

Introduction to Markov systems

1. Introduction

Up to now, we have talked a lot about building statistical models from data. However, throughout our discussion thus far, we have made the sometimes implicit, simplifying assumption that our individual data samples (or trials) are statistically independent. Let,

$$\{X_1, X_2, \dots\} \tag{1}$$

be a sequence of discrete random variables. Then the $\{X_j\}$ random variables are said to be independent if and only if,

$$P(X_t = x_t | X_{t-1} = x_{t-1}, X_{t-2} = x_{t-2}, \dots, X_1 = x_1) = P(X_t = x_t) \tag{2}$$

(Equation (2) generalizes trivially to sequences of continuous and multivariate random variables.)

Some classic examples of independent events are (1) a series of coin tosses, (2) a series of dice rolls, etc. In many instances, however, the independence assumption is not a good assumption, especially when we deal with sequential data generated over time. Consider, for example, the following random variables:

1. A person's weight.
2. Acoustic signals in speech recognition.
3. Hand gestures.
4. Robot trajectories.
5. The weather.
6. The state of a dynamic system.

In each of the above examples, knowledge of the random variable at previous times $\{t-1, t-2, \dots\}$ gives us useful information about the random variable at time t . If I know that my weight on day $(t-1)$ is 150 lb., then my weight on day t is clearly constrained to be in some interval around 150 lb. Similarly, if I measure the position of my mobile robot at time $t-1$, (x_{t-1}, y_{t-1}) , then the measurement of the robot's position at time t is constrained to be in some neighborhood of (x_{t-1}, y_{t-1}) ,

$$(x_t, y_t) = (x_{t-1}, y_{t-1}) \pm (\delta x, \delta y) \tag{3}$$

where δx and δy are determined by the robot's maximum acceleration and the robot's sensor noise models. When statistical independence is not a good assumption, the following approximation is often made:

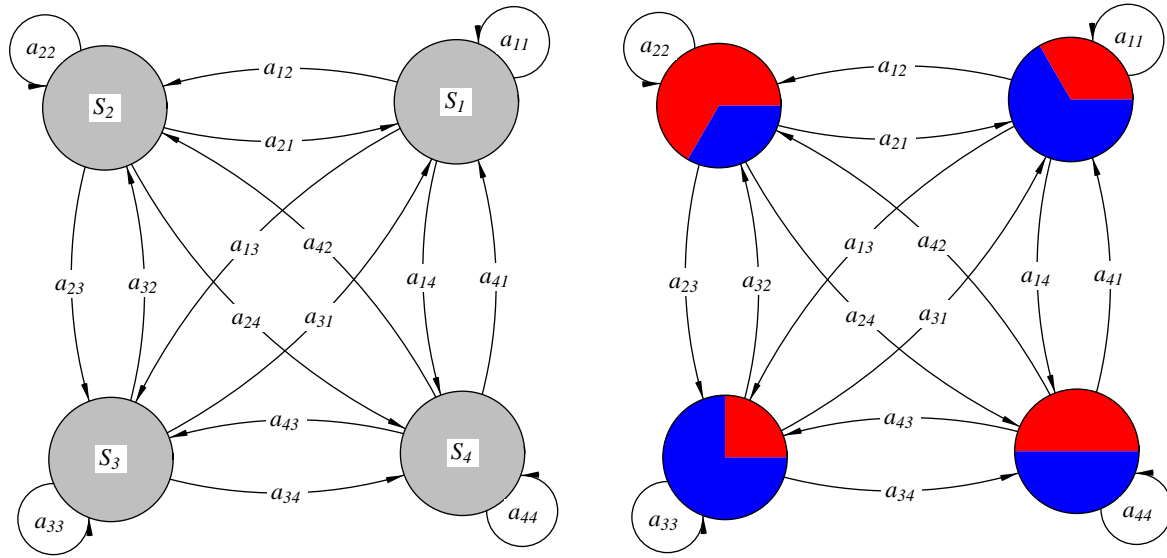
$$P(X_t = x_t | X_{t-1} = x_{t-1}, X_{t-2} = x_{t-2}, \dots, X_1 = x_1) = P(X_t = x_t | X_{t-1} = x_{t-1}) \tag{4}$$

In other words, we assume that the outcome of the t th measurement (or trial) is dependent on the outcome of the $(t-1)$ th measurement, but is conditionally independent of the outcomes of all previous measurements at times $\{t-2, t-3, \dots, 1\}$ given the outcome of the $(t-1)$ th measurement. This property is universally known as the Markov property. (The Russian mathematician Andrei Andreivich Markov was the first to extensively study stochastic systems with property (4) in the early part of the 20th century.)

Note that although the Markov assumption in equation (4) only relaxes the independence assumption in equation (2) slightly, it proves to be a remarkably powerful tool in modeling data with sequential dependencies. Many real systems have been modeled very successfully using the Markov property, even when the Markov assumption is only an approximation of reality (as is the case for many real systems). A recent query of the INSPEC database of scientific and engineering abstracts generated 25,000 hits for the keyword "Markov."

Models that make use of the Markov property can be broadly classified into four categories along two dimensions: (1) observability and (2) actions. Markov models that are fully observable and involve no actions (i.e. are passive) are called Markov chains or observable Markov models. Markov chains that are only partially observable (i.e. the state of the system is observable only indirectly) are called hidden Markov models (HMMs).

Figure 1 below gives simple examples of a Markov chain, and a hidden Markov model, respectively. In the Markov chain, the $\{S_i\}$ represent observable states of the system at a given time step, and the $\{a_{ij}\}$ represent probabilistic transitions between the states. In the hidden Markov model (HMM), the states $\{S_i\}$ are not directly observable (i.e. are hidden), but are only indirectly observed through a stochastic output observable (in this case, the color red or blue). The probability of observing a given observable (e.g red or blue) is state-dependent; in the figure below, for example, there is a one-third probability of observing red and a two-third probability of observing blue in state S_1 .



four-state observable Markov model **Figure 1** four-state, two-observable HMM

Markov systems where actions (by some agent) influence the state of the system at the next time step are studied in the broad area of reinforcement learning and are referred to as Markov Decision Processes. The table below summarizes the different types of Markov systems.

		<i>Actions</i>	
		<i>Passive</i>	<i>Choose actions</i>
<i>Observability</i>	<i>Fully observable</i>	Observable Markov Models	Markov Decision Processes (MDPs)
	<i>Partially observable</i>	Hidden Markov Models (HMMs)	Partially Observable Markov Decision Processes (POMDPs)

Before formalizing the notion of a Markov chain and a hidden Markov model, we list some of the successful real-world applications of HMMs:

1. Speech recognition
2. Language modeling
3. Gesture recognition (e.g. sign language)
4. Hand-writing recognition
5. Facial-expression recognition (e.g. sign language)
6. Human skill modeling (e.g. surgical procedures)
7. Human control strategy analysis (e.g. driving)
8. Robot control (e.g. autonomous driving)
9. And others...

2. Observable Markov models

A. Definitions

Consider a system that consists of N distinct states,

$$\{S_i\}, i \in \{1, \dots, N\}, \quad (5)$$

with the following properties. At regularly spaced, discrete-time intervals, the system undergoes a change of state (possibly back to its present state). Defining $\{q_t\}$, $t \in \{1, 2, \dots\}$, as the state of the system at time t , assume that the state transitions of this system are governed by first-order Markov state-conditional probabilities, such that,

$$P(q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \dots) = P(q_t = S_j | q_{t-1} = S_i). \quad (6)$$

Moreover, assume that the state-transition probabilities are stationary, such that,

$$P(q_t = S_j | q_{t-1} = S_i) = P(q_s = S_j | q_{s-1} = S_i), \forall s, t > 1. \quad (7)$$

Together, equations (5), (6) and (7) define a discrete, stationary, first-order Markov chain, or alternatively, an observable Markov model.

For notational convenience, let,

$$a_{ij} \equiv P(q_t = S_j | q_{t-1} = S_i), i, j \in \{1, \dots, N\}, \quad (8)$$

let the $N \times N$ state-transition matrix A be defined by,

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix} \quad (9)$$

where,

$$0 \leq a_{ij} \leq 1, i, j \in \{1, \dots, N\}, \quad (10)$$

$$\sum_{j=1}^N a_{ij} = 1, i \in \{1, \dots, N\}, \quad (11)$$

and let the N -length initial-state probability vector $\pi = \{\pi_i\}$, $i \in \{1, \dots, N\}$,

$$\pi_i = P(q_1 = S_i), i \in \{1, \dots, N\} \quad (12)$$

define the initial ($t = 1$) state probability distribution. Then, the Markov chain λ is completely defined by $\{A, \pi\}$.

B. Example

Consider Figure 2 below. It graphically describes a fully connected three-state Markov model ($N = 3$). (Fully connected means that $a_{ij} > 0$, $\forall i, j$.) This model could, for example, be a simple meteorological model. Assume that once a day at high noon, the weather conditions are observed and classified into one of three distinct states $\{S_1, S_2, S_3\}$ where,

$$S_1 = \text{rain or snow}, S_2 = \text{cloudy}, \text{ and } S_3 = \text{sunny}. \quad (13)$$

Furthermore, assume that the weather on any given day t is dependent only on the weather on the previous day ($t - 1$). Then a possible state-transition matrix A might be,

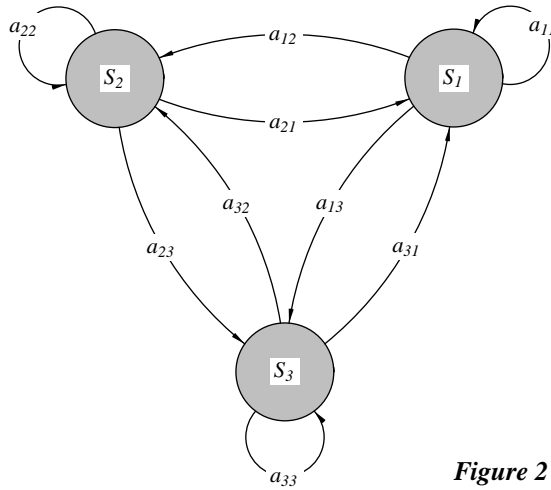


Figure 2

$$A = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix} \tag{14}$$

which is illustrated graphically in Figure 3 below.

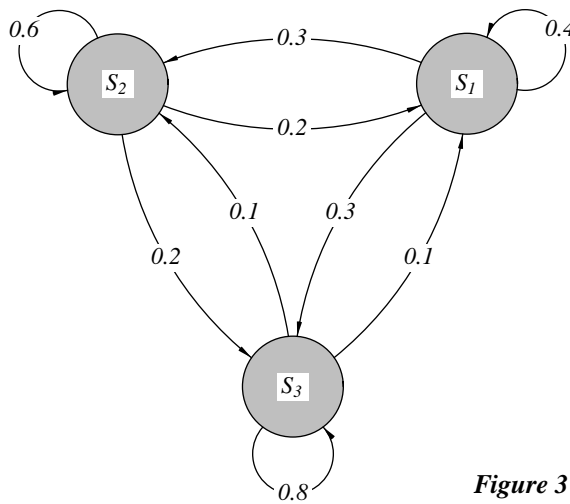


Figure 3

C. Probability of observation sequences

Given a Markov model $\lambda = \{A, \pi\}$, and an observation sequence O ,

$$O = \{q_1, q_2, \dots, q_T\} \tag{15}$$

of length T , the probability of the observation sequence O , given the model λ , $P(O|\lambda)$, is given by,

$$P(O|\lambda) = P(q_1) \times P(q_2|q_1) \times \dots \times P(q_T|q_{T-1}) \tag{16}$$

$$P(O|\lambda) = \pi_{q_1} \times a_{q_1q_2} \times \dots \times a_{q_{T-1}q_T} \tag{17}$$

(Note that in equation (17), the q_t subscripts denote the index of the state at time t .)

Example: For the example in Section 2-B, what is the probability of the observation sequence O ,

$$O = \{S_3, S_3, S_3, S_1, S_1, S_3, S_2, S_3\} \quad (18)$$

given that at time $t = 1$, the weather is sunny?

From the question,

$$\pi_1 = \pi_2 = 0, \pi_3 = P(q_1 = S_3) = 1. \quad (19)$$

Therefore,

$$P(O|\lambda) = \pi_3 a_{33} a_{33} a_{31} a_{11} a_{13} a_{32} a_{23} \quad (20)$$

$$P(O|\lambda) = 1 \times 0.8 \times 0.8 \times 0.1 \times 0.4 \times 0.3 \times 0.1 \times 0.2 \quad (21)$$

$$P(O|\lambda) \approx 1.536 \times 10^{-4} \quad (22)$$

D. Expected length of same-state sequences

Given that at time t the state of the model $\lambda = \{A, \pi\}$ is S_i , the probability that the model will stay in that state for exactly d time steps is given by,

$$P(O_d|\lambda), \quad (23)$$

where,

$$O_d = \{q_t = S_i, q_{t+1} = S_i, \dots, q_{t+d-1} = S_i, q_{t+d} = S_j\}, j \neq i. \quad (24)$$

Now,

$$P(O_d|\lambda) = P(q_t = S_i) \times \left[\prod_{\tau=t+1}^{t+d-1} P(q_\tau = S_i | q_{\tau-1} = S_i) \right] \times P(q_{t+d} \neq S_i | q_{t+d-1} = S_i) \quad (25)$$

where,

$$P(q_t = S_i) = 1 \text{ (given)}, \quad (26)$$

$$P(q_\tau = S_i | q_{\tau-1} = S_i) = a_{ii} \quad (27)$$

$$P(q_{t+d} \neq S_i | q_{t+d-1} = S_i) = 1 - P(q_{t+d} = S_i | q_{t+d-1} = S_i) = 1 - a_{ii} \quad (28)$$

so that,

$$P(O_d|A) = a_{ii}^{(d-1)} \times (1 - a_{ii}) \quad (29)$$

The expected value of d is given by,

$$E[d] = \sum_{d=1}^{\infty} d \times P(O_d|A) \quad (30)$$

$$E[d] = \sum_{d=1}^{\infty} d \times a_{ii}^{(d-1)} \times (1 - a_{ii}) \quad (31)$$

$$E[d] = \frac{1}{(1 - a_{ii})} \quad (32)$$

Example: For the example in Section 2-B,

$$E[d]_{S_1} = 5/3 \text{ days of rain or snow}, \quad (33)$$

$$E[d]_{|S_2} = 5/2 \text{ days of cloud cover,} \quad (34)$$

$$E[d]_{|S_3} = 5 \text{ days of sunshine.} \quad (35)$$

E. Probability of being in a given state at time t

Given a Markov model $\lambda = \{A, \pi\}$, the probability of being in state S_j at time t , $P(q_t = S_j)$ can be computed inductively,

$$P(q_1 = S_i) = \pi_i \text{ (by definition)} \quad (36)$$

$$P(q_t = S_j) = \sum_{i=1}^N P(q_{t-1} = S_i) a_{ij}, \quad t \in \{2, 3, \dots\} \quad (37)$$

Alternatively, we can write,

$$P(q_t = S_j) = [\pi A^{(t-1)}]_{(j)} \quad (38)$$

where, $[\mathbf{v}]_{(i)}$ denotes the i th element of the vector \mathbf{v} .

