CHAPTER 8 Hidden Markov Models

Her sister was called Tatiana.
For the first time with such a name
the tender pages of a novel,
we'll whimsically grace.
Pushkin, *Eugene Onegin*, in the Nabokov translation

Alexander Pushkin's novel in verse, *Eugene Onegin*, serialized in the early 19th century, tells of the young dandy Onegin, his rejection of the love of young Tatiana, his duel with his friend Lenski, and his later regret for both mistakes. But the novel is mainly beloved for its style and structure rather than its plot. Among other interesting structural innovations, the novel is written in a form now known as the *Onegin stanza*, iambic tetrameter with an unusual rhyme scheme. These elements have caused complications and controversy in its translation into other languages. Many of the translations have been in verse, but Nabokov famously translated it strictly literally into English prose. The issue of its translation and the tension between literal and verse translations have inspired much commentary—see, for example, Hofstadter (1997).

In 1913, A. A. Markov asked a less controversial question about Pushkin's text: could we use frequency counts from the text to help compute the probability that the next letter in sequence would be a vowel? In this chapter we introduce a descendant of Markov's model that is a key model for language processing, the **hidden Markov model** or **HMM**.

sequence model

The HMM is a **sequence model**. A sequence model or **sequence classifier** is a model whose job is to assign a label or class to each unit in a sequence, thus mapping a sequence of observations to a sequence of labels. An HMM is a probabilistic sequence model: given a sequence of units (words, letters, morphemes, sentences, whatever), they compute a probability distribution over possible sequences of labels and choose the best label sequence.

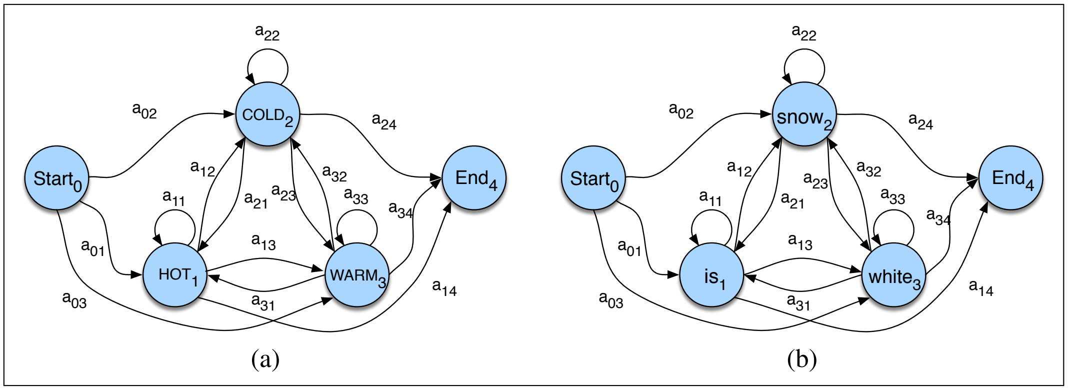
Sequence labeling tasks come up throughout speech and language processing, a fact that isn't too surprising if we consider that language consists of sequences at many representational levels. These include part-of-speech tagging (Chapter 8) named entity tagging (Chapter 18), and speech recognition (Chapter 25) among others.

In this chapter we present the mathematics of the HMM, beginning with the Markov chain and then including the main three constituent algorithms: the **Viterbi** algorithm, the **Forward** algorithm, and the **Baum-Welch** or EM algorithm for unsupervised (or semi-supervised) learning. In the following chapter we'll see the HMM applied to the task of part-of-speech tagging.

8.1 Markov Chains

Markov chain

The hidden Markov model is one of the most important machine learning models in speech and language processing. To define it properly, we need to first introduce the **Markov chain**, sometimes called the **observed Markov model**. Markov chains and hidden Markov models are both extensions of the finite automata of Chapter 3. Recall that a **weighted finite automaton** is defined by a set of states and a set of transitions between states, with each arc is associated with a weight. A **Markov chain** is a special case of a weighted automaton in which weights are probabilities (the probabilities on all arcs leaving a node must sum to 1) and in whicih the input sequence uniquely determines which states the automaton will go through. Because it can't represent inherently ambiguous problems, a Markov chain is only useful for assigning probabilities to unambiguous sequences.



(a) (b)

Figure 8.1 A Markov chain for weather (a) and one for words (b). A Markov chain is specified by the structure, the transition between states, and the start and end states.

Figure 8.1a shows a Markov chain for assigning a probability to a sequence of weather events, for which the vocabulary consists of HOT, COLD, and WARM. Figure 8.1b shows another simple example of 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. Given the two models in Fig. 8.1, we can assign a probability to any sequence from our vocabulary. We go over how to do this shortly.

First, let's be more formal and view a Markov chain as a kind of probabilistic **graphical model**: a way of representing probabilistic assumptions in a graph. A Markov chain is specified by the following components:

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{01} a_{02} \ldots a_{n1} \ldots a_{nn}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
q_0, q_F	a special **start state** and **end (final) state** that are not associated with observations

Figure 8.1 shows that we represent the states (including start and end states) as nodes in the graph, and the transitions as edges between nodes.

