Three fundamental questions with Hidden Markov Models

Once defined the structural elements of HMM we may want to ask three different kinds of questions:

1. Given the model $\lambda = (A, B, \pi)$, what is the probability of observing $O = (o_1, o_2, .., o_T)$, that is $P(O \mid \lambda)$?
2. Given the sequence of observations $O = (o_1, o_2, .., o_T)$ and the HMM defined by $\lambda = (A, B, \pi)$, what is the most likely state sequence $q = (q_1, q_2, .., q_T)$ that best explains observations?
3. How can we learn *A*, *B* e *π* given a sequence of observations such that $P(O \mid \lambda)$ is maximum?

Question 1: pattern recognition
Question 2: find the best hidden sequence of staes (not directly observable); it does not exists a correct state sequence, however we can find the one that maximize some error measure
Question 3: model training, given some example sequences we can set up a hidden Markov model. This is the first step in many applications since often we do not have the model parameters to answer the previous questions.

Solution to problem 1, probability estimation

This problem aswer the question about the probability of observing a sequence of observation $O=(o_1,o_2,..,o_T)$ given a HMM λ, that is $P(O \mid \lambda)$. Since we assume the observation to be independent from each other (recall the conditional independence property of HMM Bayesian Network), the probability of $O=(o_1,o_2,..,o_T)$ being generated from a sequence of states *q* can be computed as:

$$P(O \mid q, B) = b_{q1}(o_1) \cdot b_{q2}(o_2) \cdot \ldots \cdot b_{qT}(o_T) = \prod_{i=1}^{T} b_{qi}(o_i)$$

and the probability of the state sequence *q* is:

$$P(q \mid A, \pi) = \pi_{q1} \cdot a_{q1q2} \cdot a_{q2q3} \cdot \ldots \cdot a_{qT-1qT} = \pi_{q1} \cdot \prod_{i=1}^{T-1} a_{qi,i+1}$$

The joint probability for *O* and *q*, that is the probability of *O* e *q* verified at the same time, is the product of the two previous terms:

$$\begin{aligned} P(O,q \mid \lambda) &= P(O \mid q, B) \cdot P(q \mid A, \pi) \\ &= \pi_{q1} b_{q1}(o_1) \cdot a_{q1q2} b_{q2}(o_2) \cdot \ldots \cdot a_{qT-1qT} b_{qT}(o_T) \\ &= \pi_{q1} b_{q1}(o_1) \cdot \prod_{t=2}^{T} \left(a_{qt-1qt} \cdot b_{qt}(o_t) \right) \end{aligned}$$

We aim at finding $P(O \mid \lambda)$ obtained summing out the joint probability over all possible sequences of states q:

$$
\begin{aligned}
P(O \mid \lambda) &= \sum_{q} P(O \mid q, B) \cdot P(q \mid A, \pi) \\
&= \sum_{q1..qT} \pi_{q1} b_{q1}(o_1) \cdot a_{q1q2} b_{q2}(o_2) \cdot \ldots \cdot a_{qT-1qT} b_{qT}(o_T) \\
&= \sum_{q1..qT} \pi_{q1} b_{q1}(o_1) \cdot \prod_{t=2}^{T} \left(a_{qt-1qt} \cdot b_{qt}(o_t) \right)
\end{aligned}
$$

The interpretation of this is the following. At time $t = 1$ the process starts in state q_1 with probability π_{q1} and it generates observation o_1 with probability $b_{q1}(o_1)$. Time goes on changing from t to $t+1$ and a transition from q_1 to q_2 happens with probability a_{q1q2} generating symbol o_2 with probability $b_{q2}(o_2)$. This iterates up to time T.

This computation is infeasible in practice due to the exponential complexity with respect to sequence length T. Precisely it requires $(2T-1)N^T$ multiplications and N^T-1 sums. Also for small N and T, such as $N=5$ (states) and $T=100$ (observations), we are requested $(2 \cdot 100-1)5^{100} \approx 1.6 \cdot 10^{72}$ multiplications and $5^{100}-1 \approx 8.0 \cdot 10^{69}$ sums. To get some practical use out of HMM we need a more efficient algorithm such as the *forward-backwardalgorithm*, that is linear in T.

Forward Procedure

Consider the forward variable:

$$\alpha_t(i) = P(o_1, o_2, .., o_t, q_t = S_i \mid \lambda)$$

being *t* the time and S_i the state. $\alpha_t(i)$ is the probability of observing the sequence $o_1, o_2, .., o_t$, given the current state S_i at time *t*.

The values of $\alpha_{t+1}(i)$ can be obtained summing the *forward variable* for all the *N* states at time *t*, multipling it for the corresponding transition probabilities between states a_{ij} and for the emission probabilities $b_j(o_{t+1})$. The algorithm is the following:

1. Initialization

   t = 1

   $\alpha_1(i) = \pi_i b_i(o_1)$, $1 \le i \le N$

2. Induction

   $\alpha_{t+1}(j) = b_j(o_{t+1}) \sum_{i=1}^{N} \alpha_t(i) a_{ij}$, $1 \le j \le N$

