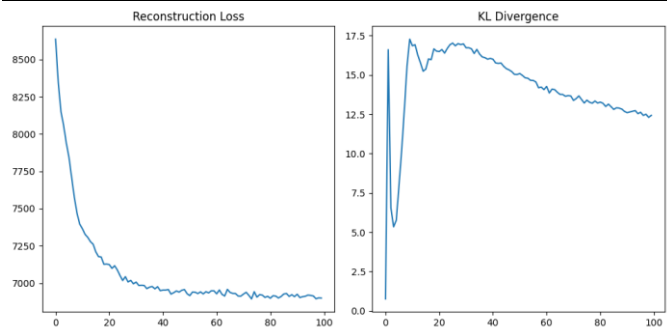
$$\log P(x) \geq \log P(x) - D_{KL}(g_{\phi}(z|x)||P(z|x)) = \mathcal{L}_{\theta, \phi}(x) \text{ (ELBO)}$$

在我們程式當中， $\log f_{\theta}(x|z)$ 稱為 reconstruction loss，而 $D_{KL}(g_{\phi}(z|x)||P(z))$ 為 KL loss。

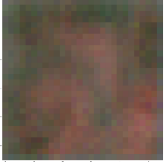









Generating roses:

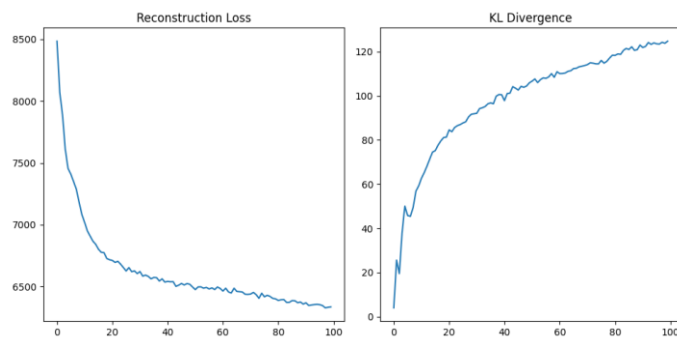
Latent dimension = 3, epochs = 100

epoch 10	epoch 20	epoch 30	epoch 40	epoch 50
				
epoch 60	epoch 70	epoch 80	epoch 90	epoch 100
				



Latent dimension = 256, epochs = 100

epoch 10	epoch 20	epoch 30	epoch 40	epoch 50
				
epoch 60	epoch 70	epoch 80	epoch 90	epoch 100
				



可以看到當 latent space 增加後 KL divergence 增加。

這是因為 $D_{KL}(p_{\theta}(z)||q_{\phi}(z|x))$ ，我們假設 $p_{\theta}(z)$ 的 prior 為 Gaussian，但我們隨著 latent space 的 dimension 增加， $q_{\phi}(z|x)$ 的自由度越高，越難被壓到和 prior 相同的 distribution，然而，reconstruction loss 比較低。因此，可以把 KL divergence 看作是一個 regularization 的 term。

但是我們很難得到 $g_{\phi}(z|x)$ 因為這是要經過 **sample** 的過程，因此我們必須要 **reparameterization trick** 才能做 **back propagation**。方法也很簡單，假設一個 $\epsilon \sim p(\epsilon) = N(0, I)$ ，我們從 **standard Gaussian** 取樣。而我們 **encoder** 就僅產出該 **Gaussian** 的平均值和 **variance**。接下來送進去 **decoder** 的方法為：

$$z = \mu(x) + \epsilon * \sigma(x)$$

也就是說，我們的 **latent variable** 其實是我們的 **prior** $N(0, I)$ 的 **mixture**。

這種方法就是將 **sample** 的過程移到 $\epsilon \sim N(0, I)$ 而非直接從 $g_{\phi}(z|x)$ 取樣。

接下來來討論一下 **back propagation** 時如何計算 **gradient**

首先，對於兩個 **Gaussian distribution** 的 **random variable** 的 **KL divergence** 為：

$$D_{KL}(q_{\phi}(z|x) || P(z)) = -\frac{1}{2} \sum_{j=1}^J (1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2)$$

J 是 **latent space** 的 **dimension**。 $P(z) \sim N(0, I)$ 。 $q_{\phi}(z|x) \sim N(\mu, K)$

