第一章:遞歸神經網絡的基礎

Recurrent Neural Networks (RNN)

在當今的人工智能領域，遞歸神經網絡（RNN）扮演著重要的角色。RNN是一種專門用於處理序列數據的神經網絡，能夠有效處理一系列值 $x(1),...,x(\tau)$。這種網絡的特點在於其能夠處理比傳統神經網絡更長的序列，這對於許多實際應用來說是非常重要的。

RNN的特點和優勢

RNN的一大優勢是其能夠處理可變長度的序列。這意味著無論輸入數據的長度如何，RNN都能夠有效地處理。這是通過在模型的不同部分共享參數來實現。參數共享不僅使模型能夠應對不同長度的數據，還能夠在不同形式的例子之間進行泛化。

核心特性:參數共享

在探索從多層網絡過渡到遞歸神經網絡（RNN）的過程中，一個關鍵的概念是在模型的不同部分之間共享參數。這種參數共享策略不僅允許模型靈活地應對各種形式的數據（在RNN中指不同長度的序列），還使得模型能夠在這些不同形式的數據之間進行有效的泛化。

如果我們為時間序列中的每個點分配獨立的參數，那麼模型將無法適應那些在訓練過程中未曾出現過的序列長度。此外，這種方法也無法在不同長度的序列和時間序列中的不同位置之間共享學習到的知識。這就是為什麼在處理包含重複或類似信息的序列時，參數共享變得尤為重要。

舉一個具體的例子，假設我們有兩個句子:“I went to Nepal in 2009”和“In 2009, I went to Nepal。”在這兩個句子中，「2009年」這個信息對於理解敘述者何時訪問尼泊爾是關鍵的，無論它出現在句子的哪個位置。如果使用一個機器學習模型來解析這些句子並提取訪問年份，理想的情況是模型能夠識別出「2009年」這一信息的重要性，不論它位於句子的開頭、中間還是結尾。這正是參數共享在遞歸神經網絡中發揮的關鍵作用，它使得模型能夠靈活地處理各種結構的語言數據，並從中提取關鍵信息。

與傳統神經網絡的比較

相比於傳統的全連接前饋網絡，RNN在處理語言等序列數據時有顯著優勢。傳統網絡需要在句子的每個位置單獨學習語言規則，而RNN通過在幾個時間步中共享相同的權重來解決這個問題。

RNN的深度和時間維度

在深入探討神經網絡的不同架構時，我們發現1-D時間序列卷積和遞歸神經網絡（RNN）在參數共享方面各有特色。1-D時間序列的卷積允許網絡在時間維度上共享參數，這種共享雖然有效，但相對來說是較為淺層的。在這種架構中，卷積的輸出形成一個序列，每個輸出元素都是基於輸入序列中一小部分鄰近元素的函數。

相比之下，遞歸網絡在參數共享方面採用了一種不同的方法。在RNN中，輸出序列的每個元素都是通過對先前的輸出應用相同的更新規則來生成的。這種遞歸的公式使得參數在一個非常深的計算圖中被共享，從而為模型提供了更深層次的時間序列處理能力。為了簡化解釋，我們通常將RNN描述為操作在包含向量 x(t) 的序列上，其中時間步索引 t 的範圍從1到 τ。值得注意的是，這裡的時間步索引並不一定與現實世界中時間的流逝直接相關，有時它僅僅指的是序列中的位置。
更進一步地，RNN的應用不僅限於一維時間序列數據。它們也可以應用於二維空間數據，如圖像處理。此外，即使在處理涉及時間的數據時，RNN也可能包含向後在時間中的連接。這種設計前提是在將整個序列提供給網絡之前，已經觀察到了序列的全部內容。這使得RNN能夠在處理時間序列數據時，提供更全面、更深入的理解和分析。

Unfolding Computational Graphs

這涉及將RNN的循環結構展開成一系列重複的網絡層，以便更好地理解和實現其運作。將遞迴或循環計算展開成一個具有重複結構的計算圖，通常對應於一連串事件，導致在深度網絡結構中共享參數。

e.g.1 考慮一個動態系統的經典形式

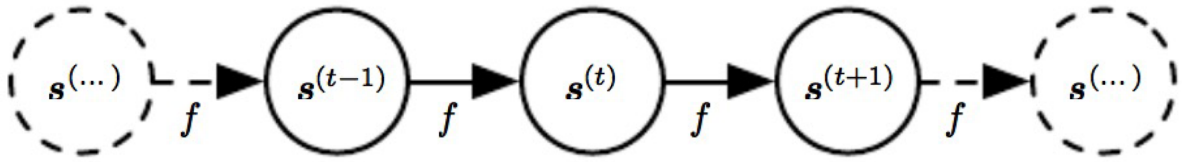
$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta}),$$

其中 s(t) 被稱為系統的狀態。
(方程式是遞迴的，因為在時間 t 的 s 的定義參考了時間 t − 1 的相同定義。)

對於有限數量的時間步驟 τ，可以通過應用定義 τ − 1 次來展開圖形。例如，如果我們對方程式$s^{(t)}$ 展開 τ = 3 時間步驟，我們得到右圖

$$\begin{aligned}\boldsymbol{s}^{(3)} &= f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}) \\ &= f(f(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta}).\end{aligned}$$

通過這種方式重複應用定義來展開方程式，產生了一個不涉及遞迴的表達式。這樣的表達式現在可以用傳統的有向無環計算圖來表示。



e.g. 2 考慮一個由外部信號 x**(t)** 驅動的動態系統

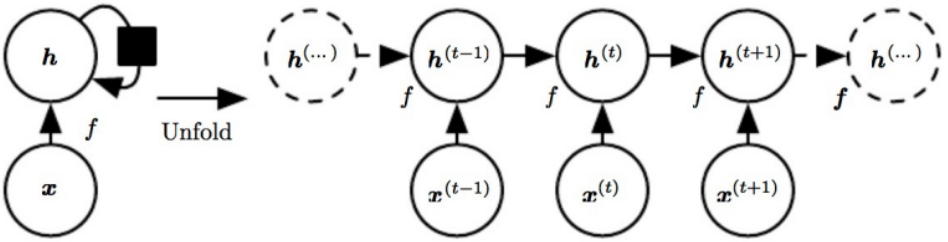
$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

我們看到狀態現在包含了關於整個過去序列的信息。遞迴神經網絡可以以許多不同的方式構建。本質上，任何涉及遞迴的函數都可以被認為是一個遞迴神經網絡。
為了表示狀態(state)是網絡的隱藏單元(hidden units of the network)，我們現在使用變量 h 來代表狀態，重寫上述方程式。此外，典型的 RNN 會添加額外的架構特徵，例如輸出層，從狀態 $\boldsymbol{h}$ 中讀取信息以進行預測。

