第六章 機器學習的工作流程

在先前的例子中，我們都假設手上的資料集已標注好，可以馬上用來訓練模型。不過在真實世界中，狀況通常不會這麼理想。

想像一下，我們創辦了一家機器學習顧問公司，需要處理的專案包括：

- 為圖片分享社群設計個人化的照片搜索引擎：輸入「婚禮」一詞就會列出我們在婚禮所拍攝的所有照片，不需要任何人為標籤。
- 在聊天軟件中自動標籤垃圾或惡意內容。
- 為線上廣播的聽眾建構一個音樂推薦系統。
- 偵測電子商務網站的信用卡詐騙案件。
- 預測廣告點擊率，進而決定在特定時間點，要為特定消費者投放什麼廣告。
- 在餅乾生產線上偵測異常的餅乾。
- 利用衛星影像尋找至今尚未發現的古蹟位置。

該注意的道德倫理問題

有時，我們可能會碰到存在道德爭議的專案，例如：「建構可以根據面孔照片，判斷當事人是否可信的AI工具」。

首先，這個專案本身是否合理就很可疑：因為沒有什麼證據表明外表和一個人的可信度有關。

其次，該專案將引發各式各樣的道德問題。在收集資料集時，將無可避免地會把標註者的主觀成見滲透進來。 因此，在這樣的資料集上所訓練出來的模型，也必然具備這樣的偏見。

在這個大多數人對技術了解不深的社會，「Al說這個人不可信」居然會比「John Smith 說這個人不可信」來得更有說服力（編註：意指比起真實人類的意見，民眾居然更相信Al的判斷），而且也更客觀，雖然Al的學習對象還是人的主觀判斷。訓練出的模型可能會擴散標註者的錯誤判斷，對真實世界的人類生活造成負面影響。

科技並不會自動中立，而是會隨人們的選擇，展現出不同的面貌。如果我們所做的事對這個世界有所影響，那麼這個影響也會與道德相關：在應用科技時的選擇也必然是道德選擇。我們一定要時刻考慮，我們希望做出的成果能體現什麼樣的價值。

在本章，我們將會介紹機器學習的一套通用流程，其適用於任何的機器學習任務。這套模版會整合第4章和第5章學過的方法，而且我們也會建立起更清晰的概念，並在接下來的章節中派上用場。

大致上，機器學習的通用工作流程可以歸納為 3個部分：

1.定義任務 （Define the task）：

瞭解客戶需求背後的問題領域與業務邏輯、收集資料、瞭解資料內容，並選擇衡量任務成功與否的標準。

2.開發模型（Develop a model）：

首先要準備好模型可以處理的資料、並選擇評估模型的機制以及要打敗的基準線。接著訓練第一個模型，這個模型要具備一定的普適化能力，並且能過度配適。然後再進行常規化和調整模型，直到可以展現出最佳的普適化表現。

3.部署模型 （Deploy a model）：

向相關人員展示開發成果，將模型部署至網路伺服器、行動 app、網頁或嵌入式裝置中，並監控模型的真實表現，然後開始收集建構下一代模型所需的資料。

6-1 定義任務

如果我們對處理的任務沒有足夠的認識，那麼就不會有好的成果。

想一想、為什麼客戶會嘗試解決這個問題？

他們會從解決方案中得出什麼價值？

我們設計出來的模型會被如何應用，能不能和客戶的業務流程彼此匹配？

目前有什麼可用的資料，或者是可以收集的資料？

對於特定的商業問題，我們可以將其對應到哪一類型的機器學習任務？

6-1-1 定義問題範圍

若想定義機器學習問題的範圍，通常需要和相關人員進行多次詳細的討論。

在討論時，以下問題應該擺在第一順位：

* 輸入資料是什麼？我們想預測什麼？我們得先有訓練資料，才能學習如何做出正確的預測：

舉例來說，我們要先有電影評論，以及對應的情緒標註（正面評論/負面評論），才能學習如何分類電腦評論的情緒。在多數狀況下，我們得自己動手收集資料並加以標註（下一節會討論到這個部分）。

* 我們面對的是什麼類型的機器學習任務？二元分類任務？多類別分類任務？純量迴歸任務？向量迴歸任務？多類別、多標籤的分類任務？影像分割任務？排序任務？又或者是資料分群任務、生成任務或強化式學任務？對某些任務來說，機器學習不見得是最好的解決方法，那些傳統的統計分析方法也許更有效。

1 照片搜尋引擎專案是一個多類別、多標籤的分類任務。

2 垃圾內容偵測專案是二元分類任務，如果將「惡意內容」設定為獨立的類別，就會變成三類別的分類任務。

3 使用矩陣分解 （也稱協同過濾）來建構音樂推薦引擎，會表現得比使用深度學習方法來得好。

4 信用卡詐騙偵測專案是二元分類任務。

5 點擊率預測專案是純量迴歸任務。

6 異常餅乾偵測是二元分類任務，不過在一開始我們也需要一個物體偵測模型，以將餅乾所在區域從原始影像中裁切出來。需注意的是，被稱為異常偵測的機器學習技巧，在這個任務中的表現並不優異。

