Blekinge Institute of Technology
Research Report 2005:02

First Order Hidden Markov Model
Theory and Implementation Issues

Mikael Nilsson

Department of Signal Processing
Blekinge Institute of Technology

First Order Hidden Markov Model

Theory and Implementation Issues

Mikael Nilsson

February, 2005

Research Report
Department of Signal Processing
Blekinge Institute of Technology, Sweden

Abstract

This report explains the theory of Hidden Markov Models (HMMs). The emphasis is on the theory aspects in conjunction with the implementation issues that are encountered in a floating point processor. The main theory and implementation issues are based on the use of a Gaussian Mixture Model (GMM) as the state density in the HMM, and a Continuous Density Hidden Markov Model (CDHMM) is assumed. Suggestions and advice related to the implementation are given for a typical pattern recognition task.

Contents

Chapter 1

Introduction

In statistical pattern recognition the aim is to make supervised or unsupervised classification. Various methods and applications have been suggested in the literature, a good review can be found in [1].

The sequential pattern recognition, which can be seen as a special case of statistical pattern recognition, is currently mostly approached by the use of the Hidden Markov Model (HMM). As a matter of fact, current state-of-the-art speech recognizer systems are based on HMMs. The fundamentals for speech recognition can be found in [2]. Although the HMMs originally emerged in the domain of speech recognition, other interesting fields have shown an interest for the use of these stochastic models, such as handwriting recognition [3, 4, 5], biological sequence analysis [6, 7, 8], face recognition [9, 10, 11], etc.

In this report the pattern recognition task using HMMs with gaussian mixtures is approached from a more general perspective. This implies that the algorithms and numerical issues addressed are essential for all pattern recognition tasks using HMMs and gaussian mixtures on a floating point processor.

The outline of this report is as follows:

Chapter 1 - Introduction

This chapter.

Chapter 2 - Discrete-Time Markov Model

Explains the fundamentals of a Markov model, i.e. the *hidden* part is uncovered. As an example, the weather is modelled by a Markov model and the state duration distribution is derived as well.

Chapter 3 - Discrete-Time Hidden Markov Model

The Markov model from the previous chapter is extended to the HMM. An

urn and ball model example is given as an introduction to HMMs. Discrete and continuous observation densities are discussed, as well as different model structures and their properties.

Chapter 4 - Three Basic Problems for Hidden Markov Models

The major problems for the HMM, probability estimation, "optimal" state sequence and parameter estimation, are addressed.

Chapter 5 - Solution to Problem 1 - Probability Evaluation

How to find the probability for an observation sequence is addressed here and the theoretical textitforward and textitbackward algorithm is explained. The chapter includes practical implementation aspects yielding the *scaled forward* and *scaled backward* algorithm.

Chapter 6 - Solution to Problem 2 - "Optimal" State Sequence

The problem of finding an "optimal" state sequence is investigated. The theoretical *Viterbi* algorithm is derived. The more practical implementation of the Viterbi algorithm leads into the *alternative Viterbi* algorithm and the *fast alternative Viterbi* algorithm.

Chapter 7 - Solution to Problem 3 - Parameter Estimation

The Baum-Welch reestimation formulas for a HMM are derived. The training formulas are extended to multiple training sequences. Numerical issues are investigated and suggestions for the initial values for the state distribution, the state transition matrix and the Gaussian mixture are given.

Chapter 8 - Clustering Algorithms

The unsupervised classification is approached, with the aim to find the initial parameters for the Gaussian mixture. Two commonly used clustering algorithms are explained: the k-means algorithm and the Self Organizing Feature Map (SOFM).

Chapter 9 - Training and Recognition using Hidden Markov Models

Addresses issues for a complete pattern recognition task using HMMs. The training steps are discussed as well as the numerical issues needed for robust HMM training. Pattern recognition and suggestions for rejection of non-seen classes are discussed.

Chapter 7 - Summary and Conclusions

Finally, the results and observations in the report are summarized and concluded.

Chapter 2

Discrete-Time Markov Model

This chapter describes the theory of Markov chains and it is supposed that the *hidden* part of the Markov chain is uncovered from the HMM, i.e. an observable Markov model is investigated. A a system is considered that may be described at any time, as being at one of N distinct states, indexed by $1, 2, \ldots, N$. At regularly spaced, discrete times, the system undergoes a change of state (possibly back to the same state) according to a set of probabilities associated with the state. The time instances for a state change is denoted t and the corresponding state q_t. In the case of a first order Markov chain, the state transition probabilities do not depend on the whole history of the process, only the preceding state is taken into account. This leads into the first order Markov property defined as

$$P\left(q_t = j | q_{t-1} = i, q_{t-2} = k, \ldots\right) = P\left(q_t = j | q_{t-1} = i\right) \tag{2.1}$$

where P is the probability and i,j,k are state indexes. Note that the right hand of Eq. (2.1) is independent of time, which gives the state transitions probabilities, a_{ij}, to be

$$a_{ij} = P\left(q_t = j | q_{t-1} = i\right), \ 1 \leq i, j \leq N \tag{2.2}$$

Due to standard stochastic constraints [2], the state probabilities, a_{ij}, have the following properties

$$\begin{aligned} a_{ij} &\geq 0 \qquad \forall j, i \\ \sum_{j=1}^{N} a_{ij} &= 1 \quad \forall i \end{aligned} \tag{2.3}$$

The state transition probabilities for all states in a model can be described by a transition probability matrix

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix}_{N \times N} \tag{2.4}$$

The probability to start in a state, i.e. the initial state distribution vector, is described by

$$\boldsymbol{\pi} = \begin{bmatrix} \pi_1 = P(q_1 = 1) \\ \pi_2 = P(q_1 = 2) \\ \vdots \\ \pi_N = P(q_1 = N) \end{bmatrix}_{N \times 1} \tag{2.5}$$

and the corresponding stochastic property for the initial state distribution vector is

$$\sum_{i=1}^{N} \pi_i = 1 \tag{2.6}$$

where the π_i is defined as

$$\pi_i = P(q_1 = i), \quad 1 \leq i \leq N \tag{2.7}$$

Equation (2.1) - (2.7) describes the properties and equations for a first order Markov process, hence the Markov model itself can be described by $\mathbf{A}$ and $\boldsymbol{\pi}$.

2.1 Markov Model Example

In this section an example of a discrete time Markov process will be presented which leads into the main ideas about Markov chains. A four state Markov model of the weather will be used as an example, see Fig. 2.1.

It is assumed that the weather is observed at a certain time every day (e.g. in the morning), and that it is observed and classified as belonging to one of the following states:

- **State 1:** cloudy
- **State 2:** sunny
- **State 3:** rainy
- **State 4:** windy

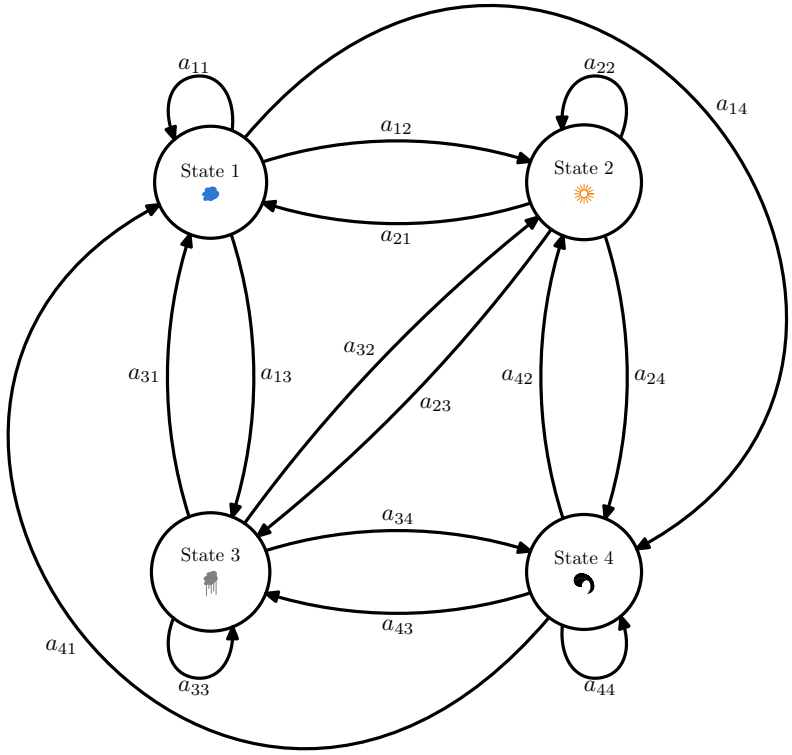


Figure 2.1: Markov model of the weather.

Given the model in Fig. 2.1, it is possible to answer several questions about the weather patterns over time. For example, what is the probability to get the sequence "sunny, rainy, sunny, windy, cloudy, cloudy" in six consecutive days? The first task is to define the state sequence, $\mathbf{O}$, as

$$\mathbf{O} = (\text{sunny}, \text{rainy}, \text{sunny}, \text{windy}, \text{cloudy}, \text{cloudy}) = (2,\ 3,\ 2,\ 4,\ 1,\ 1)$$

Given the sequence and the model of Fig. 2.1, the probability calculation of the observation sequence given a Markov model, $P(\mathbf{O}|\mathbf{A}, \boldsymbol{\pi})$, can directly be determined as

$$\begin{aligned} P(\mathbf{O}|\mathbf{A}, \boldsymbol{\pi}) &= P(2, 3, 2, 4, 1, 1|\mathbf{A}, \boldsymbol{\pi}) \\ &= P(2)P(3|2)P(2|3)P(4|2)P(1|4)P(1|1) \\ &= \pi_2 \cdot a_{23} \cdot a_{32} \cdot a_{24} \cdot a_{41} \cdot a_{11} \end{aligned}$$

In a more general sense, the probability calculation of a state sequence $\mathbf{q} = (q_1, q_2, \ldots, q_T)$ can be determined as

$$P(\mathbf{q}|\mathbf{A}, \boldsymbol{\pi}) = \pi_{q_1} \cdot a_{q_1 q_2} \cdot a_{q_2 q_3} \cdot \ldots \cdot a_{q_{T-1} q_T} \tag{2.8}$$

The second task is to find the probability for the state to remain the same for d time instances, given that the system is in a known state. In the weather example, the time instances correspond to days, yielding the following observation sequence

$$\begin{array}{rcl} \mathbf{O} & = & (i,\ i,\ i,\ \ldots,\ i,\ j \neq i) \\ \text{day} & & 1\ \ 2\ \ 3 \qquad\quad d\ \ d+1 \end{array}$$

with the probability

$$\begin{aligned} P(\mathbf{O}|\mathbf{A}, \boldsymbol{\pi}, q_1 = i) &= P(\mathbf{O}, q_1 = i|\mathbf{A}, \boldsymbol{\pi})/P(q_1 = i) \\ &= \pi_i (a_{ii})^{d-1}(1 - a_{ii})/\pi_i \\ &= (a_{ii})^{d-1}(1 - a_{ii}) \\ &= p_i(d) \end{aligned} \tag{2.9}$$

In Eq. 2.9, $p_i(d)$ can be interpreted as the state duration distribution and the characteristic exponential distribution for a Markov chain appears. Based on $p_i(d)$ in Eq. 2.9 it is possible to find the expected number of observations (duration) in a state as

$$\begin{aligned} E[d_i] &= \sum_{d=1}^{\infty} d \cdot p_i(d) \\ &= \sum_{d=1}^{\infty} d \cdot a_{ii}^{d-1}(1 - a_{ii}) \\ &= (1 - a_{ii}) \frac{\partial}{\partial a_{ii}} \left(\sum_{d=1}^{\infty} a_{ii}^{d} \right) \\ &= (1 - a_{ii}) \frac{\partial}{\partial a_{ii}} \left(\frac{a_{ii}}{1 - a_{ii}} \right) \\ &= \frac{1}{(1 - a_{ii})} \end{aligned} \tag{2.10}$$

Chapter 3

Discrete-Time Hidden Markov Model

The discrete-time Markov model described in the previous chapter is to to some extent restrictive and can not be applicable to all problems of interest. Therefore, the Hidden Markov Model (HMM)is introduced in this chapter. The extension implies that every state will be probabilistic and not deterministic (e.g. sunny). This means that every state generates an observation at time t, $\mathbf{o}_t$, according to a probabilistic function $b_j(\mathbf{o}_t)$ for each state j. The production of observations in the stochastic approach is characterized by a set of observation probability measures, $\mathbf{B} = \{b_j(\mathbf{o}_t)\}_{j=1}^{N}$, where N is the total number of states and the probabilistic function for each state j is

$$b_j(\mathbf{o}_t) = P\left(\mathbf{o}_t | q_t = j\right) \tag{3.1}$$

The concept of how the HMM works is introduced by using an example called *the urn and ball model*.

3.1 The Urn and Ball Model

Assume that there are N large glass urns with colored balls. There are M distinct colors of the balls, see Fig. 3.1.

The steps needed for generating an observation sequence are:

1. Choose an initial state $q_1 = i$ according to the initial state distribution $\boldsymbol{\pi}$. In the urn and ball model the urn corresponds to a state.
2. Set $t = 1$ (i.e. clock, $t = 1, 2, \ldots, T$).
3. Choose a ball from the selected urn (state) according to the symbol probability distribution in state i, $b_i(\mathbf{o}_t)$. This colored ball represents the observation $\mathbf{o}_t$. Put the ball back to the urn. For example, the probability for a purple ball is in the first urn 0.61, see Fig. 3.1.

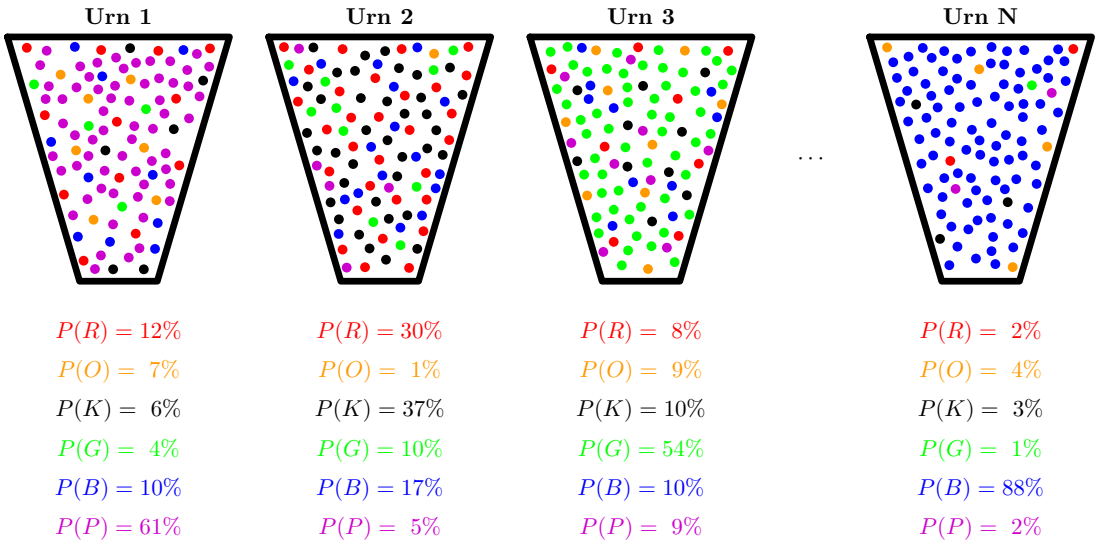


Figure 3.1: Urn and ball example with N urns and $M = 6$ different colors (R = red, O=orange, K=black, G=green, B=blue, P=purple).

4. Transit to a new state (urn) $q_{t+1} = j$ according to the state-transition probability distribution for state i, i.e. a_{ij}.

5. Set $t = t + 1$; return to step 3 if $t < T$; otherwise, terminate the procedure.

These steps describe how the HMM works when generating an observation sequence. It should be noted that the urns can contain balls of the same color, and the distinction among various urns is the way the collection of the colored balls is composed. Therefore, an isolated observation of a ball of a particular color does not immediately indicate which urn it is drawn from. Furthermore, the link between the urn and ball example and a HMM, is that the urn corresponds to a state and the color corresponds to a feature vector (the observation).

3.2 Discrete Observation Densities

The urn and ball example described in the previous section is an example of a discrete observation density HMM with M distinct codewords (colors). In general, the discrete observation density partitions the probability density function (pdf) into M codewords or cells with assigned probabilities. These codewords will be denoted $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M$ where one symbol corresponds to one cell. The partitioning is usually called *vector quantization* and for this purpose several analysis methods exist [2]. Vector quantization is performed by creating a codebook based on the mean vectors for every cluster.

The symbol for the observation is determined by the nearest neighbor rule, i.e. the symbol in the cell with the nearest codebook vector is selected. In the

urn and ball example this corresponds to the dark grey ball being classified as a black, due to that this is the most probable choice. In this case the symbols $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M$ are represented by different colors (e.g. $\mathbf{v}_1$ =red). The probability distributions for the observable symbols are denoted $\mathbf{B} = \{b_j(\mathbf{o}_t)\}_{j=1}^{N}$, where the symbol distribution in state j is defined as

$$b_j(\mathbf{o}_t) = b_j(k) = P\left(\mathbf{o}_t = \mathbf{v}_k | q_t = j\right), \quad 1 \leq k \leq M \tag{3.2}$$

This is the first step in the determination of the codebook. The second step is to estimate the sets of observation probabilities for each codebook vector in every state respectively. The major problem with discrete output probability is the vector quantization operation partitioning which destroys the original signal structure, when the continuous space is separated into regions. If the output distributions are overlapping the partitioning introduces errors. This is due to a finite training data set being used implying unreliable parameter estimates. The vector quantization errors introduced may cause performance degradation of the discrete density HMM [2]. This will occur when the observed feature vector is intermediate between two codebook symbols. This is why *continuous observation densities* can be used instead of discrete observation densities. The continuous approach avoids quantization errors. However this approach typically require more calculations.

3.3 Continuous Observation Densities

The continuous observation densities $b_j(\mathbf{o}_t)$ in the HMM are created by using a parametric probability density function or a mixture of several functions. To be able to reestimate the parameters of the probability density function (pdf) some restrictions for the pdf are needed. A common restriction is that the pdf should belong to the log-concave or elliptically symmetrical density family [2]. The most general representation of the pdf, for which a reestimation procedure has been formulated, is the finite mixture of the form

$$b_j(\mathbf{o}_t) = \sum_{k=1}^{M} c_{jk} b_{jk}(\mathbf{o}_t), \quad j = 1, 2, \ldots, N \tag{3.3}$$

where M is the number of mixtures. The stochastic constraints for the mixture weights, c_{jk}, must fulfil

$$\begin{aligned} &\sum_{k=1}^{M} c_{jk} = 1 \quad && j = 1, 2, \ldots, N \\ &c_{jk} \geq 0 && j = 1, 2, \ldots, N, \ k = 1, 2, \ldots, M \end{aligned} \tag{3.4}$$

The $b_{jk}(\mathbf{o}_t)$ can be either D-dimensional log-concave or of elliptically symmetric density with mean vector $\boldsymbol{\mu}_{jk}$ and covariance matrix $\boldsymbol{\Sigma}_{jk}$

$$b_{jk}(\mathbf{o}_t) = \mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) \tag{3.5}$$

The most utilized D-dimensional log-concave or elliptically symmetric density function, is the Gaussian density function

$$b_{jk}(\mathbf{o}_t) = \mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_{jk}|^{1/2}} e^{-\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)} \tag{3.6}$$

To approximate a basic observation source, the Gaussian mixture provides a straightforward solution to gaining a considerable accuracy. This is due to the flexibility and convenient estimation of the pdfs. If the observation source generates a complicated high-dimensional pdf, the Gaussian mixture becomes computationally difficult to treat, due to the considerable number of parameters to estimate and large covariance matrices. To estimate these parameters there is a need of a huge amount of well representative training data, this is, however, hard to achieve in a practical situation. With insufficient training data sets the parameter estimation will be unreliable. Especially the covariance matrix estimation is highly sensitive for the lack of representative training data.

The size of the covariance matrices increases with the square proportional to the vector dimension D. If the feature vectors are designed to avoid redundant components, the elements outside the diagonal of the covariance matrices are usually small. One solution is to use a diagonal covariance matrix approximation. The diagonality provides a simpler and a faster implementation for the probability computation. This is performed by reducing Eq. (3.6) to

$$\begin{aligned} b_{jk}(\mathbf{o}_t) &= \mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_{jk}|^{1/2}} e^{-\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)} \\ &= \frac{1}{(2\pi)^{D/2}\left(\prod_{d=1}^{D}\sigma_{jkd}\right)^{1/2}} e^{-\sum_{d=1}^{D}\frac{\left(o_{tl} - \mu_{jkd}\right)^2}{2\sigma_{jkd}^2}} \end{aligned} \tag{3.7}$$

where the variance terms $\sigma_{jk1}, \sigma_{jk2}, \ldots, \sigma_{jkD}$ are the diagonal elements of the covariance matrix $\boldsymbol{\Sigma}_{jk}$.

3.4 Types of Hidden Markov Models

Different kinds of structures for HMMs can be used. The structure is defined by the transition matrix, $\mathbf{A}$. The most general structure is the *ergodic* or fully

connected HMM. In this model every state can be reached from every other state of the model.

As an example in this section a $N = 4$ state model is introduced, see Fig. 3.2a. The ergodic model has the property $0 < a_{ij} < 1$ where the "zero" and the "one" have been excluded in order to fulfill the ergodic property. The state transition matrix, $\mathbf{A}$, for an ergodic model, can be described by

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}_{4\times 4} \tag{3.8}$$

In some pattern recognition tasks, it is desirable to use a model which models the observations in a successive manner. This can be done by employing the left-right model or Bakis model, see Figs. 3.2b,c. The property for a left-right model is

$$a_{ij} = 0, \quad j < i \tag{3.9}$$

This implies that no transitions can be made to previous states. The lengths of the transitions are usually restricted to some maximum length Δ, typically two or three

$$a_{ij} = 0, \quad j > i + \Delta \tag{3.10}$$

Note that for a left-right model, the state transitions coefficients for the last state have the following property

$$\begin{aligned} a_{NN} &= 1 \\ a_{Nj} &= 0, \quad j < N \end{aligned} \tag{3.11}$$

If $\Delta = 1$ yields that

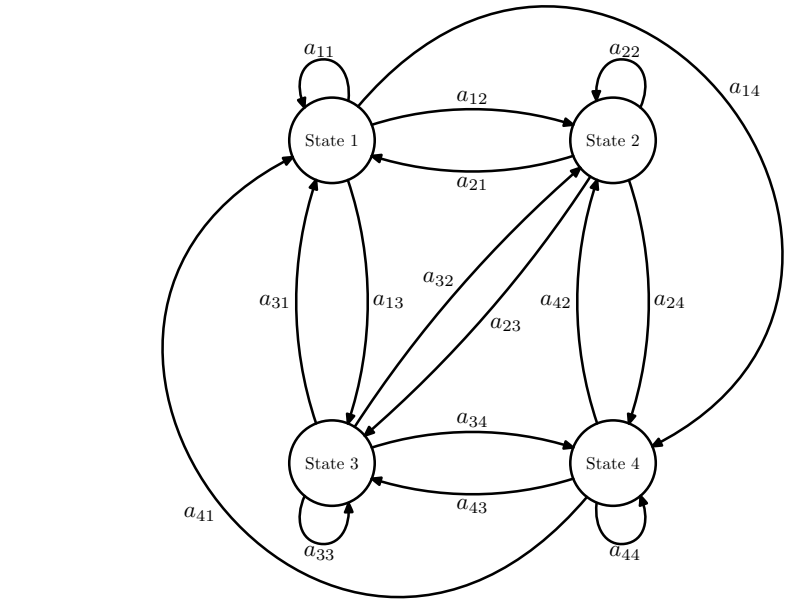
$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}_{4\times 4} \tag{3.12}$$

which corresponds to Fig. 3.2b.