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}),$$

當遞迴網絡被訓練來執行從過去預測未來的任務時，network通常學會使用 $\boldsymbol{h}^{(t)}$ 作為一種關於過去輸入序列的任務相關方面的有損摘要(lossy summary)。
這種摘要通常是有損的，因為它將任意長度的序列 ($\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, ..., \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}$) 映射到固定長度的向量 $\boldsymbol{h}^{(t)}$。根據訓練標準，這種摘要可能會選擇性地保留過去序列的某些方面，比其他方面更精確。例如，如果 RNN 用於統計語言建模，通常是根據先前的單詞預測下一個單詞，存儲到時間 t 的輸入序列中的所有信息可能不是必需的；只需存儲足夠的信息來預測句子的其餘部分就足夠了。最具挑戰性的情況是當我們要求 $\boldsymbol{h}^{(t)}$ 足夠豐富，以便大致恢復輸入序列，如在自編碼器框架中。
方程式 10.5 可以用兩種不同的方式繪製:

1. 繪製 RNN 的一種方式是使用一個包含模型物理實現中可能存在的每個組件的節點的圖。在這種觀點中，網絡定義了一個在實時運行的電路，其物理部件的當前狀態可以影響其未來狀態。



我們使用黑色方塊來表示與單個時間步驟的延遲發生交互，從時間 t 的狀態到時間 t + 1 的狀態。

2. 繪製 RNN 的另一種方式是作為一個展開的計算圖，其中每個組件由許多不同的變量表示，每個時間步驟一個變量，代表該時間點的組件狀態。每個時間步驟的每個變量都繪製為計算圖的一個單獨節點。我們所說的展開是將電路映射到具有重複部分的計算圖的操作。展開的圖現在具有取決於序列長度的大小。我們可以用函數 g(t) 來表示 t 步後的展開遞迴:

$$\begin{aligned}\boldsymbol{h}^{(t)} &= g^{(t)}(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}) && (10.6)\\ &= f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}). && (10.7)\end{aligned}$$

函數 $g^{(t)}$ 接受整個過去序列 ($\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, ..., \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}$) 作為輸入，並產生當前狀態，但展開的遞迴結構允許我們將 g(t) 分解為重複應用函數 f。

因此，展開過程具有兩個主要優勢:

1. 不論序列長度如何，學習的模型始終具有相同的輸入大小，因為它是根據從一個狀態到另一個狀態的轉換來指定的，而不是根據可變長度的狀態歷史來指定的。
2. 可以在每個時間步驟使用相同的轉換函數 f 和相同的參數。

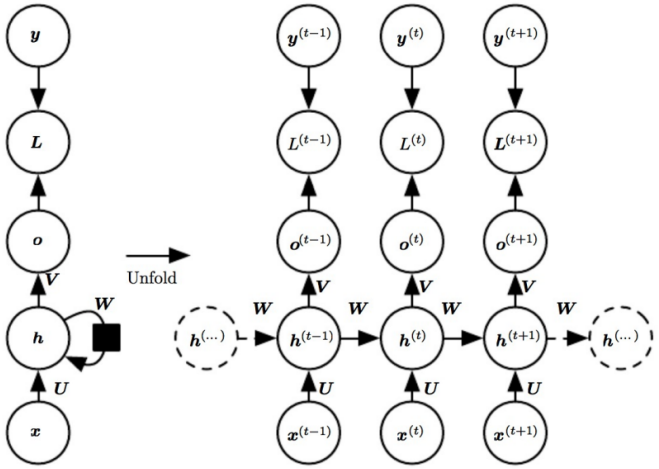
這兩個因素使得可以學習一個在所有時間步驟和所有序列長度上運行的單一模型 f。學習一個共享的單一模型允許對訓練集中未出現的序列長度進行泛化，並使模型能夠用遠少於訓練示例來估計。

遞迴圖和展開圖都有其用途:

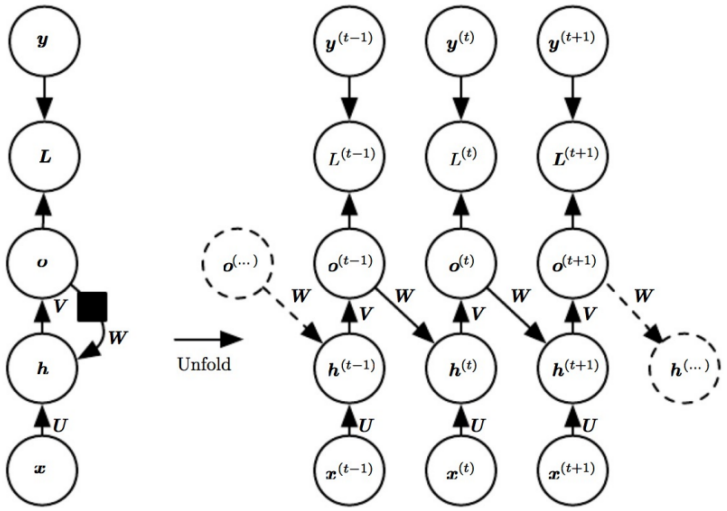
1. 遞迴圖是簡潔的。
2. 展開圖提供了執行哪些計算的明確描述。展開圖還有助於通過明確顯示信息流動的路徑來說明信息向前和向後流動的概念。
3. 展開圖提供了執行哪些計算的明確描述。展開圖還有助於通過明確顯示信息流動的路徑來說明信息向前和向後流動的概念。

下面三張圖片展示了一些遞迴神經網絡的重要設計模式的例子。

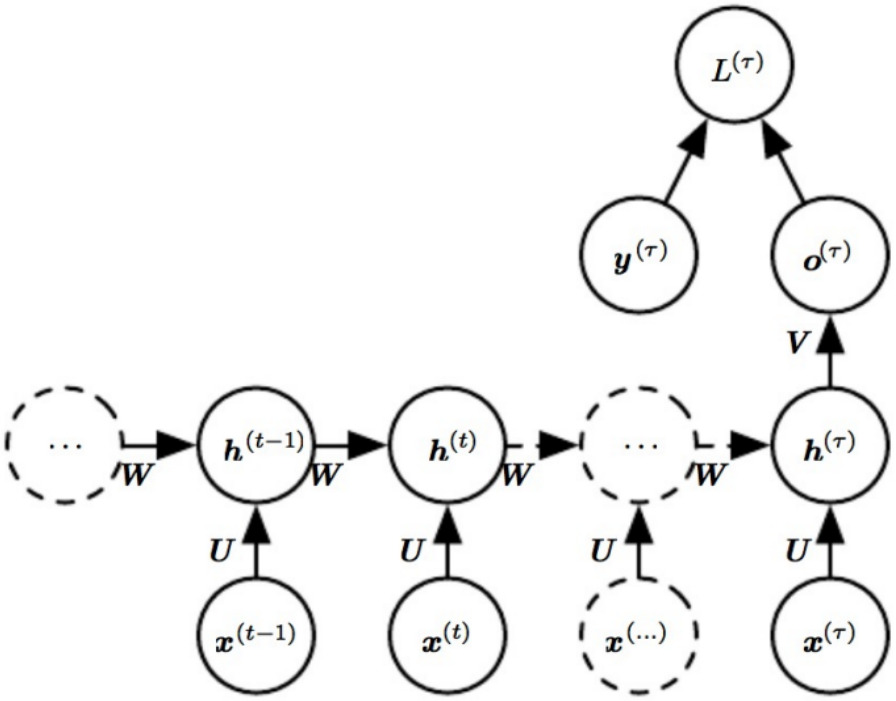
1. (圖 10.3) 在每個時間步驟產生輸出並在隱藏單元之間具有遞迴連接的遞迴網絡。



