# FDU 神经网络 5. 循环神经网络

本文参考以下教材:

- 神经网络与深度学习 (邱锡鹏) 第 $6, 8, 15$ 章
- Deep Learning (I. Goodfellow, Y. Bengio, A. Courville) Chapter $10$
- 深度学习 (I. Goodfellow, Y. Bengio, A. Courville, 赵申剑等译) 第 $10$ 章

欢迎批评指正!

## 5.1 记忆

在前馈神经网络中，信息的传递是单向的，
这种限制虽然使得网络变得更容易学习，但在一定程度上也减弱了神经网络模型的能力.
在生物神经网络中，神经元之间的连接关系要复杂得多.
前馈神经网络可以看作一个复杂的函数，每次输入都是独立的，即网络的输出只依赖于当前的输入.
但是在很多现实任务中，网络的输出不仅和当前时刻的输入相关，也和其过去一段时间的输出相关.
例如一个有限状态自动机，其下一个时刻的状态不仅仅和当前输入相关，也和当前状态 (上一个时刻的输出) 相关.
此外，前馈网络难以处理时序数据，比如视频、语音、文本等.
时序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.
因此当处理这一类和时序数据相关的问题时，就需要一种能力更强的模型.

**循环神经网络** (Recurrent Neural Network, RNN) 是一类具有短期记忆能力的神经网络.
在循环神经网络中，神经元不但可以接受其他神经元的信息，
也可以接受自身的信息，形成具有环路的网络结构.
和前馈神经网络相比，循环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.

循环神经网络的参数学习可以通过**随时间反向传播算法**来学习.
随时间反向传播算法即按照时间的逆序将梯度一步步地往前传递.
当输入序列比较长时，会存在梯度爆炸和消失问题，也称为长程依赖问题.
为解决这个问题，人们对循环神经网络进行了很多的改进，
其中最有效的改进方式引入**门控机制** (gating mechanism).
此外，循环神经网络可以很容易地扩展到两种更广义的记忆网络模型: **递归神经网络**和**图网络**.

### 5.1.1 延时神经网络

一种简单的利用历史信息的方法是建立一个额外的延时单元，
用来存储网络的历史信息 (可以包括输入、输出、隐状态等).
比较有代表性的模型是**延时神经网络** (Time Delay Neural Network, TDNN).

延时神经网络是在前馈网络中的非输出层都添加一个延时器，记录神经元的最近几次活性值.
在第 $t$ 个时刻，第 $l$ 层神经元的活性值依赖于第 $l-1$ 层神经元的最近 $K$ 个时刻的活性值，即:

$$h_t^{(l)} = f\left(h_t^{(l-1)}, \dots, h_{t-K}^{(l-1)}\right)$$

其中 $h_t^{(l)} \in \mathbb{R}^{M_l}$ 表示第 $l$ 层神经元在时刻 $t$ 的活性值，$M_l$ 为第 $l$ 层神经元的数量.
通过延时器，前馈网络就具有了短期记忆的能力.

延时神经网络在时间维度上共享权值，以降低参数数量.
因此对于序列输入来讲，延时神经网络就相当于卷积神经网络.

## 5.1.2 有外部输入的非线性自回归模型

**自回归模型** (Auto Regressive Model, AR) 是统计学上常用的一类时间序列模型，
用一个变量 $y_t$ 的历史信息来预测自己:

$$y_t = w_0 + \sum_{k=1}^{K} w_k y_{t-k} + \varepsilon_t$$

其中 $K$ 是超参数，$w_0, \ldots, w_K$ 为可学习的参数，
$\varepsilon_t \sim N(0, \sigma^2)$ 为第 $t$ 个时刻的噪声，其方差 $\sigma^2$ 与时间无关.

**有外部输入的非线性自回归模型** (Nonlinear Auto Regressive with Exogenous Inputs Model, NARX)
是自回归模型的扩展，在每个时刻 $t$ 都有一个外部输入 $x_t$，产生一个输出 $y_t$.
$\mathrm{NARX}$ 通过一个延时器记录最近 $K_x$ 次的外部输入和最近 $K_y$ 次的输出.
其第 $t$ 个时刻的输出 $y_t$ 为:

$$y_t = f(x_t, \ldots, x_{t-K_x}, y_{t-1}, \ldots, y_{t-K_y})$$

其中 $f(\cdot)$ 是某个非线性函数 (可以是一个前馈神经网络)，$K_x$ 和 $K_y$ 为超参数.

## 5.1.3 循环神经网络

**循环神经网络** (Recurrent Neural Network, RNN)
通过使用带自反馈的神经元，能够处理任意长度的时序数据.
给定输入序列 $x = (x_1, \ldots, x_t, \ldots, x_T)$
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 $h_t$:

$$h_t = f(h_{t-1}, x_t)$$

其中 $h_0 = 0$，而 $f(\cdot)$ 是某个非线性函数 (可以是一个前馈神经网络)

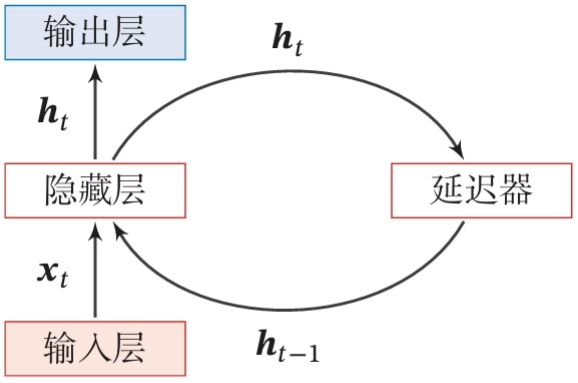



图 6.1　循环神经网络

循环神经网络可看成一个动力系统，即其状态是关于时间的函数.
因此隐藏层的活性值 $h_t$ 也称为**状态** (state).
理论上，循环神经网络可以近似任意的非线性动力系统.
前馈神经网络可以模拟任何连续函数，而循环神经网络可以模拟任何程序.

---

考虑一个完全连接的循环神经网络，其输入为 $x_t$，输出为 $y_t$:

$$h_t = f(Uh_{t-1} + Wx_t + b)$$
$$y_t = Vh_t$$

**(循环神经网络的通用近似定理, 神经网络与深度学习, 定理 6.1)**
如果一个完全连续的循环神经网络有足够数量的 Sigmoid 型隐藏神经元，
则它可以任意精度近似任何一个非线性动力系统:

$$s_t = g(s_{t-1}, x_t)$$
$$y_t = o(s_t)$$

其中 $s_t$ 为每个时刻的隐状态，$x_t$ 是外部输入，
$g(\cdot)$ 是可测的状态转换函数，$o(\cdot)$ 是连续输出函数，并且对状态空间的紧致性没有限制.

# 5.2 循环神经网络

## 5.2.1 一个简单的例子

**简单循环网络** (Simple Recurrent Network, SRN) 是一个非常简单的循环神经网络，只有一个隐藏层.
记 $x_t \in \mathbb{R}^M$ 为网络在时刻 $t$ 的输入，$h_t \in \mathbb{R}^D$ 表示隐藏层状态 (即隐藏层神经元活性值)，
则 $h_t$ 不仅和当前时刻的输入 $x_t$ 相关，也和上一个时刻的隐藏层状态 $h_{t-1}$ 相关.
简单循环网络在时刻 $t$ 的更新公式为:

$$z_t = Uh_{t-1} + Wx_t + b$$
$$h_t = f(z_t)$$

其中 $z_t \in \mathbb{R}^D$ 是隐藏层的净输入，$U \in \mathbb{R}^{D\times D}$ 为状态-状态权重矩阵，
$W \in \mathbb{R}^{D\times M}$ 为状态-输入权重矩阵，$b \in \mathbb{R}^D$ 为偏置向量，
$f(\cdot)$ 是非线性激活函数，通常为 Sigmoid 函数或 $\tanh$ 函数.

如果我们把每个时刻的状态都看作前馈神经网络的一层，
循环神经网络可以看作在时间维度上权值共享的神经网络.
下图给出了按时间展开的循环神经网络:

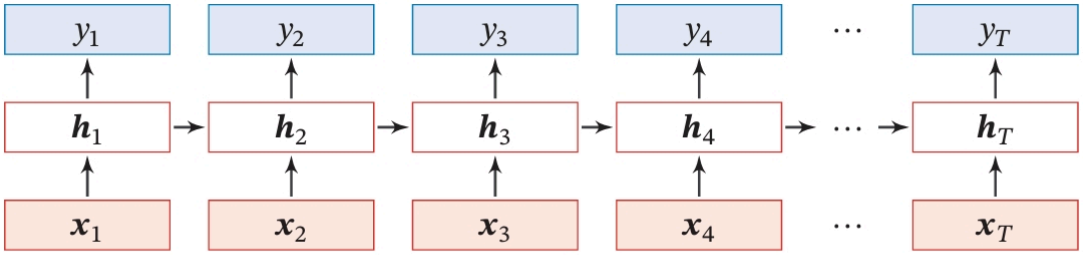



图 6.2　按时间展开的循环神经网络

## 5.2.2 应用模式

### (1) 序列到类别

**序列到类别模式**主要用于序列数据的分类问题: 输入为序列，输出为类别.
例如在文本分类中，输入数据为单词的序列，输出为该文本的类别.

假设一个样本 $x = (x_1, \ldots, x_T)$ 为一个长度为 $T$ 的序列，输出为类别标签 $y \in \{1, \ldots, C\}$.
我们可以将样本 $x$ 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 $h_1, \ldots, h_T$.

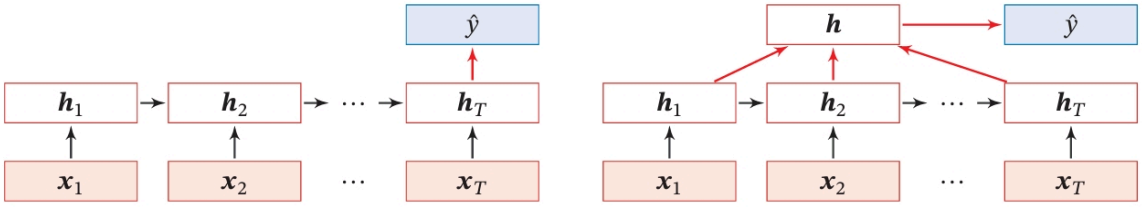
- ① 我们可以将 $h_T$ 看作整个序列的最终表示，并输入给分类器 $g(\cdot)$ 进行分类:

$$\hat{y} = g(h_T)$$

- ② 我们还可以对整个序列的所有状态进行平均，
  并用这个平均状态来作为整个序列的最终表示，输入给分类器 $g(\cdot)$ 进行分类:

$$\hat{y} = g\left(\frac{1}{T}\sum_{t=1}^{T} h_t\right)$$

其中 $g(\cdot)$ 可以是简单的线性分类器 (如 Logistic 回归) 或复杂的分类器 (如多层前馈神经网络).




