

翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

Figure 17.1 State transition diagrams for some simple Markov chains. Left: a 2-state chain. Right: a 3-state left-to-right chain.

Table 17.1 Example output from a 4-gram word model, trained using backoff smoothing on the Broadcast News corpus. The first 4 words are specified by hand, the model generates the 5th word, and then the results are fed back into the model.

Figure 17.2 Unigram and bigram counts from Darwin's On The Origin Of Species. The 2D picture on the right is a Hinton diagram of the joint distribution. The size of the white squares is proportional to the value of the entry in the corresponding vector/ matrix. Based on (MacKay 2003, p22).

Figure 17.3 A Markov chain in which we put a different Dirichlet prior on every row of the transition matrix A, but the hyperparameters of the Dirichlet are shared.

17 Markov and hidden Markov models

17.1 Introduction

In this chapter, we discuss probabilistic models for sequences of observations, X_1, \dots, X_T , of arbitrary length T . Such models have applications in computational biology, natural language processing, time series forecasting, etc. We focus on the case where the observations occur at discrete "time steps", although "time" may also refer to locations within a sequence.

17.2 Markov models

Recall from Section 10.2.2 that the basic idea behind a Markov chain is to assume that X_t captures all the relevant information for predicting the future (i.e., we assume it is a sufficient statistic). If we assume discrete time steps, we can write the joint distribution as follows:

$$p(X_1:T) = p(X_1)p(X_2|X_1)p(X_3|X_2) \dots = p(X_1) \prod_{t=2}^T p(X_t|X_{t-1})$$

(17.1)

This is called a Markov chain or Markov model. If we assume the transition function $p(X_t|X_{t-1})$ is independent of time, then the chain is called homogeneous, stationary, or time-invariant. This is an example of parameter tying, since the same parameter is shared by multiple variables. This assumption allows us to model an arbitrary number of variables using a fixed number of parameters; such models are called stochastic processes.

If we assume that the observed variables are discrete, so $X_t \in \{1, \dots, K\}$, this is called a discrete-state or finite-state Markov chain. We will make this assumption throughout the rest of this section.

17.2.1 Transition matrix

When X_t is discrete, so $X_t \in \{1, \dots, K\}$, the conditional distribution $p(X_t|X_{t-1})$ can be written as a $K \times K$ matrix, known as the transition matrix A , where $A_{ij} = p(X_t = j|X_{t-1} = i)$ is the probability of going from state i to state j . Each row of the matrix sums to one, $\sum_j A_{ij} = 1$, so this is called a stochastic matrix.

A stationary, finite-state Markov chain is equivalent to a stochastic automaton. It is common to visualize such automata by drawing a directed graph, where nodes represent states and arrows represent legal transitions, i.e., non-zero elements of A . This is known as a state transition diagram. The weights associated with the arcs are the probabilities. For example, the following 2-state chain

$$A = \begin{pmatrix} 1 - \alpha & \alpha \\ \beta & 1 - \beta \end{pmatrix} \quad (17.2)$$

is illustrated in Figure 17.1(left). The following 3-state chain

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (17.3)$$

is illustrated in Figure 17.1(right). This is called a left-to-right transition matrix, and is commonly used in speech recognition (Section 17.6.2). The A_{ij} element of the transition matrix specifies the probability of getting from i to j in one step. The n -step transition matrix $A(n)$ is defined as

$$A_{ij}(n) = p(X_{t+n} = j|X_t = i) \quad (17.4)$$

which is the probability of getting from i to j in exactly n steps. Obviously $A(1) = A$. The Chapman-Kolmogorov equations state that

$$A_{ij}(m+n) = \sum_k A_{ik}(m) A_{kj}(n) \quad (17.5)$$

In words, the probability of getting from i to j in $m+n$ steps is just the probability of getting from i to k in m steps, and then from k to j in n steps, summed up over all k . We can write the above as a matrix multiplication

$$A(m+n) = A(m)A(n) \quad (17.6)$$

Hence

$$A(n) = AA(n-1) = AAA(n-2) = \dots = A^n \quad (17.7)$$

Thus we can simulate multiple steps of a Markov chain by "powering up" the transition matrix.

17.2.2 Application: Language modeling

One important application of Markov models is to make statistical language models, which are probability distributions over sequences of words. We define the state space to be all the words in English (or some other language). The marginal probabilities $p(X_t = k)$ are called unigram statistics. If we use a first-order Markov model, then $p(X_t = k | X_{t-1} = j)$ is called a bigram model. If we use a second-order Markov model, then $p(X_t = k | X_{t-1} = j, X_{t-2} = i)$ is called a trigram model. And so on. In general these are called n-gram models. For example, Figure 17.2 shows 1-gram and 2-grams counts for the letters {a,...,z, -} (where - represents space) estimated from Darwin's On The Origin Of Species. Language models can be used for several things, such as the following:

- Sentence completion A language model can predict the next word given the previous words in a sentence. This can be used to reduce the amount of typing required, which is particularly important for disabled users (see e.g., David Mackay's Dasher system¹), or uses of mobile devices.
- Data compression Any density model can be used to define an encoding scheme, by assigning short codewords to more probable strings. The more accurate the predictive model, the fewer the number of bits it requires to store the data.
- Text classification Any density model can be used as a class-conditional density and hence turned into a (generative) classifier. Note that using a 0-gram class-conditional density (i.e., only unigram statistics) would be equivalent to a naive Bayes classifier (see Section 3.5).
- Automatic essay writing One can sample from $p(x_1:t)$ to generate artificial text. This is one way of assessing the quality of the model. In Table 17.1, we give an example of text generated from a 4-gram model, trained on a corpus with 400 million words. ((Tomas et al. 2011) describes a much better language model, based on recurrent neural networks, which generates much more semantically plausible text.)

17.2.2.1 MLE for Markov language models

We now discuss a simple way to estimate the transition matrix from training data. The probability of any particular sequence of length T is given by

$$p(x_1:T|\theta) = \pi(x_1)A(x_1,x_2) \dots A(x_{T-1},x_T) \quad (17.8)$$

$$= \prod_{j=1}^K \pi(x_j) \prod_{t=2}^T \prod_{k=1}^{A_j} A_{jk} I(x_t=k, x_{t-1}=j)$$

(17.9)

Hence the log-likelihood of a set of sequences $D = (x_1, \dots, x_N)$, where $x_i = (x_{i1}, \dots, x_{iT_i})$ is a sequence of length T_i , is given by

$$\log p(D|\theta) = \sum_{i=1}^N \sum_{j=1}^{T_i} \log p(x_{ij}|\theta) = \sum_{i=1}^N \sum_{j=1}^{T_i} \log \pi_j + \sum_{i=1}^N \sum_{j=1}^{T_i} \sum_{k=1}^{N-1} \log A_{jk}$$

(17.10)

where we define the following counts:

$$N_{1j} = \sum_{i=1}^N I(x_{i1} = j), N_{jk} = \sum_{i=1}^N \sum_{t=1}^{T_i-1} I(x_{it} = j, x_{i,t+1} = k)$$

(17.11)

Hence we can write the MLE as the normalized counts:

$$\hat{\pi}_j = \frac{N_{1j}}{N}, \hat{A}_{jk} = \frac{N_{jk}}{N_{1j}}$$

(17.12)

These results can be extended in a straightforward way to higher order Markov models. However, the problem of zero-counts becomes very acute whenever the number of states K , and/or the order of the chain, n , is large. An n -gram models has $O(K^n)$ parameters. If we have $K \sim 50,000$ words in our vocabulary, then a bi-gram model will have about 2.5 billion free parameters, corresponding to all possible word pairs. It is very unlikely we will see all of these in our training data. However, we do not want to predict that a particular word string is totally impossible just because we happen not to have seen it in our training text — that would be a severe form of overfitting.² A simple solution to this is to use add-one smoothing, where we simply add one to all the empirical counts before normalizing. The Bayesian justification for this is given in Section 3.3.4.1. However add-one smoothing assumes all n -grams are equally likely, which is not very realistic. A more sophisticated Bayesian approach is discussed in Section 17.2.2.2. An alternative to using smart priors is to gather lots and lots of data. For example, Google has fit n -gram models (for $n = 1:5$) based on one trillion words extracted from the web. Their data, which is over 100GB when uncompressed, is publically available.³ An example of their data, for a set of 4-grams, is shown below.

serve as the incoming	92
serve as the incubator	99
serve as the independent	794
serve as the index	223
serve as the indication	72
serve as the indicator	120
serve as the indicators	45
serve as the indispensable	111
serve as the indispensable	40
serve as the individual	234
...	

Although such an approach, based on “brute force and ignorance”, can be successful, it is rather unsatisfying, since it is clear that this is not how humans learn (see e.g., (Tenenbaum and Xu 2000)). A more refined Bayesian approach, that needs much less data, is described in Section 17.2.2.2.

17.2.2.2 Empirical Bayes version of deleted interpolation

A common heuristic used to fix the sparse data problem is called deleted interpolation (Chen and Goodman 1996). This defines the transition matrix as a convex combination of the bigram frequencies $f_{jk} = N_{jk}/N_j$ and the unigram frequencies $f_k = N_k/N$:

$$A_{jk} = (1 - \lambda)f_{jk} + \lambda f_k \quad (17.13)$$

The term λ is usually set by cross validation. There is also a closely related technique called backoff smoothing; the idea is that if f_{jk} is too small, we “back off” to a more reliable estimate, namely f_k . We will now show that the deleted interpolation heuristic is an approximation to the predictions made by a simple hierarchical Bayesian model. Our presentation follows (McKay and Peto 1995). First, let us use an independent Dirichlet prior on each row of the transition matrix:

$$A_j \sim \text{Dir}(\alpha_0 m_1, \dots, \alpha_0 m_K) = \text{Dir}(\alpha_0 m) = \text{Dir}(\alpha) \quad (17.14)$$

where A_j is row j of the transition matrix, m is the prior mean (satisfying $\sum m_k = 1$) and α_0 is the prior strength. We will use the same prior for each row: see Figure 17.3. The posterior is given by $A_j \sim \text{Dir}(\alpha + N_j)$, where $N_j = (N_{j1}, \dots, N_{jK})$ is the vector that records the number of times we have transitioned out of state j to each of the other states. From Equation 3.51, the posterior predictive density is

$$p(X_{t+1} = k | X_t = j, D) = A_{jk} = N_{jk} + \alpha m_k N_j + \alpha_0 = f_{jk} N_j + \alpha m_k N_j + \alpha_0 = (1 - \lambda_j) f_{jk} + \lambda_j m_k \quad (17.15)$$

where $A_{jk} = E[A_{jk}|D, \alpha]$ and

$$\lambda_j = \alpha N_j + \alpha_0 \quad (17.16)$$

This is very similar to Equation 17.13 but not identical. The main difference is that the Bayesian model uses a context-dependent weight λ_j to combine m_k with the empirical frequency f_{jk} , rather than a fixed weight λ . This is like adaptive deleted interpolation. Furthermore, rather than backing off to the empirical marginal frequencies f_k , we back off to the model parameter m_k . The only remaining question is: what values should we use for α and m ? Let’s use empirical Bayes. Since we assume each row of the transition matrix is a priori independent given α , the marginal likelihood for our Markov model is found by applying Equation 5.24 to each row:

$$p(D|\alpha) = \prod_j p(X_{t+1} = k | X_t = j, D) \quad (17.17)$$

where $N_j = (N_{j1}, \dots, N_{jK})$ are the counts for leaving state j

and $B(\alpha)$ is the generalized beta function. We can fit this using the methods discussed in (Minka 2000e). However, we can also use the following approximation (McKay and Peto 1995, p12):

$$m_k \propto |\{j : N_{jk} > 0\}| \quad (17.18)$$

This says that the prior probability of word k is given by the number of different contexts in which it occurs, rather than the number of times it occurs. To justify the reasonableness of this result, Mackay and Peto (McKay and Peto 1995) give the following example.

Imagine, you see, that the language, you see, has, you see, a frequently occurring couplet 'you see', you see, in which the second word of the couplet, see, follows the first word, you, with very high probability, you see. Then the marginal statistics, you see, are going to become hugely dominated, you see, by the words you and see, with equal frequency, you see.

If we use the standard smoothing formula, Equation 17.13, then $P(\text{you}|\text{novel})$ and $P(\text{see}|\text{novel})$, for some novel context word not seen before, would turn out to be the same, since the marginal frequencies of 'you' and 'see' are the same (11 times each). However, this seems unreasonable. 'You' appears in many contexts, so $P(\text{you}|\text{novel})$ should be high, but 'see' only follows 'you', so $P(\text{see}|\text{novel})$ should be low. If we use the Bayesian formula Equation 17.15, we will get this effect for free, since we back off to m_k not f_k , and m_k will be large for 'you' and small for 'see' by Equation 17.18. Unfortunately, although elegant, this Bayesian model does not beat the state-of-the-art language model, known as interpolated Kneser-Ney (Kneser and Ney 1995; Chen and Goodman 1998). However, in (Teh 2006), it was shown how one can build a non-parametric Bayesian model which outperforms interpolated Kneser-Ney, by using variable-length contexts. In (Wood et al. 2009), this method was extended to create the "sequence memoizer", which is currently (2010) the best-performing language model.

17.2.2.3 Handling out-of-vocabulary words

While the above smoothing methods handle the case where the counts are small or even zero, none of them deal with the case where the test set may contain a completely novel word. In particular, they all assume that

the words in the vocabulary (i.e., the state space of X_t) is fixed and known (typically it is the set of unique words in the training data, or in some dictionary). Even if all A_{jk} 's are non-zero, none of these models will predict a novel word outside of this set, and hence will assign zero probability to a test sentence with an unfamiliar word. (Unfamiliar words are bound to occur, because the set of words is an open class. For example, the set of proper nouns (names of people and places) is unbounded.) A standard heuristic to solve this problem is to replace all novel words with the special symbol `unk`, which stands for "unknown". A certain amount of probability mass is held aside for this event. A more principled solution would be to use a Dirichlet process, which can generate a countably infinite state space, as the amount of data increases (see Section 25.2.2). If all novel words are "accepted" as genuine words, then the system has no predictive power, since any misspelling will be considered a new word. So the novel word has to be seen frequently enough to warrant being added to the vocabulary. See e.g., (Friedman and Singer 1999; Griffiths and Tenenbaum 2001) for details.

已思考 35s

下面先把你给的内容逐句完整翻译；紧接着用通俗的方式解释每一段里在讲什么、为什么这么做；最后给一份要点小结。为便于阅读，我保留了原文里的公式编号与图、表编号（图里的文字我也翻译了）。

逐句翻译（无遗漏）

图 17.1 一些简单马尔科夫链的状态转移图。左：一个 2 状态链。右：一个 3 状态的“从左到右”(left-to-right) 链。

表 17.1 来自一个 4-gram 词模型的示例输出，该模型在 Broadcast News (广播新闻) 语料库上用回退 (backoff) 平滑训练得到。前 4 个词是手工指定的，模型生成第 5 个词，然后把新的结果再喂回模型继续生成。

图 17.2 达尔文《物种起源》中的一元 (unigram) 与二元 (bigram) 计数。右边的二维图是联合分布的 Hinton 图。白色小方块的大小与相应向量/矩阵条目的数值成正比。基于 (MacKay 2003, 第 22 页)。

图 17.3 一个马尔科夫链：我们给转移矩阵 A 的每一行放一个不同的狄利克雷 (Dirichlet) 先验，但这些狄利克雷的超参数是共享的。

17 马尔科夫模型与隐马尔科夫模型

17.1 引言

本章讨论对任意长度 T 的观测序列 X_1, \dots, X_T 的概率模型。这类模型在计算生物学、自然语言处理、时间序列预测等领域都有应用。我们聚焦于观测出现在离散“时间步”的情形，尽管“时间”也可以指序列中的位置。

17.2 马尔科夫模型

回忆第 10.2.2 节：马尔科夫链背后的基本思想是，假定 X_t 已经包含了预测未来所需的所有相关信息（即我们假设它是一个充分统计量）。若我们假设离散时间步，那么可以把联合分布写成

$$p(X_{1:T}) = p(X_1)p(X_2 | X_1)p(X_3 | X_2) \cdots = p(X_1) \prod_{t=2}^T p(X_t | X_{t-1}) \quad (17.1)$$

这被称为马尔科夫链 (Markov chain) 或马尔科夫模型 (Markov model)。

如果我们假设转移函数 $p(X_t | X_{t-1})$ 与时间无关，则称该链是**齐次的、平稳的或时不变的**。这是**参数共享** (parameter tying) 的一个例子，因为同一个参数被多处变量使用。这个假设让我们可以用固定数量的参数来建模任意多个变量；这样的模型称为**随机过程** (stochastic process)。

如果我们再假设观测变量是离散的，即 $X_t \in \{1, \dots, K\}$ ，则称之为**离散状态或有限状态** 马尔科夫链。下面本节余下部分都做此假设。

17.2.1 转移矩阵

当 X_t 是离散的、 $X_t \in \{1, \dots, K\}$ 时，条件分布 $p(X_t | X_{t-1})$ 可以写成一个 $K \times K$ 的矩阵，称为**转移矩阵** A ，其中

$$A_{ij} = p(X_t = j | X_{t-1} = i)$$

表示从状态 i 到状态 j 的概率。矩阵的每一行都和为 1，即 $\sum_j A_{ij} = 1$ ，因此这是一个**随机矩阵** (stochastic matrix)。

一个平稳、有限状态的马尔科夫链等价于一个**随机自动机**。通常我们用有向图把这种自动机可视化：结点代表状态，箭头代表允许的转移，即 A 中非零的元素。这叫**状态转移图**。弧上的权重就是概率。举例来说，下面这个 2 状态链

$$A = \begin{bmatrix} 1 - \alpha & \alpha \\ \beta & 1 - \beta \end{bmatrix} \quad (17.2)$$

如图 17.1 (左) 所示。下面这个 3 状态链

$$A = \begin{pmatrix} A_{11} & A_{12} & 0 \\ 0 & A_{22} & A_{23} \\ 0 & 0 & 1 \end{pmatrix} \quad (17.3)$$

如图 17.1 (右) 所示。这叫**从左到右**的转移矩阵，在语音识别中很常见（见 17.6.2 节）。

转移矩阵的元素 A_{ij} 指定了一步内从 i 到 j 的概率。 n 步转移矩阵 $A^{(n)}$ 定义为

$$A_{ij}^{(n)} = p(X_{t+n} = j \mid X_t = i) \quad (17.4)$$

即恰好在 n 步内从 i 到 j 的概率。显然 $A^{(1)} = A$ 。Chapman–Kolmogorov 方程表明

$$A_{ij}^{(m+n)} = \sum_{k=1}^K A_{ik}^{(m)} A_{kj}^{(n)} \quad (17.5)$$

换句话说， $m + n$ 步从 i 到 j 的概率，就是先在 m 步从 i 到 k ，再在 n 步从 k 到 j 的概率，对所有 k 求和。把上式写成矩阵乘法就是

$$A^{(m+n)} = A^{(m)} A^{(n)} \quad (17.6)$$

因此

$$A^{(n)} = AA^{(n-1)} = AAA^{(n-2)} = \dots = A^n \quad (17.7)$$

于是我们可以通过“把转移矩阵求幂”来模拟马尔科夫链的多步演化。

17.2.2 应用：语言建模

马尔科夫模型的一个重要应用是构造**统计语言模型**——它是对词序列的概率分布。我们把状态空间定义为英语（或其他语言）里的全部词。边缘概率 $p(X_t = k)$ 称为**一元**

(unigram) 统计。若用一阶马尔科夫模型，则 $p(X_t = k \mid X_{t-1} = j)$ 称为**二元**

(bigram) 模型。若用二阶，则 $p(X_t = k \mid X_{t-1} = j, X_{t-2} = i)$ 称为**三元 (trigram) 模型**，依此类推，统称 **n -gram 模型**。例如，图 17.2 展示了从达尔文《物种起源》估计出的字母 $\{a, \dots, z, \text{空格}\}$ 的 1-gram 与 2-gram 计数。语言模型可用于多种任务，例如：

- **句子补全**：语言模型可在给定前文时预测下一个词。这可以减少输入量——对残障用户（如 David MacKay 的 Dasher 系统）或移动设备的输入尤其有用。
- **数据压缩**：任何密度模型都可定义一种编码：为更可能的字符串分配更短的码字。预测越准确，存储所需的比特就越少。
- **文本分类**：任何密度模型都能作为类别条件密度，从而变成（生成式）分类器。注意若只用 0-gram（即仅用 unigram 统计），就等价于朴素贝叶斯分类器（见 3.5 节）。
- **自动写作**：可以从 $p(x_{1:T})$ 采样以生成“人工文本”。这也是评估模型质量的一种方式。在表 17.1 中，我们给出了一个在 4 亿词语料上训练的 4-gram 模型生成的示例文本。（Tomas 等，2011，描述了基于循环神经网络的更优秀语言模型，能生成语义上更合理的文本。）

17.2.2.1 马尔科夫语言模型的极大似然估计 (MLE)

现在讨论如何从训练数据估计转移矩阵。长度为 T 的某个具体序列的概率为

$$p(x_{1:T} \mid \theta) = \pi(x_1) A(x_1, x_2) \cdots A(x_{T-1}, x_T) \quad (17.8)$$

$$= \prod_{j=1}^K (\pi_j)^{\mathbf{I}(x_1=j)} \prod_{t=2}^T \prod_{j=1}^K \prod_{k=1}^K (A_{jk})^{\mathbf{I}(x_t=k, x_{t-1}=j)} \quad (17.9)$$

因此，一组序列 $D = (x_1, \dots, x_N)$ （第 i 条序列为 $x_i = (x_{i1}, \dots, x_{iT_i})$ ，长度 T_i ）的对数似然为

$$\log p(D \mid \theta) = \sum_{i=1}^N \log p(x_i \mid \theta) = \sum_j N_j^1 \log \pi_j + \sum_{j \neq k} N_{jk} \log A_{jk} \quad (17.10)$$

其中我们定义如下计数：

$$N_j^1 = \sum_{i=1}^N \mathbf{I}(x_{i1} = j), \quad N_{jk} = \sum_{i=1}^N \sum_{t=1}^{T_i-1} \mathbf{I}(x_{i,t} = j, x_{i,t+1} = k) \quad (17.11)$$

于是 MLE 就是把计数做归一化：

$$\hat{\pi}_j = \frac{N_j^1}{\sum_j N_j^1}, \quad \hat{A}_{jk} = \frac{N_{jk}}{\sum_k N_{jk}} \quad (17.12)$$

这些结果可以直接推广到更高阶的马尔科夫模型。然而，一旦状态数 K 和/或链的阶数 n 很大，“零计数”问题会极其严重。一个 n -gram 模型有 $O(K^n)$ 个参数。若词汇表 $K \approx 50,000$ ，那么二元模型就有约 25 亿个自由参数（所有可能的词对）。训练数据几乎不可能把它们都观察到。但我们也不希望因为训练集里没出现过某个字符串，就断言它完全不可能——那会是严重的过拟合。一个简单的解决方法是**加一平滑**：把所有经验计数都加 1 再归一化。其贝叶斯理由见 3.3.4.1 节。不过，加一平滑相当于假设所有 n -gram 先验上等可能，这并不现实。更精细的贝叶斯方法见 17.2.2.2 节。另一条路是**疯狂地多收集数据**。例如，Google 用从网络上抽取的一万亿 (trillion) 词来拟合 $n = 1 \sim 5$ 的 n -gram 模型。他们的数据（未压缩超过 100GB）是公开的。下面展示了其中一组 4-gram 的例子：

```
serve as the incoming 92
serve as the incubator 99
serve as the independent 794
serve as the index 223
serve as the indication 72
serve as the indicator 120
serve as the indicators 45
serve as the indispensable 111
serve as the indispensable 40
serve as the individual 234
...
```

尽管这种基于“蛮力和无知”的方法可能成功，但它让人意难平，因为显然这不是人类习得语言的方式（见如 Tenenbaum 与 Xu, 2000）。17.2.2.2 节将介绍一种更精致且需要更少数据的贝叶斯方法。

17.2.2.2 “删一插值”的经验贝叶斯版本

解决稀疏数据问题的一个常见启发式叫**删一插值** (deleted interpolation, Chen & Goodman 1996)。它把转移矩阵定义成二元频率 $f_{jk} = N_{jk}/N_j$ 与一元频率 $f_k = N_k/N$ 的凸组合：

$$A_{jk} = (1 - \lambda)f_{jk} + \lambda f_k \quad (17.13)$$

λ 通常用交叉验证来设定。还有一个密切相关的技术叫**回退平滑** (backoff smoothing)：如果 f_{jk} 太小，就“回退”到更可靠的估计 f_k 。下面我们说明：删一插值其实是一个简单分层

贝叶斯模型预测的近似。我们的讲法跟随 McKay & Peto (1995)。

首先，对转移矩阵每一行使用独立的狄利克雷先验：

$$A_j \sim \text{Dir}(\alpha_0 m_1, \dots, \alpha_0 m_K) = \text{Dir}(\alpha_0 \mathbf{m}) = \text{Dir}(\boldsymbol{\alpha}) \quad (17.14)$$

其中 A_j 是转移矩阵的第 j 行， \mathbf{m} 是先验均值（满足 $\sum_k m_k = 1$ ）， α_0 是先验强度。我们对每一行使用相同的先验：见图 17.3。后验为

$$A_j \sim \text{Dir}(\boldsymbol{\alpha} + \mathbf{N}_j),$$

其中 $\mathbf{N}_j = (N_{j1}, \dots, N_{jK})$ 记录了从状态 j 跳到各状态的次数。由公式 (3.51)，后验预测分布为

$$p(X_{t+1} = k \mid X_t = j, D) = E[A_{jk} \mid D, \boldsymbol{\alpha}] = \frac{N_{jk} + \alpha_0 m_k}{N_j + \alpha_0} = \frac{N_j}{N_j + \alpha_0} f_{jk} + \frac{\alpha_0}{N_j + \alpha_0} \underset{(17.15)}{=} (1 - \lambda_j) f_{jk} + \lambda_j m_k$$

其中

$$\lambda_j = \frac{\alpha_0}{N_j + \alpha_0} \quad (17.16)$$

这与 (17.13) 非常相似但不完全相同。主要区别在于：贝叶斯模型用跟上下文相关的权重 λ_j 把 m_k 与经验频率 f_{jk} 进行加权，而不是用固定的 λ 。这就像自适应的删一插值。此外，我们不是回退到经验边缘频率 f_k ，而是回退到模型参数 m_k 。

剩下的问题是： $\boldsymbol{\alpha}$ 与 \mathbf{m} 取多少？我们使用经验贝叶斯。因为先验上假设转移矩阵的各行彼此独立，所以马尔科夫模型的边际似然可由公式 (5.24) 应用于每一行得到：

$$p(D \mid \boldsymbol{\alpha}) = \prod_j \frac{B(\mathbf{N}_j + \boldsymbol{\alpha})}{B(\boldsymbol{\alpha})} \quad (17.17)$$

其中 $\mathbf{N}_j = (N_{j1}, \dots, N_{jK})$ 是离开状态 j 的计数， $B(\cdot)$ 是广义 Beta 函数。可用 (Minka 2000e) 讨论的方法来拟合。不过还可以用如下近似 (McKay & Peto 1995, 第 12 页)：

$$m_k \propto |\{j : N_{jk} > 0\}| \quad (17.18)$$

它的意思是：词 k 的先验概率由它出现过的不同上下文数量决定，而不是它出现的总次数。为说明这个结果的合理性，Mackay 和 Peto 给了如下例子：

想象一下，“you see”这对词在语言中频繁出现，也就是说第二个词 *see* 以很高概率跟在第一个词 *you* 后面。那么边缘统计就会被 *you* 和 *see* 两个词大幅主导，而且两者频率相同（各 11 次）。

