前言

發現之旅並非是尋找新的景色，
而是擁有新的視野。
——Marcel Proust（法國作家：1871-1922）

受到深度學習的影響，現在各個領域都掀起了技術革新的浪潮，包括自駕車、自動診斷系統、精準機械翻譯、先進的機器人控制等。這些宛如虛構小說般描述的內容，最近卻變得真實，而且社會上也出現幾個引起關注的例子。令人驚訝的是，這些如同虛構小說中的技術，有些可能因為深度學習而有機會成真（或者逐漸變得可能）。我們可以說正處於一個被深度學習改變的時代。

在蓬勃發展的深度學習領域，創造了許多深度學習框架，包括 PyTorch、Chainer、TensorFlow、Caffe……等。這個領域有著各式各樣的框架，而且每天仍持續開發著，世界上的研究人員及技術人員得以利用這些框架有效率地解決問題。深度學習框架可說是支撐、推動先進技術不可或缺的重要關鍵。

拿起這本書的你，或許就曾實際使用過深度學習框架。最近深度學習的資料不僅豐富，執行環境也很完整，使得編寫與深度學習有關的程式碼變得易如反掌。我們僅用數十行（或數行）的程式碼，就能展現強大的技術，也都是拜框架所賜。

那麼，這種有許多人使用、能在各種場合發揮作用的「真正」框架，究竟是以何種結構在運作？使用了哪些技術，背後的概念為何？在這些疑問的驅使下，我們將展開這段新的旅程！

動手實作才能體會

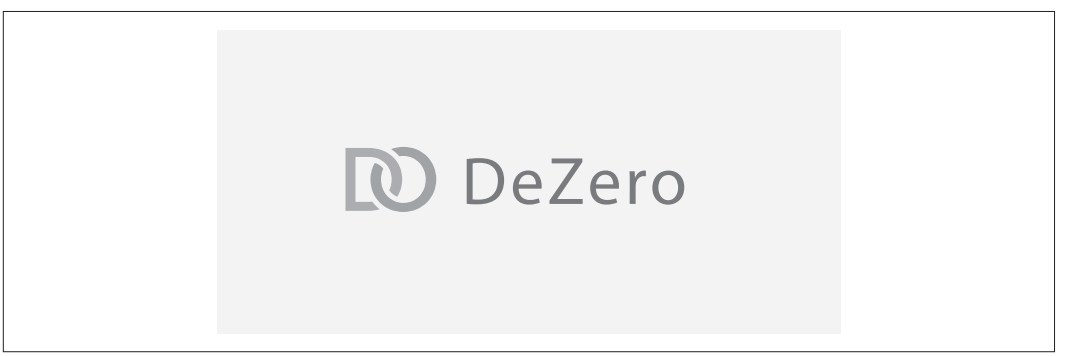
在深度學習框架中，充滿著許多令人驚訝的技術以及有趣的結構。本書就是為了解開這些疑問，正確瞭解這些技術而撰寫的。希望你可以從中體會這種技術性的「樂趣」。基於這個目的，本書將秉持著「從零開始製作」的方針，從無到有，一邊操作，一邊思考，透過實作加深理解。藉由這種經驗，發現深度學習框架的本質。

在製作框架的過程中，有許多必須學習的知識。「原來可以使用這種技術啊！」、「這個點子可以這樣執行啊！」使用現成的工具是無法體會這些樂趣的。有時得實際動手做了之後，才會有新的發現。

舉個例子來說，可能有部分讀者認為深度學習框架不過是純粹收集「層」或「函數」的函式庫。事實上，深度學習框架沒有那麼簡單。它是一種程式語言，更精準來說，是一個擁有微分運算功能的程式語言（最近也稱作「可微分的程式語言」）。這一點在閱讀本書時，透過「從零開始製作的過程」就可以理解。

