前言

2022 年新冠疫情肆虐，AI 領域在這一年出現了重大的進展，AI 已經能生成遠超乎我們想像的影像。知名的 AI 服務有 Stable Diffusion、Midjourney、DALL·E 等。這種影像生成式 AI 在各個領域都受到矚目，並運用在各式各樣的用途。值得注意的是，這些 AI 的背後都使用了深度學習的「生成模型」技術。因此，2022 年也可說是以生成模型開發的技術開花結果的一年。

本書的主題是生成模型。所謂的生成模型，就是一種產生新資料的技術。本書以生成模型為主題，廣泛涵蓋傳統與最先進的技術。本書從常態分布、最大概似估計等基本內容開始說明，接著學習高斯混合模型、EM 演算法，之後介紹使用了深度學習的手法。具體而言，我們將依序建立變分自編碼器（VAE）、階層型 VAE、擴散模型，學習其理論與實作。

本書最後完成的擴散模型因性能優異，為生成 AI 領域帶來了革命性的變化。本書以擴散模型為終極目標，將其過程分解成「10 個步驟」來說明。這 10 個步驟是有連貫性的故事，你可以依照每個步驟學習與生成模型有關的重要技術。

趣味藏在細節裡

本書將鉅細靡遺地解析生成模型的機制。除了傳授概念與結果之外，還將詳細說明「為何如此」、「如何得到這樣的結果」。因此，必須仔細了解公式，同時注意細微的環節。深入學習技術的細節是徹底掌握技術的重要關鍵，而技術的有趣之處就藏在這些細節之中。

此外，如圖 3-4 所示，計算矩陣乘積時，「形狀」是很重要的關鍵。

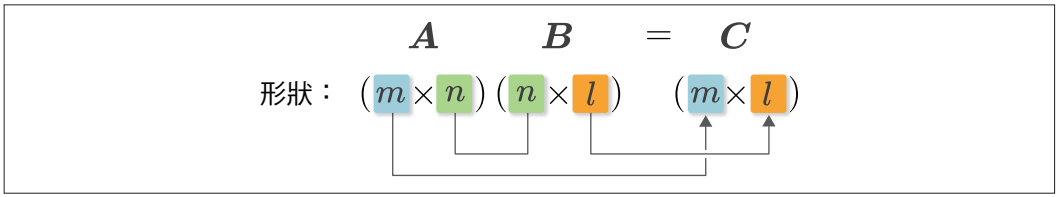

圖 3-4 矩陣乘積必須讓對應維度的元素數（此範例是指 n）一致

在圖 3-4 的範例中，利用 $m \times n$ 矩陣 $\boldsymbol{A}$ 與 $n \times l$ 矩陣 $\boldsymbol{B}$ 的乘積計算 $m \times l$ 矩陣 $\boldsymbol{C}$。此時，如圖所示，矩陣 $\boldsymbol{A}$ 與 $\boldsymbol{B}$ 對應維度的元素數（n）必須一致。矩陣 $\boldsymbol{C}$ 的形狀是由矩陣 $\boldsymbol{A}$ 的列數與矩陣 $\boldsymbol{B}$ 的行數構成。

3.2 多維常態分布

到目前為止，我們看到的常態分布都是以單一實數值（純量）的分布為對象。例如，用常態分布表示「身高」的分布，但是觀測的對象只有「身高」。接下來要思考與多個實數值「向量」有關的常態分布。例如，把「身高」和「體重」當作一個向量，如下所示。

$$\boldsymbol{x} = \begin{pmatrix} \text{身高} \\ \text{體重} \end{pmatrix}$$

如上所示，將資料整合成向量，例如，A 是（175cm, 50kg），B 是（160cm, 53kg），就可以得到每個人的測量資料。而這裡的主題是，以常態分布表示由向量組成的資料分布（圖 3-5）。

