1.1.2 各個矩陣的逐元素運算

我們將數字的集合歸納成「向量」或「矩陣」，接著要使用這些部分，進行簡單的運算。首先，要介紹「逐元素運算」。附帶一提，「逐元素」的英文是 element-wise。

```
>>> W = np.array([[1, 2, 3], [4, 5, 6]])
>>> X = np.array([[0, 1, 2], [3, 4, 5]])
>>> W + X
array([[ 1,  3,  5],
       [ 7,  9, 11]])
>>> W * X
array([[ 0,  2,  6],
       [12, 20, 30]])
```

這裡要針對 NumPy 的多維陣列進行 + 或 * 等四則運算。此時，利用多維陣列中的逐元素（各個元素獨立）進行運算。這就是 NumPy 陣列的「逐元素運算」。

1.1.3 廣播

在 NumPy 的多維陣列中，可以運算形狀不同的陣列。例如，進行以下運算：

```
>>> A = np.array([[1, 2], [3, 4]])
>>> A * 10
array([[10, 20],
       [30, 40]])
```

在此運算中，針對 2×2 矩陣 A，乘上純量 10。

此時，如**圖 1-3** 所示，純量 10 擴張成 2×2 矩陣，之後進行逐元素運算，如此聰明的功能稱作**廣播**（broadcast）。

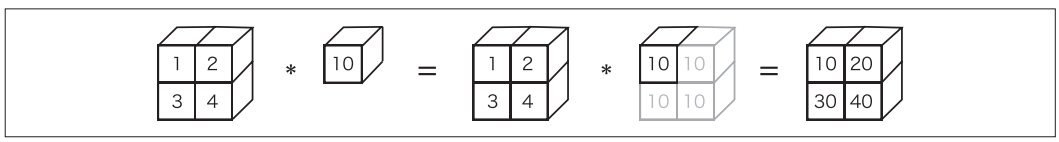

圖 1-3　廣播的範例：把純量「10」當作 2×2 矩陣處理

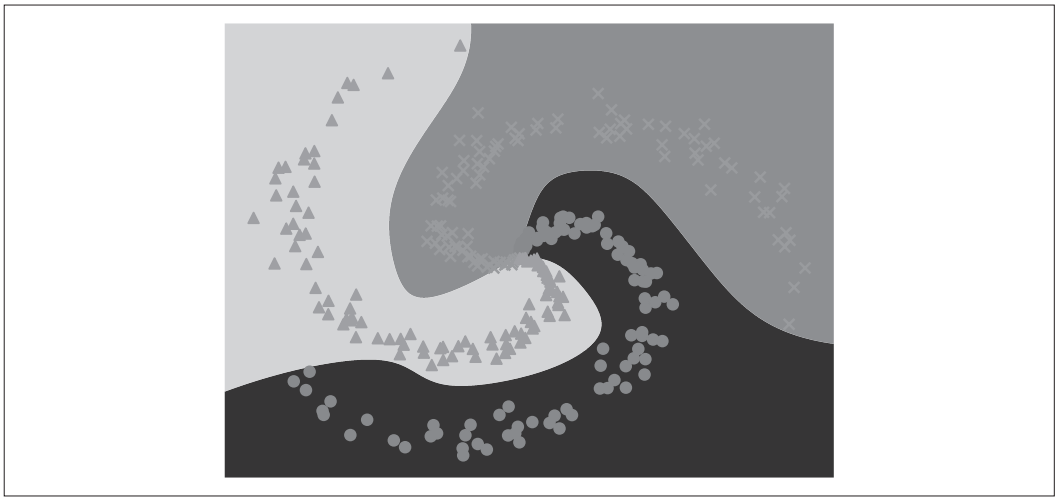

圖 1-33　學習後類神經網路的決策邊界（按照顏色繪製類神經網路判斷的各個類別區域）

如圖 1-33 所示，我們可以瞭解，學習後的類神經網路正確掌握了「螺旋」類型，亦即可以學習非線性的分離領域。像這樣，類神經網路利用隱藏層，能呈現出複雜的表現。此外，重疊多層，表現力會變得更豐富，這就是深度學習的特色。

1.4.4　Trainer 類別

前面說明過，本書有很多機會執行類神經網路的學習，因此必須像前面一樣，寫出執行學習用的程式碼，可是每次都要撰寫相同的程式碼，實在很枯燥。於是本書提供了學習用的 Trainer 類別，內容與剛才的程式碼幾乎一模一樣，部分加入了新的功能，後續將視情況說明詳細的使用方法。

