第七章　循环神经网络

前馈神经网络假设每次输入都是独立的，也就是说每次网络的输出只依赖于当前的输入。但是在很多现实任务中，不同时刻的输入可以相互影响，比如视频、语音、文本等序列结构数据。某个时刻的输出可能和前面时刻的输入相关。此外，这些序列结构数据的长度一般是不固定的。而前馈神经网络要求输入和输出的维数都是固定的，不能任意改变。当处理序列数据时，前馈神经网络就无能力为了。

循环神经网络（Recurrent Neural Network，RNN），也叫**递归神经网络**。这里为了区别与另外一种递归神经网络（Recursive Neural Network），我们称为**循环神经网络**。在前馈神经网络模型中，连接存在层与层之间，每层的节点之间是无连接的。

循环神经网络通过使用带自反馈的神经元，能够处理任意长度的序列。循环神经网络比前馈神经网络更加符合生物神经网络的结构。循环神经网络已经被广泛应用在语音识别、语言模型以及自然语言生成等任务上。

给定一个输入序列$\mathbf{x}_{1:T} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t, \dots, \mathbf{x}_T)$，循环神经网络通过下面公式更新带反馈边的隐藏层的**活性值**$\mathbf{h}_t$：

$$\mathbf{h}_t = \begin{cases} 0 & t = 0 \\ f(\mathbf{h}_{t-1}, \mathbf{x}_t) & \text{otherwise} \end{cases} \tag{7.1}$$

从数学上讲，公式7.1可以看成一个动态系统。**动态系统**是指系统的状态按照一定的规律随时间变化的系统。因此，活性值$\mathbf{h}_t$在很多文献上也称为**状态**或**隐状态**。但这里的状态是数学上的概念，区别与我们在前馈网络中定义的神经元的状态。理论上循环神经网络可以近似任意的动态系统。图7.1给出了循环神经网络的示例。

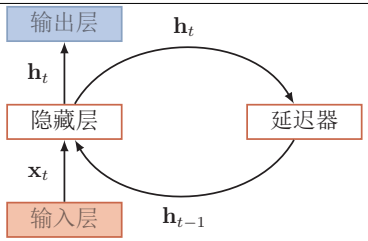


图 7.1: 循环神经网络

循环神经网络的参数训练可以通过**时序反向传播**（Backpropagation Through Time，BPTT）算法 [Werbos, 1990] 来学习。时序反向传播即按照时间的逆序将错误信息一步步地往前传递。这样，当输入序列比较长时，会存在梯度爆炸和消失问题Bengio et al. [1994], Hochreiter and Schmidhuber [1997], Hochreiter et al. [2001]，也称为长期依赖问题。为了解决这个问题，人们对循环神经网络进行了很多的改进，其中最有效的一个改进版本是**长短时记忆神经网络**（long short term memory neural network，LSTM）[Hochreiter and Schmidhuber, 1997]。

在本章中，我们先介绍循环神经网络的基本定义以及梯度计算，然后介绍两种扩展模型：长短时记忆神经网络和门限循环单元网络。

7.1 简单循环网络

我们先来看一个非常简单的循环神经网络，叫**简单循环网络**（Simple Recurrent Network，SRN）[Elman, 1990]。

假设时刻 t 时，输入为 $\mathbf{x}_t$，隐层状态（隐层神经元活性）为 $\mathbf{h}_t$。$\mathbf{h}_t$ 不仅和当前时刻的输入相关，也和上一个时刻的隐层状态相关。

一般我们使用如下函数：

$$\mathbf{h}_t = f(U\mathbf{h}_{t-1} + W\mathbf{x}_t + \mathbf{b}), \tag{7.2}$$

这里，f 是非线性函数，通常为 logistic 函数或 tanh 函数。

图7.10给出了按时间展开的循环神经网络。如果我们把每个时刻的状态都看作是前馈神经网络的一层的话，循环神经网络可以看作是在时间维度上权值共享的神经网络。

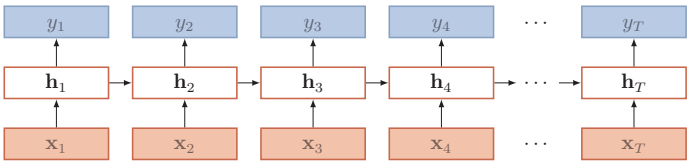


图 7.2: 按时间展开的循环神经网络

7.1.1 参数学习

循环神经网络的参数学习可以通过**随时间进行反向传播**（Backpropagation Through Time，BPTT）算法 [Werbos, 1990]。图7.3给出了随时间进行反向传播算法的示例。

