人工智慧緒論



我們需要的是一台可以從經驗中學習的機器。

−阿蘭·圖靈

1.1 人工智慧

資訊技術是人類歷史上的第三次工業革命，電腦、互聯網、智慧家居等技術的普及 極大地方便了人們的日常生活。通過程式設計的方式，人類可以將提前設計好的交互邏輯交給 機器重複且快速地執行，從而將人類從簡單枯燥的重複勞動工作中解脫出來。但是對於需 要較高智慧水準的任務，如人臉識別、聊天機器人、自動駕駛等任務，很難設計明確的邏 輯規則，傳統的程式設計方式顯得力不從心，而人工智慧(Artificial Intelligence，簡稱 AI)是有 望解決此問題的關鍵技術。

隨著深度學習演算法的崛起，人工智慧在部分任務上取得了類人甚至超人的智力水準， 如圍棋上 AlphaGo 智慧程式已經擊敗人類最強圍棋專家之一柯潔，在 Dota2 遊戲上

OpenAI Five 智慧程式擊敗冠軍隊伍 OG，同時人臉識別、智慧語音、機器翻譯等一項項實 用的技術已經進入到人們的日常生活中。現在我們的生活處處被人工智慧所環繞，儘管目 前能達到的智慧水準離通用人工智慧(Artificial General Intelligence，簡稱 AGI)還有一段距

離，我們仍堅定地相信人工智慧時代已經來臨。

接下來我們將介紹人工智慧、機器學習、深度學習的概念以及它們之間的聯繫與區 別。

**1.1.1** 人工智慧

人工智慧是讓機器獲得像人類一樣具有思考和推理機制的智慧技術，這一概念最早出 現在 1956 年召開的達特茅斯會議上。這是一項極具挑戰性的任務，人類目前尚無法對人腦 的工作機制有全面、科學的認知，希望能製造達到人腦水準的智慧型機器無疑是難於上青 天。即使如此，在某個方面呈現出類似、接近甚至超越人類智慧水準的機器被證明是可行 的。

怎麼實現人工智慧是一個非常廣袤的問題。人工智慧的發展主要經歷過三個階段，每 個階段都代表了人們從不同的角度嘗試實現人工智慧的探索足跡。早期，人們試圖通過總 結、歸納出一些邏輯規則，並將邏輯規則以電腦程式的方式實現，來開發出智慧系統。 但是這種顯式的規則往往過於簡單，並且很難表達複雜、抽象的概念和規則。這一階段被 稱為推理期。

1970 年代，科學家們嘗試通過知識庫加推理的方式解決人工智慧，通過構建龐大複雜 的專家系統來模擬人類專家的智慧水準。這些明確指定規則的方式存在一個最大的難題， 就是很多複雜、抽象的概念無法用具體的代碼實現。比如人類對圖片的識別、對語言的理 解過程，根本無法通過既定規則類比。為了解決這類問題，一門通過讓機器自動從資料中 學習規則的研究學科誕生了，稱為機器學習，並在 1980 年代成為人工智慧中的熱門學科。

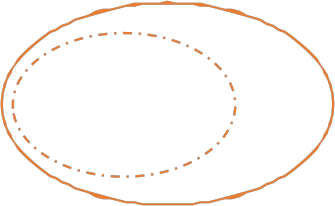
在機器學習中，有一門通過神經網路來學習複雜、抽象邏輯的方向，稱為神經網路。

神經網路方向的研究經歷了兩起兩落。2012 年開始，由於效果極為顯著，應用深層神經網

絡技術在電腦視覺、自然語言處理、機器人等領域取得了重大突破，部分任務上甚至超 越了人類智慧水準，開啟了以深層神經網路為代表的人工智慧的第三次復興。深層神經網

絡有了一個新名字，叫作深度學習。一般來講，神經網路和深度學習的本質區別並不大，

深度學習特指基於深層神經網路實現的模型或演算法。人工智慧、機器學習、神經網路和深 度學習的之間的關係如圖 1.1 所示。



深度學習

神經網路

機器學習

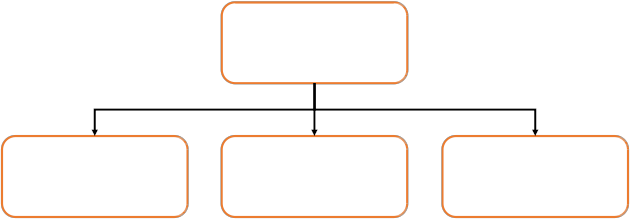
人工智慧

圖 1.1 人工智慧、機器學習、神經網路和深度學習

**1.1.2** 機器學習

機器學習可以分為有監督學習(Supervised Learning)、無監督學習(Unsupervised

Learning)和強化學習(Reinforcement Learning，簡稱 RL)，如圖 1.2 所示。



機器學習

有監督學習 無監督學習 強化學習

F𝜃 (X)🡪 y\*, label y 🡨-> target

圖 1.2 機器學習的分類

有監督學習 有監督學習的資料集包含了樣本x與樣本的標籤y，演算法模型需要學習到 映射關係f𝜃 : x → y，其中f𝜃 代表模型函數，𝜃為模型的參數。在訓練時，通過計算模型的預 測值f𝜃 (x)與真實標籤y之間的誤差來優化網路參數 𝜃，使得網路下一次能夠預測更精准。常見的有監督學習有線性回歸、邏輯回歸、支援向量機、隨機森林等。

監督學習 收集帶標籤的資料往往代價較為昂貴，對於只有樣本x的資料集，演算法需

要自行發現資料的模態，這種方式叫作無監督學習。無監督學習中有一類演算法將自身作為監督信號，即模型需要學習的映射為f𝜃 : x → x，稱為自監督學習(Self-supervised

Learning)。在訓練時，通過計算模型的預測值f𝜃 (x)與自身x之間的誤差來優化網路參數𝜃。

常見的無監督學習演算法有**自編碼器(self-encoder )**、**生成對抗網路**等。

強化學習 也稱為增強學習，通過與環境進行交互來學習解決問題的策略的一類演算法。 與有監督、無監督學習不同，強化學習問題並沒有明確的“正確的”動作監督信號，演算法 需要與環境進行交互，獲取環境回饋的滯後的獎勵信號，因此並不能通過計算動作與“正 確動作”之間的誤差來優化網路。常見的強化學習演算法有 DQN，PPO 等。

**1.1.3** 神經網路與深度學習

神經網路演算法是一類基於神經網路從資料中學習的演算法，它仍然屬於機器學習的範 疇。受限於計算能力和資料量，早期的神經網路層數較淺，一般在 1~4 層左右，網路表達 能力有限。隨著計算能力的提升和大資料時代的到來，高度並行化的 GPU 和海量資料讓大 規模神經網路的訓練成為可能。

2006 年，Geoffrey Hinton 首次提出深度學習的概念。2012 年，8 層的深層神經網路 AlexNet 發佈，並在圖片識別競賽中取得了巨大的性能提升，此後數十層、數百層、甚至 上千層的神經網路模型相繼提出，展現出深層神經網路強大的學習能力。一般將利用深層 神經網路實現的演算法稱作深度學習，本質上神經網路和深度學習可認為是相同的。

簡單來比較一下深度學習演算法與其它演算法的特點。如圖 1.3 所示。基於規則的系統一 般會編寫顯式的規則邏輯，這些邏輯一般是針對特定的任務設計的，並不適合其他任務。 傳統的機器學習演算法一般會人為設計具有一定通用性的特徵檢測方法，如 SIFT、HOG 特 征，這些特徵能夠適合某一類的任務，具有一定的通用性，但是如何設計特徵方法，以及 特徵方法的優劣性是問題的關鍵。神經網路的出現，使得人為設計特徵這一部分工作可以 通過神經網路讓機器自動學習完成，不需要人類干預。但是淺層的神經網路的特徵提取能 力較為有限，而深層的神經網路擅長提取高層、抽象的特徵，因此具有更好的性能表現。

測試版1205

輸出子網路

高層特徵提取

網路

輸出邏輯

輸出邏輯

輸出子網路

中層特徵提取

網路

針對特定任務 的檢測邏輯

人為設計的 特徵檢測方法

特徵提取網路

(淺層)

底層特徵提取 網路

基於規則的系統 傳統機器學習 淺層神經網路 深度學習

圖 1.3 深度學習與其它演算法比較

1.2 神經網路發展簡史

我們將神經網路的發展歷程大致分為淺層神經網路階段和深度學習階段，以 2006 年為 分割點。2006 年以前，深度學習以神經網路和連接主義名義發展，歷經了兩次興盛和兩次 寒冬；2006 年，Geoffrey Hinton 首次將**深層神經網路**命名為深度學習，自此開啟了深度學 習的第三次復興之路。

**1.2.1** 淺層神經網路

1943 年，心理學家 Warren McCulloch 和邏輯學家 Walter Pitts 根據生物神經元(Neuron)

結構，提出了最早的神經元數學模型，稱為 MP 神經元模型。該模型的輸出f(x)=ℎ(g(x))，其中g(x) = ∑𝑖 x𝑖 , x𝑖 ∈ {0,1}，模型通過g(x)的值來完成輸出值的預測，如圖 1.4 所示。如果g(x) ≥ 0，輸出為 1；如果g(x) < 0，輸出為 0。可以看到，MP 神經元模型並沒有學習能力，只能完成固定邏輯的判定。