Trainer 類別位於 common/trainer.py 中。這個類別的初始化是接收類神經網路（模型）與優化器。具體而言，如下所示。

```
model = TwoLayerNet(...)
optimizer = SGD(lr=1.0)
trainer = Trainer(model, optimizer)
```

這裡將指出計數手法的問題，並大致說明取而代之的推論手法有何優點。以下將介紹用類神經網路處理「字詞」的範例，當作執行 word2vec 的事前準備。

3.1.1 計數手法的問題

前面我們介紹過的計數手法，是利用周圍的詞頻來表現字詞。具體而言，就是建立字詞的共生矩陣，針對該矩陣，套用 SVD，獲得稠密矩陣（字詞的分散式表示）。可是，計數手法有個問題，在處理大型語料庫時，就會發生。

現實生活中，用語料庫處理的語彙量非常龐大，據說英文的語彙量就超過 100 萬個。假設語彙量超過 100 萬，在計數手法中，會建立 100 萬 ×100 萬的巨大矩陣。可是，對如此龐大的矩陣執行 SVD 是行不通的。

SVD 是針對 $n \times n$ 的矩陣，花費 $O(n^3)$ 的計算成本。$O(n^3)$ 是指，以 n 的三次方為比例，增加計算時間。這種計算成本，就算你擁有超級電腦，也招架不住。實際上，利用類似的方法或稀疏矩陣的性質，可以提高處理速度。不過，即便如此，仍需要耗費大量的運算資源及時間。

計數手法是利用整個語料庫的統計資料（共生矩陣及 PPMI 等），一次處理（SVD 等）獲得字詞的分散式表示。然而，在推論手法中，若使用類神經網路，一般會用小批次進行學習。用小批次進行學習是指，類神經網路一次檢視少量（小批次）的學習樣本，並且反覆更新權重。若以圖表顯示這種學習的架構差異，結果如**圖 3-1** 所示。

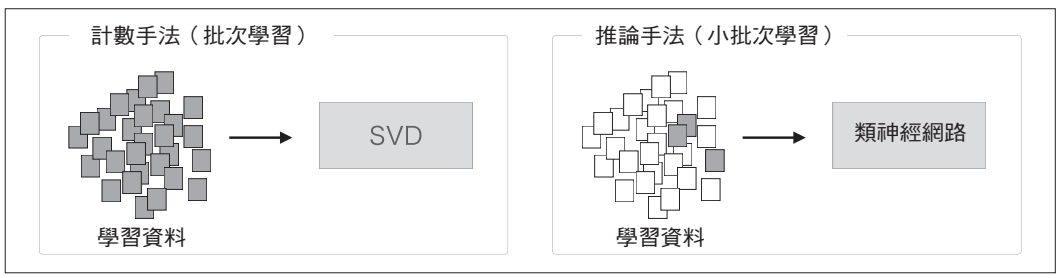

圖 3-1　比較計數手法與推論手法

如圖 3-1 所示，計數手法是一次統一處理學習資料；相對來說，推論手法是使用部分學習資料，逐次學習。這代表著在語彙量大的語料庫中，即使遇到 SVD 等運算量龐大、處理困難的情況，類神經網路也能把資料分成小部分來學習。此外，類神經網路的學習可以利用多個裝置 / 多個 GPU，進行平行計算，整體學習也能高速化。就這點來看，推論手法較有優勢。

此外，與計數手法相比，推論手法也有吸引人之處。針對這一點，我想先詳細說明推論手法（尤其是 word2vec），之後在「3.5.3 計數手法 v.s. 推論手法」再深入探討。

