Decision-Making & Modelling Under Uncertainty (DMU)

Oliver Maclaren (oliver.maclaren@auckland.ac.nz)

[10 lectures/tutorials]

- Decision-making under uncertainty [5/10]
  - Basic concepts
  - Risk, probability, utility
  - Statistical: extended setup
    - formulation & empirical risk approx.
    - minimax & Bayes
  - Tutorial sheet
- Modelling under uncertainty { models of risk & intervention [5/10]
  - probability, graphical models, & independence
  - causal interpretations of graphical models
  - stochastic process models (esp. Markov)
  - simulation & estimation tools
  - Tutorial sheet

Lecture 8: Markov models in time cont'd

- invariant & limiting distributions
- expected values under these distrib.
- estimating transition matrices

Plus:

- Further topics & applications (brief notes)
  - Filtering, smoothing, prediction
  - Markov chain Monte Carlo (MCMC)
  - Markov decision processes (MDPs)
    - dynamic programming & reinforcement learning

'Special' (marginal) distributions: invariant marginal distributions

The transition matrix $\underline{P}$ updates (maps) marginals μ_n to marginals μ_{n+1}:

$$\boxed{\mu_{n+1} = \mu_n \underline{P}}$$

There are some 'special' marginal distributions that are 'unchanged' by $\underline{P}$

→ these are like (or are!) fixed points in dynamical systems

→ useful! (see later)

These solve:

$$\boxed{\pi = \pi \underline{P}} \text{ s.t. } \boxed{\textstyle\sum \pi(i) = 1}$$

(since prob. dist.)

we call these { invariant / stationary / steady state / fixed point } distributions

↳ (equivalent terms)

Eigenvalue/vector problem!

$\pi = \pi \underline{P}$ is a (left) eigenvalue/eigenvector problem for $\lambda = 1$ (since rows)

Can solve in usual way (see below)

→ is a 'single step' property

→ 'once start at dist., stay at'

Solving: For eigenvector π (eigvalue = 1)

solve $\boxed{\pi(\underline{I} - \underline{P}) = 0}$ (vector, vector)

subject to $\boxed{\textstyle\sum \pi(i) = 1}$ (since prob. dist.)

Example →

Example

- $\underline{P} = \begin{bmatrix} 0.5 & 0.5 \\ 0.25 & 0.75 \end{bmatrix}$
- $(\underline{I} - \underline{P}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0.5 & 0.5 \\ 0.25 & 0.75 \end{bmatrix}$

$$= \begin{bmatrix} 0.5 & -0.5 \\ -0.25 & 0.25 \end{bmatrix}$$

- $\pi = [\pi_1 \ \pi_2]$, $\pi_1 + \pi_2 = 1$.

$$\Rightarrow \pi(\underline{I} - \underline{P}) = [0.5\pi_1 - 0.25\pi_2, -0.5\pi_1 + 0.25\pi_2]$$

$$= [0 \ , \ 0]$$

$\Rightarrow$ (1) $0.5\pi_1 - 0.25\pi_2$
(2) $-0.5\pi_1 + 0.25\pi_2$

} not linearly indep. as expected: defined up to norm.

So have (1) $0.5\pi_1 - 0.25\pi_2 = 0$

& (2)′ $\pi_1 + \pi_2 = 1$ } normalisation

$1/2 \times$ (2)′ $-$ (1): $0.5\pi_2 + 0.25\pi_2 = 0.5$

$3/4\,\pi_2 = 1/2 \Rightarrow \pi_2 = 2/3$

$\Rightarrow \pi_1 = 1/3$

$\pi = [1/3 \ \ 2/3]$

Example

verify?

$\pi = [1/3 \ \ 2/3]$

$$\pi \underline{P} = [1/3 \ \ 2/3] \begin{bmatrix} 1/2 & 1/2 \\ 1/4 & 3/4 \end{bmatrix}$$

$$= [1/6 + 1/6 \ , \ 1/6 + 3/6]$$

$$= [1/3 \ \ 2/3]$$

$$= \pi \ \checkmark$$

Limiting distributions

If the n-step transition matrix has a limit:

$$\underline{P}^n \xrightarrow[(n\to\infty)]{} \begin{bmatrix} — & \pi & — \\ — & \pi & — \\ — & \pi & — \end{bmatrix}$$

ie the rows of $\underline{P}^n$ tend to the same vector π, we call

- $\underline{P}^n \to \underline{P}^\infty$ the **limiting** n-step transition matrix
- π the **limiting distribution**
  - also called **equilibrium** dist.

Note: $(u\underline{P}^\infty)(j) = \sum_i u(i)\underline{P}^\infty_{ij}$ $\left.\vphantom{\sum}\right\}$ $\underline{P}^\infty_{ij} = \pi(j)$ for all i (rows)

$= \sum_i u(i)\pi(j)$

$= \pi(j)\left(\sum_i u(i)\right)$

$= \pi(j)$ since $\sum_i u(i) = 1$

$\Rightarrow u\underline{P}^\infty = \pi$ regardless of u.

Limiting?

Idea of limiting dist π & $\underline{P}^\infty$

- $\underline{P}^\infty$ maps all dist to π (ie)
- eventually reach same distribution, regardless of initial distribution
- 'stay at' this distribution once get to, $\pi\underline{P}^\infty = \pi$

Limiting vs invariant?

limiting $\Rightarrow$ invariant

invariant $\not\Rightarrow$ limiting

$\to$ might not converge to / reach invariant distribution

However, in practice / this course we will just assume **invariant = limiting**

$\to$ eg find invariant via eigenvector problem & use as limiting.

Expected values

- Consider a 'value' (ie utility!) function defined on the state space of the Markov chain:

  $u(i)$ = 'value/utility of state i'

- For a random state at stage n, ie X_n, the value is also a random variable $U = u(X_n)$

→ Suppose our Markov chain 'settles down' to a unique invariant/limiting distribution π (won't distinguish here)

→ Compute expected value (utility) under the Markov chain in two ways:

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} u(x_i) \overset{?}{\to} \mathbb{E}_\pi[U(X_n)] := \sum_j u(j)\pi(j)$$

long-term time average (sample-based)

'ensemble' average ($= u\pi^T = \pi u^T$) (distribution-based)

'Ergodic' Markov chains & expectations

"An irreducible, 'ergodic' (see Wasserman) Markov chain has a unique invariant distribution π, which is also equal to the limiting distribution & ...

$$\lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} g(x_n) \to \mathbb{E}_\pi(g) := \sum_i g(i)\pi(i)$$

time average → ensemble average."