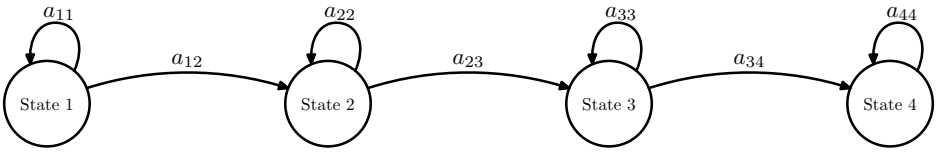
If $\Delta = 2$ yields that

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}_{4\times 4} \tag{3.13}$$

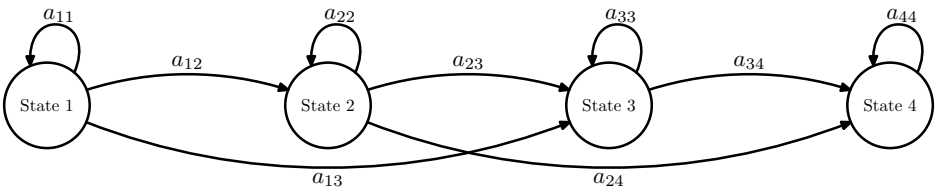
which corresponds to Fig. 3.2c.



(a)



(b)



(c)

Figure 3.2: Different structures for HMMs.

The choice of a constrained model structure (for example the left-right model) requires no modification in the training algorithms, due to the fact that an initial state transition probability set to zero will remain zero.

3.5 Summary of HMM parameters

The elements of a discrete-time hidden Markov model described in section 3.1-3.4 will be summarized in this section.

1. The number of states is denoted N. Although the states are hidden, for many practical applications there is often some physical significance attached to the states or to the sets of states of the model [2]. For instance, in the urn and ball model the states correspond to the urns. The labels for the individual states are $\{1, 2, \ldots, N\}$, and the state at time t is denoted q_t.

2. The model parameter is denoted M. If discrete observation densities are used, the parameter M is the number of classes or cells, e.g. M equals the number of colors in the urn and ball example. If continuous observation densities are used, M represents the number of mixtures in every state.

3. The initial state distribution is denoted $\boldsymbol{\pi} = \{\pi_i\}_{i=1}^{N}$, where π_i is defined as

$$\pi_i = P(q_1 = i) \tag{3.14}$$

4. The state transition probability distribution is denoted $\mathbf{A} = [a_{ij}]$, where

$$a_{ij} = P\left(q_{t+1} = j | q_t = i\right), \quad 1 \leq i, j \leq N \tag{3.15}$$

5. The observation symbol probability distribution is denoted $\mathbf{B} = \{b_j(\mathbf{o}_t)\}_{j=1}^{N}$, where the probabilistic function for each state, j, is

$$b_j(\mathbf{o}_t) = P\left(\mathbf{o}_t | q_t = j\right) \tag{3.16}$$

   The calculation of $b_j(\mathbf{o}_t)$ can be found with discrete- or continuous observation densities.

This leads into the complete specification of a HMM requiring two model parameters N and M. The specification of the three sets of probability measures $\boldsymbol{\pi}$, $\mathbf{A}$ and $\mathbf{B}$ is necessary. A simplified and commonly used notation for the probability measures is

$$\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi}) \tag{3.17}$$

Chapter 4

Three Problems for Hidden Markov Models

Given the basics of a HMM from the previous chapter, three central problems arise for applying the model to real-world applications.

Problem 1

The observation sequence is denoted $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T,)$, and the HMM $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$. The question is how to compute the probability of the observation sequence for a given HMM, $P(\mathbf{O}|\lambda)$.

Problem 2

The observation sequence is denoted $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T,)$, and the HMM $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$. The question is how to choose the corresponding state sequence, $\mathbf{q} = (q_1, q_2, \ldots, q_T)$, in an optimal sense (i.e. best "explains" the observations).

Problem 3

How can the probability measures, $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$, be adjusted to maximize $P(\mathbf{O}|\lambda)$?

The first problem can be viewed as a recognition problem. With a number of trained models the task is to find a model which best describes the observation sequence given.

In the second problem the task is to uncover the hidden part of the model. In HMMs it is more or less impossible to find the "correct" state sequence, except in the case of degenerated models. The problem should be solved in some optimal sense, which will be explained later.

The third problem can be viewed as the training problem, i.e. models are created for each specific application based on the corresponding training sequences. The training problem is the hardest and most crucial one in most applications.

The HMM should be optimally adapted based on the training data to fit and describe real world phenomena as well as possible [2].

Chapter 5

Solution to Problem 1 - Probability Evaluation

The aim is to find the probability of the observation sequence, $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T,)$, given the model λ, i.e., $P(\mathbf{O}|\lambda)$. The observations produced by the states are assumed to be independent. The probability of the observation sequence $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T)$ generated by a state sequence $\mathbf{q}$ can be calculated by the product

$$P(\mathbf{O}|\mathbf{q}, B) = b_{q_1}(\mathbf{o}_1) \cdot b_{q_2}(\mathbf{o}_2) \cdot \ldots \cdot b_{q_T}(\mathbf{o}_T) \tag{5.1}$$

The probability of the state sequence, $\mathbf{q}$ can be found as[1]

$$P(\mathbf{q}|\mathbf{A}, \boldsymbol{\pi}) = \pi_{q_1} \cdot a_{q_1 q_2} \cdot a_{q_2 q_3} \cdot \ldots \cdot a_{q_{T-1} q_T} \tag{5.2}$$

The joint probability of $\mathbf{O}$ and $\mathbf{q}$, i.e. the probability that $\mathbf{O}$ and $\mathbf{q}$ occur simultaneously, is the product of the above two terms

$$\begin{aligned} P(\mathbf{O}, \mathbf{q}|\lambda) &= P(\mathbf{O}|\mathbf{q}, B) \cdot P(\mathbf{q}|\mathbf{A}, \boldsymbol{\pi}) \\ &= \pi_{q_1} b_{q_1}(\mathbf{o}_1) a_{q_1 q_2} b_{q_2}(\mathbf{o}_2) \cdot \ldots \cdot a_{q_{T-1} q_T} b_{q_T}(\mathbf{o}_T) \\ &= \pi_{q_1} b_{q_1}(\mathbf{o}_1) \prod_{t=2}^{T} a_{q_{t-1} q_t} b_{q_t}(\mathbf{o}_t) \end{aligned} \tag{5.3}$$

The aim is to find $P(\mathbf{O}|\lambda)$, and the probability of $\mathbf{O}$ (given the model λ) is obtained by summing the joint probability of all possible state sequences $\mathbf{q}$,

[1] Explained earlier in Sec. 2.1.

$$
\begin{aligned}
P(\mathbf{O}|\lambda) &= \sum_{\text{all } \mathbf{q}} P(\mathbf{O}|\mathbf{q}, B) \cdot P(\mathbf{q}|\mathbf{A}, \boldsymbol{\pi}) \\
&= \sum_{q_1, q_2, \ldots, q_T} \pi_{q_1} b_{q_1}(\mathbf{o}_1) a_{q_1 q_2} b_{q_2}(\mathbf{o}_2) \cdot \ldots \cdot a_{q_{T-1} q_T} b_{q_T}(\mathbf{o}_T) \\
&= \sum_{q_1, q_2, \ldots, q_T} \pi_{q_1} b_{q_1}(\mathbf{o}_1) \prod_{t=2}^{T} a_{q_{t-1} q_t} b_{q_t}(\mathbf{o}_t)
\end{aligned}
\tag{5.4}
$$

The interpretation of the computation in Eq. 5.4 is the following. Initially at time $t = 1$, the process starts by jumping to state q_1 with the probability π_{q_1}. In this state the observation symbol $\mathbf{o}_1$ is generated with probability $b_{q_1}(\mathbf{o}_1)$. The clock advances from t to $t + 1$ and a transition from q_1 to q_2 will occur with the probability $a_{q_1 q_2}$. The symbol $\mathbf{o}_2$ will be generated with the probability $b_{q_2}(\mathbf{o}_2)$. The process continues in the same manner until the last transition is made (at time T), i.e. a transition from q_{T-1} to q_T will occur with the probability $a_{q_{T-1} q_T}$, and the symbol $\mathbf{o}_T$ will be generated with the probability $b_{q_T}(\mathbf{o}_T)$.

The direct computation has one major drawback. The direct computation is infeasible due to the exponential growth of computations as a function of sequence length T. To be exact, it needs $(2T-1)N^T$ multiplications, and $N^T - 1$ additions [2]. Even for small values of N and T; e.g. for $N = 5$ (states), $T = 100$ (observations), there is a need for $(2 \cdot 100 - 1)5^{100} \approx 1.6 \cdot 10^{72}$ multiplications and $5^{100} - 1 \approx 8.0 \cdot 10^{69}$ additions! It is obvious that a more efficient procedure is required to solve this kind of problem. An excellent tool, which cuts the computational cost to linear, relative to T, is the forward algorithm.

5.1 The Forward Algorithm

A forward variable $\alpha_t(i)$, is defined as

$$
\alpha_t(i) = P(\mathbf{o}_1 \mathbf{o}_2 \ldots \mathbf{o}_t, q_t = i|\lambda) \tag{5.5}
$$

where t represents time and i is the state. This gives that $\alpha_t(i)$ will be the probability of the partial observation sequence, $\mathbf{o}_1 \mathbf{o}_2 \ldots \mathbf{o}_t$, (until time t) when being in state i at time t. The forward variable can be calculated inductively, see Fig. 5.1.

The $\alpha_{t+1}(i)$ is found by summing the forward variable for all N states at time t multiplied with the corresponding state transition probability, a_{ij}, and by the emission probability $b_j(\mathbf{o}_{t+1})$. This can be performed by using the following procedure:

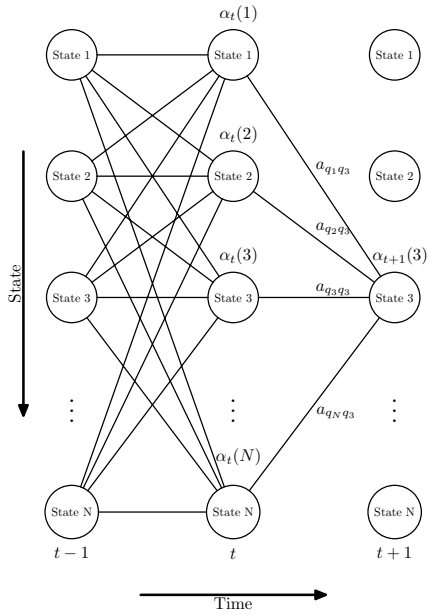


Figure 5.1: Forward Procedure - Induction Step.

1. **Initialization**

$$\begin{aligned} \text{Set } t &= 1; && (5.6)\\ \alpha_1(i) &= \pi_i b_i(\mathbf{o}_1), \quad 1 \le i \le N && (5.7) \end{aligned}$$

2. **Induction**

$$\alpha_{t+1}(j) = b_j(\mathbf{o}_{t+1}) \sum_{i=1}^{N} \alpha_t(i) a_{ij}, \quad 1 \le j \le N \tag{5.8}$$

3. **Update time**

Set $t = t + 1$;
Return to step 2 if $t < T$;
Otherwise, terminate the algorithm (goto step 4).

4. **Termination**

$$P(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{5.9}$$

The forward algorithm needs $N(N+1)(T-1)+N$ multiplications and $N(N-1)(T-1)$ additions. Again, for $N = 5$ (states), $T = 100$ (observations), this yields $5(5+1)(100-1)+5 = 2975$ multiplications and $5(5-1)(100-1) = 1980$ additions. This is a significant improvement compared to the direct method ($1.6 \cdot 10^{72}$ multiplications and $8.0 \cdot 10^{69}$ additions).

5.2 The Backward Algorithm

The recursion described in the forward algorithm can be used backwards in time as well. The backward variable $\beta_t(i)$ is defined as

$$\beta_t(i) = P\left(\mathbf{o}_{t+1}\mathbf{o}_{t+2}\ldots\mathbf{o}_T|q_t = i, \lambda\right) \tag{5.10}$$

The backward variable can be interpreted as the probability of the partial observation sequence from $t+1$ to the end (T), given state i at time t and the model λ. Note that the definition for the forward variable is a joint probability whereas the backward probability is a conditional probability. In a similar manner (according to the forward algorithm), the backward variable can be calculated inductively, see Fig. 5.2.

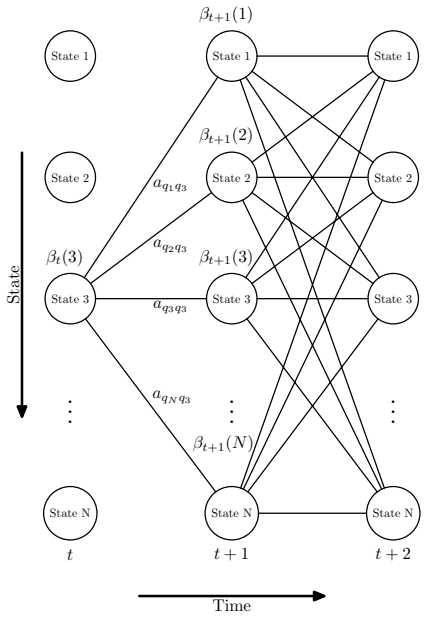


Figure 5.2: Backward Procedure - Induction Step.

The backward algorithm includes the following steps:

1. **Initialization**

$$\begin{aligned} \text{Set } t &= T-1; \\ \beta_T(i) &= 1, \quad 1 \le i \le N \end{aligned} \tag{5.11}$$

2. **Induction**

$$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(i) a_{ij} b_j(\mathbf{o}_{t+1}), \quad 1 \le i \le N \tag{5.12}$$

3. **Update time**

   Set $t = t - 1$;
   Return to step 2 if $t > 0$;
   Otherwise, terminate the algorithm.

Note that the initialization step 1 *arbitrarily* defines $\beta_T(i)$ to be 1 for all i.

5.3 Scaling the Forward and Backward Variables

The calculation of $\alpha_t(i)$ and $\beta_t(i)$ involves multiplication with probabilities. All these probabilities have a value less than one (generally significantly less than one), and as t starts to grow large, each term of $\alpha_t(i)$ or $\beta_t(i)$ starts to head exponentially to zero. For a sufficiently large t (e.g. 100 or more) the dynamic range of $\alpha_t(i)$ and $\beta_t(i)$ will exceed the precision range of essentially any machine (even in double precision) [2]. The basic scaling procedure multiplies $\alpha_t(i)$ by a scaling coefficient that is dependent only on the time t and independent of the state i. The scaling factor for the forward variable is denoted c_t (scaling is done every time t for all states i - $1 \leq i \leq N$) . This factor will also be used to scale the backward variable, $\beta_t(i)$. It is useful to scale $\alpha_t(i)$ and $\beta_t(i)$ with the same factor, see problem 3 (parameter estimation).

5.3.1 The Scaled Forward Algorithm

Consider the computation of the forward variable, $\alpha_t(i)$. In the scaled variant of the forward algorithm some extra notations will be used. $\alpha_t(i)$ denotes the unscaled forward variable, $\hat{\alpha}_t(i)$ denotes the scaled and iterated variant of $\alpha_t(i)$, $\widehat{\hat{\alpha}}_t(i)$ denotes the local version of $\alpha_t(i)$ before scaling and c_t will represent the scaling coefficient at each time. The scaled forward algorithm includes the following steps:

1. **Initialization**

$$\begin{aligned}
\text{Set } t &= 2; \\
\alpha_1(i) &= \pi_i b_i(\mathbf{o}_1), \quad 1 \leq i \leq N && (5.13) \\
\widehat{\hat{\alpha}}_1(i) &= \alpha_1(i), \quad 1 \leq i \leq N && (5.14) \\
c_1 &= \frac{1}{\sum_{i=1}^{N} \alpha_1(i)} && (5.15) \\
\hat{\alpha}_1(i) &= c_1 \alpha_1(i) && (5.16)
\end{aligned}$$

2. **Induction**

$$\widehat{\hat{\alpha}}_t(i) = b_i(\mathbf{o}_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{ji}, \quad 1 \leq i \leq N \tag{5.17}$$

$$c_t = \frac{1}{\sum_{i=1}^{N}\widehat{\hat{\alpha}}_t(i)} \tag{5.18}$$

$$\hat{\alpha}_t(i) = c_t\widehat{\hat{\alpha}}_t(i), \quad 1 \leq i \leq N \tag{5.19}$$

3. **Update time**

Set $t = t + 1$;
Return to step 2 if $t \leq T$;
Otherwise, terminate the algorithm (goto step 4).

4. **Termination**

$$\log P(\mathbf{O}|\lambda) = -\sum_{t=1}^{T}\log c_t \tag{5.20}$$

The main difference between the scaled and the default forward algorithm lies in steps 2 and 4. In step 2 can Eq. (5.19) be rewritten if Eq. (5.17) and Eq. (5.18) are used

$$\begin{aligned}
\hat{\alpha}_t(i) &= c_t\widehat{\hat{\alpha}}_t(i) \\
&= \frac{1}{\sum_{k=1}^{N}\left(b_k(\mathbf{o}_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{jk}\right)}\cdot\left(b_i(\mathbf{o}_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{ji}\right) \\
&= \frac{b_i(\mathbf{o}_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{ji}}{\sum_{k=1}^{N}b_k(\mathbf{o}_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{jk}}, \quad 1 \leq i \leq N
\end{aligned} \tag{5.21}$$

By induction, the scaled forward variable is found in terms of the none scaled

$$\hat{\alpha}_{t-1}(j) = \left(\prod_{\tau=1}^{t-1}c_\tau\right)\alpha_{t-1}(j), \quad 1 \leq j \leq N \tag{5.22}$$

The ordinary induction step can be found as (same as Eq. (5.8) but with one time unit shift)

$$\alpha_t(i) = b_i(\mathbf{o}_t) \sum_{j=1}^{N} \alpha_{t-1}(j) a_{ji}, \quad 1 \leq i \leq N \tag{5.23}$$

With Eq. (5.22) and Eq. (5.23) in mind, it is now possible to rewrite Eq. (5.21) as

$$\begin{aligned}
\hat{\alpha}_t(i) &= \frac{b_i(\mathbf{o}_t) \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ji}}{\sum_{k=1}^{N} b_k(\mathbf{o}_t) \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{jk}} \\
&= \frac{b_i(\mathbf{o}_t) \sum_{j=1}^{N} \left(\prod_{\tau=1}^{t-1} c_\tau \right) \alpha_{t-1}(j) a_{ji}}{\sum_{k=1}^{N} b_k(\mathbf{o}_t) \sum_{j=1}^{N} \left(\prod_{\tau=1}^{t-1} c_\tau \right) \alpha_{t-1}(j) a_{jk}} \\
&= \frac{\left(\prod_{\tau=1}^{t-1} c_\tau \right) \left(b_i(\mathbf{o}_t) \sum_{j=1}^{N} \alpha_{t-1}(j) a_{ji} \right)}{\left(\prod_{\tau=1}^{t-1} c_\tau \right) \sum_{k=1}^{N} \left(b_k(\mathbf{o}_t) \sum_{j=1}^{N} \alpha_{t-1}(j) a_{jk} \right)} \\
&= \frac{\alpha_t(i)}{\sum_{k=1}^{N} \alpha_t(k)}, \quad 1 \leq i \leq N
\end{aligned} \tag{5.24}$$

As Eq. (5.24) shows, when the scaled forward algorithm is applied each $\alpha_t(i)$ is scaled by the sum of all states of $\alpha_t(i)$.

The termination (step 4) of the scaled forward algorithm requires the evaluation of $P(\mathbf{O}|\lambda)$. This must be done in an indirect way. Since the sum of $\hat{\alpha}_T(i)$ can not be used. This is due to the fact that $\hat{\alpha}_T(i)$ already is scaled. However, this can be solved by employing

$$\prod_{\tau=1}^{T} c_\tau \sum_{i=1}^{N} \alpha_T(i) = 1 \tag{5.25}$$

$$\prod_{\tau=1}^{T} c_\tau \cdot P(\mathbf{O}|\lambda) = 1 \tag{5.26}$$

$$P(\mathbf{O}|\lambda) = \frac{1}{\prod_{\tau=1}^{T} c_\tau} \tag{5.27}$$

As Eq. (5.27) shows $P(\mathbf{O}|\lambda)$ can be found, but if Eq. (5.27) is used the result will still be very small (and probable be out of the machine precision for a computer). However, the logarithm on both sides of Eq. (5.27) gives the following equation

$$\log P(\mathbf{O}|\lambda) = \log \frac{1}{\prod_{\tau=1}^{T} c_\tau} = -\sum_{t=1}^{T} \log c_t \tag{5.28}$$

This expression is used in the termination step of the scaled forward algorithm. The logarithm of $P(\mathbf{O}|\lambda)$ is often just as useful as $P(\mathbf{O}|\lambda)$ as, in most cases, this measure is used as a comparison with other probabilities (for other models).