3. Time update

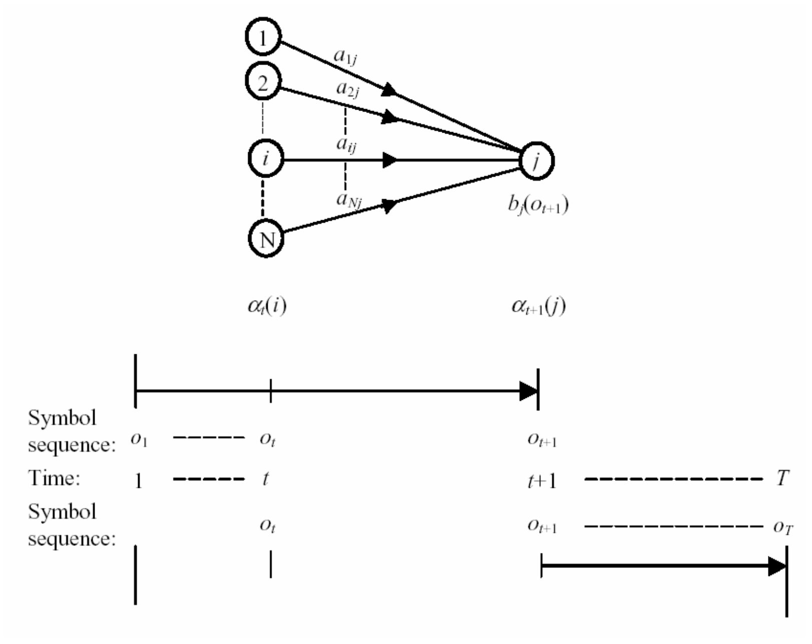
   t = t+1

   If *t<T* GOTO 2, otherwise GOTO 4

4. Final calculation

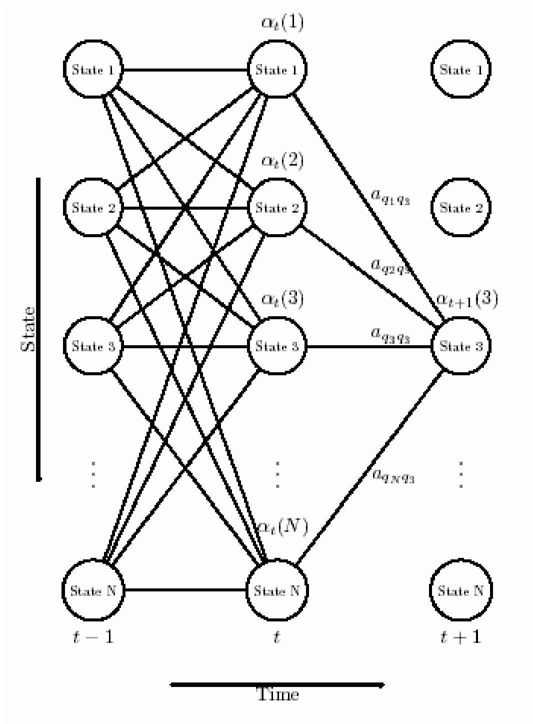
   $P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$

The single step is summarized by the following graph (Figure 3.6):


1
2
i
N
j
a_{1j}
a_{2j}
a_{ij}
a_{Nj}
$b_j(o_{t+1})$
$\alpha_t(i)$
$\alpha_{t+1}(j)$
Symbol sequence: o_1 ----- o_t o_{t+1}
Time: 1 ----- t $t+1$ ------------ T
Symbol sequence: o_t o_{t+1} ------------ o_T

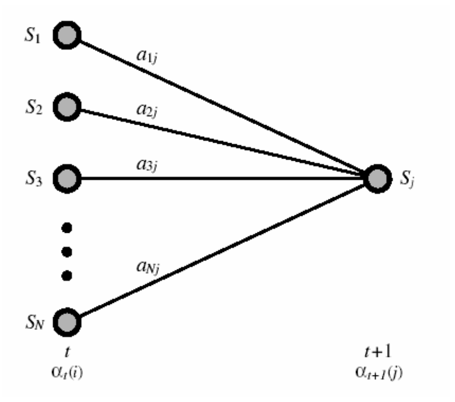
Summary of one step for *alfa* computation

The following graph shows the exponential number of paths in the space of observations.



Maximum number of transitions between states and selected paths

It is required to compute *2N* summs between *$\alpha_t(j)$* for each *$\alpha_{t+1}(i)$*, weighted by the corresponding transition probabilities between state *i* and the probability of observing *o_{t+1}*.



Number of operations required for the computation of variable *alfa*

At each time we have *N forward variables*, having each the same complexity. The induction is correct since it follow directly from:

$$P(o_1..o_t, q_{t+1} = i \mid \lambda) = \sum_{j=1}^{N} P(o_1..o_t, q_t = j \mid \lambda) a_{ij}$$

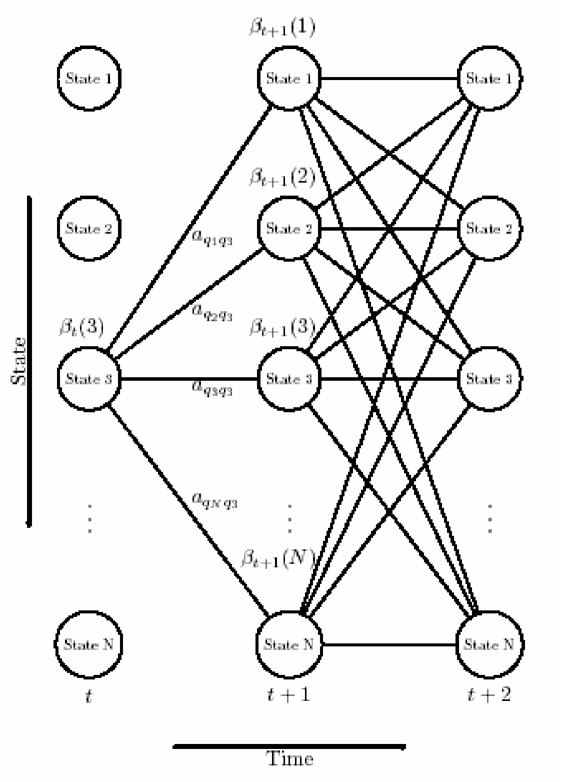
When using this algorithm, wee are required *N(N+1)(T-1)+N* multiplications and *N(N-1)(T-1)* sums. For *N = 5* state and *T = 100* observations it requires *5(5+1)(100-1)+5=2975* multiplications and *5(5-1)(100-1)=1980* sums, obtaining a significative gain with respect to the direct approach that requires 10^{72} multiplications and 10^{69} sums.

Backward Procedure

Previous recursion can be applied also backward defining the backward variable:

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, .., o_T \mid q_t = S_i, \lambda)$$

Sating the probability of observing the sequence from *t+1* to the end, given the state *i* at time *t* and the HMM *λ*. Note that e the forward vairable definition regards a joint probability while the backward variable is a conditional probability and ita can be computed similarly.



Maximum number of trasitions between states and selected paths

The *backward algorithm* is obtaind by the followinf steps:

1. Initialization

   t = T-1

   $\beta_T(i) = 1,\ \ 1 \le i \le N$

2. Induction

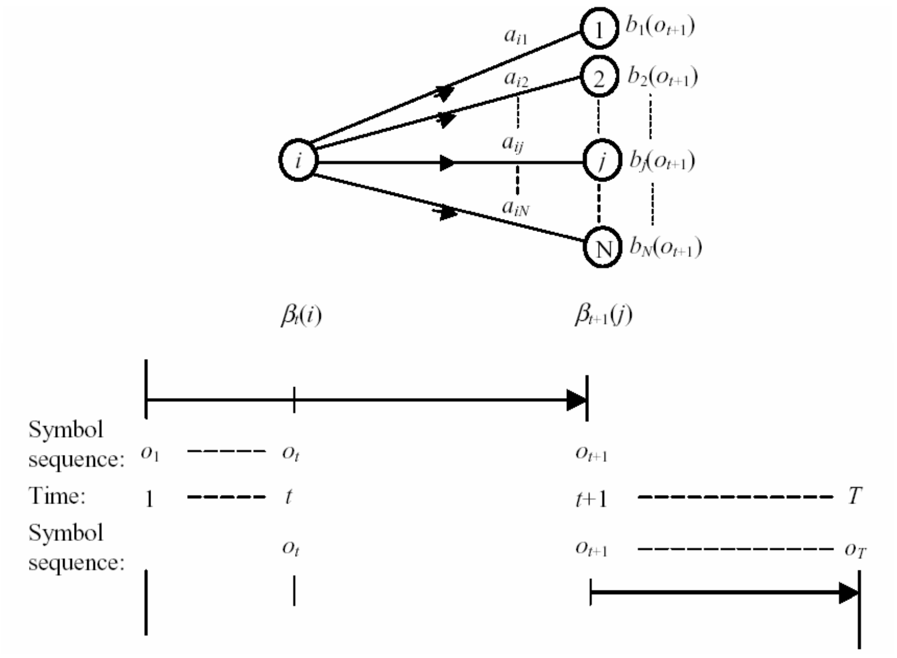
   $\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(i) a_{ij} b_j(o_{t+1}),\ \ 1 \le i \le N$

