第 14 章　序列生成模型

> 人类语言似乎是一种独特的现象，在动物世界中没有显著类似的存在。
>
> — 诺姆 • 乔姆斯基

在深度学习的应用中，有很多数据是以序列的形式存在，比如声音、语言、视频、DNA 序列或者其它的时序数据等。以自然语言为例，一个句子可以看做是符合一定自然语言规则的词（word）的序列。这些语言规则包含非常复杂的语法和语义的组合关系，很难显式地建模这些规则。在认知心理学上有一个经典的实验，当一个人看到下面两个句子：

这里假定语言的最基本单位为词（word），当然也可以为字或字母（character）。

面包上涂黄油
面包上涂袜子

后一个句子在人脑的语义整合时需要更多的处理时间，更不符合自然语言规则。从统计的角度来看，这些语言规则可以看成是一种概率分布。一个长度为 T 的文本序列看作一个随机事件 $X_{1:T} = \langle X_1, \cdots, X_T \rangle$，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 $\mathbf{x}_{1:T}$ 的样本空间为 $|\mathcal{V}|^T$。在某种程度上，自然语言也确实有很多随机因素。比如当我们称赞一个人漂亮时，可以说“美丽”，“帅”或者“好看”等。当不指定使用场合时，这几个词可以交替使用，具体使用哪个词可以看作一个随机事件。

给定一个序列样本 $\mathbf{x}_{1:T} = x_1, x_2, \cdots, x_T$，其概率可以看出是 T 个词的联合概率。

在本章中，我们用 X_t 表示位置 t 上的随机变量，$\mathbf{x}_{1:T}$ 表示一个序列样本，x_t 来表示一个序列样本在位置 t 上的值。

$$P(\mathbf{X}_{1:T} = \mathbf{x}_{1:T}) = P(X_1 = x_1, X_2 = x_2, \cdots, X_T = x_T) \tag{14.1}$$

$$= p(\mathbf{x}_{1:T}). \tag{14.2}$$

和一般的概率模型类似，序列概率模型有两个基本问题：（1）学习问题：给

定一组序列数据，估计这些数据背后的概率分布；（2）生成问题：从已知的序列分布中生成新的序列样本。

序列数据一般可以通过概率图模型来建模序列中不同变量之间的依赖关系，本章主要介绍在序列数据上经常使用的一种模型：自回归生成模型（Autoregressive Generative Model）。

不失一般性，本章以自然语言为例来介绍序列概率模型。

14.1 序列概率模型

序列数据有两个特点：（1）样本是变长的；（2）样本空间为非常大。对于一个长度为T的序列，其样本空间为$|\mathcal{V}|^T$。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。

根据概率的乘法公式，序列$\mathbf{x}_{1:T}$的概率可以写为

$$p(\mathbf{x}_{1:T}) = p(x_1)p(x_2|x_1)p(x_3|\mathbf{x}_{1:2})\cdots p(x_T|\mathbf{x}_{1:(T-1)}) \tag{14.3}$$

$$= \prod_{t=1}^{T} p(x_t|\mathbf{x}_{1:(t-1)}), \tag{14.4}$$

其中$x_t \in \mathcal{V}, t \in [1,T]$为词表$\mathcal{V}$中的一个词，$p(x_1|x_0) = p(x_1)$。

因此，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$\mathbf{x}_{1:(t-1)}$时x_t的条件概率$p(x_t|\mathbf{x}_{1:(t-1)})$。

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

$$\max_\theta \sum_{n=1}^{N} \log p_\theta\big(\mathbf{x}^{(n)}_{1:T_n}\big) = \max_\theta \sum_{n=1}^{N}\sum_{t=1}^{T_n} \log p_\theta\big(x^{(n)}_t|\mathbf{x}^{(n)}_{1:(t-1)}\big). \tag{14.5}$$

在这种序列模型方式中，每一步都需要将前面的输出作为当前步的输入，是一种自回归（autoregressive）的方式。因此这一类模型也称为自回归生成模型（Autoregressive Generative Model）。

自回归模型参见第6.1.2节。

由于$X_t \in \mathcal{V}$为离散变量，我们可以假设条件概率$p_\theta(x_t|\mathbf{x}_{1:(t-1)})$服从多项分布，然后通过不同的模型来估计。本章主要介绍两种比较主流的模型：N元统计模型和深度序列模型。

多项分布参见第D.2.2.2节。

N元统计模型参见第14.2节。

深度序列模型参见第14.3节。

14.1.1 序列生成

一旦通过最大似然估计训练了模型$p_\theta(x|\mathbf{x}_{1:(t-1)})$，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$t$时根据分布$p_\theta(x|\hat{\mathbf{x}}_{1:(t-1)})$生成的词，

$$\hat{x}_t \sim p_\theta(x|\hat{\mathbf{x}}_{1:(t-1)}), \tag{14.6}$$

其中 $\hat{\mathbf{x}}_{1:(t-1)} = \hat{x}_1, \cdots, \hat{x}_{t-1}$ 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“<eos>”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“<eos>”。在测试时，一旦生成了符号“<eos>”，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，

$$\hat{x}_t = \arg\max_{x \in \mathcal{V}} p_\theta(x|\hat{\mathbf{x}}_{1:(t-1)}), \tag{14.7}$$

其中 $\hat{\mathbf{x}}_{1:(t-1)} = \hat{x}_1, \cdots, \hat{x}_{t-1}$ 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 $\hat{\mathbf{x}}_{1:T}$ 并不保证是全局最优的。

$$\prod_{t=1}^{T} \max_{x_t \in \mathcal{V}} p_\theta(x_t|\hat{\mathbf{x}}_{1:(t-1)}) \leq \max_{\mathbf{x}_{1:T} \in \mathcal{V}^T} \prod_{t=1}^{T} p_\theta(x|\mathbf{x}_{1:(t-1)}). \tag{14.8}$$

一种常用的减少搜索错误的启发式方法是*束搜索*（Beam Search）。在每一步的生成中，生成 K 个最可能的前缀序列，其中 K 为束的大小（Beam Size），是一个超参数。

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。图14.1给出了一个束搜索过程的示例，其中词表 $\mathcal{V} = \{A, B, C\}$，束大小为 2。

参见习题14-5。

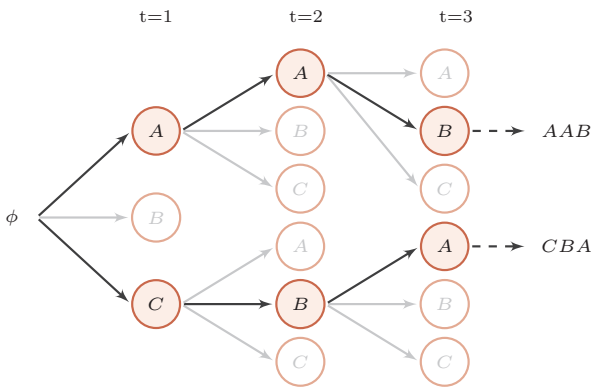


图 14.1 束搜索过程示例

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。

14.2 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$p(x_t|\mathbf{x}_{1:(t-1)})$。一个简化的方法是$N$元模型（N-Gram Model），假设每个词$x_t$只依赖于其前面的$n-1$个词（$n$阶马尔可夫性质），即

马尔可夫性质参见第D.3节。

$$p(x_t|\mathbf{x}_{1:(t-1)}) = p(x_t|\mathbf{x}_{(t-n+1):(t-1)}). \tag{14.9}$$

当$n=1$时，称为一元（unigram）模型；当$n=2$时，称为二元（bigram）模型，以此类推。

一元模型 当$n=1$时，序列$\mathbf{x}_{1:T}$中每个词都和其它词独立，和它的上下文无关。每个位置上的词都是从多项分布独立生成的。在多项分布中，$\theta=[\theta_1,\cdots,\theta_{|\mathcal{V}|}]$为词表中每个词被抽取的概率。

多项分布参见第D.2.2.2节。

在一元模型中，序列$\mathbf{x}_{1:T}$的概率可以写为

$$p(\mathbf{x}_{1:T}|\theta) = \prod_{t=1}^{T} p(x_t) = \prod_{k=1}^{|\mathcal{V}|} \theta_k^{m_k}, \tag{14.10}$$

其中m_k为词表中第k个词v_k在序列中出现的次数。公式(14.10)和标准多项分布的区别是没有多项式系数，因为这里词的顺序是给定的。

给定一组训练集$\{\mathbf{x}^{(n)_{1:T_n}}\}_{n=1}^{N'}$，其对数似然函数为：

$$\log \prod_{n=1}^{N'} p\big(\mathbf{x}_{1:T_n}^{(n)}|\theta\big) = \log \prod_{k=1}^{|\mathcal{V}|} \theta_k^{m_k} \tag{14.11}$$

$$= \sum_{k=1}^{|\mathcal{V}|} m_k \log \theta_k, \tag{14.12}$$

其中m_k为第k个词在整个训练集中出现的次数。

这样一元模型的最大似然估计可以转化为约束优化问题：

$$\max_{\theta} \quad \sum_{k=1}^{|\mathcal{V}|} m_k \log \theta_k \tag{14.13}$$

$$\text{subjuct to} \quad \sum_{k=1}^{|\mathcal{V}|} \theta_k = 1. \tag{14.14}$$

拉格朗日乘子参见第C.3节。

引入拉格朗日乘子λ，定义拉格朗日函数$\Lambda(\theta,\lambda)$为

$$\Lambda(\theta,\lambda) = \sum_{k=1}^{|\mathcal{V}|} m_k \log \theta_k + \lambda \left(\sum_{k=1}^{|\mathcal{V}|} \theta_k - 1 \right). \tag{14.15}$$