7 從衛星影像發現新的古蹟位置是物體相似度的排序任務：我們要找出跟現有古蹟最相似的影像。

* 現有的解決方案長什麼樣子？客戶手上或許已經有一些人工打造的演算法，透過一堆「if」規則來判斷垃圾內容或是偵測信用卡詐騙案件。在客戶的餅乾工廠裡，搞不好就有一個人坐在生產線旁，盯著一片片餅乾從眼前經過，只要一發現異常的餅乾，就手動把殘缺品挑出來。又或者是客戶自己設計歌曲推薦清單，然後發給特定歌星的粉絲。

總的來說，我們應該先明白目前既存的系統為何，以及它們是如何運作的。

* 是否需考量特定的限制條件？舉例來說，在建構垃圾內容偵測系統時，由於通訊軟體的設計架構是採用端到端加密的做法，因此這個偵測模型就必須在使用者的手機上運行，並使用外部資料集來訓練（編註：因為使用者的資料經過加密，我們無法取得也無權使用）。

另外，異常餅乾偵測模型可能對延遲時間有較高的要求，因此只能在工廠的嵌入式裝置上運行，而非在遠端的伺服器上。總之，我們一定要先搞清楚整體的脈絡，才能把事情做好。

在完成一開始的研究後，就應該能掌握輸入為何、目標值為何，以及當前問題對應到的機器學習任務種類為何了。請留意我們在這一階段所做的假設：

* 我們假設確實可以透過輸入來預測出目標值。
* 我們假設現有（或收集到）的資料具備足夠的資訊，可以用來學習輸入與目標值之間的關係。

我們必須等到有了一個可以運作的模型後，才能驗證這些假設是否成立。

機器學習無法解決所有問題，即使我們收集了一堆輸入樣本（統稱為X）跟目標值（統稱為Y），但這並不代表X蘊含足夠的資訊，可以用來預測出Y。

舉例來說，如果我們手上只有某支股票的歷史價格，就想藉此預測出該股票在市場中的走向，基本上是不可能成功的，因為歷史價格無法提供足夠資訊來做出可靠的預測。

6-1-2 建立資料集

當我們明白了任務的本質，而且也搞清楚輸入與目標值後，就可以開始收集資料了。在大部分機器學習專案中，這是最費力、耗時，而且成本最高的部分。

＊　在照片搜尋引擎的專案中，我們需要先挑選用來分類的一組標籤（可能是從10,000個常見的影像類別中挑選），然後用它們來手工標註使用者所上傳的幾十萬張影像。

＊　在偵測垃圾內容的專案中，由於所有對話經過了端到端加密，因此無法用這些對話來訓練模型。我們要另外取得未經過濾的社群帖文，然後將其中數以萬計的內容標註為「垃圾內容」、「惡意內容」或「正常內容」。

＊　就音樂推薦引擎而言，我們可以直接把使用者給的「讚」當作資料，不需要再收集什麼新資料。點擊率的預測也是一樣，我們已經有了歷年廣告的大量點擊資料。

＊　若要訓練異常餅乾偵測模型，就得在生產線上安裝相機來收集數以萬計的影像，並對這些影像進行手工標註。餅乾工廠內的工人適合進行標註的工作，這過程並不會太困難。

＊　在衛星影像的專案中，需要考古學家團隊收集既存古蹟的影像資料庫。對於每一個古蹟，我們需要它們在不同氣候狀況下的衛星影像。若想訓練出不錯的模型，我們至少要有幾千張不同古蹟的照片。

我們在第5章學過，模型的普適化能力幾乎來自於訓練資料的特性，包括資料集的規模、標籤的可靠性、特徵的品質等。投資在建立良好的資料集是值得的。

如果你有多出的 50 個小時可以用來處理導案，那麼把這些時間花在收集更多資料，一定會比不斷試圖修正、優化模型來得更有效率。

如果我們正在進行監督式學習，那麼在收集完輸入資料後（例如影像），還得為它們加上標註（例如為每張影像賦予標籤），也就是為每個資料樣本指定要模型預測出的目標值。

有時，標籤也可以自動取得，例如音樂推薦任務或是點擊率預測任務中的標籤。不過在大多數情況下，我們還是得自己動手標註，這是個非常辛苦的工作。

投資在資料標註工具

資料標註流程決定了目標值的品質，也間接決定了模型的品質。展開標註工作前，請先仔細考慮以下選項：

* 我們需要自己標註資料嗎？
* 我們需要使用外包平臺（例如亞馬遜經營的 Mechanical Turk）來收集標籤嗎？
* 我們需要使用專業資料標註公司的服務嗎？

將標註任務外包也許可以節省成本和時間，但同時也很難控制標註品質。使用像是 Mechanical Turk 的外包平台或許花費不多、也可以處理大量資料，但是最後得到的標註結果可能充滿雜訊。

若想找出最佳選項，請考慮以下面向：

* 資料標註者一定要是特定領域的專家，還是誰都可以做？如果只是做貓狗分類，也許誰都可以，但是如果要分類狗的品種，那就得具備專業的知識了。同樣地，想在 CT 掃描（電腦斷層掃描）圖上標註出骨折部位，就很可能需要具備醫學專業知識了。

