

第十章 序列建模：循环和递归网络

循环神经网络 (recurrent neural network) 或 RNN (Rumelhart *et al.*, 1986c) 是一类用于处理序列数据的神经网络。就像卷积网络是专门用于处理网格化数据 \mathbf{X} (如一个图像) 的神经网络, 循环神经网络是专门用于处理序列 $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}$ 的神经网络。正如卷积网络可以很容易地扩展到具有很大宽度和高度的图像, 以及处理大小可变的图像, 循环网络可以扩展到更长的序列 (比不基于序列的特化网络长得多)。大多数循环网络也能处理可变长度的序列。

从多层网络出发到循环网络, 我们需要利用上世纪 80 年代机器学习和统计模型早期思想的优点: 在模型的不同部分共享参数。参数共享使得模型能够扩展到不同形式的样本 (这里指不同长度的样本) 并进行泛化。如果我们在每个时间点都有一个单独的参数, 我们不但不能泛化到训练时没有见过序列长度, 也不能在时间上共享不同序列长度和不同位置的统计强度。当信息的特定部分会在序列内多个位置出现时, 这样的共享尤为重要。例如, 考虑这两句话: “I went to Nepal in 2009” 和 “In 2009, I went to Nepal.” 如果我们让一个机器学习模型读取这两个句子, 并提取叙述者去Nepal的年份, 无论 “2009 年” 是作为句子的第六个单词还是第二个单词出现, 我们都希望模型能认出 “2009 年” 作为相关资料片段。假设我们要训练一个处理固定长度句子的前馈网络。传统的全连接前馈网络会给每个输入特征分配一个单独的参数, 所以需要分别学习句子每个位置的所有语言规则。相比之下, 循环神经网络在几个时间步内共享相同的权重, 不需要分别学习句子每个位置的所有语言规则。

一个相关的想法是在 1 维时间序列上使用卷积。这种卷积方法是时延神经网络的基础 (Lang and Hinton, 1988; Waibel *et al.*, 1989; Lang *et al.*, 1990)。卷积操作允许网络跨时间共享参数, 但是浅层的。卷积的输出是一个序列, 其中输出中的每一项是相邻几项输入的函数。参数共享的概念体现在每个时间步中使用的相同卷积核。循环神经网络以不同的方式共享参数。输出的每一项是前一项的函数。输出的

每一项对先前的输出应用相同的更新规则而产生。这种循环方式导致参数通过很深的计算图共享。

为简单起见，我们说的 RNN 是指在序列上的操作，并且该序列在时刻 t （从 1 到 τ ）包含向量 $\mathbf{x}^{(t)}$ 。在实际情况中，循环网络通常在序列的小批量上操作，并且小批量的每项具有不同序列长度 τ 。我们省略了小批量索引来简化记号。此外，时间步索引不必是字面上现实世界中流逝的时间。有时，它仅表示序列中的位置。RNN 也可以应用于跨越两个维度的空间数据（如图像）。当应用于涉及时间的数据，并且将整个序列提供给网络之前就能观察到整个序列时，该网络可具有关于时间向后的连接。

本章将计算图的思想扩展到包括循环。这些周期代表变量自身的值在未来某一时间步对自身值的影响。这样的计算图允许我们定义循环神经网络。然后，我们描述许多构建、训练和使用循环神经网络的不同方式。

本章将简要介绍循环神经网络，为获取更多详细信息，我们建议读者参考 Graves (2012) 的著作。

10.1 展开计算图

计算图是形式化一组计算结构的方式，如那些涉及将输入和参数映射到输出和损失的计算。综合的介绍请参考第 6.5.1 节。本节，我们对展开（unfolding）递归或循环计算得到的重复结构进行解释，这些重复结构通常对应于一个事件链。展开（unfolding）这个计算图将导致深度网络结构中的参数共享。

例如，考虑动态系统的经典形式：

$$\mathbf{s}^{(t)} = f(\mathbf{s}^{(t-1)}; \boldsymbol{\theta}), \quad (10.1)$$

其中 $\mathbf{s}^{(t)}$ 称为系统的状态。

\mathbf{s} 在时刻 t 的定义需要参考时刻 $t - 1$ 时同样的定义，因此式 (10.1) 是循环的。

对有限时间步 τ , $\tau - 1$ 次应用这个定义可以展开这个图。例如 $\tau = 3$ ，我们对式 (10.1) 展开，可以得到：

$$\mathbf{s}^{(3)} = f(\mathbf{s}^{(2)}; \boldsymbol{\theta}) \quad (10.2)$$

$$= f(f(\mathbf{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta}). \quad (10.3)$$

以这种方式重复应用定义，展开等式，就能得到不涉及循环的表达。现在我们可以使用传统的有向无环计算图呈现这样的表达。

式(10.1)和式(10.3)的展开计算图如图10.1所示。

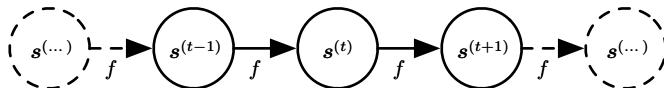


图10.1：将式(10.1)描述的经典动态系统表示为展开的计算图。每个节点表示在某个时刻 t 的状态，并且函数 f 将 t 处的状态映射到 $t+1$ 处的状态。所有时间步都使用相同的参数（用于参数化 f 的相同 θ 值）。

作为另一个例子，让我们考虑由外部信号 $\mathbf{x}^{(t)}$ 驱动的动态系统，

$$\mathbf{s}^{(t)} = f(\mathbf{s}^{(t-1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta}), \quad (10.4)$$

我们可以看到，当前状态包含了整个过去序列的信息。

循环神经网络可以通过许多不同的方式建立。就像几乎所有函数都可以被认为是前馈网络，本质上任何涉及循环的函数都可以被认为是一个循环神经网络。

很多循环神经网络使用式(10.5)或类似的公式定义隐藏单元的值。为了表明状态是网络的隐藏单元，我们使用变量 \mathbf{h} 代表状态重写式(10.4)：

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta}), \quad (10.5)$$

如图10.2所示，典型RNN会增加额外的架构特性，如读取状态信息 \mathbf{h} 进行预测的输出层。

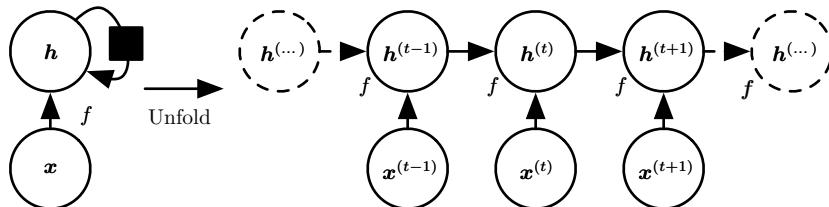


图10.2：没有输出的循环网络。此循环网络只处理来自输入 \mathbf{x} 的信息，将其合并到经过时间向前传播的状态 \mathbf{h} 。(左)回路原理图。黑色方块表示单个时间步的延迟。(右)同一网络被视为展开的计算图，其中每个节点现在与一个特定的时间实例相关联。

当训练循环网络根据过去预测未来时，网络通常要学会使用 $\mathbf{h}^{(t)}$ 作为过去序列（直到 t ）与任务相关方面的有损摘要。此摘要一般而言一定是有损的，因为其映射任意长度的序列 $(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \mathbf{x}^{(t-2)}, \dots, \mathbf{x}^{(2)}, \mathbf{x}^{(1)})$ 到一固定长度的向量 $\mathbf{h}^{(t)}$ 。根据不同的训练准则，摘要可能选择性地精确保留过去序列的某些方面。例如，如果在统计语言建模中使用的 RNN，通常给定前一个词预测下一个词，可能没有必要存储时刻 t 前输入序列中的所有信息；而仅仅存储足够预测句子其余部分的信息。最苛刻的情况是我们要求 $\mathbf{h}^{(t)}$ 足够丰富，并能大致恢复输入序列，如自编码器框架（第十四章）。

式(10.5)可以用两种不同的方式绘制。一种方法是为可能在模型的物理实现中存在的部分赋予一个节点，如生物神经网络。在这个观点下，网络定义了实时操作的回路，如图 10.2 的左侧，其当前状态可以影响其未来的状态。在本章中，我们使用回路图的黑色方块表明在时刻 t 的状态到时刻 $t+1$ 的状态单个时刻延迟中的相互作用。另一个绘制 RNN 的方法是展开的计算图，其中每一个组件由许多不同的变量表示，每个时间步一个变量，表示在该时间点组件的状态。每个时间步的每个变量绘制为计算图的一个独立节点，如图 10.2 的右侧。我们所说的展开是将左图中的回路映射为右图中包含重复组件的计算图的操作。目前，展开图的大小取决于序列长度。

我们可以用一个函数 $g^{(t)}$ 代表经 t 步展开后的循环：

$$\mathbf{h}^{(t)} = g^{(t)}(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \mathbf{x}^{(t-2)}, \dots, \mathbf{x}^{(2)}, \mathbf{x}^{(1)}) \quad (10.6)$$

$$= f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta). \quad (10.7)$$

函数 $g^{(t)}$ 将全部的过去序列 $(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \mathbf{x}^{(t-2)}, \dots, \mathbf{x}^{(2)}, \mathbf{x}^{(1)})$ 作为输入来生成当前状态，但是展开的循环架构允许我们将 $g^{(t)}$ 分解为函数 f 的重复应用。因此，展开过程引入两个主要优点：

1. 无论序列的长度，学成的模型始终具有相同的输入大小，因为它指定的是从一种状态到另一种状态的转移，而不是在可变长度的历史状态上操作。
2. 我们可以在每个时间步使用相同参数的相同转移函数 f 。

这两个因素使得学习在所有时间步和所有序列长度上操作单一的模型 f 是可能的，而不需要在所有可能时间步学习独立的模型 $g^{(t)}$ 。学习单一的共享模型允许泛化到没有见过的序列长度（没有出现在训练集中），并且估计模型所需的训练样本远远少于不带参数共享的模型。

无论是循环图和展开图都有其用途。循环图简洁。展开图能够明确描述其中的计算流程。展开图还通过显式的信息流动路径帮助说明信息在时间上向前（计算输出和损失）和向后（计算梯度）的思想。

10.2 循环神经网络

基于第 10.1 节中的图展开和参数共享的思想，我们可以设计各种循环神经网络。

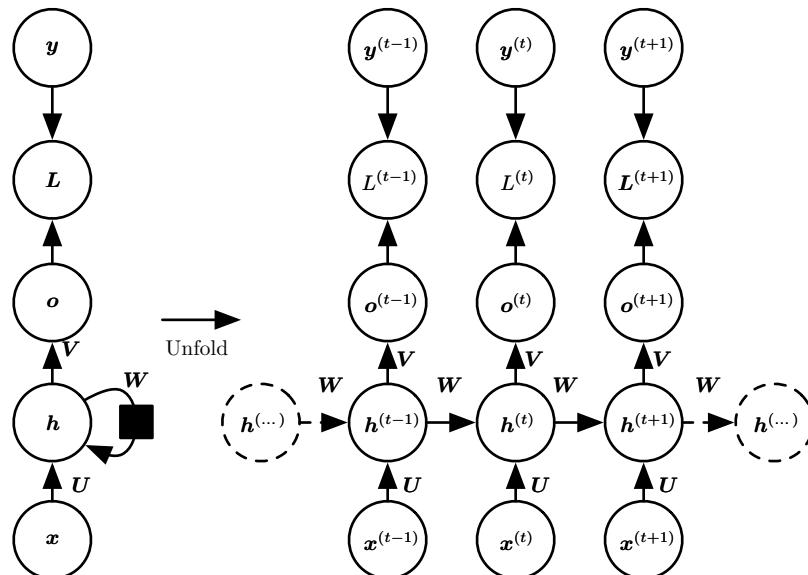


图 10.3: 计算循环网络(将 x 值的输入序列映射到输出值 o 的对应序列)训练损失的计算图。损失 L 衡量每个 o 与相应的训练目标 y 的距离。当使用 softmax 输出时，我们假设 o 是未归一化的对数概率。损失 L 内部计算 $\hat{y} = \text{softmax}(o)$ ，并将其与目标 y 比较。RNN 输入到隐藏的连接由权重矩阵 U 参数化，隐藏到隐藏的循环连接由权重矩阵 W 参数化以及隐藏到输出的连接由权重矩阵 V 参数化。式(10.8)定义了该模型中的前向传播。(左) 使用循环连接绘制的 RNN 和它的损失。(右) 同一网络被视为展开的计算图，其中每个节点现在与一个特定的时间实例相关联。

循环神经网络中一些重要的设计模式包括以下几种：

1. 每个时间步都有输出，并且隐藏单元之间有循环连接的循环网络，如图 10.3 所示。

2. 每个时间步都产生一个输出，只有当前时刻的输出到下个时刻的隐藏单元之间有循环连接的循环网络，如图 10.4 所示。
3. 隐藏单元之间存在循环连接，但读取整个序列后产生单个输出的循环网络，如图 10.5 所示。