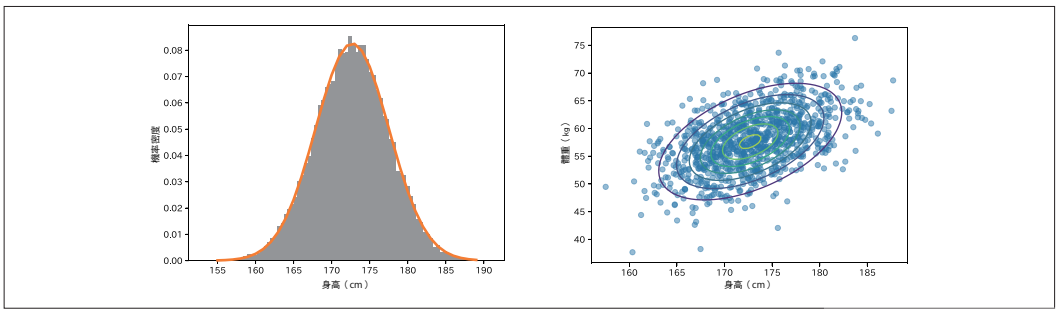

圖 3-5 一維常態分布（左圖）與二維常態分布（右圖）的範例

3.3 將二維常態分布視覺化

這裡要繪製二維常態分布的圖表。首先要介紹使用 Matplotlib 繪製 3D 圖的方法。

3.3.1 3D 圖繪製方法

使用 `plot_surface()` 函數能用 Matplotlib 繪製 3D 圖。以下是這個函數的使用範例。

step03/plot_3d.py

```
import numpy as np
import matplotlib.pyplot as plt

X = np.array([[-2, -1, 0, 1, 2],
              [-2, -1, 0, 1, 2],
              [-2, -1, 0, 1, 2],
              [-2, -1, 0, 1, 2],
              [-2, -1, 0, 1, 2]])
Y = np.array([[-2, -2, -2, -2, -2],
              [-1, -1, -1, -1, -1],
              [0, 0, 0, 0, 0],
              [1, 1, 1, 1, 1],
              [2, 2, 2, 2, 2]])
Z = X ** 2 + Y ** 2

ax = plt.axes(projection='3d') # 利用 projection='3d' 設定 3d 圖
ax.plot_surface(X, Y, Z, cmap='viridis') # 使用稱作 viridis 的顏色圖
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.show()
```

這裡請注意 `ax.plot_surface(X, Y, Z, cmap='viridis')` 的引數。引數 `X`、`Y`、`Z` 都是形狀為 `(5, 5)` 的二維陣列。上面程式碼中的 `X` 與 `Y` 對應**圖 3-9** 的格狀圓點。

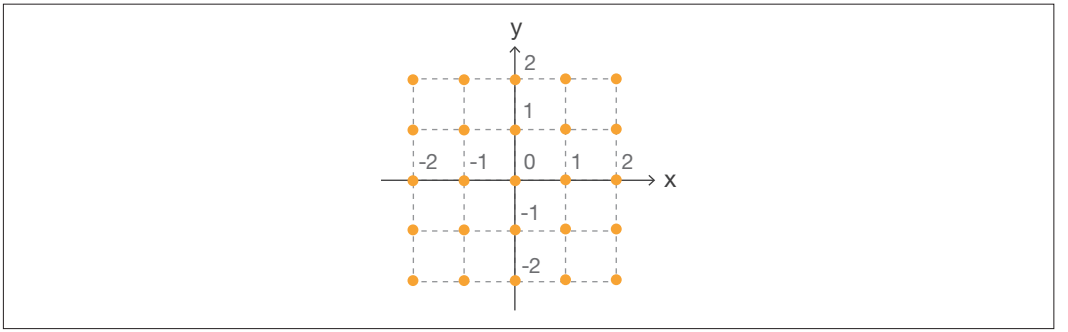

圖 3-9　對應 x 與 y 的格點

接著在 Z 設定每個格點的第三個維度值。例如，X[0, 0] 為 -2，Y[0, 0] 為 -2，因此 Z[0, 0] 設定為 $x=-2$、$y=-2$ 時的 z 值。這次是利用 Z = X**2 + Y**2 進行「逐元素運算」，因此一次就能計算出每個格點的值。

