# 前言

歡迎閱讀《從程式員到 *AI* 專家》，我在好幾年前就想要寫這本書了，但是直到機器學習（ML）和（尤其是）TensorFlow 最近的進展，我才能將這個想法付諸實踐。本書的目的是幫你這位程式員做好準備，讓你有能力面對許多機器學習場景，你不需要取得博士學位就可以成為一位 ML 和 AI 開發者！希望你以後會覺得這是一本有用的書，它將讓你充滿信心，開始這趟美妙且滿載而歸的旅程。

## 誰該閱讀本書？

如果你對 AI 和 ML 有興趣，而且想要快速地建立一個能夠從資料中學習的模型，那麼這本書就很適合你。如果你想要從常見的 AI 和 ML 概念學起（電腦視覺、自然語言處理、序列建模…等），而且想要知道如何訓練神經模型來解決這些領域的問題，我認為你會喜歡這本書。如果你已經訓練過模型，而且想要透過手機、瀏覽器或雲端來讓用戶使用，本書也是為你而寫的！

最重要的是，如果你因為覺得困難，而一直猶豫是否該踏入這個有價值的電腦科學領域，尤其是你認為你必須重拾塵封已久的微積分書本，別怕，本書採取程式優先的做法，讓你知道使用 Python 和 TensorFlow 來踏入機器學習和人工智慧的大門有多麼簡單。

## 著作動機

我是在 1992 年春天開始認真地從事人工智慧的，當時我剛取得物理學學位，住在倫敦，糟遇可怕的經濟困境，有 6 個月找不到工作。那時英國政府啟動一項專案，準備訓練 20 位 AI 技術人才，開始接受應徵申請。我是第一個被錄取的，經過三個月之後，這項專案以失敗告終，因為儘管在**理論**上，AI 可以完成大量的工作，我們卻無法用簡單的方法來**實踐**它。當時，我可以用 Prolog 語言來撰寫簡單的推理程式，並且用 Lisp 語言來執行串列處理，但沒有人知道如何在業界部署它們，然後，我進入著名的「AI 寒冬」。

2016 年，當我在 Google 參與 Firebase 專案時，Google 為所有工程師提供了機器學習訓練，當時我們一起坐在一間房間裡，聽著關於微積分和梯度下降的講座，我根本無法把那些東西與 ML 的實作聯繫起來，這讓我想到 1992 年的場景，我把這個感受，以及我們應該如何教育 ML 人才的想法回饋給 TensorFlow 團隊，他們在 2017 年錄取我。隨著 TensorFlow 2.0 在 2018 年發表，特別是它的重心是可以讓開發者更容易入門的高階 API，我認為必須用一本書來教大眾利用它使用 ML，免得只有數學家或博士能夠使用 ML。

我相信，有越多人使用這項技術，並且部署它來讓終端用戶使用，就會大幅增加 AI 與 ML 的數量，從而避免另一次 AI 寒冬的降臨，並且讓世界更加美好。我已經看到它的影響了，包括 Google 的糖尿病視網膜病變研究、賓洲洲立大學與 PlantVillage 建構了一個行動 ML 模型來協助農夫診斷木薯病、無國界醫生組織使用 TensorFlow 模型來協助診斷抗生素抗藥性，族繁不及備載！

## 本書簡介

本書有兩大部分。第一部分（第 1–11 章）討論如何在各種場景之下使用 TensorFlow 來建構機器學習模型，它將帶你從第一原理（用只有一個神經元的神經網路來建構模型）經歷電腦視覺、自然語言處理，以及序列建模。接下來的第二部分（第 12–20 章）將協助你將模型植入別人手裡的 Android 和 iOS、用 JavaScript 植到瀏覽器，以及透過雲端提供服務。大多數的章節都是互相獨立的，所以你可以跳到各章來學習新知識，當然，你也可以從頭到尾看完這本書。

## 你需要了解的技術

本書前半部分的目標是協助你學習如何使用 TensorFlow 來以各種架構建立模型，唯一的先決條件是了解 Python，尤其是 Python 的資料表示法和陣列處理法。你可能也要了解 NumPy，它是進行數值計算的 Python 程式庫。如果你不熟悉它們，它們都很容易學習，你應該可以在過程中認識它們（不過有些陣列標示法或許比較難以掌握）。

在第二部分中，我通常不會教導語言知識，而是展示如何在各種語言裡面使用 TensorFlow 模型。因此，舉例來說，在介紹 Android 的那一章（第 13 章），你將學會如何以 Kotlin 和 Android studio 來建構 app，在介紹 iOS 的那一章（第 14 章），你將學會用 Swift 和 Xcode 來建構 app。我不會教導這些語言的語法，所以如果你不熟悉它們，你可能需要閱讀入門教材，由 Jonathon Manning、Paris Buttfield-Addison 和 Tim Nugent 合著的 *Learning Swift*（O'Reilly）是很棒的選擇，繁體中文版《*Swift 學習手冊*》由碁峰資訊出版。

## 線上資源

本書使用和提供各種線上資源，建議你至少關注一下 TensorFlow（*https://www.tensorflow.org*）和它的 YouTube 頻道（*https://www.youtube.com/tensorflow*），來了解本書談到的技術有沒有任何更新和重大變化。

本書的程式碼位於 *https://github.com/lmoroney/tfbook*，我會隨著平台的演進而持續更新它。

## 本書編排慣例

本書使用下列的編排規則：

斜體字（*Italic*）

代表新術語、URL、email 地址、檔名，與副檔名。中文以楷體表示。

定寬字（`Constant width`）

在長程式中使用，或是在文章中代表變數、函式名稱、資料庫、資料型態、環境變數、陳述式、關鍵字等程式元素。

第一章

# TensorFlow 簡介

若要製作人工智慧（AI），機器學習（ML）與深度學習是很棒的起點，但是，在初期，一般人很容易被所有的選項和新術語搞得不知所措，本書希望為程式員揭開神秘的面紗，帶領你藉著編寫程式來實作機器學習和深度學習的概念，並且在電腦視覺、自然語言處理（NLP）及其他場景中，建立出展現類似人類行為的模型，因此，它們是一種合成的，或人工的智慧。

但是我們說的**機器學習**究竟是什麼東西？在深入討論之前，我們先從程式員的角度來了解它，然後，本章將告訴你如何安裝業界的工具，包括 TensorFlow 本身，以及寫程式和除錯 TensorFlow 模型的環境。