↳ key (will just assume here)

For us, key idea is we can compute with either a series of realisations over time or as an average with respect to π, depending on what info. we have

↳ sample-based = 'Monte Carlo'

(Markov chain Monte Carlo!
↳ see later)

Example

Q. Consider a Markov chain on state space

$$X = \{0, 1, 2, 3, 4\}$$

- Suppose $u(j) = 2j + j^2$ for $j \in X$

- Suppose you find invariant distrib. π is

$$\pi = \frac{1}{12}(1 \;\; 2 \;\; 3 \;\; 5 \;\; 1)$$

Calculate the expected value under this distribution.

A: Table (for convenience)

$j =$	0	1	2	3	4
$u =$	0	3	8	15	24
$\pi =$	$\frac{1}{12}$	$\frac{2}{12}$	$\frac{3}{12}$	$\frac{5}{12}$	$\frac{1}{12}$

$$\Rightarrow \mathbb{E}_\pi[u] = \pi u^T \text{ ie } (-\pi-)\begin{pmatrix} | \\ u \\ | \end{pmatrix} = \text{dot product} = u\pi^T$$

$$= 0 \times \frac{1}{12} + 3 \times \frac{2}{12} + \cdots + 24 \times \frac{1}{12}$$

$$= \underline{10.75}$$

Estimating transition matrices

- Recall that a realisation of a Markov chain is simply a sequence of state values

Define a matrix of transition counts $\underline{C}$ where $(\underline{C})_{ij}$ = ## times j follows i in sequence

Can estimate

$$(\underline{P})_{ij} \approx \frac{(\underline{C})_{ij}}{\sum_j (\underline{C})_{ij}}$$

← normalise by total counts (from i to anywhere) (row sums)

[This is the maximum likelihood estimate]

Example

Q. Consider the following sequence from a Markov chain on $\{1,2,3\}$:

$$(1,2,1,1,1,2,3,1,1,2,1,3,1,1,3,2)$$

- Estimate the transition matrix

A. $(\underline{P})_{ij} \approx \dfrac{(\underline{C})_{ij}}{\sum_j (\underline{C})_{ij}}$

&

$$\underline{C} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 4 & 3 & 2 \\ 2 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix} \end{array} \qquad \begin{array}{c} \text{row sum } \sum_j (\underline{C})_{ij} \\ \hline 9 \\ 3 \\ 3 \end{array}$$

$$\Rightarrow \underline{P} \approx \begin{bmatrix} 4/9 & 3/9 & 2/9 \\ 2/3 & 0 & 1/3 \\ 2/3 & 1/3 & 0 \end{bmatrix}$$

Further topics (brief overview)

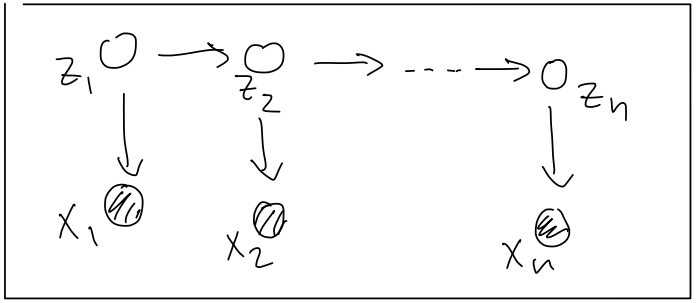
- Filtering, smoothing, prediction etc
  - time series etc
- Markov chain Monte Carlo (MCMC)
  - Bayesian statistical inference (sample posterior) etc
- Markov decision processes (MDPs)
  - Probabilistic dynamic programming
  - Reinforcement learning

Filtering, smoothing, prediction

Many discrete or continuous processes can usefully be formulated as **state-space** models, also known as **state-observation** models, **hidden Markov models** (HMMs), hierarchical process-observation models etc

---

Idea: hidden states z_n, nodes: ○
observations x_n of states, nodes: ◉



'State' evolution:

$$z_{n+1} \perp z_{\dots n-1} \mid z_n$$

Observation model:

$$x_n \perp z_{\dots, n-1} \mid z_n$$

Filtering, smoothing, prediction

Filtering:

→ | data up to x_N

→ | goal: $p(z_n \mid x_n)$ up to N

Smoothing:

→ | data up to x_N

→ | goal: $p(z_n \mid x_n)$ up to $N-L$ for lag L

Prediction:

→ | data up to x_N

→ | goal: $p(z_{N+H} \mid x_N)$ for pred. horizon H.

In principle just 'standard queries', compute in usual way

→ However, in practice can use structure to define efficient recursive computation algs.

↳ Forwards-Backwards alg. for discrete state

↳ Kalman filter for continuous state space.

etc!

Markov chain Monte Carlo (MCMC)

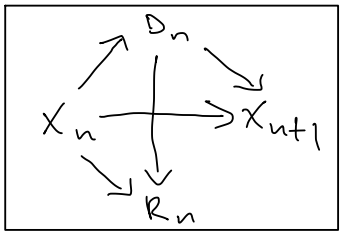
Often in eg Bayesian statistics (but also in other cases), we want samples from a distribution or expectations wrt. a distribution but we can't get a closed form soln.

→ Idea of **MCMC** is to construct an ergodic **Markov chain** with limiting distribution equal to the target (tho still not known explicitly). — Markov chain

\+

→ Time simulations will 'settle down' into draws from the distribution & eg the time averages will approximate ensemble expectations. — Monte Carlo (sample-based)

Markov decision processes (MDPs)



- **MDPs** (also POMDPs) combine sequential decision-making (eg dynamic programming) & Markov stochastic processes
  - ↳ 'influence diagrams' (see above ↑)
  - ↳ stochastic dynamic programming
- Transitions are also influenced by actions/decisions, eg consider $P(X_{n+1} | X_n, D_n)$ for decision D_n at time/stage n in state X_n.
- Get 'rewards' R_n at each stage
- Given a 'policy' (decisions for all states), reduces to Markov chain

→ Can evaluate & improve policies iteratively ('policy iteration'), among other strategies

→ MDPs form foundation for 'reinforcement learning' in machine learning
  - ↳ RL ≈ approximate dynamic programming

→ see next module on dynamic programming!

