CHAPTER

# A Hidden Markov Models

Chapter 8 introduced the Hidden Markov Model and applied it to part of speech tagging. Part of speech tagging is a fully-supervised learning task, because we have a corpus of words labeled with the correct part-of-speech tag. But many applications don't have labeled data. So in this chapter, we introduce the full set of algorithms for HMMs, including the key unsupervised learning algorithm for HMM, the Forward-Backward algorithm. We'll repeat some of the text from Chapter 8 for readers who want the whole story laid out in a single chapter.

## A.1 Markov Chains

**Markov chain**

The HMM is based on augmenting the Markov chain. A **Markov chain** is a model that tells us something about the probabilities of sequences of random variables, *states*, each of which can take on values from some set. These sets can be words, or tags, or symbols representing anything, like the weather. A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the current state. The states before the current state have no impact on the future except via the current state. It's as if to predict tomorrow's weather you could examine today's weather but you weren't allowed to look at yesterday's weather.

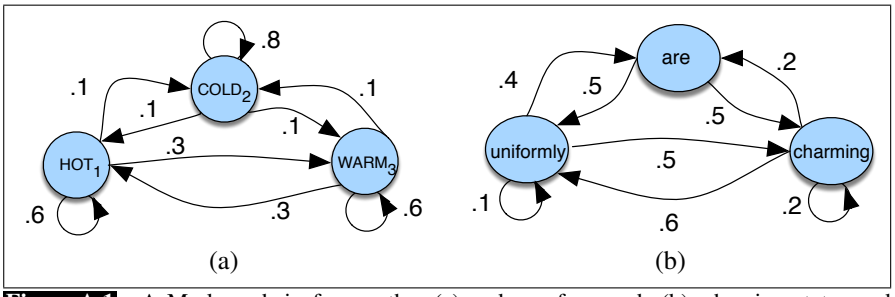


**Figure A.1** A Markov chain for weather (a) and one for words (b), showing states and transitions. A start distribution $\pi$ is required; setting $\pi = [0.1, 0.7, 0.2]$ for (a) would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

**Markov assumption**

More formally, consider a sequence of state variables $q_1, q_2, ..., q_i$. A Markov model embodies the **Markov assumption** on the probabilities of this sequence: that when predicting the future, the past doesn't matter, only the present.

$$\textbf{Markov Assumption:} \quad P(q_i = a | q_1 ... q_{i-1}) = P(q_i = a | q_{i-1}) \tag{A.1}$$

Figure A.1a shows a Markov chain for assigning a probability to a sequence of weather events, for which the vocabulary consists of HOT, COLD, and WARM. The states are represented as nodes in the graph, and the transitions, with their probabilities, as edges. The transitions are probabilities: the values of arcs leaving a given

state must sum to 1. Figure A.1b shows a Markov chain for assigning a probability to a sequence of words $w_1 ... w_n$. This Markov chain should be familiar; in fact, it represents a bigram language model, with each edge expressing the probability $p(w_i|w_j)$! Given the two models in Fig. A.1, we can assign a probability to any sequence from our vocabulary.

Formally, a Markov chain is specified by the following components:

| | |
|---|---|
| $Q = q_1 q_2 \ldots q_N$ | a set of $N$ **states** |
| $A = a_{11} a_{12} \ldots a_{N1} \ldots a_{NN}$ | a **transition probability matrix** $A$, each $a_{ij}$ representing the probability of moving from state $i$ to state $j$, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$ |
| $\pi = \pi_1, \pi_2, ..., \pi_N$ | an **initial probability distribution** over states. $\pi_i$ is the probability that the Markov chain will start in state $i$. Some states $j$ may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{N} \pi_i = 1$ |

Before you go on, use the sample probabilities in Fig. A.1a (with $\pi = [.1, .7., 2]$) to compute the probability of each of the following sequences:

(A.2) hot hot hot hot
(A.3) cold hot cold hot

What does the difference in these probabilities tell you about a real-world weather fact encoded in Fig. A.1a?

## A.2 The Hidden Markov Model

A Markov chain is useful when we need to compute a probability for a sequence of observable events. In many cases, however, the events we are interested in are **hidden**: we don't observe them directly. For example we don't normally observe part-of-speech tags in a text. Rather, we see words, and must infer the tags from the word sequence. We call the tags **hidden** because they are not observed.

**hidden**

**Hidden Markov model**

A **hidden Markov model** (**HMM**) allows us to talk about both *observed* events (like words that we see in the input) and *hidden* events (like part-of-speech tags) that we think of as causal factors in our probabilistic model. An HMM is specified by the following components:

| | |
|---|---|
| $Q = q_1 q_2 \ldots q_N$ | a set of $N$ **states** |
| $A = a_{11} \ldots a_{ij} \ldots a_{NN}$ | a **transition probability matrix** $A$, each $a_{ij}$ representing the probability of moving from state $i$ to state $j$, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$ |
| $B = b_i(o_t)$ | a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation $o_t$ (drawn from a vocabulary $V = v_1, v_2, ..., v_V$) being generated from a state $q_i$ |
| $\pi = \pi_1, \pi_2, ..., \pi_N$ | an **initial probability distribution** over states. $\pi_i$ is the probability that the Markov chain will start in state $i$. Some states $j$ may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$ |

The HMM is given as input $O = o_1 o_2 \ldots o_T$: a sequence of $T$ **observations**, each one drawn from the vocabulary $V$.

A first-order hidden Markov model instantiates two simplifying assumptions. First, as with a first-order Markov chain, the probability of a particular state depends

only on the previous state:

$$\textbf{Markov Assumption:} \quad P(q_i|q_1 \ldots q_{i-1}) = P(q_i|q_{i-1}) \tag{A.4}$$

Second, the probability of an output observation $o_i$ depends only on the state that produced the observation $q_i$ and not on any other states or any other observations:

$$\textbf{Output Independence:} \quad P(o_i|q_1 \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i|q_i) \tag{A.5}$$

To exemplify these models, we'll use a task invented by Jason Eisner (2002). Imagine that you are a climatologist in the year 2799 studying the history of global warming. You cannot find any records of the weather in Baltimore, Maryland, for the summer of 2020, but you do find Jason Eisner's diary, which lists how many ice creams Jason ate every day that summer. Our goal is to use these observations to estimate the temperature every day. We'll simplify this weather task by assuming there are only two kinds of days: cold (C) and hot (H). So the Eisner task is as follows:

> Given a sequence of observations $O$ (each an integer representing the number of ice creams eaten on a given day) find the 'hidden' sequence $Q$ of weather states (H or C) which caused Jason to eat the ice cream.

