

Chapter 4

Markov Models

Hidden Markov Models (HMMs) have been the mainstay of the language modeling used in modern speech recognition systems. Despite their limitations, variants on HMMs are still the most widely used technique in that domain, and are generally regarded as the most successful. In this chapter we will develop the basic theory of HMMs and touch on their applications. In the next chapter is found some more detailed information on extending the basic HMM model and engineering practical implementations. Further information on the application of HMMs to Statistical NLP can be found in Chapters *x* and *y*.

The theory behind Hidden Markov Models was developed by Baum and his colleagues in the late sixties and early seventies, and advocated for use in speech recognition in lectures by Jack Ferguson from the Institute for Defense Analyses. It was applied to speech processing in the 1970s by Baker at CMU and by Jelinek and colleagues at IBM, and then later found its way at IBM and elsewhere into use for other kinds of language modeling, such as part of speech tagging.

An HMM is nothing more than a probabilistic function of a Markov process. Markov processes/chains/models were first developed by Andrei A. Markov (a student of Chebyshev). Their first use was actually for a linguistic purpose – modeling the letter sequences in works of Russian literature (Markov 1913) – but Markov models were then developed as a general statistical tool. We will refer to vanilla Markov models as Visible Markov Models (VMMs) when we want to be careful to distinguish them from HMMs.

4.1 Markov models

Often we want to consider a sequence (perhaps through time) of random variables that *aren't* independent, but depend rather on previous elements in the sequence. For many such systems, it seems reasonable to assume that we can predict the future based just on the present (and don't need the past). That is, future elements of the sequence are conditionally independent of past elements,

given the present element.

Suppose (X_1, \dots, X_T) are a sequence of random variables taking values in some countable set $S = \{s_1, \dots, s_N\}$, the state space. Then the Markov Properties are:

Limited Horizon:

$$P(X_{t+1} = k | X_1, \dots, X_t) = P(X_{t+1} = k | X_t)$$

Time invariant (stationary):

$$= P(X_2 = k | X_1)$$

X is then said to be a Markov chain, or to have the Markov property. One can describe a Markov chain by a stochastic transition matrix A :

$$a_{ij} = P(X_{t+1} = s_j | X_t = s_i)$$

Here, note that $a_{ij} \geq 0, \forall i, j$ and $\sum_{j=1}^N a_{ij} = 1, \forall i$.

Additionally one needs to specify Π , the probabilities of different initial states for the Markov chain:

$$\pi_i = P(X_1 = s_i)$$

Here $\sum_{i=1}^N \pi_i = 1$. The need for this vector can be avoided by specifying that the Markov model always starts off in a certain extra initial state, s_0 , and then using transitions from that state contained within the matrix A to specify the probabilities that used to be recorded in Π .

Alternatively, one can represent a Markov chain by a state diagram as in Figure 4.1. Here, the states are shown as circles around the state name, and the single start state is indicated with an incoming arrow. Possible transitions are shown by arrows connecting states, and these arcs are labeled with the probability of this transition being followed, given that you are in the state at the tail of the arrow. Transitions with zero probability are omitted from the diagram. Note that the probabilities of the outgoing arcs from each state sum to 1. From this representation, it should be clear to people with a CS background that a Markov model can be thought of as a (nondeterministic) finite state automaton with probabilities attached to each arc.

In a visible Markov model, we know what states the machine is passing through, so the state sequence or some deterministic function of it can be regarded as the output.

The probability of a sequence of states (that is, a sequence of random variables) X_1, \dots, X_T is easily calculated for a Markov chain. We find that we need merely calculate the product of the probabilities that occur on the arcs or in the stochastic matrix:

$$\begin{aligned} P(X_1, \dots, X_T) &= P(X_1)P(X_2|X_1)P(X_3|X_1, X_2) \cdots P(X_T|X_1, \dots, X_{T-1}) \\ &= P(X_1)P(X_2|X_1)P(X_3|X_2) \cdots P(X_T|X_{T-1}) \\ &= \pi_{X_1} \prod_{t=1}^{T-1} a_{X_t X_{t+1}} \end{aligned}$$

