8.5 Markov Models and Hidden Markov Models

Andrei Andreevich Markov first introduced his mathematical model of dependence, now known as Markov chains, in 1907. A Markov model (or Markov chain) is a mathematical model used to represent the tendency of one event to follow another event, or even to follow an entire sequence of events. Markov chains are matrices comprised of probabilities that reflect the dependence of one or more events on previous events. Markov first applied his modeling technique to determine tendencies found in Russian spelling. Since then, Markov chains have been used as a modeling technique for a wide variety of applications ranging from weather systems to baseball games.

Statistical methods of Markov source or hidden Markov modeling (HMM) have become increasingly popular in the last several years. The models are very rich in mathematical structure and hence can form a basis for use in a wide range of applications. Moreover the models, when applied properly, work very well in practice for several important applications.

8.5.1 Markov Models or Markov chains

Markov models are very useful to represent families of sequences with certain specific statistical properties. To explain the idea consider a simple 3 state model of the weather. We assume that once a day, the weather is observed as being one of the following: rain (state 1); cloudy (state 2); sunny (state 3).

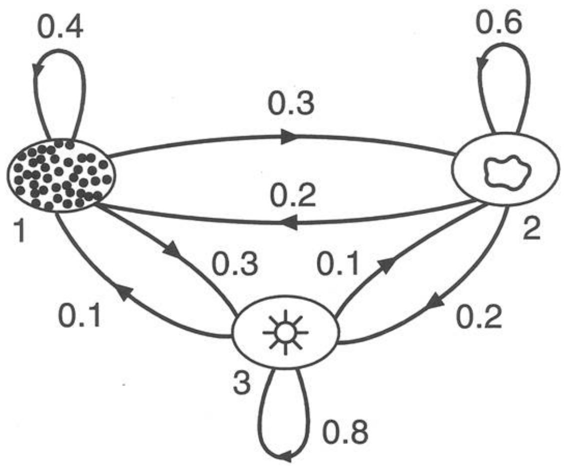


Figure 8.32: *State transition of the weather Markov model (from Rabiner 1999).*

If we examine a sequence of observation during a month, the state rain appears a few times, and it can be followed by rain, cloud or sun. Given a long sequence of observations, we can count the number of times the state rain is followed by, say, a cloudy state. From this we can estimate the probability that a rain is followed by a cloudy state. If this probability is 0.3 for example, we indicate it as shown in Figure 8.32. The figure also shows examples of probabilities for every state to transition to other states, including itself. The first row of the matrix A

$$A = \{a_{i,j}\} = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix} \tag{8.9}$$

shows the three probabilities more compactly (notice that their sum is unity). Similarly the probabilities that the cloudy state would transition into the three states can be estimated, and is shown in the second row of the matrix. This 3×3 matrix is called a state transition matrix, and is denoted as $\mathbf{A}$ and the coefficients have the properties $a_{i,j} \geq 0$ and $\sum_j a_{i,j} = 1$ since they obey standard stochastic constraints. Figure 8.32 is called a Markov model.

Formally a Markov model (MM) models a process that goes through a sequence of discrete states, such as notes in a melody. At regular spaced, discrete times, the system undergoes a change of state (possibly back to same state) according to a set of probabilities associated with the state. The time instances for a state change is denoted t and the actual state at time t as $x(t)$. The model is a weighted automaton that consists of:

- A set of N states, $S = \{s_1, s_2, s_3, \dots, s_N\}$.
- A set of transition probabilities, $\mathbf{A}$, where each $a_{i,j}$ in $\mathbf{A}$ represents the probability of a transition from s_i to s_j. I.e $a_{i,j} = P\left[x(t) = j \,|\, x(t-1) = i\right]$.
- A probability distribution, π, where π_i is the probability the automaton will begin in state s_i, i.e $\pi_i = P(x_1 = i)$, $1 \leq i \leq N$. Notice that the stochastic property for the initial state distribution vector is $\sum_i \pi_i = 1$.
- E, a subset of S containing the legal ending states.

In this model, the probability of transitioning from a given state to another state is assumed to depend only on the current state. This is known as the Markov property.

Given a sequence or a set of sequences of similar kind (e.g., a long list of melodies from a composer) the parameters of the model (the transition probabilities) can readily be estimated. The process of identifying the model parameters is called training the model. In all discussions it is implicitly assumed that the probabilities of transitions are fixed and do not depend on past transitions.

