Also appears in the *Online Symposium for Electronics Engineer 2000*
http://www.techonline.com/osee/

Hidden Markov Models: Fundamentals and Applications

Part 2: Discrete and Continuous Hidden Markov Models

Valery A. Petrushin
petr@cstar.ac.com
Center for Strategic Technology Research
Accenture
3773 Willow Rd.
Northbrook, Illinois 60062, USA.

Abstract

The objective of this tutorial is to introduce basic concepts of a Hidden Markov Model (HMM). The tutorial is intended for the practicing engineer, biologist, linguist or programmer who would like to learn more about the above mentioned fascinating mathematical models and include them into one's repertoire. This part of the tutorial is devoted to the basic concepts of a Hidden Markov Model. You will see how a Markov chain and Gaussian mixture models fuse together to form an HMM.

3 Introduction into Hidden Markov Models

3.1 Matrimonial contest problem

Welcome to the Emperor's palace! The elder daughter of Probabil the Great, beautiful Princess Variance, reached the 2-pi-square age that is considered as a mean of a normal distribution for the age when maidens of the Empire get married. Today, the traditional matrimonial contest will be held in the Palace. The winner will marry the Princess.

The contest has ancient roots. When a princess is born, the Emperor assigns her a special servant. Every day from the day of the princess' birth to the day of her 2-pi-square age her servant must visit four ponds in the Emperor's Garden, in accordance to a Markov process, and catch one hundred fishes. The servant must record fish colors (red, blue, or green) and then return the fish back to the ponds. Each pond is strictly maintained and has its own proportion of fish of different colors. Every day, the Emperor's mass media announced the results of the fishing.

Ten days before the princess comes of age, her servant will put every caught fish into a transparent numbered jar and send it to the Palace. (It is assumed that taking out 1,000 fishes does not change the statistical properties of the ponds.) In order to win the Princess' hand in marriage, a contestant has to guess as accurately as possible from which pond each fish came.

As time went by, however, the contest procedure changed due to the protests of the Wild Animal Protection Society and increasing pond maintenance expenses. Four temples replaced the four ponds. A big golden vat filled with perfume was placed in the middle of each temple. Artificial fishes of different colors were put in each vat. Each fish was made of precious stones of different colors as a mixture of three Gaussian components. Each temple had its own mixture and Gaussian distribution parameters that were kept secret. The results of every day's "fishing" of 100 fishes – the wavelength of reflected light for each fish – were available to the public in press, radio, TV, and the computer network, EmpirNet, at the site eww.emperor.gov.

Thus we have two problems:
(1) Decoding the sequence of ponds problem.
(2) Decoding the sequence of temples problem.

Let us consider the first problem. Our data is a sequence of *observations O* of length L=1000. Every data element is a color of a fish from a finite set of colors (or *a finite alphabet of symbols*). In our case the set contains three symbols: "red", "blue", and "green". Each fish was taken out of some pond, or we can say "a data point was emitted in some state q". A first-order Markov chain determines the sequence of states (see formulas (3.1) and (3.2)).

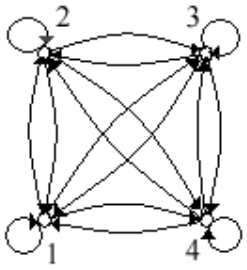
$$\pi = (\pi_1, \pi_2, ..., \pi_N) \tag{3.1}$$

$$A = \{a_{st}\}_{s,t=\overline{1,N}}, where \quad a_{st} = P(q_i = t \mid q_{i-1} = s) \tag{3.2}$$

$$E = \{e_j(k)\}_{j=\overline{1,N};\ k=\overline{1,M}}, where \quad e_j(k) = P(o_i = k \mid q_i = j) \tag{3.3}$$

Every state has its own discrete probability distribution for fish color. We shall call this distribution a *symbol emission vector in i-th state*. Collecting all vectors as columns of the matrix, we can get a *symbol emission matrix* (see formula (3.3)). A model of this sort is called a *discrete Hidden Markov Model* (HMM) because the sequence of state that produces the observable data is not available (hidden). HMM can also be considered as a double stochastic process or a partially observed stochastic process. Figure 3.1 shows an example of a discrete HMM.

