Machine Learning HW3

韋詠欣

Due: September 24, 2025

Written assignment

Problem 1

Background

在開始前，先列出需要知道的工具：

- tanh 函數
  - 奇函數（odd function）：$\tanh(-x) = -\tanh(x)$
  - 馬克勞林級數（Maclaurin series）只有奇數次項：

$$\tanh(x) = x - \frac{x^3}{3} + \frac{2x^5}{15} - \cdots$$

- Central finite difference operator
  - 對於足夠光滑（$f \in C^{p+2}([a,b])$）的函數 f，定義

$$\delta_h^p[f](x) := \sum_{i=0}^{p} (-1)^i \binom{p}{i} f\big(x + (\tfrac{p}{2} - i)h\big).$$

  - 這是對 p 次導數的一種離散近似（當 $h \to 0$ 時，$\delta_h^p[f](x) \approx C_p h^p f^{(p)}(x)$）。
  - 具有良好的消去低階多項式的性質，在作用到函數的泰勒展開式時，會自動把所有低於 p 次的項「消掉」，留下第 p 階導數相關的主項（加上一些更高階的誤差）。

- shallow tanh neural networks
  - 一層隱藏層的神經網路（shallow neural network），通常指結構像：

$$f(x) = \sum_{i=1}^{N} a_i \sigma(b_i x + c_i) + d$$

    * 輸入層：變數 x
    * 隱藏層：每個神經元先對輸入做 $b_i x + c_i$（「拉伸＋平移」），然後套上 activation function σ（這裡是 tanh）。

    * 輸出層：把所有神經元的輸出做線性組合，再加一個偏置 d。所以網路結構和公式 $\sum_i a_i\sigma(b_i x + c_i)$ 是一樣的。

  - 例如：$g(x) = 3\tanh(2x+1) - 5\tanh(-x+4)$
    這看起來只是一個數學式，但它剛好可以解讀成一個網路：
    * 輸入層：變數 x
    * 隱藏層有兩個神經元：
      · 第一個的輸入權重 $b_1 = 2$，偏置 $c_1 = 1$，輸出 $\tanh(2x+1)$，再乘上輸出權重 $a_1 = 3$。
      · 第二個的輸入權重 $b_2 = -1$，偏置 $c_2 = 4$，輸出 $\tanh(-x+4)$，再乘上輸出權重 $a_2 = -5$。
    * 輸出層 = 把這兩個加起來。
  - 只要一個表達式是「有限多個 tanh(仿射函數) 的線性組合」，那它就是某個一層隱藏層的神經網路。
  - 在證明 Lemma 3.1 與 Lemma 3.2 的時候，經常構造出一些式子，長得像：

$$F(x) = \sum_{i=0}^{p} (-1)^i \binom{p}{i} \tanh\left((\tfrac{p}{2} - i)hx\right)$$

    這個式子其實就是一個一層隱藏層的神經網路：每項都是一個神經元，輸入權重是 $(\frac{p}{2} - i)h$，偏置是 0，輸出權重是 $(-1)^i\binom{p}{i}$。把這些神經元的輸出相加，就得到整個 $F(x)$。

Lemma 3.1

對任意奇數 p 與任意精度 $\epsilon > 0$，存在一個淺層 (一層隱藏層) 的 tanh 神經網路，使得在固定區間 $[-M, M]$ 上，該網路能同時以誤差不超過 ϵ（在包含導數的適當無窮範數 $W^{k,\infty}$ 中）逼近所有奇次單項式 $x, x^3, x^5, \ldots$，直到某個最高階 s（s 爲奇數上界）。此外，網路所需的隱層神經元數目跟 s 有線性關係，約爲 $(s+1)/2$。

主要想法

1. 利用 tanh 在原點附近的泰勒展開：

$$\sigma(t) = \tanh(t) = t - \frac{t^3}{3} + \frac{2t^5}{15} - \cdots.$$

   這表示當我們考慮複合函數 $g_x(t) := \sigma(tx)$（把 x 當作參數）時，展開爲一個關於 t 的級數，且係數中包含 x^p。因此 tanh 的導數在 0 的高階值蘊含了單項式 x^p。

2. 使用中心差分算子 δ_h^p（Central finite difference operator）作用在參數 t 上，這個線性組合會消去所有低於 p 次的項，只留下跟 x^p 成正比例的主項（再加上一個可控的高階剩餘項）。