Figure A.2 shows a sample HMM for the ice cream task. The two hidden states (H and C) correspond to hot and cold weather, and the observations (drawn from the alphabet $O = \{1,2,3\}$) correspond to the number of ice creams eaten by Jason on a given day.

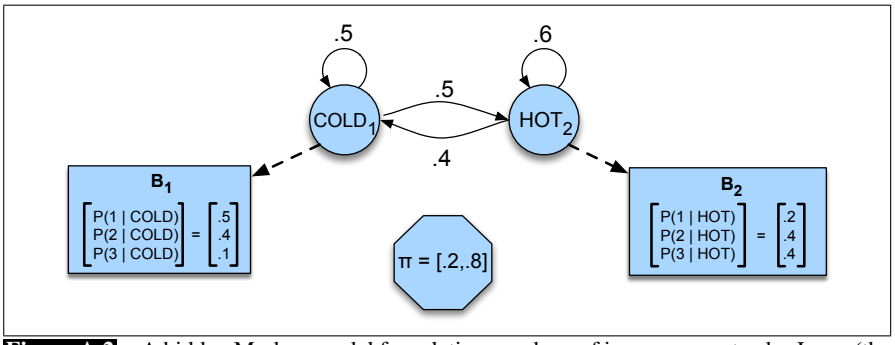


**Figure A.2** A hidden Markov model for relating numbers of ice creams eaten by Jason (the observations) to the weather (H or C, the hidden variables).

An influential tutorial by Rabiner (1989), based on tutorials by Jack Ferguson in the 1960s, introduced the idea that hidden Markov models should be characterized by **three fundamental problems**:

| | |
|---|---|
| **Problem 1 (Likelihood):** | Given an HMM $\lambda = (A,B)$ and an observation sequence $O$, determine the likelihood $P(O|\lambda)$. |
| **Problem 2 (Decoding):** | Given an observation sequence $O$ and an HMM $\lambda = (A,B)$, discover the best hidden state sequence $Q$. |
| **Problem 3 (Learning):** | Given an observation sequence $O$ and the set of states in the HMM, learn the HMM parameters $A$ and $B$. |

We already saw an example of Problem 2 in Chapter 8. In the next two sections we introduce the Forward and Forward-Backward algorithms to solve Problems 1 and 3 and give more information on Problem 2

## A.3 Likelihood Computation: The Forward Algorithm

Our first problem is to compute the likelihood of a particular observation sequence. For example, given the ice-cream eating HMM in Fig. A.2, what is the probability of the sequence *3 1 3*? More formally:

> **Computing Likelihood:** Given an HMM $\lambda = (A, B)$ and an observation sequence $O$, determine the likelihood $P(O|\lambda)$.

For a Markov chain, where the surface observations are the same as the hidden events, we could compute the probability of *3 1 3* just by following the states labeled *3 1 3* and multiplying the probabilities along the arcs. For a hidden Markov model, things are not so simple. We want to determine the probability of an ice-cream observation sequence like *3 1 3*, but we don't know what the hidden state sequence is!

Let's start with a slightly simpler situation. Suppose we already knew the weather and wanted to predict how much ice cream Jason would eat. This is a useful part of many HMM tasks. For a given hidden state sequence (e.g., *hot hot cold*), we can easily compute the output likelihood of *3 1 3*.

Let's see how. First, recall that for hidden Markov models, each hidden state produces only a single observation. Thus, the sequence of hidden states and the sequence of observations have the same length. [1]

Given this one-to-one mapping and the Markov assumptions expressed in Eq. A.4, for a particular hidden state sequence $Q = q_0, q_1, q_2, ..., q_T$ and an observation sequence $O = o_1, o_2, ..., o_T$, the likelihood of the observation sequence is

$$P(O|Q) = \prod_{i=1}^{T} P(o_i|q_i) \tag{A.6}$$

The computation of the forward probability for our ice-cream observation *3 1 3* from one possible hidden state sequence *hot hot cold* is shown in Eq. A.7. Figure A.3 shows a graphic representation of this computation.

$$P(3\ 1\ 3|\text{hot hot cold}) = P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \tag{A.7}$$

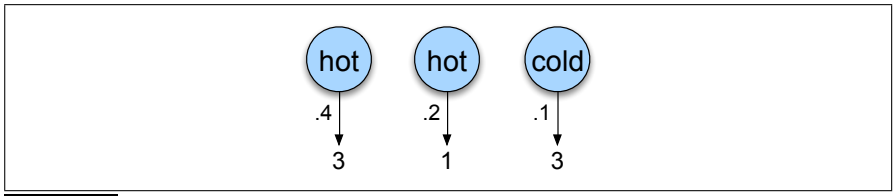


**Figure A.3** The computation of the observation likelihood for the ice-cream events *3 1 3* given the hidden state sequence *hot hot cold*.

But of course, we don't actually know what the hidden state (weather) sequence was. We'll need to compute the probability of ice-cream events *3 1 3* instead by

[1] In a variant of HMMs called **segmental HMMs** (in speech recognition) or **semi-HMMs** (in text processing) this one-to-one mapping between the length of the hidden state sequence and the length of the observation sequence does not hold.

summing over all possible weather sequences, weighted by their probability. First, let's compute the joint probability of being in a particular weather sequence $Q$ and generating a particular sequence $O$ of ice-cream events. In general, this is

$$P(O,Q) = P(O|Q) \times P(Q) = \prod_{i=1}^{T} P(o_i|q_i) \times \prod_{i=1}^{T} P(q_i|q_{i-1}) \tag{A.8}$$

The computation of the joint probability of our ice-cream observation *3 1 3* and one possible hidden state sequence *hot hot cold* is shown in Eq. A.9. Figure A.4 shows a graphic representation of this computation.

$$\begin{aligned} P(3\ 1\ 3, \text{hot hot cold}) &= P(\text{hot}|\text{start}) \times P(\text{hot}|\text{hot}) \times P(\text{cold}|\text{hot}) \\ &\quad \times P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \end{aligned} \tag{A.9}$$

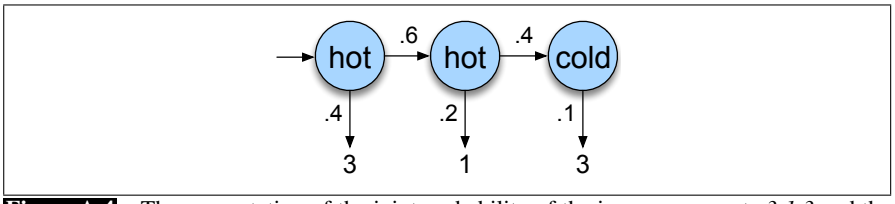


**Figure A.4** The computation of the joint probability of the ice-cream events *3 1 3* and the hidden state sequence *hot hot cold*.