O = GGBBRRRBBGBBRGRRRBGGRRBRG ...
Q = 22321113333444444444212211 ...



$$\lambda = \langle \pi, A, E \rangle$$

$$\pi = (0.1 \quad 0.4 \quad 0.4 \quad 0.1)$$

$$A = \begin{bmatrix} 0.8 & 0.05 & 0.1 & 0.25 \\ 0.05 & 0.7 & 0.15 & 0.1 \\ 0.05 & 0.15 & 0.7 & 0.1 \\ 0.05 & 0.05 & 0.1 & 0.8 \end{bmatrix}$$

$$E = \begin{bmatrix} 0.8 & 0.1 & 0.1 & 0.5 \\ 0.1 & 0.7 & 0.2 & 0.3 \\ 0.1 & 0.2 & 0.7 & 0.2 \end{bmatrix} \begin{matrix} R \\ G \\ B \end{matrix}$$

Figure 3.1 Four-pond HMM

Now, let us consider the decoding the sequence of temples problem. It only differs from the previous problem in that the emission probability distribution for color of artificial fishes is continuous in each state and can be represented by a Gaussian mixture model. In the case where every mixture has only one component, we get an emission probability density function (3.4). Returning to the general case of a Gaussian mixture probability density function we can transform a state with a mixture density into a net of multiple single-density states (see Figure 3.2). This model is called a *continuous HMM* or, speaking accurately, – a *continuous observation HMM*. Figure 3.3 shows an example of a continuous HMM.

$$E_j(o) = f(o; \mu_j, \sigma_j) \quad j = \overline{1,N} \tag{3.4}$$

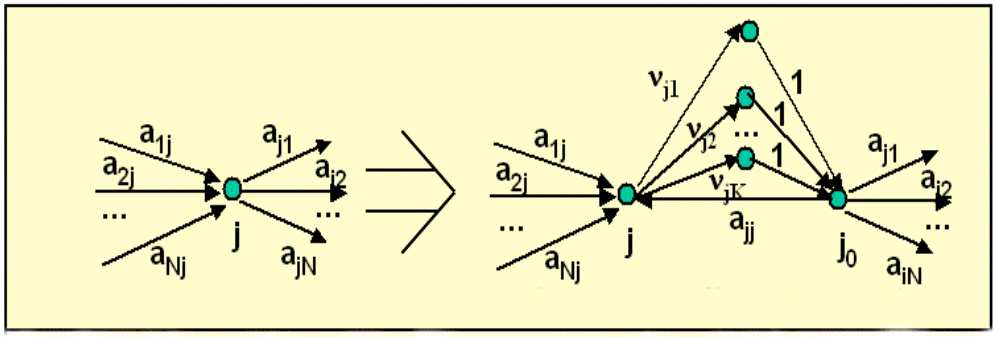
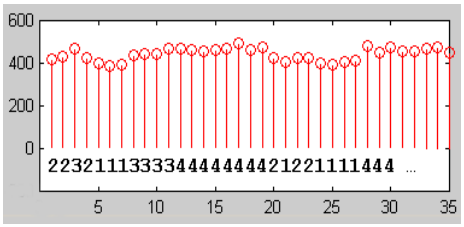
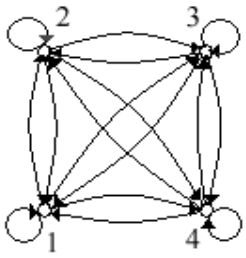


Figure 3.2. Mixture density state transformation



$$\pi = (0.1 \quad 0.4 \quad 0.4 \quad 0.1)$$

$$A = \begin{bmatrix} 0.8 & 0.05 & 0.1 & 0.05 \\ 0.05 & 0.7 & 0.15 & 0.1 \\ 0.05 & 0.15 & 0.7 & 0.1 \\ 0.05 & 0.05 & 0.1 & 0.8 \end{bmatrix}$$



$$E = \begin{bmatrix} 400 & 420 & 450 & 460 \\ 15 & 15 & 10 & 15 \end{bmatrix} \begin{matrix} \mu \\ \sigma \end{matrix}$$

Figure 3.3. Four-temple problem

3.2 Likelihood calculation

Let *O* be a sequence of observations and lambda be an HMM. How can you calculate the likelihood $P(O \mid \lambda)$ of *O* to be produced (or emitted) by λ?

First, assume we know the sequence of state *Q* that produced the sequence of observations *O*. Then the joint probability of *O* and *Q* can be calculated using formula (3.5). This formula is just a product of probabilities you meet by tracing the HMM diagram for the sequence *Q*. For example, formula (3.6) calculates the joint probability for *O* = "RGB", *Q* = "123" and the HMM depicted on Figure 3.1.

$$P(O,Q \mid \lambda) = \pi_{q_1} e_{q_1}(o_1) \cdot \prod_{i=2}^{L} a_{q_{i-1}q_i} e_{q_i}(o_i) \tag{3.5}$$

$$P(O,Q \mid \lambda) = \pi_1 e_1(R) a_{12} e_2(G) a_{23} e_3(B) = 0.3 \cdot 0.5 \cdot 0.25 \cdot 0.5 \cdot 0.4 \cdot 0.6 = 0.0045 \tag{3.6}$$

$$P(O \mid \lambda) = \sum_{all\ Q} P(O,Q \mid \lambda) \tag{3.7}$$

To calculate the likelihood, we have to sum probability over all possible state sequences (3.7). However, I do not recommend using the formula (3.7) for long observation sequences if you want to get results in your lifetime. For example, we have 4^{1000} state sequences for our

HMM of 4 states and an observation sequence of length 1000. We need approximately $2L*4^L$ operations to do the job. Let us assume we have a computer that can do 10^6 (one million) operations per second. Then it will take about 10^{200} seconds or 10^{192} years. The estimated age of the earth is less than this number. Thus, we need to use the more efficient procedure known as the *Forward-Backward Procedure*.