3.1.2 推論手法概要

推論手法主要的工作是進行「推論」。如圖 3-2 所示，當給予周圍的字詞（上下文）時，推測「？」會出現哪個字詞。

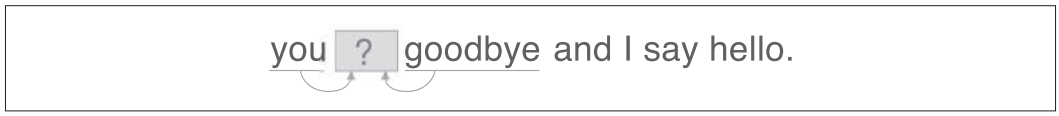

圖 3-2　把相鄰的字詞當作上下文，推測「？」會出現哪個字詞

解開如圖 3-2 的推論問題並學習，就是「推論手法」要處理的問題，亦即反覆解開推論問題，學習字詞出現的型態。若以「模型觀點」來看，推論問題如圖 3-3 所示。

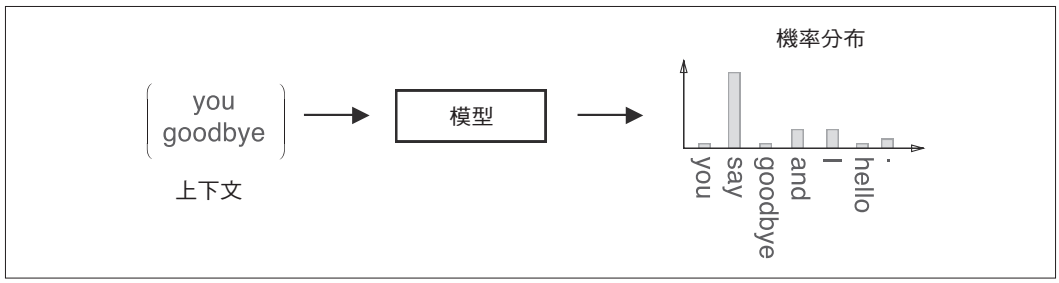

圖 3-3　推論手法：輸入上下文，模型就會輸出各字詞的出現機率

如圖 3-3 所示，在推論手法中，會出現某個模型。我們在該模型中，會使用類神經網路。模型是把取得的上下文當作輸入，再輸出各個字詞（可能出現）的出現機率。在這種架構下，使用語料庫進行模型學習，完成正確推測，最後可以獲得字詞的分散式表示，當作學習結果，這就是推論手法的整體概念。

3.2.2 CBOW 模型的學習

前面說明的 CBOW 模型是在輸出層輸出各個字詞的分數。針對該分數套用 Sotmax 函數，可以獲得「機率」（圖 3-12）。該機率代表，給予上下文（前後的字詞）時，在中央會出現哪個字詞。

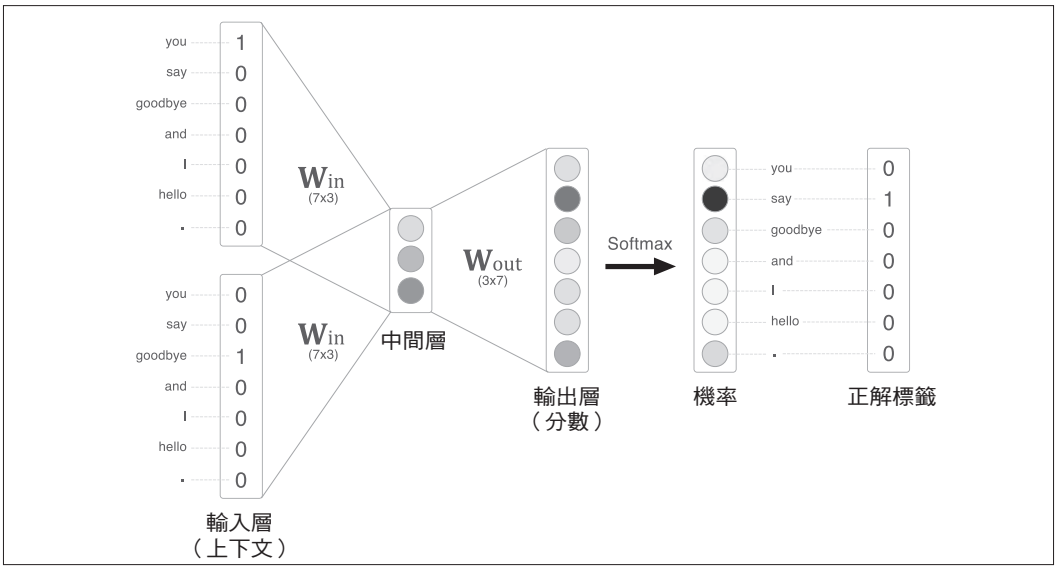

圖 3-12　CBOW 模型的具體範例（以黑白深淺顯示節點的值）

在圖 3-12 顯示的範例中，上下文是「you」及「goodbye」，正解標籤（類神經網路應該預測的字詞）是「say」。此時，若有含「優質權重」的類神經網路，可以期待對應正解的神經元「機率」會變高。