First-order Markov chain

A Markov chain embodies an important assumption about these probabilities. In a **first-order** Markov chain, the probability of a particular state depends only on the previous state:

$$\textbf{Markov Assumption:} \quad P(q_i|q_1...q_{i-1}) = P(q_i|q_{i-1}) \tag{8.1}$$

Note that because each a_{ij} expresses the probability $p(q_j|q_i)$, the laws of probability require that the values of the outgoing arcs from a given state must sum to 1:

$$\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i \tag{8.2}$$

An alternative representation that is sometimes used for Markov chains doesn't rely on a start or end state, instead representing the distribution over initial states and accepting states explicitly:

$\pi = \pi_1, \pi_2, ..., \pi_N$	an **initial probability distribution** over states. π_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, $\sum_{i=1}^{n} \pi_i = 1$
$QA = \{q_x, q_y...\}$	a set $QA \subset Q$ of legal **accepting states**

Thus, the probability of state 1 being the first state can be represented either as a_{01} or as π_1. Note that because each π_i expresses the probability $p(q_i|START)$, all the π probabilities must sum to 1:

$$\sum_{i=1}^{n} \pi_i = 1 \tag{8.3}$$

Before you go on, use the sample probabilities in Fig. 8.2b to compute the probability of each of the following sequences:

(8.4) hot hot hot hot

(8.5) cold hot cold hot

What does the difference in these probabilities tell you about a real-world weather fact encoded in Fig. 8.2b?

8.2 The Hidden Markov Model

A Markov chain is useful when we need to compute a probability for a sequence of events that we can observe in the world. In many cases, however, the events we are interested in may not be directly observable in the world. For example, in part-of-speech tagging (Chapter 8) we didn't observe part-of-speech tags in the world; we saw words and had to infer the correct tags from the word sequence. We call the part-of-speech tags **hidden** because they are not observed. The same architecture comes up in speech recognition; in that case we see acoustic events in the world and have to infer the presence of "hidden" words that are the underlying causal source of the acoustics. 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.

Hidden Markov model

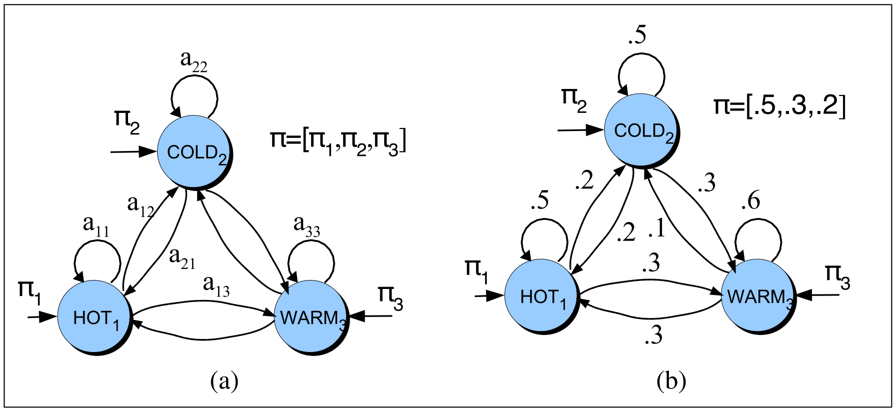


Figure 8.2 Another representation of the same Markov chain for weather shown in Fig. 8.1. Instead of using a special start state with a_{01} transition probabilities, we use the π vector, which represents the distribution over starting state probabilities. The figure in (b) shows sample probabilities.

To exemplify these models, we'll use a task conceived of 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 2007, 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 observation an integer corresponding to the number of ice creams eaten on a given day, figure out the correct 'hidden' sequence Q of weather states (H or C) which caused Jason to eat the ice cream.

Let's begin with a formal definition of a hidden Markov model, focusing on how it differs from a Markov chain. An HMM is specified by the following components:

$Q = q_1 q_2 \dots q_N$	a set of N **states**
$A = a_{11} a_{12} \dots a_{n1} \dots a_{nn}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \dots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, ..., v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i
q_0, q_F	a special **start state** and **end (final) state** that are not associated with observations, together with transition probabilities $a_{01} a_{02} \dots a_{0n}$ out of the start state and $a_{1F} a_{2F} \dots a_{nF}$ into the end state

As we noted for Markov chains, an alternative representation that is sometimes used for HMMs doesn't rely on a start or end state, instead representing the distribution over initial and accepting states explicitly. We don't use the π notation in this textbook, but you may see it in the literature[1]:

$\pi = \pi_1, \pi_2, ..., \pi_N$ an **initial probability distribution** over states. π_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, $\sum_{i=1}^{n} \pi_i = 1$

$QA = \{q_x, q_y...\}$ a set $QA \subset Q$ of legal **accepting states**

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:

$$\textbf{Markov Assumption:} \quad P(q_i|q_1...q_{i-1}) = P(q_i|q_{i-1}) \tag{8.6}$$

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:

$$\textbf{Output Independence:} \quad P(o_i|q_1 \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i|q_i) \tag{8.7}$$

Figure 8.3 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.