The Forward-Backward Procedure is based on the technique known as dynamic programming. Dynamic programming makes calculations for a small instance, stores the result, and then uses it later whenever it is needed, rather than recomputing it from scratch. To apply dynamic programming, we have to find a recursive property that allows us to do calculations for the next instance based on the current one.

Let us see how dynamic programming works for Forward-Backward Procedure.

Let $\alpha_k(i)$ be the probability of the partial observation sequence $O_{1\to k} = o_1, o_2, \ldots, o_k$ to be produced by all possible state sequences that end at i-th state (3.8). Then the probability of the partial observation sequence is the sum of $\alpha_k(i)$ over all N states (3.9).

$$\alpha_k(i) = P(o_1 o_2 \ldots o_k, q_k = i \mid \lambda) \tag{3.8}$$

$$P(o_1 o_2 \ldots o_k \mid \lambda) = \sum_{i=1}^{N} \alpha_k(i) \tag{3.9}$$

The Forward Procedure is a recursive algorithm for calculating $\alpha_k(i)$ for the observation sequence of increasing length k (see formulas from (3.10) to (3.12)). First, the probabilities for the single-symbol sequence are calculated as a product of initial i-th state probability and emission probability of the given symbol o_1 in i-th state (see formula (3.10)). Then the recursive formula (3.11) is applied. Assume we have calculated $\alpha_k(i)$ for some k. To calculate say $\alpha_{k+1}(2)$ (see Figure 3.4), we multiply every $\alpha_k(i)$ by corresponding transition probability from i-th state to the second state, sum the products over all states, and then multiply the result by the emission probability of the symbol o_{k+1}. Iterating the process, we can eventually calculate $\alpha_k(L)$, and then summing them over all states, we can obtain the required probability (see formula (3.12)).

Forward Algorithm:

Initialization:

$$\alpha_1(j) = \pi_j e_j(o_1) \quad j = \overline{1, N} \tag{3.10}$$

Recursion:

$$\alpha_{k+1}(j) = \left[\sum_{i=1}^{N} \alpha_k(i) a_{ij} \right] \cdot e_j(o_{k+1}) \quad j = \overline{1, N}; k = \overline{1, L-1} \tag{3.11}$$

Termination:

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_L(i) \tag{3.12}$$

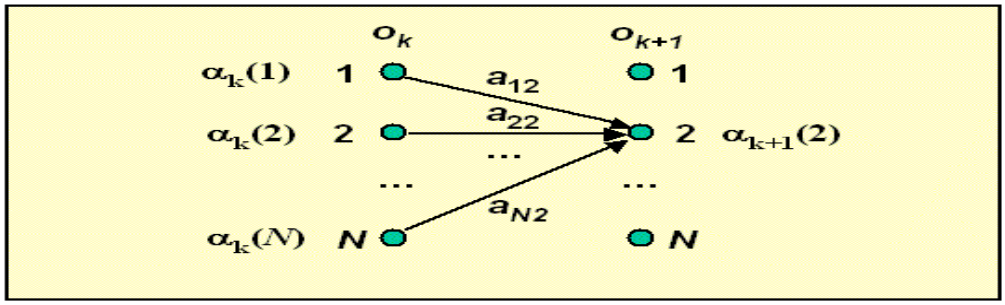


Figure 3.4. Forward variable computation

In a similar manner, we can introduce a symmetrical backward variable $\beta_k(i)$ as the conditional probability of the partial observation sequence from o_{k+1} to the end to be produced by all state sequences that start at i-th state (3.13). The Backward Procedure calculates recursively backward variables going backward along the observation sequence (see formulas from (3.15) to (3.17) and Figure 3.5).

$$\beta_k(i) = P(o_{k+1}o_{k+2}...o_L \mid q_k = i, \lambda) \tag{3.13}$$

$$P(o_{k+1}o_{k+2}...o_L \mid \lambda) = \sum_{i=1}^{N} \pi_i e_i(o_{k+1})\beta_k(i) \tag{3.14}$$

The Forward Procedure is typically used for calculating the probability of an observation sequence to be emitted by a HMM, but, as we shall see later, both procedures are heavily used for finding the optimal state sequence and estimating the HMM parameters.