令

$$\frac{\partial \Lambda(\theta,\lambda)}{\partial \theta_k} = \frac{m_k}{\theta_k} + \lambda = 0, \tag{14.16}$$

$$\frac{\partial \Lambda(\theta,\lambda)}{\partial \lambda} = \sum_{k=1}^{|\mathcal{V}|} \theta_k - 1 = 0. \tag{14.17}$$

求解上述方程得到$\lambda = -\sum_{v=1}^{V} m_v$，进一步得到

$$\theta_k^{ML} = \frac{m_k}{\sum_{k=1}^{|\mathcal{V}|} m_k} = \frac{m_k}{\bar{m}}, \tag{14.18}$$

其中$\bar{m} = \sum_{k=1}^{|\mathcal{V}|} m_k$为文档集合的长度。由此可见，最大似然估计等价于频率估计。

N元模型 同理，N元模型中的条件概率$p(x_t|\mathbf{x}_{(t-n+1):(t-1)})$也可以通过最大似然函数来得到。

参见习题14-1。

$$p(x_t|\mathbf{x}_{(t-n+1):(t-1)}) = \frac{\mathbf{m}\big(\mathbf{x}_{(t-n+1):t}\big)}{\mathbf{m}\big(\mathbf{x}_{(t-n+1):(t-1)}\big)}, \tag{14.19}$$

其中$\mathbf{m}(\mathbf{x}_{(t-n+1):t})$为$\mathbf{x}_{(t-n+1):t}$在数据集中出现的次数。

N元模型广泛应用于各种自然语言处理问题，如语音识别、机器翻译、拼音输入法，字符识别等。通过N元模型，我们可以计算一个序列的概率，从而判断该序列是否符合自然语言的语法和语义规则。

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密度估计不准确。在一元模型中，如果一个词v在训练数据集里不存在，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，由于大多数自然语言都服从Zipf定律。在一个给定自然语言数据集里，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。

Zipf定律是美国语言学家George K. Zipf提出的实验定律。

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元模型中的一项必不可少的技术，比如*加法平滑*的计算公式为：

$$p\big(x_t|\mathbf{x}_{(t-n+1):(t-1)}\big) = \frac{\mathrm{m}\big(\mathbf{x}_{(t-n+1):t}\big) + \delta}{\mathrm{m}\big(\mathbf{x}_{(t-n+1):(t-1)}\big) + \delta|\mathcal{V}|}, \tag{14.20}$$

其中 $\delta \in (0,1]$ 为常数。$\delta = 1$ 时，称为*加 1 平滑*。

除了加法平滑，还有很多平滑技术，比如 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。

参见习题14-2。

14.3 深度序列模型

深度序列模型（Deep Sequence Model）是指利用神经网络模型来估计条件概率 $p_\theta(x_t|\mathbf{x}_{1:(t-1)})$，

假设一个神经网络 $f(\cdot,\theta)$，其输入为历史信息 $h_t = \mathbf{x}_{1:(t-1)}$，输出为词表 $\mathcal{V}$ 中的每个词 $v_k (1 \leq k \leq |\mathcal{V}|$ 出现的概率，并满足

$$\sum_{k=1}^{|\mathcal{V}|} f_k\big(\mathbf{x}_{1:(t-1)}, \theta\big) = 1, \tag{14.21}$$

其中 θ 表示网络参数。条件概率 $p_\theta(x_t|\mathbf{x}_{1:(t-1)})$ 可以从神经网络的输出中得到，

$$p_\theta(x_t|\mathbf{x}_{1:(t-1)}) = f_{k_{x_t}}(\mathbf{x}_{1:(t-1)}; \theta), \tag{14.22}$$

其中 k_{x_t} 为 x_t 在词表 $\mathcal{V}$ 中索引。

深度序列模型一般可以分为三个部分：嵌入层、特征层、输出层。

嵌入层 令 $h_t = \mathbf{x}_{1:(t-1)}$ 表示输入的历史信息，一般为符号序列。由于神经网络模型一般要求输入形式为实数向量，因此为了能够使得神经网络模型能处理符号数据，需要将这些符号转换为向量形式。一种简单的转换方法是通过一个*嵌入表*（Embedding Lookup Table）来将每个符号直接映射成向量表示。嵌入表也称为*嵌入矩阵*或*查询表*。令 $M \in \mathbb{R}^{d_1 \times |\mathcal{V}|}$ 为嵌入矩阵，其中第 k 列向量 $\mathbf{m}_k \in \mathbb{R}^{d_1}$ 表示词表中第 k 个词对应的向量表示。

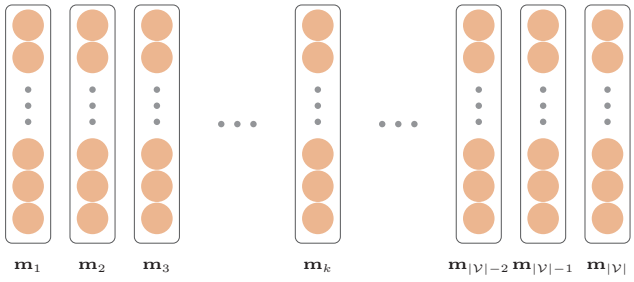


图 14.2 嵌入矩阵

假设词 x_t 对应词表中的索引为 k，则其 one-hot 向量表示为 $\delta_t \in \{0,1\}^{|\mathcal{V}|}$，即第 k 维为 1，其余为 0 的 $|\mathcal{V}|$ 维向量。词 x_t 对应的向量表示为

$$\mathbf{e}_t = M\delta_t = \mathbf{m}_k. \tag{14.23}$$

通过上面的映射可以得到序列 $x_{1:(t-1)}$ 对应的向量序列 $\mathbf{e}_1, \cdots, \mathbf{e}_{t-1}$。

特征层 特征层用于从输入向量序列 $\mathbf{e}_1, \cdots, \mathbf{e}_{t-1}$ 中提取特征，输出为一个可以表示历史信息的向量 $\mathbf{h}_t$。

特征层可以通过不同类型的神经网络来实现，比如前馈神经网络和循环神经网络。常见的网络类型有以下三种：

（1）**简单平均**

$$\mathbf{h}_t = \sum_{i=1}^{t-1} \alpha_i \mathbf{e}_i, \tag{14.24}$$

其中 α_i 为每个词的权重。

权重 α_i 可以和位置 i 及其表示 $\mathbf{e}_i$ 相关，也可以无关。简单起见，可以设置 $\alpha_i = \frac{1}{t-1}$。权重 α_i 也可以通过注意力机制来动态计算。

注意力机制参见第8.1.2节。参见习题14-3。

（2）**前馈神经网络**

前馈神经网络要求输入的大小是固定的。因此和 N 元模型类似，假设历史信息只包含前面 $n-1$ 个词。

首先将这 $n-1$ 个的词向量的 $\mathbf{e}_{t-n+1}, \cdots, \mathbf{e}_{t-1}$ 拼接成一个 $d_1 \times (n-1)$ 维的向量 $\mathbf{h}'$。

$$\mathbf{h}' = \mathbf{e}_{t-n+1} \oplus \cdots \oplus \mathbf{e}_{t-1}, \tag{14.25}$$

其中 $\oplus$ 表示向量拼接操作。

然后将 $\mathbf{h}'$ 输入到由前馈神经网络构成的隐藏层，最后一层隐藏层的输出 $\mathbf{h}_t$。

$$\mathbf{h}_t = g(\mathbf{h}', \theta_g), \tag{14.26}$$

其中 $g(\cdot, \theta_g)$ 可以为全连接的前馈神经网络或卷积神经网络，θ_g 为网络参数。

为了增加特征的多样性，前馈网络中也可以包含跳层连接（Skip-Layer Connections）[Bengio et al., 2003]，比如

$$\mathbf{h}_t = \mathbf{e}_t \oplus g(\mathbf{h}', \theta_g). \tag{14.27}$$

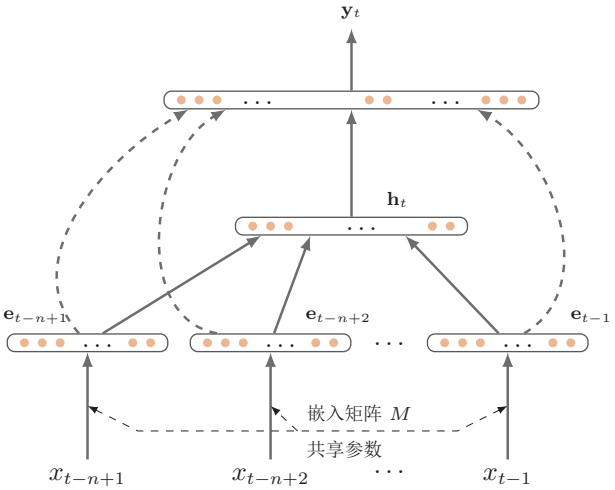
（3）**循环神经网络**

和前馈神经网络不同，循环神经网络可以接受变长的输入序列，依次接受输入 $\mathbf{e}_1, \cdots, \mathbf{e}_{t-1}$，得到时刻 t 的隐藏状态 h_t

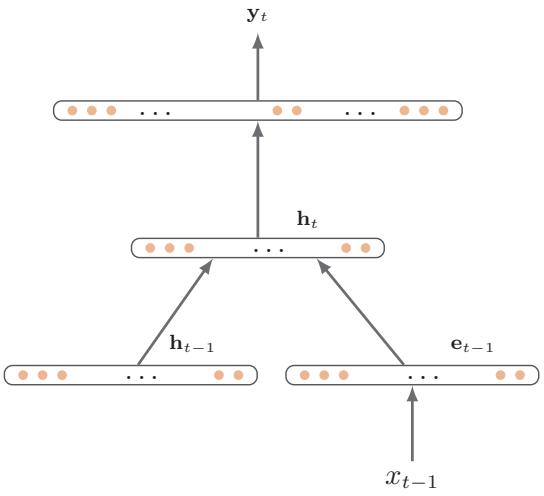
$$\mathbf{h}_t = g(\mathbf{h}_{t-1}, \mathbf{e}_t, \theta_g), \tag{14.28}$$