Now that we know how to compute the joint probability of the observations with a particular hidden state sequence, we can compute the total probability of the observations just by summing over all possible hidden state sequences:

$$P(O) = \sum_{Q} P(O,Q) = \sum_{Q} P(O|Q)P(Q) \tag{A.10}$$

For our particular case, we would sum over the eight 3-event sequences *cold cold cold*, *cold cold hot*, that is,

$$P(3\ 1\ 3) = P(3\ 1\ 3, \text{cold cold cold}) + P(3\ 1\ 3, \text{cold cold hot}) + P(3\ 1\ 3, \text{hot hot cold}) + \ldots$$

For an HMM with $N$ hidden states and an observation sequence of $T$ observations, there are $N^T$ possible hidden sequences. For real tasks, where $N$ and $T$ are both large, $N^T$ is a very large number, so we cannot compute the total observation likelihood by computing a separate observation likelihood for each hidden state sequence and then summing them.

**forward algorithm**

Instead of using such an extremely exponential algorithm, we use an efficient $O(N^2T)$ algorithm called the **forward algorithm**. The forward algorithm is a kind of **dynamic programming** algorithm, that is, an algorithm that uses a table to store intermediate values as it builds up the probability of the observation sequence. The forward algorithm computes the observation probability by summing over the probabilities of all possible hidden state paths that could generate the observation sequence, but it does so efficiently by implicitly folding each of these paths into a single **forward trellis**.

Figure A.5 shows an example of the forward trellis for computing the likelihood of *3 1 3* given the hidden state sequence *hot hot cold*.

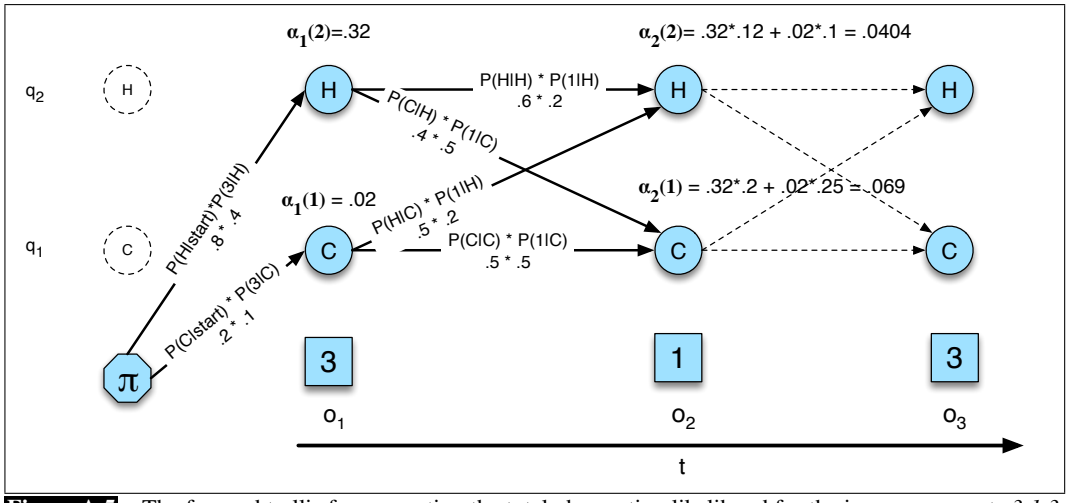


**Figure A.5** The forward trellis for computing the total observation likelihood for the ice-cream events *3 1 3*. Hidden states are in circles, observations in squares. The figure shows the computation of $\alpha_t(j)$ for two states at two time steps. The computation in each cell follows Eq. A.12: $\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t)$. The resulting probability expressed in each cell is Eq. A.11: $\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda)$.

Each cell of the forward algorithm trellis $\alpha_t(j)$ represents the probability of being in state $j$ after seeing the first $t$ observations, given the automaton $\lambda$. The value of each cell $\alpha_t(j)$ is computed by summing over the probabilities of every path that could lead us to this cell. Formally, each cell expresses the following probability:

$$\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda) \tag{A.11}$$

Here, $q_t = j$ means "the $t^{\text{th}}$ state in the sequence of states is state $j$". We compute this probability $\alpha_t(j)$ by summing over the extensions of all the paths that lead to the current cell. For a given state $q_j$ at time $t$, the value $\alpha_t(j)$ is computed as

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t) \tag{A.12}$$

The three factors that are multiplied in Eq. A.12 in extending the previous paths to compute the forward probability at time $t$ are

| | |
|---|---|
| $\alpha_{t-1}(i)$ | the **previous forward path probability** from the previous time step |
| $a_{ij}$ | the **transition probability** from previous state $q_i$ to current state $q_j$ |
| $b_j(o_t)$ | the **state observation likelihood** of the observation symbol $o_t$ given the current state $j$ |

Consider the computation in Fig. A.5 of $\alpha_2(2)$, the forward probability of being at time step 2 in state 2 having generated the partial observation *3 1*. We compute by extending the $\alpha$ probabilities from time step 1, via two paths, each extension consisting of the three factors above: $\alpha_1(1) \times P(H|C) \times P(1|H)$ and $\alpha_1(2) \times P(H|H) \times P(1|H)$.

Figure A.6 shows another visualization of this induction step for computing the value in one new cell of the trellis.

We give two formal definitions of the forward algorithm: the pseudocode in Fig. A.7 and a statement of the definitional recursion here.