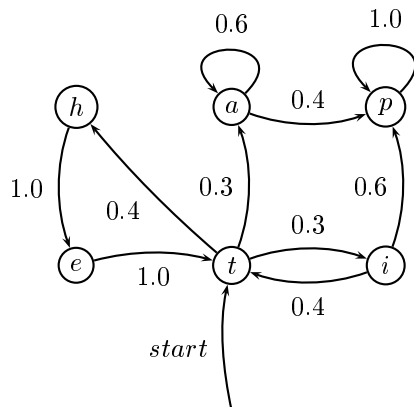


Figure 4.1: A Markov model

So, using the Markov model in Figure 4.1, we have that:

$$\begin{aligned}
 P(t, i, p) &= P(X_1 = t)P(X_2 = i|X_1 = t)P(X_3 = p|X_2 = i) \\
 &= 1.0 \times 0.3 \times 0.6 \\
 &= 0.18
 \end{aligned}$$

It is important to note that what is important is whether we *can* encode a process as a Markov process, not whether we most naturally do. For example, recall the n -gram word models that we saw in an earlier chapter. One might think that, for $n \geq 3$, such a model is not a Markov model because it violates the Limited Horizon condition – we are looking a little into earlier history. But we can reformulate any n -gram model as a visible Markov model by simply encoding the appropriate amount of history into the state space. In general, any fixed finite amount of history can always be encoded in this way by simply elaborating the state space as a crossproduct of multiple previous states. In such cases, we sometimes talk of an n^{th} order Markov model, where n is the number of previous states that we are using to predict the next state.

4.2 Hidden Markov Models

In an HMM, you don't know the state sequence that the model passes through, but only some probabilistic function of it.

Dumb example: Suppose you have a crazy soft drink machine: it can be in two states, cola preferring (CP) and iced tea preferring (IP), but it switches between them randomly after each purchase, as shown in Figure 4.2.

Now, if, when you put in your coin, the machine always put out a cola if it was in the cola preferring state and an iced tea when it was in the iced tea

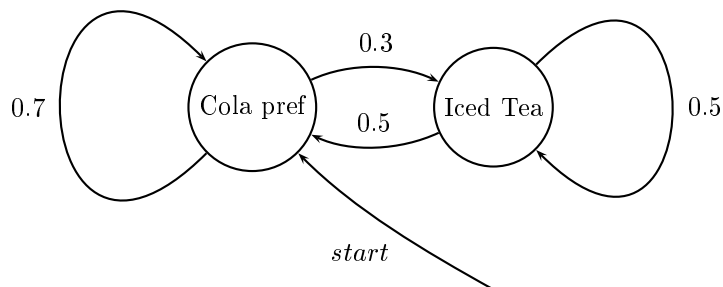


Figure 4.2: The crazy soft drink machine

preferring state, then we would have a visible Markov model. But instead, it only has a tendency to do this. So we need symbol emission probabilities for the observations:

$$P(O_n = k | X_n = s_i, X_{n+1} = s_j) = b_{ijk}$$

For this machine, the output is actually independent of s_j , and so can be described by the following probability matrix:

Output probability given From state

	cola	iced tea	lemonade
CP	0.6	0.1	0.3
IP	0.1	0.7	0.2

Example: What is the probability of seeing the output sequence {lem, ice,t} if the machine always starts off in the cola preferring state?

Answer: $0.3 \times 0.3 \times 0.7 + 0.3 \times 0.7 \times 0.1 = 0.084$. (Note that we need to consider all paths that might be taken through the HMM, and then to sum over them.)

4.2.1 Why use HMMs?

HMMs are useful when one can think of underlying events probabilistically generating surface events. One widespread use of this is tagging – assigning parts of speech (or other classifiers) to the words in a text. We think of there being an underlying Markov chain of parts of speech from which the actual words of the text are generated. Such models are discussed in the next chapter.

When this general model is suitable, the further reason that HMMs are very useful is that they are one of a class of models for which there exist efficient methods of training through use of the Expectation Maximization (EM) algorithm. Given plenty of data that we assume to be generated by *some* HMM – where the model architecture is fixed but not the probabilities on the arcs,

this algorithm allows us to automatically learn the model parameters that best account for the observed data.