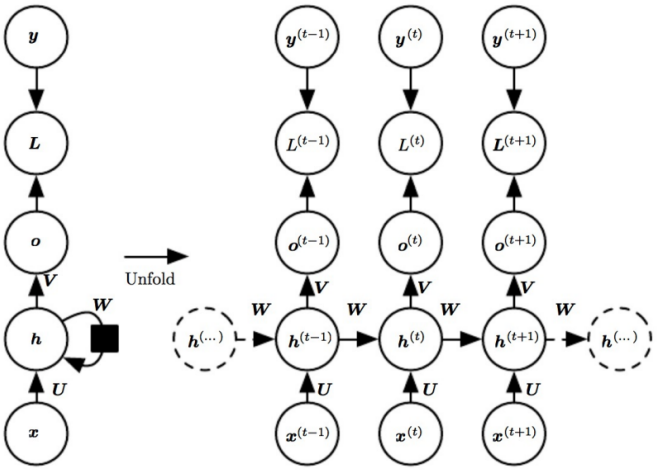
2. (圖 10.4) 在每個時間步驟產生輸出並僅從一個時間步驟的輸出到下一個時間步驟的隱藏單元之間具有遞迴連接的遞迴網絡。



3. (圖 10.5) 在隱藏單元之間具有遞迴連接，讀取整個序列然後產生單一輸出的遞迴網絡。



現在為圖 **10.3** 中描繪的 **RNN** 開發前向傳播方程式



我們假設:

1. 使用雙曲正切激活函數
2. 輸出是離散的，就像 RNN 被用來預測單詞一樣

表示離散變量的一種自然方式是將輸出 o 視為給出每個可能的離散變量值的未標準化對數概率。然後我們可以應用 softmax 操作作為後處理步驟，以獲得一個關於輸出的標準化概率向量 $\hat{y}$。

前向傳播從指定初始狀態 $\boldsymbol{h}^{(0)}$開始。然後，從 $t = 1$ 到 $t = \tau$ 的每個時間步驟，我們應用以下更新方程式:

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \quad (10.8)$$
$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)}), \quad (10.9)$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \quad (10.10)$$
$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)}), \quad (10.11)$$

其中參數是偏置向量 $\boldsymbol{b}$ 和 $\boldsymbol{c}$，以及分別用於輸入到隱藏、隱藏到輸出和隱藏到隱藏連接的權重矩陣 $\boldsymbol{U}$、$\boldsymbol{V}$ 和 $\boldsymbol{W}$。
給定一系列 x 值與 y 值配對的總損失，則是所有時間步驟損失的總和。
例如，如果 L(t) 是給定 $\boldsymbol{x}$(1), . . . , $\boldsymbol{x}$(t) 的 y(t) 的負對數似然，則

$$L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right) \quad (10.12)$$
$$= \sum_t L^{(t)} \quad (10.13)$$
$$= -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right), \quad (10.14)$$

其中$p_{model}(y^{(t)}|\{x^{(1)},..., x^{(t)}\})$ 通過從模型的輸出向量 $y^{(t)}$ 中讀取 $\hat{y}^{(t)}$ 的條目來給出。

計算這個損失函數相對於參數的梯度是一個複雜的操作。梯度計算涉及執行一次前向傳播，接著是一次反向傳播。
運行時間是 $O(\tau)$，無法通過並行化來減少，因為前向傳播圖本質上是順序的；每個時間步驟只能在前一個完成後計算。
在前向傳遞中計算的狀態必須存儲，直到在反向傳遞中重用，因此記憶成本也是 $O(\tau)$。
應用於展開圖的反向傳播算法，其成本為 $O(\tau)$，稱為時間反向傳播（BPTT）

Computing the Gradient in a RNN

通過遞歸神經網絡計算梯度是直接的。只需將第6.5.6節的推廣反向傳播算法應用於展開的計算圖，不需要專門的算法。通過反向傳播獲得的梯度，然後可以與任何通用的基於梯度的技術一起使用來訓練RNN。
為了獲得對BPTT算法行為的一些直觀理解，我們提供了一個如何通過BPTT計算方程式10.8和10.12的梯度的例子：

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \quad (10.8)$$
$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)}), \quad (10.9)$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \quad (10.10)$$
$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)}), \quad (10.11)$$

對於每個節點**N**，我們需要基於在圖中跟隨它的節點上計算的梯度遞歸地計算梯度$\nabla_{\mathbf{N}} L$。我們從緊接在最終損失之前的節點開始遞歸：

$$\frac{\partial L}{\partial L^{(t)}} = 1. \tag{10.17}$$

在這個推導中，我們假設輸出$\boldsymbol{o}^{(t)}$被用作softmax函數的參數，以獲得輸出上概率的向量$\hat{\boldsymbol{y}}$。我們還假設損失是給定到目前為止的輸入的真實目標$y^{(t)}$的負對數似然。
在時間步t的輸出上的梯度$\nabla_{\boldsymbol{o}^{(t)}} L$是：

$$\left(\nabla_{\boldsymbol{o}^{(t)}} L\right)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}}. \tag{10.18}$$

我們從序列的末尾開始向後工作。在最後的時間步τ，$\boldsymbol{h}^{(\tau)}$只有$\boldsymbol{o}^{(\tau)}$作為後代，所以它的梯度很簡單：

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^\top \nabla_{\boldsymbol{o}^{(\tau)}} L. \tag{10.19}$$

然後我們可以向後迭代時間，通過時間反向傳播梯度，從$t = \tau - 1$到$t = 1$，注意$\boldsymbol{h}^{(t)}$（對於$t < \tau$）有$\boldsymbol{o}^{(t)}$和$\boldsymbol{h}^{(t+1)}$作為後代。它的梯度因此由

$$\nabla_{\boldsymbol{h}^{(t)}} L = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top \left(\nabla_{\boldsymbol{h}^{(t+1)}} L\right) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top \left(\nabla_{\boldsymbol{o}^{(t)}} L\right) \tag{10.20}$$

$$= \boldsymbol{W}^\top \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t+1)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t+1)}} L\right) + \boldsymbol{V}^\top \left(\nabla_{\boldsymbol{o}^{(t)}} L\right), \tag{10.21}$$

給出，其中diag(1- $(\boldsymbol{h}_i^{(t+1)})^2$)表示包含元素1- $(\boldsymbol{h}_i^{(t+1)})^2$的對角矩陣。這是與時間$t$ + 1的隱藏單元i相關的雙曲正切的Jacobian matrix。