其中 $g(\cdot)$ 为一个非线性函数，θ_g 为循环网络的参数，$\mathbf{h}_0 = 0$。

前馈网络模型和循环网络模型的不同之处在于循环神经网络利用隐藏状态来记录以前所有时刻的信息，而前馈神经网络只能接受前 $n-1$ 个时刻的信息。



(a) 前馈神经网络模型



(b) 循环神经网络模型

图 14.3 深度序列模型

输出层 输出层为一般使用softmax分类器，接受历史信息的向量表示$\mathbf{h}_t \in \mathbb{R}^{d_2}$，输出为词表中每个词的后验概率，输出大小为$|\mathcal{V}|$。

$$\mathbf{o}_t = \text{softmax}(\hat{\mathbf{o}}_t) \tag{14.29}$$

$$= \text{softmax}(W\mathbf{h}_t + \mathbf{b}), \tag{14.30}$$

其中输出向量$\mathbf{o}_t \in (0,1)^{|\mathcal{V}|}$为预测的概率分布，第$k$维是词表中第$k$个词出现的条件概率；$\hat{\mathbf{o}}_t$是为归一化的得分向量；$W \in \mathbb{R}^{|\mathcal{V}| \times d_2}$是最后一层隐藏层到输出层直接的权重矩阵，$\mathbf{b} \in \mathbb{R}^{|\mathcal{V}|}$为偏置。

图14.3给出了两种不同的深度序列模型，图14.3a为前馈网络模型（虚线边为可选的跳层连接），图14.3b为循环神经网络模型。

14.3.1 参数学习

给定一个训练序列$\mathbf{x}_{1:T}$，深度序列模型的训练目标为找到一组参数θ，使得对数似然函数最大。

简要起见，这里忽略了正则化项。

$$\log p_\theta(\mathbf{x}_{1:T}) = \sum_{t=1}^{T} \log p_\theta(x_t|\mathbf{x}_{1:(t-1)}), \tag{14.31}$$

其中θ表示网络中的所有参数，包括嵌入矩阵M以及神经网络的权重和偏置。

网络参数一般通过梯度上升法来学习，

$$\theta \leftarrow \theta + \alpha \frac{\partial \log p_\theta(\mathbf{x}_{1:T})}{\partial \theta}, \tag{14.32}$$

其中α为学习率。

14.4 评价方法

构造一个序列生成模型后，需要有一个度量来评价其好坏。

14.4.1 困惑度

给定一个测试文本集合，一个好的序列生成模型应该使得测试集合中的句子的联合概率尽可能高。

困惑度（Perplexity）是信息论的一个概念，可以用来衡量一个分布的不确定性。对于离散随机变量$X \in \mathcal{X}$，其概率分布为$p(x)$，困惑度为

$$2^{H(p)} = 2^{-\sum_{x \in \mathcal{X}} p(x) \log_2 p(x)}, \tag{14.33}$$

其中$H(p)$为分布p的熵。

困惑度也可以用来衡量两个分布之间差异。对于一个未知的数据分布 $p_r(x)$ 和一个模型分布 $p_\theta(x)$，我们从 $p_r(x)$ 中采样出一组测试样本 $x^{(1)}, \cdots, x^{(N)}$，模型分布 $p_\theta(x)$ 的困惑度为

$$2^{H(\tilde{p}_r, p_\theta)} = 2^{-\frac{1}{N}\sum_{n=1}^{N} \log_2 p_\theta(x^{(n)})}, \tag{14.34}$$

其中 $H(\tilde{p}_r, p_\theta)$ 为样本的经验分布 $\tilde{p}_r$ 与模型分布 p_θ 之间的交叉熵，也是所有样本上的负对数似然函数。

困惑度可以衡量模型分布与样本经验分布之间的契合程度。困惑度越低则两个分布越接近。因此模型分布 $p_\theta(x)$ 的好坏可以用困惑度来评价。

假设测试集合共有独立同分布的 N 个序列 $\{\mathbf{x}_{1:T_n}^{(n)}\}_{n=1}^{N}$。我们可以用模型 $p_\theta(\mathbf{x})$ 对每个序列计算其概率 $p_\theta(\mathbf{x}_{1:T_n}^{(n)})$，整个测试集的联合概率为

$$\prod_{n=1}^{N} p_\theta\big(\mathbf{x}_{1:T_n}^{(n)}\big) = \prod_{n=1}^{N}\prod_{t=1}^{T_n} p_\theta\big(x_t^{(n)} | \mathbf{x}_{1:(t-1)}^{(n)}\big). \tag{14.35}$$

模型 $p_\theta(\mathbf{x})$ 的困惑度定义为

$$\begin{aligned}
\mathrm{PPL}(\theta) &= 2^{-\frac{1}{T}\sum_{n=1}^{N} \log p_\theta\left(\mathbf{x}_{1:T_n}^{(n)}\right)} && (14.36)\\
&= 2^{-\frac{1}{T}\sum_{n=1}^{N}\sum_{t=1}^{T_n} \log p_\theta\left(x_t^{(n)} | \mathbf{x}_{1:(t-1)}^{(n)}\right)} && (14.37)\\
&= \left(\prod_{n=1}^{N}\prod_{t=1}^{T_n} p_\theta\big(x_t^{(n)} | \mathbf{x}_{1:(t-1)}^{(n)}\big)\right)^{-1/T}, && (14.38)
\end{aligned}$$

其中 $T = \sum_{n=1}^{N} T_n$ 为测试数据集中序列的总长度。可以看出，困惑度为每个词条件概率 $p_\theta\big(x_t^{(n)} | \mathbf{x}_{1:(t-1)}^{(n)}\big)$ 的几何平均数的倒数。测试集中所有序列的概率越大，困惑度越小，模型越好。

几何平均数是一种求数值平均数的方法，计算公式为：$\bar{x} = \sqrt[n]{\prod_{t=1}^{n} x_t}$.

假设一个序列模型赋予每个词出现的概率均等，即 $p_\theta\big(x_t^{(n)} | \mathbf{x}_{1:(t-1)}^{(n)}\big) = \frac{1}{|\mathcal{V}|}$，则该模型的困惑度为 $|\mathcal{V}|$。以英语为例，N 元模型的困惑度范围一般为 $50 \sim 1000$ 之间。

14.4.2 BLEU

BLEU（Bilingual Evaluation Understudy）是衡量模型生成序列和参考序列之间的 N 元词组（N-Gram）的重合度，最早用来评价机器翻译模型的质量，目前也广泛应用在各种序列生成任务中。

假设从模型分布 p_θ 中生成一个候选（Candidate）序列 $\mathbf{x}$，从真实数据分布中采样出的一组参考（Reference）序列 $\mathbf{s}^{(1)}, \cdots, \mathbf{s}^{(K)}$，我们首先从生成序列中

提取N元组合的集合$\mathcal{W}$，并计算N元组合的精度（Precision），

$$P_n(\mathbf{x}) = \frac{\sum\limits_{w\in\mathcal{W}} \min\left(c_w(\mathbf{x}), \max\limits_{k=1}^{K} c_w(\mathbf{s}^{(k)})\right)}{\sum\limits_{w\in\mathcal{W}} c_w(\mathbf{x})}, \tag{14.39}$$

其中$c_w(\mathbf{x})$是N元组合w在生成序列$\mathbf{x}$中出现的次数，$c_w(\mathbf{s}^{(k)})$是N元组合w在参考序列$\mathbf{s}^{(k)}$中出现的次数。N元组合的精度$P_n(\mathbf{x})$是计算生成序列中的N元组合有多少比例在参考序列中出现。

由于精度只衡量生成序列中的N元组合是否在参考序列中出现，生成序列越短，其精度会越高，因此可以引入长度惩罚因子(Brevity Penalty)。如果生成序列的长度短于参考序列，就对其进行惩罚。

$$b(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad l_x > l_s \\ \exp\left(1 - l_s/l_x\right) & \text{if} \quad l_x \leq l_s \end{cases} \tag{14.40}$$

其中l_x为生成序列$\mathbf{x}$的长度，l_s为参考序列的最短长度。

BLEU是通过计算不同长度的N元组合的精度，并进行几何加权平均而得到。

$$\text{BLEU-N}(\mathbf{x}) = b(\mathbf{x}) \times \exp\big(\sum_{n=1}^{N} w_n \log P_n\big), \tag{14.41}$$

其中w_n为不同N元组合的权重，一般设为$\frac{1}{N}$。BLEU取值范围是$[0,1]$，越大表明生成的质量越好。但是BLEU只计算精度，而不关心召回率（即参考序列里的N元组合是否在生成序列中出现）。

参见习题14-4。

14.4.3 ROUGE

ROUGE（Recall-Oriented Understudy for Gisting Evaluation）最早应用于文本摘要领域。和BLEU类似，但ROUGE计算的是召回率（Recall）。

假设从模型分布p_θ中生成一个候选（Candidate）序列$\mathbf{x}$，从真实数据分布中采样出的一组参考（Reference）序列$\mathbf{s}^{(1)}, \cdots, \mathbf{s}^{(K)}$，令$\mathcal{W}$为从参考序列中提取N元组合的集合，ROUGE-N的定义为

$$\text{ROUGE-N}(\mathbf{x}) = \frac{\sum\limits_{k=1}^{K}\sum\limits_{w\in\mathcal{W}} \min\left(c_w(\mathbf{x}), c_w\left(\mathbf{s}^{(k)}\right)\right)}{\sum\limits_{k=1}^{K}\sum\limits_{w\in\mathcal{W}} c_w\left(\mathbf{s}^{(k)}\right)}, \tag{14.42}$$