Another simple illustration of how we can use HMMs is in generating parameters for linear interpolation of n -gram models. We discussed in an earlier chapter that one way to estimate the probability of a sentence:

$$P(\text{Sue drank her beer before the meal arrived})$$

was with an n -gram model, such as a trigram model, but just using an n -gram model with fixed n tended to suffer because of data sparseness. Recall from the discussion of smoothing that one idea of how to smooth n -gram estimates was to use linear interpolation of n -gram estimates for various n , for example:

$$P_{ti}(w_n|w_{n-1}, w_{n-2}) = \lambda_1 P_1(w_n) + \lambda_2 P_2(w_n|w_{n-1}) + \lambda_3 P_3(w_n|w_{n-1}, w_{n-2})$$

This way we would get some idea of how likely a particular word was, even if our coverage of trigrams is sparse. The question, then, is how to set the parameters λ_i . While we could make reasonable guesses as to what parameter values to use (and we know that together they must obey the stochastic constraint $\sum_i \lambda_i = 1$), it seems that we should be able to find the optimal values automatically. And, indeed, we can (Jelinek 1990).

The key insight is that we can build an HMM with hidden states that represent the choice of whether to use the unigram, bigram, or trigram probabilities. The HMM training algorithm will determine the optimal weight to give to the arcs entering each of these hidden states, which in turn represents the amount of the probability mass that should be determined by each n -gram model via setting the parameters λ_i above.

Concretely, we build an HMM with four states for each word pair, one for the basic word pair, and three representing each choice of n -gram model for calculating the next transition. A fragment of the HMM is shown in Figure 4.3. Note how this HMM assigns the same probabilities as the earlier equation: there are three ways for w_c to follow $w_a w_b$ and the total probability of seeing w_c next is then the sum of each of the n -gram probabilities that adorn the arcs multiplied by the corresponding parameter λ_i . The HMM training algorithm that we develop in this chapter can then be applied to this network, and used to improve initial estimates for the parameters λ_{iab} . There are two things to note. This conversion works by adding epsilon transitions – that is transitions that we wish to say do not produce an output symbol. Secondly, as presented, we now have separate parameters λ_{iab} for each word pair. But we would not want to adjust these parameters separately, as this would make our sparse data problem worse not better. Rather, for a fixed i , we wish to keep all (or at least classes of) the λ_{iab} parameters having the same value, which we do by using *tied states*. Discussion of both of these extensions to the basic HMM model will be deferred to the next chapter.

4.2.2 General form of a HMM

We use the following notation:

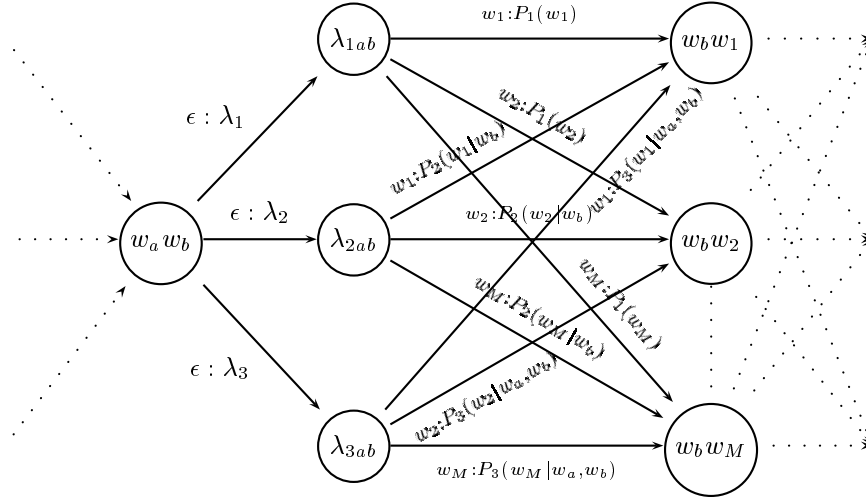


Figure 4.3: Section of HMM for a linearly interpolated language model

Set of states	$S = \{s_1, \dots, s_N\} = \{1, \dots, N\}$
Output alphabet	$K = \{k_1, \dots, k_M\}$
Initial state probabilities	$\Pi = \{\pi_i\}, i \in S$
State transition probabilities	$A = \{a_{ij}\}, i, j \in S$
Symbol emission probabilities	$B = \{b_{ijk}\}, i, j \in S, k \in K$
State sequence	$X = (X_1, \dots, X_{T+1})$
Output sequence	$O = (o_1, \dots, o_T)$

Note that while we can think of the states as having state names, we in fact represent the states as integers, since this allows us to avoid having to use a second level of subscripts in referring to the various probabilities.