Suppose we are given a Markov model (i.e., $\mathbf{A}$ given). Given an arbitrary state sequence $\mathbf{x} = [x(1), x(2), ..., x(L)]$ we can calculate the probability that **x** has been generated by our model. This is given by the product

$$P(\mathbf{x}) = P(x(1)) \times P(x(1) \rightarrow x(2)) \times P(x(2) \rightarrow x(3)) \times \cdots \times P(x(L-1) \rightarrow x(L))$$

where $P(x(1)) = \pi(x(1))$ is the probability that x(1) is the initial state, $P(x(k) \rightarrow x(m))$ is the transition probability for going from $x(k)$ to $x(m)$, and can be found from the matrix $\mathbf{A}$. For example with reference to the weather Markov model of equation 8.9, given that the weather on day 1 is sunny (state 3), we can ask the question: What is the probability that the weather for next 7 days will be "sun-sun-rain-rain-sun-cloudy-sun . ."? This probability can be evaluated as

$$\begin{aligned} P &= \pi_3 \cdot a_{33} \cdot a_{33} \cdot a_{31} \cdot a_{11} \cdot a_{13} \cdot a_{32} \cdot a_{13} \\ &= 1(0.8)(0.8)(0.1)(0.4)(0.3)(0.1)(0.2) \\ &= 1.536 \times 10^{-4} \end{aligned}$$

The usefulness of such computation is as follows: given a number of Markov models ($\mathbf{A_1}$ for a composer, $\mathbf{A_2}$ for a second composer, and so forth) and given a melody $\mathbf{x}$, we can calculate the probabilities that this melody is generated by any of these models. The model which gives the highest probability is most likely the model which generated the sequence.

8.5.2 Hidden Markov Models

A hidden Markov model (HMM) is obtained by a slight modification of the Markov model. Thus consider the state diagram shown in Figure 8.32 which shows three states numbered 1, 2, and 3. The probabilities of transitions from the states are also indicated, resulting in the state transition matrix $\mathbf{A}$ shown in equation 8.9. Now we can suppose that we can not observe directly the state, but only a symbol that is associated in a probabilistic way to the state. For example when the weather system is in a particular state, it can output one of four possible symbols L, M, H, VH (corresponding to temperature classes low, medium, high, very high), and there is a probability associated with each of these. This is summarized in the so-called output matrix $\mathbf{B}$

$$\mathbf{B} = \{b_{i,j}\} = \begin{bmatrix} 0.4 & 0.3 & 0.2 & 0.1 \\ 0.2 & 0.5 & 0.2 & 0.1 \\ 0.1 & 0.1 & 0.4 & 0.4 \end{bmatrix} \tag{8.10}$$

The element $b_{i,j}$ represents the probability of observing the temperature class j when the weather is in the (non observable) state i, i.e. $b_{i,j} = P(x(t) = s_i | x(t) = j)$. For example when the weather is rainy (state $i = 1$), the probability of measuring medium temperature (output symbol $j = 2$) is $b_{1,2} = 0.3$.

More formally, an HMM requires two things in addition to that required for a standard Markov model:

- A set of possible observations, $O = \{o_1, o_2, o_3, \dots, o_n\}$.
- A probability distribution $\mathbf{B}$ over the set of observations for each state in S.

Basic HMM problems In order to apply the hidden Markov model theory successfully there are three problems that need to be solved in practice. These are listed below along with names of standard algorithms which have been developed for these.

1. Learn structure problem. Given an HMM (i.e., given the matrices $\mathbf{A}$ and $\mathbf{B}$) and an output sequence $o(1), o(2), \dots$, compute the state sequence $x(k)$ which most likely generated it. This is solved by the famous Viterbi's algorithm (see 8.5.4.2).
2. Evaluation or scoring problem. Given the HMM and an output sequence $o(1), o(2), \dots$ compute the probability that the HMM generates this. We can also view the problem as one of scoring how well a given model matches a given output sequence. If we are trying to choose among several competing models, this ranking allow us to choose the model that best matches the observations. The forward-backward algorithm solves this (see 8.5.4.1).
3. Training problem. How should one design the model parameters $\mathbf{A}$ and $\mathbf{B}$ such that they are optimal for an application, e.g., to represent a melody? The most popular algorithm for this is the expectation maximization algorithm commonly known as the EM algorithm or the Baum-Welch algorithm (see Rabiner [1989] for more details).