圖 1.4 **MP 神經元模型**

1958 年，美國心理學家 Frank Rosenblatt 提出了第一個可以自動學習權重的神經元模

型，稱為感知機(Perceptron)，如圖 1.5 所示，輸出值o與真實值 之間的誤差用於調整神經 元的權重參數{w1 , w2 , … , wn }。Frank Rosenblatt 隨後基於“Mark 1 感知機”硬體實現感知

機模型，如圖 1.6、圖 1.7 所示，輸入為 400 個單元的圖像感測器，輸出為 8 個節點端

子，可以成功識別一些英文字母。一般認為 1943 年~1969 年為人工智慧發展的第一次興盛 期。



圖 1.5 感知機模型

測

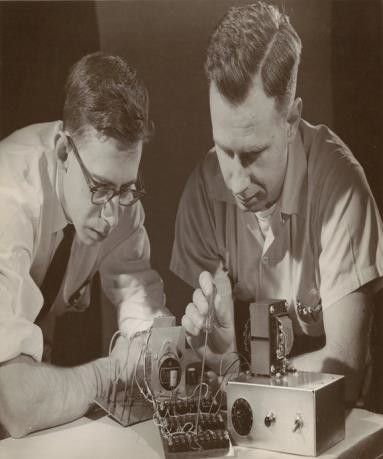
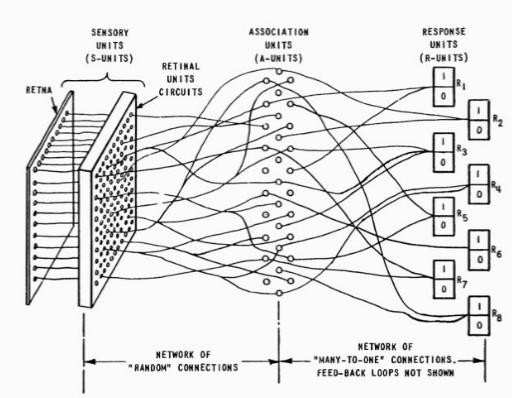


圖 1.6 Frank Rosenblatt 和 Mark 1 感知機① 圖 1.7 Mark 1 感知機網路結構②

1969 年，美國科學家 Marvin Minsky 等人在出版的《Perceptrons》一書中指出了感知

① 圖片來自 <https://slideplayer.com/slide/12771753/>

② 圖片來自 <https://www.glass-bead.org/article/machines-that-morph-logic/?lang=enview>

機等線性模型的主要缺陷，即無法處理簡單的異或 **XOR** 等線性不可分問題。這直接導致

了以感知機為代表的神經網路的相關研究進入了低谷期，一般認為 1969 年~1982 年為人工 智慧發展的第一次寒冬。

儘管處於 AI 發展的低谷期，仍然有很多意義重大的研究相繼發表，這其中最重要的 成果就是**誤差反向傳播演算法**(**Back Propagation**，簡稱 **BP 演算法**)的提出，它依舊是現代深度學習的核心理論基礎。實際上，反向傳播的數學思想早在 1960 年代就已經被推導出了，但是並沒有應用在神經網路上。**1974 年**，美國科學家 Paul Werbos 在他的博士論文中第一次提 出可以將 BP 演算法應用到神經網路上，遺憾的是，這一成果並沒有獲得足夠重視。直至

1986 年，David Rumelhart 等人在 Nature 上發表了通過 BP 演算法來進行表徵學習的論文，

BP 演算法才獲得了廣泛的關注。

1982 年，隨著 John Hopfild 的迴圈連接的 Hopfield 網路的提出，開啟了 1982 年~1995 年的第二次人工智慧復興的大潮，這段期間相繼提出了卷積神經網路、迴圈神經網路、反 向傳播演算法等演算法模型。1986 年，David Rumelhart 和 Geoffrey Hinton 等人將 BP 演算法應用在多層感知機上；1989 年 Yann LeCun 等人將 BP 演算法應用在**手寫數位圖片識別**上，取得 了巨大成功，這套系統成功商用在**郵遞區號識別**、**銀行支票識別**等系統上；1997 年，現在應用最為廣泛的迴圈神經網路變種之一 LSTM 被 Jürgen Schmidhuber 提出；同年雙向迴圈神經網路也被提出。**RNN LSTM**

遺憾的是，神經網路的研究隨著以支援向量機(Support Vector Machine，簡稱 SVM)為 代表的傳統機器學習演算法興起而逐漸進入低谷，稱為人工智慧的第二次寒冬。支持向量機 擁有嚴格的理論基礎，訓練需要的樣本數量較少，同時也具有良好的泛化能力，相比之 下，神經網路理論基礎欠缺，可解釋性差，很難訓練深層網路，性能也相對一般。圖 1.8 繪製了 1943 年~2006 年之間的重大時間節點。

圖靈測試

1950

XOR異或問題

1969

Hopfield

網路

1982

受限Boltzmann

1986

RNN

1986

雙向RNN

1997

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1943 | 1958 | 1974 | 1985 | 1986 | 1990 | 1997 | 2006 |
| 神經元模型 | 感知機模型 | BP反向傳播 | Boltzmann | MLP | LeNet | LSTM | DBN深度 |
|  |  |  | 機器 |  |  |  | 置信網路 |

圖 1.8 淺層神經網路發展時間線

**1.2.2** 深度學習

**2006 年，Geoffrey Hinton** 等人發現通過逐層預訓練的方式可以較好地訓練多層神經網 絡，並在 MNIST 手寫數位圖片資料集上取得了優於 SVM 的錯誤率，開啟了第三次人工智 能的復興。在論文中，Geoffrey Hinton 首次提出了 Deep Learning 的概念，這也是(深層)神 經網路被叫作深度學習的由來。2011 年，Xavier Glorot 提出了線性整流單元(Rectified Linear Unit，簡稱 **ReLU**)啟動函數active，這是現在使用最為廣泛的啟動函數之一。2012 年，

Alex Krizhevsky 提出了 **8 層的深層神經網路 AlexNet**，它採用了 ReLU 啟動函數，並使用

**Dropout 技術來防止過擬合**，同時拋棄了逐層預訓練的方式，直接在兩塊 NVIDIA GTX580

GPU 上訓練網路。**AlexNet 在 ILSVRC-2012 圖片識別比賽中獲得了第一名的成績**，比第二 名在 Top-5 錯誤率上降低了驚人的 10.9%。

自 AlexNet 模型提出後，各種各樣的演算法模型相繼被發表，其中有 **VGG** 系列、 **GoogLeNet** 系列、**ResNet** 系列、**DenseNet** 系列等。ResNet 系列模型將網路的層數提升至數 百層、甚至上千層，同時保持性能不變甚至更優。它演算法思想簡單，具有普適性，並且效 果顯著，是深度學習最具代表性的模型。

除了有監督學習領域取得了驚人的成果，在無監督學習和強化學習領域也取得了巨大 的成績。2014 年，Ian Goodfellow 提出了**生成對抗網路**，通過對抗訓練的方式學習樣本的 真實分佈，從而生成逼近度較高的樣本。此後，大量的生成對抗網路模型相繼被提出，最 新的圖片生成效果已經達到了肉眼難辨真偽的逼真度。2016 年，DeepMind 公司應用深度 神經網路到強化學習領域，提出了 **DQN 演算法**，在 Atari 遊戲平臺中的 49 個遊戲上取得了 與人類相當甚至超越人類的水準；在圍棋領域，DeepMind 提出的 AlphaGo 和 AlphaGo Zero 智慧程式相繼打敗人類頂級圍棋專家李世石、柯潔等；在多智慧體協作的 Dota2 遊戲 平臺，OpenAI 開發的 **OpenAI Five** 智慧程式在受限遊戲環境中打敗了 TI8 冠軍隊伍 OG

隊，展現出了大量專業級的高層智慧操作。圖 1.9 列出了 2006 年~2019 年之間重大的時間 節點。

GAN生成

TensorFlow

機器翻譯

德州撲克 TensorFlow

ImageNet

2009

對抗網路

2014

ResNet

2015

發佈

2015

AlphaGO

2016

BERT

2018

Pluribus

2019

2.0發佈

2019

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2006 | 2012 | 2014 | 2015 | 2015 | 2017 | 2019 |
| DBN深度 | AlexNet | VGG | DQN | Batch | AlphaGO | OpenAI |
| 置信網路 | 提出 | GoogLeNet |  | Normalization | Zero | Five |

圖 1.9 深度學習發展時間線



1.3 深度學習特點

與傳統的機器學習演算法、淺層神經網路相比，現代的深度學習演算法通常具有如下特 點。

**1.3.1** 數據量

早期的機器學習演算法比較簡單，容易快速訓練，需要的資料集規模也比較小，如 1936 年由英國統計學家 Ronald Fisher 收集整理的**鳶尾花卉資料集 Iris** 共包含 3 個類別花卉，每 個類別 50 個樣本。隨著電腦技術的發展，設計的演算法越來越複雜，對資料量的需求也隨 之增大。1998 年由 Yann LeCun 收集整理的 MNIST 手寫數位圖片資料集共包含 0~9 共 10