其中$c_w(\mathbf{x})$是N元组合w在生成序列$\mathbf{x}$中出现的次数，$c_w(\mathbf{s}^{(k)})$是N元组合w在参考序列$\mathbf{s}^{(k)}$中出现的次数。

14.5 序列生成模型中的学习问题

使用最大似然估计来学习自回归序列生成模型时，会存在以下三个主要问题：曝光偏差、训练目标不一致和计算效率。下面我们分别介绍这三个问题以及解决方面。

14.5.1 曝光偏差问题

在自回归生成模型中，第 t 步的输入为模型生成的前缀序列 $\hat{\mathbf{x}}_{1:(t-1)}$。而在训练时，我们使用的前缀序列是训练集中的真实数据 $\mathbf{x}_{1:(t-1)}$），而不是模型预测的 $\hat{\mathbf{x}}_{1:(t-1)}$）。这种学习方式也称为教师强制（Teacher Forcing）[Williams and Zipser, 1989]。

模型生成的分布 $p_\theta\big(\mathbf{x}_{1:(t-1)})\big)$ 和真实数据分布 $p_r\big(\mathbf{x}_{1:(t-1)})\big)$ 并不严格一致，因此条件概率 $p_\theta\big(x|\mathbf{x}_{1:(t-1)})\big)$ 在训练和测试会存在协变量偏移问题。一旦在预测前缀 $\hat{\mathbf{x}}_{1:(t-1)}$) 的过程中存在错误，会导致错误传播，使得后续生成的序列也会偏离真实分布。这个问题称为曝光偏差（Exposure Bias）。

协变量偏移问题参见第10.4.0.2节。

计划采样 为了缓解曝光偏差的问题，我们可以在训练时混合使用真实数据和模型生成数据。在第 t 步时，模型随机使用真实数据 x_{t-1} 或前一步生成的词 $\hat{x}_{t-1}$ 作为输入。

令 $\epsilon \in [0, 1]$ 为一个控制替换率的超参数，在每一步时，以 ϵ 的概率使用真实数据 x_{t-1}，以 $1-\epsilon$ 的概率来使用生成数据 $\hat{x}_{t-1}$。当令 $\epsilon = 1$ 时，训练和最大似然估计一样，使用真实数据；当令 $\epsilon = 0$ 时，训练时全部使用模型生成数据。

直觉上，如果一开始训练时的 ϵ 过小，模型相当于在噪声很大的数据上训练，会导致模型性能变差，并且难以收敛。因此，一个较好的策略是在训练初期赋予 ϵ 较大的值，随着训练次数的增加逐步减少 ϵ 的取值。这种策略称为计划采样（Scheduled Sampling）[Bengio et al., 2015]。

令 ϵ_i 为在第 i 次迭代时的替换率，在计划采样中可以通过下面几种方法来逐步降低 ϵ 的取值。

1. 线性衰减：$\epsilon_i = \max(\epsilon, k - ci)$，其中 ϵ 为最小的替换率，k 和 c 分别为初始值和衰减率。
2. 指数衰减：$\epsilon_i = k^i$，其中 $k < 1$ 为初始替换率。
3. 逆 Sigmoid 衰减：$\epsilon_i = k/(k + \exp(i/k))$，其中 $k \geq 1$ 来控制衰减速度。

计划采样的一个缺点是在每一步中不管输入如何选择，目标输出依然是来自于真实数据。这可能使得模型预测一些不正确的序列。比如一个真实的序列

是“吃饭”，如果在第一步生成时使用模型预测的词“喝”，模型就会强制记住“喝饭”这个不正确的序列。

14.5.2 训练目标不一致问题

序列生成模型一般是采用和任务相关的指标来进行评价，比如BLEU、ROUGE等，而训练时是使用最大似然估计，这导致训练目标和评价方法不一致。并且这些评价指标一般都是不可微的，无法直接使用基于梯度的方法来进行优化。

基于强化学习的序列生成 为了可以直接优化评价目标，我们可以将自回归的序列生成可以看作是一种马尔可夫决策过程，并使用强化学习的方法来进行训练。

参见第14.1.3节。

在第t步，动作a_t可以看作是从词表中选择一个词，策略为$\pi_\theta(a|s_t)$，其中状态s_t为之前步骤中生成的前缀序列$\mathbf{x}_{1:(t-1)}$。一个序列$\mathbf{x}_{1:T}$可以看作是马尔克夫决策过程的一个轨迹（trajectory）

$$\tau = \{s_1, a_1, s_2, a_2 ..., s_T, a_T\}. \tag{14.43}$$

轨迹τ的概率为

$$p_\theta(\tau) = \prod_{t=1}^{T} \pi_\theta\big(a_t = x_t | s_t = x_{1:(t-1)}\big), \tag{14.44}$$

其中状态转移概率$p\big(s_t = \mathbf{x}_{1:t} | s_{t-1} = \mathbf{x}_{1:(t-1)}, a_{t-1} = x_{t-1}\big) = 1$是确定性的，可以被忽略。

强化学习的目标是学习一个策略$\pi_\theta(a|s_t)$使得期望回报最大，

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[G(\tau)] \tag{14.45}$$

$$= \mathbb{E}_{\mathbf{x}_{1:T} \sim p_\theta(\mathbf{x}_{1:T})}[G(\mathbf{x}_{1:T})], \tag{14.46}$$

其中$G(\mathbf{x}_{1:T})$为序列$\mathbf{x}_{1:T}$的总回报，可以为BLEU、ROUGE或其它评价指标。

这样序列生成问题就转换为强化学习问题，其策略函数$\pi_\theta(a|s_t)$可以通过REINFORCE算法或Actor-Critic算法来进行学习。为了改进强化学习的效率，策略函数$\pi_\theta(a|s_t)$一般会通过最大似然估计来进行预训练。

基于强化学习的序列生成模型不但可以解决训练和评价目标不一致问题，也可以有效地解决曝光偏差问题。

14.5.3 计算效率问题

序列生成模型的输出层为词表中所有词的条件概率，需要softmax归一化。当词表比较大时，计算效率比较低。

在第 t 步时，前缀序列为 $h_t = \mathbf{x}_{1:(t-1)}$，词 x_t 的条件概率为

$$p_\theta(x_t|h_t) = \text{softmax}\Big(s(x_t, h_t; \theta)\Big) \tag{14.47}$$

$$= \frac{\exp\big(s(x_t, h_t; \theta)\big)}{\sum_{v\in\mathcal{V}} \exp\big(s(v, h_t; \theta)\big)}, \tag{14.48}$$

$$= \frac{\exp\big(s(x_t, h_t; \theta)\big)}{Z(h_t; \theta)}, \tag{14.49}$$

$s(x_t, h_t; \theta) = [\hat{\mathbf{o}}_t]_{k_{x_t}}$，$\hat{\mathbf{o}}_t$ 为未归一化的网络输出参见公式 (14.29)。

参见第11.1.4节。

其中 $s(x_t, h_t; \theta)$ 为未经过softmax归一化的得分函数，$Z(h_t; \theta)$ 为配分函数（Partition Function）。

$$Z(h_t; \theta) = \sum_{v\in\mathcal{V}} \exp\big(s(v, h_t; \theta)\big). \tag{14.50}$$

配分函数的计算需要对词表中所有的词 v 计算 $s(v, h_t; \theta)$ 并求和。当词表比较大时，计算开销非常大。比如在自然语言中，词表 $\mathcal{V}$ 的规模一般在 1 万到 10 万之间。在训练时，每个样本都要计算一次配分函数，这样每一轮迭代需要计算 T 次配分函数（T 为训练文本长度），导致整个训练过程变得十分耗时。因此在实践中，我们通常采用一些近似估计的方法来加快训练速度。常用的方法可以分为两类：（1）层次化的softmax计算，将标准softmax函数的扁平结构转换为层次化结构；（2）基于采样的方法，通过采样来近似计算更新梯度。

本节介绍三种方法加速训练的方法：层次化 softmax 方法、重要性采样和噪声对比估计。

14.5.3.1 层次化 Softmax

我们先来考虑两层的来组织词表，即将词表中词分成 k 组，并每一个词只能属于一个分组，每组大小为 $\frac{|\mathcal{V}|}{k}$。假设词 w 所属的组为 $c(w)$，则

$$p(w|h) = p(w, c(w)|h) \tag{14.51}$$

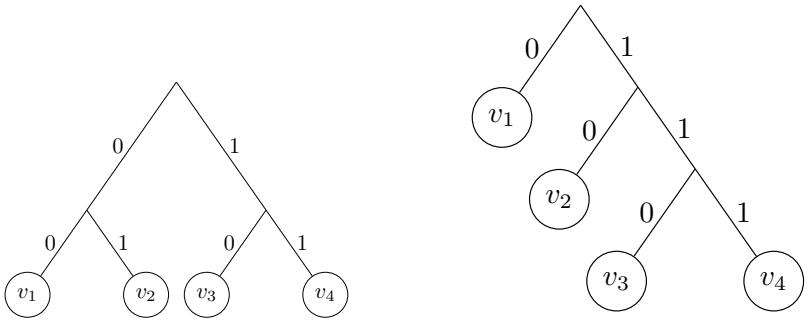
$$= p(w|c(w), h)p(c(w)|h), \tag{14.52}$$

其中 $p(c(w)|h)$ 是给定历史信息 h 条件下，类 $c(w)$ 的后验概率，$p(w|c(w), h)$ 是给定历史信息 h 和类 $c(w)$ 条件下，词 w 的后验概率。因此，一个词的概率可以分解为两个概率 $p(w|c(w), h)$ 和 $p(c(w)|h)$ 的乘积，它们可以分别利用神经网络来估计，这样计算softmax函数时分别只需要做 $\frac{|\mathcal{V}|}{k}$ 和 k 次求和，从而就大大提高了 softmax 函数的计算速度。