F. Fully connected Markov chains

Fully connected (finite-state) Markov chains ($a_{ij} > 0, \forall i, j$) are ergodic. That is, the limit $P(q_t = S_j)$ as t approaches infinity exists and is independent of the initial state probability vector π ,

$$\lim_{t \rightarrow \infty} P(q_t = S_j) = \lim_{t \rightarrow \infty} [\pi A^{(t-1)}]_{(j)} = P(S_j), \quad \forall \pi \quad (39)$$

Example: The limit in equation (39) typically converges for relatively small values of t . For the Markov chain in Section 2-B, for example, equation (39) converges to,

$$P(S_1) \approx 0.1818 \text{ (rain or snow)} \quad (40)$$

$$P(S_2) \approx 0.2727 \text{ (cloudy)} \quad (41)$$

$$P(S_3) \approx 0.5455 \text{ (sunny)} \quad (42)$$

for different initial values of π and $t \approx 20$, as illustrated in Figure 4, for example, with $\pi = [0.1 \ 0.1 \ 0.8]^T$. (This weather model is clearly not for Pittsburgh, PA, where sunshine is a rare occurrence indeed.)

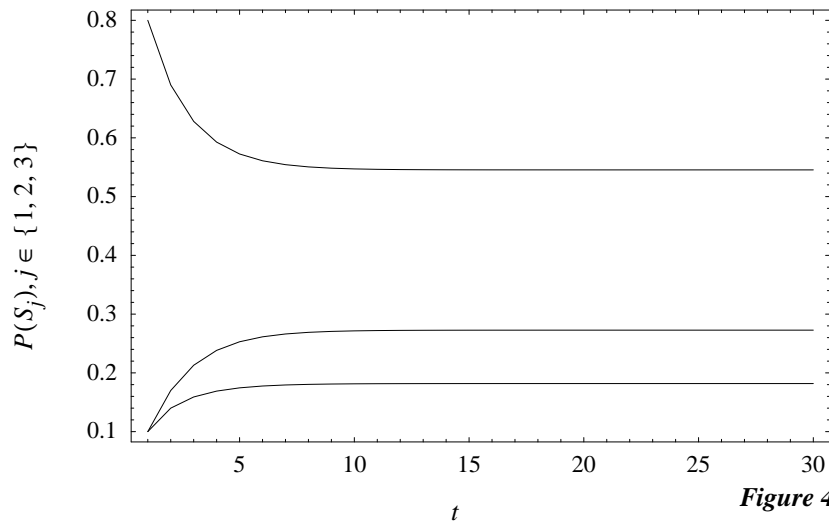


Figure 4

Example: The limit in equation (39) need not be well defined if some $a_{ij} = 0$. Consider the two-state Markov model in Figure 5 below:

The corresponding A matrix is given by,

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad (43)$$

Note that for A in equation (43),

$$A^{2t} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, A^{(2t-1)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, t \in \{1, 2, \dots\} \quad (44)$$

Therefore, the limit in equation (39) does not exist.

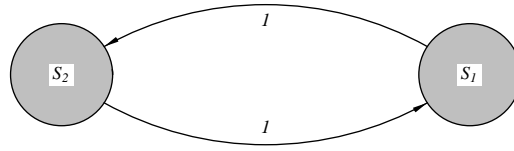


Figure 5

G. Partially connected Markov chains

When some transition probabilities $a_{ij} = 0$, states may become transient or absorbing. A transient state S_j is defined by the property,

$$\lim_{t \rightarrow \infty} P(q_t = S_j) = 0, \quad (45)$$

while an absorbing state S_j is defined by the property,

$$\lim_{t \rightarrow \infty} P(q_t = S_j) = 1 \quad (46)$$

Example: Consider the three-state Markov model in Figure 6 below, where $0 < a_{11}, a_{12}, a_{21}, a_{22}, a_{23} < 1$, $a_{13} = a_{31} = a_{32} = 0$, and $a_{33} = 1$.

The states S_1 and S_2 are transient, while the state S_3 is absorbing.

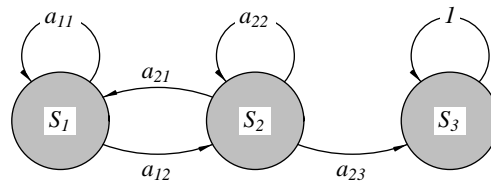


Figure 6

H. Maximum-likelihood estimation for observable Markov models

Given an observation sequence O ,

$$O = \{q_1, q_2, \dots, q_T\} \quad (47)$$

The maximum-likelihood estimate of the parameters in the Markov model $\lambda = \{A, \pi\}$ is given by,

$$a_{ij} = \frac{\text{number of transitions from state } S_i \text{ to state } S_j}{\text{number of transitions from state } S_i} \quad (48)$$

$$\pi_i = \begin{cases} 1, & \text{if } q_1 = S_i \\ 0, & \text{otherwise} \end{cases} \quad (49)$$

3. Hidden Markov models (HMMs)

A. Introduction

As we have seen, in Markov chains (or observable Markov models), it is assumed that we can observe the state q_t at each time step t . In hidden Markov models, we remove that condition. In other words, we assume that the state at each time step is *not* observable; rather, we can only observe the state at each time step t indirectly through some observable O_t , which is related to the state q_t at time step t through some output probability distribution. That is, a hidden Markov model is a doubly stochastic model, where both state transitions as well as output observables are governed by probabilistic distributions. Before we formalize this notion, let us look at a simple example.

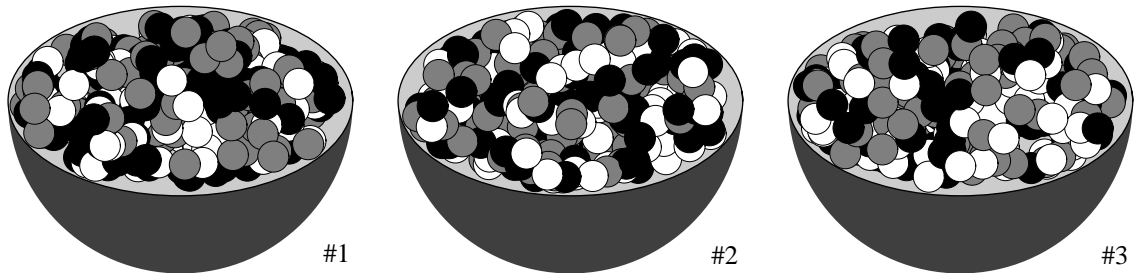


Figure 8

Suppose that we have N large urns (bowls), that are filled with differently colored balls. In Figure 2, $N = 3$, and there are three colors: (1) white, (2) gray and (3) black. We now generate a sequence of balls using the following procedure. We first pick one of the three urns at random, according to some stationary (fixed) probability distribution. Next, we pick a ball a random from that urn, and observe it's color. The ball is then put back in the urn from which it was selected. Subsequently, a new urn is selected according to some stationary probability distribution associated with the current urn. That is, which urn is selected next is conditional on only the current urn. Once again, we pick a ball from the new urn, and the process is repeated to generate an observation sequence $\{O_t\}$, $t \in \{1, 2, \dots, T\}$ of colors. A typical observation sequence is plotted in Figure 9 for $T = 15$.



Figure 9

Graphically, we can represent this process as a hidden Markov model (HMM), as shown in Figure 10. Each urn is represented by a state that is connected to all other states through probabilistic transitions. Each state also has some output probability distribution (e.g. ratio of black, gray and white balls in each urn) associated with it. This output probability distribution for each state is indicated graphically in Figure 10. For example, in state 1 (the left-most state), there are 1/4 white balls, 1/4 gray balls and 1/2 black balls.

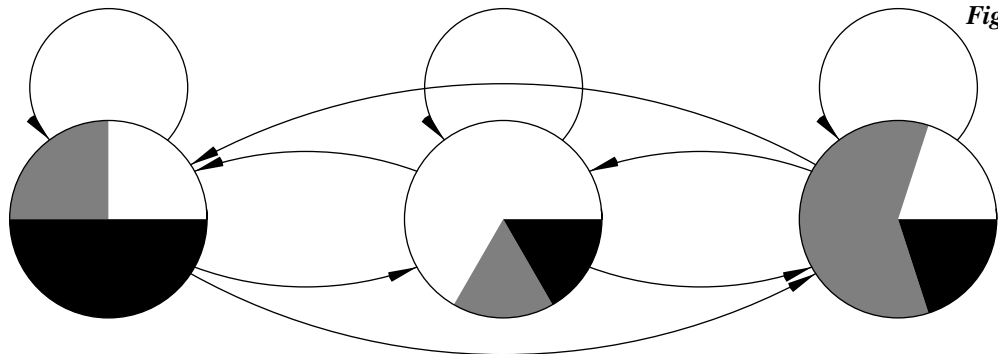


Figure 10

Note, that in this process, we can only observe the color of each ball, not the urn from which the ball was picked. Hence the name *hidden* Markov model, indicating that we cannot directly observe the underlying sequence of states that generated the observation sequence. For the sample observation sequence shown above, many different underlying state sequences are possible. In fact, there are a total of $3^{15} \approx 14.3 \times 10^6$ total possible sequences.

B. Definition of an HMM

While the above example may appear contrived, it does have all the necessary properties of a hidden Markov model. In order to define the notion of an HMM formally, we introduce the following nomenclature:

$$S_i = \text{state } i, \quad (54)$$

$$v_k = k\text{th observable}, \quad (55)$$

$$q_t = \text{state of the system at time step } t, \quad (56)$$

$$O_t = \text{observable that is observed at time step } t, t \in \{1, \dots, T\}. \quad (57)$$

Given this notation, a discrete-output¹ HMM λ is completely defined by $\lambda = \{A, B, \pi\}$, where,

$$N = \# \text{ of states in the model}, \quad (58)$$

$$L = \# \text{ of possible observables } (M \text{ in Rabiner tutorial paper [1]}), \quad (59)$$

$$A = N \times N \text{ state transition matrix with elements } \{a_{ij}\}, \quad (60)$$

$$\{a_{ij}\} = P(q_{t+1} = S_j | q_t = S_i), i, j \in \{1, \dots, N\}, \quad (61)$$

$$B = L \times N \text{ output probability distribution matrix with elements } \{b_{kj}\} = \{b_j(k)\}, \quad (62)$$

$$\{b_j(k)\} = P(O_t = v_k | q_t = S_j), j \in \{1, \dots, N\}, k \in \{1, \dots, L\} \quad (63)$$

$$\pi = N\text{-length initial state probability vector with elements } \pi_i, \quad (64)$$

$$\{\pi_i\} = P(q_1 = S_i), i \in \{1, \dots, N\} \quad (65)$$

The parameters (probabilities) of the hidden Markov model are stationary (independent of time).

Thus, the hidden Markov model is different from the observable Markov model in that we have added the output probability distribution matrix B . This makes it a much more powerful parametric formalism for modeling sequential, Markovian processes, since B can model any arbitrary discrete distribution. Of course, the B matrix also adds a large number of parameters to the model, so typically, more data is required to reliably train hidden Markov models vs. observable Markov models. For example, if $N = 5$ and $L = 100$, the observable Markov model has,

$$N^2 + N = 30 \text{ free parameters}, \quad (66)$$

while, the hidden Markov model has,

$$N^2 + N + N \times L = 530 \text{ free parameters}. \quad (67)$$

[Note that for an observable Markov model, the B matrix would be given by,

$$B = I_N \text{ (} N \times N \text{ identity matrix),} \quad (68)$$

and the observables v_i would be the states, so that $v_i = S_i$.]

1. There are also HMMs for continuous (continuous-valued) observables. These will be discussed later.

For the urn HMM, the output probability distribution matrix B is given by,

$$B = \begin{bmatrix} 1/4 & 3/5 & 1/5 \\ 1/4 & 1/5 & 3/5 \\ 1/2 & 1/5 & 1/5 \end{bmatrix} \begin{matrix} \text{(white)} \\ \text{(gray)} \\ \text{(black)} \end{matrix} \quad (69)$$

4. Three fundamental problems for HMMs

We can define three fundamental problems for hidden Markov models: (1) the *evaluation* problem, (2) the *decoding* problem and (3) the *training* problem.

A. Problem #1: Evaluation problem

Given an observation sequence $O = \{O_t\}$, $t \in \{1, \dots, T\}$, and a hidden Markov model $\lambda = \{A, B, \pi\}$, how do we efficiently compute $P(O|\lambda)$ (i.e. probability of the observation sequence O given the HMM λ)?

B. Problem #2: Decoding problem

Given an observation sequence $O = \{O_t\}$, $t \in \{1, \dots, T\}$, and a hidden Markov model $\lambda = \{A, B, \pi\}$, how do we efficiently compute the “best” (most likely) state sequence $Q = \{q_t\}$, $t \in \{1, \dots, T\}$ (i.e. the state sequence that best explains the observation sequence O given λ)?

C. Problem #3: Training problem

Given an observation sequence $O = \{O_t\}$, $t \in \{1, \dots, T\}$, the number of states N , and the number of possible observables L (implicitly defined by O), how do we efficiently compute the maximum-likelihood estimate of the parameters of the hidden Markov model $\lambda = \{A, B, \pi\}$? In other words, what HMM λ maximizes $P(O|\lambda)$?

D. Discussion

Before we derive the solutions for these three problems, it is worthwhile to examine where each of the three problems would come up in a real-world application of hidden Markov models. Historically, one of the first application areas of HMMs was in speech recognition, so we will choose that as our example. Specifically, we will be looking at isolated spoken word recognition, as illustrated in Figure 11 on the following page. In this context, we can assign physical meaning to the hidden states as the phonemes of the words in the vocabulary of the word recognizer.

A simple isolated spoken-word recognizer can be constructed as follows. First, we decide which words $\{W_i\}$, $i \in \{1, \dots, M\}$, we would like our system to be able to recognize. For example, we might be interested in recognizing the 10 spoken words corresponding to “zero,” “one,” ..., and “nine,” for an automated phone system application. Next, we record hundreds or thousands of labeled examples of different individuals saying each of the words out loud. These data will serve as our training data. For each word W_i we train a corresponding HMM λ_i to model that word (problem #3 above).

Before using the discrete hidden Markov models, we first have to decide how we will convert the time-sampled, continuous-valued acoustic speech signals to sequences of discrete observables. This conversion typically involves two steps: (1) windowing and spectral preprocessing followed by (2) vector quantization. Windowing partitions the acoustic signal into possibly overlapping segments, each of which is then filtered through some spectral preprocessor (linear predictive coding, fast Fourier transform, etc.). The acoustic speech signal is thus converted to a sequence of continuous-valued spectral feature vectors $\{v_t\}$, which are then quantized to discrete observables $\{O_t\}$ through vector quantization. It is important to realize that the spectral preprocessing and VQ codebook have to be the same for all spoken words in our training data. Therefore, the VQ codebook is typically trained (using, for example, the LBG algorithm) on all the training data.