如果按标准的平滑公式 (17.13)，对于某个从未见过的新上下文词 *novel*， $P(\text{you} \mid \text{novel})$ 与 $P(\text{see} \mid \text{novel})$ 会相同，因为 *you* 与 *see* 的边缘频率一样（都 11 次）。但这不合理：*you* 出现在许多上下文里，所以 $P(\text{you} \mid \text{novel})$ 应该很高；而 *see* 只会跟在 *you* 后，所以 $P(\text{see} \mid \text{novel})$ 应该很低。若用贝叶斯公式 (17.15)，就能自然得到这个效果，因为我们回退到的是 m_k 而不是 f_k ，且由 (17.18) 可知 *you* 的 m_k 大、*see* 的 m_k 小。

不幸的是，尽管优雅，这个贝叶斯模型并没有超过当前最强的语言模型——**插值 Kneser–Ney** (Kneser & Ney 1995; Chen & Goodman 1998)。不过，Teh (2006) 展示了如何用可变长度上下文构建**非参数贝叶斯模型**来超越插值 Kneser–Ney。Wood 等 (2009) 又把该方法扩展成“**序列记忆器**” (sequence memoizer)，在当时 (2010 年) 是性能最佳的语言模型。

17.2.2.3 处理词表外 (OOV) 词

上面的平滑方法能处理“计数很小甚至为零”的情况，但都不能处理测试集中出现**完全新词**的情形。具体说，它们都假设词汇 (即 X_t 的状态空间) 是**固定且已知的** (通常就是训练集中出现过的所有词，或某本词典里的词)。

即使所有 A_{jk} 都非零，这些模型也不会预测在该集合之外的新词，因此只要测试句子中出现不熟悉的词，它们就会给整个句子分配零概率。(新词必然会出现，因为词的集合是**开放类**。例如专有名词——人名与地名——是无界的。) 解决该问题的一个标准启发式是：把所有新词替换为特殊符号 **unk** (unknown, 未知)，并预留一部分概率质量给这个事件。一个更有原则的解决方案是使用**狄利克雷过程** (见 25.2.2 节)，随着数据增多，它能生成**可数无限**的状态空间。如果对所有新词都“照单全收”，系统就失去预测力，因为任何拼写错误都会被当成一个新词。因此，只有当新词被足够频繁地看到时，才应把它加入词表；具体标准可见如 Friedman & Singer (1999)；Griffiths & Tenenbaum (2001)。

通俗解释（把上面内容讲“人话”）

1) 什么是马尔科夫链？

把状态想成“此刻的世界”。**马尔科夫假设**说：预测下一刻，只用“此刻”就够了，不必追溯更久的历史。于是整个长序列的概率就“串起来”变成很多“一步转移”的乘积 (公式 17.1)。

2) 转移矩阵是干嘛的？

如果状态是离散的 (比如 K 种天气：晴、雨、阴……)，那“一步转移”的概率可以装进一个 $K \times K$ 的表格：第 i 行第 j 列就是“从 i 到 j 的概率”。每一行必须加和为 1 (因为“下一步总得去某处”)。

- **从左到右链** (图 17.1 右) 是个常见约束：只能往前、不许回头 (语音里常用来描述音素按顺序推进)。

3) 多步以后会到哪？

n 步转移的概率就是把转移矩阵连乘 n 次： $A^{(n)} = A^n$ 。这就是 Chapman–Kolmogorov 的本质：走 $m + n$ 步 = 先走 m 步到某个中间站，再走 n 步到终点，对所有中间站求和。

4) 语言模型为什么能用马尔科夫？

把“状态”当成“词”。

- 只看单个词的频率叫 **unigram**。
- 看前 1 个词预测下一个叫 **bigram**；看前 2 个叫 **trigram**；一般叫 **n-gram**。
- 用处：自动补词、压缩、文本分类 (用作生成式分类器)、甚至直接“胡乱写作”。表 17.1 就是 4-gram 生成文本的例子。

5) 怎么从数据估计这些概率? (MLE)

最朴素办法：数一数。

- 初始词分布 π : 看每条句子第一个词出现了多少次，归一化。
- 转移 A_{jk} : 数“j 后面跟 k”的次数 N_{jk} , 再按行归一化 (公式 17.12)。

问题：当词很多（几十上百万），更别说 n-gram 的 K^n 个组合，**大量组合根本没在训练里出现过**，于是 MLE 会给它们概率 0 —— 这太“绝对”了。

6) 平滑：别动不动给 0。

- **加一平滑**: 所有计数加 1 再归一化。简单但粗糙（相当于认为所有 n-gram 先验上一样可能）。
- **多收集数据**: Google 的 n-gram 数据就是“堆数据”的路线，但不符合人类学习的节俭性。

7) 删一插值 & 回退 (传统工程法)

把 bigram 的估计和 unigram 混合：

$$\hat{A}_{jk} = (1 - \lambda)f_{jk} + \lambda f_k$$

如果 f_{jk} 太小，就多信一点 f_k 。 λ 用验证集调。

8) 用贝叶斯把“插值”讲圆：分层 Dirichlet

- 对每一行 A_j 放 Dirichlet 先验，先验均值是 \mathbf{m} ，强度是 α_0 。
- 得到的**后验预测**恰好是

$$(1 - \lambda_j)f_{jk} + \lambda_j m_k, \quad \lambda_j = \frac{\alpha_0}{N_j + \alpha_0}.$$

含义：数据多 (N_j 大) 时更相信数据 (λ_j 小)，数据少时更靠先验 m 。这是**自适应**的插值，比固定 λ 更合理。

- **如何设 m ?** 经验贝叶斯给了一个好用的近似：

$m_k \propto$ “词 k 出现过的不同上下文数”。

这就解释了“you see”的例子：虽然 *you* 和 *see* 的总体频率一样高，但 *you* 出现在很多上下文里、*see* 只跟在 *you* 后，因此遇到新上下文时更该偏向 *you* 而不是 *see*。

9) 更强的平滑：Kneser–Ney 与非参数贝叶斯

工程界的王者是**插值 Kneser–Ney** (减去常数再回退)，效果普遍最好。后来用**可变长度上下文的非参数贝叶斯** (如 Pitman–Yor 过程) 能进一步超越它，“序列记忆器”是其中代表。

10) 词表外 (OOV) 怎么办？

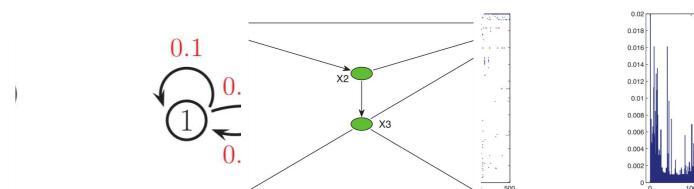
- 工程上：把没见过的词映射成统一的 *unk*，给它一定概率。
- 更“原则”的做法：用能产生**无限状态**的先验 (如狄利克雷过程/Pitman–Yor)，让模型在数据足够时自动把新词纳入词表。但要设置“阈值”(看到够多次) 避免把错别字当新词。

要点小结

- **马尔科夫假设**: 下一步只依赖当前步; 联合概率分解为一步转移的乘积 (17.1)。
- **转移矩阵**: $A_{ij} = p(X_t = j \mid X_{t-1} = i)$, 每行和为 1。多步转移用矩阵幂 A^n (17.4–17.7)。
- **语言模型**: 把“状态”当“词”，得到 unigram/bigram/trigram... (n-gram)。应用：补词、压缩、分类、生成文本。
- **MLE**: 按出现次数归一化即可 (17.12)；但高维/高阶会严重稀疏与零计数。
- **平滑**:
 - 简单: 加一平滑;
 - 传统强力工程法: 删一插值/回退、Kneser–Ney;
 - 贝叶斯视角: 对每行放 Dirichlet 先验, 后验预测 = 自适应插值

$$(1 - \lambda_j)f_{jk} + \lambda_j m_k \quad (17.15), \quad \lambda_j = \frac{\alpha_0}{N_j + \alpha_0}$$
 - 经验贝叶斯近似: m_k 与“出现过的上下文数”成正比 (17.18)，能自然区分 *you* vs *see* 的例子。
- **OOV**: 工程上用 `unk`, 更原则的办法用能扩展词表的非参数贝叶斯 (如狄利克雷/Pitman–Yor 过程)，并配合“出现频率阈值”。

如果你想，我可以把这些公式配上小例子（比如 2-3 个词的玩具语料）一步步算给你看。



翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

Figure 17.4 Some Markov chains. (a) A 3-state aperiodic chain. (b) A reducible 4-state chain.

Figure 17.5 A very small world wide web.

Figure 17.6 (a) Web graph of 500 sites rooted at www.harvard.edu. (b) Corresponding page rank vector.

17.2.3 Stationary distribution of a Markov chain

We have been focussing on Markov models as a way of defining joint probability distributions over sequences. However, we can also interpret them as stochastic dynamical systems, where we “hop” from one state to another at each time step. In this case, we are often interested in the long term distribution over states, which is known as the stationary distribution of the chain. In this

section, we discuss some of the relevant theory. Later we will consider two important applications: Google's PageRank algorithm for ranking web pages (Section 17.2.4), and the MCMC algorithm for generating samples from hard-to-normalize probability distributions (Chapter 24).

17.2.3.1 What is a stationary distribution?

Let $A_{ij} = p(X_t = j | X_{t-1} = i)$ be the one-step transition matrix, and let $\pi_t(j) = p(X_t = j)$ be the probability of being in state j at time t . It is conventional in this context to assume that π is a row vector. If we have an initial distribution over states of π_0 , then at time 1 we have

$$\pi_1(j) = \pi_0(i)A_{ij} \quad (17.19)$$

or, in matrix notation,

$$\pi_1 = \pi_0 A \quad (17.20)$$

We can imagine iterating these equations. If we ever reach a stage where

$$\pi = \pi A \quad (17.21)$$

then we say we have reached the stationary distribution (also called the invariant distribution or equilibrium distribution). Once we enter the stationary distribution, we will never leave. For example, consider the chain in Figure 17.4(a). To find its stationary distribution, we write

$$\begin{aligned} \pi_1 \pi_2 \pi_3 &= \pi_1 \pi_2 \pi_3 / (1 - A_{12} - A_{13} A_{12} A_{13} A_{21} 1 \\ &\quad - A_{21} - A_{23} A_{23} A_{31} A_{32} 1 - A_{31} - A_{32}) \end{aligned} \quad (17.22)$$

so

$$\pi_1 = \pi_1(1 - A_{12} - A_{13}) + \pi_2 A_{21} + \pi_3 A_{31} \quad (17.23)$$

or

$$\pi_1(A_{12} + A_{13}) = \pi_2 A_{21} + \pi_3 A_{31} \quad (17.24)$$

In general, we have

$$\pi_i j = i A_{ij} = j = i \pi_j A_{ji} \quad (17.25)$$

In other words, the probability of being in state i times the net flow out of state i must equal the probability of being in each other state j times the net flow from that state into i . These are called the global balance equations. We can then solve these equations, subject to the constraint that $\sum_j \pi_j = 1$.

17.2.3.2 Computing the stationary distribution

To find the stationary distribution, we can just solve the eigenvector equation $ATv = v$, and then to set $\pi = vT$, where v is an eigenvector with eigenvalue 1. (We can be sure such an eigenvector exists, since A is a row-stochastic matrix, so $A1 = 1$; also recall that the eigenvalues of A and AT are the same.) Of course, since eigenvectors are unique only up to constants of proportionality, we must normalize v at the end to ensure it sums to one. Note, however, that the eigenvectors are only guaranteed to be real-valued if the matrix is positive, $A_{ij} > 0$ (and hence $A_{ij} < 1$, due to the sum-to-one

constraint). A more general approach, which can handle chains where some transition probabilities are 0 or 1 (such as Figure 17.4(a)), is as follows (Resnick 1992, p138). We have K constraints from $\pi(I-A) = 0_{K \times 1}$ and 1 constraint from $\pi^T 1_{K \times 1} = 0$. Since we only have K unknowns, this is overconstrained. So let us replace any column (e.g., the last) of $I - A$ with 1, to get a new matrix, call it M . Next we define $r = [0, 0, \dots, 1]$, where the 1 in the last position corresponds to the column of all 1s in M . We then solve $\pi M = r$. For example, for a 3 state chain we have to solve this linear system:

$$\begin{matrix} \pi_1 & \pi_2 & \pi_3 \\ \left| \begin{array}{cccccc} 1 & -A_{11} & -A_{12} & 1 & -A_{21} & 1 & -A_{22} & 1 & -A_{31} & -A_{32} & 1 \\ \hline & & & & & 0 & 0 & 1 \end{array} \right. \end{matrix} \quad (17.26)$$

For the chain in Figure 17.4(a) we find $\pi = [0.4, 0.4, 0.2]$. We can easily verify this is correct, since $\pi = \pi A$. See mcStatDist for some Matlab code. Unfortunately, not all chains have a stationary distribution, as we explain below.

17.2.3.3 When does a stationary distribution exist?

Consider the 4-state chain in Figure 17.4(b). If we start in state 4, we will stay there forever, since 4 is an absorbing state. Thus $\pi = (0, 0, 0, 1)$ is one possible stationary distribution. However, if we start in 1 or 2, we will oscillate between those two states for ever. So $\pi = (0.5, 0.5, 0, 0)$ is another possible stationary distribution. If we start in state 3, we could end up in either of the above stationary distributions. We see from this example that a necessary condition to have a unique stationary distribution is that the state transition diagram be a singly connected component, i.e., we can get from any state to any other state. Such chains are called irreducible. Now consider the 2-state chain in Figure 17.1(a). This is irreducible provided $\alpha, \beta > 0$. Suppose $\alpha = \beta = 0.9$. It is clear by symmetry that this chain will spend 50% of its time in each state. Thus $\pi = (0.5, 0.5)$. But now suppose $\alpha = \beta = 1$. In this case, the chain will oscillate between the two states, but the long-term distribution on states depends on where you start from. If we start in state 1, then on every odd time step (1, 3, 5, ...) we will be in state 1; but if we start in state 2, then on every odd time step we will be in state 2. This example motivates the following definition. Let us say that a chain has a limiting distribution if $\pi_j = \lim_{n \rightarrow \infty} \pi_{ij}$ exists and is independent of i , for all j . If this holds, then the long-run distribution over states will be independent of the starting state:

$$P(X_t = j) = i P(X_0 = i) A_{ij}(t) \rightarrow \pi_j \text{ as } t \rightarrow \infty \quad (17.27)$$

Let us now characterize when a limiting distribution exists. Define the period of state i to be $d(i) = \gcd\{t : A_{ii}(t) > 0\}$ (17.28) where gcd stands for greatest common divisor, i.e., the

largest integer that divides all the members of the set. For example, in Figure 17.4(a), we have $d(1) = d(2) = \gcd(2, 3, 4, 6, \dots) = 1$ and $d(3) = \gcd(3, 5, 6, \dots) = 1$. We say state i is aperiodic if $d(i) = 1$. (A sufficient condition to ensure this is if state i has a self-loop, but this is not a necessary condition.) We say a chain is aperiodic if all its states are aperiodic. One can show the following important result:

Theorem 17.2.1. Every irreducible (singly connected), aperiodic finite state Markov chain has a limiting distribution, which is equal to π , its unique stationary distribution.

A special case of this result says that every regular finite state chain has a unique stationary distribution, where a regular chain is one whose transition matrix satisfies $A_{ij} > 0$ for some integer n and all i, j , i.e., it is possible to get from any state to any other state in n steps. Consequently, after n steps, the chain could be in any state, no matter where it started. One can show that sufficient conditions to ensure regularity are that the chain be irreducible (singly connected) and that every state have a self-transition. To handle the case of Markov chains whose state-space is not finite (e.g., the countable set of all integers, or all the uncountable set of all reals), we need to generalize some of the earlier definitions. Since the details are rather technical, we just briefly state the main results without proof. See e.g., (Grimmett and Stirzaker 1992) for details. For a stationary distribution to exist, we require irreducibility (singly connected) and aperiodicity, as before. But we also require that each state is recurrent. (A chain in which all states are recurrent is called a recurrent chain.) Recurrent means that you will return to that state with probability 1. As a simple example of a non-recurrent state (i.e., a transient state), consider Figure 17.4(b): state 3 is transient because one immediately leaves it and either spins around state 4 forever, or oscillates between states 1 and 2 forever. There is no way to return to state 3. It is clear that any finite-state irreducible chain is recurrent, since you can always get back to where you started from. But now consider an example with an infinite state space. Suppose we perform a random walk on the integers, $X = \{\dots, -2, -1, 0, 1, 2, \dots\}$. Let $A_{i,i+1} = p$ be the probability of moving right, and $A_{i,i-1} = 1-p$ be the probability of moving left. Suppose we start at $X_1 = 0$. If $p > 0.5$, we will shoot off to $+\infty$; we are not guaranteed to return. Similarly, if $p < 0.5$, we will shoot off to $-\infty$. So in both cases, the chain is not recurrent, even though it is irreducible.

It should be intuitively obvious that we require all states to be recurrent for a stationary distribution to exist.

However, this is not sufficient. To see this, consider the random walk on the integers again, and suppose $p = 0.5$. In this case, we can return to the origin an infinite number of times, so the chain is recurrent. However, it takes infinitely long to do so. This prohibits it from having a stationary distribution. The intuitive reason is that the distribution keeps spreading out over a larger and larger set of the integers, and never converges to a stationary distribution. More formally, we define a state to be non-null recurrent if the expected time to return to this state is finite. A chain in which all states are non-null is called a non-null chain. For brevity, we say that a state is ergodic if it is aperiodic, recurrent and non-null, and we say a chain is ergodic if all its states are ergodic.

We can now state our main theorem:

Theorem 17.2.2. Every irreducible (singly connected), ergodic Markov chain has a limiting distribution, which is equal to π , its unique stationary distribution.

This generalizes Theorem 17.2.1, since for irreducible finite-state chains, all states are recurrent and non-null.

17.2.3.4 Detailed balance

Establishing ergodicity can be difficult. We now give an alternative condition that is easier to verify. We say that a Markov chain A is time reversible if there exists a distribution π such that

$$\pi_i A_{ij} = \pi_j A_{ji} \quad (17.29)$$

These are called the detailed balance equations. This says that the flow from i to j must equal the flow from j to i , weighted by the appropriate source probabilities. We have the following important result.

Theorem 17.2.3. If a Markov chain with transition matrix A is regular and satisfies detailed balance wrt distribution π , then π is a stationary distribution of the chain.

Proof. To see this, note that

$$\sum_i \pi_i A_{ij} = \sum_i \pi_j A_{ji} = \pi_j \quad (17.30)$$

and hence $\pi = A\pi$. Note that this condition is sufficient but not necessary (see Figure 17.4(a) for an example of a chain with a stationary distribution which does not satisfy detailed balance). In Section 24.1, we will discuss Markov chain Monte Carlo or MCMC methods. These take as input a desired distribution π and construct a transition matrix (or in general, a transition kernel) A which satisfies detailed balance wrt π . Thus by sampling states from such a chain, we will eventually enter the stationary distribution, and will visit states with probabilities given by π .

17.2.4 Application: Google's PageRank algorithm for web page ranking

The results in Section 17.2.3 form the theoretical

underpinnings to Google's PageRank algorithm, which is used for information retrieval on the world-wide web. We sketch the basic idea below; see (Byran and Leise 2006) for a more detailed explanation. We will treat the web as a giant directed graph, where nodes represent web pages (documents) and edges represent hyper-links.⁵ We then perform a process called web crawling. We start at a few designated root nodes, such as dmoz.org, the home of the Open Directory Project, and then follows the links, storing all the pages that we encounter, until we run out of time. Next, all of the words in each web page are entered into a data structure called an inverted index. That is, for each word, we store a list of the documents where this word occurs. (In practice, we store a list of hash codes representing the URLs.) At test time, when a user enters a query, we can just look up all the documents containing each word, and intersect these lists (since queries are defined by a conjunction of search terms). We can get a refined search by storing the location of each word in each document. We can then test if the words in a document occur in the same order as in the query. Let us give an example, from

http://en.wikipedia.org/wiki/Inverted_index. We have 3 documents, T0 = "it is what it is", T1 = "what is it" and T2 = "it is a banana". Then we can create the following inverted index, where each pair represents a document and word location:

```
"a": {(2, 2)}  
"banana": {(2, 3)}  
"is": {(0, 1), (0, 4), (1, 1), (2, 1)}  
"it": {(0, 0), (0, 3), (1, 2), (2, 0)}  
"what": {(0, 2), (1, 0)}
```

For example, we see that the word "what" occurs at location 2 (counting from 0) in document 0, and location 0 in document 1. Suppose we search for "what is it". If we ignore word order, we retrieve the following documents: $\{T_0, T_1\} \cap \{T_0, T_1, T_2\} \cap \{T_0, T_1, T_2\} = \{T_0, T_1\}$ (17.31)

If we require that the word order matches, only document T1 would be returned. More generally, we can allow out-of-order matches, but can give "bonus points" to documents whose word order matches the query's word order, or to other features, such as if the words occur in the title of a document. We can then return the matching documents in decreasing order of their score/ relevance. This is called document ranking. So far, we have described the standard process of information retrieval. But the link structure of the web provides an additional source of information. The basic idea is that some web pages are more authoritative than others, so these should be ranked

higher (assuming they match the query). A web page is an authority if it is linked to by many other pages. But to protect against the effect of so-called link farms, which are dummy pages which just link to a given site to boost its apparent relevance, we will weight each incoming link by the source's authority. Thus we get the following recursive definition for the authoritativeness of page j , also called its PageRank:

$$\pi_j = \sum_i A_{ij} \pi_i \quad (17.32)$$

where A_{ij} is the probability of following a link from i to j . We recognize Equation 17.32 as the stationary distribution of a Markov chain. In the simplest setting, we define A_i as a uniform distribution over all states that i is connected to. However, to ensure the distribution is unique, we need to make the chain into a regular chain. This can be done by allowing each state i to jump to any other state (including itself) with some small probability. This effectively makes the transition matrix aperiodic and fully connected (although the adjacency matrix G_{ij} of the web itself is highly sparse). We discuss efficient methods for computing the leading eigenvector of this giant matrix below. But first, let us give an example of the PageRank algorithm. Consider the small web in Figure 17.5.

We find that the stationary distribution is

$$\pi = (0.3209, 0.1706, 0.1065, 0.1368, 0.0643, 0.2008) \quad (17.33)$$

So a random surfer will visit site 1 about 32% of the time. We see that node 1 has a higher PageRank than nodes 4 or 6, even though they all have the same number of in-links. This is because being linked to from an influential node helps increase your PageRank score more than being linked to by a less influential node. As a slightly larger example, Figure 17.6(a) shows a web graph, derived from the root of harvard.edu. Figure 17.6(b) shows the corresponding PageRank vector.

17.2.4.1 Efficiently computing the PageRank vector

Let $G_{ij} = 1$ iff there is a link from j to i . Now imagine performing a random walk on this graph, where at every time step, with probability $p = 0.85$ you follow one of the outlinks uniformly at random, and with probability $1 - p$ you jump to a random node, again chosen uniformly at random. If there are no outlinks, you just jump to a random page. (These random jumps, including self-transitions, ensure the chain is irreducible (singly connected) and regular. Hence we can solve for its unique stationary distribution using eigenvector methods.) This defines the following transition matrix:

$$M_{ij} = p G_{ij} / c_j + \delta \text{ if } c_j \neq 0 \quad 1/n \text{ if } c_j = 0 \quad (17.34)$$

where n is the number of nodes, $\delta = (1 - p)/n$ is the probability of jumping from one page to another without

following a link and $cj = i G_{ij}$ represents the out-degree of page j . (If $n = 4 \cdot 10^9$ and $p = 0.85$, then $\delta = 3.75 \cdot 10^{-11}$.)

Here M is a stochastic matrix in which columns sum to one. Note that $M = AT$ in our earlier notation. We can represent the transition matrix compactly as follows.

Define the diagonal matrix D with entries

$$d_{jj} = 1/c_j \text{ if } c_j > 0 \quad 0 \text{ if } c_j = 0 \quad (17.35)$$

Define the vector z with components

$$z_j = \delta \text{ if } c_j = 0 \quad 1/n \text{ if } c_j > 0 \quad (17.36)$$

Then we can rewrite Equation 17.34 as follows:

$$M = pGD + (1-p)zT \quad (17.37)$$

The matrix M is not sparse, but it is a rank one modification of a sparse matrix. Most of the elements of M are equal to the small constant δ . Obviously these do not need to be stored explicitly. Our goal is to solve $v = Mv$, where $v = \pi T$. One efficient method to find the leading eigenvector of a large matrix is known as the power method. This simply consists of repeated matrix-vector multiplication, followed by normalization:

$$v \propto Mv = pGDv + (1-p)zTv \quad (17.38)$$

It is possible to implement the power method without using any matrix multiplications, by simply sampling from the transition matrix and counting how often you visit each state. This is essentially a Monte Carlo approximation to the sum implied by $v = Mv$. Applying this to the data in Figure 17.6(a) yields the stationary distribution in Figure 17.6(b). This took 13 iterations to converge, starting from a uniform distribution. (See also the function `pagerankDemo`, by Tim Davis, for an animation of the algorithm in action, applied to the small web example.) To handle changing web structure, we can re-run this algorithm every day or every week, starting v off at the old distribution (Langville and Meyer 2006). For details on how to perform this Monte Carlo power method in a parallel distributed computing environment, see e.g., (Rajaraman and Ullman 2010).

17.2.4.2 Web spam

PageRank is not foolproof. For example, consider the strategy adopted by JC Penney, a department store in the USA. During the Christmas season of 2010, it planted many links to its home page on 1000s of irrelevant web pages, thus increasing its ranking on Google's search engine (Segal 2011). Even though each of these source pages has low PageRank, there were so many of them that their effect added up. Businesses call this search engine optimization; Google calls it web spam. When Google was notified of this scam (by the New York Times), it manually downweighted JC Penney, since such behavior violates Google's code of conduct. The result was that JC

Penney dropped from rank 1 to rank 65, essentially making it disappear from view. Automatically detecting such scams relies on various techniques which are beyond the scope of this chapter.