以随机梯度下降为例，给定一个训练样本$(\mathbf{x}, \mathbf{y})$，其中$\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$为长度是$T$的输入序列，$\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t, \ldots, \mathbf{y}_T)$是长度为$T$的标签序列。即在每个时刻$t$，都有一个监督信息$\mathbf{y}_t$，我们定义时刻$t$的损失函数为

$$\mathcal{L}_t = \mathcal{L}(\mathbf{y}, f(\mathbf{h}_{t-1}, \mathbf{x}_t)), \tag{7.3}$$

这里$\mathcal{L}$为可微分的损失函数，比如交叉熵。那么整个序列上损失函数为

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}_t. \tag{7.4}$$

整个序列的损失函数$\mathcal{L}$关于参数U的梯度为：

需要重新表述。

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial U} \tag{7.5}$$

$$= \sum_{t=1}^{T} \frac{\partial \mathbf{h}_t}{\partial U} \frac{\partial \mathcal{L}_t}{\partial \mathbf{h}_t}. \tag{7.6}$$

根据公式7.2，

$$\frac{\partial \mathbf{h}_t}{\partial U} = \mathbf{h}_{t-1} + \mathbf{U}^\top \frac{\partial \mathbf{h}_{t-1}}{\partial U}. \tag{7.7}$$

其中，$\mathbf{h}_t$是关于U和$\mathbf{h}_{t-1}$的函数，而$\mathbf{h}_{t-1}$又是关于U和$\mathbf{h}_{t-2}$的函数。因此，我们可以用链式法则得到

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathbf{h}_k}{\partial U} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{y}_t}, \tag{7.8}$$

其中，

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \tag{7.9}$$

$$= \prod_{i=k+1}^{t} U^T \, \mathbf{diag}[f'(U\mathbf{h}_{i-1} + W\mathbf{x}_i + \mathbf{b})]. \tag{7.10}$$

令$\mathbf{z}_i = U\mathbf{h}_{i-1} + W\mathbf{x}_i + \mathbf{b}$，我们可以得到

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{t=1}^{T}\sum_{k=1}^{t} \frac{\partial \mathbf{h}_k}{\partial U}\left(\prod_{i=k+1}^{t} U^T \, \mathbf{diag}(f'(\mathbf{z}_i))\right)\frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t}\frac{\partial \mathcal{L}_t}{\partial \mathbf{y}_t}. \tag{7.11}$$

我们定义$\gamma = \|U^T \, \mathbf{diag}(f'(\mathbf{z}_i))\|$，则在上面公式中的括号里面为$\gamma^{t-k}$。如果$\gamma > 1$，当$t-k \to \infty$时，$\gamma^{t-k} \to \infty$，会造成系统不稳定，也就是所谓的**梯度爆炸**问题；相反，如果$\gamma < 1$，当$t-k \to \infty$时，$\gamma^{t-k} \to 0$，会出现和深度前馈神经网络类似的**梯度消失**问题。

在训练循环神经网络时，更经常出现的是梯度消失问题。因为我们一般情况下使用的非线性激活函数为logistic函数或tanh函数，其导数值都小于1。而权重矩阵$\|U^T\|$也不会太大。我们定义$\|U^T\| \leq \gamma_u \leq 1$，$\|\mathbf{diag}[f'(\mathbf{z}_i)]\| \leq \gamma_f \leq 1$，则有

$$\|\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}}\| \leq \|U^T\|\|\mathbf{diag}[f'(\mathbf{z}_i)]\| \leq \gamma_u\gamma_f \leq 1. \tag{7.12}$$

经过$t-k$次传播之后，

$$\|\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}\| \leq (\gamma_u\gamma_f)^{t-k}. \tag{7.13}$$

如果时间间隔$t-k$过大，$\|\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}\|$会趋向于0。

因此，虽然简单循环网络从理论上可以建立长时间间隔的状态之间的依赖关系（Long-Term Dependencies），但是由于梯度爆炸或消失问题，实际上只能学习到短周期的依赖关系。这就是所谓的**长期依赖问题**。

7.1.2 改进方案

为了避免梯度爆炸或消失问题，就要尽量使得$U^T \, \mathbf{diag}(f'(\mathbf{z}_i)) = 1$。一种方式就是选取合适的参数，同时使用非饱和的激活函数。但这样的方式需要很多人工经验，同时限制了模型的广泛应用。