利用 CBOW 模型的學習，調整權重，完成正確預測。就結果而言，是在權重 $\mathbf{W}_{in}$（正確來說是 $\mathbf{W}_{in}$ 與 $\mathbf{W}_{out}$ 兩者）學習掌握了字詞出現型態的向量。利用前面的實驗，使用 CBOW 模型（及 skip-gram 模型）獲得字詞的分散式顯示（尤其是使用 Wikipedia 等大型語料庫得到字詞的分散式表示），在字意及文法上，有很多案例與我們的直覺一致。

此時，CBOW 模型進行了何種處理？實際用圖表顯示，如**圖 4-7** 所示。

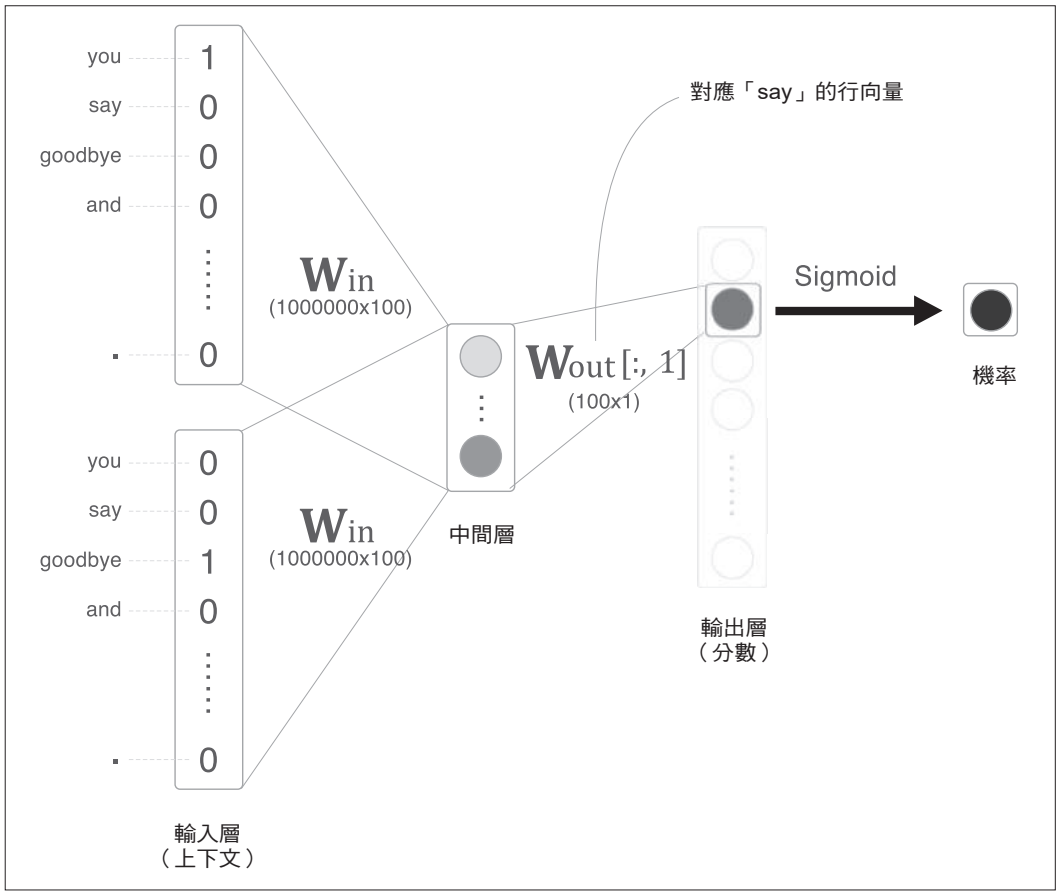

圖 4-7　這個類神經網路只計算成為目標對象的字詞分數

如**圖 4-7** 所示，輸出層只有一個神經元。因此，中間層與輸出端的權重矩陣乘積只要擷取對應「say」的行（詞向量），計算擷取出來的向量和中間層神經元的內積即可。詳細的計算過程如**圖 4-8** 所示。

4.4.1 使用了 word2vec 的應用程式範例

使用 word2vec 獲得的字詞分散式表示，可以發揮在找出相似詞的用途。可是，字詞分散式表示的優點不只這一點，在自然語言處理中，字詞的分散式表示如此重要的理由，是因為**遷移學習**（transfer learning）。這是指能將某個領域學到的知識套用在其他領域。