* 如果需要特定知識才能標註資料，那麼有可能訓練別人來做嗎？如果沒辦法，那要如何找到相關領域的專家？

* 我們自己瞭解學家怎麼進行標註嗎？如果不瞭解，那就只能把資料集當作是黑盒子，而且沒辦法手動進行特徵工程。這個問題並不是非常嚴重，但一定會造成某些限制。

如果我們決定要自己標註，那麼要用什麼軟體來記錄標註結果？我們有可能需要自己開發相關工具。具生產力的資料標註軟體可以幫我們節省大量時間，所以在專案前期對相關工具的投資是非常值得的。

留意不具代表性的資料

機器學習模型只能處理那些與曾經見過的資料相似的輸入資料。因此，訓練資料一定要足以代表實際運作的資料（編註：即那些模型實際投入運作後所會遇到的資料），這是所有資料收集工作的基礎。

假設我們正在開發一個可讓使用者對餐點拍照，進而找出餐點名稱的手機應用。我們透過某個專業美食社群上所分享的照片來訓練模型。

開發完成後，終於到了部署的階段，但使用者的不滿留言蜂擁而入：應用程式的錯誤率高達80%！ 發生了什麼事？在評估階段，模型在測試集上的準確度明明超過了90%！

原來，使用者在不同餐廳用不同手機在不同場景下隨手拍出的照片，跟我們訓練時所用的專業美食照相差甚遠：換句話說，訓練資料不足以代表實際運作的資料。

如果狀況允許，要儘可能從模型未來實際運作的環境中直接收集訓練資料。

要進行影評情緒分類，那就從IMDB上收集最新的評論，而不是從美食網站收集餐廳評論，也別從社交媒體收集資料。如果要進行社群貼文的情緒分類，就直接從社群網站上收集真實的貼文資料，並進行標註。如果無法使用實際運作的資料來訓練，那一定要搞清楚訓練資料和實際運作的資料間的差異，然後主動去修正這些差異。

你需要留意的另一個現象是概念飄移。在真實世界中，幾乎所有任務都會碰到這個狀況，特別是那些需處理使用者資料的任務。概念漂移的根源來自於實際資料的特性不斷變動，進而導致模型準確度逐漸下降。

在2013年訓練出的音樂推薦引擎，放到今天可能已經不太具參考性了。同樣地，利用2011年 IMDB資料集訓練出的模型，也很難在 2022年的影評上取得好表現，因為用詞、表達方式、電影類型等都已大不相同了。

在偵測信用卡詐騙的案例中，概念漂移的狀況更是明顯，因為詐騙手法每天都在變化。若想減緩概念漂移的問題，就需要持續收集資料、進行標註，並重新訓練模型。

請時刻記得：機器學習只能用來記憶訓練資料中的態樣，因此只能辨識曾經看過的東西。利用過去的資料來訓練模型、然後用來預測未來，其實只是假設未來的運作模式會和過去一樣，但狀況通常並非如此。

抽樣偏差（sampling bias）造成的問題

當資料集欠缺代表性時會發生很多狀況，其中最隱晦也最常見的狀況就是抽樣偏差。

抽樣偏差的根源在於：資料收集的方式與某些要預測的事物產生關聯，進而導致資料內容有所偏差。

最著名的例子發生在1948年的美國總統選舉。在投票日當晚，芝加哥論壇報的頭條為《杜威擊敗杜魯門》。到了隔天早上，真正的勝利者卻是杜魯門。

為何會出現如此嚴重的錯誤？這是因為芝加哥論壇報的編輯相信了電話民調的結果，但在1948年，不是什麼人都擁有電話。他們多半是富裕階級、保守主義者，而且支持共和黨候選人杜威，以至於無法代表投票群體的真實分佈，結果導致了抽樣偏差。

如今，每個電話民調都會考慮抽樣偏差的問題。不過，這不代表現在不會再發生抽樣偏差，但和 1948年不同的是，民調專家如今會採取各種方法來修正這個問題。

6-1-3理解資料

把資料集當成黑盒子處理，是最要不得的做法，在開始訓練模型之前，我们應該先探索及視覺化資料，以對資料有整體的概念、並思考它們如何協助實現預測能力，這樣做有助於進行特徵工程並找出潛在問題。

＊　如果資料中包括影像或自然語言，請直接抽幾個樣本（以及對應的標籤）出来看看。

＊　如果資料中包合數值特徵，可以將這些特徵值繪製出直方圖，進而對資料的分佈狀況有概括的了解。

＊　如果資料包含位置訊息，可以直接畫在地圖上，也許就會出現一些較為清晰的態樣。

＊　某些樣本是否缺少某些特徵值（即存在缺失值）？如果是的話，在準備資料的階段就要處理這個問題（下一節將介紹具體做法）。

＊　如果我們要處理分類任務，請算出每個類別的樣本數量。每個類別的樣本數量是否相近？如果不是，那就需要處理樣本不平衡的問題。

＊　檢查是否存在目標值洩漏的問題：

即訓練資料中的特徵提供了目標值的資訊，但這些資訊在實際應用場景中卻無法取得。舉例來說，假設我們利用病歷資料來訓練模型，用以判斷病人未來是否將接受癌症治療。

如果病歷資料中包含「某病人是否診斷出罹患癌症」的特徵，那就代表我們要判斷的結果（目標值），其實早就洩漏到原先的訓練資料（病歷資料）中了。因此我們應該時刻檢查，訓練資料中的特徵能否在實際場景中取得，並以相同的形式出現。