一般对于词表大小 $|\mathcal{V}|$，我们将词平均分到 $\sqrt{|\mathcal{V}|}$ 个分组中，每组 $\sqrt{|\mathcal{V}|}$ 个词。这样通过一层的分组，我们可以将softmax计算加速 $\frac{1}{2}\sqrt{|\mathcal{V}|}$ 倍。比如当词

表大小为40,000时，将词表中所有词分到200组，每组200个词。这样只需要计算两次200类的softmax，比直接计算40,000类的softmax加快100倍。

为了进一步降低softmax的计算复杂度，我们可以更深层的树结构来组织词汇表。假设用二叉树来组织词表中的所有词，二叉树的叶子节点代表词表中的词，非叶子节点表示不同层次上的类别。图14.4给出了平衡二叉树和Huffman二叉树的示例。



(a) 平衡树 (b) Huffman 树

图 14.4 层次化树结构

如果我们将二叉树上所有左链接标记为0，右链接标记为1。每一个词可以用根节点到它所在的叶子之间路径上的标记来进行编码。图14.4a中所示的四个词的编码分别为：

$$v_1 = 00, \qquad v_2 = 01, \qquad v_3 = 10, \qquad v_4 = 10. \tag{14.53}$$

假设词v在二叉树上从根节点到其所在叶子节点的路径长度为m，其编码可以表示一个位向量（bit vector）：$[b_1, \cdots, b_m]^{\mathrm{T}}$。词$v$的条件概率为

$$P(v|h) = p\big(b_1, \cdots, b_m|h\big) \tag{14.54}$$

$$= \prod_{j=1}^{m} p\big(b_j|b_1, \cdots, b_{j-1}, h\big), \tag{14.55}$$

$$= \prod_{j=1}^{m} p\big(b_j|b_{j-1}, h\big), \tag{14.56}$$

由于$b_j \in \{0, 1\}$为二值变量，$p(b_j|b_{j-1}, h)$可以看作是两类分类问题，可以使用logistic回归来进行预测。

$$p(b_j = 1|b_{j-1}, h) = \sigma\big(\mathbf{w}_{n(b_{j-1})}^{\mathrm{T}}\mathbf{h} + \mathbf{b}_{n(b_{j-1})}\big), \tag{14.57}$$

其中 $n(b_{j-1})$ 为词 v 在树 T 上的路径上的第 $j-1$ 个节点。

若使用平衡二叉树来进行分组，则条件概率估计可以转换为 $\log_2 |\mathcal{V}|$ 个两类分类问题。这时 softmax 函数可以用 logistic 函数代替，计算效率可以加速 $\frac{|\mathcal{V}|}{\log_2 |\mathcal{V}|}$ 倍。

将词表中的词按照树结构进行组织，有以下几种转换方式：

WordNet 是按照词义来组织的英语词汇知识库，由 Princeton 大学研发。

- 利用人工整理的词汇层次结构，比如利用 WordNet[Miller, 1995] 系统中的“IS-A”关系（即上下位关系）。例如，“狗”是“动物”的下位词。因为 WordNet 的层次化结构不是二叉树，因此需要通过进一步聚类来转换为二叉树。

Huffman 编码是 David Huffman 于 1952 年发明的一种用于无损数据压缩的熵编码（权编码）算法。

- 使用 Huffman 编码。Huffman 编码对出现概率高的词使用较短的编码，出现概率低的词则使用较长的编码。因此训练速度会更快。Huffman 编码的算法如算法14.1所示。

算法 14.1: Huffman 树构建算法

输入: 词表: $\mathcal{V}$

初始化：为每个词 v 建立一个叶子节点，其概率为词的出现频率;
将所有的叶子节点放入集合 $\mathcal{S}$ 中;
while $|\mathcal{S}| > 1$ **do**
    从集合 $\mathcal{S}$ 选择两棵概率最低的节点 n_1 和 n_2;
    构建一个新节点 n'，并将 n_1 和 n_2 作为 n' 的左右子节点;
    新节点 n' 的概率 n_1 和 n_2 的概率之和，;
    将新二叉树加入集合 $\mathcal{S}$ 中，并把 n_1 和 n_2 从集合 $\mathcal{S}$ 中移除;
end
集合 $\mathcal{S}$ 中最后一个节点为 n;

输出: 以 n 为根节点的二叉树 T

14.5.3.2 重要性采样

另一种加速训练速度的方法是基于采样的方法，即通过采样来近似计算训练时的梯度。

参见第11.3节。

用随机梯度上升来更新参数 θ 时，第 t 个样本 (h_t, x_t) 的目标函数关于 θ 的梯度为

$$\frac{\partial \log p_\theta(x_t|h_t)}{\partial \theta} = \frac{\partial s(x_t, h_t; \theta)}{\partial \theta} - \frac{\partial \log \left(\sum_v \exp \left(s(v, h_t; \theta) \right) \right)}{\partial \theta} \tag{14.58}$$

$$= \frac{\partial s(x_t, h_t; \theta)}{\partial \theta} - \frac{1}{\sum_v \exp\big(s(v, h_t; \theta)\big)} \cdot \frac{\partial \sum_v \exp\big(s(v, h_t; \theta)\big)}{\partial \theta} \tag{14.59}$$

$$= \frac{\partial s(x_t, h_t; \theta)}{\partial \theta} - \sum_v \frac{1}{\sum_w \exp\big(s(w, h_t; \theta)\big)} \cdot \frac{\partial \exp\big(s(v, h_t; \theta)\big)}{\partial \theta} \tag{14.60}$$

$$= \frac{\partial s(x_t, h_t; \theta)}{\partial \theta} - \sum_v \frac{\exp\big(s(v, h_t; \theta)\big)}{\sum_w \exp\big(s(w, h_t; \theta)\big)} \frac{\partial s(v, h_t; \theta)}{\partial \theta} \tag{14.61}$$

$$= \frac{\partial s(x_t, h_t; \theta)}{\partial \theta} - \sum_v p_\theta(v|h_t) \frac{\partial s(v, h_t; \theta)}{\partial \theta} \tag{14.62}$$

$$= \frac{\partial s(x_t, h_t; \theta)}{\partial \theta} - \mathbb{E}_{p_\theta(v|h_t)} \left[\frac{\partial s(v, h_t; \theta)}{\partial \theta} \right]. \tag{14.63}$$

公式（14.63）中最后一项是计算 $\frac{\partial}{\partial \theta} s(v, h_t; \theta)$ 在分布 $p_\theta(v|h_t)$ 下的期望。从公式（14.61）中可以看出，在计算每个样本的梯度时需要在整个词表上计算两次求和。一次是求配分函数 $\sum_w \exp\big(s(w, h_t; \theta)\big)$，另一次是计算所有词的梯度的期望 $\mathbb{E}[\frac{\partial}{\partial \theta} s(v, h_t; \theta)]$。由于自然语言中的词表都比较大，训练速度会非常慢。

为了提高训练效率，可以用*采样方法*来进行近似地估计公式 (14.63) 中的期望。但是我们不能使用直接根据分布 $p_\theta(v|h_t)$ 进行采样，因为直接采样需要在需要先计算分布 $p_\theta(v|h_t)$，而这正是我们希望避免的。

重要性采样参见第11.3.3节。

*重要性采样*是用一个容易采样的*提议分布* q，来近似估计分布 p。

公式 (14.63) 中最后一项可以写为：

$$\mathbb{E}_{p_\theta(v|h_t)} \left[\frac{\partial s(v, h_t; \theta)}{\partial \theta} \right] = \sum_{v \in \mathcal{V}} p_\theta(v|h_t) \frac{\partial s(v, h_t; \theta)}{\partial \theta} \tag{14.64}$$

$$= \sum_{v \in \mathcal{V}} q(v|h_t) \frac{p_\theta(v|h_t)}{q(v|h_t)} \cdot \frac{\partial s(v, h_t; \theta)}{\partial \theta} \tag{14.65}$$

$$= \mathbb{E}_{q(v|h_t)} \left[\frac{p_\theta(v|h_t)}{q(v|h_t)} \cdot \frac{\partial s(v, h_t; \theta)}{\partial \theta} \right]. \tag{14.66}$$

这样原始分布 $p_\theta(v|h_t)$ 上的期望转换为提议分布 $q(v|h_t)$ 上的期望。提议分布 q 需要尽可能和 $p_\theta(v|h_t)$ 接近，并且从 $q(v|h_t)$ 采样的代价要比较小。在实践中，提议分布 $q(v|h_t)$ 可以采用 N 元模型的分布函数。

根据分布 $Q(v|h_t)$ 独立采样 K 个样本 $v_1, \cdots, v_K$ 来近似求解公式（14.66）。

$$\mathbb{E}_{p_\theta(v|h_t)} \left[\frac{\partial s(v, h_t; \theta)}{\partial \theta} \right] \approx \frac{1}{K} \sum_{k=1}^{K} \frac{p_\theta(v_k|h_t)}{q(v_k|h_t)} \frac{\partial s(v_k, h_t; \theta)}{\partial \theta}. \tag{14.67}$$

在公式 (14.67) 中，依然需要每一个计算抽取样本的概率 $p_\theta(v_k|h)$。

$$p_\theta(v_k|h_t) = \frac{s(v_k, h_t; \theta)}{Z(h_t)}, \tag{14.68}$$

其中 $Z(h_t) = \sum_w \exp\big(s(w, h_t; \theta)\big)$ 为配分函数，需要在所有样本上进行计算 $s(w, h_t; \theta)$ 并求和。为了避免这种情况，我们可以进一步把配分函数 $Z(h_t)$ 的计算也使用重要性采样来计算。

$$Z(h_t) = \sum_w \exp\big(s(w, h_t; \theta)\big) \tag{14.69}$$

$$= \sum_w q(w|h_t) \frac{1}{q(w|h_t)} \exp\big(s(w, h_t; \theta)\big) \tag{14.70}$$

$$= \mathbb{E}_{q(w|h_t)} \left[\frac{1}{q(w|h_t)} \exp\big(s(w, h_t; \theta)\big) \right] \tag{14.71}$$