(a) 正常模式　　(b) 按时间进行平均采样模式

图 6.3　序列到类别模式

### (2) 同步序列到序列

**同步序列到序列模式**主要用于序列标注 (sequence labeling) 任务，
即每一时刻都有输入和输出，输入序列和输出序列的长度相同.
例如在词性标注 (part-of-speech tagging) 中，每一个单词都需要标注其对应的词性标签.

在同步序列到序列模式中，输入为序列 $x = (x_1, \ldots, x_T)$，输出为序列 $y = (y_1, \ldots, y_T)$.
样本 $x$ 按不同时刻输入到循环神经网络中，并得到不同时刻的隐状态 $h_1, \ldots, h_T$.
每个时刻的隐状态 $h_t$ 代表了当前时刻和历史的信息，并输入给分类器 $g(\cdot)$ 得到当前时刻的标签 $y_t$:

$$\hat{y}_t = g(h_t) \quad (t = 1, \ldots, T)$$

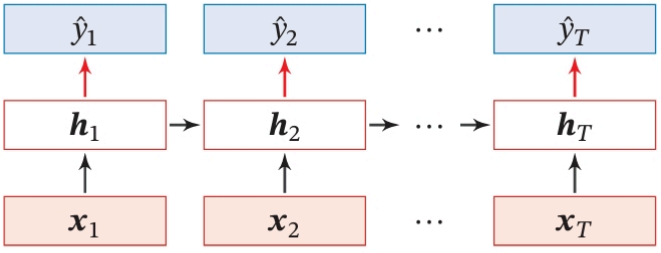



图 6.4　同步的序列到序列模式

### (3) 异步序列到序列

**异步序列到序列模式**也称为编码器-解码器 (encoder-decoder) 模型，
即输入序列和输出序列不需要有严格的对应关系，也不需要保持相同的长度.
例如在机器翻译中，输入为源语言的单词序列，输出为目标语言的单词序列.

在异步序列到序列模式中，
输入为长度为 $T$ 的序列 $x = (x_1, \ldots, x_T)$，输出为长度为 $M$ 的序列 $y = (y_1, \ldots, y_M)$.
这一般通过先编码后解码的方式来实现:
先将样本 $x$ 按不同时刻输入到一个循环神经网络 (编码器) 中，并得到其编码 $h_T$，
再使用另一个循环神经网络 (解码器)，得到输出序列 $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_M)$.
为了建立输出序列之间的依赖关系，在解码器中通常使用非线性的自回归模型.
记 $f_1(\cdot)$ 和 $f_2(\cdot)$ 为用作编码器和解码器的循环神经网络，则编码器-解码器模型可以写为:

$$
\begin{aligned}
h_t &= f_1(h_{t-1}, x_t) \quad (t = 1, \ldots, T) \\
h_{T+t} &= f_2(h_{T+t-1}, \hat{y}_{t-1}) \quad (t = 1, \ldots, M) \\
\hat{y}_t &= g(h_{T+t}) \quad (t = 1, \ldots, M)
\end{aligned}
$$

其中 $g(\cdot)$ 为分类器.
解码器通常采用自回归模型，每个时刻的输入为上一时刻的预测结果 $\hat{y}_{t-1}$.

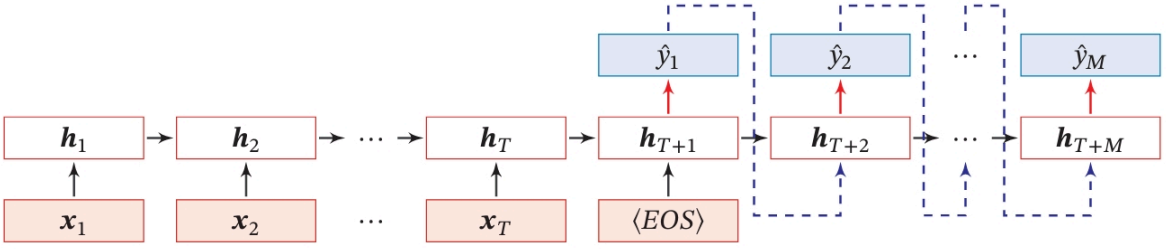



图 6.5　异步的序列到序列模式

## 5.2.3 参数学习

不失一般性，我们考虑同步序列到序列的循环神经网络:

$$
\begin{aligned}
z_t &= U h_{t-1} + W x_t + b \\
h_t &= f(z_t) \\
\hat{y}_t &= g(h_t)
\end{aligned}
$$

我们定义时刻 $t$ 的损失函数为:

$$
\mathcal{L}_t := \mathcal{L}(y_t, \hat{y}_t) = \mathcal{L}(y_t, g(h_t))
$$

其中 $\mathcal{L}$ 是可微的损失函数 (例如交叉熵损失函数)，
则从输入序列 $x = (x_1, \ldots, x_T)$ 到输出序列 $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_T)$ 的损失函数为:

$$
\mathcal{L} := \sum_{t=1}^{T} \mathcal{L}_t
$$

循环神经网络中存在一个递归调用的函数 $f(\cdot)$，
因此其计算参数梯度的方式和前馈神经网络不太相同.
在循环神经网络中主要有两种计算梯度的方式:
随时间反向传播 (BPTT) 算法和实时循环学习 (RTRL) 算法.

BPTT 算法和 RTRL 算法都是基于梯度下降的算法，分别通过前向模式和反向模式应用链式法则来计算梯度.
在循环神经网络中，一般网络输出维度远低于输入维度，因此 BPTT 算法的计算量会更小，
但是 BPTT 算法需要保存所有时刻的中间梯度，空间复杂度较高.
RTRL 算法不需要梯度回传，因此非常适合用于需要在线学习或无限序列的任务中.

### (1) 随时间反向传播算法

**随时间反向传播** (Back Propagation Through Time, BPTT) 算法
通过类似前馈神经网络的误差反向传播算法来计算梯度.

BPTT 算法将循环神经网络看作一个展开的多层前馈网络,
其中 "每一层" 对应循环网络中的 "每个时刻".

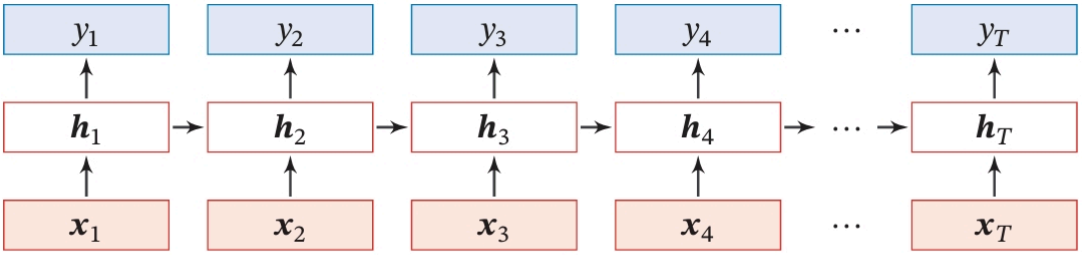



图 6.2 按时间展开的循环神经网络

这样循环神经网络就可以按照前馈网络中的反向传播算法计算参数梯度.
在 "展开" 的前馈网络中，所有层的参数是共享的,
因此参数的真实梯度是所有 "展开层" 的参数梯度之和:

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

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial W}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial b}$$

时刻 $t$ 的损失 $\mathcal{L}_t$ 对时刻 $k$ 的隐藏神经层的净输入 $z_k$ $(1 \leq k \leq t)$ 的梯度为:

$$\begin{aligned}\frac{\partial \mathcal{L}_t}{\partial z_k} &= \frac{\partial h_k}{\partial z_k} \frac{\partial z_{k+1}}{\partial h_k} \frac{\partial \mathcal{L}_t}{\partial z_{k+1}} \\ &= \mathrm{diag}(f'(z_k)) \cdot U^{\mathrm{T}} \cdot \frac{\partial \mathcal{L}_t}{\partial z_{k+1}}\end{aligned}$$

于是我们有:

$$\begin{aligned}\frac{\partial \mathcal{L}_t}{\partial U} &= \sum_{k=1}^{t} \frac{\partial z_k}{\partial U} \frac{\partial \mathcal{L}_t}{\partial z_k} \\ &= \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial z_k} h_{k-1}^{\mathrm{T}}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}_t}{\partial W} &= \sum_{k=1}^{t} \frac{\partial z_k}{\partial W} \frac{\partial \mathcal{L}_t}{\partial z_k} \\ &= \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial z_k} x_k^{\mathrm{T}}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}_t}{\partial b} &= \sum_{k=1}^{t} \frac{\partial z_k}{\partial b} \frac{\partial \mathcal{L}_t}{\partial z_k} \\ &= \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial z_k}\end{aligned}$$

因此我们有:

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial U} &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial U} \\
&= \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial z_k} h_{k-1}^{\mathrm{T}} \\
\hline
\frac{\partial \mathcal{L}}{\partial W} &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial W} \\
&= \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial z_k} x_k^{\mathrm{T}} \\
\hline
\frac{\partial \mathcal{L}}{\partial b} &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial b} \\
&= \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial z_k}
\end{aligned}
$$

在 BPTT 算法中，参数的梯度需要在一个完整的 "前向" 计算和 "反向" 计算后才能得到并进行参数更新.
记误差项 $\delta_{t,k} := \frac{\partial \mathcal{L}_t}{\partial z_k}$，则反向传播过程如下图所示:

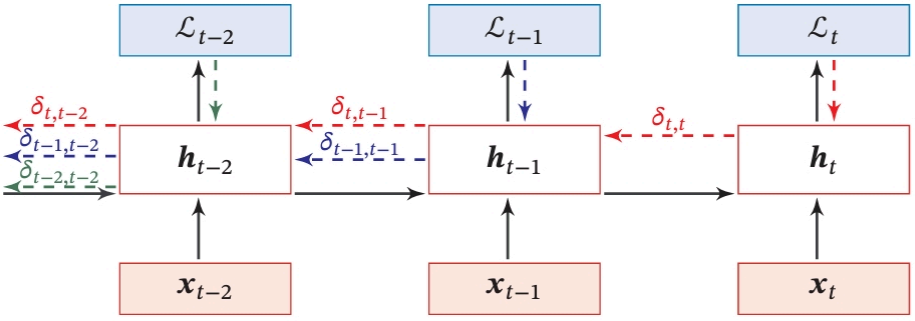



图 6.6　误差项随时间反向传播算法示例

### (2) 实时循环学习算法

与反向传播的 BPTT 算法不同的是，
**实时循环学习** (Real-Time Recurrent Learning, RTRL) 是通过前向传播的方式来计算梯度.
**(待补充: 神经网络与深度学习 6.4.2 节)**

## 5.2.4 长程依赖问题

循环神经网络在学习过程中的主要问题是由于梯度消失或爆炸问题，
很难建模**长时间间隔** (Long Range) 的状态之间的依赖关系.