因為這些參數在許多時間步中共享，我們在表示涉及這些變量的微積分操作時必須小心。我們希望實現的方程式使用第6.5.6節的bprop方法，該方法計算計算圖中單個邊對梯度的貢獻。然而，微積分中使用的$\nabla_{\boldsymbol{W}} f$運算符考慮了$\boldsymbol{W}$對於$f$的值由於計算圖中所有邊的貢獻。
為了解決這個問題，我們引入了虛擬變量$\boldsymbol{W}^{(t)}$，它們被定義為$\boldsymbol{W}$的副本，但每個$\boldsymbol{W}^{(t)}$只在時間步t使用。然後我們可以使用$\nabla_{\boldsymbol{W}^{(t)}}$來表示時間步$t$的權重對梯度的貢獻。
剩餘參數上的梯度由

$$\nabla_{\boldsymbol{c}} L = \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}} \right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L, \tag{10.22}$$

$$\nabla_{\boldsymbol{b}} L = \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}} \right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \mathrm{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L, \tag{10.23}$$

$$\nabla_{\boldsymbol{V}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}} \right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L\right) \boldsymbol{h}^{(t)\top}, \tag{10.24}$$

$$\nabla_{\boldsymbol{W}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \tag{10.25}$$

$$= \sum_t \mathrm{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{h}^{(t-1)\top}, \tag{10.26}$$

$$\nabla_{\boldsymbol{U}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \tag{10.27}$$

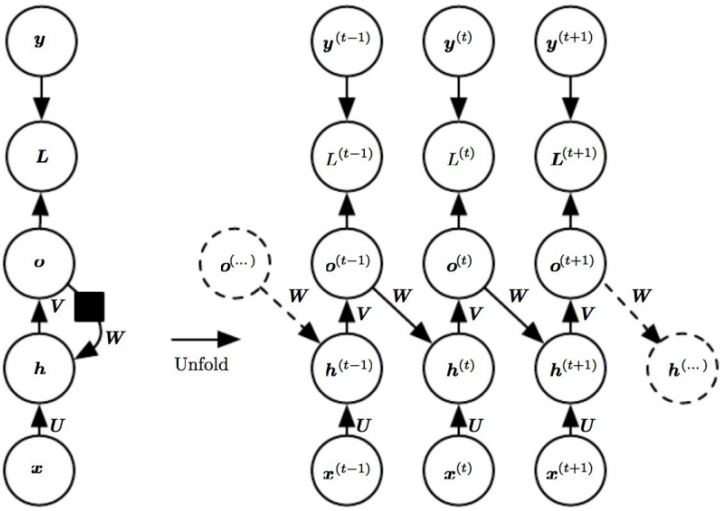
$$= \sum_t \mathrm{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{x}^{(t)\top}, \tag{10.28}$$

在訓練過程中，我們不需要計算相對於x(t)的梯度，因為在計算圖中，x(t)並不是任何參數的前驅。

第二章：RNN的訓練和優化

Teacher Forcing and Networks with Output Recurrence

在隱藏單元之間具有遞歸的網絡因此而非常強大，但訓練起來也很昂貴。有沒有其他選擇？我們可以消除隱藏層到隱藏層之間的遞歸。
只有從一個時間步的輸出到下一個時間步的隱藏單元之間的遞歸連接的網絡（如圖10.4所示）威力明顯較弱，因為它缺乏隱藏層之間的遞歸連接。



由於這個網絡缺乏隱藏層之間的遞歸，它要求輸出單元捕捉所有關於過去的信息，這些信息將被網絡用來預測未來。

由於輸出單元被明確訓練以匹配訓練集目標，除非用戶知道如何描述系統的完整狀態並將其作為訓練集目標的一部分提供，否則它們不太可能捕捉到輸入的過去歷史所需的信息。

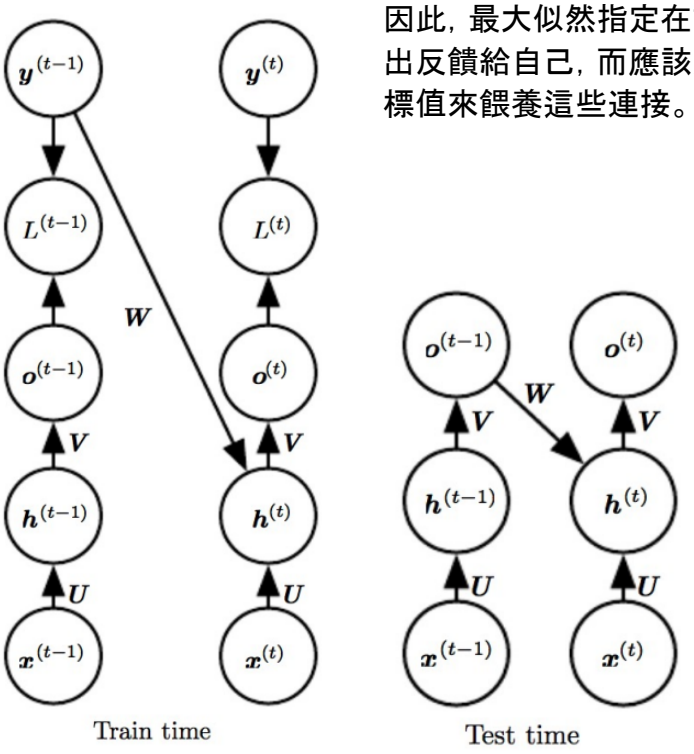
消除隱藏層之間遞歸的優勢在於，對於任何基於比較時間t的預測與時間t的訓練目標的損失函數，所有時間步都被解耦。因此，訓練可以並行化，每個步驟t的梯度可以獨立計算。

那些從輸出回流到模型中的遞歸連接的模型可以通過Teacher Forcing來訓練。Teacher Forcing 是一種在訓練過程中模型接收真實輸出$y^{(t)}$作為時間 t+1的輸入的程序。我們可以通過檢查一個有兩個時間步的序列來看到這一點。條件最大似然標準是

$$\log p\left(\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)} \mid \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right) \tag{10.15}$$

$$= \log p\left(\boldsymbol{y}^{(2)} \mid \boldsymbol{y}^{(1)}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right) + \log p\left(\boldsymbol{y}^{(1)} \mid \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right). \tag{10.16}$$

在這個例子中，我們看到在時間t = 2，模型被訓練以最大化給定到目前為止的$\boldsymbol{x}$序列和訓練集中的前一個$\boldsymbol{y}$值的$\boldsymbol{y}^{(2)}$的條件概率。



因此，最大似然指定在訓練期間，不應將模型自己的輸出反饋給自己，而應該用指定正確輸出應該是什麼的目標值來餵養這些連接。

我們最初激勵Teacher forcing 是為了避免在缺乏隱藏層到隱藏層連接的模型中進行通過時間的反向傳播。只要模型具有從一個時間步的輸出到下一個時間步計算的值的連接，Teacher forcing 仍然可以應用於具有隱藏層到隱藏層連接的模型。然而，一旦隱藏單元成為早期時間步的函數，BPTT算法就變得必要。因此，一些模型可能會同時使用Teacher forcing和BPTT進行訓練。

Clipping Gradients

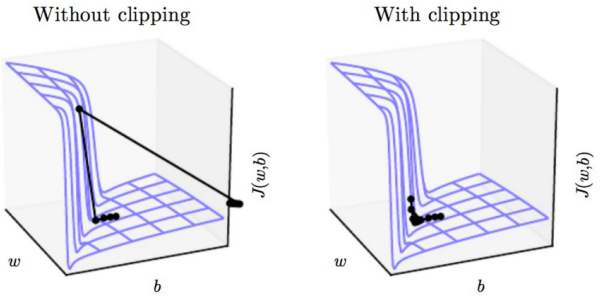
當參數梯度非常大時，梯度下降參數更新可能會將參數拋到目標函數較大的區域很遠的地方。