通过采样近似估计

$$\approx \frac{1}{K} \sum_{k=1}^{K} \frac{1}{q(v_k|h_t)} \exp\big(s(v_k, h_t; \theta)\big) \tag{14.72}$$

$$= \frac{1}{K} \sum_{k=1}^{K} \frac{\exp\big(s(v_k, h_t; \theta)\big)}{q(v_k|h_t)} \tag{14.73}$$

$$= \frac{1}{K} \sum_{k=1}^{K} r(v_k), \tag{14.74}$$

其中 $r(v_k) = \frac{\exp\left(s(v_k, h_t; \theta)\right)}{q(v_k|h_t)}$，$q(v_k|h_t)$ 为提议分布。为了提高效率，可以和公式（14.67）中的提议分布设为一致，并复用在上一步中抽取的样本。

在近似估计了配分函数以及梯度期望之后，公式（14.67）可写为

$$\mathbb{E}_{p_\theta(v|h_t)} \left[\frac{\partial s(v, h_t; \theta)}{\partial \theta} \right] \approx \frac{1}{K} \sum_{k=1}^{K} \frac{p_\theta(v_k|h_t)}{q(v_k|h_t)} \frac{\partial s(v_k, h_t; \theta)}{\partial \theta} \tag{14.75}$$

$$= \frac{1}{K} \sum_{k=1}^{K} \frac{\exp\big(s(v_k, h_t; \theta)\big)}{Z(h_t)} \frac{1}{q(v_k|h_t)} \frac{\partial s(v_k, h_t; \theta)}{\partial \theta} \tag{14.76}$$

$$= \frac{1}{K} \sum_{k=1}^{K} \frac{1}{Z(h_t)} r(v_k) \frac{\partial s(v_k, h_t; \theta)}{\partial \theta} \tag{14.77}$$

$$\approx \sum_{k=1}^{K} \frac{r(v_k)}{\sum_{k=1}^{K} r(v_k)} \frac{\partial s(v_k, h_t; \theta)}{\partial \theta} \tag{14.78}$$

$$= \frac{1}{\sum_{k=1}^{K} r(v_k)} \sum_{k=1}^{K} r(v_k) \frac{\partial s(v_k, h_t; \theta)}{\partial \theta}. \tag{14.79}$$

将公式（14.79）代入公式（14.63），得到每个样本目标函数关于 θ 的梯度可以近似为

$$\frac{\partial \log p_\theta(x_t|h_t)}{\partial \theta} = \frac{\partial s(x_t, h_t; \theta)}{\partial \theta} - \frac{1}{\sum_{k=1}^{K} r(v_k)} \sum_{k=1}^{K} r(v_k) \frac{\partial s(v_k, h_t; \theta)}{\partial \theta}, \tag{14.80}$$

其中 $v_1, \cdots, v_k$ 为从提议分布 $q(v|h_t)$ 中从词表 $\mathcal{V}$ 独立抽取的词。和公式（14.63）相比，重要性采样相当于采样了一个词表的子集 $\mathcal{V}' = \{v_1, \cdots, v_k\}$，然后在这个子集上求梯度 $\frac{\partial s(v_k, h; \theta)}{\partial \theta}$ 的期望；公式（14.63）中分布 $p_\theta(v|h_t)$ 被 $r(v_k)$ 所替代。这样目标函数关于 θ 的梯度就避免了在词表上对所有词进行计算，只需要计算较少的抽取的样本。采样的样本数量 K 越大，近似越接近正确值。在实际应用中，K 取 100 左右就能够以足够高的精度对期望做出估计。通过重要性采样的方法，训练速度可以加速 $\frac{|\mathcal{V}|}{K}$ 倍。

重要性采样的思想和算法都比较简单，但其效果依赖于建议分布 $q(v|h_t)$ 的选取。如果 $q(v|h_t)$ 选取不合适时，会造成梯度估计非常不稳定。在实践中，提议分布 $q(v|h_t)$ 经常使用一元模型的分布函数。虽然直观上 $q(v|h_t)$ 采用 N 元模型更加准确，但使用复杂的 N 元模型分布并不能改进性能，原因是 N 元模型的分布和神经网络模型估计的分布之间有很大的差异 [Bengio and Senécal, 2008]。

14.5.3.3 噪声对比估计

除重要性采样外，*噪声对比估计*（Noise-Contrastive Estimation，NCE）也是一种常用的近似估计梯度的方法。

*噪声对比估计*是将密度估计问题转换为两类分类问题，从而降低计算复杂度 [Gutmann and Hyvärinen, 2010]。噪声对比估计的思想在我们日常生活中十分常见。比如我们教小孩认识“苹果”，往往会让小孩从一堆各式各样的水果中找出哪个是“苹果”。通过不断的对比和纠错，最终小孩会知道了解“苹果”的特征，并很容易识别出“苹果”。

噪声对比估计的数学描述如下：假设有三个分布，一个是需要建模真实数据分布 $p_r(x)$；第二是模型分布 $p_\theta(x)$，并期望调整模型参数 θ 来使得 $p_\theta(x)$ 来拟合真实数据分布 $p_r(x)$；第三个是噪声分布 $q(x)$，用来对比学习。给定一个样本 x，如果 x 是从 $p_r(x)$ 中抽取的，称为*真实样本*，如果 x 是从 $q(x)$ 中抽取的，则称为*噪声样本*。为了判断样本 x 是真实样本还是噪声样本，引入一个*辨别函数* D。

噪声对比估计是通过调整模型 $p_\theta(x)$ 使得辨别函数 D 很容易能分别出样本 x 来自哪个分布。令 $y \in \{1, 0\}$ 表示一个样本 x 是真实样本或噪声样本，其条件概率为

$$p(x|y=1) = p_\theta(x), \tag{14.81}$$

$$p(x|y=0) = q(x). \tag{14.82}$$

一般噪声样本的数量要比真实样本大很多。为了提高近似效率，我们近似假设噪声样本的数量是真实样本的 K 倍，即 y 的先验分布满足

$$P(y=0) = KP(y=1). \tag{14.83}$$

根据贝叶斯公式，样本 x 来自于真实数据分布的后验概率为

$$p(y=1|x) = \frac{p(x|y=1)p(y=1)}{p(x|y=1)p(y=1)+p(x|y=0)p(y=0)} \tag{14.84}$$

$$= \frac{p_\theta(x)p(y=1)}{p_\theta(x)p(y=1)+q(x)\cdot kp(y=1)} \tag{14.85}$$

$$= \frac{p_\theta(x)}{p_\theta(x)+Kq(x)}. \tag{14.86}$$

相反，样本 x 来自于噪声分布的后验概率为 $p(y=0|x)=1-p(y=1|x)$。

从真实分布 $p_r(x)$ 中抽取 N 个样本 $x_1,\cdots,x_N$，将其类别设为 $y=1$，然后从噪声分布中抽取 KN 个样本 $x'_1,\cdots,x'_{kn}$，将其类别设为 $y=0$。噪声对比估计的目标是将真实样本和噪声样本区别开来，可以看作是一个两类分类问题。噪声对比估计的损失函数为

$$\mathcal{L}(\theta) = -\frac{1}{N(K+1)}\left(\sum_{n=1}^{N}\log p(y=1|x_n)+\sum_{n=1}^{KN}\log p(y=0|x'_n)\right). \tag{14.87}$$

通过不断采样真实样本和噪声样本，并用梯度下降法可以学习参数 θ 使得 $p_\theta(x)$ 逼近于真实分布 $p_r(x)$。

噪声对比估计相当于用判别式的准则 $\mathcal{L}(\theta)$ 来训练一个生成式模型 $p_\theta(x)$，使得判别函数 D 很容易能分别出样本 x 来自哪个分布，其思想与生成式对抗网络类似。不同之处在于，在噪声对比估计中的判别函数 D 是通过贝叶斯公式计算得到，而生成对抗网络的判别函数 D 是一个需要学习的神经网络。

生成对抗网络参见第13.3节。

基于噪声对比估计的序列模型 在计算序列模型的条件概率时，我们也可以利用噪声对比估计的思想来提高计算效率 [Mnih and Kavukcuoglu, 2013, Mnih and Teh, 2012]。在序列模型中需要建模的分布是 $p_\theta(v|h)$，原则上噪声分布 $q(v|h)$ 应该是依赖于历史信息 h 的条件分布，但实践中一般使用和历史信息无关的分布 $q(v)$，比如一元模型的分布。

$p_\theta(v|h)$ 参见公式 (14.48)。

给定历史信息 h，我们需要判断词表中每一个词 v 是来自于真实分布还是噪声分布。

$$P(y=1|v,h) = \frac{p_\theta(v|h)}{p_\theta(v|h)+Kq(v)}. \tag{14.88}$$

对于一个训练序列 $\mathbf{x}_{1:T}$，将 $\{(x_t,h_t)\}_{t=1}^T$ 作为真实样本，从噪声分布中抽取 K 个样本 $(x'_{t,1},\cdots,x'_{t,k})$。噪声对比估计的目标函数是

$$\mathcal{L}(\theta) = -\sum_{t=1}^{T}\left(\log P(y=1|x_t,h_t)+\sum_{k=1}^{K}\log(1-P(y=1|x'_{t,k},h_t))\right). \tag{14.89}$$