解決自然語言的任務時，幾乎不會從頭開始學習用 word2vec 得到的字詞分散式表示。而是先用大型語料庫（Wikipedia、Google News 的文本資料等）進行學習，在各個任務中，使用已經學習完畢的分散式表示。例如，文本分類、文件分群、詞性標記、情感分析等自然語言的任務，在一開始將字詞轉換成向量的步驟，可以利用學習完畢的字詞分散式表示。在形形色色的自然語言處理中，字詞的分散式表示幾乎都能獲得良好的結果！

字詞分散式表示的優點是，可以把字詞轉換成長度固定的向量。此外，文章（排列字詞）也可以使用字詞的分散式表示，轉換成固定長度的向量。目前正在積極研究如何把文章轉換成固定長度的向量。我們想到最單純的方法，就是把文章的各個字詞轉換成分散式表示，計算出總和。這是稱作 bag-of-words，不考慮文本順序的模型（想法）。另外，使用第五章說明的遞歸神經網路（RNN），能以更簡潔的方法，利用 word2vec 的字詞分散式表示，把文章轉換成固定長度的向量。

可以把字詞或文章轉換成固定長度的向量，這件事非常重要。因為只要能將自然語言轉換成向量，就可以套用一般的機器學習手法（類神經網路、SVM 等）。以圖表顯示，如**圖 4-21** 所示。

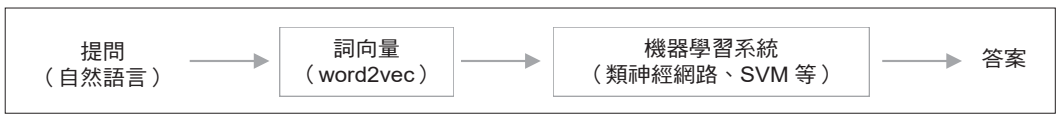

圖 4-21　使用了字詞分散式表示的系統處理流程

如**圖 4-21** 所示，如果能將以自然語言寫出來的問題，轉換成固定長度的向量，就可以把該向量輸入其他機器學習系統。將自然語言轉換成向量後，即可利用一般的機器學習系統框架，輸出（還有學習）目標答案。

CBOW 是 continuous bag-of-words 的縮寫。bag-of-words 的意思是「位於袋中的字詞」，代表忽略袋中字詞的「排列」順序。

讓我們舉個具體的例子，說明關於忽略上下文的字詞排列會產生的問題。假設要處理兩個字詞當作上下文，在 CBOW 模型中，兩個詞向量的「和」會到達中間層。以圖表顯示，如**圖 5-5** 左所示。

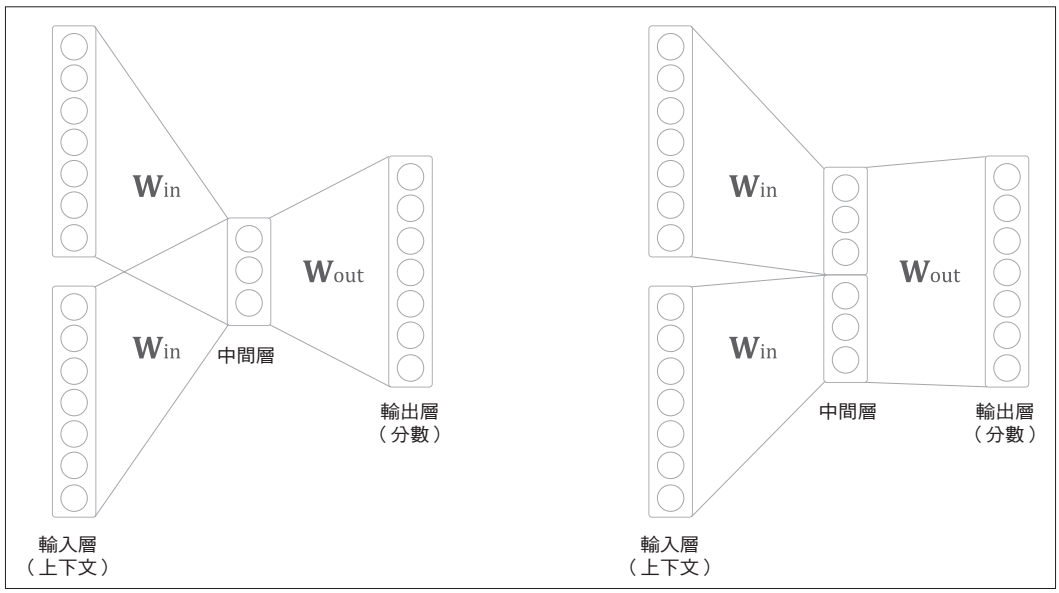

圖 5-5　左圖是一般的 CBOW 模型。右圖是在中間層，顯示「連結」各個上下文詞向量的模型（這張圖把輸入層當作 one-hot 向量顯示）

如**圖 5-5** 左所示，CBOW 模型的中間層為了計算詞向量的和，而忽略了上下文的字詞排列方式。例如，把 (you, say) 和 (say, you) 視為相同的上下文。