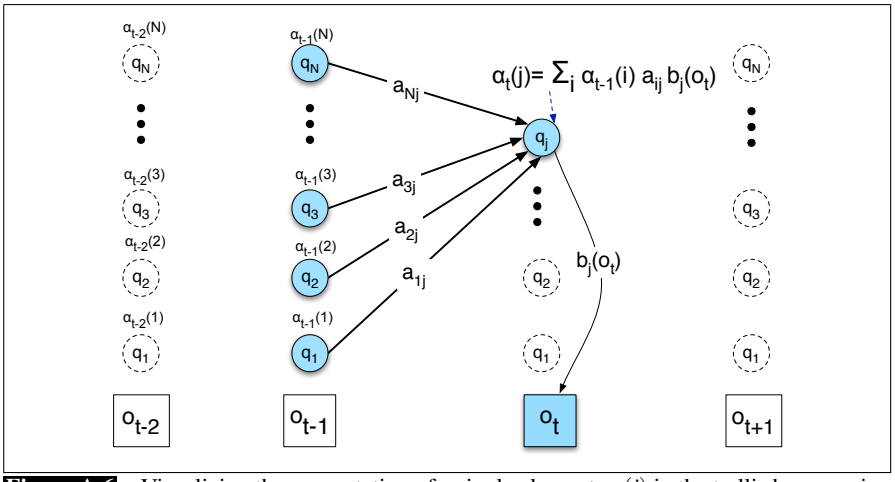


**Figure A.6** Visualizing the computation of a single element $\alpha_t(i)$ in the trellis by summing all the previous values $\alpha_{t-1}$, weighted by their transition probabilities $a$, and multiplying by the observation probability $b_i(o_t)$. For many applications of HMMs, many of the transition probabilities are 0, so not all previous states will contribute to the forward probability of the current state. Hidden states are in circles, observations in squares. Shaded nodes are included in the probability computation for $\alpha_t(i)$.

**function** FORWARD(*observations* of len $T$, *state-graph* of len $N$) **returns** *forward-prob*

create a probability matrix *forward[N,T]*
**for** each state $s$ **from** 1 **to** $N$ **do** ; initialization step
$forward[s,1] \leftarrow \pi_s * b_s(o_1)$
**for** each time step $t$ **from** 2 **to** $T$ **do** ; recursion step
**for** each state $s$ **from** 1 **to** $N$ **do**

$$forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$$

$$forwardprob \leftarrow \sum_{s=1}^{N} forward[s,T]$$ ; termination step
**return** *forwardprob*

**Figure A.7** The forward algorithm, where $forward[s,t]$ represents $\alpha_t(s)$.

1. Initialization:

$$\alpha_1(j) = \pi_j b_j(o_1) \quad 1 \le j \le N$$

2. Recursion:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$

3. Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

## A.4 Decoding: The Viterbi Algorithm

**decoding**

For any model, such as an HMM, that contains hidden variables, the task of determining which sequence of variables is the underlying source of some sequence of observations is called the **decoding** task. In the ice-cream domain, given a sequence of ice-cream observations *3 1 3* and an HMM, the task of the **decoder** is to find the best hidden weather sequence (*H H H*). More formally,

> **Decoding**: Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, ..., o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \ldots q_T$.

We might propose to find the best sequence as follows: For each possible hidden state sequence (*HHH*, *HHC*, *HCH*, etc.), we could run the forward algorithm and compute the likelihood of the observation sequence given that hidden state sequence. Then we could choose the hidden state sequence with the maximum observation likelihood. It should be clear from the previous section that we cannot do this because there are an exponentially large number of state sequences.

**Viterbi algorithm**

Instead, the most common decoding algorithms for HMMs is the **Viterbi algorithm**. Like the forward algorithm, **Viterbi** is a kind of **dynamic programming** that makes uses of a dynamic programming trellis. Viterbi also strongly resembles another dynamic programming variant, the **minimum edit distance** algorithm of Chapter 2.

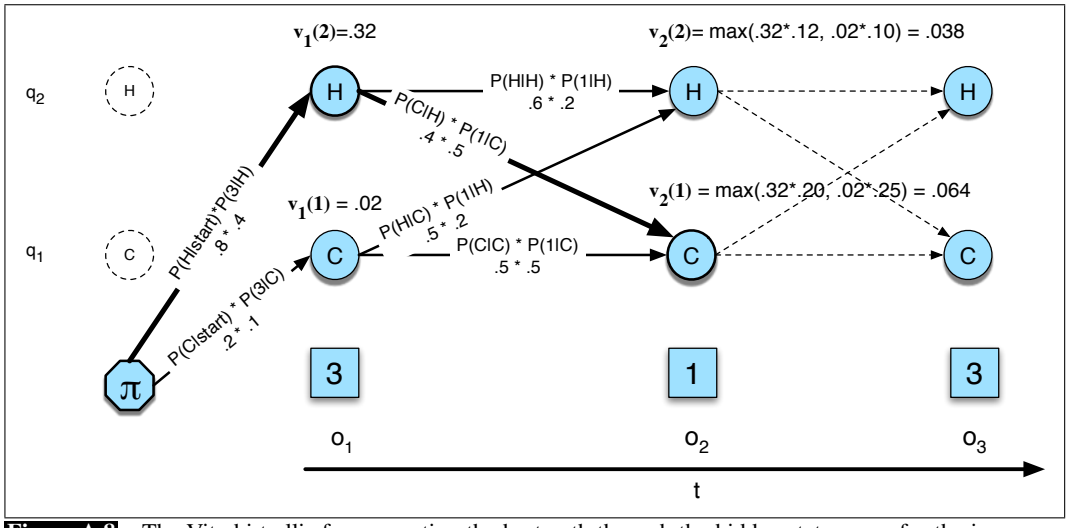


**Figure A.8** The Viterbi trellis for computing the best path through the hidden state space for the ice-cream eating events *3 1 3*. Hidden states are in circles, observations in squares. White (unfilled) circles indicate illegal transitions. The figure shows the computation of $v_t(j)$ for two states at two time steps. The computation in each cell follows Eq. A.14: $v_t(j) = \max_{1 \le i \le N-1} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$. The resulting probability expressed in each cell is Eq. A.13: $v_t(j) = P(q_0, q_1, \ldots, q_{t-1}, o_1, o_2, \ldots, o_t, q_t = j|\lambda)$.