Backward Procedure:

Initialization:

$$\beta_1(i) = 1 \quad i = \overline{1,N} \tag{3.15}$$

Recursion:

$$\beta_k(i) = \sum_{j=1}^{N} a_{ij} e_j(o_{k+1})\beta_{k+1}(j) \quad i = \overline{1,N};\ k = \overline{L-1,1} \tag{3.16}$$

Termination:

$$P(O \mid \lambda) = \sum_{i=1}^{N} \pi_i e_i(o_1)\beta_1(i) \tag{3.17}$$

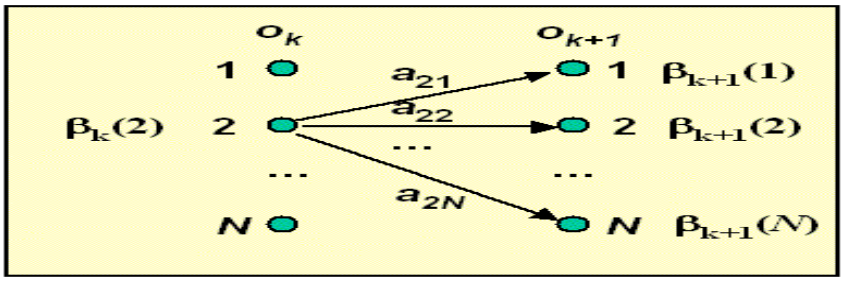


Figure 3.5. Backward variable calculation

Table 3.1. Forward and backward variable calculation

	o_1 = 'G'		o_2 = 'G'		o_3 = 'B'		o_4 = 'B'		o_5 = 'R'	
	α_1	β_1	α_2	β_2	α_3	β_3	α_4	β_4	α_5	β_5
1	0.0100	0.0033	0.0028	0.0145	0.0011	0.0730	0.0004	0.6800	0.0011	1.0
2	0.2800	0.0126	0.1470	0.0208	0.0214	0.0554	0.0039	0.1750	0.0005	1.0
3	0.0800	0.0106	0.0204	0.0545	0.0269	0.1035	0.0159	0.1750	0.0012	1.0
4	0.0300	0.0073	0.0181	0.0226	0.0063	0.0902	0.0020	0.4550	0.0018	1.0

Table 3.1 shows the results of calculation of the forward and backward variables for the HMM depicted in Figure 3.1 and observation sequence of length 5.

3.3 Posterior Decoding

All right! Now you can compute the probability of an observation sequence to be produced by an HMM. But to win the contest, you must find the sequence of hidden states that best explains the observations. But what does it mean "that best explains" or what is the criterion of optimality? There are several possible criteria. One is to choose states that are individually most likely at the time when a symbol is emitted. This approach is called *posterior decoding*.

Let $\lambda_k(i)$ be the probability of the model to emit k-th symbol being in the i-th state for the given observation sequence (see formula (3.18)). It is easy to derive the formula (3.19) that is used for calculating lambda variables. Then at each time we can select the state q_k that maximizes $\lambda_k(i)$ (see formula (3.20)). Table 3.2 presents the results of lambda variable

$$\lambda_k(i) = P(q_k = i \mid O, \lambda) \tag{3.18}$$

$$\lambda_k(i) = \frac{\alpha_k(i)\beta_k(i)}{P(O \mid \lambda)} \quad k = \overline{1, L}; i = \overline{1, N} \tag{3.19}$$

$$q_k = \arg\max_{1 \le i \le N} \{\lambda_k(i)\} \quad k = \overline{1, L} \tag{3.20}$$

calculations for the 5-symbol observation sequence and the model shown in Figure 3.1.The real sequence of states is 2-2-3-2-1 but the decoded sequence is 2-2-3-3-4.

Table 3.2. Posterior decoding results for 5-symbol sequence

	o_1 = 'G	o_2 = 'G'	o_3 = 'B'	o_4 = 'B'	o_5 = 'R'
	γ_1	γ_2	γ_3	γ_4	γ_5
1	0.0072	0.0087	0.0181	0.0537	0.2394
2	**0.7622**	**0.6616**	0.2566	0.1469	0.1131
3	0.1831	0.2409	**0.6027**	**0.6041**	0.2593
4	0.0475	0.0888	0.1226	0.1953	**0.3882**

Figure 3.6 shows the results for the same model and the observation sequence of length 300. We use the following color codes for states: 1 – blue, 2 – green, 3 – red, 4 – magenta. The accuracy is 61.33 %.