For example let us consider a simple isolated word recognizer (see Figure 8.33). For each word we want to design a separate N-state HHM. We represent the speech signal as a time sequence of coded spectral vectors. Hence each observation is the index of the spectral vector closest to the original speech signal. Thus for each word, we have a training sequence consisting of repetitions of codebook indices of the word.

The first task is to build individual word models. This task is done by using the solution to Problem 3 to estimate model parameters for each word model. To develop an understanding of physical meaning of the model state, we use the solution to Problem 1 to segment each of the word state sequence into states, and then study the properties of the spectral vectors that lead to the observations occurring in each state. Finally, once the set of HMMs has been designed, recognition of an unknown word is performed using the solution to Problem 2 to score each word model based on the observation sequence, and select the word whose model score is highest.

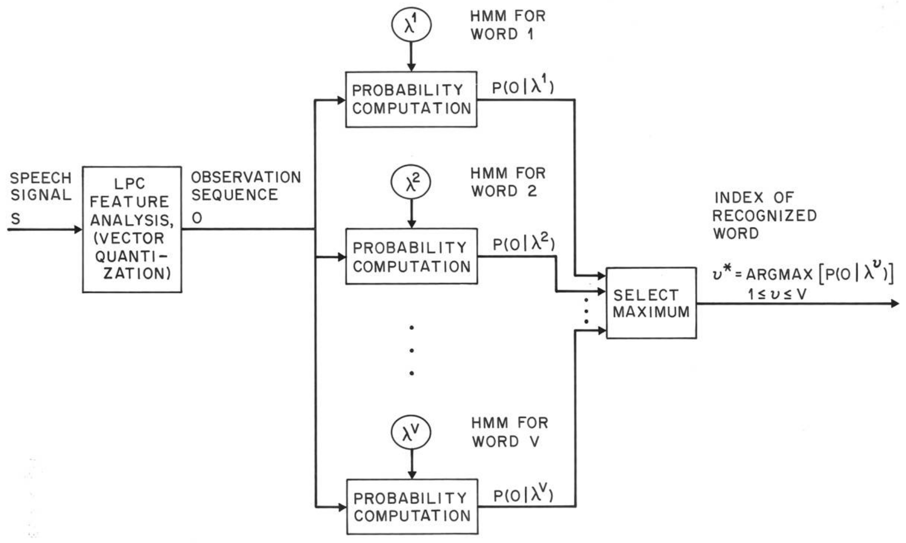


Figure 8.33: *Block diagram of an isolated word recognizer (from Rabiner 1999).*

We should remark that the HMM is a stochastic approach which models the given problem as a doubly stochastic process in which the observed data are thought to be the result of having passed the true (hidden) process through a second process. Both processes are to be characterized using only the one that could be observed. The problem with this approach, is that one do not know anything about the Markov chains that generate the speech. The number of states in the model is unknown, there probabilistic functions are unknown and one can not tell from which state an observation was produced. These properties are hidden, and thereby the name hidden Markov model.

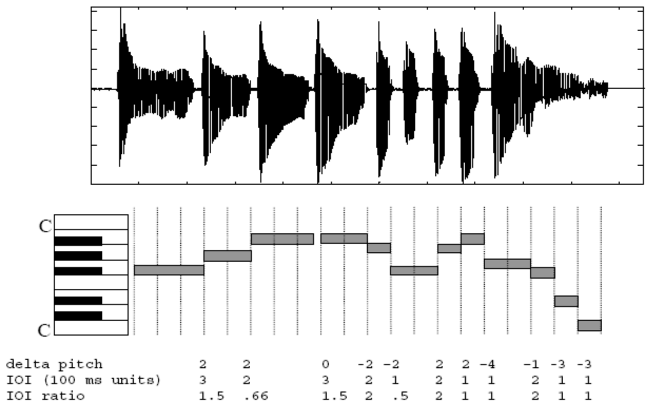
8.5.3 Markov Models Applied to Music

Hiller and Isaacson (1957) were the first to implement Markov chains in a musical application. They developed a computer program that used Markov chains to compose a string quartet comprised of four movements entitled the Illiac Suite. Around the same time period, Meyer and Xenakis (1971) realized that Markov chains could reasonably represent musical events. In his book Formalized Music Xenakis [1971], Xenakis described musical events in terms of three components: frequency, duration, and intensity. These three components were combined in the form of a vector and then were used as the states in Markov chains. In congruence with Xenakis, Jones (1981) suggested the use of vectors to describe notes (e.g., note = pitch, duration, amplitude, instrument) for the purposes of eliciting more complex musical behavior from a Markov chain. In addition, Polansky, Rosenboom, and Burk (1987) proposed the use of hierarchical Markov chains to generate different levels of musical organization (e.g., a high level chain to define the key or tempo, an intermediate level chain to select a phrase of notes, and a low level chain to determine the specific pitches). All of the aforementioned research deals with the compositional aspects and uses of Markov chains. That is, all of this research was focused on creating musical output using Markov chains.