在 BPTT 算法中，根据 $\frac{\partial \mathcal{L}_t}{\partial z_k} = \mathrm{diag}(f'(z_k)) \cdot U^{\mathrm{T}} \cdot \frac{\partial \mathcal{L}_t}{\partial z_{k+1}}$ 可得:

$$
\begin{aligned}
\frac{\partial \mathcal{L}_t}{\partial z_k} &= \mathrm{diag}(f'(z_k)) \cdot U^{\mathrm{T}} \cdot \frac{\partial \mathcal{L}_t}{\partial z_{k+1}} \\
&= \cdots \\
&= \left(\mathrm{diag}(f'(z_k)) \cdot U^{\mathrm{T}}\right) \cdots \left(\mathrm{diag}(f'(z_{t-1})) \cdot U^{\mathrm{T}}\right) \cdot \frac{\partial \mathcal{L}_t}{\partial z_{k+1}} \\
&= \left\{ \prod_{\tau=k}^{t-1} \mathrm{diag}(f'(z_k)) \cdot U^{\mathrm{T}} \right\} \cdot \frac{\partial \mathcal{L}_t}{\partial z_t} \\
&\simeq \gamma^{t-k} \frac{\partial \mathcal{L}_t}{\partial z_t}
\end{aligned}
$$

其中 $\gamma \simeq \|\mathrm{diag}(f'(z_k)) \cdot U^{\mathrm{T}}\|$

- 当 $\gamma > 1$ 且 $t-k$ 较大时，$\gamma^{t-k}$ 会很大，使得 $\frac{\partial \mathcal{L}_t}{\partial z_k}$ 很大，导致长距离的状态对训练影响很大.
  这称为**梯度爆炸问题** (gradient exploding problem)
- 当 $\gamma < 1$ 且 $t-k$ 较大时，$\gamma^{t-k}$ 会很小，使得 $\frac{\partial \mathcal{L}_t}{\partial z_k}$ 很小，导致长距离的状态对训练几乎没有影响.
  这称为**梯度消失问题** (vanishing gradient problem)

虽然简单循环网络理论上可以建立长时间间隔的状态之间的依赖关系，
但是由于梯度爆炸或消失问题，实际上只能学习到短期的依赖关系.
这称为**长程依赖问题** (long-term dependencies problem)

---

为了避免梯度爆炸或消失问题，一种最直接的方式就是选取合适的参数，
同时使用非饱和的激活函数，尽量使得 $\gamma \simeq \|\mathrm{diag}(f'(z_k)) \cdot U^{\mathrm{T}}\| \approx 1$
这种方式依赖于足够的人工调参经验，限制了模型的广泛应用.
更有效的方式是通过改进模型或优化方法来缓解循环网络的梯度爆炸和梯度消失问题.

- **梯度爆炸:**
  一般而言，循环网络的梯度爆炸问题比较容易解决，一般通过权重衰减或梯度截断来避免.
  权重衰减是通过给参数增加 $l_1$ 或 $l_2$ 范数的正则化项来限制参数的取值范围，从而使得 $\gamma \leq 1$.
  梯度截断是另一种有效的启发式方法，当梯度的模大于一定阈值时，就将它截断成为一个较小的数.
- **梯度消失:**
  梯度消失是循环网络的主要问题，比较难解决.
  我们可以将前向传播逻辑改进为:

  $$h_t = h_{t-1} + g(x_t, h_{t-1}; \theta)$$

  其中 $g(x, h; \theta)$ 为非线性函数.
  但即使解决了梯度消失问题，隐状态 $h_t$ 可以存储的信息仍是有限的，
  随着输入信息的不断累积，$h_t$ 会逐渐趋于饱和 (假设使用 Sigmoid 型激活函数)，可能会丢失更多有效信息.

为了解决这两个问题，可以通过引入门控机制来进一步改进模型.

# 5.3 基于门控的循环神经网络

为了改善循环神经网络的长程依赖问题，
一种好的解决方案是在 $h_t = h_{t-1} + g(x_t, h_{t-1}; \theta)$ 的基础上引入门控机制来控制信息的累积速度，
包括有选择地加入新的信息，并有选择地遗忘之前累积的信息.
这一类网络可以称为**基于门控的循环神经网络** (Gated RNN).

## 5.3.1 长短期记忆网络

**长短期记忆网络** (Long Short-Term Memory Network, LSTM)
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.
其主要改进如下:

- **① 引入内部状态:**
  LSTM 网络引入了**内部状态** (internal state) $c_t \in \mathbb{R}^D$，专门进行线性的循环信息传递.
  同时 (非线性地) 输出信息给隐藏层的外部状态 $h_t \in \mathbb{R}^D$:
  (假设使用 $\tanh(\cdot)$ 作为激活函数)

$$\begin{aligned} \tilde{c}_t &= \tanh(Wx_t + Uh_{t-1} + b) \\ c_t &= \mathrm{FG}_t \odot c_{t-1} + \mathrm{IG}_t \odot \tilde{c}_t \\ h_t &= \mathrm{OG}_t \odot \tanh(c_t) \end{aligned}$$

  其中遗忘门 $\mathrm{FG}_t \in [0,1]^D$，输入门 $\mathrm{IG}_t \in [0,1]^D$，输出门 $\mathrm{OG}_t \in [0,1]^D$
  在每个时刻 $t$，LSTM 网络的内部状态 $c_t$ 记录了到当前时刻为止的历史信息.

- **② 软性门:**
  软性门的取值范围为 $[0,1]$，以一定的比例允许信息通过.
  遗忘门 $\mathrm{FG}_t$ 控制上一个时刻的内部状态 $c_{t-1}$ 需要遗忘多少信息;
  输入门 $\mathrm{IG}_t$ 控制当前时刻的候选状态 $\tilde{c}_t := \tanh(Uh_{t-1} + Wx_t + b)$ 有多少信息需要保存;
  输出门 $\mathrm{OG}_t$ 控制当前时刻的内部状态 $c_t$ 有多少信息需要输出给外部状态 $h_t$.
  三个门的计算方式如下:

$$\begin{aligned} \mathrm{FG}_t &= \sigma(W_{\mathrm{FG}} \cdot x_t + U_{\mathrm{FG}} \cdot h_{t-1} + b_{\mathrm{FG}}) \\ \mathrm{IG}_t &= \sigma(W_{\mathrm{IG}} \cdot x_t + U_{\mathrm{IG}} \cdot h_{t-1} + b_{\mathrm{IG}}) \\ \mathrm{OG}_t &= \sigma(W_{\mathrm{OG}} \cdot x_t + U_{\mathrm{OG}} \cdot h_{t-1} + b_{\mathrm{OG}}) \end{aligned}$$

  其中 $\sigma(\cdot)$ 为 Logistic 函数,
  同时 $W_{\mathrm{FG}}, U_{\mathrm{FG}}, b_{\mathrm{FG}}, W_{\mathrm{IG}}, U_{\mathrm{IG}}, b_{\mathrm{IG}}, W_{\mathrm{OG}}, U_{\mathrm{OG}}, b_{\mathrm{OG}}$ 都是可学习的参数.

LSTM 网络的循环单元结构为:

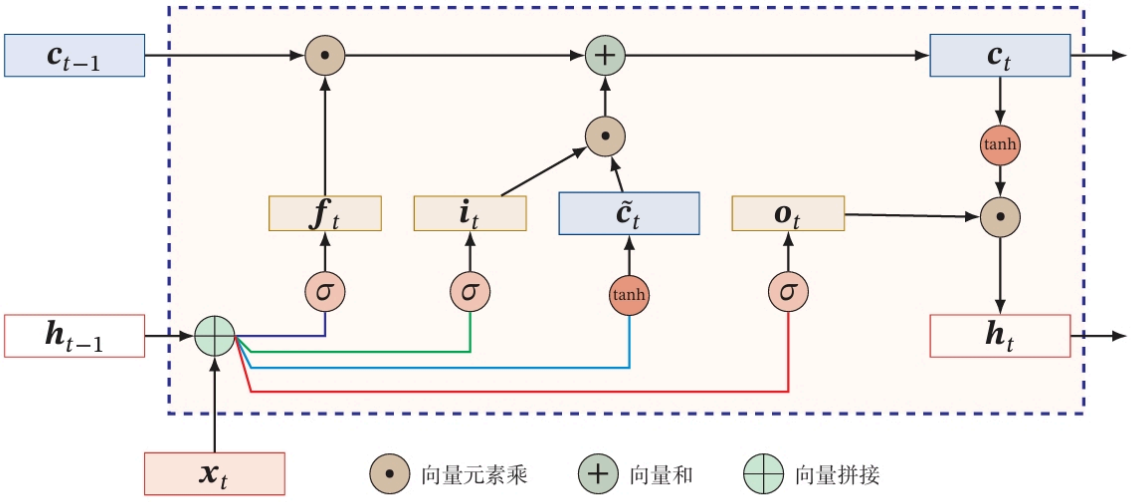



图 6.7　LSTM 网络的循环单元结构

其计算过程如下:

- ① 首先利用上一时刻的外部状态 $h_{t-1}$ 和当前时刻的输入 $x_t$，
  计算出三个门 $\mathrm{FG}_t, \mathrm{IG}_t, \mathrm{OG}_t$，以及候选状态 $\tilde{c}_t$:

$$
\begin{aligned}
\mathrm{FG}_t &= \sigma(W_{\mathrm{FG}} \cdot x_t + U_{\mathrm{FG}} \cdot h_{t-1} + b_{\mathrm{FG}}) \\
\mathrm{IG}_t &= \sigma(W_{\mathrm{IG}} \cdot x_t + U_{\mathrm{IG}} \cdot h_{t-1} + b_{\mathrm{IG}}) \\
\mathrm{OG}_t &= \sigma(W_{\mathrm{OG}} \cdot x_t + U_{\mathrm{OG}} \cdot h_{t-1} + b_{\mathrm{OG}}) \\
\tilde{c}_t &= \tanh(W x_t + U h_{t-1} + b) \\
&\Updownarrow \\
\begin{bmatrix} \mathrm{FG}_t \\ \mathrm{IG}_t \\ \mathrm{OG}_t \\ \tilde{c}_t \end{bmatrix} &= \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \left( \begin{bmatrix} W_{\mathrm{FG}} & U_{\mathrm{FG}} \\ W_{\mathrm{IG}} & U_{\mathrm{IG}} \\ W_{\mathrm{OG}} & U_{\mathrm{OG}} \\ W & U \end{bmatrix} \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + \begin{bmatrix} b_{\mathrm{FG}} \\ b_{\mathrm{IG}} \\ b_{\mathrm{OG}} \\ b \end{bmatrix} \right)
\end{aligned}
$$

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 $\mathrm{LSTM}$ 网络时，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 $b_{\mathrm{FG}}$ 设为 1 或 2.