接著執行上面的程式碼，結果得到圖 3-10。

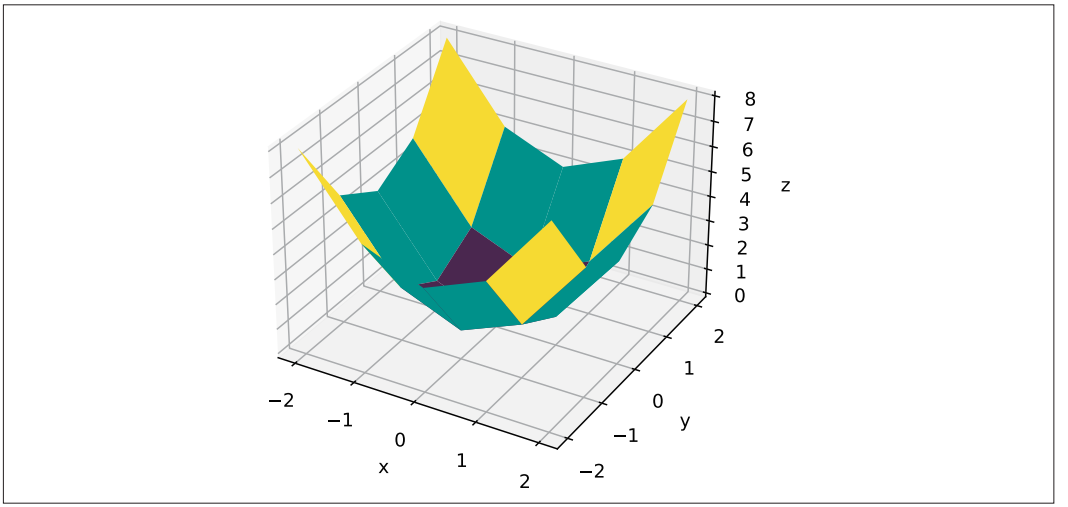

圖 3-10　用 Matplotlib 描繪的 3D 圖

接著以較密的格點繪製，可以得到如圖 3-11 的平滑圖表。

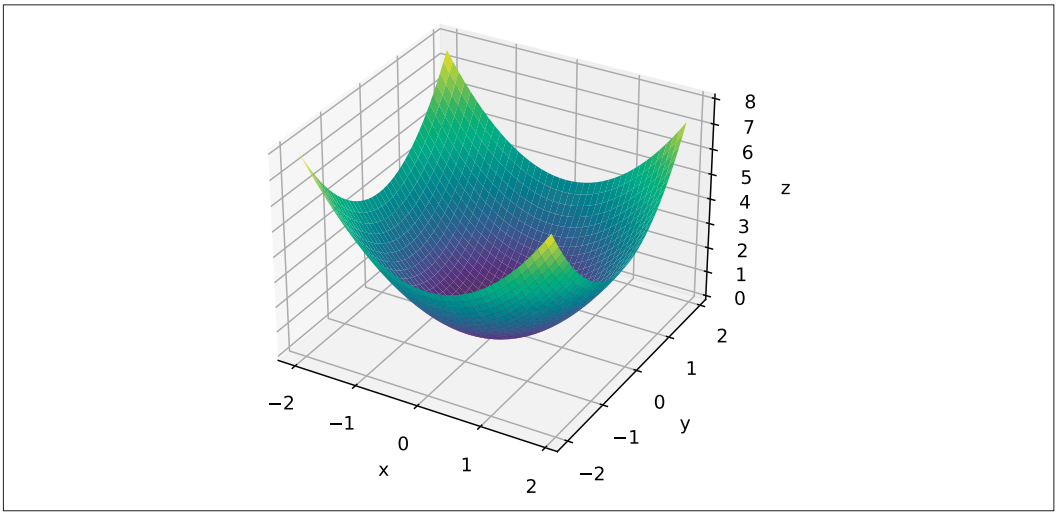

圖 3-11　更細緻的 3D 圖表

3.3.2　繪製等高線

plot_surface() 可以描繪函數的曲面（Surface），另外還有一種表現方法是「等高線」。使用 contour() 函數能繪製等高線，程式碼如下所示。

step03/plot_3d.py

```
x = np.arange(-2, 2, 0.1)
y = np.arange(-2, 2, 0.1)

X, Y = np.meshgrid(x, y)
Z = X ** 2 + Y ** 2

ax = plt.axes()
ax.contour(X, Y, Z)
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()
```