本書的原創框架

在黎明期，深度學習框架的差異甚遠，但是現在的深度學習框架已經進入成熟階段。事實上，PyTorch、Chainer、TensorFlow 等知名框架皆朝著相同方向發展（這些框架各具特色，介面亦不相同，但是設計概念卻是共通的）。基於這些共通項目及教育層面，本書設計了一個小框架，並以本書的標題為名，將這個框架稱作「DeZero」，同時製作出以下 LOGO。



DeZero 是本書的原創框架。以 Chainer 為基礎，加入 PyTorch 的設計，特色如下。

1. 精簡

DeZero 是以重視易讀性而設計的框架，盡可能只使用必要的外部函式庫，並盡量簡化程式碼，因此你不需要花太多時間來瞭解整個 DeZero 的程式碼。

2. 純 *Python*

深度學習框架通常會使用多種語言（例如 Python 與 C++ 等），但是我們只用 Python 執行 DeZero。因此具備 Python 相關知識的人，可以毫無壓力地閱讀 DeZero 的說明。由於是純 Python，所以能在智慧型手機上執行 DeZero，或使用 Google Colaboratory 等服務，輕鬆在雲端上運作。

3. 現代化的功能性

PyTorch、Chainer、TensorFlow 等現代化框架擁有許多共通的功能。最重要的共通項目之一，就是 Define-by-Run。Define-by-Run 是指依照執行運算的時機，建立深度學習運算「連結」的結構（這個部分會在內文詳細說明）。本書製作的 DeZero 是 Define-by-Run 式的框架，與現代化框架有著許多共通點。

前作《Deep Learning：用 Python 進行深度學習的基礎理論實作》、《Deep Learning 2| 用 Python 進行自然語言處理的基礎理論實作》是從零開始進行深度學習，藉此瞭解相關結構。當時以單純性為優先，而「手動」設定了運算的「連結」。真正的框架是將這個部分自動化，Define-by-Run 就是其中的一種手法，本書將利用從零開始製作 DeZero 的方式來學習這個機制。請別擔心，閱讀這本書不需要具備前作《Deep Learning：用 Python 進行深度學習的基礎理論實作》系列的知識。

漸進式製作過程

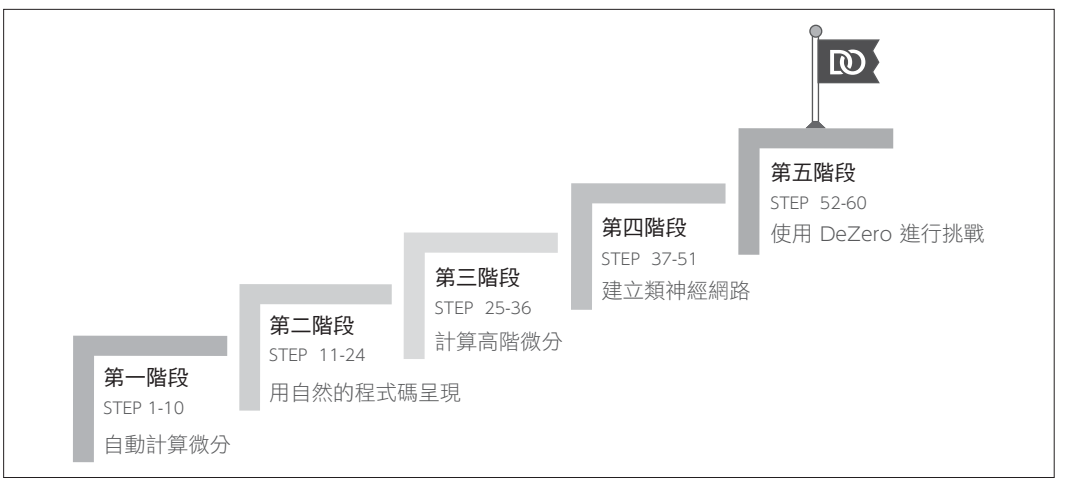
雖然 DeZero 是一個小框架，內容卻十分複雜，因此本書把「製作 DeZero」的過程細分成幾十個步驟，以化解這種複雜性。具體來說，這次的課程利用了 60 個步驟，一步一步製作出 DeZero。