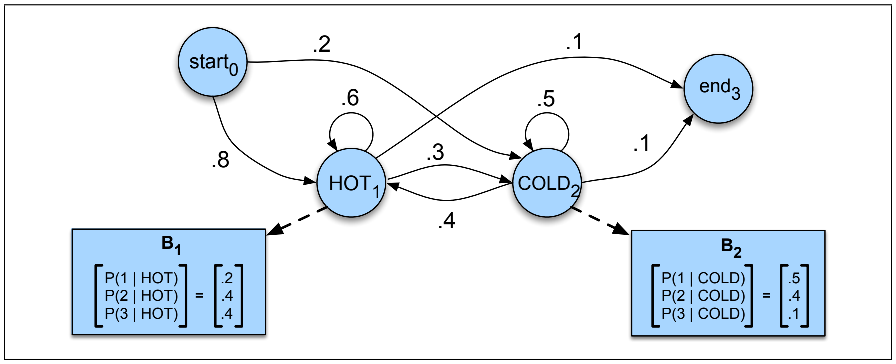

start0
.2
.1
end3
.6
.5
.8
.3
.1
HOT1
COLD2
.4
B1
P(1 | HOT) P(2 | HOT) P(3 | HOT) = .2 .4 .4
B2
P(1 | COLD) P(2 | COLD) P(3 | COLD) = .5 .4 .1

Figure 8.3 A hidden Markov model for relating numbers of ice creams eaten by Jason (the observations) to the weather (H or C, the hidden variables).

Ergodic HMM

Bakis network

Notice that in the HMM in Fig. 8.3, there is a (non-zero) probability of transitioning between any two states. Such an HMM is called a **fully connected** or **ergodic HMM**. Sometimes, however, we have HMMs in which many of the transitions between states have zero probability. For example, in **left-to-right** (also called **Bakis**) HMMs, the state transitions proceed from left to right, as shown in Fig. 8.4. In a Bakis HMM, no transitions go from a higher-numbered state to a lower-numbered state (or, more accurately, any transitions from a higher-numbered state to a lower-numbered state have zero probability). Bakis HMMs are generally used to model temporal processes like speech; we show more of them in Chapter 25.

[1] It is also possible to have HMMs without final states or explicit accepting states. Such HMMs define a set of probability distributions, one distribution per observation sequence length, just as language models do when they don't have explicit end symbols. This isn't a problem since for most tasks in speech and language processing the lengths of the observations are fixed.

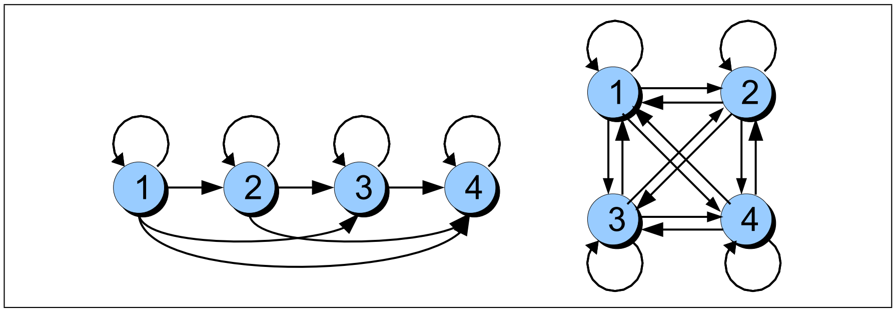


Figure 8.4 Two 4-state hidden Markov models; a left-to-right (Bakis) HMM on the left and a fully connected (ergodic) HMM on the right. In the Bakis model, all transitions not shown have zero probability.

Now that we have seen the structure of an HMM, we turn to algorithms for computing things with them. 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 three sections we introduce all three problems more formally.

8.3 Likelihood Computation: The Forward Algorithm

Our first problem is to compute the likelihood of a particular observation sequence. For example, given the HMM in Fig. 8.3, 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.[2]

Given this one-to-one mapping and the Markov assumptions expressed in Eq. 8.6, 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

$$P(O|Q) = \prod_{i=1}^{T} P(o_i|q_i) \tag{8.8}$$

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. 8.9. Figure 8.5 shows a graphic representation of this computation.

$$P(3\ 1\ 3|\text{hot hot cold}) \quad = \quad P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \tag{8.9}$$

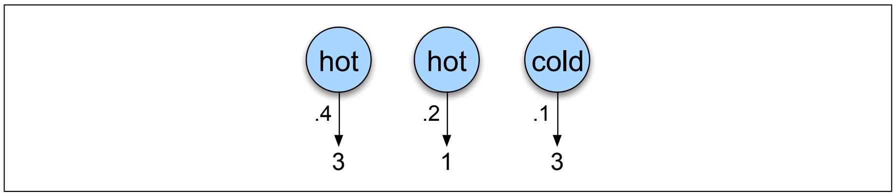


Figure 8.5 The computation of the observation likelihood for the ice-cream events *3 1 3* given the hidden state sequence *hot hot cold*.

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 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

$$P(O,Q) = P(O|Q) \times P(Q) = \prod_{i=1}^{n} P(o_i|q_i) \times \prod_{i=1}^{n} P(q_i|q_{i-1}) \tag{8.10}$$

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. 8.11. Figure 8.6 shows a graphic representation of this computation.

$$\begin{aligned} P(3\ 1\ 3, \text{hot hot cold}) \quad &= \quad P(\text{hot}|\text{start}) \times P(\text{hot}|\text{hot}) \times P(\text{cold}|\text{hot}) \\ &\quad \times P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \end{aligned} \tag{8.11}$$

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:

$$P(O) = \sum_{Q} P(O,Q) = \sum_{Q} P(O|Q)P(Q) \tag{8.12}$$

[2] 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.

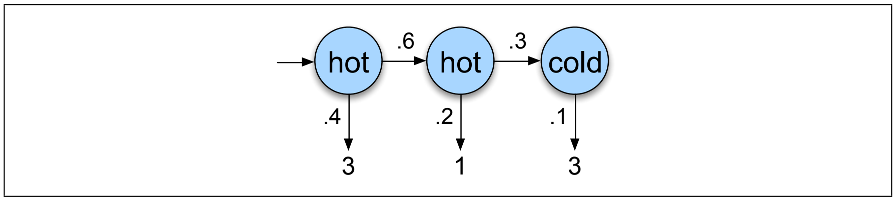


Figure 8.6 The computation of the joint probability of the ice-cream events *3 1 3* and the hidden state sequence *hot hot cold*.