4.2.1 用 GMM 生成資料

這裡將說明圖 4-6 生成資料的步驟。

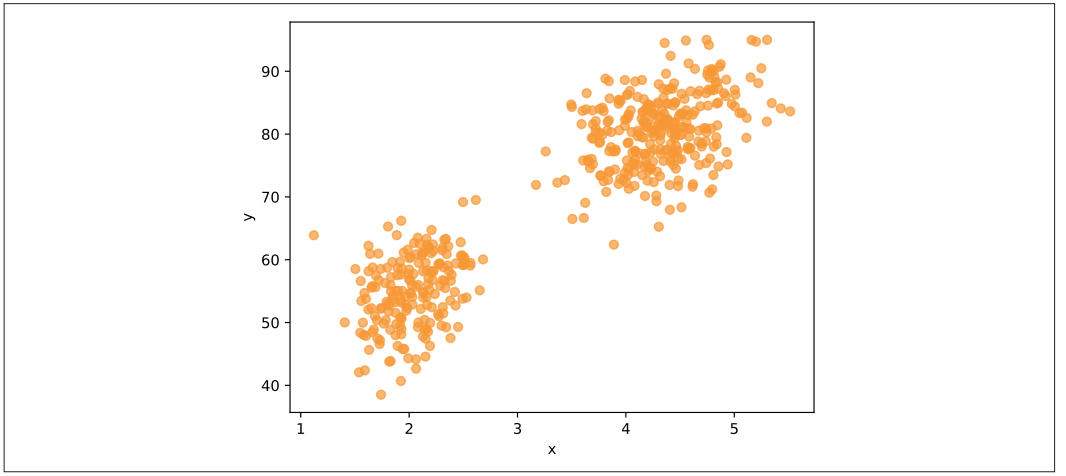

圖 4-6 GMM 生成的樣本散布圖

透過以下步驟可以生成圖 4-6 的資料。

準備兩個常態分布。
Repeat:

❶ 依某種機率分布從兩個常態分布中選出一個。

❷ 從選出的常態分布生成資料。

首先準備兩個常態分布。接著重複進行兩個階段的處理，包括從兩個常態分布中選擇其中一個，從該常態分布生成資料。重複執行這兩個階段的處理，可以得到圖 4-6 的樣本。

6.1.2 計算張量

安裝完畢後，試著使用 PyTorch 吧！首先從 tensor 類別開始。tensor 是張量，也就是多維陣列的類別。處理方式和 NumPy 的 np.ndarray 一樣。例如，執行以下程式碼。

```
import torch

x = torch.tensor(5.0)

y = 3 * x ** 2
print(y)
```

執行結果

```
tensor(75.)
```

如上所示，和 NumPy 一樣可以直覺操作。接著要計算微分。上面的程式碼進行了 y = 3 * x ** 2 的計算，現在要計算 x=5 的微分。我們先用公式算出答案。這次要計算 $y=3x^2$，其微分是 $\frac{dy}{dx}=6x$，代入 $x=5$，微分為 30。

接著試試用程式碼能否取得微分值 30。如果要在 PyTorch 計算微分，可以使用 tensor 實例的 backward() 方法，程式碼如下所示。

step06/tensor.py

```
x = torch.tensor(5.0, requires_grad=True)
y = 3 * x ** 2

y.backward()
print(x.grad)
```

執行結果

```
tensor(30.)
```

在需要微分的張量中，把生成張量時的引數設定為 requires_grad=True，就可以設定該張量必須進行微分。計算後得到結果 y。y 是 tensor 實例，對 y 呼叫 backward() 方法，就能求出微分。結果為 30.0，與公式的答案一樣。

backward 這個字來自**誤差反向傳播法（Backpropagation）**。誤差反向傳播法是可以有效計算微分的演算法，進行一般計算（Forward）與反向計算（Backward），能求得每個變數的微分（反向計算時，微分會反向傳播）。關於誤差反向傳播法的原理，請參考《Deep Learning：用 Python 進行深度學習的基礎理論實作》的說明。