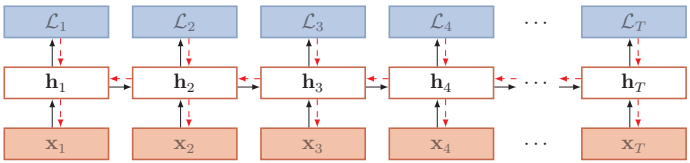


图 7.3: 按时间展开的循环神经网络

还有一种方式就是改变模型，比如让$U=1$，同时使用$f'(\mathbf{z}_i)=1$。

$$\mathbf{h}_t = \mathbf{h}_{t-1} + W\mathbf{g}(\mathbf{x}_t), \tag{7.14}$$

$\mathbf{g}$是非线性激活函数。

但这样的形式，丢失了神经元在反馈边上的非线性激活的性质。因此，一个更加有效的改进是引入一个新的状态$\mathbf{c}_t$专门来进行线性的反馈传递，同时在$\mathbf{c}_t$的信息非线性传递给$\mathbf{h}_t$。

$$\mathbf{c}_t = \mathbf{c}_{t-1} + W\mathbf{g}(\mathbf{x}_t), \tag{7.15}$$

$$\mathbf{h}_t = \tanh(\mathbf{c}_t). \tag{7.16}$$

但是，这样依然存在一定的问题。因为$\mathbf{c}_t$和$\mathbf{c}_{t-1}$是线性关系，同时不断累积$\mathbf{x}_t$的信息，会使得$\mathbf{c}_t$变得越来越大。为了解决这个问题，Hochreiter and Schmidhuber [1997]提出一个非常好的解决方案，就是引入**门机制**（Gating Mechanism）来控制信息的累积速度，并可以选择遗忘之前累积的信息。这就是下面要介绍的**长短时记忆神经网络**。

7.2 长短时记忆神经网络：LSTM

长短时记忆神经网络（Long Short-Term Memory Neural Network，LSTM）[Hochreiter and Schmidhuber, 1997] 是循环神经网络的一个变体，可以有效地解决简单循环神经网络的梯度爆炸或消失问题。

LSTM 模型的关键是引入了一个内部记忆单元（Memory Units）或内部状态，来保存历史信息。然后动态地让网络学习何时遗忘历史信息，何时用新信

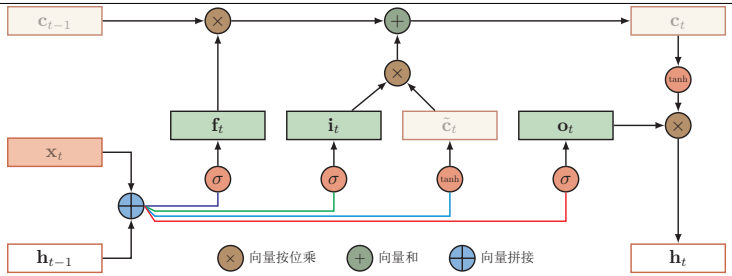


图 7.4: LSTM 网络结构示例

息更新记忆单元。在时刻 t 时，内部记忆单元 $\mathbf{c}_t$ 记录了到当前时刻为止的所有历史信息，并受三个“门”控制：输入门 $\mathbf{i}_t$，遗忘门 $\mathbf{f}_t$ 和输出门 $\mathbf{o}_t$，三个门的元素的值在 $[0, 1]$ 之间。0 代表关闭状态，不许任何信息通过；1 代表开放状态，允许所有信息通过。

在时刻 t 时 LSTM 的更新方式如下：

$$\mathbf{i}_t = \sigma(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + \mathbf{b}_i), \tag{7.17}$$

$$\mathbf{f}_t = \sigma(W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1} + \mathbf{b}_f), \tag{7.18}$$

$$\mathbf{o}_t = \sigma(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + \mathbf{b}_o), \tag{7.19}$$

$$\tilde{\mathbf{c}}_t = \tanh(W_c\mathbf{x}_t + U_c\mathbf{h}_{t-1} + \mathbf{b}_c), \tag{7.20}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \tag{7.21}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{7.22}$$

这里，$\mathbf{x}_t$ 是当前时刻的输入向量，σ 是 logistic sigmoid 函数，W，U 是参数矩阵，$\mathbf{b}$ 是偏置向量。$\odot$ 为 Hadamard 积。