## 什麼是機器學習？

在了解 ML 的來龍去脈之前，我們先從傳統程式設計的角度來看看它是如何演變的。我們先討論什麼是傳統程式設計，然後想一下它在哪些情況之下能力有限。接下來，我們會了解為了處理這些案例，ML 是如何進化並且開啟實作新場景的新機會，從而解鎖許多人工智慧的概念。

傳統的程式設計需要編寫規則，用程式語言來表達規則，讓它處理資料並提供答案。幾乎所有可以用程式來編寫的東西都是如此。

例如，受歡迎的打磚塊遊戲用程式碼來決定球的移動方向、分數，以及贏得或輸掉遊戲的各種條件，想一下球從磚塊反彈的情況，如圖 1-1 所示。

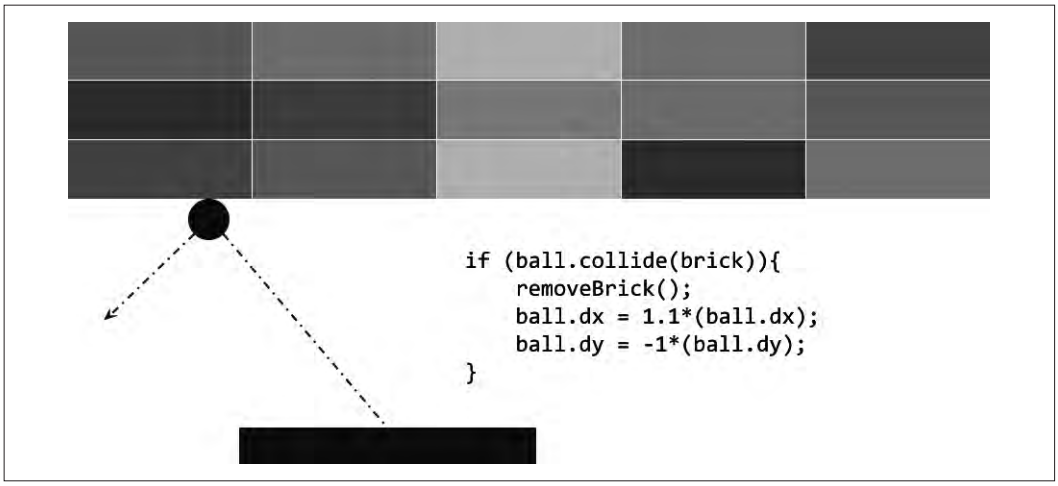


圖 1-1　打磚塊遊戲的程式

球的移動方向可以用它的 dx 與 dy 屬性來決定，當它打到磚塊時，磚塊會被移除，球的速度會增加，方向也會改變，這段程式根據與遊戲情況有關的資料採取行動。

舉另一個例子——金融服務場景。你有關於公司股票的資料，例如它的現價和目前盈餘，你可以用圖 1-2 的程式來計算一個稱為 P/E（市價除以每股盈餘）的比率。

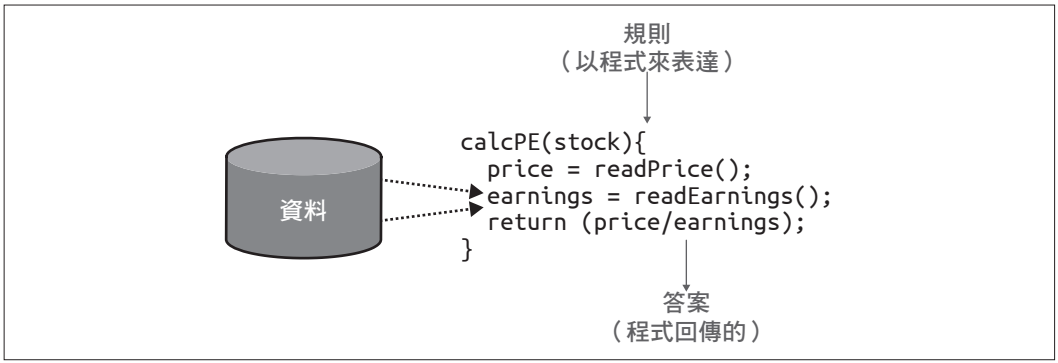


圖 1-2　金融服務場景的程式碼

你的程式會讀取價格，讀取盈餘，然後回傳一個前者除以後者的值。

如果我試著用一張圖表來總結這種傳統程式，它應該會類似圖 1-3。

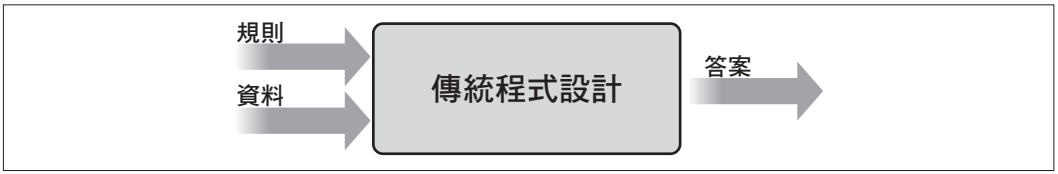


圖 1-3　傳統程式設計的高階視角

如你所見，我們用程式語言來描述規則，用這些規則來處理資料，結果就是答案。

## 傳統程式設計的限制

圖 1-3 的模式從最初就是開發的主幹，但是它有一個固有的限制：只能處理可以找出規則的場景。其他的場景呢？通常因為程式會過於複雜，所以不可能開發出來，根本無法用程式來處理。

例如，考慮偵測活動的情況。可以偵測活動的健康監視器是最近出現的一項創新，之所以直到最近才出現，不僅因為它價格低廉、體積很小，也因為進行偵測的演算法以前根本寫不出來，我們來研究一下原因。