5.3.2 The Scaled Backward Algorithm

The scaled backward algorithm can be found more easily, since it will use the same scale factor as the forward algorithm. The notation used is similar to the forward variable notation, i.e. $\beta_t(i)$ denotes the unscaled backward variable, $\hat{\beta}_t(i)$ denotes the scaled and iterated variant of $\beta_t(i)$, $\widehat{\hat{\beta}}_t(i)$ denotes the local version of $\beta_t(i)$ before scaling and c_t will represent the scaling coefficient at each time. The scaled backward algorithm includes the following steps:

1. **Initialization**

$$\text{Set } t = T - 1;$$

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{5.29}$$

$$\hat{\beta}_T(i) = c_T \beta_T(i), \quad 1 \leq i \leq N \tag{5.30}$$

2. **Induction**

$$\widehat{\hat{\beta}}_t(i) = \sum_{j=1}^{N} \hat{\beta}_{t+1}(j) a_{ij} b_j(\mathbf{o}_{t+1}), \quad 1 \leq i \leq N \tag{5.31}$$

$$\hat{\beta}_t(i) = c_t \widehat{\hat{\beta}}_t(i), \quad 1 \leq i \leq N \tag{5.32}$$

3. **Update time**

   Set $t = t - 1$;
   Return to step 2 if $t > 0$;
   Otherwise, terminate the algorithm.

Chapter 6

Solution to Problem 2 - "Optimal" State Sequence

The central problem is to find the optimal sequence of states to a given observation sequence and model used. Unlike Problem 1, an exact solution can not be found and there are several possible ways of solving this problem. The difficulty is the definition of the optimal state sequence, because there are several possible optimality criteria [2]. One optimal criterion is to choose the states, q_t, that are *individually* most likely at each time t. To find the state sequence a probability variable is needed

$$\gamma_t(i) = P\left(q_t = i|\mathbf{O}, \lambda\right) \tag{6.1}$$

This is the probability of being in state i at time t given the observation sequence $\mathbf{O}$ and the model λ. Another interpretation of $\gamma_t(i)$ is

$$\begin{aligned} \gamma_t(i) &= P\left(q_t = i|\mathbf{O}, \lambda\right) \\ &= \frac{P\left(\mathbf{O}, q_t = i|\lambda\right)}{P\left(\mathbf{O}|\lambda\right)} \\ &= \frac{P\left(\mathbf{O}, q_t = i|\lambda\right)}{\sum_{i=1}^{N} P\left(\mathbf{O}, q_t = i|\lambda\right)} \end{aligned} \tag{6.2}$$

and since $\alpha_t(i) = P\left(\mathbf{o}_1\mathbf{o}_2 \ldots \mathbf{o}_t, q_t = i|\lambda\right)$ and $\beta_t(i) = P\left(\mathbf{o}_{t+1}\mathbf{o}_{t+2} \ldots \mathbf{o}_T|q_t = i, \lambda\right)$, $P\left(\mathbf{O}, q_t = i|\lambda\right)$ can be found as the joint probability

$$P\left(\mathbf{O}, q_t = i|\lambda\right) = P\left(\mathbf{o}_1\mathbf{o}_2 \ldots \mathbf{o}_t, q_t = i|\lambda\right) \cdot P\left(\mathbf{o}_{t+1}\mathbf{o}_{t+2} \ldots \mathbf{o}_T|q_t = i, \lambda\right) = \alpha_t(i)\beta_t(i) \tag{6.3}$$

By Eq. (6.3) it is now possible to rewrite Eq. (6.2) as

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{6.4}$$

When $\gamma_t(i)$ is calculated using Eq. (6.4), the most likely state at time t, q_t^*, can be found as

$$q_t^* = \arg \max_{1 \le i \le N} [\gamma_t(i)], \quad 1 \le t \le T \tag{6.5}$$

Eq. (6.5) maximizes the expected number of correct states, however there could be some problems with the resulting state sequence, since the state transition probabilities have not been taken into consideration. For example, what happens when some state transitions have zero probability ($a_{ij} = 0$)? This means that the given optimal path may not be valid. Obviously, a method generating a guaranteed valid path is preferred. Fortunately such a method exists, based on dynamic programming, namely *the Viterbi algorithm*. The $\gamma_t(i)$ could not be used for this purpose, but it will still be useful in problem 3 (parameter estimation).

6.1 The Viterbi Algorithm

The Viterbi algorithm is similar to the forward algorithm. The main difference is that the forward algorithm uses the sum over previous states, whereas the Viterbi algorithm uses maximization. The aim for the Viterbi algorithm is to find the single best state sequence, $\mathbf{q} = (q_1, q_2, \ldots, q_T)$, for the given observation sequence $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T)$ and model λ. Consider

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P\left(q_1 q_2 \ldots q_{t-1}, q_t = i, \mathbf{o}_1 \mathbf{o}_2 \ldots \mathbf{o}_t | \lambda\right) \tag{6.6}$$

which is the probability of observing $\mathbf{o}_1 \mathbf{o}_2 \ldots \mathbf{o}_t$ using the best path that ends in state i at time t, given the model λ. By using induction, $\delta_{t+1}(i)$ can be found as

$$\delta_{t+1}(i) = b_j(\mathbf{o}_{t+1}) \max_{1 \le i \le N} [\delta_t(i) a_{ij}] \tag{6.7}$$

To retrieve the state sequence, it is necessary to keep track of the argument that maximizes Eq. (6.7), for each t and j . This is done by saving the argument in an array $\psi_t(j)$. Here follows the complete Viterbi algorithm:

1. **Initialization**

$$\begin{aligned} \text{Set } t &= 2; \\ \delta_1(i) &= \pi_i b_i(\mathbf{o}_1), \quad 1 \le i \le N \qquad (6.8) \\ \psi_1(i) &= 0, \quad 1 \le i \le N \qquad (6.9) \end{aligned}$$

2. **Induction**

$$\delta_t(j) = b_j(\mathbf{o}_t) \max_{1 \le i \le N} \delta_{t-1}(i) a_{ij}, \quad 1 \le j \le N \tag{6.10}$$

$$\psi_t(j) = \arg \max_{1 \le i \le N} \left[\delta_{t-1}(i) a_{ij}\right], \quad 1 \le j \le N \tag{6.11}$$

3. **Update time**

   Set $t = t + 1$;
   Return to step 2 if $t \le T$;
   Otherwise, terminate the algorithm (goto step 4).

4. **Termination**

$$P\left(\mathbf{O}, \mathbf{q}^* | \lambda\right) = \max_{1 \le i \le N} \left[\delta_T(i)\right] \tag{6.12}$$

$$q_T^* = \arg \max_{1 \le i \le N} \left[\delta_T(i)\right] \tag{6.13}$$

5. **Path (state sequence) backtracking**

   (a) **Initialization**

   Set $t = T - 1$

   (b) **Backtracking**

$$q_t^* = \psi_{t+1}\left(q_{t+1}^*\right) \tag{6.14}$$

   (c) **Update time**

   Set $t = t - 1$;
   Return to step (b) if $t \ge 1$;
   Otherwise, terminate the algorithm.

The algorithm involves multiplication with probabilities and this is the same problem as for the forward and backward algorithm. Therefore, a numerical stable algorithm is required.

6.2 The Alternative Viterbi Algorithm

As mentioned the original Viterbi algorithm involves multiplications with probabilities. One way to avoid this is to take the logarithm of the model parameters. Obviously this logarithm will end up with problems when some model parameters are zero. This is often the case for $\mathbf{A}$ and $\boldsymbol{\pi}$ but can be avoided by checking for zeros in $\mathbf{A}$ and $\boldsymbol{\pi}$ and replacing the output for the logarithm to the maximum negative number due to computer precision $-\delta_{max}$. The problem also arises when $b_i(\mathbf{o}_t)$ becomes zero. Hence, if $b_i(\mathbf{o}_t)$ becomes zero[1] it will be replaced by δ_{min}. Here follows the alternative Viterbi algorithm

[1] Read more about numerical problems in Sec. 7.1.2.

1. **Preprocessing**

$$\widetilde{\pi}_i = \begin{cases} \log(\pi_i) & \text{if } \pi_i > \delta_{min} \\ -\delta_{max}, & \text{otherwise} \end{cases}, \quad 1 \le i \le N \tag{6.15}$$

$$\widetilde{a}_{ij} = \begin{cases} \log(a_{ij}) & \text{if } a_{ij} > \delta_{min} \\ -\delta_{max}, & \text{otherwise} \end{cases}, \quad 1 \le i,j \le N \tag{6.16}$$

2. **Initialization**

$$\begin{aligned} \text{Set } t &= 2; \\ \widetilde{b}_i(\mathbf{o}_1) &= \log(b_i(\mathbf{o}_1)), \quad 1 \le i \le N && (6.17) \\ \widetilde{\delta}_1(i) &= \log(\delta_1(i)) \\ &= \log(\pi_i b_i(\mathbf{o}_1)) \\ &= \widetilde{\pi}_i + \widetilde{b}_i(\mathbf{o}_1), \quad 1 \le i \le N && (6.18) \\ \psi_1(i) &= 0, \quad 1 \le i \le N && (6.19) \end{aligned}$$

3. **Induction**

$$\begin{aligned} \widetilde{b}_j(\mathbf{o}_t) &= \log(b_j(\mathbf{o}_t)), \quad 1 \le j \le N && (6.20) \\ \widetilde{\delta}_t(j) &= \log(\delta_t(j)) \\ &= \log\left(b_j(\mathbf{o}_t) \max_{1\le i\le N} \delta_{t-1}(i) a_{ij}\right) \\ &= \widetilde{b}_j(\mathbf{o}_t) + \max_{1\le i\le N}\left[\widetilde{\delta}_{t-1}(i) + \widetilde{a}_{ij}\right], \quad 1 \le j \le N && (6.21) \\ \psi_t(j) &= \arg\max_{1\le i\le N}\left[\widetilde{\delta}_{t-1}(i) + \widetilde{a}_{ij}\right], \quad 1 \le j \le N && (6.22) \end{aligned}$$

4. **Update time**

Set $t = t + 1$;
Return to step 3 if $t \le T$;
Otherwise, terminate the algorithm (goto step 5).

5. **Termination**

$$\begin{aligned} \widetilde{P}(\mathbf{O}, \mathbf{q}^*|\lambda) &= \max_{1\le i\le N}\left[\widetilde{\delta}_T(i)\right] && (6.23) \\ q_T^* &= \arg\max_{1\le i\le N}\left[\widetilde{\delta}_T(i)\right] && (6.24) \end{aligned}$$

6. **Path (state sequence) backtracking**

   (a) **Initialization**

   Set $t = T - 1$

   (b) **Backtracking**

$$q_t^* = \psi_{t+1}\left(q_{t+1}^*\right) \tag{6.25}$$

   (c) **Update time**

   Set $t = t - 1$;
   Return to step (b) if $t \geq 1$;
   Otherwise, terminate the algorithm.

To get a better understanding of how the Viterbi (and the alternative Viterbi) work, consider a model with $N = 3$ states and an observation of length $T = 8$. In the initialization ($t = 1$) $\delta_1(1)$, $\delta_1(2)$ and $\delta_1(3)$ are found. Assume that $\delta_1(2)$ is the largest. Next time ($t = 2$) three variables will be used, namely $\delta_2(1)$, $\delta_2(2)$ and $\delta_2(3)$. Let us assume that $\delta_2(1)$ is now the maximum of the three. In the same manner the variables $\delta_3(3)$, $\delta_4(2)$, $\delta_5(2)$, $\delta_6(1)$, $\delta_7(3)$ and $\delta_8(3)$ will be the maximum at their time, see Fig. 6.1. From Fig. 6.1 one can see that the Viterbi algorithm is working with the lattice structure.

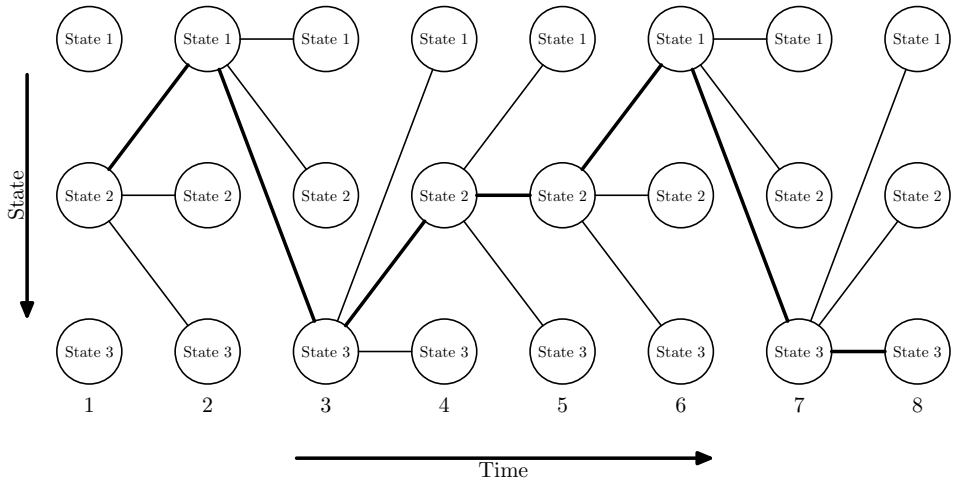


Figure 6.1: Example of Viterbi search.

An algorithm only finding the probability, $\widetilde{P}\left(\mathbf{O}, \mathbf{q}^* | \lambda\right)$, for the "optimal" state sequence and not extracting the actual state sequence can be made faster and is called *the fast alternative Viterbi Algorithm*.

6.3 The Fast Alternative Viterbi Algorithm

The alternative Viterbi finds the "optimal" state sequence, q_t^*, and the corresponding probability $\widetilde{P}(\mathbf{O}, \mathbf{q}^*|\lambda)$. In some cases the actual state sequence is not so important and only the probability for the state sequence is desired. For example, when a competition between two (or more) models is desired, the highest probability is only desired. In this case the backtracking and $\psi_t(j)$ can be removed from the algorithm, this implies that a faster and less memory required algorithm can be applied. The algorithm is called *the fast alternative Viterbi algorithm* (a similar algorithm can be constructed based on the standard Viterbi algorithm). Here follows the fast alternative Viterbi algorithm:

1. **Preprocessing**

$$\widetilde{\pi}_i = \begin{cases} \log(\pi_i) & \text{if } \pi_i > \delta_{min} \\ -\delta_{max}, & \text{otherwise} \end{cases}, \quad 1 \le i \le N \tag{6.26}$$

$$\widetilde{a}_{ij} = \begin{cases} \log(a_{ij}) & \text{if } a_{ij} > \delta_{min} \\ -\delta_{max}, & \text{otherwise} \end{cases}, \quad 1 \le i, j \le N \tag{6.27}$$

2. **Initialization**

$$\begin{aligned} \text{Set } t &= 2; \\ \widetilde{b}_i(\mathbf{o}_1) &= \log(b_i(\mathbf{o}_1)), \quad 1 \le i \le N \qquad (6.28) \\ \widetilde{\delta}_1(i) &= \log(\delta_1(i)) \\ &= \log(\pi_i b_i(\mathbf{o}_1)) \\ &= \widetilde{\pi}_i + \widetilde{b}_i(\mathbf{o}_1), \quad 1 \le i \le N \qquad (6.29) \\ & \qquad (6.30) \end{aligned}$$

3. **Induction**

$$\begin{aligned} \widetilde{b}_j(\mathbf{o}_t) &= \log(b_j(\mathbf{o}_t)), \quad 1 \le j \le N \qquad (6.31) \\ \widetilde{\delta}_t(j) &= \log(\delta_t(j)) \\ &= \log\left(b_j(\mathbf{o}_t) \max_{1 \le i \le N} \delta_{t-1}(i) a_{ij}\right) \\ &= \widetilde{b}_j(\mathbf{o}_t) + \max_{1 \le i \le N} \left[\widetilde{\delta}_{t-1}(i) + \widetilde{a}_{ij}\right], \quad 1 \le j \le N \qquad (6.32) \end{aligned}$$

4. **Update time**

Set $t = t + 1$;
Return to step 3 if $t \leq T$;
Otherwise, terminate the algorithm (goto step 5).

5. **Termination**

$$\widetilde{P}\left(\mathbf{O}, \mathbf{q}^* | \lambda\right) = \max_{1 \leq i \leq N} \left[\widetilde{\delta}_T(i)\right] \tag{6.33}$$

Chapter 7

Solution to Problem 3 - Parameter Estimation

The third problem is concerned with the estimation of the model parameters, $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$. The problem can be formulated as

$$\lambda^* = \arg\max_{\lambda} \left[P\left(\mathbf{O}|\lambda\right) \right] \tag{7.1}$$

that is, given an observation $\mathbf{O}$, the aim is to find the model λ^* from all possible λ that maximizes $P\left(\mathbf{O}|\lambda\right)$. This problem is the most difficult of the three problems, as there is no known way to analytically find the model parameters that maximize the probability of the observation sequence in a closed form. However, the model parameters can be chosen to locally maximize the likelihood $P\left(\mathbf{O}|\lambda\right)$. Some commonly used methods for solving this problem is the Baum-Welch method (which has essentially the same solution as the expectation-maximation method in this task) or gradient techniques. Both methods use iteration technique to improve the likelihood $P\left(\mathbf{O}|\lambda\right)$. However, there are some advantages with the Baum-Welch method compared to the gradient techniques [12, 13, 14]:

- Baum-Welch is numerically stable with an increasing likelihood in every iteration.
- Baum-Welch converges to a local optima.
- Baum-Welch has linear convergence.

This section will derive the reestimation equations used in the Baum-Welch method.

7.1 Baum-Welch Reestimation

The model λ has three terms, the state transition probability distribution $\mathbf{A}$, the initial state distribution $\boldsymbol{\pi}$ and the observation symbol probability distribution $\mathbf{B}$. Since continuous observation densities are used here, $\mathbf{B}$ will be represented by[1] c_{jk},$\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\Sigma}_{jk}$. To describe the procedure for reestimation, the probability $\xi_t(i,j)$ will be useful

$$\xi_t(i,j) = P\left(q_t = i, q_{t+1} = j|\mathbf{O},\lambda\right) = \frac{P\left(q_t = i, q_{t+1} = j, \mathbf{O}|\lambda\right)}{P\left(\mathbf{O}|\lambda\right)} \tag{7.2}$$

which gives the probability of being in state i at time t, and state j at time $t+1$, given the model λ and the observation sequence $\mathbf{O}$. The paths that satisfy the conditions required by Eq. (7.2) are illustrated in Fig. 7.1.

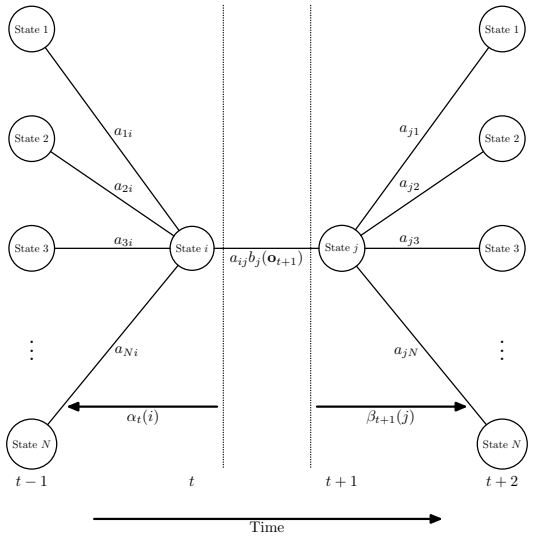


Figure 7.1: Computation of $\xi_t(i,j)$.

Considering Eqs. (2.2), (3.1), (5.5), (5.10), (6.3) and (7.2) yields that $\xi_t(i,j)$ can be found as

$$\begin{aligned}
\xi_t(i,j) &= \frac{\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)}{P\left(\mathbf{O}|\lambda\right)} \\
&= \frac{\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)}
\end{aligned} \tag{7.3}$$

[1] See Eqs. 3.3-3.6.

As mentioned in problem 2 it is the probability $\gamma_t(i)$ of being in state i at time t, given the entire observation sequence $\mathbf{O}$ and the model λ. Hence the relation between $\gamma_t(i)$ and $\xi_t(i,j)$ can be found by using Eqs. (6.1) and (7.2)

$$\gamma_t(i) = P\left(q_t = i|\mathbf{O},\lambda\right) = \sum_{j=1}^{N} P\left(q_t = i, q_{t+1} = j|\mathbf{O},\lambda\right) = \sum_{j=1}^{N} \xi_t(i,j) \tag{7.4}$$

If the sum over time t is applied to $\gamma_t(i)$, one obtains a quantity that can be interpreted as the expected number of times that state i is visited, or equivalently, the expected number of transitions made from state i (if the time slot $t = T$ is excluded) [2]. If the summation is done over $\xi_t(i,j)$ it will result in the expected number of transitions from state i to state j. The term $\gamma_1(i)$ will also prove to be useful. It can thus be concluded that

$$\gamma_1(i) = \text{probability of starting in state } i \tag{7.5}$$

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{expected number of transitions from state } i \text{ in } \mathbf{O} \tag{7.6}$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) = \text{expected number of transitions from state } i \text{ to state } j \text{ in } \mathbf{O} \tag{7.7}$$

Given the above definitions it is possible to derive the reestimation formulas for $\boldsymbol{\pi}$ and $\mathbf{A}$

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

and

$$\begin{aligned} a_{ij} &= \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} \\ &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\ &= \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)} \end{aligned}$$

In the derivation of Eqs. (7.8) and (7.8), the following probabilities are useful to get a further insight

$$\begin{aligned} P(\mathbf{O}|\lambda) &= \sum_{i=1}^{N} \alpha_t(i)\beta_t(i) = \sum_{i=1}^{N} \alpha_T(i) \\ P(\mathbf{O}, q_t = i|\lambda) &= \alpha_t(i)\beta_t(i) \\ P(\mathbf{O}, q_t = i, q_{t+1} = j|\lambda) &= \alpha_t(i) a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j) \end{aligned} \tag{7.8}$$

The Eqs. (7.3) and (6.4), and thereby Eqs. (7.8) and (7.8) can be found by using the unscaled forward and backward variables. If scaled variables are used the probability $\xi_t(i,j)$ will be found as

$$
\begin{aligned}
\xi_t(i,j) &= \frac{\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)} \\
&= \frac{\left(\prod_{\tau=1}^{t}c_\tau\right)\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\left(\prod_{\tau=t+1}^{T}c_\tau\right)\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\left(\prod_{\tau=1}^{t}c_\tau\right)\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\left(\prod_{\tau=t+1}^{T}c_\tau\right)\beta_{t+1}(j)} \\
&= \frac{\left(\prod_{\tau=1}^{t}c_\tau\right)\left(\prod_{\tau=t+1}^{T}c_\tau\right)\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)}{\left(\prod_{\tau=1}^{t}c_\tau\right)\left(\prod_{\tau=t+1}^{T}c_\tau\right)\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)} \\
&= \frac{\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)}
\end{aligned}
\tag{7.9}
$$

and the scaled $\gamma_t(i)$ as

$$
\begin{aligned}
\gamma_t(i) &= \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{i=1}^{N}\hat{\alpha}_t(i)\hat{\beta}_t(i)} \\
&= \frac{\left(\prod_{\tau=1}^{t} c_\tau\right)\alpha_t(i)\left(\prod_{\tau=t}^{T} c_\tau\right)\beta_t(i)}{\sum_{i=1}^{N}\left(\prod_{\tau=1}^{t} c_\tau\right)\alpha_t(i)\left(\prod_{\tau=t}^{T} c_\tau\right)\beta_t(i)} \\
&= \frac{\left(\prod_{\tau=1}^{t} c_\tau\right)\left(\prod_{\tau=t}^{T} c_\tau\right)\alpha_t(i)\beta_t(i)}{\left(\prod_{\tau=1}^{t} c_\tau\right)\left(\prod_{\tau=t}^{T} c_\tau\right)\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)} \\
&= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)}
\end{aligned}
\tag{7.10}
$$