图 10.3 是非常具有代表性的例子，我们将会在本章大部分涉及这个例子。

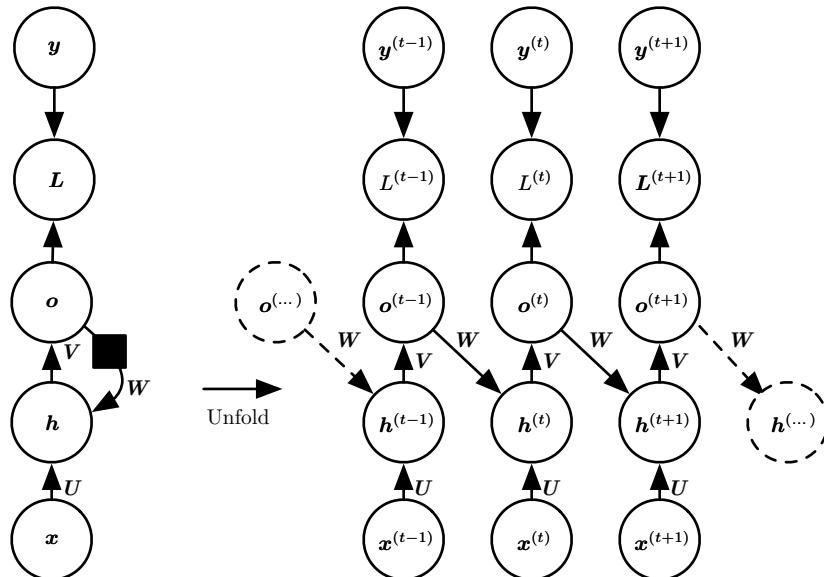


图 10.4: 此类 RNN 的唯一循环是从输出到隐藏层的反馈连接。在每个时间步 t ，输入为 \mathbf{x}_t ，隐藏层激活为 $\mathbf{h}^{(t)}$ ，输出为 $\mathbf{o}^{(t)}$ ，目标为 $\mathbf{y}^{(t)}$ ，损失为 $L^{(t)}$ 。(左)回路原理图。(右)展开的计算图。这样的 RNN 没有图 10.3 表示的 RNN 那样强大（只能表示更小的函数集合）。图 10.3 中的 RNN 可以选择将其想要的关于过去的任何信息放入隐藏表示 \mathbf{h} 中并且将 \mathbf{h} 传播到未来。该图中的 RNN 被训练为将特定输出值放入 \mathbf{o} 中，并且 \mathbf{o} 是允许传播到未来的唯一信息。此处没有从 \mathbf{h} 前向传播的直接连接。之前的 \mathbf{h} 仅通过产生的预测间接地连接到当前。 \mathbf{o} 通常缺乏过去的重要信息，除非它非常高维且内容丰富。这使得该图中的 RNN 不那么强大，但是它更容易训练，因为每个时间步可以与其他时间步分离训练，允许训练期间更多的并行化，如第 10.2.1 节所述。

任何图灵可计算的函数都可以通过这样一个有限维的循环网络计算，在这个意义上图 10.3 和式 (10.8) 的循环神经网络是万能的。RNN 经过若干时间步后读取输出，这与由图灵机所用的时间步是渐近线性的，与输入长度也是渐近线性的 (Siegelmann and Sontag, 1991; Siegelmann, 1995; Siegelmann and Sontag, 1995;

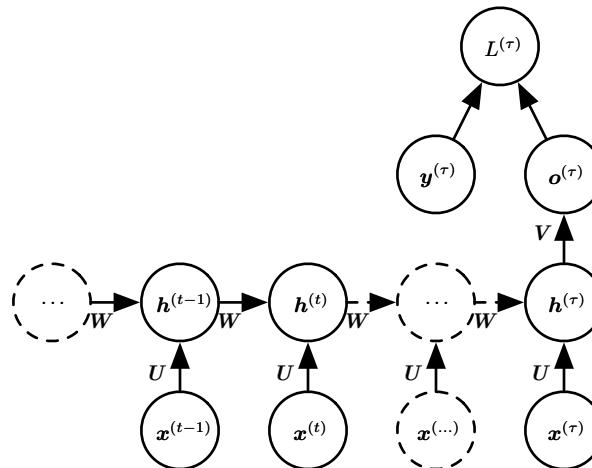


图 10.5: 关于时间展开的循环神经网络，在序列结束时具有单个输出。这样的网络可以用于概括序列并产生用于进一步处理的固定大小的表示。在结束处可能存在目标（如此处所示），或者通过更下游模块的反向传播来获得输出 $\mathbf{o}^{(t)}$ 上的梯度。

Hyötyniemi, 1996)。由图灵机计算的函数是离散的，所以这些结果都是函数的具体实现，而不是近似。RNN 作为图灵机使用时，需要一个二进制序列作为输入，其输出必须离散化以提供二进制输出。利用单个有限大小的特定 RNN 计算在此设置下的所有函数是可能的（Siegelmann and Sontag (1995) 用了 886 个单元）。图灵机的“输入”是要计算函数的详细说明 (specification)，所以模拟此图灵机的相同网络足以应付所有问题。用于证明的理论 RNN 可以通过激活和权重（由无限精度的有理数表示）来模拟无限堆栈。

现在我们研究图 10.3 中 RNN 的前向传播公式。这个图没有指定隐藏单元的激活函数。我们假设使用双曲正切激活函数。此外，图中没有明确指定何种形式的输出和损失函数。我们假定输出是离散的，如用于预测词或字符的 RNN。表示离散变量的常规方式是把输出 \mathbf{o} 作为每个离散变量可能值的非标准化对数概率。然后，我们可以应用 softmax 函数后续处理后，获得标准化后概率的输出向量 $\hat{\mathbf{y}}$ 。RNN 从特定的初始状态 $\mathbf{h}^{(0)}$ 开始前向传播。从 $t = 1$ 到 $t = \tau$ 的每个时间步，我们应用以下

更新方程：

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}, \quad (10.8)$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}), \quad (10.9)$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}, \quad (10.10)$$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)}), \quad (10.11)$$

其中的参数的偏置向量 \mathbf{b} 和 \mathbf{c} 连同权重矩阵 \mathbf{U} 、 \mathbf{V} 和 \mathbf{W} ，分别对应于输入到隐藏、隐藏到输出和隐藏到隐藏的连接。这个循环网络将一个输入序列映射到相同长度的输出序列。与 \mathbf{x} 序列配对的 \mathbf{y} 的总损失就是所有时间步的损失之和。例如， $L^{(t)}$ 为给定的 $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}$ 后 $\mathbf{y}^{(t)}$ 的负对数似然，则

$$L(\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)}\}) \quad (10.12)$$

$$= \sum_t L^{(t)} \quad (10.13)$$

$$= - \sum_t \log p_{\text{model}}(\mathbf{y}^{(t)} \mid \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}\}), \quad (10.14)$$

其中 $p_{\text{model}}(\mathbf{y}^{(t)} \mid \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}\})$ 需要读取模型输出向量 $\hat{\mathbf{y}}^{(t)}$ 中对应于 $y^{(t)}$ 的项。关于各个参数计算这个损失函数的梯度是计算成本很高的操作。梯度计算涉及执行一次前向传播（如在图 10.3 展开图中从左到右的传播），接着是由右到左的反向传播。运行时间是 $\mathcal{O}(\tau)$ ，并且不能通过并行化来降低，因为前向传播图是固有循序的；每个时间步只能一前一后地计算。前向传播中的各个状态必须保存，直到它们反向传播中被再次使用，因此内存代价也是 $\mathcal{O}(\tau)$ 。应用于展开图且代价为 $\mathcal{O}(\tau)$ 的反向传播算法称为 **通过时间反向传播** (back-propagation through time, BPTT)，将在第 10.2.2 节进一步讨论。因此隐藏单元之间存在循环的网络非常强大但训练代价也很大。我们是否有其他选择呢？

10.2.1 导师驱动过程和输出循环网络

仅在一个时间步的输出和下一个时间步的隐藏单元间存在循环连接的网络（示于图 10.4）确实没有那么强大（因为缺乏隐藏到隐藏的循环连接）。例如，它不能模拟通用图灵机。因为这个网络缺少隐藏到隐藏的循环，它要求输出单元捕捉用于预测未来的关于过去的所有信息。因为输出单元明确地训练成匹配训练集的目标，它们不太能捕获关于过去输入历史的必要信息，除非用户知道如何描述系统的全部状

态，并将它作为训练目标的一部分。消除隐藏到隐藏循环的优点在于，任何基于比较时刻 t 的预测和时刻 t 的训练目标的损失函数中的所有时间步都解耦了。因此训练可以并行化，即在各时刻 t 分别计算梯度。因为训练集提供输出的理想值，所以没有必要先计算前一时刻的输出。

由输出反馈到模型而产生循环连接的模型可用 **导师驱动过程** (teacher forcing) 进行训练。训练模型时，导师驱动过程不再使用最大似然准则，而在时刻 $t + 1$ 接收真实值 $y^{(t)}$ 作为输入。我们可以通过检查两个时间步的序列得知这一点。条件最大似然准则是

$$\log p(\mathbf{y}^{(1)}, \mathbf{y}^{(2)} \mid \mathbf{x}^{(1)}, \mathbf{x}^{(2)}) \quad (10.15)$$

$$= \log p(\mathbf{y}^{(2)} \mid \mathbf{y}^{(1)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}) + \log p(\mathbf{y}^{(1)} \mid \mathbf{x}^{(1)}, \mathbf{x}^{(2)}). \quad (10.16)$$

在这个例子中，同时给定迄今为止的 \mathbf{x} 序列和来自训练集的前一 \mathbf{y} 值，我们可以看到在时刻 $t = 2$ 时，模型被训练为最大化 $\mathbf{y}^{(2)}$ 的条件概率。因此最大似然在训练时指定正确反馈，而不是将自己的输出反馈到模型。如图 10.6 所示。

我们使用导师驱动过程的最初动机是为了在缺乏隐藏到隐藏连接的模型中避免通过时间反向传播。只要模型一个时间步的输出与下一时间步计算的值存在连接，导师驱动过程仍然可以应用到这些存在隐藏到隐藏连接的模型。然而，只要隐藏单元成为较早时间步的函数，BPTT 算法是必要的。因此训练某些模型时要同时使用导师驱动过程和 BPTT。

如果之后网络在开环 (open-loop) 模式下使用，即网络输出（或输出分布的样本）反馈作为输入，那么完全使用导师驱动过程进行训练的缺点就会出现。在这种情况下，训练期间该网络看到的输入与测试时看到的会有很大的不同。减轻此问题的一种方法是同时使用导师驱动过程和自由运行的输入进行训练，例如在展开循环的输出到输入路径上预测几个步骤的正确目标值。通过这种方式，网络可以学会考虑在训练时没有接触到的输入条件（如自由运行模式下，自身生成自身），以及将状态映射回使网络几步之后生成正确输出的状态。另外一种方式 (Bengio *et al.*, 2015b) 是通过随意选择生成值或真实的数据值作为输入以减小训练时和测试时看到的输入之间的差别。这种方法利用了课程学习策略，逐步使用更多生成值作为输入。

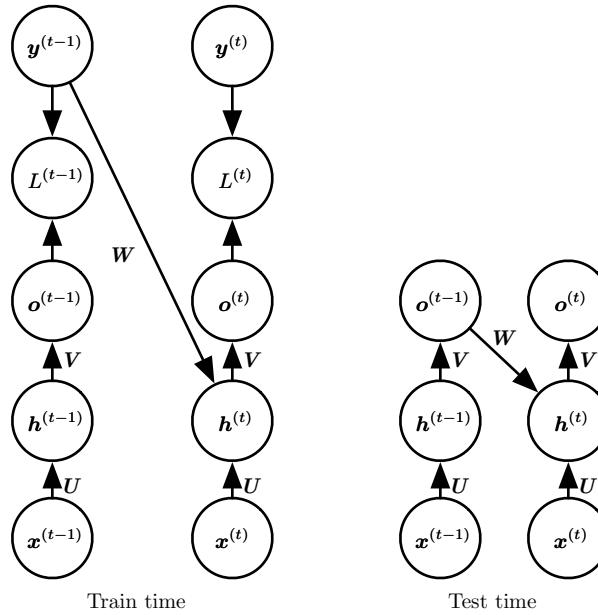


图 10.6: 导师驱动过程的示意图。导师驱动过程是一种训练技术，适用于输出与下一时间步的隐藏状态存在连接的 RNN。(左) 训练时，我们将训练集中正确的输出 $y^{(t)}$ 反馈到 $h^{(t+1)}$ 。(右) 当模型部署后，真正的输出通常是未知的。在这种情况下，我们用模型的输出 $o^{(t)}$ 近似正确的输出 $y^{(t)}$ ，并反馈回模型。