3. Time update

   t = t-1

   If *t>0* GOTO 2, otherwise end

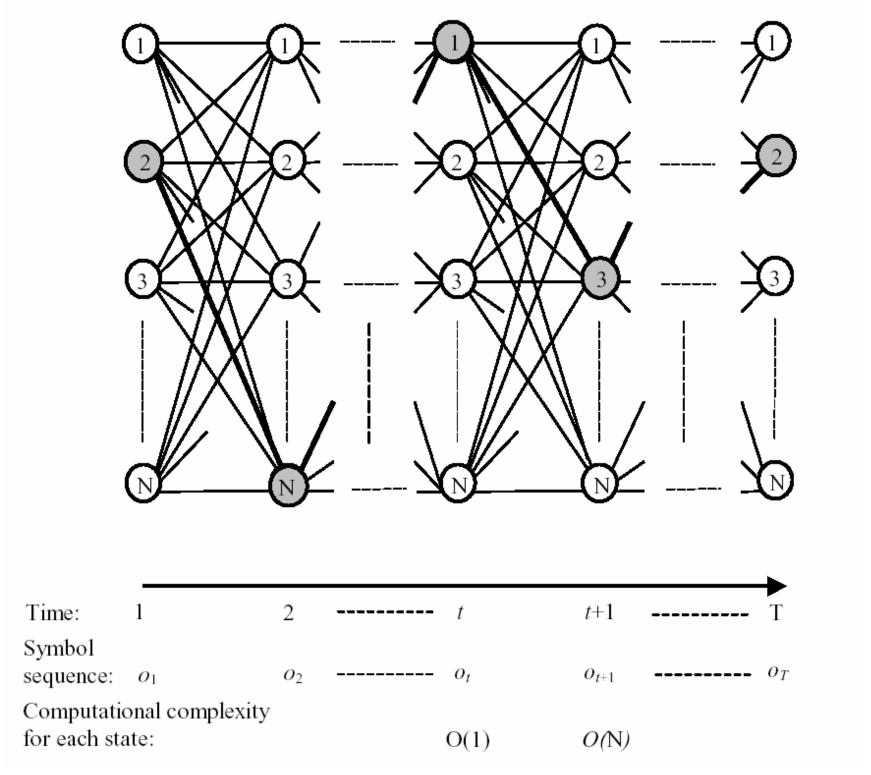
Note thate initialization sets $\beta_T(i) = 1$ for all *i*. Each step of the algorithm can be summarized as follows:



a_{i1}
a_{i2}
a_{ij}
a_{iN}
i
1
2
j
N
$b_1(o_{t+1})$
$b_2(o_{t+1})$
$b_j(o_{t+1})$
$b_N(o_{t+1})$
$\beta_t(i)$
$\beta_{t+1}(j)$
Symbol sequence: o_1 ----- o_t o_{t+1}
Time: 1 ----- t $t+1$ ------------ T
Symbol sequence: o_t o_{t+1} ------------ o_T

Summary of one step in *beta variable* computation

The whole computation of *forward* and *backward* variables for *N* state has complexity $O(N) \cdot O(N) = O(N^2)$.


1
2
3
N
Time: 1 2 t t+1 T
Symbol sequence: o_1 o_2 o_t o_{t+1} o_T
Computational complexity for each state: O(1) O(N)

***Forward* and *backward* computation has complexity proportional to the squared number of states**

Since $\alpha_t(i)\beta_t(\text{i})$ represents the probability of observing the sequence *O* and being in state *i* at time *t* given the HMM *λ*, we can write the sequence in terms of *forward* and *backward* variables, using the following:

$$P(O \mid \lambda) = \sum\nolimits_{i=1}^{N} P(q_t = i, O \mid \lambda) \text{ for all } t$$

$$= \sum\nolimits_{i=1}^{N} \alpha_t(i)\beta_t(\text{i}) \text{ for all } t$$

$$= \sum\nolimits_{i=1}^{N} \alpha_T(i)$$

Scaling of Forward and Backward variables

As usually happens in statitisc, we should deal with numerical issues. $\alpha_t(i)$ and $\beta_t(i)$ computation multiplies probabilities, having values lower than 1, thus they tend to zero exponentially with the number of symbols in the sequnces. Using common floating point for t high enough, i.e., $t>100$, we can easily underflow.

The basic procedure, multiplies $\alpha_t(i)$, at any t and in all states $i\text{-}1\leq i\leq N$, for a coefficient c_t that depends only on time and not on the state; this factor will be used also to scale $\beta_t(i)$ in the following.

Let be:

1. $\hat{\alpha}_t(i)$ the scaled variable during each iteration
2. $\hat{\hat{\alpha}}_t(i)$ the local version before scalingè
3. c_t the scaling coefficient

The algorithm becomes the following

1. Initialization

   $t = 2$

   $$\alpha_1(i) = \pi_i b_i(o_1)\,,\quad 1 \le i \le N$$

   $$\hat{\hat{\alpha}}_1(i) = \alpha_1(i)\,,\quad 1 \le i \le N$$

   $$c_1 = \frac{1}{\sum_{i=1}^{N} \alpha_1(i)}$$

   $$\hat{\alpha}_1(i) = c_1 \alpha_1(i)$$

2. Induction

   $$\hat{\hat{\alpha}}_t(i) = b_i(o_t) \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ji}\,,\quad 1 \le i \le N$$

   $$c_t = \frac{1}{\sum_{i=1}^{N} \hat{\hat{\alpha}}_t(i)}$$

   $$\hat{\alpha}_t(i) = c_t \hat{\hat{\alpha}}_t(i),\quad 1 \le i \le N$$

3. Time update

   $t = t + 1$

   If $t \le T$ GOTO 2, otherwise GOTO 4

4. Final calculation

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

Step 2 can be rewritten as:

$$\hat{\alpha}_t(i) = c_t \hat{\hat{\alpha}}_t(i)$$

$$= \frac{1}{\sum_{k=1}^{N}\left(b_k(o_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{jk}\right)} \cdot \left(b_i(o_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{ji}\right)$$

$$= \frac{\left(b_i(o_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{ji}\right)}{\sum_{k=1}^{N}\left(b_k(o_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{jk}\right)}, \quad 1 \le i \le N$$