As Eqs. (7.9) and (7.10) show, $\xi_t(i,j)$ and $\gamma_t(i)$ are the same if scaled or none scaled forward and backward variables are used. Herein lies the elegance with using a scale factor that is dependent of time, and independent of the state. A closer look at Eq. (7.9) shows that $\xi_t(i,j)$ can be found easier when the scaled forward and backward variables are used

$$
\begin{aligned}
\xi_t(i,j) &= \frac{\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)} \\
&= \frac{\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\hat{\alpha}_t(i)\left(\sum_{j=1}^{N}a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)\right)} \\
&= \frac{\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\hat{\alpha}_t(i)\widehat{\hat{\beta}}_t(i)} \\
&= \frac{\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\hat{\alpha}_t(i)\frac{\hat{\beta}_t(i)}{c_t}} \\
&= \frac{\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\left(\prod_{\tau=1}^{t}c_\tau\right)\alpha_t(i)\beta_t(i)} \\
&= \frac{\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)}{\left(\prod_{\tau=1}^{t}c_\tau\right)\sum_{i=1}^{N}P\left(\mathbf{O},q_t=i|\lambda\right)} \\
&= \hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)
\end{aligned}
\tag{7.11}
$$

Since $\boldsymbol{\pi}$ and $\mathbf{A}$ use $\xi_t(i,j)$ and $\gamma_t(i)$ for calculation, these probabilities will also be independent of which forward or backward variables are used (scaled or unscaled). Thereby Eqs. (7.8) and (7.8) can be calculated, without numerical problems, as

$$
\pi_i = \frac{\hat{\alpha}_1(i)\hat{\beta}_1(i)}{\sum_{i=1}^{N}\hat{\alpha}_T(i)} = \gamma_1(i) \tag{7.12}
$$

and

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \hat{\alpha}_t(i) a_{ij} b_j(\mathbf{o}_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{t=1}^{T-1} \hat{\alpha}_t(i) \hat{\beta}_t(i)} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{7.13}$$

The reestimation of c_{jk}, $\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\Sigma}_{jk}$ is more complicated. However, if the model only has one state N and one mixture M, it is an easy averaging task[2]

$$c_j = 1 \tag{7.14}$$

$$\boldsymbol{\mu}_j = \frac{1}{T} \sum_{t=1}^{T} \mathbf{o}_t \tag{7.15}$$

$$\boldsymbol{\Sigma}_j = \frac{1}{T} \sum_{t=1}^{T} \left(\mathbf{o}_t - \boldsymbol{\mu}_j\right) \left(\mathbf{o}_t - \boldsymbol{\mu}_j\right)' \tag{7.16}$$

In practice, there could be multiple states and multiple mixtures and there are no direct assignments of the observation vectors to individual states as the underlying state sequence is unknown. Since the full likelihood of each observation sequence is based on the summation of all possible state sequences, each observation vector $\mathbf{o}_t$ contributes to the computation of the likelihood for each state j. Instead of assigning each observation vector to a specific state, each observation is assigned to every state and weighted. This weight is the probability of the model being in a state and a specific mixture for this particulary observation vector. The probability, for state j and mixture k, is found by[3]

$$\gamma_t(j,k) = \frac{\alpha_t(j)\beta_t(j)}{\sum_{i=1}^{N} \alpha_t(j)\beta_t(j)} \cdot \frac{c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right)}{\sum_{k=1}^{M} c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right)} \tag{7.17}$$

The reestimation formula for c_{jk} is the ratio between the expected number of times the system is in state j using the kth mixture component, and the expected number of times the system is in state j. That is

$$c_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T} \sum_{k=1}^{M} \gamma_t(j,k)} \tag{7.18}$$

[2] as transpose is $'$ used , this because no confusion with observation time T should be made.
[3] $\gamma_t(j,k)$ is a generalization of $\gamma_t(j)$.

Calculating $\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\Sigma}_{jk}$ can be achieved by weighting the simple averages in Eqs. (7.15) and (7.16). As before, the weighting is the probability of being in state j and using mixture k when observing $\mathbf{o}_t$. The reestimation formulas for $\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\Sigma}_{jk}$ will hereby be found as

$$\boldsymbol{\mu}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)\mathbf{o}_t}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{7.19}$$

$$\boldsymbol{\Sigma}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)'}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{7.20}$$

The reestimation described in this section is based on one training sample, which is typically not sufficient to get reliable estimates. This is particulary the case when left-right models are used and multiple parameters should be found. To get reliable estimates it is convenient to use multiple observation sequences.

7.1.1 Multiple Observation Sequences

In the Baum-Welch method a large number of parameters are estimated and a large number of training examples are needed to get robust and reliable estimates. If only one observation sequence is used to train the model it will be overtrained to this specific sample, which can lead to numerical problems for the reestimation procedure.

The modification of the reestimation procedure is straightforward as follows. Let us denote a set of R observation sequences as

$$\mathbf{O} = \left[\mathbf{O}^{(1)}, \mathbf{O}^{(2)} \ \ldots, \mathbf{O}^{(R)}\right] \tag{7.21}$$

where $\mathbf{O}^{(r)} = \left(\mathbf{o}_1^{(r)}, \mathbf{o}_2^{(r)}, \ldots, \mathbf{o}_{T_r}^{(r)}\right)$ is the rth observation sequence and T_r is the corresponding observation sequence length. The new function to maximize $P\left(\mathbf{O}|\lambda\right)$, according to Eq. (7.1), can now be found as (if the observations sequences, $\mathbf{O}^{(r)}$, are assumed independent)

$$P\left(\mathbf{O}|\lambda\right) = \prod_{r=1}^{R} P\left(\mathbf{O}^{(r)}|\lambda\right) \tag{7.22}$$

A more complete derivation of the reestimation formulas, without the independence assumption can be found in appendix A.

The method for achieving this maximation problem is similar to the procedure with only one observation of sequences. The variables $\alpha_t(j)$, $\beta_t(j)$, $\gamma_t(j,k)$ and $\xi_t(i,j)$ are found for every training sample. Then the reestimation is performed on the accumulated values. In this way the new parameters are an average of the result from each individual training sequence. The procedure is repeated until a optima is found or the maximum limit of iterations is reached. The reestimation formulas will now be

$$\pi_i = \frac{\sum_{r=1}^{R} \hat{\alpha}_1^{(r)}(i)\hat{\beta}_1^{(r)}(i)}{\sum_{r=1}^{R}\sum_{i=1}^{N} \hat{\alpha}_{T_r}^{(r)}(i)} = \frac{1}{R}\sum_{r=1}^{R} \gamma_1^{(r)}(i) \tag{7.23}$$

$$a_{ij} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r-1} \hat{\alpha}_t^{(r)}(i)a_{ij}b_j(\mathbf{o}_{t+1}^{(r)})\hat{\beta}_{t+1}^{(r)}(j)}{\sum_{r=1}^{R}\sum_{t=1}^{T_r-1} \hat{\alpha}_t^{(r)}(i)\hat{\beta}_t^{(r)}(i)} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r-1} \xi_t^{(r)}(i,j)}{\sum_{r=1}^{R}\sum_{t=1}^{T_r-1} \gamma_t^{(r)}(i)} \tag{7.24}$$

$$c_{jk} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r} \gamma_t^{(r)}(j,k)}{\sum_{r=1}^{R}\sum_{t=1}^{T_r}\sum_{k=1}^{M} \gamma_t^{(r)}(j,k)} \tag{7.25}$$

$$\boldsymbol{\mu}_{jk} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r} \gamma_t^{(r)}(j,k)\mathbf{o}_t^{(r)}}{\sum_{r=1}^{R}\sum_{t=1}^{T_r} \gamma_t^{(r)}(j,k)} \tag{7.26}$$

$$\boldsymbol{\Sigma}_{jk} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r} \gamma_t^{(r)}(j,k)\left(\mathbf{o}_t^{(r)} - \boldsymbol{\mu}_{jk}\right)\left(\mathbf{o}_t^{(r)} - \boldsymbol{\mu}_{jk}\right)'}{\sum_{r=1}^{R}\sum_{t=1}^{T_r} \gamma_t^{(r)}(j,k)} \tag{7.27}$$

Note that the result from Eq. (7.26) is used in Eq. (7.27). These are the final formulas used for the reestimation.

7.1.2 Numerical Issues

One problem with training HMM parameters with the reestimation formulas is that the number of training observation may not be sufficient. There is always an inadequate number of occurrences of low probability events to give food estimates of the model parameters. This has the effect that the mixture weights and the diagonal values in the covariance matrix can be zero, which is numerically undesired. To avoid this a numerical floor ϵ_c is introduced

$$c_{jk} = \begin{cases} c_{jk}, & \text{if } c_{jk} \geq \epsilon_c \\ \epsilon_c & \text{otherwise} \end{cases} \tag{7.28}$$

and a similar floor ϵ_Σ is introduced for the diagonal values in the covariance matrix

$$\mathbf{\Sigma}_{jk}(d,d) = \begin{cases} \mathbf{\Sigma}_{jk}(d,d), & \text{if } \mathbf{\Sigma}_{jk}(d,d) \geq \epsilon_\Sigma \\ \epsilon_\Sigma, & \text{otherwise} \end{cases}, \quad 1 \leq d \leq D \tag{7.29}$$

The numerical floor values are typical (double precision considered) in the range of $10^{-10} \leq \epsilon_c, \epsilon_\Sigma \leq 10^{-3}$ [2]. The extreme value for ϵ_Σ can be found by $\epsilon_\Sigma > \delta_{min}^{(1/D)}$ (if diagonal covariances are used), i.e. if all diagonal elements in the covariance are zero one will multiply ϵ_Σ with itself D times. This is the determinant if diagonal covariances are used. Note that the final result can not be smaller than δ_{min} (underflow). This also indicates that a large dimension of the feature vectors will require a higher numerical precision.

Another problem is that the Gaussian mixture $b_j(\mathbf{o}_t)$ can be zero, which is not desired as there is always a small probability of an event (in the Gaussian case). This is avoided by a probability floor which is the smallest numerical value possible, δ_{min}, due to the machine precision (in double precision is $\delta_{min} = 2^{-1022} \approx 2.2251 \cdot 10^{-308}$). Hence, the Gaussian mixture is found by

$$b_j(\mathbf{o}_t) = \begin{cases} b_j(\mathbf{o}_t) & \text{if } b_j(\mathbf{o}_t) \geq \delta_{min} \\ \delta_{min}, & \text{otherwise} \end{cases} \tag{7.30}$$

Another numerical problem occurs when $\gamma_t(j,k)$ is calculated by Eq. (7.17). The problem lies in finding

$$\frac{c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \mathbf{\Sigma}_{jk}\right)}{\sum_{k=1}^{M} c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \mathbf{\Sigma}_{jk}\right)} \tag{7.31}$$

The problem is that $\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \mathbf{\Sigma}_{jk}\right)$ can actually become zero for every mixture, hence a division by zero occurs. This typically occurs when the dimension of the observations becomes large (say 50 or more using double precision). The

reason why $\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right)$ becomes zero (due to numerical precision), can be observed by the definition

$$\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_{jk}|^{1/2}} e^{-\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)} \tag{7.32}$$

If the exponent $-\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)$ is a large negative number, it results in the expression becoming zero (numerical underflow). The solution to this problem will be to multiply a scale factor e^{ν} with Eq. (7.31) in both the numerator and the denominator which yields

$$\frac{c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) \cdot e^{\nu}}{\sum_{k=1}^{M} c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) \cdot e^{\nu}} \tag{7.33}$$

The effect of multiplying with this exponential factor can be seen in the definition of the normal distribution

$$\begin{aligned} \mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) \cdot e^{\nu} &= \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_{jk}|^{1/2}} e^{-\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)} \cdot e^{\nu} \\ &= \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_{jk}|^{1/2}} e^{-\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right) + \nu} \end{aligned} \tag{7.34}$$

Hence, the large negative exponential term can be adjusted to be zero, where ν can be selected as

$$\nu = -\max_k \left[-\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)\right] \tag{7.35}$$

which implies that the denominator in Eq. (7.31) is always nonzero. Hence, division by zero will not occur, and Eq. (7.31) will still calculate the same mathematically correct value.

The logarithm of the Gaussian mixture can also be rewritten to avoid unnecessary numerical underflow (that is Eq. (7.30) must be applied). The logarithm of the Gaussian mixture is commonly used in the Viterbi algorithms, that is in Eqs. (6.17), (6.20), (6.28) and (6.31). Note that the logarithm of the Gaussian mixture can be mathematically rewritten as

$$\begin{aligned} \log \sum_{k=1}^{M} c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) &= \log \sum_{k=1}^{M} e^{\log\left(c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right)\right)} \\ &= -\kappa + \log \sum_{k=1}^{M} e^{\log\left(c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right)\right)+\kappa} \end{aligned} \tag{7.36}$$

where κ should be chosen so that the exponential in the sum will be a value close to the maximum exponent allowed by the machine precision ($\log(\delta_{max})$). This gives that κ can be formulated as

$$\kappa = \log(\delta_{max}) - \log(M+1) - \max_k \left[\log\left(c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right)\right)\right] \tag{7.37}$$

where

$$\begin{aligned} \log\left(c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right)\right) &= \log(c_{jk}) - \tfrac{D}{2}\log(2\pi) - \tfrac{1}{2}\log(|\Sigma_{jk}|) - \\ &\quad \tfrac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right) \end{aligned} \tag{7.38}$$

7.2 Initial Estimates of HMM Parameters

Before the Baum-Welch reestimation can be applied for training it is important to get good initial parameters so that the reestimation leads to the global maximum or as close as possible to it. An adequate choice for $\boldsymbol{\pi}$ and $\mathbf{A}$ is the uniform distribution when an ergodic model is used. For example, the ergodic model in Fig. 3.2a will have the following initial $\boldsymbol{\pi}$ and $\mathbf{A}$ for a four state model

$$\boldsymbol{\pi} = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}_{4\times 1} \tag{7.39}$$

$$\mathbf{A} = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}_{4\times 4} \tag{7.40}$$

If left-right models are used, $\boldsymbol{\pi}$ will have the probability one for the first state and zero for the other states. For example, the left-right model in Fig. 3.2b have the following initial $\boldsymbol{\pi}$ and $\mathbf{A}$

$$\boldsymbol{\pi} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}_{4\times 1} \tag{7.41}$$

$$\mathbf{A} = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{4\times 4} \tag{7.42}$$

The parameters for the emission distribution need a good initial estimation for rapid and proper convergence. For a left-right model this can be performed by using uniform segmentation into the states of the model, for every training sample. After segmentation, all observations from the state j are collected from all training samples. If an ergodic model is used and no apriori information about the segmentation is given, the state segmentation and the mixture segmentation can both be segmented in an unsupervised manner. This unsupervised segmentation of the observations can be performed by the *clustering algorithms*.

Chapter 8

Clustering Algorithms

Clustering can be viewed as an unsupervised procedure which classifies patterns into groups (*clusters*). The clustering problem can be referred to many contexts and disciplines, however here the clustering problem is approached from the statistical pattern recognition perspective. Many different algorithms exist, and variations of these algorithms for the clustering problem such as: *k-means*, *self organizing feature map (SOFM)*, *hierarchial algorithms*, *evolutionary algorithms (EAs)*, *simulated annealing (SA)*, etc. In this chapter the k-means and the SOFM are explained, and the aim is to use them to find good initial values for the Gaussian mixture, which should be used for the reestimation procedure described in the previous chapter.

The clustering of N data points is a method by which the data points are clustered into K groups (*clusters*) of smaller sets of similar data in an unsupervised manner. The data will here be vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathcal{R}^D$ (the notation for the clusterings algorithms will not follow the notations for the description of the HMM). A clustering algorithm attempts to find K natural groups or centroids, denoted $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K$, of data based on some similarity. To determine which cluster a point belongs to a distance measure between points will be needed, the distance being really a measure of similarity. The Euclidian distance measure is here used

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{d=1}^{D} (x_i(d) - x_j(d))^2} \tag{8.1}$$

A commonly used criterion for clustering is the error criterion which for each point computes the squared distance from the corresponding cluster center and the mean of distances for all points in the data set (that is the *mean square error*)

$$E_{mse} = \frac{1}{N}\sum_{n=1}^{N} d\left(\mathbf{x}_n - \boldsymbol{\mu}_{\arg\min_k [d(\mathbf{x}_n, \boldsymbol{\mu}_k)]}, \mathbf{x}_n - \boldsymbol{\mu}_{\arg\min_k [d(\mathbf{x}_n, \boldsymbol{\mu}_k)]}\right)^2 \tag{8.2}$$

The output from the clustering algorithm is basically a statistical description of the cluster centroids and indices telling what cluster each data point belongs to.

8.1 The K-Means Algorithm

The k-means algorithm can be considered as the workhorse of clustering algorithms. It is a popular clustering method that minimizes the clustering error criterium. However, the k-means algorithm is a local search procedure and it is well known that it suffers from the serious drawback that its performance heavily depends on the initial starting conditions [15]. Here follows the k-means algorithm:

1. **Initialization**

   Choose K vectors from the training vectors, $\mathbf{x}$, at random. These vectors will be the *centroids* $\boldsymbol{\mu}_k$.

2. **Recursion**

   For each vector in the training set, $n = 1, 2, \ldots, N$, let every vector belong to a cluster k. This is done by choosing the cluster closest to the vector

   $$k_n^* = \arg\min_k [d(\mathbf{x}_n, \boldsymbol{\mu}_k)]$$

   where $d(x_n, \mu_k)$ is the distance measure. The Euclidian distance measure is used (in vector notation)

   $$d(\mathbf{x}_n, \boldsymbol{\mu}_k) = \sqrt{(\mathbf{x}_n - \boldsymbol{\mu}_k)^T (\mathbf{x}_n - \boldsymbol{\mu}_k)}$$

3. **Test**

   Recompute the centroids $\boldsymbol{\mu}_k$ by taking the mean of the vectors that belongs to this centroid. This is done for every $\boldsymbol{\mu}_k$. If no vectors belong to $\boldsymbol{\mu}_k$ - create new $\boldsymbol{\mu}_k$ by choosing a random vector from $\mathbf{x}$. If there has not been any change of the centroids from the previous step go to step 4, otherwise go back to step 2.

4. **Termination**

From this clustering, the following parameters can be found

$$
\begin{aligned}
c_k &= \text{number of vectors classified in cluster } k \\
&\quad \text{divided by the total number of vectors} \\
\boldsymbol{\mu}_k &= \text{sample mean of vectors classified in cluster } k \\
\boldsymbol{\Sigma}_k &= \text{sample covariance matrix of the vectors classified in} \\
&\quad \text{cluster } k \\
E_{mse} &= \text{clustering mean square error, see Eq. (8.2)} \\
k_n^* &= \text{index that indicates which center each data vector} \\
&\quad \text{belongs to}
\end{aligned}
$$

To illustrate the problem of the initially selected centers, a simple clustering is made on two dimensional data which is to be clustered in four clusters. Fig. 8.1 shows the result of k-means algorithm when the initial cluster was chosen properly.

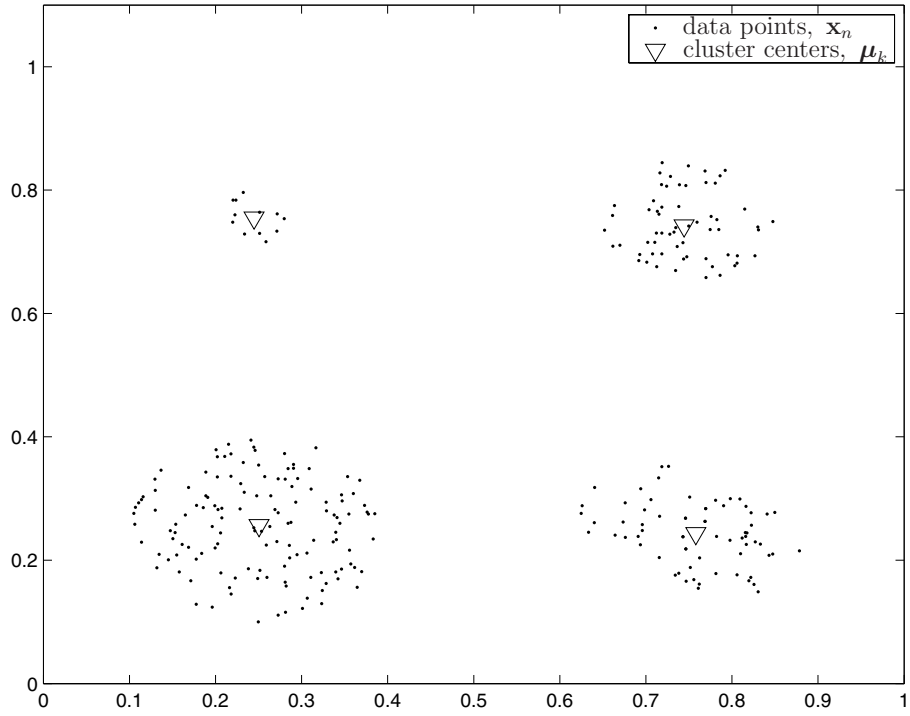


Figure 8.1: K-means clustering into four clusters, with good initial centers.

In Fig. 8.2 the result of k-means algorithm in a different run is shown. Here the initial chosen centers (see initialization of k-means algorithm) were chosen poorly.