圖 1-4 是簡單的偵測走路的活動演算法，透過人的移動速度來偵測，如果它少於某個值，我們就可以認為他們應該在走路。

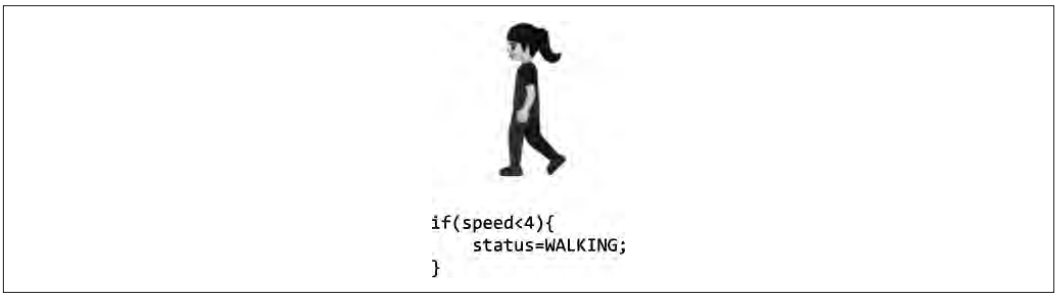


圖 1-4　偵測活動的演算法

因為我們的資料是速度，我們可以擴展程式，來偵測他們是不是在跑步（圖 1-5）。


圖 1-5　擴展演算法來偵測跑步

如你所見，我們可以藉著判斷速度低於某個值（假如是 4 英里）來認出用戶在走路，否則他就是在跑步。這在某方面來說是可行的。

假設我們要將這段程式擴展成另一種流行的健身運動——騎單車，演算法如圖 1-6 所示。

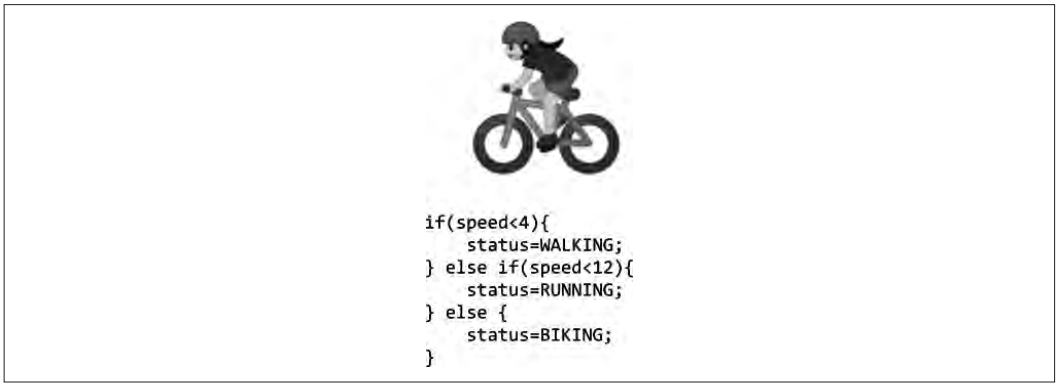


圖 1-6　擴展演算法來偵測騎單車

在這裡只偵測速度實在太草率了，舉例來說，有些人跑得比別人快，而且往下坡跑步可能比往上坡騎單車更快。但是整體來說，這種做法仍然可行。然而，如果我們想要實作另一個場景，例如打高爾夫球時，會怎樣（圖 1-7）？


圖 1-7　如果要編寫高爾夫演算法呢？

現在我們卡住了，我們該如何用這種方法來判斷用戶正在打高爾夫？那個人可能會走路一段時間，停下來，做一些動作，再走路一段時間，但是我們怎麼知道他在打高爾夫？

用傳統規則來偵測活動的做法遇到瓶頸了。但是或許有更好的方法。

也就是踏入機器學習大門。

## 從編寫程式到學習

回顧一下之前那張展示傳統程式設計的圖（圖 1-8），它用一些規則來處理資料，並產生答案。在活動偵測場景中，資料是人移動的速度，據此，我們可以編寫規則，來偵測他們的活動究竟是走路、騎單車，還是跑步。但是在偵測打高爾夫球時，我們遇到瓶頸，因為我們沒辦法用規則來判斷那個活動的樣貌。

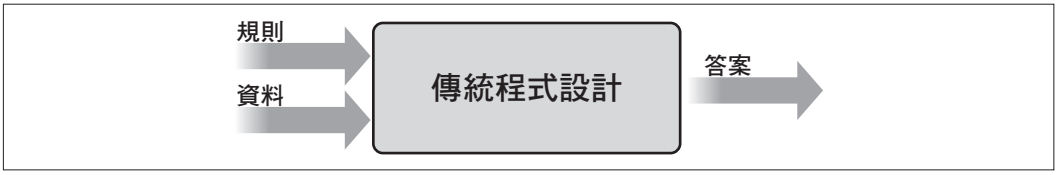


圖 1-8　傳統程式設計流程

但是，如果把這張圖左右對調呢？我們不找出**規則**，而是找出**答案**，並且結合資料來尋找規則，那會怎樣？

圖 1-9 就是這種做法，我們可以用這張高階的圖來定義**機器學習**。


圖 1-9　藉由左右對調來產生機器學習

那麼，這張圖是什麼意思？現在不是由我們試圖找出規則是什麼，而是取得許多關於場景的資料，標注那些資料，由電腦找出為一項資料指定某個標籤，為另一項資料指定另一個標籤的規則。

在活動偵測場景中，如何採取這種做法？我們可以用感應器來取得關於用戶的資料，如果讓他們使用穿戴式設備，該設備可以偵測心跳率、位置、速度等資訊，並且在他們進行各種活動時，收集這類的資料，我們就有代表「走路的樣子」、「跑步的樣子」等資料（圖 1-10）。

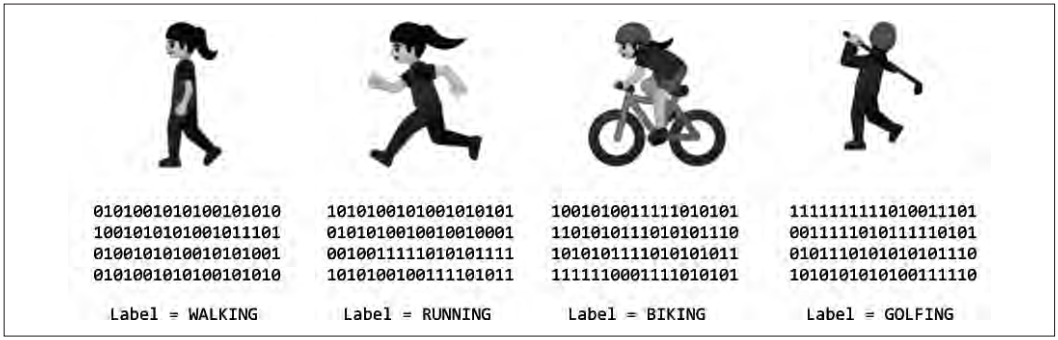


圖 1-10　從寫程式到 ML：收集和標注資料

現在程式員的工作從「找出規則」變成「確認活動」，以及寫出為資料指定標籤的程式。如果我們可以做到，我們就可以擴展原本以程式碼來實作的場景，機器學習就是做這件事的技術，但是我們需要一種框架才能開始工作，這就是為什麼要用 TensorFlow。下一節要介紹它是什麼，以及如何安裝它，在本章稍後，你將開始編寫你的第一個程式，用它來學習兩個值之間的模式，就像上述的場景那樣。它是個簡單的「Hello World」場景，但是它的基本設計模式與非常複雜的場景時所使用的一樣。