6-1-4選擇測量成效的方法

想要操控某些事物，我們必須要能夠觀察它。要在專案上取得成功，就必須先定義何謂「成功」？是準確度？還是精準度或故障召回率？又或者是客戶回流率？成功的評量指標會引導專案中的所有技術選擇。它應該直接與更高層次的目標保持一致，例如公司的營運成功與否。

對於平衡的分類問題（編註：每個類別的樣本數差異不大），每個類別具備同等的偏好度，這時準確度和 ROC （receiver operating characteristic）曲線下面積（簡稱為 ROC AUC）是常用的評量指標。

對於類別不平衡的問題，我們可以使用精準度和召回率做為評量指標。對於排名問題或多標籤分類問題，我們可以使用加權形式的準確度做為評量指標。

很多時候，我們也必須自行定義指標來評量任務成功與否。為了瞭解機器學習成功指標的多樣性，以及它們與不同問題領域的關係，瀏覽 Kaggle上的資料科學競賽是很有幫助的（https: //kaggle. com），其中的案例展示了各種領域的問題和評量指標。

6-2 開發模型

一旦搞懂如何評估模型表現後，就可以進入開發模型的階段。

大部分的教學與研究專案都認為，開發模型是整個專案中的唯一步驟：忽略了定義問題與收集資料的環節，並假設這些事情已事先做完了；而且也不談模型的部署與維護，並假設會有其他人負責這一塊。

實際上，開發模型只是機器學習工作流程中的其中一步，而且還不是最難的部分。在機器學習中，最難的部分是定義問題範圍以及收集、標註並清理資料。

6-2-1準備資料

我們先前學過，深度學習模型通常無法接受原始資料，而需要先經過預處理。

資料預處理的目的是讓手邊的原始資料更適合模型處理，其中的技巧包括向量化、正規化或處理缺失值。

大部分預處理技巧都只適合用在特定的領域（例如特別為文字資料或影像資料而設計），我們將在接下來的章節中以個別案例來說明。現在，先來看一些常見於所有資料領域的基礎技巧。

向量化

神經網路的所有輸入和目標值必須是浮點數張量（在特定情況下可以是整數或字串張量）。無論是處理什麼樣的資料（如：聲音、影像、文字等），都必須先將它們轉換成張量，這個步驟稱為資料向量化。

例如，在之前的電影評論和新聞主題兩個文字分類範例中，一開始我們就以整數串列來代表單字序列，並使用 one-hot 編碼將它們轉換成 float32 的張量。在分類 MINIST數字和預測房價的範例中，資料已經被事先向量化了，因此可以跳過此步驟。

數值正規化

在 MNIST 數字分類的範例中，影像資料原本被編碼成 0-255 的整數，用來表示其灰階值。在把這些資料輸入神經網路前，必須先將其型別轉換為 float32 並除以255， 這樣才能得到介於 0~1之間的浮點數值。同樣地，在預測房價時，我們從具有各種度量範圍的特徵開始。

某些特徵是很小的浮點數（例如犯罪率），而某些特徵是相當大的整數值（例如年齡）。在把這些資料輸入神經網路之前，必須逐項把每個特徵正規化，使其標準差為 1（編註：就是用標準差為量測單位），平均值為0（就是把分佈曲線平移到以0為中心點）。

一般來說，將相對較大的數值或異質資料（例如：資料中的某個特徵值介於 0~1，而另一個特徵值則介於100~200）輸入神經網路並不安全。兩個特徵的範圍差距太大，將不利於神經網路收斂（會觸發很大的梯度更新）。為了使神經網路更容易學習，我們的資料應具有以下特性：

* 數值較小：大部分數值應介於 0~1 的範圍內。
* 具備同質性：所有特徵都應該採用大致相同的數值範圍。

除此之外，以下更嚴謹的正規化規則很常見也很有用，但並非絕對必要（例如，在數字影像分類範例中就不需要使用）：

* 單獨正規化每個特徵，使平均值為0。
* 單獨正規化每個特徵，使標準差為1。

處理缺失值

有時候，資料中會有缺失值。例如在房價範例中，首個特徵（資料中索引為0的直行）是人均犯罪率。如果不是所有樣本都有這個特徵怎麼辦？這樣一來，訓練或測試資料中就會有缺失值了。

我們可以選擇直接忽略這個特徵，但實際上還有其它的選項。

- 如果這個特徵是分類特徵，就可以為該特徵新增一個分類值，用來代表缺失值（編註：然後再將該特徵中的缺失值都改成新增的分類值）。模型會自動學習到如何將這個分類值對應到目標值。