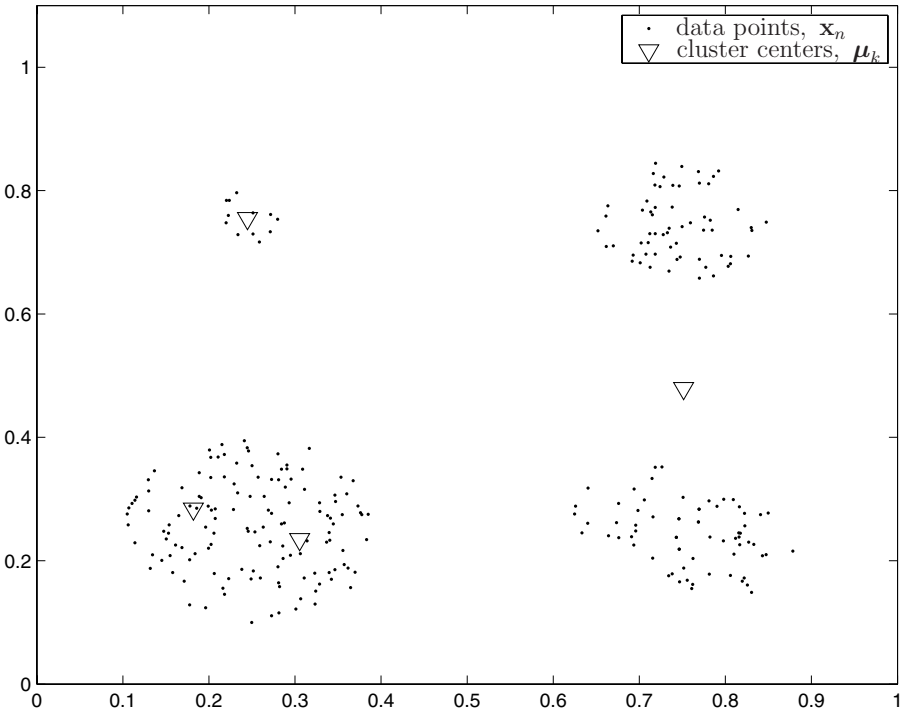


Figure 8.2: K-means clustering into four clusters, with poorly chosen initial centers.

This k-means algorithm is one tool that can be used to find the initial parameters of the Gaussian mixture.

8.2 The SOFM Algorithm

The Self-Organizing Feature Map (SOFM), also known as the Kohonen map [16], is a neural network based on competitive learning and the arrangement of the neurons. The SOFM described here can be seen as an epoch-based variant of the SOFM described in [16]. The principle of the SOFM is to transform the arbitrary dimensional input signals into a one, two or possible three-dimensional discrete map, this to more easily visualize the map. Here the arbitrary input signals will be represented by the input vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathcal{R}^D$ and the dimension to transform to is one or two-dimensional. The map to transform to will be denoted $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_K \in \mathcal{R}^{D_v}$. Two quantities are taken into consideration, the feature map $\mathbf{v}_k$ and the cluster means $\boldsymbol{\mu}_k$, meaning that every cluster centroid will have a topology position in the feature map.

For the one-dimensional feature map the topology will here be a line, see Fig. 8.3.

The two-dimensional feature map will here create a square topology. That

Figure 8.3: Line topology, number of clusters $K = 10$.

is, it will have a perfect square if the number of clusters has an even square root, otherwise the largest square possible is created and the remaining neurons are first added at the top of the square and later to the right of the square, see Fig. 8.4 and Fig. 8.5. Other topologies can of course be applied, e.g. a circle, however the square (or extended square) is here used as the basic topology for a two-dimensional feature map.

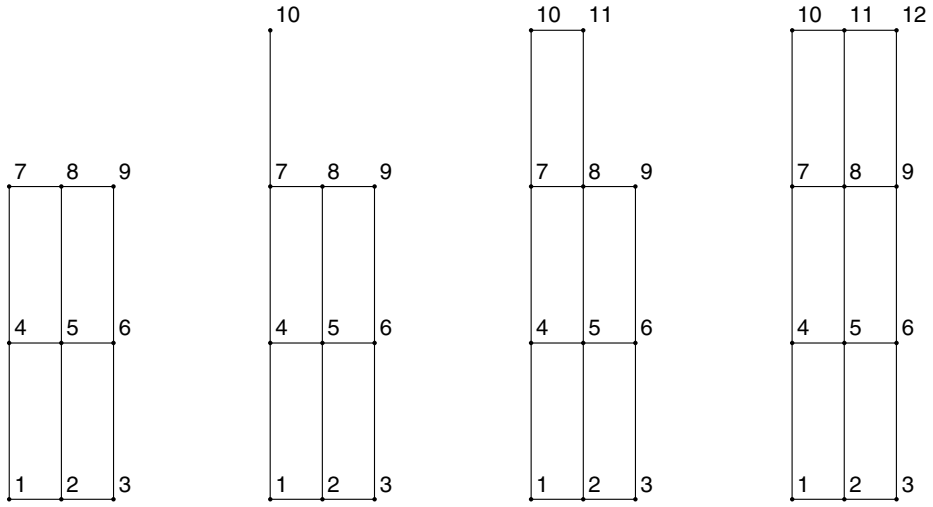


Figure 8.4: The two-dimensional feature map for clustering in $K = 9, 10, 11, 12$ clusters.

Given the selected feature map and initial cluster centroids, the SOFM can be divided into three processes [17]:

1. **Competition process**
   The competition between the neurons for a specific data vector, $\mathbf{x}_n$ will yield a winning neuron. The winning neuron index for the data vector $\mathbf{x}_n$, k_n^*, will be found as

$$k_n^* = \arg\min_k \left[d\left(\mathbf{x}_n, \boldsymbol{\mu}_k\right) \right] \tag{8.3}$$

2. **Cooperation process**
   The winning neuron in the map $\mathbf{v}_{k_n^*}$ locates the of center of cooperating neurons. The question that now arises is how a topological neighborhood

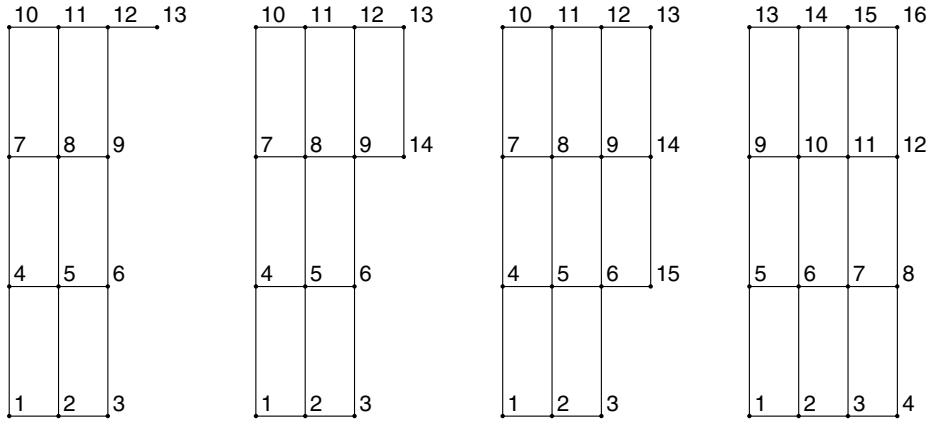


Figure 8.5: The two-dimensional feature map for clustering in $K = 13, 14, 15, 16$ clusters.

should be defined? Neurons that fire tend to excite neurons closer to the firing neuron more than neurons further away, that is in a neurobiological sense [17]. Given this fact it is desired to have a *neighborhood function* $h_{k_n^*,k}$ for the topological map. A natural choice of of the neighborhood function, is the Gaussian function, hence $h_{k_n^*,k}$ is defined as

$$h_{k_n^*,k} = e^{-\frac{\left(d\left(\mathbf{v}_{k_n^*},\mathbf{v}_k\right)\right)^2}{2\sigma^2}} \tag{8.4}$$

where $d\left(\mathbf{v}_{k_n^*},\mathbf{v}_k\right)$ is the Euclidian distance, see Eq. (8.1), between the winning neuron, $\mathbf{v}_{k_n^*}$, and a specific neuron, $\mathbf{v}_k$. The variance, σ^2, of the Gaussian neighborhood function indicates how much the neighborhood neurons should be affected by the winning neuron.

One feature for the neighborhood functions in the SOFM is that it shrinks with time. A common choice for the dependence of σ, which makes the neighborhood function shrink, is the exponential decay. Here a choice for the start variance, σ_0^2, and the end variance, σ_{end}^2, will be described. Let us consider the line topology, see Fig. 8.3, the maximum distance in the map will be denoted $d_{\mathbf{v},max}$ and the minimum distance $d_{\mathbf{v},min}$. The neighborhood function value for the maximum distance in the map h_{max} should be chosen close to one, which implies that essentially all neurons will be affected initially

$$h_{max} = e^{-\frac{d_{\mathbf{v},max}^2}{2\sigma_0^2}} \tag{8.5}$$

This gives that the initial standard deviation can be found as

$$\sigma_0 = \sqrt{-\frac{d^2_{\mathbf{v},max}}{2 \cdot \log(h_{max})}} \tag{8.6}$$

In a similar manner the neighborhood function value for the minimum distance in the map, h_{min}, will give that almost only one neuron will be affected. That is h_{min} is chosen close to zero

$$h_{min} = e^{-\frac{d^2_{\mathbf{v},min}}{2\sigma^2_{end}}} \tag{8.7}$$

The final standard deviation can be found as

$$\sigma_{end} = \sqrt{-\frac{d^2_{\mathbf{v},min}}{2 \cdot \log(h_{min})}} \tag{8.8}$$

Given the initial standard deviation, σ_0, the final, σ_{end}, and the number of iterations, the exponential factor can be found and σ can be updated at every time-step. In Fig. 8.6 the neighborhood function can be found for four time instances.

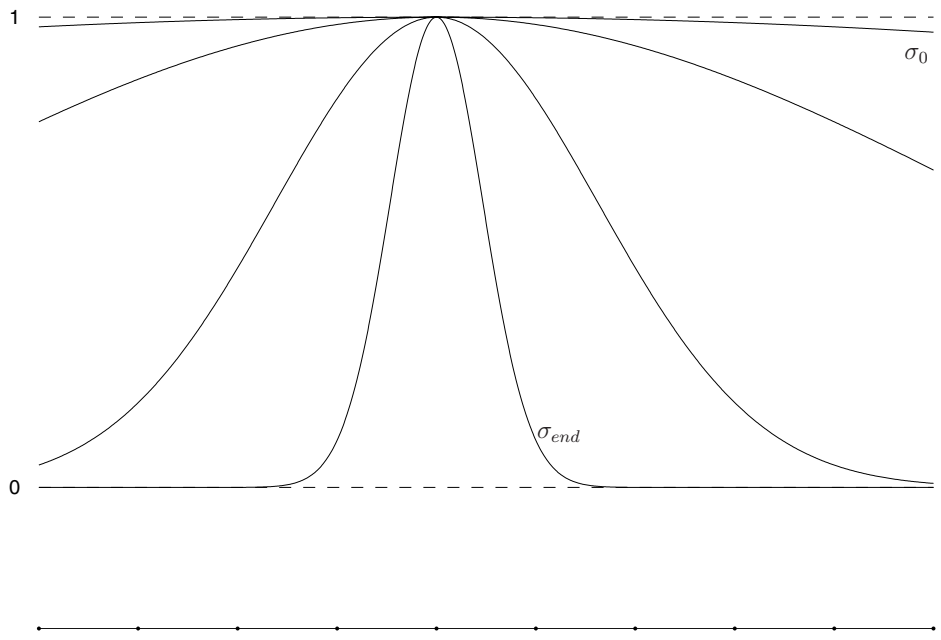


Figure 8.6: The neighborhood function, shrinkage during time, for a one-dimensional feature map. Here the winning neuron is neuron number five.

3. **Adaptive process**

From an initial disorder the SOFM algorithm will gradually converge to an organized representation of the input space, provided that the settings for the algorithm are selected properly. The update equation for the SOFM algorithm contains the neighborhood function and learning rate α. The update equation for one input vector $\mathbf{x}_n$ can be found as

$$\boldsymbol{\mu}_k = \boldsymbol{\mu}_k + \alpha h_{k_n^*,k} \left(\mathbf{x}_n - \boldsymbol{\mu}_k \right), \quad 1 \leq k \leq K \tag{8.9}$$

As mentioned before the neighborhood function can advantageously vary with time and this is also the case for the learning rate. By choosing different settings for the learning rate and the neighborhood function, the adaptation can be divided into two phases; the *ordering phase* and the *convergence phase* [17]:

(a) **Ordering phase**
The ordering phase is the initial phase of the adaptive process. Here the topological ordering of the cluster means takes place. The exact number of iterations for this phase is hard to determine because the learning is a stochastic process, however some suggestions for the number of iterations are given in [16, 17]. Here the training is suggested to be performed in epoch training, that is the N input vectors are used N_{oe} (oe - ordering epoch) times in the ordering phase. In every epoch, the input vectors are chosen in a permutated order to avoid the algorithm to run on the same sequence every epoch.
However, the number of iterations is only dependent of the number of clusters K and not the number of input vectors [16]. Hence the number of epochs for the ordering phase can be determined by

$$N_{oe} = \text{fix} \left(\frac{K \cdot M_o}{N} \right) + 1 \tag{8.10}$$

where M_o is a factor to multiply with the number of clusters to get the total number of iterations. When the number of epochs is decided it is desired, in the ordering phase, to have a shrinking neighborhood function and also a shrinking learning rate. The variance of the neighborhood function will shrink in a exponential manner, as in the cooperation process. The learning rate will decrease in a exponential manner, as well. The learning rate starts at α_0 and ends at α_{end}.

(b) **Convergence phase**
After the ordering phase, the learning rate is fixed at α_{end}, which should have a value that gives small adjustments of the map. In the

Variable	Value
M_o	1000
M_c	2000
α_0	0.5
α_{end}	0.01
h_0	0.9
h_{end}	0.1

Table 8.1: Example of initial settings for a SOFM algorithm.

neighborhood function the nearest neighbors should only affect the winning neuron, that is the variance for h_{min} is used. The number of epochs for the convergence phase N_{ce} (ce - convergence epoch) can in a similar way, as in the ordering phase, be found as

$$N_{ce} = \text{fix}\left(\frac{K \cdot M_c}{N}\right) + 1 \tag{8.11}$$

where M_c is a factor multiplied with the number of clusters K which results in the desired number of iterations N_{ce}.

Given the number of clusters desired and the feature map, in which $d_{\mathbf{v},max}$ and $d_{\mathbf{v},min}$ can be found, the initial settings for the algorithm are proposed according to Tab. 8.1. Note that this is not the optimal settings for all situations, since the learning is a stochastic process [16]. Also note that σ_0, σ_{end}, N_{oe} and N_{ce} can be found from the variables in the table, by applying Eqs. (8.6), (8.8), (8.10) and (8.11).

Here follows the SOFM algorithm:

1. **Initialization**

   Choose K vectors from $\mathbf{x}$ at random. These vectors will be the initial *centroids* $\boldsymbol{\mu}_k$, which are going to be found correctly. Choose the desired feature map dimension, D_v (typically one or two). Create a feature map $\mathbf{v}_n \in \mathcal{R}^{D_v}$ and find $d_{\mathbf{v},max}$ and $d_{\mathbf{v},min}$. Set the start and end values for the learning rate, α_0 and α_{end}. The start and end values h_{max} and h_{min}. Set the number of iterations controls M_o and M_c. Apply Eqs. (8.6), (8.8), (8.10) and (8.11) to find σ_0, σ_{end}, N_{oe} and N_{ce}.

2. **Ordering phase**

   Iterate the following N_{oe} times:

   (a) Choose a permutated order of the N data points

(b) Iterate N times (where n is an index that follows the permutated order):

i. Find the winning neuron according to:

$$k_n^* = \arg\min_k \left[d\left(\mathbf{x}_n, \boldsymbol{\mu}_k\right) \right]$$

ii. Update the cluster means according to:

$$\boldsymbol{\mu}_k = \boldsymbol{\mu}_k + \alpha h_{k_n^*,k} \left(\mathbf{x}_n - \boldsymbol{\mu}_k\right), \quad 1 \leq k \leq K$$

iii. Decrease α and σ in a exponential manner

3. **Convergence phase**

Iterate the following N_{ce} times:

(a) Choose a permutated order of the N data points

(b) Iterate the following N times (where n is an index that follows the permutated order):

i. Find the winning neuron according to:

$$k_n^* = \arg\min_k \left[d\left(\mathbf{x}_n, \boldsymbol{\mu}_k\right) \right]$$

ii. Update the cluster means according to:

$$\boldsymbol{\mu}_k = \boldsymbol{\mu}_k + \alpha h_{k_n^*,k} \left(\mathbf{x}_n - \boldsymbol{\mu}_k\right), \quad 1 \leq k \leq K$$

4. **Termination**

From the clustering, the following parameters can be found:

c_k	=	number of vectors classified in cluster k divided by the total number of vectors
$\boldsymbol{\mu}_k$	=	sample mean of vectors classified in cluster k
$\boldsymbol{\Sigma}_k$	=	sample covariance matrix of the vectors classified in cluster k
E_{mse}	=	the mean square error for the clustering, see Eq. (8.2)
k_n^*	=	index that indicates which center each data vector belongs to

For an example of how the SOFM can perform consider the input distribution in Fig. 8.7 ($\sim$ 9000 data points).

Applying the one-dimensional feature map and using a clustering into 2500 clusters yields the result as in Fig. 8.8.

Using a two-dimensional feature map and a clustering into 50×50 clusters yields the result according to Fig. 8.9.

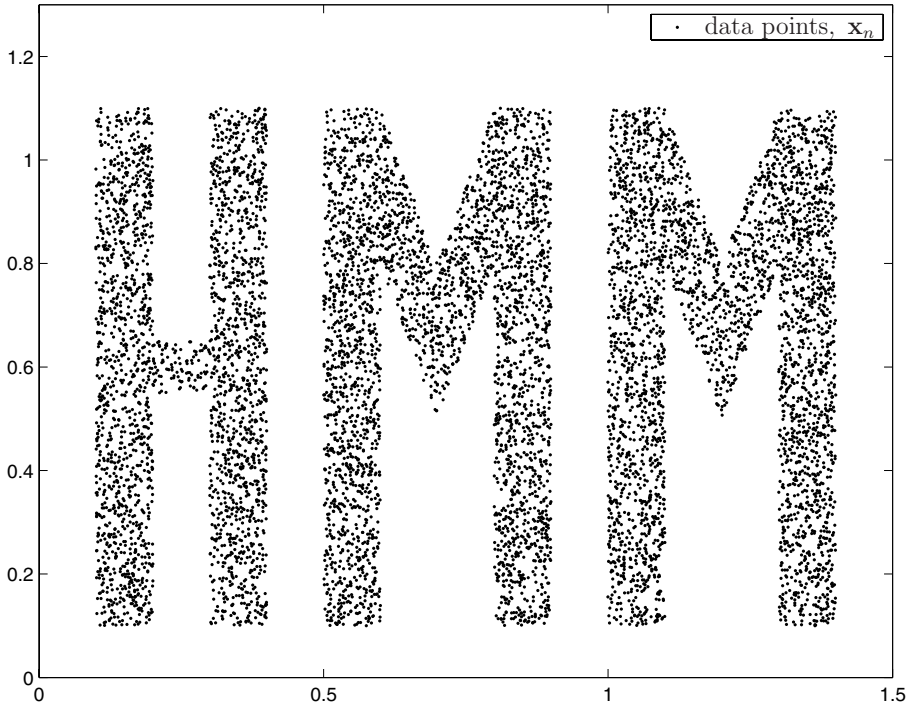


Figure 8.7: Distribution of the data points to cluster.

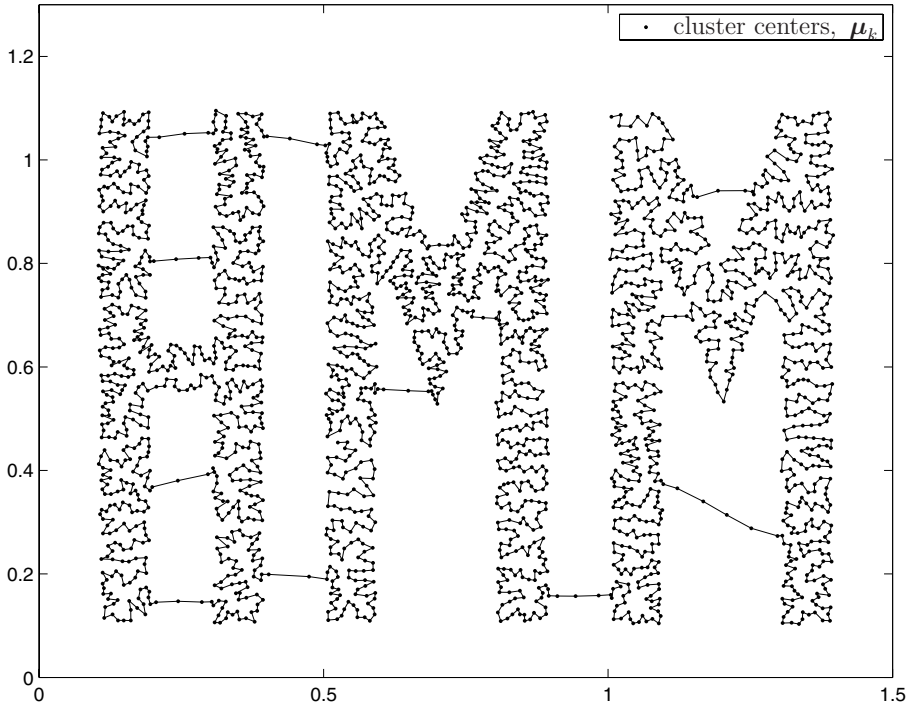


Figure 8.8: Result with a one-dimensional feature map. The settings used are found according to Tab. 8.1.

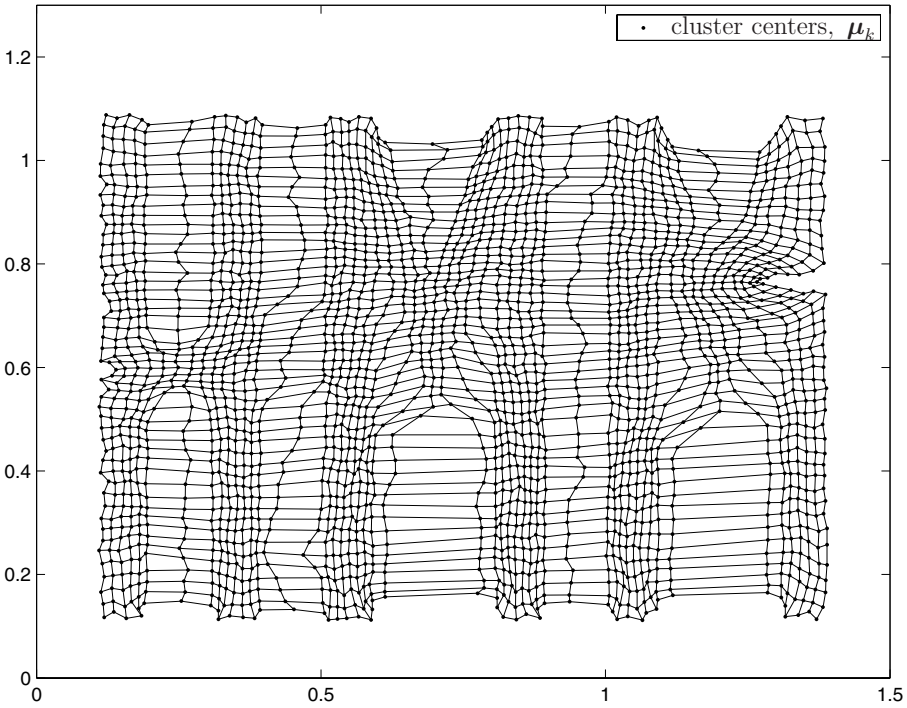


Figure 8.9: Result with a two-dimensional feature map. The settings used are found according to Tab. 8.1.