Using induction the scale variable can be obtained by the unscaled one as:

$$\hat{\alpha}_{t-1}(j) = \left(\prod_{\tau=1}^{t-1} c_\tau\right)\alpha_{t-1}(j), \quad 1 \le j \le N$$

The iduction step can be computed as before, but with one time unit shift

$$\alpha_t(i) = b_i(o_t)\sum_{j=1}^{N}\alpha_{t-1}(j)a_{ij}\ , \quad 1 \le i \le N$$

We can now write:

$$\hat{\alpha}_t(i) = \frac{\left(b_i(o_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{ji}\right)}{\sum_{k=1}^{N}\left(b_k(o_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{jk}\right)}$$

$$= \frac{b_i(o_t)\sum_{j=1}^{N}\left(\prod_{\tau=1}^{t-1} c_\tau\right)\alpha_{t-1}(j)a_{ji}}{\sum_{k=1}^{N}\left(b_k(o_t)\sum_{j=1}^{N}\left(\prod_{\tau=1}^{t-1} c_\tau\right)\alpha_{t-1}(j)a_{jk}\right)}$$

$$= \frac{\left(\prod_{\tau=1}^{t-1} c_\tau\right)\left(b_i(o_t)\sum_{j=1}^{N}\alpha_{t-1}(j)a_{ji}\right)}{\left(\prod_{\tau=1}^{t-1} c_\tau\right)\sum_{k=1}^{N}\left(b_k(o_t)\sum_{j=1}^{N}\alpha_{t-1}(j)a_{jk}\right)}$$

$$= \frac{\alpha_t(i)}{\sum_{k=1}^{N}\alpha_t(k)}, \quad 1 \le \mathrm{i} \le \mathrm{N}$$

It can be proven that any term *$\alpha_t(i)$* is scaled by the sum for all the states of *$\alpha_t(i)$*.

At the end the algorithm we cannot use $\hat{\alpha}_t(i)$ in $P(O \mid \lambda)$ computation because the variable is already scaled. However we can apply the following property:

$$\prod_{\tau=1}^{T} c_t \sum_{i=1}^{N} \alpha_T(i) = 1$$

$$\prod_{\tau=1}^{T} c_t \cdot P(O \mid \lambda) = 1$$

$$P(O \mid \lambda) = \frac{1}{\prod_{\tau=1}^{T} c_\tau}$$

Using this new estimate $P(O \mid \lambda)$, however, the result could be il so small to underflow computer precision; for this reason we suggest to use the following:

$$\log P(O \mid \lambda) = \frac{1}{\prod_{\tau=1}^{T} c_\tau} = -\sum_{t=1}^{T} \log c_t$$

The scaled *backward* algorithm can be defined more easily since uses the same factors of the *forward* one. Also the notation is similar:

1. $\hat{\beta}_t(i)$ the scaled variable during iteration
2. $\hat{\hat{\beta}}_t(i)$ the local version before scaling
3. c_t the scaling coefficient

The algorithm works as follows:

1. Initialization

$$t = T - 1$$
$$\beta_T(i) = 1, \quad 1 \le \mathrm{i} \le \mathrm{N}$$
$$\hat{\beta}_T(i) = c_T \beta_\mathrm{T}(\mathrm{i}), \quad 1 \le \mathrm{i} \le \mathrm{N}$$

2. Induction

$$\hat{\hat{\beta}}_t(i) = \sum_{j=1}^{N} \hat{\beta}_{\mathrm{t}+1}(j) a_{ij} b_j(o_{t+1}), \quad 1 \le \mathrm{i} \le \mathrm{N}$$
$$\hat{\beta}_t(i) = c_t \hat{\hat{\beta}}_\mathrm{t}(\mathrm{i}), \quad 1 \le \mathrm{i} \le \mathrm{N}$$

3. Time update

be $t = t - 1$

If *t>0* GOTO 2, otherwise end

Solution to *problem 2*, optimal state sequence

This time the problem is finding the most likely state sequence that generated a given sequence of observations. If with problem 1 we have an exact solution, in this case we can applya an approximate algorithm. This is due to the difficulty in the definition of an optimality criterion.

One common approach is to define the most likely state *q_t* at each time step *t*, introducing the variable:

$$\gamma_t(i) = P(q_t = S_i \mid O, \lambda)$$

That is the probability of being in state S_i at time *t* given the sequence of observations O and the HMM λ. An alternative formulation is:

$$\begin{aligned}\gamma_t(i) &= P(q_t = S_i \mid O, \lambda) \\ &= \frac{P(O, q_t = S_i \mid \lambda)}{P(O \mid \lambda)} \\ &= \frac{P(O, q_t = S_i \mid \lambda)}{\sum_{i=1}^{N} P(O, q_t = S_i \mid \lambda)}\end{aligned}$$

Since $\alpha_t(i) = P(o_1, o_2, .., o_t, q_t = i \mid \lambda)$ and $\beta_t(i) = P(o_{t+1}, o_{t+2}, .., o_T \mid q_t = i, \lambda)$, $P(O, q_t = S_1 \mid \lambda)$ can be written as the joint probability:

$$P(O, q_t = S_1 \mid \lambda) = P(o_1, o_2, .., o_t, q_t = S_i \mid \lambda) \cdot P(o_{t+1}, o_{t+2}, .., o_T \mid q_t = S_i, \lambda)$$

Now we can rewrite:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)}$$

The most likely state at time *t* is:

$$q_t^* = \arg\max_{1\le i\le N}\left[\gamma_t(i)\right],\quad 1\le t\le T$$

Even if *q** maximizes the expected number of correct states, there could be some issues with the whole sequence since we aer not taking into account transition probabilities. For instance, is some transition has null probability $a_{ij}=0$, the path between the optimal states cannot be valid. To generate a valid path we can apply the Viterbi algorithm, based on dynamic programming. $\gamma_t(i)$ is not used in the Viterbi algorithm, but it will be used in the solution of problem 3.

Viterbi Algorithm

To find the single best sequence of states, $q = (q_1, q_2, .., q_T)$, that is optimal with respect to the observation sequence $O = (o_1, o_2, .., o_T)$ and the HMM λ, in order to maximize $P(q \mid O, \lambda)$, we can apply a dynamic programming approach named *Viterbi algorithm* [Forney 1973] [Viterbi 1967].