Now, suppose that we now want to recognize an unknown utterance by some individual. We first convert that acoustic signal to a sequence of observables $\{O_t\}$ following the same conversion procedure as in the training

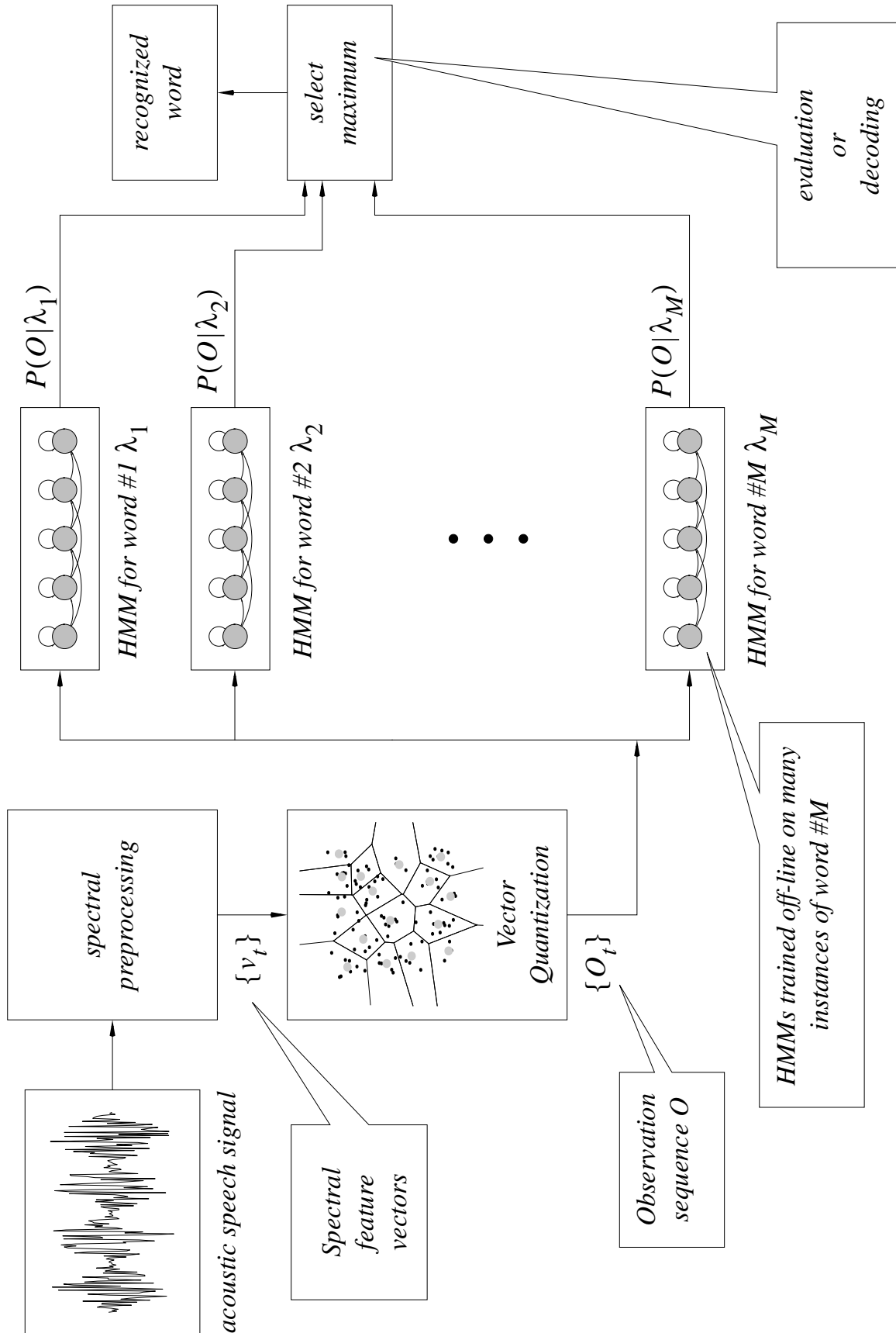


Figure 11

phase. Then, we evaluate either $P(O|\lambda_i)$ (problem #1 above), or $P(O, Q_i^*|\lambda_i)$, where Q_i^* represents the most likely state sequence (phoneme sequence) for O given λ_i (problem #2). Finally, whichever HMM λ_i yields the largest probability will correspond to the most likely word W_i .

As we shall see later, although different HMM applications vary in detail and complexity, the diagram on the previous page contains many of the essential components that make up an HMM-based system: (1) spectral preprocessing, (2) vector quantization and (3) a bank of HMMs.

5. Solution to the evaluation problem (problem #1)

Problem statement: Given an observation sequence $O = \{O_t\}$, $t \in \{1, \dots, T\}$, and a hidden Markov model $\lambda = \{A, B, \pi\}$, how do we efficiently compute $P(O|\lambda)$ (i.e. the probability of the observation sequence O given the HMM λ)?

A. Introduction

In hidden Markov models, the principal problem in computing $P(O|\lambda)$,

$$O = \{O_t\}, t \in \{1, \dots, T\}, \quad (70)$$

$$\lambda = \{A, B, \pi\}, \quad (71)$$

is that we do not know what underlying state sequence Q ,

$$Q = \{q_t\}, t \in \{1, \dots, T\} \quad (72)$$

generated the given observation sequence O . If we assume a specific underlying state sequence Q , then the problem is much simpler, since,

$$P(O|Q, \lambda) = \prod_{t=1}^T P(O_t|q_t, \lambda) = \prod_{t=1}^T b_{q_t}(O_t). \quad (73)$$

[Recall that in the definition of hidden Markov models, the observation at time t , O_t , is dependent only on the state at time t , q_t .] The probability of any specific state sequence Q can be expressed in terms of the HMM parameters as,

$$P(Q|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \dots a_{q_{T-1} q_T} = \pi_{q_1} \prod_{t=2}^T a_{q_{t-1} q_t} \quad (74)$$

[This is identical to observable Markov models.] The joint probability of O and Q given λ can be written as,

$$P(O, Q|\lambda) = P(O|Q, \lambda)P(Q|\lambda) \quad (75)$$

so that,

$$P(O|\lambda) = \sum_Q P(O, Q|\lambda) = \sum_Q P(O|Q, \lambda)P(Q|\lambda) \quad (76)$$

Although equation (76) gives a computable expression for $P(O|\lambda)$, it does not offer a *practical* algorithm for computing $P(O|\lambda)$. Since there are N^T possible state sequences, each of which requires order $2T$ operations, equation (76) requires on the order of $2TN^T$ total operations. Even for very moderately sized problems this is unreasonable. For example, if $N = 5$ and $T = 100$, we would require approximately,

$$2 \times 100 \times 5^{100} \approx 10^{72} \text{ operations.} \quad (77)$$

Clearly, we require a more efficient formulation for evaluating $P(O|\lambda)$. The *forward algorithm*, described below, does just that. A related algorithm, the *backward algorithm*, while not explicitly used to compute $P(O|\lambda)$, will be used later to efficiently solve the training problem, and is therefore also presented below. Together, these two algorithms are sometimes referred to as the *forward-backward algorithm*.

B. The forward algorithm

Let's define the following "forward" variable $\alpha_t(i)$:

$$\alpha_t(i) = P(O_1, \dots, O_t, q_t = S_i | \lambda), \quad (78)$$

which denotes the probability of the partial observation sequence $\{O_1, \dots, O_t\}$ and being in state S_i at time step t given the model λ . The $\alpha_t(i)$ variables can be computed inductively, and from them, $P(O|\lambda)$ is easily evaluated. The forward algorithm is defined below:

1. Initialization:

$$\alpha_1(i) = P(O_1, q_1 = S_i | \lambda) \quad (79)$$

$$\alpha_1(i) = P(O_1 | q_1 = S_i, \lambda) P(q_1 = S_i | \lambda) \quad (80)$$

$$\alpha_1(i) = \pi_i b_i(O_1), \quad i \in \{1, \dots, N\}. \quad (81)$$

2. Induction:

$$\alpha_{t+1}(j) = P(O_1, \dots, O_{t+1}, q_{t+1} = S_j | \lambda) \quad (82)$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^N P(O_1, \dots, O_t, q_t = S_i | \lambda) P(q_{t+1} = S_j | q_t = S_i, \lambda) \right] P(O_{t+1} | q_{t+1} = S_j, \lambda) \quad (83)$$

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^N \alpha_t(i) a_{ij} \right) b_j(O_{t+1}), \quad t \in \{1, \dots, T\}, j \in \{1, \dots, N\}. \quad (84)$$

3. Completion:

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i) \quad [\text{by definition (78)}] \quad (85)$$

The computation of $\alpha_{t+1}(j)$ in the induction step above (step 2) accounts for all possible state transitions to state S_j from time step t to time step $t+1$, and the observable O_{t+1} at time step $t+1$. Figure 12 below illustrates the induction step graphically.

For the same values of N and T as before ($N = 5, T = 100$), the computation of $P(O|\lambda)$ now only takes on the order of,

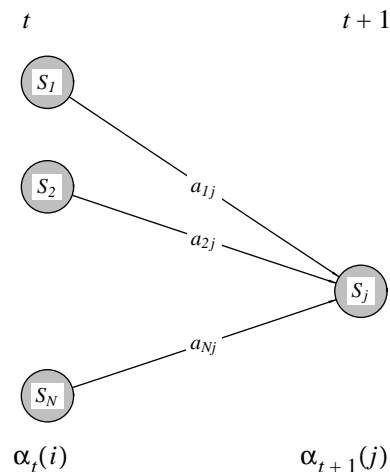


Figure 12

$$N^2T = 2500 \text{ operations,} \quad (86)$$

as opposed to 10^{72} operations. Thus, the forward algorithm offers a practical and efficient means for solving the evaluation problem in hidden Markov models.¹

C. The backward algorithm

While the backward algorithm is not used explicitly to solve the evaluation problem, it will be used later for solving the training problem in conjunction with the forward algorithm. Because of its similarity to the forward algorithm, it is presented now, however. First, we define the “backward” variables $\beta_t(i)$ (similar to the forward variables) to be,

$$\beta_t(i) = P(O_{t+1}, \dots, O_T | q_t = S_i, \lambda) \quad (87)$$

which denotes the probability of the partial observation sequence $\{O_{t+1}, \dots, O_T\}$ given that at time step t the model is in state S_i and given the model λ . As with the forward variables $\alpha_t(i)$, the backward variables $\beta_t(i)$ can be computed inductively. The backward algorithm is defined below:

1. Initialization:

$$\beta_T(i) \equiv 1, \quad i \in \{1, \dots, N\} \quad (88)$$

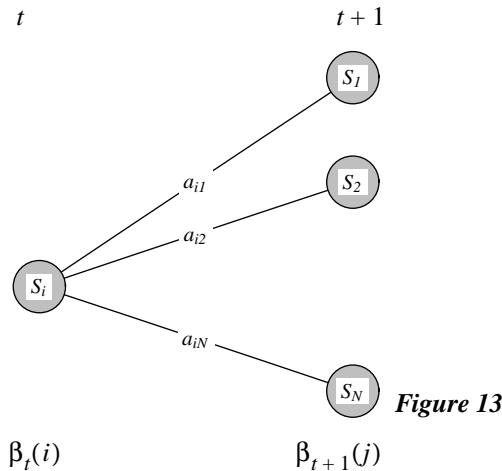
Note that equation (88) *arbitrarily* defines $\beta_T(i)$ such that,

$$P(O|\lambda) = \sum_{i=1}^N \alpha_t(i)\beta_t(i), \quad t \in \{1, \dots, T\} \quad (89)$$

2. Induction:

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \quad t \in \{T-1, T-2, \dots, 1\}, \quad i \in \{1, \dots, N\}. \quad (90)$$

Equation (90) is similar to the induction step of the forward algorithm, except that now we propagate the values back from the end of the observation sequence, rather than forward from the beginning of O . This is illustrated graphically in Figure 13 below. As with the forward algorithm, the backward algorithm requires on the order of N^2T operations.



1. As we shall see later, this algorithm will require slight modification through scaling for implementation on finite-precision computers in order to prevent numerical underflow.

6. Solution to the training problem (problem #3)¹

Problem statement: Given an observation sequence $O = \{O_t\}$, $t \in \{1, \dots, T\}$, the number of states N , and the number of possible observables L (implicitly defined by O), how do we efficiently compute the maximum-likelihood estimate of the parameters in the hidden Markov model $\lambda = \{A, B, \pi\}$? In other words, what HMM λ maximizes $P(O|\lambda)$?

A. Introduction

Baum and his colleagues developed the *Baum-Welch algorithm* for training hidden Markov models in the late 1960s and early 1970s. The development of their algorithm precedes the publication by Dempster, *et. al.* [2] of the general Expectation-Maximization (EM) algorithm (1977). It is interesting to note, however, that the Baum-Welch algorithm is simply a special case of the EM algorithm. Intuitively, we see why the EM algorithm may be applicable here, since hidden Markov models clearly have hidden information (the underlying state sequence $Q = \{q_t\}$ corresponding to the observable sequence $O = \{O_t\}$). Therefore, in developing the Baum-Welch algorithm for training HMMs, we will do so within the EM framework.

B. Initial formulation

Suppose that we knew the value of the underlying state sequence $Q = \{q_t\}$, $t \in \{1, 2, \dots, T\}$. Then, the maximum-likelihood estimate of $\lambda = \{A, B, \pi\}$ given O is straightforward and given below:

$$a_{ij} = \frac{\text{number of transitions from state } S_i \text{ to state } S_j}{\text{number of transitions from state } S_i} \quad (\text{same as for observable Markov models}) \quad (91)$$

$$b_j(k) = \frac{\text{number of times in state } S_j \text{ and observing symbol } v_k}{\text{number of times in state } S_j} \quad (92)$$

$$\pi_i = (\text{number of times in state } S_i \text{ at time } t = 1) \quad (93)$$

Unfortunately, we don't know the state sequence Q ; however, if we have a current estimate of $\lambda = \{A, B, \pi\}$, then we can compute "the expected number of..." for each of the numerators and denominators in equations (91), (92) and (93) (EM rears its ugly head again). Thus, assuming a current estimate of the hidden Markov model $\lambda = \{A, B, \pi\}$, a better (or equally good) estimate $\bar{\lambda} = \{A, B, \bar{\pi}\}$ will be given by,

$$\bar{a}_{ij} = \frac{\text{expected number of transitions from state } S_i \text{ to state } S_j}{\text{expected number of transitions from state } S_i} \quad (94)$$

$$\bar{b}_j(k) = \frac{\text{expected number of times in state } S_j \text{ and observing symbol } v_k}{\text{expected number of times in state } S_j} \quad (95)$$

$$\bar{\pi}_i = (\text{expected number of times in state } S_i \text{ at time } t = 1) \quad (96)$$

where the right-hand sides of equations (94), (95) and (96) can be expressed — as we shall see shortly — in terms of $\{A, B, \pi\}$ (i.e. the current model parameter estimates).