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

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 λ. 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:

$$\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda) \tag{8.13}$$

Here, $q_t = j$ means "the tth 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

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t) \tag{8.14}$$

The three factors that are multiplied in Eq. 8.14 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

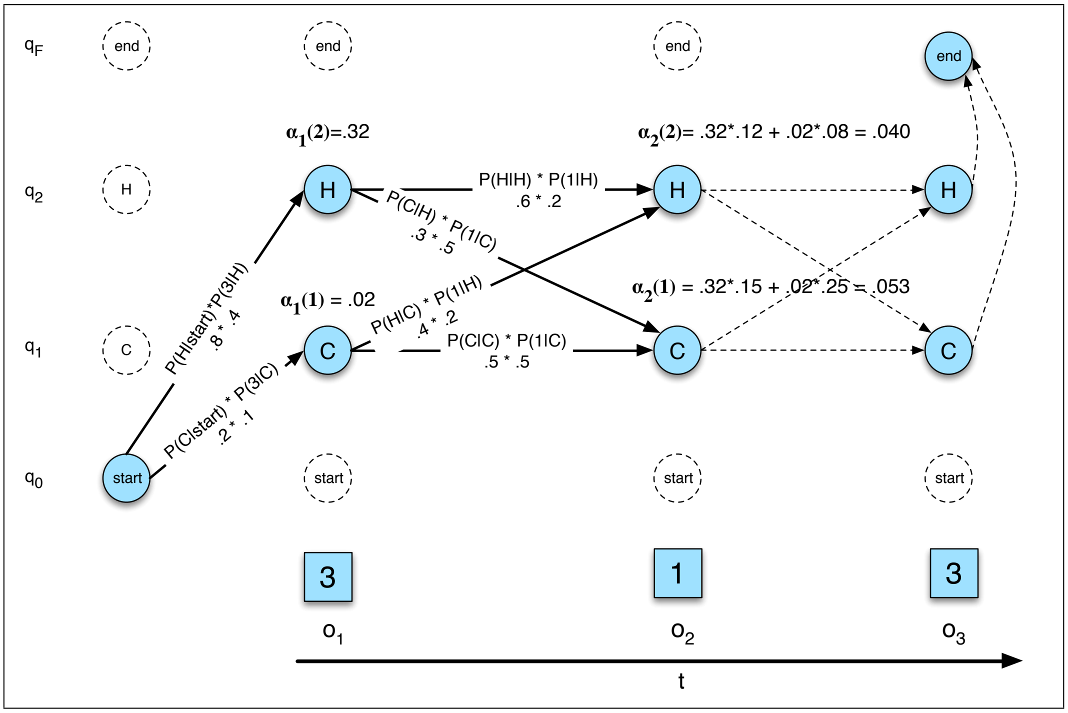


Figure 8.7 The forward trellis for computing the total observation likelihood for the ice-cream events *3 1 3*. Hidden states are in circles, observations in squares. White (unfilled) circles indicate illegal transitions. The figure shows the computation of $\alpha_t(j)$ for two states at two time steps. The computation in each cell follows Eq. 8.14: $\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i)a_{ij}b_j(o_t)$. The resulting probability expressed in each cell is Eq. 8.13: $\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda)$.

Consider the computation in Fig. 8.7 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 α probabilities from time step 1, via two paths, each extension consisting of the three factors above: $\alpha_1(1) \times P(H|H) \times P(1|H)$ and $\alpha_1(2) \times P(H|C) \times P(1|H)$.

Figure 8.8 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. 8.9 and a statement of the definitional recursion here.

1. Initialization:

$$\alpha_1(j) \;=\; a_{0j}b_j(o_1) \quad 1 \le j \le N \tag{8.15}$$

2. Recursion (since states 0 and F are non-emitting):

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i)a_{ij}b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \tag{8.16}$$

3. Termination:

$$P(O|\lambda) = \alpha_T(q_F) = \sum_{i=1}^{N} \alpha_T(i)\, a_{iF} \tag{8.17}$$

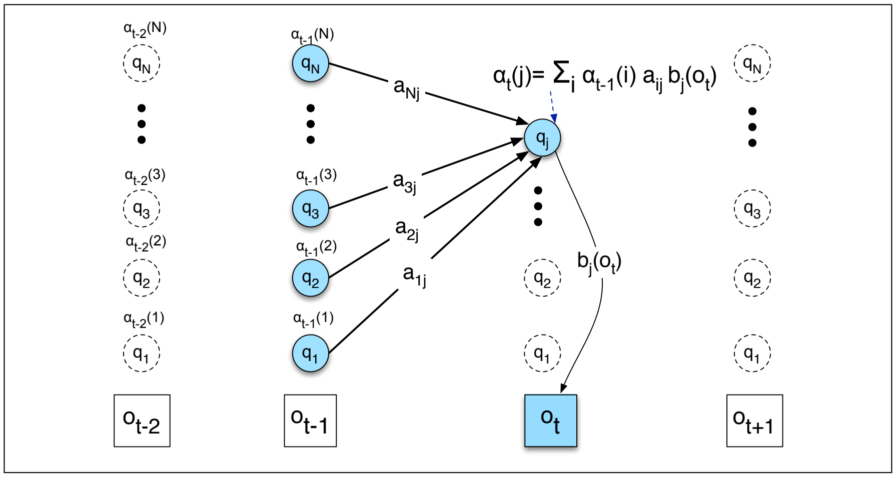


Figure 8.8 Visualizing the computation of a single element $\alpha_t(i)$ in the trellis by summing all the previous values α_{t-1}, weighted by their transition probabilities a, and multiplying by the observation probability $b_i(o_{t+1})$. For many applications of HMMs, many of the transition probabilities 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 $\alpha_t(i)$. Start and end states are not shown.

