Hidden Markov Models: All the Glorious Gory Details

Noah A. Smith
Department of Computer Science
Johns Hopkins University
nasmith@cs.jhu.edu

18 October 2004

1 Introduction

Hidden Markov models (HMMs, hereafter) are relatively simple to understand as a generative process, and they are extremely useful in many applications. Unfortunately the algorithmic details are notoriously hard to understand and tedious to work out. I have been taught these details at least three or four times, by various people and in various ways. If I can get it, so can you. It just takes some patience and working through some examples. This tutorial is intended to get you to grasp the basics by going through an example. If it insults your intelligence, please don't hold it against me; I'm just going to try to explain to you what I wish I'd been told when I was first grappling with this stuff.

2 The HMM

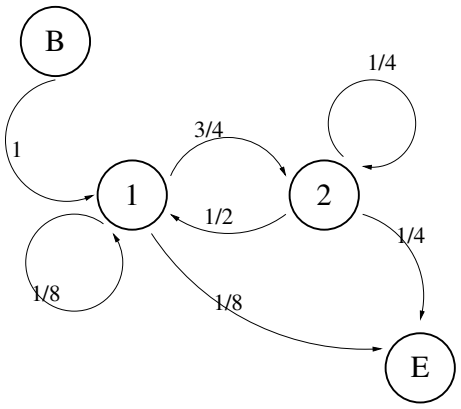
By now you should know that a HMM is a *probabilistic model* that assigns probabilities to sequences of symbols. It is a *generative* model, meaning that the probability distribution is defined by taking a series of steps that incrementally produce the sequence of symbols by making random choices. Think of an HMM as a machine that generates sequences.

How do HMMs generate sequences? An HMM consists of a (finite) set of states, which we'll call $\mathcal{S}$. In this discussion, there is a specific begin state $B \in \mathcal{S}$ and a specific end state $E \in \mathcal{S}$. The HMM always starts out in B and once it gets to E, it stops. In between, it makes random choices, walking from state to state. The choice to go from one state S to another state T is made randomly according to a specific distribution: S's *transition* distribution $\Pr(\bullet \mid S)$. The choice is made independently of all other choices.

Where do the output symbols come from? Each time we take a step, we arrive in a state. If that state is a silent state, we just decide (randomly, according to the state's transition distribution) where to go next. If the state is not silent, we pick an output symbol according to the state's *emission* distribution, output it as the next symbol in the sequence, and continue. B and E are silent states.

That's it; the generative process is very simple: Start in B. Pick the next state. Pick an output. Pick the next state. Pick an output. And so on ... until you get to state E, where you stop.

The HMM we will be working with is shown in Figure 1. We will call it $\mathcal{M}$ (for "model").



$$\begin{aligned}
\Pr(X \mid 1) &= \frac{7}{8} \\
\Pr(Y \mid 1) &= \frac{1}{8} \\
\Pr(X \mid 2) &= \frac{1}{16} \\
\Pr(Y \mid 2) &= \frac{15}{16}
\end{aligned}$$

Figure 1: The transition distributions (above) and emission distributions (below) for our HMM, $\mathcal{M}$.

3 Probabilities of Paths

Suppose you knew everything in Figure 1 (the number and names of the states, the transition probabilities, and the emission probabilities). Suppose I told you that the HMM had walked this sequence of states:

$$\vec{\pi} = B, 1, 1, 2, E$$

What is the probability that $\mathcal{M}$ would follow that state sequence (path)? More precisely, what is $\Pr(\pi \mid \mathcal{M})$?