已思考 1m 9s

下面先把你给的内容逐句完整翻译；随后用通俗解释把关键概念掰开揉碎；最后给一份要点小结。我严格保留了原文里的公式编号与图、表编号（并在有明显排版/条件写反的地方注明并给出更正）。

逐句翻译（无遗漏）

图 17.4 一些马尔科夫链。(a) 一个 3 状态的非周期 (aperiodic) 链。(b) 一个可约 (reducible) 的 4 状态链。

图 17.5 一个非常小的万维网。

图 17.6 (a) 以 www.harvard.edu 为根的 500 个站点的网页图。(b) 相应的 PageRank 向量。

17.2.3 马尔科夫链的平稳分布

我们一直把马尔科夫模型当作定义序列联合概率分布的一种方式来讨论。然而，我们也可以把它们解释为**随机动力系统**：在每个时间步我们从一个状态“跳”到另一个状态。在这种情形下，我们通常关心**长期的状态分布**，这被称为该链的**平稳分布**。本节我们讨论一些相关理论。随后我们会看两个重要应用：Google 用来给网页排序的 PageRank 算法（17.2.4 节），以及用来从难以归一化的概率分布中产生样本的 MCMC 算法（第 24 章）。

17.2.3.1 什么是平稳分布？

令 $A_{ij} = p(X_t = j \mid X_{t-1} = i)$ 是一步转移矩阵，令 $\pi_t(j) = p(X_t = j)$ 是在时刻 t 处于状态 j 的概率。在这种上下文中，惯例是假设 π 是一行向量。如果我们有关于状态的初始分布 π_0 ，那么在时刻 1 有

$$\pi_1(j) = \sum_i \pi_0(i) A_{ij} \quad (17.19)$$

用矩阵记号可写成

$$\pi_1 = \pi_0 A \quad (17.20)$$

我们可以想象对这些方程不断迭代。如果在某一步达到了

$$\pi = \pi A \quad (17.21)$$

就称我们到达了**平稳分布**（也叫不变分布或平衡分布）。一旦进入平稳分布，我们将永远不会离开。举例来说，考虑图 17.4(a) 的链。为了求它的平稳分布，我们写成

$$\pi_1 \pi_2 \pi_3 = \pi_1 \pi_2 \pi_3 \begin{pmatrix} 1 - A_{12} - A_{13} & A_{12} & A_{13} \\ A_{21} & 1 - A_{21} - A_{23} & A_{23} \\ A_{31} & A_{32} & 1 - A_{31} - A_{32} \end{pmatrix} \quad (17.22)$$

因此

$$\pi_1 = \pi_1(1 - A_{12} - A_{13}) + \pi_2 A_{21} + \pi_3 A_{31} \quad (17.23)$$

或者

$$\pi_1(A_{12} + A_{13}) = \pi_2 A_{21} + \pi_3 A_{31} \quad (17.24)$$

一般地，有

$$\pi_i \sum_{j \neq i} A_{ij} = \sum_{j \neq i} \pi_j A_{ji} \quad (17.25)$$

换句话说，“处于状态 i 的概率 \times 从 i 流出的总流量”必须等于“处于每个其他状态 j 的概率 \times 从 j 流入 i 的流量”之和。它们被称为**全局平衡方程**。然后我们在满足 $\sum_j \pi_j = 1$ 的约束下解这些方程。

17.2.3.2 计算平稳分布

求平稳分布，我们可以直接解**特征向量**方程 $A^T v = v$ ，再令 $\pi = v^T$ ，其中 v 是特征值为 1 的特征向量。（我们确信这样一个特征向量存在，因为 A 是**行随机矩阵**，所以 $A\mathbf{1} = \mathbf{1}$ ；同时回忆 A 与 A^T 的特征值相同。）当然，由于特征向量只在比例常数上唯一，我们最后必须把 v 归一化使其和为 1。

注意：只有当矩阵是**正的** ($A_{ij} > 0$, 因此在“行和为 1”的约束下也有 $A_{ij} < 1$) 时，才能保证特征向量是实数。更一般、能处理某些转移概率为 0 或 1 的链（比如图 17.4(a)）的方法如下 (Resnick 1992, 第 138 页)。由 $\pi(I - A) = \mathbf{0}_{K \times 1}$ 得到 K 个约束，又由 $\pi \mathbf{1}_{K \times 1} = 1$ 得到 1 个约束。未知量只有 K 个，于是过约束。我们把 $I - A$ 的任意一列（例如最后一列）替换为全 1，得到一个新矩阵 M 。接着定义 $r = [0, 0, \dots, 1]$ ，最后一个 1 对应 M 中那一列全 1。求解 $\pi M = r$ 。例如，对 3 状态链，我们要解线性系统

$$\pi_1 \pi_2 \pi_3 \begin{pmatrix} 1 - A_{11} & -A_{12} & 1 \\ -A_{21} & 1 - A_{22} & 1 \\ -A_{31} & -A_{32} & 1 \end{pmatrix} = 0 \ 0 \ 1 \quad (17.26)$$

对图 17.4(a) 的链我们得到 $\pi = [0.4, 0.4, 0.2]$ 。很容易验证它正确，因为 $\pi = \pi A$ 。

见 `mcStatDist` (Matlab 代码)。遗憾的是，并非所有链都有平稳分布，下面解释原因。

17.2.3.3 什么时候存在平稳分布？

考虑图 17.4(b) 的 4 状态链。如果从状态 4 开始，就会永远停在那儿，因为 4 是**吸收态**。因此 $\pi = (0, 0, 0, 1)$ 是一个可能的平稳分布。然而，如果从 1 或 2 开始，就会在这两态之间永远来回振荡。因此 $\pi = (0.5, 0.5, 0, 0)$ 是另一个可能的平稳分布。如果从状态 3 开始，可能最终进入上面两种平稳分布之一。由此可见，要想有**唯一的**的平稳分布，一个必要条件是状态转移图**连成单个连通分量**，即从任意状态都能到达任意其他状态。这样的链称为**不可约 (irreducible)**。

再看图 17.1(a) 的 2 状态链。当 $\alpha, \beta > 0$ 时它是不可约的。若 $\alpha = \beta = 0.9$, 由对称性可知该链在每个状态上花 50% 的时间, 即 $\pi = (0.5, 0.5)$ 。但若 $\alpha = \beta = 1$, 链会在两状态之间振荡, 而长期状态分布取决于初始位置: 若从 1 开始, 则每个奇数时刻 (1,3,5,...) 在状态 1; 若从 2 开始, 则每个奇数时刻在状态 2。这个例子引出如下定义。若对所有 j , 极限

$$\pi_j = \lim_{n \rightarrow \infty} A_{ij}^{(n)}$$

存在且与 i 无关, 则称该链具有**极限分布**。若成立, 则长期的状态分布与起始状态无关:

$$P(X_t = j) = \sum_i P(X_0 = i) A_{ij}^{(t)} \rightarrow \pi_j \quad (t \rightarrow \infty) \quad (17.27)$$

现在刻画何时存在极限分布。定义状态 i 的**周期**为

$$d(i) = \gcd\{t : A_{ii}^{(t)} > 0\} \quad (17.28)$$

其中 \gcd 为最大公约数。例如在图 17.4(a) 中,

$d(1) = d(2) = \gcd(2, 3, 4, 6, \dots) = 1$, 且 $d(3) = \gcd(3, 5, 6, \dots) = 1$ 。

若 $d(i) = 1$ 就说状态 i **非周期**。 (一个充分但非必要条件是状态 i 有自环。) 若所有状态都**非周期**, 则称该链**非周期**。

可以证明如下重要结果:

定理 17.2.1 每一个**不可约且非周期**的有限状态马尔科夫链都有极限分布, 它等于 π , 也就是其**唯一的平稳分布**。

该结果的一个特例说: 每个**正则 (regular) **有限状态链都有唯一的平稳分布, 这里“正则”指其转移矩阵满足: 存在某个整数 n , 对所有 i, j 有 $A_{ij}^n > 0$, 即在 n 步内从任意状态都能到达任意其他状态。于是 n 步之后, 不论从哪里出发, 链都可能处于任意状态。可证明: 若链不可约且每个状态都有自转移, 则为正则链。

若要处理状态空间**不是有限**的情形 (例如可数的整数集合, 或不可数的实数集合), 我们需要推广先前的定义。由于细节较技术性, 这里只简述主要结论 (证明见 Grimmett & Stirzaker, 1992)。要存在平稳分布, 仍需**不可约与非周期**, 此外还要每个状态都是**常返 (recurrent) 的** (**所有状态都常返的链叫常返链**)。常返意为以概率 1 还能回到该状态。一个**非常返 (即暂态 transient)** 状态的简单例子见图 17.4(b): 状态 3 是暂态, 因为一离开它就要么在 4 附近打转, 要么在 1 和 2 之间振荡, 无法回到 3。显然任意**有限状态不可约链**都是常返的, 因为总能回到出发点。

但再看一个**无限状态空间**的例子: 在整数上做随机游走 $X = \{\dots, -2, -1, 0, 1, 2, \dots\}$ 。令 $A_{i,i+1} = p$ 为向右一步的概率, $A_{i,i-1} = 1 - p$ 为向左一步的概率, 起点 $X_1 = 0$ 。若 $p > 0.5$, 会一路走向 $+\infty$, 不保证返回; 若 $p < 0.5$, 会一路走向 $-\infty$ 。两种情形链都**不可返**, 尽管它是不可约的。

直觉上, 要有平稳分布必须所有状态都常返; 但这仍**不充分**。再看整数上的随机游走, 若 $p = 0.5$ 。这时能以无限多次返回原点, 因此是常返的; 然而**返回所需的期望时间是无穷**

大，从而没有平稳分布。直观原因是分布在越来越多的整数上越摊越开，永不收敛。形式化地，若返回该状态的期望时间有限，称该状态**非零常返** (non-null recurrent/正常返、**正遍历**)。所有状态都**非零常返**的链叫**非零链**。为简洁起见，我们把**非周期 + 常返 + 非零**的状态称为**遍历** (ergodic) 状态；若所有状态都**遍历**，就称该链**遍历** (ergodic)。

现在陈述主要定理：

定理 17.2.2 每一个**不可约且遍历**的马尔科夫链都有极限分布，它等于 π ，即其唯一的平稳分布。

这推广了定理 17.2.1，因为对不可约的**有限状态链**，所有状态都既常返又非零。

17.2.3.4 细致平衡 (detailed balance)

直接验证“遍历性”可能很难。下面给出一个更易检查的替代条件。若存在某个分布 π 使得

$$\pi_i A_{ij} = \pi_j A_{ji} \quad (17.29)$$

则称马尔科夫链 A 是**时间可逆的** (time reversible)。上式称为**细致平衡方程**：从 i 到 j 的“流量”必须等于从 j 到 i 的“流量”，并由相应的起点概率加权。我们有如下重要结果。

定理 17.2.3 若转移矩阵为 A 的马尔科夫链是正则的，并且相对于分布 π 满足细致平衡，则 π 是该链的平稳分布。

证明。 注意

$$\sum_i \pi_i A_{ij} = \sum_i \pi_j A_{ji} = \pi_j \sum_i A_{ji} = \pi_j \quad (17.30)$$

因此 $\pi^T = A^T \pi^T$ ，等价地 $\pi = \pi A$ 。注意该条件是**充分而非必要的** (见图 17.4(a)——有些有平稳分布的链并不满足细致平衡)。在 24.1 节我们将讨论马尔科夫链蒙特卡洛 (MCMC) 方法：给定想要的分布 π ，构造满足相对于 π 的细致平衡的转移矩阵 (更一般地是转移核) A 。因此，从这样的链采样最终会进入平稳分布，并按 π 的概率访问各状态。

17.2.4 应用：用于网页排序的 Google PageRank 算法

17.2.3 节的结果构成了 Google 的 PageRank 算法的理论基础，该算法用于万维网上的信息检索。下面概述基本思想；更详细的解释见 (Byran and Leise 2006)。我们把整个网络看作**巨大的有向图**：结点代表网页 (文档)，边代表超链接。⁵

然后执行称为**网络爬取** (web crawling) 的过程：从一些指定的根节点开始，如 dmoz.org (开放目录项目的主页)，沿着链接往外爬，把遇到的所有页面都存下来，直到时间耗尽。接着，把每个网页中的所有词条都录入一种叫**倒排索引** (inverted index) 的数据结构：对每个词，存储包含它的文档列表 (实际中会存 URL 的哈希码)。测试时，当用户输入查询，我们只需查出包含每个词的文档，并对这些列表求交 (因为查询由若干检索词的合取定义)。若在索引中存储每个词在文档中的位置，就能得到更细致的检索：例如检查文档中的词序是否与查询中的顺序相同。

举个来自维基百科 (http://en.wikipedia.org/wiki/Inverted_index) 的例子。我们有 3 个文档，

$T_0 = \text{"it is what it is"}$, $T_1 = \text{"what is it"}$, $T_2 = \text{"it is a banana"}$ 。那么可以建立如下倒排索引，其中每个二元组代表“文档编号、词的位置”：

```
"a":      { (2, 2) }
"banana": { (2, 3) }
"is":     { (0, 1), (0, 4), (1, 1), (2, 1) }
"it":     { (0, 0), (0, 3), (1, 2), (2, 0) }
"what":   { (0, 2), (1, 0) }
```

例如，“what”这个词出现在文档 0 的位置 2（从 0 开始计），也出现在文档 1 的位置 0。

假设我们搜索 “what is it”。若忽略词序，我们检索到的文档是

$$\{T_0, T_1\} \cap \{T_0, T_1, T_2\} \cap \{T_0, T_1, T_2\} = \{T_0, T_1\} \quad (17.31)$$

若要求词序匹配，则只有文档 T_1 会被返回。更一般地，我们可以允许词序不完全匹配，但给“词序匹配查询顺序”的文档或其他特征（比如词出现在标题里）加“加分”。随后按得分/相关性从高到低返回结果，这叫**文档排序**。

到目前为止是标准的信息检索流程。但**网络的链接结构**还能提供额外信息。基本思想是：有些网页更“权威”，应该在匹配的前提下排得更靠前。若一个网页被许多其他页面链接到，它就是“权威”。但为了防止所谓**链接农场**（为了抬高某站点相关性而建立的大量虚假页面）的影响，我们把每条指向它的链接，按**来源页面的权威性**加权。因此得到网页 j 的权威度（即 PageRank）的递归定义：

$$\pi_j = \sum_i A_{ij} \pi_i \quad (17.32)$$

其中 A_{ij} 是“从 i 追随一个链接到 j ”的概率。我们认出式 (17.32) 正是马尔科夫链的平稳分布方程。最简单的设置中，把 A_{ij} 定义为“在 i 指向的所有状态上**均匀分布**”。然而，要保证分布是唯一的，我们需要把这条链做成**正则链**：允许每个状态 i 以一个很小的概率跳到任何其他状态（包括自身）。这相当于让转移矩阵**非周期且完全连通**（尽管网络本身的邻接矩阵 G_{ij} 很稀疏）。下面将讨论计算这个巨大矩阵“主特征向量”的高效方法。先给个 PageRank 的例子：考虑图 17.5 的小网络。

我们求得平稳分布为

$$\pi = (0.3209, 0.1706, 0.1065, 0.1368, 0.0643, 0.2008) \quad (17.33)$$

因此一个“随机冲浪者”大约 32% 的时间在站点 1。我们看到结点 1 的 PageRank 高于结点 4 或 6，尽管它们的入链数相同。原因是：**来自有影响力页面的链接比来自低影响力页面的链接更能提高你的 PageRank**。再大一点的例子：图 17.6(a) 展示了以 harvard.edu 为根得到的网页图，图 17.6(b) 给出相应的 PageRank 向量。

17.2.4.1 高效计算 PageRank 向量

令 $G_{ij} = 1$ 当且仅当从 j 到 i 有链接。现在想象在该图上进行随机游走：每个时间步，以概率 $p = 0.85$ 在当前页面的出链中**均匀随机**选择一条跟随；以概率 $1 - p$ 跳到一个**均匀随机**页面；若没有出链，就直接跳到随机页面。（这些随机跳转，包括自转移，使得该链**不可约且正则**。因此我们可用特征向量方法求其唯一的平稳分布。）这定义了如下转移矩阵：

$$M_{ij} = \begin{cases} p \frac{G_{ij}}{c_j} + \delta, & c_j > 0 \\ \frac{1}{n}, & c_j = 0 \end{cases} \quad (17.34)$$

其中 n 是结点数， $\delta = (1 - p)/n$ 是“不跟随链接而在页面间跳转”的概率， $c_j = \sum_i G_{ij}$ 表示页面 j 的出度。（若 $n = 4 \times 10^9$, $p = 0.85$, 则 $\delta = 3.75 \times 10^{-11}$ 。）这里 M 是**列和为 1**的随机矩阵。注意在我们此前的记号中 $M = A^T$ 。

说明：你给的文字里条件 “ $c_j = 0$ ” 两次重复，应为上面这种“ $c_j > 0$ 与 $c_j = 0$ ”的分段——原文常见的排版小错误。

我们可更紧凑地表示转移矩阵。定义对角矩阵 D ，其条目

$$d_{jj} = \begin{cases} 1/c_j, & c_j > 0 \\ 0, & c_j = 0 \end{cases} \quad (17.35)$$

定义向量 z ，分量

$$z_j = \begin{cases} \delta, & c_j > 0 \\ 1/n, & c_j = 0 \end{cases} \quad (17.36)$$

则式 (17.34) 可等价改写为

$$M = pGD + \mathbf{1}z^T \quad (17.37)$$

矩阵 M 本身不是稀疏的，但它是对一个稀疏矩阵的**秩一修正**。 M 的大多数元素都等于很小的常数 δ ，显然无需显式存储。我们的目标是求解 $v = Mv$ ，其中 $v = \pi^T$ 。求大矩阵主特征向量的一种高效方法叫**幂法** (power method)：不断进行矩阵-向量乘并归一化：

$$v \propto Mv = pGDv + \mathbf{1}(z^T v) \quad (17.38)$$

甚至可以在不做任何矩阵乘法的情况下实现幂法：直接按**转移矩阵采样**并统计访问各状态的频率；这本质上是对 $v = Mv$ 所隐含求和的**蒙特卡洛近似**。把它应用到图 17.6(a) 的数据可得到图 17.6(b) 的平稳分布。从均匀分布出发，收敛用了 13 次迭代。（另见 Tim Davis 的 pagerankDemo 函数——一个把该算法应用到“小网络示例”的动画。）为了应对网络结构变化，我们可以每天或每周重跑该算法，并用旧分布初始化 v (Langville & Meyer 2006)。关于如何在**并行分布式**环境里执行这种“蒙特卡洛幂法”，参见如 (Rajaraman and Ullman 2010)。

17.2.4.2 网络垃圾 (Web spam)

PageRank 并非万无一失。例子：美国百货公司 JC Penney 的策略。2010 年圣诞季，它在成千上万与其无关的网页上放入指向其主页的链接，从而提升了它在 Google 搜索引擎中

的排名 (Segal 2011)。尽管这些来源页面的 PageRank 都很低，但数量太多，影响累加。商家称之为**搜索引擎优化**；Google 称之为**网络垃圾**。当《纽约时报》把此事告知 Google 后，Google 手动降低了 JC Penney 的权重，因为这种行为违反其行为准则。结果 JC Penney 从第 1 名掉到第 65 名，几乎从视野中消失。自动检测此类骗局要靠各种技术，超出本章范围。

通俗解释（把难点讲“人话”）

1) 平稳分布是啥？

把马尔科夫链想成“人在地图上随机走”。走到后来，**每个地点被访问的长期比例**会稳定到一组数，这组数就是**平稳分布** π 。一旦到了这组数，再走一步仍是这组数 ($\pi = \pi A$)。

2) 怎么求 π ？

- **特征向量法**： π 是转移矩阵的“特征值为 1 的左特征向量”。解 $A^T v = v$ ，再把 v 归一化。
- **线性方程法**：用 $\pi(I - A) = 0$ 再加上 $\sum_j \pi_j = 1$ 组成线性系统求解（小规模链好用）。

3) 什么时候唯一存在？

- **不可约**：从任何状态都能到任何状态（图 17.4(b) 不是，因为 4 吸收、1 \rightarrow 2 自成小圈、3 是过路的）。
- **非周期**：别卡在“隔一步、隔两步”这种固定节奏里；有自环通常能打破周期。
- **（在无限状态时还要）常返且返回期望时间有限**：不能越走越远回不来，也不能虽然能回但平均要等无限久。

满足这些（统称**遍历**）就有**唯一**平稳分布。

4) 细致平衡（时间可逆）

比“检查遍历性”简单：若存在 π 使 $\pi_i A_{ij} = \pi_j A_{ji}$ ，那 π 就是平稳分布。这表示“从 i 到 j 的流量”和“从 j 到 i 的流量”匹配。MCMC 就是反过来：先给想要的 π ，再造一个满足细致平衡的 A 。

5) PageRank 的核心

- **倒排索引**让我们知道“哪些文档包含这些词”，解决“是否匹配”。
- **链接图**用马尔科夫链解决“匹配里谁更重要”：一个页面被一堆**重要页面**链接到，它就重要。
- 通过在随机游走里加入**小概率的“随处传送”** (teleport)，让链不可约且非周期，于是有唯一平稳分布。
- 求解用**幂法**：反复做 $v \leftarrow Mv$ 并归一化，直到不再变化。还可以用**蒙特卡洛**方式近似实现（抽样计数）。

6) 反作弊一瞥

大量垃圾站点的“低权重但多到爆”的链接也能抬高排名——这就是 web spam。搜索引擎会用规则或算法压制这类行为。

要点小结

- **平稳分布**: $\pi = \pi A$ 。表示“漫长运行后处在各状态的比例”。
- **计算**: 解 $A^T v = v$ (主特征向量)，或解线性系统；最后归一化。
- **存在与唯一性** (有限状态) : 不可约 + 非周期 \Rightarrow 唯一平稳分布；更一般还需常返且返回期望有限 (遍历)。
- **细致平衡**: 若存在 π 使 $\pi_i A_{ij} = \pi_j A_{ji}$, 则 π 为平稳分布 (充分条件)。MCMC 利用它来“按 π ”采样。
- **PageRank**: 把网络当作随机游走，加入小概率“随处跳转”使其正则；平稳分布就是网页权威度。用幂法或蒙特卡洛幂法求解。
- **工程注意**: 链接农场会操纵排名，需反作弊手段。
- **小纠错**: 式 (17.34)、(17.35)、(17.36) 中分段条件应为“ $c_j > 0$ 与 $c_j = 0$ ”而非重复“ $c_j = 0$ ”。

如果你愿意，我可以用一个 4–5 个页面的小图，手把手做一次 M 的构造与幂法迭代过程 (包含每一步的数值)。

17.2节主要在讲什么？我不太懂

已思考 23s

没问题！把 17.2 节当成“**马尔科夫链入门 + 两个重要应用**”。它主要讲这些：

一句话版

马尔科夫链 = 只看“此刻”就能预测“下一刻”的随机过程。

用它能：①给序列建模（比如语言里的词序列），②分析“长期会停在哪些状态”（平稳分布），③据此做 PageRank 等应用。

核心概念（用最少公式讲清）

1. 马尔科夫假设

序列 X_1, \dots, X_T 满足

$$p(X_{1:T}) = p(X_1) \prod_{t=2}^T p(X_t | X_{t-1})$$

意思是：下一步只依赖上一步。

2. 转移矩阵 A

离散状态时，把“一步从 i 去 j 的概率”装进矩阵 $A_{ij} = p(X_t = j | X_{t-1} = i)$ 。每行加和

为 1。

多走 n 步，就等价于把矩阵“求 n 次幂” A^n 。

3. 语言建模 (n -gram)

把“状态”当“词”。

- unigram：单词频率；bigram：看前 1 个词；trigram：看前 2 个词……
- **估计方法 (MLE)**：数次数再按行归一化。
- **平滑**：避免“没见过就给 0 概率”。有简单的“加一”，工程好用的“删一插值/回退”，以及贝叶斯版（对每行放 Dirichlet 先验，得到自适应权重 $(1 - \lambda_j)f_{jk} + \lambda_j m_k$ ）。
- **OOV (词表外)**：工程上用 “unk” 占位；更原则可用能生成无限状态的先验（如狄利克雷/ Pitman–Yor 过程）。

4. 平稳分布 π (长期行为)

若存在 π 使 $\pi = \pi A$ ，就是长期各状态出现的比例。

唯一存在的典型条件 (有限状态)：**不可约** (任意可达) + **非周期** (不陷入固定节奏)。更一般还要“常返且返回期望有限” (合称**遍历**)。

检查更容易的**细致平衡**：若存在 π 使 $\pi_i A_{ij} = \pi_j A_{ji}$ ，则 π 是平稳分布 (充分条件)。

5. 应用：PageRank

把整个网页当图；“随机冲浪者”以 0.85 的概率顺着出链走、以 0.15 的概率随机传送 (确保不可约/非周期)。

平稳分布 π 就是每页的“权威度”。求解用**幂法**：反复 $v \leftarrow Mv$ 再归一化，直到收敛。

直观小比喻

- **天气**：今天晴→明天晴/雨的概率只看“今天”。
- **打字预测**：你刚输 “New York”，下一个词“city”的概率很大，这是 bigram。
- **网页冲浪**：一直点链接（偶尔随便跳一页），长期在哪些站上停得多，就是 PageRank 高的站。

记住这 4 个式子就行

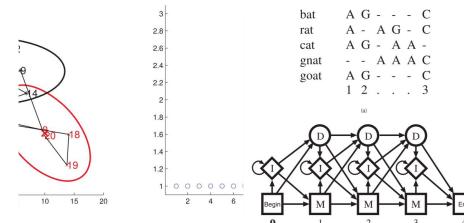
- 分解： $p(X_{1:T}) = p(X_1) \prod_{t=2}^T p(X_t | X_{t-1})$
- 转移： $A_{ij} = p(X_t = j | X_{t-1} = i)$ ，多步： $A^{(n)} = A^n$
- MLE： $\hat{A}_{jk} = N_{jk} / \sum_k N_{jk}$ (数数后归一化)
- 平稳： $\pi = \pi A$ (或解 $A^T v = v$ 并归一化)

小结

- 17.2 节用马尔科夫链来建模序列，告诉你怎么数数→得到转移概率→做平滑，再讲长期分布 (π) 以及它存在/唯一的条件，最后用 PageRank 展示“链接=转移，权威=平稳分布”的实战。

- 如果你能把“只看上一步”“转移矩阵”“平稳分布”“幂法”四件事串起来，就抓住了本节的主线。

想要的话，我可以用一个 2-3 个状态的玩具例子，带你把 \hat{A} 、 A^2 、 π 都算一遍。



翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

Figure 17.7 (a) Some 2d data sampled from a 3 state HMM. Each state emits from a 2d Gaussian. (b) The hidden state sequence.

Figure 17.8 (a) Some DNA sequences. (b) State transition diagram for a profile HMM.

17.3 Hidden Markov models

As we mentioned in Section 10.2.2, a hidden Markov model or HMM consists of a discrete-time, discrete-state Markov chain, with hidden states $z_t \in \{1, \dots, K\}$, plus an observation model $p(x_t|z_t)$. The corresponding joint distribution has the form

$$p(z_1:T, x_1:T) = p(z_1:T)p(x_1:T|z_1:T) = p(z_1) \prod_{t=2}^T p(z_t|z_{t-1}) \prod_{t=1}^T p(x_t|z_t) \quad (17.39)$$

The observations in an HMM can be discrete or continuous. If they are discrete, it is common for the observation model to be an observation matrix:

$$p(x_t = l|z_t = k, \theta) = B(k, l) \quad (17.40)$$

If the observations are continuous, it is common for the observation model to be a conditional Gaussian:

$$p(x_t|z_t = k, \theta) = N(x_t|\mu_k, \Sigma_k) \quad (17.41)$$

Figure 17.7 shows an example where we have 3 states, each of which emits a different Gaussian. The resulting model is similar to a Gaussian mixture model, except the cluster membership has Markovian dynamics. (Indeed, HMMs are sometimes called Markov switching models (Frühwirth-Schnatter 2007).) We see that we tend to get multiple observations in the same location, and then a sudden jump to a new cluster.

17.3.1 Applications of HMMs

HMMs can be used as black-box density models on sequences. They have the advantage over Markov models

in that they can represent long-range dependencies between observations, mediated via the latent variables.

In particular, note that they do not assume the Markov property holds for the observations themselves. Such black-box models are useful for timeseries prediction (Fraser 2008). They can also be used to define class-conditional densities inside a generative classifier.