一種簡單的解決方案已經被實踐者使用了多年：裁剪梯度。一種選擇是在參數更新之前，對minibatch element-wise逐個裁剪參數梯度。另一種是在參數更新之前裁剪梯度的範數$||\boldsymbol{g}||$：

$$\text{if } ||\boldsymbol{g}|| > v \tag{10.48}$$

$$\boldsymbol{g} \leftarrow \frac{\boldsymbol{g}v}{||\boldsymbol{g}||}, \tag{10.49}$$

其中v是範數閾值，$\boldsymbol{g}$用於更新參數。



因為所有參數的梯度(包括不同組的參數，如權重和偏差)是用單一的縮放因子聯合重新標準化的，裁剪範數的優勢在於保證每一步仍然是沿著梯度方向的，但實驗表明兩種形式工作得相似。事實上，即使當梯度大小超過閾值時隨機取一步也幾乎同樣有效。如果梯度爆炸嚴重到數值上是無限大或非數(Inf或Nan)，那麼可以隨機取一個大小為v的步驟，通常會遠離數值不穩定的配置。

Regularizing to Encourage Information Flow

梯度裁剪有助於處理梯度爆炸，但它對於梯度消失無效。為了解決梯度消失和更好地捕捉長期依賴，我們討論了在展開的遞歸架構的計算圖中創建路徑的想法，沿這些路徑，與弧相關的梯度乘積接近1。實現這一目標的一種方法是使用LSTM和其他自循環和門控機制。

另一個想法是規範或限制參數，以鼓勵“信息流動(information flow)”。我們希望被反向傳播的梯度向量 $\nabla_{\boldsymbol{h}^{(t)}} L$ 即使在損失函數只懲罰序列末尾的輸出時，也能保持其大小。

我們希望

$$(\nabla_{\boldsymbol{h}^{(t)}} L) \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \tag{10.50}$$

與

$$\nabla_{\boldsymbol{h}^{(t)}} L. \tag{10.51}$$

一樣大。

有了這個目標，提出了這個正則化器(regularizer)

$$\Omega = \sum_t \left(\frac{\left| \left| (\nabla_{\boldsymbol{h}^{(t)}} L) \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \right| \right|}{\left| \left| \nabla_{\boldsymbol{h}^{(t)}} L \right| \right|} - 1 \right)^2. \tag{10.52}$$

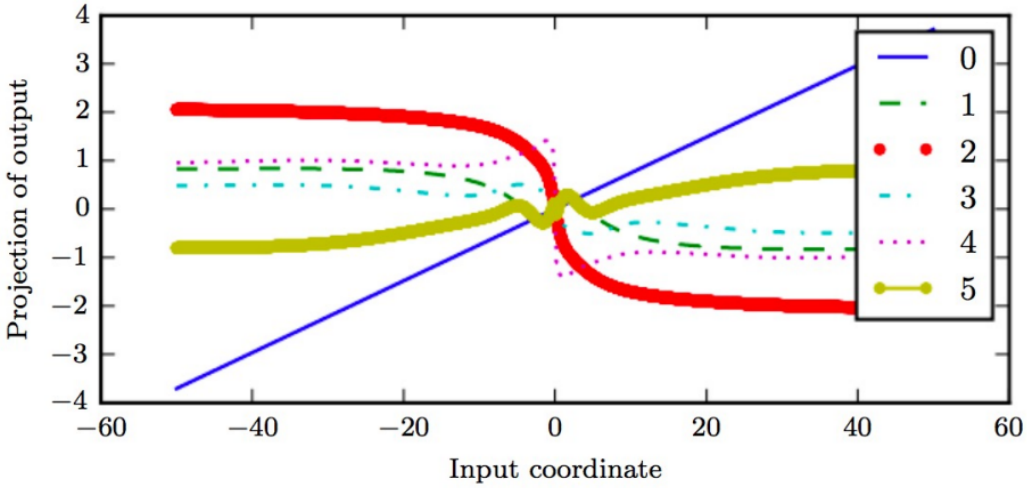
計算這個正則化器的梯度可能看起來困難，但提出了一種近似方法，我們將反向傳播的向量$\nabla_{\boldsymbol{h}^{(t)}} L$ 視為常數。

這個正則化器的實驗表明，如果與範數裁剪(norm clipping)啟發式相結合，正則化器可以顯著增加RNN可以學習的依賴範圍。因為它使RNN動態處於梯度爆炸的邊緣，梯度裁剪尤其重要。這種方法的一個主要弱點是，對於數據豐富的任務，它不如LSTM有效。

第三章：處理長期依賴問題

The Challenge of Long-Term Dependencies

在遞歸網絡中學習長期依賴的基本問題是，經過多階段傳播的梯度傾向於消失（大多數情況下）或爆炸（雖然罕見，但對優化造成很大的損害）。即使我們假設參數是使遞歸網絡穩定的（可以存儲記憶，梯度不會爆炸），長期依賴的難點來自於與短期相比，長期交互作用（涉及多個雅可比矩陣的乘積）被賦予指數級較小的權重。
遞歸網絡涉及多次復合同一函數，每個時間步一次。這些復合可能導致極端的非線性行為。



遞歸神經網絡（RNN）所使用的函數復合在某種程度上類似於矩陣乘法。遞歸關係

$$\boldsymbol{h}^{(t)} = \boldsymbol{W}^{\top} \boldsymbol{h}^{(t-1)} \tag{10.36}$$

可以被認為是一個非常簡單的遞歸神經網絡，缺乏非線性激活函數，也缺乏輸入$\boldsymbol{x}$。它可以被簡化為

$$\boldsymbol{h}^{(t)} = \left(\boldsymbol{W}^{t}\right)^{\top} \boldsymbol{h}^{(0)}, \tag{10.37}$$

如果$\boldsymbol{W}$允許形式為

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\top}, \tag{10.38}$$

的特徵分解，並且$\boldsymbol{Q}$是正交的，則遞歸可以被簡化為

$$\boldsymbol{h}^{(t)} = \boldsymbol{Q}^{\top}\boldsymbol{\Lambda}^{t}\boldsymbol{Q}\boldsymbol{h}^{(0)}. \tag{10.39}$$

特徵值被提升到t的次方，導致小於1的特徵值衰減為零，大於1的特徵值爆炸。
想像將一個權重ω多次自乘。乘積ω^{t}將根據ω的大小消失或爆炸。
如果我們製作一個非遞歸網絡，在每個時間步有不同的權重$\omega^{(t)}$，情況就不同了。如果初始狀態由1給出，那麼時間t的狀態由$\prod_{t} \omega^{(t)}$給出。假設$\omega^{(t)}$值是隨機獨立生成的，彼此獨立，平均值為零且方差為v。乘積的方差是$O(v^{n})$。為了獲得一些期望的方差v^{*}，我們可以