Well, we know the HMM started in B.[1] I claim that $\mathcal{M}$ then stepped to state 1, then from 1 to 1 again, then from 1 to 2, then from 2 to E. Each of those steps involved a random choice being made according to a probability distribution. Each step depends only on the state we are in when we choose it. So we have:

$$\begin{aligned}
\Pr(\pi \mid \mathcal{M}) &= \overbrace{\Pr(\pi_0 = B)}^{\text{known to be 1}} \cdot \Pr(\pi_1 = 1 \mid \pi_0 = B) \cdot \Pr(\pi_2 = 1 \mid \pi_1 = 1) \\
&\quad \cdot \Pr(\pi_3 = 2 \mid \pi_2 = 1) \cdot \Pr(\pi_4 = E \mid \pi_3 = 2) \\
&= 1 \cdot 1 \cdot \frac{1}{8} \cdot \frac{3}{4} \cdot \frac{1}{4} \\
&= \frac{3}{2^7}
\end{aligned}$$

[1] If $\vec{\pi}$ had started with 1, what probability would we have assigned to $\vec{\pi}$? We would have had to assign zero, because we know all paths that are generated by $\mathcal{M}$ start with B.

path	computation of its probability with X, X, X, X	$\Pr(\text{path}, \vec{S})$
B1111E	$a_B(1) \cdot e_1(X) \cdot a_1(1) \cdot e_1(X) \cdot a_1(1) \cdot e_1(X) \cdot a_1(1) \cdot e_1(X) \cdot a_1(E)$	$2401/2^{24}$
B1112E	$a_B(1) \cdot e_1(X) \cdot a_1(1) \cdot e_1(X) \cdot a_1(1) \cdot e_1(X) \cdot a_1(2) \cdot e_2(X) \cdot a_2(E)$	$2058/2^{24}$
B1121E	$a_B(1) \cdot e_1(X) \cdot a_1(1) \cdot e_1(X) \cdot a_1(2) \cdot e_2(X) \cdot a_2(1) \cdot e_1(X) \cdot a_1(E)$	$4116/2^{24}$
B1122E	$a_B(1) \cdot e_1(X) \cdot a_1(1) \cdot e_1(X) \cdot a_1(2) \cdot e_2(X) \cdot a_2(2) \cdot e_2(X) \cdot a_2(E)$	$294/2^{24}$
B1211E	$a_B(1) \cdot e_1(X) \cdot a_1(2) \cdot e_2(X) \cdot a_2(1) \cdot e_1(X) \cdot a_1(1) \cdot e_1(X) \cdot a_1(E)$	$4116/2^{24}$
B1212E	$a_B(1) \cdot e_1(X) \cdot a_1(2) \cdot e_2(X) \cdot a_2(1) \cdot e_1(X) \cdot a_1(2) \cdot e_2(X) \cdot a_2(E)$	$3528/2^{24}$
B1221E	$a_B(1) \cdot e_1(X) \cdot a_1(2) \cdot e_2(X) \cdot a_2(2) \cdot e_2(X) \cdot a_2(1) \cdot e_1(X) \cdot a_1(E)$	$588/2^{24}$
B1222E	$a_B(1) \cdot e_1(X) \cdot a_1(2) \cdot e_2(X) \cdot a_2(2) \cdot e_2(X) \cdot a_2(2) \cdot e_2(X) \cdot a_2(E)$	$42/2^{24}$
B2111E	$a_B(2) \cdot e_2(X) \cdot a_2(1) \cdot e_1(X) \cdot a_1(1) \cdot e_1(X) \cdot a_1(1) \cdot e_1(X) \cdot a_1(E)$	0
B2112E	$a_B(2) \cdot e_2(X) \cdot a_2(1) \cdot e_1(X) \cdot a_1(1) \cdot e_1(X) \cdot a_1(2) \cdot e_2(X) \cdot a_2(E)$	0
B2121E	$a_B(2) \cdot e_2(X) \cdot a_2(1) \cdot e_1(X) \cdot a_1(2) \cdot e_2(X) \cdot a_2(1) \cdot e_1(X) \cdot a_1(E)$	0
B2122E	$a_B(2) \cdot e_2(X) \cdot a_2(1) \cdot e_1(X) \cdot a_1(2) \cdot e_2(X) \cdot a_2(2) \cdot e_2(X) \cdot a_2(E)$	0
B2211E	$a_B(2) \cdot e_2(X) \cdot a_2(2) \cdot e_2(X) \cdot a_2(1) \cdot e_1(X) \cdot a_1(1) \cdot e_1(X) \cdot a_1(E)$	0
B2212E	$a_B(2) \cdot e_2(X) \cdot a_2(2) \cdot e_2(X) \cdot a_2(1) \cdot e_1(X) \cdot a_1(2) \cdot e_2(X) \cdot a_2(E)$	0
B2221E	$a_B(2) \cdot e_2(X) \cdot a_2(2) \cdot e_2(X) \cdot a_2(2) \cdot e_2(X) \cdot a_2(1) \cdot e_1(X) \cdot a_1(E)$	0
B2222E	$a_B(2) \cdot e_2(X) \cdot a_2(2) \cdot e_2(X) \cdot a_2(2) \cdot e_2(X) \cdot a_2(2) \cdot e_2(X) \cdot a_2(E)$	0

