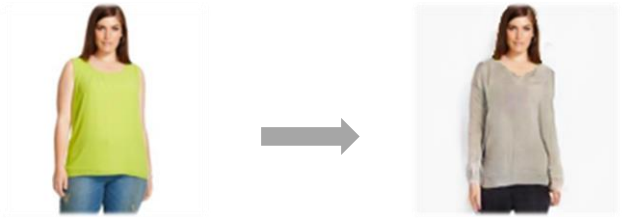

人工智能概論與 Python 入門
商汤
sensetime

人工智能驅動未來

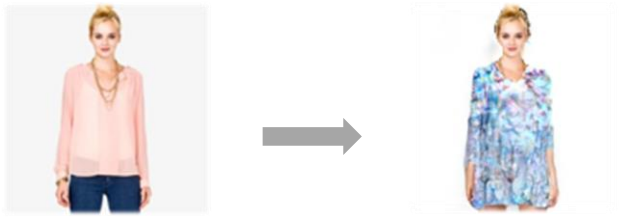
衣

用文字生成時裝

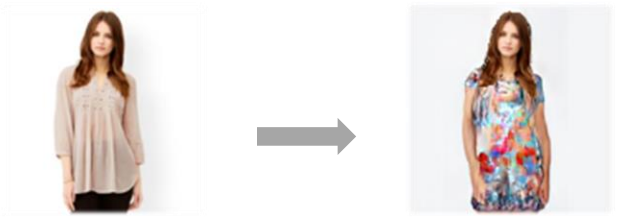




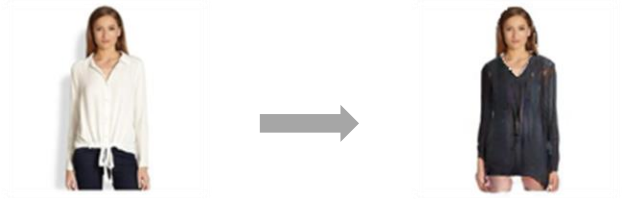
我要穿一件長袖淺灰毛衣



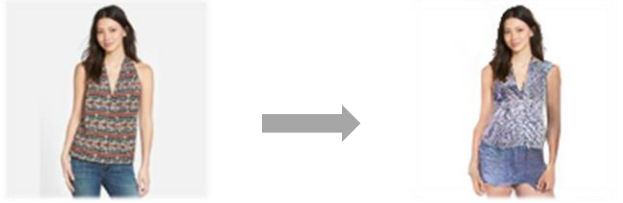
我要穿一件淺藍花案連衣裙



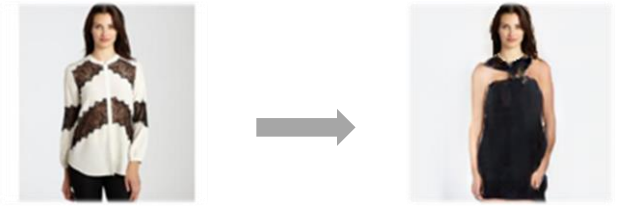
我要穿一件多色短袖連衣裙



我要穿一件黑色長袖開衫



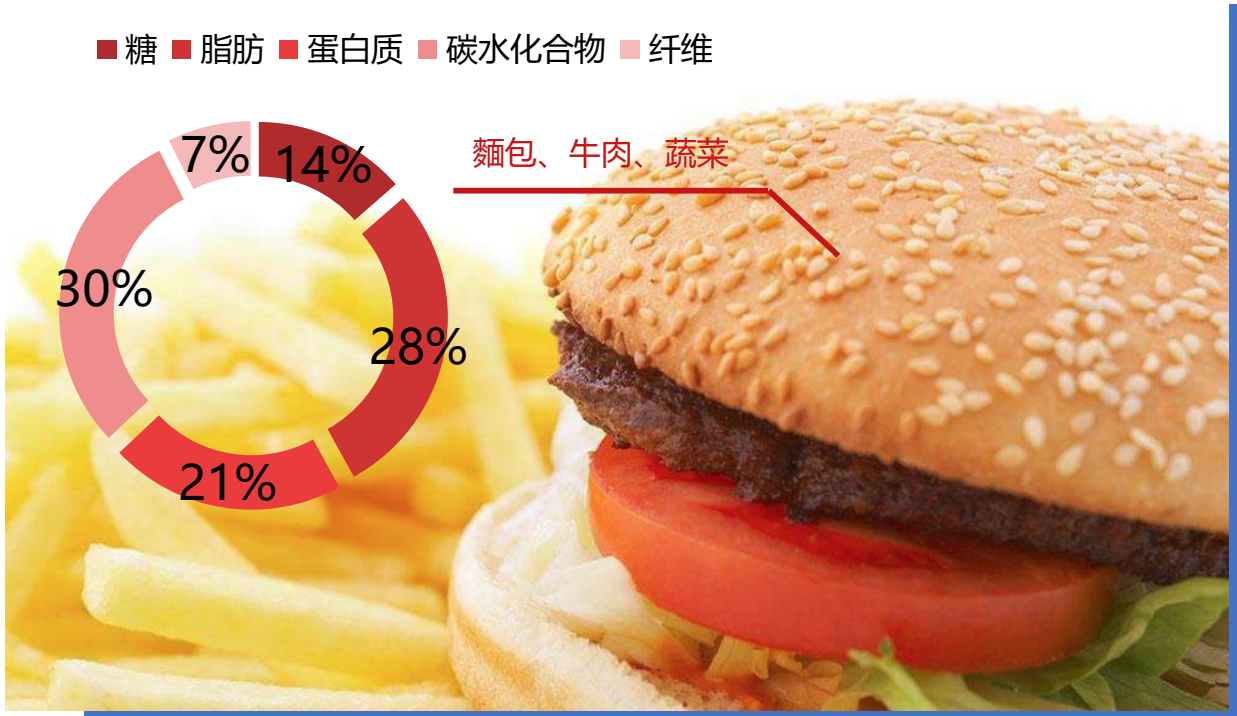
我要穿一件藍色的無袖衣服和短裙



我要穿一件黑色無袖外套的連衣裙

人工智能驅動未來

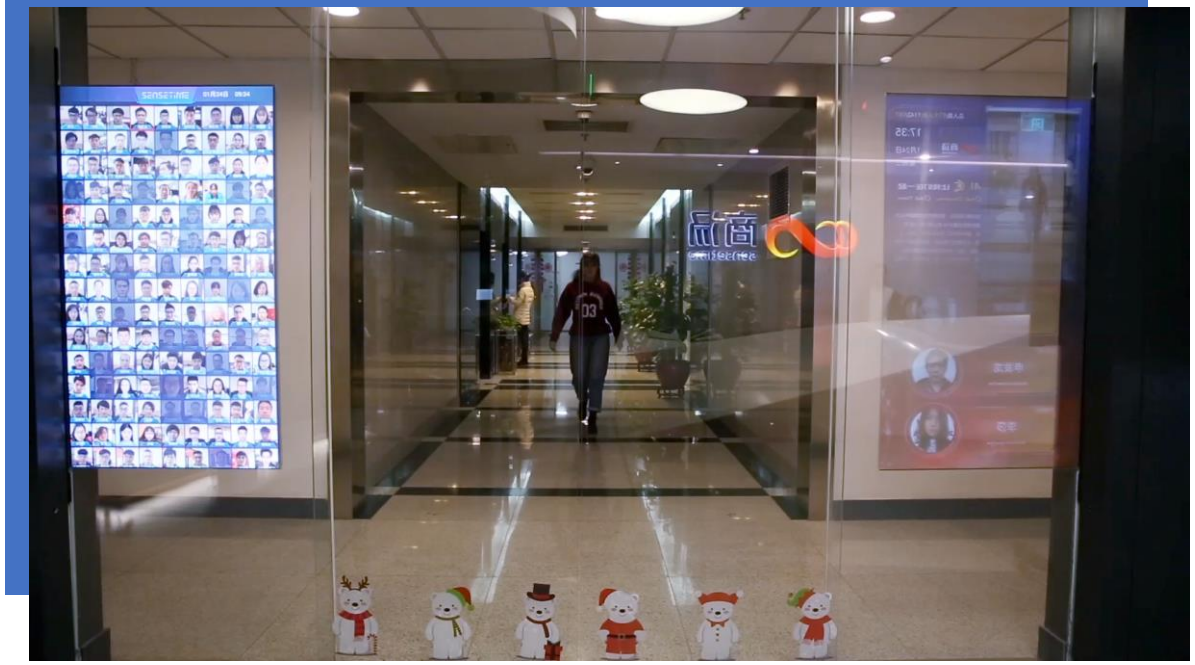
食



人工智能驅動未來

商汤 sensetime

住



人工智能驅動未來

商汤 sensetime

行

人工智能時代

18世紀60年代到19世紀中期

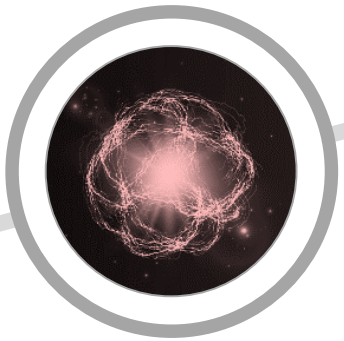
蒸汽機

蒸汽時代

19世紀下半葉到20世紀初

直流電
交流電

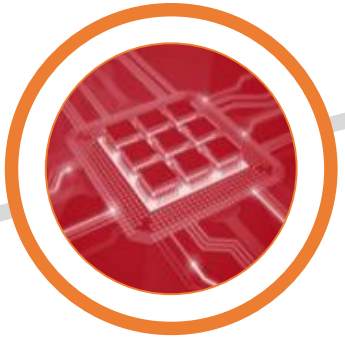
電力時代



20世紀後半期

計算機
互聯網

資訊時代



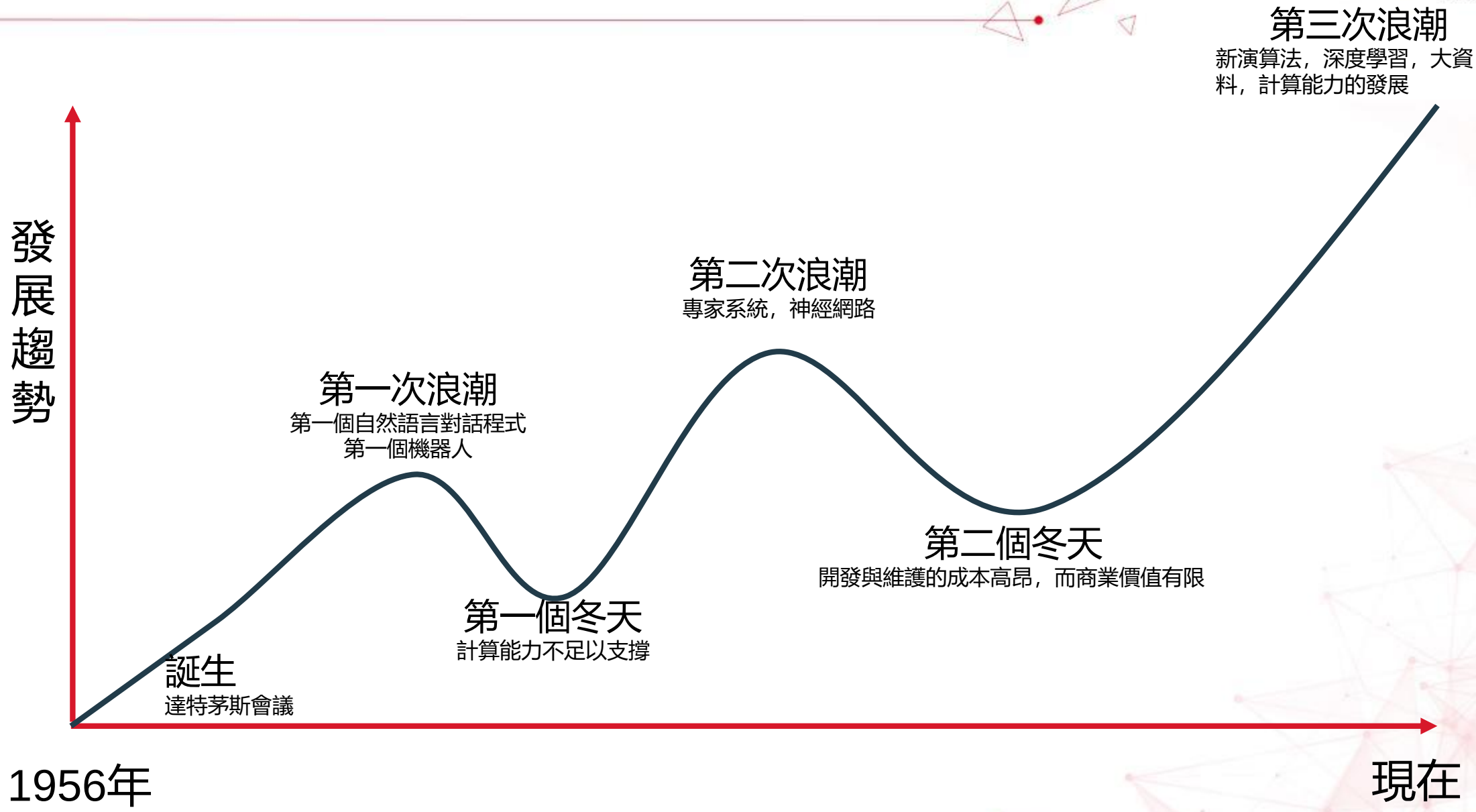
21世紀開始

深度
學習

AI時代

2016年是人工智能的商業元年。AI應用走出實驗室，並廣泛走進我們的日常生活。

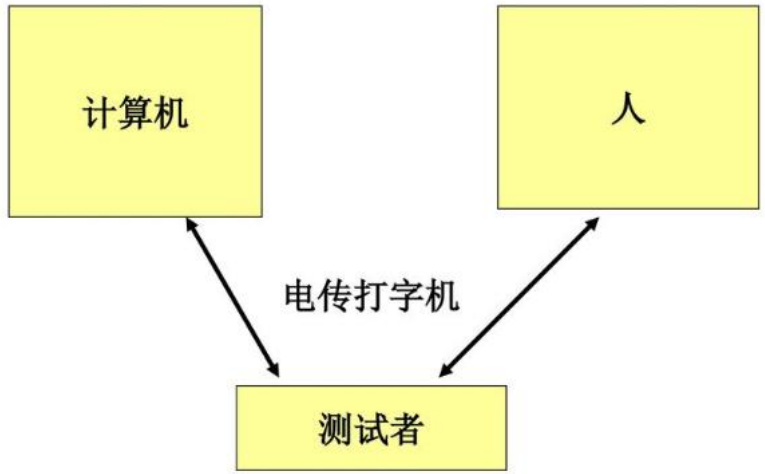