C. Detailed derivation

In order to compute the quantities in equations (94), (95) and (96), we need to define some hidden variables similar to the mixture-of-Gaussian problem. Thus, let's define the following hidden variables for the reestimation of the state-transition matrix A :

1. We first look at problem #3 rather than problem #2 (the decoding problem) because the solution of problem #3 makes extensive use of the forward-backward algorithm (i.e. solution to problem #1).

$$y_t(i, j) = \begin{cases} 1, & \text{at time } t, \text{ the system transitions from } S_i \text{ to } S_j \\ 0, & \text{otherwise} \end{cases} \quad (97)$$

$$y(i, j) = \text{number of transitions from state } S_i \text{ to state } S_j \quad (98)$$

so that,

$$y(i, j) = \sum_{t=1}^{T-1} y_t(i, j). \quad (99)$$

Let us also define,

$$y_t(i) = \begin{cases} 1, & \text{at time } t, \text{ the system transitions from state } S_i \\ 0, & \text{otherwise} \end{cases} \quad (100)$$

$$y(i) = \text{number of transitions from state } S_i \quad (101)$$

so that,

$$y(i) = \sum_{t=1}^{T-1} y_t(i). \quad (102)$$

Similarly, let us define the following hidden variables for the reestimation of the output-probability distribution matrix B :

$$z_t(j, k) = \begin{cases} 1, & \text{at time } t, \text{ the system is in state } S_j \text{ with output observable } v_k \\ 0, & \text{otherwise} \end{cases} \quad (103)$$

$$z(j, k) = \text{number of times in state } S_j \text{ and observing symbol } v_k \quad (104)$$

so that,

$$z(j, k) = \sum_{t=1}^T z_t(j, k). \quad (105)$$

Let us also define,

$$z_t(j) = \begin{cases} 1, & \text{at time } t, \text{ the system is in state } S_j \\ 0, & \text{otherwise} \end{cases} \quad (106)$$

$$z(j) = \text{number of times in state } S_j \quad (107)$$

so that,

$$z(j) = \sum_{t=1}^T z_t(j). \quad (108)$$

In terms of the above definitions, equations (94), (95) and (96) can be written as,

$$\bar{a}_{ij} = \frac{E[y(i, j)]}{E[y(i)]} \quad (109)$$

$$\bar{b}_j(k) = \frac{E[z(j, k)]}{E[z(j)]} \quad (110)$$

$$\bar{\pi}_i = E[y_1(i)] \quad (111)$$

where $E[\cdot]$ denotes the expectation operator. To complete the derivation, we need to compute the expected values in equations (109), (110) and (111). First, consider $E[y(i, j)]$:

$$E[y(i, j)] = E\left[\sum_{t=1}^{T-1} y_t(i, j)\right] = \sum_{t=1}^{T-1} E[y_t(i, j)] \quad (112)$$

$$E[y(i, j)] = \sum_{t=1}^{T-1} [0 \cdot P(y_t(i, j) = 0) + 1 \cdot P(y_t(i, j) = 1)] \quad (113)$$

$$E[y(i, j)] = \sum_{t=1}^{T-1} P(y_t(i, j) = 1) \quad (114)$$

Let us define $\xi_t(i, j)$ as,

$$\xi_t(i, j) \equiv E[y_t(i, j)] \quad (115)$$

so that,

$$\xi_t(i, j) = P(y_t(i, j) = 1) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \quad (116)$$

$$\xi_t(i, j) = \frac{P(q_t = S_i, q_{t+1} = S_j, O | \lambda)}{P(O | \lambda)} \quad (117)$$

We can compute $\xi_t(i, j)$ in terms of the forward and backward variables. Figure 14 below illustrates how this is done graphically. Recall from Section 5 that,

$$\alpha_t(i) = P(O_1, \dots, O_t, q_t = S_i | \lambda), \text{ and} \quad (118)$$

$$\beta_{t+1}(j) = P(O_{t+2}, \dots, O_T | q_{t+1} = S_j, \lambda) \quad (119)$$

so that,

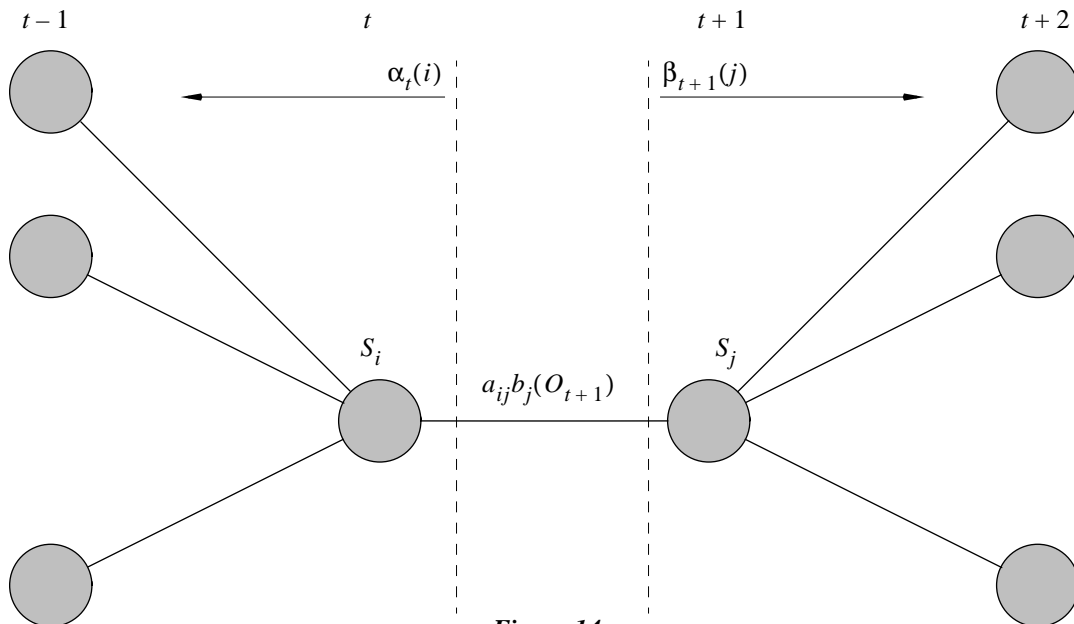


Figure 14

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} \quad (120)$$

In the numerator of equation (120), the forward variable $\alpha_t(i)$ and the backward variable $\beta_{t+1}(j)$ account for the entire observation sequence except for the observation O_{t+1} at time $t+1$ and the transition from state S_i to S_j from t to $t+1$. These are taken care of by multiplication of $b_j(O_{t+1})$ and a_{ij} , respectively.

[From the definition of $\xi_t(i, j)$, we require that,

$$\sum_{i=1}^N \sum_{j=1}^N \xi_t(i, j) = 1 \quad (121)$$

Therefore, it must be true that,

$$P(O|\lambda) = \sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \quad (122)$$

Note from equation (90) [reprinted below as equation (123)],

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \quad (123)$$

that this is consistent with equation (89) [reprinted below as equation (124)],

$$P(O|\lambda) = \sum_{i=1}^N \alpha_t(i) \beta_t(i) \quad (124)$$

Next, we consider $E[y(i)]$:

$$E[y(i)] = \sum_{t=1}^{T-1} E[y_t(i)] = \sum_{t=1}^{T-1} P(y_t(i) = 1) = \sum_{t=1}^{T-1} P(q_t = S_i | O, \lambda) \quad (125)$$

Let us define $\gamma_t(i)$ as,

$$\gamma_t(i) \equiv E[y_t(i)] \quad (126)$$

so that,

$$\gamma_t(i) = P(q_t = S_i | O, \lambda) = \sum_{j=1}^N P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \quad (127)$$

$$\gamma_t(i) = \sum_{j=1}^N \xi_t(i, j) \quad (128)$$

Combining equations (109), (112), (115), (125) and (126), we arrive at the following iterative EM update rule for the state-transition matrix A :

$$\bar{a}_{ij} = \left(\sum_{t=1}^{T-1} \xi_t(i, j) \right) / \left(\sum_{t=1}^{T-1} \gamma_t(i) \right) \quad (129)$$

where $\xi_t(i, j)$ is given in equation (120) in terms of the current model parameters, and $\gamma_t(i)$ is given in equation (128) in terms of $\xi_t(i, j)$.

The iterative EM update rule for the output-probability-distribution matrix B can be similarly derived. The numerator in equation (110) can be expressed as,

$$E[z(j, k)] = \sum_{t=1}^T P(z_t(j, k) = 1) = \sum_{t=1}^T P(q_t = S_j, O_t = v_k | O, \lambda) \quad (130)$$

$$E[z(j, k)] = \sum_{\forall O_t = v_k}^t \gamma_t(j) \quad (131)$$

Similarly, the denominator in equation (110) can be expressed as,

$$E[z(j)] = \sum_{t=1}^T P(z_t(j) = 1) = \sum_{t=1}^T P(q_t = S_j | O, \lambda) \quad (132)$$

$$E[z(j)] = \sum_{t=1}^T \gamma_t(j) \quad (133)$$

so that,

$$\bar{b}_j(k) = \left(\sum_{\forall O_t = v_k}^t \gamma_t(j) \right) / \left(\sum_{t=1}^T \gamma_t(j) \right) \quad (134)$$

Finally, from equation (111), we get that,

$$\bar{\pi}_i = E[y_1(i)] = \gamma_1(i). \quad (135)$$

D. Summary of results

We have now derived the Baum-Welch algorithm for iteratively training the parameters of a hidden Markov model. Given a current estimate of the HMM $\lambda = \{A, B, \pi\}$ and an observation sequence $O = \{O_t\}$, $t \in \{1, \dots, T\}$, the new estimate of the HMM is given by $\bar{\lambda} = \{A, B, \bar{\pi}\}$, where,

$$\bar{a}_{ij} = \left(\sum_{t=1}^{T-1} \xi_t(i, j) \right) / \left(\sum_{t=1}^{T-1} \gamma_t(i) \right), i, j \in \{1, \dots, N\} \quad (136)$$

$$\bar{b}_j(k) = \left(\sum_{\forall O_t = v_k}^t \gamma_t(j) \right) / \left(\sum_{t=1}^T \gamma_t(j) \right), j \in \{1, \dots, N\}, k \in \{1, \dots, L\} \quad (137)$$

$$\bar{\pi}_i = \gamma_1(i), i \in \{1, \dots, N\} \quad (138)$$

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O | \lambda)} \text{ and} \quad (139)$$

$$\gamma_t(i) = \sum_{j=1}^N \xi_t(i, j) \quad (140)$$

Note from equation (139) the key role that the forward and backward variables play in the HMM reestimation formulas. Also, note that the reestimation equations (136), (137) and (138) guarantee that,

$$\sum_{j=1}^N \bar{a}_{ij} = 1, i \in \{1, \dots, N\} \quad (141)$$

$$\sum_{k=1}^L \bar{b}_j(k) = 1, j \in \{1, \dots, N\} \quad (142)$$

$$\sum_{i=1}^N \bar{\pi}_i = 1 \quad (143)$$

after each update. Since the reestimation equations are EM equations, we are guaranteed that for each iteration,

$$P(O|\lambda) \leq P(O|\bar{\lambda}) \quad (144)$$

When $P(O|\bar{\lambda}) = P(O|\lambda)$, we are guaranteed to have reached a *local* maximum of the log-probability function. Equations (136), (137) and (138) can also be derived through direct maximization of the Q -function, $Q_{EM}(\lambda, \bar{\lambda})$ ¹ or constrained Lagrange optimization of $P(O|\lambda)$.

E. Simplification

The update equations (136) and (137) can be written in simplified form directly in terms of the forward and backward variables.

$$\bar{a}_{ij} = \frac{\frac{1}{P(O|\bar{\lambda})} \sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\frac{1}{P(O|\bar{\lambda})} \sum_{t=1}^{T-1} \alpha_t(i) \beta_t(i)} \quad (145)$$

$$\bar{b}_j(k) = \frac{\frac{1}{P(O|\bar{\lambda})} \sum_{\substack{t \\ \forall O_t = v_k}} \alpha_t(j) \beta_t(j)}{\frac{1}{P(O|\bar{\lambda})} \sum_{t=1}^T \alpha_t(j) \beta_t(j)} \quad (146)$$

Equations (145) and (146) above were derived by combining equations (123), (136), (137), (139) and (140) above. For training on single observation sequences, equations (145) and (146) reduce to,

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i) \beta_t(i)} \quad (147)$$

$$\bar{b}_j(k) = \frac{\sum_{\substack{t \\ \forall O_t = v_k}} \alpha_t(j) \beta_t(j)}{\sum_{t=1}^T \alpha_t(j) \beta_t(j)} \quad (148)$$

1. The Q in this context is different from the state-sequence Q used throughout these notes; it's just an unfortunate collision of notation.

7. Solution to the decoding problem (problem #2)

Problem statement: Given an observation sequence $O = \{O_t\}$, $t \in \{1, \dots, T\}$, and a hidden Markov model $\lambda = \{A, B, \pi\}$, how do we efficiently compute the “best” (most likely) state sequence $Q = \{q_t\}$, $t \in \{1, \dots, T\}$ (i.e. the state sequence that best explains the observation sequence O given λ)?

A. Introduction

Since the underlying state sequence $Q = \{q_t\}$ that generated the observation sequence $O = \{O_t\}$ is unknown, we would sometimes like to be able to determine the most likely state sequence to have generated O given the hidden Markov model λ . In speech recognition applications, for example, this is especially relevant, since the states of the hidden Markov model can be interpreted as phonemes.

One way to try to solve this problem is to choose the q_t which are *individually* most likely at each time step t . The $\gamma_t(i)$ variables that we defined in the previous section are helpful here,

$$\gamma_t(i) = P(q_t = S_i | O, \lambda) \text{ [see equations (126) and (127)].} \quad (149)$$

$$\gamma_t(i) = \frac{P(q_t = S_i, O | \lambda)}{P(O | \lambda)} \quad (150)$$

Since,

$$\alpha_t(i) = P(O_1, \dots, O_t, q_t = S_i | \lambda), \text{ and} \quad (151)$$

$$\beta_t(i) = P(O_{t+1}, \dots, O_T | q_t = S_i, \lambda) \quad (152)$$

the $\gamma_t(i)$ variables can be computed in terms of $\alpha_t(i)$ and $\beta_t(i)$,

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O | \lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^N \alpha_t(j)\beta_t(j)} \quad (153)$$

Now, in order to choose the individually optimal state sequence $Q^* = \{q_t^*\}$, we simply take the largest $\gamma_t(i)$ value for all t ,

$$q_t^* = \underset{i}{\operatorname{argmax}} \gamma_t(i), \quad t \in \{1, \dots, T\} \quad (154)$$

The basic problem with this solution is that equation (154) permits state sequences Q^* that may contain state transitions,

$$\{\dots, q_t = S_i, q_{t+1} = S_j, \dots\} \quad (155)$$

that are not possible given λ (i.e. $a_{ij} = 0$), so that $P(Q^* | \lambda) = 0$. A better solution, therefore is to try to find the state sequence Q that gives the single best path for the observation sequence O and the HMM λ ; in other words, we would like to find Q such that,

$$P(Q | O, \lambda) \quad (156)$$

is maximized. The *Viterbi algorithm*, developed in the next section, does just that.