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

  create a probability matrix forward[N+2,T]
  for each state s from 1 to N do                              ; initialization step
      $forward[s,1] \leftarrow a_{0,s} \ast b_s(o_1)$
  for each time step t from 2 to T do                          ; recursion step
    for each state s from 1 to N do
               $forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] \ast a_{s',s} \ast b_s(o_t)$
  $forward[q_F,T] \leftarrow \sum_{s=1}^{N} forward[s,T] \ast a_{s,q_F}$          ; termination step
 return $forward[q_F,T]$
```

Figure 8.9 The forward algorithm. We've used the notation *forward*$[s,t]$ to represent $\alpha_t(s)$.

8.4 Decoding: The Viterbi Algorithm

Decoding

Decoder

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 $Q = q_1 q_2 q_3 \dots q_T$.

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 3.

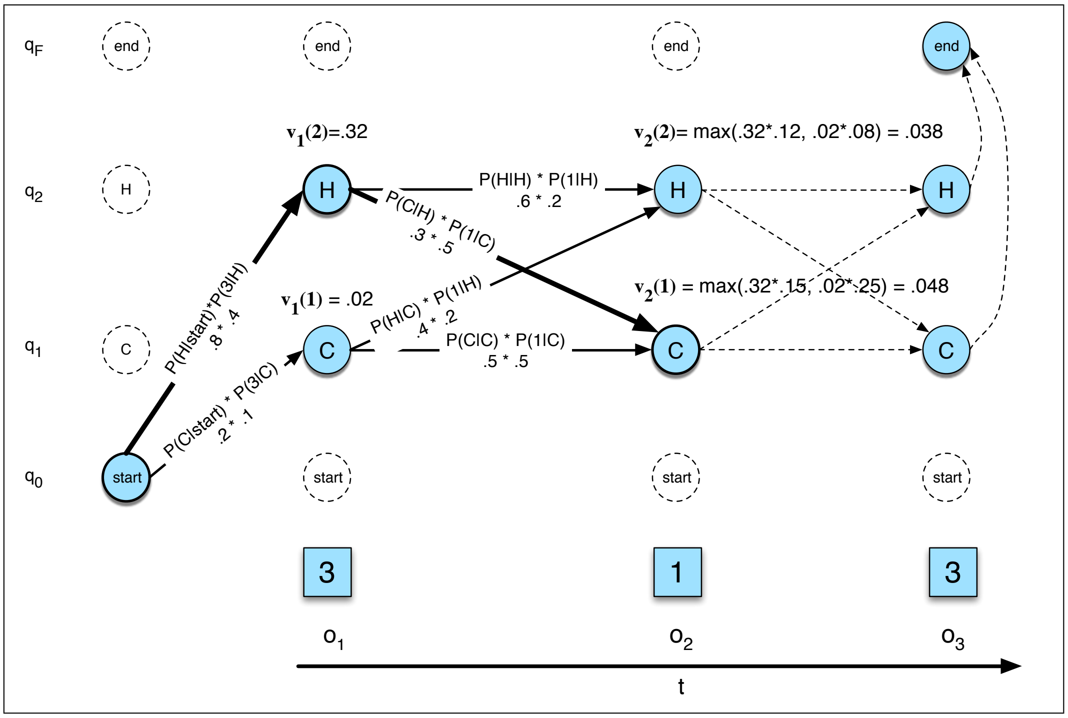


Figure 8.10 The Viterbi trellis for computing the best path through the hidden state space for the ice-cream eating events *3 1 3*. Hidden states are in circles, observations in squares. White (unfilled) circles indicate illegal transitions. The figure shows the computation of $v_t(j)$ for two states at two time steps. The computation in each cell follows Eq. 8.19: $v_t(j) = \max_{1\leq i\leq N-1} v_{t-1}(i)\ a_{ij}\ b_j(o_t)$. The resulting probability expressed in each cell is Eq. 8.18: $v_t(j) = P(q_0, q_1, \ldots, q_{t-1}, o_1, o_2, \ldots, o_t, q_t = j|\lambda)$.

Figure 8.10 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 $q_0, q_1, ..., q_{t-1}$, given the automaton λ. 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

$$v_t(j) = \max_{q_0,q_1,\ldots,q_{t-1}} P(q_0, q_1 ... q_{t-1}, o_1, o_2 \ldots o_t, q_t = j|\lambda) \qquad (8.18)$$

Note that we represent the most probable path by taking the maximum over all possible previous state sequences $\max_{q_0,q_1,\ldots,q_{t-1}}$. 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

$$v_t(j) \;=\; \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t) \tag{8.19}$$

The three factors that are multiplied in Eq. 8.19 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

  create a path probability matrix viterbi[N+2,T]
  for each state s from 1 to N do                          ; initialization step
      $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
      $backpointer[s,1] \leftarrow 0$
  for each time step t from 2 to T do                      ; recursion step
    for each state s from 1 to N do
      $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
      $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$
  $viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$          ; termination step
  $backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$     ; termination step
  return the backtrace path by following backpointers to states back in
          time from $backpointer[q_F,T]$
```

Figure 8.11 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. Note that states 0 and q_F are non-emitting.