光輝歲月： 人工智能簡史


第三次浪潮
新演算法，深度學習，大資料，計算能力的發展
發展趨勢
第二次浪潮
專家系統，神經網路
第一次浪潮
第一個自然語言對話程式
第一個機器人
第二個冬天
開發與維護的成本高昂，而商業價值有限
第一個冬天
計算能力不足以支撐
誕生
達特茅斯會議
1956年
現在

圖靈實驗

- 1950年，艾倫·圖靈（Alan Turing，被譽為電腦科學之父）在他的論文《電腦器與智慧》提出了著名的**圖靈試驗**。
- 進行多次測試後，如果有超過30%的測試者不能確定出被測試者是人還是機器，那麼這台機器就通過了測試，並被認為具有**人類智能**。

艾倫·圖靈（1912—1954）



圖靈實驗

第一個人工神經網路

- 1951年夏天，當時普林斯頓大學數學系的一位24歲的研究生馬文·閔斯基（Marvin Minsky，1927-2015）建立了**世界上第一個人工神經網路**。

- 馬文·閔斯基，最早聯合提出了“人工智能”概念，被尊為**人工智能之父**。人工智能領域首位圖靈獎獲得者，虛擬實境最早宣導者，也是世界上第一個人工智能實驗室MIT人工智能實驗室聯合創始人（1959年）。

人工智能學科誕生

- 1956年8月，在美國漢諾斯小鎮寧靜的**達特茅斯**學院中組織了一次討論會。 正式宣告了人工智能作為一門學科的誕生。

達特茅斯會議

會議原址：達特茅斯樓

第一個自然語言對話程式

- 1964到1966年間，麻省理工學院的教授建立了世界上**第一個自然語言對話程式ELIZA**，該程式通過簡單的模式匹配和對話規則與人聊天。

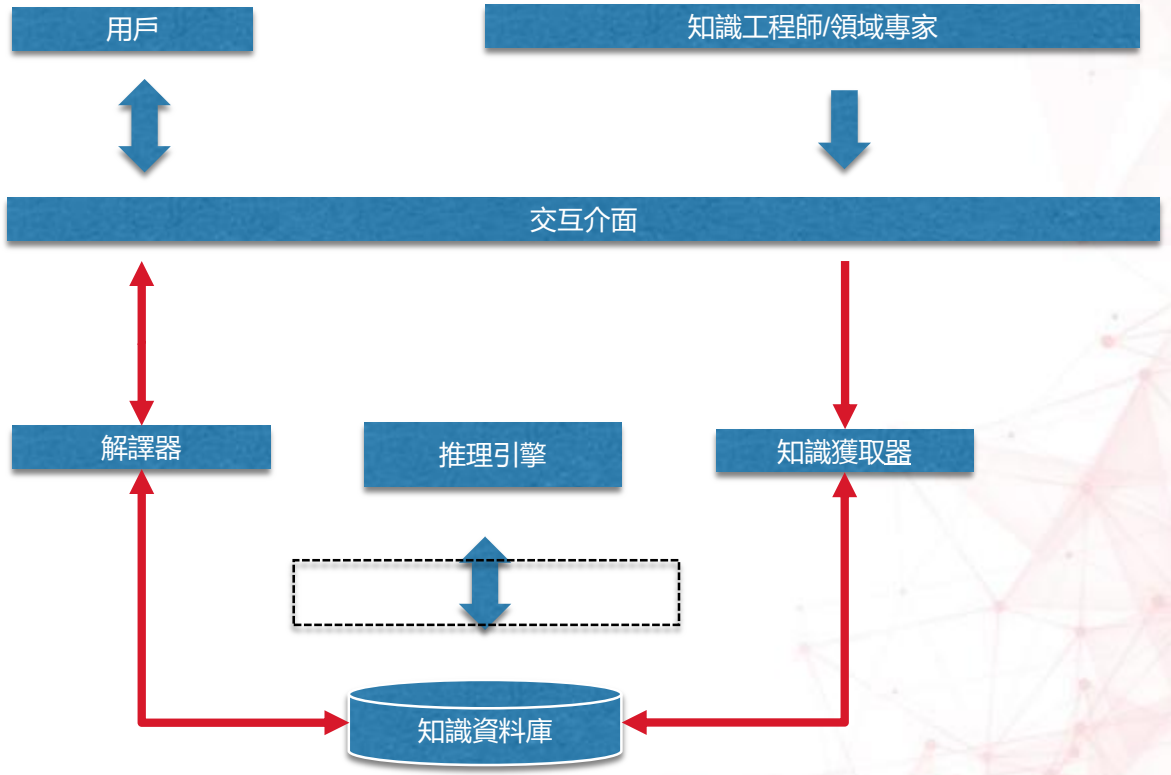
- 儘管ELIZA是一個腳本引擎，可以載入不同的腳本來進行對話。但是最為人們所熟知的還是使用一個**模擬羅傑斯式的心理治療師的腳本的ELIZA**，以致於提到ELIZA，就是特指能夠進行心理治療師式的對話的那個程式，就連Siri也說ELIZA是一位心理醫生，是她的啟蒙老師。