類數字，每個類別多達 7000 張圖片。隨著神經網路的興起，尤其是深度學習，網路層數一

般較深，模型的參數量可達百萬、千萬甚至十億個，**為了防止過擬合**，需要的資料集的規 模通常也是巨大的。現代社交媒體的流行也讓收集海量資料成為可能，如 2010 年發佈的

**ImageNet** 資料集收錄了共 14197122 張圖片，整個資料集的壓縮檔大小就有 154GB。圖

1.10、圖 1.11 列舉了一些資料集的樣本數和資料集大小隨時間的變化趨勢。

儘管深度學習對資料集需求較高，收集資料，尤其是收集帶標籤的資料，往往是代價 昂貴的。資料集的形成通常需要手動採集、爬取原始資料，並清洗掉無效樣本，再通過人 類智慧去標注資料樣本，因此不可避免地引入主觀偏差和隨機誤差。研究資料量需求較少 的演算法模型是非常有用的一個方向。

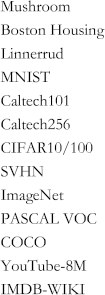
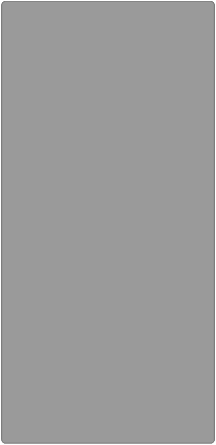


圖 1.10 資料集樣本數趨勢

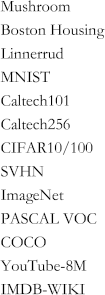
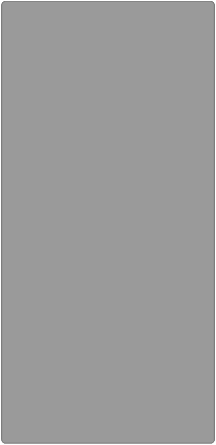


圖 1.11 資料集大小趨勢

**1.3.2** 計算力

計算能力的提升是第三次人工智慧復興的一個重要因素。實際上，現代深度學習的基 礎理論在 1980 年代就已經被提出，但直到 2012 年，基於**兩塊 GTX580 GPU 訓練的 AlexNet 發佈後**，深度學習的真正潛力才得以發揮。傳統的機器學習演算法並不像神經網路 這樣對資料量和計算能力有嚴苛的要求，通常在 CPU 上串列訓練即可得到滿意結果。但是

**深度學習非常依賴並行加速計算設備**，目前的大部分神經網路均使用 NVIDIA GPU 和

Google TPU 等並行加速晶片訓練模型參數。如圍棋程式 AlphaGo Zero 在 **64 塊 GPU** 上從 **零開始訓練了 40 天**才得以超越所有的 AlphaGo 歷史版本；**自動網路結構搜索演算法**使用了**800 塊 GPU** 同時訓練才能優化出較好的網路結構。

目前普通消費者能夠使用的深度學習加速硬體設備主要來自 NVIDIA 的 GPU 顯卡，

圖 1.12 例舉了從 2008 年到 2017 年 NVIDIA GPU 和 x86 CPU 的每秒 10 億次的浮點運算數 (GFLOPS)的指標變換曲線。可以看到，x86 CPU 的曲線變化相對緩慢，而 NVIDIA GPU 的浮點計算能力指數式增長，這主要是由日益增長的遊戲計算量和深度學習計算量等業務 驅動的。

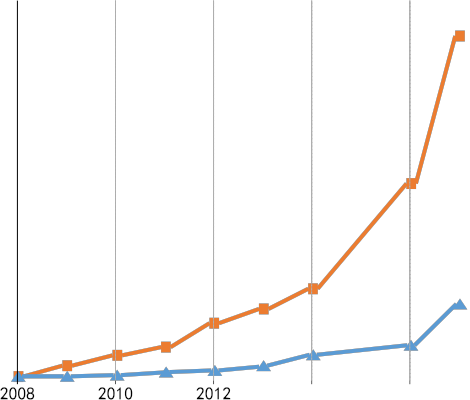


圖 1.12 NVIDIA GPU FLOPS 趨勢(資料來自 NVIDIA)

**1.3.3** 網路規模

早期的感知機模型和多層神經網路層數只有 1 層或者 2~4 層，網路參數量也在數萬左 右。隨著深度學習的興起和計算能力的提升，AlexNet(8 層)、VGG16(16 層)、 GoogLeNet(22 層)、ResNet50(50 層)、DenseNet121(121 層)等模型相繼被提出，同時輸入圖片的大小也從**28 × 28**逐漸增大，變成**224 × 224**、**299 × 299**等，這些變化使得網路的總參數量可達到千萬級別，如圖 1.13 所示。

網路規模的增大，使得神經網路的容量也相應增大，從而能夠學習到複雜的資料模態，模型的性能也會隨之提升；另一方面，網路規模的增大，意味著**更容易出現過擬合現象**，訓練需要的資料集和計算代價也會變大。

ILSVRC挑戰賽ImageNet資料集分類任務

網路模型層數 Top-5錯誤率

152

28.2 25.8

16.4

11.7

19 22

4 4 8 8

7.3 6.7 3.57

ILSVRC10 ILSVRC11 ILSVRC12 ILSVRC13 ILSVRC14 ILSVRC14 ILSVRC15

圖 1.13 網路層數變化趨勢



**1.3.4** 通用智慧

過去，為了提升某項任務上的演算法性能，往往需要利用先驗知識手動設計相應的特 徵，以説明演算法更好地收斂到最優解。這類特徵提取方法往往是與具體任務場景強相關 的，一旦場景發生了變動，這些依靠人工設計的特徵和先驗設定無法自我調整新場景，往往需要重新設計演算法模型，模型的通用性不強。

設計一種像人腦一樣可以自動學習、自我調整的通用智慧機制一直是人類的共同願 景。從目前來看，深度學習是最接近通用智慧的演算法之一。在電腦視覺領域，**過去需要針對具體的任務設計特徵、添加先驗假設的做法**，已經被深度學習演算法拋棄了，目前在**圖片識別**、**目標檢測**、**語義分割**等方向，幾乎都是基於深度學習端到端地訓練，獲得的模型 性能好，適應性強；在 Atria 遊戲平臺上，DeepMind 設計的 **DQN 演算法模型**可以在相同的 演算法、模型結構和超參數的設定下，**在 49 個遊戲上獲得人類相當的遊戲水準**，呈現出一定 程度的通用智慧。圖 1.14 是 DQN 演算法的網路結構，它並不是針對於某個遊戲而設計的， 而是可以控制 Atria 遊戲平臺上的 49 個遊戲。

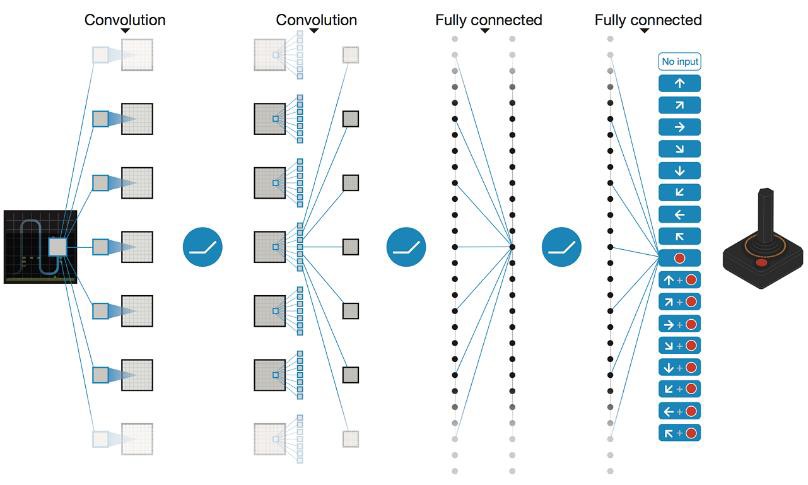


圖 1.14 DQN 演算法網路結構示意圖 [1]

1.4 深度學習應用

深度學習演算法已經廣泛應用到人們生活的角角落落，例如手機中的語音助手、汽車上 的智慧輔助駕駛、人臉支付等。我們將從電腦視覺、自然語言處理和強化學習 3 個領域 入手，為大家介紹深度學習的一些主流應用。

**1.4.1** 電腦視覺

**圖片識別(Image Classification)**  是常見的分類問題。神經網路的輸入為圖片資料，輸出 值為當前樣本屬於每個類別的概率分佈。通常選取概率值最大的類別作為樣本的預測類 別。圖片識別是最早成功應用深度學習的任務之一，經典的網路模型有 **VGG 系列**、 **Inception 系列、ResNet 系列等**。

**目標檢測(Object Detection)**  是指通過演算法自動檢測出圖片中常見物體的大致位置，通 常用邊界框(Bounding box)表示，並分類出邊界框中物體的類別資訊，如圖 1.15 所示。常 見的目標檢測演算法有 **RCNN、Fast RCNN、Faster RCNN、Mask RCNN、SSD、YOLO** 系列 等。