为了简洁起见，这里省略了系数 $\frac{1}{T(K+1)}$。

虽然通过噪声对比估计，将一个 $|\mathcal{V}|$ 类的分类问题转为一个两类分类问题，但是依然需要计算 $p_\theta(v|h)$，其中仍然涉及到配分函数的计算。为了避免计算配分函数，我们将负对数配分函数 $-\log Z(h,\theta)$ 作为一个可学习的参数 z_h（即每一个 h 对应一个参数），这样条件概率 $p_\theta(v|h)$ 重新定义为

$$p_\theta(v|h) = \exp\big(s(v,h;\theta)\big)\exp(z_h). \tag{14.90}$$

噪声对比估计方法的一个特点是会促使未归一化分布 $\exp\big(s(v,h;\theta)\big)$ 可以自己学习到一个近似归一化的分布，并接近真实的数据分布 $p_r(v|h)$ [Gutmann and Hyvärinen, 2010]。也就是说，学习出来的 $\exp(z_h) \approx 1$。这样可以直接令 $\exp(z_h) = 1, \forall h$，并用未归一化的分布 $\exp\big(s(v,h;\theta)\big)$ 来代替 $p_\theta(v|h)$。

Mnih and Teh [2012] 也在实验中证实，直接令 $\exp(z_h) = 1$ 不会影响模型的性能。因为神经网络有大量的参数，这些参数足以让模型学习到一个近似归一化的分布。

公式（14.88）可以写为

$$p(y=1|v,h) = \frac{\exp\big(s(v,h;\theta)\big)}{\exp\big(s(v,h;\theta)\big) + Kq(v)} \tag{14.91}$$

$$= \frac{1}{1 + \frac{Kq(v)}{\exp\big(s(v,h;\theta)\big)}} \tag{14.92}$$

$$= \frac{1}{1 + \exp\Big(-s(v,h;\theta) - \log\big(Kq(v)\big)\Big)} \tag{14.93}$$

$$= \frac{1}{1 + \exp(-(\Delta s(v,h;\theta)))} \tag{14.94}$$

$$= \sigma(\Delta s(v,h;\theta)), \tag{14.95}$$

其中 σ 为 logistic 函数，$\Delta s(v,h;\theta) = s(v,h;\theta) - \log(Kq(v))$ 为模型打分（未归一化分布）与放大的噪声分布之间的差。

在噪声对比估计中，噪声分布 $q(v)$ 的选取也十分关键。首先是从 $q(v)$ 中采样要十分容易。另外 $q(v)$ 要和真实数据分布 $p_r(v|h)$ 比较接近，否则分类问题就变得十分容易，不需要学习到一个接近真实分布的 $p_\theta(v|h)$ 就可以分出数据来源了。对自然语言的序列模型，$q(v)$ 取一元模型的分布是一个很好的选择。每次迭代噪声样本的个数 K 取值在 $25 \sim 100$ 左右。

总结 基于采样的方法并不改变模型的结构，只是近似计算参数梯度。在训练时可以显著提高模型的训练速度，但是在测试阶段依然需要计算配分函数。而基于层次化 softmax 的方法改变了模型的结构，在训练和测试时都可以加快计算速度。

14.6 序列到序列模型

在序列生成任务中，有一类任务是输入一个序列，生成另一个序列，比如机器翻译、语音识别、文本摘要、对话系统、图像标题生成等。

序列到序列（Sequence-to-Sequence，Seq2Seq）是一种条件的序列生成问题，给定一个序列$\mathbf{x}_{1:S}$，生成另一个序列$\mathbf{y}_{1:T}$。输入序列的长度S和输出序列的长度T可以不同。比如在机器翻译中，输入源语言，输入为目标语言。图14.5给出了基于循环神经网络的序列到序列机器翻译示例。

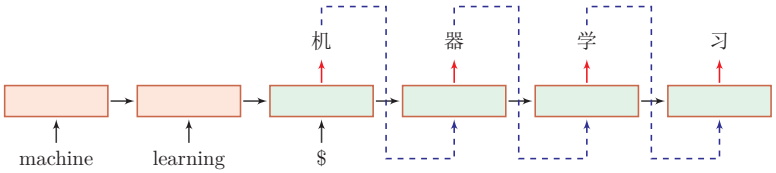


图 14.5 基于循环神经网络的序列到序列机器翻译

序列到序列模型的目标是估计条件概率

$$p_\theta(\mathbf{y}_{1:T}|\mathbf{x}_{1:S}) = \prod_{t=1}^{T} p_\theta(y_t|\mathbf{y}_{1:(t-1)}, \mathbf{x}_{1:S}), \tag{14.96}$$

其中$\mathbf{y}_t \in \mathcal{V}$为词表$\mathcal{V}$中的某个词。

给定一组训练数据$\{(\mathbf{x}_{S_n}, \mathbf{y}_{T_n})\}_{n=1}^{N}$，我们可以使用最大似然估计来训练模型参数。

$$\max_\theta \sum_{n=1}^{N} \log p_\theta(\mathbf{y}_{1:T_n}|\mathbf{x}_{1:S_n}). \tag{14.97}$$

一旦训练完成，模型就可以根据一个输入序列$\mathbf{x}$来生成最可能的目标序列，

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} p_\theta(\mathbf{y}|\mathbf{x}), \tag{14.98}$$

具体的生成过程可以通过贪婪方法或束搜索来完成。

和一般的序列生成模型类似，条件概率$p_\theta(y_t|\mathbf{y}_{1:(t-1)}, \mathbf{x}_{1:S})$可以使用各种不同的神经网络来实现。这里我们介绍三种主要的序列到序列模型。

14.6.1 基于循环神经网络的序列到序列模型

实现序列到序列的最直接方法是使用两个循环神经网络来分别进行编码和解码，也称为*编码器-解码器*（Encoder-Decoder）模型。

编码器 首先使用一个循环神经网络R_{enc}来编码输入序列$\mathbf{x}_{1:S}$得到一个固定维数的向量$\mathbf{u}$，$\mathbf{u}$一般为*编码循环神经网络*最后时刻的隐状态。

$$\mathbf{h}_t^e = f_{\text{enc}}(\mathbf{h}_{t-1}^e, \mathbf{e}_{x_{t-1}}, \theta_{\text{enc}}), \qquad \forall t \in [1:S], \tag{14.99}$$

$$\mathbf{u} = \mathbf{h}_S^e, \tag{14.100}$$

其中 $f_{\text{enc}}(\cdot)$ 为编码循环神经网络，可以为LSTM或GRU，其参数为 θ_{enc}，$\mathbf{e}_x$ 为词 x 的词向量。

解码器 在生成目标序列时，使用另外一个循环神经网络 R_{dec} 来进行解码。在解码过程的第 t 步时，已生成前缀序列为 $\mathbf{y}_{1:(t-1)}$。令 $\mathbf{h}_t$ 表示在网络 R_{dec} 的隐状态，$\mathbf{o}_t \in (0,1)^{|\mathcal{V}|}$ 为词表中所有词的后验概率，则

$$\mathbf{h}_0^d = \mathbf{u}, \tag{14.101}$$

$$\mathbf{h}_t^d = f_{\text{dec}}(\mathbf{h}_{t-1}^d, \mathbf{e}_{y_{t-1}}, \theta_{\text{dec}}), \tag{14.102}$$

$$\mathbf{o}_t = g(\mathbf{h}_t^d, \theta_o), \tag{14.103}$$

其中 $f_{\text{dec}}(\cdot)$ 为解码循环神经网络，$g(\cdot)$ 为最后一层为softmax函数的前馈神经网络，θ_{dec} 和 θ_o 为网络参数，$\mathbf{e}_y$ 为 y 的词向量，y_0 为可以为特殊的符号，比如“$”.

基于循环神经网络的序列到序列模型的缺点是：（1）向量 $\mathbf{c}$ 的容量问题，输入序列的信息很难全部保存在一个固定维度的向量中；（2）当序列很长时，由于循环神经网络的长期依赖问题，容易丢失输入序列的信息。

长期依赖问题参见第6.5节。

14.6.2 基于注意力的序列到序列模型

为了获取更丰富的输入序列信息，我们可以在每一步中通过注意力机制来从输入序列中选取有用的信息。

基于注意力的序列到序列生成过程见https://nndl.github.io/v/sgm-seq2seq

在解码过程的第 t 步时，先用上一步的隐状态 $\mathbf{h}_{t-1}^d$ 作为查询向量，利用注意力机制从所有输入序列的隐状态 $H^e = [\mathbf{h}_1^e, \cdots, \mathbf{h}_S^e]$ 中选择相关信息。

$$\mathbf{c}_t = \mathbf{att}(H^e, \mathbf{h}_{t-1}^d) = \sum_{i=1}^{S} \alpha_i \mathbf{h}_i^e \tag{14.104}$$

$$= \sum_{i=1}^{S} \text{softmax}\left(s(\mathbf{h}_i^e, \mathbf{h}_{t-1}^d)\right)\mathbf{h}_i^e \tag{14.105}$$

注意力机制参见第8.1.2节。

其中 $s(\cdot)$ 为注意力打分函数。

注意力打分函数参见第8.1.2节。

解码器第 t 步的隐状态为

$$\mathbf{h}_t^d = f_{\text{dec}}(\mathbf{h}_{t-1}^d, [\mathbf{e}_{y_{t-1}}; \mathbf{c}_t], \theta_{\text{dec}}), \tag{14.106}$$

14.6.3 基于自注意力的序列到序列模型

除长期依赖问题外，基于循环神经网络的序列到序列模型的另一个不足是无法并行计算。为了提高并行计算效率以及捕捉长距离的依赖关系，我们可以使用自注意模型来建立一个全连接的网络结构。

自注意力模型参见第8.2.2节。

本节介绍一个典型的基于自注意力的序列到序列模型：Transformer[Vaswani et al., 2017]。

14.6.3.1 自注意力

subsection 多头自注意力

对于一个向量序列 $H = [\mathbf{h}_1, \cdots, \mathbf{h}_T] \in \mathbb{R}^{d_h \times T}$，首先用自注意力模型来对其进行编码。

$$\textbf{self-att}(Q, K, V) = \text{softmax}\Big(\frac{K^{\text{T}}Q}{\sqrt{d_h}}V\Big), \tag{14.107}$$

$$Q = W_Q H, K = W_K X, V = W_V X, \tag{14.108}$$

其中 d_h 是输入向量 $\mathbf{h}_t$ 的维度，$W_Q \in \mathbb{R}^{d_k \times d_h}, W_K \in \mathbb{R}^{d_k \times d_h}, W_V \in \mathbb{R}^{d_v \times d_h}$ 为三个投影矩阵。