本書的第一個步驟是建立 DeZero 的「變數」。事實上，這個步驟的程式碼只有短短三行。而下一個步驟是加上「函數」用的程式碼。由於每個步驟都是當下就完成，因此能實際執行。透過這種方式，漸進式（階段性）地組合 DeZero，一邊操作，一邊加深理解。

另外，從本書學到的經驗也是練習軟體開發的好機會。這不但是從零開始製作複雜系統的體驗，也是學習軟體開發的最佳題材。基於這一點，本書還額外使用了一些篇幅來介紹軟體開發的作法。

本書的藍圖

前面提到這本書是由 60 個步驟組成，這 60 個步驟大致可以分成五個階段，如下圖所示。以下先簡單介紹每個階段的執行內容。



- **第一階段**是建立 DeZero 的基礎。這個階段只處理單純的問題，用最少的時間建立自動計算微分的機制（在閱讀內文的過程中，就會逐漸明白何謂「自動計算微分」）。
- **第二階段**是進階成用更自然的程式碼使用 DeZero。第二階段結束時，會變成用一般的 Python 語法，如 if 或 for 撰寫程式。
- **第三階段**將擴大 DeZero，計算二階微分。我們必須讓 DeZero 可以進行「反向傳播的反向傳播」，才能做到這一點。瞭解這個機制可以知曉 DeZero，甚至是現代化框架的新可能性。
- **第四階段**將調整 DeZero，以符合類神經網路。如此一來，使用 DeZero 就能輕鬆地建立類神經網路。

第一階段
自動計算微分

在現代的科學技術中，每個領域都有運用到微分。尤其是包含深度學習在內的機器學習領域，微分扮演著極為重要的角色。深度學習框架可以說是計算微分的工具，因此本書的主題自然就是「微分」。 換言之，這本書就是在說明如何利用電腦計算微分。

接下來要開始的第一階段共由 10 個步驟組成，這個階段將建立一個自動計算微分的框架。所謂的「自動計算微分」是指由電腦（不是人）來計算微分。具體而言，這是寫出某項運算（函數），電腦就會自動算出微分的系統。

在這個階段，為了自動計算微分，我們將建立代表「變數」與「函數」的兩個類別（`Variable` 類別與 `Function` 類別）。有了這兩個類別，就能奠定自動計算微分的基礎。在第一階段結束時，已經可以自動算出單純（函數）的微分。接下來，讓我們踏出 DeZero 的第一步吧！