- 如果這個特徵是數值特徵，應避免以隨便數字（例如：0）來代表缺失值，因為這可能會使特徵形成的潛在空間 （latent space）中出現不連續，導致導致訓練出來的模型很難具有好的普適化能力。如果要取代缺失值、可以使用該特徵的平均值或中位數。又或者，我們也可以另外再訓練一個模型，並根據其他特徵值來預測缺失的特徵值。

請注意！如果已經知道測試集（編註：或未來會强到的資料）中有缺失值，但訓練集中沒有缺失值，則神經網路將無法學會忽略缺失值！在這種情況下，我們應該人工為訓練模本製造一些缺失值，做法是多次複製數個訓練樣本，並從中刪除對應特徵的值（即刪除那些預期在測試集或未來遇到的資料中，會有缺失值的特徵的值）。

6-2-2 選擇驗證機制

先前提過，模型的最終目標是要實現普適化，而在整個模型開發的過程中，每個決定都是由驗證評量指標（例如模型對驗證集的準確度）所引導，其可用來衡量普適化的表現。

而選擇驗證機制的目的，就是希望我們的評量指標在未來的實際運作環境中，也能有很好的表現。

因此，驗證機制決定了我們是否能建構出一個有用的模型。

在第5章中，我們介紹了3種常見的驗證機制：

＊　拆分驗證集 （holdout validation set）：擁有大量資料時，這個方法最簡單。

＊　K折交叉驗證 （K-fold cross validation）：如果樣本數不夠多，這就是合適的選擇，可以確保驗證的可靠性。

＊　多次迭代的 K 折驗證(Iterated K-fold validation）：當資料很少時，這樣做可以非常準確地評估模型。

在大部分狀況下，第一種機制就夠用了。不過先前提過，我們一定要時刻留意驗證集的代表性，而且別讓訓練集與驗證集中出現重複的樣本。

6-2-3 超越基準線

當我們開始處理模型時，第一個目標就是要取得統計能力：也就是如第5章所示，開發一個能夠打敗基準線的小模型。

在這個階段，我們要專注在3件重要的事情上：

* 特徵工程：過濾掉那些不含有用資訊的特徵（也就是做特徵選擇），然後根據自己對於問題的理解，找出可能有用的新特徵。
* 選擇合適的既有架構：我們要使用何種模型架構？是密集連接網路、卷積網路、循環神經網路還是 Transformer 模型？深度學習是解決當前任務的好辦法嗎，還是我們該試試其他方法？
* 選擇足夠好的訓練配置：我們應該選擇什麼損失函數？批次量和學習率要多大？

選擇合適的損失函數

一般來說，不太可能直接對評量指標（例如：準確度）進行優化。有時，也很難將評量指標轉化為損失函數；畢竟損失函數要能在小批次上計算（理想中，即使只給定單一的資料點，也應該要能算出損失函數值），而且還必須可微分（否則就無法使用反向傳播來訓練模型）。

舉例來說，廣泛使用的分類任務評量指標 ROC/AUC 就沒辦法直接用來優化模型。因此在分類任務中，時常會用其它指標來代替ROC/AUC（例如：交叉熵）以進行優化。一般來

說，如果交叉熵越低，ROC/AUC就會越高。

下表可協助我們針對一些常見的問題類型，決定輸出層的激活函

數與損失函數：為模型輸出層選擇激活函數與損失函數

問題類型	輸出層激活函數	損失函數
二元分類	sigmoid	binary_crossentropy
多類別、單標籤分類	softmax	categorical_crossentropy
多類別、多標籤分類	sigmoid	binary_crossentropy

對於多數問題來說，都有現成的範本可供參考。我們一定不是最先嘗試開發垃圾內容偵測器、音樂推薦引擎或影像分類器的人。

請務必花點時間研究他人先前的成果，瞭解有哪些特徵工程的技巧及模型架構可以應用在自己的任務上。

請注意，有時候我們未必能成功取得統計能力。如果你已經嘗試各種合理的架構，但始終無法超越基準線，那就可能是輸入資料中不包含當前問題的答案。別忘了，我們先前做出了兩點假設：

* 我們假設根據特定的輸入，將可以預測出正確的結果。
* 我們假設現有的資料已具備足夠資訊，可以用來學習輸入與輸出之間的關係。

這些假設也有可能根本不成立，如果真是如此。那麼我們得從頭開始收集新的資料。

6-2-4 擴大規模：開發一個會過度配適的模型

一旦我們獲得了具有統計能力（表現超過基準線）的模型，緊接著的問題便是：我們的模型是否足夠強大？它是否有足夠的神經層和參數來正確擬合手上的問題？

例如，只有單隱藏層的模型或許有辨識 MINIST 數字的統計能力，但不足以很好地解決更複雜的問題。請記住，機器學習是在優化和普適化之間做取捨。

理想的模型是位於低度配適和過度配適的交界處、模型太小與模型太大之間。要找出這個邊界的位置，我們勢必要先越過它。

為了弄清楚我們需要多大的模型，就必須先開發一個會過度配適的模型。該過程並不困難，可以如第5章所學：

1. 添加更多的神經層。
2. 讓每一神經層更寬。
3. 訓練更多週期。

持續的監控訓練損失和驗證損失，以及關注任何我們重視的評量指標。當看到模型在驗證資料上的表現開始下降時，就是發生過度配適了。

6-2-5 將模型常規化並調整超參數

當模型的表現超過基準線，而且有能力過度配適後，我們的下一目標便是最大化其普適化能力。

這一步將佔用大量時間：我們會反覆修改模型、訓練它、使用驗證資料來評估（此時還沒用到測試資料）、再次修改它，然後不斷重複，直到模型表現不再進步為止。以下是我們該嘗試的做法：