How one runs an HMM is perfectly straightforward. It can be done with the pseudocode below:

```

t = 1
start in state i with probability  $\pi_i$  (i.e.,  $X_1 = i$ )
forever
{
  Move from state i to state j with probability  $a_{ij}$  ( $X_{t+1} = j$ )
  Emit observation symbol  $o_t = k$  with probability  $b_{ijk}$ 
  t = t + 1
}

```

4.3 Three fundamental questions for HMMS

There are three fundamental questions that we want to know about an HMM:

1. Given a model $\mu = (A, B, \Pi)$, how do we efficiently compute how likely a certain observation is, that is $P(O|\mu)$?
2. Given the observation sequence O and a model μ , how do we choose a state sequence (X_1, \dots, X_{T+1}) that best explains the observations?
3. Given an observation sequence O , and a space of possible models found by varying the model parameters $\mu = (A, B, \pi)$, how do we find the model that best explains the observed data?

Normally, the problems we deal with are not like the soft drink machine. We don't know the parameters and have to estimate them from data. That's the third question. The first question can be used to decide between models which is best. The second question lets us guess what path was probably followed through the Markov chain, and this hidden path can be used for classification, for instance in applications to part of speech tagging.

4.3.1 Finding the probability of an observation

Given the observation sequence $O = (o_1, \dots, o_T)$ and a model $\mu = (A, B, \Pi)$, we wish to know how to efficiently compute $P(O|\mu)$ – the probability of the observation given the model. This process is often referred to as “decoding”.

For any state sequence $X = (X_1, \dots, X_{T+1})$,

$$\begin{aligned} P(O|X, \mu) &= \prod_{t=1}^T P(o_t|X_t, X_{t+1}, \mu) \\ &= b_{x_1 x_2 o_1} b_{x_2 x_3 o_2} \cdots b_{x_T x_{T+1} o_T} \end{aligned}$$

and,

$$P(X|\mu) = \pi_{x_1} a_{x_1 x_2} a_{x_2 x_3} \cdots a_{x_T x_{T+1}}$$

Now,

$$P(O, X|\mu) = P(O|X, \mu)P(X|\mu)$$

Therefore,

$$\begin{aligned} P(O|\mu) &= \sum_X P(O|X, \mu)P(X|\mu) \\ &= \sum_{x_1 \cdots x_{T+1}} \pi_{x_1} \prod_{t=1}^T a_{x_t x_{t+1}} b_{x_t x_{t+1} o_t} \end{aligned}$$

This derivation is quite straightforward, but unfortunately, direct evaluation of the resulting expression is intractable. The complexity of the calculation is $O((T+1) \cdot N^{T+1})$ multiplications.