STEP 1
把變數當成箱子

本書的第一個步驟是建立 DeZero 的構成元素「變數」，變數是 DeZero 最重要的部分。這個步驟將說明變數的功用，並執行該功能。

1.1 何謂變數？

究竟變數是什麼？打開程式設計的入門書，通常是以**圖 1-1** 的方式來說明變數。

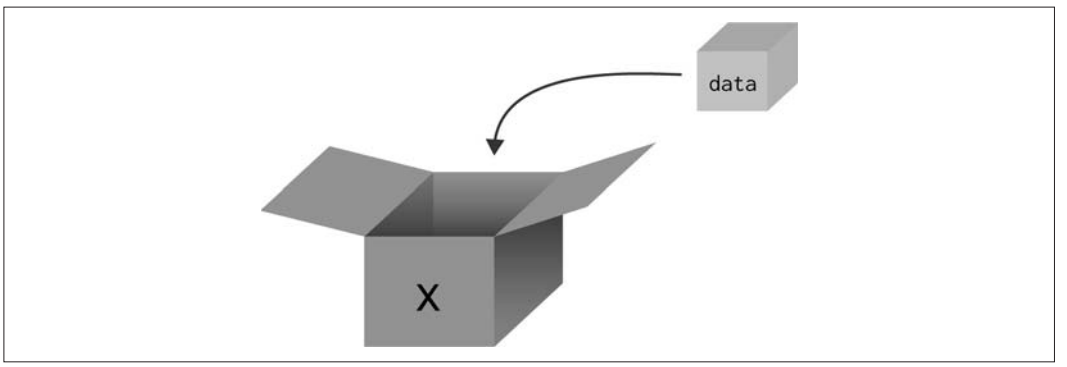


圖 1-1 變數的說明範例

我們可以看到**圖 1-1** 在箱子裡放入了資料，而這個箱子就是變數。把變數**比喻**成「箱子」，（在某種程度上）完美說明了變數的性質。變數主要有以下幾個特點。

- 箱子與資料是不同的東西
- 在箱子裡放入資料（= 代入）
- 檢視箱子就能知道是什麼資料（= 參照）

機器學習系統使用了「多維陣列」當作基本的資料結構，因此 DeZero 的 Variable 類別只能處理 NumPy 的多維陣列。另外，NumPy 多維陣列的類別是 numpy.ndarray（np.ndarray）。如上述程式碼所示，利用 np.array 函數可以產生該實例。本書在後面提到 numpy.ndarray 實例時，簡化為 ndarray 實例。

接著在上述程式碼的 x 帶入新資料，寫法如下所示。

steps/step01.py

```
x.data = np.array(2.0)
print(x.data)
```

執行結果

```
2.0
```

如上所示，寫成 x.data = ... 就會帶入新資料，把 Variable 類別當作「箱子」使用。

以上是這個步驟執行的所有內容。現在 Variable 類別只有短短三行程式碼，請以此為出發點，建立現代化框架 DeZero。

1.3 【補充說明】NumPy 多維陣列

最後簡單補充說明 NumPy 的多維陣列。多維陣列是指規律地排列數值等元素的資料結構。元素排列有「方向性」，這裡的方向稱作「維度」或「軸」。圖 1-2 就是一個多維陣列的例子。

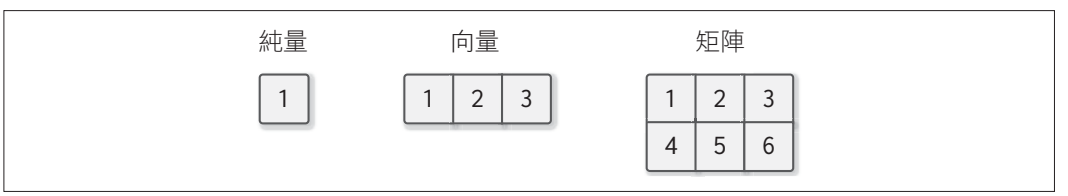


圖 1-2 多維陣列範例

圖 1-2 自左起為 0 維陣列、一維陣列、二維陣列，分別稱作「純量」、「向量」、「矩陣」。純量代表一個數值，而向量是沿著一個軸排列數值，矩陣是沿著兩個軸排列數值。

get_spiral 函數會取得旗標 train 當作引數。假如 train=True，就回傳學習（訓練）用的資料；如果 train=False，則回傳測試用的資料。實際的回傳值是 x 與 t，x 是輸入資料，t 是訓練資料（標籤）。x 是形狀為 (300, 2) 的 ndarray 實例，t 是形狀為 (300,) 的 ndarray 實例。這裡要處理三個類別的分類問題，從 0、1、2 取得一個數值當作 t 的元素。附帶一提，把 x 與 t 繪製成圖表，結果如圖 48-1 所示。