6.1.3 梯度法

接著用 PyTorch 解決一個簡單的問題。我們想找到以下公式表示的函數最小值。

$$y = 100(x_1 - x_0^2)^2 + (x_0 - 1)^2$$

這個函數稱作 Rosenbrock 函數。如**圖 6-1** 所示，函數的形狀很特別，很難找到最小值，因此常當作最佳化問題的基準測試。

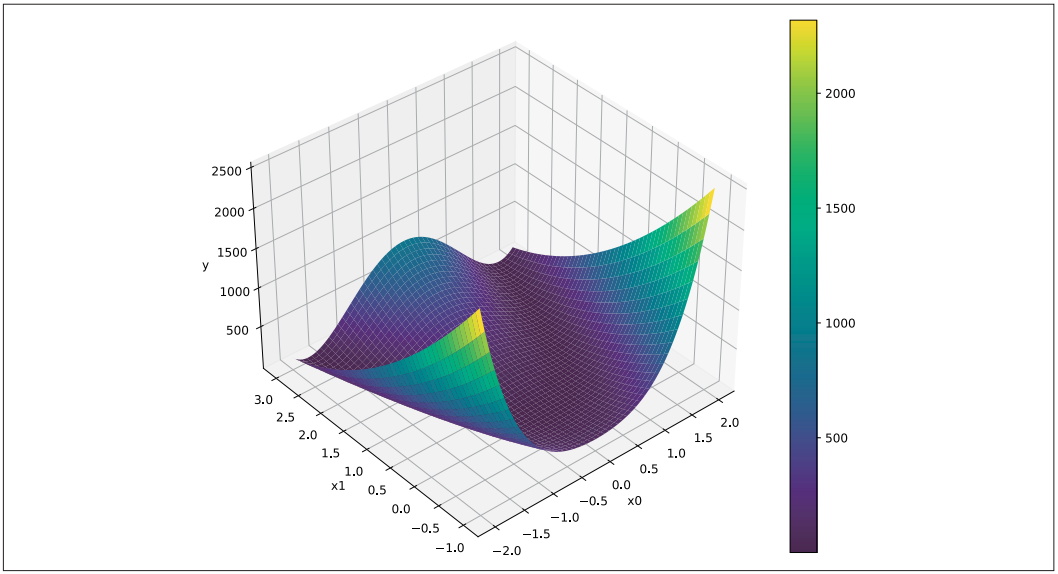

圖 6-1 Rosenbrock 函數的形狀

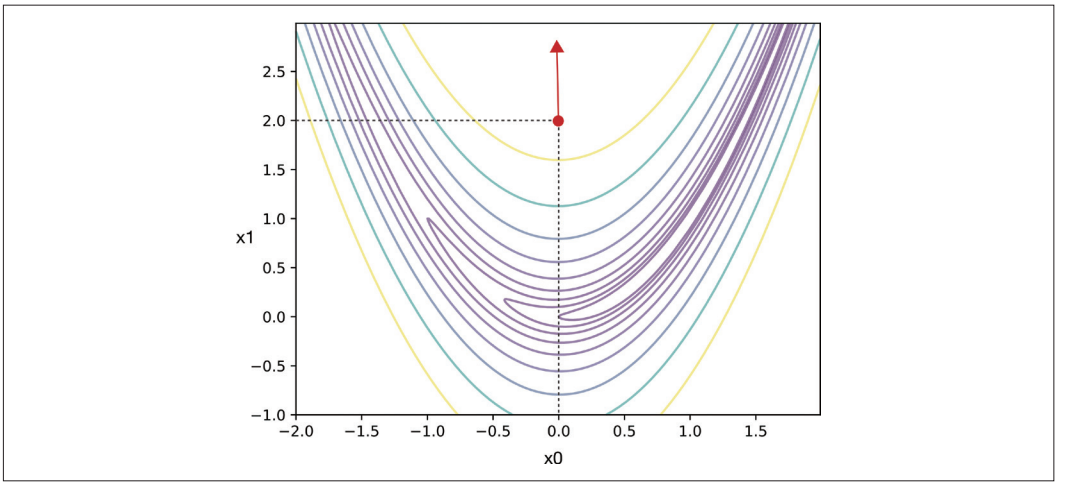

圖 6-2 Rosenbrock 函數的等高線與梯度（梯度縮小成可顯示在圖中的範圍內）

梯度是指在每個點上，函數輸出增加最多的方向。以上面的例子來說，就是指在 `(x0,x1)=(0,2)` 的位置，y 值增加最多的方向是 `(-2, 400)`。換句話說，梯度乘以負號，亦即 `(2, -400)` 的方向就是 y 值減少最多的方向。

許多形狀複雜的函數，其梯度指示的方向未必是最大值（或最小值不一定出現在梯度的反方向上）。但是，如果限制為局部的點，梯度會指出函數輸出增加最多的方向。沿著梯度方向前進一定的距離，並在該位置再次計算梯度，重複執行這些工作，就可以逐漸接近目標位置（最大值或最小值）。這種方法稱作**梯度法（Gradient Method）**。

接著將梯度法套用到我們的問題中。這裡的問題是找出 Rosenbrock 函數的最小值。因此，往梯度的反方向前進。只要注意到這一點，就能按照以下方式進行實作。

step06/gradient.py

```
x0 = torch.tensor(0.0, requires_grad=True)
x1 = torch.tensor(2.0, requires_grad=True)

lr = 0.001     # 學習率
iters = 10000  # 重複次數

for i in range(iters):
    if i % 1000 == 0:
        print(x0.item(), x1.item())
```