Figure 8.11 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. 8.12, and then at the end backtracing the best path to the beginning (the Viterbi **backtrace**).

Viterbi backtrace

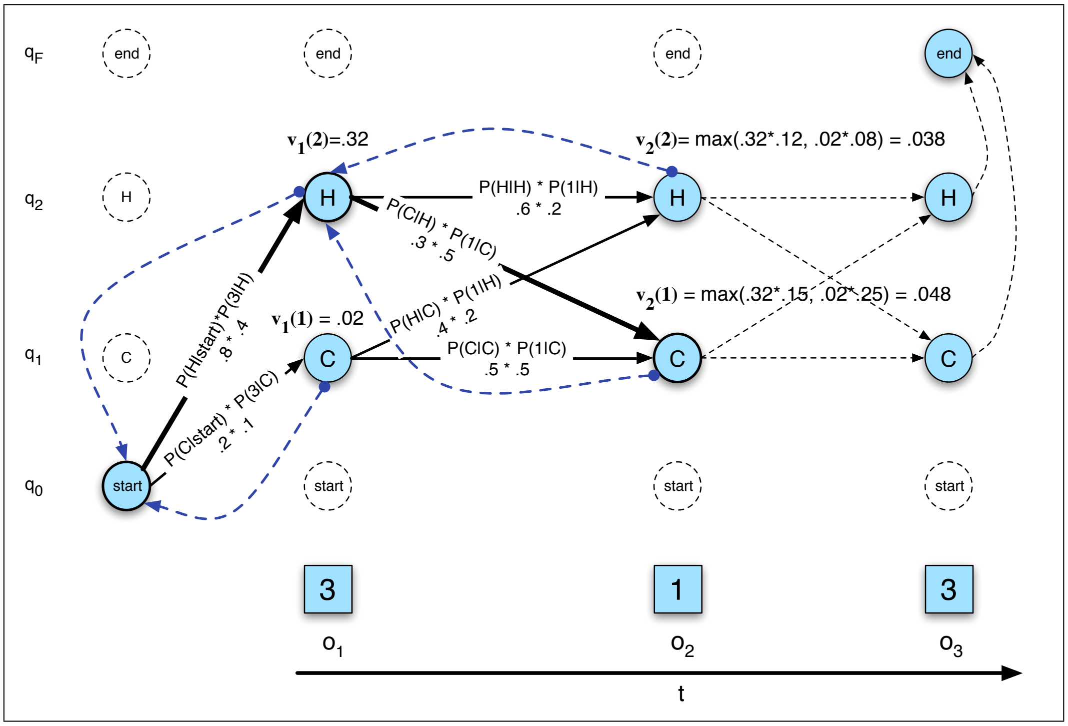


Figure 8.12 The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

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

1. **Initialization:**

$$v_1(j) = a_{0j}b_j(o_1) \quad 1 \leq j \leq N \tag{8.20}$$

$$bt_1(j) = 0 \tag{8.21}$$

2. **Recursion** (recall that states 0 and q_F are non-emitting):

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\,a_{ij}\,b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T \tag{8.22}$$

$$bt_t(j) = \operatorname{argmax}_{i=1}^{N} v_{t-1}(i)\,a_{ij}\,b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T \tag{8.23}$$

3. **Termination:**

$$\text{The best score:} \quad P* = v_T(q_F) = \max_{i=1}^{N} v_T(i) * a_{iF} \tag{8.24}$$

$$\text{The start of backtrace:} \quad q_T* = bt_T(q_F) = \operatorname{argmax}_{i=1}^{N} v_T(i) * a_{iF} \tag{8.25}$$

8.5 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. For the part-of-speech tagging task, we would start with a sequence of observations $O = \{w_1, w_2, w_3 \dots\}$ and a set of hidden states *NN, NNS, VBD, IN,...* and so on.

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.

Let us begin by considering the much simpler case of training a Markov chain rather than a hidden Markov model. Since the states in a Markov chain are observed, we can run the model on the observation sequence and directly see which path we took through the model and which state generated each observation symbol. A Markov chain of course has no emission probabilities B (alternatively, we could view a Markov chain as a degenerate hidden Markov model where all the b probabilities are 1.0 for the observed symbol and 0 for all other symbols). Thus, the only probabilities we need to train are the transition probability matrix A.

We get the maximum likelihood estimate of the probability a_{ij} of a particular transition between states i and j by counting the number of times the transition was taken, which we could call $C(i \to j)$, and then normalizing by the total count of all times we took any transition from state i:

$$a_{ij} = \frac{C(i \to j)}{\sum_{q \in Q} C(i \to q)} \tag{8.26}$$

We can directly compute this probability in a Markov chain because we know which states we were in. For an 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. The Baum-Welch algorithm uses two neat intuitions to solve this problem. The first idea is to *iteratively* estimate 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. The second idea is that we get our estimated probabilities 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 probability β 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 λ):

$$\beta_t(i) = P(o_{t+1}, o_{t+2} \dots o_T | q_t = i, \lambda) \tag{8.27}$$

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

1. **Initialization:**

$$\beta_T(i) \;=\; a_{iF}, \quad 1 \leq i \leq N \tag{8.28}$$

2. **Recursion** (again since states 0 and q_F are non-emitting):

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j), \quad 1 \leq i \leq N, 1 \leq t < T \tag{8.29}$$

3. **Termination:**

$$P(O|\lambda) = \alpha_T(q_F) = \beta_1(q_0) = \sum_{j=1}^{N} a_{0j}\, b_j(o_1)\, \beta_1(j) \tag{8.30}$$

Figure 8.13 illustrates the backward induction step.