14.6.3.2 多头自注意力

自注意力模型可以看作是在一个线性投影空间中建立向量 H 之间交互关系。为了提取更多的交互信息，我们可以使用多头注意力，在多个不同的投影空间中捕捉不同的交互信息。

$$\textbf{MultiHead}(H) = W_O[\textbf{head}_1; \cdots; \textbf{head}_M], \tag{14.109}$$

$$\textbf{head}_m = \textbf{self-att}(Q_m, K_m, V_m), \tag{14.110}$$

$$\forall m \in [1:M], \quad Q_m = W_Q^m H, K = W_K^m X, V = W_V^m X, \tag{14.111}$$

其中 $W_O \in \mathbb{R}^{d_h \times M d_v}$ 为输出投影矩阵，$W_Q^m \in \mathbb{R}^{d_k \times d_h}, W_K^m \in \mathbb{R}^{d_k \times d_h}, W_V^m \in \mathbb{R}^{d_v \times d_h}$ 为投影矩阵，$m \in [1, M]$。

14.6.3.3 基于自注意力模型的序列编码

对于一个序列 $\mathbf{x}_{1:T}$，我们可以构建一个多层的多头自注意力来对其进行编码。由于自注意力模型忽略了输入信息的位置信息，因此初始的输入序列中加入位置编码信息来进行修正。对于一个输入序列 $\mathbf{x}_{1:T}$，

$$H^{(0)} = [\mathbf{e}_{x_1} \oplus \mathbf{p}_1, \cdots, \mathbf{e}_{x_T} \oplus \mathbf{p}_T], \tag{14.112}$$

其中 $\mathbf{e}_{x_t}$ 为词 x_t 的嵌入向量表示，$\mathbf{p}_t$ 为位置 t 的向量表示。

第 l 层的隐状态 $\mathbf{H}^{(l)}$ 为

$$Z^{(l)} = \text{norm}\Big(H^{(l-1)} + \text{MultiHead}\big(H^{(l-1)}\big)\Big), \tag{14.113}$$

$$H^{(l)} = \text{norm}\Big(\mathbf{Z}^{(l)} + \text{FFN}(Z^{(l)})\Big), \tag{14.114}$$

其中$\text{norm}(\cdot)$表示层归一化，$\text{FFN}(\cdot)$表示逐位置的前馈神经网络（Position-wise Feed-Forward Network），为一个简单的两层网络。对于输入序列中每个位置上向量$\mathbf{z}$，

层归一化参见第7.5.2节。

$$\text{FFN}(\mathbf{z}) = W_2\text{ReLu}(W_1\mathbf{z} + \mathbf{b}_1) + \mathbf{b}_2, \tag{14.115}$$

其中$W_1, W_2, \mathbf{b}_1, \mathbf{b}_2$为网络参数。

基于自注意力模型的序列编码可以看作是一个全连接的前馈神经网络，第l层的每个位置都接受第$l-1$层的所有位置的输出。不同的是，其连接权重是通过注意力机制动态计算得到。

14.6.3.4 基于自注意力模型的序列到序列模型

基于Transformer的序列到序列生成过程见https://nndl.github.io/v/sgm-seq2seq

将自注意力模型应用在序列到序列任务中，其整个网络结构可以分为两部分：

编码器 编码器只包含多层的自注意力模块，每一层都接受前一层的输出作为输入。编码器的输入为序列$\mathbf{x}_{1:S}$，输出为一个向量序列$H^e = [\mathbf{h}_1^e, \cdots, \mathbf{h}_S^e]$。

解码器 解码器依是通过自回归的方式来生成目标序列。和编码器不同，解码器可以由以下三个模块构成：

1. 自注意力模块：第t步时，先使用自注意力模型对已生成的前缀序列$\mathbf{y}_{1:(t-1)}$进行编码得到$H^d = [\mathbf{h}_1^d, \cdots, \mathbf{h}_{(t-1)}^d]$。在训练时，解码器的输入为整个目标序列，这时可以通过一个掩码（mask）来阻止每个位置选择其后面的输入信息。

2. 解码器到编码器注意力模块：使用$\mathbf{h}_{(t-1)}^d$作为查询向量，通过注意力机制来从输入序列H^e中选取有用的信息。

3. 逐位置的前馈神经网络：使用一个前馈神经网络来综合得到所有信息。

将上述三个步骤重复多次，最后通过一个全连接前馈神经网络来计算输出概率。

图14.6给出了Transformer的网络结构示例。

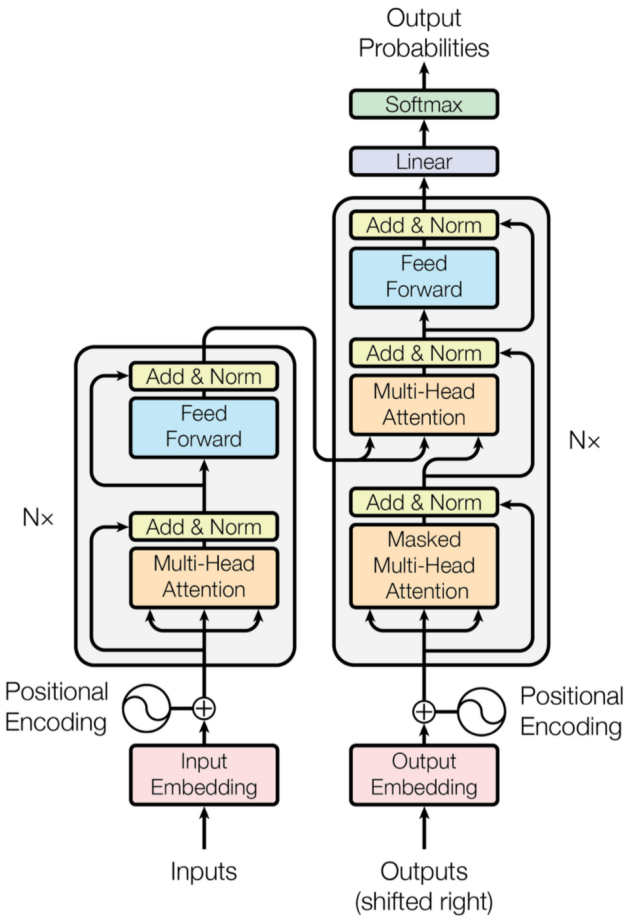


图 14.6 Transformer 网络结构

14.7 总结和深入阅读

序列生成模型主要解决序列数据的密度估计和生成问题，是一种在实际应用中十分重要的一类模型。目前主流的序列生成模型都是通过自回归模型。最早的深度序列模型是神经网络语言模型。Bengio et al. [2003] 最早提出了基于前馈神经网络的语言模型，随后Mikolov et al. [2010] 利用循环神经网络来实现语言模型。Oord et al. [2016] 针对语音合成任务提出了 WaveNet，可以生成接近自然人声的语音。

为了解决曝光偏差问题，Venkatraman et al. [2014] 提出了 DAD（Data as Demonstrator）算法，即在训练时混合使用真实数据和模型生成的数据，Bengio et al. [2015] 进一步使用课程学习（Curriculum Learning）控制使用两种数据的比例。Ranzato et al. [2015] 将序列生成看作是强化学习问题，并使用最大似然估

计来预训练模型，并逐步将训练目标由最大似然估计切换为最大期望回报。Yu et al. [2017] 进一步利用对抗生成网络的思想来进行文本生成。

由于深度序列模型在输出层使用softmax进行归一化，计算代价很高。Bengio and Senécal [2008] 提出了利用重要性采样来加速softmax的计算，Mnih and Kavukcuoglu [2013] 提出了噪声对比估计来计算非归一化的条件概率，Morin and Bengio [2005] 最早使用了层次化softmax函数来近似扁平的softmax函数。

在序列生成任务，序列到序列生成是一类十分重要的任务类型。Sutskever et al. [2014] 最早使用循环神经网络来进行机器翻译，Bahdanau et al. [2014] 使用注意力模型来改进循环神经网络的长期依赖问题，Gehring et al. [2017] 提出了基于卷积神经网络的序列到序列模型。目前最成功的序列到序列模型是全连接的自注意力模型，比如Transformer[Vaswani et al., 2017]。

Texar[1]提供了一个非常好的序列生成工具，提供了很多主流的序列生成模型。

习题

习题 14-1 证明公式(14.19)。

习题 14-2 通过文献了解N元模型中Good-Turing平滑，Kneser-Ney平滑的原理。

习题 14-3 试通过注意力机制来动态计算公式(14.24)中的权重。

习题 14-4 给定一个生成序列“The cat sat on the mat”和两个参考序列“The cat is on the mat”、“The bird sat on the bush”，分别计算BLEU-2和ROUGE-2得分。

习题 14-5 描述束搜索的实现算法。

习题 14-6 根据公式(14.89)和(14.95)，计算噪声对比估计的参数梯度，并分析其和重要性采样中参数梯度（公式（14.80））的异同点。

[1] https://github.com/asyml/texar

参考文献

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *ArXiv e-prints*, September 2014.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.

Yoshua Bengio and Jean-Sébastien Senécal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks*, 19(4):713–722, 2008.

Yoshua Bengio, Rejean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1243–1252, 2017.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS*, 2010.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems*, pages 2265–2273, 2013.

Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.

Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252, 2005.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.

Arun Venkatraman, Byron Boots, Martial Hebert, and J Andrew Bagnell. Data as demonstrator with applications to system identification. In *ALR Workshop, NIPS*, 2014.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.