Wasserman (2004) continued

that since $a_i = 1$, the chain will return to state i again. By repeating this argument, we conclude that $\mathbb{E}(Y|X_0 = i) = \infty$. If i is transient, then $a_i < 1$. When the chain is in state i, there is a probability $1 - a_i > 0$ that it will never return to state i. Thus, the probability that the chain is in state i exactly n times is $a_i^{n-1}(1 - a_i)$. This is a geometric distribution which has finite mean. ∎

23.16 Theorem. *Facts about recurrence.*

1. *If state i is recurrent and $i \leftrightarrow j$, then j is recurrent.*
2. *If state i is transient and $i \leftrightarrow j$, then j is transient.*
3. *A finite Markov chain must have at least one recurrent state.*
4. *The states of a finite, irreducible Markov chain are all recurrent.*

23.17 Theorem (Decomposition Theorem)**.** *The state space $\mathcal{X}$ can be written as the disjoint union*

$$\mathcal{X} = \mathcal{X}_T \bigcup \mathcal{X}_1 \bigcup \mathcal{X}_2 \cdots$$

where $\mathcal{X}_T$ are the transient states and each $\mathcal{X}_i$ is a closed, irreducible set of recurrent states.

23.18 Example (Random Walk)**.** Let $\mathcal{X} = \{\ldots, -2, -1, 0, 1, 2, \ldots,\}$ and suppose that $p_{i,i+1} = p$, $p_{i,i-1} = q = 1 - p$. All states communicate, hence either all the states are recurrent or all are transient. To see which, suppose we start at $X_0 = 0$. Note that

$$p_{00}(2n) = \binom{2n}{n} p^n q^n \tag{23.11}$$

since the only way to get back to 0 is to have n heads (steps to the right) and n tails (steps to the left). We can approximate this expression using Stirling's formula which says that

$$n! \sim n^n \sqrt{n} e^{-n} \sqrt{2\pi}.$$

Inserting this approximation into (23.11) shows that

$$p_{00}(2n) \sim \frac{(4pq)^n}{\sqrt{n\pi}}.$$

It is easy to check that $\sum_n p_{00}(n) < \infty$ if and only if $\sum_n p_{00}(2n) < \infty$. Moreover, $\sum_n p_{00}(2n) = \infty$ if and only if $p = q = 1/2$. By Theorem (23.15), the chain is recurrent if $p = 1/2$ otherwise it is transient. ∎

Convergence of Markov Chains. To discuss the convergence of chains, we need a few more definitions. Suppose that $X_0 = i$. Define the **recurrence time**

$$T_{ij} = \min\{n > 0 : \ X_n = j\} \tag{23.12}$$

assuming X_n ever returns to state i, otherwise define $T_{ij} = \infty$. The **mean recurrence time** of a recurrent state i is

$$m_i = \mathbb{E}(T_{ii}) = \sum_n n f_{ii}(n) \tag{23.13}$$

where

$$f_{ij}(n) = \mathbb{P}(X_1 \neq j, X_2 \neq j, \ldots, X_{n-1} \neq j, X_n = j | X_0 = i).$$

A recurrent state is **null** if $m_i = \infty$ otherwise it is called **non-null** or **positive.**

23.19 Lemma. *If a state is null and recurrent, then $p_{ii}^n \to 0$.*

23.20 Lemma. *In a finite state Markov chain, all recurrent states are positive.*

Consider a three-state chain with transition matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

Suppose we start the chain in state 1. Then we will be in state 3 at times 3, 6, 9, ... This is an example of a periodic chain. Formally, the **period** of state i is d if $p_{ii}(n) = 0$ whenever n is not divisible by d and d is the largest integer with this property. Thus, $d = \gcd\{n : \ p_{ii}(n) > 0\}$ where gcd means "greater common divisor." State i is **periodic** if $d(i) > 1$ and **aperiodic** if $d(i) = 1$. A state with period 1 is called **aperiodic**.

23.21 Lemma. *If state i has period d and $i \leftrightarrow j$ then j has period d.*

23.22 Definition. *A state is* **ergodic** *if it is recurrent, non-null and aperiodic. A chain is ergodic if all its states are ergodic.*

Let $\pi = (\pi_i : \ i \in \mathcal{X})$ be a vector of non-negative numbers that sum to one. Thus π can be thought of as a probability mass function.

23.23 Definition. *We say that π is a* **stationary** *(or* **invariant***) distribution if $\pi = \pi \mathbf{P}$.*

Here is the intuition. Draw X_0 from distribution π and suppose that π is a stationary distribution. Now draw X_1 according to the transition probability of the chain. The distribution of X_1 is then $\mu_1 = \mu_0\mathbf{P} = \pi\mathbf{P} = \pi$. The distribution of X_2 is $\pi\mathbf{P}^2 = (\pi\mathbf{P})\mathbf{P} = \pi\mathbf{P} = \pi$. Continuing this way, we see that the distribution of X_n is $\pi\mathbf{P}^n = \pi$. In other words:

> If at any time the chain has distribution π, then it will continue to have distribution π forever.

23.24 Definition. *We say that a chain has* **limiting distribution** *if*

$$\mathbf{P}^n \to \begin{bmatrix} \pi \\ \pi \\ \vdots \\ \pi \end{bmatrix}$$

for some π, that is, $\pi_j = \lim_{n\to\infty} \mathbf{P}^n_{ij}$ exists and is independent of i.

Here is the main theorem about convergence. The theorem says that an ergodic chain converges to its stationary distribution. Also, sample averages converge to their theoretical expectations under the stationary distribution.

23.25 Theorem. *An irreducible, ergodic Markov chain has a unique stationary distribution π. The limiting distribution exists and is equal to π. If g is any bounded function, then, with probability 1,*

$$\lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} g(X_n) \to \mathbb{E}_\pi(g) \equiv \sum_j g(j)\pi_j. \tag{23.14}$$

Finally, there is another definition that will be useful later. We say that π satisfies **detailed balance** if

$$\pi_i p_{ij} = p_{ji}\pi_j. \tag{23.15}$$

Detailed balance guarantees that π is a stationary distribution.

23.26 Theorem. *If π satisfies detailed balance, then π is a stationary distribution.*

PROOF. We need to show that $\pi\mathbf{P} = \pi$. The j^{th} element of $\pi\mathbf{P}$ is $\sum_i \pi_i p_{ij} = \sum_i \pi_j p_{ji} = \pi_j \sum_i p_{ji} = \pi_j$. ∎

The importance of detailed balance will become clear when we discuss Markov chain Monte Carlo methods in Chapter 24.

Warning! Just because a chain has a stationary distribution does not mean it converges.

23.27 Example. Let

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

Let $\pi = (1/3, 1/3, 1/3)$. Then $\pi P = \pi$ so π is a stationary distribution. If the chain is started with the distribution π it will stay in that distribution. Imagine simulating many chains and checking the marginal distribution at each time n. It will always be the uniform distribution π. But this chain does not have a limit. It continues to cycle around forever. ∎

EXAMPLES OF MARKOV CHAINS.