在每个时刻 t，LSTM 模型首先根据输入 $\mathbf{x}_t$ 以及上一个时刻的状态 $\mathbf{h}_{t-1}$ 分别计算三个门向量：输入门 $\mathbf{i}_t$，遗忘门 $\mathbf{f}_t$ 和输出门 $\mathbf{o}_t$，并且计算一个候选的记忆向量

遗忘门 $\mathbf{f}_t$ 控制每一个内存单元需要遗忘多少信息，输入门 $\mathbf{i}_t$ 控制每一个内存单元加入多少新的信息，输出门 $\mathbf{o}_t$ 控制每一个内存单元输出多少信息。

图7.4给出了 LSTM 模型的计算结构。

这样，LSTM 可以学习到长周期的历史信息。LSTM 已经被应用到很多的任务中，比如机器翻译 [Sutskever et al., 2014] 等。

一般在深度网络参数学习时，参数初始化的值一般都比较小。但是在训练LSTM时，过小的值会使得遗忘门的值比较小。这意味着前一时刻的信息大部分都丢失了，这样网络很难捕捉到长距离的依赖信息。并且相邻时间间隔的梯度会非常小，这会导致梯度弥散问题。因此遗忘的参数初始值一般都设得比较大，其偏置向量 $\mathbf{b}_f$ 设为 1 或 2.

7.2.1 LSTM 的各种变体

最早的LSTM模型是没有遗忘门的。

$$\mathbf{c}_t = \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t. \tag{7.23}$$

如之前的分析，内部状态 $\mathbf{c}$ 会不断增大。当输入数据的长度很大时，会导致内部记忆单元的容量不够，从而大大降低LSTM模型的性能。

peephole 连接

在计算当前时刻的三个门 $\mathbf{i}_t$，$\mathbf{f}_t$ 和 $\mathbf{o}_t$ 时，我们可以让其也和上一个时刻的记忆相关。

$$\mathbf{i}_t = \sigma(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + V_i\mathbf{c}_{t-1} + \mathbf{b}_i), \tag{7.24}$$
$$\mathbf{f}_t = \sigma(W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1} + V_f\mathbf{c}_{t-1} + \mathbf{b}_f), \tag{7.25}$$
$$\mathbf{o}_t = \sigma(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + V_o\mathbf{c}_t + \mathbf{b}_o), \tag{7.26}$$

这里，V_i，V_f 和 V_o 都为对角阵。

耦合输入门和遗忘门

在LSTM中，输入门和遗忘门是互补关系，因为同时用两个门比较冗余。因此，可以将这两门合并为一个门，让

$$\mathbf{f}_t + \mathbf{i}_t = \mathbf{1}. \tag{7.27}$$

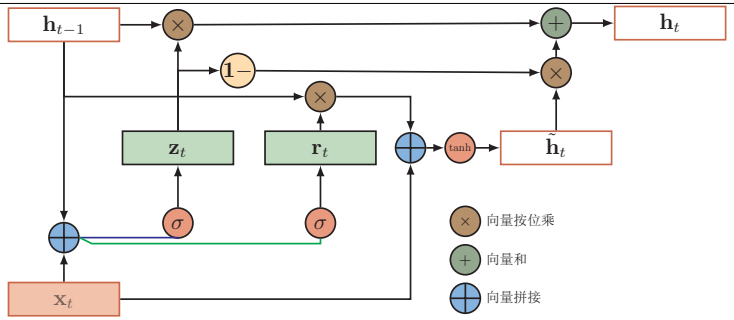


图 7.5: GRU 网络结构示例

7.2.2 门限循环单元神经网络：GRU

门限循环单元（Gated Recurrent Unit，GRU）神经网络 [Cho et al., 2014, Chung et al., 2014] 是一种比 LSTM 更加简化的版本。在 LSTM 中，输入门和遗忘门是互补关系，因为同时用两个门比较冗余。GRU 将输入门与和遗忘门合并成一个门：更新门（Update Gate），同时还合并了内部记忆状态和输出状态。

GRU 模型中有两个门：更新门（update gate）$\mathbf{z}$ 和重置门（reset gate）$\mathbf{r}$。

- 更新门 $\mathbf{z}$ 用来控制当前的状态需要遗忘多少历史信息和接受多少新信息。
- 重置门 $\mathbf{r}$ 用来控制候选状态中有多少信息是从历史信息中得到。