8.5.3.1 HMM models for music search: MuseArt

In the MuseArt system for music search and retrieval, developed at Michigan University by Jonah Shifrin, Bryan Pardo, Colin Meek, William Birmingham, musical themes are represented using a hidden Markov model (HMM).

Representation of a query. The query is treated as an observation sequence and a theme is judged similar to the query if the associated HMM has a high likelihood of generating the query. A piece of music is deemed a good match if at least one theme from that piece is similar to the query. The pieces are returned to the user in order, ranked by similarity.



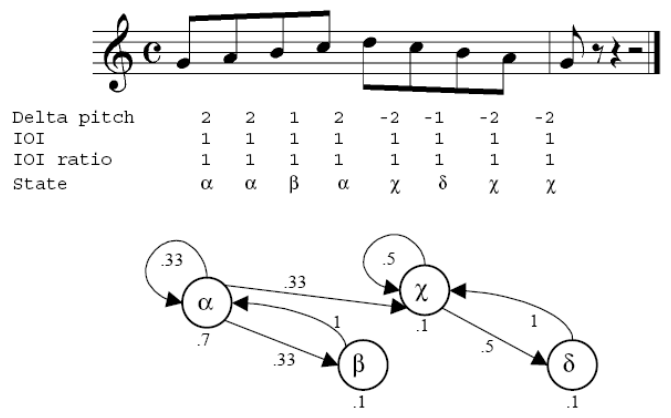
```
delta pitch          2    2         0   -2  -2     2  2 -4    -1 -3 -3
IOI (100 ms units)   3    2         3    2  1      2  1  1     2  1  1
IOI ratio            1.5  .66       1.5  2  .5     2  1  1     2  1  1
```

Figure 8.34: *A sung query (from Shifrin 2002)*

A query is a melodic fragment sung by a single individual. The singer is asked to select one syllable, such as ta or la, and use it consistently during the query. The consistent use of a single consonant-vowel pairing lessens pitch-tracker error by providing a clear onset point for each note, as well as reducing error caused by vocalic variation. A query is recorded as a .wav file and is transcribed into a MIDI based representation using a pitch-tracking system. Figure 8.34 shows a time-amplitude representation of a sung query, along with example pitch-tracker output (shown as piano roll) and a sequence of values derived from the MIDI representation (the $deltaPitch$, IOI and $IOIratio$ values). Time values in the figure are rounded to the nearest 100 milliseconds. We define the following.

- A note transition between note n and note $n + 1$ is described by the duple $<$ $deltaPitch$, $IOIratio >$.
- $deltaPitch_n$ is the musical interval, i.e. the pitch difference in semitones between note n and note $n + 1$.
- $IOIratio_n$ is IOI_n/IOI_{n+1}, where the inter onset interval (IOI_n) is the difference between the onset of notes n and $n + 1$. For the final transition, $IOI_n = IOI_n/duration_{n+1}$.

A query is represented as a sequence of note transitions. Note transitions are useful because they are robust in the face of transposition and tempo changes. The $deltaPitch$ component of a note transition captures pitch-change information. Two versions of a piece played in two different keys have the same $deltaPitch$ values. The $IOIratio$ represents the rhythmic component of a piece. This remains constant even when two performances are played at very different speeds, as long as relative durations within each performance remain the same. In order to reduce The number of possible IOI ratios is reduced by quantizing them to one of 27 values, spaced evenly on a logarithmic scale. A logarithmic scale was selected because data from a pilot study indicated that sung $IOIratio$ values fall naturally into evenly spaced bins in the log domain.



Delta pitch	2	2	1	2	-2	-1	-2	-2
IOI	1	1	1	1	1	1	1	1
IOI ratio	1	1	1	1	1	1	1	1
State	α	α	β	α	χ	δ	χ	χ

Figure 8.35: *Markov model for a scalar passage (from Shifrin 2002)*

The directed graph in Figure 8.35 represents a Markov model of a scalar passage of music. States are note transitions. Nodes represent states. The numerical value below each state indicates the probability a traversal of the graph will begin in that state. As a default, all states are assumed to be legal ending states. Directed edges represent transitions. Numerical values by edges indicate transition probabilities. Only transitions with non-zero probabilities are shown.