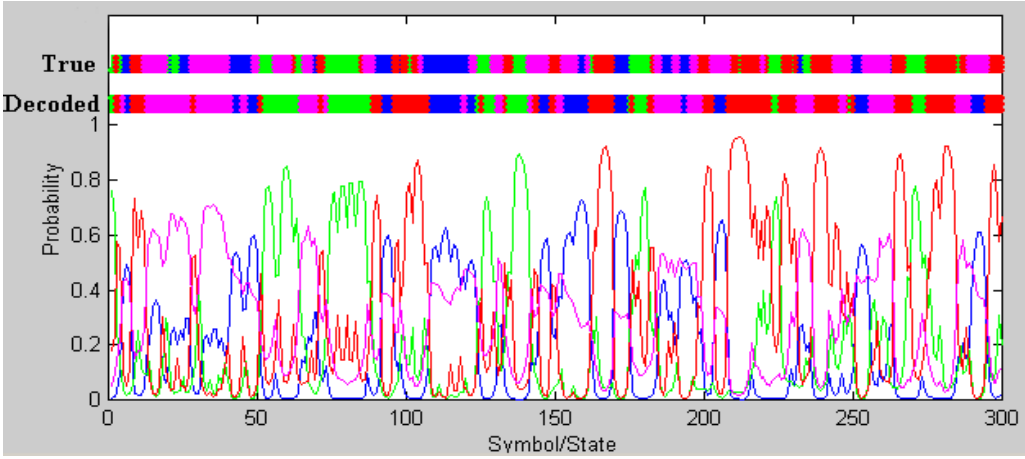


Figure 3.6. Posterior decoding results.

3.4 Viterbi decoding

Posterior decoding works fine in our case because our HMM is *ergodic*, i.e. there is transition from any state to any other state. If applied to an HMM of another architecture, this approach could give a sequence that may not be a legitimate path because some transitions are not permitted. In that case, we can use another approach known as either the *Viterbi decoding* or *Viterbi algorithm*. The Viterbi algorithm chooses one best state sequence that maximizes the likelihood of the state sequence for the given observation sequence (see formula (3.21)).

$$Q^* = \arg\max_Q \{P(Q \mid O, \lambda)\} = \arg\max_Q \{P(Q, O \mid \lambda)\} \qquad (3.21)$$

$$\delta_k(i) = \max_{q_1 q_2 \ldots q_{k-1}} P(q_1 q_2 \ldots q_k = i, o_1 o_2 \ldots o_k \mid \lambda) \qquad (3.22)$$

Let $\delta_k(i)$ be the maximal probability of state sequences of the length k that end in state i and produce the k first observations for the given model (see (3.22)). The Viterbi algorithm is a dynamic programming algorithm that uses the same schema as the Forward procedure except for two differences:

(1) It uses maximization in place of summing at the recursion and termination steps (see (3.25) and (3.27), and compare to (3.11) and (3.12)).

(2) It keeps track of the arguments that maximize $\delta_k(i)$ for each k and i storing them in the N by L matrix. This matrix is used to retrieve the optimal state sequence at the backtracking step (see formulas (3.24), (3.26), and (3.29)).

Viterbi Decoding:

Initialization:

$$\delta_1(j) = \pi_j e_j(o_1) \quad j = \overline{1, N} \qquad (3.23)$$

$$\psi(j) = 0 \qquad (3.24)$$

Recursion:

$$\delta_{k+1}(j) = \max_{1 \le i \le N} \left[\delta_k(i) a_{ij}\right] \cdot e_j(o_k) \quad j = \overline{1, N}; k = \overline{1, L-1} \qquad (3.25)$$

$$\psi_{k+1}(j) = \arg\max_{1 \le i \le N} \left[\delta_k(i) a_{ij}\right] \qquad (3.26)$$

Termination:

$$p^* = \max_{1 \le i \le N} [\delta_L(i)] \tag{3.27}$$

$$q_L^* = \arg\max_{1 \le i \le N} [\delta_L(i)] \tag{3.28}$$

Backtracking:

$$q_k^* = \psi_{k+1}(q_{k+1}^*) \quad k = \overline{L-1,1} \tag{3.29}$$

Table 3.3 shows the results of the Viterbi decoding for the 5-symbol observation sequence and the model shown in Figure 3.1.The real sequence of states is 2-2-3-2-1 but the decoded sequence is 2-2-2-3-1.

Table 3.3. Viterbi decoding for 5-symbol sequence

	o_1 = 'G'	o_2 = 'G'	o_3 = 'B'	o_4 = 'B'	o_5 = 'R'
	δ_1	δ_2	δ_3	δ_4	δ_5
1	0.0100	0.0014	0.0007	0.0001	**0.0005647**
2	**0.2800**	**0.1372**	**0.0192**	0.0027	0.0001882
3	0.0800	0.0112	0.0144	**0.0071**	0.0004941
4	0.0300	0.0072	0.0014	0.0003	0.0003529

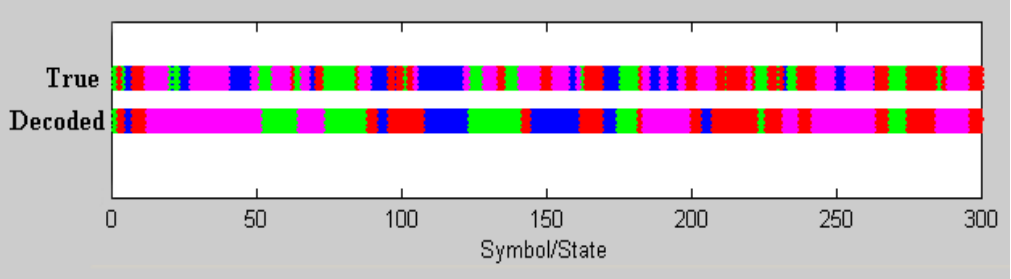


Figure 3.7. Viterbi decoding

Figure 3.7 shows the results of the Viterbi decoding for the same model and the observation sequence of length 300. . We use the same color coding for states: 1 – blue, 2 – green, 3 – red, 4 – magenta. The accuracy is 62.33 %. You can see that, in our case, the accuracy for both approaches (posterior and Viterbi decoding) is practically the same.