**語義分割(Semantic Segmentation)** 是通過演算法自動分割並識別出圖片中的內容，可以 將語義分割理解為每個圖元點的分類問題，分析每個圖元點的物體的類別資訊，如圖 1.16 所示。常見的語義分割模型有 **FCN、U-net、SegNet、DeepLab 系列**等。

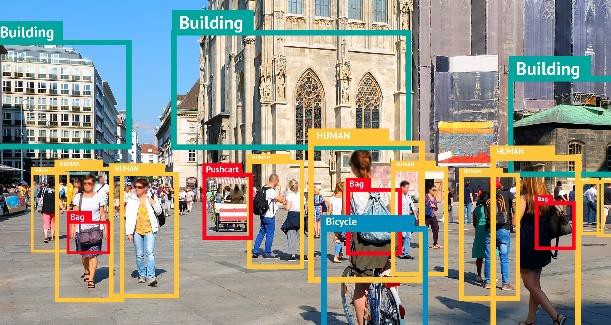
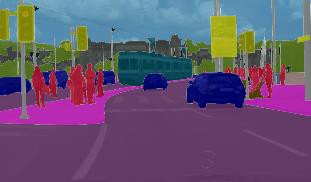


圖 1.15 目標檢測效果圖 圖 1.16 語義分割效果圖

**視頻理解(Video Understanding)**  隨著深度學習在 2D 圖片的相關任務上取得較好的效 果，具有時間維度資訊的 3D 視頻理解任務受到越來越多的關注。常見的視頻理解任務有 **視頻分類、行為檢測、視頻主體抽取**等。常用的模型有 **C3D、TSN、DOVF、TS\_LSTM** 等。

**圖片生成(Image Generation)** 通過學習真實圖片的分佈，並從學習到的分佈中採樣而獲 得逼真度較高的生成圖片。目前常見的生成模型有 **VAE 系列、GAN 系列等**。其中 GAN 系 列演算法近年來取得了巨大的進展，最新 GAN 模型產生的圖片效果達到了肉眼難辨真偽的 程度，如圖 1.17 為 GAN 模型的生成圖片。

除了上述應用，深度學習還在其它方向上取得了不俗的效果，比如**藝術風格遷移**(圖

1.18)、**超解析度、圖片去燥/去霧、灰度圖片著色**等一系列非常實用酷炫的任務，限於篇 幅，不再贅述。



圖 1.17 自動生成的圖片 圖 1.18 藝術風格遷移效果圖



**1.4.2** 自然語言處理

機器翻譯(Machine Translation) 過去的機器翻譯演算法通常是基於統計機器翻譯模型，這 也是 2016 年前 Google 翻譯系統採用的技術。2016 年 11 月，Google 基於 **Seq2Seq 模型**上 線了 **Google 神經機器翻譯系統(GNMT)**，首次實現了來源語言到目的語言的直譯技術，在多 項任務上獲得了 50~90%的效果提升。常用的機器翻譯模型有 **Seq2Seq、BERT、GPT、**

**GPT-2 等**，其中 OpenAI 提出的 GPT-2 **模型參數量高達 15 億個**，甚至發佈之初以技術安全 考慮為由拒絕開源 GPT-2 模型。

**聊天機器人(Chatbot)** 聊天機器人也是自然語言處理的一項主流任務，機器自動學習與 人類對話，對於人類的簡單訴求提供滿意的自動回復，提高客戶的服務效率和服務品質 等。常應用在諮詢系統、娛樂系統、智慧家居等中。

**1.4.3** 強化學習

虛擬遊戲 相對於真實環境，虛擬遊戲平臺既可以訓練、測試強化學習演算法，又可以避免無關因素干擾，同時也能將實驗代價降到最低。目前常用的虛擬遊戲平臺有 OpenAI Gym、OpenAI Universe、OpenAI Roboschool、DeepMind OpenSpiel、MuJoCo 等，常用的 強化學習演算法有 DQN、A3C、A2C、PPO 等。在圍棋領域，DeepMind AlaphGo 程式已經 超越人類圍棋專家；在 Dota2 和星際爭霸遊戲上，OpenAI 和 DeepMind 開發的智慧程式也 在限制規則下戰勝了職業隊伍。

**機器人(Robotics) 在真實環境中**，機器人的控制也取得了一定的進展。如 UC Berkeley 實驗室在機器人領域的 Imitation Learning、Meta Learning、Few-shot Learning 等方向上取得 了不少進展。美國波士頓動力公司在機器人應用中取得喜人的成就，其製造的機器人在複 雜地形行走、多智慧體協作等任務上表現良好(圖 1.19)。

**自動駕駛(Autonomous Driving)**  被認為是強化學習短期內能技術落地的一個應用方 向，很多公司投入大量資源在自動駕駛上，如百度、Uber、Google 無人車等，其中百度的 無人巴士“阿波龍”已經在北京、雄安、武漢等地展開試運營，圖 1.20 為百度的自動駕駛 汽車。



圖 1.19 波士頓動力公司的機器人③



圖 1.20 百度 Apollo 自動駕駛汽車④

1.5 深度學習框架

工欲善其事，必先利其器。在瞭解了深度學習相關知識後，我們來挑選一下實現深度 學習演算法所使用的工具吧。

**1.5.1** 主流框架

❑ **Theano** 是最早的深度學習框架之一，由 Yoshua Bengio 和 Ian Goodfellow 等人開發， 是一個基於 Python 語言、定位底層運算的計算庫，Theano 同時支援 GPU 和 CPU 運 算。由於 Theano 開發效率較低，模型編譯時間較長，同時開發人員轉投 TensorFlow 等原因，**Theano 目前已經停止維護**。

❑ **Scikit-learn**  是一個完整的面向機器學習演算法的計算庫，**內建了常見的傳統機器學習算 法支持**，文檔和案例也較為豐富，但是 Scikit-learn 並不是專門面向神經網路而設計 的，不支援 GPU 加速，**對神經網路相關層的實現也較欠缺**。

❑ **Caffe** 由華人賈揚清在 2013 年開發，主要面向使用**卷積神經網路**的應用場合，並不適 合其它類型的神經網路的應用。Caffe 的主要開發語言是 C++，也提供 Python 語言等 介面，支援 GPU 和 CPU。由於開發時間較早，在業界的知名度較高，2017 年 Facebook 推出了 Caffe 的升級版本 Cafffe2，Caffe2 目前已經融入到 PyTorch 庫中。

❑ **Torch** 是一個非常優秀的科學計算庫，基於較冷門的程式設計語言 **Lua 開發**。Torch 靈活性 較高，容易實現自訂網路層，這也是 PyTorch 繼承獲得的優良基因。但是由於 Lua 語言使用人群較少，Torch 一直未能獲得主流應用。

❑ **MXNet** 由華人陳天奇和李沐等人開發，是亞馬遜公司的官方深度學習框架。採用了 命令式程式設計和符號式程式設計混合方式，靈活性高，運行速度快，文檔和案例也較為豐 富。

❑ **PyTorch** 是 Facebook 基於原 **Torch** 框架推出的採用 Python 作為主要開發語言的深度學

③ 圖片來自 <https://www.bostondynamics.com/>

④ 圖片來自 [https://venturebeat.com/2019/01/08/baidu-announces-apollo-3-5-and-apollo-enterprise-says-it-has- over-130-partners/](https://venturebeat.com/2019/01/08/baidu-announces-apollo-3-5-and-apollo-enterprise-says-it-has-over-130-partners/)

習框架。PyTorch 借鑒了 Chainer 的設計風格，採用命令式程式設計，使得搭建網路和調試

網路非常方便。儘管 PyTorch 在 2017 年才發佈，但是由於精良緊湊的介面設計，

PyTorch 在學術界獲得了廣泛好評。在 PyTorch 1.0 版本後，原來的 PyTorch 與 Caffe2

進行了合併，彌補了 PyTorch 在工業部署方面的不足。總的來說，PyTorch 是一個非常 優秀的深度學習框架。

❑ **Keras** 是一個基於 Theano 和 TensorFlow 等框架提供的底層運算而實現的高層框架， 提供了大量快速訓練、測試網路的高層介面。對於常見應用來說，使用 Keras 開發效 率非常高。但是由於沒有底層實現，需要對底層框架進行抽象，運行效率不高，靈活 性一般。

❑ TensorFlow 是 Google 於 2015 年發佈的深度學習框架，最初版本只支援符號式程式設計。 得益於發佈時間較早，以及 Google 在深度學習領域的影響力，TensorFlow 很快成為最 流行的深度學習框架。但是由於 TensorFlow 介面設計頻繁變動，功能設計重複冗餘， 符號式程式設計開發和調試非常困難等問題，TensorFlow 1.x 版本一度被業界詬病。**2019 年，Google 推出 TensorFlow 2 正式版本**，將**以動態圖優先模式運行**，從而能夠避免 TensorFlow 1.x 版本的諸多缺陷，已獲得業界的廣泛認可。