Table 1: Brute force: The 16 paths that could generate X, X, X, X and their probabilities.

From here on we will use shorthand for the transition probabilities. Let

$$\Pr(\pi_t = A \mid \pi_{t-1} = B) = a_A(B) \tag{1}$$

Mistake here: should read a_B(A)

We will use this shorthand for emissions:

$$\Pr(S_t = X \mid \pi_t = A) = e_A(X) \tag{2}$$

It's very easy to compute the probability of any given path through the HMM when we know what the path is.

But suppose I gave you a sequence of symbols and told you it was generated by the HMM. Take

$$\vec{S} = X, X, X, X$$

I want to find the most probable path the HMM went through to generate that whole sequence. Before we get to the elegant solution, let's consider some naïve ways to find this path.

Brute force $\vec{S}$ is only four symbols long. Since those symbols (all of which happen to be X) could come out of either state 1 or state 2 (both states know how to generate X), we know there are $2^4 = 16$ possible paths. Table1 shows the probability of each of the paths, as computed by brute force. Notice that half of them turn out to have zero probability, because the first symbol can only come from state 1.[2]

We see from Table 1 that there are two most probable paths: $B, 1, 1, 2, 1, E$ and $B, 1, 2, 1, 1, E$. This was a painfully tedious computation, and if our sequence had been twice as long, it would have taken sixteen times as long to solve the problem this way. The brute force method is *correct*, but it is not efficient.

[2]Why? We have to start in B, and B can only transition to state 1.

Greedy guessing You might look at the sequence $\vec{S}$ and notice that it is all X. X is far more likely to come out of state 1 than state 2, so you might be tempted to say that the most likely path is $B, 1, 1, 1, 1, E$ (all X emitted by state 1). Table 1 should convince you that this is incorrect, even though it's really quick.

3.1 Best path: Viterbi algorithm

What we want is a method to find the best path that is both efficient *and* correct. The solution is the Viterbi algorithm.

Consider the following. I want to find the most probable path for the sequence $S_1, S_2, ..., S_t$ such that the path ends in some state s. I claim that if you can tell me the most probable path for $S_1, S_2, ..., S_{t-1}$ up to each state that transitions into s, then I can solve the problem.[3] Why?

To get to state s at time t, I must enter s from some other state k. I was at state k at time $t-1$. If I know the best path that ends in k at time $t-1$, then I can compute the probability of that path continuing to state s. I can then compare for the different states k (each one that transitions into s, and then I know the answer. This is the *optimal substructure* trick that dynamic programming gives us. Specifically, let $V(state, time)$ be the probability of the most likely path that covers the first *time* symbols in the emission sequence *and* ends in state *state*. Then:

$$V(s,t) = \overbrace{e_s(S_t)}^{s \text{ emits symbol } S_t} \max_{k:k \text{ transitions into } s} \overbrace{a_k(s)}^{\text{transition from } k \text{ to } s} \cdot V(k, t-1) \tag{3}$$

To make this a well-formed dynamic program, we need to give a base case (so that our equations ground out somewhere) and a goal value.

The base case is given by the fact that we *know* all productions of the HMM start in state B:

$$V(B, 0) = 1 \tag{4}$$

For all other states s not equal to B, we set $V(s, 0) = 0$; it is impossible to start in those states.