人工智慧領域既龐大且複雜，涵蓋讓電腦像人類一樣思考和行動的所有事物。人類會透過範例來學習新行為，因此，機器學習可以視為人工智慧開發的入口，透過它，機器可以學會像人類一樣看東西（一種稱為**電腦視覺**的領域）、讀取文本（text）（自然語言處理），以及其他能力。本書將使用 TensorFlow 框架來討論機器學習的基本概念。

# 什麼是 TensorFlow？

TensorFlow 是一個開源平台，可讓你建立機器學習模型和使用它。它實作了許多機器學習常用的演算法和模式，讓你不必學習底層的數學和邏輯即可直接專注於你的場景。它希望讓所有人使用，包括業餘愛好者、專業開發者，以及將人工智慧的界限往外擴展的研究員。更重要的是，它也可以讓你將模型部署到 web、雲端、行動設備，及嵌入式系統。本書將探討以上所有的場景。

圖 1-11 是 TensorFlow 的高階架構。

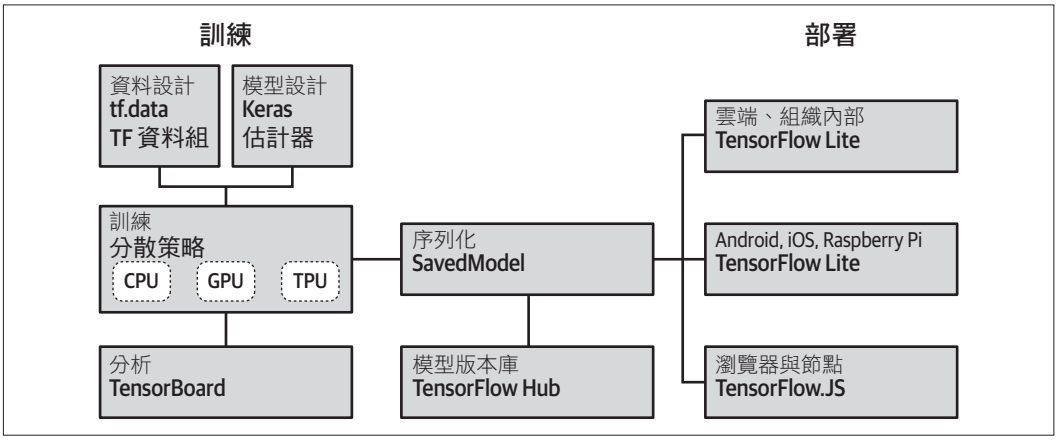


**圖 1-11　TensorFlow 的高階架構**

建立機器學習模型的程序稱為**訓練**，在這個程序中，電腦使用一組演算法來了解輸入，以及如何區分它們。因此，舉例來說，如果你想要讓電腦辨識貓與狗，你可以使用牠們的許多照片來建立一個模型，電腦會用那個模型來學習貓為什麼是貓，狗為什麼是狗。訓練好模型之後，用它來辨識未來的輸入或對輸入進行分類稱為**推理**（*inference*）。

為了訓練模型，你必須提供一些東西，第一種是一組用來設計模型的 API。在 TensorFlow 中，你可以採取三種主要的做法：你可以自己編寫所有的程式，自己想出如何讓電腦進行學習的邏輯，並且用程式來實作它（不建議）；你也可以使用內建的**估計器**（*estimator*），它們是現成的神經網路，而且你可以自訂它們；或者，你可以使用 Keras，Keras 是一種高階的 API，可讓你將常見的機器學習範式（paradigms）封裝在程式碼裡面。本書的重點是使用 Keras API 來建立模型。

訓練模型的方法有很多種，在多數情況下，你應該只有一顆晶片，可能是中央處理單元（CPU）、圖形處理單元（GPU），或是一種稱為**張量處理單元**（TPU）的新晶片。比較高級的工作環境與研究環境可能要在多顆晶片上執行平行訓練，採取**分散策略**，將訓練分給多顆晶片，TensorFlow 也支援這種技術。

資料是任何模型的命脈，如前所述，如果你想要製作一個可以辨識貓與狗的模型，你就要用許多貓與狗的樣本來訓練它。但是該如何管理這些樣本？你將會漸漸看到，做這件事需要編寫的程式碼通常比建立模型本身更多。TensorFlow 有一些簡化這個過程的 API，稱為 TensorFlow Data Services。它們有許多預先處理過的資料組可供學習，你可以用一行程式碼來使用它們，它們也有一些工具可以處理原始資料，讓它更容易使用。

除了建立模型之外，你也要在它們可以供人使用時，把它們送到人們的手裡。為此，TensorFlow 有一些伺服 API，你可以透過 HTTP 連結提供推理結果給雲端或 web 用戶。如果你打算在行動或嵌入式系統上運行模型，你可以使用 TensorFlow Lite，它提供一組工具，可以在 Android、iOS，以及 Raspberry Pi 之類的 Linux 嵌入式系統上進行模型推理。TensorFlow Lite 的分支 TensorFlow Lite Micro（TFLM）也可以透過一種稱為 TinyML 的新概念在微控制器上提供推理。最後，如果你想要讓瀏覽器或 Node.js 的用戶使用模型，TensorFlow.js 可讓你用這種方式來訓練和執行模型。

接下來，我們將告訴你如何安裝 TensorFlow，讓你可以開始用它來建立和使用 ML 模型。

## 使用 TensorFlow

這一節將介紹三種安裝和使用 TensorFlow 的方法。我們會先討論如何使用命令列來將它安裝到你的開發工具箱裡面，接下來討論如何使用流行的 PyCharm IDE（整合開發環境）來安裝 TensorFlow，最後介紹 Google Colab，以及如何在你的瀏覽器使用雲端後端來讀取 TensorFlow 程式碼。