3.5 Training algorithm (Baum-Welch)

Great! Now you can decode the sequence of temples, marry the Princess, and live happily ever after. But wait a minute! One little thing is missing – the model. You need to build a model and estimate its parameters. Fortunately, you have a lot of historical data – the sequence of 719,900 artificial fishes. How can you build and train the model? You know the structure of the model. It is a 4-state ergodic model shown in Figure 3.1. You simply need to estimate the parameters of the model, i.e. transition probabilities and emission function.

Suppose we have an observation sequence O of length L. If we knew the corresponding sequence of states Q we could count the number of times each transition or emission occurred in the training sequence O. Then we could estimate the parameters using formulas (3.33), where A_{ij} is the counter for transitions from i-th state to j-th state, and $E_i(k)$ is the number of emissions of the k-th symbol in the state i. Note, we use the same formula for transition probabilities as in the case of Markov chain parameter estimation (see (1.4)). For estimating parameters of a continuous observation HMM, we could collect data emitted in each state and apply the EM algorithm to estimate the mixture parameters in each state. But we do not know the state sequence; it is hidden. Most likely, a variant of the EM algorithm exists that can solve the problem. Indeed, such an algorithm -- the *Baum-Welch algorithm* -- was proposed in the early 1970s.

The key idea of the algorithm is to estimate the expected number of transitions from the state i to the state j and emissions of the symbol k in the state i based on the current parameter values and the training observation sequence. These estimates are then used to recalculate the parameters of the model. The process continues until the stopping criterion is reached.

The algorithm uses the probabilities of transitions and emissions to approximate the corresponding counters. The transition probability from i-th state to j-th state at time k can be calculated using formula (3.30). Here, the numerator is the joint probability of being in the state i at time k, and in the state j at time k+1 and emitting the observations O that is calculated as the product of the following factors: the forward probability of i-th state at time k, the transition probability from the state i to the state j – a_{ij}, the emission probability of the

$$\xi_k(i,j) = P(q_k = i, q_{k+1} = j \mid O, \lambda) = \frac{\alpha_k(i) a_{ij} e_j(o_{k+1}) \beta_{k+1}(j)}{P(O \mid \lambda)} \qquad (3.30)$$

symbol o_{k+1} in the state j and the backward probability of the state j at time k+1 (see Figure 3.9). Dividing the product by the probability of the observation sequence O, we obtain the conditional probability of the transition from the state i to the state j at time k (3.30). Summing the estimates for transition counters over time, we obtain the expected number of transitions (see formula (3.31). To estimate the number of emissions, we use the sum of the posterior probabilities at time when the symbol was emitted (see (3.32) and compare to (3.19)). These estimates are used to recalculate the model parameters using the formula (3.33). The algorithm stops when the difference between two consecutive values of likelihood function is less than a threshold, or the maximal number of iterations is exceeded.

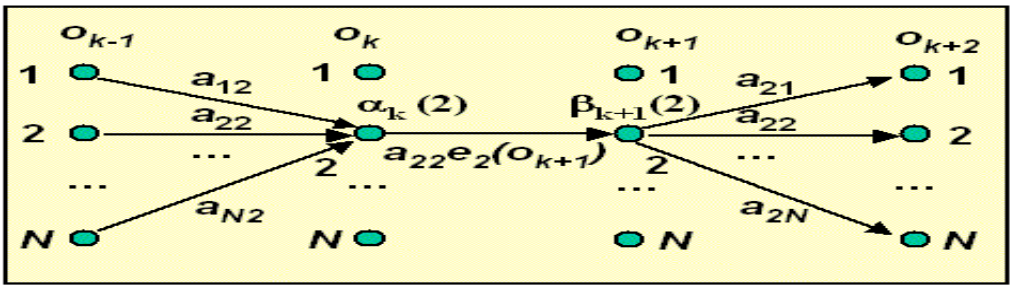


Figure 3.8. Transition counter estimation

In the case of continuous observations, we calculate the weight $W_k(j,l)$ that is the probability of the observation to belong to the l-th mixture component in the state j at time k as the product of two factors: the posterior probability of being in the state j at time k, and the posterior probability of belonging the observation o_k to the l-th mixture component (see (3.34), and compare to (2.4) and (3.32)). Then we calculate the mixture parameters using formulas (3.35), (3.36) and (3.37), which generalize the formulas (2.5), (2.6) and (2.7).