Our goal is to compute the best path ending in E, at the end of the sequence $\vec{S}$, or $t = n$: $V(E, n)$. There's one annoying little modification. If state s is silent, it won't emit a symbol, so we don't want to have the $e_s(S_t)$ term. Further, the emission sequence index won't increment when we move into a silent step, because nothing is emitted. So for silent states, we have:

$$V(s,t) = \max_{k:k \text{ transitions into } s} a_k(s) \cdot V(k, t-1) \tag{5}$$

These recurrence relations give us everything we need to figure out the probability of the best (most likely) path for $\vec{S}$. In order to actually get that path (which we'll call π^*), we need to do a little extra bookkeeping. Each time we pick a maximizing previous state (i.e., each time we compute a V value), we keep a pointer back to that maximizing state, at the given index. That way we can trace back the full path.

Let's work through the Viterbi algorithm for our example HMM, $\mathcal{M}$ and the sequence X, X, X, X. To help do this, we will set up a dynamic programming chart, just like you have seen for alignment problems. We will work left-to-right across the chart, computing the probabilities as we go, using the work we've already done to help us. Here is the initial chart, with the base cases (first column) filled in. The table on the left will hold V values, and the table on the right will hold the best preceding state for each cell.

[3] Ignore silent states for now.

	0	1: X	2: X	3: X	4: X
B	1				
1	0				
2	0				
E	0				

	1: X	2: X	3: X	4: X
B				
1				
2				
E				

We begin by computing $V(B, 1)$. This must be zero; there is no way to transition from B to any state other than 1. This means that $V(2, 1) = 0$ and also $V(E, 1) = 0$. Note also that we can never get into B again, so we will go ahead and set all $V(B, \bullet) = 0$.

	0	1: X	2: X	3: X	4: X
B	1	**0**	**0**	**0**	**0**
1	0				
2	0	**0**			
E	0	**0**			

	1: X	2: X	3: X	4: X
B	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
1				
2	$\emptyset$			
E	$\emptyset$			

Let's consider $V(1, 1)$. Looking in the 0 column, we see there is only one state with non-zero probability that could have led into state 1 at time 1, state B. The maximum, then, will be trivial, since there is only one option. Using the recurrence equation (3), we know that

$$V(1, 1) = e_1(X) \cdot a_B(1) \cdot V(B, 0) = \frac{7}{8} \cdot 1 \cdot 1 = \frac{7}{8} \tag{6}$$

We fill this in and note that the predecessor must be B:

	0	1: X	2: X	3: X	4: X
B	1	0	0	0	0
1	0	**7/8**			
2	0	0			
E	0	0			

	1: X	2: X	3: X	4: X
B	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
1	B			
2	$\emptyset$			
E	$\emptyset$			

Now we move to column 2 and compute $V(1, 2)$. Which states could transition into state 1 at this point in the sequence? Only state 1, because $\mathcal{M}$ could only have been in state 1 at the previous step. Again, things are simple and the maximum is trivial:

$$V(1, 2) = e_1(X) \cdot a_1(1) \cdot V(1, 1) = \frac{7}{8} \cdot \frac{1}{8} \cdot \frac{7}{8} = \frac{49}{2^9} \tag{7}$$

Consider $V(2, 2)$. Again, only state 1 could transition into state 2 at this point, because $\mathcal{M}$ could have only been in state 1 at the previous step.

$$V(2, 2) = e_2(X) \cdot a_1(2) \cdot V(1, 1) = \frac{1}{16} \cdot \frac{3}{4} \cdot \frac{7}{8} = \frac{21}{2^9} \tag{8}$$

	0	1: X	2: X	3: X	4: X
B	1	0	0	0	0
1	0	7/8	$\mathbf{49/2^9}$		
2	0	0	$\mathbf{21/2^9}$		
E	0	0			

	1: X	2: X	3: X	4: X
B	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
1	B	1		
2	$\emptyset$	1		
E	$\emptyset$			