STEP 9
擴散模型實作

上個步驟學習了擴散模型的「理論」，接下來要進入擴散模型的「實作」階段。常用於類神經網路的擴散模型稱作「U-Net」，這個步驟將先說明 U-Net 再進行實作，接著學習有效處理時間資料（整數資料）的「正弦波位置編碼」。之後逐一確認、建置增加高斯雜訊的「擴散過程」，最後使用 MNIST 資料集進行擴散模型的學習。

9.1 U-Net

擴散模型使用的類神經網路在公式中顯示為 $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$，這個類神經網路可以預測雜訊，如圖 9-1 所示。

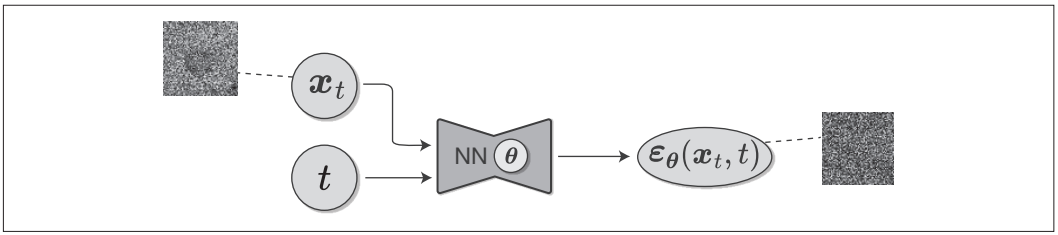

圖 9-1　擴散模型使用的類神經網路

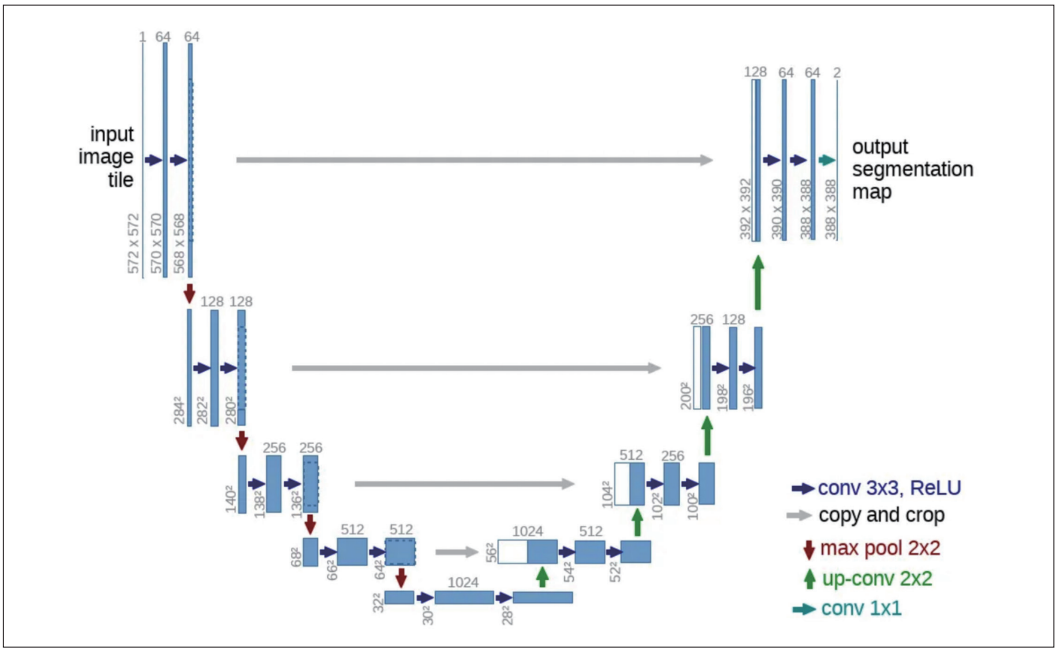

圖 9-4　U-Net 的網路結構（圖引用自論文 [16]）

U-Net 執行的處理分成前半部分「縮小階段」和後半部分「擴大階段」。前半部分的縮小階段會利用卷積層進行處理，逐漸縮小特徵圖。縮小特徵圖的層稱作「下取樣層」。在後半部分的擴大階段，透過卷積層擷取特徵，與前半部分相反，逐漸放大特徵圖。放大特徵圖的層稱作「上取樣層」。