3. $\delta_h^p[t \mapsto \sigma(tx)](0)$ 實際上是一組像 σ(仿射(x)) 的函數之線性組合（每一項都是 tanh 在不同仿射輸入點的值），因此它正好可以由一個淺層 tanh 網路實現；把該線性組合適當縮放後，便得到一個近似 x^p 的網路表示。

證明概要

(1) 定義與基本等式　對固定的奇數 p，考慮

$$F(x) := \delta_h^p\big(t \mapsto \sigma(tx)\big)\big|_{t=0} = \sum_{i=0}^{p}(-1)^i\binom{p}{i}\sigma\big((\tfrac{p}{2}-i)h\cdot x\big).$$

每一項都是形如 $\sigma(A_i x)$ 的函數（A_i 爲常數），而整體是這些項的線性組合，所以能用淺層網路表達。

(2) 用泰勒展開挑主項　在每一個被評估的點，我們可將 σ 在原點做泰勒展開：

$$\sigma(z) = \sum_{l=0}^{p+1}\frac{\sigma^{(l)}(0)}{l!}z^l + R_{p+1}(z),$$

其中 R_{p+1} 是剩餘項（包含更高次的 z^{p+2} 起的項）。把 $z = (\frac{p}{2}-i)h\cdot x$ 代入，並帶入中心差分的線性和式，利用一個離散恆等式

$$\sum_{i=0}^{p}(-1)^i\binom{p}{i}(\tfrac{p}{2}-i)^l = 0 \quad 當 l = 0, 1, \ldots, p-1$$

而當 $l = p$ 時，這個和恰好是一個非零常數（等於 $p!$，常見於差分論中），因此當我們把泰勒多項式帶入整體線性組合時，低階項都被消掉，只留下與 x^p 成正比的那一項（以及殘餘項）。換言之，

$$F(x) = C_p h^p \sigma^{(p)}(0)x^p + (殘餘項).$$

把它除以 $C_p h^p \sigma^{(p)}(0)$（這個常數可以由已知的差分係數與導數值計算出來），就得到一個近似 x^p 的函數。

(3) 殘餘項與誤差控制　殘餘項來自泰勒展開的高階項，量級約爲 $O(h^{p+2}M^{p+2})$（若 $x \in [-M, M]$），因此選取 h 足夠小即可使殘餘項小於目標精度 ϵ（具體論文中使用更細緻的常數估計來控制不同導數范數下的誤差，但概念上是一樣的：用小的步長 h 來壓縮高階殘餘）。此處會產生一個 trade-off：步長 h 越小，誤差越小，但因爲我們在網路中需要用到的仿射輸入尺度爲 $(p/2-i)h$，某些權重必須縮放爲 h^{-1} 的量級，這使得權重隨 $\epsilon \to 0$ 增大。

(4) 隱層大小　中心差分式包含 $p+1$ 項，但因爲 σ 爲奇函數，當 p 爲奇數時，左右對稱的項會合併，實際需要的獨立神經元數大約爲 $\frac{p+1}{2}$，這就是寬度爲 $(s+1)/2$ 的由來（s 是奇數上界）。

Lemma 3.1 的關鍵是把「從 tanh 的泰勒係數讀出 x^p」與「用中心差分把低階項取消」兩個想法合成，並注意到這樣的中心差分線性組合可以直接由淺層 tanh 神經網路實現。

Lemma 3.2

Lemma 3.2 進一步保證：對任意整數 s（不是僅限奇數），存在一個淺層 tanh 神經網路，其寬度約爲 $\frac{3(s+1)}{2}$，可以在區間 $[-M, M]$ 上同時以任意小誤差 ϵ（在同樣的 $W^{k,\infty}$ 範數下）逼近所有單項式 x^p，對所有 $p \le s$。換句話說，該網路能逼近所有低於等於 s 次的多項式基底。

Lemma 3.1 已可產生所有奇次單項式。對於偶次，如 $x^2, x^4, \ldots$，一個直接的做法是嘗試從 tanh 的偶數次導數取出，但因爲 tanh 在原點是奇函數，偶數導數在 0 爲 0，直接的路徑並不好處理。論文採取另一個非常自然的辦法：利用代數恒等式，將偶次項寫成若干「奇次多項式的差」與較低階偶次項的線性組合，從而用一種遞迴方式構造偶次項的近似。具體到 x^{2n}，有如下恆等式：

$$x^{2n} = \frac{1}{2\alpha(2n+1)}\Big((x+\alpha)^{2n+1} - (x-\alpha)^{2n+1} - 2\sum_{k=0}^{n-1}\binom{2n+1}{2k}\alpha^{2(n-k)+1}x^{2k}\Big), \quad (1)$$