Since:

$$P(q \mid O, \lambda) = \frac{P(q, O \mid \lambda)}{P(O \mid \lambda)}$$

$P(q \mid O, \lambda)$ maximization is equivalent to $P(q, O \mid \lambda)$ maximization in Viterbi.

The basic idea is similar to the *Forward* procedure, but in this case computation is considered only for two cosecutive time steps: *t* and *t+1*, starting with *t = 1* and finishing for *t* = *T*. The main difference is in the computation between the two time steps; *Viterbi* looks for and memorizes the highest value corresponding to the best so far sequence of states, while *Forward* procedure comuptes the summation of all results. At the end of calculation, the best values, for which the algorithm keeps track, can be used to recover the state sequence using a backtracking process.

Let us consider the following:

$$\delta_t(i) = \max_{q1,q2..qt-1} P\left(q_1..q_t = S_i, o_1..o_t \mid \lambda\right)$$

That is the probability of observing $o_1..o_t$ on the best path ending on state *i* at time *t*, given the HMM λ.

Using induction we can compute the probabibility at the next step:

$$\delta_{t+1}(i) = b_j(o_{t+1}) \max_{1 \le i \le N} [\delta_t(i) a_{ij}]$$

To find the sequence of states we need to keep track of this function argmax for all *t* and *j*. This can be easily done using an array, $\psi_t(j)$.

Here it is the complete algorithm:

1. Initialization

   t = 2;

   $\delta_1(i) = \pi_i b_i(o_1)\,, \quad 1 \le i \le N$

   $\psi_1(i) = 0\,, \quad 1 \le i \le N$

2. Induction

   $\delta_t(j) = b_j(o_t) \max_{1\le i\le N} \delta_{t-1}(i) a_{ij}\,, \quad 1 \le j \le N$

   $\psi_t(j) = \arg\max_{1\le i\le N} [\delta_{t-1}(i) a_{ij}]\,, \quad 1 \le j \le N$

3. Update

   t = t+1;

   if $t \le T$ GOTO 2, otherwise GOTO 4.

4. Final calculation

   $P^* = \max_{1\le i\le N} \left[\delta_T(i)\right]$

   $q_T^* = \arg\max_{1\le i\le N} \left[\delta_T(i)\right]$

5. State sequence reconstruction

   a. Initialization

      t = T-1

   b. Backtracking

      $q_t^* = \psi_{t+1}\left(q^*_{t+1}\right)$

c. Update

   t = t-1;

   If $t \geq 1$ GOTO *b*, otherwise end.

As previously stated we might encouter problem with underflow so we need a slightly different formulation of this algorithm.

An alternative Viterbi algorithm

To overcome the underflow issues due to probability multiplication we can use the logarithm of model parameters to transform all these multiplications into sums. We might encouter other issues when some parameters are equal to zero, sa usually happens with matrixes A e π, but we can solve them by adding a small number to them.

This new Viterbi Algorithm is:

1. Preprocessing

$$\tilde{\pi}_i = \log(\pi_i), \quad 1 \leq i \leq N$$

$$\tilde{a}_{ij} = \log(a_{ij}), \quad 1 \leq i, j \leq N$$

2. Initialization

$t = 2$;

$$\tilde{b}_i(o_1) = \log(b_i(o_1)), \quad 1 \le i \le N$$

$$\begin{aligned}\tilde{\delta}_1(i) &= \log(\delta_1(i)) \\ &= \log(\pi_i b_i(o_1)) \\ &= \tilde{\pi}_i + \tilde{b}_i(o_1), \quad 1 \le i \le N\end{aligned}$$

$$\psi_1(i) = 0, \quad 1 \le i \le N$$

3. Induction

$$\tilde{b}_j(o_t) = \log(b_j(o_t)), \quad 1 \le j \le N$$

$$\begin{aligned}\tilde{\delta}_t(j) &= \log(\delta_t(j)) \\ &= \log\left(b_j(o_t) \max_{1 \le i \le N} \delta_{t\text{-}1}(i) a_{ij}\right) \\ &= \tilde{b}_j(o_t) + \max_{1 \le i \le N}[\tilde{\delta}_{t\text{-}1}(i) + \tilde{a}_{ij}], \quad 1 \le j \le N\end{aligned}$$

$$\psi_t(j) = \arg\max_{1 \le i \le N}[\tilde{\delta}_{t\text{-}1}(i) + \tilde{a}_{ij}], \quad 1 \le j \le N$$

4. Update

$t = t+1$;

If $t \le T$ GOTO 3, otherwise GOTO 5.

5. Final computation

$$\tilde{P}^* = \max_{1 \le i \le N}[\tilde{\delta}_T(i)]$$

$$q_T^* = \arg\max_{1 \le i \le N}[\tilde{\delta}_T(i)]$$

6. State sequence reconstruction

   a. Initialization

      t = T-1

   b. Backtracking

      $$q_t^* = \psi_{t+1}(q^*_{t+1})$$

   c. Update

      t = t-1;

      if $t \geq 1$ GOTO *b*, otherwise end.

For instance, let us consider a model with $N = 3$ states and observations $T=8$. During initialization ($t = 1$) we find the values for $\delta_1(1)$, $\delta_1(2)$ and $\delta_1(3)$; suppose $\delta_1(2)$ is the maximum. In the next iteration, for $t = 2$, we find $\delta_2(1)$, $\delta_2(2)$ and $\delta_2(3)$, thi time $\delta_2(1)$ is the higher. So forth obtaining $\delta_3(3)$, $\delta_4(2)$, $\delta_5(2)$, $\delta_6(1)$, $\delta_7(3)$ and $\delta_8(3)$. The obtimal sequence is summarized in the following graph (Figure 3.12):

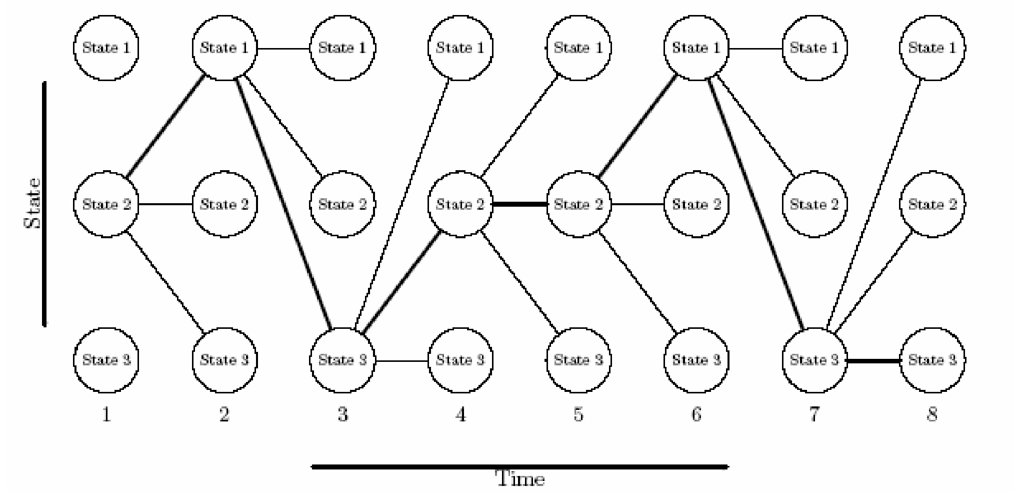


Figure Error! No text of specified style in document.**-1 Sequence of states explored by the Viterbi algorithm**