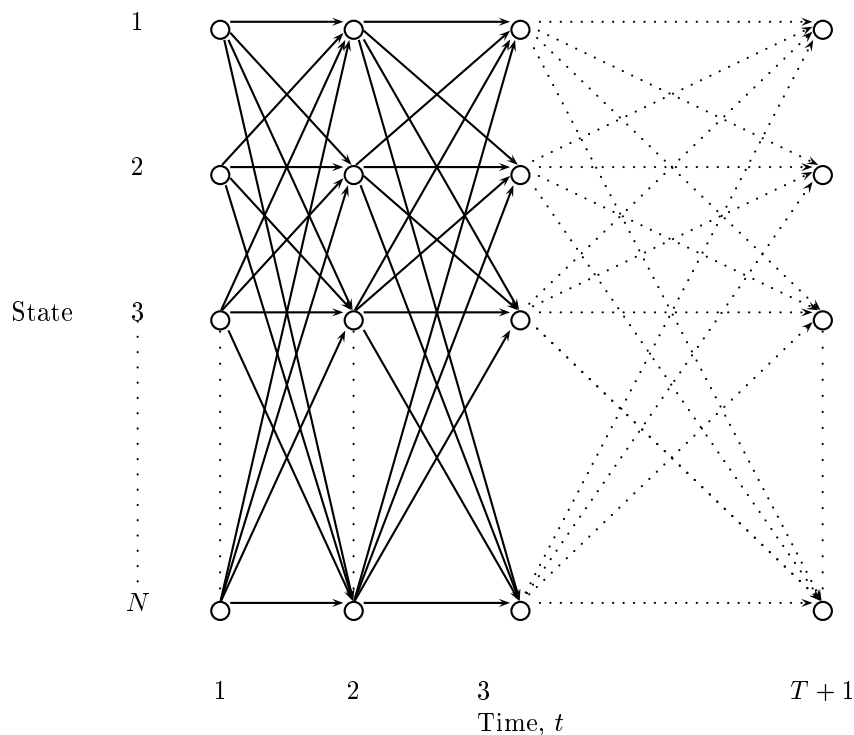


Figure 4.4: Trellis algorithms: The trellis

Exercise: Confirm this claim.

The secret to avoiding this complexity is the general technique of dynamic programming or memoization by which we remember partial results rather than recomputing them. This general concept crops up in many other places in computational linguistics, such as chart parsing, and in computer science more generally (see Cormen et al. (1990:Ch. 16) for a general introduction). For algorithms such as HMMs, the dynamic programming problem is generally described in terms of trellises (also called lattices). Here, we make a square array of states versus time, and compute the probabilities of being at each state at each time in terms of the probabilities for being in each state at the preceding time instant. This is all best seen in pictures – see Figures 4.4 and 4.5. A trellis can record the probability of all initial subpaths of the HMM that end in a certain state at a certain time. The probability of longer subpaths can then be worked out in terms of one shorter subpaths.

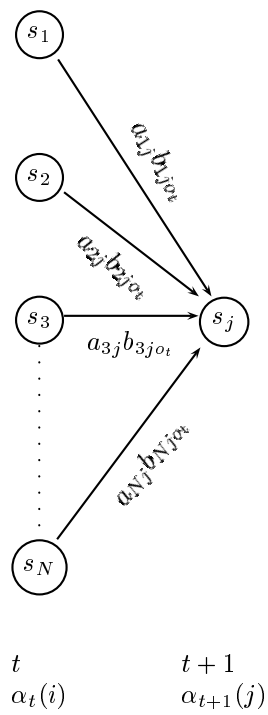


Figure 4.5: Trellis algorithms: Closeup of the computation of one node

The forward procedure

The form of caching that is indicated in these diagrams is called the forward procedure. We describe it in terms of forward variables:

$$\alpha_t(i) = P(o_1 o_2 \cdots o_{t-1}, X_t = i | \mu)$$

Forward variables may be computed via the following procedure:

1. Initialization

$$\alpha_1(i) = \pi_i, 1 \leq i \leq N$$

2. Induction

$$\alpha_{t+1}(j) = \sum_{i=1}^N \alpha_t(i) a_{ij} b_{ij o_t}, 1 \leq t \leq T, 1 \leq j \leq N$$

3. Total

$$P(O | \mu) = \sum_{i=1}^N \alpha_{T+1}(i)$$

This is a much cheaper algorithm that requires $O(N^2 T)$ multiplications.

The backward procedure

It should be obvious that we do not need to cache results working forward through time like this, but rather that we could also work backwards. The reason for introducing this less intuitive calculation here, though, is actually because use of a combination of forwards and backwards probabilities is vital for solving the third problem of parameter reestimation.

Define

$$\beta_t(i) = P(o_t \cdots o_T | X_t = i, \mu)$$

Then we can calculate backwards variables as follows:

1. Initialization

$$\beta_{T+1}(i) = 1, 1 \leq i \leq N$$

2. Induction

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_{ij o_t} \beta_{t+1}(j), 1 \leq t \leq T, 1 \leq i \leq N$$

3. Total

$$P(O | \mu) = \sum_{i=1}^N \pi_i \beta_1(i)$$

Table 4.1 shows the calculation of forward and backwards variables, and certain other variables that we will come to later for the soft drink machine, given the observation sequence $O = (\text{lem,ice,t,cola})$.