人工智能簡史 — 第二次浪潮（1980 -- 1987）：專家系統的興衰

專家系統

- 上世紀60年代，愛德華·費根鮑姆（Edward Feigenbaum）已經開始了對專家系統的早期研究。

愛德華·費根鮑姆（專家系統之父）

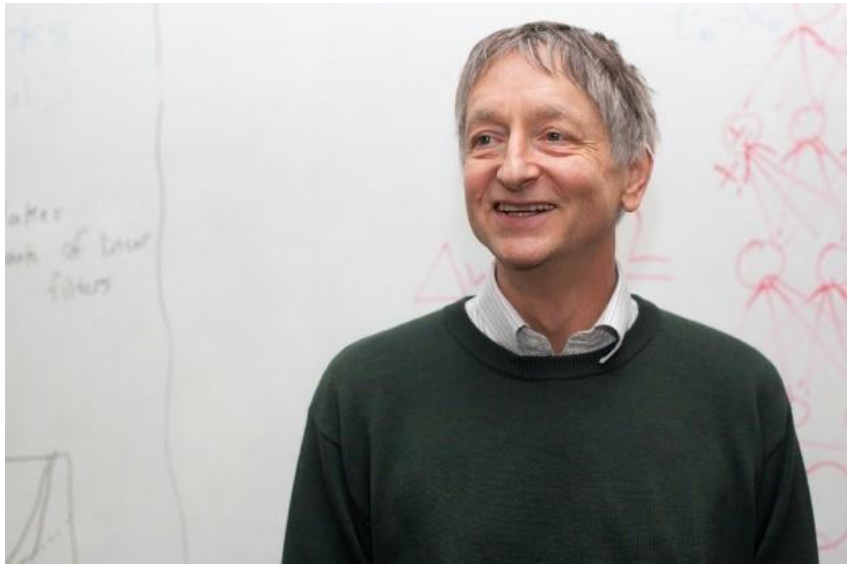


專家系統示意圖

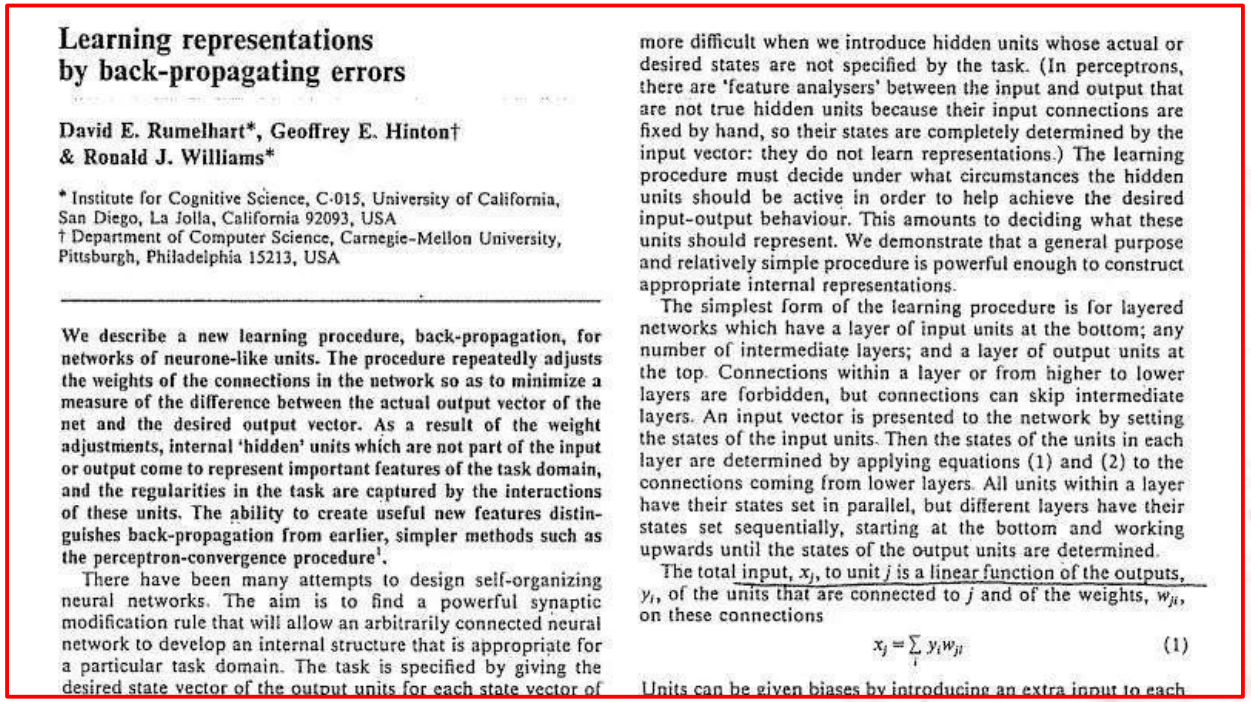
人工智能簡史 — 第二次浪潮（1980 -- 1987）：專家系統的興衰

人工神經網路的進展

- 1986年，大衛·魯梅爾哈特（David Rumelhart），傑佛瑞·辛頓（Geoffrey Hinton），和羅奈爾得·威廉姆斯（Ronald Williams）聯合發表了有里程碑意義的經典論文，提出了反向傳播演算法。



傑佛瑞·辛頓（1947- ）



Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of

more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input-output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \qquad (1)$$

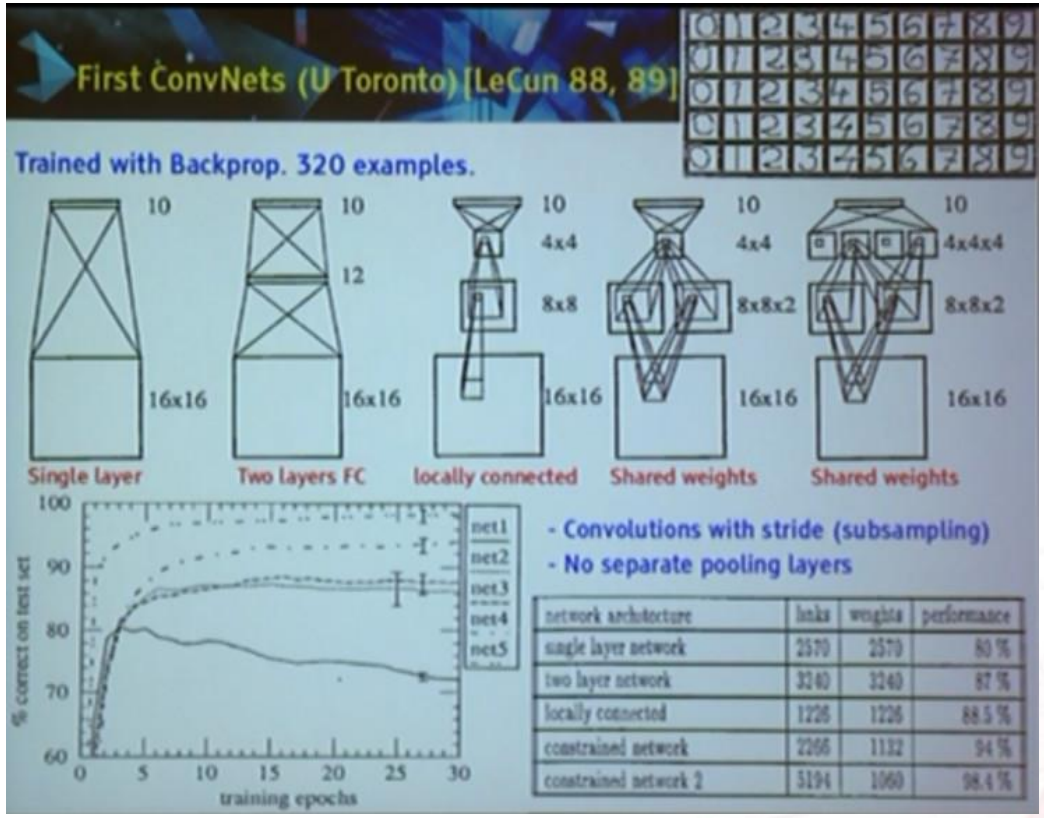
Units can be given biases by introducing an extra input to each

第二次寒冬

- 但是專家系統的實用性僅僅局限於某些特定情景，不久後人們對專家系統的狂熱追捧轉向巨大的失望。
- 另一方面，1987年到1993年現代PC的出現，其費用遠遠低於專家系統所使用的Symbolics和Lisp等機器。
- 相比于現代PC，專家系統被認為古老陳舊而非常難以維護。於是，政府經費開始下降，寒冬又一次來臨。

人工智能簡史

1989年，AT&T貝爾實驗室的雅恩·樂昆（Yann LeCun）和團隊使用卷積神經網路技術，實現了人工智能識別手寫的郵遞區號數位圖像。



人工智能簡史

2006年，傑佛瑞辛頓（Geoffrey Hinton）發表了《Learning Multiple Layers of Representation》奠定了後來神經網路的全新的架構，至今仍然是人工智能深度學習的核心技術。

Review TRENDS in Cognitive Sciences Vol.11 No.10

Learning multiple layers of representation

Geoffrey E. Hinton

Department of Computer Science, University of Toronto, 10 King's College Road, Toronto, M5S 3G4, Canada

To achieve its impressive performance in tasks such as speech perception or object recognition, the brain extracts multiple levels of representation from the sensory input. Backpropagation was the first computationally efficient model of how neural networks could learn multiple layers of representation, but it required labeled training data and it did not work well in deep networks. The limitations of backpropagation learning can now be overcome by using multilayer neural networks that contain top-down connections and training them to generate sensory data rather than to classify it. Learning multilayer generative models might seem difficult, but

digits, the complicated nonlinear features in the top layer enable excellent recognition of poorly written digits like those in Figure 1b [2].