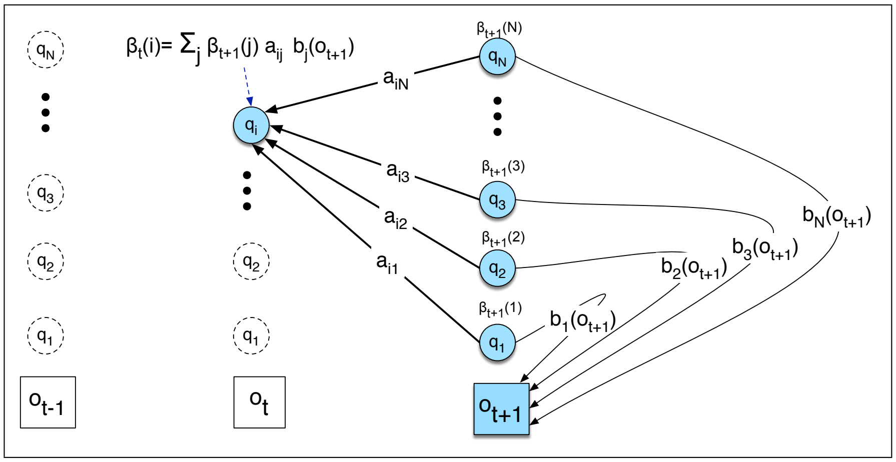


Figure 8.13 The computation of $\beta_t(i)$ by summing all the successive values $\beta_{t+1}(j)$ weighted by their transition probabilities a_{ij} and their observation probabilities $b_j(o_{t+1})$. Start and end states not shown.

We are now ready to understand how the forward and backward probabilities can help us 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 machine is hidden.

Let's begin by seeing how to estimate $\hat{a}_{ij}$ by a variant of Eq. 8.26:

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} \tag{8.31}$$

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 ξ_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:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j|O,\lambda) \tag{8.32}$$

To compute ξ_t, we first compute a probability which is similar to ξ_t, but differs in including the probability of the observation; note the different conditioning of O from Eq. 8.32:

$$\text{not-quite-}\xi_t(i,j) = P(q_t = i, q_{t+1} = j, O|\lambda) \tag{8.33}$$

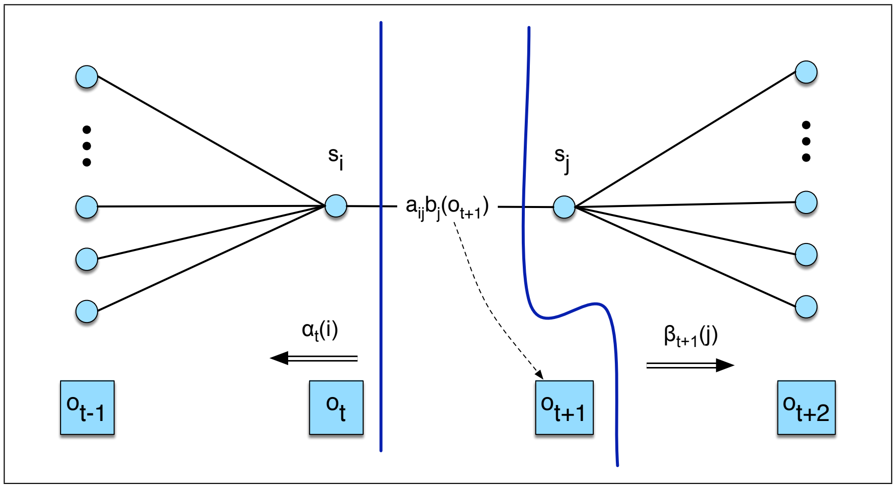


Figure 8.14 Computation of the joint probability of being in state i at time t and state j at time $t+1$. The figure shows the various probabilities that need to be combined to produce $P(q_t = i, q_{t+1} = j, O|\lambda)$: the α and β probabilities, the transition probability a_{ij} and the observation probability $b_j(o_{t+1})$. After Rabiner (1989) which is ©1989 IEEE.

Figure 8.14 shows the various probabilities that go into computing not-quite-ξ_t: the transition probability for the arc in question, the α probability before the arc, the β probability after the arc, and the observation probability for the symbol just after the arc. These four are multiplied together to produce *not-quite-*ξ_t as follows:

$$\text{not-quite-}\xi_t(i,j) = \alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j) \tag{8.34}$$

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

$$P(X|Y,Z) = \frac{P(X,Y|Z)}{P(Y|Z)} \tag{8.35}$$

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), which can thus be computed in a number of ways:

$$P(O|\lambda) = \alpha_T(q_F) = \beta_T(q_0) = \sum_{j=1}^{N} \alpha_t(j)\beta_t(j) \tag{8.36}$$

So, the final equation for ξ_t is

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \tag{8.37}$$

The expected number of transitions from state i to state j is then the sum over all t of ξ. For our estimate of a_{ij} in Eq. 8.31, 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 $\hat{a}_{ij}$:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)} \tag{8.38}$$