23.28 Example. Let $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$. Let

$$\mathbf{P} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{3}{4} & 0 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

Then $C_1 = \{1, 2\}$ and $C_2 = \{5, 6\}$ are irreducible closed sets. States 3 and 4 are transient because of the path $3 \to 4 \to 6$ and once you hit state 6 you cannot return to 3 or 4. Since $p_{ii}(1) > 0$, all the states are aperiodic. In summary, 3 and 4 are transient while 1, 2, 5, and 6 are ergodic. ∎

23.29 Example (Hardy-Weinberg)**.** Here is a famous example from genetics. Suppose a gene can be type A or type a. There are three types of people (called genotypes): AA, Aa, and aa. Let (p, q, r) denote the fraction of people of each genotype. We assume that everyone contributes one of their two copies of the gene at random to their children. We also assume that mates are selected at random. The latter is not realistic however, it is often reasonable to assume that you do not choose your mate based on whether they are AA, Aa, or aa. (This would be false if the gene was for eye color and if people chose mates based on eye color.) Imagine if we pooled everyone's genes together. The proportion of A genes is $P = p + (q/2)$ and the proportion of a genes is

$Q = r + (q/2)$. A child is AA with probability P^2, aA with probability $2PQ$, and aa with probability Q^2. Thus, the fraction of A genes in this generation is

$$P^2 + PQ = \left(p + \frac{q}{2}\right)^2 + \left(p + \frac{q}{2}\right)\left(r + \frac{q}{2}\right).$$

However, $r = 1 - p - q$. Substitute this in the above equation and you get $P^2 + PQ = P$. A similar calculation shows that the fraction of "a" genes is Q. We have shown that the proportion of type A and type a is P and Q and this remains stable after the first generation. The proportion of people of type AA, Aa, aa is thus $(P^2, 2PQ, Q^2)$ from the second generation and on. This is called the Hardy-Weinberg law.

Assume everyone has exactly one child. Now consider a fixed person and let X_n be the genotype of their n^{th} descendant. This is a Markov chain with state space $\mathcal{X} = \{AA, Aa, aa\}$. Some basic calculations will show you that the transition matrix is

$$\begin{bmatrix} P & Q & 0 \\ \frac{P}{2} & \frac{P+Q}{2} & \frac{Q}{2} \\ 0 & P & Q \end{bmatrix}.$$

The stationary distribution is $\pi = (P^2, 2PQ, Q^2)$. ∎

23.30 Example (Markov chain Monte Carlo)**.** In Chapter 24 we will present a simulation method called Markov chain Monte Carlo (MCMC). Here is a brief description of the idea. Let $f(x)$ be a probability density on the real line and suppose that $f(x) = cg(x)$ where $g(x)$ is a known function and $c > 0$ is unknown. In principle, we can compute c since $\int f(x)dx = 1$ implies that $c = 1/\int g(x)dx$. However, it may not be feasible to perform this integral, nor is it necessary to know c in the following algorithm. Let X_0 be an arbitrary starting value. Given $X_0, \ldots, X_i$, draw X_{i+1} as follows. First, draw $W \sim N(X_i, b^2)$ where $b > 0$ is some fixed constant. Let

$$r = \min\left\{\frac{g(W)}{g(X_i)},\ 1\right\}.$$

Draw $U \sim \text{Uniform}(0, 1)$ and set

$$X_{i+1} = \begin{cases} W & \text{if } U < r \\ X_i & \text{if } U \geq r. \end{cases}$$

We will see in Chapter 24 that, under weak conditions, $X_0, X_1, \ldots$, is an ergodic Markov chain with stationary distribution f. Hence, we can regard the draws as a sample from f. ∎

Inference for Markov Chains. Consider a chain with finite state space $\mathcal{X} = \{1, 2, \ldots, N\}$. Suppose we observe n observations $X_1, \ldots, X_n$ from this chain. The unknown parameters of a Markov chain are the initial probabilities $\mu_0 = (\mu_0(1), \mu_0(2), \ldots,)$ and the elements of the transition matrix $\mathbf{P}$. Each row of $\mathbf{P}$ is a multinomial distribution. So we are essentially estimating N distributions (plus the initial probabilities). Let n_{ij} be the observed number of transitions from state i to state j. The likelihood function is

$$\mathcal{L}(\mu_0, \mathbf{P}) = \mu_0(x_0)\prod_{r=1}^{n} p_{X_{r-1}, X_r} = \mu_0(x_0)\prod_{i=1}^{N}\prod_{j=1}^{N} p_{ij}^{n_{ij}}.$$

There is only one observation on μ_0 so we can't estimate that. Rather, we focus on estimating $\mathbf{P}$. The MLE is obtained by maximizing $\mathcal{L}(\mu_0, \mathbf{P})$ subject to the constraint that the elements are non-negative and the rows sum to 1. The solution is

$$\widehat{p}_{ij} = \frac{n_{ij}}{n_i}$$

where $n_i = \sum_{j=1}^{N} n_{ij}$. Here we are assuming that $n_i > 0$. If not, then we set $\widehat{p}_{ij} = 0$ by convention.

23.31 Theorem (Consistency and Asymptotic Normality of the MLE)**.** *Assume that the chain is ergodic. Let $\widehat{p}_{ij}(n)$ denote the* MLE *after n observations. Then $\widehat{p}_{ij}(n) \xrightarrow{\text{P}} p_{ij}$. Also,*

$$\left[\sqrt{N_i(n)}(\widehat{p}_{ij} - p_{ij})\right] \rightsquigarrow N(0, \Sigma)$$

where the left-hand side is a matrix, $N_i(n) = \sum_{r=1}^{n} I(X_r = i)$ and

$$\Sigma_{ij,k\ell} = \begin{cases} p_{ij}(1 - p_{ij}) & (i, j) = (k, \ell) \\ -p_{ij}p_{i\ell} & i = k, j \neq \ell \\ 0 & \text{otherwise.} \end{cases}$$

23.3 Poisson Processes

The Poisson process arises when we count occurrences of events over time, for example, traffic accidents, radioactive decay, arrival of email messages, etc. As the name suggests, the Poisson process is intimately related to the Poisson distribution. Let's first review the Poisson distribution.