Chapter 9

Training and Recognition using Hidden Markov Models

In this chapter a description of the HMM used in a pattern recognition task is addressed. First the training aspects of models are discussed followed by a description of how to use the resulting models for a recognition task. The training has already been explained to some extent in problem three - parameter estimation, however, the overview of more complete training of a HMM will be described here. The continuous mixture densities and diagonal covariance matrices are assumed here as it has been found to be more convenient and sometimes preferable to use diagonal covariances and several mixtures, rather than a few mixtures using full covariances [2]. The reason is that it is hard to get a reliable estimation of the off diagonal values in the covariance matrix, which can lead to a matrix inversion problem when the Gaussian pdf is calculated.

9.1 Training of a Hidden Markov Model

Given the training samples a suitable model should be chosen, see Fig. 9.1.

The first decision for a HMM modelling is the type of model and here two typical models are considered; the left-right model and the ergodic model. The ergodic model is the most general because all states are connected. Left-right models are commonly used for patterns where the features follow in a successive order.

The second thing to decide is the number of states N to use. The number of states is usually connected to a corresponding real phenomenon in the patterns [2]. The number of mixtures (or codewords when discrete densities are used) M has to be decided as well.

The next step is to divide the observations into states and mixtures. First the observation vectors are segmented into the model states. If the information about which state each vector belongs to is given (transcribed observations) no

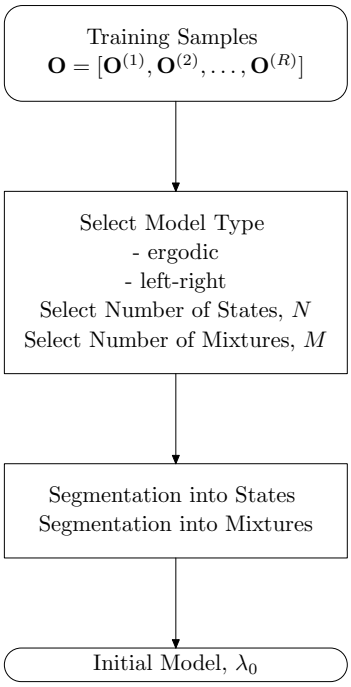


Figure 9.1: Choices to be made before reestimation.

unsupervised method is needed, otherwise some unsupervised method is used (i.e. *k-means* or *SOFM*). If an ergodic model is used and an unsupervised method applied, all observations are collected and clustered into the desired number of states. In a left-right model the uniform segmentation into states can preferably be used and every observation sequence can be divided into states uniformly as in Fig. 9.2. The vectors can be divided into the state regions according to

$$\lfloor 1 \ldots T_r/N \ldots 2 \cdot T_r/N \ldots 3 \cdot T_r/N \ldots N \cdot T_r/N \rfloor \tag{9.1}$$

which implies that the observations $\lfloor 1 \ldots T_r/N \rfloor$ will belong to state one , $\lfloor T_r/N + 1 \ldots 2 \cdot T_r/N \rfloor$ state two etc.

Given the state segmentation, the mixture segmentation takes place on each state. Vectors that belong to a state are collected for all observation sequences and an unsupervised segmentation algorithm is applied to these vectors.

From initial decisions and segmentations a good initial model, λ_0, is found. This model is then used for reestimation and given this initial model λ_0, the training will continue in an iterative manner, see Fig. 9.3.

The detailed actions for the steps in Fig.9.3 can be found below:

- **Check model**
  The model is checked for numerical stability according to Eqs. (7.28) and (7.29).

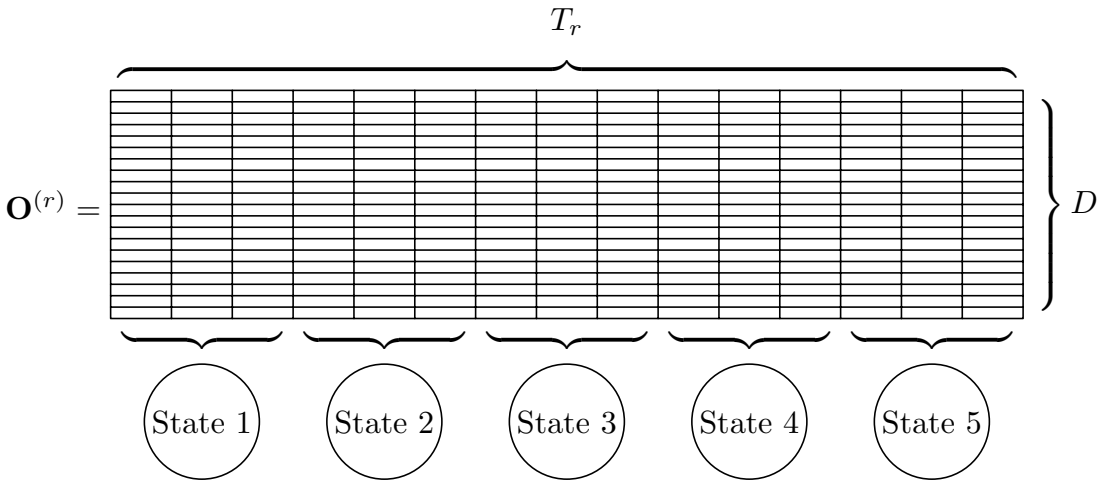


Figure 9.2: Uniform segmentation into states for a left-right model.

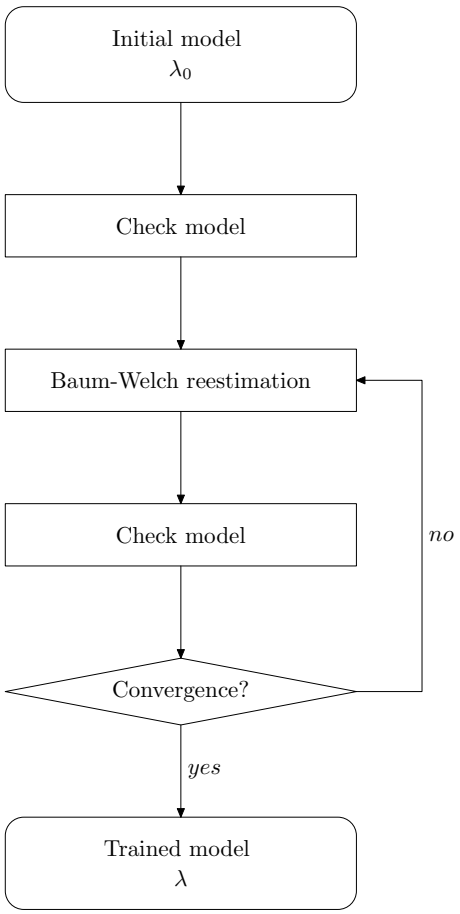


Figure 9.3: Training of a HMM.

- **Baum-Welch reestimation**
  The model is reestimated according to Eqs. (7.23), (7.24), (7.25), (7.26) and (7.27). For the reestimation process the scaled forward variable $\hat{\alpha}_t(i)$ and the scaled backward variable $\hat{\beta}_t(i)$ are used. Given these variables $\gamma_t(i)$, $\xi_t(i,j)$ and $\gamma_t(j,k)$ can be found according to Eqs. (7.10), (7.11) and (7.17), respectively. Note that Eqs. (7.33) and (7.35) should be used to avoid numerical problems in Eq. (7.17).

- **Convergence**
  The reestimation is ended when the desired maximum number of iterations is reached, denoted $MaxIter$, or when the learning curve reaches a certain limit. The learning curve is defined as

$$J(n) = \left| \frac{\left(\widetilde{P}_{tot}(n) - \widetilde{P}_{tot}(n-1) \right)}{\widetilde{P}_{tot}(n)} \right| \tag{9.2}$$

  where n is the iteration index. $\widetilde{P}_{tot}(n)$ is the total probability for all samples at iteration n, and is defined as

$$\widetilde{P}_{tot}(n) = \sum_{r=1}^{R} \widetilde{P}\left(\mathbf{O}^{(r)}, \mathbf{q}^{*(r)} | \lambda_n \right) \tag{9.3}$$

  where $\widetilde{P}\left(\mathbf{O}^{(r)}, \mathbf{q}^{*(r)} | \lambda_n \right)$ can be found by the fast alternative Viterbi algorithm (in this case only the probability is needed, the actual state sequence is not needed here). Since Eq. (7.22) is going to be maximized it will be the same as maximizing the logarithm of Eq. (7.22). This gives that the maximation can be reformulated as

$$\log \prod_{r=1}^{R} P\left(\mathbf{O}^{(r)} | \lambda \right) = \sum_{r=1}^{R} \log P\left(\mathbf{O}^{(r)} | \lambda \right) \tag{9.4}$$

  where $P\left(\mathbf{O}^{(r)} | \lambda \right)$ can be found by the Viterbi algorithm or the forward algorithm. However, the fast alternative Viterbi algorithm can advantageously be used here. The output is already in the log domain and it is faster than the forward algorithm (more about this in the next section).

  Note that the learning curve $J(n)$ can be seen as a normalized derivation of $\widetilde{P}_{tot}(n)$ and when $J(n)$ reaches a small value the iteration scheme is ended. This value will here be denoted ϵ_{stop} and is in a typical application in the range $0 - 10^{-4}$. Also a minimum number of iterations, denoted $MinIter$, can be set, to make sure that a minimum of iterations are made.

Variable	Description	Typical value(s)
N	Number of states	-
M	Number of mixtures	-
δ_{max}	Largest number due to computer precision	$1.7977 \cdot 10^{308}$
δ_{min}	Smallest number due to computer precision	$2.2251 \cdot 10^{-308}$
$MinIter$	Minimum number of reestimations	-
$MaxIter$	Maximum number of reestimations	-
ϵ_{stop}	Stopping criteria for reestimation	$0 - 10^{-4}$, $5 \cdot 10^{-5}$
ϵ_c	Numerical floor on the mixture weights	$10^{-10} - 10^{-3}$, 10^{-5}
ϵ_Σ	Numerical floor on the diagonal of the covariances	$10^{-10} - 10^{-3}$, 10^{-5}, $\delta_{min}^{1/D}$

Table 9.1: Essential variables for HMM training, double precision is considered.

Various variables and settings are needed to train a HMM, see summation in Tab. 9.1. It should be emphasized that these settings are from subjective empirical experiments in speech and face recognition using double precision.

9.2 Recognition using Hidden Markov Models

Given V trained models the task is to choose the model with the highest possible probability for an unknown observation. The probabilities that can be used are based on *the scaled forward algorithm*, *the alternative Viterbi algorithm* and *the fast alternative Viterbi algorithm*. However, the alternative Viterbi algorithm is faster than the scaled forward algorithm (a maximum operation is usually faster than a sum), and further, the fast alternative Viterbi algorithm is faster than the alternative Viterbi algorithm (due to the fact that the actual state sequence is not needed). This means that the fast alternative Viterbi algorithm could preferably be used. Note that Eqs. (7.36), (7.37) and (7.38) can advantageously be used for better numerical stability in the algorithm.

The fast alternative Viterbi algorithm can now be used to find the probabilities for each model given an unknown observation $\mathbf{O}$. The winning model can be found as

$$v^* = \arg \max_{1 \leq v \leq V} \left[\widetilde{P} \left(\mathbf{O}, \mathbf{q}^* | \lambda^v \right) \right] \tag{9.5}$$

These are the steps in the recognition, see Fig. 9.4.

In Fig. 9.4 there is no control over what happens if the observation sequence given the system is an unseen entity. This means that if the system is given some class which it is not aware of (trained on), the recognition will assign this unseen entity to the most likely model.

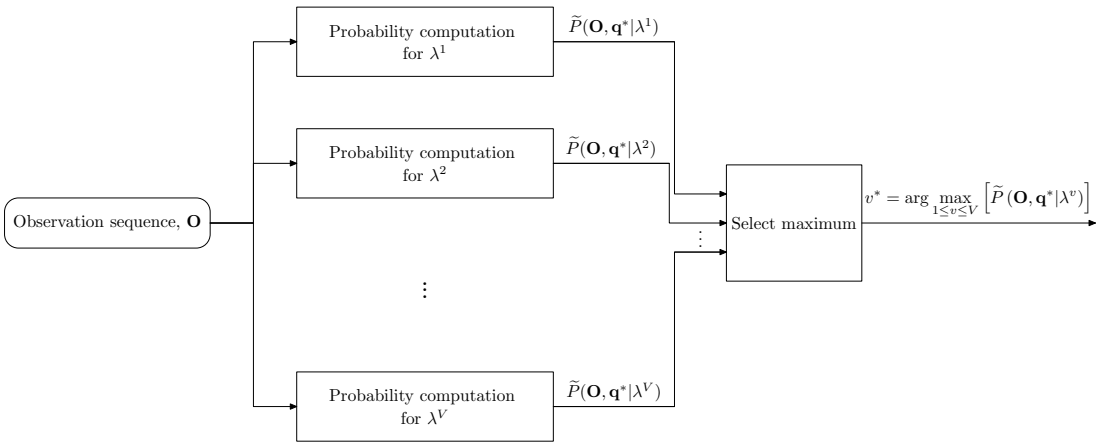


Figure 9.4: Recognition using HMMs.

This implies that *acceptance logic* could be applied in order to avoid acceptance of unknown entities, see Fig. 9.5. This acceptance logic could use some threshold acting on the probability for the models. Each model is then accepted to further compete with the remaining models according to

$$\text{if} \begin{cases} \widetilde{P}\left(\mathbf{O}, \mathbf{q}^*|\lambda^v\right) \geq P_v & \text{accept} \\ \widetilde{P}\left(\mathbf{O}, \mathbf{q}^*|\lambda^v\right) < P_v & \text{reject} \end{cases} \tag{9.6}$$

where P_v is the threshold for model v. The choice for P_v can be made experimentally or possibly by creating a function of the R training probabilities $\widetilde{P}\left(\mathbf{O}^{(r)}, \mathbf{q}^{*(r)}|\lambda^v\right)$.

Another approach could be to use some information about the relationship between the found probabilities. Consider the transformation of the output probabilities

$$\widetilde{\mathbf{p}} = \widetilde{p}_v = \widetilde{P}\left(\mathbf{O}, \mathbf{q}^*|\lambda^v\right) - \min_v \left[\widetilde{P}\left(\mathbf{O}, \mathbf{q}^*|\lambda^v\right)\right] + 1 \tag{9.7}$$

which yields a vector $\widetilde{\mathbf{p}}$. The least probable model has value one, and more likely models will have values greater than one. Furthermore, consider the normalized vector of the probabilities

$$\mathbf{p} = p_v = \frac{\widetilde{p}_v}{\sum_{i=1}^{V} \widetilde{p}_i}, \quad v = 1, 2, \ldots, V \tag{9.8}$$

and also consider the uniform probability vector

$$\overline{\mathbf{p}} = \overline{p}_v = \frac{1}{V}, \quad v = 1, 2, \ldots, V \tag{9.9}$$

Clearly, if $\mathbf{p}$ and $\overline{\mathbf{p}}$ are similar the decision of the winning model will become hard. On the other hand, if the difference between the probability vectors is higher this means that $\mathbf{p}$ has some high values and some low, which in turn makes the decision easier (less confusion)[1].

The obvious question is now how to create a similarity measure. One way is to use the *Jensen difference* [18, 19]. The Jensen difference is based on the *Shannon entropy*

$$H_{\mathbf{V}}(\mathbf{p}) = -\sum_{i=1}^{V} p_i \log(p_i) \tag{9.10}$$

and is defined as[2]

$$J_{\mathbf{V}}(\mathbf{p}, \overline{\mathbf{p}}) = H_{\mathbf{V}}\left(\frac{\mathbf{p}+\overline{\mathbf{p}}}{2}\right) - \frac{1}{2}\left(H_{\mathbf{V}}(\mathbf{p}) + H_{\mathbf{V}}(\overline{\mathbf{p}})\right) \tag{9.11}$$

The Jensen difference is always non-negative and becomes zero only if $\mathbf{p} = \overline{\mathbf{p}}$ [19]. The decision for accepting a model can be made by setting a threshold on the Jensen difference, that is

$$\text{if} \begin{cases} J_{\mathbf{V}}(\mathbf{p}, \overline{\mathbf{p}}) \geq \varphi & \text{accept} \\ J_{\mathbf{V}}(\mathbf{p}, \overline{\mathbf{p}}) < \varphi & \text{reject} \end{cases} \tag{9.12}$$

The threshold φ is typically found by experiments, since the Jensen difference will be a measure highly dependent on the various decisions made for the pattern recognition task.

Also, information from the observation sequence can directly be used in the acceptance logic. For example in some pattern recognition tasks the length of observation sequence T could be used to remove observation sequences that are of unnatural length.

[1] That is, if $V > 1$, otherwise is $\mathbf{p} = \overline{\mathbf{p}}$.

[2] The learning rate defined in Eq. (9.2) is not the same as the Jensen difference. The Jensen difference has a subindex $\mathbf{V}$ and two input arguments and the learning rate only one argument and no subindex.

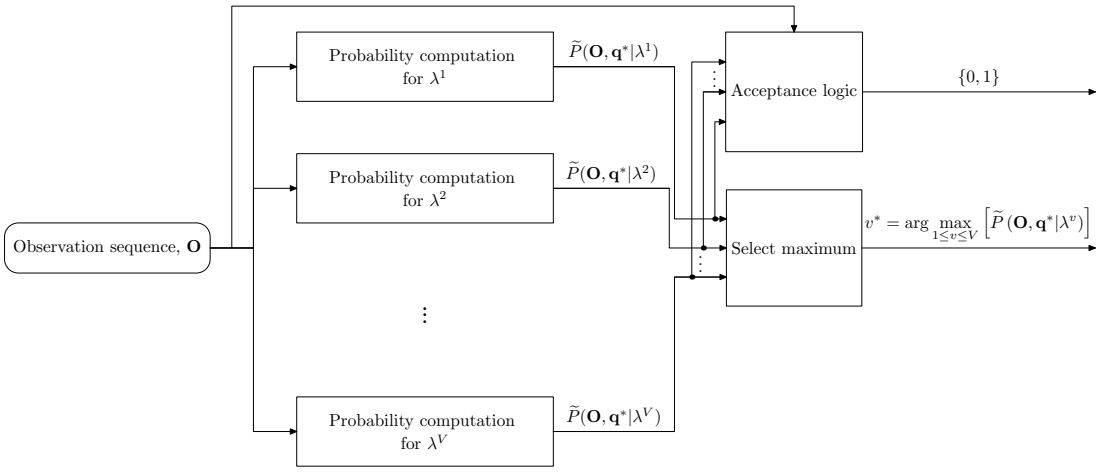


Figure 9.5: Recognition with an acceptance decision included using HMMs.

Chapter 10

Summary and Conclusions

The theory from the Markov model leading into the Hidden Markov Model (HMM) has been investigated with numerical issues on a floating point processor in consideration. Theoretical algorithms, with the emphasis on the use of Gaussian mixtures, such as *forward*, *backward* and *Viterbi*, have been extended to more applied algorithms such as *the scaled forward*, *the scaled backward*, *the alternative Viterbi*, *the fast alternative Viterbi*. Numerical issues concerning the Baum-Welch reestimation have been investigated and suggestions for numerical stability have been presented.

Unsupervised clustering using *k-means* and *SOFM* has been explored in order to get mean vectors and covariance matrices for an initial model if no transcription of the data exists.

The report summarizes the use of the described algorithms for a complete pattern recognition task using HMMs.

Appendix A

Derivation of Reestimation Formulas

The reestimation Eqs. (7.23), (7.24), (7.25), (7.26) and (7.27) are derived. Consider the observation sequences $\mathbf{O} = \left[\mathbf{O}^{(1)}, \mathbf{O}^{(2)} \ldots, \mathbf{O}^{(R)}\right]$ where $\mathbf{O}^{(r)} = \left(\mathbf{o}_1^{(r)}, \mathbf{o}_2^{(r)}, \ldots, \mathbf{o}_{T_r}^{(r)}\right)$ is the rth observation sequence and T_r is the rth observation sequence length, just as in Eq. (7.21). The reestimation formulas are found by assuming the independence between the observation sequences. However, let us start by avoiding this assumption and derive the independence equations from a more general description. The general description of the dependencies for these observation sequences can be found as [20]

$$\left\{\begin{array}{rcl} P\left(\mathbf{O}|\lambda\right) & = & P\left(\mathbf{O}^{(1)}|\lambda\right) \cdot P\left(\mathbf{O}^{(2)}|\mathbf{O}^{(1)},\lambda\right) \cdot \ldots \cdot P\left(\mathbf{O}^{(R)}|\mathbf{O}^{(R-1)},\mathbf{O}^{(R-2)},\ldots,\mathbf{O}^{(1)},\lambda\right) \\ P\left(\mathbf{O}|\lambda\right) & = & P\left(\mathbf{O}^{(2)}|\lambda\right) \cdot P\left(\mathbf{O}^{(3)}|\mathbf{O}^{(2)},\lambda\right) \cdot \ldots \cdot P\left(\mathbf{O}^{(1)}|\mathbf{O}^{(R)},\mathbf{O}^{(R-1)},\ldots,\mathbf{O}^{(2)},\lambda\right) \\ \quad \vdots & & \\ P\left(\mathbf{O}|\lambda\right) & = & P\left(\mathbf{O}^{(R)}|\lambda\right) \cdot P\left(\mathbf{O}^{(1)}|\mathbf{O}^{(R)},\lambda\right) \cdot \ldots \cdot P\left(\mathbf{O}^{(R-1)}|\mathbf{O}^{(R-2)},\mathbf{O}^{(R-3)},\ldots,\mathbf{O}^{(1)},\lambda\right) \end{array}\right. \tag{A.1}$$

Given the equations in Eq. (A.1), the multiple observation probability given the model λ can be found as

$$P\left(\mathbf{O}|\lambda\right) = \sum_{r=1}^{R} w_r P\left(\mathbf{O}^{(r)}|\lambda\right) \tag{A.2}$$

where w_r are the combinatorial weights and defined as (general case)

$$\left\{\begin{array}{rcl} w_1 & = & \frac{1}{R} P\left(\mathbf{O}^{(2)}|\mathbf{O}^{(1)},\lambda\right) \cdot \ldots \cdot P\left(\mathbf{O}^{(R)}|\mathbf{O}^{(R-1)},\mathbf{O}^{(R-2)},\ldots,\mathbf{O}^{(1)},\lambda\right) \\ w_2 & = & \frac{1}{R} P\left(\mathbf{O}^{(3)}|\mathbf{O}^{(2)}\lambda\right) \cdot \ldots \cdot P\left(\mathbf{O}^{(1)}|\mathbf{O}^{(R)},\mathbf{O}^{(R-1)},\ldots,\mathbf{O}^{(2)},\lambda\right) \\ \quad \vdots & & \\ w_R & = & \frac{1}{R} P\left(\mathbf{O}^{(1)}|\mathbf{O}^{(R)},\lambda\right) \cdot \ldots \cdot P\left(\mathbf{O}^{(R-1)}|\mathbf{O}^{(R-2)},\mathbf{O}^{(R-3)},\ldots,\mathbf{O}^{(1)},\lambda\right) \end{array}\right. \tag{A.3}$$

The aim is now to maximize $P\left(\mathbf{O}^{(r)}|\lambda\right)$ (the extension to $P\left(\mathbf{O}|\lambda\right)$ will follow). The first approach is to use a maximum likelihood (ML) estimation for this problem. However, this can not be done due to lack of information (or hidden information), i.e. the state sequence. This is why the expectation maximization (EM) algorithm can advantageously be applied [14, 12, 13, 21].