However, it is more common to imbue the hidden states with some desired meaning, and to then try to estimate the hidden states from the observations, i.e., to compute $p(z_t|x_1:t)$ if we are in an online scenario, or $p(z_t|x_1:T)$ if we are in an offline scenario (see Section 17.4.1 for further discussion of the differences between these two approaches). Below we give some examples of applications which use HMMs in this way:

- Automatic speech recognition. Here x_t represents features extracted from the speech signal, and z_t represents the word that is being spoken. The transition model $p(z_t|z_{t-1})$ represents the language model, and the observation model $p(x_t|z_t)$ represents the acoustic model. See e.g., (Jelinek 1997; Jurafsky and Martin 2008) for details.
- Activity recognition. Here x_t represents features extracted from a video frame, and z_t is the class of activity the person is engaged in (e.g., running, walking, sitting, etc.) See e.g., (Szeliski 2010) for details.
- Part of speech tagging. Here x_t represents a word, and z_t represents its part of speech (noun, verb, adjective, etc.) See Section 19.6.2.1 for more information on POS tagging and related tasks.
- Gene finding. Here x_t represents the DNA nucleotides (A,C,G,T), and z_t represents whether we are inside a gene-coding region or not. See e.g., (Schweikert et al. 2009) for details.
- Protein sequence alignment. Here x_t represents an amino acid, and z_t represents whether this matches the latent consensus sequence at this location. This model is called a profile HMM and is illustrated in Figure 17.8. The HMM has 3 states, called match, insert and delete. If z_t is a match state, then x_t is equal to the t 'th value of the consensus. If z_t is an insert state, then x_t is generated from a uniform distribution that is unrelated to the consensus sequence. If z_t is a delete state, then $x_t = -$. In this way, we can generate noisy copies of the consensus sequence of different lengths. In Figure 17.8(a), the consensus is "AGC", and we see various versions of this below. A path through the state transition diagram, shown in Figure 17.8(b), specifies how to align a sequence to the consensus, e.g., for the gnat, the most probable path is

D,D, I, I, I,M. This means we delete the A and G parts of the consensus sequence, we insert 3 A's, and then we match the final C. We can estimate the model parameters by counting the number of such transitions, and the number of emissions from each kind of state, as shown in Figure 17.8(c). See Section 17.5 for more information on training an HMM, and (Durbin et al. 1998) for details on profile HMMs.

Note that for some of these tasks, conditional random fields, which are essentially discriminative versions of HMMs, may be more suitable; see Chapter 19 for details.

下面先把你给的内容逐句完整翻译；然后做通俗解释（尽量少数学、更多直觉）；最后给要点小结。我保留了公式编号与图编号。

逐句翻译（无遗漏）

图 17.7 (a) 从一个 3 个状态的 HMM 里采样得到的一些二维数据。每个状态都从一个二维高斯分布发射（生成）观测。(b) 隐藏状态序列。

图 17.8 (a) 一些 DNA 序列。(b) 一个“轮廓 HMM (profile HMM)”的状态转移图。

17.3 隐马尔科夫模型 (HMM)

正如我们在第 10.2.2 节中提到的，**隐马尔科夫模型 (HMM)** 由离散时间、离散状态的马尔科夫链组成，其隐藏状态为 $z_t \in \{1, \dots, K\}$ ，再加上一个观测模型 $p(x_t | z_t)$ 。相应的联合分布形式为

$$p(z_{1:T}, x_{1:T}) = p(z_{1:T}) p(x_{1:T} | z_{1:T}) = p(z_1) \prod_{t=2}^T p(z_t | z_{t-1}) \prod_{t=1}^T p(x_t | z_t) \quad (17.39)$$

HMM 中的观测可以是离散的，也可以是连续的。如果是**离散观测**，常把观测模型写成一个**观测矩阵**：

$$p(x_t = l | z_t = k, \theta) = B(k, l) \quad (17.40)$$

如果是**连续观测**，常把观测模型写成一个**条件高斯**：

$$p(x_t | z_t = k, \theta) = N(x_t | \mu_k, \Sigma_k) \quad (17.41)$$

图 17.7 展示了一个例子：我们有 3 个状态，每个状态发射一个不同的高斯。得到的模型与 **高斯混合模型 (GMM) **相似，只是“簇成员身份”具有马尔科夫的动态（确实，HMM 有时也叫 **Markov switching models** (Fruhwirth-Schnatter 2007)）。我们看到：通常会在同一位置得到一串连续观测，然后突然跳到一个新的簇。

17.3.1 HMM 的应用

HMM 可以作为**黑盒的序列密度模型**使用。它相对于普通马尔科夫模型的优势在于：它可以在**隐藏变量的“中介”下表示观测之间的远距离依赖**。特别地，注意它并不假设观测自身满足马尔科夫性。这样的黑盒模型对**时间序列预测**很有用 (Fraser 2008)。它们也可以用来在生成式分类器中**定义类别条件密度**。

不过，更常见的做法是给隐藏状态赋予一些我们想要的含义，然后尝试从观测中估计隐藏状态，也就是在**在线**情景下计算 $p(z_t | x_{1:t})$ ，或在**离线**情景下计算 $p(z_t | x_{1:T})$ (关于这两种方法的差异详见 17.4.1 节)。下面给出一些以这种方式使用 HMM 的应用示例：

- **自动语音识别**。这里 x_t 表示从语音信号中提取的特征， z_t 表示当前正在发出的单词。转移模型 $p(z_t | z_{t-1})$ 表示**语言模型**，观测模型 $p(x_t | z_t)$ 表示**声学模型**。详见如 (Jelinek 1997; Jurafsky and Martin 2008)。
- **动作识别**。这里 x_t 表示从视频帧中提取的特征， z_t 是人所从事的活动类别 (例如跑、走、坐等)。详见如 (Szeliski 2010)。
- **词性标注**。这里 x_t 表示一个词， z_t 表示其词性 (名词、动词、形容词等)。关于词性标注及相关任务，见 19.6.2.1 节。
- **基因查找**。这里 x_t 表示 DNA 碱基 (A、C、G、T)， z_t 表示当前位置是否处在**基因编码区**之内。详见如 (Schweikerta 等, 2009)。
- **蛋白质序列比对**。这里 x_t 表示一个氨基酸， z_t 表示它是否与该位置上的**隐含共识序列 (consensus)** 匹配。该模型称为**轮廓 HMM (profile HMM)**，见图 17.8。HMM 有 3 类状态：**匹配 (M, match)**、**插入 (I, insert)** 和**删除 (D, delete)**。

若 z_t 是匹配状态，则 x_t 等于共识序列的第 t 个取值；

若 z_t 是插入状态，则 x_t 由一个与共识无关的**均匀分布**生成；

若 z_t 是删除状态，则 $x_t = -$ (空缺)。

这样，我们可以生成长度不同且带噪声的共识序列副本。在图 17.8(a) 中，共识为 “AGC”，下面能看到它的各种版本。穿过图 17.8(b) 所示状态转移图的一条路径，给出了如何把一个序列与共识对齐；例如对 “gnat”，最可能的路径是 D, D, I, I, I, M。这意味着：删除共识中的 A 和 G 两部分，插入 3 个 A，然后匹配最后的 C。我们可以通过**计数**这类转移的次数，以及从每种状态发射的次数，来估计模型参数，如图 17.8(c) 所示。关于如何训练 HMM，见 17.5 节；关于 profile HMM 的细节，见 (Durbin 等, 1998)。

请注意：对于上述某些任务，**条件随机场 (CRF)** ——本质上是 HMM 的判别式版本——可能更合适；详见第 19 章。

通俗解释 (尽量“人话”)

1) HMM 是什么？

它是一条“看不见的马尔科夫链”+“看得见的观测”。

- **隐藏层**：状态 z_t 按马尔科夫规则自己变化 (只依赖上一步 z_{t-1})。
- **观测层**：每一步先看 z_t 是什么，再按 $p(x_t | z_t)$ 生成一个观测 x_t 。

- **联合概率**: 先把整条隐藏链的概率乘起来，再把每步从状态到观测的概率也乘起来
(式 17.39)。

2) 观测可以长啥样?

- **离散型**: 比如语音里的“音素类别”“词表索引”，用一个“观察矩阵” $B(k, l)$ 给出“在状态 k 时看到第 l 个符号”的概率 (式 17.40)。
- **连续型**: 比如二维坐标、声学特征，用“在状态 k 时是某个高斯 $N(\mu_k, \Sigma_k)$ ”(式 17.41) 来描述。

3) 它和 GMM 有何不同?

GMM 每个点独立地选一个簇；HMM 则让“簇标签”随时间按马尔科夫链慢慢变化，所以你常常会看到连着好几步来自同一个簇，中间偶尔跳到另一个簇 (图 17.7 的直观)。

4) 为什么 HMM 有用?

- 它让“远距离信息”通过隐藏状态传递 (因为状态可以“记住”一段时间)，不强迫观测本身满足“只看上一个观测就够”的马尔科夫性。
- 你既可以把它当**黑盒密度模型** (只用来拟合/预测时间序列)，也可以把状态赋义，做**状态估计**：
 - 在线: 滤波 $p(z_t | x_{1:t})$;
 - 离线: 平滑 $p(z_t | x_{1:T})$ 。

5) 应用怎么理解?

- **语音识别**: 状态=词/音素 (语言模型)，观测=声学特征 (声学模型)。
- **动作识别**: 状态=动作类别，观测=视频特征。
- **词性标注**: 状态=词性，观测=词。
- **基因查找**: 状态=“在基因里/不在”，观测=碱基。
- **profile HMM (蛋白比对)**:
 - 设有一个“共识序列” (像模板)。
 - 三类状态: 匹配 M、插入 I、删除 D。
 - **路径**告诉你如何把输入序列对齐到模板：遇到 D=“跳过模板这一位”、I=“在这里额外插一点东西”、M=“对上模板的这位”。
 - 例子 D, D, I, I, I, M: 跳过模板里头两位 (A, G)，加 3 个 A，然后对上最后的 C。
 - **训练**: 数一数从 M 到 I、I 到 I、... 的转移次数，以及每种状态发射出什么符号的次数，做归一化就是参数。

6) 何时用 CRF?

当你更关心**判别** (直接学 $p(z | x)$) 而非**生成** (学 $p(x, z)$) 时，CRF 往往效果更好，比如序列标注任务 (详见第 19 章)。

要点小结

- **HMM 的结构**: 隐藏马尔科夫链 $z_{1:T}$ + 条件观测 $x_t | z_t$ 。联合分解如式 (17.39)。

- **观测模型**: 离散 \rightarrow 观测矩阵 $B(k, l)$ (17.40) ; 连续 \rightarrow 条件高斯 $N(\mu_k, \Sigma_k)$ (17.41) 。
- **与 GMM 的差别**: GMM 的簇独立; HMM 的“簇标签”随时间按马尔科夫规律演化，能产生段停留与偶尔跳转。
- **用途两类**:
 1. 黑盒建模/预测时间序列;
 2. 赋义后做状态估计 (在线滤波 $p(z_t | x_{1:t})$ 或离线平滑 $p(z_t | x_{1:T})$) 。
- **典型应用**: 语音识别、动作识别、词性标注、基因查找、蛋白序列比对 (profile HMM 的 M/I/D 三状态与对齐思想) 。
- **训练直觉**: 靠“**计数→归一化**”学转移概率与发射概率 (第 17.5 节会展开) 。
- **替代方案**: 若重视判别性能, 考虑 CRF (第 19 章) 。

如果你愿意, 我可以用一个极小的玩具 HMM (比如 2 个状态、离散观测), 带你把“参数估计、前向后向求 $p(z_t | x)$ 、Viterbi 找最可能路径”全部算一遍。

这一节主要在讲什么? 我不太懂

已思考 14s

下面把**17.3 节 (隐马尔科夫模型, HMM) **用最直白的方式总结给你:

一句话

HMM = 一条“看不见的马尔科夫链” (隐藏状态 z_t) + “能看到的观测 x_t ”; 隐藏状态按马尔科夫规则自己演化，并在每一步“发射”出一个观测。

核心思想 (不绕弯)

- **两层结构**
 - **隐藏层**: $z_t \in \{1, \dots, K\}$; 只依赖上一步 z_{t-1} (马尔科夫) 。
 - **观测层**: 看当前状态 z_t 决定 x_t 的分布。
- **联合分解** (知道就行, 不必推) : 先把整条隐藏链的概率乘起来, 再把每步的“发射”概率乘起来。
- **观测模型两种常见形式**
 - **离散型**: 观测矩阵 $B(k, l)$ 给出“在状态 k 看到符号 l ”的概率。
 - **连续型**: 条件高斯 $N(\mu_k, \Sigma_k)$ (每个状态一朵高斯) 。
- **跟普通马尔科夫/高斯混合的区别**
 - **比马尔科夫链**: 状态不可见, 多了“发射”这层; 能通过隐藏状态捕捉**长程依赖**, 不要求观测本身满足马尔科夫。
 - **比GMM**: GMM 的“簇”彼此独立抽; HMM 的“簇标签”随时间有**连续性** (常在同一簇停一段再跳) 。

你通常会做的事

- **状态估计**

- **在线（滤波）**：算 $p(z_t | x_{1:t})$ ，边看边猜现在处于哪个状态。
- **离线（平滑）**：算 $p(z_t | x_{1:T})$ ，看完整段后回头更准确地猜每个时刻的状态。
- **参数学习**（本节只点到为止，细节在 17.5）：数转移/发射的频次或用 EM 之类方法估。

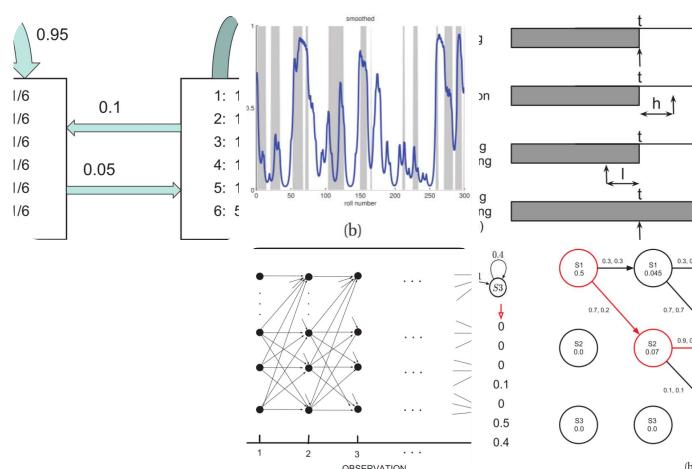
这节给的应用直觉

- **语音识别**：状态=词/音素（语言模型），观测=声学特征（声学模型）。
- **动作识别**：状态=动作类别，观测=视频特征。
- **词性标注**：状态=词性，观测=词。
- **基因查找**：状态=是否在基因编码区，观测=碱基 A/C/G/T。
- **Profile HMM (蛋白/基因序列对齐)**：三类状态
 - M (match)：对上模板这一位；
 - I (insert)：这里额外插入一些符号；
 - D (delete)：跳过模板这一位。
 路径（如 D,D,I,I,M）就是“如何把序列对齐到模板”的方案；通过统计各类转移/发射来估参数。

记住这几句就够了

- HMM = 隐藏马尔科夫 + 发射观测。
- 观测可离散（矩阵 B）或连续（每状态一朵高斯）。
- 它能表达**观测之间的远程依赖**，比 n-gram/纯马尔科夫更灵活。
- 典型任务是：**推断隐藏状态与训练转移/发射概率**；序列对齐可用 Profile HMM 的 M/I/D 机制。

如果你想，我可以用一个最小 HMM（2 个状态、几条观测）现场算一遍“滤波、平滑、Viterbi 最可能路径”，保证更直观。



翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

Figure 17.9 An HMM for the occasionally dishonest casino. The blue arrows visualize the state transition diagram A.

Figure 17.10 Inference in the dishonest casino. Vertical gray bars denote the samples that we generated using a loaded die. (a) Filtered estimate of probability of using a loaded dice. (b) Smoothed estimates. (c) MAP trajectory.
 Figure 17.11 The main kinds of inference for state-space models. The shaded region is the interval for which we have data. The arrow represents the time step at which we want to perform inference. t is the current time, T is the sequence length, l is the lag and h is the prediction horizon. See text for details.

Figure 17.12 The trellis of states vs time for a Markov chain.

Figure 17.13 Illustration of Viterbi decoding in a simple HMM for speech recognition. (a) A 3-state HMM for a single phone. We are visualizing the state transition diagram. We assume the observations have been vector quantized into 7 possible symbols, C_1, \dots, C_7 . Each state z_1, z_2, z_3 has a different distribution over these symbols. Based on Figure 15.20 of (Russell and Norvig 2002). (b) Illustration of the Viterbi algorithm applied to this model, with data sequence C_1, C_3, C_4, C_6 . The columns represent time, and the rows represent states. An arrow from state i at $t - 1$ to state j at t is annotated with two numbers: the first is the probability of the $i \rightarrow j$ transition, and the second is the probability of generating observation x_t from state j . The bold lines/ circles represent the most probable sequence of states.

17.4 Inference in HMMs

We now discuss how to infer the hidden state sequence of an HMM, assuming the parameters are known. Exactly the same algorithms apply to other chain-structured graphical models, such as chain CRFs (see Section 19.6.1). In Chapter 20, we generalize these methods to arbitrary graphs. And in Section 17.5.2, we show how we can use the output of inference in the context of parameter estimation.

17.4.1 Types of inference problems for temporal models

There are several different kinds of inferential tasks for an HMM (and SSM in general). To illustrate the differences, we will consider an example called the occasionally dishonest casino, from (Durbin et al. 1998). In this model, $x_t \in \{1, 2, \dots, 6\}$ represents which dice face shows up, and z_t represents the identity of the

dice that is being used. Most of the time the casino uses a fair dice, $z = 1$, but occasionally it switches to a loaded dice, $z = 2$, for a short period. If $z = 1$ the observation distribution is a uniform multinoulli over the symbols {1,...,6}. If $z = 2$, the observation distribution is skewed towards face 6 (see Figure 17.9). If we sample from this model, we may observe data such as the following:

Listing 17.1 Example output of casinoDemo Rolls:
 66415321616211523465321435663426165523423231514
 2464156663246 Die:
 LLLLLLLLLLFFFFFLLLLLLFLLLLLLFFFLFFFFFFFL
 LLLLLL

Here “rolls” refers to the observed symbol and “die” refers to the hidden state (L is loaded and F is fair). Thus we see that the model generates a sequence of symbols, but the statistics of the distribution changes abruptly every now and then. In a typical application, we just see the rolls and want to infer which dice is being used. But there are different kinds of inference, which we summarize below.

- Filtering means to compute the belief state $p(z_t|x_1:t)$ online, or recursively, as the data streams in. This is called “filtering” because it reduces the noise more than simply estimating the hidden state using just the current estimate, $p(z_t|x_t)$. We will see below that we can perform filtering by simply applying Bayes rule in a sequential fashion. See Figure 17.10(a) for an example.
- Smoothing means to compute $p(z_t|x_1:T)$ offline, given all the evidence. See Figure 17.10(b) for an example. By conditioning on past and future data, our uncertainty will be significantly reduced. To understand this intuitively, consider a detective trying to figure out who committed a crime. As he moves through the crime scene, his uncertainty is high until he finds the key clue; then he has an “aha” moment, his uncertainty is reduced, and all the previously confusing observations are, in hindsight, easy to explain.
- Fixed lag smoothing is an interesting compromise between online and offline estimation; it involves computing $p(z_{t-l+1:T}|x_1:t)$, where $l > 0$ is called the lag. This gives better performance than filtering, but incurs a slight delay. By changing the size of the lag, one can trade off accuracy vs delay.
- Prediction Instead of predicting the past given the future, as in fixed lag smoothing, we might want to predict the future given the past, i.e., to compute $p(z_{t+h}|x_1:t)$, where $h > 0$ is called the prediction horizon. For example, suppose $h = 2$; then we have

$$p(z_{t+2}|x_1:t) = z_{t+1} \ p(z_{t+2}|z_{t+1})p(z_{t+1}|z_t)p(z_t|x_1:t)$$

(17.42)

It is straightforward to perform this computation: we just power up the transition matrix and apply it to the current belief state. The quantity $p(z_{t+h}|x_1:t)$ is a prediction about future hidden states; it can be converted into a prediction about future observations using

$$p(x_{t+h}|x_1:t) = z_{t+h} \ p(x_{t+h}|z_{t+h})p(z_{t+h}|x_1:t) \quad (17.43)$$

This is the posterior predictive density, and can be used for time-series forecasting (see (Fraser 2008) for details). See Figure 17.11 for a sketch of the relationship between filtering, smoothing, and prediction.

- MAP estimation This means computing $\arg \max_{z_1:T} p(z_1:T|x_1:T)$, which is a most probable state sequence. In the context of HMMs, this is known as Viterbi decoding (see Section 17.4.4). Figure 17.10 illustrates the difference between filtering, smoothing and MAP decoding for the occasionally dishonest casino HMM. We see that the smoothed (offline) estimate is indeed smoother than the filtered (online) estimate. If we threshold the estimates at 0.5 and compare to the true sequence, we find that the filtered method makes 71 errors out of 300, and the smoothed method makes 49/300; the MAP path makes 60/300 errors. It is not surprising that smoothing makes fewer errors than Viterbi, since the optimal way to minimize bit-error rate is to threshold the posterior marginals (see Section 5.7.1.1). Nevertheless, for some applications, we may prefer the Viterbi decoding, as we discuss in Section 17.4.4.

- Posterior samples If there is more than one plausible interpretation of the data, it can be useful to sample from the posterior, $z_1:T \sim p(z_1:T|x_1:T)$. These sample paths contain much more information than the sequence of marginals computed by smoothing.

- Probability of the evidence We can compute the probability of the evidence, $p(x_1:T)$, by summing up over all hidden paths, $p(x_1:T) = \sum_{z_1:T} p(z_1:T, x_1:T)$. This can be used to classify sequences (e.g., if the HMM is used as a class conditional density), for model-based clustering, for anomaly detection, etc.

17.4.2 The forwards algorithm

We now describe how to recursively compute the filtered marginals, $p(z_t|x_1:t)$ in an HMM. The algorithm has two steps. First comes the prediction step, in which we compute the one-step-ahead predictive density; this acts as the new prior for time t :

$$p(z_t = j|x_1:t-1) = \sum_i p(z_t = j|z_{t-1} = i)p(z_{t-1} = i|x_1:t-1)$$

(17.44)

Next comes the update step, in which we absorb the

observed data from time t using Bayes rule:

$$\alpha_{t(j)} = p(z_t = j|x_1:t) = p(z_t = j|xt, x_1:t-1) \quad (17.45)$$

$$= \sum_j Z_t p(x_t|z_t = j, x_1:t-1) p(z_t = j|x_1:t-1) \quad (17.46)$$

where the normalization constant is given by

$$Z_t = p(x_t|x_1:t-1) = \sum_j p(z_t = j|x_1:t-1) p(x_t|z_t = j) \quad (17.47)$$

This process is known as the predict-update cycle. The distribution $p(z_t|x_1:t)$ is called the (filtered) belief state at time t, and is a vector of K numbers, often denoted by α_t .

In matrixvector notation, we can write the update in the following simple form:

$$\alpha_t \propto \psi_t (\Psi T \alpha_{t-1}) \quad (17.48)$$

where $\psi_t(j) = p(x_t|z_t = j)$ is the local evidence at time t, $\Psi(i, j) = p(z_t = j|z_{t-1} = i)$ is the transition matrix, and uv is the Hadamard product, representing elementwise vector multiplication.

In addition to computing the hidden states, we can use this algorithm to compute the log probability of the evidence:

$$\log p(x_1:T|\theta) = \sum_{t=1}^T \log p(x_t|x_1:t-1) = \sum_{t=1}^T \log Z_t \quad (17.49)$$

17.4.3 The forwards-backwards algorithm

In Section 17.4.2, we explained how to compute the filtered marginals $p(z_t = j|x_1:t)$ using online inference. We now discuss how to compute the smoothed marginals, $p(z_t = j|x_1:T)$, using offline inference.

17.4.3.1 Basic idea

The key decomposition relies on the fact that we can break the chain into two parts, the past and the future, by conditioning on z_t :

$$p(z_t = j|x_1:T) \propto p(z_t = j, x_{t+1:T}|x_1:t) \propto p(z_t = j|x_1:t) p(x_{t+1:T}|z_t = j, x_1:t) \quad (17.50)$$

Let $\alpha_t(j) = p(z_t = j|x_1:t)$ be the filtered belief state as before. Also, define

$\beta_t(j) = p(x_{t+1:T}|z_t = j)$ (17.51) as the conditional likelihood of future evidence given that the hidden state at time t is j. (Note that this is not a probability distribution over states, since it does not need to satisfy $\sum_j \beta_t(j) = 1$.) Finally, define

$\gamma_t(j) = p(z_t = j|x_1:T)$ (17.52) as the desired smoothed posterior marginal. From Equation 17.50, we have

$$\gamma_t(j) \propto \alpha_t(j) \beta_t(j) \quad (17.53)$$

We have already described how to recursively compute the α 's in a left-to-right fashion in Section 17.4.2. We now describe how to recursively compute the β 's in a right-to-left fashion. If we have already computed β_t , we can compute β_{t-1} as follows:

$$\beta_{t-1}(i) = p(x_t:T|z_{t-1} = i) \quad (17.54)$$

$$= \sum_j p(z_t = j, x_t, x_{t+1:T}|z_{t-1} = i) \quad (17.55)$$

$$= \sum_j p(x_{t+1:T}|z_t = j, z_{t-1} = i, x_t) p(z_t = j, x_t|z_{t-1} = i) \quad (17.56)$$

$$= \sum_j p(x_{t+1:T}|z_t = j) p(x_t|z_t = j, z_{t-1} = i) p(z_t = j|z_{t-1} = i)$$

(17.57)

$$= j \beta t(j) \psi t(j) \psi(i, j) \quad (17.58)$$

We can write the resulting equation in matrix-vector form as

$$\beta t-1 = \Psi(\psi t \beta t) \quad (17.59)$$

The base case is

$$\beta T(i) = p(x_{T+1:T} | z_T = i) = p(\emptyset | z_T = i) = 1 \quad (17.60)$$

which is the probability of a non-event. Having computed the forwards and backwards messages, we can combine them to compute $\gamma_t(j) \propto \alpha_t(j)\beta_t(j)$. The overall algorithm is known as the forwards-backwards algorithm. The pseudo code is very similar to the forwards case; see hmmFwdBack for an implementation. We can think of this algorithm as passing “messages” from left to right, and then from right to left, and then combining them at each node. We will generalize this intuition in Section 20.2, when we discuss belief propagation.