Recall that X has a Poisson distribution with parameter λ — written $X \sim \text{Poisson}(\lambda)$ — if

$$\mathbb{P}(X = x) \equiv p(x;\ \lambda) = \frac{e^{-\lambda}\lambda^x}{x!}, \quad x = 0, 1, 2, \ldots$$

Murphy (2012) 'Machine learning: A probabilistic approach'

Define the vector $\mathbf{z}$ with components

$$z_j = \begin{cases} \delta & \text{if } c_j \neq 0 \\ 1/n & \text{if } c_j = 0 \end{cases} \tag{17.36}$$

Then we can rewrite Equation 17.34 as follows:

$$\mathbf{M} = p\mathbf{G}\mathbf{D} + \mathbf{1}\mathbf{z}^T \tag{17.37}$$

The matrix $\mathbf{M}$ is not sparse, but it is a rank one modification of a sparse matrix. Most of the elements of $\mathbf{M}$ are equal to the small constant δ. Obviously these do not need to be stored explicitly.

Our goal is to solve $\mathbf{v} = \mathbf{M}\mathbf{v}$, where $\mathbf{v} = \boldsymbol{\pi}^T$. 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:

$$\mathbf{v} \propto \mathbf{M}\mathbf{v} = p\mathbf{G}\mathbf{D}\mathbf{v} + \mathbf{1}\mathbf{z}^T\mathbf{v} \tag{17.38}$$

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 $\mathbf{v} = \mathbf{M}\mathbf{v}$. 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 $\mathbf{v}$ 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.

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 $z_t \in \{1, \ldots, K\}$, plus an **observation** model

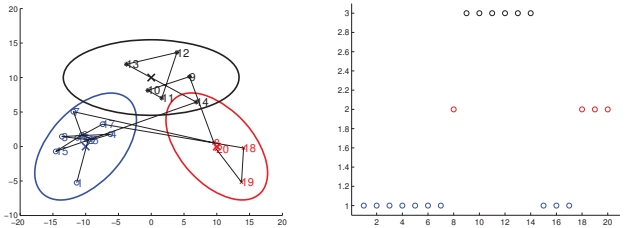


Figure 17.7 (a) Some 2d data sampled from a 3 state HMM. Each state emits from a 2d Gaussian. (b) The hidden state sequence. Based on Figure 13.8 of (Bishop 2006b). Figure generated by `hmmLillypadDemo`.

$p(\mathbf{x}_t|z_t)$. The corresponding joint distribution has the form

$$p(\mathbf{z}_{1:T}, \mathbf{x}_{1:T}) = p(\mathbf{z}_{1:T})p(\mathbf{x}_{1:T}|\mathbf{z}_{1:T}) = \left[p(z_1)\prod_{t=2}^{T} p(z_t|z_{t-1})\right]\left[\prod_{t=1}^{T} p(\mathbf{x}_t|z_t)\right] \tag{17.39}$$

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:

$$p(\mathbf{x}_t = l|z_t = k, \boldsymbol{\theta}) = B(k, l) \tag{17.40}$$

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

$$p(\mathbf{x}_t|z_t = k, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_t|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{17.41}$$

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** (Fruhwirth-Schnatter 2007).) We see that we tend to get multiple observations in the same location, and then a sudden jump to a new cluster.

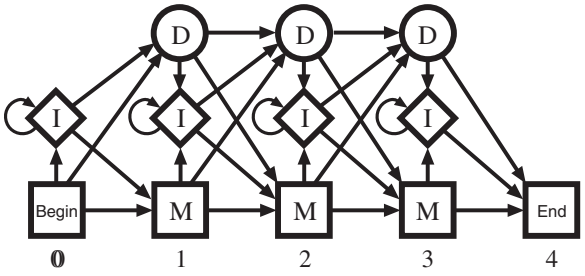
17.3.1 Applications of HMMs

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 time-series 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 $p(z_t|\mathbf{x}_{1:t})$ if we are

	x	x	.	.	.	x
bat	A	G	-	-	-	C
rat	A	-	A	G	-	C
cat	A	G	-	A	A	-
gnat	-	-	A	A	A	C
goat	A	G	-	-	-	C
	1	2	.	.	.	3

(a)



(b)

Figure 17.8 (a) Some DNA sequences. (b) State transition diagram for a profile HMM. Source: Figure 5.7 of (Durbin et al. 1998). Used with kind permission of Richard Durbin.

in an online scenario, or $p(z_t|\mathbf{x}_{1:T})$ 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 $\mathbf{x}_t$ 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 $p(\mathbf{x}_t|z_t)$ represents the acoustic model. See e.g., (Jelinek 1997; Jurafsky and Martin 2008) for details.
- **Activity recognition**. Here $\mathbf{x}_t$ 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., (Schweikerta 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.