10.2.2 计算循环神经网络的梯度

计算循环神经网络的梯度是容易的。我们可以简单地将第 6.5.6 节中的推广反向传播算法应用于展开的计算图，而不需要特殊化的算法。由反向传播计算得到的梯度，并结合任何通用的基于梯度的技术就可以训练 RNN。

为了获得 BPTT 算法行为的一些直观理解，我们举例说明如何通过 BPTT 计算上述 RNN 公式（式 (10.8) 和式 (10.12)）的梯度。计算图的节点包括参数 U, V, W, b 和 c ，以及以 t 为索引的节点序列 $x^{(t)}, h^{(t)}, o^{(t)}$ 和 $L^{(t)}$ 。对于每一个节点 \mathbf{N} ，我们需要基于 \mathbf{N} 后面的节点的梯度，递归地计算梯度 $\nabla_{\mathbf{N}} L$ 。我们从紧接着最终损失的节点开始递归：

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

在这个导数中，我们假设输出 $o^{(t)}$ 作为 softmax 函数的参数，我们可以从 softmax

函数可以获得关于输出概率的向量 $\hat{\mathbf{y}}$ 。我们也假设损失是迄今为止给定了输入后的真实目标 $y^{(t)}$ 的负对数似然。对于所有 i, t , 关于时间步 t 输出的梯度 $\nabla_{o^{(t)}} L$ 如下：

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

我们从序列的末尾开始, 反向进行计算。在最后的时间步 τ , $\mathbf{h}^{(\tau)}$ 只有 $\mathbf{o}^{(\tau)}$ 作为后续节点, 因此这个梯度很简单:

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

然后, 我们可以从时刻 $t = \tau - 1$ 到 $t = 1$ 反向迭代, 通过时间反向传播梯度, 注意 $\mathbf{h}^{(t)} (t < \tau)$ 同时具有 $\mathbf{o}^{(t)}$ 和 $\mathbf{h}^{(t+1)}$ 两个后续节点。因此, 它的梯度由下式计算

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

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

其中 $\text{diag}\left(1 - (\mathbf{h}^{(t+1)})^2\right)$ 表示包含元素 $1 - (h_i^{(t+1)})^2$ 的对角矩阵。这是关于时刻 $t+1$ 与隐藏单元 i 关联的双曲正切的Jacobian。

一旦获得了计算图内部节点的梯度, 我们就可以得到关于参数节点的梯度。因为参数在许多时间步共享, 我们必须在表示这些变量的微积分操作时谨慎对待。我们希望实现的等式使用第 6.5.6 节中的 `bprop` 方法计算计算图中单一边对梯度的贡献。然而微积分中的 $\nabla_{\mathbf{w}f}$ 算子, 计算 \mathbf{W} 对于 f 的贡献时将计算图中的所有边都考虑进去了。为了消除这种歧义, 我们定义只在 t 时刻使用的虚拟变量 $\mathbf{W}^{(t)}$ 作为 \mathbf{W} 的副本。然后, 我们可以使用 $\nabla_{\mathbf{W}^{(t)}}$ 表示权重在时间步 t 对梯度的贡献。

使用这个表示，关于剩下参数的梯度可以由下式给出：

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

$$\nabla_b L = \sum_t \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}^{(t)}} \right)^\top \nabla_{\mathbf{h}^{(t)}} L = \sum_t \text{diag}\left(1 - (\mathbf{h}^{(t)})^2\right) \nabla_{\mathbf{h}^{(t)}} L, \quad (10.23)$$

$$\nabla_v L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}} \right) \nabla_{v o_i^{(t)}} = \sum_t (\nabla_{\mathbf{o}^{(t)}} L) \mathbf{h}^{(t)\top}, \quad (10.24)$$

$$\nabla_w L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{w^{(t)}} h_i^{(t)} \quad (10.25)$$

$$= \sum_t \text{diag}\left(1 - (\mathbf{h}^{(t)})^2\right) (\nabla_{\mathbf{h}^{(t)}} L) \mathbf{h}^{(t-1)\top}, \quad (10.26)$$

$$\nabla_u L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{u^{(t)}} h_i^{(t)} \quad (10.27)$$

$$= \sum_t \text{diag}\left(1 - (\mathbf{h}^{(t)})^2\right) (\nabla_{\mathbf{h}^{(t)}} L) \mathbf{x}^{(t)\top}, \quad (10.28)$$

因为计算图中定义的损失的任何参数都不是训练数据 $\mathbf{x}^{(t)}$ 的父节点，所以我们不需要计算关于它的梯度。

10.2.3 作为有向图模型的循环网络

目前为止，我们接触的循环网络例子中损失 $L^{(t)}$ 是训练目标 $\mathbf{y}^{(t)}$ 和输出 $\mathbf{o}^{(t)}$ 之间的交叉熵。与前馈网络类似，原则上循环网络几乎可以使用任何损失。但必须根据任务来选择损失。如前馈网络，我们通常希望将 RNN 的输出解释为一个概率分布，并且我们通常使用与分布相关联的交叉熵来定义损失。均方误差是与单位高斯分布的输出相关联的交叉熵损失，例如前馈网络中所使用的。

当我们使用一个预测性对数似然的训练目标，如式 (10.12)，我们将 RNN 训练为能够根据之前的输入估计下一个序列元素 $\mathbf{y}^{(t)}$ 的条件分布。这可能意味着，我们最大化对数似然

$$\log p(\mathbf{y}^{(t)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}), \quad (10.29)$$

或者，如果模型包括来自一个时间步的输出到下一个时间步的连接，

$$\log p(\mathbf{y}^{(t)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}, \mathbf{y}^{(1)}, \dots, \mathbf{y}^{(t-1)}). \quad (10.30)$$

将整个序列 \mathbf{y} 的联合分布分解为一系列单步的概率预测是捕获关于整个序列完整联合分布的一种方法。当我们不把过去的 \mathbf{y} 值反馈给下一步作为预测的条件时，那么有向图模型不包含任何从过去 $\mathbf{y}^{(i)}$ 到当前 $\mathbf{y}^{(t)}$ 的边。在这种情况下，输出 \mathbf{y} 与给定的 \mathbf{x} 序列是条件独立的。当我们反馈真实的 \mathbf{y} 值（不是它们的预测值，而是真正观测到或生成的值）给网络时，那么有向图模型包含所有从过去 $\mathbf{y}^{(i)}$ 到当前 $\mathbf{y}^{(t)}$ 的边。

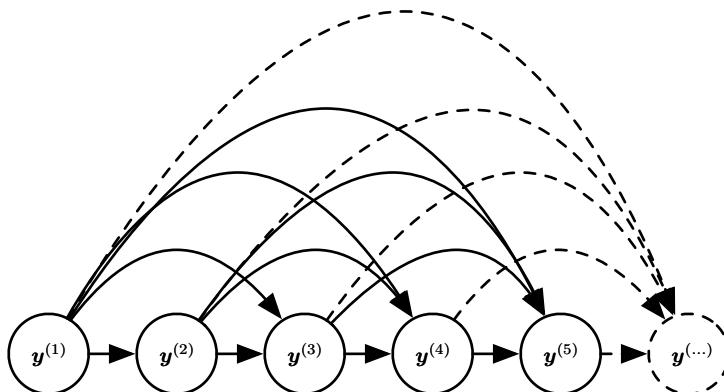


图 10.7: 序列 $y^{(1)}, y^{(2)}, \dots, y^{(t)}, \dots$ 的全连接图模型。给定先前的值，每个过去的观察值 $y^{(i)}$ 可以影响一些 $y^{(t)} (t > i)$ 的条件分布。当序列中每个元素的输入和参数的数目越来越多，根据此图直接参数化图模型（如式 (10.6) 中）可能非常低效的。RNN 可以通过高效的参数化获得相同的全连接，如图 10.8 所示。

举一个简单的例子，让我们考虑对标量随机变量序列 $\mathbb{Y} = \{y^{(1)}, \dots, y^{(\tau)}\}$ 建模的 RNN，也没有额外的输入 \mathbf{x} 。在时间步 t 的输入仅仅是时间步 $t - 1$ 的输出。该 RNN 定义了关于 \mathbf{y} 变量的有向图模型。我们使用链式法则（用于条件概率的式 (3.6)）参数化这些观察值的联合分布：

$$P(\mathbb{Y}) = P(y^{(1)}, \dots, y^{(\tau)}) = \prod_{t=1}^{\tau} P(y^{(t)} | y^{(t-1)}, y^{(t-2)}, \dots, y^{(1)}), \quad (10.31)$$

其中当 $t = 1$ 时竖杠右侧显然为空。因此，根据这样一个模型，一组值 $\{y^{(1)}, \dots, y^{(\tau)}\}$ 的负对数似然为

$$L = \sum_t L^{(t)}, \quad (10.32)$$

其中

$$L^{(t)} = -\log P(y^{(t)} = y^{(t)} \mid y^{(t-1)}, y^{(t-2)}, \dots, y^{(1)}). \quad (10.33)$$

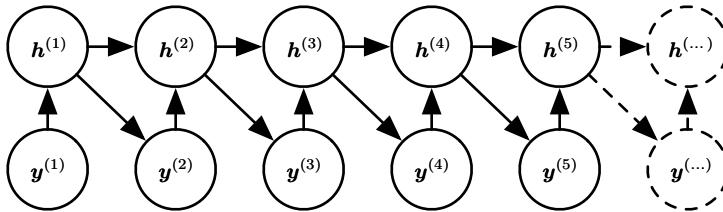


图 10.8: 在 RNN 图模型中引入状态变量, 尽管它是输入的确定性函数, 但它有助于我们根据式 (10.5) 获得非常高效的参数化。序列中的每个阶段 (对于 $h^{(t)}$ 和 $y^{(t)}$) 使用相同的结构 (每个节点具有相同数量的输入), 并且可以与其他阶段共享相同的参数。

图模型中的边表示哪些变量直接依赖于其他变量。许多图模型的目标是省略不存在强相互作用的边以实现统计和计算的效率。例如, 我们通常可以作Markov假设, 即图模型应该只包含从 $\{y^{(t-k)}, \dots, y^{(t-1)}\}$ 到 $y^{(t)}$ 的边, 而不是包含整个过去历史的边。然而, 在一些情况下, 我们认为整个过去的输入会对序列的下一个元素有一定影响。当我们认为 $y^{(t)}$ 的分布可能取决于遥远过去 (在某种程度) 的 $y^{(i)}$ 的值, 且无法通过 $y^{(t-1)}$ 捕获 $y^{(i)}$ 的影响时, RNN 将会很有用。

解释 RNN 作为图模型的一种方法是将RNN视为定义一个结构为完全图的图模型, 且能够表示任何一对 y 值之间的直接联系。图 10.7 是关于 y 值且具有完全图结构的图模型。该 RNN 完全图的解释基于排除并忽略模型中的隐藏单元 $h^{(t)}$ 。

更有趣的是, 将隐藏单元 $h^{(t)}$ 视为随机变量, 从而产生 RNN 的图模型结构¹。在图模型中包括隐藏单元预示 RNN 能对观测的联合分布提供非常有效的参数化。假设我们用表格表示法来表示离散值上任意的联合分布, 即对每个值可能的赋值分配一个单独条目的数组, 该条目表示发生该赋值的概率。如果 y 可以取 k 个不同的值, 表格表示法将有 $\mathcal{O}(k^r)$ 个参数。对比 RNN, 由于参数共享, RNN 的参数数目为 $\mathcal{O}(1)$ 且是序列长度的函数。我们可以调节 RNN 的参数数量来控制模型容量, 但不用被迫与序列长度成比例。式 (10.5) 展示了所述 RNN 通过循环应用相同的函数 f 以及在每个时间步的相同参数 θ , 有效地参数化的变量之间的长期联系。图 10.8 说

¹ 给定这些变量的父变量, 其条件分布是确定性的。尽管设计具有这样确定性的隐藏单元的图模型是很少见的, 但这是完全合理的。

明了这个图模型的解释。在图模型中结合 $h^{(t)}$ 节点可以用作过去和未来之间的中间量，从而将它们解耦。遥远过去的变量 $y^{(i)}$ 可以通过其对 h 的影响来影响变量 $y^{(t)}$ 。该图的结构表明可以在时间步使用相同的条件概率分布有效地参数化模型，并且当观察到全部变量时，可以高效地评估联合分配给所有变量的概率。

即便使用高效参数化的图模型，某些操作在计算上仍然具有挑战性。例如，难以预测序列中缺少的值。

循环网络为减少的参数数目付出的代价是优化参数可能变得困难。

在循环网络中使用的参数共享的前提是相同参数可用于不同时间步的假设。也就是说，假设给定时刻 t 的变量后，时刻 $t + 1$ 变量的条件概率分布是 **平稳的** (stationary)，这意味着之前的时间步与下个时间步之间的关系并不依赖于 t 。原则上，可以使用 t 作为每个时间步的额外输入，并让学习器在发现任何时间依赖性的同时，在不同时间步之间尽可能多地共享。相比在每个 t 使用不同的条件概率分布已经好很多了，但网络将必须在面对新 t 时进行推断。