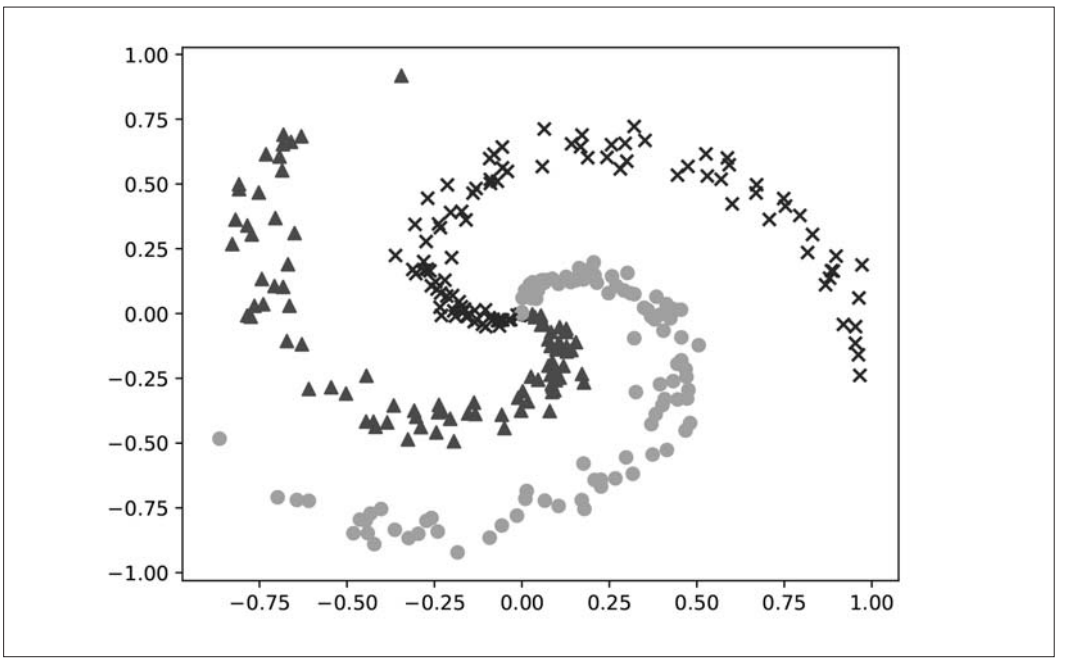


圖 48-1　分布成螺旋狀的資料集

圖 48-1 把每個類別變成符號 ○△× 再繪製成圖表。如你所見，這是呈現螺旋狀分布的資料集。接下來要使用類神經網路，確認資料集是否正確分類。

專欄：深度學習框架

早期的深度學習框架有很大的差異，可是現在的框架已經進入成熟階段。事實上，PyTorch、Chainer、TensorFlow 等知名框架的目標幾乎都是一致的。這些框架有各自的特色，介面也不同，但是成為核心的設計思想卻能看到許多共通點。具體來說，有以下幾點。

- 可以利用 Define-by-Run 建立計算圖
- 具有函數及階層等集合
- 具有更新參數的類別（優化器）集合
- 模型可以當成子類別來執行
- 具有管理資料集的類別
- 除了 CPU 之外，可以用 GPU 或獨家的 ASIC 執行
- 具有能當作靜態計算圖來執行的模式，可以提高效能（假設要當作產品使用）

以上是現代深度學習框架的共通特色。以下將針對前面三點提出具體範例詳細說明。

嚴格來說，深度學習的「框架」並非「工具」或「函式庫」。函式庫與框架的差異在於是由「誰」來控制程式。函式庫是便利函數及資料結構的集合，使用者從中取出必要的部分來運用。此時是由使用者決定程式控制，也就是在哪個步驟執行程式碼。然而框架是提供整體基礎，深度學習的框架會提供自動微分的基礎，然後使用者在上面建構必要的計算，此時是由框架進行整體控制。函式庫與框架的差別就是由「誰」來控制程式。

Define-by-Run 式的自動微分

在深度學習的框架中，最重要的功能是「自動微分」。利用自動微分，我們可以省去麻煩，立刻計算出微分。此外，現代框架是用 Define-by-Run 製作計算圖，可以立刻執行程式碼，並在背後建立計算圖。因此能使用 Python 的語法來製作計算圖。以 PyTorch 為例，可以寫出以下程式碼。