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

$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j} \tag{8.39}$$

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

$$\gamma_t(j) = P(q_t = j | O, \lambda) \tag{8.40}$$

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

$$\gamma_t(j) = \frac{P(q_t = j, O | \lambda)}{P(O|\lambda)} \tag{8.41}$$

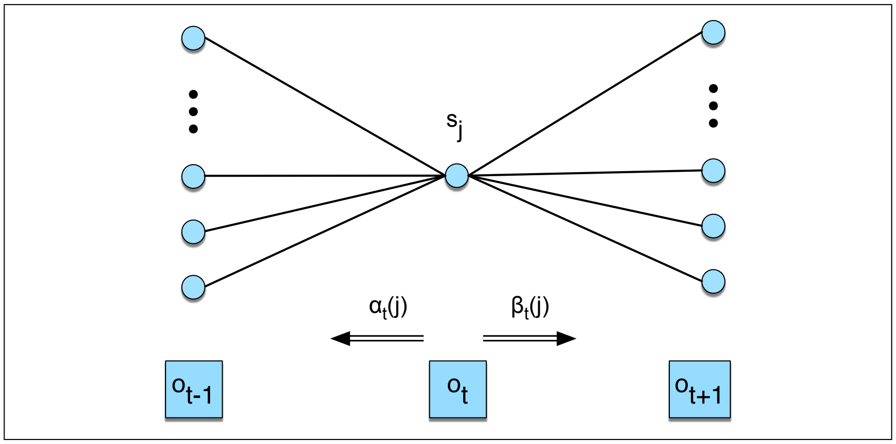


Figure 8.15 The computation of $\gamma_t(j)$, the probability of being in state j at time t. Note that γ is really a degenerate case of ξ and hence this figure is like a version of Fig. 8.14 with state i collapsed with state j. After Rabiner (1989) which is ©1989 IEEE.

As Fig. 8.15 shows, the numerator of Eq. 8.41 is just the product of the forward probability and the backward probability:

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \tag{8.42}$$

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"):

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t. O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{8.43}$$

We now have ways in Eq. 8.38 and Eq. 8.43 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.

E-step

M-step

These re-estimations form the core of the iterative forward-backward algorithm. The forward-backward algorithm (Fig. 8.16) 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**.

In the E-step, we compute the expected state occupancy count γ and the expected state transition count ξ from the earlier A and B probabilities. In the M-step, we use γ and ξ 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 8.16 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 speech recognition, in practice 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. Section **??** in Chapter 25 also discusses how initial A and B estimates are derived in speech recognition. We also show that for speech the forward-backward algorithm can be extended to inputs that are non-discrete ("continuous observation densities").

8.6 Summary

This chapter described two important models for probabilistic **sequence classification**: the **hidden Markov model** and the **maximum entropy Markov model**. Both models are widely used throughout speech and language processing.

- 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 at the end of Chapter 4, Markov chains were first used by Markov (1913, 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".[3] 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.

[3] Seven is pretty remarkable, but see page **??** for a discussion of the prevalence of multiple discovery.

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. See also the description of HMMs in other textbooks such as Manning and Schütze (1999).

Exercises

8.1 Implement the Forward algorithm and run it with the HMM in Fig. 8.3 to compute the probability of the observation sequences *331122313* and *331123312*. Which is more likely?

8.2 Implement the Viterbi algorithm and run it with the HMM in Fig. 8.3 to compute the most likely weather sequences for each of the two observation sequences above, *331122313* and *331123312*.

8.3 Extend the HMM tagger you built in Exercise 8.**??** by adding the ability to make use of some unlabeled data in addition to your labeled training corpus. First acquire a large unlabeled (i.e., no part-of-speech tags) corpus. Next, implement the forward-backward training algorithm. Now start with the HMM parameters you trained on the training corpus in Exercise 8.**??**; call this model M_0. Run the forward-backward algorithm with these HMM parameters to label the unsupervised corpus. Now you have a new model M_1. Test the performance of M_1 on some held-out labeled data.

8.4 As a generalization of the previous homework, implement Jason Eisner's HMM tagging homework available from his webpage. His homework includes a corpus of weather and ice-cream observations, a corpus of English part-of-speech tags, and a very hand spreadsheet with exact numbers for the forward-backward algorithm that you can compare against.

Baum, L. E. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In Shisha, O. (Ed.), *Inequalities III: Proceedings of the 3rd Symposium on Inequalities*, University of California, Los Angeles, pp. 1–8. Academic Press.

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

Dempster, A. P., Laird, N. M., and Rubin, D. B. (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. In *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL*, pp. 10–18.

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

Hofstadter, D. R. (1997). *Le Ton beau de Marot*. Basic Books.

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

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

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

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 Wunsch, C. D. (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 Juang, B. H. (1993). *Fundamentals of Speech Recognition*. Prentice Hall.

Reichert, T. A., Cohen, D. N., and Wong, A. K. C. (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 Chiba, S. (1971). A dynamic programming approach to continuous speech recognition. In *Proceedings of the Seventh International Congress on Acoustics*, Budapest, Vol. 3, pp. 65–69. Akadémiai Kiadó.

Sankoff, D. (1972). Matching sequences under deletion-insertion constraints. *Proceedings of the Natural Academy of Sciences of the U.S.A.*, *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 Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the Association for Computing Machinery*, *21*, 168–173.