There are several reasons for believing that our visual systems contain multilayer generative models in which top-down connections can be used to generate low-level features of images from high-level representations, and bottom-up connections can be used to infer the high-level representations that would have generated an observed set of low-level features. Single cell recordings [3] and the reciprocal connectivity between cortical areas [4] both suggest a hierarchy of progressively more complex features

人工智能簡史

2007年，在斯坦福任教的華裔科學家李飛飛，發起創建了ImageNet項目。

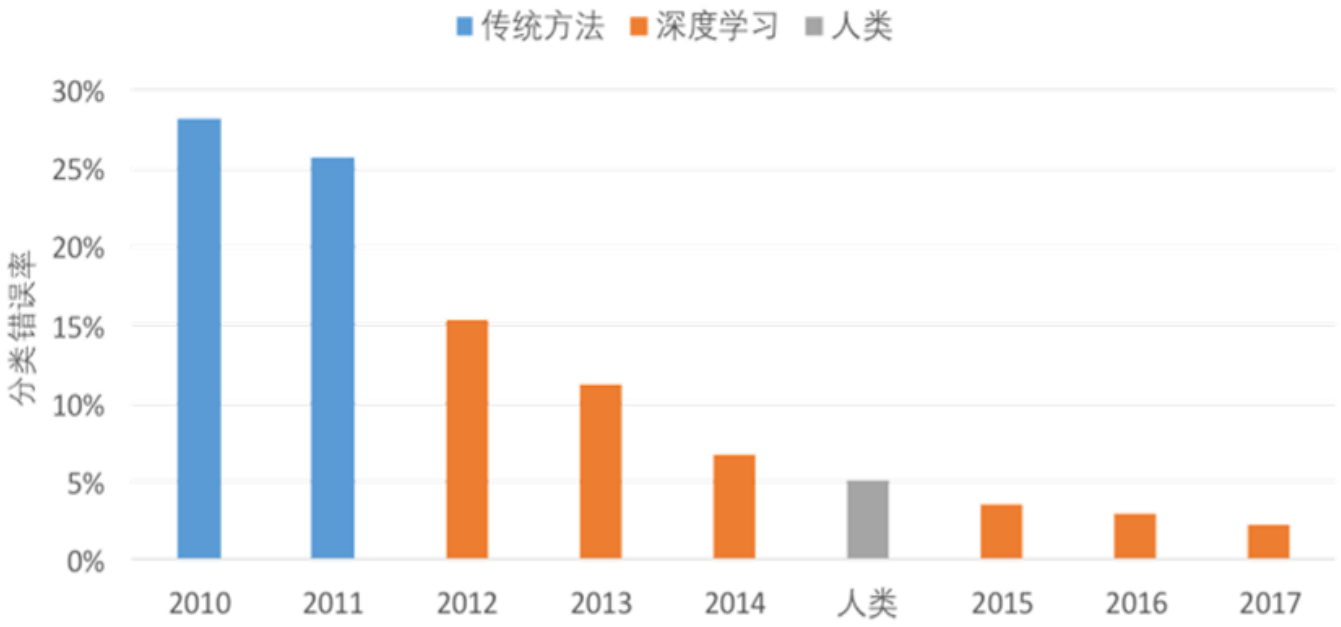
ImageNet目前已經包含了1500萬張圖片資料，超過2萬個類別。

人工智能簡史

華裔科學家吳恩達（Andrew Ng）及其團隊在2009年開始研究使用圖形處理器（GPU而不是CPU）進行大規模無監督式機器學習工作，嘗試讓人工智能程式完全自主的識別圖形中的內容。

人工智能簡史 — 第三次浪潮（2011至今）：厚積薄發，再造輝煌

- 2012年，Geoff Hinton的團隊在大型視覺識別的挑戰中擊敗穀歌。
- 2016年3月，穀歌DeepMind研發的AlphaGo在圍棋人機大戰中擊敗韓國職業九段棋手李世石。
- 經過短短3天的自我訓練， AlphaGo Zero就強勢打敗了此前戰勝李世石的舊版AlphaGo，戰績是100:0。

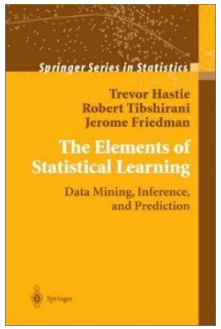


2016年3月9日

2017年10月19日

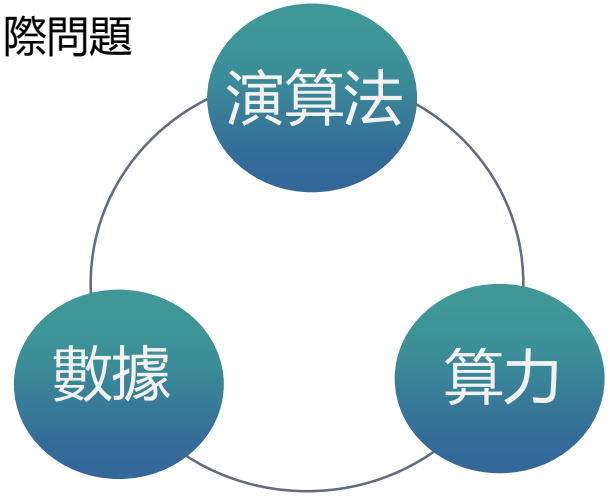
人工智能簡史 — 第三次浪潮（2011至今）：厚積薄發，再造輝煌

演算法，資料和算力的突破，支撐人工智能第三次浪潮



不同學科的數學工具開始引入，一大批新的數學模型和演算法被發展起來（比如，統計學習理論，支持向量機，概率圖模型等）。新發展的智慧演算法被逐步應用於解決實際問題

全球化的加速以及互聯網的蓬勃發展帶了全球範圍電子資料的爆炸性增長。人類邁入了“大資料”時代。



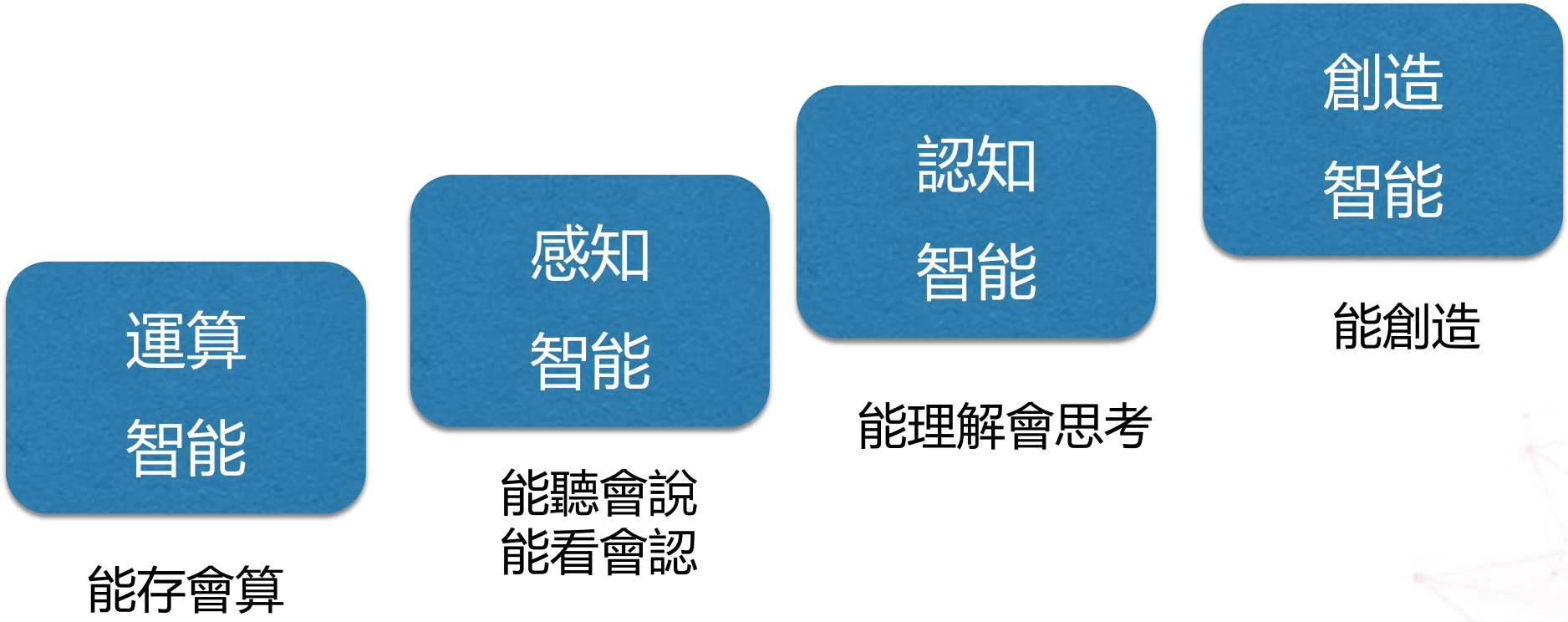
電腦晶片的計算能力持續高速增長。當代一塊最新圖形處理器的計算能力已經突破了每秒10萬億次浮點運算，超過了2001年全球最快的超級電腦。

大數據時代

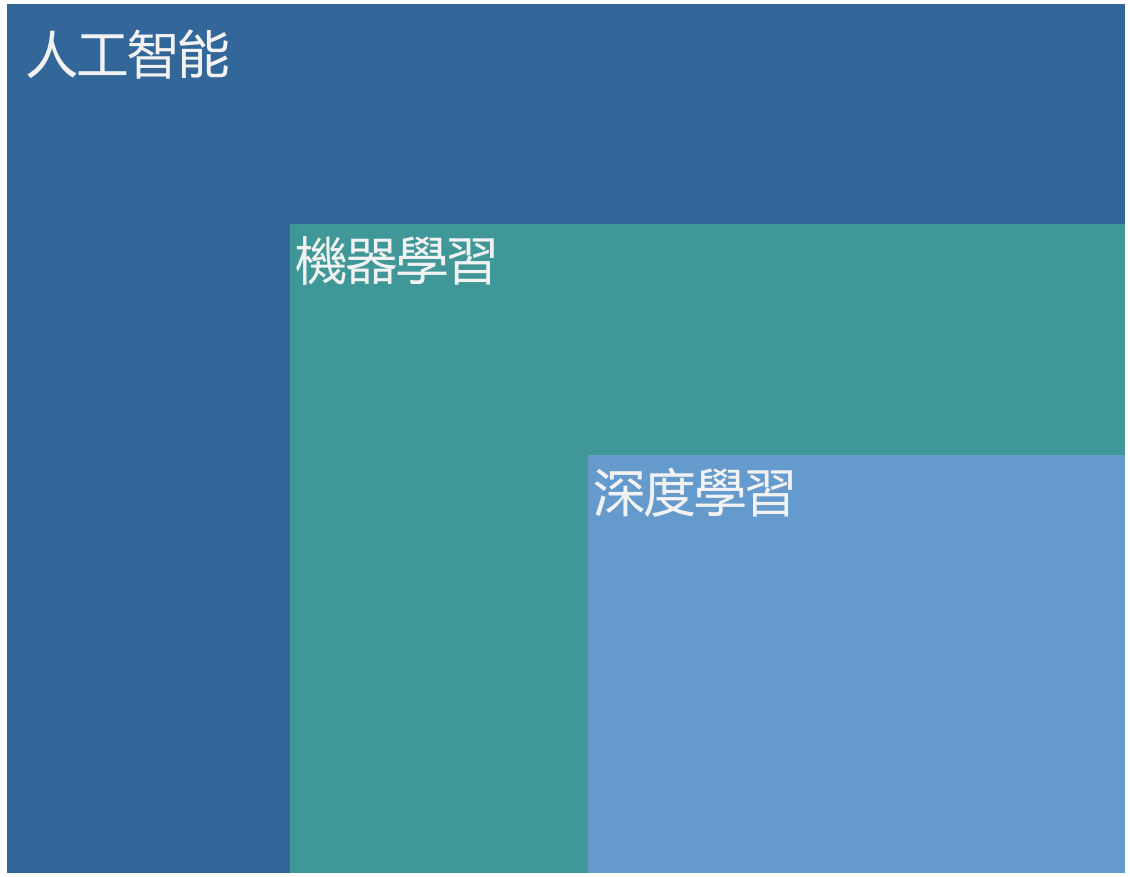
- 一分鐘內，微博推特上新發的資料量超過10萬；社交網路“臉譜”的流覽量超過600萬.....
- 這些龐大數字，意味著什麼?
- 它意味著，一種全新的致富手段也許就擺在面前，它的價值堪比石油和黃金。
- 事實上，當你仍然在把微博等社交平臺當作抒情或者發議論的工具時，華爾街的斂財高手們卻正在挖掘這些互聯網的“資料財富”，先人一步用其預判市場走勢，而且取得了不俗的收益。