In Markov model, it is implicitly assumed that whenever state s is reached, it is directly observable, with no chance for error. This is often not a realistic assumption. There are multiple possible sources of error in generating a query. The singer may have incorrect recall of the melody he or she is attempting to sing. There may be production errors (e.g., cracked notes, poor pitch control). The transcription system may introduce pitch errors, such as octave displacement, or timing errors due to the quantization of time. Such errors can be handled gracefully if a probability distribution over the set of possible observations (such as note transitions in a query) given a state (the intended note transition of the singer) is maintained. Thus, to take into account these various types of errors, the Markov model should be extended to a hidden Markov Model, or HMM. The HMM allows us a probabilistic map of observed states to states internal to the model (hidden states). In the system, a query is a sequence of observations. Each observation is a note-transition duple, $< deltaPitch, IOIratio >$. Musical themes are represented as hidden Markov models whose states also corresponds to note-transition duples. To make use of the strengths of a hidden Markov model, it is important to model the probability of each observation o_i in the set of possible observations, O, given a hidden state, s.

Making Markov Models from MIDI. Our system represents musical themes in a database as HMMs. Each HMM is built automatically from a MIDI file encoding the theme. The unique duples characterizing the note transitions found in the MIDI file form the states in the model. FigureFigure 8.35 shows a passage with eight note transitions characterized by four unique duples. Each unique duple is represented as a state. Once the states are determined for the model, transition probabilities between states are computed by calculating what proportion of the time state a follows state b in the theme. Often, this results in a large number of deterministic transitions. Figure 8.36 is an example of this, where only a single state has two possible transitions, one back to itself and the other on to the next state. Note that there is not a one-to-one correspondence between model and observation se-

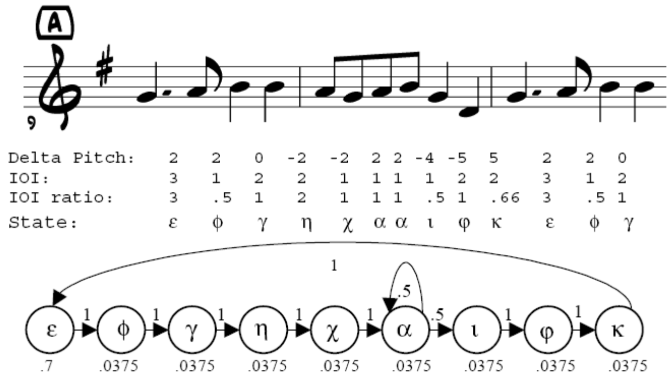

A
9
Delta Pitch: 2 2 0 -2 -2 2 2 -4 -5 5 2 2 0
IOI: 3 1 2 2 1 1 1 1 2 2 3 1 2
IOI ratio: 3 .5 1 2 1 1 1 .5 1 .66 3 .5 1
State: ε ϕ γ η χ α α ι φ κ ε ϕ γ
1
.5
1 1 1 1 1 1 .5 1 1
ε ϕ γ η χ α ι φ κ
.7 .0375 .0375 .0375 .0375 .0375 .0375 .0375 .0375

Figure 8.36: *Markov model for Alouette fragment (from Shifrin 2002)*

quence. A single model may create a variety of observation sequences, and an observation sequence may be generated by more than one model. Recall that our approach defines an observation as a duple, $< deltaPitch, IOIratio >$. Given this, the observation sequence $q = \{(2,1),(2,1),(2,1)\}$ may be generated by the HMM in Figure 8.35 or the HMM in Figure 8.36.

Finding the best target. The themes in the database are coded as HMMs and the query is treated as an observation sequence. Given this, we are interested in finding the HMM most likely to generate the observation sequence. This can be done using the Forward algorithm. The Forward algorithm, given an HMM and an observation sequence, returns a value between 0 and 1, indicating the probability the HMM generated the observation sequence. Given a maximum path length, L, the algorithm takes all paths through the model of up to L steps. The probability each path has of generating the observation sequence is calculated and the sum of these probabilities gives the probability that the model generated the observation sequence. This algorithm takes on the order of $|S|^2L$ steps to compute the probability, where $|S|$ is the number of states in the model.