## 在 Python 裡安裝 TensorFlow

TensorFlow 可讓你用多種語言來建立模型，包括 Python、Swift、Java…等，本書把重心放在 Python 上，由於 Python 廣泛支援數學模型，所以它是事實上的機器學習語言，如果你還沒有安裝它，我強烈建議你造訪 Python（*https://www.python.org*）來安裝並執行它，並且造訪 learnpython.org（*https://www.learnpython.org*）來學習 Python 語法。

使用 Python 時，安裝框架的方法有很多種，TensorFlow 團隊預設支援的是 `pip`。

因此，在 Python 環境安裝 TensorFlow 很簡單，只要使用：

```
pip install tensorflow
```

注意，從第 2.1 版開始，它預設安裝 GPU 版的 TensorFlow，在這個版本之前，它使用 CPU 版本。因此，在安裝前，請確保你有支援它的 GPU，及其所有必要的驅動程式，詳情見 TensorFlow（*https://oreil.ly/5upaL*）。

如果你沒有所需的 GPU 或驅動程式，你仍然可以在任何 Linux、PC 或 Mac 上，用這個命令來安裝 CPU 版的 TensorFlow：

```
pip install tensorflow-cpu
```

你可以在啟動並運行之後，用下列程式來測試你的 TensorFlow 版本：

```
import tensorflow as tf
print(tf.__version__)
```

你應該可以看到類似圖 1-12 所示的輸出。它會印出目前正在執行的 TensorFlow 版本，圖中安裝的是 2.0.0 版。

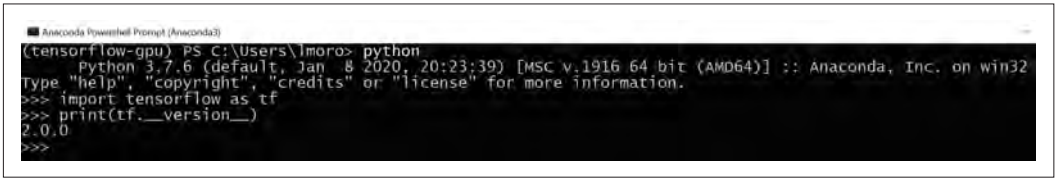


圖 1-12　在 Python 中執行 TensorFlow

## 在 PyCharm 中使用 TensorFlow

我特別喜歡用免費社群版的 PyCharm（*https://www.jetbrains.com/pycharm*）來以 TensorFlow 建構模型。PyCharm 好用的地方很多，我最喜歡的一種是它可讓我輕鬆地管理虛擬環境，這代表我可以在多個 Python 環境中使用專案專屬的工具版本，例如 TensorFlow，舉例來說，如果你想要在一項專案中使用 TensorFlow 2.0，在另一項專案中使用 TensorFlow 2.1，你可以用虛擬環境來分隔它們，如此一來，當你在這些專案之間切換時，不需要安裝 / 移除依賴項目。此外，使用 PyCharm 可以為 Python 程式進行步進除錯，這是必備的功能，尤其當你是新手時。

例如，在圖 1-13 中，有一個稱為 *example1* 的新專案，並且指定以 Conda 建立一個新環境。當我建立專案時，我會得到一個乾淨且全新的虛擬 Python 環境，可在裡面安裝任何想要的 TensorFlow 版本。

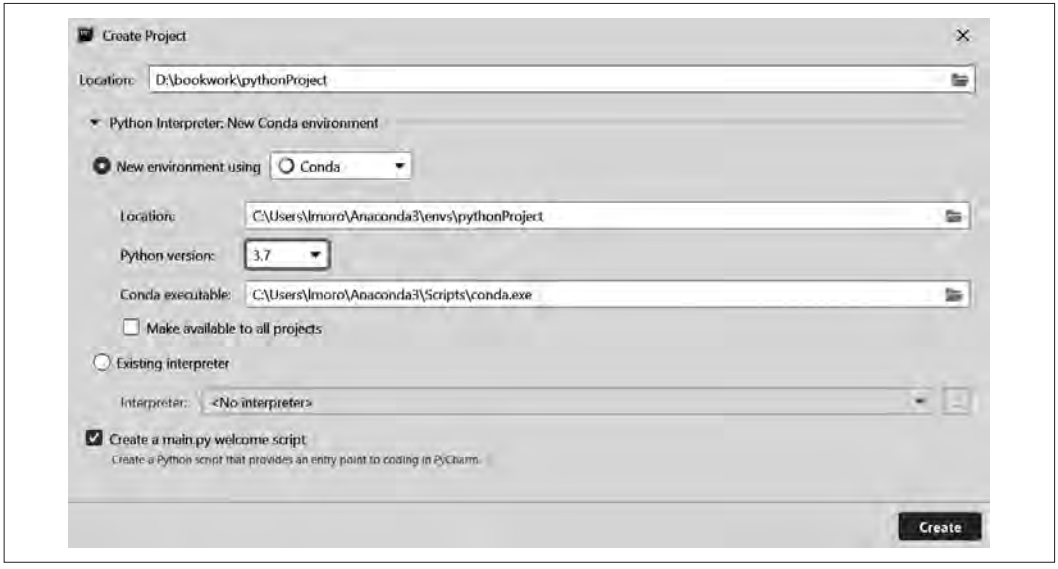


圖 1-13　用 PyCharm 建立新的虛擬環境

建立專案之後，你可以打開 File → Settings 對話框，在左邊的選單選擇「Project: < 你的專案名稱 >」，你會看到改變 Project Interpreter 與 Project Structure 的設定選項，選擇 Project Interpreter 會顯示你正在使用的 interpreter，以及一些已被安裝在這個虛擬環境裡面的程式包（圖 1-14）。