人工智能技術

人工智能的目的是使得電腦能聽、會說、理解語言、會思考、解決問題，甚至會創造。

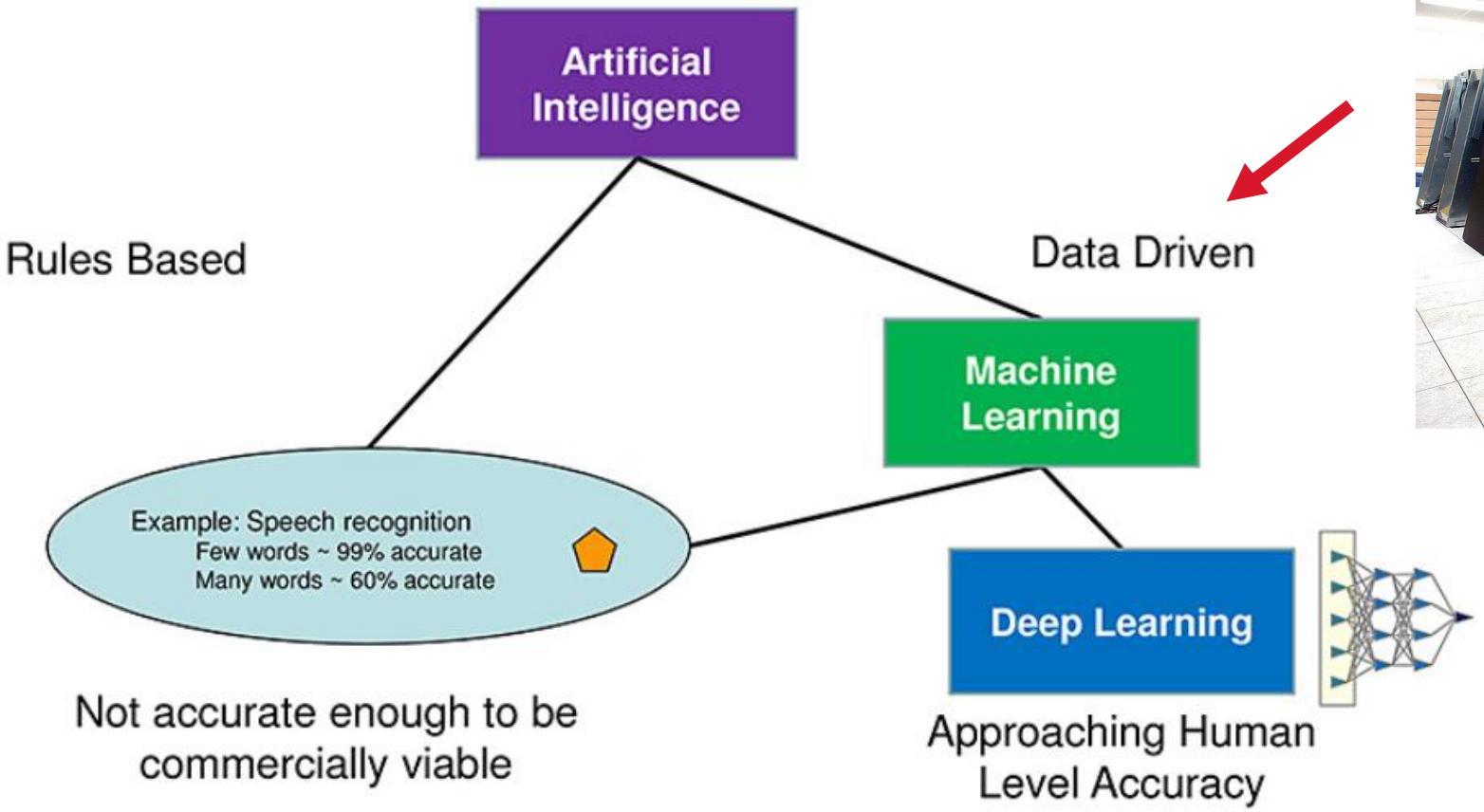


人工智能>機器學習>深度學習



目前核心技術:
人工神經網絡

目前人工智能主要方法



資料和算力的需求

深度學習

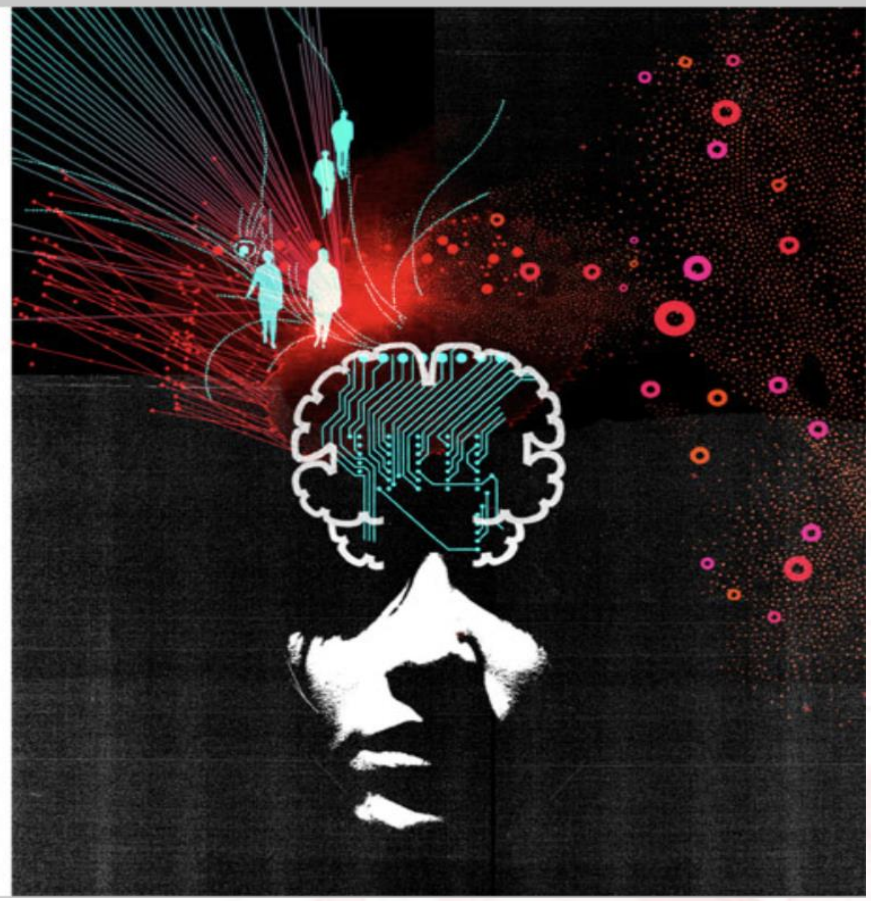
MIT Technology Review

10 BREAKTHROUGH TECHNOLOGIES 2013

Introduction The 10 Technologies Past Years

Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.