B. The Viterbi algorithm

Thus, we want to maximize $P(Q | O, \lambda)$. From basic probability theory,

$$P(Q | O, \lambda) = \frac{P(Q, O | \lambda)}{P(O | \lambda)} \quad (157)$$

so that maximizing $P(Q|O, \lambda)$ is equivalent to maximizing the joint probability of O and Q given λ , $P(Q, O|\lambda)$, since $P(O|\lambda)$ is constant with respect to Q . As was the case for the evaluation problem, we could theoretically solve this problem by computing,

$$P(Q, O|\lambda), \forall Q \text{ [see equations (73), (74) and (75)]} \quad (158)$$

and then select Q^* ,

$$Q^* = \operatorname{argmax}_Q P(Q, O|\lambda) \quad (159)$$

as the optimal state sequence. Since there are a total of N^T total possible state sequences (e.g. if $N = 5$ and $T = 100$, then $N^T \approx 10^{70}$), however, equation (159) would be impractically slow to compute. Therefore, we will derive an inductive algorithm for solving the decoding problem similar in spirit to the forward-backward algorithm. This algorithm is known as the *Viterbi algorithm*.

First, let us define another variable $\delta_t(i)$,

$$\delta_t(i) = \max_{q_1, \dots, q_{t-1}} P(q_1, \dots, q_t = S_i, O_1, \dots, O_t|\lambda) \quad (160)$$

which can be interpreted as the best score (i.e. highest probability) along a single state path at time t for the first t observations and ending in state S_i . Given the definition in equation (160), the following inductive relationship exists:

$$\delta_{t+1}(j) = [\max_i \delta_t(i) a_{ij}] b_j(O_{t+1}) \quad (161)$$

Equation (161) forms the basis of the Viterbi algorithm and is very similar to the forward algorithm used to compute the $\alpha_t(i)$ variables. In order to completely specify the Viterbi algorithm, however, we will need another (Greek) variable $\psi_t(i)$ which keeps track of which i maximizes the right-hand side of equation (161) for each time step t . The complete Viterbi algorithm for recovering the most likely state sequence Q^* is given below:

1. Initialization:

$$\delta_1(i) = P(q_1 = S_i, O_1|\lambda) \quad (162)$$

$$\delta_1(i) = \pi_i b_i(O_1), i \in \{1, \dots, N\} \quad (163)$$

2. Induction:

$$\delta_t(j) = [\max_i \delta_{t-1}(i) a_{ij}] b_j(O_t), j \in \{1, \dots, N\}, t \in \{2, \dots, T\} \quad (164)$$

$$\psi_t(j) = \operatorname{argmax}_i [\delta_{t-1}(i) a_{ij}], j \in \{1, \dots, N\}, t \in \{2, \dots, T\} \quad (165)$$

3. Termination:

$$P^* = \max_Q P(Q, O|\lambda) \quad (166)$$

$$P^* = \max_i \delta_T(i) \quad (167)$$

$$q_T^* = \operatorname{argmax}_i \delta_T(i) \quad (168)$$

4. Path (state-sequence) back tracking:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), t \in \{T-1, T-2, \dots, 1\} \quad (169)$$

where $Q^* = \{q_t^*\}, t \in \{1, \dots, T\}$, is the most likely state sequence.

As is the case for the forward-backward algorithm, the Viterbi algorithm requires slight modification for implementation on finite-precision computers in order to prevent numerical underflow.

8. Hidden Markov models with continuous-valued outputs

So far, we have looked only at discrete-output HMMs, where observations consist of sequences of discrete observables $O = \{O_t\}$. Although most applications of HMMs deal with continuous-valued signals, these types of HMMs are used most often in practice because they are by far the most computationally efficient type of HMM with which to work. Nevertheless, there do exist HMMs with continuous-valued outputs, and we review two main types of continuous-valued HMMs below.

A. Continuous-output HMMs

Assume that rather than have a sequence of discrete observables $O = \{O_t\}$, we instead have a sequence of continuous-valued vectors $X = \{\mathbf{x}_t\}$, $t \in \{1, \dots, T\}$. Without preprocessing and vector quantization of the sequence X , we can no longer use the discrete-output probability matrix B . Rather, in continuous-output HMMs, we assume that each state S_j , $j \in \{1, \dots, N\}$, is represented by an output probability density function (pdf) $b_j(\mathbf{x})$ that is a mixture of Gaussians:

$$b_j(\mathbf{x}) = \sum_{k=1}^L c_{jk} b_{jk}(\mathbf{x}), \quad j \in \{1, \dots, N\}, \quad (170)$$

where,

$$b_{jk}(\mathbf{x}) = N[\mathbf{x}, \mu_{jk}, \Sigma_{jk}] \text{ (Gaussian pdf with mean } \mu_{jk} \text{ and covariance } \Sigma_{jk}), \quad (171)$$

$$c_{jk} = P(\omega_k | S_j) \text{ (probability of class } \omega_k \text{ given state } S_j), \quad (172)$$

such that,

$$\sum_{k=1}^L c_{jk} = 1 \text{ and} \quad (173)$$

$$\int_{-\infty}^{\infty} b_j(\mathbf{x}) d\mathbf{x} = 1 \quad (174)$$

Figure 2 illustrates schematically, for example, what a three-state continuous-output HMM with one-dimensional continuous-valued outputs may look like. The main drawback of continuous-output HMMs is their substantial computational complexity. Consider for example, the number of free parameters that have to be estimated during training in continuous-output HMMs vs. discrete-output HMMs.

$$a_{ij}: N \times N \text{ free parameters (same as for discrete-output HMMs)} \quad (175)$$

$$c_{jk}: N \times L \text{ free parameters (similar to } b_j(k) \text{ parameters for discrete-output HMMs)} \quad (176)$$

$$\mu_{jk}: N \times L \times d \text{ free parameters} \quad (177)$$

$$\Sigma_{jk}: N \times L \times d^2 \text{ free parameters} \quad (178)$$

Thus, from equations (177) and (178) we see that continuous HMMs have on the order of $NLd(1+d)$ more parameters to estimate, which leads to much higher computational complexity during training. It is not just training that requires much more computation, however; even evaluation of $P(X|\lambda)$ requires an order-of-magnitude increased computation, since each table lookup $b_j(O_t)$ in the discrete case is replaced by the relatively complex computation of $b_j(\mathbf{x})$,

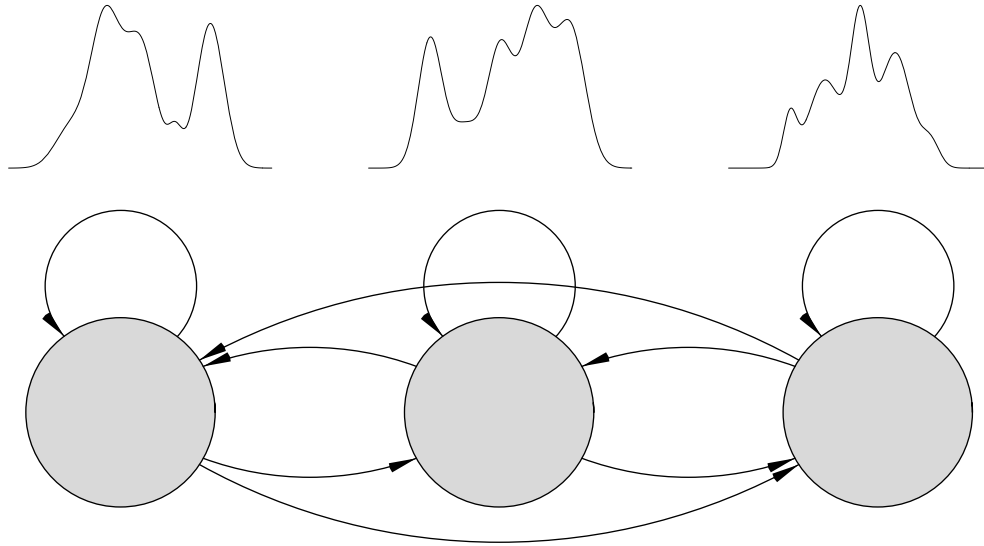


Figure 15

$$b_j(\mathbf{x}) = \sum_{k=1}^L c_{jk} b_{jk}(\mathbf{x}) \quad (179)$$

Equation (179) requires L Gaussian function evaluations, L multiplications and $L - 1$ additions, vs. one table lookup in the discrete case.

While computational complexity is a principal reason that discrete-output HMMs are often preferred to continuous-valued HMMs in practice, a secondary important reason is that continuous-valued HMMs tend to be more sensitive to initial parameter settings [3]. In other words, it is much more likely that in training continuous-valued HMMs, the model λ converges to a poor local maximum of the log-likelihood function, unless the initial parameter settings are selected with great care.

Huang [4] tried to address these concerns with development of what he referred to as “semicontinuous” HMMs, which we briefly review in the next section.

B. Semicontinuous-output HMMs

The basic idea of *semicontinuous HMMs* is to combine the computational simplicity of discrete-output HMMs with the superior modeling capacity of continuous HMMs. Given a sequence of continuous-valued vectors $X = \{\mathbf{x}_t\}$, $t \in \{1, \dots, T\}$, the distribution of the data X is first estimated as the mixture of L Gaussians, using the EM algorithms developed earlier in this course. The output probability of a vector \mathbf{x} in state S_j is then estimated as,

$$b_j(\mathbf{x}) = \sum_{k=1}^L p(\mathbf{x}|\omega_k) b_j(k) \quad (180)$$

where,

$$p(\mathbf{x}|\omega_k) = \text{the } k \text{ th Gaussian pdf estimated through the EM algorithm, and} \quad (181)$$

$$b_j(k) = P(\omega_k|S_j). \quad (182)$$

The principal difference between semicontinuous HMMs and continuous HMMs is that instead of $N \times L$ Gaussians, there are only L Gaussians in the model and that the parameters of the Gaussians are estimated prior to HMM training. This simplifies the model significantly, not in small part because a semicontinuous

HMM typically have many fewer parameters that need to be estimated. Unfortunately, semicontinuous HMMs are still significantly more computationally costly than discrete-output HMMs.

In order to clarify the difference between discrete-output and continuous-valued output HMMs, let's look at a simple example. Consider a single-state HMM with,

$$\lambda = \left\{ A = [1], B = \begin{bmatrix} 0.0 \\ 0.5 \\ 0.3 \\ 0.2 \\ 0.0 \end{bmatrix}, \pi = [1] \right\} \tag{183}$$

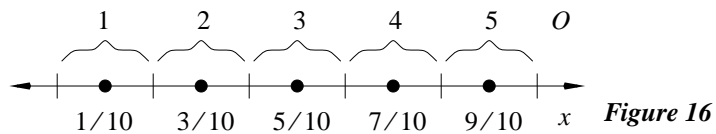
where the k th discrete observable corresponds to the real values x ,

$$\frac{k-1}{5} \leq x < \frac{k}{5}, k \in \{1, 2, 3, 4, 5\}. \tag{184}$$

In other words, the discretization in equation (184) corresponds to a VQ codebook of,

$$\left\{ \frac{1}{10}, \frac{3}{10}, \frac{5}{10}, \frac{7}{10}, \frac{9}{10} \right\} \tag{185}$$

as illustrated in Figure 16 below.

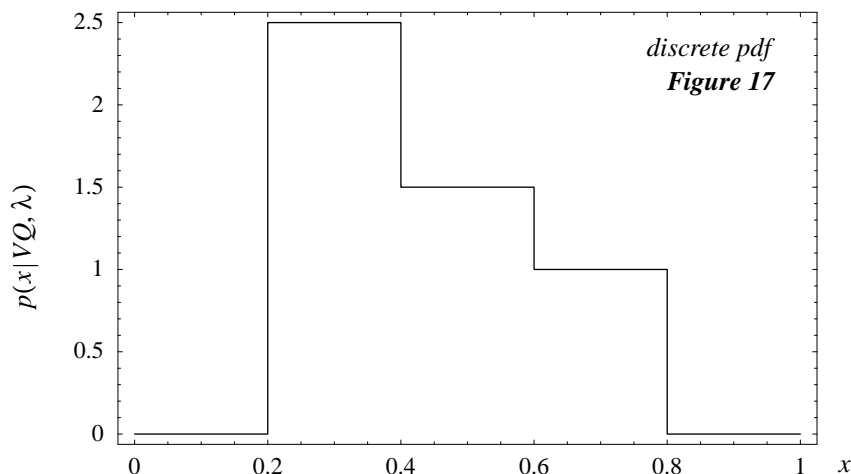


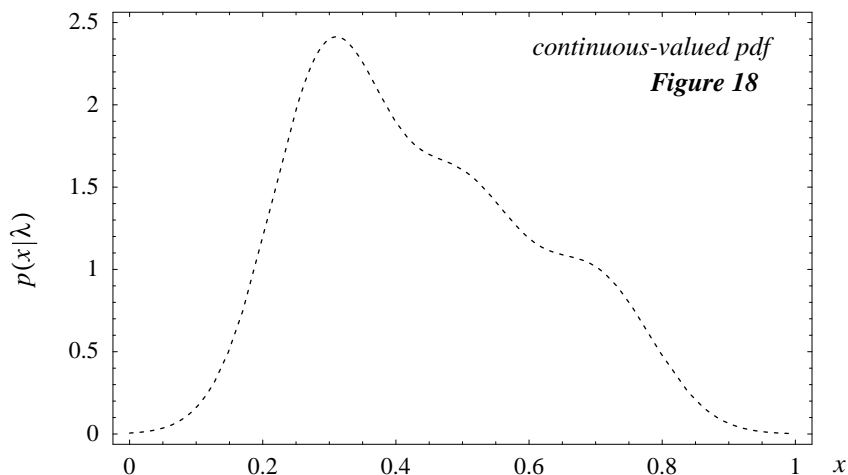
The pdf for the HMM in equation (183) and the VQ codebook in equation (185) is plotted in Figure 17 for $x \in [0, 1)$. The effects of vector quantization are clearly evident in the discontinuous profile of $p(x|VQ, \lambda)$.