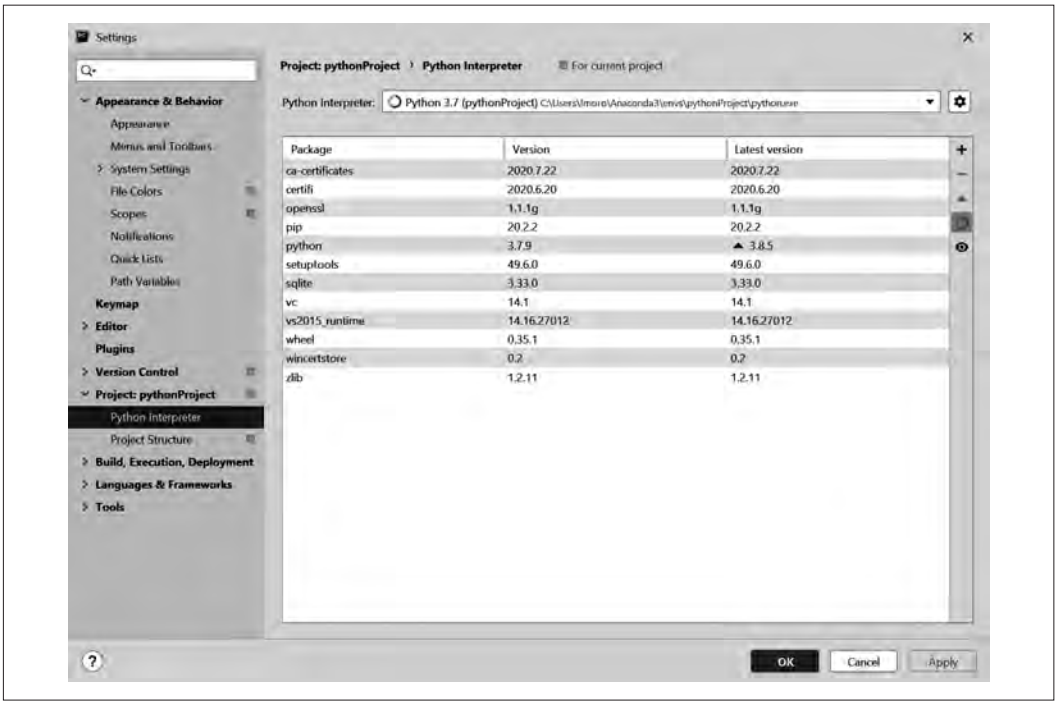


圖 1-14　將程式包加入虛擬環境

按下右邊的 + 按鈕會出現一個對話框，顯示目前可用的程式包。在搜尋框裡輸入「tensorflow」會顯示名稱有「tensorflow」的所有程式包（圖 1-15）。


圖 1-15　用 PyCharm 安裝 TensorFlow

選擇 TensorFlow 或你想要安裝的任何其他程式包之後，按下 Install Package 按鈕，PyCharm 將完成其餘的工作。

安裝 TensorFlow 之後，你就可以在 Python 裡面編寫 TensorFlow 程式碼和對它進行 debug 了。

## 在 Google Colab 裡使用 TensorFlow

另一個選項是使用 Google Colab（*https://colab.research.google.com*），這應該是最容易上手的，它是代管的 Python 環境，你可以透過瀏覽器來使用它。Colab 真正的優點在於它提供 GPU 和 TPU 後端，所以你可以免費使用最先進的硬體來訓練模型。

當你造訪 Colab 網站時，你會看到打開 Colabs 或啟動新 notebook 的選項，如圖 1-16 所示。

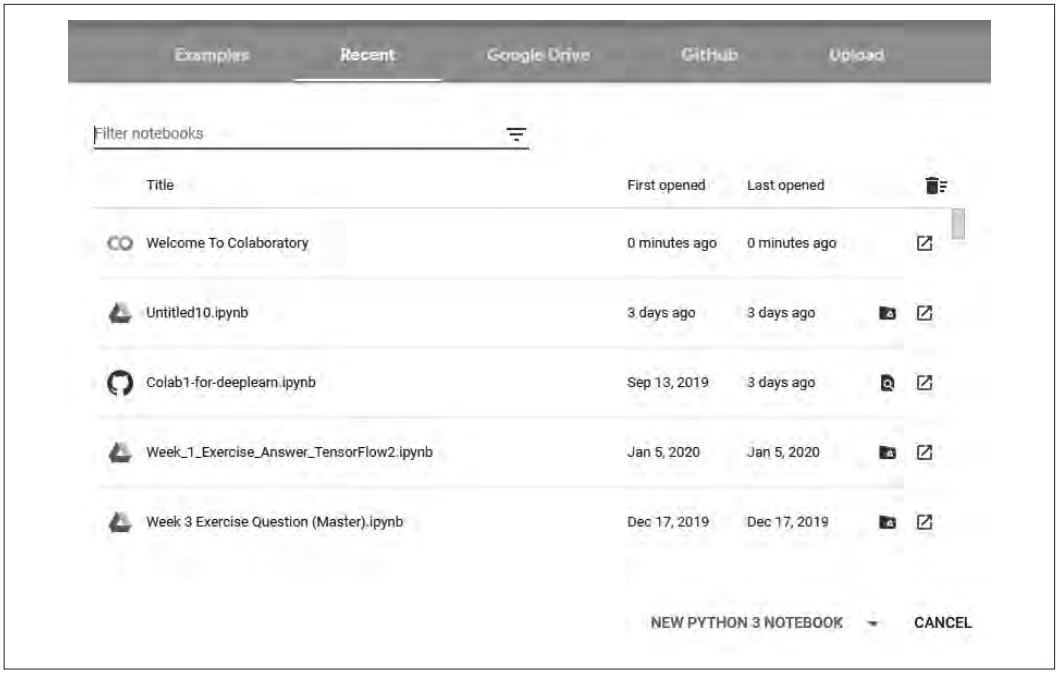


圖 1-16　使用 Google Colab 來工作

按下 New Python 3 Notebook 連結可開啟一個編輯器，你可以在那裡加入程式碼或文字的窗格（圖 1-17），你可以按下窗格左邊的 Play 按鈕（箭頭形狀）來執行程式碼。


圖 1-17　在 Colab 裡執行 TensorFlow 程式碼

像圖中那樣檢查 TensorFlow 版本是必要的，以確保你正在執行正確的版本。Colab 內建的 TensorFlow 通常會落後一到兩個版本，若是如此，你可以用 `pip install` 來更新它如下：

```
!pip install tensorflow==2.1.0
```

當你執行這個命令之後，Colab 環境就會使用你指定的 TensorFlow 版本。

## 開始進行機器學習

正如我們在本章稍早看到的，機器學習的範式就是取得資料，標注那些資料，並且找出幫資料和標籤配對的規則，若要用程式來展示這個範式，下面的例子應該是最簡單的場景。考慮兩組數字：

```
X = -1, 0, 1, 2, 3, 4
Y = -3, -1, 1, 3, 5, 7
```