深度學習帶來各項人工智能技術的突破

人臉識別

語音辨識

圖像識別

深度學習
使機器能夠從大量資料中獲取知識和技能

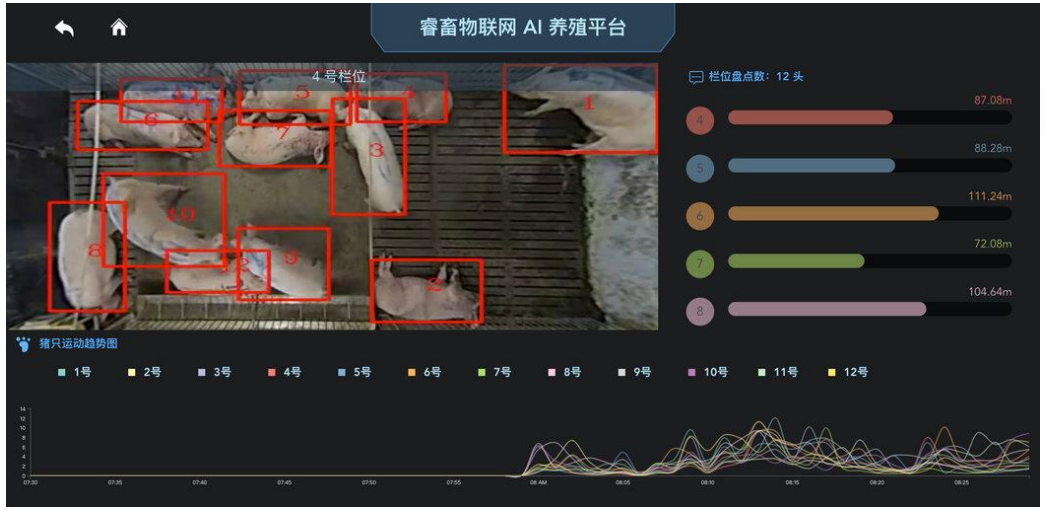
圍棋博弈

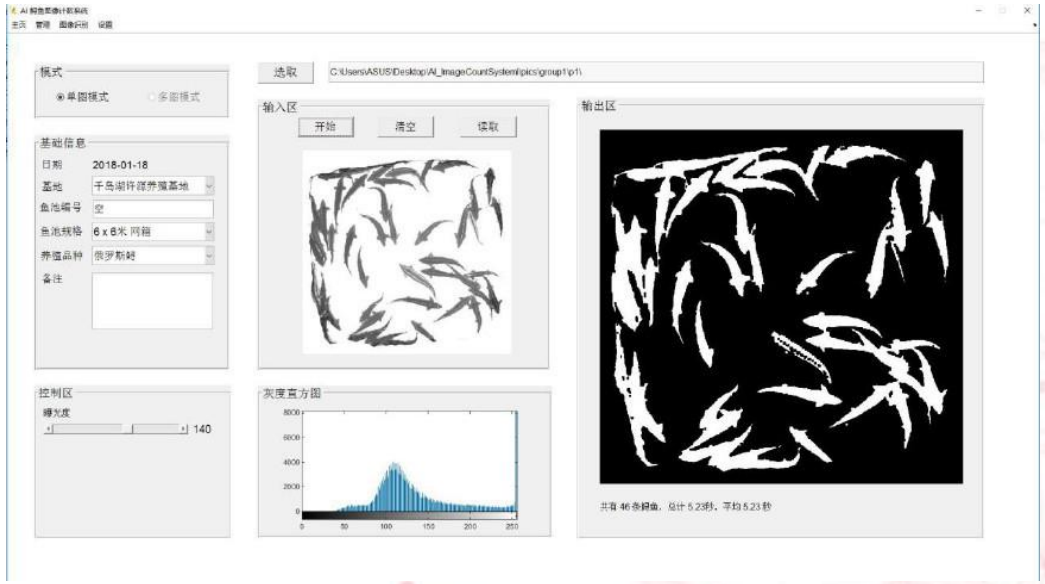
自然語言處理

自動駕駛

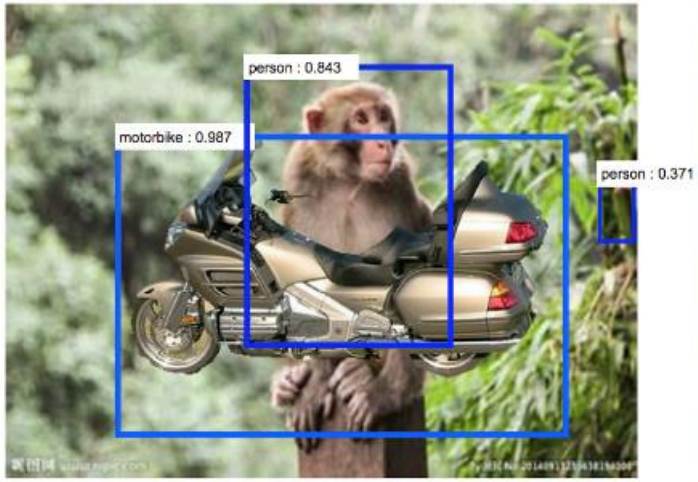
智慧投顧、交易機器人

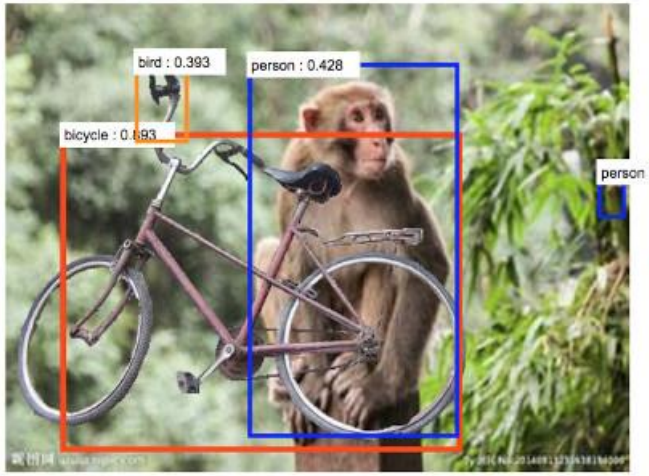
深度學習帶來各項人工智能技術的突破

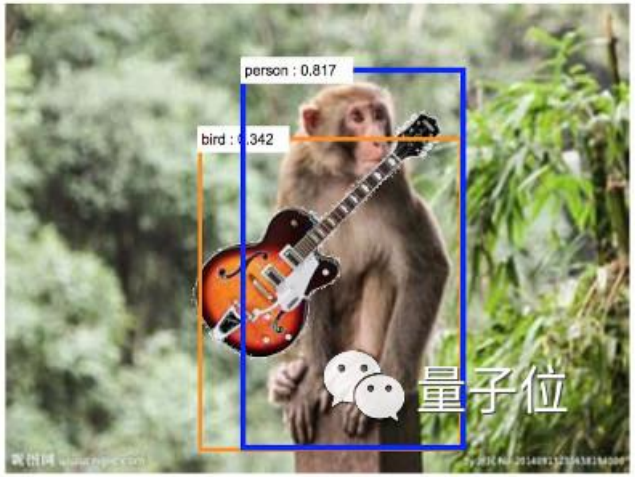




深度學習的挑戰

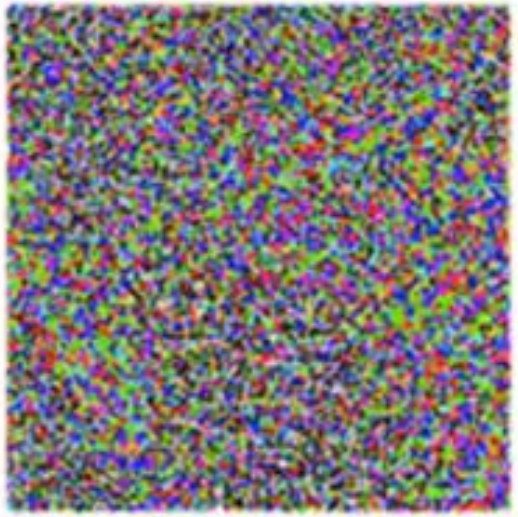






‘Duck’

+

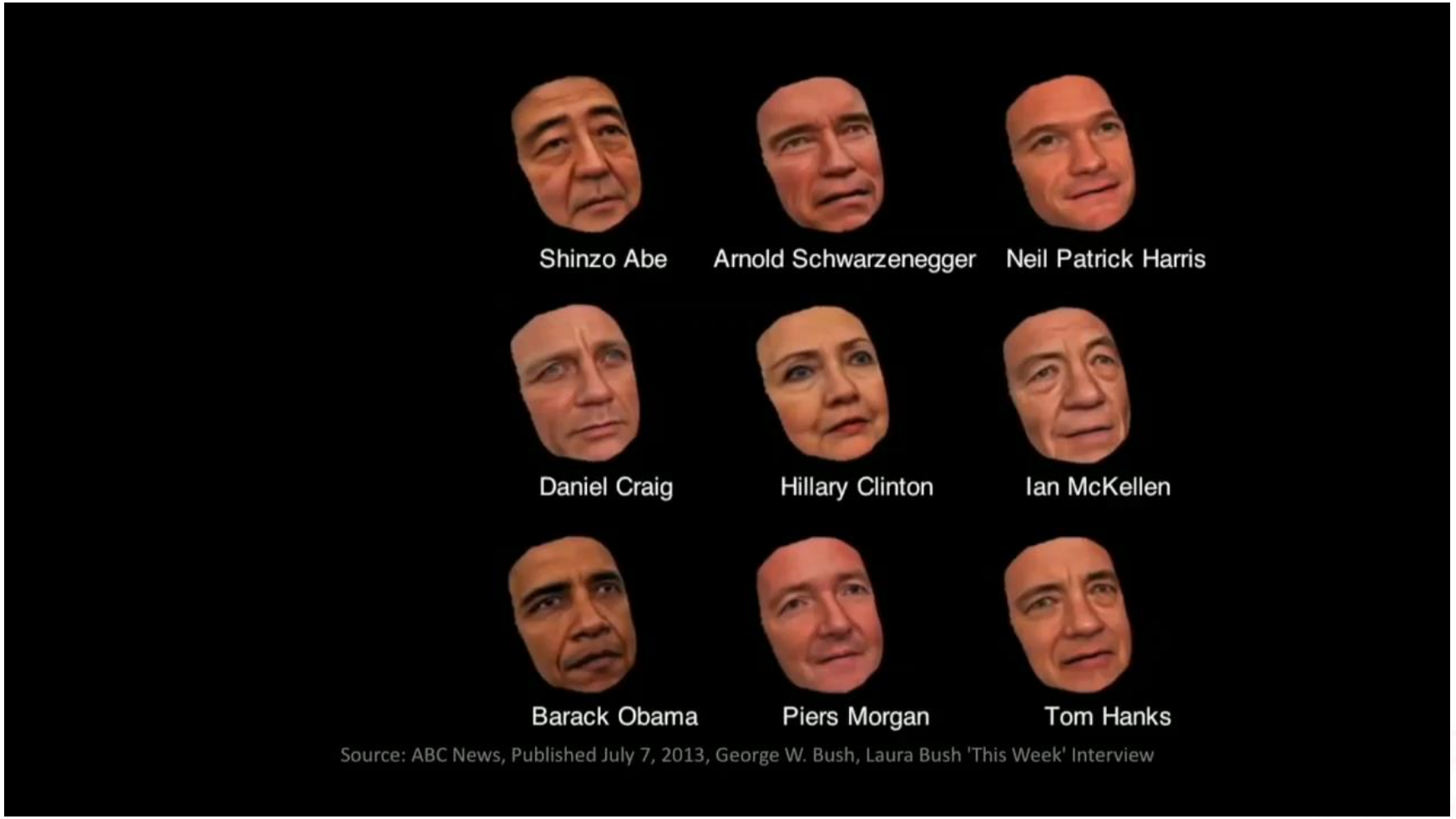


×0.07

=

‘Horse’

深度學習的挑戰


Shinzo Abe
Arnold Schwarzenegger
Neil Patrick Harris
Daniel Craig
Hillary Clinton
Ian McKellen
Barack Obama
Piers Morgan
Tom Hanks
Source: ABC News, Published July 7, 2013, George W. Bush, Laura Bush 'This Week' Interview