Consider $V(E,2)$. Because E is a silent state, we don't look at the previous timestep (column); we look in the cells for non-silent states 1 and 2, in the *same* column.[4] If $\mathcal{M}$ had entered state E at timestep 2, it came from either state 1 or state 2. We have to pick the best one:

$$V(E,2) = \begin{cases} a_1(E) \cdot V(1,2) &= \frac{1}{8} \cdot \frac{49}{2^9} &= \frac{49}{2^{12}} \\ \\ a_2(E) \cdot V(2,2) &= \frac{1}{4} \cdot \frac{21}{2^9} &= \frac{21}{2^{11}} \end{cases} \tag{9}$$

State 1 is the winner:

	0	1: X	2: X	3: X	4: X
B	1	0	0	0	0
1	0	7/8	$49/2^9$		
2	0	0	$21/2^9$		
E	0	0	$\mathbf{49/2^{12}}$		

	1: X	2: X	3: X	4: X
B	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
1	B	1		
2	$\emptyset$	1		
E	$\emptyset$	1		

Continuing, we move to $V(1,3)$. Again we must make a choice: would it be better to get to state 1 at timestep 2, then stay there for timestep 3, or to get to state 2 at timestep 2, then jump to state 1 for timestep 3?

$$\begin{aligned} V(1,3) &= \overbrace{\frac{7}{8}}^{e_1(X)} \max \begin{cases} a_1(1) \cdot V(1,2) &= \frac{1}{8} \cdot \frac{49}{2^9} \\ \\ a_2(1) \cdot V(2,2) &= \frac{1}{2} \cdot \frac{21}{2^9} \leftarrow \end{cases} \\ &= \frac{147}{2^{13}} \end{aligned}$$

with the winner indicated by the arrow (preceding state 2). We update the chart:

	0	1: X	2: X	3: X	4: X
B	1	0	0	0	0
1	0	7/8	$49/2^9$	$\mathbf{147/2^{13}}$	
2	0	0	$21/2^9$		
E	0	0	$49/2^{12}$		

	1: X	2: X	3: X	4: X
B	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
1	B	1	2	
2	$\emptyset$	1		
E	$\emptyset$	1		

The remaining steps should be clear. They are given below with arrows marking the winners (I leave it to the reader to work through the arithmetic). Note that in one case we have a tie. The formulae are followed by the completed chart.

$$\begin{aligned} V(2,3) &= e_2(X) \max \begin{cases} a_1(2)V(1,2) \leftarrow \\ a_2(2)V(2,2) \end{cases} \\ V(E,3) &= \max \begin{cases} a_1(E)V(1,3) \leftarrow \\ a_2(E)V(2,3) \end{cases} \\ V(1,4) &= e_1(X) \max \begin{cases} a_1(1)V(1,3) \leftarrow \\ a_2(1)V(2,3) \leftarrow \end{cases} \end{aligned}$$

[4] You may notice that it is pointless to compute $V(E,2)$, since we know that once we enter state E we are done. $\mathcal{M}$ could not really have entered state E at timestep 2 when $\vec{S}$ was being generated, because more symbols got generated later. Bear with.

$$
\begin{aligned}
V(2,4) &= e_2(X) \max \left\{ \begin{array}{l} a_1(2)V(1,3) \leftarrow \\ a_2(2)V(2,3) \end{array} \right. \\
V(E,4) &= \max \left\{ \begin{array}{l} a_1(E)V(1,4) \leftarrow \\ a_2(E)V(2,4) \end{array} \right.
\end{aligned}
$$

	0	1: X	2: X	3: X	4: X
B	1	0	0	0	0
1	0	7/8	$49/2^9$	$147/2^{13}$	$1029/2^{19}$
2	0	0	$21/2^9$	$147/2^{15}$	$147/2^{18}$
E	0	0	$49/2^{12}$	$147/2^{16}$	$1029/2^{22}$

	1: X	2: X	3: X	4: X
B	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
1	B	1	2	1, 2
2	$\emptyset$	1	1	1
E	$\emptyset$	1	1	1

Note that the value we have for $V(E,4)$ is equivalent to the best value we saw in Table 1. The above was tedious, true. But notice that if we double the length of $\vec{S}$, then the computation time would just double; it wouldn't go up exponentially. The run-time of Viterbi is $O(|\mathcal{S}|^2 n)$, and the space requirement is $O(|\mathcal{S}|n)$.