原本我們希望的模型是可以考量到上下文的字詞排列吧！為了達到這個目標，想到在中間層「連結（concatenate）」上下文的詞向量，如**圖 5-5** 右所示。實際上，在 Neural Probabilistic Language Model [28] 提出的模型，就採取了這種手法（詳細的模型內容請參考論文 [28]）。可是，採用了連結方法之後，參數權重會與上下文的大小等比例增加。當然，我們並不樂見這種參數增加的情況。

```
class RNN:
    def __init__(self, Wx, Wh, b):
        self.params = [Wx, Wh, b]
        self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]
        self.cache = None

    def forward(self, x, h_prev):
        Wx, Wh, b = self.params
        t = np.dot(h_prev, Wh) + np.dot(x, Wx) + b
        h_next = np.tanh(t)

        self.cache = (x, h_prev, h_next)
        return h_next
```

進行 RNN 的初始化時，會取得兩個當作引數的權重及一個偏權值。這裡把用引數傳遞過來的參數，當作清單設定在成員變數 params 中。接著，以對應各個參數的形式，將梯度初始化，並且儲存在 grads 中。最後，把計算反向傳播時，使用的中間資料當作 cache，用 None 初始化。

在正向傳播的 forward(x, h_prev) 方法中，取得兩個引數（來自下方的輸入 x 及來自左側的輸入 h_prev）。之後，只要按照算式（5.10）執行即可。附帶一提，來自上一個 RNN 層的輸入是 h_prev，此時刻的 RNN 層輸出（= 傳遞給下個時刻的輸入）是 h_next。

接著執行 RNN 的反向傳播。在此之前，我想利用圖 5-19 的計算圖，再次確認 RNN 的正向傳播。

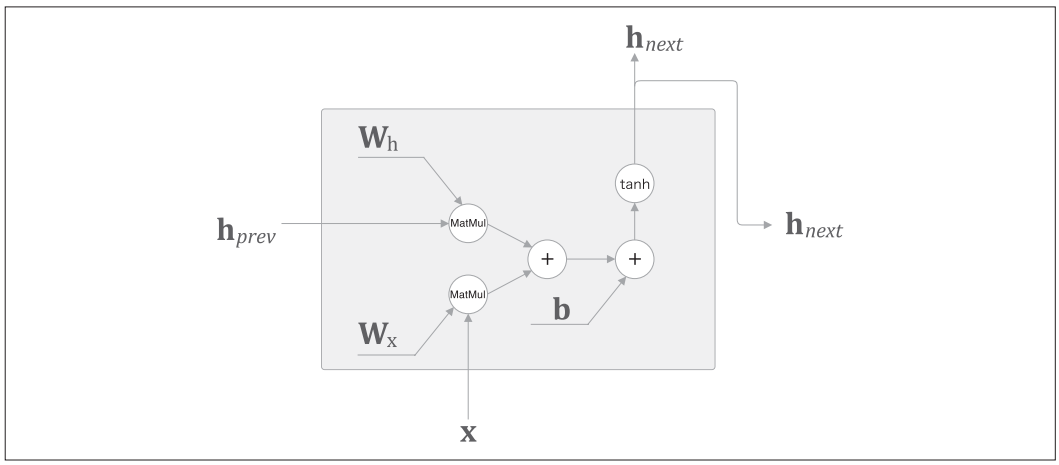

圖 5-19　RNN 層的計算圖（MatMul 層代表矩陣乘積）

如**圖 7-8** 所示，Decoder 的結構與上一節的類神經網路完全一模一樣。但是，有一點與上一節的模型不同。兩者的差別在於 LSTM 層取得向量 **h** 這一點。附帶一提，在上一節的語言模型中，LSTM 層沒有取得任何東西（嚴格來說，是 LSTM 的隱藏狀態取得「0 向量」）。這個唯一的小差異，把一般的語言模型進化成可以翻譯的 Decoder ！