深度學習的挑戰

深度學習的挑戰

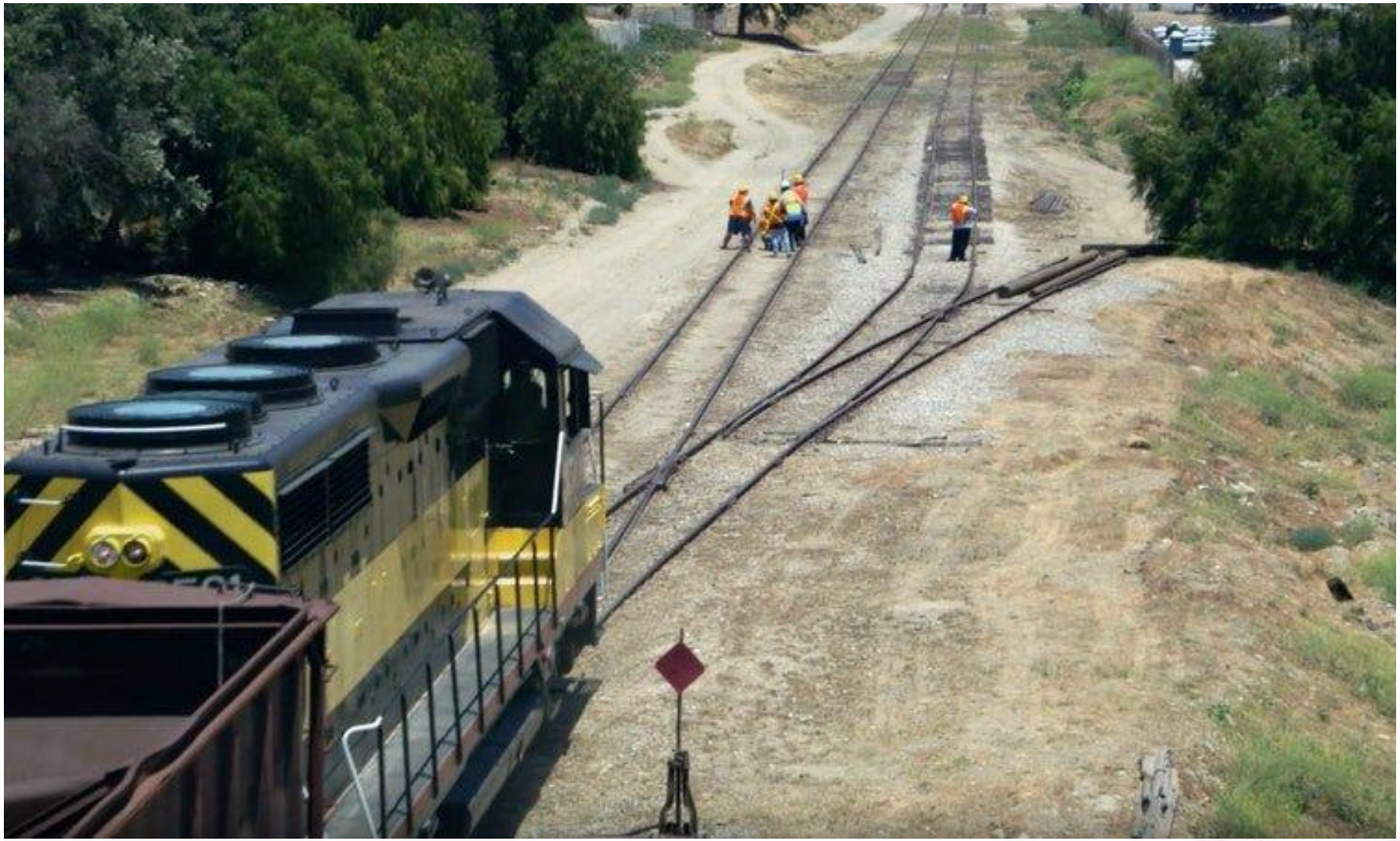
Consumption	CO_2e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	31
w/ tuning & experimentation	52,909
Transformer (big)	192
w/ neural architecture search	626,155

隨著對深度學習訓練結果的要求越來越高，在其計算過程中對環境的影響日益顯著。

某些新的模型/訓練方式取得了最優結果，但是進展微小，副作用卻是不成比例的計算量/二氧化碳排放量的增加。

如今訓練一個普通模型可以達到發論文的水平，中間的計算過程折算的二氧化碳排放量接近或超過普通汽車整個壽命周期的二氧化碳排放量。

深度學習的挑戰



人工智能領域最受歡迎的語言之一——Python

Python是一種通用的高級程式設計語言，它是一種解釋型語言，強調**代碼的可讀性**以及**簡單清晰的程式結構**。

Python可以讓開發者靈活簡潔地表達想法，而不需要在底層執行細節上花費很多精力。

因此，Python廣受歡迎，已經被廣泛應用於電腦視覺、語音辨識、自然語言理解、資料發掘、深度學習研究等重要方向。

Python中的變量

變量的定義：變量是可以存儲值的記憶體空間。當一個變量被賦值（=符號）的時候，它就被創建了。例如:

```
counter = 100   # 整數賦值創建了一個整型變量，其值為100
name = “John”   # 創建了一個字串類型變量，其值為“John”。字串用單引號'括起亦可。
```

變量的類型：變量是有不同的類型的。比如，在上面的例子中，counter表示一個數值，name表示一串文字，因此它們是屬於不同的類型的。

Python語言提供了以下幾種主要的變量類型:

數值(number)，**字串**(string)，**列表**(list)，**字典**(dict)，以及**集合**(set)。另一個常見的類型是**布林**（boolean），它只有True和False兩個值，代表條件是否成立。

Python中的縮進(indentation)

Python中一行代碼行首的空白是重要的，它稱為縮進。此空白（空格和定位字元）用來決定邏輯行的縮進層次，從而用來決定語句的分組。同一層次的語句必須有相同的縮進。每一組這樣的語句稱為一個塊。錯誤的縮進會引發錯誤。例如:

```
i = 5
  print('Value is', i)     # 縮進錯誤!
print('I repeat, the value is', i)
```

運行左邊這段代碼的時候，你會得到錯誤：Syntax Error: invalid syntax

```
def test():
      if True:
        print("a")
      else:
        print("b")
```

python這種強制的代碼縮進，好處在於在嚴格要求的代碼縮進之下，代碼非常整齊規範，賞心悅目，提高了可讀性，在一定程度上也提高了可維護性。

Python語言常見運算符

運算符	描述	示例
+	加法	1 + 1 結果是 2
-	減法	5 – 2 結果是 3
*	乘法	2 * 5 結果是 10
/	除法	5 / 2 結果是 2.5
%	取餘數	7 % 3 結果是 1
=	設定運運算元。把賦值號右邊的值賦給符號左邊的變量	a = 2 * 6
and	邏輯與運運算元。如果兩個運算元都為True，結果為True；否則為False	（True and False）結果是 False
or	邏輯或運運算元。如果兩個運算元中有一個非0，結果為True；否則為False	（True or False）結果是 True

Python程式中的注釋

寫程式時會在代碼旁邊記錄設計的思路或者其它相關資訊，但是又不想影響程式的執行，這時可以使用**注釋**。

如下所示，注釋的部分一般由一個 # 符號標注。**從 # 符號開始到每行結束**，會被認為是注釋，而不被解釋執行。

```
# 將小車反向
def turn_back():
    turn_left()          # 小車左轉90度
    turn_left()          # 小車再次左轉90度
```

此外，夾在前後兩組 ’’’ 中間的內容也屬於被注釋的部分，代碼執行時此部分不被解釋執行。

Python程式中的列表(list)

列表是Python中最基本的資料結構之一。列表中的每個元素都分配一個數字，即它的位置或索引，第一個索引是0，第二個索引是1，依此類推。

創建一個列表，只要把逗號分隔的不同的資料項目使用方括號括起來即可。

使用下標索引來訪問清單中的值，同樣你也可以使用方括號的形式截取部分清單，如下所示：

```
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5, 6, 7 ]
print "list1[0]: ", list1[0]
print "list2[1:5]: ", list2[1:5]
```

此段代碼執行的結果為：

```
list1[0]:  physics
list2[1:5]:  [2, 3, 4, 5]
```

函數

在Python語言中，**函數**是一段**可以被重複執行的程式**。一個函數通常用於實現某種特定的功能。每個函數都有一個名字，稱為**函數名**。調用一個函數可以通過函數名加括弧的方式實現，如下：

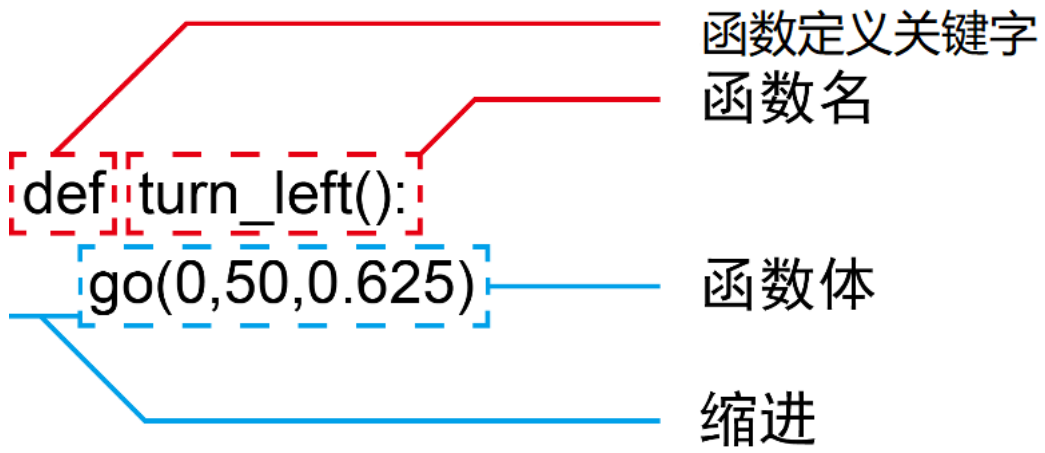
```
函數名()
```

Python的函數還可以通過**參數**讓調用者設定函數的行為。帶參數的函數，其調用方式是在括弧中加入指定的參數值。多個參數之間用逗號分隔，如下：

```
函數名(參數1)
函數名(參數1，參數2，...)
```

函數的認識

一個Python函數的定義由3個部分組成：**函數定義關鍵字**、**函數名**和**函數體**。



函式定義關鍵字	def	函式定義關鍵字(def)告訴電腦接下來要定義一個全新的函數。
函數名	turn_left()	函數名由自己確定，名字要儘量有意義，讓使用者能夠快速理解它的含義。
函數體	go(0, 50, 0.625)	函數體為實現該功能的語句，也就是turn_left()這個功能具體是由哪些指令來實現的。