Figure A.8 shows an example of the Viterbi trellis for computing the best hidden state sequence for the observation sequence *3 1 3*. The idea is to process the observation sequence left to right, filling out the trellis. Each cell of the trellis, $v_t(j)$, represents the probability that the HMM is in state $j$ after seeing the first $t$ observations and passing through the most probable state sequence $q_1, ..., q_{t-1}$, given the

automaton $\lambda$. The value of each cell $v_t(j)$ is computed by recursively taking the most probable path that could lead us to this cell. Formally, each cell expresses the probability

$$v_t(j) = \max_{q_1,\cdots,q_{t-1}} P(q_1 \ldots q_{t-1}, o_1, o_2 \ldots o_t, q_t = j|\lambda) \tag{A.13}$$

Note that we represent the most probable path by taking the maximum over all possible previous state sequences $\max_{q_1,\cdots,q_{t-1}}$. Like other dynamic programming algorithms, Viterbi fills each cell recursively. Given that we had already computed the probability of being in every state at time $t-1$, we compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell. For a given state $q_j$ at time $t$, the value $v_t(j)$ is computed as

$$v_t(j) \;=\; \max_{i=1}^{N} v_{t-1}(i)\; a_{ij}\; b_j(o_t) \tag{A.14}$$

The three factors that are multiplied in Eq. A.14 for extending the previous paths to compute the Viterbi probability at time $t$ are

| | |
|---|---|
| $v_{t-1}(i)$ | the **previous Viterbi path probability** from the previous time step |
| $a_{ij}$ | the **transition probability** from previous state $q_i$ to current state $q_j$ |
| $b_j(o_t)$ | the **state observation likelihood** of the observation symbol $o_t$ given the current state $j$ |

**function** VITERBI(*observations* of len $T$,*state-graph* of len $N$) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
**for** each state $s$ **from** 1 **to** $N$ **do** ; initialization step
  *viterbi*[$s$,1] $\leftarrow \pi_s * b_s(o_1)$
  *backpointer*[s,1] $\leftarrow 0$
**for** each time step $t$ **from** 2 **to** $T$ **do** ; recursion step
  **for** each state $s$ **from** 1 **to** $N$ **do**
    *viterbi*[s,t] $\leftarrow \max_{s'=1}^{N}$ *viterbi*$[s',t-1] * a_{s',s} * b_s(o_t)$
    *backpointer*[s,t] $\leftarrow \operatorname{argmax}_{s'=1}^{N}$ *viterbi*$[s',t-1] * a_{s',s} * b_s(o_t)$
*bestpathprob* $\leftarrow \max_{s=1}^{N}$ *viterbi*$[s,T]$ ; termination step
*bestpathpointer* $\leftarrow \operatorname{argmax}_{s=1}^{N}$ *viterbi*$[s,T]$ ; termination step
*bestpath* $\leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
**return** *bestpath*, *bestpathprob*

**Figure A.9** Viterbi algorithm for finding optimal sequence of hidden states. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

Figure A.9 shows pseudocode for the Viterbi algorithm. Note that the Viterbi algorithm is identical to the forward algorithm except that it takes the **max** over the previous path probabilities whereas the forward algorithm takes the **sum**. Note also that the Viterbi algorithm has one component that the forward algorithm doesn't

have: **backpointers**. The reason is that while the forward algorithm needs to produce an observation likelihood, the Viterbi algorithm must produce a probability and also the most likely state sequence. We compute this best state sequence by keeping track of the path of hidden states that led to each state, as suggested in Fig. A.10, and then at the end backtracing the best path to the beginning (the Viterbi **backtrace**).

**Viterbi backtrace**

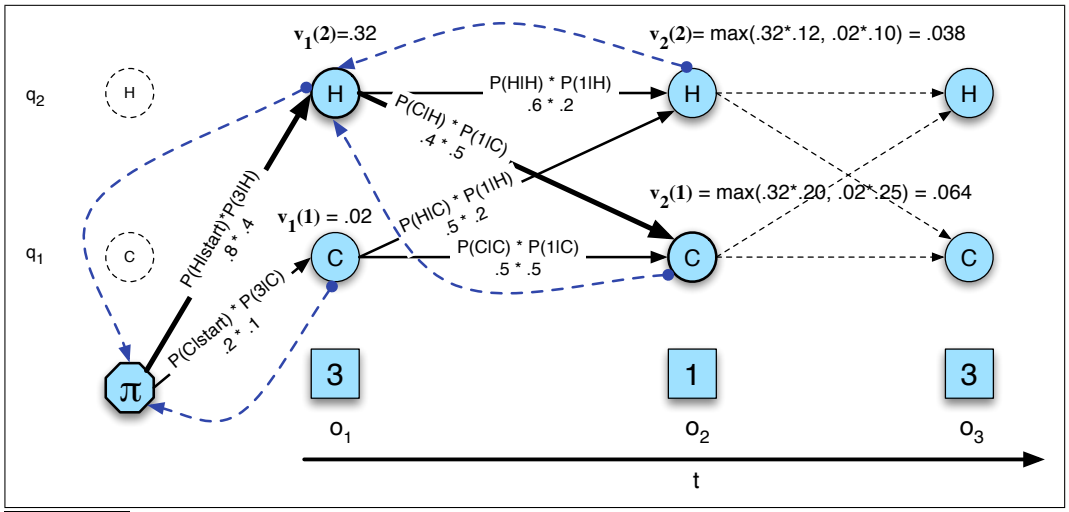


**Figure A.10** The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

Finally, we can give a formal definition of the Viterbi recursion as follows:

1. **Initialization:**

$$\begin{aligned} v_1(j) &= \pi_j b_j(o_1) \qquad 1 \le j \le N \\ bt_1(j) &= 0 \qquad 1 \le j \le N \end{aligned}$$

2. **Recursion**

$$\begin{aligned} v_t(j) &= \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \\ bt_t(j) &= \operatorname*{argmax}_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \end{aligned}$$

3. **Termination:**

$$\begin{aligned} \text{The best score:} \quad P* &= \max_{i=1}^{N} v_T(i) \\ \text{The start of backtrace:} \quad q_T* &= \operatorname*{argmax}_{i=1}^{N} v_T(i) \end{aligned}$$

## A.5 HMM Training: The Forward-Backward Algorithm

We turn to the third problem for HMMs: learning the parameters of an HMM, that is, the $A$ and $B$ matrices. Formally,

> **Learning:** Given an observation sequence $O$ and the set of possible states in the HMM, learn the HMM parameters $A$ and $B$.