選擇方差為 $v = \sqrt[n]{v^*}$ 的個別權重。因此，非常深的feedforward networks在仔細選擇縮放的情況下可以避免梯度消失和爆炸問題。
每當模型能夠表示長期依賴時，長期交互作用的梯度與短期交互作用的梯度相比具有指數級較小的大小。這意味著學習並非不可能，但可能需要很長時間才能學習長期依賴，因為這些依賴的信號往往會被來自短期依賴的最小波動所掩蓋。隨著我們增加需要捕捉的依賴範圍，基於梯度的優化變得越來越困難，通過SGD成功訓練傳統RNN的概率迅速接近於0，即使是對於長度僅為10或20的序列。

Leaky Units and Other Strategies for Multiple Time Scales

處理長期依賴問題的一種策略是建立一個能在多個時間層面上操作的模型。這樣的模型結構使得其一部分能在細微的時間尺度上進行工作，專注於處理細節問題，而另一部分則在較大的時間尺度上運作，有效地從過去傳遞信息到當前。
為了實現這種細粒度和粗粒度時間尺度的結合，可以採用多種策略，包括：

- 在不同時間點之間建立跳躍連接
- 使用“leaky units”來整合具有不同時間常數的信號
- 刪除一些用於模擬細粒度時間尺度的連接

獲得導數乘積接近於1的路徑的另一種方法是使用具有線性自我連接和權重接近於1的units。例如，當我們通過應用更新公式running average $\mu^{(t)} \leftarrow \alpha\mu^{(t-1)} + (1 - \alpha)\, v^{(t)}$來累積某個值 $v^{(t)}$ 的運行平均 $\mu^{(t)}$ 時，這裡的 α 參數就是從 $\mu^{(t-1)}$ 到 $\mu^{(t)}$ 的線性自我連接的一個實例。當 α 接近於1時，這個運行平均(running average)會長時間記住過去的信息；而當 α 接近於零時，過去的信息則會迅速被遺忘。
具有線性自連接的隱藏單元可以表現出類似於這種運行平均的行為，這樣的隱藏單元被稱為leaky units。使用接近於1的權重的線性自連接是確保單元能夠訪問過去值的另一種方式。線性自連接方法允許通過調整實數值的 α 而不是調整整數值的跳躍長度，使這種效果更加平滑和靈活地適應。

Adding Skip Connections Through Time

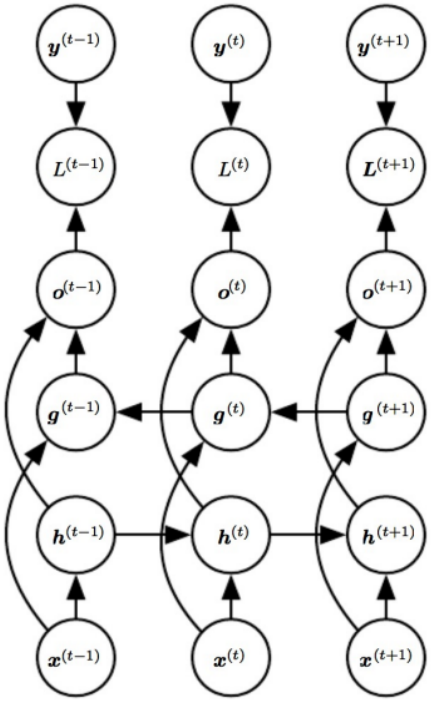
在處理長期依賴問題方面，透過時間的跳躍連接是一種有效的策略。這種方法通過在神經網絡中添加直接連接，將過去的變量與當前的變量連接起來，從而獲得粗略的時間尺度。
在這種設置中，對於具有時間延遲d的遞歸連接，梯度會以τ/d的函數指數級衰減，而不是單純的τ。這種方法有助於減緩梯度消失的問題，這是遞歸神經網絡在處理長期依賴時常見的挑戰。
然而，即使有了這些跳躍連接，梯度在τ中仍可能指數級爆炸。這表明，雖然跳躍連接可以改善長期依賴問題，但它們並不能完全解決梯度爆炸的問題。因此，在設計和訓練遞歸神經網絡時，仍需要考慮其他技術和策略，如長短期記憶（LSTM）網絡或閘控遞歸單元（GRU），這些都是專門為解決這些挑戰而設計的。
總結來說，跳躍連接通過時間提供了一種處理長期依賴問題的有效方法，但它們需要與其他技術結合使用，以充分解決遞歸神經網絡中的梯度消失和爆炸問題。

第四章:RNN的進階變體

Bidirectional RNNs(雙向循環神經網絡)

我們目前考慮的所有循環網絡都具有“因果”結構，這意味著時間t的狀態只捕捉來自過去的信息，$\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t-1)}$，以及當前輸入$\boldsymbol{x}^{(t)}$。然而，在許多應用中我們希望輸出的預測$y^{(t)}$可能依賴於整個輸入序列。例如，在語音識別中，當前聲音作為音素的正確解釋可能依賴於接下來的幾個音素，因為共同發音，甚至可能依賴於接下來的幾個詞，因為附近詞之間的語言依賴性。

雙向循環神經網絡（或雙向RNN）就是為了滿足這種需求而發明的。它們在需要這種需求的應用中非常成功，例如手寫識別、語音識別和生物信息學。正如名稱所暗示的，雙向RNN結合了一個從序列開始向前通過時間移動的RNN，和另一個從序列結束向後通過時間移動的RNN。



$\boldsymbol{h}^{(t)}$代表向前通過時間移動的子RNN的狀態，$\boldsymbol{g}^{(t)}$代表向後通過時間移動的子RNN的狀態。這允許輸出單元$\boldsymbol{o}^{(t)}$計算一個依賴於過去和未來的表示，但對時間t附近的輸入值最為敏感，而不必指定圍繞t的固定大小窗口。

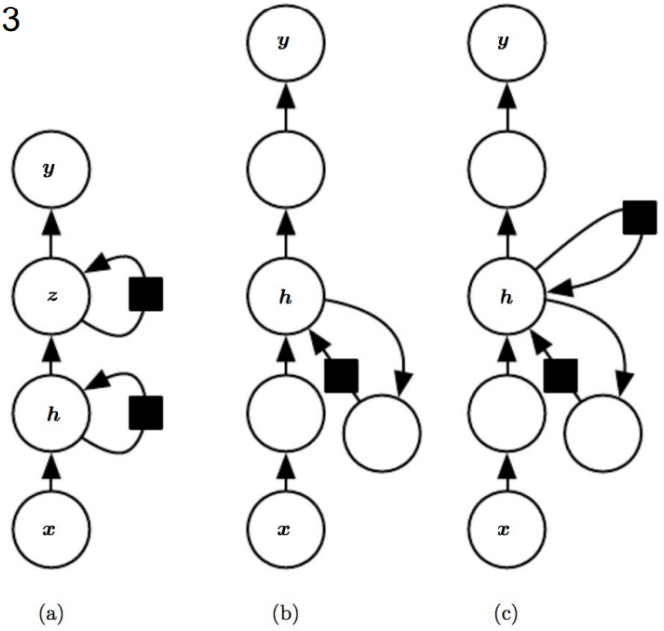
這個想法可以自然地擴展到二維輸入，例如圖像，通過擁有四個RNN，每個都朝一個方向移動：上、下、左、右。在2-D網格的每個點(i, j)，輸出$O_{i,j}$可以計算一個表示，主要捕捉局部信息，但也可能依賴於長距離輸入。