Now, however, assume that we represent the continuous-valued data x not as a VQ codebook, but rather as a mixture-of-Gaussians with means μ_i ,

$$\mu_i = \frac{i}{10}, i \in \{1, 2, 3, 4, 5\} \text{ [same as centroids in VQ codebook of equation (185)]} \tag{186}$$

and variances σ_i^2 ,





$$\sigma_i^2 = 0.0075, i \in \{1, 2, 3, 4, 5\}. \quad (187)$$

The resulting pdf is plotted above in Figure 18. (Note that since λ in equation (183) is a single-state HMM, the semicontinuous and continuous HMM representations for this problem are the same.) Clearly, the continuous-valued HMM is able to represent continuous distributions more accurately.

Evaluation, decoding and training of HMMs with continuous and semicontinuous output probability distributions will not be covered here. We simply note that algorithms developed for discrete-output HMMs previously extend in a straightforward manner to the continuous and semicontinuous case.

9. Implementation issues

A. Discretization compensation

The discrete pdf for the single-state HMM in equation (183) illustrates a key shortcoming of applying discrete-output HMMs on continuous-valued data X (as is most often the case in real-world applications). Consider, for example, two real-data sequences X_1 and X_2 :

$$X_1 = \{0.21, 0.31, 0.33, 0.64, 0.59, 0.27\} \quad (188)$$

$$X_2 = \{0.19, 0.31, 0.33, 0.64, 0.59, 0.27\} \quad (189)$$

which differ only in the first element (0.21 vs. 0.19). Assuming the VQ codebook in (185) and the single-state HMM λ in (183), we can evaluate $P(X_1|\lambda)$ and $P(X_2|\lambda)$. First we convert X_1 and X_2 to sequences of discrete observables, O_1 and O_2 , respectively:

$$O_1 = \{2, 2, 2, 4, 3, 2\} \quad (190)$$

$$O_2 = \{1, 2, 2, 4, 3, 2\} \quad (191)$$

Then, evaluating the probabilities of each sequence given λ ,

$$\ln P(X_1|\lambda) = \ln P(O_1|\lambda) = 4\ln(0.5) + \ln(0.3) + \ln(0.2) = -5.586 \quad (192)$$

$$\ln P(X_2|\lambda) = \ln P(O_2|\lambda) = \ln(0.0) + 3\ln(0.5) + \ln(0.3) + \ln(0.2) = -\infty \quad (193)$$

Thus, even though the two real sequences X_1 and X_2 are almost identical, they evaluate to radically different log-probabilities. Note that the same result could have been obtained for arbitrary long sequences X_1 and X_2 , which differ only in one element by some small ϵ .

This singularity result is not desirable; it suggests that discrete-output HMMs may not be very robust to noise. Remember that HMMs are necessarily trained on *finite*-length sequences, so that rare events with non-zero probability may be possible yet, at the same time, may not be reflected in the data (i.e. may not have been observed). The probabilities corresponding to such events will therefore converge to zero during HMM training. Alternatively, a sample sequence may have spurious readings due to sensor failure, etc., and such sequences will evaluate to zero probability on HMMs previously trained on less noisy data.

There are two basic ways to deal with this problem through *discretization compensation*: (1) semicontinuous evaluation, and (2) flooring. In semicontinuous evaluation, the HMM is first trained on discrete data (vector-quantized from real data). When new sequences of real data X need to be evaluated, we assume that the VQ codebook previously generated represents a mixture of Gaussians with some uniform variance σ^2 that can be thought of as a smoothing parameter.

By far the most common method of discretization compensation, however, is *flooring* (also referred to as smoothing). Flooring is simple and effective. As before, discrete-output HMMs are trained normally. Once the HMM training has converged, zero entries in the resulting model $\lambda = \{A, B, \pi\}$ are replaced with some small ϵ (typically around 0.0001). Rows in the A matrix, column in the B matrix, and the π vector are then renormalized to fit the probabilistic constraints,

$$\sum_{j=1}^N a_{ij} = 1, i \in \{1, \dots, N\} \quad (194)$$

$$\sum_{k=1}^L b_j(k) = 1, j \in \{1, \dots, N\} \quad (195)$$

$$\sum_{i=1}^N \pi_i = 1 \quad (196)$$

For example, the B matrix for the single-state HMM λ in equation (183) would be modified for $\epsilon = 0.01$,

$$B_f = \begin{bmatrix} 0.01/1.02 \\ 0.50/1.02 \\ 0.30/1.02 \\ 0.20/1.02 \\ 0.01/1.02 \end{bmatrix} \quad (197)$$

through flooring. With the floored B matrix B_f , the observation sequences in equations (190) and (191) evaluate to,

$$\ln P(O_1 | \lambda) = -5.705 \quad (198)$$

$$\ln P(O_2 | \lambda) = -9.617 \quad (199)$$

Clearly, the flooring procedure ensures that observation sequences evaluate to finite log-probabilities. Note that the log-probability for O_1 changes little from -5.586 ; however, the log-probability for O_2 becomes finite. The log-probabilities in (198) and (199) still appear significantly different primarily because the fraction of near-zero observables for O_2 ($1/6$) is relatively high for this simple example. Typically, rare events that lead to zero probabilities in the hidden Markov model occur much less frequently.

B. Scaling for the forward-backward algorithm

The forward-backward algorithm, as previously presented, suffers from numerical underflow problems if implemented on any finite precision computer. Recall the forward algorithm,

1. Initialization:

$$\alpha_1(i) = \pi_i b_i(O_1), i \in \{1, \dots, N\} \quad (200)$$

2. Induction:

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^N \alpha_t(i) a_{ij} \right) b_j(O_{t+1}), t \in \{1, \dots, T-1\}, j \in \{1, \dots, N\} \quad (201)$$

3. Completion:

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i) \quad (202)$$

for evaluating $P(O|\lambda)$, and the analogous backward algorithm,

1. Initialization:

$$\beta_T(i) = 1, i \in \{1, \dots, N\} \quad (203)$$

2. Induction:

$$\beta_t(i) = \left(\sum_{j=1}^N a_{ij} b_j(O_{t+1}) \right) \beta_{t+1}(j), t \in \{T-1, \dots, 1\}, i \in \{1, \dots, N\} \quad (204)$$

Note from equations (201) and (204) that the induction steps of the forward and backward algorithms, respectively, multiply together numbers less than or equal to one. This implies that for large T (i.e. long observation sequences)

$$\alpha_T(i) \rightarrow 0 \text{ and } \beta_1(i) \rightarrow 0. \quad (205)$$

Consider, for example, the simple three-state HMM with four output observables shown in Figure 19 below. This HMM was used to generate a sample observation sequence O of $T = 100$ observables. For this observation sequence, we compute $\alpha_t(i)$, $t \in \{1, \dots, 100\}$, $i \in \{1, 2, 3\}$, and plot $\log_{10} \alpha_t(i)$ as a function of t in Figure 20. Note that even for this relatively simple HMM and relatively short observation sequence,

$$\alpha_{100}(i) \approx 10^{-60} \quad (206)$$

The numerical underflow problem becomes even more severe when the number of possible observables increases (since the $b_j(k)$ values will become smaller on average as L increases). To illustrate this, we generate another HMM with the same state-transition matrix A as before, but with an output-probability distribution matrix B with $L = 100$ (i.e. 100 possible observables). We again compute $\alpha_t(i)$, $t \in \{1, \dots, 100\}$, $i \in \{1, 2, 3\}$, and plot $\log_{10} \alpha_t(i)$ as a function of t in Figure 21. Note that now,

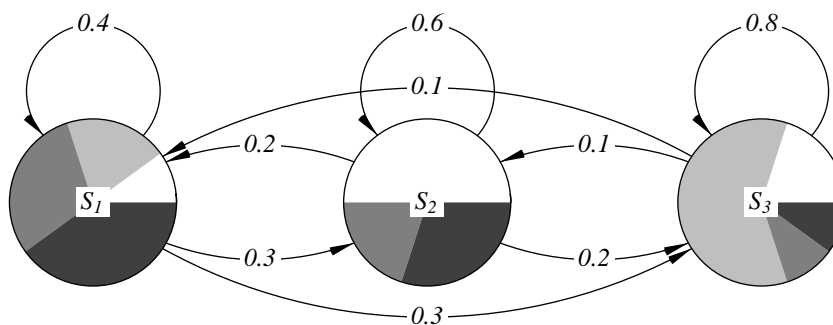
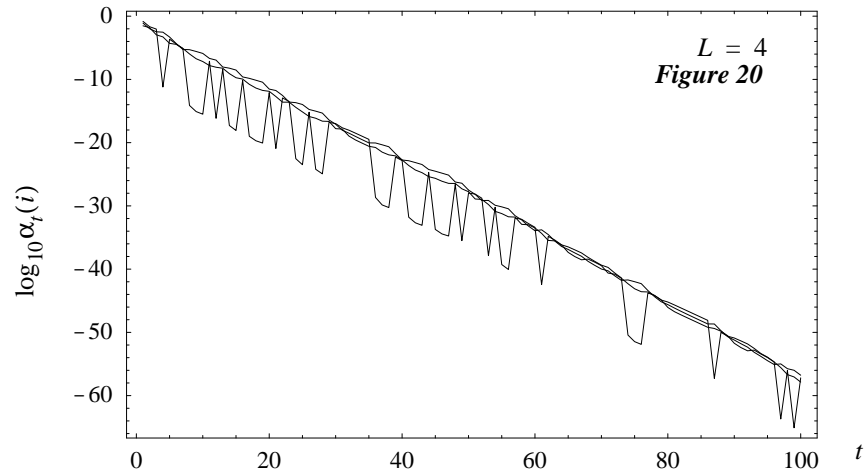


Figure 19



$$\alpha_{100}(i) \approx 10^{-180} \quad (207)$$

Clearly, for arbitrarily long observation sequences, the forward-backward algorithm will not be numerically stable. Therefore, in order to make the algorithm numerically stable, we introduce *scaling* of the forward and backward variables that will keep the values of those variables within the dynamic range of a typical floating point computer.

Below, we present the *scaled forward algorithm*:

1. Initialization:

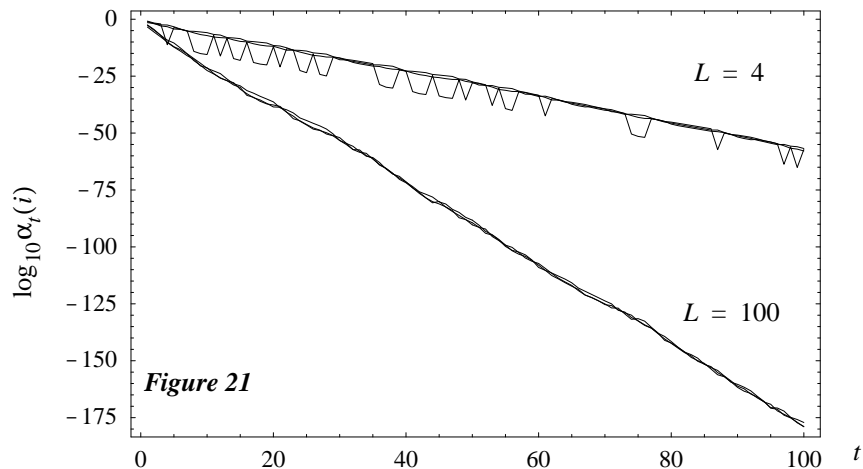
$$\tilde{\alpha}_1(i) = \pi_i b_i(O_1), \quad i \in \{1, \dots, N\} \quad (208)$$

$$\hat{\alpha}_1(i) = c_1 \tilde{\alpha}_1(i), \quad i \in \{1, \dots, N\} \quad (\text{scaling}) \quad (209)$$

2. Induction:

$$\tilde{\alpha}_{t+1}(j) = \left[\sum_{i=1}^N \hat{\alpha}_t(i) a_{ij} \right] b_j(o_{t+1}), \quad t \in \{1, \dots, T-1\}, j \in \{1, \dots, N\} \quad (210)$$

$$\hat{\alpha}_{t+1}(j) = c_{t+1} \tilde{\alpha}_{t+1}(j), \quad t \in \{1, \dots, T-1\}, j \in \{1, \dots, N\} \quad (\text{scaling}) \quad (211)$$



$$c_t = 1 / \left(\sum_{i=1}^N \tilde{\alpha}_t(i) \right), t \in \{1, \dots, T\} \text{ (scaling)} \quad (212)$$

3. Completion:

$$P(O|\lambda) = 1 / \left(\prod_{t=1}^T c_t \right) \quad (213)$$

In the scaled forward algorithm, the forward variables are scaled at each time step t by a scaling coefficient c_t so that the scaled forward variables $\hat{\alpha}_t(i)$ meet the following constraint:

$$\sum_{i=1}^N \hat{\alpha}_t(i) = 1, t \in \{1, \dots, T\} \quad (214)$$

By comparing the scaled forward algorithm [equations (208) to (213)] with the unscaled forward algorithm [equations (200) to (202)], we observe that the scaled forward variables $\hat{\alpha}_t(i)$ are related to the forward variables $\alpha_t(i)$ of the unscaled forward algorithm by the following relationship:

$$\hat{\alpha}_t(i) = \left(\prod_{\tau=1}^t c_\tau \right) \alpha_t(i) \quad (215)$$

For $t = T$,

$$\hat{\alpha}_T(i) = \left(\prod_{t=1}^T c_t \right) \alpha_T(i) \quad (216)$$

so that from equation (214),

$$\sum_{i=1}^N \hat{\alpha}_T(i) = \left(\prod_{t=1}^T c_t \right) \sum_{i=1}^N \alpha_T(i) = 1 \quad (217)$$

Since,

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i), \quad (218)$$

equation (217) can be written as,

$$\left(\prod_{t=1}^T c_t \right) P(O|\lambda) = 1 \quad (219)$$

so that we arrive at equation (213),

$$P(O|\lambda) = 1 / \left(\prod_{t=1}^T c_t \right) \quad (220)$$

For large T , $P(O|\lambda)$ will typically be very small, so that in practice we compute equation (220) instead as,

$$\log P(O|\lambda) = - \sum_{t=1}^T \log c_t \quad (221)$$

Once the scaled forward algorithm has been executed, the *scaled backward algorithm* follows:

1. Initialization:

$$\tilde{\beta}_T(i) = 1, i \in \{1, \dots, N\} \quad (222)$$

$$\hat{\beta}_T(i) = c_T \tilde{\beta}_T(i), i \in \{1, \dots, N\} \text{ (scaling)} \quad (223)$$

2. Induction:

$$\tilde{\beta}_t(i) = \sum_{j=1}^n a_{ij} b_j(o_{t+1}) \hat{\beta}_{t+1}(j) \quad t \in \{T-1, \dots, 1\}, i \in \{1, 2, \dots, N\} \quad (224)$$

$$\hat{\beta}_t(i) = c_t \tilde{\beta}_t(i), t \in \{T-1, \dots, 1\}, i \in \{1, 2, \dots, n\} \text{ (scaling)} \quad (225)$$

Note that for the scaled backward algorithm, we use the same scaling coefficients c_t computed in the scaled forward algorithm. The scaled backward variables $\hat{\beta}_t(i)$ are related to the backward variables $\beta_t(i)$ of the unscaled backward algorithm by the following relationship:

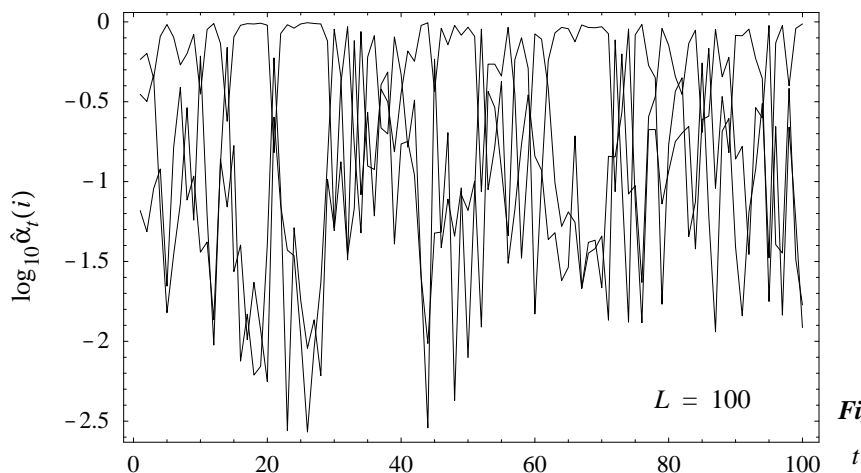
$$\hat{\beta}_t(i) = \left(\prod_{\tau=t}^T c_\tau \right) \beta_t(i) \quad (226)$$

As an example of the scaled forward-backward algorithm, let us compute $\hat{\alpha}_t(i)$, $t \in \{1, \dots, 100\}$, $i \in \{1, 2, 3\}$, and plot $\log_{10} \hat{\alpha}_t(i)$ as a function of t for the three-state HMM with $L = 100$ for which we previously computed the unscaled variables $\alpha_t(i)$ (Figure 22). Note that now the scaled forward variables stay well within the floating-point range of a typical computer, and that this result holds no matter how large T is.