17.4.3.2 Two-slice smoothed marginals

When we estimate the parameters of the transition matrix using EM (see Section 17.5), we will need to compute the expected number of transitions from state i to state j :

$$N_{ij} = T-1 \sum_{t=1}^T E[I(z_t = i, z_{t+1} = j) | x_1:T] = T-1 \sum_{t=1}^T p(z_t = i, z_{t+1} = j | x_1:T) \quad (17.61)$$

The term $p(z_t = i, z_{t+1} = j | x_1:T)$ is called a (smoothed) two-slice marginal, and can be computed as follows

$$\xi_{t,t+1}(i, j) = p(z_t = i, z_{t+1} = j | x_1:T) \quad (17.62)$$

$$\propto p(z_t | x_1:t)p(z_{t+1} | z_t, x_{t+1:T}) \quad (17.63)$$

$$\propto p(z_t | x_1:t)p(x_{t+1:T} | z_t, z_{t+1})p(z_{t+1} | z_t) \quad (17.64)$$

$$\propto p(z_t | x_1:t)p(x_{t+1} | z_t)p(x_{t+2:T} | z_{t+1})p(z_{t+1} | z_t) \quad (17.65)$$

$$= \alpha_t(i)\phi_{t+1}(j)\beta_{t+1}(j)\psi(i, j) \quad (17.66)$$

In matrix-vector form, we have

$$\xi_{t,t+1} \propto \Psi(\alpha_t\phi_{t+1}\beta_{t+1}T) \quad (17.67)$$

For another interpretation of these equations, see Section 20.2.4.3.

17.4.3.3 Time and space complexity

It is clear that a straightforward implementation of FB takes $O(K^2T)$ time, since we must perform a $K \times K$ matrix multiplication at each step. For some applications, such as speech recognition, K is very large, so the $O(K^2)$ term becomes prohibitive. Fortunately, if the transition matrix is sparse, we can reduce this substantially. For example, in a left-to-right transition matrix, the algorithm takes $O(TK)$ time. In some cases, we can exploit special properties of the state space, even if the transition matrix is not sparse. In particular, suppose the states represent a discretization of an underlying continuous state-space, and the transition matrix has the form $\psi(i, j) \propto \exp(-\sigma^2|z_i - z_j|)$, where z_i is the continuous vector represented by state i . Then one can implement the forwards/backwards

algorithm in $O(TK \log K)$ time. This is very useful for models with large state spaces. See Section 22.2.6.1 for details. In some cases, the bottleneck is memory, not time. The expected sufficient statistics needed by EM are $\xi_{t-1,t}(i, j)$; this takes constant space (independent of T); however, to compute them, we need $O(KT)$ working space, since we must store α_t for $t = 1, \dots, T$ until we do the backwards pass. It is possible to devise a simple divide-and-conquer algorithm that reduces the space complexity from $O(KT)$ to $O(K \log T)$ at the cost of increasing the running time from $O(K^2T)$ to $O(K^2T \log T)$: see (Binder et al. 1997; Zweig and Padmanabhan 2000) for details.

17.4.4 The Viterbi algorithm

The Viterbi algorithm (Viterbi 1967) can be used to compute the most probable sequence of states in a chain-structured graphical model, i.e., it can compute

$$z^* = \operatorname{argmax}_{z_1:T} p(z_1:T|x_1:T) \quad (17.68)$$

This is equivalent to computing a shortest path through the trellis diagram in Figure 17.12, where the nodes are possible states at each time step, and the node and edge weights are log probabilities. That is, the weight of a path z_1, z_2, \dots, z_T is given by

$$\log \pi_1(z_1) + \log \varphi_1(z_1) + \sum_{t=2}^T [\log \psi_{t-1,z_t} + \log \varphi_t(z_t)] \quad (17.69)$$

17.4.4.1 MAP vs MPE

Before discussing how the algorithm works, let us make one important remark: the (jointly) most probable sequence of states is not necessarily the same as the sequence of (marginally) most probable states. The former is given by Equation 17.68, and is what Viterbi computes, whereas the latter is given by the maximizer of the posterior marginals or MPM:

$$\hat{z} = (\arg \max z_1 p(z_1|x_1:T), \dots, \arg \max z_T p(z_T|x_1:T)) \quad (17.70)$$

As a simple example of the difference, consider a chain with two time steps, defining the following joint:

$$\begin{aligned} X_1 = 0 & X_1 = 1 \\ X_2 = 0 & 0.04 \quad 0.3 \quad 0.34 \\ X_2 = 1 & 0.36 \quad 0.3 \quad 0.66 \\ & 0.4 \\ & 0.6 \end{aligned}$$

The joint MAP estimate is $(0, 1)$, whereas the sequence of marginal MPMs is $(1, 1)$. The advantage of the joint MAP estimate is that it is always globally consistent. For example, suppose we are performing speech recognition and someone says "recognize speech". This could be mis-heard as "wreck a nice beach". Locally it may appear that "beach" is the most probable interpretation of that particular window of sound, but when we add the requirement that the data be explained by a single linguistically plausible path, this interpretation becomes less likely. On the other hand, the MPM estimates can be more robust (Marroquin et al. 1987). To see why, note that

in Viterbi, when we estimate z_t , we “max out” the other variables:

$$z^* t = \operatorname{argmax} z_t \max_{z_1:t-1, z_{t+1:T}} p(z_1:t-1, z_t, z_{t+1:T} | x_1:T) \quad (17.71)$$

whereas when we use forwards-backwards, we sum out the other variables:

$$p(z_t | x_1:T) = \sum_{z_1:t-1, z_{t+1:T}} p(z_1:t-1, z_t, z_{t+1:T} | x_1:T) \quad (17.72)$$

This makes the MPM in Equation 17.70 more robust, since we estimate each node averaging over its neighbors, rather than conditioning on a specific value of its neighbors.

17.4.4.2 Details of the algorithm

It is tempting to think that we can implement Viterbi by just replacing the sum-operator in forwards-backwards with a max-operator. The former is called the sum-product, and the latter the max-product algorithm. If there is a unique mode, running max-product and then computing using Equation 17.70 will give the same result as using Equation 17.68 (Weiss and Freeman 2001b), but in general, it can lead to incorrect results if there are multiple equally probable joint assignments. The reason is that each node breaks ties independently and hence may do so in a manner that is inconsistent with its neighbors. The Viterbi algorithm is therefore not quite as simple as replacing sum with max. In particular, the forwards pass does use maxproduct, but the backwards pass uses a traceback procedure to recover the most probable path through the trellis of states. Essentially, once z_t picks its most probable state, the previous nodes condition on this event, and therefore they will break ties consistently. In more detail, define

$$\delta_t(j) = \max_{z_1, \dots, z_{t-1}} p(z_1:t-1, z_t = j | x_1:t) \quad (17.73)$$

This is the probability of ending up in state j at time t , given that we take the most probable path. The key insight is that the most probable path to state j at time t must consist of the most probable path to some other state i at time $t - 1$, followed by a transition from i to j .

Hence

$$\delta_t(j) = \max_i \delta_{t-1}(i) \psi(i, j) \phi_t(j) \quad (17.74)$$

We also keep track of the most likely previous state, for each possible state that we end up in:

$$at(j) = \operatorname{argmax}_i \delta_{t-1}(i) \psi(i, j) \phi_t(j) \quad (17.75)$$

That is, $at(j)$ tells us the most likely previous state on the most probable path to $z_t = j$. We initialize by setting

$$\delta_1(j) = \pi_j \phi_1(j) \quad (17.76)$$

and we terminate by computing the most probable final state z^* :

$$z^* T = \operatorname{argmax}_i \delta_T(i) \quad (17.77)$$

We can then compute the most probable sequence of

states using traceback:

$$z^* t = \text{at} + 1(z^* t+1) \quad (17.78)$$

As usual, we have to worry about numerical underflow. We are free to normalize the δ_t terms at each step; this will not affect the maximum. However, unlike the forwards-backwards case, we can also easily work in the log domain. The key difference is that $\log \max = \max \log$, whereas $\log = \log$. Hence we can use

$$\log \delta_t(j) = \max z_{1:t-1} \log p(z_{1:t-1}, z_t = j | x_{1:t}) \quad (17.79)$$

$$= \max i \log \delta_{t-1}(i) + \log \psi(i, j) + \log \varphi_t(j) \quad (17.80)$$

In the case of Gaussian observation models, this can result in a significant (constant factor) speedup, since computing $\log p(x_t | z_t)$ can be much faster than computing $p(x_t | z_t)$ for a highdimensional Gaussian. This is one reason why the Viterbi algorithm is widely used in the E step of EM (Section 17.5.2) when training large speech recognition systems based on HMMs.

17.4.4.3 Example

Figure 17.13 gives a worked example of the Viterbi algorithm, based on (Russell et al. 1995). Suppose we observe the discrete sequence of observations $x_{1:4} = (C_1, C_3, C_4, C_6)$, representing codebook entries in a vector-quantized version of a speech signal. The model starts in state z_1 . The probability of generating C_1 in z_1 is 0.5, so we have $\delta_1(1) = 0.5$, and $\delta_1(i)=0$ for all other states. Next we can self-transition to z_1 with probability 0.3, or

transition to z_2 with probability 0.7. If we end up in z_1 , the probability of generating C_3 is 0.3; if we end up in z_2 , the probability of generating C_3 is 0.2. Hence we have

$$\delta_2(1) = \delta_1(1)\psi(1, 1)\varphi_2(1) = 0.5 \cdot 0.3 \cdot 0.3 = 0.045 \quad (17.81)$$

$$\delta_2(2) = \delta_1(1)\psi(1, 2)\varphi_2(2) = 0.5 \cdot 0.7 \cdot 0.2 = 0.07 \quad (17.82)$$

Thus state 2 is more probable at $t = 2$; see the second column of Figure 17.13(b). In time step 3, we see that there are two paths into z_2 , from z_1 and from z_2 . The bold arrow indicates that the latter is more probable. Hence this is the only one we have to remember. The algorithm continues in this way until we have reached the end of the sequence. Once we have reached the end, we can follow the black arrows back to recover the MAP path (which is 1,2,2,3).

17.4.4.4 Time and space complexity

The time complexity of Viterbi is clearly $O(K^2T)$ in general, and the space complexity is $O(KT)$, both the same as forwards-backwards. If the transition matrix has the form $\psi(i, j) \propto \exp(-\sigma_2 \|z_i - z_j\|^2)$, where z_i is the continuous vector represented by state i , we can implement Viterbi in $O(TK)$ time, instead of $O(TK \log K)$ needed by forwards-backwards. See Section 22.2.6.1 for details.

17.4.4.5 N-best list

The Viterbi algorithm returns one of the most probable paths. It can be extended to return the top N paths (Schwarz and Chow 1990; Nilsson and Goldberger 2001). This is called the N-best list. Once can then use a discriminative method to rerank the paths based on global features derived from the fully observed state sequence (as well as the visible features). This technique is widely used in speech recognition. For example, consider the sentence "recognize speech". It is possible that the most probable interpretation by the system of this acoustic signal is "wreck a nice speech", or maybe "wreck a nice beach". Maybe the correct interpretation is much lower down on the list. However, by using a re-ranking system, we may be able to improve the score of the correct interpretation based on a more global context. One problem with the N-best list is that often the top N paths are very similar to each other, rather than representing qualitatively different interpretations of the data. Instead we might want to generate a more diverse set of paths to more accurately represent posterior uncertainty. One way to do this is to sample paths from the posterior, as we discuss below. For some other ways to generate diverse MAP estimates, see e.g., (Yadollahpour et al. 2011; Kulesza and Taskar 2011).

17.4.5 Forwards filtering, backwards sampling

It is often useful to sample paths from the posterior:

$$z_{1:T} \sim p(z_{1:T}|x_{1:T}) \quad (17.83)$$

We can do this is as follow: run forwards backwards, to compute the two-slice smoothed posteriors, $p(z_{t-1:T}|x_{1:T})$; next compute the conditionals $p(z_t|z_{t-1}, x_{1:T})$ by normalizing; sample from the initial pair of states, $z_{1,2} \sim p(z_{1,2}|x_{1:T})$; finally, recursively sample $z_{t+1:T} \sim p(z_{t+1:T}|z_t, x_{1:T})$. Note that the above solution requires a forwards-backwards pass, and then an additional forwards sampling pass. An alternative is to do the forwards pass, and then perform sampling in the backwards pass. The key insight into how to do this is that we can write the joint from right to left using

$$p(z_{1:T}|x_{1:T}) = p(z_T|x_{1:T}) \prod_{t=1}^{T-1} p(z_t|z_{t+1}, x_{1:T}) \quad (17.84)$$

We can then sample z_T given future sampled states using

$$z_T \sim p(z_T|z_{T+1:T}, x_{1:T}) = p(z_T|z_{T+1}, z_{T+2:T}, x_{1:T}, x_{T+1:T}) = p(z_T|z_{T+1}, x_{1:T}) \quad (17.85)$$

The sampling distribution is given by

$$p(z_t = i|z_{t+1} = j, x_{1:t}) = p(z_t|z_{t+1}, x_{1:t}, x_{t+1}) \quad (17.86)$$

$$= p(z_{t+1}, z_t|x_{1:t+1}) p(z_{t+1}|x_{1:t+1}) \quad (17.87)$$

$$\propto p(x_{t+1}|z_{t+1}, z_t, x_{1:t}) p(z_{t+1}|z_t, x_{1:t}) p(z_{t+1}|x_{1:t+1}) \quad (17.88)$$

$$= p(x_{t+1}|z_{t+1}) p(z_{t+1}|z_t, x_{1:t}) p(z_t|x_{1:t}) p(z_{t+1}|x_{1:t+1}) \\ (17.89)$$

$$= \varphi_{t+1}(j) \psi(i, j) \alpha_t(i) \alpha_{t+1}(j) \quad (17.90)$$

The base case is

$$z_T \sim p(z_T = i|x_1:T) = \alpha T(i) \quad (17.91)$$

This algorithm forms the basis of blocked-Gibbs sampling methods for parameter inference, as we will see below.

已思考 21s

下面先把你给的内容逐句完整翻译；然后做通俗解释（配直觉和例子）；最后给要点小结。我保留了原文里的公式与编号。

逐句翻译（无遗漏）

图 17.9 “偶尔不诚实的赌场”的 HMM。蓝色箭头可视化了状态转移图 A 。

图 17.10 在“不诚实赌场”上的推断。竖直的灰条表示我们是用**假骰子** (loaded die) 生成的样本。(a) 使用**滤波** (filtered) 得到“当前在用假骰子”的概率估计。(b) 使用**平滑** (smoothed) 的估计。(c) **MAP** (最可能路径) 轨迹。

图 17.11 状态空间模型的主要推断类型。灰色区间是我们有数据的时间段。箭头表示我们要做推断的时间点。 t 为当前时刻， T 为序列长度， l 为滞后 (lag)， h 为预测视野 (prediction horizon)。详见正文。

图 17.12 马尔科夫链的**编织图** (trellis, 状态×时间的网格)。

图 17.13 在一个简单的语音识别 HMM 上演示 Viterbi 解码。(a) 一个单个音素的 3 状态 HMM。我们在可视化状态转移图。假设观测已经**向量量化**到 7 个可能的符号 C_1, \dots, C_7 。每个状态 z_1, z_2, z_3 在这些符号上有不同的分布。改编自 Russell & Norvig (2002) 图 15.20。(b) 把 Viterbi 算法应用到该模型，数据序列为 C_1, C_3, C_4, C_6 。列表示时间，行表示状态。从 $t-1$ 的状态 i 指向 t 的状态 j 的箭头上标了两个数字：第一个是 $i \rightarrow j$ 的转移概率，第二个是在状态 j 生成观测 x_t 的概率。加粗的线/圆圈表示最可能的状态序列。

17.4 HMM 中的推断

现在我们讨论在**参数已知**的前提下，如何推断 HMM 的隐藏状态序列。完全相同的算法也适用于其它**链结构**的图模型，例如链式 CRF (见 19.6.1 节)。在第 20 章我们把这些方法推广到任意图；在 17.5.2 节我们会说明如何把推断的输出用于**参数估计**。

17.4.1 时间模型中的推断类型

HMM (以及一般的状态空间模型) 里有多种推断任务。为说明差异，我们用 (Durbin et al., 1998) 的“**偶尔不诚实的赌场**”作为例子：

- $x_t \in \{1, \dots, 6\}$ 表示掷骰出现的点数；
- z_t 表示当前使用的骰子的**身份**。

大多数时候赌场用**公平骰** $z = 1$ ；偶尔短时间换成**偏置骰** $z = 2$ 。若 $z = 1$ ，观测分布

是 $\{1, \dots, 6\}$ 上的**均匀多项式**；若 $z = 2$, 观测分布对“6点”偏斜（见图 17.9）。从该模型采样，可能得到如下数据：

清单 17.1 casinoDemo 的示例输出

```
Rolls: 664153216162115234653214356634261655234232315142464156663246
Die:    LLLLLLLLLLFFFFFLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
```

其中“Rolls”是观测到的点数，“Die”是隐藏状态（L=loaded 偏置，F=fair 公平）。可见模型生成一串符号，但分布的统计特性会时不时**突变**。在典型应用里我们只看到 Rolls，想推断此时用的是哪种骰子。不同的推断方式如下：

- **滤波 (Filtering)**：在线地（递推地）计算 $p(z_t | x_{1:t})$ 。叫“滤波”的原因是它比仅用当前 $p(z_t | x_t)$ 更能**降噪**。下面会看到它就是按时间顺序应用贝叶斯公式。示例见图 17.10(a)。
- **平滑 (Smoothing)**：离线地计算 $p(z_t | x_{1:T})$ ，即利用**所有**证据。示例见图 17.10(b)。通过考虑过去与未来，我们的不确定性显著下降。直观比喻：侦探在案发现场四处摸索，直到拿到关键线索，“啊哈”一下，之前模糊的现象也都解释清楚了。
- **固定滞后平滑 (Fixed-lag smoothing)**：介于在线和离线之间——计算 $p(z_{t-l} | x_{1:t})$ ，其中 $l > 0$ 为滞后。比滤波更准，但有一点**延迟**。调节 l 可以在准确性与延迟之间折中。
- **预测 (Prediction)**：不是用未来帮助解释过去，而是用过去预测未来：计算 $p(z_{t+h} | x_{1:t})$ ，其中 $h > 0$ 为预测视野。例如 $h = 2$ 时

$$p(z_{t+2} | x_{1:t}) = \sum_{z_{t+1}, z_t} p(z_{t+2} | z_{t+1}) p(z_{t+1} | z_t) p(z_t | x_{1:t}) \quad (17.42)$$

做法：对转移矩阵**求幂**并作用到当前信念。再由

$$p(x_{t+h} | x_{1:t}) = \sum_{z_{t+h}} p(x_{t+h} | z_{t+h}) p(z_{t+h} | x_{1:t}) \quad (17.43)$$

得到对未来观测的预测（后验预测密度），可用于时间序列预测。三者关系示意见图 17.11。

- **MAP 估计**：求最可能的整条状态序列

$$\arg \max_{z_{1:T}} p(z_{1:T} | x_{1:T})$$

在 HMM 里叫**Viterbi 解码**（17.4.4 节）。图 17.10 展示了滤波、平滑和 MAP 的差别。把概率阈值设为 0.5 与真实序列对比：滤波在 300 次里错 71 次，平滑错 49/300，MAP 路径错 60/300。平滑错得更少并不奇怪：要最小化**逐点错误率**，就应对每个时刻的后验边缘取阈值（见 5.7.1.1）。不过有些任务我们会更偏好 Viterbi（见 17.4.4）。

- **后验采样**：若数据有多种合理解释，采样 $z_{1:T} \sim p(z_{1:T} | x_{1:T})$ 很有用。这些“样本路径”包含的信息比逐点的平滑边缘丰富得多。
- **证据概率**：通过对所有隐藏路径求和

$$p(x_{1:T}) = \sum_{z_{1:T}} p(z_{1:T}, x_{1:T})$$

可得到证据概率，用于**序列分类**（HMM 作类别条件密度）、**基于模型的聚类、异常检测等**。

17.4.2 前向 (forwards) 算法

下面描述如何**递推地**计算滤波边缘 $p(z_t | x_{1:t})$ 。两步：

1. **预测**（一步超前预测，作为时刻 t 的先验）

$$p(z_t = j | x_{1:t-1}) = \sum_i p(z_t = j | z_{t-1} = i) p(z_{t-1} = i | x_{1:t-1}) \quad (17.44)$$

2. **更新**（用贝叶斯公式吸收 t 时刻观测）

$$\alpha_t(j) = p(z_t = j | x_{1:t}) = p(z_t = j | x_t, x_{1:t-1}) = \frac{1}{Z_t} p(x_t | z_t = j, x_{1:t-1}) p(z_t = j | x_{1:t-1}) \quad (17.45)$$

其中归一化常数

$$Z_t = p(x_t | x_{1:t-1}) = \sum_j p(z_t = j | x_{1:t-1}) p(x_t | z_t = j) \quad (17.46)$$

这就是**预测-更新循环**。 $p(z_t | x_{1:t})$ 称为（滤波的）**信念状态**，是长度为 K 的向量，常记为 α_t 。用矩阵-向量可写成

$$\alpha_t \propto \psi_t \odot (\Psi^T \alpha_{t-1}) \quad (17.47)$$

其中 $\psi_t(j) = p(x_t | z_t = j)$ 是**局部证据**， $\Psi(i,j) = p(z_t = j | z_{t-1} = i)$ 是转移矩阵， \odot 表示按元素相乘。

此外，该算法还能得到证据的对数概率：

$$\log p(x_{1:T} | \theta) = \sum_{t=1}^T \log p(x_t | x_{1:t-1}) = \sum_{t=1}^T \log Z_t \quad (17.48)$$

17.4.3 前向-后向 (forwards-backwards) 算法

上一小节讲了如何在线算滤波边缘 $p(z_t = j | x_{1:t})$ 。现在讲如何离线算平滑边缘 $p(z_t = j | x_{1:T})$ 。

17.4.3.1 基本想法

关键分解：在给定 z_t 后，链被切成“过去”与“未来”两段：

$$p(z_t = j | x_{1:T}) \propto p(z_t = j, x_{t+1:T} | x_{1:t}) \propto p(z_t = j | x_{1:t}) p(x_{t+1:T} | z_t = j, x_{1:t}) \quad (17.49)$$

设

$$\alpha_t(j) = p(z_t = j | x_{1:t})$$

为滤波信念；再定义

$$\beta_t(j) = p(x_{t+1:T} \mid z_t = j) \quad (17.51)$$

为“从 $t + 1$ 到 T 的未来证据”在 $z_t = j$ 下的条件似然（它不是分布，不需 $\sum_j \beta_t(j) = 1$ ）；
最后

$$\gamma_t(j) = p(z_t = j \mid x_{1:T}) \quad (17.52)$$

为所求平滑边缘。由 (17.50) 得

$$\gamma_t(j) \propto \alpha_t(j)\beta_t(j) \quad (17.53)$$

我们已会递推计算 α （从左到右）。现在给出 β 的右到左递推。若已算出 β_t ，则

$$\begin{aligned} \text{\&=}\beta_{t-1}(i) &= p(x_{t:T} \mid z_{t-1}=i) \tag{17.54} \\ &\text{\&=}\sum_j p(z_{t=j}, x_{t:T} \mid z_{t-1}=i) \tag{17.55} \\ &\text{\&=}\sum_j p(x_{t:T} \mid z_{t-1}=i) p(z_{t=j} \mid z_{t-1}=i) \tag{17.57} \\ &\text{\&=}\sum_j \beta_t(j) \psi_t(j) \tag{17.58} \end{aligned}$$

矩阵-向量形式：

$$\beta_{t-1} = \Psi(\psi_t \odot \beta_t) \quad (17.59)$$

边界条件：

$$\beta_T(i) = p(x_{T+1:T} \mid z_T = i) = p(\emptyset \mid z_T = i) = 1 \quad (17.60)$$

算完前向与后向“消息”，在每个 t 处合并得到 $\gamma_t(j) \propto \alpha_t(j)\beta_t(j)$ 。整体算法就叫前向-后向。可把它理解为“从左到右传一次消息，再从右到左传一次，然后在节点处相乘”；这种思路在 20.2 节“信念传播”会推广。

17.4.3.2 两片 (two-slice) 平滑边缘

用 EM 估计转移矩阵时需要“从 i 转到 j ”的期望次数：

$$N_{ij} = \sum_{t=1}^{T-1} E[\mathbf{I}(z_t = i, z_{t+1} = j) \mid x_{1:T}] = \sum_{t=1}^{T-1} p(z_t = i, z_{t+1} = j \mid x_{1:T}) \quad (17.61)$$

其中

$$\xi_{t,t+1}(i,j) = p(z_t = i, z_{t+1} = j \mid x_{1:T}) \quad (17.62)$$

可按下式求（把 ϕ 视为“发射概率”，与上文的 ψ 含义一致）：

$$\xi_{t,t+1}(i,j) = \alpha_t(i) \phi_{t+1}(j) \beta_{t+1}(j) \Psi(i,j) \quad (\propto \text{式 } 17.63-17.66) \quad (17.66)$$

矩阵-向量形式：

$$\xi_{t,t+1} \propto \Psi(\alpha_t(\phi_{t+1} \odot \beta_{t+1})^T) \quad (17.67)$$

17.4.3.3 时间与空间复杂度

朴素实现 FB 的时间为 $O(K^2 T)$ （每步要做一次 $K \times K$ 乘法）。在语音识别等任务 K 很大，这会很贵。幸运的是，若转移矩阵稀疏，可大幅降低；例如“从左到右”的结构为

$O(KT)$ 。某些情况下即便不稀疏也能利用结构：若状态是某个连续空间的离散化，且 $\Psi(i,j) \propto \exp(-\sigma^2 \|z_i - z_j\|)$ ，可把 FB 做到 $O(TK \log K)$ （见 22.2.6.1）。

有时瓶颈是**内存**而不是时间。EM 需要的期望充分统计量 $\sum_t \zeta_{t-1,t}(i,j)$ 只要常数空间，但要计算它们，工作内存需 $O(KT)$ ，因为得把所有 α_t 存到做后向时用。可以用分治把空间从 $O(KT)$ 降到 $O(K \log T)$ ，代价是时间从 $O(K^2T)$ 增到 $O(K^2T \log T)$ (Binder et al., 1997; Zweig & Padmanabhan, 2000)。

17.4.4 Viterbi 算法

Viterbi (Viterbi 1967) 用来求**最可能的整条状态序列**

$$z^{*} = \arg \max_{z_{1:T}} p(z_{1:T} | x_{1:T}) \quad (17.68)$$

这等价于在图 17.12 的 trellis 上求**最短路**（把对数概率当边/点权）。一条路径 z_1, \dots, z_T 的权重为

$$\log \pi_1(z_1) + \log \phi_1(z_1) + \sum_{t=2}^T [\log \Psi(z_{t-1}, z_t) + \log \phi_t(z_t)] \quad (17.69)$$

17.4.4.1 MAP vs MPM (联合 MAP 与逐点 MPM)

重要提醒：**联合最可能序列** (Viterbi 求的) 与**逐点最可能状态序列** (对每个 t 最大化 $p(z_t | x)$ 的 MPM) 一般不一样。后者为

$$\hat{z} = (\arg \max_{z_1} p(z_1 | x), \dots, \arg \max_{z_T} p(z_T | x)) \quad (17.70)$$

例子 (两步链, 联合分布为)：

	$X_1 = 0$	$X_1 = 1$	合计
$X_2 = 0$	0.04	0.30	0.34
$X_2 = 1$	0.36	0.30	0.66
合计	0.40	0.60	1

联合 MAP 为 (0,1)，而逐点 MPM 为 (1,1)。

- **联合 MAP 总是全局一致** (比如语音里“recognize speech”不会被拼成若干局部最优而整体不通顺)。
- **MPM 往往更稳健**: Viterbi 在估计 z_t 时对其他变量做 **max**

$$z_t^{*} = \arg \max_{z_t} \max_{z_{1:t-1}, z_{t+1:T}} p(z_{1:T} | x) \quad (17.71)$$