Let there be an observation sequence (query), O, and a set of models (themes), M. An order may be imposed on M by performing the Forward algorithm on each model m in M and then ordering the set by the value returned, placing higher values before lower. The i-th model in the ordered set is then the i-th most likely to have generated the observation sequence. We take this rank order to be a direct measure of the relative similarity between a theme and a query. Thus, the first theme is the one most similar to the query.

8.5.3.2 Markov sequence generator

Markov models can be thought of as generative models. A generative model describes an underlying structure able to generate the sequence of observed events, called an observation sequence. Note that there is not a one-to-one correspondence between model and observation sequence. A single model may create a variety of observation sequences, and an observation sequence may be generated by more than one model.

A HMM can be used as generator to give an observation sequence O as follow

1. Choose initial state $x(1) = S_1$ according the initial state distribution π.

2. Set $t = 1$
3. Choose $o(t)$ according the symbol probability distribution in state $x(t)$ described in matrix $\mathbf{B}$
4. Transit to new state $x(t+1) = S_j$ according to the state transition probability for state $x(t) = i$, i.e. $a_{i,j}$
5. Set $t = t + 1$ and return to step 2

If a simple Markov model is used as generator, step 3 is skipped, and the state $x(t)$ is used in output.

The "hymn tunes" of Figure 8.37 were generated by computer from an analysis of the probabilities of notes occurring in various hymns. A set of hymn melodies were encoded (all in C major). Only hymn melodies in 4/4 meter and containing two four-bar phrases were used. The first "tune" was generated by simply randomly selecting notes from each of the corresponding points in the analyzed melodies. Since the most common note at the end of each phrase was 'C' there is a strong likelihood that the randomly selected pitch ending each phrase is C.

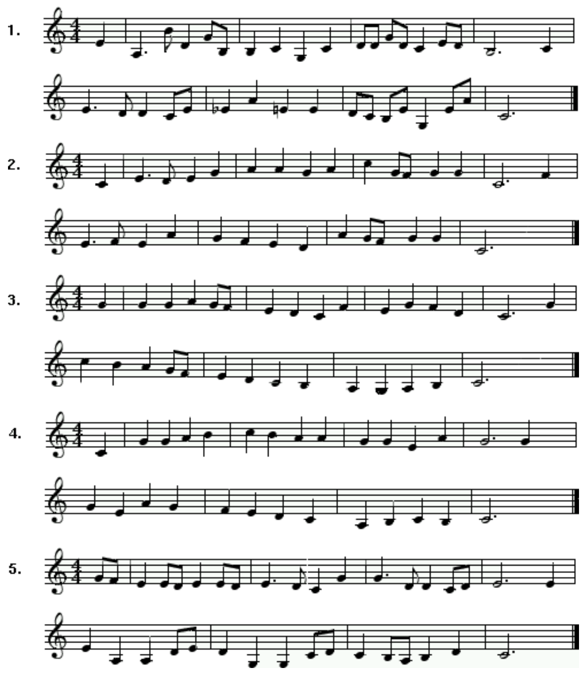


Figure 8.37: *"Hymn tunes" generated by computer from an analysis of the probabilities of notes occurring in various hymns. From Brooks, Hopkins, Neumann, Wright. "An experiment in musical composition." IRE Transactions on Electronic Computers, Vol. 6, No. 1 (1957).*

8.5.4 Algorithms

8.5.4.1 Forward algorithm

The Forward algorithm is uses to solve the evaluation or scoring problem. Given the HMM $\lambda = (A, B, \Pi)$ and an observation sequence $O = o(1)o(2)\ldots o(L)$ compute the probability $P(O|\lambda)$ that the HMM generates this. We can also view the problem as one of scoring how well a given model matches a given output sequence. If we are trying to choose among several competing models, this ranking allow us to choose the model that best matches the observations. The most straighforward procedure is through enumerating every possible state sequence of lenght L (the number of observations), computing the joint probability of the state sequence and O and finally summing the joint probabilty over all possible sate sequence. But if there are N possible states that can be reached, there are N^L possible state sequences and thus such direct approach have exponential computational complexity.