C. Scaling and the Baum-Welch algorithm

Previously, we derived the following iterative training algorithm for optimizing the parameters of a hidden Markov model λ given some observation sequence O ,

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i) \beta_t(i)} \quad (227)$$

**Figure 22**

$$\bar{b}_j(k) = \frac{\sum_{\substack{t \\ \forall O_t = v_k}} \alpha_t(j)\beta_t(j)}{\sum_{t=1} \alpha_t(j)\beta_t(j)} \quad (228)$$

in terms of the unscaled forward-backward variables, where $\lambda = \{A, B, \pi\}$ is the current estimate of the model, and $\bar{\lambda} = \{\bar{A}, \bar{B}, \bar{\pi}\}$ is the new estimate of the HMM. Let us consider how the update equations (227) and (146) change if expressed in terms of the scaled forward-backward variables. Consider equation (227) first.

First let us substitute $\hat{\alpha}_t(i)$ for $\alpha_t(i)$ and $\hat{\beta}_t(i)$ for $\beta_t(i)$ in (227) and see how we need to modify the resulting equation to make it equivalent to (227):

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{t=1}^{T-1} \hat{\alpha}_t(i) \hat{\beta}_t(i)} \quad (229)$$

Now, combine equation (229) with equations (215) and (226) so that,

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \left(\prod_{\tau=1}^t c_\tau \right) \alpha_t(i) a_{ij} b_j(O_{t+1}) \left(\prod_{\tau=t+1}^T c_\tau \right) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \left(\prod_{\tau=1}^t c_\tau \right) \alpha_t(i) \left(\prod_{\tau=t}^T c_\tau \right) \beta_t(j)} \quad (230)$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \left(\prod_{\tau=1}^T c_\tau \right) \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \left(\prod_{\tau=1}^T c_\tau \right) c_t \alpha_t(i) \beta_t(i)} \quad (231)$$

From equation (220),

$$\bar{a}_{ij} = \frac{\frac{1}{P(O|\lambda)} \sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\frac{1}{P(O|\lambda)} \sum_{t=1}^{T-1} c_t \alpha_t(i) \beta_t(i)} \quad (232)$$

Therefore, to express equation (227) in terms of the scaled forward-backward variables, we simply need to divide $\hat{\alpha}_t(i)\hat{\beta}_t(j)$ in the denominator of equation (229) by c_t . The correct update equation in terms of the scaled forward-backward variables is therefore given by,

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{t=1}^{T-1} \frac{\hat{\alpha}_t(i) \hat{\beta}_t(i)}{c_t}} \quad (233)$$

Now let us consider the update for the output probability matrix B in equation (146). Again, let us substitute $\hat{\alpha}_t(i)$ for $\alpha_t(i)$ and $\hat{\beta}_t(i)$ for $\beta_t(i)$,

$$\bar{b}_j(k) \stackrel{?}{=} \frac{\sum_{t=1}^T \hat{\alpha}_t(j) \hat{\beta}_t(j)}{\sum_{t=1}^T \hat{\alpha}_t(j) \hat{\beta}_t(j)} \quad \forall O_t = v_k \quad (234)$$

Now, combine equation (234) with equations (215) and (226) so that,

$$\bar{b}_j(k) \stackrel{?}{=} \frac{\sum_{t=1}^T \left(\prod_{\tau=1}^t c_\tau \right) \alpha_t(j) \left(\prod_{\tau=t}^T c_\tau \right) \beta_t(j)}{\sum_{t=1}^T \left(\prod_{\tau=1}^t c_\tau \right) \alpha_t(j) \left(\prod_{\tau=t}^T c_\tau \right) \beta_t(j)} \quad \forall O_t = v_k \quad (235)$$

$$\bar{b}_j(k) \stackrel{?}{=} \frac{\frac{1}{P(O|\lambda)} \sum_{t=1}^T c_t \alpha_t(j) \beta_t(j)}{\frac{1}{P(O|\lambda)} \sum_{t=1}^T c_t \alpha_t(j) \beta_t(j)} \quad \forall O_t = v_k \quad (236)$$

Therefore, to express equation (146) in terms of the scaled forward-backward variables, we need to divide $\hat{\alpha}_t(i) \hat{\beta}_t(j)$ in both the numerator and denominator of equation (234) by c_t . The correct update equation in terms of the scaled forward-backward variables is therefore given by,

$$\bar{b}_j(k) = \frac{\sum_{t=1}^T \frac{\hat{\alpha}_t(j) \hat{\beta}_t(j)}{c_t}}{\sum_{t=1}^T \frac{\hat{\alpha}_t(j) \hat{\beta}_t(j)}{c_t}} \quad \forall O_t = v_k \quad (237)$$

Equations (233) and (237) represent the Baum-Welch reestimation formulas in terms of the scaled forward-backward variables.

D. Scaling and the Viterbi algorithm

The Viterbi algorithm for finding the most likely state sequence Q given O and λ was previously derived as:

1. Initialization:

$$\delta_1(i) = P(q_1 = S_i, O_1 | \lambda) \quad (238)$$

$$\delta_1(i) = \pi_i b_i(O_1), \quad i \in \{1, \dots, N\} \quad (239)$$

2. Induction:

$$\delta_t(j) = [\max_i \delta_{t-1}(i) a_{ij}] b_j(O_t), \quad j \in \{1, \dots, N\}, \quad t \in \{2, \dots, T\} \quad (240)$$

$$\psi_t(j) = \operatorname{argmax}_i [\delta_{t-1}(i) a_{ij}], \quad j \in \{1, \dots, N\}, \quad t \in \{2, \dots, T\} \quad (241)$$

3. Termination:

$$P^* = \max_Q P(Q, O|\lambda) \quad (242)$$

$$P^* = \max_i \delta_T(i) \quad (243)$$

$$q_T^* = \operatorname{argmax}_i \delta_T(i) \quad (244)$$

4. Path (state-sequence) back tracking:

$$q_t^* = \Psi_{t+1}(q_{t+1}^*), t \in \{T-1, T-2, \dots, 1\} \quad (245)$$

where $Q^* = \{q_t^*\}$, $t \in \{1, \dots, T\}$, is the most likely state sequence.

This algorithm is easily modified to eliminate numerical underflow problems by defining,

$$\phi_t(i) = \max_{q_1, \dots, q_T} \log P(q_1, \dots, q_t = S_i, O_1, \dots, O_t|\lambda) \quad (246)$$

instead of,

$$\delta_t(i) = \max_{q_1, \dots, q_T} P(q_1, \dots, q_t = S_i, O_1, \dots, O_t|\lambda) \quad (247)$$

Consequently, equation (239) changes to,

$$\phi_1(i) = \log(\pi_i) + \log[b_i(O_1)], i \in \{1, \dots, N\}; \quad (248)$$

equation (240) changes to,

$$\phi_t(j) = \max_i [\phi_{t-1}(i) + \log a_{ij}] + \log[b_j(O_t)], j \in \{1, \dots, N\}, t \in \{2, \dots, T\}; \quad (249)$$

equation (241) changes to,

$$\Psi_t(j) = \operatorname{argmax}_i [\phi_{t-1}(i) + \log a_{ij}], j \in \{1, \dots, N\}, t \in \{2, \dots, T\}; \quad (250)$$

and equation (243) changes to,

$$\log P^* = \max_i [\phi_T(i)] \quad (251)$$

E. Multiple observation sequences

In many applications, rather than train a hidden Markov model on one long observation sequence O , we instead would like to train the model λ on M short sequences $\{O^{(m)}\}$, $m \in \{1, \dots, M\}$. For example, in speech recognition applications, where each HMM may represent one spoken word, we would like to train that HMM on many different utterances of that word. In handwritten gesture recognition, where each HMM may represent one of several different gestures (e.g. written letters), we again would like to train those HMMs on many different examples of that gesture.

Training a single HMM on multiple observation sequences leads to two changes in the general HMM framework: (1) the HMM structure that is chosen for those applications, and (2) a modified training algorithm for handling multiple observation sequences.

When training on multiple, relatively short observation sequences, typically the HMM model is restricted to a left-to-right structure. Consider, for example, the left-to-right model in Figure 23.

Except for self-transitions, connections are only allowed in one direction. A fundamental property of all left-to-right HMMs is that the state transition probabilities a_{ij} are restricted to,

$$a_{ij} = 0, \forall j < i \quad (252)$$

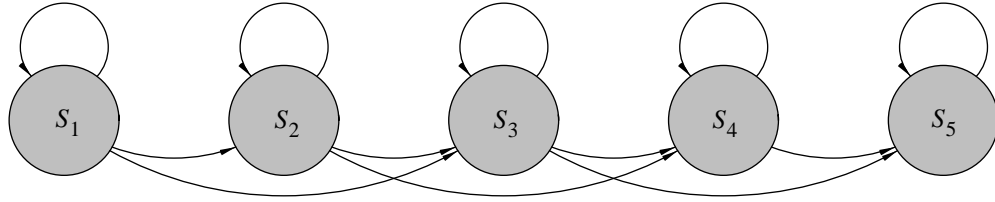


Figure 23

State transitions may further be restricted to prevent large changes in state indices. For the HMM above, for example, the state-transition matrix A is given by,

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ 0 & a_{22} & a_{23} & a_{24} & 0 \\ 0 & 0 & a_{33} & a_{34} & a_{35} \\ 0 & 0 & 0 & a_{44} & a_{45} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (253)$$

The modification of the Baum-Welch reestimation formulas for multiple observation sequences $\{O^{(m)}\}$, $m \in \{1, \dots, M\}$ is straightforward, if we assume that each observation sequence $O^{(m)}$ is independent of every other observation sequence in the training set. Our goal now is to optimize the parameters in λ to maximize,

$$P(O|\lambda) = \prod_{m=1}^M P(O^{(m)}|\lambda) \quad (254)$$

Since the reestimation formulas are based on frequencies of occurrence of various events, the reestimation formulas for multiple observation sequences are modified by adding together the individual frequencies of occurrence for each sequence. In terms of the unscaled variables,

$$\bar{a}_{ij} = \frac{\sum_{m=1}^M \frac{1}{P(O^{(m)}|\lambda)} \sum_{t=1}^{T_m-1} \alpha_t^m(i) a_{ij} b_j(O_{t+1}^{(m)}) \beta_{t+1}^m(j)}{\sum_{m=1}^M \frac{1}{P(O^{(m)}|\lambda)} \sum_{t=1}^{T_m-1} \alpha_t^m(i) \beta_t^m(i)} \quad (255)$$

$$\bar{b}_j(k) = \frac{\sum_{m=1}^M \frac{1}{P(O^{(m)}|\lambda)} \sum_{\substack{t \\ \forall O_t^{(m)} = v_k}} \alpha_t^m(j) \beta_t^m(j)}{\sum_{m=1}^M \frac{1}{P(O^{(m)}|\lambda)} \sum_{t=1}^{T_m} \alpha_t^m(j) \beta_t^m(j)} \quad (256)$$

Note that the $1/P(O^{(m)}|\lambda)$ terms were cancelled out of the numerator and denominator previously for the case of a single observation sequence. For multiple observation sequences, these terms no longer cancel out. In terms of the scaled variables,

$$\bar{a}_{ij} = \frac{\sum_{m=1}^M \sum_{t=1}^{T_m-1} \hat{\alpha}_t^m(i) a_{ij} b_j(O_{t+1}^{(m)}) \hat{\beta}_{t+1}^m(j)}{\sum_{m=1}^M \sum_{t=1}^{T_m-1} \frac{\hat{\alpha}_t^m(i) \hat{\beta}_t^m(i)}{c_t^m}} \quad (257)$$

$$\bar{b}_j(k) = \frac{\sum_{m=1}^M \sum_{\substack{t \\ \forall O_t^{(m)} = v_k}} \frac{\hat{\alpha}_t^m(j) \hat{\beta}_t^m(j)}{c_t^m}}{\sum_{m=1}^M \sum_{t=1}^{T_m} \frac{\hat{\alpha}_t^m(j) \hat{\beta}_t^m(j)}{c_t^m}} \quad (258)$$

Note that the $1/P(O^{(m)}|\lambda)$ terms no longer need to be explicitly included in equations (257) and (258), since equations (232) and (236) show that the scaled forward-backward variables implicitly include the $1/P(O^{(m)}|\lambda)$ terms. Finally, note that equations (110) and (111) in Rabiner's HMM tutorial paper [1] both contain errors, which equations (257) and (258) fix.

10. Baum-Welch convergence

Below, we illustrate some convergence issues of the Baum-Welch algorithm for a simple example and a real-world example.