與卷積網絡相比，應用於圖像的RNN通常更昂貴，但允許在同一特徵圖中的特徵之間進行長距離橫向交互。

Deep Recurrent Networks(深度循環網絡)

RNN中的計算可以分解為三個參數塊和相關變換：從輸入到隱藏狀態，從前一個隱藏狀態到下一個隱藏狀態，以及從隱藏狀態到輸出。

Fig. 10.13



在圖10.3的RNN架構中，這三個塊中的每一個都與一個單一的權重矩陣相關聯。換句話說，當網絡展開時，這些塊中的每一個都對應於一個淺層變換（由深層MLP內的單一層表示）。實驗強烈表明，在這些操作中引入深度是有利的。

將RNN的狀態分解為多個層次，如圖10.13a所示，具有顯著的好處。我們可以將層次較

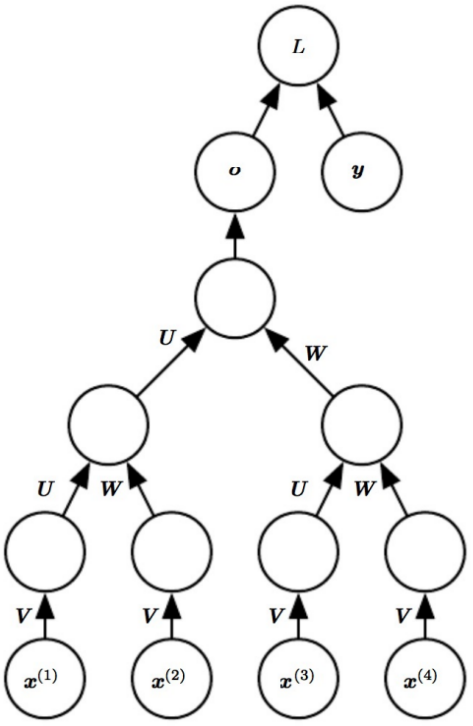
低的層視為在將原始輸入轉換為更適合於隱藏狀態較高層次的表示中發揮作用。

進一步的步驟可以通過為上述三個塊中的每一個都有一個單獨的MLP（可能是深層的），如圖10.13b所示。考慮到表示能力，建議在這三個步驟中分配足夠的容量。增加深度可能會通過使優化變得困難而損害學習。一般來說，優化較淺的架構更容易，而增加圖10.13b的額外深度使得從時間步t到時間步t+1的變量之間的最短路徑變得更長。例如，如果用單隱藏層的MLP用於狀態到狀態的轉換，我們就將任何兩個不同時間步中變量之間的最短路徑的長度加倍，與圖10.3的普通RNN相比。

然而，這可以通過在隱藏到隱藏的路徑中引入跳躍連接來緩解，如圖10.13c所示。

Recursive Neural Networks(遞歸神經網絡)

遞歸神經網絡是循環網絡的另一種推廣，具有不同類型的計算圖，其結構為深層樹狀，而不是RNN的鏈式結構。遞歸網絡已成功應用於將數據結構作為神經網絡的輸入，在自然語言處理以及計算機視覺中。
遞歸網絡相對於循環網絡的一個明顯優勢是，對於相同長度τ的序列，深度可以從τ大幅減少到$O(\log \tau)$，這可能有助於處理長期依賴性。



一個懸而未決的問題是如何最好地構建樹結構。一種選擇是擁有一個不依賴於數據的樹結構，例如平衡二叉樹。
在某些應用領域中，外部方法可以建議適當的樹結構。例如，在處理自然語言句子時，遞歸網絡的樹結構可以固定為由自然語言解析器提供的句子的解析樹結構。理想情況下，希望學習者本身能夠發現並推斷出適合任何給定輸入的樹結構。

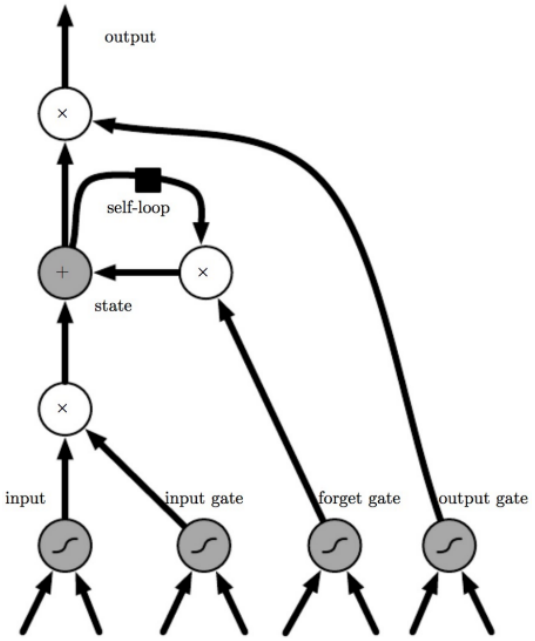
第五章:門控遞歸神經網絡

The Long Short-Term Memory and Other Gated RNNs

在實際應用中，最有效的序列模型之一被稱為閘控遞歸神經網絡（Gated Recurrent Neural Networks, RNNs）。這些模型包括長短期記憶網絡（Long Short-Term Memory, LSTM）和基於閘控遞歸單元（Gated Recurrent Unit, GRU）的網絡。這些閘控RNN的設計基於創建時間路徑的理念，旨在解決傳統RNN中的梯度消失或爆炸問題。
長短期記憶網絡（LSTM）是一種特殊的RNN架構，旨在更好地學習長期依賴關係。LSTM通過引入三個閘控機制——遺忘閘、輸入閘和輸出閘——來實現這一目標。遺忘閘控制保留多少之前的信息，輸入閘決定新信息的重要性，而輸出閘則決定從單元狀態到輸出的信息量。
閘控遞歸單元（GRU）是另一種流行的RNN變體，它簡化了LSTM的結構，通常只有兩個閘控：更新閘和重置閘。更新閘幫助模型決定在當前狀態中保留多少過去的信息，而重置閘則決定忘記多少過去的信息。
滲漏單元（Leaky units）是一種機制，它通過手動選擇的常數或參數的連接權重來實現信息的長期保留。這使得網絡能夠在長時間內積累信息，例如對特定特徵或類別的證據。然而，這種方法的局限性在於，一旦信息被使用，忘記舊狀態對於神經網絡來說可能是有益的，特別是在處理由多個子序列構成的序列時。
閘控RNN解決了這一問題，它通過動態調整連接權重來實現對信息的保留和遺忘。在這種架構下，神經網絡學會了何時保留或清除狀態，而不是依賴於手動設置。這使得閘控RNN在處理具有複雜時間依賴關係的序列數據時更加有效和靈活。