Consider one of the observations sequences $\mathbf{O}^{(r)}$, the incomplete-data likelihood function for this sequence is $P\left(\mathbf{O}^{(r)}|\lambda\right)$ whereas the complete likelihood function is $P\left(\mathbf{O}^{(r)},\mathbf{q}|\lambda\right)$. The auxiliary Baum function for sample r is then [14]

$$Q_r\left(\lambda',\lambda\right)=\sum_{\text{all }\mathbf{q}}P\left(\mathbf{O}^{(r)},\mathbf{q}|\lambda\right)\log P\left(\mathbf{O}^{(r)},\mathbf{q}|\lambda'\right) \tag{A.4}$$

where λ is the current estimate of parameters for the HMM, and λ' is the newly found parameters which will improve the probability $P\left(\mathbf{O}|\lambda\right)$. This auxiliary Baum function has the following property (which can be seen by applying the Jensen's inequality [14])

$$Q_r\left(\lambda',\lambda\right)\geq Q_r\left(\lambda,\lambda\right)\rightarrow P\left(\mathbf{O}^{(r)}|\lambda'\right)\geq P\left(\mathbf{O}^{(r)}|\lambda\right) \tag{A.5}$$

By considering Eqs. (5.3) and (A.4) the following can be concluded

$$P\left(\mathbf{O}^{(r)},\mathbf{q}|\lambda'\right)=\pi_{q_1}b_{q_1}(\mathbf{o}_1^{(r)})\prod_{t=2}^{T}a_{q_{t-1}q_t}b_{q_t}(\mathbf{o}_t^{(r)}) \tag{A.6}$$

and

$$\begin{aligned}\log P\left(\mathbf{O}^{(r)},\mathbf{q}|\lambda'\right) &= \log\left(\pi'_{q_1}b'_{q_1}(\mathbf{o}_1^{(r)})\prod_{t=2}^{T}a'_{q_{t-1}q_t}b'_{q_t}(\mathbf{o}_t^{(r)})\right)\\ &= \log\left(\pi'_{q_1}\right)+\sum_{t=2}^{T}\log\left(a'_{q_{t-1}q_t}\right)+\sum_{t=1}^{T}\log\left(b'_{q_t}(\mathbf{o}_t^{(r)})\right)\end{aligned} \tag{A.7}$$

Hence Q_r can be rewritten as

$$\begin{aligned}Q_r\left(\lambda',\lambda\right) = &\sum_{\text{all }\mathbf{q}}\left(\log\left(\pi_{q'_1}\right)\right)P\left(\mathbf{O}^{(r)},\mathbf{q}|\lambda\right)+\\ &\sum_{\text{all }\mathbf{q}}\left(\sum_{t=2}^{T_r}\log\left(a'_{q_{t-1}q_t}\right)\right)P\left(\mathbf{O}^{(r)},\mathbf{q}|\lambda\right)+\\ &\sum_{\text{all }\mathbf{q}}\left(\sum_{t=1}^{T_r}\log\left(b'_{q_t}(\mathbf{o}_t^{(r)})\right)\right)P\left(\mathbf{O}^{(r)},\mathbf{q}|\lambda\right)\end{aligned} \tag{A.8}$$

Of course the aim is to maximize $P\left(\mathbf{O}|\lambda\right)$, and the extended auxiliary function can thereby be found as

$$Q\left(\lambda',\lambda\right)=\sum_{r=1}^{R}w_rQ_r\left(\lambda',\lambda\right) \tag{A.9}$$

and similar as for the Q_r the following rule holds [20]

$$Q(\lambda', \lambda) \geq Q(\lambda, \lambda) \rightarrow P(\mathbf{O}|\lambda') \geq P(\mathbf{O}|\lambda) \tag{A.10}$$

By considering Eq. (A.8), Eq. (A.9) can be rewritten as

$$\begin{aligned} Q(\lambda', \lambda) &= \underbrace{\sum_{r=1}^{R} w_r \sum_{\text{all } \mathbf{q}} \left(\log\left(\pi_{q_1'}\right)\right) P\left(\mathbf{O}^{(r)}, \mathbf{q}|\lambda\right)}_{Q_{\pi_i}(\lambda', \lambda)} + \\ &\quad \underbrace{\sum_{r=1}^{R} w_r \sum_{\text{all } \mathbf{q}} \left(\sum_{t=2}^{T_r} \log\left(a'_{q_{t-1}q_t}\right)\right) P\left(\mathbf{O}^{(r)}, \mathbf{q}|\lambda\right)}_{Q_{a_{ij}}(\lambda', \lambda)} + \\ &\quad \underbrace{\sum_{r=1}^{R} w_r \sum_{\text{all } \mathbf{q}} \left(\sum_{t=1}^{T_r} \log\left(b'_{q_t}(\mathbf{o}_t^{(r)})\right)\right) P\left(\mathbf{O}^{(r)}, \mathbf{q}|\lambda\right)}_{Q_{b_j}(\lambda', \lambda)} \\ &= Q_{\pi_i}(\lambda', \lambda) + Q_{a_{ij}}(\lambda', \lambda) + Q_{b_j}(\lambda', \lambda) \end{aligned} \tag{A.11}$$

Since the parameters which are to be optimized now are independently divided into $Q_{\pi_i}(\lambda', \lambda)$, $Q_{a_{ij}}(\lambda', \lambda)$ and $Q_{b_j}(\lambda', \lambda)$, the optimization can be performed on each term individually.

A.1 Optimization of Q_{π_i}

Here the optimization of $Q_{\pi_i}(\lambda', \lambda)$ is performed by using Lagrange multipliers. The function to optimize is

$$\begin{aligned} Q_{\pi_i}(\lambda', \lambda) &= \sum_{r=1}^{R} w_r \sum_{\text{all } \mathbf{q}} \left(\log\left(\pi'_{q_1}\right)\right) P\left(\mathbf{O}^{(r)}, \mathbf{q}|\lambda\right) \\ &= \sum_{r=1}^{R} w_r \sum_{i=1}^{N} \left(\log\left(\pi'_i\right)\right) P\left(\mathbf{O}^{(r)}, q_1 = i|\lambda\right) \end{aligned} \tag{A.12}$$

since selecting all possible state sequences (all $\mathbf{q}$) is here really the repeatedly selection of q_1. The standard stochastic constraint for this optimization is

$$\sum_{i=1}^{N} \pi'_i = 1 \tag{A.13}$$

To determine the optimal parameters, Lagrange multipliers are applied. This gives the two partial derivatives

$$\begin{array}{l} \frac{\partial}{\partial \pi_i^{'}} \left[\sum_{r=1}^{R} w_r \sum_{i=1}^{N} \left(\log \left(\pi_i^{'} \right) \right) P \left(\mathbf{O}^{(r)}, q_1 = i | \lambda \right) + \rho \left(1 - \sum_{i=1}^{N} \pi_i^{'} \right) \right] = 0 \Leftrightarrow \\ \sum_{r=1}^{R} w_r \frac{1}{\pi_i^{'}} P \left(\mathbf{O}^{(r)}, q_1 = i | \lambda \right) - \rho = 0 \end{array} \tag{A.14}$$

and

$$\begin{array}{l} \frac{\partial}{\partial \rho} \left[\sum_{r=1}^{R} w_r \sum_{i=1}^{N} \left(\log \left(\pi_i^{'} \right) \right) P \left(\mathbf{O}^{(r)}, q_1 = i | \lambda \right) + \rho \left(1 - \sum_{i=1}^{N} \pi_i^{'} \right) \right] = 0 \Leftrightarrow \\ \sum_{i=1}^{N} \pi_i^{'} = 1 \end{array} \tag{A.15}$$

where ρ is the Lagrange multiplier. From Eq. (A.14) ρ can be found as

$$\rho = \frac{1}{\pi_i^{'}} \sum_{r=1}^{R} w_r P \left(\mathbf{O}^{(r)}, q_1 = i | \lambda \right) \tag{A.16}$$

From the same equation $\pi_i^{'}$ can be found as

$$\pi_i^{'} = \frac{1}{\rho} \sum_{r=1}^{R} w_r P \left(\mathbf{O}^{(r)}, q_1 = i | \lambda \right) \tag{A.17}$$

By inserting Eq. (A.17) into Eq. (A.15), an expression for ρ becomes eminent

$$\begin{array}{l} \sum_{i=1}^{N} \frac{1}{\rho} \sum_{r=1}^{R} w_r P \left(\mathbf{O}^{(r)}, q_1 = i | \lambda \right) = 1 \Leftrightarrow \\ \rho = \sum_{i=1}^{N} \sum_{r=1}^{R} w_r P \left(\mathbf{O}^{(r)}, q_1 = i | \lambda \right) \end{array} \tag{A.18}$$

Now two expressions for ρ are found, that is Eqs. (A.16) and (A.18). Since only Eq. (A.16) contains $\pi_i^{'}$, the equality between them can be used to extract $\pi_i^{'}$

$$\begin{array}{l} \frac{1}{\pi_i^{'}} \sum_{r=1}^{R} w_r P \left(\mathbf{O}^{(r)}, q_1 = i | \lambda \right) = \sum_{i=1}^{N} \sum_{r=1}^{R} w_r P \left(\mathbf{O}^{(r)}, q_1 = i | \lambda \right) \Leftrightarrow \\ \pi_i^{'} = \frac{\sum_{r=1}^{R} w_r P \left(\mathbf{O}^{(r)}, q_1 = i | \lambda \right)}{\sum_{i=1}^{N} \sum_{r=1}^{R} w_r P \left(\mathbf{O}^{(r)}, q_1 = i | \lambda \right)} \end{array} \tag{A.19}$$

To further express the reestimation formula for π_i, considering the equality given by Bayes rule

$$\begin{array}{rcl} P \left(\mathbf{O}^{(r)}, q_1 = i | \lambda \right) & = & P \left(\mathbf{O}^{(r)} | \lambda \right) P \left(q_1 = i | \mathbf{O}^{(r)}, \lambda \right) \\ & = & P \left(\mathbf{O}^{(r)} | \lambda \right) \gamma_1^{(r)}(i) \end{array} \tag{A.20}$$

Given Eq. (A.20) and Eq. (A.19), the reestimation formula can be found as

$$\begin{aligned}\pi_i^{'} &= \frac{\sum_{r=1}^{R} w_r P\big(\mathbf{O}^{(r)}|\lambda\big)\gamma_1^{(r)}(i)}{\sum_{i=1}^{N}\sum_{r=1}^{R} w_r P\big(\mathbf{O}^{(r)}|\lambda\big)\gamma_1^{(r)}(i)} \\ &= \frac{\sum_{r=1}^{R} w_r P\big(\mathbf{O}^{(r)}|\lambda\big)\gamma_1^{(r)}(i)}{\sum_{r=1}^{R} w_r P\big(\mathbf{O}^{(r)}|\lambda\big)\sum_{i=1}^{N}\gamma_1^{(r)}(i)} \\ &= \frac{\sum_{r=1}^{R} w_r P\big(\mathbf{O}^{(r)}|\lambda\big)\gamma_1^{(r)}(i)}{\sum_{r=1}^{R} w_r P\big(\mathbf{O}^{(r)}|\lambda\big)}\end{aligned} \tag{A.21}$$

A.2 Optimization of $Q_{a_{ij}}$

Here the optimization of $Q_{a_{ij}}(\lambda', \lambda)$ is performed by using Lagrange multipliers. The function to optimize is

$$\begin{aligned}Q_{a_{ij}}(\lambda', \lambda) &= \sum_{r=1}^{R} w_r \sum_{\text{all } \mathbf{q}} \left(\sum_{t=2}^{T_r} \log\left(a'_{q_{t-1}q_t}\right)\right) P\left(\mathbf{O}^{(r)}, \mathbf{q}|\lambda\right) \\ &= \sum_{r=1}^{R} w_r \sum_{i=1}^{N}\sum_{j=1}^{N} \left(\sum_{t=2}^{T_r} \log\left(a'_{ij}\right)\right) P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right)\end{aligned} \tag{A.22}$$

since selecting all state sequences (all q) here means all the transitions from i to j and weighting with the corresponding probability at each time t. The stochastic constraint for this function is

$$\sum_{j=1}^{N} a'_{ij} = 1 \tag{A.23}$$

To determine the optimal parameters, Lagrange multipliers are applied. This gives the two partial derivatives

$$\begin{aligned}&\tfrac{\partial}{\partial a'_{ij}}\left[\sum_{r=1}^{R} w_r \sum_{i=1}^{N}\sum_{j=1}^{N}\left(\sum_{t=2}^{T_r}\log\left(a'_{ij}\right)\right) P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right) + \sum_{i=1}^{N}\rho_i\left(1-\sum_{j=1}^{N} a'_{ij}\right)\right] = 0 \Leftrightarrow \\ &\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} \tfrac{1}{a'_{ij}} P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right) - \rho_i = 0\end{aligned} \tag{A.24}$$

and

$$\frac{\partial}{\partial \rho_i}\left[\sum_{r=1}^{R} w_r \sum_{i=1}^{N}\sum_{j=1}^{N}\left(\sum_{t=2}^{T_r}\log\left(a_{ij}^{'}\right)\right) P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right) + \sum_{i=1}^{N}\rho_i\left(1-\sum_{j=1}^{N} a_{ij}^{'}\right)\right] = 0 \Leftrightarrow$$
$$\sum_{j=1}^{N} a_{ij}^{'} = 1 \tag{A.25}$$

where ρ_i is the Lagrange multipliers. From Eq. (A.24) ρ_i can be found as

$$\rho_i = \sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} \frac{1}{a_{ij}^{'}} P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right) \tag{A.26}$$

or from the same equation also $a_{ij}^{'}$ can be found as

$$a_{ij}^{'} = \frac{1}{\rho_i}\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right) \tag{A.27}$$

By inserting Eq. (A.27) into Eq. (A.25), another expression for ρ_i becomes eminent

$$\begin{aligned}&\sum_{j=1}^{N}\frac{1}{\rho_i}\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right) = 1 \Leftrightarrow \\ &\rho_i = \sum_{j=1}^{N}\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right)\end{aligned} \tag{A.28}$$

Now two expressions for ρ_i are found, that is Eqs. (A.26) and (A.28). Since only Eq. (A.26) contains $a_{ij}^{'}$, the equality between them can be used to extract $a_{ij}^{'}$

$$\begin{aligned}&\frac{1}{a_{ij}^{'}}\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right) = \sum_{j=1}^{N}\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right) \Leftrightarrow \\ &a_{ij}^{'} = \frac{\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right)}{\sum_{j=1}^{N}\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right)} \Leftrightarrow \\ &a_{ij}^{'} = \frac{\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right)}{\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r}\sum_{j=1}^{N} P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right)} \Leftrightarrow \\ &a_{ij}^{'} = \frac{\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right)}{\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} P\left(\mathbf{O}^{(r)}, q_{t-1}=i|\lambda\right)}\end{aligned} \tag{A.29}$$

To further express the reestimation formula for $a_{ij}^{'}$, consider the equalities given by

$$\begin{aligned} P\left(\mathbf{O}^{(r)}, q_{t-1}=i, q_t=j|\lambda\right) &= P\left(\mathbf{O}^{(r)}|\lambda\right) P\left(q_{t-1}=i, q_t=j|\mathbf{O}^{(r)},\lambda\right) \\ &= P\left(\mathbf{O}^{(r)}|\lambda\right) \xi_{t-1}(i,j) \end{aligned} \tag{A.30}$$

and

$$\begin{aligned} P\left(\mathbf{O}^{(r)}, q_{t-1}=i|\lambda\right) &= P\left(\mathbf{O}^{(r)}|\lambda\right) P\left(q_{t-1}=i|\mathbf{O}^{(r)},\lambda\right) \\ &= P\left(\mathbf{O}^{(r)}|\lambda\right) \gamma_{t-1}(i) \end{aligned} \tag{A.31}$$

Given these equations and Eq. (A.29), the reestimation formula can be found as

$$\begin{aligned} a_{ij}^{'} &= \frac{\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} P\left(\mathbf{O}^{(r)}|\lambda\right)\xi_{t-1}(i,j)}{\sum_{r=1}^{R} w_r \sum_{t=2}^{T_r} P\left(\mathbf{O}^{(r)}|\lambda\right)\gamma_{t-1}(i)} \\ &= \frac{\sum_{r=1}^{R} w_r P\left(\mathbf{O}^{(r)}|\lambda\right) \sum_{t=1}^{T_r-1} \xi_t(i,j)}{\sum_{r=1}^{R} w_r P\left(\mathbf{O}^{(r)}|\lambda\right) \sum_{t=1}^{T_r-1} \gamma_t(i)} \end{aligned} \tag{A.32}$$

A.3 Optimization of Q_{b_j}

Here the optimization of $Q_{b_j}\left(\lambda', \lambda\right)$ is performed by using the Lagrange multipliers. The function to optimize is

$$\begin{aligned} Q_{b_j}\left(\lambda', \lambda\right) &= \sum_{r=1}^{R} w_r \sum_{\text{all } \mathbf{q}} \left(\sum_{t=1}^{T_r} \log\left(b_{q_t}^{'}(\mathbf{o}_t^{(r)})\right)\right) P\left(\mathbf{O}^{(r)}, \mathbf{q}|\lambda\right) \\ &= \sum_{r=1}^{R} w_r \sum_{j=1}^{N} \left(\sum_{t=1}^{T_r} \log\left(b_{j}^{'}(\mathbf{o}_t^{(r)})\right)\right) P\left(\mathbf{O}^{(r)}, q_t=j|\lambda\right) \end{aligned} \tag{A.33}$$

Selecting all state sequences (all q) in Eq. A.33 is the same as all emissions from state j and weighting with the corresponding probability at each time t. This is the general description given any distribution, however, here the gaussian mixture are considered, that is

$$b_j^{'}(\mathbf{o}_t^{(r)}) = \sum_{k=1}^{M} c_{jk}^{'} \frac{1}{(2\pi)^{D/2}|\mathbf{\Sigma}_{jk}^{'}|^{1/2}} e^{-\frac{1}{2}\left(\mathbf{o}_t^{(r)} - \boldsymbol{\mu}_{jk}^{'}\right)^T \mathbf{\Sigma}_{jk}^{'-1}\left(\mathbf{o}_t^{(r)} - \boldsymbol{\mu}_{jk}^{'}\right)} \tag{A.34}$$

One way to continue is to insert Eq. (A.34) into Eq. (A.33). However, this will be difficult to optimize because of the logarithm of the sum. The problem is that by using a Gaussian mixture we introduce a new missing (or hidden) information, that is the mixture weights. There are then two hidden parameters, *the state sequence* and *the mixture weights*. This implies that Eq. (A.4) can be rewritten for the case of Gaussian mixtures

$$\begin{aligned} Q_{r,b_j}(\lambda',\lambda) &= \sum_{\text{all } \mathbf{q}} \sum_{\text{all } \mathbf{m}} P\left(\mathbf{O}^{(r)},\mathbf{q},\mathbf{m}|\lambda\right) \log P\left(\mathbf{O}^{(r)},\mathbf{q},\mathbf{m}|\lambda'\right) \\ &= \sum_{\text{all } \mathbf{q}} \sum_{\text{all } \mathbf{m}} P\left(\mathbf{O}^{(r)},\mathbf{q},\mathbf{m}|\lambda\right) \sum_{t=1}^{T_r} \log b'_{q_t}(\mathbf{o}_t^{(r)},\mathbf{m}_{q_t t}) \\ &= \sum_{j=1}^{N}\sum_{k=1}^{M}\sum_{t=1}^{T_r} \log c'_{jk} \frac{e^{-\frac{1}{2}\ \mathbf{o}_t^{(r)}-\boldsymbol{\mu}'_{jk}\ ^T\boldsymbol{\Sigma}'^{-1}_{jk}\ \mathbf{o}_t^{(r)}-\boldsymbol{\mu}'_{jk}}}{(2\pi)^{D/2}|\boldsymbol{\Sigma}'_{jk}|^{1/2}} P\left(\mathbf{O}^{(r)},q_t=i,m_{q_t t}=k|\lambda\right) \end{aligned} \tag{A.35}$$

where $\mathbf{m} = \left[m_{q_1 1}, m_{q_2 2}, \ldots, m_{q_t t}, \ldots, m_{q_T T}\right]$ indicates the mixture component for each state at each time. This gives that the new Q_{b_j} for the gaussian mixture, incorporating both the hidden state and mixtures, and it can be found as

$$\begin{aligned} Q_{b_j}(\lambda',\lambda) &= \sum_{r=1}^{R} w_r Q_{r,b_j}(\lambda',\lambda) \\ &= \sum_{r=1}^{R} w_r \sum_{j=1}^{N}\sum_{k=1}^{M}\sum_{t=1}^{T_r} \log c'_{jk} \frac{e^{-\frac{1}{2}\ \mathbf{o}_t^{(r)}-\boldsymbol{\mu}'_{jk}\ ^T\boldsymbol{\Sigma}'^{-1}_{jk}\ \mathbf{o}_t^{(r)}-\boldsymbol{\mu}'_{jk}}}{(2\pi)^{D/2}|\boldsymbol{\Sigma}'_{jk}|^{1/2}} P\left(\mathbf{O}^{(r)},q_t=i,m_{q_t t}=k|\lambda\right) \\ &= \underbrace{\sum_{r=1}^{R} w_r \sum_{j=1}^{N}\sum_{k=1}^{M}\sum_{t=1}^{T_r} \log c'_{jk} P(\bullet)}_{Q_{b_j,c_{jk}}} + \\ &\underbrace{\sum_{r=1}^{R} w_r \sum_{j=1}^{N}\sum_{k=1}^{M}\sum_{t=1}^{T_r} \left(-\frac{D}{2}\log(2\pi) - \frac{1}{2}\log\left(|\boldsymbol{\Sigma}'_{jk}|\right) - \frac{1}{2}\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}'_{jk}\right)^T \boldsymbol{\Sigma}'^{-1}_{jk}\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}'_{jk}\right)\right) P(\bullet)}_{Q_{b_j,\{\boldsymbol{\mu}_{jk},\boldsymbol{\Sigma}_{jk}\}}} \end{aligned} \tag{A.36}$$

where $P(\bullet) = P\left(\mathbf{O}^{(r)},q_t=i,m_{q_t t}=k|\lambda\right)$. This means that $Q_{b_j,c_{jk}}$ and $Q_{b_j,\{\boldsymbol{\mu}_{jk},\boldsymbol{\Sigma}_{jk}\}}$ can be optimized independently.