Note also that the tie we saw in Table 1 is captured here, too. There are two best paths, with the same probability, and we can recover both of them by walking back through the righthand dynamic programming chart, starting in $(E,4)$. $(E,4)$ contains a 1, so we go to state 1 in the same column (because E is silent). There we have two choices: go to 1 or 2 in the preceding column (because 1 is not silent). Tracking each of those backward, we retrieve the same two paths that were found to be the best by brute force: $B,1,2,1,1,E$ and $B,1,1,2,1,E$.

3.2 Probability of a sequence: Forward algorithm

Now that you've been crunching numbers, you may have lost track of what's going on. Let's step up a level. What did we just do? We computed both π^*, the best path for sequence $\vec{S}$, and we computed $\Pr(\pi^*, \vec{S} \mid \mathcal{M})$. Another quantity that might be important is $\Pr(\vec{S} \mid \mathcal{M})$, the total probability that $\mathcal{M}$ would generate $\vec{S}$. As with finding the probability of the best path, there is a brute force way: compute the probability of every path that could generate $\vec{S}$, and sum them up, because

$$\Pr(\vec{S} \mid \mathcal{M}) = \sum_{\pi} \Pr(\pi, \vec{S} \mid \mathcal{M}) \tag{10}$$

To do that, we could sum up the values in the righthand column of Table 1, and we'd get $17143/2^{24} \approx 0.001$. This is of course inefficient.

It turns out that summing up over all possible paths doesn't require any more work than choosing the best among them (not surprising), *even* when we use dynamic programming (perhaps surprising). The Forward algorithm is just like the Viterbi algorithm, except that (a.) we don't do extra bookkeeping for the "best" path and (b.) we sum instead of maximize. Our goal now is to compute the forward probability $F(E,n)$, which is the probability of generating the sequence $S_1, ..., S_n$ and ending in E. The equations are:

$$F(s,t) = e_s(S_t) \sum_{k:k \text{ transitions into } s} a_k(s) \cdot F(k,t-1) \tag{11}$$

$$F(B,0) = 1 \tag{12}$$

$$F(s,0) = 0, \forall s \neq B \tag{13}$$

As before, there is a modified rule for silent states:

$$F(s,t) = \sum_{k:k \text{ transitions into } s} a_k(s) \cdot F(k,t-1) \tag{14}$$

I strongly urge you to compute $\Pr(X,X,X \mid \mathcal{S})$ using the Forward algorithm and convince yourself that the following chart is correct:

	0	1: X	2: X	3: X	4: X
B	1	0	0	0	0
1	0	7/8	$49/2^9$	$931/2^{15}$	$11221/2^{21}$
2	0	0	$21/2^9$	$21/2^{12}$	$2961/2^{21}$
E	0	0	$91/2^{12}$	$1267/2^{18}$	$17143/2^{24}$

3.3 Computing the same thing, another way: Backward algorithm

We could just as easily run a dynamic program in the opposite direction and get the same answer. The form of the Backward algorithm is slightly different.

In the Backward algorithm, we compute $\Pr(S_{i+1}, S_{i+2}, ..., S_n \mid \pi_i = k)$. That is, given that, at timestep i we are in state k, what is the probability of generating the rest of the sequence? This does not depend on the earlier part of the sequence at all. The dynamic program is given below.

$$\begin{aligned} B(s,t) = & \sum_{k:s \text{ transitions to } k \text{ and } k \text{ is not silent}} e_k(S_{t+1}) a_s(k) B(k,t+1) \\ & + \sum_{k:s \text{ transitions to } s \text{ and } k \textit{ is} \text{ silent}} a_s(k) B(k,t) \end{aligned} \tag{15}$$

We initialize by setting $B(E,n) = 1$. If we run the backward algorithm on our example, we have:

	0	1: X	2: X	3: X	4: X
B	$17143/2^{24}$	0	0	0	0
1	$33529/2^{27}$	$2449/2^{21}$	$181/2^{15}$	$13/2^9$	1/8
2	$37017/2^{26}$	$2731/2^{20}$	$197/2^{14}$	$15/2^8$	1/4
E	0	0	0	0	1

Again, you are encouraged to work through this one by hand and check your answers.