为了完整描述将 RNN 作为图模型的观点，我们必须描述如何从模型采样。我们需要执行的主要操作是简单地从每一时间步的条件分布采样。然而，这会导致额外的复杂性。RNN 必须有某种机制来确定序列的长度。这可以通过多种方式实现。

在当输出是从词汇表获取的符号的情况下，我们可以添加一个对应于序列末端的特殊符号 (Schmidhuber, 2012)。当产生该符号时，采样过程停止。在训练集中，我们将该符号作为序列的一个额外成员，即紧跟每个训练样本 $x^{(\tau)}$ 之后。

另一种选择是在模型中引入一个额外的 Bernoulli 输出，表示在每个时间步决定继续生成或停止生成。相比向词汇表增加一个额外符号，这种方法更普遍，因为它适用于任何 RNN，而不仅仅是输出符号序列的 RNN。例如，它可以应用于一个产生实数序列的 RNN。新的输出单元通常使用 sigmoid 单元，并通过交叉熵训练。在这种方法中，sigmoid 被训练为最大化正确预测的对数似然，即在每个时间步序列决定结束或继续。

确定序列长度 τ 的另一种方法是将一个额外的输出添加到模型并预测整数 τ 本身。模型可以采出 τ 的值，然后采 τ 步有价值的数据。这种方法需要在每个时间步的循环更新中增加一个额外输入，使得循环更新知道它是否是靠近所产生序列的末尾。这种额外的输入可以是 τ 的值，也可以是 $\tau - t$ 即剩下时间步的数量。如果没有这个额外的输入，RNN 可能会产生突然结束序列，如一个句子在最终完整前结

束。此方法基于分解

$$P(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}) = P(\tau)P(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)} | \tau). \quad (10.34)$$

直接预测 τ 的例子见 Goodfellow et al. (2014d)。

10.2.4 基于上下文的 RNN 序列建模

上一节描述了没有输入 \mathbf{x} 时，关于随机变量序列 $y^{(t)}$ 的 RNN 如何对应于有向图模型。当然，如式 (10.8) 所示的 RNN 包含一个输入序列 $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(\tau)}$ 。一般情况下，RNN 允许将图模型的观点扩展到不仅代表 y 变量的联合分布也能表示给定 \mathbf{x} 后 y 条件分布。如在第 6.2.1.1 节的前馈网络情形中所讨论的，任何代表变量 $P(\mathbf{y}; \boldsymbol{\theta})$ 的模型都能被解释为代表条件分布 $P(\mathbf{y} | \boldsymbol{\omega})$ 的模型，其中 $\boldsymbol{\omega} = \boldsymbol{\theta}$ 。我们能像之前一样使用 $P(\mathbf{y} | \boldsymbol{\omega})$ 代表分布 $P(\mathbf{y} | \mathbf{x})$ 来扩展这样的模型，但要令 $\boldsymbol{\omega}$ 是关于 \mathbf{x} 的函数。在 RNN 的情况，这可以通过不同的方式来实现。此处，我们回顾最常见和最明显的选择。

之前，我们已经讨论了将 $t = 1, \dots, \tau$ 的向量 $\mathbf{x}^{(t)}$ 序列作为输入的 RNN。另一种选择是只使用单个向量 \mathbf{x} 作为输入。当 \mathbf{x} 是一个固定大小的向量时，我们可以简单地将其看作产生 \mathbf{y} 序列 RNN 的额外输入。将额外输入提供到 RNN 的一些常见方法是：

1. 在每个时刻作为一个额外输入，或
2. 作为初始状态 $\mathbf{h}^{(0)}$ ，或
3. 结合两种方式。

第一个也是最常用的方法如图 10.9 所示。输入 \mathbf{x} 和每个隐藏单元向量 $\mathbf{h}^{(t)}$ 之间的相互作用是通过新引入的权重矩阵 \mathbf{R} 参数化的，这是只包含 y 序列的模型所没有的。同样的乘积 $\mathbf{x}^\top \mathbf{R}$ 在每个时间步作为隐藏单元的一个额外输入。我们可以认为 \mathbf{x} 的选择（确定 $\mathbf{x}^\top \mathbf{R}$ 值），是有效地用于每个隐藏单元的一个新偏置参数。权重与输入保持独立。我们可以认为这种模型采用了非条件模型的 $\boldsymbol{\theta}$ ，并将 $\boldsymbol{\omega}$ 代入 $\boldsymbol{\theta}$ ，其中 $\boldsymbol{\omega}$ 内的偏置参数现在是输入的函数。

RNN 可以接收向量序列 $\mathbf{x}^{(t)}$ 作为输入，而不是仅接收单个向量 \mathbf{x} 作为输入。式 (10.8) 描述的 RNN 对应条件分布 $P(\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)})$ ，并在条件独立

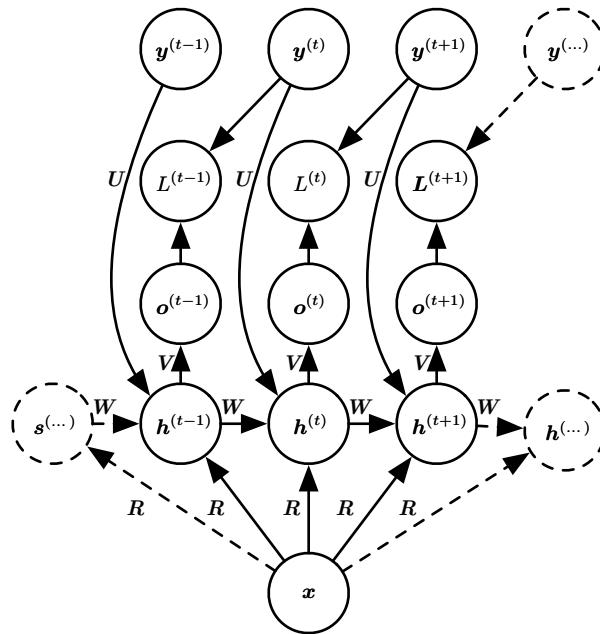


图 10.9: 将固定长度的向量 x 映射到序列 \mathbf{Y} 上分布的 RNN。这类 RNN 适用于很多任务如图注，其中单个图像作为模型的输入，然后产生描述图像的词序列。观察到的输出序列的每个元素 $y^{(t)}$ 同时用作输入（对于当前时间步）和训练期间的目标（对于前一时间步）。

的假设下这个分布分解为

$$\prod_t P(y^{(t)} \mid x^{(1)}, \dots, x^{(t)}). \quad (10.35)$$

为去掉条件独立的假设，我们可以在时刻 t 的输出到时刻 $t+1$ 的隐藏单元添加连接，如图 10.10 所示。该模型就可以代表关于 y 序列的任意概率分布。这种给定一个序列表示另一个序列分布的模型的还是有一个限制，就是这两个序列的长度必须是相同的。我们将在第 10.4 节描述如何消除这种限制。

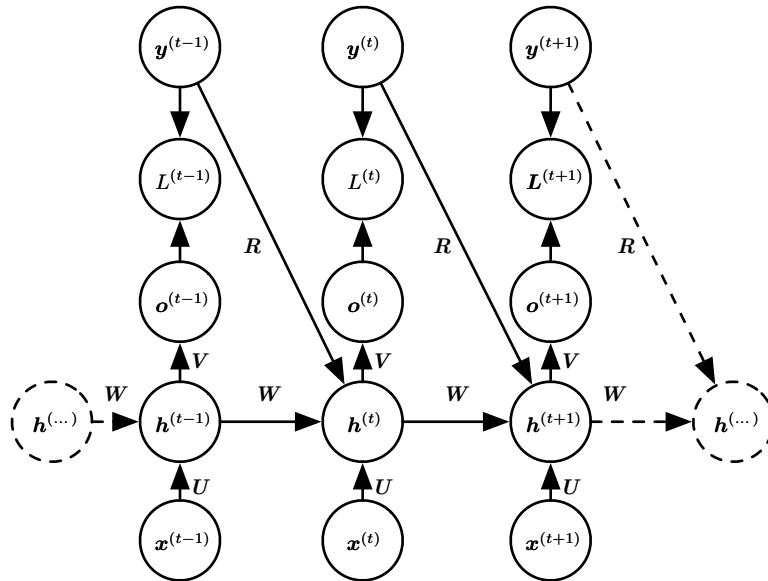


图 10.10: 将可变长度的 x 值序列映射到相同长度的 y 值序列上分布的条件循环神经网络。对比图 10.3, 此 RNN 包含从前一个输出到当前状态的连接。这些连接允许此 RNN 对给定 x 的序列后相同长度的 y 序列上的任意分布建模。图 10.3 的 RNN 仅能表示在给定 x 值的情况下, y 值彼此条件独立的分布。

10.3 双向 RNN

目前为止我们考虑的所有循环神经网络有一个“因果”结构，意味着在时刻 t 的状态只能从过去的序列 $x^{(1)}, \dots, x^{(t-1)}$ 以及当前的输入 $x^{(t)}$ 捕获信息。我们还讨论了某些在 y 可用时，允许过去的 y 值信息影响当前状态的模型。

然而，在许多应用中，我们要输出的 $y^{(t)}$ 的预测可能依赖于整个输入序列。例如，在语音识别中，由于协同发音，当前声音作为音素的正确解释可能取决于未来几个音素，甚至潜在的可能取决于未来的几个词，因为词与附近的词之间的存在语义依赖：如果当前的词有两种声学上合理的解释，我们可能要在更远的未来（和过去）寻找信息区分它们。这在手写识别和许多其他序列到序列学习的任务中也是如此，将会在下一节中描述。

双向循环神经网络（或双向 RNN）为满足这种需要而被发明 (Schuster and Paliwal, 1997)。他们在需要双向信息的应用中非常成功 (Graves, 2012)，如手写

识别 (Graves *et al.*, 2008; Graves and Schmidhuber, 2009), 语音识别 (Graves and Schmidhuber, 2005; Graves *et al.*, 2013) 以及生物信息学 (Baldi *et al.*, 1999)。

顾名思义，双向 RNN 结合时间上从序列起点开始移动的 RNN 和另一个时间上从序列末尾开始移动的 RNN。图 10.11 展示了典型的双向 RNN，其中 $h^{(t)}$ 代表通过时间向前移动的子 RNN 的状态， $g^{(t)}$ 代表通过时间向后移动的子 RNN 的状态。这允许输出单元 $o^{(t)}$ 能够计算同时依赖于过去和未来且对时刻 t 的输入值最敏感的表示，而不必指定 t 周围固定大小的窗口（这是前馈网络、卷积网络或具有固定大小的先行缓存器的常规 RNN 所必须要做的）。

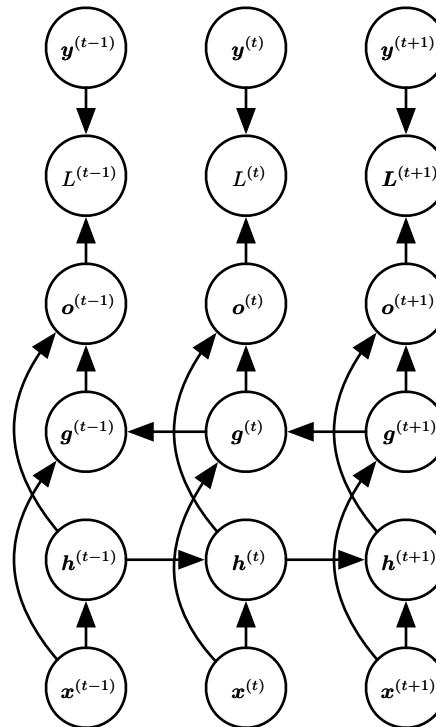


图 10.11: 典型的双向循环神经网络中的计算，意图学习将输入序列 \mathbf{x} 映射到目标序列 \mathbf{y} (在每个步骤 t 具有损失 $L^{(t)}$)。循环性 h 在时间上向前传播信息 (向右)，而循环性 g 在时间上向后传播信息 (向左)。因此在每个点 t ，输出单元 $o^{(t)}$ 可以受益于输入 $h^{(t)}$ 中关于过去的相关概要以及输入 $g^{(t)}$ 中关于未来的相关概要。

这个想法可以自然地扩展到 2 维输入，如图像，由四个 RNN 组成，每一个沿着四个方向中的一个计算：上、下、左、右。如果 RNN 能够学习到承载长期信息，

那在 2 维网格每个点 (i, j) 的输出 $O_{i,j}$ 就能计算一个能捕捉到大多局部信息但仍依赖于长期输入的表示。相比卷积网络，应用于图像的 RNN 计算成本通常更高，但允许同一特征图的特征之间存在长期横向的相互作用 (Visin *et al.*, 2015; Kalchbrenner *et al.*, 2015)。实际上，对于这样的 RNN，前向传播公式可以写成表示使用卷积的形式，计算自底向上到每一层的输入（在整合横向相互作用的特征图的循环传播之前）。

