附錄 B

進階閱讀：人工智慧的數學基礎

B-1 神經網路如何在電腦裡表示

你可能會好奇，神經網路這樣的計算模型，要怎麼在電腦裡表示呢？是要造出很多個用電路來模擬神經元，然後用很多導線把它們連起來嗎？確實有這樣的嘗試，8-3 節講的類腦計算就是這樣的嘗試，但是那需要徹底修改神經網路的工作方法，這還需要時間來等技術成熟。 對於現在通行的神經網路則不可行，當然你如果剛好有個幾百億閒錢，同時台積電也願意接你的單，那麼你當然可以這麼做。 但是這樣的電路，不但造價昂貴，而且非常耗電，也不太有彈性。

神經網路在電腦裡的表示方式，其實是線性代數裡講到的矩陣（matrix）。 如果神經網路是一個外送平台，那麼：

- 輸入值（**x**）就像是顧客的訂單資訊（3 個人點餐）
- 權重（**W**）就像是外送員的配送路線（連接 3×4 和 4×2 的路線圖）
- 輸出值（**y**）就像是最終送達的結果（2 個配送點）

用我們之前討論的 3×4×2 網路為例：

$$\text{輸入向量 } \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

第一層權重矩陣（就像外送路線表）：

$$\mathbf{W}_1 = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix}$$

第二層權重矩陣：

$$\mathbf{W}_2 = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix}$$

最終輸出：

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

這就像是外送員的配送過程：

1. 接單：讀取輸入值 $\mathbf{x}$
2. 規劃路線：乘上權重矩陣 $\mathbf{W}_1$
3. 中轉站處理：加上偏差值並通過激活函數
4. 最後一哩路：乘上 $\mathbf{W}_2$ 得到最終結果

用數學來說就是：

$$\mathbf{y} = f\left(\mathbf{W}_2 \cdot f\left(\mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2\right)$$

其中 f 是我們的激活函數，就像是外送員的「智慧判斷」：要不要繞路、要不要走捷徑等。

而 $\mathbf{b}_1$ 和 $\mathbf{b}_2$ 是偏差值，就像是外送員的「個人習慣」：

- $\mathbf{b}_1$ 是第一層的偏差值：

$$\mathbf{b}_1 = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

就像是外送員的「基本配送時間」，不管訂單內容如何，都會加上這個基本時間。

- $\mathbf{b}_2$ 是第二層的偏差值：

$$\mathbf{b}_2 = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

就像是最終配送點的「固定處理時間」，比如電梯等待時間、找零錢時間等。

這些偏差值的作用就像是「調味料」，讓神經網路能更靈活地適應各種情況。沒有它們，就像是餐廳只能照菜單的配方來煮，加了它們，就能根據客人的口味來調整！

這樣的矩陣運算方式有什麼好處呢？

- 非常快：矩陣運算可以利用 GPU 平行處理的特性，大幅提升計算速度。
- 非常整齊：所有數據都排得整整齊齊
- 好擴展：想加更多層或神經元？矩陣輕鬆搞定！

B-2 訓練神經網路的數學解釋

在 1-3-1 節我們用外送平台的例子解釋了神經網路的訓練過程，現在讓我們掀開數學的面紗，看看背後的原理！不過別擔心，我們會盡量用簡單的方式來解釋這些看起來很可怕的數學概念。

B-2-1 損失函數（Loss Function）

還記得剛才說的「差評」嗎？在數學上，我們用損失函數來計算這個「差評」有多慘。想像你在玩飛鏢，目標是紅心：

- 射中紅心，損失值 = 0（完美！）
- 射到邊邊，損失值會變大（唉呦，差一點）
- 射到牆上，損失值會超級大（完全打歪了啦）

最常用的損失函數是「均方誤差」（Mean Squared Error，MSE）：

$$\text{均方誤差} = \frac{\text{所有誤差的平方和}}{\text{資料數量}}$$

或者寫得更詳細一點：

$$\text{均方誤差} = \frac{(\text{預設1-實際1})^2 + (\text{預測2-實際2})^2 + \cdots + (\text{預測n-實際n})^2}{\text{總共有幾筆資料}}$$

為什麼要平方？因為這樣不管是預測太高還是太低，都會被懲罰！就像在調整冷氣溫度：如果你想要 26 度的室溫，設定成 20 度（太低）會讓人冷到發抖，設定成 32 度（太高）會讓人熱到冒汗，兩種情況都不舒服。所以空調系統會努力避免溫度「偏離」目標值，不管是往上偏還是往下偏。在 AI 系統裡也是一樣，預測值偏離實際值愈