However we can notice that there are only N states and we can apply a dynamic programming stategy. To this purpose let us define the forward variable $\alpha_t(i)$ as

$$\alpha_t(i) = P(o(1)o(2)\dots o(t),\, x(t) = s_i|\lambda)$$

i.e. the probability of the partial observation $o(1)o(2)\dots o(t)$ and state s_i at time t, given the model λ. The Forward algorithm solves the problem with a dynamic programming strategy, using an iteration on the sequence length (time t), as follows:

1. Initialization
$$\alpha_1(i) = \pi(i)b_i(o_1),\ 1 \le i \le N$$

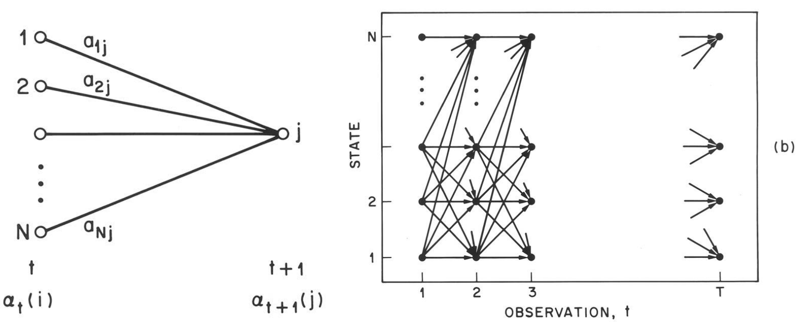
2. Induction
$$\alpha_{t+1}(i) = \left[\sum_{i=1}^{N} \alpha_t(i)a_{ij}\right] b_j(o_{t+1}) \quad 1 \le t \le L-1$$
$$1 \le i \le N$$

3. Termination
$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_L(i)$$

Step 1) initializes the forward probabilities as the joint probability of state i and initial observation $o(1)$. The induction step is illustrated in Figure 8.38(a). This figure shows that state s_j can be reached at time $t+1$ from the N possible states at time t. Since $\alpha_t(i)$ is the probability that $o(1)o(2)\dots o(t)$ is observed and $x(t) = s_i$, the product $\alpha_t(i)a_{ij}$ is the probability that $o(1)o(2)\dots o(t)$ is observed and state s_j is reached at time $t+1$ via state s_i at time t. Summing this product over all the possible states results in the probability of s_j with all the previous observations. Finally $\alpha_{t+1}(i)$ s obtained by accounting for observation o_{t+1} in state s_j, i.e. by multiplying by the probability $b_j(o_{t+1})$. Finally step 3) gives the desired $P(O|\lambda)$ as the sum of the terminal forward variables $\alpha_L(i)$. In fact $\alpha_L(i)$ is the probability of the observed sequence and that the system at time $t = L$ is in the state s_i. Hence $P(O|\lambda)$ is just the sum of the $\alpha_L(i)$'s. The time computational complexity of this algorithm is $O(N^2L)$. The forward probability calculation is based upon the lattice structure shown in figure 8.38(b). The key is that since there are only N states, all the possible state sequences will remerge into these N nodes, no matter how long the observation sequence. Notice that the calculation of $\alpha_t(i)$ involves multiplication with probabilities. All these probabilities have a value less than 1 (generally significantly less than 1), and as t starts to grow large, each term of $\alpha_t(i)$ starts to head exponentially to zero, exceed the precision range of the machine. To avoid this problem, a version of the Forward algorithm with scaling should be used. See Rabiner [1989] for more details.

8.5.4.2 Viterbi algorithm

The Viterbi algorithm, based on dynamic programming, is used to solve the structure learning problem. Given an HMM λ (i.e., given the matrices $\mathbf{A}$ and $\mathbf{B}$) and an output sequence $O = \{o(1)o(2)\dots o(L)\}$,



(a) (b)

Figure 8.38: *(a) Illustration of the sequence of operations required for the computation of the forward variable $\alpha_{t+1}(i)$. (b) Implementation of the computation of $\alpha_{t+1}(i)$ in terms of a lattice of observation t and states i.*

find the *single* best state sequence $X = \{x(1)x(2)\dots x(L\}$ which most likely generated it. To this purpose we define the quantity

$$\delta_t(i) = P\left[x(1)x(2)\dots x(t) = s_i, o(1)o(2)\dots o(t)\,|\lambda\right]$$

i.e. $\delta_t(i)$ is the best score (highest probability) along a single path at time t, which accounts for the first t observations and ends in state s_i. By induction we have