A.3.1 Optimization of $Q_{b_j,c_{jk}}$

Here the optimization of $Q_{b_j,c_{jk}}(\lambda',\lambda)$ is performed by using Lagrange multipliers. The function to optimize is

$$Q_{b_j,c_{jk}}(\lambda',\lambda) = \sum_{r=1}^{R} w_r \sum_{j=1}^{N}\sum_{k=1}^{M}\sum_{t=1}^{T_r} \log c'_{jk} P(\bullet) \tag{A.37}$$

with the stochastic constraint

$$\sum_{k=1}^{M} c'_{jk} = 1 \tag{A.38}$$

To determine the optimal parameters, Lagrange multipliers are applied. This gives the two partial derivatives

$$\begin{aligned} &\frac{\partial}{\partial c_{jk}^{'}}\left[\sum_{r=1}^{R} w_r \sum_{j=1}^{N}\sum_{k=1}^{M}\sum_{t=1}^{T_r} \log c_{jk}^{'} P(\bullet) + \sum_{j=1}^{N}\rho_j\left(1-\sum_{k=1}^{M} c_{jk}^{'}\right)\right] = 0 \Leftrightarrow \\ &\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} \frac{1}{c_{jk}^{'}} P(\bullet) - \rho_j = 0 \end{aligned} \tag{A.39}$$

and

$$\begin{aligned} &\frac{\partial}{\partial \rho_j}\left[\sum_{r=1}^{R} w_r \sum_{j=1}^{N}\sum_{k=1}^{M}\sum_{t=1}^{T_r} \log c_{jk}^{'} P(\bullet) + \sum_{j=1}^{N}\rho_j\left(1-\sum_{k=1}^{M} c_{jk}^{'}\right)\right] = 0 \Leftrightarrow \\ &\sum_{k=1}^{M} c_{jk}^{'} = 1 \end{aligned} \tag{A.40}$$

where ρ_j is the Lagrange multipliers. From Eq. (A.39) ρ_j can be found as

$$\rho_j = \sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} \frac{1}{c_{jk}^{'}} P(\bullet) \tag{A.41}$$

or from the same equation also $c_{jk}^{'}$ can be found as

$$c_{jk}^{'} = \frac{1}{\rho_j} \sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} P(\bullet) \tag{A.42}$$

By inserting Eq. (A.42) into Eq. (A.40), the expression for ρ_j becomes eminent

$$\begin{aligned} &\sum_{k=1}^{M} \frac{1}{\rho_j} \sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} P(\bullet) = 1 \Leftrightarrow \\ &\rho_j = \sum_{k=1}^{M}\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} P(\bullet) \end{aligned} \tag{A.43}$$

Two expressions for ρ_j are found, Eqs. (A.41) and (A.43). Since only Eq. (A.41) contains $c_{jk}^{'}$, the equality between them can be used to extract $c_{jk}^{'}$

$$\begin{aligned} &\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} \frac{1}{c_{jk}^{'}} P(\bullet) = \sum_{k=1}^{M}\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} P(\bullet) \Leftrightarrow \\ &c_{jk}^{'} = \frac{\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} P(\bullet)}{\sum_{k=1}^{M}\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} P(\bullet)} \end{aligned} \tag{A.44}$$

To further express the reestimation formula for $c_{jk}^{'}$, the equality given by Bayes' rule is considered

$$\begin{aligned} P(\bullet) &= P\left(\mathbf{O}^{(r)}, q_t = i, m_{q_t t} = k|\lambda\right) \\ &= P\left(\mathbf{O}^{(r)}|\lambda\right) P\left(q_t = j, m_{q_t t}|\mathbf{O}^{(r)}, \lambda\right) \\ &= P\left(\mathbf{O}^{(r)}|\lambda\right) \gamma_t(j,k) \end{aligned} \tag{A.45}$$

The considering of Eq. (A.45) gives that Eq. (A.44) can be found as

$$\begin{aligned} c_{jk}^{'} &= \frac{\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} P\left(\mathbf{O}^{(r)}|\lambda\right)\gamma_t(j,k)}{\sum_{k=1}^{M}\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} P\left(\mathbf{O}^{(r)}|\lambda\right)\gamma_t(j,k)} \Leftrightarrow \\ c_{jk}^{'} &= \frac{\sum_{r=1}^{R} w_r P\left(\mathbf{O}^{(r)}|\lambda\right) \sum_{t=1}^{T_r} \gamma_t(j,k)}{\sum_{r=1}^{R} w_r P\left(\mathbf{O}^{(r)}|\lambda\right) \sum_{t=1}^{T_r}\sum_{k=1}^{M} \gamma_t(j,k)} \end{aligned} \tag{A.46}$$

A.3.2 Optimization of $Q_{b_j,\{\boldsymbol{\mu}_{jk},\boldsymbol{\Sigma}_{jk}\}}$

Here the optimization of $Q_{b_j,\{\boldsymbol{\mu}_{jk},\boldsymbol{\Sigma}_{jk}\}}(\lambda', \lambda)$ is performed by finding out when the partial derivatives are zero for the following function

$$\begin{aligned} &Q_{b_j,\{\boldsymbol{\mu}_{jk},\boldsymbol{\Sigma}_{jk}\}} = \\ &\sum_{r=1}^{R} w_r \sum_{j=1}^{N}\sum_{k=1}^{M}\sum_{t=1}^{T_r} \left(-\tfrac{D}{2}\log(2\pi) - \tfrac{1}{2}\log\left(|\boldsymbol{\Sigma}_{jk}^{'}|\right) - \tfrac{1}{2}\left(\mathbf{o}_t^{(r)} - \boldsymbol{\mu}_{jk}^{'}\right)^T \boldsymbol{\Sigma}_{jk}^{'-1}\left(\mathbf{o}_t^{(r)} - \boldsymbol{\mu}_{jk}^{'}\right)\right) P(\bullet) \end{aligned} \tag{A.47}$$

Before the derivations are made, some matrix algebra results are needed, see Tab. A.1.

$\frac{\partial}{\partial \mathbf{x}}\left[\mathbf{x}^T\mathbf{y}\right] = \mathbf{y}$	$\frac{\partial}{\partial \mathbf{x}}\left[\mathbf{x}^T\mathbf{x}\right] = 2\mathbf{x}$
$\frac{\partial}{\partial \mathbf{x}}\left[\mathbf{x}^T\mathbf{A}\mathbf{y}\right] = \mathbf{A}\mathbf{y}$	$\frac{\partial}{\partial \mathbf{x}}\left[\mathbf{y}^T\mathbf{A}\mathbf{x}\right] = \mathbf{A}^T\mathbf{y}$
$\frac{\partial}{\partial \mathbf{x}}\left[\mathbf{x}^T\mathbf{A}\mathbf{x}\right] = \left(\mathbf{A} + \mathbf{A}^T\right)\mathbf{x}$	$\mathbf{A} + \mathbf{A}^T = 2\mathbf{A}$, if $\mathbf{A}$ is symmetric
$\mathrm{tr}\left(k\mathbf{A}\right) = k \cdot \mathrm{tr}\left(\mathbf{A}\right)$	$\mathrm{tr}\left(\mathbf{A} + \mathbf{B}\right) = \mathrm{tr}\left(\mathbf{A}\right) + \mathrm{tr}\left(\mathbf{B}\right)$
$\mathrm{tr}\left(\mathbf{A}\mathbf{B}\right) = \mathrm{tr}\left(\mathbf{B}\mathbf{A}\right)$	$\frac{\partial}{\partial \mathbf{A}}\left[\mathrm{tr}\left(\mathbf{A}\mathbf{B}\right)\right] = \mathbf{B} + \mathbf{B}^T - \mathrm{diag}\left(\mathbf{B}\right)$
$\mathbf{x}^T\mathbf{A}\mathbf{x} = \mathrm{tr}\left(\mathbf{A}\mathbf{x}\mathbf{x}^T\right)$	$\frac{\partial}{\partial \mathbf{A}}\left[\log\left(\lvert\mathbf{A}\rvert\right)\right] = 2\mathbf{A}^{-1} - \mathrm{diag}\left(\mathbf{A}^{-1}\right)$, if $\mathbf{A}$ is symmetric
$\lvert\mathbf{A}^{-1}\rvert = 1/\lvert\mathbf{A}\rvert$	

Table A.1: Matrix algebra results.

First the partial derivative for $\boldsymbol{\mu}_{jk}^{'}$ is found as

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\mu}_{jk}^{'}}\left[Q_{b_j,\{\boldsymbol{\mu}_{jk},\boldsymbol{\Sigma}_{jk}\}}\right] &= \sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\left(\frac{\partial}{\partial \boldsymbol{\mu}_{jk}}\left[-\tfrac{1}{2}\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)^T \boldsymbol{\Sigma}_{jk}^{'-1}\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\right]\right) P(\bullet) \\
&= \sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\left(\frac{\partial}{\partial \boldsymbol{\mu}_{jk}}\left[-\tfrac{1}{2}\left(\left(\mathbf{o}_t^{(r)}\right)^T \boldsymbol{\Sigma}_{jk}^{'-1}\mathbf{o}_t^{(r)} - \left(\mathbf{o}_t^{(r)}\right)^T \boldsymbol{\Sigma}_{jk}^{'-1}\boldsymbol{\mu}_{jk}^{'} - \right.\right.\right. \\
&\quad \left.\left.\left. - \left(\boldsymbol{\mu}_{jk}^{'}\right)^T \boldsymbol{\Sigma}_{jk}^{'-1}\mathbf{o}_t^{(r)} + \left(\boldsymbol{\mu}_{jk}^{'}\right)^T \boldsymbol{\Sigma}_{jk}^{'-1}\boldsymbol{\mu}_{jk}^{'}\right)\right]\right) P(\bullet) \\
&= \sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\left(-\tfrac{1}{2}\left(-\left(\boldsymbol{\Sigma}_{jk}^{'-1}\right)^T \mathbf{o}_t^{(r)} - \boldsymbol{\Sigma}_{jk}^{'-1}\mathbf{o}_t^{(r)} + \right.\right. \\
&\quad \left.\left.\left(\boldsymbol{\Sigma}_{jk}^{'-1} + \left(\boldsymbol{\Sigma}_{jk}^{'-1}\right)^T\right)\boldsymbol{\mu}_{jk}^{'}\right)\right) P(\bullet) \\
&= \sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\left(\boldsymbol{\Sigma}_{jk}^{'-1}\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\right) P(\bullet)
\end{aligned} \tag{A.48}$$

Setting Eq. (A.48) equal to zero and extracting $\boldsymbol{\mu}_{jk}^{'}$ yields the reestimation formula for $\boldsymbol{\mu}_{jk}^{'}$

$$\begin{aligned}
&\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\left(\boldsymbol{\Sigma}_{jk}^{'-1}\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\right) P(\bullet) = 0 \Leftrightarrow \\
&\boldsymbol{\Sigma}_{jk}^{'-1}\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\mathbf{o}_t^{(r)} P(\bullet) - \boldsymbol{\Sigma}_{jk}^{'-1}\boldsymbol{\mu}_{jk}^{'}\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} P(\bullet) = 0 \Leftrightarrow \\
&\boldsymbol{\mu}_{jk}^{'} = \frac{\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\mathbf{o}_t^{(r)} P(\bullet)}{\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} P(\bullet)}
\end{aligned} \tag{A.49}$$

The considering of Eq. (A.45) gives that Eq. (A.49) can be found as

$$\begin{aligned}
\boldsymbol{\mu}_{jk}^{'} &= \frac{\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\mathbf{o}_t^{(r)} P\left(\mathbf{O}^{(r)}|\lambda\right)\gamma_t(j,k)}{\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} P\left(\mathbf{O}^{(r)}|\lambda\right)\gamma_t(j,k)} \Leftrightarrow \\
\boldsymbol{\mu}_{jk}^{'} &= \frac{\sum_{r=1}^{R} w_r P\left(\mathbf{O}^{(r)}|\lambda\right)\sum_{t=1}^{T_r}\gamma_t(j,k)\mathbf{o}_t^{(r)}}{\sum_{r=1}^{R} w_r P\left(\mathbf{O}^{(r)}|\lambda\right)\sum_{t=1}^{T_r}\gamma_t(j,k)}
\end{aligned} \tag{A.50}$$

Now to the second partial derivative, with the aim of finding the reestimation formula for $\boldsymbol{\Sigma}_{jk}^{'}$. The second partial derivative can be found as

$$
\begin{aligned}
&\tfrac{\partial}{\partial \boldsymbol{\Sigma}_{jk}^{'-1}}\left[Q_{b_j,\{\boldsymbol{\mu}_{jk},\boldsymbol{\Sigma}_{jk}\}}\right] \\
&= \sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\left(\tfrac{\partial}{\partial \boldsymbol{\Sigma}_{jk}^{'-1}}\left[-\tfrac{1}{2}\log\left(\left|\boldsymbol{\Sigma}_{jk}^{'}\right|\right)-\tfrac{1}{2}\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)^T\boldsymbol{\Sigma}_{jk}^{'-1}\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\right]\right)P(\bullet) \\
&= \sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\left(\tfrac{\partial}{\partial \boldsymbol{\Sigma}_{jk}^{'-1}}\left[\tfrac{1}{2}\log\left(\left|\boldsymbol{\Sigma}_{jk}^{'-1}\right|\right)-\tfrac{1}{2}\mathrm{tr}\left(\boldsymbol{\Sigma}_{jk}^{'-1}\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)^T\right)\right]\right)P(\bullet) \\
&= \tfrac{1}{2}\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\left(2\boldsymbol{\Sigma}_{jk}-\mathrm{diag}\left(\boldsymbol{\Sigma}_{jk}\right)-2\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)^T+\right. \\
&\quad \left.\mathrm{diag}\left(\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)^T\right)\right)P(\bullet) \\
&= \sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\left(\boldsymbol{\Sigma}_{jk}-\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)^T-\right. \\
&\quad \left.\tfrac{1}{2}\mathrm{diag}\left(\boldsymbol{\Sigma}_{jk}-\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)^T\right)\right)P(\bullet)
\end{aligned}
\tag{A.51}
$$

This equation should be set to zero and the reestimation for $\boldsymbol{\Sigma}_{jk}^{'}$ can be found. Note that in Eq. (A.51) it is implied that if

$$
\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\left(\boldsymbol{\Sigma}_{jk}-\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)^T\right)P(\bullet)=0 \tag{A.52}
$$

it directly follows that

$$
\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\left(\frac{1}{2}\mathrm{diag}\left(\boldsymbol{\Sigma}_{jk}-\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)^T\right)\right)P(\bullet)=0 \tag{A.53}
$$

Given this information the reestimation formula for $\boldsymbol{\Sigma}_{jk}^{'}$ can be found by considering Eq. (A.52)

$$
\begin{aligned}
&\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}\left(\boldsymbol{\Sigma}_{jk}-\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)^T\right)P(\bullet)=0 \Leftrightarrow \\
&\boldsymbol{\Sigma}_{jk}\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}P(\bullet)=\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}P(\bullet)\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)^T \Leftrightarrow \\
&\boldsymbol{\Sigma}_{jk}=\frac{\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}P(\bullet)\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)\left(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\right)^T}{\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r}P(\bullet)}
\end{aligned}
\tag{A.54}
$$

When considering Eq. (A.45), Eq. (A.54) can be found as

$$\begin{aligned}\boldsymbol{\Sigma}_{jk} &= \frac{\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} P\big(\mathbf{O}^{(r)}|\lambda\big)\gamma_t(j,k)\big(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\big)\big(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\big)^T}{\sum_{r=1}^{R} w_r \sum_{t=1}^{T_r} P\big(\mathbf{O}^{(r)}|\lambda\big)\gamma_t(j,k)} \Leftrightarrow \\ \boldsymbol{\Sigma}_{jk} &= \frac{\sum_{r=1}^{R} w_r P\big(\mathbf{O}^{(r)}|\lambda\big) \sum_{t=1}^{T_r} \gamma_t(j,k)\big(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\big)\big(\mathbf{o}_t^{(r)}-\boldsymbol{\mu}_{jk}^{'}\big)^T}{\sum_{r=1}^{R} w_r P\big(\mathbf{O}^{(r)}|\lambda\big) \sum_{t=1}^{T_r} \gamma_t(j,k)}\end{aligned} \tag{A.55}$$

Note that the reestimation formula for $\boldsymbol{\Sigma}_{jk}^{'}$ use the new estimated mean vectors, $\boldsymbol{\mu}_{jk}^{'}$.

A.4 Summary

The derivations of the reestimation formulas for multiple observations using combinatorial weights have now been performed, resulting in the Eqs. (A.21), (A.32), (A.46), (A.50) and (A.55).

Some comments about the combinatorial weights used follow. For independent observations sequences the weights are chosen as

$$w_r = \frac{1}{R} \frac{\prod_{r=1}^{R} P\left(\mathbf{O}^{(r)}|\lambda\right)}{P\left(\mathbf{O}^{(r)}|\lambda\right)} \tag{A.56}$$

By considering Eqs. (A.56) and (A.2), the independence formula can be found as

$$P\left(\mathbf{O}|\lambda\right) = \sum_{r=1}^{R} \frac{1}{R} \frac{\prod_{r=1}^{R} P\left(\mathbf{O}^{(r)}|\lambda\right)}{P\left(\mathbf{O}^{(r)}|\lambda\right)} P\left(\mathbf{O}^{(r)}|\lambda\right) = \prod_{r=1}^{R} P\left(\mathbf{O}^{(r)}|\lambda\right) \tag{A.57}$$

Further, the reestimation formula Eqs. (7.23), (7.24), (7.25), (7.26) and (7.27) can be found by inserting Eq. (A.56) in Eqs. (A.21), (A.32), (A.46), (A.50) and (A.55).

Bibliography

[1] Anil K. Jain, Robert P. W. Duin, and Jianchang Mao, "Statistical pattern recognition: A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4–37, 2000.

[2] L. Rabiner and B.H. Juang, *Fundamentals of Speech Recognition*, Prentice-Hall, 1993, ISBN 0-13-015157-2.

[3] Zimmermann M. Bunke H., "Hidden markov model length optimization for handwriting recognition systems," in *Proceedings of the Eighth International Workshop on Frontiers in Handwriting Recognition*, 2002, pp. 369–374.

[4] Danfeng Li, Biem A., and Subrahmonia J., "HMM topology optimization for handwriting recognition," in *Procedings of ICASSP*, 2001, vol. 3, pp. 1521–1524.

[5] Gunter S. and Bunke H., "A new combination scheme for HMM-based classifiers and its application to handwriting recognition," in *Proceedings of the 16th International Conference on Pattern Recognition*, 2002, vol. 2, pp. 332–337.

[6] Cohen A., "Hidden markov models in biomedical signal processing," in *Proceedings of the 20th Annual International Conference of the IEEE*, 1998, vol. 3, pp. 1145–1150.

[7] K. Karplus, C. Barrett, and R. Hughey, "Hidden Markov models for detecting remote protein homologies," Bioinformatics, 14(10):846–856, 1998.

[8] N. Collier, C. No, and J. Tsujii, "Extracting the names of genes and gene products with a hidden markov model," Proc. COLING 2000, 201–207, 2000.

[9] Kohir V. V. and Desai U.B., "Face recognition using a DCT-HMM approach," in *Proceedings on the Forth Workshop on Applications of Computer Vision, WACV*, October 1998, pp. 226–231.

[10] Frank Wallhoff, Stefan Eickeler, and Gerhard Rigoll, "A Comparison of Discrete and Continuous Output Modeling Techniques for a Pseudo-2D Hidden

Markov Model Face Recognition System," in *IEEE Int. Conference on Image Processing (ICIP)*, Thessaloniki, Greece, 2001.

[11] Nefian A.V. and Hayes III M. H., "Face detection and recognition using hidden Markov models," in *Proceedings on the International Conference on Image Processing, ICIP*, 1998, vol. 1, pp. 141–145.

[12] Baum L.E. and Eagon J. A., "An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to model forecology," *Bulletin of American Mathematical Society*, vol. 73, pp. 360–363, 1967.

[13] Baum L.E., Petrie T., Soules G., and Weiss N., "A maximization technique in the statistical analysis of probabilistic functions of Markov chains," *Annals of mathematical Statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[14] Baum L.E., "An inequality and associated maximisation technique in statistical estimation for probabilistic functions of Markov processes," *Inequalities*, vol. 3, pp. 1–8, 1972.

[15] Pena J.M., Lozano J.A., and Larranaga P., "An empirical comparison of four initialization methods for the k-means algorithm," Pattern Recognition Letters, 20, 1999, 1027-1040. 50, 1999.

[16] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, September 1990.

[17] S. Haykin, *Neural Networks: a comprehensive foundation*, Prentice-Hall, second edition, 1999, ISBN 0-13-273350-1.

[18] J. Burbea and C.R. Rao, "On the Convexity of Some Divergence Measures Based on Entropy Functions," *IEEE Trans. Information Theory*, vol. IT-28, no. 3, pp. 489–495, 1982.

[19] R. Vergin and D. O'Shaughnessy, "On the Use of Some Divergence Measures in Speaker Recognition," in *Proceedings of ICASSP*, 1999, pp. 309–312.

[20] Xiaolin Li, M. Parizeau, and R. Plamondon, "Training hidden markov models with multiple observations - a combinatorial method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 4, pp. 371–377, April 2000.

[21] J. Bilmes, "A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models," Report, University of Berkeley, ICSI-TR-97-021, 1997.

First Order Hidden Markov Model
Theory and Implementation Issues
Mikael Nilsson

ISSN 1103-1581
ISRN BTH-RES--02/05--SE

Printed by Kaserntryckeriet AB, Karlskrona 2005