而前向-后向是对其他变量做 **sum**

$$p(z_t | x) = \sum_{z_{1:t-1}, z_{t+1:T}} p(z_{1:T} | x) \quad (17.72)$$

“求和”相当于对邻居取平均，往往更鲁棒。

17.4.4.2 算法细节

不能简单把 FB 里的“求和”换成“取最大”。Viterbi 的前向确实是 max-product，但后向要用回溯找最优路径，确保各处一致地处理“并列最优”。

定义

$$\delta_t(j) = \max_{z_1, \dots, z_{t-1}} p(z_{1:t-1}, z_t = j \mid x_{1:t}) \quad (17.73)$$

即“到达时刻 t 、位于状态 j 的最优路径概率”。关键是“到 j 的最优路径 = 某个 i 在 $t-1$ 的最优路径 + $i \rightarrow j$ 的转移”，所以

$$\delta_t(j) = \max_i \delta_{t-1}(i) \Psi(i, j) \phi_t(j) \quad (17.74)$$

并记下“最可能的上一个状态”

$$a_t(j) = \arg \max_i \delta_{t-1}(i) \Psi(i, j) \phi_t(j) \quad (17.75)$$

初始化

$$\delta_1(j) = \pi_j \phi_1(j) \quad (17.76)$$

结束时取最终最可能状态

$$z_T^* = \arg \max_i \delta_T(i) \quad (17.77)$$

再回溯得到整条路径

$$z_t^* = a_{t+1}(z_{t+1}^*) \quad (17.78)$$

数值稳定：可对 δ_t 每步归一化；更方便的是直接在对数域运算，因为 $\log \max = \max \log$ ：

$$\log \delta_t(j) = \max_i \{\log \delta_{t-1}(i) + \log \Psi(i, j) + \log \phi_t(j)\} \quad (17.80)$$

在高维高斯的观测模型中， $\log p(x_t \mid z_t)$ 往往比 $p(\cdot)$ 更快算，这是 Viterbi 在大型语音系统 EM 的 E 步里广泛使用的原因之一。

17.4.4.3 例子

图 17.13 给出一个演示（改编自 Russell 等，1995）。观测序列 $x_{1:4} = (C1, C3, C4, C6)$ 。起始于 z_1 。在 z_1 生成 $C1$ 的概率为 0.5，故 $\delta_1(1) = 0.5$ ，其余为 0。接着要么以 0.3 自转到 z_1 ，要么以 0.7 转到 z_2 。若到 z_1 生成 $C3$ 的概率为 0.3；若到 z_2 生成 $C3$ 的概率为 0.2。于是

$$\delta_2(1) = 0.5 \cdot 0.3 \cdot 0.3 = 0.045 \quad (17.81)$$

$$\delta_2(2) = 0.5 \cdot 0.7 \cdot 0.2 = 0.07 \quad (17.82)$$

因此 $t=2$ 更可能在状态 2（见图 17.13(b) 第二列）。 $t=3$ 时到 z_2 有两条路（来自 z_1 与来自 z_2 ），加粗箭头表示后者更大，只需记它即可。如此继续到序列末端，再沿黑箭头回

溯得到 MAP 路径 (1,2,2,3) 。

17.4.4.4 复杂度

Viterbi 的时间复杂度一般是 $O(K^2 T)$, 空间 $O(KT)$, 与 FB 相同。若 $\Psi(i,j) \propto \exp(-\sigma^2 \|z_i - z_j\|^2)$, 则 Viterbi 可做到 $O(TK)$ (FB 为 $O(TK \log K)$) ; 见 22.2.6.1。

17.4.4.5 N-best 列表

Viterbi 返回“一个最优路径”。可以扩展为返回前 N 条 (Schwarz & Chow, 1990; Nilsson & Goldberger, 2001), 称 N-best。之后可用判别式方法按“全局特征”重排序。在语音识别中广泛使用。例如“recognize speech”可能被系统最可能地解读为“wreck a nice speech”或“wreck a nice beach”, 真正的解释可能排得很靠后, 通过重排序可以把它提上来。N-best 的问题是前 N 条往往很像, 不够多样; 一个更好的办法是从后验采样路径 (见下文), 或者用产生多样 MAP 的方法 (如 Yadollahpour et al., 2011; Kulesza & Taskar, 2011) 。

17.4.5 前向滤波、后向采样

常常需要从后验采样路径:

$$z_{1:T}^{(s)} \sim p(z_{1:T} | x_{1:T}) \quad (17.83)$$

一种做法: 先跑 FB 得到两片平滑后验 $p(z_{t-1}, z_t | x)$, 归一化成条件 $p(z_t | z_{t-1}, x)$, 先采样 $z_{1,2} \sim p(z_{1,2} | x)$, 再递推采样 $z_t \sim p(z_t | z_{t-1}, x)$ 。

另一种更高效: 做完前向后, 在后向时采样。把后验写成“从右到左”的分解

$$p(z_{1:T} | x) = p(z_T | x) \prod_{t=1}^{T-1} p(z_t | z_{t+1}, x) \quad (17.84)$$

于是可按未来已采样的状态递推:

$$z_t^{(s)} \sim p(z_t | z_{t+1:T}, x) = p(z_t | z_{t+1}^{(s)}, x_{1:t}) \quad (17.85)$$

其采样分布为

$$p(z_t = i | z_{t+1} = j, x_{1:t}) = \frac{\phi_{t+1}(j) \Psi(i, j) \alpha_t(i)}{\alpha_{t+1}(j)} \quad (17.90)$$

边界条件:

$$z_T^{(s)} \sim p(z_T = i | x_{1:T}) = \alpha_T(i) \quad (17.91)$$

该方法是分块 Gibbs 参数推断的基础。

通俗解释 (把难点讲“人话”)

- 想象赌场**大多数**用公平骰（各点 1/6），**偶尔换成“爱出 6 点”的假骰**。我们只看到了点数（1–6），没看到何时换骰。
 - **滤波**：边看边猜“**现在**是不是在用假骰”。所以曲线会抖动（图 17.10a）。
 - **平滑**：把整段都看完再回头猜“当时是不是假骰”。因为用到了“未来线索”，曲线更平滑、错误更少（图 17.10b）。
 - **MAP (Viterbi)**：不是每个时刻独立地猜“假/真”，而是找**一条整体最可能的路径**（图 17.10c 的阶梯）。它全局自治，但逐点错误率不一定最小。
 - **前向-后向**：相当于“从左到右传一次信息（前向 α ），再从右到左传一次（后向 β ），最后相乘得平滑后验 γ ”。
 - **Viterbi**：像在网格上找最短路。前向阶段只记到达每个格子的**最优分数** $\delta_t(j)$ 和来自哪个前驱 $a_t(j)$ ，最后回溯取路径。
 - **预测**：把转移矩阵“求幂”，就能把“现在的信念”推进到未来 h 步。
 - **采样**：想看“还有哪些可能解释”，就从后验里抽几条状态路径；“前向滤波、后向采样”只需一次前向和一次带采样的后向。
-

要点小结

- 这节讲在**参数已知的 HMM 里**怎么推断隐藏状态：
 - **滤波** $p(z_t | x_{1:t})$ ：前向递推【预测+更新】；顺带得到 $\log p(x_{1:T})$ 。
 - **平滑** $p(z_t | x_{1:T})$ ：前向-后向； $\gamma_t \propto \alpha_t \beta_t$ 。
 - **两片边缘** $\zeta_{t,t+1}$ ：EM 估计转移时要用。
 - **复杂度**：一般 $O(K^2 T)$ ，稀疏或特殊结构可降到 $O(KT)$ 或 $O(TK \log K)$ 。
 - **Viterbi (MAP 序列)**：前向 max-product + 回溯；与逐点 MPM 不同。
 - **N-best**：取前 N 条路径以便重排名。
 - **前向滤波、后向采样**：高效从后验抽“整条路径”。
- 赌场例子贯穿始终：
 - 灰条 = 真正使用假骰的时间；
 - 滤波曲线抖、平滑曲线稳、Viterbi 给一条干脆的阶梯路径；
 - 平滑的逐点错误率通常最小，Viterbi 的路径整体一致。

如果你愿意，我可以用一个“2 状态、6 面骰”的微型数值例子，给你把 $\alpha, \beta, \gamma, \delta, a$ 每一步都算出来（含中间数）。

这一节主要在讲什么？我不太懂

已思考 11s

如果只记一句话：**17.4 节讲的是——在参数已知的 HMM 里，怎样从“能看到的观测”推断“看不见的状态”，以及几种常用的推断算法与它们的差别/代价。**

这节都干了啥 (超简版)

1. 要做的推断类型

- **滤波 Filtering**: 边看边猜“现在是哪种状态”。算 $p(z_t | x_{1:t})$ 。
- **平滑 Smoothing**: 看完整段再回头猜“当时是哪种状态”。算 $p(z_t | x_{1:T})$, 更准。
- **固定滞后平滑**: 晚 l 步再判断 $t - l$ 时刻, 折中“准确 vs 延迟”。
- **预测 Prediction**: 把“现在的信念”推到未来 h 步, 算 $p(z_{t+h} | x_{1:t})$ 与 $p(x_{t+h} | x_{1:t})$ 。
- **MAP 序列 (Viterbi)** : 找整条最可能路径 $\arg \max_{z_{1:T}} p(z_{1:T} | x_{1:T})$ 。
- **后验采样**: 从 $p(z_{1:T} | x_{1:T})$ 抽几条可能路径, 展示不确定性。
- **证据概率**: 求 $p(x_{1:T})$ 用于比模型/做分类。

2. 用到的核心算法

- **前向 (Forwards, 滤波)** = “预测 + 更新” 循环

预测: $\hat{\alpha}_t(j) = \sum_i \Psi(i,j) \alpha_{t-1}(i)$

更新: $\alpha_t(j) \propto \psi_t(j) \hat{\alpha}_t(j)$

(Ψ : 转移矩阵; $\psi_t(j) = p(x_t | z_t = j)$; α_t : 当前信念)

顺手得到对数似然 $\sum_t \log Z_t$ 。

- **前向-后向 (Forwards-Backwards, 平滑)**

前向得 α_t , 后向递推 $\beta_{t-1}(i) = \sum_j \Psi(i,j) \psi_t(j) \beta_t(j)$,

合并: $\gamma_t(j) \propto \alpha_t(j) \beta_t(j) = p(z_t = j | x_{1:T})$ 。

还会算“两片边缘” $\zeta_{t,t+1}(i,j)$ 给 EM 用 (统计 $i \rightarrow j$ 的期望次数)。

- **Viterbi (MAP 路径)**

动态规划记录最优得分 $\delta_t(j) = \max_i \delta_{t-1}(i) \Psi(i,j) \psi_t(j)$ 和回溯指针 $a_t(j)$; 末端取最大再回溯得到整条路径。

注意: **联合 MAP 路径** \neq **逐点最可能状态** (MPM) ; 前者全局一致, 后者逐点错误率常更低。

- **前向滤波、后向采样 (FFBS)**

先做前向 (得到 α) , 后向时按

$p(z_t = i | z_{t+1} = j, x) \propto \psi_{t+1}(j) \Psi(i,j) \alpha_t(i) / \alpha_{t+1}(j)$

逐步采样出一整条状态路径。

3. 复杂度 (算力账)

- 朴素前向-后向与 Viterbi: 时间 $O(K^2 T)$, 空间 $O(KT)$ 。
- 若转移稀疏/左到右: 时间可到 $O(KT)$ 。有些结构还能做到 $O(TK \log K)$ 或 $O(TK)$ 。
- 可用分治把空间从 $O(KT)$ 降到 $O(K \log T)$ (多一点时间换内存)。

4. 贯穿的直觉例子: 不诚实赌场

- 只看骰子点数, 想判断何时换了“偏爱 6 点”的假骰。
- **滤波曲线抖** (只用过去) ; **平滑更稳** (用前后文) ; **Viterbi**给一条“最可能”阶梯路径。

记住这 5 句就行

- 目标：从 $x_{1:T}$ 推断 $z_{1:T}$ 。
- 滤波 = 在线 $p(z_t | x_{1:t})$; 平滑 = 离线 $p(z_t | x_{1:T})$ 。
- 前向-后向： $\gamma_t \propto \alpha_t \beta_t$ 。
- Viterbi：动态规划 + 回溯求最可能整条路径。
- 需要更丰富的不确定性 → 采样整条路径（FFBS）。

想更直观，我可以用一个“2 状态骰子”小例子，给你把 $\alpha, \beta, \gamma, \delta, a$ 的数值一步步算出来。

翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

17.5 Learning for HMMs

We now discuss how to estimate the parameters $\Theta = (\pi, A, B)$, where $\pi(i) = p(z_1 = i)$ is the initial state distribution, $A(i, j) = p(z_t = j | z_{t-1} = i)$ is the transition matrix, and B are the parameters of the class-conditional densities $p(x_t | z_t = j)$.

We first consider the case where $z_{1:T}$ is observed in the training set, and then the harder case where $z_{1:T}$ is hidden.

17.5.1 Training with fully observed data

If we observe the hidden state sequences, we can compute the MLEs for A and π exactly as in Section 17.2.2.1. If we use a conjugate prior, we can also easily compute the posterior. The details on how to estimate B depend on the form of the observation model. The situation is identical to fitting a generative classifier. For example, if each state has a multinoulli distribution associated with it, with parameters $B_{jl} = p(X_t = l | z_t = j)$, where $l \in \{1, \dots, L\}$ represents the observed symbol, the MLE is given by

$$\hat{B}_{jl} = \frac{N}{N} \sum_{i=1}^N \sum_{t=1}^T I(z_{i,t} = j, x_{i,t} = l) \quad (17.92)$$

This result is quite intuitive: we simply add up the number of times we are in state j and we see a symbol l , and divide by the number of times we are in state j . Similarly, if each state has a Gaussian distribution associated with it, we have (from Section 4.2.4) the following MLEs:

$$\hat{\mu}_k = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T I(z_{i,t} = k) x_{i,t} \quad (17.93)$$

where the sufficient statistics are given by

$$\sum_{i=1}^N \sum_{t=1}^T I(z_{i,t} = k) x_{i,t} \quad (17.94)$$

$$\sum_{i=1}^N \sum_{t=1}^T I(z_{i,t} = k) x_{i,t} \quad (17.95)$$

Analogous results can be derived for other kinds of

distributions. One can also easily extend all of these results to compute MAP estimates, or even full posteriors over the parameters.

17.5.2 EM for HMMs (the Baum-Welch algorithm)

If the z_t variables are not observed, we are in a situation analogous to fitting a mixture model. The most common approach is to use the EM algorithm to find the MLE or MAP parameters, although of course one could use other gradient-based methods (see e.g., (Baldi and Chauvin 1994)). In this Section, we derive the EM algorithm. When applied to HMMs, this is also known as the Baum-Welch algorithm (Baum et al. 1970).

17.5.2.1 E step

It is straightforward to show that the expected complete data log likelihood is given by

$$Q(\theta, \theta_{\text{old}}) = \sum_{k=1}^K \sum_{i=1}^{N_1} \sum_{t=1}^{T_i} \log \pi_k + \sum_{j=1}^K \sum_{k=1}^K \sum_{i=1}^{N_j} \sum_{t=1}^{T_i} \log A_{jk} \quad (17.96)$$

$$+ \sum_{i=1}^{N_1} \sum_{t=1}^{T_i} \sum_{k=1}^K p(z_t = k | x_i, \theta_{\text{old}}) \log p(x_i, t | \phi_k) \quad (17.97)$$

where the expected counts are given by

$$\sum_{k=1}^K p(z_i = k | x_i, \theta_{\text{old}}) \quad (17.98)$$

$$E[N_{jk}] = \sum_{i=1}^{N_1} \sum_{t=1}^{T_i} p(z_i, t = j, z_t = k | x_i, \theta_{\text{old}}) \quad (17.99)$$

$$E[N_j] = \sum_{i=1}^{N_1} \sum_{t=1}^{T_i} p(z_i, t = j | x_i, \theta_{\text{old}}) \quad (17.100)$$

These expected sufficient statistics can be computed by running the forwards-backwards algorithm on each sequence. In particular, this algorithm computes the following smoothed node and edge marginals:

$$\gamma_{i,t}(j) = p(z_t = j | x_i, 1:T_i, \theta) \quad (17.101)$$

$$\xi_{i,t}(j, k) = p(z_{t-1} = j, z_t = k | x_i, 1:T_i, \theta) \quad (17.102)$$

17.5.2.2 M step

Based on Section 11.3, we have that the M step for A and π is to just normalize the expected counts:

$$\hat{\pi}_k = E[N_{jk}] / \sum_{k=1}^K E[N_{jk}], \quad \hat{\pi}_k = E[N_1] / K \quad (17.103)$$

This result is quite intuitive: we simply add up the expected number of transitions from j to k , and divide by the expected number of times we transition from j to anything else. For a multinoulli observation model, the expected sufficient statistics are

$$E[M_{jl}] = \sum_{i=1}^{N_1} \sum_{t=1}^{T_i} \gamma_{i,t}(j) I(x_i, t = l) = \sum_{i=1}^{N_1} \sum_{t=1}^{T_i} \gamma_{i,t}(j) \quad (17.104)$$

The M step has the form

$$\hat{A}_{jl} = E[M_{jl}] / E[N_j] \quad (17.105)$$

This result is quite intuitive: we simply add up the expected number of times we are in state j and we see a symbol l , and divide by the expected number of times we are in state j . For a Gaussian observation model, the expected sufficient statistics are given by

$$E[x_k] = \sum_{i=1}^{N_1} \sum_{t=1}^{T_i} \gamma_{i,t}(k) x_i, t \quad (17.106)$$

$$E[(xx)^T] = \sum_{i=1}^{N_1} \sum_{t=1}^{T_i} \gamma_{i,t}(k) x_i, t x_i, t^T \quad (17.107)$$

The M step becomes

$$\hat{\mu^k} = E[x_k | N_k], \hat{\Sigma^k} = E[(xx^T) | N_k] - \hat{\mu^k} \hat{\mu^k}^T / E[N_k] \quad (17.108)$$

This can (and should) be regularized in the same way we regularize GMMs.

17.5.2.3 Initialization

As usual with EM, we must take care to ensure that we initialize the parameters carefully, to minimize the chance of getting stuck in poor local optima. There are several ways to do this, such as

- Use some fully labeled data to initialize the parameters.
- Initially ignore the Markov dependencies, and estimate the observation parameters using the standard mixture model estimation methods, such as K-means or EM.
- Randomly initialize the parameters, use multiple restarts, and pick the best solution.

Techniques such as deterministic annealing (Ueda and Nakano 1998; Rao and Rose 2001) can help mitigate the effect of local minima. Also, just as K-means is often used to initialize EM for GMMs, so it is common to initialize EM for HMMs using Viterbi training, which means approximating the posterior over paths with the single most probable path. (This is not necessarily a good idea, since initially the parameters are often poorly estimated, so the Viterbi path will be fairly arbitrary. A safer option is to start training using forwards-backwards, and to switch to Viterbi near convergence.)

17.5.3 Bayesian methods for “fitting” HMMs

EM returns a MAP estimate of the parameters. In this section, we briefly discuss some methods for Bayesian parameter estimation in HMMs. (These methods rely on material that we will cover later in the book.) One approach is to use variational Bayes EM (VBEM), which we discuss in general terms in Section 21.6. The details for the HMM case can be found in (MacKay 1997; Beal 2003), but the basic idea is this: The E step uses forwards-backwards, but where (roughly speaking) we plug in the posterior mean parameters instead of the MAP estimates. The M step updates the parameters of the conjugate posteriors, instead of updating the parameters themselves. An alternative to VBEM is to use MCMC. A particularly appealing algorithm is block Gibbs sampling, which we discuss in general terms in Section 24.2.8. The details for the HMM case can be found in (Fruhwirth-Schnatter 2007), but the basic idea is this: we sample $z_{1:T}$ given the data and parameters using forwards-filtering, backwards-sampling, and we then sample the parameters from their posteriors, conditional on the sampled latent paths. This is simple to implement, but one does need to take care of unidentifiability (label switching), just as with mixture

models (see Section 11.3.1).

17.5.4 Discriminative training

Sometimes HMMs are used as the class conditional density inside a generative classifier. In this case, $p(x|y = c, \theta)$ can be computed using the forwards algorithm. We can easily maximize the joint likelihood $\prod_{i=1}^N p(y_i|x_i|\theta)$ by using EM (or some other method) to fit the HMM for each class-conditional density separately. However, we might like to find the parameters that maximize the conditional likelihood

$$\prod_{i=1}^N p(y_i|x_i, \theta) = \prod_{i=1}^N p(y_i|\theta)p(x_i|y_i, \theta) \propto p(y_i = c|\theta)p(x_i|c, \theta) \quad (17.109)$$

This is more expensive than maximizing the joint likelihood, since the denominator couples all C class-conditional HMMs together. Furthermore, EM can no longer be used, and one must resort to generic gradient based methods. Nevertheless, discriminative training can result in improved accuracies. The standard practice in speech recognition is to initially train the generative models separately using EM, and then to fine tune them discriminatively (Jelinek 1997).

17.5.5 Model selection

In HMMs, the two main model selection issues are: how many states, and what topology to use for the state transition diagram. We discuss both of these issues below.

17.5.5.1 Choosing the number of hidden states

Choosing the number of hidden states K in an HMM is analogous to the problem of choosing the number of mixture components. Here are some possible solutions:

- Use grid-search over a range of K 's, using as an objective function cross-validated likelihood, the BIC score, or a variational lower bound to the log-marginal likelihood.
- Use reversible jump MCMC. See (Fruhwirth-Schnatter 2007) for details. Note that this is very slow and is not widely used.
- Use variational Bayes to “extinguish” unwanted components, by analogy to the GMM case discussed in Section 21.6.1.6. See (MacKay 1997; Beal 2003) for details.
- Use an “infinite HMM”, which is based on the hierarchical Dirichlet process. See e.g., (Beal et al. 2002; Teh et al. 2006) for details.

17.5.5.2 Structure learning The term structure learning in the context of HMMs refers to learning a sparse transition matrix. That is, we want to learn the structure of the state transition diagram, not the structure of the graphical model (which is fixed). A large number of heuristic methods have been proposed. Most alternate between parameter estimation and some kind of heuristic split

merge method (see e.g., (Stolcke and Omohundro 1992)).

Alternatively, one can pose the problem as MAP estimation using a minimum entropy prior, of the form $p(A_{i,:}) \propto \exp(-H(A_{i,:}))$ (17.110)

This prior prefers states whose outgoing distribution is nearly deterministic, and hence has low entropy (Brand 1999). The corresponding M step cannot be solved in closed form, but numerical methods can be used. The trouble with this is that we might prune out all incoming transitions to a state, creating isolated "islands" in state-space. The infinite HMM presents an interesting alternative to these methods. See e.g., (Beal et al. 2002; Teh et al. 2006) for details.

翻译和解释

17.5 Hidden Markov Models (HMM) 中的学习

这节讨论的是如何估计HMM (隐马尔可夫模型) 的参数 $\theta = (\pi, A, B)$, 其中:

- $\pi(i) = p(z_1 = i)$: 表示初始状态的概率分布。
- $A(i,j) = p(z_t = j | z_{t-1} = i)$: 状态转移矩阵, 表示从状态*i*到状态*j*的转移概率。
- B : 表示观察模型的参数, 即给定状态 $z_t = j$, 观察到 x_t 的条件概率 $p(x_t | z_t = j)$ 。

我们首先讨论的是在已知完全观测数据的情况下训练模型, 之后讨论在隐藏数据的情况下如何估计参数。

17.5.1 完全观察数据下的训练

当我们能观察到隐藏的状态序列时, 可以直接计算最大似然估计 (MLE) 来估计 A 和 π 。

具体来说, MLE 计算方法可以参考第17.2.2.1节的内容。

对于观测模型的估计 (即B的估计), 则依赖于观测模型的形式。

例如, 如果每个状态都有一个与之关联的多项分布, 那么参数 $B_{jl} = p(X_t = l | Z_t = j)$ 的 MLE就可以通过以下公式计算:

$$\hat{B}_{jl} = \frac{\sum_{i=1}^N \sum_{t=1}^{T_i} I(z_{i,t} = j, x_{i,t} = l)}{\sum_{i=1}^N T_i I(z_{i,t} = j)}$$

其中 $I(\cdot)$ 是指示函数, 表示是否满足条件, $\sum_{i,t}$ 表示所有样本和时间步的求和。

如果状态与高斯分布相关联, 则MLE的公式为:

$$\hat{\mu}_k = \frac{x_k}{N_k}, \quad \hat{\Sigma}_k = \frac{(xx)_k^T}{N_k} - \hat{\mu}_k \hat{\mu}_k^T$$

其中 x_k 是样本在状态 k 下的均值, Σ_k 是协方差矩阵, N_k 是在状态 k 下的样本数。

类似的方法也可以应用于其他分布, 且这些方法同样适用于计算最大后验估计 (MAP) 或 全后验。

17.5.2 EM算法 (Baum-Welch算法)

如果我们无法直接观察到状态 z_t , 那么就像混合模型一样, 需要使用 EM (期望最大化) 算法 来估计参数, 这里称为 Baum-Welch算法。

以下是这个算法的步骤:

17.5.2.1 E步 (期望步)

在E步, 我们计算期望的完全数据对数似然:

$$Q(\theta, \theta_{\text{old}}) = \sum_{k=1}^K E[N_1^k] \log \pi_k + \sum_{j=1}^K \sum_{k=1}^K E[N_{jk}] \log A_{jk} + \sum_{i=1}^N \sum_{t=1}^{T_i} \sum_{k=1}^K p(z_{i,t} = k | x_i, \theta_{\text{old}}) \log p(x_{i,t} | \phi_k)$$

其中, 期望统计量是基于后向算法计算出来的:

- $E[N_1^k]$: 计算 $z_1 = k$ 的期望。
- $E[N_{jk}]$: 计算从状态 j 到状态 k 的期望转移次数。
- $p(z_{i,t} = k | x_i, \theta_{\text{old}})$: 通过前向-后向算法得到的平滑节点概率。

17.5.2.2 M步 (最大化步)

在M步, 我们通过归一化期望计数来估计转移矩阵 A 和初始状态分布 π :

$$\hat{A}_{jk} = \frac{E[N_{jk}]}{\sum_k E[N_{jk}]}, \quad \hat{\pi}_k = \frac{E[N_1^k]}{N}$$

类似地, 针对多项分布或高斯分布的观测模型, 期望统计量分别是:

- **多项分布**: 计算每个状态下观测到符号 l 的期望次数。
- **高斯分布**: 计算每个状态的均值和协方差矩阵。

17.5.2.3 初始化

初始化是EM算法的关键, 确保我们不会陷入局部最优解。常见的初始化方法包括:

- 使用完全标注的数据初始化参数。
- 忽略马尔可夫依赖关系, 先用标准的混合模型方法 (如K均值或EM) 初始化。
- 随机初始化参数, 使用多次重启, 选择最佳结果。

一些技巧 (如确定性退火) 可以帮助减缓局部最小值的影响。Viterbi训练也常被用来初始化HMM。

17.5.3 HMM的贝叶斯方法

EM算法给出的是最大后验估计 (MAP), 但有时我们想要进行贝叶斯推断, 获得参数的全后验分布。

贝叶斯方法的一种常用方法是 变分贝叶斯EM (VBEM), 它用变分方法来估计参数的后验分布。此外, 还可以使用 马尔可夫链蒙特卡洛 (MCMC) 方法, 比如 块Gibbs采样, 通过前向滤波和后向采样生成隐状态序列, 然后从参数的后验中进行采样。

17.5.4 判别训练

HMM 也可以作为生成式分类器的一部分。在这种情况下，我们需要最大化**条件似然**:

$$\sum_{i=1}^N p(y_i|x_i, \theta) = \sum_c p(y_i=c|\theta)p(x_i|y_i=c, \theta)$$

判别训练方法可能会提高分类精度，但它比生成训练更加复杂，因为需要将多个HMM进行联合训练。

通常，语音识别中的做法是先使用EM方法独立训练生成模型，然后用判别方法（例如条件随机场）进一步调整模型。

17.5.5 模型选择

在HMM中，主要的模型选择问题是：

1. 选择多少个隐状态 (K) 。
2. 如何确定状态转移图的结构。

17.5.5.1 选择隐状态的数量

选择隐状态的数量方法类似于混合模型的选择：

- 使用网格搜索 (Grid Search) 方法，基于交叉验证或BIC (贝叶斯信息准则) 等选择最佳的 K 。
- 使用可逆跳MCMC (Reversible Jump MCMC) 来选择状态数。
- 使用变分贝叶斯方法“消除”不需要的状态。
- 使用“无限HMM”模型，这可以通过层次Dirichlet过程来实现。

17.5.5.2 结构学习

结构学习指的是学习稀疏的状态转移矩阵，即学习哪些状态之间有转移。常见方法包括：

- 使用启发式方法来拆分或合并状态。
- 使用最小熵先验来学习转移矩阵的结构。

有些方法可以用**无限HMM**来解决这些问题，从而避免手动设计转移矩阵。

总结要点：

1. HMM的训练：

- 当**状态完全可见时**，参数的估计方法很简单，可以使用最大似然估计 (MLE) 来估计状态转移矩阵 A 和初始分布 π ，并根据观察数据估计观察概率模型 B 。
- 如果**状态不可见** (常见的)，则使用**EM算法** (Baum-Welch) 来估计参数。

2. EM算法：

- **E步**：计算每个可能路径的期望。
- **M步**：根据期望统计量更新参数。

3. **贝叶斯方法**: EM可以返回MAP估计, 变分贝叶斯 (VBEM) 和MCMC等方法可以用来计算完整的后验分布。
4. **判别训练**: 有时希望最大化条件似然, 通过判别训练来提高模型的精度。
5. **模型选择**: 通过网格搜索、变分贝叶斯等方法选择隐状态数或状态转移结构。

这一节主要在讲什么? 我不太懂

已思考 8s

给你一口气抓住 17.5 节的主线——“**怎么把 HMM 学出来**”。它讲的就是：已知观测序列 $x_{1:T}$, 如何估计 HMM 的三类参数 $\theta = (\pi, A, B)$ (初始分布、转移矩阵、发射/观测模型)。

一句话

17.5 = 监督训练 (状态可见) + EM/baum-welch (状态不可见) + 贝叶斯替代方案 + 判别式微调 + 模型选择 (状态数&结构)。

讲了哪几块?