多（不管是太高還是太低），懲罰（也就是損失值）就會愈大！ 用平方的好處是同樣多偏離一度，在已經偏差很多的時候的懲罰就更嚴重，比方說如果本來沒有偏差，突然多偏差 1 度（從 26 度變 27 度）的懲罰增加 1，但如果本來就已經偏差很多（從 26 度偏到 31 度再變 32 度），那這個 1 度的懲罰會變成 11[註 1]。這就像是餐廳服務：如果菜已經太鹹了還繼續加鹽，這樣的錯誤會比原本正好的菜多加一點鹽更嚴重！

B-2-2 梯度下降（Gradient Descent）

想像你在一個漆黑的山谷裡找最低點，你只能靠腳底的感覺。梯度下降法就是：

1. 感覺一下四周哪個方向是下坡
2. 往那個方向走一小步
3. 重複以上步驟，直到感覺到達谷底

用數學來說：新的權重＝舊的權重－學習率 × 梯度

其中「梯度」就是那個「下坡的方向」，告訴我們要往哪裡走才能讓損失變小。

B-2-3 學習率（Learning Rate）

學習率決定了每一步要走多大。這個很重要：

- 步子太大：可能直接跨過谷底，永遠找不到最低點（就像跑壘太快，結果直接跳過壘包，沒上壘成功）

註 1： $(32-26)^2-(31-26)^2=6^2-5^2=11$

- 步子太小：要花超久才能到達目標（就像每天只讀一頁書，要等到退休才能考試）
- 剛剛好：穩定地往目標前進（每天適量學習，進步最快）

一般我們會用 0.001 到 0.1 之間的數值，視情況調整。

B-2-4 反向傳播（Back Propagation）

這是最神奇的部分！反向傳播就像是在玩彈珠台遊戲，但是我們倒著看遊戲過程。 想像一個多層的彈珠台：彈珠從上面滾下來，經過很多鐵釘（就像資訊在神經網路中流動），最後彈珠停在某個位置，但不是我們想要的位置（這就是預測的誤差），現在我們要調整每個斜坡和鐵釘的角度，讓彈珠下次能滾到正確的位置。

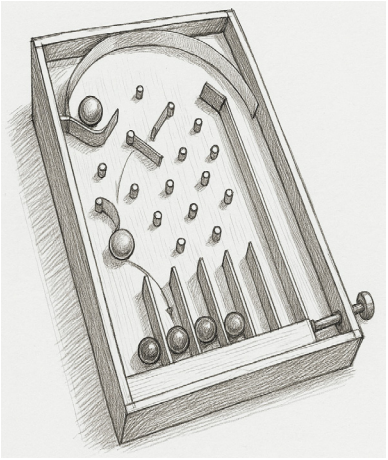

圖 B.1 彈珠台遊戲

反向傳播就是從最後的位置開始，逐步往回算：

1. 如果彈珠差了 3 公分，我們先看最後一個鐵釘：如果鐵釘位置改變 1 公分，彈珠位置會改變多少？那麼要修正 3 公分，鐵釘應該調整幾公分？
2. 接著看倒數第二層的鐵釘 如果這層鐵釘位置改變 1 公分，會讓最後的結果改變多少？要算出這個「影響力」，就是微積分最基本的概念。

3. 一直往上算，直到第一層：每一層的調整都會影響到後面所有層，離最後結果越遠的層，影響力通常越小，這就是為什麼深層神經網路很難訓練，因為前面層的調整效果會被稀釋。

這就像是在解一個連鎖反應的數學題：如果 A 影響 B、B 影響 C，那麼 A 對 C 的影響就是這些效果的乘積。就好像是在算「如果我遲到影響老闆心情，老闆心情影響我的獎金，那我遲到最後影響了多少獎金」這種悲劇性的連鎖效應（欸）。或是「如果我吃麻辣鍋影響腸胃，腸胃影響心情，最後影響我在 IG 上的自拍表現」這種生活日常。不過別擔心！我們不用真的去算這些複雜的數字，電腦會幫我們處理！（感謝電腦，讓我們能專心處理人生的其他連鎖反應）我們只要理解：每一層的調整，都是根據它對最終結果的影響程度來決定的，就像蝴蝶效應，但至少這次我們知道誰是那隻蝴蝶！

B-3 變形金剛的注意力機制

讓我們用數學的眼光來看這個過程：每個詞在 Transformer 中都會被轉換成三個向量：

- Query（Q）：就像是「我要找什麼？」
- Key（K）：就像是「我有什麼特徵？」
- Value（V）：就像是「我能提供什麼資訊？」

假設我們有一句話：「我喜歡吃巧克力」，注意力分數的計算方式是：

$$\text{Attention}\left(Q,K,V\right)=\text{softmax}\left(\frac{QK^{T}}{\sqrt{d_{k}}}\right)V$$

