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:      LLLLLLLLLLLLLLFFFFFFLLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFLLLLLLLL
```

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

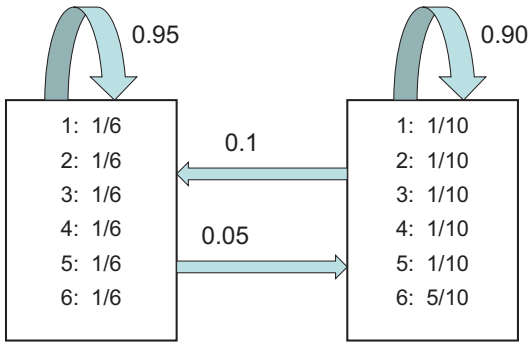
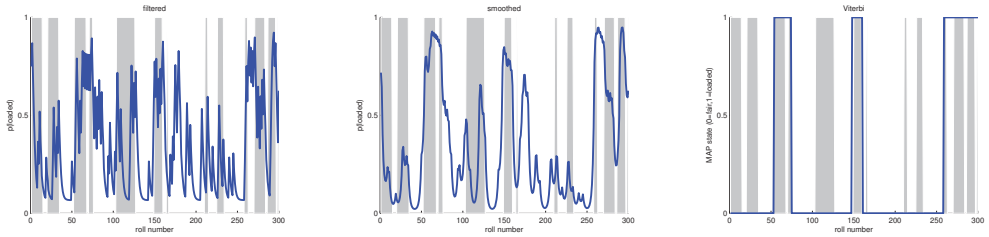


Figure 17.9 An HMM for the occasionally dishonest casino. The blue arrows visualize the state transition diagram $\mathbf{A}$. Based on (Durbin et al. 1998, p54).



(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.

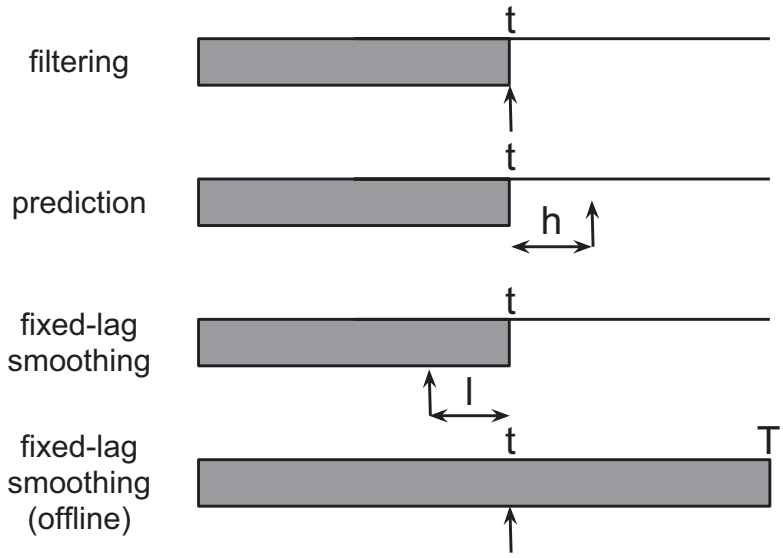


Figure 17.11 The main kinds of inference for state-space models. The shaded region is the interval for which we have data. The arrow represents the time step at which we want to perform inference. t is the current time, T is the sequence length, ℓ is the lag and h is the prediction horizon. See text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

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 $\pi(j) = p(z_1 = j)$;
$[\alpha_1, Z_1] = \text{normalize}(\boldsymbol{\psi}_1 \odot \boldsymbol{\pi})$;
for $t = 2 : T$ **do**
  $[\alpha_t, Z_t] = \text{normalize}(\boldsymbol{\psi}_t \odot (\boldsymbol{\Psi}^T \boldsymbol{\alpha}_{t-1}))$;
Return $\boldsymbol{\alpha}_{1:T}$ and $\log p(\mathbf{y}_{1:T}) = \sum_t \log Z_t$;
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

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

The base case is

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

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:

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

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

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

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

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

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

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

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**:

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

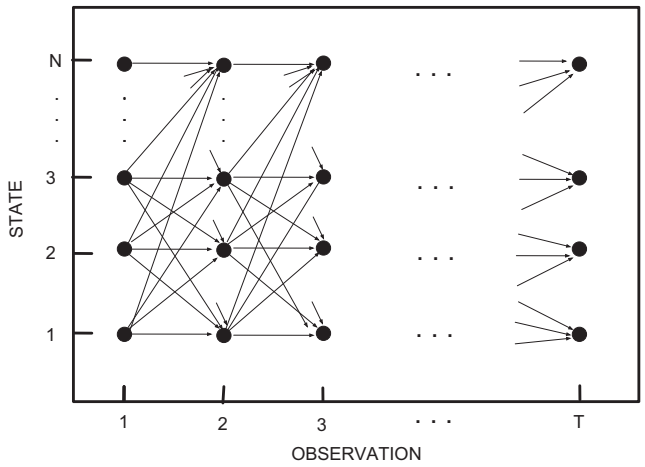


Figure 17.12 The trellis of states vs time for a Markov chain. Based on (Rabiner 1989).