A. Simple example

Here we conduct the following simple experiment to investigate how the convergence of the Baum-Welch algorithm is affected by the number of states that we assume for our model. We first generate an observation sequence O of length $T = 10,000$ for the two-observable, two-state hidden Markov model λ in Figure 24

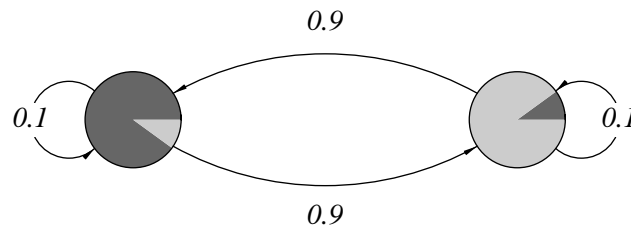


Figure 24

where,

$$A = \begin{bmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{bmatrix}, B = \begin{bmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{bmatrix}, \pi = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}. \quad (259)$$

We then train three HMMs λ_i , $i \in \{1, 2, 3\}$, from random initial parameter settings, where λ_i has $N = i$ states. The three resulting HMMs are shown in Figure 25 on the next page. Finally, we compute the normalized probability measure (normalized with respect to T , the length of the observation sequence),

$$\bar{P}(O|\lambda_i) = 10^{\log P(O|\lambda_i)/T} = P(O|\lambda_i)^{1/T} \quad (260)$$

Table 1 below reports the resulting $\bar{P}(O|\lambda_i)$ values. The final single-state HMM λ_1 cannot encode any sequential structure, and therefore captures only the aggregate distribution of observables (1/2 and 1/2). The two-state HMM λ_2 converges to the generating HMM in the above figure, which is encouraging. Although the Baum-Welch algorithm is only guaranteed to find a local maximum of the log-probability function, in this case it converges to the globally optimal value. Note also that $P(O|\lambda_i)$ improves significantly from λ_1 to

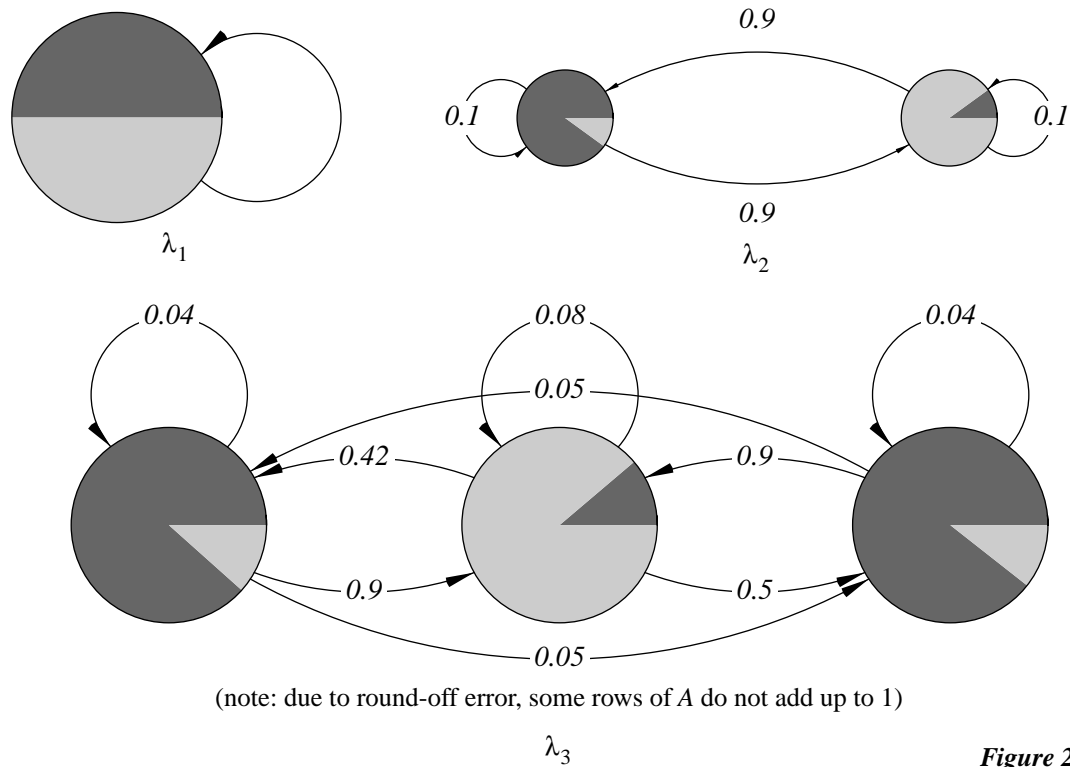


Figure 25

Table 1: Normalized probabilities for HMMs in Figure 25

i	$\bar{P}(O \lambda_i)$
1	0.5000
2	0.5875
3	0.5875

λ_2 . This implies that two states are able to capture sequential characteristics of the observation sequence O which the one-state HMM cannot capture. In this case, we know this to be true since the observation sequence O was generated from a two-state HMM.

When we increase the number of states to three, the Baum-Welch algorithm converges to λ_3 in the figure above. Although λ_3 has significantly more representational power than λ_2 , the normalized probability $P(O|\lambda_i)$ does not appreciably increase. This indicates that the two-state model λ_2 is probably sufficient for modeling the sequential properties of O , and adding more states will not improve the model substantially. In fact, note that for the model λ_3 , the single left state of λ_2 is effectively split up into two states in λ_3 .

The above results suggest that, although we frequently do not know the exact number of states that we should use for modeling specific data in real applications, we may arrive at the optimal number of states for a specific observation sequence by training models with a different number of states, and then selecting the smallest (most general) model which yields the largest normalized probability value $P(O|\lambda_i)$. To illustrate this point, we consider a more complicated example below.

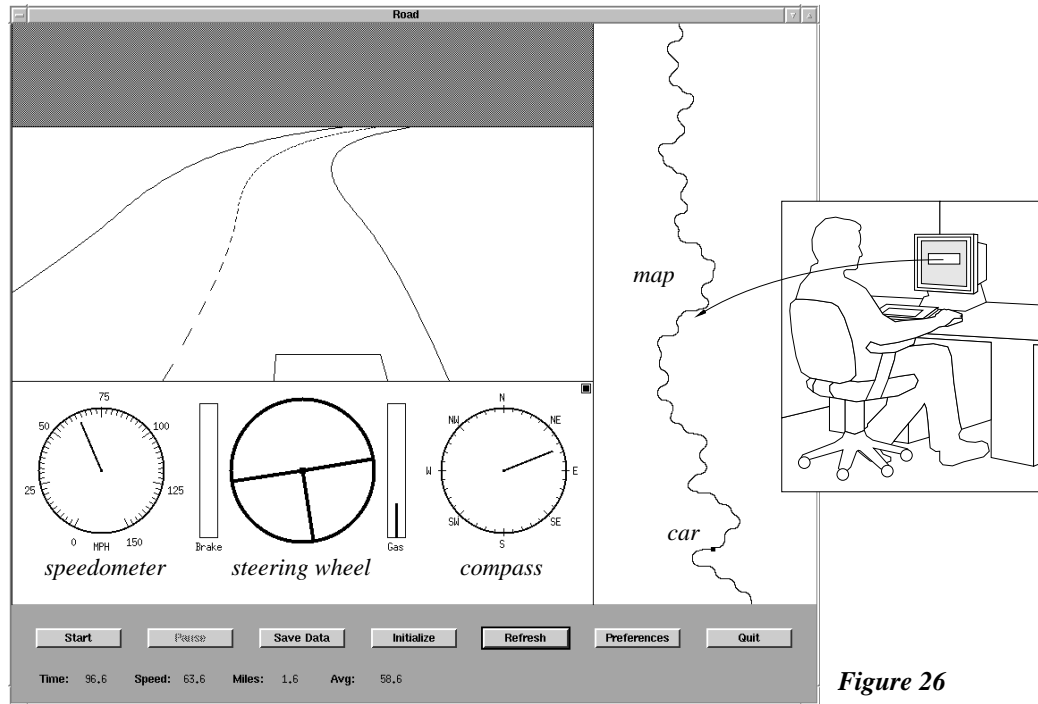


Figure 26

B. Real-world example: modeling human control trajectories

In previous work [5], hidden Markov models have been applied towards modeling and analyzing human control strategies in a dynamic graphic driving simulator (shown in Figure 26). For our example here, we preprocess and vector-quantize (to $L = 64$ levels) a sample human control trajectory in order to generate a discrete observation sequence O of length $T = 3875$. We then train an eight-state ($N = 8$) HMM λ_8 on this observation sequence using the Baum-Welch reestimation formulas.

It is instructive to look at the parameter values $\{A, B\}$ to which the final model converges. We can do this graphically by plotting the A and B matrices in Figure 27, where darker shades represent smaller values and lighter shades represent larger relative values. Although the model is parameterized by a total of,

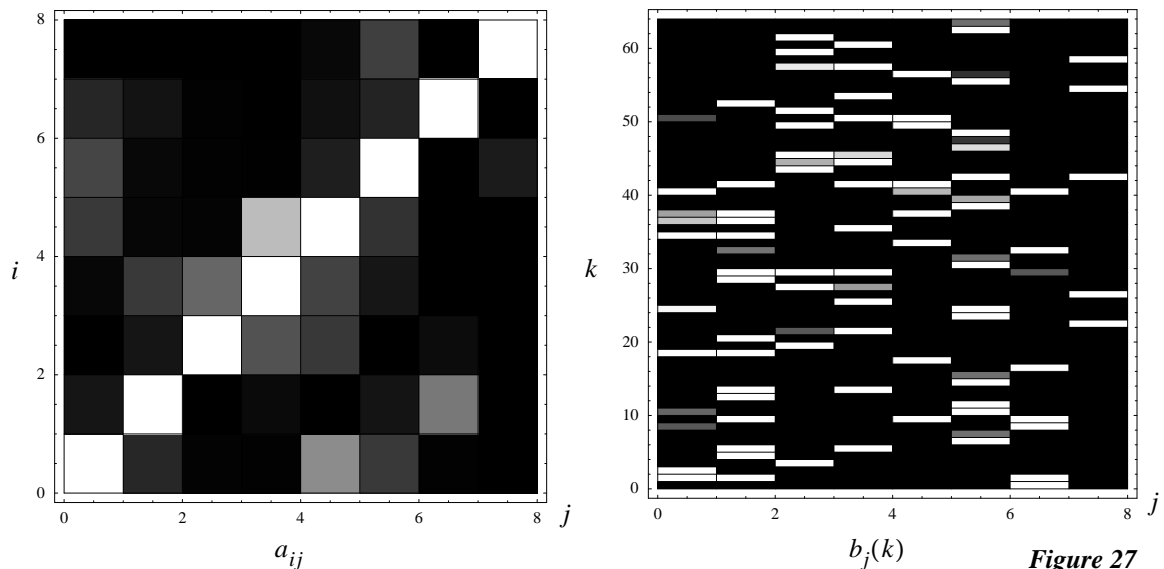


Figure 27

$$N \times N + N \times L = 576 \text{ free parameters,} \quad (261)$$

(ignoring the π vector), the final model retains only 142 nonzero parameters. In effect, the model was automatically reduced in terms of complexity during training. This type of model reduction is very typical of the Baum-Welch training algorithm.

Now, let us conduct the same experiment as in part A. We first use the final HMM λ_8 to generate a test observation sequence O' of length $T = 10000$, and then train models λ'_i , $i \in \{1, \dots, 10\}$, with $N = i$ states, from random initial parameter settings. Figure 28 below plots the normalized probabilities $P(O'|\lambda'_i)$ as a function of the number of states i . The maximum is given by,

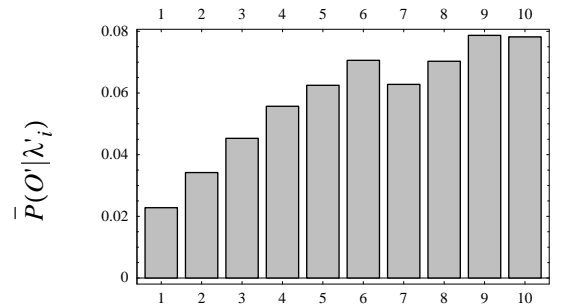


Figure 28

$$\max_i \bar{P}(O'|\lambda'_i) \approx 0.080 \text{ for } i = 9 \quad (262)$$

If we evaluate the test sequence O' on the generating HMM λ_8 , we get that,

$$\bar{P}(O'|\lambda_8) \approx 0.085 \quad (263)$$

These results indicate that when trained and evaluated on real-data, the *precise* number of states in the hidden Markov model is not so important as the *approximate* number of states required to sufficiently encode the sequential properties of the underlying data. In the above example, one or two states clearly are insufficient to capture the sequential properties of the generating HMM λ_8 . From equations (262) and (263) it is also evident that the Baum-Welch algorithm (a special case of the general EM algorithm) only converges to local maximum. Nevertheless, the local maximum to which the training algorithm converges is typically a relatively good local maximum, as is the case here.

Before we leave this topic, a quick word about the rate of convergence of the Baum-Welch algorithm. In a typical training scenario, we execute the Baum-Welch reestimation formulas until the change in the model falls below some threshold δ , such that,

$$\frac{\bar{P}(O|\lambda^{(t)}) - \bar{P}(O|\lambda^{(t-1)})}{\bar{P}(O|\lambda^{(t)})} < \delta \quad (264)$$

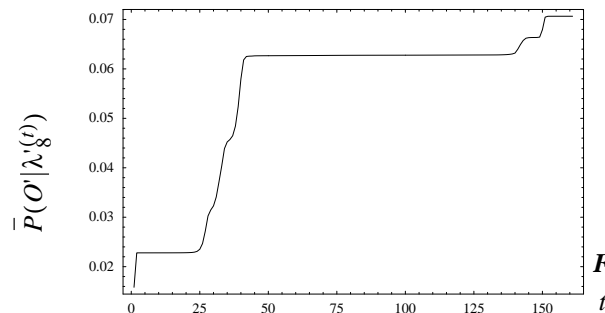


Figure 29

where $\lambda^{(t)}$ denotes the model λ after the t th step of the algorithm. From experience, that threshold should be set to some very small value (e.g. $\delta = 0.000001$) in order to ensure that the model has reached a final local maximum. Consider, for example, the convergence of the Baum-Welch algorithm for λ'_8 trained on O' as plotted in Figure 29. Note that $P(O'|\lambda'_8^{(t)})$ levels out several times before the final local maximum is reached.

- [1] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. of the IEEE*, vol. 77, no. 2, pp. 257-86, 1989.
- [2] A.P. Dempster, N.M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, no. 1, pp. 1-38, 1977.
- [3] L. R. Rabiner, B. H. Juang, S. E. Levinson and M. M. Sondhi, "Some properties of continuous hidden Markov model representations," *AT&T Technical Journal*, vol. 64, no. 6, pp. 1211-1222, 1986.
- [4] X. D. Huang, Y. Ariki and M. A. Jack, *Hidden Markov models for speech recognition*, Edinburgh University Press, Edinburgh, 1990.
- [5] M. C. Nechyba, *Learning and validation of human control strategies*, Ph.D. Thesis, Carnegie Mellon University, 1998.