目前來看，TensorFlow 和 PyTorch 框架是業界使用最為廣泛的兩個深度學習框架， TensorFlow 在工業界擁有完備的解決方案和用戶基礎，PyTorch 得益於其精簡靈活的介面 設計，可以快速搭建和調試網路模型，在學術界獲得好評如潮。TensorFlow 2 發佈後，彌 補了 TensorFlow 在上手難度方面的不足，使得用戶既能輕鬆上手 TensorFlow 框架，又能無 縫部署網路模型至工業系統。本書以 TensorFlow 2.0 版本作為主要框架，實現深度學習算 法。

這裡特別介紹 TensorFlow 與 Keras 之間的聯繫與區別。Keras 可以理解為一套高層 API 的設計規範，Keras 本身對這套規範有官方的實現，**在 TensorFlow 中也實現了這套規範， 稱為 tf.keras 模組**，並且 **tf.keras 將作為 TensorFlow 2 版本的唯一高層介面**，避免出現介面 重複冗餘的問題。如無特別說明，本書中 Keras 均指代 tf.keras。

**1.5.2 TensorFlow 2** 與 **1.x**

TensorFlow 2 是一個與 TensorFlow 1.x 使用體驗完全不同的框架，TensorFlow 2 不相容 TensorFlow 1.x 的代碼，同時在程式設計風格、函數介面設計等上也大相徑庭，TensorFlow 1.x 的代碼需要依賴人工的方式遷移，自動化遷移方式並不靠譜。Google 即將停止更新 TensorFlow 1.x，不建議學習 TensorFlow 1.x 版本。

TensorFlow 2 支援動態圖優先模式，在計算時可以同時獲得計算圖與數值結果，可以 代碼中調試並即時列印資料，搭建網路也像搭積木一樣，層層堆疊，非常符合軟體發展思 維。

以簡單的2.0 + 4.0的加法運算為例，在 TensorFlow 1.x 中，首先創建計算圖，代碼如

下：

import tensorflow as tf

# 1.創建計算圖階段，此處代碼需要使用 tf 1.x 版本運行

# 創建 2 個輸入端子，並指定類型和名字

a\_ph = tf.placeholder(tf.float32, name='variable\_a')

b\_ph = tf.placeholder(tf.float32, name='variable\_b')

# 創建輸出端子的運算操作，並命名

c\_op = tf.add(a\_ph, b\_ph, name='variable\_c')

創建計算圖的過程就類比通過符號建立公式c = a + b的過程，僅僅是記錄了公式的計算步驟，並沒有實際計算公式的數值結果，需要通過運行公式的輸出端子c，並賦值a =

2.0, b = 4.0才能獲得c的數值結果，代碼如下：

# 2.運行計算圖階段，此處代碼需要使用 tf 1.x 版本運行

# 創建運行環境

sess = tf.InteractiveSession()

# 初始化步驟也需要作為操作運行

init = tf.global\_variables\_initializer()

sess.run(init) # 運行初始化操作，完成初始化

# 運行輸出端子，需要給輸入端子賦值

c\_numpy = sess.run(c\_op, feed\_dict={a\_ph: 2., b\_ph: 4.})

# 運算完輸出端子才能得到數數值型別的 c\_numpy

print('a+b=',c\_numpy)

可以看到，在 TensorFlow 中完成簡單的2.0 + 4.0加法運算尚且如此繁瑣，更別說創建複雜

的神經網路演算法有多艱難。這種先創建計算圖後運行的程式設計方式叫做符號式程式設計。 接下來我們使用 TensorFlow 2 來完成2.0 + 4.0運算，代碼如下：

import tensorflow as tf

# 此處代碼需要使用 tf 2 版本運行

# 1.創建輸入張量，並賦初始值

a = tf.constant(2.)

b = tf.constant(4.)

# 2.直接計算，並列印結果 print('a+b=',a+b) 可以看到，計算過程非常簡潔，沒有多餘的計算步驟。

這種運算時同時創建計算圖c = a + b和數值結果6.0 = 2.0 + 4.0的方式叫做**命令式編**

**程**，也稱為**動態圖模式**。TensorFlow 2 和 PyTorch 都是採用動態圖(優先)模式開發，調試方

便，所見即所得。一般來說，動態圖模式開發效率高，但是**運行效率可能不如靜態圖模 式**。TensorFlow 2 也支持通過 **tf.function 將動態圖優先模式的代碼轉化為靜態圖模式**，實現 開發和運行效率的雙贏。

**1.5.3** 功能演示

深度學習的核心是演算法的設計思想，深度學習框架只是我們實現演算法的工具。下面我 們將演示 TensorFlow 深度學習框架的三大核心功能，從而説明我們理解框架在演算法設計中 扮演的角色。

**a)** 加速計算 神經網路本質上由大量的矩陣相乘、矩陣相加等基本數學運算構成，TensorFlow 的重

要功能就是利用 GPU 方便地實現平行計算加速功能。為了演示 GPU 的加速效果，我們通

過完成多次矩陣A和矩陣B的矩陣相乘運算，並測量其平均運算時間來比對。其中矩陣A的

shape 為[1, n]，矩陣B的 shape 為[n, 1]，通過調節n即可控制矩陣的大小。

首先我們分別創建使用 CPU 和 GPU 環境運算的 2 個矩陣，代碼如下：

# 創建在 CPU 環境上運算的 2 個矩陣

with tf.device('/cpu:0'):

cpu\_a = tf.random.normal([1, n]) cpu\_b = tf.random.normal([n, 1]) print(cpu\_a.device, cpu\_b.device)

# 創建使用 GPU 環境運算的 2 個矩陣

with tf.device('/gpu:0'):

gpu\_a = tf.random.normal([1, n])

gpu\_b = tf.random.normal([n, 1])

print(gpu\_a.device, gpu\_b.device)

接下來實現 CPU 和 GPU 運算的函數，並通過 timeit.timeit()函數來測量兩個函數的運 算時間。需要注意的是，第一次計算時一般需要完成額外的環境初始化工作，因此這段時 間不能計算在內。我們通過熱身環節將這段時間去除，再測量運算時間，代碼如下：

def cpu\_run(): # CPU 運算函數

with tf.device('/cpu:0'):

c = tf.matmul(cpu\_a, cpu\_b)

return c

def gpu\_run():# GPU 運算函數

with tf.device('/gpu:0'):

c = tf.matmul(gpu\_a, gpu\_b)

return c

# 第一次計算需要熱身，避免將初始化時間結算在內

cpu\_time = timeit.timeit(cpu\_run, number=10) gpu\_time = timeit.timeit(gpu\_run, number=10) print('warmup:', cpu\_time, gpu\_time)

# 正式計算 10 次，取平均時間

cpu\_time = timeit.timeit(cpu\_run, number=10)

gpu\_time = timeit.timeit(gpu\_run, number=10)

print('run time:', cpu\_time, gpu\_time)

我們將不同大小n下的 CPU 和 GPU 環境的運算時間繪製為曲線，如圖 1.21 所示。可 以看到，在矩陣A和矩陣B較小時，CPU 和 GPU 時間幾乎一致，並不能體現出 GPU 並行

計算的優勢；在矩陣較大時，CPU 的計算時間明顯上升，而 GPU 充分發揮平行計算優

勢，運算時間幾乎不變。

**b)** 自動梯度



圖 1.21 CPU/GPU 矩陣相乘時間

在使用 TensorFlow 構建前向計算過程的時候，除了能夠獲得數值結果，TensorFlow 還 會自動構建計算圖，通過 TensorFlow 提供的自動求導的功能，可以不需要手動推導，即可 計算輸出對網路參數的偏導數。考慮如下函數的運算式：

y= aw2 + bw + c

輸出 對於變數w的導數關係為：

dy

dw = 2aw +b

考慮在(a, b, c, w) = (1,2,3,4)處的導數，代入上式可得d𝑦 = 2 ∙ 1 ∙ 4 + 2 = 10。我們通過手

d𝑤

動推導的方式計算出導數值為 10。

通過 TensorFlow 的方式，我們可以不需要手動推導導數的運算式，直接給出函數的表 達式，即可由 TensorFlow 自動求導，代碼實現如下：

import tensorflow as tf

# 創建 4 個張量，並賦值

a = tf.constant(1.) b = tf.constant(2.) c = tf.constant(3.) w = tf.constant(4.)

with tf.GradientTape() as tape:# 構建梯度環境

tape.watch([w]) # 將 w 加入梯度跟蹤列表

# 構建計算過程，函數運算式

y = a \* w\*\*2 + b \* w + c

# 自動求導

[dy\_dw] = tape.gradient(y, [w])

print(dy\_dw) # 列印出導數

程式的運行結果為：

tf.Tensor(10.0, shape=(), dtype=float32)

可以看到，TensorFlow 自動求導的結果與手動計算的結果一致。

c) 常用神經網路介面

TensorFlow 除了提供底層的矩陣相乘、相加等數學函數，還內建了常用神經網路運算 函數、常用網路層、網路訓練、模型保存與載入、網路部署等一系列深度學習系統的便捷 功能。使用 TensorFlow 開發，可以方便地利用這些功能完成常用業務流程，高效穩定。