GRU 模型的更新方式如下：

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r), \tag{7.28}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z), \tag{7.29}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}(\mathbf{r}_t \odot \mathbf{h}_{t-1})), \tag{7.30}$$

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t, \tag{7.31}$$

这里选择 tanh 函数也是因为其导数有更大的值域。

图7.5给出了 GRU 模型的计算结构。

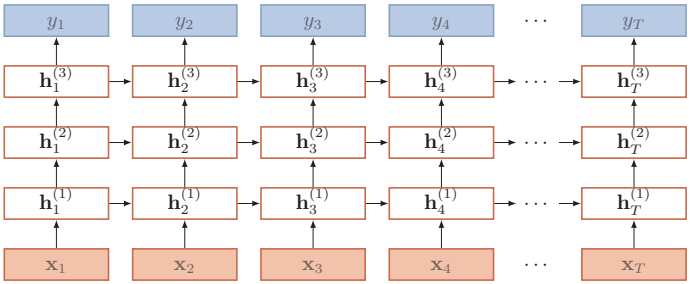


图 7.6: 按时间展开的堆叠循环神经网络

7.3 深层循环神经网络

循环神经网络的深度是一个有争议的话题。一方面来说，如果我们把循环网络按时间展开，不同时刻的状态之间存在非线性连接，循环网络已经是一个非常深的网络了。从另一方面来说，这个网络是非常浅的。任意两个相邻时刻的隐藏状态（$\mathbf{h}_{t-1} \rightarrow \mathbf{h}_t$），隐藏状态到输出（$\mathbf{h}_t \rightarrow \mathbf{y}_t$），以及输入到隐藏状态之间（$\mathbf{x}_t \rightarrow \mathbf{h}_t$）之间的转换只有一个非线性函数。

既然增加深度可以极大地增强前馈神经网络的能力，那么如何增加循环神经网络的深度呢？一种常见的做法是将多个循环网络堆叠起来，称为**堆叠循环神经网络**（Stacked Recurrent Neural Network，SRNN）。

第 l 层网络的输入是第 $l-1$ 层网络的输出。我们定义 $\mathbf{h}_t^{(l)}$ 为在时刻 t 时第 l 层的隐藏状态，定义为

$$\mathbf{h}_t^{(l)} = f(\mathbf{U}^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{W}^{(l)}\mathbf{h}_t^{(l-1)} + \mathbf{b}^{(l)}), \tag{7.32}$$

其中 $\mathbf{U}^{(l)}$，$\mathbf{W}^{(l)}$ 和 $\mathbf{b}^{(l)}$ 为权重矩阵和偏置向量。当 $l=1$ 时，$\mathbf{h}_t^{(0)} = \mathbf{x}_t$。

图7.6给出了按时间展开的堆叠循环神经网络。

7.4 双向循环神经网络

在有些任务中，一个时刻的输出不但和过去时刻的信息有关，也和后续时刻的信息有关。比如给定一个句子，其中一个词的词性由它的上下文决定，即

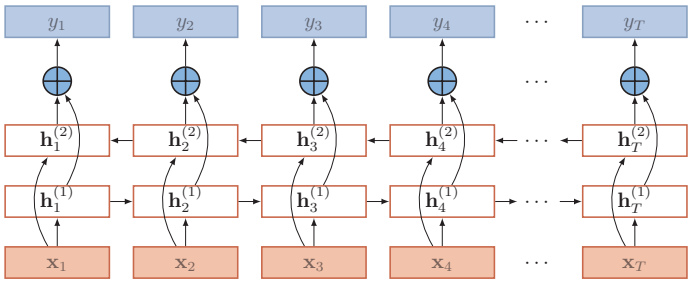


图 7.7: 按时间展开的双向循环神经网络

包含左右两边的信息。因此，在这些任务中，我们可以增加一个按照时间的逆序来传递信息的网络层，来增强网络的能力。

双向循环神经网络（Bidirectional recurrent neural networks，BRNN）由两层循环神经网络组成，它们的输入相同，只是信息传递的方向不同。