X 與 Y 的值有某個關係（例如，當 X 是 –1 時，Y 是 –3，當 X 是 3 時，Y 是 5，以此類推），看得出這個關係嗎？

看了幾秒之後，或許你可以看到它的模式是 Y = 2X – 1，你是怎麼知道的？每個人會用不同的方式來找出它，但我通常聽到人們發現：X 在序列裡每次都加 1，Y 每次都加 2，所以 Y = 2X +/– 某個數字，接下來，他們發現當 X = 0 時，Y = –1，所以答案應該是 Y = 2X – 1。他們繼續觀察其他的數字，發現這個假設是「成立」的，所以答案是 Y = 2X – 1。

這個過程很像機器學習程序，我們來看一些透過 TensorFlow 來讓神經網路來幫你解決這個問題的程式碼。

下面是完整的程式，它使用 TensorFlow Keras API。不用擔心看不懂，我會逐行講解它們：

```
import tensorflow as tf
import numpy as np
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')

xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model.fit(xs, ys, epochs=500)

print(model.predict([10.0]))
```

我們從第一行開始看起。你應該聽過神經網路，而且應該看過以互相連接的幾層神經元來解釋它們的圖，類似圖 1-18。

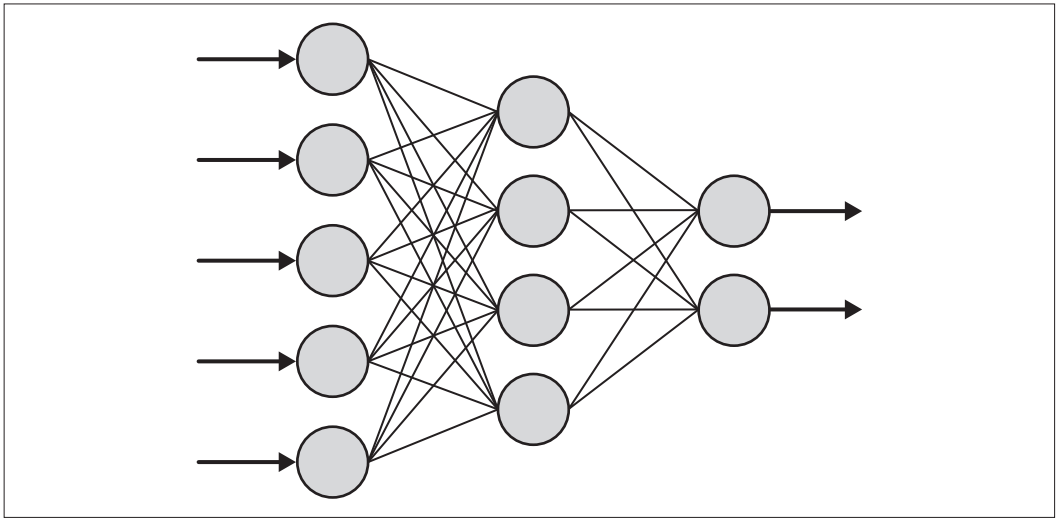

圖 1-18　典型的神經網路

當你看到這種神經網路時，可以將每一個「圓圈」視為一個**神經元**，每一排圓圈就是一個**階層**。所以，圖 1-18 有三層：第一層有五個神經元，第二層有四個，第三層有兩個。

回顧程式，從第一行可以看到，我們定義了最簡單的神經網路，它只有一層，而且裡面只有一個神經元：

```
model = Sequential([Dense(units=1, input_shape=[1])])
```

在使用 TensorFlow 時，你可以用 Sequential 來定義階層，在 Sequential 裡面，你可以設定每一層的樣子，在 Sequential 裡面只有一行，所以我們只有一層。

然後，你可以用 keras.layers API 來定義那一層的長相。神經層有許多不同的類型，我們在這裡使用 Dense 層。「Dense」代表一組完全（或稠密地（densely））互連的神經元，就像圖 1-18 那樣，每一個神經元都與下一層的每一個神經元連接，它是最常見的神經層類型。我們的 Dense 層使用 units=1，所以在整個神經網路裡，我們只有一個稠密層，裡面有一個神經元。最後，當我們在神經網路裡指定**第一層**時（在這個例子中，它是我們唯一的一層），我們必須告訴它輸入資料的外形（shape）是什麼，這個例子的輸入資料是 X，它只是一個值，所以我們指定這個外形。

從下一行開始很有趣，我們看一下：

```
model.compile(optimizer='sgd', loss='mean_squared_error')
```

如果你用機器學習做過任何事情，你應該知道它涉及許多數學，如果你有好幾年沒有碰微積分了，它看起來應該是入門的障礙，這一行就是開始使用數學的地方，它是機器學習的核心。

在這個場景中，電腦對於 X 與 Y 之間的關係**一點概念都沒有**，所以它會用猜的，假設它猜 Y = 10X + 10，接下來，它必須評估這次的猜測是好是壞，這就是**損失函數**的功能。

它已經知道當 X 是 –1、0、1、2、3 與 4 時的答案了，所以損失函數可以比較這些答案，來了解猜出來的關係對不對。如果它猜 Y = 10X + 10，那麼當 X 是 –1 時，Y 就是 0，正確的答案是 –3，所以有一點落差。但是當 X 是 4 時，猜出來的答案是 50，而正確的答案是 7，這就相去甚遠了。

掌握這個知識之後，電腦會猜另一次，這就是 *optimizer*（**優化函數**）的功能。此時就是大量使用微積分的時候。你可以為不同的場景選擇適合的優化函數，在這個例子裡，我們選擇 sgd，它是 *stochastic gradient descent*（**隨機梯度下降**）的縮寫，sgd 是一種複

雜的數學函數，當你將一個值（也就是之前的猜測），以及計算誤差（error，或損失，loss）的結果傳給它時，它可以產生另一個值。它的工作就是隨著時間過去將損失最小化，藉此讓猜出來的公式越來越接近正確的那一個。

接下來，我們將數字改成可讓神經層接收的資料格式，Python 有一個稱為 Numpy 的程式庫，TensorFlow 可以使用它，在這裡，我們將數字放入一個 NumPy 陣列，來方便它處理它們：

```
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)
```