- ② 结合遗忘门 $\mathrm{FG}_t$ 和输入门 $\mathrm{IG}_t$ 来更新记忆单元 (即内部状态) $c_t$:

$$
c_t = \mathrm{FG}_t \odot c_{t-1} + \mathrm{IG}_t \odot \tilde{c}_t
$$

- ③ 结合输出门 $\mathrm{OG}_t$，将内部状态 $c_t$ 的信息传递给外部状态 $h_t$:

$$
h_t = \mathrm{OG}_t \odot \tanh(c_t)
$$

通过 $\mathrm{LSTM}$ 循环单元，整个网络可以建立较长距离的时序依赖关系.
循环神经网络中的隐状态 $h_t$ 存储了历史信息，可以看作一种记忆.
在简单循环网络中，隐状态每个时刻都会被重写，因此可以看作一种短期记忆 (short-term memory).
在神经网络中，长期记忆 (long-term memory) 可以看作网络参数，
隐含了从训练数据中学到的经验，其更新周期要远远慢于短期记忆.
而在 $\mathrm{LSTM}$ 网络中，记忆单元 $c_t$ 可以在某个时刻捕捉到某个关键信息，
并有能力将此关键信息保存一定的时间间隔.
记忆单元 $c_t$ 中保存信息的生命周期要长于短期记忆 $h_t$，但又远远短于长期记忆.

---

**($\mathrm{LSTM}$ 网络的变体)**
目前主流的 $\mathrm{LSTM}$ 网络用三个门来动态地控制内部状态
应该遗忘多少历史信息，输入多少新信息，以及输出多少信息.
我们可以对门控机制进行改进并获得 $\mathrm{LSTM}$ 网络的不同变体.

- **无遗忘门的 $\mathrm{LSTM}$ 网络**
  最早提出的 $\mathrm{LSTM}$ 网络是没有遗忘门的，其内部状态的更新为:

$$
\begin{aligned}
\mathrm{IG}_t &= \sigma(W_{\mathrm{IG}} \cdot x_t + U_{\mathrm{IG}} \cdot h_{t-1} + b_{\mathrm{IG}}) \\
\mathrm{OG}_t &= \sigma(W_{\mathrm{OG}} \cdot x_t + U_{\mathrm{OG}} \cdot h_{t-1} + b_{\mathrm{OG}}) \\
\tilde{c}_t &= \tanh(W x_t + U h_{t-1} + b) \\
c_t &= c_{t-1} + \mathrm{IG}_t \odot \tilde{c}_t \\
h_t &= \mathrm{OG}_t \odot \tanh(c_t)
\end{aligned}
$$

  其问题是记忆单元 $c_t$ 会不断增大.
  当输入序列的长度非常大时，记忆单元的容量会饱和，从而大大降低 $\mathrm{LSTM}$ 模型的性能.

- **peephole 连接**
  另外一种变体是三个门不但依赖于输入 $x_t$ 和上一时刻的隐状态 $h_{t-1}$，
  还依赖于上一个时刻的记忆单元 $c_{t-1}$，即:

$$
\begin{aligned}
\mathrm{FG}_t &= \sigma(W_{\mathrm{FG}} \cdot x_t + U_{\mathrm{FG}} \cdot h_{t-1} + V_{\mathrm{FG}} \cdot c_{t-1} + b_{\mathrm{FG}}) \\
\mathrm{IG}_t &= \sigma(W_{\mathrm{IG}} \cdot x_t + U_{\mathrm{IG}} \cdot h_{t-1} + V_{\mathrm{IG}} \cdot c_{t-1} + b_{\mathrm{IG}}) \\
\mathrm{OG}_t &= \sigma(W_{\mathrm{OG}} \cdot x_t + U_{\mathrm{OG}} \cdot h_{t-1} + V_{\mathrm{OG}} \cdot c_{t-1} + b_{\mathrm{OG}}) \\
\tilde{c}_t &= \tanh(W x_t + U h_{t-1} + b) \\
c_t &= \mathrm{FG}_t \odot c_{t-1} + \mathrm{IG}_t \odot \tilde{c}_t \\
h_t &= \mathrm{OG}_t \odot \tanh(c_t)
\end{aligned}
$$

其中 $V_{\mathrm{FG}}, V_{\mathrm{IG}}, V_{\mathrm{OG}}$ 为对角阵.

- **耦合输入门和遗忘门**
  $\mathrm{LSTM}$ 网络中的输入门和遗忘门有些互补关系，因此同时用两个门比较冗余.
  为了减少 $\mathrm{LSTM}$ 网络的计算复杂度，将这两门合并为一个门.
  令 $\mathrm{FG}_t := 1 - \mathrm{IG}_t$，内部状态的更新方式为:

$$
\begin{aligned}
\mathrm{IG}_t &= \sigma(W_{\mathrm{IG}} \cdot x_t + U_{\mathrm{IG}} \cdot h_{t-1} + b_{\mathrm{IG}}) \\
\mathrm{OG}_t &= \sigma(W_{\mathrm{OG}} \cdot x_t + U_{\mathrm{OG}} \cdot h_{t-1} + b_{\mathrm{OG}}) \\
\tilde{c}_t &= \tanh(W x_t + U h_{t-1} + b) \\
c_t &= (1 - \mathrm{IG}_t) \odot c_{t-1} + \mathrm{IG}_t \odot \tilde{c}_t \\
h_t &= \mathrm{OG}_t \odot \tanh(c_t)
\end{aligned}
$$

## 5.3.2 门控循环单元网络

**门控循环单元** (Gated Recurrent Unit, GRU) 网络是一种比 $\mathrm{LSTM}$ 网络更加简单的循环神经网络.
$\mathrm{GRU}$ 网络引入门控机制来控制信息更新的方式，但和 $\mathrm{LSTM}$ 不同，$\mathrm{GRU}$ 不引入额外的记忆单元.
$\mathrm{GRU}$ 网络引入一个**更新门** (update gate)
来控制当前状态需要从历史状态中保留多少信息 (不经过非线性变换)，
以及需要从候选状态中接受多少新信息.
同时使用一个**重置门** (reset gate) 来控制候选状态 $\tilde{h}_t$ 依赖于上一时刻状态 $h_{t-1}$ 的程度:

$$
\begin{aligned}
\mathrm{UG}_t &= \sigma(W_{\mathrm{UG}} \cdot x_t + U_{\mathrm{UG}} \cdot h_{t-1} + b_{\mathrm{UG}}) \\
\mathrm{RG}_t &= \sigma(W_{\mathrm{RG}} \cdot x_t + U_{\mathrm{RG}} \cdot h_{t-1} + b_{\mathrm{RG}}) \\
\tilde{h}_t &= \tanh(W_{\mathrm{H}} \cdot x_t + U_{\mathrm{H}} \cdot h_{t-1} + b_{\mathrm{H}}) \\
h_t &= \mathrm{UG}_t \odot h_{t-1} + (1 - \mathrm{UG}_t) \odot \tilde{h}_t
\end{aligned}
$$

其中 $W_{\mathrm{UG}}, \mathrm{U}_{\mathrm{UG}}, b_{\mathrm{UG}}, W_{\mathrm{RG}}, U_{\mathrm{RG}}, b_{\mathrm{RG}}, W_{\mathrm{H}}, U_{\mathrm{H}}, b_{\mathrm{H}}$ 都是可学习的参数.

在 $\mathrm{LSTM}$ 网络中，输入门和遗忘门是互补关系，具有一定的冗余性.
$\mathrm{GRU}$ 网络直接使用一个门来控制输入和遗忘之间的平衡.
当 $\mathrm{UG}_t \equiv 0, \mathrm{RG}_t \equiv 1$ 时，$\mathrm{GRU}$ 网络退化为**简单循环网络**;
当 $\mathrm{UG}_t \equiv 0$ 时，$\mathrm{GRU}$ 网络的当前状态 $h_t$ 只与当前输入 $x_t$ 有关，与历史状态 $h_{t-1}$ 无关;
当 $\mathrm{UG}_t \equiv 1$ 时，$\mathrm{GRU}$ 网络的当前状态 $h_t$ 只与历史状态 $h_{t-1}$ 有关，与当前输入 $x_t$ 无关.