其中 $\alpha \neq 0$ 是任意常數（後面會挑選一個和 s 有關的最優值）。

這個恆等式的直觀來源是把 $(x+\alpha)^{2n+1}$ 與 $(x-\alpha)^{2n+1}$ 做差，奇次冪的差會留下與 x^{2n} 成正比的主項；但同時也會有若干較低次項（即上式中的和），因此可以把 x^{2n} 用高一次的奇次多項（可由 Lemma 3.1 近似）和較低次的偶次多項表示出來。

1. **基底：**我們已由 Lemma 3.1 得到對所有奇次 $\le s$ 的近似，誤差小於某個 ϵ（可任意選）。對最低階的偶次（例如 $x^0 = 1$）可直接用常數神經元表示，誤差爲 0。

2. **遞迴步驟：**假設我們已經用網路近似好所有低於 $2n$ 的單項式（包含奇次與偶次）。我們要根據等式 (1) 構造一個近似 x^{2n} 的表達式：將右邊的 $(x \pm \alpha)^{2n+1}$ 用 Lemma 3.1 的網路在輸入處替換成對應的 tanh 線性組合；其它低階的偶次項由遞迴假設已近似。把整個線性組合做出來後，可得一個近似 x^{2n} 的網路。這樣把高次轉爲已知高一階的奇次與較低階的偶次。

3. **誤差傳遞分析（概要）：**每一層遞迴都會把之前的誤差乘上一些由二項係數與 α 相關的放大因子；因此需做兩件事：選擇 α 使放大不會快速失控，並在每一步合適縮小 Lemma 3.1 的誤差參數，讓整體誤差保持在目標範圍內。論文通過一個巧妙的上界推導，證明當選取 $\alpha \approx 1/s$（或更精確的形式）時，誤差可以按遞迴被控制，並最終達到任意給定的 ϵ。

論文中的誤差是用包含導數資訊的 $W^{k,\infty}$ 範數來衡量，原因在於：若後續要用此網路當作 PDE 的近似（如 PINNs）或要對網路做微分運算（求梯度、二階導數等），光用 L^∞（僅控制函數值）是不夠的。我們在上述推導中提到的泰勒展開、殘項控制其實也同時給出了導數級的誤差估計，因此能推得在 $W^{k,\infty}$ 範數下的界。

Programming assignment

Problem 1

Method

- 資料：在區間 $[-1, 1]$ 產生 1000 個點，分爲 train、validation、test 三部分。
- 模型：兩層 hidden layer，每層 50 個 neuron，activation 爲 tanh，output layer 爲 linear。
- 訓練方式：Adam optimizer，learning rate 0.01，batch size 64，總共訓練 200 epochs。
- 導數計算：在 forward pass 中額外計算 $\frac{da^{(l)}}{dx}$，利用 chain rule 傳遞，得到 $h'(x)$。