54.1 何謂 Dropout？

Dropout 是一邊隨機刪除（變成無效）神經元，一邊學習的手法。學習時，會隨機選取並刪除隱藏層的神經元，被刪除的神經元就無法傳遞訊號，如圖 54-1 所示。

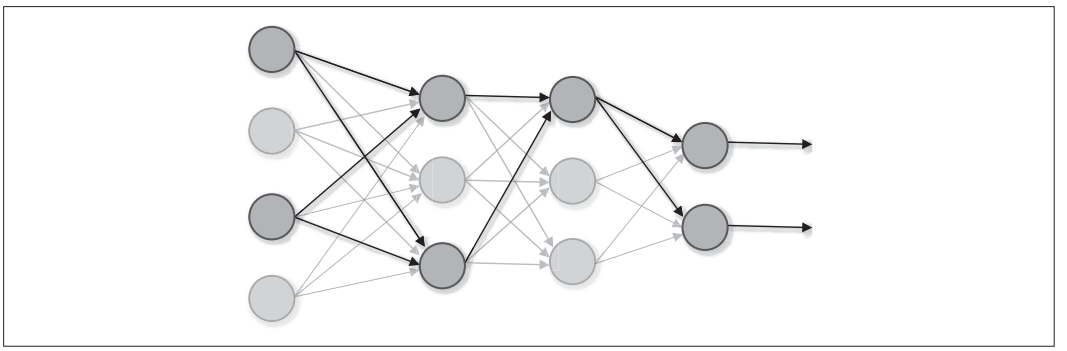


圖 54-1 Dropout 的學習行為

使用 Dropout 的學習在傳遞資料時，會隨機選取要刪除的神經元。假設有一個由 10 個神經元形成的階層，該層的下一層要使用 Dropout 層隨機刪除 60% 的神經元。以程式碼顯示此種行為，結果如下所示。

```
import numpy as np

dropout_ratio = 0.6
x = np.ones(10)

mask = np.random.rand(10) > dropout_ratio
y = x * mask
```

這裡的 mask 是元素為 True 或 False 的陣列。製作 mask 的方法是先利用 np.random.rand(10)，隨機產生 10 個介於 0.0 ～ 1.0 的值。將這些值與 dropout_ratio（＝ 0.6）比較，只有比 dropout_ratio 大的元素為 True，其餘為 False。這個範例是產生 False 的平均比例為 60% 的 mask。

建立 mask 之後，再執行 y = x * mask，這樣與 mask 的 Fasle 對應的 x，其元素為 0（亦即刪除）。結果每次平均只將 4 個神經元的輸出傳遞給下一層。Dropout 層在學習時，每次傳遞資料就會進行上述處理。

如圖 55-1 所示，CNN 增加了新的 Conv 層與 Pool 層，而 CNN 各層的連結順序是「Conv → ReLU →（Pool）」（有時也會省略 Pool 層）。我們可以當成前面的「Linear → ReLU」連結變成「Conv → ReLU →（Pool）」。另外，圖 55-1 靠近輸出的層使用了前面的「Linear → ReLU」組合，以上是一般 CNN 常見的結構。