The input to such a learning algorithm would be an unlabeled sequence of observations $O$ and a vocabulary of potential hidden states $Q$. Thus, for the ice cream task, we would start with a sequence of observations $O = \{1, 3, 2, ..., \}$ and the set of hidden states $H$ and $C$.

Forward-backward
Baum-Welch
EM

The standard algorithm for HMM training is the **forward-backward**, or **Baum-Welch** algorithm (Baum, 1972), a special case of the **Expectation-Maximization** or **EM** algorithm (Dempster et al., 1977). The algorithm will let us train both the transition probabilities $A$ and the emission probabilities $B$ of the HMM. EM is an *iterative* algorithm, computing an initial estimate for the probabilities, then using those estimates to computing a better estimate, and so on, iteratively improving the probabilities that it learns.

Let us begin by considering the much simpler case of training a fully visible Markov model, where we know both the temperature and the ice cream count for every day. That is, imagine we see the following set of input observations and magically knew the aligned hidden state sequences:

```
3    3   2        1    1    2        1    2  3
hot  hot cold     cold cold cold    cold hot hot
```

This would easily allow us to compute the HMM parameters just by maximum likelihood estimation from the training data. First, we can compute $\pi$ from the count of the 3 initial hidden states:

$$\pi_h = 1/3 \quad \pi_c = 2/3$$

Next we can directly compute the $A$ matrix from the transitions, ignoring the final hidden states:

$$p(hot|hot) = 2/3 \quad p(cold|hot) = 1/3$$
$$p(cold|cold) = 2/3 \quad p(hot|cold) = 1/3$$

and the $B$ matrix:

$$P(1|hot) = 0/4 = 0 \quad p(1|cold) = 3/5 = .6$$
$$P(2|hot) = 1/4 = .25 \quad p(2|cold = 2/5 = .4$$
$$P(3|hot) = 3/4 = .75 \quad p(3|cold) = 0$$

For a real HMM, we cannot compute these counts directly from an observation sequence since we don't know which path of states was taken through the machine for a given input. For example, suppose I didn't tell you the temperature on day 2, and you had to guess it, but you (magically) had the above probabilities, and the temperatures on the other days. You could do some Bayesian arithmetic with all the other probabilities to get estimates of the likely temperature on that missing day, and use those to get expected counts for the temperatures for day 2.

But the real problem is even harder: we don't know the counts of being in **any** of the hidden states!! The Baum-Welch algorithm solves this by *iteratively* estimating the counts. We will start with an estimate for the transition and observation probabilities and then use these estimated probabilities to derive better and better probabilities. And we're going to do this by computing the forward probability for an observation and then dividing that probability mass among all the different paths that contributed to this forward probability.

backward probability

To understand the algorithm, we need to define a useful probability related to the forward probability and called the **backward probability**. The backward probabil-

ity $\beta$ is the probability of seeing the observations from time $t+1$ to the end, given that we are in state $i$ at time $t$ (and given the automaton $\lambda$):

$$\beta_t(i) = P(o_{t+1}, o_{t+2} \dots o_T | q_t = i, \lambda) \tag{A.15}$$

It is computed inductively in a similar manner to the forward algorithm.

1. **Initialization:**

$$\beta_T(i) \;=\; 1, \quad 1 \le i \le N$$

2. **Recursion**

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j), \quad 1 \le i \le N, 1 \le t < T$$

3. **Termination:**

$$P(O|\lambda) = \sum_{j=1}^{N} \pi_j\, b_j(o_1)\, \beta_1(j)$$

Figure A.11 illustrates the backward induction step.

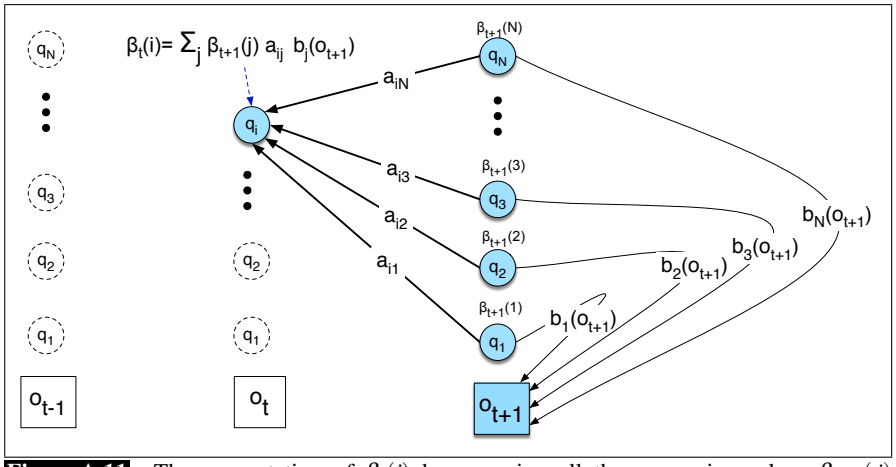


**Figure A.11** The computation of $\beta_t(i)$ by summing all the successive values $\beta_{t+1}(j)$ weighted by their transition probabilities $a_{ij}$ and their observation probabilities $b_j(o_{t+1})$.

We are now ready to see how the forward and backward probabilities can help compute the transition probability $a_{ij}$ and observation probability $b_i(o_t)$ from an observation sequence, even though the actual path taken through the model is hidden.

Let's begin by seeing how to estimate $\hat{a}_{ij}$ by a variant of simple maximum likelihood estimation:

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} \tag{A.16}$$

How do we compute the numerator? Here's the intuition. Assume we had some estimate of the probability that a given transition $i \to j$ was taken at a particular point in time $t$ in the observation sequence. If we knew this probability for each particular time $t$, we could sum over all times $t$ to estimate the total count for the transition $i \to j$.