接下來，執行 `model.fit` 就可以開始進行學習程序了：

```
model.fit(xs, ys, epochs=500)
```

你可以把它想成「將 X 擬合至 Y，並嘗試 500 次」。在第一次嘗試時，電腦會猜測一個關係（也就是 Y = 10X + 10 之類的東西），並估計那次猜測的好壞，然後將結果傳給優化函式，優化函式會產生另一次猜測，這個程序會不斷重複，它的邏輯在於讓損失（或誤差）隨著時間而下降，因此「猜測」的結果會越來越好。

圖 1-19 是在 Colab notebook 執行它的畫面。注意 loss 值是如何隨著時間改變的。

```
Epoch 1/500
6/6 [==============================] - 9s 2s/sample - loss: 3.2868
Epoch 2/500
6/6 [==============================] - 0s 652us/sample - loss: 2.7447
Epoch 3/500
6/6 [==============================] - 0s 323us/sample - loss: 2.3150
Epoch 4/500
6/6 [==============================] - 0s 411us/sample - loss: 1.9737
Epoch 5/500
6/6 [==============================] - 0s 306us/sample - loss: 1.7021
Epoch 6/500
6/6 [==============================] - 0s 496us/sample - loss: 1.4853
Epoch 7/500
6/6 [==============================] - 0s 470us/sample - loss: 1.3117
Epoch 8/500
6/6 [==============================] - 0s 405us/sample - loss: 1.1723
Epoch 9/500
6/6 [==============================] - 0s 616us/sample - loss: 1.0596
Epoch 10/500
6/6 [==============================] - 0s 669us/sample - loss: 0.9682
```

圖 1-19　訓練神經網路

我們看到，經過前 10 個 epoch 之後，loss 從 3.2868 變成 0.9682，也就是說，只需要經過 10 次嘗試，網路的表現就比第一次猜測好三倍了，我們看看第 500 個 epoch 會怎樣（圖 1-20）。

```
Epoch 495/500
6/6 [==============================] - 0s 374us/sample - loss: 2.9063e-05
Epoch 496/500
6/6 [==============================] - 0s 540us/sample - loss: 2.8466e-05
Epoch 497/500
6/6 [==============================] - 0s 382us/sample - loss: 2.7882e-05
Epoch 498/500
6/6 [==============================] - 0s 397us/sample - loss: 2.7309e-05
Epoch 499/500
6/6 [==============================] - 0s 367us/sample - loss: 2.6748e-05
Epoch 500/500
6/6 [==============================] - 0s 363us/sample - loss: 2.6199e-05
```

圖 1-20　訓練神經網路——最後五個 epoch

我們可以看到 loss 是 $2.61 \times 10^{-5}$，損失變得非常小，因此模型幾乎已經找出兩個數字間的關係是 $Y = 2X - 1$ 了，電腦已經學會它們之間的模式了。

最後一行程式使用訓練好的模型來進行預測：

```
print(model.predict([10.0]))
```

在處理 ML 模型時，大家經常使用**預測**（*prediction*）這種說法，但是千萬不要把它想成預測未來！之所以使用這個詞是因為我們處理的是某種程度的不確定性，回想一下之前談過的活動偵測場景，當用戶用某個速度移動時，代表他**應該**是在走路。類似地，當模型學會兩件事情之間的模式時，它告訴我們答案**可能**是什麼，換句話說，它是在**預測**答案。（稍後你也會學到**推理**（*inference*），它是模型從許多答案裡選出一個，並且推測它選擇了正確的那一個。）

當我們要求模型預測「當 X 是 10 時，Y 是多少」時，你認為答案是什麼？你可以馬上想到 19，但不對，它會選出一個**非常接近** 19 的值，這有許多原因，首先，我們的損失不是 0，它仍然是個很小的值，所以我們可以預期，它預測的任何答案都會有極小的誤差，其次，這個神經網路只是用少量的資料來訓練的，這個例子只有 6 對 (X,Y) 值。

這個模型只有一個神經元，那個神經元會學習一個**權重**（*weight*）與一個**偏差**（*bias*），產生 Y = WX + B，這看起來正是我們想要的 Y = 2X – 1 關係，我們希望它學到 W = 2 且 B = –1，由於這個模型只用 6 個資料項目來訓練，我們不能預期答案就是那兩個值，而是非常接近它們的值。

請自行執行程式，看看你會得到什麼。我執行它時得到 18.977888，但你的答案可能稍微不同，因為在神經網路初始化時，有一個隨機元素——你最初的猜測會與我的稍微不同，別人的也會與我們的不同。

## 觀察網路學到什麼

我們在這個場景裡，用線性關係來匹配 X 與 Y，顯然它是個非常簡單的場景，之前的小節說過，神經元會學到權重與偏差參數，所以一個神經元非常適合學習這種關係，也就是當 Y = 2X – 1 時，權重是 2，偏差是 –1。在 TensorFlow 裡，我們可以觀察學來的權重與偏差，只要稍微修改程式即可：

```
import tensorflow as tf
import numpy as np
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

l0 = Dense(units=1, input_shape=[1])
model = Sequential([l0])
model.compile(optimizer='sgd', loss='mean_squared_error')

xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model.fit(xs, ys, epochs=500)

print(model.predict([10.0]))
print("Here is what I learned: {}".format(l0.get_weights()))
```

這兩個程式之間的不同在於，我建立了一個稱為 `l0` 的變數來保存 `Dense` 層，然後，在網路完成學習之後，我印出神經層學到的值（或權重）。

我的輸出是：

```
Here is what I learned: [array([[1.9967953]], dtype=float32),
array([-0.9900647], dtype=float32)]
```

因此，模型學到 X 與 Y 之間的關係是 Y = 1.9967953X – 0.9900647。

這非常接近我們期望的結果（Y = 2X – 1），我們甚至可以說這是比較合理的結果，因為我們假設其他的值也有這種關係。

## 小結

這就是你的第一個機器學習「Hello World」，你可能認為只用模型來找出兩個值之間的線性關係實在是大材小用，沒錯，但是最酷的事情在於，我們在這裡編寫程式的模式與處理更複雜的場景時的模式是一樣的，第 2 章會開始展示複雜的場景。我們將在第 2 章探索電腦視覺技術——電腦將會學習「看出」圖片裡的模式，並且指出它們裡面有什麼東西。