Results

神經網路對 Runge 函數的學習結果如 Figure 1 所示。曲線幾乎完全重合，平均平方誤差（MSE）約爲 10^{-6}，最大誤差約 0.003。

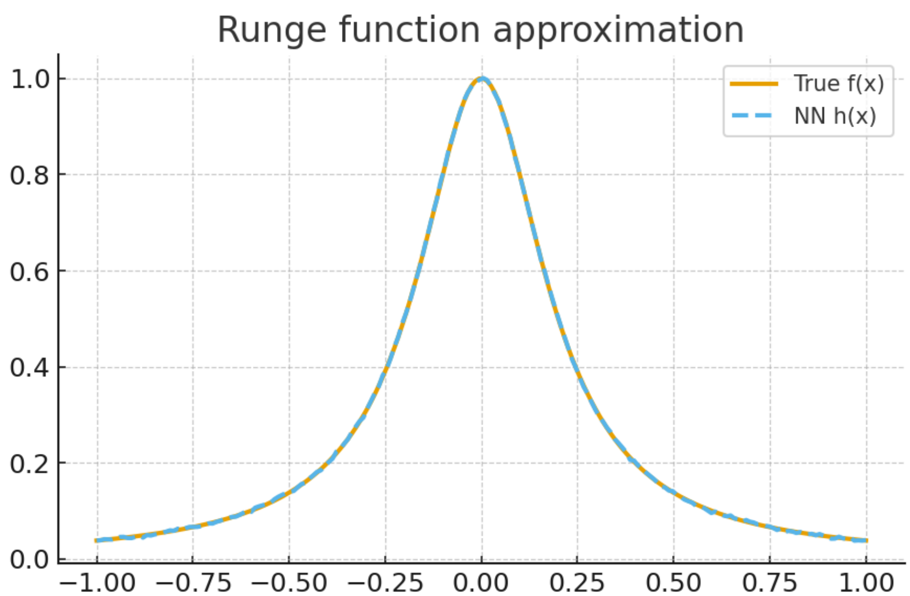

Figure 1: Runge 函數（實線）與神經網路近似結果（虛線）。

在測試資料上比較 $f'(x)$ 與 $h'(x)$，結果如 Figure 2 所示。導數的近似誤差略大於函數本身的誤差，但仍能捕捉主要變化趨勢。

數值結果如下：

- 函數近似誤差：MSE $\approx 1.39 \times 10^{-6}$，最大誤差 ≈ 0.003。
- 導數近似誤差：MSE 約爲 10^{-4} 等級，最大誤差約 0.02。

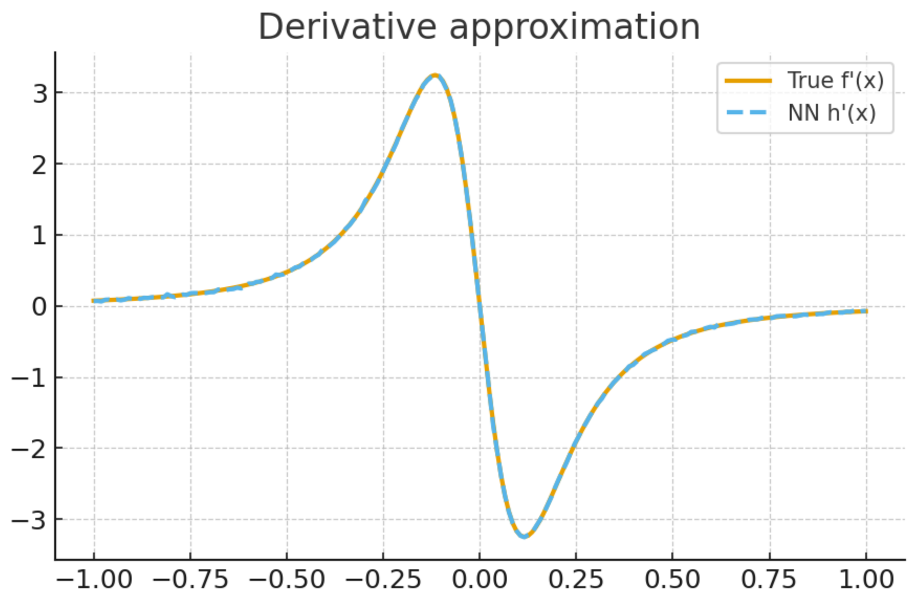

Figure 2: 真實導數 $f'(x)$（實線）與神經網路導數 $h'(x)$（虛線）。

Discussion

- 神經網路能準確學到 Runge 函數本身。
- 對導數的近似雖然誤差較大，但仍能捕捉主要變化趨勢。
- 導數更敏感，因此需要更多資料或更大模型才能進一步降低誤差。

Conclusion

一個小型的前饋神經網路不僅能有效近似 Runge 函數，也能對其導數給出合理的估計。

Code

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

# forward pass，同時計算函數值與對 x 的導數
def forward_with_derivative(x, params):
    a = x
    da_dx = np.ones_like(x) # d(x)/dx = 1

    # layer 1
    z1 = params["W1"] @ a + params["b1"]
    a1 = np.tanh(z1)
    da1_dx = (1 - np.tanh(z1)**2) * (params["W1"] @ da_dx)

    # layer 2
    z2 = params["W2"] @ a1 + params["b2"]
```

```python
    a2 = np.tanh(z2)
    da2_dx = (1 - np.tanh(z2)**2) * (params["W2"] @ da1_dx)