Interestingly, it appears that:

$$B(B,0) = F(E,n) = \Pr(\vec{S}|\mathcal{M}) \tag{16}$$

You should not be shocked. Note that:

$$\begin{aligned} B(B,0) &= \Pr(\vec{S} \mid \pi_0 = B) \text{ (by definition)} \\ &= \Pr(\vec{S}) \text{ (since we know } \mathcal{M} \text{ always starts in } B) \\ &= \Pr(\vec{S}, \pi \text{ ends in } E) \text{ (since we know } \mathcal{M} \text{ always ends in } E) \\ &= F(E,n) \text{ (by definition)} \end{aligned}$$

4 Posterior Decoding

An alternative to Viterbi decoding is to pick, for each symbol in $\vec{S}$, the most likely state to have generated it, given the whole sequence. The model defines a distribution over paths that could have generated $\vec{S}$ (this distribution is called a *posterior*; it defines $\Pr(\pi|\vec{S},\mathcal{M})$ for all paths π). Given that distribution over paths, we can compute the probability at timestep t that we were in state s.

$$\begin{aligned}
\Pr(\pi_i = s \mid \vec{S},\mathcal{M}) &\stackrel{a}{=} \frac{\Pr(\pi_i = s, \vec{S},\mathcal{M})}{\Pr(\vec{S} \mid \mathcal{M})} \\
&\stackrel{b}{=} \frac{\Pr(\pi_i = s, S_1, ..., S_i \mid \mathcal{M}) \Pr(S_{i+1}, ..., S_n \mid \pi_i = s, \mathcal{M})}{\Pr(\vec{S} \mid \mathcal{M})} \\
&\stackrel{c}{=} \frac{F(s,i)B(s,i)}{F(E,n)}
\end{aligned}$$

In the above derivation, (a) follows from the definition of conditional probability. (b) follows from the independence assumption inherent in all Markov models: everything that happens before timestep i is independent of everything that happens after timestep i, given the state you are in at timestep i. (c) follows from the definition of the Forward and Backward probabilities above.

The table below shows the posterior probability of being in each state at each timestep. It is computed by multiplying the Forward and Backward tables, cell-wise, and normalizing by $F(E,n)$:

	0	1: X	2: X	3: X	4: X
B	1	0	0	0	0
1	0	1	0.517	0.706	0.655
2	0	0	0.483	0.294	0.345
E	0	0	0	0	1

Note that each column sums to one; each column is a distribution over states, and that's just how we defined it. The posterior decoding, then, would be $B, 1, 1, 1, 1, E$. Note that this is different from the Viterbi result.

5 Training: Baum-Welch algorithm

If we had a bunch of sequences generated from a model and we knew the state sequences associated with them, then training our HMM would be easy: count the emissions and transitions, and normalize them into the proper distributions for each state.

But we can actually do training even when we do not have the state sequences. This is done using the Baum-Welch algorithm, which uses the Forward and Backward algorithms.[5]

I will sketch the idea here briefly. Suppose we start with model parameters (probability values) $\vec{\theta}^{(0)}$. (The 0 signifies that these are the initial values.) Locking these parameters, we can run the Forward-Backward to compute posteriors like we did in the last section.

The posteriors we are interested in are slightly different though. Rather than the probability of being in state s at time t, we want to know the probability of either $\mathcal{M}$ transitioning from s to k at time t, or of state s emitting symbol S_t at time t. Because we condition on $\mathcal{M}$ having generated

[5]When you run the Forward algorithm and then the Backward algorithm, we say that you have run the Forward-Backward algorihtm.