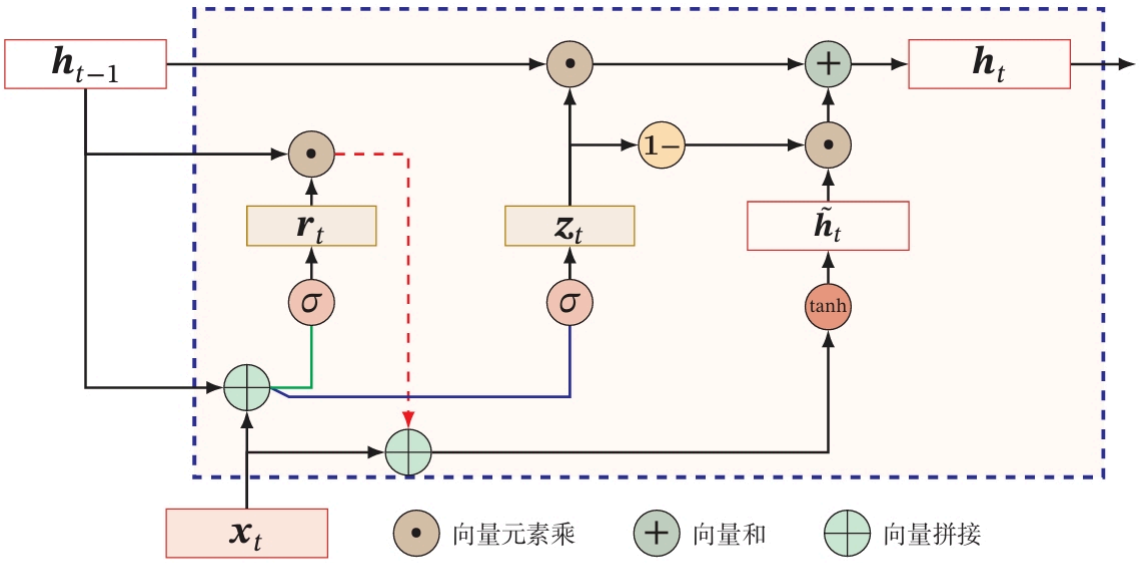



图 6.8　GRU 网络的循环单元结构

## 5.3.3 堆叠循环神经网络

如果将深度定义为网络中信息传递路径长度的话，则循环神经网络可以看作既 "深" 又 "浅" 的网络.
一方面来说，如果我们把循环网络按时间展开，长时间间隔的状态之间的路径很长，
那么循环网络可以看作一个非常深的网络;
从另一方面来说，如果同一时刻网络输入到输出之间的路径 $x_t \to y_t$，那么这个网络是非常浅的.
因此我们可以增加循环神经网络的深度从而增强循环神经网络的能力.
增加循环神经网络的深度主要是增加同一时刻网络输入到输出之间的路径 $x_t \to y_t$，
比如增加隐状态到输出 $h_t \to y_t$，以及输入到隐状态 $x_t \to h_t$ 之间的路径的深度.

一种常见的增加循环神经网络深度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)
一个堆叠的简单循环网络 (Stacked SRN)
也称为**循环多层感知器** (Recurrent Multi-Layer Perceptron, RMLP)
下图给出了按时间展开的堆叠循环神经网络:

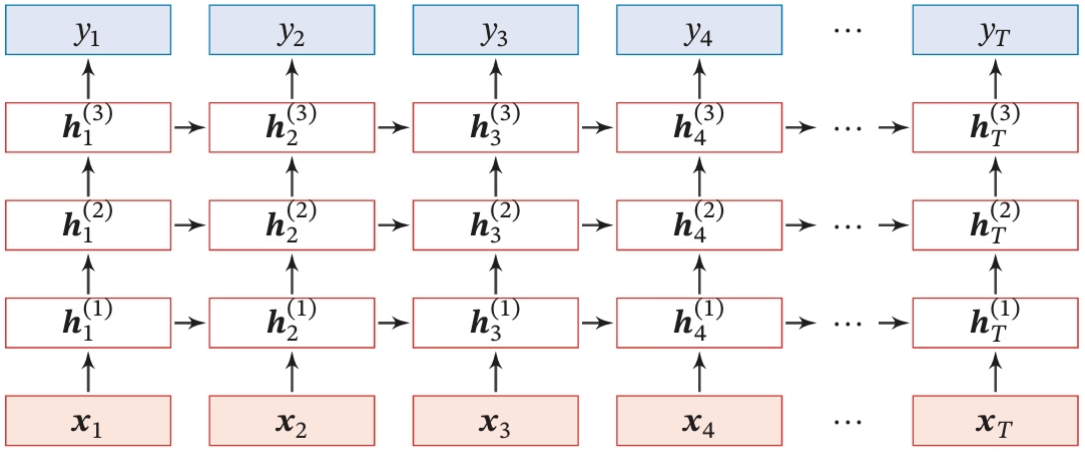



图 6.9　按时间展开的堆叠循环神经网络

其中第 $l$ 层网络的输入是第 $l-1$ 层网络的输出.
我们定义 $h_t^{(l)}$ 为在时刻 $t$ 时第 $l$ 层的隐状态:

$$h_t^{(l)} = f(W^{(l)} h_t^{(l-1)} + U^{(l)} h_{t-1}^{(l)} + b^{(l)})$$

特殊地，$h_t^{(0)} = x_t$.

### 5.3.4 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 Network, Bi-RNN)
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 $t$ 时的隐状态定义为 $h_t^{(1)}$ 和 $h_t^{(2)}$，则:

$$h_t^{(1)} = f(W^{(1)} x_t + U^{(1)} h_{t-1}^{(1)} + b^{(1)})$$
$$h_t^{(2)} = f(W^{(2)} x_t + U^{(2)} h_{t+1}^{(2)} + b^{(2)})$$
$$h_t = \begin{bmatrix} h_t^{(1)} \\ h_t^{(2)} \end{bmatrix}$$

下图给出了按时间展开的双向循环神经网络:

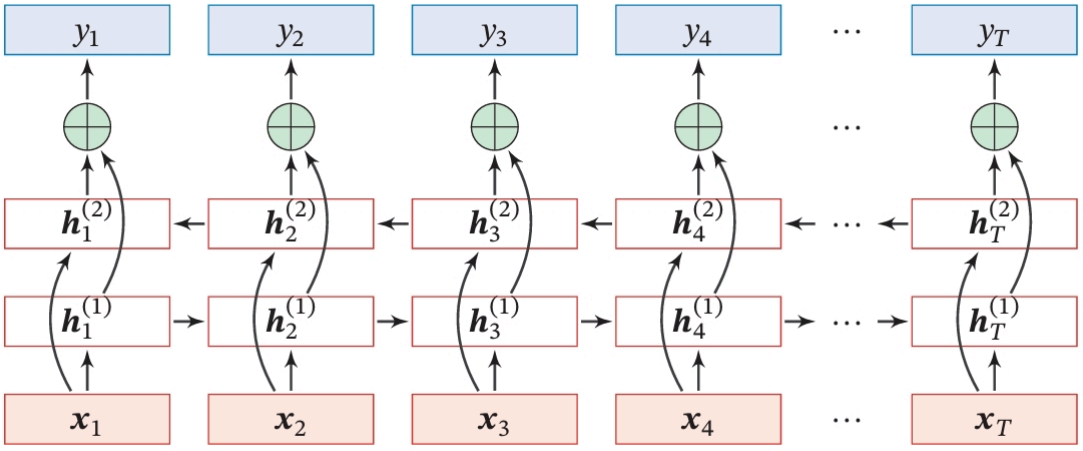



图 6.10　按时间展开的双向循环神经网络

## 5.5 注意力机制

### 5.5.1 注意力机制

在计算能力有限的情况下，
**注意力机制** (attention mechanism) 可利用有限的计算资源用来处理更重要的信息.
当用神经网络来处理大量的输入信息时，也可以借鉴人脑的注意力机制，
只选择一些关键的信息输入进行处理，来提高神经网络的效率.

在目前的神经网络模型中，我们可以将最大汇聚 (max pooling)、门控 (gating) 机制
近似地看作自下而上的基于显著性的注意力机制.
除此之外，自上而下的聚焦式注意力也是一种有效的信息选择方式.
以阅读理解任务为例，给定一篇很长的文章，然后就此文章的内容进行提问.
提出的问题只和段落中的一两个句子相关，其余部分都是无关的.
为了减小神经网络的计算负担，只需要把相关的片段挑选出来让后续的神经网络来处理，
而不需要把所有文章内容都输入给神经网络.

用 $X = [x_1, \dots, x_N] \in \mathbb{R}^{D \times N}$ 表示 $N$ 组输入信息，
其中 $D$ 维向量 $x_n \in \mathbb{R}^D, n \in [1, N]$ 表示一组输入信息.
为了节省计算资源，不需要将所有信息都输入神经网络，只需要从 $X$ 中选择一些和任务相关的信息.
注意力机制的计算可以分为两步:
一是在所有输入信息上计算注意力分布，
二是根据注意力分布来计 算输入信息的加权平均.

---

**(注意力分布)**
为了从 $N$ 个输入向量 $X = [x_1, \dots, x_N] \in \mathbb{R}^{D \times N}$ 中选择出和某个特定任务相关的信息，
我们需要引入一个和任务相关的表示，称为**查询向量** (query vector)，
并通过一个打分函数来计算每个输入向量和查询向量之间的相关性.

给定一个和任务相关的查询向量 $q$ (可以是动态生成的，也可以是可学习的参数)，
我们用注意力变量 $z \in [1, N]$ 来表示被选择信息的索引位置，即 $z = n$ 表示选择了第 $n$ 个输入向量.
为了方便计算，我们采用一种 "软性" 的信息选择机制.
首先计算在给定 $q$ 和 $X$ 下，选择第 $n$ 个输入向量的概率 $\alpha_n$:

$$\begin{aligned} \alpha_n &= p(z = n | X, q) \\ &= [\mathrm{softmax}(s(X, q))]_n \\ &= \frac{\exp(s(x_n, q))}{\sum_{j=1}^{N} \exp(s(x_j, q))} \end{aligned}$$

其中 $\alpha_n$ 称为**注意力分布** (attention distribution)，$s(x, q)$ 为**注意力打分函数**.
可以使用以下几种方式来计算:

- 加性模型: $s(x, q) = v^{\mathrm{T}} \tanh(Wx + Uq)$
- 点积模型: $s(x, q) = x^{\mathrm{T}} q$
  理论上加性模型和点积模型的复杂度差不多，
  但是点积模型在实现上可以更好地利用矩阵乘积，从而计算效率更高.
- 缩放点积模型: $s(x, q) = x^{\mathrm{T}} q / \sqrt{D}$
  当输入向量的维度 $D$ 比较高时，点积模型的值通常有比较大的方差，
  从而导致 $\mathrm{Softmax}$ 函数的梯度会比较小.
  因此缩放点积模型可以较好地解决这个问题.
- 双线性模型: $s(x, q) = x^{\mathrm{T}} W q = (Ux)^{\mathrm{T}} (Vq)$ (其中 $W = U^{\mathrm{T}} V$)
  双线性模型是一种泛化的点积模型，在计算相似度时可以引入非对称性.