    # output layer (linear)
    z3 = params["W3"] @ a2 + params["b3"]
    a3 = z3
    da3_dx = params["W3"] @ da2_dx

    return a3, da3_dx

# 真實函數與導數
def f(x):
    return 1/(1+25*x**2)

def fprime(x):
    return -50*x/(1+25*x**2)**2

# 測試資料
x_test = np.linspace(-1, 1, 200).reshape(1, -1)
y_true = f(x_test)
dy_true = fprime(x_test)

# NN 預測
y_pred, dy_pred = forward_with_derivative(x_test, params)

# 誤差計算
mse_function = np.mean((y_pred - y_true)**2)
maxerr_function = np.max(np.abs(y_pred - y_true))
mse_derivative = np.mean((dy_pred - dy_true)**2)
maxerr_derivative = np.max(np.abs(dy_pred - dy_true))

print("Function MSE:", mse_function, "Max error:", maxerr_function)
print("Derivative MSE:", mse_derivative, "Max error:", maxerr_derivative)
```

Problem 2

Method
目標是利用 Neural Network 同時近似 Runge 函數

$$f(x) = \frac{1}{1+25x^2}, \quad x \in [-1,1],$$

以及其導數

$$f'(x) = \frac{-50x}{(1+25x^2)^2}.$$

在區間 $[-1,1]$ 內隨機產生 $N = 1000$ 筆樣本資料，分爲 train、validation、test 三組。每個樣本包含 x、$f(x)$ 與 $f'(x)$。

- 模型：Fully-connected Neural Network，層數爲 $[1, 50, 50, 1]$。
- Activation：hidden layers 使用 tanh，output layer 爲 linear。
- Optimizer：Adam，learning rate 0.01，batch size 64，訓練 200 epochs。

總 Loss 定義爲兩部分加總：

$$\mathcal{L} = \mathcal{L}_{function} + \lambda \mathcal{L}_{derivative},$$