10.4 基于编码-解码的序列到序列架构

我们已经在图 10.5 看到 RNN 如何将输入序列映射成固定大小的向量，在图 10.9 中看到 RNN 如何将固定大小的向量映射成一个序列，在图 10.3、图 10.4、图 10.10 和图 10.11 中看到 RNN 如何将一个输入序列映射到等长的输出序列。

本节我们讨论如何训练 RNN，使其将输入序列映射到不一定等长的输出序列。这在许多场景中都有应用，如语音识别、机器翻译或问答，其中训练集的输入和输出序列的长度通常不相同（虽然它们的长度可能相关）。

我们经常将 RNN 的输入称为“上下文”。我们希望产生此上下文的表示， C 。这个上下文 C 可能是一个概括输入序列 $\mathbf{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n_x)})$ 的向量或者向量序列。

用于映射可变长度序列到另一可变长度序列最简单的 RNN 架构最初由 Cho *et al.* (2014a) 提出，之后不久由 Sutskever *et al.* (2014) 独立开发，并且第一个使用这种方法获得翻译的最好结果。前一系统是对另一个机器翻译系统产生的建议进行评分，而后者使用独立的循环网络生成翻译。这些作者分别将该架构称为编码-解码或序列到序列架构，如图 10.12 所示。这个想法非常简单：(1) 编码器 (encoder) 或读取器 (reader) 或输入 (input) RNN 处理输入序列。编码器输出上下文 C (通常是最终隐藏状态的简单函数)。(2) 解码器 (decoder) 或写入器 (writer) 或输出 (output) RNN 则以固定长度的向量 (如图 10.9) 为条件产生输出序列 $\mathbf{Y} = (\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(n_y)})$ 。这种架构对比本章前几节提出的架构的创新之处在于长度 n_x 和 n_y 可以彼此不同，而之前的架构约束 $n_x = n_y = \tau$ 。在序列到序列的架构中，两个 RNN 共同训练以最大化 $\log P(\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(n_y)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n_x)})$ (关于训练集中所有 \mathbf{x} 和 \mathbf{y} 对的平均)。编码器 RNN 的最后一个状态 \mathbf{h}_{n_x} 通常被当作输入的表示 C 并作为解码器 RNN 的输入。

如果上下文 C 是一个向量，则解码器 RNN 只是在第 10.2.4 节描述的向量到序

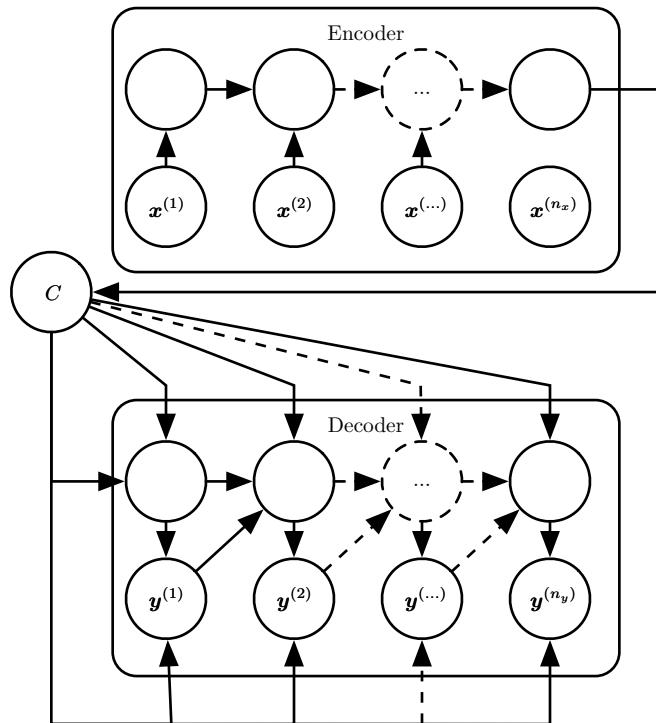


图 10.12: 在给定输入序列 $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n_x)})$ 的情况下学习生成输出序列 $(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(n_y)})$ 的编码器-解码器或序列到序列的 RNN 架构的示例。它由读取输入序列的编码器 RNN 以及生成输出序列（或计算给定输出序列的概率）的解码器 RNN 组成。编码器 RNN 的最终隐藏状态用于计算一般为固定大小的上下文变量 C ， C 表示输入序列的语义概要并且作为解码器 RNN 的输入。

列 RNN。正如我们所见，向量到序列 RNN 至少有两种接受输入的方法。输入可以被提供为 RNN 的初始状态，或连接到每个时间步中的隐藏单元。这两种方式也可以结合。

这里并不强制要求编码器与解码器的隐藏层具有相同的大小。

此架构的一个明显不足是，编码器 RNN 输出的上下文 C 的维度太小而难以适当地概括一个长序列。这种现象由 Bahdanau *et al.* (2015) 在机器翻译中观察到。他们提出让 C 成为可变长度的序列，而不是一个固定大小的向量。此外，他们还引入了将序列 C 的元素和输出序列的元素相关联的 **注意力机制** (attention mechanism)。读者可在第 12.4.5.1 节了解更多细节。

10.5 深度循环网络

大多数 RNN 中的计算可以分解成三块参数及其相关的变换：

1. 从输入到隐藏状态，
2. 从前一隐藏状态到下一隐藏状态，以及
3. 从隐藏状态到输出。

根据图 10.3 中的 RNN 架构，这三个块都与单个权重矩阵相关联。换句话说，当网络被展开时，每个块对应一个浅的变换。能通过深度 MLP 内单个层来表示的变换称为浅变换。通常，这是由学成的仿射变换和一个固定非线性表示组成的变换。

在这些操作中引入深度会有利的吗？实验证据 (Graves *et al.*, 2013; Pascanu *et al.*, 2014a) 强烈暗示理应如此。实验证据与我们需要足够的深度以执行所需映射的想法一致。读者可以参考 Schmidhuber (1992); El Hihi and Bengio (1996) 或 Jaeger (2007a) 了解更早的关于深度 RNN 的研究。

Graves *et al.* (2013) 第一个展示了将 RNN 的状态分为多层的显著好处，如图 10.13 (左)。我们可以认为，在图 10.13 (a) 所示层次结构中较低的层起到了将原始输入转化为对更高层的隐藏状态更合适表示的作用。Pascanu *et al.* (2014a) 更进一步提出在上述三个块中各使用一个单独的 MLP (可能是深度的)，如图 10.13 (b) 所示。考虑表示容量，我们建议在这三个步中都分配足够的容量，但增加深度可能会因为优化困难而损害学习效果。在一般情况下，更容易优化较浅的架构，加入图 10.13 (b) 的额外深度导致从时间步 t 的变量到时间步 $t+1$ 的最短路径变得更长。例如，如果有单个隐藏层的 MLP 被用于状态到状态的转换，那么与图 10.3 相比，我们就会加倍任何两个不同时间步变量之间最短路径的长度。然而 Pascanu *et al.* (2014a) 认为，在隐藏到隐藏的路径中引入跳跃连接可以缓和这个问题，如图 10.13 (c) 所示。

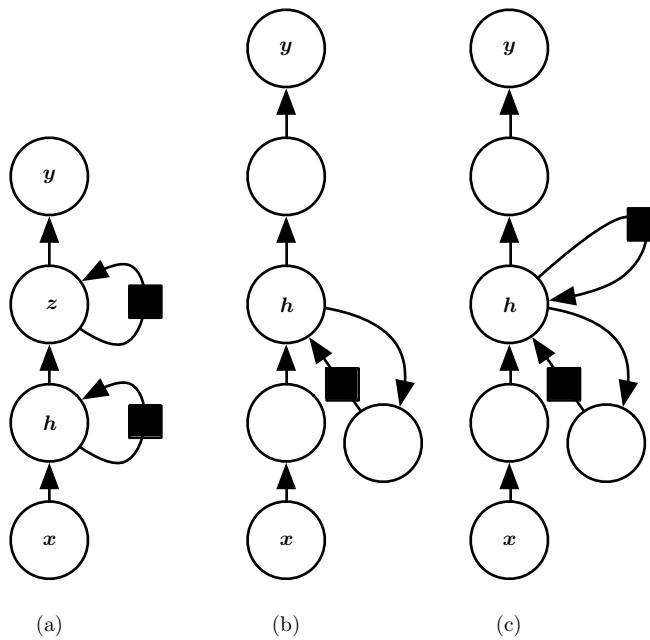


图 10.13: 循环神经网络可以通过许多方式变得更深 (Pascanu *et al.*, 2014a)。(a) 隐藏循环状态可以被分解为具有层次的组。(b) 可以向输入到隐藏, 隐藏到隐藏以及隐藏到输出的部分引入更深的计算 (如 MLP)。这可以延长链接不同时间步的最短路径。(c) 可以引入跳跃连接来缓解路径延长的效果。

10.6 递归神经网络

递归神经网络²代表循环网络的另一个扩展, 它被构造为深的树状结构而不是 RNN 的链状结构, 因此是不同类型的计算图。递归网络的典型计算图如图 10.14 所示。递归神经网络由 Pollack (1990) 引入, 而 Bottou (2011) 描述了这类网络的潜在用途——学习推论。递归网络已成功地应用于输入是数据结构的神经网络 (Frasconi *et al.*, 1997, 1998), 如自然语言处理 (Socher *et al.*, 2011a,c, 2013a) 和计算机视觉 (Socher *et al.*, 2011b)。

递归网络的一个明显优势是, 对于具有相同长度 τ 的序列, 深度 (通过非线性操作的组合数量来衡量) 可以急剧地从 τ 减小为 $\mathcal{O}(\log \tau)$, 这可能有助于解决长期

²我们建议不要将“递归神经网络”缩写为“RNN”, 以免与“循环神经网络”混淆。

依赖。一个悬而未决的问题是如何以最佳的方式构造树。一种选择是使用不依赖于数据的树结构，如平衡二叉树。在某些应用领域，外部方法可以为选择适当的树结构提供借鉴。例如，处理自然语言的句子时，用于递归网络的树结构可以被固定为句子语法分析树的结构（可以由自然语言语法分析程序提供）(Socher *et al.*, 2011a,c)。理想的情况下，人们希望学习器自行发现和推断适合于任意给定输入的树结构，如(Bottou, 2011) 所建议。

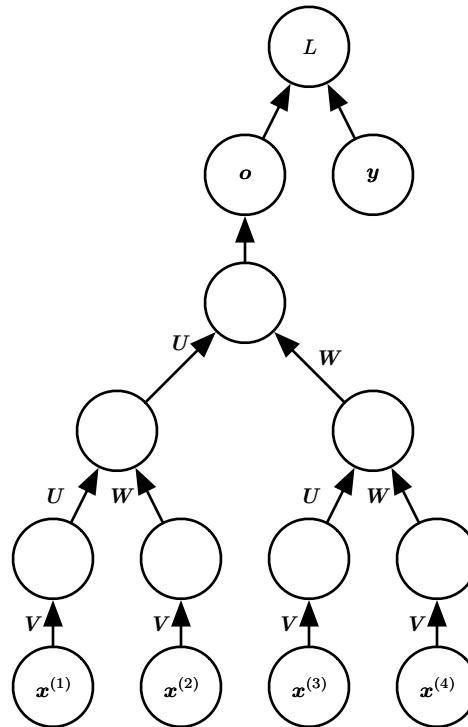


图 10.14: 递归网络将循环网络的链状计算图推广到树状计算图。可变大小的序列 $x^{(1)}, x^{(2)}, \dots, x^{(t)}$ 可以通过固定的参数集合（权重矩阵 U, V, W ）映射到固定大小的表示（输出 o ）。该图展示了监督学习的情况，其中提供了一些与整个序列相关的目标 y 。

递归网络想法的变种存在很多。例如，Frasconi *et al.* (1997) 和 Frasconi *et al.* (1998) 将数据与树结构相关联，并将输入和目标与树的单独节点相关联。由每个节点执行的计算无须是传统的人工神经计算（所有输入的仿射变换后跟一个单调非线性）。例如，Socher *et al.* (2013a) 提出用张量运算和双线性形式，在这之前人们已经发现当概念是由连续向量（嵌入）表示时，这种方式有利于建模概念之间的联系

(Weston *et al.*, 2010; Bordes *et al.*, 2012)。

10.7 长期依赖的挑战

学习循环网络长期依赖的数学挑战在第 8.2.5 节中引入。根本问题是，经过许多阶段传播后的梯度倾向于消失（大部分情况）或爆炸（很少，但对优化过程影响很大）。即使我们假设循环网络是参数稳定的（可存储记忆，且梯度不爆炸），但长期依赖的困难来自比短期相互作用指数小的权重（涉及许多 Jacobian 相乘）。许多资料提供了更深层次的讨论 (Hochreiter, 1991a; Doya, 1993; Bengio *et al.*, 1994b; Pascanu *et al.*, 2013a)。在这一节中，我们会更详细地描述该问题。其余几节介绍克服这个问题的方法。