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

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,

---

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).

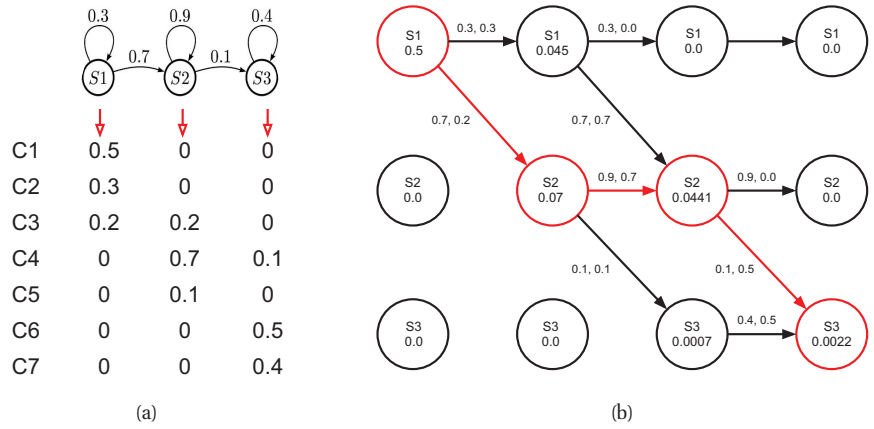


Figure 17.13 Illustration of Viterbi decoding in a simple HMM for speech recognition. (a) A 3-state HMM for a single phone. We are visualizing the state transition diagram. We assume the observations have been vector quantized into 7 possible symbols, $C_1, \ldots, C_7$. Each state z_1, z_2, z_3 has 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

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

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

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:

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

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

$$p(z_t = i|z_{t+1} = j, \mathbf{x}_{1:t}) = p(z_t|z_{t+1}, \mathbf{x}_{1:t}, \cancel{\mathbf{x}_{t+1}}) \tag{17.86}$$

$$= \frac{p(z_{t+1}, z_t|\mathbf{x}_{1:t+1})}{p(z_{t+1}|\mathbf{x}_{1:t+1})} \tag{17.87}$$

$$\propto \frac{p(\mathbf{x}_{t+1}|z_{t+1}, \cancel{z_t}, \cancel{\mathbf{x}_{1:t}})p(z_{t+1}, z_t|\mathbf{x}_{1:t})}{p(z_{t+1}|\mathbf{x}_{1:t+1})} \tag{17.88}$$

$$= \frac{p(\mathbf{x}_{t+1}|z_{t+1})p(z_{t+1}|z_t, \cancel{\mathbf{x}_{1:t}})p(z_t|\mathbf{x}_{1:t})}{p(z_{t+1}|\mathbf{x}_{1:t+1})} \tag{17.89}$$

$$= \frac{\phi_{t+1}(j)\psi(i,j)\alpha_t(i)}{\alpha_{t+1}(j)} \tag{17.90}$$

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

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