圖 7-8 使用了 <eos> 分隔字元（特殊符號）。這個「分隔字元」會當成通知 Decoder 開始產生文章的訊號。另外，Decoder 會進行字詞取樣，直到輸出 <eos> 為止，所以這也是結束的訊號。換句話說，<eos> 可以當作通知 Decoder「開始 / 結束」的分隔字元。在其他的文獻中，也有使用 <go>、<start> 或「_（underscore）」的例子。

接下來，要顯示結合 Decoder 與 Encoder 的分層結構，如**圖 7-9** 所示。

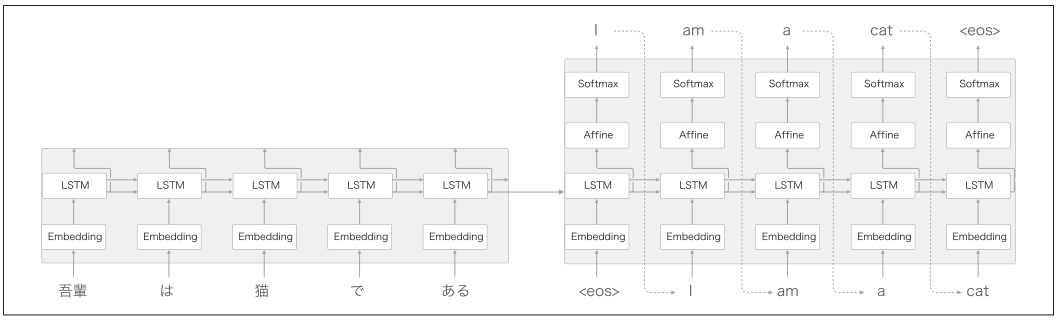

圖 7-9　整個 seq2seq 的分層結構

如**圖 7-9** 所示，seq2seq 是由兩個 LSTM（Encoder 的 LSTM 與 Decoder 的 LSTM）構成。此時，LSTM 層的隱藏狀態成為 Encoder 與 Decoder 的「橋梁」。在正向傳播中，Encoder 是透過 LSTM 層的隱藏狀態，向 Decoder 傳遞編碼後的資料。在 seq2seq 的反向傳播中，透過這個「橋梁」，由 Decoder 傳遞梯度給 Encoder。

7.2.2　轉換時間序列資料用的玩具問題

接著要實際執行 seq2seq。在此之前，先說明接下來要處理的問題。我們要用「加法」處理轉換時間序列的問題。具體而言如**圖 7-10** 所示，提供「57+5」的字串給 seq2seq，學習正確回答出「62」。附帶一提，這種為了評估機器學習而設計的簡單問題，稱作「玩具問題（toy problem）」。

長度不定的時間序列資料若要進行小批次學習，最簡單的方法，就是使用**填補**（padding）技巧。填補是利用（沒有意義）無效資料填補原本的資料，讓資料長度一致的技巧。以這次的加法問題來說，如**圖 7-11** 所示，統一所有輸入資料的長度，在其餘部分插入無效字元（這裡是指「空白字元」）。

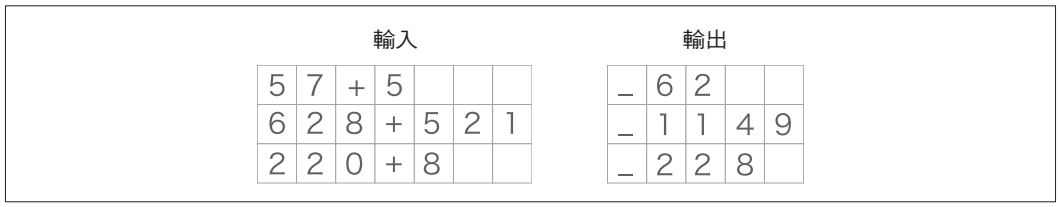

圖 7-11　進行小批次學習時，要用「空白字元」進行填補，統一輸入及輸出的資料大小

這次要處理兩個 0 ～ 999 之間的數字彼此相加的問題。因此，包含「+」符號，輸入的最大字元數是 7。另外，加法結果的最大字元數是四個字元（最大是「999+999=1998」）。因此，訓練資料也同樣進行填補，統一所有取樣資料的長度。在這次的問題中，於輸出的開頭加上「_（underscore）」當作區隔字元，用五個字元統一輸出資料。這個區隔字元是通知 Decoder 產生字串的訊號。

Decoder 的輸出可以加入通知字元輸出結束的區隔字元，當作訓練標籤（例如「_62_」或「_1149_」）。可是，這裡的內容很單純，所以沒有提供結束產生字串的區隔字元。換句話說，Decoder 在產生字串時，只會輸出已經決定的字元數（這裡包含「_」是五個字元）。