Solution to *problem 3*, parameter estimation

The learn *training* when referring to HMM is referred to $\lambda = (A, B, \pi)$ model parameters estimation. Given and observation *O* we look for the model *λ** that maximizes $P(O \mid \lambda)$ (maximum likelihood principle); this can be written as:

$$\lambda^* = \arg\max_{\lambda} \left[P(O \mid \lambda) \right]$$

This is the most challenging problem since there is no analytical solution for the paramiters that maximize the probability of observing a given sequence; for this reason a local maximum of likelihood is chosen. The most used trategies to obtain such solution are the *Baum-Welch* algorithm [Baum 1972] [Baum et al. 1970] [Dempster et al. 1977], E*xpectation-Maximization o EM*, or gradient descend. All af these use an iterative approach to get a better likelihood for $P(O \mid \lambda)$; among the others the *Baum-Welch* algorithm ha some advantages:

- Numerically stable
- It converges to a local optimum
- Has linear convergence

For these reasons Baum-Welch is the most used. This method belongs to the class of algorithms used in statistics to deal with missing values; the EM algorithm is the general form [Bilmes 1998] [McLachlan e Krishnan 1997] [Redner e Walzer 1984] and gives a procedure for local optimization.

In general with *maximum likelihood* estimation we look for the parameter *θ* that describes a distribution supposed to generate the data, *X*, so that likelihood $\xi(\theta \mid X)$ is maximum. When some data are missing, *θ* defines the distribution both for *X*, and missing data *Y*. The idea behind the EM algorithm is to compute the *complete-data likelihood* $\xi(\theta \mid X, Y)$ and find a miximum for it with the following iterative algorithm:

1. *EXPECTATION* (*E-step*): use the parameter estimated at last iteration, θ^i, and the costant *X*. Compute the expected value of *complete-data log-likelihood* and the *log-likelihood* as a function of *θ* with respect to the random variable*Y*, defined by θ^i *and X.*
2. *MAXIMIZATION* (*M-step*): select the valuo of *θ* that maximizes *expectation*, and then set $\theta^{i+1} = \theta$.

The EM algorthm does not decrease *likelihood* and converges to a local maximum, as with the Baum-Welch algorithm. To describe the estimation procedure we need to introduce the following variable:

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

It represents the probability of being in state S_i at time *t* and in state S_j at time *t+1*, given the HMM *λ* and the sequence of observations *O*, as summarized in the following graph (Figure 3.13).

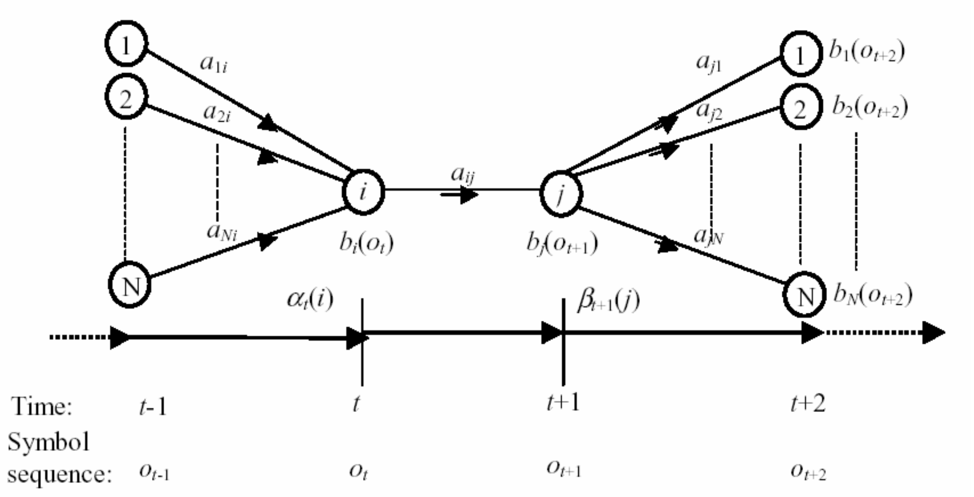


Figure Error! No text of specified style in document.**-2 Computation procedure for the probability of moving from one state the following one**

The following schema reports the posterior probability of *$\gamma_t(i)$* (Figure 3.14)

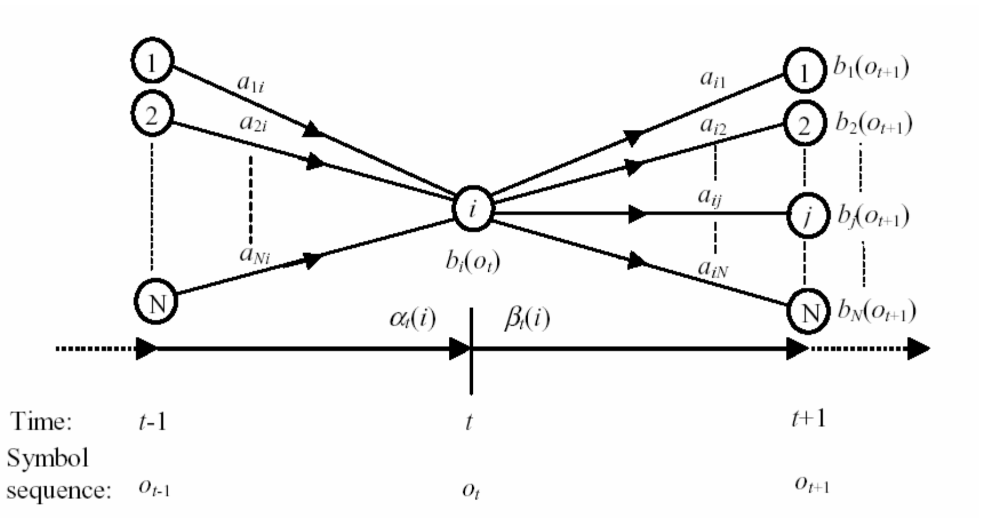


Figure Error! No text of specified style in document.**-3 Derivation of posterior probability gven a state *i***

For these reasons it holds that:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P(O\mid\lambda)}$$

$$= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}$$

As previously mentioned in problem 2, *$\gamma_t(i)$* is the probability of being in state *S_i* at time *t*, given the whole sequence of observations *O* end the HMM *λ*. At this point, we can write:

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

If we apply the summation over *t* to *$\gamma_t(i)$* we obtain a number that can be considered as the number of visits to state *S_i*, or equivalently the expected value of transitions from state *S_i*, including *$t = T$*. If we take the same summation also for *$\xi_t(i,j)$* we obtain the number of trasitions from *S_i* to *S_j*.