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, $x_t \in \{1, 2, \ldots, 6\}$ 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 $\{1, \ldots, 6\}$. 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:    664153216162115234653214356634261655234232315142464156663246
Die:      LLLLLLLLLLLLLFFFFFFLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFLLLLLLL
```

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

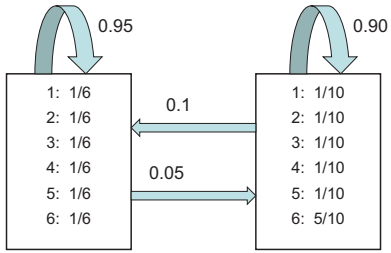
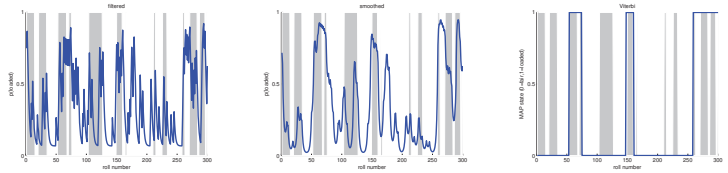


Figure 17.9 An HMM for the occasionally dishonest casino. The blue arrows visualize the state transition diagram $\mathbf{A}$. Based on (Durbin et al. 1998, p54).



(a) (b) (c)

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 generated by `casinoDemo`.

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** $p(z_t|\mathbf{x}_{1:t})$ 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, $p(z_t|\mathbf{x}_t)$. 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 $p(z_t|\mathbf{x}_{1:T})$ 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.

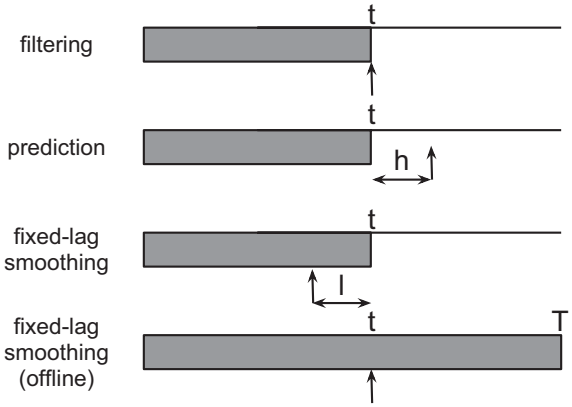


Figure 17.11 The main kinds of inference for state-space models. The shaded region is the interval for which we have data. The arrow represents the time step at which we want to perform inference. t is the current time, T is the sequence length, ℓ is the lag and h is the prediction horizon. See text for details.

- **Fixed lag smoothing** is an interesting compromise between online and offline estimation; it involves computing $p(z_{t-\ell}|\mathbf{x}_{1:t})$, where $\ell > 0$ 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 $p(z_{t+h}|\mathbf{x}_{1:t})$, where $h > 0$ is called the prediction **horizon**. For example, suppose $h = 2$; then we have

$$p(z_{t+2}|\mathbf{x}_{1:t}) = \sum_{z_{t+1}} \sum_{z_t} p(z_{t+2}|z_{t+1}) p(z_{t+1}|z_t) p(z_t|\mathbf{x}_{1:t}) \tag{17.42}$$

  It is straightforward to perform this computation: we just power up the transition matrix and apply it to the current belief state. The quantity $p(z_{t+h}|\mathbf{x}_{1:t})$ is a prediction about future hidden states; it can be converted into a prediction about future observations using

$$p(\mathbf{x}_{t+h}|\mathbf{x}_{1:t}) = \sum_{z_{t+h}} p(\mathbf{x}_{t+h}|z_{t+h}) p(z_{t+h}|\mathbf{x}_{1:t}) \tag{17.43}$$

  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 $\arg\max_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T}|\mathbf{x}_{1:T})$, 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, $\mathbf{z}_{1:T} \sim p(\mathbf{z}_{1:T}|\mathbf{x}_{1:T})$. 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**, $p(\mathbf{x}_{1:T})$, by summing up over all hidden paths, $p(\mathbf{x}_{1:T}) = \sum_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T}, \mathbf{x}_{1:T})$. 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, $p(z_t|\mathbf{x}_{1:t})$ 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:

$$p(z_t = j|\mathbf{x}_{1:t-1}) = \sum_i p(z_t = j|z_{t-1} = i)p(z_{t-1} = i|\mathbf{x}_{1:t-1}) \tag{17.44}$$

Next comes the update step, in which we absorb the observed data from time t using Bayes rule:

$$\alpha_t(j) \triangleq p(z_t = j|\mathbf{x}_{1:t}) = p(z_t = j|\mathbf{x}_t, \mathbf{x}_{1:t-1}) \tag{17.45}$$

$$= \frac{1}{Z_t} p(\mathbf{x}_t|z_t = j, \cancel{\mathbf{x}_{1:t-1}}) p(z_t = j|\mathbf{x}_{1:t-1}) \tag{17.46}$$

where the normalization constant is given by

$$Z_t \triangleq p(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = \sum_j p(z_t = j|\mathbf{x}_{1:t-1}) p(\mathbf{x}_t|z_t = j) \tag{17.47}$$

This process is known as the **predict-update cycle**. The distribution $p(z_t|\mathbf{x}_{1:t})$ is called the (filtered) **belief state** at time t, and is a vector of K numbers, often denoted by $\boldsymbol{\alpha}_t$. In matrix-vector notation, we can write the update in the following simple form:

$$\boldsymbol{\alpha}_t \propto \boldsymbol{\psi}_t \odot (\boldsymbol{\Psi}^T \boldsymbol{\alpha}_{t-1}) \tag{17.48}$$

where $\psi_t(j) = p(\mathbf{x}_t|z_t = j)$ is the local evidence at time t, $\Psi(i, j) = p(z_t = j|z_{t-1} = i)$ is the transition matrix, and $\mathbf{u} \odot \mathbf{v}$ is the **Hadamard product**, representing elementwise vector multiplication. See Algorithm 6 for the pseudo-code, and `hmmFilter` for some Matlab code.

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

$$\log p(\mathbf{x}_{1:T}|\boldsymbol{\theta}) = \sum_{t=1}^{T} \log p(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = \sum_{t=1}^{T} \log Z_t \tag{17.49}$$

(We need to work in the log domain to avoid numerical underflow.)

Algorithm 17.1: Forwards algorithm

1 Input: Transition matrices $\psi(i, j) = p(z_t = j|z_{t-1} = i)$, local evidence vectors $\psi_t(j) = p(\mathbf{x}_t|z_t = j)$, initial state distribution $\pi(j) = p(z_1 = j)$;
2 $[\alpha_1, Z_1] = \text{normalize}(\boldsymbol{\psi}_1 \odot \boldsymbol{\pi})$;
3 **for** $t = 2 : T$ **do**
4 $\quad [\alpha_t, Z_t] = \text{normalize}(\boldsymbol{\psi}_t \odot (\boldsymbol{\Psi}^T \boldsymbol{\alpha}_{t-1}))$;
5 Return $\boldsymbol{\alpha}_{1:T}$ and $\log p(\mathbf{y}_{1:T}) = \sum_t \log Z_t$;
6 Subroutine: $[\mathbf{v}, Z] = \text{normalize}(\mathbf{u}) : Z = \sum_j u_j; \quad v_j = u_j/Z$;

17.4.3 The forwards-backwards algorithm