* 嘗試不同的架構：添加或刪除神經層。
* 使用丟棄法。
* 如果模型不大，可嘗試使用L1或L2 常規化（也可同時使用）。
* 嘗試不同的超參數（例如每層的神經單元數或優化器的學習率）以找到最佳配置。
* 嘗試使用資料篩選或特徵工程：收集和標註更多資料、找出更好的新特徵，或刪除似乎沒有用（無法提供有效資訊）的特徵。

以上工作可透過自動化的超參數調整軟件（例如：KerasTuner）來進行，我們會在第13 章進行更多說明。

請注意！每次使用驗證集的回饋資訊來調整模型時，都會將「與驗證集有關的資訊」洩漏到模型中。

僅重複數次倒是無妨，但若是系統化地經過多次迭代，則最終會使模型過度配適於驗證資料（即使沒有直接以任何驗證資料去訓練模型），進而導致驗證的結果不可信。

一旦找出令人滿意的模型配置，就可以重新用訓練集和驗證集來訓練最終的成品模型，並用測試集做最後一次評估。

如果測試集上的表現明顯差於驗證資料上的表現，則可能代表我們的驗證過程有問題，或者是在調整參數的過程中，模型開始對驗證資料產生過度配適。在這種情況下，我們可嘗試切換到更可靠的評估（驗證）機制（例如多次迭代的 K折驗證）。

6-3 部署模型

現在，我們的模型已經通過測試集的最終評估，並做好部署的準備，可以正式在生產環境中運作了。

6-3-1 向客戶說明成果，並建立合理的期待

所謂的開發成功與客戶信任，都來自於滿足甚至超越人們的期待。就算我們交付了系統，也不過才完成一半的工作；另一半的工作則是在系統啓動前，先讓各戶建立合理的期待。

外行人對於 AI系統的期待通常過於理想化。例如，他們會期待這個AI 系統可以「瞭解」特定問題，而且和人類一樣，有著對特定問題的常識。

因此，我們也許要向客戶展示系統在什麼狀況下會失效（例如展示容易被錯誤分類的樣本，特別是那些會讓人覺得意外的分類錯誤）。

客戶通常也期待 AI 系統可以達到和人類一樣的水準，特別是在那些原本就是由人類處理的工作。

絕大部分的機器學習模型無法滿足這個要求，因為它們是被訓練來輸出人類所賦予的標籤（正確答案），只能盡量逼近人類的表現。因此，要很清楚地說明模型能夠輸出什麼結果，而不是使用很抽象的說明，例如：

「這個模型的準確度為98%」就過於抽象，而且大部分人會自動進位到 100%。

此時最好能提出更明確的資料，例如：偽陰性率及偽陽性率。我們可以這樣說：「在現有設定下，詐騙案件偵測模型會有5%的偽陰性率、2.5%的偽陽性率。

每天平均會發現300起疑似詐騙的案件，並將進一步送交人工審核。另外，每天平均會漏掉14起詐騙案件。平均來說，會正確捕捉到 266 起詐騙案件。」

換句話說，要將模型表現明確地與商務目標連接起來。

我們還應該和客戶及相關人士確認重要參數，例如：判斷是否標示為詐騙行為的閾值（不同閾值會導致不同的偽陰性率和偽陽性率）。這些涉及取捨的決策，都應該讓真正瞭解商業脈絡的專業人士來決定。

6-3-2 交付推論模型

機器學習專案何時完成？絕不是在 Colab 記事本上把訓練好的模型存檔的那一刻。畢竟最後送上生產環境的模型，很少會和我們在訓練過程中所操作的模型一模一樣。

首先，我們可能需要將模型匯出至 Python 以外的環境：

* 我們的生產環境（例如：移動裝置或嵌入式系統）未必支援 Python。
* 應用程式的其餘部分未必是以 Python 運行（可能是 JavaScript 或C++等）。

因此，用 Python 交付模型可能會明顯增加運行時的負擔。

其次，由於投入生產環境的模型只會用來輸出預測結果（該階段稱作「推論」），而不用進行訓練，因此我們還可以特別針對模型的預測功能進行各種優化，進而提升模型預測時的運行速度並降低記憶體耗用量。