像這樣，使用填補技巧統一資料大小，就能處理長度不定的時間序列資料。可是，使用填補，seq2seq 必須處理原本不存在的填補用字元。為了慎重起見（使用填補時），要在 seq2seq 加入填補專用處理。假設在 Decoder 輸入填補時，不計算損失結果（在 Softmax with Loss 層，加上「mask」功能就可以處理）。另外，在 Encoder 中，輸入填補後，LSTM 層會直接輸出上一個時刻的輸入。這樣 LSTM 層可以把填補當作不存在，將輸入資料編碼。

這裡的說明有點複雜，即使你無法理解也沒有關係。本章是以容易理解為優先，所以不對填補用的字元（空白字元）做特殊處理，而是當成一般資料。

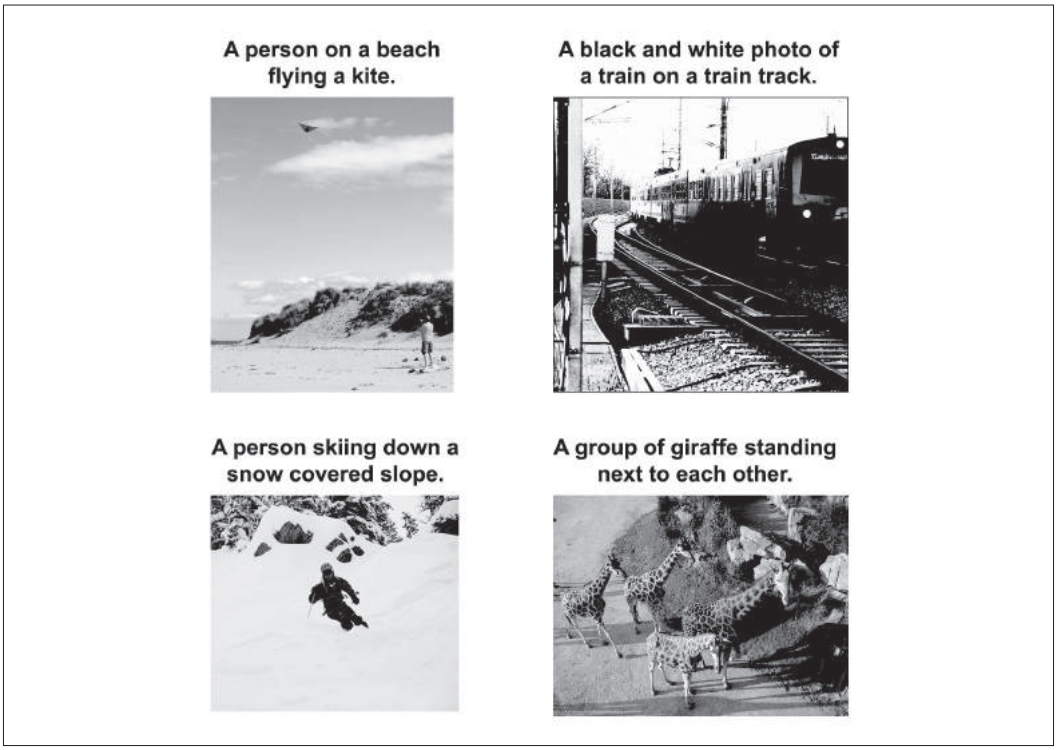

圖 7-32　圖像描述的範例：將影像轉換成文字（影像擷取自文獻 [47]）

檢視圖 7-32，「A person on a beach flying a kite（在海灘放風箏的人）」、「A black and white photo of a train on train track（軌道上的火車黑白照）」等，可以說產生了非常良好的結果。當然，能實現這一點，是因為有了大量影像與說明內容等學習資料（還有 ImageNet 等大量影像資料），再加上使用能有效學習這些學習資料的 seq2seq，才得到了如圖 7-32 的優異結果。

7.6　重點整理

本章進入利用 RNN 產生文章的主題。實際上是在上一章說明過 RNN 的語言模型中，加入一些改良，新增了產生文章的功能。本章的後半部分成功處理了 seq2seq，學習了簡單的加法問題。seq2seq 是連結 Encoder 與 Decoder 的模型，組合兩個 RNN 的單純結構。儘管非常簡單，seq2seq 卻隱藏著極大的可能性，能發揮在各式各樣的應用程式上。