To summarize:

$\gamma_1(i) = \text{probability of starting in state } S_i$

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

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

From these definitions we can derive the estimation formulas:

$$\begin{aligned}
\pi_i &= \text{expected frequency of state } S_i \text{ at time t} = 1 \\
&= \gamma_1(i) \\
&= \frac{\alpha_1(i)\beta_1(i)}{\sum_{i=1}^{N} \alpha_1(i)\beta_1(i)} \\
&= \frac{\alpha_1(i)\beta_1(i)}{\sum_{i=1}^{N} \alpha_T(i)}
\end{aligned}$$

$$a_{ij} = \frac{\text{expected number of transitions from state } S_i \text{ to state } S_j}{\text{expected number of transitions from state } S_i}$$

$$= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$= \frac{\sum_{t=1}^{T-1} P(q_t = i, q_{t+1} = j, O \mid \lambda)}{\sum_{t=1}^{T-1} P(q_t = i, O \mid \lambda)}$$

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

$$b_j(k) = \frac{\text{expected number of observations for symbol k in state } S_j}{\text{expected number of occurences for state } S_j}$$

$$= \frac{\sum_{\substack{t=1 \\ Ot=k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

$$= \frac{\sum_{t=1}^{T} P(q_t = j, O \mid \lambda) \cdot \delta(O_t, q)}{\sum_{t=1}^{T} P(q_t = j, O \mid \lambda)}$$

$$= \frac{\sum_{t=1}^{T} \alpha_t(i) \beta_t(j) \delta(O_t, o_t)}{\sum_{t=1}^{T} \alpha_t(i) \beta_t(i)}$$

where $\delta(O_t, o_t) = \begin{cases} 1 \text{ if } O_t = o_t \\ 0 \text{ otherwise} \end{cases}$

It is useful to remind that:

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_t(i) \beta_t(i) = \sum_{i=1}^{N} \alpha_T(i)$$

$$P(O, q_t = S_i \mid \lambda) = \alpha_t(i) \beta_t(i)$$

$$P(O, q_t = S_i, q_{t+1} = S_j \mid \lambda) = \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$$

These parameter needs to satisfy the stochastic constraint of normalization:

$$\sum_{i=1}^{N} \pi_i = 1$$

$$\sum_{j=1}^{N} a_{ij} = 1,\ 1 \le i \le N$$

$$\sum_{ot=0}^{M-1} b_j(o_t) = 1,\ 1 \le j \le N \quad 1 \le t \le T$$

Using the scaling approach we have:

$$\begin{aligned}
\xi_t(i,j) &= \frac{\hat{\alpha}_t(i) a_{ij} b_j(o_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \hat{\alpha}_t(i) a_{ij} b_j(o_{t+1}) \hat{\beta}_{t+1}(j)} \\
&= \frac{\left(\prod_{\tau=1}^{t} c_\tau\right) \alpha_t(i) a_{ij} b_j(o_{t+1}) \left(\prod_{\tau=t+1}^{T} c_\tau\right) \beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \left(\prod_{\tau=1}^{t} c_\tau\right) \alpha_t(i) a_{ij} b_j(o_{t+1}) \left(\prod_{\tau=t+1}^{T} c_\tau\right) \beta_{t+1}(j)} \\
&= \frac{\left(\prod_{\tau=1}^{t} c_\tau\right)\left(\prod_{\tau=t+1}^{T} c_\tau\right) \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\left(\prod_{\tau=1}^{t} c_\tau\right)\left(\prod_{\tau=t+1}^{T} c_\tau\right) \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)} \\
&= \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}
\end{aligned}$$

And also that:

$$\begin{aligned}
\gamma_t(i) &= \frac{\hat{\alpha}_t(i) \hat{\beta}_t(i)}{\sum_{i=1}^{N} \hat{\alpha}_t(i) \hat{\beta}_t(i)} \\
&= \frac{\left(\prod_{\tau=1}^{t} c_\tau\right) \alpha_t(i) \left(\prod_{\tau=t}^{T} c_\tau\right) \beta_t(i)}{\sum_{i=1}^{N} \left(\prod_{\tau=1}^{t} c_\tau\right) \alpha_t(i) \left(\prod_{\tau=t}^{T} c_\tau\right) \beta_t(i)}
\end{aligned}$$

$$= \frac{\left(\prod_{\tau=1}^{t} c_\tau\right)\left(\prod_{\tau=t+1}^{T} c_\tau\right)\alpha_t(i)\beta_t(i)}{\left(\prod_{\tau=1}^{t} c_\tau\right)\left(\prod_{\tau=t+1}^{T} c_\tau\right)\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)}$$

$$= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)}$$

It can be proven that variables do not change in both formulations. Using scaled *forward* and *backward* variables we get:

$$\xi_t(i,j) = \frac{\hat{\alpha}_t(i)a_{ij}b_j(o_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\hat{\alpha}_t(i)a_{ij}b_j(o_{t+1})\hat{\beta}_{t+1}(j)}$$

$$= \frac{\hat{\alpha}_t(i)a_{ij}b_j(o_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\hat{\alpha}_t(i)\left(\sum_{j=1}^{N}a_{ij}b_j(o_{t+1})\hat{\beta}_{t+1}(j)\right)}$$

$$= \frac{\hat{\alpha}_t(i)a_{ij}b_j(o_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\hat{\alpha}_t(i)\hat{\beta}_t(j)}$$

$$= \frac{\hat{\alpha}_t(i)a_{ij}b_j(o_{t+1})\hat{\beta}_{t+1}(j)}{\frac{1}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(j)}\sum_{i=1}^{N}\alpha_t(i)\beta_t(j)}$$

$$= \hat{\alpha}_t(i)a_{ij}b_j(o_{t+1})\hat{\beta}_{t+1}(j)$$

Since we use *$\xi_t(i,j)$* and *$\gamma_t(i)$* to compute *A* e *π* the probabilities are equivalent also after scaling so, avoiding nunmerical issues, we can write:

$$\pi_i = \frac{\hat{\alpha}_1(i)\hat{\beta}_1(i)}{\sum_{i=1}^{N}\hat{\alpha}_T(i)}$$

but also:

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

At the same time we can use these variables to estimate the coefficients of the emission matrix B.