LSTM

LSTM模型使這個自循環的權重取決於上下文，而不是固定的。通過使這個自循環的權重閘控（由另一個隱藏單元控制），根據輸入序列，集成的時間尺度可以動態改變，因為時間常數由模型輸出。LSTM遞歸網絡具有“LSTM cells”，除了RNN的外部循環之外，還具有內部循環（自循環）。每個單元具有與普通遞歸網絡相同的輸入和輸出，但還具有更多參數和一組閘控單元，控制信息流。



最重要的組件是具有類似於leaky i units的線性自循環的state unit $s_i^{(t)}$。

自循環權重(或相關的時間常數)由忘記閘單元$f_i^{(t)}$控制(for time step t and unit i)，該單元通過一個sigmoid單元將該權重設定為0到1之間的值:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right), \tag{10.40}$$

其中$\boldsymbol{x}^{(t)}$是當前輸入向量，$\boldsymbol{h}^{(t)}$是當前隱藏層向量，包含所有LSTM單元的輸出，$\boldsymbol{b}^f$, $\boldsymbol{U}^f$, $\boldsymbol{W}^f$分別是忘記閘的偏置、輸入權重和遞歸權重。

因此，LSTM單元的內部狀態如下更新，但具有條件自循環權重 $f_i^{(t)}$:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right), \tag{10.41}$$

其中$\boldsymbol{b}$, $\boldsymbol{U}$ 和 $\boldsymbol{W}$ 分別代表進入LSTM cell的偏置、輸入權重和遞歸權重。

外部輸入閘單元 $g_i^{(t)}$ 的計算類似於忘記閘(使用sigmoid單元獲得0到1之間的閘控值)，但具有其自己的參數:

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right). \tag{10.42}$$

LSTM單元的輸出$h_i^{(t)}$ 也可以通過輸出閘$q_i^{(t)}$ 關閉，該閘也使用sigmoid單元進行閘控:

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}, \tag{10.43}$$

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right), \tag{10.44}$$

其參數$\boldsymbol{b}^O$, $\boldsymbol{U}^O$, $\boldsymbol{W}^O$分別用於其偏置、輸入權重和遞歸權重。

Other Gated RNNs

LSTM架構的哪些部分實際上是必要的？可以設計哪些其他成功的架構，以允許網絡動態控制不同單元的時間尺度和遺忘行為？

在閘控遞歸單元（GRU）中，單個閘控單元同時控制遺忘因素和更新狀態單元的決定。更新方程式如下：

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)})\sigma\left(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)}\right), \tag{10.45}$$

其中$\boldsymbol{u}$代表“更新”閘，$\boldsymbol{r}$代表“重置”閘。

更新閘和重置閘的值定義為:

$$u_i^{(t)} = \sigma\left(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)}\right) \tag{10.46}$$

$$r_i^{(t)} = \sigma\left(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)}\right). \tag{10.47}$$

重置閘和更新閘可以分別“忽略”狀態向量的部分。更新閘像條件滲漏整合器一樣，可以線性地閘控任何維度，從而選擇將其復制或完全忽略，並將其替換為新的“目標狀態”值。重置閘控制用於計算下一個目標狀態的狀態部分，從而在過去狀態和未來狀態之間產生額外的非線性效應。

可以圍繞這個主題設計更多變體。例如，重置閘（或忘記閘）的輸出可以在多個隱藏單元中共享。或者，全局閘（涵蓋整個單元組，如整個層）和局部閘（每個單元）的乘積可以用於結合全局控制和局部控制。

關於LSTM和GRU的架構變化進行了多次調查，但沒有找到明顯超過這兩者的變體，適用於廣泛的任務範圍。一個關鍵成分是忘記閘，發現在LSTM忘記閘中添加偏置1可以使LSTM與探索的最佳架構變體一樣強大。

第六章:RNN的結構調整和優化

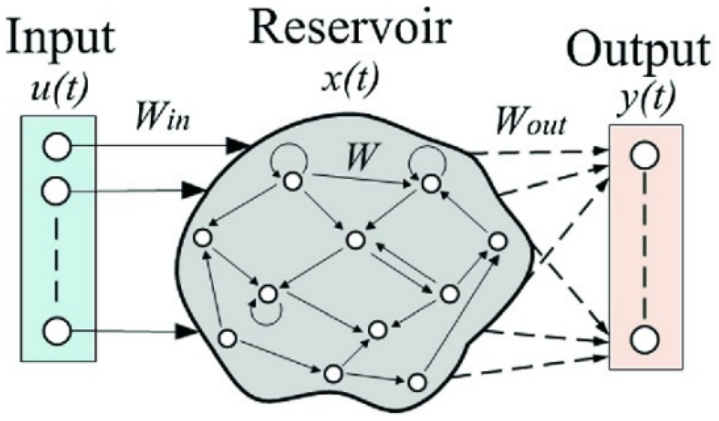
Removing Connections

處理長期依賴的另一種方法是組織循環神經網絡(RNN)在多個時間尺度上的狀態。這個想法涉及主動移除長度為1的連接，並用更長的連接替換它們。接收這種新連接的單元可能學會在長時間尺度上運作，但也可能選擇專注於其他的短期連接。
強迫一組循環單元在不同時間尺度上運作有不同的方法。一種選擇是使循環單元具有漏失性，但讓不同的單元組與不同的固定時間尺度相關聯。另一種選擇是在不同時間進行明確和离散的更新，不同組的單元具有不同的頻率。

Echo State Networks(回聲狀態網絡)

在人工智能和機器學習的領域中，回聲狀態網絡(ESN)提供了一種獨特的方法來處理循環神經網絡(RNN)的學習困難。ESN的核心思想是固定循環權重，從而使網絡能夠有效地捕捉輸入數據的歷史信息，同時僅學習輸出權重。這種方法與液態機器有著相似的理念，但ESN使用continuous-valued hidden units，而液態機器則使用尖峰神經元(spiking neurons (with binary outputs))。

ESN的工作原理
ESN的工作方式類似於核機器。它們將任意長度的序列(the history of inputs up to time t)映射到固定長度的向量(循環狀態$\boldsymbol{h}^{(t)}$)，這個狀態向量隨後可以用於線性預測。這種映射使得訓練過程變得相對簡單，因為輸出權重的學習可以通過凸優化方法有效進行。



設計**ESN**的關鍵
設計ESN的一個關鍵問題是如何設置輸入和循環權重，以便在網絡狀態中有效地表示輸入數據的歷史。這通常通過將循環網絡視為一個動態系統來實現，並調整權重以使系統維持在穩定的邊緣。具體來說，這涉及到控制雅可比矩陣的特徵值，以保持系統的動態穩定性。
使狀態到狀態轉換函數的雅可比矩陣的特徵值接近於1。循環網絡的一個重要特性是雅可比矩陣J(t)的特徵值譜。

$$J^{(t)} = \frac{\partial s^{(t)}}{\partial s^{(t-1)}}.$$

特別重要的是J(t)的譜半徑，定義為其特徵值絕對值的最大值。

回聲狀態網絡的策略簡單地是固定權重以具有某種譜半徑，例如3，信息通過時間傳遞，但由於飽和非線性（如tanh）的穩定作用而不會爆炸。