循环网络涉及相同函数的多次组合，每个时间步一次。这些组合可以导致极端非线性行为，如图 10.15 所示。

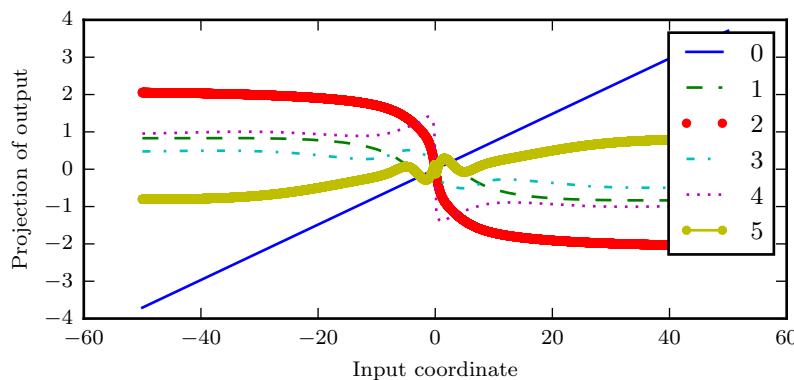


图 10.15：重复组合函数。当组合许多非线性函数（如这里所示的线性 \tanh 层）时，结果是高度非线性的，通常大多数值与微小的导数相关联，也有一些具有大导数的值，以及在增加和减小之间的多次交替。此处，我们绘制从 100 维隐藏状态降到单个维度的线性投影，绘制于 y 轴上。 x 轴是 100 维空间中沿着随机方向的初始状态的坐标。因此，我们可以将该图视为高维函数的线性截面。曲线显示每个时间步之后的函数，或者等价地，转换函数被组合一定次数之后。

特别地，循环神经网络所使用的函数组合有点像矩阵乘法。我们可以认为，循

环联系

$$\mathbf{h}^{(t)} = \mathbf{W}^\top \mathbf{h}^{(t-1)} \quad (10.36)$$

是一个非常简单的、缺少非线性激活函数和输入 \mathbf{x} 的循环神经网络。如第 8.2.5 节描述，这种递推关系本质上描述了幂法。它可以被简化为

$$\mathbf{h}^{(t)} = (\mathbf{W}^t)^\top \mathbf{h}^{(0)}, \quad (10.37)$$

而当 \mathbf{W} 符合下列形式的特征分解

$$\mathbf{W} = \mathbf{Q} \Lambda \mathbf{Q}^\top, \quad (10.38)$$

其中 \mathbf{Q} 正交，循环性可进一步简化为

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

特征值提升到 t 次后，导致幅值不到一的特征值衰减到零，而幅值大于一的就会激增。任何不与最大特征向量对齐的 $\mathbf{h}^{(0)}$ 的部分将最终被丢弃。

这个问题是针对循环网络的。在标量情况下，想象多次乘一个权重 w 。该乘积 w^t 消失还是爆炸取决于 w 的幅值。然而，如果每个时刻使用不同权重 $w^{(t)}$ 的非循环网络，情况就不同了。如果初始状态给定为 1，那么时刻 t 的状态可以由 $\prod_t w^{(t)}$ 给出。假设 $w^{(t)}$ 的值是随机生成的，各自独立，且有 0 均值 v 方差。乘积的方差就为 $\mathcal{O}(v^n)$ 。为了获得某些期望的方差 v^* ，我们可以选择单个方差为 $v = \sqrt[n]{v^*}$ 权重。因此，非常深的前馈网络通过精心设计的比例可以避免梯度消失和爆炸问题，如 Sussillo (2014) 所主张的。

RNN 梯度消失和爆炸问题是不同研究人员独立发现 (Hochreiter, 1991a; Bengio *et al.*, 1993, 1994b)。有人可能会希望通过简单地停留在梯度不消失或爆炸的参数空间来避免这个问题。不幸的是，为了储存记忆并对小扰动具有鲁棒性，RNN 必须进入参数空间中的梯度消失区域 (Bengio *et al.*, 1993, 1994b)。具体来说，每当模型能够表示长期依赖时，长期相互作用的梯度幅值就会变得指数小（相比短期相互作用的梯度幅值）。这并不意味着这是不可能学习的，由于长期依赖关系的信号很容易被短期相关性产生的最小波动隐藏，因而学习长期依赖可能需要很长的时间。实践中，Bengio *et al.* (1994b) 的实验表明，当我们增加了需要捕获的依赖关系的跨度，基于梯度的优化变得越来越困难，SGD 在长度仅为 10 或 20 的序列上成功训练传统 RNN 的概率迅速变为 0。

将循环网络作为动力系统更深入探讨的资料见 Doya (1993); Bengio *et al.* (1994b); Siegelmann and Sontag (1995) 及 Pascanu *et al.* (2013b) 的回顾。本章的其余部分将讨论目前已经提出的降低学习长期依赖（在某些情况下，允许一个 RNN 学习横跨数百步的依赖）难度的不同方法，但学习长期依赖的问题仍是深度学习中的一个主要挑战。

10.8 回声状态网络

从 $\mathbf{h}^{(t-1)}$ 到 $\mathbf{h}^{(t)}$ 的循环权重映射以及从 $\mathbf{x}^{(t)}$ 到 $\mathbf{h}^{(t)}$ 的输入权重映射是循环网络中最难学习的参数。研究者 (Jaeger, 2003; Maass *et al.*, 2002; Jaeger and Haas, 2004; Jaeger, 2007b) 提出避免这种困难的方法是设定循环隐藏单元，使其能很好地捕捉过去输入历史，并且只学习输出权重。回声状态网络 (echo state network) 或 ESN (Jaeger and Haas, 2004; Jaeger, 2007b)，以及流体状态机 (liquid state machine) (Maass *et al.*, 2002) 分别独立地提出了这种想法。后者是类似的，只不过它使用脉冲神经元 (二值输出) 而不是 ESN 中的连续隐藏单元。ESN 和流体状态机都被称为储层计算 (reservoir computing) (Lukoševičius and Jaeger, 2009)，因为隐藏单元形成了可能捕获输入历史不同方面的临时特征池。

储层计算循环网络类似于核机器，这是思考它们的一种方式：它们将任意长度的序列（到时刻 t 的输入历史）映射为一个长度固定的向量（循环状态 $\mathbf{h}^{(t)}$ ），之后可以施加一个线性预测算子（通常是一个线性回归）以解决感兴趣的问题。训练准则就可以很容易地设计为输出权重的凸函数。例如，如果输出是从隐藏单元到输出目标的线性回归，训练准则就是均方误差，由于是凸的就可以用简单的学习算法可靠地解决 (Jaeger, 2003)。

因此，重要的问题是：我们如何设置输入和循环权重才能让一组丰富的历史可以在循环神经网络的状态中表示？储层计算研究给出的答案是将循环网络视为动态系统，并设定让动态系统接近稳定边缘的输入和循环权重。

最初的想法是使状态到状态转换函数的 Jacobian 矩阵的特征值接近 1。如第 8.2.5 节解释，循环网络的一个重要特征就是 Jacobian 矩阵的特征值谱 $\mathbf{J}^{(t)} = \frac{\partial s^{(t)}}{\partial s^{(t-1)}}$ 。特别重要的是 $\mathbf{J}^{(t)}$ 的谱半径 (spectral radius)，定义为特征值的最大绝对值。

为了解谱半径的影响，可以考虑反向传播中 Jacobian 矩阵 \mathbf{J} 不随 t 改变的简单

情况。例如当网络是纯线性时，会发生这种情况。假设 \mathbf{J} 特征值 λ 对应的特征向量为 \mathbf{v} 。考虑当我们通过时间向后传播梯度向量时会发生什么。如果刚开始的梯度向量为 \mathbf{g} ，然后经过反向传播的一个步骤后，我们将得到 $\mathbf{J}\mathbf{g}$ ， n 步之后我们会得到 $\mathbf{J}^n\mathbf{g}$ 。现在考虑如果我们向后传播扰动版本的 \mathbf{g} 会发生什么。如果我们刚开始是 $\mathbf{g} + \delta\mathbf{v}$ ，一步之后，我们会得到 $\mathbf{J}(\mathbf{g} + \delta\mathbf{v})$ 。 n 步之后，我们将得到 $\mathbf{J}^n(\mathbf{g} + \delta\mathbf{v})$ 。由此我们可以看出，由 \mathbf{g} 开始的反向传播和由 $\mathbf{g} + \delta\mathbf{v}$ 开始的反向传播， n 步之后偏离 $\delta\mathbf{J}^n\mathbf{v}$ 。如果 \mathbf{v} 选择为 \mathbf{J} 特征值 λ 对应的一个单位特征向量，那么在每一步乘 Jacobian 矩阵只是简单地缩放。反向传播的两次执行分离的距离为 $\delta|\lambda|^n$ 。当 \mathbf{v} 对应于最大特征值 $|\lambda|$ ，初始扰动为 δ 时这个扰动达到可能的最宽分离。

当 $|\lambda| > 1$ ，偏差 $\delta|\lambda|^n$ 就会指数增长。当 $|\lambda| < 1$ ，偏差就会变得指数小。

当然，这个例子假定 Jacobian 矩阵在每个时间步是相同的，即对应于没有非线性循环网络。当非线性存在时，非线性的导数将在许多时间步后接近零，并有助于防止因过大的谱半径而导致的爆炸。事实上，关于回声状态网络的最近工作提倡使用远大于 1 的谱半径 (Yildiz *et al.*, 2012; Jaeger, 2012)。

我们已经说过多次，通过反复矩阵乘法的反向传播同样适用于没有非线性的正向传播的网络，其状态为 $\mathbf{h}^{(t+1)} = \mathbf{h}^{(t)\top} \mathbf{W}$ 。

如果线性映射 \mathbf{W}^\top 在 L^2 范数的测度下总是缩小 \mathbf{h} ，那么我们说这个映射是 **收缩** (contractive) 的。当谱半径小于一，则从 $\mathbf{h}^{(t)}$ 到 $\mathbf{h}^{(t+1)}$ 的映射是收缩的，因此小变化在每个时间步后变得更小。当我们使用有限精度（如 32 位整数）来存储状态向量时，必然会使网络忘掉过去的信息。

Jacobian 矩阵告诉我们 $\mathbf{h}^{(t)}$ 一个微小的变化如何向前一步传播，或等价的， $\mathbf{h}^{(t+1)}$ 的梯度如何向后一步传播。需要注意的是， \mathbf{W} 和 \mathbf{J} 都不需要是对称的（尽管它们是实方阵），因此它们可能有复的特征值和特征向量，其中虚数分量对应于潜在的振荡行为（如果迭代地应用同一 Jacobian）。即使 $\mathbf{h}^{(t)}$ 或 $\mathbf{h}^{(t)}$ 中有趣的小变化在反向传播中是实值的，它们仍可以用这样的复数基表示。重要的是，当向量乘以矩阵时，这些复数基的系数幅值（复数的绝对值）会发生什么变化。幅值大于 1 的特征值对应于放大（如果反复应用则指数增长）或收缩（如果反复应用则指数减小）。

非线性映射情况时，Jacobian 会在每一步任意变化。因此，动态量变得更加复杂。然而，一个小的初始变化多步之后仍然会变成一个大的变化。纯线性和非线性情况的一个不同之处在于使用压缩非线性（如 tanh）可以使循环动态量有界。注意，即使前向传播动态量有界，反向传播的动态量仍然可能无界，例如，当 tanh 序列

都在它们状态中间的线性部分，并且由谱半径大于 1 的权重矩阵连接。然而，所有 tanh 单元同时位于它们的线性激活点是非常罕见的。

回声状态网络的策略是简单地固定权重使其具有一定的谱半径如 3，其中信息通过时间前向传播，但会由于饱和非线性单元（如 tanh）的稳定作用而不会爆炸。

最近，已经有研究表明，用于设置 ESN 权重的技术可以用来初始化完全可训练的循环网络的权重（通过时间反向传播来训练隐藏到隐藏的循环权重），帮助学习长期依赖 (Sutskever, 2012; Sutskever *et al.*, 2013)。在这种设定下，结合第 8.4 节中稀疏初始化的方案，设置 1.2 的初始谱半径表现不错。

10.9 渗漏单元和其他多时间尺度的策略

处理长期依赖的一种方法是设计工作在多个时间尺度的模型，使模型的某些部分在细粒度时间尺度上操作并能处理小细节，而其他部分在粗时间尺度上操作并能把遥远过去的信息更有效地传递过来。存在多种同时构建粗细时间尺度的策略。这些策略包括在时间轴增加跳跃连接，“渗漏单元”使用不同时间常数整合信号，并去除一些用于建模细粒度时间尺度的连接。

10.9.1 时间维度的跳跃连接

增加从遥远过去的变量到目前变量的直接连接是得到粗时间尺度的一种方法。使用这样跳跃连接的想法可以追溯到 Lin *et al.* (1996)，紧接是向前馈网络引入延迟的想法 (Lang and Hinton, 1988)。在普通的循环网络中，循环从时刻 t 的单元连接到时刻 $t + 1$ 单元。构造较长的延迟循环网络是可能的 (Bengio, 1991)。