1.6 開發環境安裝

在領略完深度學習框架所帶來的的便利後，我們來著手在本地電腦環境安裝 TensorFlow 最新版框架。TensorFlow 框架支援多種常見的作業系統，如 Windows 10、 Ubuntu 18.04、Mac OS 等，**支援運行在 NVIDIA 顯卡上的 GPU 版本**和僅使用 CPU 完成計 算的 CPU 版本。我們以最為常見的 Windows 10 系統，NVIDIA GPU 和 Python 語言環境為 例，介紹如何安裝 TensorFlow 框架及其它開發軟體。

一般來說，開發環境安裝分為 4 大步驟：安裝 Python 解譯器 Anaconda，安裝 CUDA

加速庫，安裝 TensorFlow 框架和安裝常用編輯器。

**1.6.1 Anaconda** 安裝

Python 解譯器是讓以 Python 語言編寫的代碼能夠被 CPU 執行的橋樑，是 Python 語言 的核心軟體。用戶可以從 <https://www.python.org/>網站下載最新版本(Python 3.7)的解譯器， 像普通的應用軟體一樣安裝完成後，就可以調用 python.exe 程式執行 Python 語言編寫的源 代碼檔(.py 格式)。

我們這裡選擇安裝集成了 Python 解譯器和虛擬環境等一系列協助工具的 Anaconda 軟 件，通過安裝 Anaconda 軟體，可以同時獲得 Python 解譯器、包管理和虛擬環境等一系列 便捷功能，何樂而不為呢。我們從 <https://www.anaconda.com/distribution/#download-section>網址進入 Anaconda 下載頁面，選擇 Python 最新版本的下載連結即可下載，下載完成後安 裝即可進入安裝程式。如圖 1.22 所示，勾選”Add Anaconda to my PATH environment variable”一項，這樣可以通過命令列方式調用 Anaconda 程式。如圖 1.23 所示，安裝程式 詢問是否連帶安裝 VS Code 軟體，選擇 Skip 即可。整個安裝流程約持續 5 分鐘，具體時間 需依據電腦性能而定。

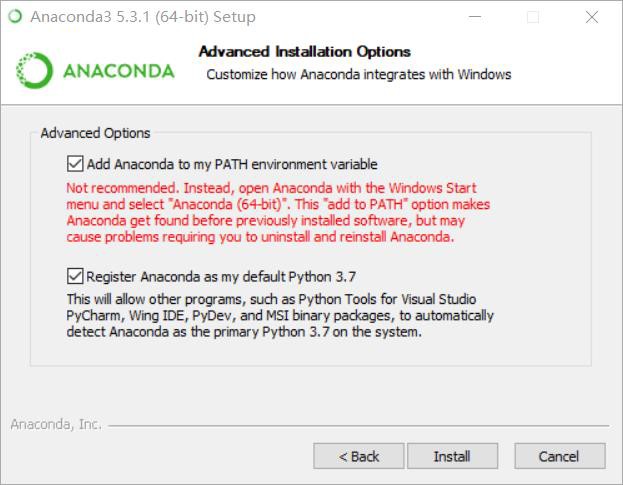
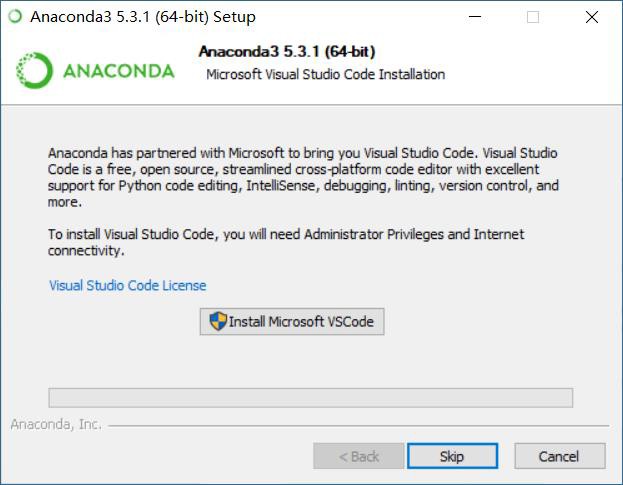


圖 1.22 Anaconda 安裝介面-1 圖 1.23Anaconda 安裝介面-2



安裝完成後，怎麼驗證 Anaconda 是否安裝成功呢？通過鍵盤上的 Windows 鍵+R 鍵， 即可調出運行程式對話方塊，輸入“cmd”並回車即打開 Windows 自帶的命令列程式 cmd.exe。或者點擊開始功能表，輸入“cmd”也可搜索到 cmd.exe 程式，打開即可。輸入 conda list 命令即可查看 Python 環境已安裝的庫，如果是新安裝的 Python 環境，則列出的 庫都是 Anaconda 自帶的軟體庫，如圖 1.24 所示。如果 conda list 能夠正常彈出一系列的庫 清單資訊，說明 Anaconda 軟體安裝成功；如果 conda 命名不能被識別，則說明安裝失敗， 需要重新安裝。

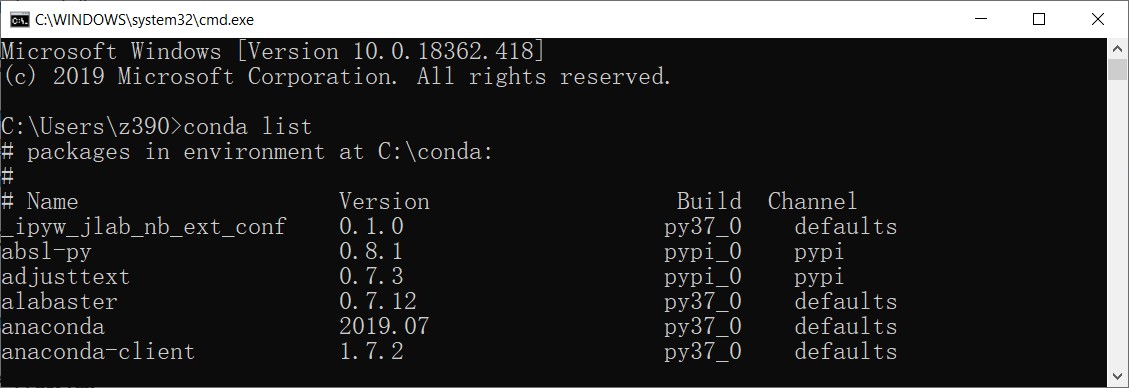


圖 1.24 Anaconda 安裝結果測試

**1.6.2 CUDA** 安裝

目前的深度學習框架大都基於 NVIDIA 的 GPU 顯卡進行加速運算，因此需要安裝 NVIDIA 提供的 GPU 加速庫 CUDA 程式。在安裝 CUDA 之前，請確認計算機具有支援 CUDA 程式的 NVIDIA 顯卡設備。如果電腦沒有 NVIDIA 顯卡，例如部分電腦顯卡生 產商為 AMD 或 Intel，則無法安裝 CUDA 程式，因此可以跳過這一步，直接進入 TensorFlow CPU 版本的安裝。

CUDA 的安裝分為 CUDA 軟體的安裝、cuDNN 深度神經網路加速庫的安裝和環境變 量配置三個步驟，安裝過程稍微繁瑣，請讀者在操作時思考每個步驟的用途，避免死記硬 背流程。

**CUDA** 軟體安裝 打開 CUDA 程式的下載官網：[https://developer.nvidia.com/cuda-10.0- download-archive](https://developer.nvidia.com/cuda-10.0-download-archive)，這裡我們使用 CUDA 10.0 版本，依次選擇 Windows 平臺，x86\_64 架 構，10 系統，exe(local)本地安裝包，再選擇 Download 即可下載 CUDA 安裝軟體。下載完 成後，打開安裝軟體。如圖 1.23 所示，選擇“Custom”選項，點擊 NEXT 按鈕進入圖

1.26 安裝程式選擇清單，在這裡選擇需要安裝和取消不需要安裝的程式元件。在 CUDA 節

點下，取消”Visual Studio Integration”一項；在“Driver components”節點下，比對目前計 算機已經安裝的顯卡驅動“Display Driver”的版本號“Current Version”和 CUDA 自帶的

顯卡驅動版本號“New Version”，如果“Current Version”大於“New Version”，則需要取

消“Display Driver”的勾，如果小於或等於，則默認勾選即可，如圖 1.27 所示。設置完成 後即可正常安裝。

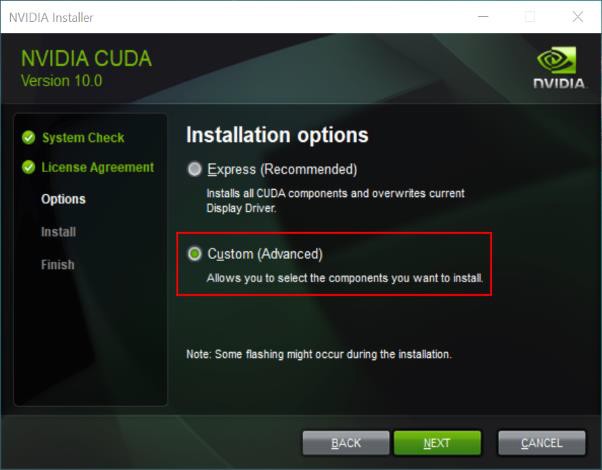
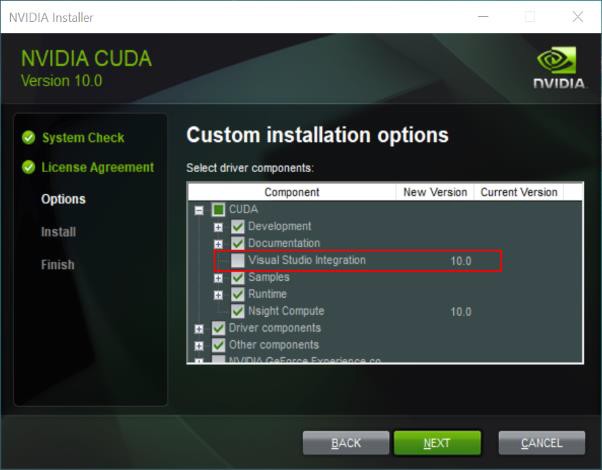