「跳躍連接（Skip Connection）」是 U-Net 的重要特色之一。這是在網路對應的縮小階段與擴大階段之間，直接傳遞特徵圖的機制。U-Net 可以利用跳躍連接掌握整個物體的特徵，同時使用更詳細的空間位置資訊進行處理。

9.1.2　U-Net 實作

假設要利用 $1 \times 28 \times 28$ 大小的 MNIST 資料集建構簡單的 U-Net，網路結構如圖 9-5 所示。

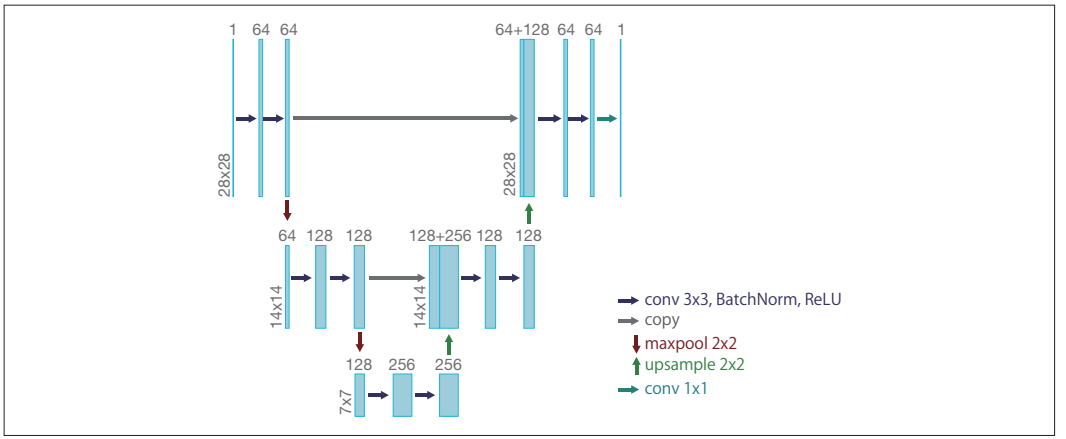

圖 9-5 本節建構的 U-Net

如圖 9-5 所示，包括兩個縮小階段與兩個擴大階段，每個階段都利用二層卷積層進行處理。建構這個 U-Net 時，要先建立 ConvBlock 類別。ConvBlock 類別如圖 9-6 所示，分別執行兩次卷積層、批次標準化層、ReLU 函數的處理。

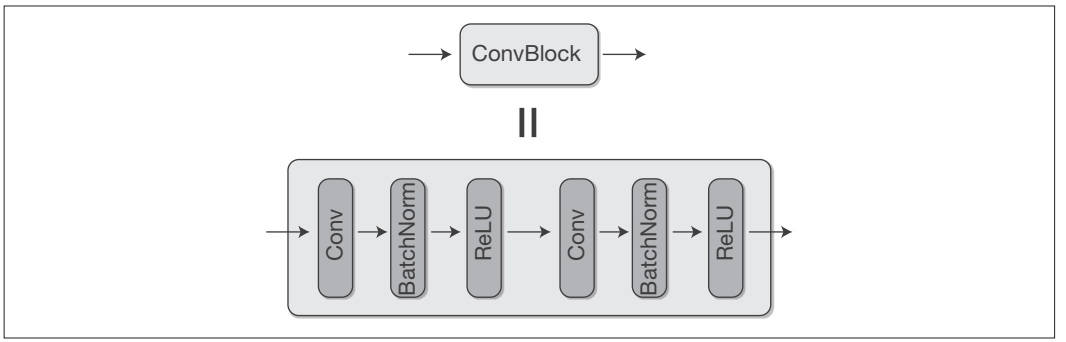

圖 9-6 ConvBlock 類別執行的處理

ConvBlock 類別的程式碼如下所示。

step09/simple_unet.py

```
import torch
from torch import nn

class ConvBlock(nn.Module):
    def __init__(self, in_ch, out_ch):
        super().__init__()
```

10.5 Stable Diffusion

到目前為止，我們建構了處理 MNIST 的「小型擴散模型」（以及「小型條件擴散模型」）。當然，現在的擴散模型相當龐大，而且進行了多次改善。這裡將以改善擴散模型的觀點，提出更進一步的指標。因此，以下將介紹知名的「Stable Diffusion」模型。Stable Diffusion 可以生成高畫質影像，如圖 10-14。程式碼和學習後的權重資料都是公開的，任何人都可以執行該程式碼。