正如我们在第 8.2.5 节看到，梯度可能关于时间步数呈指数消失或爆炸。(Lin *et al.*, 1996) 引入了 d 延时的循环连接以减轻这个问题。现在导数指数减小的速度与 $\frac{\tau}{d}$ 相关而不是 τ 。既然同时存在延迟和单步连接，梯度仍可能成 t 指数爆炸。这允许学习算法捕获更长的依赖性，但不是所有的长期依赖都能在这种方式下良好地表示。

10.9.2 渗漏单元和一系列不同时间尺度

获得导数乘积接近 1 的另一方式是设置线性自连接单元，并且这些连接的权重接近 1。

我们对某些 v 值应用更新 $\mu^{(t)} \leftarrow \alpha\mu^{(t-1)} + (1 - \alpha)v^{(t)}$ 累积一个滑动平均值 $\mu^{(t)}$ ，其中 α 是一个从 $\mu^{(t-1)}$ 到 $\mu^{(t)}$ 线性自连接的例子。当 α 接近 1 时，滑动平均值能记住过去很长一段时间的信息，而当 α 接近 0，关于过去的信息被迅速丢弃。线性自连接的隐藏单元可以模拟滑动平均的行为。这种隐藏单元称为 **渗漏单元** (leaky unit)。

d 时间步的跳跃连接可以确保单元总能被 d 个时间步前的那个值影响。使用权重接近 1 的线性自连接是确保该单元可以访问过去值的不同方式。线性自连接通过调节实值 α 更平滑灵活地调整这种效果，而不是调整整数值的跳跃长度。

这个想法由 Mozer (1992) 和 El Hihi and Bengio (1996) 提出。在回声状态网络中，渗漏单元也被发现很有用 (Jaeger *et al.*, 2007)。

我们可以通过两种基本策略设置渗漏单元使用的时间常数。一种策略是手动将其固定为常数，例如在初始化时从某些分布采样它们的值。另一种策略是使时间常数成为自由变量，并学习出来。在不同时间尺度使用这样的渗漏单元似乎能帮助学习长期依赖 (Mozer, 1992; Pascanu *et al.*, 2013a)。

10.9.3 删除连接

处理长期依赖另一种方法是在多个时间尺度组织 RNN 状态的想法 (El Hihi and Bengio, 1996)，信息在较慢的时间尺度上更容易长距离流动。

这个想法与之前讨论的时间维度上的跳跃连接不同，因为它涉及主动删除长度为一的连接并用更长的连接替换它们。以这种方式修改的单元被迫在长时间尺度上运作。而通过时间跳跃连接是添加边。收到这种新连接的单元，可以学习在长时间尺度上运作，但也可以选择专注于自己其他的短期连接。

强制一组循环单元在不同时间尺度上运作有不同的方式。一种选择是使循环单元变成渗漏单元，但不同的单元组关联不同的固定时间尺度。这由 Mozer (1992) 提出，并被成功应用于 Pascanu *et al.* (2013a)。另一种选择是使显式且离散的更新发生在不同的时间，不同的单元组有不同的频率。这是 El Hihi and Bengio (1996) 和 Koutnik *et al.* (2014) 的方法。它在一些基准数据集上表现不错。

10.10 长短期记忆和其他门控 RNN

本文撰写之时，实际应用中最有效的序列模型称为**门控 RNN**（gated RNN）。包括基于**长短期记忆**（long short-term memory）和基于**门控循环单元**（gated recurrent unit）的网络。

像渗漏单元一样，门控 RNN 想法也是基于生成通过时间的路径，其中导数既不消失也不发生爆炸。渗漏单元通过手动选择常量的连接权重或参数化的连接权重来达到这一目的。门控 RNN 将其推广为在每个时间步都可能改变的连接权重。

渗漏单元允许网络在较长持续时间内积累信息（诸如用于特定特征或类的线索）。然而，一旦该信息被使用，让神经网络遗忘旧的状态可能是有用的。例如，如果一个序列是由子序列组成，我们希望渗漏单元能在各子序列内积累线索，我们需要将状态设置为 0 以忘记旧状态的机制。我们希望神经网络学会决定何时清除状态，而不是手动决定。这就是门控 RNN 要做的事。

10.10.1 LSTM

引入自循环的巧妙构思，以产生梯度长时间持续流动的路径是初始**长短期记忆**（long short-term memory, LSTM）模型的核心贡献 (Hochreiter and Schmidhuber, 1997)。其中一个关键扩展是使自循环的权重视上下文而定，而不是固定的 (Gers *et al.*, 2000)。门控此自循环（由另一个隐藏单元控制）的权重，累积的时间尺度可以动态地改变。在这种情况下，即使是具有固定参数的 LSTM，累积的时间尺度也可以因输入序列而改变，因为时间常数是模型本身的输出。LSTM 已经在许多应用中取得重大成功，如无约束手写识别 (Graves *et al.*, 2009)、语音识别 (Graves *et al.*, 2013; Graves and Jaitly, 2014)、手写生成 (Graves, 2013)、机器翻译 (Sutskever *et al.*, 2014)、为图像生成标题 (Kiros *et al.*, 2014b; Vinyals *et al.*, 2014b; Xu *et al.*, 2015) 和解析 (Vinyals *et al.*, 2014a)。

LSTM 块如图 10.16 所示。在浅循环网络的架构下，相应的前向传播公式如下。更深的架构也被成功应用 (Graves *et al.*, 2013; Pascanu *et al.*, 2014a)。LSTM 循环网络除了外部的 RNN 循环外，还具有内部的“LSTM 细胞”循环（自环），因此 LSTM 不是简单地向输入和循环单元的仿射变换之后施加一个逐元素的非线性。与普通的循环网络类似，每个单元有相同的输入和输出，但也有更多的参数和控制信息流动的门控单元系统。最重要的组成部分是状态单元 $s_i^{(t)}$ ，与前一节讨论的渗漏

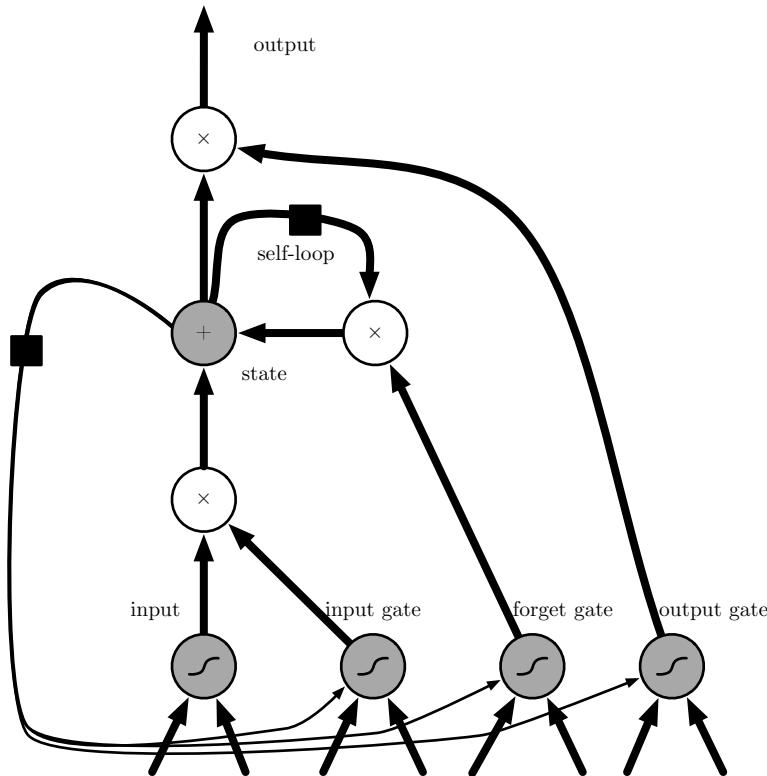


图 10.16: LSTM 循环网络“细胞”的框图。细胞彼此循环连接，代替一般循环网络中普通的隐藏单元。这里使用常规的人工神经元计算输入特征。如果 sigmoid 输入门允许，它的值可以累加到状态。状态单元具有线性自循环，其权重由遗忘门控制。细胞的输出可以被输出门关闭。所有门控单元都具有 sigmoid 非线性，而输入单元可具有任意的压缩非线性。状态单元也可以用作门控单元的额外输入。黑色方块表示单个时间步的延迟。

单元有类似的线性自环。然而，此处自环的权重（或相关联的时间常数）由遗忘门（forget gate） $f_i^{(t)}$ 控制（时刻 t 和细胞 i ），由 sigmoid 单元将权重设置为 0 和 1 之间的值：

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

其中 $x^{(t)}$ 是当前输入向量， h^t 是当前隐藏层向量， h^t 包含所有 LSTM 细胞的输出。 b^f, U^f, W^f 分别是偏置、输入权重和遗忘门的循环权重。因此 LSTM 细胞内部状态

以如下方式更新，其中有一个条件的自环权重 $f_i^{(t)}$ ：

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

其中 \mathbf{b} , \mathbf{U} , \mathbf{W} 分别是 LSTM 细胞中的偏置、输入权重和遗忘门的循环权重。外部输入门 (external input gate) 单元 $g_i^{(t)}$ 以类似遗忘门 (使用 sigmoid 获得一个 0 和 1 之间的值) 的方式更新，但有自身的参数：

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

LSTM 细胞的输出 $h_i^{(t)}$ 也可以由输出门 (output gate) $q_i^{(t)}$ 关闭 (使用 sigmoid 单元作为门控)：

$$h_i^{(t)} = \tanh(s_i^{(t)}) q_i^{(t)}, \quad (10.43)$$

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

其中 \mathbf{b}^o , \mathbf{U}^o , \mathbf{W}^o 分别是偏置、输入权重和遗忘门的循环权重。在这些变体中，可以选择使用细胞状态 $s_i^{(t)}$ 作为额外的输入 (及其权重)，输入到第 i 个单元的三个门，如图 10.16 所示。这将需要三个额外的参数。

LSTM 网络比简单的循环架构更易于学习长期依赖，先是用于测试长期依赖学习能力的人工数据集 (Bengio *et al.*, 1994c; Hochreiter and Schmidhuber, 1997; Hochreiter *et al.*, 2001)，然后是在具有挑战性的序列处理任务上获得最先进的表现 (Graves, 2012, 2013; Sutskever *et al.*, 2014)。LSTM 的变体和替代也已经被研究和使用，这将在下文进行讨论。

10.10.2 其他门控 RNN

LSTM 架构中哪些部分是真正必须的？还可以设计哪些其他成功架构允许网络动态地控制时间尺度和不同单元的遗忘行为？

最近关于门控 RNN 的工作给出了这些问题的某些答案，其单元也被称为门控循环单元或 GRU (Cho *et al.*, 2014c; Chung *et al.*, 2014, 2015a; Jozefowicz *et al.*, 2015; Chrupala *et al.*, 2015)。与 LSTM 的主要区别是，单个门控单元同时控制遗忘因子

和更新状态单元的决定。更新公式如下：

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

其中 u 代表“更新”门， r 表示“复位”门。它们的值就如通常所定义的：

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

和

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

复位和更新门能独立地“忽略”状态向量的一部分。更新门像条件渗漏累积器一样可以线性门控任意维度，从而选择将它复制（在 sigmoid 的一个极端）或完全由新的“目标状态”值（朝向渗漏累积器的收敛方向）替换并完全忽略它（在另一个极端）。复位门控制当前状态中哪些部分用于计算下一个目标状态，在过去状态和未来状态之间引入了附加的非线性效应。

围绕这一主题可以设计更多的变种。例如复位门（或遗忘门）的输出可以在多个隐藏单元间共享。或者，全局门的乘积（覆盖一整组的单元，例如整一层）和一个局部门（每单元）可用于结合全局控制和局部控制。然而，一些调查发现这些 LSTM 和 GRU 架构的变种，在广泛的任务中难以明显地同时击败这两个原始架构 (Greff *et al.*, 2015; Jozefowicz *et al.*, 2015)。Greff *et al.* (2015) 发现其中的关键因素是遗忘门，而 Jozefowicz *et al.* (2015) 发现向 LSTM 遗忘门加入 1 的偏置（由 Gers *et al.* (2000) 提倡）能让 LSTM 变得与已探索的最佳变种一样健壮。

10.11 优化长期依赖

我们已经在第 8.2.5 节和第 10.7 节中描述过在许多时间步上优化 RNN 时发生的梯度消失和爆炸的问题。

由 Martens and Sutskever (2011) 提出了一个有趣的想法是，二阶导数可能在一阶导数消失的同时消失。二阶优化算法可以大致被理解为将一阶导数除以二阶导数（在更高维数，由梯度乘以 Hessian 的逆）。如果二阶导数与一阶导数以类似的速率收缩，那么一阶和二阶导数的比率可保持相对恒定。不幸的是，二阶方法有许多缺