In Section 17.4.2, we explained how to compute the filtered marginals $p(z_t = j|\mathbf{x}_{1:t})$ using online inference. We now discuss how to compute the smoothed marginals, $p(z_t = j|\mathbf{x}_{1:T})$, 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:

$$p(z_t = j|\mathbf{x}_{1:T}) \propto p(z_t = j, \mathbf{x}_{t+1:T}|\mathbf{x}_{1:t}) \propto p(z_t = j|\mathbf{x}_{1:t}) p(\mathbf{x}_{t+1:T}|z_t = j, \cancel{\mathbf{x}_{1:t}}) \tag{17.50}$$

Let $\alpha_t(j) \triangleq p(z_t = j|\mathbf{x}_{1:t})$ be the filtered belief state as before. Also, define

$$\beta_t(j) \triangleq p(\mathbf{x}_{t+1:T}|z_t = j) \tag{17.51}$$

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

$$\gamma_t(j) \triangleq p(z_t = j|\mathbf{x}_{1:T}) \tag{17.52}$$

as the desired smoothed posterior marginal. From Equation 17.50, we have

$$\gamma_t(j) \propto \alpha_t(j)\beta_t(j) \tag{17.53}$$

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:

$$\begin{aligned}
\beta_{t-1}(i) &= p(\mathbf{x}_{t:T}|z_{t-1}=i) && (17.54)\\
&= \sum_j p(z_t=j, \mathbf{x}_t, \mathbf{x}_{t+1:T}|z_{t-1}=i) && (17.55)\\
&= \sum_j p(\mathbf{x}_{t+1:T}|z_t=j, \cancel{z_{t-1}=i}, \cancel{\mathbf{x}_t}) p(z_t=j, \mathbf{x}_t|z_{t-1}=i) && (17.56)\\
&= \sum_j p(\mathbf{x}_{t+1:T}|z_t=j) p(\mathbf{x}_t|z_t=j, \cancel{z_{t-1}=i}) p(z_t=j|z_{t-1}=i) && (17.57)\\
&= \sum_j \beta_t(j)\psi_t(j)\psi(i,j) && (17.58)
\end{aligned}$$

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

$$\boldsymbol{\beta}_{t-1} = \boldsymbol{\Psi}(\boldsymbol{\psi}_t \odot \boldsymbol{\beta}_t) \quad (17.59)$$

The base case is

$$\beta_T(i) = p(\mathbf{x}_{T+1:T}|z_T=i) = p(\emptyset|z_T=i) = 1 \quad (17.60)$$

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:

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

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

$$\begin{aligned}
\xi_{t,t+1}(i,j) &\triangleq p(z_t=i, z_{t+1}=j|\mathbf{x}_{1:T}) && (17.62)\\
&\propto p(z_t|\mathbf{x}_{1:t}) p(z_{t+1}|z_t, \mathbf{x}_{t+1:T}) && (17.63)\\
&\propto p(z_t|\mathbf{x}_{1:t}) p(\mathbf{x}_{t+1:T}|z_t, z_{t+1}) p(z_{t+1}|z_t) && (17.64)\\
&\propto p(z_t|\mathbf{x}_{1:t}) p(\mathbf{x}_{t+1}|z_{t+1}) p(\mathbf{x}_{t+2:T}|z_{t+1}) p(z_{t+1}|z_t) && (17.65)\\
&= \alpha_t(i)\phi_{t+1}(j)\beta_{t+1}(j)\psi(i,j) && (17.66)
\end{aligned}$$

In matrix-vector form, we have

$$\boldsymbol{\xi}_{t,t+1} \propto \boldsymbol{\Psi} \odot (\boldsymbol{\alpha}_t (\boldsymbol{\phi}_{t+1} \odot \boldsymbol{\beta}_{t+1})^T) \quad (17.67)$$

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 $\psi(i,j) \propto \exp(-\sigma^2|\mathbf{z}_i - \mathbf{z}_j|)$, where $\mathbf{z}_i$ 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 $\sum_t \xi_{t-1,t}(i,j)$; this takes constant space (independent of T); however, to compute them, we need $O(KT)$ working space, since we must store α_t for $t = 1, \ldots, T$ 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

$$\mathbf{z}^* = \arg\max_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}) \quad (17.68)$$

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 $z_1, z_2, \ldots, z_T$ is given by

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

17.4.4.1 MAP vs 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**:

$$\hat{\mathbf{z}} = (\arg\max_{z_1} p(z_1|\mathbf{x}_{1:T}), \ldots, \arg\max_{z_T} p(z_T|\mathbf{x}_{1:T})) \quad (17.70)$$

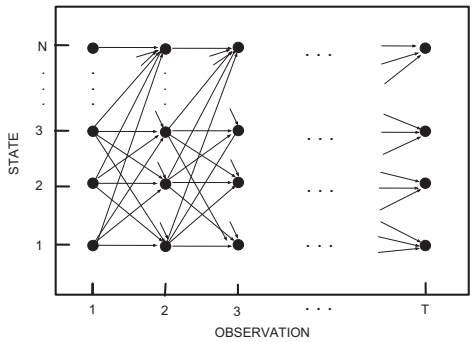


Figure 17.12 The trellis of states vs time for a Markov chain. Based on (Rabiner 1989).

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

	$X_1=0$	$X_1=1$	
$X_2=0$	0.04	0.3	0.34
$X_2=1$	0.36	0.3	0.66
	0.4	0.6	

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 is is always globally consistent. For example, suppose we are performing speech recognition and someones 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:

$$z_t^* = \arg\max_{z_t} \max_{\mathbf{z}_{1:t-1},\mathbf{z}_{t+1:T}} p(\mathbf{z}_{1:t-1}, z_t, \mathbf{z}_{t+1:T}|\mathbf{x}_{1:T}) \tag{17.71}$$

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

$$p(z_t|\mathbf{x}_{1:T}) = \sum_{\mathbf{z}_{1:t-1},\mathbf{z}_{t+1:T}} p(\mathbf{z}_{1:t-1}, z_t, \mathbf{z}_{t+1:T}|\mathbf{x}_{1:T}) \tag{17.72}$$

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.[6]

6. In general, we may want to mix max and sum. For example, consider a joint distribution where we observe v and we want to query q; let n be the remaining nuisance variables. We define the MAP estimate as $\mathbf{x}_q^* = \arg\max_{\mathbf{x}_q} \sum_{\mathbf{x}_n} p(\mathbf{x}_q, \mathbf{x}_n|\mathbf{x}_v)$, where we max over $\mathbf{x}_q$ and sum over $\mathbf{x}_n$. By contrast, we define the **MPE** or most probable explanation as $(\mathbf{x}_q^*, \mathbf{x}_n^*) = \arg\max_{\mathbf{x}_q,\mathbf{x}_n} p(\mathbf{x}_q, \mathbf{x}_n|\mathbf{x}_v)$, where we max over both $\mathbf{x}_q$ and $\mathbf{x}_n$. This terminology is due to (Pearl 1988), although it is not widely used outside the Bayes net literatire. Obviously MAP=MPE if $n = \emptyset$. However, if $n \neq \emptyset$, then summing out the nuisance variables can give different results than maxing them out. Summing out nuisance variables is more sensible, but computationally harder, because of the need to combine max and sum operations (Lerner and Parr 2001).

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 probably joint assignments. The reasons 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 max-product, 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