為了有解析解，我們讓 **K** 為對角矩陣。

因此對 μ_j 微分，

$$\frac{\partial L}{\partial \mu_j} = \frac{\partial}{\partial \mu_j} \left(-\frac{1}{2} \sum_{j=1}^J (-\mu_j^2) \right) = \mu_j$$

以向量來表示：

$$\frac{\partial L}{\partial \mu} = \mu$$

對 σ_j 微分：

$$\frac{\partial D_{KL}}{\partial \sigma_j} = -\frac{1}{2} \left(\frac{2}{\sigma_j} - 2\sigma_j \right) = \sigma_j - \frac{1}{\sigma_j}$$

以向量來表示：

$$\frac{\partial L}{\partial \sigma} = \sigma - \frac{I}{\sigma}$$

可以看到當 μ_j 遠離 0 的時候，**gradient** 會增加，讓 μ_j 接近 0。

當 σ_j 遠大於 1 或小於 1 時， σ_j 的 **gradient** 會增加，讓 σ_j 接近 1。這樣 **loss function** 的設計就是為了讓 $q_{\phi}(z|x)$ 和 **prior** $P(z)$ 接近。

Loss function 在實作方面會避免 **KL divergence** 裡面的積分而略有不同。

也就是說，

$D_{KL}(P||Q) = \int p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$ 會變成 $\log \left(\frac{p(x)}{q(x)} \right)$ 。

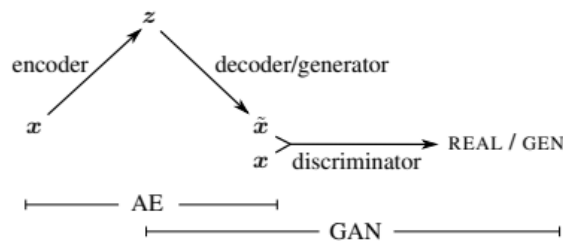
```
def compute_loss(model, x):  
    '''  
    Compute the loss of the model.  
    return the total loss, reconstruction loss, KL loss  
    '''  
    mean, logvar = model.encode(x)  
    z = model.reparameterize(mean, logvar)  
    x_decoded = model.decode(z)  
    cross_ent = tf.keras.losses.binary_crossentropy(x, x_decoded)  
    logfx_z = -tf.reduce_sum(cross_ent)  
    loggz = log_normal_pdf(z, 0., 0.)  
    logfz_x = log_normal_pdf(z, mean, logvar)  
    return -tf.reduce_mean(logfx_z + loggz - logfz_x), -tf.reduce_mean(logfx_z), -  
    tf.reduce_mean(loggz - logfz_x)
```

可以看到我們的 reconstruction loss 是一個 binary entropy。這是因為我們將圖像取為[0,1]。而我們的 KL divergence 則變為：

$$-(\log g_{\phi}(z|x) - \log P(x)) = \log P(x) - \log g_{\phi}(z|x)$$

如程式所示。

Part 3: Combination



VAE 的優點就是他比較容易 converge，這是因為我們對他的 constraint 比較大，但相對的就比較難產生新的圖片。GAN 的 constraint 比較小，但較難 converge。

因此如何把兩者結合是一個關鍵的議題。

我們在網路上找到了[這篇論文](#)，裡面提供了如何將 VAE 和 GAN 結合在一起的方法。其實方法很簡單也很直觀。首先，GAN 的 generator 的 input 為何只能是 Gaussian noise? 在 VAE 當中，我們有很好的 encoder，encoder 就可以算是特徵提取的 function，所以我們能夠先用 encoder 來做 feature extraction。因此我們 generator 可以用一個較有 information 的 latent vector 來產生圖片。這個和原始 GAN 不同的是，GAN 採用雜訊來 decode 出真正的圖片。而 VAE 的 decoder 就當作是 GAN 中的 generator。

此外，可以看到 Loss function 為：

$$\mathcal{L} = \mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{llike}}^{\text{Disl}} + \mathcal{L}_{\text{GAN}} .$$

所以這個 loss function 就是 VAE 的 reconstruction error+ KL divergence+GAN 的 generator 的 loss 。

Reference:

<https://arxiv.org/pdf/1512.09300.pdf>