Time (t):	Output			
	1	2	3	4
$\alpha_t(CP)$	1.0	0.21	0.0462	0.021294
$\alpha_t(IP)$	0.0	0.09	0.0378	0.010206
$P(o_1 \cdots o_{t-1})$	1.0	0.3	0.084	0.0315
$\beta_t(CP)$	0.0315	0.045	0.6	1.0
$\beta_t(IP)$	0.029	0.245	0.1	1.0
$P(o_1 \cdots o_T)$	0.0315			
$\gamma_t(CP)$	1.0	0.3	0.88	0.676
$\gamma_t(IP)$	0.0	0.7	0.12	0.324
\hat{X}_t	CP	IP	CP	CP
$\delta_t(CP)$	1.0	0.21	0.0315	0.019404
$\delta_t(IP)$	0.0	0.09	0.0315	0.008316
$\psi_t(CP)$		CP	IP	CP
$\psi_t(IP)$		CP	IP	CP
$(\hat{X})_t$	CP	IP	CP	CP
$P(\hat{X})$	0.019404			

Table 4.1: Variable calculations for $O = (\text{lem,ice_t,cola})$

Combining them

More generally, in fact, we can use any combination of forwards and backwards caching to work out the probability of an observation sequence. Observe that:

$$\begin{aligned}
 P(O, X_t = i | \mu) &= P(o_1 \cdots o_T, X_t = i | \mu) \\
 &= P(o_1 \cdots o_{t-1}, X_t = i, o_t \cdots o_T | \mu) \\
 &= P(o_1 \cdots o_{t-1}, X_t = i | \mu) P(o_t \cdots o_T | o_1 \cdots o_{t-1}, X_t = i, \mu) \\
 &= P(o_1 \cdots o_{t-1}, X_t = i | \mu) P(o_t \cdots o_T | X_t = i, \mu) \\
 &= \alpha_t(i) \beta_t(i)
 \end{aligned}$$

Therefore:

$$p(O | \mu) = \sum_{i=1}^N \alpha_t(i) \beta_t(i), 1 \leq t \leq T + 1$$

The previous equations were special cases of this one.

4.3.2 Finding the best state sequence

The second problem was worded somewhat vaguely as “finding the state sequence that best explains the observations”. That is because there is more than one way to think about doing this. One way to proceed would be to choose the states individually. That is, for each $t, 1 \leq t \leq T + 1$, we would find X_t that maximizes $P(X_t | O, \mu)$.

Let

$$\begin{aligned}
 \gamma_t(i) &= P(X_t = i | O, \mu) \\
 &= \frac{P(X_t = i, O | \mu)}{P(O | \mu)} \\
 &= \frac{\alpha_t(i) \beta_t(i)}{\sum_{j=1}^N \alpha_t(j) \beta_t(j)}
 \end{aligned}$$

The individually most likely state \hat{X}_t is:

$$\hat{X}_t = \arg \max_{1 \leq i \leq N} \gamma_t(i), 1 \leq t \leq T + 1$$

This quantity maximizes the expected number of states that will be guessed correctly. However, it may yield a quite unlikely state *sequence*. Therefore, this is not the method that is normally used, but rather the Viterbi algorithm, which efficiently computes the most likely state sequence.

Viterbi algorithm

Commonly we want to find the most likely complete path, that is:

$$\arg \max_X P(X | O, \mu)$$

To do this, it is sufficient to maximize (for a fixed O):

$$\arg \max_X P(X, O | \mu)$$

An efficient trellis algorithm for computing this path is the Viterbi algorithm. Define:

$$\delta_t(j) = \max_{x_1 \cdots x_{t-1}} P(x_1 \cdots x_{t-1}, o_1 \cdots o_{t-1}, X_t = j | \mu)$$

This variable stores for each point in the trellis the probability of the most probable path that leads to that node. Using dynamic programming, we then calculate the most probable path through the whole trellis as follows:

1. Initialization

$$\delta_1(j) = \pi_j, 1 \leq j \leq N$$

2. Induction

$$\delta_{t+1}(j) = \max_{1 \leq i \leq N} \delta_t(i) a_{ij} b_{ij o_t}, 1 \leq j \leq N$$

Store backtrace

$$\psi_{t+1}(j) = \arg \max_{1 \leq i \leq N} \delta_t(i) a_{ij} b_{ij o_t}, 1 \leq j \leq N$$

3. Termination and path readout (by backtracking)

$$\begin{aligned} \hat{X}_{T+1} &= \arg \max_{1 \leq i \leq N} \delta_{T+1}(i) \\ \hat{X}_t &= \psi_{t+1}(\hat{X}_{t+1}) \\ P(\hat{X}) &= \max_{1 \leq i \leq N} \delta_{T+1}(i) \end{aligned}$$

Table 4.1 above shows the computation of the most likely states and state sequence under both these interpretations – for this example, they prove to be identical.