Baum-Welch Algorithm.

Initialization: Randomly choose model parameters. Set A_{ij} and $E_i(k)$ to pseudocount values.

Recursion:

- Calculate $\alpha_i(l)$ and $\beta_i(l)$ for i=1,N and l=1,L.
- Calculate

$$A_{ij} = \frac{1}{P(O \mid \lambda)} \sum_{l=1}^{L-1} \alpha_i(l) a_{ij} e_j(o_{l+1}) \beta_j(l+1) \tag{3.31}$$

$$E_i(k) = \frac{1}{P(O \mid \lambda)} \sum_{\{l \mid o_l = k\}} \alpha_i(l) \beta_i(l) \tag{3.32}$$

- Estimate new model parameters:

$$a_{ij} = \frac{A_{ij}}{\sum_{l=1}^{N} A_{il}}, \; e_i(k) = \frac{E_i(k)}{\sum_{l=1}^{M} E_i(l)} \quad k = \overline{1,M}; \; i,j = \overline{1,N} \tag{3.33}$$

- Calculate log-likelihood.

Termination: Stop, when the difference between two consecutive values of likelihood function is less than a threshold, or the maximal number of iterations is exceeded.

$$W_k(j,l) = \left[\frac{\alpha_k(j) \beta_k(j)}{P(O \mid \lambda)} \right] \cdot \left[\frac{\nu_{jl} f(o_k; \mu_{jl}, \Sigma_{jl})}{\sum_{m=1}^{K} \nu_{jm} f(o_k; \mu_{jm}, \Sigma_{jm})} \right] \tag{3.34}$$

$$\nu_{jl} = \frac{\sum_{k=1}^{L} W_k(j,l)}{\sum_{k=1}^{L} \sum_{m=1}^{K} W_k(j,m)} \quad j = \overline{1,N}; \; l = \overline{1,K} \tag{3.35}$$

$$\mu_{jl} = \frac{\sum_{k=1}^{L} W_k(j,l) \cdot o_k}{\sum_{k=1}^{L} W_k(j,l)} \tag{3.36}$$

$$\Sigma_{jl} = \frac{\sum_{k=1}^{L} W_k(j,l) \cdot (o_k - \mu_{jl}) \cdot (o_k - \mu_{jl})^T}{\sum_{k=1}^{L} W_k(j,l)} \tag{3.37}$$

It is easy to generalize the above algorithm when several observation sequences are available. In this case, we estimate counters for each sequence using formulas (3.31) and (3.32), sum the results, and recalculate the model parameters using (3.33). To estimate the initial state probabilities, we count decoded initial states, and divide the counters by the total number of training sequences.

3.6 Viterbi training

An alternative approach to model parameters estimation is *Viterbi training*. In this approach, the most probable path for each training sequence is derived using Viterbi decoding. Then this

path is used for estimating counts for the number of transactions and symbol emissions that are used for recalculating the model parameters (see (3.33)).

Viterbi Training.

Initialization: Choose model parameters randomly.

Iteration:

- Derive the most probable state sequence Q using the Viterbi decoding algorithm.
- Calculate A_{ij} and $E_i(k)$ for the given Q.
- Estimate the new model parameters using (3.33).

Termination: Stop, if the model parameters do not change for adjacent iterations.

Figure 3.9 shows the results of model parameters estimation for a sequence of the length 5000 symbols using both the Baum-Welch algorithm and Viterbi training.

True model:	**Estimated model (Baum-Welch):**	**Estimated model (Viterbi):**
$A=\begin{bmatrix} 0.8 & 0.05 & 0.1 & 0.25 \\ 0.05 & 0.7 & 0.15 & 0.1 \\ 0.05 & 0.15 & 0.7 & 0.1 \\ 0.05 & 0.05 & 0.1 & 0.8 \end{bmatrix}$	$A=\begin{bmatrix} 0.65 & 0.1 & 0.19 & 0.13 \\ 0.04 & 0.72 & 0.11 & 0.13 \\ 0.09 & 0.16 & 0.67 & 0.08 \\ 0.08 & 0.07 & 0.04 & 0.81 \end{bmatrix}$	$A=\begin{bmatrix} 0.6 & 0.12 & 0.1 & 0.18 \\ 0.04 & 0.78 & 0.1 & 0.08 \\ 0.08 & 0.01 & 0.89 & 0.02 \\ 0.01 & 0.04 & 0.04 & 0.91 \end{bmatrix}$
$E=\begin{bmatrix} 0.8 & 0.1 & 0.1 & 0.5 \\ 0.1 & 0.7 & 0.2 & 0.3 \\ 0.1 & 0.2 & 0.7 & 0.2 \end{bmatrix}\begin{matrix} R \\ G \\ B \end{matrix}$	$E=\begin{bmatrix} 0.89 & 0.04 & 0.08 & 0.53 \\ 0.1 & 0.71 & 0.11 & 0.25 \\ 0.01 & 0.25 & 0.81 & 0.22 \end{bmatrix}\begin{matrix} R \\ G \\ B \end{matrix}$	$E=\begin{bmatrix} 0.8 & 0.05 & 0.08 & 0.6 \\ 0.05 & 0.72 & 0.1 & 0.26 \\ 0.15 & 0.23 & 0.82 & 0.14 \end{bmatrix}\begin{matrix} R \\ G \\ B \end{matrix}$

Figure 3.9. Baum-Welch and Viterbi training

So, now you have a decent weapon to compete with the other rivals for the Princess' hand. Good luck!