以REST API部署模型

將模型轉換為產品的常見方式，是在伺服器或雲端虛擬機器上安裝Tensorflow，然後應用程式透過網路以 REST API 來呼叫模型並取得預測結果。

在實務上，我們可以利用 Flask（或其他 Python網頁開發函式庫）來親手打造自己的伺服應用網站，或是利用 TensorFlow 自身的函式庫（TensorFlow Serving） 將模型輸出成可以直接運行的API 網站程式。有了 Tensorflow Serving，我們可以在幾分鐘內部署 Keras 模型。

在下列狀況中，可以考慮以 REST API 部署模型：

* 需取得模型預測結果的應用程式可以穩定地連上網路。舉例來說，如果我們的應用程式是手機 app，需透過網路從遠端 API 取得模型預測結果，則若手機處於飛航模式或網路連線不穩定時，這個應用程式就無法使用。

* 應用程式對延遲時間的要求不高：一般來說，傳送請求、模型推論、與傳回答案的時間總共需要 500毫秒左右。

* 用以進行推論的輸入資料不敏感（不及隱私性）；由於資料需要經過模型處理，因此必須以解密過的形式存在於伺服器上（但記得，你務必要使用 SSL 加密方式來傳送 HTTP 的請求與回應）。

先前介紹過的影像搜尋引擎、音樂推薦系統、信用卡詐騙偵測專案，以及衛星影像分析系統都很適合透過 REST API 來部署。

如果要使用 REST API 來部署模型，需要先考慮如何搭建服務器的問題：要麼是自己格建，要麼就是使用第三方雲端服務。

舉例來說，Google 提供的 Cloud AI Platform讓使用者可以輕鬆將 Tensorflow 模型上傳到 Google 的雲端存儲空間 （Google Cloud Storage, GCS）。

Google會提供一個 API接入點，讓使用者可以直接取用，而無需顧慮許多實作細節（例如：批次預測、負載平衡、規模擴等）。

在裝置上部署模型

有時，我們需要在運行應用程式的同一裝置上運作模型，這些裝置可能是智慧型手機、嵌入了ARM CPU 的機器人、或是小型裝置上的微控制器。你或許看過可以自動偵測人臉的相機：其中可能就有直接在相機上運作的小型深度學習模型。

在下列狀況中，可以考慮直接在裝置上部署模型：

* 模型有嚴格的延遲時間限制，同時要能夠在網路連接不穩定的環境中運行。如果我們要建構的是沉浸式 AR 應用，那麼在遠端伺服器運作的模型就不太可行（編註：因為必須及時反應，不允許任何延遲）。

* 模型可以設計得非常小，以至於能夠在目標裝置的記憶體和功率限制下運作。 相關的設計方法，可以參考 Tensorflow Model Optimization Toolkit。

* 首要任務並非取得盡可能高的準確度。我們總要在運算時間效率與準確度之間有所取捨，在記憶體與功率等都受限的模型，其表現不太可能會像在大型GPU 上運行的模型那麼好。

* 輸入資料十分敏感（隱私要求很高），因此在遠端伺服器上不應該是可解密的。

先前提到的垃圾內容偵測模型就需要部署在使用者的手機上，因為聊天記錄經過了點對點加密，無法被遠端伺服器所讀取。

另外，異常餅乾偵測模型對延遲時間的要求很高，因此需要直接在工廠內的裝置上運作。幸運的是，在異常餅乾偵測的案例中，我們並沒有任何功率或記憶體空間的限制，因此可以直接在GPU上運作模型。

如果想在智慧型手機或嵌入式裝置上部署 Keras 模型，我們可以使用Tensorflow Lite。

該框架讓我們得以在 Android 和iOS 智慧型手機、ARM-64 電腦、Raspberry Pi 或微控制器上，有效率地進行深度學習推論。它包含了一個轉換器，可以直接將 Keras 模型轉換成TensorflowLite 的格式。

在瀏覽器上部署模型

深度學習經常用在瀏覽器或桌面應用的JavaScript 程式中。儘管經常可以看到以 REST API 連線的深度學習應用，不過直接在瀏覽器或電腦程式上運作模型，還是有很多好處的（例如可以使用電腦的GPU 資源）。

在下列狀況中，可以考慮在瀏覽器上部署模型：

* 我們想把運算成本轉移給使用者，進而大幅降低伺服器成本。

* 輸入資料必須留在使用者的電腦或手機。例如在垃圾內容偵測專案中，無論是使用網頁版還是桌面版的聊天軟體，都應該在本地端運行模型。

* 我們的應用有嚴格的延遲時間限制。在使用者自己的手機或電腦上進行運算，速度一定比不上配備了大型 GPU 的伺服器，但我們可以省下收發網路封包的時間。

* 在下載模型並緩存後，我們希望應用程式在不連線的狀況下還能持續運作。

只有模型夠小，我們才可以採取這個方式，免得模型運算時完全佔用電腦或手機的 CPU、GPU 或 RAM。另外，由於模型會被下載至使用者的裝置，因此最好確保模型中沒有什麼敏感資料。請務必記得，只要是訓練完成的模型，就一定可以從中挖出一些訓練資料的資訊。因此，使用敏感資料來訓練的模型最好不要公開。