4.3.3 Third problem: parameter estimation

We want to find the optimal values of the model parameters $\mu = (A, B, \pi)$. Using Maximum Likelihood Estimation, that means we want to find the values that maximize $P(O | \mu)$:

$$\arg \max_{\mu} P(O_{\text{training}} | \mu)$$

There is no known analytic method to choose μ to maximize $P(O | \mu)$. But we can locally maximize it by an iterative hill-climbing algorithm. This algorithm is the Baum-Welch or Forwards-Backwards algorithm, which is a special case of the Expectation Maximization method which we will cover in greater generality

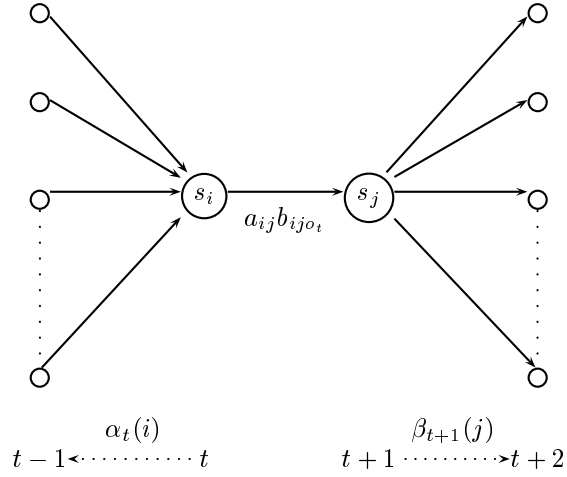


Figure 4.6: Traversing an arc

later. This maximization process is often referred to as *training* the model and is performed on *training data*.

Define $p_t(i, j), 1 \leq t \leq T, 1 \leq i, j \leq N$ as shown below. This is the probability of traversing a certain arc at time t given observation sequence O ; see Figure 4.6.

$$\begin{aligned}
 p_t(i, j) &= P(X_t = i, X_{t+1} = j | O, \mu) \\
 &= \frac{P(X_t = i, X_{t+1} = j, O | \mu)}{P(O | \mu)} \\
 &= \frac{\alpha_t(i) a_{ij} b_{ij|o_t} \beta_{t+1}(j)}{\sum_{m=1}^N \alpha_t(m) \beta_t(m)} \\
 &= \frac{\alpha_t(i) a_{ij} b_{ij|o_t} \beta_{t+1}(j)}{\sum_{m=1}^N \sum_{n=1}^N \alpha_t(m) a_{mn} b_{mn|o_t} \beta_{t+1}(n)}
 \end{aligned}$$

Note that $\gamma_t(i) = \sum_{j=1}^N p_t(i, j)$.

Now, if we sum over the time index, this gives us expectations (counts):

$$\sum_{t=1}^T \gamma_t(i) = \text{expected number of transitions from state } i \text{ in } O$$

$$\sum_{t=1}^T p_t(i, j) = \text{expected number of transitions from state } i \text{ to } j \text{ in } O$$

So we begin with some model μ (perhaps preselected, perhaps just chosen randomly). We then run O through the current model to estimate the expectations

of each model parameter. We then change the model to maximize the values of the paths that are used a lot (while still respecting the stochastic constraints). We then repeat this process, hoping to converge on optimal values for the model parameters μ .