函數的使用

完成了函數的定義之後，再想使用相同的功能時，可以直接**調用**這個新定義的函數。

要求：根據已有的函數turn_left()，定義向後轉的函數turn_back()。

實現：

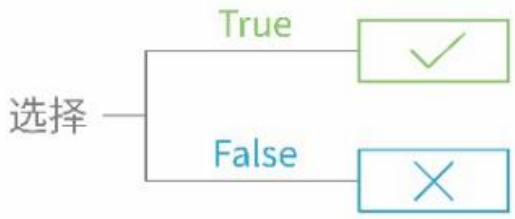
```
def turn_back():
    turn_left() # 小車左轉90度
    turn_left() # 小車再次左轉90度

turn_back()         # 調用函數
```

if語句 - 單分支結構

當感測器獲得的資料符合某種條件時，就會讓小車執行相應策略的指令，這樣的過程，在電腦中可以理解為：

如果：滿足某個條件

那麼：執行相應指令



圖：if語句的選擇

這就是一個簡單的條件結構，轉化為電腦語句就是：

```
if 條件:
    指令
```

如“當小車檢測到紅色時就向右轉”，這樣的要求可以通過下面的單分支結構語句來實現：

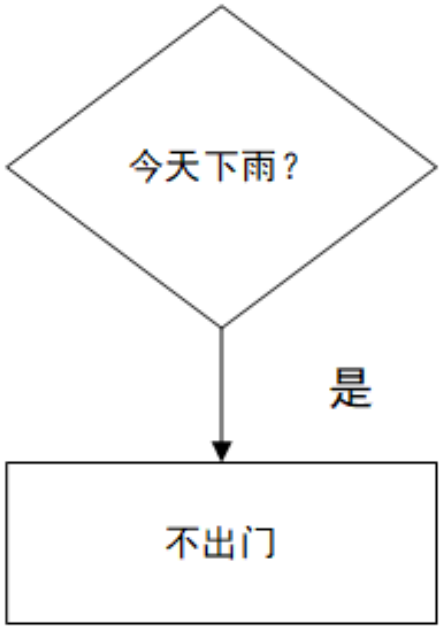
```
if get_color() == "red":
    turn_right()
```

注意，邏輯等於使用 == ，不等於使用 != 。

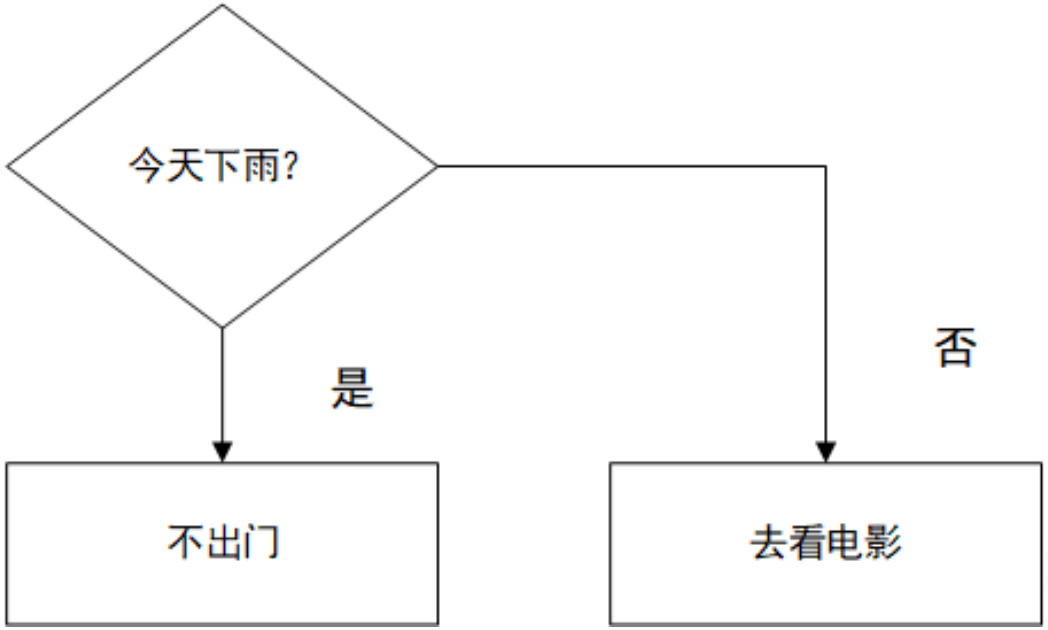
if/else語句 - 雙分支結構

現實中往往會遇到根據**一個條件是否滿足**，來執行**兩種不同指令**的情況，我們將這種情況稱為**雙分支結構。**

單分支結構



雙分支結構

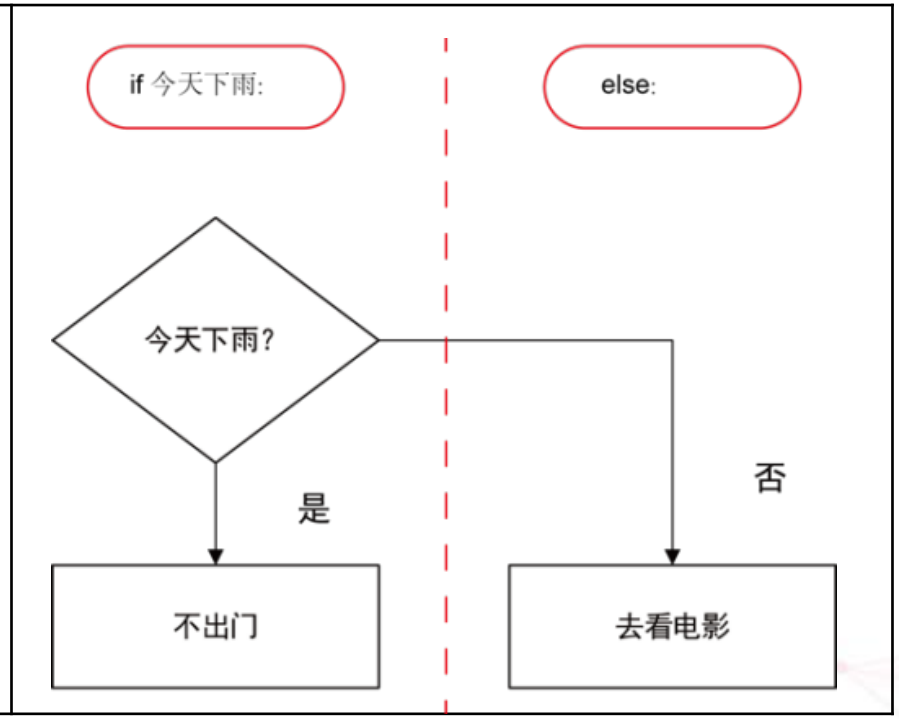


if/else語句 - 雙分支結構

在雙分支的結構裡，面臨的兩種選擇，**二者必選其一。**

- 如果條件判斷的結果為True，就執行第一個分支裡面的語句；
- 否則，執行第二個分支裡面的語句。

```
if 明天下雨：
    我不出門
else:
    我去看電影
```



Q：當選擇條件達到兩個以上時，語句的結構該如何呢?

if/else if.../else語句 - 多分支結構

當選擇條件達到兩個以上，還需要使用如下的結構：

```
if 條件A:
   指令A
elif 條件B:
   指令B
elif 條件C:
   指令C
   ...
else:
   指令Z
```

使用超聲波感測器，我們可以通過get_ultrasound()函數來獲取到小車與障礙物的距離，並把它用distance來表示。

- 如果distance為100cm以上時，列印顯示“far away”；
- 距離在50cm-100cm時，列印顯示“closer”；
- 距離不到50cm時，列印顯示“nearby”。

```
distance = get_ultrasound()
if distance > 100:
      print("far away")
elif 50 <= distance <= 100:
      print("closer")
else:
      print("nearby")
```

for語句

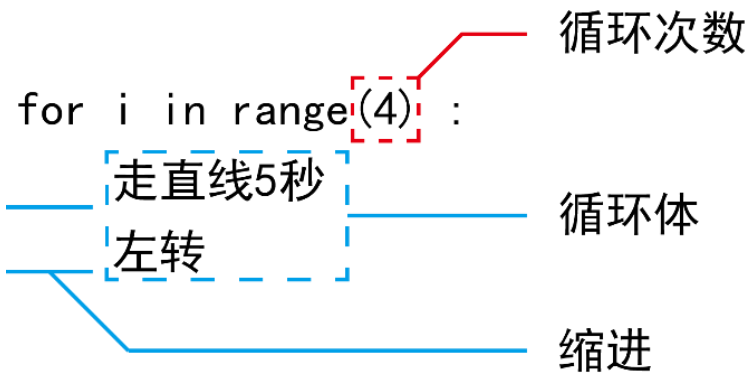
for單詞的直接翻譯是“對於”，於是，我們可以理解為，對某一個循環體反復執行多次。

採用for語句，走正方形的任務描述將會改進為：

對 循環體執行 4次：
    走直線5秒
    左轉

```
for i in range(4):
    go(60, 60, 5)
    turn_left()
```

for語句主要由**迴圈次數**和**循環體**2部分組成。



while語句

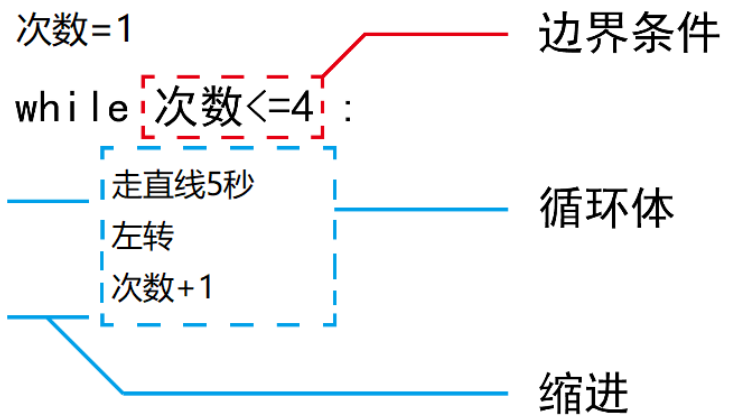
循環體重複的次數未知時，可以採用while迴圈結構。在while結構中，是否進入迴圈取決於**邊界條件是否滿足**，當邊界條件被滿足則進入迴圈；否則就退出迴圈。

如果使用while語句，迴圈走正方形的結構可描述為：

當 計數沒有超過4次的時候：
    走直線5秒
    左轉

在這樣的結構中，我們需要一個**計數器**來記錄迴圈了多少次。這其實包含了三個資訊：從哪裡開始數，每次數幾，數到什麼時候結束。

一個while迴圈結構主要包含2個組成部分：邊界條件和循環體。




```
counter = 0
while counter < 4:
  go(60, 60, 5)
  turn_left()
      counter += 1
```