This process iterates for a given number of iterations or up to a threshold in the difference between models measured as the difference between α before and after iteration. The following schema (Figure 3.15) summarrize the steps to compute these parameters starting from an initial guess.

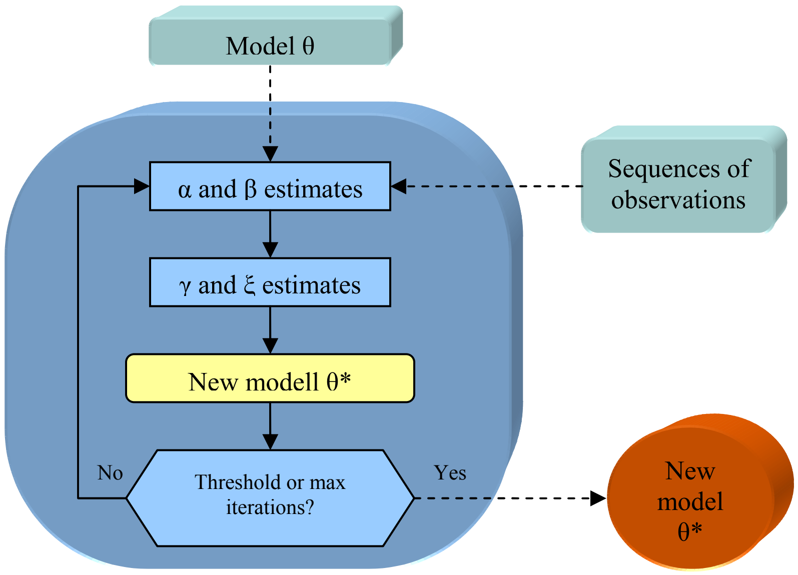


Figure Error! No text of specified style in document.**-4 Schema of the iterative algorithm for HMM parameters estimation**

Multiple sequences of observations

In the Baum-Welch algorithm we need to compute the values for many parameters; to obtain robust estimates we need a sufficint amount of training data. We cannot use a single sequence otherwise the model would be able to recognize just that one, but it would not perform properly on new ones. For this reason it needs to be changed few times (at the risk of numerical instabilities).

We need to adapt the algorithm to multiple sequences; let be *R* a set of seuences of observations:

$$O = [O^1, O^2, .., O^R]$$

Where $O^r = (o_1^r, o_2^r, .., o_T^r)$ is the r-th sequence of observation with length *T*. The new likelihood to be maximized *P(O|λ)* becomes:

$$P(O \mid \lambda) = \prod_{r=1}^{R} P(O^r \mid \lambda)$$

The soluttion to this problem is similar to the standard procedure; variables are estimated for each training example on cumuled values to get parameters averaging the optimal solution for each sequence. This procedure is then repeated until and optimal value is found or we reached a maximum number of iterations.

Formulas become:

$$\pi_i = \frac{\sum_{r=1}^{R} \hat{\alpha}_1(i)^r \hat{\beta}_1^{\,r}(i)}{\sum_{r=1}^{R} \sum_{i=1}^{N} \hat{\alpha}_T^{\,r}(i)}$$

$$a_{ij} = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T-1} \hat{\alpha}_t^{\,r}(i) a_{ij} b_j\left(o^r{}_{t+1}\right) \hat{\beta}^r{}_{t+1}(j)}{\sum_{r=1}^{R} \sum_{t=1}^{T-1} \hat{\alpha}_t^{\,r}(i) \hat{\beta}_t^{\,r}(i)}$$

$$b_j(k) = \frac{\sum_{r=1}^{R} \sum_{\substack{t=1 \\ Ot=k}}^{T} \gamma_t(j)}{\sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(j)}$$

Parameter initialization

Before we can start training, it is important to initialize in a proper way our guess for the model so that we can approach a global optimum as much as possible. Usually an uniform distribution for *π* and *A* is used, but if we apply the *left-right* model, *π* has probability 1 for the first state and zero for the others.

For instance we could have:

$$\pi = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}_{4x1}$$

$$A = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{4x4}$$

Emission matrix require a good initial estimate to get a proper convergence. This is usually obtained doing a uniform segmentation of states for each example, and collectiong all the observations for a given state *S_j* from the training examples. Then a clustering algorithm is applied to find initial estimates for each state.

The training, i.e., the sequence of observed data, could be too small since we cannot collect enough data. In this case, we might have almost no occurences for unlikely states or observations; this results in uncorrect model parameters estimation.

If the number of symbol sequences is so small not to have occurences for some observations, initial zero estimates of model parameters will not be changed and will result in zero probability for new sequences. This is due to an improper estimate of parameters due to the scarce date. To reduce this phaenomenon, there are a few strategies [Rabiner 1989] [Lee 1988], such as the use of *codebook*, reducing the number of symbols for each state or the whole number of states. The easier method to face data shortage is to introduce a constant value ε as a lower bound each model parameter at each iteration. This turns out into:

$$\pi_i = \begin{cases} \pi_i & \text{se } \pi_i \geq \varepsilon_\pi \\ \varepsilon_\pi & \text{se } \pi_i < \varepsilon_\pi \end{cases}$$

$$a_{ij} = \begin{cases} a_{ij} & \text{se } a_{ij} \geq \varepsilon_a \\ \varepsilon_a & \text{se } a_{ij} < \varepsilon_a \end{cases}$$

$$b_i(o_t) = \begin{cases} b_i(o_t) & \text{se } b_i(o_t) \geq \varepsilon_b \\ \varepsilon_b & \text{se } b_i(o_t) < \varepsilon_b \end{cases}$$

In practice $\varepsilon_\pi = \varepsilon_a = \varepsilon_b = 0{,}0001$. Moreover, as with any probability distribution, we need to satisfy the normalization property:

$$\sum_{i=1}^{N} \pi_i = 1$$

$$\sum_{j=1}^{N} a_{ij} = 1,\ 1 \leq i \leq N$$

$$\sum_{ot=0}^{M-1} b_j(o_t) = 1,\ 1 \leq j \leq N \quad 1 \leq t \leq T$$

Since these probabilities are changed at any iteration this property might not hold (expecially if we use a smoothing approach as previously described). Thi simply requires to compute at each iteration:

$$\pi_i = \frac{\pi_i}{\sum_{k=1}^{N} \pi_k}, \quad 1 \le i \le N$$

$$a_{ij} = \frac{a_{ij}}{\sum_{k=1}^{N} a_{ik}}, \quad 1 \le i, j \le N$$

$$b_j(o_t) = \frac{b_j(o_t)}{\sum_{k_t=0}^{M-1} b_j(k_t)}, \quad 1 \le j \le N \text{ e } 0 \le o_t \le M-1$$