這個公式看起來嚇人，但其實很有趣：

1. QK^T 計算每個詞和其他詞的相關性
2. $\sqrt{d_k}$ 用來調整分數的規模，避免數值太大
3. softmax 把分數轉換成機率分布
4. 最後和 V 相乘，得到加權後的結果

更厲害的是，Transformer 用了「多頭注意力」（Multi-Head Attention）：

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_h)W^O$$

每個「頭」都可以學習不同的語言特徵：

- 有的頭專注於語法結構
- 有的頭關注語義關係
- 有的頭捕捉長距離依賴

Transformer 的架構還包含了：

1. **位置編碼**（Positional Encoding）：

$$PE_{(pos,2i)} = \sin\left(pos / 10000^{2i/d_{model}}\right)$$

讓模型知道詞的位置資訊

2. **前饋神經網路**（Feed-Forward Networks）：

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

處理每個位置的特徵轉換

3. 層標準化（Layer Normalization）：

$$\text{LayerNorm}(x) = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

穩定訓練過程

這個設計的優勢在於：

- **並行計算**：RNN 像是一條生產線，必須一個字一個字處理，但 Transformer 可以同時處理所有輸入。這種設計特別適合現代 GPU 的並行運算架構，訓練速度可以提升數十倍。
- **全局視野**：不管句子有多長，每個詞都能直接關注到其他所有詞。第一個詞可以直接看到最後一個詞，完全不受距離限制。這特別適合處理長文本中的遠距離關係，徹底解決了 RNN 中的長期依賴問題。
- **可解釋性**：通過觀察注意力權重，我們可以直接看到模型在處理每個詞時到底在關注什麼。這種透明度不僅幫助研究人員理解模型的決策過程，還便於診斷和改進模型的表現。

這就是為什麼現代的大語言模型（如 GPT 系列）都採用 Transformer 架構。它不僅解決了長序列處理的問題，還帶來了前所未有的模型性能提升！

B-4 生成對抗網路技術原理

在數學上，生成對抗網路（Generative Adversarial Network, GAN）的訓練目標可以用以下的 minimax 損失函數來表示：

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\log D(x)\right] + \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

這個公式看起來複雜，但其實背後的想法就像是一場對抗比賽。G 是生成器（Generator），它負責「畫畫」，試圖模仿真實資料來製作看起來很真的假資料；D 是判別器（Discriminator），它負責「鑑定」，判斷輸入的是來自真實世界的資料，還是生成器畫出來的假資料。

在這個設定中：

- $p_{\text{data}}(x)$ 表示真實資料的機率分布。
- $p_z(z)$ 是潛在空間中的隨機雜訊分布，常見的選擇是標準常態分布 $\mathcal{N}(0,I)$。
- $G(z)$ 是將雜訊 z 映射成假樣本的生成器輸出。
- $D(x)$ 是判別器對輸入樣本 x 判斷為真實的機率，輸出值在 $[0,1]$ 之間。$D\left(G(z)\right)$ 是判別器對這個假作品的判斷結果

這個遊戲的目標是：

- 生成器希望讓假樣本 $G(z)$ 騙過判別器，使得 $D\left(G(z)\right)$ 越接近 1 越好。
- 判別器希望能正確區分真假資料，使得 $D(x)$ 趨近 1，$D\left(G(z)\right)$ 趨近 0。

這形成了一個雙方角力的對抗過程，就像是「偽造者」與「鑑定師」的拉鋸戰。這種關係在博弈論中稱為**零和遊戲**（zero-sum game），而其理想終點是一種稱為 Nash equilibrium 的平衡狀態：生成器所產生的資料與真實資料無法被判別器區分，此時 $D(x) = D(G(z)) = 0.5$，也就是說，判別器對每個輸入都無法有信心地判斷其真偽。

◆ 訓練過程中的學習策略：

在每一次訓練迭代中，判別器與生成器會輪流更新它們的參數：

判別器的目標是最大化 $V(D,G)$，也就是提高對真實樣本的信心、降低對假樣本的信心。其參數更新的方式是使用**梯度下降**（gradient ascent）：

$$D_{\text{new}} = D_{\text{old}} + \alpha \nabla_D V(D,G)$$

其中 α 是學習率，控制每次更新的步伐大小。

生成器的目標是最小化 $V(D,G)$，試圖讓 $D(G(z))$ 越來越大，也就是讓判別器越來越容易被騙。這邊通常採用 gradient descent 來更新 G 的參數。