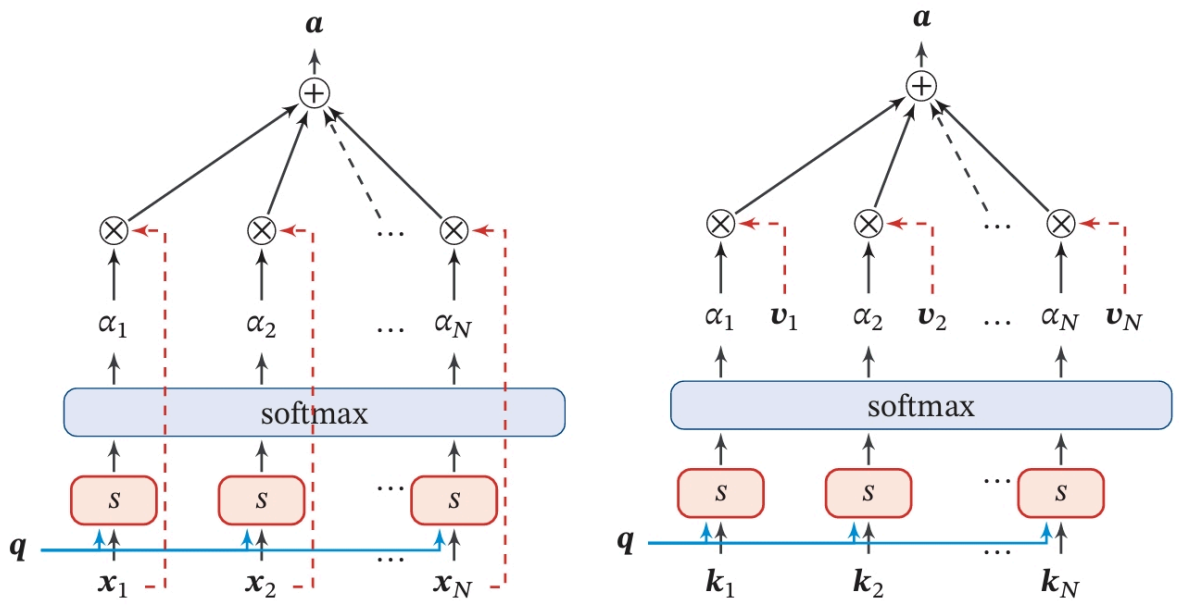
其中 $W, U, v$ 为可学习的参数，$D$ 为输入向量的维度.

## (1) 软性注意力

注意力分布 $\alpha_n$ 可以解释为在给定任务相关的查询 $q$ 时，第 $n$ 个输入向量受关注的程度.
我们采用一种 "软性" 的信息选择机制对输入信息进行汇总，
称为**软性注意力机制** (soft attension mechanism):

$$\begin{aligned}\text{attn}(X, q) &= \sum_{n=1}^{N} \alpha_n x_n \\ &= \mathbb{E}_{z \sim p(z|X,q)}[x_z]\end{aligned}$$

如下图 (a) 所示:




(a) 普通模式　　(b) 键值对模式

图 8.1　注意力机制

### (2) 硬性注意力

软性注意力选择的信息是所有输入向量在注意力分布下的期望，
与之相对的是**硬性注意力** (hard attention)，它有两种实现方式:

- 一种是选取最高概率的一个输入向量，即:

$$\begin{aligned}\text{attn}(X, q) &= x_{\hat{n}} \\ \text{where } \hat{n} &:= \arg \max_{n=1,\ldots,N} \alpha_n\end{aligned}$$

  其中 $\hat{n}$ 为概率最大的输入向量的下标.

- 另一种硬性注意力可以通过在注意力分布式上随机采样的方式实现.

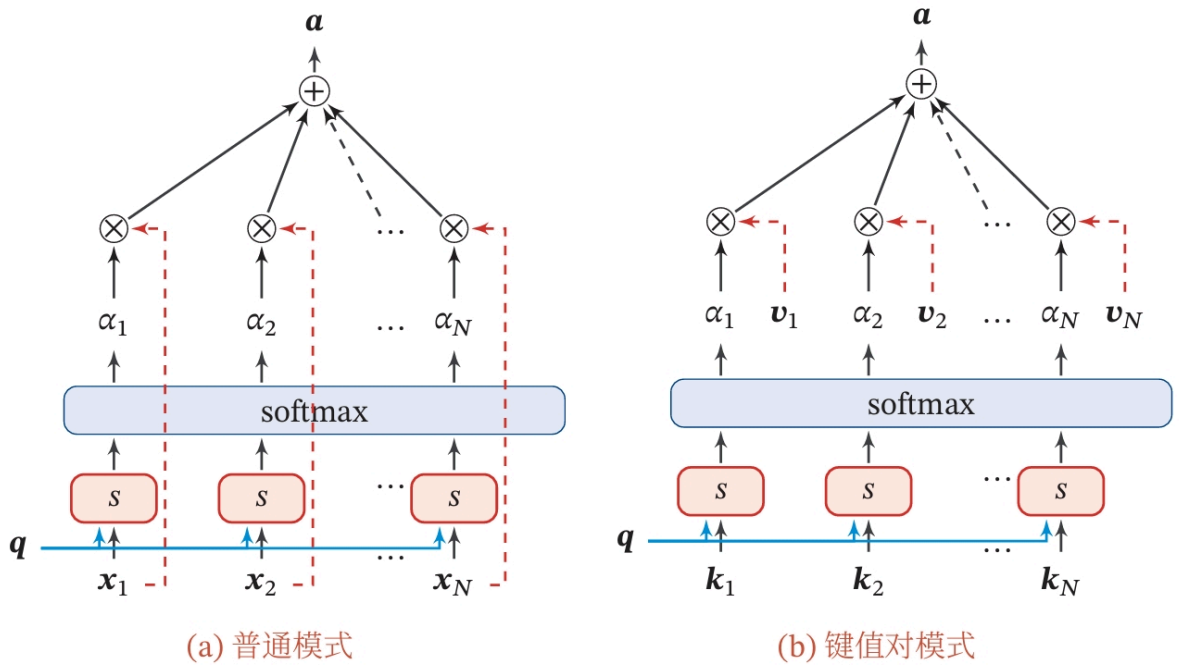
硬性注意力的一个缺点是基于最大采样或随机采样的方式来选择信息，
使得最终的损失函数与注意力分布之间的函数关系不可导，无法使用反向传播算法进行训练.
因此硬性注意力通常需要使用强化学习来进行训练.
为了使用反向传播算法，一般使用软性注意力来代替硬性注意力.

### (3) 键值对注意力

更一般地，我们可以用**键值对** (key-value pair) 格式来表示输入信息，
其中 "键" 用来计算注意力分布 $\alpha_n$，"值" 用来计算聚合信息.
用 $(K, V) = [(k_1, v_1), \ldots, (k_N, v_N)]$ 表示 $N$ 组输入信息，
给定任务相关的查询向量 $q$ 时，注意力函数为:

$$
\begin{aligned}
\text{attn}((K,V),q) &= \sum_{n=1}^{N} \alpha_n v_n \\
&= \sum_{n=1}^{N} [\text{softmax}(s(K,q))]_n \cdot v_n \\
&= \sum_{n=1}^{N} \frac{\exp(s(k_n,q))}{\sum_j \exp(s(k_j,q))} v_n \\
&\text{where } s(k,q) \text{ is score function}
\end{aligned}
$$

如下图 (b) 所示:




(a) 普通模式　　(b) 键值对模式

图 8.1　注意力机制

当 $K = V$ 时，键值对模式就等价于软性注意力机制.

### (4) 多头注意力

**多头注意力** (multi-head attention)
利用多个查询 $Q = [q_1, \ldots, q_M]$ 来并行地从输入信息中选取多组信息.
每个注意力关注输入信息的不同部分:

$$
\text{attn}((K,V),Q) = \begin{bmatrix} \text{attn}((K,V),q_1) \\ \text{attn}((K,V),q_2) \\ \vdots \\ \text{attn}((K,V),q_M) \end{bmatrix}
$$

### (5) 结构化注意力

之前我们假设所有的输入信息是同等重要的，是一种扁平结构，
但实际上注意力分布是在所有输入信息上的多项分布.
但如果输入信息本身具有层次结构，比如文本可以分为词、句子、段落、篇章等不同粒度的层次，
我们可以使用层次化的注意力来进行更好的信息选择.

此外，我们还可以假设注意力为上下文相关的二项分布，
用一种图模型来构建更复杂的结构化注意力分布.

### (6) 指针网络

我们可以将注意力分布作为一个软性的指针来指出相关信息的位置.
**指针网络** (Pointer Network) 是一种序列到序列模型，
输入是长度为 $N$ 的向量序列 $X = [x_1, \dots, x_N] \in \mathbb{R}^{D \times N}$，
输出是长度为 $M$ 的下标序列 $C = [c_1, \dots, c_M] \in [1, N]^D$，它们是输入序列的下标.

条件概率 $p(C|X)$ 可以写为:

$$\begin{aligned} p(C|X) &= \prod_{m=1}^{M} p(c_m|C_{[1:m-1]}, X) \\ &\approx \prod_{m=1}^{M} p(c_m|x_{c_1}, \dots, x_{c_{m-1}}, X) \end{aligned}$$

其中条件概率 $p(c_m|x_{c_1}, \dots, x_{c_{m-1}}, X)$ 可通过注意力分布来计算.
假设用一个循环神经网络对 $x_{c_1}, \dots, x_{c_{m-1}}, X$ 进行编码得到向量 $h_m$，
记 $s_{m,n}$ 为在解码过程的第 $m$ 步时，$h_m$ 对 $h_n$ 的未归一化的注意力分布:

$$\begin{gathered} s_{m,n} = v^{\mathrm{T}} \tanh(W x_n + U h_m) \quad (n \in [1, N]) \\ \text{where parameters } v, W, U \text{ are learnable} \end{gathered}$$

则我们有:

$$p(c_m|C_{[1:m-1]}, X) \approx p(c_m|x_{c_1}, \dots, x_{c_{m-1}}, X) = \mathrm{softmax}(s_{m,n})$$

下图给出了指针网络的示例.
其中 $h_1, h_2, h_3$ 为输入数字 $20, 5, 10$ 经过循环神经网络的隐状态，$h_0$ 对应一个特殊字符 '$<$'.
当输入 '$>$' 时，网络一步一步输出三个输入数字从大到小排列的下标.