圖 1.25 CUDA 安裝介面-1 圖 1.26 CUDA 安裝介面-2

安裝完成後，我們來測試 CUDA 軟體是否安裝成功。打開 cmd 命令列，輸入“nvcc - V”，即可列印當前 CUDA 的版本資訊，如圖 1.29 所示，如果命令無法識別，則說明安裝 失敗。同時我們也可從 CUDA 的安裝路徑“C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\bin”下找到“nvcc.exe”程式，如圖 1.28 所示。

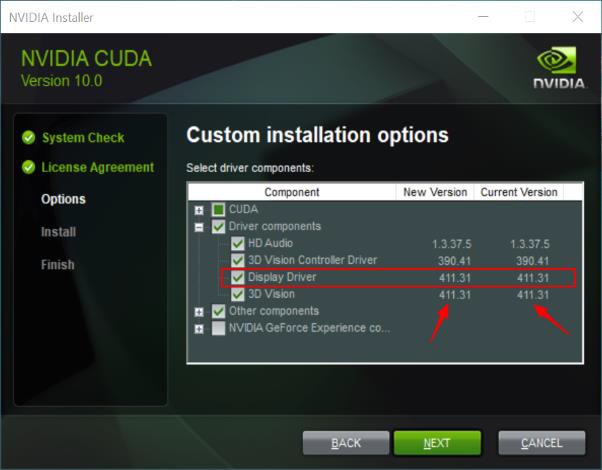
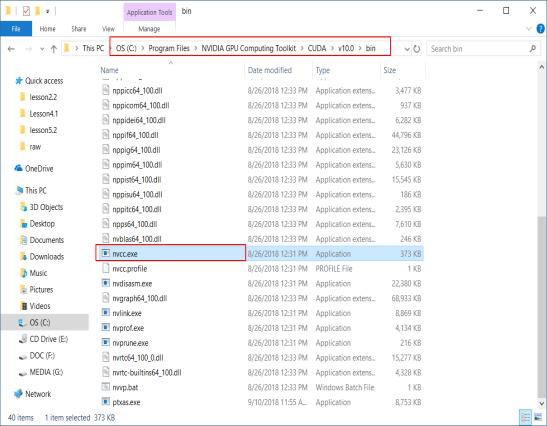


圖 1.27 CUDA 安裝介面-3 圖 1.28 CUDA 安裝結果測試-1

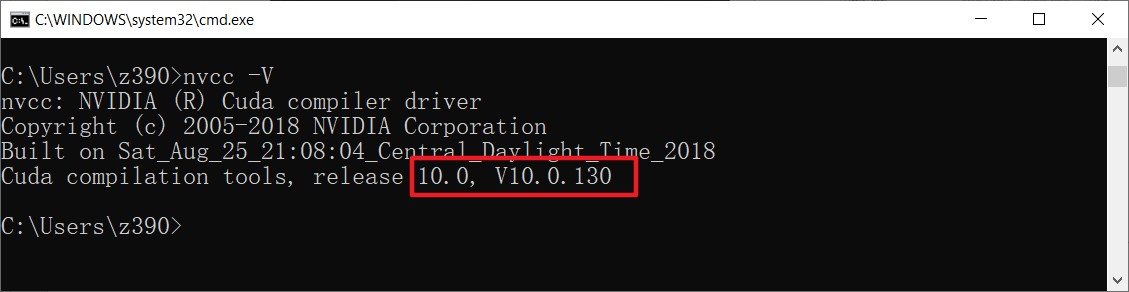


圖 1.29 CUDA 安裝結果測試-2

**cuDNN** 神經網路加速庫安裝 CUDA 並不是針對於神經網路專門的 GPU 加速庫，它面 向各種需要平行計算的應用設計。如果希望針對于神經網路應用加速，需要額外安裝

cuDNN 庫。需要注意的是，cuDNN 庫並不是運行程式，只需要下載解壓 cuDNN 檔，並

配置 Path 環境變數即可。

打開網址 <https://developer.nvidia.com/cudnn>，選擇“Download cuDNN”，由於 NVIDIA 公司的規定，下載 cuDNN 需要先登錄，因此使用者需要登錄或創建新使用者後才能繼續下 載。登錄後，進入 cuDNN 下載介面，勾選“I Agree To the Terms of the cuDNN Software License Agreement”，即可彈出 cuDNN 版本下載選項。選擇 CUDA 10.0 匹配的 cuDNN 版 本，並點擊“cuDNN Library for Windows 10”連結即可下載 cuDNN 檔，如圖 1.30 所 示。需要注意的是，cuDNN 本身具有一個版本號，同時它還需要和 CUDA 的版本號匹配 上，不能下錯不匹配 CUDA 版本號的 cuDNN 檔。

1205

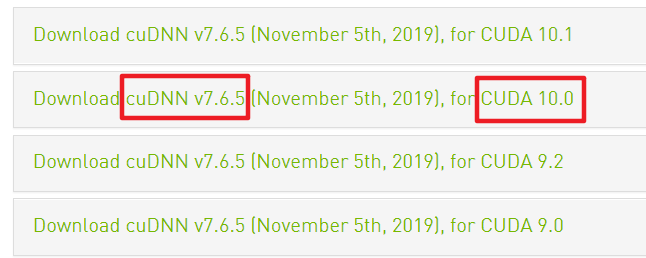


圖 1.30 cuDNN 版本選擇介面

下載完成 cuDNN 檔後，解壓並進入資料夾，我們將名為“cuda”的資料夾重命名為 “cudnn765”，並複製此資料夾。進入 CUDA 的安裝路徑 C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0，粘貼“cudnn765”資料夾即可，此處可能會彈出需要管理 員許可權的對話方塊，選擇繼續即可粘貼，如圖 1.31 所示。

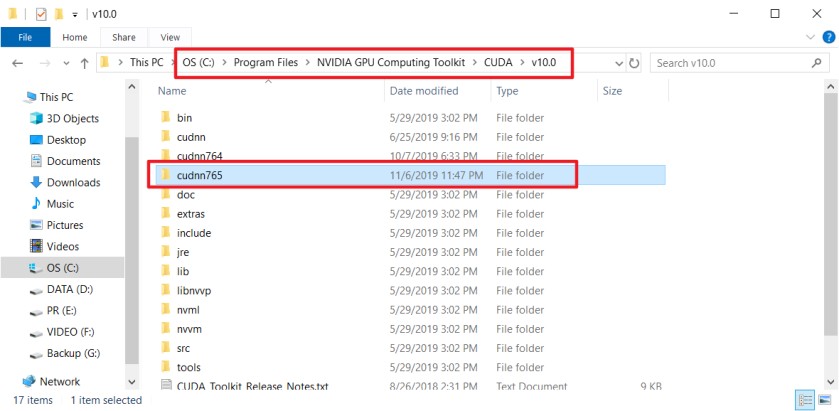


圖 1.31 cuDNN 檔的安裝

環境變數 **Path** 配置 上述 cudnn 資料夾的複製即已完成 cuDNN 的安裝，但為了讓系統 能夠感知到 cuDNN 檔的位置，我們需要額外配置 Path 環境變數。打開檔流覽器，在 “我的電腦”上右擊，選擇“屬性”，選擇“高級系統屬性”，選擇“環境變數”，如圖

1.32 所示。在“系統變數”一欄中選中“Path”環境變數，選擇“編輯”，如圖 1.33 所 示。選擇“新建”，輸入 cuDNN 的安裝路徑“C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\cudnn765\bin”，並通過“向上移動”按鈕將這一項上移置頂。

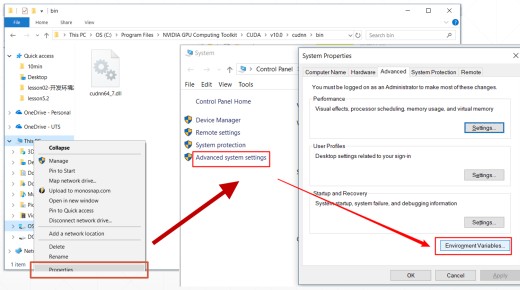
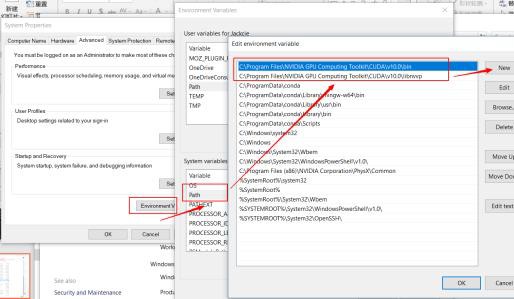