這種輪流交替更新的方式，就像兩位對手在不斷進行較勁：

- 當判別器太強時，生成器會受到更多挑戰，進而學會更真實的資料模式。
- 當生成器進步後，判別器就得進化自己的辨識能力，否則就會被騙。

◆ 一個生活化的比喻：

你可以把整個流程想成一場精彩的「貓捉老鼠」遊戲。生成器是狡猾的老鼠，一開始牠畫出來的假畫很粗糙，容易被判別器（貓）識破。但每次被抓住，牠就會學習改進，畫得更像；而貓也會記住這些技巧，變得越來越敏銳。如此往復，雙方你追我跑，技能不斷提升。

到了某個平衡點，生成器創作出來的假畫已經與真實資料難以區分，判別器的判斷也變得沒那麼有信心，這就代表整個系統達到了訓練的理想狀態。

B-5 擴散模型：從雜訊中重建世界

擴散模型（Diffusion Models）是一種生成模型，其基本想法是透過模擬「正向擴散」（加入雜訊）與「反向擴散」（去除雜訊）的過程，來學會從純隨機的雜訊中逐步生成真實圖像。這種方法不僅在圖像生成領域取得了驚人的成果，也是目前許多高品質影像生成系統（如 DALL・E、Stable Diffusion）背後的核心技術。

B-5-1 正向過程：雜訊的漸進注入

首先，我們定義一個正向馬可夫鏈 $\{x_0, x_1, \ldots, x_T\}$，其中 x_0 是真實資料，x_T 是幾乎純雜訊。正向過程的目標是逐步對資料加入高斯雜訊：

$$q(x_t | x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}\, x_{t-1}, \beta_t \mathbf{I}\right)$$

其中 β_t 是控制第 t 步雜訊強度的參數（稱為**擴散率**），通常設計成逐步遞增的序列。當 t 趨近最大值 T 時，x_T 變成幾乎純高斯白雜訊。

利用公式遞推可得：

$$q\left(x_t \middle| x_0\right) = \mathcal{N}\left(x_t; \sqrt{\overline{\alpha}_t}\, x_0, \left(1-\overline{\alpha}_t\right)\mathbf{I}\right)$$

其中 $\overline{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$，表示累積保留原始資訊的比例。

B-5-2 反向過程：從雜訊中復原資料

反向過程的目標是從雜訊資料 x_T 中一步步去除雜訊，最終還原出資料 x_0。這個過程的機率分佈定義如下：

$$p_\theta\left(x_{t-1} \middle| x_t\right) = \mathcal{N}\left(x_{t-1}; \mu_\theta\left(x_t, t\right), \Sigma_\theta\left(x_t, t\right)\right)$$

其中 μ_θ 與 Σ_θ 是透過神經網路（例如 U-Net 架構）所學習的平均與變異數函數，參數由 θ 表示。

在實作中，通常簡化假設 Σ_θ 為固定常數，模型只預測雜訊 $\epsilon_\theta(x_t, t)$，並藉由以下方式還原 x_0：

$$\hat{x}_0 = \frac{1}{\sqrt{\overline{\alpha}_t}}\left(x_t - \sqrt{1-\overline{\alpha}_t}\,\epsilon_\theta(x_t, t)\right)$$

B-5-3 訓練目標：還原雜訊

為了讓模型學會如何還原雜訊，我們隨機抽取 $t \sim \text{Uniform}(1, T)$ 並構造帶雜訊的樣本：

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon, \epsilon \sim \mathcal{N}(0, \boldsymbol{I})$$

然後令模型預測 $\hat{o}$，訓練目標是最小化下列均方誤差（MSE）損失：

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{x_0, \epsilon, t}\left[\left\| \epsilon - \epsilon_\theta(x_t, t) \right\|^2 \right]$$

這個簡化目標已被證實能有效訓練出高品質的生成模型。

B-5-4 生成過程：從雜訊走回現實

當模型訓練完成後，我們可以從標準高斯雜訊 $x_T \sim \mathcal{N}(0, \mathbf{I})$ 開始，根據模型學到的反向轉移機率 $p_\theta(x_{t-1} | x_t)$ 逐步還原出 x_0。這個過程類似於圖像的「魔術顯影」，每一步都讓影像更加清晰。

B-5-5 與 GAN 相比的優勢

與生成對抗網路（GAN）相比，擴散模型雖然生成速度較慢，但在穩定性與樣本多樣性方面表現更好。它不需要對抗式訓練，訓練目標明確，且能自然地進行多步推理與條件生成（如 text-to-image）。

擴散模型的魅力在於它結合了數理簡潔與深度學習的強大表現力。從高斯雜訊中逐步「洗回」一幅圖像的過程，不僅優雅，也打開了創意 AI 應用的新視野。