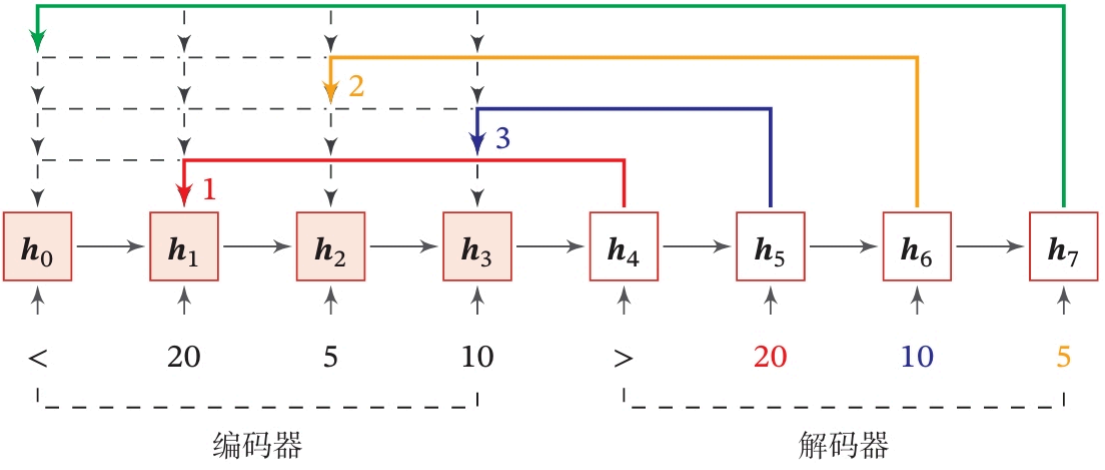
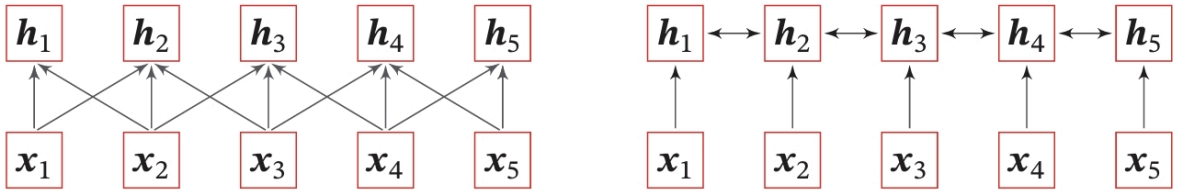



图 8.2　指针网络

## 5.5.2 自注意力模型

当使用神经网络来处理一个变长的向量序列时，
我们通常可以使用卷积网络或循环网络进行编码来得到一个相同长度的输出向量序列:




(a) 卷积网络　　(b) 双向循环网络

图 8.3　基于卷积网络和循环网络的变长序列编码

基于卷积或循环网络的序列编码都是一种局部的编码方式，只建模了输入信息的局部依赖关系.
虽然循环网络理论上可以建立长距离依赖关系，
但是由于信息传递的容量以及梯度消失问题，实际上也只能建立短距离依赖关系.

如果要建立输入序列之间的长距离依赖关系，可以使用以下两种方法:

- 一种方法是增加网络的层数，通过一个深层网络来获取远距离的信息交互;
- 另一种方法是使用全连接网络.
  全连接网络是一种非常直接的建模远距离依赖的模型，但是无法处理变长的输入序列.
  不同的输入长度，其连接权重的大小也是不同的.

我们可以利用注意力机制来 "动态" 地生成不同连接的权重，即为**自注意力模型** (Self-Attention Model)
为了提高模型能力，自注意力模型经常采用**查询-键-值** (Query-Key-Value, QKV) 模式，
其计算过程如下图所示，其中红色字母表示矩阵的维度:

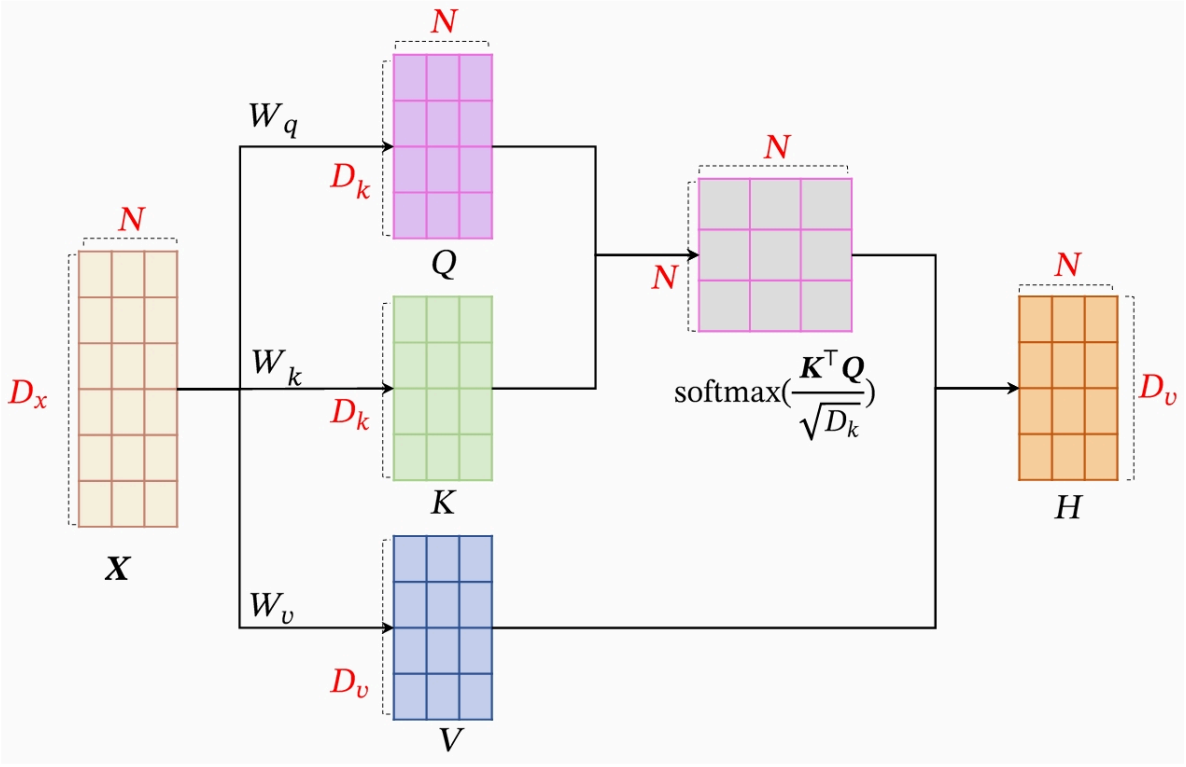



图 8.4　自注意力模型的计算过程

由于在自注意力模型中通常使用缩放点积模型来计算注意力得分，
我们默认查询向量和键向量的维度是相同的.

设输入序列 $X = [x_1, \dots, x_N] \in \mathbb{R}^{D_x \times N}$，输出序列 $H = [h_1, \dots, h_N] \in \mathbb{R}^{D_v \times N}$
则自注意力模型的具体计算过程如下:

- ① 对于每个输入 $x_i$，我们首先将其线性映射到三个不同的空间，
  得到查询向量 $q_i \in \mathbb{R}^{D_k}$、键向量 $k_i \in \mathbb{R}^{D_k}$ 和值向量 $v_i \in \mathbb{R}^{D_v}$.
  对于整个输入序列 $X$，线性映射过程可以简写为:

$$
\begin{aligned}
Q &= W_q X = [q_1, \dots, q_N] \in \mathbb{R}^{D_k \times N} \\
K &= W_k X = [k_1, \dots, k_N] \in \mathbb{R}^{D_k \times N} \\
V &= W_v X = [v_1, \dots, v_N] \in \mathbb{R}^{D_v \times N}
\end{aligned}
$$

  其中 $W_q \in \mathbb{R}^{D_k \times D_x}, W_k \in \mathbb{R}^{D_k \times D_x}, W_v \in \mathbb{R}^{D_v \times D_x}$ 分别为线性映射的参数矩阵.
  而 $Q, K, V$ 分别是由查询向量、键向量和值向量构成的矩阵.

- ② 对于每一个查询向量 $q_n \in Q = [q_1, \dots, q_N]$
  根据键值对注意力机制得到输出向量 $h_n$:

$$
\begin{aligned}
h_n &= \text{att}((K, V), q_n) \\
&= \sum_{j=1}^{N} \alpha_{n,j} v_j \\
&= \sum_{j=1}^{N} [\text{softmax}(s(k_j, q_n))]_j \cdot v_j \\
&= \sum_{j=1}^{N} \frac{\exp(s(k_j, q_n))}{\sum_l \exp(s(k_l, q_n))} v_j
\end{aligned}
$$

$$
\text{where } s(k, q) \text{ is score function, typically } s(k, q) := \frac{k^{\mathrm{T}} q}{\sqrt{\dim(k)}}
$$

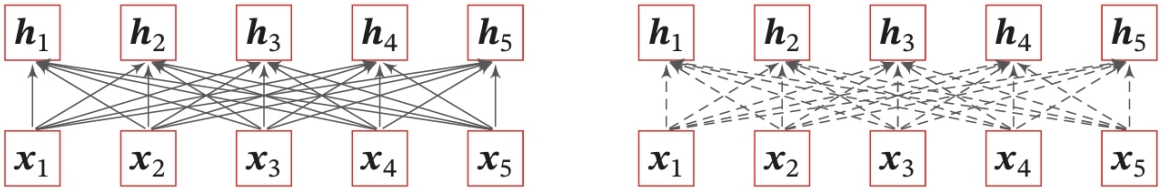
  其中 $n, j \in [1, N]$ 对于输出序列和输入序列的位置，
  $\alpha_{n,j}$ 代表第 $n$ 个输出关注到第 $j$ 个输入向量的权重.

  如果使用缩放点积来作为注意力打分函数，则输出向量序列可以简写为:

$$
H = V \cdot \text{softmax}\left(\frac{K^{\mathrm{T}} Q}{\sqrt{D_k}}\right)
$$

  其中 $\text{softmax}(\cdot)$ 为按列计算概率分布的函数.

  下图给出了全连接模型和自注意力模型的对比.
  其中实线表示可学习的权重，虚线表示动态生成的权重.
  由于自注意力模型的权重是动态生成的，因此可以处理变长的信息序列:




(a) 全连接模型　　(b) 自注意力模型

图 8.5　全连接模型和自注意力模型

自注意力模型可以作为神经网络中的一层来使用，
既可以用来替换卷积层和循环层，又可以和它们一起交替使用 (例如 $X$ 可以是卷积层或循环层的输出).
自注意力模型计算的权重 $\alpha_{i,j}$ 只依赖于 $q_i$ 和 $k_j$ 的相关性，而忽略了输入信息的位置信息.
因此在单独使用时，自注意力模型一般需要加入位置编码信息来进行修正.
自注意力模型可以扩展为多头自注意力模型，在多个不同的投影空间中捕捉不 同的交互信息.