圖 1.32 修改環境變數-1 圖 1.33 修改環境變數-2



CUDA 安裝完成後，環境變數中應該包含“C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\bin”、“C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\libnvvp”和“C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\cudnn765\bin”三項，具體的路徑可能依據實際路徑略有出入，如圖

1.34 所示，確認無誤後依次點擊確定，關閉所有對話方塊。

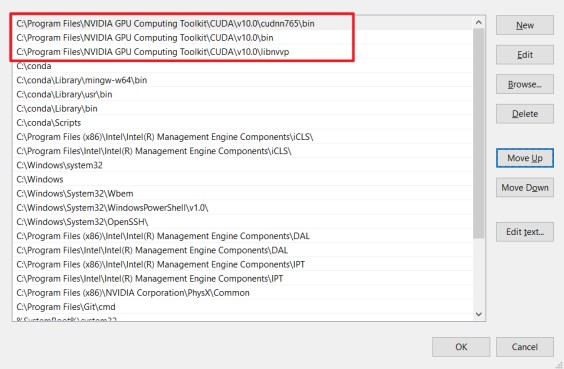


圖 1.34 CUDA 相關的環境變數

**1.6.3 TensorFlow** 安裝

TensorFlow 和其他的 Python 庫一樣，使用 Python 包管理工具 pip install 命令即可安 裝。安裝 TensorFlow 時，需要根據電腦是否具有 NVIDIA GPU 顯卡來確定是安裝性能更強 的 GPU 版本還是性能一般的 CPU 版本。

國內使用 pip 命令安裝時，可能會出現下載速度緩慢甚至連接斷開的情況，需要配置 國內的 pip 源，只需要在 pip install 命令後面帶上“-i 源地址”參數即可。例如使用清華源 安裝 numpy 包，首先打開 cmd 命令列程式，輸入：

# 使用中國清華源安裝 numpy

pip install numpy -i https://pypi.tuna.tsinghua.edu.cn/simple

即可自動下載並安裝 numpy 庫，配置上國內源的 pip 下載速度會提升顯著。 現在我們來 TensorFlow GPU 最新版本，命令如下：

# 使用清華源安裝 TensorFlow GPU 版本

pip install -U tensorflow-gpu -i https://pypi.tuna.tsinghua.edu.cn/simple

上述命令自動下載 TensorFlow GPU 版本並安裝，目前是 TensorFlow 2.0.0 正式版，“-U” 參數指定如果已安裝此包，則執行升級命令。

現在我們來測試 GPU 版本的 TensorFlow 是否安裝成功。在 cmd 命令列輸入 ipython 進

入 ipython 互動式終端，輸入“import tensorflow as tf”命令，如果沒有錯誤產生，繼續輸

入“tf.test.is\_gpu\_available()”測試 GPU 是否可用，此命令會列印出一系列以“I”開頭的 資訊(Information)，其中包含了可用的 GPU 顯卡設備資訊，最後會返回“True”或者

“False”，代表了 GPU 設備是否可用，如圖 1.35 所示。如果為 True，則 TensorFlow GPU

版本安裝成功；如果為 False，則安裝失敗，需要再次檢測 CUDA、cuDNN、環境變數等 步驟，或者複製錯誤，從搜尋引擎中尋求幫助。

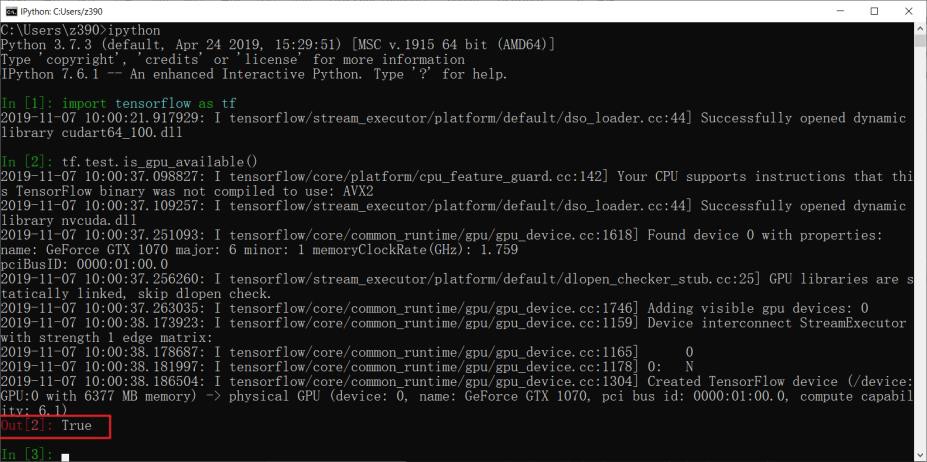


圖 1.35 TensorFlow-GPU 安裝結果測試

如果不能安裝 TensorFlow GPU 版本，則可以安裝 CPU 版本。CPU 版本無法利用 GPU 加速運算，計算速度相對緩慢，但是作為學習用途所介紹的演算法模型一般不大，使用 CPU 版本也能勉強應付，待日後對深度學習有了一定瞭解再升級 NVIDIA GPU 設備也未嘗不 可。亦或者，安裝 TensorFlow GPU 版本可能會出現安裝失敗的情況，很多讀者朋友動手能 力欠缺，如果折騰了很久還不能搞定，可以直接安裝 CPU 版本。

安裝 CPU 版本的命令為：

# 使用中國清華源安裝 TensorFlow CPU 版本

pip install -U tensorflow -i https://pypi.tuna.tsinghua.edu.cn/simple

安裝完後，在 ipython 中輸入“import tensorflow as tf”命令即可驗證 CPU 版本是否安裝成 功。

TensorFlow GPU/CPU 版本安裝完成後，可以通過“tf. version ”查看本地安裝的

TensorFlow 版本號，如圖 1.36 所示。

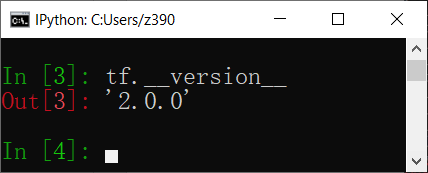


圖 1.36 TensorFlow 版本測試

上述手動安裝 CUDA 和 cuDNN，配置 Path 環境變數並安裝 TensorFlow 的流程是標準 的安裝方法，雖然步驟繁瑣，但是對於理解每個庫的功能角色有較大的幫助。實際上，對 于新手來說，可以將手動安裝 CUDA 和 cuDNN，配置 Path 環境變數並安裝 TensorFlow 這

4 大步驟通過兩條命令完成：

# 創建名為 tf2 的虛擬環境，並根據預設環境名 tensorflow-gpu

# 自動安裝 CUDA,cuDNN,TensorFlow GPU 等

conda create -n tf2 tensorflow-gpu

# 啟動 tf2 虛擬環境

conda activate tf2

這種快捷安裝方式稱為極簡版安裝方法。這也是使用 Anaconda 發行版本所帶來的便捷 之處。通過極簡版安裝的 TensorFlow，使用時需要先啟動對應的虛擬環境，這一點需要與 標準版區分。標準版安裝在 Anaconda 的默認環境 base 中，一般不需要手動啟動 base 環 境。

常用的 Python 庫也可以順帶安裝，命令如下：

# 使用清華源安裝常用 python 庫

pip install -U ipython numpy matplotlib pillow pandas - i https://pypi.tuna.tsinghua.edu.cn/simple

TensorFlow 在運行時，默認會佔用所有 GPU 顯存資源，這是非常不友好的行為，尤其 是當電腦同時有多個使用者或者程式在使用 GPU 資源時，佔用所有 GPU 顯存資源會使得

其他程式無法運行。因此，一般推薦設置 TensorFlow 的顯存佔用方式為增長式佔用模式，

即根據實際模型大小申請顯存資源，代碼實現如下：

# 設置 GPU 顯存使用方式

# 獲取 GPU 設備清單

gpus = tf.config.experimental.list\_physical\_devices('GPU')

if gpus:

try:

# 設置 GPU 為增長式佔用

for gpu in gpus:

tf.config.experimental.set\_memory\_growth(gpu, True)

except RuntimeError as e:

# 列印異常

print(e)

**1.6.4** 常用編輯器安裝

使用 Python 語言編寫程式的方式非常多，可以使用 **ipython** 或者 **ipython notebook** 方式 互動式編寫代碼，也可以利用 **Sublime Text**、**PyCharm** 和 **VS Code** 等綜合 IDE 開發中大型 專案。本書推薦使用 PyCharm 編寫和調試，使用 VS Code 互動式開發，這兩者都可以免費 使用，用戶自行下載安裝，並配置 Python 解譯器。限於篇幅，不再贅述。

接下來，讓我們開啟深度學習之旅吧！

1.7 參考文獻

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M.

Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg 和 D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature,* 卷 518, pp. 529-533, 2 2015.