假设第 1 层按时间顺序，第 2 层按时间逆序，在时刻 t 时的隐藏状态定义为 $\mathbf{h}_t^{(1)}$ 和 $\mathbf{h}_t^{(2)}$，则

$$\mathbf{h}_t^{(1)} = f(\mathbf{U}^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}_t + \mathbf{b}^{(1)}), \tag{7.33}$$

$$\mathbf{h}_t^{(2)} = f(\mathbf{U}^{(2)}\mathbf{h}_{t+1}^{(2)} + \mathbf{W}^{(2)}\mathbf{x}_t + \mathbf{b}^{(2)}), \tag{7.34}$$

$$\mathbf{h}_t = \mathbf{h}_t^{(1)} \oplus \mathbf{h}_t^{(2)}, \tag{7.35}$$

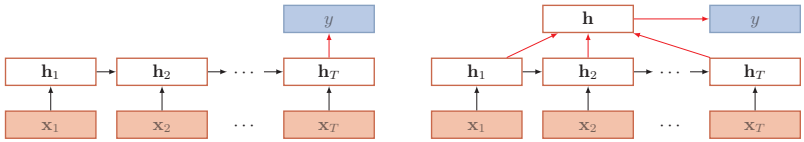
其中 $\oplus$ 为向量拼接操作。

图7.7给出了按时间展开的双向循环神经网络。

7.5 应用

循环神经网络可以应用在很多任务中。根据这个任务的特点可以分为以下几种模式：

- 序列到类别：这类任务的输入为序列，输出为类别。比如在文本分类中，输入数据为单词的序列，输出为该文本的类别。
- 序列到序列：这类任务的输入和输出都为序列。具体又可以分为两种情况：一种是输入和输出同步，即每一时刻都有输入和输出。比如在序列标注问



(a) 正常模式 (b) 按时间进行平均采样模式

图 7.8: 序列到类别的应用模式

题，每个时刻的输入都需要有一个输出。输入序列和输出序列的长度相同。另一种是输入和输出不需要有严格的对应关系。比如在机器翻译中，输入为源语言的单词序列，输出为目标语言的单词序列。输入和输出序列并不需要保持相同的长度。

下面我们分别来看下这几种应用模式。

7.5.1 序列到类别

首先，我们来看下序列到类别的应用模式。假设一个样本 $\mathbf{x} = (\mathbf{x}_1, \cdots, \mathbf{x}_T)$ 为一个长度为 T 的序列，输入为一个类别 $y \in \{1, \cdots, C\}$。我们可以将样本 $\mathbf{x}$ 按不同时刻输入到循环神经网络中，并得到不同时刻的隐藏状态 $\mathbf{h}_1, \cdots, \mathbf{h}_T$。我们可以将 $\mathbf{h}_T$ 看作整个序列的最终表示，并输入给分类器 g，

$$y = g(\mathbf{h}_T), \tag{7.36}$$

这里 g 可以是简单的分类器（比如 Logistic 回归）或复杂的分类器（比如多层感知器）。

除了将最后时刻的隐藏状态作为序列表示之外，我们还可以对整个序列的所有隐藏状态进行平均，并用这个平均状态来作为整个序列的表示。

$$y = g(\frac{1}{T}\sum_{t=1}^{T}\mathbf{h}_t). \tag{7.37}$$

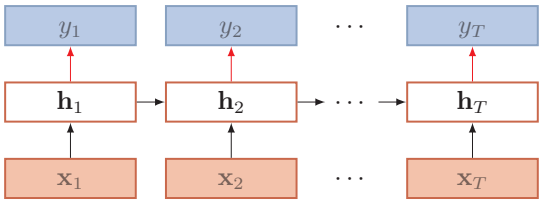


图 7.9: 同步的序列到序列模式

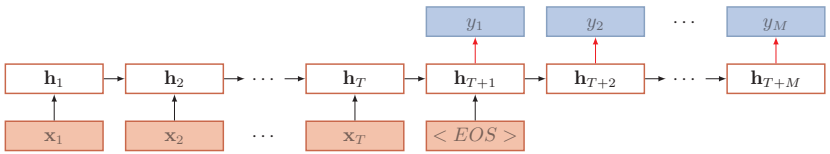


图 7.10: 异步的序列到序列模式

7.5.2 同步的序列到序列模式

7.5.3 异步的序列到序列模式

7.6 Hopfield 网络

Hopfield [1982]

7.7 总结和深入阅读

为了使得前馈神经网络能处理变长的序列数据，一种方法是使用延时神经网络（Time-Delay Neural Networks，TDNN）[?]。

参考文献

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

Kyunghyun Cho, Bart

Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79 (8):2554–2558, 1982.

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10): 1550–1560, 1990.