圖 10-14 Stable Diffusion 生成的影像範例（影像引用自 Stable Diffusion 的 GitHub [24]）

Stable Diffusion 也是服務的名稱，而且仍持續進化中。這裡提到的 Stable Diffusion 是首度在論文「High-Resolution Image Synthesis with Latent Diffusion Models [25]」上發表的模型，這個模型可以說是 Stable Diffusion 的起點。

10.5.1 Stable Diffusion 的結構

首先要介紹 Stable Diffusion 的學習過程，如圖 10-15 所示。

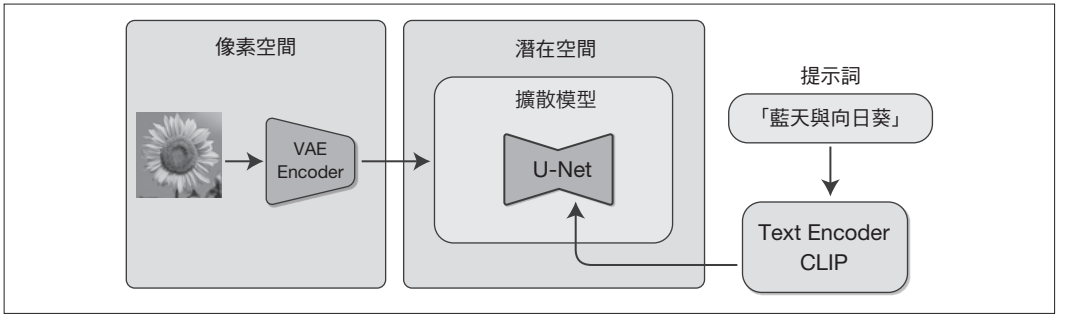

圖 10-15 Stable Diffusion 的學習過程

如圖 10-15 所示，給予影像與文字，進行擴散模型的學習，藉此更新 U-Net 的參數。此外，Stable Diffusion 有以下三個特色。

- **潛在空間：**在潛在空間進行擴散模型的處理。
- **CLIP：**使用名為 CLIP [26] 的類神經網路進行文字編碼。
- **Attention：**使用 Attention 層，將「條件」加入 U-Net。

以下將分別說明這三個特色。

潛在空間

之前介紹的擴散模型是每個時間的資料（$\boldsymbol{x}_t$）與輸入影像（$\boldsymbol{x}_0$）具有相同的元素數（像素數）。換句話說，每個時間的資料都是在與影像相同的向量空間中處理（這可以稱作「像素空間」）。然而，Stable Diffusion 是在潛在變數的空間（「潛在空間」）進行擴散模型的處理。

Stable Diffusion 使用「編碼器」將像素空間轉換成潛在空間。這個編碼器可以使用 VAE 中的模型。降低潛在變數的維數，就能減少擴散模型進行的處理。

生成新資料時，從潛在空間中的高斯雜訊開始進行反向擴散過程。結束最後的反向擴散過程時，使用 VAE 的解碼器轉換成像素空間（圖 10-16）。

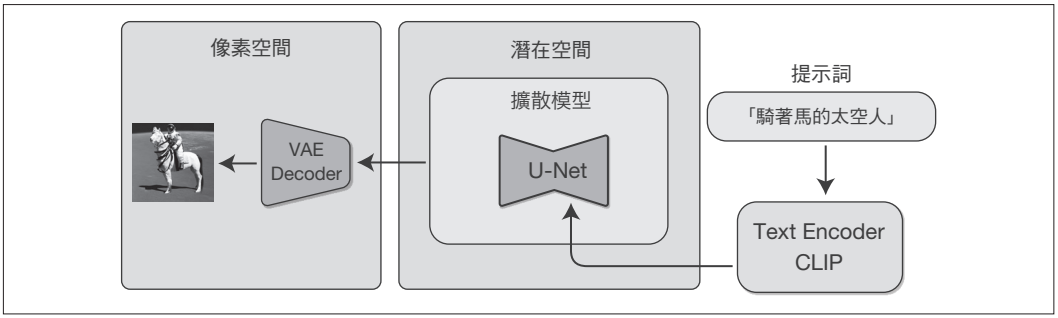

圖 10-16　Stable Diffusion 的資料生成過程