$$\delta_{t+1}(i) = \max_i \left[\delta_t(i)a_{ij}\right] b_j(o_{t+1})$$

To actually retrieve the state sequence, we need to keep track of the argument which maximized the previous expression, for each t and j using a predecessor array $\psi_t(j)$. The complete procedure of Viterbi algorithm is

```
1. Initialization
   for $1 \le i \le N$
       $\delta_1(i) = \pi(i)b_i(o_1)$
       $\psi_1(i) = 0$
2. Induction
   for $1 \le t \le L-1$
       for $1 \le j \le N$
           $\delta_{t+1}(j) = \max_i \left[\delta_t(i)a_{ij}\right] b_j(o_{t+1})$
           $\psi_{t+1}(j) = \text{argmax}_i \left[\delta_t(i)a_{ij}\right]$
3. Termination
   $P^* = \max_i \left[\delta_T(i)\right]$
   $x^*(T) = \text{argmax}_i \left[\delta_T(i)\right]$
4. Path backtracking
   for $t = L-1$ downto 1
       $x^*(t) = \psi_{t+1}(x^*_{t+1})$
```

Notice that the structure of Viterbi algorithm is similar in implementation to forward algorithm. The major difference is the maximization over the previous states which is used in place of the summing procedure in forward algorithm. Both algorithms used the lattice computational structure of figure 8.38(b) and have computational complexity N^2L. Also Viterbi algorithm presents the problem of multiplicationof probabilities. One way to avoid this is to take the logarithm of the model parameters, giving that the multiplications become additions. The induction thus becomes

$$\log[\delta_{t+1}(i)] = \max_i(\log[\delta_t(i)] + \log[a_{ij}] + \log[b_j(o_{t+1})]$$

Obviously will this logarithm become a problem when some model parameters has zeros present. This is often the case for A and π and can be avoided by adding a small number to the matrixes. See Rabiner [1989] for more details.

To get a better insight of how the Viterbi (and the alternative Viterbi) works, consider a model with $N = 3$ states and an observation of length $L = 8$. In the initialization ($t = 1$) is $\delta_1(1)$, $\delta_1(2)$ and $\delta_1(3)$ found. Lets assume that $\delta_1(2)$ is the maximum. Next time ($t = 2$) three variables will be used namely $\delta_2(1)$, $\delta_2(2)$ and $\delta_2(3)$. Lets assume that $\delta_2(1)$ is now the maximum. In the same manner will the following variables $\delta_3(3)$, $\delta_4(2)$, $\delta_5(2)$, $\delta_6(1)$, $\delta_7(3)$ and $\delta_8(3)$ be the maximum at their time, see Fig.8.39. This algorithm is an example of what is called the Breadth First Search (Viterbi employs this essentially). In fact it follows the principle: "Do not go to the next time instant $t + 1$ until the nodes at at time T are all expanded".

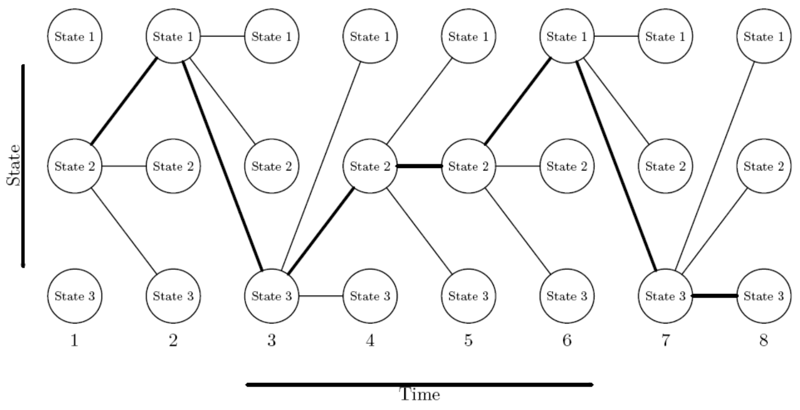


Figure 8.39: *Example of Viterbi search.*

8.6 Appendix

8.6.1 Generative Theory of Tonal Music of Lerdahl and Jackendorf

Lerdahl and Jackendoff (1983) developed a model called Generative Theory of Tonal Music (GTTM). This model offers a complementary approach to understanding melodies, based on a hierarchical structure of musical cognition. According to this theory music is built from an inventory of notes and a set of rules. The rules assemble notes into a sequence and organize them into hierarchical structures of music cognition. To understand a piece of music means to assemble these mental structures as we listen to the piece.

It seeks to elucidate a number of perceptual characteristics of tonal music - segmentation, periodicity, differential degrees of importance being accorded to the components of a musical passage or work, the flow of tension and relaxation as a work unfolds - by employing four distinct analytical levels, each with its own more-or-less formal analytical principles, or production rules. These production