The reestimation formulas are as follows:

$$\begin{aligned}\bar{\pi}_i &= \text{expected frequency in state } i \text{ at time } t = 1 \\ &= \gamma_1(i) \\ \bar{a}_{ij} &= \frac{\text{expected number of transitions from state } i \text{ to } j}{\text{expected number of transitions from state } i} \\ &= \frac{\sum_{t=1}^T p_t(i, j)}{\sum_{t=1}^T \gamma_t(i)} \\ \bar{b}_{ijk} &= \frac{\text{expected number of transitions from } i \text{ to } j \text{ when } k \text{ is observed}}{\text{expected number of transitions from } i \text{ to } j} \\ &= \frac{\sum_{\{t: o_t=k, 1 \leq t \leq T\}} p_t(i, j)}{\sum_{t=1}^T p_t(i, j)}\end{aligned}$$

Thus, from $\mu = (A, B, \Pi)$, we derive $\bar{\mu} = (\bar{A}, \bar{B}, \bar{\Pi})$. Further, as proved by Baum (see Chapter *x*), we have that:

$$P(O|\bar{\mu}) \geq P(O|\mu)$$

Therefore, iterating through a number of rounds of parameter reestimation will improve our model. Normally one continues reestimating the parameters until results are no longer improving significantly. This process of parameter reestimation does not guarantee that we will find the best model, however, because the reestimation process may get stuck in a local maximum (or even possibly just at a saddle point). In most problems of interest, the likelihood function is a complex nonlinear surface and there are many local maxima. Nevertheless, Baum-Welch reestimation is usually effective for HMMS.

To end this chapter, let us consider reestimating the parameters of the crazy soft drink machine HMM using the Baum-Welch algorithm. If we let the initial model be the model that we have been using so far, then training on the observation sequence (*lem, ice_t, cola*) will yield the following values for $p_t(i, j)$:

		Time (and <i>j</i>)								
		1			2			3		
		CP	IP	γ_1	CP	IP	γ_2	CP	IP	γ_3
<i>i</i>	CP	0.3	0.7	1.0	0.28	0.02	0.3	0.616	0.264	0.88
	IP	0.0	0.0	0.0	0.6	0.1	0.7	0.06	0.06	0.12

and so the parameters will be reestimated as follows:

		Original			Reestimated		
Π	CP	1.0			1.0		
	IP	0.0			0.0		
		CP	IP		CP	IP	
A	CP	0.7	0.3		0.5486	0.4514	
	IP	0.5	0.5		0.8049	0.1951	
		cola	ice_t	lem	cola	ice_t	lem
B	CP	0.6	0.1	0.3	0.4037	0.1376	0.4587
	IP	0.1	0.7	0.2	0.1363	0.8537	0.0

Exercise: If one continued running the Baum-Welch algorithm on this HMM and this training sequence, what value would each parameter reach in the limit? Why?

The reason why the Baum-Welch algorithm is performing so strangely here should be apparent: the training sequence is far too short to accurately represent the behavior of the crazy soft drink machine.

Exercise: Note that the parameter that is zero in Π stays zero. Is that a chance occurrence? What would be the value of the parameter that becomes zero in B if we did another iteration of Baum-Welch reestimation? What generalization can one make about Baum-Welch reestimation of zero parameters?

4.4 References

Two standard references on HMM algorithms (in the context of speech recognition) are Rabiner (1989) and Rabiner and Juang (1993). They consider continuous HMMs as well as the discrete HMMs we have considered here, contain information on applications of HMMs to speech recognition and may also be consulted for fairly comprehensive references on the development of the use of HMMs. Our presentation of HMMs is however most closely based on that of Paul (1990).

Bibliography

- Cormen, T. H., C. E. Leiserson, and R. L. Rivest. 1990. *Introduction to Algorithms*. Cambridge, MA: MIT Press.
- Jelinek, F. 1990. Self-organized language modeling for speech recognition. MS, IBM T.J. Watson research center.
- Markov, A. A. 1913. Russian [An example of statistical investigation in the text of 'Eugene Onyegin' illustrating coupling of 'tests' in chains]. In *Proceedings of the Academy of Sciences, St. Petersburg*, Vol. 7 of VI, 153.

- Paul, D. B. 1990. Speech recognition using hidden markov models. *The Lincoln Laboratory Journal* 3(1):41–62.
- Rabiner, L., and B.-H. Juang. 1993. *Fundamentals of Speech Recognition*. Englewood Cliffs, NJ: PTR Prentice-Hall.
- Rabiner, L. R. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of IEEE* 77(2):257–286.