如果要以 JavaScript 部署模型，Tensorflow生態系也提供了名為Tensorfow.js的工具。他是用來實現深度學習功能的 JavaScript 函式庫（原先叫做WebKeras），提供了幾乎所有 Keras API 的功能，也包含許多底層的 TensorFlow API。我們可以輕易地將Keras 模型匯入 Tensorflow.js，以將其作為瀏覽器或桌面程式中的 JavaSeript API 來進行查詢。

優化推論模型

當部署在記憶體與功率有所限制的環境（如智慧型手機和嵌入式裝置），或者是應用程式有低延遲的需求時，優化模型來進行推論就顯得十分重要了。在將模型匯入到 TensorFlow.js 或匯出至 TensorFlow Lite 之前，我們都應該先對模型進行優化。

我們可以採用兩種熱門的優化技巧：

* 權重剪枝 （Weight pruning）：並非每個模型參數都對預測結果有同樣的貢獻。 我們可以減少模型中的參數數量，只留下最重要的那一些。這樣可以降低模型的運算成本，而且只會讓表現變差一些些。我們可以自己決定剪枝的比例，並在模型大小與準確度之間尋找平衡點。

* 權重量化 （Weight quantization）：在訓練時，深度學習模型的權重值是單精度浮點數 。不過在進行推論時，可以將模型權重量化成8 位元整數 ，這樣便可以將模型規模縮小至原先的四分之一，但準確度仍會維持在接近原先的水準。 TensorFlow 生態系中已經準備好剪枝與量化的工具，並且也整合到 Keras API 中了

6-3-3監控模型運作狀況

至此，我們已經匯出一個推論模型，也已經把模型整合到應用中，還在實際進作的資料上測試過：模型表現就跟預期的一樣。我們也寫了一些單元測試，還有記錄運行過程與監控狀態的程式碼，一切都很完美！接下來，可以準備正式將模型部署至生產環境了！

不過事情還沒結束，就算部署了模型，還是要持續監控模型的行為、掌握模型在新資料上的表現、觀察模型與應用程式其餘部分的互動，以及模型最終如何影響商業評量指標。

* 在部署音樂推薦系統之後，聽衆的參與程度變高還是變低了？導入新的點擊率預測模型後，廣告的點擊率上升了嗎？這裡可以考慮引入 A/B 測試，獨立觀察模型效果：以新模型來運行一部分資料，其餘部分則以舊流程進行。一旦處理了夠多的案例，兩者之間的差異就很可能是由模型所造成的。

* 如果可能，請對模型在實際運作的資料上的預測結果進行人工審查。我們通常可以使用先前的工具來做資料標註：先拿一部分實際運作的資料進行人工標註，然後和模型的預測結果進行比較。這種做法適用於圖片搜尋引擎與異常餅乾偵測系統的專案中。

* 如果無法進行人工審查，也可以試著採取像是使用者調查的替代評估方案（比如在垃圾與惡意內容的偵測系統中，直接詢問使用者的用戶體驗）。

6-3-4維護模型

最後要提醒的是，模型不會永遠都表現良好。我們先前談過「概念漂移」：

隨著時間演變，實際運作的資料特性會不斷改變，導致模型的表現越來越差。

以音樂推薦系統來說，其生命週期可能只有數個星期（編註：流行音樂的榜單變動得很快）；以信用卡詐騙偵測系統來說，其生命週期可能只有幾天（編註：詐騙集團發現當前的手段行不通後，可能就會馬上想出新的詐騙伎倆）；以影像搜尋引擎來說，最佳狀況下也只能使用數年的時間。

一旦正式啟用模型，我們就該準備訓練下一代模型了。因此，我們要：

* 時刻關注實際運作的資料中的變動。是否出現了新特徵？我們是否要進行擴充？或是修正原有的標籤？

* 持續收集與標註資料，並且不斷改進標註過程。具體來說，我們要專注在收集那些現有模型很難正確分類的樣本，因為這一類樣本最有助於優化模型。

本章小結

開始新的機器學習專案時，一定要先釐清當前的問題：

最終目標是什麼？有哪些限制？

* 收集與標註資料集；確保已經深入瞭解這些資料的本質。 要如何評估結果是否成功：使用什麼評量指標來監控模型在驗證資料上的表現？

當我們瞭解問題，也收集到合適的資料集後，便可以開始動手開發模型：

* 準備資料。

* 確認驗證機制：拆分驗證？K折驗證？要取用資料集的哪個區塊來做驗證？

實現統計能力：打敗基準線。

* 擴大規模：開發能夠過度配適的模型。 根據模型在驗證資料上的表現，對模型進行常規化並調整超參數。大部分的機器學習研究都只專注在這一步驟，但要時刻記得機器學習的完整輪廓。

當模型在測試資料上的表現不錯時，就可以進入部署的階段了：

* 首先，確認客戶的期待是合理適當的。

* 優化用來進行推論的最終模型，並根據所選擇的部署環境（網路伺服器、移動裝置、瀏覽器、嵌入式裝置等）部署模型。

在投入生產環境運作後，持續監控模型表現並收集新資料，以便開發下一代的模型