More formally, let's define the probability $\xi_t$ as the probability of being in state $i$ at time $t$ and state $j$ at time $t+1$, given the observation sequence and of course the model:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j | O, \lambda) \tag{A.17}$$

To compute $\xi_t$, we first compute a probability which is similar to $\xi_t$, but differs in including the probability of the observation; note the different conditioning of $O$ from Eq. A.17:

$$\text{not-quite-}\xi_t(i,j) = P(q_t = i, q_{t+1} = j, O | \lambda) \tag{A.18}$$

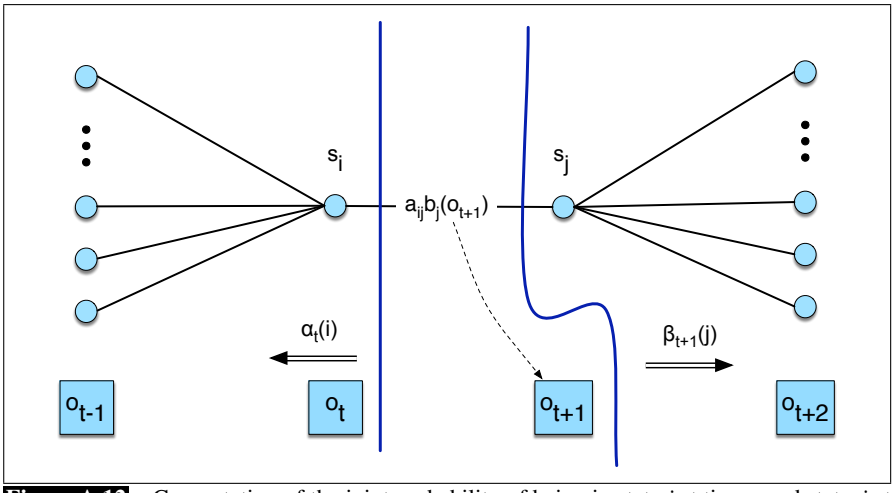


**Figure A.12** Computation of the joint probability of being in state $i$ at time $t$ and state $j$ at time $t+1$. The figure shows the various probabilities that need to be combined to produce $P(q_t = i, q_{t+1} = j, O|\lambda)$: the $\alpha$ and $\beta$ probabilities, the transition probability $a_{ij}$ and the observation probability $b_j(o_{t+1})$. After Rabiner (1989) which is ©1989 IEEE.

Figure A.12 shows the various probabilities that go into computing not-quite-$\xi_t$: the transition probability for the arc in question, the $\alpha$ probability before the arc, the $\beta$ probability after the arc, and the observation probability for the symbol just after the arc. These four are multiplied together to produce *not-quite-*$\xi_t$ as follows:

$$\text{not-quite-}\xi_t(i,j) = \alpha_t(i)\, a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \tag{A.19}$$

To compute $\xi_t$ from *not-quite-*$\xi_t$, we follow the laws of probability and divide by $P(O|\lambda)$, since

$$P(X|Y,Z) = \frac{P(X,Y|Z)}{P(Y|Z)} \tag{A.20}$$

The probability of the observation given the model is simply the forward probability of the whole utterance (or alternatively, the backward probability of the whole utterance):

$$P(O|\lambda) = \sum_{j=1}^{N} \alpha_t(j) \beta_t(j) \tag{A.21}$$

So, the final equation for $\xi_t$ is

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \tag{A.22}$$

The expected number of transitions from state $i$ to state $j$ is then the sum over all $t$ of $\xi$. For our estimate of $a_{ij}$ in Eq. A.16, we just need one more thing: the total expected number of transitions from state $i$. We can get this by summing over all transitions out of state $i$. Here's the final formula for $\hat{a}_{ij}$:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)} \tag{A.23}$$

We also need a formula for recomputing the observation probability. This is the probability of a given symbol $v_k$ from the observation vocabulary $V$, given a state $j$: $\hat{b}_j(v_k)$. We will do this by trying to compute

$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j} \tag{A.24}$$

For this, we will need to know the probability of being in state $j$ at time $t$, which we will call $\gamma_t(j)$:

$$\gamma_t(j) = P(q_t = j | O, \lambda) \tag{A.25}$$

Once again, we will compute this by including the observation sequence in the probability:

$$\gamma_t(j) = \frac{P(q_t = j, O|\lambda)}{P(O|\lambda)} \tag{A.26}$$

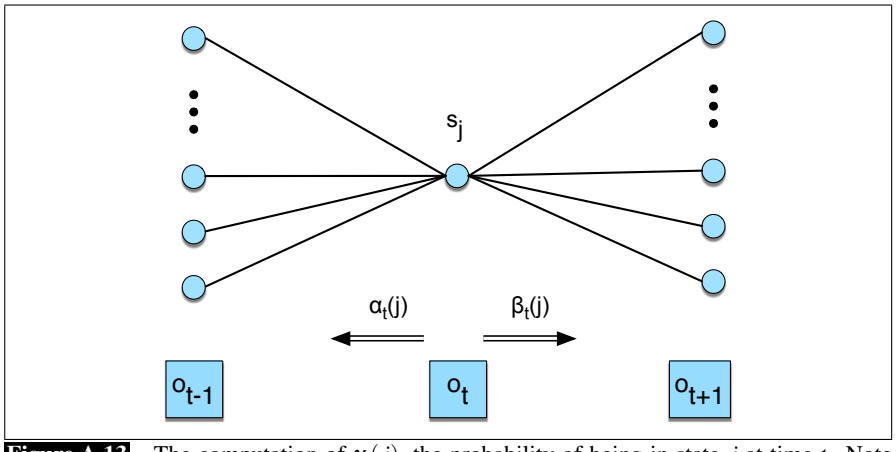


**Figure A.13** The computation of $\gamma_t(j)$, the probability of being in state $j$ at time $t$. Note that $\gamma$ is really a degenerate case of $\xi$ and hence this figure is like a version of Fig. A.12 with state $i$ collapsed with state $j$. After Rabiner (1989) which is ©1989 IEEE.

As Fig. A.13 shows, the numerator of Eq. A.26 is just the product of the forward probability and the backward probability:

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \tag{A.27}$$

We are ready to compute $b$. For the numerator, we sum $\gamma_t(j)$ for all time steps $t$ in which the observation $o_t$ is the symbol $v_k$ that we are interested in. For the denominator, we sum $\gamma_t(j)$ over all time steps $t$. The result is the percentage of the times that we were in state $j$ and saw symbol $v_k$ (the notation $\sum_{t=1\ s.t. O_t=v_k}^{T}$ means "sum over all $t$ for which the observation at time $t$ was $v_k$"):

$$\hat{b}_j(v_k) = \frac{\sum_{t=1\ s.t. O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{A.28}$$

We now have ways in Eq. A.23 and Eq. A.28 to *re-estimate* the transition $A$ and observation $B$ probabilities from an observation sequence $O$, assuming that we already have a previous estimate of $A$ and $B$.