# 5.6 序列生成模型

## 5.6.1 自回归生成模型

为描述自然语言规则，我们将一个长度为 $T$ 的文本序列看作一个随机事件 $X = \langle X_1, \dots, X_T \rangle$
其中每个位置上的变量 $X_t$ 的样本空间为一个给定的词表 (vocabulary) $\mathcal{V}$，整个序列 $X_t$ 的样本空间为 $|\mathcal{V}|^{\mathrm{T}}$
一个文本序列的概率大小可以用来评估它符合自然语言规则的程度.

给定一个序列样本 $x_{[1:T]} = [x_1, x_2, \dots, x_T]$，其概率是 $T$ 个词的联合概率:

$$\begin{aligned} p(x_{[1:T]}) &= \mathrm{P}(X_{[1:T]} = x_{[1:T]}) \\ &= \mathrm{P}(X_1 = x_1, \dots, X_T = x_T) \end{aligned}$$

和一般的概率模型类似，序列概率模型有两个基本问题:

- 概率密度估计: 给定一组序列数据，估计这些数据背后的概率分布.
- 样本生成: 从已知的序列分布中生成新的序列样本.

---

序列数据有两个特点: 样本是变长的，同时样本空间非常大.
因此我们很难用已知的概率模型来直接建模整个序列的概率.
序列数据的概率密度估计问题可以转换为单变量的条件概率估计问题，
即给定 $x_{[1:t-1]}$ 时 $x_t$ 的条件概率 $p(x_t|x_{[1:t-1]})$:

$$\begin{aligned} p(x_{[1:T]}) &= p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_{[1:2]}) \cdots p(x_t|x_{[1:t-1]}) \\ &= \prod_{t=1}^{T} p(x_t|x_{[1:t-1]}) \end{aligned}$$

其中 $x_t \in \mathcal{V}$ 为词表 $\mathcal{V}$ 中的词汇.

给定一个包含 $N$ 个序列数据的数据集 $\mathcal{D} = \{x_{[1:T_n]}^{(n)}\}_{n=1}^{N}$，
序列概率模型需要学习一个模型 $p_\theta(x|x_{[1:t-1]})$ 来最大化整个数据集的对数似然函数:

$$\max_\theta \sum_{n=1}^{N} \log(p_\theta(x_[1:T_n]^{(n)})) = \max_\theta \sum_{n=1}^{N} \sum_{t=1}^{T_n} \log(p_\theta(x_t^{(m)}|x_{[1:t-1]}^{(n)}))$$

在这种序列模型中，每一步都需要将前面的输出作为当前步的输入，
称为**自回归生成模型** (Auto Regressive Generative Model).

---

一旦通过最大似然估计训练了模型 $p_\theta(x|x_{[1:t-1]})$，
就可以通过时间顺序来生成一个完整的序列样本.
令 $\hat{x}_t$ 为在第 $t$ 步根据分布 $p_\theta(x|\hat{x}_{[1:t-1]})$ 生成的词:

$$\hat{x}_t \sim p_\theta(x|\hat{x}_[1:t-1])$$

其中 $\hat{x}_{[1:t-1]} = [\hat{x}_1, \dots, \hat{x}_{t-1}]$ 为前面 $t-1$ 步中生成的前缀序列.

自回归的方式可以生成一个无限长度的序列.
为了避免这种情况，通常会设置一个特殊的符号 $\langle \text{EOS} \rangle$ 来表示序列的结束.
在训练时，每个序列样本的结尾都加上符号 $\langle \text{EOS} \rangle$.
在测试时，一旦生成了符号 $\langle \text{EOS} \rangle$，就中止生成过程.

---

**(束搜索)**
当使用自回归模型生成一个最可能的序列时，生成过程是一种从左到右的贪婪式搜索过程.
在每一步都生成最可能的词:

$$\hat{x}_t = \arg\max_{x \in \mathcal{V}} p_\theta(x|\hat{x}_{[1:t-1]})$$

其中 $\hat{x}_{[1:t-1]} = [\hat{x}_1, \dots, \hat{x}_{t-1}]$ 为前面 $t-1$ 步中生成的前缀序列.
这种贪婪式的搜索方式是次优的，生成的序列 $\hat{x}_{[1:T]}$ 并不保证是全局最优的.

$$\prod_{t=1}^{T} \max_{x_t \in \mathcal{V}} p_\theta(x_t|\hat{x}_{[1:t-1]}) \leq \max_{x_{[1:T]} \in \mathcal{V}^T} \prod_{t=1}^{T} p_\theta(x|x_{[1:t-1]})$$

一种常用的减少搜索错误的启发式方法是**束搜索** (beam search).
在每一步中，生成 $K$ 个最可能的前缀序列，其中超参数 $K$ 为束的大小 (beam size).
下图给出束搜索的示例 (其中 $K = 2$, 词表 $\mathcal{V} = \{A, B, C\}$)

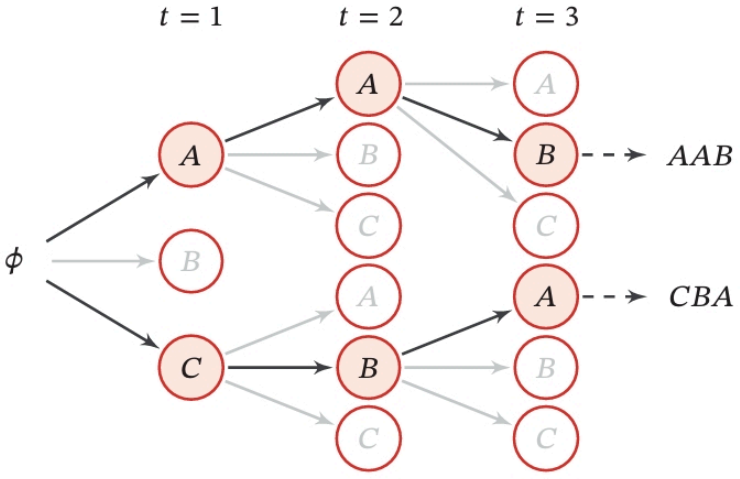



图 15.1　束搜索过程示例

束搜索的过程如下:
在第 1 步时，生成 $K$ 个最可能的词;
在后面每一步中，从 $K|\mathcal{V}|$ 个候选输出中选择 $K$ 个最可能的序列.

束的大小 $K$ 越大，束搜索的复杂度越高，但越有可能生成最优序列.
在实际应用中，束搜索可以通过调整束大小 $K$ 来平衡计算复杂度和搜索质量之间的优先级.

## 5.6.2 N 元统计模型

由于数据稀疏问题，当 $t$ 比较大时，依然很难估计条件概率 $p(x_t|x_{[1:t-1]})$.
一个简化的方法是 $N$ 元统计模型 (N-Gram Model).
假设每个词 $x_t$ 只依赖于其前面的 $N-1$ 个词 ($N$ 阶 Marcov 性质)，即:

$$p_\theta(x_t|x_{[1:t-1]}) = p_\theta(x_t|x_{[t-N+1:t-1]})$$

当 $N=1$ 时，序列 $x_{[1:T]}$ 中每个词都和其他词独立，和其上下文无关.
每个位置上的词都是根据多项分布 $\theta$ 独立生成的，
其中 $\theta = [\theta_1, \dots, \theta_{|\mathcal{V}|}]$ 为词表 $\mathcal{V}$ 中每个词被抽取的概率.
此时序列 $x_{[1:T]}$ 的概率可以写为:

$$p_\theta(x_{[1:T]}) = \prod_{t=1}^{T} p_\theta(x_t) = \prod_{k=1}^{|\mathcal{V}|} \theta_k^{m_k}$$

其中 $m_k$ 为词表 $\mathcal{V}$ 中第 $k$ 个词 $v_k$ 在序列中出现的次数.
由于词的顺序是给定的，故这里没有多项式系数.
可以证明频率估计 $\hat{\theta}_k = \frac{m_k}{\sum_{k=1}^{|\mathcal{V}|} m_k}$ 等价于最大似然估计.

---

对于一般的情况，序列 $x_{[1:T]}$ 的概率可以写为:

$$p_\theta(x_{[1:T]}) = \prod_{t=1}^{T} p_\theta(x_t|x_{[t-N+1:t-1]})$$

可以证明 $N$ 元模型中的条件概率 $p_\theta(x_t|x_{[t-N+1:t-1]})$ 的频率估计等价于最大似然估计:

$$p_\theta(x_t|x_{[t-N+1:t-1]}) = \frac{m(x_{[t-N+1:t]})}{m(x_{[t-N+1:t-1]})}$$

其中 $m(x_{[t-N+1:t]})$ 代表 $x_{[t-N+1:t]}$ 在数据集中出现的次数.

---

**(平滑技术)**
$N$ 元模型的一个主要问题是数据稀疏问题.
数据稀疏问题在基于统计的机器学习中是一个常见的问题，主要是由于训练样本不足而导致概率估计不准确.
在一元模型中，如果一个词 $v$ 在训练数据集中不存在，就会导致任何包含 $v$ 的句子的概率都为 $0$.
同样在 $N$ 元模型中，当一个 $N$ 元组合在训练数据集中不存在时，包含这个组合的句子的概率为 $0$.

数据稀疏问题最直接的解决方法就是增加训练数据集的规模，但其边际效益会随着数据集规模的增加而递减.
以自然语言为例，大多数自然语言都服从 **Zipf 定律** (Zipf's Law):

> 在一个给定自然语言数据集中，一个单词出现的频率与它在频率表里的排名成反比.
> 出现频率最高的单词的出现频率大约是出现频率第二位的单词的 $2$ 倍，大约是出现频率第三位的单词的 $3$ 倍.

因此在自然语言中，大部分的词都是低频词，很难通过增加数据集来避免数据稀疏问题.

数据稀疏问题的一种解决方法是平滑技术 (smoothing)，即给一些没有出现的词组合赋予一定先验概率.
平滑技术是 $N$ 元模型中一项必不可少的技术，例如加法平滑的计算公式为:

$$p(x_t|x_{[t-N+1:t-1]}) = \frac{m(x_{[t-N+1:t]}) + \delta}{m(x_{[t-N+1:t-1]}) + \delta|\mathcal{V}|}$$

其中常数 $\delta \in (0, 1]$ (当 $\delta = 1$ 时，称为**加 $1$ 平滑**)

除了加法平滑，还有很多平滑技术，例如 Good-Turing 平滑、Kneser-Ney 平滑等，
其基本思想都是增加低频词的频率，而降低高频词的频率.

**The End**