$$\delta_t(j) \triangleq \max_{z_1,\ldots,z_{t-1}} p(\mathbf{z}_{1:t-1}, z_t = j|\mathbf{x}_{1:t}) \tag{17.73}$$

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

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

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

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

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

$$\delta_1(j) = \pi_j \phi_1(j) \tag{17.76}$$

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

$$z_T^* = \arg\max_i \delta_T(i) \tag{17.77}$$

We can then compute the most probable sequence of states using **traceback**:

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

As usual, we have to worry about numerical underflow. We are free to normalize the $\boldsymbol{\delta}_t$ terms at each step; this will not affect the maximum. However, unlike the forwards-backwards case,

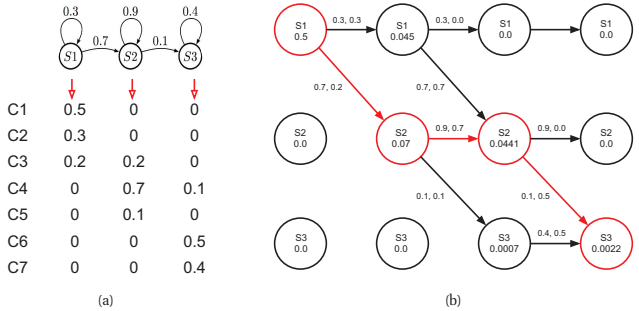


Figure 17.13 Illustration of Viterbi decoding in a simple HMM for speech recognition. (a) A 3-state HMM for a single phone. We are visualizing the state transition diagram. We assume the observations have been vector quantized into 7 possible symbols, $C_1, \ldots, C_7$. Each state z_1, z_2, z_3 has 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 $C1, C3, C4, C6$. 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 $i \to j$ transition, and the second is the probability of generating observation $\mathbf{x}_t$ from state j. The bold lines/ circles represent the most probable sequence of states. Based on Figure 24.27 of (Russell and Norvig 1995).

we can also easily work in the log domain. The key difference is that $\log \max = \max \log$, whereas $\log \sum \neq \sum \log$. Hence we can use

$$\log \delta_t(j) \triangleq \max_{\mathbf{z}_{1:t-1}} \log p(\mathbf{z}_{1:t-1}, z_t = j | \mathbf{x}_{1:t}) \quad (17.79)$$

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

In the case of Gaussian observation models, this can result in a significant (constant factor) speedup, since computing $\log p(\mathbf{x}_t|z_t)$ can be much faster than computing $p(\mathbf{x}_t|z_t)$ for a high-dimensional 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 $\mathbf{x}_{1:4} = (C_1, C_3, C_4, C_6)$, 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 proabability 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

$$\delta_2(1) = \delta_1(1)\psi(1,1)\phi_2(1) = 0.5 \cdot 0.3 \cdot 0.3 = 0.045 \quad (17.81)$$

$$\delta_2(2) = \delta_1(1)\psi(1,2)\phi_2(2) = 0.5 \cdot 0.7 \cdot 0.2 = 0.07 \quad (17.82)$$

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. One 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 $\psi(i,j) \propto \exp(-\sigma^2||\mathbf{z}_i - \mathbf{z}_j||^2)$, where $\mathbf{z}_i$ 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:

$$\mathbf{z}^s_{1:T} \sim p(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}) \quad (17.83)$$

We can do this is as follow: run forwards backwards, to compute the two-slice smoothed posteriors, $p(z_{t-1,t}|\mathbf{x}_{1:T})$; next compute the conditionals $p(z_t|z_{t-1}, \mathbf{x}_{1:T})$ by normalizing; sample from the initial pair of states, $z^*_{1,2} \sim p(z_{1,2}|\mathbf{x}_{1:T})$; finally, recursively sample $z^*_t \sim p(z_t|z^*_{t-1}, \mathbf{x}_{1:T})$.

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

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

We can then sample z_t given future sampled states using

$$z_t^s \sim p(z_t|z_{t+1:T}, \mathbf{x}_{1:T}) = p(z_t|z_{t+1}, \cancel{\mathbf{z}_{t+2:T}}, \mathbf{x}_{1:t}, \cancel{\mathbf{x}_{t+1:T}}) = p(z_t|z_{t+1}^s, \mathbf{x}_{1:t}) \tag{17.85}$$

The sampling distribution is given by

$$\begin{aligned} p(z_t = i|z_{t+1} = j, \mathbf{x}_{1:t}) &= p(z_t|z_{t+1}, \mathbf{x}_{1:t}, \cancel{\mathbf{x}_{t+1}}) && (17.86)\\ &= \frac{p(z_{t+1}, z_t|\mathbf{x}_{1:t+1})}{p(z_{t+1}|\mathbf{x}_{1:t+1})} && (17.87)\\ &\propto \frac{p(\mathbf{x}_{t+1}|z_{t+1}, \cancel{z_t}, \cancel{\mathbf{x}_{1:t}})p(z_{t+1}, z_t|\mathbf{x}_{1:t})}{p(z_{t+1}|\mathbf{x}_{1:t+1})} && (17.88)\\ &= \frac{p(\mathbf{x}_{t+1}|z_{t+1})p(z_{t+1}|z_t, \cancel{\mathbf{x}_{1:t}})p(z_t|\mathbf{x}_{1:t})}{p(z_{t+1}|\mathbf{x}_{1:t+1})} && (17.89)\\ &= \frac{\phi_{t+1}(j)\psi(i,j)\alpha_t(i)}{\alpha_{t+1}(j)} && (17.90) \end{aligned}$$

The base case is

$$z_T^s \sim p(z_T = i|\mathbf{x}_{1:T}) = \alpha_T(i) \tag{17.91}$$

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

17.5 Learning for HMMs

We now discuss how to estimate the parameters $\boldsymbol{\theta} = (\boldsymbol{\pi}, \mathbf{A}, \mathbf{B})$, 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 $\mathbf{B}$ are the parameters of the class-conditional densities $p(\mathbf{x}_t|z_t = j)$. We first consider the case where $\mathbf{z}_{1:T}$ is observed in the training set, and then the harder case where $\mathbf{z}_{1:T}$ is hidden.

17.5.1 Training with fully observed data

If we observe the hidden state sequences, we can compute the MLEs for $\mathbf{A}$ and $\boldsymbol{\pi}$ 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 $\mathbf{B}$ 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 $l \in \{1, \ldots, L\}$ represents the observed symbol, the MLE is given by

$$\hat{B}_{jl} = \frac{N_{jl}^X}{N_j}, \quad N_{jl}^X \triangleq \sum_{i=1}^{N} \sum_{t=1}^{T_i} \mathbb{I}(z_{i,t} = j, x_{i,t} = l) \tag{17.92}$$