1. 状态已知 (“完全观测”) 怎么学?

像数频率一样简单：

- $\hat{\pi}_i$ = “第一个时刻在状态 i 的次数 / 序列数”。
- $\hat{A}_{ij} = "i \rightarrow j" \text{ 转移次数} / \text{从 } i \text{ 出发的总转移次数}。$
- \hat{B} : 看观测模型。
 - **离散**: $\hat{B}_{jl} = \text{"在状态 } j \text{ 看到符号 } l \text{ 的次数} / \text{在 } j \text{ 的总次数}”。$
 - **高斯**: 每个状态各自算均值、协方差 (在该状态下的样本的样本均值/协方差)。

2. 状态未知 (常见) 怎么学? ——EM / Baum-Welch

- **E 步**: 跑前向-后向, 得到

- $\gamma_t(j) = p(z_t = j | x)$ (时刻 t 在 j 的后验)
 - $\xi_t(i,j) = p(z_{t-1} = i, z_t = j | x)$ ($i \rightarrow j$ 的后验)
- 这就是“期望计数”。

- **M 步**: 把“期望计数”当作真计数去归一化更新

$\hat{A}_{ij}, \hat{\pi}$, 以及按模型更新 \hat{B} (离散/高斯同上但用加权和)。

- **初始化要小心**: 可用少量标注数据、K-means/混合模型先估 B 、多次随机重启; Viterbi 训练可作粗初始化, 但别过早完全依赖它。

3. 贝叶斯学法 (不止点估计)

- **VBEM**: E 步照前向-后向, 但用**后验均值参数**; M 步更新先验的**参数** (而非点估计)。

- MCMC (块 Gibbs) : 先前向滤波+后向采样抽一条 $z_{1:T}$, 再给定这条路径从共轭后验采参数; 注意**标签置换**问题。

4. 判别式训练 (更关心分类准确率)

- 目标从最大化联合 $p(x, y)$ 换成最大化**条件似然** $p(y | x)$ 。
- 需要把所有类别的 HMM 一起优化 (分母耦合), 一般用梯度法; 工程上常**先用 EM 生成式预训练, 再判别式微调** (语音识别的常规做法)。

5. 模型选择

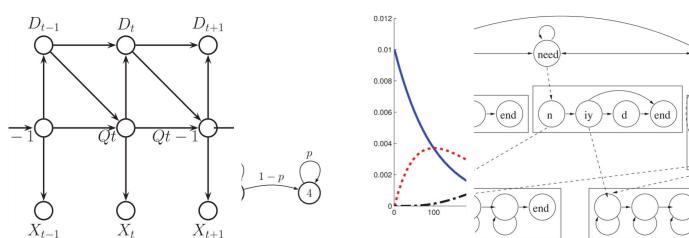
- 选状态数 K**: 交叉验证/BIC/变分下界格点搜索; 可逆跳 MCMC (慢); VB“自动关闭多余组件”; 或用**无限 HMM (HDP-HMM)**。
- 选转移结构 (稀疏拓扑)**: 启发式分裂-合并; 或在 M 步加入“**低熵先验**”鼓励每个状态的出链接近确定性; 更根本的替代是 HDP-HMM。

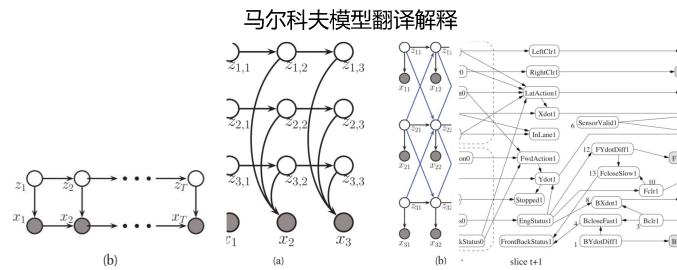
通俗图景

- 状态可见时: 就是“**数一数再除一除**”。
- 状态不可见时: **先猜软标签 (E 步) → 像有标签那样重估参数 (M 步)**, 循环到收敛。
- 想要**不止一个答案而是不确定性**: 上 VB/MCMC。
- 想要**更准的分类**: 在生成式学好后**判别式微调**。
- 不确定**有多少状态/怎么连**: 做**模型选择/结构学习**, 或用能自动扩展的**无限 HMM**。

记住 5 句就够

- HMM 要学 (π, A, B) 。
- 状态可见 \rightarrow 直接数频率 (离散) / 算均值协方差 (高斯)。
- 状态不可见 \rightarrow Baum-Welch = **前向后向(E) + 归一化计数(M)**。
- 想要后验不确定性 \rightarrow VB / MCMC (FFBS)。
- 工程上: **先生成式 EM, 后判别式微调; 用交叉验证/变分/无限 HMM 选 K 和结构**。





翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

Figure 17.14 Encoding a hidden semi-Markov model as a DGM. Dt are deterministic duration counters.

Figure 17.15 (a) A Markov chain with n = 4 repeated states and self loops. (b) The resulting distribution over sequence lengths, for p = 0.99 and various n.

Figure 17.16 An example of an HHMM for an ASR system which can recognize 3 words. The top level represents bigram word probabilities. The middle level represents the phonetic spelling of each word. The bottom level represents the subphones of each phone. (It is traditional to represent a phone as a 3 state HMM, representing the beginning, middle and end.)

Figure 17.17 An HHMM represented as a DGM. Q t is the state at time t, level l; F t |l=1 if the HMM at level l has finished (entered its exit state), otherwise F t |l=0. Shaded nodes are observed; the remaining nodes are hidden. We may optionally clamp F T |l=1, where T is the length of the observation sequence, to ensure all models have finished by the end of the sequence.

Figure 17.18 (a) Input-output HMM. (b) First-order auto-regressive HMM. (c) A second-order buried Markov model. Depending on the value of the hidden variables, the effective graph structure between the components of the observed variables (i.e., the non-zero elements of the regression matrix and the precision matrix) can change, although this is not shown.

Figure 17.19 (a) A factorial HMM with 3 chains. (b) A coupled HMM with 3 chains.

Figure 17.20 The BATnet DBN. The transient nodes are only shown for the second slice, to minimize clutter. The dotted lines can be ignored.

17.6 Generalizations of HMMs

Many variants of the basic HMM model have been proposed. We briefly discuss some of them below.

17.6.1 Variable duration (semi-Markov) HMMs

In a standard HMM, the probability we remain in state i for exactly d steps is

$$p(t_i = d) = (1 - A_{ii})^d A_{ii} \propto \exp(d \log A_{ii}) \quad (17.111)$$

where A_{ii} is the self-loop probability. This is called the

geometric distribution. However, this kind of exponentially decaying function of d is sometimes unrealistic. To allow for more general durations, one can use a semi-Markov model. It is called semiMarkov because to predict the next state, it is not sufficient to condition on the past state: we also need to know how long we've been in that state. When the state space is not observed directly, the result is called a hidden semi-Markov model (HSMM), a variable duration HMM, or an explicit duration HMM. HSMMs are widely used in many gene finding programs, since the length distribution of exons and introns is not geometric (see e.g., (Schweikert et al. 2009)), and in some chip-Seq data analysis programs (see e.g., (Kuan et al. 2009)). HSMMs are useful not only because they can model the waiting time of each state more accurately, but also because they can model the distribution of a whole batch of observations at once, instead of assuming all observations are conditionally iid. That is, they can use likelihood models of the form $p(x_{t:t+l}|z_t = k, dt = l)$, which generate l correlated observations if the duration in state k is for l time steps. This is useful for modeling data that is piecewise linear, or shows other local trends (Ostendorf et al. 1996).

17.6.1.1 HSMM as augmented HMMs

One way to represent a HSMM is to use the graphical model shown in Figure 17.14. (In this figure, we have assumed the observations are iid within each state, but this is not required, as mentioned above.) The $D_t \in \{0, 1, \dots, D\}$ node is a state duration counter, where D is the maximum duration of any state. When we first enter state j , we sample D_t from the duration distribution for that state, $D_t \sim p_j(\cdot)$. Thereafter, D_t deterministically counts down until $D_t = 0$. While $D_t > 0$, the state z_t is not allowed to change. When $D_t = 0$, we make a stochastic transition to a new state. More precisely, we define the CPDs as follows:

$$p(D_t = d | D_{t-1} = d, z_t = j) = \begin{cases} 1 & \text{if } d = 0 \\ p_j(d-1) & \text{if } d > 0 \\ 0 & \text{otherwise} \end{cases} \quad (17.112)$$

$$p(z_t = k | z_{t-1} = j, D_{t-1} = d) = \begin{cases} 1 & \text{if } d > 0 \text{ and } j = k \\ A_{jk} & \text{if } d = 0 \\ 0 & \text{otherwise} \end{cases} \quad (17.113)$$

Note that $p_j(d)$ could be represented as a table (a non-parametric approach) or as some kind of parametric distribution, such as a Gamma distribution. If $p_j(d)$ is a geometric distribution, this emulates a standard HMM. One can perform inference in this model by defining a mega-variable $Y_t = (D_t, z_t)$. However, this is rather inefficient, since D_t is deterministic. It is possible to marginalize D_t out, and derive special purpose inference procedures. See (Guedon 2003; Yu and Kobayashi 2006)

for details. Unfortunately, all these methods take $O(TK2D)$ time, where T is the sequence length, K is the number of states, and D is the maximum duration of any state.

17.6.1.2 Approximations to semi-Markov models

A more efficient, but less flexible, way to model non-geometric waiting times is to replace each state with n new states, each with the same emission probabilities as the original state. For example, consider the model in Figure 17.15(a). Obviously the smallest sequence this can generate is of length $n = 4$. Any path of length d through the model has probability $p^d n(1 - p)^{n-d}$; multiplying by the number of possible paths we find that the total probability of a path of length d is

$$p(d) = \frac{d}{n} \binom{n}{d} p^d (1-p)^{n-d} \quad (17.114)$$

This is equivalent to the negative binomial distribution. By adjusting n and the self-loop probabilities p of each state, we can model a wide range of waiting times: see Figure 17.15(b). Let E be the number of expansions of each state needed to approximate $p_j(d)$. Forwards/backwards on this model takes $O(T(KE)Fin)$ time, where Fin is the average number of predecessor states, compared to $O(TK(Fin+D))$ for the HSMM. For typical speech recognition applications, $Fin \sim 3$, $D \sim 50$, $K \sim 106$, $T \sim 105$. (Similar figures apply to problems such as gene finding, which also often uses HSMMs.) Since $Fin + D \leq Fin$, the expanded state method is much faster than an HSMM. See (Johnson 2005) for details.

17.6.2 Hierarchical HMMs

A hierarchical HMM (HHMM) (Fine et al. 1998) is an extension of the HMM that is designed to model domains with hierarchical structure. Figure 17.16 gives an example of an HHMM used in automatic speech recognition. The phone and subphone models can be “called” from different higher level contexts. We can always “flatten” an HHMM to a regular HMM, but a factored representation is often easier to interpret, and allows for more efficient inference and model fitting. HHMMs have been used in many application domains, e.g., speech recognition (Bilmes 2001), gene finding (Hu et al. 2000), plan recognition (Bui et al. 2002), monitoring transportation patterns (Liao et al. 2007), indoor robot localization (Theocharous et al. 2004), etc. HHMMs are less expressive than stochastic context free grammars (SCFGs), since they only allow hierarchies of bounded depth, but they support more efficient inference. In particular, inference in SCFGs (using the inside outside algorithm, (Jurafsky and Martin 2008)) takes $O(T^3)$ whereas inference in an HHMM takes $O(T)$ time (Murphy and Paskin 2001). We can represent an HHMM as a directed graphical model as shown in Figure

17.17. Q_t represents the state at time t and level l. A state transition at level l is only "allowed" if the chain at the level below has "finished", as determined by the F_{t-1} node. (The chain below finishes when it chooses to enter its end state.) This mechanism ensures that higher level chains evolve more slowly than lower level chains, i.e., lower levels are nested within higher levels. A variable duration HMM can be thought of as a special case of an HHMM, where the top level is a deterministic counter, and the bottom level is a regular HMM, which can only change states once the counter has "timed out". See (Murphy and Paskin 2001) for further details.

17.6.3 Input-output HMMs

It is straightforward to extend an HMM to handle inputs, as shown in Figure 17.18(a). This defines a conditional density model for sequences of the form

$$p(y_{1:T}, z_{1:T}|u_{1:T}, \theta) \quad (17.115)$$

where u_t is the input at time t; this is sometimes called a control signal. If the inputs and outputs are continuous, a typical parameterization would be

$$p(z_t|x_t, z_{t-1} = i, \theta) = \text{Cat}(z_t|S(W_{i:t})) \quad (17.116)$$

$$p(y_t|x_t, z_t = j, \theta) = N(y_t|V_{j:t}, \Sigma_j) \quad (17.117)$$

Thus the transition matrix is a logistic regression model whose parameters depend on the previous state. The observation model is a Gaussian whose parameters depend on the current state. The whole model can be thought of as a hidden version of a maximum entropy Markov model (Section 19.6.1). Conditional on the inputs u_{1:T} and the parameters θ , one can apply the standard forwards/backwards algorithm to estimate the hidden states. It is also straightforward to derive an EM algorithm to estimate the parameters (see (Bengio and Frasconi 1996) for details).

17.6.4 Auto-regressive and buried HMMs

The standard HMM assumes the observations are conditionally independent given the hidden state. In practice this is often not the case. However, it is straightforward to have direct arcs from x_{t-1} to x_t as well as from z_t to x_t, as in Figure 17.18(b). This is known as an auto-regressive HMM, or a regime switching Markov model. For continuous data, the observation model becomes

$$p(x_t|x_{t-1}, z_t = j, \theta) = N(x_t|W_{j:t-1} + \mu_j, \Sigma_j) \quad (17.118)$$

This is a linear regression model, where the parameters are chosen according to the current hidden state. We can also consider higher-order extensions, where we condition on the last L observations:

$$p(x_t|x_{t-L:t-1}, z_t = j, \theta) = N(x_t|L=1 W_j, x_{t-L:t-1} + \mu_j, \Sigma_j) \quad (17.119)$$

Such models are widely used in econometrics (Hamilton 1990). Similar models can be defined for discrete observations. The AR-HMM essentially combines two Markov chains, one on the hidden variables, to capture long range dependencies, and one on the observed variables, to capture short range dependencies (Berchtold 1999). Since the X nodes are observed, the connections between them only change the computation of the local evidence; inference can still be performed using the standard forwards-backwards algorithm. Parameter estimation using EM is also straightforward: the E step is unchanged, as is the M step for the transition matrix. If we assume scalar observations for notational simplicity, the M step involves minimizing

$$\sum_{t=1}^T \sigma_2^2(y_t - \hat{y}_t)^2 + \log \sigma_2^2 (17.120)$$

Focussing on the w terms, we see that this requires solving K weighted least squares problems:

$$\sum_{j=1}^K w_j (y_t - \hat{y}_t)^2 = \sum_{j=1}^K w_j (y_t - p(z_t=k|x_1:T))^2 (17.121)$$

where $y_t(j) = p(z_t = k|x_1:T)$ is the smoothed posterior marginal. This is a weighted linear regression problem, where the design matrix has a Toeplitz form. This subproblem can be solved efficiently using the Levinson-Durbin method (Durbin and Koopman 2001).

Buried Markov models generalize AR-HMMs by allowing the dependency structure between the observable nodes to change based on the hidden state, as in Figure 17.18(c). Such a model is called a dynamic Bayesian multi net, since it is a mixture of different networks. In the linear-Gaussian setting, we can change the structure of the arcs $x_{t-1} \rightarrow x_t$ by using sparse regression matrices, W_j , and we can change the structure of the connections within the components of x_t by using sparse Gaussian graphical models, either directed or undirected. See (Bilmes 2000) for details.

17.6.5 Factorial HMM

An HMM represents the hidden state using a single discrete random variable $z_t \in \{1, \dots, K\}$. To represent 10 bits of information would require $K = 2^{10} = 1024$ states. By contrast, consider a distributed representation of the hidden state, where each $z_{c,t} \in \{0, 1\}$ represents the c 'th bit of the t 'th hidden state. Now we can represent 10 bits using just 10 binary variables, as illustrated in Figure 17.19(a). This model is called a factorial HMM

(Ghahramani and Jordan 1997). The hope is that this kind of model could capture different aspects of a signal, e.g., one chain would represent speaking style, another the words that are being spoken. Unfortunately, conditioned on x_t , all the hidden variables are correlated (due to explaining away the common observed child x_t). This

make exact state estimation intractable. However, we can derive efficient approximate inference algorithms, as we discuss in Section 21.4.1.

17.6.6 Coupled HMM and the influence model

If we have multiple related data streams, we can use a

coupled HMM (Brand 1996), as illustrated in Figure

17.19(b). This is a series of HMMs where the state

transitions depend on the states of neighboring chains.

That is, we represent the joint conditional distribution as

$$p(z_t|z_{t-1}) = c p(z_{ct}|z_{t-1}) \quad (17.122)$$

$$p(z_{ct}|z_{t-1}) = p(z_{ct}|z_{c,t-1}, z_{c-1,t-1}, z_{c+1,t-1}) \quad (17.123)$$

This has been used for various tasks, such as audio-visual

speech recognition (Nefian et al. 2002) and modeling

freeway traffic flows (Kwon and Murphy 2000). The

trouble with the above model is that it requires $O(CK^4)$

parameters to specify, if there are C chains with K states

per chain, because each state depends on its own past

plus the past of its two neighbors. There is a closely

related model, known as the influence model

(Asavathiratham 2000), which uses fewer parameters. It

models the joint conditional distribution as

$$p(z_{ct}|z_{t-1}) = C c_{\alpha,c} = 1 \alpha_c c_{\alpha,c} p(z_{ct}|z_{c,t-1}) \quad (17.124)$$

where $c_{\alpha,c}, c_{\alpha,c} = 1$ for each c . That is, we use a convex

combination of pairwise transition matrices. The α_c parameter specifies how much influence chain c has on

chain c . This model only takes $O(C^2 + CK^2)$ parameters

to specify. Furthermore, it allows each chain to be

influenced by all the other chains, not just its nearest

neighbors. (Hence the corresponding graphical model is

similar to Figure 17.19(b), except that each node has

incoming edges from all the previous nodes.) This has

been used for various tasks, such as modeling

conversational interactions between people (Basu et al.

2001). Unfortunately, inference in both of these models

takes $O(T(KC)^2)$ time, since all the chains become fully

correlated even if the interaction graph is sparse. Various

approximate inference methods can be applied, as we

discuss later.

17.6.7 Dynamic Bayesian networks (DBNs)

A dynamic Bayesian network is just a way to represent a

stochastic process using a directed graphical model.⁷

Note that the network is not dynamic (the structure and

parameters are fixed), rather it is a network representation

of a dynamical system. All of the HMM variants we have

seen above could be considered to be DBNs. However, we

prefer to reserve the term “DBN” for graph structures that

are more “irregular” and problem-specific. An example is

shown in Figure 17.20, which is a DBN designed to

monitor the state of a simulated autonomous car known

as the “Bayesian Automated Taxi”, or “BATmobile” (Forbes et al. 1995). Defining DBNs is straightforward: you just need to specify the structure of the first time-slice, the structure between two time-slices, and the form of the CPDs. Learning is also easy. The main problem is that exact inference can be computationally expensive, because all the hidden variables become correlated over time (this is known as entanglement — see e.g., (Koller and Friedman 2009, Sec. 15.2.4) for details). Thus a sparse graph does not necessarily result in tractable exact inference. However, later we will see algorithms that can exploit the graph structure for efficient approximate inference.

下面把这一段全部逐句翻译，再配上通俗解释，最后给出要点小结。我会尽量不落下任何信息（公式和复杂度也会保留）。

图注翻译

- 图 17.14：把一个**隐半马尔可夫模型 (HSMM)** 编码成**有向图模型 (DGM)**。 D_t 是**确定性的持续时间计数器**。
- 图 17.15：(a) 一个把同一状态复制 $n = 4$ 次并各自带自环的马尔可夫链。(b) 当 $p = 0.99$ 且 n 取不同值时，所得的**序列长度分布**。
- 图 17.16：一个用于自动语音识别 (ASR) 的**分层 HMM (HHMM)**。最上层是**词级 bigram 概率**；中间层是每个词的**音素拼写**；最底层是**子音素**（传统上一个音素用 3 状态 HMM 表示：开头/中间/结尾）。
- 图 17.17：把 HHMM 画成 DGM。 Q_t^ℓ 表示**时刻 t 、层级 ℓ** 的状态；若层 ℓ 的 HMM 已经完成（进入退出状态），则 $F_t^\ell = 1$ ，否则为 0。灰色节点为观测，其余为隐藏。可以选择把 $F_T^\ell = 1$ 固定住 (T 是序列长度)，保证到序列结束所有层都“完成”。
- 图 17.18：(a) **输入-输出 HMM**。(b) **一阶自回归 HMM (AR-HMM)**。(c) **二阶 buried Markov 模型**。根据隐藏变量的取值，观测变量分量之间的**有效图结构**（回归矩阵与精度矩阵的非零位置）会改变，虽然图中没画出这种变化。
- 图 17.19：(a) **因子化 HMM (factorial HMM)**，包含 3 条链。(b) **一个耦合 HMM**，包含 3 条链。
- 图 17.20：BATnet 动态贝叶斯网络 (DBN)。为避免图太乱，只在第二个时间片画出了“瞬时”节点；虚线可忽略。

正文翻译 + 解释

17.6 HMM 的推广

已经有人提出了很多 HMM 的变体。下面简要介绍几类。

17.6.1 可变持续时间（半马尔可夫）HMMs

翻译

在标准 HMM 里，处于状态 i 连续停留恰好 d 步的概率为

$$p(t_i = d) = (1 - A_{ii})A_{ii}^d \propto \exp(d \log A_{ii}) \quad (17.111)$$

其中 A_{ii} 是自环概率。这叫做**几何分布**。然而这种随 d 指数衰减的形状有时并不现实。为了允许更一般的持续时间，可以使用**半马尔可夫模型**。之所以叫“半”，是因为预测下一状态时，除了要看上一步的状态，还需要知道当前状态已经待了多久。当状态不可见时，得到的模型就叫 HSMM / 可变时长 HMM / 显式时长 HMM。HSMM 被广泛用于许多基因识别程序（因为外显子/内含子的长度分布不是几何分布），以及一些 ChIP-Seq 数据分析程序。HSMM 有两个好处：一是更准确地描述每个状态的**等待时间**；二是可以一次性为一个**片段**的观测建模，而不必假设观测在该状态内**条件独立同分布**，即可用

$$p(x_{t:t+\ell} | z_t = k, d_t = \ell)$$

这样在状态 k 持续 ℓ 步时生成**相关的** ℓ 个观测。这对近似分段线性或具有局部趋势的数据很有用。

解释

普通 HMM 的“停留多久”由自环概率自动决定，只能是几何分布（越久越不可能）。现实里比如说话一个音素会持续几十个帧、基因片段长度有峰值，不是简单指数衰减，于是引入 HSMM：进入某状态时先抽一个**持续时间**，然后在这段时间内**不允许跳走**，到点再转移。

17.6.1.1 把 HSMM 表示成扩展 HMM

翻译

一种表示 HSMM 的方式如图 17.14。图里假设在某状态内观测是 iid（也可以放宽）。节点 $D_t \in \{0, 1, \dots, D\}$ 是**持续时间计数器**， D 是任意状态的最大持续时间。刚进入状态 j 时，从该状态的**持续时间分布** $p_j(\cdot)$ 抽样 D_t 。此后 D_t 按**确定规则递减**直到 0。只要 $D_t > 0$ ，状态 z_t 就**不能改变**；当 $D_t = 0$ 时，才按随机方式转移到新状态。更精确地，CPD 定义为

$$p(D_t = d' | D_{t-1} = d, z_t = j) = \begin{cases} p_j(d') & d' = 0 \\ 1 & d' = d-1 \text{ 且 } d \geq 1 \\ 0 & \text{其它} \end{cases} \quad (17.112)$$

$$p(z_t = k | z_{t-1} = j, D_{t-1} = d) = \begin{cases} 1 & d > 0 \text{ 且 } j = k \\ A_{jk} & d = 0 \\ 0 & \text{其它} \end{cases} \quad (17.113)$$

注意 $p_j(d)$ 可以是**表格**（非参数）或**参数分布**（如 Gamma）。如果 $p_j(d)$ 取几何分布，就退化回普通 HMM。可把 $Y_t = (D_t, z_t)$ 当成一个“**合并变量**”来做推断，但这样效率低，因为 D_t 是确定性的。可以把 D_t **积分掉**并推导专用推断算法；这些方法时间复杂度都是 $O(TK^2D)$ 。