$\vec{S}$, these are still posterior probabilities, but now we are interested in the posteriors of generative model steps (transitions and emissions) rather than of $\mathcal{M}$ being in a state at time t.

It turns out that these probabilities are very easy to compute, too, using the Forward and Backward probabilities. Consider emissions first:

$$\Pr(s \text{ emits } S_t \mid \vec{S}) = \Pr(\pi_t = s \mid \vec{S}) = \frac{F(s,t)B(s,t)}{F(E,n)} \tag{17}$$

This follows because, if $\mathcal{M}$ was in state s at time t, we know it must have emitted S_t (the sequence is given!). So the problem reduces to one we've already seen: the probability of being in state s at time t, given the sequence. Forward-Backward to the rescue!

What about transitions?

$$\begin{aligned}
&\Pr(s \rightarrow k \text{ at time } t) \qquad (18)\\
&= \Pr(\pi_t = s, \pi_{t+1} = k \mid \vec{S}, \mathcal{M})\\
&= \frac{\Pr(\pi_t = s, \pi_{t+1} = k, \vec{S} \mid \mathcal{M})}{\Pr(\vec{S} \mid \mathcal{M})}\\
&= \frac{\Pr(S_1, ..., S_t, \pi_t = s \mid \mathcal{M}) \Pr(\pi_{t+1} = k \mid \pi_t = s, \mathcal{M}) \Pr(S_{t+1} \mid \pi_{t+1} = k, \mathcal{M}) \Pr(S_{t+2}, S_{t+3}, ..., S_n \mid \pi_{t+1} = k)}{\Pr(\vec{S} \mid \mathcal{M})}\\
&= \frac{F(s,t)a_s(k)e_k(S_{t+1})B(k,t+1)}{F(E,n)}
\end{aligned}$$

The Baum Welch algorithm will construct new parameters $\vec{\theta}^{(1)}$ by taking these posteriors and treating them like fractional counts. If we had fully state-labeled data, we could ask, at each position, "did state s transition to state k here?" If it did, we would add a count of 1 for that transition. If not, we would leave the count unchanged. With *expected* counting, we don't deal in ones and zeros; we deal in probabilities. So at each timestep, we ask, "what is the probability that state s transitioned to state k here?" And we increment the count of the transition by that probability. We do the same with emissions. Note that each of these involves summing up over all timesteps in all sequences.

The expected counts for the emissions and transitions in $\mathcal{M}$, given the sequence $\vec{S}$, are shown in Table 2.

It should be clear that the total number of transitions we count (summing over all transition types from any state to any other) will be the same under expected counting, no matter how the probabilities come out, if the probabilities are well-formed.

Once we sum up the expected counts, we renormalize and get new probabilities $\vec{\theta}^{(1)}$.

This process can be repeated. It is guaranteed that each time, the total probability of the data will increase. Baum-Welch is a hill-climbing algorithm; although it finds a better solution on each iteration, it can get stuck on a small hill. We are not guaranteed to find the best model, or one that is "correct" in any sense of the word. This is an area of active research. One way to try to avoid this problem is to run Baum-Welch many times, setting the initial parameters $\vec{\theta}^{(0)}$ randomly each time, and picking the best solution after many runs.

The Baum-Welch algorithm is an instance of a more general algorithm called *Expectation-Maximization* (EM).

$B \to 1$	1
$1 \to 1$	$0.517 + 0.260 + 0.380 = 1.157$
$1 \to 2$	$0.483 + 0.257 + 0.326 = 1.066$
$1 \to E$	0.655
$2 \to 1$	$0 + 0.446 + 0.274 = 0.720$
$2 \to 2$	$0 + 0.037 + 0.020 = 0.056$
$2 \to E$	0.345
1 emits X	2.878
1 emits Y	0
2 emits X	1.122
2 emits Y	0

Table 2: Expected counts of emissions and transitions in $\vec{S}$ under model $\mathcal{M}$. Note: These have not been carefully checked; please let me know if you find an error.