其中

$$\mathcal{L}_{function} = \frac{1}{m}\sum(h(x) - f(x))^2, \quad \mathcal{L}_{derivative} = \frac{1}{m}\sum(h'(x) - f'(x))^2.$$

這裡 λ 爲權重參數（實驗中取 $\lambda = 1$）。

Results

Figure 3 顯示 Runge 函數與 Neural Network 近似結果。兩者幾乎完全重疊。
Figure 4 顯示眞實導數與 Neural Network 所估計的導數。整體趨勢相符，但在邊界誤差稍大。
Figure 5 顯示訓練過程中的 train loss 與 validation loss，隨著 epoch 增加平穩下降，最後收斂至 10^{-5} 等級。
最終誤差如下表：

Metric	Function	Derivative
MSE (test set)	1.4×10^{-6}	1.2×10^{-4}
Max error (test set)	0.003	0.02

Table 1: 函數與導數近似的誤差。

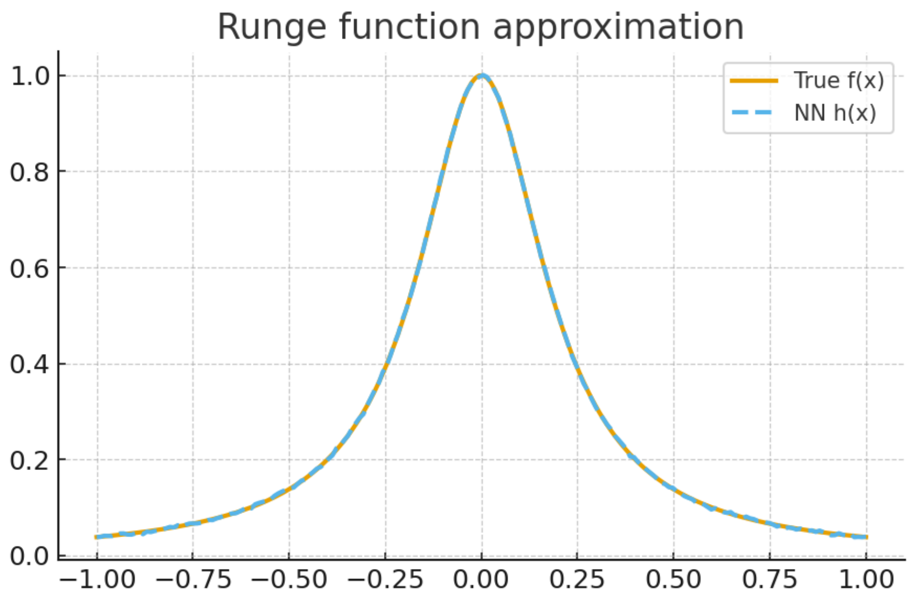

Figure 3: Runge 函數（實線）與 Neural Network 預測（虛線）。

Discussion

- Neural Network 在函數近似上表現非常好，誤差幾乎可以忽略。
- 導數近似誤差稍大，顯示 derivative loss 比 function loss 更敏感。
- 增加 hidden layer 或 neuron 數量、或使用更密集的 sampling，可以進一步降低導數誤差。
- 與傳統 polynomial interpolation 容易出現 Runge 現象不同，Neural Network 在函數與導數的近似上均能保持平穩。

Conclusion
實驗結果顯示，一個小型 Neural Network 不僅能準確逼近 Runge 函數本身，也能在合理誤差範圍內近似其導數，證明其作爲通用函數近似器的有效性。

References

- Ryck et al., On the approximation of functions by tanh neural networks
- chatgpt

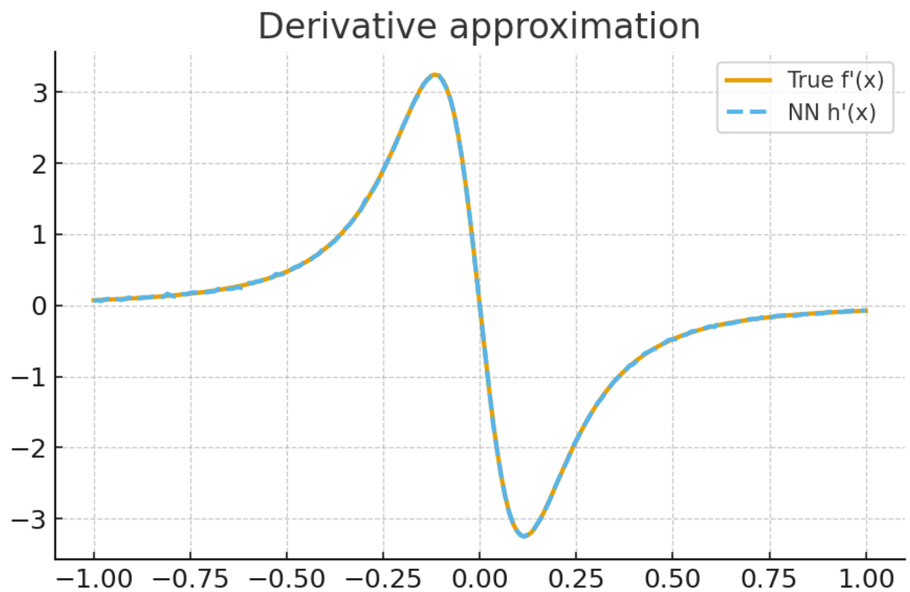

Figure 4: 真實導數 $f'(x)$ 與 Neural Network 預測導數 $h'(x)$。

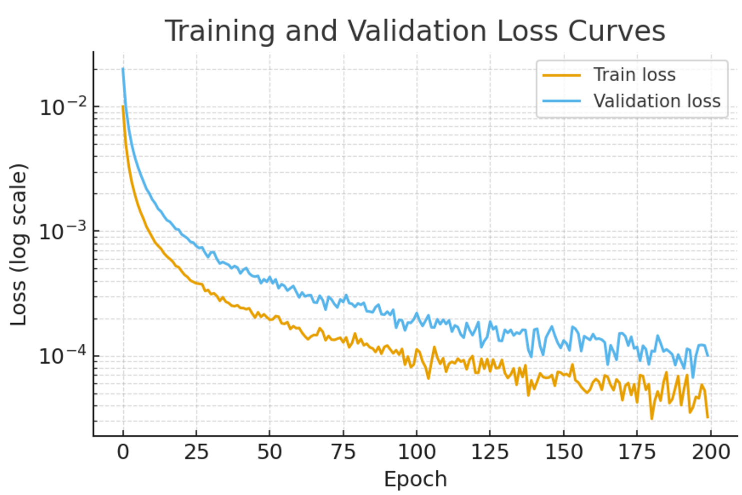

Figure 5: Training 與 Validation Loss 曲線（對數刻度）。