3.7 References and Applications

Unfortunately, a comprehensive book devoted to Hidden Markov Models does not yet exist. There are, however, several books intended for a reader with a specific background. The most famous areas of HMM application are speech recognition and bioinformatics, and books devoted to these research areas often have chapters covering HMM. It is interesting to note that speech recognition uses continuous HMMs, but bioinformatics uses discrete HMMs for gene recognition and representation of protein families.

I refer to four books. Rabiner and Juang's book [1] is indispensable. It has a chapter that covers both the discrete and continuous HMMs. Durbin's book [2] provides a very gentle introduction into the Markov chain and discrete hidden Markov models. MacDonald and Zucchini's book [3] offers a statisticians' viewpoint. Elliott's book [4] deals with the theory of HMM and requires a strong mathematical background.

Hidden Markov Models are used for a wide spectrum of applications. As I mentioned before the most famous areas are bioinformatics and speech technology. In bioinformatics HMMs are used for gene finding, modeling protein families, protein structure prediction, and multiple sequence alignment. There are many companies competing in this area, I mentioned only one of them Net-ID, Inc. (http://www.netid.com/), which produces a neat commercial tool HMMpro. The major HMM application in speech technology is speech recognition, but the models are also used for speaker recognition and language modeling (see the NSF-European

Commission survey on human language Technologies [5]). The other areas of application include image processing [6], communications, signal processing [7], finance [8-9], traffic modeling, learning behavior of live and artificial systems, etc. Recently we witness the rapidly growing wave of research and application of this technique. For example, an HMM bibliography of 1997, which is still a valuable source of information (http://tsi.enst.fr/~cappe/docs/hmmbib.html), covers about 200 papers. Now the number of publications is at least twenty times as large.

It is interesting to note that there are only a few commercial HMM software packages available. It can be explained by the fact that HMM modeling, especially for speech recognition, is going on a very low level. Many commercial tool kits and API for speech recognition are based on HMMs but allow users only to train or adapt them. I refer to two HMM tools. First is HMMpro from Net-ID, Inc. (http://www.netid.com/html/hmmpro.html) which is a tool for creating discrete HMMs for bioinformatics. The other one is HTK from Entropic, Inc. HTK was the best HMM toolkit for speech technology for several years. But the tool was discontinued when Entropic was acquired by Microsoft.

Fortunately, there are many free software packages available on the Web:

Name/Language	URL	Used for
HMMER (C)	http://hmmer.wustl.edu/	bioinformatics
Myers' HMM software (C)	http://www.itl.atr.co.jp/comp.speech/Section6/Recognition/myers.hmm.html	speech
Kanungo's HMM software (C)	http://www.cfar.umd.edu/~kanungo/software/software.html	language modeling
Murphy's HMM software (MATLAB)	http://www.cs.berkeley.edu/~murphyk/Bayes/hmm.html	speech
Cappe's HMM software (MATLAB)	http://tsi.enst.fr/~cappe/node4.html	signal processing
HME software (MATLAB)	http://www.stern.nyu.edu/~aweigend/Research/Software	market analysis
hmmlib (JAVA)	http://www.vilab.com/hmmlib/home.html	???

References

.

[1] L. Rabiner and B.-H. Juang *Fundamentals of speech recognition*. Prentice-Hall, 1993.

[2] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison *Biological sequence analysis. Probabilistic models of proteins and nucleic acids.* Cambrige University Press, 1998

[3] I.L. MacDonald and W. Zucchini *Hidden Markov and Other Models for Discrete-Valued Time Series.* Chapman and Hall, 1997

[4] R.J. Elliott, L. Aggoun and J.B. Moore *Hidden Markov Models: Estimation and Control.* Springer Verlag, 1995.

[5] http://cslu.cse.ogi.edu/HLTsurvey/HLTsurvey.html

[6] http://www.dei.unipd.it/~cuzzolin/Review.html

[7] http:://www.cssip.edu.au/~iain/otherwww/hmm.html

[8] http://www.stern.nyu.edu/~aweigend/Research/Papers/HiddenMarkov/WeigendShi_Stern98.html

[9] http://www.cs.sun.ac.za/courses/hons/project_reports_1999/deon_van_biljon/