These re-estimations form the core of the iterative forward-backward algorithm. The forward-backward algorithm (Fig. A.14) starts with some initial estimate of the HMM parameters $\lambda = (A, B)$. We then iteratively run two steps. Like other cases of the EM (expectation-maximization) algorithm, the forward-backward algorithm has two steps: the **expectation** step, or **E-step**, and the **maximization** step, or **M-step**.

E-step

M-step

In the E-step, we compute the expected state occupancy count $\gamma$ and the expected state transition count $\xi$ from the earlier $A$ and $B$ probabilities. In the M-step, we use $\gamma$ and $\xi$ to recompute new $A$ and $B$ probabilities.

**function** FORWARD-BACKWARD(*observations* of len $T$, *output vocabulary* $V$, *hidden state set* $Q$) **returns** *HMM=(A,B)*

**initialize** $A$ and $B$
**iterate** until convergence
**E-step**

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall\, t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \quad \forall\, t,\ i, \text{ and } j$$

**M-step**

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

**return** $A$, $B$

**Figure A.14** The forward-backward algorithm.

Although in principle the forward-backward algorithm can do completely unsupervised learning of the $A$ and $B$ parameters, in practice the initial conditions are very important. For this reason the algorithm is often given extra information. For example, for HMM-based speech recognition, the HMM structure is often set by hand, and only the emission ($B$) and (non-zero) $A$ transition probabilities are trained from a set of observation sequences $O$.

## A.6 Summary

This chapter introduced the **hidden Markov model** for probabilistic **sequence classification**.

- Hidden Markov models (**HMMs**) are a way of relating a sequence of **observations** to a sequence of **hidden classes** or **hidden states** that explain the observations.
- The process of discovering the sequence of hidden states, given the sequence of observations, is known as **decoding** or **inference**. The **Viterbi** algorithm is commonly used for decoding.
- The parameters of an HMM are the $A$ transition probability matrix and the $B$ observation likelihood matrix. Both can be trained with the **Baum-Welch** or **forward-backward** algorithm.

## Bibliographical and Historical Notes

As we discussed in Chapter 8, Markov chains were first used by Markov (1913) (translation Markov 2006), to predict whether an upcoming letter in Pushkin's *Eugene Onegin* would be a vowel or a consonant. The hidden Markov model was developed by Baum and colleagues at the Institute for Defense Analyses in Princeton (Baum and Petrie 1966, Baum and Eagon 1967).

The **Viterbi** algorithm was first applied to speech and language processing in the context of speech recognition by Vintsyuk (1968) but has what Kruskal (1983) calls a "remarkable history of multiple independent discovery and publication". Kruskal and others give at least the following independently-discovered variants of the algorithm published in four separate fields:

| Citation | Field |
|---|---|
| Viterbi (1967) | information theory |
| Vintsyuk (1968) | speech processing |
| Needleman and Wunsch (1970) | molecular biology |
| Sakoe and Chiba (1971) | speech processing |
| Sankoff (1972) | molecular biology |
| Reichert et al. (1973) | molecular biology |
| Wagner and Fischer (1974) | computer science |

The use of the term **Viterbi** is now standard for the application of dynamic programming to any kind of probabilistic maximization problem in speech and language processing. For non-probabilistic problems (such as for minimum edit distance), the plain term **dynamic programming** is often used. Forney, Jr. (1973) wrote an early survey paper that explores the origin of the Viterbi algorithm in the context of information and communications theory.

Our presentation of the idea that hidden Markov models should be characterized by three fundamental problems was modeled after an influential tutorial by Rabiner (1989), which was itself based on tutorials by Jack Ferguson of IDA in the 1960s. Jelinek (1997) and Rabiner and Juang (1993) give very complete descriptions of the forward-backward algorithm as applied to the speech recognition problem. Jelinek (1997) also shows the relationship between forward-backward and EM.

Baum, L. E. 1972. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities III: Proceedings of the 3rd Symposium on Inequalities*. Academic Press.

Baum, L. E. and J. A. Eagon. 1967. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73(3):360–363.

Baum, L. E. and T. Petrie. 1966. Statistical inference for probabilistic functions of finite-state Markov chains. *Annals of Mathematical Statistics*, 37(6):1554–1563.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the *EM* algorithm. *Journal of the Royal Statistical Society*, 39(1):1–21.

Eisner, J. 2002. An interactive spreadsheet for teaching the forward-backward algorithm. *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL*.

Forney, Jr., G. D. 1973. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

Jelinek, F. 1997. *Statistical Methods for Speech Recognition*. MIT Press.

Kruskal, J. B. 1983. An overview of sequence comparison. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 1–44. Addison-Wesley.

Markov, A. A. 1913. Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain ('Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain'). *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*, 7:153–162.

Markov, A. A. 2006. Classical text in translation: A. A. Markov, an example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains. *Science in Context*, 19(4):591–600. Translated by David Link.

Needleman, S. B. and C. D. Wunsch. 1970. A general method applicable to the search for similarities in the amino-acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453.

Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Rabiner, L. R. and B. H. Juang. 1993. *Fundamentals of Speech Recognition*. Prentice Hall.

Reichert, T. A., D. N. Cohen, and A. K. C. Wong. 1973. An application of information theory to genetic mutations and the matching of polypeptide sequences. *Journal of Theoretical Biology*, 42:245–261.

Sakoe, H. and S. Chiba. 1971. A dynamic programming approach to continuous speech recognition. *Proceedings of the Seventh International Congress on Acoustics*, volume 3. Akadémiai Kiadó.

Sankoff, D. 1972. Matching sequences under deletion-insertion constraints. *Proceedings of the National Academy of Sciences*, 69:4–6.

Vintsyuk, T. K. 1968. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57. Russian Kibernetika 4(1):81-88. 1968.

Viterbi, A. J. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2):260–269.

Wagner, R. A. and M. J. Fischer. 1974. The string-to-string correction problem. *Journal of the ACM*, 21:168–173.