解释

这段给出**精确的生成规则**：进入状态时抽一个“**倒计时**”，倒着数到 0 才能跳。直接把 (D, z) 拼成大状态做前向–后向虽可行，但太慢；更好的方法是**专门优化**，但复杂度仍随**最大持续时间** D 线性增长。

17.6.1.2 近似 HSMM 的方法**翻译**

更高效但较不灵活的做法：把每个状态**复制 n 份**，各份**发射参数相同**（图 17.15a）。显然生成的最短序列长度是 n 。长度为 d 的任何路径的概率是 $p^{d-n}(1-p)^n$ ；乘上长度为 d 的路径数，总概率为

$$p(d) = \binom{d-1}{n-1} p^{d-n} (1-p)^n \quad (17.114)$$

这就是**负二项分布**。通过调整 n 与自环概率 p 就能拟合各种等待时间（见图 17.15b）。若用每个状态的 E 次复制去逼近 $p_j(d)$ ，前向–后向的复杂度是 $O(T(KE)F_{\text{in}})$ ，其中 F_{in} 是平均前驱数；与 HSMM 的 $O(TK(F_{\text{in}} + D))$ 比起来，通常 $F_{\text{in}} + D \gg EF_{\text{in}}$ ，因此**复制状态法更快**。比如语音识别里常见： $F_{\text{in}} \approx 3, D \approx 50, K \approx 10^6, T \approx 10^5$ 。基因识别等问题也类似。详见 Johnson (2005)。

解释

不直接建“倒计时”，而是把一个状态拆成**链式多个微状态**，强制至少走 n 步才离开，于是**时长分布**从几何变成**负二项**，形状可调，且计算量大幅降低。

17.6.2 分层 HMM (HHMM)**翻译**

HHMM 用来刻画**层级结构**。图 17.16 给了 ASR 的例子：词层 → 音素层 → 子音素层。HHMM 可以被“**展开**”成普通 HMM，但分解形式更直观，也利于高效推断和训练。它已用于语音识别、基因发现、计划识别、交通模式监测、室内机器人定位等。HHMM 的表达力弱于**随机上下文无关文法 (SCFG)**（HHMM 的层深有上限），但推断更快：SCFG 的 inside–outside 算法是 $O(T^3)$ ，而 HHMM 是 $O(T)$ (Murphy & Paskin, 2001)。

图 17.17 展示了 DGM 表示：只有当下层链通过 $F_t^{\ell-1}$ **宣告完成**后，上层 ℓ 才允许状态转移。这样**高层演化更慢**、低层嵌套在高层里。**可变时长 HMM**可视为 HHMM 的特例：顶层是**确定性计时器**，底层是普通 HMM，只有计时到点才换状态。

解释

把“句子→词→音素→子音素”的多层规律直接放进模型里，**上层推进一次，下层可能跑多步**。既保存结构，又能线性时间推断。

17.6.3 输入–输出 HMM (IO-HMM)

翻译

可把 HMM 扩展以接收输入 u_t (图 17.18a)， 定义条件模型

$$p(y_{1:T}, z_{1:T} \mid u_{1:T}, \theta) \quad (17.115)$$

典型参数化 (连续型) :

$$p(z_t \mid u_t, z_{t-1} = i, \theta) = \text{Cat}(z_t \mid S(W_i u_t)) \quad (17.116)$$

$$p(y_t \mid u_t, z_t = j, \theta) = \mathcal{N}(y_t \mid V_j u_t, \Sigma_j) \quad (17.117)$$

即：转移概率是对 u_t 的**逻辑回归** (系数依赖前一状态)，观测是对 u_t 的**线性高斯** (参数依赖当前状态)。给定 $u_{1:T}$ 与 θ ，可直接跑前向-后向；也能推导 EM 训练 (Bengio & Frasconi, 1996)。

解释

这就是把“控制信号/外部特征”喂给 HMM，**状态转移和发射都看输入**，类似“隐式的最大熵马尔可夫模型”。

17.6.4 自回归 HMM 与 Buried Markov**翻译**

标准 HMM 假设在给定状态下**观测条件独立**。实际常不成立。可以加上 $x_{t-1} \rightarrow x_t$ 的边 (图 17.18b)，得到 AR-HMM：

$$p(x_t \mid x_{t-1}, z_t = j, \theta) = \mathcal{N}(x_t \mid W_j x_{t-1} + \mu_j, \Sigma_j) \quad (17.118)$$

或更高阶：

$$p(x_t \mid x_{t-L:t-1}, z_t = j, \theta) = \mathcal{N}(x_t \mid \sum_{\ell=1}^L W_{j,\ell} x_{t-\ell} + \mu_j, \Sigma_j) \quad (17.119)$$

这类模型广用于计量经济 (Hamilton, 1990)。本质上 AR-HMM **两条链**：隐藏链抓长程依赖，观测链抓短程依赖。因 X 可见，推断仍用标准前向-后向；EM 也简单：E 步不变；M 步里转移矩阵不变，观测参数变成带权的**线性回归**：

$$J(w_{1:K}) = \sum_j \sum_t \frac{\gamma_t(j)}{\sigma^2(j)} (y_t - y_{t-L:t-1}^\top w_j)^2 \quad (17.121)$$

设计矩阵是Toeplitz型，可用 Levinson–Durbin 高效求解。

Buried Markov 进一步允许**观测分量之间的依赖结构**随隐藏状态改变 (图 17.18c)，称为**动态贝叶斯多网**。线性-高斯情形下，可让回归矩阵 W_j 稀疏，或让 x_t 内部的高斯图 (有向/无向) 稀疏，不同状态对应不同结构 (Bilmes, 2000)。

解释

AR-HMM 就是“**状态决定哪套 AR 模型**”；Buried Markov 则更进一步，“**连变量之间怎么连也由状态决定**”。

17.6.5 因子化 HMM (Factorial HMM)

翻译

普通 HMM 用一个 K 值的离散变量 z_t 表示隐藏状态。要表示 10 比特信息需 $K = 2^{10} = 1024$ 个状态。若改为**分布式表示**: 用多条二元链 $z_{c,t} \in \{0, 1\}$ 表示各个比特 (图 17.19a) , 10 比特只需**10 个二元变量**。希望不同链能捕获信号的不同方面 (如风格/内容) 。但给定 x_t 后, 所有隐藏变量因**共同子节点的解释竞争**而相关, 精确推断**不可行**; 可用高效的**近似推断** (见 21.4.1) 。

解释

把“一个大离散状态”拆成“多条小二值链并行”, 更紧凑, 但推断要近似。

17.6.6 耦合 HMM 与 Influence Model

翻译

多路相关数据可用**耦合 HMM** (图 17.19b) 。它由多条 HMM 组成, 但每条链在 t 的状态**依赖**邻链在 $t - 1$ 的状态:

$$p(z_t | z_{t-1}) = \prod_c p(z_t^{(c)} | z_{t-1}) \quad (17.122)$$

$$p(z_t^{(c)} | z_{t-1}) = p(z_t^{(c)} | z_{t-1}^{(c)}, z_{t-1}^{(c-1)}, z_{t-1}^{(c+1)}) \quad (17.123)$$

用于视听联合识别、高速路交通流等。问题: 若有 C 条链、每条 K 状态, 参数量达 $O(CK^4)$ 。

影响模型 (influence model) 用更少参数:

$$p(z_t^{(c)} | z_{t-1}) = \sum_{c'} \alpha_{c,c'} p(z_t^{(c)} | z_{t-1}^{(c')}), \quad \sum_{c'} \alpha_{c,c'} = 1 \quad (17.124)$$

即用**成对转移矩阵的凸组合**; $\alpha_{c,c'}$ 表示 c' 对 c 的影响强度。参数量仅 $O(C^2 + CK^2)$, 且允许**全连接影响**。应用于对话建模等。

但两者推断时间都是 $O(T(KC)^2)$, 因为即使图稀疏, 链间**仍会互相关联**; 需用近似推断。

解释

多序列相互影响: 耦合 HMM 很“重”, 影响模型用**加权加和**简化影响, 既省参数又更灵活, 但推断仍不便宜。

17.6.7 动态贝叶斯网络 (DBNs)

翻译

DBN 是用有向图模型表示一个随机过程的方式 (结构/参数固定, 系统本身是“动态”的) 。上面那些 HMM 变体都可看作 DBN, 但我们通常把“结构更不规则、为具体问题定制”的模型称为 DBN。例子见图 17.20 (监控一辆模拟自动出租车 BATmobile 的状态) 。定义 DBN

很直接：给出单片结构、片间结构和各 CPD 形式即可；学习也容易。主要问题是精确推断可能很贵，因为所有隐藏变量会随时间变得相关（称为“纠缠”）。因此图稀疏也不一定可精确高效推断；后面会讲可利用图结构的近似算法。

解释

DBN 就是“时序版的贝叶斯网”。灵活，但精确推断常很难，需要近似。

小结 (抓重点)

1. HSMM (显式时长)

- 普通 HMM 时长是**几何分布**；不够现实时用 HSMM：进入状态就抽**持续时间**，计时到 0 才能跳。
- 精确推断 $O(TK^2D)$ ；更快的**复制状态近似**把时长做成**负二项分布**，计算量更低。

2. HHMM (分层)

- 多层状态嵌套：高层推进慢、低层在里面跑。**线性时间推断**，比 SCFG 快。

3. IO-HMM

- 转移/发射看输入 u_t ；可用前向-后向 + EM 训练。

4. AR-HMM / Buried Markov

- 允许 $x_{t-1} \rightarrow x_t$ （或更高阶），每个状态一套回归；训练变成**带权线性回归**。
- Buried Markov：**观测之间的连接结构**也随状态改变。

5. Factorial HMM

- 多条二值链构成“分布式隐藏状态”；更紧凑，但**精确推断不可行**，需近似。

6. Coupled HMM / Influence model

- 多序列相互影响；影响模型用**凸组合**降参数量到 $O(C^2 + CK^2)$ ，但推断仍是 $O(T(KC)^2)$ 。

7. DBN

- 通用的时序有向图框架；定义简单，但**精确推断易爆炸**，通常依赖近似法。

如果你想，我可以用一个小玩具例子（比如 2 个状态 + 负二项时长）把 HSMM 的前向-后向如何改动，给你算一遍具体数字。

这一节主要在讲什么？我不太懂

已思考 11s

这整节 (17.6) 在干一件事：**把“基础 HMM”升级扩展**，让它能应对更真实的时序数据。可以把它当成“工具箱导览”。核心扩展有 7 类：

1. 可变持续时间：HSMM (半马尔可夫)

- 解决：普通 HMM 的“停留时长=几何分布（指数衰减）”太死板。

- 做法 A：显式加一个**倒计时变量** D_t ；进入状态先抽时长，计到 0 才能跳（图 17.14），推断约 $O(TK^2D)$ 。
- 做法 B（快的近似）：把一个状态复制成 n 个**串联小状态**，得到**负二项**的时长分布（图 17.15），计算远快于 HSMM。
- 场景：基因片段长度、语音片段持续时间、分段线性信号等。

2. 分层 HMM：HHMM

- 解决：数据有**层级结构/不同时间尺度**（词→音素→子音素）。
- 思路：上层慢、下层快，只有下层“完成”上层才前进一步（图 17.16–17.17）。
- 亮点：可线性时间 $O(T)$ 推断，比 SCFG 的 $O(T^3)$ 高效。

3. 输入–输出 HMM (IO-HMM)

- 解决：状态转移/发射受**外部输入**影响（控制信号、上下文特征）。
- 做法：转移用**逻辑回归**看 u_t ，发射用**线性高斯**看 u_t （图 17.18a）；仍可前向–后向与 EM。

4. 自回归 HMM (AR-HMM) 与 Buried Markov

- 解决：观测值之间存在**短期相关**。
- 做法：加 $x_{t-1} \rightarrow x_t$ （或更高阶），每个隐藏状态一套 AR 模型（图 17.18b）；M 步变成**带权线性回归**。
- 升级：Buried Markov 让**观测间的连接结构**也随状态变化（图 17.18c）。

5. 因子化 HMM (Factorial HMM)

- 解决：单一大状态不够表达；用多条**二值链分布式表示**（图 17.19a）。
- 代价：给定观测后链间强相关，**精确推断不可行**，需用近似法。

6. 耦合 HMM 与 Influence Model

- 解决：多路序列相互影响（多传感器、多参与者对话等）。
- 耦合 HMM：每条链受邻链影响（图 17.19b），参数多。
- Influence model：用**凸组合**降参， α 表示谁影响谁；仍然推断昂贵（时间约 $O(T(KC)^2)$ ）。

7. 动态贝叶斯网络 (DBN)

- 统一框架：用有向图表达各种时序模型（图 17.20）。
- 易定义易训练，但**精确推断**常因“纠缠”变贵，所以常用近似推断。

什么时候用哪种？

- 时长不像指数衰减 → HSMM / 状态复制近似
- 任务有层级、不同节奏 → HHMM
- 有外部驱动/控制量 → IO-HMM
- 存在强短期自相关 → AR-HMM / Buried
- 隐含多个独立因素 → Factorial HMM
- 多条相关序列 → 耦合 HMM / Influence

- 结构很定制 → DBN

一句话总结：17.6 把 HMM 这把“锤子”打造成**多功能电动工具**：可控时长、分层、带输入、会自回归、能并行/相互影响，还能用 DBN 自定义你要的时序结构。

基于本对话里我贴过的整章原文，输出：
①知识点总结
(分点+关键词+用途) ②子章节关联 (依赖/对比/递进+关系句，附Mermaid可选) ③本章总结。

① 知识点总结 (分点 + 关键词 + 用途)

• 马尔可夫链 (MC)

- **定义：**一阶马氏性， $p(X_{1:T}) = p(X_1) \prod_{t=2}^T p(X_t | X_{t-1})$ 。
- **关键词：**转移矩阵 A ， n 步转移 $A^{(n)} = A^n$ ，Chapman–Kolmogorov，齐次/平稳，离散有限状态。
- **用途：**序列建模基础、语言模型 (n-gram)、随机游走、PageRank、MCMC 的状态转移核。

• 转移矩阵与状态图

- **关键词：**行随机矩阵、状态转移图 (弧上是概率)、左到右矩阵。
- **用途：**直观刻画可达性/稀疏性，指导算法复杂度 (稀疏 \Rightarrow 线性时间)。

• 平稳分布 (Stationary / Limiting)

- **定义：** $\pi = \pi A$ ；若不可约 (singly connected) 且非周期且 (在无限状态时) 遍历/非零返期 \Rightarrow 极限分布存在且唯一。
- **关键词：**不可约、周期/非周期、吸收态、常返/瞬态、正则链。
- **用途：**长程行为分析、PageRank 的理论基础、MCMC 收敛目标。

• 应用：语言模型 (LM)

- **关键词：**unigram/bigram/trigram (n-gram)、极大似然=计数归一化、加一/回退/插值平滑 (λ)、经验贝叶斯 (Dirichlet 先验)、OOV (UNK/DP)。
- **用途：**补全、压缩、分类、文本生成。

• 应用：PageRank

- **关键词：**随机游走、阻尼因子 p 、转移矩阵 $M = pGD + \mathbf{1}\mathbf{z}^\top$ 、幂法求主特征向量。
- **用途：**网页/图节点重要性排序、图挖掘。

• 隐马尔可夫模型 (HMM)

- **定义：**隐藏状态 z_t 的马尔可夫链 + 发射模型 $p(x_t | z_t)$ (离散矩阵 B 或高斯 $N(\mu, \Sigma)$)。
- **关键词：**联合 $p(z_{1:T}, x_{1:T})$ ，马尔可夫切换。
- **用途：**语音识别、活动识别、词性标注、基因/蛋白序列分析 (profile-HMM) 等。

• 时序推断任务

- **关键词:** 过滤 $p(z_t|x_{1:t})$ 、平滑 $p(z_t|x_{1:T})$ 、固定时滞、预测 $p(z_{t+h}|x_{1:t})$ 、MAP 路径 (Viterbi)、后验采样、证据概率 $p(x_{1:T})$ 。
- **用途:** 在线/离线识别、预报、异常检测、分类评分。

• 前向算法 (Filtering)

- **关键词:** 预测-更新、 $\alpha_t \propto \psi_t \odot (\Psi^\top \alpha_{t-1})$ 、增量对数似然 $\sum \log Z_t$ 。
- **用途:** 在线状态信念更新、求 $p(x_{1:T})$ 。

• 前向-后向 (Smoothing)

- **关键词:** $\alpha, \beta, \gamma: \gamma_t \propto \alpha_t \odot \beta_t$; 两切片边缘 $\zeta_{t,t+1}$; 复杂度 $O(K^2 T)$ (稀疏/左到右可降至 $O(KT)$)。
- **用途:** 离线最优边缘、EM 的期望计数、计算充分统计量。

• Viterbi (MAP 路径)

- **关键词:** $\delta_t(j) = \max_i \delta_{t-1}(i)\psi(i,j)\phi_t(j) + \text{回溯表 } a_t(j)$; MAP vs MPM (联合最优 vs 各时刻边缘最优)。
- **用途:** 得到一条一致的最可能解释 (语音、分词等)，快速 E 步近似。

• 参数学习

- **完全观测:** 频数归一化 (A, π) + 按模型估计 B (离散计数/高斯均值协方差)。
- **EM / Baum-Welch:** E 步 γ, ζ ; M 步归一化更新 A, π, B ; 初始化策略、退火/Viterbi 训练。
- **贝叶斯:** VBEM (共轭后验参数)、块 Gibbs (FFBS 采样路径 + 条件共轭采参)、标签置换。
- **判别式训练:** 最大化 $p(y|x)$, 需梯度法; 工程上“先生成式 EM, 后判别式微调”。
- **模型选择:** 状态数 K (CV/BIC/变分下界、RJMCMC、VB 熄灭、HDP-HMM); 转移结构稀疏 (启发式分裂合并、低熵先验、HDP-HMM)。

• HMM 的常见推广 (Generalizations)

- **HSMM:** 显式持续时间 D_t ; 或状态复制≈负二项时长 (快)。
- **HHMM:** 层级多时间尺度, 线性时间推断。
- **IO-HMM:** 有输入 u_t , 转移/发射依赖输入 (逻辑回归/线性高斯)。
- **AR-HMM / Buried:** 观测自回归/观测图结构随状态变化; M 步化为 (加权) 线性回归。
- **Factorial HMM:** 多条二元链的分布式隐藏表示 (需近似推断)。
- **Coupled HMM / Influence:** 多序列相互影响; 影响模型用凸组合降参; 推断 $O(T(KC)^2)$ 。
- **DBN:** 时序有向图的通用外壳; 精确推断可能昂贵 (纠缠), 常用近似。

② 子章节关联 (依赖 / 对比 / 递进)

- 17.1 → 17.2 (引入 → 基础) : 先定义“序列概率模型”的需求，再用马尔可夫链给出最简结构（依赖：一阶马氏性）。
- 17.2 → 17.2.3 (基础 → 长期行为) : 从一步转移走向平稳分布与收敛条件（不可约/非周期/遍历），用于理解长期极限（递进）。
- 17.2.3 → 17.2.4 (理论 → 应用) : 把平稳分布等价为PageRank 的特征向量问题（依赖）。
- 17.2 → 17.3 (显式状态 → 隐状态) : 用隐藏变量 z_t 放大表达力，保留链结构（对比：MC 直接建 x , HMM 通过 z 间接建 x ）。
- 17.3 → 17.4 (模型 → 推断) : 在已知参数下求 z 的边缘/MAP (依赖：链图上的动态规划)。
- 17.4 → 17.5 (推断 → 学习) : EM 的 E 步用前向-后向产生期望计数，M 步归一化（强依赖）。
- 17.5 → 17.6 (学习 → 扩展) : 在训练与推断框架就绪后，扩展到时长、层级、输入、自回归、多链、DBN 等（递进/泛化）。

Mermaid (章节关系图)

```

flowchart LR
    A[17.1 引言] --> B[17.2 马尔可夫链]
    B --> C[17.2.3 平稳分布]
    C --> D[17.2.4 PageRank]
    B --> E[17.3 HMM 定义]
    E --> F[17.4 HMM 推断  
Filtering/FB/Viterbi]
    F --> G[17.5 学习  
MLE/EM/VB/MCMC/选择]
    G --> H[17.6 扩展家族  
HSMM/HHMM/IO/AR/Factorial/Coupled/DBN]
  
```

关键对比句

- Filtering vs Smoothing vs Viterbi: 前两者给边缘概率（点wise 最优，平滑更稳）；Viterbi 给全局一致的最可能路径（序列级最优）。
- MC vs HMM: MC 依赖最近观测；HMM 通过隐状态携带长期信息、发射建模更灵活。
- HSMM vs 状态复制近似: 前者精确时长分布但 $O(TK^2D)$ ；后者负二项近似更快。
- 耦合 HMM vs Influence: 耦合直接条件依赖（参数多）；Influence 用凸组合压参（灵活但仍难推断）。

③ 本章总结

- **目标**: 建立与求解序列概率模型。从马尔可夫链起步，解决“如何表示/预测序列”，再引入隐状态形成 HMM 以提升表达力。
- **两大主线**
 - 推断**: 前向（在线过滤）、前向-后向（离线平滑与期望计数）、Viterbi (MAP 路径)、后验采样；并明确各自的适用场景与优缺点。
 - 学习**: 完全可观测的频数法；隐藏状态下的 EM/ Baum-Welch；再到贝叶斯 (VB/MCMC) 与判别式训练；配合模型选择（状态数与结构）。
- **理论与应用**: 平稳分布与不可约/非周期性为PageRank和MCMC提供数学保证；n-gram 与平滑技术连接了语言建模与贝叶斯先验。

- **工程要点：**利用稀疏/左到右结构降复杂度；初始化与多重重启防局部最优；平滑的边缘阈值常比 Viterbi 更低错率，但 Viterbi 适合需一致路径的任务。
- **扩展生态：**通过 HSMM/HHMM/IO/AR/Factorial/Coupled/DBN，HMM 成为一套可拼装的序列建模平台，覆盖可变时长、层级多尺度、外部输入、自相关、多通道交互以及通用的 DBN 表达。

一句话：第 17 章把“序列=局部转移×（可选）隐状态”的思想讲透，从**建模**→**推断**→**学习**→**应用**→**扩展**形成闭环，是经典时序概率建模的“整机说明书”。

请用通俗易懂的话总结本章，使我能理解

已思考 10s

好的，把整章用“人话”说清楚 

这章想解决什么？

要给“序列数据”建模。

像天气、语音、文字、基因、网页点击……它们都有“今天影响明天”的规律。我们需要两样东西：

1. 怎么表达这种规律（模型）；
2. 怎么根据数据推断/学习（算法）。

一、从马尔可夫链开始：只看上一步就够了

- **核心假设：**下一步只和上一步有关（不追溯更久的历史）。
- **表示法：**用一个转移矩阵 A 说“从 i 到 j 的概率”。画成图就是**状态转移图**。
- **能做什么：**
 - 语言里：只看前一个（或两个）词来猜下一个 → **n-gram 模型**。遇到没见过的搭配，用**平滑/回退**别把概率给成 0。
 - 长期会待在哪些状态？解 $\pi = \pi A$ 得到**平稳分布**。
 - PageRank** 就是把网页当状态、超链当转移，再加一点“随机跳转”，解出那个平稳分布，排名就出来了。

二、隐马尔可夫模型（HMM）：看得见结果，看不见原因

很多序列表面现象（声音、字母、传感器数据）背后有**隐藏状态**（说的话、活动类型、是否在基因区）。

- **结构：**隐藏状态自己按马氏链跳，**每个状态**按自己的规则**发射**一个观测。
- **直觉：**一段时间数据“像一类”，忽然“跳”到另一类——这就是**状态切换**。
- **常见用途：**语音识别、词性标注、活动识别、基因/蛋白序列比对（profile-HMM）。

三、我们要在 HMM 里算什么？

把“看到的数据”变成“对隐藏状态的判断”：

- **过滤（在线）**：到时间 t 为止，估计 $p(\text{状态} t \mid \text{过去观测})$ 。
算法 = **前向算法**：先预测再用当前观测更新。
 - **平滑（离线）**：拿到整段数据后，估计 $p(\text{状态} t \mid \text{全部观测})$ 。
算法 = **前向-后向**：先从左到右传消息，再从右到左回来，两个信息相乘。
 - **MAP 路径 (Viterbi)**：要一条整体最可能的状态序列。像找图上最重的路径，前向记最大概率、后向回溯。
注意：“**每个时刻最可能**”和“**整条路径最可能**”不一定一样。
 - 其他：**预测未来**、**抽样**可能的路径、**算似然**用来做分类/检测。
-

四、怎么学会模型参数？

- **状态已知**：数数归一化就行（转移矩阵、发射分布）。
 - **状态未知**：用 EM / Baum-Welch。
 - **E 步**：前向-后向得到“在 j 状态的期望次数”“从 j 到 k 的期望次数”。
 - **M 步**：用这些期望次数去更新转移/发射参数。
 - 细节：初始化要小心（可多次重启/先做聚类），必要时做**正则化/平滑**。
 - **贝叶斯版**：VBEM（更新后验的参数）或 Gibbs 抽样（先抽状态路径，再抽参数）。
 - **判别式训练**：直接最大化 $p(y|x)$ 提升准确率（工程上常“先 EM，再微调”）。
 - **模型选择**：选状态数/转移结构（交叉验证、BIC、变分下界、非参的 HDP-HMM）。
-

五、HMM 的常见“改装件”

- **HSMM (可变持续时间)**：让一个状态能待“更符合常识”的时长（非几何分布）。
 - **HHMM (分层)**：上层慢、下层快（词→音素→子音素），更贴近真实结构。
 - **IO-HMM (有输入)**：转移/发射受外部控制量影响。
 - **AR-HMM / Buried**：观测之间也相关（自回归），甚至相关结构随状态改变。
 - **Factorial HMM**：多条二值小链共同表示一个大状态（分布式编码）。
 - **Coupled / Influence**：多条序列互相影响（多传感器、多参与者）。
 - **DBN**：把上述模型放进统一的**动态贝叶斯网络**外壳里，按需要拼装。
-

六、怎么选用？

- 片段时长显著 \neq 指数衰减 \rightarrow HSMM / 状态展开近似
- 存在明显层级/多时间尺度 \rightarrow HHMM
- 有外部驱动变量 \rightarrow IO-HMM

- 观测短期相关很强 → AR-HMM / Buried
 - 想把“风格/内容/噪声”等因素拆开表示 → Factorial
 - 多路数据彼此影响 → Coupled / Influence
 - 结构很定制 → 直接画 DBN
-

一页纸的要点

- **思想**: 序列 = “局部转移规律” × (可选) “隐藏原因”。
- **三把算法锤**: 前向 (在线) 、前向-后向 (离线) 、Viterbi (最可能路径) 。
- **两类学习**: 已标注就数数; 未标注用 EM (或贝叶斯/判别式) 。
- **工程提示**: 利用稀疏/左-右结构降复杂度; 注意初始化与数值稳定; 评估时既看路径也看边缘错误率。
- **扩展**: 把持续时间、层级、输入、自回归、多链等能力拼上去, 覆盖大多数时序任务。

一句话: 这章教你用马尔可夫思想看世界——先用马尔可夫链讲“怎么走”, 再用 HMM 讲“看得见现象、推断背后原因”, 配上前向/后向/Viterbi 与 EM, 把建模-推断-学习一条龙打通, 并告诉你如何按需升级。