点，包括高的计算成本、需要一个大的小批量、并且倾向于被吸引到鞍点。Martens and Sutskever (2011) 发现采用二阶方法的不错结果。之后，Sutskever *et al.* (2013) 发现使用较简单的方法可以达到类似的结果，例如经过谨慎初始化的 Nesterov 动量法。更详细的内容参考 Sutskever (2012)。应用于 LSTM 时，这两种方法在很大程度上会被单纯的 SGD (甚至没有动量) 取代。这是机器学习中一个延续的主题，设计一个易于优化模型通常比设计出更加强大的优化算法更容易。

10.11.1 截断梯度

如第 8.2.4 节讨论，强非线性函数（如由许多时间步计算的循环网络）往往倾向于非常大或非常小幅度的梯度。如图 8.3 和图 10.17 所示，我们可以看到，目标函数（作为参数的函数）存在一个伴随“悬崖”的“地形”：宽且相当平坦区域被目标函数变化快的小区域隔开，形成了一种悬崖。

这导致的困难是，当参数梯度非常大时，梯度下降的参数更新可以将参数抛出很远，进入目标函数较大的区域，到达当前解所作的努力变成了无用功。梯度告诉我们，围绕当前参数的无穷小区域内最速下降的方向。这个无穷小区域之外，代价函数可能开始沿曲线背面而上。更新必须被选择为足够小，以避免过分穿越向上的曲面。我们通常使用衰减速度足够慢的学习率，使连续的步骤具有大致相同的学习率。适合于一个相对线性的地形部分的步长经常在下一步进入地形中更加弯曲的部分时变得不适合，会导致上坡运动。

一个简单的解决方案已被从业者使用多年：**截断梯度** (clipping the gradient)。此想法有不同实例 (Mikolov, 2012; Pascanu *et al.*, 2013a)。一种选择是在参数更新之前，逐元素地截断小批量产生的参数梯度 (Mikolov, 2012)。另一种是在参数更新之前截断梯度 \mathbf{g} 的范数 $\|\mathbf{g}\|$ (Pascanu *et al.*, 2013a)：

$$\text{if } \|\mathbf{g}\| > v \quad (10.48)$$

$$\mathbf{g} \leftarrow \frac{\mathbf{g}v}{\|\mathbf{g}\|}, \quad (10.49)$$

其中 v 是范数上界， \mathbf{g} 用来更新参数。因为所有参数（包括不同的参数组，如权重和偏置）的梯度被单个缩放因子联合重整合，所以后一方法具有的优点是保证了每个步骤仍然是在梯度方向上的，但实验表明两种形式类似。虽然参数更新与真实梯度具有相同的方向梯度，经过梯度范数截断，参数更新的向量范数现在变得有界。这种有界梯度能避免执行梯度爆炸时的有害一步。事实上，当梯度大小高于阈值时，即

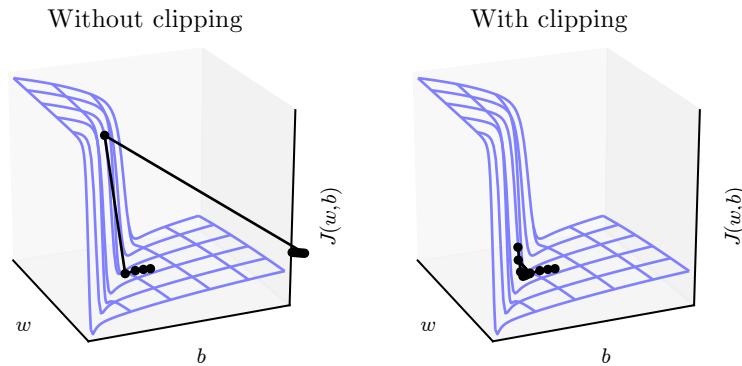


图 10.17: 梯度截断在有两个参数 w 和 b 的循环网络中的效果示例。梯度截断可以使梯度下降在极陡峭的悬崖附近更合理地执行。这些陡峭的悬崖通常发生在循环网络中，位于循环网络近似线性的附近。悬崖在时间步的数量上呈指数地陡峭，因为对于每个时间步，权重矩阵都自乘一次。(左)没有梯度截断的梯度下降越过这个小峡谷的底部，然后从悬崖面接收非常大的梯度。大梯度灾难性地将参数推到图的轴外。(右)使用梯度截断的梯度下降对悬崖的反应更温和。当它上升到悬崖面时，步长受到限制，使得它不会被推出靠近解的陡峭区域。经 Pascanu et al. (2013a) 许可改编此图。

使是采取简单的随机步骤往往工作得几乎一样好。如果爆炸非常严重，梯度数值上为 `Inf` 或 `Nan`（无穷大或不是一个数字），则可以采取大小为 v 的随机一步，通常会离开数值不稳定的状态。截断每小批量梯度范数不会改变单个小批量的梯度方向。然而，许多小批量使用范数截断梯度后的平均值不等同于截断真实梯度（使用所有的实例所形成的梯度）的范数。大导数范数的样本，和像这样的出现在同一小批量的样本，其对最终方向的贡献将消失。不像传统小批量梯度下降，其中真实梯度的方向是等于所有小批量梯度的平均。换句话说，传统的随机梯度下降使用梯度的无偏估计，而与使用范数截断的梯度下降引入了经验上是有用的启发式偏置。通过逐元素截断，更新的方向与真实梯度或小批量的梯度不再对齐，但是它仍然是一个下降方向。还有学者提出 (Graves, 2013)（相对于隐藏单元）截断反向传播梯度，但没有公布与这些变种之间的比较；我们推测，所有这些方法表现类似。

10.11.2 引导信息流的正则化

梯度截断有助于处理爆炸的梯度，但它无助于消失的梯度。为了解决消失的梯度问题并更好地捕获长期依赖，我们讨论了如下想法：在展开循环架构的计算图中，沿着与弧边相关联的梯度乘积接近 1 的部分创建路径。在第 10.10 节中已经讨论过，实现这一点的一种方法是使用 LSTM 以及其他自循环和门控机制。另一个想法是正则化或约束参数，以引导“信息流”。特别是即使损失函数只对序列尾部的输出作惩罚，我们也希望梯度向量 $\nabla_{\mathbf{h}^{(t)}} L$ 在反向传播时能维持其幅度。形式上，我们要使

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

与

$$\nabla_{\mathbf{h}^{(t)}} L \quad (10.51)$$

一样大。在这个目标下，Pascanu *et al.* (2013a) 提出以下正则项：

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

计算这一梯度的正则项可能会出现困难，但 Pascanu *et al.* (2013a) 提出可以将后向传播向量 $\nabla_{\mathbf{h}^{(t)}} L$ 考虑为恒值作为近似（为了计算正则化的目的，没有必要通过它们向后传播）。使用该正则项的实验表明，如果与标准的启发式截断（处理梯度爆炸）相结合，该正则项可以显著地增加 RNN 可以学习的依赖跨度。梯度截断特别重要，因为它保持了爆炸梯度边缘的 RNN 动态。如果没有梯度截断，梯度爆炸将阻碍学习的成功。

这种方法的一个主要弱点是，在处理数据冗余的任务时如语言模型，它并不像 LSTM 一样有效。

10.12 外显记忆

智能需要知识并且可以通过学习获取知识，这已促使大型深度架构的发展。然而，知识是不同的并且种类繁多。有些知识是隐含的、潜意识的并且难以用语言表达——比如怎么行走或狗与猫的样子有什么不同。其他知识可以是明确的、可陈述的以及可以相对简单地使用词语表达——每天常识性的知识，如“猫是一种动物”，

或者为实现自己当前目标所需知道的非常具体的事，如“与销售团队会议在 141 室于下午 3:00 开始”。

神经网络擅长存储隐性知识，但是他们很难记住事实。被存储在神经网络参数中之前，随机梯度下降需要多次提供相同的输入，即使如此，该输入也不会被特别精确地存储。Graves *et al.* (2014) 推测这是因为神经网络缺乏**工作存储** (working memory) 系统，即类似人类为实现一些目标而明确保存和操作相关信息片段的系统。这种外显记忆组件将使我们的系统不仅能够快速“故意”地存储和检索具体的事，也能利用他们循序推论。神经网络处理序列信息的需要，改变了每个步骤向网络注入输入的方式，长期以来推理能力被认为是重要的，而不是对输入做出自动的、直观的反应 (Hinton, 1990)。

为了解决这一难题，Weston *et al.* (2014) 引入了**记忆网络** (memory network)，其中包括一组可以通过寻址机制来访问的记忆单元。记忆网络原本需要监督信号指示他们如何使用自己的记忆单元。Graves *et al.* (2014) 引入的**神经网络图灵机** (neural Turing machine)，不需要明确的监督指示采取哪些行动而能学习从记忆单元读写任意内容，并通过使用基于内容的软注意机制（见 Bahdanau *et al.* (2015) 和第 12.4.5.1 节），允许端到端的训练。这种软寻址机制已成为其他允许基于梯度优化的模拟算法机制的相关架构的标准 (Sukhbaatar *et al.*, 2015; Joulin and Mikolov, 2015; Kumar *et al.*, 2015a; Vinyals *et al.*, 2015a; Grefenstette *et al.*, 2015)。

每个记忆单元可以被认为是 LSTM 和 GRU 中记忆单元的扩展。不同的是，网络输出一个内部状态来选择从哪个单元读取或写入，正如数字计算机读取或写入到特定地址的内存访问。

产生确切整数地址的函数很难优化。为了缓解这一问题，NTM 实际同时从多个记忆单元写入或读取。读取时，它们采取许多单元的加权平均值。写入时，他们对多个单元修改不同的数值。用于这些操作的系数被选择为集中在一小数目的单元，如通过 softmax 函数产生它们。使用这些具有非零导数的权重允许函数控制访问存储器，从而能使用梯度下降法优化。关于这些系数的梯度指示着其中每个参数是应该增加还是减少，但梯度通常只在接收大系数的存储器地址上变大。

这些记忆单元通常扩充为包含向量，而不是由 LSTM 或 GRU 存储单元所存储的单个标量。增加记忆单元大小的原因有两个。原因之一是，我们已经增加了访问记忆单元的成本。我们为产生用于许多单元的系数付出计算成本，但我们预期这些系数聚集在周围小数目的单元。通过读取向量值，而不是一个标量，我们可以抵

消部分成本。使用向量值的记忆单元的另一个原因是，它们允许基于内容的寻址 (content-based addressing)，其中从一个单元读或写的权重是该单元的函数。如果我们能够生产符合某些但并非所有元素的模式，向量值单元允许我们检索一个完整向量值的记忆。这类似于人们能够通过几个歌词回忆起一首歌曲的方式。我们可以认为基于内容的读取指令是说，“检索一首副歌歌词中带有‘我们都住在黄色潜水艇’的歌”。当我们要检索的对象很大时，基于内容的寻址更为有用——如果歌曲的每一个字母被存储在单独的记忆单元中，我们将无法通过这种方式找到他们。通过比较，基于位置的寻址 (location-based addressing) 不允许引用存储器的内容。我们可以认为基于位置的读取指令是说“检索 347 档的歌的歌词”。即使当存储单元很小时，基于位置的寻址通常也是完全合理的机制。

如果一个存储单元的内容在大多数时间步上会被复制（不被忘记），则它包含的信息可以在时间上向前传播，随时间向后传播的梯度也不会消失或爆炸。

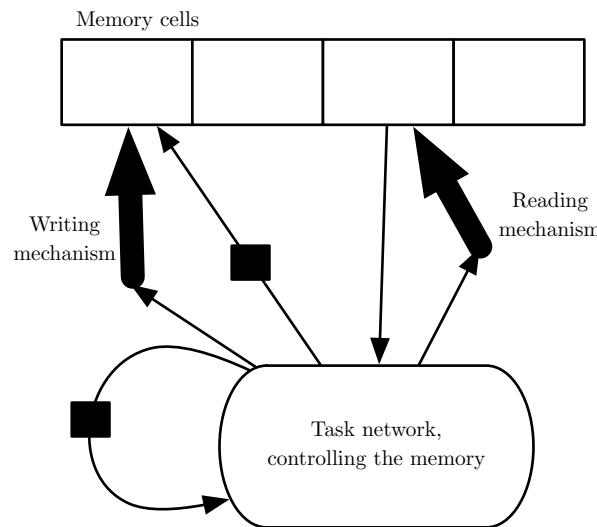


图 10.18：具有外显记忆网络的示意图，具备神经网络图灵机的一些关键设计元素。在此图中，我们将模型的“表示”部分（“任务网络”，这里是底部的循环网络）与存储事实的模型（记忆单元的集合）的“存储器”部分区分开。任务网络学习“控制”存储器，决定从哪读取以及在哪写入（通过读取和写入机制，由指向读取和写入地址的粗箭头指示）。

外显记忆的方法在图 10.18 说明，其中我们可以看到与存储器耦接的“任务神经网络”。虽然这一任务神经网络可以是前馈或循环的，但整个系统是一个循环网络。任务网络可以选择读取或写入的特定内存地址。外显记忆似乎允许模型学习普