55.2 卷積運算

CNN 使用了卷積層，卷積層會執行「卷積運算」。用影像處理來比喻，相當於「篩選器運算」。這裡以圖 55-2 為例來說明卷積運算。

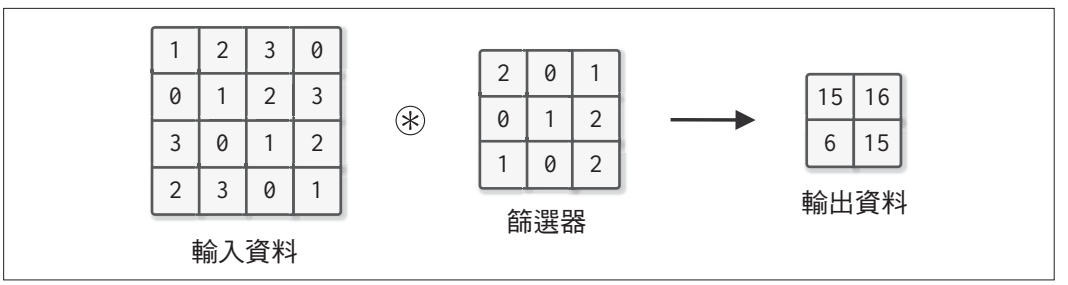


圖 55-2 卷積運算的範例（以「⊛」代表卷積運算）

如圖 55-2 所示，卷積運算的作用是篩選輸入資料。這個範例的輸入資料包括垂直、水平方向的維度，篩選器也同樣有著垂直、水平方向的維度。若以 (height, width) 的順序表記形狀，這個範例的輸入形狀為 (4, 4)，篩選器的形狀是 (3, 3)，輸出的形狀是 (2, 2)，此時會按照圖 55-3 的順序進行卷積運算。

57.1 用 im2col 展開輸入資料

im2col 是「展開」輸入資料的函數，能讓卷積運算的核心輕易展開輸入資料。圖 57-1 從三階張量的輸入資料取出套用核心的區域（正確而言是從包含批次數量在內的四階張量取出核心區域）。

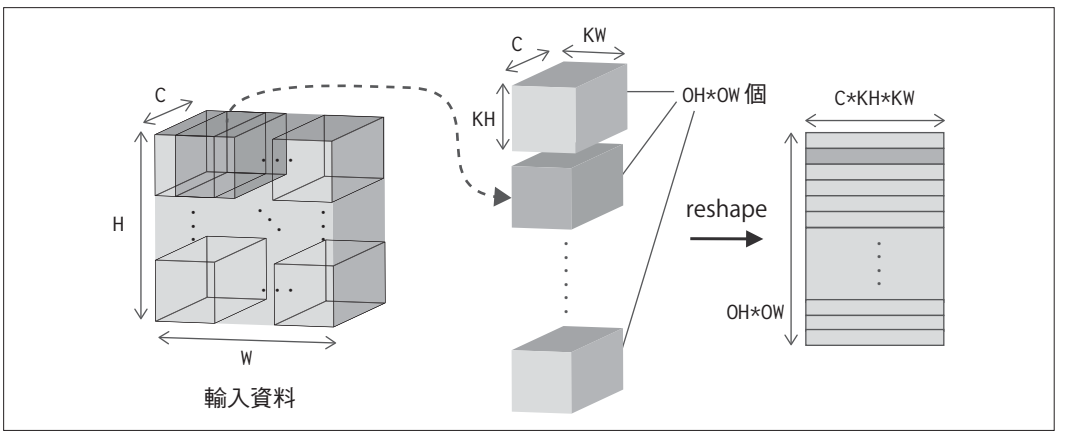


圖 57-1 展開套用核心的區域

如圖 57-1 所示，取出套用核心的區域。然後將取出的區域調整成一行，最後轉換成「矩陣（二階張量）」，這就是 im2col 函數進行的處理。

Chainer 的 im2col 會進行圖 57-1 第一階段的處理（不包含 reshape 部分的處理）。因為只要取出核心區域，之後就會按照張量乘積* 進行運算。本書為了使用矩陣乘積，必須進行 reshape 處理。只有 DeZero 使用 im2col 函數的引數具有 to_matrix 旗標，如果為 True，會同時進行圖 57-1 的 reshape 處理。

利用 im2col 展開輸入資料後，再把卷積層的核心（篩選器）展開成一行，接著如圖 57-2 所示，計算這兩個矩陣乘積。

* 簡單來說，張量乘積就是擴大矩陣乘積。在任意張量之間，設定張量軸，進行乘積累加運算。NumPy 是使用 np.tensordot 或 np.einsum 計算張量乘積。